Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Τομέας Ηλεκτρικής Ισχύος
Εργαστήριο Συστημάτων Ηλεκτρικής Ενέργειας

Electric Load Modeling Using Machine Learning

Διπλωματική Εργασία

Ορέστης Πετρόπουλος

Επιβλέπων: Δρ. Νικόλαος Χατζηαργυρίου
Ομότιμος Καθηγητής

Αθήνα, Ιούλιος 2022

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Τομέας Ηλεκτρικής Ισχύος
Εργαστήριο Συστημάτων Ηλεκτρικής Ενέργειας

Electric Load Modeling Using Machine Learning

Διπλωματική Εργασία

Ορέστης Πετρόπουλος

Επιβλέπων: Δρ. Νικόλαος Χατζηαργυρίου
Ομότιμος Καθηγητής

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή, 12 Ιουλίου 2022.

(Υπογραφή)	*(Υπογραφή)*	*(Υπογραφή)*
....	....	....
Νικόλαος Χατζηαργυρίου Καθηγητής ΕΜΠ	Σταύρος Παπαθανασίου Καθηγητής ΕΜΠ	Παύλος Γεωργιλάκης Καθηγητής ΕΜΠ

Αθήνα, Ιούλιος 2022

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

Copyright © Ορέστης Πετρόπουλος, 2022
Με επιφύλαξη παντός δικαιώματος. All rights reserved.

Απαγορεύεται η αντιγραφή, αποθήκευση και διανομή της παρούσας εργασίας, εξ ολοκλήρου ή τμήματος αυτής, για εμπορικό σκοπό. Επιτρέπεται η ανατύπωση, αποθήκευση και διανομή για σκοπό μη κερδοσκοπικό, εκπαιδευτικής ή ερευνητικής φύσης, υπό την προϋπόθεση να αναφέρεται η πηγή προέλευσης και να διατηρείται το παρόν μήνυμα. Ερωτήματα που αφορούν τη χρήση της εργασίας για κερδοσκοπικό σκοπό πρέπει να απευθύνονται προς τον συγγραφέα. Οι απόψεις και τα συμπεράσματα που περιέχονται σε αυτό το έγγραφο εκφράζουν τον συγγραφέα και δεν πρέπει να ερμηνευθεί ότι αντιπροσωπεύουν τις επίσημες θέσεις του Εθνικού Μετσόβιου Πολυτεχνείου.

Περίληψη

Τα σύγχρονα συστήματα διανομής ηλεκτρικής ενέργειας αντιμετωπίζουν αυξανόμενες προκλήσεις, τόσο λόγω της μετάβασης σε έξυπνα δίκτυα όσο και λόγω της ενσωμάτωσης νέων τύπων φορτίων, όπως τα ηλεκτρικά οχήματα και τα συστήματα αποθήκευσης ενέργειας. Οι παράγοντες αυτοί, καθιστούν αναγκαία τη μελέτη των δικτύων διανομής υπό πολλές διαφορετικές συνθήκες και με τη χρήση νέων εργαλείων, όπως η επιστήμη των δεδομένων και τα νευρωνικά δίκτυα.

Σε αυτή τη διπλωματική εργασία, εξετάζουμε την υλοποίηση ενός νευρωνικού δικτύου και συγκεκριμένα ενός Μεταβλητού Αυτοκωδικοποιητή, για τους σκοπούς που αναφέρθηκαν παραπάνω. Το μοντέλο είναι ικανό να εκπαιδευτεί πάνω σε μικρό αριθμό δεδομένων, να μάθει τα κύρια χαρακτηριστικά τους και να τα κωδικοποιήσει σε ένα χώρο απεικόνισης, με τρόπο χρήσιμο και συνάδοντα με την ανθρώπινη αντίληψη. Κατόπιν, το εκπαιδευμένο πλέον μοντέλο χρησιμοποιείται για την κωδικοποιήση και αποκωδικοποίηση υπάρχοντων δεδομένων, κρατώντας μόνο τα κύρια χαρακτηριστικά τους , δυσχεραίνοντας έτσι την ταυτοποίηση συγκεκριμένων χρηστών του δικτύου μέσω των χρονοσειρών φορτίου και καθιστώντας πιο ασφαλή το διαμοιρασμό των δεδομένων. Επίσης, χρησιμοποιώντας τη χαρτογράφηση του χώρου απεικόνισης και τον εκπαιδευμένο αποκωδικοποιητή, μπορούμε να παράξουμε κατά το δοκούν νέες χρονοσειρές φορτίων, κρατώντας τα βασικά χαρακτηριστικά και αλλάζοντας κάποια άλλα, όπως η μέγιστη τιμή και η κατανομή των διαφόρων ειδών χρονοσειρών μέσα στο χρόνο.

Μετά την εκπαίδευση, το μοντέλο είναι ικανό να ανακατασκευάσει επαρκώς τα υπάρχοντα δεδομένα, κρατώντας σχεδόν ανέπαφα τα κύρια χαρακτηριστικά τους όπως η μέση τιμή, η μέγιστη τιμή και η ώρα της μέρας που αυτή παρατηρείται. Επίσης είναι ικανό να αποκωδικοποιήσει οποιοδήποτε δείγμα 2 αριθμών σε μια πλήρη ημέρα, αποτελούμενη από 576 μετρήσεις (96 μετρήσεις ενεργού και αέργου ισχύος για 3 αλληλοσυσχετιζόμενα φορτία). Η μέθοδος που χρησιμοποιήθηκε για την παραγωγή νέων χρονοσειρών, είναι η τροποποίηση της κωδικοποιημένης μορφής των υπάρχοντων δεδομένων, μέσω της προσθήκης μιας κανονικής κατανομής, ώστε τα νέα δεδομένα να αναπαριστούν ένα υπαρκτό πλήρες έτος αλλά με τροποποιημένα χαρακτηριστικά. Ο χρήστης μπορεί επίσης να κάνει οποιαδήποτε αυθαίρετη επιλογή σημείων από το χώρο αναπαράστασης, ώστε να δημιουργήσει ένα νέο σετ δεδομένων οποιουδήποτε μεγέθους και με χαρακτηριστικά που να ταιριάζουν στη συμπεριφορά του δικτύου που θέλει να μελετήσει.

Λέξεις κλειδιά

μηχανική μάθηση, βαθιά μάθηση, νευρωνικά δίκτυα, μεταβλητός αυτοκωδικοποιητής, μοντελοποίηση φορτίου, παραγωγή δεδομένων, ανωνυμοποίηση δεδομένων, δεδομένα δικτύου

Abstract

Modern power distribution grids face growing challenges, both due to the transition to smart grids and the integration of new types of loads such as electric cars and grid-scale energy storage systems.These factors make the study of distribution networks under many different conditions and with new tools such as data science and neural networks more necessary than ever. That's contradictory to the fact that grid load data is quite difficult for researchers to access, due to the very limited sources of such data (grid operators) and their confidentiality. All the above, make the development of tools for the anonymization of existing data and the generation of new ones, a great necessity.

This thesis examines the development of a neural network and specifically a Variational Autoencoder, for the purposes mentioned above. The model is able to efficiently train on few available data, learn their underlying features and map them on the latent space, in a useful and perceptible by humans way. The trained model is then used in order to encode and decode existing data, keeping only their important features, thus making it more difficult to identify specific users of the grid from the load timelines and rendering that data much safer to share. Also, using the mapping of the latent space and the trained decoder, we are able to produce new load timelines, keeping the basic features of the original but changing characteristics such as the peak value or the distribution of different load profiles through the year.

After the training the model is able to properly reconstruct the existing data, keeping their plain features such as the average value, the peak value and the position of maximum and minimum values almost intact. It is also able to decode any user created 2-number samples into a full day, consisting of 576 measurements (96 active and reactive power measurements for 3 correlated loads). The method used for generating new data in this thesis, is adding a random normal distribution to the encoded set, so that the newly generated data represent a year of the existing measurements but with adjusted characteristics. However, any arbitrarily selected points can be chosen, in order to create an entirely new dataset with any number of samples (days) and with the characteristics suitable for studying specific aspects of grid behaviour.

Keywords

machine learning, deep learning, neural network, variational autoencoder, load modeling, data generation, data anonymization, grid data

Εκτεταμένη Περίληψη

Σκοπός της διπλωματικής εργασίας

Τα μοντέρνα δίκτυα ηλεκτρικής ενέργειας αντιμετωπίζουν αυξανόμενες προκλήσεις, τόσο από τη μετάβαση σε smart grids όσο και από την εισαγωγή νέων φορτίων όπως ηλεκτρικά αυτοκίνητα και συστήματα αποθήκευσης ενέργειας. Αυτό κάνει όλο και πιο επιτακτική τη μελέτη της συμπεριφοράς των δικτύων υπό διάφορες πιθανές συνθήκες και με τη χρήση νέων εργαλείων όπως η επιστήμη των δεδομένων και τα νευρωνικά δίκτυα. Αυτό έρχεται σε αντίθεση με το γεγονός ότι τα δεδομένα δικτύου που υπάρχουν είναι πολύ περιορισμένα, τόσο ως προς τον αριθμό τους όσο και ως προς τη δυνατότητα διάθεσής τους, λόγω της δυσκολίας συλλογής τους και του ευαίσθητου χαρακτήρα τους. Όλα τα ανωτέρω καθιστούν αναγκαία την ανάπτυξη εργαλείων επεξεργασίας των υπάρχοντων δεδομένων με σκοπό την εξασφάλιση της ανωνυμίας τους καθώς και την πιθανή παραγωγή νέων σετ δεδομένων που θα μπορούν να διευκολύνουν το ερευνητικό έργο πάνω στους συγκεκριμένους τομείς.

Βιβλιογραφία

Η πρώτη μορφή παραγωγής χρονοσειρών φορτίου και η πιο εις βάθος μελετημένη, είναι η δημιουργία χρονοσειρών για την πρόβλεψη του φορτίου, τόσο μακροπρόθεσμα, όσο και βραχυπρόθεσμα. Η οικονομική και ασφαλής λειτουργία του δικτύου καθώς και η ανάπτυξή του, απαιτούν αποδοτικά μοντέλα πρόβλεψης και για το λόγο αυτό υπάρχει εκτεταμένη έρευνα πάνω στη μαθηματική μοντελοποίηση χρονοσειρών φορτίων. Κατά καιρούς έχουν προταθεί και εφαρμοστεί διάφορες μέθοδοι, η κάθε μια με τα πλεονεκτήματα και τα μειονεκτήματά της, μπορούν όμως να χωριστούν σε 2 μεγάλες κατηγορίες: μοντέλα με αυστηρή μαθηματική δομή και αναλυτικά μοντέλα, όπως η γραμμική οπισθοδρόμηση, η πολλαπλή οπισθοδρόμηση, η εκθετική εξομάλυνση και η επαναληπτική μέθοδος ελαχίστων τετραγώνων και μοντέλα που χρησιμοποιούν τεχνικές όπως η ασαφής λογική, η γενετικοί αλγόριθμοι και τα νευρωνικά δίκτυα.

Τα νευρωνικά δίκτυα και ειδικά τα βαθιά νευρωνικά δίκτυα, έχουν εκτεταμένη χρήση σε γεννητικές εφαρμογές, δηλαδή στη δημιουργία καινούργιων δεδομένων με χαρακτηριστικά παρόμοια με εκείνα των δεδομένων στα οποία έχει εκπαιδευτεί το δίκτυο. Τέτοια παραδείγματα είναι τα μοντέλα ενέργειας (energy based models), τα γεννητικά ανταγωνιστικά δίκτυα, οι αυτοκωδικοποιητές και οι μεταβλητοί αυτοκωδικοποιητές (variational autoencoders). Υπάρχει πληθώρα εφαρμογών των ανωτέρω, συχνά με εντυπωσιακά αποτελέσματα, στους τομείς της εικόνας, του ήχου, της αποθορυβοποίησης δεδομένων και αλλού.

Μεθοδολογία Υλοποίησης

Για τους σκοπούς της παρούσας διπλωματική εργασίας, δηλαδή την μοντελοποίηση φορτίων του δικτύου με σκοπό την ανωνυμοποίηση δεδομένων και την παραγωγή νέων, θα αναπτύξουμε έναν Μεταβλητό Αυτοκωδικοποιητή (Variational Autoencoder). Η αρχιτεκτονική αυτή επιλέγεται διότι είναι ένα είδος νευρωνικού δικτύου που μπορεί να εκπαιδευτεί ικανοποιητικά χρησιμοποιώντας μικρό αριθμό δεδομένων, μπορούμε να επιλέξουμε τα χαρακτηριστικά πάνω στα οποία θέλουμε να εστιάσουμε και επίσης υπάρχει πολύ καλή εποπτεία του χρήστη πάνω στα αποτελέσματα της κωδικοποίησης και πώς αυτά διαμορφώνουν το παραγόμενο αποτέλεσμα. Με τον τρόπο αυτό ο χρήστης μπορεί να επιλέξει τι είδους αποτελέσματα θέλει να παράξει (πάντα από την ήδη υπάρχουσα κατανομή).

Ο Variational Autoencoder αποτελείται από τη δομή του Κωδικοποιητή (encoder), τη Ζώνη Δειγματοληψίας (sampling layer) και τον Αποκωδικοποιητή (decoder). Επίσης πρέπει να οριστεί μια Συνάρτηση Σφάλματος (loss function) καθώς και μια Συνάρτηση Βελτιστοποίησης (optimizer).
Ο κωδικοποιητής αποτελεί τη δομή που κωδικοποιεί τα δεδομένα εισόδου. Τα δεδομένα εισέρχονται στην πλήρη τους μορφή και περνώντας από τα διάφορα επίπεδά του, μειώνονται οι αρχικές τους διαστάσεις στο επιθυμητό επίπεδο. Η ζώνη δειγματοληψίας, η οποία είναι αυτή που καθιστά τον αυτοκωδικοποιητή, μεταβλητό αυτοκωδικοποιητή, λαμβάνει τα δεδομένα από την έξοδο του κωδικοποιητή, υπολογίζει τη μέση τιμή και τη διακύμανσή τους και κατόπιν δειγματοληπτεί μια κανονική κατανομή με αυτά τα χαρακτηριστικά.
Με τον τρόπο αυτό επιτυγχάνεται η ομαλότητα και η πληρότητα του χώρου αναπαράστασης, δηλαδή του χώρου μέσα στον οποίο έχουν κωδικοποιηθεί τα δεδομένα και είναι διαφορετικών διαστάσεων από τον αρχικό. Ομαλότητα σημαίνει ότι αν από ένα σημείο του χώρου αυτού, το οποίο αποτελεί αναπαράσταση ενός υπαρκτού δείγματος, κινηθούμε λίγο προς κάποια κατεύθυνση, θα πρέπει να πάρουμε ένα παρόμοιο αποκωδικοποιημένο δείγμα. Πληρότητα, σημαίνει ότι όποιο σημείο του χώρου αποκωδικοποιήσουμε, θα πάρουμε ένα αποτέλεσμα που έχει νόημα, διατηρεί δηλαδή, ως ένα βαθμό, τα βασικά χαρακτηριστικά που έχουν τα αρχικά δεδομένα. Κατόπιν, ο αποκωδικοποιητής παίρνει την έξοδο της ζώνης δειγματοληψίας (κωδικοποιημένο δείγμα) και την αποκωδικοποιεί πίσω σε ένα πλήρες δείγμα.

Η διαδικασία της εκπαίδευσης του αυτοκωδικοποιητή γίνεται ως εξής: τα υπάρχοντα δεδομένα εισέρχονται στον κωδικοποιητή και κατόπιν στη ζώνη δειγματοληψίας. Ύστερα, τα κωδικοποιημένα δεδομένα αποκωδικοποιούνται πίσω. Τα πραγματικά δεδομένα και τα ανακατασκευασμένα, συγκρίνονται μέσω της συνάρτησης σφάλματος για την ομοιότητά τους και κατόπιν η συνάρτηση βελτιστοποίησης αναλαμβάνει να αναπροσαρμόσει τα βάρη των επιπέδων με σκοπό την ελαχιστοποίηση της συνάρτησης σφάλματος, δηλαδή την επίτευξη ομοιότητας μεταξύ δεδομένων εισόδου και ανακατασκευής. Με τον τρόπο αυτό, το δίκτυο μαθαίνει κατά τη διαδικασία κωδικοποίησης να κρατάει τα κύρια χαρακτηριστικά των δεδομένων εισόδου και να μπορεί να τα αναπαράξει κατά την αποκωδικοποίηση.

Για την παρούσα υλοποίηση του νευρωνικού δικτύου χρησιμοποιούνται τα εργαλεία Keras API, το οποίο είναι μια διεπαφή προγραμματισμού ανοιχτού κώδικα και είναι η πρώτη βιβλιοθήκη υψηλού επιπέδου προγραμματισμού που έχει ενσωματωθεί στη βιβλιοθήκη Tensorflow, μια επίσης ευρέως χρησιμοποιούμενη βιβλιοθήκη για τον προγραμματισμό βαθέων νευρωνικών δικτύων. Η γλώσσα προγραμματισμού που χρησιμοποιείται είναι η Python, με τις βιβλιοθήκες Pandas και NumPy για τη διευκόλυνση εισόδου των δεδομένων και την αποδοτική διενέργεια αριθμητικών υπολογισμών.

Τα διαθέσιμα δεδομένα, είναι δεδομένα κατανάλωσης τριών φορτίων (ενεργή και άεργος ισχύς), κατά τη διάρκεια ενός έτους. Η περίοδος δειγματοληψίας είναι 15 λεπτά, δηλαδή 96 μετρήσεις ανά ημέρα. Τα δεδομένα, αφού διαβαστούν από τα .csv αρχεία, μετατρέπονται σε 365 πίνακες της μορφής $(96, 2, 3)$, όπου 96 είναι οι μετρήσεις της ημέρας, 2 τα είδη ισχύος και 3 τα φορτία. Με αυτή τη μορφή εισέρχονται στον κωδικοποιητή, ο οποίος αποτελείται από επάλληλα επίπεδα συνέλιξης (convolutional layer), κανονικοποίησης δέσμης (batch normalization) και γραμμικής ενεργοποίησης (ReLu activation layer). Με τον τρόπο αυτό

γίνεται κωδικοποίηση των δεδομένων. Κατόπιν η ζώνη δειγματοληψίας (sampling layer) υπολογίζει τη μέση τιμή και τη διακύμανση και παράγει δύο δείγματα με τον τυχαίο τρόπο που περιεγράφη παραπάνω. Έτσι οι διαστάσεις των δεδομένων έχουν μειωθεί σε 2. Ακολουθεί η διαδικασία της αποκωδικοποίησης, πίσω σε ένα πλήρες δείγμα.

Αν η διαδικασία της εκπαίδευσης είναι επιτυχής, έχουμε πλέον ένα μοντέλο που μπορεί να δεχθεί στην είσοδό του δεδομένα της μορφής $(96*2*3)$ και να τα αναπαράξει στην έξοδο, κρατώντας μόνο τα κύρια χαρακτηριστικά τους. Αυτό μπορεί να συμβάλλει στην ανωνυμοποίηση δεδομένων, αφαιρώντας κάποια στοιχεία που μπορούν να βοηθήσουν στην ταυτοποίηση συγκεκριμένων χρηστών του δικτύου, όπως μια διακοπή ρεύματος ή μια πολύ συγκεκριμένη μεταβολή που κάποιος παρατηρητής μπορεί να συνδέσει με συγκεκριμένο φορτίο.

Το εκπαιδευμένο πλέον δίκτυο μπορεί να χρησιμοποιηθεί και για τη δημιουργία νέων σετ δεδομένων, οποιουδήποτε μεγέθους, που φέρουν τα βασικά χαρακτηριστικά των αρχικών δεδομένων, προσαρμοσμένα όμως σύμφωνα με τις ανάγκες του χρήστη. Μπορεί δηλαδή να παράξει ένα καινούργιο σετ, το οποίο θα έχει μεγάλη μέγιστη τιμή, μεγάλη διακύμανση, μεγάλες διαφορές μεταξύ από τη μια μέρα στην άλλη ή ό,τι άλλο επιθυμεί. Αυτό μπορεί να γίνει με δύο μεθόδους.
Η μία είναι να πάρει την κωδικοποιημένη μορφή του ήδη υπάρχοντος σετ, δηλαδή ένα πίνακα της μορφής $(365, 2)$ και να εφαρμόσει πάνω του κάποιου είδους μετασχηματισμό, όπως π.χ. να πολλαπλασιάσει κάθε δείγμα με έναν τυχαίο αριθμό από μια κανονική κατανομή. Με τον τρόπο αυτό θα δημιουργήσει ένα καινούργιο σετ, το οποίο μπορεί να έχει μεγαλύτερη ή μικρότερη μέση και μέγιστη τιμή. Το πλεονέκτημα της μεθόδου αυτής, είναι ότι οι μέρες του νέου σετ που θα δημιουργηθούν, θα ακολουθούν την κατανομή των πραγματικών δεδομένων στη διάρκεια ενός έτους. Το μειονέκτημα είναι ότι δεν μπορούν να διαμορφωθούν ανεξάρτητα η μέση και η μέγιστη τιμή ή η διαφοροποίηση μεταξύ των ημερών.
Η άλλη μέθοδος είναι να δημιουργήσει εξ αρχής τα κωδικοποιημένα δεδομένα, δηλαδή να επιλέξει έναν αυθαίρετο αριθμό από σημεία δύο διαστάσεων και κατόπιν να τα αποκωδικοποιήσει. Φυσικά, όσο πιο κοντά είναι αυτά τα σημεία σε αυτά των πραγματικών δεδομένων, τόσο μεγαλύτερη θα είναι και η ομοιότητα των δύο σετ. Αυτή η μέθοδος έχει το πλεονέκτημα ότι ο χρήστης μπορεί να διαμορφώσει ανεξάρτητα τη μέγιστη και τη μέση τιμή (συνεπώς και την κατανάλωση ενέργειας) ή τη διακύμανση από μέρα σε μέρα. Το παραγόμενο σετ όμως μπορεί να μην ακολουθεί την πραγματική κατανομή μέσα στο έτος.
Η επιλογή της μεθόδου έγκειται κάθε φορά στην σκοπούμενη χρήση και τα χαρακτηριστικά τα οποία θέλουμε διατηρήσουμε ή να αλλάξουμε.

Αποτελέσματα Υλοποίησης

Το Δίκτυο κατάφερε να εκπαιδευτεί επιτυχώς πάνω στα διαθέσιμα δεδομένα. Το πρώτο κριτήριο για την ποιότητα της ανακατασκευής, είναι η σύγκριση κάποιων βασικών μεγεθών, όπως η μέση τιμή, η μέγιστη τιμή και η ετήσια κατανάλωση. Συγκρίνοντας τα προαναφερθέντα μεγέθη, βλέπουμε ότι στη μέση τιμή τόσο της ενεργούς, όσο και της αέργου ισχύος, η διαφορά είναι κάτω από 3% και στα τρία φορτία. Η μέγιστη τιμή της ενεργού ισχύος έχει διαφορά 7,5% για το πρώτο φορτίο και 3,7% για το δεύτερο και το τρίτο, ενώ η μέγιστη άεργος ισχύς έχει διαφορά 5,8% για το πρώτο, 15,6% για το δεύτερο και 2,24% για το τρίτο. Τα αποτελέσματα αυτά, με εξαίρεση ίσως τη μέγιστη άεργο ισχύ του δεύτερου φορτίου, κρίνονται

ως ικανοποιητικά. Υπενθυμίζεται πως η εκπαίδευση του δικτύου είναι μια στοχαστική διαδικασία και τα αποτελέσματα αυτά μπορεί να διαφέρουν λίγο κάθε φορά που η εκπαίδευση επαναλαμβάνεται από την αρχή.
Ο δεύτερος τρόπος που μπορεί να ελεγχθεί η ποιότητα ανακατασκευής είναι οπτικά. Παρατηρώντας τα γραφήματα της Εικόνας 4.1, που απεικονίζουν τις αυθεντικές χρονοσειρές σε σύγκριση με τις ανακατασκευασμένες, μπορούμε να δούμε πως υπάρχει πολύ καλή ποιότητα ανακατασκευής. Οι μέγιστες και οι ελάχιστες τιμές βρίσκονται χρονικά στα ίδια σημεία, η αλληλοσυσχέτιση των φορτίων έχει διατηρηθεί και οι μικρές διαφορές που παρατηρούνται κινούνται εντός ικανοποιητικού πλαισίου.

