Εθνικο Μετσοβιο Πολυτεχνειο
Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων
Τομεας Επικοινωνιων, Ηλεκτρονικης και Συστηματων Πληροφορικης

Μελέτη και Υλοποίηση Εργαλείων Μηχανικής Μάθησης στην Τομογραφία Δικτύων Υπολογιστών

Διπλωματικη Εργασια

Μιχαηλ Καλντης

Επιβλέπων: Συμεών Παπαβασιλείου
Καθηγητής Ε.Μ.Π

Εργαστηριο Διαχειρισης και Βελτιστου Σχεδιασμου Δικτυων Τηλεματικης
Αθήνα, Μάιος 2021

Εθνικο Μετσοβιο Πολυτεχνειο
Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων
Τομεας Επικοινωνιων, Ηλεκτρονικης και Συστηματων Πληροφορικης

Μελέτη και Υλοποίηση Εργαλείων Μηχανικής Μάθησης στην Τομογραφία Δικτύων Υπολογιστών

Διπλωματικη Εργασια

Μιχαηλ Καλντης

Επιβλέπων: Συμεών Παπαβασιλείου
Καθηγητής Ε.Μ.Π

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 11η Μαΐου 2021

(Υπογραφή)	*(Υπογραφή)*	*(Υπογραφή)*
.....	.....	.....
Συμεών Παπαβασιλείου	Θεοδώρα Βαρβαρίγου	Ιωάννα Ρουσσάκη
Καθηγήτης Ε.Μ.Π.	Καθηγήτρια Ε.Μ.Π	Επ. Καθηγήτρια Ε.Μ.Π

Εργαστηριο Διαχειρισης και Βελτιστου Σχεδιασμου Δικτυων Τηλεματικης
Αθήνα, Μάιος 2021

Εθνικο Μετσοβιο Πολυτεχνειο

Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων

Τομεας Επικοινωνιων, Ηλεκτρονικης και Συστηματων Πληροφορικης

(Υπογραφή)

.....

Μιχαήλ Καλντής

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

National Technical University of Athens

School of Electrical and Computer Engineering
Division of Communication, Electronic and Information Engineering

Analysis and Design of efficient Network Tomography Mechanisms with Machine Learning Tools

Diploma Thesis

Michail Kalntis

Supervisor: Symeon Papavassiliou
Professor, National Technical University of Athens

NETwork Management & Optimal DEsign laboratory (NETMODE)

Athens, May 2021

Περίληψη

Ο πίνακας κυκλοφορίας (Traffic Matrix) περιέχει το ποσό της κυκλοφορίας (σε πακέτα, ή bytes) μεταξύ όλων τον κόμβων ενός δικτύου. Δυστυχώς, η απευθείας μέτρηση των υπερβολικά πολλών πακέτων ή δεδομένων που διασχίζουν ένα δίκτυο είναι ασύμφορη, λόγω της μεγάλης χωρητικότητας και των πολλαπλών επεξεργαστικών μονάδων που χρειάζεται να λειτουργούν αδιάκοπα. Ωστόσο, ακόμα και με πολύ ισχυρά υπολογιστικά συστήματα, τα λάθη δεν είναι εφικτό να εξαλειφθούν και σημαντικά δεδομένα μπορεί να χαθούν. Μια λιγότερο κοστοβόρα προσέγγιση είναι η χρήση δεδομένων από τις ζεύξεις, τα οποία είναι εφικτό να αποκτηθούν ευκολότερα με τη χρήση του πρωτοκόλλου SNMP. Οι προαναφερθείσες μετρήσεις περιέχουν έμμεση πληροφορία για τον πίνακα κυκλοφορίας, οπότε ο τελευταίος μπορεί να εξαχθεί από αυτές. Η παρούσα διπλωματική εργασία εστιάζει στο πρόβλημα της Εκτίμησης Πίνακα κυκλοφορίας (Traffic Matrix Estimation), που ανήκει στην Τομογραφία Δικτύου (Network Tomography). Η προσέγγιση της διπλωματικής αυτής περιλαμβάνει την αξιοποίηση ενός γεννητικού μοντέλου μάθησης, που ονομάζεται Παραλλαγμένος Αυτοκωδικοποιητής (Variational Autoencoder), για να λύσει το προαναφερθέν κακώς-διατυπωμένο (ill-posed) πρόβλημα. Το πρόβλημα θεωρείται κακώς-διατυπωμένο, διότι ο πίνακας δρομολόγησης (routing matrix) δεν είναι full-rank, με αποτέλεσμα ο αριθμός των μεταβλητών να είναι πολύ μεγαλύτερος από τον αριθμό των εξισώσεων. Πιο αναλυτικά, εκπαιδεύουμε τον Αυτοκωδικοποιητή με προηγούμενα δεδομένα και αξιοποιούμε τον εκπαιδευμένο κωδικοποιητή για να μετατρέψουμε το πρόβλημά μας σε ένα πρόβλημα ελαχιστοποίησης σε έναν λανθάνοντα χώρο (latent space), το οποίο πλέον μπορεί να λυθεί εφαρμόζοντας έναν βελτιστοποιητή βασισμένο στην παράγωγο-κλίση (gradient-based optimizer). Επιπρόσθετα, ο εκπαιδευμένος κωδικοποιητής αξιοποιείται για να παράγει πίνακες κυκλοφορίας, οι οποίοι έχουν παρόμοια χαρακτηριστικά με τους υπόλοιπους που έχει εξετάσει. Τέλος, ερευνάται η σταδιακή βελτιστοποίηση (incremental optimization), η οποία κάνει χρήση των κωδικοποιητών όπως προκύπτουν κατά την εκπαίδευση του αυτοκωδικοποιητή. Τέλος, πραγματοποιείται αξιολόγηση των προτεινόμενων μεθόδων, χρησιμοποιώντας δεδομένα πινάκων κυκλοφορίας από ένα πραγματικό δίκτυο.

Λέξεις Κλειδιά:

τομογραφία δικτύου, παρακολούθηση δικτύου, πίνακας κυκλοφορίας, εκτίμηση πίνακα κυκλοφορίας, σύνθεση πίνακα κυκλοφορίας, μηχανική μάθηση, μάθηση γνωρισμάτων, βαθιά μάθηση, νευρωνικά δίκτυα, γεννητικά μοντέλα, αυτοκωδικοποιητής, παραλλαγμένος αυτοκωδικοποιητής

Abstract

Traffic Matrix (TM) captures the amount of traffic between all nodes in a network. Since a variety of network operations, such as anomaly detection and network optimization, require accurate traffic matrices as inputs, it is becoming increasingly important for operators of these systems to acquire them. Unfortunately, measuring directly the vast number of packets that traverse a network is extremely expensive, due to the enormous demanded capacity, as well as the numerous processing units that have to function incessantly. Even with very powerful systems, errors cannot be completely avoided and crucial data is likely to be missed. A less costly approach is the usage of easily obtainable link counts through SNMP measurements, which implicitly contain information about traffic matrix. If so, TM has to be inferred. In this diploma thesis, the problem of Traffic Matrix Estimation (TME) is addressed, which belongs in the class of Path-Level Network Tomography. Our approach is to take advantage of deep generative models and more precisely, Variational Autoencoder (VAE) to deal with this highly under-constrained issue. In particular, we train the VAE with historical data (previously observed TMs) and we leverage the trained decoder to transform TME into a minimization problem in the latent space, which can be in turn solved by employing a gradient-based optimizer. Furthermore, the trained decoder can be used for synthesizing traffic matrices, i.e., for generating synthetic TM examples that have "similar" properties to the samples of the training set. Finally, we explore the incremental optimization of the sequence of objectives constructed from the sequence of decoders that we obtain at different stages of the VAE training. The performance of the proposed methods is evaluated using a publicly available dataset of actual traffic matrices recorded in a real backbone network.

Keywords:

network tomography, network monitoring, traffic matrix, traffic matrix estimation, traffic matrix synthesis, machine learning, representation learning, deep learning, neural network, generative model, autoencoder, variational autoencoder

Ευχαριστίες

Η συγγραφή της παρούσας Διπλωματικής Εργασίας σηματοδοτεί την ολοκλήρωση των προπτυχιακών μου σπουδών. Πριν κλείσει το μεγάλο αυτό κεφάλαιο της ζωής μου, θα ήθελα να ευχαριστήσω τα άτομα που στάθηκαν δίπλα μου και συνέβαλαν στην μέχρι τώρα πορεία μου.

Αρχικά, θα ήθελα να ευχαριστήσω τον Καθηγητή μου, κ. Συμεών Παπαβασιλείου για την πολύτιμη βοήθειά του στην εκπόνηση του παρόντος θέματος. Ο κ. Παπαβασιλείου υπήρξε ένας καταπληκτικός μέντορας, ο οποίος με τις ιδέες του και την προθυμότητά του, με ενέπνευσε και με καθοδήγησε όλους αυτούς τους μήνες. Επιπλέον του είμαι βαθιά ευγνώμων για την πολύτιμη στήριξή του στα μελλοντικά μου σχέδια. Ήταν τιμή μου να εργαστώ μαζί του στο εργαστήριο Διαχείρισης και Βέλτιστου Σχεδιασμού Δικτύων Τηλεματικής του Εθνικού Μετσόβιου Πολυτεχνείου.

Επιπλέον, θα ήθελα να ευχαριστήσω τον υποψήφιο διδάκτορα κ. Γρηγόρη Κακκάβα και τον αναπληρωτή καθηγητή κ. Βασίλειο Καρυώτη. Η συμβολή τους ήταν καθοριστική και διαρκής, χωρίς την οποία δεν θα ήταν δυνατή η ολοκλήρωση της παρούσας εργασίας. Δε θα μπορούσε να φανταστεί κανείς καλύτερους ανθρώπους να συμβουλευτεί.

Επίσης, θα ήθελα να εκφράσω την ευγνωμοσύνη μου στους φίλους μου, με τους οποίους έχω μοιραστεί πολλές από τις πιο ξεχωριστές στιγμές της ζωής μου. Ήταν αυτοί που μου χάρισαν και θα μου χαρίζουν στιγμές ανεμελιάς, άφθονου γέλιου και αγάπης.

Είναι αλήθεια ότι δε θα αισθανόμουν τόσο ολοκληρωμένος άνθρωπος και δε θα είχα κάνει πραγματικότητα τα όνειρά μου χωρίς την αγάπη και τη φροντίδα της οικογένειάς μου. Με στηρίζει σε κάθε μου βήμα, με συμβουλεύει και μου προσφέρει ανεκτίμητα ψυχικά αγαθά. Θα είμαι πάντα κοντά τους. Θα ήθελα να αφιερώσω σε αυτούς την παρούσα εργασία.

Μιχαήλ Καλντής
Αθήνα, Μάιος 2021

Contents

Κατάλογος Σχημάτων

Κατάλογος Πινάκων

Εκτεταμένη Περίληψη

Εισαγωγή

Από τις πρώτες μέρες του ως πειραματικό δίκτυο αλλαγής πακέτων, το Διαδίκτυο έχει προχωρήσει πολύ σε σύντομο χρονικό διάστημα. Σε αντίθεση με το τηλεφωνικό δίκτυο που εξελίχθηκε με πιο αργό τρόπο, το Διαδίκτυο κυριάρχησε ταχέως [2] λόγω της καινοτόμου πρακτικότητάς του και του αποκεντρωμένου ελέγχου του, και, σήμερα, είναι το βασικό μέσο επικοινωνίας. Το 2021, περισσότεροι από 4,7 δισεκατομμύρια άνθρωποι χρησιμοποιούν το Διαδίκτυο κάθε μέρα [3]. Με την έλευση νέων εφαρμογών όπως η ροή βίντεο και την ταχεία αύξηση της κίνησης δεδομένων από κινητές συσκευές, παρατηρούμε μια παγκόσμια έκρηξη πληροφοριών.

Δεδομένης της αυξανόμενης σημασίας του Διαδικτύου, η γνώση της κίνησής του έχει γίνει όλο και πιο σημαντική. Ωστόσο, λόγω της έλλειψης κεντρικού ελέγχου, η ποσοτική αξιολόγηση της απόδοσης του δικτύου γίνεται πολύ δύσκολη. Όχι μόνο είναι διοικητικά και υπολογιστικά ασύμφορο για κάθε σταθμό να συλλέγει, να επεξεργάζεται και να μεταδίδει αυτές τις πληροφορίες, αλλά και οι γνώσεις αυτές μπορεί να θεωρηθούν ιδιωτικές και μη κοινοποιήσιμες από τους παρόχους. Αυτά τα δεδομένα είναι ζωτικής σημασίας για τους χειριστές δικτύου, οι οποίοι θέλουν να προγραμματίσουν κρίσιμες εργασίες, όπως να καθορίσουν τις πολιτικές δρομολόγησης ή να διαχειριστούν και να προγραμματίσουν το δίκτυο [4]. Τα προαναφερθέντα έχουν μεγάλη σημασία για την επίτευξη του στόχου της εξυπηρέτησης της ζήτησης με ικανοποιητική ποιότητα όσον αφορά την καθυστέρηση, το ποσοστό απώλειας πακέτων ή/και άλλες παραμέτρους ποιότητας υπηρεσίας/ποιότητας εμπειρίας.

Η Τομογραφία Δικτύου (Network Tomography) [5]–[7] αναπτύχθηκε για να ξεπεράσει τα προαναφερθέντα εμπόδια. Σκοπός της είναι να συναγάγει σημαντικά στατιστικά στοιχεία ενός δικτύου, χωρίς να χρειάζεται πλήρη συνεργασία ενδιάμεσων κόμβων. Ταυτόχρονα, οι πρόσφατες εξελίξεις στα βαθιά νευρωνικά δίκτυα [8], σε συνδυασμό με την πρόοδο στις στοχαστικές μεθόδους βελτιστοποίησης, επέτρεψαν την εξαγωγή σημαντικών και πιο περίπλοκων χαρακτηριστικών από τα διαθέσιμα δεδομένα. Με την αξιοποίηση και την ενσωμάτωση της νοημοσύνης αυτών των μοντέλων σε κατάλληλο περιβάλλον, είναι δυνατόν να αντιμετωπιστούν πολλά υποπροβλήματα της Τομογραφίας Δικτύου, συμπεριλαμβανομένης της Εκτίμησης Πίνακα Κυκλοφορίας (Traffic Matrix Estimation), η οποία είναι το βασικό θέμα αυτής της εργασίας και συμπεραίνει τα χαρακτηριστικά ενός δικτύου που δεν μπορούν να μετρηθούν άμεσα.

Κίνητρα

Το Διαδίκτυο αποτελείται από ετερογενείς συσκευές, όπως κινητά τηλέφωνα, δρομολογητές και υπολογιστές, με διαφορετικά πρωτόκολλα και λειτουργικά συστήματα. Σε αυτό το χαοτικό περιβάλλον, μία από τις πιο δύσκολες εργασίες είναι η εξερεύνηση της τεράστιας γκάμας τηλεπικοινωνιών και υπολογιστικών υποδομών και η κατανόηση των χαρακτηριστικών της, όπως η συνδεσιμότητα, το εύρος ζώνης και οι καθυστερήσεις. Οι χάρτες [9] είναι επομένως ένας από τους πιο χρήσιμους πόρους για την αποκρυπτογράφηση της μεγάλης και περίπλοκης υποδομής των τεχνολογιών πληροφοριών και επικοινωνιών, δίνοντάς μας τη δυνατότητα να κατανοήσουμε πού βρίσκονται αυτές οι τεχνολογίες, πώς διασυνδέονται, πόσες πληροφορίες μεταδίδονται ή χάνονται κλπ.

Το ping [10] ήταν μια από τις πρώτες προσπάθειες για την κατανόηση των χαρακτηριστικών του Διαδικτύου. Λειτουργεί στέλνοντας πακέτα ICMP σε έναν συγκεκριμένο προορισμό και χρησιμοποιείται κυρίως για τον έλεγχο της συνδεσιμότητας μιας συσκευής στο δίκτυο και τη μέτρηση της καθυστέρησης μεταξύ δύο υπολογιστών. Το 1987 δημιουργήθηκε ένα πιο λεπτομερές διαγνωστικό εργαλείο, που ονομάζεται traceroute [11]. Το traceroute προσδιορίζει τη διαδρομή μεταξύ ενός σημείου εκκίνησης και ενός σημείου προορισμού και τη διάρκεια για αυτό το ταξίδι (μετ΄ επιστροφής).

Ωστόσο, θα ήταν αφελές να βασιζόμασταν αποκλειστικά σε αυτά τα βοηθητικά προγράμματα λογισμικού για την παροχή ενός χαρτογραφήματος του διαδικτύου, επειδή αυτές οι τεχνικές απαιτούν τη συνεργασία ενδιάμεσων διακομιστών και δρομολογητών. Πιο συγκεκριμένα, κατά τη διαδρομή μεταξύ δύο ζευγών κόμβων, είναι πιθανό ορισμένα σημεία να μπλοκάρουν ή να αντιμετωπίζουν με πολύ χαμηλή προτεραιότητα την κίνηση που χρησιμοποιείται για διαγνωστικούς σκοπούς (κυρίως για λόγους ασφάλειας και απόδοσης), καθιστώντας την όλη διαδικασία ανεπιτυχή. Έτσι, οι χάρτες [9] που είναι διαθέσιμοι σήμερα περιέχουν μόνο μια μερική απεικόνιση του κυβερνοχώρου.

Επιπλέον, ακόμη και αν ήταν εφικτές τέτοιες απεικονίσεις, κάθε πάροχος έχει το δικαίωμα να αποφασίσει εάν θα κοινοποιήσει επιχειρησιακές πληροφορίες σχετικά, για παράδειγμα, με καθυστερήσεις και διανομή κυκλοφορίας του δικτύου του, και πρέπει να εξετάσει πιθανά μειονεκτήματα αυτής της ενέργειας. Αρχικά, η μετάδοση αυτών των στατιστικών προκαλεί σημαντική αύξηση της ροής της κυκλοφορίας. Εάν το δίκτυο δεν έχει κατασκευαστεί για να υποστηρίζει αυτόν τον όγκο δεδομένων, το σύστημα ενδέχεται να απορρίψει χρήσιμα πακέτα για την αποφυγή συμφόρησης και ακόμα και να καταρρεύσει. Επιπλέον, είναι πιθανό οι κακόβουλοι χρήστες να αναλύσουν αυτές τις λεπτομέρειες και να εκμεταλλευτούν τις ευπάθειες ενός συστήματος.

Ως εκ τούτου, οι επιστήμονες έψαχναν για μια άλλη προσέγγιση, η οποία δεν θα χρειαζόταν τη συνεργασία ενδιάμεσων κόμβων και δεν θα υπερφόρτωνε τους κόμβους του δικτύου. Το πεδίο που προέκυψε ονομάστηκε Network Tomography από τον Vardi [5], λόγω της ομοιότητάς του με την ιατρική τομογραφία, και αναφέρεται στο αντίστροφο [12] πρόβλημα της εκτίμησης των τιμών των παραμέτρων που χαρακτηρίζουν το υπό έρευνα σύστημα, λαμβάνοντας υπόψη τα αποτελέσματα των πραγματικών παρατηρήσεων. Παρόμοια προβλήματα έχουν αντιμετωπιστεί διεξοδικά στην επεξεργασία σήματος

[13], στην όραση υπολογιστών [14], στην ιατρική απεικόνιση [15] και άλλες εφαρμογές. Επομένως, υιοθετήθηκαν ορισμένα χρήσιμα συμπεράσματα για να ριχτεί φως στο σχετικά νέο κλάδο της Τομογραφίας Δικτύου.

Συνεισφορές

Οι κύριες συνεισφορές της παρούσας διπλωματικής εργασίας είναι οι εξής [16]:

- Επισκόπηση της βιβλιογραφίας σχετικά με τις συνεισφορές στην Τομογραφία Δικτύου και την Εκτίμηση Πίνακα Κυκλοφορίας.
- Σύνθεση πινάκων κυκλοφορίας που συμμορφώνονται με τους παρατηρούμενους περιορισμούς ενός συγκεκριμένου δικτύου, εκπαιδεύοντας έναν Παραλλαγμένο Αυτοκωδικοποιητή.
- Πρώτα, εκπαίδευση ενός Παραλλαγμένου Αυτοκωδικοποιητή και έπειτα, επίλυση του προβλήματος βελτιστοποίησης στον λανθάνοντα χώρο για να την αντιμετώπιση του προβλήματος Εκτίμησης Πίνακα Κυκλοφορίας.
- Μετατροπή της προαναφερθείσας μεθόδου Εκτίμησης Πίνακα Κυκλοφορίας δύο σταδίων (πρώτα εκπαίδευση Παραλλαγμένου Αυτοκωδικοποιητή και στη συνέχεια, βελτιστοποίηση στον λανθάνοντα χώρο) σε έναν τύπο «ταυτόχρονης» βελτιστοποίησης.
- Αξιολόγηση της απόδοσης των προτεινόμενων μεθόδων, χρησιμοποιώντας ένα διαθέσιμο σύνολο δεδομένων πραγματικών πινάκων κυκλοφορίας που έχουν καταγραφεί στο δίκτυο Abilene [17].

Παραλλαγμένος Αυτοκωδικοποιητής

Ο Παραλλαγμένος Αυτοκωδικοποιητής (ΠΑ, Variational Autoencoder (VAE)) [18] είναι μια παραλλαγή ενός Αυτοκωδικοποιητή [19], [20]. Είναι ένα γεννητικό μοντέλο, που σημαίνει ότι είναι ικανό να δημιουργήσει νέα δεδομένα, παρόμοια με τα δεδομένα εκπαίδευσης. Η βασική πτυχή του ΠΑ που τον διακρίνει από άλλα γεννητικά μοντέλα έγκειται στην ικανότητά του να διερευνά παραλλαγές των δεδομένων εκπαίδευσης σε μια επιθυμητή, συγκεκριμένη κατεύθυνση (και όχι να δημιουργεί τυχαία νέα δεδομένα). Αυτό μπορεί να επιτευχθεί κυρίως επειδή κάθε είσοδος αντιστοιχίζεται σε μια κατανομή (τις παραμέτρους μιας διανομής βασικά) και όχι σε ένα σταθερό διάνυσμα όπως στον Αυτοκωδικοποιητή. Με αυτόν τον τρόπο, ο λανθάνων χώρος είναι συνεχής, επιτρέποντας σε οποιοδήποτε τυχαίο σημείο να έχει μια 'ουσιαστική' αναπαράσταση μόλις αποκωδικοποιηθεί. Ο σκοπός αυτής της αντιστοίχισης έγκειται στο γεγονός ότι ακόμη και για τα ίδια δεδομένα που δίνονται ως είσοδο, η κωδικοποίηση θα διαφέρει, εξερευνώντας πιθανές παραλλαγές της εισόδου.

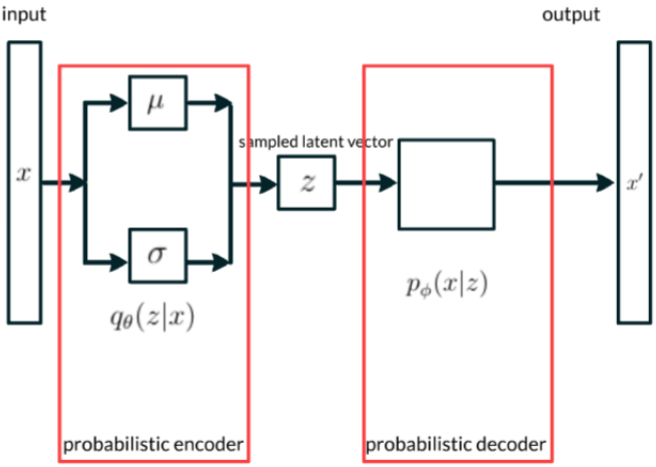
Η αρχιτεκτονική του είναι αυτή που απεικονίζεται στο Σχήμα 1. Από αυτή, μπορούμε να συμπεράνουμε τον τρόπο με τον οποίο γίνεται η εκπαίδευσή του:

Βήμα 1: Μία είσοδος $\boldsymbol{x}$ 'περνάει' μέσω ενός Πιθανοτικού Αποκωδικοποιητή και αντιστοιχίζεται σε μία κατανομή $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ (δηλαδή, στη μέση τιμή και στην τυπική απόκλιση μια Κανονικής Κατανομής) στον λανθάνοντα χώρο $\boldsymbol{z}$, από την οποία το $\boldsymbol{x}$ θα μπορούσε να δημιουργηθεί.

Βήμα 2: Ένα σημείο $\boldsymbol{z}$ δειγματοληπτείται από την κατανομή $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ στον λανθάνοντα χώρο.

Βήμα 3: Το σημείο αυτό $\boldsymbol{z}$ 'περνάει' μέσω ενός Πιθανοτικού Κωδικοποιητή (δηλαδή $p_\theta(\boldsymbol{x}|\boldsymbol{z})$), ο οποίος παράγει μια κατανομή όλων των πιθανών τιμών του $\boldsymbol{x}$.

Βήμα 4: Υπολογίζεται το λάθος της ανακατασκευής μεταξύ εισόδου και εξόδου, και διαδίδεται-προς-τα-πίσω (back-propagation) στο δίκτυο.



Σχήμα 1: *Αρχιτεκτονική Παραλλαγμένου Αυτοκωδικοποιητή*

Ο λόγος που χρησιμοποιείται η προσεγγιστική κατανομή $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ και όχι η πραγματική κατανομή $p_\theta(\boldsymbol{z}|\boldsymbol{x})$ είναι ότι για τον υπολογισμό της $p_\theta(\boldsymbol{z}|\boldsymbol{x})$, από το θεώρημα του Bayes [21], έχουμε ότι: $p(\boldsymbol{z}|\boldsymbol{x}) = \dfrac{p(\boldsymbol{x}|\boldsymbol{z})p(\boldsymbol{z})}{p(\boldsymbol{x})}$, όπου $p(\boldsymbol{x}) = \int_{\boldsymbol{z}} p(\boldsymbol{x}|\boldsymbol{z})p(\boldsymbol{z})d\boldsymbol{z}$. Το τελευταίο αυτό ολοκλήρωμα δεν μπορεί να υπολογιστεί (χρειάζεται εκθετικό χρόνο), καθώς χρειάζεται να είναι γνωστές όλες οι τιμές του $\boldsymbol{z}$.

Για να ελέγξουμε πόσο όμοιες είναι δύο κατανομές, χρησιμοποιούμε την Kullback-Leibler Divergence [22], η οποία συμβολίζεται D_{KL}. Υπάρχει η ευθεία KL απόκλιση, $D_{KL}(p_\theta(\boldsymbol{z}|\boldsymbol{x})||q_\phi(\boldsymbol{z}|\boldsymbol{x}))$ και η αντίστροφη $D_{KL}(q_\phi(\boldsymbol{z}|\boldsymbol{x})||p_\theta(\boldsymbol{z}|\boldsymbol{x}))$. Μιας και η αντίστροφη συνάδει με τη λογική του Παραλλαγμένου Αυτοκωδικοποιητή, χρησιμοποιείται αυτή και προκύπτει ότι:

$$D_{KL}(q_\phi(\boldsymbol{z}|\boldsymbol{x})||p_\theta(\boldsymbol{z}|\boldsymbol{x})) = \log p_\theta(\boldsymbol{x}) + D_{KL}(q_\phi(\boldsymbol{z}|\boldsymbol{x})||p_\theta(\boldsymbol{z})) - \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log p_\theta(\boldsymbol{x}|\boldsymbol{z})\right].$$

Επειδή ισχύει πάντοτε ότι $D_{KL} \geq 0$, προκύπτει ότι:

$$D_{KL}(q_\phi(\boldsymbol{z}|\boldsymbol{x})||p_\theta(\boldsymbol{z}|\boldsymbol{x})) \geq 0 \Rightarrow$$
$$\Rightarrow \log p_\theta(\boldsymbol{x}) \geq -D_{KL}(q_\phi(\boldsymbol{z}|\boldsymbol{x})||p_\theta(\boldsymbol{z})) + \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log p_\theta(\boldsymbol{x}|\boldsymbol{z})\right] = ELBO,$$

όπου:

- $-D_{KL}(q_\phi(\boldsymbol{z}|\boldsymbol{x})||p_\theta(\boldsymbol{z}))$: όρος κανονικοποίησης, καθώς εφαρμόζει ένα είδος περιορισμό στην προσεγγιστική κατανομή. Πρακτικά, 'κανονικοποιεί' τη μορφή του λανθάνοντος χώρου.
- $\mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log p_\theta(\boldsymbol{x}|\boldsymbol{z})\right]$: όρος ανακατασκευής, καθώς μετράει την πιθανότητα των ανακατασκευασμένων δεδομένων. Πρακτικά, συμβάλλει στην εύρεση του καλύτερου σχήματος κωδικοποιητή-αποκωδικοποιητή.

