ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

«Ανάπτυξη Μαθηματικής Βιβλιοθήκης για Μηχανική Μάθηση και Εφαρμογή σε Real-Time Rendering»

Του φοιτητή
Χαλκίδη Λάζαρου
Αρ. Μητρώου: 164773

Επιβλέπων
Ονοματεπώνυμο Δρ. Διαμαντάρας
Κωνσταντίνος
Βαθμίδα Καθηγητής

21 Απριλίου 2025

Τίτλος Π.Ε. Ανάπτυξη Μαθηματικής Βιβλιοθήκης για Μηχανική Μάθηση και Εφαρμογή σε Real-Time Rendering
Κωδικός Π.Ε. 24126
Ονοματεπώνυμο φοιτητή Χαλκίδης Λάζαρος
Ονοματεπώνυμο εισηγητή Διαμαντάρα Κωνσταντίνος
Ημερομηνία ανάληψης Π.Ε. 19-02-2024
Ημερομηνία περάτωσης Π.Ε. 15-02-2025

«Στους φίλους και την οικογένειά μου, στους καθηγητές μου και στους ανθρώπους με ακόρεστη περιέργεια»

Πρόλογος

Οι βιβλιοθήκες μηχανικής μάθησης είναι συναρπαστικές. Λειτουργούν σαν μαγικά, μπορούν να υπολογίζουν αυτόματα παράγωγους υψηλότερης τάξης και να μεταγλωττίζουν βελτιστοποιημένες συναρτήσεις σε χρόνο εκτέλεσης, απλά δίνοντας μια έκφραση. Ο θαυμασμός γρήγορα μετατράπηκε σε περιέργεια για το πώς όλα λειτουργούν στα παρασκήνια και γεννήθηκε η ιδέα να φτιάξω μια χρησιμοποιώντας την αγαπημένη μου γλώσσα προγραμματισμού. Τα οφέλη πέρα από την ικανοποίηση της περιέργειάς μου ήταν τεράστια καθώς αυτό το έργο αγγίζει μια τεράστια περιοχή προγραμματισμού χαμηλού επιπέδου και έπρεπε να προσαρμοστώ και να εκπαιδεύσω τον εαυτό μου για να εκπληρώσω αυτές τις απαιτήσεις. Τέλος, ελπίζω ότι αυτό μπορεί να είναι χρήσιμο για συμφοιτητές με παρόμοια περιέργεια.

Περίληψη

Ο στόχος αυτής της διατριβής είναι η ανάπτυξη μιας ελαφριάς αλλά πλήρως εξοπλισμένης μαθηματικής βιβλιοθήκης προσαρμοσμένης για εργασίες υπολογισμού και μηχανικής μάθησης. Αυτή η εργασία διερευνά τις προκλήσεις και τις κρίσιμες σκέψεις που εμπλέκονται στο σχεδιασμό ενός τέτοιου συστήματος, ενώ αναλύει σε βάθος τα επιμέρους στοιχεία του. Η εργασία εκτείνεται σε ολόκληρο το φάσμα ανάπτυξης, από τη υλοποίηση της ραχοκοκαλιάς χαμηλού επιπέδου έως την κατασκευή μιας διεπαφής υψηλού επιπέδου που επιτρέπει στους χρήστες να μεταγλωττίζουν βελτιστοποιημένους τελεστές και να σχεδιάζουν νευρωνικά δίκτυα ανεξαρτήτως hardware. Πρωταρχικός στόχος του έργου είναι η σπονδυλότητα, διασφαλίζοντας ότι κάθε επίπεδο της εφαρμογής παραμένει όσο το δυνατόν πιο ανεξάρτητο για να διευκολύνει τις απρόσκοπτες τροποποιήσεις στο μέλλον. Εξίσου σημαντική είναι η διατήρηση μιας απλής αρχιτεκτονικής και κώδικα για την αποτελεσματική εξυπηρέτηση των εκπαιδευτικών σκοπών. Κάθε στοιχείο είναι σχολαστικά σχεδιασμένο από την αρχή, τα οποία στη συνέχεια ενσωματώνονται καταλήγοντας στην τελική διεπαφή. Η προκύπτουσα βιβλιοθήκη θα αξιολογηθεί και θα συγκριθεί με παρόμοιες υπάρχουσες λύσεις.

«Development of a Mathematical Library for Machine Learning and Application to Real-Time Rendering»

«Lazaros Chalkidis»

Abstract

The objective of this thesis is to develop a lightweight yet fully-featured mathematical library tailored for computational and machine learning tasks. This work explores the challenges and critical considerations involved in designing such a system while analyzing each individual component in depth. The thesis spans the entire development spectrum, from implementing the low-level backbone, to constructing a device-agnostic, high-level interface that enables users to compile optimized operators and design neural networks. A primary focus of the project is modularity, ensuring that each layer of the application remains as independent as possible to facilitate seamless modifications in the future. Equally important is maintaining a clean and simple codebase to serve educational purposes effectively. All components are meticulously designed from the ground up, culminating in an integrated interface. The resulting library will be evaluated and benchmarked against similar existing solutions.

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τους φίλους και την οικογένειά μου που με στήριξαν και με ανέχτηκαν καθώς και τον επιβλέποντα καθηγητή Δρ Κωνσταντίνο Διαμαντάρα για τις πολύτιμες διαλέξεις που πρόσφερε, τις εκατοντάδες απορίες που έλυσε και που μου έδωσε την ευκαιρία να αναλάβω αυτό το απαιτητικό έργο

Περιεχόμενα

Κατάλογος Σχημάτων

Συντομογραφίες

Δ.Ε.	Διπλωματική Εργασία
ISA	instruction set architecture
JIT	Just in time (compilation)
AD	Automatic differentiation
AI	Artificial intelligence
API	Application programming interface
ML	Machine learning
NN	Neural network
CNN	Convolutional neural network
RNN	Recurrent neural network
GPU	Graphics processing unit
CPU	Central processing unit
TPU	Tensor processing Unit
DNN	Deep neural network
DL	Deep learning
RL	Reinforcement learning
CAS	Compare and swap
DAG	directed acyclic graph
AOT	ahead of time (compiled)
RAM	Random access memory
ILP	Instruction level parallelism
RMW	Read modify write
DLL	Dynamic link library
SIMD	Single instruction multiple data
ΔΙΠΑΕ	Διεθνές Πανεπιστήμιο Ελλάδος
Π.Ε.	Πτυχιακή Εργασία

Κεφάλαιο 1ο: Εισαγωγή

1.1 Εισαγωγή

Σήμερα, η δημιουργία νευρωνικών δικτύων είναι τόσο απλή όσο η στοίβαξη ενός στρώματος πάνω σε ένα άλλο και η δοκιμή του σε αμέτρητους υπολογιστικούς πόρους που προσφέρονται στο cloud, αλλά φυσικά αυτό δεν συνέβαινε πάντα. Πριν από όλα τα σύγχρονα εργαλεία, οι ερευνητές μηχανικής μάθησης έπρεπε να γράψουν χειροκίνητα όλους τους αλγόριθμους που απαιτούνταν για το μοντέλο τους, συμπεριλαμβανομένου του υπολογισμού των παραγώγων και του βρόχου εκπαίδευσης που είναι τεράστιος όγκος εργασίας, και αυτό δεν συνυπολογίζει καν τις βελτιστοποιήσεις που έπρεπε να γίνουν σε κάθε hardware ώστε το μοντέλο να μπορεί να εκτελεστεί στους περιορισμένους πόρους που υπήρχαν.

Κάπου στην πορεία δημιουργήθηκαν και υιοθετήθηκαν εξαιρετικά βελτιστοποιημένες υπολογιστικές βιβλιοθήκες και πρότυπα όπως το BLAS [1]. Προσέφεραν πολύτιμα δομικά στοιχεία που χρησίμευαν ως πρωτόγονες λειτουργίες για τις περισσότερες εργασίες μηχανικής μάθησης, έτσι ώστε οι ερευνητές να μπορούν να ενδιαφέρονται μόνο για τα μαθηματικά σε υψηλό επίπεδο αντί για ειδικές γνώσεις hardware (ISA) και βελτιστοποιήσεις αλγορίθμων. Στη συνέχεια, η υψηλή ζήτηση σε υπολογισμούς χρησιμοποιώντας πίνακες σε διάφορα πεδία προκάλεσε τη δημιουργία ακόμη μεγαλύτερων αφαιρέσεων όπως το NumPy [2]. Αυτές οι αφαιρέσεις φρόντισαν να χειριστούν πολυδιάστατους πίνακες και τα μαθηματικά σε αυτούς μειώνοντας περαιτέρω το φόρτο της δημιουργίας μιας εφαρμογής μηχανικής μάθησης. Τελικά, στα μέσα της δεκαετίας του 2010, στην παρέλαση της έκρηξης της τεχνητής νοημοσύνης, εμφανίστηκαν συστήματα που μπορούσαν να δημιουργήσουν ολόκληρα νευρωνικά δίκτυα και να τα εκπαιδεύσουν σε μια ποικιλία απο hardware σε γλώσσες υψηλού επιπέδου όπως το Pytorch [3].

Η συνεχής εξέλιξη υποδηλώνει ότι μπορεί να βρισκόμαστε στα πρόθυρα ενός ακόμη μεγαλύτερου, ενός νέου μοντέλου για τέτοιες εφαρμογές, ενός νέου παραδείγματος προγραμματισμού που θα μπορούσε να σημαίνει την εμφάνιση νέων γλωσσών που στοχεύουν ειδικά σε μια αφαίρεση για βελτιστοποιημένους υπολογισμούς σε διαφορετικό hardware και τη διαφοροποίηση των προγραμμάτων, ένα παράδειγμα που είναι γνωστό ως διαφορίσιμος προγραμματισμός. Επομένως, η εμφάνιση ορισμών για τις γλώσσες προγραμματισμού που ακολουθούν αυτό το παράδειγμα και έργα όπως η γλώσσα Mojo [4] [5].

Η παραγόμενη βιβλιοθήκη αυτής της εργασίας περιέχει όλα τα επίπεδα αφαίρεσης το καθένα με τη δική του είσοδο για επέκταση και τροποποίηση. Με υποστήριξη για τους πιο συχνά χρησιμοποιούμενους τύπους δεδομένων, ορισμένες δυνατότητες IO (φόρτωση/αποθήκευση δεδομένων από δίσκο), δυνατότητα JIT μεταγλώττισης βελτιστοποιημένων μαθηματικών συναρτήσεων για GPU και CPU, ώστε ο χρήστης να μπορεί να συγχωνεύει πράξεις και να επιτυγχάνει μεγαλύτερη απόδοση και φυσικά αυτόματος υπολογισμός παραγώγων για εργασίες μηχανικής μάθησης.

1.2 Σκοπός εργασίας

Αυτό το έργο ξεκίνησε ως μικρές υλοποιήσεις για αλγόριθμους μηχανικής μάθησης σε C++ για εξοικείωση με τη γλώσσα και τις έννοιες της μηχανικής μάθησης. Ωστόσο, καθώς η περιέργεια δεν τελειώνει ποτέ, αυτό το έργο επεκτάθηκε πέρα από αυτό που αρχικά προβλεπόταν.

Οι εκπαιδευτικοί στόχοι ήταν το κύριο κίνητρο για αυτήν την εργασία και αυτό το κίνητρο θα παραμείνει και στις μελλοντικές επεκτάσεις καθώς η εκπαίδευση δεν σταματά ποτέ. Όσον αφορά τις απαιτήσεις, ο τελικός στόχος ήταν μια βιβλιοθήκη που χρησιμοποιεί αυτόματο υπολογισμό παραγώγων για να επιτρέψει στο χρήστη να εκπαιδεύσει τα νευρωνικά δίκτυα. Η βασική απαίτηση ήταν ότι η απόδοση των αλγορίθμων δεν θα είναι τίποτα άλλο από εξαιρετική ή τουλάχιστον συγκρίσιμη με τα περισσότερα πλαίσια μηχανικής μάθησης.

Προορίζεται επίσης να δείξει εκτίμηση και θαυμασμό για όλα τα άλλα πλαίσια μηχανικής μάθησης που το ενέπνευσαν, επισημαίνοντας τις δυσκολίες στις αρχιτεκτονικές αποφάσεις, την υλοποίηση και την ενσωμάτωση των στοιχείων και την κλίμακα ενός τέτοιου συστήματος.

1.3 Δομή εργασίας

Η δομή αυτής της εργασίας αποτελείται από 5 κεφάλαια. Το πρώτο κεφάλαιο είναι η εισαγωγή όπου υπάρχει μια ιστορική αναδρομή του τρόπου με τον οποίο εξελίχθηκε ο προγραμματισμός μηχανικής μάθησης μαζί με τους στόχους αυτής της διατριβής. Στη συνέχεια, το δεύτερο κεφάλαιο περιέχει τις θεμελιώδεις γνώσεις που απαιτούνται για την κατασκευή και τη συντήρηση ενός τέτοιου συστήματος. Στο τρίτο κεφάλαιο γίνονται αναφορές και ανασκοπήσεις παρόμοιων συστημάτων από τα οποία εμπνεύστηκε η ιδέα δημιουργίας μιας τέτοιας βιβλιοθήκης. Η πλήρης ανάλυση της αρχιτεκτονικής και υλοποίησης της βιβλιοθήκης είναι το θέμα του 4ου κεφαλαίου. Τέλος, το 5ο κεφάλαιο είναι η τελική ετυμηγορία των αποτελεσμάτων της εργασίας και οι σκέψεις για το μέλλον αυτού του έργου.

Κεφάλαιο 2ο: Θεμελιώδεις γνώσεις

2.1 Gradient

Το gradient είναι μια θεμελιώδης έννοια στον λογισμό και χρησιμοποιείται εκτενώς στη βελτιστοποίηση, τη φυσική, τη μηχανική και πολλά άλλα πεδία για να χαρακτηρίσει τη συμπεριφορά των συναρτήσεων και των πεδίων. Η κλίση μιας συνάρτησης στα μαθηματικά αναφέρεται σε ένα διάνυσμα που αντιπροσωπεύει τόσο την κατεύθυνση όσο και το μέγεθος του μέγιστου ρυθμού μεταβολής της συνάρτησης σε ένα δεδομένο σημείο. Τυπικά, έστω ότι f είναι μια βαθμωτή συνάρτηση πολλών μεταβλητών, που τυπικά δηλώνεται ως $f : \mathbb{R}^n \to \mathbb{R}$. Το gradient της f, που συμβολίζεται ως ∇f ή grad(f), είναι ένα διάνυσμα μερικών παραγώγων :

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{bmatrix} \tag{2.1}$$

Εδώ, το $\frac{\partial f}{\partial x}$ υποδηλώνει τη μερική παράγωγο του f σε σχέση με την i-οστή μεταβλητή x_i. Κάθε στοιχείο $\frac{\partial f}{\partial x_i}$ δίνει το ρυθμό μεταβολής της f προς την κατεύθυνση του x_i-άξονα. Γεωμετρικά, το διάνυσμα κλίσης ∇f δείχνει προς την κατεύθυνση του μεγαλύτερου ρυθμού αύξησης της συνάρτησης f και το μέγεθός του $|\nabla f|$ δίνει το ρυθμό μεταβολής σε αυτήν κατεύθυνση.

2.2 Jacobian matrix

Η Ιακωβιανή μήτρα γενικεύει την έννοια του διανύσματος κλίσης σε υψηλότερες διαστάσεις και συναρτήσεις με διανυσματική τιμή, δηλαδή μια συλλογή όλων των μερικών παραγώγων πρώτης τάξης μιας συνάρτησης με διανυσματική τιμή που είναι διατεταγμένες σε μορφή πίνακα. Μαθηματικά, για μια συνάρτηση $f : \mathbb{R}^n \to \mathbb{R}^m$, ο πίνακας Τζακόμπι J ορίζεται ως:

$$J = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \frac{\partial f_m}{\partial x_2} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix} \tag{2.2}$$

όπου $x \in \mathbb{R}^n$ και $f(x) = (f_1(x), f_2(x), \cdots, f_m(x)) \in \mathbb{R}$›. Εναλλακτικά χρησιμοποιώντας το gradient που είναι ένα διάνυσμα στήλης:

$$\nabla f = \left(\frac{\partial f}{\partial x_1}, \cdots, \frac{\partial f}{\partial x_n} \right)^T \tag{2.3}$$

η παραπάνω έκφραση μπορεί να γραφτεί ως:

$$J = \begin{bmatrix} \nabla^T f_1 \\ \nabla^T f_2 \\ \vdots \\ \nabla^T f_m \end{bmatrix} \tag{2.4}$$

2.3 Hessian matrix

Ο Εσσιανός πίνακας είναι μια μήτρα $m \times m$ που πήρε το όνομά του από τον Γερμανό μαθηματικό Ludwig Otto Hesse που τον ανέπτυξε. Αναφέρεται σε έναν πίνακα μερικών παραγώγων δεύτερης τάξης μιας βαθμωτής συνάρτησης και περιγράφει την τοπική καμπυλότητα μιας συνάρτησης πολλαπλών μεταβλητών. Δίνεται δύο φορές παραγωγίσιμη βαθμωτή συνάρτηση $f : \mathbb{R}^n \to \mathbb{R}$, ο Εσσιανός πίνακας H(f) ορίζεται ως:

$$H(f)(x) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix} \tag{2.5}$$

Σε αυτόν τον πίνακα κάθε στοιχείο $H(f)_{ij}$ είναι η μερική παράγωγος δεύτερης τάξης της f σε σχέση με x_i και x_j, δηλαδή $H(f)_{ij} = \frac{\partial^2 f}{\partial x_i \partial x_j}$. Τα διαγώνια στοιχεία είναι οι δεύτερες επιμέρους παράγωγοι ως προς το ίδια μεταβλητή x_i, δηλαδή $\frac{\partial^2 f}{\partial x_i^2}$. Τα εκτός της διαγωνίου στοιχεία αντιπροσωπεύουν τις μικτές μερικές παραγώγους $\frac{\partial^2 f}{\partial x_i \partial x_j}$. Ο εσσιανός πίνακας μπορεί επίσης να εκφραστεί ως ο μετατιθέμενος του ιακωβιανού πίνακα του gradient μιας συνάρτησης :

$$H(f) = J(\nabla f)^T \tag{2.6}$$

2.4 Dual numbers

Οι δυϊκοί αριθμοί που εισήχθησαν για πρώτη φορά το 1873 από τον William Clifford, είναι ένα υπερσύνθετο αριθμητικό σύστημα. Ομοίως με τους μιγαδικούς αριθμούς, οι δυϊκοί αριθμοί έχουν τις δικές τους καθορισμένες πράξεις και αποτελούνται από δύο πραγματικές συνιστώσες ας πούμε $x, y \in R$. Επιπλέον, χρησιμοποιούν την απειροελάχιστη ποσότητα με το σύμβολο ϵ, η οποία έχει την ιδιότητα:

$$\epsilon^2 = 0 \text{ με } \epsilon \neq 0 \tag{2.7}$$

Τέλος, μαθηματικά μπορούν να εκφραστούν με τη μορφή:

$$dual(x, y) = x + y\epsilon \tag{2.8}$$

2.5 Machine Learning

2.5.1 Εισαγωγή

Η μηχανική μάθηση είναι ένα υποσύνολο της τεχνητής νοημοσύνης που βασίζεται σε στατιστικά μοντέλα και αναπαραστάσεις δεδομένων αντί σε καθορισμένους κανόνες. Χρησιμοποιεί αλγόριθμους βελτιστοποίησης για να προσαρμόσει μια εσωτερική κατάσταση σε δεδομένα εισόδου για γενίκευση και εξαγωγή συμπερασμάτων από άγνωστα δεδομένα. Σε αντίθεση με την τεχνητή νοημοσύνη που βασίζεται σε κανόνες, η μηχανική μάθηση είναι μια μέθοδος καλύτερης προσπάθειας προσέγγισης και αναμένεται να παρουσιάζει ποσοστά λάθους με το αντάλλαγμα ότι απαιτεί ελάχιστη ή καθόλου ανθρώπινη παρέμβαση και μπορεί να γενικεύσει, απλώς μέσω δεδομένων αντί να απαιτεί από έναν άνθρωπο να ορίζει χειροκίνητα κανόνες.

2.5.2 Neural Networks

Το νευρωνικό δίκτυο είναι ένα υπολογιστικό μοντέλο εμπνευσμένο από τη δομή και τη λειτουργία των βιολογικών νευρωνικών δικτύων, ιδιαίτερα εκείνων που βρίσκονται στον ανθρώπινο εγκέφαλο. Αποτελείται από διασυνδεδεμένα στρώματα κόμβων, γνωστών ως τεχνητοί νευρώνες ή μονάδες. Κάθε νευρώνας επεξεργάζεται δεδομένα εισόδου και παράγει μια έξοδο, συνήθως μέσω μιας μη γραμμικής συνάρτησης ενεργοποίησης. Οι συνδέσεις μεταξύ των νευρώνων, που αντιπροσωπεύονται από βάρη, προσαρμόζονται κατά τη διάρκεια της προπόνησης για να ελαχιστοποιηθεί το σφάλμα στις προβλέψεις που γίνονται από το δίκτυο. Η έννοια των νευρωνικών δικτύων έχει τις ρίζες της στις αρχές του 20ου αιώνα, εμπνευσμένη από τη λειτουργία του ανθρώπινου εγκεφάλου.

Το πρωτοποριακό έργο των Warren McCulloch και Walter Pitts το 1943 έθεσε τα θεμέλια διαμορφώνοντας ένα απλοποιημένο νευρωνικό δίκτυο χρησιμοποιώντας δυαδικά κατώφλια, τα οποία μπορούσαν να εκτελέσουν λογικές πράξεις. Αυτό το μοντέλο, γνωστό ως νευρώνας McCulloch-Pitts, έγινε θεμελιώδες δομικό στοιχείο για το πεδίο των τεχνητών νευρωνικών δικτύων [6]. Το 1958, ο Frank Rosenblatt εισήγαγε το perceptron, έναν τύπο τεχνητού νευρωνικού δικτύου σχεδιασμένο για προβλήματα δυαδικής ταξινόμησης [7]. Το perceptron ήταν σημαντικό επειδή ήταν ένα από τα πρώτα νευρωνικά δίκτυα ικανά να μαθαίνουν από δεδομένα μέσω εποπτευόμενης μάθησης. Παρά την αρχική του υπόσχεση, οι περιορισμοί του perceptron, ιδιαίτερα η αδυναμία του να λύσει μη γραμμικά διαχωρισμένα προβλήματα (π.χ. το πρόβλημα XOR), οδήγησε σε μια περίοδο μειωμένου ενδιαφέροντος για τα νευρωνικά δίκτυα κατά τη δεκαετία του 1970.

Η αναζωπύρωση του ενδιαφέροντος για τα νευρωνικά δίκτυα σημειώθηκε τη δεκαετία του 1980, που χαρακτηρίστηκε από την ανάπτυξη της backpropagation, μιας μεθόδου για την εκπαίδευση πολυεπίπεδων νευρωνικών δικτύων [8]. Αυτός ο αλγόριθμος, που διαδόθηκε από τους Rumelhart, Hinton και Williams το 1986, επέτρεψε την εκπαίδευση πιο περίπλοκων δικτύων, γνωστών ως πολυστρωματικά perceptrons (MLPs).

Οι επόμενες δεκαετίες είδαν την εξέλιξη των αρχιτεκτονικών νευρωνικών δικτύων, συμπεριλαμβανομένων των συνελικτικών νευρωνικών δικτύων (CNN) και των επαναλαμβανόμενων νευρωνικών δικτύων (RNN), τα οποία έχουν γίνει αναπόσπαστο μέρος του πεδίου της βαθιάς μάθησης [9].

Μαθηματικά, ένα νευρωνικό δίκτυο μπορεί να οριστεί ως μια σειρά από συναρτήσεις $f(x)$ ότι τα δεδομένα εισόδου χαρτών x για την παραγωγή προβλέψεων. Για ένα απλό νευρωνικό δίκτυο προώθησης με L στρώματα, η έξοδος y μπορεί να εκφραστεί ως:

$$y = f(x) = f_L(f_{L-1}(...f_2(f_1(x))...)) \tag{2.9}$$

όπου f_i αντιπροσωπεύει τον μετασχηματισμό στο στρώμα i, που συχνά αποτελείται από έναν γραμμικό μετασχηματισμό που ακολουθείται από μια μη γραμμική συνάρτηση ενεργοποίησης. Τα νευρωνικά δίκτυα είναι ικανά να μαθαίνουν πολύπλοκα μοτίβα από μεγάλα σύνολα δεδομένων, καθιστώντας τα κατάλληλα για ένα ευρύ φάσμα εφαρμογών, συμπεριλαμβανομένης της αναγνώρισης εικόνας, της επεξεργασίας φυσικής γλώσσας και των αυτόνομων συστημάτων. Η δύναμη των νευρωνικών δικτύων έγκειται στην ικανότητά τους να προσεγγίζουν οποιαδήποτε συνεχή συνάρτηση με επαρκή δεδομένα και υπολογιστικούς πόρους, μια ιδιότητα γνωστή ως θεώρημα καθολικής προσέγγισης [10].

2.5.3 Unsupervised learning

Η μάθηση χωρίς επίβλεψη είναι ένας τύπος μηχανικής μάθησης που λειτουργεί χωρίς δεδομένα με ετικέτα. Σε αυτή την προσέγγιση, το μοντέλο έχει ως αποστολή την ομαδοποίηση παρόμοιων σημείων δεδομένων (clustering) ή τη μείωση των διαστάσεων δεδομένων (dimensionality reduction). Σε αντίθεση με την μάθηση με επίβλεψη, όπου το μοντέλο εκπαιδεύεται σε ζεύγη εισόδου-εξόδου με ετικέτα, η μάθηση χωρίς επίβλεψη επικεντρώνεται στην αποκάλυψη κρυφών δομών ή αναπαραστάσεων σε δεδομένα. Οι κοινοί αλγόριθμοι περιλαμβάνουν ομαδοποίηση k-means, ανάλυση κύριου συστατικού (PCA) και αυτοκωδικοποιητές.

2.5.4 Supervised learning

Η μάθηση με επίβλεψη είναι μια μέθοδος μηχανικής μάθησης που χρησιμοποιεί δεδομένα με ετικέτα για την εκπαίδευση ενός μοντέλου. Σε αυτήν την προσέγγιση, το σύνολο δεδομένων εκπαίδευσης αποτελείται από ζεύγη εισόδου-εξόδου, με την είσοδο να είναι τα χαρακτηριστικά και την έξοδο να είναι οι ετικέτες στόχου. Το μοντέλο μαθαίνει να αντιστοιχίζει τις εισόδους στις σωστές εξόδους ελαχιστοποιώντας τη διαφορά μεταξύ των προβλέψεών του και των πραγματικών ετικετών. Αυτή η διαδικασία συνήθως καθοδηγείται από μια συνάρτηση σφάλματος, όπως το μέσο τετραγωνικό σφάλμα ή η διασταυρούμενη εντροπία. Οι κοινοί αλγόριθμοι εποπτευόμενης μάθησης περιλαμβάνουν τη γραμμική παλινδρόμηση, τις μηχανές διανυσμάτων υποστήριξης και τα νευρωνικά δίκτυα [11].

2.5.5 Deep learning

Η βαθιά μάθηση (DL) είναι ένα υποσύνολο της μηχανικής μάθησης που εστιάζει στη χρήση νευρωνικών δικτύων με πολλά επίπεδα για τη μοντελοποίηση πολύπλοκων μοτίβων στα δεδομένα. Στην καρδιά του DL, εμπνευσμένο από τη δομή και τη λειτουργία του ανθρώπινου εγκεφάλου, βρίσκονται τα βαθιά νευρωνικά δίκτυα (DNNs), όπου οι νευρώνες σε κάθε στρώμα συνδέονται με εκείνους στα γειτονικά στρώματα, σχηματίζοντας ένα δίκτυο. Στα παραδοσιακά ρηχά νευρωνικά δίκτυα, υπάρχουν συνήθως ένα ή δύο κρυφά επίπεδα, ενώ τα DNN μπορεί να έχουν δεκάδες ή και εκατοντάδες επίπεδα, καθένα

ικανό να εξάγει όλο και πιο αφηρημένα χαρακτηριστικά από τα δεδομένα που επιτρέπουν στο μοντέλο να μάθει ιεραρχικές αναπαραστάσεις [12]. Αυτά τα δίκτυα είναι ιδιαίτερα κατάλληλα για μεγάλα σύνολα δεδομένων και έχουν επιτύχει επιδόσεις αιχμής σε τομείς όπως η αναγνώριση εικόνας, η επεξεργασία φυσικής γλώσσας και η αναγνώριση ομιλίας [13].

Η εκπαίδευση των μοντέλων DL γίνεται τυπικά χρησιμοποιώντας backpropagation, όπου η κλίση της συνάρτησης απώλειας υπολογίζεται και διαδίδεται προς τα πίσω μέσω του δικτύου για την ενημέρωση των βαρών των νευρώνων. Αυτή η διαδικασία δεδομένου του μεγέθους του βαθέως δικτύου έχει υψηλές απαιτήσεις σε υπολογιστικούς πόρους, και συχνά απαιτεί εξειδικευμένο υλικό, όπως Κάρτες Γραφικών (GPU) ή εξειδικευμένες μονάδες επεξεργασίας τένσορα (TPU) για την επιτάχυνση της εκμάθησης [14].

Η βαθιά μάθηση έχει γνωστούς περιορισμούς, όπως η ανάγκη για μεγάλα σύνολα δεδομένων με ετικέτες, υψηλό υπολογιστικό κόστος και ζητήματα εξηγησιμότητας. Η πρόσφατη έρευνα επικεντρώνεται στην υπέρβαση αυτών των προκλήσεων μέσω τεχνικών όπως η transfer learning και η συμπίεση μοντέλων (model compression) [15].

2.5.6 Reinforcement learning

Η Μάθηση ενίσχυση (RL) είναι ένας τύπος μηχανικής μάθησης όπου ένας πράκτορας μαθαίνει να λαμβάνει αποφάσεις αλληλεπιδρώντας με ένα περιβάλλον, με στόχο τη μεγιστοποίηση μιας σωρευτικής ανταμοιβής. Σε αντίθεση με την μάθηση με επίβλεψη, όπου ένα μοντέλο μαθαίνει από ένα σταθερό σύνολο δεδομένων, το RL λειτουργεί με βάση την ανατροφοδότηση από τις δικές του ενέργειες, καθιστώντας το ιδιαίτερα κατάλληλο για προβλήματα όπου οι βέλτιστες ενέργειες δεν είναι γνωστές εκ των προτέρων. Τα βασικά στοιχεία των συστημάτων RL περιλαμβάνουν τον πράκτορα, το περιβάλλον, την κατάσταση, τη δράση και την ανταμοιβή [16]. Ο πράκτορας παρατηρεί την τρέχουσα κατάσταση του περιβάλλοντος, επιλέγει μια ενέργεια και λαμβάνει ανατροφοδότηση με τη μορφή ανταμοιβής, η οποία χρησιμοποιείται για να ενημερώσει τη στρατηγική του για μελλοντική λήψη αποφάσεων. Ο στόχος είναι να αναπτυχθεί μια πολιτική, που συχνά αναπαρίσταται ως συνάρτηση που αντιστοιχίζει καταστάσεις σε ενέργειες, που μεγιστοποιεί την αναμενόμενη σωρευτική ανταμοιβή με την πάροδο του χρόνου [17].

Οι βασικοί αλγόριθμοι στο RL χωρίζονται σε δύο κύριες κατηγορίες: προσεγγίσεις χωρίς μοντέλο και προσεγγίσεις που βασίζονται σε μοντέλα. Οι αλγόριθμοι χωρίς μοντέλα, όπως Q-learning και policy gradient methods, μαθαίνουν απευθείας από την αλληλεπίδραση με το περιβάλλον χωρίς να κατασκευάζουν ένα μοντέλο του. Από την άλλη πλευρά, οι προσεγγίσεις που βασίζονται σε μοντέλα όπως Deep Q-learning προσπαθούν να δημιουργήσουν ένα μοντέλο της δυναμικής του περιβάλλοντος για τη λήψη πιο τεκμηριωμένων αποφάσεων [18].

Η RL έχει επιδείξει επιτυχία σε διάφορους τομείς, όπως η ρομποτική, τα παιχνίδια, τα αυτόνομα οχήματα και τα συστήματα συστάσεων. Συγκεκριμένα, κέρδισε ευρεία προσοχή όταν το DeepMind της Google ανέπτυξε το AlphaGo, το οποίο χρησιμοποίησε έναν συνδυασμό RL και βαθιά νευρωνικά δίκτυα για να νικήσει τους παγκόσμιους πρωταθλητές στο παιχνίδι Go [19]. Ωστόσο, προκλήσεις όπως η αναποτελεσματικότητα του δείγματος, η αντιστάθμιση εξερεύνησης-εκμετάλλευσης, η εξηγησιμότητα και η επεκτασιμότητα παραμένουν κρίσιμοι τομείς της συνεχιζόμενης έρευνας [20].

2.5.7 Overfitting

Η υπερπροσαρμογή συμβαίνει όταν ένα μοντέλο μηχανικής μάθησης μαθαίνει τον θόρυβο και τα συγκεκριμένα μοτίβα στα δεδομένα εκπαίδευσης αντί για τα υποκείμενα γενικά μοτίβα που μπορούν να εφαρμοστούν σε νέα δεδομένα. Αυτό συμβαίνει συνήθως όταν το μοντέλο είναι πολύ περίπλοκο, όπως όταν έχει πάρα πολλές παραμέτρους σε σχέση με τον όγκο των δεδομένων εκπαίδευσης. Ως αποτέλεσμα, ενώ το μοντέλο αποδίδει πολύ καλά στα δεδομένα εκπαίδευσης, η απόδοσή του υποβαθμίζεται σημαντικά στα δεδομένα ανάκλησης ή σε εφαρμογές πραγματικού κόσμου. Η υπερπροσαρμογή μπορεί να εντοπιστεί όταν το σφάλμα εκπαίδευσης είναι χαμηλό, αλλά το σφάλμα στα δεδομένα της ανάκλησης είναι σημαντικά υψηλότερο. Αυτή η απόκλιση δείχνει ότι το μοντέλο έχει απομνημονεύσει τα δεδομένα εκπαίδευσης αντί να γενικεύει από αυτά. Οι συνήθεις μέθοδοι για την πρόληψη της υπερπροσαρμογής περιλαμβάνουν τη χρήση απλούστερων μοντέλων, την αύξηση του μεγέθους του συνόλου δεδομένων εκπαίδευσης, την εφαρμογή τεχνικών όπως Dropout κατά τη διάρκεια της εκπαίδευσης μοντέλων.

2.5.8 Dropout

Το Dropout είναι μια τεχνική που χρησιμοποιείται στα νευρωνικά δίκτυα, για την αποφυγή υπερπροσαρμογής. Η μέθοδος περιλαμβάνει την τυχαία απενεργοποίηση ενός υποσυνόλου νευρώνων κατά τη διάρκεια κάθε επανάληψης προπόνησης. Αυτό αναγκάζει το δίκτυο να μάθει πιο ισχυρά χαρακτηριστικά αποτρέποντας οποιονδήποτε μεμονωμένο νευρώνα να εξειδικεύεται υπερβολικά στα δεδομένα. Κατά συνέπεια, η εγκατάλειψη βοηθά το μοντέλο να γενικεύει καλύτερα σε μη ορατά δεδομένα. Κατά τη διάρκεια της προπόνησης, κάθε νευρώνας απενεργοποιείται ανεξάρτητα με μια ορισμένη πιθανότητα p, ενώ όλοι οι νευρώνες είναι ενεργοί κατά την εξαγωγή συμπερασμάτων. Μαθηματικά, αυτό μπορεί να εφαρμοστεί με δειγματοληψία από την κατανομή bernoulli για τη δημιουργία μιας μάσκας δηλαδή $M \sim Bernoulli(p)$, $M \in \mathcal{R}^{input\_shape}$ και πολλαπλασιάζοντας κατά στοιχείο με την είσοδο x.

$$Dropout(x) = \begin{cases} x \odot M, & \text{if training} \\ x, & \text{otherwise} \end{cases} \tag{2.10}$$

Το Dropout έχει αποδειχθεί ότι μειώνει σημαντικά την υπερπροσαρμογή και βελτιώνει την απόδοση των νευρωνικών δικτύων, ειδικά σε μοντέλα μεγάλης κλίμακας [21].