Όσων αφορά την ικανότητα παραγωγής νέων δεδομένων και πάλι το αποτέλεσμα κρίνεται επιτυχές. Εφαρμόζοντας την πρώτη μέθοδο που περιεγράφη ανωτέρω, παίρνουμε την κωδικοποιημένη μορφή των δεδομένων $(365, 2)$ και προσθέτουμε σε αυτά τυχαίους αριθμούς, από μια κανονική κατανομή με μέση $\mu = 0$ και $\sigma = 2$. Με τον τρόπο αυτό, όπως βλέπουμε και στην Εικόνα 4.4, κάποια σημεία έχουν διασκορπιστεί σε μεγαλύτερο μέρος του επιπέδου αναπαράστασης. Αποκωδικοποιώντας τα νέα δεδομένα, παίρνουμε ένα καινούργιο σετ στο οποίο παρατηρούμε το εξής: η μέγιστη ενεργός ισχύς έχει αυξηθεί κατά ένα σημαντικό ποσοστό και στα τρία φορτία (16,5%, 18,6% και 22,3%) ενώ η μέση τιμή και η άεργος έχουν αυξηθεί πολύ λιγότερο.
Στο δεύτερο παράδειγμα, θα πολλαπλασιάσουμε τα κωδικοποιημένα δεδομένα με μια κανονική κατανομή με $\mu = 1.7$ και $\sigma = 0, 3$. Όπως μπορούμε να δούμε στην Εικόνα 4.6, η πλειονότητα των σημείων έχει διασκορπιστεί ευρέως στο επίπεδο αναπαράστασης. Αποκωδικοποιώντας τη νέα κωδικοποιημένη μορφή, παίρνουμε ένα σετ δεδομένων στο οποίο βλέπουμε να έχει αυξηθεί η μέγιστη ενεργός ισχύς και στα τρία φορτία (κατά 46,8%, 29,9% και 57,5% αντίστοιχα) αλλά και η μέση τιμή της ισχύος (ενεργού και άεργης) όπως και η κατανάλωση ενέργειας έχουν αυξηθεί επίσης σημαντικά, όπως φαίνεται στους Πίνακες 4.7, 4.8 και 4.9.

Όπως βλέπουμε από τα στοιχεία που προκύπτουν από τις δύο παραπάνω εφαρμογές και έχοντας δείξει την ποιότητα ανακατασκευής που επιτυγχάνει το δίκτυο, προκύπτει πως έχουμε πλέον ένα σετ το οποίο θα ακολουθεί την κατανομή του πραγματικού μέσα στο έτος, έχει όμως αλλαγμένα κάποια βασικά χαρακτηριστικά

Για τη χρήση της δεύτερης μεθόδου παραγωγής νέων δεδομένων, μπορούμε να χρησιμοποιήσουμε το χάρτη του επιπέδου αναπαράστασης που φαίνεται στην Εικόνα 3.13. Όπως βλέπουμε, κινούμενο στον άξονα x αλλάζει κατά βάση η μέση και η μέγιστη τιμή του φορτίου, ενώ κινούμενοι στον άξονα y, αλλάζει κυρίως κάποιο συγκεκριμένο χαρακτηριστικό κάθε φορά, όπως η ύπαρξη τοπικών μεγίστων και το ύψος τους ή η διακύμανση του φορτίου μέσα στη μέρα. Επιλέγοντας οποιονδήποτε αριθμό σημείων από το επίπεδο και χρησιμοποιώντας το χάρτη για να δούμε τι χρονοσειρές παράγει το κάθε σημείο για καθένα από τα τρία φορτία, μπορούμε να παράξουμε καινούργια σετ δεδομένων, που να περιέχουν οποιονδήποτε αριθμό ημερών. Η επιλογή επαφίεται στο χρήστη και εξαρτάται κάθε φορά από την εφαρμογή για την οποία προορίζονται να χρησιμοποιηθούν τα δεδομένα.

Συμπεράσματα

Το δίκτυο κατάφερε επιτυχώς να εκπαιδευτεί, χρησιμοποιώντας έναν πολύ μικρό (για τα δεδομένα της βαθιάς μάθησης) αριθμό δειγμάτων. Επίσης η εκπαίδευση καθώς και η κωδικοποίηση και αποκωδικοποίηση των δεδομένων γίνονται σε πολύ μικρό χρόνο, σε επίπεδο λεπτού και δευτερολέπτων αντίστοιχα, χρησιμοποιώντας πόρους του Google Colaboratory. Αυτό είναι ιδιαίτερα σημαντικό, ειδικά σε εφαρμογές που απαιτούν την παραγωγή μεγάλου όγκου δεδομένων.

Περαιτέρω εφαρμογές του παρόντος δικτύου, μπορεί να περιλαμβάνουν υλοποιήσεις που να εστιάζουν σε διαφορετικά χαρακτηριστικά του δικτύου ή να περιλαμβάνουν μεγαλύτερο αριθμό φορτίων.
Μια επίσης ενδιαφέρουσα εφαρμογή που αξίζει διερεύνησης, θα ήταν η ενσωμάτωση στη συνάρτηση σφάλματος μιας υπολογιστικής προσομοίωσης ροής φορτίου. Με τον τρόπο αυτό, ο χρήστης μπορεί να θέσει ως κριτήριο εκπαίδευσης κάποια συγκεκριμένη κατάσταση του δικτύου και στόχος του νευρωνικού δικτύου πλέον θα είναι η δημιουργία των κατάλληλων χρονοσειρών φορτίου που θα μπορούσαν να οδηγήσουν το δίκτυο στην επιθυμητή κατάσταση, η οποία μπορεί να είναι είτε μια βέλτιστη είτε μια οριακή κατάσταση που αξίζει μελέτης.

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον κ. Χατζηαργυρίου και τον κ. Πεδιαδίτη για την εμπιστοσύνη τους και την καθοδήγηση στην εκπόνηση αυτής της διπλωματικής εργασίας.

Επίσης, θα ήθελα να ευχαριστήσω την οικογένειά μου για την αμέριστη στήριξη όλων αυτών των χρόνων κατά τη διάρκεια των σπουδών μου καθώς και τους φίλους και συμφοιτητές που μοιραστήκαμε μαζί τις κοινές μας αγωνίες μέσα και έξω από τη σχολή.

Contents

1 Motivation **1**
1.1 The need for artificial loads' data 1
1.2 The need for open data 1

2 Literature Review **3**
2.1 Demand prediction: the most common form of data generation 3
2.2 Deep generative neural architectures 6

3 Methodology **13**
3.1 Variational Autoencoders Analysis 13
3.2 Development tools 16
3.3 Dataset Architecture 16
3.4 Model Architecture 17
3.5 Using the trained model 24

4 Case Study Results **29**
4.1 Assumptions and Case Studies Description 29
4.2 Latent Space Evaluation 29
4.3 Reconstruction Evaluation 30
4.4 Data Generation 34

5 Conclusions **41**
5.1 Methodology and Results 41
5.2 Future Work 41

Bibliography **43**

A Code Listings **49**

List of Figures

2.1 Example of fuzzy logic classification [29] 5
2.2 Genetic algorithm flowchart [34] 6
2.3 General Structure of a Generative Adversarial Network [45] 7
2.4 Images generated by a GAN created by NVIDIA. [47] 8
2.5 Example of image denoising using a VAE[51] 8
2.6 Edge detection based on Sparse Autoencoder network[52]. 9
2.7 Examples of different autoencoder configurations. 10
2.8 Visualisations of learned data manifold for generative models with two-dimensional latent space, learned with AEVB. 11

3.1 Illustration of Variational Autoencoder's general structure [54] 13
3.2 Visualizations of 2-D latent representation. Points with different colors correspond to the digit labels. These were sampled from q(z|x) in the VAE and q(z|x, w) in both the CVAE and JMVAE. [55] 14
3.3 The process of learning probabilistic representations in the latent space for three schedules [56] 15
3.4 Latent space representation of a dataset after 2.000 training epochs. [57] . . . 16
3.5 Main development tools 16
3.6 Model Architecture 17
3.7 Importing and reshaping module 19
3.8 Encoder layers 20
3.9 Sampling Layers 21
3.11 Training Process 24
3.10 Decoder layers 26
3.12 Mapping of the latent space after a different number of training epochs . . . 27
3.13 Latent space mapping of the three loads 28

4.1 Reconstruction of 5 random days 32
4.2 Heatmap of the original and the reconstructed dataset 33
4.3 Mapping of the original and the reconstructed dataset on the latent space . . 33
4.4 Mapping of the original and the new dataset on the latent space 34
4.6 Plot of the original and the new code on the latent space 36
4.5 Random days generated by adding to the original encoded set 38
4.7 Random days generated by multiplying the original encoded set 39

List of Tables

4.1 Load 1 Metrics 30
4.2 Load 2 Metrics 31
4.3 Load 3 Metrics 31
4.4 Load 1 Metrics 35
4.5 Load 2 Metrics 35
4.6 Load 3 Metrics 35
4.7 Load 1 Metrics 36
4.8 Load 2 Metrics 37
4.9 Load 3 Metrics 37

1 Motivation

1.1 The need for artificial loads' data

Modern power transmission systems and their operators face growing challenges due to the ongoing energy model transformation and specifically the transformation of power grids. Grid expansion and adaptation is a very critical, time consuming and expensive process, so proper modeling is more than necessary. New types of loads, like electric cars and grid batteries are being introduced and old types of loads change their characteristics due to reasons like decentralized production [1] and fluid energy markets[2]. From an operator's point of view, new grid configurations offer new capabilities [3] but also challenges that require adaptive tools and enabling technologies[4, 5]. One of the main difficulties when dealing with grid data is the small number of data sources. Unlike social media users or images, power grids are not numbered in the billions, are not exhaustively monitored and until recently only the necessary metrics were systematically recorder. That being said, datasets of power grid data are hard to find and limited in size and time period. This poses a big challenge for researchers in the field, for whom the first step of a new project is, more than often, securing real world data. The most excessively studied aspect of electricity data has been demand prediction, due to its undisputed economic importance. Demand prediction is actually a data generation process. However, these methods focus on producing new data with the same characteristics as the old ones, taking into account factors like seasonality, economic growth, and various environmental and socioeconomic facts. They can be very useful for grid expansion or production planning. However, modern practice and scientific research has many more applications that could benefit from data whose features have been chosen on demand but still represent the real world. Big artificial datasets can prove to be very useful in This is the main research goal of this thesis.

1.2 The need for open data

Another problem is that, even when such datasets exist, they can contain very sensitive information that prohibits public distribution, making research and experimentation difficult for scientists working in the field. Creating anonymous data can help overcome this. Data anonymization is a blooming field, both scientifically and commercially. At first glance, anonymizing data may seem as simple as deleting an excel column. However new research [6] [7] always comes to show that this is not a simple task and techniques we considered sufficient are not effective at all. Building completely new datasets, by drawing samples that follow the distribution of the originals, looks like a promising technique, applicable to power data. Those two aforementioned challenges make artificial datasets an interesting field of research that can accelerate the adoption of data science in power transmission grids.

2 Literature Review

2.1 Demand prediction: the most common form of data generation

As mentioned in the previous chapter, demand prediction is the earliest and most well studied form of data generation. Operation of power system networks needs to be economic and secure. Economic operation is achieved through proper planning of power sources usage, infrastructure maintenance and expansion, and losses minimization. Secure operation is achieved through careful distribution of loads among the power sources, taking into consideration the technical limitations, adaptation capabilities, contingencies and every other operating factor of both the generating units and the distribution grid. Economic operation and secure operation contradict each other in most cases, so precise calculation and prediction of the grid's variables is essential, as bigger uncertainties cause bigger need for capacity reserves, contingency planning and operator interventions. That being said, it is no surprise that demand prediction is thoroughly studied, with various methods being proposed by researchers.
Many different techniques are being used nowadays, with each having its own advantages and disadvantages. For example, one application may require more precise prediction of the load's peak and another one better prediction of the mean value. Also some applications may have computing power or solving time restrictions, so a wide range of methods has been proposed and used by scientists. These methods can be classified in three basic categories:

2.1.1 Hard Computing Techniques

Traditional Forecasting Techniques

Traditional forecasting techniques include linear regression, multiple regression, exponenial smoothing and iterative reweighted least-squares technique.
Linear regression is one of the most common statistical techniques. It assumes that the load consists of two trends. One is the standard load trend, representing the base load that is necessary every day, under any -usual- conditions. The other one is linearly dependent on a number of factors, determined every time by the designer of the model. These factors can be the weather, holidays, seasonalities or any other condition that can affect the energy demand.
Multiple regression is probably the most popular method that incorporates a larger number of factors and also non-linear relationships. It has been used in many different ways such as predicting a daily peak and then using it to produce hourly data [8]. The method of least-squares has also been used for identification and quantification of different types of loads on the grid [9].
Exponential smoothing is another approach used in load modeling. The load is modeled based on previous data and then this model is used to forecast future demand. A very well known and simple exponential smoothing method is the Holt-Winters forecasting procedure [10]. It uses three smoothing constants, one for the base component, one for the trend component and one for the seasonal component of the load. However, it was found that analyzing fast growing areas was very difficult through the direct application of the Holt-Winters method

[11]. As a solution, the model was enhanced with power spectrum analysis and adaptive autoregressive modeling [12].
Iterative Reweighted Least-Squares is a method used to identify the model order and parameters. By controlling one variable at a a time, it determines the optimal starting point [13]. An autocorrelation and partial autocorrelation function is then utilized to build a model for the dynamics of the load. The three variables used to determine the model, are the weighting function, the tuning constants and the weighted sum of the squared residuals.

Modified traditional techniques
Modified traditional techniques are adaptations of the traditional ones, modified so as to incorporate changing environmental conditions or other factors (adaptive demand forecasting, stochastic time series, autoregressive, autoregressive moving-average and autoregressive integrated moving-average models, support vector machine based techniques).
Adaptive demand forecasting technique, uses a model that automatically corrects the parameters of the forecasting depending on real time or almost real time environmental conditions [14]. By processing the current prediction error and the current weather data, a next state vector is calculated and used as the prediction result. Enhanced algorithms for this technique have been proposed [15] with very good results.
Stochastic time series is a widely used method in load forecasting. It includes different kinds of models, such as autoregressive, autoregressive moving-average and autoregressive integrated moving-average. Based on the assumption that the load does have an internal structure, such as a seasonal variation, a trend or an autocorrelation. Autoregressive model assumes that that the load is a linear combination of previous loads. Autoregressive moving-average uses previous values and a term of white noise. A time-temperature methodology has been proposed using this method [16], for forecasting highly complex load characteristics in fast developing areas by splitting the peak demands into a deterministic and a stochastic component and predicting the latter using autoregressive moving average models. Autoregressive integrated moving-average models requires stationary form time series. The trend component is used to predict the growth of the load, the weather parameters to calculate the weather-sensitive component and autoregressive integrated moving-average to calculate the weekly cyclic component of the peak load [17].
Support vector machine based techniques is a series of machine learning methods based on statistical learning theory. It has been shown that weather conditions are not a very effective means for mid-term forecasting and time series may deliver better results [18]. By penalizing insensitive errors more heavily than the distant insensitive ones, researchers created the C-ascending support vector machines which consistently performs better than the standard support vector machines [19].

2.1.2 Soft Computing techniques

Almost every real world system is inherently imprecise. Also a huge and multifactorial system like the electric grid can never be modeled with absolute precision. In fact, the human ability to interpret the world and make decision based on the perception of it, relies almost entirely on "educated guesses" rather than excellent knowledge and exact calculations. Also, many engineering problems (including power grid management) don't really require a closed form solution but a good enough approximation, so that the system can work reliably and within

the predetermined limits of good operation. Soft computing appears to be a very promising field for such kinds of problems. The tolerance for imprecision allows the application of new kinds of solutions, some of which can answer many problems of modern day grid management such as computational speed and computational energy cost or data volumes. Soft computing techniques include genetic algorithms, fuzzy logic, neural networks, evolutionary algorithms and knowledge-based expert systems.

Fuzzy Logic

Fuzzy logic is a computing technique that is based on the degree of certainty of the produced results. The outcome of a calculations is not a definitive answer, but rather an answer and a degree of confidence about it. It is widely used in the engineering and the economics field, where exact modeling of systems is not always possible. It has an easy structure and is quite effective in controlling machines [20]. It also does not require huge amounts of data in order to function. A basic system consists of a Rule Base, which is a set of rules and membership functions that regulate and control the system, a Fuzzifier that transforms numeric inputs into fuzzy sets, an Inference Engine, which creates the rules for the input and a Defuzzifier that transforms the fuzzy sets back into an explicit output. Short-term forecasting using a fuzzy logic based system to calculate the update function has been proposed and evaluated on the Taiwan power system [21]. Fuzzy logic has been widely used in combination with other techniques, such as neural networks and expert systems and has been found to deliver good results in complex situations [22], [23],[24],[25],[26].Also several fuzzy neural network methods were examined by Dash, Leiw and Rahman [27], and also combined with fuzzy expert systems [28].

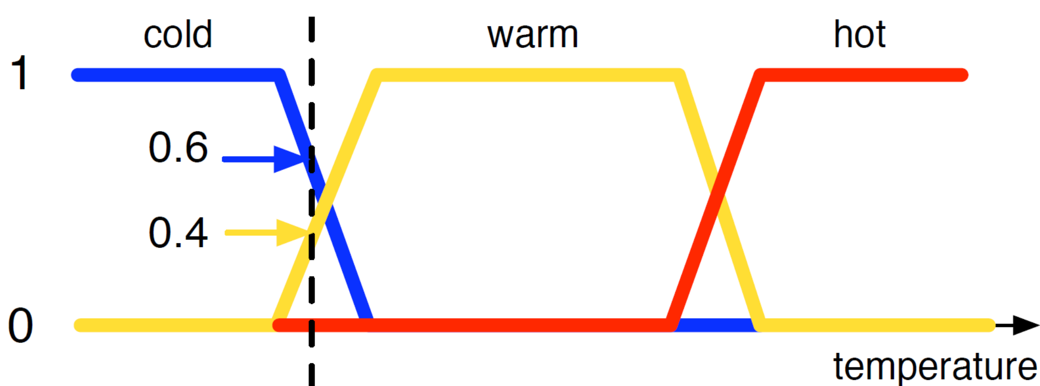


Figure 2.1: Example of fuzzy logic classification [29]

Genetic Algorithms

Genetic (or evolutionary) algorithms simultaneously evaluate many different points of the search space without having many restrictions about it (like uni-modality or differentiability). The process starts with a number of parent vectors, randomly generated from a selected range, encoding the parameters to be optimized. From these parent vectors, new offsprings are generated and evaluated. From the new population of parents and offsprings, the most

well-performing (according to the chosen criteria) are selected and passed onto the next generation (the next iteration of the algorithm) to create new offsprings and repeat the process all over, until a certain performance goal or the maximum number of generations is reached. Genetic algorithms have been used for both short and long term load generation and optimal generation units' commitment [30],[31],[32],[33].

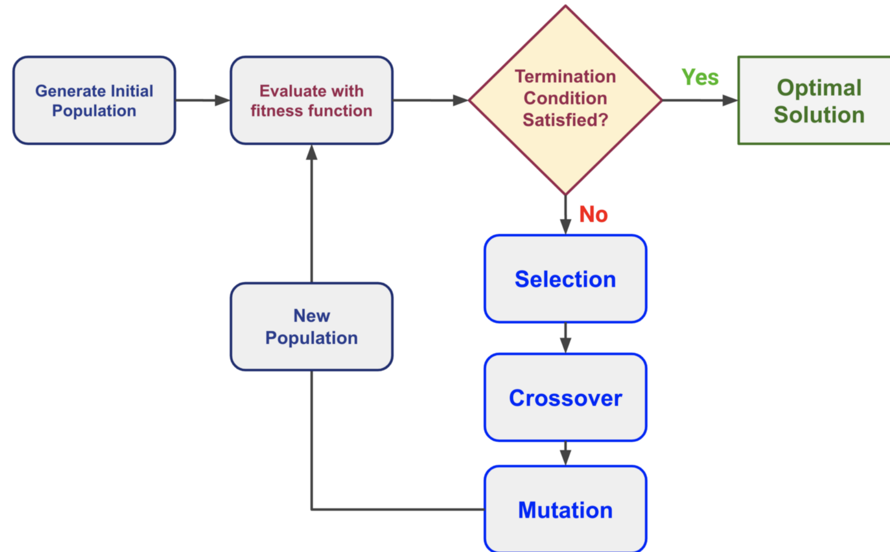


Figure 2.2: Genetic algorithm flowchart [34]

Neural Networks

Neural networks have a very wide range of applications and their popularity has exploded in recent years. They have been used in various ways for demand prediction and data generation. Neural networks have the advantage of "learning" the underlying structure of the data, thus being able to adjust in many different occasions rather than being carefully adjusted for every specific application and have been used and tested as early as 1991 [35]. Also most methods do not require a functional form of the model. Artificial neural networks have been successfully used with back propagation algorithms, fuzzy logic, genetic algorithms and particle swarm optimization [36]. Their disadvantage is that training takes time (which however is being overcome as computing power increases) and proper training requires a very large amount of data in most cases. Neural networks bibliography for data generation in general will be thoroughly examined in the next section.

2.2 Deep generative neural architectures

Generative modeling dates as early as 1980 [37]. The general idea of generative modeling, is learning the distribution of the existing (learning) data and then being able to draw samples form that distribution. It belongs to the class of unsupervised learning, since the training data does not have labeled outputs on which the training is evaluated. Such architectures are the Hopfield Networks [38], Boltzmann Machines [39], Deep Belief Networks [40], Helmholtz Ma-

chines [41]. Many different approaches have been proposed for sample generation, with the main ones being energy-based models, generative adversarial networks, variational autoencoders and many hybrid implementations.Such models have a very wide variety of applications, such as image super-resolution, text-to-image conversion, attribute manipulation, video synthesis, graphics augmentation, texture generation, character movement, speech synthesis and even drug synthesis and medical image modality conversion [42].

2.2.1 Energy Based Models

Energy-Based Models (EBMs), as described in the abstract of "A tutorial on energy-based learning" [43], "capture dependencies between variables by associating a scalar energy to each configuration of the variables. Inference consists in clamping the value of observed variables and finding configurations of the remaining variables that minimize the energy". Modeling of the underlying distribution is done by defining an energy function that gives lower energies for observed properties and higher energies for unobserved ones. Some examples of EBMs are IGEBM [44]

2.2.2 Generative Adversarial Networks

Generative Adversarial Networks (GANs) consist of two different networks, a generator and a discriminator. First the discriminator is trained on actual samples and learns to discriminate between samples that belong to the dataset and samples that don't. The generator is then used to generate samples, that are classified by the discriminator as real or made up. The main training process is the generator trying to create samples that will be labeled by the discriminator as real ones.

