Generative Modelling in Non-Euclidean Domains

Michael P. Kenning, BSc

Submitted to Swansea University in fulfilment of the requirements for the Degree of Master of Research

Swansea University
Prifysgol Abertawe

January, 2019

Summary

Machine learning has been advanced considerably by the development of deep learning. Conventional deep-learning techniques are, however, limited to topologically regular data, leaving out many data domains. A new field of deep learning in irregular domains has opened up to fill this lacuna. The graph is one mode of representing data; and many natural data are represented well as graphs. Recent work has advanced deep learning on graphs, but very little has been advanced in terms of generative methods.

In this work we present a graph-based convolutional autoencoder (GCAE) on two datasets. The first dataset is a modified MNIST dataset. We evaluate the effect of a set of parameters of the network on the reconstruction error. We find that the number of graph convolutions per block decreases the reconstruction error substantially. Increasing the number of output maps in convolutions, however, does not reduce the error. The number of tracked weights in the network does not significantly effect the time required to train the network.

The next dataset is the Temple University Hospital EEG Corpus. The dataset consists of a large collection of electroencephalography scans from over 316 unique patients. We likewise apply a GCAE, structurally inspired by the GCAE applied to the MNIST dataset. Unfortunately the model does not perform as well on the dataset as we had hoped; the reconstruction errors between the GCAE and a convolutional autoencoder are similar, for example.

A great part of the challenge in graph deep learning is constructing a graph that is appropriate to the dataset. This is a far greater challenge in itself and requires greater attention than this thesis can provide. We also posit that the dataset itself limited the ability to develop an effective model.

Declarations

Declaration

This work has not previously been accepted in substance for any degree and is not being concurrently submitted in candidature for any degree.

Signed ____________________ (candidate)

Date ____________________

Statement 1

This thesis is the result of my own independent work/investigation, except where otherwise stated. Other sources are acknowledged by giving explicit references. A bibliography is appended.

Signed ____________________ (candidate)

Date ____________________

Statement 2

I hereby give my consent for my thesis, if accepted, to be available for photocopying and for inter-library loan, and for the title and summary to be made available to outside organisations.

Signed ____________________ (candidate)

Date ____________________

Contents

Acknowledgements

Of my colleagues: I should like first of all to thank Xianghua Xie; for his supervision, attention and steadying hand; without whom my degree work would not have been possible. I should also like to thank (in no particular order) Jingjing Deng, Michael Edwards, David George and Joss Whittle for the help they afforded me throughout my degree, however begrudgingly given it sometimes was. Additionally, I thank Jay Morgan, Man Duhu and all colleagues in Faraday's fourth-floor lab for the company and conversation.

Of my family: I thank my mother Alison Royal for letting me live rent-free for the year, putting up with my reversion to type, and her support. I thank my girlfriend Kristina Seit for her tolerating me and her companionship; without her, I should have imploded.

Finally, to my uncle Timothy Royal, whom I did not know as well as one might; and my grandfather Peter Royal, who was like a father to me. This work is dedicated to their memory.

List of Figures

Abbreviations

ACNS	American Clinical Neurophysiological Society
AE	autoencoder
AMG	algebraic multigrid
CAE	convolutional autoencoder
CNN	convolutional neural network
CoD	curse of dimensionality
EDF	European Data Format
EED	European Epilepsy Database
EEG	electroencephalography
GCAE	graph-based convolutional autoencoder
GCVAE	graph-convolutional variational autoencoder
GPGPU	general-purpose graphics processing-unit
HAR	human-action recognition
ILSVRC	International Large-Scale Visual Recognition Challenge
LMS	least mean squares
LSTM	long short-term memory
MLP	multi-layer perceptron
PCA	principal component analysis
RF	random forests algorithm
SVD	singular-value decomposition
SVM	support-vector machine
TUH-EEG	Temple University Hospital EEG Corpus

We went to R's room. To look at it, you'd think everything was just exactly like my place. Same table on the wall, and the armchairs, table, chest, bed, all made with the same glass. But R had hardly entered before he moved one of the easy chairs, then the other, and the planes were dislocated, everything slipped out of the prescribed correlation and became non-Euclidean. R will never change, never.

—D-503 in *We* by Yevgeny Zamyatin

Chapter 1

Introduction

Contents

1.1 Motivation

Machine learning continues to grow and thrive, achieving undreamed-of results on problems as wide and varying as image recognition, natural-language translation—and even election manipulation [1], [2]. The meteoric rise of machine learning in recent years is mainly due to the development of more sophisticated deep-learning models. Deep-learning models are able to take data and learn automatically a way to represent the data for the purposes of classification, recognition, denoising, localisation and a variety of other purposes. Large corporations now have research groups dedicated to deep-learning research, such as Facebook's AI Research group and Alphabet's DeepMind project.

The most prominent of such learning techniques are limited to working on data in regular domains. Images and videos, for example, have a regular, grid-like structure; the pixels in an image or video have a meaningful spatial relationship. The convolutional neural network (CNN), the most prominent type of neural network today, is able to take advantage of local patterns in images, such as eyes. Eyes, for example, have a specific statistical profile in an image; a CNN can learn these local structures and leverage them in learning tasks—all automatically.

As we mentioned, CNNs are limited to regular data, so these techniques, as powerful as they are, cannot be applied to data where these spatial relations do not exist or not structured in a grid- or array-like way. One example, used in this thesis, is the electroencephalography (EEG) scan. EEG scans are recorded using electrodes placed across the surface of a human subject's scalp. These electrodes are not placed on the scalp on some regular grid; they are distributed according to angles at set intervals from the top of the scalp. The surface is also curved, like the surface of a sphere.

The surface of a sphere and the 'surface' of an image differ starkly in their characteristics. For instance, to measure distances in each case requires two different equations. These geometrical differences result in further problems in machine learning. A CNN uses convolution kernels to 'walk' over the surface of an image. These kernels are much smaller than the image, so focus on smaller sections of the image, like a spotlight, except a strict square. There are many kernels in a CNN, each of which learn to identify different features. In an image, one might learn to identify lines; another to identify blobs. These features add up to produce an eye, a mouth, a nose, etc. One cannot walk a kernel over the surface of a sphere, as there is a degree of divergence between the kernel and the surface of a sphere. (Of course, as the diameter of the sphere approaches infinity, the amount of divergence tends to zero; i.e., the surface of the sphere at a given point approximates a flat surface.)

The kernels work on an image because they work in squares, and images nicely divide into subsets of (overlapping) squares. On the contrary, the surface of a sphere does not have this necessary property. One can imagine trying to place a 3-by-3 grid on a persons scalp covered in EEG sensors; the grid will not likely encompass a constant number of sensors wherever it is placed. The same grid on an grid of pixels, however, *will* encompass a constant number of pixels—so long as it lies within the boundary of an image.

The surfaces of spheres are not the only example of spaces where convolutions do not work; there are other geometries where kernels cannot work either. Many real-world data do exhibit these irregular topologies, which means any technique that works on irregular topologies and retains the power of deep-learning algorithms would be of great value. Consider the surface of the Earth and the distribution of earthquake sensors.

Graph deep learning is precisely that area looking to develop machine learning for irregular spaces. A majority of work focuses on graph-convolutional neural networks, the irregular-domain counterpart to CNN. Little work has been conducted into graph-based convolutional autoencoders, however, with which great strides could be made in irregular-domain encoding. Developing such a method is the focus of this thesis.

1.2 Contributions

The contributions of this work are two-fold. The first contribution is incremental. We present a graph-based convolutional autoencoder (GCAE) that allows us to perform encoding on irregular domains. The architecture is the extension of existing, conventional, regular-domain techniques to irregular domains. As many advances in computer vision have been made with generative techniques, this work will provide the first stepping-stone to a wider field of generative learning in irregular domains. The findings in this work are the basis for further research.

The second contribution is the technique we have applied to EEG scans. Previous techniques have required intensive, computationally expensive feature-extraction strategies before any deep-learning methods can be applied. In our work we discard all feature-extraction strategies and work solely on raw data while leveraging the spatial information present in the graph Laplacian matrix. With deep-learning architectures being automatic learners, this may give rise to techniques that learn more valid or informative representations of the data. These advances would therefore be applicable in medicine and seizure-detection problems.

1.3 Thesis Layout

In Chapter 2 we summarise the main intuitions behind and motivations for machine learning. Linear regression is presented as a basic example of a learning problem. With the discussion of higher dimensions and the curse of dimensionality, we move onto dimensionality reduction and domain experts as feature extractors. Deep learning is introduced in this context as an automatic feature-learner in the place of domain experts; deep learning's purported strength lies in its ability to automatically learn feature representations. Finally we describe a limitation of conventional deep-learning techniques, its inability to work with topologically irregular data. We present graph deep learning as one set of techniques to address this limitation, and identify the lack of research into embedding-learning in irregular domains.

In Chapter 3 we present a GCAE. We introduce the definition of graphs and describe how convolutions can be implemented in graph domains in spite of its spatial irregularity, namely through Fourier transformation and spectral multipliers. We also present algebraic multigrid pooling as a method to reduce the dimensionality of a graph in a way that allows an 'unpooling', analogous to upsampling in regular-domain methods. We then present the results of some experiments on the GCAE in order to validate it. We show that it is able to learn encodings on a modified MNIST dataset.

In Chapter 4 we apply what we learned in Chapter 3 to real-world data; namely, EEG scans from the Temple University Hospital EEG Corpus. The architecture did not prove successful in experiments; there was no evidence to show that it is better than our regular-domain models. We ascribed its failure to problems relating to our architecture and the construction of the graph.

In Chapter 5 we conclude by outlining the more immediate research challenges and the more distance ones. The more immediate challenges constitute searching for an architecture that does work well, in order to understand why our architecture failed. The second is the use of a new dataset that might provide more reliable data. The more distant challenges are those we believe require far greater and far more prolonged attention. We ask if it is possible to *learn* the best representation of graph data; and whether we can learn from graphs with varying numbers of nodes. We leave its answer for future research.

Chapter 2

Background

Contents

Machine learning has undergone revolutionary changes over the last two decades. In this chapter we describe the movement from traditional machine learning to deep learning, and the place of irregular-domain deep learning.

2.1 Traditional Machine Learning

2.1.1 Initial Intuitions for Machine Learning

The purpose of machine learning is to find patterns from morasses of data where humans would otherwise falter or lag. With patterns one can make predictions about the world. The most basic predictor is linear regression [3, p. 11]. Linear regression involves finding a linear function $y = f(x)$ defined on a continuous variable x that best predicts another continuous variable y. This kind of modelling can be applied to stock-market price prediction, in which case x is time and y is price; or in medicine, where x is age and y is likelihood of developing a disease.

How do we find the best prediction? Suppose f has the general form $f(x) = w_0 + w_1 x$ (notice that this equation is equivalent to a line equation; we will frequently refer to this function as a line). The optimisation task is to find values for the coefficients w_i such that we minimise the 'incorrectness' of the predictions; i.e., we must find an f such that for every given data x and true output y, $f(x) - y \approx 0$. In English, we need a function that will yield the correct prediction for a given value.

This 'incorrectness' is usually measured as the distance of the predictions to the correct value. (In Figure 2.1 this distance is represented by the red lines between the 'line of best fit' and the data-points represented by dark-blue circles.)

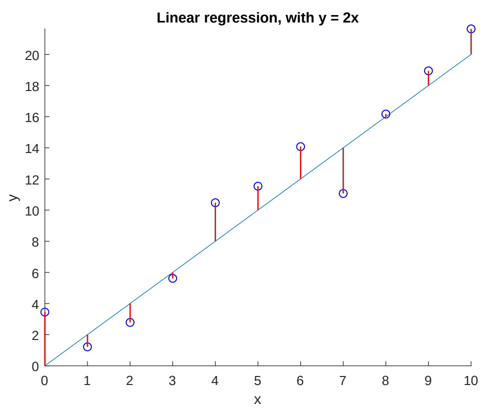


Figure 2.1: An example of linear regression. The light blue line is the 'line of best fit' through the data. The red lines between the line and the dark-blue circles, the true values for y, indicates the error of the predictions, given by the line of best fit. These errors would be measured by the sum of squared errors Equation 2.1.

How do we find w_i? The process is more mathematically rigorous than guessing w_i and happening upon the best values. At first the coefficients are randomly initialised. Then we measure the incorrectness of the function using a *cost-function* or alternatively a loss-function. The cost-function is employed in a process called *gradient-descent optimisation*.

For linear regression, the cost-function might be the least mean squares

(LMS), which uses the SSE—

$$\begin{aligned} E &= \frac{1}{2}\sum_{i=0}^{n}(f(x_i) - y_i)^2 && (2.1)\\ &= \frac{1}{2}\sum_{i=0}^{n}(w_0 + w_1 x - y_i)^2 = E_i && (2.2)\\ \text{where } n &= \text{the total number of data-points,} && (2.3)\\ x_i &= \text{the } i\text{th data-point, and} && (2.4)\\ y_i &= \text{the true output for } x_i. && (2.5)\\ & && (2.6)\end{aligned}$$

In essence, SSE severely penalises incorrect predictions. The square term means that the further away from the line the true output is, the greater the *cost*. Using the cost E we can adjust the coefficients—

$$\begin{aligned} w_i &= w_i - \alpha\frac{\partial E}{\partial w_i}E && (2.7)\\ \text{where } \alpha &= \text{the learning rate.} && (2.8)\end{aligned}$$

Gradient descent is performed iteratively until the loss falls below some predefined threshold. (Gradient-descent optimisation is not necessary when the data is non-singular; in these cases, there is a closed-form solution for the weights [3, p. 12].)