2.5.9 Weight decay

Γνωστό και ως ομαλοποίηση L2, είναι μια τεχνική που χρησιμοποιείται στη μηχανική μάθηση για την αποφυγή υπερβολικής προσαρμογής προσθέτοντας μια ποινή στη συνάρτηση κόστους κατά τη διάρκεια της εκπαίδευσης μοντέλων αλλά μπορεί επίσης να εφαρμοστεί στον κανόνα ενημέρωσης ως μέρος του βελτιστοποιητή και όχι απευθείας στη συνάρτηση κόστους [22]. Ο όρος ποινής είναι ανάλογος με το άθροισμα των τετραγωνικών τιμών των βαρών του μοντέλου. Αυτό ενθαρρύνει το μοντέλο να διατηρεί μικρότερες τιμές βάρους. Μαθηματικά, το weight decay τροποποιεί τη συνάρτηση κόστους προσθέτοντας έναν όρο ομαλοποίησης, όπως φαίνεται παρακάτω:

$$L_{\text{total}} = L_{\text{data}} + \lambda \sum_i w_i^2 \tag{2.11}$$

όπου L_{total} είναι το συνολικό κόστος, L_{data} είναι η αρχικό κόστος (π.χ. διασταυρούμενη εντροπία ή μέσο τετράγωνο σφάλμα), λ είναι η παράμετρος ομαλοποίησης και w_i είναι τα βάρη του μοντέλου. Η υπερπαράμετρος λ ελέγχει την ισχύ της ομαλοποίησης. Οι υψηλότερες τιμές ενθαρρύνουν μικρότερα βάρη, αλλά μπορεί επίσης να οδηγήσουν σε κακή προσαρμογή. Το weight decay είναι ιδιαίτερα χρήσιμο σε μοντέλα υψηλών διαστάσεων, όπου η υπερπροσαρμογή είναι συνηθισμένη. Περιορίζοντας το μέγεθος των βαρών, το μοντέλο αναγκάζεται να βασίζεται σε συνδυασμό πολλών μικρών βαρών αντί λίγων μεγάλων, προωθώντας τη γενίκευση και εμποδίζοντάς το να ταιριάζει με το θόρυβο στα δεδομένα προπόνησης [23].

2.5.10 Vanishing/Exploding gradients problem

Τα προβλήματα εξασθενούσων ή απότομων διορθώσεων (Vanishing/Exploding Gradients) είναι βασικές προκλήσεις που αντιμετωπίζονται κατά την εκπαίδευση των βαθιών νευρωνικών δικτύων. Αυτά τα ζητήματα προκύπτουν κυρίως λόγω της φύσης των τεχνικών βελτιστοποίησης που βασίζονται σε διάνυσμα κλίσης, ιδιαίτερα όταν χρησιμοποιείται backpropagation για την ενημέρωση των βαρών σε νευρωνικά δίκτυα με πολλαπλά επίπεδα.

Το πρόβλημα εξαφάνισης των gradients προκύπτει όταν τα gradients που χρησιμοποιούνται για την ενημέρωση των βαρών γίνονται υπερβολικά μικρά καθώς διαδίδονται προς τα πίσω μέσω του δικτύου. Ως αποτέλεσμα, τα προηγούμενα επίπεδα του δικτύου λαμβάνουν ελάχιστες ενημερώσεις, οδηγώντας σε αργή ή στάσιμη μάθηση. Αυτό το πρόβλημα είναι ιδιαίτερα διαδεδομένο σε βαθιά δίκτυα με συναρτήσεις ενεργοποίησης όπως η σιγμοειδής ή η υπερβολική εφαπτομένη, όπου οι παράγωγοι των συναρτήσεων περιορίζονται μεταξύ 0 και 1, προκαλώντας μια προοδευτική συρρίκνωση των κλίσεων καθώς διαδίδονται πίσω. Κατά συνέπεια, το δίκτυο δυσκολεύεται να μάθει ιεραρχικές αναπαραστάσεις [24].

Αντίθετα, το πρόβλημα της έκρηξης των gradients εμφανίζεται όταν τα gradients αυξάνονται εκθετικά κατά τη διάρκεια του backpropagation. Αυτό μπορεί να συμβεί όταν τα βάρη στο δίκτυο είναι μεγάλα, οδηγώντας σε υπερβολικά μεγάλες ενημερώσεις στις παραμέτρους του μοντέλου. Τα μεγάλα gradients μπορεί να προκαλέσουν απόκλιση του μοντέλου, οδηγώντας σε ασταθή εκπαίδευση και τελικά αποτυχία σύγκλισης [25]. Αυτό το ζήτημα παρατηρείται συχνά σε επαναλαμβανόμενα νευρωνικά δίκτυα (RNN) και σε άλλες βαθιές αρχιτεκτονικές, όπου διαδοχικοί πολλαπλασιασμοί πινάκων μπορεί να προκαλέσουν την έκρηξη των τιμών παραγώγων.

Για την εξασθένηση παραγωγών, έχουν εισαχθεί τεχνικές όπως υπολειπόμενες συνδέσεις [26] και εναλλακτικές σε παραδοσιακές συναρτήσεις ενεργοποίησης, όπως η rectified linear unit (ReLU)

$$relu(x) = \begin{cases} x, & \text{if } x > 0 \\ 0, & \text{otherwise} \end{cases} \tag{2.12}$$

καθώς βοηθούν στη διατήρηση ισχυρότερων παραγωγών αποφεύγοντας το πρόβλημα της εξαφάνισης.

Για τις εκρήξεις στις τιμές των παραγωγών, μέθοδοι όπως η αποκοπή των gradients χρησιμοποιούνται συνήθως για να περιορίσουν το μέγεθος των gradients κατά τη διάδοση προς τα πίσω [27].

2.5.11 Training

2.5.11.1 Εισαγωγή

Ένα νευρωνικό δίκτυο παίρνει εισόδους και παράγει εξόδους και είναι κατά μία έννοια συνάρτηση. Η εκπαίδευση ενός νευρωνικού δικτύου είναι η διαδικασία προσαρμογής των βαρών του W για να προσεγγίσει μια επιθυμητή συνάρτηση $g(x)$. Ο αλγόριθμος εκπαίδευσης είναι μια επαναληπτική διαδικασία που απαιτεί επισημασμένα δεδομένα,ένα νευρωνικό δίκτυο, μια συνάρτηση απώλειας και έναν βελτιστοποιητή. Για κάθε επανάληψη εκτελείται μια διαδικασία 3 βημάτων.

1. Υπολογισμός των εξόδων του νευρωνικού δικτύου για ένα σημείο δεδομένων ή παρτίδα.
2. Υπολογισμός των σφαλμάτων για κάθε στρώμα σύμφωνα με τη συνάρτηση απώλειας χρησιμοποιώντας back propagation.
3. Ενημέρωση των βαρών του νευρωνικού δικτύου ανάλογα με τα σφάλματα χρησιμοποιώντας βελτιστοποιητή.

Η παραπάνω διαδικασία θα εκτελεστεί έως ότου ικανοποιηθεί το κριτήριο διακοπής, που είναι συνήθως ένας μέγιστος αριθμός επαναλήψεων, ή η συνάρτηση απώλειας σε κάποια επανάληψη είναι μικρότερη από κάποιο χαμηλότερο όριο που έχει ορίσει ο χρήστης ή η συνάρτηση απώλειας δεν μειώνεται πλέον σημαντικά.

2.5.11.2 Loss function

Για τη μέτρηση της διαφοράς μεταξύ των κατά προσέγγιση εξόδων και των πραγματικών τιμών στόχων, χρησιμοποιείται μια συνάρτηση απώλειας ή κόστους $\mathcal{L}$. Η παράγωγος της συνάρτησης απώλειας παρέχει μια προσέγγιση του τρόπου με τον οποίο αλλάζει το $\mathcal{L}$ σε σχέση με μικρές διαταραχές του W. Επειδή το gradient $\nabla_w \mathcal{L}$ είναι ένα διάνυσμα που δείχνει προς την κατεύθυνση της πιο απότομης αύξησης, η μετακίνηση σε κατευθύνσεις αντίθετες από αυτήν θα μειώσει την τιμή του $\mathcal{L}$.

$$\nabla_w \mathcal{L} = \left[\frac{\partial \mathcal{L}}{\partial w_1}, \cdots, \frac{\partial \mathcal{L}}{\partial w_n} \right] \tag{2.13}$$

Αυτή η διαδικασία συγκλίνει σε ένα κρίσιμο σημείο, η απώλεια δεν μειώνεται πλέον και ανάλογα με την επιφάνεια της συνάρτησης αλλά και το βήμα ενημέρωσης αυτό το σημείο θα μπορούσε να είναι είτε ένα σημείο σέλας, ένα τοπικό ελάχιστο ή ένα ολικό ελάχιστο [28].

2.5.11.3 Optimizer

Στο πλαίσιο των νευρωνικών δικτύων, ένας βελτιστοποιητής είναι ένας αλγόριθμος ή ένα σύνολο τεχνικών που έχουν σχεδιαστεί για να προσαρμόζουν τα βάρη του μοντέλου κατά τη διάρκεια της εκπαίδευσης για την ελαχιστοποίηση της συνάρτησης απώλειας, Η βασική αρχή πίσω από τους περισσότερους

βελτιστοποιητές είναι η επαναληπτική ενημέρωση των παραμέτρων χρησιμοποιώντας το gradient της συνάρτησης απώλειας γνωστή και ως βαθμωτή κατάβαση:

$$W \leftarrow W - \gamma \nabla_W \mathcal{L} \tag{2.14}$$

Ο ρυθμός εκμάθησης, που συμβολίζεται ως γ, είναι μια κρίσιμη υπερπαράμετρος στη βελτιστοποίηση νευρωνικών δικτύων. Καθορίζει το μέγεθος του βήματος σε κάθε επανάληψη κατά την ενημέρωση των παραμέτρων του μοντέλου. Ο μικρός ρυθμός μάθησης οδηγεί σε αργή σύγκλιση, απαιτώντας περισσότερες επαναλήψεις για να επιτευχθεί το ελάχιστο. Ωστόσο, παρέχει πιο ακριβείς ενημερώσεις κοντά στο ελάχιστο. Ο μεγάλος ρυθμός μάθησης επιταχύνει τη σύγκλιση αλλά κινδυνεύει να υπερβεί το ελάχιστο ή να προκαλέσει αστάθεια στη διαδικασία εκπαίδευσης. Έχουν αναπτυχθεί διάφορες παραλλαγές που στοχεύουν στη βελτίωση της απόδοσης της κλίσης κατάβασης [28].

2.5.11.4 Stochastic gradient descent (SGD)

Ενώ η βαθμωτή κατάβαση υπολογίζει τη διαβάθμιση ολόκληρου του συνόλου δεδομένων πριν από την ενημέρωση των παραμέτρων του μοντέλου. Η στοχαστική βαθμωτή κατάβαση ενημερώνει τις παραμέτρους του μοντέλου χρησιμοποιώντας το gradient που υπολογίζεται από ένα μόνο τυχαία επιλεγμένο σημείο δεδομένων (ή μια μικρή παρτίδα). Αυτό έχει αποδειχθεί ότι βελτιώνει την ταχύτητα σύγκλισης [28].

2.5.11.5 Momentum

Η ορμή είναι μια τεχνική που χρησιμοποιείται στη βελτιστοποίηση για την επιτάχυνση της σύγκλισης και τη μείωση των ταλαντώσεων, ειδικά σε σενάρια με υψηλή καμπυλότητα, θορυβώδεις κλίσεις ή ρηχές κοιλάδες. Λειτουργεί ενσωματώνοντας τις προηγούμενες διορθώσεις στην τρέχουσα, εξομαλύνοντας αποτελεσματικά τη διαδρομή βελτιστοποίησης. Με ορμή, η ενημέρωση συνδυάζει την τρέχουσα διόρθωση με ένα κλάσμα της προηγούμενης, δημιουργώντας έναν όρο ταχύτητας. Χρησιμοποιεί έναν όρο ορμής μ που είναι μια υπέρ παράμετρος (συνήθως γύρω στο 0,9) και ένα διάνυσμα ταχύτητας v. Ο κανόνας ενημέρωσης στην επανάληψη t εκφράζεται ως:

$$v_t = \mu v_{t-1} + \gamma \nabla \mathcal{L}(W_t) \tag{2.15}$$

$$W_t = W_{t-1} - v_t \tag{2.16}$$

2.5.11.6 Adaptive methods

Οι μέθοδοι προσαρμοστικής βελτιστοποίησης προσαρμόζουν τον ρυθμό εκμάθησης για κάθε παράμετρο με βάση τις ιστορικές διορθώσεις, επιτρέποντας πιο αποτελεσματικές και προσαρμοσμένες ενημερώσεις κατά τη διάρκεια της προπόνησης, κλιμακώνοντας δυναμικά τους ρυθμούς με βάση το μέγεθος των προηγούμενων κλίσεων. Μια δημοφιλής επιλογή είναι ο αλγόριθμος Adam [29]:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) \nabla_W \mathcal{L} \tag{2.17}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) (\nabla_W \mathcal{L})^2 \tag{2.18}$$

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1-\beta_2^t} \tag{2.19}$$

$$W_{t+1} = W_t - \frac{\gamma \hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \tag{2.20}$$

Όπου m και v είναι οι κινητοί μέσοι όροι των gradients και των τετραγώνων gradients, β_1, και β_2 είναι ρυθμοί αποσύνθεσης, και το ϵ είναι μια μικρή σταθερά για την αποφυγή διαίρεσης με το μηδέν.

2.5.12 Recurrent Neural Networks (RNNs)

2.5.12.1 Εισαγωγή

Τα αναδρομικά νευρωνικά δίκτυα (RNN) είναι μια κατηγορία τεχνητών νευρωνικών δικτύων σχεδιασμένων για δεδομένα σε μορφή ακολουθίας, καθιστώντας τα εξαιρετικά αποτελεσματικά σε προβλήματα όπου οι προηγούμενες είσοδοι επηρεάζουν μελλοντικές, όπως η πρόβλεψη χρονοσειρών, η επεξεργασία φυσικής γλώσσας και η αναγνώριση ομιλίας. Σε αντίθεση με τα παραδοσιακά δίκτυα, τα RNN έχουν συνδέσεις που σχηματίζουν κατευθυνόμενους κύκλους, επιτρέποντάς τους μια κρυφή κατάσταση που διατηρεί πληροφορίες σχετικά με προηγούμενες εισόδους. Τα RNN λειτουργούν με την επεξεργασία ακολουθιών διανυσμάτων εισόδου $x^{(1)}, \cdots, x^{(n)}$. Σε κάθε χρονικό βήμα, το δίκτυο ενημερώνει την κρυφή του κατάσταση με βάση την τρέχουσα είσοδο και την προηγούμενη κρυφή κατάσταση. Ο κανόνας ενημέρωσης κρυφής κατάστασης δίνεται από:

$$h^{(t)} = f(W_{xh}x^{(t)} + W_{hh}h^{(t-1)} + b_h) \tag{2.21}$$

όπου: W_{xh} και W_{hh} είναι πίνακες βάρους που συνδέουν τα επίπεδα εισόδου με κρυφά και κρυφά με κρυφά, αντίστοιχα. b_h είναι το διάνυσμα πόλωσης. f είναι μια συνάρτηση ενεργοποίησης, συχνά μια μη γραμμική συνάρτηση όπως η υπερβολική εφαπτομένη (tanh) ή η ReLU. Η έξοδος σε κάθε χρονικό βήμα μπορεί στη συνέχεια να υπολογιστεί ως εξής:

$$y^{(t)} = \mathbb{F}(W_{hy}h^{(t)} + b_y) \tag{2.22}$$

όπου: W_{hy} είναι η μήτρα βάρους μεταξύ του κρυφού και του επιπέδου εξόδου. b_y είναι το διάνυσμα πόλωσης εξόδου. $\mathbb{F}$ είναι μια συνάρτηση ενεργοποίησης για το επίπεδο εξόδου, η οποία μπορεί να ποικίλλει ανάλογα με τη συγκεκριμένη εργασία (π.χ. softmax για ταξινόμηση).

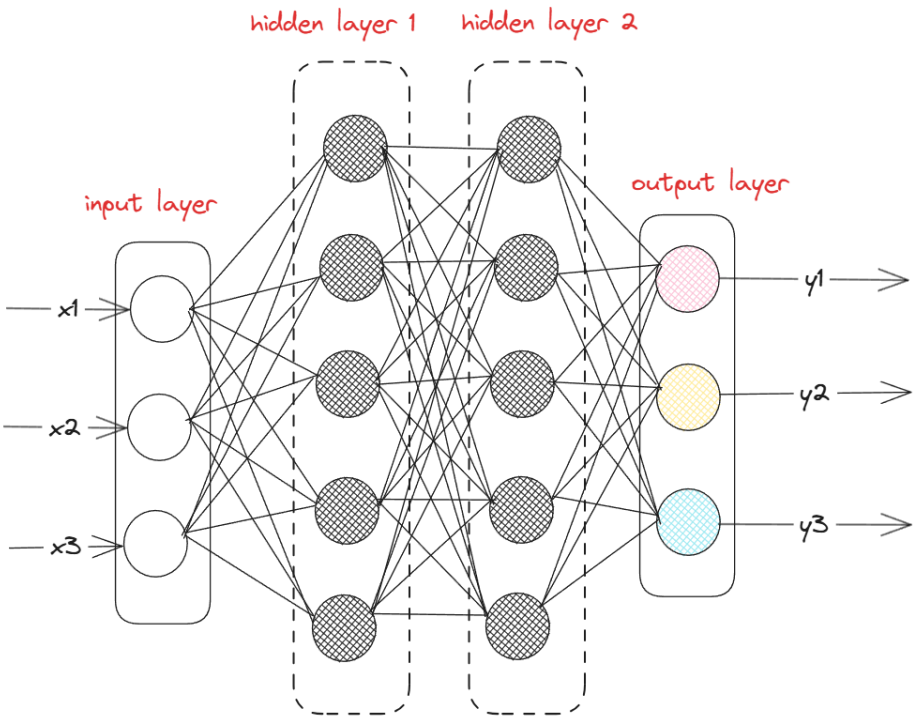
2.5.12.2 Backpropagation Through Time (BPTT)

Η εκπαίδευση RNN περιλαμβάνει την ελαχιστοποίηση μιας συνάρτησης απώλειας κατά τη διάρκεια της ακολουθίας. Μια κοινή προσέγγιση είναι η χρήση Backpropagation Through Time (BPTT), μια επέκταση της backpropagation για διαδοχικά δεδομένα. Κατά τη διάρκεια του BPTT, το δίκτυο ξετυλίγεται με την πάροδο του χρόνου, αντιμετωπίζοντας κάθε χρονικό βήμα ως επίπεδο σε ένα δίκτυο σε βάθος και, στη συνέχεια, υπολογίζει τις διαβαθμίσεις για κάθε παράμετρο με βάση το συσσωρευμένο σφάλμα σε όλη την ακολουθία. Η παράγωγος της απώλειας σε σχέση με το βάρος w_{ij} στο χρονικό βήμα t είναι:

$$\frac{\partial L}{\partial w_{ij}} = \sum_{k_1,\cdots,k_t} \frac{\partial L}{\partial y_{t,k_1}} \frac{\partial y_{t,k_1}}{\partial h_{t,k_2}} \frac{\partial h_{t,k_2}}{\partial h_{t-1,k_3}} \cdots \frac{\partial h_{2,k_t}}{\partial h_{1,i}} \frac{\partial h_{1,i}}{\partial w_{ij}} \tag{2.23}$$

Αυτή η αναδρομική εξάρτηση προκαλεί προβλήματα με τις παραγώγους (Vanishing/Exploding gradients), γεγονός που περιπλέκει την εκπαίδευση για μεγάλες ακολουθίες. Τεχνικές όπως το ντεγκραντέ και οι εναλλακτικές αρχιτεκτονικές όπως Long short-term memory (LSTM) και Gated recurrent unit (GRU) εισήχθησαν για την αντιμετώπιση αυτών των ζητημάτων ενσωματώνοντας μηχανισμούς που επιτρέπουν στο δίκτυο να διατηρεί ή να ξεχνά πληροφορίες επιλεκτικά [28].

2.5.13 Dense Layer



Σχήμα 2.1: Παράδειγμα MLP με 2 κρυφά πυκνά στρώματα

Ένα πλήρως συνδεδεμένο στρώμα, που αναφέρεται επίσης ως πυκνό στρώμα, είναι ένα θεμελιώδες δομικό στοιχείο στα τεχνητά νευρωνικά δίκτυα. Σε ένα πλήρως συνδεδεμένο στρώμα, κάθε νευρώνας στο στρώμα συνδέεται με κάθε νευρώνα στο προηγούμενο στρώμα. Το dense layer εκτελεί έναν γραμμικό μετασχηματισμό της εισόδου που μπορεί να ακολουθείται από μια μη γραμμική συνάρτηση ενεργοποίησης, επιτρέποντας στο μοντέλο να συλλαμβάνει πολύπλοκα μοτίβα στα δεδομένα. Μαθηματικά, η λειτουργία ενός πυκνού στρώματος μπορεί να οριστεί ως εξής:

$$\mathbf{y} = f(\mathbf{w}\mathbf{x} + \mathbf{b}) \tag{2.24}$$

όπου: $\mathbf{x} \in \mathbb{R}^n$ είναι το διάνυσμα εισόδου, $\mathbf{w} \in \mathbb{R}^{m \times n}$ είναι ο πίνακας βάρους, $\mathbf{b} \in \mathbb{R}^m$ είναι το διάνυσμα πόλωσης, $f(\cdot)$ είναι μια συνάρτηση ενεργοποίησης η οποία μπορεί να είναι και ταυτότητα $f(x) = x$,

και $\mathbf{y} \in \mathbb{R}^m$ είναι το διάνυσμα εξόδου. Τα εκπαιδεύσιμα βάρη $\mathbf{w}$ και $\mathbf{b}$, βελτιστοποιούνται κατά τη διάρκεια της εκπαιδευτικής διαδικασίας. Ένα multi layer perceptron (MLP) είναι ένας τύπος τεχνητού νευρωνικού δικτύου που αποτελείται από πολλαπλά στρώματα πλήρως συνδεδεμένων νευρώνων. Τα MLP περιλαμβάνουν συνήθως ένα επίπεδο εισόδου, ένα ή περισσότερα κρυφά επίπεδα και ένα στρώμα εξόδου. Κάθε κρυφό στρώμα και το επίπεδο εξόδου αποτελούνται από dense layers, ακολουθούμενα από μη γραμμικές συναρτήσεις ενεργοποίησης. Μαθηματικά, ένα MLP με επίπεδα L μπορεί να περιγραφεί ως μια ένθετη σύνθεση συνάρτησης:

$$\mathbf{y} = f_L(\mathbf{w}_L f_{L-1}(\mathbf{w}_{L-1} \dots f_1(\mathbf{w}_1\mathbf{x} + \mathbf{b}_1) \cdots + \mathbf{b}_{L-1}) + \mathbf{b}_L) \tag{2.25}$$

όπου $f_i(\cdot)$ αντιπροσωπεύει τη συνάρτηση ενεργοποίησης που εφαρμόζεται στο επίπεδο i, $\mathbf{w}_i$ είναι ο πίνακας βάρους και $\mathbf{b}_i$ είναι το διάνυσμα πόλωσης στο στρώμα i. Το βάθος του δικτύου (δηλαδή ο αριθμός των κρυφών επιπέδων) επιτρέπει στο MLP να μάθει ιεραρχικές αναπαραστάσεις των δεδομένων εισόδου.

2.5.14 Convolutional Neural Networks

2.5.14.1 Εισαγωγή

Τα συνελικτικά νευρωνικά δίκτυα (CNN) είναι ένας τύπος βαθιών νευρωνικών δικτύων που έχουν σχεδιαστεί ειδικά για την επεξεργασία δεδομένων με τοπολογία τύπου πλέγματος, όπως εικόνες. Τα CNN αποτελούνται από πολλαπλά επίπεδα, συμπεριλαμβανομένων συνελικτικών στρωμάτων, στρωμάτων υποδειγματοληψίας και πλήρως συνδεδεμένων στρωμάτων, τα οποία συνεργάζονται για να μάθουν ιεραρχικά χαρακτηριστικά από δεδομένα εισόδου και εμπνέονται από τους μηχανισμούς οπτικής επεξεργασίας του ανθρώπινου εγκεφάλου, συγκεκριμένα την ιεραρχική οργάνωση των νευρώνων στον οπτικό φλοιό. Στα τέλη της δεκαετίας του 1990, ο Yann LeCun και οι συνεργάτες του ανέπτυξαν τη σύγχρονη αρχιτεκτονική των CNN, συγκεκριμένα με τη δημιουργία του μοντέλου LeNet-5, το οποίο απέδειξε την αποτελεσματικότητα των συνελικτικών στρωμάτων στην αναγνώριση ψηφίων [9]. Η αρχιτεκτονική του LeNet-5, η οποία περιελάμβανε συνελικτικά στρώματα, στρώματα υποδειγματοληψίας και πλήρως συνδεδεμένα στρώματα, έθεσε το έδαφος για την ευρεία υιοθέτηση των CNN σε διάφορα προβλήματα τεχνητής νοημοσύνης και όρασης υπολογιστή. Όταν οι Alex Krizhevsky, Ilya Sutskever και Geoffrey Hinton παρουσίασαν το μοντέλο AlexNet το 2012, ήταν ένα σημείο καμπής για τα CNN καθώς βελτίωσε σημαντικά την απόδοση στο ImageNet Large Scale Visual Recognition Challenge [13]. Αυτή η επιτυχία οδήγησε σε μια νέα εποχή της βαθιάς μάθησης, που οδηγεί στην ανάπτυξη ολοένα και πιο εξελιγμένων αρχιτεκτονικών CNN, όπως το VGGNet, το GoogLeNet και το ResNet [30] [26].

2.5.14.2 Convolutional Layer

Το θεμελιώδες συστατικό ενός CNN είναι το συνελικτικό στρώμα. Αυτό το στρώμα δημιουργεί έναν χάρτη χαρακτηριστικών με την συνέλιξη των δεδομένων εισόδου και ενός συνόλου εκπαιδεύσιμων φίλτρων, συχνά γνωστά ως πυρήνες. Το δίκτυο μπορεί να εξάγει χωρικά χαρακτηριστικά όπως άκρες, γωνίες και υφές με τη βοήθεια της συνέλιξης. Κάθε φίλτρο εφαρμόζεται σε ολόκληρη την είσοδο, αλλά συγκεντρώνεται σε ένα στενό δεκτικό πεδίο. Τα συνελικτικά επίπεδα μπορεί να διαφέρουν ανάλογα με διαμορφώσεις όπως:

- Γέμισμα: Αναφέρεται στην προσθήκη μηδενικών γύρω από τον πίνακα εισόδου για τον έλεγχο του μεγέθους της εξόδου. Το Zero-padding διατηρεί τις χωρικές διαστάσεις της εισόδου μετά την εφαρμογή της λειτουργίας συνέλιξης, διασφαλίζοντας ότι τα χαρακτηριστικά κοντά στις άκρες δεν θα χαθούν. Το μέγεθος του γεμίσματος επηρεάζει το σχήμα εξόδου και βοηθά στη διατήρηση περισσότερων πληροφοριών, ειδικά σε βαθύτερα δίκτυα.

- Διασκελισμοί: Σε τυπικές συνελίξεις, το φίλτρο κινείται ένα pixel κάθε φορά (βήμα = 1). Ωστόσο, με τους διασκελισμούς, το φίλτρο μετατοπίζεται κατά μεγαλύτερο μέγεθος βήματος (βήμα > 1), γεγονός που μειώνει τις χωρικές διαστάσεις του χάρτη χαρακτηριστικών εξόδου. Αυτή η τεχνική βοηθά στη μείωση του υπολογιστικού κόστους και χρησιμοποιείται συχνά ως εναλλακτική λύση για τη συγκέντρωση επιπέδων για τη μείωση του δείγματος χαρτών χαρακτηριστικών.

- Διαστολές: Γνωστές και ως atrous convolutions, οι διεσταλμένες συνελίξεις εισάγουν κενά μεταξύ των στοιχείων του φίλτρου. Ένας παράγοντας διαστολής καθορίζει το μέγεθος αυτών των κενών, επιτρέποντας στη λειτουργία συνέλιξης να συλλάβει ένα μεγαλύτερο δεκτικό πεδίο χωρίς να αυξήσει τον αριθμό των παραμέτρων ή να χάσει την ανάλυση. Αυτό είναι ιδιαίτερα ωφέλιμο για προβλήματα που απαιτούν context aggregation, όπως η σημασιολογική κατάτμηση.

- Ομάδες: Σε ομαδοποιημένες συνελίξεις, τα κανάλια εισόδου χωρίζονται σε ομάδες και κάθε ομάδα συνελίσσεται χωριστά με το δικό της σύνολο φίλτρων. Τα αποτελέσματα συνενώνονται για να σχηματίσουν την τελική έξοδο. Οι ομαδοποιημένες συνελίξεις χρησιμοποιούνται σε αρχιτεκτονικές όπως το ResNeXt και αποτελούν επίσης βασικό συστατικό στην αποτελεσματική επεξεργασία μεγάλων μοντέλων [31]. Τέλος συμβάλλουν στη μείωση του υπολογισμού και επιτρέπουν την παράλληλη επεξεργασία.

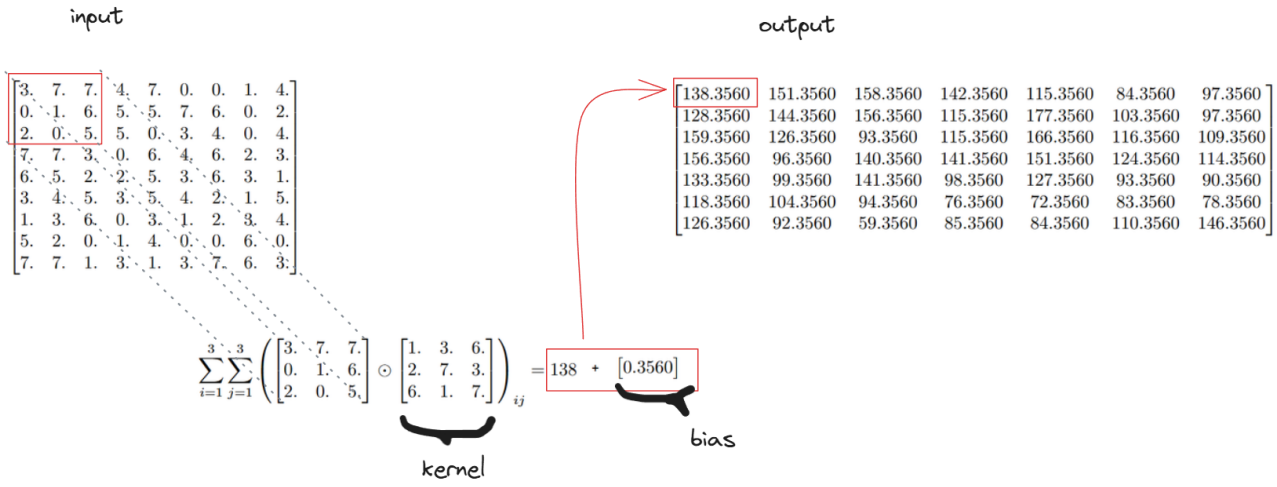
Βάζοντας τα πάντα μαζί, το στρώμα συνέλιξης με αριθμό φίλτρων μέγεθος φίλτρου (k_x, k_y), διαστολές (d_x, d_y), διασκελισμούς (s_x, s_y) και μέγεθος γεμίσματος (p_x, p_y). Δοθέντος μια παρτίδα εικόνων $\mathcal{I}$ με σχήμα (N, C, H, W), όπου N είναι το μέγεθος παρτίδας, C είναι ο αριθμός των καναλιών, H είναι το ύψος των επιπέδων εισόδου σε pixel και το W είναι το πλάτος σε pixel, καθώς ένα φίλτρο $\mathcal{K}$ σχήματος $(filters, \frac{C}{groups}, k_x, k_y)$ με $filters$ να είναι ο επιθυμητός αριθμός καναλιών που θα έχει η εικόνα εξόδου, και ένα προαιρετικό διάνυσμα πόλωσης $\mathcal{B} \in \mathbb{R}^{filters}$. Η παραγόμενη έξοδος $\mathcal{Y}$ θα είναι σχήματος $(N, filters, Ox, Oy)$ όπου:

$$O_x = \frac{H + 2p_x - d_x \times (k_x - 1) + 1}{s_x} + 1 \tag{2.26}$$

$$O_y = \frac{W + 2p_y - d_y \times (k_y - 1) + 1}{s_y} + 1 \tag{2.27}$$

Η πράξη της συνελίξης, για τις συντεταγμένες του pixel της εξόδου (i, j) και τον d-οστό χάρτη χαρακτηριστικών μπορεί να εκφραστεί μαθηματικά ως::

$$conv(\mathcal{I}, \mathcal{K})^d_{i,j} = \mathcal{B}_d + \sum_{h=0}^{C-1} \sum_{m=0}^{k_x-1} \sum_{n=0}^{k_y-1} \mathcal{K}^d_{c,m,n} \cdot \mathcal{I}^h_{s_x i + d_x m - p_x, s_y j + d_y n - p_y} \tag{2.28}$$



Σχήμα 2.2: Παράδειγμα συνελίξης σε εικόνα (9×9) με φίλτρο (3×3)

2.5.14.3 Transposed Convolutions

Οι ανεστραμένες συνελίξεις, που αναφέρονται επίσης ως "αποσυνελίξεις" (deconvolutions) είναι απαραίτητες σε εργασίες που απαιτούν υπερδειγματοληψία στη βαθιά μάθηση, όπως η δημιουργία εικόνων ή η τμηματοποίηση [32]. Σε αντίθεση με τις τυπικές συνελίξεις, οι οποίες μειώνουν τη χωρική ανάλυση, οι ανεστραμένες συνελίξεις αντιστρέφουν αυτή τη διαδικασία αυξάνοντας τις χωρικές διαστάσεις της εισόδου. Αυτό επιτυγχάνεται με την ολίσθηση ενός πυρήνα πάνω από την είσοδο και την επέκταση της εξόδου με εκπαιδεύσιμες παραμέτρους, παράγοντας αποτελεσματικά έναν μεγαλύτερο χάρτη χαρακτηριστικών. Μια βασική εφαρμογή των μετατιθέμενων συνελίξεων είναι σε παραγωγικά δίκτυα όπως τα GAN, όπου χρησιμοποιούνται για τη δημιουργία εικόνων υψηλής ανάλυσης από αναπαραστάσεις χαρακτηριστικών χαμηλών διαστάσεων [33]. Οι αποσυνέλιξεις έχουν όλες τις επιλογές διαμόρφωσης των τυπικών συνελίξεων με την πρόσθετη παράμετρο του output padding που είναι διάνυσμα ακεραίων που ελέγχει το μέγεθος που προστίθεται σε κάθε διάσταση του σχήματος εξόδου. Η έξοδος της λειτουργίας αποσυνέλιξης θα έχει μια ανάλυση (O'_x, O'_y) που υπολογίζεται από τον τύπο:

$$O_{x'} = (H-1) \times s_x - 2p_x + d_x \times (k_x - 1) + \text{output\_padding}_x + 1 \tag{2.29}$$

$$O_{y'} = (W-1) \times s_y - 2p_y + d_y \times (k_y - 1) + \text{output\_padding}_y + 1 \tag{2.30}$$

και η μαθηματική πράξη μπορεί να εκφραστεί ως εξής:

$$conv^T(\mathcal{I}, \mathcal{K})_{i,j} = \sum_{m=0}^{k_x-1} \sum_{n=0}^{k_y-1} \mathcal{K}_{m,n} \cdot \mathcal{I}_{i-m,j-n} \tag{2.31}$$

2.5.14.4 Depthwise Seperable Convolutions

Οι κατά βάθος διαχωρίσιμες συνελίξεις στοχεύουν στη βελτίωση της αποτελεσματικότητας των συνελικτικών νευρωνικών δικτύων (CNN), ιδιαίτερα σε περιβάλλοντα με περιορισμένους πόρους. Οι διαχωρίσιμες συνελίξεις ουσιαστικά αποσυνθέτουν μια τυπική συνέλιξη σε δύο μικρότερες λειτουργίες: μια συνέλιξη κατά βάθος που ακολουθείται από μια συνέλιξη κατά σημείο (1x1).

