# *Deep Learning and its Applications in Wireless Big Data Networks Performance*

A dissertation

submitted in fulfillment
of the requirements for the degree
of

Doctor of Philosophy

in
Electrical & Computer Engineering

of School of engineering

at

University of Thessaly


Evangelia Fragkou

May, 2025

# Βαθιά Μάθηση και Εφαρμογές προς την Βελτίωση των Ασυρμάτων Δικτύων Διαχείρισης Μεγάλων Δεδομένων

Διατριβή

η οποία υποβλήθηκε για τη εκπλήρωση των υποχρεώσεων απόκτησης

του

Διδακτορικού Διπλώματος

του τμήματος

Ηλεκτρολόγων Μηχανικών & Μηχανικών Η/Υ

της

Πολυτεχνικής Σχολής

του

Πανεπιστημίου Θεσσαλίας


Ευαγγελία Φράγκου

May, 2025

# *Deep Learning and its Applications in Wireless Big Data Networks Performance*


Ph.D Dissertation

Evangelia Fragkou

## Advisory Commitee

Dimitrios Katsaros, Assoc. Professor, ECE Dept., University of Thessaly

Eleftherios Tsoukalas, Professor, ECE Dept., University of Thessaly

Athanasios Korakis, Professor, ECE Dept., University of Thessaly

## Examination Commitee

Yannis Manolopoulos, Professor Emeritus, Dept. of Informatics, Aristotle University of Thessaloniki

Dimitrios Rafailidis, Assoc. Professor, ECE Dept., University of Thessaly

Spyridon-Gerasimos Lalis, Professor, ECE Dept., University of Thessaly

Dimitrios Bargiotas, Professor, ECE Dept., University of Thessaly

May, 2025

# *Βαθιά Μάθηση και Εφαρμογές προς την Βελτίωση των Επιδόσεων Δικτύων Διαχείρισης Μεγάλων Δεδομένων*


Διδακτορική Διατριβή

Ευαγγελία Φράγκου

## Συμβουλευτική Επιτροπή

Δημήτριος Κατσαρός, Αναπλ. Καθηγητής, Τμ. ΗΜΜΥ, Πανεπιστήμιο Θεσσαλίας

Ελευθέριος Τσουκαλάς, Καθηγητής, Τμ. ΗΜΜΥ, Πανεπιστήμιο Θεσσαλίας

Αθανάσιος Κοράκης, Καθηγητής, Τμ. ΗΜΜΥ, Πανεπιστήμιο Θεσσαλίας

## Εξεταστική Επιτροπή

Ιωάννης Μανωλόπουλος, Ομότ. Καθηγητής, Τμ. Πληροφορικής, Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης

Δημήτριος Ραφαηλίδης, Αναπλ. Καθηγητής, Τμ. ΗΜΜΥ, Πανεπιστήμιο Θεσσαλίας

Σπυρίδων-Γεράσιμος Λάλης, Καθηγητής, Τμ. ΗΜΜΥ, Πανεπιστήμιο Θεσσαλίας

Δημήτριος Μπαργιώτας, Καθηγητής, Τμ. ΗΜΜΥ, Πανεπιστήμιο Θεσσαλίας

May, 2025

# FUNDING

This research was largely funded by the Hellenic Foundation for Research and Innovation (HFRI) under the 3rd Call for HFRI Ph.D Fellowships (Fellowship Number: 5631).
Duration of the Project: 20/04/2022 – 19/03/2024.

# ΧΡΗΜΑΤΟΔΟΤΗΣΗ

Το μεγαλύτερο μέρος της παρούσας διδακτορικής διατριβής χρηματοδοτήθηκε από το Ελληνικό Ίδρυμα Ερευνας & Καινοτομίας (ΕΛ.ΙΔ.Ε.Κ), στο πλαίσιο της «3ης Προκήρυξης ΕΛ.ΙΔ.Ε.Κ. για Υποψήφιους/ες Διδάκτορες» (Αριθμός Υποτροφίας: 5631).
Διάρκεια υποτροφίας: 20/04/2022 – 19/03/2024.

# Acknowledgements

First and foremost, I owe special thanks to *Hellenic Foundation for Research & Innovation (H.F.R.I.)* for the financial support throughout my doctoral studies under the 3rd Call for H.F.R.I. Ph.D Fellowships (Grant Number: 5631). Their contribution has been instrumental in enabling the successful completion of this research.

Furthermore, I would like to express my sincere and profound gratitude to my supervisor, Assoc. Professor Mr. *Dimitrios Katsaros*, for his unwavering guidance, steadfast support, and insightful academic feedback throughout the duration of my doctoral research. His intellectual acumen and deep scientific expertise have been instrumental in shaping and completing this dissertation.

I am also deeply thankful to the members of the examination committee, Professor Emeritus Mr.Yannis Manolopoulos, Professor Mr.Eleftherios Tsoukalas, Professor Mr.Athanasios Korakis, Professor Mr.Spyridon-Gerasimos Lalis, Professor Mr.Dimitrios Bargiotas and Assoc. Professor Mr.Dimitrios Rafailidis for their valuable time and constructive suggestions, all of which substantially enriched quality of this work.

I also wish to extend my sincere thanks to all the collaborators who contributed to my work and supported the realization of my research objectives.

Moreover, there are no words that can truly express the depth of my gratitude to my parents, Katerina & Spyros, who have supported me unconditionally and tangibly at every step of my academic journey. Their encouragement, love, and unwavering belief in my potential have been a constant source of strength and inspiration throughout this endeavor. I am also deeply grateful to my sister Maria and to my wider family; in particular to my grandmother Maria, and my cousin Evelina, for their tangible support throughout these years.

Lastly, I wish to extend my heartfelt thanks to my partner, Aris. I am especially grateful not only for his encouragement, but also for the patience, understanding and endurance he has shown, even in moments of great difficulty.

# Ευχαριστιες

Αρχικά, θα ήθελα να εκφράσω τις ιδιαίτερες ευχαριστίες μου στο *Ελληνικό Ίδρυμα Έρευνας & Καινοτομίας (ΕΛ.ΙΔ.Ε.Κ.)* για την οικονομική υποστήριξη κατά τη διάρκεια των διδακτορικών μου σπουδών στο πλαίσιο της 3ης Προκήρυξης για Υποτροφίες Διδακτορικού ΕΛ.ΙΔ.Ε.Κ. (Αριθμός Υποτροφίας: 5631). Η συμβολή τους ήταν καθοριστική για την απρόσκοπτη έκβαση και επιτυχή ολοκλήρωση της δεδομένης έρευνας.

Εν συνεχεία, θα ήθελα να εκφράσω την ειλικρινή και βαθιά μου ευγνωμοσύνη προς τον επιβλέποντα καθηγητή μου, Αναπλ. Καθηγητή κ.Δημήτριο Κατσαρό, για την πολύτιμη καθοδήγησή του, την αμέριστη υποστήριξή του και τις εύστοχες επιστημονικές παρατηρήσεις του καθ' όλη τη διάρκεια της διδακτορικής μου έρευνας. Η ευφυΐα του και η βαθιά του επιστημονική κατάρτιση αποτέλεσαν καταλυτικό παράγοντα στην εξέλιξη και την ολοκλήρωση της παρούσας διατριβής.

Επίσης, θα ήθελα να εκφράσω τις θερμές μου ευχαριστίες στα μέλη της εξεταστικής επιτροπής, και συγκεκριμένα τον Ομότιμο Καθηγητή κ. Ιωάννη Μανωλόπουλο, τον Καθηγητή κ. Ελευθέριο Τσουκαλά, τον Καθηγητή κ. Αθανάσιο Κοράκη, τον Καθηγητή κ. Σπυρίδωνα-Γεράσιμο Λάλη, τον Καθηγητή κ.Δημήτριο Μπαργιώτα και τον Αναπλ. Καθηγητή κ. Δημήτριο Ραφαηλίδη για τον χρόνο τους και τις εύστοχες παρατηρήσεις τους, που εμπλούτισαν σημαντικά αυτή τη διατριβή.

Θα ήθελα, επίσης, να ευχαριστήσω θερμά όλους τους συνεργάτες μου, που συνέβαλαν στην εκπόνηση της παρούσας έρευνας και με βοήθησαν να ολοκληρώσω τους ερευνητικούς μου στόχους.

Επιπλέον, δεν υπάρχουν λόγια που να μπορούν να εκφράσουν πραγματικά το βάθος της ευγνωμοσύνης μου στους γονείς μου, *Κατερίνα & Σπύρο*, που με στήριξαν έμπρακτα και άνευ όρων, σε κάθε βήμα της ακαδημαϊκής διαδρομής μου. Η συνεχής ενθάρρυνση, η αγάπη και η πίστη τους στις δυνατότητές μου, ήταν μια αστείρευτη πηγή δύναμης και έμπνευσης σε όλη αυτή την προσπάθεια. Ένα μεγάλο ευχαριστώ οφείλω και στην αδερφή μου, Μαρία αλλά και στην ευρύτερη οικογένειά μου και συγκεκριμένα στη γιαγιά μου, Μαρία και στην ξαδέρφη μου, Εβελίνα, για την έμπρακτη στήριξή τους όλα αυτά τα χρόνια.

Τέλος, οφείλω ένα μεγάλο ευχαριστώ στο σύντροφό μου, Άρη, όχι μόνο για τη στήριξή του, αλλά και για την υπομονή, την κατανόηση και την αντοχή που επέδειξε, ακόμη και στις πιο δύσκολες στιγμές κατά τη διάρκεια αυτής της διαδρομής.

Dedicated to my beloved parents,

*Katerina & Spyros*
and
to my partner,
*Aris*,

who stood tirelessly by my side throughout
this journey in pursuit of knowledge

Η διατριβή αφιερώνεται στους γονείς μου

*Κατερίνα & Σπύρο*

και

στο σύντροφό μου,

*Άρη*,

που με στήριζαν αδιάλειπτα σε αυτό

το ταξίδι προς τη γνώση

# Abstract & Main Contributions

The success of Internet of Things (IoT) has led to a shift in data processing towards resource-limited, typically wireless, edge devices. This thesis examines the way cooperative neural network training among the aforementioned devices can exist, focusing not only on maintaining data autonomy, but also data security, without completely depleting devices' resources. To achieve this goal, methods for reducing neural network parameters and training time are presented, such as dynamic pruning during the training phase, based on criteria inspired by network theory (e.g. scale free networks). Moreover, we introduce algorithms leveraging pre-trained neural network weights (transfer learning), accelerating convergence and improving model performance. However, although power consumption is reduced due to decreased training time and memory requirements, the amount of data available on such a device is typically insufficient for the model to converge without encountering the phenomenon of *overfitting*. Even in scenarios where prior knowledge is utilized, the model may still train ineffectively due to *heterogeneity* between the prior and newly acquired knowledge. Consequently, *Decentralized (Distributed) Federated Learning* method gained immense popularity. In this approach, devices exchange parameters rather than raw data, to train a shared global model collaboratively.

By This way, the workload is distributed across devices, each training locally on its raw data and periodically sending updated weights for synchronization into aggregated weights. To prevent flooding or energy depletion—issues that can arise from all-to-all communication protocols—new methods are introduced to limit message exchange between nodes with similar data due to geographic proximity. However, in order to enable such communication, especially in wireless, resource-constrained environments (e.g., IoT), it is crucial to establish a suitable infrastructure (i.e., the minimal number of nodes capable of transferring all information across the network), enabling the adoption of the aforementioned learning strategy. Thus, new algorithms are introduced for constructing multi-layer backbone networks, along with new centrality measures for nodes, primarily aiming to identify the most "popular" nodes that lead to network formation using the fewest possible nodes (connected dominating sets - CDS).

To sum up, the contributions of this dissertation are as follows:

- The issue of reducing both the parameters and training time of a neural network was explored with the goal of deploying it on ultra-low-resource devices (e.g., memory $<$ 1MB or even KB, processing power $<$ 1mW, etc.), using network theory principles. The developed method can be combined with any other model simplification technique. The proposed approach outperformed competing methods. Furthermore, it was observed that the classical version of a knowledge transfer algorithm (Inductive Transfer Learning) is not well suited for convolutional neural networks (CNNs), as their final layers are responsible solely for classification. This results in no modification to the core of the network, which would otherwise improve prediction classification. The proposed method proved more

effective in terms of prediction accuracy, while only slightly lagging in energy efficiency compared to its competitor.

- The issue of constructing compact and robust multi-layer backbones was investigated, using the Internet of Battlefield Things (IoBT) as a case study. IoBT represents a more demanding and large-scale subset of traditional IoT. Algorithms were developed to construct multi-layer CDS structures, and it was proven that finding such nodes is an NP-Complete problem, making this work stand out among its competitors. To ensure optimal information dissemination through network nodes, a new centrality metric—sink group betweenness centrality—was introduced. This metric identifies nodes that can "monitor" paths leading to subsets of nodes and generalizes the traditional betweenness centrality concept. This contribution serves as a foundational step toward developing efficient algorithms to compute such metrics and analyze their distribution in real-world networks.

- Finally, new algorithms were implemented for clustering nodes in a wireless network, so that parameter (e.g., neural weights or gradients) averaging does not occur through all-to-all communication (as in classical Decentralized Federated Learning), but instead focuses on localized geographic areas. A methodology was developed whereby not all nodes in a geographic neighborhood participate in the federation—instead, nodes with similar data to those already participating are excluded. This is achieved using the Fourier theorem to detect similarity in time-series data. This approach reduces communication, conserves energy, and relieves wireless network congestion, achieving very promising results compared to existing centralized methods.

# Σύνοψη & Κυριοτερες Συνεισφορες

Δεδομένης της επιτυχίας του Διαδικτύου των πραγμάτων (Internet of Things), η διαδικασία επεξεργασίας των δεδομένων μετατοπίστηκε στις συσκευές αιχμής, οι οποίες είναι συνήθως ασύρματα συνδεδεμένες με ελάχιστους πόρους. Στην παρούσα διατριβή, μελετήθηκε το πρόβλημα του πως δύνανται οι προαναφερθείσες συσκευές να εκτελούν συνεργατική εκπαίδευση των νευρωνικών τους δικτύων, χωρίς εξάντληση των πόρων τους, με στόχο τόσο την διατήρηση της αυτονομίας, όσο και την ασφάλεια των δεδομένων που εκείνες παράγουν.

Προκειμένου να επιτευχθεί ο παραπάνω στόχος, αρχικά, μέθοδοι ελάττωσης των συνδέσεων ή παραμέτρων των νευρωνικών δικτύων αλλά και του απαιτούμενου χρόνου εκπαίδευσής του παρουσιάζονται, όπως για παράδειγμα η τεχνική του δυναμικού κλαδέματος (dynamic pruning) κατά τη διάρκεια της εκπαίδευσης του δικτύου, με κριτήρια εμπνευσμένα από τη θεωρία των δικτύων, πχ. τα scale-free δίκτυα. Επίσης, εισάγονται αλγόριθμοι οι οποίοι, αξιοποιούν την προηγούμενη γνώση ή αλλιώς τα προεκπαιδευμένα βάρη του νευρωνικού δικτύου που απέκτησε σε προηγούμενη εκπαίδευσή του (Transfer Learning), τα οποία κατά συνέπεια, συντελούν στην ταχύτερη σύγκλιση του μοντέλου και στην καλύτερη επίδοση του. Αν και το πρόβλημα της κατανάλωσης ισχύος ενός μοντέλου μειώνεται, καθώς μειώνεται τόσο ο χρόνος εκπαίδευσης, όσο και οι απαιτήσεις στη μνήμη, το πλήθος δεδομένων που έχει στη διάθεση της μια τέτοια συσκευή δεν επαρκεί, ώστε να συγκλίνει το μοντέλο χωρίς να εμφανίσει το φαινόμενο της υπερπροσαρμογής (overfitting).

Ακόμη και σε περιπτώσεις όπου πρότερη γνώση χρησιμοποιείται, υπάρχει πιθανότητα πάλι το μοντέλο να μην εκπαιδευτεί αποτελεσματικά λόγω ετερογένειας μεταξύ της προϋ-πάρχουσας γνώσης και της νεο-αποκτηθείσας. Φυσικό επακόλουθο είναι η εδραίωση της αποκεντρωμένης συνεργατικής ή αλλιώς ομόσπονδης μάθησης (Decentralized Federated Learning) μεταξύ των συσκευών, κατά την οποία οι συσκευές επικοινωνούν μεταξύ τους για να αποστείλουν παραμέτρους, όχι αυτούσια τα δεδομένα που παράγουν, ώστε όλες μαζί να εκπαιδεύουν ένα καθολικό μοντέλο. Με αυτό τον τρόπο μοιράζεται ο φόρτος εργασίας κάθε συσκευής, αφού η κάθε μία εκτελεί τοπικά εκπαίδευση με βάση τα δικά της ακατέργαστα δεδομένα και έπειτα από ορισμένους κύκλους εκπαίδευσης, όλες μαζί στέλνουν τα ανανεωμένα βάρη τους και συγχρονίζονται στα νέα συναθροισμένα βάρη. Προκειμένου να προληφθεί το φαινόμενο της πλημμύρας (flooding) ή της εξάντλησης ενέργειας, που το πρωτόκολλο επικοινωνίας όλοι-με-όλους μπορεί να προκαλέσει, παρουσιάζονται νέες μέθοδοι περιορισμού της ανταλλαγής μηνυμάτων μεταξύ κόμβων που περιέχουν παρόμοια δεδομένα, λόγω τοποθεσίας.

Ωστόσο, για την επίτευξη μίας τέτοιας επικοινωνίας, πρακτικά, σε ασύρματα, χωρίς ιδιαίτερους πόρους, περιβάλλοντα (πχ. IoT), είναι βαρύνουσας σημασίας να δημιουργηθεί η κατάλληλη υποδομή (δηλ. ελάχιστοι δυνατοί κόμβοι που φέρνουν εις πέρας την μεταφορά όλης της πληροφορίας στο δίκτυο), ώστε να μπορέσει να υιοθετηθεί η παραπάνω στρατηγική μάθησης. Παρουσιάζονται, δηλαδή, νέοι αλγόριθμοι κατασκευής πολυε-πίπεδων δικτύων (multi-layer Backbones), καθώς και νέες μετρήσεις κεντρικότητας των κόμβων (centrality measures), με κύριο στόχο την εύρεση των πιο δημοφιλών από αυτούς που οδηγούν στην κατασκευή δικτύων με τους ελάχιστους δυνατούς

κόμβους (connected dominating sets - CDS).

Επομένως, εν συντομία, οι συνεισφορές της παρούσας διατριβής είναι οι ακόλουθες:

- Μελετήθηκε το θέμα της ελάττωσης τόσο των παραμέτρων όσο και του χρόνου εκπαίδευσης ενός νευρωνικού δικτύου με στόχο την εφαρμογή του σε συσκευές με ελάχιστους πόρους (πχ. μνήμη $<$ 1 MB ή ακόμη και KB, ισχύς επεξεργαστή $<$ 1mW, Κτλ.), ακολουθώντας τις θεωρίες των δικτύων. Η μέθοδος που αναπτύχθηκε, μπορεί να εφαρμοστεί συνδιαστικά με οποιαδήποτε άλλη μέθοδο ελάττωσης των απαιτήσεων ενός μοντέλου. Η προταθείσα τεχνική αποδείχθηκε αποτελεσματικότερη σε σχεση με τους αντίστοιχους ανταγωνιστές της. Παράλληλα, εντοπίσαμε πως η κλασική εκδοχή ενός αλγορίθμου που βασίζεται στη μεταφορά γνώσης (Transfer Learning) δεν ενδείκνυται, σε ό,τι αφορά στα συνελικτικά δίκτυα (CNN), διότι τα τελευταία layers είναι υπεύθυνα μόνο για την ταξινόμηση των δεδομένων (classification), με αποτέλεσμα να μην αναδιαμορφώνεται ο κεντρικός κορμός του δικτύου που θα οδηγούσε σε βελτιστοποίηση της ταξινόμησης των προβλεψεων. Η προτεινόμενη μέθοδος αποδείχθηκε αποτελεσματικότερη συγκριτικά με την ακρίβεια των προβλέψεων, υστερώντας ελάχιστα στην απαιτούμενη ενέργεια σε σχέση με την μέθοδο-ανταγωνιστή.

- Διερευνήθηκε το ζήτημα της δημιουργίας μικρών και ανθεκτικών πολυ-επίπεδων δικτύων (Multilayer backbones), εξετάζοντας ως παράδειγμα την περίπτωση του Δικτύου των πραγμάτων σε κατάσταση μάχης (Internet of Battlefield Things - IoBT), που αποτελεί υποκατηγορία του κλασικού IoT, αλλά σε πολύ μεγαλύτερη κλίμακα και με αυστηρότερες απαιτήσεις στιβαρότητας και μείωσης καθυστέρησης. Δημιουργήσαμε αλγορίθμους κατασκευής πολυ-επίπεδων CDS, καθώς και αποδείξαμε ότι η εύρεση αυτών των κόμβων ανήκει στην κατηγορία των NP-Complete προβλημάτων, κάνοντας τη συγκεκριμένη εργασία να ξεχωρίζει ανάμεσα στους ανταγωνιστές της. Προκειμένου να διασφαλιστεί η καλύτερη διάχυση των πληροφοριών μέσω των κόμβων του δικτύου στο κομμάτι του clustering, εισαγάγαμε ένα νέο μέτρο για τον υπολογισμό της κεντρικότητας των κόμβων, το λεγόμενο sink group betweenness centrality, για τον εντοπισμό αυτών των κόμβων σε ένα δίκτυο που μπορεί να «παρακολουθήσει» τα μονοπάτια που οδηγούν σε ένα σύνολο υποσυνόλων κόμβων. Γενικεύει την παραδοσιακή έννοια της betweenness centrality μεταξύ των κόμβων και αποτελεί το πρώτο βήμα για την ανάπτυξη αποτελεσματικών αλγορίθμων για τον υπολογισμό αυτών των μέτρων και την ανάλυση της κατανομής τους σε πραγματικά δίκτυα.

- Τέλος, υλοποιήθηκαν νέοι αλγόριθμοι για ομαδοποίηση κόμβων σε ένα ασύρματο δίκτυο, ώστε το averaging των ανταλλασσόμενων παραμέτρων (neural weights ή gradients) να μην λαμβάνει χώρα με έναν τρόπο επικοινωνίας όλοι-με-όλους (κλασικό Decentralized/Distributed Federated Learning), αλλά να εστιάζει τοπικά σε τοπικό, γεωγραφικό χώρο (Clustering). Επομένως, αναπτύχθηκε μεθοδολογία, ώστε στις τοπικές γεωγραφικές γειτονιές να μην συμμετέχουν όλοι οι κόμβοι στην ομοσπονδία, αλλά να αποκλείονται όσοι κόμβοι έχουν παρόμοια δεδομένα με κάποιους οι οποίοι συμμετέχουν ήδη σ'αυτήν, κάνοντας χρήση του θεωρήματος Fourier για την αναγνώριση όμοιων δεδομένων σε χρονοσειρές. Με τον τρόπο αυτό γίνεται ελάττωση επικοινωνίας και άρα εξοικονόμηση ενέργειας, καθώς και ελάφρυνση του ασύρματου δικτύου, πετυχαίνοντας πολύ ενθαρρυντικά αποτελέσματα, σχετικά με τις Centralized προϋπάρχουσες μεθόδους.

# Table of Contents

**FUNDING** i

**ΧΡΗΜΑΤΟΔΟΤΗΣΗ** iii

**Acknowledgements** v

**Ευχαριστίες** vii

**Abstract & Main Contributions** xiii

**Σύνοψη & Κυριότερες Συνεισφορές** xv

**Table of Contents** xvii

**List of Tables** xxi

**List of Figures** xxiii

**1 Introduction** 1

1.1 Introduction . . . . . 1

1.2 Motivation & Contribution . . . . . 4

**2 Introduction to the wide area of Deep Learning for wirelessly connected resource-limited devices.** 13

2.1 Accelerating Neural Networks' deployment . . . . . 13

2.1.1 Pruning/sparsification . . . . . 16

2.1.1.1 Background on network science concepts . . . . . 17

2.1.2 Utilizing prior knowledge . . . . . 18

2.2 Network infrastructure . . . . . 20

2.2.1 Measuring the *importance* of nodes/paths . . . . . 20

2.2.1.1 Definition of Betweenness Centrality measures. . . . . 20

2.2.2 Backbone construction: The Internet of Battlefield Things (IoBT) paradigm 22

2.3 The Rising of the Federated Learning Methodology . . . . . 24
2.3.1 Problem Formulation . . . . . 25
2.3.2 The distributed approach of Federated Learning . . . . . 25

**3 Building Network Architecture for Learning 27**
3.1 Multilayer Backbones: The paradigm of IoBT . . . . . 27
3.1.1 Introduction . . . . . 27
3.1.2 Motivation and Contributions . . . . . 28
3.1.3 Formulation of the Problem . . . . . 29
3.1.4 Proposed Heuristic Distributed Algorithms . . . . . 32
3.1.4.1 Distributed $CDS$ in a Multilayer Network . . . . . 32
3.1.4.2 Centralized $CDS$ in Multilayer Networks . . . . . 34
3.1.5 Computation and Communication Complexities . . . . . 35
3.1.5.1 Complexities of $CCDS$ . . . . . 35
3.1.5.2 Complexities of $FAST\text{-}CMDSM$ . . . . . 36
3.1.6 Performance Evaluation . . . . . 36
3.1.7 Experimental Evaluation . . . . . 38
3.1.7.1 Impact of Topology Density . . . . . 38
3.1.7.2 Impact of Network Diameter . . . . . 42
3.1.7.3 Impact of the Number of Layers . . . . . 45
3.1.7.4 Impact of Layer's Size Increase . . . . . 46
3.1.7.5 Energy Awareness of the Competing Algorithms . . . . . 48
3.1.7.6 Summary of Results . . . . . 50
3.1.8 Conclusions . . . . . 50
3.2 Introducing the concept of Sink Group Node Betweenness Centrality . . . . . 53
3.2.1 Introduction . . . . . 53
3.2.2 The sink group Betweenness Centrality . . . . . 54
3.2.2.1 $SGBC$ versus its closest relatives. . . . . 56
3.2.3 Exemplifying the merits of $SGBC$ . . . . . 57
3.2.3.1 Calculation of $SGBC$ . . . . . 58
3.2.4 $SGBC$ with node weights . . . . . 60
3.2.5 Sink group edge Betweenness Centrality . . . . . 61
3.2.6 Applications of $SGBC$ . . . . . 62
3.2.6.1 $SGBC$ and influence minimization . . . . . 62
3.2.6.2 $SGBC$ and virtual currencies networks . . . . . 62
3.2.6.3 $SGBC$ and community finding . . . . . 62
3.2.7 Conclusions . . . . . 62

**4 Model Reduction 65**

4.1 Neural Topology Sparsification Prospects, derived from Network Science disciplines 65
4.1.1 Introduction . . . 65
4.1.2 The topology-based proposed techniques . . . 66
4.1.2.1 Scale-Free to Scale-Free with random rewiring (SF2SFrand). . 66
4.1.2.2 Scale-Free to Scale-Free using preferential attachment (SF2SFba). 67
4.1.2.3 Scale-Free to Scale-Free (5 strongest nodes) (SF2SF5). . . . . 68
4.1.2.4 Scale-Free to Small-World ( SF2SW) . . . 69
4.1.2.5 Small-World to Small-World (SW2SW). . . . . 70
4.1.3 Experimental evaluation . . . 72
4.1.3.1 Evaluation settings . . . 72
4.1.3.2 Evaluation results . . . 75
4.1.3.3 Summary of experimental results . . . 95
4.1.4 Experiments using dynamic connection removal protocols . . . 96
4.1.4.1 Exponential Decay . . . 98
4.1.4.2 Linear Decreasing Variation . . . 98
4.1.4.3 Oscillating Variation . . . 99
4.1.5 Experimental Evaluation using dynamic connection removal protocols . 99
4.1.5.1 Evaluation Settings using dynamic connection removal protocols 100
4.1.5.2 Evaluation Results using dynamic connection removal protocols 100
4.1.6 Conclusions . . . 104
4.2 Transfer Learning . . . 106
4.2.1 Introduction . . . 106
4.2.2 The family of $F_xC_y$Techniques . . . 106
4.2.3 Experimental evaluation . . . 107
4.2.3.1 Evaluation settings . . . 108
4.2.3.2 Evaluation results . . . 109
4.2.4 Summary of experiments . . . 113
4.2.5 Conclusions . . . 113

**5 Distributed Learning 115**
5.1 Distributed Federated Deep Learning in Clustered IoT Wireless Networks with Data Similarity-based Client Participation . . . 115
5.1.1 Introduction . . . 115
5.1.2 Problem formulation in wireless settings . . . 116
5.1.3 Data Similarity-Based Hierarchical Distributed Federated Learning (DISCREETER) . . . 117
5.1.3.1 Establishing the hierarchical IoT topology . . . 118
5.1.3.2 Data similarity-based client selection in FL . . . 119
5.1.3.3 The federated averaging procedure . . . 120

5.1.3.4 Complexities and Convergence . . . . . 121
5.1.4 Experimental Evaluation . . . . . 121
5.1.4.1 Experimental Setup . . . . . 121
5.1.4.2 Competitors . . . . . 123
5.1.4.3 Evaluation results . . . . . 123
5.1.5 Conclusions . . . . . 124

**6 Conclusions** **133**
6.1 Conclusion . . . . . 133
6.2 Exploring future avenues . . . . . 136
Bibliography . . . . . 139

# List of Tables


**Table** **Page**

2.1 Resource-limited devices/boards aiming to run deep learning tasks. . . . . 15

3.1 Competitor characteristics. . . . . 37
3.2 Experimentation parameters values. . . . . 38
3.3 Results' summary. . . . . 50

4.1 Characteristics, regarding some of the larger datasets, availed of. . . . . 70
4.2 Characteristics of smaller datasets. . . . . 73
4.3 List of hyperparameters, used. . . . . 76
4.4 Memory Footprint in MegaBytes(MB), between the Dense Network (MLP), the Sparse Network using SET algorithm and our proposed algorithms, which are SF2SFrand, SF2SFba abd SF2SF(5), SF2SW and SW2SW. . . . . 90
4.5 Measures of MLP (FC). . . . . 90
4.6 Measures of SET. . . . . 90
4.7 Measures of SF2SFrand. . . . . 91
4.8 Measures of SF2SFba. . . . . 91
4.9 Measures of SF2SF-5. . . . . 91
4.10 Measures of SF2SW. . . . . 92
4.11 Measures of SW2SW. . . . . 92
4.12 Measures of experiments with AVG weighted algorithms on Lung dataset. . . . . 92
4.13 Comparison between the old and new implementation of all algorithms. . . . . 93
4.14 Measures of $\zeta$-parametarized SF2SFrand. . . . . 93
4.15 Measures of MLP(FC), SET, SF2SFrand, SF2SFba, SF2SF(5) algorithms, respectively, in Fashion-Mnist (see Table 4.1) dataset. . . . . 94
4.16 Summary of experiments. . . . . 95
4.17 Comparing the parameter reduction between the Dense MLP and the Sparse methods. 101
4.18 Dataset characteristics, used in this experiment. . . . . 109
4.19 Trainable Layers, used in every model architecture. . . . . 110
4.20 Summary of experiments. . . . . 113

5.1 Statistics of the representative node given the final data chunk, after the global averaging, in $10_{th}$ and $20_{th}$ epochs of local training, respectively. . . . . 125

# List of Figures

**Figure** **Page**


1.1 Overview of the Internet of Things (IoT), highlighting the role of connected devices. 2
1.2 Classification of devices according to their resources vs specifications by Rajapakse et al. [1]. . . . 3
1.3 An illustration of the parameter exchange architecture. The representative node aggregates the weights from each selected participating node, which sends its local parameters (red dashed lines). After global aggregation, the aforementioned node broadcasts back the updated weights to the associated nodes in order to continue local training. . . . 8
1.4 Desideratum of the current thesis. . . . 9

2.1 An illustration of pruning either neurons or synapses of a deep neural network. . . 16
2.2 Structured vs Unstructured pruning. . . . 17
2.3 (a) A 20-node regular network (with 3 neighbors at each side of every vertex), (b) a 20-node small-world network (with initially 3 neighbors at each side of every vertex, and with 0.075 rewiring probability), (c) a 20-node random network (with average node degree 6) and (d) a 20-node scale-free network. Graphs were generated with Pajek. . . . 18
2.4 (a) Conventional ML/DL, (b)Transfer Learning mechanism. . . . 20
2.5 A sample multilayer IoBT ad hoc network. . . . 23
2.6 An illustration of the three main categories of FL architectures by Beltran et al. [2]. The numbers depicted, in the figure signify the training lifecycle. . . . 26

3.1 Impact of topology density (Medium skewness). . . . 39
3.2 Impact of topology density (Low skewness). . . . 40
3.3 Impact of topology density (High skewness). . . . 41
3.4 Impact of topology density (CDS size aggregated over all layers). . . . 41
3.5 Impact of network diameter (Medium skewness). . . . 43
3.6 Impact of network diameter (Low skewness). . . . 43
3.7 Impact of network diameter (High skewness). . . . 44

3.8 Impact of network diameter (CDS size aggregated over all layers). . . . . 44
3.9 Impact of number of layers (Medium skewness). . . . . 45
3.10 Impact of number of layers (Low skewness). . . . . 46
3.11 Impact of number of layers (High skewness). . . . . 47
3.12 Impact of number of layers (CDS size aggregated over all layers). . . . . 47
3.13 Impact of increasing layer cardinality (Medium skewness). . . . . 49
3.14 Impact of increasing layer cardinality (Low skewness). . . . . 49
3.15 Impact of increasing layer cardinality (High skewness). . . . . 50
3.16 Energy awareness of the competitors (Medium skewness). . . . . 51
3.17 Energy awareness of the competitors (Low skewness). . . . . 51
3.18 Energy awareness of the competitors (High skewness). . . . . 52
3.19 Illustration of Sink Group Betweenness Centrality. . . . . 57
3.20 Impact of grouping on Sink Group Betweenness Centrality. . . . . 59

4.1 Lung dataset overall results, using ReLU activation function. . . . . 79
4.2 Lung dataset overall results, using FReLU activation function. . . . . 80
4.3 Lung Discrete dataset overall results, using ReLU activation function. . . . . 81
4.4 Lung Discrete dataset overall results, using FReLU activation function. . . . . 81
4.5 TOX_171 dataset overall results, using ReLU activation function. . . . . 82
4.6 TOX_171 dataset overall results, using FReLU activation function. . . . . 82
4.7 CLL_SUB_111 dataset overall results, using ReLU activation function. . . . . 83
4.8 CLL_SUB_111 dataset overall results, using FReLU activation function. . . . . 83
4.9 COIL20 dataset overall results, using ReLU activation function. . . . . 84
4.10 COIL20 dataset overall results, using FReLU activation function. . . . . 85
4.11 Results of MLP(FC) and AVG-Weighted algorithms on Lung dataset, using ReLU activation function. . . . . 86
4.12 Results of MLP(FC) and AVG-Weighted algorithms on Lung dataset, using FReLU activation function. . . . . 87
4.13 Results of SET, SF2SFrand $\zeta$-parametarized SF2SFrand, using both ReLU and FRelU activation function and Lung dataset. . . . . 88
4.14 Results of SET, SF2SFrand $\zeta$-parametarized SF2SFrand, using ReLU activation function and Fashion-Mnist dataset. . . . . 88
4.15 Evolution of $\zeta$ in every epoch. . . . . 89
4.16 Overall results on five algorithms, using Fashion-Mnist dataset. . . . . 89
4.17 Accuracy achieved by the competitors for the five datasets. . . . . 96
4.18 Execution time of the competitors for the five smaller datasets. . . . . 96
4.19 Accuracy achieved by the comparable algorithms, using Fashion-Mnist, Lung and COIL20 datasets. . . . . 97

4.20 Execution time of the comparable algorithms, using Fashion-Mnist, Lung and COIL20 datasets. . . . . 97
4.21 Competitors' training time between the old and the improved Keras implementation based on the network architecture used. . . . . 102
4.22 Competitors' validation accuracy based on the network architecture used. . . . . 103
4.23 Experimental evaluation of small CNN model, pre-trained on CIFAR-100 and tested on CIFAR-10. . . . . 110
4.24 Experimental evaluation of EfficientNetB0 model, pre-trained on ImageNet and tested on CIFAR-100. . . . . 111
4.25 Experimental evaluation of EfficientNetB2 model, pre-trained on ImageNet and tested on CIFAR-100. . . . . 112
4.26 Experimental evaluation of EfficientNetB0 model, pre-trained on ImageNet and tested on Intel classification dataset. . . . . 112

5.1 Distributed FL in a clustered ad hoc network. . . . . 119
5.2 Demonstration of data chunks $s$, given to every node $i$, consecutively. Every chunk contains different input vectors $w_{i,j}(t)$, for every time-step $t \subseteq T_s$. For each chunk $s$, every node executes local training. . . . . 125
5.3 Number of nodes excluded, in respect to both the data chunks, given in the beginning of every new round of the federation process and the coefficients sent for finding similarity among the nodes' data chunks. . . . . 126
5.4 Percentage decrease and average percentage decrease (%) of the cooperating nodes as a function of the coefficients sent, during the whole federation process. . . . . 126
5.5 Representative node performance after the global averaging procedure at round-data chunk 5. . . . . 127
5.6 Representative node performance after the global averaging procedure at round-data chunk 20. . . . . 128
5.7 Representative node performance after the global averaging procedure at round-data chunk 26. . . . . 129
5.8 Progress of a representative node for DISCREETER with 5 Fourier coefficients w.r.t. the rounds-data chunks. . . . . 130
5.9 Progress of a representative node for DISCREETER with 10 Fourier coefficients w.r.t. the rounds-data chunks. . . . . 131

Chapter 1

# Introduction

## 1.1 Introduction

Nowadays, the burgeoning paradigm of using Deep (DL) Learning algorithms to enhance not only sectors of every day life, such as health care, smart devices for home automation, but also security and surveillance, industrial monitoring, smart agriculture etc [3], necessitates shifting the processing of data and the deployment of machine/deep learning algorithms towards the edge, where the data are originated. Alongside, the tremendous success of the general Internet of Things (IoT) (see Figure 1.1) networks [4] or the Internet of Battlefield Things (IoBT) [5] has solemnified the demands of real-time processing to end devices, in parallel with strong network topology infrastructure (flat or hierarchical). This occurs because the ability of an edge device, to process its own raw data locally, entails not only brisker real-time decision making and low latency (as data has not to be transmitted to a server for processing), but also data privacy, since information remains to the edge device, and thus, less total energy consumption (since less communication between the device and the server occurs).

Adversely, the hardware resources (see Figure 1.2) of the aforementioned devices (e.g. micro-controllers -MCUs that avail some MB or even KB of RAM) are limited (extreme edge), make the deployment of conventional Deep learning (DL) algorithms impossible. In literature, there is a lot of work, done, towards the optimization of the *inference* phase, in these devices, emphasizing recently on Tiny Machine Learning (*TinyML*) [6]. Specifically, the training phase of a model, which is going to be deployed on an edge device takes place on a server, where both required energy and plethora of necessary data are abundant. Then, using suitable tools,e.g TensorFlow lite [7], its size is finally up to some KB of RAM. Then, the converted compact model can be applied to resource limited target devices. Contrarily, the inference phase is a static procedure, since its main purpose is to make predictions, based on the learned task, on new unseen data in which the model is exposed. However, during inference phase, the neural network model is not modified in its core and thus it is not able to learn new patterns and make better generalizations, or else it lacks of *interpretability*. The processing of data in the edge (training on edge), necessitates the introduction of development of techniques, which lessens neural networks parameters, and

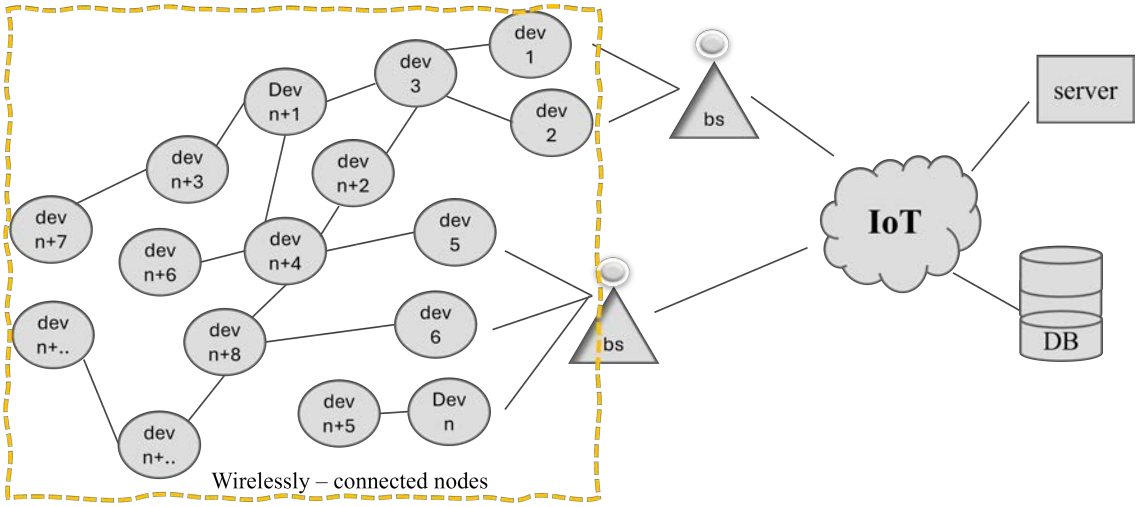



Figure 1.1: Overview of the Internet of Things (IoT), highlighting the role of connected devices.

thus execution time during training, leading to faster generalization of the processed data and further performance enhancement. For instance, *Pruning* [8] ( or *Sparsification*) is proved to be one of the most widely known mechanism of accelerating the learning process of the model, by evicting its unnecessary weights/connections/filters. *Transfer Learning (TL)* is a technique that utilizes pre-trained weights to facilitate faster model convergence, regardless of the task they were initially trained on [9]. Nevertheless, the model's core remains unaltered, presenting difficulties in adjusting to novel circumstances.

However, even if the drawback of limited resources is somehow mitigated through reducing the demands of deep learning algorithms, the data adequacy, required, in order for the model to converge remains a huge bottleneck in the revolution that training-on-edge incurs. Not only the amount, but also the type of data (e.g. unbalanced), produced by edge devices is not sufficient enough to build well- generalized models on their own. The key question is: How can we ensure that devices can conduct on-device real-time training without overfitting, due to the insufficient data or exhausting device energy?

This is how *Federated Learning* (FL) [10] brought into light. FL is a mechanism that enables participating entities to collaboratively train a global model, by exchanging necessary parameters (e.g. gradients), not data themselves. In this way, data privacy is preserved since data remains at the edge device. In centralized form of FL, there is a server -coordinator that orchestrates the whole procedure, by aggregating the devices' parameters and updating the global model. However, the cost of communicating often with the server, can cause communication overhead to the network (*flooding*) while the devices are not fully autonomous, since the processing is still draining its resources.

So, the transition from a centralized or semi-centralize topology into the complete decentralization (peer-to-peer communication) of the federation process mitigates not only the huge overhead that communicating weights among the nodes and the server entails, but also secure the data sovereignty over the devices. However, the reliability issues, incurred, by all-to-all nodes communication in an Ad hoc network is a major challenge, that has to be further examined. As a result, decentralized topologies necessitate the development of suitable communication-reliable and secure network infrastructure or else construction of strong *connected dominating sets (CDS)*. Nodes in a CDS encompass all network nodes directly or through neighbors, creating a

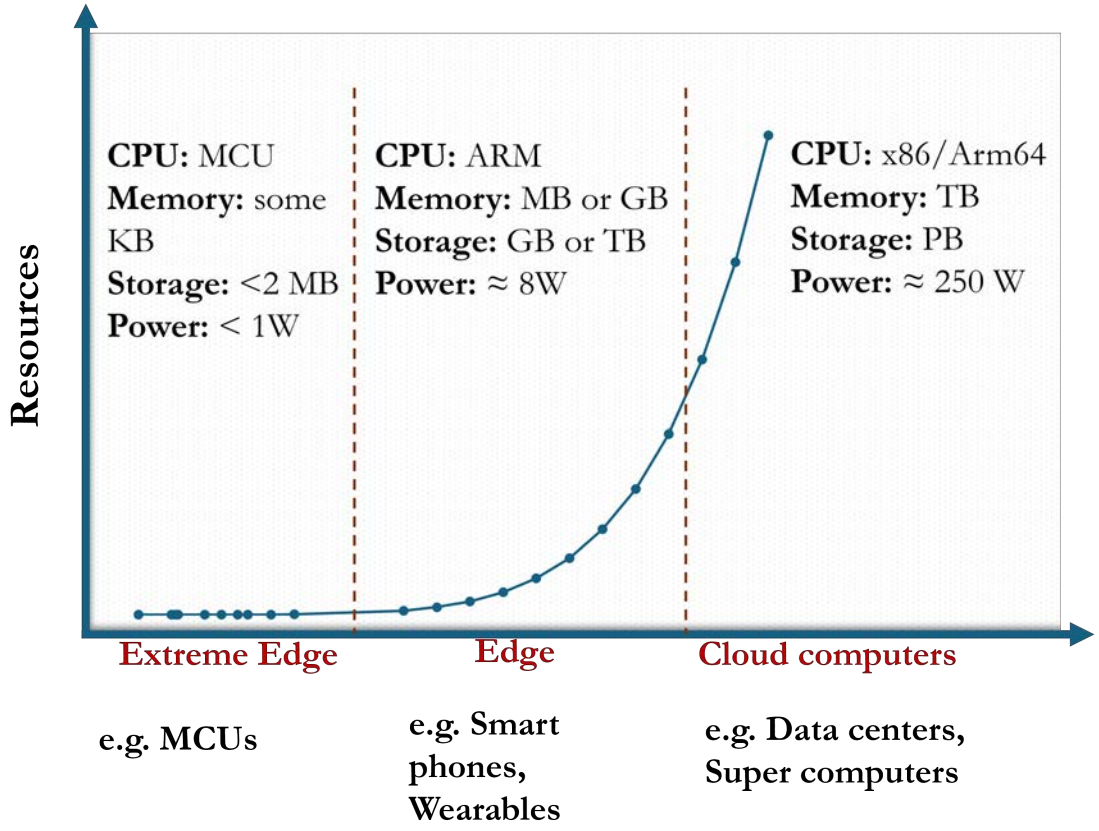



Figure 1.2: Classification of devices according to their resources vs specifications by Rajapakse et al. [1].

connected sub-network. These nodes serve as the network's "backbone," ensuring communication and control throughout.

The configuration of network backbones requires determining the minimum number of nodes in each sub-network for maximum connectivity. In this case, the measurement of the "importance" of a node is at the forefront of research in order to decide which node is suitable for being included to the sub-network. In the field of clustering methods, the word "importance" encompasses a range of meanings, resulting in diverse definitions of it. Taking into consideration the case where *centrality measures* are applied, "importance" can be defined by the flow or transfer that takes place in the center within the network. For instance, *betweenness centrality* measure computes the number of shortest paths between pairs of nodes. The higher the betweenness centrality is, the more control over the network this node has, because more information will pass through that node and hence it acts as a bridge between other nodes. So, when designing networks with the aim of reinforcing communication, nodes with high betweenness centrality may be prioritized as candidates for the backbone since they play a crucial role in connecting other nodes. Although, the most common criterion in order for the nodes to connect, is to choose a node with the shortest path (1−hop neighbor), but is this always the wisest decision? Which criterion better describes whether a node's communication link is strong enough and can be trusted? Adversely, the results indicate that a centrality measure that is suitable for one

category is likely to yield incorrect results when used in a different category. The measure of importance has to be evaluated by assessing the cohesiveness it causes to the network.

So, in Chapter 2, we provide an introduction of the concepts that will be further analyzed in this thesis. To enhance comprehension, we provide definitions and a concise overview of previous research, regarding these concepts.

Chapter 3 focuses on showcasing algorithms for building robust network infrastructure.

Chapter 4 details our proposed algorithms for enabling efficient on-device learning on resource-limited devices.

Chapter 5 delves into our analysis of reducing communication overhead in decentralized federated learning scenarios, while Chapter 6 concludes the existing work and introduces future work avenues.

## 1.2 Motivation & Contribution

Taking into consideration the challenges, that the learning process in wireless IoT networks poses, e.g. limited computational and data resources, high energy demands, or communication overhead, the main research question of the present thesis is summarized as following:

*How can we deploy deep learning (DL) algorithms in wirelessly connected resource-constrained IoT devices, without sacrificing their performance ?*

Accordingly, the original problem is decomposed into three distinct subproblems, and our main contributions can be summarized as follows:

***Question-1.1:*** *how can we construct strong and reliable multilayer backbones in order to ensure information circulation to the network with minimum communication overhead?*

The communication reliability in edge distributed environments, as introduced in chapter 3, is of a great importance, since research community focuses on the exploration of edge data processing and hence device independence. In wireless *Ad hoc* networks, every device, belonging to this network, can communicate among each other without requiring a coordination of a e.g. server node. This situation necessitates the construction of trustworthy ad hoc *backbones*, with the aim of reducing communication overhead by controlling information routing among the nodes of different LANs or sub-networks they need to interconnect. Although there is a lot of existing literature [11], referring to smart routing protocols in distributed networks, there is little contribution regarding multi-layer resource -scarce networks. The main goal is to constrain routing of information by constructing backbones with small *cardinality*, meaning that every node is urged to transfer information between the minimum nodes required in order for the message to be forwarded through the whole ad hoc network. By reducing unnecessary information exchange between the nodes, the energy consumption required for every device to collaborate is reduced, too, making these topologies, "hospitable" to support collaborative training techniques (e.g. Federated Learning methods) that will be further discussed in Chapter 5.

Thus, our work contributes the following:

- It establishes the computation complexity (NP-completeness) of the problem of calculating a minimum connected dominating set for multilayer networks.

- It presents a new distributed algorithm, namely the *Cross layer Connected Dominating Set* ($CCDS$) for calculating connected dominating sets, especially for IoBT networks, by applying an efficient mechanism to reduce the size of the dominating set.

***Question-1.2:*** *Can the traditional shortest-path betweenness centrality help us identify pivotal nodes of subsets of nodes (not regarding the whole network) so that we can make efficient clustering?*

The traditional shortest-path edge betweenness centrality [12], while effective at identifying globally influential nodes in a network ("mediators"), is often inadequate in decentralized federated learning (DFL), where communication and model updates need to be directed toward specific, high-value nodes—such as those with better data quality, higher computational resources, or more accurate local models. Standard betweenness centrality treats all node pairs equally, it may highlight nodes that are central in the overall network but irrelevant to the specific goal of reaching or aggregating from key contributors. Moreover, calculating the classic betweenness centrality across the entire decentralized network is energy-intensive, which is particularly problematic in ad hoc distributed systems involving edge devices or IoT nodes with limited battery life. As a result, relying on traditional betweenness centrality can lead to inefficient routing and suboptimal learning performance. Target-aware centrality measures, such as group or directed betweenness, offer more relevant insights by focusing on the paths that truly matter for the learning objective, enabling smarter collaboration and more efficient model convergence in federated systems. Starting from this observation, we introduce the generic concept of sink group betweenness centrality which can be used as a building block for designing algorithms to address the aforementioned problems. In section 3.2, regarding this context, we make the following contributions:

- we introduce a new centrality measure, namely the *sink group node betweenness centrality*, with the aim of considering path targeting toward specific clusters (sink groups);
- we extend the basic definition for networks weighted on the nodes;
- we extend the basic definition for the case of edge betweenness;
- we present basic algorithmic ideas for calculating the aforementioned notions.

***Question-2.1:*** *How much can we reduce model parameters or execution time in order to perform on-device training without sacrificing model performance?*

The emergence of fields such IoT networks and TinyML which necessitate the need of having a learning process appropriate for running in resource-starving devices e.g., sensors, have issued significant research questions – among others – on how to reduce the number of parameters (weights) of a neural network; such a reduced model implies a reduction of both the computational power needed to set these weights for instance via back-propagation, and also a reduction in the storage (memory) requirements [13] that this huge set of weights generates for the small/tiny device hosting the training process. Therefore, techniques falling into the generic family of model reduction and being termed as neural network sparsification [14] have been a very intense area of investigation. Indeed, topology sparsification has proved itself as a very promising technique for speeding up neural network train- ing. The efficacy of this family

of methods is supported by the fact that among the vast number of trainable weights that a neural topology has, only a moderate percentage of them differs significantly from zero; in other words the near-zero-weight links that connect neurons could be erased. Indeed some methods adopt such sparsification decisions after the training phase, in order to speed up the inference (operating) phase of the network [15].

This work addresses the problem of neural model reductions from a novel perspective. We get inspiration from findings that actual biological neural networks are scale-free [16]. Thus, we explore network science tools to reduce the number of connections (weights), according to a specific rule. We take a principled approach in the sense that we seek to produce a particularly structured topology, e.g., a scale free topology starting from another structured topology, e.g., a small-world topology. The only prior work investigating such an approach is reported in [17]; however their starting point is a 'random' network according to the Erdos-Renyi notion [18]. This initializing method might end up being not efficient at all, because depending on the linkage density (controlled by a hard-to-set parameter) the initial network might be too dense (and almost fully connected) or too sparse (and unable to reach a scale-free topology). It might take too long to reach a structured (scale-free, or small-world) topology, starting from a purely random topology. Moreover, we depart from the common approach where pruning is done after training, and we perform pruning after each epoch; contrary to the intuition, this technique does not give bad results at all, but it is very competitive and even superior in many cases. In that context, this paper makes the following contributions:

- It puts forward a systematic effort to investigate the toolbox of network science to accelerate neural network training.
- It proposes algorithms to construct structured neural topologies starting from other structured neural topologies, all different from fully connected bipartite ones, to speed up the training phase of feed forward neural networks.
- It evaluates the performance of these algorithms, and confirms their rationale. It marks a winner algorithm and detects its features that makes it prevail.

***Question-2.2:*** *How can we leverage prior knowledge to accelerate model convergence in resource scarce devices?*

TinyML contributes to the efficiency of models performance, by running mainly inference tasks on ultra-low power devices. Moreover, TL offers pre-knowledge to the target device, maximizing the levels of its performance. Most of the works, presented in existing literature, are referred to inference tasks. By combining the two aforementioned methods in this work, we try to enable step to step not only the inference but also the training procedure in resource-constrained devices, by retraining not only the last fully connected layer, but also both the fully connected and the last convolutional layer. The main prior work is the one presented in [19]).

The basic motivation is extracted by the way a CNN network performs its tasks. Specifically, in an image classification task, the first layers of a CNN network learn the basic concepts of an image, then this knowledge is transferred to the next layers in order for the final ones to be able to learn the high level concepts of the input-image. This means that the final convolutional layer/layers is/are responsible for the classification of the data, being processed by the network. So, when the network is exposed to new data, it is of high importance to update not only the categorization of data, but also the categories themselves, so as to achieve high Accuracy levels.

By this way, we start to enter the core of the network, without needing the resources, that the whole network requires, in order to operate. Overall, our work contributes in the following:

- We develop a technique where not only the final fully connected layers are retrained, but also some of the immediately previous convolution layer are retrained. This retraining can potentially take place in resource-starving devices, thus improving data privacy and being ideal for cases of federated learning [20] or Internet of Things implementations [5].
- We investigate the trade-off between training more layers versus accuracy versus energy consumption which has not been explored so far. Evidently, retraining more of the last layers (both convolution and fully-connected) will result in improved accuracy, but it will cost in energy consumption. Is there any optimal point with respect to how many retrained layers are enough before consuming too much energy without gaining in accuracy?
- We use homogeneous data to experiment with, but by affecting a convolution layer through retraining, we can more easily deal with heterogeneous data in the future, since the convolutional layers are the ones responsible for the better understanding and categorization of the hierarchical features of image data; we compare the examined methods against two baseline methods: a) the methods reported in [19] (*Baseline-1*) and b) when the whole CNN is retrained (*Baseline-2*).

***Question-3:*** *How much can we reduce the participating entities in order to alleviate communication overhead in distributed federated learning schemes?*

In order to minimize message exchange and thus, communication cost among all nodes within the network, *clustering* algorithms are used. By fitting nodes into groups, means that these nodes dispose some similar attributes and hence some nodes can be excluded of the collaborative learning process. All the works on federated learning over clustered ad hoc networks [21–23] (usually found under the term "cooperative D2D with cluster ..." ) do not pay attention to the actual clustering mechanism. However, this should be a very significant part of the whole FL process because it can have a serious impact on the communication reduction. It is well-known that the vast majority of the ad hoc network clustering schemes (if not all) are based on node IDs. Thus, it is very common the situation that cluster heads (CH), i.e., the nodes that will eventually perform the local averaging in the FL scheme are one-hop neighbors. This will negatively impact the communication efficiency of the scheme due to the interference since all the intra-cluster communication will be carried out by the cluster head. Moreover, almost all adhoc clustering algorithms produce clusters with very small cardinality, thus local averaging/consensus is not very efficient with respect to the communication reduction.

