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

Αυτόματη αναζήτηση αρχιτεκτονικής νευρωνικού δικτύου με χρήση γενετικών μοντέλων για γραφήματα

Διπλωματικη Εργασια

του

ΜΙΧΑΛΗ ΧΑΤΖΗΑΝΑΣΤΑΣΗ

Επιβλέποντες: Μιχάλης Βαζιργιάννης: Καθηγητής Ecole Polytechnique
Γεώργιος Στάμου: Αναπληρωτής Καθηγητής Ε.Μ.Π

Αθήνα, Νοέμβριος 2020

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

Αυτόματη αναζήτηση αρχιτεκτονικής νευρωνικού δικτύου με χρήση γενετικών μοντέλων για γραφήματα

Διπλωματικη Εργασια

του

ΜΙΧΑΛΗ ΧΑΤΖΗΑΝΑΣΤΑΣΗ

Επιβλέποντες: Μιχάλης Βαζιργιάννης: Καθηγητής Ecole Polytechnique
Γεώργιος Στάμου: Αναπληρωτής Καθηγητής Ε.Μ.Π

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 22α Νοεμβρίου 2020.

(Υπογραφή)	*(Υπογραφή)*	*(Υπογραφή)*
.....	.....	.....
Γεώργιος Στάμου	Μιχάλης Βαζιργιάννης	Αντρέας Σταφυλοπάτης
Αναπληρωτής Καθηγητής Ε.Μ.Π	Καθηγητής Ecole Polytechnique	Καθηγητής Ε.Μ.Π.

Αθήνα, Νοέμβριος 2020

(Υπογραφή)

.....

ΜΙΧΑΛΗΣ ΧΑΤΖΗΑΝΑΣΤΑΣΗΣ

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

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

Ευχαριστίες

Με την ολοκλήρωση των προπτυχιακών σπουδών μου, θα ήθελα να ευχαριστήσω ειδικά κάποιους ανθρώπους, οι οποίοι υπήρξαν καθοριστικοί για την πορεία μου αυτά τα χρόνια.

Αρχικά θα ήθελα να ευχαριστήσω τον επιβλέπων καθηγητή μου κ. Μιχάλη Βαζιργιάννη. Ο κ. Βαζιργιάννης είναι καθηγητής στο πανεπιστήμιο École Polytechnique στο Παρίσι, και μου έδωσε την ευκαιρία να εκπονήσω την διπλωματική μου εργασία στο εργαστήριο του. Η εμπιστοσύνη του και οι συμβουλές του σε καθημερινό επίπεδο, τους τελευταίους μήνες, ήταν καθοριστικές για την διεκπεραίωση της παρούσας εργασίας. Αισθάνομαι ιδιαίτερα τυχερός καθώς ξεκίνησα τα ερευνητικά μου βήματα υπό την καθοδήγηση του.

Στη συνέχεια, θα ήθελα να ευχαριστήσω τον καθηγητή μου Γεώργιο Στάμου, ο οποίο υπήρξε ο επιβλέπων καθηγητής από την πλευρά του Εθνικού Μετσόβιου Πολυτεχνείου. Μέσω των μαθημάτων του με εισήγαγε στον τομέα της Τεχνητής Νοημοσύνης, και μου έδωσε ισχυρά θεμέλια ώστε να πραγματοποιήσω την παρούσα εργασία.

Επίσης, θα ήθελα να ευχαριστήσω τον κ. Γεώργιο Σιόλα, ο οποίος υπήρξε ιδιαίτερα βοηθητικός όλους αυτούς τους μήνες καθώς και τον διδακτορικό φοιτητή Γεώργιο Δασούλα, οι ιδέες του οποίου ήταν σημαντικές για την δημιουργία αυτής της εργασίας.

Θα ήθελα ακόμη, να ευχαριστήσω όλους τους φίλους μου για αυτά τα υπέροχα φοιτητικά χρόνια που περάσαμε μαζί. Κυρίως, όμως, θα ήθελα να ευχαριστήσω την οικογένειά μου, τους γονείς μου Ιωάννη και Τσαμπίκα, και τον αδερφό μου Εμμανουήλ, οι οποίοι με υποστήριξαν από την πρώτη στιγμή, και με τις θυσίες τους μου έδωσαν τις ευκαιρίες και τα εφόδια που χρειαζόμουν. Τέλος, θα ήθελα να αφιερώσω την παρούσα Διπλωματική εργασία στην μνήμη της γιαγιάς μου Χριστίνας, η οποία υπήρξε υπόδειγμα ανθρώπου για εμένα.

Acknowledgements

Upon completion of my undergraduate studies, I would like to thank specific people who have been important to me.

First of all, I would like to thank my supervisor, Professor Michalis Vazirgiannis. Mr. Vazirgiannis is a professor in École Polytechnique, in France, and he gave me the opportunity to pursue my thesis in his laboratory. His advice on a daily basis, has been crucial in carrying out this work. I feel very fortunate as I started my research career under his guidance.

In addition, I would like to thank my professor Georgios Stamou, who was my supervisor in National Technical University of Athens. Through his courses he introduced me to the field of Artificial Intelligence, and gave me a strong foundation to achieve my goals.

I would also like to thank Dr. Georgios Siolas, who has been by my side all these months as well as the doctoral student Georgios Dasoulas, whose ideas were significant for the creation of this work.

Moreover, I would like to thank all my friends for these wonderful student years we spent together. Above all, I would like to thank my family, my parents Ioannis and Tsampika, and my brother Emmanouil, who supported me from the first moment, and with their sacrifices, they gave me the opportunities I needed. Finally, I would like to dedicate this Diploma Thesis to the memory of my grandmother Christina, who was a role model to me.

Περίληψη

Η αναζήτηση αρχιτεκτονικής νευρωνικού δικτύου (Neural Architecture Search) έχει αποκτήσει μεγάλη προσοχή πρόσφατα, καθώς προσπαθεί να αυτοματοποιήσει την εύρεση της αρχιτεκτονικής του νευρωνικού δικτύου για διάφορα προβλήματα. Ωστόσο, η αναπαράσταση της αρχιτεκτονικής είναι μια σύνθετη διαδικασία. Οι υπάρχουσες μέθοδοι αντιπροσωπεύουν την αρχιτεκτονική του νευρωνικού δικτύου είτε ως μια ακολουθία από συμβολοσειρές, η οποία δεν συλλαμβάνει τις δομικές ιδιότητες της αρχιτεκτονικής, είτε χρησιμοποιούν έναν γράφημα ώστε να καθορίσουν τις συνδέσεις μεταξύ των επιπέδων, σε συνδυασμό με πίνακες ονε-ηοτ κωδικοποίησης, οι οποίοι αναπαριστούν την λειτουργία που εκτελείται σε κάθε επίπεδο. Σε αυτή τη διπλωματική, προτείνουμε έναν εναλλασσόμενο αυτόματο κωδικοποιητή για γραφήματα (Graph Variational Autoencoder), που κωδικοποιεί τις αρχιτεκτονικές σε έναν λανθάνοντα χώρο (latent space), χρησιμοποιώντας πυκνές αναπαραστάσεις πινάκων για τις λειτουργίες κάθε επιπέδου της αρχιτεκτονικής (operation embeddings). Με αυτόν τον τρόπο, μπορούμε να συλλάβουμε τις συσχετίσεις μεταξύ των διαφορετικών λειτουργιών πιο αποτελεσματικά, και ως εκ τούτου το μοντέλο μας παράγει με ακρίβεια έναν ομαλό λανθάνοντα χώρο, χαρτογραφώντας αρχιτεκτονικές παρόμοιων επιδόσεων κοντά την μια στην άλλη. Η πραγματοποίηση βελτιστοποίησης σε έναν τόσο ομαλό λανθάνοντα χώρο, είναι πολύ αποτελεσματική για το πρόβλημα της αναζήτησης αρχιτεκτονικής νευρικού δικτύου. Η προσέγγιση μας αξιολογήθηκε με πολλαπλά πειράματα, τα αποτελέσματα των οποίων έδειξαν πολύτιμα πλεονεκτήματα της μεθόδου μας σε σχέση με άλλες υπάρχοντες τεχνικές που στοχεύουν στη δημιουργία αρχιτεκτονικών νευρωνικών δικτύων υψηλής απόδοσης.

Λέξεις Κλειδιά

μηχανική μάθηση, βαθιά μάθηση, αναζήτηση αρχιτεκτονικής νευρωνικού δικτύου, γραφήματα, γενετικά μοντέλα, αυτόματοι εναλλασσόμενοι κωδικοποιητές, μπεϋζιανή βελτιστοποίηση.

Abstract

Neural Architecture Search (NAS) has gained increased attention recently, as a class of approaches that automates the architecture engineering. However, the representation of a neural architecture is no trivial. Existing methods either represent neural network architectures as a sequence of strings that do not capture the structural properties of the architectures, or they use a graph based representation along with one-hot vectors to indicate the connections between the layers and the operation performed in every layer respectively. In this thesis, we propose a graph variational autoencoder, that encodes network architectures in a latent space, using operation embeddings. We can capture the relations between the different operations more effectively, and therefore our model can accurately produce a smooth latent space, by mapping similar performance architectures close to each other. Performing optimization in such a smooth latent space, is very efficient for the neural network architecture search task. The introduced approach has been evaluated over series of experiments and demonstrates valuable advantages over other existing techniques targeting the generation of high performing neural network architectures.

Keywords

machine learning,deep learning, neural architecture search, graphs, graph generative models, variational autoencoders, operation embeddings, bayesian optimization

Contents

Εκτεταμένη Περίληψη στα Ελληνικά

Εισαγωγή

Κίνητρο

Τα τελευταία χρόνια υπάρχει μια εκτεταμένη βιομηχανική και ερευνητική δραστηριότητα στον τομέα της βαθιάς μάθησης και συνεπώς έχει δημιουργηθεί μια συνεχής ανάγκη για όλο και καλύτερες αρχιτεκτονικές νευρωνικών δικτύων. Για αυτό απαιτείται από τον μηχανικό, να δώσει ιδιαίτερη έμφαση στην κατασκευή της αρχιτεκτονικής. Δηλαδή, στο πλήθος των νευρώνων που υπάρχουν σε κάθε επίπεδο, στις συνδέσεις μεταξύ των επιπέδων, στην λειτουργία κάθε επιπέδου, καθώς και στην συνάρτηση ενεργοποίησης. Όμως όλες αυτές οι αρχιτεκτονικές επιλογές, επιβάλλονται από τον άνθρωπο, συνεπώς αναπόφευκτα τα μοντέλα περιορίζονται από τις γνώσεις του κάθε μηχανικού. Επίσης, η διαδικασία της κατασκευής της αρχιτεκτονικής είναι αρκετά χρονοβόρα, και οδηγεί πολλές φορές σε λάθη. Τέλος, δεν έχουμε ακόμη θεμελιωμένη θεωρία για το ποια χαρακτηριστικά είναι αυτά που οδηγούν σε καλές αρχιτεκτονικές.

Η αυτόματη αναζήτηση αρχιτεκτονικής (Neural Architecture Search) έχει αποκτήσει μεγάλη προσοχή πρόσφατα, ως μια κατηγορία προσεγγίσεων που αυτοματοποιεί την εύρεση της αρχιτεκτονικής, προκειμένου να παράγει καλύτερα μοντέλα για διάφορους τομείς [135, 96]. Η αναζήτηση της αρχιτεκτονικής του νευρωνικού δικτύου μπορεί να διατυπωθεί ως ένα πρόβλημα βελτιστοποίησης σε έναν χώρου εισόδου των αρχιτεκτονικών. Ουσιαστικά ψάχνουμε μια αρχιτεκτονική από έναν χώρο εισόδου, η οποία μεγιστοποιεί την απόδοση σε ένα συγκεκριμένο πρόβλημα της επιλογής μας. Αυτή η αναζήτηση είναι ιδιαίτερα απαιτητική λόγω δύο συγκεκριμένων ιδιοτήτων. Καταρχήν, ο χώρος εισόδου είναι υψηλών διαστάσεων, με συνήθως μια διακριτή ή μη ευκλείδεια δομή, όπως ένα γράφημα [128] ή μια ακολουθία [134]. Η άμεση αναζήτηση μιας αρχιτεκτονικής σε αυτόν τον χώρο είναι αναποτελεσματική δεδομένης της εκθετικής αύξησης του χώρου αναζήτησης, καθώς ο αριθμός των επιπέδων ή των λειτουργιών

αυξάνεται [77]. Επιπλέον, η αντικειμενική συνάρτηση που θέλουμε να βελτιστοποιήσουμε, δηλαδή η απόδοση της αρχιτεκτονικής, είναι δύσκολο να εκτιμηθεί, καθώς απαιτείται πρώτα η εκπαίδευσης της αρχιτεκτονικής, η οποία έχει υψηλό κόστος.

Συμβολή

Σε αυτήν τη Διπλωματική Εργασία, προκειμένου να αντιμετωπίσουμε τις παραπάνω προκλήσεις, χρησιμοποιούμε τεχνικές βελτιστοποίησης σε λανθάνοντα χώρο (latent space optimization) [36, 77, 75]. Συγκεκριμένα, χρησιμοποιούμε έναν εναλλασσόμενο αυτόματο κωδικοποιητή προκειμένου να αντιστοιχήσουμε τις αρχιτεκτονικές σε έναν συνεχή και ομαλό λανθάνοντα χώρο. Στη συνέχεια, εκτελούμε μπεϋζιανή βελτιστοποίηση στον λανθάνοντα χώρο για να βρούμε την καλύτερη αρχιτεκτονική.

Η κωδικοποίηση της αρχιτεκτονικής είναι κρίσιμη για την επίλυση του προβλήματος. Εκμεταλλευόμαστε την γραφηματική δομή της αρχιτεκτονικής και την αναπαριστούμε ως ένα ακυκλικό κατευθυνόμενο γράφημα. Χρησιμοποιούμε νευρωνικά δίκτυα για γραφήματα (Graph Neural Networks), ώστε να κωδικοποιήσουμε κάθε αρχιτεκτονική. Αναπαριστώντας την αρχιτεκτονική ως ένα κατευθυνόμενο ακυκλικό γράφημα, εκμεταλλευόμαστε τις δομικές πληροφορίες της. Οι δομικές πληροφορίες, δηλαδή οι συνδέσεις των επιπέδων μεταξύ τους, μαζί με τις λειτουργικές πληροφορίες, δηλαδή με την λειτουργία κάθε επιπέδου, ορίζουν την αρχιτεκτονική. Συνεπώς, για να κατασκευάσουμε ένας ομαλό λανθάνοντα χώρο, δηλαδή οι αρχιτεκτονικές που έχουν παρόμοια απόδοση να αντιστοιχίζονται σε κοντινές αναπαραστάσεις στον λανθάνοντα χώρο, πρέπει να εκμεταλλευτούμε και τις λειτουργικές πληροφορίες. Για αυτό τον λόγο, προτείνουμε τις πυκνές αναπαραστάσεις πινάκων για την λειτουργία κάθε επιπέδου (operation embeddings). Οπότε, αντί να αναπαριστούμε κάθε λειτουργία με one-hot κωδικοποίηση, μαθαίνουμε με αυτόματο τρόπο μικρής διαστατικότητας πυκνούς πίνακες, και εκμεταλλευόμαστε τις συσχετίσεις μεταξύ των λειτουργιών. Για παράδειγμα, ένα συνελικτικό επίπεδο (convolutional layer) με διαστάσεις πυρήνα 5x5 , είναι πιο κοντά σαν λειτουργικότητα με ένα συνελικτικό επίπεδο με διαστάσεις πυρήνα 3x3 , παρά με ένα επίπεδο χωρικής υποδειγματοληψίας (pooling layer).

Τα ευρήματά μας δείχνουν ότι η πρότασή μας για πυκνές αναπαραστάσεις των λειτουργιών, οδηγεί στην ομαδοποίηση των καλύτερων αρχιτεκτονικών, παράγοντας έτσι έναν ομαλό λανθάνοντα χώρο, ωφελώντας την εύρεση της καλύτερης αρχιτεκτονικής.

Θεωρητικό Υπόβαθρο

Βαθιά Νευρωνικά Δίκτυα για Γραφήματα (Graph Neural Networks)

Οι αρχιτεκτονικές βαθιάς μάθησης, όπως τα συνελικτικά νευρωνικά δίκτυα (Convolutional Neural Networks) ή τα αναδρομικά νευρωνικά δίκτυα (Reccurent Neural Networks), είναι εξαιρετικά χρήσιμα όταν επεξεργάζονται Ευκλείδιες δομές δεδομένων, όπως για παράδειγμα

εικόνες. Προκειμένου όμως να επεξεργαστούν δεδομένα που έχουν την μορφή γραφημάτων, μια απλή μέθοδος είναι να αναπαρασταθεί το γράφημα με τον πίνακα γειτνίασης του και να χρησιμοποιηθεί ο πίνακας ως είσοδο στο νευρωνικό δίκτυο. Το πρόβλημα με αυτήν την προσέγγιση, είναι ότι τα κλασσικά νευρικά δίκτυα παράγουν διαφορετικές εξόδους για διαφορετικές μεταθέσεις των κόμβων στον πίνακα γειτνίασης. Όμως όλοι αυτοί οι πίνακες, αντιπροσωπεύουν το ίδιο γράφημα. Επομένως, είναι απαραίτητο να διασφαλίσουμε ότι η αρχιτεκτονική μας είναι αμετάβλητη σε αυτές τις μεταθέσεις, δηλαδή ότι επεξεργάζεται με τον ίδιο τρόπο τα ομοιόμορφα γραφήματα.

Η βασική κλάση αρχιτεκτονικών που χρησιμοποιείται ώστε να επεξεργαστεί δεδομένα με την μορφή γραφημάτων ονομάζεται Graph Neural Networks (GNN) [99]. Ο στόχος του GNN είναι να μάθει μια αναπαράσταση $h_u \in \mathbb{R}^s$ για κάθε κόμβο u του γραφήματος, καταγράφοντας τις γειτονικές πληροφορίες κάθε κόμβου. Για ένα πρόβλημα ταξινόμησης των κόμβων, αυτές οι αναπαραστάσεις δίνονται ως είσοδο σε έναν ταξινομητή που παράγει την τελική ετικέτα κάθε κόμβου. Εάν θέλουμε να αντιμετωπίσουμε ένα πρόβλημα ταξινόμησης γραφημάτων, μπορούμε να συγκεντρώσουμε όλες τις αναπαραστάσεις των κόμβων, προκειμένου να λάβουμε την αναπαράσταση του γραφήματος h_G και, στη συνέχεια, να τις τροφοδοτήσουμε σε έναν ταξινομητή ώστε να παράξει την ετικέτα του γραφήματος. Για να μάθει αυτές τις αναπαραστάσεις, το GNN χρησιμοποιεί μια διαδικασία, η οποία ονομάζεται neural message passing [34], στο οποίο κάθε κόμβος ανταλλάσσει μηνύματα με τους γείτονές του, ώστε να σχηματίσει την αναπαράσταση του.

Γενετικά Μοντέλα για Γραφήματα (Deep Graph Generative Models)

Στην προηγούμενη υποενότητα, δείξαμε πώς μπορούμε να μάθουμε αναπαραστάσεις γραφημάτων, προκειμένου να κάνουμε προβλέψεις για τους κόμβους (πρόβλημα ταξινόμησης κόμβων), για τις ακμές (πρόβλημα πρόβλεψης συνδέσμου) ή για ολόκληρο το γράφημα (πρόβλημα ταξινόμησης γραφήματος). Σε αυτήν την υποενότητα, παρέχουμε τεχνικές για ένα παρόμοιο πρόβλημα, αυτό της δημιουργίας νέων γραφημάτων. Ο στόχος είναινα εξάγουμε γραφήματα που έχουν συγκεκριμένες ιδιότητες ή που δημιουργούνται από μια συγκεκριμένη κατανομή. Για παράδειγμα, μπορεί να θέλουμε να δημιουργήσουμε μόρια με συγκεκριμένες ιδιότητες [125] ή να δημιουργήσουμε γραφήματα σκηνών από εικόνες [123]. Με τη βαθιά μάθηση, εφαρμόζουμε τεχνικές ώστε να μάθουμε ένα γενετικό μοντέλο γραφημάτων, με βάση τα γραφήματα εισόδου και να παράγουμε νέα γραφήματα με παρόμοιες ιδιότητες. Ταξινομούμε αυτές τις τεχνικές με βάση δύο ιδιότητες:

1. **Αναπαράσταση του γραφήματος**. Υπάρχουν τρεις βασικοί τύποι αναπαράστασης του γραφήματος: βασισμένοι σε συμβολοσειρές, βασισμένοι σε πίνακες γειτνίασης, και βασισμένοι σε γραφηματικές δομές. Οι προσεγγίσεις που βασίζονται σε συμβολοσειρές αντιπροσωπεύουν ένα γράφημα ως ακολουθία συμβολοσειρών. Για παράδειγμα, τα γραφήματα μορίων μπορούν να αναπαρασταθούν με τη γραμματική SMILES [113, 21]. Οι μέθοδοι αυτοί χρησιμοποιούν αναδρομικά νευρωνικά δίκτυα λόγω του πλεονεκτήματός τους στην αναπαράσταση αλληλουχίας συμβολοσειρών. Η άλλη κατηγορία

αντιπροσωπεύει το γράφημα με τον πίνακα γειτνίασης [101, 126]. Αυτές οι μέθοδοι πρέπει να λαμβάνουν υπόψη ότι υπάρχουν διαφορετικοί πίνακες που αντιπροσωπεύουν το ίδιο γράφημα, λόγω μετάθεσης των κόμβων. Επομένως, είναι απαραίτητη η χρήση συναρτήσεων αμετάβλητων στις μεταθέσεις [84]. Οι προσεγγίσεις βάσει γραφημάτων, οι οποίες τράβηξαν μεγάλη προσοχή τα τελευταία χρόνια, χρησιμοποιούν Graph Neural Networks προκειμένου να παράγουν την αναπαράσταση των κόμβων και του γραφήματος. Λειτουργούν απευθείας στην δομή του γραφήματος, αποτυπώνοντας αποτελεσματικά τις σχέσεις μεταξύ των κόμβων. [129, 127, 67].

2. **Το γενετικό πλαίσιο**. Οι πιο δημοφιλείς τεχνικές είναι οι αυτόματοι εναλλασσόμενοι κωδικοποιητές (Variational Autoencoders (VAE)), τα αυτοπαλινδρομούμενα μοντέλα και τα γενετικά ανταγωνιστικά δίκτυα (Γενερατιε Αδερσαριαλ Νετωορκς).

Αναζήτηση Αρχιτεκτονικής Νευρωνικού Δικτύου (Neural Architecture Search)

Όλες οι αρχιτεκτονικές βαθιάς μάθησης, από τα συνελικτικά και αναδρομικά νευρωνικά δίκτυα, έως τα νευρωνικά δίκτυα για γραφήματα, απαιτούν εκτεταμένη προσοχή στην σχεδίαση τους. Η διαδικασία κατασκευής της κατάλληλης αρχιτεκτονικής είναι εξαιρετικά χρονοβόρα καθώς ο ερευνητής-μηχανικός πρέπει να σχεδιάσει την αρχιτεκτονική χειροκίνητα, με βάση την εμπειρία που έχει στον τομέα. Αναπόφευκτα οι επιλογές των αρχιτεκτονικών περιορίζονται από τον ανθρώπινο παράγοντα. Ως εκ τούτου, τα τελευταία χρόνια, υπάρχει μια αυξανόμενη προσοχή στην αυτοματοποίηση της αρχιτεκτονικής αρχιτεκτονικής. Το πεδίο αυτό ονομάζεται Neural Architecture Search, έχει αποδείξει την ικανότητά να κατασκευάζει αρχιτεκτονικές που επιτυγχάνουν κορυφαία αποτελέσματα, σε διάφορα προβλήματα [135, 136, 96, 90, 33], με ελάχιστη ανθρώπινη παρέμβαση. Σε αυτήν την ενότητα, παρέχουμε μια σύντομη επισκόπηση του τομέα.

Χώρος Αναζήτησης (Search Space)

Ο χώρος αναζήτησης καθορίζει το σύνολο όλων των αρχιτεκτονικών νευρωνικών δικτύων που μπορούν να αναπαρασταθούν. Προκειμένου να οριστεί μια αρχιτεκτονική, πρέπει να καθοριστούν δύο ιδιότητες: η λειτουργία που σχετίζεται με κάθε κόμβο ή επίπεδο και οι συνδέσεις μεταξύ τους. Στο Σχήμα 2.17, παρατίθενται παραδείγματα χώρων αναζήτησης. Το μέγεθος του χώρου αναζήτησης, παίζει καθοριστικό ρόλο στην εκφραστικότητα του μοντέλου, αλλά επηρεάζει και το υπολογιστικό κόστος του συστήματος [116]. Επομένως, πρέπει να υπάρχει ένας συμβιβασμός μεταξύ του αριθμού των διαφορετικών αρχιτεκτονικών που θέλουμε να αντιπροσωπεύσουμε και του κόστους που θα έχει ο αλγόριθμός μας. Καθώς τα βαθιά νευρωνικά δίκτυα έχουν συνήθως εκατοντάδες επίπεδα, ο αριθμός των διαφορετικών επιλογών αυξάνεται εκθετικά. Για την αντιμετώπιση αυτού του προβλήματος, συνήθως επιβάλουμε ορισμένους περιορισμούς στη δομή και τη λειτουργία των επιπέδων, περιορίζοντας έτσι το μέγεθος του χώρου αναζήτησης.