Οι συνέλιξεις κατά βάθος (Depthwise Convolutions) περιλαμβάνουν την εφαρμογή ενός μόνο συνελικτικού φίλτρου ανά κανάλι εισόδου, με αποτέλεσμα τον ίδιο αριθμό καναλιών εξόδου με τα κανάλια εισόδου, σε αντίθεση με μια τυπική συνέλιξη, η οποία εφαρμόζει φίλτρα σε όλα τα κανάλια ταυτόχρονα. Η συνέλιξη κατά σημείο (Pointwise Convolution) συνδυάζει αυτά τα κανάλια εξόδου, αναμειγνύοντας πληροφορίες μεταξύ τους και ενδεχομένως αυξάνοντας τον αριθμό των καναλιών εξόδου. Αυτή η αποσύνθεση μειώνει την υπολογιστική πολυπλοκότητα (δηλαδή τον αριθμό των πολλαπλασιασμών) και τον αριθμό των παραμέτρων, καθώς κάθε φίλτρο αλληλεπιδρά μόνο με ένα μόνο κανάλι χωρίς να επηρεάζει σημαντικά την απόδοση. Οι κατά βάθος διαχωρίσιμες συνελίξεις αποτελούν αναπόσπαστο κομμάτι σε ελαφριές αρχιτεκτονικές όπως το MobileNets και το Xception, επιτρέποντας την αποτελεσματική ανάπτυξη των CNN σε εφαρμογές και συστήματα πραγματικού χρόνου [34] [35].

Υποθέτοντας ένα στρώμα συνέλιξης κατά βάθος με έναν πυρήνα $\mathcal{K}_d$ και ένα διάνυσμα πόλωσης $\mathcal{B}_d$, καθώς και ένα στρώμα συνέλιξης κατά σημείο με έναν πυρήνα $\mathcal{K}_p$ και $\mathcal{B}_p$ η πόλωση αντίστοιχα, τότε για τις συντεταγμένες του pixel εξόδου (i, j) του χάρτη χαρακτηριστικών k η κατά βάθος διαχωρίσιμη συνέλιξη μπορεί να εκφραστεί μαθηματικά ως εξής:

$$\mathcal{Y}(k,i,j) = \sum_{c=0}^{C-1} \left(\sum_{m=0}^{k_x-1} \sum_{n=0}^{k_y-1} \mathcal{I}(c, i+m, j+n) \cdot \mathcal{K}_d(m,n,0,c) + \mathcal{B}_d(c) \right) \cdot \mathcal{K}_p(0,0,c,k) + \mathcal{B}_d(k) \tag{2.32}$$

2.5.14.5 Pooling Layer

Τα στρώματα υποδειγματοληψίας είναι ένα κρίσιμο συστατικό στα συνελικτικά νευρωνικά δίκτυα (CNN), που χρησιμοποιούνται κυρίως για τη μείωση των χωρικών διαστάσεων (ύψος και πλάτος) των χαρτών χαρακτηριστικών διατηρώντας παράλληλα σημαντικές πληροφορίες. Αυτή η μείωση διαστάσεων βοηθά στον έλεγχο του overfitting και μειώνει την υπολογιστική πολυπλοκότητα του μοντέλου. Τα στρώματα υποδειγματοληψίας χρησιμοποιούν συνήθως ένα παράθυρο σταθερού μεγέθους και ένα βήμα ή διασκελισμό που καθορίζει το μέγεθος βήματος του παραθύρου καθώς μετακινείται στον χάρτη χαρακτηριστικών εισόδου. Με την υποδειγματοληψία του χάρτη χαρακτηριστικών, τα στρώματα αυτά παρέχουν μεταφορική αμεταβλητότητα, που σημαίνει ότι μικρές μετατοπίσεις στα δεδομένα εισόδου δεν επηρεάζουν σημαντικά την έξοδο.

Οι δύο πιο συνηθισμένοι τύποι υποδειγματοληψίας είναι:

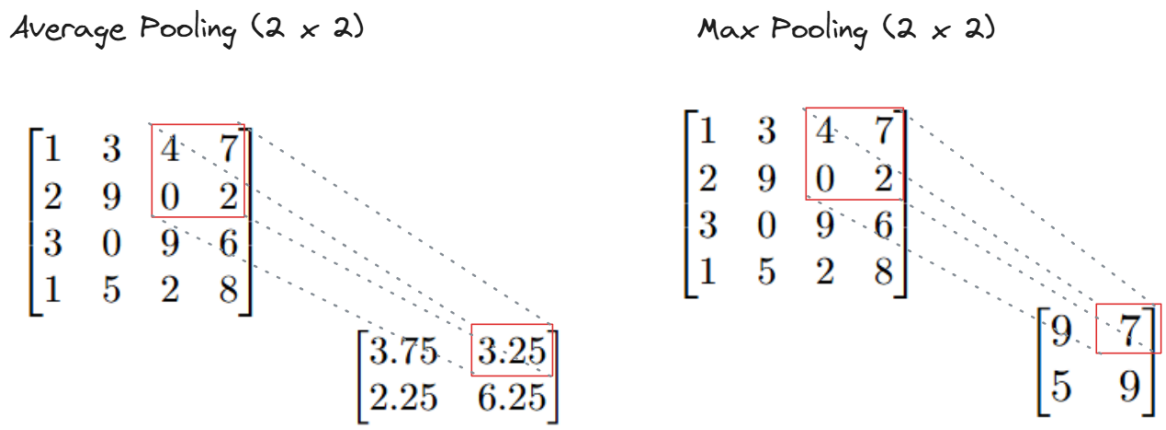
- Μέγιστης τιμής (max pooling): επιλέγεται η μέγιστη τιμή σε κάθε patch, διατηρώντας αποτελεσματικά τα πιο σημαντικά χαρακτηριστικά. Το Max pooling προτιμάται συχνά στην πράξη, καθώς τείνει να καταγράφει πιο ευκρινή χαρακτηριστικά που είναι κρίσιμα σε προβλήματα όπως η ανίχνευση αντικειμένων και η αναγνώριση εικόνας.Ο τύπος που εφαρμόζει το max pooling είναι:

$$\mathcal{Y}_{i,j} = \mathbf{max}(\mathcal{I}_{s_x i:s_x i+w_x, s_y j:s_y j+w_y}) \tag{2.33}$$

- Μέσης τιμής (average pooling): Υπολογίζει τον μέσο όρο των τιμών στο παράθυρο, προσφέροντας μια πιο ομαλή, πιο γενικευμένη αναπαράσταση χαρακτηριστικών [26]. Ο μαθηματικός ορισμός

έχει ως εξής:

$$\mathcal{Y}_{i,j} = \frac{1}{w_x \times w_y} \sum_{m=0}^{w_x-1} \sum_{n=0}^{w_y-1} \mathcal{I}_{s_x i+m, s_y j+n} \tag{2.34}$$



Σχήμα 2.3: Παράδειγμα υποδειγματοληψίας μέσης και μέγιστης τιμής

2.5.14.6 Global Pooling

Η ολική υποδειγματοληψία είναι μια τεχνική που χρησιμοποιείται για τη μείωση των χωρικών διαστάσεων των χαρτών χαρακτηριστικών σε μια ενιαία τιμή ανά χάρτη χαρακτηριστικών. Τα στρώματα ολικής υποδειγματοληψίας καθώς δεν έχουν εκπαιδεύσιμες παραμέτρους, μπορούν να χρησιμοποιηθούν στο τέλος ενός CNN ως εναλλακτική λύση που προσφέρει μείωση της υπολογιστικής πολυπλοκότητας και των απαιτήσεων μνήμης, αποφεύγοντας επίσης το overfitting του πλήρως συνδεδεμένου στρώματος του δικτύου του ταξινομητή [36]. Οι δύο πιο κοινές μορφές global pooling είναι και πάλι της μέγιστης τιμής και της μέσης τιμής και ο γενικός μαθηματικός τύπος είναι:

$$\mathcal{Y} = \mathcal{F}(\mathcal{I}_{:,:}) \tag{2.35}$$

Όπου για GlobalAveragePooling η συνάρτηση θα είναι:

$$\mathcal{F}(\mathcal{I}) = \frac{1}{H \times W} \sum_{i=0}^{H} \sum_{j=0}^{W} \mathcal{I}_{i,j} \tag{2.36}$$

Ενώ για GlobalMaxPooling η συνάρτηση θα είναι:

$$\mathcal{F}(\mathcal{I}) = \max_{1 \le i \le H, 1 \le J \le W} \mathcal{I}_{i,j} \tag{2.37}$$

2.5.15 Residual connections

Εισήχθη από τους He et al. [26], οι υπολειπόμενες συνδέσεις (residual connections ή skip connections), είναι ένα κρίσιμο αρχιτεκτονικό στοιχείο σε βαθιά νευρωνικά δίκτυα που έχουν σχεδιαστεί για την α-

ντιμετώπιση του προβλήματος υποβάθμισης που εμφανίζεται με την αύξηση του βάθους του δικτύου. Διευκολύνουν την εκπαίδευση δικτύων πολύ βαθιάς επιτρέποντας την αποτελεσματική ροή gradients μέσω του δικτύου. Έχουν σχεδιαστεί για να επιτρέπουν στην είσοδο σε ένα μπλοκ ή στρώμα να παρακάμπτει ένα ή περισσότερα ενδιάμεσα στρώματα και να προστίθεται απευθείας στην έξοδο αυτού του μπλοκ. Αυτό μπορεί να εκφραστεί μαθηματικά ως:

$$\mathbf{y} = \mathcal{F}(\mathbf{x}) + \mathbf{x} \tag{2.38}$$

όπου **x** αντιπροσωπεύει την είσοδο στο υπολειπόμενο μπλοκ, $\mathcal{F}(\cdot)$ υποδηλώνει τη συνάρτηση που εφαρμόζει το υπολειπόμενο μπλοκ (που συνήθως αποτελείται από ένα ή περισσότερα συνελικτικά στρώματα και μη γραμμικότητες) και **y** είναι η έξοδος του μπλοκ. Ο όρος **x** είναι η υπολειπόμενη σύνδεση που προστίθεται στην έξοδο του μπλόκ $\mathcal{F}$. Αυτή η τεχνική μπορεί επίσης να μετριάσει το πρόβλημα των εξασθενούσων διορθώσεων παρέχοντας μια άμεση διαδρομή για τη ροή των διαβαθμίσεων μέσω του δικτύου, διευκολύνοντας τη βελτιστοποίησή τους. Αυτό είναι ιδιαίτερα ωφέλιμο σε πολύ βαθιά δίκτυα όπου τα gradients μπορούν να μειωθούν ή να ενισχυθούν εκθετικά. Οι υπολειπόμενες συνδέσεις μπορούν να υλοποιηθούν με διάφορες μορφές, συμπεριλαμβανομένων αντιστοιχίσεων ταυτότητας (όπου η είσοδος προστίθεται απευθείας στην έξοδο) και αντιστοιχίσεων προβολής (όπου εφαρμόζεται γραμμικός μετασχηματισμός στην είσοδο πριν από την προσθήκη).

2.5.16 Normalization Layer

2.5.16.1 Εισαγωγή

Τα επίπεδα κανονικοποίησης παίζουν κρίσιμο ρόλο στην εκπαίδευση των βαθιών νευρωνικών δικτύων σταθεροποιώντας και επιταχύνοντας τη διαδικασία μάθησης. Λειτουργούν προσαρμόζοντας και κλιμακώνοντας τις ενδιάμεσες ενεργοποιήσεις, διασφαλίζοντας ότι τα δεδομένα που τροφοδοτούνται σε κάθε επίπεδο διατηρούν μια πιο σταθερή κατανομή. Αυτό όχι μόνο μειώνει τη μετατόπιση της εσωτερικής συνδιακύμανσης, όπου οι αλλαγές στις παραμέτρους του δικτύου προκαλούν διακυμάνσεις στην κατανομή των εισόδων επιπέδου, αλλά επίσης βελτιώνει τους ρυθμούς σύγκλισης και επιτρέπει τη χρήση υψηλότερων ρυθμών εκμάθησης. Η κανονικοποιημένη έξοδος $\hat{x}$ υπολογίζεται συνήθως ως:

$$\hat{x} = \frac{x - \mu_g}{\sqrt{\sigma_g^2 + \epsilon}} \tag{2.39}$$

όπου μ_g και σ_g^2 είναι ο μέσος όρος και η διακύμανση, ϵ είναι μια μικρή σταθερά για την αριθμητική σταθερότητα και το x αντιπροσωπεύει την είσοδο. Προαιρετικές εκπαιδεύσιμες παράμετροι γ και β που συνήθως αρχικοποιούνται σε άσσους και το μηδενικά, εφαρμόζονται για την κλιμάκωση και τη μετατόπιση της κανονικοποιημένης εξόδου:

$$y = \gamma\hat{x} + \beta. \tag{2.40}$$

2.5.16.2 Batch Normalization

Η κανονικοποίηση παρτίδας κανονικοποιεί τις εισόδους προσαρμόζοντας και κλιμακώνοντας τις ενερ-

γοποιήσεις σε κάθε μίνι παρτίδα [37]. Για την κανονικοποίηση παρτίδας, ο μέσος όρος μ και η διακύμανση σ^2 υπολογίζονται σε ολόκληρη τη μίνι παρτίδα και τις χωρικές διαστάσεις (δηλαδή, πάνω από $N \times H \times W$) για κάθε κανάλι χαρακτηριστικών c.

$$\mu_c = \frac{1}{N \times H \times W} \sum_{n=1}^{N} \sum_{h=1}^{H} \sum_{w=1}^{W} x_{nchw} \tag{2.41}$$

$$\sigma_c^2 = \frac{1}{N \times H \times W} \sum_{n=1}^{N} \sum_{h=1}^{H} \sum_{w=1}^{W} (x_{nchw} - \mu_c)^2 \tag{2.42}$$

Διατηρώντας μια πιο σταθερή κατανομή των ενεργοποιήσεων, το BN επιτρέπει τη χρήση υψηλότερων ρυθμών μάθησης διευκολύνοντας ταχύτερη σύγκλιση επιτρέποντας στο δίκτυο να εκπαιδεύεται πιο αποτελεσματικά, αλλά μπορεί επίσης να μειώσει την ανάγκη για τεχνικές όπως το dropout. Επιπλέον, η κανονικοποίηση παρτίδας καθιστά το δίκτυο λιγότερο ευαίσθητο στην αρχηκοποίηση και μειώνει την εξάρτηση από προσεκτικό συντονισμό υπερπαραμέτρων. Η κανονικοποίηση παρτίδας χρησιμοποιείται ευρέως σε διάφορες αρχιτεκτονικές, συμπεριλαμβανομένων των συνελικτικών και πλήρως συνδεδεμένων δικτύων, και έχει γίνει ένα τυπικό στοιχείο στα σύγχρονα μοντέλα βαθιάς μάθησης.

2.5.16.3 Layer Normalization

Η κανονικοποίηση στρώματος κανονικοποιεί τις ενεργοποιήσεις σε όλα τα χαρακτηριστικά για κάθε σημείο δεδομένων ξεχωριστά, παρά σε μια μίνι-παρτίδα, γεγονός που το καθιστά ανεξάρτητο από το μέγεθος της παρτίδας [38]. Για την κανονικοποίηση στρώματος, ο μέσος όρος μ και η διακύμανση σ^2 υπολογίζονται σε ολόκληρο τον χάρτη χαρακτηριστικών (δηλαδή, κατά $C \times H \times W$) για κάθε μεμονωμένο δείγμα εκπαίδευσης n.

$$\mu_n = \frac{1}{C \times H \times W} \sum_{c=1}^{C} \sum_{h=1}^{H} \sum_{w=1}^{W} x_{nchw} \tag{2.43}$$

$$\sigma_n^2 = \frac{1}{C \times H \times W} \sum_{c=1}^{C} \sum_{h=1}^{H} \sum_{w=1}^{W} (x_{nchw} - \mu_n)^2 \tag{2.44}$$

Σε αντίθεση με την κανονικοποίηση παρτίδας, η κανονικοποίηση στρώματος δεν βασίζεται σε στατιστικά στοιχεία παρτίδας, καθιστώντας την ιδιαίτερα χρήσιμη για μοντέλα που εκπαιδεύονται με πολύ μικρές μίνι παρτίδες ή ακόμη και μεμονωμένα δείγματα. Ως αποτέλεσμα, η κανονικοποίηση στρώματος χρησιμοποιείται συνήθως σε RNNs και μετασχηματιστές, όπου λαμβάνει χώρα διαδοχική επεξεργασία δεδομένων και η διατήρηση σταθερής κανονικοποίησης μεταξύ των ακολουθιών είναι κρίσιμη. Η κανονικοποίηση στρώματος βελτιώνει τη σταθερότητα της διαδικασίας εκπαίδευσης και αποτρέπει μεγάλες διακυμάνσεις στα gradients, ιδιαίτερα σε βαθιά δίκτυα. Η εφαρμογή του έχει αποδειχθεί ότι ενισχύει τον ρυθμό σύγκλισης και την απόδοση των μοντέλων σε διάφορα συστήματα επεξεργασίας φυσικής γλώσσας και μάθησης με ενίσχυση.

2.5.16.4 Group normalization

Η κανονικοποίηση ομάδας (GN) είναι μια τεχνική κανονικοποίησης που γενικεύει την κανονικοποίηση παρτίδας και στρώματος διαιρώντας τα κανάλια ενός στρώματος σε μικρότερες ομάδες και κανονικοποιώντας μέσα σε κάθε ομάδα [39]. Είναι ιδιαίτερα αποτελεσματική σε περιπτώσεις όπου το μέγεθος της παρτίδας είναι μικρό, αντιμετωπίζοντας τους περιορισμούς της κανονικοποίησης παρτίδας, η οποία βασίζεται σε στατιστικά mini-batch. Σε αντίθεση με την κανονικοποίηση παρτίδας, η οποία υπολογίζει τη μέση τιμή και τη διακύμανση σε ολόκληρη τη μίνι παρτίδα, η κανονικοποίηση ομάδας χωρίζει τα κανάλια χαρακτηριστικών σε ομάδες και υπολογίζει τη μέση τιμή και τη διακύμανση σε κάθε ομάδα. Αυτή η διαδικασία εκτελείται ανεξάρτητα για κάθε δείγμα εκπαίδευσης, καθιστώντας το ανθεκτικό σε διάφορα μεγέθη παρτίδων. Για την κανονικοποίηση ομάδας, ο μέσος όρος μ και η διακύμανση σ^2 υπολογίζονται σε κάθε ομάδα (δηλαδή, πάνω από $\frac{C}{G} \times H \times W$) για κάθε ομάδα καναλιών σε κάθε δείγμα εκπαίδευσης n. Έστω G ο αριθμός των ομάδων και $C_G = \frac{C}{G}$ ο αριθμός των καναλιών ανά ομάδα. Ο μέσος όρος και η διακύμανση υπολογίζονται ως εξής:

$$\mu_{ng} = \frac{1}{C_G \times H \times W} \sum_{c \in g} \sum_{h=1}^{H} \sum_{w=1}^{W} x_{nchw} \tag{2.45}$$

$$\sigma_{ng}^2 = \frac{1}{C_G \times H \times W} \sum_{c \in g} \sum_{h=1}^{H} \sum_{w=1}^{W} (x_{nchw} - \mu_{ng})^2 \tag{2.46}$$

Η ευελιξία της κανονικοποίηση ομάδας πηγάζει από την ικανότητά της να παρεμβάλλεται μεταξύ κανονικοποίησης στρώματος και κανονικοποίησης στιγμιότυπου, ανάλογα με τον αριθμό των ομάδων. Για παράδειγμα, ο καθορισμός του αριθμού των ομάδων ίσος με τον αριθμό των καναλιών χαρακτηριστικών έχει ως αποτέλεσμα την κανονικοποίηση του στιγμιότυπου, ενώ η ρύθμιση του σε 1 προσεγγίζει την κανονικοποίηση του στρώματος. Συνήθως, το GN διαιρεί τα κανάλια σε έναν σταθερό αριθμό ομάδων (π.χ. 32), διασφαλίζοντας αποτελεσματική κανονικοποίηση σε διάφορες αρχιτεκτονικές δικτύου και τύπους δεδομένων. Το GN έχει επιδείξει ανώτερη απόδοση σε συστήματα όρασης υπολογιστή όπου χρησιμοποιούνται συνήθως μικρά μεγέθη παρτίδων, όπως η ανίχνευση αντικειμένων και η κατάτμηση (image segmentation) . Έχει γίνει δημοφιλής επιλογή σε περιπτώσεις όπου το μέγεθος της μίνι παρτίδας περιορίζεται από περιορισμούς μνήμης ή όπου τα στατιστικά στοιχεία παρτίδας είναι αναξιόπιστα.

2.5.16.5 Instance normalization

Η κανονικοποίηση στιγμιότυπου (IN) είναι μια τεχνική κανονικοποίησης που κανονικοποιεί κάθε χάρτη χαρακτηριστικών ενός δείγματος ανεξάρτητα, εξαλείφοντας την εξάρτηση κατά παρτίδες και διασφαλίζοντας ότι το δίκτυο είναι ανθεκτικό στις αλλαγές στην αντίθεση και το στυλ [40]. Για την κανονικοποίηση του στιγμιότυπου, ο μέσος όρος μ και η διακύμανση σ^2 υπολογίζονται στις χωρικές διαστάσεις (δηλαδή, πάνω από $H \times W$) για κάθε κανάλι c από κάθε δείγμα εκπαίδευσης n.

$$\mu_{nc} = \frac{1}{H \times W} \sum_{h=1}^{H} \sum_{w=1}^{W} x_{nchw} \tag{2.47}$$

$$\sigma_{nc}^2 = \frac{1}{H \times W} \sum_{h=1}^{H} \sum_{w=1}^{W} (x_{nchw} - \mu_{nc})^2 \tag{2.48}$$

Είναι ιδιαίτερα χρήσιμη σε εργασίες όπως η μεταφορά στυλ, επειδή καταργεί τις παραλλαγές αντίθεσης και φωτισμού για συγκεκριμένες περιπτώσεις στις εικόνες εισόδου. Επιτρέπει στο μοντέλο να εστιάζει σε χαρακτηριστικά ανεξάρτητα από το περιεχόμενο και το στυλ, κάτι που είναι κρίσιμο για εργασίες όπου επιθυμούνται στιλιστικές αλλαγές στις εικόνες εισόδου. Το IN διαφέρει από την κανονικοποίηση παρτίδας επειδή είναι εντελώς ανεξάρτητο από το μέγεθος παρτίδας, καθώς υπολογίζει τα στατιστικά στοιχεία κανονικοποίησης με βάση αποκλειστικά τον χάρτη χαρακτηριστικών κάθε δείγματος.

2.5.16.6 Root Mean Square normalization

Η κανονικοποίηση μέσης τετραγωνικής ρίζας (RMSNorm) σε αντίθεση με την κανονικοποίηση παρτίδας ή την κανονικοποίηση στρώματος, κανονικοποιεί τις ενεργοποιήσεις χρησιμοποιώντας τη ρίζα του μέσου τετραγώνου (RMS) των χαρακτηριστικών εισόδου χωρίς να τις κεντράρει και δεν βασίζεται σε στατιστικά παρτίδας [41]. Για τον μαθηματικό ορισμό, με δεδομένη την είσοδο $x \in \mathbb{R}^n$, μια μικρή σταθερά ϵ για αριθμητική σταθερότητα και ένα εκπαιδεύσιμο διάνυσμα κλιμάκωσης $\gamma \in \mathbb{R}^n$, η κανονικοποιημένη έξοδος $y \in \mathbb{R}^n$ χρησιμοποιώντας τη μέθοδο RMSNorm υπολογίζεται ως εξής:

$$\text{RMS}(x) = \sqrt{\epsilon + \frac{1}{n} \sum_{i=1}^{n} x_i^2} \tag{2.49}$$

$$\hat{x}_i = \frac{x_i}{\text{RMS}(x)} \tag{2.50}$$

$$y_i = \hat{x}_i \odot \gamma_i \tag{2.51}$$

Το RMSNorm έχει προταθεί ως εναλλακτική λύση στο Layer Norm για δίκτυα μετασχηματιστών με εξαιρετικά αποτελέσματα. Με το να μην βασίζεστε σε στατιστικά παρτίδας διασφαλίζετε ότι η απόδοση θα παραμείνει όταν τα μεγέθη παρτίδων είναι μικρά και σε συνδυασμό με την έλλειψη μετατόπισης, το υπολογιστικό κόστος μειώνεται δραστικά καθιστώντας το μοντέλο φθηνότερο και ταχύτερο. Έχει εφαρμοστεί σε αρχιτεκτονικές μοντέλων όπως LLaMA και DeepSeek για τη βελτίωση της ταχύτητας σύγκλισης, της σταθερότητας και της γενίκευσης [42] [43].

2.5.17 Attention

Ο μηχανισμός προσοχής έχει γίνει μια κεντρική έννοια στη βαθιά μάθηση, ιδιαίτερα στην επεξεργασία φυσικής γλώσσας (NLP) και στην όραση υπολογιστή. Οι μηχανισμοί προσοχής επιτρέπουν στα μοντέλα να εστιάζουν επιλεκτικά σε συγκεκριμένα μέρη μιας ακολουθίας εισόδου, αντιμετωπίζοντας τους περιορισμούς των παραδοσιακών μοντέλων ακολουθιών, όπως τα αναδρομικά νευρωνικά δίκτυα (RNNs) που συχνά δυσκολεύονται να καταγράψουν εξαρτήσεις μεγάλης εμβέλειας. Με τη δυναμική σημαντικότητας της σημαντικότητας των διαφορετικών στοιχείων εισόδου, η προσοχή επιτρέπει την πιο ευέλικτη και αποτελεσματική μοντελοποίηση [44]. Μαθηματικά, ο μηχανισμός προσοχής μπορεί να οριστεί ως εξής. Δεδομένου ενός διανύσματος ερωτήματος $q \in \mathbb{R}^{d_k}$, ενός συνόλου διανυσμάτων

κλειδιών $K = \{k_0, k_1, \ldots, k_n\}$ με $k_i \in \mathbb{R}^{d_k}$ και διανυσμάτων τιμών που συνδυάζονται με κάθε κλειδί $V = \{v_0, v_1, \ldots, v_n\}$ με $v_i \in \mathbb{R}^{d_v}$, η βαθμολογία προσοχής για κάθε κλειδί είναι η ομοιότητα μεταξύ του ερωτήματος και του κλειδιού που συνήθως υπολογίζεται μέσω ενός εσωτερικού γινόμενου. Στη συνέχεια ακολουθείται από ένα στρώμα softmax που διασφαλίζει ότι οι βαθμολογίες της προσοχής αθροίζονται στο 1, καθιστώντας τις ερμηνεύσιμες ως πιθανότητες [45].

$$softmax(x)_i = \frac{\exp(x_i)}{\sum_{j=0}^{n} \exp(x_j)} \tag{2.52}$$

Ωστόσο, πριν από αυτό, οι βαθμολογίες κλιμακώνονται με την αντίστροφη τετραγωνική ρίζα της διάστασης των κλειδιών $\frac{1}{\sqrt{d_k}}$, αυτή είναι μια μορφή κανονικοποίησης για να αποτραπεί η είσοδος πολύ μεγάλων τιμών στο στρώμα softmax που θα προκαλούσε το πρόβλημα των εξασθενούσων διορθώσεων. Η έξοδος προσοχής είναι τότε το σταθμισμένο άθροισμα των τιμών, με τα βάρη που δίνονται από τις κανονικοποιημένες βαθμολογίες. Αυτή η διαδικασία μπορεί να εκφραστεί μαθηματικά ως εξής:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.53}$$

όπου $K \in \mathbb{R}^{n \times d_k}$ και $V \in \mathbb{R}^{n \times d_v}$ είναι πίνακες n κλειδιών και τιμών, αντίστοιχα, και $Q \in \mathbb{R}^{m \times d_k}$ είναι ένα σύνολο m ερωτημάτων. Τέλος, η έξοδος του επιπέδου προσοχής θα είναι της μορφής $y \in \mathbb{R}^{m \times d_v}$. Ο μηχανισμός προσοχής διατίθεται σε διάφορες παραλλαγές, όπως:

- Αυτοπροσοχή (Self Attention) : Δεδομένου του πίνακα εισόδου $X \in \mathbb{R}^{n \times d}$, όπου n είναι ο αριθμός των στοιχείων και d είναι η διάσταση κάθε στοιχείου, Ο μηχανισμός αυτοπροσοχής περιλαμβάνει τρία βασικά στοιχεία: το ερώτημα $Q = XW_Q$, το κλειδί $K = XW_K$ και την τιμή $V = XW_V$, όπου $W_Q \in \mathbb{R}^{d \times d_k}$, $W_K \in \mathbb{R}^{d \times d_k}$, και $W_V \in \mathbb{R}^{d \times d_v}$ είναι εκπαιδεύσιμοι πίνακες βάρους. Οι βαθμοί προσοχής υπολογίζονται με τον τύπου του μηχανισμόυ προσοχής (**??**).

- Διασταυρούμενη Προσοχή (Cross Attention) : επιτρέπει στα στοιχεία μιας ακολουθίας να παρακολουθούν μια άλλη ακολουθία. Δεδομένων δύο ακολουθιών, το ερώτημα προέρχεται από μια ακολουθία (ας πούμε X_1) ενώ το κλειδί και η τιμή προέρχονται από μια άλλη ακολουθία (ας πούμε X_2).

  $$\text{CrossAttention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.54}$$

  Εδώ, $Q = X_1W_Q$, $K = X_2W_K$ και $V = X_2W_V$. Αυτό επιτρέπει στις πληροφορίες από την ακολουθία X_1 να επηρεάζονται από το περιβάλλον από την ακολουθία X_2.

- Προσοχή πολλαπλών κεφαλών (MultiHead Attention) : επιτρέπει σε πολλαπλούς μηχανισμούς προσοχής (κεφαλές) να λειτουργούν παράλληλα, ο καθένας με το δικό του σύνολο εκπαιδεύσιμων πινάκων βάρους. Για κάθε κεφαλή $i \in (0, h]$ όπου h είναι ο αριθμός των κεφαλών, η προσοχή υπολογίζεται ως:

  $$\text{head}_i = \text{Attention}(Q_i, K_i, V_i) \tag{2.55}$$

όπου $Q_i = XW_i^Q$, $K_i = XW_i^K$, και $V_i = XW_i^V$, με $d_v = d_k = \frac{d}{h}$. Μετά τον υπολογισμό της προσοχής για όλες τις κεφαλές, τα αποτελέσματα συνενώνονται και μετασχηματίζονται γραμμικά:

$$\text{Multihead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W_O \tag{2.56}$$

όπου $W_O \in \mathbb{R}^{hd_v \times d}$ είναι ένας εκπαιδεύσιμος πίνακας προβολής. Αυτό επιτρέπει στο μοντέλο να παρακολουθεί διαφορετικές πτυχές της ακολουθίας εισόδου.

- Προσοχή με μάσκα (Masked Attention) : εισάγει περιορισμούς σε αυτή τη διαδικασία εμποδίζοντας ορισμένα tokens να παρακολουθούν άλλα. Αυτό μπορεί να επιτευχθεί με την εισαγωγή ενός πίνακα μάσκας $M \in \mathbb{R}^{n \times n}$ και μπορεί να εκφραστεί ως εξής:

$$\text{MaskedAttention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}} + M\right)V \tag{2.57}$$

Εδώ, $M_{ij} = -\infty$ εάν το token i δεν πρέπει να παρακολουθεί το token j (δηλαδή, $j > i$ για causal masking), διασφαλίζοντας ότι τα μελλοντικά tokens αγνοούνται αφού οι βαθμοί προσοχής τους θα είναι μηδέν μετά την εφαρμογή του softmax. Η μάσκα είναι συνήθως άνω τριγωνική για αυτοπαλινδρομικές εργασίες (autoregressive tasks) όπου το μοντέλο δεν πρέπει να παρακολουθεί μελλοντικά tokens. Αυτός ο τύπος μάσκας διασφαλίζει ότι κάθε token έχει πρόσβαση μόνο στον εαυτό του και σε όλα τα προηγούμενα tokens της σειράς, η τεχνική ονομάζεται επίσης Causal Masking. Αναλυτικά, σε μια ακολουθία μήκους n, ο πίνακας μάσκας M έχει μέγεθος $n \times n$. Η μάσκα είναι γεμάτη έτσι ώστε για οποιοδήποτε token i, μπορεί να παρακολουθεί μόνο tokens $j \leq i$. Αυτό σημαίνει ότι $M_{ij} = -\infty$ για $j > i$, και $M_{ij} = 0$ για $j \leq i$. Αυτή η τριγωνική κάλυψη διασφαλίζει ότι ο μηχανισμός προσοχής δεν παραβιάζει την αυτοπαλινδρομική ιδιότητα της διαδικασίας δημιουργίας ακολουθίας αποτρέποντας την εξέταση μελλοντικών πληροφοριών κατά την πρόβλεψη.

2.5.18 Transformers

Βασιζόμενοι στην επιτυχία των μηχανισμών προσοχής, οι μετασχηματιστές εισήχθησαν από τους Vaswani et al., η αρχιτεκτονική του Transformer βασίζεται στην αυτοπροσοχή, επιτρέποντας στο μοντέλο να συλλαμβάνει τόσο τις τοπικές όσο και τις ολικές εξαρτήσεις [45] [46]. Στην αρχιτεκτονική του μετασχηματιστή, ο κωδικοποιητής αποτελείται από πολλαπλά επίπεδα νευρωνικών δικτύων αυτοπροσοχής και πλήρως συνδεδεμένων δικτύων (MLP), ενώ ο αποκωδικοποιητής ενσωματώνει τόσο την αυτοπροσοχή όσο και τη διασταυρούμενη προσοχή για να επιτρέπει την αλληλεπίδραση με την έξοδο του κωδικοποιητή. Αυτή η αρχιτεκτονική έχει φέρει επανάσταση στο NLP επιτρέποντας την παράλληλη επεξεργασία και βελτιώνοντας την επεκτασιμότητα, οδηγώντας σε μοντέλα όπως το BERT και το GPT που κυριαρχούν σε ένα ευρύ φάσμα εργασιών από την κατανόηση γλώσσας έως τη δημιουργία κειμένου [46] [47]. Οι μετασχηματιστές έχουν επίσης εφαρμοστεί σε συστήματα όρασης υπολογιστή. Οι μετασχηματιστές όρασης (ViTs) αντιμετωπίζουν τις εικόνες ως ακολουθίες μπαλωμάτων, χρησιμοποιώντας τον ίδιο μηχανισμό προσοχής που έχει σχεδιαστεί για την επεξεργασία κειμένου. Αυτή η προσέγγιση έχει δείξει ανταγωνιστική απόδοση σε σύγκριση με τις συνελικτικές αρχιτεκτονικές σε εργασίες ταξινόμησης και τμηματοποίησης εικόνων [48].

2.5.19 Embedding Layer

Ένα επίπεδο ενσωμάτωσης είναι ένα κρίσιμο στοιχείο στην επεξεργασία φυσικής γλώσσας (NLP) και στα συστήματα μηχανικής μάθησης, που επιτρέπει τη μετατροπή διακριτών κατηγορικών δεδομένων, όπως λέξεις ή tokens, σε συνεχείς διανυσματικούς χώρους σταθερών διαστάσεων. Το επίπεδο λειτουργεί ως μηχανισμός αναζήτησης όπου κάθε διακριτός δείκτης εισόδου σχετίζεται με μια πυκνή διανυσματική αναπαράσταση. Αυτές οι αναπαραστάσεις, αποθηκευμένες στα βάρη του στρώματος ενσωμάτωσης, αποτυπώνουν σημασιολογικές σχέσεις στα δεδομένα. Λέξεις με παρόμοιες έννοιες συχνά έχουν ενσωματώσεις που βρίσκονται κοντά στον διανυσματικό χώρο [49]. Μαθηματικά, το στρώμα ενσωμάτωσης ορίζεται ως μήτρα $E \in \mathbb{R}^{V \times d}$, όπου το V αντιπροσωπεύει το μέγεθος του λεξιλογίου και το d υποδηλώνει τη διάσταση ενσωμάτωσης. Δεδομένης μιας ακολουθίας εισόδου από tokens $x = \{x_1, x_2, \ldots, x_n\}$, το επίπεδο ενσωμάτωσης εξάγει μια ακολουθία διανυσμάτων $\{e_{x_1}, e_{x_2}, \ldots, e_{x_n}\}$, πού $e_{x_i} \in \mathbb{R}^d$ αντιστοιχεί στην ενσωμάτωση του i-οστού token. Αντιπροσωπεύοντας κάθε token x_i εισόδου ως διάνυσμα $v \in \mathbb{R}^V$ όπου $v_{x_i} = 0$ και $v_j = 0$ για $j \notin \{x_i\}$ (επίσης γνωστό ως one-hot διάνυσμα) το διάνυσμα εισόδου x γίνεται ένας αραιός δυαδικός πίνακας $x' \in \mathbb{R}^{n \times V}$ και η ακολουθία εξόδου $Y \in \mathbb{R}^{n \times d}$ υπολογίζεται ως :

$$Y = x'E \tag{2.58}$$

Ωστόσο, επειδή αυτή η διαδικασία είναι υπολογιστικά ακριβή, οι περισσότερες υλοποιήσεις θα αναζητήσουν απλώς τη μνήμη του E χρησιμοποιώντας το x ως δείκτες και η πραγματική πράξη γίνεται :

$$e_{x_i} = E[x_i], \quad \forall i \in \{1, 2, \ldots, n\}. \tag{2.59}$$

$$Y = \begin{bmatrix} e_{x_1} \\ e_{x_2} \\ \vdots \\ e_{x_n} \end{bmatrix} = \begin{bmatrix} E[x_1] \\ E[x_2] \\ \vdots \\ E[x_n] \end{bmatrix} \tag{2.60}$$

Η χρήση προεκπαιδευμένων ενσωματώσεων, όπως το Word2Vec ή το GloVe [49] [50], είναι συνηθισμένη για την επιτάχυνση της εκπαίδευσης και την αξιοποίηση της προηγούμενης γνώσης. Επιπλέον, οι ενσωματώσεις έχουν επεκταθεί πέρα από τις λέξεις για να αναπαραστήσουν οντότητες όπως:

- Γραφήματα: Χρησιμοποιούνται για την αναπαράσταση κόμβων ή ακμών σε μοντέλα που βασίζονται σε γραφήματα. Τα νευρωνικά δίκτυα γραφημάτων (GNN) χρησιμοποιούν ενσωματώσεις για να μετατρέψουν δεδομένα γραφημάτων, όπως κοινωνικά δίκτυα ή γραφήματα παραπομπών, σε συνεχείς διανυσματικούς χώρους για εργασίες όπως η ταξινόμηση κόμβων ή η πρόβλεψη συνδέσμων [51].