There are a few issues that come to mind when considering linear regression. Firstly, the data may be more complex. What if there is more than one dimension of input data? In the simplest case, this only requires a definition of the above functions, substituting $\mathbf{x}$ for x and $\mathbf{w}$ for w_i [4, p.105], such that—

$$\begin{aligned} f(\mathbf{x}) &= \mathbf{w}^\top\mathbf{x} = y, \text{ and} && (2.9)\\ E &= \frac{1}{2}\sum_{i=0}^{n}(y - \mathbf{w}^\top\mathbf{x}). && (2.10)\end{aligned}$$

There is also the issue of which data we train the algorithm over. In machine learning, a dataset is usually split into two sets: the *training set* and *testing set*. The training set is usually larger than the testing set. In linear regression, as in any other technique, we train the model on the training set and assess the accuracy of the predictions on the testing set. The error rate is used to assess the *generalisability* of the model (the error rate is also known as the *generlisation error*).

Equation 2.9, regardless of the dimensionality, is only suitable for data that is linear, however. Problems arise when we consider data that is non-linearly related to y (Figure 2.2). It is possible to transform the data using a non-linear kernel. But even if this transform were made, there is the overriding problem of dimensionality. If the dimensionality of data is too great, then the problem is not an appropriate model but tractability: The number of weights in a model increase linearly with dimensionality, and each additional weight is an additional optimisation problem. When the number of dimensions in the data is high, optimisation problems become exceedingly difficult [4, p.152]. These problems are part of the phenomenon termed *the curse of dimensionality* [3, pp. 22–27].

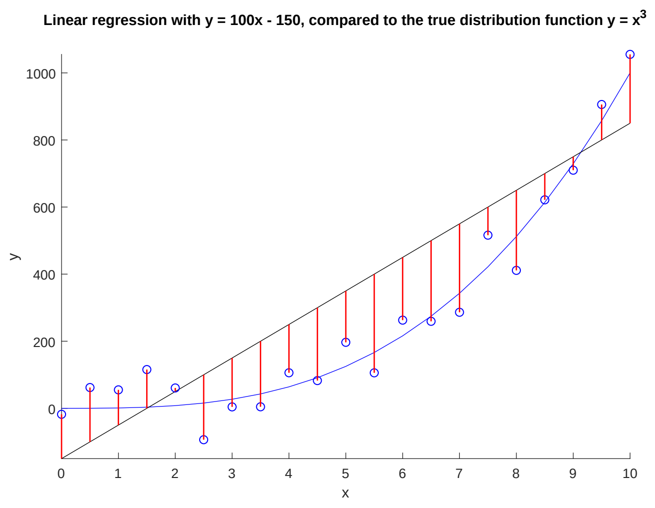


Figure 2.2: Linear and non-linear data compared. Linear regression is best suited to data that exhibit linear relationships with respect to an input x. The straight, black line above indicates the line of best fit that might be given for the data indicated by the dark-blue circles. The dark-blue line describes the true relationship of x and y.

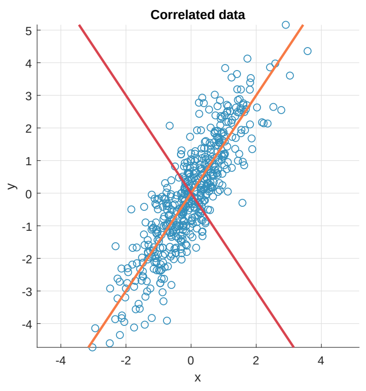
2.1.2 Dimensionality Reduction and Domain Experts

The curse of dimensionality can be mitigated by reducing the dimensionality of the data. Before we can reduce the data, however, we need to know which dimensions or *features* contain the most information. Some dimensions may contain useless information, characterised by low variance.

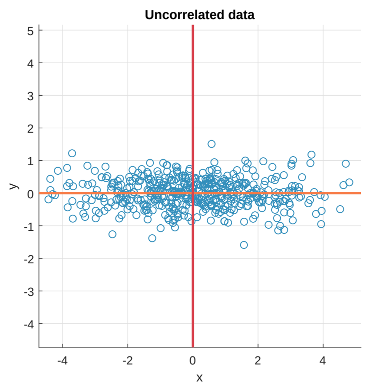
One technique for discovering the important dimensions is principal component analysis (PCA). Suppose we have a matrix $X \in \mathbb{R}^{n \times m}$ where n is the number of samples and m is the number of features. The singular-value decomposition (SVD) of X has the form

$$X = UDV^{\top}. \tag{2.11}$$

D is an m-by-m diagonal matrix of eigenvalues, and U and V are the n-by-m and m-by-m orthogonal matrices that span the column space and row space of X respectively [3, p. 66], also referred to as the left and right eigenvectors respectively. The diagonal entries of D are in descending order, and correspond to the eigenvectors in U and V. Each eigenvector describes the degree of variation in each eigenvector; this allows us to retain the first c components that account for a given percentage of the data's variation. The first c components are therefore called the *principal components* (Figure 2.3).



(a) $\Sigma = \begin{bmatrix} 1.0 & 1.5 \\ 1.5 & 3.0 \end{bmatrix}$



(b) $\Sigma = \begin{bmatrix} 3.7 & 0.0 \\ 0.0 & 0.2 \end{bmatrix}$

Figure 2.3: The left plot is randomly generated data correlated according to the covariance beneath. The right plot is the data in the left plot projected into eigenspace. The data on the right is obtained by taking the right eigenvector from the decomposition $X = U\Lambda V$ and performing matrix multiplications on the data. The result is that the principal directions of variation in the correlated data, given by the orange and red lines in the first figure (a), are aligned to the cardinal axes, as in the second figure (b). This is the function performed by PCA known as *decorrelation*.

PCA is not always suitable, however; like linear regression in Section 2.1.1, it does not work when the data is non-linear. In such cases, one might look elsewhere for a non-linear technique, such as kernel PCA [5].

In other cases it may not be enough to be equipped with a good method. Data may be so large or complex in some domains that they require specialist knowledge to understand the data's dynamics. In these cases a *domain expert* may be consulted about how best to digest or compress data.

Take EEG signals as an example. In this work we use the Temple University Hospital EEG Corpus (TUH-EEG); it contains thousands of millions of seconds of data on patient electroencephalography (EEG) scans, with a substantial quantity of samples containing seizures labelled by qualified doctors [6]. Determining whether an EEG scan contains a seizure is a difficult process due to the variability of seizures [7] and the necessity for information from a number of sources that builds a picture of the evidence over time 'until an accepted threshold is achieved' [8]. An algorithm would have to reduce the data down to some manageable size while also remaining representative of the original data.

The job of the domain expert is not simply to guide dimensionality reduction, however. To build a seizure-detecting algorithm would require a comprehensive understanding of the heuristics doctors use to determine whether a seizure is occurring; and furthermore an understanding of how these heuristics manifest themselves in data and the dynamics of the variables in the data.

2.2 Deep Learning

As discussed in Section 2.1.2, domain experts were once brought in to help understand data and create algorithms that could validly perform on the data tasks that would otherwise be carried out by an expert.

Domain experts have their disadvantages, however. The machine-learning techniques that rely on domain experts to propose dimensionality reduction strategies cannot capture 'the variability and richness of natural data [. . .] mak[ing] it almost impossible to build an accurate recognition system entirely by hand' [9]. Deep-learning techniques could achieve state-of-the-art results on handwritten digit recognition—which demonstrated, for the authors of the paper [9], that 'better recognition systems can be built by relying more on automatic learning, and less on hand-designed heuristics'.

Deep-learning techniques were initially sidelined, but gained traction with the development of the general-purpose graphics processing-unit (GPGPU) to the point where they are now considered to have 'dramatically improved the state-of-the-art' [10]. The term 'deep learning' encompass a wide range of neural-network techniques from multi-layer perceptrons (MLPs) and feed-forward networks to convolutional neural networks (CNNs) [9], [11] and long short-term memory (LSTM) [12]. The basic unit of any neural network is a perceptron. (*Perceptrons* [13] is an early comprehensive, theoretical treatment of perceptrons.)

2.2.1 From Perceptrons to Multi-Layer Perceptrons

A perceptron takes a weighted summation of its input $\mathbf{x}$ and passes it through an activation function (Figure 2.4). One activation function is the sigmoid function, which takes a value z and outputs a value in the range $(0, 1)$ across an infinite domain (Figure 2.5). This structure is similar to the linear-regression model presented in Section 2.1.1 with a constraining function. A perceptron with a linear activation function $f(x) = x$ can therefore model any linear regression problem. Though introducing the non-linearity allows us to extend our model beyond linear problems, such that it is able to 'understand the interaction between any two input variables' [4, p. 165].

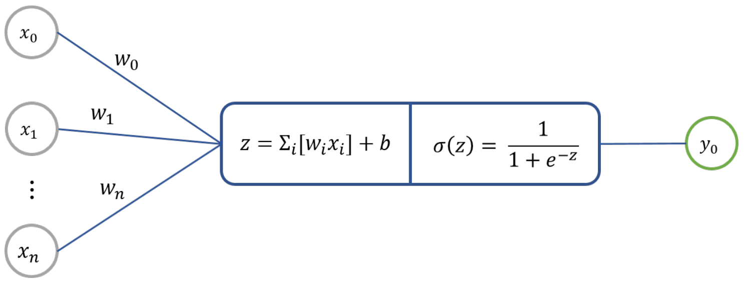


Figure 2.4: A perceptron takes a sum of weighted inputs, adds a bias b and passes it through some activation function, the sigmoid function σ in this case (Figure 2.5). Note that for each connection there is an associated weight. Each node between the input layer (the grey nodes) and output layer (the green node) has an associated bias. This is better shown in Figure 2.6.

Multiplying these perceptrons means we can learn many functions simultaneously. We can increase the capacity to an even greater extent by adding additional layers; these are called *hidden layers*. Suppose we have two perceptrons in the first hidden layer and a third perceptron in a second hidden layer, which takes the output of the first hidden layer as its input (Figure 2.6). Further imagine the input to the input layer is a picture. The two perceptrons in the first hidden layer are designed to learn different tasks: one learns whether there is a smile in the picture; the other learns whether there is a pair of eyes in the image. The perceptron in the second layer takes the output of the first layer (of both perceptrons) and makes a prediction about the existence of a face in the input: if the output value is 0.5 or greater, there is a face; if lower, there is no face. The input data is correspondingly labelled 1 if there is a face and 0 otherwise. (This is just a toy example. The functions are more basic than this; they look for low-level structures. This model would likely fail for a face-recognition task for the additional reason that the linear mappings of the perceptrons are

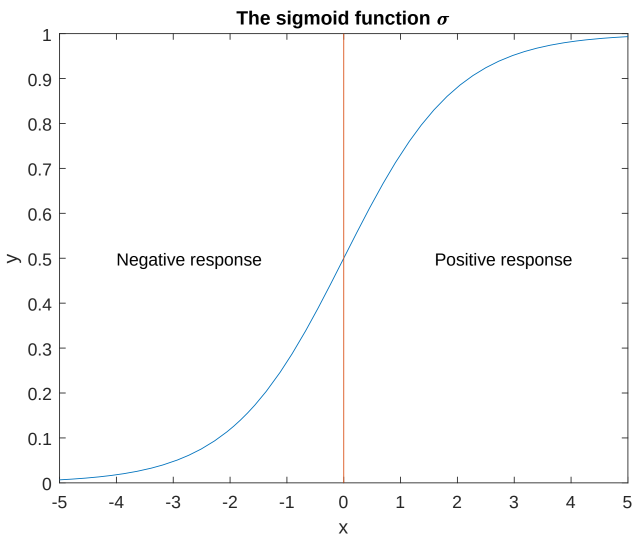


Figure 2.5: The sigmoid function $\sigma(x) = 1/(1+e^{-x})$ takes a value x and computes a value in the range $[0,1]$ over an infinite domain. The function asymptotes at 0 and 1.

not translation invariant—an issue whose solution we will discuss in the section on convolutional neural networks.)

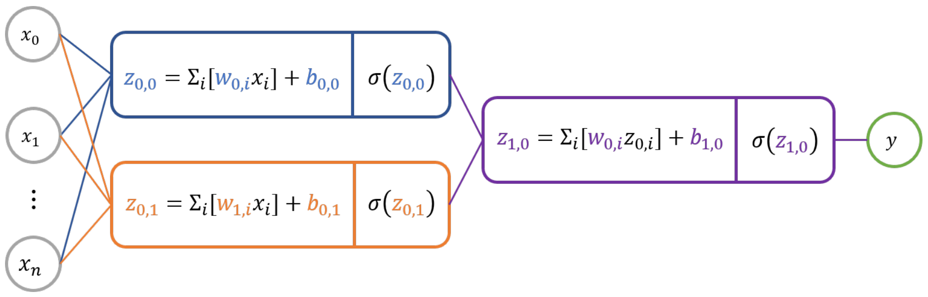

x_0
x_1
$\vdots$
x_n
$z_{0,0} = \Sigma_i[w_{0,i}x_i] + b_{0,0}$
$\sigma(z_{0,0})$
$z_{0,1} = \Sigma_i[w_{1,i}x_i] + b_{0,1}$
$\sigma(z_{0,1})$
$z_{1,0} = \Sigma_i[w_{0,i}z_{0,i}] + b_{1,0}$
$\sigma(z_{1,0})$
y

Figure 2.6: The first layer of a multi-layer perceptron is the input layer, indicated by the grey nodes; the final, output layer is the final layer indicated by the single green node. All intermediate layers are referred to as *hidden layers*. Each jth node in the hidden layer has its own set of weights $w_{i,j}$ for each ith node in the previous layer; each node also has its own bias $b_{k,j}$.

The output of the second layer in this scenario represents something like the certainty of the machine's prediction—which we want to maximise. After measuring the error of the model over a training set, the weights in the network (the w_i's in Figure 2.6) can then be adjusted to the degree of error indicated by the cost-function, most commonly the cross-entropy (Figure 2.7). In back-propagation, a process for gradient optimisation in neural networks [11], this process is iterated until the cost stabilises and the model can be optimised no more.