Additionally, the communication range restrictions (a range of a few hundred meters) results in the wireless nodes being frequently in very close proximity, thus producing very similar data, e.g., sensor measuring temperature. This situation turns almost all proposed schemes for client participation to the federation, e.g., based on the past workload completion capability [24], on secretary selection formulation [25] as being not appropriate.

Based on the above shortcomings, in the present thesis we contribute mainly the following methods towards developing an appropriate federated deep learning algorithm for wireless ad hoc network environments (See Figure 1.3):

- We develop an ad hoc network clustering scheme appropriate for performing local and global aggregations that reduces wireless interference.

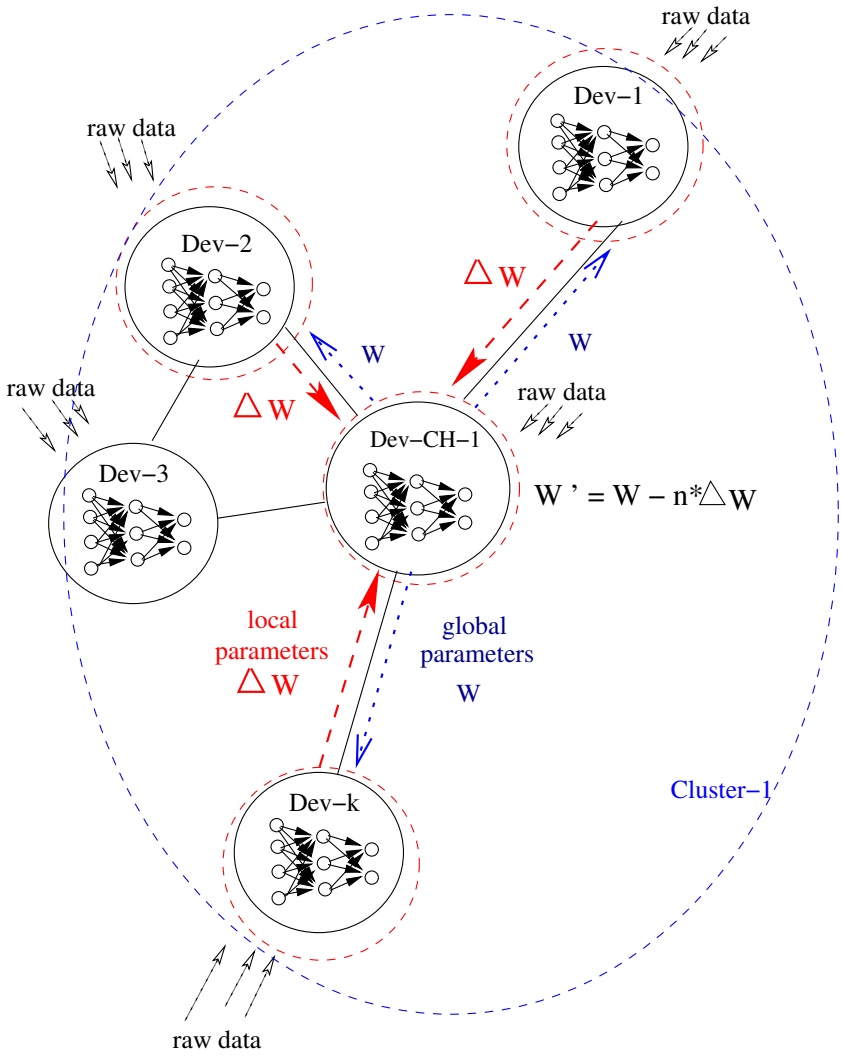



Figure 1.3: An illustration of the parameter exchange architecture. The representative node aggregates the weights from each selected participating node, which sends its local parameters (red dashed lines). After global aggregation, the aforementioned node broadcasts back the updated weights to the associated nodes in order to continue local training.

- We develop a client participation to the federation that is based on data-similarity in order to reduce communication during aggregations.

- We develop a suite of federation algorithms based on variations of the basic methods, and evaluate them based on the speed of convergence.

Despite the fact that all related works on federated learning without the coordination of a central server use the term "decentralized federated learning" to describe their architecture, in the present work we specifically use the term "distributed federated learning", because the architecture of a wireless ad hoc network is a classic paradigm of a distributed (or networked) architecture and the literature on ad hoc networking appears consistently with this term during its lifetime. Moreover, we use this term to emphasize that all operating components and

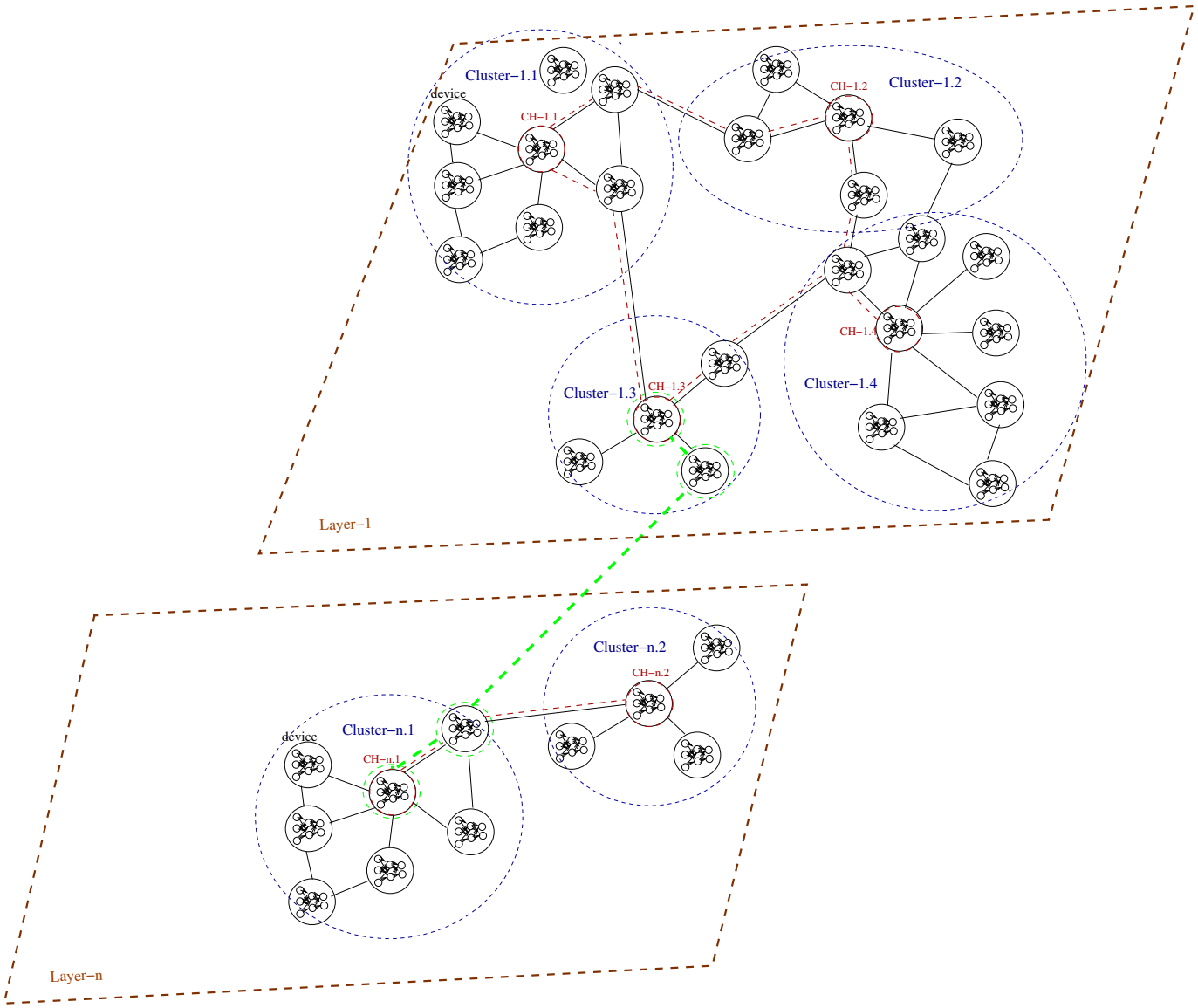



Figure 1.4: Desideratum of the current thesis.

operations are taking place in a distributed fashion utilizing only wireless message exchanges and concepts and protocols well-established in the distributed community.

So, in Figure 1.4, the desideratum of current thesis is depicted. We aim at advocating a joint design among efficient sparsified neural network, running on resource-constrained wirelessly-connected devices, belonging to the same and/or different layers of a network. We put nodes with similar data in the same group (cluster - blue circles) and we elect the representative node (CH) attaching importance on placing it to the center of the cluster. Every node executes its own training task, but only the selected ones send their learned parameters to their CH. The red dashed lines represent the links where CHs communicate with one other in order to send weights for global aggregations (intra-layer communication). The green dashed lines illustrate the path of communication across CH of different layers (inter-layer communication) of a network/subnetwork.

# Publications

### Articles in Book Chapters

[BC 1] **E. Fragkou**, D. Katsaros, *Non-static tinyml for ad hoc networked devices*, in: B. S. Chaudhari, S. N. Ghorpade, M. Z. R. Paškauskas (Eds.), TinyML for Edge Intelligence in IoT and LPWAN Networks, **Elsevier**, 2024, pp. 229–251.

### Articles in Journals

[J 1] **E. Fragkou**, E. Chini, M. Papadopoulou, D. Papakostas, D. Katsaros and S. Dustdar, *Distributed Federated Deep Learning in Clustered Internet of Things Wireless Networks With Data Similarity-Based Client Participation*, **IEEE Internet Computing magazine**, vol.28 (6), 2024, pp. 53-61.

[J 2] **E. Fragkou**, D. Katsaros, *A joint survey in decentralized federated learning and tinyML: A brief introduction to swarm learning*, **Future Internet (MDPI)**, Vol.16 (11), 2024.

[J 3] **E. Fragkou**, M. Koultouki, D. Katsaros, *Model reduction of feed forward neural networks for resource-constrained devices*, **Applied Intelligence (Springer)** Vol.53 (11), 2023, pp. 14102—14127.

[J 4] **E. Fragkou**, D. Papakostas, T. Kasidakis, D. Katsaros, *Multilayer backbones for internet of battlefield things*, **Future Internet (MDPI)**, vol.14 (6), 2022.

### Articles in Conference Proceedings

[C 1] **E. Fragkou**, V. Lygnos, D. Katsaros, *Transfer learning for convolutional neural networks in tiny deep learning environments*, **Proceedings of the 26th Pan-Hellenic Conference on Informatics (PCI22)**, hybrid, November 2022, pp. 145—150.

[C 2] **E. Fragkou**, D. Katsaros, Y. Manolopoulos, *Sink group betweenness centrality*, **Proceedings of the International Database Engineering and Applications Symposium (IDEAS)**, Montreal, Canada, July 2021, pp. 21–26.

[C 3] A. Chouliaras, **E. Fragkou**, D. Katsaros, *Feed forward neural network sparsification with dynamic pruning*, **Proceedings of the 25th Pan-Hellenic Conference on Informatics (PCI21)**, Volos, Greece, November 2021, pp. 12—17.

[C 4] D. Papakostas, T. Kasidakis, **E. Fragkou**, D. Katsaros, *Backbones for internet of battlefield things*, **Procededings of 16th Annual Conference on Wireless On-demand Network Systems and Services Conference IEEE/IFIP (WONS)**, 2021, pp. 1—8.

**Poster Presentations**

[P 1] Evangelia Fragkou, Dimitrios Katsaros, *Distributed Federated Learning in Wireless Ad hoc Networks*, Poster, **6th Summit on Gender Equality in Computing (GEC24)**, Nicosia, Cyprus, June 13–14, 2024.

[P 2] Evangelia Fragkou, Dimitrios Katsaros, *Transfer Deep Learning for TinyML*, **Poster, 5th Summit on Gender Equality in Computing (GEC23)**, Athens, Greece, June 26–27, 2023.

[P 3] Evangelia Fragkou, Dimitrios Katsaros, *Memory Reduction and Training Acceleration of Neural Networks for Tiny Machine Learning*, Poster, **6th Summit on Gender Equality in Computing (GEC22)**, Thessaloniki, Greece, June 16–17, 2022.

Chapter 2

# Introduction to the wide area of Deep Learning for wirelessly connected resource-limited devices.

## 2.1 Accelerating Neural Networks' deployment

Machine/Deep Learning methods have high computational demands, since they need both sufficient data and high training execution time, in order to learn these data and hence, converge. Optimizations like multi-core parallel execution regarding both CPUs and GPUs, deep learning caching techniques [26], etc are efforts, trying to bridge the gap between the efficient training/inference of a model and the enormous execution time/computational cost, needed by such large models e.g., GPT-x [27], having billions of connections among its neurons. However, talking about resource constrained devices/environments (see Table 2.1), e.g. Internet of Things (IoT), this is not feasible, taking into account that these devices extremely lack of computational power and memory (approximately MegaBytes or KiloBytes of Ram, e.g. sensors, embedded systems, etc., and the existence of a tremendous number of parameters, e.g. weights, impacts both inference and training time.

The research on speeding up neural network training dates back to the late '80 – early '90. One of the first families of acceleration methods includes members that meant to replace the traditional gradient (steepest) descent optimization method. Steepest descent is based on a first order Taylor series approximation of the performance function (mean square error) and is very slow. Therefore, methods based on second order Taylor series were investigated, such as Newton's method. Other algorithms that departed from the first order gradient concept, are those based on conjugate gradient, and the similar in spirit quasi-Newton method of Broyden-Fletcher-Goldfarb-Shanno (BFGS), along with its variations, e.g., L-BFGS [28]. Adam, Adadelta, and Nadam are some of the recently proposed fast optimizers [29, 30]. Another

approach to accelerate neural training involves developing variable learning rates [31]. The topic of network architecture search [32, 33], which aims to design a neural architecture that achieves the best possible performance using limited computing resources in an automated way with minimal human intervention, bears similarity with the work, presented in this thesis and it is a very hot research area [34]. Nevertheless, these methodologies often necessitate a substantial investment of computing resources (e.g. consuming hundreds of GPU days), involve large search spaces, numerous hyper-parameters, and intricate implementation techniques. Hardware-based accelerators comprise another family and architectures such as FPGAs, multicore CPUs [35], TPUs [36] are increasingly used for neural training and inference. Furthermore, the technique of dropout [37, 38] constitutes the founding member of a new family, which randomly drops neurons during training. Similar in spirit, are the methods which compute only a subset of gradients during back propagation, e.g., meProp [39, 40] which prunes neurons based on how many times the back propagation updates a neuron.

A recent, very promising line of research suggests that only a sub-network of the topology is responsible for carrying out accurately a particular task, and thus if we can detect which is this sub-network and then train only this, pruning the rest of the network, then we can gain significant speedups in training.

Contemporary experience showed that there exists a small part of a dense neural network that can achieve (almost) the same test accuracy as the whole network. This observation was intially tested on fully connected multilayer perceptrons and convolutional neural networks in vision tasks and was summarized in the famous *Lottery Ticket Hypothesis* quotated from [41, 42].

In order to identify a winning ticket, an appropriate algorithm would initialize and trainee the whole network and after the completion of training, would remove the connections that correspond to the weights with the smallest magnitudes. After that pruning step, the algorithm would restore the weights of the survived connection to the values they got at the initialization step! The training and pruning could take place once at the end of the training (one-shot), or every after a specific number of iterations pruning only a percentage of the survived connections (iterative pruning).

A surge of future works followed the initial article, that generalized this technique to the *multi-prize lottery ticket hypothesis* [43], or applied the initial ideas to spiking neural networks [44–46], in few-shot learning [47] and so on.

However, the complete understanding of trade-offs involved in pruning is yet to be understood [48].

Table 2.1: Resource-limited devices/boards aiming to run deep learning tasks.

| **Devices/Boards** | **Power** | **Instruction Set** | **SRAM** | **Flash Memory** | **CPU Clock** |
|---|---|---|---|---|---|
| *Raspberry Pi family* ([49–51]) Raspberry Pi 3B+ | Low | ARM (Cortex-A53) | 1 GB SDRAM | – | 1.4 GHz |
| Raspberry Pi 4B | Low | ARM (Cortex-A72) | 256KB | – | 1.5 GHz |
| Raspberry Pi Pico | Ultra low | ARM (Dual core Cortex-M0+) | 264 KB | 2 MB | 133 MHz |
| *Arduino family* [51–53] Arduino Nano 33 BLE Sense (nrF52840 SoC) | Ultra low | ARM (Cortex-M4) | 256 KB | 1 MB | 64MHz |
| Arduino Portenta | Low | ARM (Cortex-M7−M4) | 8 MB SDRAM | 16 MB | 480-240MHz |
| *STM Microcontrollers (MCU)* [46, 49, 54] STM32F7 board | Ultra low *(high-performance MCU)* | ARM (Cortex-M7) | 512 KB | 2 MB | 216 MHz |
| STM32H743 | Ultra low *(high-performance MCU)* | ARM (Cortex-M7) | 1 MB | 1−2 MB | 480 MHz |
| STM32F401 | Ultra low *(general-purpose MCU)* | ARM (Cortex-M4) | 96 KB | 128 KB− 512 KB | 84 MHz |
| STM NUCLEO **L4**96ZG | Ultra low *(general-purpose MCU)* | ARM (Cortex-M4) | 320 KB | 1 MB | 80 MHz |
| STM NUCLEO **F7**67ZI | Ultra low *(high-performance MCU)* | ARM (Cortex-M7) | 512 KB | 2 MB | 216 MHz |
| *Adafruit Feather Family* [51, 55] Bluefruit Sense board (nrF52840 SoC) | Ultra low | ARM (Cortex-M4F) | 256 KB | 1 MB | 64MHz |
| M4 express | Ultra low | ARM (Cortex -M7) | 192 KB | 2 MB | 120MHz |
| *Microprocessors* GAP8 [56, 57] | Parallel ultra-low-power processing platform (PULP) | RISC-V (FC) | 80 KB+ 8 MB SDRAM | 512 KB | 250MHz |
| VEGA[57] (22nm technology) | Parallel ultra-low-power processing platform (PULP) | RISC-V (FC) | L2 (interleaved) 1.5 MB + 64 KB | 64 MB | 250MHz |
| Mr.Wolf[58] (40nm LP CMOS technology) | Parallel ultra-low-power processing platform (PULP) | RISC-V (RVC32IMF) | latch-based memory instead of SRAM (25 Gbit/s at 100 MHz) | – | 32KHz - 450MHz |

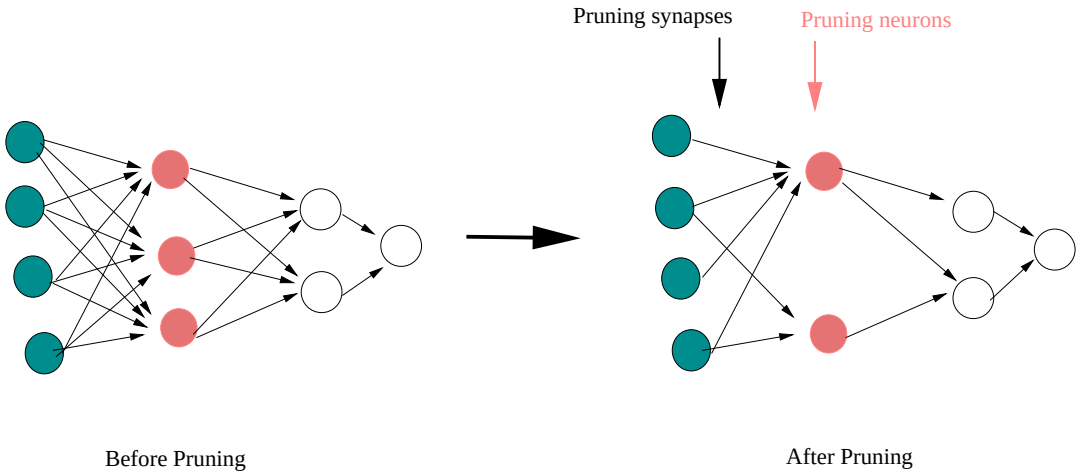



Figure 2.1: An illustration of pruning either neurons or synapses of a deep neural network.

### 2.1.1 Pruning/sparsification

A promising and extensively demonstrated, in literature, concept of lessening computational demands of ML/DL methods is to reduce both the size and complexity of the neural network we want to train, by making it more "sparse", meaning that we can drop off (prune) synapses/neurons of the network that they really does not contribute to its performance. We can either talk about *static* pruning, which is performed once, usually after training, and the pruned structure remains fixed throughout the model's use or *dynamic* pruning, which involves constantly modifying which connections or weights are pruned, depending on the network's current state. Despite the phase in which the pruning is conducted (static or dynamic), we can make another categorization of Pruning as follows [59]: the structured or structural pruning (like channel pruning), and the unstructured one (see Figure 2.2). Structural pruning removes entire structures, such as neurons, filters, or layers, making the model smaller while maintaining its original architecture. In contrast, unstructured pruning discards individual weights, producing a sparse model with the largest size reduction possible, without introducing structural alterations. So, the crucial part is to find which type/criterion for pruning works better in every case, regarding the nature of the network we use for. It is observed, according to [60] and [8] that random unstructured pruning techniques are more efficient when they are applied for weights removal rather than neurons/filters removal in the phase of the network initialization. However, even in this case randomly reducing information of the network may cause severe damage to the network performance.

The family of algorithms related mostly to the presented work are those based on topology *Sparsification* [14]. The technique, called *Sparsification*, is promising since aims at reducing model size and hence memory requirements, by dropping out unnecessary - regarding to the training procedure - connection of the network. Methods have been proposed which prune connections between the neurons; for instance, the works [61], [62] and [63, 64]. However, these linkage sparsification techniques do not aim at mimicking the topological structure of real neural

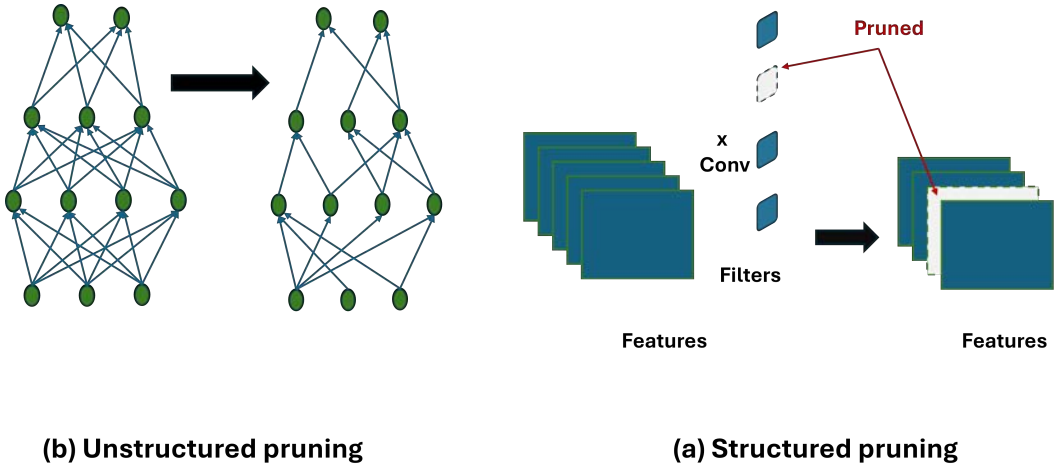



Figure 2.2: Structured vs Unstructured pruning.

networks, but are mainly based on eliminating close-to-zero weighted connections. Additionally, there is great inspiration in the structure of the synapses in the human brain and hence, the way human brain controls incoming information. More specifically, the human brain creates thousands of synapses every day as it is exposed to new data (and consequently produces information), while it has the ability to make a decision of which of them to keep, since there are not all useful, to the same extent. At the same time, network science introduces the concepts of scale free networks and based on them, describes a variety of networks that we use in everyday life, such as the world-wide-web (web). In scale free networks, the nodes follow a power low distribution, i.e. the nodes where most synapses end up, are considered the most popular and consequently they are the nodes that undertake to transfer information between the nodes of the network.

The work, described in [17] was the first to correlate the functionality of scale free networks with the one of the deep neural networks, but the initialization of synapses in the first untrained network is done in a random way (Erdos Renyi type of network), while only the final network is structured, following scale free law and in [65] there is an optimized version of it.

#### 2.1.1.1 Background on network science concepts

Network science is the discipline that analyzes the properties and function of complex networks, such as technological, social, biological, physical and so on. Complex network analysis consists of algorithms and methodologies for studying and developing: centralities, communities, network growth and the respective models [18], diffusion processes [66], etc. Well-studied network models comprise random networks, regular lattices, small-world networks, and scale-free networks.

**Definition 2.1.1** (Erdos-Renyi Network)**.** *A random (or Erdos-Renyi) network is a network that consists of $n$ nodes where each node pair is connected with probability $p$. The degree distribution*

*of a random network follows a Poisson distribution.*

**Definition 2.1.2.** *A regular lattice is a network consisting of n nodes where each node has the same number and pattern of connections with every other node in the network. The degree distribution of a regular lattice is uniform (constant).*

**Definition 2.1.3.** *A small-world (or Watts-Strogatz) network that consists of n nodes interpolates between a regular lattice and a random network. In other words, it presents the regularity in neighborhood connectivity that a regular network has, but it also has some random connections towards random nodes of the topology. The degree distribution of a small-world network follows a Poisson-like bounded degree distribution.*

**Definition 2.1.4.** *A scale-free network is a network that consists of n nodes whose degree distribution follows a power-law; i.e., there are hub nodes (highly connected nodes) at any scale of observation. A particular type of scale-free network is generated by a process known as preferential attachment - known as Barabasi-Albert (BA) model [67] - according to which when a new node is added into the network it wires itself towards nodes which have already high degrees.*

An example of these networks, namely regular, small-world, random and scale-free is depicted in Figure 2.3.

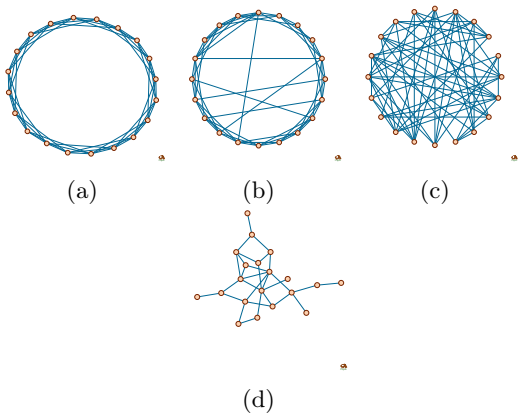



Figure 2.3: (a) A 20-node regular network (with 3 neighbors at each side of every vertex), (b) a 20-node small-world network (with initially 3 neighbors at each side of every vertex, and with 0.075 rewiring probability), (c) a 20-node random network (with average node degree 6) and (d) a 20-node scale-free network. Graphs were generated with Pajek.

### 2.1.2 Utilizing prior knowledge

Real-time data processing encounters various challenges, regarding both data adequacy in order for the model to generalize well and memory/energy limitations or enormous training time, required. In literature, the only way of implementing deep learning algorithms to the

ultra-low power devices, e.g. sensors, micro-controllers, IoT devices etc. is to pre-train the ML/DL model to a server [68, 69], which avails both resources and plethora of data and then the ML/DL model has to be compiled through suitable software, eg micro TVM, TensorFlow lite, etc, so as to run to MCUs. After that, the model run on devices without updating its core weights, they only run inference tasks (*TinyML*). Nevertheless, this standpoint leads to inflexible models that cannot accommodate new scenarios,e.g. different distribution of the streaming data - non Independent and Identically Distributed (non-IID) data [70], triggering severe problems e.g., concept drift [71], [72], [73], which decrease neural network accuracy. Taking into account that the pre-trained weights, regardless of their original training task [9], can enhance model convergence, the *Transfer Learning (TL)* methodology (see Definition 2.1.5) gained immense popularity. TL is a widely-known method, used for leveraging knowledge of a domain (source), trained on large-scale data, to the target domain, which is used to be a smaller and less data-processing-capable one, with the aim of boosting the learning ability of the latter one [74–76]. Transfer learning can be classified into three primary types: *inductive*, *transductive*, and *unsupervised* transfer learning [77]. These types handle distinct relationships between the source and target tasks or domains. When the source and target tasks differ, inductive transfer learning takes place and the knowledge from the source task aids in improving learning in the target task. It includes self-taught learning, where tasks differ but domains remain the same, and multi-task learning, where related tasks are learned simultaneously. The tasks in transductive transfer learning are the same, but the source and target domains differ. When neither source nor target data is labeled, unsupervised transfer learning becomes valuable for tasks like clustering or dimensionality reduction across different domains by transferring knowledge between tasks. These categories facilitate transfer learning in handling real-world scenarios with limited data or task shifts.

**Definition 2.1.5** (Transfer Learning)**.** *Given a learning task $T_t$ based on $D_t$, and we can get help from $D_s$ for the learning task $T_s$. Transfer learning aims to improve the performance of the predictive function $f_T(\cdot)$ for the learning task $T_t$ by discovering and transferring latent knowledge from $D_s$ and $T_s$, where $D_s \neq D_t$ and/or $T_s \neq T_t$. In addition, in most cases, the size of $D_s$ is much larger than the size of $D_t$, $N_s \gg N_t$.*

As a consequence, TL mechanisms achieve high performance in real – world applications, such as cross-language text categorization, text-to-image classification, etc. Some already known deep-learning-based approaches are DAN [78], DCORAL [79], and DANN [80], [81] which are applied for solving image classification problems.

However, by fine-tuning the last layers (usually the fully connected ones) may cause imperceptible improvement or no improvement at all, since in this way we affect only the weights of the classifier and we do not enter the main core of the network. For instance, inspired by the way a Convolution Neural Network (CNN) works, in which every layer has a specific task to learn, by retraining only the last fully connected layer will not probably be effective, since we modify

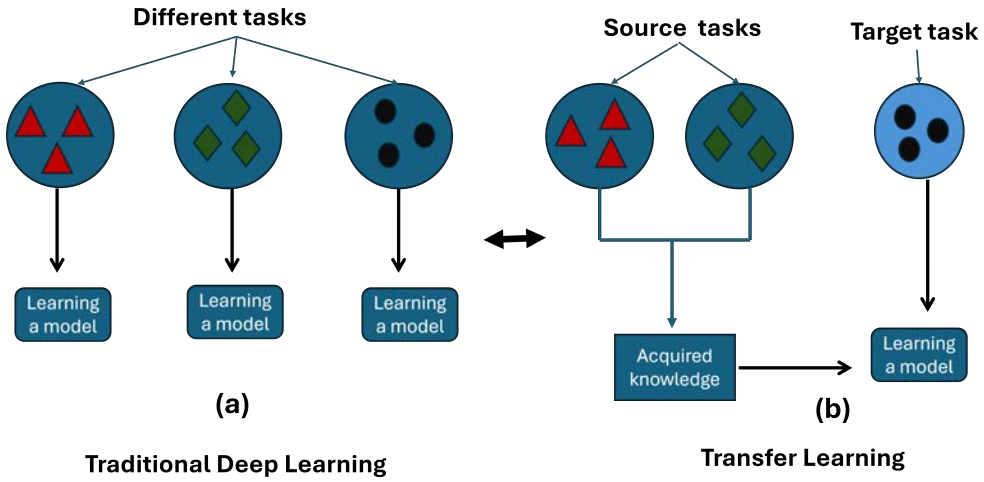



Figure 2.4: (a) Conventional ML/DL, (b)Transfer Learning mechanism.

the results of the training procedure and not the procedure itself in order to embody the weights of the new task before the classification outcome. In literature, some valuable endeavors are introduced, for example, in contrast with conventional TL, TinyOL [82] doesn't retrain the last fully connected layer but it functions as an extra layer, connected to the last layer of the already pre – trained model. It supports training on streaming data using online SGD in order for the weights to be adapted to new data, while it runs in the RAM and the pre-trained weights are frozen by the time, TinyOL starts being fine tuned. Furthermore, a scheme based on Transfer learning in conjunction with K-nearest neighbors, which is capable of running training (not inference) in embedded systems and IoT units, which are firstly trained with a small amount of data is presented in [49]. In this way, networks can evolve over time, enhancing its performance by implementing local training. Last but not least, the [19] combines TL and collaborative learning (see Section 2.3), enabling a kind of on-device training in devices with less than 1 MB of memory. Nevertheless, the pre-trained weights are some kind of pre-knowledge, after many connection modifications and hence weights replacements, the first task, for which weights were trained for, tends to be "forgotten" (catastrophic forgetting [83] - one of the crucial problems in deep learning methodologies, attracting great scientific interest) and it is a prospect that have to be further researched and discussed.

## 2.2 Network infrastructure

### 2.2.1 Measuring the *importance* of nodes/paths

#### 2.2.1.1 Definition of Betweenness Centrality measures.

*Betweenness centrality* (2.1) is a graph theory measure that determines the most influential nodes or edges in a network. It evaluates the extent to which a node is situated on the *shortest*

*paths* (or else the minimum edge count for traversing between two nodes in the network) linking other nodes. In simpler terms, it measures the occurrence of a node acting as a bridge on the shortest path linking two other nodes.

The betweenness centrality $C_B(v)$ of a node $v$ is given by the following formula:

$$C_B(v) = \sum_{s \neq v \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}} \tag{2.1}$$

where:

- $s$ and $t$ are different nodes in the network.
- $\sigma_{st}$ is the total number of shortest paths from node $s$ to node $t$.
- $\sigma_{st}(v)$ is the number of shortest paths from $s$ to $t$ that pass through node $v$.

Betweenness centrality helps identify nodes or edges that are critical for connectivity and information transfer in a network. The more high a betweenness centrality value is, the more important the key role of the node is, in connecting different parts of the network. The initial concept of (shortest path) betweenness centrality [84] gave birth to concepts such as the proximal betweenness, bounded distance betweenness and distance-scaled betweenness [85], bridging centrality [86] to help discover bridging nodes, routing betweenness centrality [87] to account for the paths followed by routed packets in a networks, percolation centrality [88] to help measure the importance of nodes in terms of aiding the percolation through the network. There are so many offsprings of the initial concept, that even a detailed survey would found practically impossible to record each one published! The realization that the computation of betweenness centrality requires global topology knowledge and network-wide manipulation, which is computationally very expensive, spawned research into distributed algorithms for its computation [89], and inspired variants aiming at facilitating the distributed computation of notions similar to the original betweenness centrality, such as load centrality [90]. Betweenness centrality and its variations have found many applications not only in classical fields in network mining, but also in delay-tolerant networks [91], ad hoc networks [92], in distributed systems e.g., for optimal service placement [93], etc. Approximate Betweenness Centrality. Many applications working over modern networks require the calculation of betweenness centrality in nearly a real-time fashion or have to deal with a huge number of nodes and edges. So, the decision of trading off the accuracy of betweenness centrality computation with speed arises as a natural option. Some early works considered approximating the exact values of betweenness centrality [94], [95]. Later on, this became a very active research field [89]; a survey can be found at [96]. Group Betweenness Centrality. Group betweenness centrality indices [97] measure the importance of groups of nodes in networks, i.e., they measure the percentage of shortest paths that pass through at least one of the nodes of the group. On the other hand, co-betweenness

centrality [98] measures the percentage of shortest paths that pass through all vertices of the group. Algorithms for fast calculation of these group centrality measures have been developed [99], [100] even for diverse types of networks [101]; group (co-)betweenness or their variations have found applications in monitoring [102], in network formation [103], etc. Sink Betweenness Centrality. The notion of Sink Betweenness centrality [104], which is a specialization of our Sink Group Betweenness Centrality, was developed in the context of wireless sensor network to capture the position of nodes which lie in many paths leading to a specific node, i.e., the sink. So, the sink betweenness of a sensor node was correlated to the energy consumption of than node, since it had to relay a lot of messages towards the sink node.

### 2.2.2 Backbone construction: The Internet of Battlefield Things (IoBT) paradigm

The methodology of *Connected dominating sets* (CDS 2.2.1) play a critical role in applications like network design. For example, in wireless communication networks, a CDS can serve as a backbone for efficient routing. By ensuring that only a small, connected set of central nodes handles the communication, the network can minimize resource usage, while still maintaining full coverage and connectivity. Thus, CDS is crucial for optimizing network structure, guaranteeing reachability and connectivity with minimal nodes. As a result, the literature on designing backbones for routing support in wireless ad hoc networks is literally enormous during the past two decades [105–107].

**Definition 2.2.1.** *A subset $D$ of vertices in a graph $G = (V, E)$ is called a* ***connected dominating set (CDS)*** *if:*

- *$D$ is a dominating set, meaning that every vertex $v \in V$ is either in $D$ or adjacent to at least one vertex in $D$.*
- *The subgraph induced by $D$, denoted as $G[D]$, is connected, meaning that there is a path between any two vertices in $D$.*

In recent years, the *Internet of battlefield things* (IoBT) has emerged as a practical example of a wireless network [108, 109]. The IoBT refers to the integration of smart, networked devices and technologies in military environments to enhance situational awareness, decision-making, and operational efficiency. Similar to the broader Internet of Things (IoT), IoBT involves the use of interconnected sensors, drones, autonomous vehicles, wearables, and other advanced technologies that collect, process, and share data in real-time on the battlefield. These devices work together to provide soldiers and military commanders with timely and accurate information about the environment, enemy forces, and logistical factors, enabling more informed and precise actions. IoBT also supports automation, intelligent decision-making, and advanced communications, which are critical in complex and dynamic combat scenarios. The key challenges in IoBT

include ensuring security, managing vast amounts of data, maintaining reliable connectivity in hostile environments, and preventing cyber threats. Overall, IoBT represents a shift toward network-centric warfare, where advanced technology plays a central role in modern military operations. since then, the research is further conducted in designing backbones [110, 111], in designing reconfigurable and secure IoBT networks [112], in developing solutions for enemy localization [113], in controling/monitoring communication links, in combating attacks at nodes [114], in detecting malware and fake news [115, 116], and in protecting human assets [117, 118].

However, the same issue in the realm of military/IoBT networks is different; these networks are actually multilayer networks [110, 112], and therefore this past literature is in principle inappropriate, because it concentrates only on single layer networks. Moreover, it is shown in ([110] Theorem 1), that these algorithms usually produce suboptimal solutions in the sense that, instead of announcing as a dominator a node with a few interlayer links, they tend to announce someone with many intralayer links. Finally, the work in [119] deals with domination in multiplex networks which is a far more restricted version of multilayer networks, since interlayer links exist only between *clones* of the same node in different layers; moreover, that work does not establish the computational complexity of their problem.

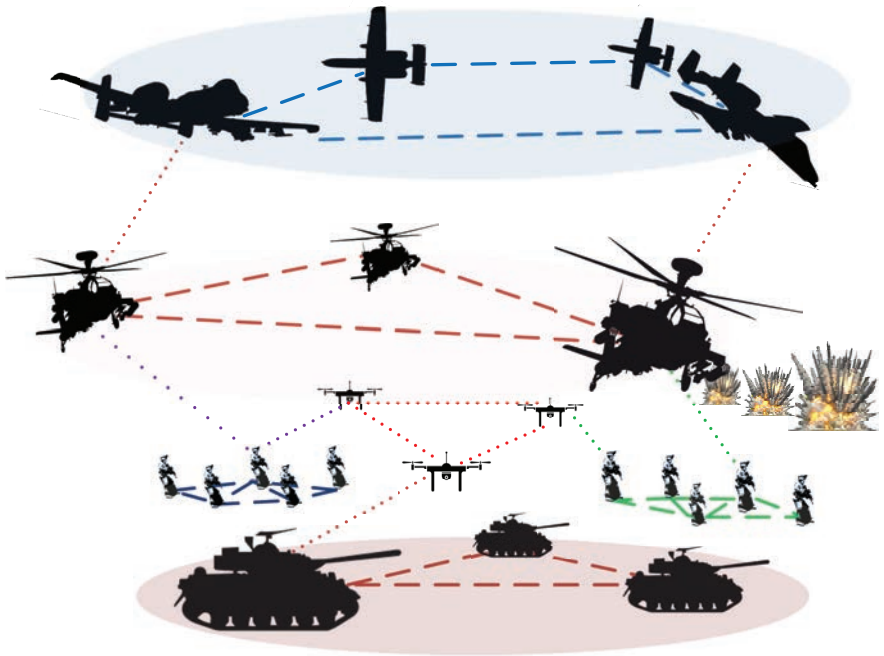


Figure 2.5: A sample multilayer IoBT ad hoc network.

The very first algorithm for calculating a backbone for multilayer military networks is described in [110], where a distributed backbone establishment algorithm, namely *clPCI* was presented. It is based on the concept of CDS. The concept of a CDS was selected for the following reasons: (a) only/mainly distributed algorithms are appropriate in the battlefield, and

despite the fact that finding a Minimum CDS is a NP-complete problem even in the centralized setting, there exist some very efficient distributed algorithms for the problem; (b) the battlefield requires resilient solutions, and fortunately the scientific knowledge on how to build a multi-connected, multi-dominating CDS is quite mature [120]; and finally (c) an IoBT is composed by many energy-starving devices, and therefore energy-aware solutions (e.g., sleep scheduling) is almost mandatory; again, the theory on how to create multiple dominating sets, i.e., domatic partitions [121] is quite rich. Until before the introduction of *clPCI* for constructing a connected dominating set in a distributed fashion for multilayer networks, there was no prior work on the topic. This complete lack may be attributed to the mistaken assumption that a multilayer network is equivalent (from the perspective of computing a CDS) to a single layer network after ignoring layer information. Multilayer networks have also been used in biological sciences in order to model interacting networks. In [122], $FAST\text{-}MDSM$ is developed to calculate a dominating sets in multilayer networks. However, this algorithm is *centralized*, and it results in an *unconnected* dominating set, incurring also a high computation cost because it runs as an integer programming problem (Calculating a minimum (connected) dominating set after formulating it as an integer programming model is a mature technique [123]). In conclusion, all aforementioned algorithms consider only single layer networks, with the exception of [110], and of the centralized $FAST\text{-}MDSM$ which comprises a departure from this literature.

## 2.3 The Rising of the Federated Learning Methodology

The *Federated Learning* (FL) technique was firstly introduced by McMahan et al. [10] and was the alternative of a privacy preserving Machine/Deep Learning paradigm. In original form of FL, there are three basic steps in order to perform FL: *selection*, *configuration* and *reporting*. First of all, the *selection* process takes place when we select which ones the participating nodes-devices will be. After that, the aggregation technique, adopted by the server is declared and the server sends the model to all participants (*configuration* process). The nodes collaboratively train the defined neural network model, by sharing their gradients/weights and not their exact data with the server. Finally, in phase of *Reporting*, the server is informed regarding the nodes updates (about sending gradients/weights) and through the specific algorithm, namely FedAvg, it sends the renewed model to the nodes, participating.

The initial, basic model of that learning architecture consists of a set of independent nodes who train their own neural network with their own data and periodically (synchronously or asynchronously) send their neural network's (i.e., model) parameters (either weights, or gradients or any other relevant information) to a server who performs the averaging of the parameters, and then returns the averaged model back to the nodes (clients). The new model is adopted by the clients and a new round of *local* training, and then parameter sharing takes place until convergence (see Figure 2.6). A plethora of variations and advances to the initial model has

been proposed in the literature [124–127].

### 2.3.1 Problem Formulation

More elaborately, let's assume that every node $i$ produces a dataset $D_i$ with $|D_i|$ number of data points. Each data point $(\mathbf{x}, y) \in D_i$ consists of a multi-dimensional feature vector $\mathbf{x} \in R^m$ and a label $y \in R$ space. We, also assume that the $f(\mathbf{x}, y, \mathbf{w})$ declare the loss function, regarding the data point $(\mathbf{x}, y)$, based on the learning model parameter vector $\mathbf{w} \in R^m$. Then, the local loss function at node $i$ is defined as:

$$F_i(\mathbf{w}) = \frac{1}{|D_i|} \sum_{(\mathbf{x},y)\in D_i} f(\mathbf{x}, y, \mathbf{w}). \tag{2.2}$$

and the total loss function is then defined as the average loss of the aforementioned local losses, as follows:

$$F(\mathbf{w}^*) = \frac{1}{|D|} \sum_{(\mathbf{x},y)\in D} f(\mathbf{x}, y, \mathbf{w}). \tag{2.3}$$

where, the goal is to find the optimal learning parameters $\mathbf{w}^\star$ for $F$, so that the total loss function is minimized, as follows:

$$\mathbf{w}^\star = argmin_{\mathbf{w}\in R^m} F(\mathbf{w}). \tag{2.4}$$

### 2.3.2 The distributed approach of Federated Learning

Although the Centralized federated learning paved the way for on-device secure learning, there are however environments – mostly wireless – where the existence of a central coordinating server is problematic for various reasons. For instance in Internet of Things [128] or in Internet of Battlefield Things [5, 129] applications, the existence of a central coordinating server is not a viable or realistic assumption due to excessive communication or fast energy depletion of the nodes close to the server because they relay the data of the rest of the nodes to the server. In such situations, the architecture of federated learning needs to be reconsidered. Very recent proposals towards alleviating this problem focus on a hybrid (or semi-decentralized) architecture [22, 130, 131] where group (clusters) of nodes collaborate to local train a model and periodically send their averaged models to the server to perform global aggregation. This model also suffers from the aforementioned problems. The work described in [132] has considered the purely distributed nature of wireless ad hoc networks, but it limits itself to determining which wireless links could/should activate concurrently, so as to avoid interference. Similarly, a purely distributed model with only device-to-device communication has been described in [133], but this work produces excessive communication since it does not use clustering of the wireless network. Similar efforts can be found in the surveys reported in [134–136]. The most close work

to this thesis is the work described in [137], where purely device-to-device communication is assumed, but that work is tailored to work in a function space.

Also, there is a very rich literature on client participation in federated learning schemes. Some works address it a secretary selection problem [25], others as a sampling scheme [138], other based on the workload-completion capability of the nodes [24], and so on [139–141].

In this thesis, we depart from all the aforementioned works, since we address it as a problem of detecting data-dissimilarity among clients. Specifically, we develop a federated learning scheme for wireless ad hoc IoT environments a) assuming a purely distributed scheme without a "central" coordinating server thus avoiding excessive communication and fast energy depletion of nodes, b) avoiding a completely flat wireless network thus preventing an all-to-all communication pattern, and c) deciding the client participation in the federation based on data-similarity with its neighboring clients rather than adopting data-agnostic methodologies.

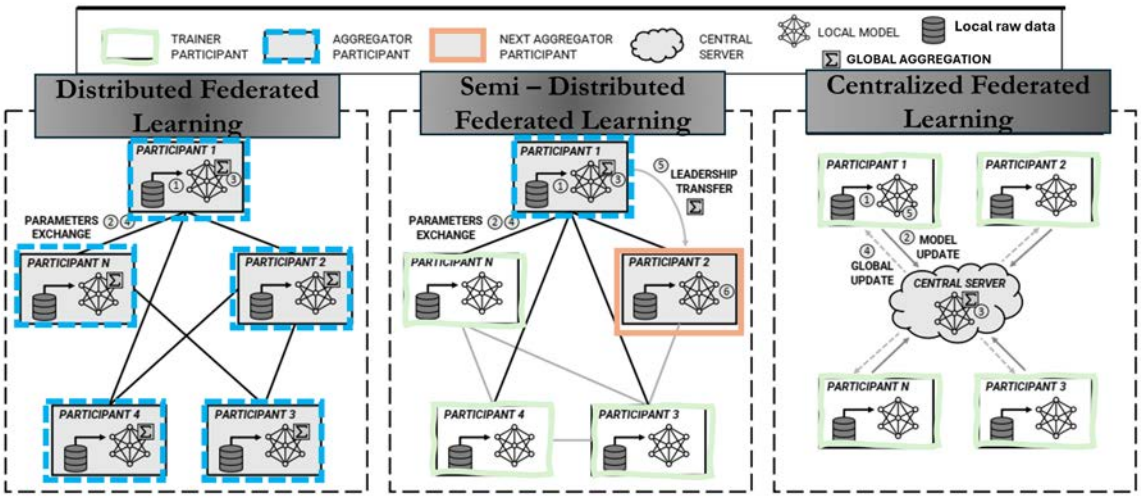



Figure 2.6: An illustration of the three main categories of FL architectures by Beltran et al. [2]. The numbers depicted, in the figure signify the training lifecycle.

Chapter 3

# Building Network Architecture for Learning

## 3.1 Multilayer Backbones: The paradigm of IoBT

### 3.1.1 Introduction

The progress in Internet of Things (IoT) inevitably impacted upon the modern battlefield, which is populated by thousands of "things", such as humans, sensors, vehicles, unmanned aerial vehicles (UAVs), aircrafts, etc. carrying out various tasks including environmental sensing, communicating, acting in isolation and/or in cooperation [108, 142]. Therefore, the Internet of Battle(field) Things [108] or the Internet of Military Things (IoMT) was born, which interconnects "devices" aiming to meet multiple and diverse missions, to operate in a (semi)autonomic mode, and to execute battlefield operations supporting end-to-end control and command; its ultimate goal is to carry out a commander's intent in a safe, responsive and resilient manner. Thus, IoBT presents at the same time all of the following significant and very challenging characteristics [143] which distinguish it from traditional IoT:

- *Diversity in tasks and aims.* There will be many networks operating simultaneously to achieve a particular goal, e.g., tracking, surveillance, attack.
- *Operation in dynamically changing and resource starving environments.* Some "devices" of the IoBT network might be energy-starving (sensors, drones), others might not have energy issues (armored vehicles), others might be obliged to travel in non-chartered territories (e.g., planes).

[0]Related publication [J4]:E. Fragkou, D. Papakostas, T. Kasidakis, D. Katsaros, *Multilayer backbones for internet of battlefield things*, Future Internet (MDPI), vol.14 (6), 2022, pp.186. & **[C4]**:D. Papakostas, T. Kasidakis, E. Fragkou, D. Katsaros, *Backbones for internet of battlefield things*, Procedings of 16th Annual Conference on Wireless On-demand Network Systems and Services Conference (WONS), 2021, pp. 1—8.

- *Extreme device heterogeneity.* IoBT networks are expected to include from tiny little sensors to some as big as armored fighting vehicles.

- *High variance in network's density and size.* An IoBT network might be comprised e.g., by the cluttered network of a swarm of drones, or by the "union" of the ad hoc network of a battalion's soldiers and the ad hoc network of a tank platoon.

The so-called *assured synthesis* is one of the major challenges identified for IoBT, and in particular the *recruitment* and *network composition* [143] tasks. The former task is about the discovery of cyberphysical assets, the human assets and their particularities, and also about the resilience to adversaries. The latter task is about the issue of dictating the nodes that must be considered in order for the requirements and constraints of the planned mission to be satisfied.

### 3.1.2 Motivation and Contributions

In [110], Papakostas et al. dealt with the heterogeneity of an IoBT network and modeled it as a *multilayer network*. It is analytically proven that if it is treated as a plain union of independent subnetworks, then efficient solutions for facilitating fast information routing are not reaped. So, despite the past, very rich literature on the domination concept in ad hoc networks, the problem in IoBT networks is different, since these are multilayer networks and considering them as a single (flat) network or considering each layer in isolation and calculating dominating set produces either suboptimal or bad solutions. Therefore, the whole past literature is in principle inappropriate. Later, energy issues were incorporated in this first work [111], and Papakostas et al. devised algorithms with fact-finding methods to address aspects of social sensing, e.g., characterize human assets/sources. They note here, that in [110] they developed distributed, communication-efficient algorithms based on the concept of node domination for choosing a set of nodes to be a backbone for a multilayer network, i.e., the IoBT network. In the biological sciences area, the work described in [122] developed centralized algorithms for computing dominating sets in multilayer networks.

Nevertheless, neither the algorithms developed in [110] (and of course in [111]) nor those developed in [122] can adequately serve the purposes of an IoBT backbone network. The main issue with the former algorithms is that they do not produce very compact network spanners, so there is plenty of room for improvement with regard to the produced connected dominating set. This is very significant if we wish to design solutions scalable to enormous IoBT networks. The two main disadvantages of the latter algorithm is its centralized nature, and thus it can not work in IoBT, and additionally because it produces unconnected spanners, and therefore it can not guarantee network-wide flow of information. Thus, this thesis contributes the following:

- It establishes the computation complexity (NP-completeness) of the problem of calculating a minimum connected dominating set for multilayer networks.

- It presents a new distributed algorithm, namely the *Cross layer Connected Dominating Set* ($CCDS$) for calculating connected dominating sets for IoBT networks, by applying an efficient mechanism to reduce the size of the dominating set.
- It enhances $FAST\text{-}MDSM$ [122] so as to produce connected dominating sets thus getting a new algorithm called $FAST\text{-}CMDSM$, and compares our proposed algorithm against the only existing competitors, namely the algorithm in [110] and $FAST\text{-}CMDSM$.

The rest of this section is organized as follows: subsection 3.1.3 formulates the problem and proves its complexity, and subsection 3.1.4 develops distributed solutions for it. Subsection 3.1.6 evaluates the competing algorithms, and finally, subsection 4.1.6 summarizes presented work.

### 3.1.3 Formulation of the Problem

A dominating set (DS) [144] of a graph (i.e., the set of *dominators*) is defined as a subset of the nodes with the property that the rest of the nodes are adjacent (i.e., 1-hop neighbors) from some dominator(s). If the network induced by the DS is additionally connected, then the DS is called connected DS (CDS). In the IoBT setting, we seek for minimum CDS (MCDS), i.e., CDS with minimum cardinality. We treat an IoBT network as a multilayer network, i.e., as a multilayer graph [110, 112]. We assume that there exist only bidirectional links (In principle, the paper ideas can be applied to networks with unidirectional links as well). A multilayer network comprised of $n$ layers is a pair $(G^{ML}, E^{ML})$, where $G^{ML} = \{G^i, i = 1, \ldots, n\}$ is a set of networks $(G_i, E_i)$ (i.e., with $G_i$ nodes and $E_i$ links), and a set of interlayer links $E^{ML} = \{E_{i,j} \subseteq G_i \times G_j; i, j \in \{1, \ldots, n\}, i \neq j\}$. Figure 2.5 illustrates such a network, where we can see a *layer* of soldiers, a layer comprised by helicopters, the *intralayer* links connecting "nodes" of the same layer, and *interlayer* links connecting "nodes" belonging to different layers.

Apparently, the requirement of calculating a CDS with minimum cardinality stems from scalability issues [143]. Additionally, from the discussion in Section 3.1.1 and in ([143] Section III.B), where "... resilience and latency requirements for synthesizing a near-optimal network" [143] are emphasized, we conclude that the more nodes with many interlayer links are included in our IoBT backbone the better it is. The inclusion of many nodes with "a lot" of interlayer links supports low-latency communication among layers; we can consider them as the hubs encountered in the literature on complex networks that reduce the "degrees of separation". Moreover, the existence within the backbone of many nodes with "a lot" of interlayer links, reduces the danger of partitioning among layers. Therefore, we provide Definition 3.1.1 for the problem of calculating a MCDS for IoBT networks, and establish its computational complexity (Proposition 3.1.1) in the centralized setting, closing a gap in the literature which is open since [110] and it was not dealt with in [5].

**Definition 3.1.1** (Multilayer MCDS problem for IoBT)**.** *Solve the $MCDS$ for a multilayer graph in a distributed manner, i.e., calculate the set $MCDS^{ML}$ comprised by the minimum*

*number of nodes with the following properties: (a) their induced subgraph is connected (with intra and/or inter-layer links) and the rest of the nodes are adjacent to one node (or more) belonging to $MCDS^{ML}$, (b) maximize the number of dominators with many interlayer links, (c) knowing only the k-hop neighborhood of each node; we work with $k = 2$ here. (Working with broader neighborhoods (i.e., $k > 2$) would cause a severe broadcast storm problem [145] in order to acquire the topology deploying successive rounds of "beacon" messages).*

In the rest of this section, we will investigate and establish the computational complexity of the centralized version of our problem for connected dominating sets. In particular, we will define the decision and optimization version of the examined problem, and then establish their complexities. The validity of these results for the case of connected dominating sets calculated in a distributed fashion is straightforward.

**MULTILAYER CONNECTED DOMINATING SET PROBLEM**
INSTANCE: A positive integer $K$, and a multilayer network consisting of $n$ layers, i.e., a set of $n$ pairs $(G_i, E_i)$, where $G_i$ is the node set of a usual network and $E_i$ is a set of edges, for $1 \leq i \leq n$, and a set of interlayer links $E^{ML} = \{E_{i,j} \subseteq G_i \times G_j; i, j \in \{1, \ldots, n\}, i \neq j\}$.
QUESTION: Is there a dominating set of size $K$ for $(G^{ML}, E^{ML})$, i.e., a subset $V' \subseteq (\cup V_i)$ with $|V'| = K$ such that for all $u \in (\cup V_i) - V'$ there is some $v_j \in (\cup V_i)$ for which $(u, v_j) \in ((\cup E_i) \cup E^{ML})$, and moreover the induced subgraph defined by $V'$ is connected?

