Εθνικο Μετσοβιο Πολυτεχνειο
Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων
Τομεας Τεχνολογιας Πληροφορικης και Υπολογιστων

Πρόβλεψη εντολών διακλάδωσης με τεχνητά νευρωνικά δίκτυα

Διπλωματικη Εργασια

του

ΑΡΙΣΤΟΤΕΛΗ Κ. ΒΟΝΤΖΑΛΙΔΗ

Επιβλέπων: Διονύσιος Πνευματικάτος
Καθηγητής

Αθήνα, Ιούλιος 2021

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών

Πρόβλεψη εντολών διακλάδωσης με τεχνητά νευρωνικά δίκτυα

Διπλωματικη Εργασια

του

ΑΡΙΣΤΟΤΕΛΗ Κ. ΒΟΝΤΖΑΛΙΔΗ

Επιβλέπων: Διονύσιος Πνευματικάτος
Καθηγητής

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 1η Ιουλίου 2021.

(Υπογραφή)	*(Υπογραφή)*	*(Υπογραφή)*
.....	.....	.....
Διονύσιος Πνευματικάτος	Γεώργιος Γκούμας	Νεκτάριος Κοζύρης
Καθηγητής	Αναπληρωτής Καθηγητής	Καθηγητής

Αθήνα, Ιούλιος 2021

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών

ΔΗΛΩΣΗ ΜΗ ΛΟΓΟΚΛΟΠΗΣ ΚΑΙ ΑΝΑΛΗΨΗΣ ΠΡΟΣΩΠΙΚΗΣ ΕΥΘΥΝΗΣ

Με πλήρη επίγνωση των συνεπειών του νόμου περί πνευματικών δικαιωμάτων, δηλώνω ενυπογράφως ότι είμαι αποκλειστικός συγγραφέας της παρούσας Πτυχιακής Εργασίας, για την ολοκλήρωση της οποίας κάθε βοήθεια είναι πλήρως αναγνωρισμένη και αναφέρεται λεπτομερώς στην εργασία αυτή. Έχω αναφέρει πλήρως και με σαφείς αναφορές, όλες τις πηγές χρήσης δεδομένων, απόψεων, θέσεων και προτάσεων, ιδεών και λεκτικών αναφορών, είτε κατά κυριολεξία είτε βάσει επιστημονικής παράφρασης. Αναλαμβάνω την προσωπική και ατομική ευθύνη ότι σε περίπτωση αποτυχίας στην υλοποίηση των ανωτέρω δηλωθέντων στοιχείων, είμαι υπόλογος έναντι λογοκλοπής, γεγονός που σημαίνει αποτυχία στην Πτυχιακή μου Εργασία και κατά συνέπεια αποτυχία απόκτησης του Τίτλου Σπουδών, πέραν των λοιπών συνεπειών του νόμου περί πνευματικών δικαιωμάτων. Δηλώνω, συνεπώς, ότι αυτή η Πτυχιακή Εργασία προετοιμάστηκε και ολοκληρώθηκε από εμένα προσωπικά και αποκλειστικά και ότι, αναλαμβάνω πλήρως όλες τις συνέπειες του νόμου στην περίπτωση κατά την οποία αποδειχθεί, διαχρονικά, ότι η εργασία αυτή ή τμήμα της δεν μου ανήκει διότι είναι προϊόν λογοκλοπής άλλης πνευματικής ιδιοκτησίας.

(Υπογραφή)

.....
Αριστοτέλης Βοντζαλίδης

1 Ιουλίου 2021

Περίληψη

Η μηχανική μάθηση και συγκεκριμένα τα νευρωνικά δίκτυα έχουν κάνει μεγάλη πρόοδο σε διάφορους τομείς τα τελευταία χρόνια και έχουν επιτρέψει την αξιοποίηση των ηλεκτρονικών υπολογιστών σε πολλά πεδία. Ωστόσο έχουν γίνει λίγα βήματα στην χρήση τους με σκοπό την βελτίωση της αρχιτεκτονικής των υπολογιστών [1] [2] [3] [4]. Οι σύγχρονοι επεξεργαστές αποτελούνται από πολλά ξεχωριστά, πολύπλοκα στοιχεία τα οποία θα μπορούσαν να αντικατασταθούν με νευρωνικά δίκτυα, τα οποία στη συνέχεια είτε να υλοποιηθούν σε επίπεδο υλικού (hardware), είτε να χρησιμοποιηθούν παρατηρήσεις με σκοπό τον σχεδιασμό νέων και αποτελεσματικότερων τμημάτων.

Στόχος της διπλωματικής εργασίας είναι η μελέτη υπαρχόντων, αλλά και η ανάπτυξη νέων, νευρωνικών δικτύων, με σκοπό την χρήση τους ως μονάδων προβλεπτών διακλαδώσεων (branch prediction units). Πρόσφατες εργασίες εντόπισαν κάποιες λίγες εντολές διακλάδωσης οι οποίες προκαλούν συστηματικά λάθος προβλέψεις και είναι ανεξάρτητες των δεδομένων εισόδου μιας εφαρμογής· τις χαρακτήρισαν ως Δύσκολες-να-Προβλεφθούν διακλαδώσεις [5] και έδειξαν ότι η σωστή πρόβλεψη τους προσφέρει σημαντική αύξηση της απόδοσης του επεξεργαστή. Συνεπώς οι συγκεκριμένες διακλαδώσεις αποτελούν ευκαιρία για βελτίωση μέσω τεχνητών νευρωνικών δικτύων, ενώ η μη εξάρτηση από την είσοδο επιτρέπει την προγενέστερη εκπαίδευση τους (offline) και την χρήση τους σε μελλοντικές εκτελέσεις. Το πρώτο τεχνητό νευρωνικό δίκτυο που προτάθηκε για το σκοπό αυτό, χρησιμοποιεί ένα σχετικά βαθύ ιστορικό, το οποίο κωδικοποιείται και στη συνέχεια αναλύεται από ένα συνελικτικό δίκτυο το οποίο εντοπίζει διακλαδώσεις σχετικές με την διακλάδωση προς πρόβλεψη [2]. Πάνω εκεί βασίστηκε και το BranchNet [1], το οποίο χρησιμοποιεί ένα μεγαλύτερο (βαθύτερο) ιστορικό εντολών διακλάδωσης για να κάνει μία πρόβλεψη παρουσιάζοντας δύο βελτιώσεις· μία καλύτερη κωδικοποίηση των δεδομένων και κάποια επιπλέον στρώματα τα οποία ευνοούν την επεξεργασία των περισσότερων δεδομένων. Αν αυτά τα μοντέλα χρησιμοποιηθούν για πρόβλεψη Hard-to-Predict εντολών διακλάδωσης και συνδυαστούν με κάποιον καλό προβλεπτή για τις υπόλοιπες εντολές, το αποτέλεσμα είναι υψηλή επίδοση σε συγκεκριμένες εφαρμογές.

Εμείς μελετάμε αυτά τα τεχνητά νευρωνικά δίκτυα και προτείνουμε ένα δικό μας μοντέλο, το οποίο χρησιμοποιεί δίκτυα μακράς-βραχυπρόθεσμης μνήμης (Long Short-Term Memory) [6]. Όπως στο BranchNet [1]· κωδικοποιούμε το ιστορικό διακλαδώσεων και το επεξεργαζόμαστε με συνελικτικά στρώματα. Ωστόσο, ενσωματώνουμε και ένα αναδρομικό δίκτυο (το LSTM) το οποίο διαβάζει το επεξεργασμένο ιστορικό και μας δίνει την πρόβλεψη. Επιπλέον, ενσωματώνουμε τα τεχνητά νευρωνικά δίκτυα στον προσομοιωτή αρχιτεκτονικής ChampSim [7], ώστε να τα αξιολογήσουμε. Παρατηρούμε ότι τα τεχνητά νευρωνικά δίκτυα αποτελούν ελκυστική επιλογή στον τομέα πρόβλεψης διακλαδώσεων· συγκεκριμένα επιτυγχάνουν αύξηση μεγαλύτερη του 10% στις εντολές ανά κύκλο στην εφαρμογή 541.leela [8]. Μελλοντικά θα μπορούσαμε να υλοποιήσουμε πιο περίπλοκα μοντέλα, ικανά να επιτύχουν καλύτερη πρόβλεψη, ή να σχεδιάσουμε τεχνητά νευρωνικά δίκτυα σε επίπεδο υλικού.

Λέξεις Κλειδιά

Τεχνητά νευρωνικά δίκτυα, μηχανική μάθηση, αρχιτεκτονική υπολογιστών, πρόβλεψη διακλαδώσεων, προσομοιωτής, ChampSim, BranchNet, BPU, Hard-to-Predict, H2P, SPEC2017

Abstract

Machine learning and specifically neural networks have made a major progress in various fields, in recent years. While they made it possible for computers to be used successfully in numerous fields, few steps have been made towards using them for improving and accelerating computer hardware [1] [2] [3] [4]. Therefor, we can examine how neural networks can be used for enhancing computer architecture and subsequently maximizing processor efficiency overall. Designing an advanced computer engineering, is composed of many distinctive parts, each possessing its own complexity and significance. Some of these parts can be replaced with neural networks, which can either be then implemented in hardware level, or we can extract information based on their function, in order to create new and more advanced units.

This diploma thesis aims to examine existing neural networks, as well as developing new ones, to be used as branch prediction units. Recently Stephen Tarsa and others, identified a few branch instructions that systematically produce misspredictions, execute multiple times and are independent from program input; they labeled them as Hard-to-Predict [5] and showed that predicting them correct can offer a significant improvement for the processor. Because these branches have a high execution rate, training and making use of artificial neural networks can be an option, whilst their independence from program input allow offline training and usage of pre-trained models for predictions in future executions. The first artificial neural network for this task was also proposed by Stephen Tarsa and others [2]; they used a deep branch history which they encoded and analyzed using convolutional neural networks, which can identify related branches to the one being predicted. Siavash Zangeneh and others continued their work to develop BranchNet [1], a more complex but also more accurate model. BranchNet uses a deeper (longer) sequence of history branches to make a prediction by introducing 2 improvements; one better way of encoding data and some extra neural layers, both of which allow it to process deeper branch histories. If we use these models for Hard-to-Predict branches and pair them with a state-of-the-art existing branch predictor for the rest of the branches, we can achieve high performance at specific tasks.

We studied these artificial neural networks and we propose a model of our own, using Long Short-Term Memory networks [6]. Inspired from BranchNet, we encode branch history and process it using convolutional layers in a similar way. The key difference is that we include an LSTM network as last layer, in which we feed the processed branch history to make a prediction. Moreover we embed neural networks in a computer architecture simulator, ChampSim [7], so to evaluate them. We see that neural networks can be a powerful approach at predicting branches; they boost performance of 541.leela [8] more than 10%, regarding instructions per cycle - IPC. In the future we can either develop more sophisticated models, capable of achieving higher accuracy, or design artificial neural networks on a hardware level.

Keywords

Artificial neural networks, machine learning, computer architecture, branch prediction, simulator, ChampSim, BranchNet, BPU, Hard-to-Predict, H2P, SPEC2017

στους γονείς μου

Ευχαριστίες

Θα ήθελα καταρχήν να ευχαριστήσω τον καθηγητή κ. Διονύσιο Πνευματικάτο για την επίβλεψη αυτής της διπλωματικής εργασίας και για την ευκαιρία που μου έδωσε να την εκπονήσω στον τομέα Τεχνολογίας Πληροφορικής και Υπολογιστών (CSLAB). Επίσης ευχαριστώ ιδιαίτερα τον Δρ. Βασίλειο Καρακώστα για την καθοδήγησή του και την εξαιρετική συνεργασία που είχαμε. Ιδιαίτερα θα ήθελα να ευχαριστήσω τον συνάδελφο και πολύ καλό μου φίλο Ρολάνδο Ποταμιά καθώς και τον Γιώργο Βαβουλιώτη για την βοήθεια που μου προσέφεραν. Τέλος θα ήθελα να ευχαριστήσω τους γονείς μου για την καθοδήγηση και την ηθική συμπαράσταση που μου προσέφεραν όλα αυτά τα χρόνια.

Αθήνα, Ιούλιος 2021

Αριστοτέλης Βοντζαλίδης

Περιεχόμενα

Περίληψη 1

Abstract 3

Ευχαριστίες 7

Απόδοση ξενόγλωσσων όρων 17

1 Εισαγωγή 19
1.1 Το πρόβλημα της πρόβλεψης διακλαδώσεων . . . 19
1.2 Προσέγγιση με νευρωνικά δίκτυα . . . 20
1.3 Αντικείμενο της διπλωματικής . . . 20
1.4 Διάρθρωση της διπλωματικής . . . 21

2 Θεωρητικό υπόβαθρο και Σχετική βιβλιογραφία 23
2.1 Αρχιτεκτονική κεντρικής μονάδας επεξεργασίας . . . 23
2.1.1 Εντολές μηχανής . . . 23
2.1.2 Ο μετρητής εντολής . . . 24
2.1.3 Κύκλος εντολών . . . 24
2.1.4 Σωλήνωση εντολών . . . 25
2.1.5 Υπερβαθμωτή Οργάνωση Υπολογιστών . . . 26
2.1.5.1 Δυναμική δρομολόγηση εντολών . . . 26
2.1.5.2 Περιορισμοί υπερβαθμωτών αρχιτεκτονικών . . . 27
2.2 Πρόβλεψη διακλαδώσεων . . . 28
2.2.1 Τι είναι η εντολή διακλάδωσης . . . 28
2.2.1.1 Παραδείγματα εντολών άλματος υπό συνθήκη . . . 28
2.2.2 Η ανάγκη για πρόβλεψη εντολών διακλάδωσης . . . 29
2.2.3 Στατικές τεχνικές πρόβλεψης εντολών διακλάδωσης . . . 29
2.2.3.1 Διακλάδωση Not-Taken . . . 29
2.2.3.2 Διακλάδωση Taken . . . 29
2.2.3.3 Εμπρός Not-Taken, Πρός-τα-πίσω Taken . . . 30
2.2.4 Δυναμικές τεχνικές πρόβλεψης εντολών διακλάδωσης . . . 30
2.2.4.1 Πρόβλεψη διακλαδώσεων ενός επιπέδου . . . 30
2.2.4.2 Πρόβλεψη διακλαδώσεων διπλού επιπέδου . . . 31
2.2.4.3 Καθολικό ιστορικό διακλαδώσεων . . . 31
2.2.4.4 Τοπικό ιστορικό διακλαδώσεων . . . 31

2.2.4.5 Υβριδική πρόβλεψη . . . 32
2.2.5 Προηγμένοι προβλεπτές αιχμής . . . 33
2.2.5.1 Μερική αντιστοίχηση μοτίβων - Partial pattern matching . . 33
2.2.5.2 Νευρωνικός προβλεπτής . . . 33
2.2.5.3 Μοντέλα ειδικού σκοπού . . . 33
2.2.5.4 Ο προβλεπτής TAGE-SC-L . . . 34

3 Μηχανική μάθηση & Τεχνητά νευρωνικά δίκτυα 35
3.1 Τύποι αλγορίθμων μηχανικής μάθησης . . . 35
3.1.1 Μάθηση με Επίβλεψη . . . 35
3.1.1.1 Λογιστική παλινδρόμηση (Logistic Regression) . . . 36
3.1.1.2 Κατηγοριοποίηση (Classification) . . . 36
3.1.2 Μάθηση χωρίς Επίβλεψη . . . 36
3.1.3 Ενισχυμένη μάθηση . . . 36
3.2 Δομή τεχνητών νευρωνικών δικτύων . . . 36
3.2.1 Τεχνητοί νευρώνες - Perceptrons . . . 37
3.2.2 Πολυεπίπεδα Perceptron - Multi Layer Perceptron . . . 37
3.2.3 Συνάρτηση ενεργοποίησης - Activation function . . . 38
3.3 Εκπαίδευση νευρωνικών δικτύων . . . 40
3.3.1 Συνάρτηση κόστους - Cost function . . . 41
3.3.2 Αλγόριθμος βελτιστοποίησης . . . 41
3.4 Συνελικτικά νευρωνικά δίκτυα . . . 42
3.5 Επαναλαμβανόμενα νευρωνικά δίκτυα . . . 44
3.5.1 Δίκτυα μακράς βραχυπρόθεσμης μνήμης . . . 44
3.5.2 Μηχανισμός προσοχής . . . 45
3.6 Transformer . . . 45

4 Πρόβλεψη Διαδκλαδώσεων με Τεχνητά Νευρωνικά Δίκτυα 49
4.1 Ο προσομοιωτής ChampSim . . . 49
4.1.1 Δημιουργία των traces . . . 50
4.1.1.1 Είσοδοι εφαρμογής . . . 50
4.1.1.2 Εργαλείο SimPoint . . . 50
4.2 Δύσκολες-να-προβλεφθούν διακλαδώσεις - (Hard-to-Predict Branches) . . . 50
4.2.1 Αναγνώριση και εντοπισμός των Hard-to-Predict branches . . . 51
4.2.2 Υλοποίηση στον προσομοιωτή . . . 51
4.3 Μοντέλα νευρωνικών δικτύων . . . 52
4.3.1 Προσέγγιση προβλήματος . . . 52
4.3.2 Είδη νευρωνικών δικτύων . . . 53
4.3.3 Συνελικτικά δίκτυα . . . 53
4.3.3.1 Tarsa Predictor . . . 53
4.3.3.2 BranchNet Predictor . . . 54
4.3.4 Convolutional - LSTM νευρωνικό δίκτυο . . . 55
4.4 Δημιουργία των δεδομένων (Dataset) . . . 56

4.4.1 Μεθοδολογία συλλογής δεδομένων . . . 56
4.4.2 Υλοποίηση συλλογής δεδομένων (Dataset) . . . 57
4.4.2.1 Δημιουργία H5PY datasets . . . 57
4.4.3 Κωδικοποίηση των δεδομένων - History hashing . . . 57
4.4.3.1 Κωδικοποίηση των δεδομένων σε πίνακα . . . 57
4.5 Εκπαίδευση νευρωνικών δικτύων . . . 59

5 Πειραματική Αξιολόγηση **61**
5.1 Μεθοδολογία ελέγχου ακρίβειας . . . 61
5.1.1 Ενσωμάτωση νευρωνικών στον προσομοιωτή ChampSim . . . 61
5.1.1.1 Ενσωμάτωση κώδικα Python - PyTorch στον ChampSim . . . 62
5.1.1.2 Έλεγχος νευρωνικών εκτός ChampSim και χρήση log files . . . 63
5.2 Αποτελέσματα . . . 63
5.2.1 Mispredictions per Kilo Instructions . . . 64
5.2.2 Instructions Per Cycle - IPC . . . 66
5.3 Συμπεράσματα . . . 68
5.3.1 Σχολιασμός νευρωνικών δικτύων . . . 68
5.3.1.1 Tarsa Predictor . . . 68
5.3.1.2 BranchNet . . . 68
5.3.1.3 Convolutional - LSTM νευρωνικό δίκτυο . . . 68
5.3.2 Σχεδιασμός σε επίπεδο υλικού . . . 68
5.3.3 Ζήτημα βελτίωσης του branch prediction . . . 69

6 Επίλογος **71**
6.1 Σύνοψη . . . 71
6.2 Μελλοντικές Επεκτάσεις . . . 71

Παραρτήματα **73**

Α΄ Παράρτημα πηγαίου κώδικα **75**
Α΄.1 Πηγαίος κώδικας Tarsa . . . 75
Α΄.2 Πηγαίος κώδικας της υλοποίησης μας . . . 75

Βιβλιογραφία **81**

Συντομογραφίες - Αρκτικόλεξα - Ακρωνύμια **83**

Κατάλογος Σχημάτων

2.1 Ο κύκλος εντολών. 25
2.2 Η αρχική σωλήνωση 5 σταδίων μιας RISC αρχιτεκτονικής. 26
2.3 Υπερβαθμωτή σωλήνωση. 27
2.4 Διάγραμμα καταστάσεων ενός μετρητή κορεσμού. 30
2.5 Πρόβλεψη ενός επιπέδου. 31
2.6 Πρόβλεψη με χρήση πίνακα τοπικού ιστορικού. 32

3.1 Διάταξη τεχνικού νευρώνα - Perceptron. 37
3.2 Απλό διάγραμμα πολυεπίπεδου νευρωνικού δικτύου. 38
3.3 Σιγμοειδής συνάρτηση. 39
3.4 Υπερβολική εφαπτόμενη συνάρτηση. 39
3.5 Rectified Linear Unit και Leaky ReLu. 40
3.6 Gradient Descent. 42
3.7 Παράδειγμα ενός συνελικτικού δικτύου. 43
3.8 Επαναλαμβανόμενο νευρωνικό δίκτυο. 44
3.9 Δίκτυο μακράς βραχυπρόθεσμης μνήμης. 45
3.10 Μετάφραση πρότασης. 46
3.11 Το μοντέλο Transformer από τη δημοσίευση Attention is all you need. . . . 47

4.1 Ένας απλός CNN predictor. 54
4.2 Διάγραμα του BranchNet predictor για κάποιο Hard-to-Predict branch. . . . 55
4.3 Διάγραμα του LSTM predictor. 56
4.4 Παράδειγμα κωδικοποίησης ενός ιστορικού σε 1-hot πίνακα. 58

5.1 Απλή διάταξη μιας μονάδας πρόβλεψης που χρησιμοποιεί νευρωνικά δίκτυα για Hard-to-Predict branches. 62
5.2 Σύγκριση MPKI ανά Benchmark. 65
5.3 Σύγκριση IPC ανά Benchmark. 67

Κατάλογος Πινάκων

2.1 Παραδείγματα εντολών άλματος υπό συνθήκη για την αρχιτεκτονική Intel x86. 28

4.1 Hard-to-Predict Branches ανά benchmark. 52
4.2 Παραδείγματα History Hashing. 58

5.1 Παράμετροι σωλήνωσης και κλιμακούμενες (scaled-up) τιμές. 66

Απόδοση ξενόγλωσσων όρων

Απόδοση	Ξενόγλωσσος όρος
εντολή διακλάδωσης	branch instruction
προβλεπτής διακλαδώσεων	branch predictor
τελευταίας τεχνολογίας	state-of-the-art
Δύσκολα-να-Προβλεφθούν	Hard-to-Predict
προσομοιωτής	simulator
Κεντρική Μονάδα Επεξεργασίας (ΚΜΕ)	Central Processing Unit (CPU)
δείκτης εντολής	instruction pointer
μετροπρογράμματα	benchmarks
νευρωνικά δίκτυα	neural networks
μηχανική μάθηση	machine learning
σύνολο δεδομένων	dataset
κωδικοποίηση	hashing
Συνελικτικά Νευρωνικά Δίκτυα	Convolutional Neural Networks
δίκτυα μακράς βραχυπρόθεσμης μνήμης	long-short term memory - LSTM

Κεφάλαιο 1

Εισαγωγή

Ο τομέας των τεχνητών νευρωνικών δικτύων (ή απλά νευρωνικών δικτύων) τα τελευταία χρόνια εξελίσσεται με ραγδαίους ρυθμούς. Τα νευρωνικά δίκτυα δεν αποτελούν μία καινούργια ιδέα· αρχικά προτάθηκαν από τους Warren McCulloch και Walter Pitts[9], οι οποίοι έθεσαν τις πρώτες θεωρητικές βάσεις. Ωστόσο για πολλά χρόνια το πεδίο έμεινε πρακτικά ανεξερεύνητο αφού η απαραίτητη εκπαίδευση και άρα επιτυχία των νευρωνικών δικτύων, έγκειται στην μεγάλη ποσότητα τόσο των δεδομένων, όσο και των υπολογιστικών πόρων.

Τα τελευταία χρόνια η μηχανική μάθηση έχει κάνει άλματα σε πολλούς διαφορετικούς τομείς, όπως στην όραση υπολογιστών, μετάφραση και παραγωγή κειμένου, επαλήθευση δεδομένων, ιατρική, φαρμακευτική κ.α. [10] [11]. Καθημερινά παρουσιάζονται νέοι τρόποι και τομείς αξιοποίησης των νευρωνικών δικτύων αλλά και νέα και διαφορετικά μοντέλα. Εμείς στη συγκεκριμένη διπλωματική προσπαθούμε να χρησιμοποιήσουμε νευρωνικά δίκτυα προκειμένου να βελτιώσουμε την αρχιτεκτονική των υπολογιστών και συγκεκριμένα την μονάδα πρόβλεψης διακλαδώσεων.

1.1 Το πρόβλημα της πρόβλεψης διακλαδώσεων

Η μονάδα πρόβλεψης διακλαδώσεων (branch prediction unit) αποτελεί ένα από τα πιο σημαντικά στοιχεία ενός επεξεργαστή. Παρά το γεγονός ότι οι εντολές διακλαδώσεων αποτελούν συνήθως ένα μικρό κομμάτι του κώδικα προς εκτέλεση, στις σύγχρονες αρχιτεκτονικές αποτελούν καθοριστικό παράγοντα στην ταχύτητα εκτέλεσης εντολών του επεξεργαστή. Οι τεχνικές σωλήνωσης και υπερβαθμωτών αρχιτεκτονικών επηρεάζονται από την κατεύθυνση μιας εντολής διακλάδωσης. Ακόμα η τεχνική εκτέλεσης εντολών εκτός-σειράς εξαρτάται εξ ολοκλήρου από την κατεύθυνση τέτοιων εντολών, καθώς εκεί καθορίζεται ποιες εντολές πρέπει να εκτελεστούν στη συνέχεια.

Η τεχνική πρόβλεψης διακλαδώσεων, ή αλλιώς branch prediction, αποτελεί για τους παραπάνω λόγους μία από τις πιο σημαντικές μονάδες του επεξεργαστή. Η υλοποίηση ενός αξιόπιστου και ακριβή προβλεπτή μπορεί να προσφέρει εξαιρετική αξιοποίηση των διαθέσιμων πόρων και συνεπώς επιτάχυνση της επεξεργαστικής μονάδας συνολικά. Σήμερα οι καλύτεροι διαθέσιμοι branch predictors επιτυγχάνουν ακρίβεια μεγαλύτερη του 99%, ωστόσο υπάρχει ακόμα περιθώριο για βελτίωση.

1.2 Προσέγγιση με νευρωνικά δίκτυα

Παρά την μεγάλη πρόοδο σε αυτόν τον τομέα, υπάρχει ακόμα ένα μεγάλο περιθώριο βελτίωσης. Το συγκεκριμένο ζήτημα αναδείχθηκε από τους Chit-Kwan Lin, Stephen J. Tarsa [5]. Παραδοσιακά, η πρόβλεψη διακλαδώσεων γίνεται κατά την εκτέλεση, με χρήση προβλεπτών που ανακαλύπτουν μοτίβα και προβλέπουν ανάλογα με τα δεδομένα της στιγμής. Ωστόσο μπορεί να υπάρξει και μια διαφορετική προσέγγιση στον τρόπο πρόβλεψης διακλαδώσεων, αυτή με προ-εκπαιδευμένα νευρωνικά δίκτυα. Σε αυτή τη περίπτωση, χρησιμοποιούνται δεδομένα από προηγούμενες εκτελέσεις κάποιας εφαρμογής με άλλες εισόδους, με σκοπό να προ-εκπαιδεύσουμε νευρωνικά δίκτυα.

Πρώτοι με το συγκεκριμένο θέμα ασχολήθηκαν οι Stephen J. Tarsa, Chit-Kwan Lin, Gokce Keskin, Gautham Chinya, Hong Wang [2]. Αρχικά παρατήρησαν ότι υπάρχει ένα μεγάλο περιθώριο βελτίωσης της απόδοσης ενός επεξεργαστή, αν καταφέρουμε να προβλέψουμε σωστά, κάποια λίγες στατικές εντολές διακλάδωσης, τις οποίες θα ονομάσουμε Δύσκολες-να-Προβλεφθούν (Hard-to-Predict) [5]. Το βασικό χαρακτηριστικό αυτών των διακλαδώσεων είναι οι πολλές εκτελέσεις τους σε κάποια συγκεκριμένη εφαρμογή (benchmark), χωρίς όμως να εξαρτώνονται από την είσοδο της εφαρμογής. Η πρόταση τους είναι η εκπαίδευση τεχνητών νευρωνικών δικτύων στην πρόβλεψη τέτοιων εντολών διακλάδωσης πριν την εκτέλεση (offline), με σκοπό την χρήση τους σε επόμενες εκτελέσεις (online) με διαφορετική είσοδο. Η υλοποίηση τους βασίζεται στα συνελικτικά νευρωνικά δίκτυα (convolutional neural networks)· χρησιμοποιούν μια απλή κωδικοποίηση του ιστορικού των εντολών διακλάδωσης, την οποία τροφοδοτούν σε ένα συνελικτικό δίκτυο για να παράγουν μία πρόβλεψη.

Οι Siavash Zangeneh and Stephen Pruett and Sangkug Lym and Yale N. Patt [1], βασίστηκαν στην ιδέα του Tarsa και πρότειναν ένα νέο μοντέλο το οποίο είναι σχεδιασμένο για να προβλέπει Hard-to-Predict εντολές διακλάδωσης. Η βασική ιδέα είναι ίδια· εκπαιδεύουν το μοντέλο offline και το χρησιμοποιούν σε επόμενες εκτελέσεις. Η πρόταση τους σε αντίθεση με τον Tarsa χρησιμοποιεί μία πιο σύνθετη κωδικοποίηση του ιστορικού, καθώς επίσης και μερικά επιπλέον στρώματα εκτός των συνελικτικών· και οι δύο βελτιώσεις επιτρέπουν στο μοντέλο να επεξεργάζεται μεγαλύτερες ακολουθίες ιστορικών και να προσφέρει καλύτερες προβλέψεις.