Το δεξί μέρος της παραπάνω εξίσωσης, που ονομάζεται ELBO, αποτελεί ένα φράγμα στην πιθανότητα να δημιουργηθούν πραγματικά δεδομένα, δηλαδή το αριστερό μέρος της εξίσωσης, το οποίο και μας ενδιαφέρει να μεγιστοποιήσουμε. Οπότε μεγιστοποιώντας το ELBO, μεγιστοποιούμε και τη ζητούμενη πιθανότητα. Η παραπάνω πρόταση είναι ισοδύναμη με την ελαχιστοποίηση του αρνητικού ELBO, το οποίο αποτελεί και τη συνάρτηση απωλειών μας:

$$\mathcal{L}(\theta, \phi) = -ELBO = D_{KL}(q_\phi(\boldsymbol{z}|\boldsymbol{x})||p_\theta(\boldsymbol{z})) - \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log p_\theta(\boldsymbol{x}|\boldsymbol{z})\right].$$

Οπότε, ο σκοπός του Παραλλαγμένου Αυτοκωδικοποιητή είναι η εύρεση των καλύτερων κωδικοποιητών και αποκωδικοποιητών (δηλαδή των καλύτερων παραμέτρων θ^*, ϕ^*), που ελαχιστοποιούν την απώλεια $\mathcal{L}$:

$$\theta^*, \phi^* = \arg\min_{\theta,\phi} \mathcal{L}(\theta, \phi).$$

Για την κατανομή του Πιθανοτικού Αποκωδικοποιητή, υποθέτουμε ότι είναι Κανονική με $q_\phi(\boldsymbol{z}|\boldsymbol{x}^{(i)}) = \mathcal{N}(\mu, \sigma^2 I)$. Για την εκ των προτέρων πεποίθησή μας σχετικά με την κατανομή των λανθανόντων μεταβλητών $\boldsymbol{z}$, θεωρούμε ότι ακολουθεί την τυποποιημένη Κανονική κατανομή $p_\theta(\boldsymbol{z}) = \mathcal{N}(0, I)$. Τέλος, υποθέτουμε ότι η κατανομή του Πιθανοτικού Κωδικοποιητή είναι Κανονική. Έτσι, μετά από πράξεις, η προαναφερθείσα συνάρτηση απωλειών γίνεται:

$$\mathcal{L}(\theta, \phi) = -\frac{1}{2}\sum_{j=1}^{J}\left[1 + \log(\sigma_j^2) - \mu_j^2 - \sigma_j^2\right] - \frac{1}{L}\sum_{l=1}^{L}\log p_\theta(\boldsymbol{x}|\boldsymbol{z}^{(l)}),$$

όπου J οι διαστάσεις του $\boldsymbol{z}$, άρα σ_j, μ_j τα $j-$οστά στοιχεία των σ, μ αντίστοιχα.

Προηγουμένως, αναφέραμε ότι γίνεται δειγματοληψία στον λανθάνοντα χώρο. Ωστόσο, η διάδοση-προς-τα-πίσω των σφαλμάτων στο δίκτυο δεν είναι εφικτή, όταν υπάρχει η τυχαιότητα αυτή. Έτσι, προτάθηκε το reparameterization trick [18], σύμφωνα με το οποίο: αντί να πάρουμε δείγμα από την $\mathcal{N}(\mu, \sigma^2 I)$, δηλαδή, $\boldsymbol{z} \sim q_\phi(\boldsymbol{z}|\boldsymbol{x}) = \mathcal{N}(\mu, \sigma^2 I)$, η δειγματοληψία πραγματοποιείται από το $\boldsymbol{z} = \mu + \sigma \odot \boldsymbol{\epsilon}$, με $\odot$ να αποτελεί τον πολλαπλασιαμό κατά στοιχείο και $\boldsymbol{\epsilon} \sim \mathcal{N}(0, I)$.

Μετρήσεις Ζεύξεων

Οι μετρήσεις στις ζέυξεις μπορούν να ληφθούν μέσω του πρωτοκόλλου Simple Network Management Protocol (SNMP) [23]. Το SNMP είναι ένα από τα ευρέως αποδεκτά πρωτόκολλα που διαχειρίζεται και παρακολουθεί συσκευές δικτύου σε Internet Protocol (IP) δίκτυα, όπως δρομολογητές, μόντεμ, διακομιστές και σταθμούς εργασίας.

Δημιουργία Συνθετικών Πινάκων Κίνησης

Οι πίνακες κυκλοφορίας (ΠΚ) έχουν πολλές χρήσεις, εκτός από το απλό γεγονός ότι παρέχουν καλύτερη κατανόηση ενός δικτύου. Μπορούν να χρησιμοποιηθούν για ανίχνευση ανωμαλιών, διαχείριση και σχεδιασμό δικτύου, εξισορρόπηση φορτίου και σχεδιασμό πρωτοκόλλου δικτύου. Δυστυχώς, όχι μόνο είναι υπολογιστικά δύσκολο να μετρηθούν οι ροές κυκλοφορίας άμεσα, αλλά και ο αριθμός των διαθέσιμων στο κοινό δεδομένων ΠΚ είναι περιορισμένος. Αυτή η ανεπαρκής ποσότητα συνόλων δεδομένων αποτελεί σοβαρό εμπόδιο για τους χειριστές δικτύου ή τους επιστήμονες, οι οποίοι ενδιαφέρονται να δοκιμάσουν και να αξιολογήσουν την αποτελεσματικότητα των προτεινόμενων αλγορίθμων σε πραγματικές καταστάσεις.

Για να αντιμετωπισθεί αυτό το πρόβλημα, η τεχνητή κατασκευή πινάκων κυκλοφορίας είναι μια εξαιρετική λύση. Σημειώνεται εδώ ότι η παραγωγή πινάκων κυκλοφορίας πρέπει να εστιάζει στη διατήρηση των χαρακτηριστικά τους. Εκτός από την προαναφερθείσα χρήση, η σύνθεση πινάκων κυκλοφορίας είναι ένα απαραίτητο ενδιάμεσο στάδιο στην εκτίμηση του πίνακα κυκλοφορίας.

Στην τρέχουσα εργασία, αξιοποιούμε έναν Παραλλαγμένο Αυτοκωδικοποιητή. Ο στόχος είναι η παροχή ΠΚ που έχουν παρατηρηθεί προηγουμένως σε αυτό το βαθύ μοντέλο δημιουργίας, προκειμένου να κατασκευασθούν νέοι ΠΚ που έχουν παρόμοια χαρακτηριστικά με τους προαναφερθέντους.

Όταν δημιουργούμε ΠΚ, εάν δοκιμάσουμε ένα διάνυσμα από την ίδια προηγούμενη κατανομή $p_\theta(\boldsymbol{z}) = \mathcal{N}(0, I)$ των κωδικοποιημένων διανυσμάτων, ο αποκωδικοποιητής θα δημιουργήσει ένα νέο σημείο. Αυτό το σημείο είναι εγγυημένο ότι έχει χαρακτηριστικά παρόμοια με τα παρατηρούμενα σημεία, επειδή κατά τη διάρκεια της εκπαίδευσης, έχουμε διαβεβαιώσει ότι ο λανθάνων χώρος είναι συνεχής, επιτρέποντας σε οποιοδήποτε τυχαίο σημείο να έχει ουσιαστική αναπαράσταση μόλις αποκωδικοποιηθεί.

Εκτίμηση Πίνακα Κυκλοφορίας

Η Εκτίμηση Πίνακα Κυκλοφορίας περιλαμβάνει το συμπερασμό της κίνησης που μεταδίδεται μεταξύ κάθε ζεύγους κόμβων σε ένα δίκτυο, από μετρήσεις δεδομένων σε όλες τις ζεύξεις. Μπορεί να διατυπωθεί ως εξής:

Έστω n οι κόμβοι και m οι ζεύξεις ενός δικτύου. Η σχέση μεταξύ του πίνακα κυκλοφορίας, του πίνακα δρομολόγησης και των μετρήσεων στις ζεύξεις μπορεί να περιγραφεί από ένα σύστημα γραμμικών εξισώσεων:

$$\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x},$$

όπου:

- $\boldsymbol{y}$ είναι το $m \times 1$ διάνυσμα των μετρήσεων στις ζεύξεις. Υποθέτουμε ότι η διαδρομή δεν αλλάζει σε όλη τη διάρκεια της περιόδου μέτρησης.
- $\boldsymbol{A}$ είναι ο $m \times n^2$ πίνακας δρομολόγησης, που περιγράφει την τοπολογία. Το στόχείο του $a_{i,j}$ είναι ίσο με 1, αν το ζεύγος προέλευσης-προορισμού j διασχίζει τη ζεύξη i, ή 0 διαφορετικά.
- $\boldsymbol{x}$ είναι ο $n \times n$ πίνακας κυκλοφορίας, του οποίου το στοιχείο στη γραμμή i και τη στήλη j αντιπροσωπεύει την κυκλοφορία (κίνηση) μεταξύ του κόμβου προέλευσης i και του κόμβου προορισμού j. Συνήθως, απεικονίζεται ως ένα διάνισμα $n^2 \times 1$.

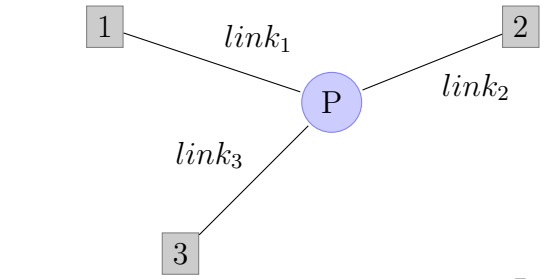
Στο πλαίσιο αυτό, το πρόβλημα είναι να εκτιμηθεί ο πίνακας κυκλοφορίας $\boldsymbol{x}$, δεδομένων των μετρήσεων στις ζέυξεις $\boldsymbol{y}$ και του πίνακα δρομολόγησης $\boldsymbol{A}$. Κάτι τέτοιο δεν είναι τετριμμένο, καθώς ο αριθμός των ζευγών προέλευσης-προορισμού ισούνται με n^2 (άγνωστες ποσότητες) και είναι σχεδόν πάντα πολύ μεγαλύτερος από τον αριθμό των ζεύξεων m (γνωστές ποσότητες). Επομένως, το προαναφερθέν σύστημα γραμμικών εξισώσεων είναι σε μεγάλο βαθμό υπο-προσδιορισμένο, ή κακώς-διατυπωμένο (δηλαδή, ο πίνακας $\boldsymbol{A}$ δεν είναι πλήρους τάξης (full rank) και υπάρχουν πολλές λύσεις που ταιριάζουν στις παρατηρήσεις). Η φύση αυτή του προβλήματος ΕΠΚ αντιμετωπίζεται συνήθως με τη χρήση στατιστικών μοντέλων και την εφαρμογή πρόσθετων υποθέσεων.

Ένα απλό παράδειγμα ενός δικτύου με την κυκλοφορία του φαίνεται στο Σχήμα 2. Είναι προφανές ότι υπάρχουν πολλές περισσότερες ροές ΠΠ (OD), σε σχέση με τις μετρήσεις συνδέσμων (l).

Πιο συγκεκριμένα, εκμεταλλευόμαστε τον εκπαιδευμένο αποκωδικοποιητή που μπορεί να συνθέσει τεχνητά παραδείγματα από τον λανθάνοντα χώρο $\boldsymbol{z}$ και υποθέτουμε ότι η λύση που αναζητούμε (ποιος είναι ο ΠΚ $\boldsymbol{x}$) μπορεί να δημιουργηθεί από αυτόν τον αποκωδικοποιητή. Ως εκ τούτου, μετατρέπουμε το πρόβλημα της ΕΠΚ σε ένα πρόβλημα ελαχιστοποίησης στον λανθάνοντα χώρο:

$$\arg\min_{\boldsymbol{z}} \left[\left\| \boldsymbol{y} - \boldsymbol{A} d(\boldsymbol{z}) \right\|_2^2 \right],$$

όπου $d(.)$ είναι το εκπαιδευμένο γεννητικό μοντέλο (δηλαδή, ο αποκωδικοποιητής). Έτσι, ο στόχος είναι να βρεθεί η καλύτερη παράμετρος $\boldsymbol{z}$, προκειμένου να ελαχιστοποιηθεί η διαφορά μεταξύ των παρατηρούμενων μετρήσεων ζεύξεων $\boldsymbol{y}$ και των εκτιμώμενων μετρήσεων ζεύξεων $\boldsymbol{A}d(\boldsymbol{z})$, τα οποία σχηματίζονται 'περνώντας' το $\boldsymbol{z}$ από τον αποκωδικοποιητή. Αφού βρούμε τα καλύτερα $\boldsymbol{z}$, έχουμε αποκτήσει την εκτίμηση



$$\begin{bmatrix} l_1 \\ l_2 \\ l_3 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} OD_{1,1} \\ OD_{1,2} \\ OD_{1,3} \\ OD_{2,1} \\ OD_{2,2} \\ OD_{2,3} \\ OD_{3,1} \\ OD_{3,2} \\ OD_{3,3} \end{bmatrix}$$

Σχήμα 2: *Ένα απλό δίκτυο με την κυκλοφορία του. Το R αντιπροσωπεύει ένα δρομολογητή και τα $1, 2, 3$ είναι οι κόμβοι. Στο κάτω μέρος απεικονίζεται εξίσωση της ΕΠΚ που έχει αντικατασταθεί.*

του καλύτερου πίνακα κυκλοφορίας $\boldsymbol{x} \simeq d(\boldsymbol{z})$. Ο αποκωδικοποιητής είναι διαφοροποιήσιμος, επομένως το πρόβλημα επιλύεται με την ανάπτυξη ενός βελτιστοποιητή και πιο συγκεκριμένα του Adam.

Μια δεύτερη μέθοδος που προτείνεται για την ΕΠΚ περιλαμβάνει την προσθήκη ενός όρου κανονικοποίησης στην προαναφερθείσα συνάρτηση:

$$\arg\min_{\boldsymbol{z}} \left[\left\| \boldsymbol{y} - \boldsymbol{A}d(\boldsymbol{z}) \right\|_2^2 + c \left\| \boldsymbol{z} \right\|_2^2 \right],$$

όπου το c είναι ένα μέτρο αντιστάθμισης μεταξύ του σφάλματος μέτρησης και της σημασίας της από-πριν πεποίθησή μας για την κατανομή του $\boldsymbol{z}$, ωθώντας την εξέταση των περιοχών που επιλέγονται από τον αποκωδικοποιητή. Ο βελτιστοποιητής Adam χρησιμοποιείται επίσης σε αυτήν τη μέθοδο.

Και στις δύο προαναφερθείσες εξισώσεις, πρέπει να καθοριστεί ένα σημείο εκκίνησης για το $\boldsymbol{z}$. Το αρχικό διάνυσμα $\boldsymbol{z_0}$ θα μπορούσε να επιλεγεί τυχαία, αλλά αυτό μπορεί να καθυστερήσει τη διαδικασία, καθώς θα χρειαστούν περισσότερα βήματα για να προσεγγιστεί το ελάχιστο. Οι συνολικές επαναλήψεις θα μπορούσαν να μειωθούν, σε περίπτωση που επιλέξουμε ένα 'καλό' αρχικό σημείο. Έτσι, διερευνούμε K τυχαία λανθάνοντα διανύσματα και επιλέγουμε αυτό με την ελάχιστη απόσταση από τις μετρήσεις ζεύξεων:

$$\boldsymbol{z_0} = \boldsymbol{z_k} : \ \left\| \boldsymbol{y} - \boldsymbol{A}d(\boldsymbol{z_k}) \right\|_2^2 \leq \left\| \boldsymbol{y} - \boldsymbol{A}d(\boldsymbol{z_i}) \right\|_2^2, \ \text{για } i \in 1, ..., K,$$

ή:

$$\boldsymbol{z_0 = z_k}: \ \left\|\boldsymbol{y} - \boldsymbol{A}d(\boldsymbol{z_k})\right\|_2^2 + c\left\|\boldsymbol{z_k}\right\|_2^2 \leq \left\|\boldsymbol{y} - \boldsymbol{A}d(\boldsymbol{z_i})\right\|_2^2 + c\left\|\boldsymbol{z_i}\right\|_2^2, \ \text{για } i \in 1, ..., K.$$

Σταδιακή Βελτιστοποίηση

Οι προαναφερθείσες μέθοδοι αποτελούνται από δύο βήματα

Βήμα 1: Εκπαίδευση Παραλλαγμένου Αυτοκωδικοποιητή με προηγούμενους Πίνακες Κυκλοφορίας.

Βήμα 2: Μετατροπή της Εκτίμησης Πίνακα Κυκλοφορία σε ένα πρόβλημα ελαχιστοποίησης στον λανθάνοντα χώρο και επίλυσή του με τον βελτιστοποιητή Adam.

Ωστόσο, συχνά οι προαναφερθείσες συναρτήσεις για ελαχιστοποίηση έχουν πολλά τοπικά ελάχιστα. Έτσι, οι αλγόριθμοι βελτιστοποίησης μπορεί να 'εγκλωβιστούν' σε κάποιο τοπικό ελάχιστο και να μην συγκλίνουν στο ολικό ελάχιστο. Αυτές οι μη κυρτές συναρτήσεις οδήγησαν σε μια εναλλακτική προσέγγιση στην ΕΠΚ. Η παραπάνω διαδικασία δύο βημάτων ενώνεται σε μία: βελτιστοποιούμε σταδιακά την ακολουθία των συναρτήσεων ελαχιστοποίησης που έχουν κατασκευαστεί με την ακολουθία των δικτύων αποκωδικοποιητή που λαμβάνονται κατά τη διάρκεια διαφορετικών σταδίων της εκπαίδευσης του Παραλλαγμένου Αυτοκωδικοποιητή, υιοθετώντας τις αντίστοιχες τιμές παραμέτρων. Αυτή η 'ταυτόχρονη' βελτιστοποίηση έχει χρησιμοποιηθεί στη βιβλιογραφία [24] για προβλήματα με πολλά τοπικά ελάχιστα και έχει αποδειχθεί ότι συγκλίνει στο ολικό ελάχιστο.

Έστω $\phi_0, \phi_1, ..., \phi_T$ οι παράμετροι των αποκωδικοποιητών $d_0, d_1, ..., d_T$ που λαμβάνονται ανά κάποιο συγκεκριμένο αριθμό εποχών εκπαίδευσης. Για τον αποκωδικοποιητή d_i, οι προαναφερθείσες εξισώσεις ελαχιστοποίησης γίνονται:

$$\arg\min_{\boldsymbol{z}} \left[\left\|\boldsymbol{y} - \boldsymbol{A}d_i(\boldsymbol{z})\right\|_2^2\right],$$
$$\arg\min_{\boldsymbol{z}} \left[\left\|\boldsymbol{y} - \boldsymbol{A}d_i(\boldsymbol{z})\right\|_2^2 + c\left\|\boldsymbol{z}\right\|_2^2\right].$$

Από αυτές τις εξισώσεις, βρίσκουμε το καλύτερο λανθάνων διάνυσμα $\boldsymbol{z_i^*}$ χρησιμοποιώντας τον Adam. Αυτό το 'τρέχον' βέλτιστο θα χρησιμοποιηθεί για την αρχικοποίηση του Adam κατά τη βελτιστοποίηση του επόμενου στόχου. Έτσι, υποθέτοντας τον αποκωδικοποιητής d_{i+1} μετά από έναν προκαθορισμένο αριθμό εποχών εκπαίδευσης, θα χρησιμοποιήσουμε το $\boldsymbol{z_i^*}$ ως σημείο εκκίνησης για τον Adam στη νέα συνάρτηση ελαχιστοποίησης:

$$\arg\min_{\boldsymbol{z}} \left[\left\|\boldsymbol{y} - \boldsymbol{A}d_{i+1}(\boldsymbol{z})\right\|_2^2\right],$$
$$\arg\min_{\boldsymbol{z}} \left[\left\|\boldsymbol{y} - \boldsymbol{A}d_{i+1}(\boldsymbol{z})\right\|_2^2 + c\left\|\boldsymbol{z}\right\|_2^2\right].$$

Από αυτό, θα λάβουμε το νέο βέλτιστο $\boldsymbol{z}^*_{\boldsymbol{i+1}}$, κοκ. Τέλος, αν d_T είναι ο τελικός αποκωδικοποιητής, η εκτίμηση για τον πίνακα κυκλοφορίας θα είναι: $\hat{\boldsymbol{x}} = d_T(\boldsymbol{z}^*_{\boldsymbol{T}})$.

Δεδομένα: Δίκτυο Abilene

Για την αξιολόγηση των προαναφερθέντων μεθόδων, χρησιμοποιήθηκαν δεδομένα κίνησης που συλλέχθηκαν από το δίκτυο Abilene [17]. Τα δεδομένα κίνησης προκύπτουν κυρίως από μεγάλα πανεπιστήμια στις ΗΠΑ και έχουν χρησιμοποιηθεί σε διάφορες μελέτες [1], [25]–[27].

Όπως φαίνεται από το Σχήμα 3, αυτό το δίκτυο κορμού βρίσκεται στη Βόρεια Αμερική και αποτελείται από 12 κύριους κόμβους (η Ατλάντα αποτελείται από 2 κόμβους), οι οποίοι διανέμουν την κίνησή τους μέσω $12 \cdot 12 = 144$ ροών Προέλευσης-Προορισμού. Η τοπολογία έχει επίσης $15 \cdot 2 = 30$ κύριες ζεύξεις που συνδέουν αυτές τις περιοχές μεταξύ τους. Κάθε κόμβος έχει επιπλέον 2 εξωτερικούς συνδέσμους (1 για την είσοδο από εξωτερικό κόμβο και 1 για την έξοδο προς κάποιον εξωτερικό κόμβο), με αποτέλεσμα $30 + 12 \cdot 2 = 54$ συνολικές ζεύξεις. Η χωρητικότητα όλων των εσωτερικών συνδέσεων είναι 9920000 kbps, εκτός από την ζεύξη Ατλάντα-Ιντιανάπολη και την Ιντιανάπολη-Ατλάντα που έχουν χωρητικότητα 2480000 kbps. Τα δεδομένα κίνησης περιέχουν μέσους όρους για διαστήματα 5 λεπτών, για 24 εβδομάδες, από την 1η Μαρτίου έως τις 10 Σεπτεμβρίου 2004 (υπάρχουν περιόδους που λείπουν). Έτσι, υπάρχουν $(60/5) \cdot 24 = 288$ δείγματα την ημέρα και $288 \cdot 7 = 2016$ δείγματα την εβδομάδα. Ένα παράδειγμα ενός Πίνακα Κίνησης φαίνεται στον Πίνακας 1.

Σχήμα 3: *Τοπολογία του δικτύου Abilene [1].*

Υλοποίηση

Η αρχιτεκτονική του Κωδικοποιητή απεικονίζεται στον Πίνακα 2, ενώ αυτή του Αποκωδικοποιητή στον Πίνακα 3. Είναι σημαντικό να αναφέρουμε ότι οι πρώτες 13 βδομάδες (δηλαδή, τους πρώτους $13 \cdot 2016 = 26208$ ΠΚ) χρησιμοποιούνται για την εκπαίδευση και η 14^{η} εβδομάδα (δηλαδή, οι επόμενοι 2016 ΠΚ) για έλεγχο.

Πίνακας 1: *Ο 5-λεπτών πίνακας κυκλοφορίας (πραγματικές τιμές) από το δίκτυο Abilene, την 1η Μαρτίου 2014, μεταξύ 00:00 και 00:05, σε kbps.*

Προέλευση \ Προορισμός	1	2	3	4	5	6	7	8	9	10	11	12
1	26.667	522.208	1641.339	335.728	413.032	489.875	365.077	817.869	452.061	747.405	388.317	3141.640
2	445.149	82660.749	16283.117	5169.035	3931.403	27351.896	4844.035	18231.909	12803.955	1421.888	8957.483	51748.245
3	3793.693	12735.325	28744.432	13738.253	9830.336	26359.776	6537.749	27775.901	14098.339	1816.941	4495.256	10647.053
4	230.805	2341.483	38518.189	3761.989	8408.763	7207.741	3948.899	21375.568	5723.408	14385.464	10609.464	12248.861
5	239.043	2956.779	16891.501	3693.019	5606.299	9143.904	7130.168	98457.928	7265.264	2947.376	2237.597	8506.651
6	4766.925	9254.733	122044.576	21378.197	33173.784	22849.056	9890.840	24405.043	40616.099	3466.387	13892.187	41138.093
7	420.960	4443.563	26972.272	5394.304	5476.104	8017.757	4022.899	8673.584	12842.411	1223.752	2444.272	12048.437
8	339.661	19394.589	89723.683	9039.381	9030.867	42720.251	13570.251	11369.131	61164.419	2311.541	25519.453	55726.387
9	3897.640	40887.840	53674.288	16345.053	23987.787	83325.448	24767.283	71022.560	136796.045	9591.352	21934.557	111860.741
10	26.667	1041.205	5046.067	10406.912	1436.683	3861.611	2097.072	2000.211	2211.461	14269.608	3297.648	1987.707
11	111.019	15881.547	22512.587	4341.176	11302.768	7691.184	2260.848	17766.373	24845.373	4755.005	1038.357	10517.379
12	11219.101	125937.728	66541.197	36063.421	15439.312	62781.813	32642.733	91675.627	133661.405	1980.576	29760.203	187653.483

Πίνακας 2: *Δομή του Κωδικοποιητή στον Παραλλαγμένο Αυτοκωδικοποιητή.*

Τύπος επιπέδου	**Πυρήνας**	**Βήμα**	**Συμπλήρωση**	**Έξοδος**
Input ($\boldsymbol{x}$)	-	-	-	(12, 12, 1)
Dropout	-	-	-	(12, 12, 1)
Conv2D	(3, 3)	(2, 2)	SAME	(6, 6, 32)
Conv2D	(3, 3)	(2, 2)	SAME	(3, 3, 64)
Conv2D	(3, 3)	(1, 1)	SAME	(3, 3, 128)
Flatten	-	-	-	(1152)
Dense	-	-	-	(64)
Dense (μ)	-	-	-	(10)
Dense (σ)	-	-	-	(10)
Sampling ($\boldsymbol{z}$)	-	-	-	(10)

Πίνακας 3: *Δομή του Αποκωδικοποιητή στον Παραλλαγμένο Αυτοκωδικοποιητή.*

Τύπος επιπέδου	**Πυρήνας**	**Βήμα**	**Συμπλήρωση**	**Έξοδος**
Input ($\boldsymbol{z}$)	-	-	-	(10)
Dense	-	-	-	(64)
Dense	-	-	-	(576)
Reshape	-	-	-	(3, 3, 64)
Conv2DTranspose	(3, 3)	(1, 1)	SAME	(3, 3, 128)
Conv2DTranspose	(3, 3)	(2, 2)	SAME	(6, 6, 64)
Conv2DTranspose	(3, 3)	(2, 2)	SAME	(12, 12, 32)
Conv2DTranspose	(3, 3)	(1, 1)	SAME	(12, 12, 1)

Αποτελέσματα

Η αξιολόγηση των προτεινόμενων μεθόδων γίνεται με τη βοήθεια των παρακάτω μετρικών: Root Mean Square Error (RMSE), Normalized Mean Absolute Error (NMAE), Spatial Relative Error (SRE) που εκφράζει το σχετικό σφάλμα εκτίμησης κάθε μεμονωμένης ροής ΠΠ σε όλη τη διάρκεια ζωής της και το Temporal Relative Error (TRE) που συνοψίζει το σχετικό σφάλμα εκτίμησης όλων των ροών ΠΠ σε ένα δεδομένο χρονικό σημείο. Τα σφάλματα αυτά υπολογίζονται ως εξής:

$$\mathrm{RMSE}(t) = \frac{\left\|\hat{\boldsymbol{x}}_t - \boldsymbol{x}_t\right\|_2}{\sqrt{N}} = \sqrt{\frac{\sum_{i=1}^{N}\left(\hat{\boldsymbol{x}}_t(i) - \boldsymbol{x}_t(i)\right)^2}{N}},$$

$$\mathrm{NMAE}(t) = \frac{\left\|\hat{\boldsymbol{x}}_t - \boldsymbol{x}_t\right\|_1}{\left\|\boldsymbol{x}_t\right\|_1} = \frac{\sum_{i=1}^{N}\left|\hat{\boldsymbol{x}}_t(i) - \boldsymbol{x}_t(i)\right|}{\sum_{i=1}^{N}\left|\boldsymbol{x}_t(i)\right|},$$

$$\mathrm{TRE}(t) = \frac{\left\|\hat{\boldsymbol{x}}_t - \boldsymbol{x}_t\right\|_2}{\left\|\boldsymbol{x}_t\right\|_2} = \frac{\sqrt{\sum_{i=1}^{N}\left(\hat{\boldsymbol{x}}_t(i) - \boldsymbol{x}_t(i)\right)^2}}{\sqrt{\sum_{i=1}^{N}\left(\boldsymbol{x}_t(i)\right)^2}},$$

$$\mathrm{SRE}(i) = \frac{\left\|\hat{\boldsymbol{x}}_{1:T}(i) - \boldsymbol{x}_{1:T}(i)\right\|_2}{\left\|\boldsymbol{x}_{1:T}(i)\right\|_2} = \frac{\sqrt{\sum_{t=1}^{T}\left(\hat{\boldsymbol{x}}_t(i) - \boldsymbol{x}_t(i)\right)^2}}{\sqrt{\sum_{t=1}^{T}\left(\boldsymbol{x}_t(i)\right)^2}},$$

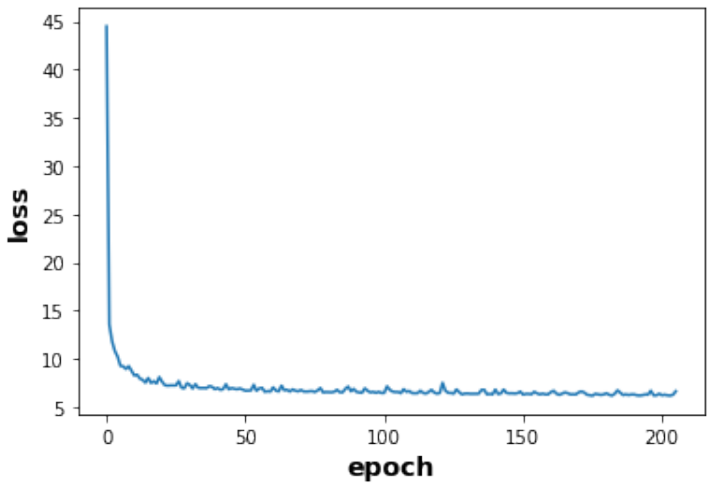
όπου:

- $\boldsymbol{x}$ είναι ο πραγματικός ΠΚ.

