Εθνικο Μετσοβιο Πολυτεχνειο
Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων

Τομεας Τεχνολογιας Πληροφορικης και Υπολογιστων
Εργαστηριο Υπολογιστικων Συστηματων

Πρόβλεψη χρόνου εκπαίδευσης νευρωνικών δικτύων μέσω πειραματικής αποτίμησης επίδοσης τελεστών σε κατανεμημένο περιβάλλον

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

Βλαντισλάβ Π.
Λαμπρινίδη-Λέντελ

Επιβλέπων: Νεκτάριος Κοζύρης
Καθηγητής Ε.Μ.Π.

Αθήνα, Οκτώβριος 2020

Εθνικο Μετσοβιο Πολυτεχνειο
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

Πρόβλεψη χρόνου εκπαίδευσης νευρωνικών δικτύων μέσω πειραματικής αποτίμησης επίδοσης τελεστών σε κατανεμημένο περιβάλλον

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

Βλαντισλάβ Π.
Λαμπρινίδη-Λέντελ

Επιβλέπων: Νεκτάριος Κοζύρης
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 14η Οκτωβρίου 2020.

.....	.....	.....
Νεκτάριος Κοζύρης	Γεώργιος Γκούμας	Ιωάννης Κωνσταντίνου
Καθηγητής	Επίκ. Καθηγητής	Επίκ. Καθηγητής
Ε.Μ.Π.	Ε.Μ.Π.	Παν. Θεσσαλίας

Αθήνα, Οκτώβριος 2020

.....

Βλαντισλάβ Π. Λαμπρινίδης-Λέντελ
Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών

Περίληψη

Τα τεχνητά νευρωνικά δίκτυα αποτελούν δημοφιλή τεχνική του τομέα της Μηχανικής Μάθησης, η οποία προσομοιώνει, ως ένα βαθμό, τον μηχανισμό μάθησης των βιολογικών οργανισμών. Τροφοδοτούνται με πληθώρα δεδομένων ώστε να εκπαιδευτούν, και, ύστερα, να πραγματοποιούν ποικίλες εργασίες, όπως είναι η αναγνώριση αντικειμένων σε μία φωτογραφία, ή ακόμα και η διάγνωση κάποιας ασθένειας. Παράλληλα, ο όγκος των δεδομένων, που συνεχώς αυξάνεται, έχει οδηγήσει στην στην ανάπτυξη πληθώρας κατανεμημένων συστημάτων εκπαίδευσης νευρωνικών δικτύων, με πιο δημοφιλή το TensorFlow και το PyTorch. Ωστόσο, οι ποικίλες διαθέσιμες επιλογές θέτουν ένα σημαντικό ερώτημα ως προς το ποια είναι η κατάλληλη επιλογή συστήματος για την εκπαίδευση του εκάστοτε δικτύου. Τέτοιου είδους ερωτήματα μπορούν να απαντηθούν με την υλοποίηση και χρήση κατάλληλων χρονοδρομολογητών, οι οποίοι παρέχουν πληροφορίες για την εκπαίδευσή τους, όπως, για παράδειγμα, ο εκτιμώμενος χρόνος εκτέλεσης.

Στην παρούσα διπλωματική γίνεται υλοποίηση ενός συστήματος πρόβλεψης του χρόνου εκπαίδευσης νευρωνικών δικτύων σε κατανεμημένο περιβάλλον μέσω πειραματικών δεδομένων που αφορούν την επίδοση των επιμέρους τελεστών που το απαρτίζουν. Η πειραματική αξιολόγηση του συστήματος μας στο TensorFlow και το PyTorch έδειξε ότι μπορεί να εκτιμήσει με ικανοποιητική ακρίβεια, στις περισσότερες περιπτώσεις, τους χρόνους εκτέλεσης της κατανεμημένης εκπαίδευσης των νευρωνικών δικτύων. Συγκεκριμένα, η διάμεση τιμή του σφάλματος πρόβλεψης διαμορφώθηκε περίπου στο 19%, ενώ, σε μεγάλο ποσοστό των πειραμάτων, το σφάλμα κυμάνθηκε σε επίπεδα χαμηλότερα του 10%.

Abstract

Artificial neural networks are a popular Machine Learning technique that loosely simulates biological neurons. They are fed with a great amount of data, used for their training, so as to accomplish a variety of tasks, such as recognising an object in a given image, or, even, diagnosing an ill patient. Meanwhile, the amount of data, which is constantly growing, has led to the development of numerous distributed systems for training neural networks, with the most popular being TensorFlow and PyTorch. However, the variety of available choices raises the important question of which framework is optimal for training each network. This can be answered by implementing and using appropriate schedulers that can provide information regarding their training, such as the estimated execution time.

In this diploma thesis, we implement a system which can predict the training time of neural networks in a distributed environment, through profiling the operators that constitute them. Our experimental evaluation of this system in both TensorFlow and PyTorch has proven that it can estimate with satisfactory accuracy, in most cases, the execution time of the neural-networks' distributed training. More specifically, the median of the error amounted to approximately 19%, while, in many cases, the error accounted for less than 10%.

Ευχαριστίες

Ολοκληρώνοντας τις σπουδές μου, θα ήθελα να ευχαριστήσω όλους τους ανθρώπους που μου προσέφεραν βοήθεια όλα αυτά τα χρόνια. Αρχικά, θα ήθελα να ευχαριστήσω θερμά τα μέλη του εργαστηρίου υπολογιστών που με καθοδήγησαν στην εκπόνηση της διπλωματικής μου εργασίας, και ιδιαίτερα τον Νικόδημο Προβατά χωρίς τον οποίο δεν θα είχα καταφέρει να την υλοποιήσω. Επίσης, θα ήθελα να ευχαριστήσω τον κύριο Νεκτάριο Κοζύρη ο οποίος, ως επιβλέπων καθηγητής, με καθοδηγούσε καθ΄ όλη τη διάρκεια της διπλωματικής εργασίας μου, αλλά θα ήθελα να εκφράσω τις ευχαριστίες μου και σε όλους τους καθηγητές που με οδήγησαν στο να αγαπήσω το αντικείμενο της Επιστήμης Υπολογιστών, και ιδιαίτερα τον κύριο Νικόλαο Παπασπύρου, χάρη στον οποίο αγάπησα ακόμα περισσότερο το αντικείμενο των γλωσσών προγραμματισμού. Τέλος, θα ήθελα να ευχαριστήσω την οικογένειά μου και όλους μου τους φίλους που στάθηκαν δίπλα μου στις πιο δύσκολες στιγμές, εκ των οποίων ευχαριστώ ιδιαίτερα τους συμφοιτητές μου Δήμητρα Νικητοπούλου και Ιάσονα Μαρμάνη που με βοήθησαν να φέρω εις πέρας τις σπουδές μου.

Περιεχόμενα

Λίστα Πινάκων

Λίστα Σχημάτων

Κεφάλαιο 1

Εισαγωγή

1.1 Κίνητρο

Η βαθιά μάθηση έχει αποκτήσει σημαντικό ρόλο στην δημιουργία πληθώρας εφαρμογών τα τελευταία χρόνια, λόγω της δυνατότητάς του να ξεπερνά τις αποδόσεις ποικίλων άλλων μεθόδων, αλλά και ανθρώπων, στην επίλυση προβλημάτων, με ένα παράδειγμα να αποτελεί η αναγνώριση κειμένου από φωτογραφίες [1]. Ωστόσο, η ταχεία ανάπτυξη και βελτίωση της βαθιάς μάθησης συνοδεύεται από κάποια κόστη, μεταξύ των οποίων είναι τόσο η αύξηση του υπολογιστικού χρόνου εκπαίδευσης των νευρωνικών δικτύων, που οφείλεται στην επέκταση του βάθους τους, όσο και η ανάγκη τροφοδότησής του με ένα σύνολο δεδομένων ολοένα αυξανόμενου μεγέθους. Για να διευθετήσουμε αυτά τα θέματα, λοιπόν, είναι απαραίτητη η επιλογή λογισμικού που να μπορεί να πραγματοποιήσει την διαδικασία εκπαίδευσης στο συντομότερο δυνατό χρονικό διάστημα, και να μπορεί να διαχειριστεί, παράλληλα, έναν τεράστιο όγκο δεδομένων. Στην επίτευξη του παραπάνω στόχου δύναται να βοηθήσει σημαντικά και η εκτέλεση του λογισμικού αυτού σε κατανεμημένο περιβάλλον, ώστε να επιταχυνθεί ακόμα περισσότερο η διαδικασία και να αξιοποιηθούν αποτελεσματικότερα τα δεδομένα μεγάλης κλίμακας.

Τα διαθέσιμα εργαλεία με τα οποία μπορούμε να κατασκευάσουμε ένα νευρωνικό δίκτυο και να το εκπαιδεύσουμε, μεταξύ πολλαπλών πόρων, είναι πολυάριθμα. Μερικά παραδείγματα αποτελούν το TensorFlow [2] και το PyTorch [3], από τα πιο δημοφιλή συστήματα δημιουργίας εφαρμογών Μηχανικής Μάθησης, ικανά να εκτελεστούν σε κατανεμημένα συστήματα με στόχο να μοιράσουν τον φόρτο τους. Τις τεχνικές με τις οποίες το καταφέρνουν τις συνιστούν η παραλληλοποίηση δεδομένων, ο διαμοιρασμός, δηλαδή, του συνόλου δεδομένων μεταξύ υπολογιστικών πόρων, και η παραλληλοποίηση μοντέλου, που ισοδυναμεί με την αντιστοίχηση λειτουργιών με πόρους.

Η πληθώρα ύπαρξης διαφορετικών εργαλείων οδηγεί πολλές φορές στον προβληματισμό σχετικά με την επιλογή του κατάλληλου εργαλείου. Σε αντίστοιχες

περιπτώσεις κατασκευάζονται χρονοδρομολογητές, οι οποίοι λαμβάνουν υπόψιν ένα σύνολο πληροφοριών για την επερχόμενη εκτέλεση ενός προγράμματος στα διαθέσιμα συστήματα, και αποφασίζουν σε πιο σύστημα είναι προτιμότερο να γίνει η εκτέλεση. Τέτοιες πληροφορίες συνήθως συμπεριλαμβάνουν, μεταξύ άλλων, τον χρόνο εκτέλεσης και το κόστος εκτέλεσης (είτε σε πόρους είτε σε χρηματικές μονάδες).

Στην παρούσα διπλωματική εργασία επιδιώκεται η κατασκευή ενός συστήματος πρόβλεψης του χρόνου εκτέλεσης διαφόρων συστημάτων εκπαίδευσης νευρωνικών δικτύων. Για να επιτευχθεί η σωστή λειτουργία του προβλεπτικού συστήματος, γίνεται αρχικά πειραματική αποτίμηση της επίδοσης τελεστών σε κατανεμημένα περιβάλλοντα διαφόρων μεγεθών, και, στη συνέχεια, μέσω αυτών, κατασκευάζεται ένας προβλεπτικός μηχανισμός για την εκτίμηση του χρόνου εκτέλεσης.

1.2 Δομή

Η παρούσα εργασία ξεκινά με την παράθεση πληροφοριών σχετικά με μοντέλα παλινδρόμησης (κεφάλαιο 2), καθώς καταλήγουμε στην επιλογή εκείνου με το βέλτιστο σκορ ως μηχανισμό πρόβλεψης του χρόνου εκπαίδευσης ενός νευρωνικού δικτύου. Έπειτα, αναλύουμε στο κεφάλαιο 3 την δομή και μαθηματική υπόσταση που συνιστούν ένα τροφοδοτικό δίκτυο, σε συνδυασμό με τον αλγόριθμο που χρησιμοποιείται για την εκπαίδευσή τους, ενώ στο αμέσως επόμενο κεφάλαιο (4) παρουσιάζουμε τα δομικά συστατικά των συνελικτικών νευρωνικών δικτύων, με τα οποία και ασχολούμαστε στο υπόλοιπο της εργασίας μας. Εφόσον είναι απαραίτητα κάποια εργαλεία που να επιτρέπουν την υλοποίηση των συνελικτικών νευρωνικών δικτύων, εξηγούμε στο κεφάλαιο 5 την φιλοσοφία πίσω από τα συστήματα TensorFlow και PyTorch, που αποτελούν τη βάση της αρχιτεκτονικής του συστήματός μας, την οποία και αναλύουμε στο κεφάλαιο 6. Τέλος, παραθέτουμε στο κεφάλαιο 7 τα αποτελέσματα, αφενός, της συλλογής των μετρήσεων που πραγματοποιήσαμε, και, αφετέρου, της αξιολόγησης του προβλεπτή, πριν εξηγήσουμε στο κεφάλαιο 8 μία σχετική εργασία που έχει γίνει πάνω στο θέμα της πρόβλεψης του υπολογιστικού χρόνου των νευρωνικών δικτύων, ολοκληρώνοντας στο κεφάλαιο 9 με κάποια συμπεράσματα και πιθανές επεκτάσεις του μοντέλου μας.

Κεφάλαιο 2

Προβλήματα Παλινδρόμησης

2.1 Περιγραφή

Για την επίλυση προβλημάτων αναζήτησης και εντοπισμού προτύπων σε ένα σύνολο δεδομένων χρησιμοποιούνται ευρέως μέθοδοι από τη Μηχανική Μάθηση και Αναγνώριση Προτύπων [4]. Ένα τέτοιο πρόβλημα αποτελεί, για παράδειγμα, η αναγνώριση χειρόγραφων ψηφίων από εικόνες. Για να κατασκευάσουμε ένα μοντέλο μηχανικής μάθησης που να το υλοποιεί, θα χρειαστούμε ως είσοδο ένα σύνολο από δεδομένα που, αφενός, περιλαμβάνουν εικόνες από χειρόγραφα ψηφία και, αφετέρου, μία "ετικέτα" που να προσδιορίζει το ψηφίο στο οποίο, πράγματι, αντιστοιχεί. Το σύνολο από τις εικόνες θα το ονομάζουμε σύνολο εκπαίδευσης, ενώ τα ψηφία που αντιπροσωπεύουν, δηλαδή οι κατηγορίες στις οποίες ανήκουν τα στοιχεία του συνόλου εκπαίδευσης, θα τα ονομάζουμε τιμές-στόχος. Προκειμένου το τελικό μοντέλο να μπορεί να προβλέψει, ύστερα, το ψηφίο στο οποίο αντιστοιχεί μία εικόνα, θα πρέπει πρώτα να περάσει από τη διαδικασία της εκπαίδευσης με είσοδο το σύνολο εκπαίδευσης και τις τιμές-στόχους.

Η εφαρμογή των αλγορίθμων αυτών που λαμβάνουν ως είσοδο τα δεδομένα εκπαίδευσης μαζί με το διάνυσμα-στόχο του ονομάζεται επιβλεπόμενη μάθηση. Μερικά προβλήματα, όπως το παράδειγμα με την αναγνώριση ψηφίων, στοχεύουν στην κατάταξη των δεδομένων εισόδου σε μία από τις πεπερασμένου πλήθους, διακριτές κατηγορίες και ονομάζονται προβλήματα ταξινόμησης. Αν, ωστόσο, η επιθυμητή έξοδος του μοντέλου αποτελείται από μία ή περισσότερες συνεχείς μεταβλητές τότε έχουμε **πρόβλημα παλινδρόμησης** (**regression problem**).

Αναλυτικότερα, η παλινδρόμηση είναι μια στατιστική τεχνική αναζήτησης της συσχέτισης μίας εξαρτημένης μεταβλητής από μία ή περισσότερες ανεξάρτητες μεταβλητές. Δοσμένου, λοιπόν, ενός συνόλου N παρατηρήσεων $\boldsymbol{x}_n$, με $n = 1, 2, \ldots, N$, της ανεξάρτητης μεταβλητής $\boldsymbol{x}$ μαζί με τις αντίστοιχες επιθυμητές τιμές y_n της εξαρτημένης μεταβλητής $y_{\boldsymbol{x}}$, η τεχνική αυτή στοχεύει στην κατασκευή ενός μοντέλου που να προβλέπει τις τιμές της $y_{\boldsymbol{x}}$ για μια νέα τιμή του $\boldsymbol{x}$ με τέτοιο τρόπο ώστε να ελαχιστοποιεί την αναμενόμενη τιμή μίας κατάλληλα επιλεγμένης

συνάρτησης κόστους. Με άλλα λόγια, η παλινδρόμηση κατασκευάζει μία συνάρτηση $\hat{y}(\boldsymbol{x})$ η οποία για νέες τιμές της μεταβλητής $\boldsymbol{x}$ δίνει ως έξοδο μία πρόβλεψη για τις αντίστοιχες τιμές της $y_{\boldsymbol{x}}$. Προϋποθέτει, ωστόσο, ότι τα σχετικά δεδομένα ταιριάζουν με κάποια γνωστά είδη συναρτήσεων, ώστε να καθορίσει την καλύτερη συνάρτηση αυτού του είδους που να τα μοντελοποιεί. Η πιο συνηθισμένη συνάρτηση κόστους, έστω l, είναι η συνάρτηση τετραγωνικού σφάλματος:

$$l_{squared}\left(\boldsymbol{x}, y_{\boldsymbol{x}}, \hat{y}\right) \coloneqq \left(y_{\boldsymbol{x}} - \hat{y}(\boldsymbol{x})\right)^2 \tag{2.1}$$

2.2 Γραμμική παλινδρόμηση

Μία τεχνική παλινδρόμησης είναι η γραμμική [5], κατά την οποία η μεταβλητή $y_{\boldsymbol{x}}$ αναμένεται να είναι γραμμικός συνδυασμός των παραμέτρων $\boldsymbol{w} = \begin{bmatrix} w_0 & w_1 & w_2 & \cdots & w_P \end{bmatrix}^T$ του μοντέλου, με είσοδο την P-διαστάσεων μεταβλητή $\boldsymbol{x} = \begin{bmatrix} x_1 & x_2 & \cdots & x_P \end{bmatrix}^T$. Η πρόβλεψη, δηλαδή, του μοντέλου θα είναι η συνάρτηση:

$$\hat{y}(\boldsymbol{x}) = w_0 + w_1 x_1 + \cdots + w_P x_P \tag{2.2}$$

Αν το σφάλμα μεταξύ της πρόβλεψης $\hat{y}(\boldsymbol{x})$ και της αντίστοιχης τιμής της μεταβλητής $y_{\boldsymbol{x}}$ το συμβολίσουμε με ϵ, τότε θα ισχύει:

$$y_{\boldsymbol{x}} = \hat{y}(\boldsymbol{x}) + \epsilon \tag{2.3}$$

ή

$$y_{\boldsymbol{x}} = w_0 + w_1 x_1 + \cdots + w_P x_P + \epsilon \tag{2.4}$$

2.2.1 Μέθοδος Ελάχιστων Τετραγώνων

Έστω ότι για τις τιμές $\boldsymbol{x_i} = \begin{bmatrix} x_{i,1} & x_{i,2} & \cdots & x_{i,P} \end{bmatrix}^T$, με $i = 1, 2, \ldots, N$, της ανεξάρτητης μεταβλητής $\boldsymbol{x} = \begin{bmatrix} x_1 & x_2 & \cdots & x_P \end{bmatrix}^T$ έχουμε λάβει τις τιμές $y_1, y_2, \ldots, y_N$ της εξαρτημένης μεταβλητής $y_{\boldsymbol{x}}$. Τότε, από την (2.4), θα ισχύει:

$$y_i = w_0 + w_1 x_{i,1} + w_2 x_{i,2} + \cdots + w_P x_{i,P} + \epsilon_i, \text{ για } i = 1, 2, \ldots, N \tag{2.5}$$

και σε μορφή πίνακα

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} = \begin{bmatrix} x_{1,0} & x_{1,1} & x_{1,2} & \cdots & x_{1,p} \\ x_{2,0} & x_{2,1} & x_{2,2} & \cdots & x_{2,p} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{n,0} & x_{n,1} & x_{n,2} & \cdots & x_{n,p} \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_P \end{bmatrix} + \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_N \end{bmatrix} \tag{2.6}$$

ή

$$Y = X\boldsymbol{w} + \boldsymbol{\epsilon} \tag{2.7}$$

με $x_{1,0} = x_{2,0} = \cdots = x_{n,0} = 1$.

Η **μέθοδος των ελαχίστων τετραγώνων (ordinary least squares)** [6] στοχεύει στην ελαχιστοποίηση του τετραγωνικού σφάλματος, δηλαδή του $\sum_i {\epsilon_i}^2$, ως προς την παράμετρο $\boldsymbol{w}$. Ισοδύναμα, επιλύει το πρόβλημα:

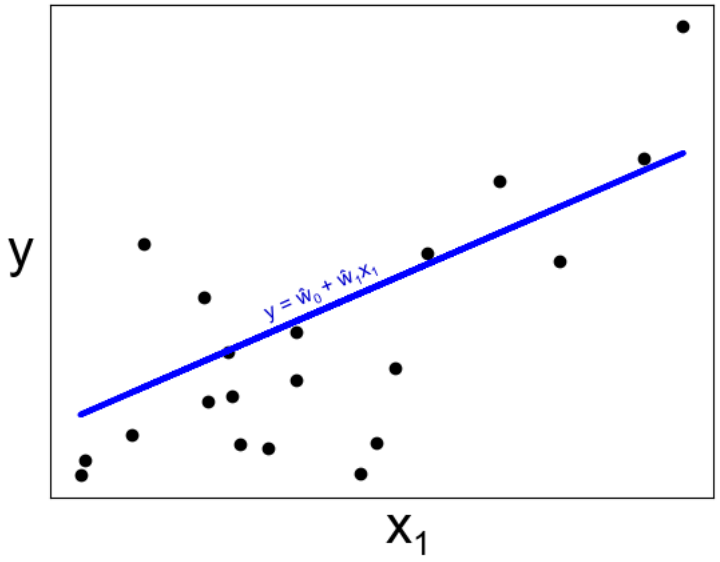
$$\min_{\boldsymbol{w}} \|X\boldsymbol{w} - Y\|^2 \tag{2.8}$$

όπου, για ένα διάνυσμα-στήλη $\boldsymbol{v} = \begin{bmatrix} v_1 & v_2 & \cdots & v_D \end{bmatrix}^T$, $\|\boldsymbol{v}\| = \sum_{i=1}^{D} \sqrt{v_i^2}$, και δίνει την εξής εκτίμηση για τις παραμέτρους του μοντέλου:

$$\hat{\boldsymbol{w}} = \left(X^T X\right)^- X^T Y \tag{2.9}$$

όπου με A^- συμβολίζουμε κάποιον γενικευμένο αντίστροφο του πίνακα A.

Στην περίπτωση που έχουμε μία ανεξάρτητη μεταβλητή x_1, η μέθοδος ελαχίστων τετραγώνων υπολογίζει την βέλτιστη ευθεία $y = \hat{w}_0 + \hat{w}_1 x_1$ που ελαχιστοποιεί το τετραγωνικό σφάλμα μεταξύ των σημείων της ευθείας και των παρατηρήσεων. Η οπτικοποίηση της ευθείας και των παρατηρήσεων φαίνεται στην εικόνα 2.1.



Εικόνα 2.1: Η ευθεία $y = \hat{w}_0 + \hat{w}_1 x_1$ μαζί με τις τιμές του πειράματος [6].

Η μέθοδος αυτή προϋποθέτει ότι οι μεταβλητές εισόδου $x_1, x_2, \ldots, x_P$ είναι γραμμικά ανεξάρτητες. Εάν, ωστόσο, είναι (σχεδόν) γραμμικά εξαρτημένες, η μέθοδος μπορεί να οδηγήσει σε εκτίμηση $\hat{\boldsymbol{w}}$ που να διαφέρει σημαντικά από τις πραγματικές τιμές των παραμέτρων $\boldsymbol{w}$.

Προκειμένου να υπολογίσουμε τις παραπάνω παραμέτρους, είναι απαραίτητος κάποιος αποδοτικός αλγόριθμος, ένας εκ των οποίων είναι ο αλγόριθμος Singular Value Decomposition, με πολυπλοκότητα $O(NP^2)$, εάν ο πίνακας X έχει μέγεθος $N \times P$.

2.2.2 Ridge

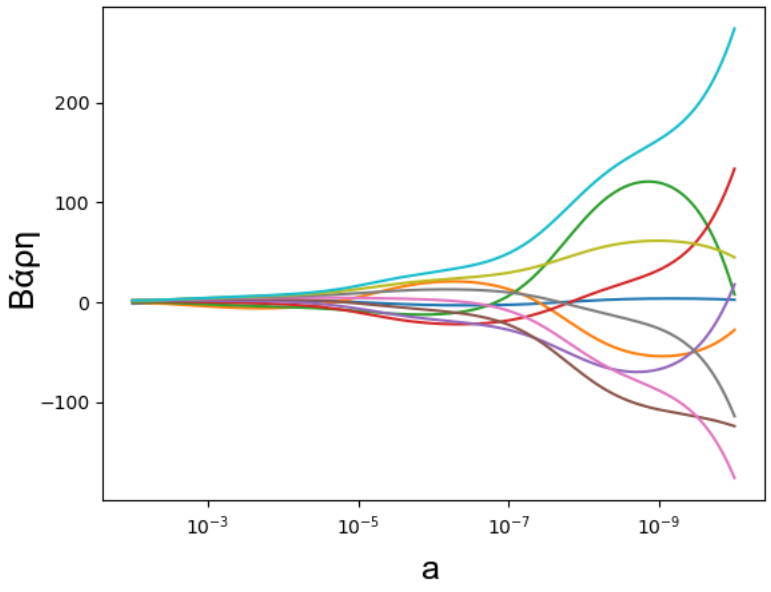
Μια άλλη μέθοδος γραμμικής παλινδρόμησης ονομάζεται **Ridge Regression** και αποσκοπεί στην επίλυση των προβλημάτων που εμφανίζονται με τη μέθοδο ελαχίστων τετραγώνων όταν υπάρχει συγγραμμικότητα (collinearity) στα δεδομένα εισόδου, όπως αναφέρθηκε παραπάνω. Ουσιαστικά, το πρόβλημα που επιλύει [7] είναι μία παραλλαγή της προβλήματος των ελάχιστων τετραγώνων, καθώς εισάγει μία ποινή στην τιμή των παραμέτρων $\boldsymbol{w}$:

$$\min_{\boldsymbol{w}} \left(\|X\boldsymbol{w} - Y\|^2 + a\|\boldsymbol{w}\|^2 \right) \text{, όπου } a > 0 \tag{2.10}$$

με λύση την εκτίμηση:

$$\hat{\boldsymbol{w}} = \left(X^T X + a\boldsymbol{I}\right)^{-1} X^T Y \tag{2.11}$$

Όσο μεγαλύτερη είναι η παράμετρος a τόσο μειώνεται η διακύμανση των παραμέτρων w_i, γεγονός το οποίο δίνει λύση στην περίπτωση που τα w_i εμφανίζουν σημαντικές διαφορές ακόμα και για μικρές μεταβολές στις τιμές της μεταβλητής $y_{\boldsymbol{x}}$. Από την άλλη πλευρά, όσο τείνει η a στο 0, τόσο η μέθοδος αυτή προσεγγίζει την μέθοδο ελαχίστων τετραγώνων. Στην πραγματικότητα, η παράμετρος a επιλέγεται με τέτοιον τρόπο ώστε να διατηρείται μια ισορροπία μεταξύ των δύο αυτών περιπτώσεων. Μια οπτικοποίηση του τρόπου που μεταβάλλονται τα βάρη της μεθόδου Ridge συναρτήσει της a φαίνεται στην εικόνα 2.2.



Εικόνα 2.2: Βάρη της μεθόδου Ridge συναρτήσει της παραμέτρου a [8].

2.2.3 LASSO και Elastic-Net

Η μέθοδος **LASSO (Least Absolute Shrinkage and Selection Operator)** [9] είναι μία ακόμη μέθοδος γραμμικής παλινδρόμησης, η οποία περιλαμβάνει και μηδενικές τιμές στην εκτίμηση των παραμέτρων $\boldsymbol{w}$ που υπολογίζει, μειώνοντας το πλήθος των χαρακτηριστικών ($\boldsymbol{x}_i$) από τα οποία εξαρτάται η τελική λύση. Αν ο πίνακας X έχει μέγεθος $N \times P$, το πρόβλημα που επιλύει η μέθοδος LASSO, είναι:

$$\min_{\boldsymbol{w}} \left(\frac{1}{2N} \|X\boldsymbol{w} - NY\|^2 + a\|\boldsymbol{w}\|_1 \right) \tag{2.12}$$

όπου $a > 0$ και $\|\boldsymbol{w}\|_1 = \sum_i |w_i|$.

Η μέθοδος **Elastic-Net** είναι ένας συνδυασμός των Ridge και LASSO [10]. Λειτουργεί πάνω σε ένα σύνολο με λίγες μη μηδενικές παραμέτρους, διατηρώντας όμως την σταθερότητα του Ridge, και είναι χρήσιμη όταν υπάρχει συσχέτιση σε πολλά από τα χαρακτηριστικά εισόδου.

Το πρόβλημα που επιλύει το Elastic-Net είναι το:

$$\min_{\boldsymbol{w}} \left(\frac{1}{2N} \|X\boldsymbol{w} - NY\|^2 + a\rho\|\boldsymbol{w}\|_1 + \frac{a(1-\rho)}{2}\|\boldsymbol{w}\|^2 \right) \tag{2.13}$$

όπου $0 \leq \rho \leq 1$.