Το αποτέλεσμα είναι νευρωνικά δίκτυα τα οποία είναι σε θέση να προβλέπουν κάποια λίγα Δύσκολα-να-Προβλεφθούν, Hard-to-Predict branches, με πολύ μεγάλη ακρίβεια σε σχέση με τον state-of-the-art predictor, τον Tage [12], σε κάποιες συγκεκριμένες εφαρμογές που έχουν εκπαιδευτεί.

1.3 Αντικείμενο της διπλωματικής

Στη διπλωματική αυτή στόχος μας είναι να αξιολογήσουμε κατά πόσο τα τεχνητά νευρωνικά δίκτυα μπορούν να χρησιμοποιηθούν στον τομέα της πρόβλεψης εντολών διακλάδωσης. Εμπνευσμένοι από πρόσφατες δουλειές προτείνουμε ένα νευρωνικό δίκτυο το οποίο χρησιμοποιεί συνελικτικά νευρωνικά δίκτυα σε συνδυασμό με δίκτυα μακράς-βραχυπρόθεσμης μνήμης. Η δουλειά μας βασίστηκε στην πρόταση των Siavash Zangeneh, Stephen Pruett, Sangkug Lym Yale, N. Patt[1], όπου χρησιμοποιούν συνελικτικά δίκτυα για εξαγωγή χαρακτηριστικών

από μακρές ακολουθίες ιστορικών διακλαδώσεων.

Αρχικά αναγνωρίζουμε εντολές διακλάδωσης (branches), οι οποίες μπορούν να χαρακτηριστούν ως Δύσκολες-να-Προβλεφθούν (Hard-to-Predict ή H2P) [5]. Στην συνέχεια εκπαιδεύουμε τεχνητά νευρωνικά δίκτυα, στην πρόβλεψη τέτοιων H2P branches χρησιμοποιώντας δεδομένα από πολλές εκτελέσεις, κάποιας εφαρμογής, με διαφορετικές εισόδους. Τα τεχνητά νευρωνικά δίκτυα που εκπαιδεύουμε είναι από προηγούμενες δουλειές που έχουν παρουσιαστεί [2] [1], με χρήση του κώδικα που παρέχεται από το repository του BranchNet [13]. Επιπλέον εκπαιδεύουμε το μοντέλο που προτείνουμε εμείς και παρουσιάζουμε μια σύγκριση μεταξύ των τεχνητών νευρωνικών δικτύων.

Ενσωματώνουμε κάθε τεχνητό νευρωνικό δίκτυο σε έναν προσομοιωτή αρχιτεκτονικής, τον ChampSim [7], ο οποίος έχει χρησιμοποιηθεί στον 3ο διαγωνισμό προανάκτησης δεδομένων [14]. Έπειτα εξάγουμε πληροφορίες σχετικά με την απόδοση του κάθε τεχνητού νευρωνικού δικτύου, σε 4 διαφορετικές εφαρμογές από τη σουίτα μετροπρογραμμάτων Σπες2017. Επιπλέον χρησιμοποιούμε 4 διαφορετικές παραμέτρους συστήματος (configurations) του προσομοιωτή, με σκοπό να αναδείξουμε τη σημασία της πρόβλεψης εντολών διακλάδωσης, σε μελλοντικές αρχιτεκτονικές, οι οποίες θα χρησιμοποιούν περισσότερους πόρους.

Στα αποτελέσματα βλέπουμε ότι τα τεχνητά νευρωνικά δίκτυα και η offline εκπαίδευση τους, είναι μία τεχνική η οποία μπορεί να χρησιμοποιηθεί και να αυξήσει την απόδοση του επεξεργαστή. Ειδικότερα βλέπουμε ότι το BranchNet μπορεί να προσφέρει αύξηση των εντολών ανά κύκλο (Instructions per Cycle ή IPC) ακόμα και 14% σε συγκεκριμένες εφαρμογές. Επίσης παρατηρούμε ότι η υλοποίηση μας με δίκτυα μακράς βραχυπρόθεσμης μνήμης αξίζει να διερευνηθεί περαιτέρω. Τέλος παρατηρήσαμε ότι ο τομέας της πρόβλεψης εντολών διακλάδωσης, παρά την μεγάλη πρόοδο του, έχει ακόμα πολλά να προσφέρει στην συνολική απόδοση ενός επεξεργαστή.

1.4 Διάρθρωση της διπλωματικής

Στην διπλωματική αυτή ασχοληθήκαμε με την ανάλυση και εκπαίδευση νευρωνικών δικτύων, ικανών να προβλέπουν συνθήκες άλματος. Ξεκινάμε στο κεφάλαιο 2 παρουσιάζοντας την αρχιτεκτονική των υπολογιστών και κυρίως πως η σύγχρονη δομή της και σχεδίαση της επωφελείται από αποτελεσματικούς branch predictors. Επιπλέον κάνουμε αναφορά και αναλύουμε τους πρώτους predictors που υλοποιήθηκαν και καταλήγουμε μέχρι τους πιο καινοτόμους και κορυφαίους που έχουν ξεχωρίσει σε διαγωνισμούς, όπως στον διαγωνισμό *Championship Branch Prediction* [15]. Στο κεφάλαιο 3 γίνεται μια θεωρητική ανάλυση των νευρωνικών δικτύων, πως αυτά λειτουργούν και εκπαιδεύονται, καθώς και μια αναφορά σε μερικά μοντέλα. Στη συνέχεια στο κεφάλαιο 4 περνάμε στο πρακτικό κομμάτι. Εκεί θα ασχοληθούμε με την μεθοδολογία που ακολουθήσαμε, ποια νευρωνικά μοντέλα επιλέξαμε και πως τα υλοποιήσαμε, πως συλλέξαμε δεδομένα και πως τελικά χρησιμοποιήσαμε τα δεδομένα αυτά για την εκπαίδευση νευρωνικών δικτύων. Τα αποτελέσματα και η σύγκριση μεταξύ των διαφορετικών νευρωνικών δικτύων παρουσιάζονται και σχολιάζονται στο κεφάλαιο 5. Τέλος, στο κεφάλαιο 6 έχουμε τον επίλογο της διπλωματικής όπου περιλαμβάνεται μια μικρή σύνοψη, μαζί με πιθανές μελλοντικές επεκτάσεις των όσων παρουσιάσαμε.

Κεφάλαιο 2

Θεωρητικό υπόβαθρο και Σχετική βιβλιογραφία

Στο κεφάλαιο αυτό παρουσιάζονται αναλυτικά το ζήτημα της πρόβλεψης διακλαδώσεων (branch prediction), τι είναι η διακλάδωση (branch) και πως η σωστή πρόβλεψη βελτιώνει την απόδοση μιας κεντρικής μονάδας επεξεργασίας (K.M.E.). Αρχικά γίνεται μια προσπάθεια ανάλυσης της αρχιτεκτονικής της KME και στη συνέχεια δίνεται μια πιο λεπτομερής περιγραφή των υπάρχων τεχνικών πρόβλεψης branches.

2.1 Αρχιτεκτονική κεντρικής μονάδας επεξεργασίας

Πριν μπορέσουμε να αναλύσουμε την πρόβλεψη διακλαδώσεων και πως αυτή επηρεάζει την απόδοση του επεξεργαστή, είναι βασικό να αναλύσουμε μερικά πράγματα για την δομή και λειτουργία της κεντρικής μονάδας επεξεργασίας.

2.1.1 Εντολές μηχανής

Όλα τα εκτελέσιμα προγράμματα αποτελούνται από απλά 'βήματα', ή αλλιώς *εντολές*, οι οποίες αποτελούν ορισμένες λειτουργίες μιας κεντρικής μονάδας επεξεργασίας (Central processing unit ή απλά CPU). Όλες οι εντολές που μπορούν να εκτελεστούν ονομάζονται **Σύνολο εντολών** ή αλλιώς **Instruction Set**. Οι εντολές αυτές μπορεί να είναι:

- **Αριθμητικές**, όπως πρόσθεσης ή αφαίρεσης.
  π.χ. `ADD, SUB, MUL`
- **Λογικής**, όπως λογικό ΚΑΙ ή Ή.
  π.χ. `AND, OR, NOT`
- **Δεδομένων**, για αποθήκευση στη μνήμη ή έξοδο σε κάποιο μέσο.
  π.χ. `LD, ST`
- **Ελέγχου ροής**, για αλλαγή της ροής των εντολών, κλήση συνάρτησης κτλ.
  π.χ. `JMP, JE, JZ`

Εμείς θα ασχοληθούμε κατά κύριο λόγο με εντολές διακλαδώσεων, οι οποίες ανήκουν στην κατηγορία εντολών **αλλαγής ροής**.

2.1.2 Ο μετρητής εντολής

Μετρητή εντολής, ή αλλιώς **Program Counter**(PC), ή διαφορετικά δείκτης εντολής, ή αλλιώς **Instruction Pointer**(IP), ορίζουμε τον καταχωρητή ο οποίος ανά πάσα στιγμή περιέχει τη διεύθυνση μνήμης, της επόμενης εντολής για εκτέλεση. Κάθε εντολή ενός προγράμματος είναι αποθηκευμένη σε κάποια συγκεκριμένη θέση μνήμης η οποία είναι μοναδική. Καθώς εκτελούνται εντολές από την ΚΜΕ, ο καταχωρητής PC περιέχει πάντα τη θέση μνήμης της επόμενης εντολής. Οι εντολές φορτώνονται από την μνήμη συνήθως ακολουθιακά, ωστόσο οι εντολές *ελέγχου ροής* αλλάζουν την τιμή του PC προκειμένου να φορτωθούν εντολές από κάποιο διαφορετικό σημείο της μνήμης. Αυτές μπορεί να είναι οι εντολές άλματος υπό συνθήκη, στις οποίες η τιμή του μετρητή εντολής αλλάζει μόνο εάν πραγματοποιείται κάποια συνθήκη.

2.1.3 Κύκλος εντολών

Ως **κύκλο εντολών**, ή αλλιώς **Instruction Cycle** ορίζεται ο *κύκλος* που ακολουθεί η ΚΜΕ από την εκκίνηση της λειτουργίας του υπολογιστή, μέχρι τον τερματισμό του, με σκοπό την εκτέλεση εντολών. Τα κύρια στάδια εκτέλεσης μια εντολής είναι: η παραλαβή εντολής (**fetch stage**), η αποκωδικοποίηση εντολής (**decode stage**) και η εκτέλεση εντολής (**execute stage**). Τα στάδια αυτά αναλύονται σε περισσότερα και απλούστερα βήματα, τα οποία διαφοροποιούνται από αρχιτεκτονική σε αρχιτεκτονική, ωστόσο η βασική λειτουργία των σταδίων αυτών παραμένει παντού ίδια και συνοψίζεται ως εξής:

1. *Στάδιο παραλαβής ή* ***Fetch Stage***. Η επόμενη εντολή για εκτέλεση έρχεται απο τη διεύθυνση μνήμης, που αυτή τη στιγμή δείχνει ο δείκτης εντολών (Program Counter) και αποθηκεύεται στον καταχωρητή εντολής (Instruction Register). Στο τέλος ο δείκτης εντολής αλλάζει τιμή και δείχνει στην επόμενη εντολή που θα διαβαστεί στον επόμενο κύκλο.

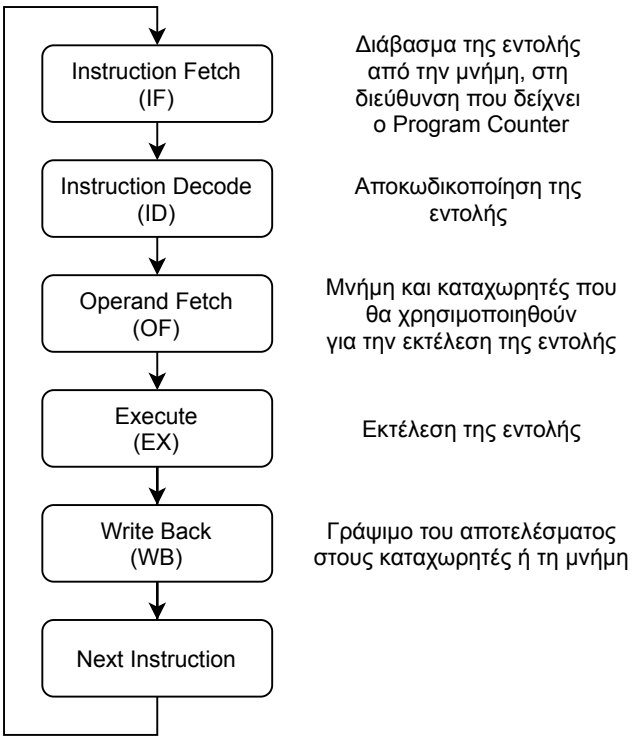
2. *Στάδιο αποκωδικοποίησης ή* ***Decode Stage***. Σε αυτό το στάδιο η κωδικοποιημένη εντολή που βρίσκεται στον *καταχωρητή εντολής* μεταφράζεται από τον αποκωδικοποιητή. Η μονάδα ελέγχου (Control Unit) της ΚΜΕ αποφασίζει ποιες μονάδες και ποιοι καταχωρητές απαιτούνται προκειμένου να εκτελεστεί η εντολή.

3. *Στάδιο εκτέλεσης ή* ***Execute Stage***. Η μονάδα ελέγχου της ΚΜΕ στέλνει, με βάση την αποκωδικοποιημένη πληροφορία, σήματα στα διάφορα τμήματα της να εκτελέσουν λειτουργίες που απαιτούνται από την εντολή. Για παράδειγμα αυτό μπορεί να είναι διάβασμα δεδομένων από καταχωρητές, χρήση της Αριθμητικής-Λογικής μονάδας (Arithmetic-Logic Unit ή ALU) με σκοπό την εκτέλεση πράξεων ή γράψιμο δεδομένων πίσω στους καταχωρητές. Το αποτέλεσμα της διαδικασίας μπορεί να αποθηκευτεί στη μνήμη ή να σταλεί σε κάποια συσκευή εξόδου. Ακόμα, όταν πρόκειται για εντολές άλματος υπό συνθήκη, το αποτέλεσμα που θα προκύψει από την ALU, ίσως αλλάξει την τιμή του Program Counter και συνεπώς οδηγήσει σε διαφορετική εντολή για εκτέλεση.

4. **Επανάληψη κύκλου.**

Στο σχήμα 2.1 φαίνονται τα ξεχωριστά στάδια ενός κύκλου εντολών.

Στις απλές αρχιτεκτονικές, ο κύκλος εντολής γίνεται σειριακά. Σε αυτή την απλή περίπτωση, κάθε εντολή περνάει από όλα τα στάδια του κύκλου εντολών πριν ξεκινήσει η επεξεργασία της επόμενης εντολής. Έτσι σε κάθε κύκλο ρολογιού εξετάζεται μόνο μία εντολή, με αποτέλεσμα όταν για παράδειγμα η εντολή βρίσκεται στο στάδιο Εκτέλεσης, τα τμήματα της ΚΜΕ που χρησιμοποιούνται για τα στάδια Παραλαβής και Αποκωδικοποίησης να μην αξιοποιούνται. Για το λόγο αυτό σε πιο σύγχρονες και πιο περίπλοκες αρχιτεκτονικές ο κύκλος εντολών 'σπάει' σε διαφορετικά κομμάτια. Συνεπώς σε κάθε κύκλο ρολογιού μπορούν να χρησιμοποιούνται τα τμήματα της ΚΜΕ για διαφορετικές εντολές. Η διαδικασία αυτή ονομάζεται **σωλήνωση** ή **pipelining** και θα αναλυθεί στη συνέχεια.

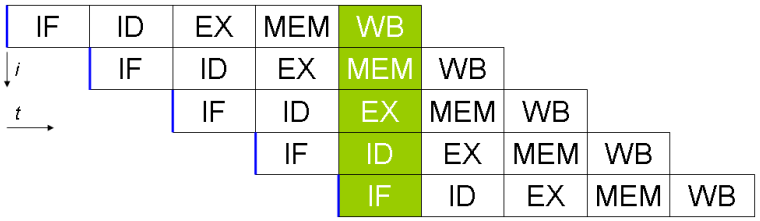


Σχήμα 2.1: *Ο κύκλος εντολών.*

2.1.4 Σωλήνωση εντολών

Σωλήνωση εντολών (instruction pipelining) ονομάζεται η τεχνική με την οποία μπορούμε να παραλληλοποιήσουμε την εκτέλεση εντολών σε έναν μονοπύρηνο επεξεργαστή. Με την σωλήνωση προσπαθούμε να 'σπάσουμε' σε όσο το δυνατόν περισσότερα και απλούστερα βήματα την εκτέλεση μιας εντολής, με σκοπό να αξιοποιούμε ταυτόχρονα όσο τον δυνατόν περισσότερα διαφορετικά τμήματα μιας ΚΜΕ. Για παράδειγμα ο κύκλος εντολών που περιγράφηκε στο 2.1.3 μπορεί να χωριστεί σε τρία βήματα και σε μια αρχιτεκτονική που χρησιμοποιεί σωλήνωση, να εκτελούνται κάθε φορά ταυτόχρονα πολλές εντολές, όπου η κάθε μια βρίσκεται και σε διαφορετικό στάδιο. Με αυτό τον τρόπο επιτυγχάνεται επιτάχυνση, αφού παράγονται περισ-

σότερες εντολές, σε λιγότερο χρόνο. Στο σχήμα 2.2 φαίνεται η κλασσική RISC[1] σωλήνωση που χρησιμοποιήθηκε στις πρώτες αρχιτεκτονικές όπως η αρχιτεκτονική MIPS ή SPARC.



Σχήμα 2.2: *Η αρχική σωλήνωση 5 σταδίων μιας RISC αρχιτεκτονικής. (IF = Instruction Fetch, ID = Instruction Decode, EX = Execute, MEM = Memory access, WB = Register Write Back). Ο κάθετος άξονας είναι οι διαδοχικές εντολές, ο οριζόντιος άξονας είναι ο χρόνος. Στη πράσινη στήλη βλέπουμε ότι η πρώτη εντολή βρίσκεται στο στάδιο WB, ενώ η πιο πρόσφατη εντολή έρχεται αυτή τη στιγμή από τη μνήμη.*

Η σωλήνωση RISC χωρίζει τον κύκλο εντολών σε πέντε διαφορετικά στάδια, ωστόσο σύγχρονες αρχιτεκτονικές μπορεί να έχουν σωληνώσεις πολύ περισσότερων σταδίων. Για παράδειγμα η Intel στις πιο πρόσφατες αρχιτεκτονικές, χρησιμοποιεί απο 14 έως και 20 στάδια σωλήνωσης [16].

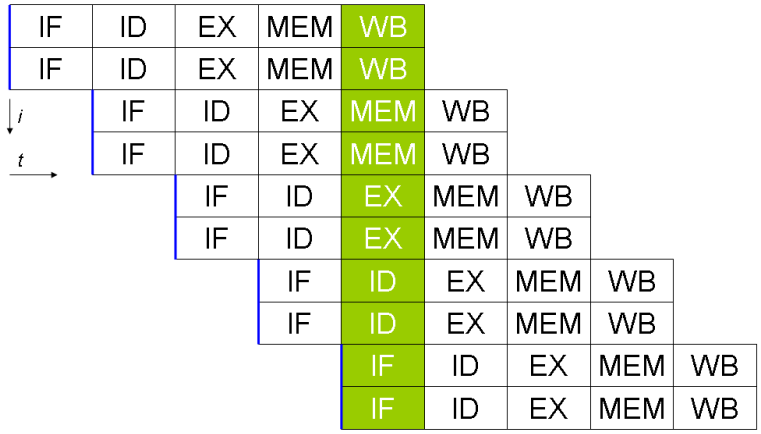
2.1.5 Υπερβαθμωτή Οργάνωση Υπολογιστών

Μια υπερβαθμωτή αρχιτεκτονική (superscalar architecture) διαφέρει με μια απλή βαθμωτή, υπό την έννοια ότι σε κάθε κύκλο ρολογιού, προσπαθούν να εκτελεστούν παράλληλα, παραπάνω από μία εντολή [17]. Σε μια τέτοια αρχιτεκτονική σε κάθε κύκλο ρολογιού, μπορεί να διαβάζονται ταυτόχρονα 2, 4, 8 ή ακόμα και παραπάνω εντολές από την μνήμη, για να εκτελεστούν παράλληλα. Αντίστοιχα με αυτό τον τρόπο αυξάνεται η διεκπεραιωτική ικανότητα (throughput) του επεξεργαστή, οι εντολές που ολοκληρώνονται δηλαδή σε έναν κύκλο ρολογιού. Οι υπερβαθμωτές αρχιτεκτονικές διαφέρουν από πολυπύρηνους επεξεργαστές, αφού η παραλληλοποιήση δεν επιτυγχάνεται χρησιμοποιώντας πολλούς επεξεργαστές, αλλά πολλαπλά τμήματα ενός επεξεργαστή, ταυτόχρονα. Στο σχήμα 2.3 φαίνεται ένα στιγμιότυπο εκτέλεσης μιας υπερβαθμωτής σωλήνωσης, όπου χρησιμοποιούνται τα πέντε κλασσικά στάδια σωλήνωσης μιας RISC αρχιτεκτονικής.

2.1.5.1 Δυναμική δρομολόγηση εντολών

Όπως περιγράψαμε, σε μία υπερβαθμωτή αρχιτεκτονική, καθίσταται δυνατή η εκτέλεση πολλών εντολών ταυτόχρονα. Ωστόσο πολλές εντολές δεν απαιτούν τον ίδιο χρόνο για να ολοκληρωθούν. Για παράδειγμα μια εντολή πρόσθεσης διαρκεί μερικούς κύκλους ρολογιού, ενώ μια εντολή φόρτωσης δεδομένων από τη μνήμη μπορεί να απαιτεί ακόμα και εκατοντάδες κύκλου ρολογιού, εάν τα απαιτούμενα δεδομένα βρίσκονται σε κάποιο σκληρό δίσκο. Για το λόγο αυτό σε σύγχρονες αρχιτεκτονικές ο επεξεργαστής εκτελεί εντολές εκτός σειράς, δυναμικά, ανάλογα με τα διαθέσιμα δεδομένα που υπάρχουν κάθε στιγμή. Η σειρά εκτέλεσης σε αυτή τη περίπτωση μπορεί να διαφέρει από την σειρά εντολών που έχει αρχικά οριστεί από τον προγραμματιστή. Με τον τρόπο αυτό μειώνεται ο χρόνος ο οποίος ο επεξεργαστής μένει

[1] Reduced Instruction Set Computer



Σχήμα 2.3: *Υπερβαθμωτή σωλήνωση.*
(IF = Instruction Fetch, ID = Instruction Decode, EX = Execute, MEM = Memory access, WB = Register Write Back).

ανενεργός (π.χ. ενώ περιμένει για δεδομένα από τη μνήμη) και η παραγωγή εντολών, στον ίδιο χρόνο, αυξάνεται.

Η δυναμική δρομολόγηση σε συνδυασμό με την σωλήνωση μπορούν να προσφέρουν πολύ μεγάλη αύξηση στη διεκπεραιωτική ικανότητα (throughput) του επεξεργαστή, ωστόσο υπάρχουν κάποιοι περιορισμοί στην εφαρμογή της συγκεκριμένης τεχνικής.

2.1.5.2 Περιορισμοί υπερβαθμωτών αρχιτεκτονικών

Όπως περιγράψαμε παραπάνω, στην υπερβαθμωτή σωλήνωση, εκτελούνται ταυτόχρονα πολλές εντολές, ενώ παράλληλα με χρήση δυναμικής δρομολόγησης ένας επεξεργαστής μπορεί να εκτελέσει παραπάνω εντολές σε λίγοτερο χρόνο. Ωστόσο αυτό δεν είναι πάντα εφικτό· η υπερβαθμωτή σωλήνωση έχει κάποιους περιορισμούς.

1. **Εξάρτηση δεδομένων.** Προκειμένου να καθίσταται εφικτή η εκτός-σειράς εκτέλεση εντολών πρέπει οι εντολές προς εκτέλεση να μην έχουν εξαρτήσεις δεδομένων. Για παράδειγμα εάν έχουμε μια εντολή φόρτωσης δεδομένων από τη μνήμη, τα οποία δεδομένα δεν χρησιμοποιούνται στις επόμενες εντολές τότε η εκτός-σειράς εκτέλεση είναι εφικτή. Διαφορετικά αν η φόρτωση δεδομένων γίνεται με σκοπό να χρησιμοποιηθούν τα δεδομένα σε επόμενες αριθμητικές εντολές (π.χ. πρόσθεσης), τότε ο επεξεργαστής είναι αναγκασμένος να σταματήσει την εκτέλεση εντολών, μέχρις ότου τα δεδομένα να είναι διαθέσιμα.

2. **Πολυπλοκότητα συστήματος.** Προκειμένου ο επεξεργαστής να ελέγξει για εξάρτηση δεδομένων, είναι απαραίτητο να μπορεί να κάνει ελέγχους, ώστε να γνωρίζει ποιες είναι οι εξαρτήσεις μεταξύ των εντολών. Όσο αυξάνουμε το μέγεθος της σωλήνωσης και συνεπώς την ποσότητα των εντολών που εκτελούνται παράλληλα, η πολυπλοκότητα της ΚΜΕ αυξάνεται σε μεγάλο βαθμό.

3. **Διαδικασία εντολών άλματος.** Όταν κατά την εκτέλεση εντολών υπάρχει εντολή διακλάδωσης, υπάρχει περίπτωση η ροή των εντολών να αλλάζει, ανάλογα με την συνθήκη που εξετάζεται. Ο επεξεργαστής πρέπει να μπορεί να προβλέψει την τιμή

της συνθήκης προκειμένου να μπορεί να αποφασίσει ποιες εντολές θα εκτελέσει στη συνέχεια, διαφορετικά υπάρχει καθυστέρηση και άρα μη αξιοποίηση της σωλήνωσης.

Οι υπερβαθμωτές αρχιτεκτονικές μπορούν να παράγουν πολλές εντολές σε μόλις έναν κύκλο ρολογιού. Ωστόσο όπως αναφέραμε παραπάνω στο 2.1.5.2, περίπτωση 3, η πρόβλεψη άλματος συνθηκών αποτελεί έναν βασικό περιορισμό. Ειδικά όταν αυξάνουμε το πλάτος της σωλήνωσης, την ποσότητα δηλαδή των παράλληλων εκτελούμενων εντολών, η σωστή πρόβλεψη αλμάτων υπό συνθήκη αποτελεί σημαντικό κομμάτι προκειμένου να υπάρχει καλή αξιοποίηση της σωλήνωσης. Η μονάδα πρόβλεψης διακλαδώσεων (**Branch Predictor**), αποτελεί πολύ σημαντικό τμήμα των σύγχρονων αρχιτεκτονικών. Παρακάτω θα γίνει μια προσπάθεια ανάλυσης τέτοιων μονάδων.

2.2 Πρόβλεψη διακλαδώσεων

2.2.1 Τι είναι η εντολή διακλάδωσης

Η εντολή διακλάδωσης ή άλματος υπό συνθήκη (ή branch instruction ή πιο απλά branch), είναι εντολές σε ένα εκτελέσιμο οι οποίες μπορεί να αλλάξουν τη σειρά εκτέλεσης των εντολών του επεξεργαστή, βασιζόμενες σε μία συνθήκη. Εάν η συνθήκη που εξετάζεται είναι 'Αληθής' τότε το άλμα χαρακτηρίζεται ως 'Taken' και η εκτέλεση του προγράμματος συνεχίζεται σε διαφορετικό σημείο στη μνήμη, όπου οι εντολές του δεύτερου κλάδου βρίσκονται αποθηκευμένες. Διαφορετικά εάν η συνθήκη είναι 'Ψευδής' τότε το άλμα χαρακτηρίζεται ως 'Not-Taken' και η ροή του προγράμματος συνεχίζεται με την επόμενη εντολή στη μνήμη. Οι εντολές αυτές υπάρχουν σε μεγάλο βαθμό μέσα σε όλα τα εκτελέσιμα και επηρεάζουν σε μεγάλο βαθμό την απόδοση του επεξεργαστή όπως αναλύθηκε στο 2.1.5.2.

2.2.1.1 Παραδείγματα εντολών άλματος υπό συνθήκη

Γενικότερα όλη η λίστα με το σύνολο τον εντολών που μπορούν να εκτελεστούν από έναν επεξεργαστή, παρέχονται από τον κατασκευαστή, ωστόσο γενικά οι εντολές αυτές είναι κοινές. Στόν πίνακα 2.1 φαίνονται μερικά παραδείγματα για την αρχιτεκτονική Intel x86.

Εντολή	**Περιγραφή εντολής**
JE Addr	Άλμα στη διεύθυνση Addr αν υπάρχει ισότητα (καταχωρητής ZF = 1).
JNE Addr	Άλμα στη διεύθυνση Addr αν δεν υπάρχει ισότητα (καταχωρητής ZF = 0).
JCXZ Addr	Άλμα στη διεύθυνση Addr αν ο καταχωρητής CX = 0.
JO Addr	Άλμα στη διεύθυνση Addr αν υπήρξε υπερχείλιση (καταχωρητές OF = 1).