- $\hat{\boldsymbol{x}}$ είναι ο ΠΚ που εκτιμάμε .
- $i = 1, 2, ..., N$ αντιπροσωπεύει την κάθε ροή ΠΠ.
- $t = 1, 2, ..., T$ δηλώνει το κάθε χρονικό σημείο (δηλαδή, ΠΚ).
- $\|.\|_1$ και $\|.\|_2$ είναι η πρώτη και η δεύτερη νόρμα αντίστοιχα.

Επιλέγοντας ένα 'καλό' αρχικό λανθάνων διάνυσμα, μπορούμε να μειώσουμε τον αριθμό των απαιτούμενων επαναλήψεων βελτιστοποίησης. Σημειώνεται ότι όλα τα αποτελέσματα που αναφέρονται στη συνέχεια λαμβάνονται σύμφωνα με βάση αυτήν την παραδοχή.

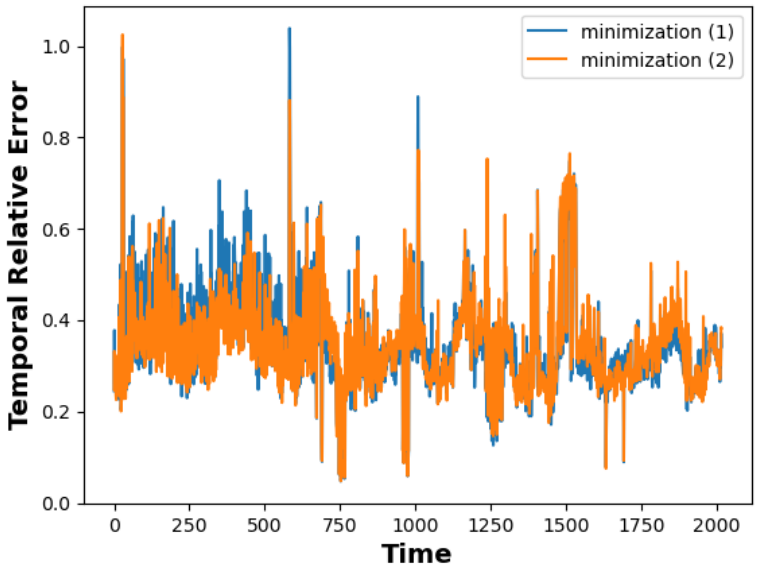
Το Σχήμα 4 απεικονίζει τη συνολική απώλεια του Εκπαιδευμένου Αυτοκωδικοποιητή κατά τη εκπαίδευση. Από αυτό το σχήμα, είναι προφανές ότι η μέγιστη τιμή της συνολικής απώλειας είναι στις πρώτες εποχές. Ωστόσο, μετά από περίπου 1 έως 3 εποχές, η συνολική απώλεια μειώνεται σημαντικά και στη συνέχεια, καθώς περνούν οι εποχές, υπάρχει μια ελαφρά μείωση της συνολικής απώλειας. Ορίσαμε επίσης ένα callback EarlyStopping, το οποίο είναι υπεύθυνο για τον τερματισμό της εκπαίδευσης όταν η απώλεια φτάσει στο ελάχιστο, ή ισοδύναμα, η απώλεια σταματήσει να μειώνεται.



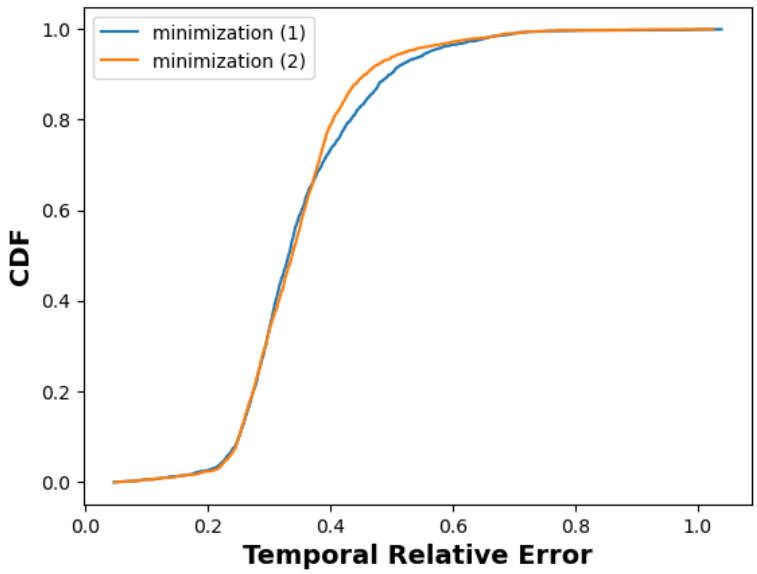
Σχήμα 4: *Απώλεια μοντέλου στα δεδομένα εκπαίδευσης.*

Το Σχήμα 5 απεικονίζει τα χρονικά σχετικά σφάλματα. Ο χρόνος t του άξονα $\boldsymbol{x}$ αντιστοιχεί στους 2016 ΠΚ, οπότε $T = 2016$. Τα χρονικά σφάλματα για την ελαχιστοποίηση (2) είναι μικρότερα από την ελαχιστοποίηση (1) σχεδόν για κάθε ΠΚ. Αυτο το συμπέρασμα επιβεβαιώνεται από το Σχήμα 6, όπου απεικονίζει την αθροιστική συνάρτηση κατανομής των χρονικών σχετικών σφαλμάτων.

Το Σχήμα 7 απεικονίζει τα χωρικά σχετικά σφάλματα κάθε ζεύγους ΠΠ, άρα $N = 144$ (υπάρχουν συνολικά 144 ζευγάρια ΠΠ). Γενικά, τα χωρικά σφάλματα λαμβάνουν μικρές τιμές, εκτός από μια ροή μεταξύ 107 και 109, η οποία εκτοξεύεται στα

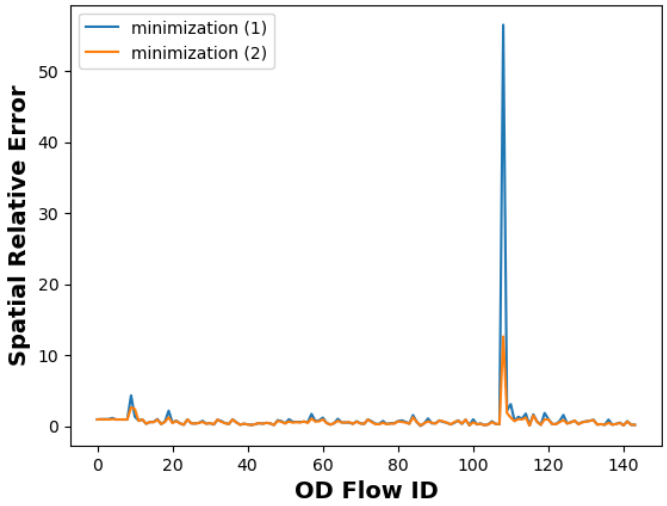


Σχήμα 5: *Χρονικά σχετικά σφάλματα (TRE).*

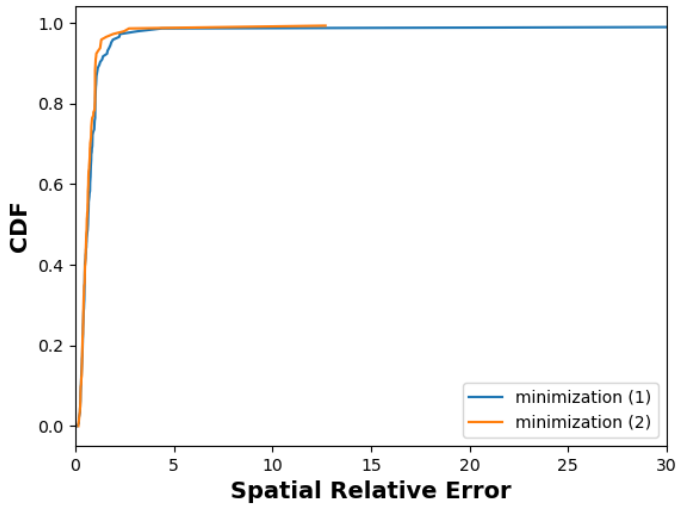


Σχήμα 6: *Αθροιστική συνάρτηση κατανομής (CDF) των χρονικών σχετικών σφαλμάτων.*

ύψη. Αυτή η συμπεριφορά θα μπορούσε να οφείλεται, είτε σε σφάλμα σε αυτό το συγκεκριμένο ζεύγος ΠΠ, είτε σε απότομη αύξηση της ροής κίνησης αυτού του συγκεκριμένου ζεύγους. Ωστόσο, το SRE στην ελαχιστοποίηση (2) είναι περίπου το ένα πέμπτο του αντίστοιχου σφάλματος στην ελαχιστοποίησης (1). Από το Σχήμα 8, είναι προφανές ότι η αθροιστική συνάρτηση κατανομής των ελαχιστοποίησεων (1) και (2) είναι πολύ κοντά μεταξύ τους.



Σχήμα 7: *Χωρικά σχετικά σφάλματα (SRE).*



Σχήμα 8: *Αθροιστική συνάρτηση κατανομής (CDF) των χωρικών σχετικών σφαλμάτων.*

Ο Πίνακας 4 συνοψίζει τα αποτελέσματα για τις τρεις εξεταζόμενες παραλλαγές για ΕΠΚ. Όπως φαίνεται, η ελαχιστοποίηση (2) με $c = 0.1$ βελτιώνει πράγματι όλες τις μετρικές (ιδιαίτερα το SRE).

Πίνακας 4: *Εκτίμηση Απωλειών.*

Εκπαίδευση και Ελαχιστοποίηση (1)				
Σφάλμα	**Μέση Τιμή**	**Διάμεσος**	**Τυπική Απόκλιση**	**Μέγιστο**
RMSE (Mbps)	13.9045	12.6126	5.2089	59.6762
NMAE	0.3544	0.3415	0.0824	0.8176
TRE	0.3557	0.3320	0.1098	1.0387
SRE	1.1233	0.6323	4.6677	56.5583
Εκπαίδευση και Ελαχιστοποίηση (2)				
Σφάλμα	**Μέση Τιμή**	**Διάμεσος**	**Τυπική Απόκλιση**	**Μέγιστο**
RMSE (Mbps)	13.6893	12.6701	5.0450	60.6747
NMAE	0.3466	0.3350	0.0748	0.9496
TRE	0.3488	0.3381	0.0989	1.0249
SRE	0.7330	0.5769	1.0698	12.6865
Ταυτόχρονη Εκπαίδευση και Ελαχιστοποίηση (2)				
Σφάλμα	**Μέση Τιμή**	**Διάμεσος**	**Τυπική Απόκλιση**	**Μέγιστο**
RMSE (Mbps)	13.3673	12.4220	5.5080	43.2572
NMAE	0.4262	0.3972	0.1473	1.3309
TRE	0.4033	0.3791	0.1505	1.3824
SRE	1.7464	0.8027	6.9723	81.7783

Συμπεράσματα

Στην παρούσα διπλωματική εργασία, εξετάσαμε αρχικά τις σημαντικές συνεισφορές στη βιβλιογραφία σχετικά με τον τομέα της Τομογραφίας Δικτύου και της Εκτίμησης Πίνακα Κυκλοφορίας (ΕΠΚ). Έπειτα, αναλύσαμε το πρόβλημα της ΕΠΚ εκμεταλλευόμενοι

τα βαθιά γεννητικά μοντέλα και πιο συγκεκριμένα, έναν Παραλλαγμένο Αυτοκωδικοποιητή, αντί να βασιζόμαστε στην αραιότητα (sparsity) [7] όπως συνηθίζεται. Αξιοποιήσαμε ένα διαθέσιμο σύνολο δεδομένων με μετρήσεις από το δίκτυο Abilene για να εκπαιδεύσουμε το μοντέλο. Χρησιμοποιώντας τον αποκωδικοποιητή, μετατρέψαμε το ΕΠΚ σε ένα πρόβλημα ελαχιστοποίησης στον λανθάνοντα χώρο. Ο αποκωδικοποιητής χρησιμοποιήθηκε επίσης για τη δημιουργία νέων Πινάκων Κίνησης (ΠΚ), που έχουν παρόμοια χαρακτηριστικά με τα δείγματα στη διαδικασία εκπαίδευσης. Έτσι, μπορεί εύκολα να γίνει αντιληπτό ότι η επιλογή του κατάλληλου συνόλου εκπαίδευσης είναι ζωτικής σημασίας, προκειμένου να περιέχει όλα τις δυνατές τιμές που μπορεί να πάρει μια ροή Προέλευσης-Προορισμού (ΠΠ). Επιπλέον, διερευνήσαμε την εναλλακτική προσέγγιση της σταδιακής βελτιστοποίησης και αξιολογήθηκε η απόδοση των προτεινόμενων μεθόδων.

Μελλοντικές Προεκτάσεις

Σε αντίθεση με τα κλασικά νευρικά δίκτυα που αποτελούνται από μεγάλο αριθμό παραμέτρων και απαιτούν μεγάλα σύνολα δεδομένων για να εκπαιδευτούν, πρόσφατα, προτάθηκε μια καινοτόμος προσέγγιση που περιόρισε τις προαναφερόμενες προϋποθέσεις. Αυτά τα μοντέλα ονομάζονται Untrained Generative Models και βασίζονται σε βαθιά νευρωνικά δίκτυα, με πολύ λίγες παραμέτρουςκαι απλή αρχιτεκτονική που δεν χρειάζονται εκπαίδευση. Τα μοντέλα έχουν αποδειχθεί ότι επιτυγχάνουν θετικά αποτελέσματα [28], [29].

Μια άλλη κατεύθυνση που αξίζει να εξερευνθεί είναι η χρήση Conditional Generative Models [30], [31]. Αυτά τα γεννητικά μοντέλα χρησιμοποιούνται για τη δειγματοληψία από κάποια άγνωστη υψηλών διαστάσεων κατανομή, η οποία επιτρέπει τη δημιουργία πολλαπλών λύσεων από τις ίδιες μετρήσεις.

Η μελλοντική έρευνα σε αυτόν τον τομέα θα μπορούσε να περιλαμβάνει τη χρήση Deep Reinforcement Learning [32]. Για να εκτιμηθεί με ακρίβεια ο ΠΚ, ένας πράκτορας πρέπει να εκπαιδευτεί για να συμπεριφέρεται βέλτιστα σε ένα περιβάλλον, λαμβάνοντας αποφάσεις και αντίστοιχες ανταμοιβές.

Chapter 1

Introduction

From its early days as an experimental small packet-switching network, Internet has come a long way in a short period of time. In contrast to the telephone network which evolved in a slower manner, Internet dominated rapidly [2] due to its innovative practicality and decentralized control, and, nowadays, it is the key means of communication. In 2021, more than 4.7 billions of humans utilize the Internet every day [3]. With the advent of new applications such as video streaming, and the rapid growth of data traffic from mobile devices, we are witnessing a worldwide information blast.

Given the Internet's expanding significance, knowledge of its traffic has become increasingly important. However, because of the lack of centralized control, quantitative assessment of network performance becomes very difficult. Not only is it administratively and computationally cumbersome for each station to collect, process and transmit this information, but also such knowledge might be considered private and non-shareable by the providers. This data is pivotal for network operators, who want to schedule vital tasks, such as routing policies and network management and planning [4]. The latter are all of high significance towards meeting the goal of serving the demand with satisfactory quality in terms of delay, packet loss rate and/or other Quality of Service (QoS) / Quality of Experience (QoE) parameters.

Network Tomography [5]–[7] was developed to overcome the aforementioned obstacles. Its purpose is to infer important statistics of a network, without needing full cooperation of intermediate nodes. At the same time, recent advances in deep neural networks [8], combined with progress in stochastic optimization methods, have enabled the extraction of meaningful and more intricate features from the data studied. By exploiting and integrating the intelligence of these models in a suitable environment, it is possible to confront many sub-problems of Network Tomography, including Traffic Matrix Estimation, which is the core subject of this thesis, and infer not directly observed characteristics of a network.

1.1 Motivation

The Internet is comprised of heterogeneous devices, such as mobile phones, routers and computers, with different protocols and operating systems. In this chaotic environment, one of the most challenging tasks is to delineate the vast array of telecommunication and computing infrastructures and delve into their meaningful attributes, such as connectivity, bandwidth and delays. Maps [9] are therefore one of the most useful resources for deciphering the large and complicated infrastructure of information and communication technologies, enabling us to fathom where such technologies are located, how they interconnect, how much information is transmitted or lost, etc.

Ping [10] was one of the first attempts towards the understanding of Internet's characteristics. It operates by sending ICMP packets to a specific destination and is basically utilized for checking the connectivity of a device to the network and measuring the delay between two computers. In 1987, a more detailed diagnostic tool was generated, called traceroute [11]. Traceroute is meant to identify the path between a source and a destination point and the time for this round-trip.

Nevertheless, it would be naive to rely solely on these software utilities to provide a purposeful cartogram of the web, because these techniques require the cooperation of intermediate servers and routers. More specifically, in the path between two pair of nodes, it is possible that some points block or treat with very low priority the traffic used for diagnostic purposes (due to security and performance reasons mainly), rendering the whole procedure unsuccessful. Subsequently, the maps [9] that are currently available contain only a partial illustration of the cyberspace.

Moreover, even if such depictions were obtainable, each provider has the right to decide whether to share operational information concerning, for example, delays and traffic distribution of their network, and has to contemplate possible drawbacks of this action. First and foremost, transmitting these statistics causes a significant increase in the traffic flow. If the network is not constructed to support this amount of data, the system might discard useful packets to avoid congestion and even crash. In addition, it is probable that malicious users analyze these details and exploit vulnerabilities of a system.

Hence, scientists [33] were in the quest for another approach, which would not need the collaboration of intermediate nodes and would not overload the network's links. The field that emerged was named Network Tomography by Vardi [5], due to the similarity between network inference and medical tomography, and refers to the inverse [12] problem of estimating the values of the parameters characterizing the system under investigation, given the results of actual observations. Similar problems have been addressed thoroughly in signal processing [13], computer vision [14], medical imaging [15] and other applications, and some useful insights were adopted to shed light on the relatively newborn subject of Network Tomography.

1.2 Objectives

The target of this diploma thesis is the Traffic Matrix Estimation (TME) problem, which is a form of Network Tomography. The Traffic Matrix (TM) quantifies the demand between all possible pairs of origin and destination entities (usually nodes or set of nodes) in a network. It is of paramount importance for network operators, who want to know how traffic, described by TM, flows between all possible pairs of Origin and Destination (OD) nodes of the network, as it is important for a number of tasks [4], such as anomaly detection, network management and planning, future prediction of traffic trends, load balancing and design of routing policy. This issue falls in the realm of statistical inverse problems, which, as mentioned above, has a very extensive literature.

There is a plethora of approaches for constructing a TM, ranging from direct measurements, to hypothesizing special correlations and more recently to incorporating learning techniques. This diploma thesis aims to analyze previous methods and propose a novel one, together with some variations, which are based on Variational Autoencoders (VAEs) [18]; a prominent Generative Model [34]. The suggested models, apart from estimating traffic matrices, can be utilized to generate new data, that resemble the observed one in terms of statistical properties. The effectiveness of our contributions are evaluated in a real context, with data from an accessible backbone network.

1.3 Contributions

The foremost contributions of this diploma thesis are the following [16]:

- Literature overview, concerning the latest contributions in TMs and TME.
- Synthesize traffic matrices that conform to the observed constraints of a particular network, by training a Variational Autoencoder.
- First, train a VAE and, then, solve an optimization problem in the latent space to address the Traffic Matrix Estimation problem.
- Transform the two-stage TME method (first VAE training and, then, optimization in the latent space) into a type of "concurrent" optimization.
- Evaluate the performance of the proposed methods, using a publicly available dataset of real traffic matrices recorded in the Abilene [17] backbone network.

1.4 Thesis Outline

The current diploma thesis is organized as follows. In Chapter §1, a brief introduction to the problem is provided, together with the mainsprings of the field, some fundamental definitions and the goals & contributions of this thesis. In Chapter §2, the

field of Network Tomography is addressed and the basic notions and sub-problems are discussed. Chapter §3 summarizes concepts of Representation Learning and focuses on Variational Autoencoders. Chapter §4 is the main part. It starts with the prior work and describes the key contributions of this thesis on the subject of Traffic Matrix Estimation & Synthesis. The proposed methods are also evaluated on a real dataset and results are presented. This diploma thesis is concluded in Chapter §5, where suggestions for future work are also made. A List of Figures and a List of Tables are also included, just before the References.

Chapter 2

Network Tomography

2.1 Basics

Network tomography (NT) [5]–[7] is a network-monitoring technique that refers to the inference of network performance parameters based on traffic measurements at a limited subset of the network's nodes. To introduce smoothly the topic, some crucial notions are mentioned below.

2.1.1 Structure

Definition 2.1.1 (Graph)**.** A directed/undirected *graph* [35] is an ordered pair $G(V, E)$, composed of:

- V, a set of nodes (or points);
- $E \subseteq \{\{x, y\} \mid x, y \in V \text{ and } x \neq y\}$, a set of links (or edges), which are ordered /unordered pairs of nodes.

Definition 2.1.2 (Path)**.** A *path* [35] in a graph is a finite or infinite sequence of distinct links which joins a sequence of distinct nodes.

Generally, the following are assumed to be satisfied in network tomography:

Assumption 1 (Topology)**.** The topology of a network can be perceived as a directed or undirected graph.

Assumption 2 (Node)**.** Each node may represent an end-system, a router, or a subnetwork.

Assumption 3 (Link)**.** Each link can constitute a chain of physical links connected by intermediate routers.

Assumption 4 (Traffic)**.** Traffic [36], which illustrates the data moving across a network, encapsulated mostly in network packets, originates from a source node and is delivered to a destination node (or multiple destinations). It traverses a set of links between some set of nodes and may be split across several paths by load balancing [37], or may keep to a single path.

Assumption 5 (Spatial Independence)**.** All additive metrics (i.e., metrics whose value over a path is the sum of the values at each node), such as delay and loss rate experienced by packets are independent along different links.

Assumption 6 (Stationarity)**.** Topology and paths are fixed during the observation period.

Figure 2.1 depicts an example of a topology of a network (undirected graph), which is made up of 7 nodes, that are connected with 8 bidirectional links. A path from node D to node C is also visible with the red color.

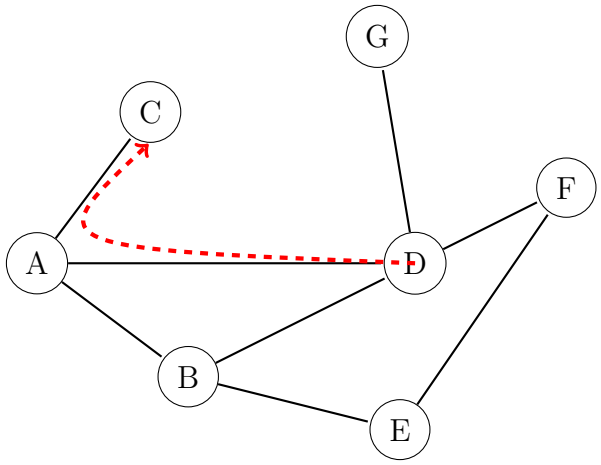


Figure 2.1: *Example of a network topology, with 7 nodes and 8 bidirectional links. Red dashed arrow shows the path from node D (origin) to node C (destination), following links DA and AC.*

2.1.2 Identifiability

Theorem 1 (Identifiability)**.** *Let* $\mathcal{P} = \{P_\theta : \theta \in \Theta\}$ *be a statistical model where the parameter space* Θ *is either finite or infinite dimensional.* $\mathcal{P}$ *is identifiable [38] if the mapping* $\theta \mapsto P_\theta$ *is one-to-one:*

$$P_{\theta_1} = P_{\theta_2} \quad \Rightarrow \quad \theta_1 = \theta_2 \quad \forall \theta_1, \theta_2 \in \Theta.$$

Corollary 1.1. *Let* $\mathcal{P} = \{P_\theta : \theta \in \Theta\}$ *be an identifiable statistical model. Then, if* $\theta_1 \neq \theta_2$*, then* $P_{\theta_1} \neq P_{\theta_2}$*.*

2.1.3 Model

The aim of NT is to use aggregate values that can be easily obtained through measurements at accessible nodes/links, in order to infer characteristics that are difficult to quantify directly, due to extreme communication overhead [39]. In contrast to the aforementioned tools (see section 1.1), the role of intermediate routers and servers that required complete orchestration is now reduced and simultaneously, the network's measurement traffic is diminished significantly.

Many network tomography problems can be approximated by the linear model:

$$\boldsymbol{y}_t = \boldsymbol{A}\boldsymbol{x}_t + \boldsymbol{\epsilon}, \tag{1}$$

where t denotes time so as to reflect more accurately the dynamical nature of the true networks; $\boldsymbol{y}_t$ is a vector of measurements (i.e., packet counts or delays) taken at time t at various nodes; $\boldsymbol{A}$ is the routing matrix, which is mainly represented as a binary matrix (could be also a probability matrix in case of multiple paths) that captures the topology of the network; $\boldsymbol{x}_t$ is a vector of network parameters (i.e., mean delays, logarithms of loss probabilities or traffic flow counts) at time t; and $\boldsymbol{\epsilon}$ is a noise term.

Network Tomography refers to the inverse problem of estimating the unobserved network parameters $\boldsymbol{x}_t$ given $\boldsymbol{y}_t$ and either a set of assumptions regarding the statistical distribution of the noise $\boldsymbol{\epsilon}$ or the introduction of some form of regularization to induce identifiability.

In order for accurate inference of a model's parameters to be feasible, the Identifiability property must hold. Mathematically, as can be deduced from corollary 1.1, this is equivalent to saying that different values of the parameters must generate different probability distributions of the observable variables. If this is the case, from the true probability distribution P_θ (which is found from an infinite number of observations), true values of the parameters that generated the aforementioned distribution can be inferred by inverting the map $\theta \mapsto P_\theta$. In this way, it is guaranteed that with an infinite number of observations, the model's underlying parameters can be detected.