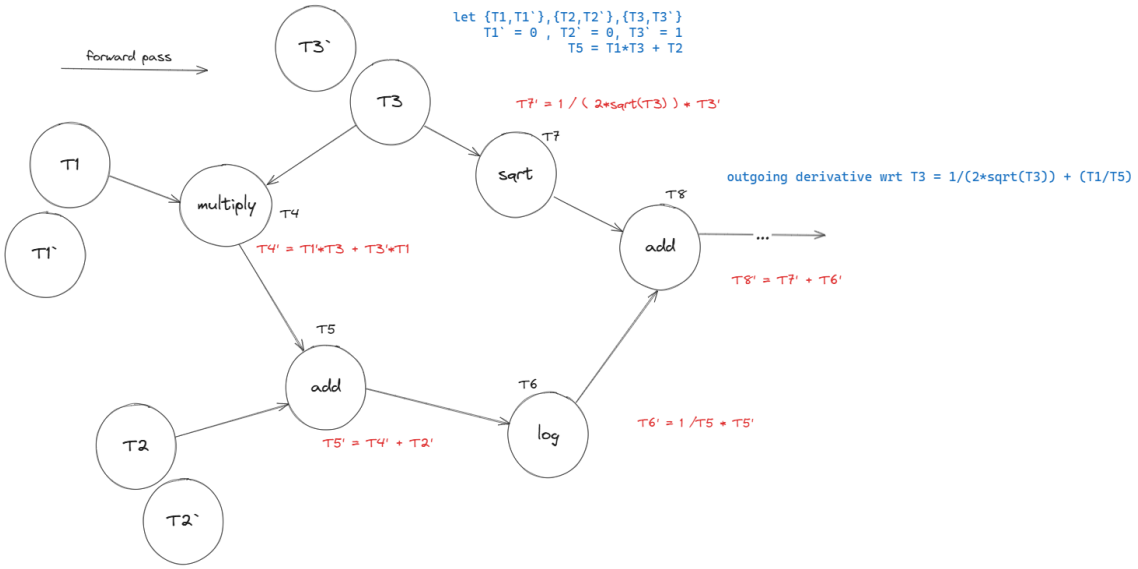
- Εικόνες: Σε ένα μοντέλο Visual Transformer (ViT), οι εικόνες χωρίζονται σε μικρά κομμάτια και κάθε κομμάτι αντιμετωπίζεται ως token παρόμοιο με τον τρόπο με τον οποίο αντιμετωπίζονται οι λέξεις στις εργασίες επεξεργασίας φυσικής γλώσσας (NLP). Αυτές οι υποδιαιρέσεις εικόνας στη συνέχεια ισοπεδώνονται και προβάλλονται σε διανύσματα υψηλών διαστάσεων (ενσωματώσεις) χρησιμοποιώντας μια γραμμική προβολή [48].

2.6 Automatic differentiation

2.6.1 Εισαγωγή

Η Αυτόματη Διαφόριση (AD) είναι μια υπολογιστική τεχνική που χρησιμοποιείται για την αποτελεσματική και ακριβή αξιολόγηση των τιμών των παραγώγων των συναρτήσεων. Άλλες μέθοδοι περιλαμβάνουν τη συμβολική διαφοροποίηση, η οποία μπορεί να παράγει σύνθετες εκφράσεις και αναφέρεται στον χειρισμό μαθηματικών παραστάσεων για να βρεθεί ο τύπος της παραγώγου και όχι η τιμή, και την αριθμητική διαφοροποίηση, που είναι ένα σύνολο αλγορίθμων αριθμητικής ανάλυσης που χρησιμοποιούνται για την εκτίμηση της τιμής μιας παραγώγου, η οποία πάσχει από σφάλματα προσέγγισης. Η αυτόματη διαφόριση βασίζεται στη συστηματική εφαρμογή του κανόνα της αλυσίδας για τον υπολογισμό των παραγώγων. Διασπά σύνθετες συναρτήσεις σε μια ακολουθία διαφοροποιήσιμων στοιχειωδών πράξεων (π.χ. πρόσθεση, διαίρεση, δύναμη) και μέσω της χρήσης ενός υπολογιστικού γραφήματος όπου κάθε κόμβος είναι μια πράξη με τους κομβους φύλλα να ειναι οι μεταβλητές, παρακολουθεί τις παραγώγους τους παράλληλα με τους αρχικούς υπολογισμούς.

2.6.2 Forward mode automatic differentiation



Σχήμα 2.4: Παράδειγμα ευθείας AD

Εισήχθη από R.E. Wengert το 1964 [52]. Στην ευθεία AD, οι παράγωγοι διαδίδονται παράλληλα με την αξιολόγηση της συνάρτησης, ξεκινώντας από τις μεταβλητές εισόδου και προχωρώντας προς τα εμπρός μέσω του υπολογιστικού γραφήματος της συνάρτησης. Κάθε πράξη στο υπολογιστικό γράφημα επεκτείνεται ώστε να λειτουργεί όχι μόνο στις τιμές αλλά και στις παραγώγους οι οποίες συνδυάζονται μέσω κανόνα αλυσίδας:

$$\frac{\partial w_i}{\partial x} = \sum_{j \in \{\text{parent nodes of } i\}} \frac{\partial w_j}{\partial x} \frac{\partial w_i}{\partial w_j} \tag{2.61}$$

Αυτό επιτυγχάνεται μέσω της χρήσης δυϊκών αριθμών, όπου το πρόσθετο στοιχείο κάθε αριθμού θα αντιπροσωπεύει την παράγωγο της συνάρτησης στον αριθμό. Η ευθεία AD υλοποιείται ως εξής, πρώτα

όλες οι μεταβλητές αντιπροσωπεύονται με τη μορφή δυϊκών αριθμών:

$$var_i = \text{dual}(w_i, \frac{\partial w_i}{\partial x_k}) \tag{2.62}$$

με $x_k = w_k$ να είναι η μεταβλητή ενδιαφέροντος που σημαίνει αρχικοποίηση της παραγώγου σε 1

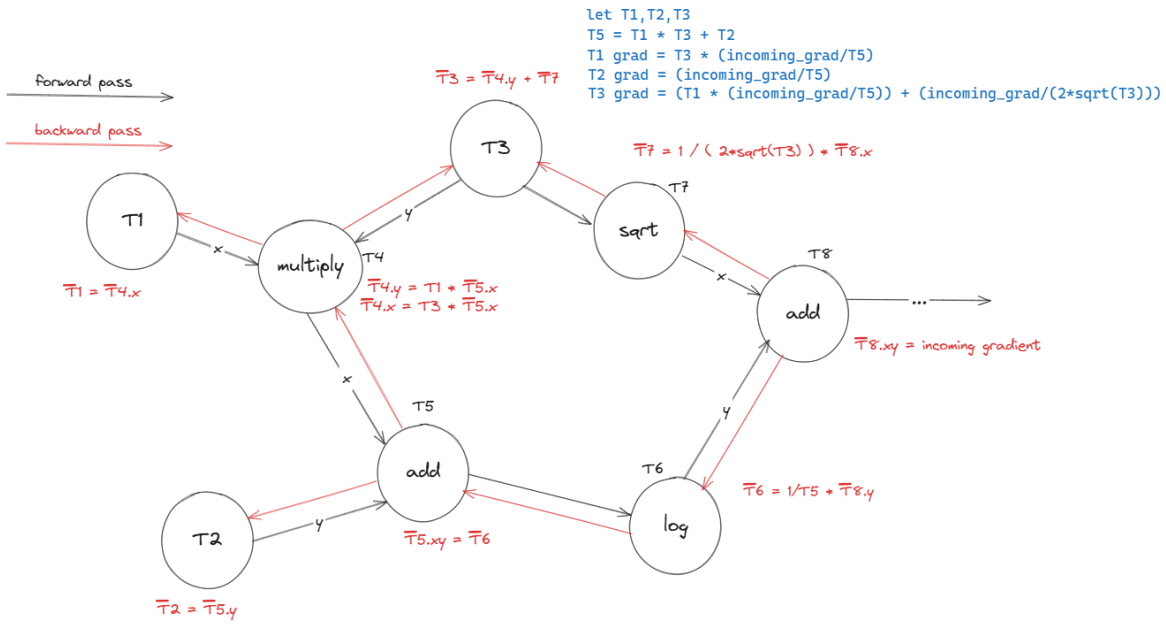
$$\frac{\partial w_k}{\partial x_k} = 1 \tag{2.63}$$

και αρχικοποίηση της παραγώγου σε 0 για κάθε άλλη μεταβλητη

$$\frac{\partial w_i}{\partial x_k} = 0 \quad \forall i \notin \{k\} \tag{2.64}$$

Στη συνέχεια, κάθε παράγωγος μιας υποέκφρασης του γραφήματος υπολογίζεται μαζί με την αξιολόγησή του και οι παράγωγοι συνδυάζονται χρησιμοποιώντας τον κανόνα της αλυσίδας. Ο υπολογισμός του Jacobian μιας συνάρτησης $f : \mathbb{R}^n \to \mathbb{R}^m$ ($m \times n$ matrix) με χρήση forward AD απαιτεί n περάσματα χρησιμοποιώντας την παραπάνω μεθοδολογία ορίζοντας τη δεύτερη συνιστώσα της μεταβλητής στόχου σε 1 και τη δεύτερη συνιστώσα όλων των άλλων μεταβλητών σε 0 για κάθε μεταβλητή.

2.6.3 Reverse mode automatic differentiation



Σχήμα 2.5: Παράδειγμα αντίστροφης AD

Ο εφευρέτης της αντίστροφης AD είναι άγνωστος ωστόσο ο Seppo Linnainmaa δημοσίευσε την αντίστροφη αυτόματη διαφόριση το 1976 [53] [54]. Η αντίστροφη AD είναι μια διαδικασία δύο περασμάτων. Αρχικά το πρώτο πέρασμα διασχίζει το γράφημα με σειρά εκτέλεσης και υπολογίζει την έξοδο της συνάρτησης και τυχόν ενδιάμεσες τιμές που απαιτούνται για τον υπολογισμό των παραγώγων. Έπειτα το

δεύτερο πέρασμα θα διασχίσει το γράφημα με αντίστροφη σειρά και θα υπολογίσει τις παραγώγους των μεταβλητών σε σχέση με την έξοδο εφαρμόζοντας τον κανόνα της αλυσίδας:

$$\frac{\partial y}{\partial w_i} = \sum_{j \in \{\text{child nodes of } i\}} \frac{\partial y}{\partial w_j} \frac{\partial w_j}{\partial w_i} \tag{2.65}$$

Η αποθήκευση δεδομένων και οδηγιών για το αντίστροφο πέρασμα υλοποιείται συνήθως με μια δομή δεδομένων που ονομάζεται ”tape” (η εγγραφή των λειτουργιών σε ένα υπολογιστικό γράφημα και στη συνέχεια η αναπαραγωγή του προς τα πίσω μοιάζει με την εγγραφή βίντεο ή ήχου σε κασέτα) ή λίστα του Wengert [55]. Ωστόσο, υπάρχουν υλοποιήσεις που δεν χρησιμοποιούν αυτήν την τεχνική [56]. Ο υπολογισμός του Jacobian μιας συνάρτησης $f : \mathbb{R}^n \to \mathbb{R}^m$ με χρήση reverse AD απαιτεί m περάσματα με το κάθε πέρασμα για κάθε έξοδο f_i να υπολογίζει την γραμμή του πίνακα που θα είναι το ανάστροφο gradient:

$$\nabla^T f_i = \begin{bmatrix} \frac{\partial f_i}{\partial w_1} & \cdots & \frac{\partial f_i}{\partial w_n} \end{bmatrix} \tag{2.66}$$

2.6.4 Θέματα απόδοσης

Όπως αναφέρθηκε, η πολυπλοκότητα της αντίστροφης AD εξαρτάται από τον αριθμό των εξόδων και της μπροστινή AD εξαρτάται από τον αριθμό των εισόδων, η επιλογή της πλησιέστερης στη βέλτιστη μέθοδο μπορεί να ερευνηθεί εξετάζοντας αυτές τις μεταβλητές για κάθε πρόβλημα, ωστόσο ο υπολογισμός του Jacobian matrix μιας διανυσματικής συνάρτησης με τη μικρότερη υπολογιστική πολυπλοκότητα είναι ένα πλήρες μη ντετερμινιστικό πρόβλημα πολυωνυμικού χρόνου [57]. Στα περισσότερα προβλήματα μηχανικής μάθησης, οι είσοδοι στο νευρωνικό δίκτυο υπερτερούν κατά πολύ των εξόδων του δικτύου και επομένως η αντίστροφη AD (backpropagation) είναι το πιο κοινό εργαλείο. Ωστόσο, η χρήση της ευθείας AD είναι υπό έρευνα καθώς μπορεί να μετριάσει το αποτύπωμα της μνήμης σε αντίθεση με την αντίστροφη AD που απαιτεί την αποθήκευση ενδιάμεσων τιμών για χρήση στο backward-pass [58].

2.6.5 Differentiable Programming

Ο διαφορίσιμος προγραμματισμός είναι ένα προγραμματιστικό παράδειγμα που διαδόθηκε ιδιαίτερα από τον Yann Lecun στα τέλη της δεκαετίας του 2010. Η βασική του αρχή είναι ότι τα προγράμματα υπολογιστών είναι δομημένα με τρόπο που να μπορούν να υπολογίστουν παράγωγοι αυτόματα χρησιμοποιώντας την αυτόματη διαφόριση ως κύριο συστατικό. Πλαισιώνεται ως τρόπος σχεδιασμού περίπλοκων συστημάτων που συνδυάζουν παραδοσιακές δομές προγραμματισμού (π.χ. ροή ελέγχου, βρόχους και συνθήκες) με διαφοροποιήσιμες πράξεις, επιτρέποντας τη βελτιστοποίηση από άκρο σε άκρο [59]. Το παράδειγμα επεκτείνεται πέρα από τα νευρωνικά δίκτυα, καθώς στοχεύει να κάνει κάθε συνάρτηση και πράξη ενός προγράμματος υπολογιστή διαφοροποιήσιμη, καθιστώντας το ως μια γενίκευση για ιδέες όπως το back propagation και οι διαφορίσιμοι προσομοιωτές φυσικής όπως η τεχνική του διαφορίσιμου rendering [60] [61].

2.6.6 Computational graphs

2.6.6.1 Εισαγωγή

Τα υπολογιστικά γραφήματα είναι μια θεμελιώδης αναπαράσταση που χρησιμοποιείται σε διάφορους τομείς της επιστήμης των υπολογιστών, ιδιαίτερα στους τομείς της μηχανικής μάθησης και βελτιστοποίησης. Ένα υπολογιστικό γράφημα είναι ένα κατευθυνόμενο ακυκλικό γράφημα (DAG) όπου οι κόμβοι αντιπροσωπεύουν μαθηματικές πράξεις και οι ακμές υποδηλώνουν τη ροή δεδομένων μεταξύ αυτών των πράξεων. Αυτή η αφαίρεση παρέχει έναν δομημένο τρόπο μοντελοποίησης πολύπλοκων υπολογισμών με την αποσύνθεσή τους σε μια σειρά μικρότερων, διασυνδεδεμένων λειτουργιών. Η δομημένη αναπαράσταση επιτρέπει τον αποτελεσματικό υπολογισμό των παραγώγων χρησιμοποιώντας αλγόριθμους όπως backpropagation, ο ακρογωνιαίος λίθος της εκπαίδευσης του σύγχρονου νευρωνικού δικτύου.

Η modular φύση των υπολογιστικών γραφημάτων ενισχύει επίσης την ευελιξία και την επεκτασιμότητα. Οι κόμβοι μπορούν να αντιπροσωπεύουν μια ποικιλία λειτουργιών, συμπεριλαμβανομένων γραμμικών μετασχηματισμών, συναρτήσεων ενεργοποίησης ή ακόμα και συναρτήσεων που καθορίζονται από τον χρήστη. Αυτή η ευελιξία επιτρέπει την σχεδίαση και την εκπαίδευση πολύπλοκων μοντέλων, με παράλληλη βελτιστοποίηση υπολογιστικών πόρων.

2.6.6.2 Static graph

Το στατικό γράφημα είναι μια μέθοδος καθορισμού και μετά εκτέλεσης, αυτός ο τύπος γραφήματος κατασκευάζεται μία φορά και στη συνέχεια εκτελείται σε επανάληψη. Μπορεί να επιτρέψει τη βελτιστοποίηση ολόκληρου του δικτύου με το κλάδεμα αχρησιμοποίητων ή διπλών κόμβων, τη σύντηξη κόμβων που μπορούν να συγχωνευθούν και πιθανή παραλληλοποίηση ανεξάρτητων κόμβων. Λόγω της φύσης του γραφήματος που έχει σταθερή δομή, η ροή ελέγχου και η εκτέλεση υπό όρους είναι δύσκολη και οι απλές δομές προγραμματισμού δεν θα λειτουργήσουν. Τα στατικά γραφήματα θεμελιώνουν αυτό που είναι γνωστό ως νωχελική αξιολόγηση (lazy evaluation), μια μέθοδος αξιολόγησης που συναντάται συνήθως σε συναρτησιακές γλώσσες προγραμματισμού όπου η αξιολόγηση κάθε έκφρασης αναβάλλεται μέχρι να χρειαστεί ή να ζητηθεί ρητά, πράγμα που σημαίνει ότι δεν χρειάζεται να διατεθούν εκατομμύρια byte δεδομένων εκτός και αν χρειαστεί [62].

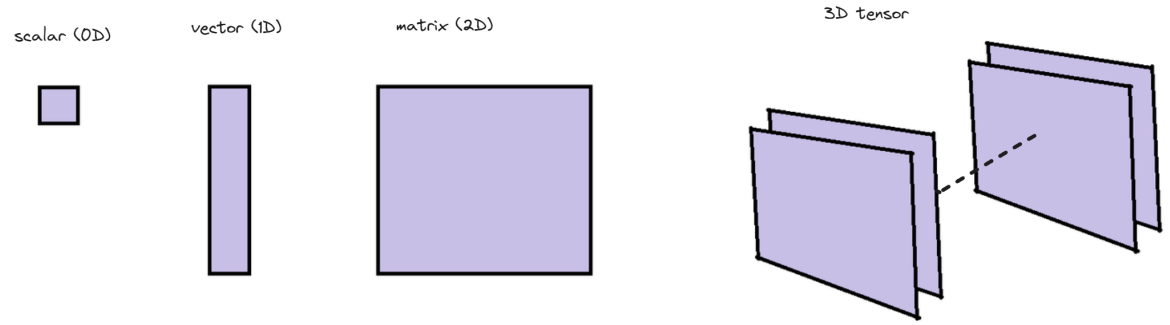
2.6.6.3 Dynamic graph

Το δυναμικό γράφημα είναι μια μέθοδος καθορισμού με εκτέλεση, αυτός ο τύπος γραφήματος κατασκευάζεται εκ νέου κάθε φορά μαζί με την εκτέλεση. Η δυναμική του φύση προσφέρει μεγαλύτερη ευελιξία όπως η ροή ελέγχου που μπορεί να εφαρμοστεί χρησιμοποιώντας ήδη υπάρχουσες δομές προγραμματισμού, αλλά δεν έχει τη δυνατότητα να κάνει βελτιστοποιήσεις σε ολόκληρο το γράφημα που σημαίνει ότι μπορεί να εισαγάγει μια πρόσθετη ποινή χρόνου εκτέλεσης. Για να αντιπαραβάλει τη στρατηγική αξιολόγησης στατικών γραφημάτων, ένα δυναμικό γράφημα θα εφαρμόσει αυτό που είναι γνωστό ως ανυπόμονη αξιολόγηση (eager evaluation) που σημαίνει ότι αξιολογεί εκφράσεις καθώς τις συναντά παράλληλα με τη δημιουργία του γραφήματος, παρόμοια με τις περισσότερες επιτακτικές γλώσσες προγραμματισμού.

2.7 Tensor

2.7.1 Εισαγωγή

Με τον όρο τανυστής στο πλαίσιο των βιβλιοθηκών μηχανικής μάθησης θα αναφερόμαστε σε μια γενίκευση πινάκων σε N διαστάσεις. Επομένως, ένας πίνακας είναι ένας δισδιάστατος τανυστής, ένα διάνυσμα είναι ένας τανυστής μιας διάστασης, ακόμη, ένας αριθμός μπορεί να θεωρηθεί ως τανυστής 0 διαστάσεων και ένας τρισδιάστατος τανυστής μπορεί να είναι μια στοίβα πινάκων.



Σχήμα 2.6: Παράδειγμα τανυστών διαφορετικών διαστάσεων

Οι τανυστές έχουν σχήμα (shape) το οποίο είναι ένα διάνυσμα που υποδηλώνει τις διαστάσεις του τανυστή $(d_0, d_1, \cdots, d_N)$ για έναν τανυστή N-διάστασης. Η κατάταξη ή η διάσταση ενός τανυστή είναι ο αριθμός των στοιχείων του διανύσματος σχήματος που σημαίνει ότι ένας τανυστής N-διάστασης έχει κατάταξη N.

2.7.2 Tensor memory formats

2.7.2.1 Εισαγωγή

Η μορφή μνήμης ενός τανυστή αναφέρεται στον τρόπο με τον οποίο τα δεδομένα ενός πολυδιάστατου πίνακα τοποθετούνται σε γραμμική μονοδιάστατη μνήμη. Μπορούν να έχουν σημαντικό αντίκτυπο στην υπολογιστική απόδοση επηρεάζοντας τα μοτίβα πρόσβασης στη μνήμη, τη χρήση της κρυφής μνήμης και τον παραλληλισμό [63]. Βελτιστοποιώντας τον τρόπο αποθήκευσης και πρόσβασης στα δεδομένα, οι διαφορετικές μορφές μνήμης μπορούν να βελτιώσουν ή να μειώσουν την απόδοση για ένα ευρύ φάσμα αριθμητικών και επιστημονικών υπολογισμών.

2.7.2.2 Matrix formats

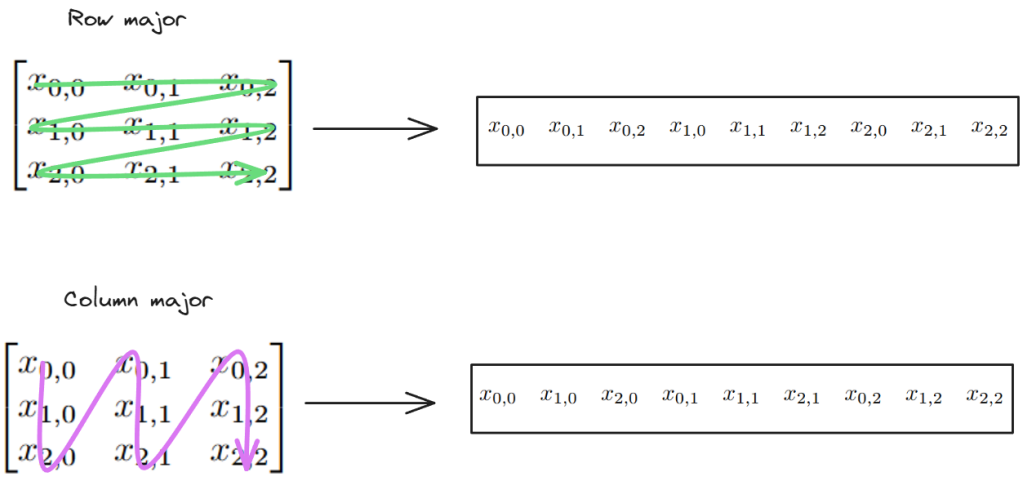
Στους υπολογισμούς πυκνής μήτρας, υπάρχουν δύο κύριες μορφές που μπορούν να επιτρέψουν την αποτελεσματική πρόσβαση σε στοιχεία ευθυγραμμίζοντας τα δεδομένα με ιεραρχίες μνήμης. Η διαδοχική πρόσβαση σε στοιχεία που είναι αποθηκευμένα συνεχόμενα στη μνήμη ελαχιστοποιεί τις αστοχίες της κρυφής μνήμης και αξιοποιεί την τοπικότητα της αναφοράς. Οι δύο αυτές μορφές είναι:

- Row Major: Σε μορφή μείζονος σειράς, τα στοιχεία μιας μήτρας αποθηκεύονται σειρά προς σειρά σε συνεχόμενες θέσεις μνήμης. Αυτή η μορφή χρησιμοποιείται σε γλώσσες όπως η C. Είναι ωφέλιμη σε αλγόριθμους προσανατολισμένους στη σειρά, όπως η Γκαουσιανή απαλοιφή. Ο υπολογισμός της διεύθυνσης για έναν πίνακα διαστάσεων $D \in \mathcal{R}^N$ που δίνεται σε ένα διάνυσμα δεικτών $I \in \mathcal{R}^M$ με $M <= N$ θα είναι:

$$\sum_{i=1}^{M} \left(\prod_{j=i+1}^{N} D_j \right) I_i \tag{2.67}$$

- Column Major: Σε μορφή μείζονος στήλης, τα στοιχεία αποθηκεύονται στήλη προς στήλη σε συνεχόμενες θέσεις μνήμης. Αυτή η μορφή υιοθετείται σε γλώσσες όπως η Fortran και είναι επωφελής σε εφαρμογές όπου κυριαρχούν οι αλγόριθμοι προσανατολισμένοι στη στήλη. Είναι πολύ συνηθισμένη στις βιβλιοθήκες μαθηματικών αλγορίθμων αφού η Fortran ήταν ευρέως διαδεδομένη. Ο τύπος υπολογισμού διεύθυνσης σε αυτήν την περίπτωση θα είναι:

$$\sum_{i=1}^{M} \left(\prod_{j=1}^{i-1} D_j \right) I_i \tag{2.68}$$



Σχήμα 2.7: Παράδειγμα διάταξης μνήμης για τις μορφές Row Major και Column Major

2.7.2.3 Image formats

Οι μορφές μνήμης εικόνας καθορίζουν τον τρόπο με τον οποίο αποθηκεύονται και επεξεργάζονται πολυδιάστατα δεδομένα εικόνας. Οι δύο μορφές που χρησιμοποιούνται συνήθως είναι:

- Channels-Last (NHWC) : Η μορφή οργανώνει τα δεδομένα με τη σειρά (ύψος, πλάτος, κανάλια), που συχνά αντιπροσωπεύεται ως HWC για δεδομένα 3D ή NHWC για δεδομένα 4D, όπου N είναι

το μέγεθος παρτίδας.

- Channels-First (NCHW) : Αυτή η μορφή οργανώνει τα δεδομένα με τη σειρά (κανάλια, ύψος, πλάτος), που αντιπροσωπεύεται ως CHW για δεδομένα 3D ή NCHW για δεδομένα 4D.

2.7.2.4 Arbitary strides format

Όλες οι μορφές μνήμης μπορούν να γενικευθούν στη μορφή strided. Αυτή η μορφή, μαζί με το διάνυσμα διαστάσεων $\left[d_0, d_1, \cdots, d_n\right]$, απαιτεί την αποθήκευση ενός επιπλέον διανύσματος $\left[s_0, s_1, \cdots, s_n\right]$ όπου κάθε s_i υποδηλώνει το βήμα για την i-οστή διάσταση. Ο υπολογισμός της διεύθυνσης είναι τότε ένα απλό εσωτερικό γινόμενο, για ένα διάνυσμα δεικτών $\left[i_0, i_1, \cdots, i_m\right]$ με $m <= n$ η μετατόπιση στη μνήμη θα είναι:

$$\sum_{j=0}^{m} i_j s_j \tag{2.69}$$

Αυτή η μορφή μνήμης είναι φυσικά η πιο ευέλικτη αλλά απαιτεί ένα πρόσθετο διάνυσμα και η ύπαρξη αυθαίρετων βημάτων μπορεί να προκαλέσει ανεπιθύμητες ποινές απόδοσης όταν οι τανυστές δεν είναι συσκευασμένοι σε μια μορφή που ελαχιστοποιεί τις αστοχίες της κρυφής μνήμης, για παράδειγμα εάν κάθε επόμενο στοιχείο απέχει περισσότερο από ένα μέγεθος γραμμής κρυφής μνήμης σε byte από το προηγούμενο.

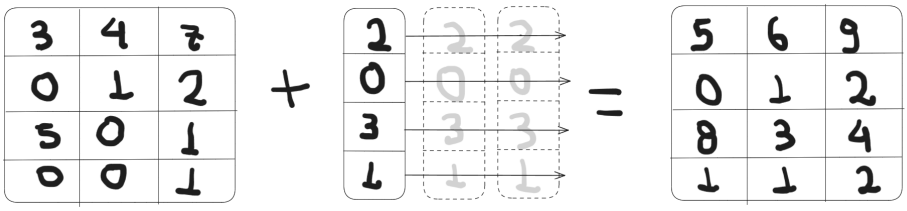
2.7.3 Broadcasting

Η μετάδοση (broadcasting) είναι ένα χαρακτηριστικό που υπάρχει σε όλες σχεδόν τις μαθηματικές βιβλιοθήκες που ασχολούνται με πολυδιάστατους πίνακες. Με τον όρο μετάδοση θα αναφερθούμε στην τεχνική που χρησιμοποιείται για να γίνει αριθμητική σε τανυστές διαφορετικών σχημάτων [64].

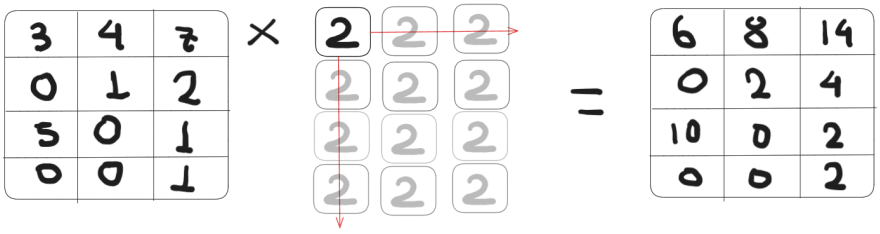
Για να εκτελεστεί μια πράξη σε N τανυστές, αυτοί οι τανυστές συνήθως απαιτείται να έχουν το ίδιο σχήμα. Ωστόσο, υπό ορισμένες συνθήκες, ένας μικρότερος τανυστής μπορεί να μεταδοθεί σε μεγαλύτερο εάν οι διαστάσεις μπορούν να μεταδοθούν. Οι κανόνες και οι προϋποθέσεις μετάδοσης είναι οι εξής: ξεκινώντας από τα δεξιά προς τα αριστερά δύο διαστάσεις είναι συμβατές εάν είναι ίσες ή εάν μία από αυτές είναι 1 η οποία στη συνέχεια θα μεταδοθεί στη μεγαλύτερη διάσταση.

Στην περίπτωση όπου η διάσταση του τανυστή A είναι μικρότερη από αυτή του τανυστή B, οι διαστάσεις που λείπουν θεωρούνται στα αριστερά και είναι ίσες με 1, καθιστώντας έτσι ένα (1D) διάνυσμα $v \in \mathcal{R}^n$ μεταδόσιμο σε μήτρα $M \in \mathcal{R}^{m \times n}$ αφού η πιο δεξιά διάσταση ταιριάζει.

Η περίπτωση της κλιμάκωσης ενός τανυστή με έναν αριθμό μπορεί να εκφραστεί ως ειδική περίπτωση μετάδοσης εάν θεωρήσουμε τον αριθμό ως βαθμωτό τανυστή (0-διάστατο) που μεταδίδεται σε έναν πολυδιάστατο τανυστή, επειδή όλες οι διαστάσεις που λείπουν θεωρούνται στο 1.



Σχήμα 2.8: Παράδειγμα broadcasting $v \in \mathcal{R}^n$ σε $M \in \mathcal{R}^{m \times n}$



Σχήμα 2.9: Παράδειγμα κλιμάκωσης ενός τανυστή με το 2

2.8 Low precision arithmetic

2.8.1 Εισαγωγή

Οι υπολογισμοί χαμηλής ακρίβειας έχουν γίνει κρίσιμοι στα νευρωνικά δίκτυα για τη βελτίωση της υπολογιστικής απόδοσης και τη μείωση της χρήσης πόρων. Η χρήση μορφών μειωμένου πλάτους bit μειώνει τη μνήμη και την κατανάλωση ενέργειας, ωφελώντας τις εφαρμογές σε συσκευές και συστήματα μεγάλης κλίμακας [65]. Τα νευρωνικά δίκτυα μπορούν να ανεχθούν μικρά αριθμητικά σφάλματα, καθιστώντας τους υπολογισμούς χαμηλής ακρίβειας βιώσιμους χωρίς να επηρεάζουν σημαντικά την ακρίβεια. Τεχνικές όπως η κβάντιση (quantization) και η εκπαίδευση μεικτής ακρίβειας (mixed precision training) συμβάλλουν στη βελτιστοποίηση της απόδοσης διατηρώντας παράλληλα την ποιότητα του μοντέλου [66] [67] [68]. Ως αποτέλεσμα, η αριθμητική χαμηλής ακρίβειας παίζει βασικό ρόλο στο σύγχρονο hardware βαθιάς μάθησης και στις μεθόδους βελτιστοποίησης.

2.8.2 Float16

Μία από τις αξιοσημείωτες προσθήκες στην αναθεώρηση του 2008 του προτύπου IEEE 754 ήταν η συμπερίληψη της μορφής κινητής υποδιαστολής μισής ακρίβειας, γνωστή και ως binary16. Αυτή η μορφή χρησιμοποιεί 16 bit τα οποία κατανέμονται ως εξής: 1 bit πρόσημο, 5 bit εκθέτης και 10 bit mantissa.

Η μορφή μισής ακρίβειας παρέχει μικρότερο εύρος και μικρότερη ακρίβεια σε σύγκριση με τις μορφές μονής και διπλής ακρίβειας. Το εύρος του εκθέτη είναι από -14 έως +15 και η μικρότερη θετική μη μηδενική τιμή είναι περίπου $5,96 \times 10^{-8}$ με την μεγαλύτερη να είναι 65504 [69].

2.8.3 BFloat16

Το bfloat16 (Brain Floating Point) της Google Brain είναι μια αναπαράσταση κινητής υποδιαστολής 16 bit που χρησιμοποιείται κυρίως σε εφαρμογές μηχανικής μάθησης και νευρωνικών δικτύων, μειώνοντας τις απαιτήσεις μνήμης και εύρους ζώνης σε σύγκριση με την παραδοσιακή μορφή κινητής υποδιαστολής 32 bit (float32). Τα 16 bit κατανέμονται ως εξής: 1 bit πρόσημο, 8 bit εκθέτης, όπως το float32 και 7 bit mantissa [70] [71]. Σε αντίθεση με το float16, η μετατροπή από και σε float32 όταν χρησιμοποιείται bfloat16 είναι εύκολη επειδή μοιράζονται τα ίδια bits εκθέτη. Το εύρος των τιμών bfloat16 είναι παρόμοιο με αυτό του float32 επειδή μοιράζονται τον ίδιο αριθμό bit εκθέτη. Η ακρίβεια του bfloat16 είναι χαμηλότερη από το float32 και το τυπικό binary16 του IEEE 754 λόγω του μειωμένου μεγέθους mantissa.