Πίνακας 2.1: *Παραδείγματα εντολών άλματος υπό συνθήκη για την αρχιτεκτονική Intel x86.*

2.2.2 Η ανάγκη για πρόβλεψη εντολών διακλάδωσης

Όπως περιγράψαμε στο 2.1.5 οι υπερβαθμωτές αρχιτεκτονικές επωφελούνται σε μεγάλο βαθμό εκτελώντας εντολές εκτός σειράς (Out-of-Order execution). Κάθε φορά που συναντάται μια εντολή διακλάδωσης η ΚΜΕ πρέπει να αποφασίσει με ποιες εντολές θα τροφοδοτήσει την σωλήνωση. Ωστόσο για να γνωρίζουμε ποια είναι η ροή του προγράμματος και άρα ποιες είναι οι επόμενες εντολές προς εκτέλεση, πρέπει να εκτελεστεί πλήρως η εντολή διακλάδωσης. Η μονάδα πρόβλεψης διακλάδωσης (ή αλλιώς **Branch Predictor**) προσπαθεί να λύσει το πρόβλημα αυτό, δίνοντας μια πρόβλεψη για το αποτέλεσμα της εντολής. Ο επεξεργαστής 'φορτώνει' τη σωλήνωση με εντολές που προκύπτουν από τη ροή εκτέλεσης της πρόβλεψης χωρίς να υπάρχει ανάγκη για καθυστέρηση. Εάν η πρόβλεψη αποδειχθεί λάθος σε επόμενους κύκλους, τότε ο επεξεργαστής χρειάζεται να αναιρέσει ό,τι εντολές εκτέλεσε και να φέρει από τη μνήμη τις σωστές εντολές προς εκτέλεση [18].

Οι τεχνικές πρόβλεψης διακλαδώσεων χωρίζονται σε δύο κατηγορίες, τις **στατικές** και τις **δυναμικές**.

2.2.3 Στατικές τεχνικές πρόβλεψης εντολών διακλάδωσης

Οι στατική πρόβλεψη είναι η πιο απλή τεχνική πρόβλεψης διακλαδώσεων καθώς δεν βασίζεται σε καμία πληροφορία από το δυναμικό ιστορικό που δημιουργείται κατά την εκτέλεση εντολών. Αντιθέτως η απόφαση για μια πρόβλεψη γίνεται βασίζοντας σε κανόνες που έχουν αποφασισθεί από πριν, μη έχοντας καμία γνώση για τον κώδικα που εκτελείται. Παρακάτω παρουσιάζονται μερικές τεχνικές στατικής πρόβλεψης

2.2.3.1 Διακλάδωση Not-Taken

Στη συγκεκριμένη τεχνική θεωρούμε πως πάντα μια διακλάδωση θα έχει ψευδή συνθήκη και άρα η ροή του προγράμματος θα συνεχίσει με την επόμενη εντολή στη μνήμη. Οι πρώτες υλοποιήσεις των αρχιτεκτονικών MIPS [19] και SPARC [20] χρησιμοποιούσαν αυτή τη τεχνική. Με την μέθοδο αυτή προβλέπεται ότι πάντα μια διακλάδωση θα χαρακτηριστεί ως Not-Taken και άρα παραλαμβάνονται για εκτέλεση οι αμέσως επόμενες εντολές στη μνήμη. Η συγκεκριμένη τεχνική έχει πολύ απλή υλοποίηση ωστόσο έχει και πολύ χαμηλά ποσοστά επιτυχίας.

2.2.3.2 Διακλάδωση Taken

Με την συγκεκριμένη τεχνική προβλέπουμε πάντα πως μια διακλάδωση θα χαρακτηριστεί ως Taken. Έτσι, κάθε φορά που εκτελείται ένα άλμα υπό συνθήκη, η συνθήκη θεωρείται αληθής και οι εντολές φορτώνονται από την 'μακρινή' διεύθυνση μνήμης που δείχνει η εντολή διακλάδωσης. Η μέθοδος αυτή παρουσιάζει πολύ καλύτερη απόδοση σε σχέση με την 2.2.3.1, κυρίως λόγο των εντολών διακλάδωσης που αφορούν βρόγχους επανάληψης (loops).

2.2.3.3 Εμπρός Not-Taken, Πρός-τα-πίσω Taken

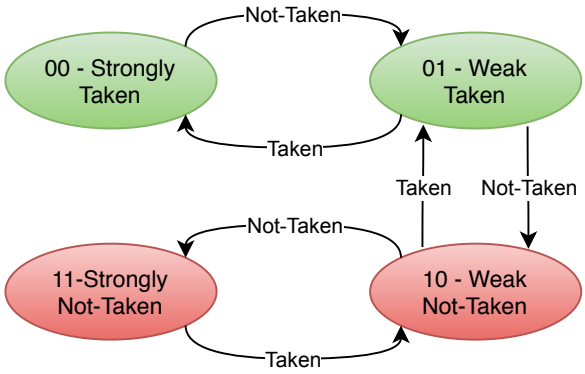
Στην τεχνική αυτή χρησιμοποιείται ένας συνδυασμός των δύο προηγούμενων. Κάθε φορά όπου συναντάμε μία εντολή διακλάδωσης που δείχνει σε θέση μνήμης μικρότερη της τωρινής (προς-τα-πίσω), προβλέπουμε ως Taken, διαφορετικά εάν δείχνει σε θέση μνήμης μεγαλύτερη από την τωρινή (εμπρός), τότε προβλέπουμε Not-Taken. Η μέθοδος αυτή βασίζεται στην ιδέα ότι εντολές διακλάδωσης που δείχνουν σε θέση μνήμης 'προς-τα-πίσω', πιθανότατα να είναι εντολές που υλοποιούν βρόγχους επανάληψης και συνεπώς χαρακτηρίζονται συχνότερα ως Taken παρά ως Not-Taken.

2.2.4 Δυναμικές τεχνικές πρόβλεψης εντολών διακλάδωσης

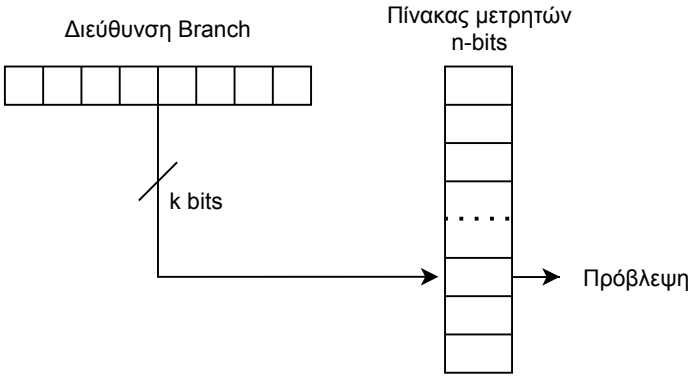
Η δυναμική τεχνική πρόβλεψης χρησιμοποιεί πληροφορίες από τα αποτελέσματα προηγούμενων διακλαδώσεων (αν ήταν Taken ή Not-Taken) τα οποία συλλέγονται κατά τον χρόνο εκτέλεσης. Υπάρχουν πολλοί τρόποι δυναμικής πρόβλεψης, εκ των οποίων μερικοί παρουσιάζονται στη συνέχεια [21].

2.2.4.1 Πρόβλεψη διακλαδώσεων ενός επιπέδου

Η πρόβλεψη ενός επιπέδου (ή αλλιώς **One-level branch prediction**) πρόκειται για την πιο απλή τεχνική δυναμικής πρόβλεψης. Για κάθε εντολής διακλάδωσης αντιστοιχούμε έναν μετρητή 'κορεσμού' (σχήμα 2.4). Ο μετρητής μπορεί να είναι του ενός, δύο ή τριών bit και κάθε φορά που η διακλάδωση υπολογίζεται ως Taken, ή Not-Taken, τότε ο μετρητής αυξάνεται, ή μειώνεται, κατά ένα, αντίστοιχα. Κάθε φορά που θέλουμε να κάνουμε μία πρόβλεψη για κάποια εντολή διακλάδωσης κοιτάμε την τιμή του αντίστοιχου μετρητή. Εάν το περισσότερο σημαντικό bit είναι ίσο με ένα (ή αλλιώς εάν η τιμή είναι μεγαλύτερη από τη μεσαία τιμή των διαθέσιμων τιμών του μετρητή), η πρόβλεψη είναι Taken, διαφορετικά Not-Taken. Η αντιστοίχηση μεταξύ εντολών διακλάδωσης και μετρητών στην πιο απλή περίπτωση μπορεί να γίνει χρησιμοποιώντας τα k ελάχιστα σημαντικά ψηφία του Program Counter (βλ. 2.1.2) της εντολής διακλάδωσης που θέλουμε να προβλέψουμε, όπως φαίνεται στο σχήμα 2.5.



Σχήμα 2.4: *Διάγραμμα καταστάσεων ενός μετρητή κορεσμού.*



Σχήμα 2.5: *Πρόβλεψη ενός επιπέδου.*

2.2.4.2 Πρόβλεψη διακλαδώσεων διπλού επιπέδου

Στην πρόβλεψη διακλαδώσεων διπλού επιπέδου, ή αλλιώς στον προβλεπτή συσχετισμού (**Correlation-Based Branch predictor**) [22, 23, 24], έχουμε όπως και πριν έναν πίνακα με πολλούς διαφορετικούς μετρητές κορεσμού, τον πίνακα 'Μοτίβων ιστουρικού' (Pattern History Table). Αυτό που αλλάζει ωστόσο είναι η αντιστοίχηση των εντολών διακλάδωσης με τους μετρητές. Αντί να χρησιμοποιηθούν μόνο τα λιγότερα σημαντικά bit της εντολής, χρησιμοποιούνται επιπλέον bit από έναν καταχωρητή που θα ονομάσουμε 'Καταχωρητή ιστορικού' (Branch History Register). Ο καταχωρητής αυτός περιέχει το αποτέλεσμα των τελευταίων διακλαδώσεων. Κάθε φορά που μια διακλάδωση υπολογίζεται ως Taken, γίνεται ολίσθηση στον καταχωρητή ενός άσσου, διαφορετικά ενός μηδενικού. Έτσι για έναν καταχωρητή n bits, χρησιμοποιούμε $n+k$ ψηφία (όπου k τα ελάχιστα σημαντικά ψηφία της διεύθυνσης εντολής), για να αντιστοιχήσουμε μια εντολή με τον αντίστοιχο μετρητή και να κάνουμε τελικά μία πρόβλεψη.

2.2.4.3 Καθολικό ιστορικό διακλαδώσεων

Όπως αναφέραμε παραπάνω στο 2.2.4.2 η πρόβλεψη διπλού επιπέδου μπορεί να γίνει με χρήση ενός καταχωρητή, στον οποίο αποθηκεύουμε κάθε αποτέλεσμα μιας διακλάδωσης. Στην περίπτωση που χρησιμοποιούμε έναν καταχωρητή για όλες τις διακλαδώσεις, ο καταχωρητής αυτός ονομάζεται καταχωρητής καθολικού ιστορικού (**Global History Register**) [25] [26]. Το βασικό πλεονέκτημα της συγκεκριμένης τεχνικής είναι ότι ο συνδυασμός αυτός μπορεί να μάθει γρήγορα και αποτελεσματικά να προβλέπει πολλά διαφορετικά μοτίβα ιστορικών. Πολλές φορές φαίνεται το αποτέλεσμα κάποιων διακλαδώσεων να επηρεάζουν την κατεύθυνση άλλων. Με τη χρήση του καθολικού καταχωηρητή ιστορικού, μπορούμε να εντοπίσουμε τέτοιες συσχετίσεις και να κάνουμε εύστοχες προβλέψεις. Παρακάτω στον αλγόριθμο 2.1 παρουσιάζεται ένα παράδειγμα τέτοιας συσχέτισης.

2.2.4.4 Τοπικό ιστορικό διακλαδώσεων

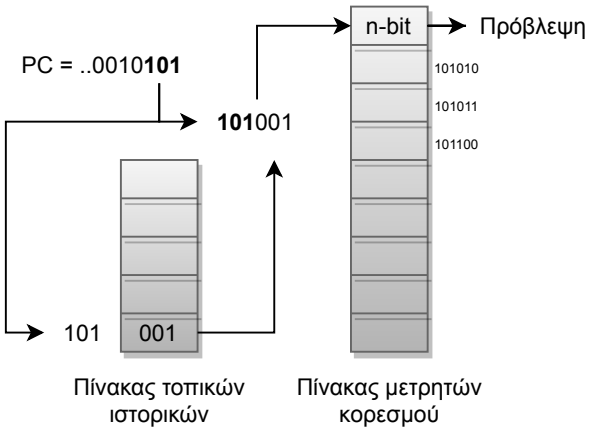
Εναλλακτικά, μπορούμε να χρησιμοποιήσουμε έναν καταχωρητή ιστορικού, ξεχωριστό για κάθε εντολή διακλάδωσης, σε κάποιον πίνακα, στον οποίο η αντιστοίχηση γίνεται χρησιμοποιώντας τα λιγότερο σημαντικά ψηφία της διεύθυνσης εντολής [25] [26]. Η τεχνική αυτή χρη-

Αλγοριθμος 2.1: *Το αποτέλεσμα της δεύτερης διακλάδωσης καθορίζεται αποκλειστικά από το αποτέλεσμα της πρώτης.*

```
1    ...
2    x=1;
3    // Branch A
4    if (a==0)
5        x=0;
6    // Branch B
7    if(x==0){
8        ...
9    }
10
```

σιμοποιεί το ιστορικό μιας διακλάδωσης, μαζί με τον αριθμό εντολής της ίδιας, προκειμένου να γίνει η αντιστοίχηση σε κάποιον μετρητή κορεσμού. Κάθε φορά που έχουμε το αποτέλεσμα κάποιας διακλάδωσης, ενημερώνεται τόσο ο μετρητής που χρησιμοποιήθηκε, όσο και ο αντίστοιχος καταχωρητής ιστορικού. Επειδή ο πίνακας με τους καταχωρητές ιστορικού είναι περιορισμένος και η αντιστοίχηση χρησιμοποιεί τα ελάχιστα σημαντικά ψηφία της διεύθυνσης εντολής, δεν εξασφαλίζεται ότι δεν υπάρχει επικάλυψη. Συνεπώς, υπάρχει η περίπτωση δύο διακλαδώσεις να αναγκάζονται να χρησιμοποιήσουν τους ίδιους καταχωρητές ιστορικού και μετρητές.

Στο σχήμα 2.6 βλέπουμε ένα διάγραμμα προβλεπτή διπλού επιπέδου που χρησιμοποιεί πίνακα τοπικών ιστορικών.



Σχήμα 2.6: *Πρόβλεψη με χρήση πίνακα τοπικού ιστορικού.*

2.2.4.5 Υβριδική πρόβλεψη

Δεν υπάρχει 'καλύτερη' τεχνική πρόβλεψης όσον αφορά τις τεχνικές καθολικού ιστορικού ή τοπικού. Υπάρχουν φορές όπου ο ένας καθολικός καταχωρητής ιστορικού για όλες τις διακλαδώσεις μπορεί να αποδίδει καλύτερα, ενώ άλλες όπου ένας τοπικός καταχωρητής ιστορικού, ξεχωριστός για κάθε διακλάδωση, δίνει πιο εύστοχες προβλέψεις. Κάθε διακλάδωση μπορεί να παρουσιάζει διαφορετική συμπεριφορά με τον αντίστοιχο προβλεπτή να παρουσιάζει καλύτερη ευστοχία. Ο S. McFarling παρουσίασε ένα μοντέλο υβριδικού προβλεπτή, (Hybrid predictor

ή αλλιώς Tournament Predictor) [27]. Σε αυτή την περίπτωση χρησιμοποιούμε ταυτόχρονα και τις δύο τεχνικές και χρησιμοποιούμε έναν επιπλέον καταχωρήτη, τον οποίο ενημερώνουμε κάθε φορά κατάλληλα, ανάλογα με το ποιος προβλεπτής κάνει σωστές προβλέψεις. Κάθε φορά επιλέγουμε ποια τεχνική είναι πιο αξιόπιστη για μία πρόβλεψη σύμφωνα με αυτόν τον καταχωρητή.

2.2.5 Προηγμένοι προβλεπτές αιχμής

Όλες οι παραπάνω τεχνικές πρόβλεψης αν και αποτελεσματικές, δεν είναι οι πιο πρωτοποριακές. Σε σύγχρονες αρχιτεκτονικές η πρόβλεψη εντολών διακλάδωσης γίνεται χρησιμοποιώντας τρεις διαφορετικές πληροφορίες: (1) *το καθολικό ιστορικό διακλαδώσεων*, τα διαδοχικά, δηλαδή, αποτελέσματα όλων των εντολών διακλάδωσης που εκτελούνται· (2) το *τοπικό* ιστορικό της κάθε διακλάδωσης, τα διαδοχικά δηλαδή αποτελέσματα για κάθε εντολή ξεχωριστά· και τέλος (3) το *ιστορικό μονοπατιού*, το οποίο αποτελείται από τους αριθμούς (ή αλλιώς διευθύνσεις) των εντολών διακλάδωσης που εκτελούνται. Παρακάτω παρουσιάζονται μερικές συμμετοχές από τον πιο πρόσφατο διαγωνισμό προβλεπτών (Championship Branch Prediction [15]).

2.2.5.1 Μερική αντιστοίχηση μοτίβων - Partial pattern matching

Ο συγκεκριμένος προβλεπτής [28, 29] δουλεύει συγκρίνοντας ακολουθίες δεδομένων (π.χ. αποτελέσματα διακλαδώσεων), με ακολουθίες αυξανόμενου μήκους που έχουν παρατηρηθεί, και επιστρέφει αυτή με το μεγαλύτερο μήκος ταύτισης. Η υλοποίηση επιτυγχάνεται κατακερματίζοντας δεδομένα ιστορικών, τα οποία χρησιμοποιούνται για αντιστοίχηση σε πίνακα. Ο πίνακας περιέχει εγγραφές οι οποίες δείχνουν τη πρόβλεψη μιας διακλάδωσης, χρησιμοποιώντας μετρητές κορεσμού.

2.2.5.2 Νευρωνικός προβλεπτής

Ο νευρωνικός προβλεπτής (perceptron predictor) [30, 31] λειτουργεί περίπου όπως η μερική αντιστοίχηση μοτίβων (MAM), μαθαίνοντας 'βάρη' για διαφορετικά σημεία του ιστορικού. Με αυτό τον τρόπο βελτιώνεται η ευστοχία όταν δύο διακλαδώσεις συσχετίζονται, καθώς δεν δίνεται σημασία σε μη-σχετικές διακλαδώσεις. Για την MAM, μη σχετικά ιστορικά, έχουν ως αποτέλεσμα να εκτινάσσεται ο αριθμός των διαφορετικών ακολουθιών που χρησιμοποιούνται για προβλέψεις. Αντίθετα ο νευρωνικός προβλεπτής εκπαιδεύεται και αποθηκεύει βάρη για διαφορετικές θέσεις του ιστορικού. Κατά την πρόβλεψη τα βάρη πολλαπλασιάζονται με την ακολουθία του ιστορικού της διακλάδωσης, αθροίζονται και δημιουργείται μία πρόβλεψη.

2.2.5.3 Μοντέλα ειδικού σκοπού

Μία άλλη προσέγγιση είναι η δημιουργία προβλεπτών που ανταποκρίνονται πολύ καλά σε πολύ συγκεκριμένες διακλαδώσεις, οι οποίες όμως εμφανίζονται συχνά σε όλα τα προγράμματα. Μερικά παραδείγμα είναι ο προβλεπτής εξόδου βρόχου επανάληψης [32]· ο προβλεπτής Wormhole [33] και ο μετρητής του πιο εσωτερικού βρόχου (Inner-Most Iteration loop counter) [34], οι οποίοι εντοπίζουν συσχετίσεις μεταξύ διακλαδώσεων σε εμφωλευμένες επαναλήψεις· και ο

προβλεπτής Αποθήκευσης/Φόρτωσης (Store/Load predictor), ο οποίος παρακολουθεί συσχετίσεις δεδομένων που επηρεάζουν αποτελέσματα διακλαδώσεων [35]. Οι προβλεπτές αυτοί προκύπτουν από λεπτομερής ανάλυση ειδικών και στοχεύουν σε συγκεκριμένες συμπεριφορές προγραμμάτων, που προκαλούν λάθος προβλέψεις.

2.2.5.4 Ο προβλεπτής TAGE-SC-L

Ο προβλεπτής TAGE-SC-L (ή απλα Tage) [12] είναι ο νικητής του 5ου διαγωνισμού προβλέψεων διακλαδώσεων (Championship Branch Prediction 2016 [15]) και χρησιμοποιεί ταυτόχρονα έναν Partial pattern matching προβλεπτή, όπου χρησιμοποιεί ιστορικά των οποίων τα μήκη ακολουθούν γεωμετρικές σειρές, μαζί με έναν Inner-Most Loop Iteration predictor. Για κάθε πρόβλεψη χρησιμοποιεί και τους δύο διαθέσιμους προβλεπτές και διαλέγει κάθε φορά αυτόν που τείνει να αποδίδει καλύτερα. Ο Tage προτάθηκε με 8KB μνήμης, ωστόσο υπάρχει και η υλοποίηση του με 64KB. Μεγαλύτερη μνήμη συνεπάγεται λιγότερες συγκρούσεις κατά το pattern matching και άρα μεγαλύτερη ακρίβεια προβλέψεων. Ωστόσο στη δημοσίευση *Branch Prediction Is Not A Solved Problem* [5] παρατηρήθηκε ότι περαιτέρω αύξηση των πόρων του προβλεπτή δεν συνεπάγεται ανάλογη αύξηση της ακρίβειας του.

Κεφάλαιο 3

Μηχανική μάθηση & Τεχνητά νευρωνικά δίκτυα

Μηχανική μάθηση (Machine Learning-ML) ονομάζουμε τον τομέα της τεχνητής νοημοσύνης που προσδοκεί την εκμάθηση υπολογιστικών μηχανών στην αυτόματη λήψη αποφάσεων. Για απλές εργασίες είναι εφικτό η υλοποίηση αλγορίθμων οι οποίες περιλαμβάνουν τα βήματα, προκειμένου ένας υπολογιστής να επιλύσει κάποιο πρόβλημα. Ωστόσο αυτό δεν είναι πάντα εφικτό, υπάρχουν περιπτώσεις απαιτητικών προβλημάτων όπου η δημιουργία ενός αλγορίθμου είναι μια πολύ απαιτητική διαδικασία. Εναλλακτικά μπορούμε να χρησιμοποιήσουμε αλγορίθμους και μεθόδους οι οποίοι βοηθούν τον υπολογιστή να μάθει μόνος του ποια είναι τα σωστά βήματα προκειμένου να επιλυθεί ένα πρόβλημα. Αυτό επιτυγχάνεται παρέχοντας στον υπολογιστή δεδομένα τα οποία αναλύει με σκοπό την εκμάθηση της διαδικασίας που απαιτείται προκειμένου να επιλύει μελλοντικά προβλήματα ίδιου τύπου με διαφορετικά δεδομένα. Στη συγκεκριμένη διπλωματική ασχολούμαστε κυρίως με τον τομέα των **τεχνητών νευρωνικών δικτύων**. Παρακάτω γίνεται μια μικρή εισαγωγή στους τύπους αλγορίθμων μηχανικής μάθησης και στη συνέχεια αναλύονται περαιτέρω τα νευρωνικά δίκτυα.

3.1 Τύποι αλγορίθμων μηχανικής μάθησης

Όταν προσπαθούμε να επιλύσουμε ένα πρόβλημα μηχανικής μάθησης υπάρχουν τρεις βασικοί τρόποι προσέγγισης. Αυτοί είναι:

3.1.1 Μάθηση με Επίβλεψη

Στην μάθηση με επίβλεψη προσπαθούμε να χαρακτηρίσουμε δεδομένα βασιζόμενοι σε κάποια δεδομένα εκπαίδευσης. Στον αλγόριθμο παρουσιάζεται ένα σύνολο δεδομένων εκπαίδευσης (training dataset) τα οποία περιλαμβάνουν δυάδες συνόλων εισόδου και εξόδου. Παρατηρώντας τα δεδομένα ο υπολογιστής προσπαθεί να μάθει τη συσχέτιση μεταξύ εισόδου και εξόδου ώστε να μπορεί να προβλέπει μελλοντικές τιμές με βάση κάποια 'αθέατη' είσοδο. Στην συγκεκριμένη μέθοδο υπάρχουν δύο υποκατηγορίες:

3.1.1.1 Λογιστική παλινδρόμηση (Logistic Regression)

Σε προβλήματα λογιστικής παλινδρόμησης σκοπός είναι η πρόβλεψη μιας τιμής με δεδομένη κάποια είσοδο. Παραδείγματα τέτοιων προβλημάτων μπορεί να είναι η πιθανότητα βροχής με βάση τωρινά καιρικά δεδομένα, η τιμή μιας μετοχής στο χρηματιστήριο κ.α.

3.1.1.2 Κατηγοριοποίηση (Classification)

Σε προβλήματα κατηγοριοποίησης σκοπός του εκπαιδευμένου αλγορίθμου είναι η επιλογής μιας κατηγορίας για κάποιο σύνολο δεδομένων. Παραδείγματα τέτοιων προβλημάτων είναι ο διαχωρισμός φωτογραφιών ζώων, η πρόβλεψη ενός χειρόγραφου αριθμού ή η αναγνώριση κάποιων ασθενειών με βάση τα συμπτώματα ενός ασθενή.

3.1.2 Μάθηση χωρίς Επίβλεψη

Στη συγκεκριμένη μέθοδο τα δεδομένα δεν χρειάζονται να χαρακτηριστούν εκ των προτέρων προκειμένου να γίνει εκπαίδευση όπως στην μάθηση με επίβλεψη (3.1.1). Αντιθέτως αφήνουμε το μοντέλο να επεξεργαστεί μόνο του τα δεδομένα ώστε να ανακαλύψει μοτίβα και πληροφορίες τα οποία δεν είναι ήδη γνωστά. Η ικανότητα της μεθόδου να ανακαλύπτει συσχετίσεις και διαφορές ανάμεσα σε δεδομένα, την κάνει ιδανική για ομαδοποίηση δεδομένων (clustering), εξαγωγή χαρακτηριστικών (feature extraction) και γενικότερα για προβλήματα τα οποία δεν είναι πλήρως σαφή και το σύνολο των ομάδων δεν είναι γνωστό εξ αρχής.

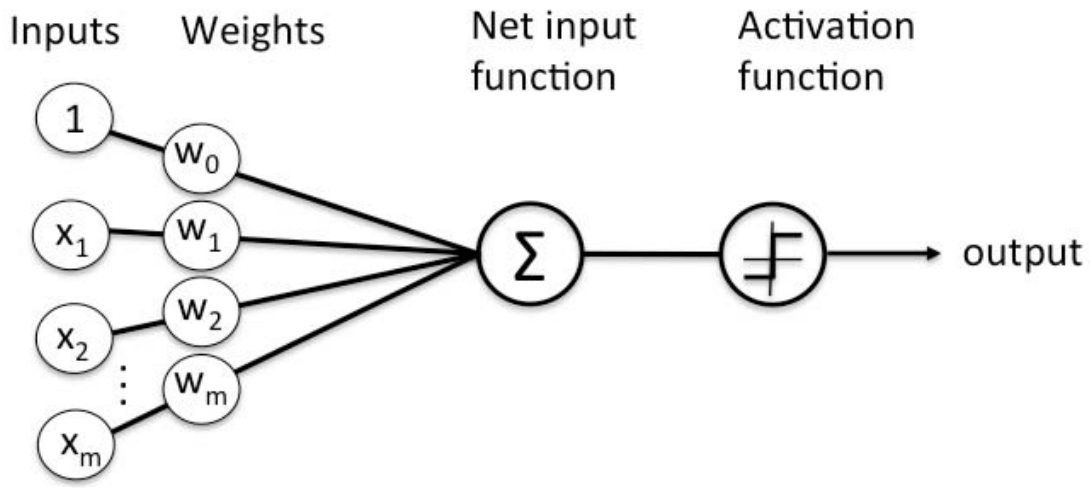
3.1.3 Ενισχυμένη μάθηση

Η ενισχυμένη μάθηση χρησιμοποιείται για την εκπαίδευση μοντέλων τα οποία μετέπειτα αξιοποιούνται για να παίρνουν διαδοχικές αποφάσεις. Το μοντέλο εκπαιδεύεται να πετυχαίνει κάποιο σκοπό σε κάποιο ενδεχομένως περίπλοκο περιβάλλον. Για κάθε δοκιμαστική απόφαση που παίρνει, το μοντέλο επιβραβεύεται ή τιμωρείται ανάλογα με το εάν φτάνει κοντά στην επίτευξη του στόχου. Η μέθοδος θυμίζει την διαδικασία ενός παιχνιδιού· το μοντέλο καθώς φτάνει κοντά στο στόχο επιβραβεύεται με πόντους, διαφορετικά χάνει.