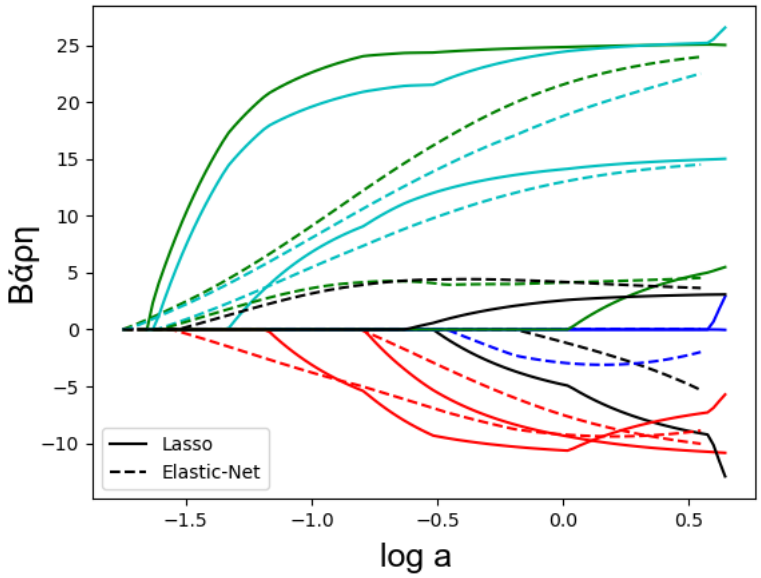
Ένας αλγόριθμος που υπολογίζει τις εκτιμήσεις $\hat{\boldsymbol{w}}$ και για τις δύο μεθόδους είναι ο Coordinate descent [11]. Μπορούμε να δούμε τον τρόπο που μεταβάλλονται, με την εφαρμογή του αλγορίθμου, οι παράμετροι του μοντέλου στην εικόνα 2.3.

2.3 K-Κοντινότεροι Γείτονες

Ο αλγόριθμος **k-κοντινότεροι γείτονες (k-Nearest Neighbors ή k-NN)** είναι μία μη-παραμετρική μέθοδος που χρησιμοποιείται στην ταξινόμηση και παλινδρόμηση [13] [14], δηλαδή δε θεωρεί ότι τα δοσμένα δεδομένα ακολουθούν κάποια κατανομή πιθανότητας. Ο αλγόριθμος k-NN περιλαμβάνει τον υπολογισμό του μέσου όρου των τιμών-στόχων, των k κοντινότερων γειτόνων, όπου k είναι μια σταθερά που ορίζεται κατά την εκτέλεση του αλγορίθμου. Κοντινότεροι γείτονες θεωρούνται οι ανεξάρτητες μεταβλητές του πειράματος που έχουν τη μικρότερη απόσταση, η οποία μπορεί να είναι μία εκ των εξής συναρτήσεων [15]:

- Ευκλείδεια: $\sqrt{\sum_{i=1}^{k} (x_i - y_i)^2}$
- Manhattan: $\sum_{i=1}^{k} |x_i - y_i|$
- Minkowski: $\left(\sum_{i=1}^{k} (|x_i - y_i|)^q\right)^{1/q}$

Μία άλλη προσέγγιση του αλγορίθμου περιλαμβάνει τον υπολογισμού του σταθμισμένου μέσου των τιμών-στόχων, των k κοντινότερων γειτόνων, πολλαπλασιασμένο με το αντίστροφο της απόστασής τους.



Εικόνα 2.3: Βάρη των μεθόδων Lasso and Elastic-Net συναρτήσει της παραμέτρου a [12].

Συνοπτικά, ο αλγόριθμος ακολουθεί τα εξής βήματα:

1. Υπολογίζει την απόσταση των μεταβλητών.
2. Ταξινομεί τις τιμές-στόχους με βάσει τις αποστάσεις.
3. Υπολογίζει, με έναν ευριστικό τρόπο, τον βέλτιστο αριθμό k για τους κοντινότερους γείτονες, βασισμένο στο ελάχιστο τετραγωνικό σφάλμα.
4. Υπολογίζει τον (σταθμισμένο) μέσο όρο των τιμών που αντιστοιχούν στους k κοντινότερους γείτονες.

2.4 Διανύσματα Υποστήριξης

Η μέθοδος παλινδρόμησης με **Διανύσματα Υποστήριξης (Support Vector Regression - SVR)** διατηρεί τις ίδιες αρχές με την μέθοδο ταξινόμησης με μηχανές διανυσμάτων υποστήριξης (Support Vector Machines - SVM), με κύρια διαφορά την επιλογή ενός πραγματικού αριθμού ϵ ως το περιθώριο ανοχής, ώστε να προσεγγίζει το αποτέλεσμα της SVM μεθόδου.

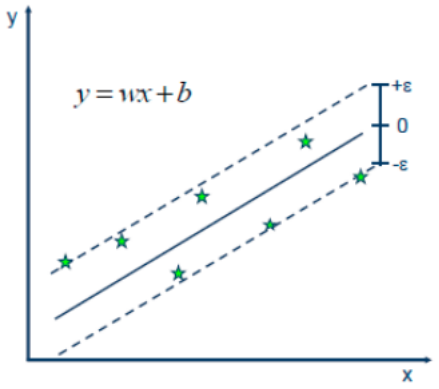
Η βασική ιδέα της υλοποίησης του αλγορίθμου SVR είναι η εξής [16]: Έστω ότι έχουμε ένα σύνολο εκπαίδευσης $\{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_l, y_l)\}$. Σκοπός της μεθόδου SVR (πιο συγκεκριμένα της μεθόδου ϵ - SV regression) είναι η εύρεση μίας συνάρτησης $f(\boldsymbol{x})$ που να έχει απόκλιση το πολύ ϵ από τις πραγματικές τιμές y_i, για όλα τα δεδομένα

εκπαίδευσης, ενώ η συνάρτηση είναι όσο τον δυνατόν περισσότερο «επίπεδη» (flat functions) [17], δηλαδή οι παράγωγοί της «εξαφανίζονται» σε ένα δεδομένο σημείο x_0.

Στην περίπτωση της γραμμικής συνάρτησης $f(\boldsymbol{x}) = \boldsymbol{w}\boldsymbol{x} + b$, η αναζήτηση επίπεδης συνάρτησης σημαίνει εύρεση μικρής τιμής για το $\boldsymbol{w}$, και ένας τρόπος για να το πετύχουμε είναι να ελαχιστοποιήσουμε την νόρμα $\|\boldsymbol{w}\|$. Ισοδύναμα, το πρόβλημα γράφεται:

$$\min\left(\frac{1}{2}\|\boldsymbol{w}\|^2\right), \text{ υπό την προϋπόθεση ότι } -\epsilon \leq y_i - \boldsymbol{w}\boldsymbol{x}_i - b \leq \epsilon$$

Μία οπτικοποίηση του προβλήματος για μονοδιάστατες τιμές x_i φαίνεται στην εικόνα 2.4.



Εικόνα 2.4: Γραμμική παλινδρόμηση με διανύσματα υποστήριξης [18].

Σε περίπτωση που το πρόβλημα δεν επιλύεται σε αυτή τη μορφή, μπορούμε να εισάγουμε δύο νέες παραμέτρους ξ και ξ^* ώστε να χαλαρώσουμε τους περιορισμούς. Τότε, το πρόβλημα, για κάποια σταθερά $C > 0$, θα είναι το:

$$\min\left(\frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{l}(\xi + \xi^*)\right)$$

υπό την προϋπόθεση ότι:

$$-\epsilon - \xi^* \leq y_i - \boldsymbol{w}\boldsymbol{x}_i - b \leq \epsilon + \xi$$

και

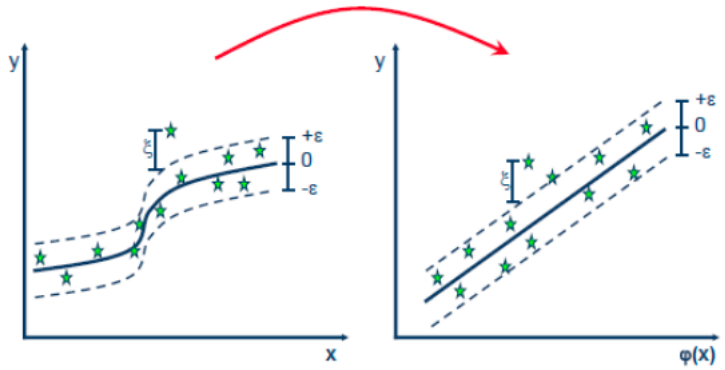
$$\xi, \xi^* \geq 0$$

Η λύση του προβλήματος θα έχει τη μορφή:

$$\boldsymbol{w} = \sum_{i=1}^{l}(a_i - a_i^*)\,\boldsymbol{x}_i$$

Δηλαδή, όπως υποδηλώνει και το όνομα της μεθόδου, τα βάρη του προβλήματος είναι ένας γραμμικός συνδυασμός των διανυσμάτων $\boldsymbol{x}_i$.

Το πρόβλημα γενικεύεται και για μη-γραμμικές συναρτήσεις, εφαρμόζοντας μία συνάρτηση μετασχηματισμού $\phi(\boldsymbol{x})$ στα δεδομένα, όπως φαίνεται στην εικόνα 2.5, με τη συνάρτηση να είναι της μορφής $f(\boldsymbol{x}) = \boldsymbol{w}\phi(\boldsymbol{x}) + b$. Η λύση τότε του προβλήματος



Εικόνα 2.5: Μη γραμμική παλινδρόμηση με διανύσματα υποστήριξης [18].

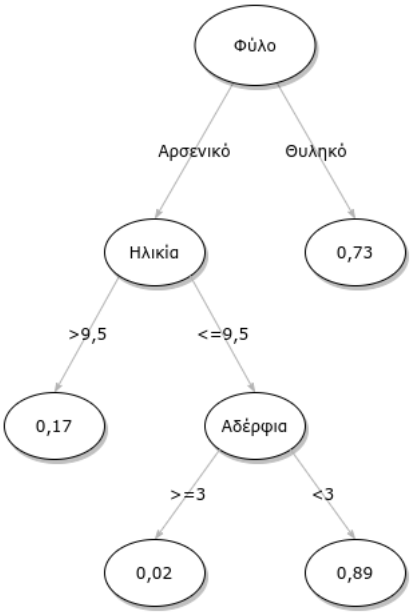
θα έχει τη μορφή:

$$\boldsymbol{w} = \sum_{i=1}^{l} (a_i - a_i^*)\, \phi(\boldsymbol{x}_i)$$

2.5 Δέντρα Απόφασης

Η παλινδρόμηση με **δέντρα απόφασης (Decision Tree Regression)** [19] [20] περιλαμβάνει την διάσπαση του συνόλου δεδομένων σε υποσύνολα, κατασκευάζοντας παράλληλα μία δομή δεδομένων τύπου δέντρου . Το τελικό αποτέλεσμα είναι ένα δέντρο που αποτελείται από κόμβους απόφασης και κόμβους-φύλλα. Κάθε εσωτερικός κόμβους διαθέτει μία ετικέτα και δύο ή περισσότερα κλαδιά, τα οποία με τη σειρά τους καταλήγουν είτε σε άλλους κόμβους απόφασης είτε σε φύλλα, τα οποία χαρακτηρίζονται από μία τιμή. Οι ετικέτες αντιπροσωπεύουν μία μεταβλητή του πειράματος, ενώ οι τιμές στα φύλλα αντιστοιχούν στην τιμή προς πρόβλεψη. Ένα παράδειγμα δέντρου απόφασης φαίνεται στην εικόνα 2.6, το οποίο αντιστοιχεί σε ένα μοντέλο που υπολογίζει την πιθανότητα επιβίωσης ενός επιβάτη του Τιτανικού.

Ο αλγόριθμος που υλοποιεί την δημιουργία ενός τέτοιου δέντρου ονομάζεται CART (Classification and Regression Trees), και είναι παρόμοιος με τον αλγόριθμο C4.5, ο οποίος, με τη σειρά του, είναι μία τροποποίηση του ID3 (Iterative Dichotomiser 3) [22], αλλά διαφέρει στο ότι πραγματοποιεί παλινδρόμηση. Ο CART πραγματοποιεί άπληστη αναζήτηση από πάνω προς τα κάτω, και, ξεκινώντας από την



Εικόνα 2.6: Δέντρο απόφασης [21].

ρίζα, χωρίζει τα δεδομένα σε υποσύνολα με «παρόμοιες» τιμές, δηλαδή τιμές με τη μικρότερη δυνατή τυπική απόκλιση.

2.6 Τυχαία Δάση

Η μέθοδος **τυχαίων δασών (Random Forests)** [23] είναι ένας συνδυαστικός τρόπος παλινδρόμησης, ο οποίος περιλαμβάνει την δημιουργία πολλαπλών δέντρων απόφασης κατά την διαδικασία της εκπαίδευσης, επιλέγοντας ως την έξοδό του την μέση τιμή όλων των δέντρων. Η μέθοδος αυτή διορθώνει την τάση των απλών δέντρων απόφασης να κάνουν overfitting στα δεδομένα εκπαίδευσης, και, γενικά, έχουν καλύτερη απόδοση.

Ο αλγόριθμος που υλοποιεί την παλινδρόμηση τυχαίων δασών [24] περιλαμβάνει την δημιουργία του κάθε δέντρου απόφασης με την επιλογή ενός τυχαίου δείγματος από το σύνολο δεδομένων, ενώ κατά τον διαχωρισμό των δεδομένων σε κάποιο κόμβο του δέντρου, ο καλύτερος διαχωρισμός θα προκύπτει από ένα τυχαίο υποσύνολο των χαρακτηριστικών εισόδου.

2.7 Μετρικές Αξιολόγησης Μοντέλων

Είναι απαραίτητο, ύστερα από την εκπαίδευση ενός μοντέλου παλινδρόμησης πάνω σε κάποια δεδομένα, να μπορούμε να αξιολογήσουμε το αποτέλεσμα.

Χρησιμοποιούμε, λοιπόν, κάποιες μετρικές, δηλαδή συναρτήσεις (γνωστές και ως **συναρτήσεις κόστους**), οι οποίες λαμβάνουν ως είσοδο τις προβλέψεις του επιλεγμένου μοντέλου παλινδρόμησης, αλλά και τις πραγματικές τιμές του πειράματος, για έναν κοινό συνδυασμό από τιμές των μεταβλητών εισόδου, ενώ η έξοδος των συναρτήσεων αυτών είναι μια τιμή που περιγράφει το πόσο αποτελεσματικό είναι το μοντέλο.

Η διαδικασία υπολογισμού της τιμής μίας μετρικής πραγματοποιείται πριν και μετά την εκπαίδευση. Συγκεκριμένα, το σύνολο δεδομένων εκπαίδευσης διαχωρίζεται τυχαία σε δύο υποσύνολα, με το πρώτο να χρησιμοποιείται για την εκπαίδευση, και το δεύτερο να συμβάλει στην αξιολόγηση του μοντέλου, καθώς αποτελεί είσοδο της μετρικής.

Μία τέτοια μετρική είναι ο **συντελεστής προσδιορισμού (Coefficient of Determination)** [25] [26], και συμβολίζεται με R^2. Αν $\hat{y}_i$ είναι οι τιμές που προβλέπει το μοντέλο παλινδρόμησης, y_i οι αντίστοιχες πραγματικές τιμές του πειράματος, και $E(y) = \frac{1}{n}\sum_{i=1}^{n} y_i$ η μέση τιμή των τιμών y_i, τότε:

$$R^2 = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - E(y))^2} \tag{2.14}$$

Το καλύτερο δυνατό σκορ είναι η τιμή 1, ενώ οι τιμές μπορεί να είναι και αρνητικές, γεγονός που μπορεί να οφείλεται στο ότι το μοντέλο, με τα δεδομένα που επιλέχθηκαν, δεν είναι αποδοτικό ως προς τις προβλέψεις του.

Μία ακόμα μετρική είναι το **μέσο τετραγωνικό σφάλμα (Mean Squared Error - MSE)** [27], το οποίο ορίζεται ως εξής:

$$MSE = \frac{1}{n}\sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{2.15}$$

Ουσιαστικά, πρόκειται για τον μέσο όρο του τετραγωνικού σφάλματος μεταξύ των τιμών της πρόβλεψης και των πραγματικών τιμών. Η μετρική MSE είναι πάντα μη αρνητική, ενώ όσο πιο κοντά είναι στο μηδέν τόσο καλύτερη είναι η ποιότητα του μοντέλου πρόβλεψης. Επίσης, σε πολλές περιπτώσεις, για να υπάρχει καλύτερη κατανόηση των τιμών του MSE, είναι βοηθητικό να λάβουμε τη ρίζα του, οπότε και θα έχουμε τη μετρική που ονομάζεται **ρίζα του μέσου τετραγωνικού σφάλματος (Root Mean Squared Error - RMSE)**, δηλαδή ορίζεται ως:

$$RMSE = \sqrt{MSE} = \frac{1}{n}\sum_{i=1}^{n} \sqrt{(y_i - \hat{y}_i)^2} \tag{2.16}$$

Το μέσο τετραγωνικό σφάλμα δεν έχει τις ίδιες μονάδες μέτρησης με τις τιμές του πειράματος, με αποτέλεσμα να μην είναι φανερό το πόσο μεγάλη είναι η απόκλιση των τιμών της πρόβλεψης από τις πραγματικές. Η λύση σε αυτό το πρόβλημα δίνεται από την μετρική RMSE.

Κεφάλαιο 3

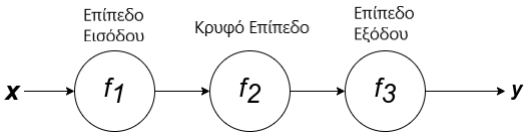
Τροφοδοτικά Νευρωνικά Δίκτυα

Τα **τροφοδοτικά (feedforward) νευρωνικά δίκτυα**, γνωστά και ως **multilayer perceptrons**, αποτελούν δομικό εργαλείο της μηχανικής μάθησης και χρησιμοποιούνται ως βάση για τη δημιουργία πολλών εμπορικών εφαρμογών. [28] Για παράδειγμα, τα συνελικτικά νευρωνικά δίκτυα, μία χρήση των οποίων είναι η αναγνώριση αντικειμένων σε φωτογραφίες, είναι μια ειδικευμένη μορφή των τροφοδοτικών δικτύων. Σκοπός των τροφοδοτικών νευρωνικών δικτύων είναι η προσέγγιση μίας συνάρτησης f^*. Συγκεκριμένα, αν $y = f^*(x)$ η αντιστοίχηση της εισόδου x στην κατηγορία y, τότε το δίκτυο, μέσω της αντιστοίχησης $y = f(x; \theta)$, «μαθαίνει» για ποιες παραμέτρους θ οδηγούμαστε σε μια καλύτερη προσέγγιση της συνάρτησης. Ο λόγος για τον οποίο ονομάζονται με αυτόν τον τρόπο, δηλαδή τροφοδοτικά, είναι ότι η πληροφορία ρέει από την είσοδο x του δικτύου, και περνάει μέσα από τους υπολογισμούς της συνάρτησης f που προσδιορίζει το δίκτυο, καταλήγοντας στην έξοδο y, χωρίς να υπάρχουν σύνδεσμοι ανατροφοδότησης, στους οποίους οι έξοδοι χρησιμοποιούνται ως είσοδοι του μοντέλου. Σε περίπτωση που τα δίκτυα αυτά αποκτήσουν συνδέσμους ανατροφοδότησης, τότε ονομάζονται **Ανατροφοδοτικά Νευρωνικά Δίκτυα (Recurrent Neural Networks - (RNN)**. Επομένως, η έννοια των τροφοδοτικών δικτύων, με τα οποία και ασχολούμαστε στην παρούσα διπλωματική εργασία, αποτελεί θεμελιώδες βήμα προς την κατασκευή των RNN.

3.1 Δομή

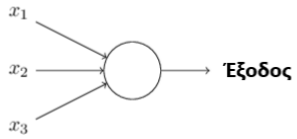
Τα τροφοδοτικά νευρωνικά δίκτυα είναι «δίκτυα» καθώς αναπαριστώνται από την σύνθεση συναρτήσεων, και αντιστοιχούν σε έναν κατευθυνόμενο ακυκλικό γράφο. Για παράδειγμα, εάν έχουμε ένα δίκτυο, με είσοδο x και έξοδο y, το οποίο αναπαριστάται από τις διαδοχικά συνδεδεμένες συναρτήσεις f_1, f_2, f_3, σχηματίζοντας

μια αλυσίδα (εικόνα 3.1), τότε θα ισχύει $y = f_3(f_2(f_1(\boldsymbol{x})))$. Κάθε μία από τις συναρτήσεις που αποτελούν το δίκτυο ονομάζονται **επίπεδα** (**layers**). Θεωρούμε, επίσης, ότι η συνάρτηση που λαμβάνει την είσοδο και η συνάρτηση που παράγει την έξοδο του δικτύου, αποτελούν το επίπεδο εισόδου και το επίπεδο εξόδου, αντίστοιχα, ενώ τα υπόλοιπα επίπεδα (μεταξύ επιπέδου εισόδου και επιπέδου εξόδου), έχει καθιερωθεί να ονομάζονται **κρυφά επίπεδα (hidden layers)**. **Βάθος (depth)** δικτύου ονομάζουμε το πλήθος των επιπέδων που το απαρτίζουν. Με βάση αυτήν την ορολογία έχει προκύψει και η έννοια των βαθιών νευρωνικών δικτύων (deep neural networks), που είναι, ουσιαστικά, νευρωνικά δίκτυα αποτελούμενα από ένα σχετικά μεγάλο πλήθος από επίπεδα.



Εικόνα 3.1: Ένα απλό νευρωνικό δίκτυο τριών επιπέδων.

Επιπλέον, τα δίκτυα αυτά τα ονομάζουμε «νευρωνικά» καθώς η δημιουργία τους έχει εμπνευστεί από την Νευροεπιστήμη, δεδομένου ότι κάθε επίπεδο ενός νευρωνικού δικτύου αποτελούνται από τους αντίστοιχους τεχνητούς **νευρώνες**.



Εικόνα 3.2: Perceptron [29].

Η πιο απλή μορφή τεχνητού νευρώνα αποτελεί το **perceptron** [29] (εικόνα 3.2), μια συνάρτηση που λαμβάνει ως είσοδο τις δυαδικές τιμές $x_1, x_2, \ldots, x_n$ και παράγει μία δυαδική έξοδο y με βάση την εξής συνθήκη:

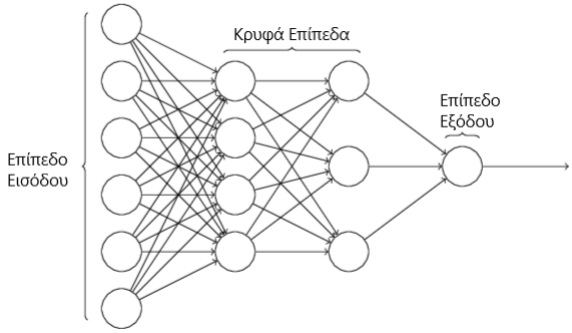
$$y = \begin{cases} 0, & \text{if } \sum_{i=1}^{n} w_i x_i \leq \mathit{threshold} \\ 1, & \text{if } \sum_{i=1}^{n} w_i x_i > \mathit{threshold} \end{cases}$$

ή, αλλιώς, σε απλούστερη μορφή:

$$y = \begin{cases} 0, & \text{if } w \cdot x + b \leq 0 \\ 1, & \text{if } w \cdot x + b > 0 \end{cases}$$

όπου $w = [w_1, w_2, \ldots, w_n]$ και $x = [x_1, x_2, \ldots, x_n]$, με w_i να είναι τα λεγόμενα **βάρη (weights)** των αντίστοιχων εισόδων, και b να είναι η **μεροληψία (bias)** του perceptron, όλα εκ των οποίων είναι πραγματικοί αριθμοί. Επομένως, τα επίπεδα ενός

νευρωνικού δικτύου δεν είναι παρά πολλαπλοί νευρώνες που λειτουργούν παράλληλα ώστε να παράξουν το καθένα μία έξοδο, η οποία, με την σειρά της, αποτελεί την είσοδο στο επόμενο διαδοχικό επίπεδο, με ένα τέτοιο παράδειγμα να φαίνεται στην εικόνα 3.3.



Εικόνα 3.3: Νευρωνικό δίκτυο ως αλληλουχία νευρώνων [29].

Κατά τη μάθηση του δικτύου, οι τιμές στα βάρη των νευρώνων αλλάζουν ώστε η έξοδος του δικτύου να ταξινομεί σωστά την είσοδο. Αυτό σημαίνει ότι, με τη χρήση των perceptron, μια μικρή αλλαγή σε ένα βάρος του δικτύου μπορεί να οδηγήσει στην αντιστροφή της εξόδου. Μπορούμε να αντιμετωπίσουμε αυτό το πρόβλημα με την χρήση των **σιγμοειδών νευρώνων (sigmoid neurons)**, οι οποίοι λειτουργούν παρόμοια με τα perceptron, όμως σαν είσοδο λαμβάνουν συνεχείς τιμές στο διάστημα $[0, 1]$ και παράγουν έξοδο $\sigma(wx + b)$ με τιμές από 0 έως 1, όπου σ είναι η σιγμοειδής συνάρτηση [30], και ορίζεται ως:

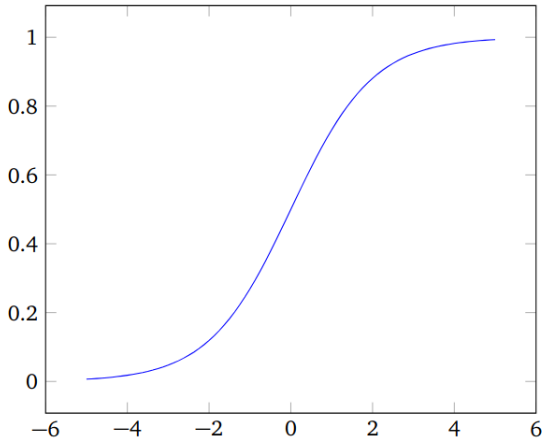
$$\sigma(z) \equiv \frac{1}{1 + e^{-z}}$$

Η γραφική της παράσταση φαίνεται στην εικόνα 3.4.

Στην πραγματικότητα, αυτό που ενδιαφέρει περισσότερο είναι το γεγονός ότι η σιγμοειδής συνάρτηση έχει «ομαλή» μορφή. Συγκεκριμένα, όπως αναφέρθηκε, μικρή αλλαγή στα βάρη και στη μεροληψία σημαίνει μικρή αλλαγή και στην έξοδο, με αποτέλεσμα το νευρωνικό να μπορεί να "κατανοήσει" καλύτερα την σχέση εισόδου-εξόδου. Μάλιστα, αντί για την συνάρτηση σ μπορεί να χρησιμοποιηθεί διαφορετική συνάρτηση με παρόμοιες ιδιότητες. Αυτές οι συναρτήσεις ονομάζονται **συναρτήσεις ενεργοποίησης (activation functions)**.

3.2 Εκπαίδευση

Ένα νευρωνικό δίκτυο, ουσιαστικά, υπολογίζει έναν μη-γραμμικό, παραμετρικό μετασχηματισμό $f_\theta : X \to Y$, όπου το θ αντιπροσωπεύει ένα σύνολο παραμέτρων, ή, αλλιώς, βαρών, με το X να είναι το σύνολο δεδομένων που θα χρησιμοποιηθεί για



Εικόνα 3.4: Σιγμοειδής συνάρτηση [29].

την εκπαίδευση του μοντέλου, ενώ ως Y θεωρούμε το σύνολο με τις «επιγραφές» ή κατηγορίες στις οποίες αντιστοιχεί κάθε στοιχείο του X. Προκειμένου να μπορέσει να πραγματοποιηθεί η διαδικασία της εκπαίδευσης (**training**), είναι απαραίτητος ένας αλγόριθμος εκπαίδευσης.

Σκοπός της εκπαίδευσης είναι να βρεθεί ένα σύνολο παραμέτρων θ^* που να ελαχιστοποιεί τη διαφορά μεταξύ του συνόλου Y, δηλαδή η αληθινή τιμή στην οποία αντιστοιχούν τα στοιχεία του X, με το σύνολο $\hat{Y}$, που είναι η έξοδος του νευρωνικού δικτύου με είσοδο το X:

$$\hat{Y} = f_\theta(X)$$

Συνήθως, για την μάθηση του δικτύου, χρησιμοποιείται μία παραλλαγή του αλγορίθμου **καθόδου κλίσεων (gradient descent)** [31], σε συνδυασμό με μία διαφορίσιμη συνάρτηση κόστους $C(\hat{Y}, Y)$, κατά τον οποίο ενημερώνουμε επαναληπτικά τις παραμέτρους του δικτύου με τον εξής τρόπο:

- Βήμα 1: $E = \frac{1}{N} \sum_{i=1}^{N} C(\hat{Y}_i, Y_i)$
- Βήμα 2: $\nabla\theta_k(t) = (\partial E / \partial \theta_k)(t)$
- Βήμα 3: $\theta_k(t+1) = \theta_k(t) - \alpha(t)\nabla\theta_k(t)$

Με $\theta_k(t)$ συμβολίζουμε την k-οστή παράμετρο στην επανάληψη t, με α το βήμα (ή ρυθμό μάθησης) και με N το πλήθος των δεδομένων εκπαίδευσης στο X. Ωστόσο, για να υπολογιστεί το μέγεθος $\nabla\theta_k$ υπολογίζονται πρώτα τα $\nabla C(\hat{Y}_i, Y_i)$ (σε κάθε δεδομένο εισόδου) και, στην συνέχεια, λαμβάνεται ο μέσος όρος τους ως εξής:

$$\nabla C(\hat{Y}_i, Y_i) = \frac{\partial C(\hat{Y}_i, Y_i)}{\partial \theta_k}, \tag{3.1}$$

$$\nabla \theta_k = \frac{1}{N} \sum_{i=1}^{N} \nabla C(\hat{Y}_i, Y_i) \tag{3.2}$$

Επομένως, για μεγάλο αριθμό δεδομένων εκπαίδευσης οι υπολογισμοί θα χρειαστούν, ενδεχομένως, πολύ χρόνο και η εκπαίδευση θα είναι αργή. [29] Προκειμένου να λυθεί το πρόβλημα αυτό, εφαρμόζεται ο αλγόριθμος **στοχαστικής καθόδου κλίσεων (stochastic gradient descent)** με βάση τον οποίο, αντί να χρησιμοποιείται ολόκληρο το σύνολο δεδομένων για τον υπολογισμό του κόστους, λαμβάνεται ένα μικρό μέρος τυχαία επιλεγμένων δεδομένων με αποτέλεσμα να προσεγγίζεται ικανοποιητικά η αληθινή τιμή του κόστους σε λιγότερο χρόνο. Έτσι, η εκπαίδευση πραγματοποιείται σε μικρότερα σύνολα δεδομένων και σε τόσα βήματα ώστε να έχει εφαρμοστεί ο αλγόριθμος σε όλα τα δεδομένα που δόθηκαν. Οι επαναλήψεις του αλγορίθμου σε ολόκληρο το σύνολο δεδομένων, δηλαδή η προσπέλαση όλων των συνόλων των **τεμαχίων (batch)** που απαρτίζουν το σύνολο των δεδομένων εκπαίδευσης, ονομάζονται **εποχές**.