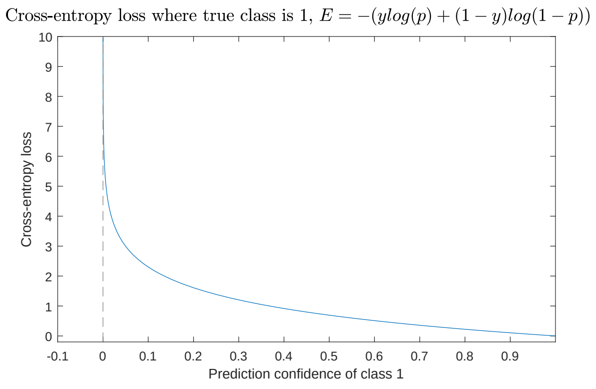


Figure 2.7: The cross-entropy loss function penalises an incorrect prediction more severely the more incorrect the prediction is. The x-axis of the plot above is the confidence of a model that an observation is class 1, if the observation's true class is class 1.

One can build much wider and deeper models in this way—i.e., models with more perceptrons or *nodes* per layer and more layers overall—and indeed this is often the case. Complex data requires either a complex function or a multitude of simple functions to approximate the complex one. Universal approximation theorem states that a sufficiently deep neural network with a linear output layer and at least one hidden layer with a 'squashing function' can approximate any Borel measurable function [14], i.e., 'any continuous function on a closed and bounded subset of $\mathbb{R}$' (see Goodfellow, Bengio, and Courville [4, p. 194–197] for further explanation of the theorem).

MLPs have their limitations, however. As shown in Figure 2.6, immediate layers are fully connected. As the number of perceptrons grow, the number of connections grow exponentially; consequently the number of weights grow too. This was the limitation pointed out at the end of Section 2.1.1: every additional weight is an additional optimisation problem. Too many weights is still a problem today, but it was an even greater problem before the development of GPGPUs when the limited parallelisability of CPU programmes placed the greatest constraint on performance [4, p. 441].

The MLP's architecture also means that it is susceptible to translated input. To return to the example of face recognition, in the MLP above, the perceptron that is trained to find eyes in the input image should ideally respond to eyes *anywhere* in the image. The linear map does not permit this, as it is forced to

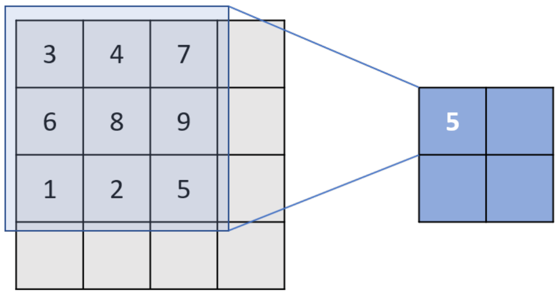


Figure 2.8: Convolution kernels iterate over the rows and columns of an input (the grey grid) performing some operation along the way. The operation can either be linear or non-linear. The above example is a 3-by-3 convolution kernel over a 4-by-4 grid calculating the average of the 9 values within its boundaries.

learn an eye in a fixed location.

If eyes appeared in the same set of pixels in every image, this would not be a problem; but images of fixes vary significantly with respect to the relative location of eyes. Yet the problem should be no harder; the eye looks the same no matter where it is—an example of what is referred to as *statistical stationarity*. The linear maps in perceptrons are unable to take advantage of this statistical property because they consider the image as a whole, not as its parts. CNNs on the other hand are able to respond to and exploit statistical stationarity—which leads us to our next topic.

2.2.2 Convolutional Neural Networks

Convolutional neural networks (CNNs) were first presented by LeCun, Boser, Denker, et al. in 1989 [11] in application to handwritten ZIP code recognition. Instead of a linear map (Section 2.2.1), each layer after the input layer would be the convolution of several compact kernels over the input (Figure 2.8). The compact kernels were able to learn sub-structures in the input of the layer. This meant that unlike perceptrons, CNNs are able to leverage statistical stationarity [15]. It also cuts down on the number of weights required, easing considerably the overall optimisation problem.

CNNs were set aside for a long time, however, and in their place random forests algorithms (RFs) and support-vector machines (SVMs) were used. CNNs came to the fore after the 2012 International Large-Scale Visual Recognition Challenge (ILSVRC) after a convolutional neural network trounced the second-place, non-CNN architecture by 11.1 percentage points with a top-5 error rate of 15.3% [16]. Every year since until the final competition in 2017, a CNN has come in first place in classification tasks (Table 2.1).

Zeiler and Fergus [17], the winners of ILSVRC 2013, were able to improve

Year	**Top-5 Error Rate (%)**	**Paper**
2012	15.3	[16]
2013	11.2	[17]
2014	6.7	[18]
2015	3.6	[19]
2016	3.1	No paper
2017	2.3	[20]

Table 2.1: The winners of the International Large-Scale Visual Recognition Challenge challenge in order of year. Each year a convolutional neural network (CNN) of some kind was the winning architecture.

upon the design of the previous year's winners after drawing some insights from the previous year's winning architecture. They sought to understand why CNNs work so well and developed a technique called *deconvolution* to understand how Krizhevsky, Sutskever, and Hinton's [16] model worked. They learnt that it was sensitive to high and low-frequency information and insensitive to mid-range frequencies; and they implicitly compute the spatial relationships of artefacts within images. Furthermore, invariance to scale and translation improves with the depth of the model; CNNs are not invariant to rotations of non-rotationally symmetric images; the depth of a model is more crucial to its precision than the size of a given individual layer; changing the size of fully connected layers contributes little to a model's precision; increasing the size of internal convolution layers gives a boost to the model's precision; and lastly increasing the size of both the middle and final fully connected layers results in overfitting.

2.3 Irregular-Domain Deep Learning

While CNNs work well on images and videos, the validity of extending them to topologically irregular data is questionable. Take the problem of human-action recognition (HAR). There are a number of ways of describing the movement of a person through a scene—using optical-flow images [21] and action maps obtained from scene flows [22]. In recent years, the widespread availability of cheap motion-tracking hardware such as Microsoft's Kinect has led to the more frequent use of skeletal maps [23] in HAR research. Skeletal maps exhibit an irregular topology as its nodes are (1) not fully connected and (2) where connected, pairs of nodes are not equidistant with other pairs of nodes.

In order to apply conventional techniques to skeletal maps, the data must be embedded in Cartesian space or correlate vaguely related features [24], [25]. But this has a questionable validity as it involves discarding the spatial relations between vertices and channel data.

Instead of embedding the irregular data in Cartesian space, Edwards and Xie

[26] proposed the skeletal models as graphs and using graph convolutional techniques to learn appropriate models for human actions. Such techniques are part of a larger class of deep-learning techniques, termed 'geometric deep learning' [27]. In this section we focus solely on graph deep-learning techniques. First, however, we will look at the mathematical background to irregular domains, or non-Euclidean geometries.

2.3.1 Non-Euclidean Geometries

Euclid's *Elements* established what we now know as Euclidean geometry as a set of definitions, axioms and postulates. The definitions defined points, lines, planes, surfaces, volumes, etc. Axioms were assertions of observable facts of geometry, 'some inherent attribute that is known at once to one's auditors—such as that fire is hot'; for example, 'two equal straight lines will coincide with each other' [28, p. 142].

Much debate was aroused by Euclid's fifth postulate. It states—

> if a straight line falling on two straight lines makes the interior angles on the same side less than two right angles, the straight lines, if produced indefinitely, will meet on that side on which are the angles less than the two right angles. [28, p. 150]

This is the definition of a pair of non-parallel lines. In the fifth century CE, seven hundred years after *Elements* first appeared, mathematicians were still contesting its status as a postulate[1] [29, pp. 88-9]. The disagreement centred on the postulate's independence from the other postulates and axioms. Its independence was firmly established through Nikolai Ivanovich Lobachevsky's work on hyperbolic, hence non-Euclidean geometry. This development in geometry 'lead to enormous practical consequences' that Einstein incorporated into his General Theory of Relativity [29, p. 89].

Consider a regular triangle. It has three equal sides that subtend three angles which add up to 180°. If we project this shape onto the surface of a sphere, the shape is deformed, such that the interior angles sum to greater than 180°. Instead of a sphere, we could consider an oblate spheroid like the earth; the fact of deformation would remain. (See Figure 2.9 for a visual illustration.)

Many geometries can be generated by changing the Fifth Postulate. Non-Euclidean or irregular spaces don't necessarily take the form of a sphere; there are other geometric structures to consider, such as manifolds [27]. One of the most prominent representations of such geometries in deep learning is the graph.

[1] Proclus: 'This ought to be struck from the postulates altogether [...] the converse of it is proved by Euclid himself as a theorem.' [28, p. 150]

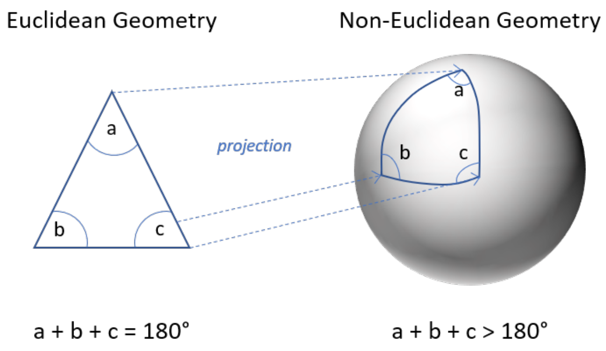


Figure 2.9: A projection of a Euclidean shape onto a non-Euclidean geometry. Left is regular, Euclidean geometry. Right is irregular, non-Euclidean geometry—more specifically, an elliptic geometry. The regular triangle (*left*) has been projected onto the surface of a sphere (*right*). Projection does not change the length of the sides; but it changes the size of the interior angles.

2.3.2 Deep Learning on Graphs

Graphs are a form of data representation, consisting of vertices joined by edges. Mathematically it is defined as an object $G = \langle V, W \rangle$ where V is the set of n vertices and W is the n-by-n weighted adjacency matrix, where 0 indicates no edge between a pair of vertices and a positive, real number indicates an edge-weight (Figure 2.10).

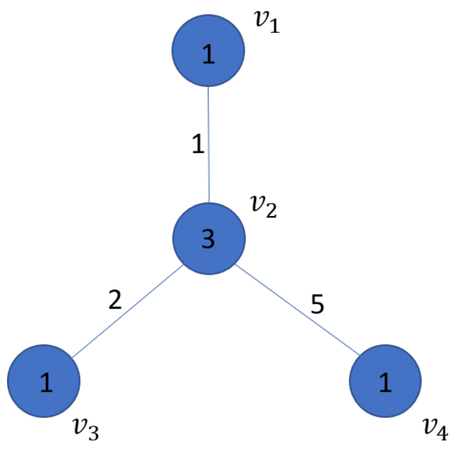


Figure 2.10: The above shows a graph with 4 vertices labelled v_1 to v_4 and 3 edges. Each pair of vertices with an edge has an associated weight, given as a number on the line. The degree for each node is given as a number in the vertex; the degree corresponds to the number of incident edges.

The weight matrix thus represents the graph in the spatial domain. Though its spatial irregularity means we cannot perform conventional spatial convolution described in Section 2.2.2. Instead, Bruna, Zaremba, Szlam, et al. [15] suggested a decomposition of the weight matrix into eigenvectors, which can then be used in Fourier transformations of graph data. According to convolution theorem [30, p. 163] a point-wise multiplication in the spectral domain is equivalent to a convolution in the spatial domain. A convolution can therefore be implemented in graphs by linearly filtering the Fourier-transformed graph data. Edwards and Xie [31] later improved on the method simply by transposing the Fourier basis in the Fourier transform.

Implementing graph convolutions in the way described above has its disadvantages, however. The operation is computationally expensive: it is an $O(|V|^2)$ for the forward Fourier transformation and $O(|V|^2)$ on the way back [32]. Defferrard, Bresson, and Vandergheynst [33] suggest using Chebyshev polynomials. Levie, Monti, Bresson, et al. [34] proposed a further improvement using CayleyNets. For the purposes of this work, we have decided to use Fourier-transform method presented by Edwards and Xie [31] as the paper also presents a method for graph pooling using the same mathematical objects.

One example of a data domain that is represented well by a graph is the skeletal model [26], but there are many other domains with existing work in graph deep learning. Monti, Bronstein, and Bresson [35] proposed a recommender system using graphs. Traffic data is represented well by graph structures and has been used in application to traffic prediction [36]. Molecules can also be represented as a graph to be used to learn molecular fingerprints [37].

Dimension-reducing operations are common in regular CNNs, as they provide a means for generalising and compacting information as the network is forced to learn only the most discriminant information. Graph convolutions, however, do not reduce the dimensionality of input. In regular CNN, the pooled data are second-order statistics, such as means, maxima or minima. It also provides nice properties, such as robustness to slight translations in the input data [4, p. 335 *ff.*]. The standard pooling operations used in CNNs cannot be used due to the graph's property of topological irregularity. Instead one must use graph-based clustering. Three clustering methods to be found in the literature are Kron's reduction [38], Graclus multi-level clustering [39] and algebraic multigrid clustering (AMG) [40]. All three methods have been used in graph deep-learning applications [26], [31], [33]; but AMG is most suitable for graph autoencoding. In contrast to Kron's and Graclus, AMG provides projection and restriction matrices to coarsen and uncoarsen the graph respectively, and it does not assume anything about the topology of the graph; instead it works on the weight matrix. There is, however, an element of randomness in the coarsening of the graph using AMG, meaning that two algorithm will not yield the same coarsening.

The above are examples of graph convolution applied to discriminative tasks. Little work, however, has been conducted into generative techniques. Wang, Pan, Long, et al. [41] proposed a marginalised graph autoencoder to cluster citation data, though they treat each node of a graph as a separate sample in its own right; we are interested in graph-wise representation learning as opposed to vertex-wise learning. Litany, Bronstein, Bronstein, et al. [42] presented a graph-convolutional variational autoencoder (GCVAE) to complete partial body-shapes. Their technique does not involve representation learning, however; it learns a latent space which is searched at inference for a shape that minimises the dissimilarity between the input and output. Guo, Nejati, and Cheung [43] leverage information present in the adjacency matrix of the graph to learn a transform of the graph-signal channels; this transform is then fed to a fully connected neural network to learn a low-dimension representation of the graph data.