**MULTILAYER MINIMUM CONNECTED DOMINATING SET PROBLEM**
INSTANCE: A multilayer network consisting of $n$ layers, i.e., a set of $n$ pairs $(G_i, E_i)$, where $G_i$ is the node set of a usual network and $E_i$ is a set of edges, for $1 \leq i \leq n$, and a set of interlayer links $E^{ML} = \{E_{i,j} \subseteq G_i \times G_j; i, j \in \{1, \ldots, n\}, i \neq j\}$.
QUESTION: What is the minimum cardinality dominating set for $(G^{ML}, E^{ML})$, i.e., what is the minimum cardinality subset $V' \subseteq (\cup V_i)$ such that for all $u \in (\cup V_i) - V'$ there is some $v_j \in (\cup V_i)$ for which $(u, v_j) \in ((\cup E_i) \cup E^{ML})$, and moreover the induced subgraph defined by $V'$ is connected?

We can now proceed to establish the computational complexity of the second problem. Our proof method will also establish the complexity of the first problem as well. The result is described in Proposition 3.1.1, but firstly we remind some background on proving the computational complexity of problems. So, a problem is NP-complete if the following two conditions hold:

(a) we can prove that the problem belongs to the class of NP problems, *and*

(b) we can

(b1) *either* transform in polynomial-time a known NP-complete problem to the problem at hand ([146] p. 45), *or*

(b2) prove that the problem at hand contains a known NP-complete problem as a special case ([146] p. 63, Section 3.2.1); this is called the "restriction" approach to proving NP-completeness.

We will now construct the proof that the problem of finding a minimum connected dominating set for $(V, E_i)$ is NP-complete. Recall that our problem is an *optimization* problem, i.e., "... finding the *minimum* dominating set".

**Proposition 3.1.1.** *The Multilayer MDS problem is NP-complete.*

**Proof.** Our proof consists of three steps: (a) Transform the problem into a decision version, (b) Prove that it belongs to the class of NP problems. (c) Establish its NP-completeness complexity by following the "restriction" approach, i.e., by showing that our problem contains a known NP-complete problem as a special case.

[**STEP 1**. Transform our optimization problem into the decision version of the same problem.] Without violating the validity of the proof, we will work with the *decision* version of the problem:

*Given an integer $k > 0$, does our multilayer network contain a connected dominating set of cardinality equal to $k$?*

Apparently, the smallest $k$ for which the answer to the above problem is 'yes' is the size of the minimum connected dominating set of our multilayer network.

From the perspective of computational complexity, the two problems are equivalent (see item (b1) above): With the use of binary search, we need to solve the decision version for $O(log(|\cup_i G^i|))$ different values of $k$, i.e., which is a polynomial time of searches.

[**STEP 2**. Proving that the decision problem belongs to the NP class.]

The MULTILAYER CONNECTED DOMINATING SET PROBLEM is clearly in NP, as given a graph $(G^{ML}, E^{ML})$, a set $S \subseteq \cup_i G^i$ of nodes, and a number $k$, we can test if $S$ is a connected dominating set of $(G^{ML}, E^{ML})$ of size $k$ or less by first checking if its cardinality is less than or equal to $k$ and then checking if every node in $(G^{ML}, E^{ML})$ is either in $S$ or adjacent to a node in $S$. This process clearly takes polynomial time, i.e., $O(|\cup_i G^i| + |\cup_i E^i| + |E^{ML}|)$. Should we wish to check whether $S$ is actually connected, we can resort to any polynomial-time algorithm, e.g., breadth/depth first search. Therefore, the MULTILAYER CONNECTED DOMINATING SET PROBLEM is indeed in NP.

[**STEP 3**. Proving that the decision problem contains a known NP-complete problem as a special case.]

The third step of our proof involves proving that the MULTILAYER CONNECTED DOMINATING SET PROBLEM contains as a special case the problem of finding a connected dominating set for a single layer graph $G(V, E)$—a known NP-complete problem ([146] p. 190, Problem GT2).

We state the following corollary:

**Corollary 3.1.1.** *From ([110] Theorem 1), it follows immediately that if we wish to connect with each other two connected dominating sets of two separate graphs by adding a single edge among the two graphs, then we will need at most 2 more nodes (one from each separate graph) to be included in the single "united" connected dominating set.*

We assume the existence of a single layer graph $G(V, E)$, and we construct a multilayer graph $(G^{ML}, E^{ML})$ with two layers, where $G^{ML} = \{G^1, G^2\} = \{(V, E), (V, E)\}$ and $E^{ML}$ is any non-empty set of interlayer edges between $G^1$ and $G^2$. Then, we erase all but one interlayer edges from $E^{ML}$. We assume that this edge is the $(\alpha, \beta)$ with $\alpha \in G^1$ and $\beta \in G^2$.

Then, the problem of finding whether the graph $(G^{ML}, E^{ML})$ contains a connected dominating set of size $2k+|\{\alpha, \beta\}| = 2k+2$ is in an obvious one-to-one correspondence with the problem of finding whether the graph $G(V, E)$ contains a connected dominating set of size $k + 1$. Note here, that there is no need—from the computational complexity perspective—for the size-$2k + 2$ connected dominating set of $(G^{ML}, E^{ML})$ to be a double-size clone of the size-$k + 1$ connected dominating set of $G(V, E)$, and thus the two problems do not need to be an exact duplicate of each other.

Thus, the MULTILAYER CONNECTED DOMINATING SET problem is NP-complete. ■

### 3.1.4 Proposed Heuristic Distributed Algorithms

#### 3.1.4.1 Distributed $CDS$ in a Multilayer Network

The calculation of a $MCDS$ by any heuristic algorithm means in practice that we are seeking for "strategically" positioned nodes in the network topology having many connections in order to decrease the size of the obtained $CDS$. In the case of IoBT networks, we seek for such nodes but with the additional property that they should have many interlayer links. So, the use of the $clPCI$ *centrality* measure [110] is a perfect fit. We exploit $clPCI$ and incorporate it into a distributed algorithm for calculating a $CDS$; we name this distributed algorithm as the *Cross layer Connected Dominating Set* algorithm ($CCDS$). $CCDS$ is composed by the *CDS construction task*, and the *redundant relay node pruning task*. As it is common in almost any distributed algorithm for wireless ad hoc networks, each node learns the topology of its neighborhood with the exchange of beacon messages. In $CCDS$, each node learns its 2-hop neighborhood $N^2(u)$; then, it calculates its $clPCI$ and broadcasts it to its neighbours and, by mutuality of the distributed algorithm, it becomes aware of its neighbors' $clPCI$ values.

The initial $CDS$ construction task is a *source-initiated* relay node selection type of algorithm, and is divided into the *neighbor prioritization subtask* and the *construction subtask*. Each node $u$ prioritizes (i.e., sorts) its 1-hop neighbors in decreasing order of their $clPCI$ values and *progressively* selects from this sorted list neighbors to include in its relay set $R(u)$ those neighbors that have the largest $clPCI$ value and that cover at least one new node in the respective 2-hop neighborhood $N^2(u)$, until all $N^2(u)$ is covered. Then, the *pruning task* follows, in order to

reduce (if possible) the size of its relay set $R(u)$. $CCDS$ uses the *restricted pruning Rule k* because it is more efficient in reducing the relay node set than several existing schemes that still ensure full broadcast coverage. Differently from this scheme, we exploit connectivity information as this is quantified by $clPCI$ in order to establish a total order among nodes that participate in the $CDS$. $CCDS$'s pseudocode is Algorithm 1.

**Algorithm 1** CCDS

**Precondition:** Known clPCI index values of nodes in $N(u)$ and $N^2(u)$
**Postcondition:** Completed MCDS election process
**Remarks:**
  $G = (V, E)$: graph where $V$ is the vertex set and $E$ is the edge set
  $R(u)$: relay node set of node $u \in V$
  $M(u)$: True/False indicator for node $u$ being a DS node
**repeat**
  Add to $R(u)$ the node $l \in N(u)$ with the largest $clPCI$ that covers at least one new node in $N^2(u)$
**until** each node in $N^2(u)$ is covered by node(s) in $R(u)$
Announce $R(u)$
**if** selected as a relay node **then**
  $M(u) \leftarrow$ True
  Announce status change
  Build $S_{(u)}^{\text{constrained}} = \{u_1, u_2, \ldots, u_n \mid u_k \in N(u) \cap N^2(u), M(u_k) = \text{True}, clPCI(u) < clPCI(u_k)\}$
  **if** $S_{(u)}^{\text{constrained}}$ satisfies $N(u) \subseteq N(u_1) \cup N(u_2) \cup \cdots \cup N(u_n)$ and $u_1, u_2, \ldots, u_n$ form a connected graph **then**
    $M(u) \leftarrow$ False
    Announce status change
    **Return** ▷ CDS Pruning
  **end if**
**end if**

#### 3.1.4.2 Centralized *CDS* in Multilayer Networks

The centralized $FAST\text{-}CMDSM$ algorithm consists of the *MDS discovery task*, the *CDS construction task*, and the *redundant DS node pruning task*. The calculation the minimum dominating set (MDS) is treated as an Integer Programming problem [122]. The $CDS$ construction task aims at adding in the $DS$ the least possible—*per node*—number of nodes, such that the 2-hop neighborhood of each node is covered, and as result the multilayer network is connected in the sense that any pair of nodes can communicate via the $DS$. In the last task, we remove any redundant DS nodes by seeking alternative paths, and we substitute information flow via these paths. $FAST\text{-}CMDSM$'s pseudocode is Algorithm 2.

**Algorithm 2** FAST-CMDSM Algorithm

**Precondition:** All nodes are designated as dominators
**Postcondition:** Completed MCDS election process
**Remarks:**
  $G = (V, E)$: graph where $V$ is the vertex set and $E$ is the edge set
  $M(u)$: True/False indicator for node $u$ being a DS node
  $Vm$: DS node set
  $MDS_V$: Minimum DS node set of $V$
  $CDS_V$: Connected DS node set of $V$
**repeat**
  **if** $\exists u_j \in V \mid d(u_j) = 1$ and $u_i \in N(u_j)$ for $i \neq j$ **then**
    $M(u_i) \leftarrow$ True
    **if** $u_i \notin Vm$ **then**
      Add $u_i$ to $Vm$
    **end if**
  **end if**
**until** all nodes at every layer have been examined
**repeat**
  **if** $M(u_j) =$ True **then**
    **if** $\exists u_i \in N(u_j) \mid M(u_i) =$ True **then**
      $M(u_j) \leftarrow$ False
      **continue**
    **else**
      **if** $u_j \notin Vm$ **then**
        Add $u_j$ to $Vm$
      **end if**
    **end if**
  **end if**
**until** no more nodes are added to $Vm$
$MDS_V \leftarrow$ **Minimize** $\sum_{i=1}^{n} x_i$
**Subject to**: $x_i + \sum_{(u_j,u_i)\in E_k} x_j \geq 1 \quad \forall u_i \in V_k$
**repeat**
  Add to $CDS_V$ the least possible nodes from $N(u)$ that are needed to cover the $N^2(u)$ neighborhood
**until** any node $u \in MDS_V$ has been examined
$CDS_V \leftarrow Vm \cup (CDS_V - (CDS_V \cap Vm))$
Use pruning to decrease the size of $CDS_V$

### 3.1.5 Computation and Communication Complexities

#### 3.1.5.1 Complexities of $CCDS$

$CCDS$ requires 7 rounds of communication among nodes to complete. In each round, at most one packet is sent over the wireless channel.

- The 2-hop neighborhood information used by the relay node set election process is collected via two rounds of information exchanges.
- In round 1, each node advertises its ID and builds its 1-hop neighbor set based on the advertisement of its neighbors.
- In round 2, each node advertises its 1-hop neighbor set and identifies links among 1-hop neighbors.
- In round 3, each node calculates its clPCI index value and advertises it together with its 2-hop neighbor set. Then, it identifies links among 2-hop neighbors.
- In round 4, each node calculates and advertises its own relay node set and updates 1-hop neighbor status.
- In round 5, the restricted Rule-$k$ is applied to each relay node and each one of them advertises its status.
- In round 6, each node advertises its updated relay node set
- In round 7, the composition of the updated relay node set is advertised (if needed).

The computation complexity of its comprising parts is: $O(\Delta^2)$ for the $clPCI$ calculation, and $O(\Delta^3)$ for the relay node set election process and for the pruning phase, where $\Delta$ is the maximum node degree found in the network.

#### 3.1.5.2 Complexities of $FAST$-$CMDSM$

$FAST$-$CMDSM$ is a centralized algorithm and so its communication complexity is not an issue for investigation.

$FAST$-$CMDSM$' s computation complexity is exponential, similar to all branch-and-cut integer programming solvers.

### 3.1.6 Performance Evaluation

**Competing Algorithms.** We compare the proposed algorithms, namely $CCDS$ and $FAST$-$CMDSM$, but in order to explain the usefulness of their pruning procedures, we develop and include in our comparison their annotated with an asterisk version of them, namely $CCDS^*$ and $FAST$-$CMDSM^*$; these versions are simply the same algorithms but without activating the pruning procedure. Solely for illustrative purposes we include the experimentation, and $FAST$-$MDSM$ algorithm which constructs minimum but *unconnected* $DS$; Papakostas et al. proved in [110] that any (unconnected) $DS$ of size $\|DS\|$ can be turned into a $CDS$ by adding $2 \times \|DS\|$ additional nodes in the $DS$ in the worst case. Table 3.1 summarizes the competing algorithms.

Table 3.1: Competitor characteristics.

| Competitor | CDS Calculation | Pruning Heuristic | Complexities CPU [a] / Comm [b] | |
|---|---|---|---|---|
| $CCDS$ | Distributed | ✓ | $\Delta^3$ | / 7 |
| $CCDS^*$ | Distributed | - | $\Delta^3$ | / 7 |
| $FAST\text{-}CMDSM$ | Centralized | ✓ | $exp$ | / $c$ |
| $FAST\text{-}CMDSM^*$ | Centralized | - | $exp$ | / $c$ |
| $FAST\text{-}MDSM$ | Centralized | - | $exp$ | / $c$ |

[a] $\Delta$ is the maximum node degree; [b] Number of transmitted messages per node; [c] Not applicable due to its centralized nature.

**Simulation testbed.** Due to the lack of available, real military networks, and the inability (The requirement of modern battlefields is to able to operate ad hoc networks consisting of an order of magnitude more nodes; for instance a battalion would need a thousand nodes. e.g., `https://www.darpa.mil/news-events/2013-04-30` (accessed on 15 May 2022)) of wireless testbeds and emulation environments for ad hoc networks to deal with several hundreds of nodes, we developed a generator for multilayer networks in MATLAB. The details of our generator can be found in [5, 111], and here we present its basic features. The construction of a multilayer network is driven by the average node degree, by the nodes' number per layer (i.e., the so-called layer size), and the number of layers. The interconnection of the layers is driven by: (a) the number of a node's links towards nodes in different layers, and (b) the distribution of the interconnections to the nodes within a particular layer. We apply the *Zipfian* distribution to our connectivity generator. Skewness is managed by parameter $s \in (0, 1)$. We make use of four distinct *Zipfian* distributions, one per parameter of interest:

- $s_{degree}$: to generate the frequency of appearance of highly interconnected nodes,
- $s_{layer}$: to choose how frequently a specific layer is selected,
- $s_{node}$: to choose how frequently a specific node is selected in a specific layer.
- $s_{weight}$: to choose how uniformly the energy is distributed in the nodes.

We call these parameters as the *topology skewness*, and represent it as a sequence of four floats, meaning that $s_{degree} = 0.5$, $s_{layer} = 0.5$, $s_{node} = 0.5$ and $s_{weight} = 0.5$ (which are the default settings we used to create the datasets).

**Performance measures.** The competitors are compared in terms of the cardinality of the $CDS$; apparently an algorithm is more efficient than another, if it generates a $CDS$ having smaller cardinality [107]. Moreover, an algorithm that establishes a (per node) relay set with larger residual energy is naturally considered to be more efficient in terms of energy than another algorithm whose per node relay set has less residual energy.

**Datasets.** We created networks which vary with respect to the topology density, the network diameter, the number of network layers and their size. The topology density's impact on the performance is evaluated with 4-layer networks. Each layer consists of 50 nodes, and the mean node degree is 3, or 6, or 10, or 16, or 20. The network diameter's impact on the performance is evaluated with 4-layer networks as well. Each layer consists of 50 nodes, and the mean node degree is 6. The diameter of each layer is 3, or 5, or 8, or 12, or 17. The number of layers' impact is examined in networks with 2, or, 3, or 4, or 5, or 7 layers. Each layer consists of 50 nodes, and the mean node degree is 6. The increase in the layer size's impact is evaluated with 4-layer networks. The "base" layer consists of 50 nodes, and each next layer is larger than the previous by 10%, or 20%, or 30%, or 50%, or 70%. The mean node degree in each layer is 6. Table 3.2 records all the independent parameters.

Table 3.2: Experimentation parameters values.

| Parameter | Range | Default |
|---|---|---|
| avg. node degree (D) | 3, 6, 10, 16, 20 | 6 |
| network diameter (H) | 3, 5, 8, 12, 17 | 5 |
| number of network layers (L) | 2, 3, 4, 5, 7 | 4 |
| size of a layer relative to its adjacent layers | 10%, 20%, 30%, 50%, 70% | - |

### 3.1.7 Experimental Evaluation

Each experiment was repeated 5 times. The variation around the mean was so negligible that the error bars are hardly recognizable in the plots. We conduct experiments with $s_{degree}$, $s_{layer}$, $s_{node}$, $s_{weight}$ parameters into $0.5 - 0.5 - 0.5 - 0.5$ (Medium skewness), and $0.1 - 0.1 - 0.1 - 0.1$ (Low skewness), and $0.9 - 0.9 - 0.9 - 0.9$ (High skewness) setting, respectively.

#### 3.1.7.1 Impact of Topology Density

Firstly, we consider the impact of topology density on the performance of each competitor. In Figure 3.1 we evaluate, for medium skewness, the *per layer* size of the $CDS$. The main conclusion is that the size of the CDS is practically a decreasing function of the node density. This happens because the higher the network density is, the greater the coverage capability of the network nodes becomes, and therefore the size of the $CDS$ gets smaller. In the case we succeed medium skewness, there is no clear winner between $CCDS$ and $FAST\text{-}CMDSM$ as the topology becomes denser (degree $\geq 6$) and both competitors present similar performance (<10% variance). The explanation is that in such topologies, there exist multiple redundant paths towards nodes, and thus both pruning mechanisms work equally well. On the contrary, in sparse topologies (degree = 3) $FAST\text{-}CMDSM$ is up to 15% more efficient. This is due to the fact that during the pruning process, the redundant paths are less in sparse topologies, and their discovery

requires knowledge of the topology further away than 2-hops, which is beyond the capabilities of localized algorithms deployed in wireless ad hoc networks. Apparently, the centralized nature of $FAST\text{-}CMDSM$ provides a clear advantage, since its pruning task has a broader overview of the topology. If we exclude the pruning task, then $CCDS^*$ and $FAST\text{-}CMDSM^*$ present similar performance (less than 10% variance) when degree $\leq$10, and the performance of the latter is up to 15% better to the former when degree >10. However, these results are not good news, because both algorithms do not perform well in terms of the—*per layer*—CDS size; i.e., the—*per layer*—$CDS$ size is up to 98% of that of the total number of nodes in that layer. This is considered natural in multilayer networks when the traditional methods are used (2-hop neighborhood coverage). $DS$ redundancy justifies the use of the pruning mechanism (up to 88% and 85% $CDS$ size reduction for $CCDS$ and $FAST\text{-}CMDSM$, respectively, in this particular case).

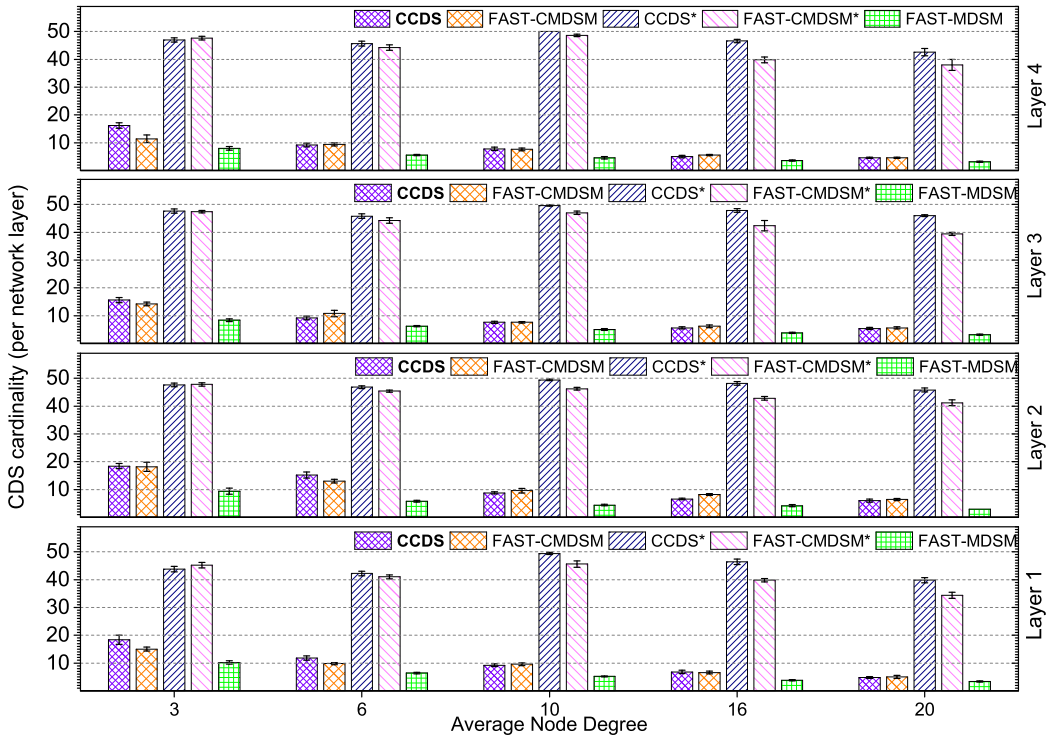



Figure 3.1: Impact of topology density (Medium skewness).

We expanded our experiments in case of low skewness. The results, depicted in Figure 3.2 show that there are no important differences regarding the efficiency, between $CCDS$ and $FAST\text{-}CMDSM$, when topology becomes denser (degree $\geq$ 6). As mentioned above, this happens because pruning mechanisms work well, when nodes have multiple ways to connect with one another. When degree = 3 (sparse topologies), $CCDS$ is about 10% less efficient than $FAST\text{-}CMDSM$, something which is justified due to the centralized control of the $FAST\text{-}CMDSM$. In this case, the reduction size of $CDS$ is about 90% for $CCDS$ algorithm and up to 88%.

In case of high skewness (see Figure 3.3), when degree $\geq 10$, both the proposed algorithms behaves in a quite similar way (less than 10% variance), too, regarding the size of $CDS$. Here, $FAST\text{-}CMDSM$ outperforms $CCDS$, due to the centralized control of the first one which makes the procedure of finding surplus paths easier inside the Multilayer network. Examining the case when topology is sparse (degree = 3), we see that $CCDS$ is less efficient than the other pruning proposed algorithm, bearing in mind that due to the lack of enough connections, it is more difficult for a distributed algorithm to find minimum $DS$. Also, when degree = 6, $CCDS$ continues to outperform $FAST\text{-}CMDSM$, but both algorithms create $DS$ with smaller size than the above implementations do. So, in this case the efficiency of the proposed algorithms are up to 86% and up to 82% for both $CCDS$ and $FAST\text{-}CMDSM$, respectively.

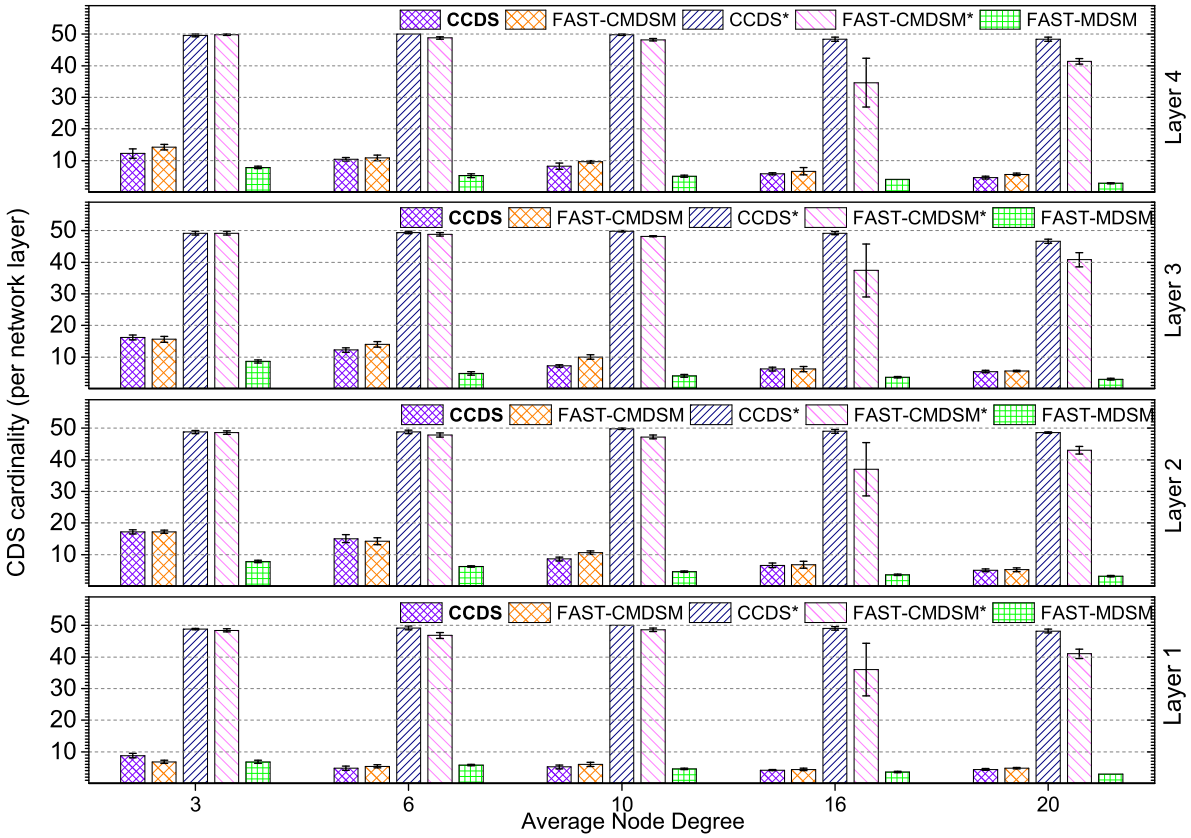



Figure 3.2: Impact of topology density (Low skewness).

So, as it is also shown in Figure 3.4 which shows CDS sizes aggregated over all layers, for all the parameters tested, the results are quite similar regarding both $CCDS$ and $FAST\text{-}CMDSM$ algorithm, with the corresponding results with parameters $0.1 - 0.1 - 0.1 - 0.1$ to be a little more efficient, probably due to the more uniform distribution this case implements.

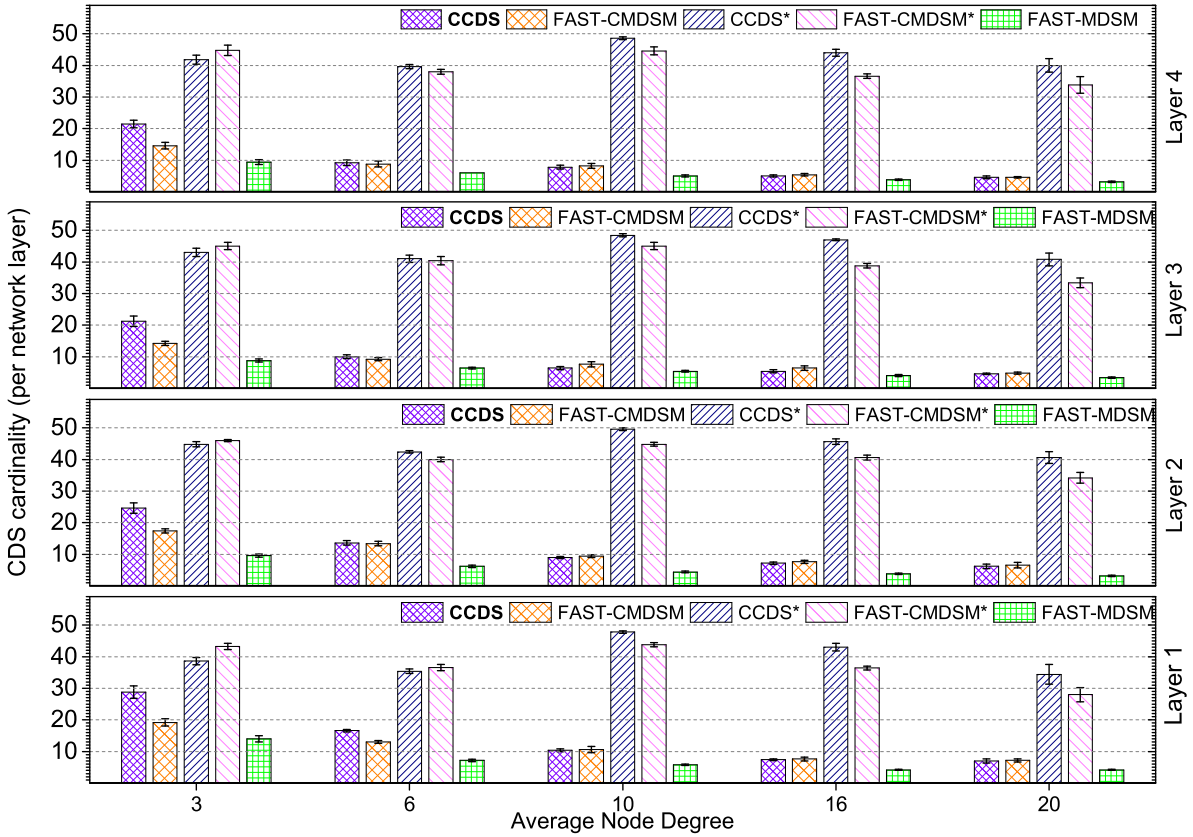



Figure 3.3: Impact of topology density (High skewness).

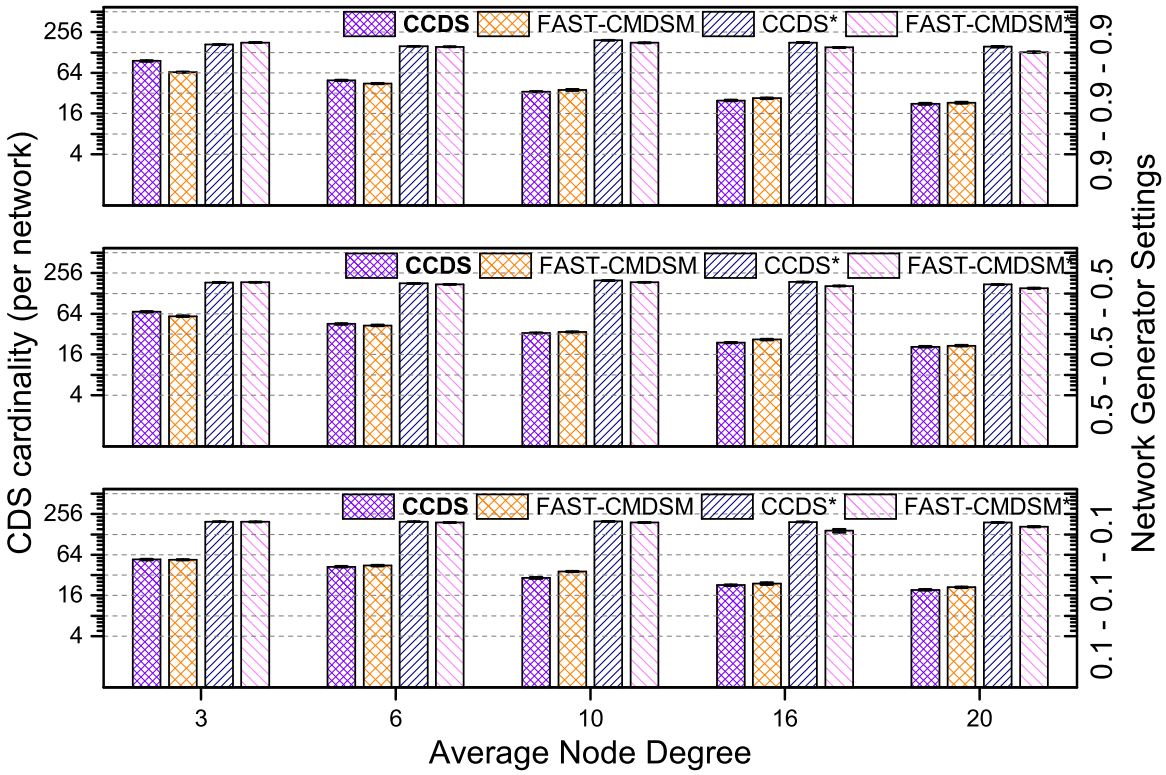



Figure 3.4: Impact of topology density (CDS size aggregated over all layers).

#### 3.1.7.2 Impact of Network Diameter

In Figures 3.5–3.7, we evaluate the effect of the network diameter in the size of the $CDS$. The main conclusion is that as the network diameter increases the size of the resulting $CDS$ increases, and this observations is valid for all competitors. In case of medium skewness (Figure 3.5), the diameter increases and also the topology gets sparser. So, we have fewer, longer (in hops), and less redundant paths. In that way, the selection of 1-hop neighbor nodes that cover the $N^2$ neighborhood of a particular node requires more 1-hop neighbors. Now, in terms of the competitors' performance, we see that in bushy topologies (diameter $\leq 5$) $CCDS$ presents up to 18% smaller $CDS$ compared to $FAST\text{-}CMDSM$. This gain is attributed to the employment of the $clPCI$ by $CCDS$ and which helps in getting a better pruning. On the other hand, in bushy topologies an unfortunate erase from the $CDS$ of a strategically located $DS$ node will probably result in keeping a lot of (practically "useless") $DS$ nodes just to ensure connectivity. When diameter = 8 or diameter = 12 the competitors have similar performance (less than 10% variance). As expected, in long and skinny topologies (diameter = 17) $FAST\text{-}CMDSM$ outperforms $CCDS$ by 16%, due to its centralized nature. Examining the pruning-free versions of the competitors, we see that both of them practically exhibit the worst-case behaviour, i.e., almost all nodes are selected as $DS$ nodes; the performance of the pruning mechanism for $CCDS$ and $FAST\text{-}CMDSM$ regarding the $CDS$ size reduction is up to 83% and 79%, respectively.

In case of low skewness as it is illustrated in Figure 3.6, we can also say that when diameter $\leq 5$, $CCDS$ algorithm outperforms FAST-CMDSM and when diameter = 8 or diameter = 12, both proposed algorithms have similar performance (less than 10% variance). Finally, we see that for diameter = 17, $FAST\text{-}CMDSM$ has a better performance than $CCDS$, due to the benefits of its centralized form, in long paths. So, in this case, $CDS$ reduction is up to 85% for $CCDS$ and 81% for $FAST\text{-}CMDSM$.

Finally, when the interconnectivity generator parameters are $0.9 - 0.9 - 0.9 - 0.9$ (high skewness), as it is depicted in Figure 3.7, we observe a general increase in $CDS$ size, due to the fact that the degrees of the nodes have a great variance in this case and as a result, more nodes needed to join the $DS$. When diameter = 3, $CCDS$ has better performance than the other proposed algorithm, due to the fact that we have smaller paths to check and as a result the distributed algorithm can discover easier the redundant paths (2-hop coverage). When diameter = 5, or diameter = 8, $FAST\text{-}CMDSM$ starts to have a better performance, as it creates $CDS$ with smaller size. This is expected, since the centralized control of this algorithm contributes efficiently in finding the redundant paths. The longer the diameter is, the better performance $FAST\text{-}CMDSM$ has as it is shown in Figure 3.7. To sum up, in this case, $CCDS$ can achieve up to 81% reduction in the size of total $CDS$, while $FAST\text{-}CMDSM$ can achieve 79%. As a summary, we provide Figure 3.8 to show average performance of the competitors.

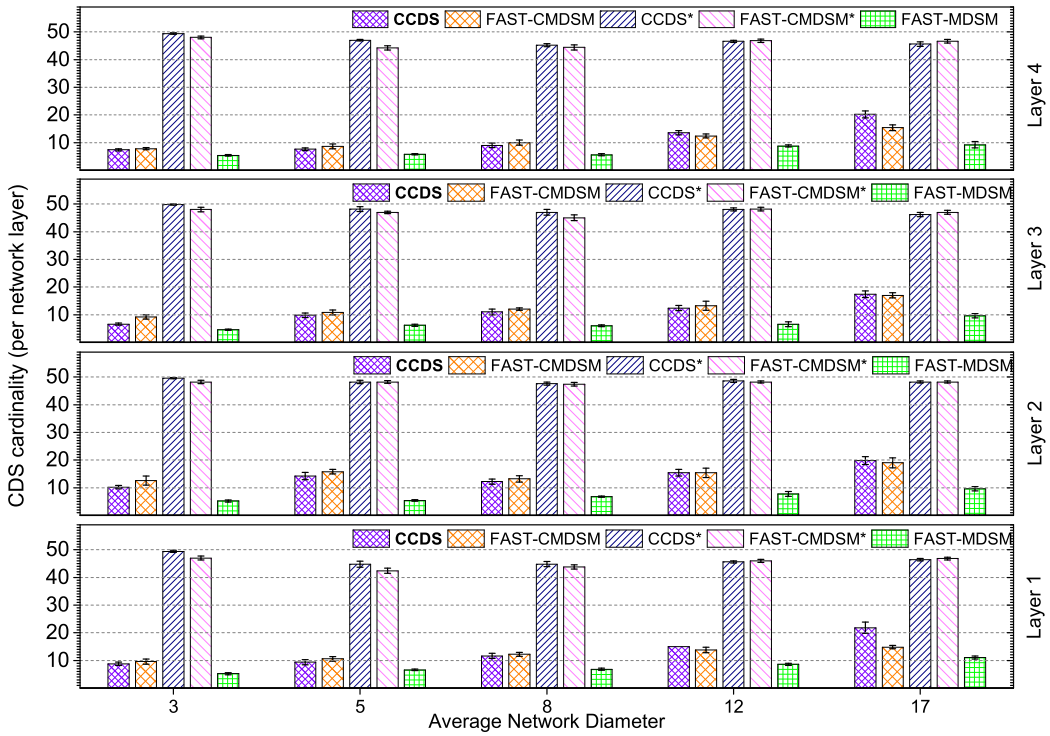



Figure 3.5: Impact of network diameter (Medium skewness).

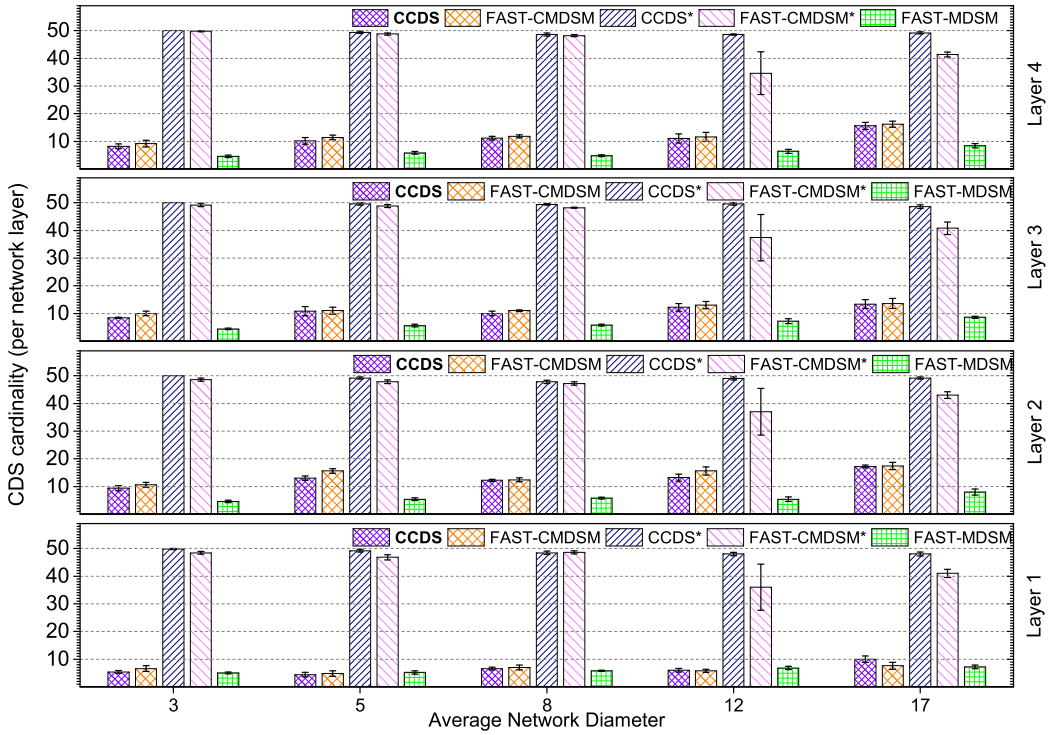



Figure 3.6: Impact of network diameter (Low skewness).

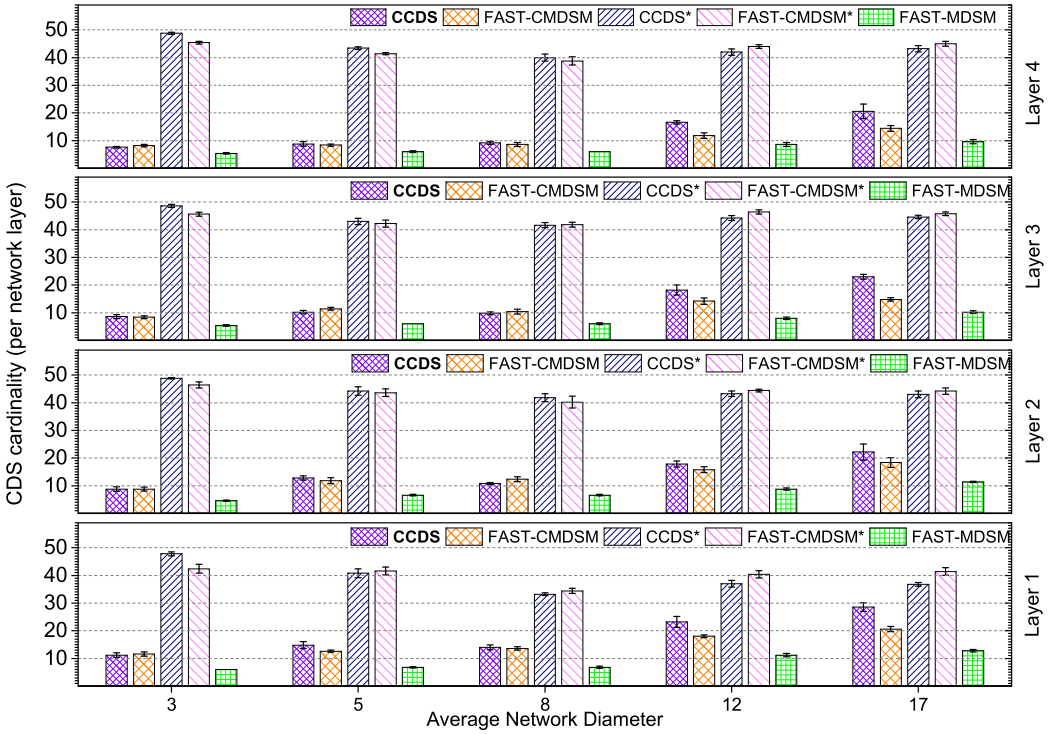



Figure 3.7: Impact of network diameter (High skewness).

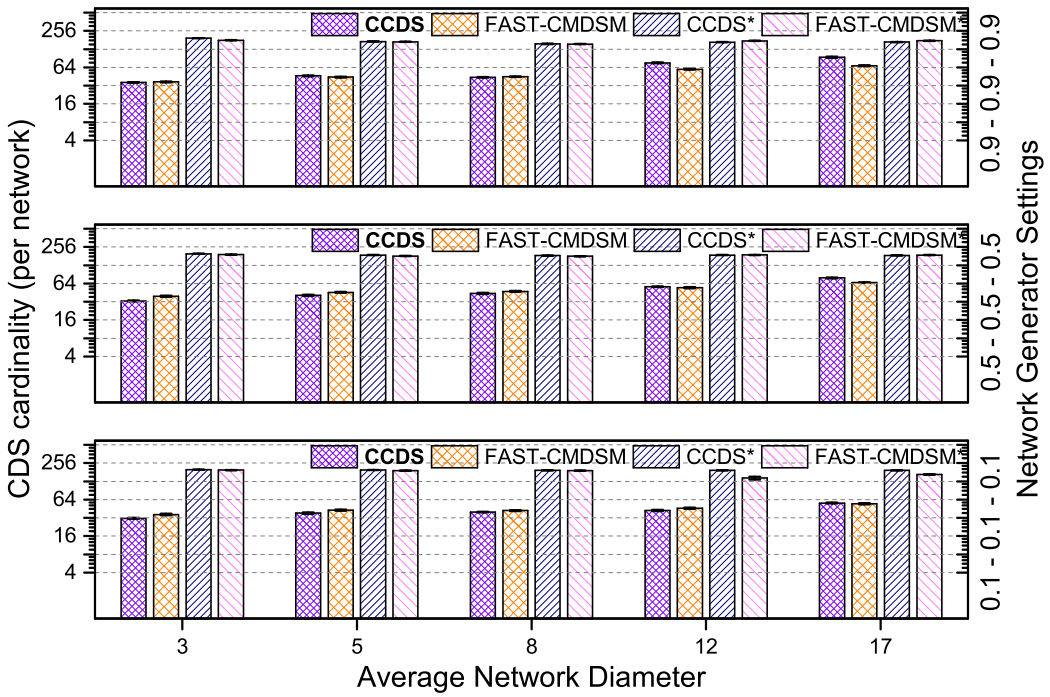



Figure 3.8: Impact of network diameter (CDS size aggregated over all layers).

#### 3.1.7.3 Impact of the Number of Layers

We investigated the impact of the network layers' number on the competitors. In Figure 3.9 we show the *per layer* CDS cardinality for the competing algorithms (for medium skewness). The first observation we make is somewhat counter-intuitive: we see that the *per layer* CDS cardinality is independent on the number of layers(!) even though we would expect to be a decreasing function of the number of layers, because when a multilayer network has more layers, then it will (most probably) have more connections among layers (i.e., interlayer links), and thus the network will become more dense. So, even though we expected an increase in the coverage capability of the nodes, this does not happen, and it is attributed to the generic topology of the network. Turning now our focus on the competitors, we observe that with 5 or less layers both *CCDS* and *FAST-CMDSM* perform very good (10% or less variance). However, *CCDS* is the best of the two presenting a 14% better performance. Overall, the obtained results are consistent with our earlier which state that both competitors perform very good in dense networks. In general, the main reason for *CCDS*' superior performance with respect to *FAST-CMDSM*' s performance is the very effective pruning mechanism. When looking at the version of these algorithms without the pruning mechanism, we see that as expected they perform poor; in particular the performance of the pruning mechanism for *CCDS* and *FAST-CMDSM* regarding the *CDS* size reduction is up to 79% and 75%, respectively.

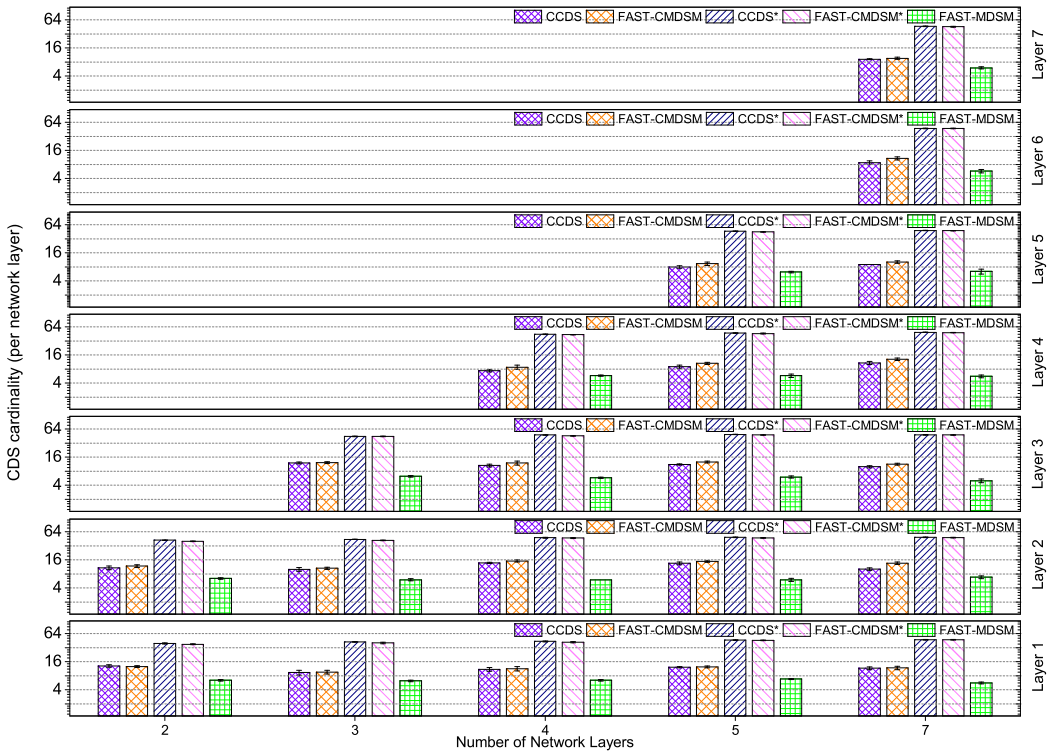



Figure 3.9: Impact of number of layers (Medium skewness).

Next, we present the case where the parameters of the interconnectivity generator are:

$0.1 - 0.1 - 0.1 - 0.1$ (low skewness). We see in Figure 3.10, that both the proposed algorithms achieve similar performance, with the $CCDS$ algorithm to be a little bit more efficient (less than 10% variance). The total reduction in $CDS$ size is up to 80% for $CCDS$ and up to 78% for $FAST\text{-}CMDSM$. The last experiment in this section is conducted by setting the respective interconnectivity generator variables to: $0.9 - 0.9 - 0.9 - 0.9$ (high skewness). In this case, when number of layers is less than 4, $FAST\text{-}CMDSM$ outperforms $CCDS$, while when the number of layers is greater than 4, then the two proposed algorithms are barely equally efficient, as it is shown in Figure 3.11. Specifically, the total reduction provided by this experiment is up to 74% for CCDS and up to 73% for $FAST\text{-}CMDSM$ (1% variance). In Figure 3.12 we show the competitors average performance.

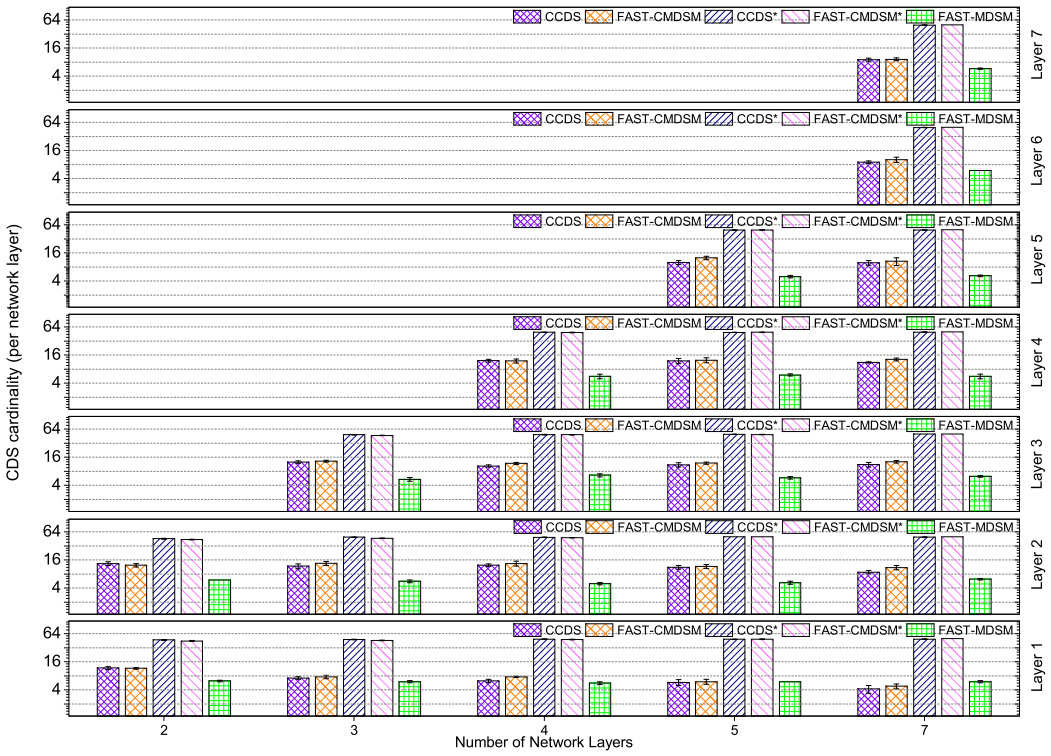



Figure 3.10: Impact of number of layers (Low skewness).

#### 3.1.7.4 Impact of Layer's Size Increase

Here we consider the impact of layer's size increase on the competitors' performance; we evaluate the *per layer* size of the $CDS$ for medium skewness, and depict the results in Figure 3.13. A first, generic observation is that the cardinality of CDS increases with increasing layer size. This is easily explained by the fact that the as the size of each layer increases, so does the need for more nodes to act as connectors, and consequently we get a larger CDS. Looking at the performance of the competitors, we note that an increase in the size of each subsequent layer by 30% or less results in having $CCDS$ to outperform $FAST\text{-}CMDSM$ with a margin from 11% up

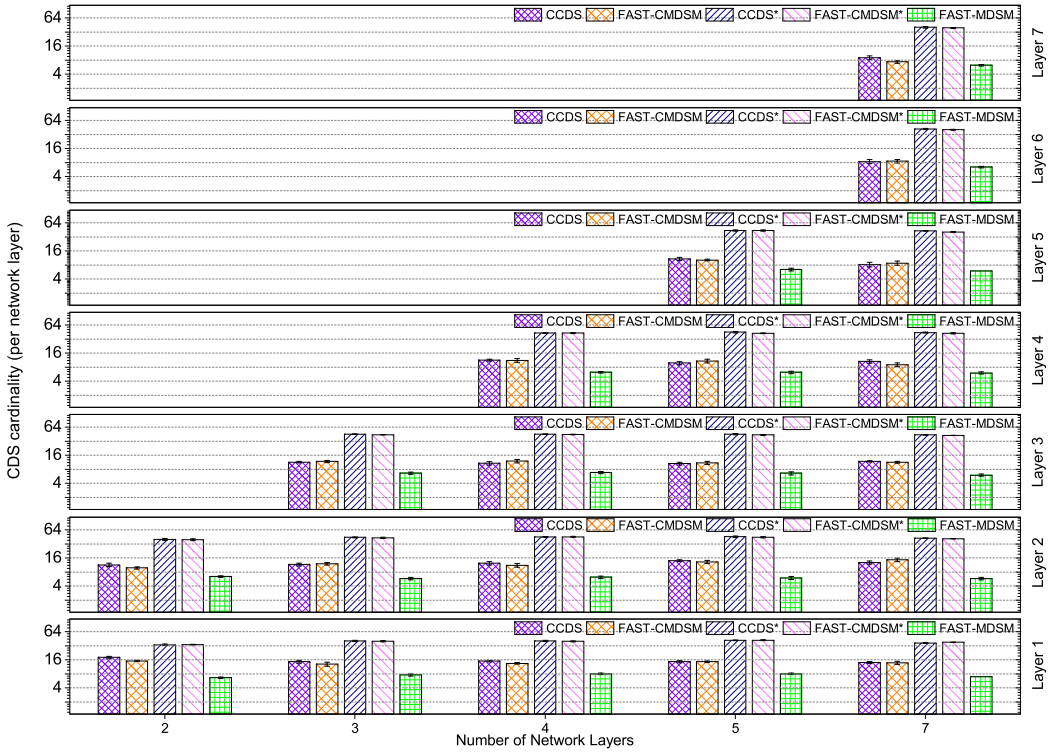



Figure 3.11: Impact of number of layers (High skewness).