The core pitfall lies in the potentially very large dimension of $\boldsymbol{A}$. From the perspective of linear algebra, the components of $\boldsymbol{x}_t$ are uniquely identifiable if and only if the number of linear independent measurement paths equals the number of $\boldsymbol{x}_t$-components. The challenging task concerning $\boldsymbol{A}$ is that usually, it is an ill-posed matrix (i.e., not full-rank) and, hence, non-invertible. Thus, the number of variables to be estimated is much larger than the number of equations, resulting in the non-uniqueness of solutions.

2.2 Subclasses

Network Tomography can be categorized Based on Measurements-Parameters, Based on Measurement Methodology or Based on Delivery Schemes.

2.2.1 Based on Measurements-Parameters

Network Tomography is divided into three subclasses, depending on the type of measurements and the parameters of interest [6]:

Link-Level Network Tomography

Link-Level Tomography [40]–[50] is the inference of link-level network parameters (i.e., $\boldsymbol{x}_t$) from path-level measurements (i.e., $\boldsymbol{y}_t$). On the one hand, link-level parameters include chiefly link loss rates, link delay distributions and link bandwidths, which are addressed in loss tomography [40], [45], [46], delay tomography [41], [47] and bandwidth tomography [49] respectively. On the other hand, path-level measurements consist principally of accounts of delivered/lost packets and time delays. Link-level statistics are important not only to characterize the performance of a network, but also to identify and avoid congestions, or other deviant behaviors in a network.

Generally, if all nodes in a network cooperate and exchange freely information, the aforementioned link-level parameters could be estimated from direct measurements. Ping and traceroute introduced in section 1.1 are designed for this purpose. However, it is a frequent phenomenon that many intermediate nodes do not respond to the packets sent by these diagnostic tools in order not to spend time processing requests that are subordinate to the process of communication and with the fear of malicious attacks, rendering the whole procedure unsuccessful.

As far as path-level measurements (i.e., $\boldsymbol{y}_t$) are concerned, they can be calculated through a coordinated measurement scheme between the sender and the receiver. The sender records whether a packet reached its destination or was dropped/lost and determines the transmission delay by some form of acknowledgment from the receiver to the sender upon successful packet reception. Definitely, it is not possible for the sender to determine where the packet was lost or measure delays/bandwidths on any link in the path.

As far as link-level metrics (i.e., $\boldsymbol{x}_t$) are concerned, they are typically additive. This means that the path metric obtained by combining multiple serial links is the sum of the individual link metrics. Delays are a good example of an additive metric, while loss rates, which are a multiplicative metric, can be expressed in an additive form by using the logarithmic function.

The end-to-end path measurements are carried out between pairs of nodes with monitoring capabilities. That is why link-level inference was mainly addressed in tree-structured networks. A simple example can be viewed in Figure 2.2. Undoubtedly, the decision of which nodes will be chosen as monitors becomes a key problem.

As aforementioned, most of the existing solutions assume that routes from a single source form a tree. However, with the rapid deployment of Software Defined Networking (SDN) [51] and Network Function Virtualization (NFV) [52], the routing

paths in modern networks are becoming more complex. To address this problem, scientists proposed lately some novel approaches for general routing paths in general topologies [42], [48].

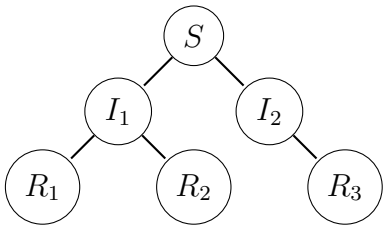


Figure 2.2: *Simple tree-structured network, consisting of a single sender (S), two internal nodes (I_1, I_2) and three receivers (R_1, R_2, R_3).*

Path-Level Network Tomography

Path-Level Network Tomography [5], [6] (or Origin-Destination Network Tomography) is the antithesis of Link-Level Network Tomography: the goal is to estimate path-level network parameters (i.e., $\boldsymbol{x}_t$) from measurements made on individual links (i.e., $\boldsymbol{y}_t$).

In this category, one of the most known problems is *Traffic Matrix Estimation*. Its goal is to estimate how much traffic originated from a specified node (origin) and was destined for a specified receiver (destination). The combination of the traffic intensities of all these origin-destination pairs forms the *origin-destination (OD) traffic matrix*, or simply, *traffic matrix (TM)*. This is the original Network Tomography problem studied in [5]. The OD traffic matrix is useful for a number of tasks, such as anomaly detection, prediction of traffic trends, load balancing and design of routing policy.

For this problem, the linear Equation 1 is used without the error term $\boldsymbol{\epsilon}$, because the stochasticity that is induced by noise is already captured in $\boldsymbol{x}_t$:

$$\boldsymbol{y}_t = \boldsymbol{A}\boldsymbol{x}_t. \tag{2}$$

The problem of Traffic Matrix Estimation is analyzed extensively in chapter 4.

Topology Inference

In Topology Inference [42], [50], [53]–[57], the network topology expressed by the routing matrix $\boldsymbol{A}$ is not known. The goal is the inference of the topology $\boldsymbol{A}$ based on end-to-end measurements obtained without the cooperation of the internal nodes.

Most diagnostic tools designed for discovering the map of a network, such as traceroute, require complete cooperation of all intermediate routers and servers in a path. Due to the decentralized nature of Internet and the conflicting interests of its providers, the aforementioned requirements are rarely met. Because only end-to-end

measurements are used, it is possible to identify the logical topology of the network. In other words, internal nodes where no branching of traffic occurs do not appear in the logical topology. In case internal nodes cooperated and provided additional information, the real, or physical topology, could have been estimated. A simple example between physical and logical topology can be viewed in Figure 2.3.

Most of the existing topology inference methods require tree structure for the network. The central idea is to observe metrics at pairs of receivers that behave as a monotonically increasing function of the number of shared links or common queues between the two receiving nodes, such as covariance. The more common links in the paths towards two receivers, the greater this metric will be. Knowledge of a pairwise similarity metric values under an additive metric, such as counts of losses and delay differences, is sufficient to completely identify the logical topology by employing various statistical techniques such as hierarchical clustering [53], maximum likelihood [54], and Bayesian inference [55].

Recently, scientists work with topology inference, where routing paths do not strictly form a tree [42].

(a) *Physical topology* **(b)** *Logical topology*

Figure 2.3: *Physical (a) and logical (b) topology of the simple-structured network of Figure 2.2. There is no branching in node I_2, so it is not visible in logical topology.*

2.2.2 Based on Measurement Methodology

Network Tomography is classified in two subclasses, depending on the way measurements are acquired [5]–[7]:

Active Tomography

In Active Tomography, probe packets are sent from nodes located on the periphery to the network, in order to recover concealed information about the internal links. Ping and traceroute (see section 1.1) are two classic examples of this method.

Since test traffic mimics the service traffic, active testing is ideal for providing a real-time view of the performance with regard to link-level parameters such as delay, or packet loss. It also provides a punctual response in case of an abnormal behavior in the network and identifies quickly its root cause. Despite their utility in the field, injecting traffic solely for diagnostic purpose is not treated benevolently by

many service providers. More and more firewalls are designed to ignore these type of packages and protect the leakage of their operational information (topology, loss or delay in each link, etc), creating an uncooperative and suspicious environment, where trust is difficult to be established. This has prompted investigations into more passive monitoring techniques.

Passive Tomography

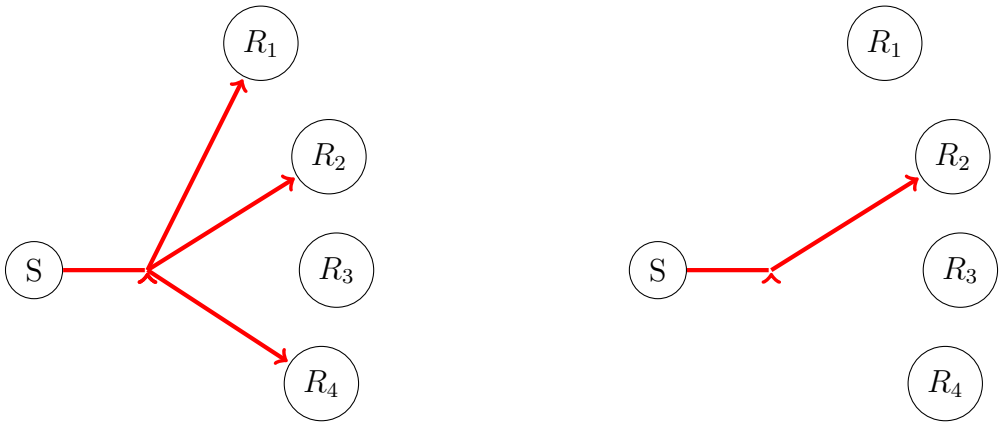
In Passive Tomography, packets already existing in the network traffic are observed, so as to extract useful hidden characteristics of internal links.

At its simplest, passive monitoring may be nothing more than the periodic collecting of port statistics, like bytes and packets transmitted. More typically, it involves capturing some, or all, of the traffic flowing through a port for detailed, non-real-time, analysis of things like bandwidth usage and detection of anomalous traffic. Since only regular data flow is analyzed, no extra network resources are consumed. However, because of exclusively observing existing communications, there is a negative impact in terms of flexibility.

The main problem of interest in Passive Tomography is Traffic Matrix Estimation, which is also the core subject of this diploma thesis.

2.2.3 Based on Delivery Schemes

Network Tomography, based on delivery schemes, can be split in two common modes of communication in networks (see also Figure 2.4 for a graphical representation):



(a) *Multicast communication* **(b)** *Unicast communication*

Figure 2.4: *In multicast communication (a), a sender (S) is addressed to a group of receivers (R_1, R_2, R_4), while in unicast communication (b), a sender (S) is addressed to a single receiver (R_2).*

Multicast

In multicast communication [40]–[44], [50], each packet is sent to a group of receivers simultaneously. It can be one-to-many or many-to-many transmission.

Network Tomography that relies on multicast communication to infer missing link-level characteristics was one of the first attempts in this field. With this technique, packets are sent repeatedly from a sender to a group of destinations. From the receivers' responses, it is possible to know whether a packet was delivered successfully (acknowledgment is sent back to the sender in case of correct transmission) and infer the rate of loss on the existing link, or measure the delay for the trip. Nonetheless, these methods require simple tree topologies.

In larger and more general trees, the task is more sophisticated. That is why advanced algorithms have been developed for multicast-based tomography on arbitrary tree-structured networks [43]. Recently, scientists experiment with link-level tomography, where routing paths may not follow tree structure [42].

It is true however, that many networks do not support multicast transmission, due to the significant increase in traffic which might lead to congestion and even network failure. That is why unicast-based tomography is of considerable practical interest.

Unicast

In unicast communication [45]–[48], each packet is sent to one and only one receiver. It is an one-to-one transmission.

Unicast measurements are more difficult to work with than multicast, but since many networks do not support multicast transmission, unicast-based tomography needs to be analyzed. The core difficulty is that although single unicast packet measurements allow the estimation of path loss rates and delay distributions, there is not a unique mapping of these path-level parameters to the corresponding individual link-by-link parameters (routing matrix $\boldsymbol{A}$ is not full-rank). If a packet was lost or delayed, it is not profound in which link exactly that loss or delay occurred.

There is an extensive literature [45]–[47] that strives for tackling these issues and the main focus is on the use of back-to-back packet pairs or sequences of packets (to "imitate" multicast probing). Recently, there is an effort to infer internal network performances using unicast probes in general topologies [48]. Unicast measurement can be conducted either actively or passively (see subsection 2.2.2).

Chapter 3

Representation Learning

3.1 Introduction

Representation (or feature) learning [58] is the process of automatically learning representations of input data (by transforming or extracting features from it) in order to make tasks, such as classification, easier to perform. Interest in this field stems from the fact that problems in machine learning [59] depend on the data representation on which they are applied. In a plethora of problems, like speech recognition [60] or natural language processing [61], it is necessary to analyze the complex input and extract meaningful information about it, which will encourage effective processing. The difficulty of representation learning lies in the absence of a particular objective. What are the ways to determine a "good" representation and distinguish it from another "worse" representation? And moreover, how can this be modeled into specific training criteria?

Supervised-Unsupervised Learning

Representation learning can be either *supervised*, or *unsupervised* [62].

In *supervised feature learning*, the training data are labeled; meaning that learning is based on already known input-output pairs, that are used in training. After that, the algorithm measures its accuracy through a loss function and adjusts its parameters until the error has been sufficiently minimized.

In *unsupervised feature learning*, learning is accomplished with no hint at all (i.e., unlabeled data) about the correct outputs. Often in representation learning, features extracted from an unlabeled dataset are leveraged in supervised tasks to ameliorate their performance (*semi-supervised learning*).

Back-propagation & Gradient Descent

In most of these techniques, it is crucial to monitor the error between input and output every time a prediction is made. More specifically, in neural networks [63], the central mechanism by which these models learn is called *back-propagation*. In

general, neural networks consist of nodes interconnected to each other with links, that have some weight. These weights capture the knowledge of the system for a problem, but frequently, they are poorly calibrated.

Back-propagation is used to fine-tune the weights of a neural net based on the error obtained in the previous epoch (i.e., iteration). Proper tuning of the weights leads to reduced errors, making the model reliable by increasing its generalization. Basically, this method calculates efficiently the gradient of a loss function with respects to all the weights in the network. These gradients can be utilized by an optimization algorithm, such as gradient descent, or stochastic gradient descent [64], to minimize the loss of the model with regard to a training dataset.

Deep Architectures

Deep learning [8] is one of the many ways to learn features; though, its contribution is of paramount importance. Deep architectures, as depicted in Figure 3.1, in contrast to "swallow" ones, refer to multiple levels of representation or learning a hierarchy of features.

According to [58], the dominant idea is "*to learn a hierarchy of features one level at a time, using unsupervised feature learning to learn a new transformation at each level to be composed with the previously learned transformations; essentially, each iteration of unsupervised feature learning adds one layer of weights to a deep neural network. Finally, the set of layers could be combined to initialize a deep supervised predictor, such as a neural network classifier, or a deep generative model*".

The biggest advantage of multiple intermediate layers is the ability to process vast number of features and extract useful information from compound and large datasets. Deep architectures can lead to more abstract representations, which are invariant to small, local changes of the input. This abstraction can be, either constructed such as in the convolutional neural network [65], or inferred from the intermediate layers.

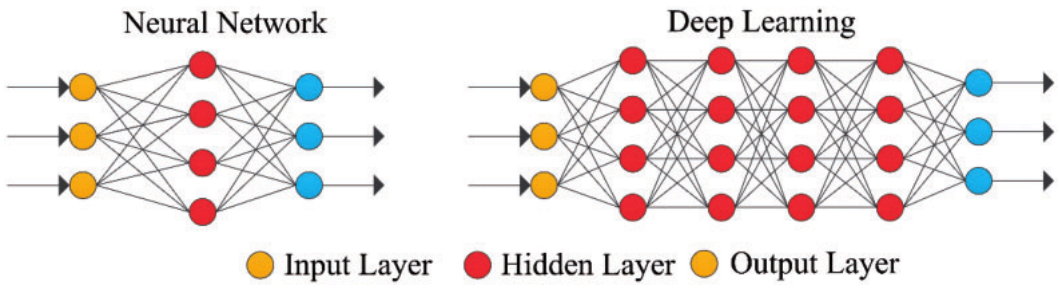


Figure 3.1: *Swallow or traditional architecture (left) vs. deep architecture (right). Image source [66]*

Generative Models

In machine learning, *generative modeling* [67] is an unsupervised task that involves automatically discovering and learning regularities or patterns in input data so that the model can be used to produce new examples that could derive from the original dataset. Practically, they try to model how the input data is placed in the space, rather than just drawing boundaries (*discriminative models*). Most known examples include Generative Adversarial Network (GAN) [68] and Variational Autoencoder (VAE) [18].

3.2 Dimensionality Reduction - Latent Space

Dimensionality reduction [69] is the transformation of data from a high-dimensional space into a meaningful representation of low-dimensional space. This low-dimensional space is called *latent space*.

In order to deal with real-world, high dimensional, raw, sparse data, it is crucial to reduce their dimensions into more dense representations. This low-dimensional representation should, ideally, match the data's intrinsic dimensionality. Data's intrinsic dimensionality refers to the smallest number of parameters needed to account for the data's observed properties.

Dimensionality Reduction can be performed, using either *linear*, or *non-linear* techniques:

- *Linear techniques*, such as *Principal Components Analysis (PCA)* [70] allow to handle simple, linear data.

- *Non-linear techniques*, like *Autoencoders* [19], [20], are capable of learning a non-linear mapping between the high-dimensional and low-dimensional data representation and thus managing more intricate data.

In addition, Dimensionality Reduction is accomplished, with *feature selection* or *feature extraction*:

- *Feature selection* provides opportunity for discarding some useless features, so as to reduce the dimensionality of the data. In this way, only useful features, which form a lower-dimensional space, are preserved, where analysis will be performed more precisely.

- *Feature extraction* enables the creation of a reduced number of features in a lower-dimensional space, based on the features of high-dimensional space. This transformation might be *linear*, or *non-linear*, taking into consideration the complexity of the data.

3.3 Autoencoder

3.3.1 Description & Properties

An *autoencoder* [19], [20] is a feedforward (i.e., information always moves one direction; it never goes backwards forming loops), non-recurrent neural network designed to copy its input to its output. Internally, it has a hidden layer (*bottleneck layer*) that describes a *code* used to represent the input. It is an unsupervised learning model (no labeled inputs required) and consists of two networks (see Figure 3.2):

- An *encoder*, which takes in an input (in high-dimensional space), and converts it into a denser, compressed, latent representation (in low-dimensional/latent space), called *code*, *latent variables*, or *latent representation*. It basically performs *dimensionality reduction*.
- A *decoder*, which is used for the exact reverse process: it takes the low-dimensional code of bottleneck layer and converts (or decompresses) it back to the original input (in high-dimensional space).

The aforementioned reverse procedure cannot always be without loss. Over-reducing the dimensions of latent space during the encoding might cause information loss, which will never be recovered in the decoding process.

Mathematically, an autoencoder can be defined as:

$$\begin{aligned} g_\phi &: \mathcal{X} \to \mathcal{Z}, \\ f_\theta &: \mathcal{Z} \to \mathcal{X}, \\ \phi^*, \theta^* &= \arg\min_{\phi,\theta} \mathcal{L}(\boldsymbol{x}, f_\theta(g_\phi(\boldsymbol{x}))). \end{aligned} \tag{3}$$

The encoder g_ϕ (encoder function $g(.)$, parameterized by ϕ) takes an input $\boldsymbol{x} \in \mathbb{R}^p = \mathcal{X}$ and maps it to $\boldsymbol{z} \in \mathbb{R}^q = \mathcal{Z}$, with $p > q$. In this way, $\boldsymbol{z} = g_\phi(\boldsymbol{x})$. The decoder f_θ (decoder function $d(.)$, parameterized by θ) takes $\boldsymbol{z}$ and maps it to $\boldsymbol{x}' \in \mathbb{R}^p$. In this way, $\boldsymbol{x}' = f_\theta(\boldsymbol{z}) = f_\theta(g_\phi(\boldsymbol{x}))$.

The goal is to find the best encoding-decoding scheme (ϕ^*, θ^*), such that output $\boldsymbol{x}'$ is as close as possible to $\boldsymbol{x}$, using an iterative optimisation process. This scheme keeps the maximum of information when encoding and, so, has the minimum of reconstruction error when decoding. So, at each iteration, the autoencoder is fed with data (i.e., $\boldsymbol{x}$), the output (i.e., $\boldsymbol{x}'$) is compared with the initial data and the error/loss is backpropagated through the architecture to update the weights of the networks. There are various metrics to quantify the reconstruction error/loss (i.e., $\mathcal{L}(.)$) and penalize the network for creating output different from the input. *Mean Squared Error (MSE)* or *Cross-Entropy* between the output and the input are most commonly used.

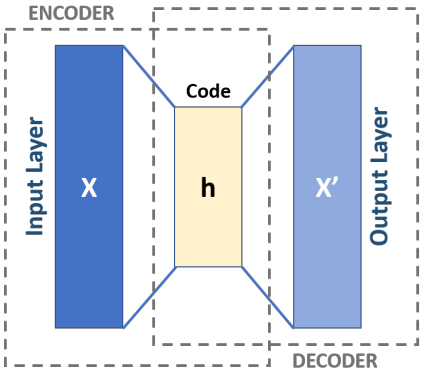


Figure 3.2: *Architecture of Autoencoder. Image source: Wikipedia*

3.3.2 Limitations

In general, autoencoders have been used in literature to confront mainly dimensionality reduction and information retrieval tasks. The characteristics of these models are convenient, for instance, for image denoising [71] or anomaly detection [72].

Apart from that, their usage is limited. The core drawback of autoencoders is that the encoder learns to map each input into a fixed-vector, leading to a latent space with discontinuities. If a point is selected randomly from that latent space and passed through the decoder, the result will probably be an arbitrary, noisy representation (unless, of course, the random point accidentally coincides with an already known transformation).

An example of a latent space can be viewed in Figure 3.3, where four animals are distinguished. It is obvious that if a random point (red point in the figure) is opted from this low-dimensional space and fed to the decoder, the result cannot be defined, simply because the model does not contain any knowledge for this particular point. The red point will probably contain some characteristics of blue and yellow representations (it lies between these two clusters), yet the output will be meaningless and neither close to yellow or blue animal.

Unquestionably, due to the small, undefined region of latent space explored during training with the aforementioned architecture, it is not reasonable to expect this space to be well-organized. In case of a regularized space, it would be possible to explore variations of existing inputs and thus, create novel data. A model designed for this purpose is called *Variational Autoencoder (VAE)* and is described in the following section.

3.4 Variational Autoencoder

Variational Autoencoder (VAE) [18] is a variation of an Autoencoder (see §3.3). It is a generative model, which means that it is capable of generating a new output, similar to the training data. The key aspect of VAE that distinguishes it from other

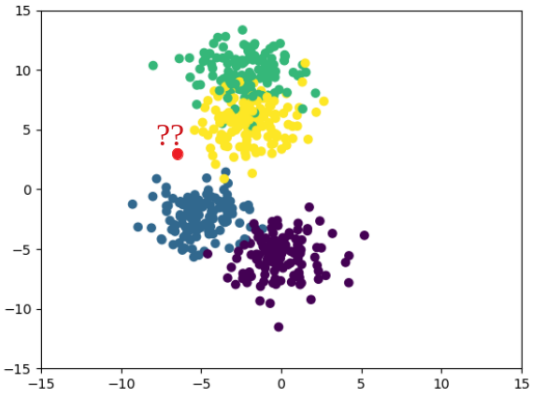


Figure 3.3: *Exemplary latent space of an autoencoder, which discriminates between four different clusters. Red dot is a random point.*

generative models lies in its ability to explore variations on training data in a desired, specific direction; not randomly creating new data. This can be achieved primarily because each input is mapped into a distribution (the parameters of a distribution basically), and not into a fixed vector as in Autoencoder. In this way, the latent space is continuous, allowing any random point to have a meaningful representation once decoded.

3.4.1 Notation

Table 3.1: *Notation used in Variational Autoencoder.*

Symbol	Explanation
N	Number of samples
θ	Generative model parameters
ϕ	Recognition (variational) model parameters
$\boldsymbol{x}$	Observed variable (data, or evidence)
$\boldsymbol{z}$	Latent (i.e., unobserved) variable or *code*
$\boldsymbol{x}^{(i)}$	i-th sample (data point) of $\boldsymbol{x}$
$\boldsymbol{z}^{(i)}$	i-th sample (data point) of $\boldsymbol{z}$
$\mathcal{X}$	Dataset $\mathcal{X} = \{\boldsymbol{x}^{(i)}\}, 1 \leq i \leq N$
$p_\theta(\boldsymbol{x})$	Data distribution, parametrized by θ
$p_\theta(\boldsymbol{z})$	Prior distribution, parametrized by θ
$p_\theta(\boldsymbol{x}\|\boldsymbol{z})$	Likelihood distribution, parametrized by θ / Probabilistic decoder
$p_\theta(\boldsymbol{z}\|\boldsymbol{x})$	True Posterior distribution, parametrized by θ
$q_\phi(\boldsymbol{z}\|\boldsymbol{x})$	Approximate Posterior distribution, parametrized by ϕ / Probabilistic encoder

3.4.2 Bayesian Statistics

In *Bayesian statistics* [21], probability expresses a degree of belief in an occurrence. The core of this statistical field is *Bayes' theorem*, which is utilized for computing and updating one's belief about a hypothesis (e.g., parameter), after observing new data (i.e., evidence). The aforementioned theorem is used in a Bayesian statistical model called *Bayesian Inference*. There, it can be leveraged to estimate the parameters of a probability distribution or statistical model by assigning probabilities to model parameters and updating them after evidence is obtained:

Theorem 2 (Bayes' theorem)**.** *Given a hypothesis (or parameter)* $\boldsymbol{z}$ *and a new data* $\boldsymbol{x}$*, the conditional probability of* $\boldsymbol{z}$ *given* $\boldsymbol{x}$*, called posterior probability, is expressed as follows:*

$$p(\boldsymbol{z}|\boldsymbol{x}) = \frac{p(\boldsymbol{x}|\boldsymbol{z})p(\boldsymbol{z})}{p(\boldsymbol{x})},$$

where:

- $p(\boldsymbol{x})$ is the probability of the new data $\boldsymbol{x}$, which, for continuous $\boldsymbol{z}$, is calculated using the law of total probability [73] as:

$$p(\boldsymbol{x}) = \int_{\boldsymbol{z}} p(\boldsymbol{x}|\boldsymbol{z})p(\boldsymbol{z})d\boldsymbol{z}. \tag{4}$$

- $p(\boldsymbol{z})$ is the probability of the hypothesis (or parameter) $\boldsymbol{z}$, called *prior probability*, because it expresses one's belief or knowledge about $\boldsymbol{z}$ prior to seeing any data.

- $p(\boldsymbol{x}|\boldsymbol{z})$ is the *likelihood*, which expresses the probability of the data $\boldsymbol{x}$, given hypothesis (or parameter) $\boldsymbol{z}$.

3.4.3 Problem Outline

As mentioned above, VAE was proposed to tackle the problem of irregularity of latent space (see §3.3.2). This issue is responsible for the inability of simple Autoencoder to explore this space and generate new data. To establish this "arrangement", VAE encodes inputs as distributions instead of fixed points, but the procedure entails deliberate work. In general, the purpose of this mapping is that even for the same observed data, the *code* will differ, exploring possible variations of the input.

First and foremost, it is essential to introduce some assumptions, that will hold for the rest of the book. It is presumed that distribution is defined as p_θ, where θ describes its parameters and that $p_\theta(\boldsymbol{z})$, $p_\theta(\boldsymbol{x}|\boldsymbol{z})$ and $p_\theta(\boldsymbol{z}|\boldsymbol{x})$ characterize the prior, likelihood and posterior distribution respectively. Moreover, $\boldsymbol{x}$ is the observed variable and $\boldsymbol{z}$ is the latent one.

Supposing the true parameter θ^* of the distribution and the values $\boldsymbol{z}^{(i)}$ of the latent variable $\boldsymbol{z}$ are known, generation of new data $\boldsymbol{x}^{(i)}$ is conducted as follows:

Step 1: Take a sample $\boldsymbol{z}^{(i)}$ from prior distribution $p_{\theta^*}(\boldsymbol{z})$.

Step 2: Generate $\boldsymbol{x}^{(i)}$ from the likelihood $p_{\theta^*}(\boldsymbol{x}|\boldsymbol{z} = \boldsymbol{z}^{(i)})$.

Ideally, taking into consideration the aforementioned process, encoder and decoder can be redefined. The novel *probabilistic encoder* is determined by $p_\theta(\boldsymbol{z}|\boldsymbol{x})$, which describes the distribution of the observed variable given the latent one, while the *probabilistic decoder*, $p_\theta(\boldsymbol{x}|\boldsymbol{z})$, represents the distribution of the latent variable given the observed one. In this way, the problem of discontinuous latent space of Autoencoders starts to vanish, due to the generation procedure: encoded representations $\boldsymbol{z}$ in the latent space are indeed assumed to follow the prior distribution $p_\theta(\boldsymbol{z})$, leading to a more structured space.