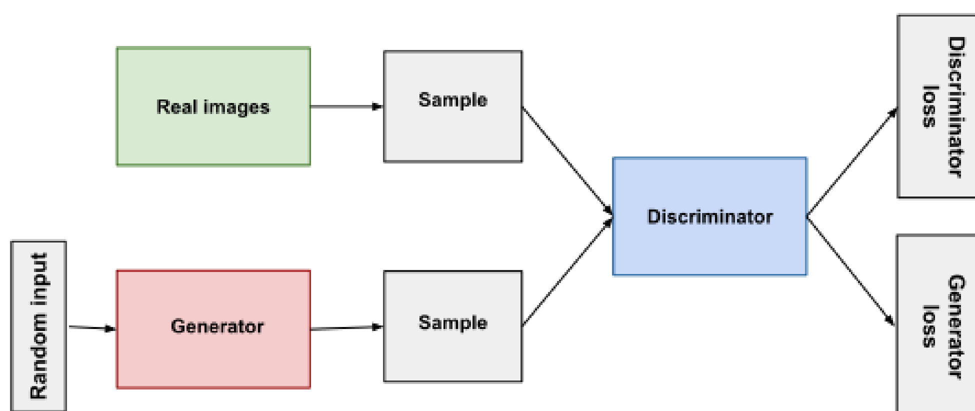


Figure 2.3: General Structure of a Generative Adversarial Network [45]

Generative Adversarial Networks have evolved greatly in recent years and have some very impressive applications to showcase, like styleGAN [46], [47] ([48]) that creates extremely realistic random images of peoples' faces from scratch and pixray/text2image ([49]) that combines natural language processing perception engines and GANs, giving the user the ability to

create custom images with the content being dictated in natural language.

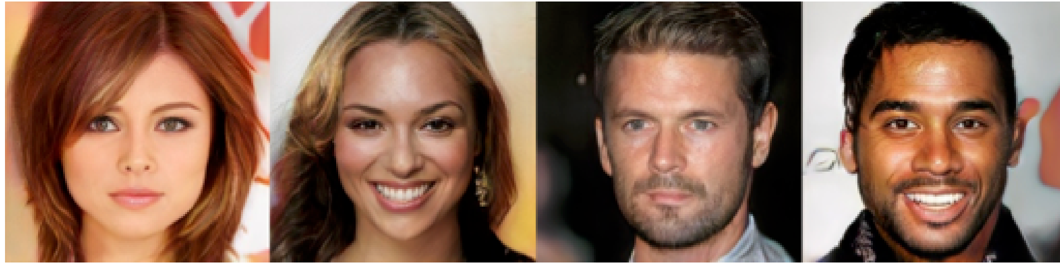


Figure 2.4: Images generated by a GAN created by NVIDIA. [47]

2.2.3 Autoencoders

An Autoencoder is a neural network that uses consecutive encoding and decoding of the samples provided, in order to learn the underlying characteristics of the data. Autoencoders have delivered great results in dimensionality reduction, semantic hashing, denoising and information retrieval. Such implementations are very common in speeding up database operations[50] and denoising. The encoder's job is to transform the input in a certain way through its hidden layers and then pass it on to the decoder. The decoder takes the transformed input (known as *code*) and tries to rebuild it as close as possible to the original data. The result is evaluated by the loss function (also referred to as *reconstruction error*), which the network tries to minimize.

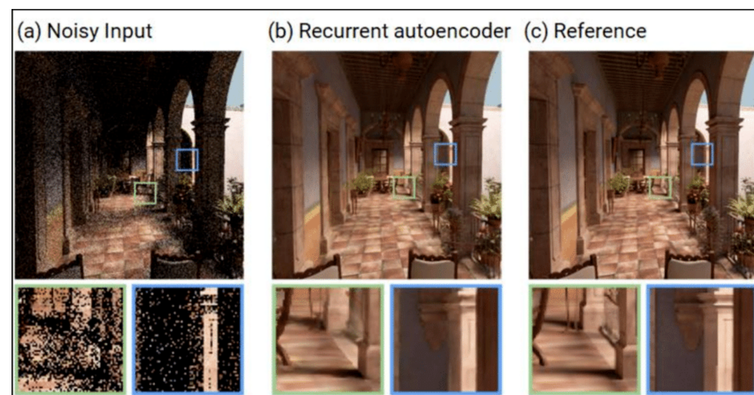


Figure 2.5: Example of image denoising using a VAE[51]

This process of transforming the samples and trying to build them back, forces the encoder to learn the principal characteristics of the input[50]. Autoencoders can be trained with the techniques used in feed-forward networks, like gradient descent but also with recirculation, which compares the neurons' activation by the original sample and the reconstructed one. The latter is more similar to the way biological neurons work, but is quite uncommon in artificial neural networks. The most common types of layers used in autoencoders are fully connected and convolutional layers. Autoencoders are divided into two general categories, according to

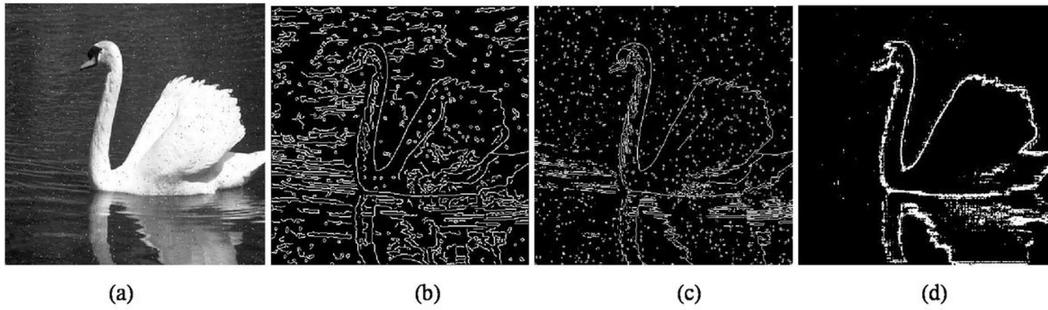


Figure 2.6: Edge detection based on Sparse Autoencoder network[52].

their hidden layers size relatively to the sample size.

Overcomplete autoencoders, are built with hidden layers larger that the input. These autoencoders may learn to simply copy the input to the output, thus learning nothing useful about the features of the data. This can be avoided with the use of an extra function that adds a penalty to the reconstruction error when the input is exactly copied to the output.
Undercomplete autoencoders are built with hidden layers smaller than the input. This, of course, is the most common configuration. The type of layers, the depth, the activation functions and the final size of the coded sample are design choices, based on multiple factors such as the complexity of the input data, the desired level of size reduction or the characteristics of the sample that we want to be omitted or preserved.

2.2.4 Variational Autoencoder

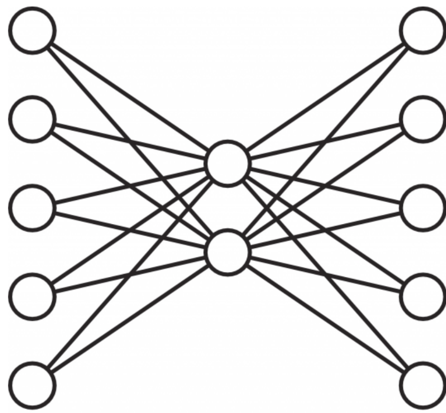
The first Variational Autoencoder is presented in the paper "Auto-Encoding Variational Bayes"[53]. That paper introduces the Stochastic Gradient Variational Bayes (SGVB) estimator and the Autoencoding Variational Bayes (AEVB) algorithm, which is shown below

---

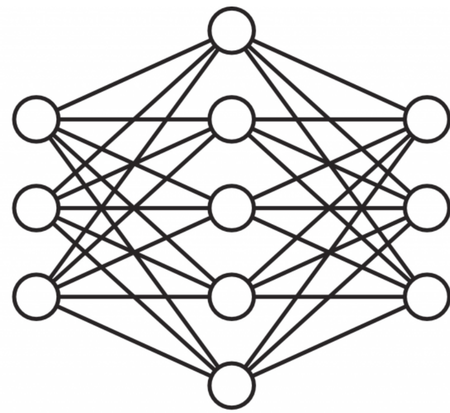
Algorithm 1 Minibatch version of the Auto-Encoding VB (AEVB) algorithm. Either of the two SGVB estimators in section 2.3 can be used. We use settings M = 100 and L = 1 in experiments.

θ, φ ⟵ initialize parameters
repeat
$X^M \leftarrow$ Random minibatch of *M* datapoints (drwan from full dataset)
$\epsilon \leftarrow$ Random samples from noise distribution $p(\epsilon)$
$g \leftarrow \nabla_\theta, \phi \mathcal{L}^M(\theta, \phi; X^M, \epsilon)$ (Gradients of minibatch estimator)
$\theta, \phi, \leftarrow$ Update parameters using gradients g (e.g. SGD or Adagrad)
until convergence of parameters (θ, ϕ)
return θ, ϕ

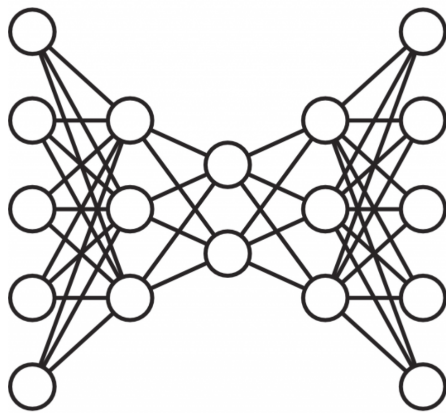
In their paper, the authors used the above algorithm to implement the first Variational Autoencoder (VAE), which they trained on the Frey Face Images and the MNIST dataset. The achieved results are shown in fig. 2.8.



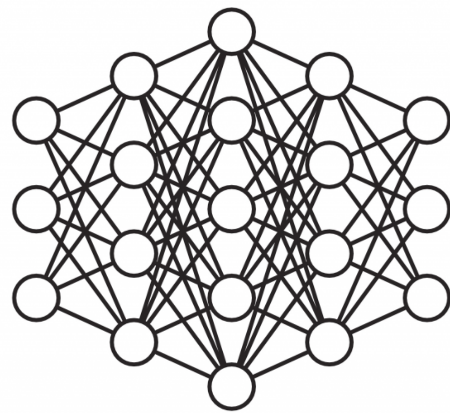
(a) Shallow undercomplete



(b) Shallow overcomplete



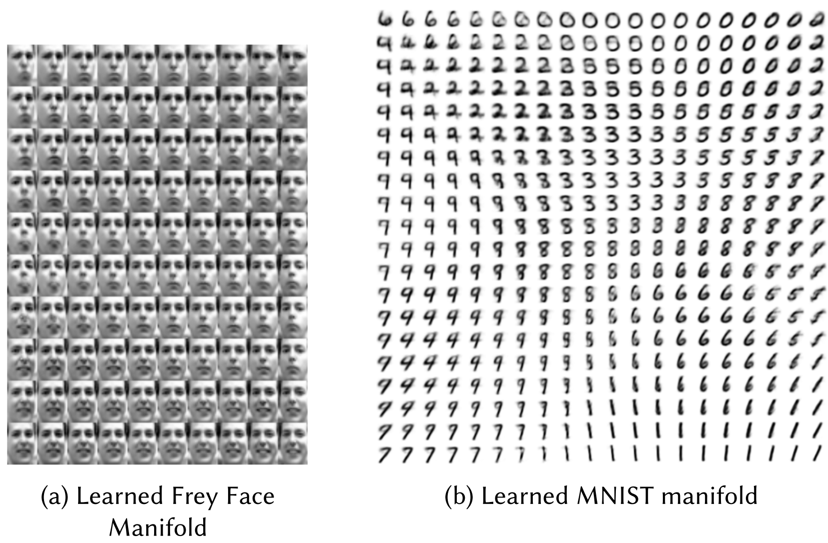
(c) Deep undercomplete



(d) Deep overcomplete

Figure 2.7: Examples of different autoencoder configurations.

Variational Autoencoders will be thoroughly examined in the next section, as the chosen methodology for this thesis.



(a) Learned Frey Face Manifold

(b) Learned MNIST manifold

Figure 2.8: Visualisations of learned data manifold for generative models with two-dimensional latent space, learned with AEVB.

3 Methodology

In order to accomplish the goal of this thesis, which is to develop a neural network able to learn the characteristics of an electric load and produce samples on demand, a Variational Autoencoder will be built. The VAE will be programmed using Python programming language and the Keras API, which is an open-source software library for artificial neural networks. The VAE is chosen because it allows the generation of samples with characteristics chosen by the user, since the user can choose the part of the latent space that he/she wants to sample from, unlike other architectures that generate very truthful and complex but random samples, like GANs. Also, VAEs do not require a large amount of data in order to be trained, which is a prohibiting factor since real-world grid data are not easily accessible (and that's the main motivation behind this thesis).

3.1 Variational Autoencoders Analysis

In this section we will explain in detail the individual building blocks of a VAE.

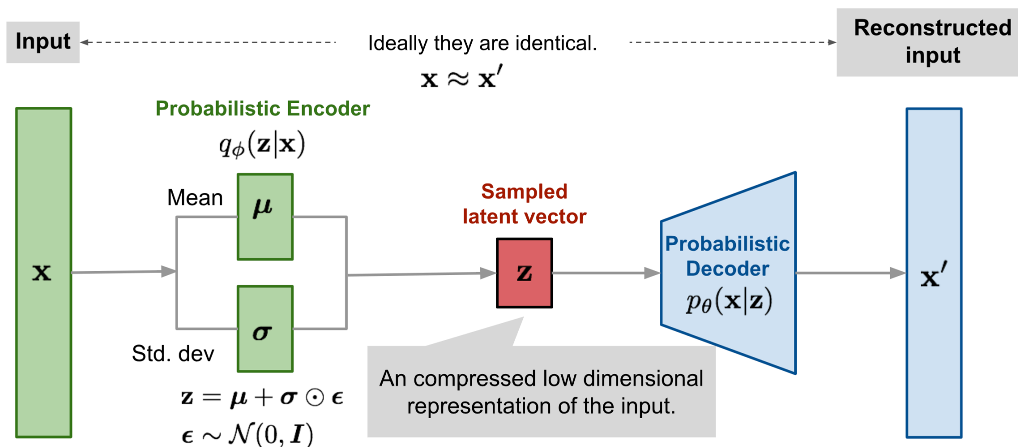


Figure 3.1: Illustration of Variational Autoencoder's general structure [54]

3.1.1 Encoder

The first block of the neural network is the Encoder, whose job is, as described above, to encode every sample of the data (the *input sample*) into a new, usually smaller one (the *code*). The first layer of the encoder (the *input layer*) must have the size of one sample. The input layer is then connected to the rest of the encoder layers (the *hidden layers*). Most commonly used architectures are series of dense layers and series of convolutional layers. However, any combination is possible, depending on the application. Dense layers offer a more open approach to the characteristics of the data and few hyperparameters for tuning, like activation

function, bias and reguralizers. Convolutional layers offer more parameters for adjustment, including the above plus convolution window, filters number, stride size, padding and dilation rate. These are particularly useful, especially when we know on what characteristics we want to focus. These could be, for example, edge detection and color patterns in an image, a frequency progress in sound or a certain periodicity in a timeline. The network may consist of any number of hidden layers, interconnected in any possible way. VAEs with many or few hidden layers are characterized as *deep* or *shallow*, respectively. The last layer of the encoder, the *output layer* is connected to the sampling layer.

3.1.2 The Sampling Layer

The sampling layer is the distinctive feature of the ***Variational*** Autoendocer. Instead of passing the coded sample straight into the decoder, we run it through the sampling layer. This layer creates a normal distribution with mean value and standard deviation accordingly and then samples this distribution and passes the sampled values to the decoder. The reason behind this will be clearly understood after the Latent Space subsection that follows.

3.1.3 Latent Space

Latent space is the place on which the features of the input are mapped. It can have any number of dimensions, and the number of these dimensions is a critical design choice. Theoretically, any initial dimensionality could be reduced to a single dimension. If we give the encoder and decoder large enough degrees of freedom, they could just number an N number of samples (1,2,3...N) and then reconstruct every sample without any reconstruction error. That of course would have nothing to offer, since our model would learn nothing about the important features of the data but simply copy the input to the output. So a good model would produce a latent space that every dimensions corresponds to a specific feature and that space can produce every possible sample of the original distribution.

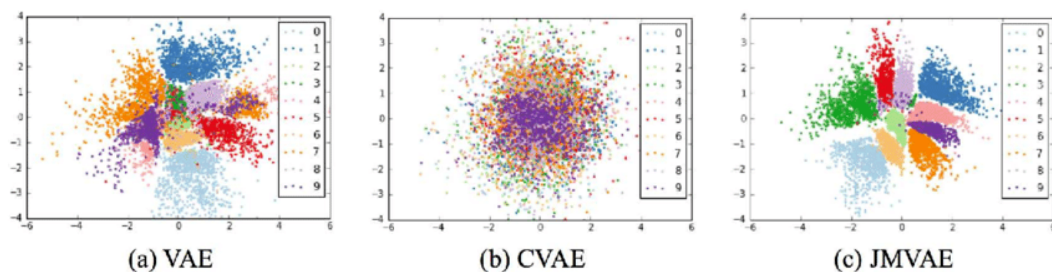


Figure 3.2: Visualizations of 2-D latent representation. Points with different colors correspond to the digit labels. These were sampled from q(z|x) in the VAE and q(z|x, w) in both the CVAE and JMVAE. [55]

As we can clearly see in Figure 3.2, different models can produce very different results of the latent space for the same training data. The best scenario is that the features learnt by the model and represented in each dimension, correspond to those of human perception, like the shape of smile and the face orientation in Figure ??. This is not always an easy task and success cannot be guaranteed from the beginning.

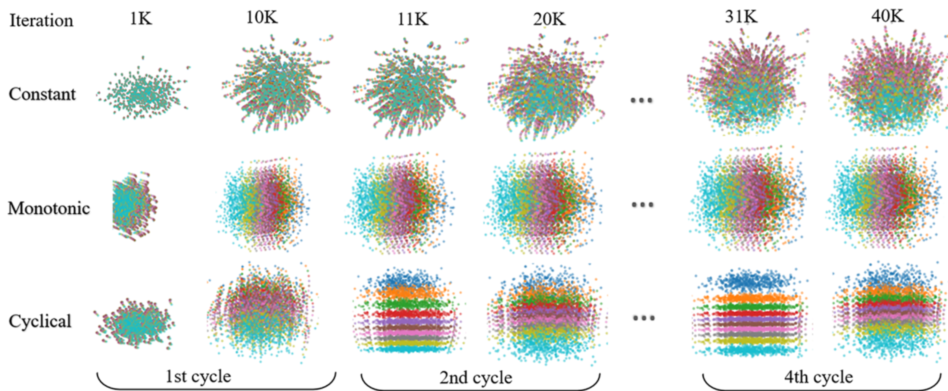


Figure 3.3: The process of learning probabilistic representations in the latent space for three schedules [56]

The effect of the sampling layer on the latent space

As described above, the decoding of different points of the latent space will produce different samples. However, in order for the latent space to be useful for data generation, some qualities must be met. First of all, excessive dimensionality and reconstruction loss reduction can lead to overfitting. This will result in some latent space points giving exact copies of the original data and some other meaningless results. The other thing to consider is that the latent space must have some regularity, meaning that small changes in the position within the latent space, should not result in huge differences of the decoded sample. In order to achieve a regularized latent space, we add an extra step in the encoding process. Instead of mapping the input to a single point, we encode it as a distribution, by adding a probabilistic layer that takes the mean and standard deviation (often with some small randomness added) of the sample. This distribution, usually chosen to be normal, is then sampled and the sampled representation is passed on to the decoder and follows the regular training path of reconstruction, evaluation and error propagation. This, very simply, forces the distributions produced by the decoder to be close to the normal distribution, thus providing the regularity of the latent space

3.1.4 Decoder

The opposite process is done by the decoder. The decoder takes the *code* and expands it, using deconvolution layers, dense layers of growing size or whatever type and combination is fit for the application. After training, the decoder is essentially the ”builder” that we use to produce new instances of data. By inserting a small amount of data in the format of the code, we can create new samples drawn from the distribution of the original data. This opens up many new possibilities, for example it allows us to use encoders to encode completely different types of data, like words, and then use the decoder to create images based on these words.

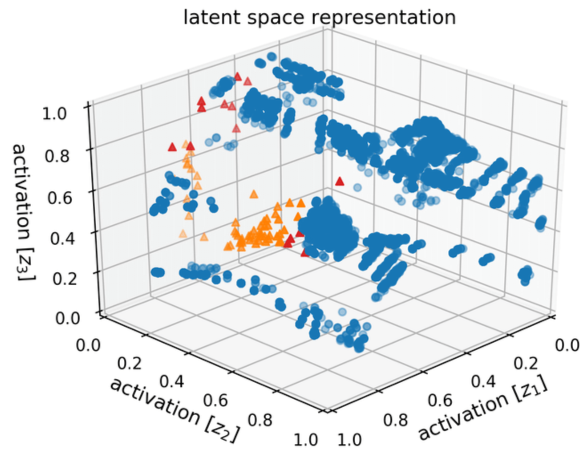


Figure 3.4: Latent space representation of a dataset after 2.000 training epochs. [57]

3.2 Development tools

In order to generate artificial data, we are going to build a Variational Autoencoder (VAE) using the Keras API. Keras is an open-source library developed by Francois Chollet, AI researcher at Google and is used for programming neural networks. It is the first high-level library added to the core of interface for Tensorflow. Keras uses high level commands in order to allow for easy modeling of deep neural networks and uses Python language. Other python libraries used are NumPy for arithmetic operations, Pandas framework for data matrices manipulation and matplotlib for data graphs. We are also provided with a dataset of 3 loads, that will be used for training the model.

(a) (b) (c)

Figure 3.5: Main development tools

3.3 Dataset Architecture

The dataset consists of the active and reactive power (in MW) of 3 different load types. The types are not specified, only numbered. For every load type we have a full year of samples with a sampling period of 15 minutes. That makes a total of $365 * 24 * 4 = 35,040$ measurements of active and $35,040$ measurements of reactive power for every type of load. Load 1 has a

maximum active power of $77MW$ and reactive power of $21MW$, Load 2 has a maximum active power of $116MW$ and reactive power of $15MW$ and Load 3 has a maximum active power of $76MW$ and reactive power of $17MW$

3.4 Model Architecture

The model we are building is going to follow the standard structure of a Variational Autoencoder. First we will import the data and reshape them into a convenient form. Then we are going to build the encoder, the sampling layer and the decoder. After the VAE has been built, we will train it, assess the training results and then use it for data generation. The general structure of the VAE is shown in figure 3.1. In the following subsections we will examine in detail every part of the model.

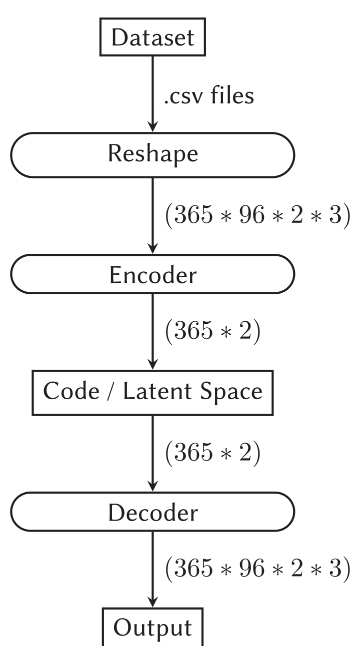


Figure 3.6: Model Architecture

3.4.1 Libraries dependencies and report file

First of all we need to install the dependencies of our code. The libraries that will be used are Numpy, for easier manipulation of large, multi-dimensional matrices, Pandas, for reading .csv files and constructing arrays, Tensorflow, which is a library for machine learning and artificial intelligence particularly focused on Deep Learning, Keras, which is a higher level interface for the Tensorflow library with some additional functionalities and Matplotlib, a well established plotting library for Python. A report .txt file will also be created, that will record the structure of the network and the training results after every run. This way we can keep track of the changes that we make in the structure of the model and the effect that these changes have in the training results.