3.2 Δομή τεχνητών νευρωνικών δικτύων

Με τον όρο τεχνητά νευρωνικά δίκτυα (Artificial Neural Networks - ANN), ή πιο απλά Νευρωνικά Δίκτυα (Neural Networks - NN) αναφερόμαστε σε υπολογιστικά συστήματα τα οποία προσπαθούν να μιμηθούν τον τρόπο λειτουργίας ενός βιολογικού εγκεφάλου. Το νευρωνικό δίκτυο, αποτελείται από πολλά δομικά στοιχεία, τους τεχνητούς νευρώνες (perceptrons), τα οποία επικοινωνούν το ένα με το άλλο, στέλνοντας σήματα, ακριβώς όπως οι νευρώνες σε έναν βιολογικό εγκέφαλο. Το «σήμα» μεταξύ δύο νευρώνων είναι η τιμή κάποιου αριθμού, μέσω μαθηματικών υπολογισμών βασισμένων στην είσοδο του νευρωνικού δικτύου. Επιπλέον οι υπολογισμοί αυτοί καθορίζονται από τα «βάρη» που χαρακτηρίζουν τον κάθε νευρώνα, τα οποία τροποποιούνται κατά το στάδιο εκπαίδευσης του δικτύου. Στο συγκεκριμένο τμήμα του κεφαλαίου θα ασχοληθούμε κυρίως με τα νευρωνικά δίκτυα που χρησιμοποιήθηκαν στην δι-

πλωματική, τα Συνελικτικά νευρωνικά δίκτυα (Convolutional Neural Networks - CNN) καθώς και τα δίκτυα Μακράς Βραχυπρόθεσμης μνήμης (Long Short-Term Memory - LSTM).



Σχήμα 3.1: *Διάταξη τεχνικού νευρώνα - Perceptron.*

3.2.1 Τεχνητοί νευρώνες - Perceptrons

Ο perceptron είναι το βασικότερο δομικό στοιχείο ενός νευρωνικού δικτύου. Η πρώτη υλοποίηση του αλγορίθμου έγινε από τον Rosenblatt το 1957 [36]. Εν συντομία, ένας perceptron αποτελεί το απλούστερο τεχνητό νευρωνικό δίκτυο που μπορεί να υπάρξει, ενός επιπέδου. Όπως βλέπουμε στο σχήμα 3.1, τα βασικά μέρη ενός perceptron είναι οι τιμές εισόδου, τα βάρη-weights, το άθροισμα και η συνάρτηση ενεργοποίησης - activation function. Επί της ουσίας αυτό που κάνει ένας perceptron είναι να υπολογίζει την εξής μαθηματική σχέση:

$$output = f(\sum_{i=0}^{m} x_i w_i + b) \tag{3.1}$$

Ο τεχνητός νευρώνας δέχεται όλες τις εισόδους πλήθους m, πολλαπλασιάζει τη κάθε μία με το αντίστοιχο βάρος w_i, τις προσθέτει και τέλος περνάει το άθροισμα από την συνάρτηση ενεργοποίησης για να παράγει το τελικό αποτέλεσμα.

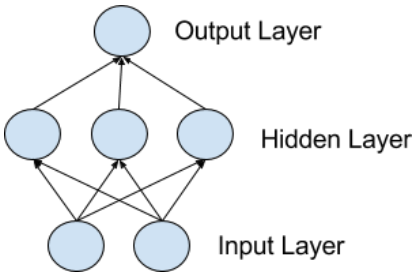
Τα βάρη καθορίζουν ποιες εισόδους είναι πιο σημαντικές προκειμένου να υπολογιστεί το επιθυμητό αποτέλεσμα. Η συνάρτηση ενεργοποίησης χρησιμοποιείται προκειμένου να αντιστοιχίζεται το αποτέλεσμα του νευρώνα σε κάποιο φραγμένο διάστημα τιμών (συνήθως (0,1) ή (-1,1)).

3.2.2 Πολυεπίπεδα Perceptron - Multi Layer Perceptron

Τα συγκεκριμένα νευρωνικά δίκτυα αποτελούνται από διατάξεις πολλών νευρώνων - perceptrons και αποτελούν την αρχή των βαθιών νευρωνικών δικτύων (Deep Neural Networks) [37]. Τα βασικά επίπεδα είναι, το επίπεδο εισόδου (input layer), το κρυφό επίπεδο (hidden layer) και το επίπεδο εξόδου (output layer). Τα επίπεδα εισόδου και εξόδου είναι πάντα από ένα, ωστόσο σε ένα βαθύ νευρωνικό δίκτυο μπορεί να έχουμε πολλά διαδοχικά κρυφά επίπεδα.

Η είσοδος δέχεται τα σήματα τα οποία επεξεργάζονται από τα κρυφά επίπεδα προκειμένου να φτάσει κάποιο τελικό σήμα στο επίπεδο εξόδου. Στο σχήμα 3.2 βλέπουμε ένα απλό τεχνητό νευρωνικό δίκτυο με ένα κρυφό επίπεδο.

Κάθε επίπεδο αποτελείται από πολλούς νευρώνες, το πλήθος των οποίων ονομάζεται πλάτος επιπέδου (layer width). Οπως περιγράφηκε στο 3.2.1, κάθε νευρώνας δέχεται ως είσοδο σήματα, τα οποία στη συγκεκριμένη περίπτωση αποτελούνται από εξόδους νευρώνων κάποιου προηγούμενου επιπέδου. Στη συνέχεια τα σήματα αυτά επεξεργάζονται από τον κάθε perceptron προκειμένου να τροφοδητηθούν στο επόμενο επίπεδο μέχρι την τελική έξοδο. Ο αριθμός των νευρώνων εισόδου και εξόδου καθορίζεται αποκλειστικά από το είδος του προβλήματος. Για παράδειγμα εάν η είσοδος πρόκειται για μια εικόνα, τότε το πλήθος των perceptrons στην είσοδο, θα μπορούσε να ταυτίζεται με την ποσότητα των pixel στην εικόνα. Ενώ αν έχουμε έναν πρόβλημα κατηρηγοποίησης (classification) τότε η έξοδος θα έχει πλάτος, ίσο με το πλήθος των κλάσεων. Ωστόσο το πλήθος των κρυφών επιπέδων, καθώς και το πλάτος αυτών, δεν είναι συγκεκριμένο. Κατά κανόνα, πολλά κρυφά επίπεδα τείνουν να επιλύουν δυσκολότερα και πολυπλοκότερα προβλήματα.

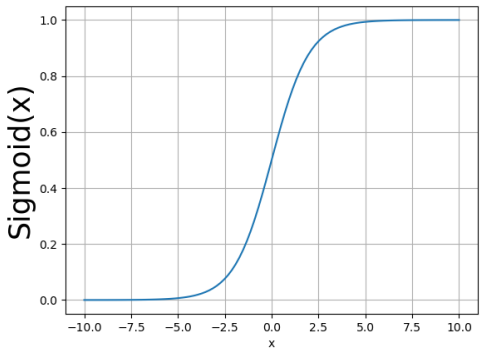


Σχήμα 3.2: *Απλό διάγραμμα πολυεπίπεδου νευρωνικού δικτύου.*

3.2.3 Συνάρτηση ενεργοποίησης - Activation function

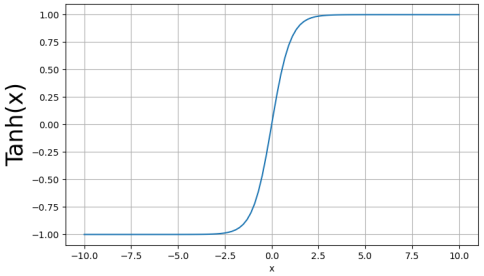
Η συνάρτηση ενεργοποίησης αποτελεί τον συνδετικό κρίκο μεταξύ των διάφορων στρωμάτων μεταξύ των νευρώνων. Επί της ουσίας είναι ένας κόμβος απόφασης στον οποίο καθορίζεται εάν η επεξεργασμένη πληροφορία του νευρώνα θα περάσει στο επόμενο στρώμα. Οι συναρτήσεις ενεργοποίησης αντιστοιχίζουν το αποτέλεσμα μεταξύ ενός φραγμένου πεδίου τιμών, συνήθως το $(0, 1)$. Μερικές από τις πιο βασικές συναρτήσεις ενεργοποίησης παρουσιάζονται συνοπτικά παρακάτω:

- Σιγμοειδής συνάρτηση (Sigmoid function): αντιστοιχεί τιμές στο διάστημα $(0, 1)$ με την εξής ιδιαιτερότητα· οι πολύ μεγάλες τιμές συγκλίνουν στο 1 και οι πολύ μικρές στο 0. Το θετικό στη συγκεκριμένη συνάρτηση είναι ότι προσφέρει ξεκάθαρες τιμές πρόβλεψης, όπως φαίνεται και στο διάγραμα 3.3. Ωστόσο το γεγονός αυτό οδηγεί στο λεγόμενο πρόβλημα της εξασθένισης κλίσης (Vanishing gradient), το οποίο κάνει πιο αργή τη διαδικασία της μάθησης, όπως θα αναπτύξουμε παρακάτω.



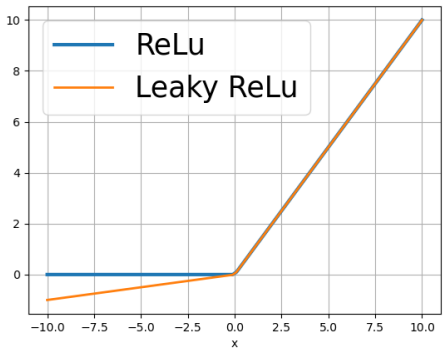
Σχήμα 3.3: *Σιγμοειδής συνάρτηση.*

- Υπερβολική εφαπτόμενη (Hyperbolic Tangent): Μοιάζει αρκετά με την σιγμοειδή με την διαφορά ότι αντιστοιχεί στο διάστημα $(-1, 1)$. Το βασικό πλεονέκτημα της είναι ότι δεν μεταβάλει σημαντικά τις τιμές κοντά στο 0. Όπως βλέπουμε στο γράφημα 3.4 υπάρχει αντιστοίχηση των πολύ μεγάλων τιμών στο 1, των πολύ μικρών στο -1 αντίστοιχα, και συνεπώς παρουσιάζεται και εδώ το πρόβλημα Vanishing Gradient, σε πολύ μικρότερο όμως βαθμό.



Σχήμα 3.4: *Υπερβολική εφαπτόμενη συνάρτηση.*

- Rectified Linear Unit [38]: Αποτελεί από τις πλέον διαδεδομένες συναρτήσεις ενεργοποίησης. Όπως βλέπουμε στο σχήμα 3.5 οι αρνητικές τιμές μηδενίζονται με αποτέλεσμα να παύουν να λαμβάνουν μέρος στη διαδικασία της μάθησης. Το γεγονός αυτό εξασφαλίζει χαμηλό υπολογιστικό κόστος και μειώνει το χρόνο εκμάθησης του δικτύου. Ωστόσο δημιουργεί ένα βασικό πρόβλημα καθώς «νεκρώνονται» νευρώνες που λαμβάνουν αρνητικές τιμές και δεν χρησιμοποιούνται ποτέ κατά τη διάρκεια της εκμάθησης. Το πρόβλημα αυτό συναντάται συχνά στη βιβλιογραφία ως 'dying ReLu', δηλώνοντας την αδυναμία της ReLu να αξιοποιήσει κάποιο νευρώνα που έχει λάβει αρνητική τιμή. Την αδυναμία αυτή έρχεται να αντιμετωπίσει μια παραλλαγή της, η Leaky ReLu, η οποία αντιστοιχεί τις αρνητικές τιμές με μία γραμμική συνάρτηση πολύ μικρής κλίσης, ώστε να δίνει την δυνατότητα σε ένα νευρώνα να επανέλθει.



Σχήμα 3.5: *Rectified Linear Unit και Leaky ReLu.*

- Softmax: Η συγκεκριμένη συνάρτηση δεν χρησιμοποιείται ως ενδιάμεση συνάρτηση ενεργοποίησης, αλλά συναντάται στην έξοδο του νευρωνικού δικτύου, ανεξάρτητα από τις ενδιάμεσες συναρτήσεις ενεργοποίησης. Όταν έχουμε ένα πρόβλημα κατηγοριοποίησης (classification), στο τελευταίο στρώμα ο κάθε νευρώνας αντιστοιχεί σε μία διαφορετική κλάση. Οι τιμές που λαμβάνει ο κάθε νευρώνας υποδεικνύει την πιθανότητα η είσοδος να «ταιριάζει» στη συγκεκριμένη κλάση. Εφόσον λοιπόν έχουμε να κάνουμε με πιθανότητες απαιτείται μια κανονικοποίηση των τιμών στην έξοδο. Αυτό ακριβώς επιτυγχάνεται με την συνάρτηση softmax. Διαμορφώνει τις τιμές εξόδου ώστε το συνολικό άθροισμα να είναι ίσο με 1 και να εκφράζονται καλύτερα οι πιθανότητες της κάθε κλάσης. Ο τύπος της συνάρτησης softmax είναι:

$$Softmax(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{J} e^{x_j}} \tag{3.2}$$

3.3 Εκπαίδευση νευρωνικών δικτύων

Η διαδικασία με την οποία μεταβάλλουμε τις παραμέτρους του νευρωνικού δικτύου (τα βάρη), προκειμένου να μάθει πως να προβλέπει σωστά αποτελέσματα, ονομάζεται εκπαίδευση του νευρωνικού δικτύου. Εκτός από την αρχιτεκτονική που χαρακτηρίζει κάθε νευρωνικό δίκτυο, εξίσου σημαντική παράμετρος αποτελούν τα «βάρη» του κάθε νευρώνα, τα οποία καθορίζουν πως η πληροφορία διέρχεται μέσα από το δίκτυο, προκειμένου να υπολογιστεί η κατάλληλη τιμή στην έξοδο. Ένα αποτελεσματικό νευρωνικό δίκτυο πρέπει να μπορεί να εκπαιδεύεται γρήγορα, σε κάποιο πεπερασμένο χρόνο και χρησιμοποιώντας όσο το δυνατόν λιγότερο υπολογιστικούς πόρους. Παράλληλα πρέπει να είναι σε θέση να γενικεύει στο πρόβλημα το οποίο εκπαιδεύεται. Με πιο απλά λόγια αυτό σημαίνει πως αφού εκπαιδευτεί με βάση κάποια γνωστά δεδομένα, θα μπορεί να κάνει σωστές προβλέψεις για δεδομένα που δεν του έχουν ξαναπαρουσιαστεί. Η εκπαίδευση του νευρωνικού δικτύου αποτελεί μια επαναληπτική διαδικασία κατά την οποία τροποποιούμε παραμέτρους του, προκειμένου να επιτύχουμε την επιθυμητή έξοδο με βάση ένα σύνολο δεδομένων εκπαίδευσης (**training set**). Η κάθε επανάληψη της διαδικασίας ονομάζεται εποχή (**epoch**) και ο αριθμός τον εποχών που χρησιμοποιούμε παίζει καθοριστικό ρόλο στην απόδοση του νευρωνικού δικτύου. Σπουδαίο ρόλο εδώ παίζει και η

συνάρτηση κόστους καθώς και ο αλγόριθμος βελτιστοποίησης που παρουσιάζονται παρακάτω.

3.3.1 Συνάρτηση κόστους - Cost function

Η συνάρτηση κόστους αποσκοπεί στον έλεγχο της εκπαίδευσης και υπολογίζει πόσο κοντά βρίσκεται η έξοδος στο επιθυμητό αποτέλεσμα και συνήθως συμβολίζεται με $J(\theta)$. Η πιο γνωστή συνάρτηση κόστους χρησιμοποιεί εντροπία και ονομάζεται Crossentropy Loss. Το κόστος υπολογίζεται με βάση τα βάρη του δικτύου όπως φαίνεται και στον τύπο 3.3

$$H(p,q) = -\sum_{i=0}^{N} p(x_i)\log\left(q(x_i)\right) \tag{3.3}$$

όπου N το συνολικό πλήθος των κατηγοριών - κλάσεων, $p(x_i)$ η εκτιμώμενη τιμή για την παρατήρηση x_i και $q(x_i)$ η *posterior* πιθανότητα το δείγμα x_i να ανήκει στη κλάση i. Η λειτουργία αυτής της συνάρτησης είναι η σύγκριση των δύο πιθανοτηκών κατανομών, της πρόβλεψης και της αναμενόμενης εξόδου.

3.3.2 Αλγόριθμος βελτιστοποίησης

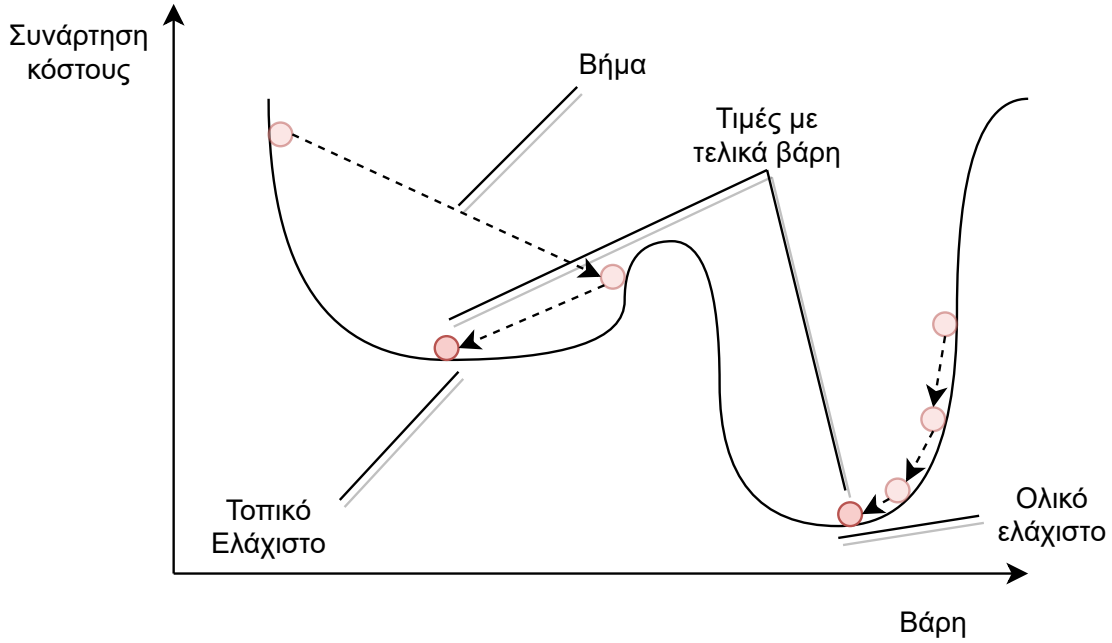
Με τον όρο αλγόριθμος βελτιστοποίησης **(Optimization algorithm)**, αναφερόμαστε στην διαδικασία εύρεσης της κατάλληλης εισόδου, ή παραμέτρων, μιας συνάρτησης προκειμένου να επιτύχουμε την μέγιστη, ή ελάχιστη, έξοδο της συνάρτησης αυτής. Κατά τη διαδικασία εκμάθησης του δικτύου, ο σκοπός είναι η κατάλληλη τροποποίηση των βαρών του κάθε νευρώνα, προκειμένου να επιτύχουμε σωστές προβλέψεις, όσο τον δυνατόν πιο σύντομα. Οι αλγόριθμοι βελτιστοποίησης αποτελούν τον συνδετικό κρίκο μεταξύ της συνάρτησης κόστους και των παραμέτρων του δικτύου. Με απλά λόγια ο βελτιστοποιητής (optimizer), αλλάζει και διαμορφώνει το νευρωνικό δίκτυο, προκειμένου να έρθει στο επιθυμητό αποτέλεσμα, με οδηγό την συνάρτηση κόστους, η οποία υποδεικνύει πότε οι αλλαγές που κάνει επιφέρουν θετικά ή αρνητικά αποτελέσματα.

Ένας από τους αρχικούς optimizers είναι ο Gradient Descent. Στόχος είναι η ελαχιστοποίηση της συνάρτησης κόστους, με χρήση της κλήσης (gradient) των παραμέτρων του προβλήματος, όπου παράμετροι αποτελούν τα βάρη του νευρωνικού δικτύου. Ο αλγόριθμος εκτελείται επαναληπτικά, και οι παράμετροι του δικτύου ανανεώνονται με βάση την παρακάτω εξίσωση:

$$\theta_{t+1} = \theta_t - \lambda\, \nabla_\theta\, J(\theta) \tag{3.4}$$

με θ το σύνολο των παραμέτρων της εξίσωσης, λ τον ρυθμό μάθησης (learning rate) και $J(\theta)$ την συνάρτηση κόστους. Εν γένει χρησιμοποιώντας παραγώγους της συνάρτησης κόστους, προσπαθούμε να επαναπροσδιορίσουμε τα βάρη, τα οποία θα μας εξασφαλίσουν το ολικό ελάχιστο της συνάρτησης κόστους (σχήμα 3.6).

Μια παραλλαγή του Gradient Descent είναι ο Stochastic Gradient Descent (SGD) κατά τον οποίο ακολουθείται η ίδια διαδικασία με την διαφορά όμως ότι δεν χρησιμοποιείται όλο το dataset κατά τον υπολογισμό παρά μόνο μικρά πακέτα (Mini-batches). Εμπειρικά τα πακέτα αυτά έχουν μέγεθος 16 έως 64 δείγματα άνα επανάληψη. Η συγκεκριμένη μέθοδος εξασφαλίζει ταχύτητα, καθώς οι υπολογισμοί μπορούν να εκτελεστούν ταυτόχρονα σε πολ-



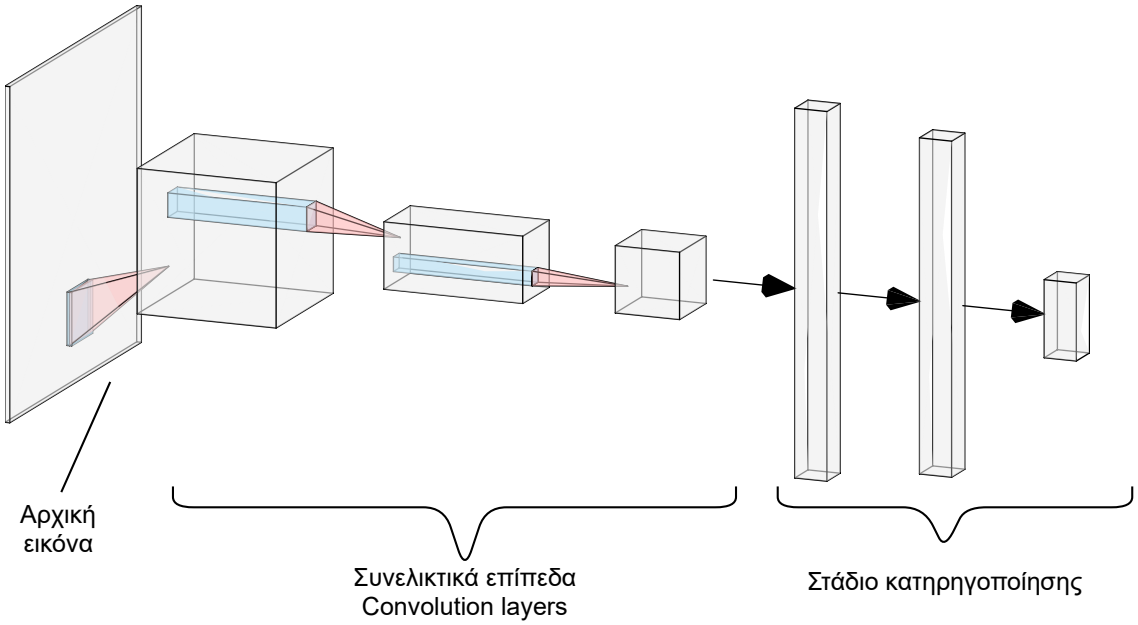
Σχήμα 3.6: *Gradient Descent.*

λούς διαφορετικούς επεξεργαστές καθώς το σύνολο των δεδομένων μπορεί να είναι τεράστιο. Ο συγκεκριμένος αλγόριθμος ωστόσο παρουσιάζει κάποια προβλήματα. Ο σταθερός ρυθμός μάθησης λ σε συνδυασμό με τη μεγάλη διασπορά σύγκλησης σε προβλήματα με πολλά δεδομένα, οδήγησε στην αναζήτηση, άλλων, αποδοτικότερων αλγορίθμων βελτιστοποίησης.

Ένας τέτοιος αποδοτικότερος αλγόριθμος είναι ο Adam (Adaptive moment estimation) ο οποίος είναι επέκταση του SGD. Ο αλγόριθμος Adam χρησιμοποιεί μεταβλητό ρυθμό μάθησης, διαφορετικό για κάθε βάρος (παράμετρο) του δικτύου, ενώ αντλεί πληροφορία τόσο από τη πρώτη βαθμίδα κλίσης όσο και από τη δεύτερη. Έτσι επιτυγχάνεται από τη μία γρήγορη σύγκλιση στο ολικό ελάχιστο χωρίς να υπάρχει κίνδυνος εγκλωβισμού σε κάποιο τοπικό ελάχιστο όπως φαίνεται στο σχήμα 3.6.

3.4 Συνελικτικά νευρωνικά δίκτυα

Τα συνελικτικά νευρωνικά δίκτυα (Convolutional neural networks), γνωστά και ως CNN ή ConvNet, είναι ομάδα νευρωνικών δικτύων που εξειδικεύονται στην επεξεργασία δεδομένων τα οποία έχουν χωρική τοπολογία, όπως οι εικόνες. Είναι εμπνευσμένα από τον τρόπο που λειτουργεί η βιολογική όραση και στην σύγχρονη μορφή τους παρουσιάστηκαν από τον Y. Lecun [39] ο οποίος τα εκπαίδευσε ώστε να αναγνωρίζουν χειρόγραφους αριθμούς. Μια ψηφιακή εικόνα απαρτίζεται από μια σειρά εικονοκυττάρων (pixels) διατεταγμένα σε ένα πλέγμα, όπου το κάθε ένα περιέχει πληροφορία για την φωτεινότητά του καθώς και το χρώμα που περιέχει. Το ανθρώπινο μάτι όταν κοιτάει μια εικόνα, κάθε κύτταρο-νευρώνας επικεντρώνεται σε διαφορετικό σημείο και όλα μαζί συνεργάζονται για να προσφέρουν μια τελική εικόνα. Κατά αναλογία έτσι δουλεύει και ένα CNN μοντέλο. Τα στρώματα-layers του δικτύου διατάσσονται με τέτοιο τρόπο ώστε πρώτα να αναλύονται απλά σχήματα (γραμμές, κύκλοι) και στη συνέχεια πιο περίπλοκα (πρόσωπα, αντικείμενα).



Σχήμα 3.7: *Παράδειγμα ενός συνελικτικού δικτύου.*

Το στρώμα συνέλιξης είναι το βασικό τμήμα ενός συνελικτικού δικτύου. Επί της ουσίας υπολογίζει το εσωτερικό γινόμενο (dot-product) μεταξύ δύο πινάκων, όπου ο ένας πίνακας είναι ένα σύνολο «βαρών» προς εκπαίδευση, γνωστά και ως *kernel*, και ο άλλος πίνακας είναι ένα υποσύνολο των δεδομένων προς ανάλυση, π.χ. ένα κομμάτι μιας εικόνας. Το kernel είναι χωρικά μικρότερο από την εικόνα, έχει όμως το ίδιο βάθος. Ως βάθος ορίζουμε τα τα «κανάλια» (*channels*) που απαρτίζουν την είκονα, για παράδειγμα μια έγχρωμη εικόνα αποτελείται από τρία κανάλια για κάθε διαφορετικό χρώμα (το γνωστό RGB). Σε αυτή τη περίπτωση το kernel θα είχε πολύ μικρότερο μήκος και πλάτος από την εικόνα, ωστόσο το ίδιο βάθος. Κατά την επεξεργασία το kernel *περνά* πάνω από την εικόνα υπολογίζοντας νέες τιμές. Αυτό έχει σαν αποτέλεσμα μία νέα εικόνα, τον χάρτη ενεργοποίησης - activation map, ο οποίος μας δείχνει το αποτέλεσμα του kernel σε κάθε χωρικά αντίστοιχο σημείο της εικόνας. Το *πόσο* μετακινείται το kernel ονομάζεται βήμα - stride. Στο σχήμα 3.7 βλέπουμε ένα παράδειγμα συνελικτικού δικτύου.

Εάν έχουμε εικόνα μήκους και πλάτους $\mathbf{W}$, βάθους $\mathbf{D}$, και ορίσουμε ως βάθος του kernel την τιμή $\mathbf{D}_{out}$, τότε μπορούμε να υπολογίσουμε τις διαστάσεις του αποτελέσματος, του χάρτη ενεργοποίησης - activation map, με τον εξής τύπο:

$$W_{out} = \frac{W - F + 2P}{S} + 1 \tag{3.5}$$

όπου $\mathbf{F}$ οι διαστάσεις του kernel, $\mathbf{P}$ το padding, οι διαστάσεις του υποσύνολου της εικόνας που χρησιμοποιεί το kernel και $\mathbf{S}$ το stride. Έτσι ο χάρτης ενεργοποίησης θα είχε διαστάσεις $W_{out} \times W_{out}$ και βάθος D_{out}.

3.5 Επαναλαμβανόμενα νευρωνικά δίκτυα

Τα επαναλαμβανόμενα νευρωνικά δίκτυα (recurrent neural networks), εξειδικεύονται σε ακολουθιακά προβλήματα, όπως μετάφραση κειμένου, παραγωγή κειμένου και αναγνώριση φωνής [40]. Το βασικότερο χαρακτηριστικό τους είναι ότι περιέχουν εσωτερικά βρόγχους, οι οποίοι τους επιτρέπουν να κρατούν πληροφορίες (μνήμη) και να χρησιμοποιούν προηγούμενες εισόδους σε συνδυασμό με νέες.