2.4 Summary

In Section 2.1.1 we described the necessary intuitions for a basic machine-learning technique called linear regression. We saw that the curse of dimensionality means that such linear techniques are not suitable for learning from high-dimensional data. In Section 2.1.2 we described two ways of addressing this: an unsupervised method such as principal component analysis; and the consulting of 'domain experts' who understand their domain's data comprehensively and can advise on how best to reduce the data.

In Section 2.2.2 we introduced convolutional neural networks (CNNs) which were built upon earlier work on multi-layer perceptrons summarised in Section 2.2.1. CNNs were promoted for their ability to learn data representations automatically with the belief that handcrafted feature-extractors and human-designed heuristics could not capture the complex interactions that were possible with non-linear models. The crowning moment for CNNs was the 2012 International Large-Scale Visual Recognition Challenge in which a CNN came in first place. That a CNN had won every year until the final competition in 2017 is emblematic of the position of CNNs in modern machine learning.

In Section 2.3 we showed how CNNs have a limited application beyond regular domains. Existing techniques require irregular data to be embedded in regular domains, in the process losing some crucial spatial information. A whole new area of deep learning called geometric deep learning has been built to address this problem, built upon the concept of 'non-Euclidean geometries', described in Section 2.3.1.

Graphs have a number of uses that are briefly explored in Section 2.3.2. We define a graph and consider a few data domains that exhibit irregular topologies. While much work has been conducted into discriminative techniques, comparatively little has been conducted into generative techniques.

The focus of this work is thus on generative deep-learning tasks in irregular domains, particularly on graphs. In Chapter 3 we apply graph-convolutional techniques to a modified MNIST dataset. In Chapter 4 we apply the techniques defined in Chapter 3 to EEG scans.

Chapter 3

Graph Convolutional Autoencoder

Contents

Simply stated, the task for autoencoders (AEs) is to replicate its input. Though we do not expect nor want a perfect replication of the input. The most frequent purpose of the autoencoder is to learn a 'smaller' representation of the input, a representation with a dimensionality that is far smaller than the input dimensionality. Such autoencoders are referred to as *undercomplete* [4, p. 499] and performs what is referred to as *encoding.* In reducing the data like this, we *expect* some loss—but only so much loss as is 'acceptable' [44].

Put in formal mathematics, we would like to learn a pair of functions f and g. Given an input $x \in \mathbb{R}^n$, f should encode the data in some lower dimensional space, such that $f(x) = h \in \mathbb{R}^m$ where $m \ll n$ and $m, n \in \mathbb{N}_{>0}$. The function g should decode h such that $g(h) = \tilde{x} \approx x$ (Figure 3.1).

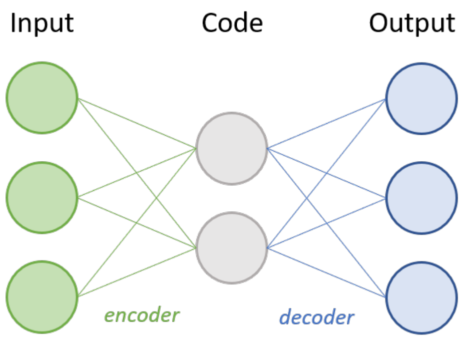


Figure 3.1: The structure of an autoencoder. The input layer is larger than the hidden layer or *code*. The output is the same size as the input. The encoder must learn an appropriate compression of the input data such that the encoding contains enough information from which the decoder can reconstruct the input.

As we have seen in Section 2.2.2, the convolutional operators in convolutional neural networks (CNNs) and convolutional autoencoders (CAEs) alike assume spatial regularity in the input data, which means a convolutional autoencoding method is not applicable in the case of spatially irregular data. In this chapter we present a method for encoding spatially irregular data, drawing upon the methods presented by Edwards and Xie [31]. We evaluate the method on a modified MNIST dataset.

This work has been published in the Proceedings of the IEEE International Conference on Image Processing for 2018.

3.1 A Modified MNIST Dataset: Building the Graph

The MNIST dataset consists of 60,000 28-by-28-pixel greyscale images of handwritten digits 0 through 9. The typical task on MNIST is to classify correctly the digits. The property of being handwritten means that the a learning algorithm cannot simply memorise a single shape for each character; the handwritten digits deviate so far from the 'standard' shape that the algorithm must learn their distribution and make probabilistic inferences from the learned distributions. In an unsupervised task, the algorithm would learn the optimal clustering of the dataset.

The pixel-grid of the images is regularly spaced: each pixel is equidistant to its von Neumann neighbours (see Figure 3.2) giving rise to a regular grid.

Hence the dataset itself yields graphs with regular topologies. In the interest of evaluation of our technique, we would like irregular data. We can *irregularise* this data in the manner described in Edwards and Xie [31] which we describe next. N.B.: We interchangeably use the terms *node* and *vertex* throughout the work.

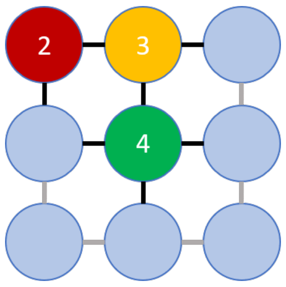


Figure 3.2: Von Neumann neighbours. Consider each circle in the above 3-by-3 grid as pixels in a 3-by-3 image. Each pixel's von Neumann neighbours are those immediately upward, downward, leftward and rightward. Consider these three examples, which describe the three general cases: The green circle has four neighbours, one in each direction. The yellow circle has only three neighbours since there is no upward neighbour. The red circle has only two neighbours, one downward and one rightward. This patterning gives rise to a regular grid. This figure may also be considered as an image-as-graph visualisation, where each vertex corresponds to a pixel, the black edges describe the von Neumann neighbours, and the numbers in the circles represent adjacencies.

Irregularisation of the MNIST dataset involved (1) creating a graph from an image; (2) selecting and removing a random subset of vertices from remove from the graph; and (3) removing those vertices and corresponding features from the dataset. (We have visualised the process on one example image in Figure 3.3.)

Creating a graph. The images in the dataset were 28-by-28 pixels large. Constructing the vertices V for this image is simple; each pixel is simply interpreted as a graph vertex. The adjacency matrix A records whether a pairs of pixels are von Neumann neighbours (see Figure 3.2). The adjacency matrix is equivalent to the weight matrix W as each edge has a weight of 1 as all von Neumann neighbours are equidistant. This comprehension of the images yields a *regular* graph. This only needs to be done once for the whole dataset, as the graph is fixed for every example in the dataset.

Selecting and removing a random subset of vertices. In the regular graph there are 784 vertices corresponding to the 784 pixels in the original image. We chose to exclude 84 random vertices, like Edwards and Xie [31].

The particular number of nodes to remove is an arbitrary choice, motivated by three concerns. The first is information loss. If we removed all but one node, we would leave ourselves with two little information to learn on; the intuition follows for if we keep all but one, all but two, etc., until we reach a point where our intuition is less clear. Essentially, we want enough data to discriminate on.

The second concern is *which* nodes we choose to remove. The salient information in MNIST images is focussed around the centre. If we choose only to remove points from the centre of the image, we would lose the most important information. The best way to select nodes is random selection where each node has an equal probability of being chosen.

We cannot simply randomly choose 84 vertices, however: our method requires the graph to be non-partitioned. Therefore we cannot allow for vertex-selections that lead to orphans or segments (see Figures 3.3(d) and 3.3(e) for disallowed vertex-selections). The method was implemented by reducing the ith row and column from the weight matrix for each corresponding ith vertex.

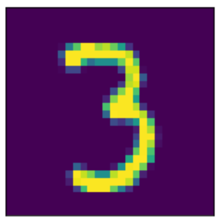
The final concern is the choosing of enough nodes to motivate the use of graph convolution. One might ask what the difference is between an irregular graph and an image with zero-valued entries. A proper treatment of this issue is beyond the scope of this thesis, however. We believe that 84 nodes is a sufficient number to disrupt the regularity of the grid (Figure 3.3).

Removing corresponding features from the dataset. As the corresponding row and column for each removed vertex was removed from the weight matrix, so was the ith pixel in the image. An array stored the numbers of the vertices to keep. The dataset was then vectorised along the feature axes

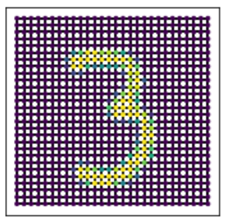
Once irregularised, the dataset was split into a 55,000-large training set and a 5,000-large validation set.

3.2 Methodology

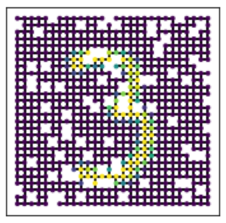
As explained in Chapter 2, CNNs exploit useful statistical properties in data, such as statistical stationarity and compositionality. It is not possible to operate on irregular domains in the way conventional way: the lack of a 'translation property' [15] in irregular domains makes convolution impossible; the lack of



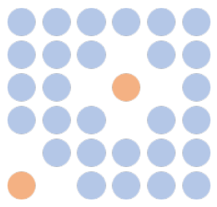
(a) Original image.



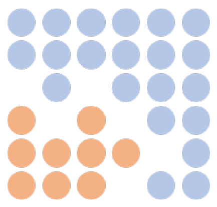
(b) Regular graph.



(c) Irregular graph.



(d) Orphaned vertices.



(e) A partitioned graph.

Figure 3.3: The process of irregularisation. Each image from the dataset (a) is comprehended as a graph (b). A random subset of vertices are removed to produce an irregular graph (c). When we choose the random subset of vertices, we must avoid situations where individual vertices are orphaned or the graph is partitioned. In the cases above, six vertices are arbitrarily removed leading to two orphans in (d) and a partition in (e).

this property also prohibits straightforward pair-wise pooling or clustering of the data (see Figure 3.4).

3.2.1 Convolution

From the convolution theorem described by Bracewell [30], we can induce that convolution in the spatial domain is identical to a multiplication in the spectral domain [45, p. 163]. This permits defining convolution in the spectral domain rather than the spatial domain.

Suppose we have a graph G defined on a set of n vertices V and weight

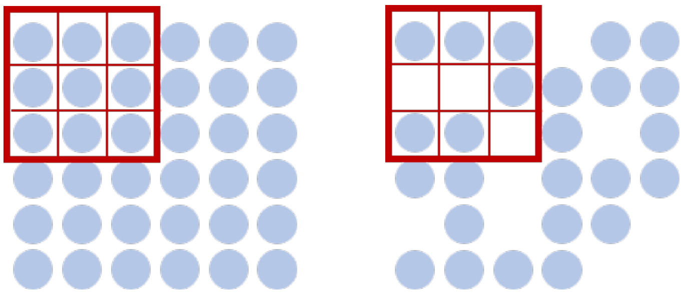

(a) Translation property. (b) No translation property.

Figure 3.4: The translation property visualised. Informally speaking, the translation property exists for a data-domain if a window can be slid unit-by-unit across the entire space with a constant number of pixels/vertices in its boundaries. In the drawing (a) above, the window, represented by a red square, can be placed anywhere within the boundaries of the data, represented by the vertices, and contain nine vertices. If the data is irregular, however, as in (b), the window will not always contain 9 vertices wherever it is put.

matrix $W \in \mathbb{R}^{n \times n}$. We can compute the Laplacian matrix of the graph—

$$\mathcal{L} = D - W,$$
$$\text{where } D_{i,j} = \begin{cases} \sum_j A_{i,j} & \text{if } i = j, \\ 0 & \text{otherwise,} \end{cases}$$
$$\text{where } A_{i,j} = \begin{cases} 1 & \text{if } W_{i,j} > 0, \\ 0 & \text{otherwise,} \end{cases}$$

where A is the adjacency matrix, a binary matrix where each entry $A_{i,j}$ is 1 if there is a connection between nodes i and j or 0 otherwise; and D is the degree matrix, a diagonal matrix where each entry $D_{i,i}$ holds a value corresponding to the ith node's degree, the equivalent of the sum of the ith row or column of the adjacency matrix A.

In decomposing the Laplacian matrix, we obtain the graph's eigenvectors U and eigenvalues Λ:

$$\mathcal{L} = U^\top \Lambda U.$$

The eigenvectors allow us to perform an operation that is analogous to a Fourier transformation [46]. The eigenvectors can therefore be considered to form the Fourier basis. We can therefore define the graph's Fourier transformation. Suppose $f \in \mathbb{R}^n$ is some n-dimensional graph signal in the spatial domain. The Fourier transform of the signal is

$$\hat{f} = U^\top f,$$

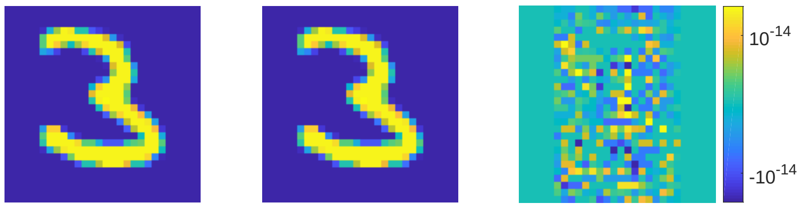


Figure 3.5: Artefacts introduced by discrete Fourier transformations [30, p. 281]. On the left is a random image from the MNIST dataset. Fourier-transforming that image forward and backward gives the centre image. There are no perceptible differences. The right image shows quite how much noise is introduced; it shows the normalised per-pixel difference between the left and centre images.

where $\hat{f} \in \mathbb{R}^n$ is the graph signal in the spectral domain. The reverse Fourier transformation is defined as

$$f = U\hat{f}.$$

The inverse Fourier transformation on a computer does not f, however, as computer-implementations of the Fourier transformation are discrete. Owing to the discrete nature of computers, in practice, a discrete forward and inverse Fourier transformation of f yields a slightly-modified $\tilde{f}$ (*viz.* [30], p. 281, 'Is the Discrete Fourier Transform Correct?'). As the sampling of discrete Fourier transforms is 'not sufficiently closely spaced to represent high-frequency components' (*ibid.*), artefacts, such as ringing [45, pp. 169–175], are introduced to the data merely by the discrete forward and inverse Fourier transformations (see Figure 3.5). The discrete reverse Fourier transformation is thus defined as

$$\tilde{f} = U\hat{f}.$$

As mentioned above, the convolution theorem states that a convolution over the spatial domain is identical to a multiplication in the spectral domain. If we consider a convolution kernel h in the spatial domain, there is a vector $k \in \mathbb{R}^n$ a filter in the spectral domain, such that

$$f \star_G h \equiv \hat{f} \odot k, \tag{3.1}$$

where $\star_G$ denotes spatial convolution on a graph and $\odot$ denotes the Hadamard product between two equal-length vectors.