3.4.2 Importing and Shaping the input data

First of all, the data available are imported as Pandas Dataframes, each from a different .csv file. The chosen period of our sampling is one day. With an original sampling frequency of 4 samples/hour, that makes a total of 96 samples per day for every type of power and every load. So the input data has to be reshaped into two-dimensional matrices, with a shape of $(365, 96)$ and each assigned to a different variable. The variables are shown below.

$$active_power1, reactive_power1$$
$$active_power2, reactive_power2$$
$$active_power3, reactive_power3$$

Next, the active and reactive power for each load are concatenated into a new matrix with an extra dimension, corresponding to the power type (active or reactive). This brings us to having 3 variables, each with a shape of $(365 * 96 * 2)$.

$$power1$$
$$power2$$
$$power3$$

The three loads are not independent of each other but correlated. They are part of a system and considering them independent would ignore that aspect and lose some real world characteristics of the loads, such as their synchonized seasonality highs and lows or weekdays and weekends. In order to preserve the above characteristics, they are again concatenated into a new variable, named $power$. The new variable has a shape of $(365, 96, 2, 3)$, with each dimension corresponding to $(day, sample, power\ type, load\ number)$. An intuitive way to understand the structure of the data, is that of an image. Each image has 96 2-color pixels (96 measurements of active and reactive power) and every day consists of 3 images (loads that are being monitored). The dataset consists of 365 such sets. Each day will be then passed on to the network for processing. In this way, the correlation of the loads remains intact, since every point of the latent space represents a 3-image snapshot.
After the data has been brought to the correct shape, they need to be normalized. This is a common practice in neural networks, since the activation functions of the neurons work better when presented with values between 0 and 1. For that reason, we find the biggest value of active and reactive power and divide respectively.
We now have brought our data into an easily usable and manipulated form

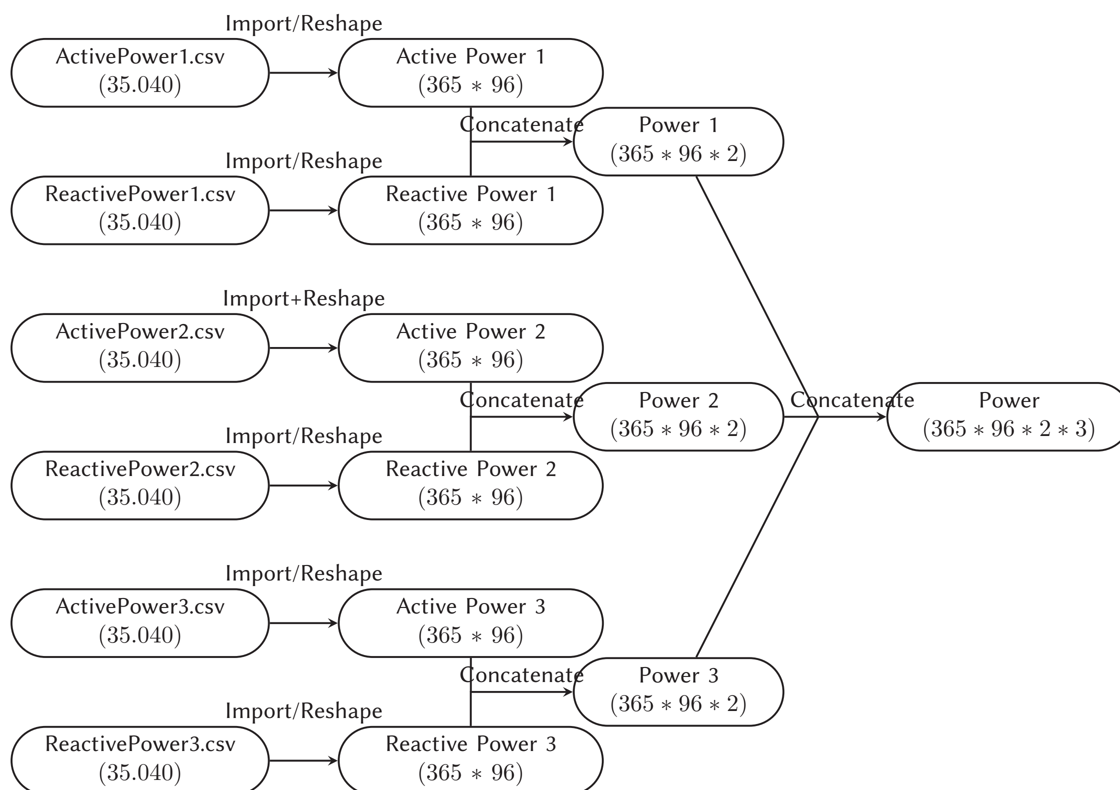


Figure 3.7: Importing and reshaping module

3.4.3 The Encoder

The encoder is the first building block of the actual neural network. Its job is to encode the data, reducing their dimensions to a predefined extent. This process, combined with the reconstruction and the error propagation that happens next, forces the encoder to keep only the important features of the input. The amount of reduction is a design choice, with too little reduction preventing the model from learning useful features since it can copy the input to the output and too much reduction can cause loss of information, thus making a proper reconstruction impossible. The encoder in this model, starts of course with an input layer of shape $(None, 96, 2, 3)$, where $None$ means it can be any number of samples, followed by consecutive blocks consisting of a 2D convolution layer, a batch normalization layer and an activation layer of Rectified Linear Units.
The first 2D convolution layer has a kernel and stride size of 1, meaning it just does a pass over the input data. The rest of the convolutional layers have a kernel and stride size of $(3, 1)$ and $(2, 1)$ respectively, meaning they determine every new point's value by examining itself and its two neighbours (left and right) and then moving two points to the right. So the convolution window does a pass over the active power values and then a pass over the reactive power values. This produces a new array of half the length (since it skips horizontally every other point) and equal height (since it does not skip vertically).

The batch normalization layer transforms the data of the batch so as to maintain the output close to 0 and the standard deviation close to 1. This helps the network maintain values that work well with the activation functions, avoiding (to extent that is possible) very large or very small values.
The ReLU layer is a simple activation layer, that keeps track of which neurons are activated together, to what extent and provides a number of weights available for adjustment.
The following layers pass the data into the z_mean and z_log_var ”variables” (which are actually layers) to be passed on to the sampling layer. Then the network is ”compiled” into a model named ”encoder”, that takes as input the ”encoder_inputs” and produces the encoded sample [z_mean, z_log_var, z].

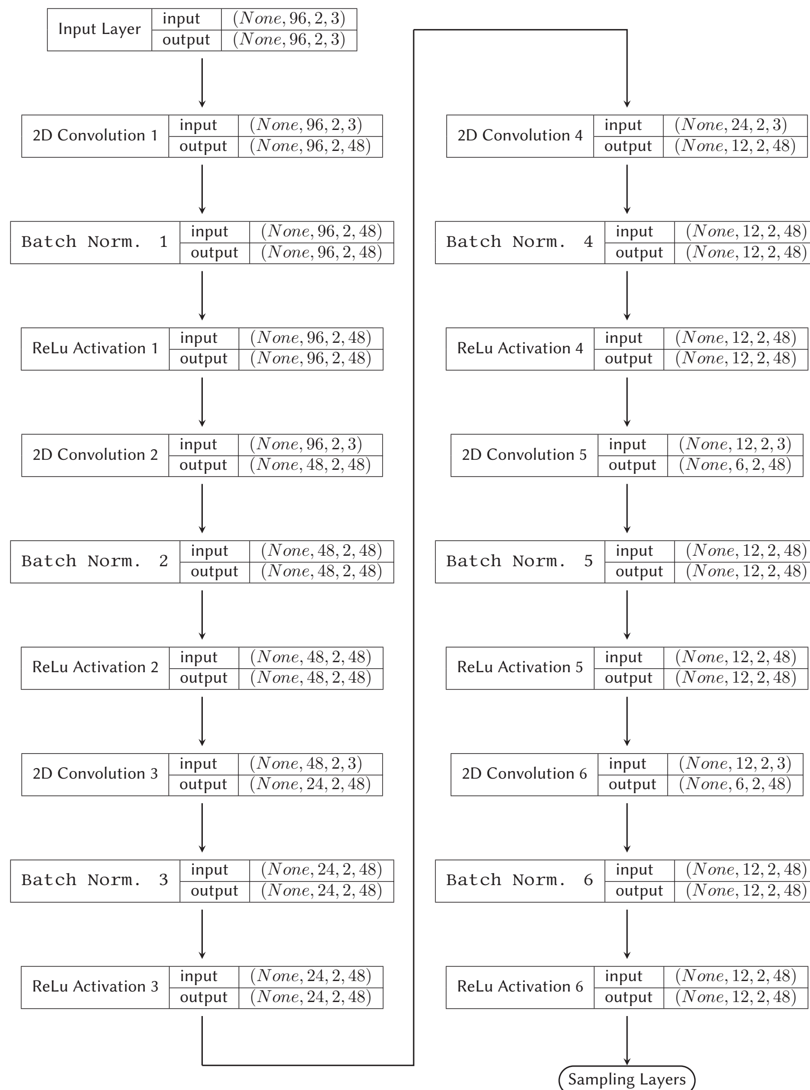


Figure 3.8: Encoder layers

3.4.4 Sampling layer

The sampling layer, as mentioned above, is the core component of the model. At this point, the data has a shape of $(None, 3, 2, 12)$, meaning that each day is represented by 12 filters, each with size $(3, 2)$, so a total of 72 numbers. In order to map this sample on the latent space as a normal distribution, we calculate the mean and the log variance of it. First we need to flatten it, using the layer provided by Keras for that reason, then pass it through a Dense layer which reduces the 72 numbers into 36 learning some characteristics of the flattened sample and then through the Z-mean and the Z-log-var layers that calculate the mean and the log variance of the sample, respectively. After that, the sampling layer takes a sample from the normal distribution created with the above values.

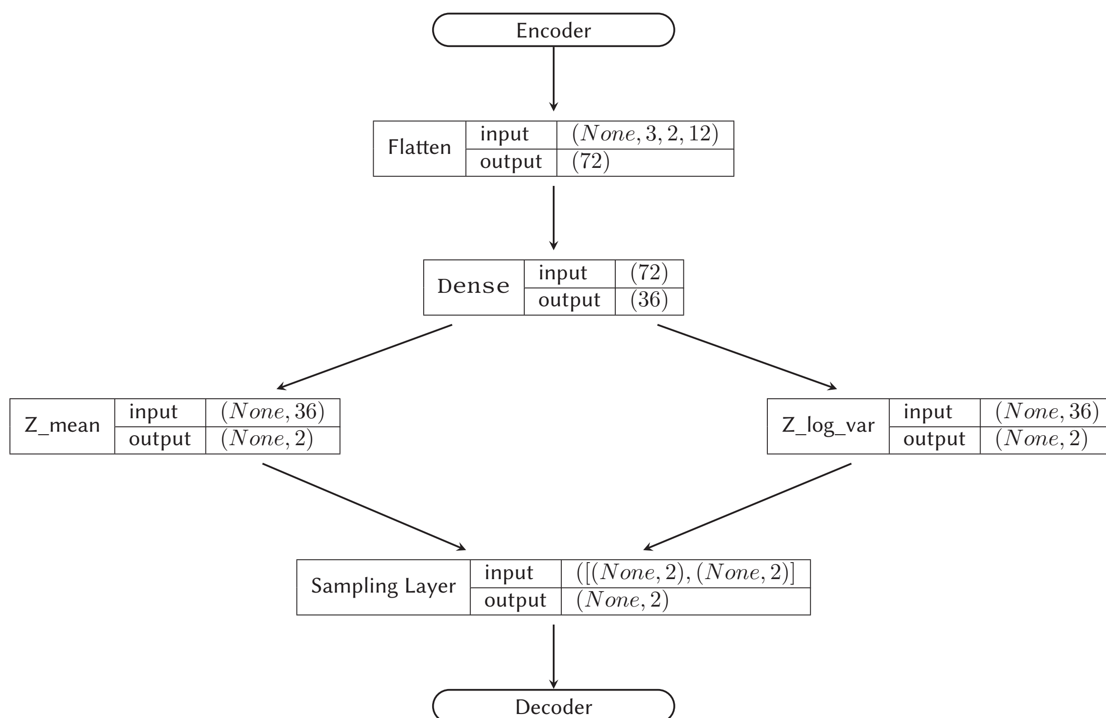


Figure 3.9: Sampling Layers

The sampling layer is probably the heart of the network and what turns an autoencoder into a variational autoencoder. As mentioned earlier in the literature review, in order for the autoencoder to be useful for data generation, we need to create a latent space that has some regularity. Small variations in the latent space should not result in hugely different generated samples or give meaningless results. In order to achieve this, the final step of the encoding process is not a sample (a set of points) but rather a normal distribution with a small randomness added to it.

3.4.5 The decoder

The decoder does the opposite from the encoder. It takes the encoded sample, and decodes it back to full dimensions. The goal of the decoder is to achieve a perfect reconstruction of the original sample. Depending on the application, different levels of precision are acceptable. The difference between the original sample and the decoded one, is called *reconstruction loss* and minimizing it is the main goal of the autoencoder training. However, we must remind that the main goal of this model is not the perfect reconstruction of samples, but the production of new data. A smaller reconstruction loss does not always mean better generated data. An example of perfect reconstruction is the exact copy of the input to the output and that of course would have no purpose. Training hyperparameters must be carefully considered in order to avoid overfitting and achieve the regularity of the latent space. The architecture of the decoder is almost like a mirrored version of the encoder. It has an input layer and a dense layer connected to the output of the sampling layer. The sample is reshaped into $(3, 2, 12)$ and the process of deconvolution begins.

3.4.6 Defining the Loss Function and the training Step

After creating the encoder, the decoder and the sampling layer, we need to define the losses that the model will monitor and try to minimize and also the training step. As mentioned above, each sample is reduced into just two numbers, reconstructed through the decoder and the compared with the original. The metric of the difference between, the $reconstruction error$ has to be explicitly declared when defining the model. We are going to use two types of losses, the binary crossentropy for evaluating the precision of the reconstruction sample by sample and the Kullback - Leibler divergence which is a type of statistical distance; it measures the distance between two probability distributions and is the most suitable for this application. The sum of these two is defined as *total loss* and is what our model tries to minimize during training

3.4.7 Training the model

Once all the other aspects of the model have been properly defined, we can proceed with the training.

First of all we have to define the *training step*. One complete training step is one cycle of encoding - sampling - decoding - calculating reconstruction loss - adjusting the layer weights, as seen in fig. 3.11.

Then we must define the *batch size*. The batch size is the number of samples (days, in this model) that are fed to the input of the network in every training step, and is an important hyperparameter. A very small batch size will cause the training to be slower. Also readjusting the weights every few samples, may prevent the Variational Autoencoder from learning the overall characteristics of the dataset, since it will struggle to adjust over a few specific samples every time, thus losing the big picture.In this application a batch size of 20 was found to be the most suitable

The other important hyperparameter of the training is the number of *epochs*. One epoch is one complete iteration over the whole dataset. Again, it takes careful consideration to set the proper number. As it is easily understood, too few epochs can stop the process early but too many epochs can cause overfitting, which means that although the loss has been minimized

the samples produced no longer represent real world data. The produced results where found to be the best for 80 epochs of training

The way that the weights are recalculated after every training step is set by the *optimizer*. The most well known optimizers are SGD (Stochastic Gradient Descent), RMSprop (Root Means Square Propagation) and Adam (Adaptive Movement Estimation) which is the one that we are going to use, since it delivered the best results in this application.

Another aspect of the training is the *callbacks*. Callbacks are some functions provided by the Keras interface and allow us to better monitor the process and the outcomes of the training. These include, for example, early stopping options when a certain target is reached, like some specific loss minimization, model checkpoint, which saves the weights during specified intervals throughout the training or CSV logger, which outputs certain aspects of the model in a .csv file. We are going to use the CSVlogger in order to output the architecture of the model, the training hyperparameters and the training results in a .csv file, every time we run the training process.

During the training process the model gets better at reconstructing the encoded data and so the latent space starts to represent accurately different waveforms that can be seen in out dataset. This advancement can be seen in fig. 3.12. However we can also notice that after 1000 epochs the latent space no longer represents real samples. That's because an excessive number of training iterations has resulted in overfitting.

It is also worth noting that the training process is stochastic and not deterministic. This means that every time we train the model the outcome is a little different. There have also been very few instances where the network didn't manage to train properly and learn the features of the data resulting in a very big reconstruction error that didn't reduce during the process and the output results where meaningless.

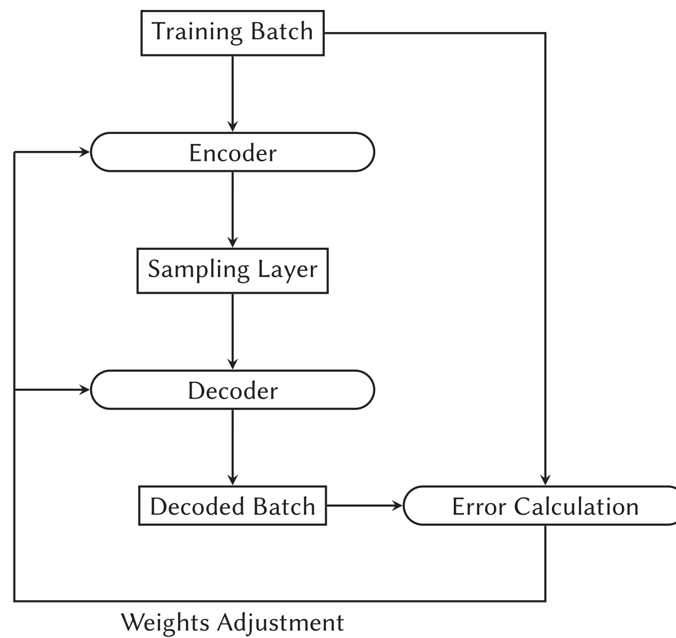


Figure 3.11: Training Process

3.5 Using the trained model

3.5.1 Data reconstruction

If the training process has gone well, we now have a network capable of taking as input real world load data samples with a shape of $(96 * 3 * 2)$, encoding each one of them into just 2 numbers and the decoding back to the initial dimensions, with proper reconstruction accuracy. This means that the model has learned the underlying features of the original data and that the data produced has preserved them to a very good extent. So now we have a whole new dataset, following the same distribution as the original but stripped of some very sample-specific characteristics that could be used in order to identify the specific load represented in a sample. Those could be the likes of a load shutdown, or some other events that through careful observation could result in someone matching load data considered to be anonymous at first sight, with specific consumers.

3.5.2 New Data Generation

During the encoding process, the original dataset of $(365*96*2*3)$ is being encoded into the *code*, a new array with a shape of $(365 * 2)$. This is the original dataset, mapped as 365 points onto the latent space. By decoding this code, we can take back the original dataset. However, the purpose of this thesis is also to produce new data, with some different characteristics but preserving the underlying features. Earlier, when describing the latent space, we mentioned that one of the reasons we use a *Variational* Autoencoder is the regularity of the latent space. This means that little variations in the coordinates of a coded sample, should result in small variations in the decoded data. This is the most important aspect of the model. By taking the

code produced by the original dataset and applying some transformations to it, we can create a new dataset with the same underlying features of the original but with some differentiated characteristics, with the most obvious being the mean value and the divergence. By decoding a canvas of points from the talent space, we can create a ”map”, a representation of the results produced by every point of the latent space and use that ”map” to navigate through it. We can pick any number of points from the latent space and create a whole new dataset with the desired characteristics. Or use the existing code of the dataset and shift it towards the proper part of the latent space in order to change some of its features at will, by adding for example a random normal distribution generator to it and creating a similar dataset but with lower peak or mean values, or greater divergence.