Τελικά, εάν $m << N$ και το σύνολο των τυχαία επιλεγμένων δεδομένων στο βήμα t είναι το $\{(X_1^{(t)}, Y_1^{(t)}), (X_2^{(t)}, Y_2^{(t)}), \ldots, (X_m^{(t)}, Y_m^{(t)})\}$, το οποίο θα ονομάζεται mini-batch με μέγεθος m **(μέγεθος τεμαχίου - batch size)**, ο αλγόριθμος γίνεται:

- Βήμα 1: $\nabla C(\hat{Y_i}^{(t)}, Y_i^{(t)}) = \frac{\partial C(\hat{Y_i}^{(t)}, Y_i^{(t)})}{\partial \theta_k(t)}$
- Βήμα 2: $\nabla \theta_k(t) = \frac{1}{m} \sum_{i=1}^{m} \nabla C(\hat{Y}_i, Y_i)$
- Βήμα 3: $\theta_k(t+1) = \theta_k(t) - \alpha(t) \nabla \theta_k(t)$

3.3 Οπισθοδρόμηση

Κατά τη διάρκεια της εκπαίδευσης του δικτύου, είναι απαραίτητο να υπολογίσουμε με αποδοτικό τρόπο την **κλίση (gradient)** της συνάρτησης κόστους, η οποία εκφράζει τον τρόπο που αλλάζει η συνάρτηση κόστους με βάση την μεταβολή των βαρών και των bias στο δίκτυο. Τη λύση σε αυτό το πρόβλημα τη δίνει ο αλγόριθμος **οπισθοδρόμησης (back-propagation)** [29], ο οποίος υπολογίζει, ουσιαστικά, τις μερικές παραγώγους $\frac{\partial C}{\partial w}$ και $\frac{\partial C}{\partial b}$, όπου C είναι η συνάρτηση κόστους, w ένα βάρος του δικτύου και b η αντίστοιχη μεροληψία.

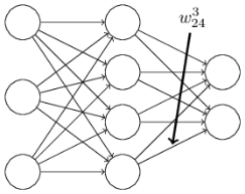
3.3.1 Συμβολισμοί

Προκειμένου να συνεχίσουμε, θεωρούμε τους εξής συμβολισμούς:

- w^l_{jk}: το **βάρος** που συνδέει το k-οστό νευρώνα στο $l-1$-οστό επίπεδο με το j-οστό νευρώνα στο l-οστό επίπεδο (εικόνα 3.5).
- b^l_j: η **μεροληψία** του j-οστού νευρώνα στο l-οστό επίπεδο (3.6).
- a^l_j: η **ενεργοποίηση** (έξοδος της συνάρτησης ενεργοποίησης) του j-οστού νευρώνα στο l-οστό επίπεδο.
- L: το πλήθος των επιπέδων του δικτύου.
- n: το πλήθος των δεδομένων εκπαίδευσης.
- x: η είσοδος στο δίκτυο, δηλαδή ένα δεδομένο εκπαίδευσης.
- $y = y(x)$: η αντίστοιχη επιθυμητή έξοδος του δικτύου με είσοδο το x.

Ορίζουμε, επιπλέον, το **σφάλμα** δ^l_j του νευρώνα j στο επίπεδο l ως εξής:

$$\delta^l_j \equiv \frac{\partial C}{\partial z^l_j} \tag{3.3}$$



Εικόνα 3.5: Βάρος ακμής [29].

Κατ' επέκταση, για τη διευκόλυνσή μας, θα έχουμε τα βάρη σε μορφή πίνακα, δηλαδή $w^l = \left[w^l_{jk}\right]$, και σε διανυσματική μορφή το κάθε bias και ενεργοποίηση, δηλαδή $b^l = \left[b^l_1, b^l_2, \dots\right]$ και $a^l = \left[a^l_1, a^l_2, \dots\right]$, αντίστοιχα. Ως συνάρτηση ενεργοποίησης θεωρούμε τη συνάρτηση σ, η οποία με είσοδο πίνακα ή διάνυσμα θα εφαρμόζεται στοιχείο προς στοιχείο.

Με βάση τους παραπάνω συμβολισμούς, η ενεργοποίηση θα υπολογίζεται από την εξής σχέση:

$$a^l_j = \sigma\left(\sum_k w^l_{jk} a^{l-1}_j + b^l_j\right) \tag{3.4}$$

Η ισότητα αυτή προκύπτει από το γεγονός ότι η ενεργοποίηση σε έναν νευρώνα j του επιπέδου l υπολογίζεται ως η εφαρμογή της συνάρτησης ενεργοποίησης στο

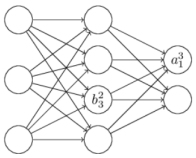
σταθμισμένο άθροισμα των εξόδων όλων των νευρώνων στο $l-1$ επίπεδο. Σε διανυσματική μορφή, η σχέση αυτή θα είναι:

$$a^l = \sigma\Big(w^l a^{l-1} + b^l\Big) \tag{3.5}$$

Σημειώνουμε, επιπλέον, ότι η ενδιάμεση ποσότητα:

$$z^l \equiv w^l a^{l-1} + b^l \tag{3.6}$$

ονομάζεται σταθμισμένη είσοδος στους νευρώνες στο επίπεδο l.



Εικόνα 3.6: Μεροληψία και ενεργοποίηση νευρώνα [29].

Τέλος, στον αλγόριθμο χρησιμοποιείται μία πράξη που ονομάζεται **γινόμενο Hadamard**, το οποίο ορίζεται ως το γινόμενο στοιχείου προς στοιχείο δύο διανυσμάτων s και t, και συμβολίζεται με: $s \odot t$. Τα στοιχεία, δηλαδή του διανύσματος αυτού θα είναι τα $(s \odot t)_j = s_j t_j$. Για παράδειγμα:

$$\begin{bmatrix}1\\2\end{bmatrix} \odot \begin{bmatrix}3\\4\end{bmatrix} = \begin{bmatrix}1\cdot 3\\2\cdot 4\end{bmatrix} = \begin{bmatrix}3\\8\end{bmatrix}$$

3.3.2 Παραδοχές για τη συνάρτηση κόστους

Για να προχωρήσουμε με τον αλγόριθμο, χρειαζόμαστε δύο παραδοχές για τη συνάρτηση κόστους.

- Πρώτον, καθώς ο αλγόριθμος, για να υπολογίσει τα μεγέθη $\frac{\partial C}{\partial w^l_{jk}}$ και $\frac{\partial C}{\partial b^l_j}$, υπολογίζει πρώτα τα $\frac{\partial C_x}{\partial w}$ και $\frac{\partial C_x}{\partial b}$, όπου με C_x συμβολίζουμε τη συνάρτηση κόστους με είσοδο στο δίκτυο μόνο το δεδομένο x, θα είναι απαραίτητο η συνάρτηση κόστους να μπορεί να γραφτεί ως: $C = \frac{1}{n}\sum_x C_x$.

- Δεύτερον, υποθέτουμε ότι η συνάρτηση κόστους μπορεί να γραφτεί συναρτήσει των εξόδων του δικτύου, δηλαδή $C = C(a^L)$.

Για παράδειγμα, θεωρούμε την τετραγωνική συνάρτηση κόστους:

$$C = \frac{1}{2n}\sum_x \big\|y(x) - a^L(x)\big\|^2$$

Τότε θα είναι $C_x = \frac{1}{2}\left\|y(x) - a^L(x)\right\|^2$, και, πράγματι, θα ισχύει ότι $C = \frac{1}{n}\sum_x C_x$. Επίσης, δεδομένου ότι κατά τη διάρκεια της εκπαίδευσης τα δεδομένα εισόδου, και κατά συνέπεια οι επιθυμητές έξοδοι, δεν μεταβάλλονται με την αλλαγή των βαρών και των bias, η συνάρτηση κόστους μεταβάλλεται συναρτήσει μόνο των a^L, δηλαδή των εξόδων. Πράγματι, με είσοδο ένα δεδομένο x, η συνάρτηση γράφεται:

$$C_x = \frac{1}{2}\|y - a^L\|^2 = \frac{1}{2}\sum_j (y_j - a_j^L)^2 = C_x(a_j^L)$$

3.3.3 Ο αλγόριθμος

Ακολουθούμε την εξής πορεία βημάτων ώστε να εφαρμόσουμε το αλγόριθμο οπισθοδρόμησης:

1. **Είσοδος** x: Θέτουμε τις ενεργοποιήσεις a^1 στο επίπεδο εισόδου, που είναι και το πρώτο επίπεδο του δικτύου, ως την είσοδο του δεδομένου x στο δίκτυο.

2. **Forward propagation**: Για καθένα από τα επόμενα επίπεδα $l = 2, 3, \ldots, L$ υπολογίζουμε τις εξόδους των αντίστοιχων νευρώνουν ως $a^l = \sigma\left(z^l\right)$, έχοντας υπολογίσει τις σταθμισμένες εισόδους $z^l = w^l a^{l-1} + b^l$.

3. **Σφάλμα εξόδου**: Υπολογίζουμε το σφάλμα στην έξοδο του επιπέδου εξόδου με τον τύπο $\delta^L = \nabla_a C \odot \sigma'\left(z^L\right)$.

4. **Οπισθοδρόμηση του σφάλματος**: Για κάθε επίπεδο $l = L-1, L-2, \ldots, 2$ υπολογίζουμε το σφάλμα δ^l στις εξόδους των νευρώνων με τον τύπο $\delta^l = \left((w^{l+1})^T \delta^{l+1}\right) \odot \sigma'\left(z^l\right)$

5. **Έξοδος αλγορίθμου**: Οι ζητούμενες μερικές παράγωγοι που υπολογίζονται από τις σχέσεις $\frac{\partial C}{\partial w_j^l k} = a_k^{l-1}\delta_j^l$ και $\frac{\partial C}{\partial b_j^l} = \delta_j^l$.

Σημειώνουμε ότι οι σχέσεις που χρησιμοποιούνται αποτελούν ισότητες που αποδεικνύονται με την χρήση του κανόνα της αλυσίδας της Διανυσματικής Ανάλυσης.

Από τον αλγόριθμο παρατηρούμε ότι μπορούμε να υπολογίσουμε το σφάλμα στο κάθε επίπεδο με βάση το σφάλμα και τις παραμέτρους στο προηγούμενο επίπεδο. Έτσι, έχοντας υπολογίσει, αρχικά, το σφάλμα στο τελευταίο επίπεδο L με την σχέση $\delta_j^L = \frac{\partial C}{\partial a_j^L}\sigma'(z_j^l)$, μπορούμε να υπολογίζουμε κάθε φορά το σφάλμα του αμέσως προηγούμενου επιπέδου χρησιμοποιώντας την σχέση $\delta^l = \left((w^{l+1})^T \delta^{l+1}\right) \odot \sigma'(z^l)$. Επομένως, οι υπολογισμοί σε αυτήν την περίπτωση ακολουθούν την αντίθετη κατεύθυνση από αυτή του forward propagation, κατά το οποίο υπολογίζεται η έξοδος του δικτύου. Αυτός είναι και ο λόγος που ο αλγόριθμος ονομάζεται back-propagation (οπισθοδρόμησης).

Θυμίζουμε ότι ο αλγόριθμος back-propagation στην πραγματικότητα υπολογίζει αποδοτικά και προσεγγιστικά την κλίση του κόστους. Η εκπαίδευση γίνεται, τελικά,

με την βοήθεια ενός αλγορίθμου καθόδου κλίσεων, όπως ο SGD που έχει αναφερθεί, έχοντας υπολογίσει προηγουμένως τις παραπάνω μερικές παραγώγους. Σημειώνουμε, επίσης, ότι η συνάρτηση ενεργοποίησης δεν είναι απαραίτητο να είναι η σ, καθώς κατά τη εφαρμογή των σχέσεων του αλγορίθμου δεν χρησιμοποιήθηκε κάποια από τις ιδιότητές της.

Κεφάλαιο 4

Συνελικτικά Νευρωνικά Δίκτυα

Τα **συνελικτικά νευρωνικά δίκτυα (Convolutional Neural Network - CNN)** αποτελούν μία κατηγορία νευρωνικών δικτύων που είναι ιδιαίτερα αποτελεσματικά στην αναγνώριση και κατηγοριοποίηση οπτικού υλικού [32], όπως, για παράδειγμα, στον εντοπισμό προσώπων και αντικείμενων σε μία εικόνα. Σε σύγκριση με άλλες κατηγορίες αλγορίθμων ταξινόμησης, η προ-επεξεργασία των δεδομένων εισόδου που απαιτείται είναι αρκετά χαμηλότερη. Επιπλέον, ένα βασικό πλεονέκτημά τους αποτελεί το γεγονός ότι τα CNNs έχουν την δυνατότητα να «μαθαίνουν» από μόνα τους τα φίλτρα και χαρακτηριστικά που τα δομούν, με αποτέλεσμα να κατανοούν καλύτερα τα πολύπλοκα στοιχεία μιας εικόνας.

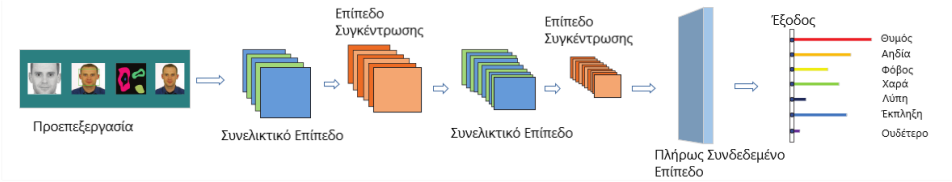
Ένα συνελικτικό νευρωνικό δίκτυο αποτελείται από ένα επίπεδο εισόδου, ένα επίπεδο εξόδου καθώς και από πολλαπλά κρυφά επίπεδα τα οποία απαρτίζονται από:

- **Συνελικτικά επίπεδα**, που είναι και τα δομικό στοιχεία που χαρακτηρίζουν τα CNNs
- **Επίπεδα συγκέντρωσης**
- **Πλήρως συνδεδεμένα επίπεδα**
- **Συναρτήσεις ενεργοποίησης**

Ένα παράδειγμα συνελικτικού νευρωνικού δικτύου φαίνεται στην εικόνα 4.1, το οποίο λαμβάνει ως είσοδο μία προ-επεξεργασμένη εικόνα δύο διαστάσεων με σκοπό να αναγνωρίσει το συναίσθημα που κυριαρχεί στο πρόσωπο ενός ανθρώπου.

4.1 Συνελικτικά επίπεδα

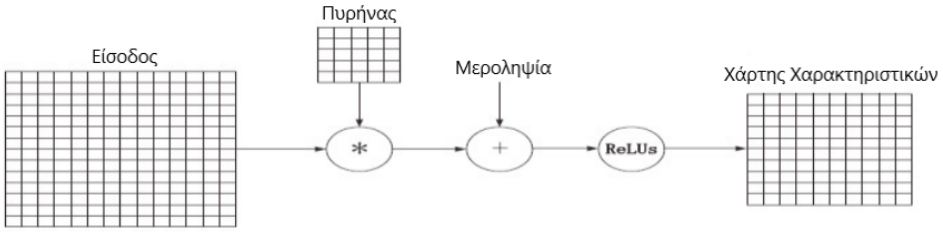
Ένα **συνελικτικό επίπεδο** έχει ως είσοδο, κατά κύριο λόγο, διανύσματα 3 διαστάσεων που αντιπροσωπεύουν το ύψος, μήκος και βάθος μίας εικόνας, ενώ το



Εικόνα 4.1: Η γενική δομή ενός CNN [33].

ίδιο χαρακτηρίζεται από τις υπερπαραμέτρους του που είναι:

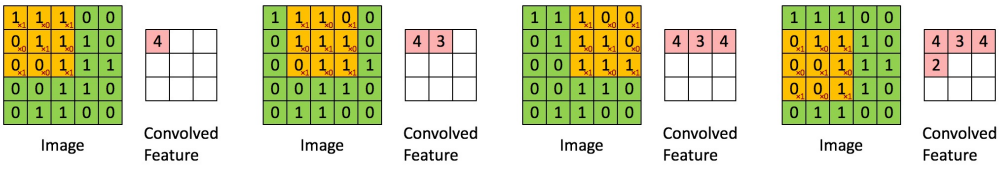
- ο **πυρήνας** (**kernel**), δηλαδή πίνακας με διαστάσεις ύψος x μήκος
- το πλήθος **καναλιών εισόδου (input channels)**, που ισούται με το βάθος της εικόνας εισόδου
- το πλήθος **καναλιών εξόδου (output channels)** ή **φίλτρα**
- **βήμα ολίσθησης** (**stride**)



Εικόνα 4.2: Συνέλιξη εικόνας 14×14 με πυρήνα 5×5 και βήμα 1×1, με αποτέλεσμα έναν 10×10 χάρτη χαρακτηριστικών (feature map) [34].

Το αποτέλεσμα μετά από είσοδο μίας εικόνας είναι ο πολλαπλασιασμός στοιχείου προς στοιχείο του πυρήνα με ένα κομμάτι του πίνακα, και στην συνέχεια το άθροισμα όλων αυτών των στοιχείων, δίνοντας ένα από τα στοιχεία του τελικού πίνακα που ονομάζουμε **χάρτη χαρακτηριστικών** (**feature map**). Η διαδικασία αυτή συνεχίζεται με τον πυρήνα να «ολισθαίνει» πάνω στην εικόνα, μετακινούμενος προς όλες τις πιθανές κατευθύνσεις [34]. Μπορούμε να δούμε μια οπτική αναπαράσταση της διαδικασίας αυτής στις εικόνες 4.2 και 4.3.

Σκοπός ενός συνελικτικού επιπέδου είναι να εξάγει χαρακτηριστικά υψηλού επιπέδου (π.χ. ακμές), μειώνοντας, παράλληλα, το πλήθος των ελεύθερων παραμέτρων του δικτύου, με αποτέλεσμα να έχουμε περιθώριο το δίκτυο να είναι πιο βαθύ και να περιορίζονται τα θέματα που εμφανίζονται κατά την οπισθοδρόμηση. Ενώ, βέβαια, η ανάλυση εικόνων είναι εφικτή και με πλήρως συνδεδεμένα -



Εικόνα 4.3: Παράδειγμα συνέλιξης [35], με βήμα 1×1, της εισόδου με τον 3×3 πυρήνα: $\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$.

τροφοδοτικά δίκτυα, δεν θα ήταν καλή επιλογή δεδομένου του πολύ μεγάλου πλήθους νευρώνων που θα ήταν αναγκαίοι, ένα πρόβλημα στο οποίο δίνουν λύση τα συνελικτικά επίπεδα.

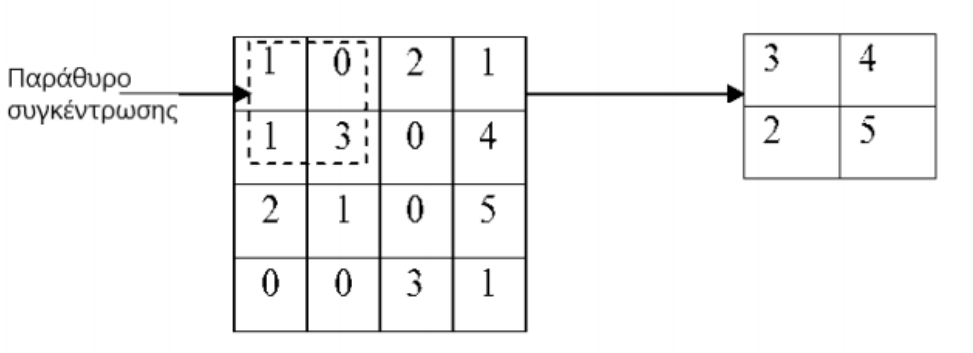
4.2 Επίπεδα Συγκέντρωσης

Ένα πρόβλημα που παρουσιάζουν τα συνελικτικά επίπεδα είναι ότι καταγράφουν την ακριβή θέση ενός χαρακτηριστικού στην εικόνα, με συνέπεια να προκύπτει διαφορετικός χάρτης χαρακτηριστικών για την ίδια εικόνα, με διαφορές όπως η περιστροφή της, η ολίσθησή της και η αποκοπή τμημάτων της που δεν παρέχουν σημαντική πληροφορία. Αυτό μπορούμε να το αντιμετωπίσουμε με την υποδειγματοληψία της εικόνας, κάτι που επιτυγχάνεται και στα συνελικτικά επίπεδα αυξάνοντας το βήμα ολίσθησης του πυρήνα. Ωστόσο, ένας πιο συνήθης και αποτελεσματικός τρόπος είναι η χρήση **επιπέδων συγκέντρωσης (pooling layers)**, τα οποία τοποθετούνται στο δίκτυο μετά τα συνελικτικά επίπεδα [34].

Για τα επίπεδα συγκέντρωσης, ορίζουμε την πράξη η οποία θα πραγματοποιείται, πριν την διαδικασία εκπαίδευσης, και αυτή θα είναι μία εκ των:

- **Συγκέντρωση μέσης τιμής (Average Pooling)**
- **Συγκέντρωση μέγιστης τιμής (Max Pooling)**

δηλαδή η εξαγωγή μέσου όρου και ελάχιστης τιμής, αντίστοιχα, για το κάθε τμήμα του χάρτη χαρακτηριστικών (εικόνα 4.4). Η αντίστοιχη υπερπαράμετρος του επιπέδου συγκέντρωσης, δηλαδή το **βήμα ολίσθησης**, καθορίζει τα τμήματα δύο διαστάσεων της εικόνας εισόδου, πάνω στα οποία θα πραγματοποιηθούν οι κατάλληλες πράξεις. Το αποτέλεσμα είναι μια περιληπτική περιγραφή των χαρακτηριστικών της εικόνας εισόδου, το οποίο, μάλιστα, θα είναι το ίδιο ακόμη και για μικρές αλλαγές της θέσης μιας συγκεκριμένης τιμής. Σημειώνουμε, επίσης, ότι, ενώ θα έχει μειωθεί η ανάλυση των δεδομένων, θα έχουν διατηρηθεί τα σημαντικά δομικά στοιχεία τους απαραίτητα για την εκπαίδευση.



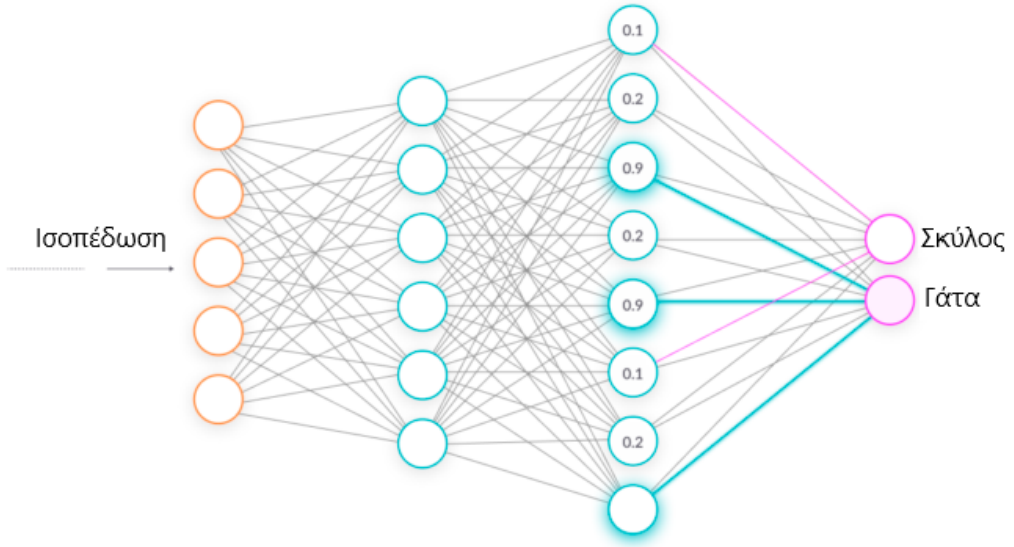
Εικόνα 4.4: Max pooling [34].

Παρόμοια με τα συνελικτικά επίπεδα, τα επίπεδα συγκέντρωσης συμβάλλουν στην μείωση των διαστάσεων των δεδομένων μέσα στο δίκτυο, τη μείωση των παραμέτρων του και των συνολικών υπολογισμών. Ειδικότερα, τα επίπεδα μέγιστης τιμής πραγματοποιούν μείωση θορύβου που, ενδεχομένως, έχει προκύψει μετά από συνάρτηση ενεργοποίησης, με τα επίπεδα μέσης τιμής, από την άλλη, να επιτυγχάνουν αποθορυβοποίηση μέσω της μείωσης των διαστάσεων των δεδομένων.

4.3 Πλήρως Συνδεδεμένα Επίπεδα

Ένα **πλήρως συνδεδεμένο επίπεδο (fully connected layer)** [36] είναι ουσιαστικά ένα πολυεπίπεδο perceptron, καθώς κάθε κόμβος του αντιπροσωπεύει τον πολλαπλασιασμό της εισόδου με το αντίστοιχο βάρος, ακολουθούμενο από την πρόσθεση της μεροληψίας. Κάθε νευρώνας σε προηγούμενο επίπεδο είναι συνδεδεμένος σε κάθε νευρώνα αυτού του επιπέδου, οπότε και ονομάζεται «πλήρως συνδεδεμένο».

Στην πράξη, η είσοδος σε αυτό θα πρέπει να περάσει πρώτα από ένα **επίπεδο ισοπέδωσης** (**flatten layer**), έτσι ώστε τα δεδομένα να μετατραπούν σε ένα πίνακα μίας διάστασης προκειμένου να επεξεργαστούν. Σε ένα δίκτυο μπορούμε να έχουμε πολλαπλά τέτοια επίπεδα, με την είσοδο στο πρώτο από αυτά να αποτελεί την έξοδο από ένα συνελικτικό ή ένα επίπεδο συγκέντρωσης. Σκοπός τους είναι να χρησιμοποιήσουν τα δεδομένα που προέκυψαν σε προηγούμενο στάδιο ώστε να τα κατηγοριοποιήσουν, το οποίο επιτυγχάνεται στο τελευταίο επίπεδο με τη βοήθεια μιας συνάρτησης ενεργοποίησης η οποία δίνει την πιθανότητες η είσοδος να ανήκει σε μία τάξη (ταξινόμηση ή classification) και είναι συνήθως η **softmax**. Στην εικόνα 4.5 βλέπουμε μια οπτική αναπαράσταση ενός πλήρως συνδεδεμένου επιπέδου, το οποίο, έχοντας λάβει μία εικόνα που, ενδεχομένως, έχει υποστεί επεξεργασία από προηγούμενα επίπεδα του συνελικτικού νευρωνικού δικτύου, αναγνωρίζει αν πρόκειται για απεικόνιση σκύλου ή γάτας.



Εικόνα 4.5: Fully connected layer [37].

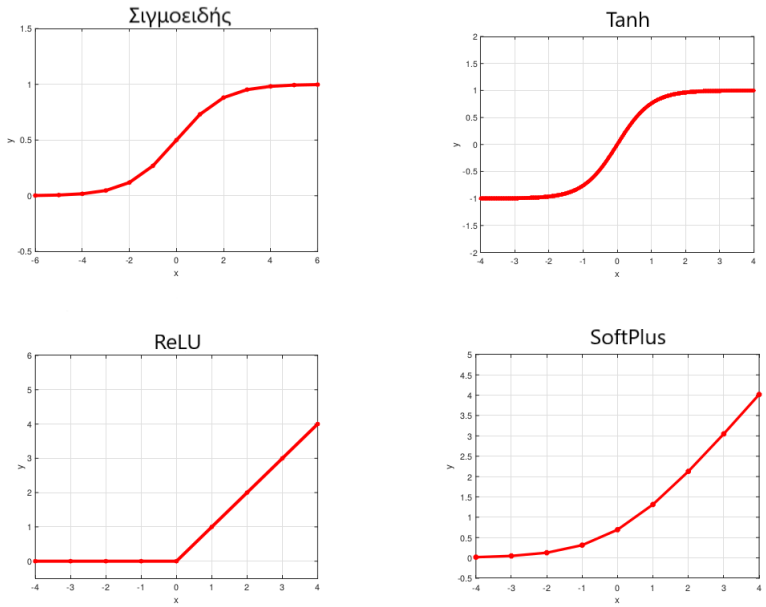
4.4 Συναρτήσεις Ενεργοποίησης

Συνήθεις **συναρτήσεις ενεργοποίησης (activation functions)** σε ένα βαθύ νευρωνικό δίκτυο αποτελούν οι:

- Sigmoid: $f(x) = \frac{1}{1+e^{-x}}$
- Tanh: $f(x) = \frac{1-e^{-2x}}{1+e^{2x}}$
- ReLU: $f(x) = max(0, x)$
- Softplus: $f(x) = ln(1 + e^{x})$

των οποίων τις γραφικές παραστάσεις μπορούμε να δούμε στην εικόνα 4.6. Από τις παραπάνω συναρτήσεις, προτιμάται η χρήση της **ReLU** καθώς, σε σύγκριση με τις Sigmoid και Tanh, επιλύουν το gradient diffusion πρόβλημα. Ενώ αυτό το επιτυγχάνει και η Softplus, η ReLU υπερτερεί διότι πραγματοποιεί ταχύτερα τους υπολογισμούς στο δίκτυο αυξάνοντας την απόδοσή του. Σημειώνουμε ότι η **softmax** συνάρτηση ενεργοποίησης χρησιμοποιείται, συνήθως, στο τελευταίο και όχι στα ενδιάμεσα επίπεδα.

