National Technical University of Athens
School of Electrical and Computer Engineering
division of signals, control and robotics

Effective Methods for Deep Neural Network Sparsification

DIPLOMA THESIS

of

Athanasios Glentis Georgoulakis

Supervisor: Petros Maragos
Professor NTUA

Co-supervisor: Georgios Retsinas
Postdoctoral Researcher NTUA

COMPUTER VISION, SPEECH COMMUNICATION AND SIGNAL PROCESSING GROUP
Athens, October 2023

National Technical University of Athens
School of Electrical and Computer Engineering
Division of Signals, Control and Robotics
Computer Vision, Speech Communication and Signal Processing Group

Effective Methods for Deep Neural Network Sparsification

DIPLOMA THESIS

of

Athanasios Glentis Georgoulakis

Supervisor: Petros Maragos
Professor NTUA

Co-supervisor: Georgios Retsinas
Postdoctoral Researcher NTUA

Approved by the Examining Committee on 20th October, 2023.

.....	.....	.....
Petros Maragos	Athanasios Rontogiannis	Gerasimos Potamianos
Professor NTUA	Associate Professor NTUA	Associate Professor UTH

Athens, October 2023

.....
Athanasios Glentis Georgoulakis
Graduate of Electrical and Computer Engineering NTUA

Abstract

In recent years, Deep Neural Networks (DNNs) have significantly advanced the state-of-the-art in numerous machine learning tasks. Unfortunately, most compact devices that rely on embedded computing systems with limited resources cannot support the deployment of such powerful DNNs. This has driven considerable research efforts towards creating compact and efficient versions of these models. A prominent method among the model compression literature is neural network pruning, involving the removal of unimportant network elements with the goal of obtaining compressed, yet highly effective models. In this thesis, we focus on removing individual weights based on their magnitudes encompassing the sparsification process during the standard course of training (sparse training), therefore avoiding multi-cycle training and fine-tuning procedures. In the first part of the thesis we propose a pruning solution that tackles the problem of sparsity allocation over the different layers of the DNN. Modeling the distributions of the weights per-layer in a novel way as Gaussian or Laplace enables the method to learn the pruning thresholds through the optimization process, resulting in an effective non-uniform sparsity allocation for a requested overall sparsity target. In the second part of this work, recognizing that the Straight-Through Estimator is a crucial component of the aforementioned method and of sparse training in general, we devote our efforts into improving its effectiveness. This leads to the introduction of *Feather*, a novel sparse training module utilizing the powerful Straight-Through Estimator as its core, coupled with a new thresholding operator and a gradient scaling technique that enable robust state-of-the-art sparsification performance. More specifically, the thresholding operator balances the currently used ones, namely the hard and soft operators, combining their advantages, while gradient scaling controls the sparsity pattern variations, leading to a more stable training procedure. Both proposed methods are tested on the CIFAR and ImageNet datasets for image classification using various architectures, resulting into state-of-the-art performances. In particular, Feather achieves Top-1 validation accuracies on ImageNet using the ResNet-50 architecture that surpass those obtained from existing methods, including more complex and computationally demanding ones, by a considerable margin.

Keywords — DNN-compression, sparsification, weight-pruning, unstructured-pruning, sparse-training, efficient-vision

Περίληψη

Τα τελευταία χρόνια, τα Βαθιά Νευρωνικά Δίκτυα (Deep Neural Networks - DNNs) έχουν βελτιώσει σημαντικά τις επιδόσεις πολλών συστημάτων μηχανικής μάθησης. Δυστυχώς, οι περισσότερες συσκευές που βασίζονται σε ενσωματωμένα υπολογιστικά συστήματα με περιορισμένους πόρους δεν μπορούν να υποστηρίξουν την ανάπτυξη τόσο ισχυρών μοντέλων. Αυτό έχει οδηγήσει σε σημαντικές ερευνητικές προσπάθειες για τη δημιουργία συμπαγών και αποδοτικών εκδόσεων αυτών των μοντέλων. Μια εξέχουσα μέθοδος για τη συμπίεση μοντέλων είναι το κλάδεμα των νευρωνικών δικτύων, που περιλαμβάνει την αφαίρεση μη σημαντικών στοιχείων του δικτύου με στόχο την απόκτηση συμπιεσμένων, αλλά ιδιαίτερα αποτελεσματικών μοντέλων. Στην παρούσα εργασία, εστιάζουμε στην αφαίρεση μεμονωμένων βαρών με βάση το μέτρο τους, περιλαμβάνοντας τη διαδικασία κλαδέματος κατά τη διάρκεια της τυπικής πορείας εκπαίδευσης (αραιή εκπαίδευση), αποφεύγοντας έτσι διαδικασίες εκπαίδευσης πολλών κύκλων και προσαρμογών. Στο πρώτο μέρος της εργασίας προτείνουμε μια λύση κλαδέματος που αντιμετωπίζει το πρόβλημα της κατανομής της αραιότητας στα διάφορα επίπεδα του δικτύου. Η μοντελοποίηση των κατανομών των βαρών ανά επίπεδο με έναν καινοτόμο τρόπο ως Gaussian ή Laplace επιτρέπει στη μέθοδο να μαθαίνει τα κατώφλια κλαδέματος μέσω της διαδικασίας βελτιστοποίησης, με αποτέλεσμα μια αποτελεσματική μη ομοιόμορφη κατανομή αραιότητας για έναν ζητούμενο συνολικό στόχο μηδενισμένων παραμέτρων. Στο δεύτερο μέρος αυτής της εργασίας, αναγνωρίζοντας ότι ο Straight-Through Estimator αποτελεί κρίσιμο συστατικό της προαναφερθείσας μεθόδου και της αραιής εκπαίδευσης γενικότερα, αφιερώνουμε τις προσπάθειές μας στη βελτίωση της αποτελεσματικότητάς του. Αυτό οδηγεί στην εισαγωγή του *Feather*, μιας νέας μονάδας αραιής εκπαίδευσης, που χρησιμοποιεί τον ισχυρό Straight-Through Estimator ως πυρήνα της, σε συνδυασμό με έναν νέο τελεστή κατωφλίωσης και μια τεχνική κλιμάκωσης των gradients που επιτρέπουν εύρωστες και κορυφαίες επιδόσεις αραιοποίησης. Πιο συγκεκριμένα, ο τελεστής κατωφλίωσης εξισορροπεί τους δύο που χρησιμοποιούνται σήμερα στην πράξη, δηλαδή τον hard και soft τελεστή, συνδυάζοντας τα πλεονεκτήματά τους, ενώ η κλιμάκωση των gradients ελέγχει τις μεταβολές των μοτίβων αραιότητας, οδηγώντας σε μια πιο σταθερή διαδικασία εκπαίδευσης. Και οι δύο προτεινόμενες μέθοδοι δοκιμάζονται στα σύνολα δεδομένων CIFAR και ImageNet για κατηγοριοποίηση εικόνων χρησιμοποιώντας διάφορες αρχιτεκτονικές, με αποτέλεσμα να επιτυγχάνονται κορυφαίες επιδόσεις. Ειδικότερα, το Feather επιτυγχάνει Top-1 ακρίβειες επικύρωσης στο ImageNet χρησιμοποιώντας την αρχιτεκτονική ResNet-50 που ξεπερνούν εκείνες που επιτυγχάνονται από τις υπάρχουσες μεθόδους, συμπεριλαμβανομένων των πιο σύνθετων και υπολογιστικά απαιτητικών, με σημαντική διαφορά.

Λέξεις Κλειδιά — συμπίεση-DNN, αραιοποίηση, κλάδεμα-βαρών, μη-δομημένο-κλάδεμα, αραιή-εκπαίδευση, αποδοτική-όραση

Acknowledgments

To begin with, I would like to thank Professor Petros Maragos for giving me the opportunity to conduct my Diploma thesis in the Computer Vision, Speech Communication and Signal Processing Laboratory. Through his teaching of undergraduate courses at the National Technical University of Athens and his research work he inspired me to actively pursue this research path in the fields of Machine Learning and Computer Vision. Moreover, throughout my thesis he provided me with valuable guidance and advice that contributed to its successful completion.

Also, I would like to thank Dr. Georgios Retsinas, Postdoctoral Researcher at the National Technical University of Athens, for our constructive collaboration. The continuous support he offered me both in research and technical issues played a crucial role in the completion of my thesis.

I am particularly excited that in parallel with my Diploma thesis I published my first research paper in an international conference with co-authors Dr. Georgios Retsinas and Prof. Petros Maragos.

Finally, I would like to thank my family for their understanding and support throughout my studies and my friends for the great experiences we had and for our constructive discussions about scientific issues.

Athanasios Glentis Georgoulakis
October 2023

Ευχαριστίες

Αρχικά, θα ήθελα να ευχαριστήσω τον Καθηγητή κ. Πέτρο Μαραγκό που μου έδωσε την ευκαιρία να εκπονήσω τη Διπλωματική μου εργασία στο Εργαστήριο Όρασης Υπολογιστών, Επικοινωνίας Λόγου και Επεξεργασίας Σημάτων. Μέσα από τη διδασκαλία των προπτυχιακών του μαθημάτων στο Εθνικό Μετσόβιο Πολυτεχνείο αλλά και από το ερευνητικό του έργο με ενέπνευσε να ακολουθήσω ενεργά τη συγκεκριμένη ερευνητική πορεία στους τομείς της Μηχανικής Μάθησης και Όρασης Υπολογιστών. Επιπλέον, καθ' όλη τη διάρκεια αυτού του ερευνητικού μου έργου μου παρείχε πολύτιμη καθοδήγηση και συμβουλές που συνέβαλαν στην επιτυχή διεκπεραίωση του.

Στη συνέχεια, θα ήθελα να ευχαριστήσω τον Δρ. Γεώργιο Ρετσινά, μεταδιδακτορικό ερευνητή του Εθνικού Μετσόβιου Πολυτεχνείου, για την εποικοδομητική συνεργασία μας. Η συνεχής υποστήριξη που μου προσέφερε τόσο σε ερευνητικά όσο και σε τεχνικά ζητήματα έπαιξαν καθοριστικό ρόλο στην ολοκλήρωση της Διπλωματικής μου εργασίας.

Είμαι ιδιαίτερα ενθουσιασμένος που παράλληλα με τη Διπλωματική μου εργασία δημοσίευσα σε διεθνές συνέδριο την πρώτη μου ερευνητική εργασία με συν-συγγραφείς τον Δρ. Γεώργιο Ρετσινά και τον Καθηγητή κ. Πέτρο Μαραγκό.

Τέλος, ευχαριστώ την οικογένειά μου για την κατανόηση και την υποστήριξη που μου προσέφεραν καθ' όλη τη διάρκεια των σπουδών μου, αλλά και τους φίλους μου για τις ωραίες εμπειρίες που περάσαμε και για τις εποικοδομητικές συζητήσεις μας αναφορικά με επιστημονικά ζητήματα.

Αθανάσιος Γλεντής Γεωργουλάκης
Οκτώβριος 2023

Contents

List of Figures

List of Tables

Chapter 1

Εκτεταμένη Περίληψη στα Ελληνικά

1.1 Εισαγωγή

Τα Βαθιά Νευρωνικά Δίκτυα έχουν επιδείξει εντυπωσιακές ικανότητες στην επίλυση διαφόρων σύνθετων προβλημάτων μηχανικής μάθησης, με τις επιδόσεις τους να βελτιώνονται συνεχώς χάρη στις προσπάθειες της ερευνητικής κοινότητας [33, 48, 25]. Για να επιτευχθεί αυτό, τα μοντέλα νευρωνικών δικτύων έχουν γίνει μεγαλύτερα, βαθύτερα και πιο πολύπλοκα και συνεπώς απαιτούν σημαντικούς πόρους για την εκπαίδευση και την εξαγωγή συμπερασμάτων, συμπεριλαμβανομένων μεγάλων ποσοτήτων μνήμης συστήματος για την αποθήκευση των παραμέτρων και των ενδιάμεσων υπολογισμών, σημαντικής υπολογιστικής ισχύος προκειμένου να διατηρούνται λογικοί χρόνοι επεξεργασίας, καθώς και άφθονη παροχή ενέργειας στα εμπλεκόμενα συστήματα [14].

Δυστυχώς, η ραγδαία αύξηση των απαιτήσεων σε πόρους των DNNs τελευταίας τεχνολογίας δεν έχει καλυφθεί από αντίστοιχη πρόοδο στο υλικό (επεξεργαστές και μνήμες) που χρησιμοποιούν οι φορητές συσκευές, οι οποίες περιορίζονται από το μικρό μέγεθος και τις χαμηλές ενεργειακές απαιτήσεις [14]. Κατά συνέπεια, πολλές εφαρμογές του πραγματικού κόσμου, όπως στη ρομποτική, στα έξυπνα wearables, στα αυτοκίνητα με αυτόνομη οδήγηση και στις έξυπνες πόλεις, μεταξύ άλλων, που βασίζονται σε συστήματα μηχανικής μάθησης που λειτουργούν σε συσκευές με περιορισμένους πόρους, δεν μπορούν να επωφεληθούν από τις βελτιωμένες δυνατότητες αυτών των δικτύων.

Ως απάντηση, τα τελευταία χρόνια έχει καταβληθεί μεγάλη προσπάθεια για τη δημιουργία συμπαγών και αποδοτικών (δηλαδή συμπιεσμένων) εκδόσεων αυτών των DNNs [14, 57, 26]. Μεταξύ των πολυάριθμων προσεγγίσεων συμπίεσης μοντέλων, το κλάδεμα [54, 21, 57] έχει μελετηθεί ευρέως και έχει δείξει πολύ ελπιδοφόρα αποτελέσματα [21, 73, 97, 84]. Περιλαμβάνει την αφαίρεση στοιχείων του δικτύου (όπως μεμονωμένα βάρη ή ολόκληρα φίλτρα και κανάλια) που θεωρούνται περιττά και έχουν μικρή ή καθόλου επίδραση στην τελική του απόδοση. Παραδόξως, έχει παρατηρηθεί ότι τα αρχικά μεγάλα πυκνά μοντέλα που έχουν γίνει αραιά μέσω κλαδέματος υπερτερούν έναντι των μικρότερων πυκνών μοντέλων που έχουν τον ίδιο αριθμό μη μηδενικών παραμέτρων, παρουσιάζοντας συχνά καθόλου απώλειες ακρίβειας έως και μέτριους λόγους συμπίεσης [98, 73]. Στην ουσία, αυτό το φαινόμενο, αν και δεν έχει ακόμη μια εμπεριστατωμένη θεωρητική κατανόηση, αποδίδεται γενικά στην επίδραση που έχει η υπερπαραμετροποίηση στη δυνατότητα αποτελεσματικότερης εκπαίδευσης μέσω Stochastic Gradient Decent [94, 7], καθώς και στους επιπλέον βαθμούς ελευθερίας που εισάγονται όσον αφορά την κατανομή της αραιότητας [67, 84].

Πρόσφατα, υπήρξε ιδιαίτερο ενδιαφέρον στην κοινότητα συμπίεσης των DNNs για την ανάπτυξη μεθόδων που εκτελούν αποτελεσματικά το κλάδεμα εντός της τυπικής πορείας εκπαίδευσης [73, 44, 84, 86], δηλαδή χωρίς την ανάγκη για περαιτέρω γύρους επανεκπαίδευσης και κλαδέματος. Στην παρούσα Διπλωματική Εργασία, στηριζόμενοι στο σύνολο ερευνητικών έργων της περιοχής αυτής, εστιάζουμε στην αφαίρεση μεμονωμένων βαρών με βάση τα μεγέθη τους κατά την εκπαίδευση.

1.2 Μηχανική Μάθηση

Με τον όρο μηχανική μάθηση [66, 4] αναφερόμαστε σε ένα εκτεταμένο φάσμα μεθόδων που χρησιμοποιούνται για τον προγραμματισμό υπολογιστών ώστε να ανακαλύπτουν μοτίβα και να λαμβάνουν αποφάσεις βάσει δεδομένων. Αυτή η προσέγγιση με βάση τα δεδομένα έχει επιτρέψει στους υπολογιστές να διαπρέψουν σε διάφορους τομείς, όπως στην όραση, που θα ήταν ανέφικτο να προγραμματιστούν ρητά για να λειτουργήσουν με τον επιθυμητό τρόπο. Για την επίλυση μιας εργασίας με τη χρήση μιας μεθόδου μηχανικής μάθησης απαιτείται ένα σύνολο δεδομένων. Αυτό το σύνολο δεδομένων μπορεί να παρουσιαστεί στον υπολογιστή με διάφορους τρόπους, ανάλογα με την εφαρμογή, οδηγώντας κυρίως σε τρία διαφορετικά παραδείγματα μηχανικής μάθησης, την επιβλεπόμενη μάθηση, τη μη επιβλεπόμενη μάθηση και την ενισχυτική μάθηση.

Επιβλεπόμενη Μάθηση

Μακράν η πιο συνηθισμένη προσέγγιση στη μηχανική μάθηση ονομάζεται μάθηση με επίβλεψη. Χαρακτηρίζεται από τη χρήση επισημασμένων συνόλων δεδομένων, όπου η επιθυμητή έξοδος (ετικέτα) αντιστοιχίζεται με κάθε αντίστοιχο δείγμα δεδομένων. Αυτές οι ετικέτες χρησιμοποιούνται στη συνέχεια για να "διδάξουν" ή να "επιβλέψουν" το σύστημα να επιστρέφει τις σωστές εξόδους για δεδομένα που δέχεται ως είσοδο. Πιο συγκεκριμένα, ένας αλγόριθμος μάθησης με επίβλεψη στοχεύει στην εκμάθηση μιας αντιστοίχισης μεταξύ των εισόδων x του προβλήματος και των εξόδων y, που παρέχονται με ένα σύνολο δεδομένων, $\{(x_1, y_1), ..., (x_N, y_N)\}$, με x_i και y_i να είναι το i-οστό δείγμα εισόδου (που συνήθως αναφέρεται ως διάνυσμα χαρακτηριστικών) και η i-οστή

έξοδος ή πρόβλεψη, αντίστοιχα. Τα προβλήματα μάθησης με επίβλεψη διακρίνονται περαιτέρω σε προβλήματα ταξινόμησης εάν οι έξοδοι y είναι διακριτές ή σε προβλήματα παλινδρόμησης εάν οι έξοδοι λαμβάνουν συνεχείς τιμές.

1.2.1 Έννοιες Μηχανικής Μάθησης

Συνάρτηση Σφάλματος

Οι τυπικοί αλγόριθμοι μηχανικής μάθησης χρησιμοποιούν διαδικασίες βελτιστοποίησης με βάση την κλίση [75] προκειμένου να προσαρμόσουν τις παραμέτρους του μοντέλου θ, ώστε να αναπαραστήσουν καλύτερα τα δεδομένα. Αυτό επιτυγχάνεται με την ελαχιστοποίηση μιας βαθμωτής συνάρτησης $L(\theta)$ (σχεδόν παντού) διαφορίσιμης ως προς τις παραμέτρους, γνωστή ως συνάρτηση σφάλματος, η οποία μετρά το σφάλμα του μοντέλου σε ένα σύνολο δεδομένων με βάση την απόσταση των ετικετών y και των εξόδων του μοντέλου $\hat{y} = f_\theta(x)$. Ανάλογα με την εφαρμογή, υπάρχουν πολυάριθμες παραστάσεις της συνάρτησης σφάλματος, όπως η συνάρτηση Μέσου Τετραγωνικού Σφάλματος (MSE) και η συνάρτηση Μέσου Απόλυτου Σφάλματος (MAE) για παλινδρόμηση και η συνάρτηση Διασταυρωμένης Εντροπίας (Cross Entropy) για ταξινόμηση, μεταξύ άλλων.

Μετρικές αξιολόγησης

Μια συνάρτηση που χρησιμοποιείται για την αξιολόγηση της απόδοσης ενός μοντέλου μηχανικής μάθησης, είτε σε ένα ενδιάμεσο βήμα βελτιστοποίησης είτε μετά τη διαδικασία εκπαίδευσης, ονομάζεται μετρική. Αν και οποιαδήποτε συνάρτηση σφάλματος μπορεί να χρησιμοποιηθεί ως μετρική, η διαφορισιμότητα των μετρικών συναρτήσεων (ως προς τις παραμέτρους του μοντέλου) δεν αποτελεί απαίτηση, καθώς δεν χρησιμοποιούνται άμεσα για τη βελτιστοποίηση του μοντέλου. Επομένως, ανάλογα με τη φύση του προβλήματος, μπορεί να επιλεγεί μια μεγάλη ποικιλία συναρτήσεων για το σκοπό της αξιολόγησης του μοντέλου.

1.3 Βαθιά Μάθηση

Έχοντας καλύψει τα βασικά στοιχεία της μηχανικής μάθησης, θα εστιάσουμε τώρα την προσοχή μας στα μοντέλα και τις μεθόδους βαθιάς μάθησης [53, 25], καθώς αυτό είναι το κύριο πεδίο εφαρμογής των διαφόρων τεχνικών συμπίεσης μοντέλων, συμπεριλαμβανομένου του κλαδέματος. Το σημαντικότερο πλεονέκτημα των μεθόδων βαθιάς μάθησης είναι ότι μπορούν να λαμβάνουν ως είσοδο ακατέργαστα δεδομένα (π.χ. τιμές εικονοστοιχείων εικόνας) και να επιτυγχάνουν εντυπωσιακά αποτελέσματα, επιτρέποντας τη χρήση της μηχανικής μάθησης για εφαρμογές που η χειροκίνητη εξαγωγή χαρακτηριστικών θα ήταν δύσκολη και πολύ λιγότερο αποτελεσματική. Αυτό έρχεται σε αντίθεση με τις παραδοσιακές μεθόδους μηχανικής μάθησης, οι οποίες απαιτούσαν πρώτα τη χειροκίνητη μετατροπή των ακατέργαστων δεδομένων σε χαρακτηριστικά που το σύστημα μπορούσε να χειριστεί αποτελεσματικά. Ο τρόπος με τον οποίο τα μοντέλα βαθιάς μάθησης, που ονομάζονται Βαθιά Νευρωνικά Δίκτυα (DNNs), επεξεργάζονται με επιτυχία τα ακατέργαστα δεδομένα εισόδου είναι εκτελώντας αυτόματα το έργο της εξαγωγής χαρακτηριστικών κατά την εκπαίδευσή τους σε μεγάλα σύνολα δεδομένων (συνήθως πολύ μεγαλύτερα από αυτά που απαιτούνται για τις παραδοσιακές μεθόδους μηχανικής μάθησης), όντας σε θέση να μαθαίνουν χαρακτηριστικά με διαφορετικά επίπεδα αφαίρεσης λόγω του τρόπου κατασκευής τους.

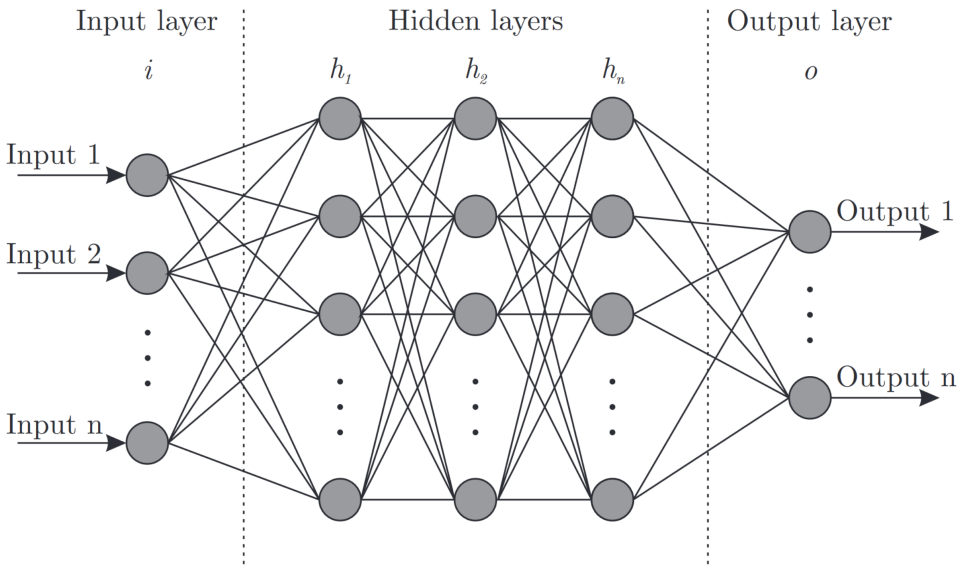
1.3.1 Feedforward Νευρωνικά Δίκτυα

Ίσως τα πιο δημοφιλή μοντέλα βαθιάς μάθησης είναι τα Feedforward Νευρωνικά Δίκτυα (FFNN), με τον όρο feedforward να υποδηλώνει ότι τα σήματα ρέουν από το στρώμα εισόδου στο στρώμα εξόδου, μέσω των ενδιάμεσων κρυφών στρωμάτων, χωρίς τη χρήση συνδέσεων ανατροφοδότησης. Μαθηματικά, ένα FFNN μπορεί να θεωρηθεί ως μια συνάρτηση $f(\boldsymbol{x}, \boldsymbol{W})$ που, δεδομένου του σήματος εισόδου $\boldsymbol{x}$ και ενός πίνακα βαρών προς μάθηση, $\boldsymbol{W}$, παράγει μια έξοδο $\boldsymbol{y}$. Δύο πολύ συνηθισμένοι τύποι FFNNs είναι τα Πλήρως Συνδεδεμένα Νευρωνικά Δίκτυα (FCNN) και τα Συνελικτικά Νευρωνικά Δίκτυα (CNNs), τα οποία αναλύονται συνοπτικά ακολούθως.

Πλήρως Συνδεδεμένα Νευρωνικά Δίκτυα:

Αναφέρονται και ως Πολυεπίπεδα Perceptrons (MLPs), είναι οι απλούστεροι τύποι FFNNs και αποτελούνται από έναν αριθμό πλήρως συνδεδεμένων στρωμάτων, όπου κάθε νευρώνας σε ένα στρώμα συνδέεται με κάθε νευρώνα του επόμενου στρώματος. Ο αριθμός των νευρώνων ενός στρώματος ονομάζεται πλάτος, ενώ διαφορετικά

στρώματα μπορούν να έχουν διαφορετικό πλάτος, με το πλάτος των στρωμάτων εισόδου και εξόδου να είναι φυσικά ίσο με τις διαστάσεις των διανυσμάτων εισόδου και εξόδου, αντίστοιχα. Από την άλλη πλευρά, ο αριθμός των στρωμάτων ενός μοντέλου ορίζεται ως το βάθος του. Μια γραφική αναπαράσταση ενός πλήρως συνδεδεμένου δικτύου απεικονίζεται στο Σχήμα 1.3.1.



Σχήμα 1.3.1: Γραφική αναπαράσταση ενός πλήρους συνδεδεμένου νευρωνικού δικτύου. Από [6].

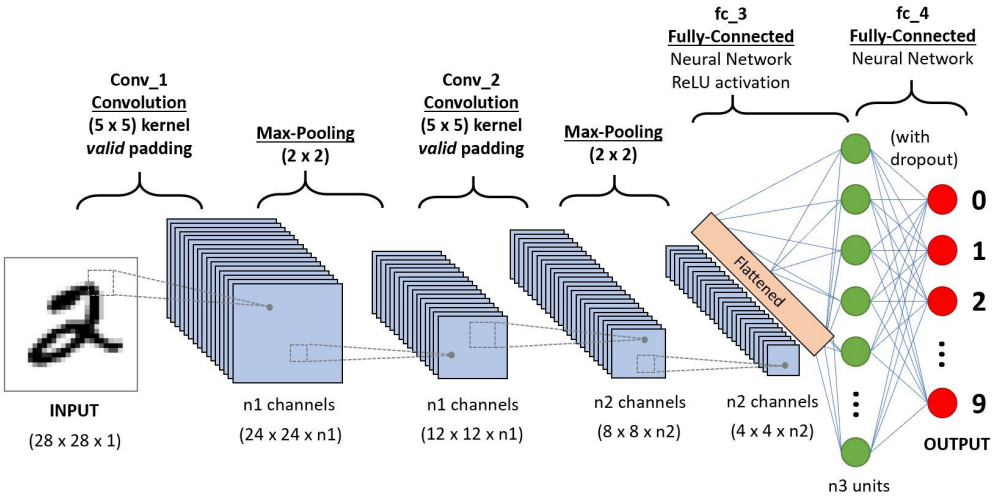
Συνελικτικά Νευρωνικά Δίκτυα:

Τα Συνελικτικά Νευρωνικά Δίκτυα (CNNs) ανήκουν στην υποκατηγορία των feedforward νευρωνικών δικτύων που είναι ειδικά βελτιστοποιημένα για την επεξεργασία δεδομένων που έχουν τοπολογία που μοιάζει με πλέγμα, όπως οι εικόνες. Όπως και τα πλήρως συνδεδεμένα δίκτυα, αποτελούνται επίσης από πολλαπλούς νευρώνες οι οποίοι αποτελούνται από εκπαιδεύσιμες παραμέτρους. Λαμβάνουν ως είσοδο ακατέργαστα δεδομένα που είναι συνήθως εικονοστοιχεία και παράγουν τα αποτελέσματα πρόβλεψης ως έξοδο. Καθώς σχεδιάζονται με βάση την υπόθεση ότι τους παρέχονται ρητά εικόνες ως είσοδοι, τα CNN μπορούν να εξάγουν χωρικά χαρακτηριστικά και να αναγνωρίζουν κρίσιμα δομικά πρότυπα της εισόδου τους. Ως συνέπεια της αποτελεσματικής κωδικοποίησης σημαντικών ιδιοτήτων της εικόνας, μπορούν να υλοποιηθούν πιο αποτελεσματικά για την εκτέλεση του εμπρόσθιου περάσματος, ενώ απαιτούν εξαιρετικά μειωμένο αριθμό παραμέτρων, σε σύγκριση με τα τυπικά νευρωνικά δίκτυα.

Συνήθως η αρχιτεκτονική ενός CNN αποτελείται από πολλαπλά στρώματα συνέλιξης, pooling και ενδεχομένως και στρωμάτων batch normalization, ακολουθούμενων από έναν αριθμό πλήρως συνδεδεμένων στρωμάτων. Στο Σχήμα 1.3.2 παραθέτουμε ένα παράδειγμα αρχιτεκτονικής CNN που αποτελείται από τα προαναφερθέντα στρώματα.

1.4 Συμπίεση Βαθιών Νευρωνικών Δικτύων

Τα τελευταία χρόνια, τα βαθιά νευρωνικά δίκτυα έχουν γίνει η πλέον σύγχρονη προσέγγιση (SoA) στην αντιμετώπιση πολλών πολύπλοκων προβλημάτων μηχανικής μάθησης, σε τομείς όπως η όραση υπολογιστών, η επεξεργασία φυσικής γλώσσας, η επεξεργασία ομιλίας και ήχου και η ρομποτική [33, 33, 48]. Αν και η εφαρμογή των DNNs σε προβλήματα των προαναφερθέντων πεδίων μπορεί να οδηγήσει σε δραματικές βελτιώσεις των επιδόσεων σε σύγκριση με τη χρήση παραδοσιακών μεθόδων μηχανικής μάθησης, αυτό έχει ως κόστος τη σημαντική αύξηση της υπολογιστικής πολυπλοκότητας. Αυτό αποδίδεται στην τάση των DNNs να βασίζονται στην ύπαρξη πολύ μεγάλων συνόλων παραμέτρων προς μάθηση, που συχνά αριθμούν δεκάδες ή εκατοντάδες εκατομμύρια, και συνεπώς απαιτούν τεράστια ποσά μνήμης και υπολογιστικών πόρων κατά την εκπαίδευση και το inference [14]. Τέτοιες μεγάλες απαιτήσεις σε πόρους εμποδίζουν την χρήση των συστημάτων που βασίζονται στη βαθιά μάθηση σε συσκευές με περιορισμένους πόρους, όπως κινητά τηλέφωνα, φορητές συσκευές, έξυπνα



Σχήμα 1.3.2: Μια απεικόνιση της αρχιτεκτονικής ενός μοντέλου CNN. Από [80].

ρομπότ και πολλές άλλες έξυπνες φορητές συσκευές που βασίζονται σε ενσωματωμένα υπολογιστικά συστήματα με περιορισμένους πόρους επεξεργασίας, μνήμης και ισχύος.

Αναγνωρίζοντας τις μεγάλες προοπτικές της ανάπτυξης μοντέλων βαθιάς μάθησης σε φορητές συσκευές, τα τελευταία χρόνια έχει αυξηθεί το ερευνητικό ενδιαφέρον για τη συμπίεση και την επιτάχυνση των DNNs με τη χρήση διαφόρων τεχνικών [14, 57, 26]. Με τη χρήση συμπιεσμένων μοντέλων DNNs, μπορούν να επιτευχθούν ορισμένα σημαντικά πλεονεκτήματα, τα οποία οδηγούν προς την κατεύθυνση της χρήσης των DNNs σε περιβάλλοντα με περιορισμένους πόρους, μερικά από τα πιο σημαντικά να είναι οι μειωμένες απαιτήσεις μνήμης για τις διάφορες παραμέτρους του μοντέλου, οι χαμηλότερες απαιτήσεις σε FLOPs κατά τη διάρκεια του inference και η εξοικονόμηση ενέργειας. Μια πολύ δημοφιλής τεχνική συμπίεσης που έχει μελετηθεί εκτενώς τα τελευταία χρόνια είναι το κλάδεμα [5], η οποία είναι η διαδικασία αφαίρεσης παραμέτρων του δικτύου με βάση κάποιο κριτήριο, με στόχο τη μείωση του μεγέθους του δικτύου (δηλαδή του συνολικού αριθμού των παραμέτρων του), διατηρώντας παράλληλα την απώλεια απόδοσης όσο το δυνατόν χαμηλότερη. Η προσέγγιση αυτή, που αποτελεί το επίκεντρο της παρούσας εργασίας, θα εξηγηθεί αναλυτικότερα στην Ενότητα 1.4.5, όπου θα αναλυθούν οι διάφορες κατηγορίες κλαδέματος. Επιπλέον, θα παρουσιαστούν εν συντομία διάφορες δημοφιλείς μέθοδοι συμπίεσης, οι οποίες είναι η κβάντιση [57], η αποσύνθεση τανυστών [43, 68], η απόσταξη γνώσης [26] και ο σχεδιασμός συμπαγών μοντέλων [14], έτσι ώστε ο αναγνώστης να έχει μια ευρύτερη κατανόηση των διαφόρων προσεγγίσεων συμπίεσης.

1.4.1 Κβάντιση

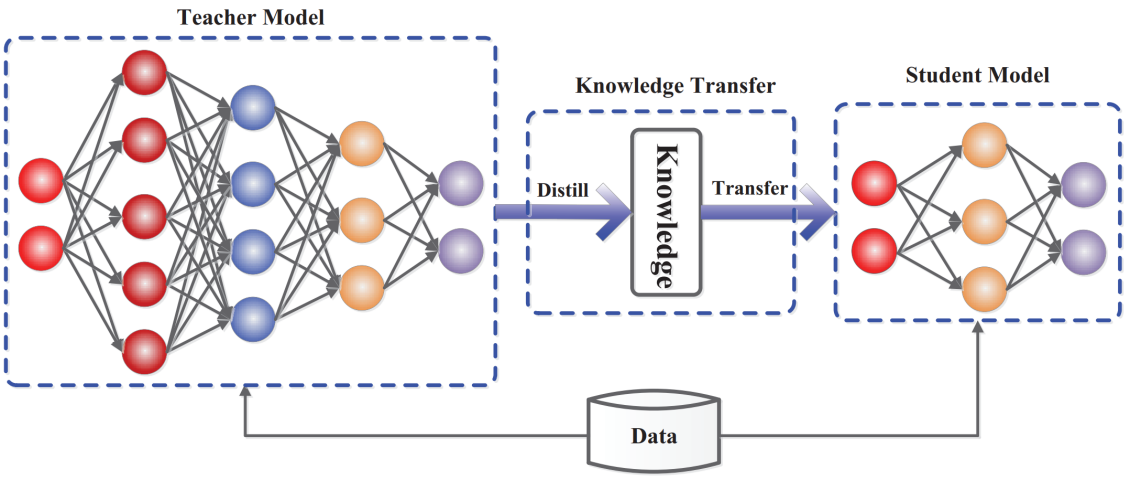
Η κβάντιση είναι η διαδικασία αναπαράστασης των τιμών ενός συνεχούς σήματος χρησιμοποιώντας ένα μικρό σύνολο διακριτών συμβόλων ή ακέραιων τιμών. Στα DNNs, τα κβαντισμένα στοιχεία αντιστοιχούν συνήθως σε βάρη και biases, τιμές των συναρτήσεων ενεργοποίησης ή τιμές του gradient. Οι τεχνικές κβάντισης για τη συμπίεση των DNNs μπορούν να χωριστούν σε γενικές γραμμές σε δύο κατηγορίες, την αριθμητική κβάντιση χαμηλών bit [89, 81] και τις τεχνικές μερικής κβάντισης που βασίζονται κυρίως στη χρήση κοινών βαρών που υπολογίζονται μέσω τεχνικών ομαδοποίησης [28].

1.4.2 Αποσύνθεση Τανυστών

Η εφαρμογή τεχνικών αποσύνθεσης τανυστών (συμπεριλαμβανομένων και πινάκων) στους τανυστές των βαρών έχει μελετηθεί εκτενώς ως ένας τρόπος συμπίεσης και επιτάχυνσης μοντέλων νευρωνικών δικτύων. Τέτοιες μέθοδοι αποσυνθέτουν τους πολυδιάστατους τανυστές βαρών σε προσεγγίσεις χαμηλής τάξης, με στόχο την αφαίρεση περιττών παραμέτρων και την εξοικονόμηση υπολογιστικού χρόνου. Πολλαπλές τεχνικές και αλγόριθμοι αποσύνθεσης έχουν χρησιμοποιηθεί για το στόχο αυτό, με μερικές από τις πιο δημοφιλείς στη σχετική

βιβλιογραφία να είναι η Singular Value Decomposition (SVD) [62, 68], η Canonical Polyadic (CP) decomposition [52, 83] ή η εκμάθηση λεξικών [74].

1.4.3 Απόσταξη Γνώσης



Σχήμα 1.4.1: Μια απεικόνιση της προσέγγισης με το μοντέλο δασκάλου-μαθητή για την απόσταξη της γνώσης. Από [26].