Αλγόριθμος Αναζήτησης (Search Algorithm)

Έχουν χρησιμοποιηθεί πολλοί διαφορετικοί αλγόριθμοι βελτιστοποίησης για την εύρεση της καλύτερης αρχιτεκτονικής στον χώρο αναζήτησης, όπως η τυχαία αναζήτηση [64], η ενισχυτική μάθηση [135], μπεϋζιανή βελιστοποίηση [49], εξελικτικές μέθοδοι [89] και μέθοδοι που βασίζονται σε κλίση καθόδου (ενγραδιεντ δεσςεντ) [72]. Η επιλογή της μεθόδου βελτιστοποίησης επηρεάζεται σε μεγάλο βαθμό από τον χώρο αναζήτησης, καθώς ορισμένοι αλγόριθμοι απαιτούν συγκεκριμένα χαρακτηριστικά από τον χώρο αναζήτησης. Για παράδειγμα, για να εφαρμοστούν gradient descent τεχνικές, ο χώρος αναζήτησης πρέπει να είναι συνεχής.

Εκτίμηση Απόδοσης Performance Evaluation

Κάθε στρατηγική αναζήτησης, στοχεύει να βρει μια αρχιτεκτονική που μεγιστοποιεί μια μετρική απόδοσης, όπως η ακρίβεια στο σύνολο επικύρωσης. Ωστόσο, πριν από την εκτίμηση αυτής της απόδοσης, η αρχιτεκτονική πρέπει να εκπαιδευτεί σε ένα σετ δεδομένων. Λόγω του γεγονότος ότι ένας κοινός χώρος αναζήτησης περιέχει έναν πολύ μεγάλο αριθμό αρχιτεκτονικών, η διαδικασία εκπαίδευσης όλων των αρχιτεκτονικών έχει ένα ιδιαίτερα υψηλό υπολογιστικό κόστος. Στον Πίνακα 2.1, απεικονίζεται το υπολογιστικό κόστος πολλών μεθόδων αναζήτησης αρχιτεκτονικής νευρωνικού δικτύου στο σύνολο δεδομένων CIFAR-10. Ακόμη και για ένα μικρό σύνολο δεδομένων όπως το CIFAR-10, η διαδικασία αναζήτησης μπορεί να διαρκέσει εκατοντάδες GPU-ημέρες. Για να γίνει πιο αποτελεσματική η αναζήτηση της αρχιτεκτονικής, είναι απαραίτητο να ξεπεραστεί το συγκεκριμένο εμπόδιο, δηλαδή η διαδικασία εκπαίδευσης των αρχιτεκτονικών. Επομένως, οι πρόσφατες προσπάθειες στον τομέα προσπαθούν να μειώσουν το κόστος της εκπαίδευσης ή να εκτιμήσουν την απόδοση μιας αρχιτεκτονικής χωρίς να την εκπαιδεύσουν πλήρως. Μια σύνοψη αυτών των προσεγγίσεων παρουσιάζεται στον Πίνακα 2.2. Αυτές οι τεχνικές έκαναν την αναζήτηση νευρικής αρχιτεκτονικής προσβάσιμη σε περισσότερους ερευνητές, καθώς το υπολογιστικό κόστος έχει μειωθεί δραματικά, χωρίς να επηρεάζεται η απόδοση των μεθόδων [87].

Μεθοδολογία

Αυτή η ενότητα περιέχει την κύρια συμβολή της διατριβής μας. Αρχικά, παρέχουμε μια λεπτομερή βιβλιογραφική επισκόπηση παρόμοιων εργασιών. Στη συνέχεια προχωράμε στην διατύπωση του προβλήματος και έπειτα αναλύουμε το μοντέλο που αναπτύξαμε.

Συναφής Βιβλιογραφία

Οι εναλλασσόμενοι αυτόματοι κωδικοποιητές για γραφήματα έχουν δείξει πολλά υποσχόμενα αποτελέσματα στο πρόβλημα της αναζήτησης αρχιτεκτονικής νευρωνικού δικτύου. Η πλησιέστερη εργασία με την δική μας, είναι από τους Zhang et al. [129], οι οποίοι χρησιμοποιούν έναν εναλλασσόμενο αυτόματο κωδικοποιητή (D-VAE) προκειμένου να κατασκευάσουν έναν

λανθάνοντα χώρο και στη συνέχεια, εφαρμόζουν μπεϋζιανή βελτιστοποίηση σε αυτόν τον χώρο. Η βασική διαφορά είναι ότι εμείς χρησιμοποιούμε operation embeddings ώστε να αναπαραστήσουμε τις λειτουργίες στα επίπεδα της αρχιτεκτονικής, ενώ αυτοί χρησιμοποιούν πίνακες με κωδικοποίηση one-hot. Έτσι δεν μπορούν να καταγράψουν τις σχέσεις μεταξύ των λειτουργιών και τελικά παράγουν έναν λιγότερο ομαλό λανθάνοντα χώρο.

Οι Li et al. [63] πρότειναν επίσης έναν εναλλασσόμενο αυτόματο κωδικοποιητή (NGAE) προκειμένου να εφαρμόσουν μεθόδους βελτιστοποίησης στον συνεχή λανθάνοντα χώρο. Το μοντέλο τους δέχεται 3 εισόδους: την αρχιτεκτονική, την απόδοσή της (π.χ. ακρίβεια ταξινόμησης σε ένα συγκεκριμένο σύνολο δεδομένων) και την υπολογιστική πολυπλοκότητά της αρχιτεκτονικής (π.χ. το πλήθος των υπολογισμών). Μαζί με τον εναλλασσόμενο αυτόματο κωδικοποιητή, εκπαιδεύουν από κοινού και έναν predictor της απόδοσης και έναν predictor της πολυπλοκότητας της αρχιτεκτονικής. Όμως λόγω της χρήση ετικετών από ένα συγκεκριμένο σύνολο δεδομένων, ο λανθάνων χώρος τους δεν μπορεί να εφαρμοστεί για βελτιστοποίηση σε άλλα σύνολα δεδομένων. Χρησιμοποιούν επίσης πίνακες με κωδικοποίηση one-hot για την αναπαράσταση της λειτουργίας των επιπέδων.

Οι Luo et al. [77] πρότειναν την μέθοδο που ονομάζεται Neural Architecture Optimization (NAO). Μαθαίνουν από κοινού έναν αυτόματο κωδικοποιητή μεταξύ των αρχιτεκτονικών και επίσης έναν predictor που προβλέπει την απόδοση μιας αρχιτεκτονικής σε ένα δεδομένο σύνολο δεδομένων, από τη συνεχή της αναπαράσταση στον λανθάνοντα χώρο. Στη συνέχεια, χρησιμοποιούν gradient descent για τη βελτιστοποίηση της απόδοσης του predictor ώστε να βρουν καλύτερες αρχιτεκτονικές στον συνεχή χώρο. Υπάρχουν μερικές σημαντικές διαφορές με την προσέγγισή μας. Προκειμένου να αντιπροσωπεύουν την αρχιτεκτονική χρησιμοποιούν συμβολοσειρές, ενώ εμείς χρησιμοποιούμε μια ένα ακυκλικό κατευθυνόμενο γράφημα, το οποίο εκμεταλλεύεται τις δομικές πληροφορίες της αρχιτεκτονικής. Επιπλέον, χρησιμοποιούν εποπτευόμενη μάθηση για να εκπαιδεύσουν το μοντέλο τους αντί για μη εποπτευόμενη όπως εμείς. Οπότε πρέπει να εκπαιδεύσουν πρώτα και να αξιολογήσουν πολλές αρχιτεκτονικές και να χρησιμοποιήσουν αυτά τα αποτελέσματα στην επίβλεψη της εκπαίδευσης του μοντέλου.

Διατύπωση Προβλήματος

Έστω X ο χώρος εισόδου, ο οποίος αποτελείτε απο γραφήματα τα οποία αναπαριστούν αρχιτεκτονικές νευρωνικών δικτύων, και έστω $f : X \to \mathbb{R}$ η αντικειμενική συνάρτηση, η οποία δίνει ως έξοδο την απόδοση της αρχιτεκτονικής σε ένα συγκεκριμένο σύνολο δεδομένων D. Ο πρώτος στόχος είναι να βρεθεί ένας κωδικοποιητής $q : X \to Z$, ο οποίος αντιστοιχεί σε ένα λανθάνοντα σημείο z_i, κάθε σημείο $x_i \in X$, καθώς και ένας αποκωδικοποιητής $p : Z \to X$, ο οποίος αντιστοιχεί σημεία από τον χώρο Z πίσω στον χώρο X. Ο δεύτερος στόχος είναι να βρεθεί ένα λανθάνοντα αντικειμενικό μοντέλο $h : Z \to \mathbb{R}$, ώστε να προσεγγίσει την αντικειμενική συνάρτηση f στην έξοδο του p, έτσι ώστε να ισχύει $f(p(z)) \approx h(z), \forall z \in Z$. Η έξοδος του συστήματος πρέπει να είναι ο γράφος $x_{sol} = p(z_{sol}) \in X$, έτσι ώστε $z_{sol} = \text{argmax}_z\, h(z)$

Graph Variational Autoencoder with Operation Embeddings

Σε αυτήν την υποενότητα, παρουσιάζουμε τον προτεινόμενο μας εναλλασσόμενο αυτόματο κωδικοποιητή (Graph Variational Autoencoder), για το πρόβλημα της αναζήτησης αρχιτεκτονικής νευρωνικού δικτύου. Ακολουθούμε την τυπική διαδικασία εκμάθησης ενός εναλλασσόμενου αυτόματου κωδικοποιητή [51]. Αρχικά Θέλουμε να μάθουμε ένα πιθανοτικό γενετικό μοντέλο $p_\theta(x|z)$ καθώς και μια προσεγγιστική οπίσθια κατανομή $q_\phi(z|x)$. Αναφερόμαστε στο γενετικό μοντέλο ως *αποκωδικοποιητής* και στην οπίσθια κατανομή ως *κωδικοποιητή*. Το σύστημα εκπαιδεύεται μεγιστοποιώντας την ποσότητα:

$$L(\phi, \theta; x) = \mathbb{E}_{z\sim q_\phi(z|x)}[\log p_\theta(x|z)] - KL[q_\phi(z|x)||p(z)], \tag{1}$$

, η οποία ονομάζεται evidence lower bound (ELBO). Σημειώνεται ότι η εκπαίδευση του μοντέλου γίνεται με μη επιβλεπόμενη μάθηση, καθώς δεν χρησιμοποιούμε κάποια ετικέτα απόδοσης των αρχιτεκτονικών. Συνεπώς, ο λανθάνοντας χώρος που κατασκευάζουμε μπορεί να χρησιμοποιηθεί για την επίλυση διαφόρων προβλημάτων.

Αναπαράσταση Εισόδου

Κάθε αρχιτεκτονική νευρωνικού δικτύου αντιπροσωπεύει έναν υπολογισμό, ο οποίος εφαρμόζεται σε ένα σήμα εισόδου χρησιμοποιώντας ορισμένες λειτουργίες. Επομένως, μοντελοποιούμε κάθε αρχιτεκτονική ως ένα υπολογιστικό γράφημα (computational graph), με τους κόμβους να αντιστοιχούν στις λειτουργίες και τις κατευθυνόμενες ακμές να αντιπροσωπεύουν τη ροή του σήματος μεταξύ των λειτουργιών. Το υπολογιστικό γράφημα είναι ακυκλικό, προκειμένου να συμβολίζει έναν πεπερασμένο αριθμό λειτουργιών. Έχοντας στην είσοδο μια αρχιτεκτονική, κατασκευάζουμε το αντίστοιχο ακυκλικό γράφημα $G = (V, E)$. Κάθε κόμβος $u \in V$ συσχετίζεται με μια ετικέτα y_u που αντιστοιχεί στη λειτουργία του κόμβου u. Παρατηρήστε ότι δύο ίδιες αρχιτεκτονικές με διαφορετικά βάρη αντιστοιχούν στον ίδιο υπολογισμό, καθώς δεν μοντελοποιούμε τα βάρη μέσα στο γράφημα. Στο γράφημα προσθέτουμε επίσης έναν αρχικό κόμβο χωρίς προκάτοχους, που ονομάζεται κόμβος εισόδου και έναν τελικό κόμβο χωρίς διαδόχους, που ονομάζεται κόμβος εξόδου.

Πίνακες Λειτουργιών (Operation Embeddings)

Μια αρχιτεκτονική νευρωνικού δικτύου μπορεί να θεωρηθεί ως ένα ετερογενές γράφημα λόγω των διαφορετικών ετικετών που έχουν οι κόμβοι του. Κάθε διαφορετική ετικέτα αντιπροσωπεύει και μια διαφορετική λειτουργία. Σε προηγούμενες παρόμοιες εργασίες, οι λειτουργίες παρουσιάζονται με κωδικοποίηση πινάκων one-hot [129]. Όμως η κωδικοποίηση one-hot δεν λαμβάνει υπόψη τις σημασιολογικές σημασίες των διαφορετικών λειτουργιών. Για παράδειγμα, ένα συνελικτικό επίπεδο (convolutional layer) διαστάσεων πυρήνα 5×5 είναι περισσότερο όμοιο με ένα συνελικτικό επίπεδο διαστάσεων πυρήνα 3×3 παρά με ένα επίπεδο υποδειγματοληψίας (pooling). Αυτό οφείλεται στο γεγονός ότι κάθε πίνακας μια λειτουργίας είναι μια ορθογώνια αναπαράσταση σε μια διάσταση και όλες οι λειτουργίες ισαπέχουν, καθώς δεν υπάρχει συγκεκριμένη διάταξη.

Καθώς ο στόχος μας είναι να μάθουμε έναν ομαλό λανθάνοντα χώρο σε σχέση με την απόδοση κάθε αρχιτεκτονικής, είναι απαραίτητο να χαρτογραφήσουμε παρόμοιες αρχιτεκτονικές την μία κοντά στην άλλη. Διαισθητικά, παρόμοιες αρχιτεκτονικές έχουν παρόμοια δομή γραφήματος και επίσης παρόμοιες λειτουργίες στους κόμβους τους. Επομένως, προκειμένου να καταγράψουμε την ομοιότητα μεταξύ των λειτουργιών, αλλάζουμε την one-hot αναπαράσταση που έχουν. Εμπνευσμένοι από την επιτυχία των πυκνών πινάκων για τις λέξεις (word embeddings) [80], προτείνουμε πυκνούς πίνακες μικρής διαστατικότητας για τις λειτουργίες operation embeddings. Τα operation embeddings είναι πυκνοί πίνακες χαμηλής διάστασης, που μπορούν να συλλάβουν τους συσχετισμούς μεταξύ των λειτουργιών. Μια οπτικοποίηση τέτοιων πινάκων φαίνεται στο σχήμα 3.3. Για να εκπαιδεύσουμε αυτούς τους πίνακες, τους αντιμετωπίζουμε ως παραμέτρους του αυτόματου κωδικοποιητή και τις βελτιστοποιούμε με την τεχνική gradient descent, μαζί με τις υπόλοιπες παραμέτρους τους συστήματος. Αναλύουμε τα οφέλη από τη χρήση των operation embeddings παρακάτω:

- Το σημαντικότερο πλεονέκτημα είναι ότι έχουμε παρατηρήσει έναν πιο ομαλό λανθάνοντα χώρο, από τις προσεγγίσεις με κωδικοποίηση one-hot, πιθανώς λόγω του γεγονότος ότι το μοντέλο μας μαθαίνει τις σχέσεις και τις ομοιότητες μεταξύ των λειτουργιών. Αυτή η ομαλότητα (smoothness) μπορεί να επηρεάσει σε μεγάλο βαθμό τη βελτιστοποίηση στον λανθάνοντα χώρο, καθώς οι αρχιτεκτονικές παρόμοιας απόδοσης τείνουν να εντοπίζονται η μια κοντά στην άλλη, με αποτέλεσμα η βελτιστοποίηση σε έναν τέτοιο ομαλό χώρο να είναι ευκολότερη.

- Επιπλέον, έχουμε μια μείωση στην κατανάλωσης της μνήμης που χρησιμοποιεί το μοντέλο. Αν θεωρήσουμε ως n τον αριθμό των διαφορετικών λειτουργιών, τότε σε μια one-hot κωδικοποίηση, απαιτείται ένας πίνακας με n στοιχεία, για να αντιπροσωπεύει μια συγκεκριμένη λειτουργία. Στην προτεινόμενη μας κωδικοποίηση (operation embeddings), απαιτούμε μόνο d στοιχεία για να αντιπροσωπεύσουμε μια λειτουργία κόμβου, όπου d είναι η διάσταση των embeddings , με $d < n$. Η διάσταση των embeddings είναι μικρότερη από τη διάσταση των διανυσμάτων one-hot, καθώς οι δικοί μας πίνακες είναι πυκνοί, σε αντίθεση με τα διανύσματα one-hot, ώστε να μπορούν να κωδικοποιούν πληροφορίες σε λιγότερα bits.

- Επίσης, παρατηρήσαμε μια επιτάχυνση στη σύγκλιση του αυτόματου κωδικοποιητή κατά την εκπαίδευση του. Το μοντέλο έμαθε να ανακατασκευάζει με ακρίβεια τα γραφήματα εισόδου, σε λιγότερες εποχές εκπαίδευσης. Διαισθητικά, μαθαίνοντας τις σχέσεις μεταξύ των λειτουργιών, το μοντέλο μας μαθαίνει αυτές τις επιπλέον πληροφορίες, οι οποίες απουσιάζουν όταν το μοντέλο εκπαιδεύεται με μια κωδικοποίηση one-hot. Επομένως, μπορεί να συγκλίνει σε λιγότερες εποχές και με μεγαλύτερη ακρίβεια. Όλα αυτά τα συμπεράσματα, τα επιβεβαιώνουμε και με πειράματα, τα οποία παρουσιάζονται σε επόμενη ενότητα.

Κωδικοποιητής (Encoder)

Το πρώτο μέρος του μοντέλου μας είναι ο κωδικοποιητής encoder. Ο κύριος στόχος του κωδικοποιητή είναι να μάθει μια αντιστοίχηση του γράφήματος της αρχιτεκτονικής σε ένα σημείο σε έναν συνεχή λανθάνοντα χώρο, έτσι ώστε παρόμοια γραφήματα να χαρτογραφούνται το ένα κοντά στο άλλο. Η διαδικασία κωδικοποίησης μπορεί να συνοψιστεί στα ακόλουθα δύο βήματα:

- Ένα νευρωνικό δίκτυο για γραφήματα (Graph Neural Network), το οποίο χρησιμοποιεί το ασύγχρονο σχήμα μετάδοσης μηνυμάτων που προτείνεται από τους Ζηανγ κ.ά. [129], δημιουργεί μια διακριτή αναπαράσταση h_G του γραφήματος G.
- Δύο νευρωνικά δίκτυα πρόσθιας τροφοδότησης (Feedforward Neural Networks) λαμβάνουν ως είσοδο τη διακριτή αναπαράσταση h_G της αρχιτεκτονικής και παράγουν το μέσο όρο μ και τη διακύμανση σ^2 της κατανομής $q_\phi(z|G)$.

Το νευρωνικό δίκτυο για γραφήματα, ανταλλάσσει πληροφορίες μεταξύ των κόμβων ακολουθώντας την τοπολογική διάταξη του γραφήματος, προκειμένου τελικά να παράγει για κάθε κόμβο u μια κρυφή αναπαράσταση h_u. Στη συνέχεια, χρησιμοποιούμε την κρυφή κατάσταση h_{u_n} του τελικού κόμβου ως την αναπαράσταση ολόκληρου του γραφήματος h_G. Για να δημιουργήσουμε αυτές τις αναπαραστάσεις, χρησιμοποιούμε την ακόλουθη διαδικασία. Πρώτον, για κάθε κόμβο u συγκεντρώνουμε τα μηνύματα από τους προκατόχους γείτονές του, χρησιμοποιώντας μια συνάρτηση συνάθροισης (aggregation function)A,

$$h_u^{in} = A(\{Concat(h_v, x_v) : (v \to u)\}) \tag{2}$$

, όπου x_v είναι το **operation embedding** του κόμβου v. Δεύτερον, ενημερώνουμε την αναπαράσταση κάθε κόμβου u, με βάση το εισερχόμενο συγκεντρωτικό μήνυμα από τους γείτονές του και το operation embedding x_u του κόμβου u, χρησιμοποιώντας μια συνάρτηση ενημέρωσης U,

$$h_u = U(h_u^{in}, x_u). \tag{3}$$

Καθώς ο στόχος μας είναι να κάνουμε βελτιστοποίηση στο λανθάνοντα χώρο, ο κωδικοποιητής μας πρέπει να αντιστοιχεί διαφορετικές αρχιτεκτονικές σε διαφορετικές αναπαραστάσεις h_G. Για να το επιτύχει αυτό, η συνάρτηση συνάθροισης A και η συνάρτηση ενημέρωσης U πρέπει να είναι [129]. Επίσης, θέλουμε ο κωδικοποιητής μας να είναι αμετάβλητος στις μεταθέσεις των κόμβων, έτσι ώστε τα ισομορφικά γραφήματα, που αντιπροσωπεύουν την ίδια αρχιτεκτονική, να χαρτογραφούνται στην ίδια αναπαράσταση. Για να το επιτύχουμε αυτό, η συνάρτηση συνάθροισης πρέπει να είναι αμετάβλητη στις μεταθέσεις [129]. Για να μοντελοποιήσουμε αυτές τις δύο συναρτήσεις, χρησιμοποιούμε βαθιά νευρωνικά δίκτυα [41]. Ακολουθούμε την μεθοδολογία από το Zhang et al. [129] και χρησιμοποιούμε ένα περιφραγμένο άθροισμα (gated sum) ως συνάρτηση συνάθροισης:

$$h_u^{in} = \sum_{u \to v} g(Concat(h_v, x_v)) \odot m(Concat(h_v, x_v)). \tag{4}$$

Αυτό το περιφραγμένο άθροισμα ικανοποιεί τις παραπάνω ιδιότητες [121]. Ως συνάρτηση ενημέρωσης, χρησιμοποιούμε ένα gated reccurent unit (GRU) για να καταγράψουμε τις εξαρτήσεις μεταξύ των επιπέδων:

$$h_u = GRU(x_u, h_u^{in}) \tag{5}$$

Αποκωδικοποιητής (Decoder)

Ο αποκωδικοποιητής μας δημιουργεί γραφήματα από τις λανθάνουσες αναπαραστάσεις με αυτοπαλινδρομούμενο (autoregressive) τρόπο και ακολουθεί τις ίδιες τεχνικές με τον κωδικοποιητή. Αρχικά, χρησιμοποιούμε ένα νευρωνικό δίκτυο εμπρόσθιας τροφοδότησης, προκειμένου να αντιστοιχίσουμε τη συνεχή αναπαράσταση z στην αρχικό κρυφή αναπαράσταση του γραφήματος h_0. Στη συνέχεια, χρησιμοποιούμε ένα gated reccurent unit, προκειμένου να ενημερώσουμε τις κρυφές καταστάσεις και να δημιουργήσουμε διαδοχικά τους κόμβους και τις ακμές. Τα βήματα της δημιουργίας του i^{th} κόμβου συνοψίζονται στα ακόλουθα βήματα:

1. **Υπολογισμός της κατανομής λειτουργίας κόμβου:** Τροφοδοτούμε την αναπαράσταση $h_{u_{i-1}}$ του τελευταίου κόμβου u_{i-1} , σε ένα νευρικό δίκτυο εμπρόσθιας τροφοδότησης που ακολουθείται από μια συνάρτηση softmax , για να προσδιοριστεί η πιθανότητα κάθε λειτουργίας για τον κόμβο.

2. **Δημιουργία κόμβου:** Επιλέγουμε την λειτουργία με την υψηλότερη πιθανότητα και δημιουργούμε έναν κόμβο με αυτήν την λειτουργία. Εάν η λειτουργία είναι ο τύπος λήξης, σταματάμε την αποκωδικοποίηση, συνδέουμε τους κόμβους χωρίς διάδοχους στο u_i και εξάγουμε το κατευθυνόμενο γράφημα, αλλιώς συνεχίζουμε τη δημιουργία των κόμβων.

3. **Ενημέρωση κατάστασης κόμβου** u_i**:** Ενημερώνουμε την κρυφή κατάσταση του νέου κόμβου που δημιουργήθηκε σύμφωνα με την σχέση $h_{u_i} = GRU(x_{u_i}, h_{u_i}^{in})$, όπου x_{u_i} είναι το operation embedding της λειτουργίας του κόμβου και το $h_{u_i}^{in}$ είναι το συγκεντρωτικό μήνυμα από τους προκατόχους του, όπως περιγράψαμε στη διαδικασία κωδικοποίησης.

4. **Δημιουργία ακμών:** Μετά τη δημιουργία του κόμβου u_i, πρέπει να αποφασίσουμε ποιοι από τους κόμβους που έχουν ήδη δημιουργηθεί θα συνδεθούν με τον νέο κόμβο. Επομένως, για κάθε ζεύγος κόμβων $(u_j, u_i), j < i$, τροφοδοτούμε τις αναπαραστάσεις των κόμβων h_{u_j}, h_{u_i} σε ένα νευρικό δίκτυο εμπρόσθιας τροφοδότησης που ακολουθείται από μια softmax συνάρτηση, ώστε να υπολογίσουμε την πιθανότητα της ακμής (u_j, u_i). Εάν η πιθανότητα είναι μεγαλύτερη από $0,5$, τότε προσθέτουμε αυτήν τη σύνδεση και ενημερώνουμε την κρυφή κατάσταση του κόμβου u_i χρησιμοποιώντας το τρίτο βήμα.