Η σημασία τους έγκειται στην επίλυση **μη γραμμικών συστημάτων** με τα νευρωνικά δίκτυα, στο οποίο οι ταξινομητές, ακόμα και με διάφορους συνδυασμούς τους, αποτυγχάνουν καθώς οι ίδιοι αποτελούν γραμμικές εξισώσεις. Με αυτόν τον τρόπο, οι συναρτήσεις ενεργοποίησης αυξάνουν την ικανότητα του νευρωνικού



Εικόνα 4.6: Οι γραφικές παραστάσεις τεσσάρων συναρτήσεων ενεργοποίησης [33].

δικτύου να εκφράζεται, ενισχύοντας τον ρόλο του ως μοντέλο τεχνητής νοημοσύνης [32] [33].

4.5 Επίπεδα Εγκατάλειψης

Οι τρόποι για να μειώσουμε το σφάλμα κατά τον έλεγχο περιλαμβάνουν τον συνδυασμό των προβλέψεων πολλών διαφορετικών μοντέλων, το οποίο, ωστόσο, θα έχει επίπτωση την αύξηση, αφενός, του χρόνου εκπαίδευσης και, αφετέρου, του απαιτούμενου αριθμού πόρων. Προκειμένου, λοιπόν, να διατηρήσουμε τη λογική αυτή των πολλαπλών μοντέλων αλλά να κρατήσουμε χαμηλό το κόστος, μπορούμε να εισάγουμε στο δίκτυο μία τεχνική που ονομάζεται **εγκατάλειψη (dropout)**, με βάση την οποία θέτουμε την έξοδο κάθε κρυφού νευρώνα σε μηδενική τιμή, με μία συγκεκριμένη πιθανότητα p. Με την εισαγωγή, λοιπόν, των λεγόμενων **επιπέδων εγκατάλειψης**, σε κάθε επανάληψη του αλγορίθμου μάθησης θα έχουμε στην πραγματικότητα ένα διαφορετικό μοντέλο, το οποίο, όμως, θα μοιράζεται τα αντίστοιχα βάρη, αναγκάζοντας τους νευρώνες να μην εξαρτώνται ο ένας από τον άλλο, συμβάλλοντας στην μάθηση πιο πολύπλοκων χαρακτηριστικών της εισόδου [32].

Κεφάλαιο 5

Συστήματα Εκπαίδευσης Νευρωνικών Δικτύων

Η συνεχής αύξηση του ενδιαφέροντος για την μηχανική μάθηση, και κατ' επέκταση για τα νευρωνικά δίκτυα, έχει οδηγήσει στην υλοποίηση εργαλείων ανοιχτού κώδικα (**«συστήματα» ή «frameworks»**) [38] [39] που αποσκοπούν στην επίλυση προβλημάτων με τη χρήση αλγορίθμων τύπου Μηχανικής Μάθησης (Machine Learning - ML).

Μερικοί τομείς στους οποίους ανήκουν τα εργαλεία αυτά φαίνονται στον πίνακα 5.1.

Τομέας	Παράδειγμα
Συστήματα πρόβλεψης	Αυτόματη συμπλήρωση κειμένου
Τα συστήματα συστάσεων	Προτάσεις ταινιών βάσει ιστορικού
Πλατφόρμες ανάλυσης δεδομένων	Διαχείριση βάσης δεδομένων μιας επιχείρησης
Επεξεργασία εικόνας, ήχου ή βίντεο	Αναγνώριση προσώπου σε φωτογραφία

Πίνακας 5.1: Παραδείγματα εφαρμογής ML

Κανένα, όμως, από τα παραπάνω εργαλεία δεν είναι ιδανικό για την επίλυση κάθε προβλήματος. Εν αντιθέσει, πολλές φορές είναι αναγκαίος ο συνδυασμός τους προκειμένου να υπάρξει επιτυχία.

Ως υποκατηγορία της Μηχανικής Μάθησης, η βαθιά μάθηση, με τα CNNs να αποτελούν αρχιτεκτονική τέτοιας μορφής, συνιστά μία από τις κυρίαρχες πλέον τάσεις καθώς έχει καταφέρει να δώσει λύσεις που υπερτερούν αυτών της «απλής» μηχανικής μάθησης, σε τομείς όπως η επεξεργασία εικόνας και η επεξεργασία φυσικής γλώσσας. Για την αποτελεσματικότερη χρήση και εξέλιξή τους έχει δημιουργηθεί πληθώρα εργαλείων που υλοποιούν ποικίλους αλγορίθμους Βαθιάς

Μάθησης με το δικό τους μοναδικό τρόπο.

Μερικά από τα πιο δημοφιλή συστήματα είναι τα εξής:

- TensorFlow [2]
- PyTorch [3]
- MXNet [40]
- Theano [41]
- CNTK [42]
- Caffe [43]
- Chainer [44]

καθένα από τα οποία έχει τα πλεονεκτήματά του ως προς την επίλυση ενός προβλήματος με τη χρήση αλγορίθμων μηχανικής μάθησης. Μάλιστα, ακόμα και αν ένας τέτοιος αλγόριθμος έχει υλοποιηθεί σε κάθε σύστημα, η απόδοση εκτέλεσης είναι πολύ πιθανό να διαφέρει σημαντικά. Σε συνδυασμό και με την ύπαρξη ενός ευρέος φάσματος από τα εργαλεία αυτά, η βέλτιστη επιλογή κάποιου για έναν συγκεκριμένο σκοπό καθίσταται αρκετά δύσκολη, δεδομένου, επίσης, ότι η έρευνα πάνω στην συγκριτική αξιολόγησή τους είναι ακόμα περιορισμένη.

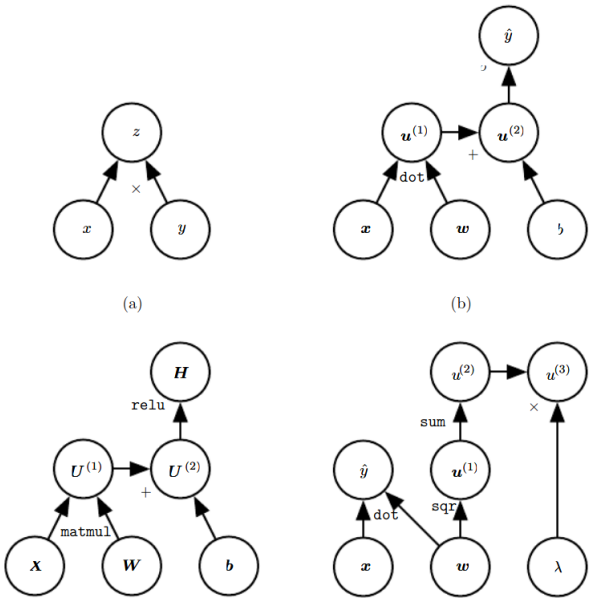
5.1 Υπολογιστικοί Γράφοι

Προκειμένου να αναλύσουμε τα συστήματα θα ήταν χρήσιμο να ορίσουμε για το νευρωνικό δίκτυο έναν γράφο ροής δεδομένων ή **υπολογιστικό γράφο (computational graph)** [28], ο οποίος και αποτελεί την βάση της υλοποίησης τόσο του TensorFlow όσο και του PyTorch. Συγκεκριμένα, ένας κόμβος αντιστοιχεί σε μία μεταβλητή, η οποία μπορεί να είναι μία αριθμητική τιμή, διάνυσμα, πίνακας, τανυστής (tensor) ή και ένας τύπος δεδομένων. Επιπλέον, κάθε κόμβος συνοδεύεται από έναν **«τελεστή» (operator)**, ο οποίος πραγματοποιεί μία λειτουργία (operation), δηλαδή αποτιμά μία συνάρτηση μίας ή περισσότερων μεταβλητών. Αν μία μεταβλητή y υπολογίζεται με την εφαρμογή ενός operation στην μεταβλητή x, τότε σχεδιάζουμε μία κατευθυνόμενη ακμή από τον κόμβο που αντιστοιχεί στην x στον κόμβο που αντιστοιχεί στην y. Στον κόμβο εξόδου σημειώνουμε την λειτουργία που πραγματοποιείται, εκτός αν είναι προφανής.

Μερικά τέτοια παραδείγματα αφηρημένου γράφου φαίνονται στην εικόνα 5.1.

5.2 TensorFlow

Το σύστημα TensorFlow [2] είναι μία βιβλιοθήκη ανοιχτού κώδικα με εφαρμογή, αφενός, στον διαφορικό προγραμματισμό, δηλαδή τον υπολογισμό της παραγώγου



Εικόνα 5.1: Παραδείγματα υπολογιστικών γράφων [28].

μίας συνάρτησης με τεχνικές αυτόματης διαφοροποίησης ενός υπολογιστικού προγράμματος, και, αφετέρου, στον προγραμματισμό τύπου ροής δεδομένων, κατά τον οποίο ένα πρόγραμμα μοντελοποιείται ως ένας κατευθυνόμενος γράφος, στον οποίο τα δεδομένα ρέουν από μία «λειτουργία» σε μία άλλη. Έχει υλοποιηθεί, αλλά και χρησιμοποιείται, από την Google Brain, μία ομάδα ερευνητών της Google που ασχολείται με τους τομείς της Βαθιάς Μάθησης και Τεχνητής Νοημοσύνης.

Μία εφαρμογή του TensorFlow διακρίνεται, τυπικά, σε δύο φάσεις [45]:

- Η πρώτη φάση περιλαμβάνει τον ορισμό του προγράμματος προς εκτέλεση (για παράδειγμα, ένα νευρωνικό δίκτυο προς εκπαίδευση) με τη μορφή γράφου.
- Η δεύτερη φάση περιλαμβάνει την εκτέλεση μιας βελτιστοποιημένης μορφής του προγράμματος πάνω σε ένα σύνολο από διαθέσιμες συσκευές.

Η εκτέλεση του προγράμματος εκ των υστέρων, δηλαδή αφότου είναι πλήρως έτοιμο, βελτιώνει την απόδοσή του ως προς τον χρόνο που απαιτεί, αυξάνει, όμως, την πολυπλοκότητα του κώδικα.

Σημειώνουμε, ωστόσο, ότι η λογική του TensorFlow διαφέρει μεταξύ των, μέχρι τώρα, δύο εκδόσεων που το απαρτίζουν. Σύμφωνα με την πρώτη έκδοση, προκειμένου να υλοποιηθεί ένα πρόγραμμα Μηχανικής Μάθησης, είναι απαραίτητο ο προγραμματιστής να υλοποιεί ένα αφηρημένο συντακτικό δέντρο, δηλαδή τον υπολογιστικό γράφο, χρησιμοποιώντας αναφορές σε συναρτήσεις της βιβλιοθήκης του TensorFlow [46], μία διαδικασία που εισάγει πολυπλοκότητα στην συγγραφή του κώδικα, καθώς η λογική της είναι αρκετά χαμηλού επιπέδου. Από την άλλη πλευρά,

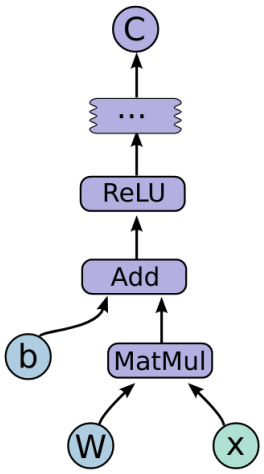
με την δεύτερη έκδοση του TensorFlow, η εκτέλεση του κώδικα είναι «ανυπόμονη» **(eager execution)**. Ειδικότερα, η ροή του προγράμματος ακολουθεί την ροή της Python και όχι την ροή του γράφου, με την έννοια ότι ο χρήστης, πλέον, δεν είναι απαραίτητο να συντάξει έναν γράφο που θα εκτελεστεί αργότερα, αλλά γράφει κώδικα ο οποίος θα εκτελεστεί με την σειρά που είναι προφανής από την σειρά σύνταξής του.

Πρέπει να τονίσουμε, όμως, ότι κάποιες βιβλιοθήκες της δεύτερης έκδοσης διατηρούν την λογική της εκτέλεσης του γράφου, για να υπάρχει καλύτερη απόδοση ως προς το χρόνο εκτέλεσης. Ένα τέτοιο περιβάλλον, το οποίο έχει ενταχθεί στο TensorFlow, είναι το **Keras**, μία βιβλιοθήκη υψηλού επιπέδου που απλοποιεί ακόμα περισσότερο την δημιουργία νευρωνικών μοντέλων [47] [48].

5.2.1 Δομή

Όπως αναφέρθηκε, το TensorFlow μπορεί να αναπαρασταθεί από έναν κατευθυνόμενο γράφο που αποτελείται από ένα σύνολο κόμβων και ακμών. Ένα τέτοιο παράδειγμα γράφου φαίνεται στην εικόνα 5.2. Τα δεδομένα του γράφου «ρέουν» πάνω στις ακμές του από εξόδους σε εισόδους, και ονομάζονται **τανυστές (tensors)**, δηλαδή πίνακες αυθαίρετων διαστάσεων με τύπο δεδομένων που ορίζεται κατά την κατασκευή του γράφου. Υπάρχουν, επίσης, ειδικές ακμές που ονομάζονται εξαρτήσεις ελέγχου, στις οποίες δεν ρέει κάποιο δεδομένο, υποδεικνύουν, όμως, ότι ο πηγαίος κόμβος πρέπει να ολοκληρώσει την λειτουργία του πριν τον κόμβο άφιξης.

Κάθε ένας από τους κόμβους του γράφου μπορεί να έχει μηδέν ή περισσότερες εισόδους, και μηδέν ή περισσότερες εξόδους, ενώ, παράλληλα, αντιπροσωπεύει μία **λειτουργία (operation)**. Μία λειτουργία συνοδεύεται από το όνομά της και ισοδυναμεί με μια αφηρημένη μορφή υπολογισμού, όπως, για παράδειγμα, είναι ο πολλαπλασιασμός πινάκων. Κάθε λειτουργία διαθέτει, επιπλέον, **χαρακτηριστικά (attributes)** τα οποία πρέπει, κατά την δημιουργία του γράφου, να έχουν οριστεί, ώστε να είναι έτοιμος ο κόμβος να εκτελέσει τη λειτουργία του. Μια συνηθισμένη χρήση των χαρακτηριστικών αυτών είναι να καθορίζουν την πολυμορφία μιας πράξης για τιμές διαφορετικού τύπου. Τέτοιο παράδειγμα είναι ο ορισμός της λειτουργίας της πρόσθεσης, με τέτοια χαρακτηριστικά που να της επιτρέπουν να δέχεται ως είσοδο τιμές με τύπο τους δεκαδικούς αριθμούς, αλλά και τιμές με τύπο τους ακέραιους αριθμούς. Ακόμη, ονομάζουμε **πυρήνα (kernel)** την υλοποίηση μίας λειτουργίας, η οποία της επιτρέπει να εκτελεστεί σε μία συγκεκριμένη συσκευή (CPU ή GPU). Τέλος, υπάρχει ένα ακόμα είδος λειτουργίας που ονομάζεται Μεταβλητή (Variable), και αποτελεί αναφορά σε τανυστή που «επιβιώνει» μεταξύ διαφορετικών εκτελέσεων του γράφου, καθώς, τυπικά, οι τανυστές δεν «ζουν» μετά το πέρας μίας εκτέλεσης.



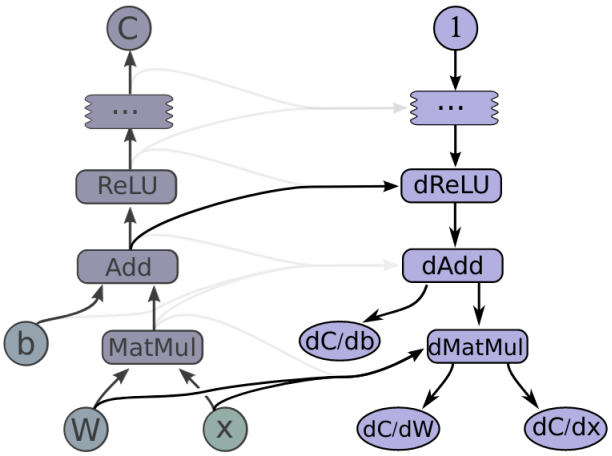
Εικόνα 5.2: Ένας απλός υπολογιστικός γράφος στο TensorFlow [2].

5.2.2 Υπολογισμός κλίσης

Εφόσον πολλοί αλγόριθμοι βελτιστοποίησης στη Μηχανική Μάθηση, όπως ο αλγόριθμος εκπαίδευσης νευρωνικών δικτύων Stochastic Gradient Descent, περιλαμβάνουν την κλίση της συνάρτησης κόστους ως προς τις εισόδους, είναι αναγκαίος ένας αποδοτικός υπολογισμός της κλίσης αυτής. Το TensorFlow, για να ικανοποιήσει την παραπάνω ανάγκη, διαθέτει μηχανισμό αυτόματου υπολογισμού της κλίσης μίας συνάρτησης. Ο τρόπος με τον οποίο το πραγματοποιεί είναι να επεκτείνει, κατά την διάρκεια της οπισθοδρόμησης, τον υπάρχοντα υπολογιστικό γράφο. Συνοπτικά, όταν το TensorFlow απαιτεί τον υπολογισμό της κλίσης του τανυστή C ως προς τον τανυστή I, από τον οποίο και εξαρτάται ο C, βρίσκει το μονοπάτι που τους ενώνει στον γράφο, και το ακολουθεί κατευθυνόμενο από το C στο I, δηλαδή οπισθοδρομεί. Για κάθε λειτουργία που συναντάει, προσθέτει έναν κόμβο που υπολογίζει τη μερική παράγωγο της λειτουργίας αυτής, με την χρήση του κανόνα της αλυσίδας, με είσοδο τόσο τις μερικές παραγώγους που έχουν υπολογιστεί σε προηγούμενο στάδιο, όσο και τις εισόδους και εξόδους που χρησιμοποιήθηκαν στον αρχικό γράφο. Στην εικόνα 5.3 φαίνεται ο γράφος που δημιουργείται με βάση τον γράφο της εικόνας 5.2.

5.2.3 Υλοποίηση

Η βιβλιοθήκη **TensorFlow** είναι υλοποιημένη στη γλώσσα C++, για λόγους απόδοσης και φορητότητας, ενώ οι χρήστες της μπορούν να συντάσσουν κώδικα σε διάφορες γλώσσες προγραμματισμού, μεταξύ των οποίων είναι η Python και η C++. Σημειώνουμε ότι τον κώδικα του χρήστη, με τον κώδικα του πυρήνα της βιβλιοθήκης, τον διαχωρίζει μία διεπαφή προγραμματισμού εφαρμογών (Application Programming Interface ή αλλίως API) της γλώσσας C. Το TensorFlow μπορεί να εκτελεστεί σε



Εικόνα 5.3: Τοποθέτηση κόμβων κατά την οπισθοδρόμηση [2].

διάφορα λειτουργικά συστήματα, όπως το Linux, Mac OS X, Windows, Android και iOS, αλλά και σε διάφορες αρχιτεκτονικές CPU και GPU.

Κατά τον χρόνο εκτέλεσης είναι διαθέσιμοι πάνω από 200 τελεστές, που υλοποιούν, μεταξύ άλλων, μαθηματικές πράξεις, χειρισμό πινάκων, διαχείριση της ροής ελέγχου και της κατάστασης του προγράμματος. Έχουν συνταχθεί, μάλιστα, πυρήνες που είναι, ουσιαστικά, η συγχώνευση κάποιων πυρήνων **(fused kernels)** που καταναλώνουν σημαντικό χρόνο εκτέλεσης, όπως η ReLU και η σιγμοειδής συνάρτηση ενεργοποίησης, για να αυξήσουν την συνολική απόδοση.

Παράλληλα με τον πυρήνα της βιβλιοθήκης, οι προγραμματιστές του TensorFlow έχουν δημιουργήσει εργαλεία που παρακολουθούν την πορεία της εκπαίδευσης ενός δικτύου, οπτικοποιούν τον αντίστοιχο γράφο αλλά και παρέχουν πληροφορίες για την εκτέλεση του προγράμματος.

Τα κύρια γνωρίσματα σε ένα σύστημα TensorFlow είναι οι εξής διεργασίες (tasks):

- ο **πελάτης (client)** που επικοινωνεί με τον **αφέντη (master)**
- ένας ή περισσότεροι **εργάτες (workers)** που είναι υπεύθυνοι για την πρόσβαση σε υπολογιστικές συσκευές (πυρήνες CPU ή κάρτες GPU) και την εκτέλεση κόμβων πάνω σε αυτές, με τρόπο που ορίζει ο master.

Η εκτέλεση του TensorFlow μπορεί να γίνει είτε τοπικά είτε και κατανεμημένα.

5.2.4 Κατανεμημένη εκτέλεση

Το TensorFlow υποστηρίζει την κατανεμημένη εκτέλεσή του σε πολλαπλές GPU και σε πολλαπλές μηχανές. Η τελευταία έκδοσή του προσφέρει ένα σχετικό API, με

βάση το οποίο μπορεί ο χρήστης να εκτελεί τον κώδικά του είτε με τη μέθοδο του γράφου, είτε ανυπόμονα (συνιστάται, όμως, η πρώτη μέθοδος) [49]. Η κατανεμημένη εκτέλεση μπορεί να γίνει με δύο τρόπους, την **παραλληλοποίηση δεδομένων**, είτε **σύγχρονα** είτε **ασύγχρονα**, και την **παραλληλοποίηση μοντέλου**.

Η παραλληλοποίηση δεδομένων προσφέρεται από το API του TensorFlow που ονομάζεται **Strategy**. Στην **σύγχρονη εκτέλεση**, όλοι οι workers εκπαιδεύονται πάνω σε ένα διαφορετικό κομμάτι των δεδομένων εισόδου, και σε κάθε βήμα συγχρονίζονται και συναθροίζουν το αποτέλεσμά τους. Μία από τις τεχνικές που προσφέρει το TensorFlow σε αυτήν την περίπτωση περιλαμβάνει την αντιγραφή του μοντέλου σε κάθε συσκευή GPU, εάν έχουμε ένα μηχάνημα, και σε κάθε αντίγραφο «καθρεφτίζεται» κάθε μία από τις Μεταβλητές, αποτελώντας μία ενιαία (κατανεμημένη) οντότητα. Οι Μεταβλητές αυτές ενημερώνονται με τις ίδιες αντίστοιχες τιμές σε κάθε βήμα και, έτσι, παραμένουν συγχρονισμένες. Για να επικοινωνήσουν, εφαρμόζονται βελτιστοποιημένοι αλγόριθμοι τύπου **All-reduce** [50], οι οποίοι συναθροίζουν τους τανυστές κάθε συσκευής. Στην περίπτωση που έχουμε πολλά μηχανήματα, το καθένα, ενδεχομένως, με πολλές συσκευές, πάλι δημιουργούνται αντίγραφα των Μεταβλητών σε κάθε μηχάνημα και σε κάθε worker. Για την επικοινωνία, χρησιμοποιείται μία τεχνική που περιλαμβάνει ένα **συλλεκτικό τελεστή (collective op)**, ένας μοναδικός τελεστής στον γράφο του TensorFlow, ο οποίος διατηρεί συγχρονισμένες τις Μεταβλητές διαλέγοντας αυτόματα έναν κατάλληλο αλγόριθμο All-reduce, ανάλογα με τη διαθέσιμη αρχιτεκτονική, την τοπολογία του δικτύου των μηχανημάτων, και τα μεγέθη των τανυστών.

Στην **ασύγχρονη εκτέλεση**, οι workers εκπαιδεύονται ανεξάρτητα ο ένας με τον άλλο, και ενημερώνουν τις αντίστοιχες μεταβλητές τους ασύγχρονα. Τυπικά, η ασύγχρονη εκπαίδευση υποστηρίζεται από τεχνικές τύπου Εξυπηρετητή Παραμέτρων (Parameter Server) [51], κατά την οποία κάποια μηχανήματα αποκτούν τον ρόλο των εργατών, και άλλα τον ρόλο του εξυπηρετητή παραμέτρων. Κάθε μεταβλητή του μοντέλου τοποθετείται σε έναν εξυπηρετητή, ενώ οι υπολογισμοί καθρεφτίζονται και πραγματοποιούνται σε όλες τις GPUs των εργατών.

Με την μέθοδο της **παραλληλοποίησης μοντέλου**, ο γράφος κατανέμεται στις διαθέσιμες συσκευές για να κλιμακώσει η εκπαίδευση. Η διανομή μπορεί να πραγματοποιηθεί είτε σε ένα μηχάνημα είτε σε πολλαπλά, με το καθένα να περιλαμβάνει 0 ή περισσότερες GPUs.

5.3 PyTorch

Το σύστημα **PyTorch** είναι μία βιβλιοθήκη της γλώσσας Python που βελτιώνει την ταχύτητα της εκτέλεσης του κώδικά της με την υποστήριξη καρτών επεξεργασίας γραφικών (Graphics Processing Unit ή αλλιώς GPU), και χρησιμοποιείται για Βαθιά Μάθηση (Deep Learning) [38]. Χαρακτηρίζεται από τον προστατικό τρόπο προγραμματισμού του, που συμπληρώνει τον τρόπο γραφής σε

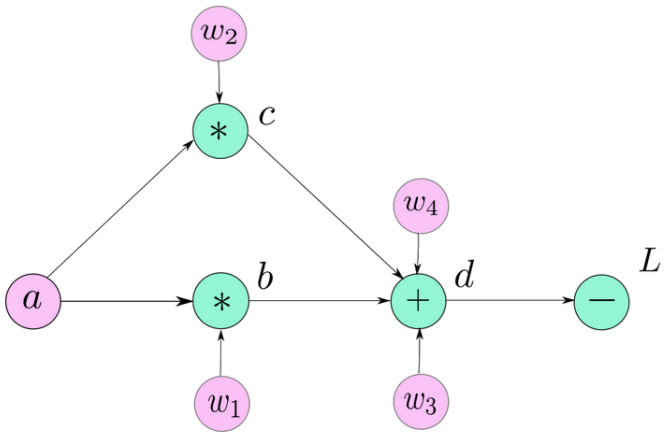
Python, και διακρίνεται για την απόδοσή του και την ευκολία αποσφαλμάτωσης του κώδικα [3]. Υλοποιήθηκε από την ομάδα έρευνας του Facebook (Facebook's AI Research lab, ή FAIR) το 2016, και είναι μια διεπαφή προγραμματισμού εφαρμογών των βιβλιοθηκών της C που χρησιμοποιεί η (πλέον χωρίς υποστήριξη) βιβλιοθήκη Torch.

Το σύστημα PyTorch είναι γραμμένο σε Python, C++ και CUDA (πλατφόρμα παράλληλων υπολογισμών, και API που επιτρέπει τη χρήση GPU για υπολογισμούς). Υποστηρίζει, αφενός, τον υπολογισμό τανυστών, ο οποίος μπορεί να επιταχυνθεί σημαντικά με τη χρήση των GPUs, και, αφετέρου, την κατασκευή βαθιών νευρωνικών δικτύων μέσω ενός συστήματος αντίστροφης αυτόματης διαφόρισης βαθμωτών συναρτήσεων. Χρησιμοποιείται τόσο από την επιστημονική όσο και την βιομηχανική κοινότητα.

5.3.1 Δομή

Ένα κύριο γνώρισμα του συστήματος PyTorch είναι το γεγονός ότι η εκτέλεση ενός προγράμματός είναι δυναμική. Συγκεκριμένα, το PyTorch δημιουργεί έναν δυναμικό υπολογιστικό γράφο, ο οποίος, πριν το μοντέλο μηχανικής μάθησης αρχίσει να εκτελείται, δεν υπάρχει. Κάθε φορά που εκτελεί μία συνάρτηση, τότε και δημιουργεί την αναπαράστασή της (**τεχνική υπερφόρτωσης τελεστών**). Έτσι, ο χρήστης δεν αναμένει την μεταγλώττιση του προγράμματός του για να αρχίσει να το εκτελεί, και κάθε ενδιάμεσος υπολογισμός μπορεί να παρατηρείται, επιτρέποντας την ευκολότερη κατανόηση του μοντέλου και τον εντοπισμό πιθανών σφαλμάτων.

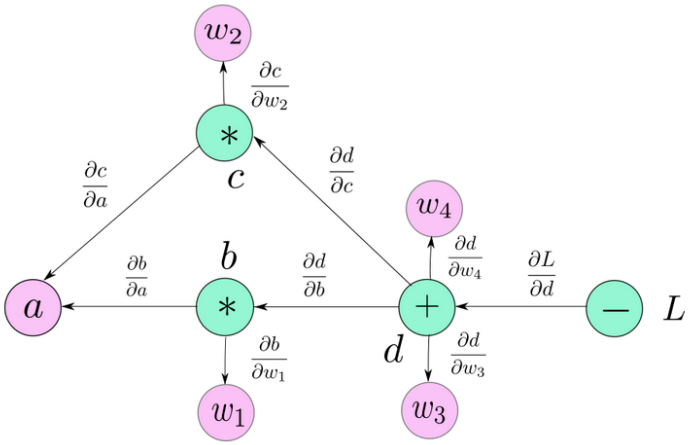
5.3.2 Υπολογισμός κλίσης



Εικόνα 5.4: Στάδιο τροφοδότησης (ή forward propagation) [52].

Για να υπολογίσει τις απαιτούμενες παραγώγους, το PyTorch χρησιμοποιεί την τυπική τεχνική της **αντίστροφης αυτόματης διαφόρισης**, που υλοποιεί, ουσιαστικά, τον αλγόριθμο back-propagation. Ο τυπικός τρόπος για να γίνει αυτό περιλαμβάνει την καταγραφή μίας «ταινίας» που περιγράφει την σειρά με την οποία πραγματοποιούνται οι λειτουργίες (operations), αποφεύγοντας, έτσι, την υλοποίηση τοπολογικής διάταξης. Το PyTorch αποφεύγει, εν μέρη, την μέθοδο αυτή με το να καταγράφει μόνο ένα υποσύνολο του υπολογιστικού γράφου, το οποίο σχετίζεται με τον υπολογισμό των παραγώγων [53], επιτρέποντας στους χρήστες να μπορούν να συνδυάζουν ανεξάρτητους γράφους, και, μάλιστα, να τους τοποθετούν σε διάφορα νήματα (threads), χωρίς να τα συγχρονίζουν οι ίδιοι. Ένα πλεονέκτημα αυτής της μεθόδου είναι ότι, όταν ένα κομμάτι του υπολογιστικού γράφου δεν είναι χρήσιμο, τότε ελευθερώνεται η αντίστοιχη μνήμη.