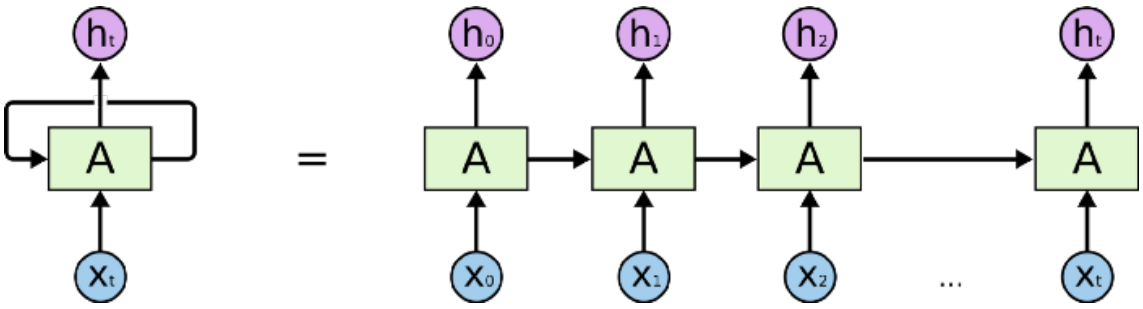
Στο σχήμα 3.8 βλέπουμε πως το RNN μπορεί να παρομοιαστεί με μία αλυσίδα του απλού κυκλώματος A, όπου κάθε κελί προσφέρει πληροφορία στο επόμενο. Έτσι εάν θέλουμε να μεταφράσουμε μια πρόταση, η κάθε είσοδος x_i στο δίκτυο θα μπορούσε να είναι μία λέξη της πρότασης. Το κάθε κύκλωμα A θα μετέφραζε τη κάθε λέξη αντλώντας πληροφορία τόσο από τη λέξη προς μετάφραση, όσο και από τις προηγούμενες στην πρόταση, πράγμα ιδιαίτερα σημαντικό για το συγκεκριμένο πρόβλημα. Τα RNN αν και ικανά να χρησιμοποιούν προηγούμενες πληροφορίες σε μια ακολουθία, στη πράξη προκύπτει ότι δεν αποδίδουν σε ακολουθίες όπου η σημαντική πληροφορία για κάποιο στάδιο, μπορεί να βρίσκεται αρκετά βήματα πίσω. Το πρόβλημα αυτό προσπαθούν να λύσουν τα δίκτυα Μακράς Βραχυπρόθεσμης Μνήμης - Long Short-Term Memory (LSTM) τα οποία θα αναλυθούν στην συνέχεια.

3.5.1 Δίκτυα μακράς βραχυπρόθεσμης μνήμης

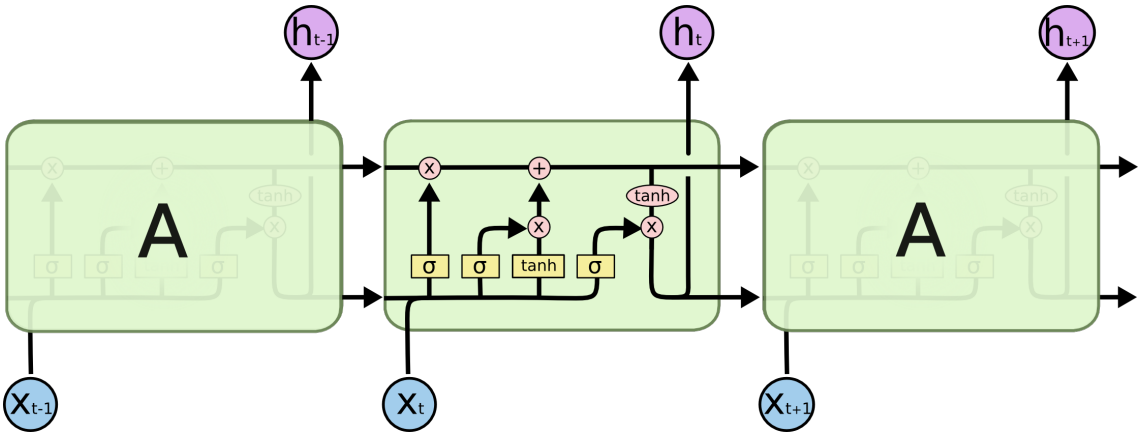
Τα δίκτυα μακράς βραχυπρόθεσμης μνήμης (Long Short-Term Memory - LSTM) [6] επιτελούν μικρές τροποποιήσεις στα δεδομένα μέσω πολλαπλασιασμών και προσθέσεων. Η πληροφορία διέρχεται μέσα από έναν μηχανισμό που ονομάζεται Κατάσταση Κελιού (Cell State). Με τον τρόπο αυτό το δίκτυο αποκτά την ικανότητα να μπορεί να «ξεχνά» ή να «θυμάται» δεδομένα ανάλογα με το εάν θεωρούνται σημαντικά ή όχι.

Κάθε κελί δέχεται την είσοδο x_t, την κατάσταση του προηγούμενο κελιού και την έξοδο του προηγούμενου κελιού. Επεξεργάζεται τα δεδομένα και παράγει τόσο μια έξοδο, όσο και μια κρυφή κατάσταση κελιού η οποία αξιοποιείται από το επόμενο κελί στην αλυσίδα. Τα βήματα αυτά μπορούμε να τα δούμε στο διάγραμμα 3.9, μαζί με κάποιες περισσότερες λεπτομέρειες για την εσωτερική λειτουργία του κελιού.

Παρά την βελτιωμένη απόδοση των LSTM στο πρόβλημα των μακρών ακολουθιών, υπάρχει όριο στο πόσο καλά αποδίδουν σε μεγάλες ακολουθίες δεδομένων. Όσο αυξάνουμε το μήκος των ακολουθιών το μοντέλο δυσκολεύεται να αναλύσει ποια δεδομένα είναι σημαντικά και ποια όχι. Ακόμα ένα πρόβλημα των LSTM, και γενικότερα των RNN, είναι πως απαιτούν



Σχήμα 3.8: *Επαναλαμβανόμενο νευρωνικό δίκτυο.*



Σχήμα 3.9: *Δίκτυο μακράς βραχυπρόθεσμης μνήμης.*

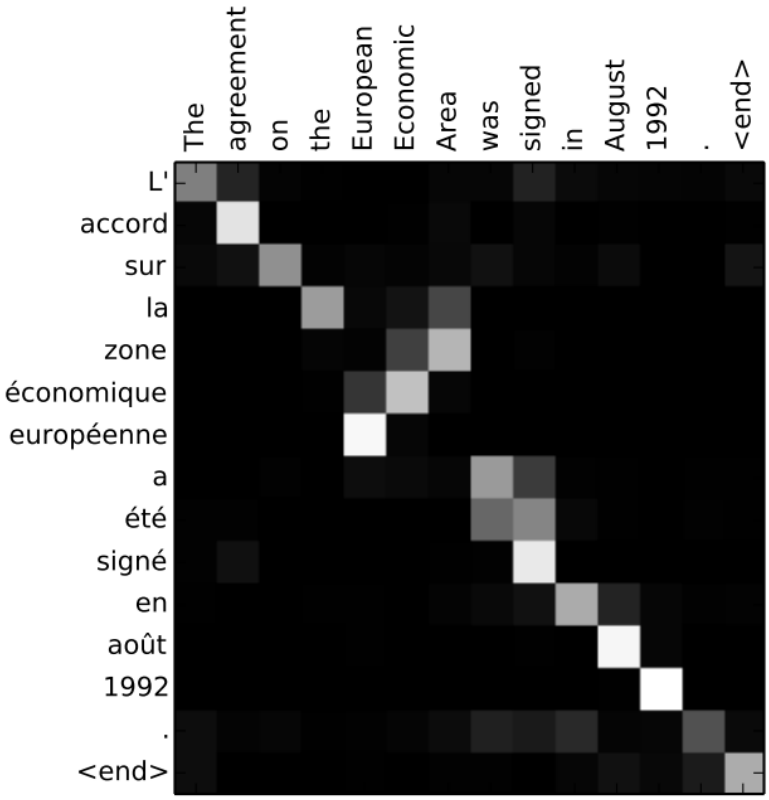
πολλούς υπολογιστικούς πόρους για να μπορέσουν να εκπαιδευτούν. Λόγο της φύσης τους δεν μπορούν να παραλληλοποιηθούν σωστά, αφού επεξεργάζονται τα δεδομένα ένα-ένα.

3.5.2 Μηχανισμός προσοχής

Ο μηχανισμός προσοχής (attention mechanism/layer) προτάθηκε αρχικά για προβλήματα ακολουθίας-σε-ακολουθία [41] (Sequence-to-Sequence) (π.χ. μετάφραση κειμένου), ωστόσο μπορεί να εφαρμοστεί σε οποιοδήποτε πρόβλημα σχετίζεται με ακολουθίες. Με τον συγκεκριμένο μηχανισμό, κάθε έξοδος δεν παράγεται από σε συνδυασμό με την αμέσως προηγούμενη κρυφή κατάσταση (όπως στο LSTM), αλλά σε συνδυασμό με όλες τις προηγούμενες κρυφές καταστάσεις. Δεν πρόκειται για μία απλή πράξη πολλαπλασιασμού πινάκων, αλλά μια διαδικασία «προσοχής». Κάθε ξεχωριστή έξοδος προκύπτει από το *sum-product* των βαρών του μηχανισμού προσοχής (**attention weights**) και όλων των κρυφών καταστάσεων. Τα βάρη προσοχής είναι ξεχωριστά για κάθε έξοδο συνεπώς η κάθε έξοδος εστιάζει, 'προσέχει', σε διαφορετικές κρυφές καταστάσεις σε όλη την ακολουθία. Στην εικόνα 3.10 βλέπουμε πως λειτουργεί ο μηχανισμός προσοχής, ώστε να μεταφράσει μια φράση. Η αγγλική φράση «European Economic Area» στα γαλλικά έχει σε διαφορετική σειρά τις λέξεις, ωστόσο με τον μηχανισμό προσοχής δεν αποδίδεται σωστά μόνο η μετάφραση του κάθε όρου ξεχωριστά. Δίνοντας 'προσοχή' σε προηγούμενες και επόμενες λέξεις μέσα στη φράση, το μοντέλο καταφέρνει να αποδώσει σωστά την μετάφραση.

3.6 Transformer

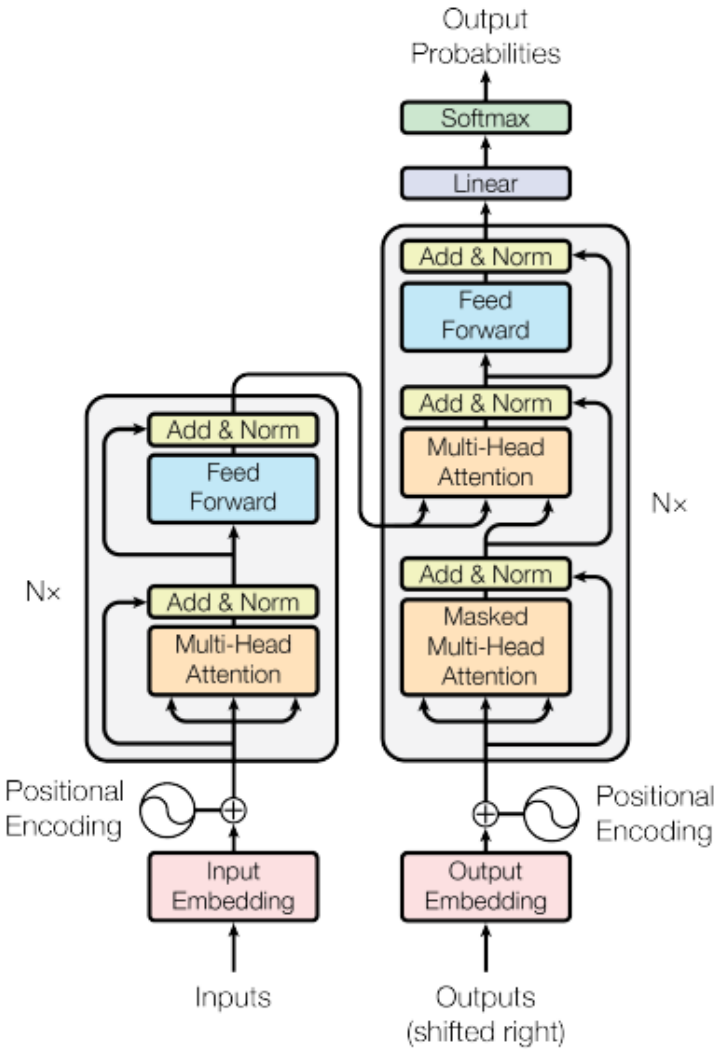
Στο κεφάλαιο 3.5.2 περιγράψαμε τον μηχανισμό προσοχής και πως αυτός βελτιώνει την απόδοση Επαναλαμβανόμενων Νευρωνικών Δικτύων (RNN), ωστόσο τα συγκεκριμένα δίκτυα εκπαιδεύονται αρκετά αργά, ειδικότερα όταν έχουμε να κάνουμε με μεγάλα δεδομένα και μεγάλες σε μήκος ακολουθίες. Στη δημοσίευση **Attention is all you need** [42] προτάθηκε ένα νέο είδος μοντέλου, το **Transformer**, το οποίο βασίζεται εξ ολοκλήρου στον μηχανισμό προσοχής, παρακάμπτοντας εντελώς την χρήση των RNN, προσφέροντας από τη μία επιπλέον απόδοση ενώ ταυτόχρονα μειώνει τον απαιτούμενο χρόνο εκπαίδευσης του μοντέλου. Στο σχήμα 3.11 βλέπουμε ένα διάγραμμα της αρχιτεκτονικής του.



Σχήμα 3.10: *Παρατηρούμε πως το μοντέλο δίνει σωστή μετάφραση της φράσης European Economic Area, δίνοντας προσοχή στα κατάλληλα σημεία. Στα γαλλικά η φράση αντιστρέφεται ("européenne économique zone") σε σύγκριση με τα αγγλικά. Ωστόσο δίνοντας προσοχή σε προηγούμενες και επόμενες λέξεις, καταφέρνει να αποδώσει σωστά τη μετάφραση.*

Το βασικότερο πλεονέκτημα του είναι ότι σε αντίθεση με τα κλασσικά RNN που επεξεργάζονται τα δεδομένα μιας ακολουθίας ένα-ένα, όπως το διάβασμα μιας πρότασης, ο Transformer έχει τη δυνατότητα να επεξεργάζεται όλα τα δεδομένα παράλληλα, μειώνοντας δραματικά το χρόνο υπολογισμού. Τα συγκεκριμένα μοντέλα ειδικεύονται σε προβλήματα ακολουθίας-σε-ακολουθία, όπως η παραγωγή κειμένου, η μετάφραση κειμένου κ.α. [11].

Ο Transformer ακολουθεί την κλασσική αρχιτεκτονική κωδικοποιητή-αποκωδικοποιητή (encoder-decoder) που χρησιμοποιείται για πολλά προβλήματα ακολουθίας-σε-ακολουθία (sequence-2-sequence). Ωστόσο εσωτερικά κάθε μέρος του αποτελείται από πολλούς στοιβαγμένους κωδικοποιητές (και αποκωδικοποιητές αντίστοιχα). Ο κάθε κωδικοποιητής αποτελείται από έναν μηχανισμό προσοχής και ένα τροφοδοτούμενο προς τα εμπρός δίκτυο (feed forward network). Σκοπός του κωδικοποιητή είναι να αναγνωρίσει την συσχέτιση του κάθε στοιχείου της ακολουθίας με τα υπόλοιπα. Από την άλλη ο αποκωδικοποιητής δέχεται σαν είσοδο την ακολουθία εξόδου, μετατοπισμένη ένα βήμα πίσω σε σχέση με την είσοδο. Σκοπός του είναι να βρει συσχετίσεις των δεδομένων της ακολουθίας εξόδου, ώστε σε συνδυασμό με τα δεδομένα που λαμβάνει από τον κωδικοποιητή να μπορέσει να προβλέψει το επόμενο βήμα της ακολουθίας εξόδου. Η διαδικασία μπορεί να γίνεται παράλληλα κατά την εκπαίδευση αφού δεν χρειάζεται να επεξεργάζονται τα δεδομένα της ακολουθίας ένα-ένα (όπως για παράδειγμα σε ένα LSTM, όπου ανανεώνεται η κρυφή κατάσταση των κελιών), προσδίδοντας έτσι δυνατότητα παραλληλοποίησης του μοντέλου και μειώνοντας κατά πολύ τον απαιτούμενο χρόνο



Σχήμα 3.11: *Το μοντέλο Transformer από τη δημοσίευση Attention is all you need.*

εκπαίδευσης.

Κεφάλαιο 4

Πρόβλεψη Διαδκλαδώσεων με Τεχνητά Νευρωνικά Δίκτυα

Στο κεφάλαιο αυτό παρουσιάζονται η μεθοδολογία που ακολουθήσαμε προκειμένου να βελτιώσουμε την πρόβλεψη διακλαδώσεων - branch prediction. Προκειμένου να βελτιώσουμε την πρόβλεψη διακλαδώσεων, αναγνωρίζουμε μια ειδική κατηγορία, τα Δύσκολα-Να-Προβλεφθούν branches (Hard-To-Predict branches). Αρχικά παρουσιάζεται πως έγινε η κατηγοριοποίηση των δύσκολων-να-προβλεφθούν διακλαδώσεων (H2P branches). Στη συνέχεια γίνεται ανάλυση για την συλλογή των δεδομένων (dataset) που χρησιμοποιήθηκαν για να εκπαιδευτούν τα νευρωνικά δίκτυα. Τέλος παρουσιάζεται η κατασκευή των μοντέλων, οι λόγοι που επιλέχθηκαν τα συγκεκριμένα μοντέλα καθώς και πως έγινε η εκπαίδευση καθώς και ο έλεγχος της επίδοσής τους. Όλος ο πηγαίος κώδικας που χρησιμοποιήθηκε για την υλοποίηση των μοντέλων βρίσκεται στο repository [43].

4.1 Ο προσομοιωτής ChampSim

Αρχικά θα αναφερθούμε στο βασικότερο εργαλείο αυτής της διπλωματικής, τον προσομοιωτή ChampSim [7].

Ο προσομοιωτής ChampSim έχει χρησιμοποιηθεί από διαγωνισμούς σχετικά με Branch Prediction και Data Prefetching. Λειτουργεί με traces τα οποία έχουν δημιουργηθεί με το εργαλείο Pin [44]. Για κάθε προσομοίωση δέχεται σαν είσοδο κάποιο trace και στη συνέχεια προσομοιώνει ολόκληρη την εκτέλεση του προγράμματος, υπολογίζοντας παράλληλα την απόδοση όλων των μερών της εκάστοτε αρχιτεκτονικής (απόδοση branch predictor, απόδοση πολιτικών αντικατάστασης της cache, απόδοση του pipeline κτλ.).

Χρησιμοποιήσαμε τον συγκεκριμένο προσομοιωτή τόσο για να εντοπίσουμε Hard-to-Predict branches όσο και για να συλλέξουμε dataset για εκπαίδευση των νευρωνικών μοντέλων, αλλά και για να ελέγξουμε την απόδοση των μοντέλων αυτών. Δημιουργήσαμε την κλάση Branch-Statistics η οποία κρατάει στατιστικά για κάθε branch κατά την εκτέλεση της προσομοίωσης. Ακόμα επεκτείναμε τον ChampSim προκειμένου να τον χρησιμοποιήσουμε σαν εργαλείο συλλογής δεδομένων (Dataset), προκειμένου να χρησιμοποιηθούν για εκπαίδευση των νευρωνικών δικτύων. Τέλος συνδέσαμε τον ChampSim με τα νευρωνικά μοντέλα, προκειμένου να ελέγξουμε την απόδοση που προσφέρουν αυτά, σε μια ολοκληρωμένη αρχιτεκτονική.

4.1.1 Δημιουργία των traces

Με τον όρο trace αναφερόμαστε σε αρχεία τα οποία περιέχουν όλη την πληροφορία για την εκτέλεση ενός προγράμματος. Τα traces δίνονται σαν είσοδο σε έναν προσομοιωτή αρχιτεκτονικής (στην περίπτωση μας αυτός είναι ο ChampSim) με σκοπό την ανάλυση μιας εκτέλεσης ενός προγράμματος. Προκειμένου να συλλέξουμε περισσότερη πληροφορία, ώστε να επιτύχουμε καλύτερη απόδοση των νευρωνικών δικτύων, δημιουργήσαμε δικά μας traces από την σουίτα προγραμμάτων SPEC 2017 [8],χρησιμοποιώντας επιπλέον εισόδους.

4.1.1.1 Είσοδοι εφαρμογής

Για να επιτύχουμε καλύτερα αποτελέσματα και να μπορέσουμε να αναλύσουμε περισσότερα δεδομένα δημιουργήσαμε δικά μας traces, από πολλές διαφορετικές εκτελέσεις, με διαφορετικές εισόδους. Οι είσοδοι που χρησιμοποιήσαμε ήταν από τη σχετική δημοσίευση Alberta workloads for SPEC 2017 [45]. Συγκεκριμένα δημιουργήσαμε traces για τουλάχιστον 10 διαφορετικές εισόδους για κάθε benchmark [8]. Οι διαφορετικές είσοδοι αποτελούν πολύ σημαντικό κομμάτι των δεδομένων, καθώς επιτρέπουν στο νευρωνικό δίκτυο να εντοπίσει συσχετισμούς ανάμεσα στις διακλαδώσεις, οι οποίες εξαρτώνται κυρίως από τη λογική ροή ενός προγράμματος και όχι από την είσοδο του. Αυτό ενισχύει την απόδοση του νευρωνικού καθώς εξασφαλίζει την απόδοση του σε μελλοντικές εκτελέσεις μιας εφαρμογής, με διαφορετική είσοδο.

4.1.1.2 Εργαλείο SimPoint

Επειδή η κάθε προσομοίωση αποτελείται από πολλές δισεκατομμύρια εντολές χρησιμοποιήσαμε το εργαλείο SimPoint [46], για να εντοπίσουμε τις πιο αντιπροσωπευτικές περιόδους της κάθε εκτέλεσης. Το εργαλείο αυτό ανιχνεύει τις διαφορετικές φάσεις μια εκτέλεσης με σκοπό να εστιάσουμε σε συγκεκριμένα μεμονωμένα τμήματά της, για να μπορέσουμε πιο αποδοτικά να επιτύχουμε το ίδιο αποτέλεσμα, με το εάν εξετάζαμε όλες τις εντολές του προγράμματος. Το εργαλείο δέχεται σαν είσοδο ένα 'παράθυρο' από το οποίο περνά ολόκληρη την εκτέλεση του προγράμματος και σαν έξοδο μας ενημερώνει σε ποια σημεία της εκτέλεσης αντιστοιχεί στο αντιπροσωπευτικό κομμάτι του προγράμματος. Με αυτόν τον τρόπο μπορούμε να εντοπίσουμε Hard-to-Predict branches, καθώς και να συλλέξουμε δεδομένα, εξετάζοντας μόνο μια μικρή περιοχή της εκτέλεσης, πολύ πιο αποδοτικά.

4.2 Δύσκολες-να-προβλεφθούν διακλαδώσεις - (Hard-to-Predict Branches)

Η πρώτη απόπειρα εντοπισμού και αναγνώρισης των Hard-to-Predict branches (ή πιο σύντομα H2P branches) έγινε από τους καθηγητές Stephen J. Tarsa και Chit-Kwan Lin. Βασιστήκαμε στη δημοσίευσή τους *Branch Prediction is not a solved problem* [5], και προσπαθήσαμε να εντοπίσουμε Hard-to-Predict branches με τα ίδια κριτήρια. Τα κριτήρια αυτά χρησιμοποιούνται ώστε να εντοπίσουμε στατικά branches τα οποία από τη μία δημιουργούν πολλές λάθος προβλέψεις ενώ παράλληλα εμφανίζονται αρκετές φορές, ώστε να μπορούν να

χρησιμοποιηθούν για εκπαίδευση νευρωνικών δικτύων. Από τις μετρήσεις μας παρατηρήσαμε ότι υπάρχουν κάποια σχετικά λίγα branches τα οποία αν βελτιωθούν προσφέρουν σημαντική απόδοση στην εκτέλεση του εκάστοτε προγράμματος.

4.2.1 Αναγνώριση και εντοπισμός των Hard-to-Predict branches

Όπως και στη σχετική δημοσίευση [5], για κάθε benchmark, ανά 30 εκατομμύρια εντολές, κρατήσαμε στατιστικά για όλες τις διακλαδώσεις και χαρακτηρίσαμε ως Δύσκολες-Να-Προβλεφθούν αυτές που ικανοποιούν και τα τρία ακόλουθα κριτήρια:

- Εμφανίζονται τουλάχιστον 15.000 φορές
- Έχουν τουλάχιστον 1000 λάθος προβλέψεις (misspredicts)
- Έχουν λιγότερο απο 99% επιτυχία (accuracy) κάτω από τον Tage Predictor

Τα κριτήρια επιλογής γίνονται με την εξής λογική: Τα branches αυτά, αφενός προκαλούν πολλά misspredicts και άρα υπάρχει περιθώριο βελτίωσης ως προς την σωστή πρόβλεψη τους και αφετέρου παρουσιάζονται αρκετές φορές, ώστε να μπορεί να μπορούν να χρησιμοποιηθούν για εκπαίδευση νευρωνικών μοντέλων.

4.2.2 Υλοποίηση στον προσομοιωτή

Επεκτείναμε τις λειτουργίες του προσομοιωτή, δημιουργώντας την κλάση BranchStatistcs με την οποία κρατήσαμε στατιστικά για όλα τα branches του trace. Χρησιμοποιήσαμε τον πηγαίο κώδικα από το σχετικό repository [;], ενώ όλες οι αλλαγές που αναφέρουμε στη συνέχεια υπάρχουν αντίστοιχα στο [47]. Όπως είπαμε στο 4.2.1, τα κριτήρια χαρακτηρισμού μιας διακλάδωσης ως Hard-to-Predict είναι τρία. Το πρώτο είναι ο αριθμός εμφάνισης της εντολής, το δεύτερο ο αριθμός των λάθος προβλέψεων και το τρίτο η ευστοχία που έχει κάποιος Predictor σε αυτό. Μια επιπλέον παράμετρος είναι αν αυτά τα κριτήρια επιθυμούμε να πληρούνται σε ολόκληρο το εκτελέσιμο, ή ανά κάποιο αριθμό εντολών. Η κλάση BranchStatistics που δημιουργήσαμε, περιέχει πεδία για όλες αυτές τις παραμέτρους και αναλόγως κατηγοριοποιεί ένα branch ως Hard-to-Predict ή οχι. Κατά την εκκίνηση μιας προσομοίωσης, με σκοπό την καταγραφή Hard-to-Predict branches, χρειάζεται να περάσουμε αυτές τις παραμέτρους ως ορίσματα προκειμένου να αρχικοποιηθούν. Οι παράμετροι που απαιτούνται είναι οι εξής:

- `-occurrences`
  Ελάχιστος αριθμός εμφανίσεων της διακλάδωσης
- `-misspredictions`
  Ελάχιστος αριθμός λάθος προβλέψεων της διακλάδωσης
- `-accuracy`
  Μέγιστη ευστοχία του predictor
- `-reset_window`
  Αριθμός εντολών ανά τον οποίο γίνεται επαναφορά των μετρητών (0 για άπειρο αριθμό)

Αφού ορίσαμε τις παραμέτρους, όπως ορίσαμε τα Hard-to-Predict στην παράγραφο 4.2.1, συλλέξαμε δεδομένα από 4 διαφορετικές εφαρμογές (benchmarks) [8], σε διαφορετικά Simpoints [46] και διαφορετικές εισόδους [45]. Στον πίνακα 4.1 βλέπουμε τα αποτελέσματα· στην πρώτη στήλη βλέπουμε το όνομα του benchmark, στη δεύτερη τα συνολικά διαφορετικά Hard-to-Predict branches που εντοπίστηκαν για την εφαρμογή σε όλα τα διαφορετικά simpoints και εισόδους, στη τρίτη στήλη βλέπουμε το πλήθος των εισόδων, ενώ στη τελευταία Hard-to-Predict branches τα οποία εντοπίστηκαν κοινά στο 40% των διαθέσιμων εισόδων.

Πίνακας 4.1: *Hard-to-Predict Branches ανά benchmark.*

Benchmark	**Συνολικά H2P**	**Πλήθος εισόδων**	**Κοινά H2P σε ποσοστό 40% των διαθέσιμων εισόδων**
531.deepsjeng	91	9	32
541.leela	85	9	42
557.xz	140	8	17
505.mcf	32	8	11

Αυτό που μας ενδιαφέρει να εστιάσουμε είναι η τελευταία στήλη, τα κοινά δηλαδή, Hard-to-Predict branches. Καθώς τα branches αυτά είναι κοινά σε διαφορετικές εισόδους, μπορούμε να συμπεράνουμε ότι είναι ανεξάρτητα των εισόδων της εφαρμογής και εμφανίζονται σε κάθε εκτέλεση της, ανεξάρτητα αυτών.

4.3 Μοντέλα νευρωνικών δικτύων

Μετά τον εντοπισμό των Hard-to-Predict branches που περιγράψαμε παραπάνω, το επόμενο βήμα είναι ο σχεδιασμός και η εκπαίδευση νευρωνικών δικτύων, εκ των οποίων κάθε ένα θα είναι υπεύθυνο για την πρόβλεψη ενός συγκεκριμένου Hard-to-Predict branch.