Διαδικασία Εκπαίδευσης

Βελτιστοποιούμε όλο το σύστημα χρησιμοποιώντας κλίση κατάβασης (gradient descent). Ο στόχος της εκπαίδευσης είναι η μεγιστοποίηση της συνάρτησης ELBO 3.1. Για να υπολο-

γίσουμε τον πρώτο όρο της συνάρτησης, δηλαδή το κόστος ανακατασκευής (reconstruction loss), χρησιμοποιούμε την τεχνική teacher forcing [46]. Ακολουθώντας την τοπολογική διάταξη του γραφήματος και αθροίζουμε την αρνητική λογαριθμική πιθανοφάνεια των κόμβων και των ακμών. Μετά τις προβλέψεις σε κάθε βήμα, τις αντικαθιστούμε με τις πραγματικές τιμές των κόμβων και των ακμών. Επομένως, το μοντέλο κάνει προβλέψεις χρησιμοποιώντας το σωστό ιστορικό.

Μπεϋζιανή Βελτιστοποίηση

Μετά την εκπαίδευση του αυτόματου κωδικοποιητή, θέλουμε να δημιουργήσουμε αρχιτεκτονικές υψηλής απόδοσης μέσω μπεϋζιανής βελτιστοποίησης στον λανθάνοντα χώρο. Στόχος μας ήταν να κατασκευάσουμε έναν ομαλό λανθάνοντα χώρο σε σχέση με την ακρίβεια, προκειμένου να κάνουμε τη βελτιστοποίηση ευκολότερη, καθώς οι αρχιτεκτονικές με τις καλύτερες επιδόσεις χαρτογραφούνται η μια κοντά στην άλλη. Η διαδικασία βελτιστοποίησης περιγράφεται ως εξής:

- Ορίζουμε μια γκαουσιανή διαδικασία [102] ώστε να μοντελοποιήσουμε την πραγματική αντικειμενική συνάρτηση f, δηλαδή την απόδοση της αρχιτεκτονικής.
- Ακολουθώντας την προσέγγιση του [59], ορίζουμε το κριτήριο αναμενόμενης βελτίωσης (EI) ως συνάρτηση απόκτησης (acquisition function). Επιλέγουμε την επόμενη αρχιτεκτονική ως αυτήν που έχει την υψηλότερη αναμενόμενη βελτίωση σε σχέση με το τρέχον μέγιστο.
$$EI(z) = \mathbb{E}(max(f(z) - f(z^+), 0)), \tag{6}$$
όπου $f(z^+)$ είναι το τρέχον μέγιστο και z^+ είναι το σημείο έτσι ώστε, $z^+ = \text{argmax}_{z_i \in z_{1:t}} f(z_i)$
- Ενημερώνουμε τα μοντέλα μας, με βάση την απόδοση των επιλεγμένων αρχιτεκτονικών και συνεχίζουμε τις επαναλήψεις.
- Τέλος, επιλέγουμε τις αρχιτεκτονικές με την καλύτερη απόδοση.

Πειράματα

Στην ενότητα αυτή παρουσιάζουμε τα πειράματα που πραγματοποιήσαμε και συγκρίνουμε τα αποτελέσματα του μοντέλου μας, με παρόμοιες εργασίες. Για να έχουμε μια δίκαιη σύγκριση με άλλες προσεγγίσεις, ακολουθούμε το πειραματικό σχήμα των [59, 129] και εκτελούμε τα ακόλουθα πειράματα:

- **Βασικές ικανότητες του αυτόματου κωδικοποιητή:** Πραγματοποιούμε δοκιμές για να αξιολογήσουμε τις ικανότητες ανοικοδόμησης και δημιουργίας νέων γραφημάτων του αυτόματου εναλλασσόμενου κωδικοποιητή.

- **Προγνωστική απόδοση λανθάνουσας αναπαράστασης:** Εξετάζουμε την ικανότητα να προβλέψουμε την απόδοση της αρχιτεκτονικής, από την αναπαράσταση της στον λανθάνοντα χώρο.
- **Απόδοση των δημιουργούμενων αρχιτεκτονικών:** Εξετάζουμε την απόδοση των αρχιτεκτονικών που δημιουργήθηκαν από την μπεϋζιανή βελτιστοποίηση στον λανθάνοντα χώρο.
- **Οπτικοποίηση λανθάνοντα χώρου:** Οπτικοποιούμε τον λανθάνοντα χώρο σε τρεις διαστάσεις προκειμένου να δούμε κατά πόσο παρόμοιας απόδοσης αρχιτεκτονικές, έχουν αντιστοιχηθεί σε κοντινές αναπαραστάσεις.

Συγκρίνουμε το μοντέλο μας με τις παρακάτω πέντε βασικές παρόμοιες εργασίες προσαρμοσμένες για κατευθυνόμενα ακυκλικά γραφήματα: D-VAE [129], NGAE [63], S-VAE [11], GraphRNN [126] και GCN [**κιπφ2017σεμισυπερισιον**]. Λόγω περιορισμένων υπολογιστικών πόρων, χρησιμοποιούμε τα αναφερόμενα αποτελέσματα για αυτά τα βασικά μοντέλα από την βιβλιογραφία [129, 63], και τρέχουμε τα πειράματα μόνο για το μοντέλο μας.

Σύνολο Δεδομένων

Το σύνολο δεδομένων εκπαίδευσης αποτελείται από 19.020 αρχιτεκτονικές νευρωνικών δικτύων από το λογισμικό ENAS [87]. Κάθε αρχιτεκτονική έχει 6 κόμβους (επίπεδα) εκτός από τον κόμβο εισόδου και τον κόμβο εξόδου. Κάθε κόμβος σχετίζεται με μια λειτουργία. Υπάρχουν έξι διαθέσιμες λειτουργίες: 3×3 και 5×5 convolution, 3×3 και 5×5 depthwise separable convolution, 3×3 max pooling και 3×3 average pooling. Χρησιμοποιούμε το 90 % του συνόλου δεδομένων ως δεδομένα εκπαίδευσης και το υπόλοιπο 10 % μόνο για αξιολόγηση. Λόγω των περιορισμένων υπολογιστικών πόρων μας, αξιολογούμε την απόδοση κάθε αρχιτεκτονικής χρησιμοποιώντας την τεχνική weight sharing [87] που αποτελεί προσέγγιση της πραγματικής απόδοσης, στο CIFAR-10 [54]. Ως μελλοντική εργασία, θέλουμε να δοκιμάσουμε το μοντέλο μας χρησιμοποιώντας το σύνολο δεδομένων NAS-Bench-101 [124] που περιέχει 423 χιλιάδες μοναδικές εκπαιδευμένες αρχιτεκτονικές με την απόδοση τους στο CIFAR10. Ένα άλλο σύνολο δεδομένων που μπορεί να χρησιμοποιηθεί είναι το NAS-BENCH-201 [25] το οποίο αποτελεί επέκταση του NAS-Bench-101 και περιέχει αρχιτεκτονικές που έχουν εκπαιδευτεί στο CIFAR10, CIFAR100 [55] και στο Imagenet [57].

Βασικές ικανότητες του αυτόματου κωδικοποιητή

Η ικανότητα να ανακατασκευάσουν με ακρίβεια τα δεδομένα εισόδου και να δημιουργήσουν νέα μοναδικά δείγματα, αποτελούν σημαντικές ιδιότητες για τα μοντέλα αυτόματου κωδικοποιητή. Επομένως, για να αξιολογήσουμε αυτές τις ιδιότητες, χρησιμοποιούμε τις εξής μετρικές:

- **Ακρίβεια (Accuracy)**. Μετρά το ποσοστό των τέλεια ανακτημένων γραφημάτων.
- **Εγκυρότητα (Validity)**. Μετρά το ποσοστό των έγκυρων αρχιτεκτονικών νευρωνικών δικτύων.

- **Μοναδικότητα (Uniqueness)**. Δείχνει την αναλογία των μοναδικών γραφημάτων από τα έγκυρα γραφήματα.

- **Καινοτομία (Novelty)**. Μετρά την αναλογία των νέων γραφημάτων από τα έγκυρα γραφήματα που δεν υπήρξαν ποτέ στο σύνολο εκπαίδευσης.

Παρουσιάζουμε τα αποτελέσματα στον Πίνακα 4.1. Το μοντέλο μας ξεπερνά τα άλλα σε όλες τις κατηγορίες. Η ακρίβεια της ανακατασκευής μας είναι κοντά στο 100%, επομένως οι αναπαραστάσεις από τον λανθάνοντα χώρο χαρτογραφούνται με ακρίβεια στις αρχικές τους δομές. Το D-VAE έχει μικρότερη ακρίβεια ανοικοδόμησης, επειδή η one-hot αναπαράσταση τους δεν καταγράφει με ακρίβεια τις πληροφορίες λειτουργίας.

Στα διαγράμματα 4.1, 4.2 και 4.3, παρουσιάζουμε την σύγκλιση των μοντέλων κατά την διάρκεια της εκπαίδευσης. Παρατηρούμε ότι η απώλεια ανακατασκευής του αυτόματου κωδικοποιητή μας μπορεί να συγκλίνει σε πολύ λιγότερες εποχές. Διαισθητικά, το μοντέλο μας χρησιμοποιώντας τα operation embeddings, μαθαίνει επιπλέον πληροφορίες καταγράφοντας τις σχέσεις μεταξύ των λειτουργιών. Αυτές οι σχέσεις δεν μπορούν να αξιοποιηθούν στο μοντέλο D-VAE, το οποίο χρησιμοποιεί one-hot πίνακες για την αναπαράσταση των λειτουργιών. Επομένως, το μοντέλο μας μπορεί να συγκλίνει σε λιγότερες εποχές και με μεγαλύτερη ακρίβεια.

Προγνωστική απόδοση λανθάνουσας αναπαράστασης

Σε αυτό το πείραμα, εξετάζουμε πόσο αντιπροσωπευτικές είναι οι αναπαραστάσεις στον λανθάνοντα χώρο, για την απόδοση της αρχιτεκτονικής του νευρικού δικτύου. Εάν μπορούμε να προβλέψουμε με ακρίβεια την απόδοση, από τις αναπαραστάσεις του λανθάνοντος χώρου, τότε μπορούμε επίσης να ανακαλύψουμε εύκολα τις καλύτερες αρχιτεκτονικές από τον λανθάνοντα χώρο. Χρησιμοποιούμε δύο μετρικές αξιολόγησης, το Root Mean Square Error (RMSE) μεταξύ των προβλέψεων και των πραγματικών επιδόσεων και του συντελεστή συσχέτισης Pearson. Ο συντελεστής συσχέτισης Pearson μετρά τη γραμμική συσχέτιση μεταξύ των προβλέψεων και των πραγματικών επιδόσεων. Επομένως, ένα μοντέλο για να έχει καλές προγνωστικές ικανότητες, θα πρέπει να έχει ένα μικρό RMSE και ένα υψηλό Pearson's r Δείχνουμε τα αποτελέσματα στον Πίνακα 4.2. Το προτεινόμενο μοντέλο μας ξεπερνά τους άλλους αυτόματους κωδικοποιητές, και στις δύο μετρικές. Αυτό δείχνει ότι ο λανθάνοντα χώρος μας είναι πιο κατάλληλος για αναζήτηση μιας αρχιτεκτονικής υψηλής απόδοσης.

Απόδοση των δημιουργούμενων αρχιτεκτονικών

Σε αυτό το πείραμα, πραγματοποιούμε μπεϋζιανή βελτιστοποίηση προκειμένου να δημιουργήσουμε αρχιτεκτονικές υψηλής απόδοσης από τον λανθάνοντα χώρο. Αυτό είναι το κύριο κίνητρο της διατριβής μας. Απεικονίζουμε τις κορυφαίες 5 αρχιτεκτονικές που ανακαλύφθηκαν από κάθε μοντέλο στα διαγράμματα 4.4 4.5 και 4.6. Το μοντέλο μας δημιούργησε αρχιτεκτονικές υψηλότερης απόδοσης από τα υπόλοιπα μοντέλα. Η καλύτερη αρχιτεκτονική μας επιτυγχάνει απόδοση ίση με 95,33 %, ενώ η υψηλότερη απόδοση του D-VAE ήταν 94,80 % και η

υψηλότερη ακρίβεια του S-VAE ήταν μόνο 92,79%. Όπως και σε προηγούμενα πειράματα, αυτό δείχνει ότι η μέθοδος των operation embeddings, βοηθά το μοντέλο να κατασκευάσει έναν λανθάνοντα χώρο που είναι πολύ αποτελεσματικός για την αναζήτηση αποδοτικών αρχιτεκτονικών νευρωνικών δικτύων. Επιπλέον, όλες οι κορυφαίες αρχιτεκτονικές μας, επιτυγχάνουν ακρίβεια υψηλότερη από 95%, αποδεικνύοντας ότι το μοντέλο μας μαθαίνει όχι μόνο μια καλή αρχιτεκτονική, αλλά μια κατανομή αρχιτεκτονικών υψηλής απόδοσης.

Οπτικοποίηση του λανθάνοντος χώρου

Σε αυτό το πείραμα απεικονίζουμε τον λανθάνοντα χώρο μας σε δύο διαστάσεις χρησιμοποιώντας την τεχνική μείωσης της διαστατικότητας t-SNE [78]. Παρουσιάζουμε τα αποτελέσματα στα διαγράμματα 4.8, 4.9. Παρατηρούμε, ότι ακόμη και από την 100ή εποχή το μοντέλο μας μπορεί να συγκεντρώσει με ακρίβεια τις αρχιτεκτονικές υψηλής απόδοσης σε κοντινές αναπαραστάσεις. Αυτή είναι η πιο σημαντική ιδιότητα στην εφαρμογή μας, καθώς ένας ομαλός λανθάνων χώρος είναι απαραίτητος για τη βελτιστοποίηση των αρχιτεκτονικών. Αντιθέτως, στον λανθάνοντα χώρο του D-VAE η μετάβαση της ακρίβειας δεν είναι ομαλή. Αυτό υποδηλώνει ότι η μέθοδος των operation embeddings βοηθάει στη διαδικασία αντιστοίχησης παρόμοιων λειτουργιών μαζί και έτσι και στην αντιστοίχησης παρόμοιων αρχιτεκτονικών μαζί.

Επίλογος

Συμπεράσματα

Σε αυτή τη διπλωματική εργασία, προτείναμε έναν εναλλασσόμενο αυτόματο κωδικοποιητή για γραφήματα (Graph Variational Autoencoder) που αντιστοιχεί την αρχιτεκτονική ενός νευρωνικού δικτύου σε έναν συνεχή και ομαλό λανθάνοντα χώρο. Στη συνέχεια, εφαρμόσαμε μπεϋζιανή βελτιστοποίηση στον λανθάνοντα χώρο, προκειμένου να βρούμε αρχιτεκτονικές υψηλής απόδοσης. Για να αναπαραστήσουμε την αρχιτεκτονική ενός νευρωνικού δικτύου, αξιοποιούμε τη γραφηματική δομή της, και την αντιπροσωπεύουμε ως ένα κατευθυνόμενο ακυκλικό γράφημα, με τις ακμές να αντιστοιχούν στην ροή των δεδομένων και τους κόμβους στις λειτουργίες των επιπέδων. Η κύρια συνεισφορά μας είναι ότι χρησιμοποιήσαμε πυκνές αναπαραστάσεις πινάκων (operation embeddings) αντί για one-hot κωδικοποίηση προκειμένου να αναπαραστήσουμε τις λειτουργίες των επιπέδων. Επομένως, το μοντέλο μας είναι σε θέση να μάθει τους συσχετισμούς μεταξύ των λειτουργιών, και τελικά να παράγει έναν πιο ομαλό λανθάνοντα χώρο, χαρτογραφώντας παρόμοιες αρχιτεκτονικές σε κοντινά σημεία στον λανθάνοντα χώρο. Τα πειράματά μας δείχνουν την αποτελεσματικότητα των operation embeddings σε πολλαπλούς τομείς, καθώς βοηθούν στην αναζήτηση καλύτερης αρχιτεκτονικής από τον λανθάνοντα χώρο, και επίσης μειώνουν το χρόνο εκπαίδευσης του μοντέλου και την κατανάλωση μνήμης.

Μελλοντικές Προεκτάσεις

Αυτή η εργασία μπορεί να επεκταθεί σε διάφορες μελλοντικές κατευθύνσεις. Πρώτα απ ́όλα, μπορεί να μελετηθούν οι δομικές ιδιότητες της αρχιτεκτονικής ενός νευρωνικού δικτύου που επηρεάζουν την τελική της απόδοση. Με την προσέγγισή μας, δεν βρίσκουμε απλά την καλύτερη αρχιτεκτονική, αλλά μαθαίνουμε μια κατανομή αρχιτεκτονικών υψηλής απόδοσης, και έτσι μπορούμε να διερευνήσουμε τα χαρακτηριστικά των γραφημάτων αυτών των αρχιτεκτονικών. Τα τρέχοντα ευρήματά μας δείχνουν ότι η σύνδεση των επιπέδων του δικτύου, είναι ένας εξαιρετικά σημαντικός δείκτης της απόδοσής του [117]. Αυτό έρχεται σε αντίθεση με πολλές προσεγγίσεις στον τομέα του neural architecture search, οι οποίες έχουν προκαθορισμένο σύνδεση επιπέδων, και εστιάζουν στην εύρεση των λειτουργιών κάθε επιπέδου. Ελπίζουμε να εμπνεύσουμε περισσότερα έργα τα οποία εστιάζουν στην γραφηματική σκοπιά των νευρωνικών δικτύων, ώστε να βοηθήσουμε την επεξηγηματικότητα, το σχεδιασμό, την αναζήτηση και την δημιουργία καλύτερων αρχιτεκτονικών νευρωνικών δικτύων.

Μια άλλη ενδιαφέρουσα ερευνητική ιδέα είναι η χρήση των operation embeddings όχι μόνο για την σύλληψη των σχέσεων μεταξύ των λειτουργιών, αλλά και την δημιουργία εντελώς νέων λειτουργιών. Επομένως, αντί να γίνεται η αναζήτηση σε ένα προκαθορισμένο σύνολο λειτουργιών, μπορεί να χρησιμοποιηθεί ένας αντίστροφος μετασχηματισμός από τον χώρο των λειτουργιών ώστε να δημιουργηθούν νέες λειτουργίες. Επίσης, μια πιο απλή προσέγγιση προς αυτήν την κατεύθυνση είναι η δημιουργία της αρχιτεκτονικής προσθέτοντας διαδοχικά γραφηματικά μοτίβα αντί για μεμονωμένους κόμβους, θεωρώντας έτσι αυτά τα μοτίβα ως νέα μπλοκ λειτουργιών.

Chapter 1

Introduction

1.1 Motivation

Deep learning has been in the middle of a research outburst the last years and the constant need for even improving architectures requires extensive architecture engineering. The engineer must make several choices about the architecture of the neural network. Among others, he must decide the number of neurons in every layer, the connections between the layers, the operation of every layer, and the activation function. However, all these architectural options are enforced by the human, therefore the model is limited by the human knowledge. Moreover, this procedure is time consuming and usually leads to errors and bugs. Finally, we do not have a grounded theory about the characteristics that lead to high performing architectures.

Neural Architecture Search (NAS) has gained increased attention recently, as a class of approaches that automate the architecture engineering, in order to produce state-of-the-art models for numerous tasks [135, 96]. Neural architecture search can be formulated as an optimization problem over an input space of architectures. This problem is highly challenging due to two specific properties. First of all, the input space is high-dimensional with usually a discrete or non-Euclidean structure like a graph [128] or a sequence [134]. Directly searching an architecture within this space is inefficient given the exponential growth of the search space, as the number of layers or operations increases [77]. Secondly, the objective function which we want to optimize (i.e. the performance of the architecture), is expensive to evaluate, as the cost of a neural network training is high.

1.2 Approach and Contribution

In this Diploma Thesis, in order to tackle the above challenges, we employ latent space optimization techniques [36, 77, 75]. Specifically, we use a graph variational autoencoder in order to map architectures into a continuous and smooth latent space. Then, we perform bayesian optimization in the learned latent space in order to find the best architecture.

The representation of the neural network architecture is crucial for the solution. We

capitalize on the graph structure of the architecture, and represent it as a directed acyclic graph. We use graph neural networks [129, 128] to obtain the representation of the architecture. By modelling an architecture as a directed acyclic graph with operation labels on nodes, we can simulate the forward and the backward propagation that occurs in neural networks, and take advantage of its structural information. The structural information (i.e. the connections between the layers) along with the operation information (i.e. the operation that every layer perform) define the architecture. But in order to produce a smooth latent space, similar architectures should be mapped together, and the operation information must also be exploited. To address this problem, we propose **operation embeddings**. Instead of specifying every operation with one-hot vector encoding, we learn low-dimensional embeddings, in order to take advantage of the correlations between the operations. For instance, a convolutional 5x5 layer is more similar to convolutional 3x3 layer rather than to a pooling layer.

Our findings suggest that the operation embeddings approach encourages the mapping of similar performance architectures to the same regions in the latent space. This makes the latent space smooth with respect to accuracy, which benefits our neural architecture search task.

1.3 Thesis Structure

In Chapter 2 we provide the theoretical background of our work. Specifically, we first introduce the basic concepts of machine learning and then focus on the deep learning techniques, that are most relevant to our work. We also explain the basic concepts of both graph representation learning and neural architecture search.

In Chapter 3 we present the main contribution of our thesis. We first provide a section of related work and then proceed to the problem definition. In the rest chapter, we analyze our graph variational autoencoder model for neural architecture search.

In Chapter 4 we present our experiments and compare the results with other baseline papers.

In Chapter 5 we conclude our thesis, providing a summary of our work, and some future research ideas.

Chapter 2

Theoretical Background

In this chapter, we provide the basic information that is necessary for the reader to understand the contribution of the thesis. We start by presenting the foundations of machine learning, and the most famous deep learning architectures. After, we analyze how deep learning can be used for graph data. In the last section we explore the core aspects of neural architecture search.

2.1 Introduction to Machine Learning

Machine learning is an area of applied statistics that studies algorithms which computer systems can use, in order to estimate functions from data, without explicit instructions. These learning algorithms, can be classified as **supervised** or **unsupervised**, based on the kind of data we use.

Supervised learning algorithms use data points that each one contains features and is associated with a label. That means that every sample is tagged with the answer that we are trying to predict. So for example, labeled image data set with animals would tell the model which images are dogs,cats,etc. Given a specific training set, a supervised learning algorithm try to learn the function that maps the features of the input to the label. In math notation, we denote the input features as x^i and the label as y^i . A data set D contains many data points , so $D = \{(x^i, y^i); i = 1, ..., n\}$. Also , we denote X the input space , and Y the output space. Our goal is to learn a function $f : X \mapsto Y$ using the dataset D in order to $f(x)$ correctly determine the label y of x. Based on the possible values of output space, the supervised learning problems can be grouped into classification and regression problems.

- **Classification** problems have categorical output values. Examples of classification problems are email spam detection [44], image classification [57] and DNA sequence classification [83].
- **Regression** problems have output values that are real numbers. Examples of regression problems are house price prediction [86] and prediction of wind energy production [98].

Unsupervised learning algorithms use data points without an explicit target associated with them. The machine learning algorithm attempt to identify interesting structural properties and extract useful features of the data. Two of the most important unsupervised techniques are **clustering** and **autoencoders**.

- **Clustering** algorithms try to discover clusters of similar samples.
- **Autoencoders** try to learn a representation of the data inputs, usually by performing dimensionality reduction using neural networks. We will explore in depth autoencoders in the next sections , as they are an important aspect of our thesis.

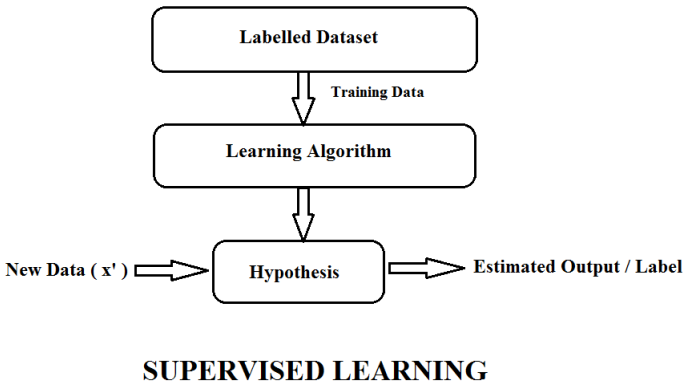


Figure 2.1: Supervised learning pipeline. Source [3]

2.2 The Deep Learning Era

Nowadays, deep learning has achieved major breakthroughs in many fields, including computer vision [38, 56, 91], natural language processing [24, 5, 13] and molecular optimization [133, 27, 30]. These achievements became possible mainly due to the following reasons:

- High availability of massive data sets.
- Increased performance of computer processors, and especially GPUs.
- More complex and deeper neural network architectures.

But before deep learning dominates the field, there was a major drawback in the classical approaches that were used. The feature representations from the input , had to be manually extracted , based on the task. For example, in a computer vision task, the engineer had to manually use specific algorithms that extracts edges,corners,blobs,etc. from images [15]. This unfortunately put up barriers to anyone want to pursue machine learning, as it requires a lot of domain knowledge. In contrast, artificial neural networks have automated the feature extraction procedure. They are composed from many neurons, that are connected to each other, inspired by the functionality of the brain, and can construct features

from raw input. Now, it is not a prerequisite to be an expert in the domain of the problem, in order to use machine learning techniques. Deep learning brought a fundamental change, and enabled more and more people to get involved in the area. In this section we describe common deep learning architectures.