Μια προσέγγιση συμπίεσης μοντέλων που περιλαμβάνει τη διδασκαλία ενός μικρού μοντέλου, το οποίο ονομάζεται μοντέλο μαθητής, για να εκτελέσει μια εργασία (π.χ. ταξινόμηση εικόνων) με βάση τις απαντήσεις (ή τη γνώση) ενός μεγαλύτερου προ-εκπαιδευμένου μοντέλου (ή ενός συνόλου μοντέλων), το οποίο ονομάζεται μοντέλο δάσκαλος, είναι γνωστή ως Απόσταξη Γνώσης (Knowledge Distillation - KD). Το γενικό σχήμα KD απεικονίζεται γραφικά στο Σχήμα 1.4.1. Η ιδέα προτάθηκε αρχικά από τους Buciluǎ *et al.* [8] και γενικεύτηκε περαιτέρω από το έργο των Hinton *et al.* [32], όπου προτάθηκε ένα σύστημα KD το οποίο παράγει συμπαγή μοντέλα που υπερτερούν έναντι αυτών που εκπαιδεύονται από το μηδέν (χωρίς KD).

1.4.4 Σχεδιασμός Συμπαγών Μοντέλων

Αν και δεν πρόκειται για τεχνική συμπίεσης μοντέλων με την αυστηρή έννοια, ο σχεδιασμός συμπαγών μοντέλων από το μηδέν που επιτυγχάνουν αποδεκτή ακρίβεια είναι ένας απλός τρόπος για να καταστεί περαιτέρω δυνατή η χρήση της βαθιάς μάθησης σε κινητές συσκευές και εφαρμογές με περιορισμένους πόρους. Ευρέως χρησιμοποιούμενες αρχιτεκτονικές συμπαγών μοντέλων περιλαμβάνουν το MobileNet [35], το SqueezeNet [38] και το DenseNet [36], μεταξύ άλλων.

1.4.5 Κλάδεμα

Το κλάδεμα (pruning) είναι μια από τις πιο δημοφιλείς τεχνικές για τη δραστική μείωση του μεγέθους και την επιτάχυνση του inference των μοντέλων βαθιάς μάθησης. Βασίζεται στη διαπίστωση ότι τα DNNs τείνουν να είναι σε μεγάλο βαθμό υπερ-παραμετροποιημένα και έτσι οι μη σημαντικές παράμετροι του δικτύου μπορούν να αφαιρεθούν, με βάση κάποιο κριτήριο κατάταξης, με μικρή έως μηδενική επίδραση στην απόδοση του μοντέλου. Η μέθοδος χρονολογείται από το 1990 και πριν, με πρωτοποριακές εργασίες όπως αυτές των LeCun *et al.* [54] και Hassibi και Stork [30] και έχει κερδίσει μεγάλη ερευνητική προσοχή τα τελευταία χρόνια ως ένας τρόπος καταπολέμησης του συνεχώς αυξανόμενου μεγέθους των νευρωνικών δικτύων [14]. Η μεθοδολογία κλαδέματος, η οποία έχει εξελιχθεί με τα χρόνια, μπορεί να κατηγοριοποιηθεί σε γενικές γραμμές ως προς την δομή των στοιχείων του δικτύου που αραιώνονται, το κριτήριο κλαδέματος που χρησιμοποιείται για την κατάταξη της σημαντικότητας των στοιχείων που είναι υποψήφια για κλάδεμα και με βάση το χρονικό διάστημα που η αραιότητα εισάγεται στο μοντέλο. Οι προαναφερθείσες κατηγορίες κλαδέματος αναλύονται στις επόμενες ενότητες.

Κριτήρια Κλαδέματος

Κλάδεμα με Βάση το Μέτρο

Ένα πολύ απλό και ευρέως αποδεκτό ευρετικό κριτήριο για τον προσδιορισμό της σημασίας των βαρών του δικτύου βασίζεται στις τιμές του μέτρου τους [57]. Ακολουθώντας αυτό το κριτήριο, τα βάρη με μικρά μέτρα θεωρούνται λιγότερο σημαντικά για την έξοδο του δικτύου από αυτά με μεγάλα μέτρα. Σύμφωνα με αυτό το σκεπτικό, για την περίπτωση που τα στοιχεία που αραιώνονται είναι μεμονωμένα βάρη (μη δομημένο κλάδεμα), τα βάρη που βρίσκονται κάτω από κάποια τιμή κατωφλίου μέτρου, T, αφαιρούνται από το δίκτυο, ενώ τα υπόλοιπα διατηρούν μια μη μηδενική τιμή.

Κλάδεμα Βασιζόμενο στον Εσσιανό Πίνακα

Η χρήση παραγώγων δεύτερης τάξης (Εσσιανός πίνακας) της συνάρτησης σφάλματος (ως προς τις παραμέτρους του δικτύου) για τον προσδιορισμό της σημασίας κάθε παραμέτρου έχει διερευνηθεί εκτενώς, ήδη από το 1990. Αν και οι μέθοδοι κλαδέματος με βάση τον Εσσιανό πίνακα προτάθηκαν τότε ως μια πιο ακριβής προσέγγιση κλαδέματος από το κλάδεμα μέτρου, εφαρμόστηκαν σε πρώιμες αρχιτεκτονικές νευρωνικών δικτύων που ήταν πολύ πιο ρηχές και με σημαντικά λιγότερες παραμέτρους από τα σημερινά DNNs. Εξαιτίας αυτού, ο υπολογισμός του Εσσιανού πίνακα για τα περισσότερα DNNs που χρησιμοποιούνται σήμερα δεν είναι εφικτός, με αποτέλεσμα οι πιο πρόσφατες μέθοδοι κλαδέματος με βάση τον Εσσιανό πίνακα να βασίζονται σε τεχνικές προσέγγισης χαμηλού κόστους [88, 78, 92].

Κλάδεμα με βάση την εξομάλυνση

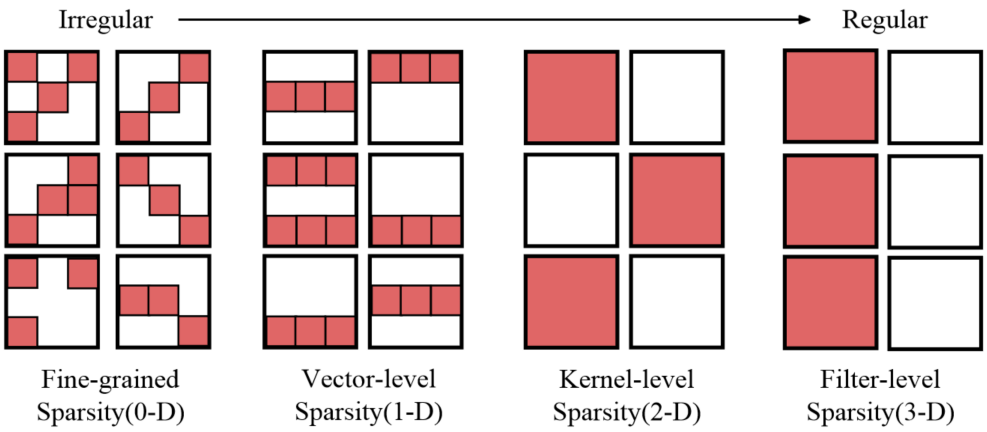
Τα δύο προηγούμενα κριτήρια κλαδέματος στόχευαν στην άμεση αξιολόγηση της σημαντικότητας κάθε υποψήφιου στοιχείου προς αφαίρεση. Αντίθετα, ορισμένες εργασίες εισάγουν την αραιότητα με πιο έμμεσο τρόπο χρησιμοποιώντας τεχνικές κανονικοποίησης. Ιδανικά, η (μη-δομημένη) διαδικασία κλαδέματος μπορεί να μοντελοποιηθεί με τη χρήση L_0 κανονικοποίησης, δηλαδή εισάγοντας έναν όρο $R_{L_0}(\boldsymbol{w}) = \sum_i \mathbb{I}[w_i \neq 0]$ στη συνάρτηση σφάλματος, που προωθεί τη μείωση των μη-μηδενικών παραμέτρων. Δυστυχώς, η άμεση ελαχιστοποίηση της συνάρτησης σφάλματος με τον όρο L_0 είναι δυσεπίλυτη, καθώς αυτός ο όρος είναι μη διαφορίσιμος και επιτρέπει $2^{|w|}$ διακριτές καταστάσεις του διανύσματος $\boldsymbol{w}$. Λόγω των προαναφερθέντων προβλημάτων της νόρμας L_0, οι περισσότερες εργασίες είτε προσπαθούν να επαναπαραμετροποιήσουν τις παραμέτρους του μοντέλου και να επιτρέψουν την αποτελεσματική βελτιστοποίηση, είτε χρησιμοποιούν τη νόρμα L_1 η οποία θεωρείται μια καλή κυρτή προσέγγιση της νόρμας L_0.

Δομή των Στοιχείων προς Αραίωση

Με βάση το ποια στοιχεία αραιώνονται, οι διάφορες προσεγγίσεις κλαδέματος μπορούν να χωριστούν σε γενικές γραμμές σε δύο κατηγορίες, μη-δομημένες ή δομημένες. Οι μη-δομημένες μέθοδοι αφήνουν τον αλγόριθμο κλαδέματος ελεύθερο να κλαδέψει τις παραμέτρους ανεξάρτητα από το πού ανήκουν μέσα στο μοντέλο. Από την άλλη πλευρά, οι δομημένες μέθοδοι κλαδέματος αφαιρούν ομάδες παραμέτρων και, συνεπώς, μπορούν να χαρακτηριστούν περαιτέρω από τον τύπο των ομάδων που κλαδεύονται (π.χ. διανύσματα, kernels ή ολόκληρα φίλτρα). Παραδείγματα των διαφορετικών τύπων μοτίβων αραιότητας που λαμβάνονται με μη-δομημένο ή δομημένο κλάδεμα απεικονίζονται στο Σχήμα 1.4.2.

Χρονικό Πλαίσιο της Αραίωσης

Η μεθοδολογία κλαδέματος μπορεί να κατηγοριοποιηθεί περαιτέρω με βάση το πότε γίνεται η αφαίρεση των βαρών. Συγκεκριμένα, η αραίωση μπορεί να πραγματοποιηθεί μετά το τέλος της τυπικής διαδικασίας εκπαίδευσης του πυκνού δικτύου, κατά τη διάρκεια της τυπικής εκπαίδευσης ή πριν από την έναρξη της εκπαίδευσης. Οι δύο πρώτες περιπτώσεις αναφέρονται συχνά ως πυκνή προς αραιή εκπαίδευση, καθώς ένα πλήρως πυκνό DNN φτάνει στο επιθυμητό επίπεδο αραιότητας στο τέλος της εκπαίδευσης, ενώ η τελευταία προσέγγιση, η οποία ξεκινά την εκπαίδευση με ένα ήδη αραιό μοντέλο, αναφέρεται ως αραιή προς αραιή εκπαίδευση [50].



Σχήμα 1.4.2: Σύγκριση των μοτίβων αραιότητας που προκαλούνται από μη-δομημένη αραιότητα (αριστερή πρώτη εικόνα) και διάφορους τύπους δομημένης αραιότητας, σε ένα σύνολο δύο φίλτρων CNN που το καθένα έχει τρία kernels 3×3. Από [63].

1.5 Προσαρμοστικό Κλάδεμα Μέτρου μέσω Μοντελοποίησης των ανά-επίπεδο Κατανομών

Αναγνωρίζοντας ότι τα DNNs είναι σε μεγάλο βαθμό υπερ-παραμετροποιημένα [82], το κλάδεμα των παραμέτρων τους (που εξετάζεται λεπτομερώς στην Ενότητα 1.4.5) έχει μελετηθεί εκτενώς τα τελευταία χρόνια ως ένας τρόπος δραστικής μείωσης του μεγέθους των μοντέλων και των υπολογιστικών τους απαιτήσεων [11, 14, 57, 34]. Η υπάρχουσα μεθοδολογία αν και αναφέρεται ότι επιτυγχάνει πολλά υποσχόμενα αποτελέσματα, (α) συνήθως απαιτεί εκτεταμένους χρόνους εκπαίδευσης, είτε με τη μορφή πολλαπλών γύρων επανεκπαίδευσης και fine-tunning [72] είτε προκαλώντας μη αμελητέα υπολογιστική επιβάρυνση [84, 97], (β) εξαρτάται από την εύρεση περίπλοκων ρυθμίσεων υπερπαραμέτρων προκειμένου να καταλήξει στο επιθυμητού μεγέθους μοντέλο [50] ή (γ) δεν έχει προσαρμοστικότητα στην εφαρμογή με πιο περίπλοκες αρχιτεκτονικές μοντέλων, ιδίως σε υψηλούς λόγους αραιότητας, με αποτέλεσμα ασυνέπειες στην απόδοση [98].

Σε αυτό το κεφάλαιο, προτείνουμε έναν πολύ αποδοτικό αλγόριθμο μη-δομημένου κλαδέματος, που επιτυγχάνει αποτελέσματα SoA, ενώ δεν εμφανίζει τα προαναφερθέντα μειονεκτήματα.

Ο προτεινόμενος αλγόριθμος αραιώνει τα βάρη των δικτύων μέσω κατωφλιοποίησης των βαρών κατά τη διάρκεια της εκπαίδευσης (χωρίς την ανάγκη επιπλέον εποχών επανεκπαίδευσης). Ο κύριος στόχος αυτής της προσέγγισης είναι η αποτελεσματική "εκμάθηση" πολλαπλών κατάλληλων προσαρμοστικών κατωφλίων για την εφαρμογή τους ξεχωριστά ανά επίπεδο του δικτύου.

Συγκεκριμένα, ακολουθώντας την προσέγγιση του [73], για την επίτευξη ενός συγκεκριμένου συνολικού λόγου αραιότητας S (τον οποίο αυξάνουμε προοδευτικά κατά τη διάρκεια της διαδικασίας εκπαίδευσης) θεωρούμε τα κατώφλια ανά επίπεδο $\{r_l\}$ (βάσει των οποίων πραγματοποιείται το κλάδεμα) εκπαιδεύσιμες παραμέτρους οι οποίες περιλαμβάνονται σε έναν πρόσθετο όρο σφάλματος, $L_s(\{r_l\}) = \left[S - \hat{S}(\{r_l\})\right]^2$, όπου $\hat{S}(\{r_l\})$ είναι μια εκτίμηση της αραιότητας του μοντέλου που υπολογίζεται με βάση τις υποθέσεις για τις κατανομές των βαρών ανά επίπεδο.

Για να διατηρήσουμε το $\hat{S}(\{r_l\})$ διαφορίσιμο, και με βάση προηγούμενες παρατηρήσεις, υποθέτουμε ότι τα βάρη κάθε επιπέδου αποκτούν κατανομές κοντά στο να είναι Gaussian ή Laplace, με τον συγκεκριμένο τύπο να αποφασίζεται αυτόματα ανά επίπεδο κατά τη διάρκεια της εκπαίδευσης, ώστε να ελαχιστοποιηθεί το σφάλμα εκτίμησης της αραιότητας. Το $L_s(\{r_l\})$ προστίθεται στη συνέχεια κατάλληλα στο τυπικό σφάλμα εκπαίδευσης, έτσι ώστε τα κατώφλια να βελτιστοποιούνται κατά τη διάρκεια της εκπαίδευσης, με αποτέλεσμα να μαθαίνεται μια μη ομοιόμορφη αραιότητα. Όλη η προηγούμενη ανάλυση γίνεται για να ισχύει κατά το κλάδεμα του δικτύου με τη χρήση του Straight-Through Estimator [3] για την ενημέρωση του πυκνού συνόλου των βαρών. Ως τελικό βήμα, για να επιτύχουμε το ακριβές καθορισμένο επίπεδο αραιότητας προσαρμόζουμε ελαφρώς τα κατώφλια που βρέθηκαν μερικές εποχές πριν από το τέλος της εκπαίδευσης, υπολογίζοντας τώρα τα ακριβή κατώφλια με

ταξινόμηση των βαρών ανά επίπεδο.

Οι συνεισφορές αυτής της εργασίας είναι οι εξής:

- Η μέθοδος που προτείνουμε, χωρίς σχεδόν καμία επιπλέον επιβάρυνση της εκπαίδευσης και χωρίς την ανάγκη για τον καθορισμό πρόσθετων υπερπαραμέτρων, μπορεί να αραιώσει ένα μοντέλο μέχρι ένα ακριβές επίπεδο που ορίζει ο χρήστης, διατηρώντας ένα αποτελεσματικό, μη ομοιόμορφο αριθμό παραμέτρων ανά επίπεδο.
- Δείχνουμε ότι η μοντελοποίηση των κατανομών βαρών ανά επίπεδο ως Gaussian και Laplace είναι επαρκής για την εκμάθηση των κατάλληλων κατωφλίων ανά επίπεδο για το κλάδεμα για διαφορετικές αρχιτεκτονικές μοντέλων, ακόμη και για υψηλές αραιότητες.
- Εκτεταμένα πειράματα τόσο στο σύνολο δεδομένων CIFAR [47] όσο και στο ImageNet [13] καταδεικνύουν ότι η μέθοδός μας επιτυγχάνει SoA ακρίβεια ξεπερνώντας πιο σύνθετες και μη αποδοτικές μεθόδους.

1.5.1 Προτεινόμενη Μέθοδος

Κριτήριο Κλαδέματος

Ακολουθούμε την προσέγγιση κλαδέματος με βάση το μέτρο όπου ένα βάρος διατηρείται μόνο αν το μέτρο του ξεπερνά μια τιμή κατωφλίου r. Αυτό το απλό κριτήριο, που κρίνει τη σημασία ενός βάρους με βάση το μέτρο του, είναι αποδοτικό στον υπολογισμό και έχει βρεθεί ότι είναι αποτελεσματικό στη βιβλιογραφία για το κλάδεμα δικτύων, επιτυγχάνοντας υψηλούς λόγους αραιότητας με ελάχιστη απώλεια απόδοσης [29, 21]. Το κύριο ζήτημα είναι ότι αυτό το κατώφλι πρέπει να επιλεχθεί κατάλληλα για κάθε επίπεδο, προκειμένου να επιτευχθεί ο ζητούμενος λόγος αραιότητας. Σημειώνουμε ότι ακόμη και αν σκοπεύουμε να διατηρήσουμε τον ίδιο λόγο αραιότητας για όλα τα επίπεδα, το r πρέπει να υπολογιστεί ξεχωριστά για κάθε επίπεδο, καθώς τα μεγέθη των βαρών διαφέρουν μεταξύ των διαφορετικών στρωμάτων. Πολλές προσεγγίσεις ταξινομούν τα βάρη για να βρουν τα κατώτατα όρια-στόχους για κάθε βήμα εκπαίδευσης [86, 98], γεγονός που μπορεί να οδηγήσει σε επιβάρυνση της εκπαίδευσης, ειδικά για μεγαλύτερα δίκτυα. Η μέθοδός μας, δεδομένου ενός ζητούμενου λόγου αραιότητας του δικτύου, βρίσκει τα κατώφλια αποτελεσματικά και οδηγεί σε μη ομοιόμορφη αραιότητα για βελτιστοποιημένη απόδοση (βλ. Ενότητα 1.5.1).

Μάθηση των Κατωφλίων

Μια προσέγγιση για την επίτευξη προσαρμοστικής αραιότητας ανά επίπεδο είναι να θεωρήσουμε τα κατώφλια r_l, για κάθε στρώμα l, εκπαιδεύσιμες παραμέτρους. Στη συνέχεια, χρησιμοποιώντας μια κατάλληλη *διαφορίσιμη* συνάρτηση, $\hat{s}_l(r_l) \in [0,1]$, που εκτιμά τον λόγο αραιότητας κάθε επιπέδου l, δεδομένου του κατωφλίου r_l, η συνολική εκτιμώμενη αραιότητα του μοντέλου μπορεί να οριστεί ως εξής:

$$\hat{S}(\{r_l\}) = \sum_{l}^{N} c_l \hat{s}_l(r_l), \tag{1.5.1}$$

όπου $c_l = \frac{\#\boldsymbol{W}_l}{\sum_i^N \#\boldsymbol{W}_i}$ είναι η συνεισφορά του επιπέδου l στις συνολικές παραμέτρους του δικτύου.

Εάν μπορέσουμε να εκφράσουμε την αραιότητα ανά επίπεδο $\hat{s}_l(r_l)$ ως αναλυτική συνάρτηση σε σχέση με το κατώφλι r_l θα μπορούμε να έχουμε ένα πλήρως εκπαιδεύσιμο σύστημα που μπορεί να προσαρμόζει αυτόματα την αραιότητα ανά επίπεδο σύμφωνα με ένα έξτρα όρο σφάλματος μέσω της Εξ. 1.5.1. Για το σκοπό αυτό, [73] θεωρήθηκε ότι, για κάθε επίπεδο, τα βάρη μπορούν να προσεγγιστούν από μια κατανομή Gauss. Σύμφωνα με αυτή την εμπειρική παραδοχή, μπορεί κανείς πράγματι να γράψει την αραιότητα του επιπέδου ως μια αναλυτική συνάρτηση ως προς ένα κατώφλι, ακολουθώντας μια λογική διαστημάτων εμπιστοσύνης. Επεκτείνουμε αυτή τη διατύπωση για να συμπεριλάβουμε επίσης κατανομές Laplace για βελτιωμένη εκτίμηση της αραιότητας (βλ. Ενότητα 1.5.1 για περισσότερες λεπτομέρειες).

Για την προώθηση ενός συγκεκριμένου λόγου αραιότητας του μοντέλου, $S \in (0,1)$, μπορεί κανείς να ορίσει ένα σφάλμα αραιότητας $L_s(\{r_l\})$ ως εξής:

$$L_s(\{r_l\}) = \left[S - \hat{S}(\{r_l\})\right]^2. \tag{1.5.2}$$

Αυτή η διατύπωση επιβάλλει ένα στόχο αραιότητας που καθορίζεται από τον χρήστη για πρακτικές εφαρμογές. Το $L_s(\{r_l\})$ ελαχιστοποιείται όταν τα κατώφλια r_l έχουν λάβει τιμές που οδηγούν σε $S = \hat{S}(\{r_l\})$. Αυτό το σφάλμα μπορεί στη συνέχεια να προστεθεί στο τυπικό σφάλμα εκπαίδευσης $L(\{\boldsymbol{W}_l\}, \{r_l\})$ για να προκύψει μια συνάρτηση σφάλματος πολλαπλών εργασιών

$$L(\{\boldsymbol{W}_l\}, \{r_l\}) + \lambda_s L_s(\{r_l\}). \tag{1.5.3}$$

Ορίζοντας το συνολικό σφάλμα όπως παραπάνω και ελαχιστοποιώντας σε σχέση με τα δύο σύνολα παραμέτρων, $\{\boldsymbol{W}_l\}$ και $\{r_l\}$, οι όροι της συμπεριφέρονται με ανταγωνιστικό τρόπο, δηλαδή η L ελαχιστοποιείται ευκολότερα αν το μοντέλο παραμείνει πυκνό, συνεπώς η L_s παραμένει κοντά στη μέγιστη τιμή της. Επομένως, απαιτείται κάποιος συντελεστής κλιμάκωσης λ_s για να σταθμιστεί η συνεισφορά του L_s.

Το γεγονός ότι οι τιμές των κατωφλίων θα μαθαίνονται μέσω της διαδικασίας βελτιστοποίησης (και συνεπώς δεν θα είναι απλώς τυχαία ή ευρετικά βασισμένα κατώφλια που δίνουν τη ζητούμενη συνολική αραιότητα) υποδεικνύει ότι βρίσκεται μια ιδιαίτερα αποτελεσματική ανά επίπεδο κατανομή αραιότητας (βλ. Ενότητα 1.5.2).

Τέλος, για να λειτουργήσει αυτή η διαδικασία, η χρήση του Straight-Through Estimator για back-propagation είναι επιβεβλημένη για τη διατήρηση μονοτροπικών κατανομών και διευκολύνοντας έτσι τη χρήση ενός αναλυτικού τύπου για την προσέγγιση του σφάλματος αραιότητας.

Στην συγκεκριμένη εργασία, διαπιστώσαμε ότι το σφάλμα αραιότητας, ειδικά για υψηλές τιμές της αραιότητας στόχου S, θα πρέπει να είναι υψηλά σταθμισμένο με βάρος προκειμένου να είναι σημαντικό σε μέγεθος και, συνεπώς, να ελαχιστοποιείται από τη διαδικασία βελτιστοποίησης. Για να το επιτύχουμε αυτό, το κλιμακώνουμε περαιτέρω προσαρμοστικά με το αντίστροφο του τετραγωνικού budget $B^2 = (1-S)^2$, με αποτέλεσμα την τελική συνολική εξίσωση σφάλματος που χρησιμοποιείται στη μέθοδό μας

$$L(\{\boldsymbol{W}_l\}, \{r_l\}) + \frac{\lambda_s}{B^2} L_s(\{r_l\}). \tag{1.5.4}$$

Η προαναφερθείσα κλιμάκωση είναι πιο κρίσιμη για τη μέθοδό μας, καθώς, σε αντίθεση με την [73], αυξάνουμε τον λόγο αραιότητας του στόχου σταδιακά κατά την εκπαίδευση (όπως περιγράφεται στην Ενότητα 5.3.6) για μια πιο ομαλή μετάβαση από ένα πυκνό σε ένα αραιό μοντέλο και επομένως ένα σταθερό βάρος λ_s που καθορίζεται από τον χρήστη δεν θα ήταν κατάλληλο. Με αυτό το τέχνασμα κλιμάκωσης, το λ_s είναι σχεδόν ασήμαντο και μπορεί να έχει κάποια σταθερή τυπική τιμή (την ορίσαμε σε $\lambda_s = 10$) ανεξάρτητα από τον ζητούμενο λόγο αραιότητας και το πείραμα.

Μοντελοποίηση Κατανομών Βαρών

Θεωρώντας τα βάρη ανά επίπεδο $\{W_l \in \boldsymbol{W}_l\}$ ως τυχαίες μεταβλητές w_l που ακολουθούν μια συμμετρική κατανομή, ο λόγος αραιότητας $s_l(r_l)$ για μια δεδομένη τιμή κατωφλίου r_l εκτιμάται ως η πιθανότητα το w_l να εμπίπτει στο εύρος $[-r_l,\ r_l]$ ως εξής

$$\hat{s}_l(r_l) = \int_{-r_l}^{r_l} f_l(w_l) dw_l \tag{1.5.5}$$

με το $f_l(w_l)$ να αντιπροσωπεύει τη συνάρτηση πυκνότητας πιθανότητας του στοιχείου w_l στο l-οστό επίπεδο.

Τα βάρη τείνουν να αποκτήσουν κατανομές ανά επίπεδο που είναι παρόμοιες με τις κατανομές Gaussian ή Laplace. Με βάση αυτή την υπόθεση μπορούμε να υπολογίσουμε την $\tilde{s}_l(r_l)$ μέσω της εξίσωσης (1.5.5) που οδηγεί σε εκφράσεις κλειστής μορφής ως εξής

$$\hat{s}_l(r_l) = \begin{cases} \operatorname{erf}\left(r_l/\sigma_l\sqrt{2}\right), & \text{αν } w_l \sim N(0, \sigma_l^2) \\ 1 - \exp\left(-r_l/\beta_l\right), & \text{αν } w_l \sim L(0, \beta_l) \end{cases} \tag{1.5.6}$$

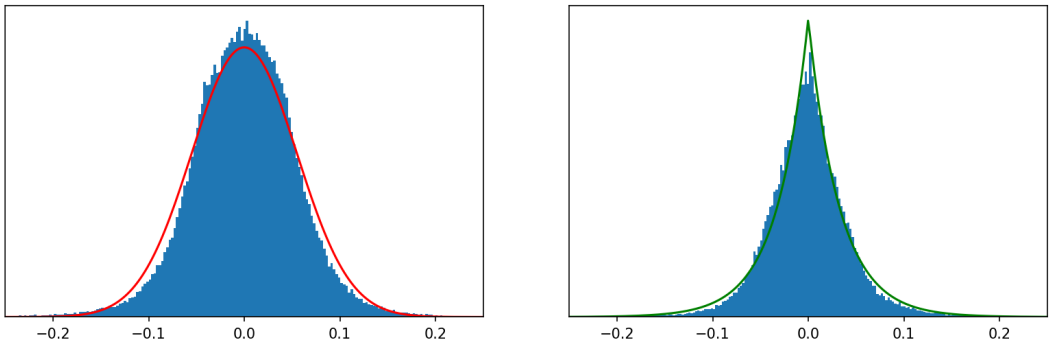
$$\text{όπου} \quad \operatorname{erf}(x) = \frac{1}{\sqrt{\pi}} \int_{-x}^{x} e^{-t^2} dt \tag{1.5.7}$$

Το πλεονέκτημα της υιοθέτησης αυτών των υποψήφιων μοντέλων κατανομής βαρών είναι ότι το $\hat{s}_l(r_l)$ μπορεί να υπολογιστεί αποτελεσματικά και παραμένει διαφορίσιμο.

Σημειώνουμε ότι έχουμε υποθέσει κατανομές βαρών με μηδενική μέση τιμή ανά επίπεδο, αλλά η προσέγγισή μας μπορεί εύκολα να προσαρμοστεί ώστε να ισχύει για μη μηδενική μέση τιμή. Επίσης, στην πράξη εκτιμούμε τις παραμέτρους κατανομής σ_l^2 και β_l των βαρών χρησιμοποιώντας τις εκτιμήσεις μέγιστης πιθανοφάνειας

$$\hat{\sigma_l}^2 = \frac{1}{N}\sum_{i=1}^{N} W_{li}^2 \text{ και } \hat{\beta}_l = \frac{1}{N}\sum_{i=1}^{N} |W_{li}|.$$

Στη μέθοδό μας δεν περιορίζουμε τις κατανομές μοντελοποίησης σε έναν συγκεκριμένο τύπο (Gaussian ή Laplace) όπως γίνεται σε προηγούμενες εργασίες [73, 41]. Ο τύπος της κατανομής ανά επίπεδο επιλέγεται δυναμικά καθ' όλη τη διάρκεια της εκπαίδευσης με σκοπό την ελαχιστοποίηση του σφάλματος μοντελοποίησης. Αυτό είναι σημαντικό, καθώς διαφορετικά επίπεδα (ή ακόμη και το ίδιο επίπεδο σε διαφορετικές εποχές) λαμβάνουν κατανομές βαρών που είναι πολύ πιο κοντά σε έναν από τους δύο τύπους μοντελοποίησης από τον άλλο (π.χ. όπως φαίνεται στο Σχήμα 1.5.1).



(a) (b)

Σχήμα 1.5.1: Παράδειγμα επιπέδου με βάρη κατανεμημένα κατά Gauss (α) και με βάρη κατανεμημένα κατά Laplacian (β). Οι κατανομές προέρχονται από τα επίπεδα layer2.0.conv2 και layer2.1.conv1 του ResNet50.

Αλλάζοντας Κατανομές

Η κατανομή μοντελοποίησης που αποδίδεται σε κάποιο επίπεδο l επανεξετάζεται στο τέλος κάθε εποχής εκπαίδευσης k με βάση απλά την ελαχιστοποίηση του σφάλματος εκτίμησης της αραιότητας, το οποίο ορίζεται ως εξής

$$es_l^G(r_l) = |\hat{s}_l^G(r_l) - s_l(r_l)| \tag{1.5.8}$$

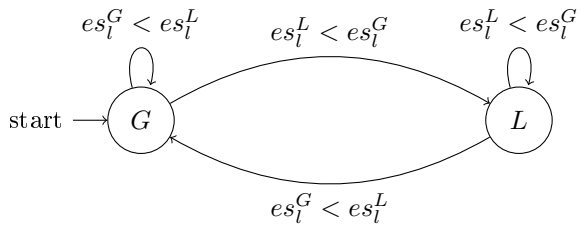
$$es_l^L(r_l) = |\hat{s}_l^L(r_l) - s_l(r_l)| \tag{1.5.9}$$

όπου r_l είναι το κατώφλι που λαμβάνεται στο τέλος της εποχής k, $\hat{s}_l^G(r_l)$ και $\hat{s}_l^L(r_l)$ είναι οι τιμές αραιότητας που προσεγγίζονται με τη χρήση μιας κατανομής Gaussian και μιας κατανομής Laplace αντίστοιχα, με αποτέλεσμα τα σφάλματα εκτίμησης $es_l^G(r_l)$ και $es_l^L(r_l)$. Επομένως, εάν η κατανομή του στρώματος στην εποχή $= k$ είναι Gaussian και $es_l^L(r_l) < es_l^G(r_l)$ μεταβαίνουμε στη χρήση της κατανομής Laplace για να μοντελοποιήσουμε τα βάρη του επιπέδου για την επόμενη εποχή $k{+}1$. Αντίστοιχα αν $es_l^G(r_l) < es_l^L(r_l)$ και η κατανομή μοντελοποίησης είναι η Laplace μεταβαίνουμε στην Gaussian. Αυτή η στρατηγική απεικονίζεται σε ένα διάγραμμα καταστάσεων στο Σχήμα 1.5.2.

Χρονοπρογραμματισμός και Τελική Διόρθωση Αραιότητας

Ο ακόλουθος χρονοπρογραμματιστής αραιότητας, βασισμένος στο [98] χρησιμοποιείται για τη σταδιακή αύξηση του λόγου αραιότητας στόχου

$$S_i = S_f \left[1 - \left(1 - \frac{i}{n}\right)^3\right], \text{ for } i = 1, 2, ..., n \tag{1.5.10}$$



Σχήμα 1.5.2: Διάγραμμα μετάβασης καταστάσεων όπου οι καταστάσεις είναι οι δύο υποψήφιες κατανομές μοντελοποίησης (G: Gaussian, L: Laplace) για το επίπεδο l. Οι μεταβάσεις γίνονται σύμφωνα με τους περιγραφόμενους κανόνες στο τέλος κάθε εποχής. Υποθέτουμε ότι όλα τα επίπεδα είναι αρχικά (στην εποχή 1) μοντελοποιημένα με μια κατανομή Gauss.

όπου i είναι ένα βήμα εκπαίδευσης, n ο συνολικός αριθμός των βημάτων κλαδέματος, S_i ο λόγος αραιότητας στόχου στο βήμα i και S_f ο τελικός λόγος αραιότητας στόχου. Στα πειράματά μας ορίσαμε το n ίσο με τον αριθμό των βημάτων εκπαίδευσης μέχρι το 80% των εποχών εκπαίδευσης. Κατά τη διάρκεια αυτής της περιόδου τα κατώφλια κλαδέματος μαθαίνονται όπως περιγράφεται στην Ενότητα 1.5.1.

Παρόλο που η εκτίμηση της αραιότητας με τη χρήση των κατανομών βαρών είναι αρκετά ακριβής, ο τελικός λόγος αραιότητας μπορεί να αποκλίνει ελαφρώς από το ζητούμενο. Εάν το S_f επιτυγχάνεται στο βήμα $m < n$, ο λόγος αραιότητας κάθε επιπέδου, s_l, μειώνεται γραμμικά για μερικές επαναλήψεις ώστε να φτάσει στην τελική του τιμή. Εάν στο βήμα n η μετρούμενη αραιότητα $\bar{S}_n$ είναι μικρότερη από την S_f, για το επόμενο 10% των εποχών οι λόγοι αραιότητας αυξάνονται γραμμικά μέχρι την τελική τους τιμή. Μόνο κατά τη διάρκεια αυτής της φάσης της μεθόδου οι ακριβείς τιμές κατωφλίου υπολογίζονται μέσω της ταξινόμησης των βαρών κάθε στρώματος. Αφού η αραιότητα φτάσει στο S_f διατηρείται σταθερή για τα υπόλοιπα βήματα εκπαίδευσης. Σημειώνουμε ότι συνήθως $s_l \approx s'_l$ αφού το σφάλμα εκτίμησης της αραιότητας αναμένεται να είναι αρκετά χαμηλό.

Πειραματική Αξιολόγηση

CIFAR-100

Ratio	90%	95%	98%
ResNet-20 (1.096M Params): 73.59 ±0.44			
GMP [98]	70.34 ±0.33	69.38 ±0.29	64.46 ±0.36
ST-3 [86]	72.86 ±0.20	71.95 ±0.12	**67.73** ±0.10
Ours	**73.18** ±0.11	**72.30** ±0.22	66.69 ±0.24
MobileNetV1 (3.315M Params): 71.15 ±0.17			
GMP [98]	61.82 ±0.18	52.87 ±0.33	32.72 ±1.09
ST-3 [86]	71.02 ±0.09	**70.50** ±0.08	**69.18** ±0.34
Ours	**71.11** ±0.12	70.37 ±0.14	67.53 ±0.22
DenseNet40-24 (0.714M Params): 74.70 ±0.51			
GMP [98]	70.72 ±0.29	68.29 ±0.37	61.56 ±0.19
ST-3 [86]	72.82 ±0.43	**71.66** ±0.38	**65.99** ±0.38
Ours	**74.00** ±0.30	71.63 ±0.32	63.20 ±0.31

Πίνακας 1.1: Ακρίβεια των ResNet20, MobileNetV1 και DenseNet40-24 στο CIFAR-100 σε διαφορετικούς λόγους αραιότητας. Αναφέρεται επίσης η αρχική ακρίβεια του πυκνού μοντέλου για λόγους σύγκρισης.

Ο Πίνακας 1.1 παρέχει τα αποτελέσματα που προέκυψαν στο CIFAR-100 με τη χρήση των GMP [98], ST-3 [86] και της μεθόδου μας σε τρεις διαφορετικές αρχιτεκτονικές και τρία διαφορετικά επίπεδα αραιότητας. Τα αποτελέσματα καταρχάς δείχνουν την ευρωστία και την προσαρμοστικότητα της προσέγγισής μας χρησιμοποιώντας τις κατανομές Gaussian και Laplace για τη μοντελοποίηση των βαρών ανά στρώμα τριών διαφορετικών μοντέλων σε διαφορετικούς λόγους κλαδέματος, ξεπερνώντας εύκολα τα αποτελέσματα που λαμβάνονται από το GMP (το οποίο αγνοεί τη σημασία της εύρεσης ενός μη ομοιόμορφου budget αραιότητας ανά στρώμα). Σε σύγκριση με το ST-3, παρατηρούμε βελτιωμένη απόδοση για τον λόγο αραιότητας 90%, παρόμοια απόδοση για τον λόγο 95%, ενώ η προσέγγισή μας υπολείπεται ελαφρώς στον λόγο 98% (βλ. Ενότητα 1.5.3). Θέλουμε να τονίσουμε ότι η

ST-3, αν και μπορεί να παρέχει αποτελέσματα SoA, είναι πιο δαπανηρή υπολογιστικά από τη μέθοδό μας λόγω του γεγονότος ότι πρέπει να ταξινομήσει όλα τα βάρη προκειμένου να βρει το κατώφλι σε κάθε βήμα της εκπαίδευσης. Η μέθοδός μας, που είναι η πιο φιλική προς τον υπολογισμό μεταξύ όλων των εξεταζόμενων μεθόδων, βρίσκει τα κατώφλια χωρίς τη χρήση ταξινόμησης για το μεγαλύτερο μέρος της εκπαίδευσης (απαιτούνται μόνο στατιστικά στοιχεία πρώτης και δεύτερης τάξης, τα οποία μπορούν να υπολογιστούν σε γραμμικό χρόνο σε σχέση με τον αριθμό των βαρών ανά επίπεδο) και ταξινομεί τα βάρη ανά επίπεδο μόνο κατά τις τελευταίες εποχές. Παρόλα αυτά, διαπιστώνεται ότι έχει την καλύτερη απόδοση για τον λόγο αραιότητας 90% με αποτέλεσμα να επιτυγχάνονται ακρίβειες χωρίς σχεδόν καμία πτώση σε σχέση με αυτές που προκύπτουν από τα πυκνά δίκτυα. Αυτό είναι δυνατό λόγω των αποτελεσματικών λόγων αραιότητας ανά επίπεδο που βρίσκονται αυτόματα κατά τη διάρκεια της εκπαίδευσης, όπως περιγράφεται στην Ενότητα 5.3.2. Αυτό αναλύεται περαιτέρω στην Ενότητα 1.5.2.