Nevertheless, the form of the *probabilistic encoder* $p_\theta(\boldsymbol{z}|\boldsymbol{x})$ evokes an impediment in this approach. From Bayes' theorem, in order to calculate posterior probability $p_\theta(\boldsymbol{z}|\boldsymbol{x})$, it is necessary to compute $p_\theta(\boldsymbol{x})$ from (4). Besides, the objective of VAE is to find the optimal parameter θ^* that maximizes the probability of the data (i.e., probability of generating real data):

$$\theta^* = \arg\max_\theta \prod_{i=1}^{N} p_\theta(\boldsymbol{x}^{(i)}) = \arg\max_\theta \sum_{i=1}^{N} \log p_\theta(\boldsymbol{x}^{(i)}). \tag{5}$$

Unfortunately, this computation is intractable (i.e. requires exponential time to compute), because it is very expensive to check all the possible values of $\boldsymbol{z}$ and sum them up. For that reason, it is required to use an approximation function for $p_\theta(\boldsymbol{z}|\boldsymbol{x})$: $q_\phi(\boldsymbol{z}|\boldsymbol{x})$, parametrized by ϕ. The considered model is graphically delineated in Figure 3.4.

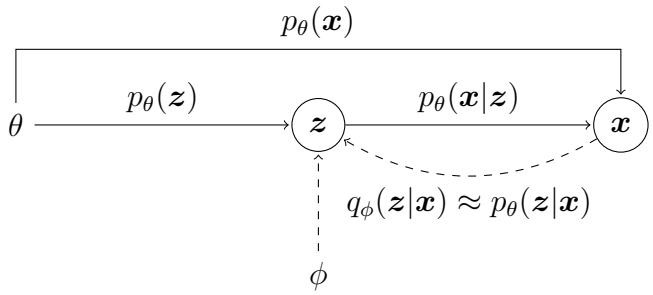


Figure 3.4: *Graphical representation of VAE's model. Solid lines denote the generative procedure and dashed lines denote the approximation $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ of the intractable posterior $p_\theta(\boldsymbol{x}|\boldsymbol{z})$.*

The objective of the considered approximation $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ is to be as close as possible to the intractable posterior $p_\theta(\boldsymbol{x}|\boldsymbol{z})$. To determine the degree to which one probability distribution varies from another, *Kullback-Leibler Divergence* will be introduced in §3.4.4.

3.4.4 Kullback-Leibler Divergence

Definition 3.4.1 (Kullback-Leibler Divergence)**.** *Kullback-Leibler (KL) Divergence* (also called *relative entropy*) $D_{KL}(p||q)$ [22] measures how much probability distribution p differs from another distribution q with the same support $\mathcal{Z}$:

$$D_{KL}(p||q) = \int_{\mathcal{Z}} p(\boldsymbol{z}) \log \frac{p(\boldsymbol{z})}{q(\boldsymbol{z})} d\boldsymbol{z}. \tag{6}$$

The definition of KL divergence of q with respect to p can be formulated using entropy $H(p)$ and cross-entropy $H(p, q)$ [74]:

$$D_{KL}(p||q) = H(p, q) - H(p). \tag{7}$$

Properties

- KL-divergence is always non-negative, i.e., $D_{KL}(p||q) \geq 0$. This observation is known as Gibbs' inequality [75], with $D_{KL}(p||q) = 0$ if and only if $p = q$ almost everywhere.
- $D_{KL}(p||q) \neq D_{KL}(q||p)$ (derives from definition).

3.4.5 Loss Function - ELBO

In VAE's case, the goal is to quantify how much "information" is lost, if (intractable) posterior $p_\theta(\boldsymbol{z}|\boldsymbol{x})$ is approximated by another (approximate) posterior $q_\phi(\boldsymbol{z}|\boldsymbol{x})$. This estimation could be expressed, either by minimizing $D_{KL}(p_\theta(\boldsymbol{z}|\boldsymbol{x})||q_\phi(\boldsymbol{z}|\boldsymbol{x}))$, known as *Forward KL*, or $D_{KL}(q_\phi(\boldsymbol{z}|\boldsymbol{x})||p_\theta(\boldsymbol{z}|\boldsymbol{x}))$, known as *Reverse KL*. The difference between these two approaches, specified in [76], can be seen below:

Case 1: In *Forward KL*, the log term of KL divergence is weighted by $p_\theta(\boldsymbol{z}|\boldsymbol{x})$:

a) If $p_\theta(\boldsymbol{z}|\boldsymbol{x}) = 0$, then the log term of KL divergence does not have any impact in the value of KL. Actually, KL divergence equals 0 and $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ is ignored.

b) If $p_\theta(\boldsymbol{z}|\boldsymbol{x}) > 0$, then the log term plays a role in the value of KL divergence. If $q_\phi(\boldsymbol{z}|\boldsymbol{x}) = 0$, then the log term and of course, KL divergence, are infinite. Therefore, minimization process of KL divergence happens for $q_\phi(\boldsymbol{z}|\boldsymbol{x}) > 0$. The avoidance of $q_\phi(\boldsymbol{z}|\boldsymbol{x}) = 0$, whenever $p_\theta(\boldsymbol{z}|\boldsymbol{x}) > 0$ (*zero-avoiding*), leads to an "outspread" $q_\phi(\boldsymbol{z}|\boldsymbol{x})$, covering the entire $p_\theta(\boldsymbol{z}|\boldsymbol{x})$. Typically, $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ over-estimates the support of $p_\theta(\boldsymbol{z}|\boldsymbol{x})$.

Case 2: In *Reverse KL*, the log term of KL divergence is weighted by $q_\phi(\boldsymbol{z}|\boldsymbol{x})$:

a') If $q_\phi(\boldsymbol{z}|\boldsymbol{x}) = 0$, then the log term of KL divergence does not affect KL, which is 0.

b') If $q_\phi(\boldsymbol{z}|\boldsymbol{x}) > 0$, then the log term does contribute in KL divergence. If $p_\theta(\boldsymbol{z}|\boldsymbol{x}) = 0$, then the log term becomes extreme large, together with KL divergence (infinity). So, the minimization procedure in this case takes place when $p_\theta(\boldsymbol{z}|\boldsymbol{x}) > 0$ and "oblige" $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ to exclude all the areas of space for which $p_\theta(\boldsymbol{z}|\boldsymbol{x}) = 0$. This target is accomplished by fitting well some portion of $p_\theta(\boldsymbol{z}|\boldsymbol{x})$ with $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ and ignoring, by setting $q_\phi(\boldsymbol{z}|\boldsymbol{x}) = 0$, other parts of the area, even if $p_\theta(\boldsymbol{z}|\boldsymbol{x}) > 0$ (*zero-forcing*). Typically, $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ under-estimates the support of $p_\theta(\boldsymbol{z}|\boldsymbol{x})$.

The objective of VAE is consistent with the *Reverse KL*, which will be expanded from (6) below:

$$
\begin{aligned}
&D_{KL}(q_\phi(\boldsymbol{z}|\boldsymbol{x})||p_\theta(\boldsymbol{z}|\boldsymbol{x})) = \\
&= \int_{\mathcal{Z}} q_\phi(\boldsymbol{z}|\boldsymbol{x}) \log \frac{q_\phi(\boldsymbol{z}|\boldsymbol{x})}{p_\theta(\boldsymbol{z}|\boldsymbol{x})} d\boldsymbol{z} \\
&= \int_{\mathcal{Z}} q_\phi(\boldsymbol{z}|\boldsymbol{x}) \log \frac{q_\phi(\boldsymbol{z}|\boldsymbol{x}) p_\theta(\boldsymbol{x})}{p_\theta(\boldsymbol{x}|\boldsymbol{z}) p_\theta(\boldsymbol{z})} d\boldsymbol{z} && \text{; from Bayes' theorem} \\
&= \int_{\mathcal{Z}} q_\phi(\boldsymbol{z}|\boldsymbol{x}) \left[\log \frac{q_\phi(\boldsymbol{z}|\boldsymbol{x})}{p_\theta(\boldsymbol{x}|\boldsymbol{z}) p_\theta(\boldsymbol{z})} + \log p_\theta(\boldsymbol{x}) \right] d\boldsymbol{z} \\
&= \int_{\mathcal{Z}} q_\phi(\boldsymbol{z}|\boldsymbol{x}) \log \frac{q_\phi(\boldsymbol{z}|\boldsymbol{x})}{p_\theta(\boldsymbol{x}|\boldsymbol{z}) p_\theta(\boldsymbol{z})} d\boldsymbol{z} \\
&\quad + \int_{\mathcal{Z}} q_\phi(\boldsymbol{z}|\boldsymbol{x}) \log p_\theta(\boldsymbol{x}) d\boldsymbol{z} \\
&= \log p_\theta(\boldsymbol{x}) + \int_{\mathcal{Z}} q_\phi(\boldsymbol{z}|\boldsymbol{x}) \log \frac{q_\phi(\boldsymbol{z}|\boldsymbol{x})}{p_\theta(\boldsymbol{x}|\boldsymbol{z}) p_\theta(\boldsymbol{z})} d\boldsymbol{z} && \text{; from } \int_{\mathcal{Z}} q_\phi(\boldsymbol{z}|\boldsymbol{x}) d\boldsymbol{z} = 1 \\
&= \log p_\theta(\boldsymbol{x}) + \int_{\mathcal{Z}} q_\phi(\boldsymbol{z}|\boldsymbol{x}) \left[\log \frac{q_\phi(\boldsymbol{z}|\boldsymbol{x})}{p_\theta(\boldsymbol{z})} - \log p_\theta(\boldsymbol{x}|\boldsymbol{z}) \right] d\boldsymbol{z} \\
&= \log p_\theta(\boldsymbol{x}) + \int_{\mathcal{Z}} q_\phi(\boldsymbol{z}|\boldsymbol{x}) \log \frac{q_\phi(\boldsymbol{z}|\boldsymbol{x})}{p_\theta(\boldsymbol{z})} d\boldsymbol{z} \\
&\quad - \int_{\mathcal{Z}} q_\phi(\boldsymbol{z}|\boldsymbol{x}) \log p_\theta(\boldsymbol{x}|\boldsymbol{z}) d\boldsymbol{z} && \text{; from } \mathop{\mathbb{E}}_{q_\phi}[p_\theta] = \int_{\mathcal{Z}} p_\theta q_\phi d\boldsymbol{z} \\
&= \log p_\theta(\boldsymbol{x}) + D_{KL}(q_\phi(\boldsymbol{z}|\boldsymbol{x})||p_\theta(\boldsymbol{z})) - \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}[\log p_\theta(\boldsymbol{x}|\boldsymbol{z})] . && (8)
\end{aligned}
$$

From the Properties of KL divergence, (8) becomes:

$$
\begin{aligned}
&D_{KL}(q_\phi(\boldsymbol{z}|\boldsymbol{x})||p_\theta(\boldsymbol{z}|\boldsymbol{x})) \geq 0 \Rightarrow \\
&\quad \Rightarrow \log p_\theta(\boldsymbol{x}) + D_{KL}(q_\phi(\boldsymbol{z}|\boldsymbol{x})||p_\theta(\boldsymbol{z})) - \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}[\log p_\theta(\boldsymbol{x}|\boldsymbol{z})] \geq 0 \\
&\quad \Rightarrow \log p_\theta(\boldsymbol{x}) \geq -D_{KL}(q_\phi(\boldsymbol{z}|\boldsymbol{x})||p_\theta(\boldsymbol{z})) + \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}[\log p_\theta(\boldsymbol{x}|\boldsymbol{z})] . && (9)
\end{aligned}
$$

The LHS of the equation (9) reflects the (log)-probability of the data (i.e., the probability of generating real data), which we strove for maximization from the

beginning. As mentioned in Problem Outline, it is computationally intractable to compute data probability directly. The RHS of (9) is called *Evidence Lower BOund* (*ELBO*), or *Variational Lower BOund*, since it bounds the probability of the data. Therefore, *maximization of the ELBO results in maximization of the data probability.*

$$ELBO = -D_{KL}(q_\phi(\boldsymbol{z}|\boldsymbol{x})||p_\theta(\boldsymbol{z})) + \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log p_\theta(\boldsymbol{x}|\boldsymbol{z})\right], \tag{10}$$

where:

- $-D_{KL}(q_\phi(\boldsymbol{z}|\boldsymbol{x})||p_\theta(\boldsymbol{z}))$: *regularization* term, as it applies a constraint on the approximate posterior. Practically, it regularises the organisation of the latent space.
- $\mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log p_\theta(\boldsymbol{x}|\boldsymbol{z})\right]$: *reconstruction* term, since it measures the likelihood of the reconstructed data. Practically, it contributes to the form of the best possible encoding-decoding scheme.

Instead of maximizing the *ELBO*, it is equivalent to minimize the negative *ELBO*, which denotes the loss function:

$$\mathcal{L}(\theta, \phi) = -ELBO = D_{KL}(q_\phi(\boldsymbol{z}|\boldsymbol{x})||p_\theta(\boldsymbol{z})) - \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log p_\theta(\boldsymbol{x}|\boldsymbol{z})\right]. \tag{11}$$

Therefore, the goal in VAE is to find the best encoding-decoding scheme (i.e., the best parameters θ^*, ϕ^*) that minimize $\mathcal{L}$:

$$\theta^*, \phi^* = \arg\min_{\theta,\phi} \mathcal{L}(\theta, \phi). \tag{12}$$

3.4.6 Reparameterization Trick

In order to optimize loss function (11), it is necessary to differentiate it with respect to the parameters θ and ϕ. However, the gradient with respect to ϕ is troublesome, due to the fact that *reconstruction* term requires sampling from $q_\phi(\boldsymbol{z}|\boldsymbol{x})$. More analytically, the gradient of loss with respect to ϕ is:

$$\nabla_\phi \mathcal{L}(\theta, \phi) = \nabla_\phi D_{KL}(q_\phi(\boldsymbol{z}|\boldsymbol{x})||p_\theta(\boldsymbol{z})) - \nabla_\phi \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log p_\theta(\boldsymbol{x}|\boldsymbol{z})\right]. \tag{13}$$

The authors of [77] approximated the term $\nabla_\phi \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log p_\theta(\boldsymbol{x}|\boldsymbol{z})\right]$ using Monte-Carlo integration [78] and proved that with this stochastic approximation, the variance of the gradient approximation was significantly high.

To overcome this obstacle, the *reparameterization trick* was proposed [18]:

Let $\boldsymbol{z}$ be a continuous random variable. The goal is to generate samples from $q_\phi(\boldsymbol{z}|\boldsymbol{x})$, where $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ is a conditional distribution. It is possible to express the random variable $\boldsymbol{z}$ as a deterministic variable $\boldsymbol{z} = g_\phi(\boldsymbol{\epsilon}, \boldsymbol{x})$, where $\boldsymbol{\epsilon}$ is an auxiliary independent random variable and $g_\phi(.)$ is a transformation function parametrized by ϕ, which converts $\boldsymbol{\epsilon}$ to $\boldsymbol{z}$.

A graphical representation of this logic is viewed in Figure 3.5. Sampling is stochastic process and thus, it is not possible for the error to be backpropagated through the network (left figure). With reparameterization trick (right figure), not only the error can be backpropagated through the deterministic nodes that express the distribution's parameters, but also stochasticity is maintained through node $\boldsymbol{\epsilon}$.

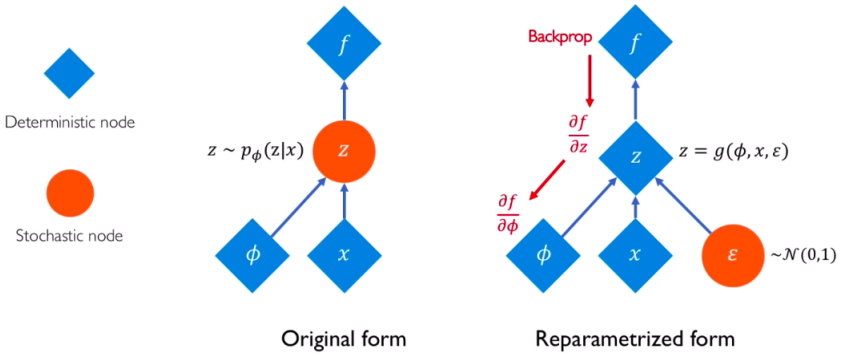


Figure 3.5: *Illustration of reparameterization trick. Image source: Towards Data Science*

3.4.7 Problem Insight

Let the prior $p_\theta(\boldsymbol{z})$ be the standard multivariate Gaussian distribution $p_\theta(\boldsymbol{z}) = \mathcal{N}(0, I)$. Also, likelihood (i.e., probabilistic decoder) $p_\theta(\boldsymbol{x}|\boldsymbol{z})$ is assumed to be, either a multivariate Gaussian $p_\theta(\boldsymbol{x}|\boldsymbol{z}) = \mathcal{N}(\mu, \sigma^2 I)$ or a multivariate Bernoulli $p_\theta(\boldsymbol{x}|\boldsymbol{z}) = Bernoulli(p)$, depending on the type of data. As mentioned in the previous sections, true posterior $p_\theta(\boldsymbol{z}|\boldsymbol{x})$ is intractable. Hence, it will be approached by $q_\phi(\boldsymbol{z}|\boldsymbol{x})$. Let approximate posterior (i.e., probabilistic encoder) be a multivariate Gaussian with a diagonal covariance structure $q_\phi(\boldsymbol{z}|\boldsymbol{x}^{(i)}) = \mathcal{N}(\mu, \sigma^2 I)$, where μ and σ are the mean and standard deviation of the sample $\boldsymbol{x}^{(i)}$ respectively. With j being an index to the latent vector $\boldsymbol{z}$ (of dimensionality J), μ_j, σ_j are the j-th elements of μ, σ respectively.

If the data takes real values, the probabilistic decoder is modeled as a neural network with Gaussian output, while for binary data, it is modeled as a neural network with Bernoulli output. For the encoder, a neural network with Gaussian output is used. More precisely, the encoder outputs two vectors: a vector of means, μ, and another vector of standard deviations, σ. Intuitively, the mean determines where the latent representation of an input should be located in the low-dimensional space, while the standard deviation controls the distance from the aforementioned point the encoding could have. This permits the decoder to not just decode fixed vectors in the latent space, but ones that marginally differ. This happens because the decoder is exposed to a variations of the encoding of the same input during training.

The purpose of Variational Autoencoder is to sample from $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ and generate almost similar output.

Training

The training of this model (depicted in Figure 3.6) is conducted as follows:

Step 1: Input $\boldsymbol{x}$ is passed through probabilistic encoder and mapped into a distribution $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ (i.e., the mean and the standard deviation of a Gaussian distribution) over the latent space $\boldsymbol{z}$, from which $\boldsymbol{x}$ could have been generated.

Step 2: A point $\boldsymbol{z}$ is sampled from that distribution $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ in the latent space.

Step 3: The sampled point $\boldsymbol{z}$ is passed through the probabilistic decoder (i.e., $p_\theta(\boldsymbol{x}|\boldsymbol{z})$), which produces a distribution over the possible values of $\boldsymbol{x}$.

Step 4: The reconstruction error between input and output is evaluated and backpropagated through the network.

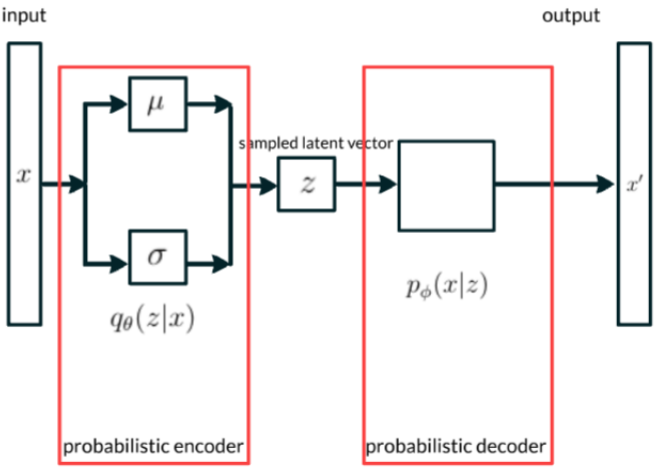


Figure 3.6: *Architecture of Variational Autoencoder*

Reparameterization trick

Due to the choice of a multivariate Gaussian with a diagonal covariance structure as the probabilistic encoder, reparameterization trick [18] is modified as follows:

Instead of sampling from $\mathcal{N}(\mu, \sigma^2 I)$, i.e., $\boldsymbol{z} \sim q_\phi(\boldsymbol{z}|\boldsymbol{x}) = \mathcal{N}(\mu, \sigma^2 I)$, sampling is conducted from $\boldsymbol{z} = \mu + \sigma \odot \boldsymbol{\epsilon}$, where $\odot$ refers to the element-wise product and $\boldsymbol{\epsilon} \sim \mathcal{N}(0, I)$.

Regularization term

The *regularization* term of loss function, for the j-th element of $\boldsymbol{z}$, is:

$$
\begin{aligned}
&-D_{KL}(q_\phi(z_j|\boldsymbol{x}^{(i)})||p_\theta(z_j)) = \\
&\quad = \int \frac{1}{\sigma_j\sqrt{2\pi}} exp(-\frac{(z_j-\mu_j)^2}{2\sigma_j^2}) \log \left(\frac{\frac{1}{\sqrt{2\pi}} exp(-\frac{z_j^2}{2})}{\frac{1}{\sigma_j\sqrt{2\pi}} exp\left(-\frac{(z_j-\mu_j)^2}{2\sigma_j^2}\right)} \right) dz_j \\
&\quad = \int \frac{1}{\sigma_j\sqrt{2\pi}} exp(-\frac{(z_j-\mu_j)^2}{2\sigma_j^2}) \log \left(\frac{exp(-\frac{z_j^2}{2})}{\frac{1}{\sigma_j} exp\left(-\frac{(z_j-\mu_j)^2}{2\sigma_j^2}\right)} \right) dz_j \\
&\quad = \int \frac{1}{\sigma_j\sqrt{2\pi}} exp(-\frac{(z_j-\mu_j)^2}{2\sigma_j^2}) \left[\log(exp(-\frac{z_j^2}{2})) - \log(\frac{1}{\sigma_j}) - \log(exp(-\frac{(z_j-\mu_j)^2}{2\sigma_j^2})) \right] dz_j \\
&\quad = \int \frac{1}{\sigma_j\sqrt{2\pi}} exp(-\frac{(z_j-\mu_j)^2}{2\sigma_j^2}) \left[-\frac{z_j^2}{2} + \log(\sigma_j) + \frac{(z_j-\mu_j)^2}{2\sigma_j^2} \right] dz_j \\
&\quad = \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})} \left[-\frac{\boldsymbol{z_j}^2}{2} + \log(\sigma_j) + \frac{(\boldsymbol{z_j}-\mu_j)^2}{2\sigma_j^2} \right] \qquad \text{; from the definition of expectation.} \\
&\quad = \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})} \left[-\frac{\boldsymbol{z_j}^2}{2} + \log(\sigma_j) + \frac{(\boldsymbol{z_j}-\mu_j)^2}{2\sigma_j^2} \right] \qquad \text{; bold is used for random variable.} \\
&\quad = -\frac{1}{2}\mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})} \left[\boldsymbol{z_j}^2 \right] + \log(\sigma_j) + \frac{1}{2\sigma_j^2}\mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})} \left[(\boldsymbol{z_j}-\mu_j)^2 \right]. \qquad (14)
\end{aligned}
$$

from $\sigma_j^2 = \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}[(\boldsymbol{z_j}-\mu_j)^2]$, (14) becomes:

$$
\begin{aligned}
&-D_{KL}(q_\phi(z_j|\boldsymbol{x}^{(i)})||p_\theta(z_j)) = \\
&= -\frac{1}{2}\mathbb{E}_{q_\phi(z_j|\boldsymbol{x})}\left[\boldsymbol{z_j^2}\right] + \log(\sigma_j) + \frac{1}{2} \\
&= -\frac{1}{2}\mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[(\boldsymbol{z_j} - \mu_j + \mu_j)^2\right] + \log(\sigma_j) + \frac{1}{2} \\
&= -\frac{1}{2}\mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[(\boldsymbol{z_j} - \mu_j)^2 + 2\mu_j(\boldsymbol{z_j} - \mu_j) + \mu_j^2\right] + \log(\sigma_j) + \frac{1}{2} \\
&= -\frac{1}{2}\mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[(\boldsymbol{z_j} - \mu_j)^2\right] - \frac{1}{2}2\mu_j\mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[(\boldsymbol{z_j} - \mu_j)\right] - \frac{1}{2}\mu_j^2 + \log(\sigma_j) + \frac{1}{2} \\
&= -\frac{1}{2}\sigma_j^2 - 0 - \frac{1}{2}\mu_j^2 + \frac{1}{2}\log(\sigma_j^2) + \frac{1}{2} \\
&= \frac{1}{2}\left[1 + \log(\sigma_j^2) - \mu_j^2 - \sigma_j^2\right].
\end{aligned} \tag{15}
$$

From the above equation, *regularization* term for all the elements of $\boldsymbol{z}$ becomes:

$$
-D_{KL}(q_\phi(\boldsymbol{z}|\boldsymbol{x}^{(i)})||p_\theta(\boldsymbol{z})) = \sum_{j=1}^{J}\frac{1}{2}\left[1 + \log(\sigma_j^2) - \mu_j^2 - \sigma_j^2\right]. \tag{16}
$$

Reconstruction term

To determine the *reconstruction* term, the reparameterization trick will be used. Assuming L samples (i.e., stochastic draws) from the posterior $q_\phi(\boldsymbol{z}|\boldsymbol{x}^{(i)})$ and according to Monte-Carlo estimate of expectation:

$$
\mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log p_\theta(\boldsymbol{x}|\boldsymbol{z})\right] \simeq \frac{1}{L}\sum_{l=1}^{L}\log p_\theta(\boldsymbol{x}|\boldsymbol{z}^{(l)}), \tag{17}
$$

where $\boldsymbol{z}^{(l)} = \mu + \sigma\boldsymbol{\epsilon}^{(l)}$ and $\boldsymbol{\epsilon}^{(l)} \sim \mathcal{N}(0, 1)$ from reparameterization trick.

Loss function

Combining (16) and (17), the total loss from (11) is:

$$
\mathcal{L}(\theta, \phi) = -\frac{1}{2}\sum_{j=1}^{J}\left[1 + \log(\sigma_j^2) - \mu_j^2 - \sigma_j^2\right] - \frac{1}{L}\sum_{l=1}^{L}\log p_\theta(\boldsymbol{x}|\boldsymbol{z}^{(l)}). \tag{18}
$$

Chapter 4

Traffic Matrix Estimation

4.1 Problem Statement

Traffic Matrix Estimation involves the inference of traffic transmitted between every pair of nodes in a network from link-level measurements. It can be stated as follows.

Let n denote the nodes and m the links of a network. The relationship between the traffic matrix, the routing matrix and the link counts can be described by a system of linear equations:

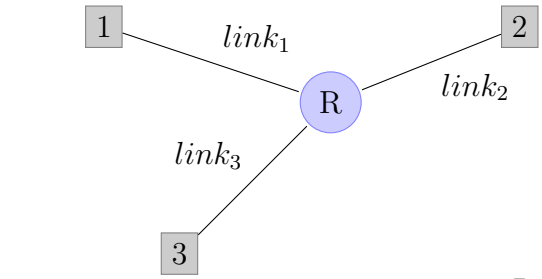
$$\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}, \tag{19}$$

where:

- $\boldsymbol{y}$ is the $m \times 1$ vector of link counts. It is assumed that routing is fixed during the measurement period.
- $\boldsymbol{A}$ is the $m \times n^2$ routing matrix, describing the topology. An element $a_{i,j}$ is equal to 1, if OD flow j traverses link i, or 0 otherwise.
- $\boldsymbol{x}$ is the $n \times n$ traffic matrix, whose element at row i and column j represents the traffic between origin node i and destination node j. It is usually organized as a $n^2 \times 1$ vector.

In this context, the problem is to estimate $\boldsymbol{x}$, given $\boldsymbol{y}$ and $\boldsymbol{A}$. This is not straightforward, as the number of OD flows n^2 (unknown quantities) is almost always much larger than the number of link measurements m (known quantities). Therefore, the aforementioned system of linear equations is heavily under-determined, or ill-posed (i.e., matrix $\boldsymbol{A}$ is not full rank and there are many solutions that fit the observations). This ill-posed nature of the TME NT problem is usually addressed by using statistical models and regularization to impose additional structural assumptions.