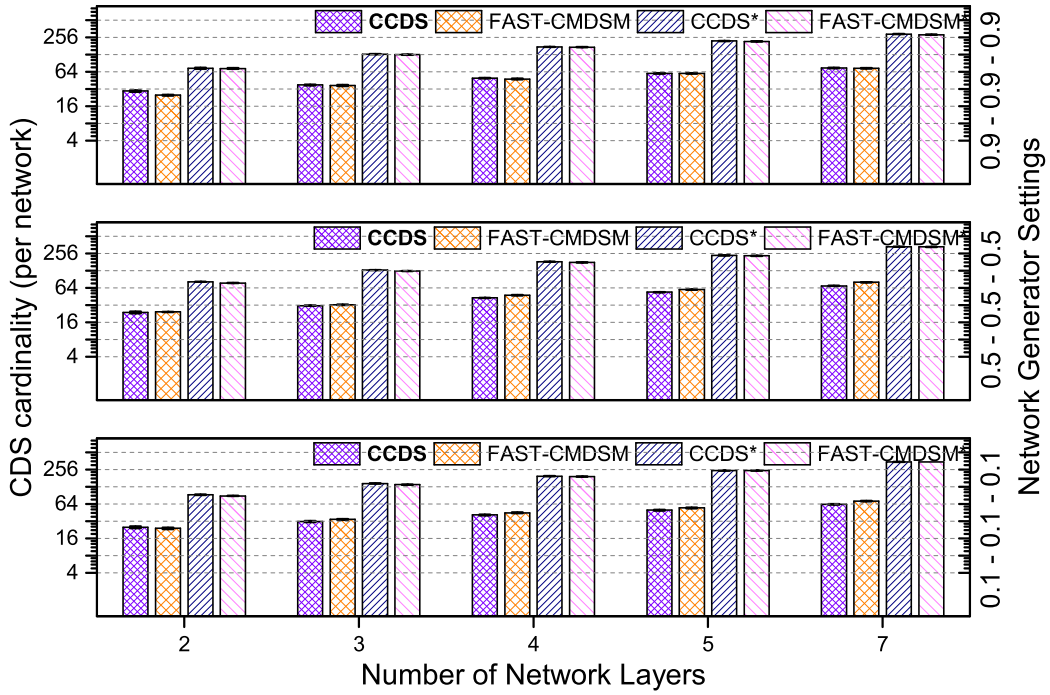



Figure 3.12: Impact of number of layers (CDS size aggregated over all layers).

to 16%. The basic reason behind that is the dense topology, with the consequence that many redundant paths remain within the vicinity of $CCDS$ (i.e., 2-hop), and therefore the pruning process eliminates many redundant dominators. On the contrary, an increase in the size of each subsequent layer by 50% or more results in having the centralized $FAST\text{-}CMDSM$ outperform $CCDS$ from 18% up to 21%. This is expected, since the large difference in the cardinality of the layers implies a large number of interlayer links, and therefore the calculation of the redundant paths by the pruning process requires a broader/global view of the topology, which is only available to the centralized *FAST-CMDSM* algorithm and not the distributed $CCDS$ algorithm. As a last comment to this experiment, we see that the versions without pruning of the algorithms select almost all nodes as dominators; in particular the performance of the pruning mechanism for $CCDS$ and $FAST\text{-}CMDSM$ regarding the $CDS$ size reduction is up to 80% and 83%, respectively.

Conducting the experiment by changing the set of variables in the interconnectivity generator to $0.1 - 0.1 - 0.1 - 0.1$ (almost uniform distribution), we observe that $CCDS$ is the champion algorithm, because of the dense topologies formed, as it is shown in Figure 3.14. In this case, $CCDS$ achieves 82% reduction in the total $CDS$ size, while $FAST\text{-}CMDSM$ achieves 80%. The last experiment in this section is conducted by setting the respective interconnectivity generator parameters to: $0.9 - 0.9 - 0.9 - 0.9$ and the results are presented in Figure 3.15. In this case, we have a barely arbitrarily-formed distribution (high skewness). We see that while increasing the number of nodes, set in every layer, the $CCDS$ algorithm remains our best option, as it is justified above. In this case, $CCDS$ achieves 76% reduction in the total $CDS$ size, while $FAST\text{-}CMDSM$ achieves 75%.

Comparing the experiments, done, by changing the values in variables of skewness, we conclude that having a more uniform distribution (Low Skewness) is the best scenario, in which, our algorithms achieves their highest efficiency. This happens because this type of distribution, normalizes well the degree of every node in every layer, which means less nodes are important for the coverage of the network and this results in smaller $CDS$ size. On the other hand, when we have high skewness, nodes are arbitrarily connected which means, more nodes in total $DS$ are needed in order to cover the whole network properly. Furthermore, in every case, $CCDS$ is the more efficient algorithm to use, bearing in mind that in every section, it provides better reduction from $FAST\text{-}CMDSM$, except for the one in which we examine low skewness and our variables are set to 0.5 in which $FAST\text{-}CMDSM$ outperforms $CCDS$ (3% variance).

#### 3.1.7.5 Energy Awareness of the Competing Algorithms

We repeated all experiments taking now into account the residual energy of each node, and we show here the obtained results which concern the aggregated over all network layers performance of the competitors. In Figure 3.16, we report the results for medium skewness and we see that $CCDS$ selects the most energy efficient $CDS$ in, almost, any case, followed by $FAST\text{-}CMDSM$.

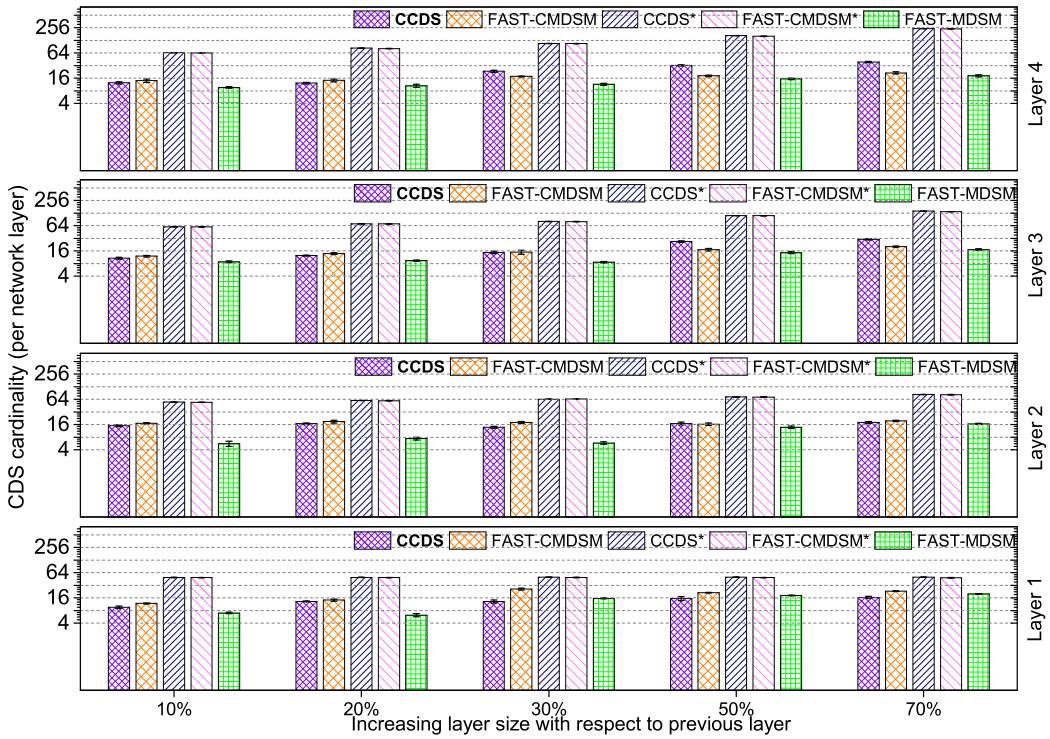



Figure 3.13: Impact of increasing layer cardinality (Medium skewness).

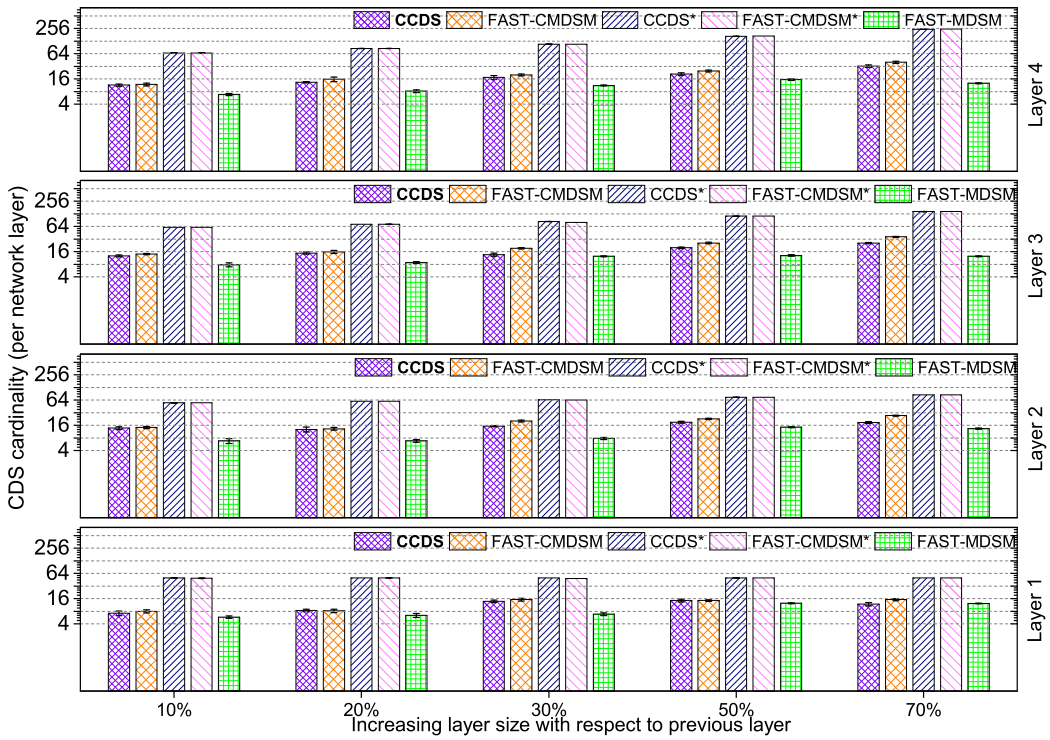



Figure 3.14: Impact of increasing layer cardinality (Low skewness).

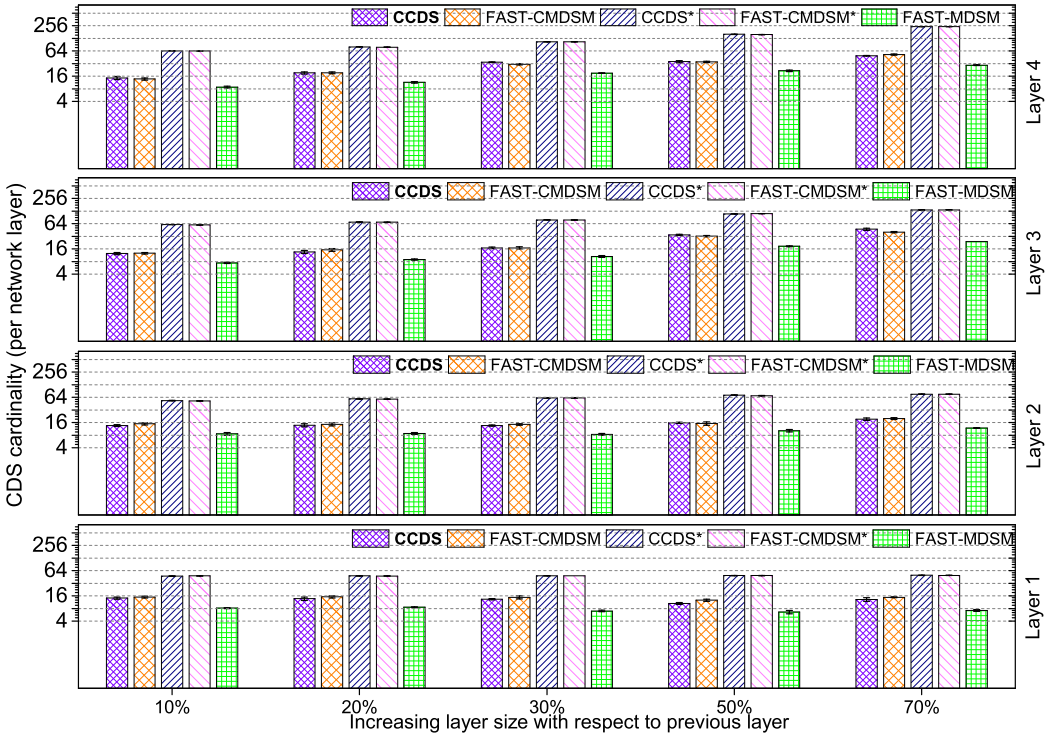



Figure 3.15: Impact of increasing layer cardinality (High skewness).

We end up in the same conclusion about *CCDS* algorithm's energy consumption regarding both the experiments of low and high skewness, as it is depicted in both Figures 3.17 and 3.18, respectively.

#### 3.1.7.6 Summary of Results

In Table 3.3 we provide a summary of *CCDS* average performance gain (percentage-wise) over the second best performing distributed algorithm across the independent parameter space.

Table 3.3: Results' summary.

| **Parameter** | **Avg Performance Gain of *CCDS*** |
|---|---|
| topology density ($\geq 6$) | $\approx$12% |
| network diameter ($\leq 5$) | $\approx$8.5% |
| number of layers | $\approx$14% |
| layers' heterogeneity ($\leq 30\%$) | $\approx$18% |

### 3.1.8 Conclusions

The Internet of Battlefield Things is a new cyberphysical system originating from the Internet of Things, but at a much larger scale, and with stringent robustness and latency requirements. Its most significant and challenging goal is to carry out commander's intent in a safe, responsive

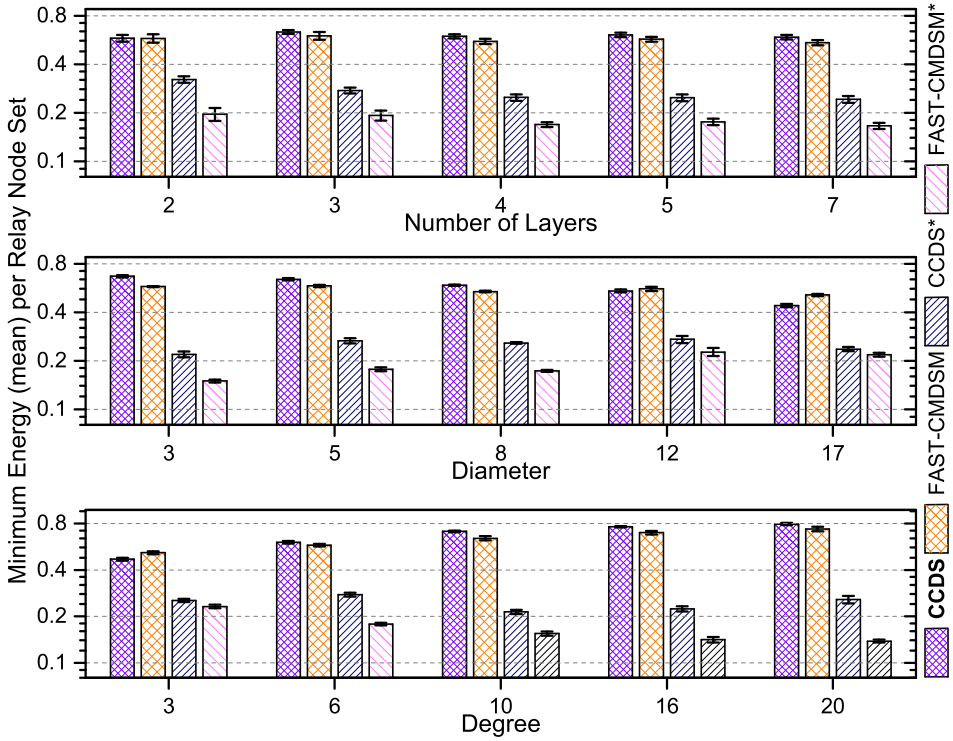



Figure 3.16: Energy awareness of the competitors (Medium skewness).

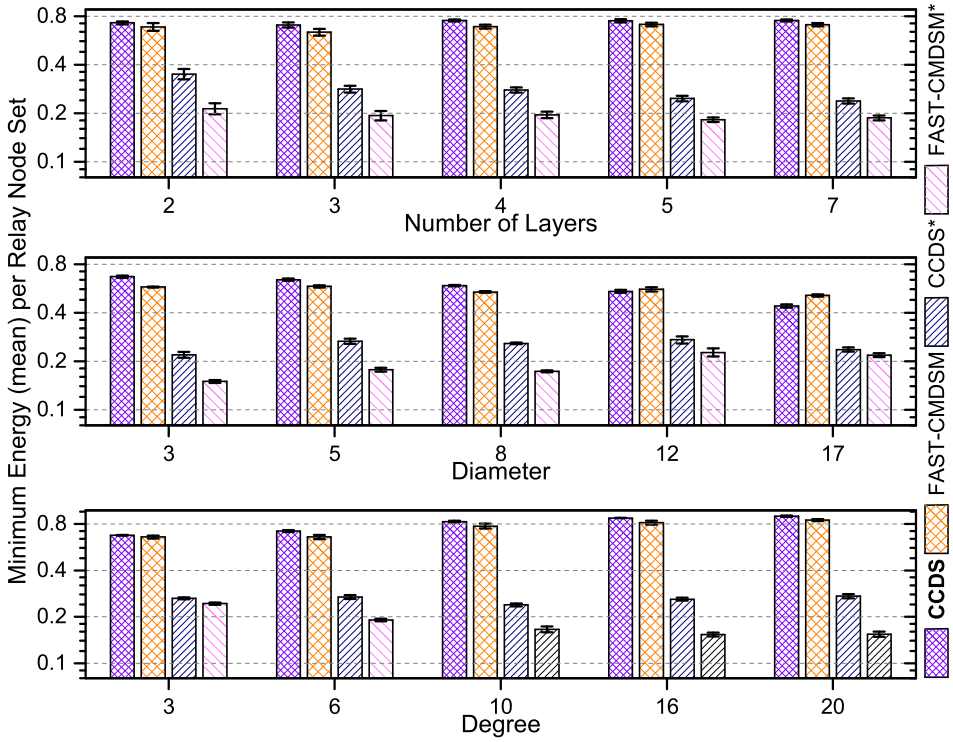



Figure 3.17: Energy awareness of the competitors (Low skewness).

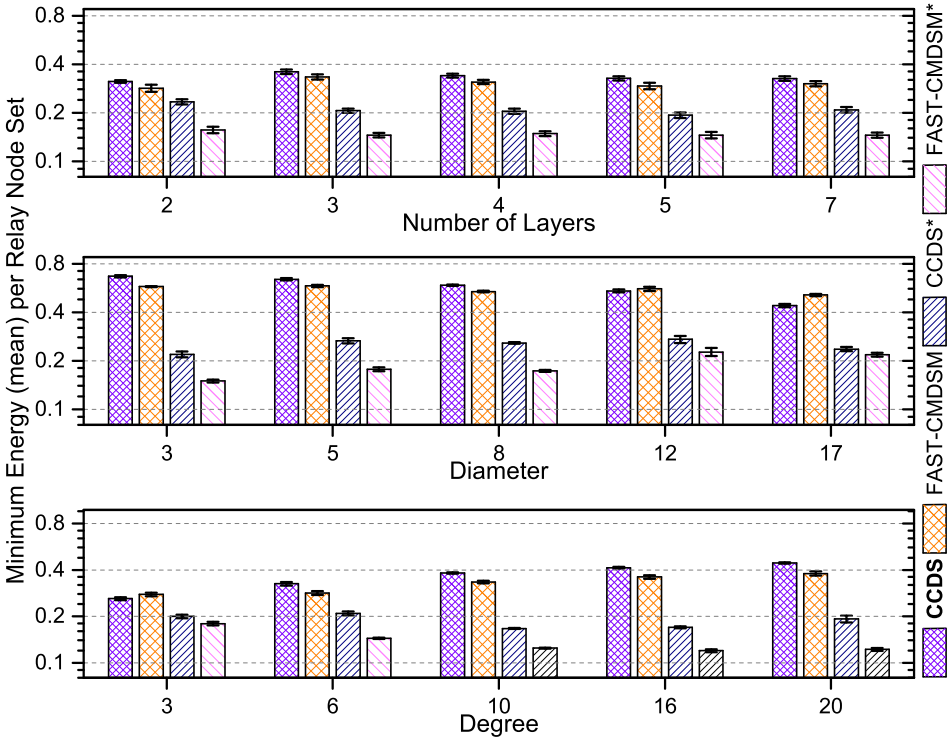



Figure 3.18: Energy awareness of the competitors (High skewness).

and resilient manner. In this work, we investigated the issue of building small and resilient backbones for IoBT networks. We abstracted the topology of an IoBT network as a multilayer graph, and we resorted to the concept of dominating sets to achieve our goal. Then, we presented a distributed algorithm for calculating connected dominating sets with small-cardinality in this multilayer network context. We implemented and contrasted our proposed algorithm to two state-of-the-art algorithms for computing connected dominating sets for multilayer networks using generated topologies. Our algorithm showed constantly better performance against these competitors across a range of topologies with various representative features (see Table 3.3).

## 3.2 Introducing the concept of Sink Group Node Betweenness Centrality

### 3.2.1 Introduction

The dramatic growth of online social networks during the past fifteen years, the evolution of Internet of Things and of the emerging Internet of Battlefield Things, the extensive study and recording of large human social networks is offering an unprecedented amount of data concerning (mainly) binary relationships among 'actors'. The analysis of such graph-based data becomes a challenge not only because of their sheer volume, but also of their complexity that presents particularities depending on the type of application that needs to mine these data and support decision making.

So, the field termed network science has spawned research in a wide range of topics, for instance: a) in network growth, developing models such as the preferential attachment [147], b) in new centrality measures for the identification of the most important actors in a social network developing measures such as the bridging betweenness centrality [86], c) in epidemics/diffusion processes [148], d) in finding community structure [149], [150], i.e., finding network compartments which comprise by sets of nodes with a high density of internal links, whereas links between compartments have comparatively lower density, e) in developing new types of networks different from static and single-layer networks, such temporal [151] and multilayer networks [152], and of course analogous concepts for these types of networks such as centralities [153], communities [154], epidemic processes [155], and so on.

Despite the rich research and the really large number of concepts developed during the past twenty years and the diverse areas where is has been applied e.g., ad hoc networking [92], the field of network is continuously flourishing; the particular needs of graph-data analytics create the need for diverse concepts. For instance, let us examine a very popular and well-understood concept in the analysis of complex networks which is the notion of centrality [18], [156], [157], being either graph-theoretic [84], or spectral [158] or control theoretic [159]. Betweenness centrality [84] in particular has been very successful and used for the design of effective attacks on network integrity, and also for discovering good "mediators" (nodes able to monitor communication among any pair of nodes in a network), but it is not effective in identifying influential spreaders; for that particular problem $K - shell$ decomposition [160] proved a much better alternative. However subsequent research [161] proved that a node's spreading capabilities in the context of rumor spreading do not depend on its $K - shell$ index, whereas other concepts such as the PCI index [162] can perform better. So, our feeling is that the field of network science, even some very traditional concepts such as betweenness centrality could give birth to very useful and practical variants of them. Let us describe a commonly addressed problem in network

[0]Related publication [C2]:E. Fragkou, D. Katsaros, Y. Manolopoulos, *Sink group betweenness centrality*, Proceedings of the International Database Engineering and Applications Symposium (IDEAS), 2021, pp. 21—26.

science concerning malware spreading minimization problem. In particular, we are given a set of subsets of computer nodes that we need to protect against a spreading malware which has already infected some nodes in the network, but we only have a limited number of vaccines (i.e., a "budget") to use. If we had to select some healthy (susceptible) nodes to vaccinate, then which would these nodes be? Our decision would of course depend on the infection spreading model, but usually routing in computer networks implements a shortest-path algorithm. So can the traditional shortest path node betweenness centrality [84] help us to identify such nodes?

Additionally, we will describe a similar problem that a modern army could possible face due to the rapid deployment of Internet of Battlefield Things [5]. The army needs to destroy by jamming the communications towards a set of subset of nodes of the enemy, but it has available only a limited number of jammer or time to deploy them. The question is now which links should the army select to attack? So can the traditional shortest-path edge betweenness centrality [12] help us to identify such links? The answer to previous questions is negative, because node/edge betweenness centrality calculates the importance of a node/edge lying on many shortest paths connecting any pair of network nodes, whereas in our problems we are interested in paths leading towards specific sets of subsets of network nodes. Starting from this observation, in this thesis we introduce the generic concept of sink group betweenness centrality which can be used as a building block for designing algorithms to address the aforementioned problems. The aim of the present work is to introduce the new centrality concept for various types of complex networks and also to present some potential uses of it for solving some network science problems. In this context, the present work makes the following contributions:

- it introduces a new centrality measure, namely the sink group node betweenness centrality;
- it extends the basic definition for networks weighted on the nodes;
- it extends the basic definition for the case of edge betweenness;
- it presents basic algorithmic ideas for calculating the aforementioned notions.

The rest of the section is structured as follows: subsection 3.2.2 defines the sink group betweenness centrality concept, and in section 3.2.3 we provide a detailed example to exemplify the strengths of the new concept. Then, in subsections 3.2.4 and 3.2.5 we present the definitions of sink group betweenness centrality for node-weighted networks and edge sink group betweenness centrality, respectively. In subsection 3.2.6 the application of the aforementioned concepts are described, and finally, subsection 3.2.7 concludes the presented work.

### 3.2.2 The sink group Betweenness Centrality

We assume a complex network $G(V, E)$, consisting of $n$ nodes, where $V = \{u_i \leq i \leq n\}$ is the set of nodes and $E = \{(u_i, u_j), i, j \in V\}$ is the set of edges. We make no particular assumptions

about the network being directed or undirected; this will be handled seamlessly by the underlying shortest-path finding algorithm. We assume that the network is unweighted, that is, neither the nodes nor the edges carry any weights; however, the former assumption will be reconsidered in section 3.2.4. Recall that the goal of sink group betweenness centrality (see Figure 3.20) is to discover which nodes lie in many paths leading towards a particular set of designated nodes. To achieve our purpose we combine the concepts of Group Betweenness (GBC) [97], [98] (although in a different fashion than in the initial definition) and the concept of Sink Betweenness (SBC) [104]. In the following we will define the measure of Sink Group Node Betweenness Centrality (SGBC), but firstly we will remind to the reader some definitions. Recall that the Shortest Path Betweenness (SPBC) Centrality is defined as follows [1]:

**Definition 3.2.1** ((Shortest Path) Betweenness Centrality [84]). *The (Shortest Path) Betweenness Centrality (SPBC) of a node u is the fraction of shortest paths between any pair of nodes that u lies on. Equation 3.1 calculates the SPBC of node u.*

$$SPBC(v) = \sum_{i=1}^{n} \sum_{j=1}^{n} \frac{\sigma_{ij}(v)}{\sigma_{ij}} \quad where \quad i \neq j \neq v \tag{3.1}$$

where $\sigma_{ij}$ is the number of shortest paths from node $i$ to $j$, $\sigma_{ij}(u)$ is the number of shortest paths from $i$ to $j$ where node $u$ lies on. This is the non-normalized version of betweenness centrality, i.e., we do not divide by the total number of node pairs.

**Definition 3.2.2** (Sink Betweenness Centrality [104]). . *The Sink Betweenness Centrality (SBC) of a node u is the fraction of shortest paths leading to a specific sink node s, that u lies on. Formally, we provide Equation 3.2 for calculating the SBC of node u.*

$$SBC(v) = \sum_{\substack{i=1 \\ S \text{ is the sink node}}}^{n} \frac{\sigma_{is}(v)}{\sigma_{is}} \quad where \quad i \neq s \neq v \tag{3.2}$$

Apparently, $SBC$ is a specialization of $SPBC$ i.e., $j$ does not iterate over all nodes of the network but it is kept fixed and coincides with the sink node $s$ ($j \equiv s$). Suppose now that there is some “abstract grouping" process that defines non-overlapping clusters of nodes over this network. In principle, we do not need to apply any grouping algorithm at all, but we can assume that some application selects the members of each cluster as part of our input. The union of these clusters may not comprise the whole complex network. We expect that

[1] When using the term “betweenness centrality" we refer to the concept of node betweenness centrality. When we wish to refer to the “edge betweenness centrallity" we will explicitly make use of this term.

Usually the size of the aggregation of all these clusters comprises a small fraction of the complex network. Let us assume that we have defined $z$ non-overlapping clusters, namely $C_1, C_2, ..., C_z$ with cardinalities $|C_1|, |C_2|, ..., |C_z|$, respectively. Then, the Sink Group Betweenness Centrality is defined as follows:

**Definition 3.2.3** (Sink Group Betweenness Centrality)**.** *. The Sink Group Betweenness Centrality (SGBC) of a node u is the fraction of shortest paths leading to any node, which is a member of any designated cluster, that u lies on. Formally, we provide Equation 3.3 for calculating the SGBC of node u.*

$$SGBC(v) = \sum_{i=1}^{n} \sum_{\substack{j \in \bigcup_{k=1}^{z} C_k \\ v \notin \bigcup_{z=1}^{k} C_k}} \frac{\sigma_{ij}(v)}{\sigma_{ij}} \tag{3.3}$$

where $n$ is the number of nodes, $C_k$ is the $k-$th cluster, $\sigma_{ij}$ is the number of shortest paths from node $i$ to $j$ and $\sigma_{ij}(u)$ is the number of shortest paths from $i$ to $j$ where node $u$ lies on. Intuitively, a node has high Sink Group Betweenness Centrality if it sits in many shortest paths leading towards nodes belonging to any group. Definition 3.2.3 requires that the node whose $SGBC$ we calculate is not part of any existing cluster. This is not mandatory in general, but since we are looking for nodes which can act as mediators in the communication with the clusters' nodes, it makes sense to exclude the clusters' nodes from being considered as potential mediators. By removing this constraint, we get the concept of Generalized $SGBC$, but in this thesis we consider it to be equivalent to the plain $SGBC$.

We have considered unweighted and undirected complex networks. The extension of the definition of Sink Group Betweenness Centrality for directed networks is straightforward, because it is handled by the path finding algorithm. On the other hand, the extension to node or edge weighted networks is less straightforward and we will provide some ideas in a later section.

#### 3.2.2.1 $SGBC$ versus its closest relatives.

$SGBC$ has as its special cases the concepts of betweenness centrality and of sink betweenness centrality. In particular, $SGBC$ is related to its closest relatives as follows:

- Clearly, $SGBC$ is related to the $SPBC$ in the following way: when the union of all clusters comprises the whole network, then Generalized $SGBC$ and $SPBC$ coincide.
- Moreover, $SGBC$ is a generalization of $SBC$ in the following way: when we have only one cluster which contains a single node, then $SGBC$ and $SBC$ coincide.
- At this point we need to make clear the difference between Group Betweenness Centrality [97] and $SGBC$; the former seeks for the centrality of a group of nodes with respect to

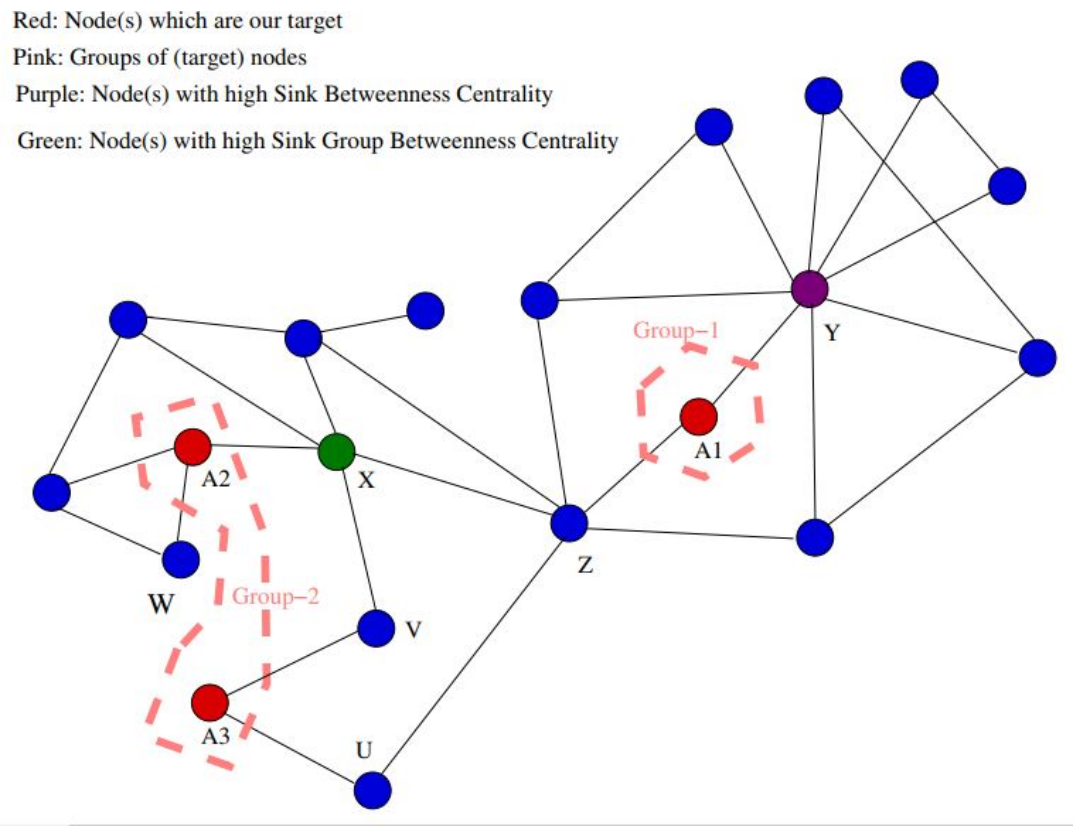



Figure 3.19: Illustration of Sink Group Betweenness Centrality.

the rest of the nodes of the network, whereas the latter seeks for the centrality of a single node with respect – not the nodes of the whole network but – to the nodes belonging to some groups (clusters). Apparently, we can generalize Definition 3.2.3 and Equation 3.3 to follow the ideas of [97].

### 3.2.3 Exemplifying the merits of *SGBC*

Let us now look at the small complex network of Figure 3.19. This network represents a collection of communicating nodes administered by an overseeing authority. We have no weights on links, but we have denoted the nodes that are mostly important for the authority – and thus must be protected better – with red color. We have not designated any attacker or attacked node in the figure, because in many situations the attack might be known where it will initiate.

Suppose that this authority is interested in investing a limited amount of money to buy hardware/software to equip some nodes with hoping that these will protect the significant ones, e.g., by stopping any cascade of infections. Let us call such nodes safeguarding nodes. Moreover, it is obvious that the authority needs to identify a limited number of safeguarding nodes, due to the limited budget. So, how we identify safeguarding nodes by exploiting the topological characteristics of the complex network?

The obvious solution is to look at the one-hop neighbors of the safeguarding nodes. Then, we can make use of the SBC [104] and say that the purple node ($Y$) is a safeguarding node. The purple node sits on many shortest paths towards the red node $A1$ (Group-1). In this way, we can select the set of nodes $\{V, W, Y\}$as safeguarding nodes. When the constraint of reducing the cardinality of this set comes into play, then we must somehow group safeguarding nodes, and seek for safeguarding nodes which are close to each group or to many groups. (This tradeoff with be analysed in the sequel.) Concerning the structure of groups, we have the following characteristics:

- Clusters might contain only one node.
- Nodes comprising a cluster need not be one-hop neighbors

$SBC$ is of no use anymore, but we can use the concept of $SGBC$ defined earlier. Using this concept, the green node $X$ has high Sink Group Betweenness Centrality because it sits on many shortest paths towards the two red nodes $A2$ and $A3$ comprising Group-2. Notice, here that the nodes with high $SGBC$ are not correlated to nodes with high $SPBC$; for instance, the node with the highest $SPBC$ in the network is node $Z$ (it is an articulation point or bridge node). Now let us look at the impact of safeguarding nodes' grouping on the existence (and/or $SGBC$ value) of safeguarding nodes. As said earlier, the grouping creates the following tradeoff: the larger the groups we define, the less the nodes (if any) with high Sink Group Betweenness Centrality we can find.

If we unite Group-1 and Group-2 into a single group and ask the question 'which node(s) sit in many shortest paths towards ALL members of this new group', then we can safely respond that only node $Z$ is such a node, because of the particular structure of the network of Figure 3.19. Recall that node $Z$ is a bridge node, thus all shortest paths from the right of $Z$ to nodes $A2$ and $A3$ will pass via $Z$ and all shortest paths from the left of $Z$ towards node $A1$ will pass via $Z$. If we now examine Figure 3.20, and consider initially a grouping comprised by two groups, i.e., Group-1 and Group-2, then we can clearly identify the two green nodes as those with the highest $SGBC$. If we create a single large group comprised by all red nodes, then we can not find any node with high enough $SGBC$ to act as safeguarding nodes. For this grouping, the blue nodes have also $SGBC$ comparable to that of green nodes. Thus, with this grouping the identification of safeguarding nodes becomes problematic.

#### 3.2.3.1 Calculation of *SGBC*

The calculation of $SGBC$ can be carried out using as basis the Dijkstra's algorithm. Algorithm 3 is a baseline one:

**Algorithm 3** Calculation of SGBC

1: **for** each node $s \in \bigcup_{k=1}^{z} C_k$ to $j \in \{V - \bigcup_{k=1}^{z} C_k\}$ **do**
2: $SP_s \leftarrow$ Dijkstra to find shortest-path from $s \rightarrow j$
3: **end for**
4: $SP \leftarrow \bigcup_{\forall s} SP_s$
5: **for** each path $p \in SP$ **do**
6: Use a hash table to group paths based on start-end
7: **end for**
8: **for** each hash table bucket **do**
9: Use a hash table to accumulate node appearance in paths
10: **end for**

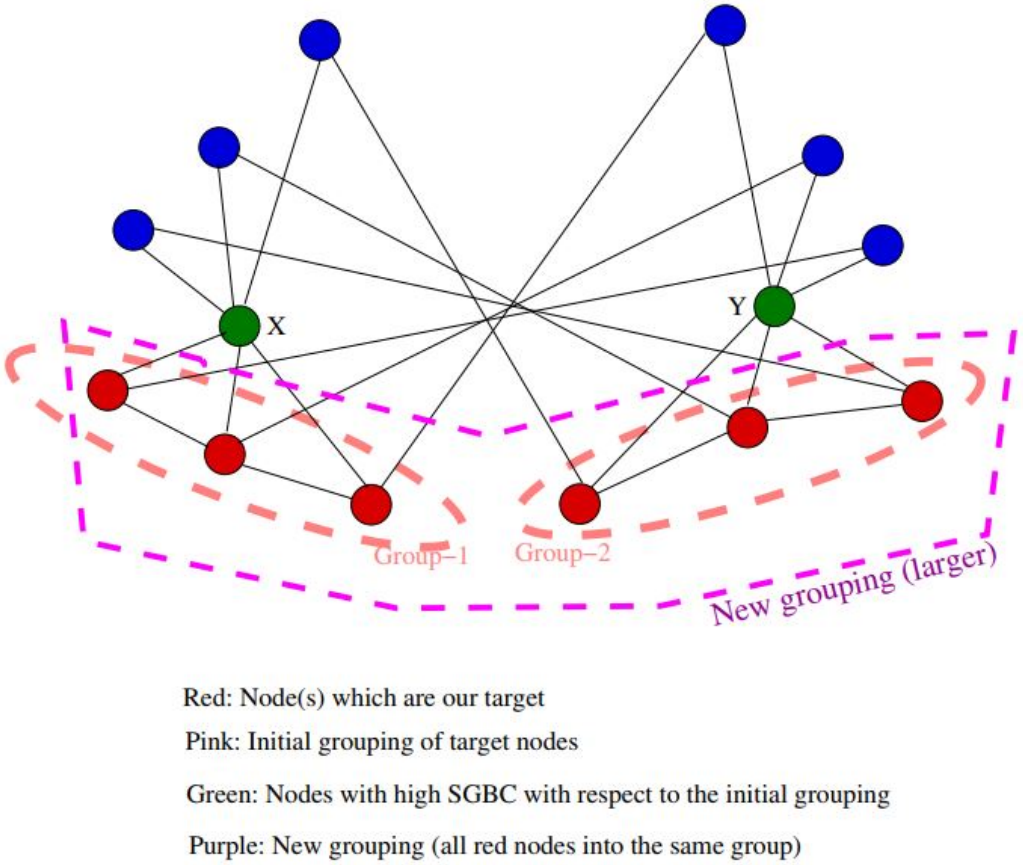



Figure 3.20: Impact of grouping on Sink Group Betweenness Centrality.

**Proposition 3.2.1.** *The worst-case computational complexity of Algorithm 3 is*

$$O\left(\left|\bigcup_{k=1}^{z} C_k\right| \times n^2\right) \tag{3.4}$$

*, where $n$ is the number of network nodes.*

PROOF. Assuming a network with $n$ nodes and $m$ edges, and an implementation [2] of Dijkstra's algorithm that costs

$$O(n^2 + m), \text{ i.e., } O(n^2)$$

then Lines 1–2 cost

$$O\left(\left|\bigcup_{k=1}^{z} C_k\right| \times n^2\right);$$

lines 3–5 cost $O(n^2)$ since there are at most $n^2$ paths, and lines 6–7 cost $O(n^2)$.

For unweighted and undirected networks, we can design a faster algorithm along the ideas of breadth-first traversal and the algorithm by Brandes [163], but this is beyond the scope of the present work.

### 3.2.4 *SGBC* with node weights

Now suppose that each node has a weight associated with it (e.g., depicting its trustworthiness, its balance in a transaction network, etc), then we need to define the SGBC in such a way that it will take into account these weights. Note that this is different from having weights in edges (i.e., a weighted network) because that weights are handled by the shortest-path finding algorithm. We can have the following options:

- We can apply the straightforward idea of multiplying by a node's weight as in [164], i.e.,

$$SGBC^{nw}(v) = node\_weight(v) \times \sum_{i=1}^{n} \sum_{\substack{j \in \bigcup_{k=1}^{z} C_k \\ v \notin \bigcup_{k=1}^{z} C_k}} \frac{\sigma_{ij}(v)}{\sigma_{ij}} \tag{3.5}$$

  with appropriate normalization in the interval [0...1] for both node's weight and $SGBC$.

- Apply standard procedures for turning the problem of finding shortest paths in networks with weights on edges and/or nodes into a shortest path finding problems with weights only on edges. The methods are the following ($n$ - number of nodes, $m$ - number of edges):

  – We can split each node apart into two nodes as follows: for any node $u$, make two new nodes, $u_1$ and $u_2$. All edges that previously entered node $u$ now enter node $u_1$,

[2] There are various implementations with even smaller cost utilizing sophisticated data structures or taking advantage of network sparsity

and all edges that leave node $u$ now leave $u_2$. Then, put an edge between $u_1$ and $u_2$ whose cost is the cost node $u_1$. In this new graph, the cost of a path from one node to another corresponds to the cost of the original path in the original graph, since all edge weights are still paid and all node weights are now paid using the newly-inserted edges. Constructing this graph can be in time $O(m+n)$, since we need to change each edge exactly once and each node exactly once. From there, we can just use a normal Dijkstra's algorithm to solve the problem in time $O(m+nlogn)$, giving an overall complexity of $O(m+nlogn)$. If negative weights exist, then we can use the Bellman-Ford algorithm instead, giving a total complexity of $O(mn)$.

– Alternatively, we can think as follows: since we have both edges and nodes weighted, when we move from $i$ to $j$, we know that total weight to move from $i$ to $j$ is weight of edge$(i \rightarrow j)$ plus weight of $j$ itself, so lets make $i \rightarrow j$ edge weight sum of these two weights and the weight of $j$ zero. Then we can find shortest path from any node to any other node to in $O(mlogn)$.

- We can multiply each fraction (of shortest paths) in the summation formula with the minimum weight (positive or negative) found along the path.

### 3.2.5 Sink group edge Betweenness Centrality

The plain edge-betweenness centrality measure is used to identify the edges which lie in many shortest-paths among pair of network nodes. It has been widely used in network science and not only, e.g., for discovering communities in networks [165], for designing topology control algorithms for ad hoc networks [12], etc. In our context, we ask the question whether we can identify the edges which lie in many paths towards a set of subsets of network nodes. The extension of SGBC to the case of edges is easy. Thus, the Sink Group Edge Betweenness Centrality is defined as follows:

**Definition 3.2.4** (Sink Group Edge Betweenness Centrality.)**.** *The Sink Group Edge Betweenness Centrality (SGEBC) of an edge e is the fraction of shortest paths leading to any node, which is a member of any designated cluster, that e lies on. Formally, we provide Equation 3.6 for calculating the SGEBC of edge e.*

$$SGEBC(e) = \sum_{i=1}^{n} \sum_{\substack{j \in \bigcup_{k=1}^{z} C_k \\ v \notin \bigcup_{k=1}^{z} C_k}} \frac{\sigma_{ij}(e)}{\sigma_{ij}} \tag{3.6}$$

*where $n$ is the number of nodes, $C_k$ is the $k-th$ cluster, $\sigma_{ij}$ is the number of shortest paths from node $i$ to $j$ and $\sigma_{ij}(e)$ is the number of shortest paths from $i$ to $j$, where edge $e$ lies on.*

### 3.2.6 Applications of *SGBC*

#### 3.2.6.1 *SGBC* and influence minimization

We have already explained in the introduction how sink group betweenness centrality can be used to limit the spreading in the context of influence or infection minimization problems under budget constraints. Especially relevant becomes for those problems that are online [166], i.e., require a continuous combat against the spreading while the infection evolves in various parts of the network.

#### 3.2.6.2 *SGBC* and virtual currencies networks

Let us now look again at the network of Figure 3.19, but this time assume that the graph represents a network of transactions being made using a virtual currency, e.g., a community currency [167], which – differently from BitCoin – is administered by an overseeing authority. We have denoted the nodes in deficit (i.e., with lack of virtual money) with red color. Suppose that this authority is interested in injecting a limited amount of money into some nodes with the hope that these nodes will buy something from the red nodes and therefore with reduce their deficit. Let us call such nodes deficit balancers. Suppose that the authority needs to identify a limited number of deficit balancers, otherwise will have to divide this amount of money into too many deficit balancers, and eventually an even smaller amount of money will end up to the nodes with deficit. So, how we identify deficit balancers? Exactly as before, the obvious solution is to look at the one-hop neighbors of the deficited nodes. Then, we can make use of the SBC and say that the purple node ($Y$) is a deficit balancer. The purple node sits on many shortest paths towards the red node $A1$ (Group-1). In this way, we can select the set of nodes $\{V, W, Y\}$ as deficit balancers. If the authority is constrained to reduce the cardinality of this set, then we must seek for deficit balancers which are close to each group or to many groups. From this point, we can continue along the same reasoning as we did in section 3.2.3.

#### 3.2.6.3 *SGBC* and community finding

The concept of *SGEBC* can be used as a component of an algorithm for discovering the community where a particular set of nodes belongs. The idea is that if this collection of nodes is relative close to each other, then repeated deletion of high *SGEBC* edges will gradually isolate this community of nodes from the rest of the network nodes.

### 3.2.7 Conclusions

Network science continues to be a very fertile research and development area. The more our world becomes connected via 5G and Internet of Things (or Everything), the more network science evolves into a precious tool for graph-data analysis. One of the central concepts in this field, namely centralities despite counting half a century of life, is still hot giving new

definitions, and new insights into the networks' organization. in this thesis, we introduced a new member into the family of shortest path betweenness centralities, namely sink group betweenness centrality. The purpose of this measure is to identify nodes which are in positions to monitor/control/mediate the communication towards subsets of networks nodes. We provided a simple algorithm to calculate this measure, and also extended it for node-weighted networks, and also for the case of edge betweenness. This effort is simply out first step in a long journey to develop efficient algorithms for the calculation of these measures, to analyze its distribution in real networks, to extend them to consider sink group betweenness centrality computed for a collections of network nodes, and develop techniques for approximating them.

Chapter 4

# Model Reduction

## 4.1 Neural Topology Sparsification Prospects, derived from Network Science disciplines

### 4.1.1 Introduction

The tremendous success of deep learning in various domains such as image understanding, speech recognition, and data analysis, in general, has made neural networks among the most successful machine learning methods. Feed-forward networks comprised of fully connected layers of neurons form the basis for many of these popular neural architectures [168]. The connections among neurons are weighted, and the task of any neural training process is to set the value of these weights appropriately to minimize the value of a selected performance (error) function. For fully connected neurons, the number of weighted connections is quadratic concerning the number of neurons. Thus, the computational complexity of a training algorithm such as the popular back-propagation for a deep architecture will be a significant challenge for adopting a particular neural architecture. Recent trends such as tiny deep learning [169] and the deployment of deep

[0]Related publication **[J3]**:E. Fragkou, M. Koultouki, D. Katsaros, *Model reduction of feed forward neural networks for resource-constrained devices*, Applied Intelligence (Springer) Vol.53 (11), 2023, pp. 14102—14127 & **[C3]**:A. Chouliaras, E. Fragkou, D. Katsaros, *Feed forward neural network sparsification with dynamic pruning*, Proceedings of the 25th Pan-Hellenic Conference on Informatics (PCI21), Volos, Greece, 2021, pp. 12—17.
Related Book Chapter **[BC1]**: E. Fragkou, D. Katsaros, *Non-static tinyml for ad hoc networked devices*, in: B. S. Chaudhari, S. N. Ghorpade, M. Z. R. Paškauskas (Eds.), TinyML for Edge Intelligence in IoT and LPWAN Networks, Elsevier, 2024, pp. 229–251.
Related Poster presentation [P1]:Evangelia Fragkou, Dimitrios Katsaros, Memory Reduction and Training Acceleration of *Neural Networks for Tiny Machine Learning*, Poster, 6th Summit on Gender Equality in Computing (GEC22), Thessaloniki, Greece, June 16–17, 2022 & Evangelia Fragkou, Dimitrios Katsaros, *Transfer Deep Learning for TinyML*, Poster, 5th Summit on Gender Equality in Computing (GEC23), Athens, Greece, June 26–27, 2023.

learning in modern IoT environments such as the modern battlefield [5] make the development of fast neural network training methods as significant as their inference accuracy and even more critical.

There are various directions in speeding up the training process of a multilayer neural network, and they will be presented in more detail in section 2.1. In summary, past efforts focused: a) on developing faster approaches than the traditional steepest descent optimization algorithm, b) on developing variations of back-propagation, e.g., with adaptive/different learning rates, c) on the technique of dropout, d) on neural network topology pruning/sparsification, and e) on hardware-based solutions.

In this work, we develop methods for defining how neurons from adjacent layers are connected to each other, or to tell it in different words, we define methods for altering the topological structure of the neural network before, during, and after its training. Here, starting from the observation that biological neural networks are not fully connected but sparse, and they are scale-free or small-world [16], we investigate the usefulness of network science concepts, and in particular, the use- fulness of the theories concerning complex network growth models in speeding up a neural network training. Network science is the discipline that studies complex networks and their properties, and it has already proven its value in online social networks [66], wireless networking, and other fields, in being - apart from a "descriptive" discipline - an arsenal for developing solutions for many problems. In particular, we examine the impact of scale-free and small-world topologies (instead of the traditional bipartite fully-connected ones) on the training speed and the accuracy in training data of a neural network.

The rest of the section is organized as follows: in subsection 4.1.2 we elaborately describe the proposed techniques and in subsection 4.1.3 we make the experimental evaluation of our methods. In subsections 4.1.4,4.1.5 we extend our experimental cases and finally, in subsection 4.1.6, we summarize the presented work.

### 4.1.2 The topology-based proposed techniques

This section describes five algorithms for creating topologies inspired by network science. They all share the common feature, namely they start from a structured topology and following and algorithmic procedure, they produce another structured topology. Before presenting the details and the pseudocode for each of the methods developed here, we provide a short description of the randomized technique for rewiring the links among nodes which constitutes the basis of our algorithms.

#### 4.1.2.1 Scale-Free to Scale-Free with random rewiring (SF2SFrand).

In our first method, we start from a scale-free network and we end up by creating a new topology which is again scale-free based on the following principle:

- We create a sparse table, representing the connections between the nodes in each layer.

- Then, we remove the weights close to zero (sparsity) and then add as many links as we removed to the most powerful node. If a link that has to be reconnected, already exists, then no change takes place.

The main difference between SET and SF2SFrand lies in the construction of the initial network topology. SET creates the initial network connecting arbitrarily pairs of nodes. On the contrary, SF2SFrand initializes the network immediately as a scale-free topology; i.e., SF2SFrand connects each node of every layer to the next layer's most "powerful" node. Thus, SF2SFrand starts from a power-law (scale-free) topology. Even though this seems as a slight deviation, in practise it makes a great difference in performance, and it also needs intelligent, algorithmic solution to the definition and selection of a "powerful" node (described in the next paragraph). SF2SFrand's different initial topology reinforces weights with a strong value and in this case important information is kept and passed through layers in the training phase. Consequently, SF2SFrand achieves faster convergence due to well-organized and well-distributed information in the early neural network. Moreover, its accuracy increases abruptly in the early training iterations. Thus, the always-scale-free topology of SF2SFrand results in significantly less time to converge. (cf. section 4.1.3.2). Finally, the purely random rewiring choices of SET implies that all attempted reconnections succeed, whereas the principled rewiring choices of SF2SFrand might result in that some attempted reconnections already exist because too many nodes are already connected to the powerful/hub node; thus we earn a further reduction in the number of weights.

In order to find the most powerful node, we calculate the link-attraction probability of each node, which is defined as the ratio of the incoming connections of this node to the total number of connections present in the graph. Every link is reconnected to the node with the largest such probability. We end up with a graph similar to scale-free, with the property that during the training it removes the links with weight close to zero and adds as many links as removed randomly after each epoch. The algorithm is illustrated in Algorithm 4.

#### 4.1.2.2 Scale-Free to Scale-Free using preferential attachment (SF2SFba).

In this algorithm, a scale-free topology is constructed, which means that links are connected to the nodes such that the degree distribution follows a power-law. In other words, links are added to the node according to their current degree [67] implementing a Barabasi-Albert attachment policy.

We start by creating a scale-free network, using the algorithm described in section 4.1.2.1. Then, in the part of the code, where network is sparsed, we remove the weights close to zero

**Algorithm 4** Scale-Free to Scale-Free algorithm with random rewiring (SF2SFrand)

Initialize a sparse table randomly;
Remove links, whose weights are close to zero;
**while** each node $i$ of every layer **do**
    calculate the maximum degree probability as follows:
    $p_i = \frac{P(\text{incoming links}_i)}{P(\text{links in network})}$;
**end while**
Reconnect the nodes, whose link was removed, with the node that has the maximum $p$;
The weight of the new connection is assigned randomly;
**if** this link exists **then**
    nothing is done;
**end if**
Weights Evolution via BackPropagation();
**while** each epoch and during BackPropagation() **do**
    remove links, whose weights are close to zero;
    reconnect the nodes, whose link was removed, randomly;
    the weight of the new connection is assigned randomly;
**end while**

and add as many links as we removed to the most powerful node. The algorithm is depicted in Algorithm 5.

**Algorithm 5** Scale-Free to Scale-Free algorithm using preferential attachment (SF2SFba)

Initialize a sparse table randomly
Remove links whose weights are close to zero
**while** each node $i$ of every layer **do**
    Calculate the maximum degree probability:
    $p_i = \frac{\sum(\text{incoming\_links}_i)}{\sum(\text{links\_in\_network})}$
**end while**
Reconnect the nodes whose link was removed with the node that has the maximum $p$
Assign a random weight to the new connection
**if** this link already exists **then**
    Do nothing
**end if**
Weights Evolution via BackPropagation()
**while** each epoch and during BackPropagation **do**
    **for** each layer **do**
        Perform steps 2 to 12
    **end for**
**end while**

#### 4.1.2.3 Scale-Free to Scale-Free (5 strongest nodes) (SF2SF5).

In this particular algorithm, we start with a scale-free implementation, as described in section 4.1 and end up in a similar type of network, using an alternative version of scale-free technique. In

every epoch, except last one, we remove the links with weight close to zero and add as many links as we removed (sparsity) to the most powerful node (node with the greatest probability).

In the original method, if a link, from one node, in a layer, to another, which has to be reconnected, already exists, then changes do not occur. So in this version, if the link we want to add to the most powerful node already exists, then we try to add a connection to the second most potent node and so on, untill the fifth strongest node. The weights are randomly given in the original version and in our alternative one.

We developed this variant of scale-free to scale-free algorithm in order to add as many links as we can. If a link we try to reconnect already exists, we try to reconnect it to the successive (regarding the degree probability) node and so on, trying to maintain the balance between the removed links and those that have to be reconnected, in the network. The algorithm is illustrated in Algorithm 6.

**Algorithm 6** Pseudocode of Scale-free to Scale-free (5 strongest nodes) (SF2SF5) algorithm

Initialize a sparse table randomly
Remove links whose weights are close to zero
**for** each node $i$ of every layer **do**
    Calculate the maximum degree probability:
    $p_i = \frac{\sum(\text{incoming\_links}_i)}{\sum(\text{links\_in\_network})}$
**end for**
Reconnect the nodes whose link was removed with the node that has the maximum $p$
Assign a random weight to the new connection
**if** this link exists **then**
    Do nothing
**end if**
Weights Evolution via BackPropagation()
**while** each epoch and during BackPropagation **do**
    **for** each layer **do**
        **for** each node **do**
            Repeat steps 4 to 7
            Reconnect all removed links as follows:
            **for** each $j$ in the 5 strongest nodes of the next layer **do**
                **if** the link in $j$ node does not exist **then**
                    Connect node to $j$
                    Break
                **end if**
            **end for**
        **end for**
    **end for**
**end while**

#### 4.1.2.4 Scale-Free to Small-World ( SF2SW)

.

Here, we start again from a scale-free network, and after the procedure of training, we construct a small-world type of network.