Στις εικόνα 5.4 βλέπουμε το στάδιο τύπου τροφοδότησης, κατά το οποίο δημιουργείται ο γράφος του νευρωνικού δικτύου στο PyTorch, ενώ στην εικόνα 5.5 φαίνεται η αντίστροφη πορεία, το στάδιο της οπισθοδρόμησης, κατά το οποίο, μάλιστα, ο γράφος καταστρέφεται απελευθερώνοντας μνήμη, εκτός αν έχει δοθεί από τον προγραμματιστή εντολή που να δηλώνει το αντίθετο.



Εικόνα 5.5: Στάδιο οπισθοδρόμησης (ή back-propagation) [52].

5.3.3 Υλοποίηση

Η βιβλιοθήκη PyTorch διαχωρίζει την ροή ελέγχου του προγράμματος (π.χ. διακλαδώσεις, βρόχοι επανάληψης) από την ροή των δεδομένων (π.χ. τανυστές και οι λειτουργίες πάνω σε αυτούς). Αναλυτικότερα, η ροή του ελέγχου διαχειρίζεται από κώδικα γραμμένο σε Python, και σε βελτιστοποιημένη C++, που εκτελείται σε μία κεντρική μονάδα επεξεργασίας (Central Processing Unit, ή CPU), ενώ, στη συνέχεια, προκαλείται η εκτέλεση μίας ακολουθίας από λειτουργίες πάνω σε CPU ή GPU.

Οι τελεστές της βιβλιοθήκης PyTorch είναι υλοποιημένοι, στην πλειοψηφία τους, σε γλώσσα C++. Ο λόγος που αποφεύγεται η συγγραφή τους σε Python είναι η αποφυγή του επιπλέον χρόνου εκτέλεσης που εισάγει ο διερμηνέας της. Ακόμη, δεδομένου ότι η Python διαθέτει ένα **καθολικό κλείδωμα του διερμηνέα (Global Interpreter Lock)** [54], ο οποίος περιορίζει τον έλεγχο του διερμηνέα αποκλειστικά σε ένα νήμα, η συγγραφή των operators σε C++ αναιρεί το πρόβλημα αυτό και, επομένως, επιτρέπει την εκτέλεση του κώδικα σε πολλαπλά νήματα, γεγονός που είναι πολύ σημαντικό για την περίπτωση που είναι επιθυμητή η χρήση των GPUs. Λόγω της φύσης της Python, ο παραπάνω μηχανισμός είναι σχεδόν αόρατος προς τον χρήστη.

Μία σημαντική χρήση του PyTorch είναι η εκτέλεση σε GPU, η οποία, όμως, υστερεί στην ύπαρξη αρκετής διαθέσιμης μνήμης. Επομένως, η βιβλιοθήκη PyTorch φροντίζει να απομακρύνει κάθε ενδιάμεση τιμή, με το που δεν είναι, πλέον, χρήσιμη. Αυτό το καταφέρνει με τη χρήση συλλέκτη σκουπιδιών του τύπου **μετρητή αναφοράς (reference counting)** [55]. Επιπλέον, υποστηρίζει επί-τόπου (in-place) λειτουργίες πάνω σε τανυστές, ώστε να μη δεσμεύει μνήμη για κάποιον νέο τανυστή, τον οποίο γνωρίζει, εκ των προτέρων, ότι δεν χρειάζεται. Συνοπτικά, αυτό το υλοποιεί κρατώντας έναν μετρητή έκδοσης για κάθε τανυστή, ακυρώνοντας προηγούμενες τιμές όταν αλλάζει η τιμή του τανυστή, ενώ φροντίζει τις αλλαγές σε κάποιον τανυστή να τις μεταφέρει σε όλους τους εξαρτημένους τανυστές, δηλαδή τανυστές με βάση τους οποίους ορίστηκε και αποτελεί, ουσιαστικά, μία μετονομασία τους (alias).

5.3.4 Κατανεμημένη εκτέλεση

Όπως και το TensorFlow, το PyTorch προσφέρει δύο μεθόδους κατανεμημένης εκτέλεσης [56], την παραλληλοποίηση δεδομένων και την παραλληλοποίηση μοντέλου.

Το API που εξυπηρετεί την **παραλληλοποίηση δεδομένων** σε πολλαπλά μηχανήματα είναι το DistributedDataParallel [57], το οποίο υλοποιεί μια σύγχρονη εκπαίδευση του δικτύου. Με βάση αυτό, κάθε διεργασία είναι τύπου worker διεργασία, και διατηρεί ένα πλήρες αντίγραφο των βαρών του μοντέλου. Σε κάθε βήμα, το αποτέλεσμα της εκπαίδευσης στο αντίστοιχο κομμάτι των δεδομένων εκπαίδευσης συναθροίζεται με των υπολοίπων, ενώ η επικοινωνία πραγματοποιείται, πάλι, με κάποιον αλγόριθμο τύπου All-reduce. Συνοπτικά:

1. Κάθε worker διατηρεί το δικό του αντίγραφο τόσο των βαρών όσο και των δεδομένων.

2. Μόλις λάβει το κατάλληλο σήμα, κάθε worker λαμβάνει ένα κομμάτι (batch) από τα δεδομένα και υπολογίζει την αντίστοιχη κλίση.

3. Οι workers χρησιμοποιούν τον All-reduce αλγόριθμο για να συγχρονιστούν.

4. Κάθε worker ενημερώνει την κλίση στο δικό του αντίγραφο του μοντέλου.

5. Επαναλαμβάνεται η ίδια διαδικασία με το επόμενο batch των δεδομένων.

Σε περίπτωση που χρειάζεται εκπαίδευση σε ένα μηχάνημα, είναι διαθέσιμο το DataParallel API [58] (συστήνεται, όμως, και πάλι η χρήση του DistributedDataParallel για λόγους απόδοσης), ενώ για **ασύγχρονη** εκπαίδευση ή **παραλληλοποίηση μοντέλου**, χρησιμοποιείται το κατανεμημένο RPC σύστημα [59].

Κεφάλαιο 6

Αρχιτεκτονική Συστήματος Πρόβλεψης

6.1 Περιγραφή

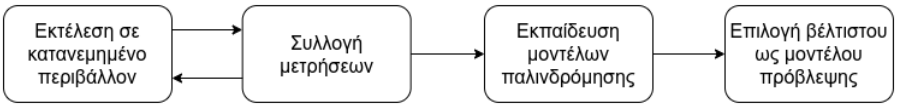
Στόχος της παρούσας διπλωματικής εργασίας είναι η δημιουργία ενός μοντέλου που να μπορεί να προβλέψει τον χρόνο εκτέλεσης των κυριότερων τελεστών του κάθε επιπέδου ενός νευρωνικού δικτύου σε κατανεμημένο περιβάλλον, οι οποίοι πραγματοποιούν τους υπολογισμούς τους σε δύο φάσεις της εκπαίδευσης, αφενός σε αυτήν της τροφοδότησης του δικτύου και αφετέρου στην φάση της οπισθοδρόμησης. Προκειμένου να κατασκευάσουμε το επιθυμητό μοντέλο πρόβλεψης, ήταν απαραίτητο να συλλέξουμε ένα σύνολο δεδομένων που αποτελείται από τους χρόνους εκτέλεσης των παραπάνω τελεστών, για ποικίλους συνδυασμούς τιμών των παραμέτρων εισόδου.

Η γλώσσα προγραμματισμού που χρησιμοποιήσαμε είναι η Python, ενώ για την υλοποίηση των δικτύων χρησιμοποιήσαμε τα συστήματα TensorFlow και PyTorch, σε συνδυασμό με τα αντίστοιχα εργαλεία τους προκειμένου να εξάγουμε τους επιθυμητούς χρόνους εκτέλεσης των τελεστών. Για την πραγματοποίηση της εργασίας μας είχαμε στη διάθεσή μας συνολικά τρία μηχανήματα, το καθένα από τα οποία διαθέτει 4 πυρήνες CPU και 16 GB μνήμης RAM.

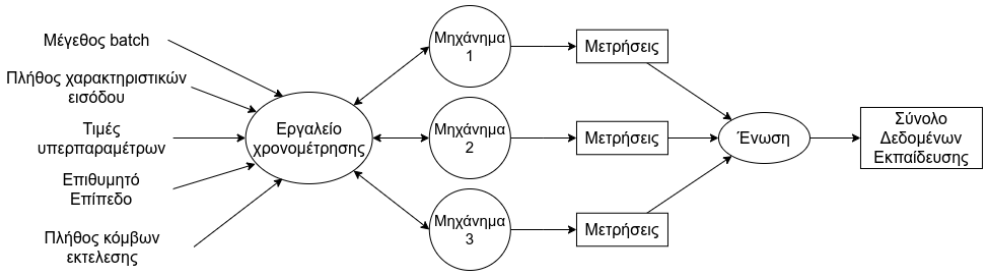
Οι παράμετροι ως προς τις οποίες μελετήσαμε την συμπεριφορά των δικτύων, για κάθε είδος επιπέδου (layer) του κάθε συστήματος υλοποίησής τους (TensorFlow ή PyTorch), είναι οι εξής:

1. Πλήθος μηχανημάτων εκτέλεσης
2. Μέγεθος των τεμαχίων (batch size) των δεδομένων εκπαίδευσης
3. Πλήθος χαρακτηριστικών (number of features) εισόδου στο νευρωνικό δίκτυο
4. Συνδυασμός τιμών των υπερπαραμέτρων για το κάθε επιπέδων στο δίκτυο

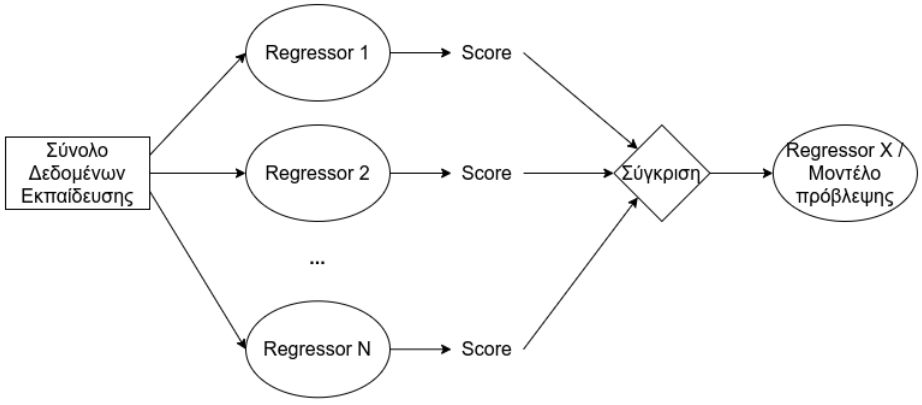
Στην συνέχεια, τα δεδομένα που συγκεντρώσαμε σε μορφή αρχείου τροφοδοτούνται στον προβλεπτή, ο οποίος είναι, ουσιαστικά, ένα εκπαιδευμένο μοντέλο παλινδρόμησης (regressor). Καθώς υπάρχει πληθώρα τέτοιων μοντέλων, εξετάσαμε την απόδοση πολλαπλών από αυτά, με το βέλτιστο, ως προς το σκορ τους, να αποτελεί και τον τελικό μας προβλεπτή χρόνου (time predictor). Σχηματικά, η διαδικασία που ακολουθήσαμε φαίνεται στις εικόνες 6.1, 6.2 και 6.3.



Εικόνα 6.1: Συνολική διαδικασία.



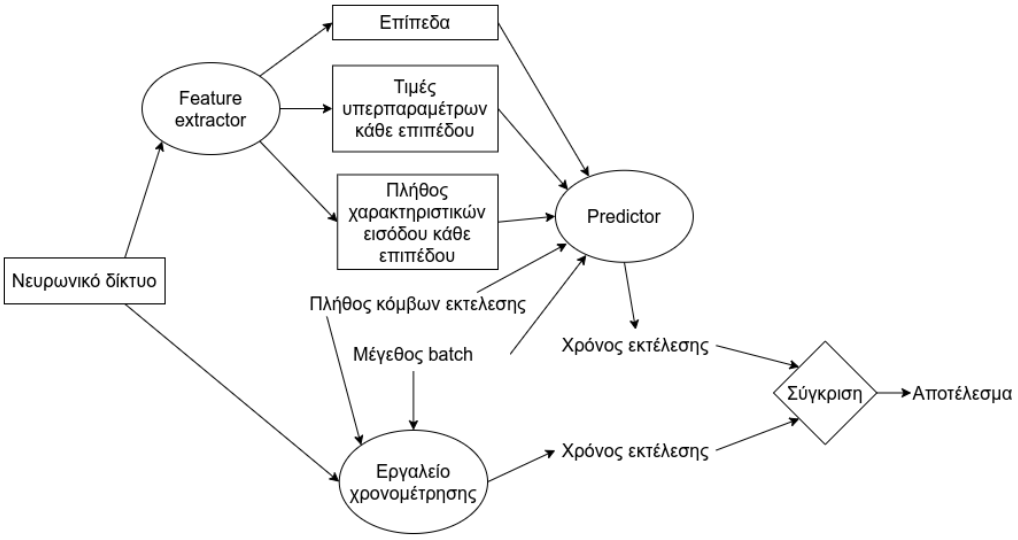
Εικόνα 6.2: Συλλογή μετρήσεων.



Εικόνα 6.3: Εξαγωγή μοντέλου πρόβλεψης.

Το τελικό στάδιο της εργασίας μας, ύστερα, δηλαδή, από τη συλλογή των απαραίτητων δεδομένων και την επιλογή του regressor, αποσκοπεί στην αξιολόγηση του μοντέλου πρόβλεψης σε κάποια νευρωνικά δίκτυα της επιλογής μας. Συγκεκριμένα, έχοντας υλοποιήσει σε κάποιο αρχείο ένα δίκτυο, συλλέγουμε τους

χρόνους εκτέλεσης των τελεστών με τα εργαλεία προγραμματισμού που χρησιμοποιήσαμε σε προηγούμενο στάδιο, για διάφορους συνδυασμούς του πλήθους μηχανημάτων και του μεγέθους των τεμαχίων. Τέλος, προκειμένου να λάβουμε την πρόβλεψη του μοντέλου μας ώστε να συγκρίνουμε τα αποτελέσματα, για τους ίδιους συνδυασμούς τιμών, είναι απαραίτητο να δοθούν στον προβλεπτή τα χαρακτηριστικά του παραπάνω δικτύου, τα οποία τα λαμβάνουμε με τη βοήθεια μιας κατάλληλης συνάρτησης εξαγωγής χαρακτηριστικών (feature extractor), υλοποιημένη σε Python (εικόνα 6.4).



Εικόνα 6.4: Πρόβλεψη και σύγκριση.

6.2 Συλλογή Μετρήσεων

Η συλλογή των μετρήσεων στοχεύσαμε να πραγματοποιηθεί πάνω σε διάφορα επίπεδα ενός συνελικτικού νευρωνικού δικτύου. Επομένως, υλοποιήσαμε νευρωνικά δίκτυα, τόσο με TensorFlow όσο και με PyTorch, το καθένα από τα οποία διαθέτει ένα επίπεδο από κάθε κατηγορία. Η είσοδος για την εκπαίδευσή τους αποτελούσε ένα σύνολο δεδομένων τυχαίων αριθμών, με διάφορα μεγέθη, καθώς μας ενδιαφέρει μόνο ο χρόνος εκτέλεσης και όχι η ακρίβεια ταξινόμησης του νευρωνικού δικτύου. Ύστερα, σε κάθε εκτέλεση, για το κάθε σύστημα χρησιμοποιήσαμε το αντίστοιχο εργαλείο τους για να εξάγουμε μόνο τους χρόνους εκτέλεσης των τελεστών που σχετίζονται με το επίπεδο που μας ενδιαφέρει, και να τους αποθηκεύσουμε σε κάποιο αρχείο, πριν προχωρήσουμε σε κάποια επόμενη εκτέλεση.

Σε κάθε γραμμή των αποθηκευμένων αρχείων παραθέτονται οι πληροφορίες που σχετίζονται με την είσοδο στο εργαλείο χρονομέτρησης, δηλαδή:

- πλήθος εποχών

- μέγεθος συνόλου δεδομένων εκπαίδευσης
- πλήθος χαρακτηριστικών εισόδου (ή αλλιώς μέγεθος εικόνας)
- πλήθος καναλιών εισόδου
- μέγεθος τεμαχίων
- πλήθος μηχανημάτων εκτέλεσης
- συνδυασμός τιμών υπερπαραμέτρων

Για κάθε είδος επιπέδου, κρατήσαμε και από ένα αρχείο, το οποίο και διαθέτει το δικό του συνδυασμό υπερπαραμέτρων. Στον πίνακα 6.1 παρουσιάζονται τα επίπεδα τα οποία παρακολουθήσαμε, και οι υπερπαράμετροι που τους αντιστοιχούν, τις τιμές των οποίων και αποθηκεύσαμε στην κατάλληλη θέση του αρχείου τους.

Επίπεδο	Υπερπαράμετρος
Convolutional	μέγεθος πυρήνα, βήμα ολίσθησης, πλήθος φίλτρων εξόδου
Average & Max Pooling	μέγεθος συγκέντρωσης, βήμα ολίσθησης
Dropout	πιθανότητα εγκατάλειψης
Fully-connected	πλήθος μονάδων
ReLU	-
Tanh	-
Batch Normalization	-
Flatten	-

Πίνακας 6.1: Επίπεδα και υπερπαράμετροι

Σημειώνουμε ότι για τη συλλογή των μετρήσεών μας, περιορίσαμε τις τιμές των παραμέτρων και υπερπαραμέτρων σε ένα καθορισμένο σύνολο, το οποίο και φαίνεται στον πίνακα 6.2, λόγω του περιορισμού του διαθέσιμου χρόνου και πόρων. Κάθε εκτέλεση πραγματοποιήθηκε για συνολικά 5 εποχές, πάνω σε ένα σύνολο δεδομένων από 4608 στοιχεία.

Σε κάθε επίπεδο, η είσοδος είναι μία εικόνα, με μήκος αλλά και πλάτος ίσο με το «πλήθος χαρακτηριστικών εισόδου», η οποία διαθέτει και κανάλια, το πλήθος των οποίων είναι ίσο με το «πλήθος καναλιών εισόδου». Με άλλα λόγια, το μέγεθός της υπολογίζεται ως:

$$S(\textit{εικόνα}) = N^2(\textit{χαρακτηριστικά εισόδου}) \times N(\textit{κανάλια}) \tag{6.1}$$

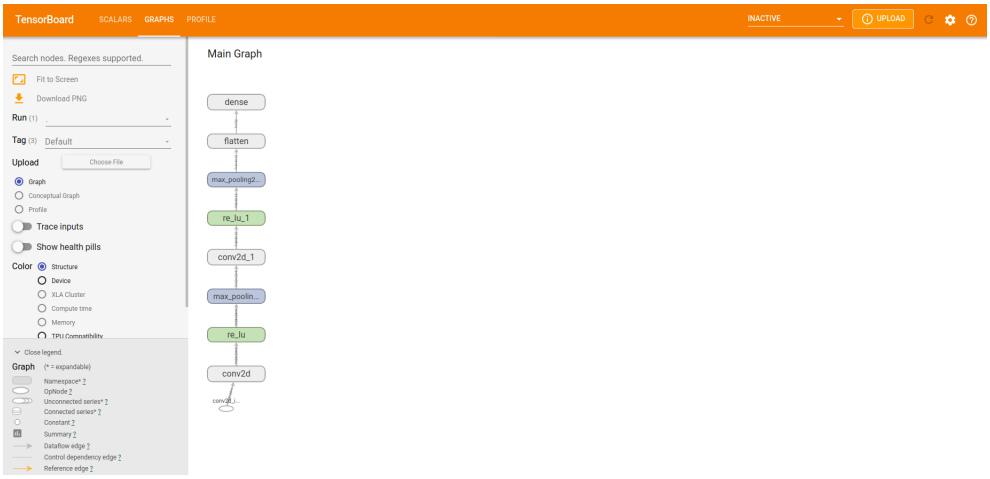
όπου με $N(\cdot)$ και με $S(\cdot)$ συμβολίζουμε το πλήθος και το μέγεθος κάποιου στοιχείου, αντίστοιχα.

(Υπερ)παράμετρος	Τιμές
πλήθος χαρακτηριστικών εισόδου	16, 32, 64
πλήθος καναλιών εισόδου	1, 2, 4, 6, 8, 16
μέγεθος τεμαχίων	16, 32, 64, 128, 256, 512
πλήθος μηχανημάτων εκτέλεσης	1, 2, 3
μέγεθος πυρήνα	2, 4, 8
πλήθος φίλτρων εξόδου	1, 2, 4, 8, 16
βήμα ολίσθησης	1, 2, 4
μέγεθος συγκέντρωσης	2, 4, 8
πιθανότητα εγκατάλειψης	0.2, 0.4, 0.8
πλήθος μονάδων	8, 16, 32, 64, 128, 256, 512, 1024

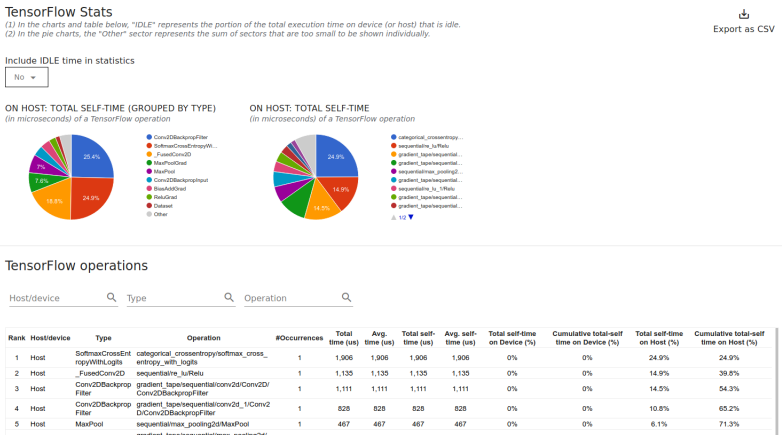
Πίνακας 6.2: Τιμές παραμέτρων και υπερπαραμέτρων

6.2.1 TensorFlow

Το εργαλείο συλλογής χρόνων εκτέλεσης για το TensorFlow το οποίο χρησιμοποιήσαμε είναι το TensorBoard [60], και μπορεί να εξυπηρετήσει ποικίλους σκοπούς, όπως είναι η παρακολούθηση της απώλειας και της ακρίβειας του δικτύου με γραφική παράσταση, η οπτικοποίηση του νευρωνικού δικτύου με τη μορφή γράφου, και, φυσικά, η καταμέτρηση χρόνων εκτέλεσης τελεστών. Υποστηρίζεται, μάλιστα, και η χρήση του από το PyTorch [61], με σκοπό την εξαγωγή αντίστοιχων πληροφοριών. Για παράδειγμα, στην εικόνα 6.5 μπορούμε να δούμε την αναπαράσταση του γράφου του δικτύου LeNet-1, ενώ στην εικόνα 6.6 απεικονίζεται ένας πίνακας που περιλαμβάνει τους χρόνους εκτέλεσης των τελεστών του δικτύου.



Εικόνα 6.5: Απεικόνιση γράφου με τη βοήθεια του TensorBoard.



Εικόνα 6.6: Πίνακας χρόνων εκτέλεσης του TensorBoard.

Για να συλλέξουμε τις επιθυμητές μετρήσεις, φροντίσαμε να «σηκώσουμε» πρωτίστως σε κάθε μηχάνημα το TensorBoard σε κάποια θύρα, και να ορίσουμε μία διαδρομή στην οποία θα αποθηκεύονται οι κατάλληλες πληροφορίες. Στη συνέχεια, με το πέρας μίας εκτέλεσης, λαμβάνουμε τον πίνακα με τους χρόνους εκτέλεσης σε μορφή αρχείου «csv», επικοινωνώντας με τη θύρα του TensorBoard. Από το αρχείο αυτό, κρατήσαμε τις μετρήσεις οι οποίες αφορούσαν τους επιθυμητούς τελεστές, και τους αποθηκεύσαμε με τη σειρά τους, σε αρχείο csv και πάλι.

Τα ονόματα των τελεστών των επιπέδων με τους οποίους ασχοληθήκαμε στο TensorFlow παρουσιάζονται στον πίνακα 6.3.

Επίπεδο	Τελεστής
Convolutional 2D	_FusedConv2D, Conv2DBackpropFilter
Average Pooling 2D	AvgPool
Max Pooling 2D	MaxPool
Fully Connected	MatMul, _FusedMatMul
Dropout 2D	RandomUniform
Batch Normalization 2D	FusedBatchNormV3, FusedBatchNormGradV3
ReLU 2D	Relu
Tanh 2D	Tanh

Πίνακας 6.3: Τελεστές TensorFlow

6.2.2 PyTorch

Το εργαλείο καταμέτρησης στο PyTorch είναι το API με όνομα «autograd.profiler» [62], το οποίο υπηρετεί, όπως και το TensorBoard, διάφορους

σκοπούς, όπως η καταγραφή του χρόνου εκτέλεσης των τελεστών που έχουν κληθεί, αλλά και ο εντοπισμός της σειράς των κλήσεων αυτών, και η παρακολούθηση της μνήμης που χρησιμοποιήθηκε. Ο profiler του PyTorch δεν αποθηκεύει από μόνος του τις πληροφορίες του σε κάποια διαδρομή, αλλά τις επιστρέφει με τη μορφή δομής δεδομένων κατά την εκτέλεση , από την οποία λαμβάνουμε το συνολικό χρόνο εκτέλεσης των τελεστών που μας ενδιαφέρουν, και τον αποθηκεύουμε σε csv αρχείο.

Τα ονόματα των τελεστών των επιπέδων με τους οποίους ασχοληθήκαμε στο PyTorch παρουσιάζονται στον πίνακα 6.4.

Επίπεδο	Τελεστής
Convolutional 2D	conv2d, MkldnnConvolutionBackward
Average Pooling 2D	avg_pool2d
Max Pooling 2D	max_pool2d
Fully Connected	addmm, AddmmBackward
Dropout 2D	feature_dropout
Batch Normalization 2D	batch_norm, NativeBatchNormBackward
ReLU 2D	relu
Tanh 2D	tanh

Πίνακας 6.4: Τελεστές PyTorch

6.3 Προβλεπτής Χρόνου

Σε αυτό το στάδιο της εργασίας, αξιοποιούμε τα δεδομένα που προέκυψαν κατά τη συλλογή, τροφοδοτώντας τα σε κάποια μοντέλα παλινδρόμησης. Ωστόσο, προκειμένου να μπορούμε να συγκρίνουμε κάθε φορά τους χρόνους εκτέλεσης μεταξύ τους, τα δεδομένα υπόκεινται σε μία φάση προεπεξεργασίας. Ειδικότερα, κλιμακώνουμε κάθε έναν από τους χρόνους με τρόπο ώστε να αναλογεί σε ένα βήμα εκπαίδευσης του νευρωνικού δικτύου, τους διαιρούμε δηλαδή με τον αριθμό των συνολικών βημάτων, που υπολογίζονται από τη σχέση:

$$N(\beta\acute{\eta}\mu\alpha\tau\alpha) = N(\epsilon\pi o\chi\acute{\epsilon}\varsigma) \times \frac{S(\sigma\acute{\upsilon}\nu o\lambda o\ \delta\epsilon\delta o\mu\acute{\epsilon}\nu\omega\nu)}{S(\delta\acute{\epsilon}\sigma\mu\eta\ \tau\epsilon\mu\alpha\chi\acute{\iota}\omega\nu))} \tag{6.2}$$

Εφόσον έχουμε λάβει csv αρχεία για κάθε επίπεδο, κατασκευάζουμε έναν regressor για κάθε ένα από αυτά. Έτσι, για το κάθε ένα framework, και το κάθε ένα επίπεδο θα έχουμε από έναν διαφορετικό regressor. Επίσης, λάβαμε διαφορετικό regressor για κάθε συνδυασμό πλήθους μηχανημάτων, καθώς τα πλήθη μηχανημάτων θα είναι πάντα από ένα έως τρία, δηλαδή δεν θα κάνουμε κάποια πρόβλεψη για 4 ή περισσότερα μηχανήματα, διότι δεν θα μπορούμε να την ελέγξουμε. Για παράδειγμα, για τους χρόνους του συνελικτικού επιπέδου

υλοποιημένο σε TensorFlow, το οποίο είχε εκπαιδευτεί σε 3 μηχανήματα, θα έχουμε τον regressor με όνομα «conv2d-tensorflow-3».

Τα μοντέλα παλινδρόμησης ανάμεσα στα οποία θα επιλέξουμε το τελικό προβλεπτή θα είναι όλα όσα έχουν αναφερθεί στο κεφάλαιο 2. Τα σκορ με βάση τα οποία θα συγκρίνουμε τα μοντέλα θα είναι ο συντελεστής προσδιορισμού και η ρίζα τετραγωνικού σφάλματος. Μόλις έχουμε επιλέξει τα κατάλληλα μοντέλα παλινδρόμησης, μας απομένει να υλοποιήσουμε έναν ολικό προβλεπτή, ο οποίος λαμβάνει ως είσοδο τη δομή ενός νευρωνικού δικτύου, και για το κάθε επίπεδο του δικτύου επιλέγει το σωστό regressor και εξάγει την πρόβλεψη με βάση τα χαρακτηριστικά του. Η τελική πρόβλεψη θα είναι το άθροισμα των προβλέψεων όλων των επιπέδων εισόδου. Για να επιτύχουμε το παραπάνω, κατασκευάσαμε μία συνάρτηση που να λαμβάνει ως είσοδο μία κλάση της Python, στην οποία είναι υλοποιημένο το προς-έλεγχο νευρωνικό δίκτυο, και να δίνει ως έξοδο μία λίστα που να περιλαμβάνει το όνομα του κάθε επιπέδου και οποιαδήποτε άλλη σημαντική πληροφορία που σχετίζεται αυτό.