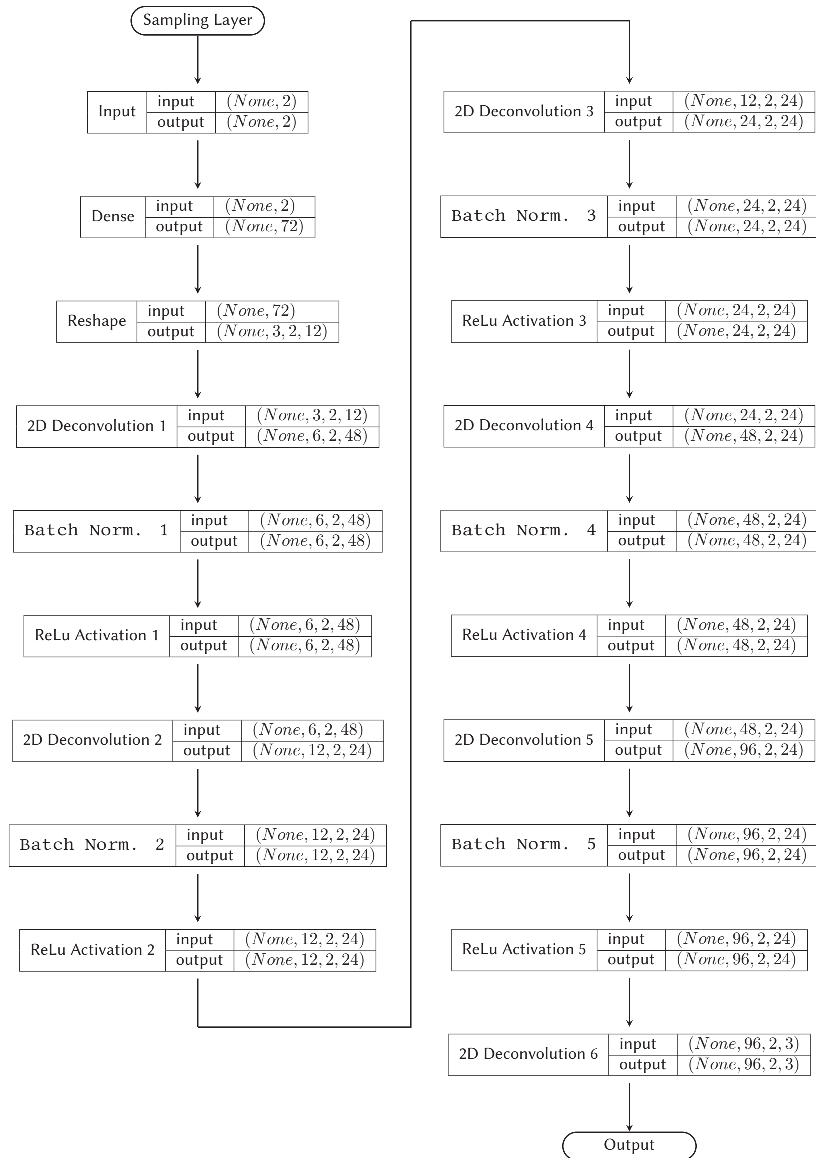
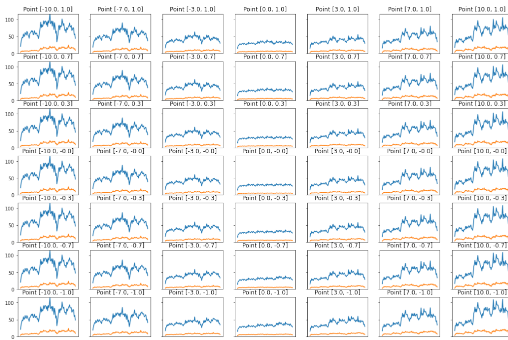
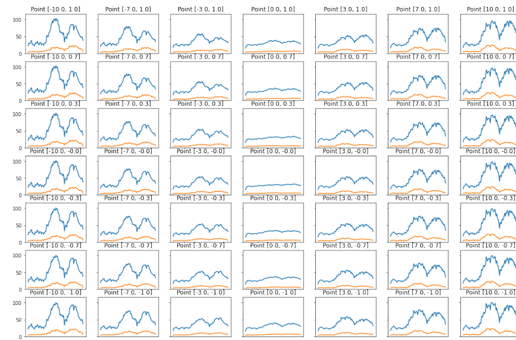


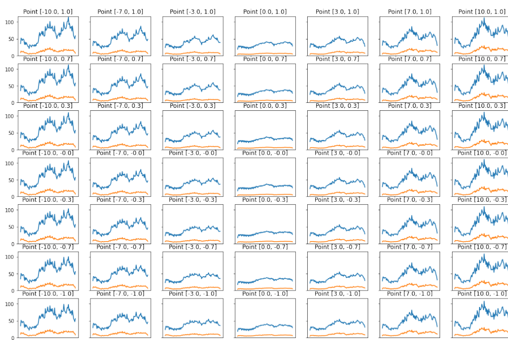
Figure 3.10: Decoder layers



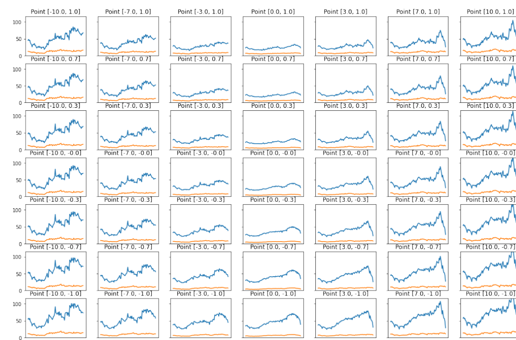
(a) 5 Training Epochs



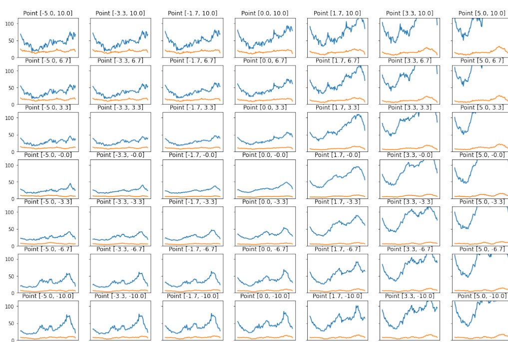
(b) 10 Training Epochs



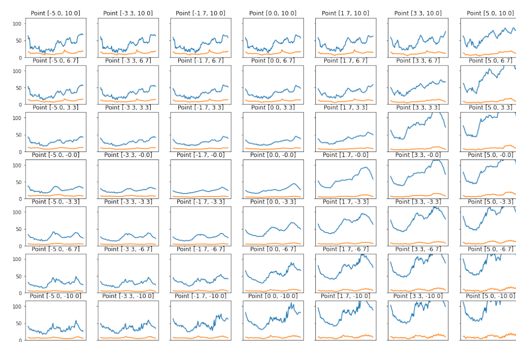
(c) 25 Training Epochs



(d) 40 Training Epochs

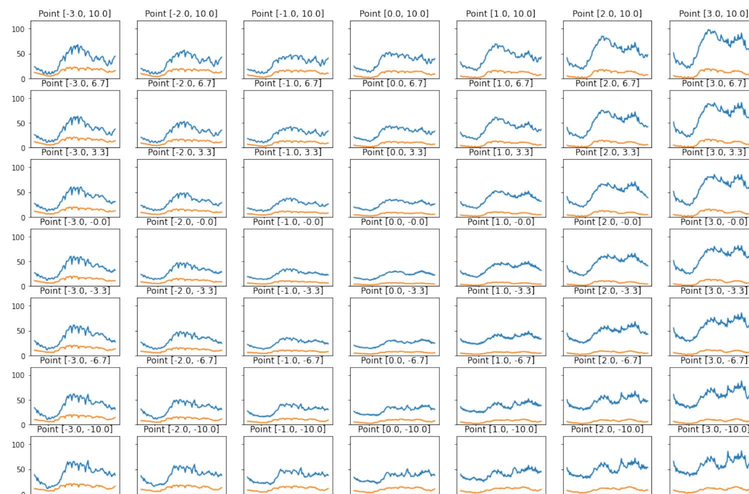


(e) 100 Training Epochs

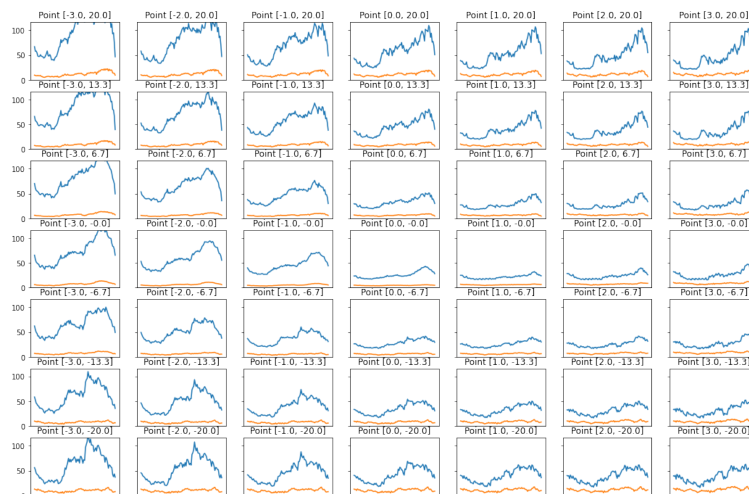


(f) 1000 Training Epochs

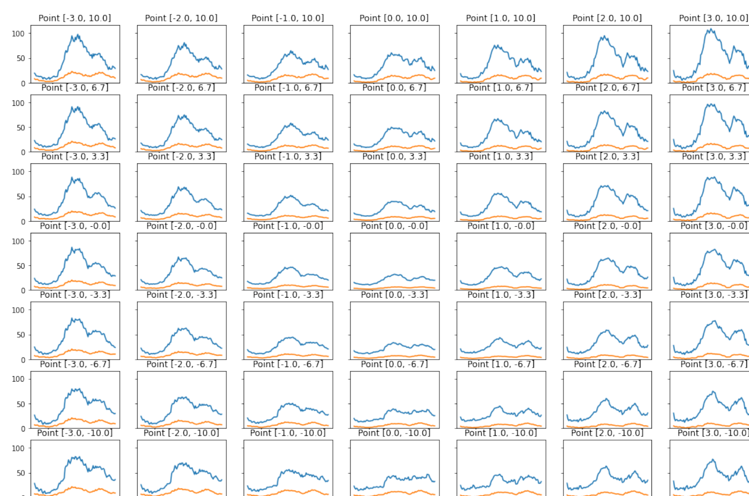
Figure 3.12: Mapping of the latent space after a different number of training epochs



(a) Latent space mapping of Load 1



(b) Latent space mapping of Load 2



(c) Latent space mapping of Load 3

Figure 3.13: Latent space mapping of the three loads

4 Case Study Results

4.1 Assumptions and Case Studies Description

The expected behaviour of the model is to train on the existing data and through the encoding-decoding process learn the important features of the dataset. This will give us the ability to do two things. First of all, encode and decode the dataset keeping its most important features intact. This can be used for eliminating some very load-specific information but keeping the useful information for the intended purpose. Secondly, the trained decoder will be able to decode any code that the user creates. This means that the user can create entire new datasets and using the mapping of the latent space, create any possible combination of days (3-load samples), suitable for the intended application.

In the following sections, we examine the quality of the latent space created, which is probably the most important factor of success for the purposes described above, the reconstruction capabilities of the network, through numerical and visual criteria and the generation capabilities by creating new codes and assessing the outcome.

4.2 Latent Space Evaluation

The core of the model and its most definitive part is the Latent Space. If the training has gone well, we should have a regular and complete latent space, with different sections of it representing different load profiles.Ideally, the different features represented in every section, will be those perceived as such by the human perception and are the most useful in terms of grid load data characteristics. That is not always the case with VAEs. If we look back at fig. 3.2(a) and fig. 3.2(b) that represent numerical digits (each with a different color) we can see two different organisations for the same data, one that has spatially separated the digits and one that has not.The latter, has organised the latent space in a way that the model understands, but is not useful for a human who would like to navigate the latent space and choose the features of the digit that he wants to generate. In fig. 3.13, we can see decoded samples from different parts of the latent space. It is pretty clear that the model has learnt the underlying characteristics of the input data and has mapped them on the latent space in a very convenient way. In fig. 3.13 we see a mapping of Load 1 samples and the part of the latent space that they come from. We can see that moving along the x-axis changes the number of highs and lows during the day, with negative values producing samples that have only one peak during the day, while positive values of x produce samples that have two distinctive peaks and the greater the value, the more distinctive those two become. The same goes for Load 2 and Load 3. This is a very good first indicator that we have a working model. If we set a very large number of epochs, after some time the latent space will stop producing good samples, despite the reconstruction error staying the same or even becoming less. This can be seen in fig. 3.12 where we observe how the produced results for Load 2 change as the number or training iterations increases.

4.3 Reconstruction Evaluation

The reconstruction of the dataset is one of the goals of this network. Evaluating this feature requires viewing it from different perspectives. As mentioned above, the reconstruction error itself and the exact reproduction of samples cannot be used as absolute metrics of success because this will miss the goal of anonymization and probably generation too. Instead, a series of metrics and visualizations will be used in order to show that the reconstructed dataset preserves the features of the original without being exactly the same.

Load 1			
Metric	Original Data	Reconstructed Data	Error
Maximum Active Power(MW)	77.78	72.48	-7.44%
Maximum Reactive Power (MVAr)	21.12	19.96	-5.79%
Average Active Power (MW)	27.68	27.75	0.27%
Average Reactive Power (MVAr)	6.36	6.51	2.21%
Active Energy Consumption (MWh)	$242,499.00$	$243,169.25$	0.28%
Reactive Energy Consumption (MVAr)	$55,733.75$	$56,995.25$	2.21%

Table 4.1: Load 1 Metrics

4.3.1 Numerical Metrics

First we are going to use some numerical metrics. When dealing with load data, one definitive aspect of the load is its peak power. Peak power is very important in grid analysis because it puts a lot of stress on the transmission lines and the generators. It is also a major factor of failures, grid reliability and pricing. In the original dataset, Load 1 has a maximum active power of 77.78 MW, Load 2 116.22 MW and Load 3 76.08 MW. In the reconstructed dataset those values are 72.48 MW, 112.08 MW and 73.37 MW respectively.

The second metric is the average power. Average power, in combination with an accurate maximum active power, can show an overall good reconstruction, meaning that the general shape of the reconstructed data follows that of the original. The results in average power are even better, with the error being less than 1% for the active power and less than 3% for the reactive power. An example of average power importance is when examining the effects of a random failure of a grid component.

Minimum power has not been taken into consideration because in every load there are spontaneous measurements that show no power. This could mean a grid outage, a failure on the consumer side or wrong data, so every load has a minimum value of 0. The network has no such prediction in itself and such incidents are erased when the dataset is reconstructed. That is one of the useful features for the anonymization of data. If the user wants and depending on the intended application, such points can be easily inserted at random after the data has been reconstructed.

The final metric is energy consumption throughout the year. It can be easily understood that it shows the same results as the average power, since one can be derived from the other by multiplying with 8,760. However it is presented here for easier comparison of the results.

Load 2			
Metric	Original Data	Reconstructed Data	Error
Maximum Active Power(MW)	116.22	112.08	-3.68%
Maximum Reactive Power (MVAr)	15.36	13.28	-15.60%
Average Active Power (MW)	33.21	33.47	0.79%
Average Reactive Power (MVAr)	5.95	6.11	2.57%
Active Energy Consumption (MWh)	$290,975.75$	$293,282.0$	0.79%
Reactive Energy Consumption (MVAr)	$52,187.25$	$53,562.75$	2.57%

Table 4.2: Load 2 Metrics

Load 3			
Metric	Original Data	Reconstructed Data	Error
Maximum Active Power(MW)	76.08	73.37	-3.69%
Maximum Reactive Power (MVAr)	17.04	19.96	2.24%
Average Active Power (MW)	25.15	25.31	0.61%
Average Reactive Power (MVAr)	5.41	5.47	1.06%
Active Energy Consumption (MWh)	$220,387.25$	$221,744.75$	0.61%
Reactive Energy Consumption (MVAr)	$47,441.0$	$47,953.25$	1.06%

Table 4.3: Load 3 Metrics

4.3.2 Visual Representations

An easy and accurate way to evaluate the reconstruction of the data is to plot side by side original samples and reconstructed. This way we can see if the reconstructed samples follow the shape of the original and not just the mean or the maximum value. In fig. 4.1 we see 3 random days (day 341, day 43, day 22) and their reconstruction, blue color for active and yellow color for reactive power. All plots have the same axis limits for easier comparison. As we can see, the reconstructed samples follow the shape of the original ones, which means that the network has actually learned their underlying features and is able to reproduce them. It also produces better results for the active than the reactive power and for days with higher load. We can also notice that the original samples have greater variation from one measurement to the other than the reconstructed ones and that the zero measurements in day 43 have been eliminated.

Another way to visualize the similarity of the two datasets is by plotting the heatmap of the entire 365 days. That's a way of showing all the 365 samples in one plot and see that the complete original and reconstructed datasets follow the same distribution. The darker the green color, the more often that value appears in each load and the y axis has been limited to include the most common values for better color perception, since the values vary a lot.

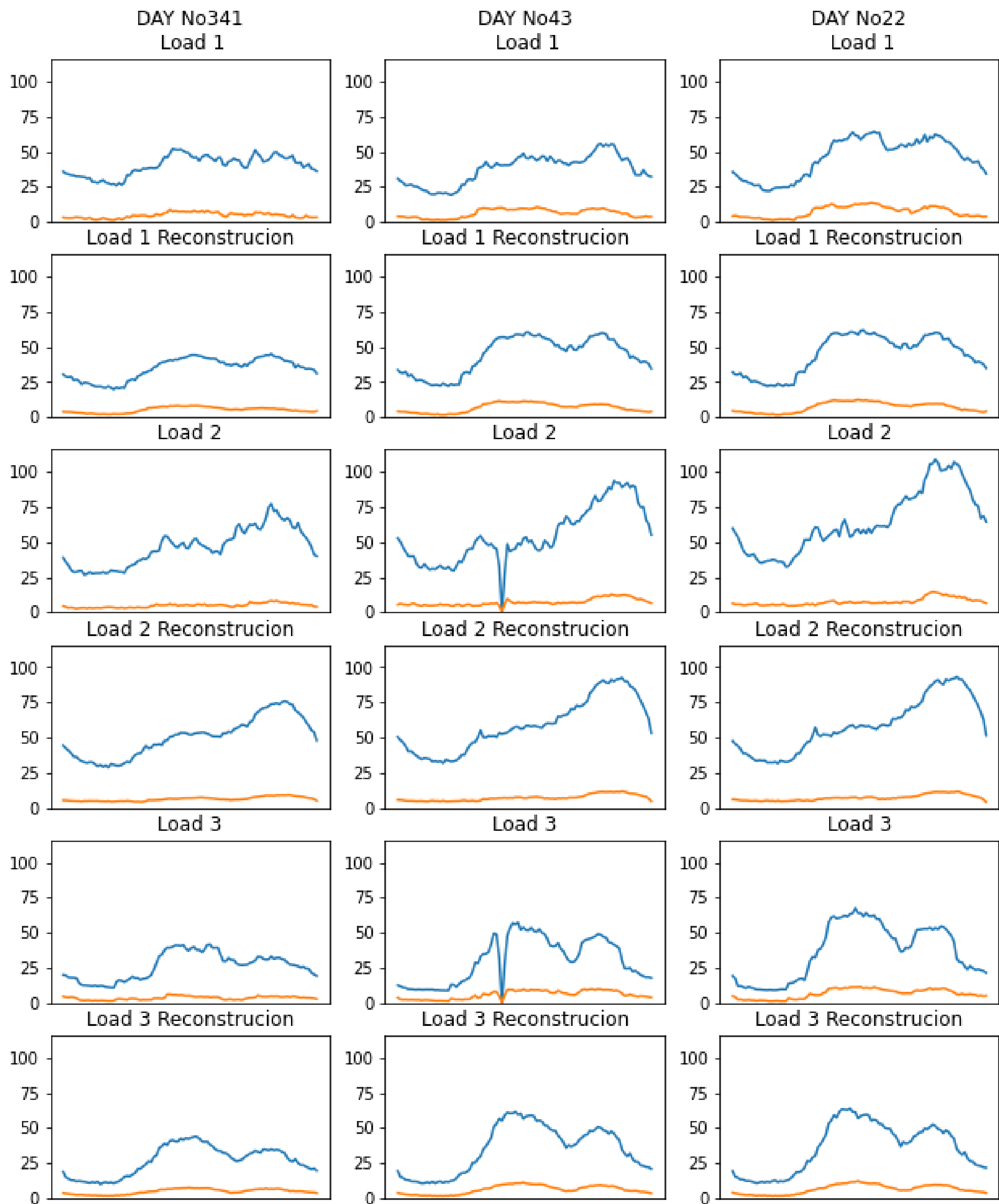


Figure 4.1: Reconstruction of 5 random days

Finally, we will plot a mapping of the original dataset and the reconstructed dataset on the latent space. Note that different parts of the latent space represent different features and

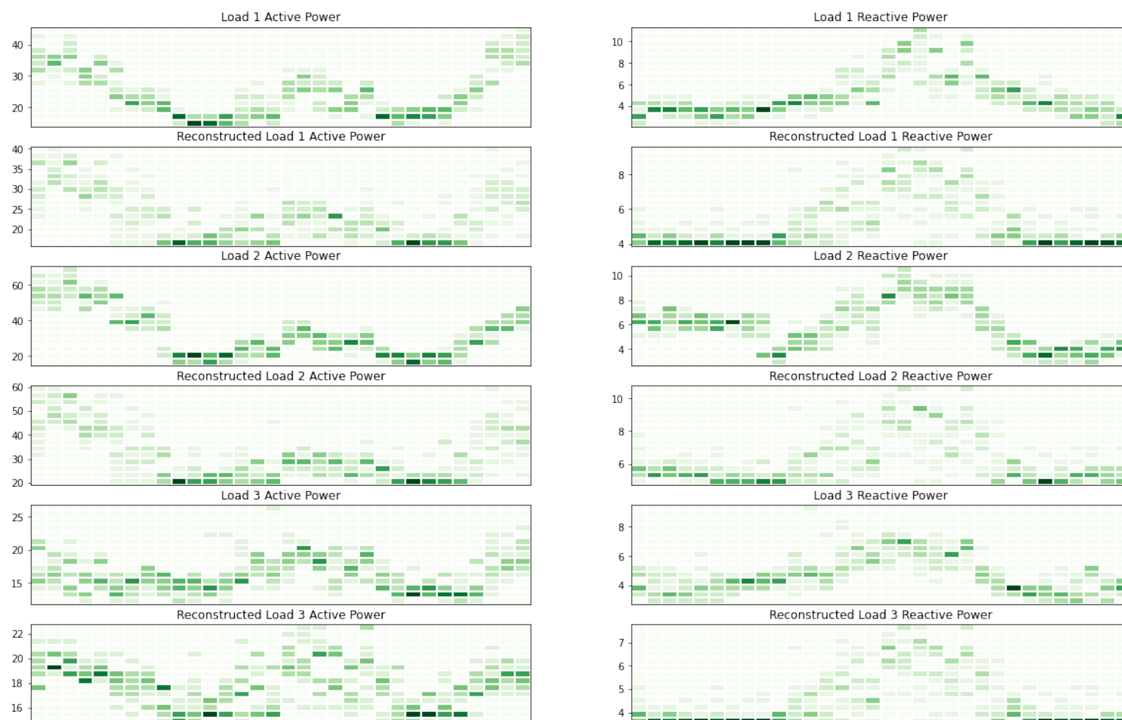
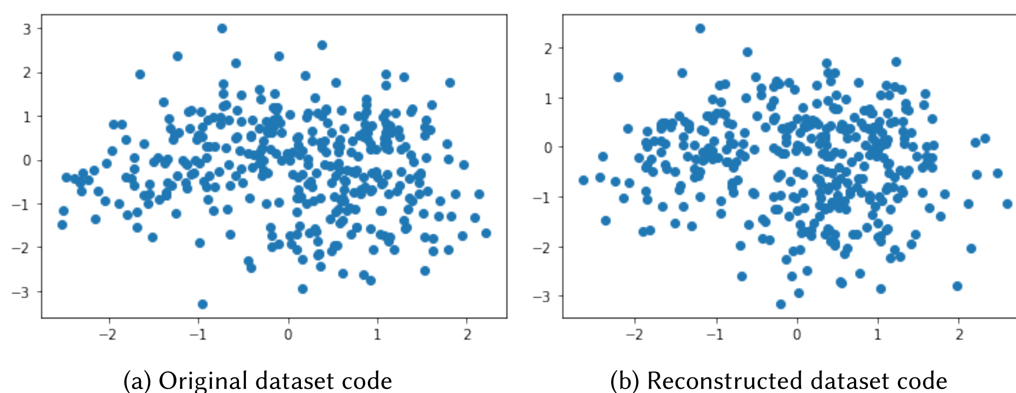


Figure 4.2: Heatmap of the original and the reconstructed dataset

we have already shown that the encoder does learn the important features of the input. If the original and the reconstructed data have similar properties, they should have the same mapping on the latent space. This is especially important, both for the current goal which is producing a similar dataset and the goal of producing an entirely new dataset. In fig. 4.3 we see a scatter plot of the 365 days of the original and the reconstructed set mapped on the latent space.



(a) Original dataset code (b) Reconstructed dataset code

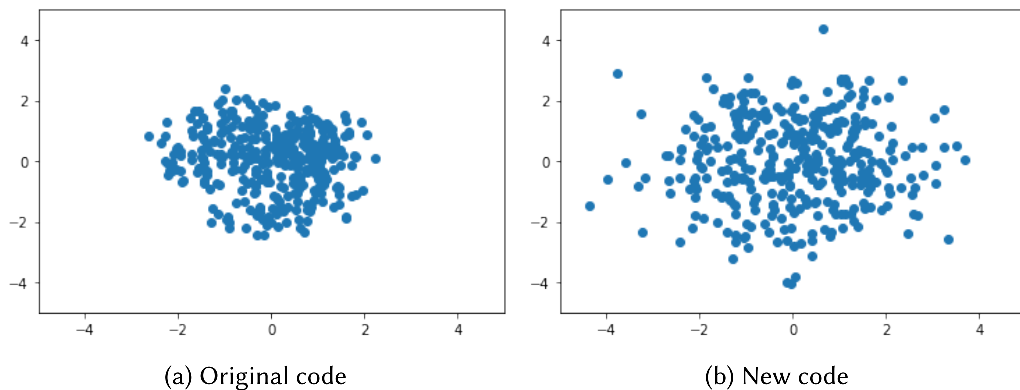
Figure 4.3: Mapping of the original and the reconstructed dataset on the latent space

4.4 Data Generation

The second goal was that the network is able to produce new but truthful data at will, with the user being able to chose their characteristics. As we have shown above, we can pick random points from the latent space and produce meaningful results. In order to create an entirely new dataset comprising of these three loads we can arbitrarily sample the latent space as many times as the days that we need. Having knowledge of the latent space will allow us to create datasets suitable for the specific application that we want. For example, we can create datasets with greater mean value, with greater variation within the day or both. We can also create datasets with great variation between consecutive days, with seasonality or without. Also there is no limit to the size of the dataset that we want to create. It could be decades long and also the decoding process is really fast and resource-efficient, meaning we could have many years of new data within minutes or even seconds.
Another very useful application and probably closer to the use in a real grid, is tweaking already existing data. By running the data through the encoder, we get an encoded version of them, which is called the code. In order to adjust some characteristics of it, we can modify the code in a certain way to fit our needs. This way, the reconstructed dataset will not be a copy of the original but a new version of it, with the main features kept intact but some others changed in a way that the new dataset fulfills the purpose for which we built it.