In the first place, after removing connections with no critical impact, we calculate the degree probability of every node and then reconnect these nodes (in every layer), whose link was deleted, to the most potent node of the next layer. After the training part of the algorithm we proposed, a small-world network is created.

Specifically, a rewiring probability is defined. In order to construct the network, we chose small values of this probability (p=0.02 and p=0.075) because the smaller the probability is, the less density the network has (sparsity). Then, for each node in every layer, we find its links, whose weight is non-zero, and give them a random probability. After that, links whose probability is smaller than the rewiring probability are disconnected, and then we try to rewire them in a random node, giving them a random weight value (only if the connection to this randomly chosen node does not exist, else, we try to find another random node to connect to). In this case, we want to see how much the performance (not only the accuracy but also the training time) of the network is affected when we start from a strictly structured network and through the training process; we create a network with a more arbitrary structure. The algorithm is illustrated in Algorithm 7.

Table 4.1: Characteristics, regarding some of the larger datasets, availed of.

| ***Name of dataset*** | ***Instances*** | | ***Features*** | ***Classes*** |
|---|---|---|---|---|
| Fashion-Mnist [170] | train samples 60,000 | test samples 10,000 | 28x28 | 10 |
| CIFAR-10 [171] | train samples 50,000 | test samples 10,000 | 32x32 | 10 |
| CIFAR-100 [171] | train samples 50,000 | test samples 10,000 | 32x32 | 100 |
| ImageNet [172] | train samples 1,281,167 | test samples 100,000 | 224x224 | 1,000 |
| Intel Classification Dataset [173] | train samples 14,000 | test samples 3,000 | 150x150 | 6 |
| Kaggle [174] - temperature of five Chinese cities for a time period of five years (2010-2015). | 52,585 measurements of real temperatures in Celsius degrees. | | | |

#### 4.1.2.5 Small-World to Small-World (SW2SW).

In the last algorithm, we implement a network based on small-world technique, and the same type of network is constructed through the training process. It is interesting to see how the transition from a less randomly constructed network (small-world) to another affects the network

**Algorithm 7** Scale-Free to Small-World (SF2SW) pseudocode

Initialize a sparse table randomly
Remove links whose weights are close to zero
**while** each node $i$ of every layer **do**
    Calculate the maximum degree probability:
    $p_i = \frac{\sum(\text{incoming\_links}_i)}{\sum(\text{links\_in\_network})}$
**end while**
Reconnect the nodes whose link was removed with the node that has the maximum $p$
Assign a random weight to the new connection
**if** this link already exists **then**
    Do nothing
**end if**
Weights Evolution via BackPropagation()
**while** each epoch and during BackPropagation() **do**
    Define a threshold probability $P$
    **for** each layer **do**
        **for** each node $n$ in this layer **do**
            Find links whose weight is non-zero
            Assign those links a random probability $P_{\text{link}}$
            Find $N$ of those links such that $P_{\text{link}} < P$
            **for** each $N$ **do**
                Select a node $m$ in the next layer randomly, ensuring no connection exists between $m$ and $n$
                Rewire the current node $n$ to node $m$
            **end for**
        **end for**
    **end for**
**end while**

performance. These small-world networks are created in the same way described in Algorithm 7. We show the pseudocode of this new method in Algorithm 8.

**Algorithm 8** Small-World to Small-World (SW2SW) pseudocode

```
Initialize a sparse table randomly
Define a threshold probability P
for each layer do
    for each node in this layer do
        Find links whose weight is non-zero
        Assign those links a random probability P_node
        Find numbers N of nodes such that P_node < P
        for each N do
            Select a node m in the next layer randomly, ensuring the link weight is zero
            Rewire the current node to node m
        end for
    end for
end for
Weights Evolution via BackPropagation()
while each epoch and during BackPropagation() do
    Repeat steps 2 to 10
end while
```

### 4.1.3 Experimental evaluation

This section presents details about the competitors, the datasets used, and about the size of the neural network we experiment with; then, it presents the results of the experimental evaluation of the developed algorithms.

#### 4.1.3.1 Evaluation settings

In this work, we deal with a classification task using data regarding the disease of cancer and fashion.

**Competitors**. In our experiments, we used as competitors the algorithms proposed in the current work and presented in sections 4.1.2.1–4.1.2.5. We use the following keys for the proposed algorithms: *SF2SFrand* for Algorithm 4, *SF2SFba* for Algorithm 5, *SF2SF5* for Algorithm 6, *SF2SW* for Algorithm 7 with two versions based on different probabilities, i.e., *SF2SW(p*=0.02*)* and *SF2SW(p*=0.075*)*, *SW2SW* for Algorithm 8 with two versions based on different probabilities, namely *SW2SW(p*=0.02*)* and *SW2SW(p*=0.075*)*. The competitors include the SET algorithm [17], and also the fully-connected (FC) neural network without any pruning. Moreover, we implemented a recent algorithm inspired by the concept, proposed in [63] and a set of variations implementing the logic of the algorithm reported in [175]. We call the former algorithm as the $\zeta$ - *parameterized - (SF2SFrand)*, which is a variant of SF2SFrand and it differs in the values of $\zeta$ parameter, which is responsible for the number of the close-to-zero weight links, being erased in every epoch. In simple SF2SFrand, the value of $\zeta$ is stable during the training procedure while in the last implementation this value is tuned and specifically is reduced

in every epoch, exponentially,so as to observe if the performance of the network is better, if different and gradually,smaller values of $\zeta$ are used. The set of the latter algorithms, called the *AVG-weighted algorithms* produce a new variant of each one of our algorithms. Specifically, they are modified only in the part which is responsible for assigning weight values to the new connections; i.e., instead of placing a random value as a connection weight, we aggregate the values of all the links, being pruned, in this layer, for every epoch and then we give the average value of this sum, as the weight of a new connection. For every new incoming link, we assign this value, being modified about 2%, each time.

**Datasets**. In order to evaluate our algorithms, we use the classification accuracy as a measure, which is the percentage evaluation between the predicted and target value. Specifically, we present the evolution of accuracy, in every epoch, during the whole training process, until the model converges. Furthermore, we show the time, needed for a model to generalize the given data.

In order to test the algorithms, we used datasets that have a hundred instances and a few thousand features and they are encoded with one-hot encoding, as described in Tables 4.1 and 4.2.

By the word instances, we define the number of input vectors, which are composed of as many components as features, given to the neural network to be trained and evaluated. Every dataset also has a different number of classes, the groups in which data are separated.

Regarding the technical details, we used a laptop computer with 6GB of RAM and an Intel Core i7 processor regarding all datasets except Fashion - Mnist, which was tested on a desktop computer with 16 GB of RAM and an Intel Xeon W 2123 processor, clocked at 3.60 GHz.

Table 4.2: Characteristics of smaller datasets.

| Filename(.mat) | Instances | Features | Classes |
|---|---|---|---|
| lung | 203 | 3312 | 5 |
| lung_discrete | 73 | 325 | 7 |
| TOX_171 | 171 | 5748 | 4 |
| CLL_SUB_111 | 111 | 11340 | 3 |
| COIL20 | 1440 | 1024 | 20 |

**Lung** and **Lung_discrete**. These data were used by Hong and Young to illustrate the power of the optimal discriminant plane even in ill-posed settings. Applying the KNN method in the resulting plane gave 77% accuracy. However, these results are strongly biased. The data described three types of pathological lung cancers. The authors give no information on the individual variables nor on where the data was originally used [176].

**TOX-171**. This database is an example of the use of toxicology to integrate diverse biological data, such as clinical chemistry, expression, and other types of data. The database contains the

profiles resulting from the three toxicants: alpha-naphthyl-isothiocyanate, dimethylnitrosamine, and N-methylfor-ma-mide administered to rats. The classification task is to identify whether the samples are toxic, non toxic or control. Sample is toxic if alpha-naphthylisothiocyanate, or dimethylnitrosamine or n-methylfor-ma-mide is administered, non-toxic if caerulein or dinitrophenol or rosiglitazone is administered and control if untreated [177].

**CLL-SUB-111**. The database has gene expressions from high density oligonucleotide arrays containing genetically and clinically distinct subgroups of B-cell chronic lymphocytic leukemia (B-CLL). The dataset is formed of 11340 attributes and 111 instances, as referred in Table 4.2 [177].

**COIL20**. Columbia Object Image Library (COIL-20) is a database of gray-scale images of 20 objects. The objects were placed on a motorized turntable against a black background.The turntable was rotated through 360 degrees to vary object pose with respect to a fixed camera. Images of the objects were taken at pose intervals of 5 degrees. This corresponds to 72 images per object. The database has two sets of images. The first set contains 720 unprocessed images of 10 objects. The second contains 1,440 size normalized images of 20 objects. COIL-20 is available online via ftp [178].

**Fashion-Mnist**. Fashion-MNIST is a new dataset, consisting of 70000 samples, that represent fashion items. Each one of them is a $28 \times 28$ grayscale image, belonging to one of 10 different categories. There are about 7000 samples per category. The dataset is separated from the training set and the testing set, which contains 60, 000 and 10, 000 images, respectively. Fashion-MNIST is intended to replace the original MNIST dataset for benchmarking machine learning algorithms, taking into account that it has the same structure as MNIST does[170].

Furthermore, the first 5 datasets, mentioned in Table 4.2, are split in two parts, the training and the testing set. The $\frac{2}{3}$ of the total size of the dataset is used in order for the neural network to be trained, and the $\frac{1}{3}$, remaining, is used for testing its ability to learn the training set. Regarding the last dataset, Fashion-Mnist is already separated in training and testing parts, with 60000 and 10000 samples, respectively.

The method, used for preprocessing features, in order for them to be converted into a form that can be more "understandable" to our deep learning algorithms, is one-hot encoding, which creates new (binary) columns, indicating the presence of each possible value from the original data.

An MLP (Multilayer Perceptron) neural network model is used in all cases. More specifically, it is about a neural network with an input level, three hidden layers, and one output level. Each hidden layer has 1000 neurons, and the number of neurons in the input and output levels depends on the dataset features. The application of our algorithms to other feed-forward networks, such as convolutional neural networks (CNN), is possible. So, It is necessary to deal with their particularities and introduce some more algorithmic techniques, which we are currently developing in order for our algorithms to be adapted to the function of the convolutional-pooling

layers instead of simply introducing the existing techniques into the final fully-connected layers of a CNN, as it was done by the SET algorithm [17]. Moreover, inspiration from the ideas developed in [179] could also assist in this goal.

**Optimizers and activation functions**. In our experiments, we use stochastic gradient descent with momentum with parameter $\gamma$ equal to 0.9. For the learning rate $\alpha$ and batch size parameter $bs$, we used the values of 0.01 and 2, respectively.

As activation functions we used the popular ReLU and FReLU functions for the hidden layers, and the sigmoid function for the last level. The activation functions we used are described below. The Rectified Linear Unit (ReLU) is defined as follows : $ReLU(x) = (x > 0)?x : 0$, and the Flexible Rectified Linear Unit (FReLU) is defined in [180] as follows: $FReLU(x) = ReLU(x) + b$ with $b$ being a learnable parameter.

All algorithms functioned repetitively until convergence; here, for visual comparison purposes, we show the performance after the first 500 epochs when we practically have convergence, except for some very few cases where converge is achieved after 50 epochs. We also show the accuracy calculated at each time according to the mean squares error method. Each training attempt was repeated five times, and the performance was averaged.

Finally, in cases where we implement regularization techniques, the weight decay parameter ($wdec$) was set to 0.0002 for L2 regularization, and 0.0000001 for L1 regularization method. Moreover, the number of connections being modified depends on the value of $\zeta$ parameter, which is set to 0.3 in our experiments. In other words, the $\zeta = 30\%$ of the real connections of the network are pruned and restored in every epoch. All hyperparameters used are illustrated in Table 4.3. We use prediction accuracy as the performance measure.

In the following subsections, we show for each dataset and each competitor the "evolution" of the accuracy achieved until the last epoch; at the end of these sets of plots, we show aggregate results concerning the average accuracy achieved during the last epochs, and also the average total training time required. However, some methods did not converge in "reasonable" time while processing some of the datasets, so these measurements are excluded from the presented results.

#### 4.1.3.2 Evaluation results

In all Figures presented in the sequel, the x-axis represents the epochs; each one of them is a single step in training a neural network. The y-axis describes the percentage evaluation between the predicted and target value (i.e., accuracy).

In the subsequent Figures, we present the results of our methods in comparison with the accuracy of both the Fully-Connected (FC) MLP network and the results of the SET algorithm in all datasets mentioned.

Figures 4.1, 4.2 display all algorithms' accuracy, taking into account the Lung.mat dataset. Specifically, the accuracy that all algorithms achieve is depicted with respect to the epochs.

Table 4.3: List of hyperparameters, used.

| **List of hyperparameters** | |
|---|---|
| **Number of layers** | 1 input layer (with #neurons,equal to the features of dataset)<br>3 hidden layers<br>1 output layer (with #neurons,equal to the classes of dataset) |
| **Number of neurons per layer** | 1000 |
| **Activation functions** | ReLU<br>FReLU<br>Sigmoid (in last layer) |
| **weights regularization techniques** | **L1, L2** |
| **weight_decay** ($wdec$) | 0.0000001 for L1<br>0.0002 for L2 |
| **batch size** ($bs$) | 2 |
| **learning_rate** ($\alpha$) | 0.001 |
| **momentum** ($\gamma$) | 0.9 |
| **percentage of connections pruned/restored** ($\zeta$) | 30% |

In the top Figure 4.1, we see the results that ReLU activation produces for Lung.mat dataset, and in the bottom plot, the results produced by FReLU function. When we apply the SF2SFrand algorithm (which starts from a randomly constructed network and ends up in a scale-free-like one), we see that the accuracy varies from 70% to 99%, and we conclude that the algorithm has a better performance when L2 regularization is used (approximately 93% accuracy, when using FReLU).

So, L2 (L2 regularization makes the loss function smoother) manages to decrease the noise in the training data so that the estimated coefficients (weights) can generalize well to the future data. According to the time, the best performance is achieved in 10 minutes when SF2SFrand is used, which means this kind of modification that this algorithm does in topology, positively affects the network. Similar performances are shown in Figures 4.3 to 4.10 during the execution of the algorithm, using the other datasets which are mentioned in Table 4.2.

In Figures 4.1 to 4.10, we illustrate the results for the algorithm that starts from a scale-free network and, through the procedure of training using back-propagation, ends up constructing a scale-free network again (SF2SFba). This means that the network is always modified following a power law, and hence the randomness is decreased. Using SF2SFba, in turn, implies better communication between the nodes. According to the plot, this variant of the algorithm has 95.6% accuracy using FReLU and without any regularization. So, taking into consideration that FReLU has faster convergence than ReLU [180], concerning also the network topology, we conclude that the less random a network structure is, the higher performance the network gets without any regularization technique. In addition, the algorithm takes 29 minutes (with 92% accuracy) to train the network, using ReLU (recall that this function is computationally efficient since it just outputs zero for negative inputs) and 35 minutes while implementing the FReLU,

which is still better in both time and accuracy than the competitor.

Furthermore, the exact Figures show the accuracy of a variant of scale-free to a scale-free algorithm, described in subsection 4.1.2.3. The algorithm achieves approximately 93% accuracy while implementing FReLU activation function and no regularization techniques due to the faster convergence of this function. Furthermore, both ReLU and FReLU activations affect the network performance positively, except in the case where L1 regularization, combined with FReLU, is used. We have approximately 79% accuracy.

We conclude that L1 regularization technique makes the graph curve falls off abruptly because of a code error in weight update (appears nan values). The algorithm takes approximately the same time to train the network as scale-free to scale-free version does.

Our following technique was the one that started from a scale-free and created a small-world network (SF2SW). Our results (when using p=0.02, aw rewiring probability) show that the accuracy is decreased when a network, following a power-law degree distribution, transits to a more randomly linked topology.

Specifically, in Figures 4.1, 4.2, 4.9 and 4.10 both in ReLU and FReLU implementation, NoL and L1 regularized curves are overlapped (same accuracy) and they differ only in execution time. We see that L1 parameter contributes more efficiently to the network when combined with FReLU function (regarding accuracy).

This approach also has a much larger execution time than the rest, owing to its great computational complexity, as Algorithm 7 implies.

We run the same algorithm with a more significant probability (p=0.075) and show the results in the exact Figures. While the probability becomes larger than the previous one, the average length becomes smaller, which means more connections between the nodes (density). Hence this algorithm needs more time to make all these computations between the nodes (enormous execution time).

In this case, the network, constructed after the training procedure, tends to be more like the same random graph as in Algorithm 7. In particular, a node can be connected to a less powerful node, so, in the next epoch, the information saved in this node, maybe, is not going to be transferred to the next layer because the links reconnected randomly, and so on.

In this last case, we study the performance of the neural network system in terms of using only the small-world method and also depict the results in Figures 4.1, 4.2, 4.9 and 4.10. Using FReLU activation, the algorithm achieves better accuracy (81%) but the problem is that algorithm takes enough time to train the neural network (enormous execution time - because of computational complexity). Also, the rewiring probability is p=0.02 (small enough), meaning the reconstructed network is not random enough. Hence the accuracy is stabilized in high levels of values. Therefore, the less a topology is random, the more efficient the network becomes, in learning and generalizing the data.

Then we use a much larger probability, which makes the graph denser. Not only for its

density, but also for its computational complexity, this variant tends to need more training time. Due to its randomness, information is distributed in every possible node (not in the popular ones), meaning that the information is not retained while passing through the epochs. Those affect the network and so as the accuracy, which is, slow enough (65%). The graphs show that L1 regularized curve has many fluctuations while FReLU function is used. This is because FReLU provides more capacity than ReLU, which leads the model to not generalize well from its training data to unseen data. Moreover, the time needed for the model to be trained while using the Scale-Free-to-Small-World or the Small-World-to-Small-World algorithm is vast enough and the accuracy is moderate regarding the other algorithms' performance, so we test these algorithms only to these two datasets (Lung.mat and COIL20.mat - we chose specifically these, due to the different classification task they do.)

It is worth highlighting that we compare our sparse model results with the performance of the fully-connected, unpruned neural network. As we can see in the above Figures, the accuracy of the fully-connected model can be high enough (up to 95% in some cases) mainly when we have the combination of ReLU and L1 regularization, in contradiction to the cases, we use L1 or no regularization technique and FReLU activation function, in which the model achieves about to 20% accuracy or lower. However the main problems of a fully-connected model are its efficacy, comparatively with the time needed to be trained and its memory requirements, especially when we have devices with memory and storage constraints. In 4.4, we see the memory footprint of all the algorithms, mentioned and in 4.16 we see the overall results of them. In our approaches and specifically, when we implement a scale-free network topology, we reduce the network's training time up to 93% and the memory requirements for the models to be deployed up to 76% in comparison with the unmodified one. At the same time, we achieve better results regarding the accuracy in most cases and almost in all datasets used.

Furthermore, we achieved up to 76% reduction in model size, using the topology mentioned above (three hidden layers of one thousand neurons each of them) and the SF2SFrand algorithm, due to the redundant weights, removed, making our sparse implementations more suitable to run on devices with storage limitations, minimizing the computational cost. It is also of great importance to mention that one of our algorithms -the SF2SFrand algorithm- needs about 1% less storage than SET algorithm, while it has a better efficacy (it is much faster than SET while preserving approximately the same accuracy as SET)as it is shown in 4.7, which is the only former work, close to the concept of ours, presented in the literature.

Hence, there is no reason to show the exact training time results of the unpruned model (due to the significant deviation, regarding our methods).

We summarize the average accuracy (just before convergence), and the average training times of the algorithms in Figures 4.17–4.18. There we can see that SF2SFrand achieves almost

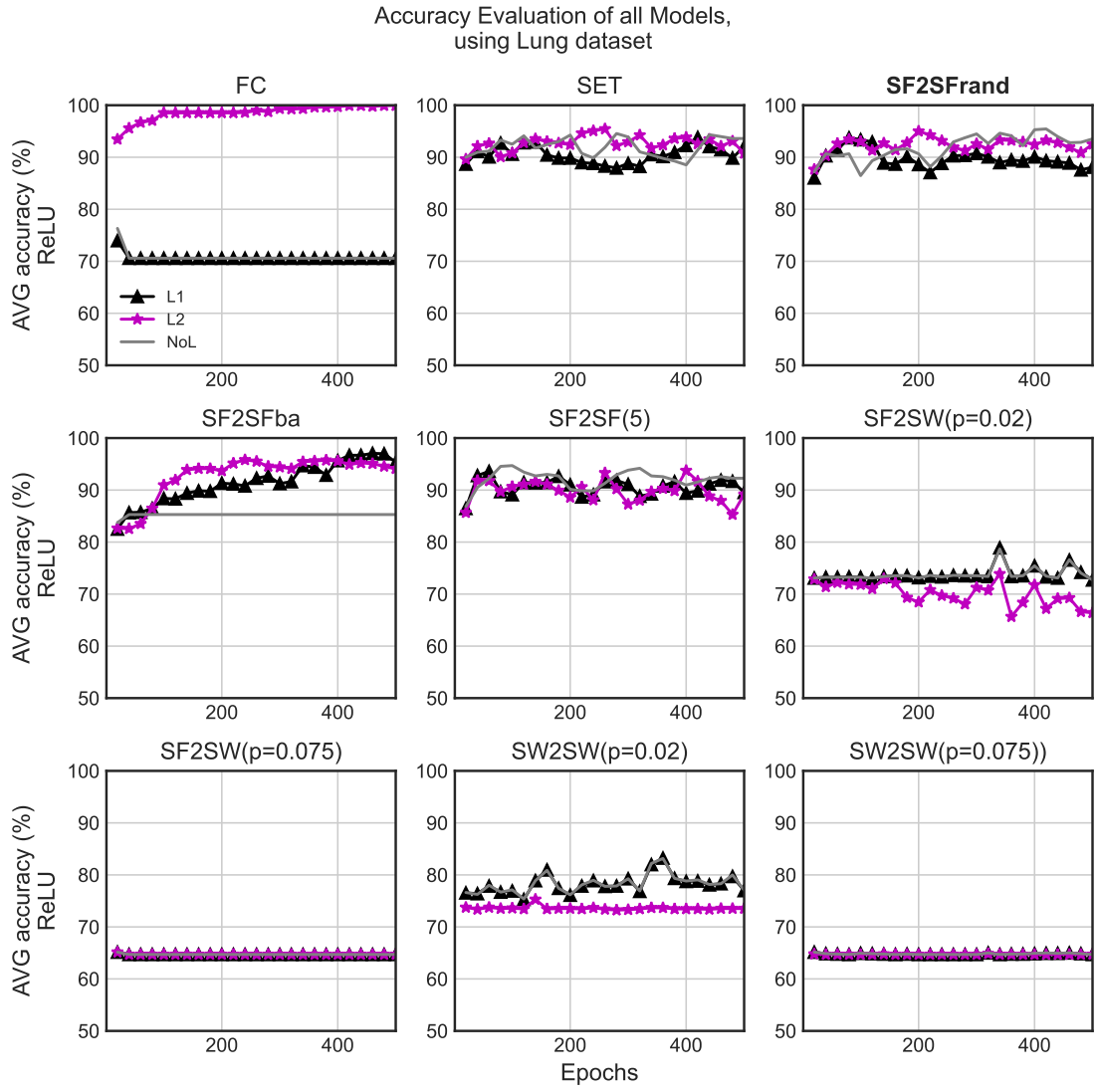



Figure 4.1: Lung dataset overall results, using ReLU activation function.

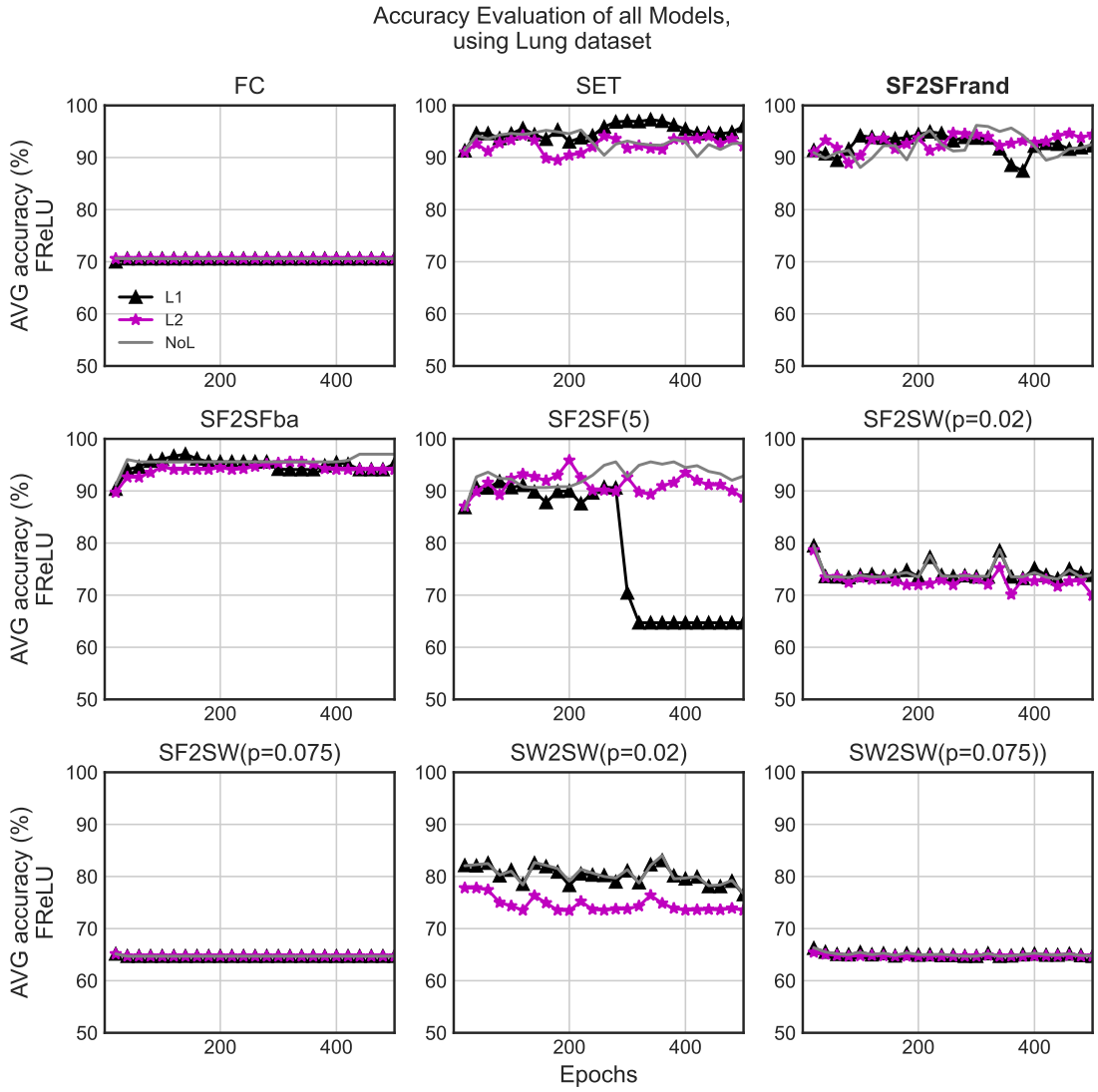



Figure 4.2: Lung dataset overall results, using FReLU activation function.

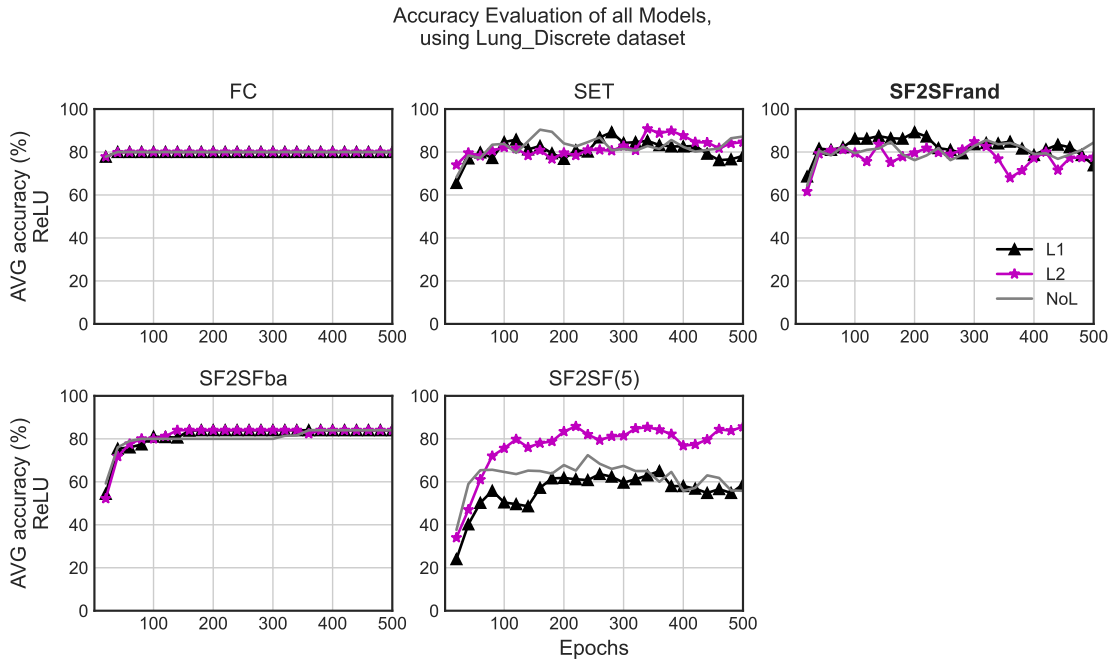



Figure 4.3: Lung Discrete dataset overall results, using ReLU activation function.

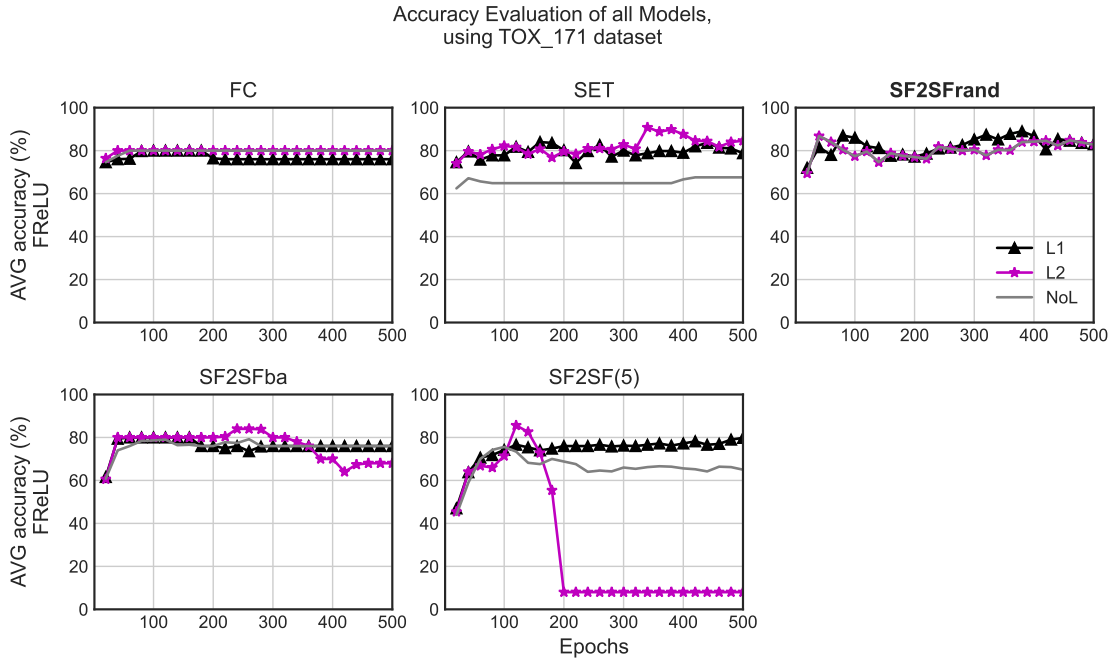



Figure 4.4: Lung Discrete dataset overall results, using FReLU activation function.

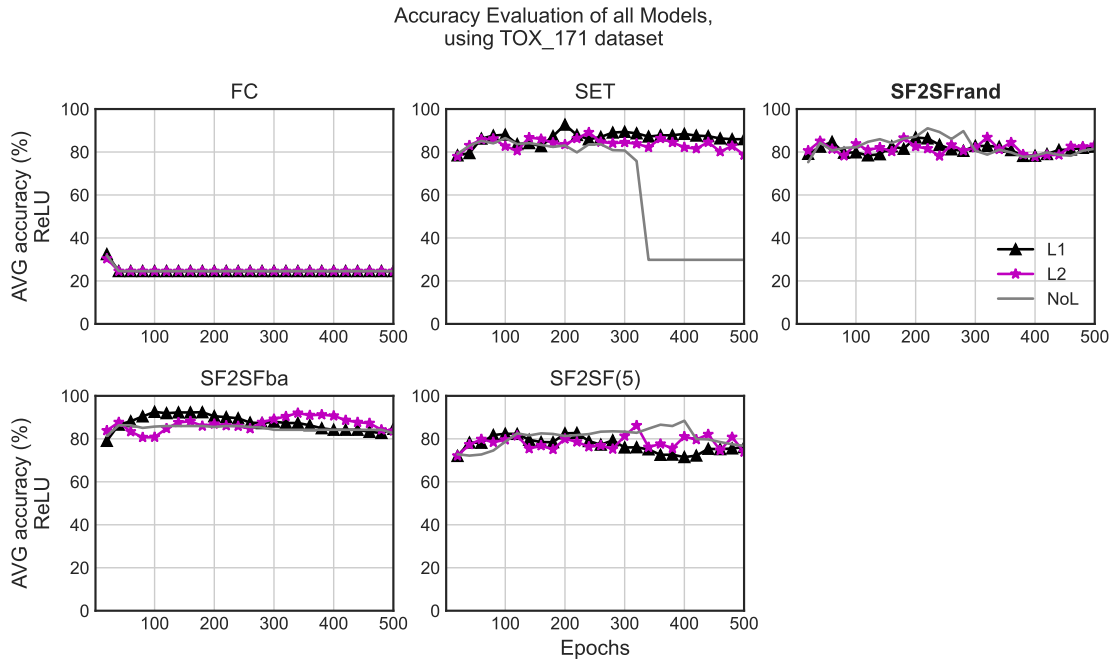



Figure 4.5: TOX_171 dataset overall results, using ReLU activation function.

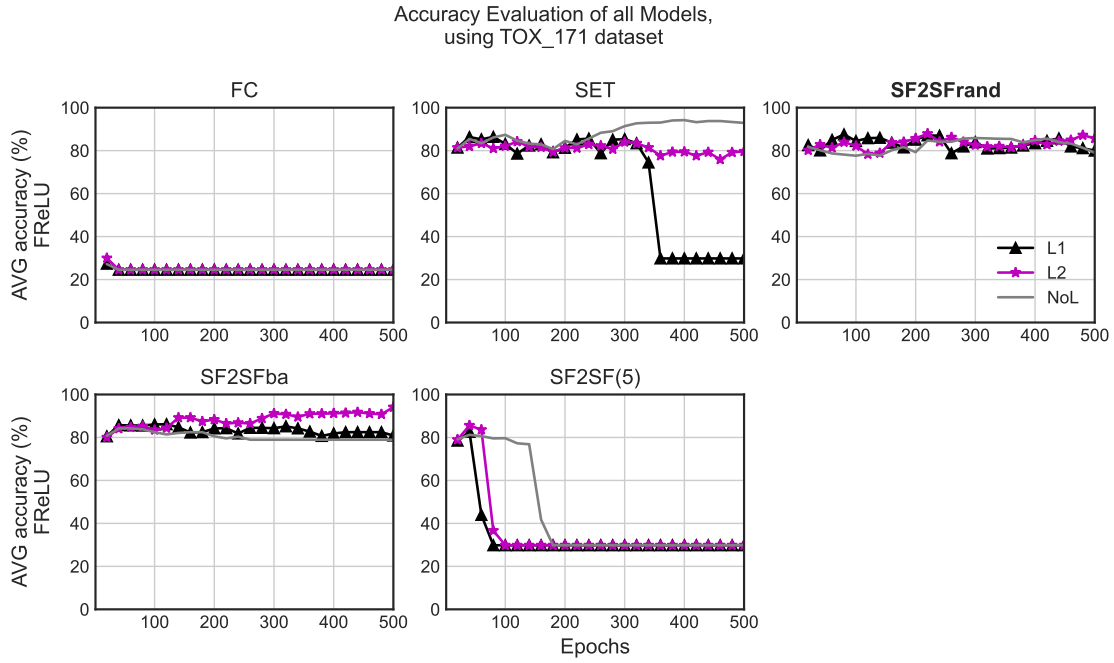



Figure 4.6: TOX_171 dataset overall results, using FReLU activation function.

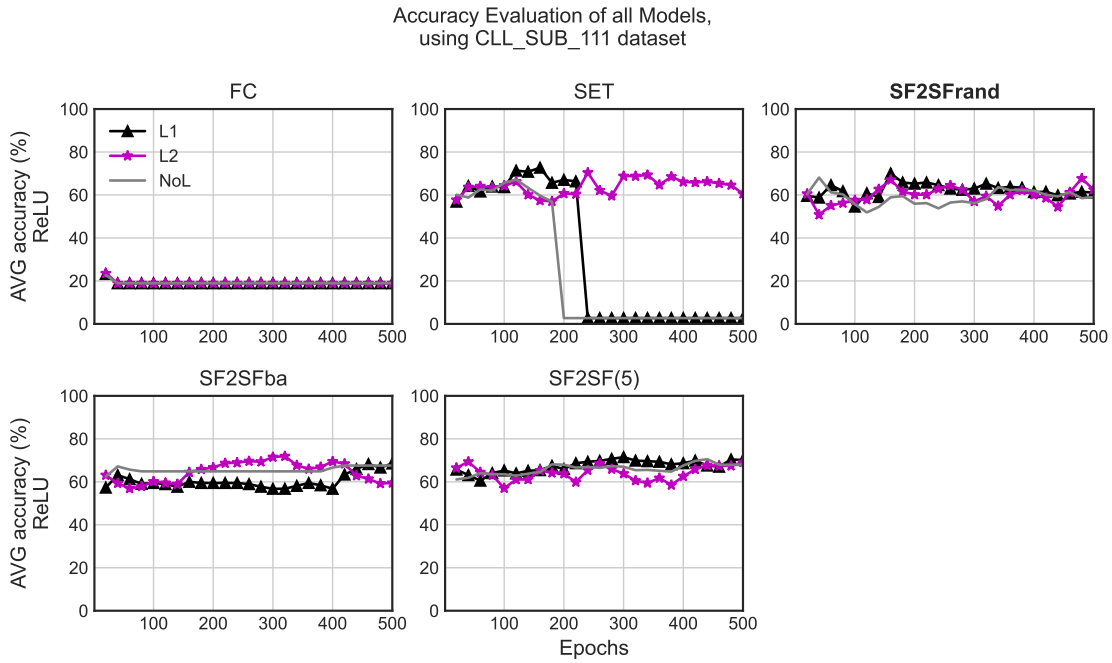



Figure 4.7: CLL_SUB_111 dataset overall results, using ReLU activation function.

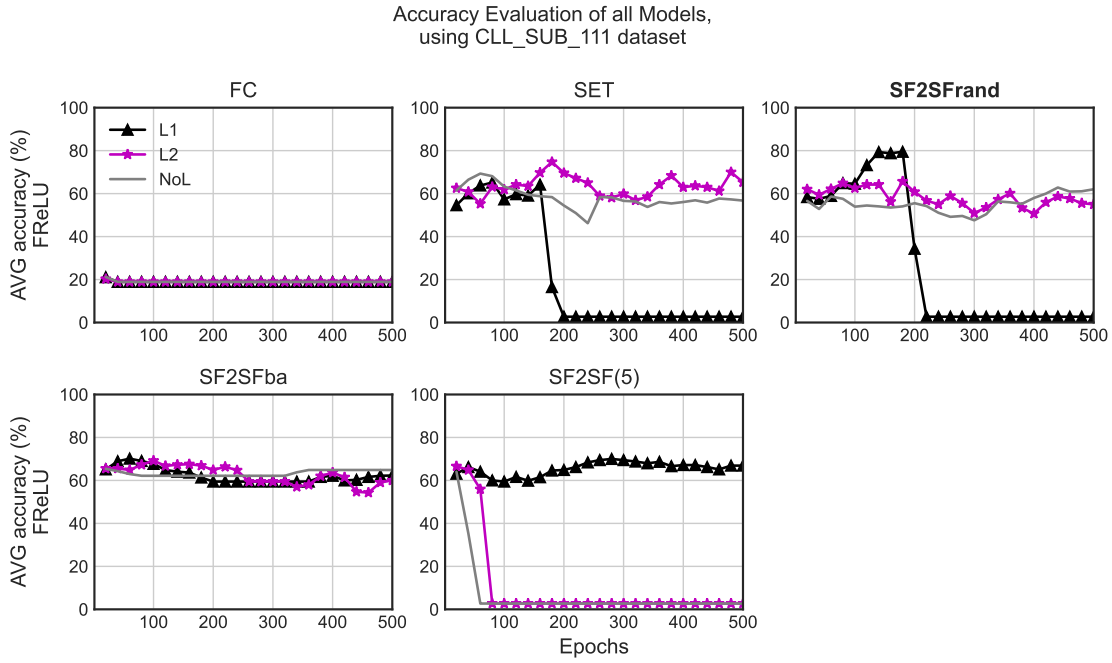



Figure 4.8: CLL_SUB_111 dataset overall results, using FReLU activation function.

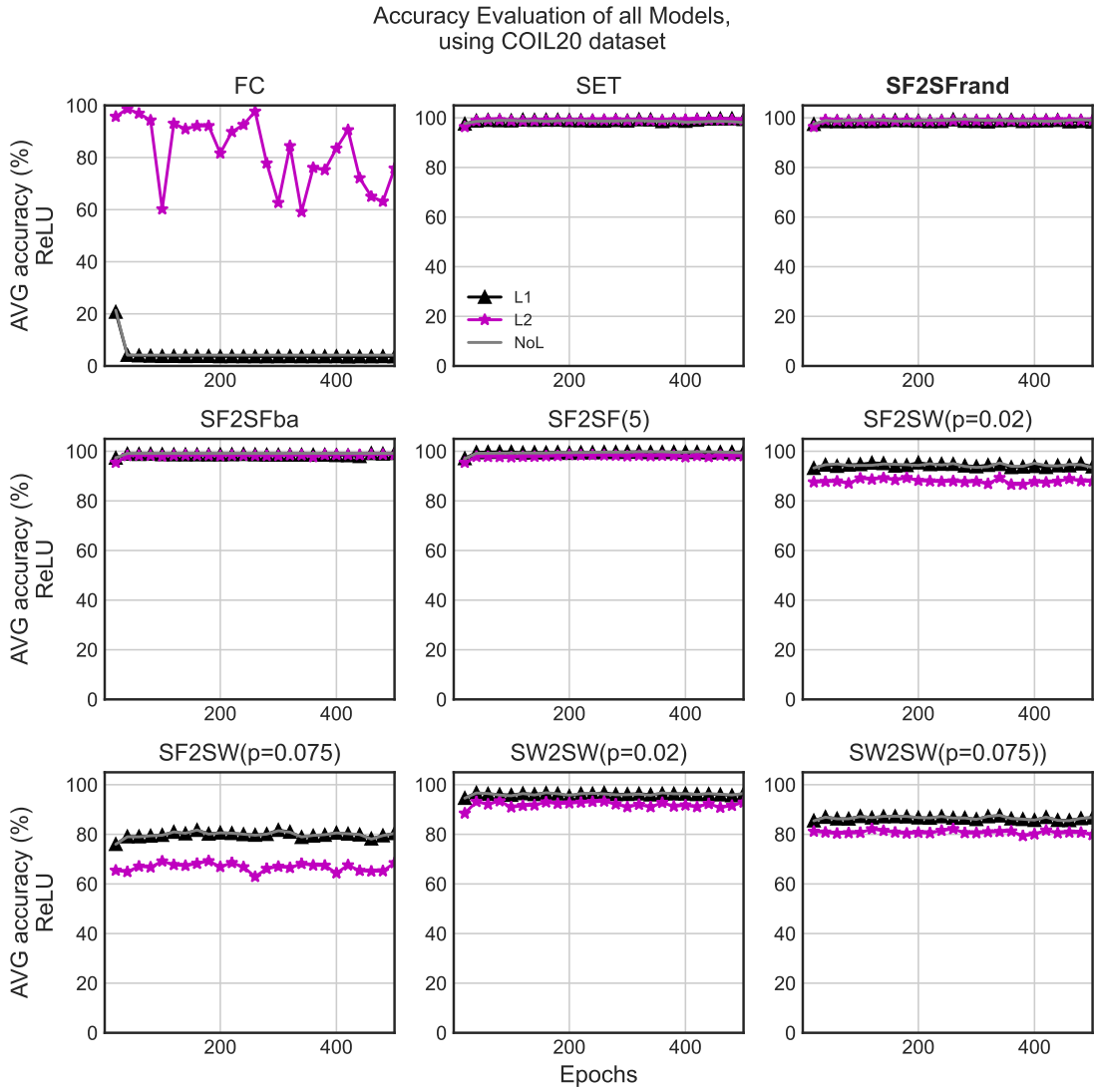



Figure 4.9: COIL20 dataset overall results, using ReLU activation function.

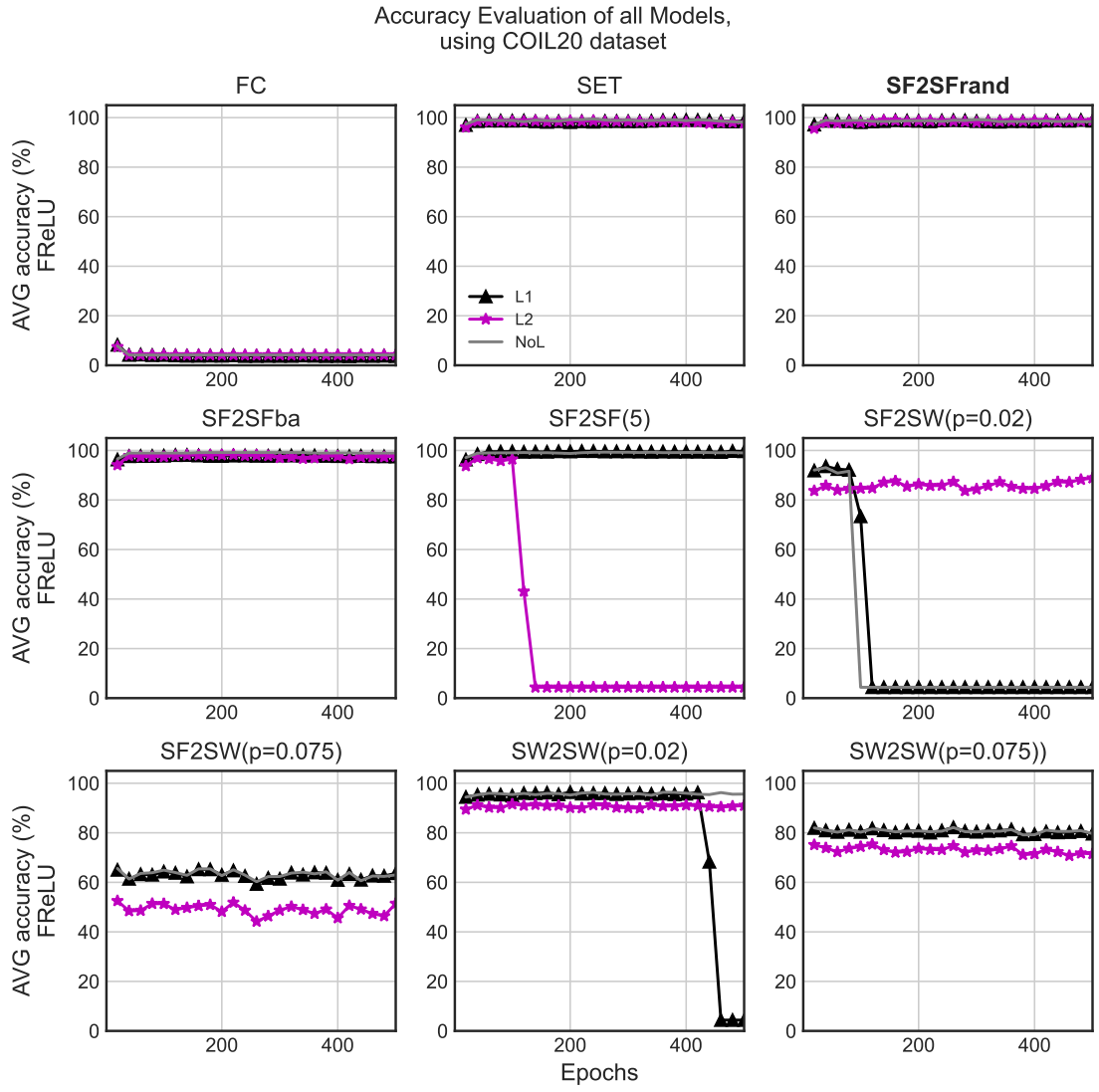



Figure 4.10: COIL20 dataset overall results, using FReLU activation function.

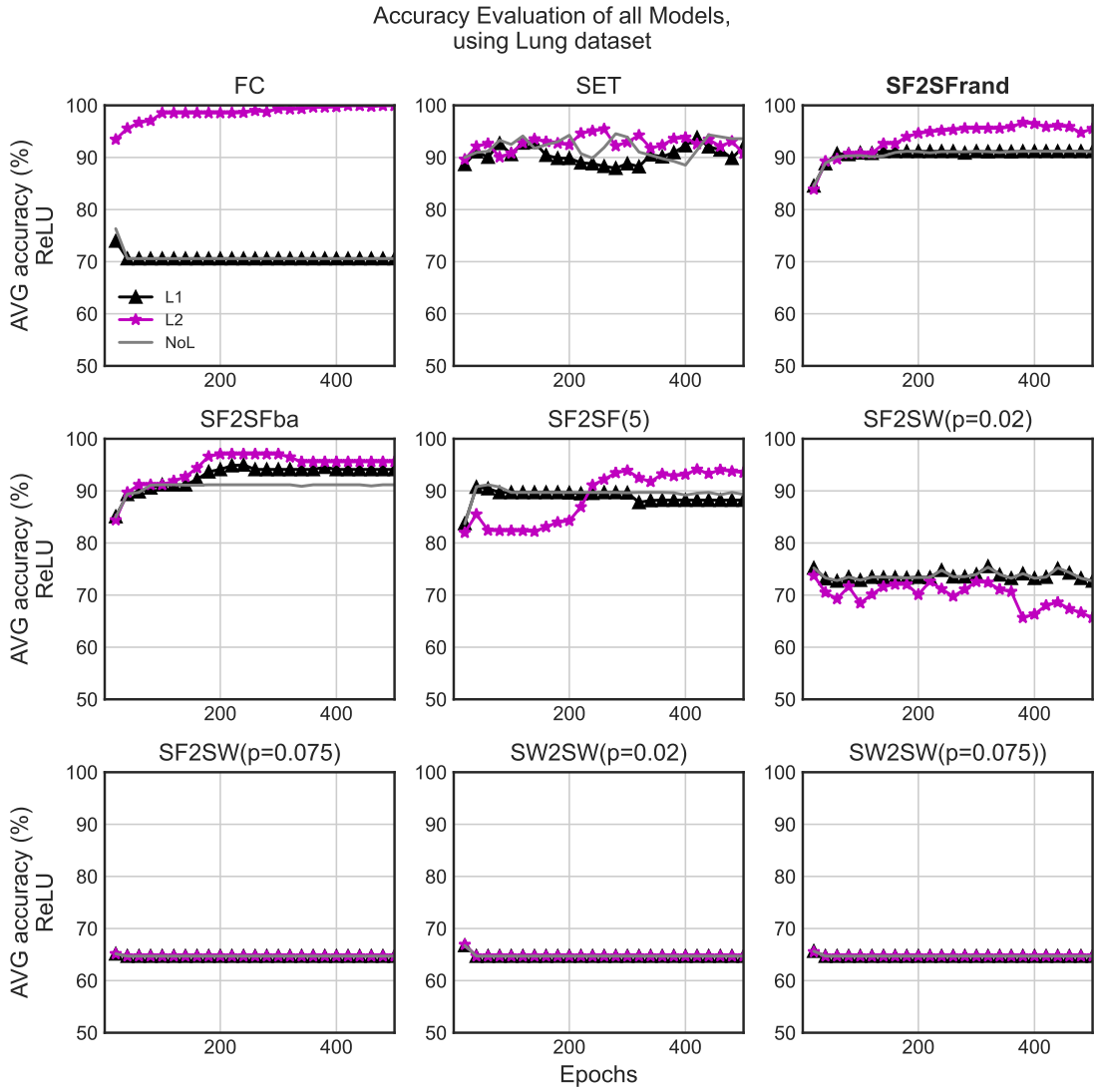



Figure 4.11: Results of MLP(FC) and AVG-Weighted algorithms on Lung dataset, using ReLU activation function.

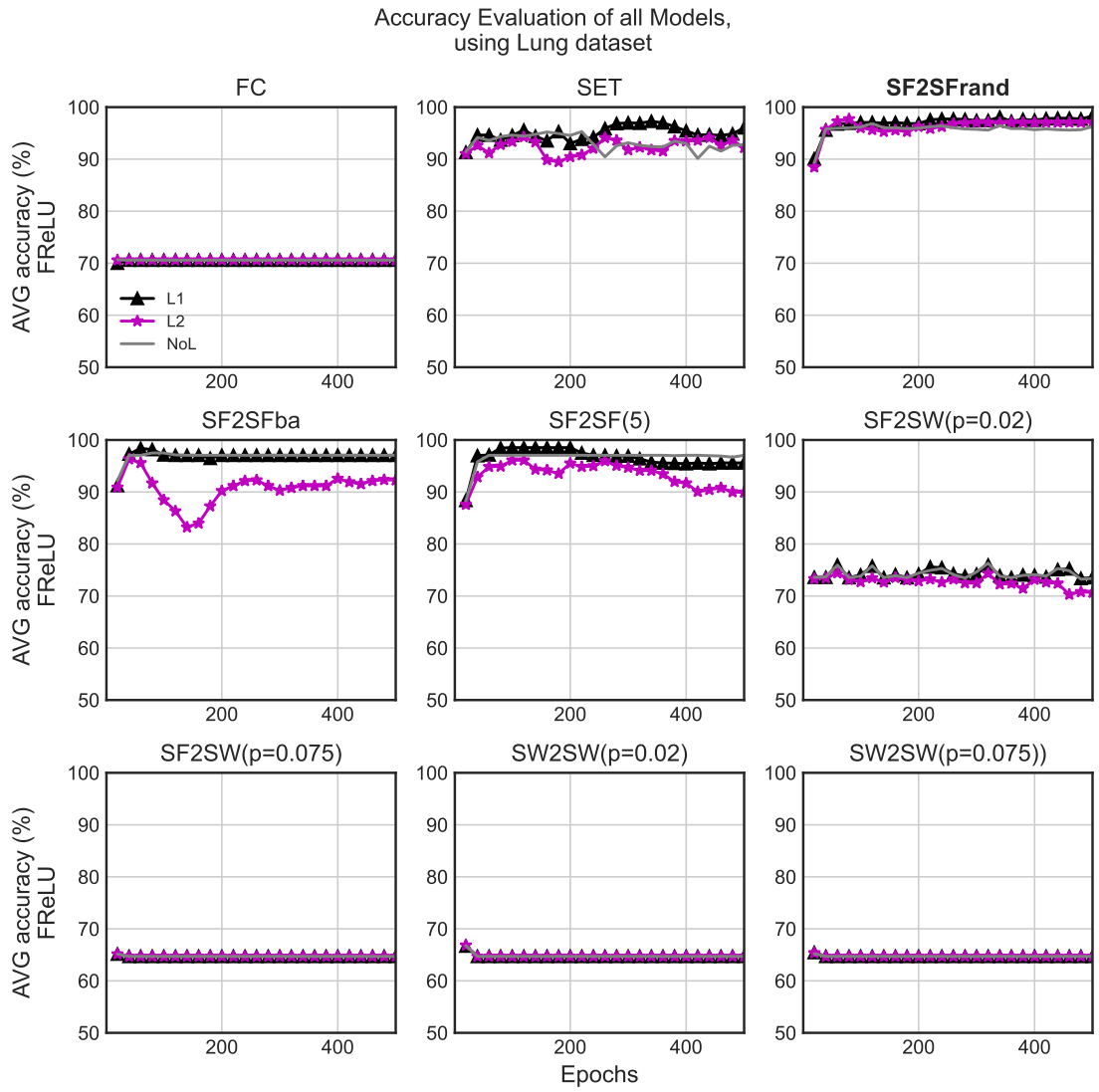



Figure 4.12: Results of MLP(FC) and AVG-Weighted algorithms on Lung dataset, using FReLU activation function.