2.8.4 Float8

Το FP8 κωδικοποιεί αριθμούς κινητής υποδιαστολής χρησιμοποιώντας μόνο 8 bit, επιτρέποντας σημαντική μείωση στις απαιτήσεις μνήμης και εύρους ζώνης. Παρά το μειωμένο πλάτος bit, το FP8 διατηρεί επαρκή ακρίβεια για να διευκολύνει την εκπαίδευση μεικτής ακρίβειας και το fine-tuning σε μεγάλα νευρωνικά δίκτυα [72]. Δύο βασικές μορφές FP8 που υιοθετούνται ευρέως είναι οι E4M3 και E5M2 [73]. Αυτές οι μορφές διαφέρουν στην κατανομή των bit για τον εκθέτη και την mantissa:

- E4M3: Εκθέτης 4 bit και mantissa 3 bit, μεγαλύτερη ακρίβεια σε βάρος του μειωμένου εύρους.
- E5M2: Εκθέτης 5 bit και mantissa 2 bit, εκτεταμένο εύρος με κόστος την ακρίβεια.

2.9 Flynn's Taxonomy

Η κατηγοριοποίηση του Flynn, που προτάθηκε από τον Michael J. Flynn, είναι ένα σύστημα ταξινόμησης για αρχιτεκτονικές υπολογιστών που βασίζεται στον αριθμό των ταυτόχρονων ροών εντολών και ροών δεδομένων που μπορούν να χειριστούν [74] [75]. Αυτή η ταξινόμηση βοηθά στην κατανόηση των δυνατοτήτων και των περιορισμών διαφορετικών αρχιτεκτονικών παράλληλων υπολογιστών, καθοδηγώντας τόσο το σχεδιασμό του hardware όσο και την ανάπτυξη λογισμικού για βέλτιστη απόδοση σε διάφορες εφαρμογές.

2.10 Superscalar CPUs & SIMD

Ο υπερβαθμωτός (supercalar) επεξεργαστής με απλά λόγια συνήθως περιγράφει έναν μικροεπεξεργαστή που προσπαθεί να κάνει περισσότερα από ένα πράγματα τη φορά. Βελτιώνει την απόδοση εκτελώντας πολλές εντολές ταυτόχρονα κατά τη διάρκεια κάθε κύκλου ρολογιού. Ανεξάρτητες οδηγίες προγραμματίζονται και αποστέλλονται σε διαφορετικές μονάδες εκτέλεσης. Προηγμένες τεχνικές, όπως η εκτέλεση

εκτός παραγγελίας, η πρόβλεψη διακλάδωσης και η διοχέτευση εντολών, χρησιμοποιούνται για τη μεγιστοποίηση της απόδοσης και την ελαχιστοποίηση των σημείων συμφόρησης. Σε αντίθεση με τους βαθμωτούς επεξεργαστές που λειτουργούν μόνο σε ένα στοιχείο δεδομένων τη φορά με τρόπο SISD (single instruction single data) όπως ορίζεται στην κατηγοριοποίηση του Flynn [74] [75]. Ένας σημαντικός τύπος υπερβαθμωτών επεξεργαστών ονομάζεται διανυσματικός επεξεργαστής που περιγράφει έναν επεξεργαστή που λειτουργεί σε διανύσματα δεδομένων αντί για μεμονωμένα στοιχεία δεδομένων με τρόπο SIMD (single instruction multiple data), ο οποίος ορίζεται επίσης στην κατηγοριοποίηση του Flynn [74] [75].

2.11 Θέματα παραλληλισμού

2.11.1 Εισαγωγή

Η παραλληλοποίηση είναι η διαδικασία εκτέλεσης πολλαπλών υπολογισμών ή εργασιών ταυτόχρονα. Έχει διαφορετικές μορφές, μπορεί είτε να σημαίνει ότι ένας αλγόριθμος χωρίζεται σε μικρότερα κομμάτια που μπορούν να προγραμματιστούν να εκτελούνται σε διαφορετικά νήματα ή προγραμματισμός ανεξάρτητων εργασιών που δεν έχουν καμία σχέση μεταξύ τους για να εκτελούνται παράλληλα. Δύο σημαντικές μορφές παραλληλισμού είναι:

- Ο παραλληλισμός επιπέδου εντολών (ILP): Αναφέρεται στην εκτέλεση μιας σειράς εντολών ταυτόχρονα και μπορεί να γίνει από τον επεξεργαστή ή τον μεταγλωττιστή χρησιμοποιώντας διάφορες τεχνικές όπως η διοχέτευση εντολών και η εκτέλεση εκτός σειράς. Η υπόδειξη στον μεταγλωττιστή και/ή στον επεξεργαστή για χρήση του ILP μπορεί να γίνει με τον προγραμματισμό μιας σειράς εντολών που δεν έχουν καμία εξάρτηση μεταξύ τους.
- Ο παραλληλισμός επιπέδου δεδομένων(Data parallelism): Αναφέρεται στη διαίρεση των δεδομένων σε κομμάτια και στον προγραμματισμό μιας εργασίας ή υπολογισμού σε κάθε κομμάτι παράλληλα. Ο παραλληλισμός δεδομένων μπορεί να επιτευχθεί χρησιμοποιώντας πολλαπλά νήματα εκτέλεσης, αλλά και η χρήση του SIMD θεωρείται μια μορφή παραλληλισμού δεδομένων.

2.11.2 Cache coherence

Η συνοχή της κρυφής μνήμης είναι μια θεμελιώδης πρόκληση στα συστήματα πολλαπλών επεξεργαστών που αναφέρεται στη συνοχή των κοινόχρηστων δεδομένων σε ένα περιβάλλον πολλαπλών επεξεργαστών. Όταν πολλοί επεξεργαστές έχουν τις δικές τους κρυφές μνήμες καθώς και πρόσβαση στον ίδιο χώρο διευθύνσεων, μπορεί να προκύψουν ασυνέπειες λόγω ανεξάρτητων ενημερώσεων κρυφής μνήμης. Το πρόβλημα συνοχής παρουσιάζεται όταν ένας επεξεργαστής τροποποιεί ένα μπλοκ δεδομένων, αλλά οι άλλες κρυφές μνήμες εξακολουθούν να διατηρούν παλιά αντίγραφα του ίδιου μπλοκ.

Για την αντιμετώπιση προβλημάτων συνοχής της κρυφής μνήμης, έχουν αναπτυχθεί διάφορα πρωτόκολλα, τα οποία μπορούν να κατηγοριοποιηθούν ευρέως σε:

- Directory-Based: Αυτά τα πρωτόκολλα χρησιμοποιούν έναν κεντρικό κατάλογο που διατηρεί τα κοινόχρηστα δεδομένα. Οι επεξεργαστές ζητούν δεδομένα από τον κατάλογο που διασφαλίζει τη συνέπεια συντονίζοντας ενημερώσεις και ακυρώσεις.

- Bus Snooping: Σε αυτά τα πρωτόκολλα, οι κρυφές μνήμες παρακολουθούν έναν κοινόχρηστο δίαυλο επικοινωνίας [76]. Όταν ένας επεξεργαστής τροποποιεί ένα μπλοκ δεδομένων, άλλες κρυφές μνήμες εντοπίζουν αυτήν την αλλαγή και ενημερώνουν ή ακυρώνουν τα αντίγραφά τους ανάλογα.

Ενώ οι μηχανισμοί συνοχής της κρυφής μνήμης βελτιώνουν τη συνέπεια και την αξιοπιστία, εισάγουν επιβάρυνση λόγω αυξημένης κίνησης μνήμης και καθυστερήσεων συγχρονισμού.

2.11.3 False sharing

Το false sharing είναι ένα ζήτημα απόδοσης που παρουσιάζεται σε προγράμματα πολλαπλών νημάτων όταν τα νήματα μοιράζονται ακούσια γραμμές κρυφής μνήμης, οδηγώντας σε περιττό συγχρονισμό και περιττή εκτέλεση πρωτοκόλλων συνοχής της κρυφής μνήμης.

Στους σύγχρονους επεξεργαστές πολλαπλών πυρήνων, κάθε πυρήνας έχει τη δική του τοπική κρυφή μνήμη και οι γραμμές κρυφής μνήμης αποτελούνται συνήθως από 64 byte. Εάν δύο ή περισσότερα νήματα λειτουργούν σε διαφορετικές μεταβλητές που τυχαίνει να βρίσκονται στην ίδια γραμμή κρυφής μνήμης, μπορεί να προκύψει false sharing. Παρόλο που τα νήματα δεν μοιράζονται στην πραγματικότητα δεδομένα, οι ενημερώσεις τους σε διαφορετικές μεταβλητές εντός της ίδιας γραμμής κρυφής μνήμης μπορούν να ενεργοποιήσουν πρωτόκολλα συνοχής της κρυφής μνήμης και να προκαλέσουν περιττές ακυρώσεις κρυφής μνήμης (cache invalidation), μειώνοντας την αποτελεσματικότητα της παράλληλης εκτέλεσης. Αυτό μπορεί να προκαλέσει σημαντική υποβάθμιση της απόδοσης, ειδικά σε προγράμματα με υψηλό ανταγωνισμό μεταξύ νημάτων.

Για τον μετριασμό του προβλήματος, οι προγραμματιστές μπορούν να χρησιμοποιήσουν τεχνικές όπως padding ή η ευθυγράμμιση μεταβλητών με συχνή πρόσβαση. Με την ευθυγράμμιση των μεταβλητών στα όρια των γραμμών της κρυφής μνήμης ή την προσθήκη εικονικών bytes μεταξύ τους, ελαχιστοποιείται η πιθανότητα του προβλήματος false sharing.

2.11.4 Data race

Το data race είναι ένα πρόβλημα στην ακεραιότητα των δεδομένων. Ορίζεται ως μια κατάσταση κατά την οποία δύο ή περισσότερες διεργασίες έχουν ταυτόχρονα πρόσβαση σε έναν πόρο και τουλάχιστον μία από αυτές τον τροποποιεί.. Τα αποτελέσματα θα είναι απροσδιόριστα και ο πόρος μπορεί να καταλήξει να περιέχει οποιαδήποτε τιμή των υποψηφίων ενημερώσεων ή έναν ανούσιο συνδυασμό των bits των ενημερώσεων εάν περισσότερες από μία διεργασίες προσπαθούν να τροποποιήσουν ταυτόχρονα και οι αναγνώστες μπορούν να λάβουν τα δεδομένα σε οποιαδήποτε τυχαία κατάσταση που θα μπορούσε να είναι άκυρη.

2.11.5 Critical section

Για την αποφυγή των data races, η περίπτωση κατά την οποία γίνεται ταυτόχρονη ανάγνωση και εγγραφή σε έναν πόρο και η περίπτωση στην οποία υπάρχουν περισσότερες από μία διεργασίες που προσπαθούν να τροποποιήσουν ταυτόχρονα τον πόρο πρέπει να πραγματοποιηθούν σειριακά, πράγμα που σημαίνει

ότι πρέπει να συμβεί το καθένα μετά ή πριν από το άλλο. Για να γίνει αυτό ορίζουμε ένα κρίσιμο τμήμα ως προστατευμένη περιοχή του κώδικα στην οποία επιτρέπεται η είσοδος μόνο σε μία διεργασία τη φορά.

2.11.6 Mutual exclusion

2.11.6.1 Εισαγωγή

Αμοιβαίος αποκλεισμός, είναι η προστασία κρίσιμων τμημάτων από την ταυτόχρονη πρόσβαση. Μια ιδιότητα σύμφωνα με την οποία εάν μια διεργασία προσπαθήσει να αποκτήσει πρόσβαση σε ένα κρίσιμο τμήμα που εκτελεί μία άλλη διαδικασία, τότε της απαγορεύεται η πρόσβαση μέχρι να ολοκληρωθεί η επεξεργασία της διαδικασίας που είναι σε εξέλιξη, εισήχθη από τον Edsger W. Dijkstra [77].

2.11.6.2 Sempaphore

Ο Σημαφόρος, που εισήχθη επίσης από τον Edgar D. Dijkstra, είναι μια μεταβλητή που είναι υπεύθυνη για τον έλεγχο πρόσβασης και μπορεί να βοηθήσει στην επιβολή του αμοιβαίου αποκλεισμού [78]. Διαφορετικές υλοποιήσεις για διαφορετικά μοτίβα πρόσβασης ελέγχου μπορούν να προέλθουν από την αφηρημένη έννοια του σημαφόρου, όπως ένας σημαφόρος μέτρησης που αντιπροσωπεύει τον αριθμό των διαθέσιμων επιτρεπόμενων ταυτόχρονων συμμετεχόντων. Η διαδικασία που φτάνει στο σημείο συγχρονισμού θα ζητήσει πρώτα πρόσβαση από τον σημαφόρο και εάν το πλήθος του είναι μεγαλύτερο από μηδέν, τότε ο μετρητής μειώνεται και παρέχεται πρόσβαση, αφού η διεργασία βγει από το προστατευμένο τμήμα, σηματοδοτεί τον σημαφόρο και ο μετρητής αυξάνεται, οπότε μια διαδικασία που ενδέχεται να περιμένει στο σημείο συγχρονισμού μπορεί τώρα να λάβει πρόσβαση.

2.11.6.3 Lock

Το lock ή το mutex, που χρησιμοποιείται επίσης για την εφαρμογή αμοιβαίου αποκλεισμού, μπορεί να ταξινομηθεί ως δυαδικός σημαφόρος. Τα νήματα θα αποκτήσουν το mutex πριν εισέλθουν σε ένα κρίσιμο τμήμα και κανένα άλλο νήμα δεν θα μπορεί να εισέλθει στο τμήμα έως ότου το νήμα που απέκτησε το mutex το απελευθερώσει.

2.11.7 Lock-free-programming

2.11.7.1 Εισαγωγή

Lock-free-programming και non-blocking-algorithms αναφέρονται σε προγράμματα που αποφεύγουν εντελώς το κλείδωμα των διαδικασιών ή την αναμονή νημάτων, π.χ. αποφεύγοντας τη χρήση mutexes και σημαφόρους. Η αποφυγή των κλειδωμάτων και αναμονών οδηγεί σε αυξημένο occupancy (κατάληψη/μισθωση/εκμεταλλευση) που μπορεί να οδηγήσει σε υψηλότερη απόδοση, καθώς κάθε πόρος αξιοποιείται πλήρως ανά πάσα στιγμή αντί να σταματά την πρόοδό του.

2.11.7.2 Atomics

Οι ατομικές πράξεις, από την ελληνική λέξη άτομο που δηλώνει την ελάχιστη/αδιαίρετη μονάδα, είναι το κύριο εργαλείο του προγραμματισμού χωρίς κλειδώματα. Τα atomic instructions παρέχονται από το

hardware με το πιο αξιοσημείωτο να είναι το atomic-compare-and-swap (CAS) που μπορεί να χρησιμοποιηθεί για την υλοποίηση οποιασδήποτε άλλης ατομικής λειτουργίας π.χ. atomic-add , atomic-max κ.λπ. Μια πράξη θεωρείται ατομική εάν πραγαμοποιείτε χωρίς διακοπή, διασφαλίζοντας ότι ολοκληρώνεται ως ενιαία, αδιαίρετη ενότητα. είτε πετυχαίνει ή αποτυγχάνει χωρίς απροσδιόριστες παρενέργειες (π.χ. data races), εάν η λειτουργία αποτύχει, η διαδικασία ειδοποιείται και μπορεί να συνεχίσει να προσπαθεί μέχρι να πετύχει.

2.11.7.3 RMW

Read-modify-write (RMW), είναι μια αξιοσημείωτη κατηγορία ατομικών πράξεων που εκτελεί ανάγνωση και εγγραφή μιας διεύθυνσης μνήμης ταυτόχρονα είτε με μια παρεχόμενη τιμή είτε μια συνάρτηση της παλιάς τιμής, συνήθως επιστρέφει την παλιά τιμή ή true/false ώστε ο χρήστης να μπορεί να ειδοποιηθεί εάν η λειτουργία πέτυχε ή απέτυχε, δύο από τις πιο κοινές οδηγίες RMW είναι το CAS και το fetch-and-add.

2.11.7.4 Atomic CAS

Η ατομική σύγκριση και ανταλλαγή όπως αναφέρθηκε είναι μια από τις πιο σημαντικές πράξεις και υλοποιεί την παρακάτω λογική. Δεδομένης μιας τιμής για σύγκριση, μιας τιμής ενημέρωσης και μιας

Αλγόριθμος 1 Atomic Compare-And-Swap (CAS)

Require: Address A, Expected Value E, New Value N
1: $V \leftarrow$ Value at address A
2: **if** $V = E$ **then**
3: Value at address $A \leftarrow N$
4: **return true**
5: **else**
6: **return false**
7: **end if**

διεύθυνσης μνήμης, θα διαβάσει τη διεύθυνση μνήμης, θα συγκρίνει την τιμή που διάβασε με την παρεχόμενη τιμή, εάν οι δύο τιμές είναι ίσες, τότε θα ενημερώσει τη διεύθυνση μνήμης με την παρεχόμενη ενημέρωση, θα επιστρέψει είτε παλιά τιμή ή true/false για ειδοποίηση του χρήστη. Όπως αναφέρθηκε προηγουμένως, το CAS μπορεί να χρησιμοποιηθεί για την υλοποίηση οποιασδήποτε ατομικής πράξης απλά διαβάζοντας την τιμή και χρησιμοποιώντας την ως τιμή για σύγκριση, εκτελώντας τη πράξη στην αντιγραμμένη τιμή και στη συνέχεια χρησιμοποιώντας το αποτέλεσμα ως τιμή ενημέρωσης, ο αλγόριθμος μπορεί προαιρετικά να συνεχίσει να προσπαθεί σε έναν βρόχο που λαμβάνει μια νέα τιμή για σύγκριση κάθε φορά. Πράξεις όπως το fetch-and-add που επιχειρεί να αυξήσει την τιμή στη διεύθυνση κατά μιας τιμή ενημέρωσης και επιστρέφει την παλιά τιμή, μπορούν εύκολα να υλοποιηθούν χρησιμοποιώντας τον παραπάνω αλγόριθμο.

Αλγόριθμος 2 Fetch-And-Add using Atomic Compare-And-Swap (CAS)

Require: Address A, Increment I

1: **while true do**
2: $V \leftarrow$ Value at address A {Read current value}
3: $N \leftarrow V + I$ {Compute new value and attempt update}
4: **if** CAS(A, V, N) **then**
5: **return** V {Return the original value}
6: **end if**
7: **end while**

2.12 AI accelerators

2.12.1 Εισαγωγή

Ο αυξανόμενος φόρτος εργασίας AI και ML οδήγησε στη δημιουργία εξειδικευμένου hardware σχεδιασμένου να αντιμετωπίσει τους υπολογιστικούς περιορισμούς των παραδοσιακών επεξεργαστών. Οι επιταχυντές τεχνητής νοημοσύνης είναι στοιχεία hardware που έχουν σχεδιαστεί για να βελτιστοποιούν την απόδοση των εργασιών βαθιάς μάθησης επιταχύνοντας λειτουργίες όπως συνελίξεις και πολλαπλασιασμούς πινάκων. Παρόμοια με τον τρόπο με τον οποίο οι διανυσματικοί επεξεργαστές ενίσχυσαν την απόδοση λειτουργώντας σε 1D διανύσματα δεδομένων, οι επιταχυντές τεχνητής νοημοσύνης συνήθως περιλαμβάνουν οδηγίες που λειτουργούν σε δισδιάστατους πίνακες δεδομένων. Μια άλλη κοινή τάση που φαίνεται να ακολουθούν αυτοί οι επιταχυντές είναι η βελτιστοποίηση για υπολογισμούς μικτής ή μειωμένης ακρίβειας.

2.12.2 TPU

Τα Tensor Processing Units (TPU) είναι επιταχυντές τεχνητής νοημοσύνης που έχουν σχεδιαστεί από την Google ειδικά για την επιτάχυνση του φόρτου εργασίας μηχανικής μάθησης [65]. Είναι προσαρμοσμένα για χρήση σε εργασίες βαθιάς μάθησης νευρωνικών δικτύων. Εκμεταλλεύονται τεχνικές γραμμικής άλγεβρας χαμηλής ακρίβειας όπως τη μορφή κινητής υποδιαστολής bfloat16 για να ξεπεράσουν την πρόκληση των λειτουργιών που περιορίζονται από το εύρος ζώνης μνήμης επιτυγχάνοντας υψηλή απόδοση και είναι διαθέσιμα στο Cloud.

2.12.3 Tensor cores

Τα tensor cores είναι εξειδικευμένα στοιχεία hardware εντός των GPU της NVIDIA που έχουν σχεδιαστεί για να επιταχύνουν τους μαθηματικούς υπολογισμούς νευρωνικών δικτύων. Εισήχθησαν στην μικροαρχιτεκτονική volta και στοχεύουν να επιταχύνουν λειτουργίες που είναι θεμελιώδεις για τη βαθιά μάθηση, όπως οι πολλαπλασιασμοί πινάκων και οι συνελίξεις [79]. Είναι βελτιστοποιημένα για μαθηματικές πράξεις μειωμένης και μικτής ακρίβειας.

2.12.4 Matrix cores & AMX

Παρόμοια στοιχεία hardware με τα tensor cores συμπεριλήφθηκαν σε GPU της AMD με εισαγωγή της μικροαρχιτεκτονικής CDNA με το όνομα matrix cores [80]. Η Intel πρόσθεσε επίσης υλικό ειδικό για

υπολογισμούς πινάκων στη σειρά επεξεργαστών Sapphire Rapids. Εισήγαγε δισδιάστατους καταχωρητές ως επέκταση του x86 ISA με τα AMX (Advanced Matrix eXtensions) που υποστηρίζουν επίσης μορφές μειωμένης ακρίβειας όπως το bfloat16 [81].

2.13 Google/Highway

Η Highway είναι ένα έργο που στοχεύει να κάνει τη διανυσματοποίηση (SIMD) φορητή σε διαφορετικές συσκευές. Είναι μια αφαίρεση γραμμένη σε C++ που καλύπτει ένα ευρύ φάσμα σύγχρονων επεξεργαστών και εκθέτει δομές SIMD κάτω από την ίδια διεπαφή χωρίς να θυσιάζει την απόδοση [82].

2.14 XLA

Το XLA είναι μια συντομογραφία του Accelerated Linear Algebra και είναι ένας μεταγλωττιστής μηχανικής μάθησης για πολλαπλές συσκευές και επιταχυντές μηχανικής μάθησης (όπως TPU, GPU, CPU) που χρησιμοποιείται από τα περισσότερα σύγχρονα μεγάλα πλαίσια βαθιάς μάθησης [83].

2.15 ONNX

Το Open Neural Network Exchange (ONNX) είναι ένα format ανοιχτού κώδικα που έχει σχεδιαστεί για να διευκολύνει τη διαλειτουργικότητα μεταξύ διαφορετικών πλαισίων μηχανικής μάθησης [84]. Το ONNX παρέχει μια τυποποιημένη αναπαράσταση μοντέλων, επιτρέποντας απρόσκοπτες μεταβάσεις μεταξύ περιβαλλόντων εκπαίδευσης και εξαγωγής συμπερασμάτων. Το ONNX ορίζει ένα επεκτάσιμο υπολογιστικό μοντέλο γραφήματος που αναπαριστά μοντέλα μηχανικής μάθησης μέσω ενός κατευθυνόμενου ακυκλικού γραφήματος (DAG) λειτουργιών ή «κόμβων», που ο καθένας εκτελεί έναν συγκεκριμένο υπολογισμό. Η μορφή υποστηρίζει μια ποικιλία τελεστών, συμπεριλαμβανομένων αυτών για συνελικτικά νευρωνικά δίκτυα (CNN), επαναλαμβανόμενα νευρωνικά δίκτυα (RNN) και αρχιτεκτονικές transformer. Ένα από τα βασικά πλεονεκτήματα του ONNX είναι η συμβατότητά του με πολλαπλά πλαίσια μηχανικής μάθησης, επιτρέποντας στους χρήστες να εκπαιδεύουν μοντέλα σε ένα περιβάλλον και να τα αναπτύσσουν σε άλλο.

2.16 C++

Η C++, μια επέκταση γλώσσας προγραμματισμού που δημιουργήθηκε από τον Bjarne Stroustrup στις αρχές της δεκαετίας του 1980, συνδυάζει παραδείγματα διαδικαστικού, αντικειμενοστραφούς και συναρτησιακού προγραμματισμού καθιστώντας την κατάλληλη για μια ποικιλία εφαρμογών.

Η ανάπτυξη εφαρμογών όπου η απόδοση είναι κρίσιμη μπορεί να ωφεληθεί πολύ από την άμεση πρόσβαση σε πόρους hardware και τις δυνατότητες χαμηλού επιπέδου χειρισμού δεδομένων. Χρησιμοποιείται σε πολλούς διαφορετικούς τομείς, όπως ενσωματωμένα συστήματα, ο προγραμματισμός συστημάτων, η ανάπτυξη παιχνιδιών, scientific computing και άλλα πολλά.

Η C++ περιλαμβάνει μεταπρογραμματισμό προτύπων που επιτρέπει στους χρήστες να γράφουν γενικές δομές δεδομένων για οποιονδήποτε τύπο δεδομένων χωρίς επιβάρυνση χρόνου εκτέλεσης, ενώ επίσης

επιτρέπει την παραλλαγή συμπεριφορών για εξειδικευμένους τύπους μέσω αντιπαραβολής προτύπων. Επιπλέον δίνει τη δυνατότητα εγγραφής κώδικα που θα εκτελεστεί σε χρόνο μεταγλώττισης προσφέροντας νέες ευκαιρίες βελτιστοποίησης.

Τέλος, η C++ είναι συμβατή με αντικειμενοστραφή παραδείγματα προγραμματισμού, τα οποία επιτρέπουν τη χρήση κλάσεων, κληρονομικότητας, πολυμορφισμού και ενθυλάκωσης για τη δημιουργία επαναχρησιμοποιήσιμων προγραμμάτων.

2.17 BLAS

Το BLAS είναι συντομογραφία του Basic Linear Algebra Subprograms. Είναι μια προδιαγραφή που ορίζει ένα σύνολο πράξεων γραμμικής άλγεβρας χωρισμένες σε 3 επίπεδα με σειρά αυξανόμενης χρονικής πολυπλοκότητας [1]. Αυτά τα τρία επίπεδα είναι:

- Επίπεδο 1: Αλγόριθμοι που εκτελούνται σε γραμμική πολυπλοκότητα $O(N)$ όπως εσωτερικά γινόμενα διανυσμάτων:
$$\langle x, y \rangle = x_j y^j \tag{2.70}$$
- Επίπεδο 2: Αλγόριθμοι που εκτελούνται σε τετραγωνική πολυπλοκότητα $O(N^2)$ όπως γινόμενα διανυσμάτων μήτρας:
$$v^i \leftarrow \alpha A^i{}_j x^j + \beta v^i \tag{2.71}$$
- Επίπεδο 3: Αλγόριθμοι που εκτελούνται σε κυβική πολυπλοκότητα $O(N^3)$ όπως πολλαπλασιασμός πινάκων :
$$C^i{}_k \leftarrow \alpha A^i{}_j B^j{}_k + \beta C^i{}_k \tag{2.72}$$

Υπάρχουν διάφορες υλοποιήσεις ανοιχτού και κλειστού κώδικα για το BLAS. Αυτές οι βιβλιοθήκες είναι συνήθως βελτιστοποιημένες για ένα ευρύ φάσμα σύγχρονων επεξεργαστών και χρησιμοποιούν αλγόριθμους με επιδόσεις αιχμής όπως ο αλγόριθμος GotoBLAS για πολλαπλασιασμό πινάκων [85].

2.18 LAPACK

Το LAPACK είναι μια συντομογραφία του Linear Algebra Package και είναι μια βιβλιοθήκη που ορίζει διάφορους επιλυτές γραμμικών συστημάτων και παραγοντοποιήσεις πινάκων χρησιμοποιώντας μεθόδους αριθμητικής ανάλυσης για απόδοση τελευταίας τεχνολογίας [86]. Η αρχική βιβλιοθήκη γράφτηκε σε Fortran, αλλά ports της μπορούν να βρεθούν στη C και σε άλλες γλώσσες. Αυτές οι ρουτίνες συνήθως υλοποιούνται και παρέχονται μαζί με BLAS.

2.19 CUDA

2.19.1 Εισαγωγή

Compute Unified Device Architecture (CUDA) είναι μια πλατφόρμα και ένα μοντέλο παράλληλου προγραμματισμού που αναπτύχθηκε από την NVIDIA. Η CUDA είναι επίσης μια πλατφόρμα ετερογενών

υπολογιστικών συστημάτων, καθώς επιτρέπει την αποτελεσματική αλληλεπίδραση μεταξύ του κεντρικού υπολογιστή (host που σημαίνει την CPU) και της συσκευής (device που σημαίνει GPU). Παρέχει μια διεπαφή προγραμματισμού που επεκτείνει γνωστές γλώσσες όπως C, C++ και Fortran με εξειδικευμένες δομές για την ανάπτυξη προγραμμάτων GPU [87].

Κύριο στοιχείο της CUDA είναι ο πυρήνας (kernel), ο οποίος είναι μια συνάρτηση που θα μεταγλωττιστεί για παράλληλη εκτέλεση στη GPU, αλλά είναι προσβάσιμη για κλήση από οποιαδήποτε συσκευή (ακόμη και ένας πυρήνας μπορεί να καλέσει έναν άλλο πυρήνα). Το μοντέλο προγραμματισμού cuda χρησιμοποιεί μια ιεραρχία από νήματα που είναι οργανωμένα σε μπλοκ και μπλοκ που είναι οργανωμένα στο πλέγμα για να μοντελοποιηθεί η παραλληλοποίηση για διαφορετικές αρχιτεκτονικές.

- Thread: Νήμα εκτέλεσης που θα εκτελέσει ολόκληρο τον πυρήνα, κάθε νήμα σχετίζεται με ένα μοναδικό αναγνωριστικό που είναι οι συντεταγμένες (xyz) στο μπλοκ και είναι προσβάσιμες μέσω της καθολικής μεταβλητής threadIdx.

- Warp: Ένα στημόνι (προέρχεται από τον όρο ύφανσης) είναι μια ομάδα νημάτων (συνήθως 32 αλλά μπορεί να είναι 64 σε ορισμένες GPU της AMD) που έχουν διαδοχικό αναγνωριστικό νήματος. Όλα τα νήματα σε ένα στημόνι εκτελούν την ίδια εντολή σε στυλ SIMD, ωστόσο οι αρχιτέκτονες της NVIDIA αποφεύγουν να χρησιμοποιούν τον όρο SIMD και αντ' αυτού επέκτειναν την κατηγοριοποίηση του Flynn εφευρίσκοντας τον όρο SIMT (single instruction multiple threads).

- Block: Το μπλοκ είναι μια ομάδα νημάτων (λιγότερα από ή ίσα με 1024 ή εναλλακτικά 32 warps) τα οποία εκτελούνται είτε σειριακά είτε παράλληλα προγραμματισμένα από έναν πολυεπεξεργαστή ροής (streaming multiprocessor ή SM). Κάθε μπλοκ γνωρίζει τη θέση του στο πλέγμα (xyz). Οι συντεταγμένες είναι προσβάσιμες μέσω της μεταβλητής blockIdx, κατ' επέκταση κάθε νήμα γνωρίζει επίσης τη θέση του στο πλέγμα συνδυάζοντας τη θέση του στο μπλοκ με τη θέση του μπλοκ στο πλέγμα.

- Grid: Πλέγμα ορίζεται ως μια ομάδα μπλοκ, ο αριθμός των μπλοκ στο πλέγμα συνήθως εξαρτάται από το μέγεθος των δεδομένων με το μέγιστο αριθμό των μπλοκ να είναι κάτι κοντά στα $x = 2^{31},\ y = 65535,\ z = 65535$.

- Stream: Μια ροή είναι σαν μια ουρά εντολών που θα εκτελέσει όλες τις εργασίες GPU όπως οι πυρήνες που έχουν προγραμματιστεί στη ροή σειριακά. Ωστόσο, διαφορετικές ροές μπορεί να εκτελούνται παράλληλα ή με παρεμβολή, οδηγώντας σε πιθανή αυξημένη παραλληλοποίηση εάν χρησιμοποιηθούν σωστά. Η CUDA προσφέρει ένα API για τη δημιουργία, τη χρήση και το συγχρονισμό ροών.

Ένας πυρήνας για να εκτελεστεί, πρέπει να διαμορφωθεί με 4 παραμέτρους με τις δύο τελευταίες να είναι προαιρετικές. Η πρώτη παράμετρος είναι μια τρισδιάστατη δομή που αντιπροσωπεύει τον αριθμό μπλοκ στο πλέγμα, η δεύτερη παράμετρος είναι και πάλι μια τρισδιάστατη δομή που είναι ο αριθμός των νημάτων στο μπλοκ, η τρίτη είναι το μέγεθος της κοινόχρηστης μνήμης (προεπιλεγμένη τιμή 0) και τέλος ή ροή στην οποία έχει προγραμματιστεί να εκτελεστεί ο πυρήνας με προεπιλεγμένη τιμή το 0 ή null δείκτη που υποδηλώνει την προεπιλεγμένη ροή του συστήματος.

2.19.2 Memory model

2.19.2.1 Εισαγωγή

Η ιεραρχία της μνήμης CUDA χωρίζεται σε κατηγορίες με κάθε τύπο μνήμης να προσφέρει διαφορετικά επίπεδα λανθάνοντος χρόνου, διαφορετική διεκπεραιωτική ικανότητα και διαφορετική εμβέλεια στο πρόγραμμα [88]. Σε αυτήν την υποενότητα θα εξετάσουμε ορισμένους σημαντικούς τύπους μνήμης και διάφορες καταστάσεις που μπορούν να υποβαθμίσουν ή να αυξήσουν την απόδοση.

2.19.2.2 Global memory

Η διαχείριση της παγκόσμιας μνήμης γίνεται παρόμοια με την κανονική μνήμη στην C, ο κεντρικός υπολογιστής εκχωρεί μια συγκεκριμένη ποσότητα byte και στη συνέχεια πρέπει να την ελευθερώσει μετά τη χρήση. Βρίσκεται στη συσκευή που σημαίνει ότι έχει υψηλό λανθάνων χρόνο. Η διάρκεια ζωής της εκχωρημένης καθολικής μνήμης τελειώνει μόνο μετά την απελευθέρωσή της, επομένως μπορεί να χρησιμοποιηθεί σε διαφορετικούς πυρήνες. Αυτό σημαίνει ότι η πρόσβαση σε αυτήν την μνήμη πρέπει να γίνεται με προσοχή, καθώς μπορεί να τροποποιηθεί από οποιοδήποτε μέρος του προγράμματος ανά πάσα στιγμή.

2.19.2.3 Local memory

Η τοπική μνήμη είναι ιδιωτική σε κάθε νήμα και η εκχώρηση της γίνεται από το driver ανάλογα με τη χρήση των τοπικών μεταβλητών στον πυρήνα. Για παράδειγμα, αν χρησιμοποιήσουμε έναν πολύ μεγάλο στατικό πίνακα στον πυρήνα, αυτός συνήθως θα εκχωρείται στην τοπική μνήμη. Είναι μνήμη στη συσκευή που σημαίνει ότι αναμένεται να έχει υψηλό λανθάνων χρόνο.

2.19.2.4 Registers

Οι καταχωρητές είναι μνήμη στο τσιπ και είναι ιδιωτικοί σε κάθε νήμα, αναμένεται να έχουν τον χαμηλότερο λανθάνων χρόνον και την μεγαλύτερη απόδοση, αλλά είναι περιορισμένοι σε αριθμό. Η CUDA επιτρέπει στο χρήστη να υποδείξει έναν μέγιστο αριθμό καταχωρητών, αυτό φυσικά μπορεί να επηρεάσει την απόδοση. Εάν οι τοπικές μεταβλητές στον πυρήνα δεν μπορούν να χωρέσουν σε καταχωρητές τότε η πλεονάζουσα μνήμη ”χυθεί” (register spilling) στην τοπική μνήμη προκαλώντας καθυστερήσεις κατά την πρόσβαση σε αυτήν. Ωστόσο, η χρήση υπερβολικού αριθμού καταχωρητών μειώνει τον αριθμό των νημάτων που μπορούν να εκτελούνται παράλληλα. Η βέλτιστη χρήση μνήμης καταχωρητών περιλαμβάνει την επίτευξη ισορροπίας μεταξύ της διατήρησης υψηλής πληρότητας (που σημαίνει ότι υπάρχουν πολλά ενεργά νήματα) και της διασφάλισης ότι κάθε νήμα έχει επαρκείς καταχωρητές για την αποφυγή υπερβολικής διαρροής στην τοπική μνήμη. Η σωστή ισορροπία εξαρτάται επίσης από την αρχιτεκτονική της GPU.