Αν συμβολίσουμε με N το εξεταζόμενο νευρωνικό δίκτυο, με f τη συνάρτηση εξαγωγής χαρακτηριστικών και, τέλος, με R τον regressor που επιλέξαμε ως προβλεπτή, τότε η πρόβλεψη P του μοντέλου μας θα είναι η:

$$P(N) = \sum_i R(X_i), \text{ όπου } [X_i] = f(N), \quad \text{και } R(X_i) \in \mathbb{R}_+ \tag{6.3}$$

Κεφάλαιο 7

Αποτελέσματα

7.1 Γραφικές Παραστάσεις των Δεδομένων της Συλλογής

Στις παρακάτω εικόνες παρουσιάζουμε τα αποτελέσματα που προέκυψαν από την συλλογή δεδομένων με τη μορφή γραφικών παραστάσεων. Κάθε μία από αυτές, αφορά κάποιο επίπεδο και περιλαμβάνει τον χρόνο εκτέλεσης ανά βήμα (κάθετος άξονας) συναρτήσει μία εκ των παραμέτρων του (οριζόντιος άξονας). Στην πραγματικότητα, ο χρόνος εκτέλεσης κάθε επιπέδου είναι συνάρτηση πολλών μεταβλητών. Επομένως, για τη δική μας διευκόλυνση και με σκοπό να διατηρήσουμε την δισδιάστατη απεικόνισή του, λάβαμε τους μέσους όρους του χρόνου, ως προς όλες τις υπόλοιπες μεταβλητές εκτός αυτής που θα αντιστοιχεί στον οριζόντιο άξονα. Επιπλέον, στο ίδιο διάγραμμα φαίνονται οι χρόνοι εκτέλεσης με ένα, δύο και τρία μηχανήματα, και έχουν σημειωθεί με διαφορετικό χρώμα αλλά και σύμβολο.

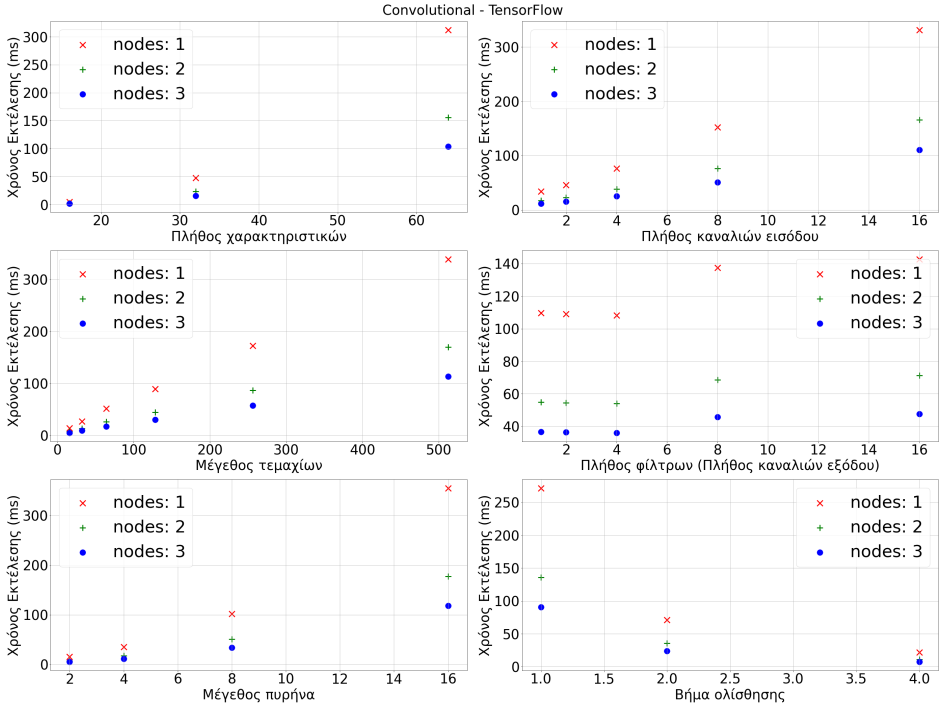
7.1.1 TensorFlow

Σε όλα τα γραφήματα εκτός από το 7.5, παρατηρούμε ότι η σχέση του μέσου χρόνου εκτέλεσης με το πλήθος των χαρακτηριστικών εισόδου είναι εκθετικά αυξανόμενη, γεγονός όμως που οφείλεται στο ότι, στη δική μας αρχιτεκτονική, αν η είσοδος στο δίκτυο έχει πλήθος χαρακτηριστικών n, τότε το μέγεθος της εισόδου σε pixels, αφού πρόκειται για εικόνα δύο διαστάσεων, είναι n^2. Αν στα γραφήματα ο οριζόντιος άξονας, αντί για πλήθος χαρακτηριστικών, είχαμε το πλήθος των pixels εισόδου, τότε θα παρατηρούσαμε γραμμική σχέση.

Ως προς το μέγεθος των τεμαχίων, ο μέσος χρόνος αυξάνει γραμμικά, κάτι που περιμέναμε, αφού με μεγαλύτερο μέγεθος τεμαχίου έχουμε μεγαλύτερους τανυστές ως είσοδο σε κάθε βήμα, και, συνεπώς, εκτελούνται περισσότερες πράξεις.

Σχετικά με το πλήθος των καναλιών εισόδου, ο μέσος χρόνος σε όλα τα γραφήματα φαίνεται να αυξάνει γραμμικά. Ωστόσο, στα επίπεδα συγκέντρωσης

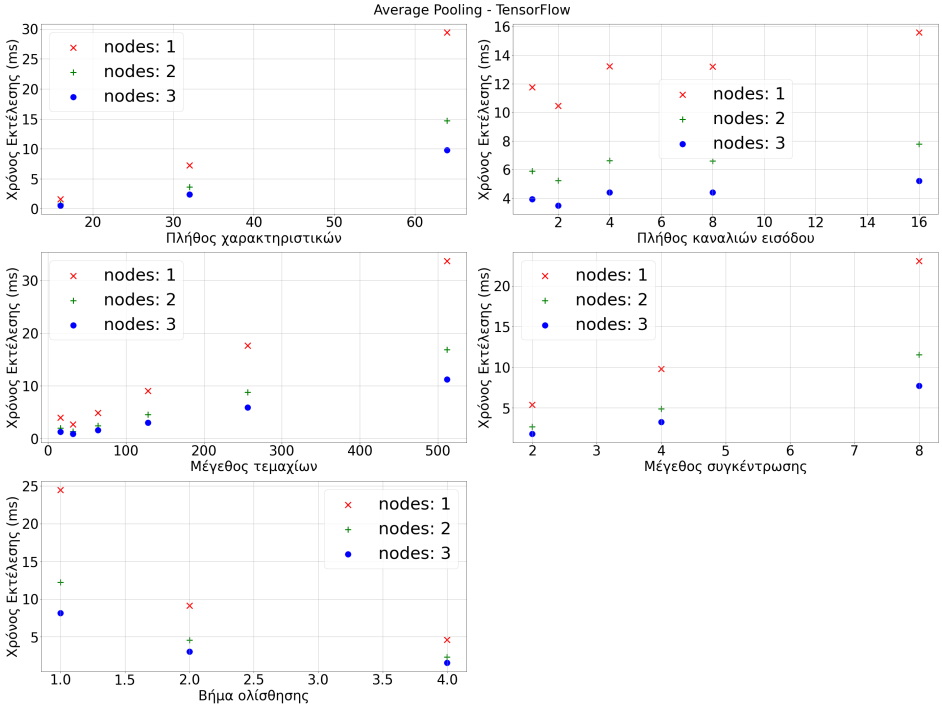
(εικόνες 7.2, 7.3) και στο επίπεδο κανονικοποίησης (εικόνα 7.6), η σχέση του χρόνου με το πλήθος καναλιών δεν φαίνεται να είναι σαφής. Αυτό μπορεί να οφείλεται σε ποικίλους λόγους, όπως είναι η εισαγωγή ανεπιθύμητου θορύβου στο πείραμα, ή, ακόμα, μπορεί η υλοποίηση του μηχανισμού τους να είναι αρκετά πιο πολύπλοκη. Προκειμένου να κατανοήσουμε την συμπεριφορά τους, είναι απαραίτητο να εκτελέσουμε το πείραμα για περισσότερες τιμές του πλήθους καναλιών, αλλά και για μεγαλύτερο εύρος. Υποπτευόμαστε, όμως, ότι για μεγαλύτερο πλήθος καναλιών εισόδου, ο χρόνος εκτέλεσης θα αυξάνεται, είτε με σταθερό, είτε με μεταβαλλόμενο ρυθμό, καθώς περισσότερα κανάλια οδηγούν σε μεγαλύτερο φόρτο πράξεων του αντίστοιχου τελεστή.



Εικόνα 7.1: Συνελικτικό επίπεδο - TensorFlow

Στο συνελικτικό επίπεδο (εικόνα 7.1), αλλά και στα επίπεδα συγκέντρωσης (εικόνες 7.2, 7.3), παρατηρούμε ότι ο χρόνος σε σχέση με το μέγεθος του πυρήνα και το μέγεθος συγκέντρωσης, αντίστοιχα, αυξάνει εκθετικά. Σημειώνουμε ότι τα μεγέθη αυτά είναι, ουσιαστικά, το μέγεθος του δισδιάστατου πίνακα που θα αποτελεί τμήμα της εικόνας εισόδου, πάνω στο οποίο θα γίνεται η κατάλληλη πράξη του αντίστοιχου τελεστή σε κάθε βήμα του. Αν, λοιπόν, είναι n το μέγεθος του πυρήνα ή της συγκέντρωσης, τότε το μέγεθος του πίνακα αυτού θα είναι $n \times n$, και ο τελεστής θα έχει να εκτελέσει n^2 πράξεις, σε κάθε βήμα του. Όσον αφορά το βήμα ολίσθησης του πίνακα, τότε όσο μεγαλύτερο είναι, τόσο λιγότερα θα είναι και τα συνολικά βήματα του τελεστή, και, έτσι, η σχέση του χρόνου εκτέλεσης προκύπτει αντιστρόφως ανάλογη των τιμών της παραμέτρου αυτής.

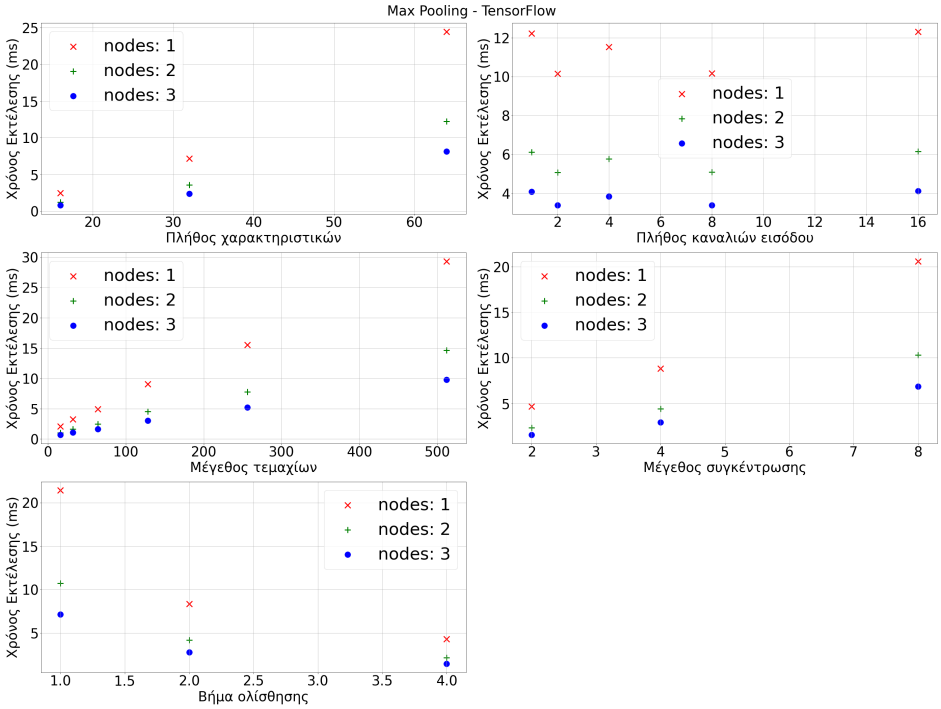
Σχετικά με το συνελικτικό επίπεδο (εικόνα 7.1), ο χρόνος εκτέλεσης σε σχέση με το πλήθος των φίλτρων, ή, ισοδύναμα, το πλήθος των καναλιών της εικόνας εξόδου, τείνει να αυξάνει. Και πάλι, μεγαλύτερο εύρος και περισσότερες τιμές στα φίλτρα εξόδου, είναι αναγκαίες για καλύτερη κατανόηση του ρυθμού αύξησης.



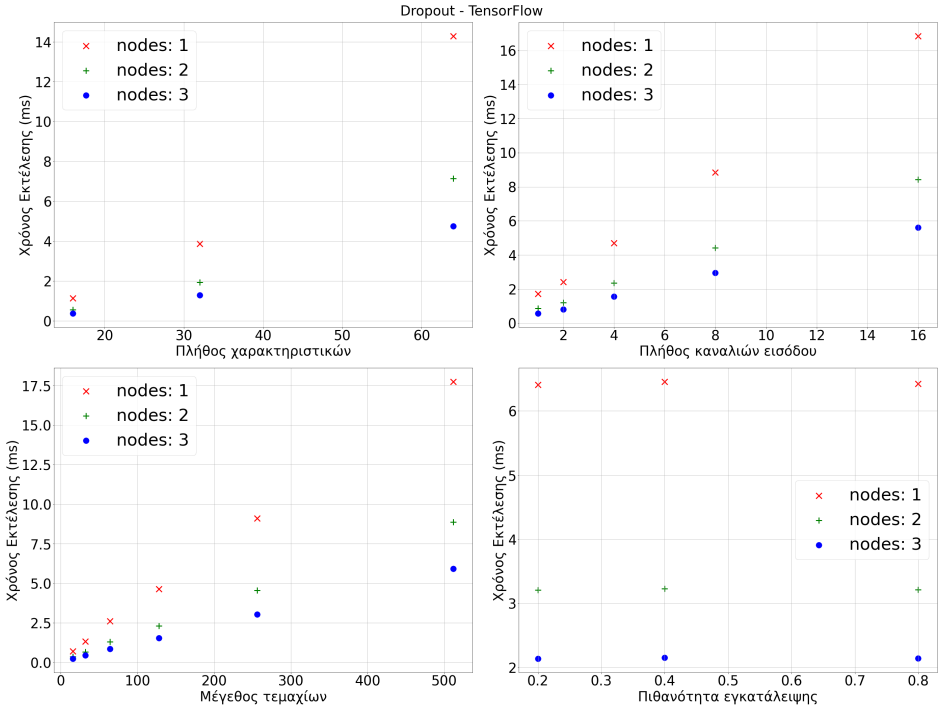
Εικόνα 7.2: Επίπεδο συγκέντρωσης μέσης τιμής - TensorFlow

Στο γράφημα με το επίπεδο εγκατάλειψης (7.4), παρατηρούμε ότι ο χρόνος εκτέλεσης είναι σταθερός ως προς την πιθανότητα εγκατάλειψης, εφόσον, πράγματι, η αύξηση της πιθανότητας δεν επηρεάζει το πλήθος των πράξεων που μπορεί να εκτελούνται.

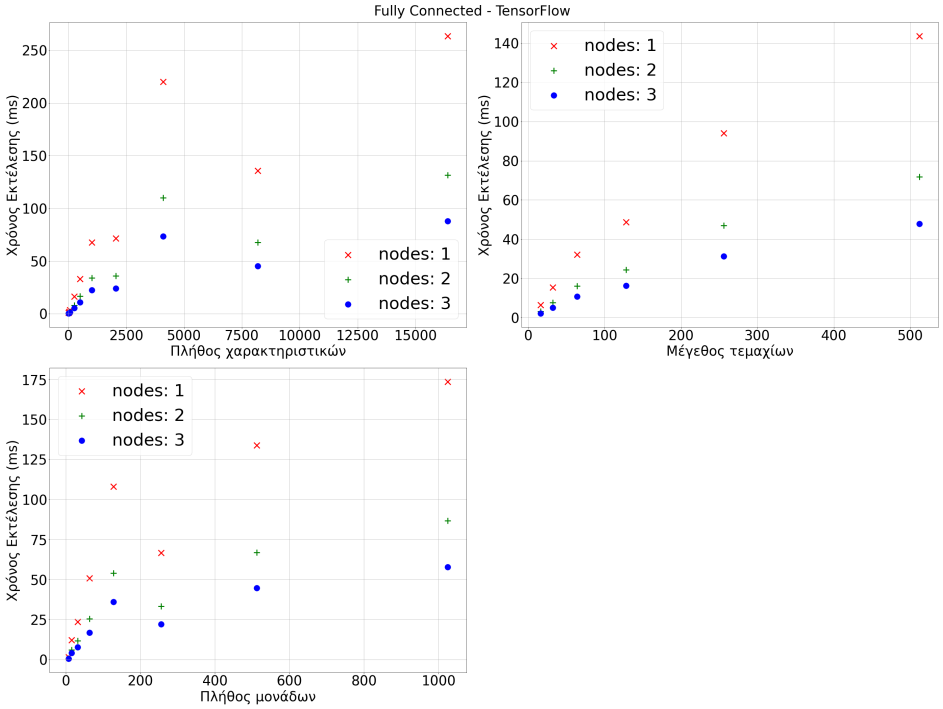
Στην εικόνα 7.5 που αφορά το πλήρως συνδεδεμένο επίπεδο, η σχέση δείχνει να είναι σχεδόν γραμμική, εφόσον το επίπεδο αυτό δεν λαμβάνει είσοδο δύο διαστάσεων, αλλά έναν μονοδιάστατο πίνακα, ο οποίος, στην δική μας περίπτωση, περιλαμβάνει τις τιμές των pixels της εικόνας σε σειριακή μορφή. Επιπλέον, το πλήθος των μονάδων αποτελεί και το πλήθος των τιμών εξόδου του επιπέδου, το οποίο επηρεάζει το πλήθος των πράξεων που θα εκτελούνται. Ειδικότερα, περισσότερες μονάδες σημαίνουν μεγαλύτερο χρόνο εκτέλεσης, γεγονός που αιτιολογεί την τάση του χρόνου να αυξάνεται, αν και όχι απαραίτητα σταθερά, σε σχέση με το πλήθος των μονάδων εξόδου.



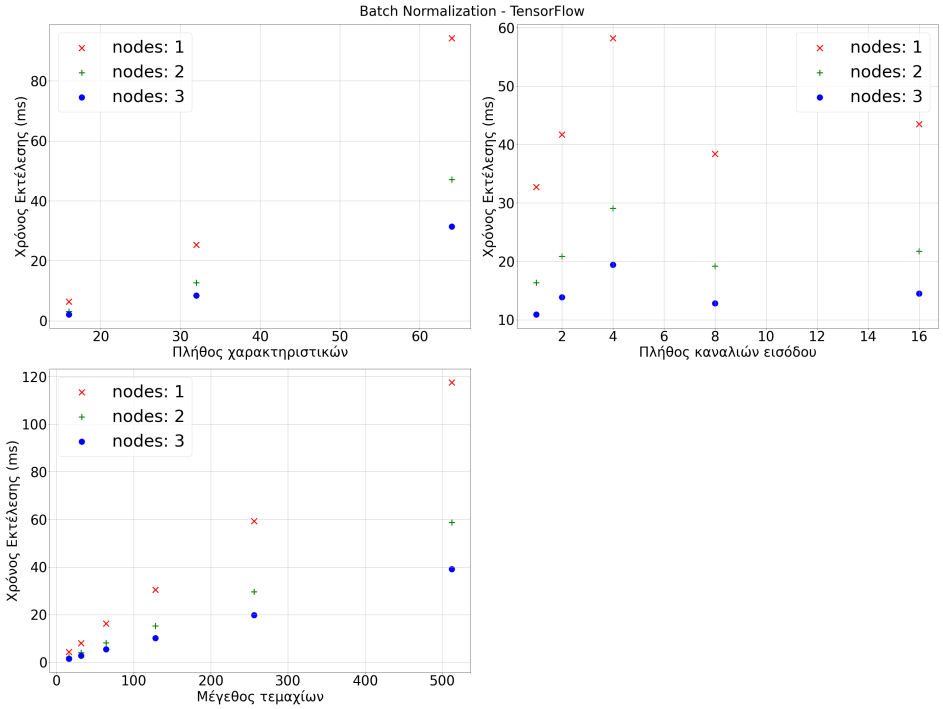
Εικόνα 7.3: Επίπεδο συγκέντρωσης μέγιστης τιμής - TensorFlow



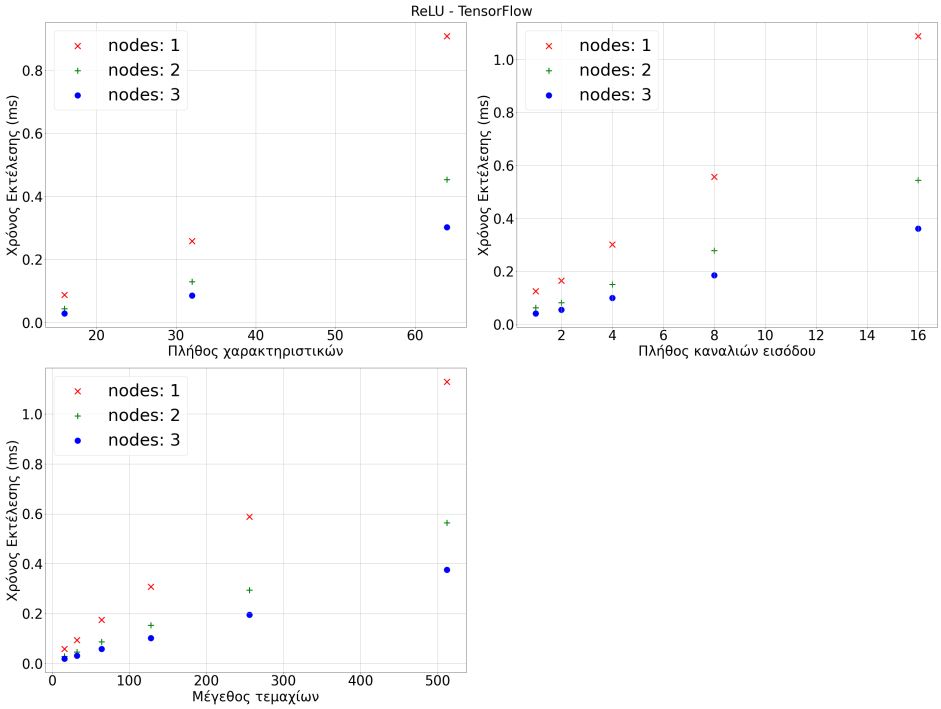
Εικόνα 7.4: Επίπεδο εγκατάλειψης - TensorFlow



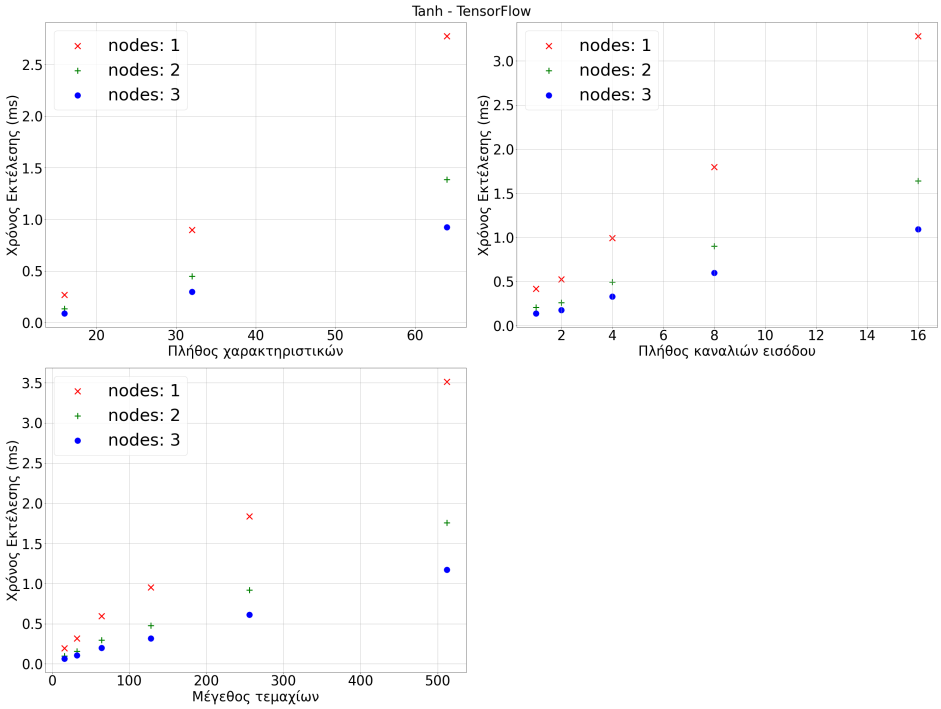
Εικόνα 7.5: Πλήρως συνδεδεμένο επίπεδο - TensorFlow



Εικόνα 7.6: Επίπεδο κανονικοποίησης - TensorFlow

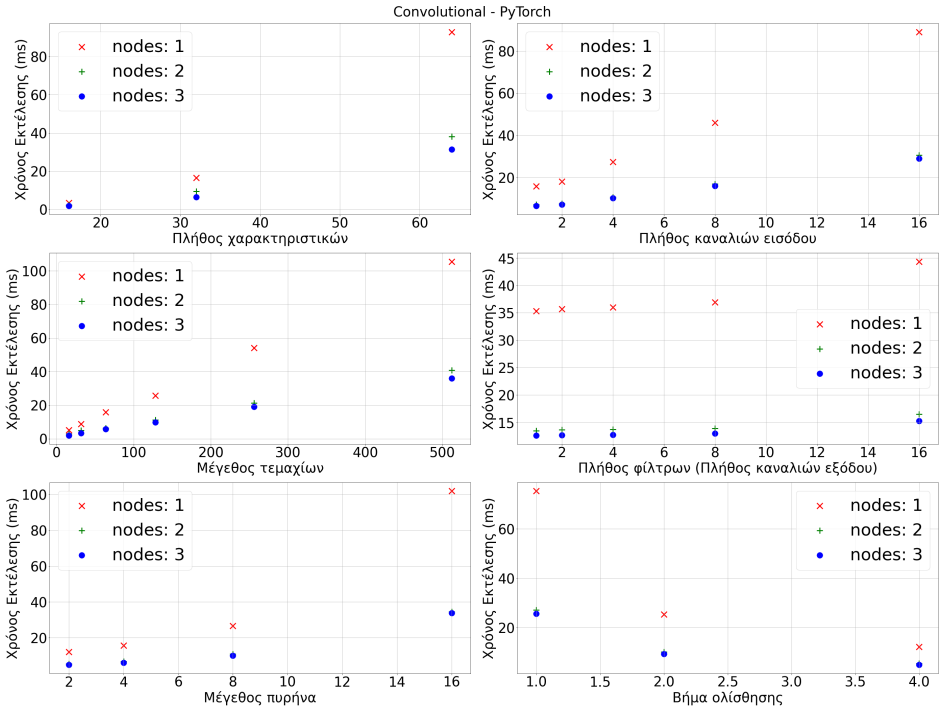


Εικόνα 7.7: ReLU - TensorFlow



Εικόνα 7.8: Tanh - TensorFlow

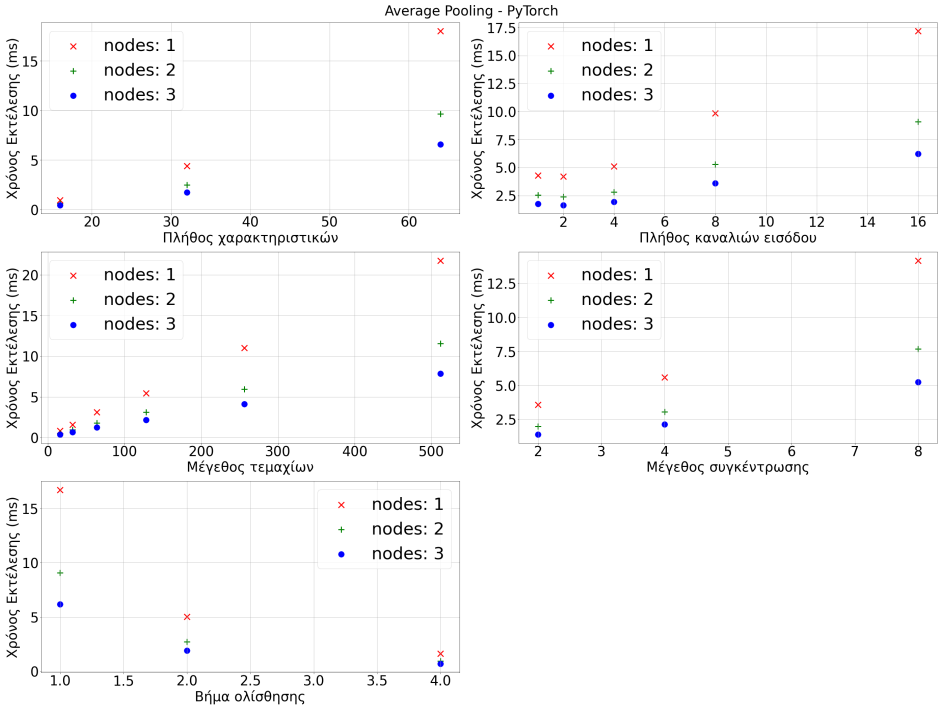
7.1.2 PyTorch



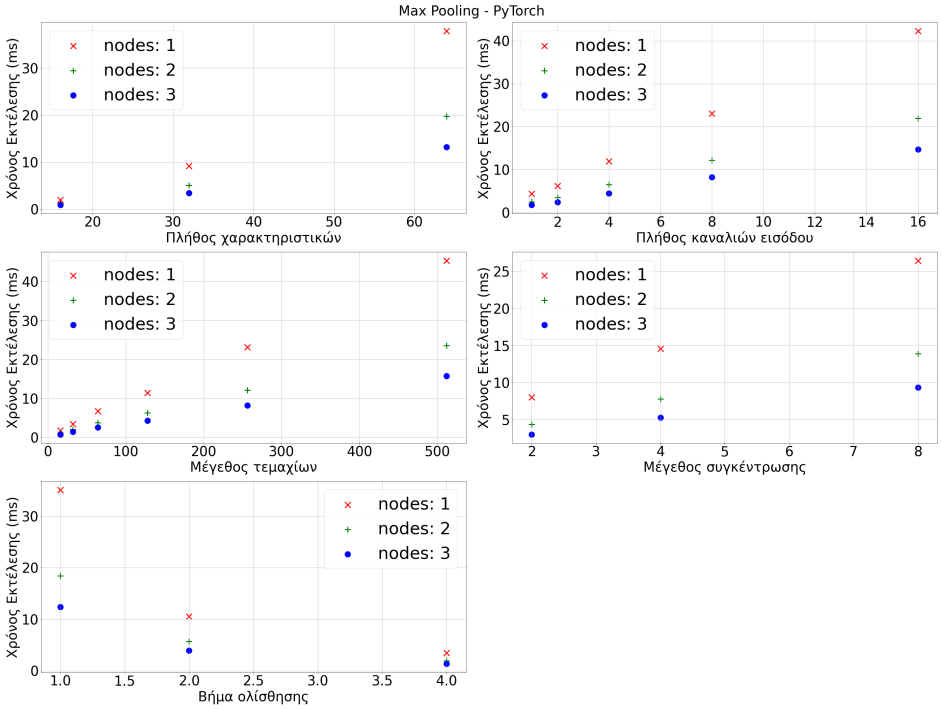
Εικόνα 7.9: Συνελικτικό επίπεδο - PyTorch

Σε γενικές γραμμές, ο χρόνος εκτέλεσης των τελεστών του PyTorch παρουσιάζει συμπεριφορά παρόμοια με του TensorFlow. Πράγματι, ως προς το πλήθος των χαρακτηριστικών εισόδου, στην εικόνα 7.13, ο χρόνος εκτέλεσης του πλήρως συνδεδεμένου επιπέδου τείνει να αυξάνεται σχεδόν γραμμικά, ενώ σε όλα τα υπόλοιπα γραφήματα, ο χρόνος αυξάνεται εκθετικά. Γραμμική σχέση (με θετικό ρυθμό αύξησης) παρατηρούμε και σε σχέση με το μέγεθος των τεμαχίων, σε κάθε ένα από τα γραφήματα.