A simple example of a network with its traffic can be seen in Figure 4.1. It is obvious that there are many more OD flows than link measurements.



$$\begin{bmatrix} l_1 \\ l_2 \\ l_3 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} OD_{1,1} \\ OD_{1,2} \\ OD_{1,3} \\ OD_{2,1} \\ OD_{2,2} \\ OD_{2,3} \\ OD_{3,1} \\ OD_{3,2} \\ OD_{3,3} \end{bmatrix}$$

Figure 4.1: *Simple network and traffic. R is a router and $1, 2, 3$ are nodes. (19) is replaced for this example.*

4.2 Link-Level Measurements

Link-level measurements are obtainable via the Simple Network Management Protocol (SNMP) [23]. SNMP is one of the widely accepted protocols that manages and monitors network devices in Internet Protocol (IP) networks; from routers, modems and switches, to servers and workstations.

The SNMP data that is available on a device is defined in a abstract data structure known as a Management Information Base (MIB). An SNMP poller periodically requests the appropriate SNMP MIB data from a device through an interface, typically UDP port 161 [4]. The polling period varies from 1 minute to several minutes, but the default is 5 minutes. That is why the traffic data that we acquire contain averages over 5 minutes intervals. Since every router maintains a cyclic counter of the number of bytes transmitted and received on each of its interfaces, we can obtain basic traffic statistics for the entire network with little additional infrastructure support – all we need is an SNMP poller that periodically records these counters.

Nonetheless, SNMP data have a few drawbacks. First and foremost, it is susceptible to error, due to their transmission via unreliable UDP protocol [79]. Moreover, link measurements might be inaccurate, due to poor vendor implementations. Another disadvantage of SNMP data is that it only provides aggregate link statistics, omitting details such as types of traffic on the link and the traffic source and destination.

Despite all these, SNMP data is, at present, the easiest way to obtain large-scale traffic data.

4.3 Prior Work

The problem of Traffic Matrix Estimation has been studied thoroughly in the past decades. The main focus of the majority of the proposed methods was on confronting the ill-posed nature of TME, by leveraging a variety of techniques. In [5], Vardi assumed that OD pairs are generated from a collection of independent Poisson distribution, while in [80], Cao et al. conjectured that each OD pair follows a Gaussian distribution. However, studies [81] showed that the aforementioned assumptions were particularly hinged on the characteristics of data and were not true in any general case.

To tackle the under-constrained system, the authors in [82], [83] used information from additional sources. More precisely, Zhang et al. [82] exploited a spatial model (i.e., OD flows are dependent of one another) called gravity model and used additional SNMP data to calibrate their model. In [83], authors' goal was to increase the rank of routing matrix $\boldsymbol{A}$, by altering the weights of each link (and therefore the paths that each OD pair follows). For that reason, they introduced a temporal model (i.e., each OD flow is dependent on its past) called route change.

Due to the fact that most known models using purely spatial, or purely temporal information do not have a good performance in estimating TMs, authors in [39], [84], [85] leveraged both spatial and temporal information for better results. In [39], fanout method was presented, which utilized solely measurements every few days to obtain traffic matrix (we have no information about routing matrix). Besides, in [84], Soule et al. introduced two novel methods: PCA method and Kalman method. In the former, rather than estimating all OD flows (ill-posed problem), only the most important eigenflows are determined, while in the latter, state space models from dynamic linear systems theory were exploited to capture the evolution of a system. All these methods made use of spatial flow measurements. The authors of [85] came up with a different technique, called Sparsity Regularized Matrix Factorization (SRMF), which finds sparse, low-rank approximations of TMs that account for spatial and temporal properties of real TMs.

The authors of [86] realized that two-dimensional matrices are not sufficient to capture all spatial and temporal information, such as traffic periodicity, which can be employed in TME. For that reason, they proposed higher-order tensors, and more precisely 3D tensors to model traffic data.

Nowadays, the network traffic flows have become much more compound as a result of the advancement of modern network applications. The traffic flows experience a much wider range of statistical characteristics [87], [88]. With the progress of neural networks [63], many scientists focused on leveraging them to solve the problem of TME.

One of the first who implemented neural networks for traffic matrix estimation was Jiang et al. [89]. More specifically, their method relied on a back-propagation neural network (BPNN), combined with the iterative proportional fitting procedure

(IPFP). After time-frequency analysis on end-to-end traffic, authors in [90] proposed the decomposition into low-frequency and high-frequent component. The former reflected the change trend of traffic and was defined by an auto-regressive (AR) model, whilst the latter expressed the fluctuations and mutations of traffic and was formulated by an artificial neural network (namely a BPNN). BPNN is also used in [91], where input is formed as the product of the Moore-Penrose inverse of the network's routing matrix and the link load vector. The expectation maximization (EM) algorithm is applied to the output of the BPNN in order to improve the estimation accuracy. A feedforward back-propagation neural network trained with the Levernberg-Marquardt algorithm (LMA) is employed in [92].

Nie et al. [93] were at the forefront of incorporating Deep Learning [8], [34] methods, and more precisely Deep Belief Networks (DBN), to estimate large-scale traffic matrices. Based on historical data (i.e., previously obtained TMs), the objective lies in learning the characteristics of these priors and predicting future TMs, that have similar properties with the observed traffic matrices. Zhao et al. [94] decompose the original TM series into multilevel subseries using the discrete wavelet decomposition. A CNN extracts the spatial dependencies among flows, whilst a Long Short-Term Memory neural network (LSTM) with a self-attention mechanism captures the temporal evolution features. In [95], authors made use of a Generative Adversarial Network (GAN) to address traffic matrix estimation problem.

4.4 Optimization of Objective Functions

Gradient descent is a first-order iterative optimization algorithm for minimizing a differentiable objective function which, in our case, is the loss function of the neural network. This is achieved by computing the gradient, or slope, (i.e., the first-order derivative) of the loss function with respect to its parameters and updating them in the opposite direction of the slope increase.

The amount that the weights are updated during training is referred to as the *step size* or the *learning rate*. It is challenging to determine the value of learning rate (i.e., η), because small values might result in a long or endless training process, while big values may lead to sub-optimal solutions.

Depending on the amount of data used for the computation of the gradient of the objective function, the following variants of gradient descent are defined [64]:

- *Batch gradient descent*. In batch gradient descent, the entire dataset is taken into consideration to take a single step. In other words, the gradients for the whole dataset have to be calculated so as to perform just one update. This method is guaranteed to converge to the global minimum for convex error surfaces and to a local minimum for non-convex surfaces. Despite that, batch gradient descent can be very slow and is intractable for datasets that don't fit in memory. It is also unsuitable for online (i.e., on-the-fly) learning. Given a function $\mathcal{L}(\theta)$, the updated parameters θ, taking into consideration the learning rate η, are calculated as follows:

$$\theta = \theta - \eta \cdot \nabla_\theta \mathcal{L}(\theta). \tag{20}$$

- *Stochastic gradient descent* (*SGD*). In stochastic gradient descent, one example (i.e., training data point) is considered to take a single step. On the one hand, it converges faster when the dataset is large, (e.g., in online learning), as it causes updates to the parameters more frequently. On the other hand, frequent updates of high variance make the objective function to fluctuate a lot, which might hinder convergence to the exact minimum. Given a function $\mathcal{L}(\theta; x_i; y_i)$, the updated parameters θ for a datapoint x_i and its label y_i, taking into consideration the learning rate η, are calculated as follows:

$$\theta = \theta - \eta \cdot \nabla_\theta \mathcal{L}(\theta; x_i; y_i). \tag{21}$$

- *Mini-batch gradient descent*. Mini-batch gradient descent takes the best of both worlds and performs an update for every mini-batch of n training examples:

$$\theta = \theta - \eta \cdot \nabla_\theta \mathcal{L}(\theta; x_{i:i+n}; y_{i:i+n}). \tag{22}$$

In our proposed methods, we exploit the Adam (adaptive moment estimation) optimizer [96] instead of the classical stochastic gradient descent procedure to update network parameters based on training data. This method computes individual

adaptive learning rates for different parameters from estimates of first and second moments of the gradients. It is achieved by maintaining an exponentially decaying average of past gradients and past squared gradients, using bias-corrected estimates of their first and second moments. It has been demonstrated to be fast in terms of convergence and highly robust for modeling complex structures.

4.5 Proposed Methods

4.5.1 Traffic Matrix Synthesis

Traffic matrices have many uses, apart from the simple fact that they provide a better understanding of a network. They can be utilized for anomaly detection, network management and planning, load balancing and designing network protocol. Unfortunately, not only is it computationally difficult to measure traffic flows directly, but also, the number of publicly available TM datasets is limited. This inadequate quantity of datasets constitutes a severe impediment to network operators or scientists, who are interested in testing and evaluating the efficiency of their proposed algorithms in real situations.

To get around this problem, constructing plausible traffic matrices artificially is an excellent solution. The key note here lies in the production of traffic matrices, that do not deviate substantially from the observed ones; namely, retaining their spatio-temporal attributes. Aside from the aforementioned usage, traffic matrix synthesis is an indispensable intermediate stage in traffic matrix estimation, that will be discussed in the following subsection.

In the current thesis, we leverage a Variational Autoencoder for Traffic Matrix Synthesis. The goal is to provide previously observed TMs (or, explicitly constructed TMs for this specific objective) in this deep generative model, in order to generate new TMs that have similar characteristics to the employed examples.

VAE is responsible for identifying the underlying distribution of training data (i.e., how they are made). From the loss function (11) that has to be minimized during training, the *regularization* term $D_{KL}(q_\phi(\boldsymbol{z}|\boldsymbol{x})||p_\theta(\boldsymbol{z}))$ -which also has to be minimized- measures how much approximate probability distribution $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ differs from prior $p_\theta(\boldsymbol{z})$ that is assumed to be a standard Gaussian (see subsection 3.4.7). This means that, when randomly generating, if we sample a vector from the same prior distribution $p_\theta(\boldsymbol{z}) = \mathcal{N}(0, I)$ of the encoded vectors, the decoder will generate a new synthetic data point. This data point is guaranteed to have characteristics similar to the observed points, because during training, we have assured that latent space is continuous, allowing any random point to have a meaningful representation once decoded.

4.5.2 Traffic Matrix Estimation

From the training procedure described in §3.4.7, we discovered the best encoding-decoding scheme of VAE (i.e., the best parameters θ and ϕ of encoder and decoder) by minimizing loss function (11). As a result, the trained decoder learns the underlying distribution of training data and is able to generate new synthetic data points, respecting the characteristics of the flows already grasped.

Based on (19), TME involves the estimation of traffic matrix $\boldsymbol{x}$, given link-level measurements $\boldsymbol{y}$ and routing matrix $\boldsymbol{A}$. More precisely, we take advantage of the trained decoder that can synthesize artificial examples from the latent space $\boldsymbol{z}$ and we assume that the solution we are looking for (what is the TM $\boldsymbol{x}$) can be generated by this decoder. Hence, we transform TME into a minimization problem in the latent space:

$$\arg\min_{\boldsymbol{z}} \left[\left\| \boldsymbol{y} - \boldsymbol{A}d(\boldsymbol{z}) \right\|_2^2 \right], \tag{23}$$

where $d(.)$ is the trained generative model (i.e., decoder). So, the goal is to find the best parameter $\boldsymbol{z}$, in order to minimize the difference between observed link counts $\boldsymbol{y}$ and estimated link counts $\boldsymbol{A}d(\boldsymbol{z})$, which are formed by passing $\boldsymbol{z}$ through the decoder. After obtaining the best $\boldsymbol{z}$, we acquire the estimation of best traffic matrix $\hat{\boldsymbol{x}} \simeq d(\boldsymbol{z})$. The decoder is differentiable, thus the problem is solved by deploying a gradient based optimizer and more specifically, Adam optimizer.

A second method proposed for TME involves the addition of a regularization term to the aforementioned objective function:

$$\arg\min_{\boldsymbol{z}} \left[\left\| \boldsymbol{y} - \boldsymbol{A}d(\boldsymbol{z}) \right\|_2^2 + c \left\| \boldsymbol{z} \right\|_2^2 \right], \tag{24}$$

where c is a measure of trade-off between measurement error and the importance of prior, prompting the examination of regions that are selected by the decoder. The Adam optimizer is also utilized in this method.

In both (23) and (24), a starting point for $\boldsymbol{z}$ has to be determined. This initial vector z_0 could be chosen at random, but this might hamper the procedure, as more steps will be necessary to reach the minimum of the respective objective function. The total iterations could be diminished, in case we opt for a "good" initial point. For that reason, we explore K random latent vectors and select the one with the minimum distance to the measured link counts:

$$\boldsymbol{z_0} = \boldsymbol{z_k} \text{ s.t. } \left\| \boldsymbol{y} - \boldsymbol{A}d(\boldsymbol{z_k}) \right\|_2^2 \leq \left\| \boldsymbol{y} - \boldsymbol{A}d(\boldsymbol{z_i}) \right\|_2^2, \text{ for } i \in 1, ..., K, \tag{25}$$

or:

$$\boldsymbol{z_0} = \boldsymbol{z_k} \text{ s.t. } \left\| \boldsymbol{y} - \boldsymbol{A}d(\boldsymbol{z_k}) \right\|_2^2 + c \left\| \boldsymbol{z_k} \right\|_2^2 \leq \left\| \boldsymbol{y} - \boldsymbol{A}d(\boldsymbol{z_i}) \right\|_2^2 + c \left\| \boldsymbol{z_i} \right\|_2^2, \text{ for } i \in 1, ..., K. \tag{26}$$

4.5.3 Incremental Optimization

The aforementioned TME methods consist of two steps:

Step 1: Train VAE with previously obtained TMs;

Step 2: Transform TME problem into a minimization problem in the low-dimensional space and solve it using Adam optimizer.

Nevertheless, it is often that objective functions (23) and (24) have plenty of local minima; thus, optimization algorithms can get stuck in a local minimum and not converge to the global minimum. These non-convex functions gave rise to an alternative approach in TME. In this case, the above-noted two-step procedure is combined in one: we incrementally optimize the sequence of objective functions constructed with the sequence of decoder networks that are obtained during different stages of the VAE training by adopting the respective parameter values. This "concurrent" optimization has been used in literature [24] for problems with many local minima and has been proven to converge to the global minimum.

Let $\phi_0, \phi_1, ..., \phi_T$ be the parameters of decoders $d_0, d_1, ..., d_T$ obtained every some number of training epochs. For decoder d_i, equations (23) and (24) are transformed into:

$$\arg\min_{\boldsymbol{z}} \left[\left\| \boldsymbol{y} - \boldsymbol{A} d_i(\boldsymbol{z}) \right\|_2^2 \right], \tag{27}$$

$$\arg\min_{\boldsymbol{z}} \left[\left\| \boldsymbol{y} - \boldsymbol{A} d_i(\boldsymbol{z}) \right\|_2^2 + c \left\| \boldsymbol{z} \right\|_2^2 \right]. \tag{28}$$

From these equations, the best latent vector $\boldsymbol{z_i^*}$ can be found using Adam optimizer. This "current" optimum will be used for the initialization of Adam when optimizing the next objective. So, assuming the decoder d_{i+1} after a predetermined number of training epochs, we will utilize $\boldsymbol{z_i^*}$ as the starting point for the Adam optimizer on the new objective:

$$\arg\min_{\boldsymbol{z}} \left[\left\| \boldsymbol{y} - \boldsymbol{A} d_{i+1}(\boldsymbol{z}) \right\|_2^2 \right], \tag{29}$$

$$\arg\min_{\boldsymbol{z}} \left[\left\| \boldsymbol{y} - \boldsymbol{A} d_{i+1}(\boldsymbol{z}) \right\|_2^2 + c \left\| \boldsymbol{z} \right\|_2^2 \right]. \tag{30}$$

From this, we will get the optimum $\boldsymbol{z_{i+1}^*}$, and so on. Finally, if d_T is the final decoder, the TM estimate will be $\hat{\boldsymbol{x}} = d_T(\boldsymbol{z_T^*})$.

4.6 Dataset: Abilene Network

4.6.1 Overview

To evaluate the aforementioned methods, traffic data collected from the Abilene network [17] was used. Its traffic data arises mainly from major universities in the US and was utilized in several studies [1], [25]–[27].

As can be seen from Figure 4.2, this backbone network is located in North America and comprises of 12 principal nodes (Atlanta is composed of 2 nodes), which distribute information through $12 \cdot 12 = 144$ Origin-Destination (OD) pairs. The topology has also $15 \cdot 2 = 30$ principal-internal links that connect these regions one another. Each node has additionally 2 external links (1 for ingress, 1 for egress traffic) that bridges it with the "outer" world, resulting in $30 + 12 \cdot 2 = 54$ links in total. The capacity of all internal links is 9920000kbps, except Atlanta-Indianapolis and Indianapolis-Atlanta that have capacity 2480000kbps. The traffic data contains averages over 5 minute intervals, for 24 weeks, from March 1st to September 10th, 2004 (there are some missing periods). So, there are $(60/5) \cdot 24 = 288$ samples per day and $288 \cdot 7 = 2016$ samples per week.

Figure 4.2: *Topology of the Abilene network. Image source: [1]*

In Table 4.1, an example of a TM can be viewed. It is obvious that the matrix is not symmetric and the values of each OD pair ranges from a few (or zero, if no packets traverse this link) kbps, to the link capacity. Moreover, the diagonals of the TM are not zero, due to the external links, i.e., there is traffic that enters in the network, and then exits at the same node.

4.6.2 Preprocessing

The provided dataset [17] requires modifications in order to be processed:

Table 4.1: *The 5-minute traffic matrix (real values) in Abilene network from March 1st, 2004 between 00:00 and 00:05, in kbps.*

Origin \ Destination	1	2	3	4	5	6	7	8	9	10	11	12
1	26.667	522.208	1641.339	335.728	413.032	489.875	365.077	817.869	452.061	747.405	388.317	3141.640
2	445.149	82660.749	16283.117	5169.035	3931.403	27351.896	4844.035	18231.909	12803.955	1421.888	8957.483	51748.245
3	3793.693	12735.325	28744.432	13738.253	9830.336	26359.776	6537.749	27775.901	14098.339	1816.941	4495.256	10647.053
4	230.805	2341.483	38518.189	3761.989	8408.763	7207.741	3948.899	21375.568	5723.408	14385.464	10609.464	12248.861
5	239.043	2956.779	16891.501	3693.019	5606.299	9143.904	7130.168	98457.928	7265.264	2947.376	2237.597	8506.651
6	4766.925	9254.733	122044.576	21378.197	33173.784	22849.056	9890.840	24405.043	40616.099	3466.387	13892.187	41138.093
7	420.960	4443.563	26972.272	5394.304	5476.104	8017.757	4022.899	8673.584	12842.411	1223.752	2444.272	12048.437
8	339.661	19394.589	89723.683	9039.381	9030.867	42720.251	13570.251	11369.131	61164.419	2311.541	25519.453	55726.387
9	3897.640	40887.840	53674.288	16345.053	23987.787	83325.448	24767.283	71022.560	136796.045	9591.352	21934.557	111860.741
10	26.667	1041.205	5046.067	10406.912	1436.683	3861.611	2097.072	2000.211	2211.461	14269.608	3297.648	1987.707
11	111.019	15881.547	22512.587	4341.176	11302.768	7691.184	2260.848	17766.373	24845.373	4755.005	1038.357	10517.379
12	11219.101	125937.728	66541.197	36063.421	15439.312	62781.813	32642.733	91675.627	133661.405	1980.576	29760.203	187653.483

Traffic matrix $\boldsymbol{x}$

In the experiments, each traffic matrix is organized as a vector (i.e., 144 rows, due to 144 OD pairs and 1 column).

As mentioned above, traffic data is measured for 24 weeks, so there are 24 different files, one for each week. Each file consists of $(60/5) \cdot 24 \cdot 7 = 2016$ 5-min traffic matrices and each line belongs to one TM. Each line (traffic matrix) contains $144 \cdot 5 = 720$ values, because OD pairs are estimated with 5 different approaches: the real method (i.e., their real values), using SimpleGravity method, SimpleTomogravity method, generalGravity method and generalTomogravity method. *In our models, we use the real values of the OD pairs.*

The units of each real OD pair is expressed in (100 bytes / 5 minutes). In order to convert them in kbps, each value has to be multiplied by $8/(3 \cdot 10^3)$.

During preprocessing, we have to ensure that there are neither faults, nor missing values in the measurements. For the former, if an OD pair surpasses the corresponding link's capacity, the value is considered as mistaken and is replaced with the capacity of the link. For the latter, each traffic matrix is checked to contain 720 values in total and 144 after keeping solely the real values.

Routing matrix $\boldsymbol{A}$

In the experiments, $\boldsymbol{A}$ is $0-1$ matrix with dimensions 30×144; we are interested in the 30 internal links and the 144 OD pairs.

The dataset contains a separate file that corresponds to routing matrix $\boldsymbol{A}$. It is made of five columns, though the 5^{th} is ignored; its value is constant for every line.

The 1st and the 3rd column represent the link's name and the link's index respectively, while 2nd and the 4th describe the OD's name and OD's index respectively. The index of 3rd column indicates the row and the 4th the column that will become 1 in the routing matrix $\boldsymbol{A}$; all other cells are 0.

Link counts $\boldsymbol{y}$

In the experiments, link counts are organized as a vector (i.e., 30 rows, due to 30 internal links and 1 column).

The provided dataset includes the routing matrix and the traffic matrix of the network. However, it does not contain the link counts, which are essential for solving the minimization problem (see (23), (24)). Finding the link counts, given the routing and traffic matrix is straightforward: from (19), it is just a matrix multiplication.

4.7 Implementation

Encoder

The architecture of VAE Encoder is depicted in Table 4.2. The encoder takes a TM $\boldsymbol{x}$ of shape $12 \times 12 \times 1$ as input and after a series of Dropout and Convolutions, ends up with a Dense layer of output shape 64. The output of the latter layer is fed into two separate Dense layers of output 10 (10 is the dimensionality of the latent space). These Dense layers represent the mean and the standard deviation of the Gaussian approximate posterior $q_\phi(\boldsymbol{z}|\boldsymbol{x})$. Finally, the mean and the standard deviation Dense layers are combined in a Sampling Layer, which performs the reparameterization trick and returns $\boldsymbol{z}$. ReLU [97] is chosen as the activation function.

Decoder

Decoder is responsible for the exact opposite procedure; it takes as input the latent representation $\boldsymbol{z}$ and by using Dense and Transposed Convolution layers, it produces its estimation for $\boldsymbol{x}$. ReLU [97] is chosen as the activation function. The architecture of VAE decoder is visible in Table 4.3.

General

- We used the first 13 weeks (i.e., the first $13 \cdot 2016 = 26208$ TMs) as the training set and the 14th week (i.e., the next 2016 TMs) as the testing set.

- The implementation was done using the Keras [98] deep learning API, running on top of the TensorFlow [99] machine learning platform. The code is implemented and run as a Notebook document, through Jupyter Notebook [100].

Table 4.2: *Structure of VAE Encoder.*

Layer type	Kernel	Stride	Padding	Output Shape
Input ($\boldsymbol{x}$)	-	-	-	(12, 12, 1)
Dropout	-	-	-	(12, 12, 1)
Conv2D	(3, 3)	(2, 2)	SAME	(6, 6, 32)
Conv2D	(3, 3)	(2, 2)	SAME	(3, 3, 64)
Conv2D	(3, 3)	(1, 1)	SAME	(3, 3, 128)
Flatten	-	-	-	(1152)
Dense	-	-	-	(64)
Dense (μ)	-	-	-	(10)
Dense (σ)	-	-	-	(10)
Sampling ($\boldsymbol{z}$)	-	-	-	(10)

Table 4.3: *Structure of VAE Decoder.*

Layer type	Kernel	Stride	Padding	Output Shape
Input ($\boldsymbol{z}$)	-	-	-	(10)
Dense	-	-	-	(64)
Dense	-	-	-	(576)
Reshape	-	-	-	(3, 3, 64)
Conv2DTranspose	(3, 3)	(1, 1)	SAME	(3, 3, 128)
Conv2DTranspose	(3, 3)	(2, 2)	SAME	(6, 6, 64)
Conv2DTranspose	(3, 3)	(2, 2)	SAME	(12, 12, 32)
Conv2DTranspose	(3, 3)	(1, 1)	SAME	(12, 12, 1)

- For the training and testing procedures, the units of traffic matrices $\boldsymbol{x}$ and link counts $\boldsymbol{y}$ are $Mbps/100$. Nonetheless, the errors are computed in $Mbps$.
- Adam [96] optimizer was used, with learning rate of 0.001.
- The number of dimensions of the latent space was set to 10.
- Training was executed with batch size 128 and 1000 epochs, though EarlyStopping was implemented to stop training before overfitting occurs. With this method, training runs for approximately 200 epochs (see Figure 4.3).

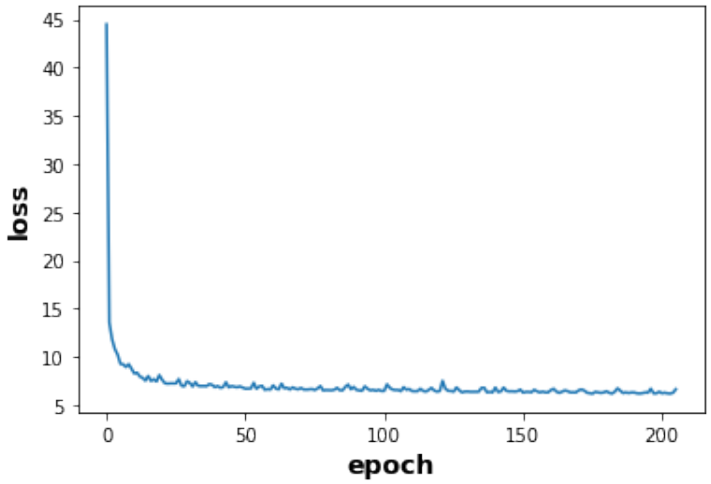


Figure 4.3: *Model loss on training dataset.*

4.8 Results

The performance of the proposed TME methods is evaluated using the following metrics: the Root Mean Square Error (RMSE), the Normalized Mean Absolute Error (NMAE), the Spatial Relative Error (SRE) that expresses the relative estimation error of each individual OD flow over its entire lifetime, and the Temporal Relative Error (TRE) that summarizes the relative estimation error of all OD flows (i.e., the entire TM) at a given time point.

The aforementioned errors are calculated as follows:

$$\text{RMSE}(t) = \frac{\left\|\hat{\boldsymbol{x}}_t - \boldsymbol{x}_t\right\|_2}{\sqrt{N}} = \sqrt{\frac{\sum_{i=1}^{N}\left(\hat{\boldsymbol{x}}_t(i) - \boldsymbol{x}_t(i)\right)^2}{N}}, \tag{31}$$

$$\text{NMAE}(t) = \frac{\left\|\hat{\boldsymbol{x}}_t - \boldsymbol{x}_t\right\|_1}{\left\|\boldsymbol{x}_t\right\|_1} = \frac{\sum_{i=1}^{N}\left|\hat{\boldsymbol{x}}_t(i) - \boldsymbol{x}_t(i)\right|}{\sum_{i=1}^{N}\left|\boldsymbol{x}_t(i)\right|}, \tag{32}$$

$$\text{TRE}(t) = \frac{\left\|\hat{\boldsymbol{x}}_t - \boldsymbol{x}_t\right\|_2}{\left\|\boldsymbol{x}_t\right\|_2} = \frac{\sqrt{\sum_{i=1}^{N}\left(\hat{\boldsymbol{x}}_t(i) - \boldsymbol{x}_t(i)\right)^2}}{\sqrt{\sum_{i=1}^{N}\left(\boldsymbol{x}_t(i)\right)^2}}, \tag{33}$$

$$\text{SRE}(i) = \frac{\left\|\hat{\boldsymbol{x}}_{1:T}(i) - \boldsymbol{x}_{1:T}(i)\right\|_2}{\left\|\boldsymbol{x}_{1:T}(i)\right\|_2} = \frac{\sqrt{\sum_{t=1}^{T}\left(\hat{\boldsymbol{x}}_t(i) - \boldsymbol{x}_t(i)\right)^2}}{\sqrt{\sum_{t=1}^{T}\left(\boldsymbol{x}_t(i)\right)^2}}, \tag{34}$$

where:

- $\boldsymbol{x}$ specifies the true traffic matrix.
- $\hat{\boldsymbol{x}}$ stipulates the estimated traffic matrix.
- $i = 1, 2, ..., N$ indicates each individual OD flow.
- $t = 1, 2, ..., T$ denotes each measurement time point (i.e., TM).
- $\left\|.\right\|_1$ and $\left\|.\right\|_2$ are the absolute and euclidean norm respectively.

Below, minimization(1) refers to the minimization of the objective function (23) and minimization(2) to (24).