4.4.1 Random addition on the existing code

The possible methods to do this are endless. In this example we are going to use a normal distribution added to the original code, in order change the variance of it and then compare the original and the new dataset, both numerically and visually. The normal distribution that we are going to add on both the x and the y axis, has a mean value of 0 and variance equal to 2.



(a) Original code (b) New code

Figure 4.4: Mapping of the original and the new dataset on the latent space

The results on the code are shown in fig. 4.4. We see the code of the original dataset and the same code after the normal distribution randomization has been applied to it. The points on the latent space have been spread out.By decoding the newly generated code in fig. 4.4b we get

a new dataset and using the mapping of the latent space shown in fig. 3.13, we can expect it to have higher maximum power, higher average power and higher total energy consumption.

Load 1

Metric	Original Data	Generated Data	Variation
Maximum Active Power(MW)	77.78	93.81	16.49%
Maximum Reactive Power (MVAr)	21.12	21.89	3.52%
Average Active Power (MW)	27.68	28.73	3.67%
Average Reactive Power (MVAr)	6.36	6.62	3.96%
Active Energy Consumption (MWh)	242, 499.00	251, 745.50	3.67%
Reactive Energy Consumption (MVAr)	55, 733.75	58, 033.75	3.96%

Table 4.4: Load 1 Metrics

Load 2

Metric	Original Data	Generated Data	Variation
Maximum Active Power(MW)	116.22	142.69	18.55%
Maximum Reactive Power (MVAr)	15.36	16.35	5.31%
Average Active Power (MW)	33.21	35.07	5.30%
Average Reactive Power (MVAr)	5.96	6.10	2.38%
Active Energy Consumption (MWh)	290, 975.75	307, 281.25	5.31%
Reactive Energy Consumption (MVAr)	52, 187.25	53, 460.25	2.38%

Table 4.5: Load 2 Metrics

Load 3

Metric	Original Data	Reconstructed Data	Variation
Maximum Active Power(MW)	76.08	97.91	22.30%
Maximum Reactive Power (MVAr)	17.04	21.89	16.91%
Average Active Power (MW)	25.15	26.51	5.09%
Average Reactive Power (MVAr)	5.41	5.67	4.56%
Active Energy Consumption (MWh)	220, 387.25	232, 211.75	5.09%
Reactive Energy Consumption (MVAr)	47, 441.0	49, 709.25	4.56%

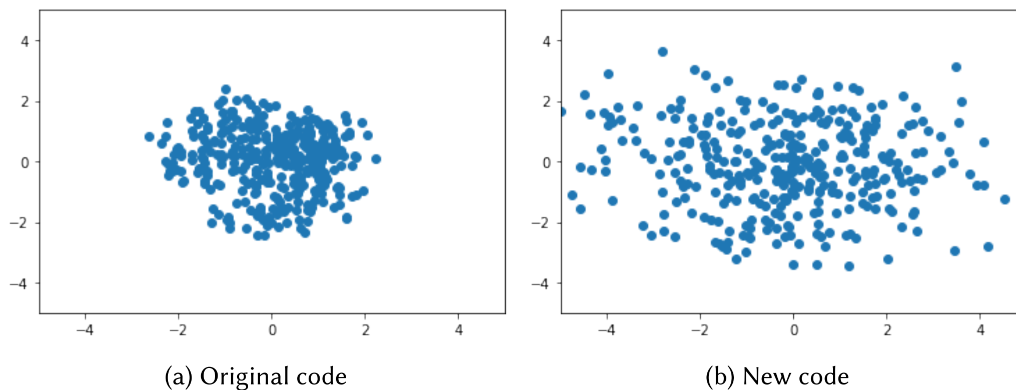
Table 4.6: Load 3 Metrics

The maximum and average power (both active and reactive) and total energy consumption of the newly generated dataset are shown in tables 4.4 to 4.6. As expected, the maximum value has been increased, as some samples have moved further away from the center of the latent space. However, the average power (and the total energy as a consequence) have not changed as mush. That is because we set the mean value of the normal distribution to be 0.
The next step will be to plot 3 random days and their reconstruction after the adjustment of the code.

As we can see in fig. 4.5, loads of Day 59 have been reduced, those of Day 214 have been increased and those of Day 352 have also been decreased. However, all 3 loads have kept their basic features and the correlation between them, which are the two very important factors regarding the integration of a load into a grid.

4.4.2 Random multiplication on the existing code

In the second example we are going to alter the existing dataset by multiplying the code with a random normal distribution. The distribution has a mean value of 1.7 and variance equal to 0.3. By applying this transformation to the existing encoded set, we get the code seen in fig. 4.6b



(a) Original code (b) New code

Figure 4.6: Plot of the original and the new code on the latent space

The new code is mode widely spread out across the latent space and from the mapping seen in fig. 3.13, we can predict that the new dataset will have greater peak values but also greater average power and energy consumption. By multiplying the existing code with values from 1.4 to 2, all samples are expected to be of equal or greater average and maximum value. That can be seen in tables 4.7 to 4.9.

Load 1			
Metric	Original Data	Generated Data	Variation
Maximum Active Power(MW)	77.78	114.46	46.79%
Maximum Reactive Power (MVAr)	21.12	32.93	55.92%
Average Active Power (MW)	27.68	33.82	22.19%
Average Reactive Power (MVAr)	6.36	7.83	23.05%
Active Energy Consumption (MWh)	242,499.00	296,304.50	22.19%
Reactive Energy Consumption (MVAr)	55,733.75	68,582.25	23.05%

Table 4.7: Load 1 Metrics

Load 2			
Metric	Original Data	Generated Data	Variation
Maximum Active Power(MW)	116.22	150.97	29.90%
Maximum Reactive Power (MVAr)	15.36	19.51	27.04%
Average Active Power (MW)	33.21	37.16	11.89%
Average Reactive Power (MVAr)	5.96	6.67	11.90%
Active Energy Consumption (MWh)	290, 975.75	325, 569.25	11.89%
Reactive Energy Consumption (MVAr)	52, 187.25	58, 397.75	11.90%

Table 4.8: Load 2 Metrics

Load 3			
Metric	Original Data	Reconstructed Data	Variation
Maximum Active Power(MW)	76.08	119.82	57.49%
Maximum Reactive Power (MVAr)	17.04	29.43	72.77%
Average Active Power (MW)	25.15	29.87	18.72%
Average Reactive Power (MVAr)	5.41	6.66	22.92%
Active Energy Consumption (MWh)	220, 387.25	261, 636.75	18.72%
Reactive Energy Consumption (MVAr)	47, 441.0	58, 314.25	29.92%

Table 4.9: Load 3 Metrics

It is worth noticing that by applying this transformation on the code, apart from the maximum power, the average power and therefore the total energy consumption have also been increased by a noticeable amount, in contrast to the previous example where the random distribution was added to the code. This shows that different aspects of the dataset can be manipulated by trying different kinds of modifications to the code, in a random or a predetermined way.

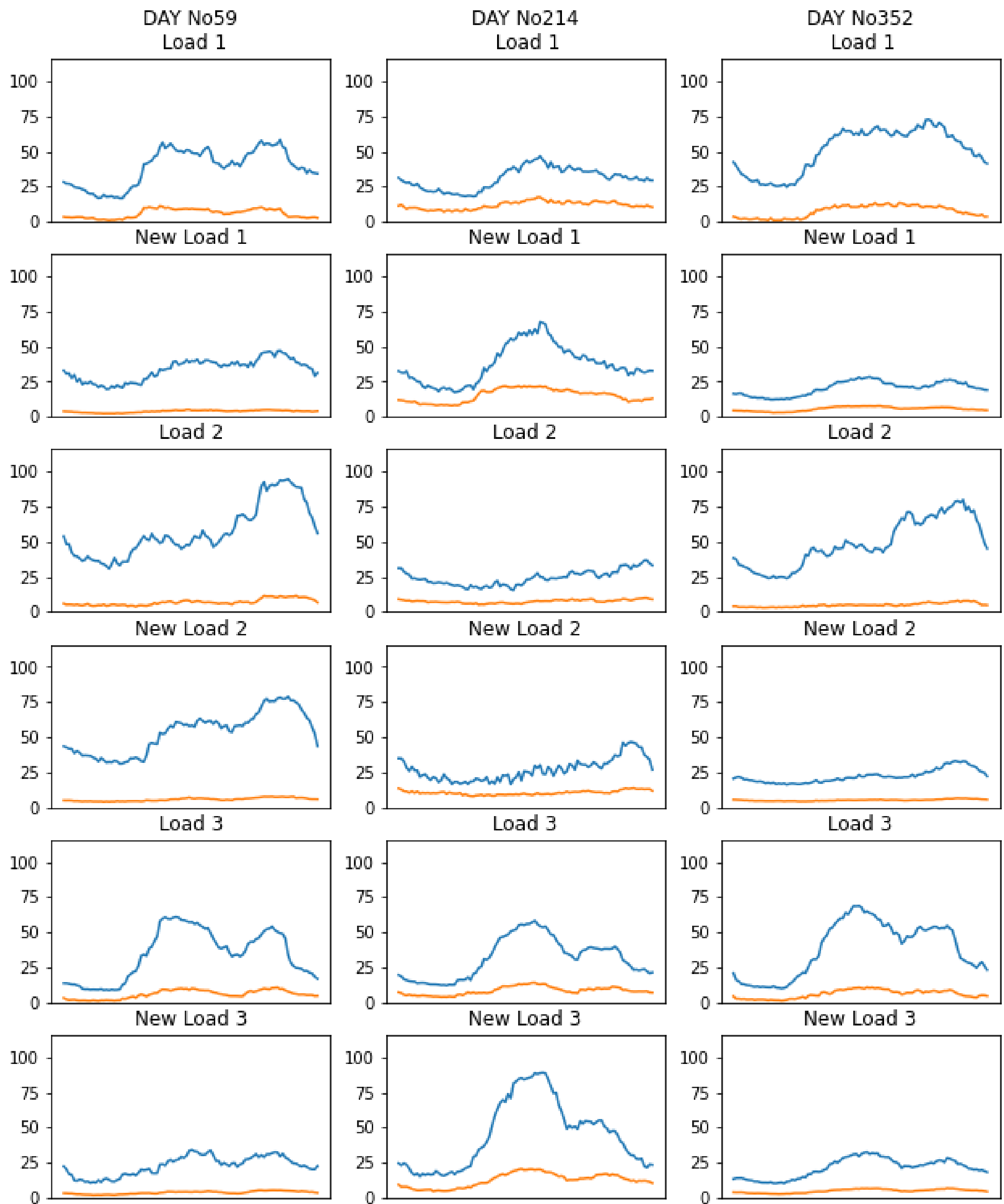


Figure 4.5: Random days generated by adding to the original encoded set

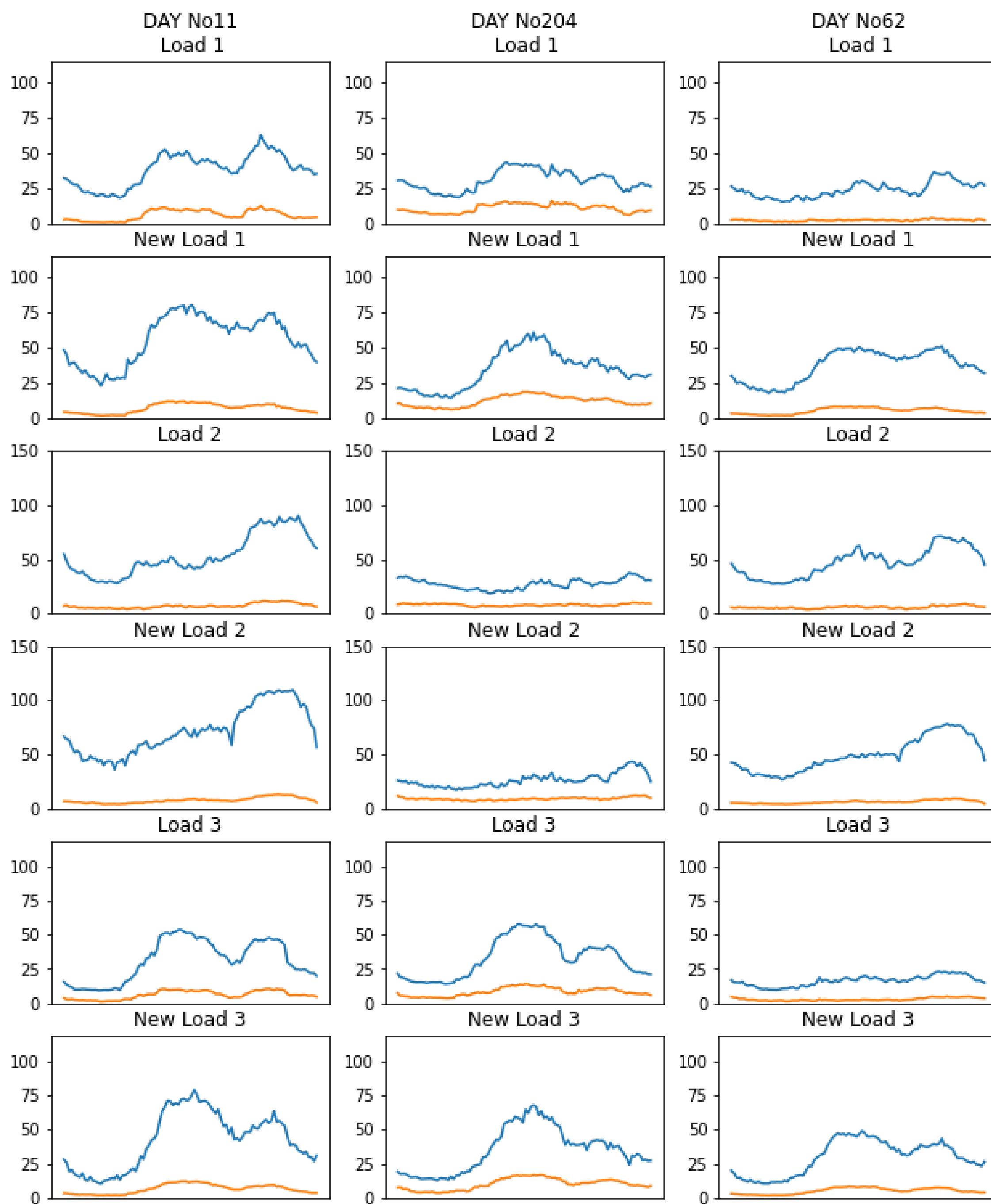


Figure 4.7: Random days generated by multiplying the original encoded set

5 Conclusions

5.1 Methodology and Results

In the age of data science and the rapid development of smart grids, demand for grid data is only going to increase. However acquiring and sharing such data can be a difficult thing, due to the cost of applying extensive measurements on grid level and the sensitivity of the data. In this thesis we gave an example of how neural networks and Variational Autoencoders specifically, can prove to be useful in the generation of new data when only a limited amount of training data is available for training.
The network managed to learn the underlying features of the data and use them for two reasons: reconstruct the data in order to hide potentially identifying features of them and also create a latent space that can be sampled and produce entirely new datasets that still follow the original distribution but their features can be adjusted at will. Both of these functionalities require very few resources (training the model for 100 epochs takes less than a minute using Google Colaboratory) and producing a new dataset consisting of 3 Loads for 365 days takes less than a second. The number of samples available for every load (365) is considered minuscule in the field of neural networks, however the model still managed to train properly. This is one of the advantages of Variational Autoencoders. The quality of the output is satisfactory as it has been shown in Chapter 4. It built a latent space with distinctive characteristics and easily usable for data generation.

5.2 Future Work

This network was built as a general proof-of-concept application. In case we need data for a very specific use, different layers can be used in order to focus on the features that are especially important for the case, like peak values, intraday variations, load correlation or power factor. It can also be easily expanded to include a greater numbers of loads. All the above can prove very useful for researchers running extended simulations of grids and power flow analysis.
Another interesting possibility for further research could be the incorporation of the grid simulation or the power flow analysis into the model itself (probably the error function). In this way, the model can be trained not on producing just the load data that we know we want, but producing the *grid state* that we want (and maybe don't know the load conditions that create it), thus building a latent space representation of many possible grid states that we can easily navigate in order to find optimal or risky conditions.

Bibliography

[1] Panagiotis Pediaditis, Charalampos Ziras, Junjie Hu, Shi You, and Nikos Hatziargyriou. "Decentralized DLMPs with synergetic resource optimization and convergence acceleration". In: *Electric Power Systems Research* 187 (2020), p. 106467. DOI: https://doi.org/10.1016/j.epsr.2020.106467.

[2] Panagiotis Pediaditis, Dimitrios Papadaskalopoulos, Nikos Hatziargyriou, and Dušan Prešić. "Cross-border Shared Sizing of Frequency Restoration Reserves: Insights from the H2020 CROSSBOW Project". In: *2021 International Conference on Smart Energy Systems and Technologies (SEST)*. IEEE. 2021, pp. 1–6. DOI: https://doi.org/10.1109/SEST50973.2021.9543208.

[3] Panagiotis Pediaditis, Katja Sirviö, Charalampos Ziras, Kimmo Kauhaniemi, Hannu Laaksonen, and Nikos Hatziargyriou. "Compliance of Distribution System Reactive Flows with Transmission System Requirements". In: *Applied Sciences* 11.16 (2021), p. 7719. DOI: https://doi.org/10.3390/app11167719.

[4] Panagiotis Pediaditis, Dimitrios Papadaskalopoulos, Anthony Papavasiliou, and Nikos Hatziargyriou. "Bilevel optimization model for the design of distribution use-of-system tariffs". In: *IEEE Access* 9 (2021), pp. 132928–132939. DOI: https://doi.org/10.1109/ACCESS.2021.3114768.

[5] Panagiotis Pediaditis, Charalampos Ziras, Dimitrios Papadaskalopoulos, and Nikos Hatziargyriou. "Synergies between Distribution Use-of-System Tariffs and Local Flexibility Markets". In: *2022 International Conference on Smart Energy Systems and Technologies (SEST)*. 2022.

[6] Luc Rocher, Julien M. Hendrickx, and Yves-Alexandre de Montjoye. "Estimating the success of re-identifications in incomplete datasets using generative models". In: *Nature Communications* 10.1 (July 2019), p. 3069. ISSN: 2041-1723. DOI: 10.1038/s41467-019-10933-3. URL: https://doi.org/10.1038/s41467-019-10933-3.

[7] P Ohm. *Broken Promises of Privacy: Responding to the Surprising Failure of Anonymization'57 UCLA LAW REVIEW 1701 (2010) 1701-1711.* 2010.

[8] A.D. Papalexopoulos and T.C. Hesterberg. "A regression-based approach to short-term system load forecasting". In: *IEEE Transactions on Power Systems* 5.4 (1990), pp. 1535–1547. DOI: 10.1109/59.99410.

[9] Srinivas Varadan and Elham B. Makram. "Harmonic load identification and determination of load composition using a least squares method". In: *Electric Power Systems Research* 37.3 (1996), pp. 203–208. ISSN: 0378-7796. DOI: https://doi.org/10.1016/S0378-7796(96)01059-0. URL: https://www.sciencedirect.com/science/article/pii/S0378779696010590.

[10] C. Chatfield. "The Holt-Winters Forecasting Procedure". In: *Journal of the Royal Statistical Society. Series C (Applied Statistics)* 27.3 (1978), pp. 264–279. ISSN: 00359254, 14679876. URL: http://www.jstor.org/stable/2347162 (visited on 06/15/2022).

[11] R.P. Broadwater, A. Sargent, A. Yarali, H.E. Shaalan, and J. Nazarko. “Estimating substation peaks from load research data”. In: *IEEE Transactions on Power Delivery* 12.1 (1997), pp. 451–456. DOI: 10.1109/61.568270.

[12] A.A. El-Keib, X. Ma, and H. Ma. “Advancement of statistical based modeling techniques for short-term load forecasting”. In: *Electric Power Systems Research* 35.1 (1995), pp. 51–58. ISSN: 0378-7796. DOI: https://doi.org/10.1016/0378-7796(95)00987-6. URL: https://www.sciencedirect.com/science/article/pii/0378779695009876.

[13] G.A.N. Mbamalu and M.E. El-Hawary. “Load forecasting via suboptimal seasonal autoregressive models and iteratively reweighted least squares estimation”. In: *IEEE Transactions on Power Systems* 8.1 (1993), pp. 343–348. DOI: 10.1109/59.221222.

[14] Q.-C. Lu, W.M. Grady, M.M. Crawford, and G.M. Anderson. “An adaptive nonlinear predictor with orthogonal escalator structure for short-term load forecasting”. In: *IEEE Transactions on Power Systems* 4.1 (1989), pp. 158–164. DOI: 10.1109/59.32473.

[15] W.M. Grady, L.A. Groce, T.M. Huebner, Q.C. Lu, and M.M. Crawford. “Enhancement, implementation, and performance of an adaptive short-term load forecasting algorithm”. In: *IEEE Transactions on Power Systems* 6.4 (1991), pp. 1404–1410. DOI: 10.1109/59.116982.

[16] S.A. Al Rashed. “New model for peak demand forecasting applied to highly complex load characteristics of a fast developing area”. English. In: *IEE Proceedings C (Generation, Transmission and Distribution)* 139 (2 Mar. 1992), 136–140(4). ISSN: 0143-7046. URL: https://digital-library.theiet.org/content/journals/10.1049/ip-c.1992.0022.

[17] E.H. Barakat, M.A. Qayyum, M.N. Hamed, and S.A. Al Rashed. “Short-term peak demand forecasting in fast developing utility with inherit dynamic load characteristics. I. Application of classical time-series methods. II. Improved modelling of system dynamic load characteristics”. In: *IEEE Transactions on Power Systems* 5.3 (1990), pp. 813–824. DOI: 10.1109/59.65910.

[18] Bo-Juen Chen, Ming-Wei Chang, and Chih-Jen lin. “Load forecasting using support vector Machines: a study on EUNITE competition 2001”. In: *IEEE Transactions on Power Systems* 19.4 (2004), pp. 1821–1830. DOI: 10.1109/TPWRS.2004.835679.

[19] Francis E.H. Tay and L.J. Cao. “Modified support vector machines in financial time series forecasting”. In: *Neurocomputing* 48.1 (2002), pp. 847–861. ISSN: 0925-2312. DOI: https://doi.org/10.1016/S0925-2312(01)00676-2. URL: https://www.sciencedirect.com/science/article/pii/S0925231201006762.

[20] Abdel Ghani Aissaoui, Mohamed Abid, Hamza Abid, Ahmed Tahour, and Abdel Kader Zeblah. “A fuzzy logic controller for synchronous machine”. In: *JOURNAL OF ELECTRICAL ENGINEERING-BRATISLAVA-* 58.5 (2007), p. 285.