4.3.1 Προσέγγιση προβλήματος

Όταν προσπαθούμε να επιλύσουμε ένα πρόβλημα μηχανικής μάθησης υπάρχουν τρεις βασικοί τρόποι προσέγγισης. Αυτοί είναι

1. Κατηγοριοποίηση (Classification)
2. Λογιστική παλινδρόμηση (Logistic Regression)
3. Ομαδοποίηση (Clustering)

Στο πρόβλημα της πρόβλεψης διακλαδώσεων μπορούμε να δοκιμάσουμε δύο διαφορετικές προσεγγίσεις. Η μία είναι αυτή της κατηγοριοποίησης, όπου το νευρωνικό μοντέλο προσπαθεί να προσδιορίσει εάν ένα ιστορικό διακλαδώσεων ανήκει στην κατηγορία των ‘‘Μη αληθών‘‘ ή στην κατηγορία των ‘‘Αληθών‘‘. Η άλλη προσέγγιση είναι αυτή της λογιστικής παλινδρόμησης όπου το μοντέλο προβλέπει μια τιμή ανάμεσα σε δύο όρια που θέτουμε εμείς. Τα δύο όρια είναι οι δύο πιθανές απαντήσεις στην πρόβλεψη της διακλάδωσης και έτσι εάν το νευρωνικό βρίσκεται πιο κοντά στο εκάστοτε όριο, θεωρούμε ότι κάνει και την αντίστοιχη πρόβλεψη. Στη συγκεκριμένη διπλωματική χρησιμοποιήσαμε τη μέθοδο της Λογιστικής Παλινδρόμησης, όπως και στη σχετική βιβλιογραφία [2] [1].

4.3.2 Είδη νευρωνικών δικτύων

Υπάρχει τεράστια ποικιλία όσον αφορά τα μοντέλα νευρωνικών δικτύων και η σωστή επιλογή ενός μοντέλου για κάποιο πρόβλημα, αποτελεί πολύ σημαντικό βήμα για την σωστή επίλυση του προβλήματος. Μερικά από τα διαθέσιμα νευρωνικά μοντέλα είναι:

- Συνελικτικά νευρωνικά δίκτυά (Convolutional Neural Networks)
- Αναδρομικά Νευρωνικά Δίκτυα (Recurrent Neural Networks)
- Long Short-Term Memory αναδρομικά νευρωνικά δίκτυα
- Transformer μοντέλο

Γενικά δεν υπάρχει κάποιος κανόνας ή μεθοδολογία για την επιλογή ενός συγκεκριμένου μοντέλου. Συνήθως η καλύτερη προσέγγιση είναι η επιλογή ενός μοντέλου που έχει αποδώσει σε παρόμοια προβλήματα στο παρελθόν. Ωστόσο γενικά τα συνελικτικά μοντέλα, φαίνονται να αποδίδουν περισσότερο σε 'χωρικά' προβλήματα. Τέτοια προβλήματα μπορεί να είναι η αναγνώριση αντικειμένων που υπάρχουν μέσα σε μια εικόνα. Από την άλλη τα αναδρομικά νευρωνικά δίκτυα, φαίνεται να αποδίδουν καλύτερα σε ¨χρονικά' προβλήματα. Τέτοια προβλήματα αποτελούν η μετάφραση κειμένου ή παραγωγή κειμένου, όπου κάθε δεδομένο εξαρτάται από το προηγούμενο.

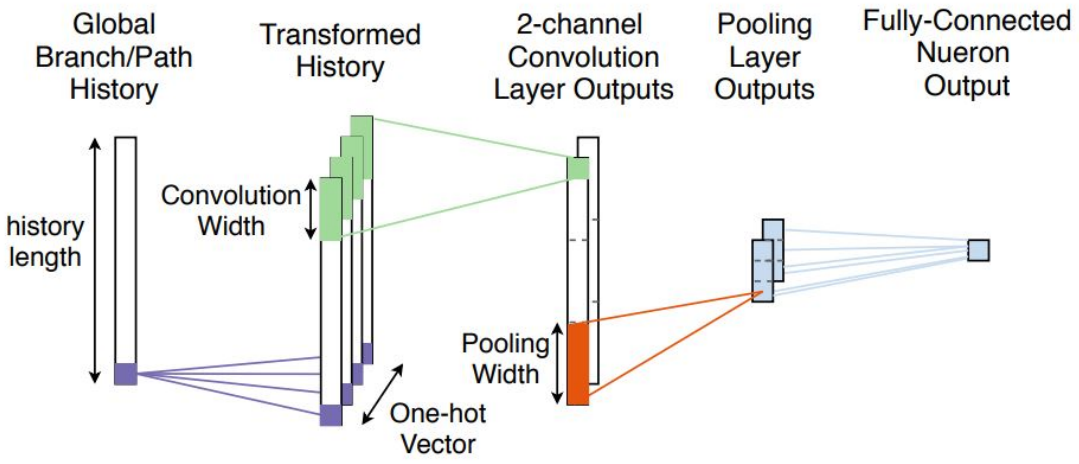
4.3.3 Συνελικτικά δίκτυα

Σε μία πρώτη ματιά, η πρόβλεψη διακλαδώσεων φαίνεται να είναι ένα 'χρονικό' πρόβλημα, όπου η πρόβλεψη ενός branch εξαρτάται από την ακολουθία των προηγούμενων. Την είσοδο σε μία τέτοια περίπτωση θα μπορούσε να αποτελεί μία ακολουθία ιστορικού, όπως το διάβασμα μιας πρότασης. Ωστόσο μπορεί να αντιμετωπιστεί και σαν 'χωρικό' πρόβλημα, αφού γίνει η κατάλληλη κωδικοποίηση των δεδομένων, όπου σε αυτή την περίπτωση το νευρωνικό δίκτυο μπορεί να δέχεται σαν είσοδο έναν πίνακα, όπου έχει κωδικοποιηθεί όλο το ιστορικό της διακλάδωσης που επιχειρεί να προβλέψει. Με αυτό τον τρόπο, ένα συνελικτικό νευρωνικό δίκτυο είναι ικανό να παρατηρήσει συσχετίσεις μεταξύ των branches και εν τέλει να κάνει πιο αποτελεσματική πρόβλεψη. Επίσης, ένας ακόμα λόγος επιλογής των συγκεκριμένων μοντέλων είναι ότι είναι πιο εύκολο να αναπαρασταθούν με τέτοιο τρόπο στο υλικό, ώστε να μπορούν να ενσωματωθούν σε μια κεντρική μονάδα επεξεργασίας, τόσο από άποψη χώρου όσο και από άποψη καθυστέρησης (latency) στον υπολογισμό μιας πρόβλεψης. Τα συνελικτικά μοντέλα μπορούν σχετικά πιο εύκολα να ανακατασκευαστούν χρησιμοποιώντας λογικές πύλες και κυκλώματα (σε αντίθεση με τα αναδρομικά μοντέλα), πράγμα ουσιαστικό στη περίπτωση μας, όπου τελικός σκοπός αποτελεί η δημιουργία μιας μονάδας πάνω στον επεξεργαστή.

4.3.3.1 Tarsa Predictor

Ένας απλός predictor που χρησιμοποιεί συνελικτικά δίκτυα παρουσιάστηκε από τον Stephen Tarsa [2]. Το σχετικά απλό αυτό νευρωνικό δέχεται σαν είσοδο 1-hot κωδικοποίηση (την οποία αναλύουμε στο 4.4.3.1) ενός ιστορικού ζευγαριών (branch Program Counter - Αποτέλεσμα branch), μήκους 200. Στη συνέχεια χρησιμοποιεί ένα Convolutional επίπεδο για να εντοπίσει

συσχετίσεις μεταξύ των branches. Μία παρόμοια απλή αρχιτεκτονική ενός μοντέλου που βασίζεται στην ίδεα του Tarsa μπούμε να δούμε στο διάγραμμα 4.1.



Σχήμα 4.1: *Ένας απλός CNN predictor.*

Όπως σε μία ανάλυση εικόνας ένα συνελικτικό νευρωνικό δίκτυο εντοπίζει σχήματα στις εικόνες, με παρόμοιο τρόπο εδώ μπορεί να εντοπίζει σε ένα ιστορικό, τα σημαντικά branches που μας ενδιαφέρουν για να γίνει μια πρόβλεψη. Αφού γίνει το pattern matching το αποτέλεσμα τροφοδοτείται σε ένα γραμμικό επίπεδο (linear layer) το οποίο επιλέγει την τελική πρόβλεψη.

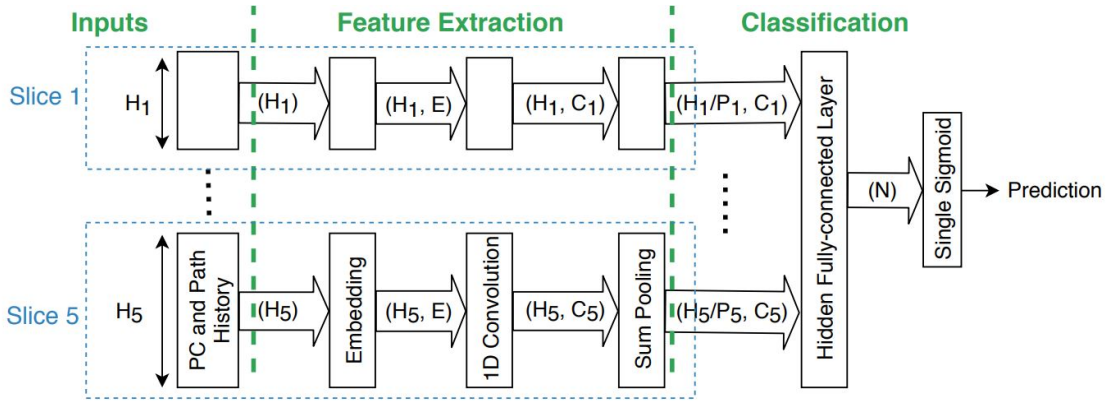
Στη διπλωματική αυτή χρησιμοποιήσαμε τον Predictor του καθηγητή Tarsa, όπως περιλαμβάνεται στο repository του BranchNet [13], ωστόσο στο παράρτημα Α΄, στην ενότητα Α΄.1 αλγόριθμος Α΄.1, βλέπουμε πως θα μπορούσε να υλοποιηθεί [2].

4.3.3.2 BranchNet Predictor

Ένας πιο εξελιγμένος προβλεπτής, που βασίστηκε στην ιδέα του Tarsa, είναι αυτός που αφορά την υλοποίηση του BranchNet [1]. Η βασική διαφορά σε σχέση με την δουλειά του πρώτου έγκειται στο γεγονός ότι μεγαλύτερα ιστορικά δημιουργούν προβλήματα στο τελευταίο επίπεδο (το γραμμικό) καθώς δημιουργείται μεγάλος όγκος δεδομένων. Το BranchNet για να αξιοποιήσει μεγαλύτερα ιστορικά και να επιλύσει αυτό το πρόβλημα κάνει χρήση sum pooling στρωμάτων, τα οποία δρουν δειγματοληπτικά [48], δίνοντας έτσι την δυνατότητα επεξεργασίας μεγαλύτερων ιστορικών σε συνδυασμό με περισσότερα PC branch bits από το τελευταίο γραμμικό στρώμα. Η αρχιτεκτονική του BranchNet είναι εμπνευσμένη από παραδοσιακούς predictors και χρησιμοποιεί γεωμετρικά αυξανόμενα ιστορικά σαν είσοδο. Όπως βλέπουμε στο διάγραμμα 4.2, για κάθε ιστορικό (slice) αρχικά έχουμε κωδικοποίηση με χρήση embeddings, τα οποία θα αναλύσουμε παρακάτω στο 4.4.3.1, στη συνέχεια ένα συνελικτικό δίκτυο το οποίο εντοπίζει μοτίβα, έπειτα γίνεται δειγματοληψία (sum pooling) και τελικά στα επόμενα στρώματα περνάει μόνο η πληροφορία των σχετικών branches. Έτσι στο τελευταίο στρώμα, όπως και στον Tarsa, το γραμμικό, φτάνουν μόνο οι πληροφορίες για τα σχετικά branches, παρά το μεγάλο μήκος ιστορικού που χρησιμοποιείται. Η αρχιτεκτονική αυτή χρησιμοποιεί 582 θέσεις ιστορικού (σε αντίθεση με 200), ενώ κωδικοποιεί τα 11 λιγότερα σημαντικά bits

του program counter της εντολής διακλάδωσης (branch).

Στη διπλωματική αυτή χρησιμοποιήσαμε το BranchNet, έτοιμο, όπως δίνεται στο σχετικό repository [13].



Σχήμα 4.2: *Διάγραμα του BranchNet predictor για κάποιο Hard-to-Predict branch.*

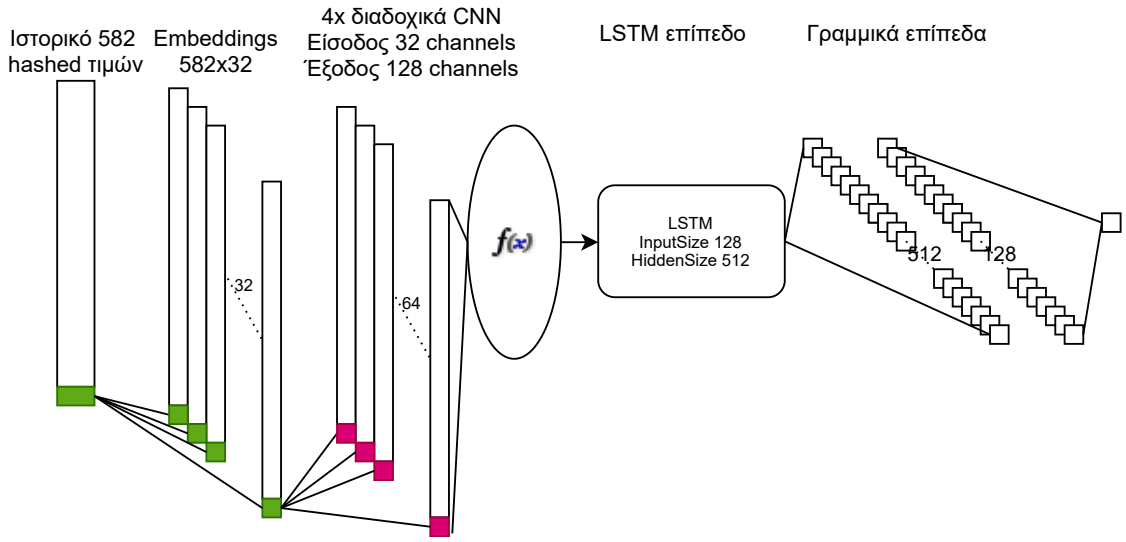
4.3.4 Convolutional - LSTM νευρωνικό δίκτυο

Σε αυτή την διπλωματική εργασία σχεδιάζουμε και προτείνουμε ένα συνδυαστικό μοντέλο που περιέχει convolutional στρώματα για εξαγωγή χαρακτηριστικών (feature extraction) και στη συνέχεια τροφοδότηση των αποτελεσμάτων σε ένα Μακράς-Βραχέως προσωρινής μνήμης (Long Short-Term Memory - LSTM) νευρωνικό.

Στο σχήμα 4.3 βλέπουμε τη διαδικασία πρόβλεψης με το συγκεκριμένο μοντέλο. Όπως και στο BranchNet χρησιμοποιούμε 11 bits της διεύθυνσης του branch και μήκος ιστορικού 582, ενώ αφού κωδικοποιήσουμε τα ζευγάρια (branch PC bits - Αποτέλεσμα branch) κάνουμε χρήση embeddings σε διάσταση (1,32). Η σημασία τόσο της κωδικοποίησης των δεδομένων, όσο και η χρήση των embeddings θα αναλυθεί περαιτέρω στην ενότητα 4.4.3. Εν συντομία το επίπεδο embedding ανάγει κάθε κωδικοποιημένη τιμή σε έναν πίνακα διαστάσεων (1,32) στο οποίο τα συνελικτικά στρώματα μπορούν να εκτελέσουν πράξεις. Στην συνέχεια, λοιπόν, έχουμε 4 διαδοχικά συνελικτικά στρώματα τα οποία εξάγουν χαρακτηριστικά και αναδεικνύουν ποια branches είναι σημαντικά στο ιστορικό (feature extraction), ώστε το LSTM να μπορέσει να επεξεργαστεί την ακολουθία αποτελεσματικότερα. Η ιδέα πίσω από αυτό βασίζεται στο BranchNet με μόνη διαφορά ότι χρησιμοποιούμε 4 διαδοχικά συνελικτικά δίκτυα (το BranchNet 1) με σκοπό να επιτύχουμε καλύτερη εξαγωγή χαρακτηριστικών, ανάγωντας τις 32 διαστάσεις σε 128. Το αποτέλεσμα από το αρχικό στάδιο του ιστορικού κωδικοποιημένων τιμών διαστάσεων (1,582), είναι ένας πίνακας διαστάσεων (128, 582). Κατόπιν το LSTM, διαστάσεων εισόδου 128 και κρυφών κελιών 512, δέχεται σειριακά κάθε γραμμή του πίνακα, διαβάζοντας την ακολουθία και εντοπίζει συσχετίσεις μεταξύ των διαφορετικών κωδικοποιημένων θέσεων του ιστορικού. Στο τελικό στάδιο χρησιμοποιούμε τα 512 κρυφά κελιά (hidden states) του LSTM, σε γραμμικά επίπεδα (linear layers) για να κάνουμε την τελική πρόβλεψη. Σημειώνουμε ότι ενδιάμεσα των επιπέδων υπάρχουν συναρτήσεις ενεργοποίησης (activation functions) για να αυξήσουμε την ευστοχία του μοντέλου και να επιταχύνουμε την διαδικασία εκπαίδευσης.

Η πρόταση μας προσπαθεί να συνδυάσει την ιδέα του BranchNet, η οποία χρησιμοποιεί συνελικτικά δίκτυα για να εντοπίσει branches που σχετίζονται με την ζητούμενη πρόβλεψη, σε ένα μεγάλο ιστορικό, αλλά ταυτόχρονα προσπαθεί να χρησιμοποιήσει τα LSTM· τα οποία αξιοποιούνται σε ένα μεγάλο εύρος ακολουθιακών προβλημάτων [49], ακριβώς όπως η πρόβλεψη εντολών διακλάδωσης.

Στο παράρτημα Α′ παρουσιάζεται ο κώδικας σε PyTorch, ενότητα Α′.2, αλγόριθμος Α′.2, ο οποίος μπορεί να βρεθεί και στο repository [43].



Σχήμα 4.3: *Διάγραμα του LSTM predictor.*

4.4 Δημιουργία των δεδομένων (Dataset)

4.4.1 Μεθοδολογία συλλογής δεδομένων

Αρχικά έγινε συλλογή του ιστορικού ολόκληρης της προσομοίωσης. Όπως και στη σχετική δημοσίευση BranchNet [1], καταγράψαμε ολόκληρο το ιστορικό καθενός trace (ζευγάρια από Program Counter και Αποτέλεσμα Branch), καθώς και όλες τις θέσεις των Hard-to-Predict στην εν λόγο καταγραφή.

Προκειμένου να εκπαιδευτεί σωστά ένα νευρωνικό μοντέλο, δίνεται μεγάλη σημασία στα δεδομένα που θα χρησιμοποιηθούν. Κατά τη συλλογή των δεδομένων έχουμε σαν πρώτη επιλογή να κρατήσουμε ιστορικό εντολών, για κάθε branch που συναντάμε. Προφανώς η προσέγγιση αυτή, αφενός δημιουργεί τεράστιο όγκο δεδομένων και αφετέρου υπάρχει η πιθανότητα να παρουσιάζει στο νευρωνικό περιττή πληροφορία. Παρατηρούμε ότι δύο συνεχόμενες εκτελέσεις μιας διακλάδωσης, η οποίες θα έχουν κατά συνέπεια σχεδόν ακριβώς το ίδιο ιστορικό εντολών, δεν προσφέρουν πολύτιμη πληροφορία στο νευρωνικό μοντέλο, αντιθέτως, δημιουργούν θόρυβο και δυσκολεύουν την εκπαίδευση του νευρωνικού δικτύου. Για το λόγο αυτό αντί να χρησιμοποιήσουμε όλες τις ακολουθίες ιστορικών, επιλέγουμε στην τύχη μόλις λίγα χιλιάδες δείγματα, σε αντίθεση με τα εκατομμύρια που υπάρχουν σε όλη την εκτέλεση (τα οποία μπορεί να είναι και επαναλαμβανόμενα). Το νευρωνικό με αυτό τον τρόπο συγκλίνει καλύτερα και αναγνωρίζει μοτίβα που το βοηθούν να προβλέπει σωστά αυτά τα branch, αφού υπάρχει

περισσότερη πληροφορία από δύο ιστορικά εντολών, τα οποία απέχουν χρονικά μεταξύ τους και άρα διαφέρουν αρκετά. Η εκπαίδευση των νευρωνικών μοντέλων γίνεται χρησιμοποιώντας το μικρό σύνολο δειγμάτων ιστορικών και ο έλεγχος της απόδοσης τους γίνεται κάνοντας προβλέψεις για όλα τα ιστορικά που παρουσιάζονται στην εκτέλεση.

4.4.2 Υλοποίηση συλλογής δεδομένων (Dataset)

Επεκτείναμε τον προσομοιωτή και δημιουργήσαμε την κλάση Branch_History. Αν θέλουμε να γίνει καταγραφή του ιστορικού, σε μία προσομοίωση, χρειάζεται να περάσουμε δύο παραμέτρους στο εκτελέσιμο. Πρώτα χρησιμοποιούμε το flag `-collect_all_branches` και έπειτα περνάμε μέσω του ορίσματος `-collect_all_branches_file` το όνομα του αρχείου όπου θέλουμε να συλλέξουμε δεδομένα. Στο αρχείο υπάρχουν όλες οι διακλαδώσεις που εμφανίζονται στην εκτέλεση ακολουθούμενες από ένα κενό και τον αριθμό 0 για τις διακλαδώσεις που το αποτέλεσμα είναι Not-Taken ή τον αριθμό 1 αντίστοιχα για Taken.

4.4.2.1 Δημιουργία H5PY datasets

Όπως και στο BranchNet [1] χρησιμοποιήσαμε την βιβλιοθήκη H5PY, η οποία μας επιτρέπει την αποθήκευση και διαχείριση τεράστιων dataset. Αποθηκεύουμε ολόκληρο το ιστορικό, καθώς και κλειδιά για κάθε H2P, τα οποία μας δείχνουν σε ποιες θέσεις του ιστορικού μπορούμε να βρούμε το εκάστοτε H2P. Η συγκεκριμένη μέθοδος προσφέρει από τη μία ευελιξία στα δεδομένα, καθώς κάθε φορά μπορούμε να αντλήσουμε διαφορετικό μέγεθος ιστορικού, ενώ ταυτόχρονα μπορούμε κάθε φορά να επιλέγουμε τυχαία ιστορικά για κάποιο branch από κάποιο trace.

Σημαντικό είναι πως στα αρχεία H5py δεν αποθηκεύσαμε ζευγάρια από *(Program_Counter , Branch_Outcome)* αλλά κωδικοποιημένη πληροφορία όπως αναλύουμε παρακάτω.

4.4.3 Κωδικοποίηση των δεδομένων - History hashing

Η κωδικοποίηση των δεδομένων, αποτελεί την μετατροπή της πληροφορίας που συλλέξαμε στο 4.4.1, σε κάτι που μπορεί να ερμηνεύσει και να αναγνωρίσει καλύτερα ένα νευρωνικό δίκτυο. Όπως είπαμε, έχουμε εξάγει N τούπλες της μορφής *(Program_Counter , Branch_Outcome)*, οι οποίες όμως πρέπει να κωδικοποιηθούν. Εδώ αξίζει να παρατηρήσουμε ότι αφενός ο αριθμός εντολής είναι ένας πολύ μεγάλος αριθμός και αφετέρου ότι η σημαντική πληροφορία υπάρχει στα χαμηλότερης τάξης ψηφία του, αφού πρόκειται για διακλαδώσεις που βρίσκονται κοντά στη διακλάδωση που εξετάζουμε για πρόβλεψη. Συνεπώς ένα πρώτο βήμα είναι να κρατήσουμε μόνο τα ελάχιστα σημαντικά ψηφία του αριθμού της εντολής (Least Significant bits - LSB). Έπειτα εάν εκεί συνδέσουμε σειριακά το Branch Outcome καταλήγουμε με έναν αριθμό που περιέχει όλη την απαραίτητη πληροφορία.

Παρακάτω στον πίνακα 4.2 βλέπουμε μερικά παραδείγματα της εξής διαδικασίας.

4.4.3.1 Κωδικοποίηση των δεδομένων σε πίνακα

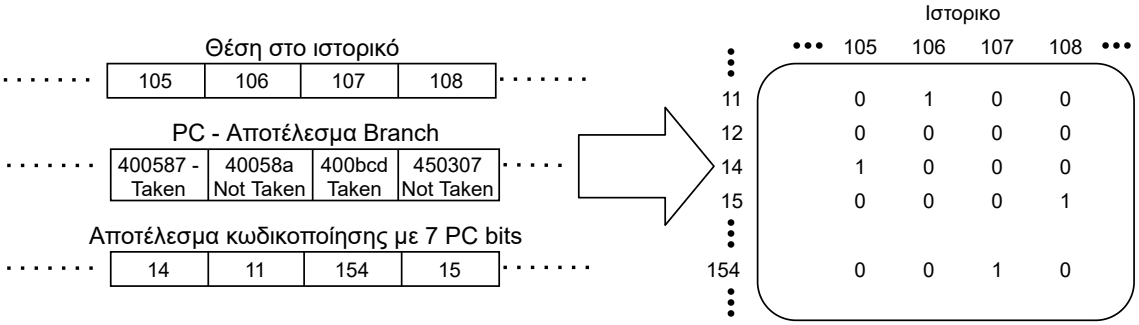
Στη προηγούμενη παράγραφο (4.4.3) δείξαμε πως μπορούμε να κωδικοποιήσουμε τον αριθμό εντολής, σε ένα μικρότερο, πιο ‘‘διαχειρίσιμο’’ αριθμό. Ωστόσο στην περίπτωση των

Πίνακας 4.2: *Παραδείγματα History Hashing. Θεωρούμε ότι κρατάμε τα 7 λιγότερο σημαντικά bits του branch Program Counter.*

Εντολές	**10 Λιγότερα σημαντικά ψηφία PC**	**Αποτέλεσμα για Not-Taken**	**Αποτέλεσμα για Taken**
400587: jle 400599	..0110000111	14	15
40058a: ja 400599	..0110001010	10	11
400bcd: je 400599	..1111001101	154	155
450307: jle 450a03	..0000000111	14	15

νευρωνικών μοντέλων χρειάζεται να έχουμε μια διαφορετική προσέγγιση καθώς η είσοδος πρέπει να είναι πίνακες αριθμών.

Η μετατροπή της ακολουθίας των N εντολών σε πίνακα, μπορεί να γίνει χρησιμοποιώντας 1-hot πίνακες. Οι πίνακες αυτοί περιέχουν σε κάθε γραμμή μηδενικά, εκτός από μία μόνο θέση όπου υπάρχει άσσος. Στην περίπτωση μας, σε έναν 1-hot πίνακα η κάθε στήλη αντιστοιχεί σε έναν κωδικοποιημένο αριθμό της ακολουθίας του ιστορικού, ενώ ο άσσος βρίσκεται στην κατάλληλη γραμμή από το 0 έως το $2^p + 1$, ανάλογα με την κωδικοποίηση που προκύπτει, όπου p ο αριθμός των Least Significant Bits που χρησιμοποιούμε από το Program Counter του branch. Στην εικόνα 4.4 παρουσιάζεται η αναπαράσταση του ιστορικού του πίνακα 4.2 με 1-hot κωδικοποίηση.



Σχήμα 4.4: *Παράδειγμα κωδικοποίησης ενός ιστορικού σε 1-hot πίνακα.*

Ένας ακόμα τρόπος κωδικοποίησης είναι με τη χρήση **Embeddings**. Η διαδικασία 1-hot που περιγράψαμε παραπάνω, είναι στην ουσία ένα πολύ απλό Embedding. Στην διαδικασία αυτή αντιστοιχούμε *λέξεις*, ή στην δική μας περίπτωση Hashed αριθμούς, σε αριθμητικούς πίνακες. Ιδανικά το επίπεδο Embedding, εκπαιδεύεται ταυτόχρονα με το νευρωνικό και μαθαίνει να αποδίδει παρόμοιες τιμές σε λέξεις, ή εδώ hashes, που παρουσιάζουν ομοιότητες.

Στο πρόβλημα μας η χρήση των embeddings μας επιτρέπει να αυξήσουμε τον αριθμό ελαχίστων σημαντικών bits του program counter ενός branch που χρησιμοποιούμε. Για παράδειγμα η αναπαράσταση με 1-hot μιας κωδικοποίησης που χρησιμοποιεί 11 LSB του branch program counter, μαζί με το αποτέλεσμα του branch απαιτεί πίνακα με $2^{12} = 4096$ γραμμές. Αντίθετα με embeddings, μπορούμε να περιορίσουμε τον αριθμό αυτό σε πολύ μικρότερο (π.χ. 32), αλλάζοντας τις τιμές σε κάθε κελί· όχι χρησιμοποιώντας μόνο 0 και 1. Η ιδέα χρησιμοποιήθηκε από τους Milad Hashemi, Kevin Swersky, Jamie A. Smith, Grant Ayers,

Heiner Litz, Jichuan Chang, Christos Kozyrakis, Parthasarathy Ranganathan [3] για την αναπαράσταση Program Counters και διευθύνσεις μνήμης για κάποιο μοντέλο, με σκοπό την προανάκτηση δεδομένων (data prefetching).