2.2.1 Feed Forward Neural Networks (FFNN)

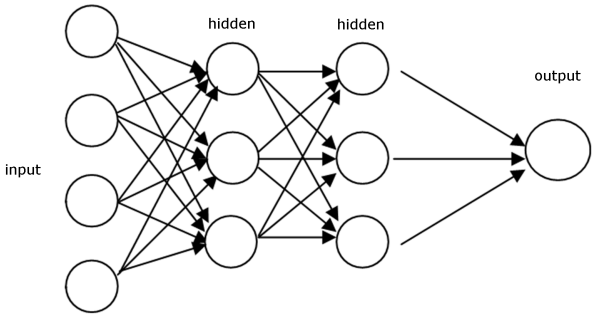


Figure 2.2: Illustration of a FFNN with 2 hidden layers

Feed Forward neural networks or multilayer perceptrons (MLPs) are the most typical example of deep learning models. These networks form a computational graph that is acyclic , so the input flows in one direction, from the input nodes to the output nodes. In order to approximate some function f^* with FFNN , we have n functions $f^{(1)}, f^{(2)}, ..., f^{(n)}$ that are connected such that our prediction f is given by $f(x) = f^{(n)}(f^{(n-1)}(...(f^{(1)}(x))...))$. In this setting, $f^{(1)}$ is called the input layer , $f^{(n)}$ is called the output layer , and the intermediate functions are called hidden layers. By altering these chain structures, we can construct different architecture blocks, such as recurrent neural networks, presented later.

As shown in Figure 2.2, every layer consists of many neurons, that compute the weighted sum of their inputs, and usually followed by a non-linear activation function, such as ReLU [82]. The goal of the model's training procedure is to learn these weights, also called trainable parameters, such that $f(x)$ be as close as possible with the labels $y = f^*(x)$ of every data point x. This learning process demands the computation of the gradients of many functions and is usually computationally expensive. The algorithm to compute these gradients is called back-propagation [95]. Given a dataset that contains many training examples, it is specified how the output layer must act for every example; it should produce a value close to y. But the behavior of the other layers is not specified. The neural network should decide by each own how to use these layers in order to approximate as best as possible the f^*. That's why we often call these hidden layers , as feature extractors. The difference with other machine learning algorithms , is that the neural network learn these features with the hidden layers, instead of manually engineer them. But even in the FFNN , the human effort is not limited. We must design the architecture of the network, including the number of hidden layers and the number of neurons in each

layer.

FFNNs have demonstrated good experimental performance in many tasks, but they also have theoretical guarantees. The universal approximation theorem [41] states that every continuous function from one finite-dimensional space to another, can be approximated arbitrarily closely by a FFNN with at least one hidden layer. Here, an important question may be arise. Given such a continuous function that we want to approximate, are we sure that we will learn it with our training algorithm? The answer is no. Even if the FFNN is able to represent this function, it is not guaranteed, that the optimization algorithm will find the values of the weights, that correspond to the global optimum. Also, our learning algorithm may chose a wrong function due to over fitting [16] in the training set. This is a common situation in deep learning , where our model does not generalize in examples that have not seen in training. To sum up, there is no universal procedure that given some training examples will generalize to samples that are not presented in the training set.

2.2.2 Convolutional Neural Networks (CNN)

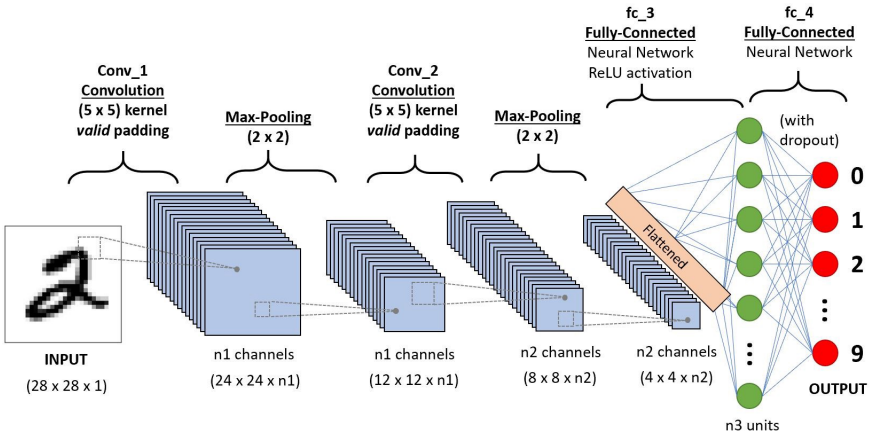


Figure 2.3: Illustration of a CNN. Source [1]

Convolutional neural networks (CNNs) [60] are a specific class of deep neural networks, that is specialized for processing data that have a grid topology, such as images. These networks are considered as one of the greatest biological inspirations in artificial intelligence, as their key design concepts are borrowed from neuroscience, and especially from the organization of the human visual cortex. Individual neurons in the brain fire due to a stimuli, only in a specific region of the visual field. These regions overlap, to cover the entire visual scene. A convolution network layer is designed to capture this property, by applying convolution[1] operations on the input image with many low dimensional filters.

[1]Given two functions f, w, the convolution on f, w is defined as $(f * w)(t) = \int_{-\infty}^{\infty} f(x)g(t-x)\,dx$. The first argument is often referred to as the **input**, the second argument as the **kernel** and the output as the **feature map**.

Motivation

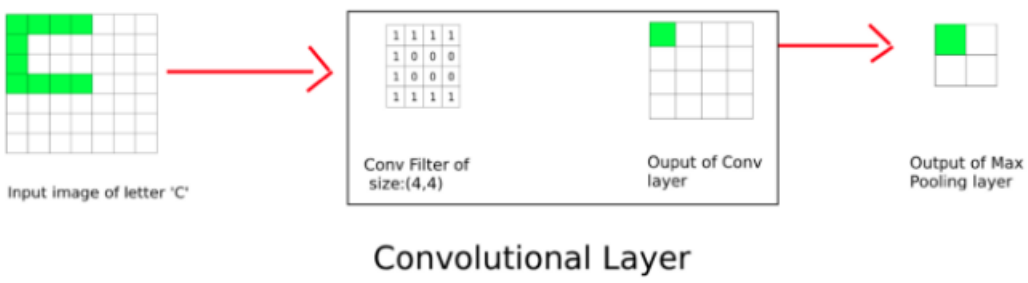
Before we analyze the core functionality of the CNN, it is important to understand the intuition behind them and the drawbacks of the Feed Forward Neural Networks.

In general, a neural network receives an input and transform it through hidden layers. In the FFNN, every hidden layer is composed by a set of neurons where is neuron is fully connected to all neurons in the previous layer. As there is no sharing connections, this number cannot scale very well. For example, assume that we have an image of size $100 * 100 * 3$ (100 wide, 100 high, 3 color channels). Then the first hidden layer has $m * 100 * 100 * 3 = m * 30000$ parameters, where m is the number of neurons in the first hidden layer. It is obvious that if we have multiple layers, that setting cannot scale. This full connectivity leads to very high computational cost and also makes our system prone to overfitting.

In order to reduce the number of parameters, CNN take advantage of the fact that the input consists of images. Their specific topology has 2 interesting properties that we exploit by using convolution operations.

1. **Neighbors pixels are related to each other.** Making a random permutation on the pixels of an image will corrupt the data point. So we want to process together groups of pixels that are close to each other. We put a bias in our architecture, in order to exploit this relation, by applying convolution operations and making the kernel smaller than the input. For example, when processing an image with thousand of pixels, we can detect small features with kernels that are applied only on tens of neighbor pixels.

2. **In most cases, we want to ignore positional shifts and translations of the target in the image.** For example, we want to recognize a dog, whether it appears in the top half or the bottom half of the image. To achieve this translation invariance property, we share the trainable weights along the image and use max pooling layers. By sharing the weights, we can identify the same features along the image, and the max pooling layer takes the output from the convolution layer and reduce its resolution. It does so by keeping only the max value of a grid, so the information about the exact location is discarded.



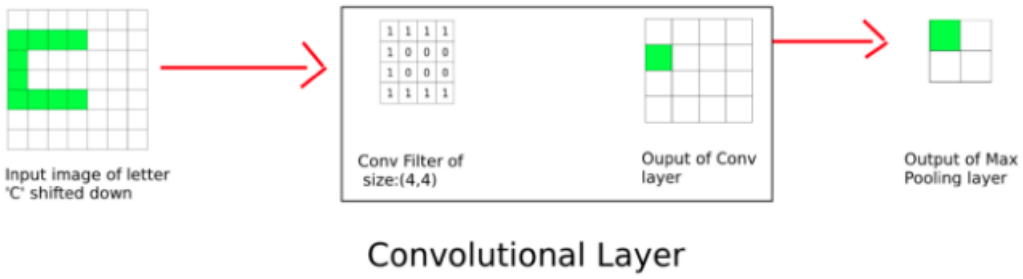


Figure 2.4: CNN filter trained to detect the letter 'C'. Source [108]

As we see in Figure 2.4, the max pooling layer gives the same output for the two images. The CNN is invariant to the shift in the image.

Types of Layers

The most important aspects of a CNN architecture are the convolutional and the pooling layers.

The weights of a **convolution layer** are a set of filters with small width and height, that traverse the image. Most typical sizes of filters are 3x3 and 5x5. Every filter is convolved across the two dimensions of the image (width,height) and outputs a 2-dimensional feature map. By stacking these features maps of every filter we produce the final 3-dimensional feature map. These learned features have usually hierarchical structure. The first convolutional layer capture low-level characteristics of the image, and as we go deeper in the layers, we capture more high-level task specific features. This compositionality can be observed as the features transform, for example, from raw pixels, to edges, to shapes and finally to objects.

Pooling layers are usually placed between following convolutional layers. The role of pooling layer is to reduce the spatial size of the features, in order to limit the weights of the network. As mentioned before, with pooling layers we can extract features that are position invariant, a very useful property. The most well used pooling layers use the max or the average function in order to perform dimensionality reduction, called max pooling and avg pooling respectively. In Figure 2.5 we saw how a max and an average pooling layer operates.

As we have seen, a convolution neural network is a complex structure with many hyperparameters. The engineer must take several choices, regarding the architecture of the network, such as the dimensions of the filters, the function used in the pooling layers and many more that are not covered in this section.

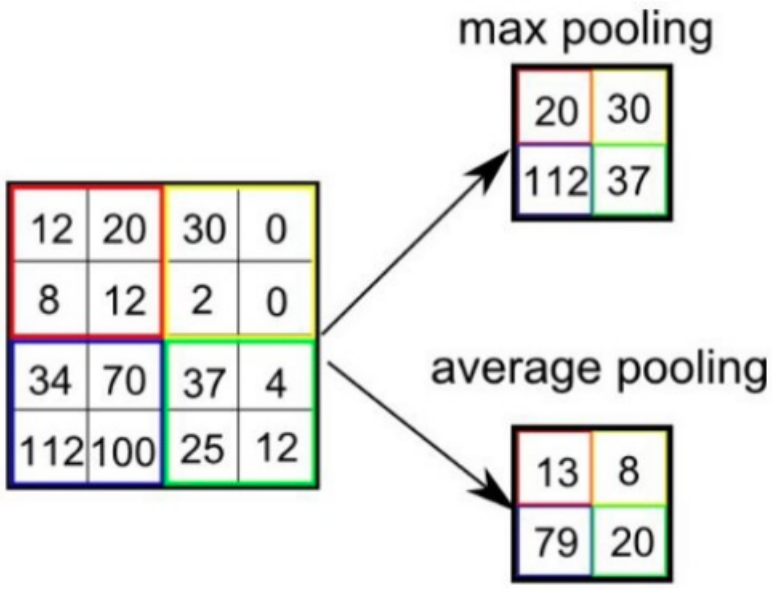


Figure 2.5: Two types of pooling. Source [2]

2.2.3 Recurrent Neural Networks (RNN)

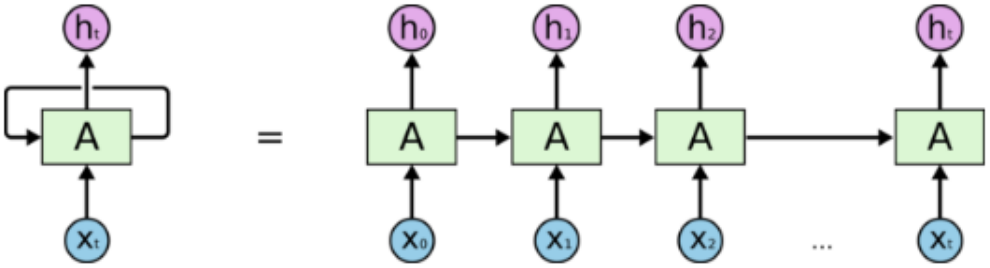


Figure 2.6: An unrolled recurrent neural network. Source [110]

Recurrent neural networks (RNNs) is a specific class of neural networks, that is specialized for processing data that have a temporal relation, such as text. These networks have internal memory state in order to process variable length sequence data. In contrast with feed forward neural networks, that process all the inputs independent of each other, in RNNs the inputs are related to each other. These networks have internal loops , and use a recurrent function in order to produce the output from the input sequence. In every step t , the output o_t is a function of the input x_t and the previous state s_{t-1} . Specifically, in math notation we have:

$$s_t = a_1(W_1 \cdot x_t + W_2 \cdot s_{t-1} + b_1) \tag{2.1}$$

$$o_t = a_2(W_3 \cdot s_t + b_2) \tag{2.2}$$

where a_1 and a_2 are some activation functions, W_1 , W_2 , W_3 are the weight matrices, and $b1$, b_2 are the bias vectors. As we mentioned, RNN can process sequence data with arbitrary length. But as the size goes bigger, it is difficult to maintain the information from the earliest steps [9], due to the vanishing gradient problem [39]. Some gradients shrink when they propagate many steps in the past, so the trainable weights are not changing their values, and eventually this part of the network does not learn anything useful. Usually this occurs in the first layers of the network, so in long sequences, RNN may forget the first part of the input. In order to face this problem, many approaches has been proposed, with Long Short Term Memory [40] and Gated Recurrent Unit [17, 18] be the most famous.

Long Short Term Memory

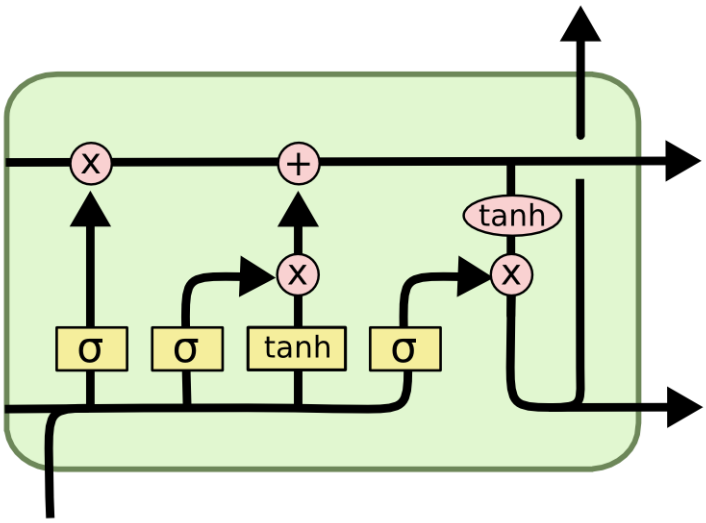


Figure 2.7: LSTM cell. Source [109]

This special class of RNN, can face the vanishing gradient problem, and eventually it is able to learn long term dependencies [104, 32, 43]. The key concept of LSTMs is the cell state, denoted as c_t. In math notation we have the following equations that describe the LSTM network:

$$f_t = \sigma(W_f \cdot [x_t, h_{t-1}] + b_f) \tag{2.3}$$

$$i_t = \sigma(W_i \cdot [x_t, h_{t-1}] + b_i) \tag{2.4}$$

$$\overline{C_t} = tanh(W_c \cdot [h_{t-1}, x_t] + b_c) \tag{2.5}$$

$$C_t = f_t \cdot C_{t-1} + i_t \cdot \overline{C_t} \tag{2.6}$$

$$o_t = \sigma(W_o \cdot [x_t, h_{t-1}] + b_o) \tag{2.7}$$

$$h_t = o_t \cdot tanh(C_t) \tag{2.8}$$

These equations are explained below:

- Equation 2.3: First of all, we must decide which information should be discarded from the cell state(as it has limited memory), based on the current input and the previous state. The output is a real number in range $[0, 1]$. As close the output is to 1, the more information we will keep. We call this gate as *forget* gate.
- Equation 2.4: In the next step, we must decide which values we will update, same as with the conventional RNN. We call this gate as *input* gate.
- Equation 2.5: Next, we must create the new information that will be added to the state.
- Equation 2.6: We update the current state, based on the new information and the discarded information from equation 2.3
- Equations 2.7, 2.8: Finally, we update the hidden state and the output of the network

Gated Recurrent Unit

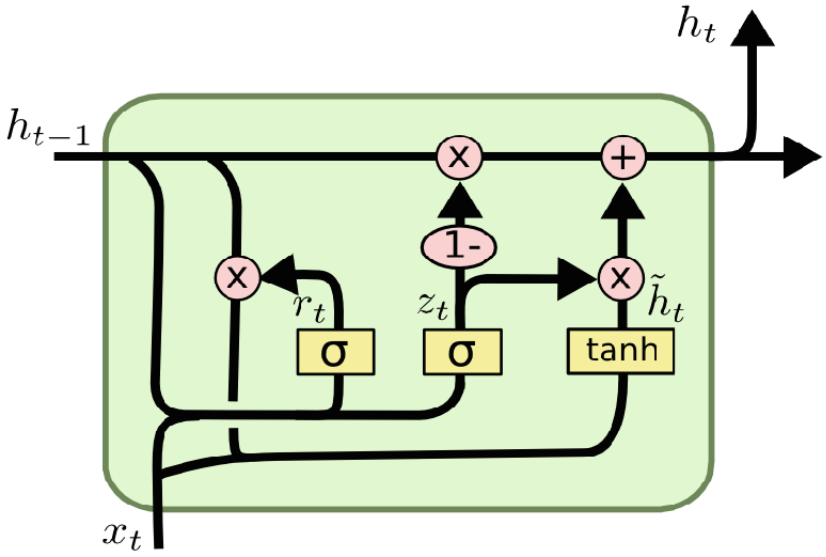


Figure 2.8: GRU cell. Source [92]

A variation of the LSTM network, is the Gated Recurrent Unit, which has fewer parameters but achieves competitive performance in several tasks [18, 130, 81]. It combines the forget and the input gates into one gate, called update gate. It also merge the hidden

state and the cell state, producing a simplified model. The following equations describe the GRU:

$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t]) \tag{2.9}$$

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t]) \tag{2.10}$$

$$\overline{h_t} = tanh(W \cdot [r_t * h_{t-1}, x_t]) \tag{2.11}$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \overline{h_t} \tag{2.12}$$

These equations are explained below:

- Equation 2.9: Same as in the LSTM network, we must decide which information should be passed from the previous steps. We now refer to this gate now as *update* gate.
- Equation 2.10: We must also decide, how much of the past information must be discarded. We refer to this gate as *reset* gate.
- Equation 2.11, 2.12: Finally, we update the hidden state , based on the update gate and the reset gate.

2.3 Graph Representation Learning

Graphs are fundamental mathematical structures which describe relations between objects. They play an important role in many fields, and they have numerous applications. For instance, in medicine, drug discovery relies on multiple types of interactions between proteins, that can be modeled by graphs. In computer vision, 3D objects are represented as graphs. As a consequence, more and more graph-structured data are being produced and are available to the community [62, 85]. For this reason, there is an increasing attention in applying deep learning techniques on graphs. Before we analyze how neural networks can handle these type of data, it is necessary to give some formal definitions on graphs.

2.3.1 Graph Basics

A graph $G = (V, E)$ is comprised of a set of nodes V and a set of edges E. An edge $e \in E$ is associated with two distinct nodes $u, v \in V$, and we denote it as $e = (u, v)$. The most famous way to represent a graph is through an adjacency matrix $A \in \mathbb{R}^{|V|\times|V|}$. To construct this matrix, we correspond every node index to a particular row and column and then for every edge $(u, v) \in E$, we assign $\mathbf{A}[u, v] = 1$. We assign 0 to the rest values. In some graphs, the edges have weights, so we assign the value of the weight in the adjacency matrix. If the edges have no direction, i.e.,$(u, v) \in E \iff (v, u) \in E$, then the graph is called undirected.

A very important class of graphs are the **heterogeneous** graphs. In this setting, we have different types of nodes. For instance, in a recommendation system we might want to diverse the users and the movies. To represent this graph, we construct an adjacency tensor $A \in \mathbb{R}^{|V|\times|R|\times|V|}$, where R is the set of node types.

Computational Graphs

A computation graph is a directed graph where each node correspond to variable or mathematical operation. Following the edges of the graph, variables flow through and they are given as input to the operations. With these graphs, we can represent functions as complex as we want, and visualize them. An example of this type of graph is presented on Figure 2.9. Nodes x, y, z correspond to variables, and nodes $+, *$ correspond to the operations.

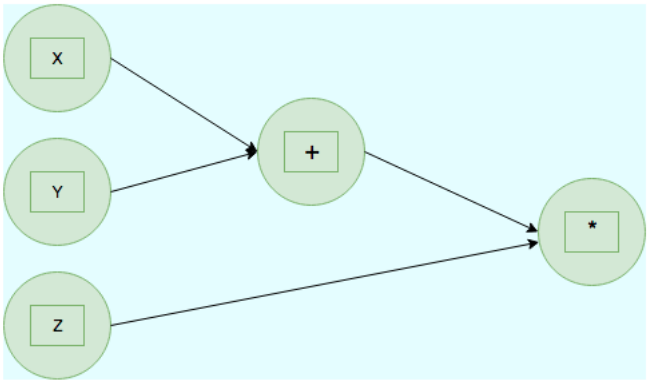


Figure 2.9: A computation graph that represent the function $f(x, y, z) = (x + y) * z$. Source [22]

2.3.2 Machine Learning on Graphs

Plenty of machine learning applications require a graph representation of the data [47, 4], but the tasks differ from problem to problem. Below we present the most important of them.

- **Node Classification**: It is probably the most famous task on graph data. Given a graph $G = (V, E)$ and labels on a set of nodes $V_{train} \in V$, our goal is to predict the labels of the rest nodes. Examples of node classification problems are finding malicious accounts on social networks [74] and classifying a paper given a citation graph [93]. Essentially, we assume relations between the nodes, in order to find the missing labels. These relations can exploit homophily [79], which states that nodes have high probability to share characteristics with their neighbors. Another concept is structural equivalence [97], which is the the tendency of nodes that have similar neighborhoods structures, to share similar labels.

- **Link Prediction**: Link prediction or relation prediction is the problem of predicting the missing interactions between the nodes. Given a graph $G = (V, E)$ and a set of edges $E_{train} \in E$, our goal is to predict the missing edges $E \setminus E_{train}$. Examples of link prediction include predicting co-authorship links in a citation graph [68] and predicting protein-protein interactions in biological processes [10]. The task can also

have temporal orientation, where, we want to predict how the edges of the graph will change in the next time step [26]. Both node classification and link prediction use a supervised signal in order to predict the missing information, so they considered as supervised problems.

- **Community Detection**: Community detection is a variation of the unsupervised clustering problem. Given an input graph $G = (V, E)$, we want to find communities of clusters of nodes, with many edges between the clusters [31, 112]. Real world application of community detection range from social networks [100] to biological networks [48] and citation networks [94]

- **Graph Classification**: It is associated with the node classification problem, but instead of making predictions about individual nodes, we make predictions over entire graphs. Therefore, our input contains multiple graphs with their labels. Note that the label now is associated with the entire graph. Common applications involve predicting properties of molecular graphs [19] or predicting the accuracy of a neural network architecture [76].

All of these tasks pose a difficulty, as the most learning algorithms operate on vectors, due to the Euclidean underlying structure of most signals, like images. It is a great challenge to apply these methods on non-Euclidean data like graphs. To address this problem, explicitly techniques are used, that can operate on graphs. Their central goal is to comprise the structure and the features of the graph into vectors. Before we proceed to the models used in deep learning on graphs, we mention some classical methods.

Traditional Approaches

Traditional approaches in order to obtain information from graphs, use the standard pipeline that was dominated prior to the rise of deep learning. In start, we extract some handcrafted features based on the graph statistics [69] or by using kernel functions [111]. Then we feed these features as input to a machine learning classifier in order to make the final prediction. These methods have the same drawbacks that we mentioned in Section 2.2. They require high domain expertise, they are time-consuming and their performance is not as good as their deep learning counterparts.

2.3.3 Graph Neural Networks (GNN)

Why traditional deep neural network architectures do not work in graph data?