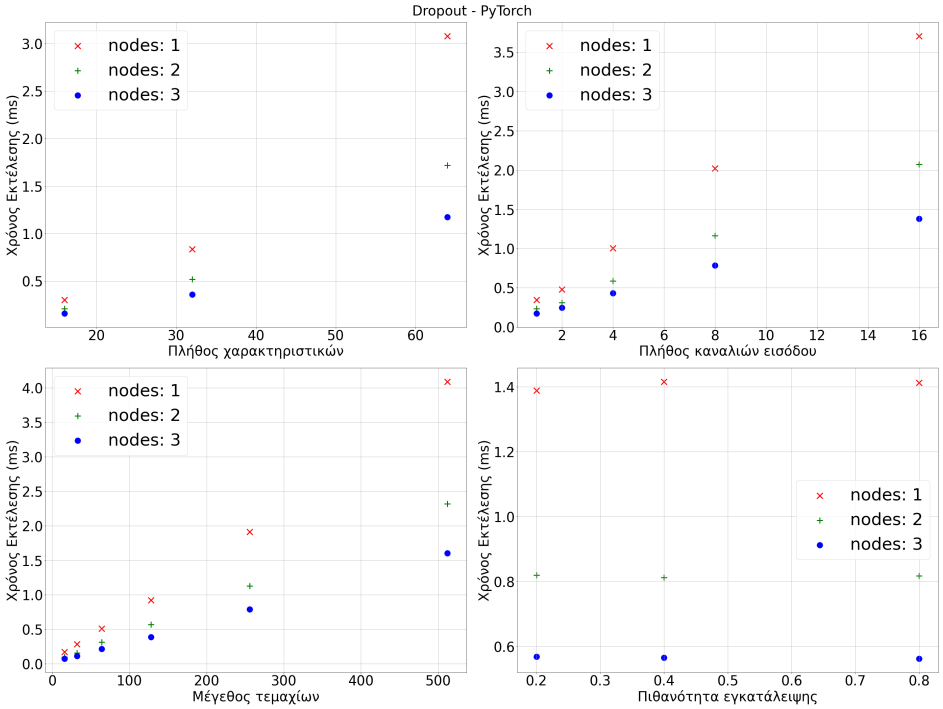
Στις εικόνες 7.9, 7.10 και 7.11, ο χρόνος αυξάνεται εκθετικά σε σχέση με το μέγεθος πυρήνα ή συγκέντρωσης, και αντιστρόφως ανάλογα με το βήμα ολίσθησης, όπως και στο TensorFlow. Ως προς το συνελικτικό επίπεδο, παρατηρούμε ότι ο χρόνος σχετικά με το πλήθος καναλιών εξόδου, αυξάνεται σχεδόν γραμμικά, με αρκετά μικρό, όμως, ρυθμό.



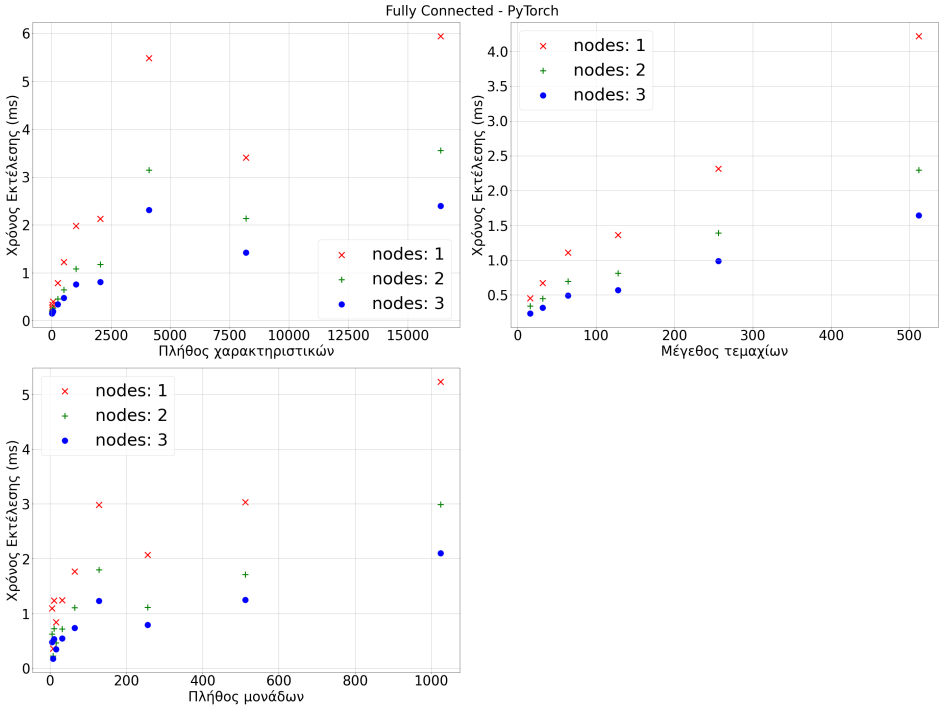
Εικόνα 7.10: Επίπεδο συγκέντρωσης μέσης τιμής - PyTorch



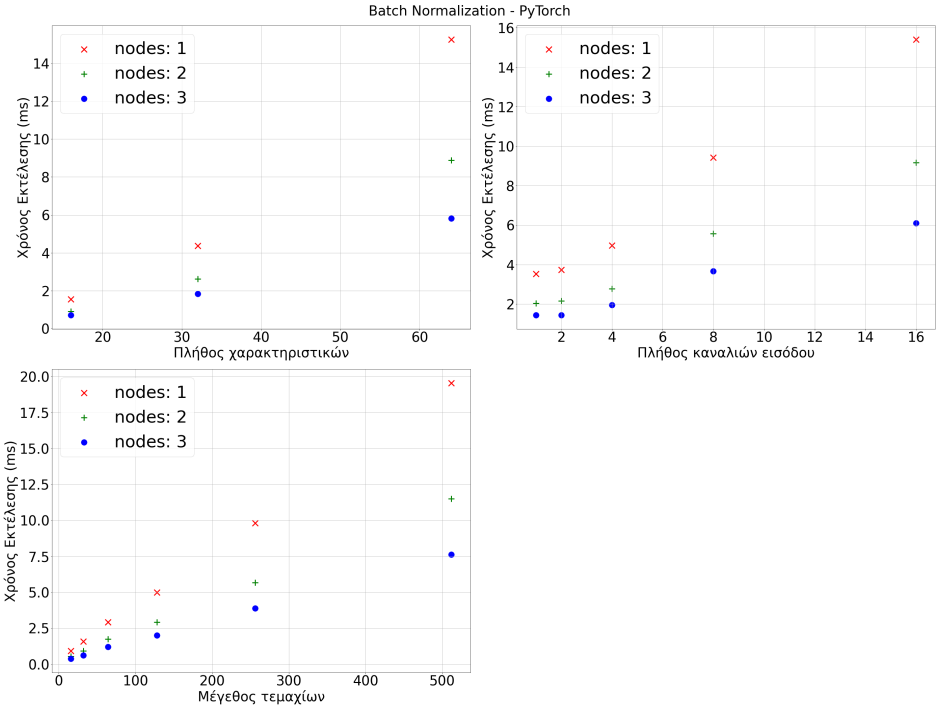
Εικόνα 7.11: Επίπεδο συγκέντρωσης μέγιστης τιμής - PyTorch



Εικόνα 7.12: Επίπεδο εγκατάλειψης - PyTorch

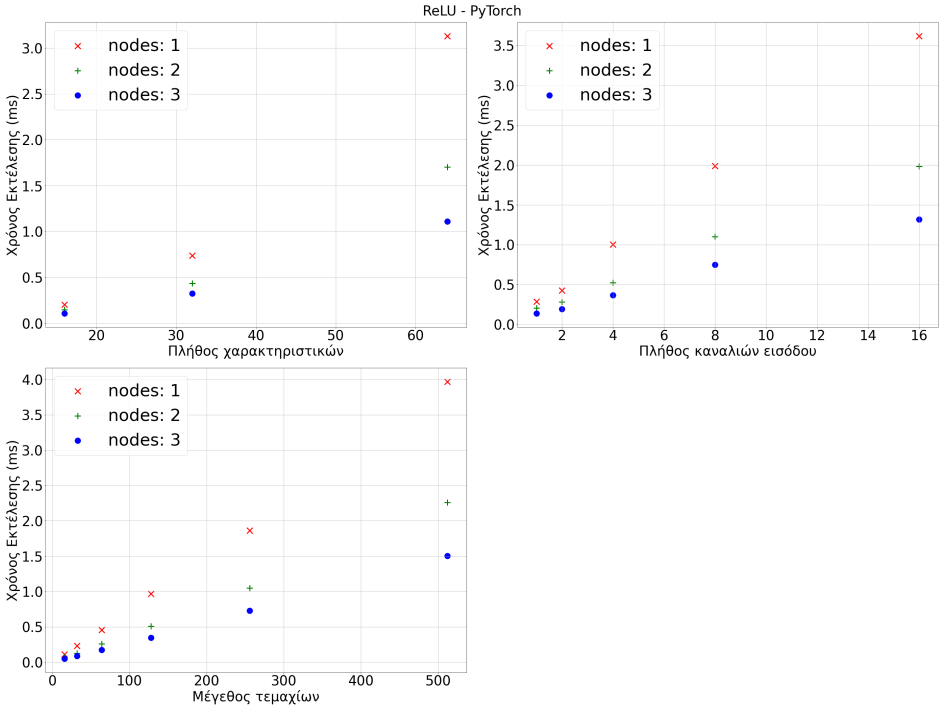


Εικόνα 7.13: Πλήρως συνδεδεμένο επίπεδο - PyTorch

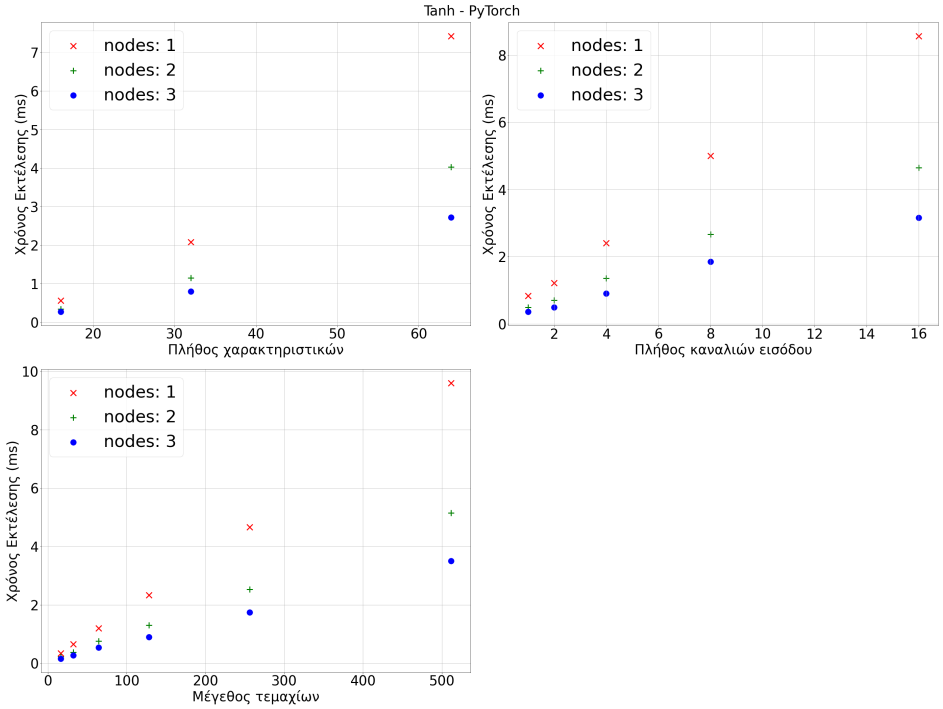


Εικόνα 7.14: Επίπεδο κανονικοποίησης - PyTorch

Τέλος, αναφορικά με τα επίπεδα συγκέντρωσης (7.10 και 7.11) και το επίπεδο κανονικοποίησης (7.14), ο χρόνος, συναρτήσει του πλήθους των καναλιών εισόδου, είναι σαφές ότι αυξάνεται γραμμικά, κάτι που δεν μπορέσαμε με ευκολία να διαπιστώσουμε με τα γραφήματα του TensorFlow.



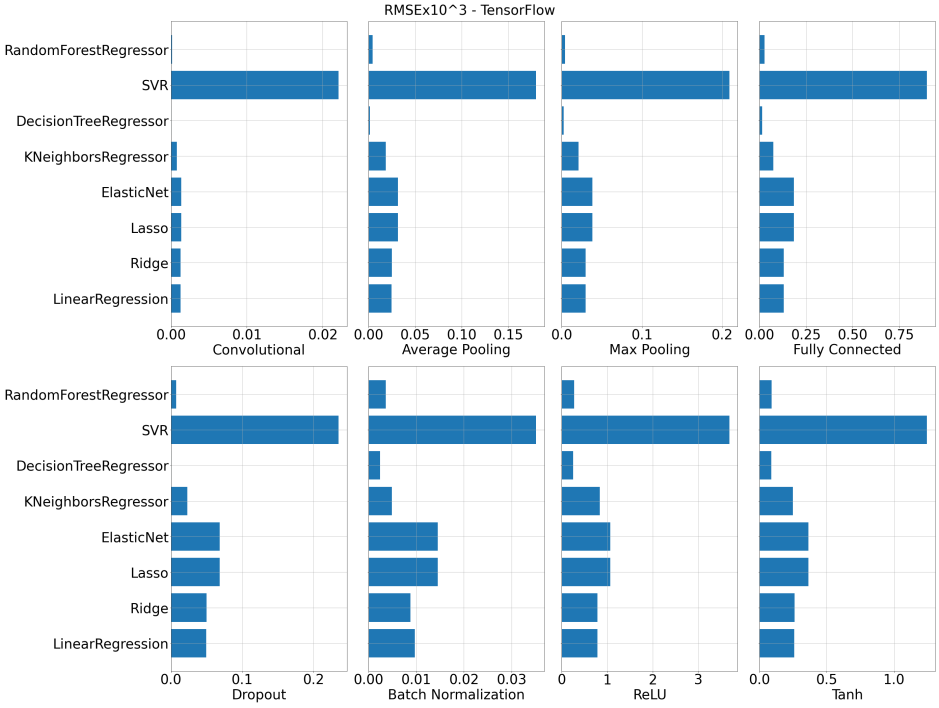
Εικόνα 7.15: ReLU - PyTorch



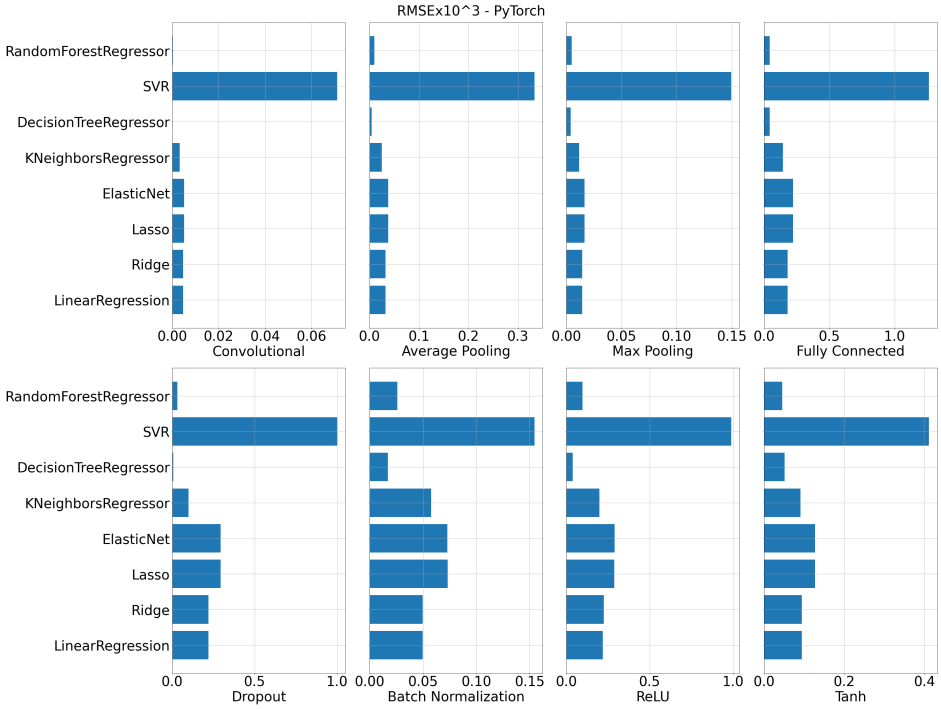
Εικόνα 7.16: Tanh - PyTorch

7.2 Σύγκριση Προβλεπτικών Μοντέλων Χρόνων

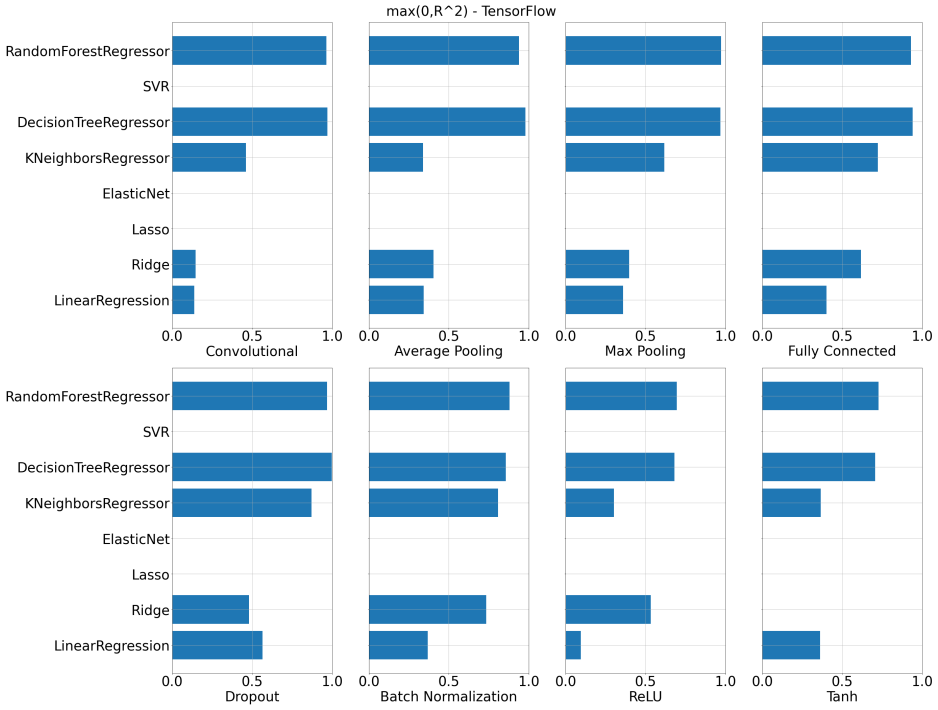
Παρακάτω παραθέτουμε ραβδογράμματα τα οποία θα μας βοηθήσουν να συγκρίνουμε τα σκορ R^2 και $RMSE$, που έχουν προκύψει για το συνδυασμό κάθε επιπέδου με κάθε μοντέλο παλινδρόμησης. Σημειώνουμε ότι ένα καλό σκορ τύπου $RMSE$ σημαίνει όσο το δυνατόν χαμηλότερη τιμή, ενώ το βέλτιστο σκορ R^2 είναι για τιμή ίση με 1, που είναι και το άνω όριο. Το R^2 μπορεί, όμως, να λάβει και αρνητικές τιμές, και σε αυτήν την περίπτωση, αποτελεί ένδειξη ότι δεν έχουμε καλό μοντέλο παλινδρόμησης, με τα συγκεκριμένα δεδομένα. Μάλιστα, για τη διευκόλυνση μας, στα γραφήματα με το R^2 έχουμε παρουσιάσει μόνο τις θετικές τιμές. Ειδάλλως, μερικές αρνητικές τιμές είχαν τόσο μεγάλη απόλυτη τιμή που καθιστούσε αδύνατο να μπορούμε οπτικά να συγκρίνουμε τα μοντέλα με σκορ που να ανήκει στο εύρος τιμών $[0, 1]$.



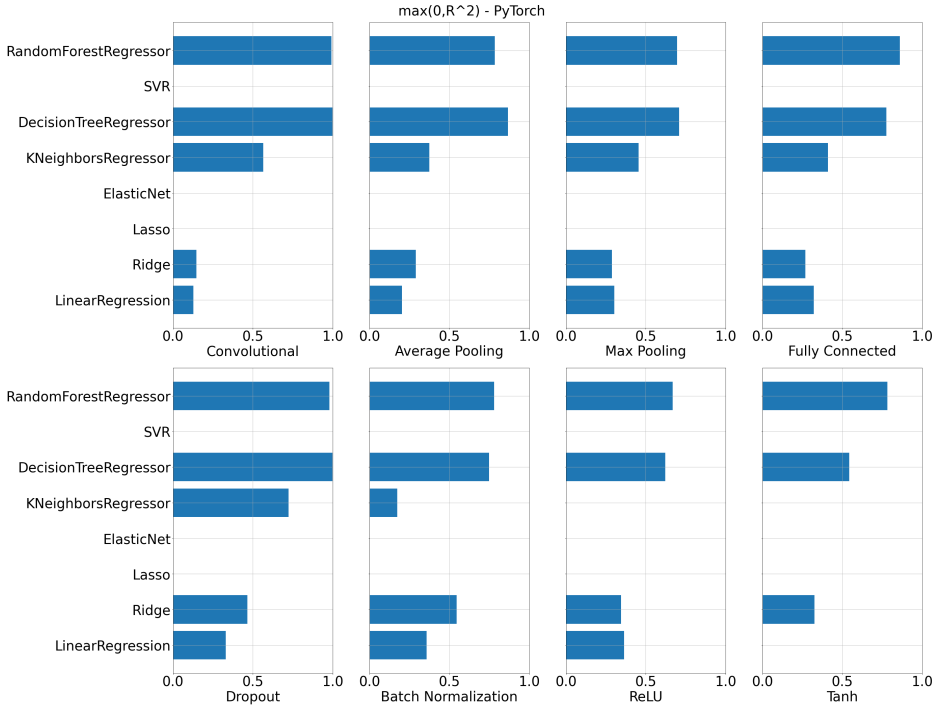
Εικόνα 7.17: $RMSE$ - TensorFlow



Εικόνα 7.18: $RMSE$ - PyTorch



Εικόνα 7.19: R^2 - TensorFlow



Εικόνα 7.20: R^2 - PyTorch

Από τις εικόνες, λοιπόν, 7.17, 7.19 7.18 και 7.20, διαπιστώνουμε ότι την χειρότερη απόδοση την έχει ο SVR, ενώ ακολουθούν τα γραμμικά μοντέλα παλινδρόμησης. Από την άλλη, τα καλύτερα σκορ ως προς και τις δύο μετρικές, τα παρουσιάζουν οι παλινδρομητές Δέντρων Απόφασης και Τυχαίων Δασών, καθώς έχουν πολύ χαμηλό $RMSE$, αλλά και R^2 που σε όλες τις περιπτώσεις ξεπερνάει την τιμή 0.5, ενώ, σε αρκετές, πλησιάζει την μονάδα.

Εφόσον ο παλινδρομητής Τυχαίων Δασών αποτελεί επέκταση του παλινδρομητή Δέντρων Απόφασης, τον επιλέξαμε ως το μοντέλο πρόβλεψης, δεδομένου ότι και οι δύο παρουσιάζουν τα βέλτιστα σκορ.

7.3 Αξιολόγηση

Για να αξιολογήσουμε το μοντέλο πρόβλεψης, υπολογίσαμε τους χρόνους εκτέλεσης των τελεστών των επιπέδων των δικτύων LeNet-1, LeNet-5 και VGG-11, με σύνολο δεδομένων μεγέθους 4608 και 36864, για να μπορέσουμε να συγκρίνουμε την απόδοση για μικρό αλλά και μεγάλο πλήθος στοιχείων. Ωστόσο, εφόσον το δίκτυο VGG-11 λαμβάνει ως είσοδο εικόνες μεγέθους 224×224, ενώ τα δεδομένα του πειράματός μας ήταν εικόνες με μέγεθος το πολύ 64×64, εκτελέσαμε τις μετρήσεις πάνω σε μια τροποποιημένη μορφή του δικτύου αυτού, η οποία δέχεται ως είσοδο εικόνες μεγέθους 64×64.

Οι πίνακες στο τρέχων υποκεφάλαιο περιλαμβάνουν τις μετρήσεις και τις προβλέψεις για διάφορους συνδυασμούς του πλήθος μηχανημάτων και μεγέθους τεμαχίων. Στον τίτλο κάθε πίνακα έχουμε σημειώσει το είδος του δικτύου, και σε παρένθεση το μέγεθος του συνόλου δεδομένων. Συγκεκριμένα, οι πίνακες 7.2, 7.3, 7.4, 7.5 και 7.6 αφορούν το σύστημα TensorFlow, ενώ οι πίνακες 7.7, 7.8, 7.9, 7.10 και 7.11 αφορούν το PyTorch. Τέλος, στον πίνακα 7.1 βλέπουμε τη μέση τιμή και την διάμεσο του σφάλματος, για κάθε σύστημα.

Πίνακας 7.1: Μέση τιμή και τυπική απόκλιση σφάλματος

Σύστημα	Μέση τιμή %	Διάμεσος %
TensorFlow	21.52	19.25
PyTorch	22.44	19.80

Σε γενικές γραμμές, το σφάλμα μεταξύ του χρόνου εκτέλεσης και της πρόβλεψης τείνει να είναι μικρότερο όσο έχουμε μικρότερο μέγεθος τεμαχίων. Αυτό μπορεί να οφείλεται στο ότι για μεγαλύτερο μέγεθος batch, έχουμε συνολικά λιγότερα βήμα ανά εποχή. Εφόσον κατά την πειραματική αποτίμηση δεν αλλάζαμε το μέγεθος του συνόλου δεδομένων, τα βήματα ανά εποχή ήταν λιγότερα, με κάθε αύξηση του μεγέθους τεμαχίων, με αποτέλεσμα να μην έχουμε αρκετές μετρήσεις ώστε ο μέσος χρόνος να έχει προλάβει να συγκλίνει, ενδεχομένως, σε κάποια τιμή.

Βέβαια, παρατηρούμε ότι στο δίκτυο VGG-11 (πίνακες 7.6 και 7.11) το σφάλμα, ξεκινώντας από υψηλότερη τιμή συγκριτικά με τα προηγούμενα δίκτυα, έχει την τάση να μειώνεται όσο αυξάνει το μέγεθος batch. Σε αυτό μπορεί να οφείλεται το ότι το VGG έχει αρκετά περισσότερα επίπεδα, και έτσι, με μικρά τεμάχια (άρα περισσότερα βήματα) εκτελούνται αρκετά περισσότερες πράξεις. Έτσι, ο προβλεπτής μπορεί να εισάγει σε κάθε ένα επίπεδο ένα ποσοστό σφάλματος, το οποίο ενισχύεται με την αύξηση των βημάτων εκτέλεσης. Παράλληλα, ένα ακόμα αίτιο έγκεινται στην ύπαρξη αρκετών επιπέδων εγκατάλειψης, τα οποία πιθανώς επηρεάζουν τον τρόπο ή το φόρτο των συνολικών πράξεων, κάτι που το μοντέλο μας δεν λαμβάνει υπόψιν του, καθώς έχουμε μελετήσει κάθε ένα επίπεδο ξεχωριστά από τα υπόλοιπα. Με άλλα λόγια, θα ήταν αναγκαία η συλλογή μετρήσεων πάνω σε έναν συνδυασμό από συνελικτικά επίπεδα και επίπεδα εγκατάλειψης, προκειμένου να παρακολουθήσουμε την συμπεριφορά του χρόνου εκτέλεσης σε αυτήν την περίπτωση.