As mentioned in Problem Statement and validated during our experiments, by choosing a "good" initial latent vector we can reduce the number of the required optimization iterations. In particular, after selecting the "best" initial latent vector in terms of distance to the measured link counts among 3000 random candidates for minimization(1) and 500 for minimization(2), we managed to reduce the optimization iterations from 10000 to 5000 for both cases. We note that all the subsequently reported results are obtained following this approach.

Figure 4.3 illustrates the the VAE's total loss (i.e., reconstruction error and KL divergence) over the training epochs. From this figure, it is obvious that the maximum value of total loss is in the first epochs. Nevertheless, after approximately 1 to 3 epochs, total loss is decreased significantly and then, as the epochs pass, there is a slight reduction in the total loss. We also defined an EarlyStopping callback, called *EarlyStoppingAtMinLoss*, which is responsible for terminating training when the loss reaches its min, or equivalently, the loss stops decreasing. It takes as argument the patience, i.e., the number of epochs to wait after min has been hit and no improvement is observed. Every time loss is minimized, the weights of the model are stored. If training is terminated because of this callback, the model weights from the end of the best epoch are restored.

Definition 4.8.1 (Cumulative Distribution Function (CDF) [95])**.** Given a set of data points $y_1 \leq y_2 \leq ... \leq y_n$, the cumulative distribution function (cdf) of these points is a step function that jumps up by $\frac{1}{n}$ at each of the n data points. Its value at any specified value $\boldsymbol{z}$, is the fraction of observations of the measured variable that are less than or equal to $\boldsymbol{z}$.

Figure 4.4 depicts the temporal relative errors. The time t of the x-axis corresponds to the 2016 TMs of the testing set for minimization(1) and minimization(2). In this way, $T = 2016$. It can be deduced that temporal errors for minimization(2) are less than minimization(2) almost for every TM. This induction is confirmed from Figure 4.5, where cdf of temporal relative errors is plotted. For instance, there are more points less than 0.5 TRE in minimization(2), comparing to minimization(1). Be that as it may, in both minimization(1) and minimization(2), TRE in almost all TMs is less than 0.6-0.7.

Figure 4.6 illustrates the spatial relative errors of every OD pair. In this way, $N = 144$, because there are 144 OD pairs in total. In general, spatial errors take small values, except one OD flow between 107 and 109, which seems to skyrocket. This behavior could be, either due to an error in this specific OD pair, or due to a steep increase in the traffic flow of this particular OD pair. However, the SRE in minimization(2) is approximately one fifth of the respective error in minimization(1). From Figure 4.7, it is evident that the CDFs of minimization(1) and minimization(2) are very close to each other.

Table 4.4 summarizes the results for the three examined variations of the proposed TME method. As can be seen, minimization(2) with $c = 0.1$ does indeed improve all the employed metrics (particularly SRE) by incorporating the regularization term. Regarding the concurrent incremental optimization, we have used the objective (24). The number of running loops concerning the training of each model was 6 and the number of training epochs after which we obtain each decoder was set to 20. Lastly, the number of optimization iterations for every objective of the sequence was set to 3000. For this case only, the results reported regard the first 500 TMs of the testing set.

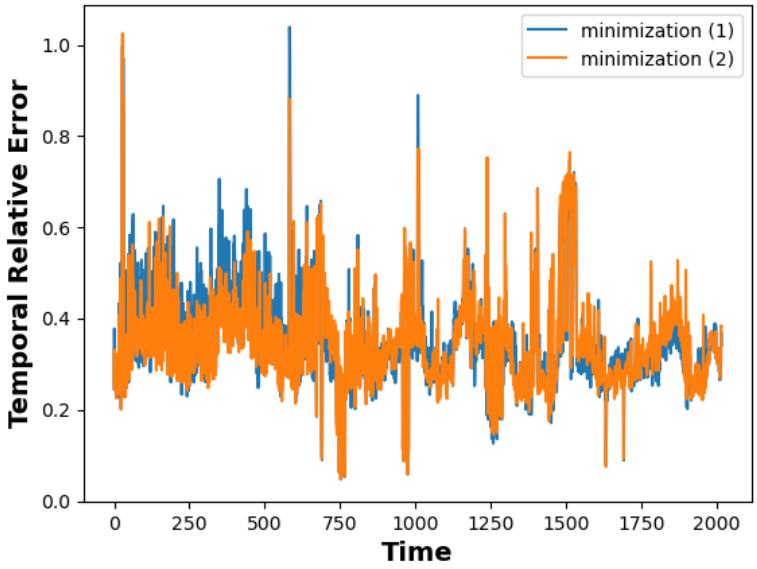


Figure 4.4: *Temporal relative errors.*

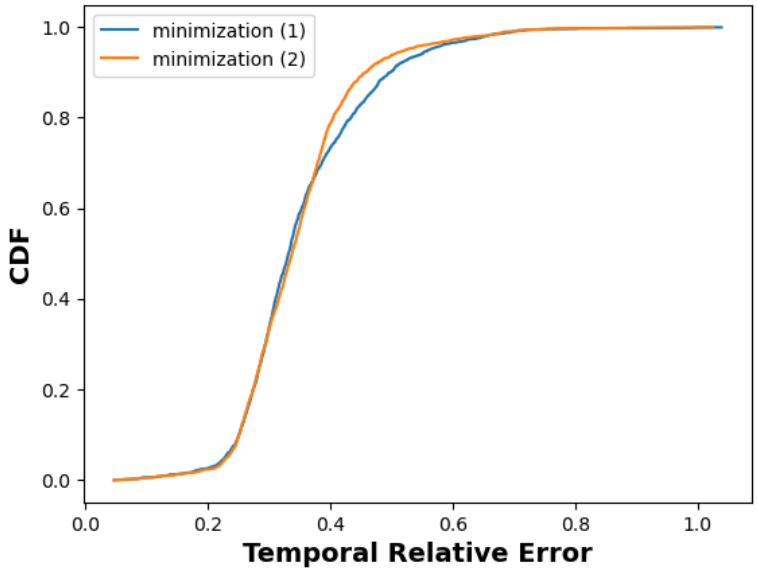


Figure 4.5: *Cumulative distribution function (CDF) of TREs.*

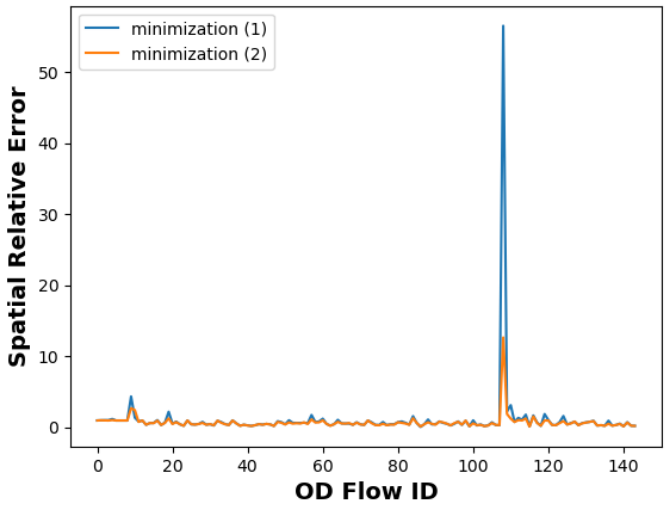


Figure 4.6: *Spatial relative errors.*

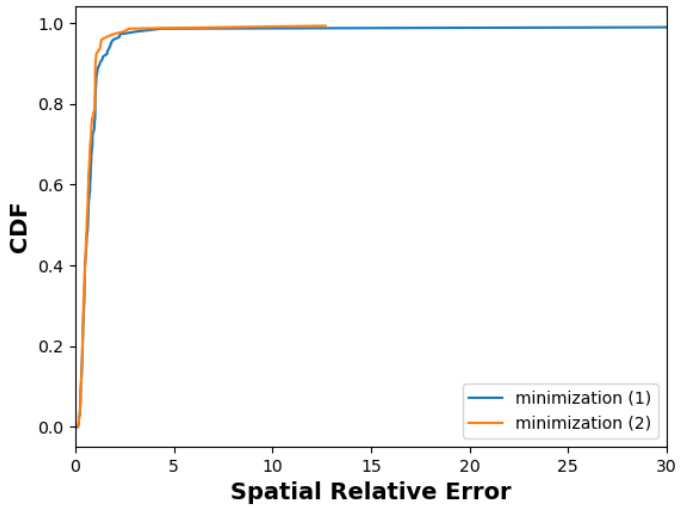


Figure 4.7: *Cumulative distribution function (CDF) of SREs.*

Table 4.4: *Estimation Errors*

VAE Training and Minimization (1)				
Error	**Mean**	**Median**	**Std**	**Max**
RMSE (Mbps)	13.9045	12.6126	5.2089	59.6762
NMAE	0.3544	0.3415	0.0824	0.8176
TRE	0.3557	0.3320	0.1098	1.0387
SRE	1.1233	0.6323	4.6677	56.5583
VAE Training and Minimization (2)				
Error	**Mean**	**Median**	**Std**	**Max**
RMSE (Mbps)	13.6893	12.6701	5.0450	60.6747
NMAE	0.3466	0.3350	0.0748	0.9496
TRE	0.3488	0.3381	0.0989	1.0249
SRE	0.7330	0.5769	1.0698	12.6865
Concurrent Training and Minimization (2)				
Error	**Mean**	**Median**	**Std**	**Max**
RMSE (Mbps)	13.3673	12.4220	5.5080	43.2572
NMAE	0.4262	0.3972	0.1473	1.3309
TRE	0.4033	0.3791	0.1505	1.3824
SRE	1.7464	0.8027	6.9723	81.7783

Chapter 5

Conclusion & Future Work

5.1 Conclusion

In this diploma thesis, after reviewing the major contributions in literature concerning the field of Network Tomography and Traffic Matrix Estimation (TME), the latter is addressed. To deal with the ill-posed inverse problem of TME from link-level measurements, we exploited deep generative models and more precisely, Variational Autoencoder (VAE), instead of relying on sparsity [7] as usual. We leveraged a publicly available dataset with measurements from Abilene network to train the model. By employing the decoder, we transformed TME into a minimization problem in the latent space. The decoder was also utilized for generating new TMs, that are akin to the samples already explored in the training process. Thus, it can be easily perceived that choosing the proper training set is a crucial task; it is important to contain all possible variations of the values of Origin-Destination pairs. Furthermore, we explored the alternative approach of incrementally optimizing the sequence of objective functions corresponding to the sequence of the decoder's parameters, as the latter are produced at different stages of the VAE's training process. Finally, the performance of the proposed methods was assessed and valuable results were deduced.

5.2 Future Work

The current diploma thesis could give rise to future work. Some of the most promising prospects are the following:

Untrained Generative Models

In contrast to classical deep neural networks that consist of a large number of parameters and require big datasets in order to be trained, recently, an innovative approach was proposed that confined the aforementioned prerequisites. These models are called Untrained Generative Models and are based on deep neural networks, with very few parameters, simple architecture and do not need training. Somewhat

counter-intuitively, the models have been shown to achieve propitious results [28], [29].

Conditional Generative Models

Another direction worth exploring is the use of conditional generative models [30], [31]. These generative models are used to sample from a high-dimensional unknown posterior distribution, which allows the generation of multiple solutions from the same measurements.

Deep Reinforcement Learning

To contribute to future research in that domain, a further approach might include the usage of Deep Reinforcement Learning [32]. To grasp the ill-posed inverse inference system and estimate the Traffic Matrix accurately, an agent has to be trained to behave optimally in an environment -by taking actions and receiving rewards-.

Appendix A

Source code

```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from collections import defaultdict
import operator
import os
import gzip
from pathlib import Path
```

```
# calculate A (Routing Matrix)
A = np.zeros((30, 12*12))

cur_dir = Path(os.getcwd())
for entry in os.listdir(cur_dir.parent):
    if os.path.isfile(os.path.join(cur_dir.parent, entry)) and entry == 'A':
        with open(os.path.join(cur_dir.parent, entry), 'r') as fp:
            line = fp.readline()
            while line:
                if not line.startswith('#'):
                    token = line.split()
                    # keep only internal links
                    if int(token[2]) < 31:
                        link_id = int(token[2]) - 1
                        OD_id = int(token[3]) - 1
                        A[link_id][OD_id] = 1
                line = fp.readline()
```

```
# calculate X (Traffic Matrix)
X = []
files = []

cur_dir = Path(os.getcwd())
for entry in os.listdir(cur_dir.parent):
    if os.path.isfile(os.path.join(cur_dir.parent, entry)) and
       entry.startswith('X'):
        files.append(entry)

files = sorted(files)

for file in files:
    with gzip.open(os.path.join(cur_dir.parent, file), 'r') as fp:
        line = fp.readline()
        while line:
            token = line.split()
            for i in range(0, len(token), 5):
                # in kbps, discard outliers
                if float(token[i])*8 / (3*10**3) > 9920000:
                    X.append(9920000)
                else:
                    X.append(float(token[i])*8 / (3*10**3))
            line = fp.readline()

X = np.array(X).reshape((-1,12,12))
```

```
# training is executed for 13 weeks
train_size = 13*24*7*12
# testing is executed for 1 week
test_size = 1*24*7*12

# in Mbps/100
X = X[:(train_size+test_size)] / 100000

# expand dimensions of each TM from (12,12) to (12,12,1)
X_train = np.expand_dims(X[:train_size], -1)
X_test = np.expand_dims(X[train_size:(train_size+test_size)], -1)
```

```
latent_dim = 10
```

```
class Sampling(tf.keras.layers.Layer):
'''
(1) Sampling Layer is a subclass of tf.keras.layers.Layer
(2) Reparameterization trick: use z_mean and z_log_var to sample z
'''
def call(self, inputs):
    z_mean, z_log_var = inputs
    epsilon = tf.random.normal(shape=tf.shape(z_mean))
    return z_mean + tf.math.exp(0.5*z_log_var) * epsilon
```

```
# Encoder
encoder_inputs = tf.keras.Input(shape=(12, 12, 1))
x = tf.keras.layers.Dropout(0.25)(encoder_inputs)
x = tf.keras.layers.Conv2D(
        32, 3, activation="relu", strides=2, padding="same"
    )(x)
x = tf.keras.layers.Conv2D(
        64, 3, activation="relu", strides=2, padding="same"
    )(x)
x = tf.keras.layers.Conv2D(
        128, 3, activation="relu", strides=1, padding="same"
    )(x)
x = tf.keras.layers.Flatten()(x)
x = tf.keras.layers.Dense(64, activation="relu")(x)
z_mean = tf.keras.layers.Dense(latent_dim, name="z_mean")(x)
z_log_var = tf.keras.layers.Dense(latent_dim, name="z_log_var")(x)
z = Sampling()([z_mean, z_log_var])
encoder = tf.keras.Model(
            encoder_inputs, [z_mean, z_log_var, z], name="encoder"
          )
```

```
# Decoder
latent_inputs = tf.keras.Input(shape=(latent_dim,))
x = tf.keras.layers.Dense(64, activation="relu")(latent_inputs)
x = tf.keras.layers.Dense(3 * 3 * 64, activation="relu")(x)
x = tf.keras.layers.Reshape((3, 3, 64))(x)
x = tf.keras.layers.Conv2DTranspose(
        128, 3, activation="relu", strides=1, padding="same"
    )(x)
x = tf.keras.layers.Conv2DTranspose(
        64,  3, activation="relu", strides=2, padding="same"
```

```
    )(x)
x = tf.keras.layers.Conv2DTranspose(
        32,  3, activation="relu", strides=2, padding="same"
    )(x)
decoder_outputs = tf.keras.layers.Conv2DTranspose(
                    1, 3, activation="relu", padding="same"
                  )(x)
decoder = tf.keras.Model(latent_inputs, decoder_outputs, name="decoder")
```

```
class VAE(tf.keras.Model):
'''
Our VAE is a subclass of tf.keras.Model
'''
def __init__(self, encoder, decoder, **kwargs):
    super(VAE, self).__init__(**kwargs)
    self.encoder = encoder
    self.decoder = decoder
    self.total_loss_tracker = tf.keras.metrics.Mean(name="total_loss")

@property
def metrics(self):
    '''
    Model calls automatically reset_states() on any object listed here,
    at the beginning of each fit() epoch or at the beginning of a call
    to evaluate().
    In this way, calling result() would return per-epoch average and
    not an average since the start of training.
    '''
    return [self.total_loss_tracker]


def train_step(self, x_true):
    '''
    (1) Override train_step(self, x_true) to customize what fit() does.
    (2) We use GradientTape() in order to record operations for
    automatic differentiation.
    '''
    with tf.GradientTape() as tape:
        z_mean, z_log_var, z = self.encoder(x_true)
        x_pred = self.decoder(z)
        # reconstruction loss: mean squared error
        reconstruction_loss = tf.reduce_sum(
            tf.keras.losses.mean_squared_error(x_true, x_pred), axis=(1, 2)
```

```
            )
            # regularization term: KL divergence
            kl_loss = tf.reduce_sum(
                -0.5 * (1 + z_log_var - tf.math.square(z_mean) -
                tf.math.exp(z_log_var)), axis=1
            )
            total_loss = tf.reduce_mean(reconstruction_loss + kl_loss)
        # Get gradients of total loss with respect to the weights.
        grads = tape.gradient(total_loss, self.trainable_weights)
        # Update the weights of the model.
        self.optimizer.apply_gradients(zip(grads, self.trainable_weights))
        self.total_loss_tracker.update_state(total_loss)
        return {
            "loss": self.total_loss_tracker.result()
        }
```

```
# Customize Early Stopping

class EarlyStoppingAtMinLoss(tf.keras.callbacks.Callback):
'''
Stop training when the loss is at its min, i.e. the loss stops decreasing.

Arguments:
    patience: Number of epochs to wait after min has been hit.
              After this number of no improvement, training stops.
'''

def __init__(self, patience=30):
    super(EarlyStoppingAtMinLoss, self).__init__()
    self.patience = patience
    # best_weights to store the weights at which the minimum loss occurs.
    self.best_weights = None

def on_train_begin(self, logs=None):
    # The number of epoch it has waited when loss is no longer minimum.
    self.wait = 0
    # The epoch the training stops at.
    self.stopped_epoch = 0
    # Initialize the best as infinity.
    self.best = np.Inf

def on_epoch_end(self, epoch, logs=None):
    current = logs.get("loss")
```

```
        if np.less(current, self.best):
            self.best = current
            self.wait = 0
            # Record the best weights if current results is better (less).
            self.best_weights = self.model.get_weights()
        else:
            self.wait += 1
            if self.wait >= self.patience:
                self.stopped_epoch = epoch
                self.model.stop_training = True
                self.model.set_weights(self.best_weights)

    def on_train_end(self, logs=None):
        if self.stopped_epoch > 0:
            print("Epoch %05d: early stopping" % (self.stopped_epoch + 1))
```

```
    # Useful functions

    def TRE(truth_seq, est_seq):
        # truth_seq and est_seq are test_sizex12x12 arrays
        res = [0] * truth_seq.shape[0]
        for i in range(truth_seq.shape[0]):
            res[i] = np.linalg.norm(est_seq[i]-truth_seq[i]) /
                     np.linalg.norm(truth_seq[i])
        return res


    def SRE(truth_seq, est_seq):
        # truth_seq and est_seq are test_sizex12x12 arrays
        w, k, l = truth_seq.shape
        true_flows = defaultdict(list)
        est_flows = defaultdict(list)
        res = [0] * (k*l)
        for i in range(w):
            t = truth_seq[i].reshape(-1)
            e = est_seq[i].reshape(-1)
            for j in range(k*l):
                true_flows[j].append(t[j])
                est_flows[j].append(e[j])
        for i in range(k*l):
            if np.linalg.norm(true_flows[i]) == 0.0:
                if np.linalg.norm(est_flows[i]) == 0.0:
                    res[i] = 0.0
```

```
            else:
                new_true_flows = [x+1e-7 for x in true_flows]
                res[i] = np.linalg.norm(
                    list(
                        map(operator.sub, new_true_flows[i], est_flows[i])
                    )
                ) / np.linalg.norm(new_true_flows[i])
        else:
            res[i] = np.linalg.norm(
                list(
                    map(operator.sub, true_flows[i], est_flows[i])
                )
            ) / np.linalg.norm(true_flows[i])
    return res


def RMSE(truth_seq, est_seq):
    res = [0] * truth_seq.shape[0]
    for i in range(truth_seq.shape[0]):
        res[i] = np.sqrt(
            np.mean(
                tf.keras.losses.mean_squared_error(truth_seq[i], est_seq[i])
            )
        )
    return (np.mean(res), np.median(res), np.std(res), np.amax(res))


def NMAE(truth_seq, est_seq):
    # truth_seq and est_seq are test_sizex12x12 arrays
    res = [0] * truth_seq.shape[0]
    for i in range(truth_seq.shape[0]):
        res[i] = np.sum(np.abs(est_seq[i]-truth_seq[i])) /
                 np.sum(np.abs(truth_seq[i]))
    return (np.mean(res), np.median(res), np.std(res), np.amax(res))
```

```
# Plotting - Results
def results(X_test, X_pred):
    rmse = RMSE(X_test, X_pred)
    print('mean, median, std and maximum of RMSE are: {}, {}, {} and {}'
        .format(rmse[0], rmse[1], rmse[2], rmse[3]))

    nmae = NMAE(X_test, X_pred)
    print('mean, median, std and maximum of NMAE are: {}, {}, {} and {}'
```

```
            .format(nmae[0], nmae[1], nmae[2], nmae[3]))

        tres = TRE(X_test, X_pred)
        sres = SRE(X_test, X_pred)

        print('mean, median, std and maximum of TRE are: {}, {}, {} and {}'
            .format(np.mean(tres), np.median(tres), np.std(tres), np.amax(tres)))
        print('mean, median, std and maximum of SRE are: {}, {}, {} and {}'
            .format(np.mean(sres), np.median(sres), np.std(sres), np.amax(sres)))


        plt.plot(tres)
        plt.xlabel('Time', fontsize=14, fontweight='bold')
        plt.ylabel("Temporal Relative Error",  fontsize=14, fontweight='bold')
        plt.show()

        plt.plot(sres)
        plt.xlabel('OD Flow ID', fontsize=14, fontweight='bold')
        plt.ylabel("Spatial Relative Error",  fontsize=14, fontweight='bold')
        plt.show()

        x, f = sorted(tres), np.arange(len(tres)) / len(tres)
        plt.plot(x, f)
        plt.xlabel('Temporal Relative Error', fontsize=14, fontweight='bold')
        plt.ylabel("CDF",  fontsize=14, fontweight='bold')
        plt.show()

        x, f = sorted(sres), np.arange(len(sres)) / len(sres)
        plt.plot(x, f)
        plt.xlabel('Spatial Relative Error', fontsize=14, fontweight='bold')
        plt.ylabel("CDF",  fontsize=14, fontweight='bold')
        plt.show()
```

```
    # Training

    model = VAE(encoder, decoder)
    model.compile(optimizer=tf.keras.optimizers.Adam())
    history = model.fit(
        X_train, epochs=1000, batch_size=128, callbacks=[EarlyStoppingAtMinLoss()],
    )
    plt.plot(history.history['loss'])
    plt.ylabel('loss', fontsize=14, fontweight='bold')
    plt.xlabel('epoch', fontsize=14, fontweight='bold')
```

```
plt.show()
```

```
# Testing

A_var = tf.Variable(A, dtype=tf.float32)
optimizer = tf.keras.optimizers.Adam()
y_test = np.array(
    [
        np.dot(A, X_test[i,:,:,0].reshape(144,1)) for i in
        range(X_test.shape[0])
    ]
)
```

```
# Iterative optimization, starting from a "good" z

max_optimization_iterations = 5000
max_initial_point_iterations = 3000
X_pred = np.empty(shape=(test_size,12,12))


for i, y in enumerate(y_test):
    z = tf.Variable(tf.random.normal(shape=(1,latent_dim), dtype=tf.float32))
    x_pred_start = model.decoder(z)[0,:,:,0]
    y_pred_start = tf.tensordot(A_var, tf.reshape(x_pred_start, (144,1)), 1)
    loss = tf.reduce_mean(
                tf.keras.losses.mean_squared_error(y, y_pred_start)
            )
    for iteration in range(max_initial_point_iterations):
        z_new = tf.Variable(
                    tf.random.normal(
                                shape=(1,latent_dim), dtype=tf.float32
                    )
                )
        x_pred_new = model.decoder(z_new)[0,:,:,0]
        y_pred_new = tf.tensordot(A_var, tf.reshape(x_pred_new, (144,1)), 1)
        loss_new = tf.reduce_mean(
                        tf.keras.losses.mean_squared_error(y, y_pred_new)
                    )
        if loss_new < loss:
            z = z_new
            loss = loss_new
```

```
        # initialize values
        minimum_rmse = np.Inf
        x_pred_best = None

        for iteration in range(max_optimization_iterations):
            with tf.GradientTape() as tape:
                x_pred = model.decoder(z)[0,:,:,0]
                y_pred = tf.tensordot(A_var, tf.reshape(x_pred, (144,1)), 1)
                loss = tf.reduce_mean(
                            tf.keras.losses.mean_squared_error(y, y_pred)
                        )
                rmse = tf.math.sqrt(
                        tf.reduce_mean(
                         tf.keras.losses.mean_squared_error(
                          100*X_test[i,:,:,0], 100*x_pred)
                        )
                       )
                if tf.math.less(rmse, minimum_rmse):
                    x_pred_best = x_pred
                    minimum_rmse = rmse
            # get gradient of loss with respect to z
            grads = tape.gradient(loss, z)
            # update the value of z
            optimizer.apply_gradients(zip([grads], [z]))

        print()
        print('minimum RMSE between X_test[{}] and x_pred_best: {} Mbps'.
                format(i, minimum_rmse))
        X_pred[i] = np.array(100*x_pred_best)

    # get results
    results(100*X_test[:,:,:,0], X_pred)
```

```
    # Iterative optimization, starting from a "good" z, with regularization term

    max_optimization_iterations = 5000
    max_initial_point_iterations = 500
    X_pred = np.empty(shape=(test_size,12,12))


    for i, y in enumerate(y_test):
        z = tf.Variable(tf.random.normal(shape=(1,latent_dim), dtype=tf.float32))
        x_pred_start = model.decoder(z)[0,:,:,0]
```

```
        y_pred_start = tf.tensordot(A_var, tf.reshape(x_pred_start, (144,1)), 1)
        loss = tf.reduce_mean(
                    tf.keras.losses.mean_squared_error(y, y_pred_start)
                ) + 0.1*tf.norm(z)**2
        for iteration in range(max_initial_point_iterations):
            z_new = tf.Variable(
                        tf.random.normal(
                            shape=(1,latent_dim), dtype=tf.float32
                        )
                    )
            x_pred_new = model.decoder(z_new)[0,:,:,0]
            y_pred_new = tf.tensordot(A_var, tf.reshape(x_pred_new, (144,1)), 1)
            loss_new = tf.reduce_mean(
                            tf.keras.losses.mean_squared_error(y, y_pred_new)
                        ) + 0.1*tf.norm(z)**2
            if loss_new < loss:
                z = z_new
                loss = loss_new

        # initialize values
        minimum_rmse = np.Inf
        x_pred_best = None

        for iteration in range(max_optimization_iterations):
            with tf.GradientTape() as tape:
                x_pred = model.decoder(z)[0,:,:,0]
                y_pred = tf.tensordot(A_var, tf.reshape(x_pred, (144,1)), 1)
                loss = tf.reduce_mean(
                            tf.keras.losses.mean_squared_error(y, y_pred)
                        ) + 0.1*tf.norm(z)**2
                rmse = tf.math.sqrt(
                        tf.reduce_mean(
                         tf.keras.losses.mean_squared_error(
                          100*X_test[i,:,:,0], 100*x_pred
                         )
                        )
                       )
                if tf.math.less(rmse, minimum_rmse):
                    x_pred_best = x_pred
                    minimum_rmse = rmse
            # get gradient of loss with respect to z
            grads = tape.gradient(loss, z)
            # update the value of z
            optimizer.apply_gradients(zip([grads], [z]))
```

```
        print()
        print('minimum RMSE between X_test[{}] and x_pred_best: {} Mbps'
            .format(i, minimum_rmse))
        X_pred[i] = np.array(100*x_pred_best)

    # get results
    results(100*X_test[:,:,:,0], X_pred)
```

```
    # Concurrent optimization, starting from a "good" z, with regularization term

    round_optimization_iterations = 3000
    round_training_epochs = 20
    X_pred = np.empty(shape=(test_size,12,12))
    max_initial_point_iterations = 500



    for i, y in enumerate(y_test):

        # new model for each test
        optimizer = tf.keras.optimizers.Adam()

        # encoder
        encoder_inputs = tf.keras.Input(shape=(12, 12, 1))
        x = tf.keras.layers.Dropout(0.25)(encoder_inputs)
        x = tf.keras.layers.Conv2D(
                32, 3, activation="relu", strides=2, padding="same"
            )(x)
        x = tf.keras.layers.Conv2D(
                64, 3, activation="relu", strides=2, padding="same"
            )(x)
        x = tf.keras.layers.Conv2D(
                128, 3, activation="relu", strides=1, padding="same"
            )(x)
        x = tf.keras.layers.Flatten()(x)
        x = tf.keras.layers.Dense(64, activation="relu")(x)
        z_mean = tf.keras.layers.Dense(latent_dim, name="z_mean")(x)
        z_log_var = tf.keras.layers.Dense(latent_dim, name="z_log_var")(x)
        z = Sampling()([z_mean, z_log_var])
        encoder = tf.keras.Model(
                    encoder_inputs, [z_mean, z_log_var, z], name="encoder"
                  )
```

```
#decoder
latent_inputs = tf.keras.Input(shape=(latent_dim,))
x = tf.keras.layers.Dense(64, activation="relu")(latent_inputs)
x = tf.keras.layers.Dense(3 * 3 * 64, activation="relu")(x)
x = tf.keras.layers.Reshape((3, 3, 64))(x)
x = tf.keras.layers.Conv2DTranspose(
        128, 3, activation="relu", strides=1, padding="same"
    )(x)
x = tf.keras.layers.Conv2DTranspose(
        64,  3, activation="relu", strides=2, padding="same"
    )(x)
x = tf.keras.layers.Conv2DTranspose(
        32,  3, activation="relu", strides=2, padding="same"
    )(x)
decoder_outputs = tf.keras.layers.Conv2DTranspose(
                    1, 3, activation="relu", padding="same"
                  )(x)
decoder = tf.keras.Model(latent_inputs, decoder_outputs, name="decoder")

model = VAE(encoder, decoder)
model.compile(optimizer=tf.keras.optimizers.Adam())
earlystoppingmonitor = EarlyStoppingAtMinLoss()

# find good starting point
z = tf.Variable(tf.random.normal(shape=(1,latent_dim), dtype=tf.float32))
x_pred_start = model.decoder(z)[0,:,:,0]
y_pred_start = tf.tensordot(A_var, tf.reshape(x_pred_start, (144,1)), 1)
loss = tf.reduce_mean(
            tf.keras.losses.mean_squared_error(y, y_pred_start)
        ) + 0.1*tf.norm(z)**2
for iteration in range(max_initial_point_iterations):
    z_new = tf.Variable(
                tf.random.normal(shape=(1,latent_dim), dtype=tf.float32)
            )
    x_pred_new = model.decoder(z_new)[0,:,:,0]
    y_pred_new = tf.tensordot(A_var, tf.reshape(x_pred_new, (144,1)), 1)
    loss_new = tf.reduce_mean(
                    tf.keras.losses.mean_squared_error(y, y_pred_new)
                ) + 0.1*tf.norm(z)**2
    if loss_new < loss:
        z = z_new
        loss = loss_new
```

```
        # initialize values
        minimum_rmse = np.Inf
        x_pred_best = None

        loops_run = 0
        while loops_run < 7:
            # continue training from last loop iteration
            model.fit(
                X_train, epochs=round_training_epochs, batch_size=128,
                callbacks=[earlystoppingmonitor],
                )
            for iteration in range(round_optimization_iterations):
                with tf.GradientTape() as tape:
                    # use optimal z previous optimization
                    x_pred = model.decoder(z)[0,:,:,0]
                    y_pred = tf.tensordot(A_var, tf.reshape(x_pred, (144,1)), 1)
                    loss = tf.reduce_mean(
                                tf.keras.losses.mean_squared_error(y, y_pred)
                           ) + 0.1*tf.norm(z)**2
                    rmse = tf.math.sqrt(
                            tf.reduce_mean(
                             tf.keras.losses.mean_squared_error(
                              100*X_test[i,:,:,0], 100*x_pred
                              )
                             )
                            )
                    if tf.math.less(rmse, minimum_rmse):
                        x_pred_best = x_pred
                        minimum_rmse = rmse
                    # get gradient of loss with respect to z
                    grads = tape.gradient(loss, z)
                    # update the value of z
                    optimizer.apply_gradients(zip([grads], [z]))
                loops_run += 1

            print()
            print('minimum RMSE between X_test[{}] and x_pred_best: {} Mbps'
                .format(i, minimum_rmse))

            X_pred[i] = np.array(100*x_pred_best)

    # get results
    results(100*X_test[:,:,:,0], X_pred)
```