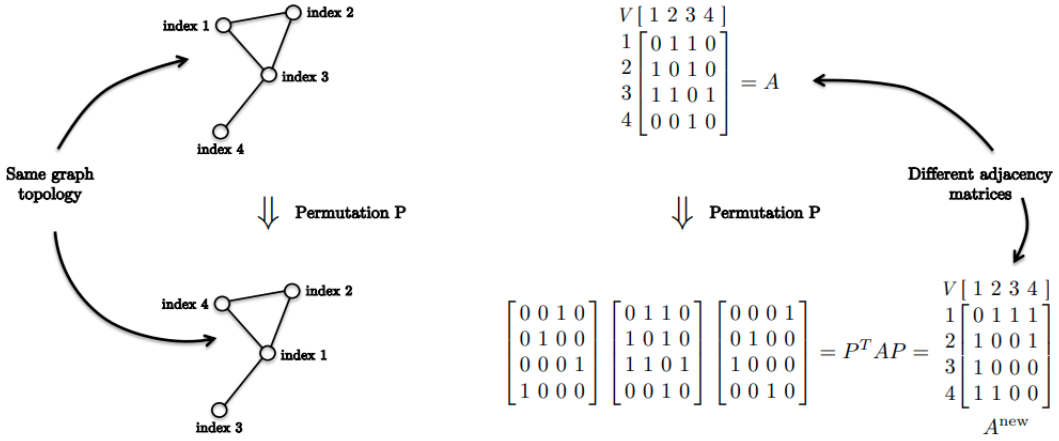


Figure 2.10: Same graph represented with different adjacency matrices. Source [12]

Deep learning architectures like convolutional neural networks or recurrent neural networks, presented in Section 2.2, are extremely useful when dealing with Euclidean data structures, like images. In order to deal with graph data, a simple method is to represent the graph with it's adjacency matrix and use it as the input to a feed forward neural network. The problem with this approach, is that feed forward neural network produce different outputs for different node orderings in the adjacency matrix. But permutations of the adjacency matrix, represent the same graph. Therefore, it is necessary to ensure that our architecture is permutation invariant.

Neural Message Passing

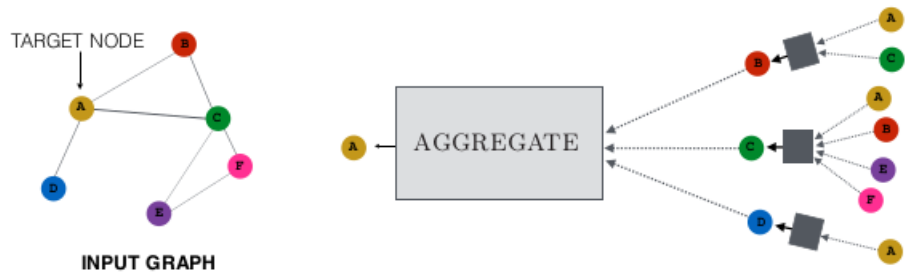


Figure 2.11: Illustration of how a single node aggregates messages from its neighbors. Source [37]

The basic architecture currently used in graph domain, is called Graph Neural Network [99]. The goal of GNN is to learn a representation $h_u \in \mathbb{R}^s$ for every node u in graph,

that capture the neighborhood information. The representation h_u of every node u can be used in order to produce the label of the node u for a node classification task. If we want to apply GNN in a graph classification task, we can aggregate all the representations of the nodes, in order to obtain the representation of the graph h_G, and then feed it to a classifier. To learn these representations, GNN uses a general framework, called neural message passing [34], in which every node exchanges messages with its neighbors. In start, every node u is initialized with a random state h_u. In every iteration, we update the state of the node, based on its previous state and the representations of its neighbors. In math notation, we have the following equations:

$$m_u^{k+1} = AGGREGATE(h_v^k, \forall v \in N(u)) \tag{2.13}$$

$$h_u^{k+1} = UPDATE(h_u^k, m_u^{k+1}), \tag{2.14}$$

where $N(u)$ is the neighborhood of node u, AGGREGATE and UPDATE are differentiable functions, usually approximated by neural networks [41], m_u^{k+1} is the aggregated message from the neighbors in $k+1$ iteration and h_u^{k+1} is the representation of the node u in $k+1$ iteration. The AGGREGATE function should be permutation invariant, as we mentioned before. For the graph classification setting, we need an extra step, where we obtain the graph representation by aggregating the representations of the nodes. This step is usually called readout step. It is presented in the following equation:

$$h_G = READOUT(h_u^K, \forall u \in V), \tag{2.15}$$

where READOUT is a readout function, and K refers to the final iteration.

2.4 Deep Generative Models of Graphs

In the previous section, we showed how can we learn representations of graphs, in order to make predictions about the nodes (node classification task) , about the edges (link prediction task) or about the whole graph (graph classification task). In this section, we will provide techniques for a related problem, that of graph generation. In this task, we want to output graphs that have specific properties, or generated from a specific distribution. For instance, we may want to generate molecules with high drug-likeness [125] or generate scene graphs from images [123]. With deep learning, we are able to learn a generative model of graphs, based on our input training graphs, and output graphs with similar properties as the training set. We classify these techniques based on two properties:

1. **The representation of the graph**. There are three main types: token-based, adjacency-matrix based, and graph-based. Token-based approaches represent a graph as a sequence of strings. For example, molecule graphs can be represented with SMILES grammar [113, 21]. These methods use RNNs due to their advantage in sequence representation. Adjacency-matrix approaches represent the graph with

its adjacency-matrix [101, 126]. These methods must take into account that permuted adjacency matrices, represent the same graph. Therefore, it is necessary to use permutation invariant functions[84]. Graph-based approaches, which attracted much attention in the last years, use Graph Neural Networks in order to produce the representation of the nodes and the graph. They operate directly on the graph structure, reducing the dependencies on the node ordering and efficiently capturing the edges relations. [129, 127, 67].

2. **The generative framework**. The most popular techniques are variational autoencoders (VAEs), autoregressive models and generative adversarial networks (GANs). We provide a short review for each of them, applied on graph domain.

2.4.1 Autoencoders

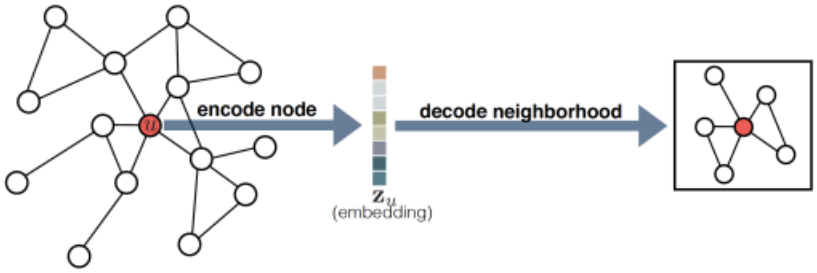


Figure 2.12: Overview of a graph autoencoder . The encoder maps a node to an embedding. The decoder then use the embedding to reconstruct the neighborhood of the node. Source [37]

Before we analyze how VAEs work, we provide some general information about autoencoders on graph data. Autoencoders learn compact representations of data, by using an optimization process with an encoder and a decoder. The encoder model maps each sample into a low dimensionality embedding. Then, the decoder model, try to reconstruct the sample, given the embedding as input. These encoder and decoder functions, are usually approximated by neural networks. In order to train this model, the standard pipeline is to minimize the reconstruction loss $L = \sum_{u \in V_{train}} l(DEC(ENC(u)), u)$, between the input sample and the decoded sample. In general, the dimensions of the embedding space, are lower than the dimensions of the input. Therefore, the system learns to maintain only the important information, and discard the rest. So if autoencoders learn these representations, why do not use them for our graph generation problem? Is it possible to just take samples from the learned space, and decode them into new data? To answer these questions, it is helpful to look at Figure 2.13. The learned space of an autoencoder, which is optimized only with the reconstruction loss, is usually discontinuous and does not allow easy interpolation. This leads to high overfitting on the training set. As a result, some

points of the latent space will give meaningless content once decoded. To address this problem, variational autoencoders have been proposed [51].

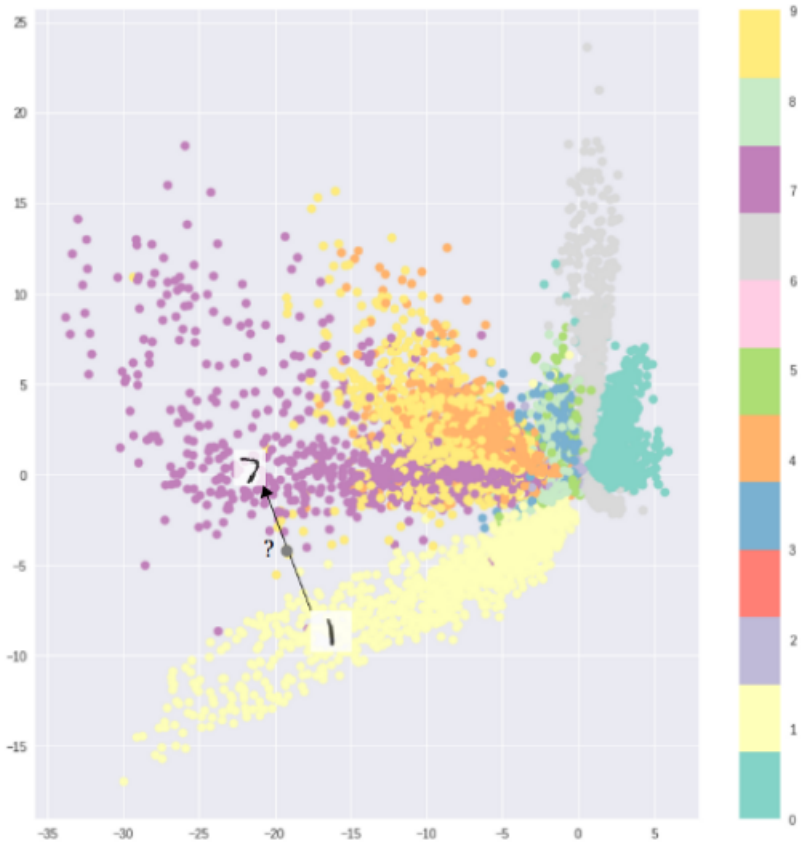


Figure 2.13: Visualization of the the learned space of an autoencoder trained on MNIST dataset, optimized only with reconstruction loss. The learned space is discontinuous with distinct clusters. Source [45]

2.4.2 Variational Autoencoders (VAEs)

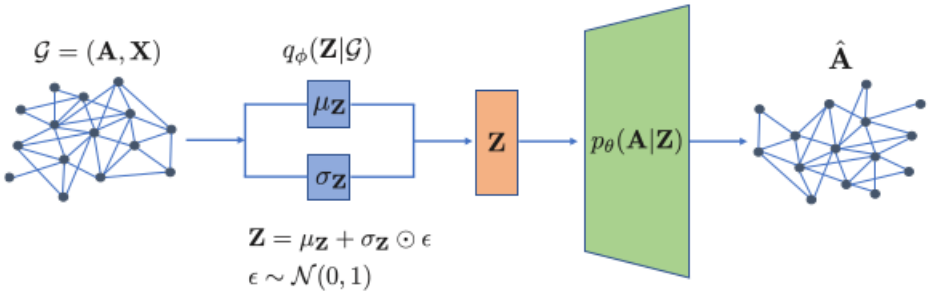


Figure 2.14: Illustration of a graph VAE model. Source [37]

Variational autoencoders (VAEs) are powerful models, that use variational inference in order to generate new data [53]. The main difference with traditional autoencoders, is that their learned space has two important properties:

- **Continuity** so that nearby points in the embedding space, produce related decoded points.
- **Completeness** so that every decoded point is meaningful.

To achieve this, the encoder outputs the mean and the standard deviation of a normal distribution, instead of a single vector, and introduce a new loss function. In math notation, VAE for graphs has the following components:

- A **stochastic encoder** q_ϕ, that takes a graph G as input. It defines a Gaussian distribution $q_\phi(Z|G)$ over latent representations $Z \sim N(\mu_\phi(G), \sigma_\phi(G))$. The mean and the covariance of this distribution, are produced from neural networks.
- **A stochastic decoder** p_θ. Its input is the representation Z, and outputs the probability distribution of the data $p(x|Z)$. The decoder is also parameterized by a neural network.

Also, we assume a prior Gaussian distribution $p(Z)$ over the latent representations. In order to train the model, with a training set of graphs $D_{train} = G_1, .., G_n$, we minimize the loss function:

$$L = \sum_{G_i \in D_{train}} \mathbb{E}_{q_\theta(Z|G_i)}[p_\theta(G_i|Z)] - KL(q_\theta(Z|G_i)||p(Z), \tag{2.16}$$

We call this loss function as evidence likelihood lower bound (ELBO). The first term of the loss function is the reconstruction loss. This term encourages the decoder to reconstruct the graph from the embedding space, such that the decoded and the input graph be similar. The second term is the Kullback–Leibler divergence [58] between the prior distribution $p(Z)$ and the encoder's distribution $q_\theta(Z|G)$. It acts as a regularizer, in order to keep the encoders representations as close as possible to the normal distribution.

2.4.3 Autoregressive models

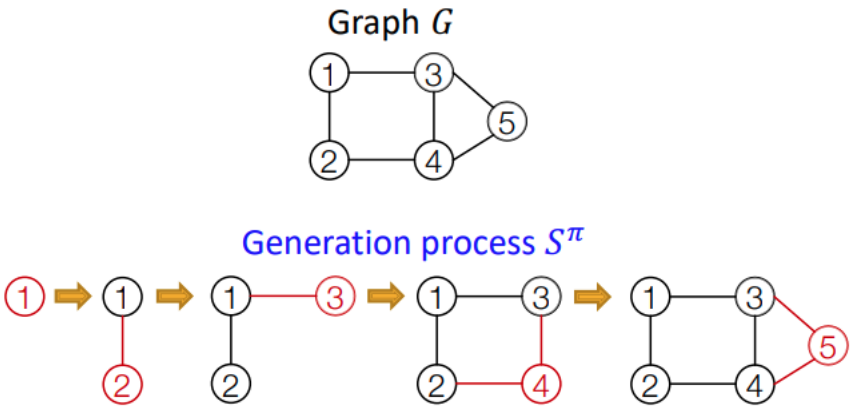


Figure 2.15: Illustration of a graph generation, using an autoregressive model. Source [61]

Autoregressive models assume relations between the edges and the nodes, and process them one by one, such that the probability of every edge and every node is conditioned on the previous edges and nodes. We treat the generation problem sequentially, therefore we usually use a recurrent network that capture the graph state in every step, in order to encode the graph representation [127, 66]. In training, in order to maximize the likelihood of the training data, we employ the *teacher forcing* strategy [46], i.e., we force the recurrent network to generate the truth values of nodes and edges at each step. After training, in order to generate new graphs, we can use an RNN to sequentially generate the graph, starting from an initial hidden state h_0 from the learned space Z. Autoregressive models have shown promising results when combining with a VAE pipeline, or with graph neural networks [129]

2.5 Neural Architecture Search (NAS)

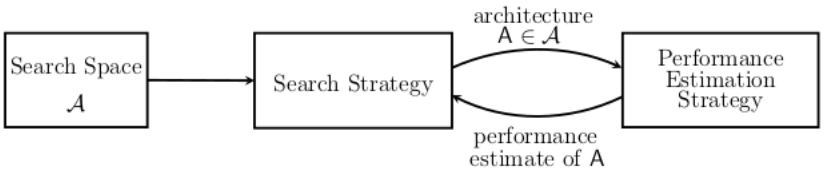
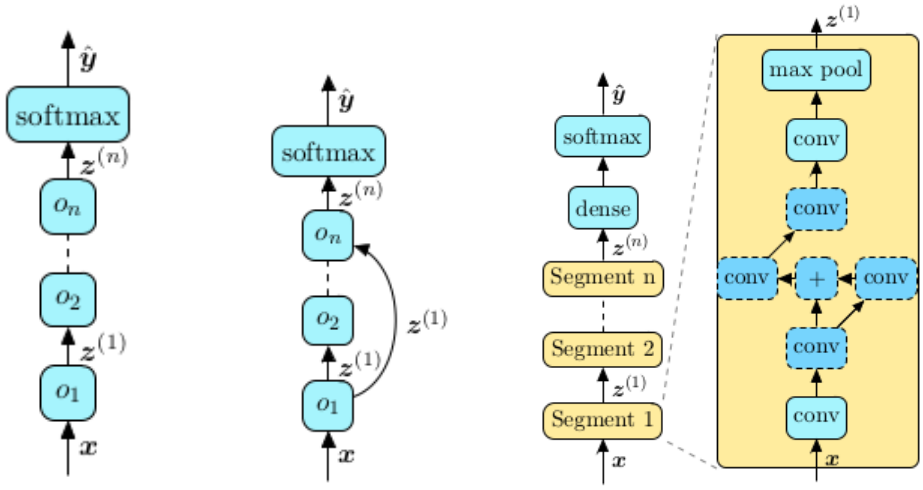


Figure 2.16: An illustration of Neural Architecture Search pipeline. An architecture A is selected from a search space $\mathcal{A}$, using a specific search strategy. The performance of the architecture is estimated, and the search strategy receive the performance signal. Source [29]

All the deep learning architectures presented in the previous sections (2.2, 2.3, 2.4), from convolutional and recurrent neural networks, to graph neural networks and generative models, require an extensive architecture engineering. This procedure is extremely time-consuming as the researcher must design the architecture manually, based on his domain expertise and experience. Inevitably, a human bias is introduced, and the architecture choices are limited from the research knowledge. Therefore, in the last years, there has been an increasing attention to automating architecture engineering. The field called Neural Architecture Search, has proven its ability to construct architectures that achieve state of the art results, in various tasks, with little human intervention [135, 136, 96, 90, 33]. In this section, we follow the categorization of [29], and analyze the three basic components of Neural Architecture Search: search space, search strategy and performance evaluation.

2.5.1 Search Space Design



(a) A chain structured search space [7]

(b) A search space with skip connections [136]

(c) A search space with an architecture template. Only the connections between the dark blue (dashed) operations are not fixed. [118]

Figure 2.17: Examples of different search spaces

The search space defines the set of all neural network architectures that can be represented. In order to define an architecture, two properties must be specified: the operation associated with every node or layer, and the connections between them. In Figure 2.17, examples of search spaces are illustrated. The size of the search space, plays a crucial role on the model's expressiveness but has an influence on the computational cost too [116]. Therefore, there is an important trade-off between the number of different architectures that we want to represent, and the running cost of our algorithm. As deep neural networks

usually have hundreds of layers, the number of different choices grows exponentially. To deal with this problem, we usually enforce some constraints both on the structure and the operations of layers, limiting the size of the search space.

Cell-Based Search Space

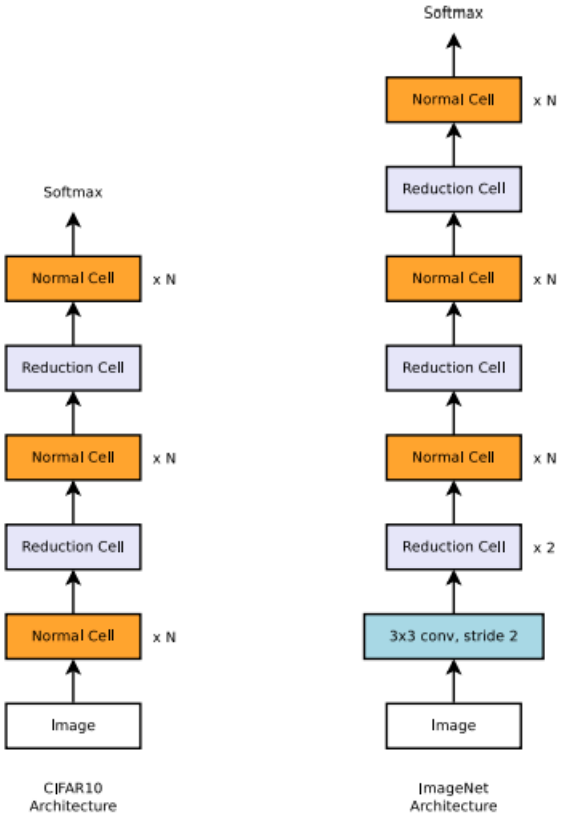


Figure 2.18: The NASNet search space design the architecture as a repeated stack of cells. Source [136]

Inspired by human engineered architectures, that contains repeated motifs [107, 42], some neural architecture search approaches instead of searching the whole architecture, they search for a motif or cell [136, 70, 132]. Once they found it, this cell is stacked multiple times to form a larger architecture. For instance, [136] try to learn two types of cells:

- A normal cell, which the input and the output have the same dimensions
- A reduction cell, which the output has the half width and height from the input.

This method has multiple advantages:

1. The size of the search space is reduced, as the cell is much smaller than the whole architecture [136].

2. These cell-based architectures are easily transferable to other tasks by simply stack more cells. For instance, a cell discovered on CIFAR-10 [54] can be transferred to ImageNet [23] and achieve great performance [136].

To sum up, the search space is one of the key points of neural architecture search, so it must be carefully designed. The human heuristics introduced in it, can lead to better performance [136], but may also limit the opportunities to discover new architecture principles [119, 88].

2.5.2 Optimization methods

Many different optimization algorithms has been used in order to find the best architecture in the search space, such as random search [64], reinforcement learning [135], Bayesian optimization [49], evolutionary methods [89] and gradient-based methods [72]. The selection of the optimization method is highly influenced from the search space, as some algorithms require specific settings. For instance, in order to apply gradient-based methods, the search space must be continuous. In the following paragraphs, we review the most important optimization methods.

Reinforcement Learning (RL)

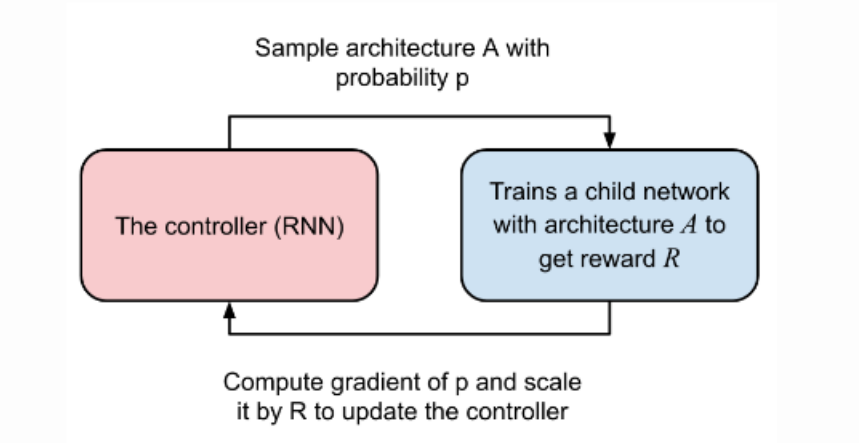


Figure 2.19: An illustration of NAS pipeline using reinforcement learning. Source [134]

One of the most famous optimization techniques for neural architecture search is reinforcement learning-based [7, 135, 131]. A reinforcement learning based controller propose child model architectures for evaluation, from the search space. An architecture is modeled as a configuration string, and the controller is usually a recurrent neural network, that outputs a variable-length sequence of tokens in order to specify the network architecture. After the sampling of the architecture, we train and evaluate its performance. This performance is the reward that our controller receives. Because the reward signal is non-differentiable, we use a policy gradient method and update the controller. In math

notation, we want to maximize the expected reward

$$J(\theta_c) = E_{P(a_{1:T};\theta_c)[R]}, \tag{2.17}$$

where $a_{1:T}$ are the actions of the controller, T is the total number of tokens,θ_c are the parameters of the controller and R is the reward. Using REINFORCE rule [115] as the policy gradient method, we obtain the gradient of the expected reward as

$$\nabla_{\theta_c} J(\theta_c) = \sum_{t=1}^{T} E_{P(a_{1:T};\theta_c)}[\nabla_{\theta_c} \log P(a_t|a_{t-1:1};\theta_c)R]. \tag{2.18}$$

Evolutionary Methods

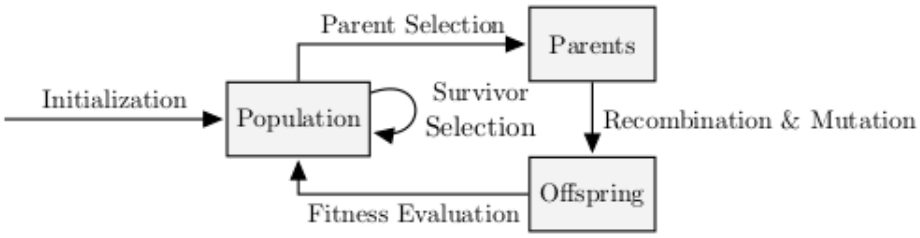


Figure 2.20: An illustration of NAS pipeline using evolutionary methods. Source [29]

Evolutionary algorithms evolve a population of architectures in order to find an optimal network [89, 90, 103]. In every evolution step, using a parent selection method, some architectures are sampled from the population and become parents. They generate offsprings by applying mutations to it. These mutations can alter the structure of the network, such as adding or removing a layer, or alter the operations of the network, such as change a convolutional layer to a max pooling layer. After training the offsprings, their fitness (e.g., the performance on the test set) is evaluated and the most promising are added to the population pool. There are many parent selection methods in neural architecture search, such as tournament selection [35, 90, 73], multi-objective Pareto method using an inverse density [28], or simply removing the worst or the oldest model in the population [89, 90].