In the same way that filters in CNNs do not need to be the same size as its layer's input, e.g. [9], there does not need to be the same number of learned weights as there are components of the spectral signal. Observe above that the Fourier-transformed graph signal $\hat{f}$ is the same length as the non-transformed graph signal f. Suppose the signal-domain filter is given by $k \in \mathbb{R}^n$. The filter k is the same length as $\hat{f}$; for each element $k_i \in k$, there is a corresponding element in $\hat{f}$ that k_i alone multiplies. The vector k therefore acts as a set of scalars of $\hat{f}$. These scalars could be learned, with the aim of representing a certain pattern present in space. These scalars can be considered as weights and so analogous to the kernel weights in CNNs. If, however, we do decide to learn these weights, then each weight is a variable, and therefore an optimisation problem. Thus, as the size of the graph signal increases, the optimisation problem increases linearly. This introduces scaling problems.

Rather than learn a vector of weights k, equal in length to the size of the graph signal, we instead learn a much smaller vector of weights, and interpolate these weights up to the size of the signal. For instance, instead of $k \in \mathbb{R}^n$, we have $\kappa \in \mathbb{R}^m$ such that $m \ll n$. The full-size vector of weights k is thus distinguished from the much smaller vector of *tracked* weights κ; instead of learning the weights, we learn the tracked weights. (Henceforth we refer to the tracked weights simply as weights.) The size of κ is an arbitrary choice, like the size of kernels in CNNs. The interpolation matrix Φ is a choice to be made by the model-architect; in our case, we use bicubic interpolation.

A convolution in the spectral domain is therefore,

$$f' = U(\hat{f} \odot \Phi\kappa) = U(U^\top f \odot \Phi\kappa),$$

where U is the Fourier basis; $\Phi\kappa$ is the interpolated signal-domain filter; the convolved graph signal f'.

In the graph convolution layer in our deep-learning model, an arbitrary number of filters may be used yielding an equal number of output maps [31]. These output maps are equivalent conceptually to the output maps of CNNs The previous layer may have more than one map, too. If the number of input maps is $\mathbf{i}$ and the number of output maps is $\mathbf{o}$, the equation for each output map $0 \leq o < \mathbf{o}$ is—

$$f'_o = U \sum_{i=0}^{\mathbf{i}} \left[(U^\top f_i) \odot (\Phi\kappa) \right]. \tag{3.2}$$

where f_i is the input map.

In this chapter we experiment with the number of weights to study its effect on encoding quality. The number of nodes in the modified MNIST dataset is 700, with the number of weights in vector κ varying from 10 to 60 in increments of 10.

3.2.2 Pooling

In feed-forward networks and CNNs, reducing the dimensionality of the input data forces the model to learn more general features. For similar purposes, AEs and CAEs contain pooling layers in order to encode in a smaller dimension than the input; these kinds of autoencoder are termed *undercomplete* [4, §14.1]. The convolution layers in CNNs and CAEs implement another form of dimensionality reduction as kernel strides.

Both kernel strides and conventional pooling exploit a spatial regularity that is not present in irregular domains. Graph convolutions are implemented by multiplication in the spectral domain (3.1), so there is no kernel to stride from the outset. (Besides, the definition of graph convolution (3.2) requires a fixed graph.) Conventional pooling techniques require a regularity, or as Bruna, Zaremba, Szlam, et al. [15] term it, a 'multiscale dyadic clustering of the grid', i.e., the ability of element-pairs to be merged somehow (minimum, maximum, average) at multiple levels.

Conventional techniques are not possible. Fortunately there are many techniques for pooling or *coarsening* graphs. Unfortunately it is an NP-hard problem with a concomitant abundance of literature the problem [47].

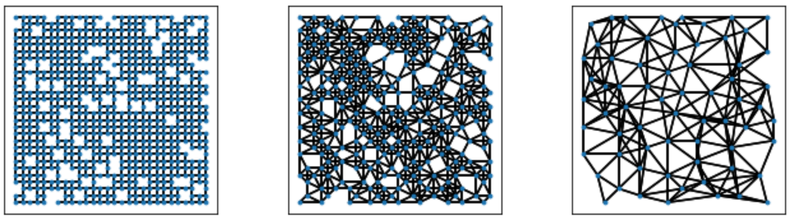
Coarsening is defined formally as this: Suppose we have a graph $G = \langle V, W \rangle$ where $|V| = n$. A coarsening scheme reduces G by some cut-metric to give a coarser graph $\hat{G} = \langle \hat{V}, \hat{W} \rangle$ where $|\hat{V}| = \hat{n}$, $\hat{n} < n$ and $W \in \mathbb{R}^{\hat{n} \times \hat{n}}$. As stated in the literature review Section 2.3, we are using algebraic multigrid (AMG) to coarsen the graph. The algorithm is described in pseudo-code in Algorithm 1. Clustering is performed on the weight-matrix, where each entry $W_{i,j}$ describes the edge-weight between vertices i and j. The algorithm selects a random vertex from the graph uniformly at each step (as not to privilege any one node over any other; Line 6). For each edge and each vertex incident to the current vertex along that edge, the edge-weight is added entry in an array corresponding to the incident vertex (the array to which we add the edge-weights is called *column_sums* in Algorithm 1). A vertex is only observed so long as its current value (stored in *column_sums*) is less than the threshold. If its value is less than the threshold, it is also marked to be kept (Line 8). The algorithm works through every node in the graph (thus it is a greedy algorithm).

For our model, the restriction and projection matrices are precomputed; the pooling levels and hence graph sizes are thus fixed in our model. In our case there are two pooling levels. Before pooling there are 700 vertices in the graph; after level-one pooling there are 276 vertices; after level-two pooling there are 66 vertices. (See Figure 3.6 for a visualisation of these graphs.) Two different runs of the algorithm won't necessarily yield the same result due to

Algorithm 1 The algebraic multigrid (AMG) coarsening algorithm [40]. This algorithm gives a single coarsening given a set of vertices V, a normalised weight matrix W and a coarsening factor α.

```
1:  procedure AMGPooling($V$, $W$, $\alpha$)
2:      $n \leftarrow |V|$
3:      $random\_vertices \leftarrow$ randomise_order(array($n$))
4:      $column\_sums \leftarrow$ zeros($n, 1$)
5:      $vertices\_to\_keep \leftarrow$ falses($n, 1$)
6:      for $vertex$ in $random\_vertices$ do
7:          if $columns\_sums[vertex] \leq \alpha$ then
8:              $vertices\_to\_keep[vertex] \leftarrow$ True
9:              for $col = 0 \dots n-1$ do
10:                 $column\_sums[col] \leftarrow column\_sums[col] + W[vertex, col]$
11:             end for
12:         end if
13:     end for
14:     $P \leftarrow W[:, vertices\_to\_keep]$
15:     $R \leftarrow P^\top \times (1/sum\_columns(P))$ // element-wise division
16:     $\hat{V} \leftarrow V[vertices\_to\_keep]$
17:     $\hat{W} \leftarrow W[vertices\_to\_keep, vertices\_to\_keep]$
18:     return $P, R$
19: end procedure
```

the aforementioned randomisation (see Line 3 in Algorithm 1). In addition to uniformly randomly selecting nodes for study, the algorithm tends to merge vertices based on how strongly related they are, corresponding to the strength of their connection, i.e., their respective weights in the weight matrix [47, p. 194]. In our model we used the same graph-coarsenings across all experiments.



(a) No pooling. (b) Pooling level one. (c) Pooling level two.

Figure 3.6: Two-level AMG pooling on an arbitrarily irregular graph with a coarsening factor $\alpha = 0.05$. The above example uses the same graph obtained for Figure 3.3(c). With no pooling there are 700 vertices (a). After level-one pooling there are 276 vertices. After level-three pooling there are 66 vertices.

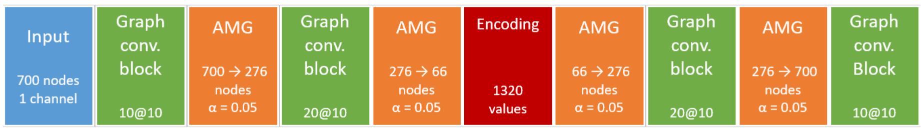


Figure 3.7: A visual representation of the autoencoder model for the MNIST data, read from left to right.

3.3 Results

All experiments ran on a Windows 10 machine equipped with an NVIDIA GeForce 1080Ti. The model code was written in Python with the TensorFlow library. Figure 3.7 is a visual representation of our model. A *graph-convolution block* refers to a set of repeating *convolution units*. A *convolution unit* refers to a single graph-convolution layer, a rectified linear unit as an activation function and a dropout layer. The AMG block is a AMG restriction to left of the Encoding block; to the right, it is an AMG projection. A *filter* is a single set of weights, equivalent to a set of weight in a CNN. The weights in the encoder and decoder are not shared. The model receives as input the spatial signals of the graph for the MNIST dataset. There is only one channel since the data is grayscale; thus the input data consists of 700 node-values.

The model was optimised using the Adam optimiser with a 0.001 learning rate and a 256-sample batch size. The AMG coarsening factor was 0.05 and a dropout drop-probability of 20%. By default, the number of weights was 10; the number of output maps in the first convolution block was 10; the number of output maps in the second convolution block was 20; and the number of convolution units per block was 1—but these changed according to the experiment.

We ran experiments on four parameters of the network to check the encoding quality: the number of convolution units per block; the number of output maps in the first convolution block; the number of output maps in the second convolution block; and the number of weights in the graph convolution unit. By conducting these experiments we hoped to learn something about the effect of these parameters on autoencoding quality in irregular domains.

3.3.1 Experiment 1: Convolution Units per Block

The number of convolution units per block varied from 1 to 5. The reconstruction error over the validation set over 100 epochs for each variation of the number of convolution units is plotted in Figure 3.8. The optimal number of convolution layers, i.e., the encoding that yielded the lowest reconstruction cost, was 1 convolution layer. Increasing the number of units beyond one yields a lower-quality encoder.

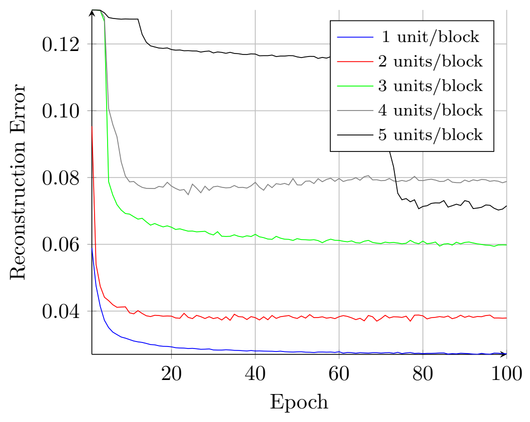


Figure 3.8: The reconstruction error over 100 epochs on the validation set as the number of convolution units in the convolution blocks is varied. The optimal number of convolution units is 1.

3.3.2 Experiment 2: The Number of Output Maps in the First Convolution Block

The number of output maps in the first convolution block was varied from 10 to 50. Figure 3.9 plots the reconstruction error on the validation set over the 100 epochs of training for each independent variation. Increasing the number of output maps increased the fidelity of the encoding. The overall effect was, however, unremarkable.

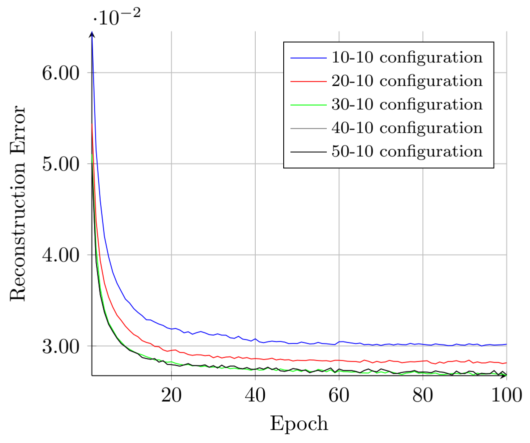


Figure 3.9: The reconstruction error over 100 epochs on the validation set as the number of output maps in the first convolution block was varied from 10 to 50.

3.3.3 Experiment 3: The Number of Output Maps in the Second Convolution Block

As in Experiment 2, the number of output maps in the second convolution block was varied from 10 to 50. Figure 3.10 plots the reconstruction error on the validation set over the 100 epochs of training for each independent variation. The overall effect of increasing the number of output maps in the second convolution block was greater than that for Experiment 2. Though the increase was still unremarkable.

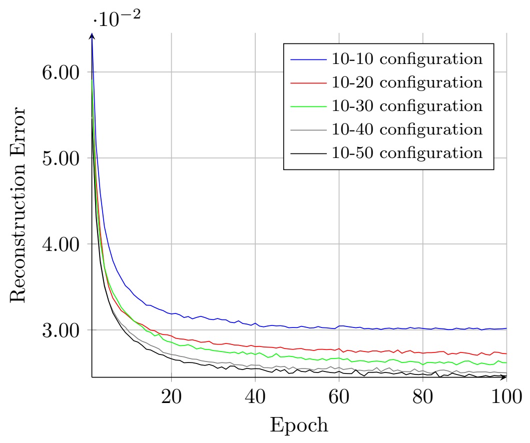


Figure 3.10: The reconstruction error over 100 epoch on the validation set as the number of output maps in the second convolution block was varied from 10 to 50.

3.3.4 Experiment 4: Weights

The number of weights in the convolution blocks was varied from 10 to 60. The number of weights was changed for every convolution unit. The upper bound on the number of weights was 60 to account for the size of the coarsest graph (see Figure 3.6). The reconstruction error over the validation set over 100 epochs for each independent variation is plotted in Figure 3.11. Increasing the number of weights in the convolution units improved encoding quality, but in the same order of magnitude as observed in Experiment 2 and 3.