List of Figures

List of Tables

References

[1] W. Silva, «Make Flows Great Again: A Hybrid Resilience Mechanism for OpenFlow Networks», *Information*, vol. 9, p. 146, Jun. 2018. DOI: 10.3390/info9060146.

[2] B.-K. Kim, *Internationalizing the Internet: The Co-evolution of Influence and Technology*. Edward Elgar Publishing, Jan. 1, 2005, 320 pp., ISBN: 978-1-84542-675-0.

[3] *Cisco Annual Internet Report - Cisco Annual Internet Report (2018–2023) White Paper*, Cisco. [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html (visited on 03/29/2021).

[4] P. Tune and M. Roughan, «Internet traffic matrices: A primer», in *Recent Advances in Networking*, H. Haddadi and O. Bonaventure, Eds., vol. 1, ACM SIGCOMM eBook, 2013, ch. 3, pp. 108–163.

[5] Y. Vardi, «Network Tomography: Estimating Source-Destination Traffic Intensities from Link Data», *Journal of the American Statistical Association*, vol. 91, no. 433, pp. 365–377, 1996, ISSN: 0162-1459. DOI: 10.2307/2291416.

[6] A. Coates, A. O. Hero III, R. Nowak, and Bin Yu, «Internet tomography», *IEEE Signal Processing Magazine*, vol. 19, no. 3, pp. 47–65, May 2002, ISSN: 1558-0792. DOI: 10.1109/79.998081.

[7] G. Kakkavas, D. Gkatzioura, V. Karyotis, and S. Papavassiliou, «A Review of Advanced Algebraic Approaches Enabling Network Tomography for Future Network Infrastructures», *Future Internet*, vol. 12, no. 2, 2020, ISSN: 1999-5903. DOI: 10.3390/fi12020020. [Online]. Available: https://www.mdpi.com/1999-5903/12/2/20.

[8] Y. LeCun, Y. Bengio, and G. Hinton, «Deep learning», *Nature*, vol. 521, pp. 436–444, May 2015, ISSN: 1476-4687. DOI: 10.1038/nature14539.

[9] M. Dodge and R. Kitchin, *Atlas of cyberspace*, 1. publ. Harlow: Addison-Wesley, 2001, 268 pp., ISBN: 978-0-201-74575-7.

[10] *ping(8) - Linux man page*. [Online]. Available: https://linux.die.net/man/8/ping (visited on 03/31/2021).

[11] *traceroute(8) - Linux man page*. [Online]. Available: https://linux.die.net/man/8/traceroute (visited on 03/31/2021).

[12] A. Tarantola, *Inverse Problem Theory and Methods for Model Parameter Estimation*. Society for Industrial and Applied Mathematics, Jan. 2005, ISBN: 978-0-89871-572-9. DOI: 10.1137/1.9780898717921.

[13] R. Mammone, «Inverse problems and signal processing», in *The Digital Signal Processing Handbook*, Boca Raton, FL: CRC Press, 1998.

[14] G. Ongie, A. Jalal, C. A. Metzler, R. G. Baraniuk, A. G. Dimakis, and R. Willett, *Deep Learning Techniques for Inverse Problems in Imaging*, 2020. arXiv: 2005.06001 [eess.IV].

[15] M. Bertero and M. Piana, «Inverse problems in biomedical imaging: modeling and methods of solution», in *Complex Systems in Biomedicine*, A. Quarteroni, L. Formaggia, and A. Veneziani, Eds., Springer Milan, 2006, pp. 1–33, ISBN: 978-88-470-0396-5. DOI: 10.1007/88-470-0396-2_1.

[16] G. Kakkavas, M. Kalntis, V. Karyotis, and S. Papavassiliou, «Future Network Traffic Matrix Synthesis and Estimation Based on Deep Generative Models», accepted for presentation in the 30th International Conference on Computer Communication and Networks (ICCCN), 2021.

[17] Y. Zhang. (2004). «Abilene network topology data and traffic traces», [Online]. Available: https://www.cs.utexas.edu/~yzhang/research/AbileneTM/ (visited on 04/02/2021).

[18] D. P. Kingma and M. Welling, *Auto-Encoding Variational Bayes*, 2014. arXiv: 1312.6114 [stat.ML].

[19] D. H. Ballard, «Modular Learning in Neural Networks», in *Proceedings of the Sixth National Conference on Artificial Intelligence - Volume 1*, ser. AAAI'87, Seattle, Washington: AAAI Press, 1987, pp. 279–284, ISBN: 0934613427.

[20] D. Bank, N. Koenigstein, and R. Giryes, *Autoencoders*, 2021. arXiv: 2003.05991 [cs.LG].

[21] A. Gelman, J. B. Carlin, H. S. Stern, D. B. Dunson, A. Vehtari, and D. B. Rubin, *Bayesian Data Analysis*, 3rd edition. Boca Raton: Chapman and Hall/CRC, Nov. 1, 2013, 675 pp., ISBN: 978-1-4398-4095-5.

[22] S. Kullback and R. A. Leibler, «On Information and Sufficiency», *The Annals of Mathematical Statistics*, vol. 22, no. 1, pp. 79–86, Mar. 1951, ISSN: 0003-4851, 2168-8990. DOI: 10.1214/aoms/1177729694.

[23] J. Case, M. Fedor, M. Schoffstall, and D. JR, «Simple network management protocol (SNMP)», *RFC 1157*, Apr. 1989.

[24] G. Song, Z. Fan, and J. Lafferty, *Surfing: Iterative optimization over incrementally trained deep networks*, 2019. arXiv: 1907.08653 [stat.ML].

[25] Y. Zhang, Z. Ge, A. Greenberg, and M. Roughan, «Network Anomography.», Jan. 2005, pp. 317–330. DOI: 10.1145/1330107.1330146.

[26] J. Zhang, K. Xi, M. Luo, and H. J. Chao, «Load balancing for multiple traffic matrices using SDN hybrid routing», in *2014 IEEE 15th International Conference on High Performance Switching and Routing (HPSR)*, 2014, pp. 44–49. DOI: 10.1109/HPSR.2014.6900880.

[27] A. Lakhina, K. Papagiannaki, M. Crovella, C. Diot, E. D. Kolaczyk, and N. Taft, *Structural Analysis of Network Traffic Flows*, 2004.

[28] R. Heckel and P. Hand, *Deep Decoder: Concise Image Representations from Untrained Non-convolutional Networks*, 2019. arXiv: 1810.03982 [cs.CV].

[29] V. Lempitsky, A. Vedaldi, and D. Ulyanov, «Deep Image Prior», in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 9446–9454. DOI: 10.1109/CVPR.2018.00984.

[30] J. Adler and O. Öktem, *Deep Bayesian Inversion*, 2018. arXiv: 1811.05910 [stat.ML].

[31] M. Mirza and S. Osindero, *Conditional Generative Adversarial Nets*, 2014. arXiv: 1411.1784 [cs.LG].

[32] Y. Li, *Deep Reinforcement Learning*, 2018. arXiv: 1810.06339 [cs.LG].

[33] G. Kakkavas, A. Stamou, V. Karyotis, and S. Papavassiliou, «Network Tomography for Efficient Monitoring in SDN-Enabled 5G Networks and Beyond: Challenges and Opportunities», *IEEE Communications Magazine*, vol. 59, no. 3, pp. 70–76, 2021. DOI: 10.1109/MCOM.001.2000458.

[34] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[35] A. Bondy and U. S. R. Murty, *Graph Theory*, 2008th edition. New York: Springer, Oct. 19, 2010, 675 pp., ISBN: 978-1-84996-690-0.

[36] M. Joshi and T. Aldhayni, «A Review of Network Traffic Analysis and Prediction Techniques», Jul. 2015.

[37] A. Alakeel, «A Guide to Dynamic Load Balancing in Distributed Computer Systems», *International Journal of Computer Science and Network Security (IJCSNS)*, vol. 10, Nov. 2009.

[38] E. L. Lehmann and G. Casella, *Theory of Point Estimation*, 2nd ed., ser. Springer Texts in Statistics. New York: Springer-Verlag, 1998, ISBN: 978-0-387-98502-2. DOI: 10.1007/b98854.

[39] K. Papagiannaki, N. Taft, and A. Lakhina, «A Distributed Approach to Measure IP Traffic Matrices», Jan. 2004, pp. 161–174. DOI: 10.1145/1028788.1028808.

[40] N. G. Duffield, J. Horowitz, F. Lo Presti, and D. Towsley, «Explicit Loss Inference in Multicast Tomography», *IEEE Transactions on Information Theory*, vol. 52, no. 8, pp. 3852–3855, 2006. DOI: 10.1109/TIT.2006.878228.

[41] F. Lo Presti, N. G. Duffield, J. Horowitz, and D. Towsley, «Multicast-based inference of network-internal delay distributions», *IEEE/ACM Transactions on Networking*, vol. 10, no. 6, pp. 761–775, 2002. DOI: 10.1109/TNET.2002.805026.

[42] Y. Lin, T. He, S. Wang, K. Chan, and S. Pasteris, «Multicast-Based Weight Inference in General Network Topologies», in *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, 2019, pp. 1–6. DOI: 10.1109/ICC.2019.8761099.

[43] N. G. Duffield and F. Lo Presti, «Multicast inference of packet delay variance at interior network links», in *Proceedings IEEE INFOCOM 2000. Conference on Computer Communications. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (Cat. No.00CH37064)*, vol. 3, 2000, 1351–1360 vol.3. DOI: 10.1109/INFCOM.2000.832532.

[44] R. Caceres, N. G. Duffield, J. Horowitz, and D. F. Towsley, «Multicast-based inference of network-internal loss characteristics», *IEEE Transactions on Information Theory*, vol. 45, no. 7, pp. 2462–2480, 1999. DOI: 10.1109/18.796384.

[45] M. Coates and R. Nowak, «Network Loss Inference Using Unicast End-to-End Measurement», Jul. 2000.

[46] M. J. Coates and R. D. Nowak, «Network tomography for internal delay estimation», in *2001 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings (Cat. No.01CH37221)*, vol. 6, 2001, 3409–3412 vol.6. DOI: 10.1109/ICASSP.2001.940573.

[47] Y. Gu, G. Jiang, V. Singh, and Y. Zhang, «Optimal Probing for Unicast Network Delay Tomography», in *2010 Proceedings IEEE INFOCOM*, 2010, pp. 1–9. DOI: 10.1109/INFCOM.2010.5461920.

[48] M. Rahali, J. Sanner, and G. Rubino, «Unicast Inference of Additive Metrics in General Network Topologies», in *2019 IEEE 27th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2019, pp. 107–115. DOI: 10.1109/MASCOTS.2019.00021.

[49] B. K. Dey, D. Manjunath, and S. Chakraborty, «Estimating network link characteristics using packet-pair dispersion: A discrete-time queueing theoretic analysis», *Computer Networks*, vol. 55, no. 5, pp. 1052–1068, 2011, ISSN: 1389-1286. DOI: https://doi.org/10.1016/j.comnet.2010.05.008.

[50] G. Kakkavas, V. Karyotis, and S. Papavassiliou, «A Distance-based Agglomerative Clustering Algorithm for Multicast Network Tomography», in *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*, 2020, pp. 1–7. DOI: 10.1109/ICC40277.2020.9149412.

[51] P. Goransson, C. Black, and T. Culver, *Software Defined Networks: A Comprehensive Approach*. Morgan Kaufmann, Oct. 25, 2016, 438 pp., ISBN: 978-0-12-804579-4.

[52] B. Han, V. Gopalakrishnan, L. Ji, and S. Lee, «Network function virtualization: Challenges and opportunities for innovations», *IEEE Communications Magazine*, vol. 53, no. 2, pp. 90–97, 2015. DOI: 10.1109/MCOM.2015.7045396.

[53] R. M. Castro, M. J. Coates, and R. D. Nowak, «Likelihood based hierarchical clustering», *IEEE Transactions on Signal Processing*, vol. 52, no. 8, pp. 2308–2321, 2004. DOI: 10.1109/TSP.2004.831124.

[54] M. Coates, R. Castro, R. Nowak, M. Gadhiok, R. King, and Y. Tsang, «Maximum Likelihood Network Topology Identification from Edge-Based Unicast Measurements», *SIGMETRICS Perform. Eval. Rev.*, vol. 30, no. 1, pp. 11–20, Jun. 2002, ISSN: 0163-5999. DOI: 10.1145/511399.511337.

[55] M. Coates and R. Nowak, «Networks for networks: Internet analysis using graphical statistical models», in *Neural Networks for Signal Processing X. Proceedings of the 2000 IEEE Signal Processing Society Workshop (Cat. No.00TH8501)*, vol. 2, 2000, 755–764 vol.2. DOI: 10.1109/NNSP.2000.890155.

[56] J. Ni, H. Xie, S. Tatikonda, and Y. R. Yang, «Efficient and Dynamic Routing Topology Inference From End-to-End Measurements», *IEEE/ACM Transactions on Networking*, vol. 18, no. 1, pp. 123–135, 2010. DOI: 10.1109/TNET.2009.2022538.

[57] B. Eriksson, G. Dasarathy, P. Barford, and R. Nowak, «Efficient Network Tomography for Internet Topology Discovery», *IEEE/ACM Transactions on Networking*, vol. 20, no. 3, pp. 931–943, 2012. DOI: 10.1109/TNET.2011.2175747.

[58] Y. Bengio, A. Courville, and P. Vincent, *Representation Learning: A Review and New Perspectives*, 2014. arXiv: 1206.5538 [cs.LG].

[59] T. M. Mitchell, *Machine Learning*, 1st edition. New York: McGraw-Hill Education, Mar. 1, 1997, 432 pp., ISBN: 978-0-07-042807-2.

[60] G. E. Dahl, D. Yu, L. Deng, and A. Acero, «Context-Dependent Pre-Trained Deep Neural Networks for Large-Vocabulary Speech Recognition», *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 20, no. 1, pp. 30–42, 2012. DOI: 10.1109/TASL.2011.2134090.

[61] Y. Bengio, «Neural net language models», *Scholarpedia*, vol. 3, p. 3881, Jan. 2008. DOI: 10.4249/scholarpedia.3881.

[62] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd edition. Upper Saddle River: Pearson, Dec. 1, 2009, 1152 pp., ISBN: 978-0-13-604259-4.

[63] W. S. McCulloch and W. Pitts, «A logical calculus of the ideas immanent in nervous activity», *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[64] S. Ruder, *An overview of gradient descent optimization algorithms*, 2017. arXiv: 1609.04747 [cs.LG].

[65] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, «Gradient-based learning applied to document recognition», *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998. DOI: 10.1109/5.726791.

[66] W. Xing and D. Du, «Dropout Prediction in MOOCs: Using Deep Learning for Personalized Intervention», *Journal of Educational Computing Research*, vol. 57, p. 073 563 311 875 701, Mar. 2018. DOI: 10.1177/0735633118757015.

[67] H. GM, M. K. Gourisaria, M. Pandey, and S. S. Rautaray, «A comprehensive survey and analysis of generative models in machine learning», *Computer Science Review*, vol. 38, p. 100 285, 2020, ISSN: 1574-0137. DOI: https://doi.org/10.1016/j.cosrev.2020.100285.

[68] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, «Generative Adversarial Nets», in *Advances in Neural Information Processing Systems*, Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Q. Weinberger, Eds., vol. 27, Curran Associates, Inc., 2014.

[69] L. Van Der Maaten, E. Postma, and J. Van den Herik, «Dimensionality reduction: a comparative review», *J Mach Learn Res*, vol. 10, pp. 66–71, 2009.

[70] I. T. Jolliffe and J. Cadima, «Principal component analysis: a review and recent developments», *Philosophical transactions. Series A, Mathematical, physical, and engineering sciences*, vol. 374, no. 2065, Apr. 13, 2016. DOI: 10.1098/rsta.2015.0202.

[71] K. Cho, *Boltzmann Machines and Denoising Autoencoders for Image Denoising*, 2013. arXiv: 1301.3468 [stat.ML].

[72] M. Sakurada and T. Yairi, «Anomaly Detection Using Autoencoders with Nonlinear Dimensionality Reduction», in *Proceedings of the MLSDA 2014 2nd Workshop on Machine Learning for Sensory Data Analysis*, ser. MLSDA'14, Gold Coast, Australia QLD, Australia: Association for Computing Machinery, 2014, pp. 4–11, ISBN: 9781450331593. DOI: 10.1145/2689746.2689747.

[73] R. B. Ash and C. A. Doléans-Dade, *Probability and Measure Theory*, 2nd edition. San Diego: Academic Press, Dec. 20, 1999, 528 pp., ISBN: 978-0-12-065202-0.

[74] K. P. Murphy, *Machine Learning: A Probabilistic Perspective*, Illustrated edition. Cambridge, MA: The MIT Press, Aug. 24, 2012, 1104 pp., ISBN: 978-0-262-01802-9.

[75] T. O. Kvalseth, «Generalized divergence and Gibbs' inequality», in *1997 IEEE International Conference on Systems, Man, and Cybernetics. Computational Cybernetics and Simulation*, vol. 2, 1997, 1797–1801 vol.2. DOI: 10.1109/ICSMC.1997.638293.

[76] K. P. Murphy, *Probabilistic Machine Learning: An introduction*. MIT Press, 2021. [Online]. Available: probml.ai.

[77] J. Paisley, D. Blei, and M. Jordan, *Variational Bayesian Inference with Stochastic Search*, 2012. arXiv: 1206.6430 [cs.LG].

[78] R. E. Caflisch, «Monte Carlo and quasi-Monte Carlo methods», *Acta Numerica*, vol. 7, pp. 1–49, 1998. DOI: 10.1017/S0962492900002804.

[79] J. Kurose and K. Ross, *Computer Networking: A Top-Down Approach*, 7th edition. Boston: Pearson, Apr. 26, 2016, 864 pp., ISBN: 978-0-13-359414-0.

[80] J. Cao, R. Davis, S. Vander, and B. Yu, «Time-Varying Network Tomography: Router Link Data», *Journal of the American Statistical Association*, vol. 95, Apr. 2000. DOI: 10.2307/2669743.

[81] A. Medina, N. Taft, K. Salamatian, S. Bhattacharyya, and C. Diot, «Traffic Matrix Estimation: Existing Techniques and New Directions», *SIGCOMM Comput. Commun. Rev.*, vol. 32, no. 4, pp. 161–174, Aug. 2002, ISSN: 0146-4833. DOI: 10.1145/964725.633041.

[82] Y. Zhang, M. Roughan, N. Duffield, and A. Greenberg, «Fast Accurate Computation of Large-Scale IP Traffic Matrices from Link Loads», in *Proceedings of the 2003 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, ser. SIGMETRICS '03, San Diego, CA, USA: Association for Computing Machinery, 2003, pp. 206–217, ISBN: 1581136641. DOI: 10.1145/781027.781053.

[83] A. Soule, A. Nucci, R. Cruz, E. Leonardi, and N. Taft, «How to Identify and Estimate the Largest Traffic Matrix Elements in a Dynamic Environment», ser. SIGMETRICS '04/Performance '04, New York, NY, USA: Association for Computing Machinery, 2004, pp. 73–84, ISBN: 1581138733. DOI: 10.1145/1005686.1005698.

[84] A. Soule, A. Lakhina, N. Taft, K. Papagiannaki, K. Salamatian, A. Nucci, M. Crovella, and C. Diot, «Traffic Matrices: Balancing Measurements, Inference and Modeling», ser. SIGMETRICS '05, Banff, Alberta, Canada: Association for Computing Machinery, 2005, pp. 362–373, ISBN: 1595930221. DOI: 10.1145/1064212.1064259.

[85] Y. Zhang, M. Roughan, W. Willinger, and L. Qiu, «Spatio-Temporal Compressive Sensing and Internet Traffic Matrices», in *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication*, ser. SIGCOMM '09, Barcelona, Spain: Association for Computing Machinery, 2009, pp. 267–278, ISBN: 9781605585949. DOI: 10.1145/1592568.1592600.

[86] K. Xie, L. Wang, X. Wang, G. Xie, J. Wen, and G. Zhang, «Accurate recovery of Internet traffic data: A tensor completion approach», in *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*, 2016, pp. 1–9. DOI: 10.1109/INFOCOM.2016.7524463.

[87] M. Polverini, A. Iacovazzi, A. Cianfrani, A. Baiocchi, and M. Listanti, «Traffic matrix estimation enhanced by SDNs nodes in real network topology», *Proceedings - IEEE INFOCOM*, vol. 2015, pp. 300–305, Aug. 2015. DOI: 10.1109/INFCOMW.2015.7179401.

[88] Z. Hu, Y. Qiao, and J. Luo, «ATME: Accurate Traffic Matrix Estimation in both Public and Private Datacenter Networks», *IEEE Transactions on Cloud Computing*, vol. 6, pp. 60–73, Jan. 2018. DOI: 10.1109/TCC.2015.2481383.

[89] D. Jiang, X. Wang, L. Guo, H. Ni, and Z. Chen, «Accurate estimation of large-scale IP traffic matrix», *AEU - International Journal of Electronics and Communications*, vol. 65, no. 1, pp. 75–86, 2011, ISSN: 1434-8411. DOI: https://doi.org/10.1016/j.aeue.2010.02.008.

[90] D. Jiang, Z. Zhao, Z. Xu, C. Yao, and H. Xu, «How to reconstruct end-to-end traffic based on time-frequency analysis and artificial neural network», *AEU - International Journal of Electronics and Communications*, vol. 68, no. 10, pp. 915–925, 2014, ISSN: 1434-8411. DOI: https://doi.org/10.1016/j.aeue.2014.04.011.

[91] H. Zhou, L. Tan, Q. Zeng, and C. Wu, «Traffic matrix estimation: A neural network approach with extended input and expectation maximization iteration», *Journal of Network and Computer Applications*, vol. 60, pp. 220–232, 2016, ISSN: 1084-8045. DOI: https://doi.org/10.1016/j.jnca.2015.11.013.

[92] S. Hussain, A. Sultan, S. Qazi, and M. Ameer, «Intelligent Traffic Matrix Estimation Using LevenBerg-Marquardt Artificial Neural Network of Large Scale IP Network», Dec. 2019, pp. 1–5. DOI: 10.1109/MACS48846.2019.9024765.

[93] L. Nie, D. Jiang, L. Guo, and S. Yu, «Traffic matrix prediction and estimation based on deep learning in large-scale IP backbone networks», *Journal of Network and Computer Applications*, vol. 76, pp. 16–22, 2016, ISSN: 1084-8045. DOI: https://doi.org/10.1016/j.jnca.2016.10.006.

[94] J. Zhao, H. Qu, J. Zhao, and D. Jiang, «Spatiotemporal traffic matrix prediction: A deep learning approach with wavelet multiscale analysis», *Transactions on Emerging Telecommunications Technologies*, vol. 30, Jun. 2019. DOI: 10.1002/ett.3640.

[95] S. Xu, M. Kodialam, T. V. Lakshman, and S. Panwar, *Learning Based Methods for Traffic Matrix Estimation from Link Measurements*, 2020. arXiv: 2008.00905 [cs.NI].

[96] D. P. Kingma and J. Ba, *Adam: A Method for Stochastic Optimization*, 2017. arXiv: 1412.6980 [cs.LG].

[97] V. Nair and G. E. Hinton, «Rectified Linear Units Improve Restricted Boltzmann Machines», in *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ser. ICML'10, Haifa, Israel: Omnipress, 2010, pp. 807–814, ISBN: 9781605589077.

[98] F. Chollet *et al.* (2015). «Keras», [Online]. Available: https://github.com/fchollet/keras.

[99] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mane, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viegas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, *TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems*, 2016. arXiv: 1603.04467 [cs.DC].

[100] T. Kluyver, B. Ragan-Kelley, F. Pérez, B. Granger, M. Bussonnier, J. Frederic, K. Kelley, J. Hamrick, J. Grout, S. Corlay, P. Ivanov, D. Avila, S. Abdalla, and C. Willing, «Jupyter Notebooks – a publishing format for reproducible computational workflows», in *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, F. Loizides and B. Schmidt, Eds., IOS Press, 2016, pp. 87–90.