Gradient-Based Methods

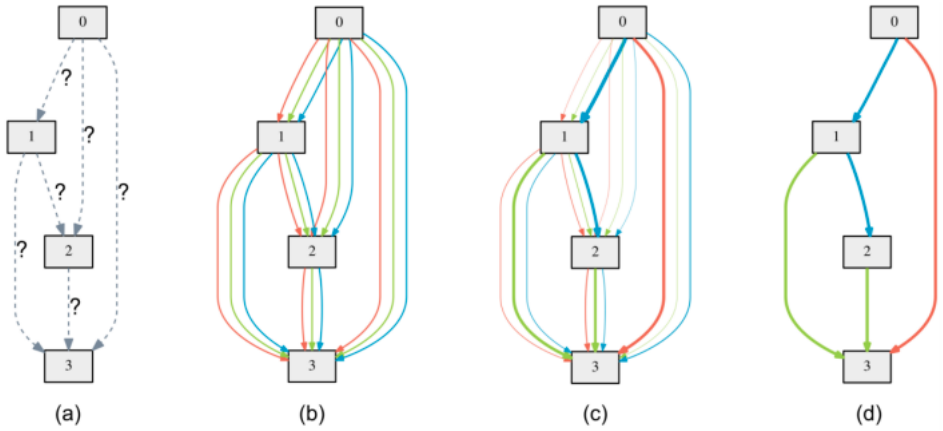


Figure 2.21: An illustration of a continuous optimization pipeline in NAS. (a) Operations on the edges are initially unknown. (b) Continuous relaxation of the search space by placing a mixture of the operations on every edge. (c) Joint optimization of the mixing probabilities and the network weights by solving a bilevel optimization problem. (d) Selecting the final discrete architecture from the learned mixing probabilities. Source [71]

All the previous methods operate on a discrete search space, therefore a gradient-based optimization is not applicable. To overcome this difficulty, a continuous relaxation of the search space has been proposed [72]. Instead of executing a single operation in every layer, a linear combination of all the operations is introduced. In math notation, for a discrete search space, the output of each node can be computed as $x_j = \sum_{i<j} o_{(i,j)}(x_i)$, where $o^{(i,j)}$ is the operation that apply the node j to the output of the node i. In contrast, in the continuous space, the categorical choice of each operation is transformed into a softmax over all possible operations:

$$\bar{o_{(i,j)}} = \sum_{o \in O} \frac{\exp(a_o^{(i,j)})}{\sum_{\grave{o} \in O} \exp(a_{\grave{o}}^{(i,j)})} o(x) \tag{2.19}$$

, where the operation weights are parameterized by the vector $a^{(i,j)}$. The goal is to learn these vectors for all pair of nodes. Then, the discrete architecture is obtained by replacing each mixed operation $\overline{o_{(i,j)}}$, with the most likely operation $o^{(i,j)} = argmax_{o \in O} a_o^{(i,j)}$ Therefore the element $a_o^{(i,j)}$ can be seen as the probability of the operation o applied from node j given as input the output of node i. The learned parameters $a^{(i,j)}$ are jointly trained along with the weights of the architecture. Formally, the neural architecture task in this setting can be described as a bilevel optimization problem [20]. Given a train D_{train} and a validation set D_{val}, the goal is to find an architecture a^* that minimizes the validation loss $L_{val}(w^*, a^*)$, where the weights w^* are optimized by minimizing the training loss

$w^* = argmin_w L_{train}(w, a^*)$. The architecture a is the upper-level variable and the weights w are the lower-level variable.

$$\min_a L_{val}(w^*(a), a) \tag{2.20}$$

$$\text{s.t } w^*(a) = argmin_w \ L_{train}(w, a) \tag{2.21}$$

Bayesian Optimization

Neural architecture search can be formulated as a black box optimization problem, with an expensive objective function f (i.e. the performance of the architecture), as it requires training the architecture before evaluating it. Bayesian optimization is a common choice in this setting [29, 50], as it constructs a surrogate model for modeling the objective function f and define its prior. After some evaluations of f, it used Bayes rule to obtain the posterior of the objective function f. In the next step, it uses an acquisition function $\alpha(x)$, which is a function of the posterior, to decide the next sample point $x_t = \text{argmax}_x \alpha(x)$ that maximizes the acquisition function. The main limitation is that typical bayesian optimization methods are effective only on a low-dimensional continuous space as they are based on Gaussian processes, so it is difficult to apply them directly on the discrete architecture search space [29]. To face this problem, some approaches used kernel functions for architecture search spaces [50, 106].

2.5.3 Performance Evaluation

Table 2.1: Classification error and computational cost of many NAS methods in CIFAR-10. Source [116]

Reference	Error(%)	Params(Millions)	GPU Days	Search Method
Baker et al. [7]	6.92	11.18	100	Reinforcement Learning
Zoph and Le [135]	3.65	37.4	22400	Reinforcement Learning
Cai et al. [14]	4.23	23.4	10	Reinforcement Learning
Zoph et al. [136]	3.41	3.3	2000	Reinforcement Learning
Real et al. [89]	5.4	5.4	2600	Evolution
Liu et al. [73]	3.75	15.7	300	Evolution
Real et al. [90]	3.35	3.2	3150	Evolution

Every search strategy in neural architecture search, aims to find an architecture that maximizes a performance metric, such as accuracy on validation set. However, before estimate this performance, the architecture must be trained on a training set. Due to the fact that a common search space contains a very big number of architectures, the training procedure of each of them, has an intractable computational cost. In Table 2.1, the computational cost of many neural architecture search methods in CIFAR-10 is illustrated.

Even for a small dataset like CIFAR-10, the search procedure can take hundreds of GPU-days. In order to make neural architecture search more efficient, it is necessary to overcome its bottleneck: the training procedure of the architectures. Therefore, recent efforts try to reduce the cost of training or to estimate the performance of an architecture without fully training it. A summary of these approaches are presented in Table 2.2.

Table 2.2: Overview of methods for reducing the cost of performance estimation in NAS. Source [29]

Speed-up Method	**Description**	**Reference**
Lower fidelity estimates	Training time reduced by training for fewer epochs, on subset of data, downscaled models, downscaled data.	Li et al. [65], Zoph et al. [136], Real et al. [90], Runge et al. [96]
Learning Curve Extrapolation	Training time reduced as performance can be extrapolated after just a few epochs of training	Swersky, Snoek, and Adams [105], Baker et al. [6]
Weight Inheritance Network Morphisms	Instead of training models from scratch, they are warm-started by inheriting weights of, e.g., a parent model.	. Real et al. [89], Elsken, Metzen, and Hutter [28]
One-Shot Models-Weight Sharing	Only the one-shot model needs to be trained; its weights are then shared across different architectures that are just subgraphs of the one-shot model	Pham et al. [87], Bender et al. [8], Liu, Simonyan, and Yang [72], Xie et al. [120]

The above techniques made neural architecture search accessible to more researches, as the computational cost has dramatically reduced, without compromising the performance [87].

Chapter 3

Generating Neural Network Architectures with Graph Variational Autoencoder

This chapter contains the main contribution of our thesis. First, we provide a detailed related work section. Then we proceed with the problem definition and our methodology.

3.1 Related Work

3.1.1 Graph Variational Autoencoders for Neural Architecture Search

Graph variational autoencoders have shown promising results in neural architecture search. The closest work to ours is by Zhang et al. [129], who uses a graph variational autoencoder (D-VAE) in order to construct a latent space, and then apply bayesian optimization on it, just like our method. The key difference is that we use operation embeddings to represent operations on layers, instead of one-hot vectors, so we can capture the relations between the operations, and eventually produce a smoother latent space.

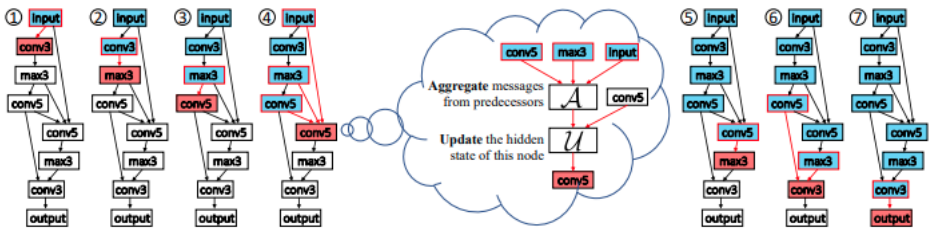


Figure 3.1: An illustration of the encoding procedure of a neural architecture of the DVAE. Source [129]

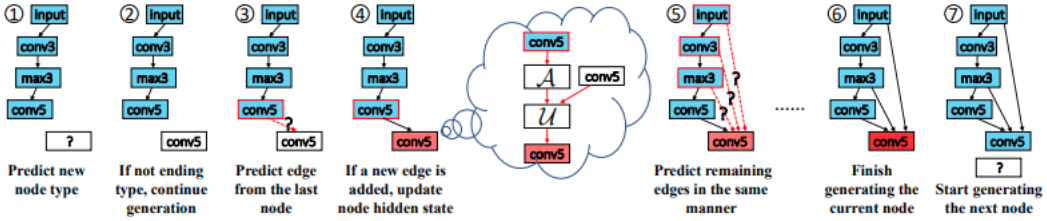


Figure 3.2: An illustration of the process of node's generation of DVAE. Source [129]

Li et al. [63] also proposed a variational autoencoder (NGAE) in order to apply gradient optimization methods in a continuous latent space. Their input consists of three parts: the architecture, its performance (e.g. classification accuracy on a specific dataset) and its computational complexity (e.g. the amount of computations). Along with the autoencoder, they jointly train a performance predictor and a complexity predictor. Due to the use of labels from a specific dataset, their latent space cannot be applied for optimization in other datasets. They also use one-hot vectors for the operation representation.

Yan et al. [122] proposed a Variational Graph Isomorphism Autoencoder in order to obtain unsupervised representations of neural network architectures. They leveraged Graph Isomorphism Networks [121] in order to encode the graph architectures into the latent space. However, in order to represent the operation of every node, they used one-hot vectors.

3.1.2 Gradient-Based Optimization

Luo et al. [77] proposed an approach called Neural Architecture Optimization (NAO). They jointly learn an autoencoder between networks and also a performance predictor that predicts the performance of an architecture on a given dataset, from its continuous representation. Then, they use gradient descent in order to optimize the performance predictor and find better architectures in the continuous space. There are some important differences with our approach. In order to represent the architecture they use strings, instead we use a directed acyclic graph representation, which exploit the structural information of the architecture. Moreover, they use supervised learning instead of unsupervised learning, so they need to train first and evaluate many architectures, and use these results to supervise the training of the model.

3.1.3 Bayesian Optimization

Kandasamy et al. [50] proposed a Bayesian optimization approch for neural architecture search. They define a kernel that computes the similarities between architectures by solving an optimal transport problem. Then, they develop a bayesian optimization framework for optimizing functions on neural network architectures, using an evolution-

ary algorithm to optimize the acquisition function. This work is related to ours in terms of the Bayesian optimization approach. However, the usage of a kernel adds extra complexity to the system, instead of our end-to-end approach with a variational autoencoder. In addition, they search directly in the discrete space, extrapolating near existing architectures, and do not use a more global search method. On the contrary, we perform bayesian optimization in the entire continuous learned latent space, enabling more global optimization.

3.2 Problem Setup

Let X be an input space of graphs that represent neural network architectures, and let $f : X \to \mathbb{R}$ be an objective function, that represents the performance of a neural network architecture in a specific dataset D. The first task is to learn an encoder $q : X \to Z$, that find a corresponding latent point z_i for every data graph x_i and a decoder $p : Z \to X$, that maps from Z to X. The second task is to learn a latent objective model $h : Z \to \mathbb{R}$ in order to approximate f at the output of p, i.e. such that $f(p(z)) \approx h(z), \forall z \in Z$. The output of the system must be the graph $x_{sol} = p(z_{sol}) \in X$ such that $z_{sol} = \text{argmax}_z\, h(z)$.

3.3 Graph Variational Autoencoder with Operation Embeddings

In this section, we introduce our proposed graph variational autoencoder model for neural architecture search. We follow the typical learning procedure of a varational autoencoder [51]. We want to learn a probabilistic generative model $p_\theta(x|z)$ as well as an approximated posterior distribution $q_\phi(z|x)$. We refer to the probabilistic generative model as the *decoder*, and to the approximated posterior distribution as the *encoder*. The system is trained through maximizing the evidence lower bound:

$$L(\phi, \theta; x) = \mathbb{E}_{z \sim q_\phi(z|x)}[\log p_\theta(x|z)] - KL[q_\phi(z|x)||p(z)], \tag{3.1}$$

where $p(z)$ is the prior that follows a normal distribution. Note that the training of the autoencoder is unsupervised, as we do not use any accuracy signal of the architectures. Therefore, our learned latent space can be used for searching architectures for different tasks.

3.3.1 Input Representation

Every neural network architecture represents a computation, that is applied to an input signal using some operations. Therefore, we model every architecture as a computational graph, with nodes correspond to operations, and directed edges represent the signal flow among the operations. The computational graph is acyclic, in order to have a finite number of performed operations. Given as input an architecture, we construct the corresponding

acyclic graph $G = (V, E)$. Each node $u \in V$ is associated with a label y_u that corresponds to the operation of node u, such as 5×5 convolution or 2×2 max pooling. Notice that two same architectures with different weights correspond to the same computation. We also add a starting node without any predecessors, called the input node, and an ending node without any successors, called the output node.

3.3.2 Operation Embeddings

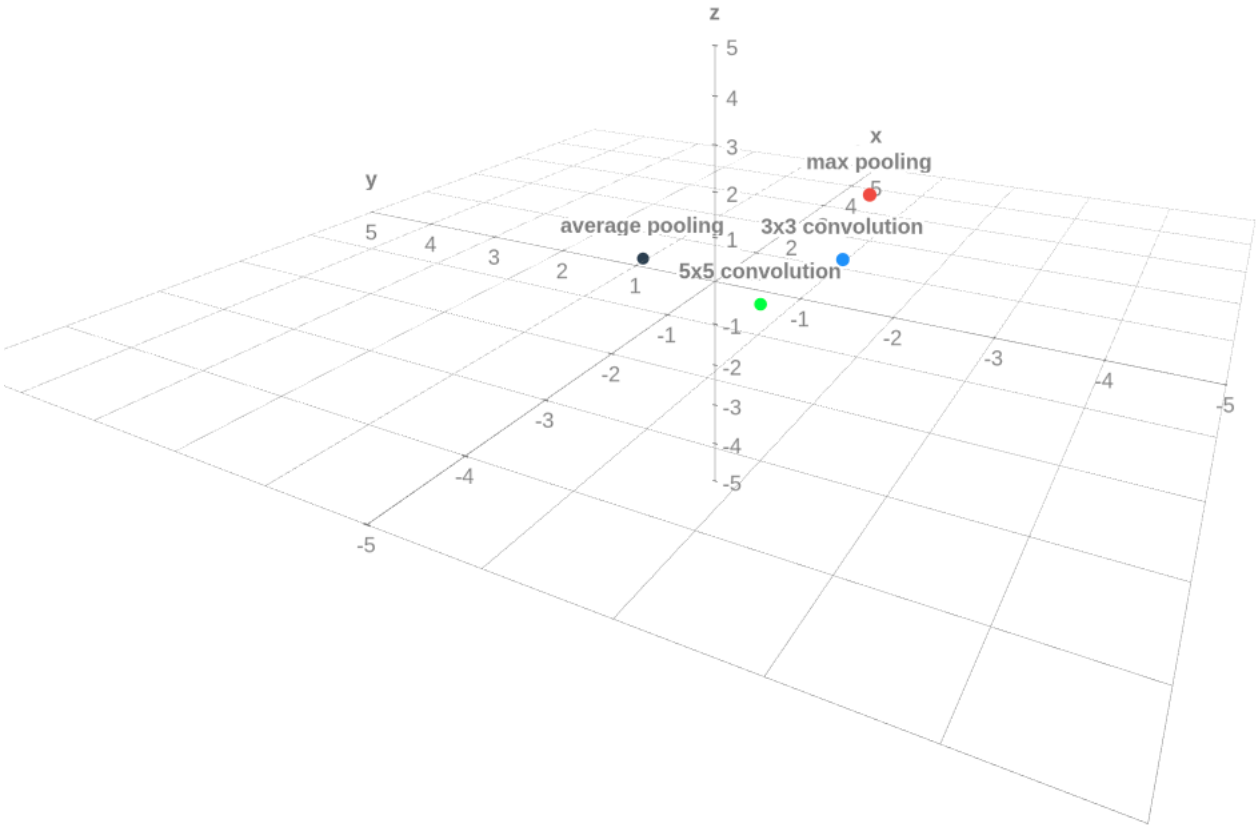


Figure 3.3: An illustration of three dimensional operation embeddings.

A neural network architecture can be seen as a heterogeneous graph due to the different node labels. Each node label represents a different operation. In the previous approaches for neural architecture search, the operations are represented with one-hot encoding [129]. One-hot encoding does not take into account the semantic meanings of different operations. For instance, a 5 × 5 convolutional layer is more similar to a 3 × 3 convolutional layer rather than to a max pooling layer. This is due to the fact that each vectorization is an orthogonal representation in another dimension and all the operations are equal distance apart, as there is no specific ordering.

As our goal is to learn a smooth latent space with respect to accuracy, it is neces-

sary to map similar architectures close to each other. Intuitively, similar architectures have similar graph structure and also similar operations on their nodes. Therefore, in order to capture the similarity between the operations, we change their one-hot vector representation. Inspired from the success of word embeddings [80], we propose **operation embeddings**. Operation embeddings are low dimensional dense vectors, that can capture the correlations between the operations. We visualize three dimensional operation embeddings in Figure 3.3. In order to learn these embeddings, we treat them as parameters of the autoencoder, and optimize them with gradient descent, along with the other weights.

We analyze the benefits of using operation embeddings below:

- Most importantly, we have observed a smoother latent space, than with the one-hot vector approaches, probably due to the fact that the model learns the relations and the similarities between the operations. Such smoothness can greatly influence the optimization in the latent space, since similar-performance architectures tend to locate close to each other, and modeling a smoothly-changing performance estimation function is easier. We present more details on the chapter 4, where we analyze our experiments.

- Moreover, we have a reduction in the memory consumption. Let n be the number of different operations, then in one-hot encoding, we require a vector with n elements to represent a node operation. In operation embeddings encoding, we only require d elements to represent a node operation, with d be the dimensionality of the embeddings and $d < n$. The dimensionality of the embeddings is smaller than the dimensionality of the one-hot vectors, as the embeddings are dense, unlike the one-hot vectors, so they can encode information in less bits.

- Finally, we observed a speed up in the convergence of the autoencoder. The model learned to accurately reconstruct the graph input data, in less training epochs. Intuitively, by capturing the relations between the operations, the model learns this extra information, that it is absence when the model is trained with the one hot-vector approach. Therefore, it can converge in less epochs, and more accurately. We also explore this aspect on the chapter 4.

3.3.3 Encoder

The first component of our model is the encoder. The main goal of the encoder is to learn a mapping that embeds the whole architecture graph as a point in a low-dimensional continuous vector space, such that similar computational graphs are mapped close to each other. The encoding procedure can be summarized in the following two steps:

- A graph neural network, that uses the asynchronous message passing scheme proposed by Zhang et al. [129], obtain a discrete representation h_G of the graph.

- Two feedforward neural networks take as input the discrete representation h_G of the architecture, and produce the mean μ and the variance σ^2 of the posterior approximation $q_\phi(z|G)$.

Our graph neural network exchanges information between the nodes following the topological order, in order to produce for every node u a hidden state vector h_u. Then, we use the hidden state vector h_{u_n} of the ending node as the whole graph representation h_G. To generate these representations, we use a two-step procedure. Firstly, for every node u we aggregate the messages from its predecessors neighbors, using an aggregation function A,

$$h_u^{in} = A(\{Concat(h_v, x_v) : (v \to u)\}) \tag{3.2}$$

, where x_v is the **operation embedding** of node v. Secondly, we update the representation of every node u, based on the incoming aggregated message from its neighbors and the operation embedding x_u of node u,

$$h_u = U(h_u^{in}, x_u). \tag{3.3}$$

As our goal is to perform optimization in the continuous learned space, our encoder must map different architectures to different representations h_G. To achieve this, the aggregation function A and the update function U must be injective [129]. Also, we want our encoder to be invariant to node permutations, such that isomorphic graphs, that represent the same architecture, be mapped in the same representation. To achieve this, our aggregation function must be permutation invariant [129]. To model these two functions, we use deep neural networks [41]. We follow the settings from Zhang et al. [129] and use a gated sum as an aggregation function:

$$h_u^{in} = \sum_{u \to v} g(Concat(h_v, x_v)) \odot m(Concat(h_v, x_v)). \tag{3.4}$$

This gated sum satisfies the above properties [121]. As an update function, we use a gated recurrent unit (GRU) in order to capture the temporal dependencies:

$$h_u = GRU(x_u, h_u^{in}) \tag{3.5}$$

3.3.4 Decoder

Our decoder generates graphs from the latent representations in an autoregressive manner, and follows the same techniques as the encoder. In start, we use a feed forward neural network, in order to map the continuous representation z to the initial graph hidden representation h_0. Then, we use a gated recurrent unit, in order to update the hidden states, and sequentially generate the nodes and the edges. The steps of the generation of i^{th} node can be summarized in the following bullets:

1. **Compute node's type distribution:** We feed the representation $h_{u_{i-1}}$ of the last generated node u_{i-1}, into a feed forward neural network followed by a softmax to determine the probability of every node type.

2. **Generate node:** We pick the type with the highest probability, and generate a node with this type. If the sampled type is the ending type, we stop the decoding, we connect the nodes without successors to u_i and output the DAG, else we continue the generation.

3. **Update state of node u_i:** We update the hidden state of the new generated node by $h_{u_i} = GRU(x_{u_i}, h^{in}_{u_i})$, where x_{u_i} is the operation embedding of the node type and $h^{in}_{u_i}$ is the aggregated message from the predecessors, as we described in the encoding process.

4. **Generate edges:** After the generation of the node u_i, we must decide which of the already generated nodes will be connected to the new node. Therefore, for every pair of nodes $(u_j, u_i), j < i$, we feed the representations of the nodes h_{u_j}, h_{u_i} into a feed forward neural network followed by a softmax in order to compute the edge probability of (u_j, u_i). If the probability is higher than 0.5, then we add this connection, and update the hidden state of node u_i using the third step.

3.3.5 Training Procedure

We optimize the whole system in an end-to-end manner using gradient descent. The objective of the training is to maximize the evidence lower bound 3.1. In order to measure the first term in the loss function (i.e. reconstruction loss) we use teacher forcing [46]. We follow the topological order of the graph, and sum the negative log likelihood of each generated node and edge. After the predictions at each step, we replace them with their ground truth. Therefore, the model makes predictions given the correct histories.

3.4 Optimization in the Learned Latent Space

After the training of the autoencoder, we want to generate high-performance architectures through Bayesian optimization in the learned latent space. Our goal was to construct a smooth latent space with respect to accuracy, in order to make the optimization easier, as the best performing architectures would be mapped close to each other. The optimization procedure it is described as follows:

- We define a sparse gaussian process [102] as the surrogate model for modeling the true objective function f.

- Following the approach of [59] , we define the expected improvement (EI) criterion as an acquisition function. We choose the next query architecture as the one which has the highest expected improvement over the current maximum.

$$EI(z) = \mathbb{E}(max(f(z) - f(z^+), 0)), \tag{3.6}$$

  where $f(z^+)$ is the current maximum, and z^+ is the location such that, $z^+ = \text{argmax}_{z_i \in z_{1:t}} f(z_i)$

- We update our models, based on the performance of the selected architectures and continue the iterations.
- Finally, we pick the best-performing architectures.

Chapter 4

Expirements

In this chapter we present our experiments. In order to have a fair comparison with other approaches, we follow the experimental scheme of [59, 129], and perform the following tasks:

- **Basic abilities of Graph Variational Autoencoder models:** We perform tests to evaluate the reconstructive and generative abilities of a graph variational autoencoder model.
- **Predictive performance of latent representation:** We test how representative the latent space embeddings are, regarding the performance of the neural network architecture.
- **Best performing architectures obtained from the latent space:** We test the accuracy of the best performing architectures obtained through bayesian optimization.
- **Latent space visualization:** We visualize the latent space in two dimensions in order to evaluate its smoothness with respect to accuracy.

We compare our autoencoder model with five baselines adapted for directed acyclic graphs: D-VAE [129], NGAE [63], S-VAE [11], GraphRNN [126], and GCN [52]. Due to limited computational resources, we use the reported results from [129, 63], that contain these baselines. Therefore, we have only run our model for 300 epochs, and borrow the other results from D-VAE paper.

4.1 Baselines

In this section we provide a summary of every baseline model.

4.1.1 D-VAE

As described in the related work section 3.1, it is the most closely related work to ours. The main difference is that they use one-hot vectors to represent the operations on nodes.

Therefore, D-VAE cannot capture the semantic meaning and the relations between the operations. As a result, the relations and the similarity between the architectures are not fully exploited, thus producing less smooth latent spaces.

4.1.2 NGAE

NGAE as described in the related work section, is a variational autoencoder along with a performance predictor and a complexity predictor. The three components are jointly train, with supervised labels of the accuracy and the complexity of every architecture in a specific dataset. In contrast with our approach, their supervised latent space cannot be extended to other datasets.

4.1.3 S-VAE

The S-VAE baseline represents a directed acyclic graph as a sequence of node strings. Each node is represented as the one-hot encoding of its operation type concatenated with a binary vector indicating which previous nodes have directed edges to it. S-VAE used an RNN-based variational autoencoder, and was applied in sequence generation problems.

4.1.4 GraphRNN

GraphRNN generates graphs sequentially, by generating parts of the adjacency matrix using a GRU. GraphRNN is a generative model which does not have an encoder, thus cannot perform optimization in the latent space. In the reported results, we combine the S-VAE's encoder with the GraphRNN in order to perform the optimization task. GraphRNN treats the architecture as sequence of strings, and it is not permutation invariant, as an architecture can have multiple string representations.