3.4 Discussion

The experiments demonstrate that our convolutional autoencoder appropriately encodes and learns representational features. Given how simple the MNIST dataset is, it is no surprise that the model performs best with a single graph

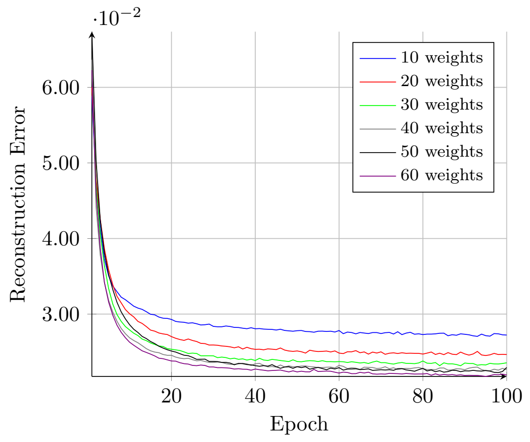


Figure 3.11: The reconstruction error over 100 epochs for each variation in the number of weights in the convolution units.

convolution unit (Experiment 1), and does not improve much with additional output maps (Experiment 2 and 3) and weights (Experiment 4). Increasing the number of convolution units and output maps not only yielded in one case worse results and in the other insubstantially better results, they also increased the time to train the algorithm. Every additional convolution unit increased training time on average by a factor of 1.68. Every additional 10 output maps in the first convolution block increased the time to train by a factor of 1.34; every additional 10 output maps in the second block increased the training time by a factor of 1.47.

The data shows that our model is particularly sensitive to the number of graph-convolution layers in each convolution block. The number of filters on the other hand does not affect the reconstruction cost significantly; though the cost does decrease with an increased number of filters. In increasing the number of weights, we expected the autoencoder cost to reduce as more weights means a coarser filter, which should capture more local information. Indeed, this is what is borne out in the results; but the effect is not significant. Nonetheless, the results show that our graph-convolutional autoencoder is suitable for encoding graph-data.

3.5 Summary

In this section, we presented a model that encodes irregular data. Our dataset was a modified version of MNIST, where we represented the image data as graphs and removed a random subset of nodes, yielding a domain with an irregular topology. Conventional techniques would not work with such data, so we had to define convolution and pooling operations. Convolution in the irregular domain was achieved by a multiplication in the spectral domain of the graph. Pooling was implemented using the algebraic multigrid (AMG) coarsening algorithm.

With our modified MNIST dataset and convolution and pooling defined on the graph, we created a model to encode the images from the modified dataset. We found upon experimentation that a simple model architecture yielded a high-fidelity encoding. This is most likely due to the simplicity of the MNIST dataset.

Chapter 4

Graph Convolutional Autoencoder Applied to EEG Data

Contents

In Chapter 3 we defined convolution and pooling on irregular domains, and built a model to learn representative features of graph data. In this chapter we will apply these techniques to another data-domain with greater significance—electroencephalography (EEG).

Electroencephalograms record brain activity in the form of electrical signals through the scalp of a subject and take the form of a cap. The 10/20 system recommended by the American Clinical Neurophysiological Society (ACNS) [48] has acted as the main standard for the design of EEG cap, but there are other systems available, too [49]. The ACNS standard defines 81 standard electrodes, positioned concentrically around a central node called Cz. (In the 10/20 configuration only 21 electrodes are used; see Figure 4.1 for a visual representation of these sensors. N.B.: These graphs are *not related* to the graphs we use for deep learning. Please see Section 4.2 to read how we constructed the graph for deep learning.)

Because of the spherical, non-Euclidean surface of the scalp and the regu-

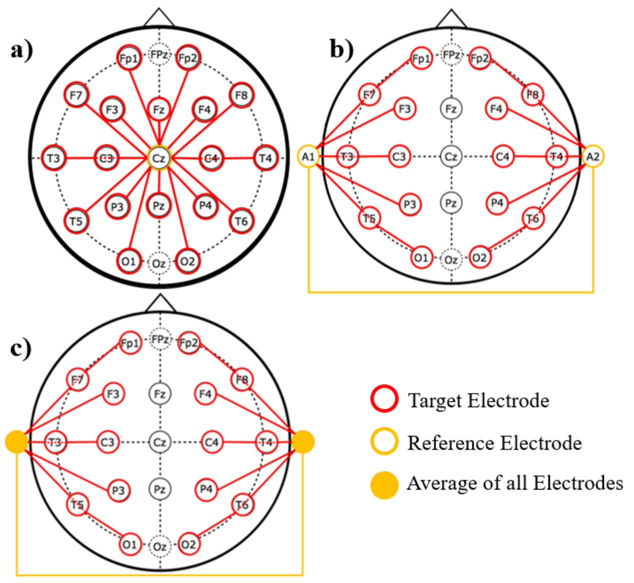


Figure 4.1: The three electrode reference systems used by the TUH EEG corpus. This figure is adapted from work by Lopez, Gross, Yang, et al. [50]. From that paper: ‘a) the Common Vertex Reference (C_z), b) the Linked Ears Reference (LE) and c) the Average Reference (AR)’. In this work we solely work on data using the AR system.

lar placement of electrodes across a non-Euclidean geometry, graph techniques should perform well on electroencephalographys data. In this chapter we present a graph-based convolutional autoencoder (GCAE) for encoding EEG data.

4.1 The Temple University Hospital EEG Corpus

The Temple University Hospital EEG Corpus (TUH-EEG) [6] is a massive, professionally labelled EEG dataset containing abnormal EEG data from over 25,000 studies. The subset we are interested in for our work is the TUH-EEG Seizure Corpus, specifically version 1.2.1 [51].

The seizure dataset is already divided into disjoint training and testing sets. The training set consists of 264 unique patients, 78,838.0892 seconds of seizure data and 1,109,474.9108 seconds of background data. The testing data is smaller, with 50 unique patients, 58,322.3671 seconds of seizure data and 558,779.6329 seconds of background data. The training set is further divided into two disjoint training and validation sets. The validation set consists of 243 patient-sequences selected from the original training set.

The dataset is 18 gigabytes large. The EEG data resides in European Data

Format (EDF) files, and the labels for the data reside in separate label files. For each EDF file there are two label files: one contained many labels for different kinds of seizures for multi-class classification tasks; the other dataset contained binary labels indicating which parts of the EEG data recorded seizures. These labels constitute the ground-truth. Each time-step is labelled either seizure or non-seizure.

To simplify our task, we restricted our use of the dataset to the reference system with the greatest amount of data, the AR reference system (see Figure 4.1), which accounts for 51.5% of the dataset [50].

In preparing the data for our model, we had to prepare the data by—

Subsampling the data and selecting for common channels. The lowest frequency in the dataset was 250 Hz, so all sequences were subsampled to this frequency. The greatest common set of electrode channels was 17 large—C3, C4, P3, P4, O1, O2, F7, F8, T3, T4, T5, T6, Fz, Cz, Pz, A1, A2. By subsampling the EEG data to the same, common frequency, equally-sized data-windows would correspond to equal time-lengths in different sequences. Selecting the greatest common set of channels would ensure (1) a fixed graph for all data and (2) the inclusion of the greatest amount of variance in the data. This subsampling of the data follows the example of previous work [7], [50], [52], [53].

Labelling the time-steps in the data. The label files specified where seizures started and ended in each sequence. The start and stop times were given in seconds rather than time-steps or indices. To convert these times to indices was trivial. The index of the start time-step/index is given by

$$i_{start} = \lfloor t_{start} \cdot freq \rfloor.$$

The index of the end of the seizure was calculated similarly:

$$i_{end} = \lceil t_{end} \cdot freq \rceil.$$

With these indices calculated, we labelled each time-step seizure/non-seizure according to whether it fell between i_{start} and i_{end} inclusively.

We compiled a list of the indices for the time-steps corresponding to seizures and another for those corresponding to non-seizures for each sequence of EEG data. During training, these indices were sampled according to a uniform distribution with replacement. It is possible that a time-step could be sampled twice. However, due to the size of the dataset, this was very unlikely. For this reason we did not make an effort to ensure it did not happen. We also did

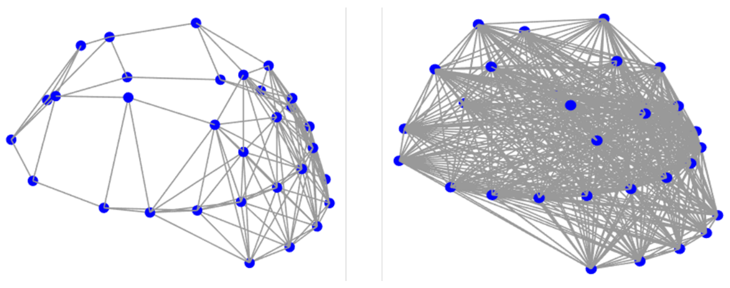


Figure 4.2: A visualisation of an EEG cap with two different heuristics to determine the connectivity of the graph. On the left, vertices were connected only if they were at least within one-fourth of greatest Euclidean distance between all nodes in the graph. On the right-hand-side graph, all nodes are connected; i.e., it is fully connected. N.B.: This graph is similar to the one we used for our EEG data, but *it is not the same.*

not see any problem with the model being trained on a batch containing one already-observed sequence while the rest had not. It would have been even less likely that the entire batch consisted of already-observed sequences.

The data was not normalised for three reasons. Firstly, the measurements of EEG signals occupy the same range of numbers. Secondly, it raises the question of what exactly we normalise against. The maximum of the patient? or the maximum of the entire dataset? Thirdly, we found that batch-normalisation hampered the performance of the model to such a degree during training that it was finally dropped.

4.2 Graph Construction

Before we can perform any graph deep learning on the EEG data, we must create the graph to represent the spatial locations of the 17 electrodes. The edges should ideally describe some real relationship between two sensors in the graph, such as their physical distance, so a unit-weight for the edges is not useful. (Figure 4.2 for a visualisation of an EEG cap with two different connectivities.) An additional question is *how many* connections. It may also describe other, real attributes. Rui, Nejati, and Cheung [54] [54] describe a few alternative 'brain connectivities' that could be applied in constructing the graph; namely, functional connectivity and effective connectivity. The easiest approach, requiring little expert knowledge, is to use standardised measurements used in constructing EEG caps to calculate physical distance. As we will see, we have no radii measurements to use. The second-best measure is therefore the angular relationship between two sensors.

The ACNS standard supplies spherical coordinates for each of the sensors, recording their location relative to the top of the scalp [48, p. 58]. Unfortunately the radii of the locations was not specified—presumably because the radii change so drastically between patients since peoples' heads are not all one size. The proximity of the sensors to one another can be inferred from their spherical distances. These distances were therefore used to populate a weight matrix. The spherical coordinates for each electrode is given by a pair of values, the elevation θ and the azimuth ϕ. The electrode Cz is assigned an elevation and azimuth of 0; all other electrodes are located relative to this. The origin is located at the centre of the sphere. The ACNS standard specifies the values in degrees, so they were first converted to radians before calculating spherical distances.

First we needed an equation to measure the spherical distance. We know how to calculate the Euclidean distance in Cartesian space:

$$\text{distance} = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}. \tag{4.1}$$

We can convert between Cartesian and spherical coordinates using the following equations:

$$\begin{aligned} x &= r\sin(\theta)\cos(\phi), \\ y &= r\sin(\theta)\sin(\phi), \\ z &= r\cos(\theta). \end{aligned} \tag{4.2}$$

By substituting the equations in 4.2 for x, y and z in 4.1 and rearranging the resultant equation, we get an equation for the calculation of spherical distance:

$$\text{distance} = \sqrt{r_1^2 + r_2^2 + 2r_1r_2[\sin(\theta_1)\sin(\theta_2)\cos(\phi_1 - \phi_2) + \cos(\theta_1)\cos(\theta_2)]} \tag{4.3}$$

As this equation requires radii measurements that we do not have, we cannot use this equation to compute the edge-weights for the graph. But the angular component of the above equation,

$$\text{angular\_component} = \sin(\theta_1)\sin(\theta_2)\cos(\phi_1 - \phi_2) + \cos(\theta_1)\cos(\theta_2)^1. \tag{4.4}$$

does measure the angular divergence or convergence of two spherical coordinates. If two sensors lie on top of one another, the angular component yields 1; if they on opposite sides of the unit sphere, the value is -1. The closer two sensors, the higher the value of the angular component. Assuming a unit-sphere, we can use this angular component to measure the relatedness of two sensors.

[1]The equation for spherical distance was given at `https://math.stackexchange.com/questions/833002`

This equation for the angular component cannot in its current state be used for calculating edge-weights, as negative values are not permitted to be used in the weight matrix. Instead we can translate and scale the range from $[-1, 1]$ to $[0, 1]$ by first adding one and dividing the whole by 2:

$$W_{i,j} = \frac{(\sin(\theta_1)\sin(\theta_2)\cos(\phi_1 - \phi_2) + \cos(\theta_1)\cos(\theta_2)) + 1}{2} \tag{4.5}$$

Each entry of the weight matrix for the graph is then defined by a transformed calculation of the angular component (4.4).

The lack of radii measurements means it is not possible to visualise the graph accurately. This does not imply the weight matrix has no bearing on reality, however. Assuming a perfect sphere, the weights used to define the edges of the graph correspond inversely to the angles between sensors. The scalp is certainly not a perfect sphere but we believe the two are close enough for our method to be valid.

4.3 Results

All experiments ran on a Windows 10 machine equipped with an NVIDIA GeForce 1080Ti. The model code was written in Python with the TensorFlow library. Figures 4.3 to 4.5 are diagrams of the three models we compare experimentally. Dropout was implemented with a probability of 50%. The leaky ReLU layers were implemented with an alpha of 0.2, the default value in Tensorflow. The graph convolutions used a number of tracked weights equal to the ceiling of the square-root of its number of vertices; there were 17 vertices in the graph convolution, so 5 tracked weights.