2.19.2.5 Shared memory

Η κοινόχρηστη μνήμη είναι σαν μια κρυφή μνήμη που διαχειρίζεται το λογισμικό αντί του hardware. Είναι μνήμη στο τσιπ που σημαίνει χαμηλό λανθάνων χρόνο. Η διεκπεραιωτική ικανότητα της κοινόχρηστης μνήμης εξαρτάται από τα σωστά μοτίβα πρόσβασης στη μνήμη καθώς μπορεί να υπάρχει υψηλός ανταγωνισμός μεταξύ των νημάτων που μπορεί να προκαλέσει την εκτέλεση των αιτημάτων σειριακά.

Η κοινόχρηστη μνήμη μοιράζεται μεταξύ νημάτων σε ένα μπλοκ αλλά όχι μεταξύ μπλοκ που σημαίνει ότι έχει το εμβέλεια και διάρκεια ζωής ενός μπλοκ. Μπορεί να εκχωρηθεί είτε στατικά στον πυρήνα είτε δυναμικά στη διαμόρφωση του πυρήνα με το μέγιστο μέγεθος να είναι μερικά kilobyte. Χρησιμοποιείται συνήθως για επικοινωνία και ανταλλαγή δεδομένων μεταξύ νημάτων.

2.19.2.6 Bank conflicts

Η κοινόχρηστη μνήμη είναι οργανωμένη σε τράπεζες μνήμης και η τράπεζα οργανώνει τη μνήμη της σε λέξεις με κάθε λέξη να είναι 4 byte. Κάθε διαδοχική λέξη στην κοινόχρηστη μνήμη ανατίθεται σε διαφορετική τράπεζα. Ορισμένα μοτίβα πρόσβασης στη μνήμη μπορούν να βελτιώσουν την απόδοση ενώ άλλα μπορούν να την υποβαθμίσουν.

Εάν όλα τα νήματα σε ένα στημόνι ζητούν την ίδια τιμή από μια τράπεζα, τότε αυτή η τιμή διαβάζεται μία φορά και η μετάδοση (broadcast) σε όλα τα νήματα, πράγμα που είναι γρήγορο. Εάν όλα τα νήματα σε ένα warp ζητούν διαδοχικές λέξεις από κοινή μνήμη, που σημαίνει από διαδοχικές τράπεζες (π.χ. μέσω ευρετηρίασης χρησιμοποιώντας το αναγνωριστικό τους), τότε δεν υπάρχει σύγκρουση και η ανάγνωση είναι πάλι γρήγορη. Στις περισσότερες σύγχρονες αρχιτεκτονικές, εάν ορισμένα νήματα σε ένα warp ζητούν την ίδια τιμή, τότε η τιμή διαβάζεται μία φορά και εκτελείται μια πολυεκπομπή (multicast) που είναι παρόμοια με το broadcast και εξακολουθεί να είναι γρήγορη, αν και ελαφρώς πιο αργή. Μια σύγκρουση τράπεζας προκύπτει όταν διαφορετικά νήματα σε ένα warp ζητούν διαφορετικές τιμές από την ίδια τράπεζα, στη συνέχεια τα αιτήματα πρέπει να σειριοποιηθούν πλήρως και αυτό οδηγεί σε υποβάθμιση της απόδοσης

2.19.2.7 Constant memory

Η σταθερή μνήμη εκχωρείται και γράφεται μόνο από τον κεντρικό υπολογιστή και τα προγράμματα GPU επιτρέπεται μόνο να τη διαβάζουν. Περιορίζεται σε μερικά kilobyte και παρόλο που είναι αποθηκευμένη στη συσκευή, θα τοποθετηθεί σε μια ειδική προσωρινή μνήμη στο τσιπ που είναι μόνο για ανάγνωση. Δεδομένου ότι η προαναφερθείσα κρυφή μνήμη μοιράζεται μεταξύ των νημάτων σε έναν πολυεπεξεργαστή ροής, οι προσβάσεις είναι αποτελεσματικές όταν όλα τα νήματα ζητούν την ίδια σταθερή τιμή, αξιοποιώντας τον μηχανισμό μετάδοσης. Ωστόσο, εάν τα threads σε ένα warp ζητήσουν διαφορετικές τιμές σταθερής μνήμης, μπορεί να υποβαθμίσουν την απόδοση, καθώς οδηγούν στη σειριακή εκτέλεση των αιτημάτων μνήμης. Τέλος, η σταθερή μνήμη είναι συνήθως μια καθολική μεταβλητή που κάνει τη διάρκεια ζωής της ίση με τη διάρκεια ζωής της εφαρμογής

2.19.2.8 Memory coalescing

Μια συγχωνευμένη συναλλαγή (coalesced memory transaction) εμφανίζεται όταν διαδοχικά νήματα ζητούν διαδοχική και ευθυγραμμισμένη παγκόσμια μνήμη. Αυτό έχει ως αποτέλεσμα οι συναλλαγές μνήμης να συγχωνεύονται σε όσο το δυνατόν λιγότερες, αυξάνοντας την απόδοση.

2.19.2.9 Read-only cache

Τα δεδομένα που ζητούνται από την καθολική μνήμη μπορούν να τοποθετηθούν σε μια γρήγορη κρυφή

μνήμη μόνο για ανάγνωση στο τσιπ, εάν ο μεταγλωττιστής εντοπίσει ότι δεν έχουν τροποποιηθεί κατά τη διάρκεια της εκτέλεσης του πυρήνα. Αυτό δεν είναι τόσο απλό όσο η επισήμανση ενός δείκτη με `const` καθώς μπορεί να αναφέρεται στην ίδια διεύθυνση με έναν άλλο δείκτη από τον οποίο εκτελούμε εγγραφές, επομένως ο μεταγλωττιστής δεν μπορεί να εγγυηθεί ή να αποδείξει ότι τα δεδομένα δεν έχουν τροποποιηθεί. Για να αντιμετωπιστεί αυτό, ο χρήστης μπορεί να καθορίσει ότι δύο δείκτες δεν αναφέρονται σε περιοχές μνήμης που επικαλύπτονται με το `__restrict__`. Εναλλακτικά, μπορούν επίσης να γράψουν την εντολή που θα έκανε μια ανάγνωση με αποθήκευση στη read-only cache είτε χρησιμοποιώντας γλώσσα assembly είτε με τη συνάρτηση `__ldg`.

2.19.3 Synchronization

Η CUDA παρέχει πολλαπλούς μηχανισμούς συγχρονισμού, καθώς είναι ζωτικής σημασίας για τη διασφάλιση της σωστής εκτέλεσης και της συνέπειας των δεδομένων στον παράλληλο υπολογισμό.

Για τον συγχρονισμό όλων των νημάτων μέσα σε ένα μπλοκ που σημαίνει ότι απαιτείται να φτάσουν όλα τα νήματα σε ένα ορισμένο σημείο πριν μπορέσει να προχωρήσει οποιοδήποτε υπάρχει η συνάρτηση `syncthreads`.

Ο συγχρονισμός warp είναι παρόμοιος με τον προγραμματισμό με λωρίδες SIMD. Προκειμένου η CUDA να επιτρέπει τη ροή ελέγχου και τη διακλάδωση χρησιμοποιεί masking, που σημαίνει ότι ανάλογα με την ενεργή μάσκα ορισμένα νήματα θα είναι ανενεργά σε έναν κλάδο. Η χρήση και η ανάγνωση αυτής της μάσκας είναι διαθέσιμη στον χρήστη για να συγχρονίσει νήματα με μια συγκεκριμένη μάσκα, καθώς και διαφορετικές λειτουργίες που μοιάζουν με SIMD (π.χ. ανακάτεμα).

Για πιο καθολικό εύρος, η CUDA προσφέρει επίσης ατομικές συναρτήσεις. Τα Atomics μπορούν να χρησιμοποιηθούν για την προστασία της ταυτόχρονης πρόσβασης από μόνα τους, αλλά μπορούν να χρησιμοποιηθούν για την υλοποίηση συγχρονισμού υψηλότερου επιπέδου όπως τα spinlocks.

Από την πλευρά του κεντρικού υπολογιστή, ο χρήστης έχει τη δυνατότητα συγχρονισμού μεταξύ ροών και συσκευών. Υπάρχουν λειτουργίες που μπλοκάρουν το νήμα τον καλούντα έως ότου μια ροή CUDA ή ολόκληρη η συσκευή ολοκληρώσει όλες τις εργασίες της, συμπεριλαμβανομένων των κλήσεων πυρήνα και των λειτουργιών μνήμης. Αυτό είναι χρήσιμο για παράδειγμα κατά τη μεταφορά δεδομένων μεταξύ συσκευών, επειδή ορισμένες λειτουργίες CUDA, όπως οι κλήσεις πυρήνα, είναι ασύγχρονες.

2.19.4 Benchmarking

Η συγκριτική αξιολόγηση είναι κρίσιμη για την ανάπτυξη και τη βελτιστοποίηση εφαρμογών υψηλής απόδοσης. Σε CUDA είναι οι χρόνοι του ρολογιού μπορούν να μετρώνται με τρόπο C χρησιμοποιώντας τη συνάρτηση `clock()` μέσα στον πυρήνα και στη συνέχεια αποθήκευση των αποτελεσμάτων σε καθολική μνήμη, ώστε να να μετακινηθούν στη μνήμη του κεντρικού υπολογιστή. Ωστόσο, η CUDA προσφέρει καλύτερες λύσεις για τη αξιολόγηση επιδόσεων προγραμμάτων GPU.

Τα συμβάντα CUDA (CUDA Events) χρησιμοποιούνται για την ακριβή μέτρηση του χρόνου που απαιτείται για συγκεκριμένα τμήματα της εκτέλεσης κώδικα GPU. Τοποθετώντας συμβάντα πριν και μετά την εκκίνηση του πυρήνα ή τις λειτουργίες μνήμης, είναι δυνατό να μετρηθεί ο χρόνος που έχει πα-

ρέλθει με υψηλή ακρίβεια. Υπάρχει ένα API με λειτουργίες που χρησιμοποιούνται για την καταγραφή συμβάντων και τον υπολογισμό των διαφορών χρόνου. Αυτή η προσέγγιση παρέχει μια λεπτομερή άποψη της απόδοσης σε επίπεδο πυρήνα με έναν εύκολο και μη παρεμβατικό τρόπο μέτρησης εντός του προγράμματος.

Το nvprof είναι ένας profiler γραμμής εντολών που αναπτύχθηκε από τη NVIDIA και προσφέρει λεπτομερείς πληροφορίες για τα χαρακτηριστικά απόδοσης των εφαρμογών CUDA. Συλλέγει μετρήσεις όπως χρόνους εκτέλεσης πυρήνα, χρήση μνήμης και χρήση hardware. Μπορεί να εκτελεστεί απευθείας από τη γραμμή εντολών, παρέχοντας μια σύνοψη υψηλού επιπέδου της απόδοσης μιας εφαρμογής, συμπεριλαμβανομένων συγκεκριμένων μετρητών hardware και μετρικών, ακόμη και κλήσεων πυρήνα με τα ονόματά τους, καθιστώντας το ένα εξαιρετικό εργαλείο εντοπισμού σφαλμάτων. Αυτές οι δυνατότητες δίνουν μια βαθύτερη κατανόηση του τρόπου με τον οποίο οι εφαρμογές χρησιμοποιούν τη GPU και μπορούν να εντοπίσουν πιθανά σημεία συμφόρησης στην απόδοση.

2.19.5 NVRTC

Το NVRTC (NVIDIA Runtime Compilation) είναι μια ιδιόκτητη βιβλιοθήκη CUDA της NVIDIA που επιτρέπει τη μεταγλώττιση κώδικα CUDA C++ σε χρόνο εκτέλεσης (JIT). Σε αντίθεση με την παραδοσιακή μεταγλώττιση CUDA, η οποία πραγματοποιείται πριν από την εκτέλεση, το NVRTC επιτρέπει στα προγράμματα να μεταγλωττίζουν και να εκτελούν δυναμικά πυρήνες CUDA κατά τη διάρκεια του χρόνου εκτέλεσης, παρέχοντας ευελιξία για εφαρμογές που απαιτούν δυναμική δημιουργία πυρήνα.

2.19.6 cuSOLVER & cuBLAS

Το cuSOLVER είναι βιβλιοθήκη που μοιάζει με LAPACK της NVIDIA για την επίλυση γραμμικών συστημάτων και παραγοντοποιήσεις πινάκων στη GPU. Ενώ το cuBLAS είναι μια υλοποίηση NVIDIA της προδιαγραφής BLAS για γραμμική άλγεβρα σε GPU. Είναι και τα δύο, ιδιόκτητα και κλειστού κώδικα.

2.19.7 cuDNN

Το cuDNN είναι μια βιβλιοθήκη πρωτόγονων μπλοκ νευρωνικών δικτύων που υλοποιεί διάφορες κοινές λειτουργίες νευρωνικών δικτύων, όπως συνέλιξεις, attention, pooling κ.α, με κορυφαίες επιδόσεις [89]. Εκτός από το C API, η NVIDIA παρέχει επίσης μια διεπαφή σε C++ με το όνομα cudnn_frontend που έχει σκοπό να απλοποιήσει τη χρήση του χρησιμοποιώντας τις δυνατότητες υψηλού επιπέδου της C++, το C++ API είναι ανοιχτού κώδικα και είναι επίσης διαθέσιμο στην Python μέσω του pybind [90].

2.19.8 CUB

Το NVIDIA CUDA Unbound (CUB) είναι μια βιβλιοθήκη ανοιχτού κώδικα C++ υψηλής απόδοσης, μέρος των NVIDIA CUDA core compute libraries (CCCL) [91]. Παρέχει βελτιστοποιημένες παράλληλες μαθηματικές συναρτήσεις για προγραμματιστές CUDA. Το CUB έχει σχεδιαστεί για να απλοποιεί την υλοποίηση κοινών παράλληλων αλγορίθμων, προσφέροντας εξαιρετικά βελτιστοποιημένες και με επίγνωση της αρχιτεκτονικής συναρτήσεις που αξιοποιούν τον παραλληλισμό σε επίπεδο παραμόρφωσης, μπλοκ και σε επίπεδο συσκευής.

2.19.9 Thrust

Το Thrust είναι μια βιβλιοθήκη προτύπων C++ που έχει σχεδιαστεί για να παρέχει μια υψηλού επιπέδου, διεπαφή σαν τυπική βιβλιοθήκη για προγραμματισμό GPU. Είναι επίσης μέρος των NVIDIA CUDA core compute libraries [91]. Χτισμένο σε CUDA, το Thrust απλοποιεί την ανάπτυξη παράλληλων εφαρμογών προσφέροντας μια συλλογή επαναχρησιμοποιήσιμων, εξαιρετικά βελτιστοποιημένων αλγορίθμων και δομών δεδομένων. Ο σχεδιασμός του είναι εμπνευσμένος από την C++ τυπική βιβλιοθήκη προτύπων, καθιστώντας το εύχρηστο για χρήστες που είναι εξοικειωμένοι με την C++.

2.19.10 Jitify

Το jitify είναι ένα έργο ανοιχτού κώδικα που αποτελείται από ένα αρχείο header C++ [92]. Αξιοποιώντας τις δυνατότητες υψηλού επιπέδου της C++, παρέχει μια διεπαφή που έχει σκοπό να απλοποιήσει τη διαδικασία χρήσης του NVRTC και τη σύνδεση με άλλα εκτελέσιμα στο χρόνο εκτέλεσης. Το jitify παρέχει επίσης caching και επαναχρησιμοποίηση πυρήνων που είχαν μεταγλωττιστεί προηγουμένως, σειριοποίηση προγραμμάτων και πυρήνων και εύκολη χρήση προτύπων C++ σε JIT μεταγλωττισμένο κώδικα.

2.19.11 HIP

Το HIP (Heterogenous-Computer Interface for Portability) είναι ένα κιτ ανάπτυξης λογισμικού που αναπτύχθηκε από την AMD [93]. Το HIP υλοποιεί τη διεπαφή CUDA, συμπεριλαμβανομένων των βιβλιοθηκών (π.χ. cuBLAS, NVRTC), καθώς στοχεύει να είναι πλήρως συμβατό με τον ήδη υπάρχοντα κώδικα CUDA, επιτρέποντας στους χρήστες να μεταφέρουν απρόσκοπτα την εργασία τους στην πλατφόρμα HIP (και πιθανώς στις GPU της AMD). Υποστηρίζει GPU NVDIA και AMD ενώ επιπλέον προσφέρει διάφορες βιβλιοθήκες υψηλής απόδοσης για εργασίες όπως η γραμμική άλγεβρα και ray tracing.

Κεφάλαιο 3ο: Σχετικές εργασίες

3.1 Εισαγωγή

Αυτό το έργο δεν είναι φυσικά το πρώτο στο είδος του, άντλησε έμπνευση από διάφορα άλλα έργα ανοιχτού κώδικα. Αυτά τα έργα είναι γνωστά εργαλεία που χρησιμοποιούνται για εργασίες επιστημονικού υπολογισμού και μηχανικής μάθησης. Η χρήση αυτών των εργαλείων έφερε ένα αίσθημα θαυμασμού και στη συνέχεια η περιέργεια οδήγησε στη μελέτη του πηγαίο κώδικα τους. Τμήματα κώδικα και τροποποιημένες λύσεις από ορισμένα από αυτά τα έργα μπορούν να βρεθούν στην παραγόμενη βιβλιοθήκη αυτής της πτυχιακής. Το παρακάτω είναι μια αναδρομή όλων αυτών των σημαντικών εργαλείων που τροφοδοτούν τη σημερινή υποδομή βαθιάς μάθησης και υπολογισμών υψηλής απόδοσης, κάνοντας εργασίες που ήταν πολύπλοκες, όπως η εκπαίδευση ενός νευρωνικού δικτύου, να φαίνονται προσιτές και ακόμη και εύκολες.

3.2 NumPy

Το NumPy είναι ένα πακέτο ανοιχτού κώδικα σε Python, ξεκίνησε την ανάπτυξή του το μακρινό 1995 και είναι ακόμα σε ενεργό ανάπτυξη και μια από τις μεγαλύτερες πλατφόρμες για scientific computing μέχρι σήμερα [2]. Παρέχει μια πολύ φιλική διεπαφή για υπολογισμούς σε πολυδιάστατους πίνακες. Χρησιμοποιεί διάφορες τεχνικές βελτιστοποίησης, όπως device agnostic SIMD και ενσωμάτωση αριθμητικών βιβλιοθηκών υψηλής απόδοσης όπως BLAS και LAPACK.

3.3 CuPy

Το CuPy είναι μια βιβλιοθήκη Python ανοιχτού κώδικα που μπορεί να χρησιμοποιηθεί μεταξύ άλλων ως ενα drop-in-replacement του NumPy για υπολογισμούς στη GPU [94]. Επιπλέον, δεδομένου ότι το CuPy λειτουργεί με μεταγλώττιση JIT, εκθέτει αυτή τη λειτουργία στον χρήστη μέσω μιας φιλικής διεπαφής και ο χρήστης μπορεί να συνενώσει πολλούς τελεστές σε έναν πυρήνα για να επιτύχει υψηλότερη απόδοση και χαμηλότερη κατανάλωση μνήμης.

3.4 Theano

Το Theano ήταν ένας από τους πρωτοπόρους των πακέτων βαθιάς μάθησης με μια βολική φιλική προς τον χρήστη διεπαφή, η ανάπτυξή του αφήνει τα ίχνη της από το 2008 [95]. Πήρε το όνομά του από την αρχαία Ελληνίδα φιλόσοφο. Ήταν open source και αναπτύχθηκε κυρίως από το Montreal Institute for Learning Algorithms (MILA) στο Université de Montréal με την ικανότητα έκφρασης μοντέλων ως μαθηματικών εκφράσεων, επανεγγραφής και βελτιστοποίησης υπολογιστικών γραφημάτων, υπολογισμών σε GPU και αυτόματης διαφόρισης υψηλότερης τάξης. Τον Σεπτέμβριο του 2017 το theano διέκοψε την ανάπτυξή του, αλλά διάφορα forks του (πχ aesara , PyTensor) βρίσκονται σε ενεργό ανάπτυξη μέχρι σήμερα.Το Theano με τη σύνταξη του που μοιάζει με NumPy συνέβαλε αναμφισβήτητα στην κουλτούρα βαθιάς μάθησης ανοιχτού κώδικα και αύξησε τον πήχη για τις διεπαφές βαθιάς μάθησης, ενώ άνοιξε τον ασκό του Αιόλου για περισσότερες πλατφόρμες βαθιάς μάθησης ανοιχτού κώδικα που θα έρθουν τα επόμενα χρόνια.

3.5 Caffe

Το Caffe ξεκίνησε την ανάπτυξή του το 2013 στο Berkeley Vision and Learning Center γραμμένο σε C++. Εφαρμόζει αυτόματη διαφόριση και παρέχει προεκπαιδευμένα μοντέλα, ενώ παράλληλα διαθέτει υπολογισμούς υψηλής απόδοσης σε CPU και GPU και έχει επίσης ενσωματωμένη υποστήριξη για CNN [96]. Το Caffe διαδραματίζει καθοριστικό ρόλο στη διάδοση ενός αλγορίθμου που ονομάστηκε "im2col" μια συντομογραφία για τον μετασχηματισμό εικόνας σε στήλη που χρησιμεύει ως τρόπος μετατροπής των λειτουργιών συνέλιξης σε γρήγορη αντιγραφή και στη συνέχεια πολλαπλασιασμό πινάκων, αξιοποιώντας τις ισχυρές βιβλιοθήκες BLAS και cuBLAS. Αυτός ο αλγόριθμος εξακολουθεί να χρησιμοποιείται μέχρι σήμερα στις περισσότερες βιβλιοθήκες που διαθέτουν CNN όπως το cuDNN και επινοήθηκε ανεξάρτητα από τον προγραμματιστή του Caffe Yangqing Jia και νωρίτερα από τους K. Chellapilla, S. Puri, P. Simard και άλλους. Αργότερα το 2017 το facebook θα ανακοίνωνε το Caffe2 μια επέκταση στο αρχικό Caffe που στη συνέχεια θα συγχωνευόταν στο PyTorch το 2018.

3.6 Tensorflow

Το 2011, όταν ο χώρος των πλατφορμών βαθιάς μάθησης έκανε τα πρώτα του πρωτόγονα βήματα, η Google Brain άρχισε να αναπτύσσει μια ιδιόκτητη πλατφόρμα βαθιάς μάθησης για εσωτερική χρήση στην έρευνα και την παραγωγή της με το όνομα DistBelief. Αυτό θα κυκλοφορούσε αργότερα ως έργο ανοιχτού κώδικα TensorFlow το 2015 και θα εξελισσόταν σε μια από τις μεγαλύτερες πλατφόρμες βαθιάς εκμάθησης του σήμερα με ευρεία υιοθέτηση και δυνατότητα εκτέλεσης σε μια ποικιλία συσκευών, συμπεριλαμβανομένων μικροεπεξεργαστών και ενσωματωμένων συστημάτων και επίσης δυνατότητα διανομής δεδομένων σε πολλαπλές συσκευές που λύνει το πρόβλημα της ύπαρξης ογκωδών συνόλων δεδομένων που δεν μπορούν να χωρέσουν σε μία συσκευή αυξάνοντας και τον βαθμό παραλληλισμού. Το Tensorflow είναι στον πυρήνα του μια device agnostic μαθηματική βιβλιοθήκη για πολυδιάστατους πίνακες και ενσωματώνει αριθμητικές βιβλιοθήκες υψηλής απόδοσης όπως CUB, cuSOLVER, cuBLAS, BLAS, LAPACK και άλλα. Ωστόσο, το tensorflow εφαρμόζει επίσης αυτόματη διαφόριση υψηλότερης τάξης για υπολογισμό gradients, επανεγγραφή και βελτιστοποίηση γραφημάτων, μεταγλώττιση JIT, σειριοποίηση μοντέλων και πολλά άλλα [14]. Το Tensorflow είναι επίσης εξαιρετικά επεκτάσιμο και υπάρχει ένα σημείο εισόδου σε κάθε επίπεδο της πλατφόρμας ξεκινώντας από τα χαμηλότερα επίπεδα όπου ο χρήστης μπορεί να γράψει μια προσαρμοσμένη λειτουργία ή πυρήνα και να την καταχωρήσει για χρήση σε python και επίσης σε υψηλότερα επίπεδα όπου ο χρήστης μπορεί να ορίσει μια συνάρτηση και την παράγωγο της για το backward pass σε python με μια εύχρηστη διεπαφή.

3.7 Tinygrad

Το tinygrad είναι ένα από τα νεότερα πλαίσια βαθιάς μάθησης που ξεκίνησε το 2020 από τον George Hotz και συντηρείται από την tinycorp [97]. Όπως και άλλα πλαίσια βαθιάς μάθησης, το tinygrad υποστηρίζει διάφορες συσκευές και αυτόματη διαφόριση για την εκπαίδευση νευρωνικών δικτύων σε μια φιλική προς το χρήστη διεπαφή Python. Στοχεύει στην εξαιρετική απλότητα και ευκολία εισαγωγής νέων επιταχυντών αφού χρησιμοποιεί μόνο 25 πρωτόγονες πράξεις και κάθε άλλη πράξη παράγεται από έναν συνδυασμό αυτών. Επιτυγχάνει υψηλή απόδοση αξιοποιώντας μια στρατηγική νωχελικής εκτέλεσης και επιθετική βελτιστοποίηση και επανεγγραφή του υπολογιστικού γραφήματος για τη συνένωση πολ-

λαπλών λειτουργιών σε όσο το δυνατόν λιγότερους πυρήνες που θα μεταγλωστούν JIT ειδικά για αυτό το μοντέλο, μειώνοντας έτσι τόσο τον χρόνο εκτέλεσης όσο και την κατανάλωση μνήμης.

3.8 JAX

Το JAX είναι μια σύγχρονη υπολογιστική βιβλιοθήκη ανοιχτού κώδικα υψηλής απόδοσης που ξεκίνησε την ανάπτυξή της από την Google [98] [99]. Υποστηρίζει κατανεμημένους υπολογισμούς σε πολλές συσκευές, συμπεριλαμβανομένων των GPU και των TPU. Εφαρμόζει μεταξύ άλλων, αυτόματη διαφόριση υψηλότερης τάξης, JIT μεταγλώττιση κανονικού κώδικα NumPy και Python με XLA και αυτόματη διανυσματοποίηση. Το JAX διαφέρει από τις υπόλοιπες πλατφόρμες ως προς την αρχιτεκτονική του καθώς ακολουθεί το παράδειγμα συναρτησιακού προγραμματισμού. Μέσω του παραδείγματος συναρτησιακού προγραμματισμού είναι σε θέση να συνθέτει απρόσκοπτα μετασχηματισμούς σε συναρτήσεις όπως η διαφόριση, η μεταγλώττιση jit και η διανυσματοποίηση. Πάνω από το JAX υπάρχει επίσης το FLAX που είναι μια φιλική προς το χρήστη διεπαφή για την ανάπτυξη και την εκπαίδευση νευρωνικών δικτύων [100].

3.9 PyTorch

Το PyTorch είναι ένα ανοιχτού κώδικα πλαίσιο βαθιάς μάθησης που αναπτύχθηκε από το ερευνητικό εργαστήριο AI του Facebook που παρέχει μια πλατφόρμα για τη δημιουργία και την εκπαίδευση νευρωνικών δικτύων [3]. Εξελίχτηκε από το Torch, μια επιστημονική υπολογιστική βιβλιοθήκη βασισμένη σε γλώσσα Lua, η οποία χρησιμοποιήθηκε ευρέως στον ακαδημαϊκό χώρο και τη βιομηχανία για έρευνα βαθιάς μάθησης. Η ανάπτυξή του ξεκίνησε το 2016 ως διάδοχος του Torch στην Python, προσφέροντας δυναμικά υπολογιστικά γραφήματα και ένα εύκολο στη χρήση API. Το 2022, το PyTorch υπέστη μια σημαντική δομική μετάβαση με το σχηματισμό του PyTorch Foundation υπό το Linux Foundation. Στο πνεύμα των μεγάλων αλλαγών, το 2018 έγινε ενοποίηση με το Caffe2, ένα άλλο πλαίσιο βαθιάς μάθησης που αναπτύχθηκε από το Facebook, το οποίο δημιουργήθηκε ως διάδοχος του αρχικού Caffe. Η συγχώνευση επέτρεψε στο PyTorch να ενσωματώσει αποτελεσματικές δυνατότητες ανάπτυξης μοντέλων από το Caffe2, διατηρώντας παράλληλα την ευκολία χρήσης και την ευελιξία που προσανατολίζεται στην έρευνα. Το PyTorch υποστηρίζει την αυτόματη διαφοροποίηση μέσω της μονάδας Autograd που διαθέτει, επιτρέποντας εύκολο backpropagation για πολύπλοκα μοντέλα. Επιπλέον, παρέχει ένα ισχυρό οικοσύστημα, συμπεριλαμβανομένου του TorchVision για εργασίες όρασης υπολογιστή και του TorchScript για σειριοποίηση και βελτιστοποίηση μοντέλων χρησιμοποιώντας τη μεταγλώττιση JIT. Ο modular σχεδιασμός του είναι επεκτάσιμος σε οποιοδήποτε επίπεδο και υποστηρίζει τόσο ανάπτυξη βασισμένη σε API υψηλού επιπέδου όσο και υπολογισμούς τανυστών χαμηλού επιπέδου. Το PyTorch προσφέρει επίσης διαλειτουργικότητα με το ONNX (Open Neural Network Exchange) και το TensorFlow, ενισχύοντας τις δυνατότητες ανάπτυξης του μοντέλου. Τέλος, έχει κερδίσει σημαντική υιοθέτηση στον ακαδημαϊκό κόσμο και τη βιομηχανία, συχνά χρησιμεύει ως το θεμέλιο για την έρευνα αιχμής στην τεχνητή νοημοσύνη. Παραμένει ένα κορυφαίο πλαίσιο για την έρευνα και την παραγωγή μηχανικής μάθησης λόγω της ισορροπίας απόδοσης, επεκτασιμότητας και χρηστικότητας.

3.10 ORT

Το ONNX Runtime (ORT) είναι μια μηχανή υψηλής απόδοσης για εξαγωγή συμπερασμάτων που έχει σχεδιαστεί για την αποτελεσματική εκτέλεση μοντέλων μηχανικής μάθησης σε μορφή ONNX. Αναπτύχθηκε και συντηρείται από τη Microsoft, το ORT παρέχει ταχεία εκτέλεση σε διάφορες πλατφόρμες hardware, συμπεριλαμβανομένων των CPU, των GPU, των TPU και άλλων εξειδικευμένων επιταχυντών AI [101]. Το ORT βελτιστοποιεί την εκτέλεση του μοντέλου μέσω προηγμένων τεχνικών όπως η σύντηξη τελεστών και πράξεων, ο υπολογισμός χαμηλής ακρίβειας και οι βελτιστοποιήσεις ειδικά για κάθε hardware. Υποστηρίζει πολλαπλά backends εκτέλεσης και επιτρέπει την προσθήκη επιταχυντών, επιτρέποντας την αποτελεσματική ανάπτυξη σε περιβάλλοντα cloud, edge και κινητών. Ένα από τα βασικά πλεονεκτήματα του ORT είναι η ικανότητά του να βελτιώνει την ταχύτητα εξαγωγής συμπερασμάτων μέσω βελτιστοποιήσεων γραφήματος, όπως η σύμπτυξη σταθερών και η εξάλειψη περιττών κόμβων, για μείωση των υπολογιστικών επιβαρύνσεων. Επιπλέον, το ORT παρέχει υποστήριξη για κατανεμημένη εκτέλεση, επιτρέποντας την αποτελεσματική εκτέλεση του μοντέλου σε πολλές συσκευές. Τέλος, αλληλεπιδρά εύκολα με άλλα πλαίσια όπως το Tensorflow και το Pytorch. Ο χρήστης μπορεί απλώς να εξάγει το μοντέλο από το πλαίσιο μηχανικής εκμάθησης σε μορφή ONNX, το οποίο είναι μια ευρέως υποστηριζόμενη δυνατότητα.

3.11 Keras

Το Keras, που προέρχεται από την ελληνική λέξη κέρας, είναι ένα ανοιχτού κώδικα βαθιάς εκμάθησης API γραμμένο σε Python [102]. Σχεδιάστηκε για να επιτρέπει τον γρήγορο πειραματισμό με μοντέλα βαθιάς μάθησης, διατηρώντας παράλληλα την ευελιξία και την επεκτασιμότητα. Το Keras κατασκευάστηκε αρχικά για να υποστηρίζει πολλαπλά backends όπως το theano και το tensorflow. Ωστόσο, καθώς το tensorflow συνέχιζε να μεγαλώνει και άλλες πλατφόρμες σταμάτησαν την ανάπτυξή τους, έγινε μέρος του οικοσυστήματος tensorflow ως το επίσημο API υψηλού επιπέδου και στην έκδοση 2.4 διέκοψε την υποστήριξη για πολλαπλά backend, μεταβαίνοντας στην αποκλειστική υποστήριξη του tensorflow. Από την έκδοση 3.0, το keras επέστρεψε στις ρίζες του και υποστηρίζει πολλαπλά backend, συμπεριλαμβανομένων των JAX και Pytorch. Ένα από τα βασικά πλεονεκτήματα του Keras είναι η φιλική προς το χρήστη διεπαφή, η οποία αφαιρεί περίπλοκες λειτουργίες και επιτρέπει στους χρήστες να ορίζουν μοντέλα χρησιμοποιώντας το Functional API. Το Functional API παρέχει ευελιξία για τον καθορισμό πολύπλοκων αρχιτεκτονικών, όπως μοντέλα πολλαπλών εισόδων και πολλαπλών εξόδων. Αυτή η πολύπλοκη διαδικασία καθορισμού ενός νευρωνικού δικτύου γίνεται τόσο εύκολη όσο η δημιουργία στρωμάτων και η μετάδοση των εξόδων τους ως είσοδοι σε άλλα στρώματα. Πιο προηγμένος προγραμματισμός μηχανικής μάθησης, όπως ο καθορισμός ενός προσαρμοσμένου στρώματος ή ενός βρόχου εκπαίδευσης, είναι επίσης εύκολα προσβάσιμα στα keras μέσω του αντικειμενοστρεφούς προγραμματισμού, όπου ο χρήστης μπορεί να κληρονομήσει από διάφορες κλάσεις και να υλοποιήσει μεθόδους. Επιπλέον, το Keras διαθέτει ενσωματωμένα εργαλεία για αξιολόγηση μοντέλων, προεπεξεργασία δεδομένων και συντονισμό υπερπαραμέτρων, καθιστώντας το μια προτιμώμενη επιλογή τόσο για ερευνητές όσο και για επαγγελματίες

3.12 Numba

Το έργο Numba είναι ένας μεταγλωττιστής JIT ανοιχτού κώδικα για την Python [103]. Στοχεύει στη διευκόλυνση της επιτάχυνσης των αριθμητικών υπολογισμών, επιτρέποντας στον κώδικα Python υψηλού επιπέδου να εκτελείται με ταχύτητες κοντά σε αυτές του C ή του Fortran χωρίς να απαιτείται επανεγγραφή του κώδικα σε αυτές τις γλώσσες χαμηλότερου επιπέδου. Τα βασικά χαρακτηριστικά περιλαμβάνουν αυτόματη παραλληλοποίηση και αυτόματη διανυσματοποίηση (SIMD), επιτάχυνση GPU (μέσω CUDA) και υποστήριξη για ένα ευρύ φάσμα μαθηματικών συναρτήσεων. Το Numba μπορεί να μεταγλωττίσει μια συνάρτηση Python με έναν απλό διακοσμητή `@jit`, ο οποίος προσδιορίζει τη συνάρτηση που πρόκειται να μεταγλωττιστεί, βελτιστοποιώντας την απόδοση για μεγάλα σύνολα δεδομένων και διευκολύνοντας βελτιστοποιήσεις όπως η συγχώνευση πράξεων. Επιπλέον, το Numba υποστηρίζει NumPy και οι περισσότερες εφαρμογές που χρησιμοποιούν NumPy μπορούν να επιταχυνθούν με ελάχιστες αλλαγές κώδικα.