4.1.5 GCN

The graph convolutional network (GCN) is a graph neural network that performs a simultaneous message passing scheme. The nodes receive messages from their neighbors and update their own hidden states simultaneously instead of following the topological order of the graph. Due to the simultaneous message passage scheme they use, they can not efficiently represent the computation on the graph, and only focus on the local node features.

4.2 Dataset

Our training dataset consists of 19.020 neural architectures from the ENAS software [87]. Each architecture has 6 nodes(layers) except the input node and the output node. Each node is associated with an operation. There are six available operations: 3×3 and 5×5 convolutions, 3×3 and 5×5 depthwise separable convolutions, 3×3 max

pooling, and 3×3 average pooling. We use the 90% of the dataset as training data, and the remaining 10% only for evaluation. Due to our limited computational resources, we evaluate each neural architecture's weight-sharing accuracy [87] which is an approximation of the true accuracy, on CIFAR-10 [54]. As future work, we want to test our model using the NAS-Bench-101 dataset [124] which contains 423k unique trained convolutional architectures with their accuracy on CIFAR10. Another dataset that can be used is NAS-BENCH-201 [25] which is an extension to NAS-Bench-101 and contains architectures that are trained on CIFAR10, CIFAR100 [55] and Imagenet [57].

4.3 Basic abilities of Graph Variational Autoencoders

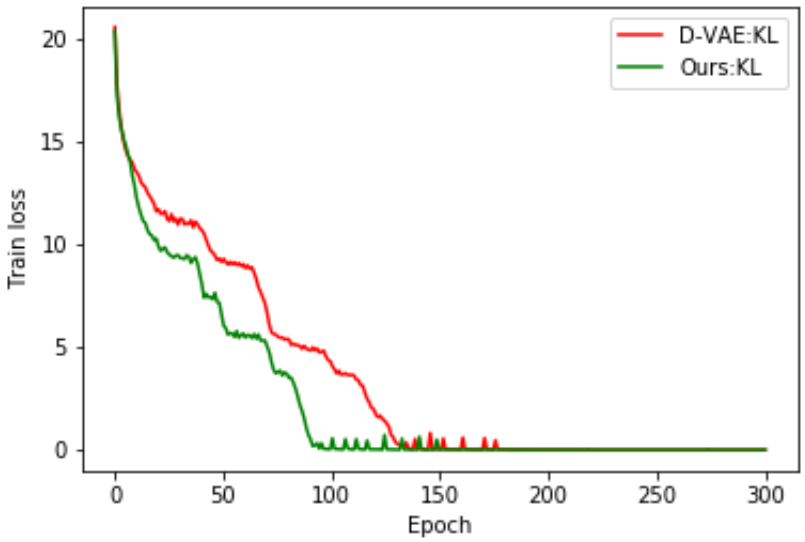


Figure 4.1: The training loss during the training process. The red function corresponds to D-VAE, and the green function to our model.

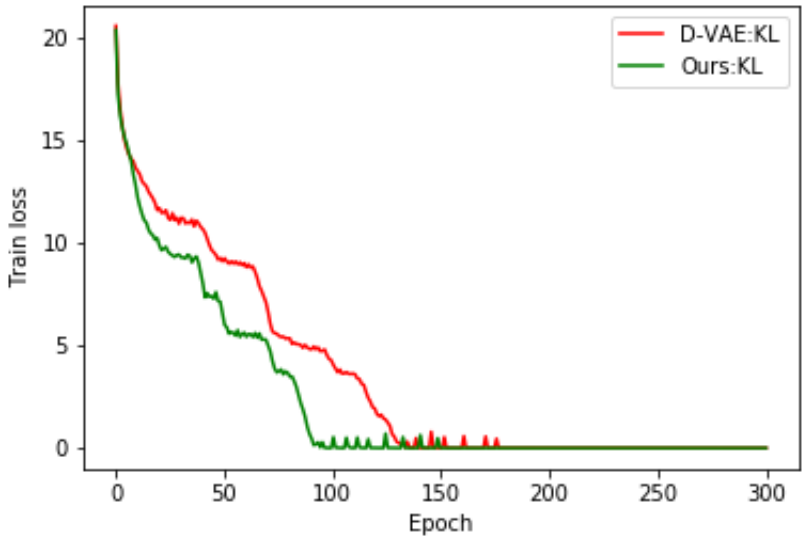


Figure 4.2: The reconstruction loss during the training process. The red function corresponds to D-VAE, and the green function to our model.

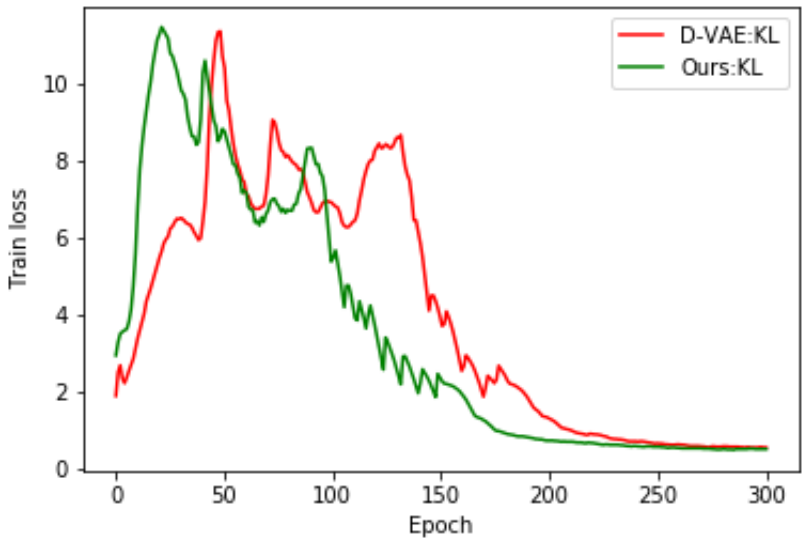


Figure 4.3: The KL divergence during the training process. The red function corresponds to D-VAE, and the green function to our model.

Table 4.1: Reconstruction accuracy, prior validity, uniqueness and novelty (%)

Models	Accuracy	Validity	Uniqueness	Novelty
Ours	**99.99**	**100.00**	**39.15**	**100.00**
D-VAE	99.96	100.00	37.26	100.00
NGAE	99.99	100	38.56	100
S-VAE	99.98	100.00	37.03	99.99
GraphRNN	99.85	99.84	29.77	100.00
GCN	98.70	99.53	34.00	100

The ability to reconstruct accurately the input data, and generate new unique samples, are prerequisites for the variational autoencoder models. Therefore, in order to evaluate these properties, we measure the following:

- **Accuracy**. Measures the rate of perfectly recovered graphs.
- **Validity**. Measures the proportion of valid neural network architectures generated from the prior distribution.
- **Uniqueness**. Shows the proportion of unique graphs from valid graphs.
- **Novelty**. Measures the proportion of novel graphs from valid graphs that are never seen in the training set.

We show the results in Table 4.1. Our model outperforms the others in all the categories. Our reconstruction accuracy is perfect, therefore the embeddings are accurately mapped to their original structures after latent space optimization. D-VAE has smaller reconstruction accuracy, because their one-hot vector representation fails to capture the operation information accurately.

We also visualize the training loss, the reconstruction loss and the KL divergence during the training of both D-VAE, and our proposed model, in Figures 4.1, 4.2 and 4.3. We observe two common characteristics in both models:

- The reconstruction loss has a high influence on the training loss, and it is the dominant term. This is natural, as the main objective of the autoencoder is to learn to reconstruct the input data.
- In the first epochs, the encoder is quite simple, therefore the posterior approximation $q_\phi[(z|x)$ is close to the prior $p(z)$. Consequently the KL divergence is small. As the training procedure continues, the encoder becomes more complex, and the posterior approximation gets away from the prior. As a result the KL divergence grows. However the reconstruction loss decreases, because the encoder can more accurately reconstruct the input data, due to its complexity.

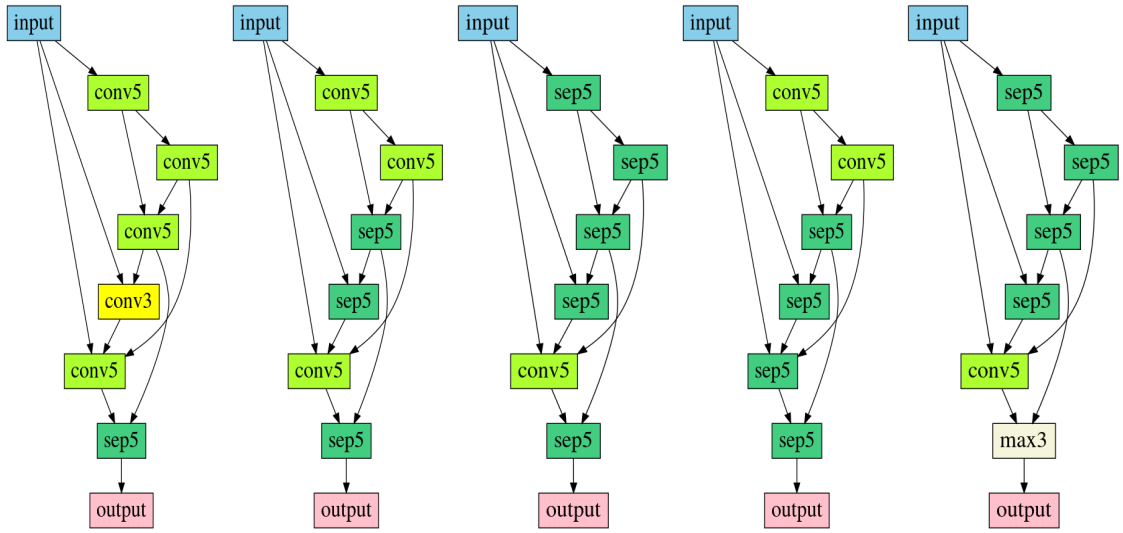
Despite these common properties, we observe that the reconstruction loss of our autoencoder can converge in far less epochs. Intuitively, our model by using operation embeddings, learns extra information by capturing the relations between the operations. These relations can not be exploited in the D-VAE model, which it uses one hot-vectors for the representation of the operations. Therefore, our model can converge in less epochs, and more accurately.

4.4 Predictive performance of encoded latent embeddings

Table 4.2: Predictive performance of encoded latent embeddings (%)

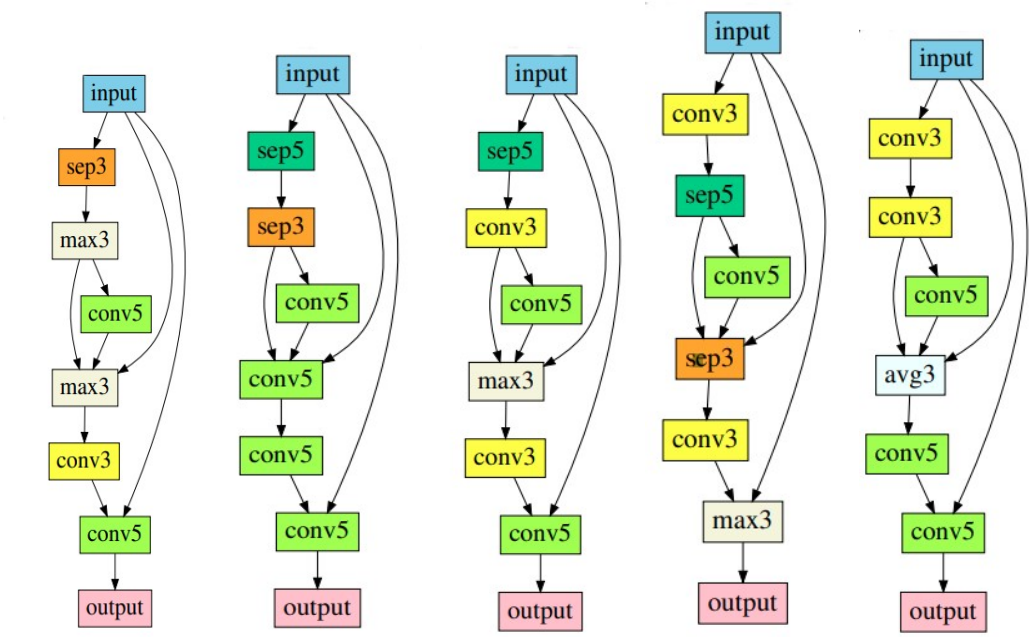
Models	RMSE	Pearson's r
Ours	**0.371±0.003**	**0.925±0.001**
D-VAE	0.384±0.002	0.920±0.001
S-VAE	0.478±0.002	0.873±0.001
GraphRNN	0.726±0.002	0.669±0.001
GCN	0.485±0.006	0.870±0.001

In this experiment, we test how representative the latent space embeddings are, for the performance of the neural network architecture. Therefore we test the unsupervised representation learning ability of the autoencoders. If we can accurately predict the performance, from the latent space embeddings, then we can also easily discover the best architectures from the latent space. We follow the procedure of [129], and train a sparse Gaussian process with 500 inducing points on the latent space embeddings of the training data, in order to predict the accuracy of unseen test architectures. We use two evaluation metrics, the Root Mean Square Error (RMSE) between the gaussian process predictions and the true performances, and the Pearson correlation coefficient (Pearson's r). Pearson correlation coefficient measures the linear correlation between the predictions and the true performances. Therefore, a model in order to have good predictive abilities, should have a small RMSE and a high Pearson's r We show the results in Table 4.2. Our proposed model outperforms the other autoencoders, in both metrics. This indicates that our latent space is more suitable for searching a high performance neural architecture.



(a) **Acc: 95.33** (b) **Acc: 95.17** (c) **Acc: 95.11** (d) **Acc: 95.10** (e) **Acc: 94.94**

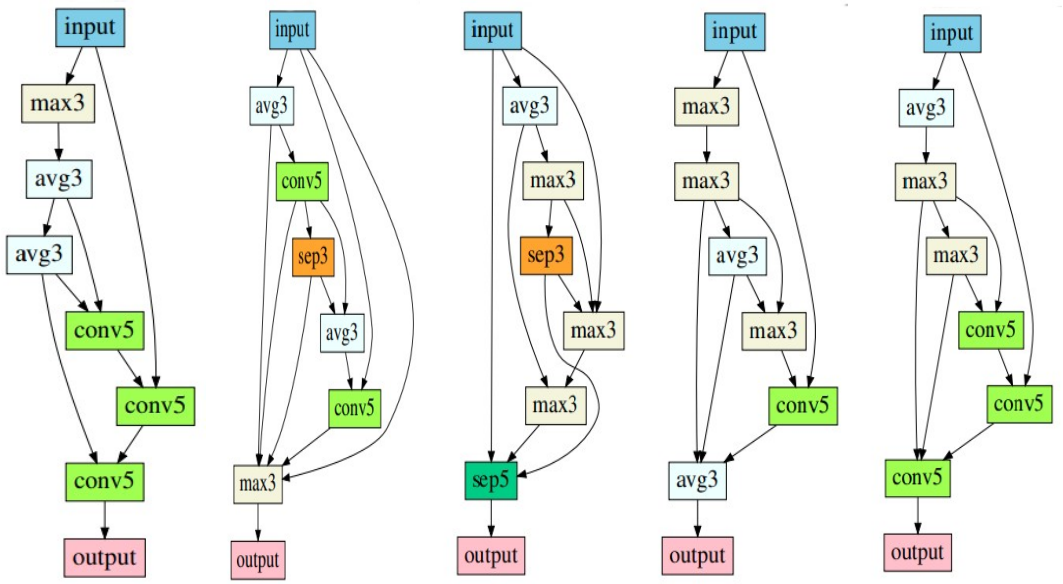
Figure 4.4: Illustration of the top 5 performing architectures found by **our proposed model**, and their test accuracies.



(a) **Acc: 94.80** (b) **Acc: 94.74** (c) **Acc: 94.70** (d) **Acc: 94.63** (e) **Acc: 94.63**

Figure 4.5: Illustration of the top 5 performing architectures found by **D-VAE**, and their test accuracies.

4.5 Best performing architectures obtained from bayesian optimization



(a) **Acc: 92.79** (b) **Acc: 92.00** (c) **Acc: 91.39** (d) **Acc: 91.38** (e) **Acc: 90.77**

Figure 4.6: Illustration of the top 5 performing architectures found by **S-VAE**, and their test accuracies.

In this experiment, we perform Bayesian optimization in order to generate high performance architectures. This is the main motivation of our Thesis, and therefore the most important expirement. Due to our limited computational resources, we used the weight sharing accuracy of the architectures [87], and then we selected the top 5 architectures, fully train them on CIFAR-10 and evaluate their performance.

We visualize the top 5 architectures discovered by each model in Figures 4.4 to 4.6. Our model generated higher performing neural architectures than D-VAE and S-VAE. Our best model achieves accuracy equal to 95.33, while D-VAE's highest accuracy was 94.80% and S-VAE's highest accuracy was only 92.79%. As in previous experiments, this indicates that our operation embeddings method, helps the model to construct a latent space that is very efficient for searching neural network architectures. Moreover, all our top 4 architectures, achieve accuracy higher than 95% , thus our model learns a distribution of high performing architectures, and not a single one.

4.6 Latent Space Visualization

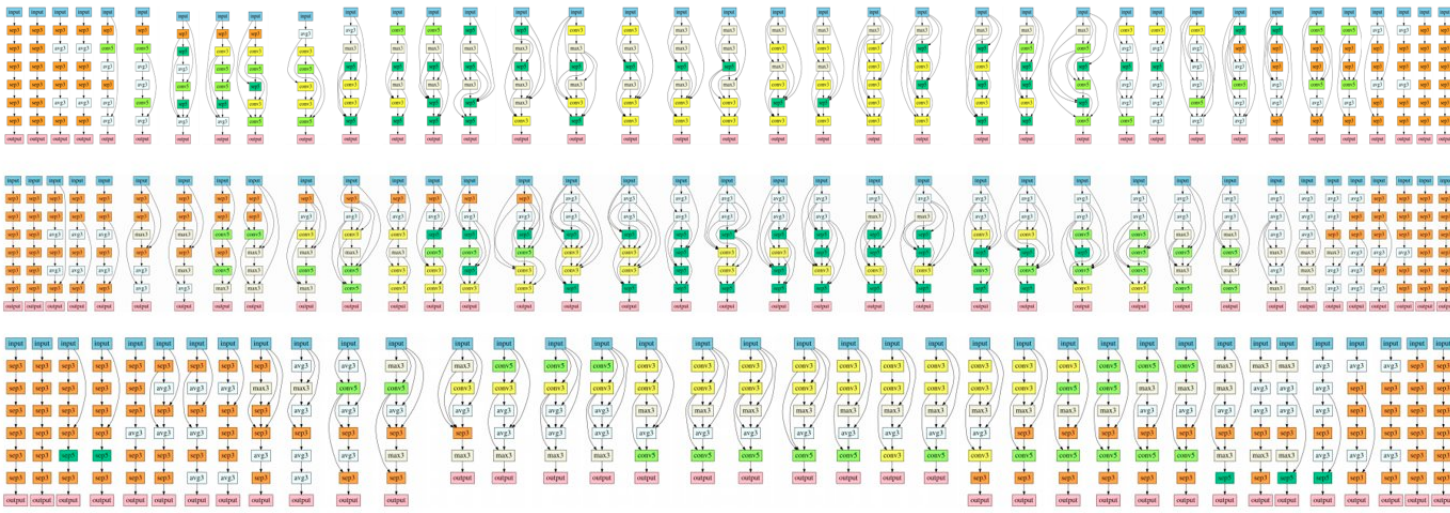
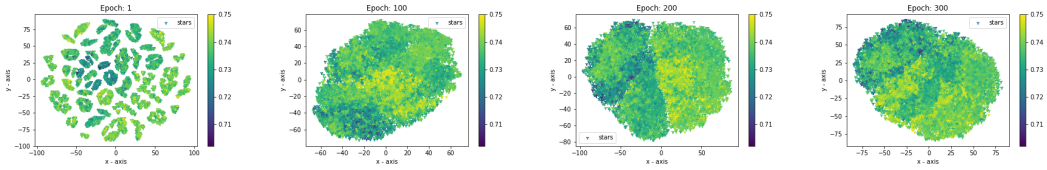
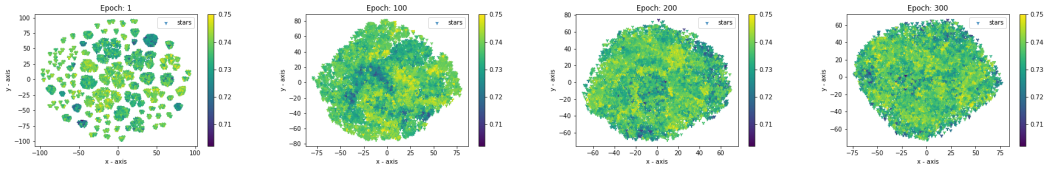


Figure 4.7: Circle interpolation starting from a point and returning to itself. Upper: Our Model. Mid: D-VAE. Lower: S-VAE



(a) (b) (c) (d)

Figure 4.8: 2-D Visualization of the latent space learned by our model during training



(a) (b) (c) (d)

Figure 4.9: 2-D Visualization of the latent space learned by D-VAE during training

In this experiment, we perform two tasks. First we visualize decoded architectures from embedding points along a great circle in the latent space [129, 114]. We start from a randomly point z_{start} in the circle and we evenly pick 35 points while returning to z_{start}.

Then we decode these latent points into the neural network architectures and visualize them. From this task we want to examine how smooth the latent space is, and if close by embeddings in the latent space, correspond to similar decoded architectures. In Figure 4.7 we present the decoded architectures. All three models have smooth transitions across the circle. In our model the architectures change smoothly and embrace not only similar connections but also similar operations.

In the second task, we visualize our latent space in two dimensions using t-SNE [78], along with the weight sharing accuracy of every encoded architecture. We present the results in Figures 4.8 and 4.9. As we can see, even from the epoch 100 our model can accurately cluster together the high performing architectures. This is the most important property in our application, as a smooth latent space is effective for optimizing the architectures. In contrast, in the D-VAE's latent space the transition of accuracy is non-smooth. This indicates that our operation embeddings method helps the process of mapping similar operations together and thus mapping similar performance architectures together. We also observe that the embeddings of both models span the whole latent space. This indicates that the autoencoders can injectively encode architectures in the latent space. In contrast, when the autoencoders are not trained (Figures 4.8(a) and 4.9(a)), the embeddings are discontinuous in the latent space. Therefore, multiple architectures are mapped to the same embedding, which makes the continuous optimization techniques not feasible.

Chapter 5

Conclusion

5.1 Summary

In this thesis, we proposed a graph variational autoencoder that maps neural network architectures into a continuous and smooth latent space. Then we applied bayesian optimization in the learned latent space, in order to find high-performing architectures. In order to represent a neural network architecture, we capitalize on the graph structure of the architecture, and represent it as a directed acyclic graph, with edges correspond to data flow, and nodes to operations of the layers. Our main contribution is that we use **operation embeddings** instead of one-hot vectors in order to represent the operations of the layers. Therefore our model is able to learn the correlations between the operations, and eventually produce a smoother latent space, by mapping similar performing architectures to similar latent space embeddings. Our experiments demonstrate the effectiveness of operation embeddings in multiple concepts, as our model not only generate better performing architectures from the latent space, but also reduces the training time and the memory consumption.

5.2 Future Work

This thesis can be extended to many interesting future extensions. First of all, one direction is to study the structural properties that affects the performance of a neural network architecture. As with our approach we obtain a distribution of high-performing neural networks instead of a specific one, we can investigate the graph characteristics of these architectures. Our current findings suggest that the macro space, i.e. the wiring of the network, is an extremely important indicator of its performance [117]. This is contrary to the most neural network architectures approaches, that have a predefined macro space, and highly focus on the micro space. We hope to inspire more works in the field of Neural Networks from a Network Science perceptive, in order to help the explainability, design, search and generation of neural network architectures.

An another interesting research idea is to use operation embeddings and not only cap-

ture the relations between the operations, but generate completely new operation blocks. Therefore instead of searching in a predefined set of operations like convolution and max pooling, we can use an inverse transformation from the operation embedding space, and generate new operations. Also, a more straightforward approach in this direction is to generate the architecture by sequentially adding network motifs instead of single nodes, and consider these motifs as new operation blocks.

List of Figures

List of Tables

Bibliography

[1] *A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way — by Sumit Saha — Towards Data Science*. https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53. (Accessed on 10/30/2020).

[2] *A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way — by Sumit Saha — Towards Data Science*. https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53. (Accessed on 11/01/2020).

[3] Anuj. *Supervised learning pipeline*. URL: https://anujdutt9.github.io/machine-learning-basics.

[4] L. Backstrom and J. Leskovec. *Supervised Random Walks: Predicting and Recommending Links in Social Networks*. 2010. arXiv: 1011.4071 [cs.SI].

[5] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. *Neural Machine Translation by Jointly Learning to Align and Translate*. 2016. arXiv: 1409.0473 [cs.CL].

[6] Bowen Baker et al. *Accelerating Neural Architecture Search using Performance Prediction*. 2017. arXiv: 1705.10823 [cs.LG].

[7] Bowen Baker et al. *Designing Neural Network Architectures using Reinforcement Learning*. 2017. arXiv: 1611.02167 [cs.LG].

[8] Gabriel Bender et al. "Understanding and Simplifying One-Shot Architecture Search". In: ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. Stockholmsmässan, Stockholm Sweden: PMLR, Oct. 2018, pp. 550–559. URL: http://proceedings.mlr.press/v80/bender18a.html.