Each model was optimised using RMSProp [55] over 200 epochs with a 8-sample batch size and an initial learning-rate of 0.001 for 10 epochs that decayed linearly over the remaining 190 epochs to 0.0001 (using numpy's linspace function). This learning-rate was settled upon after experimenting with learning-rates orders of magnitude apart; we found that 0.001 yielded a stable learning-rate that didn't learn so quickly or so slowly that nothing was learned. In any case, the choice of learning-rate is an arbitrary choice, and this choice cannot be said with certainty to be the most optimal. The decaying learning-rate allows for smaller adjustments in the learning and is intended to allow the model to fine-tune its parameters.

Each batch consisted of one-half seizure samples and one-half non-seizure samples. The samples were 1,024 time-steps long, corresponding to 4.096 seconds of EEG data at 250 Hz. The samples were randomly chosen from the data using the index-lists extracted before training (Section 4.1).

There are two axes present in the data: a temporal axis and a spatial axis. As mentioned in above, there were 1,024 time-steps for each sampled sequence and 17 nodes. Therefore there are 17,408 data-points per sample. The graph deals with the spatial structure of the data, i.e., the 17 node. The seizures are assumed to have a stationary distribution along the temporal axis. For this reason we decided to use one-dimensional convolutions to learn temporal patterns in the data. This is an assumption that underlies machine-learning approaches proposed in previous work [7], [50], [52], [53].

As a basis for comparison, we implemented three different architectures with analogous structures. The basis for comparison is the one-dimensional (1D) model described in Figure 4.3. In this model the only convolution is along the temporal axis, disregarding the spatial axis. The purpose of the 1D model is to assess what can be discriminated solely on the basis of what is present temporally. This can then be compared to versions of the 1D model where there exists some comprehension of spatial information. Overall this will permit us to assess and compare the contributions of the two axes.

The 1D model consists of three subnetworks: an encoder, a decoder and a classifier. The encoder and decoder subnetworks (the autoencoder subnetwork) is built of one-dimensional convolution blocks. Similarly with the convolution blocks in Section 3.3, the one-dimensional convolution blocks consist of a single temporal one-dimensional convolution layer; a leaky ReLU layer as an activation layer; followed by a dropout layer for more robust training. There are three of these blocks, each of which is followed by a temporal pooling 5 units long. The final pooling layer is followed by a single one-dimensional convolution and a leaky ReLU layer. We do not use a dropout layer at the end of the autoencoder since we will be using this data to train the classifier, within which there is already a dropout layer present. N.B.: The one-dimensional convolutions solely in the 1D model have 85 nodes/outputs, corresponding to 5 nodes for each graph-node in the input. This was intended to replicate the capacity of the 2D and graph models, described below.

The decoder subnetwork is a mirror-image of the encoder, except the final graph-convolution, one-dimensional convolution and leaky ReLU are not present at the beginning (see Figure 4.3). The weights are not shared between encoder and decoder. The classifier consists of two fully-connected layers of 100 neurones with leaky ReLUs and dropout layers in between the first two fully-connected layers. An additional, final layer consists of two neurones trained as a binary classifier. (Figure 4.3 describes the size of the layers at each step in a more comprehensible way.)

The two comparison models that we developed extend the 1D model. The first, which we name the graph model (see Figure 4.4), includes graph-convolution layers before each one-dimensional convolution in the encoder of the 1D model, and after each one-dimensional convolution block in the decoder of the 1D model. We decided not to include graph-clustering/-pooling layers since the number of nodes in the graph (17) is low. Each graph-convolution layer yielded five output-maps. The purpose of this model is to assess the contribution of graph-understanding to the performance of the model.

The two-dimensional (2D) model substitutes two-dimensional convolutions for the one-dimensional convolutions in the 1D model (see Figure 4.5). The kernel-size of the two-dimensional convolutions is 5-by-5, 5 units in the temporal domain (as in the 1D model) with 5 units in the spatial domain. In using these layers, we are making an incorrect assumption of the spatial relationship of the nodes, an assumption that is commonly made when dealing graphs in other problem-domains (*viz.* [24], [25]).

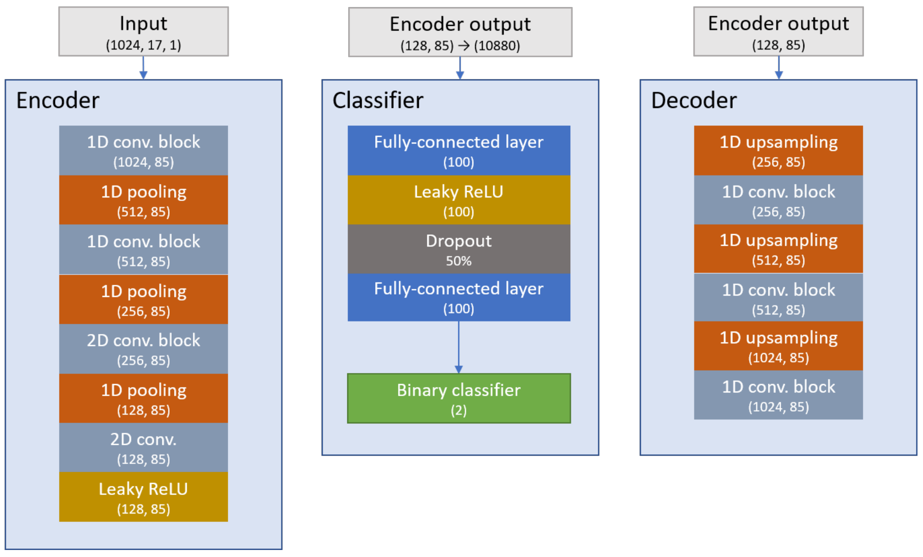


Figure 4.3: The one-dimensional-convolution (1D) model is nearly identical to the 2D model: all 2D convolutions are replaced with 1D convolutions with a receptive field 5 units long. N.B.: The output-dimensions of the one-dimensional convolution blocks are on the surface different to those in Figures 4.4 and 4.5. The nodes in the 1D model equal 85 to correspond to the 17 graph-nodes and 5 graph-channels in the 2D and graph models.

The Charbonnier loss [56] (4.6) was used as the cost-function for the autoencoders. It has a steep slope resembling L_1 loss for extreme values of $\tilde{x} - x$, with a curve resembling an L_2 loss around 0 due to an epsilon term ($\epsilon = 0.0001$; see Figure 4.6). The Charbonnier loss hence retains the curve around zero as

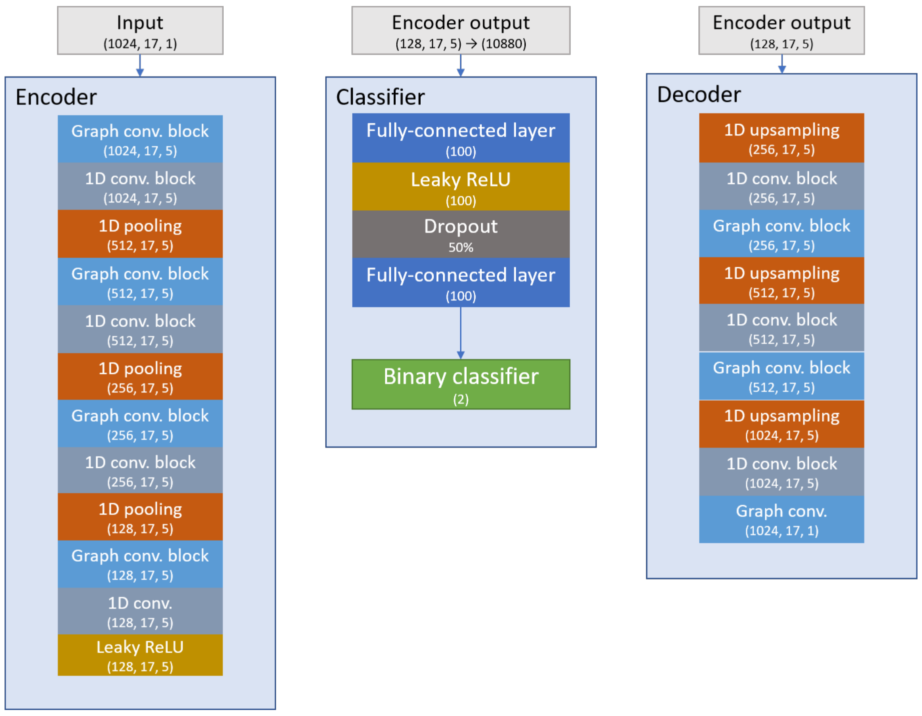


Figure 4.4: The graph model consists of a series of graph convolution blocks, one-dimensional convolution blocks and one-dimensional pooling. The values in the brackets state the output of each layer or set of layers. Each convolution block consists of a graph convolution, a leaky ReLU layer and a dropout layer. Similarly, each one-dimensional convolution block consists of a one-dimensional convolution, a leaky ReLU layer and a dropout layer. The final layer in the encoder does not contain a full one-dimensional convolution block; instead, there is only a one-dimensional convolution and a leaky ReLU. The decoder contains identical components to the encoder, with one-dimensional upsampling layers replacing the pooling layers. The final layer of the decoder is non-activated.

with L_2 loss and the straight edges beyond 0 as with the L_1 loss. For these reasons, optimising this function results in more stable training than L_1 and L_2 losses. In computing the cost, each time-step was considered an independent observation, so the loss was calculated based on the mean of all time-steps in all batches. The classification cost was measured using a sigmoidal cross-entropy function, as the output of the classifier part of the model is one unit large.

The models were optimised according to the sum of the autoencoder and classification costs. Balancing these costs is integral to the linear separation of data. The purpose is not classification; this is a generative approach and so it will not give competitive performance. The purpose is to learn which features

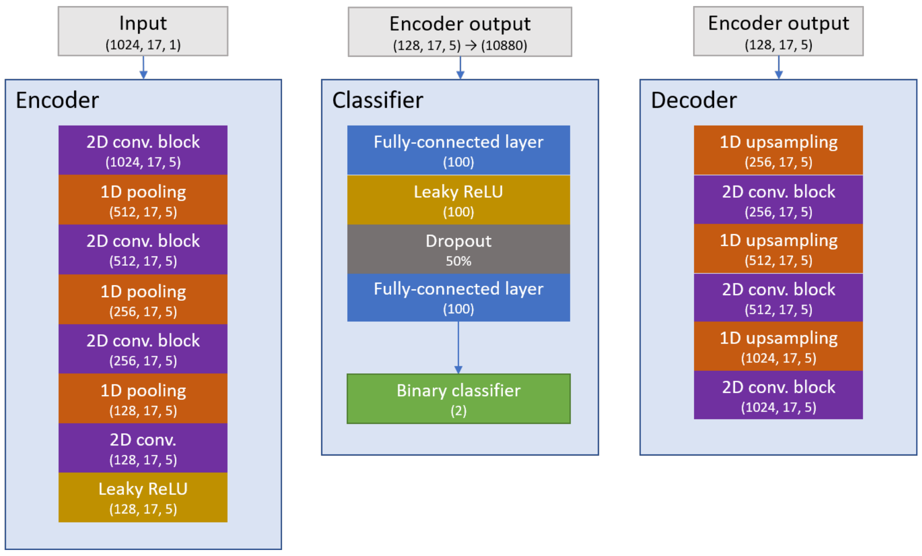


Figure 4.5: The two-dimensional-convolution (2D) model is similar to the graph model, except all graph blocks have been removed and replaced with 2D convolutions with a 5-by-5 receptive field.

best correspond to seizures. The classification accuracy is an assessment of the performance of the autoencoder, applied in learning to adjust the encoding in a way that increases the separability of the data. The two should be considered in conjunction.

$$E(x) = \sqrt{(\tilde{x} - x) + \epsilon^2} \tag{4.6}$$

Due to the size of the database, we could not run through the full dataset every epoch. Instead we randomly chose 60,800 EEG samples every epoch, half of which were seizure samples and half non-seizure. Over 200 epochs this result in a good coverage of the dataset: 6,080,000 seizure samples of a possible 6,254,608, and the same number of non-seizure samples from a total of 125,368,700. The samples are likely to overlap between any epoch, however; by samples, we refer to the time-steps upon which a 1,024-unit window of EEG data can be taken. The justification for this overlap is our preference for the model to become temporally invariant. It is harder to say the same thing is true for the non-seizure data. There is a far greater quantity of non-seizure samples available, so the likelihood of overlap is far lower in this case.

The validation and testing set are sampled likewise. From the validation set, 12,800 samples are taken every epoch. Over the 200 epochs of training,

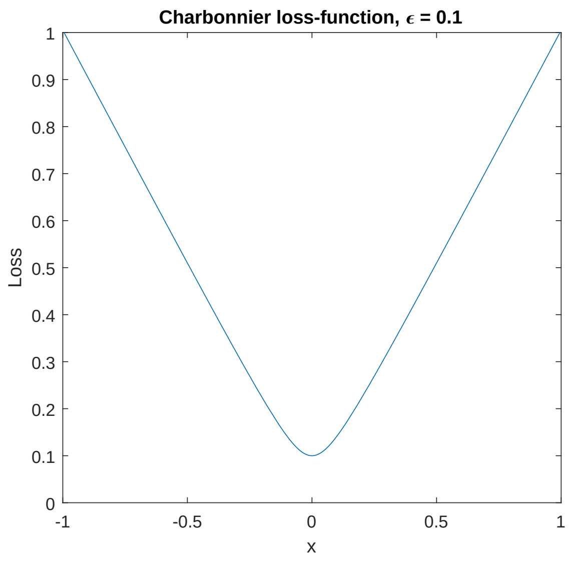


Figure 4.6: The Charbonnier cost-function combines the linear slope of an L_1 cost-function for values greater than zero, with a curve that reaches its minimum near zero, but never reaches $y = 0$ due to a small epsilon term (see Equation 4.6).

2,560,000 samples are made from a total of 30,424,383 possible samples with replacement; 1,467,058 of all possible samples are classified as seizures, from which 1,280,000 are sampled with replacement. Upon completion of training, 480,000 samples, half of which are classified as seizures, are taken from a total of 98,108,159 samples, 12,330,283 of which are labelled as seizures.