ImageNet

Ratio	90%	95%	98%
ResNet-50 (25.6M Params): 77.10			
GMP [98]	73.91	70.59	57.90
STR [50]	74.31	70.40	61.46
ProbMask [97]	74.68	71.50	66.83
ST-3 [86]	76.03	74.46	**70.46**
Spartan [84]	76.17	**74.68**	-
Ours	**76.27**	74.27	67.83

Πίνακας 1.2: Ακρίβεια του ResNet50 στο ImageNet.

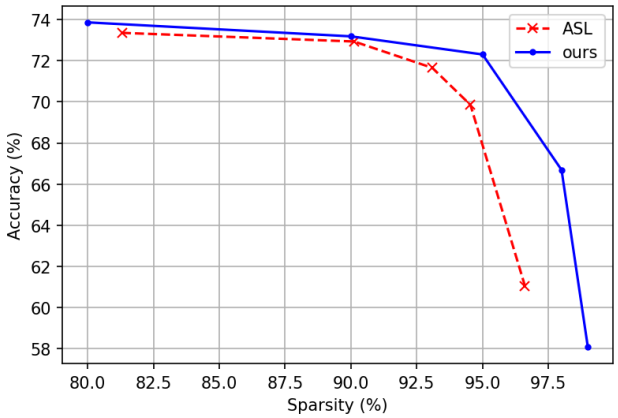
Για να αναδείξουμε τις ικανότητες γενίκευσης της μεθόδου μας και την ικανότητά της να σταθεί απέναντι σε άλλες, πιο ακριβές υπολογιστικά και εξαρτώμενες από τις υπερπαραμέτρους SoA μεθόδους, διεξάγουμε πειράματα στο ImageNet με το ResNet50. Εκτός από τα αποτελέσματα από τις GMP και ST-3, σε αυτή την ενότητα περιλαμβάνουμε αποτελέσματα από τις μεθόδους STR [50], ProbMask [97] και Spartan [84], οι οποίες είναι πρόσφατες μέθοδοι μη-δομημένου κλαδέματος. Η STR χρησιμοποιεί έναν τελεστή soft κατωφλίου για να καταλήξει σε μη ομοιόμορφη αραιότητα, η οποία ελέγχεται έμμεσα από την weight decay παράμετρο. Αυτό καθιστά τη μέθοδό αυτή πολύ δύσκολο να πετύχει έναν καθορισμένο λόγο αραιότητας. Η Probmask προσεγγίζει το πρόβλημα κλαδέματος χρησιμοποιώντας μια πιθανοτική μάσκα η οποία μετατρέπεται σε μια ντετερμινιστική δυαδική μάσκα χρησιμοποιώντας δείγματα από μια κατανομή Gumbel. Η συγκεκριμένη προσέγγιση απαιτεί πολλαπλούς γύρους δειγματοληψίας με υπολογισμό gradients που αυξάνουν σημαντικά το χρόνο εκπαίδευσης. Τέλος, η Spartan μαθαίνει μια soft top-k μάσκα χρησιμοποιώντας ένα regularized optimal transportation πρόβλημα, αποτελώντας έτσι επίσης μια υπολογιστικά βαρύτερη μέθοδο από τη δική μας.

Με βάση τα αποτελέσματα που παρέχονται στον Πίνακα 1.2, η μέθοδός μας επιτυγχάνει κορυφαίες επιδόσεις σε ποσοστά 90% και 95%, ξεπερνώντας τις GMP, STR και ProbMask και φτάνοντας τα αποτελέσματα των ST-3 και Spartan. Στο λόγο 98% ξεπερνά επίσης τις τρεις πρώτες μεθόδους και έρχεται δεύτερη μετά την ST-3. Σε αυτό το σημείο πρέπει να σημειώσουμε ότι χρησιμοποιήσαμε τις ίδιες, προεπιλεγμένες υπερπαραμέτρους εκπαίδευσης και για τους τρεις λόγους αραιότητας, ενώ στην ST-3 το weight decay μειώνεται προοδευτικά καθώς αυξάνονται οι λόγοι αραιότητας - δεν έγινε καμία διερεύνηση σχετικά με την αναγκαιότητα μιας τέτοιας επιλογής λεπτομερούς ρύθμισης. Επιλέξαμε να δείξουμε ότι η μέθοδός μας είναι σε θέση να αποδώσει πολύ καλά ανεξάρτητα από τις χρησιμοποιούμενες υπερπαραμέτρους και με τη χαμηλότερη υπολογιστική επιβάρυνση μεταξύ των ανταγωνιστικών μεθόδων. Ο ισχυρισμός αυτός υποστηρίζεται από τα αποτελέσματα που προέκυψαν, τόσο στο CIFAR-100 όσο και στο ImageNet.

1.5.2 Ablation Μελέτες

Σύγκριση με την ASL

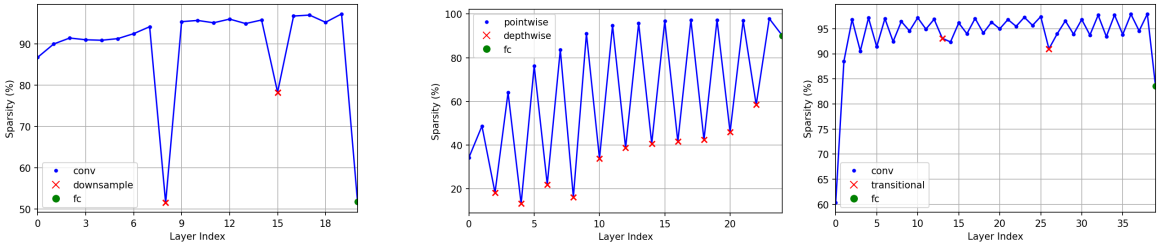
Σε αυτό το υποκεφάλαιο συγκρίνουμε άμεσα τη μέθοδό μας με τη στενά συνδεδεμένη προσέγγιση Adaptive Sparsity Loss (ASL) από το [73] για να αναδείξουμε περαιτέρω την αυξημένη αποτελεσματικότητα των προτεινόμενων τροποποιήσεων. Η ASL, δεδομένου ότι αποτελεί το αρχικό κίνητρο για την εργασία μας, βασίζεται επίσης στην ιδέα της εύρεσης των κατωφλίων μέσω της εκπαίδευσης, χρησιμοποιώντας το σφάλμα της Εξίσωσης 1.5.2, αλλά χρησιμοποιώντας μόνο Gaussian κατανομές για τη μοντελοποίηση των βαρών. Επιπλέον, στην ASL



Σχήμα 1.5.3: Ακρίβεια του ResNet20, που έχει κλαδευτεί με τη δική μας μέθοδο και την ASL στο CIFAR-100 σε διαφορετικούς λόγους αραιότητας.

η αραιότητα εισάγεται απότομα από την αρχή της εκπαίδευσης, καθώς δεν χρησιμοποιείται χρονοπρογραμματιστής αραιότητας. Επίσης, δεν υπάρχει φάση διόρθωσης της αραιότητας, επομένως η τελική αραιότητα μπορεί να διαφέρει από την ζητούμενη, ένα πολύ συνηθισμένο πρόβλημα όταν εξετάζονται υψηλά επίπεδα αραιότητας. Στο Σχήμα 1.5.3 παρουσιάζουμε αποτελέσματα από την εκπαίδευση του ResNet20 στο CIFAR-100 για πολλαπλούς λόγους αραιότητας χρησιμοποιώντας τις ίδιες ρυθμίσεις όπως στα πειράματα της Ενότητας 1.5.1. Λόγω του γεγονότος ότι το σφάλμα αραιότητας της ASL δεν κλιμακώνεται προσαρμοστικά, πρέπει να αυξήσουμε χειροκίνητα τον συντελεστή βαρύτητάς της λ_s (του multi-task σφάλματος της ASL) καθώς αυξάνουμε τη ζητούμενη τελική αραιότητα, προκειμένου να αποφύγουμε μεγάλες αποκλίσεις στην επιτευχθείσα αραιότητα και να παρέχουμε ουσιαστικές συγκρίσεις. Τα αποτελέσματα παρουσιάζονται σε γράφημα, δεδομένου ότι η ASL δεν μπορούσε να παράγει την ακριβή ζητούμενη αραιότητα. Από το γράφημα είναι σαφές ότι η μέθοδός μας, εκτός του ότι μπορεί να επιτύχει την ακριβή ζητούμενη αραιότητα, οδηγεί σε καλύτερες ακρίβειες, ιδίως για δύσκολες αναλογίες αραιότητας, όπου η βελτιωμένη εκτίμηση της αραιότητας και η σταδιακή εισαγωγή της αραιότητας αποκτούν ιδιαίτερη σημασία.

Κατανομή Αραιότητας Ανά-επίπεδο



(a) ResNet20, 95% αραιό. (b) MobileNetV1, 95% αραιό (c) DenseNet40-24, 95% αραιό

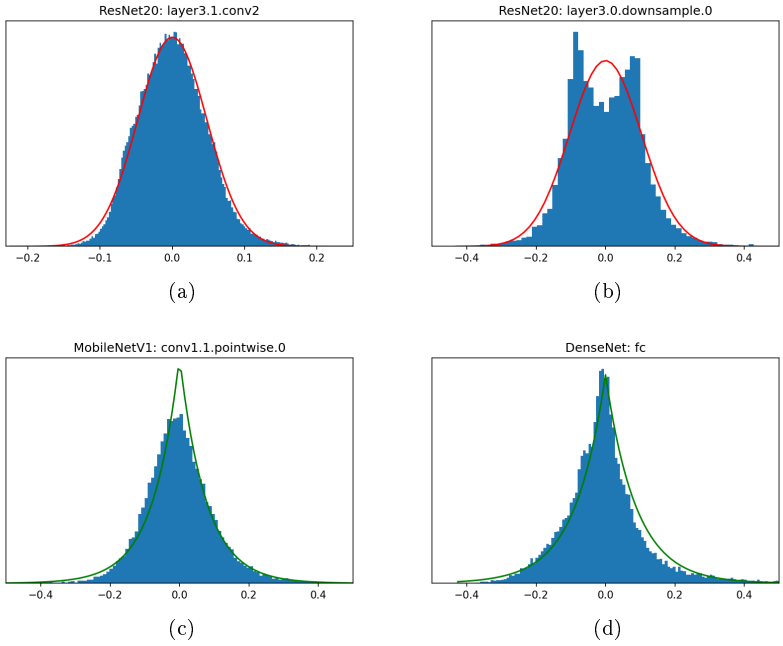
Σχήμα 1.5.4: Αναλογίες αραιότητας ανά επίπεδο (%) για μοντέλα που εκπαιδεύτηκαν στο σύνολο δεδομένων CIFAR-100. Διαφορετικά επίπεδα το δικτύου (π.χ. conv, fc) σημειώνονται με διαφορετικά χρώματα.

Οι κατανομές αραιότητας ανά επίπεδο που προέκυψαν από την εκπαίδευση των μοντέλων ResNet20, MobileNetV1 και DenseNet40-24 στο CIFAR-100 με τελική αραιότητα 95% παρουσιάζονται στο Σχήμα 1.5.4. Για το ResNet20 η μέθοδός μας αυξάνει τον λόγο αραιότητας σταδιακά από τα πρώτα προς τα τελευταία επίπεδα, με εξαίρεση τα δύο επίπεδα υποδειγματοληψίας και το πλήρως συνδεδεμένο επίπεδο στα οποία ανατίθενται σημαντικά χαμηλότεροι λόγοι αραιότητας. Για την περίπτωση του MobileNetV1, η κατανομή της αραιότητας αυξάνεται επίσης σταδιακά, αλλά σε διαφορετική κλίμακα για τους δύο τύπους επιπέδων, τα pointwise και τα depthwise, με τα τελευταία να ανατίθενται πολύ χαμηλότεροι λόγοι αραιότητας. Τέλος, η αραιότητα ανά επίπεδο του DenseNet40-24 δείχνει ότι στο πρώτο και το τελευταίο επίπεδο ανατίθεται πολύ μικρότερη αραιότητα από τα υπόλοιπα, τα οποία φαίνεται

να έχουν ένα κυμαινόμενο μοτίβο αραιότητας (εν μέρει με βάση το μέγεθός τους). Αξίζει να σημειωθεί ότι στα δύο μεταβατικά επίπεδα αποδίδεται μικρότερη αραιότητα από τα περισσότερα γειτονικά τους στρώματα, παρά το γεγονός ότι έχουν περισσότερες παραμέτρους. Αυτή η τελευταία παρατήρηση δείχνει ότι η μέθοδός μας κατανέμει την αραιότητα όχι μόνο με βάση τον αριθμό των παραμέτρων ανά επίπεδο (όπως κάνουν οι περισσότερες ευρετικές μέθοδοι), αλλά και με βάση τη λειτουργικότητα κάθε επίπεδου, με αποτέλεσμα να είναι σε θέση να εντοπίζει τα bottleneck επίπεδα και να τα διατηρεί επαρκώς πυκνά για να αποφεύγεται η υποβάθμιση της απόδοσης.

1.5.3 Περιορισμοί και Μελλοντικές Προεκτάσεις

Ο κύριος περιορισμός της προτεινόμενης μεθόδου αφορά την απαίτηση πολύ υψηλών αραιοτήτων (γενικά πάνω από 98%) από σχετικά μικρά μοντέλα. Για τέτοιες περιπτώσεις, ενώ εξακολουθεί να είναι σε θέση να παρέχει μοντέλα με ακρίβεια καλύτερη από τις περισσότερες μεθόδους, υπολείπεται των πολύ πρόσφατων SoA μεθόδων (βλ. Πίνακες 1.1, 1.2). Αυτό είναι αναμενόμενο, καθώς η μοντελοποίηση των κατανομών των βαρών, αν και πολύ ακριβής για τις περισσότερες περιπτώσεις, δεν είναι απόλυτα ακριβής (όπως φαίνεται στο Σχήμα 1.5.5 (b) - σημειώνουμε ωστόσο ότι πολλά επίπεδα εξακολουθούν να διατηρούν κατανομές βαρών κοντά στις εξεταζόμενες). Αποτελέσματα όπως αυτό του Σχήματος 1.5.5 (b) είναι πιο συχνά καθώς ωθούμε την απαίτηση αραιότητας σε εξαιρετικά υψηλά ποσοστά ($> 98\%$). Επομένως, ακόμη και μια μικρή απόκλιση από τον στόχο της αραιότητας μπορεί να οδηγήσει σε σημαντική απόκλιση από το αντίστοιχο budget και άρα πρέπει να διορθωθεί από την τελευταία φάση της μεθόδου (Ενότητα 1.5.1).



Σχήμα 1.5.5: Παραδείγματα των κατανομών βαρών ανά επίπεδο που προέκυψαν κατά την εκπαίδευση στο σύνολο δεδομένων CIFAR-100 με τη χρήση της μεθόδου μας. Ορισμένα επίπεδα μοντελοποιούνται καλύτερα από άλλα.

1.5.4 Συμπεράσματα

Στην παρούσα εργασία προτείνεται μια υπολογιστικά αποδοτική και αποτελεσματική μέθοδος κλαδέματος, όπως καταδεικνύεται από εκτεταμένα πειράματα στα σύνολα δεδομένων CIFAR και ImageNet, όπου ξεπερνά σε απόδοση πιο εξεζητημένες μεθόδους που επιφέρουν αυξημένο κόστος εκπαίδευσης. Επιπλέον, η επιτυχία της υποδεικνύει ότι η μη ομοιόμορφη αραιότητα των επιπέδων μπορεί να μαθευτεί αποτελεσματικά καθορίζοντας ένα επιπλέον όρο σφάλματος που βασίζεται στην μοντελοποίηση των κατανομών των βαρών, με τις κατανομές Gaussian και Laplace να κρίνονται επαρκείς, ακόμη και για σχετικά υψηλά ποσοστά αραιότητας.

1.6 Feather: Μια Κομψή Λύση για Αποτελεσματική Αραίωση Νευρωνικών Δικτύων

Μια πρόσφατη τάση στη βιβλιογραφία για το κλάδεμα νευρωνικών δικτύων είναι να πραγματοποιείται το κλάδεμα κατά τη διάρκεια της τυπικής εκπαίδευσης (αραιή εκπαίδευση) χωρίς την ανάγκη για πρόσθετους κύκλους επανεκπαίδευσης. Ορισμένες εργασίες [73, 44, 84, 86] οι οποίες βασίζονται στον Straight-Through Estimator (STE) [3] έχουν δείξει ότι μπορούν να επιτευχθούν αποτελέσματα SoA με τη χρήση αυτής της προσέγγισης. Η αραιή εκπαίδευση με τον STE εκτελείται με τον υπολογισμό του εμπρόσθιου περάσματος χρησιμοποιώντας την κατωφλιωμένη (κλαδεμένη) έκδοση των βαρών, ενώ ενημερώνει τα πυκνά βάρη κατά τη διάρκεια του οπίσθιου περάσματος, αντιμετωπίζοντας τη συνάρτηση κατωφλίωσης (που εκτελεί το κλάδεμα) ως την ταυτοτική.

Σε αυτό το κεφάλαιο, εστιάζουμε στη βελτίωση της αραιής εκπαίδευσης με τον STE, αντιμετωπίζοντας ορισμένες αδυναμίες της μεθόδου, δημιουργώντας τελικά μία νέα μονάδα κλαδέματος. Προτείνουμε (i) μια νέα συνάρτηση κατωφλίου που χρησιμοποιείται για το κλάδεμα με βάση το μέτρο και (ii) έναν απλό τρόπο ελέγχου της ροής των gradients των κλαδεμένων βαρών. Πιο συγκεκριμένα, αντί να χρησιμοποιούμε hard ή soft κατωφλίωση [18], τις οποίες χρησιμοποιούν κυρίως οι προγενέστερες μέθοδοι που βασίζονται στον STE, προτείνουμε μια οικογένεια συναρτήσεων κατωφλίωσης που βρίσκονται μεταξύ των δύο προαναφερθέντων και συνδυάζουν τα πλεονεκτήματά τους, δηλαδή μειωμένο bias μεταξύ των κατωφλιωμένων βαρών και των πυκνών ομολόγων τους και μια ομαλή περιοχή μετάβασης κοντά στο κατώφλι. Συμπληρωματικά με την προτεινόμενη προσέγγιση κατωφλίου, προτείνουμε την κλιμάκωση των gradients που αποδίδονται στις κλαδεμένες παραμέτρους με μια παράμετρο $\theta \in (0, 1)$, με στόχο τη βελτίωση της σταθερότητας της μάσκας κλαδέματος, ένας παράγοντας που θεωρούμε ότι είναι κρίσιμος όταν στοχεύουμε σε πολύ υψηλούς λόγους αραιότητας.

Επιδεικνύουμε την αποτελεσματικότητα της προσέγγισής μας για αραιή εκπαίδευση όταν εφαρμόζεται σε συστήματα μη-δομημένου κλαδέματος με βάση το μέτρο, τα οποία επιτυγχάνουν έναν προκαθορισμένο από τον χρήστη λόγο αραιότητας με σταδιακό κλάδεμα του δικτύου κατά τη διάρκεια της εκπαίδευσης. Αναλυτικότερα, εκτελούμε εκτεταμένα πειράματα τόσο στο σύνολο δεδομένων CIFAR [47] όσο και στο ImageNet [13] χρησιμοποιώντας μια πολύ απλοϊκή διαδικασία κλαδέματος με καθολική κατωφλίωση ως βάση και επιπλέον με ένα πρόσφατα προτεινόμενο σύστημα κλαδέματος ανά επίπεδο [73]. Η δική μας προσέγγιση αραιής εκπαίδευσης ξεπερνά τους (γενικά πιο δαπανηρούς υπολογιστικά) τρέχοντες SoA αλγορίθμους μη-δομημένου κλαδέματος, βελτιώνοντας σημαντικά τις προηγουμένως επιτευχθείσες ακρίβειες γενίκευσης των αραιών μοντέλων.

Συνολικά, η συμβολή της εργασίας μας μπορεί να συνοψιστεί ως εξής:

- Παρουσιάζουμε το Feather, μια ευέλικτη μονάδα αραιής εκπαίδευσης που μπορεί να χρησιμοποιηθεί για την αποδοτική και αποτελεσματική αποκοπή νευρωνικών δικτύων μέχρι και σε ακραία επίπεδα αραιότητας. Η προτεινόμενη μονάδα αξιολογήθηκε σε διαφορετικά συστήματα κλαδέματος και πέτυχε συνεχείς βελτιώσεις σε σχέση με το τρέχον SoA.
- Τονίζουμε τη σημασία μιας καλά σχεδιασμένης συνάρτησης κατωφλίου που βρίσκει μια λεπτή ισορροπία μεταξύ των δύο τυπικών, δηλαδή των hard και των soft τελεστών.
- Επισημαίνουμε τη συσχέτιση μεταξύ της κλιμάκωσης των gradients των κλαδεμένων βαρών και του ζητούμενου λόγου αραιότητας: οι στόχοι υψηλής αραιότητας πρέπει να συνοδεύονται από χαμηλότερες τιμές κλιμάκωσης για την παροχή κλαδεμένων μοντέλων υψηλής απόδοσης.

1.6.1 Προτεινόμενη Μέθοδος

Σε αυτή την εργασία, αναπτύξαμε μια νέα μονάδα κλαδέματος, που ονομάστηκε Feather. Το όνομα συμβολίζει την ανάλαφρη φύση της, την κομψότητα με την οποία επιτυγχάνει τη αραιότητα, καθώς και την ελαφρότητα των προκυπτόντων κλαδεμένων δικτύων. Μπορεί να χρησιμοποιηθεί σε διάφορα σχήματα κλαδέματος με βάση το μέτρο, συμπεριλαμβανομένων στρατηγικών όπου το κλάδεμα μπορεί να εκτελεστεί καθολικά ή κατά στρώματα. Μας ενδιαφέρει το κλάδεμα με βάση το μέτρο, όπου ένα βάρος διατηρείται μόνο εάν το μέτρο του υπερβαίνει μια τιμή κατωφλίου T, και ακολουθείται μια αραιή διαδικασία εκπαίδευσης (δηλ. εκτελείται η διαδικασία κλαδέματος κατά την τυπική πορεία της εκπαίδευσης), όπως γίνεται στα περισσότερα συστήματα SoA. Στην ουσία, αξιοποιούμε την αποτελεσματικότητα των σύγχρονων προσεγγίσεων που βασίζονται σε STE, αντιμετωπίζοντας προσεκτικά τα πιθανά τους προβλήματα. Από πλευράς υλοποίησης, η προτεινόμενη μονάδα εφαρμόζεται σε κάθε

επίπεδο, αντικαθιστώντας μια τυπική λειτουργία κλαδέματος, επηρεάζοντας τόσο το εμπρόσθιο όσο και στο οπίσθιο βήμα της εκπαίδευσης.

Στη συνέχεια, αρχικά παρέχουμε τις απαραίτητες λεπτομέρειες σχετικά με την αραιή εκπαίδευση με τον STE και έπειτα περιγράφουμε την προτεινόμενη μέθοδο, δίνοντας έμφαση στις τροποποιήσεις τόσο στο εμπρόσθιο όσο και στο οπίσθιο βήμα και τονίζοντας τα κίνητρα που την διέπουν.

Εισαγωγικά: Αραιή εκπαίδευση

Σε αυτή την ανάλυση εξετάζουμε τον τρόπο με τον οποίο εκτελείται το κλάδεμα και η επακόλουθη ενημέρωση των βαρών (κατά τη διάρκεια μιας επανάληψης εκπαίδευσης) σε ένα μόνο επίπεδο στο πλαίσιο του STE.

Αρχικά, ας θεωρήσουμε έναν τελεστή κατωφλίου, τον πυρήνα των προσεγγίσεων κλαδέματος με βάση το μέτρο, ως μια συνάρτηση $\mathcal{P}_{(T)}(x)$ που εκτελεί το κλάδεμα δεδομένης μιας τιμής κατωφλίου T. Κατά γενικό κανόνα η συνάρτηση αυτή παίρνει μηδενικές τιμές όταν $|x| \leq T$ και μη μηδενικές τιμές διαφορετικά. Τυπικές υλοποιήσεις αυτής της συνάρτησης είναι ο hard και ο soft τελεστής κατωφλίου.

Μια πρώτη προσέγγιση είναι να γίνει απευθείας οπισθοδιάδοση του τελεστής κατωφλίωσης. Με αυτόν τον τρόπο, τα gradients που ανήκουν στα κλαδεμένα βάρη μηδενίζονται αποκλείοντάς τα έτσι από το βήμα ενημέρωσης. Δυστυχώς, λόγω των αραιών gradients που προκύπτουν, αυτή η προσέγγιση μπορεί να οδηγήσει σε ανεπιθύμητο decay βαρών και σε αργή εξερεύνηση των πιθανών μοτίβων αραιότητας [44, 84].

Τελευταία, για να παρακαμφθούν τα προαναφερθέντα ζητήματα, ένα πλήθος μεθόδων κλαδέματος [73, 44, 84, 86] έχουν επιτύχει αποτελέσματα SoA με βάση τον Straight-Through Estimator [3]. Με λίγα λόγια, με σύμφωνα με την διατύπωση του STE, οι εξισώσεις ενημέρωσης βαρών και κατωφλίου αποσυνδέονται πλέον σε:

$$\tilde{\boldsymbol{w}}_k = \mathcal{P}_{(T)}(\boldsymbol{w}_k) \tag{1.6.1}$$

$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k - \eta \cdot \nabla \mathcal{L}(\tilde{\boldsymbol{w}}_k), \tag{1.6.2}$$

όπου $\boldsymbol{w}$ είναι τα βάρη του επιπέδου σε μορφή διανύσματος, $\tilde{\boldsymbol{w}}$ τα κλαδεμένα βάρη μετά την εφαρμογή της κατωφλίωσης, $\mathcal{L}(\boldsymbol{w})$ η συνάρτηση σφάλματος και η το μέγεθος βήματος. Οι αναφερόμενες σχέσεις αντιστοιχούν στην k-οστή επανάληψη μιας απλοϊκής υλοποίησης του Gradient Descent για να τονιστεί η επίδραση του STE. Η ίδια διαδικασία επεκτείνεται με τετριμμένο τρόπο σε οποιοδήποτε βελτιστοποιητή απαιτείται.

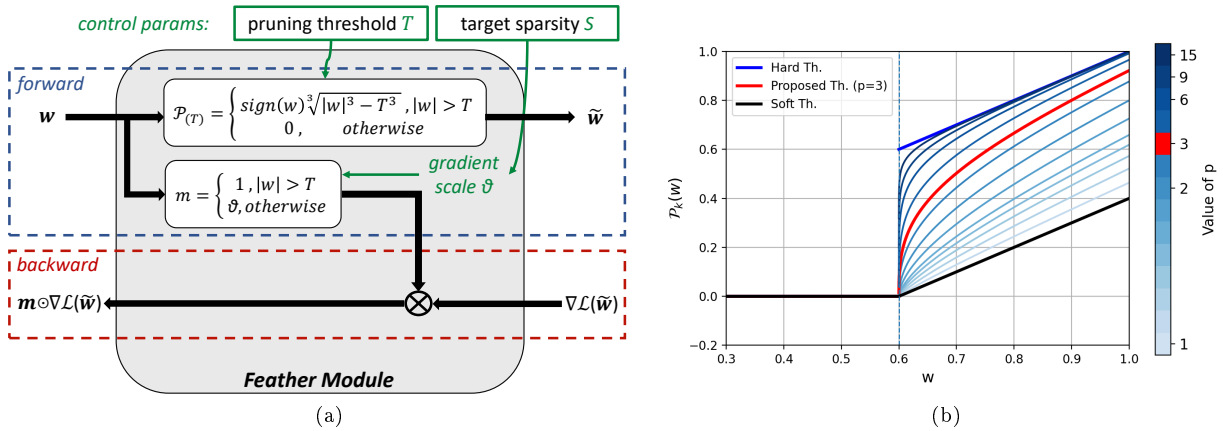
Η βασική ιδέα είναι να θεωρηθεί ο τελεστής κατωφλίου ως η ταυτοτική συνάρτηση κατά την οπισθοδιάδοση και να ενημερωθούν τόσο τα κλαδεμένα όσο και τα μη κλαδεμένα βάρη με βάση τα gradients του σφάλματος ως προς το αραιό σύνολο βαρών. Ωστόσο, κατά τη διάρκεια του εμπρόσθιου περάσματος, χρησιμοποιούνται μόνο τα αραιά βάρη, έτσι ώστε το δίκτυο να εκπαιδεύεται υπό τον περιορισμό της αραιότητας. Το βασικό πλεονέκτημα της αραιής εκπαίδευσης με τον STE είναι ότι επιτρέπει στα κλαδεμένα βάρη να γίνουν και πάλι ενεργά, εάν σε κάποιο σημείο κατά τη διάρκεια της εκπαίδευσης αποκτήσουν αρκετά μεγάλο μέτρο. Η διαδικασία αυτή προωθεί επομένως την εξερεύνηση διαφορετικών μοτίβων αραιότητας και έχει βρεθεί ότι οδηγεί σε καλύτερες επιδόσεις αραιών δικτύων [44, 84].

Προτεινόμενη Μονάδα Αραιής Εκπαίδευσης

Η προαναφερθείσα μοντελοποίηση με βάση τον STE αποτέλεσε το έναυσμα για την προτεινόμενη μονάδα; ο πρωταρχικός μας στόχος ήταν να διατηρήσουμε την απλότητα της αρχικής ιδέας και, ως εκ τούτου, επικεντρωθήκαμε στη βελτίωση δύο θεμελιωδών στοιχείων: του τελεστή κατωφλίωσης κατά το εμπρόσθιο βήμα και του χειρισμού των gradients κατά το βήμα οπισθοδιάδοσης. Συνολικά, οι προτεινόμενες βελτιώσεις αποσκοπούν στην υποβοήθηση της διαδικασίας εκπαίδευσης, προωθώντας τη σύγκλιση σε λύσεις με καλές επιδόσεις αλλά και μεγάλη αραιότητα. Η λειτουργικότητα του Feather, της προτεινόμενης μονάδας, συνοψίζεται στο Σχήμα 1.6.1a, όπου τα απεικονιζόμενα στοιχεία θα περιγραφούν λεπτομερώς στη συνέχεια.

STE Τελεστής Κατωφλίου

Εστιάζοντας στον τελεστή κατωφλίωσης που χρησιμοποιείται κατά τη διάρκεια του εμπρόσθιου περάσματος, ο πιο απλός τρόπος για τον ορισμό ενός βήματος κλαδέματος με βάση το μέτρο είναι μέσω της hard συνάρτησης κατωφλίωσης, η οποία είναι ασυνεχής στην τιμή κατωφλίου. Αυτή η ασυνέχεια μπορεί να οδηγήσει σε αστάθεια της εκπαίδευσης όταν τα βάρη περνούν από κλαδεμένες σε ενεργές καταστάσεις και αντίστροφα, καθώς το gradient που δέχονται μπορεί να μην δικαιολογεί την τιμή μετά την κατωφλίωση. Για την αντιμετώπιση αυτού



(a) (b)

Σχήμα 1.6.1: (α) Η προτεινόμενη μονάδα αραιής εκπαίδευσης, που χρησιμοποιεί το νέο τελεστή κατωφλίωσης και τη μάσκα κλιμάκωσης των gradients (β) Η προτεινόμενη οικογένεια τελεστών κατωφλίωσης για ποικίλες τιμές του p. Υιοθετούμε $p = 3$, με αποτέλεσμα την ισορροπία μεταξύ των δύο άκρων, hard και soft κατωφλίου αντίστοιχα.

του ζητήματος, μια πρόσφατη μέθοδος [86] πρότεινε ότι ο soft τελεστής κατωφλίωσης είναι προτιμότερος στο πλαίσιο του STE, ενώ η soft κατωφλίωση είναι μια κοινή επιλογή σε προσεγγίσεις κλαδέματος γενικά [50]. Η soft κατωφλίωση, αν και καταστέλλει την ασυνέχεια, προκαλεί ένα σταθερό bias (με τιμή ίση με αυτή του κατωφλίου) μεταξύ των ενεργών βαρών και των πυκνών ομολόγων τους, που ενημερώνονται κατά τη διάρκεια της φάσης οπισθοδιάδοσης του STE. Σημειώνουμε ότι από τη φύση του ο STE εισάγει μια ασυνέπεια μεταξύ του εμπρόσθιου και του οπίσθιου περάσματος, καθώς κατά τη διάρκεια του πρώτου χρησιμοποιείται το αραιό σύνολο βαρών, ενώ κατά τη διάρκεια του δεύτερου χρησιμοποιούνται τα υπολογισμένα gradients για την ενημέρωση των πυκνών βαρών.

Παρακινούμενοι από εργασίες στον τομέα της παλινδρόμησης, με στόχο την εύρεση τελεστών κατωφλίωσης που βρίσκονται μεταξύ soft και hard κατωφλίωσης [58, 27], προτείνουμε την ακόλουθη οικογένεια τελεστών για να θα χρησιμοποιηθούν με τον STE:

$$\mathcal{P}_{(T)}(w) = \begin{cases} \text{sign}(w) \cdot (|w|^p - T^p)^{1/p}, & \text{αν } |w| > T \\ 0, & \text{διαφορετικά} \end{cases} \tag{1.6.3}$$

Η συμπεριφορά του $\mathcal{P}_{(T)}$ απεικονίζεται στο Σχήμα 1.6.1b, για ποικίλες τιμές της παραμέτρου δύναμης p. Διαισθητικά, καθώς αυξάνεται το p αποκλίνουμε από τον soft ($p = 1$) και πλησιάζουμε τον hard τελεστή κατωφλίου. Υπό αυτή την οπτική γωνία, ο προτεινόμενος τελεστής μπορεί να θεωρηθεί γενίκευση των δύο υπαρχόντων. Σημειώστε ότι η προτεινόμενη συνάρτηση, όταν αποφεύγει ακραίες επιλογές για το p, προσπαθεί να ισορροπήσει μεταξύ των δύο προαναφερθέντων ιδιοτήτων: της συνέχειας και του bias. Με βάση αυτό, προτείνουμε την τιμή $p = 3$ ως έναν λογικό συμβιβασμό που αντιμετωπίζει επαρκώς και τα δύο ζητήματα.

Σε μια τελευταία σημείωση, θέλουμε να τονίσουμε ότι ο προτεινόμενος τελεστής κατωφλίωσης, εκτός του ότι έχει μια διαισθητική εξήγηση για την προκύπτουσα βελτιωμένη απόδοση (όπως αξιολογείται εμπειρικά στην Ενότητα 1.6.3), λόγω της απλότητάς του, πρακτικά δεν επιβαρύνει την εκπαίδευση και μπορεί να χρησιμοποιηθεί απευθείας σε συστήματα κλαδέματος σε συνδυασμό με τον STE.

STE και Κλιμάκωση των Gradients

Το κύριο κίνητρο πίσω από τον STE είναι να επιτραπεί η ροή των gradients στα κλαδεμένα βάρη και έτσι να καταστεί δυνατή η εξερεύνηση πολλαπλών μοτίβων αραιότητας κατά τη διάρκεια της αραιής εκπαίδευσης. Όπως αναφέρθηκε προηγουμένως, αυτό επιτυγχάνεται με τη θεώρηση της συνάρτησης κατωφλίου ως την ταυτοτική κατά την οπισθοδιάδοση. Ωστόσο, σε ορισμένες περιπτώσεις μπορεί να είναι επωφελές να περιοριστούν οι μεταβολές της μάσκας και να ευνοηθεί ένα πιο σταθερό μοτίβο αραιότητας. Διαπιστώνουμε ότι ένας μάλλον απλός, αλλά και διαισθητικός τρόπος ελέγχου της σταθερότητας της μάσκας είναι η κλιμάκωση των κλίσεων των κλαδεμένων βαρών με μια σταθερή τιμή $\theta \in [0, 1]$, τροποποιώντας ουσιαστικά το βήμα ενημέρωσης της Εξ. 1.6.2 σε:

$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k - \eta \cdot \boldsymbol{m}_k \odot \nabla\mathcal{L}(\tilde{\boldsymbol{w}}_k), \tag{1.6.4}$$

όπου $\boldsymbol{m}_k \in \{\theta, 1\}^N$ έτσι ώστε $m_{i,k} = 1$ αν $w_{i,k} > T$ και $m_{i,k} = \theta$ διαφορετικά, με $\odot$ να δηλώνει το γινόμενο στοιχείο προς στοιχείο.

Στις δύο ακραίες τιμές, $\theta = 0$ και $\theta = 1$, η μέθοδος ευθυγραμμίζεται με τις προσεγγίσεις μη-STE και την τυπική STE. Για ενδιάμεσες τιμές του θ, τα κλαδεμένα βάρη συνεχίζουν να λαμβάνουν gradients, αλλά με μειωμένα μεγέθη, και επομένως, σε κάποιο βαθμό, προωθούνται να γίνουν ανενεργά, με αποτέλεσμα μια πιο σταθερή (αλλά όχι εντελώς σταθερή) μάσκα.

Στα πειράματά μας (Ενότητα 1.6.3) διαπιστώσαμε ότι η κλιμάκωση των gradients γίνεται επωφελής όταν στοχεύουμε σε πολύ υψηλούς λόγους αραιότητας (π.χ. 98% και πάνω), όπου, λόγω της ύπαρξης πολύ λίγων ενεργών βαρών, οι πολύ συχνές μεταβολές της μάσκας φαίνεται να αποσταθεροποιούν το δίκτυο. Για πιο συντηρητικούς λόγους αραιότητας, η χρήση $\theta < 1$ δεν φαίνεται να βελτιώνει τα αποτελέσματα και μάλιστα οδηγεί σε μικρή μείωση της ακρίβειας σε ορισμένες περιπτώσεις.