[9] Y. Bengio, P. Simard, and P. Frasconi. "Learning Long-Term Dependencies with Gradient Descent is Difficult". In: *Trans. Neur. Netw.* 5.2 (Mar. 1994), pp. 157–166. ISSN: 1045-9227. DOI: 10.1109/72.279181. URL: https://doi.org/10.1109/72.279181.

[10] Joel R Bock and David A Gough. "Predicting protein–protein interactions from primary structure". In: *Bioinformatics* 17.5 (2001), pp. 455–460.

[11] Samuel R. Bowman et al. *Generating Sentences from a Continuous Space*. 2016. arXiv: 1511.06349 [cs.LG].

[12] Xavier Bresson. *Recent Developments inGraph Network Architectures*. 2020. URL: https://www.dropbox.com/s/sig9x7orjl6c9qu/lecture15_recent_graph_neural_netorks.pdf.

[13] Tom B. Brown et al. *Language Models are Few-Shot Learners*. 2020. arXiv: 2005.14165 [cs.CL].

[14] Han Cai et al. *Efficient Architecture Search by Network Transformation*. 2017. arXiv: 1707.04873 [cs.LG].

[15] J Canny. "A Computational Approach to Edge Detection". In: *IEEE Trans. Pattern Anal. Mach. Intell.* 8.6 (June 1986), pp. 679–698. ISSN: 0162-8828. DOI: 10.1109/TPAMI.1986.4767851. URL: https://doi.org/10.1109/TPAMI.1986.4767851.

[16] Rich Caruana, Steve Lawrence, and Lee Giles. "Overfitting in Neural Nets: Backpropagation, Conjugate Gradient, and Early Stopping". In: *Proceedings of the 13th International Conference on Neural Information Processing Systems*. NIPS'00. Denver, CO: MIT Press, 2000, pp. 381–387.

[17] Kyunghyun Cho et al. *On the Properties of Neural Machine Translation: Encoder-Decoder Approaches*. 2014. arXiv: 1409.1259 [cs.CL].

[18] Junyoung Chung et al. *Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling*. 2014. arXiv: 1412.3555 [cs.NE].

[19] Connor W. Coley et al. "Convolutional Embedding of Attributed Molecular Graphs for Physical Property Prediction". In: *Journal of Chemical Information and Modeling* 57.8 (2017). PMID: 28696688, pp. 1757–1772. DOI: 10.1021/acs.jcim.6b00601. eprint: https://doi.org/10.1021/acs.jcim.6b00601. URL: https://doi.org/10.1021/acs.jcim.6b00601.

[20] Benoît Colson, Patrice Marcotte, and Gilles Savard. "An overview of bilevel optimization". In: *Annals of Operations Research* 153.1 (Sept. 2007), pp. 235–256. DOI: 10.1007/s10479-007-0176-2. URL: https://ideas.repec.org/a/spr/annopr/v153y2007i1p235-25610.1007-s10479-007-0176-2.html.

[21] Hanjun Dai et al. *Syntax-Directed Variational Autoencoder for Structured Data*. 2018. arXiv: 1802.08786 [cs.LG].

[22] *Deep Neural Networks As Computational Graphs — by Tyler Elliot Bettilyon — Teb's Lab — Medium*. https://medium.com/tebs-lab/deep-neural-networks-as-computational-graphs-867fcaa56c9. (Accessed on 11/04/2020).

[23] J. Deng et al. "ImageNet: A Large-Scale Hierarchical Image Database". In: *CVPR09*. 2009.

[24] Jacob Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2019. arXiv: 1810.04805 [cs.CL].

[25] Xuanyi Dong and Yi Yang. *NAS-Bench-201: Extending the Scope of Reproducible Neural Architecture Search*. 2020. arXiv: 2001.00326 [cs.CV].

[26] Daniel M Dunlavy, Tamara G Kolda, and Evrim Acar. "Temporal link prediction using matrix and tensor factorizations". In: *ACM Transactions on Knowledge Discovery from Data (TKDD)* 5.2 (2011), pp. 1–27.

[27] David Duvenaud et al. *Convolutional Networks on Graphs for Learning Molecular Fingerprints*. 2015. arXiv: 1509.09292 [cs.LG].

[28] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. *Efficient Multi-objective Neural Architecture Search via Lamarckian Evolution*. 2019. arXiv: 1804.09081 [stat.ML].

[29] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. *Neural Architecture Search: A Survey*. 2018. arXiv: 1808.05377 [stat.ML].

[30] Daniel C. Elton et al. "Deep learning for molecular design—a review of the state of the art". In: *Molecular Systems Design and Engineering* 4.4 (2019), pp. 828–849. ISSN: 2058-9689. DOI: 10.1039/c9me00039a. URL: http://dx.doi.org/10.1039/C9ME00039A.

[31] Santo Fortunato. "Community detection in graphs". In: *Physics Reports* 486.3 (2010), pp. 75–174. ISSN: 0370-1573. DOI: https://doi.org/10.1016/j.physrep.2009.11.002. URL: http://www.sciencedirect.com/science/article/pii/S0370157309002841.

[32] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. "Learning to forget: Continual prediction with LSTM". In: (1999).

[33] Golnaz Ghiasi et al. *NAS-FPN: Learning Scalable Feature Pyramid Architecture for Object Detection*. 2019. arXiv: 1904.07392 [cs.CV].

[34] Justin Gilmer et al. *Neural Message Passing for Quantum Chemistry*. 2017. arXiv: 1704.01212 [cs.LG].

[35] David E. Goldberg and Kalyanmoy Deb. "A Comparative Analysis of Selection Schemes Used in Genetic Algorithms". In: ed. by GREGORY J.E. RAWLINS. Vol. 1. Foundations of Genetic Algorithms. Elsevier, 1991, pp. 69–93. DOI: https://doi.org/10.1016/B978-0-08-050684-5.50008-2. URL: http://www.sciencedirect.com/science/article/pii/B9780080506845500082.

[36] Rafael Gómez-Bombarelli et al. "Automatic Chemical Design Using a Data-Driven Continuous Representation of Molecules". In: *ACS Central Science* 4.2 (Jan. 2018), pp. 268–276. ISSN: 2374-7951. DOI: 10.1021/acscentsci.7b00572. URL: http://dx.doi.org/10.1021/acscentsci.7b00572.

[37] William L. Hamilton. "Graph Representation Learning". In: *Synthesis Lectures on Artificial Intelligence and Machine Learning* 14.3 (), pp. 1–159.

[38] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV].

[39] Sepp Hochreiter. "The Vanishing Gradient Problem during Learning Recurrent Neural Nets and Problem Solutions". In: *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.* 6.2 (Apr. 1998), pp. 107–116. ISSN: 0218-4885. DOI: 10.1142/S0218488598000094. URL: https://doi.org/10.1142/S0218488598000094.

[40] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Computation* 9.8 (1997), pp. 1735–1780.

[41] K. Hornik, M. Stinchcombe, and H. White. "Multilayer Feedforward Networks Are Universal Approximators". In: *Neural Netw.* 2.5 (July 1989), pp. 359–366. ISSN: 0893-6080.

[42] Gao Huang et al. *Densely Connected Convolutional Networks*. 2018. arXiv: 1608.06993 [cs.CV].

[43] Zhiheng Huang, Wei Xu, and Kai Yu. "Bidirectional LSTM-CRF models for sequence tagging". In: *arXiv preprint arXiv:1508.01991* (2015).

[44] Ismaila Idris et al. "A combined negative selection algorithm–particle swarm optimization for an email spam detection system". In: *Engineering Applications of Artificial Intelligence* 39 (2015), pp. 33–44.

[45] *Intuitively Understanding Variational Autoencoders — by Irhum Shafkat — Towards Data Science*. https://towardsdatascience.com/intuitively-understanding-variational-autoencoders-1bfe67eb5daf. (Accessed on 11/09/2020).

[46] Wengong Jin, Regina Barzilay, and Tommi Jaakkola. *Junction Tree Variational Autoencoder for Molecular Graph Generation*. 2019. arXiv: 1802.04364 [cs.LG].

[47] Wengong Jin et al. *Learning Multimodal Graph-to-Graph Translation for Molecular Optimization*. 2019. arXiv: 1812.01070 [cs.LG].

[48] Bjrn H. Junker and Falk Schreiber. *Analysis of Biological Networks (Wiley Series in Bioinformatics)*. USA: Wiley-Interscience, 2008. ISBN: 0470041447.

[49] Kirthevasan Kandasamy et al. *Neural Architecture Search with Bayesian Optimisation and Optimal Transport*. 2018. arXiv: 1802.07191 [cs.LG].

[50] Kirthevasan Kandasamy et al. *Neural Architecture Search with Bayesian Optimisation and Optimal Transport*. 2019. arXiv: 1802.07191 [cs.LG].

[51] Diederik P Kingma and Max Welling. *Auto-Encoding Variational Bayes*. 2014. arXiv: 1312.6114 [stat.ML].

[52] Thomas N. Kipf and Max Welling. *Semi-Supervised Classification with Graph Convolutional Networks*. 2017. arXiv: 1609.02907 [cs.LG].

[53] Thomas N. Kipf and Max Welling. *Variational Graph Auto-Encoders*. 2016. arXiv: 1611.07308 [stat.ML].

[54] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. "CIFAR-10 (Canadian Institute for Advanced Research)". In: (). URL: http://www.cs.toronto.edu/~kriz/cifar.html.

[55] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. "CIFAR-100 (Canadian Institute for Advanced Research)". In: (). URL: http://www.cs.toronto.edu/~kriz/cifar.html.

[56] Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Neural Information Processing Systems* 25 (Jan. 2012). DOI: 10.1145/3065386.

[57] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems*. Ed. by F. Pereira et al. Vol. 25. Curran Associates, Inc., 2012, pp. 1097–1105. URL: https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[58] S. Kullback and R. A. Leibler. "On Information and Sufficiency". In: *Ann. Math. Statist.* 22.1 (Mar. 1951), pp. 79–86. DOI: 10.1214/aoms/1177729694. URL: https://doi.org/10.1214/aoms/1177729694.

[59] Matt J. Kusner, Brooks Paige, and José Miguel Hernández-Lobato. *Grammar Variational Autoencoder*. 2017. arXiv: 1703.01925 [stat.ML].

[60] Y. LeCun et al. "Backpropagation Applied to Handwritten Zip Code Recognition". In: *Neural Computation* 1 (1989), pp. 541–551.

[61] Jure Leskovec. *10-graph-gen.pdf*. http://web.stanford.edu/class/cs224w/slides/10-graph-gen.pdf. (Accessed on 11/10/2020).

[62] Jure Leskovec and Andrej Krevl. *SNAP Datasets: Stanford Large Network Dataset Collection*. http://snap.stanford.edu/data. June 2014.

[63] Jian Li et al. *Neural Architecture Optimization with Graph VAE*. 2020. arXiv: 2006.10310 [cs.LG].

[64] Liam Li and Ameet Talwalkar. *Random Search and Reproducibility for Neural Architecture Search*. 2019. arXiv: 1902.07638 [cs.LG].

[65] Lisha Li et al. *Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization*. 2018. arXiv: 1603.06560 [cs.LG].

[66] Yujia Li et al. *Learning Deep Generative Models of Graphs*. 2018. arXiv: 1803.03324 [cs.LG].

[67] Renjie Liao et al. "Efficient Graph Generation with Graph Recurrent Attention Networks". In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach et al. Vol. 32. Curran Associates, Inc., 2019, pp. 4255–4265. URL: https://proceedings.neurips.cc/paper/2019/file/d0921d442ee91b896ad95059d13df618-Paper.pdf.

[68] David Liben-Nowell and Jon Kleinberg. "The Link Prediction Problem for Social Networks". In: *Proceedings of the Twelfth International Conference on Information and Knowledge Management*. CIKM '03. New Orleans, LA, USA: Association for Computing Machinery, 2003, pp. 556–559. ISBN: 1581137230. DOI: 10.1145/956863.956972. URL: https://doi.org/10.1145/956863.956972.

[69] David Liben-Nowell and Jon Kleinberg. "The Link-Prediction Problem for Social Networks". In: *J. Am. Soc. Inf. Sci. Technol.* 58.7 (May 2007), pp. 1019–1031. ISSN: 1532-2882.

[70] Chenxi Liu et al. *Progressive Neural Architecture Search*. 2018. arXiv: 1712.00559 [cs.CV].

[71] Hanxiao Liu, Karen Simonyan, and Yiming Yang. *DARTS: Differentiable Architecture Search*. 2018. arXiv: 1806.09055 [cs.LG].

[72] Hanxiao Liu, Karen Simonyan, and Yiming Yang. *DARTS: Differentiable Architecture Search*. 2019. arXiv: 1806.09055 [cs.LG].

[73] Hanxiao Liu et al. *Hierarchical Representations for Efficient Architecture Search*. 2018. arXiv: 1711.00436 [cs.LG].

[74] Ziqi Liu et al. *Heterogeneous Graph Neural Networks for Malicious Account Detection*. 2020. arXiv: 2002.12307 [cs.LG].

[75] Xiaoyu Lu et al. "Structured Variationally Auto-encoded Optimization". In: ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. Stockholmsmässan, Stockholm Sweden: PMLR, Oct. 2018, pp. 3267–3275. URL: http://proceedings.mlr.press/v80/lu18c.html.

[76] Jovita Lukasik et al. *Neural Architecture Performance Prediction Using Graph Neural Networks*. 2020. arXiv: 2010.10024 [cs.CV].

[77] Renqian Luo et al. *Neural Architecture Optimization*. 2019. arXiv: 1808.07233 [cs.LG].

[78] Laurens van der Maaten and Geoffrey Hinton. "Visualizing Data using t-SNE". In: *Journal of Machine Learning Research* 9 (2008), pp. 2579–2605. URL: http://www.jmlr.org/papers/v9/vandermaaten08a.html.

[79] Miller McPherson, Lynn Smith-Lovin, and James M Cook. "Birds of a Feather: Homophily in Social Networks". In: *Annual Review of Sociology* 27.1 (2001), pp. 415–444. DOI: 10.1146/annurev.soc.27.1.415. eprint: https://doi.org/10.1146/annurev.soc.27.1.415. URL: https://doi.org/10.1146/annurev.soc.27.1.415.

[80] Tomas Mikolov et al. "Distributed Representations of Words and Phrases and their Compositionality". In: *Advances in Neural Information Processing Systems*. Ed. by C. J. C. Burges et al. Vol. 26. Curran Associates, Inc., 2013, pp. 3111–3119. URL: https://proceedings.neurips.cc/paper/2013/file/9aa42b31882ec039965f3c4923ce901b-Paper.pdf.

[81] Dang Lien Minh et al. "Deep learning approach for short-term stock trends prediction based on two-stream gated recurrent unit network". In: *Ieee Access* 6 (2018), pp. 55392–55404.

[82] Vinod Nair and Geoffrey E. Hinton. "Rectified Linear Units Improve Restricted Boltzmann Machines." In: *ICML*. Ed. by Johannes Fürnkranz and Thorsten Joachims. Omnipress, 2010, pp. 807–814. URL: http://dblp.uni-trier.de/db/conf/icml/icml2010.html#NairH10.

[83] Ngoc Giang Nguyen et al. "DNA sequence classification by convolutional neural network". In: *Journal of Biomedical Science and Engineering* 9.05 (2016), p. 280.

[84] Chenhao Niu et al. *Permutation Invariant Graph Generation via Score-Based Generative Modeling*. 2020. arXiv: 2003.00638 [cs.LG].

[85] Philipp Pagel et al. "The MIPS mammalian protein–protein interaction database". In: *Bioinformatics* 21.6 (Nov. 2004), pp. 832–834. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/bti115. eprint: https://academic.oup.com/bioinformatics/article-pdf/21/6/832/508331/bti115.pdf. URL: https://doi.org/10.1093/bioinformatics/bti115.

[86] Byeonghwa Park and Jae Kwon Bae. "Using machine learning algorithms for housing price prediction: The case of Fairfax County, Virginia housing data". In: *Expert Systems with Applications* 42.6 (2015), pp. 2928–2934.

[87] Hieu Pham et al. *Efficient Neural Architecture Search via Parameter Sharing*. 2018. arXiv: 1802.03268 [cs.LG].

[88] Esteban Real et al. *AutoML-Zero: Evolving Machine Learning Algorithms From Scratch*. 2020. arXiv: 2003.03384 [cs.LG].

[89] Esteban Real et al. *Large-Scale Evolution of Image Classifiers*. 2017. arXiv: 1703.01041 [cs.NE].

[90] Esteban Real et al. *Regularized Evolution for Image Classifier Architecture Search*. 2019. arXiv: 1802.01548 [cs.NE].

[91] Shaoqing Ren et al. *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*. 2016. arXiv: 1506.01497 [cs.CV].

[92] *RNN, Talking about Gated Recurrent Unit - DLBT — Deep learning benchmark tool*. https://technopremium.com/blog/rnn-talking-about-gated-recurrent-unit/. (Accessed on 11/02/2020).

[93] Ryan A. Rossi and Nesreen K. Ahmed. "The Network Data Repository with Interactive Graph Analytics and Visualization". In: *AAAI*. 2015. URL: http://networkrepository.com.

[94] Martin Rosvall and Carl T. Bergstrom. "Maps of random walks on complex networks reveal community structure". In: *Proceedings of the National Academy of Sciences* 105.4 (2008), pp. 1118–1123. ISSN: 0027-8424. DOI: 10.1073/pnas.0706851105. eprint: https://www.pnas.org/content/105/4/1118.full.pdf. URL: https://www.pnas.org/content/105/4/1118.

[95] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. "Learning Representations by Back-Propagating Errors". In: *Neurocomputing: Foundations of Research*. Cambridge, MA, USA: MIT Press, 1988, pp. 696–699. ISBN: 0262010976.

[96] Frederic Runge et al. *Learning to Design RNA*. 2019. arXiv: 1812.11951 [cs.LG].

[97] "Lee Douglas Sailer". ""Structural equivalence: Meaning and definition, computation and application"". In: *"Social Networks"* "1"."1" ("1978"), "73–90". ISSN: "0378-8733". DOI: "https://doi.org/10.1016/0378-8733(78)90014-X". URL: http://www.sciencedirect.com/science/article/pii/037887337890014X.

[98] Ismael Sanchez. "Short-term prediction of wind energy production". In: *International Journal of Forecasting* 22.1 (2006), pp. 43–56.

[99] Franco Scarselli et al. "The Graph Neural Network Model". In: *Trans. Neur. Netw.* 20.1 (Jan. 2009), pp. 61–80. ISSN: 1045-9227. DOI: 10.1109/TNN.2008.2005605. URL: https://doi.org/10.1109/TNN.2008.2005605.

[100] John P. Scott and Peter J. Carrington. *The SAGE Handbook of Social Network Analysis*. Sage Publications Ltd., 2011. ISBN: 1847873952.

[101] Martin Simonovsky and Nikos Komodakis. *GraphVAE: Towards Generation of Small Graphs Using Variational Autoencoders*. 2018. arXiv: 1802.03480 [cs.LG].

[102] Edward Snelson and Zoubin Ghahramani. "Sparse Gaussian Processes using Pseudo-inputs". In: *Advances in Neural Information Processing Systems*. Ed. by Y. Weiss, B. Schölkopf, and J. Platt. Vol. 18. MIT Press, 2006, pp. 1257–1264. URL: https://proceedings.neurips.cc/paper/2005/file/4491777b1aa8b5b32c2e8666dbe1a495-Paper.pdf.

[103] Masanori Suganuma, Shinichi Shirakawa, and Tomoharu Nagao. *A Genetic Programming Approach to Designing Convolutional Neural Network Architectures*. 2017. arXiv: 1704.00764 [cs.NE].

[104] Martin Sundermeyer, Ralf Schlüter, and Hermann Ney. "LSTM neural networks for language modeling". In: *Thirteenth annual conference of the international speech communication association*. 2012.

[105] Kevin Swersky, Jasper Snoek, and Ryan Prescott Adams. *Freeze-Thaw Bayesian Optimization*. 2014. arXiv: 1406.3896 [stat.ML].

[106] Kevin Swersky et al. *Raiders of the Lost Architecture: Kernels for Bayesian Optimization in Conditional Parameter Spaces*. 2014. arXiv: 1409.4011 [stat.ML].

[107] Christian Szegedy et al. *Rethinking the Inception Architecture for Computer Vision*. 2015. arXiv: 1512.00567 [cs.CV].

[108] *Translation Invariance in Convolutional Neural Networks — by Divyanshu Soni — Medium*. https://medium.com/@divsoni2012/translation-invariance-in-convolutional-neural-networks-61d9b6fa03df. (Accessed on 10/30/2020).

[109] *Understanding LSTM Networks – colah's blog*. https://colah.github.io/posts/2015-08-Understanding-LSTMs/. (Accessed on 11/02/2020).

[110] *Understanding RNN and LSTM. What is Neural Network? — by Aditi Mittal — Towards Data Science*. https://towardsdatascience.com/understanding-rnn-and-lstm-f7cdf6dfc14e. (Accessed on 11/02/2020).

[111] S. V. N. Vishwanathan et al. *Graph Kernels*. 2008. arXiv: 0807.0093 [cs.LG].

[112] Chun Wang et al. "MGAE: Marginalized Graph Autoencoder for Graph Clustering". In: Nov. 2017, pp. 889–898. DOI: 10.1145/3132847.3132967.

[113] David Weininger. "SMILES, a chemical language and information system. 1. Introduction to methodology and encoding rules". In: *Journal of Chemical Information and Computer Sciences* 28.1 (1988), pp. 31–36. DOI: 10.1021/ci00057a005. eprint: https://pubs.acs.org/doi/pdf/10.1021/ci00057a005. URL: https://pubs.acs.org/doi/abs/10.1021/ci00057a005.

[114] Tom White. *Sampling Generative Networks*. 2016. arXiv: 1609.04468 [cs.NE].

[115] Ronald J. Williams. "Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning". In: *Mach. Learn.* 8.3–4 (May 1992), pp. 229–256. ISSN: 0885-6125. DOI: 10.1007/BF00992696. URL: https://doi.org/10.1007/BF00992696.

[116] Martin Wistuba, Ambrish Rawat, and Tejaswini Pedapati. *A Survey on Neural Architecture Search*. 2019. arXiv: 1905.01392 [cs.LG].

[117] Mitchell Wortsman, Ali Farhadi, and Mohammad Rastegari. *Discovering Neural Wirings*. 2019. arXiv: 1906.00586 [cs.LG].

[118] Lingxi Xie and Alan Yuille. *Genetic CNN*. 2017. arXiv: 1703.01513 [cs.CV].

[119] Saining Xie et al. *Exploring Randomly Wired Neural Networks for Image Recognition*. 2019. arXiv: 1904.01569 [cs.CV].

[120] Sirui Xie et al. *SNAS: Stochastic Neural Architecture Search*. 2020. arXiv: 1812.09926 [cs.LG].

[121] Keyulu Xu et al. *How Powerful are Graph Neural Networks?* 2019. arXiv: 1810.00826 [cs.LG].

[122] Shen Yan et al. *Does Unsupervised Architecture Representation Learning Help Neural Architecture Search?* 2020. arXiv: 2006.06936 [cs.CV].

[123] Jianwei Yang et al. "Graph R-CNN for Scene Graph Generation". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. Sept. 2018.

[124] Chris Ying et al. *NAS-Bench-101: Towards Reproducible Neural Architecture Search*. 2019. arXiv: 1902.09635 [cs.LG].

[125] Jiaxuan You et al. *Graph Convolutional Policy Network for Goal-Directed Molecular Graph Generation*. 2019. arXiv: 1806.02473 [cs.LG].

[126] Jiaxuan You et al. *GraphRNN: Generating Realistic Graphs with Deep Auto-regressive Models*. 2018. arXiv: 1802.08773 [cs.LG].

[127] Jiaxuan You et al. "GraphRNN: Generating Realistic Graphs with Deep Auto-regressive Models". In: ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. Stockholmsmässan, Stockholm Sweden: PMLR, Oct. 2018, pp. 5708–5717. URL: http://proceedings.mlr.press/v80/you18a.html.

[128] Chris Zhang, Mengye Ren, and Raquel Urtasun. *Graph HyperNetworks for Neural Architecture Search*. 2019. arXiv: 1810.05749 [cs.LG].

[129] Muhan Zhang et al. *D-VAE: A Variational Autoencoder for Directed Acyclic Graphs*. 2019. arXiv: 1904.11088 [cs.LG].

[130] Rui Zhao et al. "Machine health monitoring using local feature-based gated recurrent unit networks". In: *IEEE Transactions on Industrial Electronics* 65.2 (2017), pp. 1539–1548.

[131] Zhao Zhong et al. *BlockQNN: Efficient Block-wise Neural Network Architecture Generation*. 2018. arXiv: 1808.05584 [cs.CV].

[132] Zhao Zhong et al. *Practical Block-wise Neural Network Architecture Generation*. 2018. arXiv: 1708.05552 [cs.CV].

[133] Zhenpeng Zhou et al. "Optimization of Molecules via Deep Reinforcement Learning". In: *Scientific Reports* 9.1 (July 2019). ISSN: 2045-2322. DOI: 10.1038/s41598-019-47148-x. URL: http://dx.doi.org/10.1038/s41598-019-47148-x.

[134] Barret Zoph and Quoc V Le. "Neural architecture search with reinforcement learning". In: *arXiv preprint arXiv:1611.01578* (2016).

[135] Barret Zoph and Quoc V. Le. *Neural Architecture Search with Reinforcement Learning*. 2017. arXiv: 1611.01578 [cs.LG].

[136] Barret Zoph et al. *Learning Transferable Architectures for Scalable Image Recognition*. 2018. arXiv: 1707.07012 [cs.CV].