[21] K.-L. Ho. “Fuzzy expert systems: an application to short-term load forecasting”. English. In: *IEE Proceedings C (Generation, Transmission and Distribution)* 139 (6 Nov. 1992), 471–477(6). ISSN: 0143-7046. URL: https://digital-library.theiet.org/content/journals/10.1049/ip-c.1992.0066.

[22] H-C Wu and C-N Lu. “Automatic fuzzy model identification for short-term load forecast”. In: *IEE Proceedings-Generation, Transmission and Distribution* 146.5 (1999), pp. 477–482.

[23] Hiroyuki Mori, Yasuyuki Sone, Daisuke Moridera, and Toru Kondo. “Fuzzy inference models for short-term load forecasting with tabu search”. In: *IEEE SMC’99 Conference Proceedings. 1999 IEEE International Conference on Systems, Man, and Cybernetics (Cat. No. 99CH37028).* Vol. 6. IEEE. 1999, pp. 551–556.
[24] Mo-yuen Chow and Hahn Tram. “Application of fuzzy logic technology for spatial load forecasting”. In: *Proceedings of 1996 Transmission and Distribution Conference and Exposition.* IEEE. 1996, pp. 608–614.
[25] Mo-yuen Chow, Jinxiang Zhu, and Hahn Tram. “Application of fuzzy multi-objective decision making in spatial load forecasting”. In: *IEEE Transactions on Power Systems* 13.3 (1998), pp. 1185–1190.
[26] Tomonobu Senjyu, Shuzo Higa, and K Uezato. “Future load curve shaping based on similarity using fuzzy logic approach”. In: *IEE Proceedings-Generation, Transmission and Distribution* 145.4 (1998), pp. 375–380.
[27] P. K. DASH, A. C. LEIW, and S. RAHMAN. “A comparison of fuzzy neural networks for the generation of daily average and peak load profiles”. In: *International Journal of Systems Science* 26.11 (1995), pp. 2091–2106. DOI: 10.1080/00207729508929156.
[28] P.K. Dash, A.C. Liew, and Saifur Rahman. “Fuzzy neural network and fuzzy expert system for load forecasting”. In: *Generation, Transmission and Distribution, IEE Proceedings-* 143 (Feb. 1996), pp. 106–114. DOI: 10.1049/ip-gtd:19960314.
[29] Anna Förster. “Teaching Networks How To Learn: Machine Learning for Data Dissemination in Wireless Sensor Networks”. PhD thesis. Jan. 2009.
[30] Hong-Tzer Yang, Chao-Ming Huang, and Ching-Lien Huang. “Identification of ARMAX model for short term load forecasting: an evolutionary programming approach”. In: *Proceedings of Power Industry Computer Applications Conference.* IEEE. 1995, pp. 325–330.
[31] X Ma, AA El-Keib, RE Smith, and H Ma. “A genetic algorithm based approach to thermal unit commitment of electric power systems”. In: *Electric Power Systems Research* 34.1 (1995), pp. 29–36.
[32] Dong Gyu Lee, Byong Whi Lee, and Soon Heung Chang. “Genetic programming model for long-term forecasting of electric power demand”. In: *Electric power systems research* 40.1 (1997), pp. 17–22.
[33] P Schuster. “Effects of finite population size and other stochastic phenomena in molecular evolution”. In: *Complex Systems—Operational Approaches in Neurobiology, Physics, and Computers.* Springer, 1985, pp. 16–35.
[34] Neelarghya. *Reinforcement learning vs genetic algorithm-AI for simulations.* July 2021. URL: https://medium.com/xrpractices/reinforcement-learning-vs-genetic-algorithm-ai-for-simulations-f1f484969c56.
[35] Dong C Park, MA El-Sharkawi, RJ Marks, LE Atlas, and MJ Damborg. “Electric load forecasting using an artificial neural network”. In: *IEEE transactions on Power Systems* 6.2 (1991), pp. 442–449.
[36] Arjun Baliyan, Kumar Gaurav, and Sudhansu Kumar Mishra. “A Review of Short Term Load Forecasting using Artificial Neural Network Models”. In: *Procedia Computer Science* 48 (2015). International Conference on Computer, Communication and Conver-

gence (ICCC 2015), pp. 121–125. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2015.04.160. URL: https://www.sciencedirect.com/science/article/pii/S1877050915006699.

[37] Sam Bond-Taylor, Adam Leach, Yang Long, and Chris G. Willcocks. *Deep Generative Modelling: A Comparative Review of VAEs, GANs, Normalizing Flows, Energy-Based and Autoregressive Models*. 2021. arXiv: 2103.04922 [cs.LG].

[38] John J Hopfield. "Hopfield network". In: *Scholarpedia* 2.5 (2007), p. 1977.

[39] Geoffrey E Hinton. "Boltzmann machine". In: *Scholarpedia* 2.5 (2007), p. 1668.

[40] Geoffrey E Hinton. "Deep belief networks". In: *Scholarpedia* 4.5 (2009), p. 5947.

[41] Peter Dayan, Geoffrey E Hinton, Radford M Neal, and Richard S Zemel. "The helmholtz machine". In: *Neural computation* 7.5 (1995), pp. 889–904.

[42] Shizuo Kaji and Satoshi Kida. *Overview of image-to-image translation by use of deep neural networks: denoising, super-resolution, modality conversion, and reconstruction in medical imaging*. 2019. arXiv: 1905.08603 [physics.med-ph].

[43] Yann Lecun, Sumit Chopra, Raia Hadsell, Marc Aurelio Ranzato, and Fu Jie Huang. "A tutorial on energy-based learning". English (US). In: *Predicting structured data*. Ed. by G. Bakir, T. Hofman, B. Scholkopt, A. Smola, and B. Taskar. MIT Press, 2006.

[44] Yilun Du and Igor Mordatch. "Implicit generation and generalization in energy-based models". In: *arXiv preprint arXiv:1903.08689* (2019).

[45] *Overview of gan structure*. URL: https://developers.google.com/machine-learning/gan/gan_structure.

[46] Tero Karras, Samuli Laine, and Timo Aila. *A Style-Based Generator Architecture for Generative Adversarial Networks*. 2019. arXiv: 1812.04948 [cs.NE].

[47] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. *Analyzing and Improving the Image Quality of StyleGAN*. 2020. arXiv: 1912.04958 [cs.CV].

[48] *This person does not exist*. URL: https://thispersondoesnotexist.com/.

[49] *Replicate reproducible machine learning*. URL: https://replicate.com/.

[50] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[51] Cassidy Silbernagel. "Investigation of the design, manufacture and testing of additively manufactured coils for electric motor applications". PhD thesis. Oct. 2019.

[52] Yingwei Liu, Xiaorong Gao, and Jinlong Li. "Image edge detection based on Sparse Autoencoder network". In: *Tenth International Conference on Information Optics and Photonics*. Ed. by Yidong Huang. Vol. 10964. International Society for Optics and Photonics. SPIE, 2018, pp. 625–632. DOI: 10.1117/12.2505873. URL: https://doi.org/10.1117/12.2505873.

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

[54] Lilian Weng. *From autoencoder to beta-vae*. Aug. 2018. URL: https://lilianweng.github.io/lil-log/2018/08/12/from-autoencoder-to-beta-vae.html.

[55] Masahiro Suzuki, Kotaro Nakayama, and Yutaka Matsuo. "Joint Multimodal Learning with Deep Generative Models". In: (Nov. 2016).

[56] Chunyuan Li. *Less pain, more gain, anyone? this vae trainer alleviates KL vanishing*. June 2019. URL: https://www.microsoft.com/en-us/research/blog/less-pain-more-gain-a-simple-method-for-vae-training-with-less-of-that-kl-vanishing-agony/.

[57] Marco Schreyer, Timur Sattarov, Damian Borth, Andreas Dengel, and Bernd Reimer. "Detection of Anomalies in Large Scale Accounting Data using Deep Autoencoder Networks". In: (Aug. 2018).

A Code Listings

```
import numpy as np                      #NumPy
import pandas as pd                     #Pandas
import tensorflow as tf                 #Tensorflow
from tensorflow import keras            #Keras
from tensorflow.keras import layers
from tensorflow.keras import callbacks
import matplotlib.pyplot as plt
%matplotlib inline

#Report file of model.summary and loss for every training session of the
    model
report_file = '/content/drive/MyDrive/Colab Notebooks/vae1 ds1.csv'
```

Listing A.1: Dependencies

```
# Import from csv files and reshape

# 4 samples/hr * 24hr/day = 96 samples/day
period_samples = 96

#Import 35.040 samples of active_power1 from Sub1act_630.csv
active_power1 = pd.read_csv('/content/drive/MyDrive/Datasets/
    MV_Greece_3customers_2018/Sub1act_630.csv', usecols=[4], header=None )


#Reshape into (-1, period_samples)=(365,96)
active_power1 = np.array(active_power1).reshape(-1, period_samples)

#Import 35.040 samples of reactive_power1 from Sub1react_630.csv
reactive_power1 = pd.read_csv('/content/drive/MyDrive/Datasets/
    MV_Greece_3customers_2018/Sub1react_630.csv', usecols=[4], header=None )

#Reshape into (-1, period_samples)=(365,96)
reactive_power1 = np.array(reactive_power1).reshape(-1, period_samples)

#Concatenate active and reactive power into a (365,96,2) numpy array
power1 = np.concatenate((active_power1[:,:,np.newaxis], reactive_power1[:,:,
    np.newaxis]), axis=2)


#Same as above for Load 2
active_power2 = pd.read_csv('/content/drive/MyDrive/Datasets/
    MV_Greece_3customers_2018/Sub2act_630.csv', usecols=[4], header=None )
active_power2 = np.array(active_power2).reshape(-1, period_samples)
reactive_power2 = pd.read_csv('/content/drive/MyDrive/Datasets/
    MV_Greece_3customers_2018/Sub2react_630.csv', usecols=[4], header=None )
reactive_power2 = np.array(reactive_power2).reshape(-1, period_samples)
power2 = np.concatenate((active_power2[:,:,np.newaxis], reactive_power2[:,:,
    np.newaxis]), axis=2)
```

```
#Same as above for Load 3
active_power3 = pd.read_csv('/content/drive/MyDrive/Datasets/
    MV_Greece_3customers_2018/Sub3act_630.csv', usecols=[4], header=None )
active_power3 = np.array(active_power3).reshape(-1, period_samples)
reactive_power3 = pd.read_csv('/content/drive/MyDrive/Datasets/
    MV_Greece_3customers_2018/Sub3react_630.csv', usecols=[4], header=None )
reactive_power3 = np.array(reactive_power3).reshape(-1, period_samples)
power3 = np.concatenate((active_power3[:,:,np.newaxis], reactive_power3[:,:,
    np.newaxis]), axis=2)
```

Listing A.2: Data import

```
#All three loads are concatenated into one, with shape (365,96,2,3)
power = np.concatenate((power1[...,np.newaxis],power2[...,np.newaxis],
    power3[...,np.newaxis]), axis=3)

#Maximum active and reactive power are determined
active_max = int(np.amax(power[:,:,0,:]))
reactive_max = int(np.amax(power[:,:,1,:]))

#Fill 2 numpy arrays for faster division
active_max_list=np.full(shape=(365,96,1,3), fill_value=active_max)
reactive_max_list=np.full(shape=(365,96,1,3), fill_value=reactive_max)

#Normalize the values
active_power_norm = np.true_divide(power[:,:,0,:], active_max)
reactive_power_norm = np.true_divide(power[:,:,1,:], reactive_max)

#Concatenate the normalized active and reactive power data
power_norm = np.concatenate((active_power_norm[:,:,np.newaxis,:],
    reactive_power_norm[:,:,np.newaxis,:]), axis=2)
```

Listing A.3: Data normalization

```

#Define the sampling function
class Sampling(layers.Layer):

    def call(self, inputs):
        z_mean, z_log_var = inputs    #The inputs of the sampling layer
        batch = tf.shape(z_mean)[0]   #The size of the 1st dimension
        dim = tf.shape(z_mean)[1]     #The size of the 2nd dimensions
                                      #Generate epsilon with the proper
                                          shape
        epsilon = tf.keras.backend.random_normal(shape=(batch, dim))
                                      #Return the sample
        return z_mean + tf.exp(0.5 * z_log_var) * epsilon
```

Listing A.4: Sampling layer

```
#The number of latent dimensions used
latent_dim = 2
```

```
#The input layer of the encoder
encoder_inputs = keras.Input(shape=(period_samples, 2, 3), name="samples")

#2D Convolution layer
x = layers.Conv2D(
      filters=48,        #The amount of filters used
      kernel_size=(1),   #The size of the "window" used for convolution
      activation="relu",#The activation function
      strides=1,         #The amount of "steps" walked every time
      padding="same"     #Add zeros when the convoluting kernel is outside
          the edge of the shape
      )(encoder_inputs) #This layer is connected to "encoder_inputs"

#Batch normalization
x = layers.BatchNormalization()(x)

#ReLU activation layer
x = layers.ReLU()(x)#96

#Reduce the number of samples to 48
x = layers.Conv2D(
      filters=48,
      kernel_size=(3,1),
      activation="relu",
      strides=(2,1),
      padding="same")(x)
x = layers.BatchNormalization()(x)
x = layers.ReLU()(x)#48

#Reduce the number of samples to 24
x = layers.Conv2D(filters=48, kernel_size=(3,1), activation="relu", strides
    =(2,1), padding="same")(x)
x = layers.BatchNormalization()(x)
x = layers.ReLU()(x)#24

#Reduce the number of samples to 12
x = layers.Conv2D(filters=48, kernel_size=(3,1), activation="relu", strides
    =(2,1), padding="same")(x)
x = layers.BatchNormalization()(x)#12
x = layers.ReLU()(x)

#Reduce the number of samples to 6
x = layers.Conv2D(filters=24, kernel_size=(3,1), activation="relu", strides
    =(2,1), padding="same")(x)
x = layers.BatchNormalization()(x)#6
x = layers.ReLU()(x)

#Reduce the number of samples to 3
x = layers.Conv2D(filters=12, kernel_size=(3,1), activation="relu", strides
    =(2,1), padding="same")(x)
x = layers.BatchNormalization()(x)
x = layers.ReLU()(x)
```

```
#Flatten the samples
x = layers.Flatten()(x)
x = layers.Dense(36, activation="relu")(x)

#Find the mean
z_mean = layers.Dense(latent_dim, name="z_mean")(x)

#Find the log variance
z_log_var = layers.Dense(latent_dim, name="z_log_var")(x)

#Encode the sample into vector z
z = Sampling()([z_mean, z_log_var])

#Group the layers into a Model object named "encoder"
encoder = keras.Model(
    encoder_inputs,
    [z_mean, z_log_var, z],
    name="encoder")

#Print a layout of the model created
encoder.summary()

#Record the model in the report file
with open(report_file,'a') as fh:
    encoder.summary(print_fn=lambda x: fh.write(x + '\n'))
```

Listing A.5: Encoder

```
#Input layer
latent_inputs = keras.Input(shape=(latent_dim,))

#An encoded sample is picked from the latent space
#with a shape of (3,2,12)->(samples, load type, filters)
x = layers.Dense(72, activation="relu")(latent_inputs)
x = layers.Reshape((3, 2, 12))(x)

#Increase the number of samples to 6
x = layers.Conv2DTranspose(
    filters=48,
    kernel_size=(3,1),
    activation="relu",
    strides=(2,1),
    padding="same")(x)

#Batch normalization
x = layers.BatchNormalization()(x)

#ReLU activation layer
x = layers.ReLU()(x)

#Increase the number of samples to 12
x = layers.Conv2DTranspose(filters=24, kernel_size=(3,1), activation="relu",
     strides=(2,1), padding="same")(x)
```

```
x = layers.BatchNormalization()(x)
x = layers.ReLU()(x)

#Increase the number of samples to 24
x = layers.Conv2DTranspose(filters=24, kernel_size=(3,1), activation="relu",
     strides=(2,1), padding="same")(x)
x = layers.BatchNormalization()(x)
x = layers.ReLU()(x)

#Increase the number of samples to 48
x = layers.Conv2DTranspose(filters=48, kernel_size=(3,1), activation="relu",
     strides=(2,1), padding="same")(x)
x = layers.BatchNormalization()(x)
x = layers.ReLU()(x)

#Increase the number of samples to 96
x = layers.Conv2DTranspose(filters=24, kernel_size=(3,1), activation="relu",
     strides=(2,1), padding="same")(x)
x = layers.BatchNormalization()(x)
x = layers.ReLU()(x)

#2D Convolution layer
decoder_outputs = layers.Conv2DTranspose( filters=3, kernel_size=1,
    activation="relu", padding="same")(x)

#Group the layers into a Model object named "decoder"
decoder = keras.Model(latent_inputs, decoder_outputs, name="decoder")

#Print a layout of the model created
decoder.summary()

#Record the model in the report file
with open(report_file,'a') as fh:
    decoder.summary(print_fn=lambda x: fh.write(x + '\n'))
```

Listing A.6: Decoder

```
#Define the model structure, loss functions and training step.
class VAE(keras.Model):
    def __init__(self, encoder, decoder, **kwargs):
        super(VAE, self).__init__(**kwargs)
        self.encoder = encoder
        self.decoder = decoder

#Define the training step
    def train_step(self, data):                    #Define the train step
        if isinstance(data, tuple):
            data = data[0]
        with tf.GradientTape() as tape:
#Take the encoded data from the encoder
            z_mean, z_log_var, z = encoder(data)
#Build a reconstruction in the decoder
            reconstruction = decoder(z)
#Define the reconstrucion loss
```

```
                reconstruction_loss = tf.reduce_mean(
                    keras.losses.binary_crossentropy(data, reconstruction)
                )
                reconstruction_loss *= period_samples
#Define the -KullbackLeibler divergence
                kl_loss = 1 + z_log_var - tf.square(z_mean) - tf.exp(z_log_var)
                kl_loss = tf.reduce_mean(kl_loss)
                kl_loss *= -0.5
#Calculate total loss
                total_loss = reconstruction_loss + kl_loss
            grads = tape.gradient(total_loss, self.trainable_weights)
            self.optimizer.apply_gradients(zip(grads, self.trainable_weights))
            return {
                "loss": total_loss,
                "reconstruction_loss": reconstruction_loss,
                "kl_loss": kl_loss,
            }
```

Listing A.7: Model definition

```
#Callback options, log the trianing to "report file"
callback = tf.keras.callbacks.CSVLogger(report_file, separator=",", append=
    True)


vae = VAE(encoder, decoder)

#Compile the model and use Adam optimizer
vae.compile(optimizer=keras.optimizers.Adam())

#Fit the model with the following options
vae.fit(
#The training data
     power_norm,
#The number of training iterations
     epochs=40,
#The amount of samples after which the network recalculates the weights
     batch_size=5,
#Callback options stated above
     callbacks=callback,
#The amount of information printed for every epoch
     verbose=1
     )
```

Listing A.8: Training the model

```
Epoch 1/40
73/73 [==============================] - 6s 20ms/step - loss: 88.8009 -
    reconstruction_loss: 88.5913 - kl_loss: 0.2096

Epoch 2/40
73/73 [==============================] - 1s 18ms/step - loss: 56.4877 -
    reconstruction_loss: 56.3671 - kl_loss: 0.1206
```

```

Epoch 3/40
73/73 [==============================] - 1s 19ms/step - loss: 54.9244 -
    reconstruction_loss: 54.8856 - kl_loss: 0.0387

Epoch 4/40
73/73 [==============================] - 1s 20ms/step - loss: 54.6932 -
    reconstruction_loss: 54.4776 - kl_loss: 0.2156

Epoch 5/40
73/73 [==============================] - 2s 21ms/step - loss: 54.3461 -
    reconstruction_loss: 54.0350 - kl_loss: 0.3111

Epoch 6/40
73/73 [==============================] - 1s 20ms/step - loss: 54.0534 -
    reconstruction_loss: 53.6039 - kl_loss: 0.4495

Epoch 7/40
73/73 [==============================] - 1s 19ms/step - loss: 54.0228 -
    reconstruction_loss: 53.6012 - kl_loss: 0.4215

Epoch 8/40
73/73 [==============================] - 2s 22ms/step - loss: 53.6642 -
    reconstruction_loss: 53.1139 - kl_loss: 0.5503

Epoch 9/40
73/73 [==============================] - 1s 18ms/step - loss: 53.4581 -
    reconstruction_loss: 52.9072 - kl_loss: 0.5510

Epoch 10/40
73/73 [==============================] - 2s 21ms/step - loss: 53.2399 -
    reconstruction_loss: 52.7238 - kl_loss: 0.5161

Epoch 11/40
73/73 [==============================] - 1s 18ms/step - loss: 53.1989 -
    reconstruction_loss: 52.6347 - kl_loss: 0.5642

Epoch 12/40
73/73 [==============================] - 2s 21ms/step - loss: 53.2086 -
    reconstruction_loss: 52.5887 - kl_loss: 0.6199

Epoch 13/40
73/73 [==============================] - 2s 21ms/step - loss: 53.5418 -
    reconstruction_loss: 52.9276 - kl_loss: 0.6142

Epoch 14/40
73/73 [==============================] - 1s 19ms/step - loss: 53.2940 -
    reconstruction_loss: 52.6971 - kl_loss: 0.5969

Epoch 15/40
73/73 [==============================] - 1s 19ms/step - loss: 53.1278 -
    reconstruction_loss: 52.5326 - kl_loss: 0.5952

```

```
Epoch 16/40
73/73 [==============================] - 1s 18ms/step - loss: 53.1007 -
    reconstruction_loss: 52.5071 - kl_loss: 0.5936

Epoch 17/40
73/73 [==============================] - 1s 18ms/step - loss: 53.0862 -
    reconstruction_loss: 52.4662 - kl_loss: 0.6200

Epoch 18/40
73/73 [==============================] - 1s 19ms/step - loss: 53.0682 -
    reconstruction_loss: 52.4596 - kl_loss: 0.6086

Epoch 19/40
73/73 [==============================] - 1s 18ms/step - loss: 52.9723 -
    reconstruction_loss: 52.3740 - kl_loss: 0.5983

Epoch 20/40
73/73 [==============================] - 1s 19ms/step - loss: 53.0760 -
    reconstruction_loss: 52.4725 - kl_loss: 0.6035

Epoch 21/40
73/73 [==============================] - 1s 18ms/step - loss: 53.0464 -
    reconstruction_loss: 52.4420 - kl_loss: 0.6044

Epoch 22/40
73/73 [==============================] - 1s 13ms/step - loss: 53.0808 -
    reconstruction_loss: 52.4362 - kl_loss: 0.6447

Epoch 23/40
73/73 [==============================] - 1s 11ms/step - loss: 53.0594 -
    reconstruction_loss: 52.4735 - kl_loss: 0.5858

Epoch 24/40
73/73 [==============================] - 1s 11ms/step - loss: 53.0553 -
    reconstruction_loss: 52.4092 - kl_loss: 0.6461

Epoch 25/40
73/73 [==============================] - 1s 11ms/step - loss: 53.0455 -
    reconstruction_loss: 52.4263 - kl_loss: 0.6191

Epoch 26/40
73/73 [==============================] - 1s 11ms/step - loss: 53.0377 -
    reconstruction_loss: 52.4527 - kl_loss: 0.5850

Epoch 27/40
73/73 [==============================] - 1s 11ms/step - loss: 53.0735 -
    reconstruction_loss: 52.4077 - kl_loss: 0.6658
```

Listing A.9: Training epochs

```
#Plot grid of n*n plots from (-scale, scale) of latent space

def plot_latent(encoder, decoder):