4.5 Εκπαίδευση νευρωνικών δικτύων

Εκπαιδεύουμε ένα νευρωνικό μοντέλο για **κάθε ένα** Branch. Η εκπαίδευση γίνεται χρησιμοποιώντας τυχαία δείγματα από τα H5PY datasets, τα οποία περιέχουν ιστορικά από πολλαπλά Benchmark inputs και Simpoints . Για την εκπαίδευση χρησιμοποιούνται Traces τα οποία δημιουργήθηκαν αποκλειστικά από τα Alberta Workloads [45]. Αφού εκπαιδευτούν τα νευρωνικά η επαλήθευση της απόδοσης τους γίνεται με τα reference inputs που περιέχονται στη σουίτα Spec2017.

Για την εκπαίδευση βασιστήκαμε στον κώδικα του Branchnet [13]. Η εκπαιδεύση γίνεται σε 3 φάσεις. Ξεκινήσαμε με learning rate 0.001 και σε κάθε φάση μειώνουμε το learning rate με σκοπό να επιτύχουμε μεγαλύτερη σύγκλιση και άρα καλύτερη απόδοση του νευρωνικού. Optimizer χρησιμοποιήθηκε ο Adam και για τον υπολογισμό του loss ο αλγόριθμος MSE. Όλος ο πηγαίος κώδικας υπάρχει διαθέσιμος στο [43].

Κεφάλαιο 5

Πειραματική Αξιολόγηση

Η μέτρηση απόδοσης των νευρωνικών δικτύων αποτέλεσε ένα σημαντικό κομμάτι αυτής της διπλωματικής. Στην υπάρχουσα βιβλιογραφία [1] το προτεινόμενο μοντέλο (BranchNet) αξιολογήθηκε μόνο ως προς το MPKI, δηλαδή τις λάθος προβλέψεις ανά 1000 εντολές. Εμείς αξιολογούμε τα νευρωνικά μοντέλα της διπλωματικής, τόσο ως προς τη μετρική του MPKI, όσο όμως και ως προς την συνολική απόδοση που προσφέρουν στις εφαρμογές μέσω προσομοίωσης σε μια αρχιτεκτονική (αυτή του ChampSim [7]). Στο συγκεκριμένο κεφάλαιο παρουσιάζεται η μεθοδολογία που ακολουθήσαμε για να μπορέσουμε να υπολογίσουμε την μετρική Εντολές-ανά-Κύκλο (Instructions Per Cycle - IPC) καθώς και τα αποτελέσματα που βρήκαμε. Επίσης, υπολογίσαμε την αύξηση του IPC σε θεωρητικά μελλοντικές αρχιτεκτονικές επεξεργαστών, κάνοντας scale up τα διάφορα μέρη του επεξεργαστή (pipeline capacity κτλ.).

5.1 Μεθοδολογία ελέγχου ακρίβειας

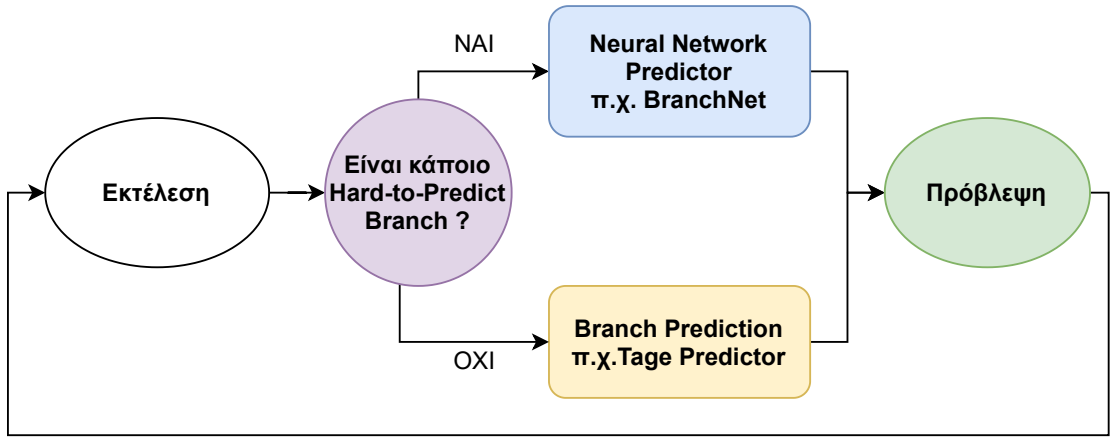
Ένας απλός τρόπος για να ελέγξουμε την απόδοση του νευρωνικού δικτύου είναι απλά να ελέγξουμε τις προβλέψεις του για όλα τα (Hard-to-Predict) branches που παρουσιάζονται σε κάποιο trace - simpoint. Η μέθοδος αυτή μπορεί να μας παρέχει μια τιμή ευστοχίας (accuracy) του νευρωνικού μοντέλου η οποία μπορεί να συγκριθεί εύκολα με τις αντίστοιχες τιμές άλλων μοντέλων ή ήδη υπαρχόντων προβλεπτών, για κάποιο branch. Ωστόσο δεν μας δίνει καμία πληροφορία για την επίπτωση που έχει ο προβλεπτής συνολικά στη επίδοση εφαρμογών πάνω σε μία αρχιτεκτονική. Ειδικά στο συγκεκριμένο τομέα του Branch Prediction γνωρίζουμε ότι ακόμα και μια βελτίωση *1%* του προβλεπτή μπορεί να επιφέρει μεγάλη επιτάχυνση του επεξεργαστή.

Επομένως γίνεται κατανοητό ότι για να μπορέσουμε να έχουμε μια καλύτερη εικόνα για την απόδοση των νευρωνικών μοντέλων, χρειάζεται να τα ενσωματώσουμε σε κάποια μονάδα πρόβλεψης εντολών διακλάδωσης (Branch Prediction Unit - BPU) ώστε μέσω προσομοίωσης να ελέγξουμε την επιτάχυνση που προκύπτει στον επεξεργαστή.

5.1.1 Ενσωμάτωση νευρωνικών στον προσομοιωτή ChampSim

Η χρήση του προσομοιωτή αρχιτεκτονικής ChampSim σε συνδυασμό ταυτόχρονων προβλέψεων νευρωνικών δικτύων, ως Branch Prediction Unit (BPU) της αρχιτεκτονικής, αποτέλεσε ένα σημαντικό τεχνικό ζήτημα αυτής της διπλωματικής.

Όπως έχουμε ήδη αναφέρει, τα νευρωνικά δίκτυα προσφέρουν προβλέψεις για κάποιο συγκεκριμένο branch και όχι για όλα. Η μονάδα πρόβλεψης που προτείνουμε περιέχει μοντέλα νευρωνικών δικτύων για την πρόβλεψη των συγκεκριμένων branches και για τα υπόλοιπα χρησιμοποιείται κάποιος κλασσικός branch predictor. Παρακάτω γίνεται μια μικρή παρουσίαση των διαφορετικών μεθόδων που δοκιμάσαμε για την υλοποίηση του σκοπού αυτού, με τα πλεονεκτήματα και τα μειονεκτήματα τους, καθώς και η τελική μέθοδος που επιλέξαμε. Και οι δύο προσεγγίσεις περιλαμβάνονται στον κώδικα του αντίστοιχου repository [47].



Σχήμα 5.1: *Απλή διάταξη μιας μονάδας πρόβλεψης που χρησιμοποιεί νευρωνικά δίκτυα για Hard-to-Predict branches.*

5.1.1.1 Ενσωμάτωση κώδικα Python - PyTorch στον ChampSim

Μια πρώτη προσέγγιση είναι ο συνδυασμός των δύο διαφορετικών λογισμικών, αυτό του προσομοιωτή και αυτό των νευρωνικών δικτύων. Ο ChampSim είναι ένα πρόγραμμα εξ ολοκλήρου γραμμένο σε γλώσσα προγραμματισμού C++. Ωστόσο η υλοποίηση των νευρωνικών δικτύων είναι σε κώδικα Python. Συνεπώς χρειάζεται να υλοποιήσουμε έναν δίαυλο επικοινωνίας μεταξύ προσομοιωτή και νευρωνικών, προκειμένου να έχουμε ζωντανές προβλέψεις κατά την εκτέλεση.

Το framework PyTorch [50] που χρησιμοποιήσαμε προσφέρει διαδραστικότητα μέσω δύο γλωσσών προγραμματισμού, μέσω C++. Η διαδικασία διαχείρισης ενός εκπαιδευμένου νευρωνικού δικτύου από τη μία γλώσσα προγραμματισμού στην άλλη, μπορεί να γίνει με κάποια απλά βήματα που παρέχονται στη σχετική βιβλιογραφία του framework. Έτσι μπορούμε να εκτελέσουμε την προσομοίωση μέσω ChampSim και κάθε φορά που θέλουμε πρόβλεψη για Hard-to-Predict branches, να στέλνουμε το ιστορικό στο νευρωνικό δίκτυο το οποίο παράγει μια πρόβλεψη.

Η συγκεκριμένη μέθοδος ήταν η πρώτη που δοκιμάσαμε, ωστόσο στην πράξη δεν ήταν αποτελεσματική. Ένα εμπρός πέρασμα (ή αλλιώς forward pass) ενός νευρωνικού δικτύου (μια πρόβλεψη στη περίπτωση μας) μπορεί να παραλληλοποιηθεί σε μεγάλο βαθμό και μάλιστα με δύο τρόπους. Ο ένας τρόπος είναι η παραλληλοποίηση των πράξεων, εσωτερικά των στοιχείων του νευρωνικού δικτύου. Ο δεύτερος είναι ως προς τις ταυτόχρονες προβλέψεις που μπορούμε να κάνουμε. Είναι εφικτό να φορτώσουμε στη μνήμη πολλά διαφορετικά παραδείγματα και

να πάρουμε πρόβλεψη για όλα ταυτόχρονα. Φυσικά και οι δύο τρόποι απαιτούν από τη μία πολλούς διαφορετικούς επεξεργαστικούς πυρήνες και από την άλλη επαρκή μνήμη για την φόρτωση των δεδομένων. Για τους λόγους αυτούς προτιμάται η χρήση GPUs έναντι CPUs κατά την εκπαίδευση (αλλά και έλεγχο) νευρωνικών δικτύων. Η εκτέλεση μέσω προσομοιωτή εισάγει μεγάλη καθυστέρηση στην εκτέλεση καθώς χρειάζεται είτε να επεξεργαζόμαστε τα δεδομένα 'αργά' εντός της CPU είτε μεταφέροντας κάθε φορά δεδομένα στην GPU.

Για τον λόγο αυτό χρειάστηκε να προσεγγίσουμε διαφορετικούς τρόπους επαλήθευσης των νευρωνικών καθώς ο απαιτούμενος χρόνος για την ολοκλήρωση μιας μόνο προσομοίωσης, ξεπερνούσε τις μερικές μέρες, ακόμα και σε πολυπύρηνα επεξεργαστικά συστήματα.

5.1.1.2 Έλεγχος νευρωνικών εκτός ChampSim και χρήση log files

Για να επωφεληθούμε την ταχύτητα που προσφέρουν οι GPUs είναι απαραίτητο να εγκαταλείψουμε την παραπάνω ιδέα των ταυτόχρονων προβλέψεων κατά την εκτέλεση της προσομοίωσης. Προσεγγίσαμε το πρόβλημα διαφορετικά, ελέγχοντας τα νευρωνικά δίκτυα offline, αποθηκεύσαμε τις προβλέψεις σε αρχεία καταγραφής, τα οποία έπειτα χρησιμοποιήσαμε για προβλέψεις σε κάποια προσομοίωση. Για κάθε προσομοίωση χρησιμοποιήσαμε αντίστοιχα datasets που χρησιμοποιήσαμε για την εκπαίδευση των νευρωνικών δικτύων, μόνο που αυτή τη φορά περιείχαν ιστορικά από τα benchmarks [8] και Simpoints που θέλουμε να ελέγξουμε την απόδοση. Διαβάζουμε σειριακά τα ιστορικά και παράγουμε προβλέψεις. Στη συνέχεια εκτελούμε την προσομοίωση με ChampSim αλλά αντί να περιμένουμε για την πρόβλεψη ενός Hard-to-Predict branch απλά την διαβάζουμε από το βιβλίο καταγραφής - log files. Η συγκεκριμένη μέθοδος αποτελεί δύο διαφορετικά βήματα, ένα της δημιουργίας των αρχείων καταγραφής και ένα της εκτέλεσης της προσομοίωσης, ωστόσο με την χρήση GPU η διαδικασία επιταχύνεται σημαντικά.

5.2 Αποτελέσματα

Σε αυτό το σημείο θα παρουσιάσουμε τα αποτελέσματα αυτής της διπλωματικής. Αρχικά κάνουμε μια σύγκριση του MPKI μεταξύ των διάφορων predictors σε 4 Benchmarks και στη συνέχεια μία σύγκριση ως προς τον αριθμό εκτέλεσης εντολών ανά κύκλο (Instructions Per Cycle - IPC).

Τα benchmarks που επιλέξαμε είναι τα ακόλουθα:

- **505.mcf** Σχεδιασμός διαδρομών (Route planning)
- **531.deepsjeng** Τεχνητή νοημοσύνη: Αναζήτηση Άλφα-Βήτα δέντρων - Σκάκι (Artificial Intelligence: alpha-beta tree search - Chess)
- **541.leela** Τεχνητή νοημοσύνη: Αναζήτηση Monte Carlo δέντρων - παιχνίδι Go (Artificial Intelligence: Monte Carlo tree search - Go game)
- **557.xz** Συμπίεση δεδομένων (Data compression)

Η επιλογή των συγκεκριμένων benchmarks βασίζεται κυρίως στη βιβλιογραφία. Για το BranchNet [1] η μεγαλύτερη αύξηση απόδοσης επιτυγχάνεται σε αυτά τα 4 benchmarks.

Συνεπώς θεωρούμε ότι αποτελεί ένα καλό σημείο για να ξεκινήσουμε να αξιολογούμε και τα δικά μας μοντέλα.

Υπενθυμίζουμε ότι τα νευρωνικά έχουν εκπαιδευτεί με χρήση των Alberta Workloads [45], ενώ τα αποτελέσματα που παρουσιάζονται εδώ είναι τα reference inputs από SPEC2017 [8]. Επίσης κάθε predictor αποτελείται από το νευρωνικό μοντέλο για προβλέψεις Hard-to-Predict branches και από τον Tage Predictor [12] για τα υπόλοιπα branches, σε 2 διαφορετικές εκδόσεις, μία των 8KB και μία των 64KB.

Στις παρακάτω γραφικές παραστάσεις τα μοντέλα που συγκρίνονται είναι τα παρακάτων:

- **Perfect BR**
  Σωστή πρόβλεψη όλων των Branches
- **Perfect H2P Predictor**
  Σωστή πρόβλεψη όλων των Hard-to-Predict (H2P) branches και χρήση TagePredictor8KB για τα υπόλοιπα.
- **BranchNet + Tage8/64KB**
  Χρήση του BranchNet [1] για πρόβλεψη Hard-to-Predict branches και Tage Predictor 8/64KB για τα υπόλοιπα.
- **Tarsa + Tage8/64KB**
  Χρήση του Tarsa predictor [2] για πρόβλεψη Hard-to-Predict branches και Tage Predictor 8/64KB για τα υπόλοιπα.
- **LSTM + Tage8/64KB**
  Χρήση του LSTM νευρωνικού που προτείναμε σε αυτή την διπλωματική στο κεφάλαιο 4.3.4 για τα Hard-to-Predict branches και αντίστοιχα Tage predictor για τα υπόλοιπα.

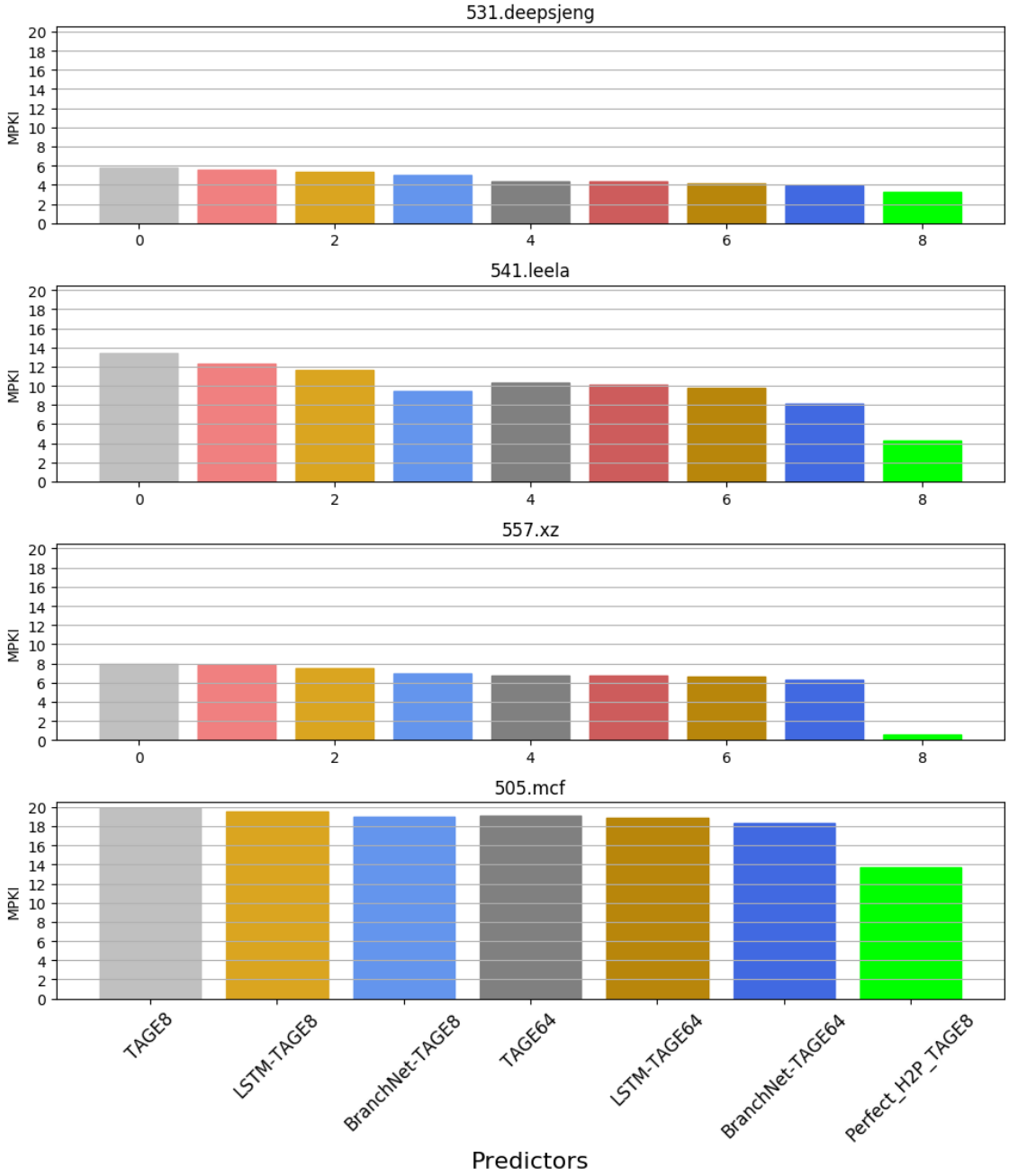
5.2.1 Misspredictions per Kilo Instructions

Η μέτρηση του Misspredictions per Kilo Instructions ή απλά MPKI μπορεί να γίνει σχετικά εύκολα, δεδομένου ότι χρησιμοποιούμε τα δεδομένα (βλ. 4.4.2.1) που παράχθηκαν για την εκπαίδευση των νευρωνικών μοντέλων.

Ως MPKI ορίζουμε τις λανθασμένες προβλέψεις ανά 1000 εντολές. Αν, λοιπόν, ελέγξουμε το νευρωνικό μοντέλο πάνω σε όλα τα ιστορικά κάποιου branch, σε ένα SimPoint, τότε εφόσον γνωρίζουμε τον συνολικό αριθμό εντολών που περιέχονται στο εν λόγο SimPoint μπορούμε εύκολα να υπολογίσουμε το MPKI.

Παρακάτω παρουσιάζονται γραφικές παραστάσεις (5.2), στις οποίες συγκρίνουμε το MPKI 4 Predictors σε 4 διαφορετικά Benchmarks. Αρχικά παρατηρούμε ένα προβάδισμα στους προβλεπτές που χρησιμοποιούν την έκδοση 64KB του Tage. Οι επιπλέον διακλαδώσεις που μπορεί να προβλέπει ο Tage λόγο της αυξημένης χωρητικότητας του, τείνει να καλύψει το κενό για την τέλεια πρόβλεψη Hard-to-Predict branches (πράσινη μπάρα). Το BranchNet [1] είναι το νευρωνικό που επιτυγχάνει την καλύτερη απόδοση σε κάθε περίπτωση. Η υλοποίηση μας με LSTM ακολουθεί αμέσως μετά πλησιάζοντας αρκετά τον πρώτο, χωρίς ωστόσο να καταφέρνει καλύτερη απόδοση σε κανένα από τα 4 Benchmarks. Τέλος, έχουμε την υλοποίηση του Tarsa [2], η οποία αν και δείχνει να βελτιώνει λίγο τον Tage των 8KB, όταν χρησιμοποιούμε

περισσότερους πόρους (έκδοση 64KB), η βελτίωση που προσφέρει είναι ελάχιστη. Επίσης αξίζει να σημειώσουμε ότι παρά την βελτίωση που προσφέρουν τα τεχνητά νευρωνικά δίκτυα, υπάρχει ακόμα μεγάλο περιθώριο βελτίωσης, καλύτερης πρόβλεψης Hard-to-Predict branches. Ειδικά στα benchmarks 505.mcf και 557.xz, μια τέλεια πρόβλεψη όλων των H2P θα πρόσφερε μεγάλη μείωση του MPKI.



Σχήμα 5.2: *Σύγκριση MPKI ανά Benchmark Σημειώνεται ότι για το 505.mcf δεν συλλέξαμε πληροφορίες για τον Tarsa Predictor.*

5.2.2 Instructions Per Cycle - IPC

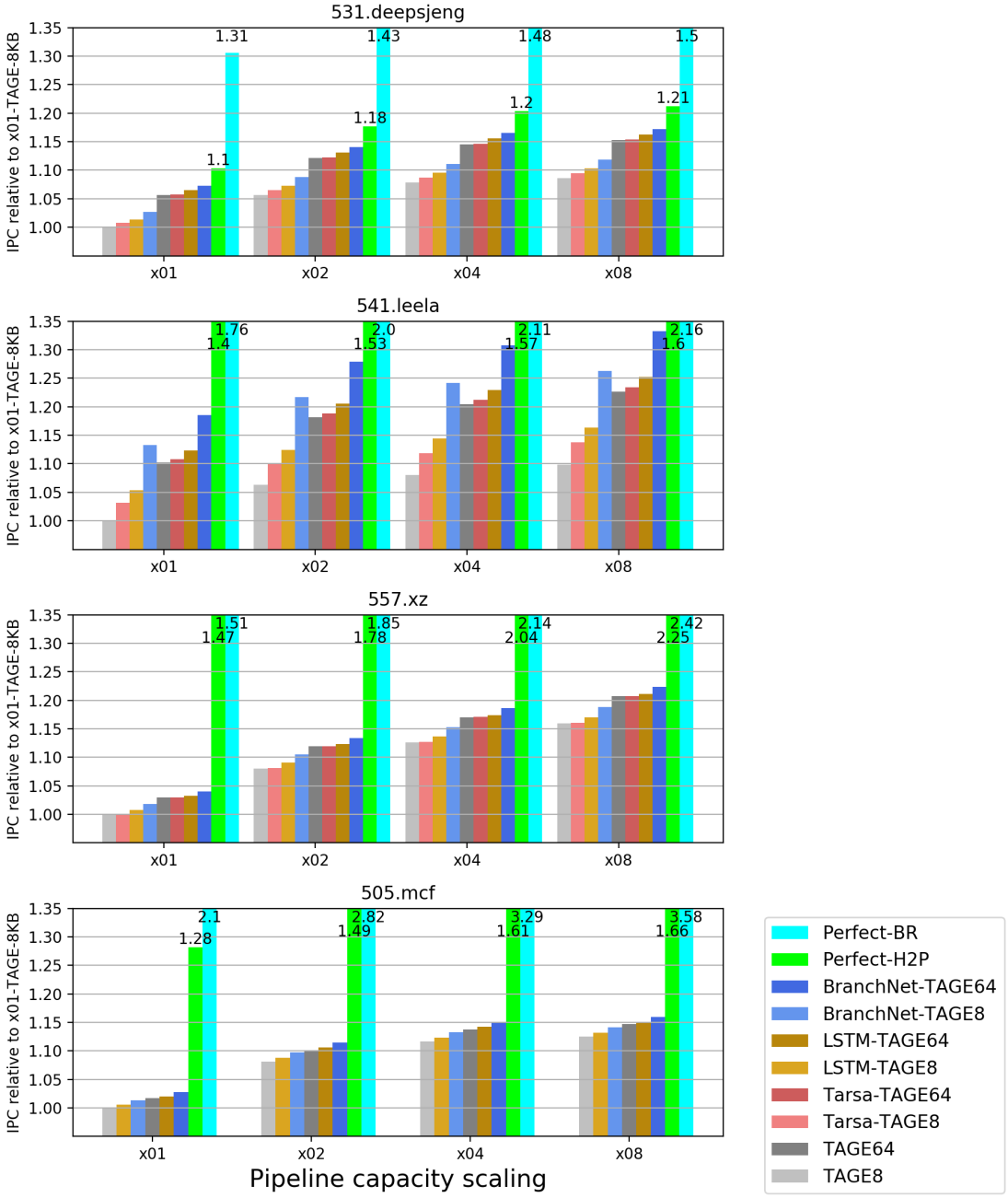
Αν και το MPKI μπορεί να μας δώσει μια σύγκριση μεταξύ των διαφορετικών branch predictors, δεν μπορεί να μας δώσει την επιτάχυνση που προσφέρει συνολικά στην επεξεργαστική μονάδα. Το IPC μας δείχνει τις εντολές που εκτελούνται σε έναν κύκλο ρολογιού του επεξεργαστή. Για απλές αρχιτεκτονικές που δεν χρησιμοποιούν σωλήνωση προφανώς η τιμή αυτή βρίσκεται πάντα κάτω από το 1, αφού στην καλύτερη περίπτωση θα εκτελείται μία εντολή σε κάθε κύκλο. Σε αρχιτεκτονικές όμως με σωλήνωση, ή ακόμα περισσότερο σε υπερβαθμωτές (superscalar) αρχιτεκτονικές, η τιμή αυτή μπορεί να είναι και μεγαλύτερη της μονάδας.

Στα αποτελέσματα παρακάτω παρουσιάζεται σύγκριση του IPC ανά predictor. Επίσης για να αναδείξουμε τη σημασία του branch prediction (όπως και στο [5]), παρουσιάζουμε την επιτάχυνση που θα μπορούσαν να προσφέρουν σε μελλοντικές υλοποιήσεις αρχιτεκτονικής οι οποίες θα είχαν περισσότερους πόρους, αυξάνοντας τη χωρητικότητα της σωλήνωσης (pipeline). Στον πίνακα 5.1 παρουσιάζουμε αναλυτικά τους πόρους που πολλαπλασιάσαμε, μαζί με τις αρχικές τιμές τους:

Πίνακας 5.1: *Παράμετροι σωλήνωσης και κλιμακούμενες (scaled-up) τιμές. Στην πρώτη στήλη βλέπουμε ποιες τιμές αλλάξαμε, στη δεύτερη στήλη βλέπουμε τις αρχικές τιμές (baseline configuration) και στη συνέχεια οι τιμές για κάθε αύξηση.*

Παράμετροι	**Αρχικές τιμές (Baseline)**	**×2**	**×4**	**×8**
Μέγεθος Re-Order buffer	352	704	1408	2816
Μέγεθος Scheduler	128	256	512	1024
Αριθμός εντολών που γίνονται fetch, decode και execute σε κάθε κύκλο	6	12	24	48
Μέγεθος Load queue	128	256	512	1024
Μέγεθος Store queue	72	144	288	576
Αριθμός Load και Store εντολών που φορτώνονται σε κάθε κύκλο	2	4	8	16
Αριθμός εντολών που γίνονται retire σε κάθε κύκλο	4	8	16	32
Μέγεθος Decoded instruction cache	2048	4096	8192	16384

Παρακάτω, στο γράφημα 5.3 βλέπουμε μια σύγκριση της αύξησης του IPC, που προσφέρει το κάθε νευρωνικό δίκτυο, σε 4 διαφορετικούς παραμέτρους συστήματος (configurations), σε 4 διαφορετικά benchmarks. Όπως είναι αναμενόμενο, η ταξινόμηση των τεχνητών νευρωνικών δικτύων ακολουθεί την διάταξη που παρατηρήσαμε στην μέτρηση του MPKI στο 5.2.1. Επιπλέον περιλαμβάνουμε μία ακόμα μπάρα, αυτή της αύξησης του IPC που θα είχαμε με μία τέλεια πρόβλεψη όλων των εντολών διακλάδωσης. Παρακάτω στην ενότητα 5.3 αναλύουμε τα αποτελέσματα ξεχωριστά, για κάθε τεχνητό νευρωνικό δίκτυο.