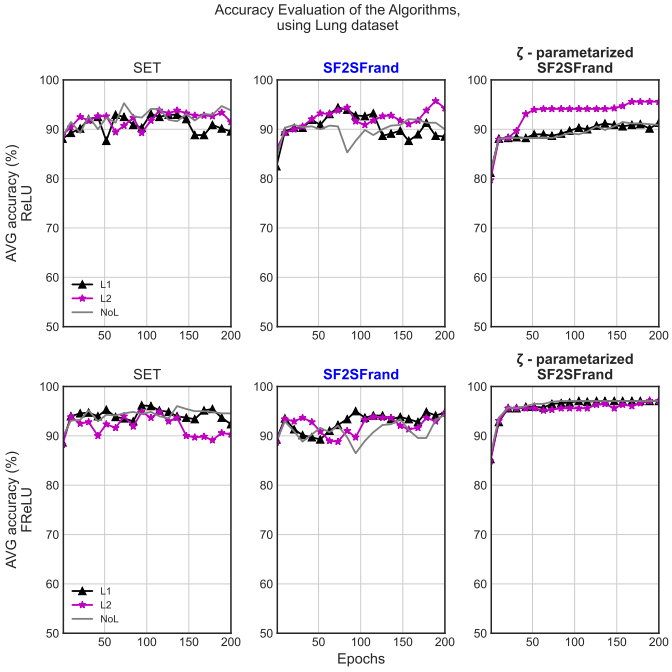



Figure 4.13: Results of SET, SF2SFrand ζ-parametarized SF2SFrand, using both ReLU and FRelU activation function and Lung dataset.

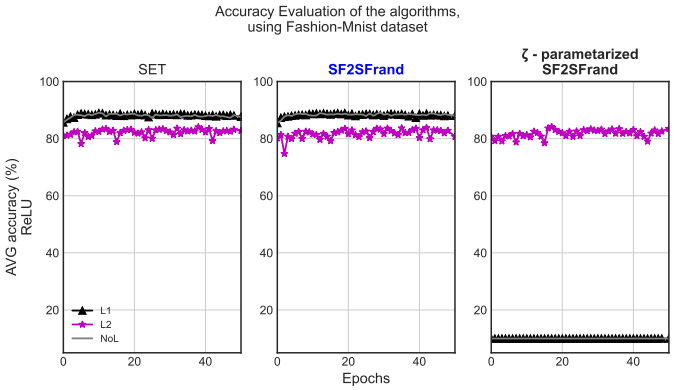



Figure 4.14: Results of SET, SF2SFrand ζ-parametarized SF2SFrand, using ReLU activation function and Fashion-Mnist dataset.

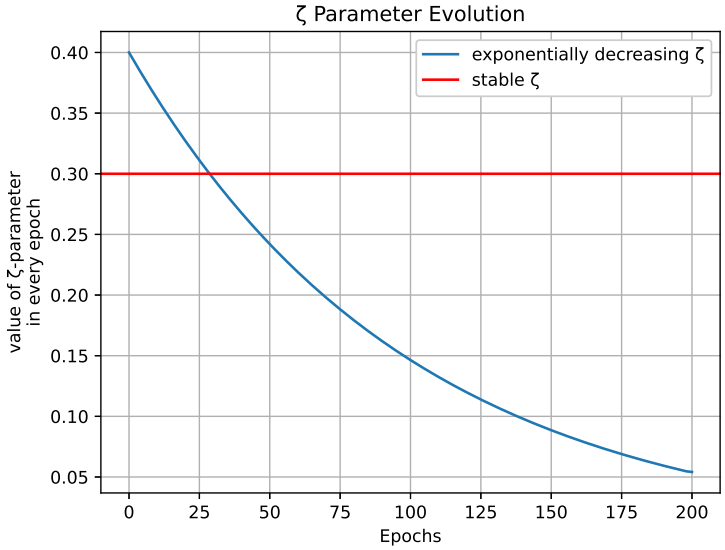



Figure 4.15: Evolution of $\zeta$ in every epoch.

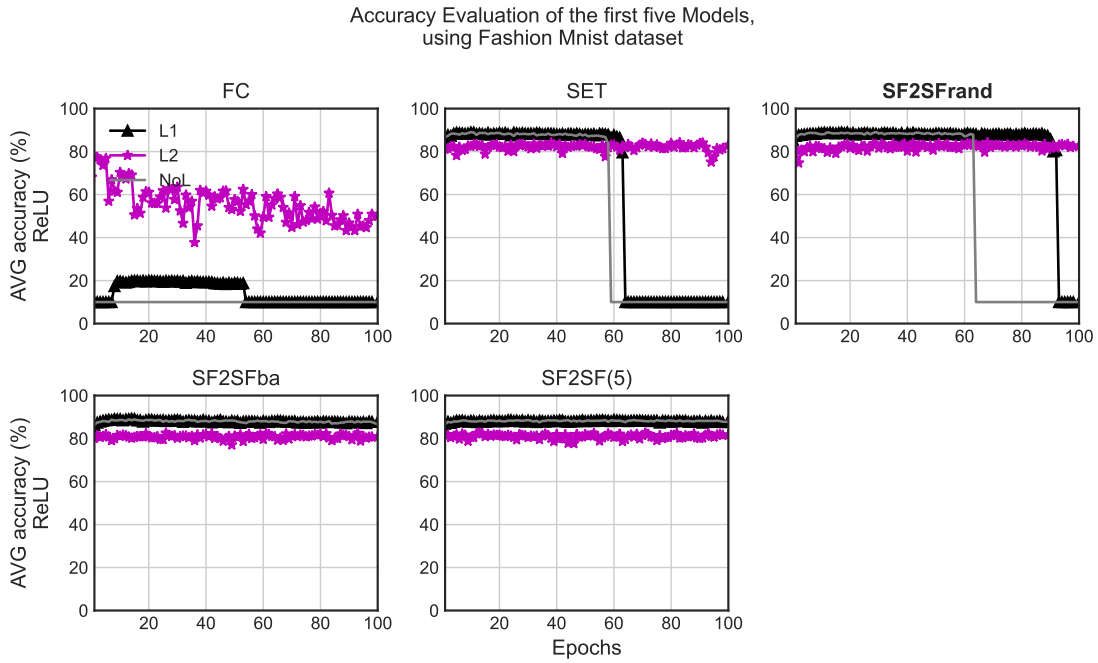



Figure 4.16: Overall results on five algorithms, using Fashion-Mnist dataset.

Table 4.4: Memory Footprint in MegaBytes(MB), between the Dense Network (MLP), the Sparse Network using SET algorithm and our proposed algorithms, which are SF2SFrand, SF2SFba abd SF2SF(5), SF2SW and SW2SW.

| Memory Footprint | | | | |
|---|---|---|---|---|
| **Algorithm** | **Kind of Network** | **Hidden Layer Architecture** | **Size (MB)** | **Percentage reduction** |
| MLP (FC) | Fully-Connected | 1000 - 1000 - 1000 | 77.68 | —- |
| SET | Sparse | 1000 - 1000 - 1000 | 19.18 | 75.28 % |
| **SF2SFrand** | Sparse | 1000 - 1000 - 1000 | 18.39 | 76.31 % |
| SF2SFba | Sparse | 1000 - 1000 - 1000 | 20.68 | 73.36 % |
| SF2SF(5) | Sparse | 1000 - 1000 - 1000 | 20.18 | 74.00 % |
| SF2SW | Sparse | 1000 - 1000 - 1000 | 19.84 | 74.44 % |
| SW2SW | Sparse | 1000 - 1000 - 1000 | 21.30 | 72.55 % |

Table 4.5: Measures of MLP (FC).

| Measures - MLP (FC) | | | | | |
|---|---|---|---|---|---|
| Datasets-Activ.Fun.+Regul | Lung *Acc/Time* | Lung_Discrete *Acc/Time* | TOX_171 *Acc/Time* | CLL_SUB_111 *Acc/Time* | COIL20 *Acc/Time* |
| ReLU + NoL | 70.58% /1h 15 min | 80% / 5 min | 24.56% /75 min | 19.07% /1h 5 min | 4.1% /2h 8 min |
| ReLU + L1 | 70.58% /2h 22 min | 80 % / 9 min | 24.56% /1h 26 min | 18.91% /1h 35 min | 3.5% /9h 2 min |
| ReLU + L2 | 99.52% /1h 23 min | 80 % / 7 min | 24.56% /1h 21 min | 18.91% /1h 31 min | 62.5% /6h 3 min |
| FReLU + NoL | 70.58% /1h 28 min | 80 % /8 min | 24.56% /1h 03 min | 18.81% /1h 9 min | 4.1% /2h 56 min |
| FReLU + L1 | 70.58% /2h | 76 % / 11 min | 24.56% /1h 37 min | 18.91% /1h 39min | 3.5% /8h 15 min |
| FReLU + L2 | 70.58% /1h 26 min | 80 % / 10 min | 24.56% /1h 26 min | 18.91% /1h 34 min | 4.1% /3h 36 min |

Table 4.6: Measures of SET.

| Measures - SET | | | | | |
|---|---|---|---|---|---|
| Datasets-Activ.Fun.+Regul | Lung *Acc/Time* | Lung_Discrete *Acc/Time* | TOX_171 *Acc/Time* | CLL_SUB_111 *Acc/Time* | COIL20 *Acc/Time* |
| ReLU + NoL | 92.02% /35 min | 82.59% / 11 min | 38.9% /23 min | 2.3% /45 min | 98.75% /47 min |
| ReLU + L1 | 90.53% /47 min | 80.90% /13 min | 86.51% /35 min | 3.3% /55 min | 99.37% /1h 28 min |
| ReLU + L2 | 92.71% /38 min | 82.10%/ 11 min | 83.69% /27 min | 65.84% /45 min | 99.16% /54 min |
| FReLU + NoL | 93.02% /45 min | 65.50% /13 min | 88.43% /32 min | 58.15% /56 min | 98.12% /1h 25 min |
| FReLU + L1 | 95.04% /53 min | 79.62% /12 min | 38.2% /36 min | 2.1% /58 min | 98.75% /2h |
| FReLU + L2 | 92.38% /49 min | 82.11% /11 min | 80.90% /33 min | 63.54% /57 min | 98.12% /1h 30 min |

Table 4.7: Measures of SF2SFrand.

| **Measures - SF2SFrand** | | | | | |
|---|---|---|---|---|---|
| Datasets-Activ.Fun.+Regul | Lung *Acc/Time* | Lung_Discrete *Acc/Time* | TOX_171 *Acc/Time* | CLL_SUB_111 *Acc/Time* | COIL20 *Acc/Time* |
| ReLU + NoL | 91.8% /10 min | 80% /6 min | 82.5% /10 min | 58.9% /15 min | 98.95% /36 min |
| ReLU + L1 | 89.6% /15 min | 82.4% /8 min | 81.4% /18 min | 62.4% /19 min | 97.91% /1h 8 min |
| ReLU + L2 | 92.3% /11 min | 77.6% /7 min | 81.6% /12 min | 60% /14 min | 98.95% /44 min |
| FReLU + NoL | 92.1% /13 min | 80.3% /6 min | 82.3% /16 min | 55.4% /18 min | 98.75%/1h 14min |
| FReLU + L1 | 92.4% /21 min | 82.5% /7 min | 83.3%/22 min | 2.7% /17 min | 99.16%/1h 42min |
| FReLU + L2 | 92.9% /16 min | 80.3% /6 min | 83.3% /14 min | 58.2% /15 min | 98.54%/1h 28min |

Table 4.8: Measures of SF2SFba.

| Measures - SF2SFba | | | | | |
|---|---|---|---|---|---|
| Datasets-Activ.Fun.+Regul | Lung *Acc/Time* | Lung_Discrete *Acc/Time* | TOX_171 *Acc/Time* | CLL_SUB_111 *Acc/Time* | COIL20 *Acc/Time* |
| ReLU + NoL | 85.2% /29 min | 81% /15 min | 84.8% /34 min | 65.5% /1h | 99.16% /53 min |
| ReLU + L1 | 91.5% /32 min | 81.4% /18 min | 87.5% /39 min | 64.5% /1h 5min | 99.16% /1h 22min |
| ReLU + L2 | 92.8% /30 min | 82.4% /15 min | 86.8% /37 min | 60.4% /1h 7min | 98.75% /1h 18min |
| FReLU + NoL | 95.6% /32 min | 78.9% /18 min | 80.4% /38 min | 63.3% /1h | 98.95% /1h 29min |
| FReLU + L1 | 94.9% /41 min | 79.3% /20 min | 83.5%/1h 19min | 62.4% /1h 5min | 97.5% /1h 58min |
| FReLU + L2 | 94.1% /39 min | 76% /18 min | 95% /42 min | 62.6% /1h 12min | 97.29% /1h 51min |

Table 4.9: Measures of SF2SF-5.

| Measures - SF2SF(5) | | | | | |
|---|---|---|---|---|---|
| Datasets-Activ.Fun.+Regul | Lung *Acc/Time* | Lung_Discrete *Acc/Time* | TOX_171 *Acc/Time* | CLL_SUB_111 *Acc/Time* | COIL20 *Acc/Time* |
| ReLU + NoL | 92% /28 min | 62.4% /13 min | 81% /40 min | 65.92% /1h 15 min | 99.42% /41 min |
| ReLU + L1 | 90.60% /37 min | 59.7% /13 min | 77.2% /50 min | 70.27% /1h 20 min | 99.49% /45 min |
| ReLU + L2 | 89.84% /30 min | 90.6% /14 min | 77.2% /43 min | 70.27% /1h 13 min | 97.98% /45 min |
| FReLU + NoL | 92.8% /35 min | 70.1% /13 min | 30.4% /39 min | 9.8% /55 min | 99.79% /1h 13 min |
| FReLU + L1 | 64.70% /37 min | 80.3% /14 min | 30.4% /34 min | 66.7% /1h 26 min | 99.79% /1h 25 min |
| FReLU + L2 | 91.2% /37 min | 10.5% /11 min | 30.4% /33 min | 9.8% /57 min | 4.3% /1h 7 min |

Table 4.10: Measures of SF2SW.

| Measures - SF2SW | | | | | |
|---|---|---|---|---|---|
| **p = 0.02** | | | **p = 0.075** | | |
| Datasets-Activ.Fun.+Regul. | Lung *Acc/Time* | COIL20 *Acc/Time* | Datasets-Activ.Fun.+Regul. | Lung *Acc/Time* | COIL20 *Acc/Time* |
| ReLU + NoL | 74.7% /4h 13 min | 94.36% /2h | ReLU + NoL | 64.7% /6h 3 min | 79.85% /4h 3 min |
| ReLU + L1 | 74.8% /4h 17 min | 94.35% /2h 48 min | ReLU + L1 | 64.7% /6h 20 min | 79.85% /4h 30 min |
| ReLU + L2 | 73.52% /4h 14 min | 89.37% /2h 13 min | ReLU + L2 | 64.7% /6h 15 min | 66.79% /4h 8 min |
| FReLU + NoL | 74.52% /4h 20 min | 04.37% /2h 19 min | FReLU + NoL | 64.7% /6h 52 min | 63.26% /4h 37 min |
| FReLU + L1 | 74.52% /4h 17 min | 04.37% /3h 47 min | FReLU + L1 | 64.7% /6h 20 min | 62.99% /5h 12 min |
| FReLU + L2 | 74.52% /4h 19 min | 85.83% /2h 46 min | FReLU + L2 | 64.7% /6h 54 min | 52.29% /4h 55 min |

Table 4.11: Measures of SW2SW.

| Measures - SW2SW | | | | | |
|---|---|---|---|---|---|
| **p = 0.02** | | | **p = 0.075** | | |
| Datasets-Activ.Fun.+Regul. | Lung *Acc/Time* | COIL20 *Acc/Time* | Datasets-Activ.Fun.+Regul. | Lung *Acc/Time* | COIL20 *Acc/Time* |
| ReLU + NoL | 80% /4h 9 min | 96.14% /2h 7 min | ReLU + NoL | 64.8% /6h 3 min | 86.58% /3h 28 min |
| ReLU + L1 | 80% /4h 12 min | 96.15% /2h 44 min | ReLU + L1 | 64.8% /6h 55 min | 86.57% /4h 4 min |
| ReLU + L2 | 78% /4h 8 min | 95.41% /2h 7 min | ReLU + L2 | 64.7% /6h 6 min | 80.81% /3h 35 min |
| FReLU + NoL | 81% /4h 13 min | 95.62% /2h 41 min | FReLU + NoL | 65% /6h 5 min | 80.67% /4h 4 min |
| FReLU + L1 | 81% /5h | 04.37% /3h 14 min | FReLU + L1 | 65% /5h 50 min | 80.63% /4h 40 min |
| FReLU + L2 | 78% /4h 15 min | 90.62% /2h 52 min | FReLU + L2 | 65% /6h 11 min | 73.00% /4h 10 min |

Table 4.12: Measures of experiments with AVG weighted algorithms on Lung dataset.

| Measures - Experiments with AVG weights in Lung.mat dataset | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Algorithm-Activ.Fun.+Regul. | MLP (FC) *Acc/Time* | SET *Acc/Time* | **SF2SFrand** *Acc/Time* | Sf2SFba *Acc/Time* | SF2SF(5) *Acc/Time* | SF2SW-0.02 *Acc/Time* | SF2SF-0.075 *Acc/Time* | SW2SW-0.02 *Acc/Time* | SW2SW-0.075 *Acc/Time* |
| ReLU + NoL | 70.58% /1h 15 min | 92.02% /35 min | 90% /1h 20min | 90.7% /32min | 89.54% / 29min | 73.79% /4h 16min | 64.72% /6h 49min | 64.78% 4h 29min | 64.74% 6h 26min |
| ReLU + L1 | 70.58% /2h 22 min | 90.53% /47 min | 93.7% / 1h 22min | 92.9% /38min | 88.90%/31min | 73.69% /4h 20min | 64.72% 6h 54min | 64.78% 4h 35min | 64.74% 6h 37min |
| ReLU + L2 | 99.52% /1h 23 min | 92.71% /38 min | 90% /1h 24min | 94.4% /34min | 88.81%/35min | 69.97% /4h 16min | 64.72% 6h 47min | 64.79% 4h 30min | 64.74% 6h 31min |
| FReLU + NoL | 70.58% /1h 28 min | 93.02% /45 min | 87% /1h 26min | 96.9% /38min | 96.63%/35min | 74.26% /4h 20min | 64.72% 6h 52min | 64.78% 4h 31min | 64.74% 6h 33min |
| FReLU + L1 | 70.58% /2h | 95.04% /53 min | 97% /1h 24min | 96.9% /43min | 96.56%/36min | 74.29% 4h 26min | 64.72% 6h 54min | 64.78% 4h 30min | 64.74% 6h 36min |
| FReLU + L2 | 70.58% /1h 26 min | 92.38% /49 min | 86% /1h 26min | 90.7% /39min | 93.28%/39min | 72.69% /4h 20min | 64.72% 6h 54min | 64.79% 4h 34min | 64.74% 6h 33min |

equal accuracy to SET, but it does this with almost 30% of its training time.

In Figures 4.11, and 4.12, the accuracy of the baseline method (SET), our best method (SF2SFrand) and the new average-weighted-SF2SFrand(AVG-weighted) algorithm is illustrated, using both ReLU and FReLU activation function. The results (in Tables 4.12, and 4.13) shown that there is no crucial difference between these algorithms regarding accuracy but there is so, in the time needed for their training. For example, picking AVG-SF2SFrand algorithm in case of using ReLU activation function and L2 regularization technique, the accuracy is about 0.18% higher than the one achieved by the old implementation of SF2SFrand. However, the time needed for the new implementation of the algorithm to converge is 80.4% larger. Generally,

Table 4.13: Comparison between the old and new implementation of all algorithms.

| **Comparison between the old and the new AVG Experiment on Lung dataset** | | |
|---|---|---|
| Algorithms | Old Experiment | New AVG Experiment |
| Dataset<br>Lung.mat | Activ.Fun.+Regul.<br>$Acc/Time$ | Activ.Fun.+Regul.<br>$Acc/Time$ |
| **SF2SFrand** | FReLU + L2<br>92.9% /16 min | ReLU+L2<br>93.07%/1h 22min |
| SF2SFba | FReLU + NoL<br>95.6% /32 min | FReLU+NoL<br>95.5%/38min |
| SF2SF(5) | FReLU + NoL<br>92.8% /35 min | FReLU+NoL<br>92.63%/35min |
| SF2SW(p=0.02) | ReLU + L1<br>74.8% /4h 17 min | FReLU+L1<br>74.29%/4h 26min |
| SF2SW(p=0.075) | ReLU + NoL<br>64.7% /6h 3 min | ReLU+L2<br>64.72%/6h 47min |
| SW2SW(p=0.02) | FReLU + NoL<br>81% /4h 13 min | ReLU+L2<br>64.79%/4h 30min |
| SW2SW(p=0.075) | FReLU + L1<br>65% /5h 50 min | ReLU+NoL<br>64.74%/6h 26min |

Table 4.14: Measures of $\zeta$-parametarized SF2SFrand.

| **Measures - $\zeta$-parametarized SF2SFrand for 40 epochs** | | | |
|---|---|---|---|
| **Lung dataset** | | **Fashion-Mnist dataset** | |
| Activ.Fun.+Regul. | $Acc/Time$ | Activ.Fun.+Regul. | $Acc/Time$ |
| ReLU+L1 | 91.17%/25min | ReLU+L1 | 88.55%/4h 33min |
| ReLU+L2 | 92.64%26min | ReLU+L2 | 83.47%/3h 35min |
| ReLU+NoL | 95.58%/25min | ReLU+NoL | 88.05% 1d 1h 55min |
| FReLU+L1 | 96.05%/27min | FReLU+L1 | 87.28%/9h 7min |
| FReLU+L2 | 96.07%/28min | FReLU+L2 | 10%/1d 5h 31min |
| FReLU+NoL | 96.05%/33min | FReLU+NoL | 10%/3h 33 min |

Table 4.15: Measures of MLP(FC), SET, SF2SFrand, SF2SFba, SF2SF(5) algorithms, respectively, in Fashion-Mnist (see Table 4.1) dataset.

| Measures - Fashion-Mnist dataset | | | | | |
|---|---|---|---|---|---|
| Datasets-<br>Activ.Fun.+Regul. | MLP (FC)<br>*Acc/Time* | SET<br>*Acc/Time* | **SF2SFrand**<br>Activ.Fun.+Regul. | SF2SFba<br>*Acc/Time* | SF2SF(5)<br>*Acc/Time* |
| ReLU + NoL | 10%/5d 1h 14min | 83.74%/11h 22min | 87.86%/ 9h 2min | 86.60%/ 21h 40min | 87.74% /5h 4min |
| ReLU + L1 | 10%/2d 16h 51min | 79.63%/15h 23min | 80.65% /12h 56min | 87.8% / 13h 7min | 88.22%/ 7h 46min |
| ReLU + L2 | 49.94%/2d 13h 5min | 83% /12h 50min | 82.68% 18h 49min | 80.81% /10h 56min | 81.65% /5h 54min |

in most of the cases, the accuracy of the AVG-weighted algorithms and the proposed algorithms is approximately the same, while the increase of the training time, needed, when an AVG-weighted algorithm achieves better performance, fluctuates between 18% to 80%. So, taking into consideration that our algorithms must be capable of running on devices with memory constraints, it is imperative for the algorithm to process the training phase as fast as possible. Therefore, our proposed implementations perform better, being applied on resource-constrained devices.

In Figures 4.13, and 4.14 the results regarding the accuracy of SET, SF2SFrand and $\zeta$-parametarized SF2SFrand on Lung and Fashion-Mnist dataset,respectively, are depicted. In this experiment, we initialize $\zeta$ parameter to 40% and then, it declines during the training phase, while in the previous algorithms we put $\zeta$ value to 30%. We can conclude that $\zeta$-parametarized SF2SFrand achieved approximately the same or a lower level of accuracy. Also,the last one seems to start stabilizing its accuracy evolution faster than the other two algorithms, when a smaller dataset is used(lung), still, the training time needed for the algorithm to converge is twice as long as the SF2SFrand algorithm(as it is shown in table 4.14), since the number of links that the algorithm has to process is much larger. This happens because the number of pruned links is not stable, but it decreases epoch after epoch exponentially. Therefore, the network tends to become less sparse and more similar to the dense network epoch after epoch, since after the 130th epoch, the number of links, being pruned,is very small and this value tends to be stabilized since, $\zeta$ fluctuates among 10% and 5%, as it is illustrated in Figure 4.15. So, our proposed method - SF2SFrand is also the winner since it achieves comparable accuracy(,even comparing it with the case that uses small values of $\zeta$) regarding the other methods, mentioned. At the same time, it reduces, concurrently, the memory footprint of the network, since it cuts more links than the $\zeta$-parametarized SF2SFrand does.

Furthermore, testing our five proposed algorithms on Fashion-Mnist dataset (which is a large scale dataset) for 100 epochs and ReLU activation function, as it is illustrated in Figure 4.16, we can confirm that all our proposed methods retain high accuracy regarding not only the fully-connected model but also the baseline model - SET. The most interesting thing here is that our best method - SF2SFrand, continues to generalize the given data faster than the SET implementation does, while in some cases, achieves about 4.9% higher accuracy than SET. For example, when ReLU activation function and L1 regularization technique are used, the

MLP(FC) network needs extremely huge time - about 2 days 16 hours and 51 minutes to be trained, the SET algorithm needs 11 hours and 22 minutes to converge, while our winning method needs 9 hours and 2 minutes to converge, too. We can conclude, that our method achieves approximatelly 20.5% faster convergence than SET does,in most of the cases, as it is shown in Figures 4.19, and 4.20 and in Table 4.15, in detail. Due to the extended training time needed from the algorithms for their convergence, we test our algorithms with the fastest activation function,having at our disposal, the ReLU.

#### 4.1.3.3 Summary of experimental results

To sum up, in Table 4.16, we demonstrate the overall results of our best sparse model training techniques in comparison with both the fully-connected model and the model constructed by SET, which is the only prior work close to ours. We start with a sparse network implemented with a randomly initialized sparse table, and we continue to cut and reconnect links of the network, during the training phase, in order for the model to converge. We see that SF2SFrand is our winning method because it achieves to compress the network capacity by reducing its trainable parameters and consequently, the memory requirements of the network up to 76% in comparison with the memory that the fully connected network needs, and up to 1% regarding the memory that our competitor (SET implementation) requires.

Furthermore, we manage to reduce the training time up to 50% in most of the cases, while we almost preserve the training accuracy of our competitors (for example, SF2SFrand achieves about up to 92.9% training accuracy in 16 minutes, while the dense model achieves about 99.5% accuracy in 1 hour and 23 minutes and the SET implementation achieves 92.71% in 38 minutes when Lung dataset is used), as it is also illustrated in Table 4.16. It is worth highlighting that our proposed schemes try to sparsify the network by deleting the connections that do not have substantial weights to contribute to the data generalization; concurrently, it reconnects as many connections as the ones being deleted, in a carefully designed way, creating sale free-like topology schemes, so as for dataset information to be distributed effectively and hence, achieve outstanding performance, regarding not only the training time but also the accuracy and the memory footprint.

Table 4.16: Summary of experiments.

| Summary of experiments | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Algorithm** | **Memory Footprint** | **Percentage Reduction in Memory** | Lung Acc/Time | Lung Discrete Acc/Time | TOX-171 Acc/Time | CLL-SUB-111 Acc/Time | COIL20 Acc/Time | Fashion-Mnist Acc/Time |
| MLP (FC) | 77.64 MB | — | 99.52% /1h 23min | 80% /5min | 24.56% /75min | 19.07% /1h 5min | 62.5% /6h 3min | 49.94%/2d 13h 5min |
| SET | 19.18 MB | 75.2% | 92.71% /38 min | 82.59% /11 min | 88.43% /32 min | 65.84% /45 min | 99.37% /1h 28 min | 83.74%/11h 22min |
| SF2SFrand | 18.39 MB | 76.31% | 92.9% /16 min | 82.5% /7 min | 83.3% /14 min | 62.4% /19 min | 99.16% /1h 42 min | 87.86%/9h 2min |
| SF2SFba | 20.68 MB | 73.36% | 95.6% /32 min | 82.4% /15 min | 95% /42 min | 65.5% /1h | 99.16% /53 min | 87.8%/13h 7min |
| SF2SF(5) | 20.18 MB | 74.00 % | 92.8% /35 min | 90.6% /14 min | 81% /40 min | 70.25% /1h 13min | 99.79% /1h 13min | 88.22%7h 46min |

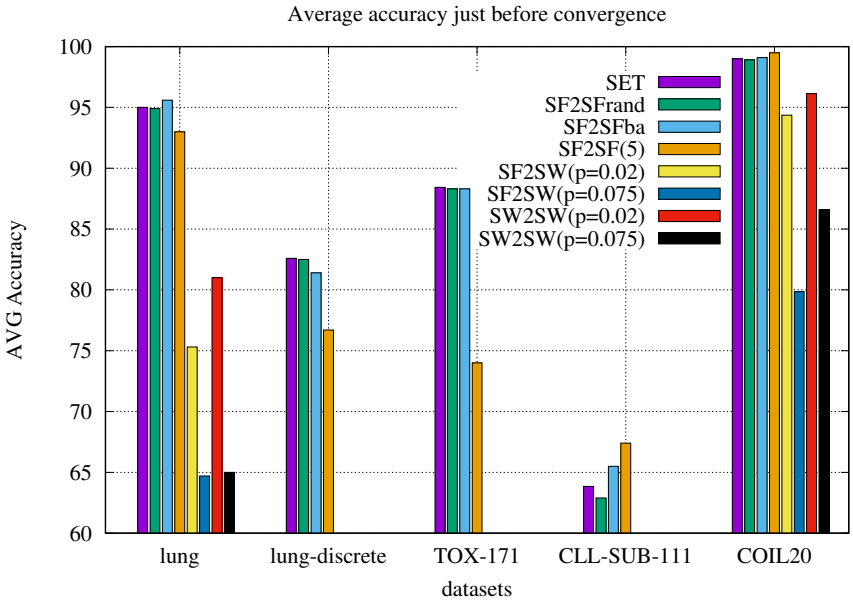



Figure 4.17: Accuracy achieved by the competitors for the five datasets.

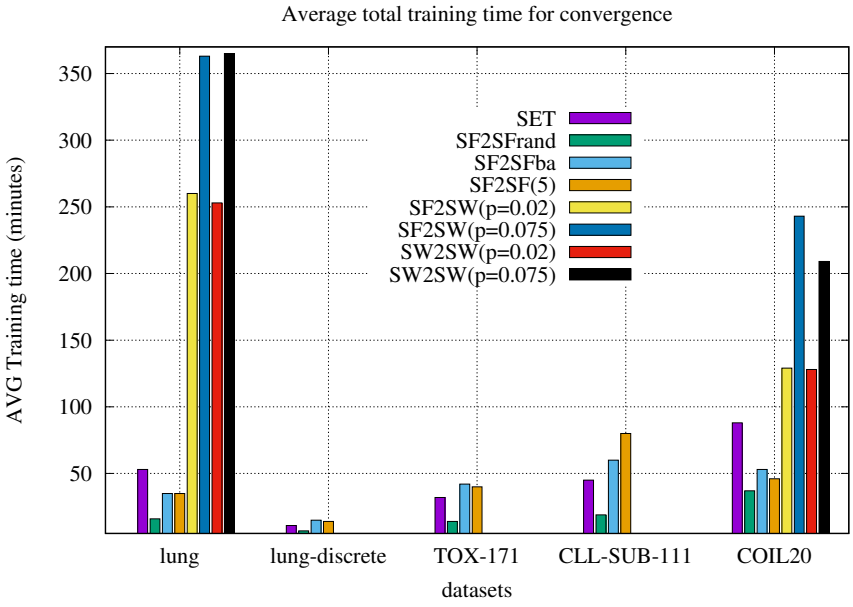



Figure 4.18: Execution time of the competitors for the five smaller datasets.

### 4.1.4 Experiments using dynamic connection removal protocols

There was further, examination, regarding the case where the percentage of connections, both pruned and restored, is not stable. In the previous experiments, we use a constant zeta parameter equal to 0.3 or 30% that keeps the evolution rate unchanged throughout training. The weight evolution does not differentiate between first and last epochs, even thought it obvious that a

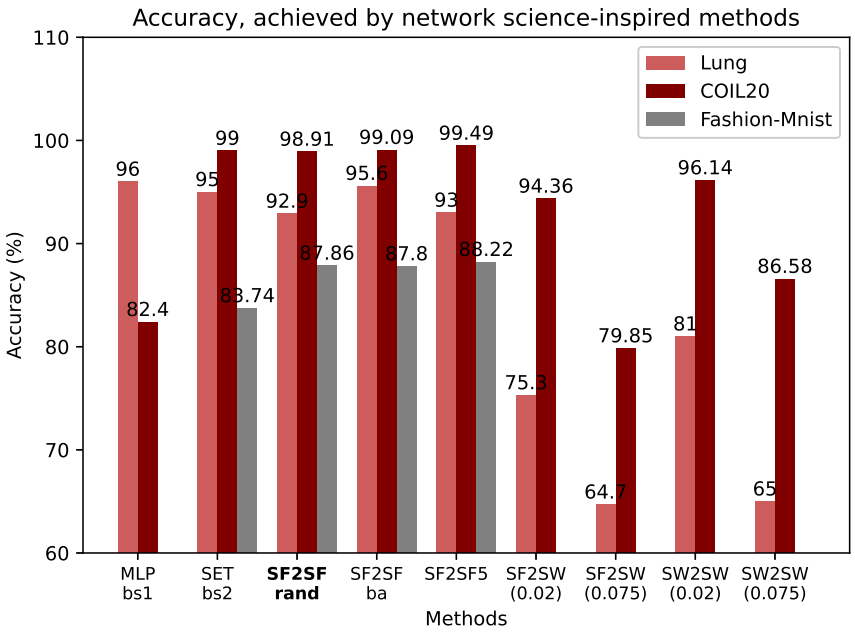



Figure 4.19: Accuracy achieved by the comparable algorithms, using Fashion-Mnist, Lung and COIL20 datasets.

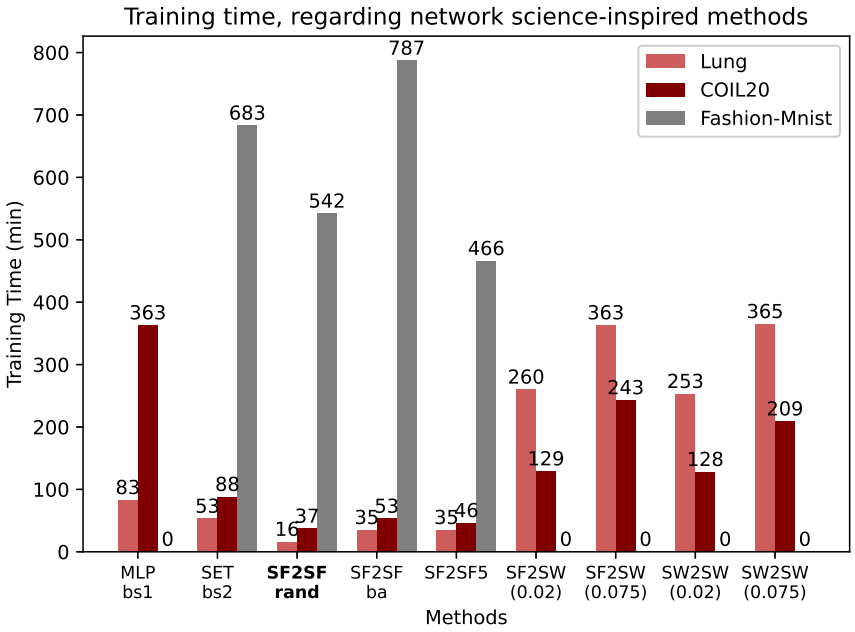



Figure 4.20: Execution time of the comparable algorithms, using Fashion-Mnist, Lung and COIL20 datasets.

neural network's behavior and stability changes as it converges towards the minimum over the training. Drawing inspiration from biology, where the synapse remodeling rate in biological

brains changes throughout an animal's life, we tried making this weight evolution rate change during training.

The methods we proposed cover two main categories. Firstly, by making the hyperparameter zeta follow a genuinely declining function, of a linear one and a much steeper exponential one, we simulate the aging factor in biological brains that may help the neural network in our case 'mature' in a much smoother rate. Secondly, causing the parameter zeta to oscillate, we display a seasonal pattern that tries to simulates the seasonal changes in an animal's brain that occur during a chronological year (hibernation, brumation etc.).

The truth is, that a lot of functions can fall into these categories but the methods we implemented for our experiments, were designed to introduce as few new hyperparameters as possible but with decent performance. The idea of making the parameter zeta a variable also draws inspiration from variable learning rate techniques, a concept that has already existed for many years. Our implementation was based on the original author's Keras implementation but that one was implemented in a way that caused the sparse methods to run a lot slower than their dense MLP counterpart. A year or two after the original paper's release, Keras introduced the Callbacks API and using it, we constructed an implementation that is more efficient and closer to the desired concept.

#### 4.1.4.1 Exponential Decay

Drawing inspiration from the field of finance we used a simple exponential decreasing function that makes the parameter zeta decay each epoch by a constant fraction called interest, as depicted in the following equation:

$$\zeta_i = \zeta_0(1 - interest)^{curr\_iter}$$

where $\zeta_0$ is the starting zeta value equal to 0.3 just as the SET procedure and the interest was chosen to be equal to 0.01 for our experiments after trying several values, so that the decay in the zeta parameter was neither too steep nor too gentle that it didn't approach the zero for the number of epochs we used. We call this method Exponential Decay (EXD) and the relevant pseudocode is depicted in Algorithm 9.

#### 4.1.4.2 Linear Decreasing Variation

In a similar fashion to the variable learning rate methods used by Abbas et al. [181] we applied a linear decreasing curvature to the parameter zeta that is adjusted to the maximum number of epochs. We call this method Linear decreasing Variation (LDV), with the relevant pseudocode in Algorithm 9. The following equation depicts this procedure:

$$\zeta_i = \zeta_{min} + (\zeta_{max} - \zeta_{min}) \times \frac{max\_iter - curr\_iter}{max\_iter}$$

We set a maximum zeta value of 0.3, just as the SET procedure, and a minimum one of 0.01 so that it starts with the maximum zeta value at the start of the training and linearly decreases to the minimum zeta value over the entire training process. We wanted the minimum to be close to zero but not equal so the procedure won't shut down completely by the end.

#### 4.1.4.3 Oscillating Variation

Here, once again following the steps of Abbas et al. [181] we applied an oscillating variation curve to the zeta parameter, adjusted to the maximum number of epochs. We call this method Oscillating Variation (OSV). Just like the LDV method we need a minimum and maximum zeta value that is defined as 0.01 and 0.3 respectively. The following equation shows how this is done:

$$\zeta_i = \frac{\zeta_{max} + \zeta_{min}}{2} + \frac{\zeta_{max} - \zeta_{min}}{2} \times cos\left(\frac{2\pi \cdot curr\_iter}{T}\right) \qquad \text{where} \quad T = \frac{2 \cdot max\_iter}{3 + 2k}$$

In the definition of the period $T$ we use the parameter $k$ to control the frequency of the oscillations and was set equal to 1 for our experiments after performing a small random search. The relevant pseudocode can be seen in Algorithm 9.

**Algorithm 9** Pseudocode of the Proposed Techniques

**Procedure** *Sparse_ Training* (Sparsity level $\varepsilon$, $\zeta_{max}$, $\zeta_{min}$, Frequency $\kappa$)
Initialize Fully Connected Neural Network model;
Sparsify_network($\varepsilon$);
Initialize training algorithm parameters;
**for** $i$ in range(1,epochs) **do**
    Feed_Forward();
    Back_Propagation();
    **if** method == EXD **then**
        Apply_EXD_rule( $\zeta_0$, interest, i, epochs);  // *See EXD equation*
    **else if** method == LDV **then**
        Apply_rule( $\zeta_{max}$, $\zeta_{min}$, i, epochs);  // *See LDV equation*
    **else if** method == OSV **then**
        Apply_rule( $\zeta_{max}$, $\zeta_{min}$, $\kappa$, i, epochs);  // *See OSV equation*
    **end if**
    Weight_evolution($\zeta$);
**end for**

### 4.1.5 Experimental Evaluation using dynamic connection removal protocols

We evaluate our methods using the following metrics: a) memory footprint, b) Accuracy on the test dataset and c) training time. The training time metric depicts the total elapsed time it needs to train a model. We conducted three experiments for each method and the results are calculated by their mean values.

#### 4.1.5.1 Evaluation Settings using dynamic connection removal protocols

For our experiments, we used the Fashion MNIST dataset, one of datasets also used by Mocanu et al. One of the main reasons for this decision is that we can have a common ground for comparing our methods to the SET procedure and the Dense MLP networks, making our results more understandable. The Fashion MNIST is a dataset of 60,000 training examples and 10,000 test examples of 28 by 28 gray scale images, depicting clothes that belong into 10 classes. We also use data augmentation techniques to make the models generalize better and show greater accuracy.

As far as the models are concerned we chose three neural network setups. We use the Multi Layer Perceptron for all topologies in our experiments. The first topology has three hidden layers with 1000 neurons in each one (called MLP-1K). The second one is closely related to the experiments of Mocanu et al. and uses three hidden layers with 4000 neurons on the first, 1000 neurons on second and 4000 neurons on third (called MLP-4K). The final topology uses four hidden layers with with 4000 neurons on the first, 2000 neurons on second and third layer and 1000 neurons on fourth (called MLP-4K4L). Although, deeper and larger DNN architectures are the most common approaches, they may incur substantial redundancy [182], which may lead in heavy computational cost, without any special contribution, regarding accuracy. After a lot of experiments, we decided to use the aforementioned topologies to cover a wide range of topology characteristics that will help outline differences in the behavior of the tested methods. We train the models for 500 epochs using the default values for most of the hyper parameters. The experiments were executed on a desktop computer with 16 GB of RAM and a Quad-Core Intel Core i7 processor clocked at 4.0 GHz. For the sparse models (SET, LDV, EXD, OSV) we assign the sparsity level $\varepsilon$ equal to 20. The value zeta stays at 30% for the SET procedure, as this was the proposed value from its creators, and we use this same value as well for the initialization of zeta on our methods. The parameters' initialization values that we used for each method, are reported along with the methods they belong to at subsection 4.1.4.

#### 4.1.5.2 Evaluation Results using dynamic connection removal protocols

The fist benefit of our methods is their reduced memory footprint especially compared to their dense MLP counterpart. Both the SET procedure and our methods use the same Erdős–Rényi random graph procedure to make their networks sparse at the beginning of the algorithm. As seen in Table 4.17 we achieve at least 95,7% reduction in parameters compared to the dense topology. This creates compression rates from $\times 23$ to $\times 45$ depending on the topology. Due to the nature of both SET and our methods, we know the exact model size before training, because our methods belong to the sparse training category of model pruning, which means that even though the model's parameter's are pruned and reconnected throughout the whole training process, the total amount of parameters at the end of each epoch stays the same. This is very beneficial feature, as we can tune the parameter $\varepsilon$ before training to achieve the desired

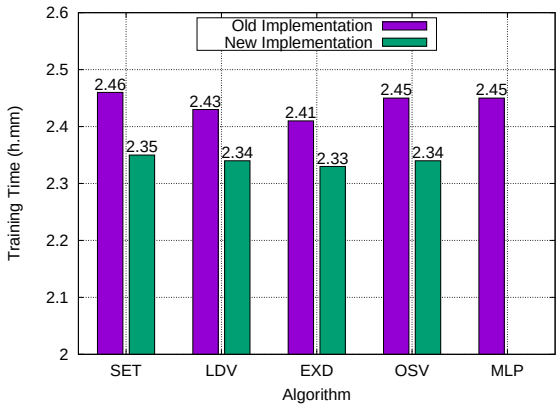
Table 4.17: Comparing the parameter reduction between the Dense MLP and the Sparse methods.

| Hidden Layer Architecture | Dense Size | Sparse Size | Compression rate | Percentage Reduction |
|---|---|---|---|---|
| MLP 1000-1000-1000 (MLP-1K) | 2.797.010 | ≈ 120.000 | ×23 | 95,7% |
| MLP 4000-1000-4000 (MLP-4K) | 11.185.010 | ≈ 350.000 | ×32 | 96,8% |
| MLP 4000-2000-2000-1000 (MLP-4K4L) | 17.155.010 | ≈ 380.000 | ×45 | 97,7% |

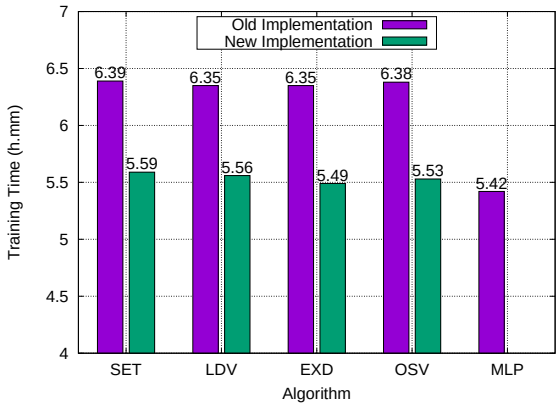
compression rate, which helps on models that are designed to run in IoT devices with small memory capacities.

The second part of evaluating our methods is the training time. As illustrated in Figure 4.21 the speed improvement comes in two ways. Firstly, when comparing the dense MLP to the sparse models we see same training time costs in the MLP-1K topology, but the sparse models become progressively slower in comparison as the model grows in size MLP-4K and MLP-4K4L (as it is depicted in Figure 4.21 - from up to bottom, respectively), even though their memory footprints are much better than the dense MLP. Using Keras Callbacks these differences are now greatly reduced as the weight evolution is now performed more smoothly. We would expect that the sparse models should be faster than the dense, considering the compression rate achieved, but as Hoefler et al. [59] pointed out this in not always the case when using such libraries. The reason is that, most modern libraries are not designed to use sparse matrices and as a result they don't always exploit their reduced size efficiently. Another speed improvement that is evident from the same Figures is how our methods compare to SET and the dense MLP network. Our methods are consistently slightly faster on average than the SET procedure with the Exponential Decay (EXD) method being the winner among the sparse methods when also taking the improved implementation into consideration. We attribute this behavior due to the reduced weight pruning and reconnecting that happens as a direct result of the parameter zeta reduction.

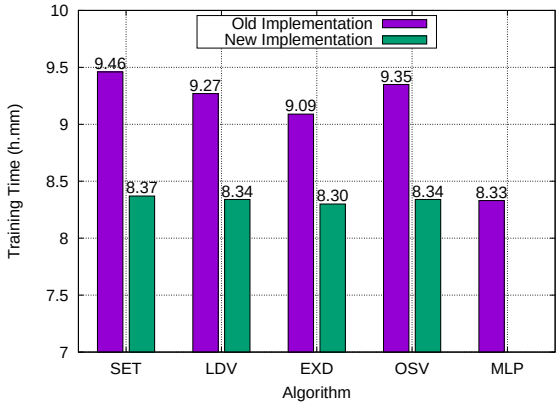
The final part of the evaluation involves accuracy, a metric usually considered the most important for the model's performance. As we can observe from the Figure 4.22 our methods remain competent to the SET and dense MLP models. We can see that the EXD method outperforms the rest, for the topologies MLP-4K and MLP-4K4L. In the smaller MLP-1K topology though we can see that the reduced number of total parameters in the network has a toll on the performance of all the sparse methods. We can therefore safely declare the EXD method as a clear winner in accuracy as well among the sparse methods. Our other two methods although not as good as the EXD, we can see that they still manage to outperform the SET method by a margin depending to the topology used. We can observe that all our methods and especially the EXD method have the potential to outperform the dense MLP when using




(a) MLP 1000-1000-1000 (MLP-1K)

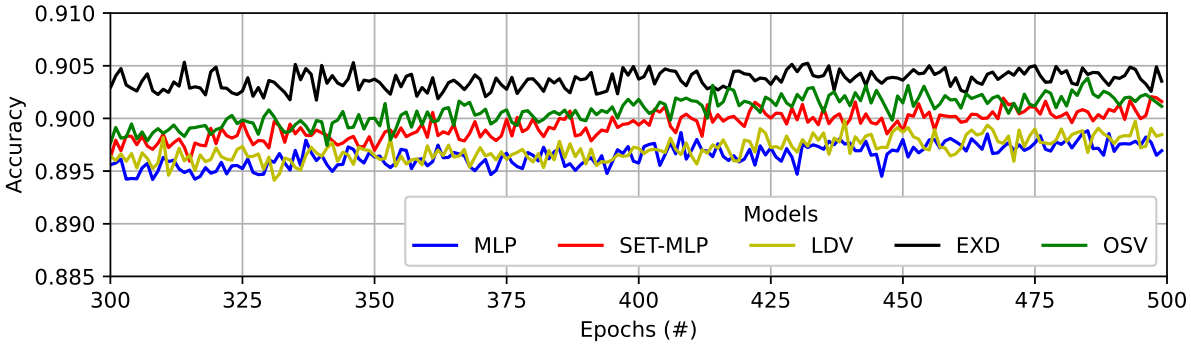



(b) MLP 4000-1000-4000 (MLP-4K)

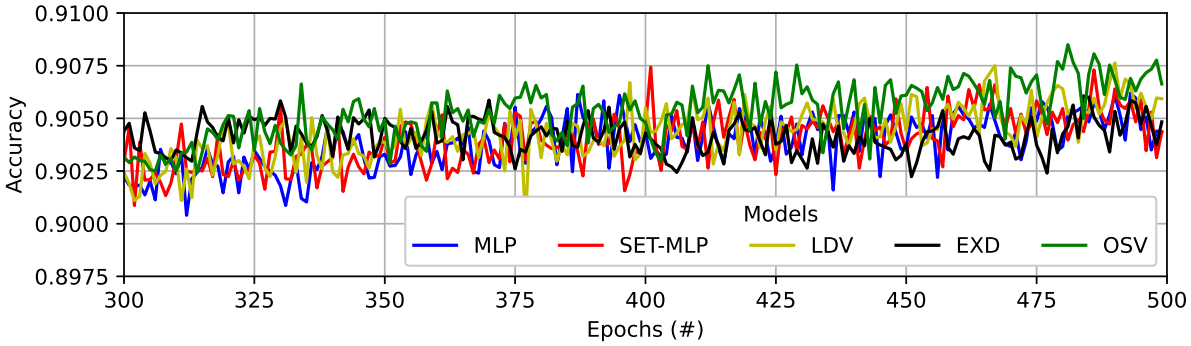



(c) MLP 4000-2000-2000-1000 (MLP-4K4L)

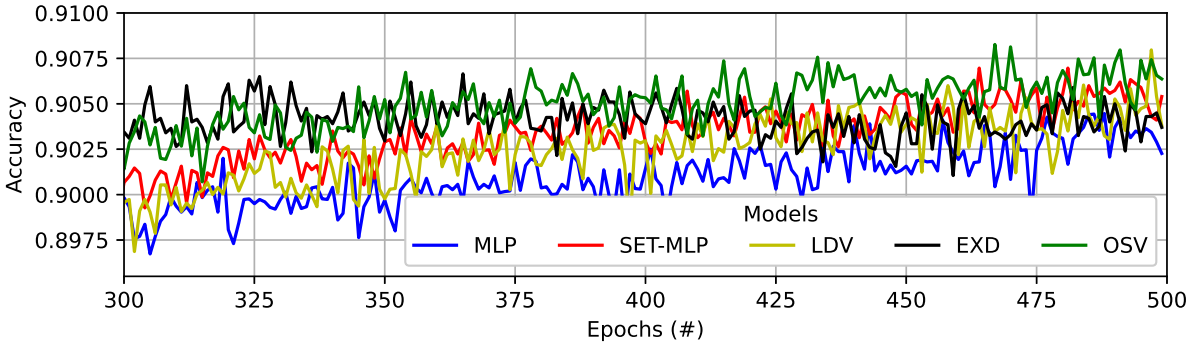
Figure 4.21: Competitors' training time between the old and the improved Keras implementation based on the network architecture used.




(a) MLP 1000-1000-1000(MLP-1K)




(b) MLP 4000-1000-4000 (MLP-4K)




(c) MLP 4000-2000-2000-1000 (MLP-4K4L)

Figure 4.22: Competitors' validation accuracy based on the network architecture used.

sufficiently large topologies.

We should keep in mind however that the dataset used belong into the image classification category and is only one kind of problems that neural networks are used for. Our methods may perform differently on other types of datasets, and this is mainly the reason that we didn't discarded the losers, so that they may provide some inspiration for other experiments or to maybe find application to other areas. But if anyone would consider trying one of our methods, the EXD method should be the default or at least the first choice.

### 4.1.6 Conclusions

The tremendous success of deep learning has brought neural networks to the forefront of machine learning research and development. Due to the large size of a neural network - in the number of neurons and the number of hidden layers - training the network in a relative short time is a challenge. Various methods have been developed for accelerating neural training over the past thirty years. We focus here on the family of methods based on linkage sparsification, i.e., instead of fully connected bipartite neural topologies, we reduce the number of connections in an algorithmic (or random) way. In particular, we employ concepts developed in the realm of network science, in order to sparsify the neural network with the aim of keeping the most significant linkages, which are responsible for better information distribution in the network. Thus, we reduce drastically both the number of trainable variables and the size of the model, which leads to training acceleration. We base our motivation on observations in real neural networks whose actual topology is scale-free (or small-world). We designed algorithms that start from a particular structured but not fully connected bipartite topology and end up with another structured topology. Here, in this first investigation, we experimented with scale-free and small-world topologies either as starting or final topologies. We evaluated the algorithms' performance on a moderate-sized neural network in a publicly available dataset and examined their classification accuracy and training time. We concluded that the proposed techniques can reap performance gains, achieving high accuracy with a short training time. The 'champion' algorithm was the one that produced scale-free-like topologies starting from scale-free topologies. Specifically, the SF2SFrand algorithm outperforms every other method in most of the cases mentioned, achieving high percentages of both memory (approximately 76.31% less than FC-MLP dense model and 3% less than the baseline method - SET) and training time (about 50% less in most of the cases, compared to baselines(FC-MLP, SET) and 20.5% less in the experiments with large scale datasets, like Fashion-Mnist) reduction and, while retaining classification accuracy of baseline methods. Intuitively this is expected since only a handful of connections carry most of the weight even in fully connected topologies. Our results are consistent with recent but different types of approaches [40, 41] to the problem of neural training acceleration.

There was further examination, regarding the case where the percentage of connections both pruned and restored, is not stable but varies, (drawing inspiration from the ideas, presented in [181]), linearly (Linear Decreasing Variation - $LDV$ method), exponentially (Exponential Decay - $EXD$ method) or following the cosine rule (Oscillating Variation - $OSV$), at each epoch. The results shown that when the percentage of close-to-zero synapses is modified exponentially or is oscillated following a cosine rule function, the accuracy of the model remains approximately 90%, and the time needed is also in the same levels as the baseline methods, since the technique, used, demands enough calculations, even if the number of connection, removed is larger. As far as memory footprint is concerned, we confirm the dominance of the sparse methods as opposed to the dense MLP by achieving at least 95.7% reduction in memory footprint, indicating

them as a potential great fit for small IoT devices. Furthermore, the methods we propose also seem to display some benefits over the main baseline procedure in training time while staying competitive in accuracy. Among them, the Exponential Decay method (EXD) really stands out, surpassing the other methods on almost every experimental scenario. The EXD method is around 8-13% faster in training speed and constantly slightly more accurate than the SET procedure. We chose to keep a simple and elegant approach when designing our methods, and we hope this work will inspire further efforts.[1]

[1]GitHub Code: `https://github.com/achouliaras/sparse_ann_training_dynamic_pruning`

## 4.2 Transfer Learning

### 4.2.1 Introduction

Tiny Machine/Deep Learning *(Tiny ML/DL)* is a fast-growing field, which boosts machine learning tasks deployment in devices that lack of computational resources, like sensors, IoT devices etc. It is a useful tool since it combines both software and hardware optimization and hence it enables the running of ML inference tasks in those devices [183], etc. Moreover, TinyML, by encouraging data processing on the edge device, contributes to the maintenance of the privacy of data, since information that is collected, stays to the device, while in parallel, it lessens the energy cost of the possible and now not necessary communication between the edge device and the server[184].

Furthermore, Transfer Learning *(TL)*, is a method in which pre-knowledge of a source model is used, in order to enhance the learning ability of the device in which this model is further applied. TL means either retraining the whole model or the last fully connected layer to the resource-constrained (target) device and hence boosting its performance with both reducing the training time needed for the model to converge, as it inherits the already learned features and by giving the opportunity for the model to be trained with sufficient already-existed data. In this work, we combine the two aforementioned techniques, by proposing algorithms that go the common TL and the already known TinyML optimization techniques a step further, in order to maximize the performance of low-powered devices. Specifically, our endeavor is inspired by the work [19], in which authors propose the retraining of the fully connected layers to low-powered device. We present methods of retraining, which include also one or more convolutional layers, with the one that uses only one convolutional layer to be our best method, due to the fact that we achieve increase in accuracy levels, with the less energy consumption, regarding to the other two methods, proposed in this work. Specifically, $F_xC1$ method gives approximately 19% increase in classification accuracy of the model.

The rest of the section is structured as follows: subsection 4.2.2 describes the proposed techniques. In subsection 4.2.3, we evaluate the neural network's accuracy, loss and energy consumption, and in subsection 4.2.4 we summarize the described results. Finally, section 4.2.5 concludes this work.