Για το σκοπό αυτό, βασιζόμενοι σε πειραματικά στοιχεία, που ουσιαστικά στηρίζονται σε μια προσέγγιση profiling, ορίζουμε έναν "αυτόματο" τρόπο για τον καθορισμό του θ. Αναλυτικότερα, επιλέγουμε $\theta = g(S)$ ως μια απλή βηματική συνάρτηση, όπου $\theta = 0.5$ εάν η τελική αραιότητα στόχου S είναι πάνω από 95% και $\theta = 1.0$ όταν $S \leq 95\%$. Η τιμή του θ επιλέγεται στην αρχή της εκπαίδευσης και παραμένει σταθερή καθ' όλη τη διάρκεια της διαδικασίας εκπαίδευσης. Είναι ενδιαφέρον ότι δεν παρατηρήθηκαν κέρδη απόδοσης με την υιοθέτηση πιο σύνθετων τακτικών χρονοπρογραμματισμού (π.χ. με σταδιακή ομαλή μετάβαση από το θ από το 1 σε μια χαμηλότερη τιμή) κατά τη διάρκεια της εκπαίδευσης. Σημειώνουμε ότι ιδανικά θα μπορούσε να οριστεί μια πιο σύνθετη συνάρτηση δεδομένης μιας εξαντλητικής διαδικασίας profiling, αλλά θεωρήσαμε ότι τέτοιες ιδέες δεν εμπίπτουν στο πεδίο εφαρμογής της παρούσας εργασίας.

1.6.2 Εφαρμογή σε Συστήματα Κλαδέματος

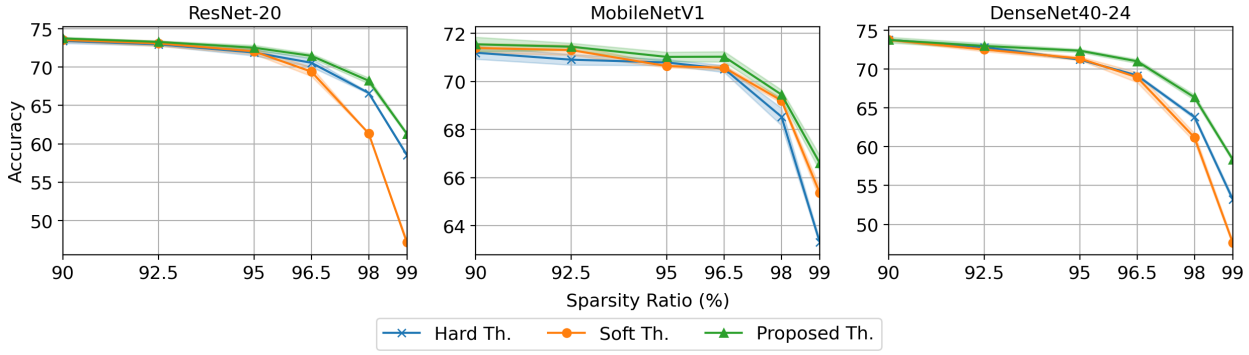
Η προτεινόμενη μονάδα αραιής εκπαίδευσης είναι ευέλικτη και δεν περιορίζεται σε ένα συγκεκριμένο σύστημα κλαδέματος. Θα επιδείξουμε την αποτελεσματικότητά της χρησιμοποιώντας δύο διαφορετικά συστήματα, τόσο ένα με καθολικό κατώφλι όσο και ένα σύστημα κλαδέματος που λειτουργεί με κατώφλια ανά επίπεδο. Το πρώτο χρησιμοποιεί καθολική κατωφλίωση, με την έννοια ότι επιλέγεται ένα ενιαίο κατώφλι T για όλα τα επίπεδα, το οποίο υπολογίζεται με ταξινόμηση όλων των βαρών, προκειμένου να κλαδέψει το δίκτυο μέχρι έναν καθορισμένο λόγο αραιότητας. Το τελευταίο, βασίζεται στο σύστημα ASL, βελτιωμένο όπως εξηγήθηκε προηγουμένως, το οποίο τώρα ονομάζεται ASL+. Και οι δύο προσεγγίσεις χρησιμοποιούν έναν χρονοπρογραμματιστή αραιότητας παρόμοιο με τον [98], όπου το δίκτυο εκπαιδεύεται πυκνά για έναν μικρό αριθμό εποχών προθέρμανσης, ακολουθούμενο από μια κυβική αύξηση του λόγου αραιότητας, μέχρι να φτάσει στον τελικό λόγο-στόχο, ο οποίος στη συνέχεια διατηρείται σταθερός για τις υπόλοιπες εποχές εκπαίδευσης.

1.6.3 Πειραματική αξιολόγηση

Ablation Μελέτες

Οι μελέτες ablation πραγματοποιήθηκαν στο σύνολο δεδομένων CIFAR-100, χρησιμοποιώντας το σύστημα κλαδέματος καθολικού κατωφλίου ως τη βάση για τη μονάδας κλαδέματος Feather. Σημειώνουμε ότι παρόμοιες συμπεριφορές παρατηρήθηκαν με τη χρήση του Feather σε συνδυασμό με το ASL+. Όλα τα σημεία του σχήματος αντιπροσωπεύουν μέσους όρους 3 εκτελέσεων και οι αντίστοιχες τυπικές αποκλίσεις εμφανίζονται ως σκιασμένες περιοχές.

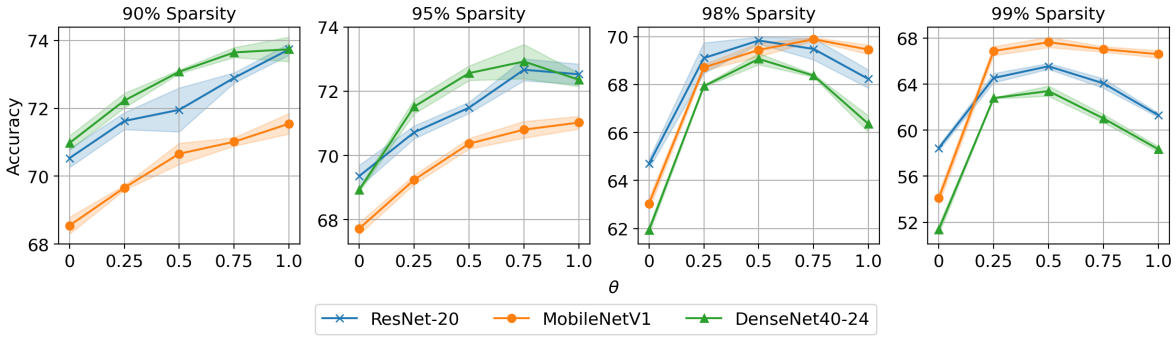
Επιπτώσεις του Τελεστή Κατωφλίωσης: Η αποτελεσματικότητα της προτεινόμενης συνάρτησης κατωφλίου (με $p = 3$) αξιολογείται εμπειρικά και συγκρίνεται με εκείνη των hard και soft συναρτήσεων κατωφλίου στο Σχήμα 1.6.2, ενώ δεν έγινε κλιμάκωση των gradients για αυτό το πείραμα (χρησιμοποιήσαμε τον τυπικό STE). Το προτεινόμενο κατώφλι επιτρέπει την εκπαίδευση ακριβέστερων αραιών δικτύων, ιδίως σε υψηλές αναλογίες κλαδέματος (95% και άνω). Αξίζει να σημειωθεί ότι η προσέγγιση του hard κατωφλίου επιτυγχάνει σημαντικά χαμηλά αποτελέσματα με το δίκτυο MobileNetV1, ενώ το soft κατώφλι με τα άλλα δύο. *Συνολικά, η προσέγγισή μας οδηγεί σταθερά σε καλύτερα εκπαιδευμένα δίκτυα, ανεξάρτητα από την αρχιτεκτονική και τον λόγο αραιότητας, υποστηρίζοντας τους ισχυρισμούς μας ότι συνδυάζουμε αποτελεσματικά τα πλεονεκτήματα τόσο της soft όσο και της hard κατωφλίωσης.*



Σχήμα 1.6.2: Μελέτη της επίδρασης του τελεστή κατωφλίωσης στην ακρίβεια του τελικού αραιού μοντέλου. Το προτεινόμενο κατώφλι υπερτερεί σταθερά έναντι των hard και soft τελεστών.

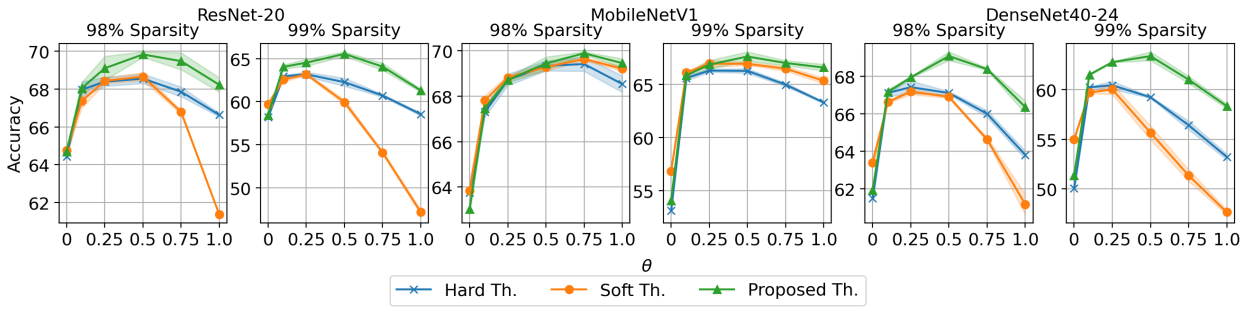
Επιπτώσεις της Κλιμάκωσης των Gradients: Στο Σχήμα 1.6.3 διερευνάται η σχέση μεταξύ της τιμής της παραμέτρου $\theta \in [0, 1]$, η οποία κλιμακώνει τα gradients των κλαδεμένων βαρών, και της τελικής ακρίβειας του μοντέλου για διαφόρους λόγους αραιότητας. Σημειώστε ότι η περίπτωση υποαπόδοσης με $\theta = 0$ είναι ισοδύναμη με μια παραλλαγή μη-STE. *Σε όλα τα εξεταζόμενα μοντέλα, μπορεί να παρατηρηθεί μια σαφής τάση - όταν στοχεύουμε σε χαμηλότερα επίπεδα αραιότητας, τα καλύτερα αποτελέσματα επιτυγχάνονται με τιμές του θ κοντά στη μονάδα, ενώ σε πιο ακραία ποσοστά αραιότητας (όπως 98% και 99%) οι βέλτιστες τιμές του θ φαίνεται να μετατοπίζονται προς το μέσο του εύρους του.*

Αυτή η παρατηρούμενη εξάρτηση είναι το κίνητρο πίσω από μια αυτόματη επιλογή του θ, όπως περιγράφεται στην Ενότητα 1.6.1, όπου $\theta = 0.5$ για $S > 95\%$ και $\theta = 1$ διαφορετικά. Σημειώστε ότι αυτή η επιλογή απλώς ευθυγραμμίζεται με την αναφερόμενη τάση και δεν είναι βέλτιστη για κάθε περίπτωση που αναφέρεται. Παρά το γεγονός ότι πρόκειται για μια πολύ "χονδροειδή" επιλογή, είναι πολύ αποτελεσματική, όπως υποδηλώνουν οι επερχόμενες πειραματικές αξιολογήσεις. Παρ' όλα αυτά, το συμπέρασμα αυτού του πειράματος δεν είναι μια απλή συνάρτηση, αλλά η ανάδειξη αυτής της σχέσης στο προσκήνιο. Με βάση αυτή την παρατήρηση, ανοίγουμε το δρόμο προς πιο σύνθετες συναρτήσεις ή, το πιο ενδιαφέρον, προς την κατεύθυνση της ύπαρξης διαφορετικών κλιμάκων ανά επίπεδο, βασιζόμενοι στη αραιότητα ανά επίπεδο και όχι στη συνολική αραιότητα. Η τελευταία ιδέα, μια πιθανή μελλοντική κατεύθυνση με πρακτική αξία, δηλώνει απλώς ότι τα πιο πυκνά επίπεδα μπορούν να είναι πιο ευέλικτα στις μεταβολές των προτύπων αραιότητας από τα υπερβολικά κλαδεμένα.



Σχήμα 1.6.3: Μελέτη της επίδρασης της κλιμάκωσης των gradients. Υπό συντηρητική τελική αραιότητα, το θ κοντά στη μονάδα είναι προτιμότερο, ενώ όταν στοχεύουμε σε υψηλή αραιότητα, τα μοντέλα επωφελούνται από το θ κοντά στη μέση του εύρους του.

Κλιμάκωση των Gradients υπό Διαφορετικές Συναρτήσεις Κατωφλίωσης: Τέλος, συγκρίνουμε τον αντίκτυπο της κλιμάκωσης των gradients χρησιμοποιώντας τους τρεις διαφορετικούς τύπους τελεστών κατωφλίου. Το Σχήμα 1.6.4 δείχνει ότι, ανεξάρτητα από την επιλογή του τελεστή, η χρήση κλίμακας θ $στο (0, 1)$ είναι επωφελής για την τελική ακρίβεια στα καθεστώτα υψηλής αραιότητας. Παρόλα αυτά, η συνάρτηση κατωφλίωσης που προτείνουμε διατηρεί το προβάδισμα στην απόδοση σε σύγκριση με τις δύο τυπικές.



Σχήμα 1.6.4: Η κλιμάκωση των gradients βελτιώνει την τελική ακρίβεια σε υψηλή αραιότητα, ανεξάρτητα από τον τελεστή κατωφλίωσης, ενώ η μέγιστη απόδοση επιτυγχάνεται αν συνδυαστεί με το προτεινόμενο κατώφλι.

1.6.4 Σύγκριση με το SoA

CIFAR-100: Για αυτά τα πειράματα, συγκρίνουμε άμεσα τα αποτελέσματά μας με αυτά των ST-3 [86] και Spartan [84], που είναι οι δύο πιο πρόσφατες και με τις καλύτερες επιδόσεις προσεγγίσεις αραιής εκπαίδευσης. Οι μέθοδοι είναι και οι δύο αλγόριθμοι κλαδέματος βασισμένοι σε καθολικό κατώφλι μέτρου, οι οποίοι εισάγουν σταδιακά αραιότητα κατά τη διάρκεια της εκπαίδευσης, χρησιμοποιώντας παραλλαγές του STE. Συγκεκριμένα, η ST-3 υιοθετεί soft κατωφλίωση και μια τεχνική αναπροσαρμογής βάρους παρόμοια με αυτή που χρησιμοποιείται από το dropout [79], ενώ η Spartan υπολογίζει μια soft top-k μάσκα επιλύοντας ένα regularized Optimal Transportation πρόβλημα, επομένως είναι πιο δαπανηρή υπολογιστικά από την προσέγγισή μας.

Συγκεκριμένα, ο Πίνακας 1.3 παρέχει τα αποτελέσματα που προέκυψαν στο CIFAR-100 με τη χρήση των προαναφερθέντων μεθόδων σε τρεις διαφορετικές αρχιτεκτονικές και τέσσερα διαφορετικά επίπεδα αραιότητας. Τα αναφερόμενα αποτελέσματα αντιστοιχούν σε μέσους όρους 3 εκτελέσεων με τις αντίστοιχες τυπικές αποκλίσεις. Τα αποτελέσματα καταδεικνύουν ότι η αραιή εκπαίδευση με το Feather αποδίδει σταθερά ακριβέστερα μοντέλα σε σύγκριση με τις δύο SoA μεθόδους, είτε χρησιμοποιώντας την απλή προσέγγιση καθολικού κατωφλίου (Feather-Global) είτε σε συνδυασμό με το ASL+. Αξίζει να σημειωθεί ότι η διαφορά στην ακρίβεια μεταξύ της προσέγγισής μας και της αμέσως καλύτερης μεθόδου αυξάνεται έως και 4% όταν εξετάζονται τα 99% αραιά μοντέλα ResNet-20 και DenseNet40-24. Μια άλλη ενδιαφέρουσα παρατήρηση είναι ότι το Feather μπορεί να οδηγήσει σε 90% αραιά μοντέλα ResNet-20 και MobileNetV1 με ελαφρώς καλύτερες ακρίβειες γενίκευσης από εκείνες των πυκνών αντίστοιχων μοντέλων, που εκπαιδεύονται με τον ίδιο αριθμό εποχών. Η τελευταία παρατήρηση υποδηλώνει ότι μια καλά σχεδιασμένη μέθοδος αραιής εκπαίδευσης μπορεί ακόμη και να είναι επωφελής, όχι μόνο για την παραγωγή συμπαγών μοντέλων, αλλά και για τη βελτίωση των επιδόσεων γενίκευσης, όταν λαμβάνονται υπόψη σχετικά συντηρητικοί λόγοι αραιότητας. Σημειώνουμε ότι παρατηρούνται παρόμοια αποτελέσματα SoA με τη χρήση και των δύο εξεταζόμενων συστημάτων βάσης (Global και ASL+), ένα σημείο που επικυρώνει περαιτέρω την ευελιξία του Feather.

ImageNet: Για τα πειράματα στο ImageNet συμπεριλαμβάνουμε αποτελέσματα από τη βιβλιογραφία από έναν εκτεταμένο αριθμό μεθόδων κλαδέματος που επιτεύχθηκαν χρησιμοποιώντας τον ίδιο αριθμό εποχών (100) και επαύξησης δεδομένων. Συγκεκριμένα, αποτελέσματα από τις σχετικές μεθόδους GMP [98], DNW [91], STR [50], ProbMask [97], OptG [95], ST-3 [86] και Spartan [84] παρουσιάζονται στον Πίνακα 1.4. Με βάση τα αποτελέσματα που παρέχονται στον Πίνακα 1.4, η μέθοδος αραιής εκπαίδευσής μας, χρησιμοποιώντας την προσέγγιση καθολικού κατωφλίου, μπορεί να παρέχει σημαντικά καλύτερα αποτελέσματα από εκείνα των προηγούμενων SoA μεθόδων, ειδικά για πολύ απαιτητικούς λόγους αραιότητας. Θέλουμε να τονίσουμε ότι οι βελτιωμένες επιδόσεις με το Feather δεν έρχονται με κόστος στο χρόνο εκπαίδευσης ή την ανάγκη για περίπλοκες ρυθμίσεις υπερπαραμέτρων, σε αντίθεση με πολλές από τις υπάρχουσες μεθόδους. Αντίθετα, τα προκύπτοντα κέρδη σε ακρίβεια μπορούν να αποδοθούν στην απλή, αλλά προσεκτικά διατυπωμένη τροποποιημένη προσέγγιση αραιής εκπαίδευσης με πυρήνα τον STE.

1.6.5 Συμπεράσματα

Αυτή η εργασία προτείνει το Feather, μια αποτελεσματική και αποδοτική μονάδα αραιής εκπαίδευσης που μπορεί εύκολα να εφαρμοστεί σε διάφορα συστήματα κλαδέματος. Ειδικότερα, όπως αποδεικνύεται από εκτεταμένα

Ratio	90%	95%	98%	99%
ResNet-20 (1.096M Params): 73.59 ±0.44				
ST-3 [86]	72.81 ±0.13	71.72 ±0.20	67.53 ±0.53	58.32 ±0.17
Spartan [84]	72.56 ±0.35	71.60 ±0.40	67.27 ±0.31	61.70 ±0.21
Feather-Global	**73.74** ±0.17	**72.53** ±0.32	**69.83** ±0.14	**65.55** ±0.25
Feather-ASL+	72.86 ±0.10	72.42 ±0.17	69.76 ±0.09	64.95 ±0.47
MobileNetV1 (3.315M Params): 71.15 ±0.17				
ST-3 [86]	70.94 ±0.25	70.44 ±0.23	69.40 ±0.06	66.63 ±0.15
Spartan [84]	70.52 ±0.51	69.01 ±0.11	65.52 ±0.24	60.65 ±0.22
Feather-Global	**71.55** ±0.30	71.03 ±0.20	**69.44** ±0.29	67.64 ±0.45
Feather-ASL+	71.10 ±0.31	**71.26** ±0.10	69.42 ±0.12	**67.86** ±0.03
DenseNet40-24 (0.714M Params): 74.70 ±0.51				
ST-3 [86]	72.56 ±0.31	71.21 ±0.35	65.48 ±0.18	56.18 ±0.60
Spartan [84]	73.13 ±0.25	71.61 ±0.04	65.94 ±0.07	58.64 ±0.18
Feather-Global	73.75 ±0.36	72.36 ±0.21	69.06 ±0.23	**63.40** ±0.44
Feather-ASL+	**73.92** ±0.19	**72.47** ±0.12	**69.08** ±0.19	62.94 ±0.14

Πίνακας 1.3: Σύγκριση της ακρίβειας Top-1 στο CIFAR-100.

Ratio	90%	95%	98%	99%
ResNet-50 (25.6M Params): 77.10				
GMP [98]	73.91	70.59	57.90	44.78
DNW [91]	74.00	68.30	58.20	-
STR [50]	74.31	70.40	61.46	51.82
ProbMask [97]	74.68	71.50	66.83	61.07
OptG [95]	74.28	72.45	67.20	62.10
ST-3 [86]	76.03	74.46	70.46	63.88
Spartan [84]	76.17	74.68	-	63.87
Feather-Global	**76.93**	**75.27**	**72.92**	**68.85**

Πίνακας 1.4: Σύγκριση της ακρίβειας Top-1 στο ImageNet.

πειράματα στα σύνολα δεδομένων CIFAR και ImageNet, χρησιμοποιώντας τόσο μια καθολική όσο και μια ανά επίπεδο προσέγγιση κατωφλίωσης, οδηγεί σε βελτίωση των SoA αποτελεσμάτων, ιδίως σε υψηλούς λόγους κλαδέματος. Επιπλέον, η επιτυχία της μεθόδου μας υποδεικνύει τις μεγάλες προοπτικές της σωστής κατανόησης και, κατά συνέπεια, της βελτίωσης της αραιής εκπαίδευσης με τη χρήση μιας προσέγγισης βασισμένης στον STE, η οποία παρά την απλότητά της αποδεικνύεται ιδιαίτερα αποτελεσματική.

Chapter 2

Introduction

2.1 Motivation

Deep Neural Networks have shown impressive capabilities in solving various complex machine learning problems, with their performance being continuously enhanced thanks to the efforts of the research community [33, 48, 25]. To achieve that, the neural network models have been made larger, deeper and more complex and thus require significant resources for training and inference, including large amounts of system memory to store the parameters and intermediate calculations, considerable computing power in order to maintain reasonable processing times as well as ample of energy supply to the involved systems [14]. Unfortunately, the rapid growth in resource demands of the state-of-the-art DNNs has not been met by an equivalent advancement in the hardware (processors and memories) utilized by compact devices, which is limited by small size and low energy requirements [14]. As a consequence, many real-world applications, such as in robotics, smart wearables, self driving cars and smart cities, among others, that rely on machine learning systems running on resource-limited devices cannot benefit from the enhanced capabilities of such networks.

As a response, a lot of effort has been devoted in recent years towards creating compact and efficient (*i.e.* compressed) versions of these DNNs [14, 57, 26]. Among the numerous model compression approaches, DNN pruning [54, 21, 57] has been widely studied and has shown very promising results [21, 73, 97, 84]. It involves the removal of network elements (such as individual weights or entire filters and channels) that are considered unnecessary and have minor or no effect to its final performance. Surprisingly, it has been observed that originally large dense models that have been made sparse via pruning consistently outperform smaller dense models that have the same number of non-zero parameters, often exhibiting no losses in accuracy up to moderate compression ratios [98, 73]. In essence, this phenomenon, although still lacking a thorough theoretical understanding, is generally attributed to the effect overparameterization has on enabling a more effective training with stochastic gradient decent [94, 7], as well as to the extra degrees of freedom introduced in terms of the allocation of sparsity [67, 84].

2.2 Contributions

Recently, there has been particular interest within the DNN compression community for the development of methods that perform the pruning efficiently within the standard course of training [73, 44, 84, 86], *i.e.* without the need for further retraining and pruning rounds. In this thesis, we build upon this body of work, focusing on removing individual weights based on their magnitudes during training. Our main contributions are as follows:

- In the first main part of the thesis (Chapter 5), we propose a layer-wise pruning framework which, with almost no extra training overhead and without the need for setting additional hyperparameters, can sparsify a model during training up to an exact user-defined level while keeping an effective non-uniform per-layer budget.
- Additionally, we provide evidence that the per-layer weights can be effectively modeled by probability distributions, in particular the Gaussian and the Laplace, in order for the appropriate per-layer thresholds for magnitude based pruning to be learned, even at high sparsity regimes.
- Subsequently, we introduce Feather (Chapter 6), a novel sparse training module that utilizes an enhanced version of the Straight-Through Estimator [3], based on a new thresholding operator and a gradient scaling technique.
- Through numerous ablation studies we highlight the importance Feather's well-crafted thresholding function that finds a fine balance between the two standard ones, namely hard and soft operators. Additionally, we showcase the correlation between scaling the gradients of the pruned weights and the targeted sparsity ratio. We find that high sparsity targets should be accompanied with lower scaling values to provide high performing pruned models.
- All our findings are supported by extensive experiments on CIFAR [47] and ImageNet [13] datasets. Our layer-wise pruning framework can perform on par with most recent methods, while utilizing the Feather module to common frameworks (including ours) results to considerably improving the state-of-the-art in terms of Top-1 validation accuracy, particularly at very challenging sparsities.

Part of our work has been accepted as an Oral paper [24] in the 34th British Machine Vision Conference (BMVC 2023), with the authors being Athanasios Glentis Georgoulakis, George Retsinas and Petros Maragos.

2.3 Thesis Outline

The rest of this thesis is divided into 5 chapters, as described below:

- Chapter 3 provides an overview of the machine learning and deep learning theoretical concepts that are relevant to this work.
- Chapter 4 briefly introduces related DNN compression techniques and provides a more in depth analysis of the various pruning approaches.
- Chapter 5 presents a framework that trains accurate sparse networks without prolonging the training time, using adaptive weight distribution modeling for finding the layer-wise pruning thresholds.
- Chapter 6 deals with enhancing the Straight-Through Estimator based sparse training approach by introducing a novel pruning module, Feather, that based on extensive experimentation surpasses state-of-the-art methods, producing more accurate sparse models.
- Chapter 7 provides a summary of the thesis and its contributions and briefly discuses potential future directions.

Chapter 3

Theoretical Background

3.1 Machine Learning

This section is intended to provide the reader with an overview of the necessary theoretical background on the field of machine learning [66, 4]. By the term machine learning we refer to an extensive range of methods used to program computers to uncover patterns and make decisions based on data. This data-driven approach has enabled computers to excel at various fields, such as in vision, that would have been unfeasible to be explicitly programmed to perform the desired way. In the following sections the main machine learning categories, based on the task and the data at hand, will be analyzed while a number of key concepts of the field will be described.

3.1.1 Machine Learning Paradigms

In order to solve a task using a machine learning method a set of data, the dataset, is needed. This dataset can be presented to the computer in a number of ways, depending on the application, leading to mainly to three different machine learning paradigms, supervised leaning, unsupervised Learning and reinforcement learning. The details of each paradigm will be discussed in the next paragraphs.

Supervised Leaning

By far the most common approach in machine learning is called supervised learning. It is characterized by the use of labeled datasets, where the desired output (label) is paired with each corresponding data sample. Those labels are then used to "teach" or "supervise" the system to return the correct outputs given seen or (most importantly) unseen data samples. More specifically, a supervised learning algorithm aims at learning a mapping between the problem's inputs x and the outputs y, provided with a dataset, $\{(x_1, y_1), ..., (x_N, y_N)\}$, with x_i and y_i being the i-th input sample (commonly referred as a feature vector) and the i-th output or prediction, respectively. Supervised learning problems are further divided into classification if the outputs y are discrete or regression if the outputs take continuous values.

Unsupervised Learning

As the name suggests, unsupervised learning methods lack "supervision", meaning that the system is optimized using unlabeled data. The objectives of such methods can vary, with some prominent examples being clustering, density estimation and dimensionality reduction. Clustering is defined as the task of grouping input samples that are similar to each other to the same groups and thus discovering the hidden clusters of the dataset. Density estimation, another popular unsupervised learning application, refers to computing an estimate of the unobservable distribution of the input data. Finally, dimensionality reduction aims at projecting the input data, which are usually high dimensional, into a lower dimensional space, in such a way that meaningful properties of the original data are preserved or better revealed.

Reinforcement learning

A third category of machine learning is reinforcement learning, commonly used in Robotics and game playing, where autonomous agents learn to take sequences of actions in their environment. In contrast to supervised or unsupervised learning, the notion of dataset does not apply for this type of learning. Instead, the data are obtained progressively from the environment as the agent interacts with it, makes observations and receives rewards. The aim of reinforcement learning is to teach the agent to take actions given the state it is in and its past experiences, i.e. to find a policy. Typically to goal is to learn a policy that results to the maximization of the rewards that correspond to the observations made, through trial and error, using feedback.

3.1.2 Machine Learning Concepts

Loss function

Typical machine learning algorithms use gradient-based optimization procedures [75] in order to adjust the model's parameters θ, to best represent the data. This is achieved by minimizing a scalar function $L(\theta)$ (almost everywhere) differentiable w.r.t. the parameters, known as the loss function, that measures the loss of the model on a given dataset as a notion of the distance between the labels y and the model's outputs

$\hat{y} = f_\theta(x)$. Depending on the application, there exist numerous instantiations of the loss function. Some common ones for regression and classification are explained as follows:

Regression Losses:

- **Mean Squared Error (MSE):** A wildly used loss function for regression problems, known also as the L2 loss, that measures the average squared difference (error) of the labels y and the model's outputs $\hat{y}$. Due to the use of squared differences, large errors are penalized more than small ones. Mathematically is defined as:

$$L_{\text{MSE}}(\theta) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{3.1.1}$$

- **Mean Absolute Error (MAE):** Measures the mean of the absolute prediction errors and is defined as:

$$L_{\text{MAE}}(\theta) = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i| \tag{3.1.2}$$

  Note that this loss is not differentiable at zero, i.e. when $y_i = \hat{y}_i$, although when handled appropriately this does not causes problems to the optimization.

Classification Losses:

- **Cross-Entropy Loss:** By far the most common loss function for classification, is based on the definition of cross-entropy between two probability distributions. For discrete probability distributions p and q (with the same support $\mathcal{X}$) the cross-entropy is defined as:

$$H(p, q) = -\sum_{x\in\mathcal{X}} p(x)\log q(x) \tag{3.1.3}$$

  For the case of binary classification, assuming labels $y \in \{0, 1\}$ and predictions as probability values, the Binary Cross-Entropy loss over a dataset of N samples is defined as:

$$L_{\text{BCE}}(\theta) = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log \hat{y}_i + (1 - y_i)\log(1 - \hat{y}_i)] \tag{3.1.4}$$

  The above loss increases as the predicted probabilities $\hat{y}_i$ differ from the labels y_i, with confident incorrect predictions being penalized the most. This type of loss can be extended to multi-class classification problems, resulting to the Categorical Cross-Entropy loss:

$$L_{\text{CCE}}(\theta) = -\frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{M} y_{i,k}\log(\hat{y}_{i,k}), \tag{3.1.5}$$

  where M is the number of classes, $\hat{y}_{i,k}$ represents the probability that the i-th data sample belongs to the class k and $y_{i,k} = 1$ only if the i-th sample belongs to the k-th class, otherwise $y_{i,k} = 0$.

- **Kullback-Leibler Divergence Loss:** A loss similar to the cross-entropy one, which is based on the Kullback-Leibler divergence between two probability distributions p and q. It quantifies the difference between the distribution p from the reference one, q. For the discrete case (with p, q having the same support $\mathcal{X}$) is defined as:

$$D_{KL}(p||q) = \sum_{x\in\mathcal{X}} p(x)\log\left(\frac{p(x)}{q(x)}\right) \tag{3.1.6}$$

Evaluation Metrics

A function which is used to evaluate the performance of a machine learning model, either at an intermediate optimization step or after the training process, is called a metric. Although any loss function can be used as a metric, differentiability of metric functions (w.r.t. the model's parameters) is not a requirement since

they are not used directly for optimizing the model. Therefore, depending on the nature of the task, a wide variety of functions can be chosen. Specifically for the task of classification, some popular metrics are listed bellow:

- **Accuracy:** Is the fraction of the correct classified samples. That is, it is defined as the ratio of the number of correct predictions to the total number of samples of the dataset.
- **Precision:** Is the fraction of the number of true positive (TP) samples of a class to the total number of samples predicted to belong to that class, i.e. true positive and false positive (FP).

$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

  It is preferred as a metric for tasks that prioritize the false positive errors to be avoided.

- **Recall:** Is the fraction of the number of true positive samples of a class to the total number of samples of that class, i.e. true positive and false negative (FN).

$$\text{recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

  It is usually used as a metric for tasks that is important to minimize false negatives that are consider more "costly" errors.

- **F1-score:** Combines the two previous metrics using their harmonic mean:

$$\text{f1-score} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

Train-Validation-Test Sets

In order for a model to be capable of performing a classification or a regression task it first needs a set of data to be used for the optimization of its parameters. This procedure of fitting the parameters of the model to the dataset is called training while the dataset used for this purpose is called the training set. Since the goal is for the model to be used on unseen data in the real world, evaluating it (using evaluation metrics) on the same set it was trained on is not suited for providing an estimate of its expected performance. For that reason a separate dataset is used, which is called the test set, comprising of samples that were not used for the optimization of the parameters (unseen samples). Finally, for models and training algorithms that have additional controlling parameters, called hyperparameters, which influence the learning process and the values of the trainable model parameters, a third type of dataset is used. This is the validation set and is used to determine the values of the hyperparameters.

Generalization - Underfitting - Overfitting

For any machine learning model to have practical value for real world applications it needs to be able to handle well not only training data but also previously unseen inputs. This desired ability of a model to process new data successfully is called generalization. As previously mentioned, there is a specific dataset for evaluating the model's performance on unseen inputs (i.e. the generalization performance), the test set. In addition, comparing the train set performance with the test set one can provide us information about how to improve the later. A common case is when the algorithm performs accurately on the training data but poorly on unseen test set data. This is a clear indication that the involved parameters have obtained values that fit the training data too closely, resulting to falsely modeling the contained noise and data irregularities. This behavior is called overfitting and needs to be addressed in order for the generalization performance to increase. A number of methods may be used to combat overfitting, such as lowering the complexity of the model, training with more data or using regularization techniques (discussed in the following section). The opposite to overfitting is called underfitting and occurs when the model is unable to achieve an adequately good fit of the training data, due to reasons such as performing an inadequate number of training iterations or having a model with insufficient complexity.

Regularization

A very popular way to avoid overfitting is called regularization [9]. Generally the term regularization in machine learning refers to any kind of alterations to the learning algorithm that aim at improving the model's generalization performance by preventing overfitting during training. One common way to explicitly introduce regularization involves adding an extra term to the loss function, the regularization or penalty term, that promotes simpler solutions by penalizing parameters with large magnitudes. Denoting this new term as $R(\theta)$, the loss function is now written as:

$$L_\lambda(\theta) = L(\theta) + \lambda R(\theta), \tag{3.1.7}$$

where $L(\theta)$ is some loss term (such as the ones previously described) and λ a hyperparameter that determines the strength of the regularization imposed. A very small λ value may result to being inadequate to prevent overfitting, while a very high one may lead an underfitted model. The two most frequently used regularization terms, namely the L_1 and $L2$ are briefly discussed as follows.

- **L_2 regularization:** Sometimes also called Ridge regression or weight decay, is based on using the L_2-norm on the parameters θ, resulting to the following form of the term R:

$$R_{L_2}(\theta) = ||\theta||_2 = \sum_i \theta_i^2 \tag{3.1.8}$$

  The use of this squared parameter sum term results to large weights being penalized and thus gives rise to simpler solutions, that tend to model the data more accurately avoiding overfitting.

- **L_1 regularization:** Also known as LASSO, now the term R is defined as the L_1-norm of the parameters θ:

$$R_{L_1}(\theta) = ||\theta||_1 = \sum_i |\theta_i| \tag{3.1.9}$$

  This regularizer tends to shrink the parameters to zero, promoting sparse solutions that consist of many zeroed parameters. Therefore, apart from the purpose of improving the generalization, it can also be used as a feature selection mechanism since zeroing a subset of parameters results to not using the corresponding input features.

Certainly, apart from adding an extra term to the loss function, there exist many other ways of to regularize the learning process. Some popular ones include early stopping, which refers to terminating the training sooner by monitoring the validation performance in order to avoid overfitting or dropout [79], a method designed for neural networks that randomly selects neurons to be dropped during training.

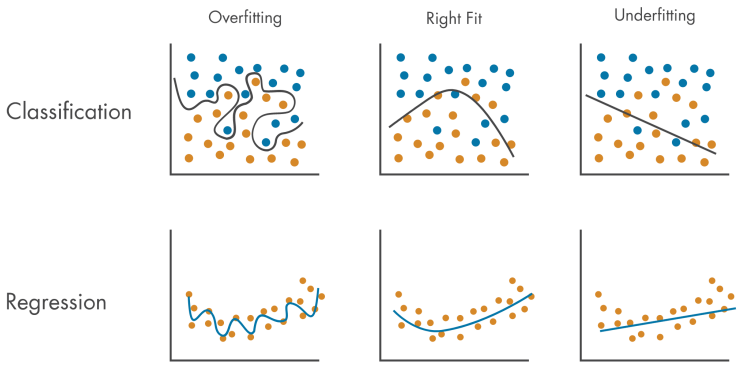
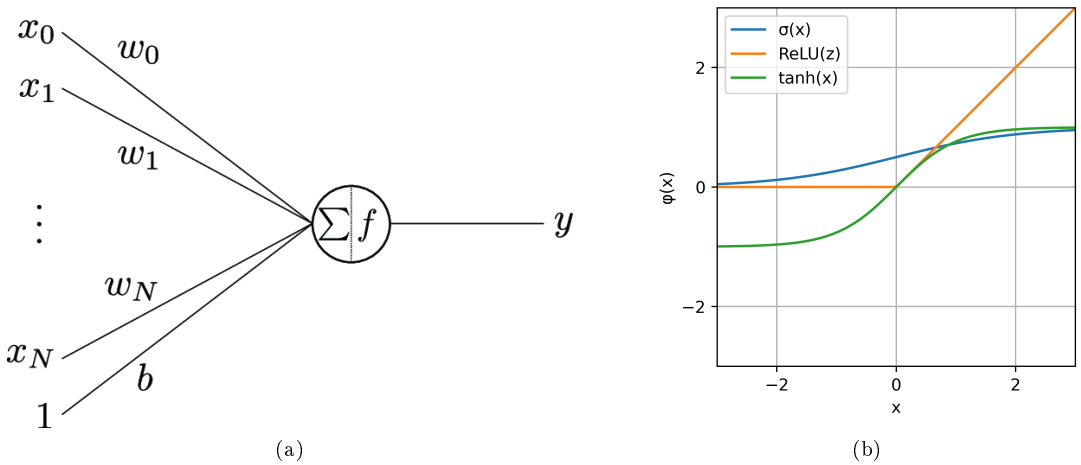


Figure 3.1.1: Illustration of overfitting and underfitting for classification and regression. From [65].



(a) (b)

Figure 3.2.1: A neuron (a) and some common activation functions (b). (a) from [39].