```

```
    n = 7           #Number of samples per axis
    digit_size = period_samples
    x_scale = 5     #Range of the x axis
    y_scale = 5    #Range of the y axis
    figsize_x = 18  #Width of the plot grid
    figsize_y = 12  #Height of the plot grid
                    #Create linear space for x
    grid_x = np.linspace(-x_scale, x_scale, n)
                    #Create inear space for y
    grid_y = np.linspace(-y_scale, y_scale, n)

                    #Create n*n grid of subplots
    fig, axs = plt.subplots(n, n)

#For every subplot
    for i, yi in enumerate(grid_y):
        for j, xi in enumerate(grid_x):
#Sample from point [xi, -yi]
            z_sample = np.array([[xi, -yi]])
#Decode the sample
            x_decoded = decoder.predict(z_sample)
            prediction = x_decoded[0,:,:,2]
#Multiply the result with active_max to get results in MW
            axs[i, j].plot(prediction[:,0])
#Same for the reactive power
            axs[i, j].plot(prediction[:,1]*float(reactive_max))
#Set the title of the subplot and the axis limits
            axs[i, j].set_title(('Point ['+str(round(xi, 1))+', '+str(round
                (-yi, 1))+']'))
            axs[i, j].set_ylim(0, int(active_max))
            axs[i, j].set_xticks([])

    for ax in axs.flat:
      ax.label_outer()

#Set the main plot size and print
    fig.set_size_inches(figsize_x, figsize_y)
    plt.subplots
    plt.show()

plot_latent(encoder, decoder)
```

Listing A.10: Plot mapping of the latent space

```

#Encode power_norm dataset using the encoder
encoded_set = encoder.predict(power_norm)
encoded_z = encoded_set[2]

#Decode (reconstruct) the data
decoded_set = decoder.predict(encoded_z)

reconstructed_active = np.multiply(decoded_set[:,:,0,:], active_max)
reconstructed_reactive = np.multiply(decoded_set[:,:,1,:], reactive_max)
```

```
reconstructed_power = np.concatenate((reconstructed_active[:,:,np.newaxis
    ,:], reconstructed_reactive[:,:,np.newaxis,:]), axis=2)
```

Listing A.11: Encoding and decoding of the dataset using the trained model

```
num_cols = power.shape[0]
samples_num = 5
loads_num = power.shape[3]

fig, axs = plt.subplots(2*loads_num, samples_num)

for j in range(samples_num):
  sample_id = np.random.randint(0,num_cols)

  for i in range(2*loads_num):
    if i==0:
      axs[i, j].plot(power[sample_id,:,:,0])
      axs[i, j].set_title(('DAY No'+str(sample_id)+'\n Load 1'))
      axs[i, j].set_ylim(0, int(active_max))
      axs[i, j].set_xticks([])
    elif i==1:
      axs[i, j].plot(decoded_set[sample_id,:,0,0]*float(active_max))
      axs[i, j].plot(decoded_set[sample_id,:,1,0]*float(reactive_max))
      axs[i, j].set_title(('Load 1 Reconstrucion'))
      axs[i, j].set_ylim(0, int(active_max))
      axs[i, j].set_xticks([])
    elif i==2:
      axs[i, j].plot(power[sample_id,:,:,1])
      axs[i, j].set_title(('Load 2'))
      axs[i, j].set_ylim(0, int(active_max))
      axs[i, j].set_xticks([])
    elif i==3:
      axs[i, j].plot(decoded_set[sample_id,:,0,1]*float(active_max))
      axs[i, j].plot(decoded_set[sample_id,:,1,1]*float(reactive_max))
      axs[i, j].set_title(('Load 2 Reconstrucion'))
      axs[i, j].set_ylim(0, int(active_max))
      axs[i, j].set_xticks([])
    elif i==4:
      axs[i, j].plot(power[sample_id,:,:,2])
      axs[i, j].set_title(('Load 3'))
      axs[i, j].set_ylim(0, int(active_max))
      axs[i, j].set_xticks([])
    elif i==5:
      axs[i, j].plot(decoded_set[sample_id,:,0,2]*float(active_max))
      axs[i, j].plot(decoded_set[sample_id,:,1,2]*float(reactive_max))
      axs[i, j].set_title(('Load 3 Reconstrucion'))
      axs[i, j].set_ylim(0, int(active_max))
      axs[i, j].set_xticks([])
fig.set_size_inches(18, 13)
plt.show
```

```

x_ticks = np.linspace(0,96,365, endpoint=False)

fig, axs = plt.subplots(6, 2)

for j in range(2):
   for i in range(6):

    if i==0:
      axs[i, j].hist2d(x_ticks, power[:,0,j,0], bins=[32, 15], density=True,
           cmap='Greens', alpha=1, edgecolor='white')
      if j==0:
       axs[i, j].set_title(('Load '+str(i+1)+' Active Power'))
       axs[i, j].set_xticks([])
      elif j==1:
        axs[i, j].set_title(('Load '+str(i+1)+' Reactive Power'))
        axs[i, j].set_xticks([])
    elif i==1:
      axs[i, j].hist2d(x_ticks, reconstructed_power[:,0,j,0], bins=[32, 15],
           density=True, cmap='Greens', alpha=1, edgecolor='white')
      if j==0:
       axs[i, j].set_title(('Reconstructed Load '+str(1)+' Active Power'))
       axs[i, j].set_xticks([])
      elif j==1:
        axs[i, j].set_title(('Reconstructed Load '+str(1)+' Reactive Power')
            )
        axs[i, j].set_xticks([])
    elif i==2:
      axs[i, j].hist2d(x_ticks, power[:,0,j,1], bins=[32, 15], density=True,
           cmap='Greens', alpha=1, edgecolor='white')
      if j==0:
       axs[i, j].set_title(('Load '+str(i)+' Active Power'))
       axs[i, j].set_xticks([])
      elif j==1:
        axs[i, j].set_title(('Load '+str(i)+' Reactive Power'))
        axs[i, j].set_xticks([])
    elif i==3:
      axs[i, j].hist2d(x_ticks, reconstructed_power[:,0,j,1], bins=[32, 15],
           density=True, cmap='Greens', alpha=1, edgecolor='white')
      if j==0:
       axs[i, j].set_title(('Reconstructed Load '+str(i-1)+' Active Power'))
       axs[i, j].set_xticks([])
      elif j==1:
        axs[i, j].set_title(('Reconstructed Load '+str(i-1)+' Reactive Power
            '))
        axs[i, j].set_xticks([])
    elif i==4:
      axs[i, j].hist2d(x_ticks, power[:,0,j,2], bins=[32, 15], density=True,
           cmap='Greens', alpha=1, edgecolor='white')
      if j==0:
       axs[i, j].set_title(('Load '+str(i-1)+' Active Power'))
       axs[i, j].set_xticks([])
      elif j==1:
```

```
            axs[i, j].set_title(('Load '+str(i-1)+' Reactive Power'))
            axs[i, j].set_xticks([])
        elif i==5:
          axs[i, j].hist2d(x_ticks, reconstructed_power[:,0,j,2], bins=[32, 15],
              density=True, cmap='Greens', alpha=1, edgecolor='white')
          if j==0:
           axs[i, j].set_title(('Reconstructed Load '+str(i-2)+' Active Power'))
           axs[i, j].set_xticks([])
          elif j==1:
            axs[i, j].set_title(('Reconstructed Load '+str(i-2)+' Reactive Power
                '))
            axs[i, j].set_xticks([])

fig.set_size_inches(20, 13)
plt.show
```

Listing A.12: Heatmap of the original and the reconstructed dataset

```
plt.scatter(encoded_z[:,0], encoded_z[:,1]);
double_encoded_set = encoder.predict(decoded_set)
double_encoded_z = double_encoded_set[2]
plt.scatter(double_encoded_z[:,0], double_encoded_z[:,1])
```

Listing A.13: Scatter plot of the original and the reconstructed code

```
reconstructed_active = np.multiply(decoded_set[:,:,0,:], active_max)
reconstructed_reactive = np.multiply(decoded_set[:,:,1,:], reactive_max)
reconstructed_power = np.concatenate((reconstructed_active[:,:,np.newaxis
    ,:], reconstructed_reactive[:,:,np.newaxis,:]), axis=2)
[['Load 1'],
 [np.amax(power[:,:,0,0]),np.amax(reconstructed_power[:,:,0,0]), (1-(float(
     np.amax(power[:,:,0,0])/float(np.amax(reconstructed_power[:,:,0,0])))))
     *100],
 [np.amax(power[:,:,1,0]),np.amax(reconstructed_power[:,:,1,0]), (1-(float(
     np.amax(power[:,:,1,0])/float(np.amax(reconstructed_power[:,:,1,0])))))
     *100],
 [np.average(power[:,:,0,0]),np.average(reconstructed_power[:,:,0,0]), (1-(
     float(np.average(power[:,:,0,0])/float(np.average(reconstructed_power
     [:,:,0,0])))))*100],
 [np.average(power[:,:,1,0]),np.average(reconstructed_power[:,:,1,0]), (1-(
     float(np.average(power[:,:,1,0])/float(np.average(reconstructed_power
     [:,:,1,0])))))*100],
 [int(np.sum(power[:,:,0,0]))/4,int(np.sum(reconstructed_power[:,:,0,0]))/4,
      (1-(float(np.sum(power[:,:,0,0])/float(np.sum(reconstructed_power
     [:,:,0,0])))))*100],
 [int(np.sum(power[:,:,1,0]))/4,int(np.sum(reconstructed_power[:,:,1,0]))/4,
      (1-(float(np.sum(power[:,:,1,0])/float(np.sum(reconstructed_power
     [:,:,1,0])))))*100],
 [ 'Load 2' ],
 [np.amax(power[:,:,0,1]),np.amax(reconstructed_power[:,:,0,1]), (1-(float(
     np.amax(power[:,:,0,1])/float(np.amax(reconstructed_power[:,:,0,1])))))
     *100],
```

```
 [np.amax(power[:,:,1,1]),np.amax(reconstructed_power[:,:,1,1]), (1-(float(
    np.amax(power[:,:,1,1])/float(np.amax(reconstructed_power[:,:,1,1]))))
    *100],
 [np.average(power[:,:,0,1]),np.average(reconstructed_power[:,:,0,1]), (1-(
    float(np.average(power[:,:,0,1])/float(np.average(reconstructed_power
    [:,:,0,1])))))*100],
 [np.average(power[:,:,1,1]),np.average(reconstructed_power[:,:,1,1]), (1-(
    float(np.average(power[:,:,1,1])/float(np.average(reconstructed_power
    [:,:,1,1])))))*100],
 [int(np.sum(power[:,:,0,1]))/4,int(np.sum(reconstructed_power[:,:,0,1]))/4,
     (1-(float(np.sum(power[:,:,0,1])/float(np.sum(reconstructed_power
    [:,:,0,1])))))*100],
 [int(np.sum(power[:,:,1,1]))/4,int(np.sum(reconstructed_power[:,:,1,1]))/4,
     (1-(float(np.sum(power[:,:,1,1])/float(np.sum(reconstructed_power
    [:,:,1,1])))))*100],
 ['Load 3'],
 [np.amax(power[:,:,0,2]),np.amax(reconstructed_power[:,:,0,2]), (1-(float(
    np.amax(power[:,:,0,2])/float(np.amax(reconstructed_power[:,:,0,2]))))
    *100],
 [np.amax(power[:,:,1,2]),np.amax(reconstructed_power[:,:,1,0]), (1-(float(
    np.amax(power[:,:,1,2])/float(np.amax(reconstructed_power[:,:,1,2]))))
    *100],
 [np.average(power[:,:,0,2]),np.average(reconstructed_power[:,:,0,2]), (1-(
    float(np.average(power[:,:,0,2])/float(np.average(reconstructed_power
    [:,:,0,2])))))*100],
 [np.average(power[:,:,1,2]),np.average(reconstructed_power[:,:,1,2]), (1-(
    float(np.average(power[:,:,1,2])/float(np.average(reconstructed_power
    [:,:,1,2])))))*100],
 [int(np.sum(power[:,:,0,2]))/4,int(np.sum(reconstructed_power[:,:,0,2]))/4,
     (1-(float(np.sum(power[:,:,0,2])/float(np.sum(reconstructed_power
    [:,:,0,2])))))*100],
 [int(np.sum(power[:,:,1,2]))/4,int(np.sum(reconstructed_power[:,:,1,2]))/4,
     (1-(float(np.sum(power[:,:,1,2])/float(np.sum(reconstructed_power
    [:,:,1,2])))))*100],
```

Listing A.14: Numerical metrics

```
random_1A = np.random.normal(loc=0, scale=1, size=365)
random_1B = np.random.normal(loc=0, scale=1, size=365)
random_1 = np.stack((random_1A, random_1B), axis=-1 )
random_gen_code_1 = np.add(encoded_z, random_1)
sample_1 = decoder(random_gen_code_1)
random_active_1 = np.multiply(sample_1[:,:,0,:], active_max)
random_reactive_1 = np.multiply(sample_1[:,:,1,:], reactive_max)
random_power_1 = np.concatenate((random_active_1[:,:,np.newaxis,:],
    random_reactive_1[:,:,np.newaxis,:]), axis=2)

[["Max Original", 'Max Reconstructed'],
 [np.amax(power[:,:,0,:]),np.amax(random_power_1[:,:,0,:])],
 [np.amax(power[:,:,1,:]),np.amax(random_power_1[:,:,1,:])],
 ['Average Original', 'Average Reconstructed'],
 [np.average(power[:,:,0,:]),np.average(random_power_1[:,:,0,:])],
 [np.average(power[:,:,1,:]),np.average(random_power_1[:,:,1,:])],
 ['Variance Original', 'Variance Reconstructed'],
```

```
[np.var(power[:,:,0,:]),np.var(random_power_1[:,:,0,:])],
[np.var(power[:,:,1,:]),np.var(random_power_1[:,:,1,:])],
['Total Energy Original', 'Total Energy Recostructed'],
[int(np.sum(power[:,:,1,:]))/4,int(np.sum(random_power_1[:,:,1,:]))/4]]
```

Listing A.15: Create new dataset by adding a normal distribution

```
num_cols = power.shape[0]
samples_num = 5
loads_num = power.shape[3]

fig, axs = plt.subplots(2*loads_num, samples_num)

for j in range(samples_num):
  sample_id = np.random.randint(0,num_cols)

  for i in range(2*loads_num):
    if i==0:
      axs[i, j].plot(power[sample_id,:,:,0])
      axs[i, j].set_title(('DAY No'+str(sample_id)+'\n Load 1'))
      axs[i, j].set_ylim(0, int(active_max))
      axs[i, j].set_xticks([])
    elif i==1:
      axs[i, j].plot(random_power_1[sample_id,:,0,0])
      axs[i, j].plot(random_power_1[sample_id,:,1,0])
      axs[i, j].set_title(('New Load 1'))
      axs[i, j].set_ylim(0, int(active_max))
      axs[i, j].set_xticks([])
    elif i==2:
      axs[i, j].plot(power[sample_id,:,:,1])
      axs[i, j].set_title(('Load 2'))
      axs[i, j].set_ylim(0, int(active_max))
      axs[i, j].set_xticks([])
    elif i==3:
      axs[i, j].plot(random_power_1[sample_id,:,0,1])
      axs[i, j].plot(random_power_1[sample_id,:,1,1])
      axs[i, j].set_title(('New Load 2'))
      axs[i, j].set_ylim(0, int(active_max))
      axs[i, j].set_xticks([])
    elif i==4:
      axs[i, j].plot(power[sample_id,:,:,2])
      axs[i, j].set_title(('Load 3'))
      axs[i, j].set_ylim(0, int(active_max))
      axs[i, j].set_xticks([])
    elif i==5:
      axs[i, j].plot(random_power_1[sample_id,:,0,2])
      axs[i, j].plot(random_power_1[sample_id,:,1,2])
      axs[i, j].set_title(('New Load 3'))
      axs[i, j].set_ylim(0, int(active_max))
      axs[i, j].set_xticks([])
fig.set_size_inches(18, 13)
plt.show
```

Listing A.16: Plot of 5 random samples and their reconstruction

```
x_ticks = np.linspace(0,96,365, endpoint=False)

fig, axs = plt.subplots(6, 2)

for j in range(2):
   for i in range(6):

     if i==0:
       axs[i, j].hist2d(x_ticks, power[:,0,j,0], bins=[32, 15], density=True,
            cmap='Greens', alpha=1, edgecolor='white')
       if j==0:
         axs[i, j].set_title(('Load '+str(i+1)+' Active Power'))
         axs[i, j].set_xticks([])

       elif j==1:
          axs[i, j].set_title(('Load '+str(i+1)+' Reactive Power'))
          axs[i, j].set_xticks([])
     elif i==1:
       axs[i, j].hist2d(x_ticks, random_power_1[:,0,j,0], bins=[32, 15],
           density=True, cmap='Greens', alpha=1, edgecolor='white')
       if j==0:
         axs[i, j].set_title(('Reconstructed Load '+str(1)+' Active Power'))
         axs[i, j].set_xticks([])
       elif j==1:
          axs[i, j].set_title(('Reconstructed Load '+str(1)+' Reactive Power')
              )
          axs[i, j].set_xticks([])
     elif i==2:
       axs[i, j].hist2d(x_ticks, power[:,0,j,1], bins=[32, 15], density=True,
            cmap='Greens', alpha=1, edgecolor='white')
       if j==0:
         axs[i, j].set_title(('Load '+str(i)+' Active Power'))
         axs[i, j].set_xticks([])
       elif j==1:
          axs[i, j].set_title(('Load '+str(i)+' Reactive Power'))
          axs[i, j].set_xticks([])
     elif i==3:
       axs[i, j].hist2d(x_ticks, random_power_1[:,0,j,1], bins=[32, 15],
           density=True, cmap='Greens', alpha=1, edgecolor='white')
       if j==0:
         axs[i, j].set_title(('Reconstructed Load '+str(i-1)+' Active Power'))
         axs[i, j].set_xticks([])
       elif j==1:
          axs[i, j].set_title(('Reconstructed Load '+str(i-1)+' Reactive Power
              '))
          axs[i, j].set_xticks([])
     elif i==4:
       axs[i, j].hist2d(x_ticks, power[:,0,j,2], bins=[32, 15], density=True,
            cmap='Greens', alpha=1, edgecolor='white')
       if j==0:
         axs[i, j].set_title(('Load '+str(i-1)+' Active Power'))
         axs[i, j].set_xticks([])
       elif j==1:
```

```
        axs[i, j].set_title(('Load '+str(i-1)+' Reactive Power'))
        axs[i, j].set_xticks([])
    elif i==5:
      axs[i, j].hist2d(x_ticks, random_power_1[:,0,j,2], bins=[32, 15],
          density=True, cmap='Greens', alpha=1, edgecolor='white')
      if j==0:
       axs[i, j].set_title(('Reconstructed Load '+str(i-2)+' Active Power'))
       axs[i, j].set_xticks([])
      elif j==1:
        axs[i, j].set_title(('Reconstructed Load '+str(i-2)+' Reactive Power
            '))
        axs[i, j].set_xticks([])

fig.set_size_inches(20, 13)
plt.show
```

Listing A.17: Heatmaps of the original and the new dataset

```
plt.scatter(random_gen_code_1[:,0], random_gen_code_1[:,1])
plt.xlim(-5, 5)
plt.ylim(-5, 5)
plt.show
```

Listing A.18: Scatterplot of the new code

```

[['Load 1'],
 [np.amax(power[:,:,0,0]),np.amax(random_power_1[:,:,0,0]), (1-(float(np.
     amax(power[:,:,0,0])/float(np.amax(random_power_1[:,:,0,0])))))*100],
 [np.amax(power[:,:,1,0]),np.amax(random_power_1[:,:,1,0]), (1-(float(np.
     amax(power[:,:,1,0])/float(np.amax(random_power_1[:,:,1,0])))))*100],
 [np.average(power[:,:,0,0]),np.average(random_power_1[:,:,0,0]), (1-(float(
     np.average(power[:,:,0,0])/float(np.average(random_power_1[:,:,0,0])))))
     *100],
 [np.average(power[:,:,1,0]),np.average(random_power_1[:,:,1,0]), (1-(float(
     np.average(power[:,:,1,0])/float(np.average(random_power_1[:,:,1,0])))))
     *100],
 [int(np.sum(power[:,:,0,0]))/4,int(np.sum(random_power_1[:,:,0,0]))/4, (1-(
     float(np.sum(power[:,:,0,0])/float(np.sum(random_power_1[:,:,0,0])))))
     *100],
 [int(np.sum(power[:,:,1,0]))/4,int(np.sum(random_power_1[:,:,1,0]))/4, (1-(
     float(np.sum(power[:,:,1,0])/float(np.sum(random_power_1[:,:,1,0])))))
     *100],
 [ 'Load 2' ],
 [np.amax(power[:,:,0,1]),np.amax(random_power_1[:,:,0,1]), (1-(float(np.
     amax(power[:,:,0,1])/float(np.amax(random_power_1[:,:,0,1])))))*100],
 [np.amax(power[:,:,1,1]),np.amax(random_power_1[:,:,1,1]), (1-(float(np.
     amax(power[:,:,1,1])/float(np.amax(random_power_1[:,:,1,1])))))*100],
 [np.average(power[:,:,0,1]),np.average(random_power_1[:,:,0,1]), (1-(float(
     np.average(power[:,:,0,1])/float(np.average(random_power_1[:,:,0,1])))))
     *100],
 [np.average(power[:,:,1,1]),np.average(random_power_1[:,:,1,1]), (1-(float(
     np.average(power[:,:,1,1])/float(np.average(random_power_1[:,:,1,1])))))
     *100],
```

```
[int(np.sum(power[:,:,0,1]))/4,int(np.sum(random_power_1[:,:,0,1]))/4, (1-(
    float(np.sum(power[:,:,0,1])/float(np.sum(random_power_1[:,:,0,1])))))
    *100],
[int(np.sum(power[:,:,1,1]))/4,int(np.sum(random_power_1[:,:,1,1]))/4, (1-(
    float(np.sum(power[:,:,1,1])/float(np.sum(random_power_1[:,:,1,1])))))
    *100],
['Load 3'],
[np.amax(power[:,:,0,2]),np.amax(random_power_1[:,:,0,2]), (1-(float(np.
    amax(power[:,:,0,2])/float(np.amax(random_power_1[:,:,0,2])))))*100],
[np.amax(power[:,:,1,2]),np.amax(random_power_1[:,:,1,0]), (1-(float(np.
    amax(power[:,:,1,2])/float(np.amax(random_power_1[:,:,1,2])))))*100],
[np.average(power[:,:,0,2]),np.average(random_power_1[:,:,0,2]), (1-(float(
    np.average(power[:,:,0,2])/float(np.average(random_power_1[:,:,0,2])))))
    *100],
[np.average(power[:,:,1,2]),np.average(random_power_1[:,:,1,2]), (1-(float(
    np.average(power[:,:,1,2])/float(np.average(random_power_1[:,:,1,2])))))
    *100],
[int(np.sum(power[:,:,0,2]))/4,int(np.sum(random_power_1[:,:,0,2]))/4, (1-(
    float(np.sum(power[:,:,0,2])/float(np.sum(random_power_1[:,:,0,2])))))
    *100],
[int(np.sum(power[:,:,1,2]))/4,int(np.sum(random_power_1[:,:,1,2]))/4, (1-(
    float(np.sum(power[:,:,1,2])/float(np.sum(random_power_1[:,:,1,2])))))
    *100],
```

Listing A.19: Numerical metrics of the new dataset

National Technical
University of
Athens

9 Iroon Polytechniou Str, Zografou, 15232
Athens
Tel. +30 210 772 3699

www.ece.ntua.gr