### 4.2.2 The family of $F_xC_y$Techniques

The most widespread technique in Transfer Learning is to train the model to a dataset, then 'freeze' the weights and after that either retrain the last fully connected layer, or retrain the whole model to the new data. In our research, we conducted experiments in order to observe how

[0]Related publication **[C1]**:E. Fragkou, V. Lygnos, D. Katsaros, *Transfer learning for convolutional neural networks in tiny deep learning environments*, Proceedings of the 26th Pan-Hellenic Conference on Informatics (PCI22), hybrid, 2022, pp. 145—150.

different the model will perform when we retrain more layers than just the fully connected ones and compare our results with the already existed TL methods. So, we created three different experimental cases:

**F$_x$C1**. In this case, we train not only the last fully connected layers, but also the last convolution layer, while the remaining model is frozen.

**F$_x$C2**. In this case, we train the fully connected layers and the last two convolution layers, while the remaining model is frozen.

**F$_x$C3+**. In the last case, we train the fully connected layers and up to three convolution layers, while the remaining model is frozen.

In our experiments, the fully- connected layers of the networks, used, are more than one, so in order to elaborately present our Figures, we display exactly the number of both fully-connected and convolutional layers, taking place in training, by using the abbreviation $F_xC_y$, while $x$ is the number of final fully- connected layers and $y$ the number of Convolution layers, re-trained in every experiment, as it is illustrated in Table 4.19.

**Algorithm 10** Pseudocode of the family of $F_xC_y$ techniques

```
procedure PRE-TRAINING(on a large dataset)
    for i ∈ range(1, epochs) do
        if method == F_xC_1 then
            Re-train(FC, Conv.Layer-1)                          ▷ 1 conv. Layer
        else if method == F_xC_2 then
            Re-train(FC, Conv.Layer-1, Conv.Layer-2)            ▷ 2 conv. Layers
        else if method == F_xC_3+ then
            Re-train(FC, Conv.Layer-1, Conv.Layer-2, Conv.Layer-3, ...)   ▷ Up to 3 conv. Layers
        end if
    end for
end procedure
```

The experiments emphasize on the large models, in most of the cases, since it is more difficult to optimize them, without sacrificing performance. For each case, we have computed the flops needed for them to run for one epoch so we have made an estimation of how much energy they will consume, taking into account a CMOS 45nm processor.

### 4.2.3 Experimental evaluation

This section introduces details about the competitors, the datasets used, the size of the neural networks we experiment with and then it presents the results of the experimental evaluation of the proposed algorithms.

#### 4.2.3.1 Evaluation settings

**Competitors.** In order to evaluate our algorithms, we compare the performance of our method with the two following baseline methods.
*Baseline-1.* In this method, only the last fully connected layer is retrained, while the remaining model is frozen [19].
*Baseline-2.* In this method, the whole model is retrained.

**Architectures of neural networks.** We will use 3 CNN models, a small one which has an input of 32x32x1 and 550 thousand parameters with 6 convolution layers with relu as the activation function except for the last dense layer, in which we used softmax activation function, called Small CNN. The other two models are from the family of EfficientNet, which is a group of convolutional network models that acheive high accuracy, with very few parameters and hence is a viable solution for resource-constrained devices. (specifically, EfficientNetB0 and EfficientNetB2). We picked EfficientNetB0 and EfficientNetB2 as our large models with 4.05 million and 7.76 million parameters, respectively, as we see in Table 4.19. In the fully connected layers, we used softmax as the activation function.

**Measures.** For the small CNN model, we did 50 epochs for every case we tested, while for the EfficientNet models we did 10 epochs, respectively. Considering we have a classification problem to tackle, we used the 'Accuracy' measure (implemeted in Keras) in order to evaluate and compare our methods. Although we address a multi-class problem, we used *accuracy* metric instead of *F1-measure/Score*, taking into consideration that our first experiments were conducted, using the family of CIFAR datasets (see the following paragraph), whose classes are equally important, or else every class constitues by the same number of instances (balanced classification dataset), and consequently, these two metrics can equally produce safe results, in this case. We continued our experiments, using other two datasets, ImageNet and Intel classification, which don't have exactly the same number of instances in every class, but the difference between the number of data tends to be very small. The proportion of data in the first dataset is 1:7 and it is mainly used for training the model, since it contains 1,000 classes of images, which means plenty of images and therefore, better learning of the model. On the contrary, the second one (whose difference among class instances is about 2% - almost balanced) is used only for testing purposes, so in these cases, we can safely use *accuracy* metric, too. Moreover, we used both categorical cross-entropy (especially used for one-hot encoded vectors) and sparse categorical cross-entropy function (class vectors contain only the integer that stipulates their input data category - we gain processing time during training), implemented in Keras API, as the loss function for small CNN model and the EfficientNet models, respectively. The last measure we used, is the one referred to the energy, required by our methods. The most reliable way to estimate it, is through the FLOPS of every model needed for one Epoch. Therefore, we computed how many FLOPS each test we did, needs for one epoch and how much energy it would consume, being implemented to a certain processor (45nm CMOS technology processor)

Table 4.18: Dataset characteristics, used in this experiment.

| Dataset | Size of Images | Train Samples | Test Samples | Classes |
|---|---|---|---|---|
| ImageNet | 224x224 | 1, 281, 167 | 100, 000 | 1, 000 |
| CIFAR-100 | 32x32 | 50, 000 | 10, 000 | 100 |
| CIFAR-10 | 32x32 | 50, 000 | 10, 000 | 10 |
| Intel Image Classification | 150x150 | 14, 000 | 3, 000 | 6 |

with the admission that a MAC operation is roughly equal to two flops. Even if the admission is far from reality, the percentage difference between the two cases would be the same, if FLOPS remain the main comparing factor. Nano Joule (nJ) is the unit of measurement in this case. The type we used is the following:

$$ENERGY = \frac{FLOPS}{2} * Emac$$

.

**Datasets**. In this work, we deal with an image classification task. In order to test our methods, we used datasets that include both colour and greyscale images. Furthermore, datasets have a lot of instances and the same feature space (homogeneous data) to a great extend, so as to enhance the adaptation of data from the source to the target domain. The test samples we used, are 20% smaller in size compared to the training samples ragarding the CIFAR-10, CIFAR-100 and Intel classification datasets and approximately 8% smaller, compared to Imagenet dataset (due to the tremendous training time, needed in order to run our experiments).Besides, the most important thing is for the test dataset to be smaller than the training one, or else it will cause accuracy drops. Information about the datasets is illustrated in Table 4.18. Moreover, we conducted four experiments, in which we use different combination between the datasets for both pre-trained and retrained networks.

**Hardware used.** Our methods were tested on a desktop computer with 16 GB of RAM and an Intel Xeon W 2123 processor, clocked at 3.60 GHz.

#### 4.2.3.2 Evaluation results

In the following experiments, we compare the results with the two baseline methods, mentioned in section 4.1. In plots, every method is represented by the number of Layers that are unfreezed during the retraining phase of the model, so as to know exactly which architecture we use. In Table 4.19, we see the total number of layers, taking place in retraining and how many of them are convolutional ones.

**Experiment-1 (Evaluation of Small CNN with CIFAR-100 and CIFAR-10).**

In our first experiment, we used CIFAR-100 to train the model and CIFAR-10 [171] to evaluate it. As the Figure 4.23 (left) illustrates, training both the fully connected and the last

Table 4.19: Trainable Layers, used in every model architecture.

| Model | **Small CNN** | | **EfficientNetB0** | | **EfficientB2** | |
|---|---|---|---|---|---|---|
| Trainable Parameters | 550, 000 | | 4, 049, 571 | | 7, 768, 569 | |
| Method/ Trainable Layers | Total | #of Conv. | Total | #of Conv. | Total | #of Conv. |
| $F_x$C1 | 9 | 1 | 3 | 1 | 3 | 1 |
| $F_x$C2 | 11 | 2 | 5 | 2 | 7 | 2 |
| $F_x$C3+ | 15 | 3 | 8 | 4 | 10 | 4 |
| $F_x$C3+ | - | - | 16 | 5 | 15 | 5 |
| $F_x$C3+ | - | - | 20 | 6 | 23 | 8 |
| $F_x$C3+ | - | - | 23 | 8 | 31 | 9 |
| $F_x$C3+ | - | - | 31 | 10 | 38 | 12 |

convolution layer, gets almost a 10% boost in accuracy, regarding *Baseline-1* method. Hence, adding more layers to the training procedure, finally contributes to the accuracy improvement, but it is expensive regarding the energy requirements.

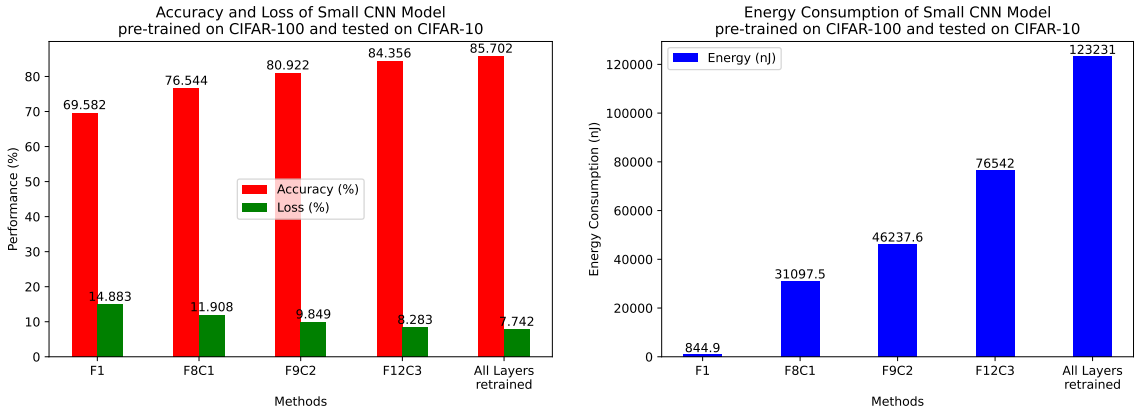



Figure 4.23: Experimental evaluation of small CNN model, pre-trained on CIFAR-100 and tested on CIFAR-10.

Regarding the energy consumption, depicted in Figure 4.23 (right), we verify our intuition, regarding low power consumption of all models being presented against the *Baseline-2* method. This does happen due to the smaller amount of trainable parameters of both the *Baseline-1* method and our proposed techniques, which makes the training procedure faster and therefore, less energy-expensive. As we can understand, the energy consumption increases proportionately with the number of layers, being unfreezed in every experiment.

**Experiment-2 (Evaluation of EfficientNetB0 with Imagenet/ CIFAR-100.**

In this test, EfficientNetB0 is used and trained with ImageNet dataset [172]. We will evaluate it with CIFAR-100 and the results are depicted in Figure 4.24. The difference in both accuracy and loss regarding the two baseline methods is significant. We can easily observe that our method, namely $F_xC1$, continues to stands out against *Baseline-1* method, while having a bigger and more structurally-complicated network, compared to Experiment-1. Training the fully connected layer and the last convolution layer we get an increase of 20% compared to the training of only the fully connected layers. Also, the accuracy of this case is slightly less than the accuracy, retained by training all layers of the model, which means that we don't have important information loss.

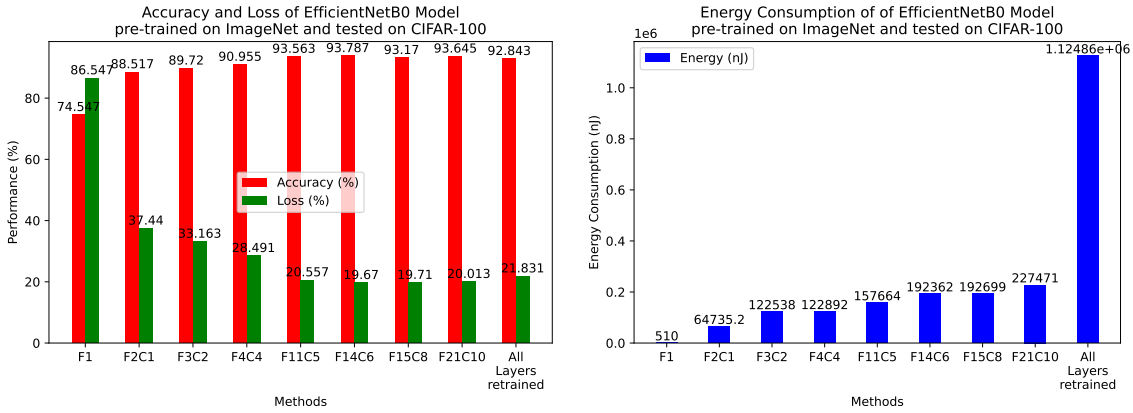



Figure 4.24: Experimental evaluation of EfficientNetB0 model, pre-trained on ImageNet and tested on CIFAR-100.

**Experiment-3 (Evaluation of EfficientNetB2 with Imagenet/ CIFAR-100.** In this experiment, we use the same order of datasets of the previous experiment (ImageNet for pre-training and CIFAR-100 for fine-tuning of the network respectively), whereas a quite larger network, the EfficientNetB2 one is used. It is of a great importance that the results are encouraging in this experiment, too. Specifically, we notice a small increase of 18.87% in accuracy, as it is shown in Figure 4.25 (left), while the amount of the trainable parameters of the network is extremely large, compared to both the small CNN and EfficientNetB0. Moreover, it is worth highlighting that our method $F_xC1$ generalizes the given data faster (about $60,81\%$ faster convergence), compared to *Baseline-1*, in the same number of epochs. Furthermore, the more the unfreezed layers are, the greater the improvement of accuracy measure is, bearing in mind the results in Figure 4.25. Regarding the energy consumption, we can conclude that the less the unfreezed layers are, the less the energy consumption of every model, being tested, is but prioritizing the measure of accuracy, we can conclude that $F_xC1$ method outperforms any other method mentioned, since it offers better accuracy, regarding the common baseline-1 method, while it increases the energy requirements of the model, to a small extend, comparing with the other proposed methods.

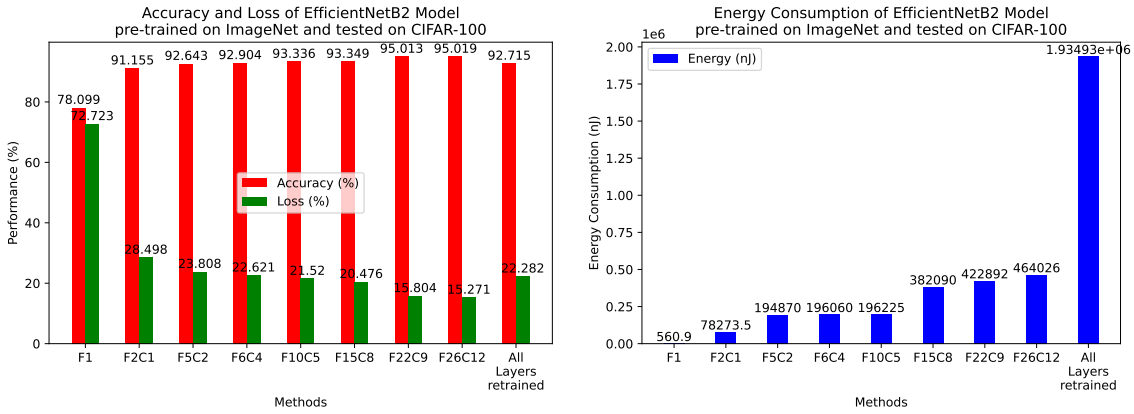



Figure 4.25: Experimental evaluation of EfficientNetB2 model, pre-trained on ImageNet and tested on CIFAR-100.

**Experiment-4 (Evaluation of EfficientNetB0 with Imagenet/Intel Classification Dataset).** In the last experiment, we use EfficientNetB0 network, being pre-trained on ImageNet dataset. In this test, we use a smaller dataset to evaluate it, which is the Intel classification dataset [173]. As we expected, the pattern in the bar graph, being illustrated in Figure 4.26 is similar to all the experiments, presented so far(experiment-1-2-3). Furthermore, because of the smaller size of the dataset, used, we achieve high accuracy, faster than the *Baseline-1* method, despite the handred of parameters used in this network. Specifically, in this experiment, $F_xC_1$ method achieves about $6,09\%$ boost in accuracy and approximately, $39,38\%$ faster model convergence.

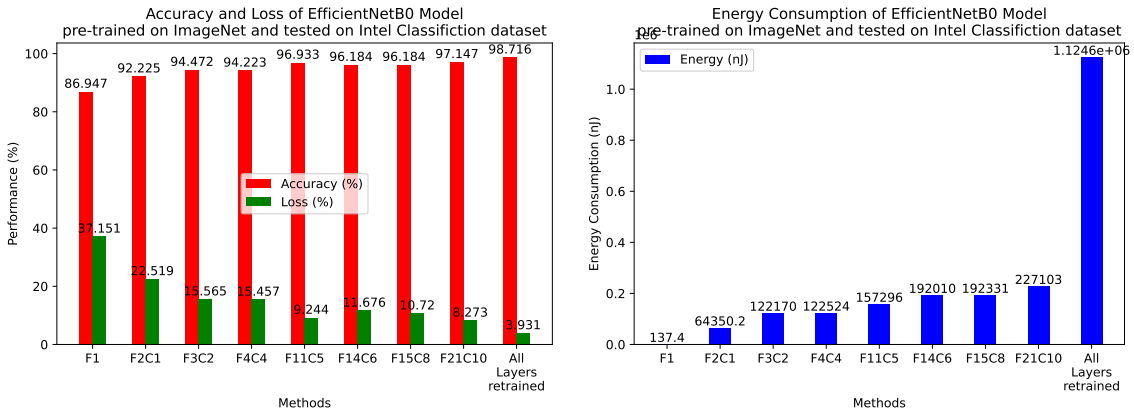



Figure 4.26: Experimental evaluation of EfficientNetB0 model, pre-trained on ImageNet and tested on Intel classification dataset.

### 4.2.4 Summary of experiments

To sum up, in Table 4.20, we compare the performance of the classic TL method (*Baseline-1*) and the one ($F_xC1$) we end up proposing, after conducting our experiments. We compare only the *Baseline-1* model since the *Baseline-2* consists of the whole model and therefore it is not a viable solution for a TinyML device, due to extravagant energy consumption (up to $143x$ larger than the one needed by the *Baseline-1* method). We conducted several experiments, using three types of neural networks (see Table 4.19) and four large-scale datasets (see Table 4.18) in a pair of two (for pre-training or re-training). In every test $F_xC1$ method outperforms (regarding accuracy) the *Baseline-1* method in a range from $10,7\%$ to $18,79\%$, proportionately to both the datasets and the model, used, while it also achieves faster convergence than *Baseline-1* method, in a range from $19,98\% to 60,81\%$, as we can see in Figures 4.23, 4.24, 4.25, 4.26. However, the energy consumption required is a tradeoff since it is a bit larger than the one, needed for the *Baseline-1* method (see Table 4.20). The rest of the methods proposed, even if they had even better results regarding accuracy, they can't run on resource-starving devices due to the large energy consumption they require, which increases proportionately to the layers that are retrained in each case.

Table 4.20: Summary of experiments.

| **Model** | Accuracy (%) | | Energy (nJ) | | Percentage Increase in Accuracy, compared | Percentage of faster convergence, compared |
|---|---|---|---|---|---|---|
| Datasets Sets (pre-trained/tested) | FC (*Baseline-1*) | $\mathbf{F_xC1}$ | FC (*Baseline-1*) | $\mathbf{F_xC1}$ | to *Baseline-1* (%) | to *Baseline-1* (%) |
| **Small CNN** (CIFAR-100/CIFAR-10) | 69,5 | 76,5 | 844,9 | 31097,5 | 10,01 | 19,98 |
| **EfficientNetB0** (ImageNet/CIFAR-100) | 74,5 | 88,5 | 510,0 | 64735,2 | 18,79 | 56,74 |
| **EfficientNetB0** (ImageNet/Intel Classification) | 86,9 | 92,2 | 137,4 | 64350,2 | 6,09 | 39,38 |
| **EfficientNetB2** (ImageNet/CIFAR-100) | 78,0 | 91,1 | 560,9 | 78273,5 | 16,79 | 60,81 |

### 4.2.5 Conclusions

Overall, we presented new Transfer Learning methods in order to enhance the performance of a network, being deployed to resource-constrained environments. We present a simple but efficient method, in which we don't fine-tune only the last fully connected layer of a pre-trained network, but we also retrain one or more of the convolutional layers of three kinds of

a CNN network (a Small CNN, EficientNetB0, EficientNetB2). The results shown that $F_xC1$ proposed method acheives approximately 19% increase in network accuracy and about 61% faster convergence, whereas it is a model-aware method since it outperforms the classic Transfer Learning techniques in all cases, tested, with almost a small increase in energy consumption, compared to the *Baseline-1* method (retraining the last fully connected layer) and incomparably better performance, regarding the energy consumption, needed by *Baseline-2* (retraining the whole network) method.

Chapter 5

# Distributed Learning

## 5.1 Distributed Federated Deep Learning in Clustered IoT Wireless Networks with Data Similarity-based Client Participation

### 5.1.1 Introduction

Federated (Deep) Learning – F(D)L [10] emerged in order to cope with data privacy/security issues that occurred in the traditional cloud-based (deep) learning. In the basic form of FDL, a set of nodes each performs rounds of local training of their deep learning model, and then sends the resulting weights, i.e., their model to a server that performs aggregation and sends back the averaged weights; these weights are then adopted by the nodes, which iterate again the same procedure until convergence. Still, FDL fails short for servicing a lot of modern applications running over the Internet of Things (IoT) [185]. Such applications are based on a large number of distributedly and ad hoc interconnected devices, which generate private data that cannot be sent wirelessly over the network to the server due to privacy and network capacity issues, and moreover their lack of resources does not allow them to perform massive wireless transmission of their local trained models for many obvious reasons; e.g., nodes residing on the paths leading to the server will rapidly deplete their energy and die out due to the excessive communication. To be adjusted to the wireless ad hoc network environments, hybrid FDL schemes with device-to-device communication and a coordinating server have been proposed [23], as well as schemes based on only device-to-device communication [186]. However, neither

[0]Related Publication **[J1]**:E. Fragkou, E. Chini, M. Papadopoulou, D. Papakostas, D. Katsaros, S. Dustdar, *Distributed federated deep learning in clustered internet of things wireless networks with data similarity-based client participation*, IEEE Internet Computing magazine, 2024, vol.28 (6), 2024, pp. 53-61.

of the proposed approaches as explained in Chapter 2 (see Section 2.3.2) can cope with the grand challenge on developing federated deep learning schemes for ad hoc IoT networks, namely the purely distributed nature of the architecture and communication efficiency to alleviate the broadcast storm problem [132] in resource starving environments. Even, some proposals that assumed network backbone construction based on clustering [187] which is classic technique to cope with broadcast storms, fail to fully address the FDL requirements in a wireless ad hoc environment. In particular, a communication efficient FDL scheme for ad hoc IoT networks:

A Requires a radically different approach than all previously proposed ad hoc network clustering schemes which are based on node IDs. This is because it is very common in these protocols that the resulting cluster heads (CHs), i.e., the nodes that will eventually perform the local averaging in the FDL scheme, are each other's one-hop neighbors. This will negatively impact the communication efficiency of the scheme due to the interference, since all the intra-cluster communication will be carried out by the CH. Moreover, almost all ad hoc clustering algorithms produce clusters with very small cardinality, thus local averaging/consensus is not very efficient with respect to the communication reduction.

B Should take into consideration that communication range restrictions (a range of a few hundred meters) results in the wireless nodes being frequently in very close proximity, thus producing very similar data, e.g., sensor measuring temperature.

So, based on the above challenges/opportunities, we contribute a purely distributed FDL scheme called DISCREETER, in which:

- We develop an ad hoc network clustering scheme appropriate for performing local and global aggregations that reduces wireless interference,
- We develop a client participation scheme into the federation that is based on data-similarity to reduce communication during local aggregation and reduce the overall data transmitted over the network in a principled way,
- We evaluate DISCREETER on convergence speed and communication reduction proving its virtues.

The rest of the section is organized as follows: in subsection 5.1.2 we describe the problem formulation in wireless settings, in subsection 5.1.3 we elaborately describe the proposed technique, in subsection 5.1.4, we make the experimental evaluation of our protocol and finally, in subsection 5.1.5, we summarize the presented work.

### 5.1.2 Problem formulation in wireless settings

Our network consists of a flat wireless ad hoc network comprised by $n$ nodes (devices), connected to each other through wireless bidirectional links, without a central server. Nodes $i$ and $j$ are

connected via link $(i, j)$ if such a link exists. The existence of a link implies that node can communicate directly, without any intermediate relays.

Each node $i$ owns (generates) a dataset $D_i$ with $D_i = |D_i|$ data points. Each data point $(\mathbf{x}, y) \in D_i$ consists of a multi-dimensional feature vector $\mathbf{x} \in R^m$ and a label $y \in R$. We let $f(\mathbf{x}, y, \mathbf{w})$ denote the loss associated with the data point $(\mathbf{x}, y)$ based on learning model parameter vector $\mathbf{w} \in R^m$. The local loss function at node $i$ is defined as:

$$F_i(\mathbf{w}) = \frac{1}{D_i} \sum_{(\mathbf{x},y)\in D_i} f(\mathbf{x}, y, \mathbf{w}). \tag{5.1}$$

In a clustered ad hoc network, we also define for a specific cluster $C$ the cluster loss function as the average local loss across the cluster:

$$F_C(\mathbf{w}) = \sum_{i\in C} F_i(\mathbf{w}). \tag{5.2}$$

The total loss function is then defined as the average loss across the whole set of clusters:

$$F(\mathbf{w}) = \sum_{\forall C} F_C(\mathbf{w}). \tag{5.3}$$

The goal of our FL model training is to find the optimal model parameters $\mathbf{w}^\star$ for $F$:

$$\mathbf{w}^\star = argmin_{\mathbf{w}\in R^m} F(\mathbf{w}). \tag{5.4}$$

In practical terms, the problem is as follows: there is a set of nodes connected over a wireless ad hoc network, and each one of them owns a data set. The goal is to train a specific neural network until convergence performing a specific learning task using these data. All nodes train the same architecture neural network[1] in iterations. Each node is willing to periodically exchange its neural model, i.e., either the neural weights, or the gradients or whatever information is appropriate so as to participate in a federation with the rest of nodes with the ultimate goal to reach a consensus neural model. Apparently, the communication overhead is tremendous and the broadcast-storm problem [188] will make the network to collapse soon and/or the nodes will quickly deplete their energy. The aforementioned settings is generic and it can accommodate any learning architecture, and any learning task. in this thesis, the data owned by each node is a time-series, the neural network is a recurrent neural network (i.e., GRU), and the learning task is that of time-series prediction.

### 5.1.3 Data Similarity-Based Hierarchical Distributed Federated Learning (DISCREETER)

The proposed federated learning method is purely distributed, and strives for communication efficiency by leveraging two ideas:

[1]This is a standard assumption; only the work [137] allows for nodes training a different neural network.

- Imposition of a hierarchy over the ad hoc network so that weight exchange is localized in neighborhoods (and not across the whole network), and moreover this local communication is performed in a way that avoids (most of) the interference with nearby neighborhoods.

- Selection of the nodes that will participate in the federation based on the how similar data they generate. Thus, handling of non-IID and sparse data becomes feasible in a natural way.

These two ideas define the two major components of the proposed methodology, and are explained in the next paragraphs.

#### 5.1.3.1 Establishing the hierarchical IoT topology

The world of ad hoc networking has developed many protocols for creating a one-level hierarchy over a flat ad hoc network; these are termed as backbone or clustering protocols. However, they fall short for the case of federated learning over ad hoc networks because: a) they are based on node IDs and thus make bad ClusterHead (CH) selections by selecting CHs not based on the topological position of the CH, and b) create a lot of small (in cardinality) clusters. In FL we would need CHs to be in a proper position so as to communicate quickly with every cluster member and also to communicate quickly and with no interference with other CHs.

To address both of these challenges, we adopt and customize to the FL context a method for creating generalized dominating sets that will act as the backbone. We adopt the max-min d-cluster formation protocol [189], which creates dominating sets with the property that a dominated node (i.e., cluster member) can be at a maximum distance of d hops from its dominator (i.e., its CH). This algorithm operates d rounds of flood-max where the largest node-IDs travel and "prevail" (i.e., become CHs), and then d rounds of flood-min take place, giving the opportunity for some not so large node-ID to "prevail" thus reducing the size of the clusters created by the flood-max rounds.

The main virtue of this algorithm is that it can significantly reduce the number of clusters for any $d \geq 2$; however the CHs of nearby clusters can still lie at one hope distance from each other causing severe interference effects. To alleviate this drawback, after the execution of a converge cast operation that allows each CH to become aware of its cluster members and of its cluster topology, the CH calculates the betweenness centrality (BC) [155] of its cluster members and of itself, and assigns the CH role to the member with the highest value of BC value or retains this role if no member has larger BC that itself. In this way, CH becomes the node that can communicate with its members in the fastest possible way and the same time it is not located in the network "border". Thus, we avoid interference with nearby clusters, and also facilitate inter-cluster communication for carrying out the global averaging task.

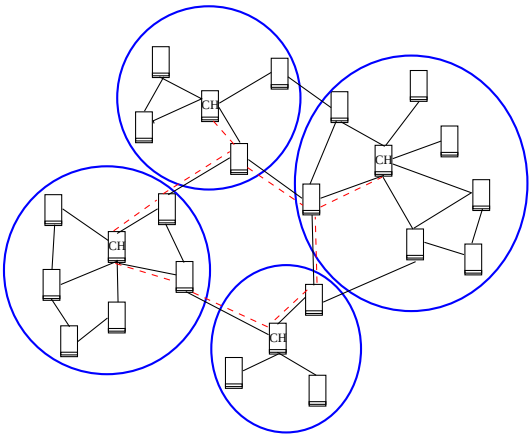



Figure 5.1: Distributed FL in a clustered ad hoc network.

#### 5.1.3.2 Data similarity-based client selection in FL

In Chapter 2 (see Section 2.3.2), we briefly mentioned various approaches for selecting the nodes that will participate in the federation; these range from complete random to various "principled" sampling. We depart from all these approaches and consider the necessity of node participation from the perspective of whether the node produces data distributed "differently" from the data of its nearby nodes. This situation occurs very frequent in scenarios where the nodes are sensors sampling temperature, moisture, capturing photos of the same scene and so on. In all these cases, data can be seen as a series of numbers; even a photo can be seen as a series of its pixel intensities. Due to geographical proximity, many sensors may produce alike data.

Thus we are facing the question of how can we calculate the similarity among timeseries (data produced by nearby nodes) in a distributed and communication-efficient way, when a) the actual timeseries are not allowed to be shared among nodes (for privacy and security issues), and b) similarity calculation must be done with minimal communication among nodes. We address these challenges at the same time by having each node calculate a sketch (i.e., a concise summary fitting in one or two wireless packets) of its time-series and sending this sketch to its CH. Then, the CH computes the similarities among the sketches and decides which node(s) will participate in the federation for the next rounds. What is the nature of a sketch?

Since we are dealing with time-series of multidimensional points x, each node computes the Discrete Fourier Transform (DFT) (in the experiments we implemented with the Fast Fourier Transform) and saves the first few coefficients. This set of a handful of coefficients comprises the sketch that it sends to its CH to perform the similarity check among its cluster members. The DFT transforms a signal from the time domain to the energy domain. The next question is whether the similarity check on the energy domain performed by the CH is equivalent to the similarity check on the time domain. This is guaranteed by the Parseval's theorem, which states that if $\overrightarrow{X}$ is the DFT of a n-point signal $\overrightarrow{x}$ , then

$$\sum_{i=0}^{n-1} |\mathbf{x_i}|^2 = \sum_{f=0}^{n-1} |\mathbf{X_f}|^2. \tag{5.5}$$

and for any two signals x and y, it holds that Distance(x,y) = Distance(X, Y), i.e., the DFT preserves the Euclidean distance between two signals.

Should each node choose to send the entire set of DFT coefficients, then we would gain nothing it terms of communication efficiency and privacy. Instead, each node sends the most significant (with highest energy) DFT coefficients to its CH, and thus the CH performs an approximate similarity check among the data produced by its cluster members. The fewer coefficients are sent to the CH, the less accurate is the similarity check and thus fewer nodes are possible to be excluded from the federation. The more coefficients are sent to the CH, the more accurate is the similarity check and thus more nodes are possible to be excluded from the federation; on the other hand, in this case we increase the communication efficiency, and energy consumption as well, but we compromising privacy, since reconstruction of the original data is possible. Of course, we have silently assumed that in our clustered architecture it holds that "nodes feel less anxiety when sending information to acquaintances" [190] (i.e., a cluster member to its CH). The issue of privacy violation (reconstruction of the original data) is also a major concern/problem for traditional federated learning since even gradient sharing can cause privacy violation. In practice, in each training round, each node generates and transmits a sketch of around a dozen coefficients. This sketch-based approach can be applied in finding similarities among weights as well [191].

#### 5.1.3.3 The federated averaging procedure

Each node i has access only to its own local dataset $D_i$ and all nodes share a common neural network model. In the proposed method, the nodes of each cluster, train their local model and share their DFT coefficients with their CH, which selects from which node it collects all the local updates, including its own and implements the averaging. Afterwards, CH broadcasts through the selected gateway nodes, the updated parameters to its related nodes in order for them to continue training. Periodically, global averaging is implemented among CHs by exchanging their parameters. Specifically, the parameters are computed and propagated via the backbone to every CH to start the next round of training. The possible communication links between the devices of each cluster (blue circles) are represented with the solid lines. Within each cluster devices communicate, only with the CH. For the global averaging there is a backbone network established by the clustering procedure that connects the CHs with each other, which is represented with the red, dashed line (see Figure 5.1). Notice there is alternative red path among CHs to avoid using the same paths for conserving energy. In introduction of the thesis, in Figure 1.3, the aforementioned procedure is depicted. Pseudocode for the DISCREETER protocol is illustrated in Algorithm 11.

#### 5.1.3.4 Complexities and Convergence

The proposed methodology involves computational and communication steps for setting up the backbone (i.e., clustering) among nodes, and also computational and communication steps for extracting and transmitting the FFT coefficients. The computational complexity of clustering is O(n), where n is the number of network nodes, and each node incurs O(1) communication complexity, i.e., O(n) in total, for establishing the backbone. Moreover, the step of deciding the final CH by calculating centralities incurs a $O(v \times e)$ cost, where is the number of nodes and edges inside a cluster. Typically, v and e are of the order of $O(\sqrt{n})$.

The extraction of the Fourier coefficients, using FFT algorithm entails a computational complexity of $O(mlog_2m)$, where m is the size of the data. Since the total size of Fourier coefficients sent to CHs is orders of magnitude smaller than the size of exchanged weights and additionally, the coefficients can be packed in the beacon messages by nodes, we do not count this overhead as communication overhead. And this comprises one more contribution of the present work, i.e., detecting data similarity among nodes with practically zero communication cost. Cluster-local and global weight convergence can be proven for all standard convexity assumptions on the loss function.

### 5.1.4 Experimental Evaluation

#### 5.1.4.1 Experimental Setup

We developed a simulation framework in Python that accommodates a wireless network topology, generated as described in [155], reliable communication means for message exchange among nodes, specific neural network architecture identical for all network nodes, and local storage. The generic workflow runs as follows: each node performs local training using owned data, then exchanges wirelessly its local weights (if it participates in the federation) and performs aggregation (as described in Section 5.1.3.3), then adopts the new weights, and continues the local training, until (local/global) convergence ratio reaches the percentage of a threshold, bigger than zero (in our case the threshold is defined as 5% of loss). Once the loss values coincide with the threshold, we stop the learning process, in order to prevent the overfitting effect which is often observed when the loss values become exactly equal to zero. For measuring performance, we used:

- Accuracy, defined as the ratio of correct predictions to total predictions (correct predictions are considered the ones with error $= \frac{|Y_{pred} - Y_{test}|}{|Y_{test}|} \leq 10\%$),
- Mean Squared Error (MSE) loss function values in every round of training for every node, and
- Communication overhead measured by the number of nodes participating in the federation.

**Algorithm 11** DISCREETER Protocol

Perform network clustering with max-min d cluster formation algorithm ▷ Elected CHs are aware of their cluster topology
Calculate BC of CH and cluster members
Reelect CHs based on Betweenness Centrality
Announce CHs and their IDs
**At EACH NODE:**
**repeat**
    Perform local training and weight update
    **if** every $k$ rounds **then**
        Compute Fourier coefficients
        Send Fourier to node's CH
    **end if**
    **if** node is elected to participate in the federation **then**
        Send local weights to its CH
        Adopt average weights received from its CH
    **end if**
**until** convergence
**At EACH CH:**
**repeat**
    Perform local training and updating weights ▷ Acts also as a normal node
    **if** every $k$ rounds **then**
        Receive Fourier coefficients from cluster members
        Compute data similarity among members
        Notify the selected cooperation nodes
        Receive node's local weights
        Perform aggregation of these weights
        Send cluster-local aggregated weights to other CHs
        Receive aggregated weights from other CHs
        Average all received weights from other CHs
        Transmit the average weights back to the selected cluster nodes
    **end if**
**until** convergence

We worked on a 20 node network clustered with the proposed variation of the max-min 2-cluster formation algorithm (d=2), and with timeseries data with 262940 samples in total [192], recording the temperatures of five Chinese cities for the period 2010-2015 for a prediction task. Each node is assigned a unique part (see an illustration in Figure 5.2) of the data (around 13000 samples each). The assignment of data parts to nodes is done so that parts belonging to the same city temperatures are assigned to topologically neighboring nodes. Therefore, the data of the network nodes are not IID, and moreover, there is probably some similarity among data of neighboring nodes. Finally, each part is divided into 27 chunks of 500 data points each, so that we have 27 rounds of training for each node which performs local training until its loss function reaches an aforementioned threshold in every round or global averaging procedure interrupts the local training. We used 80% of the dataset for training, and the rest for testing. Therefore, we used a 15-units Gated Recurrent Unit (GRU) trained with Stochastic Gradient Descent with the Adam optimizer.

We again emphasize that our solution idea is adaptive to any deep learning model with any type/kind of parameters, as long as the cooperative training takes place by weight (or gradient) exchange.

#### 5.1.4.2 Competitors

In order to compare the efficiency of our proposed method, we used two baseline methods (and their L2-based variants), focusing on the communication reduction and prediction accuracy. The first one is called FEDLwa_ANP, in which all nodes participate in the federation process, meaning that all nodes communicate with their CH and sends their parameters. This method exploits of all available data and we expect that it will incur the maximum communication cost and probably the best accuracy. The second one is called FEDLwa_RNP, in which a portion (we experimented with 25%, 50%, 75% respectively) of nodes that are randomly chosen, participate in the federation process. In this case, there is obvious communication reduction by excluding nodes from the training but the selection process is arbitrary which may cause accuracy degradation of the model, because of not selecting representative distribution data.

#### 5.1.4.3 Evaluation results

Firstly, we conducted experiments with 5, 10 and 15 Fourier coefficient variants, which resulted in 42.96% (varies from 10% to 50%) 21,48% (varies from 10% to 35%) and 13.7% (varies from 10% to 20%) average communication reduction in the network, respectively, preserving the high performance, as it is illustrated in Figures 5.3, 5.4 and Table 5.1. The experimental evaluation shows that using 15 coefficients instead of 10 or 5 is a more reliable option for approximating our data, since the more the coefficients are, the more accurate the representation of the original data is (i.e., using more coefficients implies that we detect less nodes as similar from the perspective of their data similarity). This is experimentally justified since we have a percentage drop of

almost 60% among the respective maximum communication reduction percentages, meaning that the number of excluded nodes lessens when switching from 5-coeffecients to 15-coeffecients.

Then, as it is depicted in the following Figures 5.8,5.9 and 5.5,5.6,5.7 , DISCREETER not only retains the high performance, attained when all nodes globally train the neural network model (FEDLwa_ANP method: 96,59%) but also it outperforms this baseline, while it reduces the communication rounds of total training. More elaborately, DISCREETER's prediction accuracy (about 97,26%), in case where 10 coefficients are selected for calculating Euclidean distance and thus the data similarity, exceeds about 0,7% the levels of the corresponding FEDLwa_ANP method, with a significant average reduction of approximately 21.48% (varies from 10% to 35%) in the communication overhead among the participating nodes. It is shown that the wiser selection of the data leads to less noise diffusion in the network, which justifies the reason DISCREETER achieves better performance than the baseline one. Compared to a heuristic of arbitrarily chosen participating nodes (FEDLwa_RNP), DISCREETER reaches up to 34,3% increase in accuracy levels in some cases (see Table 5.1). The accuracy drop in FEDLwa_RNP is predictable, since omitting significant data, without a selection criterion may harm model generalization, irredeemably. Especially, in case where 75% of nodes are excluded, we have an accuracy reduction of 11.15% comparatively with the FEDLwa_RNP25% implementation. Albeit, L2 regularization term mitigates the problem of exploding gradients; in our case, it decelerates convergence of DISCREETER at a ratio equal to 55,1%, since it transforms the loss function into a quadratic-like one. Hence, it might suppress the ability of a GRU to retain its memory and thus learn complex tasks. Conversely, when L2 is applied in FEDLwa_RNP, it accelerates its convergence speed about 17%, especially in case of the exclusion of the 25% of nodes, which is due to the many nodes and data being excluded from the federation. The aforementioned results are illustrated in Table 5.1 and in Figures (allChunks) and (random rounds).

### 5.1.5 Conclusions

In this work, we proposed a purely distributed FL methodology for small devices, connected via wireless ad hoc networks, with the aim of overcoming both privacy-preserving problems and excessive communication costs. Towards these goals, we developed an ad hoc network clustering scheme that creates relatively large clusters with the CH appropriately placed at the "center" of the cluster. Local averaging takes place inside each cluster and a new node participation protocol into the federation was proposed, namely DISCREETER. It is based on discovering similarity of the data from nodes that belongs to the same cluster by calculating small summaries that are transmitted to the CH which then decides which cluster members possess "different data" so as to enter the federation. Results shown that the proposed heuristic reduces up to 42% the communication overhead while it preserves its high accuracy up to 96%, compared to baseline heuristics (FEDLwa_ANP and FEDLwa_RNP).

Table 5.1: Statistics of the representative node given the final data chunk, after the global averaging, in $10_{th}$ and $20_{th}$ epochs of local training, respectively.

| Algorithm | Loss | | Accuracy(%) |
|---|---|---|---|
| | $epoch_{10_{th}}$ | $epoch_{20_{th}}$ | |
| $FEDLwa\_ANP$ | 7.35 | 8.44 | 96.59 |
| $L2-FEDLwa\_ANP$ | 11.76 | 9.12 | 96.73 |
| $FEDLwa\_RNP_{25\%}$ | 29.67 | 32.83 | 81.47 |
| $FEDLwa\_RNP_{50\%}$ | 42.31 | 35.57 | 74.99 |
| $FEDLwa\_RNP_{75\%}$ | 50.708 | 46.64 | 72.38 |
| $L2-FEDLwa\_RNP_{25\%}$ | 24.99 | 27.22 | 85.63 |
| $L2-FEDLwa\_RNP_{50\%}$ | 48.95 | 38.61 | 74.72 |
| $L2-FEDLwa\_RNP_{75\%}$ | 45.17 | 41.52 | 76.22 |
| **DISCREETER$_{\mathbf{5coef}}$** | 9.88 | 7.28 | **96.43** |
| **DISCREETER$_{\mathbf{10coef}}$** | 10.31 | 10.47 | **97.26** |
| **DISCREETER$_{\mathbf{15coef}}$** | 8.30 | 9.14 | **96.78** |
| $L2-DISCREETER_{5coef}$ | 19.03 | 16.24 | 96.45 |
| $L2-DISCREETER_{10coef}$ | 16.87 | 12.30 | 96.15 |

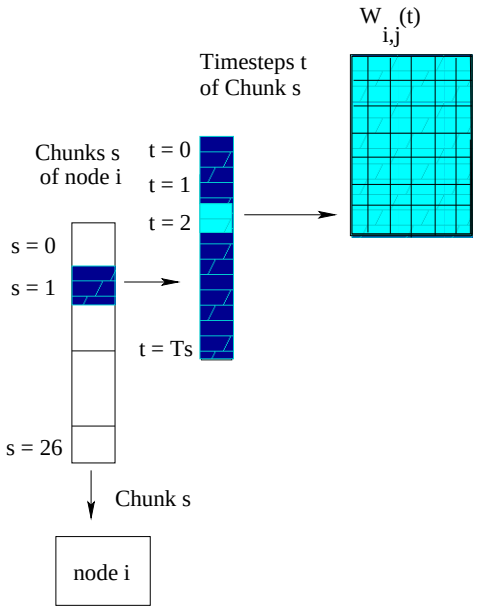



Figure 5.2: Demonstration of data chunks $s$, given to every node $i$, consecutively. Every chunk contains different input vectors $w_{i,j}(t)$, for every time-step $t \subseteq T_s$. For each chunk $s$, every node executes local training.

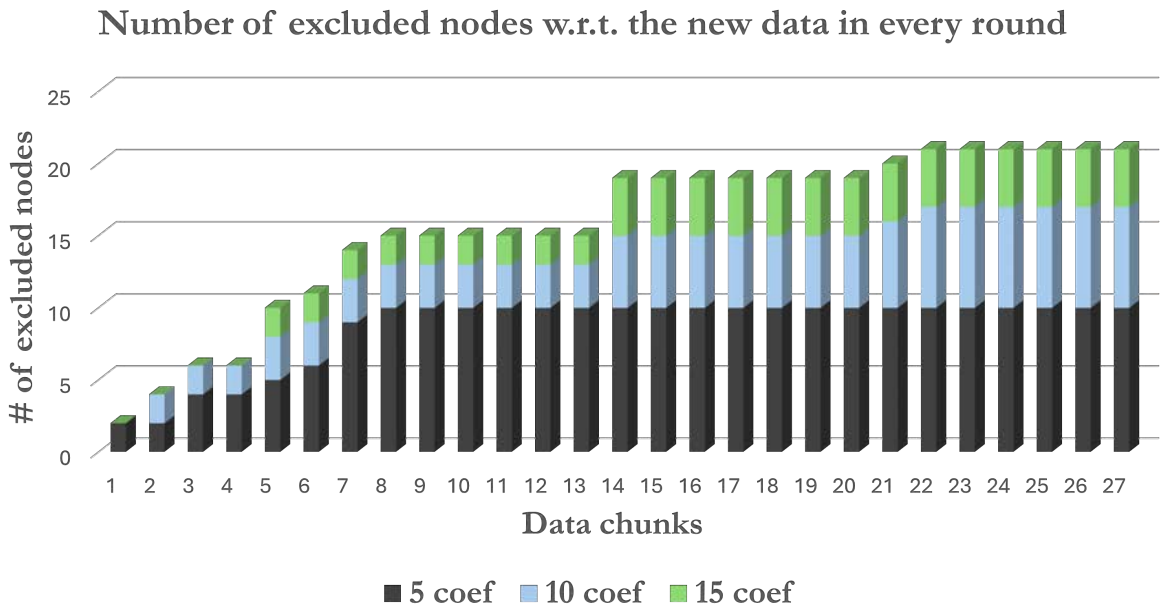



Figure 5.3: Number of nodes excluded, in respect to both the data chunks, given in the beginning of every new round of the federation process and the coefficients sent for finding similarity among the nodes' data chunks.

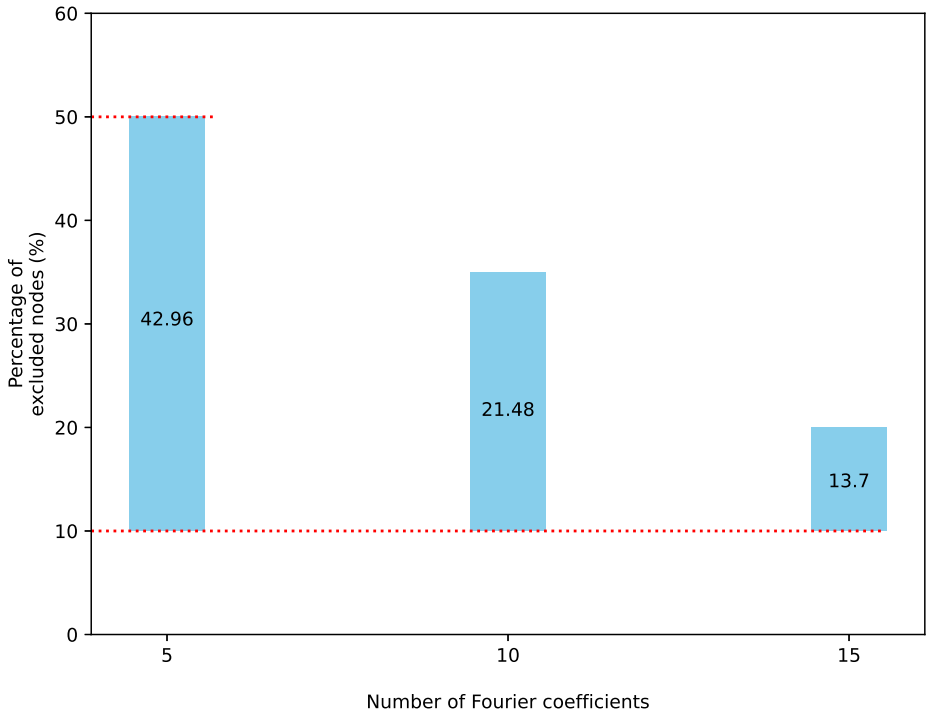



Figure 5.4: Percentage decrease and average percentage decrease (%) of the cooperating nodes as a function of the coefficients sent, during the whole federation process.

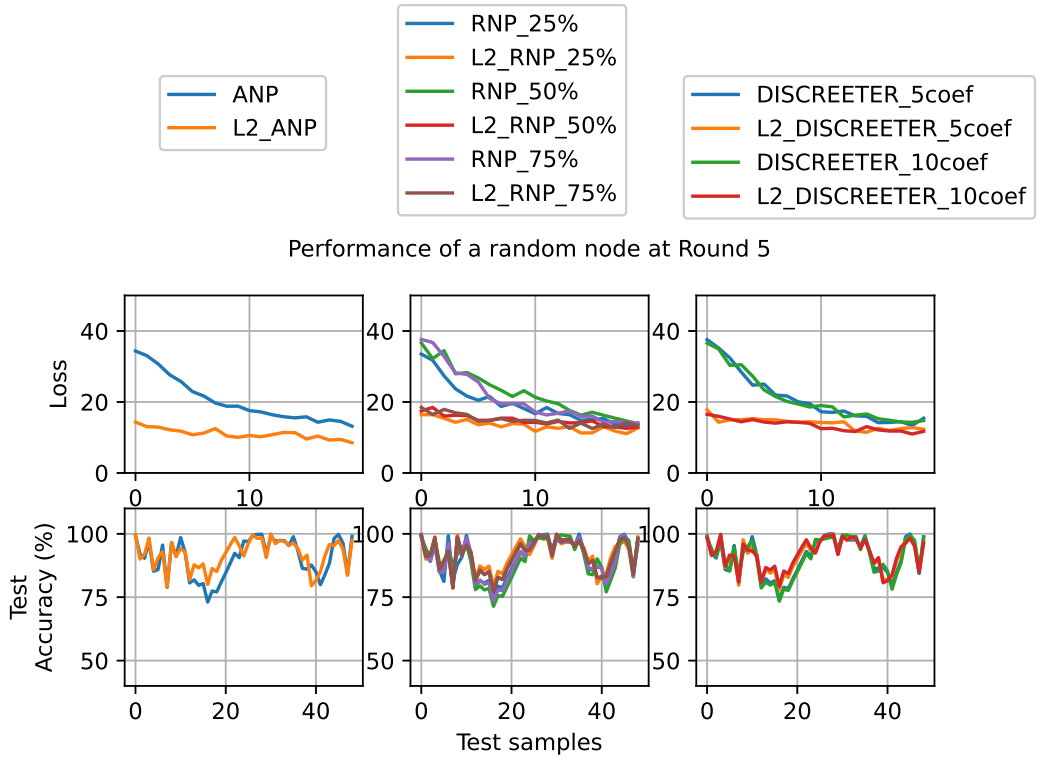



Figure 5.5: Representative node performance after the global averaging procedure at round-data chunk 5.

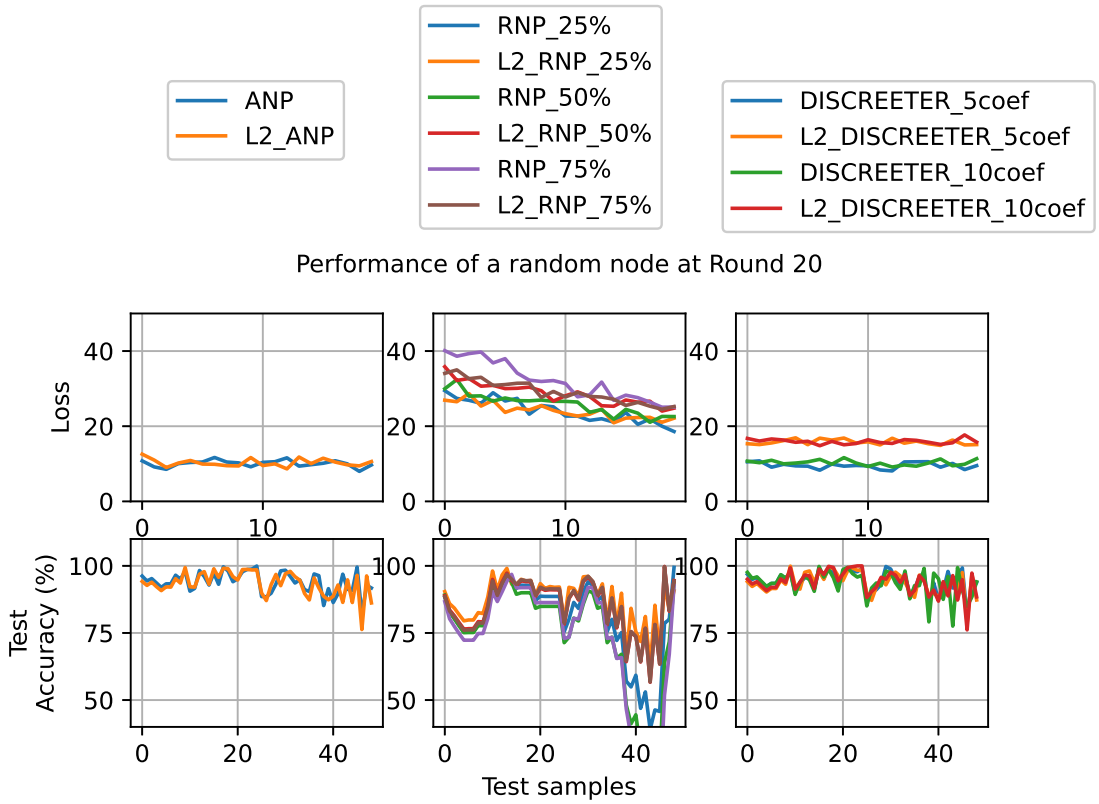



Figure 5.6: Representative node performance after the global averaging procedure at round-data chunk 20.

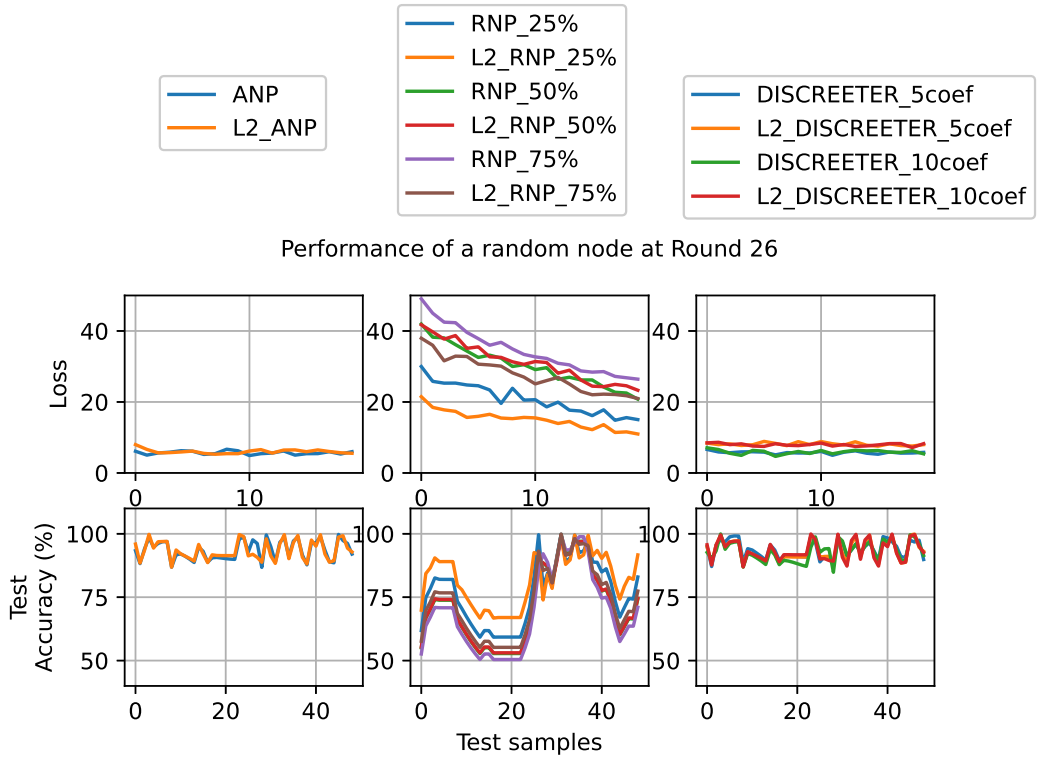



Figure 5.7: Representative node performance after the global averaging procedure at round-data chunk 26.