3.13 Συγκρίσεις

Αυτή η εργασία μπορεί να συγκριθεί άμεσα μόνο με πλατφόρμες που προσφέρουν ολόκληρη την εργαλειοθήκη βαθιάς μάθησης. Οι περισσότερες σύγχρονες λύσεις προσφέρουν σχεδόν τα πάντα με συγκρίσιμες επιδόσεις, ενώ όλες εξελίσσονται συνεχώς, κάτι που μπορεί εν μέρει να αποδοθεί στην κουλτούρα του ανοιχτού κώδικα. Λαμβάνοντας αυτό υπόψη, η επιλογή ενός εργαλείου βαθιάς εκμάθησης εξαρτάται κυρίως από τις προτιμήσεις του χρήστη. Μια σημαντική διάκριση είναι ότι οι διαθέσιμες λύσεις στοχεύουν στην υποστήριξη της έρευνας και της παραγωγής τεχνητής νοημοσύνης, ενώ αυτό ήταν ένα έργο που έγινε για εκπαιδευτικούς σκοπούς. Οι διαθέσιμες πλατφόρμες προσφέρουν ένα πολύ φιλικό προς τον χρήστη Python API, αλλά η χρήση τους σε C++ μπορεί να μην είναι δυνατή ή να είναι πολύ δύσκολη. Η βιβλιοθήκη αυτής της εργασίας από την άλλη σχεδιάστηκε για να είναι πρώτα C++, με στόχο να είναι διαθέσιμη σε εφαρμογές που δεν μπορούν να ενσωματώσουν το οικοσύστημα Python, παρόλο που ένα Python API είναι υπό ανάπτυξη. Είναι επίσης σχεδιασμένο να είναι απλό και ευέλικτο στην αλλαγή ακόμη και κατά το χρόνο εκτέλεσης με τη δυνατότητα λειτουργιών που ορίζονται από το χρήστη.

Κεφάλαιο 4ο: Ανάλυση αρχιτεκτονικής και υλοποίησης

4.1 Εισαγωγή

Σε αυτό το κεφάλαιο, παρουσιάζουμε μια λεπτομερή εξερεύνηση της αρχιτεκτονικής του συστήματος, παρέχοντας μια εις βάθος ανάλυση κάθε στοιχείου και του ρόλου του στο συνολικό πλαίσιο. Η ανάλυση θα καλύψει τον σχεδιασμό και την υλοποίηση μεμονωμένων μονάδων, αναφέροντας τις τεχνικές επιλογές που έγιναν κατά την ανάπτυξη. Επιπλέον, θα εξετάσουμε πώς αυτά τα στοιχεία αλληλεπιδρούν μεταξύ τους για την επίτευξη των στόχων του συστήματος.

4.2 Αρχιτεκτονική

4.2.1 Εισαγωγή

Η αρχιτεκτονική του συστήματος βασίζεται σε επίπεδα, όπου κάθε επίπεδο εξαρτάται από τις εξόδους των προηγούμενων αλλά όχι από την υλοποίησή του όπως φαίνεται στο σχήμα 4.1. Τα κύρια επίπεδα είναι 3 το καθένα υλοποιώντας μια αυξανόμενα πιο αφηρημένη διεπαφή με χώρο για ενδιάμεσα επίπεδα, εάν κρίνονται απαραίτητες αφαιρέσεις μεταξύ των επιπέδων για την αντιμετώπιση εξαρτήσεων.

4.2.2 Layer 0

Το επίπεδο 0 είναι το σημείο όπου υλοποιούνται πραγματικά τα μαθηματικά και οι αλγόριθμοι. Η υλοποίηση διαφέρει από συσκευή σε συσκευή καθώς οι είσοδοι σε αυτό το επίπεδο είναι βασικά δεδομένα (π.χ. μια διεύθυνση μνήμης και ένα διάνυσμα σχήματος). Το επίπεδο 0 είναι μια συλλογή ”προμηθευτών” όπου κάθε προμηθευτής είναι υπεύθυνος να παρέχει ένα ”backend” για τη συσκευή του που θα παρέχει στο σύστημα όλους τους απαιτούμενους αλγόριθμους για να εκπληρώσει τις διεπαφές υψηλότερου επιπέδου. Σε αυτό το επίπεδο βρίσκουμε τους πυρήνες CUDA και την παραλληλοποίηση με χρήση νημάτων και διανυσματοποίηση στην CPU. Τα πακέτα που λειτουργούν σε αυτό το επίπεδο είναι πάροχοι πρωτόγονων λειτουργιών όπως διάφορες υλοποιήσεις BLAS, cuDNN κ.λπ.

4.2.3 Layer 1

Το επίπεδο 1 είναι ο πολυδιάστατος πίνακας. Ο πίνακας n-d απαιτείται να έχει μια γενική διεπαφή ανεξάρτητη από συσκευές και τύπους δεδομένων, ώστε ο χρήστης να μπορεί να κάνει μαθηματικές πράξεις σε διαφορετικές συσκευές χρησιμοποιώντας διαφορετικούς τύπους δεδομένων χωρίς να χρειάζεται να το εκφράσει διαφορετικά (θέλουμε να μπορούμε απλώς να γράφουμε x + y). Αυτό το επίπεδο προσφέρει σχεδόν όλη τη μαθηματική ραχοκοκαλιά της βιβλιοθήκης κάτω από μια ενιαία διεπαφή, αλλά χωρίς ακόμη να μπορεί να υπολογίσει αυτόματα παραγώγου. Τα πακέτα που μπορούμε να βρούμε σε αυτό το επίπεδο είναι τα NumPy, CuPy και σχεδόν όλα τα πακέτα μηχανικής μάθησης όπως το PyTorch.

4.2.4 Layer 0.5

Το επίπεδο 0.5 είναι ένα ενδιάμεσο στρώμα που λειτουργεί ως μεσάζων μεταξύ του επιπέδου 0 και του επιπέδου 1 για να μετριάσει τις εξαρτήσεις μεταξύ των δύο. Είναι μια αφαίρεση για έναν αποστολέα

(dispatcher) που είναι υπεύθυνος να λάβει τα απαιτούμενα δεδομένα από το επίπεδο 1 (που σημαίνει διεύθυνση, σχήμα, συσκευή, τύπος δεδομένων, τύπος λειτουργίας) και χρησιμοποιώντας αυτά τα δεδομένα να συμπεράνει τη σωστή ρουτίνα που θα κληθεί. Στη συνέχεια, ο αποστολέας θα αποστείλει τα δεδομένα στη ρουτίνα που προκύπτει, θα εκτελέσει τη ρουτίνα και, τέλος, θα επιστρέψει τις εξόδους της ρουτίνας στο πρώτο επίπεδο με τη μορφή νέου πίνακα, εάν χρειάζεται. Στην τρέχουσα υλοποίηση αυτό το επίπεδο αποστέλλει ρουτίνες μόνο σύμφωνα με τη βάση της συσκευής. Αυτό επιτρέπει σε κάθε backend (προμηθευτής επιπέδου 0) να απορρίπτει αιτήματα για λειτουργία που δεν ταιριάζουν στην υλοποίησή του (π.χ. εάν δεν υποστηρίζονται ορισμένοι τύποι δεδομένων ή μορφές μνήμης).

4.2.5 Layer 2

Το επίπεδο 2 είναι όπου εφαρμόζεται η αυτόματη διαφόριση. Αυτό το επίπεδο απαιτεί τη διεπαφή του επιπέδου 1 που σημαίνει έναν πολυδιάστατο πίνακα και όλες τις μαθηματικές πράξεις. Στη συνέχεια «τυλίγει» την υλοποίηση του πίνακα στον πλήρη τανυστή, ο οποίος ενισχύεται με επιπλέον πληροφορίες που απαιτούνται για την εκτέλεση αυτόματης διαφοροποίησης, όπως το ιστορικό των μαθηματικών πράξεων που εκτελούνται σε αυτόν. Το κύριο καθήκον αυτού του επιπέδου είναι να κατασκευάσει το υπολογιστικό γράφημα, κάθε κόμβος στο γράφημα υποδηλώνει μια μαθηματική πράξη και έχει ακμές εισόδου και εξόδου, όπου κάθε ακμή υποδηλώνει έναν τανυστή παρόμοιο με τον τρόπο που μια συνάρτηση έχει εισόδους και εξόδους. Τα πακέτα που υλοποιούν αυτό το επίπεδο είναι τα περισσότερα πλαίσια μηχανικής μάθησης όπως το JAX και το tensorflow.

4.2.6 Layer 3

Το επίπεδο 3 είναι όπου όλα κορυφώνονται, για να γίνουν τα νευρωνικά δίκτυα φιλικά προς τον χρήστη. Αυτό το επίπεδο απαιτεί έναν τανυστή με μαθηματικές λειτουργίες και δυνατότητες αυτόματης διαφοροποίησης για την κατασκευή μιας διεπαφής υψηλότερου επιπέδου ειδικών βοηθητικών προγραμμάτων μηχανικής εκμάθησης. Αυτά τα βοηθητικά προγράμματα περιλαμβάνουν βελτιστοποιητές, βρόχους εκπαίδευσης, διαφορετικούς τύπους επιπέδων νευρωνικών δικτύων και ακόμη και διαχείριση ολόκληρων μοντέλων. Τα πακέτα που μπορούν να βρεθούν σε αυτό το στρώμα περιλαμβάνουν torch.nn και keras.

4.3 The Tensor

4.3.1 Εισαγωγή

Ο τανυστής είναι η βασική μονάδα ολόκληρου του πακέτου, λειτουργεί ως αφηρημένο δοχείο δεδομένων και συμπεριφέρεται παρόμοια ανεξάρτητα από τη φυσική θέση των δεδομένων (π.χ. μνήμη GPU, μνήμη CPU). Για αυτήν την υλοποίηση ο τανυστής συμμετέχει επίσης ενεργά στην αυτόματη διαφόριση αφού κάθε ακμή ενός κόμβου στο υπολογιστικό γράφημα είναι ένας τανυστής. Παρόλο που ο τανυστής δεν είναι τεχνικά μια συμπαγής μονάδα και περιέχει μια αφαίρεση από ένα χαμηλότερο στρώμα όπως ήδη αναφέρθηκε, θα τον αντιμετωπίσουμε ως μονάδα για την ανάλυσή του.

4.3.2 Shape & Strides

Ένας τανυστής περιέχει ένα διάνυσμα σχήματος για να δηλώσει κάθε διάσταση, αυτό είναι απλώς μια επανερμηνεία των δεδομένων και όχι απαραίτητα το φυσικό σχήμα στη μνήμη. Για αυτήν την υλοποίηση, ο τανυστής περιέχει επίσης ένα διάνυσμα διασκελισμού που σημαίνει ότι υλοποιεί την πιο ευέλικτη μορφή μνήμης, αυτή η μορφή μπορεί να διευκολύνει διάφορες βελτιστοποιήσεις με ορισμένους συμβιβασμούς που θα συζητηθούν στα υποκεφάλαια 4.3.6 και 4.4.1.

4.3.3 Memory formats

Αν και αυτός ο τανυστής έχει διάνυσμα διασκελισμού, περιέχει επίσης ορισμένες σημαίες για πληροφορίες μορφής μνήμης. Αυτό συμβαίνει επειδή ένας χρήστης μπορεί να απαιτήσει τη διαχείριση των μορφών μνήμης των τανυστών. Η κατοχή αυτών των πληροφοριών είναι επίσης πολύτιμη για τη βελτιστοποίηση ορισμένων λειτουργιών χωρίς να χρειάζεται να συμπεράνουμε μια μορφή μνήμης από το διάνυσμα διασκελισμού, καθώς πολλές ρουτίνες έχουν περιορισμούς σχετικά με τις μορφές μνήμης στις οποίες λειτουργούν καλύτερα.

4.3.4 Data types

Ο τύπος δεδομένων του τανυστή είναι πληροφορία, ώστε το επίπεδο 0 να γνωρίζει πώς να ερμηνεύει τη μνήμη του. Οι τύποι δεδομένων που υποστηρίζονται αυτή τη στιγμή είναι οι πιο συχνά χρησιμοποιούμενοι αριθμητικοί τύποι:

- Bool (1 ή 0)
- Int8 (8bit ακέραιος αριθμός με πρόσημο)
- Int32 (32bit ακέραιος αριθμός με πρόσημο)
- Int64 (64bit ακέραιος αριθμός με πρόσημο)
- Half (IEEE754 binary16)
- BFloat16 (brain float 16)
- Float (IEEE754 float32)
- Double (IEEE754 float64)
- Complex64 (μιγαδικός αριθμός όπου κάθε συστατικό είναι ένας float32)
- Complex128 (μιγαδικός αριθμός όπου κάθε συστατικό είναι ένας float64)

4.3.5 Devices, memory

Ο τανυστής πρέπει να περιέχει τις πληροφορίες σχετικά με τη φυσική θέση της μνήμης του. Αυτό είναι χρήσιμο, καθώς ο χρήστης μπορεί να θέλει να διαχειριστεί διαφορετικούς τανυστές χρησιμοποιώντας

διαφορετικές συσκευές, αλλά απαιτείται επίσης, ώστε η μνήμη να μπορεί να αποσταλεί στη σωστή ρουτίνα.

Οι εκχωρήσεις μνήμης θεωρούνται λειτουργίες επιπέδου 0. Το επίπεδο 0.5, δεδομένου του ζητούμενου σχήματος, τη μορφή μνήμης ή το διάνυσμα διασκελισμού και της συσκευής θα δημιουργήσει ένα αντικείμενο εκχώρησης για τον τανυστή. Αυτό το αντικείμενο εκχώρησης περιέχει τη διεύθυνση μνήμης καθώς και μια συνάρτηση απελευθέρωσης. Το αντικείμενο εκχώρησης μπορεί να είναι κοινόχρηστο και όταν δεν θα αναφέρεται από κανένα μέρος του προγράμματος, η μνήμη θα ελευθερωθεί χρησιμοποιώντας τη συνάρτηση απελευθέρωσης.

Επιπλέον, ο τανυστής αποθηκεύει επίσης μια μετατόπιση. Επειδή ο τανυστής δεν επιτρέπεται να κάνει απευθείας αριθμητική δείκτη στο αντικείμενο εκχώρησης για να αποφύγει την καταστροφή του και τη διαρροή της μνήμης, η μετατόπιση παρέχεται ως εναλλακτική λύση, ώστε η ρουτίνα που διαβάζει τα δεδομένα να μπορεί να παρακάμψει byte. Αυτό χρησιμοποιείται για την υλοποίηση λειτουργιών που θα συζητηθούν στο κεφάλαιο 4.4.2.

Ο αριθμός των byte που έχουν εκχωρηθεί είναι φυσικά ανεξάρτητος από τη συσκευή και για έναν τανυστή με σχήμα shape $\in \mathcal{R}^n$, διασκελισμούς stride $\in \mathcal{R}^n$, μετατόπιση offset και sizeof(type) που επιστρέφει το μέγεθος σε byte του τύπου δεδομένων (π.χ. 4 byte για 32bit float), τα συνολικά byte που θα εκχωρηθούν μπορούν να υπολογιστούν με τον ακόλουθο τύπο:

$$s = \begin{cases} 0, & \text{if } \exists i \in [0, n) : \text{shape}_i = 0, \\ 1 + \sum_{i=0}^{n-1} \text{stride}_i \cdot (\text{shape}_i - 1), & \text{otherwise.} \end{cases} \tag{4.1}$$

$$\text{bytes} = \text{sizeof(type)} \cdot (\text{offset} + s) \tag{4.2}$$

Ο λόγος για την περίπτωση όπου το μέγεθος εκχώρησης byte γίνεται 0 όταν μια διάσταση του τανυστή είναι 0 είναι επειδή ο συνολικός αριθμός στοιχείων σε έναν τανυστή δίνεται από το γινόμενο του σχήματος.

$$\text{numel} = \prod_{i=0}^{n-1} \text{shape}_i \tag{4.3}$$

Επομένως, εάν οποιαδήποτε διάσταση του τανυστή είναι 0, τότε ο τανυστής έχει 0 στοιχεία και δεν πρέπει να εκτελεστεί η εκχώρηση μνήμης.

4.3.6 Broadcasting

Κάθε τανυστής έχει τη δυνατότητα να μεταδοθεί σε άλλο σχήμα εάν πληρούνται οι προϋποθέσεις. Αυτό διευκολύνει τις λειτουργίες μεταξύ τανυστών διαφορετικού σχήματος. Η υλοποίηση της μετάδοσης μπορεί να γίνει με διαφορετικούς τρόπους, ένας τρόπος είναι ένας αλγόριθμος που δημιουργεί έναν νέο τανυστή που είναι η μετάδοση του τανυστή με σχήμα shape_a σε σχήμα shape_b. Ορισμένες άλλες υλοποιήσεις χρησιμοποιούν ενσωματωμένη μετάδοση εντός της λειτουργίας (π.χ. μια συνάρτηση BroadcastAdd που κάνει πρόσθεση με μετάδοση). Η υλοποίηση μας είναι παρόμοια με άλλα πακέτα όπως το NumPy και το Pytorch, λαμβάνοντας υπόψη ότι χρησιμοποιούμε ένα διάνυσμα διασκελισμού. Έχει σταθερή πολυπλοκότητα τόσο στο χρόνο όσο και στη μνήμη $O(1)$ σε σχέση με το μέγεθος του τανυστή. Η μετάδοση

χρησιμοποιώντας ένα διάνυσμα διασκελισμών απλώς ορίζει το σχήμα του τανυστή ως το νέο σχήμα και, στη συνέχεια, για κάθε διάσταση που μεταδόθηκε ορίζει τον διασκελισμό στο 0.

4.3.7 Differentiable Tensor

Ένας διαφορίσιμος τανυστής ενθυλακώνει τον μαθηματικό τανυστή και τον επεκτείνει ώστε να μπορεί επίσης να είναι μια ακμή στο υπολογιστικό γράφημα. Ο νέος τανυστής θα περιέχει τώρα πληροφορίες όπως ο κόμβος που τον παρήγαγε ως έξοδο και έναν άλλο τανυστή που θα χρησιμοποιηθεί για την αποθήκευση του gradient. Αυτός ο τανυστής μπορεί στη συνέχεια να τροφοδοτηθεί ως ακμή εισόδου σε άλλους κόμβους στη διαδικασία κατασκευής του υπολογιστικού γραφήματος. Σε οποιοδήποτε σημείο του προγράμματος, το AD μπορεί να υπολογίσει gradients σε σχέση με οποιονδήποτε τανυστή χρησιμοποιώντας backpropagation από τον κόμβο που παρήγαγε αυτόν τον τανυστή ως έξοδο στις ακμές (τανυστές) που ήταν είσοδοι σε αυτόν τον κόμβο αναδρομικά για κάθε ακμή. Αυτή η διαδικασία θα εξηγηθεί περαιτέρω στο υποκεφάλαιο 4.7.

4.4 Operations

4.4.1 Εισαγωγή

Οι operators είναι από τις πιο σημαντικές πτυχές του πακέτου, περιγράφουν όλους τους μετασχηματισμούς ή τη δημιουργία δεδομένων που μπορεί να κάνει το πακέτο. Η υλοποίηση για τους περισσότερους παρέχεται συνήθως από έναν προμηθευτή όπως αναφέρθηκε προηγουμένως με ορισμένες εξαιρέσεις. Βρίσκονται σε ένα μητρώο, ώστε να μπορεί να έχει πρόσβαση σε αυτούς το επίπεδο 0.5, αυτό το μοντέλο προσθέτει επίσης κάποια επιπλέον ευελιξία στη δυνατότητα προσθήκης πρόσθετων τελεστών ή εγγραφής εναλλακτικών υλοποιήσεων τελεστών που ορίζονται από τον χρήστη.

4.4.2 Initialization

Οι τελεστές αρχικοποίησης είναι υπεύθυνοι για τη δημιουργία ενός τανυστή και την αρχικοποίηση των τιμών του. Η εκχώρηση μνήμης θεωρείται τελεστής αρχικοποίησης καθώς αρχικοποιεί έναν κενό τανυστή. Η συμπλήρωση όλων των στοιχείων ενός τανυστή με μια τιμή (π.χ. δημιουργία ενός μηδενικού τανυστή) ή με διάφορες τυχαίες τιμές που δειγματοληπτήθηκαν από κάποια κατανομή είναι επίσης τελεστές αυτής της κατηγορίας. Στο πλαίσιο των νευρωνικών δικτύων, αυτοί οι τελεστές κάνουν την αρχικοποίηση βαρών αλλά δεν περιέχονται στο υπολογιστικό γράφημα.

4.4.3 Tensor manipulation

Οι τελεστές χειρισμού τανυστών βελτιστοποιούνται ώστε μερικές φορές να είναι σταθερής πολυπλοκότητας χρησιμοποιώντας το διάνυσμα διασκελισμού. Αυτό σημαίνει ότι το αντικείμενο εκχώρησης θα είναι κοινόχρηστο μεταξύ του αρχικού και του νέου τανυστή και ο νέος τανυστής θα είναι αυτό που συνήθως ονομάζεται ”όψη” του αρχικού τανυστή. Το broadcasting είναι ένα παράδειγμα χειρισμού ενός τανυστή. Άλλες λειτουργίες περιλαμβάνουν την αλλαγή του σχήματος του τανυστή για την εκ νέου ερμηνεία των δεδομένων σε διαφορετική διάσταση, την ευρετηρίαση των δεδομένων του τανυστή, τη στοίβαξη τανυστών σε ένα ή το διαχωρισμό ενός τανυστή σε μικρότερους.

Μια σύνηθες πράξη είναι ο ανάστροφος πίνακας A^T το οποίο είναι μια μετάθεση διαστάσεων και μπορεί να εφαρμοστεί σε σταθερό χρόνο απλώς μεταθέτοντας τα διανύσματα διασκελισμών και σχήματος.

Η εξαγωγή πραγματικών ή φανταστικών μερών ενός μιγαδικού τανυστή είναι επίσης σταθερής πολυπλοκότητας, αφού το μόνο που πρέπει να γίνει είναι να διπλασιαστεί το διάνυσμα διασκελισμών και να μετατοπιστεί από το μισό μέγεθος του μιγαδικού αριθμού, εάν πρόκειται για εξαγωγή του φανταστικού μέρους. Αυτό συμβαίνει επειδή οι μιγαδικοί αριθμοί στη μνήμη για αυτήν την υλοποίηση αποθηκεύονται σε ζεύγη συστατικών ενός μιγαδικού αριθμού.

Ο διαχωρισμός, ο τεμαχισμός και ακόμη και η λήψη της διαγώνιας ενός τανυστή γίνονται όλα σε σταθερό χρόνο με κόλπα διασκελισμού, ενώ άλλα όπως η συνένωση, η στοίβαξη και το padding απαιτούν τη δημιουργία ενός νέου τανυστή.

Όλες αυτές οι πράξεις ακόμη και η λειτουργία της μετακίνησης των δεδομένων σε άλλη συσκευή είναι κόμβοι στο υπολογιστικό γράφημα και η «παράγωγός» τους είναι απλώς το αντίστροφο της πράξης. Για παράδειγμα, στο πέρασμα προς τα πίσω της λειτουργίας reshape, το εισερχόμενο gradient που θα έχει το ίδιο σχήμα με τον ανασχηματισμένο τανυστή θα αναδιαμορφωθεί στο αρχικό σχήμα για να λειτουργήσουν οι επόμενοι υπολογισμοί.

4.4.4 Element wise

Οι πράξεις βάσει στοιχείων είναι πράξεις που είναι συνήθως γραμμικής πολυπλοκότητας $O(N)$ και εκτελούν έναν μετασχηματισμό για κάθε στοιχείο του τανυστή ή παράγουν μια τιμή με μια συνάρτηση άλλων τανυστών. Ένα παράδειγμα είναι η απλή προσθήκη ή διάφορες λειτουργίες ενεργοποίησης. Αυτές οι λειτουργίες στην τρέχουσα υλοποίηση υποστηρίζουν πλήρως τη μορφή διασκελισμού, αν και είναι πιο βελτιστοποιημένες όταν τα στοιχεία αποθηκεύονται συνεχόμενα στη μνήμη, καθώς αυτό μειώνει τις αστοχίες της κρυφής μνήμης και επιτρέπει τη διανυσματοποίηση.

4.4.5 Reduction

Οι πράξεις μείωσης θα εφαρμόσουν μια αθροιστική συνάρτηση για τη μείωση μιας ή περισσότερων ή ακόμα και όλων των διαστάσεων ενός τανυστή σε 1. Αυτές περιλαμβάνουν τη συλλογή στατιστικών στοιχείων στο στώμα κανονοποίησης παρτίδας των νευρωνικών δικτύων και άλλες κοινές πράξεις, όπως η άθροιση. Παρόμοια με τις element wise, υποστηρίζουν τη μορφή strided.

4.4.6 Linear algebra

Οι λειτουργίες γραμμικής άλγεβρας παρέχονται από μια εξωτερική βιβλιοθήκη όπως οι BLAS, cuBLAS και LAPACK καθώς αυτές είναι οι πιο σύγχρονες λύσεις για αυτά τα προβλήματα. Περιέχουν διάφορους επιλύτες γραμμικών συστημάτων και τον αγαπημένο πολλαπλασιασμό πινάκων. Τα πακέτα για τέτοιες λειτουργίες συνήθως δεν υποστηρίζουν μια γενική μορφή strided, επομένως ενδέχεται να χρειαστεί να δημιουργηθούν πρόσθετα αντίγραφα κατά τη διαδικασία ολοκλήρωσης της λειτουργίας.

Συνήθως απαιτούν οι πίνακες να είναι σε μορφή Column-Major και στην περίπτωση επίλυσης γραμμικών συστημάτων όπου δεν είναι συνηθισμένο πρόβλημα μηχανικής μάθησης, στις περισσότερες περιπτώσεις

γίνεται αντιγραφή σε αυτήν την μορφή. Για τον πολλαπλασιασμό πινάκων, όταν οι μορφές είναι είτε column major είτε row major, μπορούμε να παρακάμψουμε τα αντίγραφα χρησιμοποιώντας τα ορίσματα για προαιρετική αναστροφή του BLAS. Για παράδειγμα, μπορούμε να χρησιμοποιήσουμε τον αλγόριθμο με πίνακες row major χρησιμοποιώντας τις ακόλουθες πληροφορίες. Πρώτον, ο ανάστροφος ενός πίνακα row major είναι ο ίδιος πίνακας στη column major. Δεύτερον, η μαθηματική ιδιότητα των πινάκων:

$$(AB)^T = B^T A^T \tag{4.4}$$

Δεδομένων αυτών και υποθέτοντας ότι τα A και τα B είναι row major, μπορούμε να τα αντιμετωπίσουμε ως ανάστροφα column major και να αλλάξουμε τα ορίσματα για να δημιουργήσουμε το $(AB)^T$. Επειδή το αποτέλεσμα αυτών των συναρτήσεων είναι επίσης column major, σημαίνει ότι το αποτέλεσμα είναι το AB σε row major. Οι περισσότερες σύγχρονες υλοποιήσεις BLAS χρησιμοποιούν ένα όρισμα μορφής μνήμης και κάνουν ήδη αυτό το κόλπο.

4.4.7 NN primitives

Οι πρωτόγονες λειτουργίες νευρωνικών δικτύων είναι λειτουργίες που για λόγους βελτιστοποίησης, ακόμη και όταν μπορούν να αποσυντεθούν σε μια σειρά από άλλες λειτουργίες, συγχωνεύονται σε μια μονάδα. Οι περισσότερες από τις υλοποιήσεις προσφέρονται από πακέτα όπως το cuDNN καθώς προσφέρουν επιδόσεις αιχμής για συνελίξεις, προσοχή, κανονικοποίηση παρτίδας κ.λπ., αλλά εφεδρικές υλοποιήσεις υπάρχουν επίσης όταν δεν μπορεί να βρεθεί μια βελτιστοποιημένη υλοποίηση. Αυτές οι πράξεις είναι συνήθως ένα ολόκληρο στρώμα ενός νευρωνικού δικτύου με την παράγωγό του, ώστε να μπορούν να ολοκληρωθούν γρήγορα σε ένα βήμα χωρίς να δημιουργηθεί ένα πολύπλοκο υπολογιστικό γράφημα.

4.4.8 User defined

Το πακέτο επιτρέπει τελεστές που ορίζονται από το χρήστη και το ενθαρρύνει καθώς προσφέρει πρότυπα για εύκολη σύνθεση μιας λειτουργίας reduction ή element wise. Αυτές οι λειτουργίες αποτελούνται από συναρτήσεις μετασχηματισμού και έναν βρόχο, που σημαίνει ότι ο βρόχος μπορεί να γραφτεί μία φορά και διαφορετικές λειτουργίες μπορούν να δημιουργηθούν παρέχοντας διαφορετικές συναρτήσεις. Κοινή περίπτωση χρήσης είναι η συγχώνευση διαφορετικών λειτουργιών σε έναν ενιαίο πυρήνα για καλύτερη απόδοση. Εάν ο χρήστης θέλει να κάνει τη λειτουργία διαφοροποιήσιμη, θα πρέπει επίσης να την καταχωρήσει στο επίπεδο 2 δημιουργώντας έναν νέο τύπο κόμβου και παρέχοντας την παράγωγο. Επιπλέον, ένας εύκολος τρόπος δημιουργίας μιας νέας λειτουργίας είναι η χρήση των χαρακτηριστικών μεταγλώττισης JIT και η παροχή του μαθηματικού τύπου σε κείμενο, αυτό έχει μερικές ενδιαφέρουσες πιθανές μελλοντικές επεκτάσεις στον υπολογισμό της συμβολικής παραγώγου του τύπου και την αυτόματη καταχώρηση ενός νέου τύπου κόμβου. Η παραγόμενη νέα λειτουργία θα απολαμβάνει όλα τα πλεονεκτήματα βελτιστοποίησης των ήδη υπαρχόντων συμπεριλαμβανομένης της διανυσματοποίησης όποτε είναι δυνατόν.

4.4.9 Optimizations

Για βελτιστοποιήσεις, η παραλληλοποίηση με νήματα και SIMD χρησιμοποιείται μέσω Google/Highway στην CPU. Ο συγχρονισμός στις περισσότερες περιπτώσεις δεν αποτελεί πρόβλημα, καθώς όλα τα κελιά του τανυστή εξόδου γράφονται μία φορά, για παράδειγμα σε element wise πράξεις, ένα παράδειγμα παράλληλης μείωσης χωρίς κλείδωμα φαίνεται στο σχήμα 4.2. Όταν υπάρχουν εγγραφές στις ίδιες διευθύνσεις παράλληλα, τα atomics προτιμώνται έναντι των mutexes, επειδή τις περισσότερες φορές το στοιχείο δεν είναι μεγαλύτερο από το μέγεθος λέξης, επομένως ένα atomic store φροντίζει για αυτό. Για τη GPU, η διανυσματοποίηση αποθήκευσης και φόρτωσης γίνεται όποτε είναι δυνατόν, χρήση συγχωνευμένων συναλλαγών μνήμης και γενικά καλή χρήση του μοντέλου προγραμματισμού CUDA συμπεριλαμβανομένης της χρήσης βελτιστοποιημένων βιβλιοθηκών όπως cuBLAS, cuDNN και CUB. Και οι δύο συσκευές έχουν παρόμοια συμπεριφορά κρυφής μνήμης, πράγμα που σημαίνει ότι η συνεχόμενη μνήμη χωρίς διασκελισμό προτιμάται για τη μείωση των αστοχιών της κρυφής μνήμης, καθώς θα επιτρέψει επίσης τη διανυσματοποίηση. Ωστόσο, σε πολλές περιπτώσεις όπου χειρισμός των σχημάτων γίνεται εκτενώς (transpose, reshape κ.λπ.), η αποφυγή επιπλέον αντιγράφων και κλήσεων πυρήνα υπερτερεί κατά πολύ της περιστασιακής υποβέλτιστης element wise πράξης ή της δημιουργίας ενός μόνο αντιγράφου για μια πράξη γραμμικής άλγεβρας.

4.5 JIT

Το πακέτο προσφέρει μεταγλώττιση χρόνου εκτέλεσης με χρήση μαθηματικών τύπων σε κείμενο. Τα οφέλη για την εκφόρτωση της μεταγλώττισης των αλγορίθμων κατά το χρόνο εκτέλεσης είναι το μειωμένο μέγεθος εφαρμογής και η ευελιξία σε διαφορετικούς τύπους δεδομένων και μορφές μνήμης. Οι λειτουργίες που είναι διαθέσιμες και μεταγλωττίζονται εκ των προτέρων χρησιμοποιούν κάποια μορφή προώθησης των τελεστών σε έναν κοινό τύπο δεδομένων και μορφή μνήμης για να εξαλείψουν την ανάγκη μεταγλώττισης κάθε λειτουργίας για κάθε πιθανή μετάθεση τύπων δεδομένων και τύπων βελτιστοποίησης. Ωστόσο, όλες αυτές οι μεταγλωττισμένες εκδόσεις λειτουργιών βρίσκονται στην εφαρμογή και ενδέχεται να μην χρησιμοποιηθούν ποτέ, επομένως θα ήταν ωφέλιμο να υπάρχει μόνο αυτό που χρειάζεται ο χρήστης για την περίπτωση χρήσης του. Η μεταγλώττιση στο χρόνο εκτέλεσης μπορεί να έχει ένα επιπλέον κόστος χρόνου εκτέλεσης την πρώτη φορά, αλλά η εκπαίδευση των νευρωνικών δικτύων είναι μια επαναληπτική διαδικασία και η πληρωμή του κόστους μία φορά μπορεί να γίνει ανεκτή.

Οι μεταγλωττισμένες JIT πράξεις επωφελούνται από όλες τις βελτιστοποιήσεις των AOT και πιθανώς ακόμη περισσότερες, καθώς οι πληροφορίες χρόνου εκτέλεσης μπορούν πλέον να μετατραπούν σε πληροφορίες χρόνου μεταγλώττισης. Ένα καλό παράδειγμα είναι ότι όλες οι λειτουργίες reduction στη GPU είναι προς το παρόν μεταγλωττισμένες JIT και δεν υπάρχουν προμεταγλωττισμένες υλοποιήσεις.

Για την GPU, το JIT υλοποιείται με τη βοήθεια του έργου jitify. Για την CPU δημιουργείται ένα νέο αρχείο C++ με όλες τις απαιτούμενες πληροφορίες για τη δημιουργία και τη βελτιστοποίηση της συνάρτησης και στη συνέχεια μεταγλωττίζεται χρησιμοποιώντας τον μεταγλωττιστή C++ του συστήματος και φορτώνεται δυναμικά ως DLL στο κύριο πρόγραμμα. Αυτή η υλοποίηση είναι κάπως αναποτελεσματική και υπόκειται σε αλλαγές.

Επιπλέον, η μονάδα JIT έχει τη δυνατότητα εξαγωγής του κώδικα assembly που δημιουργεί. Αυτό προ-

στέθηκε για εκπαιδευτικούς σκοπούς για τη μελέτη του κώδικα, αλλά η γνώση του τι παράγει η C++ του χρήστη μπορεί επίσης να είναι πολύ χρήσιμη για τη βελτιστοποίηση και τον εντοπισμό σφαλμάτων.

Μια μικρή βιβλιοθήκη python εμπνευσμένη από numba δημιουργήθηκε ως μέσο δοκιμής των δυνατοτήτων σύντηξης λειτουργιών με ένα παράδειγμα χρησιμοποιώντας numpy arrays. Τα αποτελέσματα είναι στα σχήματα 4.3 και 4.4.

4.6 IO module

Περιλαμβάνεται μια μονάδα IO που υλοποιεί τις πολύ βασικές δυνατότητες φόρτωσης και αποθήκευσης δεδομένων από δίσκο. Οι μορφές που υποστηρίζονται αυτήν τη στιγμή είναι οι npy και npz του NumPy για συμβατότητα. Αυτό είναι σίγουρα το πιο ελλιπές μέρος της βιβλιοθήκης και απαιτεί τις περισσότερες επεκτάσεις και βελτιώσεις.

4.7 AD module

Ο υπολογισμός gradient είναι σίγουρα ένα από τα πιο σημαντικά μέρη της μηχανικής μάθησης. Επομένως, αυτή η μονάδα είναι ζωτικής σημασίας. Αυτή η ενότητα είναι το επίπεδο 2 της αρχιτεκτονικής και είναι υπεύθυνη για την κατασκευή ενός υπολογιστικού γραφήματος και την εκτέλεση της backpropagation. Η λειτουργία ευθείας AD δεν έχει δημιουργηθεί ακόμη, καθώς δεν είναι τόσο συνηθισμένη όσο η αντίστροφη. Η κατασκευή του υπολογιστικού γραφήματος γίνεται στο παρασκήνιο καθώς ο χρήστης κάνει πράξεις σε τανυστές.