3.2 Deep Learning

Having covered the basics of machine learning, we will now focus our attention on deep learning models and methods [53, 25], since this is the main filed of application of the various model compression techniques, including pruning. Deep learning methods' most prominent advantage is that they can receive raw data (e.g. image pixel values) as input and still reach impressive levels of performance, enabling the use of machine learning for applications that manual feature extraction was difficult and much less effective. This is in contrast to traditional machine learning methods, which required the raw data to be first manually transformed into features that the system could effectively handle. The way deep learning models, called Deep Neural Networks (DNNs), process raw input data successfully is by automatically performing the feature extraction task during their training on large datasets (usually much larger than the ones needed for traditional ML methods), being able to learn features with different abstraction levels due to the way they are constructed. In the following sections we will explain in more detail how the most common DNN models work and will briefly discuss the various methods involved in deep learning training.

3.2.1 Deep Learning Architectures

Artificial Neural Networks (ANNs) are an important category of machine learning models, being loosely designed based on their biological counterparts. Their key structural elements are units (or nodes) called artificial neurons. Those units typically receive a number of input values, multiply them with weights and finally pass the sum of the resulting products (plus a bias term) through a non-linearity, called the activation function, in order to produce their final output. Mathematically, the output y of a neuron can be written as:

$$y = \phi\left(\sum_{i=0}^{N} w_i x_i + b\right), \tag{3.2.1}$$

where $\boldsymbol{x} = \{x_i\}$ is the input of the neuron, $\boldsymbol{w} = \{w_i\}$ the weights, b a bias term and $\phi()$ the activation function. A graphical depiction is also shown in Figure 3.2.1.

Common instantiations of the activation function include the Sigmoid function, the Rectified Linear Unit or ReLU and the hyperbolic tangent or Tanh:

$$\sigma(z) = \frac{1}{1+e^{-z}} \tag{3.2.2}$$

$$\text{ReLU}(z) = \max(0, z) \tag{3.2.3}$$

$$\tanh(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}} \tag{3.2.4}$$

In short, the use of non-linear activation functions is essential in order for ANNs, comprising of multiple neurons, to be able to produce complex, non-linear decision boundaries using non-linear combinations of inputs and weights. Neurons are typically structured in layers, with the one closer to the input being called

the input layer, the last called the output layer and the intermediate ones the hidden layers. Information passes from the input to the output layer through the hidden layers, possibly with feedback. Usually ANNs are referred as deep when having two or more hidden layers.

Feedforward Neural Networks

Perhaps the most popular deep learning models are the Feedforward Neural Networks (FFNN), with the term feedforward implying that signals flow from the input to the output layer, through the intermediate hidden layers, without the use of feedback connections. Mathematically, an FFNN can be considered as a function $f(\boldsymbol{x}, \boldsymbol{W})$ that, given the input signal $\boldsymbol{x}$ and a matrix of learnable weights $\boldsymbol{W}$, produces an output $\boldsymbol{y}$. Two very common types of FFNNs are the Fully Connected Neural Networks (FCNN) and the Convolutional Neural Networks (CNNs), which are covered as follows.

Fully Connected Neural Networks:

Also referred as Multilayer Perceptrons (MLPs), are the simplest types of FFNNs and consist of a number of fully connected layers, where every neuron in one layer is connected to every neuron of the next layer. The number of neurons of a layer is called its width, while different layers can have different widths, with the widths of the input and the output layers being of course equal to the dimensions of the input and output vectors, respectively. On the other hand, the number of layers of a model is called its depth. A graphical representation of a fully connected network is depicted in Figure 3.2.2.

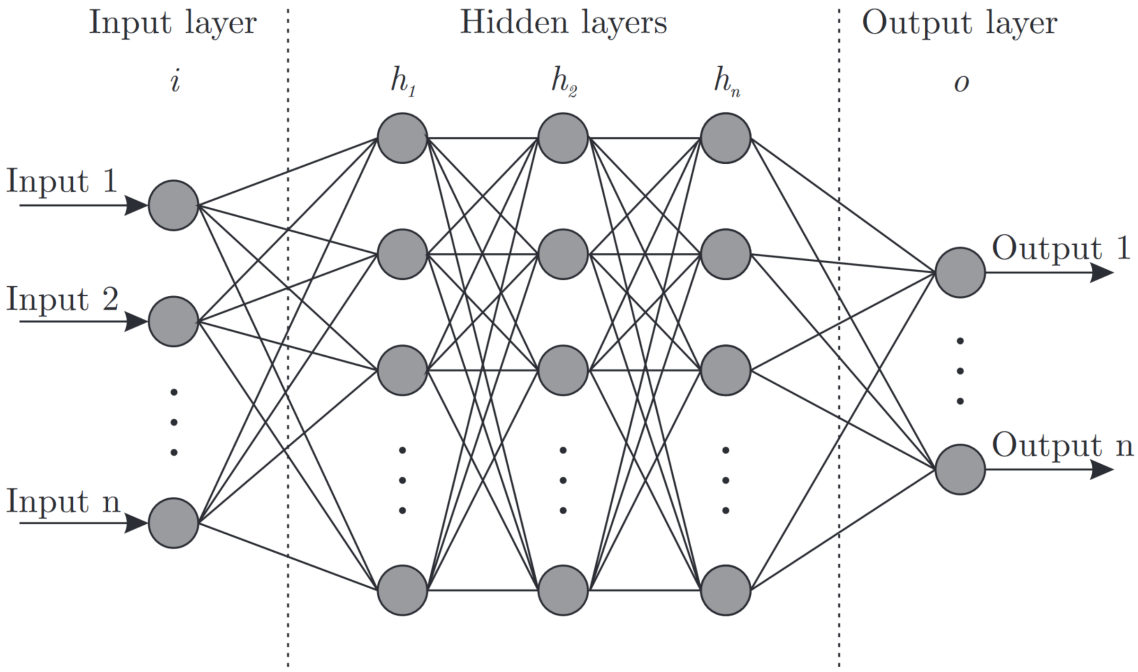


Figure 3.2.2: Graphical representation of a Fully Connected Neural Network. From [6].

The use of multiple hidden layers, which the reason this models are called deep, enable FCNNs (and other types of networks in general) to effectively extract feature representations of increasingly complex and more abstract nature, as the information is being processed from lower layer (closer to the input) to higher ones (closer to the output).

FCNNs are build in a way to be general enough to accept various types of inputs, i.e. there is no need to make restricting assumptions about the structure of the input. Although this property of FCNNs makes them effective for a wide variety of applications, for some tasks it tend to be very beneficial to use more specialized versions of neural networks that exploit the structural properties of the input they are intended for.

Convolutional Neural Networks:

Convolutional Neural Networks (CNNs) belong to subclass of feedforward neural networks that are specifically optimized for processing data that have a grid-like topology, such as images. As with the fully connected networks, they are also made up of multiple neurons which consist of learnable parameters. They receive as input raw data which are usually image pixels and produce the prediction scores as output. Being designed based on the assumption that they are provided explicitly with images as inputs, CNNs can extract spatial features and identify crucial structural patterns of their input. As a consequence of effectively encoding important image properties, they can be more efficiently implemented to perform the forward pass while requiring highly reduced numbers of parameters, compared to typical neural networks.

Typically a CNN's architecture consists of multiple instances of convolutional, pooling and possibly batch normalization layers, followed by a number of fully connected layers. In Figure 3.2.3 we provide an example of a CNN architecture that is build up from the aforementioned layers.

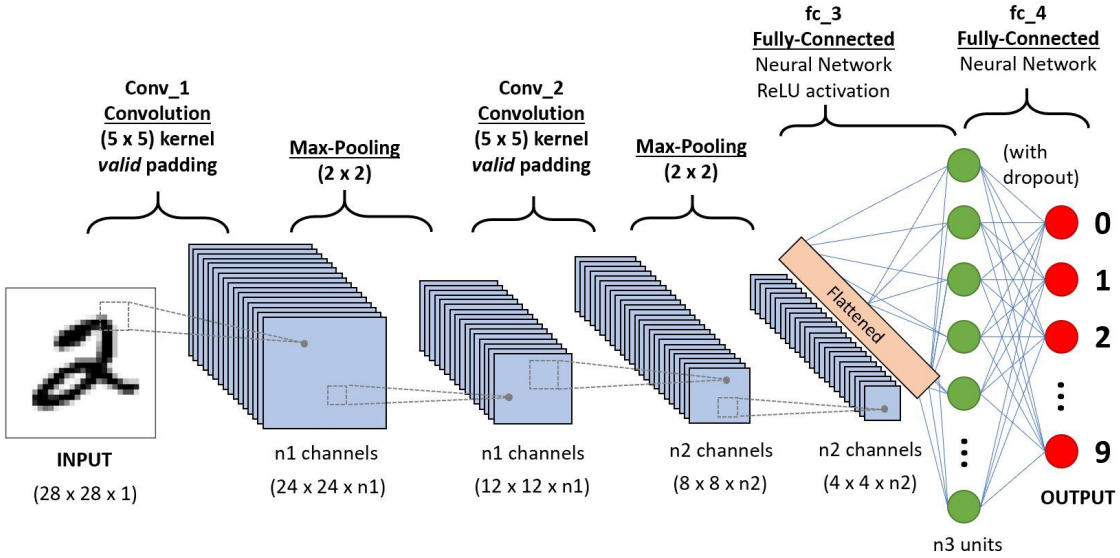


Figure 3.2.3: An illustration of the architecture of a CNN model. From [80].

The functionality of each CNN layer is explained as follows:

Convolutional Layer: Convolutional layers (or conv layers in short) are the fundamental building blocks of CNNs that handle the feature extraction task and perform most of the network's required computations. They receive as input 3D volumes and transform them to output 3D volumes. For the case of the first convolutional layer, the input volume is usually an image of dimensions $w \times h \times d$, where w denotes the width, h the height and d the color channel depth of the image.

The conv layer processes the input volume using a set of K filters with learnable parameters. Those filters have small spatial dimensions (common dimensions being 3×3 or 5×5) but cover the entire depth of the input volume. Convolving each filter across the input volume's width and height and computing the dot products between the filters' weights the corresponding input values, results to a 2D output called an activation map. For the 2D case an illustration of this process is provided in Figure 3.2.4. Stacking the different activation maps generated by the K filters will produce the final output 3D volume of depth K.

The spatial dimensions of this volume depend on two additional parameters, the step size with which each filter is slided across the spatial dimensions input volume, called the stride and the amount of zero padding P on input's borders. Therefore, given an input volume of dimensions $W_{in} \times H_{in} \times D_{in}$, K filters of size $F_w \times F_h \times D_{in}$, stride S and P amount of zero padding, the output width will be $W_{out} = (W_{in} - F_w + 2P)/S + 1$ and the output height $H_{out} = (H_{in} - F_h + 2P)/S + 1$ while the output depth is $D_{out} = K$, as previously mentioned.

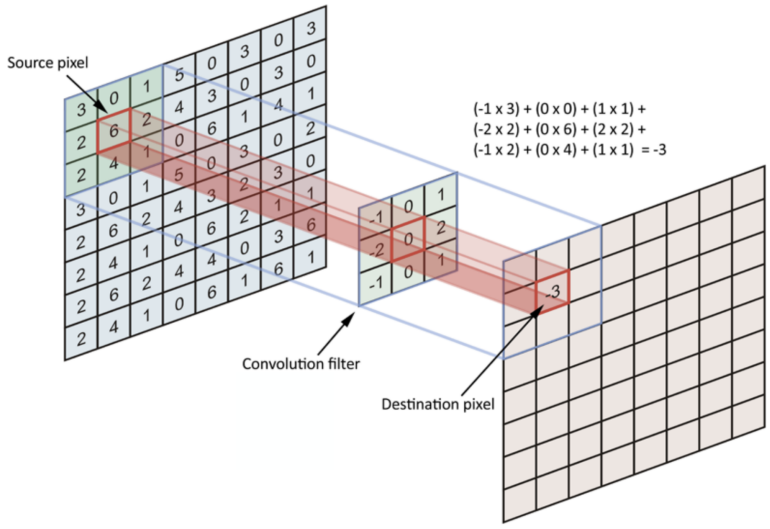


Figure 3.2.4: An illustration of a convolutional operation for the case of 2D input. From [15].

Pooling Layer: The pooling layers, which are regularly inserted between the conv layers of the model, are responsible for reducing the spatial size of the activation maps and thus limit the total number of learnable parameters of the network. This results to making the architectures more efficient, while at the same time it protects from overfitting to the training set data. Moreover, by performing this downsampling operation on the input volume, this layers tend to introduce the property of translation invariance, up to some extent. The pooling layer performs this process by sliding a 2D filter spatially on each activation map, producing an output volume of the same depth as the input volume but of reduced width and height.

The pooling type is characterized based on the type of operation the filter performs, with the two types of pooling mainly used in practice being the following:

- **Average pooling:** Tis pooling type uses an average filter is used to perform the spatial downsampling operation. As a result, it has a smoothing effect to the resulting activation maps of the output volume.

- **Max pooling:** The max pooling type uses a max filter which returns the maximum value being under the patch of the 2D activation map it covers. This results to most prominent features being preserved.

An example of the two types of pooling on a 2D array is shown in Figure 3.2.5 bellow.

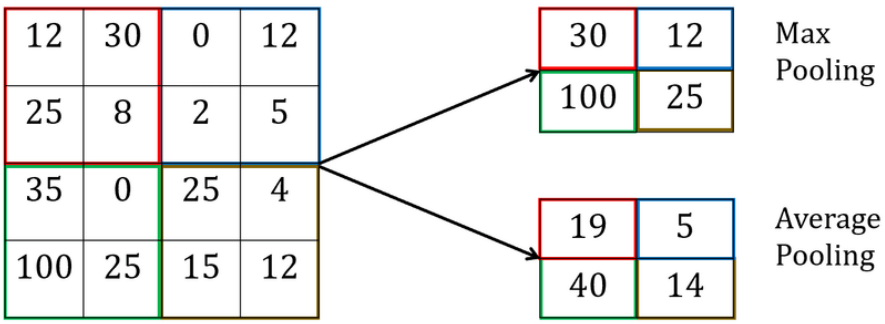


Figure 3.2.5: An illustration of the pooling operation. From [1].

Batch Normalization Layer: The Batch Normalization (BN) layer [40] has the role of normalizing its input by performing a technique known as Batch Normalization. This procedure is used to make the training process faster (by allowing the use of a higher learning rate) and more stable. More specifically, it applies a transformation to the input which keeps the output mean and standard deviation close to 0 and 1, respectively.

Assuming input values $\boldsymbol{x}$ of a mini-batch (i.e. a fixed number of training samples, with its use in training explained in the next section), during training the BN layer generates its output $\boldsymbol{y}$ based on the following equations:

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{3.2.5}$$

$$\sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2 \tag{3.2.6}$$

$$\hat{x}_i = \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}} \tag{3.2.7}$$

$$y_i = \gamma \cdot \hat{x}_i + \beta \tag{3.2.8}$$

The parameters β and γ of the last equation are learned during the training and perform a shift and a scale operation to the normalized values x_i, respectively. This way, the outputs y_i are not restricted to being strictly of zero mean and unit standard deviation. Therefore, the expressive power of the network is not affected, while at the same time the first and second order statistics of y_i are decoupled from interactions with previous layers and depend only on those two learnable parameters.

During inference, since the model needs to work with individual samples as inputs, the calculation of new input statistics is usually not feasible. Instead, the BN layer normalizes its output using a moving average of the statistics calculated from the mini-batches it received during the training.

Fully Connected Layer: The last layers of the CNN architecture usually consist of FC layer that are responsible for predicting the final outputs of the network. For example, regarding the case of image classification, those outputs are the probabilities of each class. As the fully connected layers receive 1D vectors as inputs, the activation maps of the last output 3D volume of the CNN have to be converted into an 1D vector by a process known as flattening, illustrated in Figure 3.2.6.

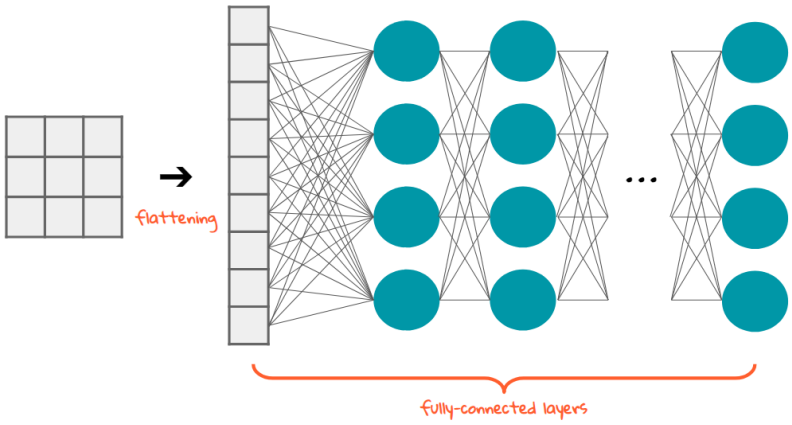


Figure 3.2.6: Example of flattening a 2D activation map that is feed as input to a series of FC layers. From [45].

3.2.2 Deep Learning Training

In the previous section we analyzed two important types of deep learning models, in particular the fully connected and the convolutional neural networks. As mentioned, both network architectures (and DNNs in general) consist of numerous learnable parameters. Those parameters, which, depending on the network's type and the intended application, can range from a few thousands to hundreds of million, have to be appropriately adjusted during the training process in order for the network's predictions to training inputs to begin matching the corresponding input labels. The similarity of the predictions to the labels is measured by a loss function (such as the ones described in Section 3.1.2). With the aim of minimizing this loss measurement, an optimizer modifies the parameters' values iteratively through the training, using a gradient decent based process that relies its efficiency on a technique called backpropagation [77]. In the following paragraphs, after briefly explaining important concept of backpropagation, we will cover the fundamental gradient decent optimization algorithm and two of its most popular extensions, the Stochastic Gradient Decent or SGD and the Adaptive Moment Estimation or Adam.

Backpropagation

The various gradient decent based optimization algorithms require the calculation of the gradient of the loss function with respect to the network's parameters at every training iteration. Due to neural networks having large numbers of parameters arranged at multiple layers, this procedure needs to be implemented in an efficient manner. In practice, this is performed using the famous backpropagation algorithm, which is considered to have two execution stages, the Forward Pass (also known as Forward Propagation) and the Backward Pass (also known as Backward Propagation), as briefly explained bellow:

- **Forward Pass:** It refers to feeding the neural network with some input sample vector and performing all the necessary calculations (and storing their results) in a successive way, starting from the input layer, computing the hidden layers' output and finally the output of the last layer. For all the calculations the current weight values are used.
- **Backward Pass** At this stage, the direction of the data flow is the opposite of the forward pass. In particular, after calculating the error at the output layer, this error is distributed within the network in reverse order, until it reaches the input layer. During the error back-propagation, the partial derivatives of the loss function with respect to the several model's parameters are calculated. This is done in an efficient way that avoids redundant computations and is based on the application of a mathematical formula, used for finding the derivatives of composite functions, known as the chain rule [64].

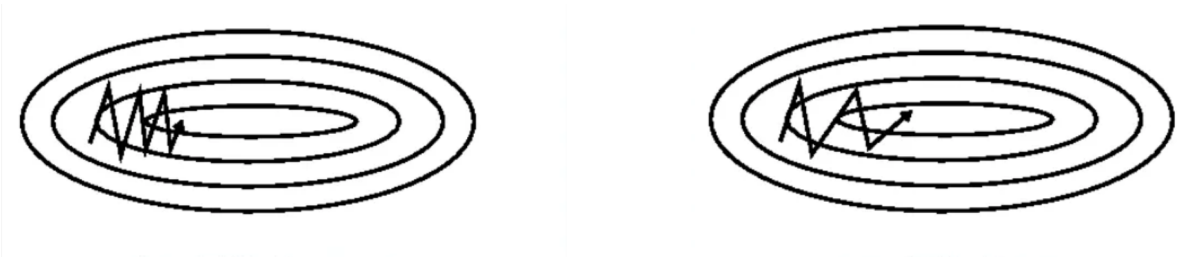
Gradient Decent

Gradient decent is an iterative optimization algorithm that is used for finding a local minimum of a differentiable function. Given a differentiable function $f(\boldsymbol{x})$ to be minimized, at each optimization step it relies on moving $\boldsymbol{x}$ towards the direction of the negative gradient of the function f at point $\boldsymbol{x}$ (i.e. the direction of the steepest decent), defined as $-\nabla f(\boldsymbol{x})$. In the concept of neural network training, the parameter update equation from step k to $k+1$ can be written as:

$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k - \eta \nabla L(\boldsymbol{w}_k), \tag{3.2.9}$$

where $\boldsymbol{w}$ are the vectorized parameters, $L(\boldsymbol{w})$ the loss function to be minimized and η a hyperparameter called step size or learning rate that determines the rate of the learning process. Regarding the learning rate, there is a trade-off between the speed of convergence and the stability of the training process. Setting the learning rate to a very small value can substantially prolong the optimization or lead to the model getting stuck to a suboptimal local minimum (i.e. one far from the global minimum). On the other hand, setting a too high learning rate may result to non convergence due to overshooting the local minima.

The above formulation of gradient decent, also known as the vanilla gradient decent, computes the gradient used to update the weights by averaging the gradients of the loss derived from each sample of the training set. As deep learning models usually have thousands or millions of parameters and are trained on very large datasets, it is to be expected that using the entire dataset for a single update step is computationally prohibiting. This is the main reason that variants of the vanilla gradient decent are used in practice, such as its stochastic version described next.



(a) SGD without momentum

(b) [SGD with momentum

Figure 3.2.7: An example illustration of convergence process using the SGD with and without momentum. From [76].

Stochastic Gradient Decent

Gradient decent in its stochastic form, known as Stochastic Gradient Decent or SGD in short, instead of using the full amount of training data to compute the gradient of the loss, it relies on sampling individual samples $\{\boldsymbol{x}_i, y_i\}$ from the dataset, which are then used in the following update formula:

$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k - \eta \nabla L(\boldsymbol{w}_k; \boldsymbol{x}_i, y_i) \tag{3.2.10}$$

Therefore, as the above equation suggests, the parameters are updated at every iteration based on an estimate of the gradient of the loss, calculated using single sampled training sample.

As described above, the SGD algorithm is much less computationally demanding than the vanilla one. Despite that, using a single sample for the gradient estimation tends to result to large fluctuations of the loss function (e.g. if one sample is not descriptive enough of the distribution of the dataset) and thus can negatively influence the convergence of the model.

Mini-Batch SGD: A popular variant of the SGD that aims at mitigating both computational and convergence issues is the Mini-Batch SGD that uses a small subset of datasets samples for estimating the gradient, known as the mini-batch (often also referred simply as batch). In practice, in the context of training deep neural networks, the use of an SGD optimizer refers to this particular version, with the term mini-batch omitted by convention.

SGD with Momentum: An extension of the SGD, known as SGD with momentum, is widely used in order to improve the convergence speed and avoid undesirable local minima, especially in the presence of high curvature or noisy gradients. More specifically, the momentum SGD optimizer uses the following parameter update rule:

$$\begin{aligned} \boldsymbol{v}_k &= \beta \boldsymbol{v}_{k-1} + \eta \nabla L(\boldsymbol{w}_k; \boldsymbol{x}_i, y_i) \\ \boldsymbol{w}_{k+1} &= \boldsymbol{w}_k - \boldsymbol{v}_k \end{aligned}$$

In the above equations, the term $\boldsymbol{v}_k$, that is used to update the current parameter vector $\boldsymbol{w}_k$, consists of the loss term $\eta \nabla L(\boldsymbol{w}_k; \boldsymbol{x}_i, y_i)$ also used in the original SGD algorithm combined with the previous update term $\boldsymbol{v}_{k-1}$, weighted by a hyperparameter $\beta \in (0, 1)$. This formulation, which can be considered as including an exponentially decaying moving average of the past gradients the the current update term, creates a momentum-like effect that helps at increasing the convergence stability and speed, as depicted in Figure 3.2.7.

Adaptive Moment Estimation (Adam)

Adaptive Moment Estimation or Adam [46] is an extension to the SGD optimization algorithm that is the optimizer of choice for many deep learning training scenarios. It combines the advantages of two SGD based algorithms, the Adaptive Gradient Algorithm (AdaGrad) and the Root Mean Square Propagation (RMSProp). Specifically, Adam calculates exponential moving averages of the gradient and the gradient

squared controlling the decay rates using two hyperparameters, β_1 and β_2, as formulated by the equations below:

$$\boldsymbol{m_k} = \beta_1 \boldsymbol{m}_{k-1} + (1-\beta_1)\nabla L(\boldsymbol{w}_k) \tag{3.2.11}$$

$$\boldsymbol{s}_k = \beta_2 \boldsymbol{s}_{k-1} + (1-\beta_2)\left(\nabla L(\boldsymbol{w}_k)\right)^2, \tag{3.2.12}$$

where vectors $\boldsymbol{m_k}$ and $\boldsymbol{s}_k$ correspond estimates of the gradient's first and second moments (mean and variance) respectively, while the hyperparameters $\beta_1, \beta_2 \in (0,1)$ take default values in popular deep learning frameworks $\beta_1 = 0.9$ and $\beta_2 = 0.999$. This vectors are initialized as $\mathbf{0}$, resulting to zero-biased moment estimates. Due to this reason, their bias-corrected versions have been proposed:

$$\hat{\boldsymbol{m}}_k = \frac{\boldsymbol{m}_k}{1-\beta_1^k} \tag{3.2.13}$$

$$\hat{\boldsymbol{s}}_k = \frac{\boldsymbol{s}_k}{1-\beta_2^k} \tag{3.2.14}$$

Using the above bias-corrected estimates, the final parameter update equation can be written as:

$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k - \frac{\eta}{\sqrt{\hat{\boldsymbol{s}}_k} + \epsilon}\hat{\boldsymbol{m}}_k, \tag{3.2.15}$$

where η is the learning rate hyperparameter, as usual.

Chapter 4

Compression of Deep Neural Networks

4.1 Introduction

In recent years, deep neural networks have become the state-of-the-art (SoA) approach at tackling many complex machine learning problems, in fields such as computer vision, natural language processing, speech and audio processing and robotics [33, 33, 48]. While the application of DNNs to problems of the aforementioned fields can result to dramatic performance improvements compared to using traditional machine learning methods, this comes at the cost of significant increases in computational complexity. This is attributed to the tendency of DNNs to rely upon having very large sets of learnable parameters, often numbering in tens or hundreds of millions, and therefore demand huge amounts of memory and computational resources during training and inference [14]. Such large resource requirements hinder the adaptation of the SoA deep learning based systems to resource-constrained devices, such as mobile phones, wearables, intelligent robots and many other smart portable devices that rely on embedded computing systems with limited processing, memory and power resources.

Recognizing the large potential of deploying deep learning models to compact devices, there has been increased research interest over the recent years in compressing and accelerating DNNs using various techniques [14, 57, 26]. By using compressed DNN models, a number of important advantages can be achieved, leading towards enabling the use of DNNs to compact environments, some of the most prominent being the reduced storage demands for the various model parameters, lower FLOPs requirements during inference and energy consumption savings. One very popular compression technique which has been studied extensively over the recent years is *network pruning* [5], which is the process of removing network parameters based on some criterion, with the aim of reducing the network's size (*i.e.* its total number of parameters) while keeping the performance loss as low as possible. This approach, being the focus of this thesis, will be explained in detail in Section 4.3 , where the various categories of pruning will be analyzed. Before covering the pruning methodologies, in Section 4.2 a number of different popular compression methods will be briefly introduced, those being quantization [57], tensor decomposition [43, 68], knowledge distillation [26] and compact model design [14], so that the reader can have a broader understanding of the various compression approaches.

4.2 Related Compression Approaches

4.2.1 Quantization

Quantization is the process of representing the values of a continuous signal using a small set of discrete symbols or integer values. In DNNs, the quantized elements usually correspond to weights and biases, activations or gradient values. Quantization techniques for compressing DNNs can be broadly divided into two categories, numerical low-bit quantization [89, 81] and partial quantization techniques that are mainly based on weight sharing via clustering [28].

Numerical Low-bit Quantization

This is the most straightforward type of DNN quantization, where the network's parameter values are represented using lower-precision arithmetic. More specifically, while DNN values are typically stored using 32-bit or 16-bit floating point (FP) numbers, the values of quantized DNNs are represented using integer precision arithmetic, such as INT-8, INT-4, INT-2 or even INT-1, being a special case of quantization that produces models known as binarized neural networks [37]. In practice, one can obtain a network with low-precision parameters in two ways:

1. Post-training quantization
2. Quantization-aware training

Post-training quantization [81] refers to quantizing a neural network after it has been trained using standard floating point arithmetic. By down-scaling parameter values from floating point to low integer values (usually INT-8 or INT-4), the network's storage requirements can be reduced, while at the same time it can benefit from inference time acceleration, with the exact speedup being dependent on the hardware and inference optimizer used. For example, using the NVIDIA TensorRT model inference framework [71], $2-4\times$ acceleration can be achieved (depending on the batch size used) using INT-8 quantized DNNs over FP-32 ones. As expected, the aforementioned memory and inference speed advantages come at the cost of accuracy drops

over the original, unquantized network, due to the approximation errors introduced by the quantization process, which can become substantial if very low precision is used, e.g. INT-2 or INT-1 [57].

Quantization-aware training (QAT) [42] is aimed at simulating the quantization effects during training and therefore compensating the various approximation errors introduced. This is achieved by using the quantized version of the network parameters during the forward pass, while treating the quantizing operation as the identity function during the backward pass, thus avoiding the zeroing of the gradients that backpropagating through the quantizing function would have otherwise caused [12]. The gradient values found are then used to update the unquantized versions of the parameters, which are retained during the entire training process. This process is known as training using the Straight-Through Estimator (STE), originally introduced by Hinton *et al.* [3]. The QAT process is illustrated via an example in Figure 4.2.1. After training, the weights are replaced with their low-precision versions, allowing faster inference and memory saving, at reduced accuracy drops compared to post-training quantization [57].

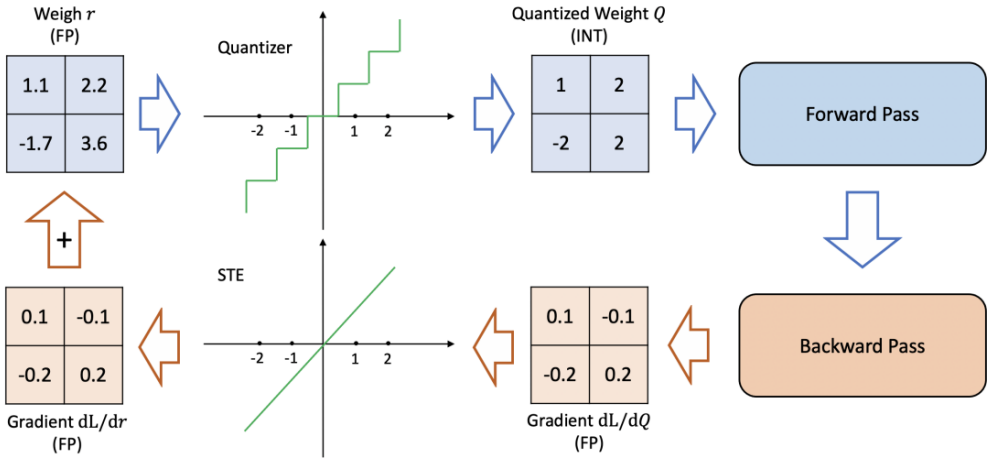


Figure 4.2.1: An illustration of the quantization-aware training process (forward and backward pass) using an example array of 4 weights. From [23].

Weight Sharing

As an alternative DNN quantization approach, *weight sharing* [28, 10] can be considered. Weight sharing refers to reducing the number of unique weights values that appear within the network. This is usually achieved using some clustering algorithm such as k-means to identify the weight clusters, making weights that belong to the same cluster to share a common weight value, the cluster's centroid. Although the storage demands of the model compressed using weight sharing can be reduced (since for each weight only a small index to a centroid needs to be stored), the inference speed remains mostly unaffected, as during inference the weights are assigned their shared values (based on some lookup table) which are typically still represented by the original FP precision.

As an example implementation of weight sharing, Han *et al.* [28] use the scheme illustrated in Figure 4.2.2. Assuming a model with 4 input and output neurons (16 weights in total), the weights are made to share 4 centroids, which are found via k-means clustering. In this particular weight sharing implementation, the centroids are further fine-tuned by grouping and then summing the gradients based on the clustering of the weights and using the produced values to update the corresponding centroids (as illustrated by the bottom part of Figure 4.2.2).

4.2.2 Tensor Decomposition

Applying tensor (including matrix) decomposition techniques on weight tensors has been extensively studied as a way for compressing and accelerating neural network models. Such methods decompose the multi-dimensional weight tensors into low rank approximations, aiming at removing redundant parameters and

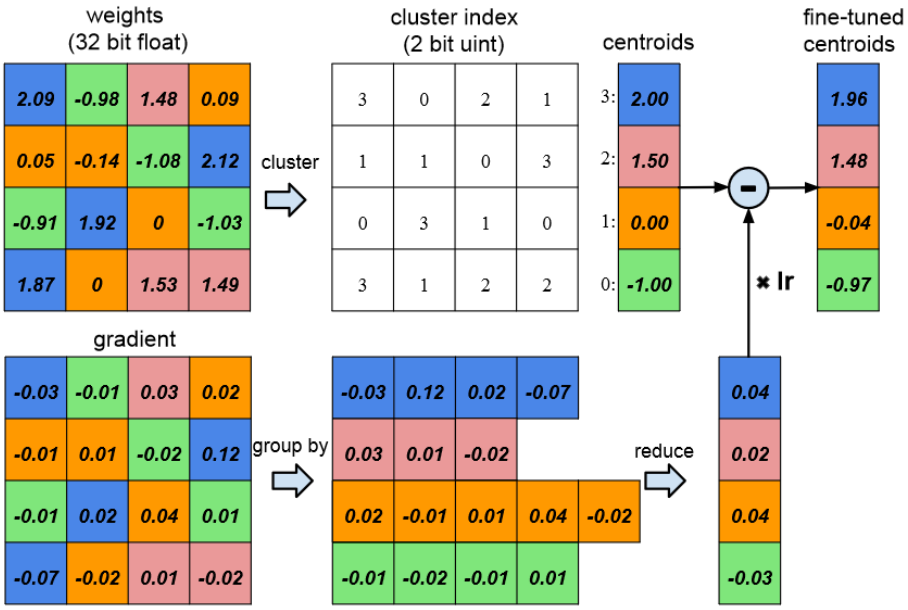


Figure 4.2.2: Illustration of a weight sharing scheme. From [28].

saving computational time. Multiple decomposition techniques and algorithms have been utilized for this task, with some of the most popular among related literature being the Singular Value Decomposition (SVD) [62, 68], Canonical Polyadic (CP) decomposition [52, 83] or dictionary learning [74].

For instance, Rigamonti *et al.* [74] introduced the idea of separable 1D filter learning for reducing the computation in CNNs. Their approach was based on dictionary learning. A subsequent work by Jaderberg *et al.* [43] proposed to approximate CNN filters using a low-rank basis of spatially separable filters, as demonstrated in Figure 4.2.3. Their approach showed up to 4.5× speedup with 1% accuracy drop in standard text recognition benchmarks. A more recent work by Yu *et al.* [93] suggested that due to the tendency of weight filters to be both low rank and sparse, it is beneficial to combine sparse and low rank decompositions to the compression process. They reported high compression rates for popular CNN architectures such as AlexNet and VGG-16.

As a final note, although applying tensor decomposition techniques to DNNs' weights is a straightforward way for achieving compression, implementing the decomposition process can be computationally expensive. Furthermore, in most methods the decompositions are performed layer by layer. To compensate for the inevitable accuracy losses, after decomposing a layer's parameters, the following layers need to be fine-tuned. This procedure can result to large training overheads, especially when considering applying the methods to very large models.

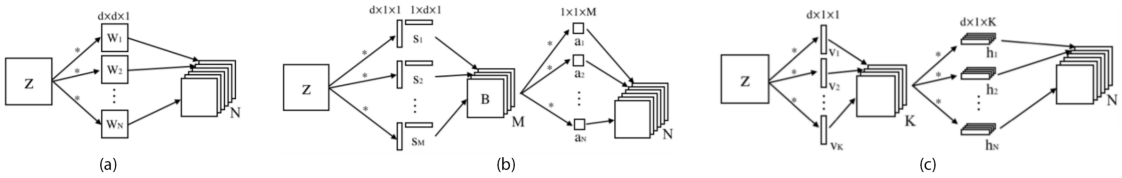


Figure 4.2.3: (a) typical CNN layer acting on single channel input. (b) and (c) approximating the layer's filters based on the two proposed schemes by Jaderberg *et al.* From [43].

4.2.3 Knowledge Distillation

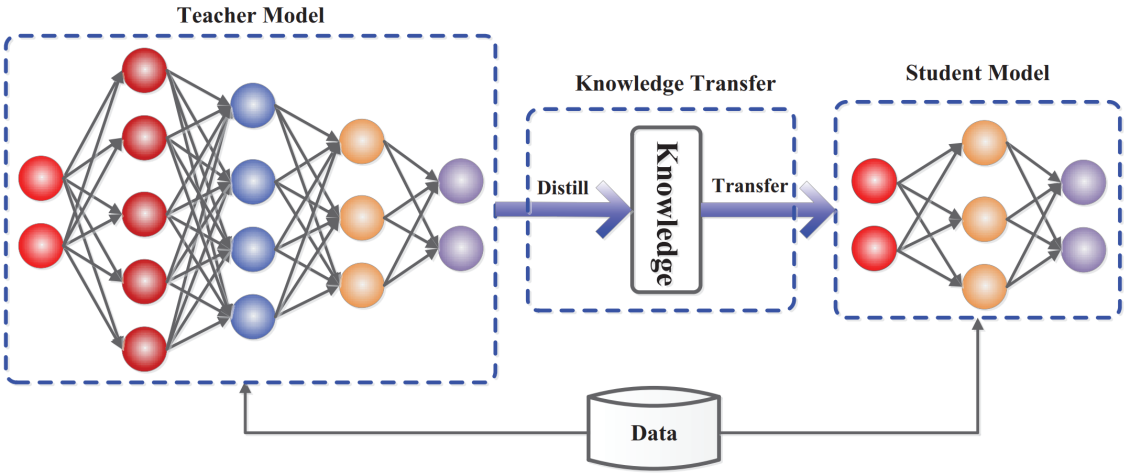


Figure 4.2.4: An illustration of the teacher-student model approach for knowledge distillation. From [26].

A model compression approach that involves teaching a small model, called the student model, to perform perform a task (*e.g.* image classification) based on the responses (or knowledge) of a larger pre-trained model (or an ensemble of model), called the teacher model, is known as *Knowledge Distillation* (KD). The general KD scheme is depicted graphically in Figure 4.2.4. The idea was pioneered by Buciluǎ *et al.* [8] and further generalized by the work of Hinton *et al.* [32], where a KD framework was introduced, producing compact models outperforming the ones training from scratch (without KD).