Παρατηρούμε ότι η πρόβλεψη για το TensorFlow είναι πιο ικανοποιητική για τα δίκτυα LeNet-1 και LeNet-5, καθώς έχουν λιγότερα επίπεδα, ενώ είναι ακόμα πιο ικανοποιητική όσο μικρότερο είναι το σύνολο εκπαίδευσης, δηλαδή όσο λιγότερες οι συνολικές πράξεις. Παρόμοια συμπεριφορά παρατηρούμε και στις προβλέψεις για το PyTorch, οι οποίες, ωστόσο, για μικρά μεγέθη τεμαχίων, έχουν χαμηλότερο σφάλμα σε σχέση με του TensorFlow, ενώ σε μία εκτέλεση, μάλιστα, βλέπουμε ποσοστό σφάλματος ίσο με 0,4% (πίνακας 7.8). Είναι αρκετά πιθανό τα υψηλότερα ποσοστά σφάλματος του TensorFlow να ευθύνονται στον ίδιο τον μηχανισμό συλλογής δεδομένων, καθώς η χρήση του TensorBoard, το οποίο, ως ανεξάρτητο πρόγραμμα,

Πίνακας 7.2: TensorFlow-LeNet1 (4608)

Batch size	Nodes	Time (s)	Predicted(s)	Difference(s)	Error%
16.0	1.0	5.552860	6.163743	0.610883	11.0
16.0	2.0	2.776430	3.042127	0.265696	9.6
16.0	3.0	1.850953	2.031219	0.180266	9.7
32.0	1.0	5.129190	5.283608	0.154419	3.0
32.0	2.0	2.564595	2.632613	0.068018	2.7
32.0	3.0	1.709730	1.761924	0.052194	3.1
64.0	1.0	4.768607	4.854553	0.085946	1.8
64.0	2.0	2.384304	2.439148	0.054844	2.3
64.0	3.0	1.589536	1.629168	0.039632	2.5
128.0	1.0	3.726668	4.394104	0.667436	17.9
128.0	2.0	1.863334	2.199796	0.336462	18.1
128.0	3.0	1.242223	1.462638	0.220416	17.7

Πίνακας 7.3: TensorFlow-LeNet1 (36864)

Batch size	Nodes	Time (s)	Predicted(s)	Difference(s)	Error%
16.0	1.0	46.546199	49.309944	2.763745	5.9
16.0	2.0	23.273100	24.337013	1.063913	4.6
16.0	3.0	15.515400	16.249752	0.734352	4.7
32.0	1.0	43.244172	42.268868	0.975304	2.3
32.0	2.0	21.622086	21.060901	0.561185	2.6
32.0	3.0	14.414724	14.095389	0.319335	2.2
64.0	1.0	36.546891	38.836426	2.289534	6.3
64.0	2.0	18.273446	19.513185	1.239739	6.8
64.0	3.0	12.182297	13.033342	0.851045	7.0
128.0	1.0	28.224814	35.152835	6.928022	24.5
128.0	2.0	14.112407	17.598369	3.485962	24.7
128.0	3.0	9.408271	11.701106	2.292835	24.4

Πίνακας 7.4: TensorFlow-LeNet5 (4608)

Batch size	Nodes	Time (s)	Predicted(s)	Difference(s)	Error%
16.0	1.0	7.766443	9.291140	1.524698	19.6
16.0	2.0	3.883221	4.630809	0.747587	19.3
16.0	3.0	2.588814	3.110816	0.522002	20.2
32.0	1.0	6.550888	7.496448	0.945560	14.4
32.0	2.0	3.275444	3.641876	0.366432	11.2
32.0	3.0	2.183629	2.442835	0.259206	11.9
64.0	1.0	6.107675	7.257536	1.149861	18.8
64.0	2.0	3.053838	3.697000	0.643163	21.1
64.0	3.0	2.035892	2.599901	0.564010	27.7
128.0	1.0	5.306523	7.795960	2.489437	46.9
128.0	2.0	2.653261	3.897692	1.244431	46.9
128.0	3.0	1.768841	2.556995	0.788154	44.6

Πίνακας 7.5: TensorFlow-LeNet5 (36864)

Batch size	Nodes	Time (s)	Predicted(s)	Difference(s)	Error%
16.0	1.0	62.361444	74.329123	11.967679	19.2
16.0	2.0	31.180722	37.046469	5.865747	18.8
16.0	3.0	20.787148	24.886529	4.099381	19.7
32.0	1.0	50.713444	59.971585	9.258141	18.3
32.0	2.0	25.356722	29.135009	3.778287	14.9
32.0	3.0	16.904481	19.542684	2.638202	15.6
64.0	1.0	46.328977	58.060286	11.731310	25.3
64.0	2.0	23.164488	29.576004	6.411516	27.7
64.0	3.0	15.442992	20.799210	5.356218	34.7
128.0	1.0	40.410446	62.367677	21.957231	54.3
128.0	2.0	20.205223	31.181539	10.976316	54.3
128.0	3.0	13.470149	20.455963	6.985814	51.9

Πίνακας 7.6: TensorFlow-VGG11 (4608)

Batch size	Nodes	Time (s)	Predicted(s)	Difference(s)	Error%
16.0	1.0	332.963549	190.555578	142.407972	42.8
16.0	2.0	166.481775	94.839170	71.642605	43.0
16.0	3.0	110.987850	63.407179	47.580671	42.9
32.0	1.0	304.234759	196.951800	107.282959	35.3
32.0	2.0	152.117379	98.618777	53.498602	35.2
32.0	3.0	101.411586	65.736335	35.675251	35.2
64.0	1.0	257.896816	177.631899	80.264917	31.1
64.0	2.0	128.948408	88.616883	40.331525	31.3
64.0	3.0	85.965605	58.823642	27.141963	31.6
128.0	1.0	238.213484	166.891352	71.322132	29.9
128.0	2.0	119.106742	83.215154	35.891587	30.1
128.0	3.0	79.404495	55.448271	23.956224	30.2

Πίνακας 7.7: PyTorch-LeNet1 (4608)

Batch size	Nodes	Time (s)	Predicted(s)	Difference(s)	Error%
16.0	1.0	4.841353	4.714269	0.127085	2.6
16.0	2.0	2.953216	2.763334	0.189882	6.4
16.0	3.0	2.238375	2.291058	0.052683	2.4
32.0	1.0	3.231895	3.472674	0.240779	7.5
32.0	2.0	1.947263	1.994999	0.047737	2.5
32.0	3.0	1.380933	1.418788	0.037855	2.7
64.0	1.0	2.199215	2.745934	0.546720	24.9
64.0	2.0	1.416036	1.705176	0.289141	20.4
64.0	3.0	1.278067	1.126374	0.151693	11.9
128.0	1.0	1.707888	2.337805	0.629916	36.9
128.0	2.0	1.084042	1.450441	0.366399	33.8
128.0	3.0	0.803701	1.235830	0.432130	53.8

τρέχει παράλληλα με την εκτέλεση του νευρωνικού δικτύου, και έτσι εισάγει, ενδεχομένως, κάποιο σφάλμα στις μετρήσεις.

Πρέπει να σημειώσουμε ότι, ως προς και τα δύο συστήματα, οι μετρήσεις έχουν συλλεχθεί για κάθε ένα βήμα εκπαίδευσης. Αυτό αποτελεί πρόβλημα διότι στα αρχικά βήματα πραγματοποιείται αρχικοποίηση των παραμέτρων προς μάθηση στα επίπεδα, με αποτέλεσμα να προστίθεται παραπανίσιος χρόνος εκτέλεσης, ο οποίος έχουμε παρατηρήσει ότι όχι μόνο δεν είναι αμελητέος, αλλά είναι και σημαντικός συγκριτικά με τον χρόνο των υπόλοιπων βημάτων. Επηρεάζονται, συνεπώς, κατ' αυτόν τον τρόπο οι προβλέψεις του μοντέλου μας, αυξάνοντας το σφάλμα. Το πρόβλημα αυτό καθίσταται ακόμα πιο δύσκολο να λυθεί, δεδομένου ότι τα δίκτυα που μελετήσαμε, διαθέτοντας πληθώρα επιπέδων, εισάγουν χρόνο αρχικοποίησης που είναι πιθανό να εξαρτάται από την τοπολογία του δικτύου. Επομένως, μία λύση, πέρα από την μελέτη του χρόνου αρχικοποίησης σε διάφορες τοπολογίες, αποτελεί η απομόνωση των πρώτων βημάτων από τη χρονομέτρηση, τόσο κατά τη διαδικασία συλλογής, όσο και κατά την αξιολόγηση.

Πίνακας 7.8: PyTorch-LeNet1 (36864)

Batch size	Nodes	Time (s)	Predicted(s)	Difference(s)	Error%
16.0	1.0	38.327092	37.714148	0.612944	1.6
16.0	2.0	22.420943	22.106670	0.314273	1.4
16.0	3.0	15.920173	18.328463	2.408291	15.1
32.0	1.0	27.669423	27.781394	0.111971	0.4
32.0	2.0	14.959365	15.959995	1.000630	6.7
32.0	3.0	10.714472	11.350304	0.635832	5.9
64.0	1.0	17.926321	21.967474	4.041153	22.5
64.0	2.0	9.358408	13.641409	4.283001	45.8
64.0	3.0	6.939062	9.010995	2.071933	29.9
128.0	1.0	14.587319	18.702437	4.115118	28.2
128.0	2.0	8.830282	11.603528	2.773246	31.4
128.0	3.0	5.644226	9.886643	4.242418	75.2

Πίνακας 7.9: PyTorch-LeNet5 (4608)

Batch size	Nodes	Time (s)	Predicted(s)	Difference(s)	Error%
16.0	1.0	8.264500	8.720397	0.455897	5.5
16.0	2.0	5.122397	4.955213	0.167185	3.3
16.0	3.0	3.517558	3.387470	0.130087	3.7
32.0	1.0	5.259099	5.915696	0.656597	12.5
32.0	2.0	3.015016	3.629569	0.614553	20.4
32.0	3.0	2.275057	2.420035	0.144978	6.4
64.0	1.0	3.676717	4.991270	1.314553	35.8
64.0	2.0	2.246776	3.077824	0.831047	37.0
64.0	3.0	1.656369	2.015847	0.359478	21.7
128.0	1.0	2.753077	4.326122	1.573045	57.1
128.0	2.0	1.770129	2.477885	0.707756	40.0
128.0	3.0	1.297618	1.875709	0.578091	44.6

Πίνακας 7.10: PyTorch-LeNet5 (36864)

Batch size	Nodes	Time (s)	Predicted(s)	Difference(s)	Error%
16.0	1.0	67.580614	69.763174	2.182560	3.2
16.0	2.0	38.234519	39.641702	1.407183	3.7
16.0	3.0	24.124433	27.099762	2.975329	12.3
32.0	1.0	44.999646	47.325569	2.325922	5.2
32.0	2.0	25.137565	29.036552	3.898987	15.5
32.0	3.0	15.841280	19.360277	3.518997	22.2
64.0	1.0	29.199851	39.930156	10.730305	36.7
64.0	2.0	15.941019	24.622589	8.681570	54.5
64.0	3.0	10.822569	16.126776	5.304207	49.0
128.0	1.0	23.673212	34.608979	10.935768	46.2
128.0	2.0	13.373167	19.823080	6.449913	48.2
128.0	3.0	8.967288	15.005672	6.038385	67.3

Πίνακας 7.11: PyTorch-VGG11 (4608)

Batch size	Nodes	Time (s)	Predicted(s)	Difference(s)	Error%
16.0	1.0	646.877741	519.393130	127.484611	19.7
16.0	2.0	371.149326	292.619074	78.530252	21.2
16.0	3.0	251.230275	194.483900	56.746375	22.6
32.0	1.0	560.893453	468.878296	92.015158	16.4
32.0	2.0	312.533140	251.414662	61.118478	19.6
32.0	3.0	216.792405	175.382484	41.409920	19.1
64.0	1.0	513.599871	411.235609	102.364262	19.9
64.0	2.0	276.949144	230.451464	46.497680	16.8
64.0	3.0	190.731009	151.815008	38.916001	20.4
128.0	1.0	487.839548	376.521669	111.317879	22.8
128.0	2.0	250.151275	215.909932	34.241343	13.7
128.0	3.0	171.857160	150.458068	21.399092	12.5

Κεφάλαιο 8

Σχετική Εργασία

Ως προς την πρόβλεψη χρόνου εκτέλεσης, έχει πραγματοποιηθεί παρόμοια εργασία [63], στην οποία ο Daniel Justus κ.ά. υλοποίησαν ένα μοντέλο βαθιάς μάθησης που μπορεί να προβλέψει με ακρίβεια τον απαιτούμενο χρόνο εκπαίδευσης των πιο συχνά χρησιμοποιούμενων δομικών στοιχείων των νευρωνικών δικτύων. Το μοντέλο αυτό μπορεί να βοηθήσει στην επιλογή κατάλληλου υλικού για την εκπαίδευση ενός δικτύου, αλλά και στην λήψη αποφάσεων για την σχεδίασή του.

Όπως και στην δική μας περίπτωση, ο προβλεπτής υπολογίζει τον απαιτούμενο χρόνο εκπαίδευσης για ένα βήμα, κλιμακώνοντας, στην συνέχεια, κατάλληλα την τιμή του για να υπολογίσει τον συνολικό χρόνο. Ωστόσο, δεδομένου ότι τον σημαντικότερο χρόνο καταναλώνουν η συνέλιξη και τα πλήρως συνδεδεμένα επίπεδα, η εργασία τους εστιάζει στην μελέτη μόνο αυτών των επιπέδων. Για να τα εξετάσουν, λοιπόν, πραγματοποίησαν συλλογή μετρήσεων, πάνω στο σύστημα TensorFlow, για ποικίλους συνδυασμούς παραμέτρων εισόδου.

Στα εξεταζόμενα επίπεδα, το μέγεθος τεμαχίου αποτελεί κοινή παράμετρο. Στα συνελικτικά επίπεδα, οι παράμετροι είναι οι εξής:

- Μέγεθος εισόδου
- Μέγεθος πυρήνα
- Πλήθος καναλιών εισόδου
- Πλήθος καναλιών εξόδου
- Βήμα ολίσθησης
- Μέγεθος padding
- Πρόσθεση ή μη του bias

ενώ στα πλήρως συνδεδεμένα επίπεδα, οι εξεταζόμενες παράμετροι είναι το πλήθος νευρώνων εισόδου και το πλήθος νευρώνων εξόδου. Εφόσον, όμως, το πλήθος των

συνδυασμών που προκύπτουν είναι πολύ μεγάλο, η συλλογή δεδομένων πραγματοποιήθηκε για ένα τυχαίο υποσύνολο συνδυασμών των τιμών των παραμέτρων. Παράλληλα, ένα ακόμα χαρακτηριστικό ως προς το οποίο έγινε η μελέτη είναι το υλικό, καθώς χρησιμοποιήθηκε μεταβλητός αριθμός από κάρτες GPU, διαφόρων ειδών τεχνολογίας.

Τα δεδομένα που προέκυψαν, στην συνέχεια, τροφοδοτήθηκαν σε νευρωνικό δίκτυο μίας εξόδου, το οποίο, τελικά, χρησιμοποιήθηκε ως προβλεπτής διότι, σε σύγκριση με άλλα μοντέλα που δοκιμάστηκαν, όπως ο γραμμικός παλινδρομητής, έδωσε ικανοποιητικό αποτέλεσμα. Παρόμοια με το μοντέλο μας, η πρόβλεψη δίνεται από την άθροιση των χρόνων εκτέλεσης των επιμέρους επιπέδων του δοθέντος δικτύου. Σημειώνουμε ότι, ενώ η δική μας μελέτη περιλαμβάνει την συγκέντρωση των χρόνων κατά την φάση τροφοδότησης και οπισθοδρόμησης αθροιστικά, η συλλογή των δεδομένων στη σχετική εργασία πραγματοποιήθηκε, επιπλέον, και κατά την φάση της τροφοδότησης ξεχωριστά. Ακόμη μία διαφορά έγκειται στο ότι χρησιμοποιήθηκαν διάφορες τεχνικές βελτιστοποίησης πέραν της στοχαστικής καθόδου κλίσεων (SGD).

Από τα δεδομένα που συλλέχθηκαν, προέκυψε το συμπέρασμα ότι το μέγεθος τεμαχίων επηρεάζει σημαντικά τον χρόνο εκτέλεσης στα συνελικτικά επίπεδα, κάτι που δεν ισχύει στα πλήρως συνδεδεμένα επίπεδα. Η αξιολόγηση του μοντέλου πραγματοποιήθηκε στο δίκτυο VGG-16. Τελικά, διαπιστώθηκε ότι, ενώ οι τιμές μεταξύ πραγματικού και του χρόνου πρόβλεψης παρουσιάζουν μερική παρέκκλιση, ιδιαίτερα για μεγαλύτερα μεγέθη τεμαχίων, το μοντέλο επιτρέπει τόσο την κατανόηση της πολυπλοκότητας υπολογισμού ενός δικτύου όσο και την εκτίμηση του χρόνου εκπαίδευσής του, δίνοντας, παράλληλα, πληροφορίες σχετικά με την επίδραση του υλικού που θα χρησιμοποιηθεί.

Κεφάλαιο 9

Συμπεράσματα - Επεκτάσεις

Ο μέσος όρος του σφάλματος που προέκυψε ισούται με περίπου 22%, και για τα δύο συστήματα, ενώ η διάμεσος ισούται με περίπου 19.5%. Με βάση αυτές τις μετρικές και τις προηγούμενες παρατηρήσεις, βλέπουμε ότι έχουμε ένα σχετικά καλό μοντέλο υπό προϋποθέσεις, όπως είναι η χαμηλή τιμή του μεγέθους τεμαχίων, το οποίο, όμως, απαιτεί βελτίωση για να έχουμε πιο αξιόπιστα αποτελέσματα. Αυτό μπορούμε να το επιτύχουμε με ποικίλους τρόπους, μερικοί από τους οποίους είναι οι εξής:

- Συλλογή μετρήσεων για περισσότερες τιμές των παραμέτρων εισόδου, αλλά και για περισσότερες τιμές του πλήθους χαρακτηριστικών, για να έχουμε μεγαλύτερη ακρίβεια.
- Συλλογή μετρήσεων για μεγαλύτερο πλήθος βημάτων, ιδιαίτερα στην περίπτωση που έχουμε υψηλή τιμή του μεγέθους τεμαχίων.
- Μελέτη της συμπεριφοράς του χρόνου που καταλαμβάνει η αρχικοποίηση του δικτύου. Για παράδειγμα, η συλλογή μετρήσεων μπορεί να υλοποιηθεί αφενός για τον συνολικό χρόνο εκτέλεσης και αφετέρου δίχως τα πρώτα βήματα.
- Μελέτη του χρόνου αρχικοποίησης σε διάφορες τοπολογίες δικτύων. Για παράδειγμα, μπορεί να μετρηθεί σε ένα δίκτυο με 1 συνελικτικό επίπεδο, και ύστερα για περισσότερα, με στόχο την εύρεση κάποιου μοτίβου.
- Μελέτη των χρόνων εκπαίδευσης για συνδυασμό επιπέδων, όπως, για παράδειγμα, για ένα συνελικτικό με ένα επίπεδο συγκέντρωσης, ή για ένα επίπεδο εγκατάλειψης που ακολουθείται από ένα συνελικτικό, προκειμένου να γίνει κατανοητή η επίδρασή τους στον συνολικό χρόνο εκτέλεσης.

Πέρα από την βελτίωση του μοντέλου πρόβλεψης, είναι αναγκαία και η επέκτασή του, ώστε να μπορεί να διαχειριστεί μοντέλα με μεγαλύτερο πλήθος χαρακτηριστικών, αλλά και να μπορεί να προβλέπει περισσότερα συστήματα, πέραν του TensorFlow και PyTorch. Παράλληλα, θα ήταν αρκετά βοηθητικές οι κάρτες

επεξεργασίας γραφικών (GPU), διότι η χρήση τους μπορεί να επιταχύνει σημαντικά την εκτέλεση ενός νευρωνικού δικτύου. Σε αυτήν την περίπτωση, όμως, πρέπει να ληφθεί υπόψιν η πολυπλοκότητα που εισάγεται στην συγγραφή του κώδικα.

Υπό την προϋπόθεση ότι το μοντέλο πρόβλεψης έχει βελτιωθεί και επεκταθεί, μπορεί να χρησιμοποιηθεί στην σύνταξη ενός προγράμματος, τύπου χρονοδρομολόγησης, το οποίο λαμβάνει σαν είσοδο ένα νευρωνικό δίκτυο και αποφασίζει ποιο σύστημα θα συμβάλλει στην ταχύτερη εκτέλεσή του.

Αναφορές

[1] Max Jaderberg et al. *Synthetic Data and Artificial Neural Networks for Natural Scene Text Recognition*. https://arxiv.org/pdf/1406.2227.pdf.

[2] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems*. 2016. arXiv: 1603.04467 [cs.DC].

[3] Adam Paszke et al. *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. http://papers.nips.cc/paper/9015-pytorch-an-imperative-stylehigh-.

[4] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[5] Alan J. Lee George A. F. Seber. *Linear Regression Analysis*. Wiley, 2003.

[6] scikit-learn developers. *Linear Models*. https://scikit-learn.org/stable/modules/linear_model.html.

[7] NCSS Statistical Software. *Ridge Regression*. http://ncss.wpengine.netdna-cdn.com/wp-content/themes/ncss/pdf/Procedures/NCSS/Ridge_Regression.pdf.

[8] scikit-learn developers. *Plot Ridge coefficients as a function of the regularization*. https://scikit-learn.org/stable/auto_examples/linear_model/plot_ridge_path.html.

[9] Robert Tibshirani. *Regression Shrinkage and Selection via the Lasso*. https://statweb.stanford.edu/~tibs/lasso/lasso.pdf.

[10] Hui Zou and Trevor Hastie. *Regularization and variable selection via theelastic net*. http://www.recognition.mccme.ru/pub/papers/L1/elasticnet.pdf.

[11] S.J. Wright. *Coordinate descent algorithms*. https://doi.org/10.1007/s10107-015-0892-3. 2015.

[12] *Lasso and Elastic Net*. https://scikit-learn.org/stable/auto_examples/linear_model/plot_lasso_coordinate_descent_path.html.

[13] N.S. Altman. *An Introduction to Kernel and Nearest Neighbor Nonparametric Regression*. https://ecommons.cornell.edu/bitstream/handle/1813/31637/BU-1065-MA.pdf;jsessionid=E656C69705F6B1535998C46FC15DD78C?sequence=1. 1991.

[14] *Nearest Neighbors*. https://scikit-learn.org/stable/modules/neighbors.html.

[15] *K Nearest Neighbors - Regression*. https://www.saedsayad.com/k_nearest_neighbors_reg.htm.

[16] Alex J. Smola and Bernhard Schölkopf. *A tutorial on support vector regression*. https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.114.4288. 2004.

[17] JonJacobsen. *AsFlatAsPossible*. https://pdfs.semanticscholar.org/899b/8dae38d81c1f0a93256a378968f108c3c91d.pdf.

[18] *Support Vector Machine - Regression (SVR)*. https://www.saedsayad.com/support_vector_machine_reg.htm.

[19] David M. Magerman. *Statistical Decision-Tree Models for Parsing*. https://arxiv.org/pdf/cmp-lg/9504030.pdf.

[20] *Decision Tree - Regression*. https://www.saedsayad.com/decision_tree_reg.htm.

[21] *Decision tree learning*. https://en.wikipedia.org/wiki/Decision_tree_learning.

[22] J. R. Quinlan. *Induction of decision trees*. https://link.springer.com/article/10.1007/BF00116251.

[23] Leo Breiman and Adele Cutler. *Random Forests*. https://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm.

[24] *Ensemble methods*. https://scikit-learn.org/stable/modules/ensemble.html.

[25] RENAUD et al. *A robust coefficient of determination for regression*. https://archive-ouverte.unige.ch/unige:5672.

[26] N. J. D. Nagelkerke. *A Note on a General Definition of the Coefficient of Determination*. https://www.cesarzamudio.com/uploads/1/7/9/1/17916581/nagelkerke_n.j.d._1991_-_a_note_on_a_general_definition_of_the_coefficient_of_determination.pdf.

[27] *Mean Squared Error (MSE)*. https://www.probabilitycourse.com/chapter9/9_1_5_mean_squared_error_MSE.php.

[28] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[29] Michael A. Nielsen. *Neural Networks and Deep Learning*. http://neuralnetworksanddeeplearning.com/. Determination Press, 2015.

[30] XINYOU YIN et al. «A Flexible Sigmoid Function of Determinate Growth». In: *Annals of Botany* 91.3 (Feb. 2003), pp. 361–371. ISSN: 0305-7364. DOI: 10.1093/aob/mcg029. eprint: https://academic.oup.com/aob/article-pdf/91/3/361/627093/mcg029.pdf. URL: https://doi.org/10.1093/aob/mcg029.

[31] Fei Wang Suyog Gupta Wei Zhang. *Model Accuracy and Runtime Tradeoff inDistributed Deep Learning: A Systematic Study*. https://par.nsf.gov/servlets/purl/10026400. 2016.

[32] Geoffrey E. Hinton Alex Krizhevsky Ilya Sutskever. *ImageNet Classification with Deep Convolutional Neural Networks*. https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf. 2012.

[33] *The Influence of the Activation Function in a Convolution Neural Network Model of Facial Expression Recognition*. https://www.mdpi.com/2076-3417/10/5/1897/htm.

[34] *Conceptual Understanding of Convolutional Neural Network- A Deep Learning Approach*. https://www.sciencedirect.com/science/article/pii/S1877050918308019.

[35] *An Intuitive Explanation of Convolutional Neural Networks*. https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/.

[36] Zenghui Wang Waseem Rawat. *Deep Convolutional Neural Networks for Image Classification: A Comprehensive Review*. https://www.mitpressjournals.org/doi/pdfplus/10.1162/neco_a_00990.

[37] *Fully Connected Layers in Convolutional Neural Networks: The Complete Guide*. https://missinglink.ai/guides/convolutional-neural-networks/fully-connected-layers-convolutional-neural-networks-complete-guide/.

[38] *Machine Learning and Deep Learning frameworks and libraries for large-scale data mining: a survey*. https://link.springer.com/article/10.1007/s10462-018-09679-z.

[39] *A Comparative Study of Open Source Deep Learning Frameworks.* https://www.researchgate.net/publication/324454774_A_Comparative_Study_of_Open_Source_Deep_Learning_Frameworks.

[40] Tianqi Chen et al. *MXNet: A Flexible and Efficient Machine LearningLibrary for Heterogeneous Distributed Systems.* https://arxiv.org/pdf/1512.01274.pdf.

[41] The Theano Development Team. *Theano: A Python framework for fast computation of mathematical expressions.* https://arxiv.org/pdf/1605.02688.pdf.

[42] *The Microsoft Cognitive Toolkit.* https://docs.microsoft.com/en-us/cognitive-toolkit/.

[43] Yangqing Jia et al. *Caffe: Convolutional Architecturefor Fast Feature Embedding.* https://arxiv.org/pdf/1408.5093.pdf.

[44] Seiya Tokui et al. *Chainer: a Next-Generation Open SourceFramework for Deep Learning.* http://learningsys.org/papers/LearningSys_2015_paper_33.pdf.

[45] Martín Abadi et al. *TensorFlow: A System for Large-Scale Machine Learning.* https://www.usenix.org/system/files/conference/osdi16/osdi16-abadi.pdf. 2016.

[46] *Effective TensorFlow 2.* https://www.tensorflow.org/guide/effective_tf2.

[47] *Introduction to modules, layers, and models.* https://www.tensorflow.org/guide/intro_to_modules.

[48] *The Sequential model.* https://www.tensorflow.org/guide/keras/sequential_model.

[49] *Distributed training with TensorFlow.* https://www.tensorflow.org/guide/distributed_training.

[50] XinYuan Pitch Patarasuk. *Bandwidth Optimal All-reduce Algorithms for Clusters of Workstations.* https://core.ac.uk/download/pdf/193017016.pdf.

[51] Mu Li et al. *Parameter Server for Distributed Machine Learning.* http://www.cs.cmu.edu/~feixia/files/ps.pdf.

[52] Ayoosh Kathuria. *PyTorch 101, Part 1: Understanding Graphs, Automatic Differentiation and Autograd.* https://blog.paperspace.com/pytorch-101-understanding-graphs-and-automatic-differentiation/.

[53] Adam Paszke et al. *Automatic differentiation in PyTorch*. https://openreview.net/forum?id=BJJsrmfCZ.

[54] Abhinav Ajitsaria. *What is the Python Global Interpreter Lock (GIL)?* https://realpython.com/python-gil/.

[55] Xi Yang Rifat Shahriyar Stephen M. Blackburn and Kathryn S. McKinley. *Taking Off the Gloves with Reference Counting Immix*. https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/rcix-oopsla-2013.pdf.

[56] Shen Li et al. *PyTorch Distributed: Experiences on Accelerating Data Parallel Training*. https://arxiv.org/abs/2006.15704.

[57] *Distributed Data Parallel*. https://pytorch.org/docs/stable/notes/ddp.html.

[58] *DataParallel*. https://pytorch.org/docs/master/generated/torch.nn.DataParallel.html.

[59] *Distributed RPC Framework*. https://pytorch.org/docs/1.5.1/rpc.html.

[60] *TensorBoard: TensorFlow's visualization toolkit*. https://www.tensorflow.org/tensorboard.

[61] *How to use TensorBoard with PyTorch*. https://pytorch.org/tutorials/recipes/recipes/tensorboard_with_pytorch.html.

[62] *PyTorch Profiler*. https://pytorch.org/tutorials/recipes/recipes/profiler.html.

[63] Stephen Bonner Daniel Justus John Brennan and Andrew Stephen McGough. *Predicting the Computational Cost of DeepLearning Models*. https://arxiv.org/pdf/1811.11880.pdf.