Στην αντίστροφη AD κάθε κόμβος θα υπολογίσει τις παραγώγους κάθε τανυστή εισόδου που είχε στο forward pass και στη συνέχεια θα τις προωθήσει στις συναρτήσεις που παρήγαγαν αυτούς τους τανυστές αναδρομικά. Εάν περισσότεροι από ένας κόμβοι προωθούν παραγώγους σε κάποια άλλη συνάρτηση στον ίδιο δείκτη ορίσματος, τότε όλες οι προωθημένες τιμές προστίθενται μαζί. Αυτός ο σχεδιασμός προώθησης των παραγώγων ως εισόδων σε μια συνάρτηση διασφαλίζει ότι τα gradients που δεν χρειάζεται να ζουν μετά το backpropagation δεν θα αποθηκευτούν στους αντίστοιχους τανυστές τους εκτός εάν το ζητήσει ο χρήστης.

Τελικά ορισμένοι τανυστές δεν θα έχουν καμία σύνδεση με έναν προηγούμενο κόμβο που σημαίνει ότι μπορούν να αντιμετωπιστούν ως κόμβοι φύλλων. Συνήθως αυτοί οι τανυστές είναι τα βάρη του νευρωνικού δικτύου και τα gradients τους θα συμπληρωθούν και θα προστεθούν εκτός εάν ο χρήστης τους επισήμανε ως μη απαραίτητους (ή μη εκπαιδεύσιμους).

Είναι απαραίτητο η πρόσβαση σε κάθε κόμβο να γίνεται μόνο αφού συμπληρωθούν πλήρως και προστεθούν όλες οι είσοδοι του για το backward-pass, πράγμα που σημαίνει ότι θα επιλυθούν όλες οι εξαρτήσεις του πριν από την επίκληση του υπολογισμού της παραγώγου. Για την εκτέλεση της αντίστροφης AD, το γράφημα ταξινομείται πρώτα σε τοπολογική σειρά χρησιμοποιώντας τον αλγόριθμο αναζήτησης κατά βάθος (DFS) ως διέλευση και στη συνέχεια διασχίζεται με αντίστροφη σειρά. Αυτό διασφαλίζει ότι όταν ένας υπολογισμός παραγώγου καλείται από έναν κόμβο, όλες οι εξαρτήσεις του έχουν ήδη επιλυθεί.

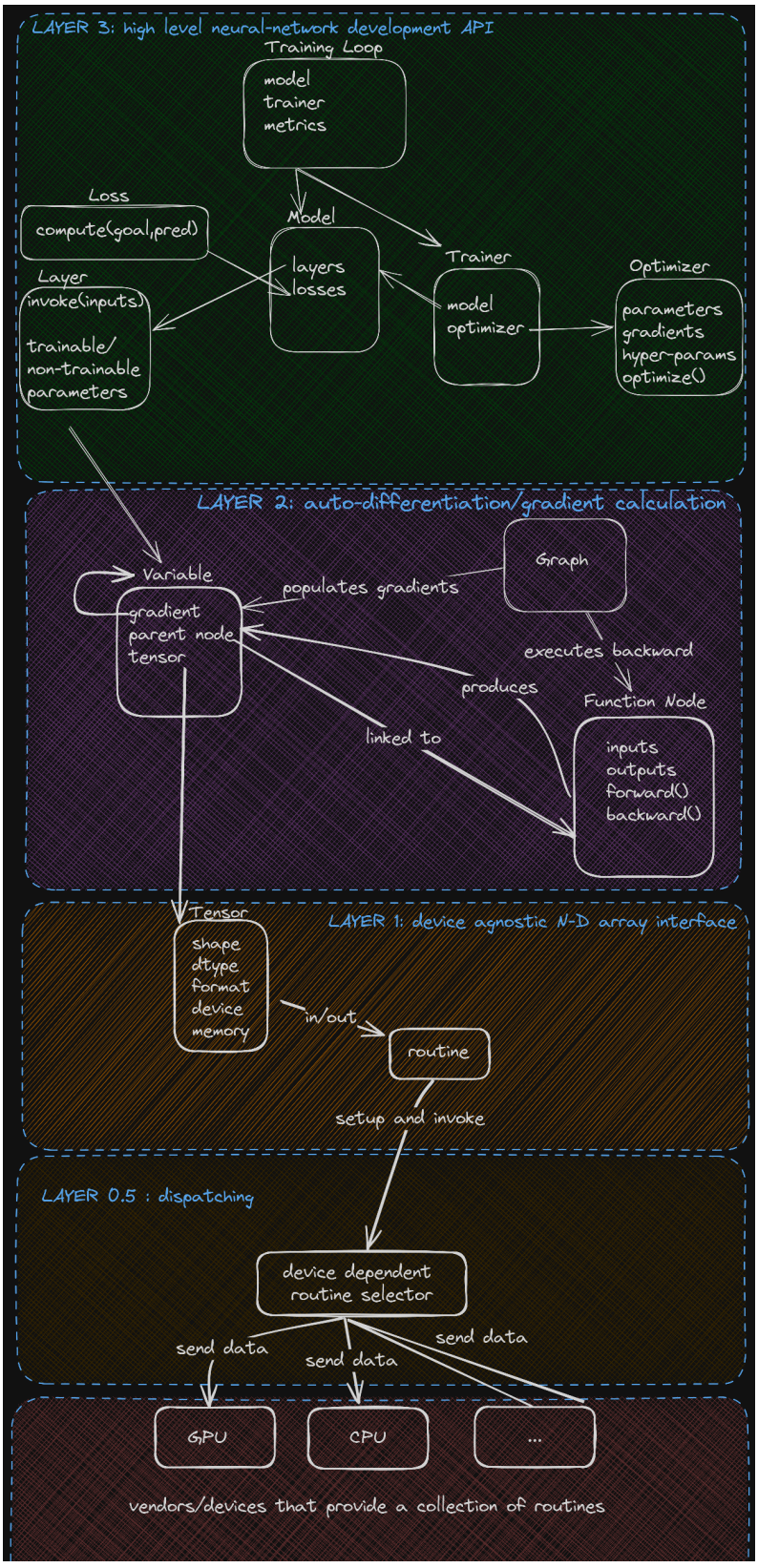
Οι τανυστές με τύπο δεδομένων ακέραιο αριθμό δεν είναι διαφορίσιμοι, καθώς για να αντιπροσωπεύουν το ρυθμό μεταβολής μιας συνάρτησης τα δεδομένα πρέπει να είναι συνεχόμενα έτσι ώστε μικρές αλλαγές

στην είσοδο να έχουν ως αποτέλεσμα ομαλές αλλαγές στην έξοδο, ένα χαρακτηριστικό που δεν παρέχουν οι ακέραιοι αριθμοί. Αυτή η συμπεριφορά είναι η ίδια σε άλλα πλαίσια μηχανικής μάθησης, όπως το Pytorch και το tensorflow.

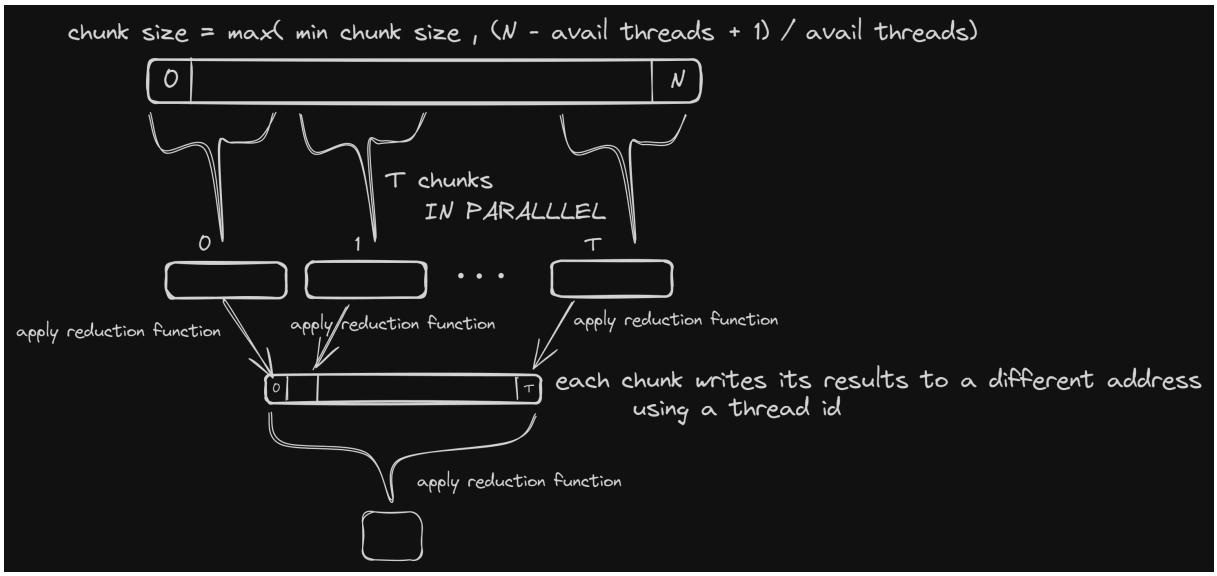
Ο υπολογισμός των παραγώγων υψηλότερης τάξης μπορεί να ενεργοποιηθεί επιτρέποντας στο backward pass να δημιουργήσει ένα επόμενο υπολογιστικό γράφημα με όλες τις πράξεις που εκτελέστηκαν κατά τον υπολογισμό των gradients.

4.8 Μονάδα δημιουργίας νευρωνικών δικτύων

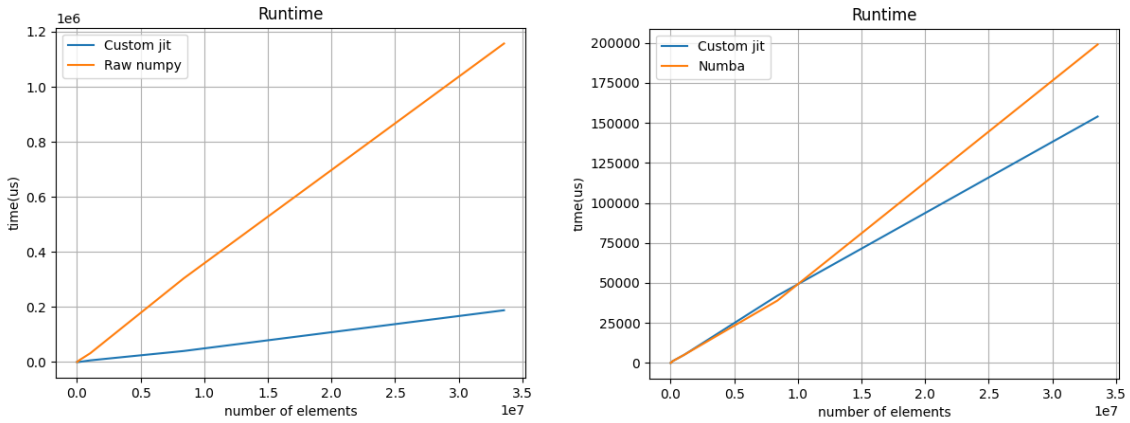
Αυτό αφορά το επίπεδο 3 της εφαρμογής που σημαίνει μια φιλική προς το χρήστη διεπαφή για γρήγορη δημιουργία πρωτοτύπων αρχιτεκτονικών νευρωνικών δικτύων χρησιμοποιώντας επίπεδα ως δομικά στοιχεία και διαχείριση μοντέλων. Η διεπαφή στην τρέχουσα κατάστασή της δεν είναι τόσο φιλική προς το χρήστη, αλλά βελτιώνεται. Μετά από αρκετή σκέψη αποφασίστηκε ότι η καλύτερη λύση είναι να εκτεθούν τα πάντα μέχρι το επίπεδο 2 ως βιβλιοθήκη python και η δημιουργία του επιπέδου 3 να γίνει σε python χρησιμοποιώντας όλες τις ισχυρές δυνατότητες μεταπρογραμματισμού ή εναλλακτικά μια επέκταση keras που υποστηρίζει αυτό το backend .



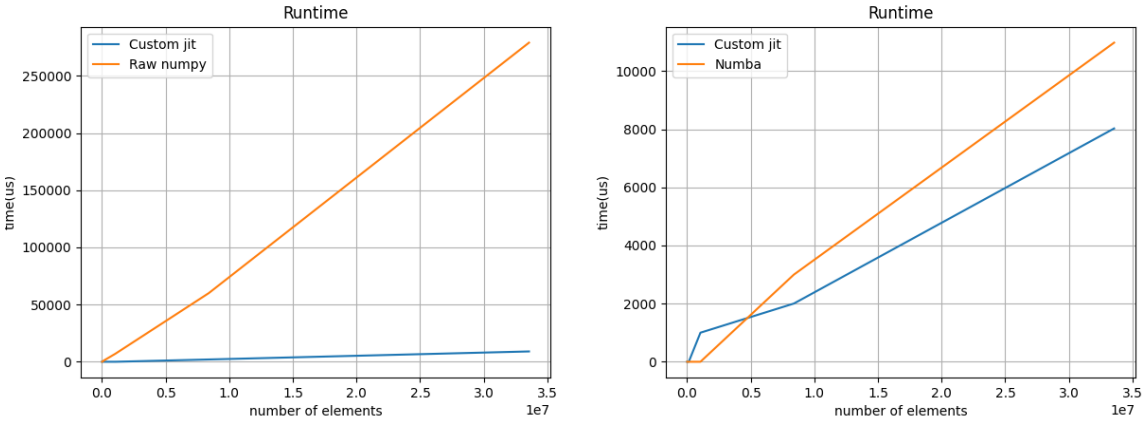
Σχήμα 4.1: Αρχιτεκτονική εφαρμογής



Σχήμα 4.2: Παράδειγμα αλγόριθμου παράλληλης μείωσης που χρησιμοποιείται



Σχήμα 4.3: Σύγκριση χρόνου εκτέλεσης της σύντηξης λειτουργιών βάσει στοιχείων με JIT



Σχήμα 4.4: Σύγκριση χρόνου εκτέλεσης της σύντηξης λειτουργιών μείωσης με JIT

Κεφάλαιο 5ο: Συμπεράσματα και το μέλλον

5.1 Συμπεράσματα

Γενικά είμαι πολύ ικανοποιημένος με τα αποτελέσματα σε αυτό το συγκεκριμένο χρονοδιάγραμμα, λαμβάνοντας υπόψη την εκπαίδευση και την εξέλιξη που μου έδωσε αυτό το έργο σε κάθε βήμα του ταξιδιού. Το έργο είναι πράγματι μια πλήρως εξοπλισμένη βιβλιοθήκη μηχανικής εκμάθησης με εξαιρετική απόδοση, αν και υπάρχει ακόμη περιθώριο βελτίωσης. Το όλο εγχείρημα ήταν σκληρό και πολύ πιο δύσκολο από το αναμενόμενο, όπως είναι τα περισσότερα πράγματα, αλλά τα αποτελέσματα δικαιολογούν τον κόπο.

Ένα σημείο άξιο αναφοράς είναι ότι ένα τέτοιο σύστημα με την πληθώρα χαρακτηριστικών που διαθέτουν τα περισσότερα πλαίσια μηχανικής μάθησης, αγγίζει πολύ περισσότερες έννοιες προγραμματισμού και γνώσεις από ό,τι φαίνεται. Τομείς όπως η αρχιτεκτονική υπολογιστών και οι βελτιστοποιήσεις, προβλήματα γραφημάτων, μεταγλωττιστές, σχεδιασμός API, διαλειτουργικότητα μεταξύ γλωσσών προγραμματισμού, αριθμητικές μέθοδοι ακόμη και IO και χειρισμός ποικίλων μορφών αρχείων. Γνώσεις από αυτούς και άλλους τομείς συνδυάζονται σε μια βιβλιοθήκη μηχανικής μάθησης.

Αυτή η βιβλιοθήκη κάνει τα πρώτα της βήματα και δεν υπάρχει καμία ανάγκη για συμβατότητα προς τα πίσω. Ως εκ τούτου, αναμένεται να ακολουθήσουν δραστικές αλλαγές και πειραματισμοί με νέες τεχνικές. Το έργο ήταν διασκεδαστικό στη χρήση και την ανάπτυξη. Τέλος, ένα πράγμα που λείπει είναι ένας μοναδικός χαρακτήρας, κάποιο πικάντικο συστατικό που θα το διαφοροποιήσει από τα υπόλοιπα με έναν ενδιαφέροντα τρόπο, αλλά αυτή η αναζήτηση θα διεξαχθεί μέσω της περισσότερης έρευνας και περισσότερης ανάπτυξης.

5.2 Το μέλλον

Αυτό το έργο απαιτεί πολλή δουλειά, αυτό ήταν στην πραγματικότητα μόνο η αρχή. Πρώτον, πολλές αρχιτεκτονικές βελτιώσεις μπορούν να γίνουν σχετικά με τον τρόπο αλληλεπίδρασης των στοιχείων μεταξύ τους. Ίσως η επόμενη μεγαλύτερη προτεραιότητα είναι η κατάργηση των περισσότερων από τις υλοποιήσεις του επιπέδου 0 και αντ ’αυτού, η χρήση της μονάδας JIT ή των εξωτερικών βιβλιοθηκών, αυτό θα κάνει το έργο πιο ελαφρύ και πιο συντηρήσιμο. Η υποστήριξη για περισσότερες συσκευές και επιταχυντές θα ήταν φυσικά καλή. Η δημιουργία μιας διεπαφής python είναι επίσης στις προτεραιότητες, καθώς θα διευκολύνει επίσης το testing του λογισμικού. Τέλος επεκτάσεις της μονάδας IO ώστε να μπορεί να υποστηρίξει διαφορετικές μορφές αρχείων και κυρίως τη μορφή ONNX για διαλειτουργικότητα.

Βιβλιογραφία

[1] L. S. Blackford, A. Petitet, R. Pozo, K. Remington, R. C. Whaley, J. Demmel, J. Dongarra, I. Duff, S. Hammarling, G. Henry, *et al.*, "An updated set of basic linear algebra subprograms (BLAS)," *ACM Transactions on Mathematical Software*, vol. 28, no. 2, pp. 135–151, 2002.

[2] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, pp. 357–362, Sept. 2020.

[3] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," 2019.

[4] M. Abadi and G. D. Plotkin, "A simple differentiable programming language," *Proceedings of the ACM on Programming Languages*, vol. 4, p. 1–28, Dec. 2019.

[5] Modular, "Mojo: Powerful cpu+gpu programming," 2024. `https://www.modular.com/mojo` [Accessed: (2024-07-04)].

[6] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The Bulletin of Mathematical Biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[7] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain.," *Psychological review*, vol. 65 6, pp. 386–408, 1958.

[8] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533–536, 1986.

[9] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[10] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989.

[11] C. Cortes and V. Vapnik, "Support-vector networks," *Chem. Biol. Drug Des.*, vol. 297, pp. 273–297, 01 2009.

[12] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–44, 05 2015.

[13] A. Krizhevsky, I. Sutskever, and G. Hinton, "Imagenet classification with deep convolutional neural networks," *Neural Information Processing Systems*, vol. 25, 01 2012.

[14] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mane, R. Monga, S. Moore, D. Murray, C. Olah,

M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viegas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," 2016.

[15] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," 2015.

[16] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," 2018.

[17] D. Silver, A. Huang, C. Maddison, A. Guez, L. Sifre, G. Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, pp. 484–489, 01 2016.

[18] M. Wiering and M. Otterlo, *Reinforcement Learning: State-Of-The-Art*, vol. 12. 01 2012.

[19] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. baker, M. Lai, A. Bolton, Y. Chen, T. P. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis, "Mastering the game of go without human knowledge," *Nature*, vol. 550, pp. 354–359, 2017.

[20] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," 2018.

[21] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 06 2014.

[22] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," 2019.

[23] A. Ng, "Feature selection, l 1 vs. l 2 regularization, and rotational invariance," *Proceedings of the Twenty-First International Conference on Machine Learning*, 09 2004.

[24] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 157–166, 1994.

[25] F. Informatik, Y. Bengio, P. Frasconi, and J. Schmidhuber, "Gradient flow in recurrent nets: the difficulty of learning long-term dependencies," *A Field Guide to Dynamical Recurrent Neural Networks*, 03 2003.

[26] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

[27] J. Zhang, T. He, S. Sra, and A. Jadbabaie, "Why gradient clipping accelerates training: A theoretical justification for adaptivity," 2020.

[28] K. Diamantaras and D. Botsis, *Μηχανική Μάθηση*. Κλεδάριθμος, 07 2019.

[29] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017.

[30] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” 2015.

[31] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, “Aggregated residual transformations for deep neural networks,” in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5987–5995, 2017.

[32] J. Long, E. Shelhamer, and T. Darrell, “Fully convolutional networks for semantic segmentation,” 2015.

[33] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative adversarial networks,” *Commun. ACM*, vol. 63, p. 139–144, Oct. 2020.

[34] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, “Mobilenets: Efficient convolutional neural networks for mobile vision applications,” 2017.

[35] F. Chollet, “Xception: Deep learning with depthwise separable convolutions,” in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1800–1807, 2017.

[36] M. Lin, Q. Chen, and S. Yan, “Network in network,” 2014.

[37] S. Ioffe and C. Szegedy, “Batch normalization: accelerating deep network training by reducing internal covariate shift,” in *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML’15, p. 448–456, JMLR.org, 2015.

[38] J. L. Ba, J. R. Kiros, and G. E. Hinton, “Layer normalization,” 2016.

[39] Y. Wu and K. He, “Group normalization,” 2018.

[40] D. Ulyanov, A. Vedaldi, and V. Lempitsky, “Instance normalization: The missing ingredient for fast stylization,” 2017.

[41] B. Zhang and R. Sennrich, “Root mean square layer normalization,” 2019.

[42] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, and G. Lample, “Llama: Open and efficient foundation language models,” 2023.

[43] DeepSeek-AI, A. Liu, B. Feng, B. Xue, B. Wang, B. Wu, C. Lu, C. Zhao, C. Deng, C. Zhang, C. Ruan, D. Dai, D. Guo, D. Yang, D. Chen, D. Ji, E. Li, F. Lin, F. Dai, F. Luo, G. Hao, G. Chen, G. Li, H. Zhang, H. Bao, H. Xu, H. Wang, H. Zhang, H. Ding, H. Xin, H. Gao, H. Li, H. Qu, J. L. Cai, J. Liang, J. Guo, J. Ni, J. Li, J. Wang, J. Chen, J. Chen, J. Yuan, J. Qiu, J. Li, J. Song, K. Dong, K. Hu, K. Gao, K. Guan, K. Huang, K. Yu, L. Wang, L. Zhang, L. Xu, L. Xia, L. Zhao, L. Wang, L. Zhang, M. Li, M. Wang, M. Zhang, M. Zhang, M. Tang, M. Li, N. Tian, P. Huang, P. Wang, P. Zhang, Q. Wang, Q. Zhu, Q. Chen, Q. Du, R. J. Chen, R. L. Jin, R. Ge, R. Zhang, R. Pan, R. Wang, R. Xu, R. Zhang, R. Chen, S. S. Li, S. Lu, S. Zhou, S. Chen, S. Wu, S. Ye, S. Ye, S. Ma, S. Wang, S. Zhou, S. Yu, S. Zhou, S. Pan, T. Wang, T. Yun, T. Pei, T. Sun, W. L. Xiao, W. Zeng, W. Zhao, W. An, W. Liu, W. Liang, W. Gao, W. Yu, W. Zhang, X. Q. Li, X. Jin, X. Wang, X. Bi,

X. Liu, X. Wang, X. Shen, X. Chen, X. Zhang, X. Chen, X. Nie, X. Sun, X. Wang, X. Cheng, X. Liu, X. Xie, X. Liu, X. Yu, X. Song, X. Shan, X. Zhou, X. Yang, X. Li, X. Su, X. Lin, Y. K. Li, Y. Q. Wang, Y. X. Wei, Y. X. Zhu, Y. Zhang, Y. Xu, Y. Xu, Y. Huang, Y. Li, Y. Zhao, Y. Sun, Y. Li, Y. Wang, Y. Yu, Y. Zheng, Y. Zhang, Y. Shi, Y. Xiong, Y. He, Y. Tang, Y. Piao, Y. Wang, Y. Tan, Y. Ma, Y. Liu, Y. Guo, Y. Wu, Y. Ou, Y. Zhu, Y. Wang, Y. Gong, Y. Zou, Y. He, Y. Zha, Y. Xiong, Y. Ma, Y. Yan, Y. Luo, Y. You, Y. Liu, Y. Zhou, Z. F. Wu, Z. Z. Ren, Z. Ren, Z. Sha, Z. Fu, Z. Xu, Z. Huang, Z. Zhang, Z. Xie, Z. Zhang, Z. Hao, Z. Gou, Z. Ma, Z. Yan, Z. Shao, Z. Xu, Z. Wu, Z. Zhang, Z. Li, Z. Gu, Z. Zhu, Z. Liu, Z. Li, Z. Xie, Z. Song, Z. Gao, and Z. Pan, "Deepseek-v3 technical report," 2024.

[44] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," 2016.

[45] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2023.

[46] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," 2019.

[47] A. Radford and K. Narasimhan, "Improving language understanding by generative pre-training," 2018.

[48] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," 2021.

[49] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," 2013.

[50] J. Pennington, R. Socher, and C. Manning, "Glove: Global vectors for word representation," vol. 14, pp. 1532–1543, 01 2014.

[51] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," 2017.

[52] R. E. Wengert, "A simple automatic derivative evaluation program," *Communications of the ACM*, vol. 7, pp. 463 – 464, 1964.

[53] A. Griewank, *Who invented the reverse mode of differentiation?*, pp. 389–400. 01 2012.

[54] S. Linnainmaa, "Taylor expansion of the accumulated rounding error," *BIT*, vol. 16, p. 146–160, jun 1976.

[55] M. Bartholomew-Biggs, S. Brown, B. Christianson, and L. Dixon, "Automatic differentiation of algorithms," *Journal of Computational and Applied Mathematics*, vol. 124, no. 1, pp. 171–190, 2000. Numerical Analysis 2000. Vol. IV: Optimization and Nonlinear Equations.

[56] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.

[57] U. Naumann, "Optimal jacobian accumulation is np-complete," *Mathematical Programming*, vol. 112, pp. 427–441, 01 2008.

[58] C. Guo, Y. Qiu, J. Leng, C. Zhang, Y. Cao, Q. Zhang, Y. Liu, F. Yang, and M. Guo, "Nesting forward automatic differentiation for memory-efficient deep neural network training," 2022.

[59] M. Blondel and V. Roulet, "The elements of differentiable programming," 2024.

[60] H. Kato, D. Beker, M. Morariu, T. Ando, T. Matsuoka, W. Kehl, and A. Gaidon, "Differentiable rendering: A survey," 2020.

[61] R. Newbury, J. Collins, K. He, J. Pan, I. Posner, D. Howard, and A. Cosgun, "A review of differentiable simulators," 2024.

[62] P. Hudak, "Conception, evolution, and application of functional programming languages," *ACM Comput. Surv.*, vol. 21, p. 359–411, Sept. 1989.

[63] B. Jang, D. Schaa, P. Mistry, and D. Kaeli, "Exploiting memory access patterns to improve memory performance in data-parallel architectures," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 1, pp. 105–118, 2011.

[64] NumPY, "Broadcasting," 2024. `https://numpy.org/doc/stable/user/basics.broadcasting.html` [Accessed: (2024-07-13)].

[65] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, P. luc Cantin, C. Chao, C. Clark, J. Coriell, M. Daley, M. Dau, J. Dean, B. Gelb, T. V. Ghaemmaghami, R. Gottipati, W. Gulland, R. Hagmann, C. R. Ho, D. Hogberg, J. Hu, R. Hundt, D. Hurt, J. Ibarz, A. Jaffey, A. Jaworski, A. Kaplan, H. Khaitan, A. Koch, N. Kumar, S. Lacy, J. Laudon, J. Law, D. Le, C. Leary, Z. Liu, K. Lucke, A. Lundin, G. MacKean, A. Maggiore, M. Mahony, K. Miller, R. Nagarajan, R. Narayanaswami, R. Ni, K. Nix, T. Norrie, M. Omernick, N. Penukonda, A. Phelps, J. Ross, M. Ross, A. Salek, E. Samadiani, C. Severn, G. Sizikov, M. Snelham, J. Souter, D. Steinberg, A. Swing, M. Tan, G. Thorson, B. Tian, H. Toma, E. Tuttle, V. Vasudevan, R. Walter, W. Wang, E. Wilcox, and D. H. Yoon, "In-datacenter performance analysis of a tensor processing unit," 2017.

[66] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," 2017.

[67] P. Micikevicius, S. Narang, J. Alben, G. Diamos, E. Elsen, D. Garcia, B. Ginsburg, M. Houston, O. Kuchaiev, G. Venkatesh, and H. Wu, "Mixed precision training," 2018.

[68] M. Courbariaux, Y. Bengio, and J.-P. David, "Training deep neural networks with low precision multiplications," 2015.

[69] "Ieee standard for floating-point arithmetic," *IEEE Std 754-2008*, pp. 1–70, 2008.

[70] D. Kalamkar, D. Mudigere, N. Mellempudi, D. Das, K. Banerjee, S. Avancha, D. T. Vooturi, N. Jammalamadaka, J. Huang, H. Yuen, J. Yang, J. Park, A. Heinecke, E. Georganas, S. Srinivasan,

A. Kundu, M. Smelyanskiy, B. Kaul, and P. Dubey, "A study of bfloat16 for deep learning training," 2019.

[71] S. Wang and P. Kanwar, "Bfloat16: The secret to high performance on cloud tpus," 2019. https://cloud.google.com/blog/products/ai-machine-learning/bfloat16-the-secret-to-high-performance-on-cloud-tpus [Accessed: (2024-05-22)].

[72] H. Peng, K. Wu, Y. Wei, G. Zhao, Y. Yang, Z. Liu, Y. Xiong, Z. Yang, B. Ni, J. Hu, R. Li, M. Zhang, C. Li, J. Ning, R. Wang, Z. Zhang, S. Liu, J. Chau, H. Hu, and P. Cheng, "Fp8-lm: Training fp8 large language models," 2023.

[73] P. Micikevicius, D. Stosic, N. Burgess, M. Cornea, P. Dubey, R. Grisenthwaite, S. Ha, A. Heinecke, P. Judd, J. Kamalu, N. Mellempudi, S. Oberman, M. Shoeybi, M. Siu, and H. Wu, "Fp8 formats for deep learning," 2022.

[74] M. J. Flynn, "Some computer organizations and their effectiveness," *IEEE Transactions on Computers*, vol. C-21, no. 9, pp. 948–960, 1972.

[75] M. Flynn, "Very high-speed computing systems," *Proceedings of the IEEE*, vol. 54, no. 12, pp. 1901–1909, 1966.

[76] C. V. Ravishankar and J. R. Goodman, "Cache implementation for multiple microprocessors," *Proceedings of IEEE COMPCON*, p. 346–350, 02 1983.

[77] E. W. Dijkstra, "Solution of a problem in concurrent programming control," *Commun. ACM*, vol. 8, p. 569, Sept. 1965.

[78] E. W. Dijkstra, "Cooperating sequential processes (ewd 35)," 1972. Accessed: March 03, 2024.

[79] NVIDIA Corporation, "NVIDIA Tesla V100 GPU ARCHITECTURE: THE WORLD'S MOST ADVANCED DATA CENTER GPU," tech. rep., NVIDIA Corporation, 2017. Accessed: March 27, 2024.

[80] AMD, "Introducing AMD CDNA architecture: The all-new AMD GPU architecture for the modern era of HPC & AI," tech. rep., Advanced Micro Devices, Inc., 2020. Accessed: March 29, 2024.

[81] N. Nassif, A. O. Munch, C. L. Molnar, G. Pasdast, S. V. Lyer, Z. Yang, O. Mendoza, M. Huddart, S. Venkataraman, S. Kandula, R. Marom, A. M. Kern, B. Bowhill, D. R. Mulvihill, S. Nimmagadda, V. Kalidindi, J. Krause, M. M. Haq, R. Sharma, and K. Duda, "Sapphire rapids: The next-generation intel xeon scalable processor," in *2022 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 65, pp. 44–46, 2022.

[82] Google, "Highway: Performance-portable, length-agnostic simd with runtime dispatch," 2019.

[83] OpenXLA, "XLA: A machine learning compiler for gpus, cpus, and ml accelerators," 2018.

[84] LF AI Data Foundation, "ONNX: Open standard for machine learning interoperability," 2017.

[85] K. Goto and R. A. v. d. Geijn, "Anatomy of high-performance matrix multiplication," *ACM Trans. Math. Softw.*, vol. 34, May 2008.

[86] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *LAPACK Users' Guide*. Philadelphia, PA: Society for Industrial and Applied Mathematics, third ed., 1999.

[87] J. Nickolls, I. Buck, M. Garland, and K. Skadron, "Scalable parallel programming with cuda," in *ACM SIGGRAPH 2008 Classes*, SIGGRAPH '08, (New York, NY, USA), Association for Computing Machinery, 2008.

[88] J. Sanders and E. Kandrot, *CUDA by Example: An Introduction to General-Purpose GPU Programming*. Addison-Wesley Professional, 1st ed., 2010.

[89] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer, "cudnn: Efficient primitives for deep learning," 2014.

[90] NVIDIA Corporation, "cudnn frontend api," 2020.

[91] CCCL Development Team, *CCCL: CUDA Core Compute Libraries*, 2023.

[92] NVIDIA Corporation, "jitify: A single-header C++ library for simplifying the use of CUDA Runtime Compilation (NVRTC).," 2017.

[93] AMD ROCm™ Software, "HIP: C++ Heterogeneous-Compute Interface for Portability," 2016.

[94] R. Okuta, Y. Unno, D. Nishino, S. Hido, and C. Loomis, "Cupy: A numpy-compatible library for nvidia gpu calculations," in *Proceedings of Workshop on Machine Learning Systems (LearningSys) in The Thirty-first Annual Conference on Neural Information Processing Systems (NIPS)*, 2017.

[95] T. T. D. Team, R. Al-Rfou, G. Alain, A. Almahairi, C. Angermueller, D. Bahdanau, N. Ballas, F. Bastien, J. Bayer, A. Belikov, A. Belopolsky, Y. Bengio, A. Bergeron, J. Bergstra, V. Bisson, J. B. Snyder, N. Bouchard, N. Boulanger-Lewandowski, X. Bouthillier, A. de Brébisson, O. Breuleux, P.-L. Carrier, K. Cho, J. Chorowski, P. Christiano, T. Cooijmans, M.-A. Côté, M. Côté, A. Courville, Y. N. Dauphin, O. Delalleau, J. Demouth, G. Desjardins, S. Dieleman, L. Dinh, M. Ducoffe, V. Dumoulin, S. E. Kahou, D. Erhan, Z. Fan, O. Firat, M. Germain, X. Glorot, I. Goodfellow, M. Graham, C. Gulcehre, P. Hamel, I. Harlouchet, J.-P. Heng, B. Hidasi, S. Honari, A. Jain, S. Jean, K. Jia, M. Korobov, V. Kulkarni, A. Lamb, P. Lamblin, E. Larsen, C. Laurent, S. Lee, S. Lefrancois, S. Lemieux, N. Léonard, Z. Lin, J. A. Livezey, C. Lorenz, J. Lowin, Q. Ma, P.-A. Manzagol, O. Mastropietro, R. T. McGibbon, R. Memisevic, B. van Merriënboer, V. Michalski, M. Mirza, A. Orlandi, C. Pal, R. Pascanu, M. Pezeshki, C. Raffel, D. Renshaw, M. Rocklin, A. Romero, M. Roth, P. Sadowski, J. Salvatier, F. Savard, J. Schlüter, J. Schulman, G. Schwartz, I. V. Serban, D. Serdyuk, S. Shabanian, Étienne Simon, S. Spieckermann, S. R. Subramanyam, J. Sygnowski, J. Tanguay, G. van Tulder, J. Turian, S. Urban, P. Vincent, F. Visin, H. de Vries, D. Warde-Farley, D. J. Webb, M. Willson, K. Xu, L. Xue, L. Yao, S. Zhang, and Y. Zhang, "Theano: A python framework for fast computation of mathematical expressions," 2016.

[96] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," 2014.

[97] the tiny corp, "You like pytorch? you like micrograd? you love tinygrad!," 2020.

[98] R. Frostig, M. Johnson, and C. Leary, "Compiling machine learning programs via high-level tracing," 2018.

[99] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang, "JAX: composable transformations of Python+NumPy programs," 2018.

[100] J. Heek, A. Levskaya, A. Oliver, M. Ritter, B. Rondepierre, A. Steiner, and M. van Zee, "Flax: A neural network library and ecosystem for JAX," 2024.

[101] ONNX Runtime developers, "Onnx runtime." `https://onnxruntime.ai/`, 2018.

[102] F. Chollet *et al.*, "Keras." `https://keras.io`, 2015.

[103] S. K. Lam, A. Pitrou, and S. Seibert, "Numba: a llvm-based python jit compiler," in *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, LLVM '15, (New York, NY, USA), Association for Computing Machinery, 2015.