More specifically, in [32] a student model is trained on some transfer set using a loss function featuring two terms, the student loss and the distillation loss terms. The student loss is standard cross entropy loss between the labels of the data samples $\boldsymbol{y}$ and the corresponding student's predictions (class probabilities). The distillation loss is an additional cross entropy loss term between the soften versions of the student's and teacher's predictions. By defining the logits (*i.e.* the outputs of the layer prior to the softmax) of the student and teacher network for class i as z_i and v_i, respectively, the soften version of the predicted class probabilities of the student and teacher networks are:

$$q_i = \sigma(\boldsymbol{z}; T = \tau) = \frac{\exp\left(\frac{z_i}{\tau}\right)}{\sum_j \exp\left(\frac{z_i}{\tau}\right)}, \quad p_i = \sigma(\boldsymbol{v}; T = \tau) = \frac{\exp\left(\frac{v_i}{\tau}\right)}{\sum_j \exp\left(\frac{v_i}{\tau}\right)}, \tag{4.2.1}$$

where T is a temperature parameter for the softmax function σ. As this temperature parameter T becomes larger, the prediction probabilities that the softmax function outputs become softer, thus can potentially provide more knowledge to the student model, as opposed to having a distribution where the most likely class has very high probability while the rest close to zero.

Based on the above, the loss function for the knowledge distillation compression method is written as:

$$L_{KD} = \alpha_1 \cdot L_{CE}\left(\boldsymbol{y}, \sigma(\boldsymbol{z}; T = 1)\right) + \alpha_2 \cdot L_{CE}\left(\sigma(\boldsymbol{z}; T = \tau), \sigma(\boldsymbol{v}; T = \tau)\right) \tag{4.2.2}$$

where L_{CE} is the cross entropy loss function and α_1, α_2 are two hyperparameters that control the contribution of each loss term to the overall loss function. This framework for knowledge distillation is illustrated in Figure 4.2.5.

In the knowledge distillation formulation described above, the knowledge is dubbed as *response-based knowledge*, since it comes from the teacher model's output layer's predictions. Other types of knowledge that can be extracted from the teacher network for the purpose of assisting the training of a student model include *feature-based knowledge* and *relation-based knowledge*, where the knowledge is distiled from intermediate layers and between the feature maps, respectively [26].

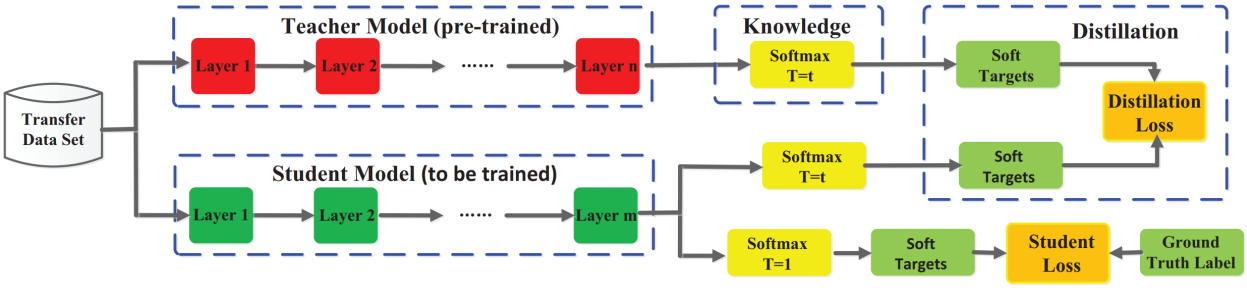


Figure 4.2.5: The knowledge distillation framework by Hinton *et al.* From [32].

4.2.4 Compact Model Design

Although not a model compression technique in the strict sense, designing compact models from scratch that achieve acceptable accuracy is a straightforward way to further enable the use of deep learning to mobile and resource constrained applications. Widely used compact model architectures include MobileNet [35], SqueezeNet [38] and DenseNet [36] and are briefly explained bellow.

MobileNet

Proposed by Howard *et al.* MobileNets are a family of efficient CNN models targeted for resource limited computer vision applications. Those networks are based on using depthwise separable convolutions, a factorized version of the standard convolution operation. More specifically, a depth-wise separable convolution consists of a depthwise convolution followed by a 1×1 pointwise convolution. During the depthwise convolution, a single filter is applied to each input channel. The outputs of depthwise convolution are then combined using the pointwise convolutions. Furthermore, the trade off between accuracy and latency can be controlled by two global hyperparameters. The particular architecture, compared to much larger networks achieved similar accuracy on the ImageNet dataset [13].

SqueezeNet

SqueezeNet is a small CNN architecture designed by Iandola *et al.* [38] that was able to reach accuracy scores on ImageNet similar to the $50\times$ larger in parameters AlexNet [48]. To achieve a compact yet well performing model, three main strategies are employed. Many 3×3 filters are replaced by 1×1 ones, calling layers comprising only of 1×1 *squeeze layers*. Also, the number of input channels to 3×3 is decreased in order to keep the total parameters to a minimum. This is achieved by placing squeeze layers having relatively few filters before the *expand layers* that have a mixture of 1×1 and 3×3 filters. The combination of the squeeze and expand layers was dubbed the *fire module* which is the building block of the SqueezeNet architecture. Finally, to maximize accuracy given a constrained parameter budget, downsampling is performed late in the architecture in order to retain large activation maps.

DenseNet

DenseNets are a class of CNN models proposed by Huang *et al.* [36] that obtained high performance of popular object recognition databases, although having less parameters and computation than many standard CNN models. The main idea is to divide the network into multiple *dense blocks* where within each block each layer takes all the previous feature maps as input, resulting in $\frac{L(L+1)}{2}$ direct connections for a block with L layers. In between dense blocks *transition layers* are utilized to change the feature map sizes using convolution and pooling. Due to their design using dense connections, DenseNets can address the vanishing gradient problem while the reuse of feature maps allows them to work well with low numbers of total parameters. Despite that, it needs to be noted that DenseNets, due to the particular way that are designed, can be more memory and compute intensive to run that other compact DNNs.

4.3 Pruning - Sparse Neural Networks

4.3.1 Introduction

Pruning is one of the most popular techniques for drastically reducing the size and accelerating the inference of deep learning models. It is based on the realization that DNNs tend to be heavily over-parameterized and thus unimportant network parameters can be removed, based on some ranking criterion, with little to no effect on the model's performance. The method dates back to 1990 and before, with pioneering works such those of LeCun *et al.* [54] and Hassibi and Stork [30] and has gained much research attention in the recent years as a way of combating the continuously growing size of SoA neural networks [14]. The pruning methodology, which has evolved over the years of its study, can be broadly categorized in terms of the granularity of the network's elements being sparsified, the pruning criterion being used to rank the importance of the elements candidate for pruning and based on the timeframe the sparsity introduced to the model. The aforementioned categories of pruning are analyzed in detail in the following sections, highlighting the various popular methods and important considerations for each scheme.

4.3.2 Pruning Criteria

Magnitude-based pruning

A very straightforward and widely accepted heuristic criterion to determine the importance of network weights is based on their magnitude values [57]. Following this criterion, weights with small magnitudes are considered less important to the network's output than the ones with large magnitudes. According to this reasoning, for the case where the elements sparsified are individual weights (unstructured pruning), weights that lie below some magnitude threshold value T are removed from the network, while the others retain a non-zero value. This operation is known as thresholding and assuming a threshold operator $P_T(x)$ and an network weight w, it is written as:

$$\tilde{w} = P_T(w) = \begin{cases} f(w), & \text{if } |w| > T \\ 0, & \text{otherwise} \end{cases} \tag{4.3.1}$$

The instantiation of the function $f(w)$ defines the type of threshold operator used. The most common operators in the magnitude pruning literature are the Hard and the Soft [18] threshold operators, where for each case f is defined as:

$$f_{\text{hard}}(w) = w, \quad f_{\text{soft}}(w) = \begin{cases} w - T, & \text{if } w > T \\ w + T, & \text{if } w < -T \end{cases} \tag{4.3.2}$$

Pruning of groups of weights (including entire channels or filters) can also be based on the magnitude criterion. In this case, the magnitudes of the individual weights of each group are usually aggregated using some norm (such as L_1 or L_2) and the groups with the smallest resulting norms are removed. As an example, Hao Li *et al.* [56] prune CNN filters with the smallest L_1 norm values, achieving reduced inference costs for common architectures on CIFAR-10 and using retraining to regain accuracies near those of the unpruned networks.

Hessian-based pruning

The use of second order derivatives (Hessian matrix) of the objective function (w.r.t. the network parameters) to determine the importance of each parameter has been explored extensively, dating as far back as 1990.

The seminal work of LeCun *et al.* [54] following this approach, proposes a method called Optimal Brain Damage (OBD) for pruning parameters from a trained network, one at a time. OBD introduced some simplifying approximations. More specifically, it was assumed that (i) the objective function is nearly quadratic, (ii) the parameters are pruned after the convergence of the network and (iii) the change δL to the objective function L caused by pruning multiple parameters is equal to the sum of the δL's, each attributed to pruning a parameter individually (diagonal assumption for the Hessian). Using OBD, the authors pruned a 2600-parameter network used for handwritten digit recognition by a factor of four, reporting a slight increase in accuracy.

An extension to the OBD, Optimal Brain Surgeon (OBS) [30], utilized a similar Hessian-based pruning approach but preserved the off diagonal values of the Hessian matrix, arguing that the Hessian is non-diagonal for most applications used. The OBS method, which requires the calculation of the inverse Hessian, relies on a recursion relation to calculate it from the training data and structural information of the network. It is reported to improve the OBD method, achieving up to 90% parameter reduction for XOR networks.

Although the aforementioned Hessian-based pruning methods were proposed at that time as a more precise pruning approach than magnitude pruning, they were applied to early neural network architectures that were much shallower and with significantly less parameters than current DNNs. Due to that, calculating the Hessian for most DNNs used today is not feasible, making most recent Hessian-based pruning methods to rely on low-cost approximation techniques [88, 78, 92].

Regularization-based pruning

The two previous pruning criteria aimed at directly evaluating the importance of each candidate element for removal. In contrast, a number of works introduce sparsity zeroing parameters in a more indirect way using regularization techniques. Ideally, the (unstructured) pruning task can be modeled by using L_0 regularization, *i.e.* by introducing a term $R_{L_0}(\boldsymbol{w}) = \sum_i \mathbb{I}[w_i \neq 0]$ to the loss function, explicitly penalizing the number of non-zero parameters. Unfortunately, directly minimizing the loss function with the L_0 regularization term is intractable since this penalty is non-differentiable and allows $2^{|w|}$ discrete states of the vector $\boldsymbol{w}$. Due to the aforementioned issues of the L_0 norm, most works either attempt to reparametrize the model's parameters and enable efficient optimization or utilize the L_1 norm which is considered a good convex approximation of the L_0 norm.

Louizos *et al.* [61] propose a method for efficient L_0 regularization aimed at pruning DNNs during training. They address the non-differentiability of the L_0 norm by reparametrizing the network's weights as the product of a weight and a non-negative stochastic gate variable which determines if the weight should be set to zero. They propose the hard-concrete distribution as a suitable sampling distribution for the gates, having parameters that can be optimized together with the model's parameters during training. Using this formulation, the value of the CDF of the stochastic gate evaluated at zero can be used to find the expected L_0 norm.

Regularization-based approaches have also been adopted for structured pruning. For instance, the authors of [51] propose the use of a group sparsity regularizer on the CNN filters in order to prune the elements of their kernels in a group-wise manner. The use of structured sparsity regularization was also employed in [90] to penalize various structures of DNNs such as filters or channels.

4.3.3 Granularity of Sparsified Elements

Based on granularity of the sparsified elements the various pruning approaches can be broadly divided into two categories, unstructured or structured. Unstructured methods leave the pruning algorithm free to prune parameters regardless where they belong within the model. On the other hand, structured pruning methods remove groups of parameters and thus can be further characterized by the type of groups being pruned (e.g. vectors, kernels or entire filters). Examples of the different types of sparsity patterns obtained by unstructured or structured pruning are illustrated in Figure 4.3.1. Bellow we will discuss in more detail the two types sparsity, highlighting the trade-offs of each approach.

Unstructured pruning

In unstructured pruning, the least important weights are removed according to the pruning criterion, irrespectively of where they are within the network. This pruning procedure results to irregular distributions of the non-zero parameters in the DNN's tensors. Due to the unconstrained nature of the induced sparsity, the unstructured pruning algorithms typically achieve superior performance compared to structured ones, obtaining better accuracies at the same sparsity levels (acting as a form of upper bound regarding the best obtainable accuracies) and in addition being able to sparsify networks to extreme extents while still retaining acceptable levels of performance [21]. On the negative side, the irregular type of sparsity patterns means that it is typically less suitable to be utilized by commodity hardware for computational gains. Despite

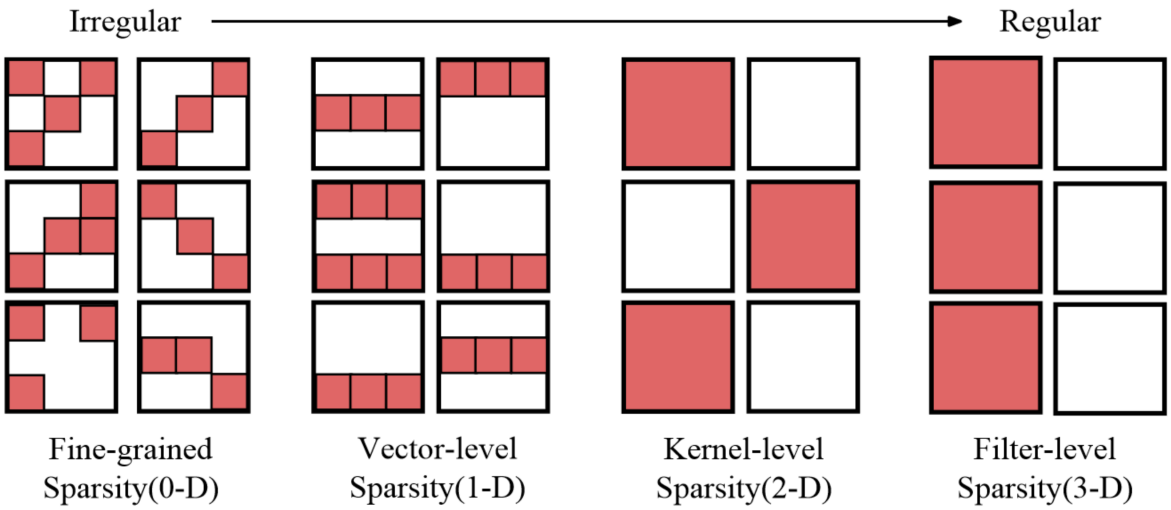


Figure 4.3.1: Comparison of the sparsity patterns induced by unstructured (or fine-grained) sparsity (left first image) and various types of structured sparsity, on a set of two CNN filters each having three 3×3 kernels. From [63].

that, extensive research efforts are being devoted to improve the hardware and software support for unstructured sparsity, with recent solutions such as the libraries NVIDIA cuSPARSE [70] and Sputnik [22] or the NVIDIA Ampere Architecture [69] that supports fine-grained 2:4 sparsity patterns, a sparsity type in between structured and unstructured that suitable for inference acceleration while retaining the dense performance [96].

Structured pruning

Structured pruning methods on the other hand remove entire groups of parameters and thus produce neural networks that can more easily be accelerated by every-day hardware. The types of groups considered vary, with some of the most frequently selected being parameter blocks [16], neurons [59], kernels, filters or channels [51, 90] or N:M structured sparsity [96]. The filter or channel pruning in particular, produces networks that are practically equivalent to smaller dense networks, therefore their acceleration at inference is straightforward and without overheads. The other types of structured sparsity patterns (e.g. parameter block pruning or N:M structured sparsity) tend to favor the final performance (although it is generally still lower than that form unstructured pruning) at the cost of requiring more specialized hardware to utilize the induced sparsity for speedups.

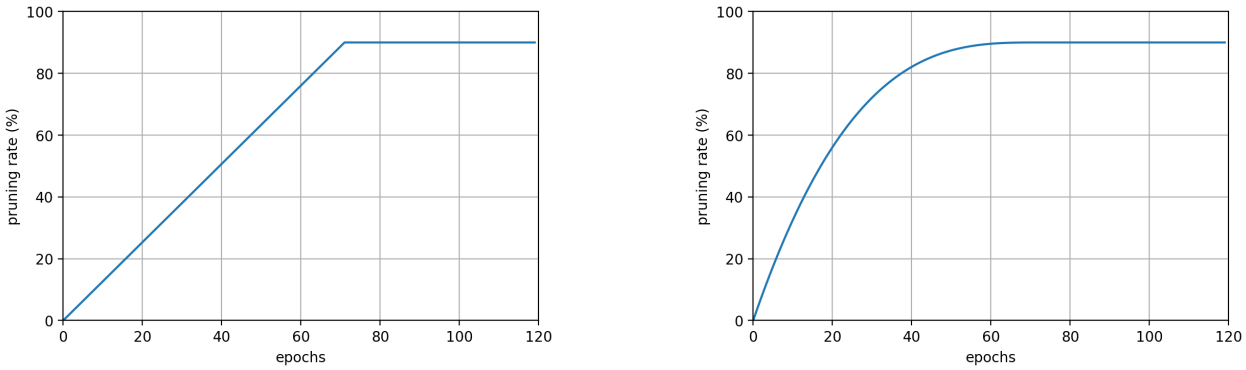
4.3.4 Timeframe of Sparsification

Pruning methodology can be further categorized based on when the removal of weights occurs. Specifically, the sparsification can be performed after the end of the standard training procedure of the dense network *post-training pruning*, along the course of the standard training (*pruning along training*) or before the training has started (*pruning at initialization*). The first two cases are often refereed as dense-to-sparse training, since the a fully dense DNN reaches the desired sparsity level at the end of training, while the last approach, which starts training with an already sparse model, is refereed as sparse to sparse training [50]. Below each case of pruning will be briefly analyzed.

Post-training pruning

Early pruning methods [54, 30] as well as more recent approaches [29, 56] work by pruning an already trained network, typically following a three-stage pipeline [60]:

1. Train a dense model until convergence (or use a pretrained model if available)
2. Prune the trained model based on the selected criterion



(a) Ramp scheduler

(b) Non-linear scheduler

Figure 4.3.2: Examples of pruning schedulers used to gradually reach a 90% pruning rate at epoch 120.

3. Retrain the pruned model to recover the lost accuracy

During the retraining stage, the pruned elements stay inactive and a fine-tuning process is usually followed to train the remaining parameters. Note that the last two steps, pruning and retraining can be repeated for multiple rounds, incrementally increasing the pruning ratio, in order to obtain a better final accuracy [29]. Although this type of pruning procedure is straight forward to implement since the pruning is performed statically at the end of training (and therefore it does not directly interfere with the training procedure), methods of this kind usually require prolonged training times due to the multiple pruning and fine-tuning rounds [60].

Pruning along training

A recent trend in the pruning literature is to perform the pruning process throughout the standard training procedure [98, 50, 97, 86, 84]. Usually this involves pruning the network in a gradual manner, using some pruning scheduler that pre-defines the pruning rate at each training iteration or epoch. This scheduler can be as simple as a ramp function, starting for 0% pruning rate at the first iteration and linearly increasing until it reaches the final pruning rate. The final pruning rate may be reached at the last iteration, although usually a phase with the final pruning rate kept constant for a number of iterations is present, in order to allow the network to better adjust for the desired sparsity ratio before the end of training. More sophisticated schedulers have been proposed, such as one following a cubical increase of sparsity [98] that aims at pruning the network more aggressively at the beginning of the training. The two types of schedulers mentioned are depicted in Figure 4.3.2.

A further consideration regarding pruning along training methods is how to handle backpropagating through the thresholding function that performs the pruning operation. A first approach is to directly back-propagate through it, as done in early methods such as [98], resulting to zeroing the gradients of the pruned weights, thus excluding them from being updated. Unfortunately, this approach is reported to lead to an undesired decay of immaturely trained parameters and a slow exploration of the possible sparsity patterns [44, 84]. A more recent technique is to incorporate the Straight-Through Estimator in a similar way as in quantization-aware training, mentioned in Section 4.2, now considering the thresholding function as the identity during the backpropagation step and updating both pruned and unpruned weights based on the gradients of the loss w.r.t. the sparse set of weights. However, during the forward pass only the sparse weights are used so that the network is trained under the sparsity constrain. By allowing gradient flow to the pruned weights, it becomes possible for them to become active again if they get large enough magnitude. This way the exploration of different sparsity patterns is promoted, something found to be beneficial to the final performance of the sparse networks [44, 84]. Although the vanilla STE as explained above was an important step towards improving pruning along training methods, further improvements can be made by carefully combining the STE with new components (*i.e.* a new thresholding operator and a gradient scaling technique), this being a main contribution of this thesis and is explored in detail in Chapter 6.

Pruning at initialization

A final class of pruning algorithms seeks to use sparsity not only to increase inference speeds but to also benefit from decreased training times by training an already sparse network. A key motivation leading research towards pruning at initialization are the financial and environmental benefits that come along with reducing the training times required (which for many current SoA networks can be substantial) [20]. Popular methods of this kind include SNIP [55], a method that prunes the models before training using a data-dependent way based on a saliency criterion that identifies important connections, GraSP [87], also an pruning at initialization approach based on a pruning criterion aimed at preserving the gradient flow through the network and SynFlow [85], a data-independent pruning at initialization algorithm that targets at avoiding layer-collapse (pruning of an entire layer that results to untrainable networks) by removing weights having the lowest "synaptic strengths". It needs to be noted that many of the recent pruning algorithms of this class have been influenced by the Lottery Ticket Hypothesis of Frankle and Carbin [19], stating that in dense randomly-initialized networks exist subnetworks (winning tickets) that are able to reach accuracy similar to that of the dense network, when trained at isolation given the same training budget.

Chapter 5

Adaptive Magnitude Pruning via Layer-wise Weight Distribution Modeling

5.1 Abstract

Neural Network pruning is an increasingly popular way for producing compact and efficient models, suitable for resource-limited environments, while preserving high performance. Existing methods that can produce acceptable results generally rely on heavy hyperparameter tuning and introduce non-negligible training overheads. To address that, in this chapter of the thesis we propose a method that can achieve robust, out-of-box performance with minimal computational overhead during training. By modeling the distributions of the weights per-layer in a novel way as Gaussian or Laplace it is able to learn the pruning thresholds accurately through the optimization process, resulting in an effective non-uniform sparsity for a requested sparsity target. Our method's effectiveness and adaptability is demonstrated using various architectures on the CIFAR dataset, while on ImageNet it achieves state-of-the-art Top-1 accuracy using the ResNet50 architecture, surpassing more complex and computationally heavy methods.

5.2 Introduction

Recognizing that DNNs are heavily over-parametrized [82], network pruning (surveyed in detail in Section 4.3) has been studied extensively in the recent years as a way of drastically reducing the model's size and computational footprint [11, 14, 57, 34]. Existing methodology while reported to achieve promising results, (a) usually requires extended training times, either in the form of multiple retraining and fine-tuning rounds [72] or by inducing non-negligible computational overhead [84, 97], (b) depends on finding complicated hyperparameter settings in order to result to the desired sized model [50] or (c) lacks adaptability in working with more complicated model architectures, especially at high sparsity ratios, resulting in performance inconsistencies [98].

In this chapter, we propose a very efficient unstructured pruning algorithm, reaching state-of-the-art results while not displaying the aforementioned disadvantages. The proposed algorithm sparsifies the networks' weights through a threshold operation during training (without the need of extra retraining epochs). The main goal of this approach is to effectively "learn" multiple appropriate adaptive thresholds for applying distinct per-layer thresholding operations. Specifically, following the approach of [73], to reach a specified total sparsity ratio S (which we progressively increase throughout the training process) we consider the per-layer thresholds $\{r_l\}$ (based on which magnitude pruning is performed) trainable parameters which are included in an additional loss term, $L_s(\{r_l\}) = \left[S - \hat{S}(\{r_l\})\right]^2$, where $\hat{S}(\{r_l\})$ is an estimate of the model sparsity calculated based on assumptions for the per-layer distributions. To keep $\hat{S}(\{r_l\})$ differentiable, and based on previous observations, we assume that the weights of each layer obtain distributions close to being Gaussian or Laplace, with the specific type being automatically decided per-layer throughout the course of training, in order to minimize the sparsity estimation error. $L_s(\{r_l\})$ is then appropriately added to the typical training loss so that the thresholds are optimized during training, resulting into a learned non-uniform sparsity. All the previous analysis is made to hold while pruning the network by using the Straight-Through Estimator [3] for updating the dense set of the weights. As a final step, to achieve the exact specified level of sparsity we slightly adjust the found thresholds a few epochs before the end of training, now calculating the exact thresholds by sorting the weights per-layer.

The overall contributions of this work are as follows:

- The presented method, with almost no extra training overhead and without the need for setting additional hyperparameters, can sparsify a model up to an exact user-defined level while keeping an effective non-uniform per-layer budget.
- We first show that modeling per-layer weight distributions as Gaussian and Laplace is sufficient to learn appropriate per-layer thresholds for magnitude based pruning for different model architectures, even at high sparsity rations.
- Extensive experiments on both CIFAR [47] and ImageNet [13] datasets demonstrate that our method achieves state-of-the-art accuracy surpassing more complex and inefficient methods.

5.3 Proposed Method

5.3.1 Pruning Criterion

We follow the magnitude pruning approach where a weight is kept only if its magnitude surpasses a threshold value r. This simple criterion, judging a weight's importance based on its magnitude, is efficient to compute and is found to be effective in the pruning literature, achieving high sparsity ratios with minimal performance loss [29, 21]. The main issue is that this threshold needs to be selected appropriately for each layer in order for the target sparsity ratio to be obtained. Note that even if we intend to keep the same sparsity ratio for all layers, r needs to be calculated individually for every layer since the magnitudes of the weights vary between different layers. Many approaches sort the weights to find the target thresholds for every training step [86, 98], which can lead to training overhead, especially for larger networks. Our method, given a total target sparsity ratio, finds the thresholds efficiently and leads to a non uniform sparsity for optimized performance (see Section 5.3.2).

5.3.2 Learning the Thresholds

One approach to obtain layer-wise adaptive sparsity is to consider the thresholds r_l, for each layer l, trainable parameters. Then, using a suitable *differentiable* function, $\hat{s}_l(r_l) \in [0,1]$, that estimates the sparsity ratio of each layer l, given threshold r_l, the total estimated sparsity of the model can be defined as

$$\hat{S}(\{r_l\}) = \sum_{l}^{N} c_l \hat{s}_l(r_l), \tag{5.3.1}$$

where $c_l = \frac{\#\boldsymbol{W}_l}{\sum_i^N \#\boldsymbol{W}_i}$ is the contribution of layer l to the total network parameters.

If we can cast the per-layer sparsity $\hat{s}_l(r_l)$ as an analytical formulation with respect to the threshold r_l we can have a fully trainable pipeline that can automatically adjust the per-layer sparsity according to a sparsity-oriented loss through Eq. 5.3.1. To this end, [73] considered that, for each layer, the weights can be approximated by a Gaussian distribution. According to this empirical assumption, one can indeed write the sparsity of the layer as an analytical function w.r.t. a threshold, following a confidence interval rationale. We extend this formulation to also include Laplace distributions for improved estimation of sparsity (see Section 5.3.3 for more details).

To promote a specific model sparsity ratio, $S \in (0,1)$, one can define a sparsity loss $L_s(\{r_l\})$ as:

$$L_s(\{r_l\}) = \left[S - \hat{S}(\{r_l\})\right]^2. \tag{5.3.2}$$

This formulation enforces a user-specified sparsity target for practical applications. $L_s(\{r_l\})$ is minimized when the thresholds r_l have obtained values that result to $S = \hat{S}(\{r_l\})$.

This loss can then be added to the typical training loss $L(\{\boldsymbol{W}_l\}, \{r_l\})$ to result to an overall multi-task loss

$$L(\{\boldsymbol{W}_l\}, \{r_l\}) + \lambda_s L_s(\{r_l\}). \tag{5.3.3}$$

Defining the overall loss as above and minimizing with respect to both parameter sets, $\{\boldsymbol{W}_l\}$ and $\{r_l\}$, its terms behave in a competitive way, *i.e.* L is more easily minimized if the model is left dense, hence L_s stays near its maximum value. Therefore some scaling factor λ_s is needed to weight the contribution of L_s.

The fact that the values of the thresholds will be learned through the optimization process (and thus will not be just random or heuristic based thresholds that give the requested total sparsity) should indicate that a highly effective per-layer budget allocation is found (see Section 5.5.4).

Finally, for this process to work, the use of Straight-Through Estimator for back-propagation is imperative to retain unimodal distributions (see Section 5.3.4) and thus facilitating the use of an analytical formula for the sparsity loss approximation.

In our work, we found that the sparsity loss, especially for high values of the target sparsity S, should be highly weighted in order to be substantial and thus minimized by the optimization process. To achieve that,

we further scale it adaptively with the inverse of the squared budget $B^2 = (1 - S)^2$, resulting into the final overall loss equation used in our method

$$L(\{\boldsymbol{W}_l\}, \{r_l\}) + \frac{\lambda_s}{B^2} L_s(\{r_l\}) \tag{5.3.4}$$

The aforementioned scaling is more crucial to our method since, in contrast to [73], we increase the target sparsity ratio gradually through training (as described in Section 5.3.6) for a more smooth transition from a dense to a sparse model and therefore a constant user-defined weight λ_s is not suitable (as shown in Section 5.5.1). With this scaling trick, λ_s is almost unimportant and can have some constant typical value (we set it to $\lambda_s = 10$) regardless the requested sparsity ratio and the experiment.

The key to the above strategy is choosing a function $\hat{s}_l()$ that obtains low sparsity estimation error $es_l(r_l) = |\hat{s}_l(r_l) - \bar{s}_l(r_l)|$, where $\bar{s}_l(r_l)$ is the measured (actual) sparsity ratio. Otherwise, if the estimation error is significant, the resulting sparsity will differ from the requested, which may lead to sparser models with lower accuracy or denser ones with unacceptably high memory requirements or flops for the intended application. Our choice of $\hat{s}_l()$ functions, motivated by the observed distributions of the per-layer weights, is described in the following section.

5.3.3 Modeling Weight Distributions

Considering the per-layer weights $\{W_l \in \boldsymbol{W}_l\}$ as random variables w_l that follow a symmetrical distribution, the sparsity ratio $s_l(r_l)$ at a given threshold value r_l is estimated as the probability that w_l falls in the range $[-r_l,\ r_l]$ as

$$\hat{s}_l(r_l) = \int_{-r_l}^{r_l} f_l(w_l) dw_l \tag{5.3.5}$$

with $f_l(w_l)$ representing the probability density function of the element w_l at the $l - th$ layer.

The weights tend to obtain per-layer distributions that are similar to the Gaussian or Laplace Distributions. Based on that assumption we can compute $\hat{s}_l(r_l)$ by Equation (5.3.5) resulting to closed form expressions as

$$\hat{s}_l(r_l) = \begin{cases} \operatorname{erf}\left(r_l/\sigma_l\sqrt{2}\right), & \text{if } w_l \sim N(0, \sigma_l^2) \\ 1 - \exp\left(-r_l/\beta_l\right), & \text{if } w_l \sim L(0, \beta_l) \end{cases} \tag{5.3.6}$$

$$\text{where} \quad \operatorname{erf}(x) = \frac{1}{\sqrt{\pi}} \int_{-x}^{x} e^{-t^2} dt \tag{5.3.7}$$

The benefit of adopting those candidate weight distribution models is that $\hat{s}_l(r_l)$ can be computed efficiently and remains differentiable. We note that we have assumed zero-mean per-layer weight distributions but our approach can be easily adjusted to hold for non-zero mean weights. Also, in practice we estimate the distribution parameters σ_l^2 and β_l of the weights using their Maximum Likelihood estimates

$$\hat{\sigma_l}^2 = \frac{1}{N} \sum_{i=1}^{N} W_{l_i}^2 \text{ and } \hat{\beta}_l = \frac{1}{N} \sum_{i=1}^{N} |W_{l_i}|.$$

In our method we do not restrict the modeling distributions to one particular type (Gaussian or Laplace) as done in previews works [73, 41]. The type of the per-layer distribution is selected dynamically throughout the training in order to minimize the modeling error. This is important since different layers (or even the same layer at different epochs) obtain weight distributions that are much closer to one of the two modeling types than the other (e.g. see Figure 5.3.1).

5.3.4 Straight-Through Estimator

For the analysis of the previous section to hold, the distributions of the weights during sparse training need to preserve their Gaussian or Laplacian attributes. At the same time the network has to be optimized for the given sparsity ratio. Only updating the active weights, as done in many pruning algorithms [29, 98], is not a suitable approach since this will create non-unimodal distributions [73]. To circumvent this effect caused

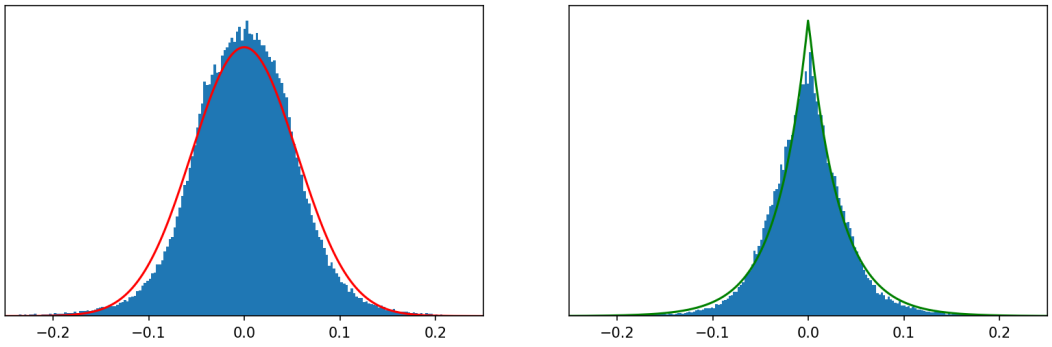


Figure 5.3.1: Example of a layer with Gaussian-like distributed weights (a) and one with Laplacian-like distributed (b). The distributions are from layers layer2.0.conv2 and layer2.1.conv1 of ResNet50.

by zeroing the gradients of the pruned weights, we employ the idea of training with a Straight-Through Estimator (STE). STE was originally introduced by Hinton *et al.* [3] and has been successfully utilized in the context of quantized neural networks [12, 17, 2] and more recently in sparse training techniques [73, 86]. As the name suggests, the operation of thresholding the weights is circumvented during the backward pass, allowing all the weights to be updated based on the gradients computed from loss with respect to the sparse subset of weights. Besides preserving the distributions, sparse training using this scheme has the added advantage of enabling weight reuse. That is, if at some point during the training a weight that has been pruned gets magnitude large enough to pass the pruning threshold, it can safely return to being active due to the fact that is has revived gradient updates throughout the training process.

5.3.5 Switching Distributions

The modeling distribution attributed to some layer l is reconsidered at the end of every training epoch k based simply on minimizing the sparsity estimation error, which is defined as

$$es_l^G(r_l) = |\hat{s}_l^G(r_l) - s_l(r_l)| \tag{5.3.8}$$

$$es_l^L(r_l) = |\hat{s}_l^L(r_l) - s_l(r_l)| \tag{5.3.9}$$

where r_l is the threshold obtained at the end of epoch k, $\hat{s}_l^G(r_l)$ and $\hat{s}_l^L(r_l)$ are the sparsities approximated using a Gaussian and a Laplace distribution respectively, resulting to the estimation errors $es_l^G(r_l)$ and $es_l^L(r_l)$. Therefore if the layer's distribution at epoch=k is Gaussian and $es_l^L(r_l) < es_l^G(r_l)$ switch to using the Laplace distribution to model the layer's weights for the next epoch k+1. Respectively if $es_l^G(r_l) < es_l^L(r_l)$ and the modeling distribution is the Laplace switch to the Gaussian. We note that we keep r_l unchanged regardless if a new distribution is chosen to ensure its continuity. The optimization process should smoothly adjust the thresholds during the next epochs to account for the change of distributions. This strategy is depicted in a state diagram in Figure 5.3.2.

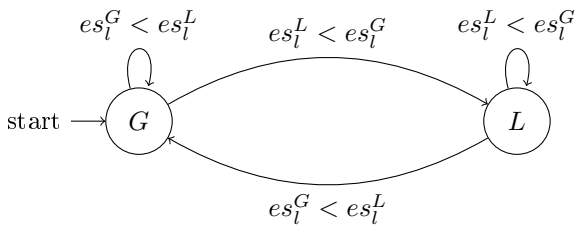


Figure 5.3.2: State transition diagram where states are the two candidate modeling distributions (G: Gaussian, L: Laplace) for layer l. The transitions happen according to the described rules at the end of each epoch. We assume that all layers are initially (at epoch 1) best modeled with a Gaussian distribution.

As an additional method for choosing the distributions we can follow a Ratio of Maximum Likelihood based approach [49]. Assume a sample $w_1, ..., w_n$ generated from either the Gaussian $N(\mu_N, \sigma^2)$ or the Laplace $L(\mu_L, \beta)$ distribution. The likelihood functions of $N(\mu_N, \sigma^2)$ and $L(\mu_L, \beta)$ are

$$l_N(\mu_N, \sigma) = \prod_{i=1}^{n} f_N(w_i; \mu_N, \sigma), \tag{5.3.10}$$

$$l_L(\mu_L, \beta) = \prod_{i=1}^{n} f_L(w_i; \mu_L, \beta) \tag{5.3.11}$$

and the logarithm of the ratio of the two likelihoods is equal to

$$R = \ln\left(\frac{l_N(\hat{\mu_N}, \hat{\sigma})}{l_L(\hat{\mu_L}, \hat{\beta})}\right) \tag{5.3.12}$$

where $(\hat{\mu}_N, \hat{\sigma})$ and $(\hat{\mu}_L, \hat{\beta})$ are the MLEs of (μ_N, σ) and (μ_L, β) respectively. R can be written as

$$R = \frac{n}{2}(1 + \ln 2 - \ln \pi) + n(\ln \hat{\beta} - \ln \hat{\sigma}), \tag{5.3.13}$$

where $\hat{\mu}_N = \frac{1}{n}\sum_{i=1}^{n} w_i$, $\hat{\sigma}^2 = \frac{1}{n}\sum_{i=1}^{n}(w_i - \hat{\mu}_N)^2$, $\hat{\mu}_L = \text{median}(w_1, ..., w_n)$ and $\hat{\beta} = \frac{1}{n}\sum_{i=1}^{n}|w_i - \hat{\mu}_L|$. One can then choose the Gaussian distribution if $R > 0$, otherwise the Laplace distribution to represent the given data. In our case the data samples are the per-layer weights and based on R we can decide the type of distribution being the more appropriate.