Σχήμα 5.3: *Σύγκριση IPC ανά Benchmark. Σημειώνεται ότι για το 505.mcf δεν συλλέξαμε πληροφορίες για τον Tarsa Predictor.*

5.3 Συμπεράσματα

Από τις παραπάνω γραφικές παραστάσεις (5.2.1, 5.3) καταλήγουμε σε δύο πορίσματα. Το ένα αφορά την αξιοποίηση των νευρωνικών δικτύων στον τομέα του branch prediction. Το δεύτερο σχετίζεται με το περιθώριο βελτίωσης που ξεκάθαρα φαίνεται να υπάρχει στο συγκεκριμένο τομέα, παρά την μεγάλη τεχνολογική πρόοδο που έχουμε αναπτύξει.

5.3.1 Σχολιασμός νευρωνικών δικτύων

Ακολουθεί ένας σύντομος σχολιασμός του κάθε νευρωνικού δικτύου, σύμφωνα με τα αποτελέσματα που προέκυψαν σε αυτή τη διπλωματική.

5.3.1.1 Tarsa Predictor

Η αρχική πρόταση των Stephen J. Tarsa, Chit-Kwan Lin, Gokce Keskin, Gautham Chinya, Hong Wang [2]προσφέρει πολύ μικρή βελτίωση. Όπως είναι λογικό βρίσκεται τελευταίο στην κατάταξη των νευρωνικών δικτύων, αφού αφενός έχει μια πιο απλή αρχιτεκτονική και αφετέρου δέχεται σαν είσοδο μικρότερη πληροφορία (μικρότερο ιστορικό και απλούστερη κωδικοποίηση). Παρόλα αυτά αποτέλεσε σαν πρώτη προσέγγιση μια ένδειξη ότι τα τεχνητά νευρωνικά δίκτυα αξίζουν να ερευνηθούν περαιτέρω στον τομέα πρόβλεψης διακλαδώσεων.

5.3.1.2 BranchNet

Είναι προφανές ότι το BranchNet προσφέρει την καλύτερη δυνατή πρόβλεψη Hard-to-Predict διακλαδώσεων, από όλα τα τεχνητά νευρωνικά δίκτυα που εξετάσαμε. Βλέπουμε ότι ακόμα και όταν τον ενσωματώνουμε στην σχετικά μεγάλης χωρητικότητας έκδοση του Tage, μπορεί να επιταχύνει αρκετά τον επεξεργαστή καθώς προβλέπει Branches στα οποία ο Tage αποτυγχάνει. Ειδικά σε σύγκριση με την πιο ρεαλιστική έκδοση των 8KB, βλέπουμε ένα ξεκάθαρο πλεονέκτημα της χρήσης του BranchNet για πρόβλεψη Hard-to-Predict branches.

5.3.1.3 Convolutional - LSTM νευρωνικό δίκτυο

Η πρόταση μας μπορεί να μην υπερέχει του BranchNet, παρατηρούμε όμως ότι ανταγωνίζεται σε ικανοποιητικό βαθμό την αρκετά πιο περίπλοκη αρχιτεκτονική του πρώτου, παρά το γεγονός ότι αποτελείται από μόνο δύο επίπεδα, ένα εξαγωγής χαρακτηριστικών (feature extraction) και ένα LSTM. Η διαφορά του σε σχέση με τον Tage παρατηρείται κυρίως στην μικρότερη έκδοση, ενώ περισσότεροι πόροι στον Tage (έκδοση των 64KB) τείνουν να καλύψουν το κενό. Ωστόσο αξίζει να αναφέρουμε ότι η υλοποίηση μας με LSTM είναι μια πρώτη προσέγγιση και απαιτείται περαιτέρω ερεύνα η οποία πιθανόν να οδηγήσει σε καλύτερα αποτελέσματα.

5.3.2 Σχεδιασμός σε επίπεδο υλικού

Τα τεχνητά νευρωνικά δίκτυα που παρουσιάζουμε σε αυτή τη διπλωματική είναι σε επίπεδο λογισμικού· πρακτικά, όμως, ένας προβλεπτής εντολών διακλάδωσης λειτουργεί σε επίπεδο

υλικού (hardware level). Στη συγκεκριμένη διπλωματική δεν ασχοληθήκαμε με την αναπαράσταση των τεχνητών νευρωνικών δικτύων σε πρακτικό επίπεδο, ικανό δηλαδή να μπορεί να χρησιμοποιηθεί ενσωματωμένο (on-chip) πάνω σε μια αρχιτεκτονική ενός επεξεργαστή. Παραπάνω, στις περιπτώσεις του Tarsa (βλ. 5.3.1.1) και BranchNet (βλ. 5.3.1.2) έχει υπάρξει στη σχετική βιβλιογραφία και μία πρόταση των μοντέλων σε επίπεδο υλικού [2] [1]. Παρά την πολυπλοκότητα των μοντέλων, μπορούν να περιοριστούν σε εκδόσεις οι οποίες χρησιμοποιούν λίγους πόρους (μέσα στα πρακτικά όρια ενός branch predictor), χωρίς να χάνουν μεγάλο μέρος της απόδοσης τους [1].

Όσον αφορά την δική μας πρόταση, δεν κάναμε κάποια πρόταση ως προς τον σχεδιασμό ενός πρακτικού predictor. Παρόλα αυτά θεωρούμε ότι μια υλοποίηση του είναι εφικτή. Τα πρώτα επίπεδα κωδικοποίησης με embeddings και τα συνελικτικά στρώματα (convolutional layers), τα οποία εκτελούν εξαγωγή χαρακτηριστικών (feature extraction), μπορούν να σχεδιαστούν όπως σχεδιάστηκε και το BranchNet [1], το οποίο περιλαμβάνει αντίστοιχα επίπεδα στην αρχιτεκτονική του. Για το αναδρομικό στρώμα του μοντέλου, το δίκτυο μακράς βραχυπρόθεσμης μνήμης (long short-term memory) δεν υπάρχει αντίστοιχη δουλειά πάνω στην ενσωμάτωση του σε έναν πρακτικό προβλεπτή εντολών διακλάδωσης. Ωστόσο υπάρχουν προηγούμενες δουλειές που αναφέρονται σε υλοποίηση με LSTM σε hardware level. Οι Guan, Yijin and Yuan, Zhihang and Sun, Guangyu and Cong, Jason αναφέρθηκαν σε μια υλοποίηση με FPGA επιταχυντές για LSTM [51]. Ακόμα οι Song Han, Junlong Kang, Huizi Mao, Yiming Hu, Xin Li, Yubin Li, Dongliang Xie, Hong Luo, Song Yao, Yu Wang, Huazhong Yang, William J. Dally, παρουσίασαν μια μέθοδο συμπίεσης ενός μεγάλου LSTM μοντέλου και την σχεδίασή του με FPGA [52]. Τέλος οι Kazybek Adam, Kamilya Smagulova, Alex Pappachen James, πρότειναν LSTM μοντέλο με ηλεκτρονικά κυκλώματα [53]. Θεωρούμε, λοιπόν, ότι μια πρακτική υλοποίηση μιας μονάδας πρόβλεψης εντολών διακλάδωσης με δίκτυα μακράς βραχυπρόθεσμης μνήμης, είναι πραγματοποιήσιμη.

5.3.3 Ζήτημα βελτίωσης του branch prediction

Παρά το γεγονός ότι ο Tage Predictor προσφέρει πάνω από 99% επιτυχία σε όλα τα branches μιας προσομοίωσης, βλέπουμε ότι μια τέλεια πρόβλεψη για όλα τα branches, ή έστω των Hard-to-Predict, έχει πολύ μεγάλο αντίκτυπο στην επιτάχυνση του επεξεργαστή. Σε όλα τα benchmarks βλέπουμε ότι ακόμα και με την υπάρχουσα αρχιτεκτονική μια τέλεια πρόβλεψη όλων των διακλαδώσεων θα μπορούσε ακόμα και να διπλασιάσει την ταχύτητα εκτέλεσης (όπως για παράδειγμα στην περίπτωση του 505.mcf). Επιπλέον σε μελλοντικές υλοποιήσεις, όπου πιθανότητα η τεχνολογία να μας επιτρέψει κατασκευή αρχιτεκτονικών, ικανών να διαχειρίζονται πολύ περισσότερες εντολές ταυτόχρονα, οι σύγχρονοι state-of-the-art predictors, θα δημιουργούσαν φραγμούς στην αξιοποίηση τους.

Μπορεί τα Hard-to-Predict branches να αποτελούν κάποιες λίγες εντολές (βλ. 4.1), όμως η σωστή πρόβλεψη τους προσφέρει μεγάλη επιτάχυνση του επεξεργαστή. Επιπλέον έχουμε ορίσει τα branches αυτά με την προϋπόθεση ότι εκτελούνται πολλές χιλιάδες φορές, συνεπώς προσφέρουν τη δυνατότητα εκπαίδευσης νευρωνικών δικτύων. Βλέπουμε ότι στα benchmarks 531.deepsjeng και 541.leela τα νευρωνικά δίκτυα επιτυγχάνουν το σκοπό τους, ωστόσο δεν μπορούμε να πούμε το ίδιο για τα 557.xz και 505.mcf. Παρόλα αυτά η σωστή πρόβλεψη

αποκλειστικά και μόνο αυτών των λιγοστών branches θα προσφέρουν εξαιρετικά υψηλή επιτάχυνση του επεξεργαστή στις συγκεκριμένες εφαρμογές. Εν κατακλείδι, θεωρούμε ότι αξίζει η έρευνα επιπρόσθετων νευρωνικών μοντέλων, ικανών να εκπαιδευτούν και να ανταποκριθούν στις ιδιαιτερότητες διαφορετικών εφαρμογών.

Κεφάλαιο 6

Επίλογος

6.1 Σύνοψη

Η πρόβλεψη διακλαδώσεων (branch prediction) αποτελεί ένα πολύ σημαντικό τμήμα των σύγχρονων επεξεργαστών. Η αξιοποίηση και η μεγιστοποίηση της διεκπεραιωτικής ικανότητας των Κ.Μ.Ε. είναι άρρηκτα συνδεδεμένη με την επίδοση της μονάδας branch prediction. Παρά την τεράστια πρόοδο στον συγκεκριμένο τομέα, και την υψηλή επίδοση των υφιστάμενων προβλεπτών - predictors, υπάρχει ακόμα ένα μεγάλο περιθώριο βελτίωσης. Όπως δείξαμε, ειδικότερα σε μελλοντικές αρχιτεκτονικές θα χρειαστεί να παρουσιαστούν αποτελεσματικότεροι μέθοδοι branch prediction προκειμένου να εκμεταλλευτούμε στο έπακρο τους διαθέσιμους πόρους ενός επεξεργαστή.

Τα τεχνητά νευρωνικά δίκτυα αποτελούν μία ελκυστική προσέγγιση στο ζήτημα πρόβλεψης διακλαδώσεων. Μπορεί οι κλασσικοί predictors να λειτουργούν ταυτόχρονα με το περιβάλλον εκτέλεσης του προγράμματος και να αναγνωρίζουν μοτίβα σύγχρονα· ωστόσο θα μπορούσαμε να εκπαιδεύσουμε προβλεπτές εκτός σύνδεσης - offline για κάποιες δύσκολες - Hard-to-Predict διακλαδώσεις, στις οποίες οι state-of-the-art predictors αποτυγχάνουν.

Δείξαμε πως οι υπάρχουσες προτάσεις [2] [1] προσφέρουν σημαντική αύξηση στην παραγωγή εντολών ανά κύκλο - IPC, ενώ υλοποιήσαμε και μια δική μας εναλλακτική πρόταση η οποία περιλαμβάνει μακράς-βραχυπρόθεσμης μνήμης (LSTM) νευρωνικά δίκτυα. Είδαμε πως όλες οι προτάσεις, σε συνδυασμό με τον Tage predictor, μπορούν να προσφέρουν μια σημαντική αύξηση της απόδοσης του προσομοιωτή ChampSim, σε στοχευμένες εφαρμογές.

Τέλος θεωρούμε ότι τα τεχνητά νευρωνικά δίκτυα, και γενικότερα ο τομέας της μηχανικής μάθησης, φαίνεται να μπορούν να αξιοποιηθούν στον τομέα του branch prediction. Τόσο τα συνελικτικά όσο και τα αναδρομικά νευρωνικά δίκτυα, δείχνουν να αποδίδουν αρκετά καλά και να επιτυγχάνουν υψηλή απόδοση στην πρόβλεψη διακλαδώσεων, εκεί που οι κλασσικοί predictors αποτυγχάνουν. Τα τεχνητά νευρωνικά δίκτυα φαίνεται να αποτελούν ένα εξαιρετικά δυνατό εργαλείο, το οποίο χρήζει περαιτέρω διερεύνησης.

6.2 Μελλοντικές Επεκτάσεις

Η πρόβλεψη διακλαδώσεων με τεχνητά νευρωνικά δίκτυα που αναπτύξαμε στα πλαίσια αυτής της διπλωματικής, προσφέρει την δυνατότητα διερεύνησης σε πολλές διαφορετικές κατευθύνσεις. Συγκεκριμένα, αναφέρονται τα ακόλουθα:

- **Διερεύνηση μοντέλων τεχνητών νευρωνικών δικτύων**
  Μια πρώτη προφανής επέκταση είναι η εξέταση επιπλέον, πιο σύνθετων μοντέλων τα οποία ενδεχομένως να ανταποκριθούν καλύτερα στις απαιτήσεις των διακλαδώσεων, αφού, όπως δείξαμε στο κεφάλαιο 6, υπάρχει ακόμα ένα μεγάλο περιθώριο σωστής πρόβλεψης των Δύσκολων-να-Προβλεφθούν διακλαδώσεων που εντοπίσαμε.

- **Δεδομένα εισόδου**
  Τα δεδομένα πάνω στα οποία εκπαιδεύονται τα νευρωνικά μοντέλα αποτελούν έναν καθοριστικό παράγοντα στην απόδοση τους. Στην υπάρχουσα βιβλιογραφία, όπως και σε αυτή την διπλωματική, γίνεται μελέτη με χρήση μόνο του ιστορικού προηγούμενων διακλαδώσεων. Μια διαφορετική προσέγγιση θα ήταν η αξιοποίηση επιπλέον δεδομένων τιμών, όπως των καταχωρητών του επεξεργαστή. Η τροφοδότηση τέτοιων τιμών σε ένα τεχνητό νευρωνικό δίκτυο, πιθανότατα να ενισχύσει την ευστοχία του.

- **Σχεδιασμός σε επίπεδο υλικού**
  Στην διπλωματική αυτή ασχοληθήκαμε με την υλοποίηση νευρωνικών δικτύων σε επίπεδο λογισμικού. Μια μονάδα πρόβλεψης διακλαδώσεων όμως, λειτουργεί σε επίπεδο υλικού - hardware. Είναι, λοιπόν, απαραίτητο να παρουσιαστούν τρόποι αναπαράστασης των νευρωνικών δικτύων με τρόπους τέτοιους, ικανούς να ενσωματωθούν πάνω στον επεξεργαστή. Ήδη έχουν γίνει προτάσεις στην βιβλιογραφία [1] [2]· ωστόσο οι προτάσεις αυτές έχουν μειωμένη απόδοση σε σχέση με τα αρχικά νευρωνικά μοντέλα, ενώ πολύ μικρά βήματα έχουν γίνει όσον αφορά την αναπαράσταση αναδρομικών μοντέλων, όπως τα LSTM.

Παραρτήματα

Παράρτημα Α΄

Παράρτημα πηγαίου κώδικα

Α΄.1 Πηγαίος κώδικας Tarsa

Αλγοριθμος Α΄.1: *Ο Tarsa CNN predictor [2] σε κώδικα Python - PyTorch.*

```
    class TarsaCNNPredictor(nn.Module):
        def __init__(self, tableSize=256, numFilters=2,
history_length=200):
            super().__init__()
            # 1-hot encoding
            self.E = torch.eye(tableSize, dtype=torch.float32)
            self.c1 = nn.Conv2d(tableSize, numFilters, (1,1))
            self.tahn = nn.Tanh()
            self.l = nn.Linear(history_length*numFilters, 1)
            self.sigmoid = nn.Sigmoid()

        def forward(self, seq):
            # Embed by expanding sequence of ((IP << 7) + dir ) &
(255)
            # integers into 1-hot history matrix during training
            xFull = self.E[seq.data.long()]
            # Permute so encodings equal the number of input
channels
            xFull = xFull.permute(0,3,1,2)#.to(device)
            h1 = self.c1(xFull)
            h1a = self.tahn(h1)
            # Reshape for Linear
            h1a = h1a.reshape(h1a.size(0),h1a.size(1)*h1a.size(3))
            out = self.l(h1a)
            out = self.sigmoid(out)
            return out

```

Α΄.2 Πηγαίος κώδικας της υλοποίησης μας

Αλγοριθμος Α΄.2: *Ο CNN - LSTM predictor που υλοποιήσαμε σε κώδικα Python - PyTorch.*

```
class CNN_LSTM(nn.Module):
  def __init__(self,  input_dim, out_dim, num_layers, pc_hash_bits=12, batch_first=True, history_length=582):
      super().__init__()
      self.input_dim = input_dim
      self.out_dim = out_dim
      self.num_layers = num_layers
      self.batch_first = batch_first
      self.pc_hash_bits = pc_hash_bits
      self.history_length = history_length

      self.embedding1 = nn.Embedding(2**self.pc_hash_bits+1,self.input_dim)
      self.conv1 = nn.Conv1d(in_channels=self.input_dim, out_channels=64, kernel_size=1)
      self.relu1 = nn.ReLU()
      self.conv2 = nn.Conv1d(in_channels=64, out_channels=128, kernel_size=1)
      self.relu2 = nn.ReLU()
      self.conv3 = nn.Conv1d(in_channels=128, out_channels=self.out_dim, kernel_size=1)
      self.tanh = nn.ReLU()
      self.rnn = nn.LSTM(input_size = self.out_dim, hidden_size=512,\
                      num_layers= self.num_layers,
                      batch_first=self.batch_first,
                      bidirectional=False)
      self.fc1 = nn.Linear(512, 128)
      self.relufc = nn.ReLU()
      self.fc2 = nn.Linear(128, 1)
      self.finalActivation = nn.Tanh()
  def forward(self, seq):
      x = self.embedding1(seq.long())
      # Change shape (batchSize, historyLength, features)
      # to (batchSize, features, historyLength)
      x = x.transpose(1,2)
      # Feature extraction
      x = self.relu1(self.conv1(x))
      x = self.relu2(self.conv2(x))
      x = self.tanh(self.conv3(x))
      x=x.squeeze(dim=2)
      # Convert again to (batchSize, historyLength, features)
      x = x.transpose(1,2)
      # LSTM
      _, (x, _) = self.rnn(x)
      x = x.transpose(0,1)
      x = x.flatten(1)
      # Linear layers
      x = self.relufc(self.fc1(x))
      x = self.fc2(x)
      return self.finalActivation(x).squeeze(dim=1)

```

Βιβλιογραφία

[1] Siavash Zangeneh, Stephen Pruett, Sangkug Lym και Yale N. Patt. *BranchNet: A Convolutional Neural Network to Predict Hard-To-Predict Branches. 2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, σελίδες 118–130, 2020.

[2] Stephen Tarsa, Chit Kwan Lin, Gokce Keskin, Gautham Chinya και Hong Wang. *Improving Branch Prediction By Modeling Global History with Convolutional Neural Networks*. 2019.

[3] Milad Hashemi, Kevin Swersky, Jamie A. Smith, Grant Ayers, Heiner Litz, Jichuan Chang, Christos Kozyrakis και Parthasarathy Ranganathan. *Learning Memory Access Patterns*, 2018.

[4] H. Khalid και M. S. Obaidat. *Application of Neural Networks to Cache Replacement. Neural Computing & Applications*, 8(3):246–256, 1999.

[5] Chit-Kwan Lin και Stephen J. Tarsa. *Branch Prediction Is Not a Solved Problem: Measurements, Opportunities, and Future Directions. CoRR*, αβς/1906.08170, 2019.

[6] Sepp Hochreiter και Jürgen Schmidhuber. *Long Short-Term Memory. Neural Computation*, 9(8):1735–1780, 1997.

[7] *ChampSim Repository*. https://github.com/ChampSim/ChampSim. Ημερομηνία πρόσβασης: 17-09-2020.

[8] *SPEC CPU 2017*. https://www.spec.org/cpu2017/. Ημερομηνία πρόσβασης: 17-06-2021.

[9] Warren S. McCulloch και Walter Pitts. *A logical calculus of the ideas immanent in nervous activity. The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[10] Rolandos Alexandros Potamias και Georgios Siolas. *NTUA-ISLab at SemEval-2019 Task 3: Determining emotions in contextual conversations with deep learning. Proceedings of the 13th International Workshop on Semantic Evaluation*, σελίδες 277–281, 2019.

[11] Rolandos Alexandros Potamias, Georgios Siolas και Andreas Georgios Stafylopatis. *A transformer-based approach to irony and sarcasm detection. Neural Computing and Applications*, 32(23):17309–17320, 2020.

[12] André Seznec. *TAGE-SC-L Branch Predictors Again. 5th JILP Workshop on Computer Architecture Competitions (JWAC-5): Championship Branch Prediction (CBP-5)*, Seoul, South Korea, 2016.

[13] Siavash Zangeneh. *BranchNet*. https://github.com/siavashzk/BranchNet, 2020.

[14] *The 3rd Data Prefetching Championship*. https://dpc3.compas.cs.stonybrook.edu/.

[15] *Championship Branch Prediction*. https://www.jilp.org/cbp2016/.

[16] G. Hinton. *The microarchitecture of the Pentium 4 processor*. 2001.

[17] W. Hwu και Y. N. Patt. *HPSm, a High Performance Restricted Data Flow Architecture Having Minimal Functionality. SIGARCH Comput. Archit. News*, 14(2):297–306, 1986.

[18] Scott McFarling και John Hennessy. *Reducing the cost of branches*. σελίδες 117–124, 1995.

[19] James R Larus. *Spim s20: A mips r2000 simulator*. Τεχνική Αναφορά με αριθμό, University of Wisconsin-Madison Department of Computer Sciences, 1990.

[20] R.B. Garner, A. Agrawal, F. Briggs, E.W. Brown, D. Hough, B. Joy, S. Kleiman, S. Muchnick, M. Namjoo, D. Patterson, J. Pendleton και R. Tuck. *The scalable processor architecture (SPARC). Digest of Papers. COMPCON Spring 88 Thirty-Third IEEE Computer Society International Conference*, σελίδες 278–283, 1988.

[21] Chih Cheng Cheng. *The Schemes and Performances of Dynamic Branch predictors*. 2000.

[22] Shien Tai Pan, Kimming So και Joseph T. Rahmeh. *Improving the Accuracy of Dynamic Branch Prediction Using Branch Correlation. Proceedings of the Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS V, σελίδα 76–84, New York, NY, USA, 1992. Association for Computing Machinery.

[23] Lee και Smith. *Branch Prediction Strategies and Branch Target Buffer Design. Computer*, 17(1):6–22, 1984.

[24] Tse Yu Yeh και Yale N. Patt. *Two-Level Adaptive Training Branch Prediction. Proceedings of the 24th Annual International Symposium on Microarchitecture*, MICRO 24, σελίδα 51–61, New York, NY, USA, 1991. Association for Computing Machinery.

[25] Tse Yu Yeh και Yale N. Patt. *Alternative Implementations of Two-Level Adaptive Branch Prediction. Proceedings of the 19th Annual International Symposium on Computer Architecture*, ISCA '92, σελίδα 124–134, New York, NY, USA, 1992. Association for Computing Machinery.

[26] Tse Yu Yeh και Yale N. Patt. *A Comparison of Dynamic Branch Predictors That Use Two Levels of Branch History. Proceedings of the 20th Annual International Symposium on Computer Architecture*, ISCA '93, σελίδα 257–266, New York, NY, USA, 1993. Association for Computing Machinery.

[27] S. McFarling. *Combining Branch Predictors*. 1993.

[28] T. Mudge, I. Chen και J. T. Coffey. *Limits to Branch Prediction*. 2000.

[29] J. Cleary και I. Witten. *Data Compression Using Adaptive Coding and Partial String Matching. IEEE Transactions on Communications*, 32(4):396–402, 1984.

[30] D. A. Jimenez και C. Lin. *Dynamic branch prediction with perceptrons. Proceedings HPCA Seventh International Symposium on High-Performance Computer Architecture*, σελίδες 197–206, 2001.

[31] D.A. Jimenez. *Fast path-based neural branch prediction. Proceedings. 36th Annual IEEE/ACM International Symposium on Microarchitecture, 2003. MICRO-36.*, σελίδες 243–252, 2003.

[32] Timothy Sherwood και Brad Calder. *Loop Termination Prediction. High Performance Computing*Mateo Valero, Kazuki Joe, Masaru Kitsuregawa και Hidehiko Tanaka, επιμελητές, σελίδες 73–87, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[33] J. Albericio, J. S. Miguel, N. E. Jerger και A. Moshovos. *Wormhole: Wisely Predicting Multidimensional Branches. 2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, σελίδες 509–520, 2014.

[34] A. Seznec, J. S. Miguel και J. Albericio. *The inner most loop iteration counter: A new dimension in branch history. 2015 48th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, σελίδες 347–357, 2015.

[35] M. U. Farooq, Khubaib και L. K. John. *Store-Load-Branch (SLB) predictor: A compiler assisted branch prediction for data dependent branches. 2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*, σελίδες 59–70, 2013.

[36] Frank Rosenblatt. *The Perceptron—a perceiving and recognizing automaton*. Cornell Aeronautical Laboratory, 1957.

[37] M.W Gardner και S.R Dorling. *Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences. Atmospheric Environment*, 32(14):2627–2636, 1998.

[38] Andrew L. Maas. *Rectifier Nonlinearities Improve Neural Network Acoustic Models*. 2013.

[39] Y. Lecun, L. Bottou, Y. Bengio και P. Haffner. *Gradient-based learning applied to document recognition. Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[40] David E. Rumelhart, Geoffrey E. Hinton και Ronald J. Williams. *Learning representations by back-propagating errors. Nature*, 323(6088):533–536, 1986.

[41] Dzmitry Bahdanau, Kyunghyun Cho και Yoshua Bengio. *Neural Machine Translation by Jointly Learning to Align and Translate. arXiv e-prints*, σελίδα arXiv:1409.0473, 2014.

[42] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser και Illia Polosukhin. *Attention Is All You Need.* 2017.

[43] Aristotelis Vontzalidis. *Πρόβλεψη διακλαδώσεων με τεχνητά νευρωνικά δίκτυα*. https://github.com/aristotelis96/diploma, 2021.

[44] *Pin tool.* https://software.intel.com/sites/landingpage/pintool/docs/81205/Pin/html/. Ημερομηνία πρόσβασης: 17-09-2020.

[45] J. N. Amaral, E. Borin, D. R. Ashley, C. Benedicto, E. Colp, J. H. S. Hoffmam, M. Karpoff, E. Ochoa, M. Redshaw και R. E. Rodrigues. *The Alberta Workloads for the SPEC CPU 2017 Benchmark Suite. 2018 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, σελίδες 159–168, 2018.

[46] Erez Perelman, Greg Hamerly, Michael Biesbrouck, Timothy Sherwood και Brad Calder. *Using SimPoint for accurate and efficient simulation. ACM SIGMETRICS Performance Evaluation Review*, 31:318–319, 2003.

[47] Aristotelis Vontzalidis. *ChampSim*. https://github.com/aristotelis96/ChampSim, 2021.

[48] Ian Goodfellow, Yoshua Bengio και Aaron Courville. *Deep Learning.* MIT Press, 2016.

[49] Greg Van Houdt, Carlos Mosquera και Gonzalo Nápoles. *A Review on the Long Short-Term Memory Model. Artificial Intelligence Review*, 53, 2020.

[50] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai και Soumith Chintala. *PyTorch: An Imperative Style, High-Performance Deep Learning Library. Advances in Neural Information Processing Systems 32*, σελίδες 8024–8035. Curran Associates, Inc., 2019.

[51] Yijin Guan, Zhihang Yuan, Guangyu Sun και Jason Cong. *FPGA-based accelerator for long short-term memory recurrent neural networks. 2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*, σελίδες 629–634, 2017.

[52] Song Han, Junlong Kang, Huizi Mao, Yiming Hu, Xin Li, Yubin Li, Dongliang Xie, Hong Luo, Song Yao, Yu Wang, Huazhong Yang και William J. Dally. *ESE: Efficient Speech Recognition Engine with Sparse LSTM on FPGA*, 2017.

[53] Kazybek Adam, Kamilya Smagulova και Alex James. *Memristive LSTM network hardware architecture for time-series predictive modeling problems.* σελίδες 459–462, 2018.

Συντομογραφίες - Αρκτικόλεξα - Ακρωνύμια

βλ.	βλέπε
κτλ.	και τα λοιπά
κ.ο.κ	και ούτω καθεξής
π.χ.	παραδείγματος χάριν
ΚΜΕ	Κεντρική Μονάδα Επεξεργασίας
SGD	Stochastic Gradient Descent
RNN	Recurrent Neural Network
LSTM	Long-Short Term Memory