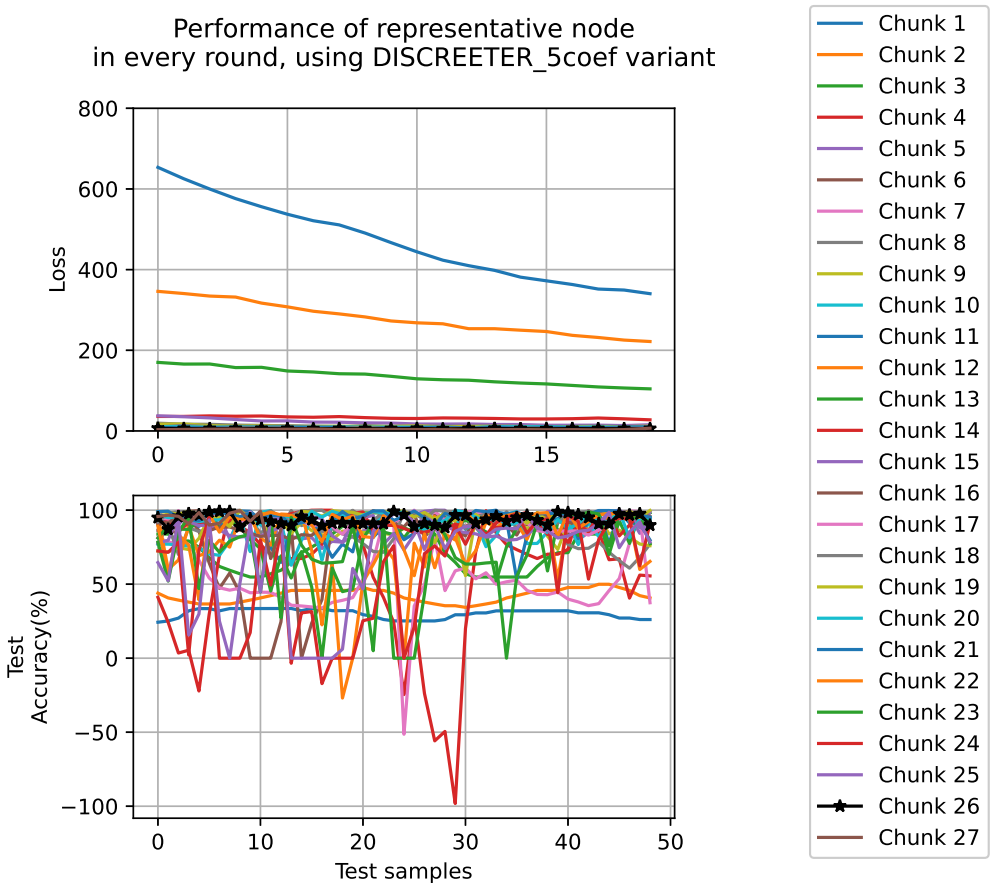



Figure 5.8: Progress of a representative node for DISCREETER with 5 Fourier coefficients w.r.t. the rounds-data chunks.

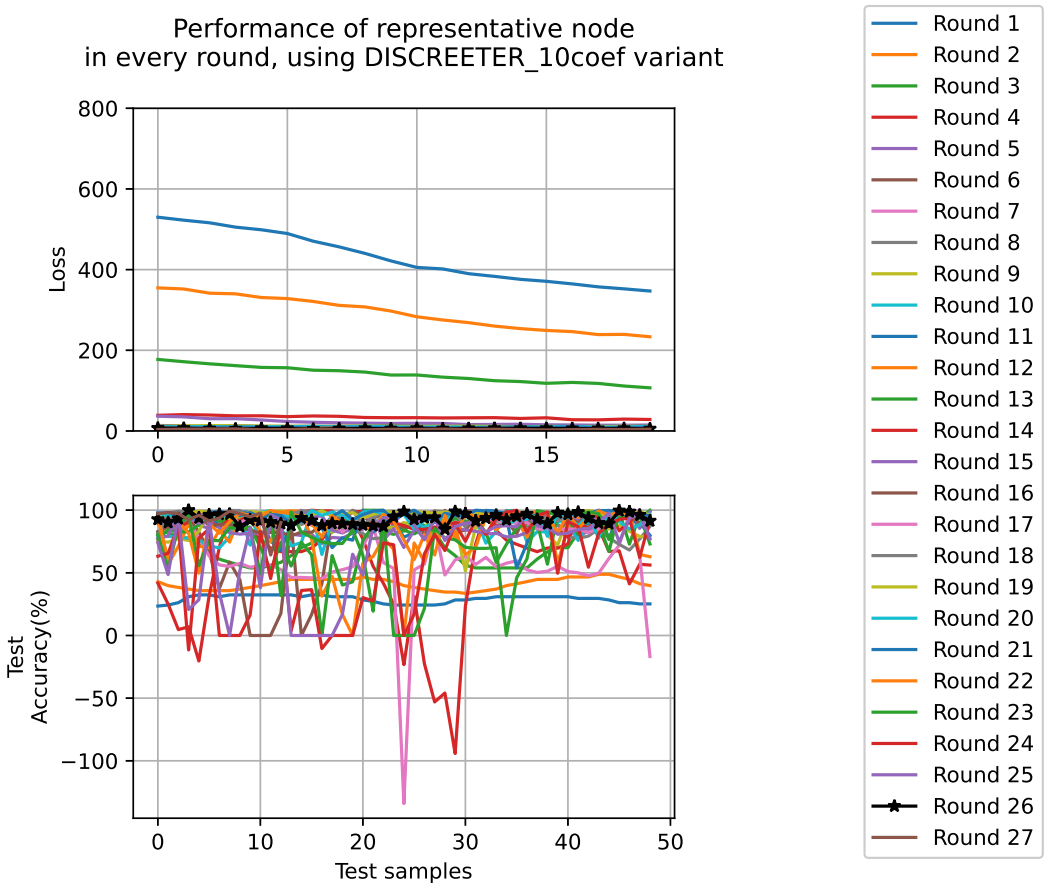



Figure 5.9: Progress of a representative node for DISCREETER with 10 Fourier coefficients w.r.t. the rounds-data chunks.

Chapter 6

# Conclusions

## 6.1 Conclusion

All things considered, the necessity of developing new Deep Learning techniques, applicable to edge resource-scarce devices e.g. sensors, MCUs, etc. in networks like Internet of Things, led to on-device learning, or else the *TinyML*, which reinforces device independence, by accelerating its*inference* state. Adversely, the inference policy is not able to address the problem of heterogeneity of data, since the classifiers of the network cannot be modified and further evolved in order to adapt to further data scenarios (interpretability). So, in this dissertation we proposed mechanisms in order for resourcelimited wirelesslyconnected devices to process their own data, by enabling the deployment of neural network training.

First of all, the challenge lies in training a neural network quickly when it has both a large number of neurons and hidden layers. Over the past thirty years, multiple techniques have emerged to speed up neural training. In this dissertation, our main emphasis is on methods based on linkage sparsification, which involve reducing the number of connections in neural topologies, in an algorithmic way, leveraging network science principles to reduce the complexity of the neural network while preserving the critical connections that facilitate efficient information flow. We draw inspiration from real neural networks, which display a scale-free (or small-world) topology. We designed algorithms that start from a particular structured but not fully connected bipartite topology and end up with another structured topology. We explored scale-free and smallworld topologies as potential starting or ending structures, by pruneandregrow mechanism of a stable amount of close-to-zero synapses (*Dynamic sparsity during training*). We examined both the classification accuracy and training time of the algorithms by evaluating their performance on publicly available datasets with a moderate-sized neural network. The outperforming algorithm was the one that produced scale-free-like topologies starting from scale-

free topologies. Specifically, the proposed method, namely SF2SFrand algorithm outperforms every other method in most of the cases mentioned, as it seems to retain high orders of accuracy. Alongside, it reduces training time approximately to 50% in some cases, compared to the dense network baselines and 20.5% less in the experiments with large scale datasets, like Fashion-Mnist). It is worth highlighting that reinforcing the strongest nodes of every layer (power-law weight distribution ), the performance of the network is proportionately enhanced, too. There was further, examination, regarding the case where the percentage of connections, both pruned and restored, is not stable but varies, dynamically, at each epoch. The most impressive result, making the aforementioned methods congruous for resource-scarce devices, is that the memory footprint is further reduced (around 95%) (while tested on the fashionmnist dataset and four different kind of neural network topologies), as more superfluous connections were removed from the network.

Since, *Transfer Learning (TL)* mechanism bridges the gap, between the random weight initialization and the alllayers training of neural network weights, new efficient inductive TL algorithm is demonstrated, in which not only the last fullyconnected, as usual in similar approaches, but also the last convolution layer are both retrained during training phase, so as for the model to update its classifiers, responsible for the better generalization of the given data. Thus, this method incurs increase in performance, by using the acquired trained model weights as a pre-knowledge, while retraining only some of the layers, consisting the neural network - not the whole network. According to the findings, the proposed method, namely $F_xC1$, achieved an approximately 19% boost in network accuracy and a convergence rate that was about 61% faster. Moreover, it is a model- aware method since it outperforms the classic Transfer Learning techniques in all cases, tested, with almost a small increase in energy consumption, compared either to the method of retraining the last fully connected layer. In this case, the model convergence is expedited, reducing the required training time, with a bit tradeoff in energy consumption.

However, even if the reduction of both training time and model size shed light on efficient DL algorithms deployment to the aforementioned devices, the amount of data, required in order for the model to converge without causing overfitting, is great. This is how federated learning (FL) come into light, in which devicesnodes cooperatively train a global modal by exchanging only some KB of information (e.g. gradients). Many modern deep learning tasks running over small devices connected via wireless ad hoc networks such as those over the Internet of Things can not benefit from the traditional federated learning architectures in which “central" server sovereignty exists for nodes coordination, due to excessive communication costs and subsequent fast depletion of node’s energies. In this thesis, We developed a distributed FL method for small devices connected through wireless ad hoc networks, addressing privacy and communication challenges. In order to reach these targets, we devised a clustering scheme for ad hoc networks that generates sizable clusters with the CH strategically positioned as the cluster’s “center”, so

as to avoid interference with nearby clusters, and also facilitate inter-cluster communication for carrying out the global averaging task. Local averaging takes place inside each cluster and a new node participation protocol into the federation was proposed, namely DISCREETER. The process involves finding similarities between data in nodes of the same cluster using small summaries. These summaries are then sent to the CH, which determines which cluster members have "different data", vi discrete Fourier Transform and should join the federation. The proposed heuristic reduces communication overhead by up to 42% while maintaining a high accuracy of up to 96% compared to baseline heuristics (when all nodes participate in the process or randomly chosen nodes participate in the process).

As a result, decentralized topologies necessitate the development of suitable communication-reliable network infrastructure or else construction of durable and compact *backbones* for facilitating federated learning techniques. So, establishing the most influential nodes in the network is of great importance. As 5G and the Internet of Things expand, network science remains always, an active and fruitful area for research and development and becomes an invaluable tool for graph-data analysis. Even after fifty years, centralities in this field are still generating new definitions and insights, making it a hot topic in network organization. In this thesis, we introduced a new member, called sink group betweenness centrality, to the family of shortest path betweenness centralities. The goal of this measure is to pinpoint nodes capable of monitoring or mediating communication to subsets of network nodes and thus being a part of a general backbone. We developed a straightforward algorithm to compute this metric and expanded it to handle node-weighted networks and edge betweenness. Our initial effort is the first step towards developing efficient algorithms for calculating these measures, analyzing their distribution in real networks, expanding them to incorporate sink group betweenness centrality for collections of network nodes, and devising approximation methods.

However, in reality, the nodesdevices are not part of simple flat networks, but they can also interact with devices from different subnets to form multilayer networks. IoBT netwroks served as a real-time case study in the current thesis, since it is a larger-scale cyberphysical system derived from the Internet of Things, with strict robustness and latency demands, as the key and arduous goal is to fulfill the commander's intent while prioritizing safety and adaptability. Utilizing the concept of *dominating sets*, we were able to achieve our goal by abstracting the IoT network topology as a multilayer graph. We then showcased a distributed algorithm designed for calculating connected dominating sets with small cardinality in the multilayer network scenario. We applied and contrasted our proposed algorithm with two state-of-the-art algorithms for calculating connected dominating sets in multilayer networks, using generated topologies. Across a range of topologies with diverse representative features, our algorithm consistently demonstrated superior performance compared to these competitors. Moreover, we proved that the problem of finding CDS in multilayer networks belongs to the family of wellknown np-complete problems, providing evidence of our contributions to the scientific world.

## 6.2 Exploring future avenues

Despite the fact that ML/DL or DFL existing methodologies have paved the way for efficient training on edge devices, there are shortcomings that have yet to be solved. For instance, the exploration of new efficient methods of implementing structural pruning in *convolution neural networks* or *transformers* is of great scientific research. Taking into consideration the complex mechanism that filters/kernels work (compared to MLP networks), we can introduce ways in order to recognize the features/weights that are amenable to pruning, without harming model's accuracy.

Moreover, the *scalability* of communication reliability in ad hoc large-scale networks entails difficulty to a great extent. The direct communication among nodes in decentralized networks is beneficial, regarding reduced latency, bandwidth rates, etc, however it requires strong network infrastructure, when nodes are geographical expanded.The major question is: how do we ensure the connection stability and thus the reliable message transmission through a geographically expanded network? New methods for determining the security of a communication link must be introduced in this scenario. Due to close proximity, nodes in distributed networks are encouraged to communicate with their one-hop neighbors, however, it doesn't ensure that the neighbor nodes are also the most trustworthy ones.

In a similar vein, a promising direction for future research involves the *identification* of trustworthy nodes for participation in federated learning frameworks. As federated learning increasingly relies on distributed edge devices or nodes, ensuring the integrity and reliability of participants becomes critical to maintaining both the accuracy and security of the global model. This raises the need for robust mechanisms capable of assessing the trustworthiness of individual nodes prior to or during their participation in training. Potential approaches may involve trust scoring systems based on historical behavior or decentralized reputation models. Addressing this challenge is essential for the advancement of secure and resilient federated learning architectures, particularly in environments where node behavior cannot be guaranteed a priori.

Exploring *Graph Neural Networks (GNNs)* for forming topologies instead of using conventional backbone-based approaches is an appealing research question. Unlike static or manually designed backbones, GNNs naturally model the irregular, dynamic, and spatially distributed nature of wireless networks, treating nodes and their interactions as a graph structure. This allows the model to learn complex relational patterns, including interference, proximity, and traffic load, which are often difficult to capture with fixed topologies. Furthermore, GNNs can generalize across varying network configurations, enabling more adaptive and scalable grouping strategies that respond to changes in node density or mobility and therefore could potentially improve communication efficiency and overall network performance.

In the context of reducing communication overhead in *federated learning*, further investigation into the criteria for identifying nodes with similar data is warranted. One interesting approach involves leveraging Parseval's theorem to assess similarity based on the weights of individual

models rather than solely relying on raw data comparisons. This method may provide a more accurate reflection of node similarity, as model weights encapsulate the learned representations and acquired knowledge of each local model, whereas raw data merely captures sensor readings at specific time instances.

Furthermore, continual adaptation of the global model in federated learning can be an effective strategy to address concept drift, particularly in environments where data distributions evolve across diverse or non-stationary feature spaces. However, this approach risks inducing catastrophic forgetting, a phenomenon where previously acquired knowledge is overwritten as the model adapts to new data. Unlike the human brain, which can retain and integrate multiple tasks over time, deep neural networks tend to "overwrite" their internal representations—effectively replacing existing weights in an effort to fit newly encountered data. The central challenge, therefore, lies in developing computationally efficient strategies to mitigate catastrophic forgetting while operating under the constraints of resource-limited devices and the demands of real-time data stream processing.

Beyond federated learning, there is also growing interest in enabling efficient inference of generative AI models—such as large language models (e.g., GPT [193])—on mobile and edge devices, which are inherently constrained in terms of computational power, memory capacity, and energy availability. To address these challenges, model compression techniques—including quantization, pruning, and knowledge distillation—should be further explored and adapted to the specific requirements of generative tasks involving text, image, or audio synthesis. Achieving real-time, personalized generative capabilities on-device could significantly enhance user experience while promoting data privacy and reducing reliance on centralized infrastructure.

In parallel, an increasingly pressing research challenge involves the integration of Green AI principles [194] into the lifecycle of generative model development and deployment. Large-scale generative architectures, such as diffusion models and transformers, are known for their substantial computational and energy requirements, which raise concerns regarding their environmental impact and long-term sustainability. Investigating strategies to mitigate this impact—such as energy-efficient training paradigms, resource-aware architecture design, and the aforementioned model compression techniques—may lead to more ecologically responsible AI systems. Moreover, examining the trade-offs between performance and efficiency could inform the establishment of new evaluation benchmarks that account for both model quality and environmental considerations. Addressing these challenges is not only critical for sustainable AI development but also essential for broadening access to advanced generative technologies in resource-constrained environments.

Lastly, as part of future research, it is imperative to advance generative AI models with a strong emphasis on ethical robustness. According to linguist Noam Chomky's perspective [195], these systems only replicate language using statistical patterns, lacking comprehension of meaning or context. While these models have demonstrated impressive capabilities, their

potential to generate biased, discriminatory, or tampered content raises serious ethical concerns. Ensuring fairness, transparency, and accountability in generative outputs must become a central objective. This includes developing mechanisms for bias detection and mitigation, traceability of synthetic content, and enforcing safeguards against malicious use such as deepfake generation [196]. Incorporating ethical/Responsible AI principles into both the training and deployment phases will be essential for building socially responsible systems that align with human values and minimize harm.

## Bibliography


[1] V. Rajapakse, I. Karunanayake, N. Ahmed, Intelligence at the extreme edge: A survey on reformable tinyml, ACM Computing Surveys (2022).

[2] E. T. M. Beltrán, M. Q. Pérez, P. M. S. Sánchez, S. L. Bernal, G. Bovet, M. G. Pérez, G. M. Pérez, A. H. Celdrán, Decentralized federated learning: Fundamentals, state of the art, frameworks, trends and challenges, IEEE Communications Surveys & Tutorials 25 (2023) 2983–3013.

[3] T. S. Ajani, A. L. Imoize, A. A. Atayero, An overview of machine learning within embedded and mobile devices–optimizations and applications, Sensors 21 (13) (2021).

[4] A. Moin, M. Challenger, A. Badii, S. Günnemann, Supporting ai engineering on the iot edge through model-driven tinyml, in: 2022 IEEE 46th Annual Computers, Software, and Applications Conference (COMPSAC), 2022, pp. 884–893.
doi:10.1109/COMPSAC54236.2022.00140.

[5] D. Papakostas, T. Kasidakis, E. Fragkou, D. Katsaros, Backbones for internet of battlefield things, in: Proceedings of the IEEE/IFIP Conference on Wireless On-demand Network Systems & Services (WONS), 2021, pp. 116–123.

[6] Y. Abadade, A. Temouden, H. Bamoumen, N. Benamar, Y. Chtouki, A. S. Hafid, A comprehensive survey on tinyml, IEEE Access 11 (2023) 96892–96922.

[7] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, X. Zheng, TensorFlow: Large-scale machine learning on heterogeneous systems, software available from tensorflow.org (2015).
URL https://www.tensorflow.org/

[8] T. Hoefler, D. Alistarh, T. Ben-Nun, N. Dryden, A. Peste, Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks, J. Mach. Learn. Res. 22 (241) (2021) 1–124.

[9] J. Yosinski, J. Clune, Y. Bengio, H. Lipson, How transferable are features in deep neural networks?, Advances in neural information processing systems 27 (2014).

[10] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, B. A. y Arcas, Communication-efficient learning of deep networks from decentralized data, in: Proceedings of the

International Conference on Artificial Intelligence and Statistics (AISTATS), 2017, pp. 1273–1282.

[11] N. Saputro, K. Akkaya, S. Uludag, A survey of routing protocols for smart grid communications, Comput. Networks 56 (2012) 2742–2771.

[12] A. Cuzzocrea, A. Papadimitriou, D. Katsaros, Y. Manolopoulos, Edge betweenness centrality: A novel algorithm for qos-based topology control over wireless sensor networks, Journal of Network & Computer Applications 35 (4) (2012) 1210–1217.

[13] H. Cai, C. Gan, L. Zhu, S. Han, TinyTL: Reduce memory, not parameters for efficient on-device learning, in: Proceedings of the Conference on Neural Information Processing Systems (NeurIPS), 2020.

[14] T. Hoefler, T. Alistart, D. ad Ben-Nun, N. Dryden, A. Peste, Sparsity in Deep Learning: Pruning and growth for efficient inference and training in neural networks, Journal of Machine Learning Research 23 (2021) 1–124.

[15] S. Han, J. Pool, J. Tran, W. Dally, Learning both weights and connections for efficient neural network, in: Proceedings of Advances in Neural Information Processing Systems, 2015, pp. 1135–1143.

[16] E. Bullmore, O. Sporns, Complex brain networks: Graph theoretical analysis of structural and functional systems, Nature Reviews on Neuroscience 10 (2009) 186–198.

[17] D. C. Mocanu, E. Mocanu, P. Stone, P. H. Nguyen, M. Gibesce, A. Liotta, Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science, Nature Communications 9 (2018).

[18] A.-L. Barabasi, Network Science, Cambridge University Press, 2016.

[19] K. Kopparapu, E. Lin, Tinyfedtl: Federated transfer learning on tiny devices, ArXiv abs/2110.01107 (2021).

[20] W. Y. B. Lim, N. C. Luong, D. T. Hoang, Y. Jiao, Y.-C. Liang, Q. Yang, D. T. Niyato, C. Miao, Federated learning in mobile edge networks: A comprehensive survey, IEEE Communications Surveys & Tutorials 22 (2020) 2031–2063.

[21] F. P.-C. Lin, S. Hosseinalipour, S. S. Azam, Federated learning beyond the star: Local D2D model consensus with global cluster sampling, in: Proceedings of the IEEE Global Communications Conference (GLOBECOM), 2021, pp. 1–6.

[22] F. P.-C. Lin, S. Hosseinalipour, S. S. Azam, C. Brinton, N. Michelusi, Semi-decentralized federated learning with cooperative D2D local model aggregations, IEEE Journal on Selected Areas in Communications 39 (12) (2021) 3851–3869.

[23] S. Hosseinalipour, S. S. Azam, C. G. Brinton, N. Michelusi, V. Aggarwal, D. J. Love, H. Dai, Multi-stage hybrid federated learning over large-scale d2d-enabled fog networks, IEEE/ACM transactions on networking 30 (4) (2022) 1569–1584.

[24] L. Li, M. Duan, Y. Zhang, A. Ren, X. Chen, Y. Tan, C. Wang, Federated learning with workloiad-aware client scheduling in heterogeneous systems, Neural Networks 154 (2022) 560–573.

[25] I. Mohammed, et al., Budgeted online selection of candidate IoT clients to participate in federated learning, IEEE Internet of Things Journal 8 (7) (2021) 5938–5952.

[26] T. S. Ajani, A. L. Imoize, Atayero, An overview of machine learning within embedded and mobile devices–optimizations and applications, Sensors (Basel, Switzerland) 21 (2021).

[27] OpenAI, et al, Gpt-4 technical report, https://arxiv.org/abs/2303.08774 (2021).

[28] A. Mokhtari, A. Ribeiro, Global convergence of online limited memory BFGS, Journal of Machine Learning Research 16 (2015) 3151–3181.

[29] S. J. Reddi, S. Kale, S. Kumar, On the convergence of Adam and beyond, in: Proceedings of the International Conference on Learning Representations (ICLR), 2018.

[30] O. Erkaymaz, Resilient back-propagation approach in small-world feed-forward neural network topology based on Newman-Watts algorithm, Neural Computing and Applications 32 (2020) 16279–16289.

[31] H. Iiduka, Appropriate learning rates of adaptive learning rate optimization algorithms for training deep neural networks, IEEE Transactions on CyberneticsTo appear (2022).

[32] H. Diao, G. Li, Y. Hao, PA-NAS: Partial operation activation for memory-efficient architecture search, Applied IntelligenceTo appear (2022).

[33] P. Ren, Y. Xiao, X. Chang, P.-Y. Huang, Z. Li, X. Chen, W. Wang, A comprehensive survey of neural architecture search: Challenges and solutions, ACM Computing Surveys 54 (76) (2021) 1–34.

[34] J. Hao, Z. Cai, R. Li, W. William Zhu, Saliency: A new selection criterion of important architectures in neural architecture search, Neural Computing and ApplicationsTo appear (2021).

[35] A. Zlateski, K. Lee, H. S. Seung, Scalable training of 3d convolutional networks on multi- and many-cores, Journal of Parallel and Distributed Computing 106 (2017) 195–204.

[36] N. P. Jouppi, C. Young, N. Patil, D. Patterson, Domain-specific architecture for deep neural networks, Communications of the ACM 61 (9) (2018) 50–59.

[37] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, Dropout: A simple way to prevent neural networks from overfitting, Journal of Machine Learning Research 15 (1) (2014) 1929–1958.

[38] P. Baldi, P. Sadowski, The dropout learning algorithm, Artificial Intelligence 210 (2014) 78–122.

[39] X. Sun, X. Ren, S. Ma, H. Wang, meProp: Sparsified back propagation for accelerted deep learning with reduced overfitting, Proceedings of Machine Learning Research 70 (2017) 3299–3308.

[40] X. Sun, X. Ren, S. Ma, B. Wei, W. Li, J. Xu, H. Wang, Y. Zhang, Training simplification and model simplification for deep learning: A minimal effort back propagation method, IEEE Transactions on Kowledge and Data EngineeringTo appear (2019).

[41] J. Frankle, M. Carbin, The lottery ticket hypothesis: Finding sparse, trainable neural networks, in: Proceedings of the International Conference on Learning Representations (ICLR), 2019.

[42] A. Renda, J. Frankle, M. Carbin, Comparing rewinding and fine-tuning in neural network pruning, in: Proceedings of the International Conference on Learning Representations (ICLR), 2020.

[43] J. Diffenderfer, B. Kailkhura, *Multi-Prize Lottery Ticket Hypothesis*: Finding accurate binary neural networks by pruning a randomly weighted networks, in: Proceedings of the International Conference on Learning Representations (ICLR), 2021.

[44] Y. Kim, Y. Li, H. Park, Y. Venkatesha, R. Yin, P. Panda, Lottery ticket hypothesis for spiking neural networks, in: Proceedings of the European Conference on Computer Vision (ECCV), 2022.

[45] M. Yao, Y. Chou, G. Zhao, X. Zheng, Y. Tian, B. Xu, G. Li, Probabilistic modeling: Proving the lottery ticket hypothesis in spiking neural network, available at: https://arxiv.org/pdf/2305.12148 (2023).

[46] J. D. D. Leon, R. Atienza, Depth pruning with auxiliary networks for tinyml, IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) (2022) 3963–3967.

[47] Y. Xie, Q. Sun, Y. Fu, Exploring lottery ticket hypothesis in few-shot learning, Neurocomputing 550 (2023) 126426.
URL `https://api.semanticscholar.org/CorpusID:259462331`

[48] L. Liebenwein, C. Baykal, B. Carter, D. Gifford, D. Rus, Lost in pruning: The effects of pruing neural networks beyond test accuracy, in: Proceedings of the International Conference on Learning Representations (ICLR), 2021.

[49] S. Disabato, M. Roveri, Incremental on-device tiny machine learning, Proceedings of the 2nd International Workshop on Challenges in Artificial Intelligence and Machine Learning for Internet of Things (2020).

[50] R. Aloufi, H. Haddadi, D. Boyle, Emotion filtering at the edge, Proceedings of the 1st Workshop on Machine Learning on Edge in Sensor Systems (2019).

[51] B. Sudharsan, J. G. Breslin, M. Tahir, M. I. Ali, O. F. Rana, S. Dustdar, R. Ranjan, S. Dustdar, Ota-tinyml: Over the air deployment of tinyml models and execution on iot devices, IEEE Internet Computing magazine 26 (2022) 69–78.

[52] L. Heim, A. Biri, Z. Qu, L. Thiele, Measuring what really matters: Optimizing neural networks for tinyml, ArXiv2104.10645 (2021).

[53] J. Li, R. Kuang, Split federated learning on micro-controllers: A keyword spotting showcase, ArXiv abs/2210.01961 (2022).

[54] C. R. Banbury, C. Zhou, I. Fedorov, R. M. Navarro, U. Thakker, D. Gope, V. J. Reddi, M. Mattina, P. N. Whatmough, Micronets: Neural network architectures for deploying tinyml applications on commodity microcontrollers, ArXiv abs/2010.11267 (2021).

[55] B. Sievers, S. Hauschild, H. Hellbrück, Inference and performance analysis of convolutional neural networks used for human gesture recognition on iot-devices, Tech. rep., Technische Hochschule Lubeck (2021).

[56] M. de Prado, M. Rusci, A. Capotondi, R. Donze, L. Benini, N. Pazos, Robustifying the deployment of tinyml models for autonomous mini-vehicles, Sensors 21 (4) (2021). doi:10.3390/s21041339.
URL https://www.mdpi.com/1424-8220/21/4/1339

[57] L. Ravaglia, M. Rusci, D. Nadalini, A. Capotondi, F. Conti, L. Benini, A tinyml platform for on-device continual learning with quantized latent replays, IEEE Journal on Emerging and Selected Topics in Circuits and Systems 11 (2021) 789–802.

[58] A. Pullini, D. Rossi, I. Loi, G. Tagliavini, L. Benini, Mr.wolf: An energy-precision scalable parallel ultra low power soc for iot edge processing, IEEE Journal of Solid-State Circuits 54 (2019) 1970–1981.

[59] T. Hoefler, D. Alistarh, T. Ben-Nun, N. Dryden, A. Peste, Sparsity in deep learning: Pruning and growh for efficient inference and training in neural networks, available at https://arxiv.org/abs/2102.00554 (2021).

[60] A. N. Gomez, I. Zhang, K. Swersky, Y. Gal, G. E. Hinton, Learning sparse networks using targeted dropout, ArXiv abs/1905.13678 (2019).

[61] S. Han, H. Mao, W. J. Dally, Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding, in: Proceedings of the International Conference on Learning Representations (ICLR), 2016.

[62] S. Narang, G. Diamos, S. Sengupta, E. Elsen, Exploring sparsity in recurrent neural networks, in: Proceedings of the International Conference on Learning Representations (ICLR), 2017.

[63] L. Cavallaro, O. Bagdasar, P. D. Meo, G. Fiumara, A. Liotta, Artificial neural networks training acceleration through network science strategies, Soft Comput. 24 (2020) 17787–17795.

[64] S. Liu, D. C. Mocanu, A. R. R. Matavalam, Y. Pei, M. Pechenizkiy, Sparse evolutionary deep learning with over one million artificial neurons on commodity hardware, Neural Computing and Applications 33 (2020) 2589–2604.

[65] X. Ma, M. Qin, F. Sun, Z. Hou, K. Yuan, Y. Xu, Y. Wang, Y. kuang Chen, R. Jin, Y. Xie, Effective model sparsification by scheduled grow-and-prune methods, ArXiv abs/2106.09857 (2021).

[66] P. Basaras, D. Katsaros, L. Tassiulas, Detecting influential spreaders in complex, dynamic networks, IEEE Computer magazine 46 (4) (2013) 26–31.

[67] A.-L. Barabasi, R. Albert, Emergence of scaling in random networks, Science 286 (5439) (1999) 509–512.

[68] J. Kwon, D. Park, Toward data-adaptable tinyml using model partial replacement for resource frugal edge device, The International Conference on High Performance Computing in Asia-Pacific Region (2021).

[69] Y. Li, Z. Li, T. Zhang, P. Zhou, S. Feng, K. Yin, Design of a novel neural network compression method for tiny machine learning, Proceedings of the 2021 5th International Conference on Electronic Information Technology and Computer Engineering (2021).

[70] H. Zhu, J. Xu, S. Liu, Y. Jin, Federated learning on non-iid data: A survey (2021). doi:10.48550/ARXIV.2106.06843.
URL https://arxiv.org/abs/2106.06843

[71] S. Disabato, M. Roveri, Tiny machine learning for concept drift, ArXiv abs/2107.14759 (2021).

[72] Y. Chen, Z. Chai, Y. Cheng, H. Rangwala, Asynchronous federated learning for sensor data with concept drift, in: IEEE International Conference on Big Data (Big Data), 2021, pp. 4822–4831.
doi:10.1109/BigData52589.2021.9671924.

[73] J. Lu, A. Liu, F. Dong, F. Gu, J. Gama, G. Zhang, Learning under concept drift: A review, IEEE Transactions on Knowledge and Data Engineering 31 (12) (2019) 2346–2363.
doi:10.1109/TKDE.2018.2876857.

[74] S. J. Pan, Q. Yang, A survey on transfer learning, IEEE Transactions on Knowledge and Data Engineering 22 (10) (2010) 1345–1359.
doi:10.1109/TKDE.2009.191.

[75] F. Zhuang, Z. Qi, K. Duan, D. Xi, Y. Zhu, H. Zhu, H. Xiong, Q. He, A comprehensive survey on transfer learning, Proceedings of the IEEE 109 (1) (2021) 43–76.
doi:10.1109/JPROC.2020.3004555.

[76] O. Day, T. M. Khoshgoftaar, A survey on heterogeneous transfer learning, Journal of Big Data 4 (2017) 1–42.

[77] F. Zhuang, Z. Qi, K. Duan, D. Xi, Y. Zhu, H. Zhu, H. Xiong, Q. He, A comprehensive survey on transfer learning, Proceedings of the IEEE 109 (1) (2020) 43–76.

[78] M. Long, Y. Cao, J. Wang, M. I. Jordan, Learning transferable features with deep adaptation networks, ArXiv abs/1502.02791 (2015).

[79] B. Sun, K. Saenko, Deep coral: Correlation alignment for deep domain adaptation, in: Computer Vision–ECCV 2016 Workshops: Amsterdam, The Netherlands, October 8-10 and 15-16, 2016, Proceedings, Part III 14, Springer, 2016, pp. 443–450.

[80] Y. Ganin, V. S. Lempitsky, Unsupervised domain adaptation by backpropagation, ArXiv abs/1409.7495 (2015).

[81] Y. Ganin, E. Ustinova, H. Ajakan, P. Germain, H. Larochelle, F. Laviolette, M. March, V. Lempitsky, Domain-adversarial training of neural networks, Journal of machine learning research 17 (59) (2016) 1–35.

[82] H. Ren, D. Anicic, T. A. Runkler, Tinyol: Tinyml with online-learning on microcontrollers, International Joint Conference on Neural Networks (IJCNN) (2021) 1–8.

[83] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, et al., Overcoming catastrophic forgetting in neural networks, Proceedings of the national academy of sciences 114 (13) (2017) 3521–3526.

[84] L. Freeman, A set of measures of centrality based on betweenness, Sociometry 40 (1) (1977) 35–41.

[85] U. Brandes, On variants of shortest-path betweenness centrality and their generic computation, Social Networks 30 (2) (2008) 136–145.

[86] W. Hwang, T. Kim, M. Ramanathan, A. Zhang, Bridging centrality: Graph mining from element level to group level, in: Proceedings of the ACM Conference on Knowledge Discovery & Data Mining (KDD), 2008, pp. 336–344.

[87] S. Dolev, Y. Elovici, R. Puzis, Routing betweenness centrality, Journal of the ACM 57 (4) (2010).

[88] P. Mahendra, P. Mikhail, H. Liaquat, Percolation centrality: Quantifying graph-theoretic impact of nodes during percolation in networks, PLOS One 8 (1) (2013).

[89] M. Bonchi, F. De Francisci, R. M, Centrality measures on big graphs: Exact, approximated, and distributed algorithms, in: Proceedings of the ACM International Conference on the World Wide Web (WWW), 2016, pp. 1017–1020.

[90] L. Maccari, L. Ghiro, A. Guerrieri, A. Montresor, R. Lo Cigno, Exact distributed load centrality computation: Algorithms, convergence, and applications to distance vector routing, IEEE Transactions on Parallel & Distributed Systems 31 (7) (2020) 1693–1706.

[91] N. Magaia, A. Francisco, P. Pereira, M. Correia, Betweenness centrality in delay tolerant networks: A survey, Ad Hoc Networks 33 (2015) 284–305.

[92] D. Katsaros, N. Dimokas, L. Tassiulas, Social network analysis concepts in the design of wireless ad hoc network, IEEE Network Magazine 24 (6) (2010) 23–29.

[93] P. Pantazopoulos, M. Karaliopoulos, I. Stavrakakis, Distributed placement of autonomic internet services, IEEE Transactions on Parallel & Distributed Systems 25 (7) (2014) 1702–1712.

[94] D. Bader, S. Kintali, K. Madduri, M. Mihail, Approximating betweenness centrality, in: Proceedings of the International Workshop on Algorithms & Models for the Web-Graph (WAW), 2007, pp. 124–137.

[95] R. Geisberger, P. Sanders, D. Schultes, Better approximation of betweenness centrality, in: Proceedings of the Meeting on Algorithm Engineering & Experiments (ALENEX), 2008, pp. 90–100.

[96] J. Matta, G. Ercal, K. Sinha, Comparing the speed and accuracy of approaches to betweenness centrality approximation, Computational Social Networks 6 (2019).

[97] M. G. Everett, S. P. Borgatti, The centrality of groups and classes, The Journal of mathematical sociology 23 (3) (1999) 181–201.

[98] E. Kolaczyk, D. Chua, M. Barthelemy, Group betweenness and co-betweenness: Inter-related notions of coalition centrality, Social Networks 31 (2009) 190–203.

[99] M. Chehreghani, Effective co-betweenness centrality computation, in: Proceedings of the ACM Conference on Web Search & Data Mining (SDM), 2014, pp. 423–432.

[100] R. Puzis, Y. Elovici, S. Dolev, Fast algorithm for successive computation of group betweenness centrality, Physical Review E 76 (2007) 056709.

[101] L. Li, G. Wang, M. Yu, Pairwise co-betweenness for several types of network, Journal of Networks 10 (2) (2015) 91–98.

[102] S. Dolev, Y. Elovici, R. Puzis, P. Zilberman, Incremental deployment of network monitors based on group betweenness centrality, Information Processing Letters 109 (2009) 1172–1176.

[103] X. Bei, W. Chen, S. Teng, J. Zhang, J. Zhu, Bounded budget betweenness centrality game for strategic network formations, Theoretical Computer Science 412 (2011) 7147–7168.

[104] H. Ramos, A. Frery, A. Boukerche, E. Oliveira, A. Loureiro, Topology-related metrics and applications for the design and operation of wireless sensor networks, ACM Transactions on Sensor Networks 10 (3) (2014) 1–35.

[105] I. Stojmenovic, M. Seddigh, J. Zunic, Dominating sets and neighbor elimination-based broadcasting algorithms in wireless networks, IEEE Transactions on Parallel and Distributed Systems 13 (1) (2002) 14–25.

[106] J. Wu, H. Li, On calculating connected dominating set for efficient routing in ad hoc wireless networks, in: Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications, 1999, pp. 7–14.

[107] J. Yu, N. Wang, G. Wang, D. Yu, Connected dominating sets in wireless ad hoc and sensor networks: A comprehensive survey, Computer Communications 24 (6) (2013) 121—-134.

[108] A. Kott, A. Swami, B. J. West, The Internet of Battle Things, IEEE Computer magazine (2016) 70–75.

[109] T. Abdelzaher, et al., Toward and Internet of Battle Things: A resilience perspective, IEEE Computer magazine 51 (11) (2018) 24–36.

[110] D. Papakostas, P. Basaras, D. Katsaros, L. Tassiulas, Backbone formation in military multi-layer ad hoc networks using complex network concepts, in: IEEE Military Communications Conference, 2016, pp. 842–848.

[111] D. Papakostas, S. Eshghi, D. Katsaros, L. Tassiulas, Energy-aware backbone formation in military multilayer ad hoc networks, Ad Hoc Networks 81 (2018) 17–44.

[112] M. J. Farooq, Q. Zhu, On the secure and reconfigurable multi-layer network design for critical information dissemination in the Internet of Battlefield Things (IoBT), IEEE Transactions on Wireless Communications 17 (4) (2018) 2618–2632.

[113] N. B. Gaikwad, H. Ugale, A. Keskar, N. C. Shivaprakash, The Internet-of-Battlefield-Things (IoBT)-based enemy localization using soldiers location and gunshot direction, IEEE Internet of Things Journal 7 (12) (2020) 11725–11734.

[114] N. Abuzainab, W. Saad, Dynamic connectivity game for adversarial Internet of Battlefield Things Systems, IEEE Internet of Things Journal 5 (1) (2018) 378–390.

[115] N. Abuzainab, W. Saad, A multiclass mean-field game for thwarting misinformation spread in the Internet of Battlefield Things, IEEE Transactions on Communications 66 (12) (2018) 6643–6658.

[116] A. Azmoodeh, A. Dehghantanha, K.-K. R. Choo, Robust malware detection for Internet of (Battlefield) Things devices using deep eigenspace learning, IEEE Transactions on Sustainable Computing 4 (1) (2019) 88–95.

[117] A. Castiglione, K.-K. R. Choo, M. Nappi, S. Ricciardi, Context aware ubiquitous biometrics in edge of military things, IEEE Cloud Computing magazine 4 (6) (2017) 16–20.

[118] I. Nasim, S. Kim, Human EMf exposure in wearable networks for Internet of Battlefield Things, in: IEEE Military Communications Conference, 2019.

[119] D. Zhao, G. Xiao, Z. Wang, L. Wang, L. Xu, Minimum dominating set of multiplex networks: Definition, application, and identification, IEEE Transactions on Systems, Man, and Cybernetics: Systems 51 (12) (2021) 7823–7837.

[120] Y. Wu, Y. Li, Construction algorithms for $k$-connected $m$-dominating sets in wireless sensor networks, in: ACM International Symposium on Mobile Ad Hoc Networking and Computing, 2008, pp. 83–90.

[121] S. V. Pemmaraju, I. A. Pirwani, Energy conservation via domatic partitions, in: ACM International Symposium on Mobile Ad Hoc Networking and Computing, 2006, pp. 143–154.

[122] J. C. Nacher, M. Ishitsuka, S. Miyazaki, T. Akutsu, Finding and analysing the minimum set of driver nodes required to control multilayer networks, Nature Scientific Reports 9 (576) (2019).

[123] N. Fan, J.-P. Watson, Solving the connected dominating set problem and power dominating set problem by integer programming, in: International Conference on Combinatorial Optimization and Applications, 2012, pp. 371–383.

[124] Z. Li, Q. Wen, Z. Wu, S. Hu, N. Wang, Y. Li, X. Liu, B. He, A survey on federated learning systems: Vision, hype and reality for data privacy and protection, IEEE Transactions on Knowledge and Data Engineering 35 (4) (2023) 3347–3366.

[125] Q. Yang, Y. Liu, Y. Cheng, Y. Kang, T. Chen, H. Yu, Federated Learning, Synthesis Lectures on Artificial Intelligence and Machine Learning, Morgan-Claypool Publishers, 2019.

[126] J. Wen, Z. Zhang, Y. Lan, Z. Cui, J. Cai, W. Zhang, A survey on federated learning: Challenges and applications, International Journal of Machine Learning and Cybernetics 14 (2023) 513–535.

[127] C. Zhang, Y. Xie, H. Bai, B. Yu, W. Li, Y. Gao, A survey on federated learning, Knowledge-Based Systems 216 (2021).

[128] J. Lin, W. Yu, N. Zhang, X. Yang, H. Zhang, W. Zhao, A survey on Internet of Things: Architecture, enabling technologies, security and privacy, and applications, IEEE Internet of Things Journal 4 (5) (2017) 1125–1142.

[129] D. Papakostas, P. Basaras, D. Katsaros, L. Tassiulas, Backbone formation in military multi-layer ad hoc networks using complex network concepts, in: Proceedings of the IEEE Military Communications Conference (MILCOM), 2016, pp. 842–848.

[130] S. Hosseinalipour, S.-S. Azam, C. G. Brinton, V. Michelusi, N. abd Aggrawal, D. J. Love, H. Dai, Multi-stage hybrid federated learning over large-scale D2D-enabled fog networks, IEEE/ACM Transactions on Networking 30 (4) (2022) 1569–1584.

[131] S. Klara, J. Wen, J. C. Cresswell, M. Volkovs, H. R. Tizhoosh, Decentralized federated learning through proxy model sharing, Nature Communications 14 (2899) (2023).

[132] J. Wang, A. K. Sahu, Z. Yang, G. Joshi, S. Kar, MATCHA: Speeding up decentralized SGD via matching decomposition sampling, in: Proceedings of the Indian Control Conference (ICC), 2019, pp. 299–300.

[133] S. Savazzi, M. Nicoli, V. Rampa, Federated learning with cooperating devices: A consensus approach for massive IoT networks, IEEE Internet of Things Journal 7 (5) (2020) 4641–4654.

[134] E. T. M. Beltran, M. Q. Perez, P. M. S. Sanchez, S. L. Bernal, G. Bovet, M. G. Perez, G. M. Perez, A. H. Celdran, Decentralized federated learning: Fundamentals, state of the art, frameworks, trends and challenges, IEEE Communications Surveys & Tutorials 25 (4) (2023) 2983–3013.

[135] L. Yuan, L. Sun, P. S. Yu, Z. Wang, Decentralized federated learning: A survey and perspective, available at: https://arxiv.org/abs/2306.01603 (2023).

[136] E. Gabrielli, G. Pica, G. Tolomei, A survey on decentralized federated learning, available at: https://arxiv.org/abs/2308.04604 (2023).

[137] A. Taya, T. Nishio, M. Morikura, K. Yamamoto, Decentralized and model-free federated learning: Consensus-based distillation in function space, IEEE Transactions on Signal and Information Processing over Networks 8 (2022) 799–814.

[138] Y. Fraboni, R. Vidal, L. Kamenu, M. Lorenzi, A general theory for client sampling in federated learning, in: Proceedings of the International Workshop on Trustworthy Federated Learning, 2022.

[139] G. Tan, H. Yuan, H. Hu, S. Zhou, Z. Zhang, A framework of decentralized federated learning with soft clustering and 1-bit compressed sensing for vehicular networks, IEEE Internet of Things JournalTo appear (2024).

[140] Q. Pan, H. Cao, Y. Zhu, J. Liu, B. Li, Contextual client selection for efficient federated learning over edge devices, IEEE Transactions on Mobile ComputingTo appear (2024).

[141] Q. Pan, H. Cao, Y. Zhu, J. Liu, B. Li, Contextual client selection for efficient federated learning over edge devices, IEEE Transactions on Mobile ComputingTo appear (2024).

[142] S. Russel, T. Abdelzaher, The Internet of Battlefield Things: The next generation of command, control, communications and intelligence (C3I) decision-making, in: IEEE Conference on Military Communications, 2018, pp. 737–742.

[143] T. Abdelzaher, et al., Will distributed computing revolutionize peace? The emergence of Battlefield IoT, in: IEEE International Conference on Distributed Computing Systems, 2018, pp. 1129–1138.

[144] T. W. Haynes, S. Hedetniemi, P. Slater, Fundamentals of Domination in Graphs, Chapman & Hall/CRC Pure and Applied Mathematics, CRC Press, 1998.

[145] Y.-C. Tseng, S.-Y. Ni, Y.-S. Chen, J.-P. Sheu, The broadcast storm problem in a mobile ad hoc network, ACM/Springer Wireless Networks 8 (2-3) (2002) 153–167.

[146] M. R. Garey, D. S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W. H. Freenan and Company, 1979.

[147] A. Barabasi, R. Albert, Emergence of scaling in random networks, Science 286 (5439) (1999) 509–512.

[148] D. Easley, J. Kleinberg, Networks, Crowds and Markets: Reasoning About a Highly Connected World, Cambridge University Press, 2010.

[149] S. Fortunato, Community detection in graphs, Physics Reports 486 (3-5) (2010) 75–174.

[150] S. Fortunato, D. Hric, Community detection in networks: A user guide, Physics Reports 659 (2016) 1–44.

[151] N. Masuda, R. Lambiotte, A Guide to Temporal Networks, Vol. 4 of Series on Complexity Science, World Scientific, 2016.

[152] G. Bianconi, Multilayer Networks: Structure and Function, Oxford University Press, 2018.

[153] A. Halu, R. Mondragón, P. Panzarasa, G. Bianconi, Multiplex pagerank, PLOS One 8 (10) (2013) e78293.

[154] X. Huang, D. Chen, T. Ren, D. Wang, A survey of community detection methods in multilayer networks, Data Mining & Knowledge Discovery 35 (2021) 1–45.

[155] P. Basaras, G. Iosifidis, D. Katsaros, L. Tassiulas, Identifying influential spreaders in complex multilayer networks: A centrality perspective, IEEE Transactions on Network Science & Engineering 6 (1) (2019) 31–45.

[156] V. Lattora, V. Nicosia, G. Russo, Complex Networks: Principles, Methods and Applications, Cambridge University Press, 2017.

[157] M. Newman, Networks: An Introduction, Oxford University Press, 2010.

[158] A. Langville, C. Meyer, Google's PageRank and Beyond: The Science of Search Engine Rankings, Princeton University Press, 2006.

[159] D. Katsaros, P. Basaras, Detecting influential nodes in complex networks with range probabilistic control centrality, in: J. van Schuppen, T. Villa (Eds.), Coordination Control of Distributed Systems, Vol. 456 of Lecture Notes in Control and Information Sciences, Springer-Verlag, 2015, pp. 265–272.

[160] M. Kitsak, L. Gallos, S. Havlin, F. Liljeros, L. Muchnik, H. Stanley, H. Makse, Identification of influential spreaders in complex networks, Nature Physics 6 (2010) 888–893.

[161] J. Borge-Holthoefer, Y. Moreno, Absence of influential spreaders in rumor dynamics, Physical Review E 85 (2) (2012).

[162] P. Basaras, D. Katsaros, L. Tassiulas, Detecting influential spreaders in complex, dynamic networks, IEEE Computer Magazine 46 (4) (2013) 26–31.

[163] U. Brandes, A faster algorithm for betweenness centrality, Journal of Mathematical Sociology 25 (2) (2001) 163–177.

[164] G. Akanmu, F. Wang, H. Chen, Introducing weighted nodes to evaluate the cloud computing topology, Journal of Software Engineering & Applications 5 (11) (2012) 961–969.

[165] M. Girvan, M. Newman, Community structure in social and biological networks, Proceedings of the National Academy of Sciences 99 (2002) 7821–7826.

[166] P. Basaras, D. Katsaros, L. Tassiulas, Dynamically blocking contagions in complex networks by cutting vital connections, in: Proceedings of the IEEE International Conference on Communications (ICC), 2015, pp. 1170–1175.

[167] G. Seyfang, N. Longhurst, Growing green money? mapping community currencies for sustainable development, Ecological Economics 86 (2013) 65–77.

[168] I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, The MIT Press, 2016.

[169] P. P. Ray, A review on TinyML: State-of-the-art and prospects, Journal of King Saud University – Computer and Information SciencesTo appear (2022).

[170] H. Xiao, K. Rasul, R. Vollgraf, Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, ArXiv1708.07747 (2017).

[171] A. Krizhevsky, V. Nair, G. Hinton, Learning multiple layers of features from tiny images, technical Report, Computer Science Department, University of Toronto (2009).

[172] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, L. Fei-Fei, Imagenet: A large-scale hierarchical image database, in: IEEE conference on computer vision and pattern recognition, Ieee, 2009, pp. 248–255.

[173] P. Bansal, Image classification, intel, https://www.kaggle.com/datasets/puneet6060/ intel-image-classification (2019).

[174] S. Chen, Beijing, Pm2.5, UCI, machine learning repository, https://doi.org/10. 24432/ C5JS49 (2017).

[175] X. Wang, Z. Zheng, Y. He, F. Yan, Z. qiang Zeng, Y. Yang, Soft person reidentification network pruning via blockwise adjacent filter decaying, IEEE Transactions on Cybernetics (2021).

[176] Z.-Q. Hong, J.-Y. Yang, Optimal discriminant plane for a small number of samples and design method of classifier on the plane, Pattern Recognition 24 (1991) 317–324.

[177] A. P. James, S. Dimitrijev, Feature selection using nearest attributes, available at: https://arxiv.org/abs/1201.5946 (2012).

[178] S. A. Nene, S. K. Nayar, H. Murase, Columbia Object Image Library (COIL-20), Tech. Rep. CUCS-006-96, Columbia University (1996).

[179] S. Xu, H. Chen, X. Gong, K. Liu, J. Lu, B. Zhang, Efficient structured pruning based on deep feature stabilization, Neural Computing and Applications 33 (2021) 7409–7420.

[180] S. Qiu, X. Xu, B. Cai, FReLU: Flexible Rectified Linear Units for improving convolutional neural networks, available at: https://arxiv.org/abs/1706.08098. (2019).

[181] Q. Abbas, F. Ahmad, M. Imran, Variable learning rate based modification in backpropagation algorithm (MBPA) of artificial neural network for data classification, Science International 28 (3) (2016) 2369–2380.

[182] X. Dai, H. Yin, N. K. Jha, NeST: A neural network synthesis tool based on a grow-and-prune paradigm, ArXiv1711.02017 (2018).

[183] M. Zemlyanikin, A. Smorkalov, T. Khanova, A. Petrovicheva, G. Serebryakov, 512kib ram is enough! live camera face recognition dnn on mcu, 2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW) (2019) 2493–2500.

[184] C. R. Banbury, V. J. Reddi, M. Lam, W. Fu, A. Fazel, J. Holleman, X. Huang, R. Hurtado, D. Kanter, A. Lokhmotov, D. A. Patterson, D. Pau, J. sun Seo, J. Sieracki, U. Thakker, M. Verhelst, P. Yadav, Benchmarking tinyml systems: Challenges and direction, ArXiv abs/2003.04821 (2020).

[185] Q. Li, Z. Wen, Z. Wu, S. Hu, N. Wang, Y. Li, X. Liu, B. He, A survey on federated learning systems: Vision, hype and reality for data privacy and protection, IEEE Transactions on Knowledge and Data Engineering (TKDE) 35 (2023) 3347–3366.

[186] S. Savazzi, M. Nicoli, V. Rampa, Federated learning with cooperating devices: A consensus approach for massive iot networks, IEEE Internet of Things Journal 7 (2020) 4641–4654.

[187] H. Ochiai, Y. Sun, Q. Jin, N. Wongwiwatchai, H. Esaki, Wireless ad hoc federated learning: A fully distributed cooperative machine learning, arXiv2205.11779 (2022).

[188] S.-Y. Ni, Y.-C. Tseng, Y.-S. Chen, J.-P. Sheu, The broadcast storm problem in a mobile ad hoc network, in: Proceedings of the ACM/IEEE International Conference on Mobile Computing and Networking (MOBICOM), 1999, pp. 151–162.

[189] A. D. Amis, R. Prakash, T. H. Vuong, D. T. Huynh, Max-min d-cluster formation in wireless ad hoc networks, in: Proceedings IEEE INFOCOM 2000. Conference on

Computer Communications. 19th Annual Joint Conference of the IEEE Computer and Communications Societies, Vol. 1, IEEE, 2000, pp. 32–41.

[190] A. Taya, T. Nishio, M. Morikura, K. Yamamoto, Decentralized and model-free federated learning: Consensus-based distillation in function space, IEEE Transactions on Signal and Information Processing over Networks 8 (2022) 799–814.

[191] P. Tian, W. Liao, W. Yu, E. Blasch, Wscc: A weight-similarity-based client clustering approach for non-iid federated learning, IEEE Internet of Things Journal 9 (2022) 20243–20256.

[192] S. Chen, Beijing, Pm2.5, UCI, machine learning repository, https://doi.org/10. 24432/C5JS49 (2017).

[193] P. Zhang, M. N. Kamel Boulos, Generative ai in medicine and healthcare: Promises, opportunities and challenges, Future Internet 15 (9) (2023) 286.

[194] D. Thakur, A. Guzzo, G. Fortino, F. Piccialli, Green federated learning: A new era of green aware ai, ACM Comput. Surv. 57 (8) (Mar. 2025).
doi:10.1145/3718363.
URL https://doi.org/10.1145/3718363

[195] N. Chomsky, I. Roberts, J. Watumull, Noam chomsky: The false promise of chatgpt, The New York Times 8 (2023).

[196] K. Kenthapadi, H. Lakkaraju, N. Rajani, Generative ai meets responsible ai: Practical challenges and opportunities, in: ACM Conference on Knowledge Discovery and Data Mining (KDD), 2023, pp. 5805–5806.