For the main experiments we use the error based method as we find it being simpler while giving similar estimates (as demonstrated in Section 5.5.2).

5.3.6 Sparsity Scheduling and Sparsity Fine-tuning Phase

The following sparsity scheduler, based on [98] is used to gradually increase the target sparsity ratio

$$S_i = S_f\left[1 - \left(1 - \frac{i}{n}\right)^3\right], \text{ for } i = 1, 2, ..., n \tag{5.3.14}$$

where i is one training step, n the total number of pruning steps, S_i the target sparsity ratio at step i and S_f the final target sparsity ratio. In our experiments we set n equal to the number of training steps upto 80% of the training epochs. During that time the pruning thresholds are learned as described in Section 5.3.2. We call this the adaptive phase of our method.

Although the sparsity estimation using the weight distributions is rather accurate, the final sparsity ratio might deviate slightly from the requested. If S_f is reached at step $m < n$, the sparsity ratio of each layer, s_l, is linearly decreased for a few iterations to reach its final value. If at step n the measured sparsity $\bar{S}_n$ is lower that S_f, for the next 10% of the epochs the sparsity ratios are linearly increased up to their final values. During only that phase of the method the exact threshold values are calculated via sorting the weights of each layer. For each case the final values of the sparsity ratios (which guarantee that the sparsity becomes exactly S_f) are

$$s'_l = \begin{cases} 1 - \dfrac{1 - S_f}{1 - \bar{S}_n}(1 - s_l), & \text{if } \bar{S}_n < S_f \\ \dfrac{S_f}{\bar{S}_m} s_l, & \text{if } m < n \text{ and } \bar{S}_m > S_f \end{cases} \tag{5.3.15}$$

After the sparsity reaches S_f it is kept constant for the remaining of the training steps. Note that usually $s_l \approx s'_l$ since the sparsity estimation error is expected to be quite low.

5.4 Experimental Evaluation

The present section provides the experimental results of our method as well as comparison with relevant baselines and SoA unstructured pruning algorithms. In the first part of the section we experiment with modern compact architectures ResNet20 [31], MobileNetV1 [35] and DenseNet40-24 [36] on CIFAR-100[1] [47]. We directly compare our results with the ones of GMP [98] and ST-3 [86] generated on the same settings in our environment. Both GMP and ST-3 are dense-to-sparse, magnitude based pruning algorithms which gradually introduce sparsity during training. GMP allocates uniform sparsity across the different layers while ST-3 follows an STE approach with global magnitude threshold. The second part of the section further verifies that our method is state-of-the-art based on ResNet50 [31] experiments on ImageNet [13]. For those experiments we include results from literature (as is standard practice) from an extended number of pruning methods achieved using the same number of epochs (100) and data augmentation. For all models we find most insightful to experiment on sparsity levels 90%, 95% and 98%, since that range of sparsity is challenging but at the same time methods still have a chance to provide models with acceptable accuracy for real-world deployments. All our results are obtained using the same training hyperparameteres (typically used in literature), as shown in Table 5.1.

Dataset	CIFAR-100	ImageNet
Epochs	160	100
Batch Size	128	256
Weight Decay	5e-4	5e-5
Optimizer	SGD	SGD
LR	0.1	0.2
LR-Scheduler	Cosine	Cosine+Warmup
Momentum	0.9	0.9
Label Smoothing	-	0.1

Table 5.1: Training Hyperparameters used for all our experiments on CIFAR-100 and ImageNet datasets.

5.4.1 CIFAR-100

Ratio	90%	95%	98%
ResNet-20 (1.096M Params): 73.59 ±0.44			
GMP [98]	70.34 ±0.33	69.38 ±0.29	64.46 ±0.36
ST-3 [86]	72.86 ±0.20	71.95 ±0.12	**67.73** ±0.10
Ours	**73.18** ±0.11	**72.30** ±0.22	66.69 ±0.24
MobileNetV1 (3.315M Params): 71.15 ±0.17			
GMP [98]	61.82 ±0.18	52.87 ±0.33	32.72 ±1.09
ST-3 [86]	71.02 ±0.09	**70.50** ±0.08	**69.18** ±0.34
Ours	**71.11** ±0.12	70.37 ±0.14	67.53 ±0.22
DenseNet40-24 (0.714M Params): 74.70 ±0.51			
GMP [98]	70.72 ±0.29	68.29 ±0.37	61.56 ±0.19
ST-3 [86]	72.82 ±0.43	**71.66** ±0.38	**65.99** ±0.38
Ours	**74.00** ±0.30	71.63 ±0.32	63.20 ±0.31

Table 5.2: Accuracy of ResNet20, MobileNetV1 and DenseNet40-24 on CIFAR-100 at varying sparsity ratios. The initial accuracy of the dense model is also reported for comparison.

Table 5.2 provides the results obtained on CIFAR-100 using GMP, ST-3 and our method over three different architectures and three different levels of sparsity. The results first indicate the robustness and adaptability of our approach using the Gaussian and Laplace distributions for modeling the per-layer weights of three different compact models at varying pruning ratios, easily surpassing the results obtained from GMP (which ignores the importance of finding a per-layer non-uniform budget). Compared to ST-3, we observe improved performance for the 90% sparsity ratio, similar performance for the 95% ratio, while our approach slightly

[1] The channels of ResNet20 are doubled for the experiments on CIFAR, as also done in [97, 87]. Also for CIFAR we do not prune layers with under 1000 parameters as we consider those having negligible overall contribution to the total budget.

under-performs at the 98% ratio (see Section 5.6). We want to emphasize that ST-3, although can provide SoA results, is more computationally expensive than our method due to the fact that it needs to sort all the weights (which are stacked together in a tensor) in order to find the global threshold at every update step. Our method, being the most compute friendly among all considered baselines, finds the thresholds without the use of sorting for the most part of the training (only first and second-order statistics are required, which can be calculated in linear time with respect to the number of the per-layer weights) and only sorts the *per-layer* weights without aggregating them (thus still more efficient) for the last epochs. Still, it is found to be the best performing for the 90% sparsity ratio resulting in accuracies with almost no degradation from the ones obtained from dense training. This is possible due to the effective per-layer sparsity ratios that are found automatically during training as described in Section 5.3.2. This is further analyzed in Section 5.5.4.

5.4.2 ImageNet

Ratio	90%	95%	98%
ResNet-50 (25.6M Params): 77.10			
GMP [98]	73.91	70.59	57.90
STR [50]	74.31	70.40	61.46
ProbMask [97]	74.68	71.50	66.83
ST-3 [86]	76.03	74.46	**70.46**
Spartan [84]	76.17	**74.68**	-
Ours	**76.27**	74.27	67.83

Table 5.3: Accuracy of ResNet50 on ImageNet.

To prove the generalization abilities of our method and its ability to stand against other, more computationally expensive and hyperparameter dependent SoA methods, we conduct experiments on ImageNet with ResNet50. In addition to results from GMP and ST-3, in this section we include results from methods STR [50], ProbMask [97] and Spartan [84], all being recent SoA dense-to-sparse unstructured pruning methods. STR uses a soft-threshold operator to obtain learned non-uniform sparsity which is controlled indirectly by the weight decay parameter. This makes their method very hard to obtain a specified sparsity ratio. Probmask approximates the pruning problem using a probabilistic mask which is transformed into a deterministic binary mask using samples from a Gumbel distribution. The particular approach requires multiple sampling rounds with gradient computation which significantly increase the training time. Finally, Spartan learns a soft top-k mask using a regularized optimal transportation problem, thus being also a computational heavier method than ours.

Based on the results provided in Table 5.3, our method reaches top performances at 90% and 95% ratios, surpassing GMP, STR and ProbMask and matching in results of ST-3 and Spartan. At the 98% ratio it also surpasses the first three methods and comes second to ST-3. At this point we have to note that we used the same, default training hyperparameters for all three sparsity ratios while in ST-3 the weight decay is progressively decreased as the sparsity ratios are increased - no exploration was provided on the necessity of such fine-tuning choice. We opted to show that our method is able to perform very well independently of the hyperparameters used and with the lowest overhead among the competing methods. This claim is supported by the obtained results, both on CIFAR-100 and on ImageNet.

5.5 Ablation Studies

In this section we explore various aspects of our approach, highlight the effectiveness of the proposed components and provide an in-depth analysis of the sparsity distribution across different layers.

5.5.1 Impact of Scaling the Sparsity Loss

Compared to typical gradual magnitude pruning approaches, where only the target sparsity is increased, in our work we also need to consider the gradual amplification of the sprasity loss. Specifically, the sparsity loss $L_s(\{r_l\})$ is adaptively scaled by the inverse of the squared training budget before it gets added to the overall loss (Equation 5.3.4). This is found to be necessary in order to force the threshold parameters to grow

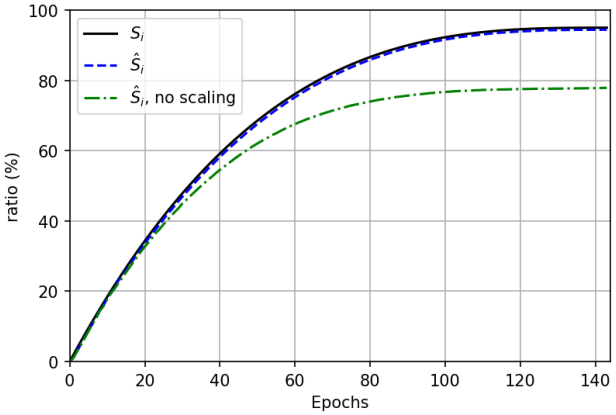


Figure 5.5.1: The estimated sparsity at each epoch, $\hat{S}_i$, closely follows the requested one, S_i. Without adaptively scaling the sparsity loss fails to grow and ($\hat{S}_i$, no scaling) deviates from S_i. Plot values from training ResNet20 on CIFAR-100.

(and hence sparsify the model) as the sparsity scheduler progresses. Otherwise after some point in training, the sparsity loss, which works in a competitive way with the task-related loss (e.g., Cross Entropy for the explored classification tasks), will not be high enough to be considered resulting in a severe deviation of the estimated sparsity from the requested (Figure 5.5.1).

5.5.2 Impact of Using Both Distributions

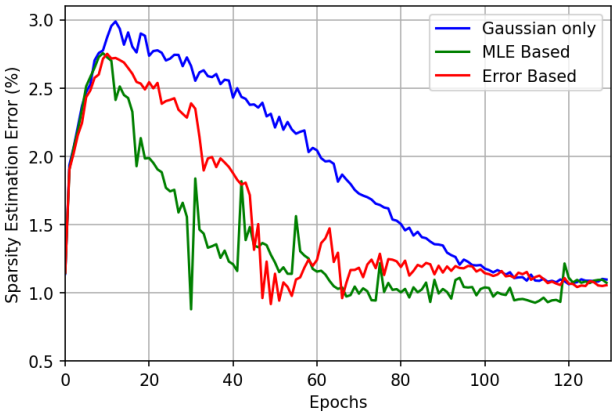


Figure 5.5.2: Total sparsity estimation error per epoch, $es = \bar{S}(\{r_l\}) - \hat{S}(\{r_l\})$ (both quantities in es represent sparsity ratios in %), for the adaptive part of training ResNet20 to 95%.

To better understand the importance of using both the Gaussian and the Laplace distributions, we display the total sparsity estimation error per epoch, $es = \bar{S}(\{r_l\}) - \hat{S}(\{r_l\})$, where $\bar{S}(\{r_l\})$ is the measured (actual) sparsity and $\hat{S}(\{r_l\})$ is the sparsity estimated by the distributions. It is clear that the addition of the Laplace distribution as potential model for the weights of some layers results to lowering the estimation error for a good portion of the training epochs, until the last few ones when the distributions are observed to converge to more Gaussian-like (at least for the particular experiment). We also observe that the two approaches described in Section 5.3.5 (Error and MLE based) for the selection of the per-layer distributions result to similar error plots.

5.5.3 Comparison with ASL

In this subsection we directly compare our method to the closely-related adaptive sparsity loss (ASL) approach from [73] to further highlight the enhanced effectiveness of the proposed modifications. ASL, since being the

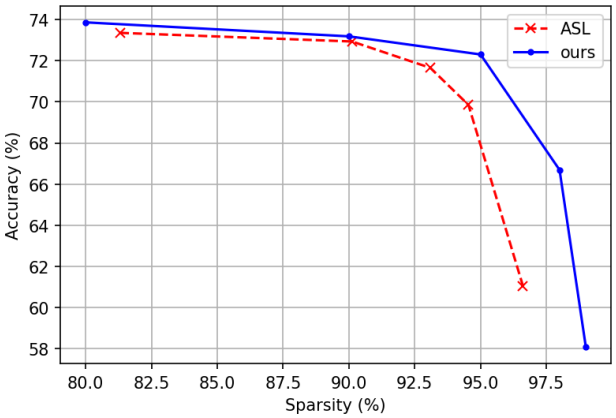
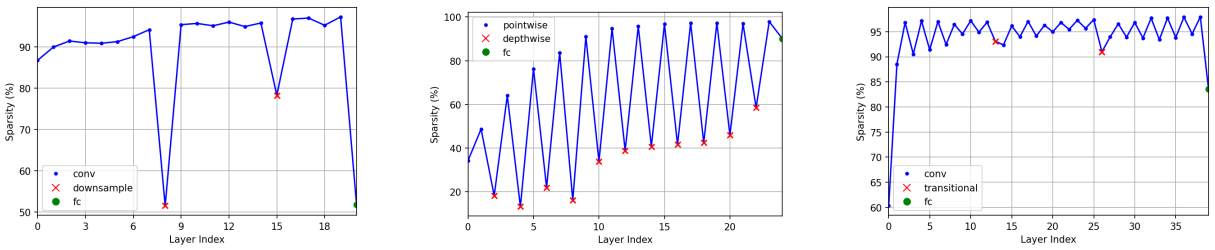


Figure 5.5.3: Accuracy of ResNet20, pruned using our method and ASL on CIFAR-100 at varying sparsity ratios.

initial motivation for our work, is also based on the idea of finding the thresholds through training, using the sparsity loss of Equation 5.3.2, but only using Gaussian distributions for modeling the weights. Moreover in ASL the sparsity is introduced abruptly from the beginning of the training since no sparsity scheduler is used. Also there is no sparsity fine-tuning phase thus the final sparsity can vary from the requested, a very common problem when considering high sparsity levels. In Figure 5.5.3 we present results from training ResNet20 on CIFAR-100 for multiple sparsity ratios using the same settings as in the experiments in Section 5.4.1. Due to the fact that the sparsity loss of ASL is not adaptively scaled we need to manually increase its weight factor λ_s (of the ASL muilti-task loss) as we increase the requested final sparsity in order to avoid large deviations in obtained sparsity and provide meaningful comparisons. The results are presented in a graph since ASL could not produce the exact requested sparsity. It is clear from the graph that our method, apart from being able to achieve the exact requested sparsity, results in better accuracies, especially for challenging sparsity ratios, where the improved (as shown is Section 5.5.2) sparsity estimation and the gradual introduction of sparsity become especially important.

5.5.4 Per-Layer Sparsity Distribution



(a) ResNet20, 95% sparse. (b) MobileNetV1, 95% sparse (c) DenseNet40-24, 95% sparse

Figure 5.5.4: Per-layer sparsity ratios (%) for models trained on the CIFAR-100 dataset. Different modules (e.g., conv, fc) are marked with different colors.

The per-layer sparsity distributions obtained from training the models ResNet20, MobileNetV1 and DenseNet40-24 on CIFAR-100 at a 95% final sparsity are shown in Figure 5.5.4. For ResNet20 our method increases the sparsity ratio gradually from the first to the last layers, with the exception of the two down-sampling layers and the fully connected layer which are assigned considerably lower sparsity ratios. For the case of MobileNetV1, the sparsity allocation also increases gradualy but at a different level for the two types of layers, pointwise and depthwise, with the latter being assigned much lower sparsity ratios. Finally the per-layer sparsity of the DenseNet40-24 shows that the first and last layers are assigned much less sparsity than the rest, which appear to have a fluctuating sparsity pattern (partially based on their size). Notably,

the two transition layers are assigned lower sparsity than most of their neighboring layers despite having more parameters. This last observation indicates that our method allocates sparsity not only based on the per-layer number of parameters (as done by most heuristic methods) but also based on the functionality of each layer, thus is able to detect bottleneck layers and keep them adequately dense to avoid performance degradation.

5.6 Limitations and Future Work

The main limitation of the proposed method concerns requesting very high sparsities (generally above 98%) from relatively small models. For such cases, while still able to provide models with accuracy better than most baselines, it falls behind very recent SoA methods (see Tables 5.2, 5.3). This is to be expected as the modeling of the distributions of weights, although surprisingly accurate for most cases, is not exact (as shown in Figure 5.6.1 (b) - note however that many layers still retain a shape closely to the considered distributions). Effects like that of Figure 5.6.1 (b) are more frequent as we push the sparsity requirement to extremely high percentages ($> 98\%$). Therefore even a slight deviation from the sparsity target can result in a considerable deviation from the respective budget, and has to be corrected by the last phase of the method (Section 5.3.6).

Working towards making the sparsity estimation even more accurate is a possible future direction. This could be done by enforcing the distributions in some way to resemble even more one of the two used modeling types, or by considering more modeling distribution types. Following the first approach one has to be careful not to affect the model accuracy by imposing restrictions on the weights or influencing the learning process. For the second approach the difficulty is identifying appropriate models for the weights' distributions which are accurate but at the same time simple enough to be differentiable in order for the sparsity ratios to be learned by the optimization process.

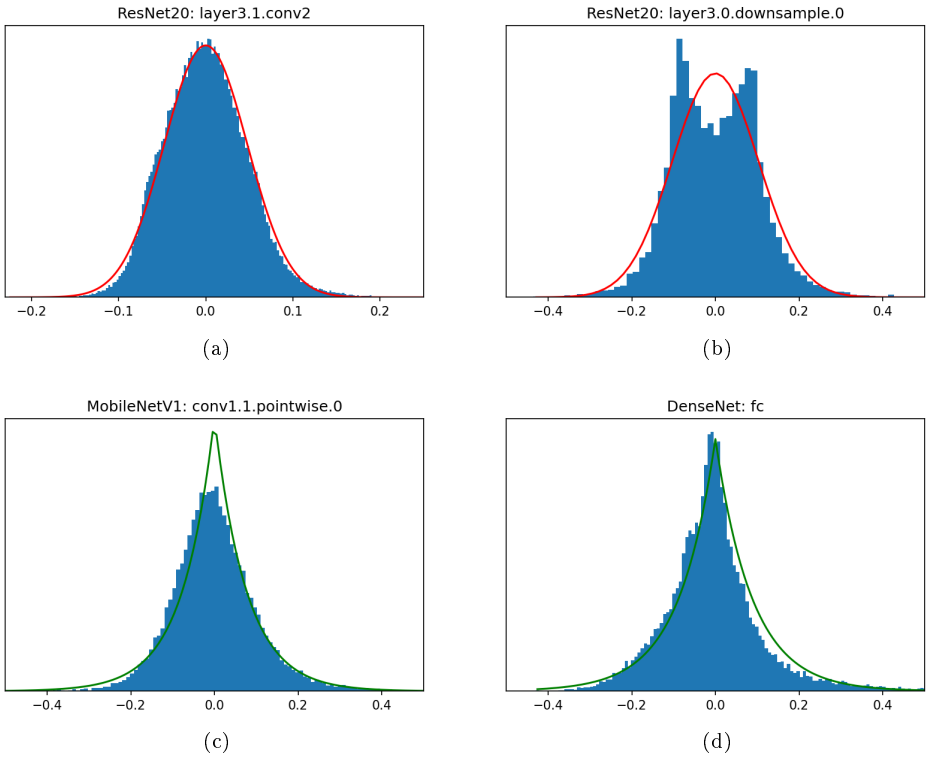


Figure 5.6.1: Examples of per-layer weight distributions obtained during training on the CIFAR-100 dataset using our pruning method. Some layers are modeled better than others.

5.7 Conclusions

This work proposes a computationally efficient and SoA performing pruning method, as demonstrated by extensive experiments on CIFAR and ImageNet datasets, where it surpasses more sophisticated methods with increased training overheads. Furthermore, its success indicates that non-uniform layer-wise sparsity can be effectively learned defining a extra loss based on a weight distribution modeling way, with the Gaussian and Laplace distributions being adequate, even for relatively high sparsity ratios.

Chapter 6

Feather: An Elegant Solution to Effective DNN Sparsification

6.1 Abstract

Neural Network pruning has attracted a lot of attention in the recent years due to its ability to produce highly compressed, yet accurate models, enabling their use to resource-constrained environments. While the pruning can be performed using a multi-cycle training and fine-tuning process, the recent trend is to encompass the sparsification process during the standard course of training. To this end, we introduce Feather, an efficient sparse training module utilizing the powerful Straight-Through Estimator as its core, coupled with a new thresholding operator and a gradient scaling technique, enabling robust, out-of-the-box sparsification performance. Feather's effectiveness and adaptability is demonstrated using various architectures on the CIFAR dataset, while on ImageNet it achieves state-of-the-art Top-1 validation accuracy using the ResNet-50 architecture, surpassing existing methods, including more complex and computationally heavy ones, by a considerable margin.

6.2 Introduction

A recent trend in the pruning literature is to perform the pruning along the normal course of training (sparse training) without the need for additional retraining cycles. A number of works [73, 44, 84, 86] which are built around the concept of the Straight-Through Estimator (STE) [3] have demonstrated that SoA results can be achieved using this approach. Sparse training with the STE is performed by computing the forward pass using the thresholded (pruned) version of the weights while updating the dense weights during the backward pass, treating the thresholding function (that performs the pruning) as the identity.

In this chapter, we focus on improving sparse training with the STE by addressing a number of overlooked shortcomings of the method, eventually introducing a novel pruning module. We propose (i) a new thresholding function used for magnitude pruning and (ii) a straightforward way to control gradient flow of the pruned weights. More specifically, instead of using hard or soft thresholding [18], which previous STE based methods mostly use, we propose a family of thresholding functions that lie in between the aforementioned two and combine their advantages, namely reduced bias between the thresholded weights and their dense counterparts and a smooth transition region near the threshold. Complementary to the proposed thresholding approach, we suggest scaling the gradients attributed to pruned parameters by a parameter $\theta \in (0, 1)$, aiming on improving the stability of the pruning mask, a factor that we find to be crucial when targeting very high sparsity ratios.

We demonstrate the effectiveness of our sparse training approach when applied to magnitude based unstructured pruning frameworks, which reach a user-specified sparsity ratio by incrementally pruning the network during training. In more detail, we perform extensive experiments on both CIFAR [47] and ImageNet [13] datasets using a very simplistic global thresholding pruning procedure as a backbone and additionally with a recently proposed layer-wise pruning framework [73]. Our sparse training approach surpasses the (generally more computationally expensive) current SoA unstructured pruning algorithms, significantly improving the previously achieved generalization accuracies of the resulting sparse models.

Overall, the contributions of our work can be summarized as follows:

- We introduce Feather, a versatile sparse training module that can be used to efficiently and effectively prune neural networks up to extreme sparsity levels. The proposed module was evaluated on different magnitude pruning backbones and achieved consistent improvements over the prior state-of-the-art.
- We highlight the importance of a well-crafted thresholding function that finds a fine balance between the two standard ones, namely hard and soft operators.
- We highlight the correlation between scaling the gradients of the pruned weights and the targeted sparsity ratio: high sparsity targets should be accompanied with lower scaling values to provide high performing pruned models.

6.3 Proposed Method

In this work, we developed a novel pruning module, dubbed as Feather. The name symbolizes its lightweight nature, the elegance with which it achieves sparsity, as well as the lightness of the resulting pruned networks. It can be utilized in various magnitude pruning schemes, including strategies where pruning can be performed globally or in a layer-wise fashion. The setting of interest is magnitude pruning, where a weight is kept only if its magnitude surpasses a threshold value T, and a sparse training process is followed (i.e., perform the pruning procedure along the standard course of training), as done in most of the SoA systems. In essence, we capitalize on the effectiveness of modern STE-based approaches, carefully addressing potential issues. Implementation-wise, the proposed module is applied at each layer, replacing a typical pruning operation, affecting both the forward and the backward step.

In what follows, we first provide the necessary prerequisites for sparse training with STE and then we describe the proposed module, emphasizing on the proposed modifications on both the forward and the backward steps and highlighting the underlying motivations.

6.3.1 Preliminaries: Sparse Training

In this analysis we examine how the pruning and the subsequent weight update is performed (during a training iteration) on a single layer under the STE framework.

First, let us consider a thresholding operator, the core of the magnitude pruning approaches, as a function $\mathcal{P}_{(T)}(x)$ that performs the pruning given a threshold value T. As a general rule this function takes zero values when $|x| \leq T$ and non-zero values otherwise. Typical instantiations of this function are the hard and the soft thresholding operators.

A first approach is to directly back-propagate through the thresholding operator. By doing so, the gradients belonging to the pruned weights will be zeroed thus excluding them from the update step. Regrettably, due to the resulting sparse gradient, this approach may lead to unwanted decay of immaturely trained weights and a slow exploration of the possible sparsity patterns [44, 84].

Lately, to circumvent the aforementioned issues, a number of pruning methods [73, 44, 84, 86] have achieved SoA results relying on the concept of a Straight-Through Estimator [3] training approach. In a nutshell, with the STE formulation, the weight update and thresholding equations are now decoupled into:

$$\tilde{\boldsymbol{w}}_k = \mathcal{P}_{(T)}(\boldsymbol{w}_k) \tag{6.3.1}$$

$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k - \eta \cdot \nabla \mathcal{L}(\tilde{\boldsymbol{w}}_k), \tag{6.3.2}$$

where $\boldsymbol{w}$ are the vectorized layer weights, $\tilde{\boldsymbol{w}}$ the pruned weights after applying the thresholding operation, $\mathcal{L}(\boldsymbol{w})$ the loss function and η the step size. The reported expressions correspond to k-th iteration of a Gradient Descent formulation to highlight the effect of STE. The same procedure is trivially extended to any optimizer required.

The main idea is to consider the thresholding operator as the identity function during back-propagation and update both pruned and unpruned weights based on the gradients of the loss *w.r.t.* the sparse set of weights. During the forward pass however, only the sparse weights are used so that the network is trained under the sparsity constrain. The key benefit of sparse training with the STE is that it allows for pruned weights to become active again, if at some point during the training they get a large enough magnitude. This process therefore promotes the exploration of different sparsity patterns and is found to result to better performing sparse networks [44, 84].

6.3.2 Proposed Sparse Training Module

The aforementioned STE pipeline was a stimulus for the proposed module; our primary goal was to maintain the simplicity of the original idea, and thus we concentrated on enhancing two fundamental elements: the thresholding operator during the forward step and the gradient manipulation during the backpropagation step. Overall the proposed enhancements aim to assist the training process by promoting convergence to well-performing yet highly sparse solutions. The functionality of Feather, the proposed module, is summarized in Figure 6.3.1a, where the depicted components will be described in detail in what follows.

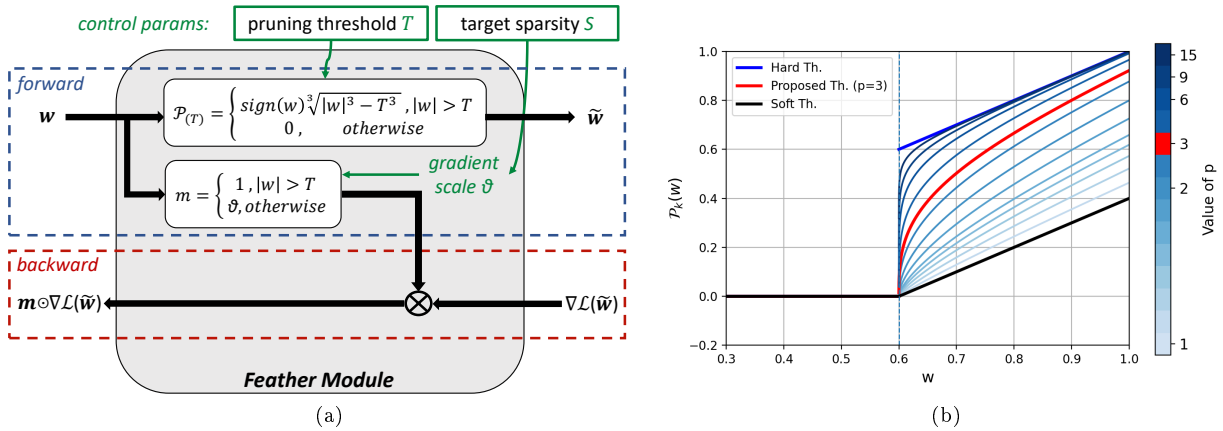


Figure 6.3.1: (a) The proposed sparse training block, utilizing the new thresholding operator and the gradient scaling mask (b) The proposed family of thresholding operators for varying values of p. We adopt $p = 3$, resulting to a fine balance between the two extremes, hard and soft thresholding respectively.

STE Thresholding Operator

Focusing on the thresholding operator used during the forward pass, the most straightforward way to define a magnitude pruning step is through the hard thresholding function, which is discontinuous at the threshold value. This discontinuity might result to training instabilities when weights pass from pruned to active states and vice versa, since the gradient received might not justify the value after thresholding. To addressing this matter, a recent method [86] suggested that the soft thresholding operator is preferable in the context of STE, while soft thresholding is a common choice in pruning approaches in general [50]. Soft thresholding, although suppressing the discontinuity, induces a constant bias (equal to the threshold value) between the active weights and their dense counterparts, updated during the STE back-propagation phase. Note that by its nature the STE introduces an inconsistency between the forward and the backward pass, since during the former the sparse set of weights is used whereas during the later the computed gradients are used to update the dense weights.

Motivated by works in the area of sparse regression, aiming on finding thresholding operators that lie in between hard and soft thresholding [58, 27], we propose the following family of operators to be used with STE:

$$\mathcal{P}_{(T)}(w) = \begin{cases} \text{sign}(w) \cdot (|w|^p - T^p)^{1/p}, & \text{if } |w| > T \\ 0, & \text{otherwise} \end{cases} \tag{6.3.3}$$

The behavior of $\mathcal{P}_{(T)}$ is depicted in Figure 6.3.1b, for varying values of the power parameter p. Intuitively, as p grows we deviate from the soft ($p = 1$) and approach the hard thresholding operator. Under that point of view, the proposed operator can be considered a generalization of the existing two. Note that the proposed function, when avoiding extreme selections for p, tries to balance between the two aforementioned properties: continuity and bias. Based on that, we suggest a value of $p = 3$ being a reasonable compromise that adequately addresses both issues. We provide further validating experiments in the appendix.

In a final note, we want to emphasize that our proposed thresholding operator, apart from having an intuitive explanation for the resulting improved performance (as empirically evaluated in Section 6.5), due to its simplicity it practically imposes no training overhead and can be used off-the-self in pruning frameworks in conjunction with the STE.

STE and Gradient Scaling

The main motivation behind the STE is to allow gradient flow to pruned weights and thus enable the exploration of multiple sparsity patterns during sparse training. As stated before, this is achieved by treating the thresholding function as the identity during the back-propagation. However, in certain cases it might be beneficial to limit the variations of the mask and favor a more stable sparsity pattern. We find that a rather straightforward, yet intuitive way to control the mask's stability is to scale the gradients of the pruned

weights by a constant value $\theta \in [0, 1]$, effectively modifying the update step of Eq. 6.3.2 into:

$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k - \eta \cdot \boldsymbol{m}_k \odot \nabla \mathcal{L}(\tilde{\boldsymbol{w}}_k), \tag{6.3.4}$$

where $\boldsymbol{m}_k \in \{\theta, 1\}^N$ such that $m_{i,k} = 1$ if $w_{i,k} > T$ and $m_{i,k} = \theta$ otherwise, with $\odot$ denoting element-wise product.

At the two extreme points, $\theta = 0$ and $\theta = 1$, the method aligns with the non-STE and the standard STE approaches. For in-between values of θ, pruned weights continue to receive gradient updates but with scaled down magnitudes, and therefore are, to some extent, promoted to decay, resulting to a more stable (but not completely fixed) mask.

In our experiments (Section 6.5.1) we find that scaling the gradients becomes beneficial when targeting very high sparsity ratios (*e.g.* 98% and above) where, due to having very few active weights, too frequent mask variations appear to destabilize the network. For more conservative ratios, using $\theta < 1$ appears not to improve the results and even lead to a small decline in accuracy at some cases.

To this end, based on experimental evidence, essentially relying on a profiling approach, we define an "automatic" way to set θ. In more detail, we select $\theta = g(S)$ as a simple step function, where $\theta = 0.5$ if the final target sparsity S is over 95% and $\theta = 1$ when $S \leq 95\%$. The value of θ is selected at the beginning of training and remains constant throughout the training process. Interestingly, no performance gains were observed by adopting more complex scheduling tactics (e.g. having a gradual smooth transition over θ from 1 to a lower value) during training. Note that ideally a more complex function could be defined given an exhaustive profiling procedure, but we considered such ideas out of scope for this work.

6.4 Application on Pruning Frameworks

Our proposed sparse training module is versatile and not restricted to a particular pruning framework. We will demonstrate its effectiveness using two distinct frameworks, both a global and a layer-wise magnitude pruning backbone, as described bellow.

Global Magnitude Pruning: Most pruning methods that utilize the STE (or some variant of it) use global thresholding, in the sense that a single threshold T is selected for all layers, computed by sorting all weights, in order to prune the network up-to a specified sparsity ratio [44, 86, 84]. Common practice is to incrementally increase the requested ratio throughout the training process, thus giving the network time to adjust to different sparsity levels [98, 50]. A popular sparsity schedule was proposed in [98], where the network is trained densely for a small number of warm up epochs followed by a cubical increase of the sparsity ratio, until it reaches the final target ratio, which is then kept constant for the rest of the training epochs. For simplicity and since the exact sparsity schedule is not the focus of this work, we reach the final target ratio at 50% of the epochs, without having a densely training warm up phase which we found not to have an effect to the final performance.

Layer-wise Magnitude Pruning: To further reveal the efficacy of Feather, we adopted a considerably different pruning framework and pair it with the proposed module. We considered the Adaptive Sparsity Loss framework (ASL) [73], which tackles the magnitude pruning problem in an explicit layer-wise formulation. In particular, we employ our enhanced version of the framework, as presented in Chapter 5. As explained in detail in the aforementioned part of this work, ASL considers the per-layer pruning thresholds as trainable parameters which are included in an additional loss term (Sparsity Loss), constructed based on assumptions for the per-layer distributions. The thresholds are consequently learned during training by minimizing the extra loss term, resulting into a learned non-uniform per-layer sparsity. The overall sparsity of the DNN is constrained, via the loss, to a specific target sparsity. It should be noted that STE is also a core element of this approach. In addition to the improvements proposed in Chapter 5, to correct any deviations of the measured (actual) sparsity $\bar{s}(\{r_l\})$ from the estimated one, $\hat{s}(\{r_l\})$, we corrected the per-layer trained r_l by adding a non differentiable term d_l to it, such that $\bar{s}(r_l + d_l) = \hat{s}(r_l)$. We compute d_l by sorting the per-layer weights. This additional step, although generally has a very minor effect to the final results of the method,

alleviates the need for the sparsity fine-tuning phase of Section 5.3.6. We refer to this version of the method as ASL+.

6.5 Experimental Evaluation

The present section provides the experimental validation of the proposed method's effectiveness. First, we perform ablation studies to showcase the efficacy of each element of our approach. Then, we provide comparisons with relevant baselines and SoA unstructured pruning algorithms. Specifically, we experiment with modern compact architectures ResNet-20 [31], MobileNetV1 [35] and DenseNet40-24 [36] on CIFAR-100[1] [47]. Furthermore, we provide large-scale experiments on the ImageNet[13] dataset using the ResNet50 [31] architecture in order to further verify the generalization abilities of our method and its performance edge over the current SoA. The reported results are obtained using SGD optimizer along with a Cosine Annealing scheduler, while the training hyperparameters are the typically used in literature for such settings (described in detail in the appendix).

6.5.1 Ablation Studies

The ablation studies were performed on the CIFAR-100 dataset, using the global magnitude pruning framework as the backbone to the Feather pruning module. We note that similar behaviors were observed using the Feather in conjunction with the layer-wise ASL+ framework. All figure data points represent averages of 3 runs and the corresponding standard deviations are shown as shaded regions.

Impact of Thresholding Operator: The effectiveness of the proposed thresholding function (with $p = 3$) is empirically evaluated and compared against that of the hard and soft threshold functions in Figure 6.5.1, while no gradient scaling was considered for this experiment (we used typical STE). The proposed threshold enables the training of more accurate sparse networks, especially at high pruning ratios (above 95%). Notably, the hard thresholding approach achieves considerably low results with the MobileNetV1 network, while the soft threshold with the other two. *Overall, our approach consistently leads to better trained networks, regardless the architecture and the sparsity ratio, supporting our claims of effectively combining the strengths of both soft and hard thresholding.*

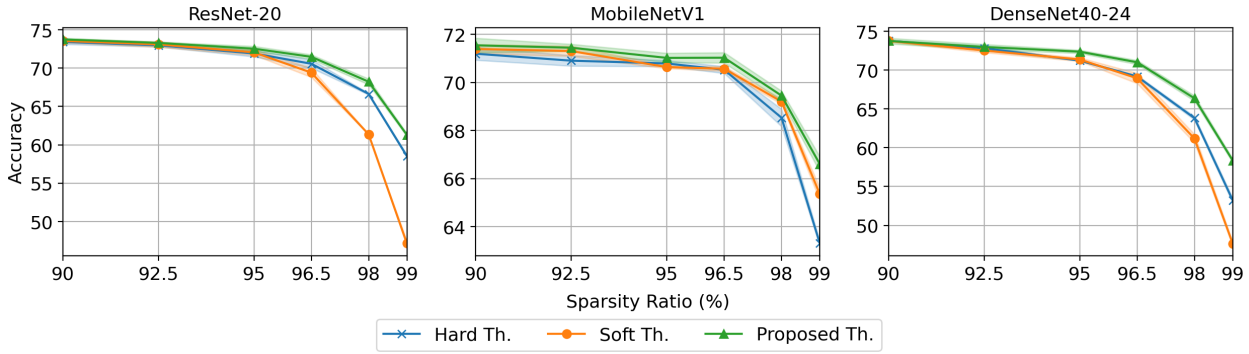


Figure 6.5.1: Study of the effect of the thresholding operator on the final sparse model accuracy. The proposed threshold steadily outperforms the hard and soft operators.

Impact of Gradient Scaling: In Figure 6.5.2 we investigate the relation between the value of the parameter $\theta \in [0, 1]$, that scales the gradient of the pruned weights, and the final model accuracy when requesting different sparsity ratios. Note that the under-performing case of $\theta = 0$ is equivalent to a non-STE variant. *Across all considered models, a clear trend can be seen; When targeting lower sparsity levels, best results are achieved with values of θ close to unity, whereas at more extreme sparsity ratios (such as 98% and 99%) the optimal values of θ seem to shift towards the middle of its range.*

[1] The channels of ResNet-20 are doubled for the experiments on CIFAR, as also done in [97, 87].

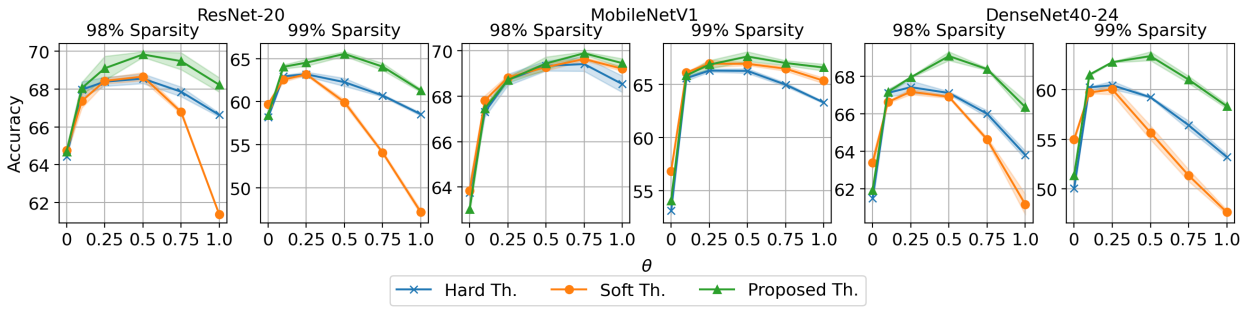


Figure 6.5.3: Gradient scaling improves the final accuracy at high sparsity, regardless the thresholding operator, while maximum performance is achieved if combined with the proposed threshold.

This observed dependency is the motivation behind an automatic selection of θ, as described in Section 6.3.2, where $\theta = 0.5$ for $S > 95\%$ and $\theta = 1$ otherwise. Note that this selection just aligns with the reported trend and is not optimal for every case reported. Despite this being a very "crude" selection, it is very effective, as the forthcoming experimental evaluations hint. Nonetheless the takeaway of this experiment is not a simple function of two modes, but bringing this relation to the spotlight. Based on this observation, we pave the way towards more complex functions or, more interestingly, towards having different scales per layer, relying on the per-layer sparsity rather than the overall sparsity. The later idea, a possible future direction of practical value, simply states that under-pruned layers can be more flexible to sparsity pattern variations than the overly pruned ones.

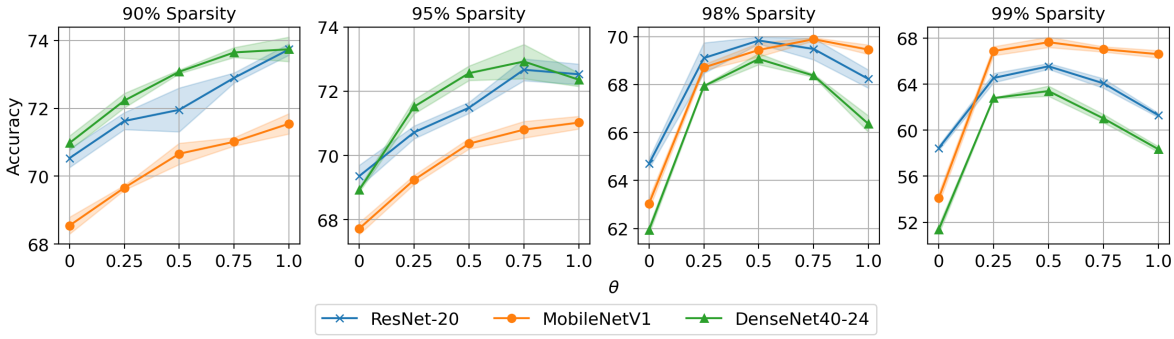


Figure 6.5.2: Study of the effect of gradient scaling. Under conservative final sparsity, θ near unity is preferable, while when targeting high sparsity, models benefit from θ near the middle of its range.

Gradient Scaling under different Thresholding Functions: Finally, we compare the impact of gradient scaling using the three different types of thresholding operators. Figure 6.5.3 shows that, regardless the choice of operator, using a scale $\theta \in (0, 1)$ is beneficial to the final accuracy at the high sparsity regimes. Nevertheless, our threshold maintains the lead in performance compared to the two standard ones.

6.5.2 Comparison to SoA

CIFAR-100: For these experiments, we directly compare our results with the ones of ST-3 [86] and Spartan [84], being the two most recent and best performing sparse training approaches. The methods are both dense-to-sparse, global magnitude based pruning algorithms, which gradually introduce sparsity during training, utilizing variants of the STE. Specifically, ST-3 adopts soft thresholding and a weight rescaling technique similar to the one used by dropout [79], while Spartan computes a soft top-k mask by solving a regularized Optimal Transportation problem, therefore is more computationally expensive than our approach.

In particular, Table 6.1 provides the results obtained on CIFAR-100 using the aforementioned methods over three different architectures and four different levels of sparsity. The reported results correspond to averages

Ratio	90%	95%	98%	99%
ResNet-20 (1.096M Params): 73.59 ±0.44				
ST-3 [86]	72.81 ±0.13	71.72 ±0.20	67.53 ±0.53	58.32 ±0.17
Spartan [84]	72.56 ±0.35	71.60 ±0.40	67.27 ±0.31	61.70 ±0.21
Feather-Global	**73.74** ±0.17	**72.53** ±0.32	**69.83** ±0.14	**65.55** ±0.25
Feather-ASL+	72.86 ±0.10	72.42 ±0.17	69.76 ±0.09	64.95 ±0.47
MobileNetV1 (3.315M Params): 71.15 ±0.17				
ST-3 [86]	70.94 ±0.25	70.44 ±0.23	69.40 ±0.06	66.63 ±0.15
Spartan [84]	70.52 ±0.51	69.01 ±0.11	65.52 ±0.24	60.65 ±0.22
Feather-Global	**71.55** ±0.30	71.03 ±0.20	**69.44** ±0.29	67.64 ±0.45
Feather-ASL+	71.10 ±0.31	**71.26** ±0.10	69.42 ±0.12	**67.86** ±0.03
DenseNet40-24 (0.714M Params): 74.70 ±0.51				
ST-3 [86]	72.56 ±0.31	71.21 ±0.35	65.48 ±0.18	56.18 ±0.60
Spartan [84]	73.13 ±0.25	71.61 ±0.04	65.94 ±0.07	58.64 ±0.18
Feather-Global	73.75 ±0.36	72.36 ±0.21	69.06 ±0.23	**63.40** ±0.44
Feather-ASL+	**73.92** ±0.19	**72.47** ±0.12	**69.08** ±0.19	62.94 ±0.14

Table 6.1: Comparison of Top-1 accuracy on CIFAR-100.

Ratio	90%	95%	98%	99%
ResNet-50 (25.6M Params): 77.10				
GMP [98]	73.91	70.59	57.90	44.78
DNW [91]	74.00	68.30	58.20	-
STR [50]	74.31	70.40	61.46	51.82
ProbMask [97]	74.68	71.50	66.83	61.07
OptG [95]	74.28	72.45	67.20	62.10
ST-3 [86]	76.03	74.46	70.46	63.88
Spartan [84]	76.17	74.68	-	63.87
Feather-Global	**76.93**	**75.27**	**72.92**	**68.85**

Table 6.2: Comparison of Top-1 accuracy on ImageNet.

of 3 runs with the corresponding standard deviations. The results demonstrate that sparse training with Feather yields steadily more accurate models compared to both SoA methods, either using the simple global pruning approach or combined with ASL+. Notably, the gap in accuracy between our approach and that of the next best performing baseline grows up to 4% when considering the 99% sparse ResNet-20 and DenseNet40-24 models. Another interesting remark is that Feather can result to 90% sparse ResNet-20 and MobileNetV1 models with slightly better generalization accuracies that those of their dense counterparts, trained using the same number of epochs. The last remark hints that a well designed sparse training method can even be beneficial, not only for producing compact models, but also for improving the generalization performances, when considering relatively conservative sparsity ratios. Note that similar SoA results are observed using both tested backbones, a point that further validates Feather's versatility.

ImageNet: For the ImageNet experiments we include results from literature from an extended number of pruning methods achieved using the same number of epochs (100) and data augmentation. In particular, results from the relevant methods GMP [98], DNW [91], STR [50], ProbMask [97], OptG [95], ST-3 [86] and Spartan [84] are presented in Table 6.2. We adopt the global pruning scheme combined with Feather module, being conceptually the simplest approach, in order to highlight our method's effectiveness compared to more sophisticated baselines. As we can see, it provides considerably better results than those from previous SoA, especially at very challenging pruning ratios over 98%. We want to emphasize that the improved performance with Feather does not come at the cost of higher training overheads or the need for complicated hyperparameter settings, in contrast to certain baselines (*e.g.* [50, 97]). Instead, the resulting accuracy gains can be attributed to the simple, yet carefully formulated modifications of the proposed module that fully utilizes the STE-based sparse training approach.

6.6 Conclusions

This work proposes Feather, an effective and efficient sparse training module that can be easily applied to pruning frameworks. In particular, as demonstrated by extensive experiments on CIFAR and ImageNet datasets, using both a global and a layer-wise approach, it results to improving the previous SoA results, especially at high pruning ratios. Furthermore, ours method's success indicates the large potential of properly understanding and consequently improving the sparse training dynamics using an STE based approach, that despite its simplicity is shown to be highly effective.

6.7 Appendix

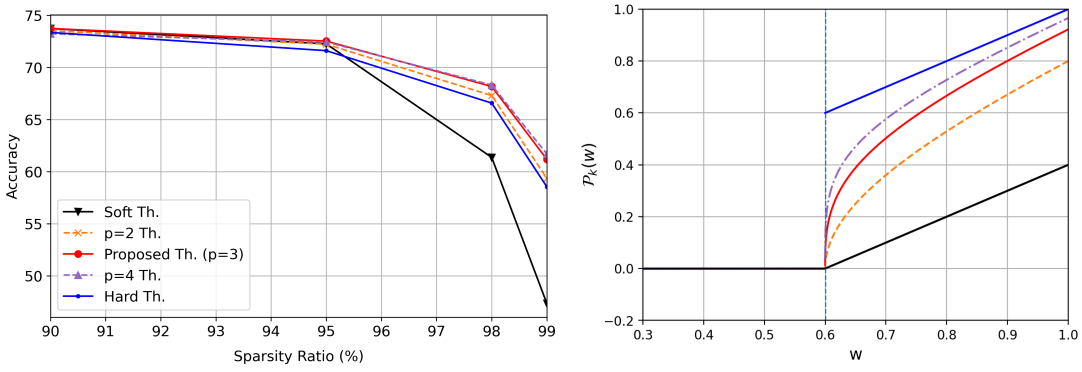
6.7.1 Training Hyperparameters

Dataset	CIFAR-100	ImageNet
Epochs	160	100
Batch Size	128	256
Weight Decay	5e-4	5e-5
Optimizer	SGD	SGD
LR	0.1	0.2
LR-Scheduler	Cosine	Cosine+Warmup
Momentum	0.9	0.9
Label Smoothing	-	0.1

Table 6.3: Training hyperparameters used for all our experiments on CIFAR-100 and ImageNet datasets.

Table 6.3 summarizes the training hyperparameters used for our experiments on CIFAR-100 [47] and ImageNet [13] datasets. The chosen hyperparameters are selected based on standard practices for the particular datasets and are kept the same regardless the network architecture or the target sparsity ratio (in contrast to *e.g.* [50, 86] where the Weight Decay is adjusted among different runs, based on the target sparsity ratio). By adopting commonly used hyperparameters and keeping them unchanged among all our experiments we opted to show that our method is able to achieve SoA results without the need of fine-tuning and complicated training configurations.

6.7.2 Impact of Threshold's p-value



(a) (b)

Figure 6.7.1: A study of the effect of the p value of the proposed family of thresholds on the final sparse model accuracy. Results from ResNet-20 trained on CIFAR-100 (a) and the corresponding thresholds used (b).

The proposed threshold with $p = 3$ is compared with the ones with $p = 2$ and $p = 4$ in Figure 6.7.1. We

observe that $p = 3$ is preferable to $p = 2$ based on the resulting final accuracy while $p = 4$ results to no further improvement (Figure 6.7.1a). Due to that, $p = 3$ is chosen to give a fine balanced threshold between the two extremes, Hard and Soft thresholds respectively, although, as shown, good results are obtainable even with values of p near 3. A reasonable explanation for the slight under-performance using $p = 2$ is that the resulting threshold still leads to a considerable amount of shrinkage (Figure 6.7.1b), thus induces more bias between the thresholded weights and their dense counterparts. Notably, even for $p = 2$ the results are favorable compared to those obtained by using the Hard and Soft Thresholds, further validating the robustness of our family of threshold operators.

6.7.3 Stability of the Sparsity Mask *vs.* Gradient Scaling

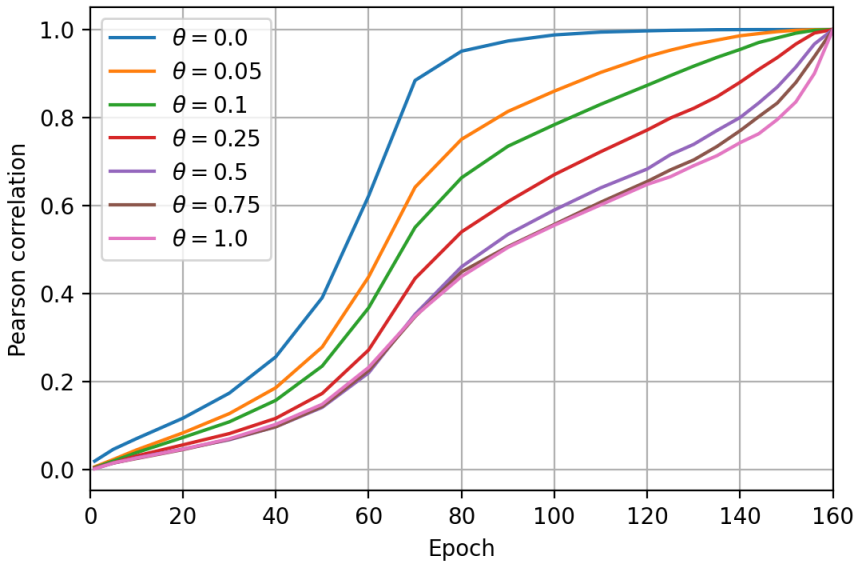
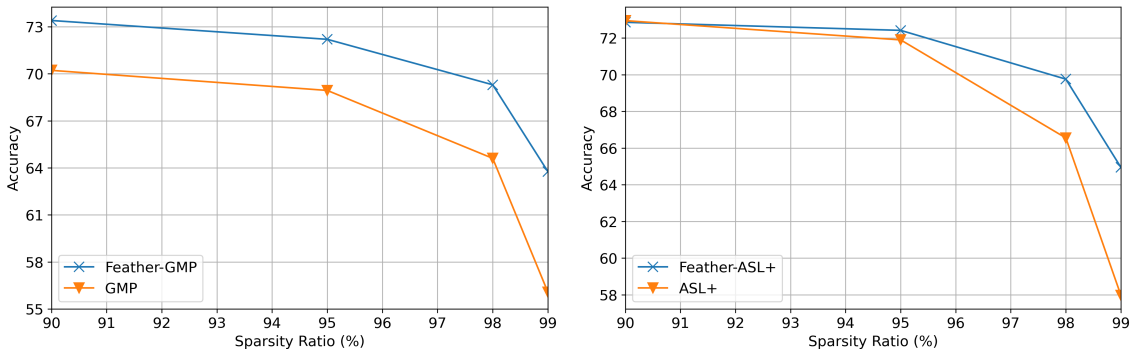


Figure 6.7.2: Plot of Pearson correlation coefficients between the sparsity mask obtained at the end of each epoch and the mask at the end of training, for varying values of the gradient scaling parameter θ. Results from ResNet-20 trained on CIFAR-100.

Figure 6.7.2 empirically validates that the gradient scaling parameter $\theta \in [0, 1]$ influences the stability of the sparsity mask, *i.e.* the mask that indicates which parameters are pruned and which are active during sparse training. Specifically, for each experiment, using a specified (constant) value for θ, the Pearson correlation coefficients between the mask at the end of every training epoch and the final mask, obtained at the end of training, are shown. Experiments with θ near zero result to curves that converge to 1 more rapidly, compared to the ones from experiments with θ close to unity. This indicates that when using θ near zero (or at the extreme case $\theta = 0$) the sparsity mask (and thus the sparsity pattern) is stabilized earlier during the training process, compared to when using larger values of θ. Based on our empirical analysis, the suitable amount of stability for the sparsity mask relates to the sparsity target; The higher the requested final sparsity the more beneficial is to keep the mask more stable (up to a reasonable extent) to avoid destabilizing the highly pruned network. We note that the mask's stability is also studied in [84], where a soft top-k mask is computed by solving a regularized Optimal Transportation problem in order to regulate its stability, although our approach using gradient scaling (combined with the proposed threshold operator) is considerably less computationally expensive while resulting to favorable final accuracies.

6.7.4 Feather Improves Pruning Backbones



(a) (b)

Figure 6.7.3: Feather improves the accuracy of common sparse training backbones: (a) GMP, a uniform layer-wise sparsity pruning backbone (b) the ASL+ framework. Results from ResNet-20 trained on CIFAR-100.

Combining the Feather module with existing backbones results to more accurate networks, as shown in Figure 6.7.3. In 6.7.3a Feather is used to improve the accuracy of GMP [98], a layer-wise magnitude pruning backbone that prunes all layers[2] to the same (uniform) amount of sparsity, gradually increasing the pruning ratio. Our module significantly improves the resulting accuracy when combined with the very simplistic GMP backbone. Furthermore, in 6.7.3b we compare the accuracy of the sparse models obtained with Feather combined with ASL+ [73] and the ones using only ASL+, showing that our module leads to accuracy improvements for the challenging sparsity ratios (95% and above).

6.7.5 MobileNetV1 on ImageNet

Ratio	89%	94.1%
MobileNetV1 (4.21M Params): 71.95		
GMP [98]	61.80	-
STR [50]	62.10	-
ProbMask [97]	65.19	60.10
ST-3 [86]	66.67	61.19
Feather-Global	**68.13**	**63.63**

Table 6.4: Top-1 accuracy of MobileNetV1 on ImageNet.

In Table 6.4 we provide additional experiments on ImageNet [13] using the MobileNetV1 [35] architecture. More specifically, we compare the accuracies obtained by using Feather combined with the global pruning backbone with the ones from GMP [98], STR [50], ProbMask [97] and ST-3 [86] which report results for the 89% and 94.1% sparsity ratios, using the same number of epochs (100) and data augmentation as in our experiments. Our approach surpasses the previous SoA by 1.46% and 2.44% Top-1 accuracy at the 89% and 94.1% sparsity ratios respectively, a result that further validates Feather's effectiveness and generalization abilities on large datasets with different model architectures.

[2]With the exception of the first convolutional layer, which was left dense when using GMP in our experiments due to having a very small number of parameters.

6.7.6 Accuracy *vs.* FLOP Measurements

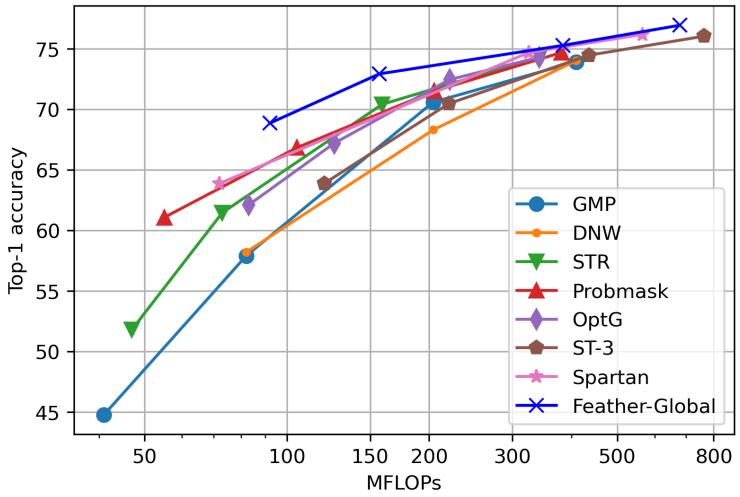


Figure 6.7.4: Top-1 accuracy *vs.* FLOPs of ResNet-50 on ImageNet.

The Feather module, combined with the global pruning backbone, leads to favorable Top-1 accuracy results over the ones from the baselines under similar FLOPs requirements of the sparsified ResNet-50 [31], as shown by the frontier curve in Figure 6.7.4. We note that the per-layer sparsity distribution obtained by the global pruning backbone by default does not prioritize FLOPs reduction, while layer-wise methods such as GMP [98] and STR [50] tend to result to sparse models with minimum FLOPs for a given sparsity ratio, although at a cost of considerable accuracy drops.

While extended analysis on optimizing FLOPs for a given sparsity target is not the scope of this work, to further showcase the efficacy of Feather we experimented with biasing the global pruning backbone towards pruning earlier layers more aggressively, as suggested in [86]. With the FLOPs-biased global pruning backbone, training the ResNet-50 on ImageNet at 99% sparsity, Feather resulted in a model with **67.2% Top-1 accuracy**, now requiring only **42MFLOPs**. Therefore, the superior accuracy of our sparse model was greatly preserved, still achieving the best accuracy (by a 3.3% margin) among the baselines at the 99% ratio, now for considerably fewer FLOP requirements, matching those of GMP (41MFLOPs), the baseline resulting to the fewer FLOPs, although having accuracy more than 20% higher. Having showcased Feather's great potential at obtaining models with superior accuracy and FLOPs, we leave further experimentation (possibly with more sophisticated backbones) as future work.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

In this thesis, we address the problem of compressing deep neural networks, following the weight pruning approach. In order to gain a thorough understanding over the existing compression approaches and eventually introduce novel pruning schemes, after summarizing the required background knowledge on machine learning in Chapter 3, we covered the relevant literature in Chapter 4. Our aim was the development of methods that minimize the memory and computational requirements of DNN models by inducing unstructured sparsity on their weight tensors, while preserving as much of the original accuracy as possible. Key requirement for our methods was to impose minimal computational overheads and to be able to perform the sparsification process within the normal course of training.

To this end, in Chapter 5 we experimented with performing the weight pruning in a layer-wise fashion, modeling the weights per-layer using differentiable probability distribution functions in order to automatically learn the pruning thresholds through the optimization process. Specifically, we showcased that the combined use of the Gaussian and Laplace distributions is adequate for this purpose, resulting into effective non-uniform layer-wise sparsity budgets, as demonstrated by extensive experiments on the CIFAR and ImageNet datasets.

Subsequently, in Chapter 6, motivated by the recent popularity of the Straight-Through Estimator in the context of sparse training, we focused our attention on improving its two main components, *i.e.* the thresholding operator and the gradient manipulation during the forward and backward training steps, respectively. We introduced a family of thresholding operators for the STE, bridging the popular Hard and Soft operators. Moreover, gradient scaling was proposed as a way to control the stability of the pruning procedure. The resulting module, Feather, thoroughly tested on CIFAR and ImageNet, improved the prior state-of-the-art results, by a considerable margin especially at high sparsity regimes.

7.2 Thoughts on Future Work

Towards further improving the schemes proposed in this thesis, a number of areas of future work have been highlighted, which are briefly summarized bellow:

- **Enhancing the sparsity estimation accuracy of our layer-wise pruning approach.** As discussed in Section 5.6, there is still room for reducing the modeling error of the per-layer weight distributions, either by enforcing them to resemble more one of the two modeling types used, or by considering additional distribution types.
- **Gaining a deeper understanding of the STE sparse training dynamics.** Motivated by the success of the Feather module, an intriguing research direction would be to further shed light into the theory behind the application of the STE and its variants for the purpose of DNN sparsification, focusing on mathematically explaining the effectiveness of the proposed thresholding approach (Section 5.3.2).
- **Developing more sophisticated gradient scaling schemes.** Despite effectiveness of the current simplistic approach, an adaptive algorithm for selecting θ dynamically at each training iteration step could be developed, as hinted in Section 6.5.1, with the purpose of providing a fully automated solution for every model and sparsity scenario considered.

Bibliography

[1] Deem Alsaleh and Souad Larabi-Marie-Sainte. "Arabic text classification using convolutional neural network and genetic algorithms". *IEEE Access* 9 (2021), pp. 91670–91685.

[2] Yu Bai, Yu-Xiang Wang, and Edo Liberty. "Proxquant: Quantized neural networks via proximal operators". *arXiv preprint arXiv:1810.00861* (2018).

[3] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. "Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation". *arXiv preprint arXiv:1308.3432* (2013).

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

[5] Davis Blalock et al. "What is the State of Neural Network Pruning?" In: *Proceedings of Machine Learning and Systems (MLSys)*. 2020.

[6] Facundo Bre, Juan M Gimenez, and Víctor D Fachinotti. "Prediction of wind pressure coefficients on building surfaces using artificial neural networks". *Energy and Buildings* 158 (2018), pp. 1429–1441.

[7] Alon Brutzkus et al. "SGD Learns Over-parameterized Networks that Provably Generalize on Linearly Separable Data". In: *Proceedings of the International Conference on Learning Representations (ICLR)*. 2018.

[8] Cristian Buciluǎ, Rich Caruana, and Alexandru Niculescu-Mizil. "Model Compression". In: *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2006.

[9] Peter Bühlmann and Sara Van De Geer. *Statistics for High-Dimensional Data: Methods, Theory and Applications*. Springer Science & Business Media, 2011.

[10] Wenlin Chen et al. "Compressing Neural Networks with the Hashing Trick". In: *Proceedings of the International Conference on Machine Learning (ICML)*. 2015.

[11] Yu Cheng et al. "Model Compression and Acceleration for Deep Neural Networks: The Principles, Progress, and Challenges". *IEEE Signal Processing Magazine* 35.1 (2018), pp. 126–136.

[12] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. "Binaryconnect: Training deep neural networks with binary weights during propagations". In: *Advances in Neural Information Processing Systems*. 2015.

[13] Jia Deng et al. "ImageNet: A large-scale hierarchical image database". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2009.

[14] Lei Deng et al. "Model Compression and Hardware Acceleration for Neural Networks: A Comprehensive Survey". *Proceedings of the IEEE* 108.4 (2020), pp. 485–532.

[15] Arden Dertat. *Applied Deep Learning - Part 4: Convolutional Neural Networks*. URL: https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2.

[16] Caiwen Ding et al. "CirCNN: Accelerating and Compressing Deep Neural Networks Using Block-Circulant Weight Matrices". In: *Proceedings of the IEEE/ACM International Symposium on Microarchitecture*. 2017.

[17] Tim Dockhorn et al. "Demystifying and Generalizing BinaryConnect". In: *Advances in Neural Information Processing Systems*. 2021.

[18] David L Donoho. "De-noising by soft-thresholding". *IEEE Transactions on Information Theory* 41.3 (1995), pp. 613–627.

[19] Jonathan Frankle and Michael Carbin. "The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks". In: *Proceedings of the International Conference on Learning Representations (ICLR)*. 2018.

[20] Jonathan Frankle et al. "Pruning Neural Networks at Initialization: Why Are We Missing the Mark?" In: *Proceedings of the International Conference on Learning Representations (ICLR)*. 2020.

[21] Trevor Gale, Erich Elsen, and Sara Hooker. "The State of Sparsity in Deep Neural Networks". *arXiv preprint arXiv:1902.09574* (2019).

[22] Trevor Gale et al. "Sparse GPU Kernels for Deep Learning". In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 2020.

[23] Amir Gholami et al. "A Survey of Quantization Methods for Efficient Neural Network Inference". In: *Low-Power Computer Vision*. Chapman and Hall/CRC, 2022, pp. 291–326.

[24] Athanasios Glentis Georgoulakis, George Retsinas, and Petros Maragos. "Feather: An Elegant Solution to Effective DNN Sparsification". In: *Proceedings of the British Machine Vision Conference (BMVC)*. 2023.

[25] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT press, 2016.

[26] J Gou et al. "Knowledge Distillation: A Survey". *International Journal of Computer Vision* 129 (2021), pp. 1789–1819.

[27] Katsuyuki Hagiwara. "Bridging between Soft and Hard Thresholding by Scaling". *IEICE Transactions on Information and Systems* 105.9 (2022), pp. 1529–1536.

[28] Song Han, Huizi Mao, and William J Dally. "Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding". *arXiv preprint arXiv:1510.00149* (2015).

[29] Song Han et al. "Learning both weights and connections for efficient neural network". In: *Advances in Neural Information Processing Systems*. 2015.

[30] Babak Hassibi and David Stork. "Second order derivatives for network pruning: Optimal brain surgeon". In: *Advances in Neural Information Processing Systems*. 1992.

[31] Kaiming He et al. "Deep residual learning for image recognition". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.

[32] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. "Distilling the knowledge in a neural network". *arXiv preprint arXiv:1503.02531* (2015).

[33] Geoffrey Hinton et al. "Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups". *IEEE Signal Processing Magazine* 29.6 (2012), pp. 82–97.

[34] Torsten Hoefler et al. "Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks". *The Journal of Machine Learning Research* 22.1 (2021), pp. 10882–11005.

[35] Andrew G Howard et al. "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications". *arXiv preprint arXiv:1704.04861* (2017).

[36] Gao Huang et al. "Densely Connected Convolutional Networks". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017.

[37] Itay Hubara et al. "Binarized Neural Networks". In: *Advances in Neural Information Processing Systems*. 2016.

[38] Forrest N Iandola et al. "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and $<$ 0.5 MB model size". *arXiv preprint arXiv:1602.07360* (2016).

[39] Yani Andrew Ioannou. "Structural Priors in Deep Neural Networks". PhD thesis. University of Cambridge, UK, 2018.

[40] Sergey Ioffe and Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *Proceedings of the International Conference on Machine Learning (ICML)*. 2015.

[41] Berivan Isik, Tsachy Weissman, and Albert No. "An Information-Theoretic Justification for Model Pruning". In: *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*. 2022.

[42] Benoit Jacob et al. "Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018.

[43] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. "Speeding up Convolutional Neural Networks with Low Rank Expansions". *arXiv preprint arXiv:1405.3866* (2014).

[44] Siddhant Jayakumar et al. "Top-KAST: Top-K Always Sparse Training". In: *Advances in Neural Information Processing Systems*. 2020.

[45] Jiwon Jeong. *The Most Intuitive and Easiest Guide for Convolutional Neural Network*. URL: https://towardsdatascience.com/the-most-intuitive-and-easiest-guide-for-convolutional-neural-network-3607be47480.
[46] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *Proceedings of the International Conference on Learning Representations (ICLR)*. 2015.
[47] Alex Krizhevsky. *Learning Multiple Layers of Features from Tiny Images*. Master's thesis. Department of Computer Science, University of Toronto, 2009.
[48] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems*. 2012.
[49] Debasis Kundu. "Discriminating between normal and Laplace distributions". *Advances in Ranking and Selection, Multiple Comparisons, and Reliability: Methodology and Applications* (2005), pp. 65–79.
[50] Aditya Kusupati et al. "Soft Threshold Weight Reparameterization for Learnable Sparsity". In: *Proceedings of the International Conference on Machine Learning (ICML)*. 2020.
[51] Vadim Lebedev and Victor Lempitsky. "Fast ConvNets Using Group-wise Brain Damage". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.
[52] Vadim Lebedev et al. "Speeding-up convolutional neural networks using fine-tuned cp-decomposition". *arXiv preprint arXiv:1412.6553* (2014).
[53] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". *Nature* 521.7553 (2015), pp. 436–444.
[54] Yann LeCun, John Denker, and Sara Solla. "Optimal Brain Damage". In: *Advances in Neural Information Processing Systems*. 1989.
[55] Namhoon Lee, Thalaiyasingam Ajanthan, and Philip HS Torr. "Snip: Single-shot network pruning based on connection sensitivity". *arXiv preprint arXiv:1810.02340* (2018).
[56] Hao Li et al. "Pruning Filters for Efficient ConvNets". *arXiv preprint arXiv:1608.08710* (2016).
[57] Tailin Liang et al. "Pruning and quantization for deep neural network acceleration: A survey". *Neurocomputing* 461 (2021), pp. 370–403.
[58] Haoyang Liu and Rina Foygel Barber. "Between hard and soft thresholding: optimal iterative thresholding algorithms". *Information and Inference: A Journal of the IMA* 9.4 (2020), pp. 899–933.
[59] Zhuang Liu et al. "Learning Efficient Convolutional Networks through Network Slimming". In: *Proceedings of the IEEE International Conference on Computer Vision (CVPR)*. 2017.
[60] Zhuang Liu et al. "Rethinking the Value of Network Pruning". In: *Proceedings of the International Conference on Learning Representations (ICLR)*. 2018.
[61] Christos Louizos, Max Welling, and Diederik P Kingma. "Learning Sparse Neural Networks through L_0 Regularization". In: *Proceedings of the International Conference on Learning Representations (ICLR)*. 2018.
[62] Yongxi Lu et al. "Fully-adaptive Feature Sharing in Multi-Task Networks with Applications in Person Attribute Classification". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017.
[63] Huizi Mao et al. "Exploring the Granularity of Sparsity in Convolutional Neural Networks". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. 2017.
[64] Jerrold E Marsden and Anthony Tromba. *Vector calculus*. Macmillan, 2003.
[65] *MathWorks: Overfitting*. URL: https://www.mathworks.com/discovery/overfitting.html.
[66] Tom M Mitchell. *Machine Learning*. McGraw-Hill Education, 1997.
[67] Hesham Mostafa and Xin Wang. "Parameter Efficient Training of Deep Convolutional Neural Networks by Dynamic Sparse Reparameterization". In: *Proceedings of the International Conference on Machine Learning (ICML)*. 2019.
[68] Alexander Novikov et al. "Tensorizing Neural Networks". In: *Advances in Neural Information Processing Systems*. 2015.
[69] *NVIDIA A100 Tensor Core GPU Architecture*. URL: https://images.nvidia.com/aem-dam/en-zz/Solutions/data-center/nvidia-ampere-architecture-whitepaper.pdf.
[70] *NVIDIA cuSPARSE Library*. URL: https://docs.nvidia.com/cuda/pdf/CUSPARSE_Library.pdf.
[71] *NVIDIA TensorRT*. URL: https://docs.nvidia.com/deeplearning/tensorrt/pdf/TensorRT-Developer-Guide.pdf.

[72] Alex Renda, Jonathan Frankle, and Michael Carbin. "Comparing Rewinding and Fine-tuning in Neural Network Pruning". *arXiv preprint arXiv:2003.02389* (2020).
[73] George Retsinas et al. "Online Weight Pruning Via Adaptive Sparsity Loss". In: *Proceedings of the IEEE International Conference on Image Processing (ICIP)*. 2021.
[74] Roberto Rigamonti et al. "Learning Separable Filters". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2013.
[75] Sebastian Ruder. "An overview of gradient descent optimization algorithms". *arXiv preprint arXiv:1609.04747* (2016).
[76] Sebastian Ruder. *An overview of gradient descent optimization algorithms*. URL: https://www.ruder.io/optimizing-gradient-descent.
[77] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. "Learning representations by back-propagating errors". *Nature* 323.6088 (1986), pp. 533–536.
[78] Sidak Pal Singh and Dan Alistarh. "Woodfisher: Efficient second-order approximation for neural network compression". In: *Advances in Neural Information Processing Systems*. 2020.
[79] Nitish Srivastava et al. "Dropout: A Simple Way to Prevent Neural Networks From Overfitting". *The Journal of Machine Learning Research* 15.1 (2014), pp. 1929–1958.
[80] Saha Sumit. *A comprehensive guide to convolutional neural networks - the eli5 way*. URL: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53.
[81] Wonyong Sung, Sungho Shin, and Kyuyeon Hwang. "Resiliency of Deep Neural Networks under Quantization". *arXiv preprint arXiv:1511.06488* (2015).
[82] Vivienne Sze et al. "Efficient Processing of Deep Neural Networks: A Tutorial and Survey". *Proceedings of the IEEE* 105.12 (2017), pp. 2295–2329.
[83] Cheng Tai et al. "Convolutional neural networks with low-rank regularization". *arXiv preprint arXiv:1511.06067* (2015).
[84] Kai Sheng Tai, Taipeng Tian, and Ser Nam Lim. "Spartan: Differentiable Sparsity via Regularized Transportation". In: *Advances in Neural Information Processing Systems*. 2022.
[85] Hidenori Tanaka et al. "Pruning neural networks without any data by iteratively conserving synaptic flow". In: *Advances in Neural Information Processing Systems*. 2020.
[86] Antoine Vanderschueren and Christophe De Vleeschouwer. "Are Straight-Through gradients and Soft-Thresholding all you need for Sparse Training?" In: *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*. 2023.
[87] Chaoqi Wang, Guodong Zhang, and Roger Grosse. "Picking Winning Tickets Before Training by Preserving Gradient Flow". In: *Proceedings of the International Conference on Learning Representations (ICLR)*. 2020.
[88] Chaoqi Wang et al. "EigenDamage: Structured Pruning in the Kronecker-Factored Eigenbasis". In: *Proceedings of the International Conference on Machine Learning (ICML)*. 2019.
[89] Peisong Wang et al. "Two-Step Quantization for Low-Bit Neural Networks". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018.
[90] Wei Wen et al. "Learning Structured Sparsity in Deep Neural Networks". In: *Advances in Neural Information Processing Systems*. 2016.
[91] Mitchell Wortsman, Ali Farhadi, and Mohammad Rastegari. "Discovering Neural Wirings". In: *Advances in Neural Information Processing Systems*. 2019.
[92] Shixing Yu et al. "Hessian-Aware Pruning and Optimal Neural Implant". In: *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*. 2022.
[93] Xiyu Yu et al. "On Compressing Deep Models by Low Rank and Sparse Decomposition". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017.
[94] Chiyuan Zhang et al. "Understanding deep learning (still) requires rethinking generalization". *Communications of the ACM* 64.3 (2021), pp. 107–115.
[95] Yuxin Zhang et al. "OptG: Optimizing Gradient-driven Criteria in Network Sparsity". *arXiv preprint arXiv:2201.12826* (2022).
[96] Aojun Zhou et al. "Learning N: M Fine-grained Structured Sparse Neural Networks From Scratch". In: *Proceedings of the International Conference on Learning Representations (ICLR)*. 2020.
[97] Xiao Zhou et al. "Effective Sparsification of Neural Networks with Global Sparsity Constraint". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2021.

[98] Michael Zhu and Suyog Gupta. "To prune, or not to prune: exploring the efficacy of pruning for model compression". *arXiv preprint arXiv:1710.01878* (2017).