The key difference between our approach and others before us [7], [50], [52], [53] is that our approach does not train on extracted features; we use the raw EEG data to train our model. While we cut down on time by avoiding data-processing procedures, our model's performance is affected by the greater degree of noise; this corresponds to a poorer classification performance. Since the model learns on this raw data, it may pick up on structures these other techniques miss.

We expected the experimental data to show that the performance of the graph model is superior to the other two; the graph-understanding of the data should allow the model to learn the dynamics in the data. The three models were trained according to the same scheme over the same dataset. We assessed them on their accuracy and reconstruction errors. The validation errors were used to assess the training for generalisation errors, while the test errors will tell us of the performance on totally unobserved data.

The results present a mixed picture. The graph model is not distinctly better than the 2D and 1D models with respect to classification and encoding performance. The graph model performed generally better than the 2D and 1D

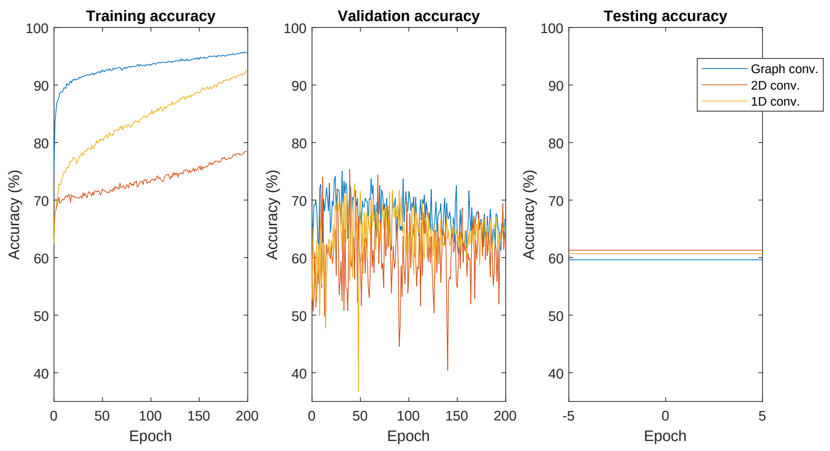


Figure 4.7: The results of the experiment on the three models for classification accuracy. The graph model far outperformed the 2D and 1D models on the training set. The graph model only marginally beat the 2D model.

models on the validation set; yet it was outperformed every so slightly on the testing set by the 1D and 2D models (Figure 4.7). Nonetheless, the performance differences were insignificant, and therefore it cannot be said that one method can be favoured over any other.

The encoding performance as reconstruction error or autoencoder cost does not give a clear result in favour of any of the models. Despite the 2D model performing better over the training set than the graph model, their performance was fairly even on the validation set; yet when it came to the testing set the 2D model had a slightly lower reconstruction cost (Figure 4.8).

Overfitting is the major problem here. The training accuracy gradually increases with the epoch, but the validation accuracy is erratic. The accuracy occasionally dips below 50%, indicating that the model is performing worse than mere guess-work. The reconstruction errors for the models also reveal problems. The models become only marginally better over time on the training data, and perform erratically on the validation set.

4.4 Discussion

The results of the experiment are not what we originally hoped. The graph convolutions (defined in Section 3.2.1) take into account the spatial relationships of the EEG channels. Two-dimensional convolutions make different assumptions about the spatial relationships in the data they operate on; applying them to domains where the assumptions no longer hold should result in poor results when compared to an approach like graph convolution that *does* take into account

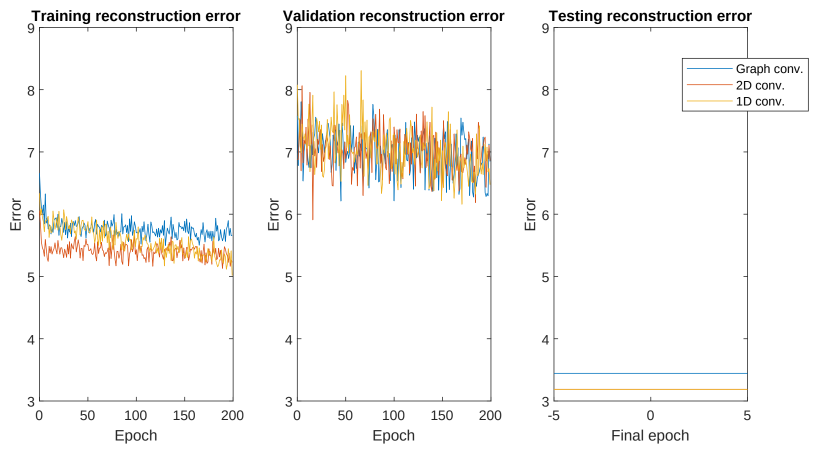


Figure 4.8: The results of the experiment on the three models for autoencoder cost. None of the models appeared to learn an encoding any better than the others. The reconstruction error on the testing set was better for the 1D and 2D models than the graph model. (Note that the 2D and 1D models had the reconstruction error on the test set.)

these spatial relationships.

There are several possible reasons for the graph convolution's on-par performance with the graph convolution. First of all, there's the architecture itself (Figure 4.4). Would graph pooling layers reduce the reconstruction error by forcing the data into a more compact encoding space? Architectures that use two-dimensional convolutions (unlike ours) pools the features in all directions in order to generalise features. We refrained from doing so in our code for fear that it would prevent the model from learning anything (time was at a premium). Also, should we have put a one-dimensional convolution and graph convolution before the first pooling layer in the decoder? The capacities of the encoder and decoder are imbalanced without it. Perhaps this imbalance is highly consequential to the performance of the autoencoder.

Secondly, perhaps there was some dependency in the temporal domain that would improve the separability of the classes? The 1D convolutions were intended to understand any salient information in the temporal axis; but they are not appropriate for learning time dependencies. A graph-based long short-term memory (LSTM) layer instead of a one-dimensional convolution layer would address this problem by learning any time dependencies; unfortunately, to the best of our knowledge, there are no examples in the literature of architectures using graph LSTMs.

Thirdly, the representation of graph using the measure stated in Equation 4.5 may not be adequate to represent the graph. Human skulls vary in size, radius, curvature, etc.; it is not possible to construct a graph that is representative of the whole population. Human-action recognition research is subject to the same troubles in constructing skeletal models [26].

To consider too is the fact that there are different ways to represent the connectivity of the brain from the outset. We mentioned in Section 4.1 that we decided on a structural representation—i.e., one that simply considers the relative distances between the sensors of the EEG cap. Rui, Nejati, and Cheung [54] mentions two 'brain connectivities': functional connectivity and effective connectivity. Further work into how one might represent a graph in these ways may yield more fruitful alternatives for graph representation; or, indeed, work on how to *learn* an appropriate graph representation.

Finally, there is the dataset to consider. Seventeen sensors were selected after searching for the greatest common set of sensors in the EEG data. Might more sensors have yielded a better encoding? The paper for the TUH-EEG Seizure Corpus [51] suggests that some of the seizures were labelled by doctors and others algorithmically. A possible alternative is the European Epilepsy Database (EED)[2]. The website advertises 40,000 hours of data from 250 epilepsy patients, 50 of which with intercranial recordings, and up to 122 channels. The dataset is expensive, however: a three-year license for 30 patient datasets costs 3,000€.

Given the results for reconstruction error (Figure 4.8), we think the greatest failure is the architecture. The lack of great distinction in the reconstruction error between the three methods suggests the problem is the architecture or the construction of the graph. The purpose of the 1D convolutions was to show that adding spatial comprehension to the network (the graph convolutions) or even invalid spatial comprehension (2D convolutions) would improve the encoding in some way, but the more valid method would improve the encoding to a greater extent. If this were the case, it would have shown in the reconstruction error *and* the accuracy. The fact that all three did not perform in radically different ways suggests that the method is flawed somehow. To understand why requires further research.

4.5 Summary

In this chapter we presented the results of an experiment comparing the graph convolution operation with a two conventional methods. The models were trained on a subset of the Temple University Hospital EEG Corpus consisting of seizures from 264 unique patients. We constructed a graph for the 17

[2]`http://epilepsy-database.eu/`

sensors using spherical coordinates specified in the American Clinical Neurophysiological Society's specification for electroencephalography (EEG) caps.

The results did not reveal any improvement in testing accuracy in the use of graph convolutions compared to two-dimensional convolutions. There may be a number of reasons for the poorer performance. The architecture may not be balanced; and graph pooling may increase encoding performance. A graph-based long short-term memory (LSTM) layer may also yield a better performance if it were used in the place of the one-dimensional convolutions; however, to the best of our knowledge, no graph LSTM implementations exist in the literature. Our graph representation of the EEG cap may itself be invalid, too (this is a research problem we outline in Section 5.2). Finally, the dataset itself may be of poor quality.

Chapter 5

Conclusions and Future Work

Contents

5.1 Conclusions

In Chapter 4, we sought to apply what we learned in Chapter 3 to a real task. We decided to apply our work to the classification of electroencephalography (EEG) scans. We described the Temple University Hospital EEG Corpus and used it to train the graph-based convolutional autoencoder (GCAE). As a point of comparison, we compared this model to a couple of analogous architectures one implementing two-dimensional convolutions and the other implementing one-dimensional convolutions. Unfortunately the graph-convolution was not distinctly better than other models; in fact it performed slightly worse on the testing set.

5.2 Future Work

There are two avenues for future research immediately available. The first is the study of better deep-learning architectures that implement graph convolution. Part of the reason for the failure of our graph model is poor understanding and inexperience. Graph deep learning is a young field, with no textbook on the subject. Further work on a more appropriate model for EEG data would yield fruitful insight into why this model does not work; it may guide future researchers with heuristics for building graphs.

The second is the use of a different dataset. The TUH-EEG dataset's greatest set of common sensors is fairly low (only 17). A consequence of this low number was that we refrained from using graph pooling layers. Instead we proposed using the European Epilepsy Database which contains potentially better-quality labelling and higher number of sensors common to all patient datasets.

There are several, more distant possibilities for research. In Section 4.4 we discussed the validity of the way we defined the graph representing the EEG sensors. The challenge of defining graphs based on humans is not unique; as we noted, human-action recognition has similar problems. But whether it was the optimal representation of the graph is in question and an ongoing research problem. Can we build a model that will learn the best representation for a graph representation of a data domain?

Finally, there is the question of whether we can learn on data with different graph-dimensionalities. As mentioned above and in Section 4.4, we had to use a restricted number of sensors from the TUH-EEG dataset. Is it possible to build a model that will learn irrespective of the number of active nodes? Answering this question may have significant implications for graph deep learning in other domains, such as learning the activity on computer networks and hence cybersecurity.

Bibliography

[1] Alexus Strong. *How Cambridge Analytica Used Machine Learning to Mine Facebook Data.* Online. Mar. 2018. URL: `https://news.codecademy.com/cambridge-analytica-machine-learning-facebook-data/`.

[2] Carole Cadwalladr and Emma Graham-Harrison. *Revealed: 50 million Facebook profiles harvested for Cambridge Analytica in major data breach.* Online. Mar. 2018. URL: `https://www.theguardian.com/news/2018/mar/17/cambridge-analytica-facebook-influence-us-election`.

[3] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning.* Springer-Verlag New York Inc., Feb. 9, 2009. 745 pp. ISBN: 0387848576. URL: `https://www.ebook.de/de/product/8023140/trevor_hastie_robert_tibshirani_jerome_friedman_the_elements_of_statistical_learning.html`.

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

[5] Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. "Nonlinear Component Analysis as a Kernel Eigenvalue Problem". In: *Neural Computation* 10.5 (1998), pp. 1299–1319. DOI: `10.1162/089976698300017467`.

[6] Iyad Obeid and Joseph Picone. "The Temple University Hospital EEG Data Corpus". In: *Frontiers in Neuroscience* 10 (2016), p. 196.

[7] David James, Xianghua Xie, and Parisa Eslambolchilar. "A discriminative approach to automatic seizure detection in multichannel EEG signals". In: *Signal Processing Conference (EUSIPCO), 2014 Proceedings of the 22nd European.* IEEE. 2014, pp. 2010–2014.

[8] Scott B. Wilson, Mark L. Scheuer, Ronald G. Emerson, et al. "Seizure detection: evaluation of the Reveal algorithm". In: *Clinical Neurophysiology* 115.10 (2004), pp. 2280–2291. DOI: `10.1016/j.clinph.2004.05.018`.

[9] Y. LeCun, L. Bottou, Y. Bengio, et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324. DOI: `10.1109/5.726791`.

[10] Geoffrey Hinton Yann LeCun Yoshua Bengio. “Deep learning”. In: *Nature* (2015).

[11] Y. LeCun, B. Boser, J. S. Denker, et al. “Backpropagation Applied to Handwritten Zip Code Recognition”. In: *Neural Computation* 1.4 (1989), pp. 541–551. DOI: 10.1162/neco.1989.1.4.541.

[12] Sepp Hochreiter and Jürgen Schmidhuber. “Long short-term memory”. In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[13] Marvin L Minsky and S Papert. *Perceptrons*. Expanded Edition. MIT Press, Cambridge, MA, 1988.

[14] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. “Multilayer feedforward networks are universal approximators”. In: *Neural Networks* 2.5 (1989), pp. 359–366. DOI: 10.1016/0893-6080(89)90020-8.

[15] Joan Bruna, Wojciech Zaremba, Arthur Szlam, et al. “Spectral Networks and Locally Connected Networks on Graphs”. In: *ArXiv* (Dec. 21, 2013). arXiv: http://arxiv.org/abs/1312.6203v3 [cs.LG].

[16] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. “ImageNet classification with deep convolutional neural networks”. In: *Communications of the ACM* 60.6 (2017), pp. 84–90. DOI: 10.1145/3065386.

[17] Matthew D. Zeiler and Rob Fergus. “Visualizing and Understanding Convolutional Networks”. In: *Computer Vision – ECCV 2014*. Springer International Publishing, 2014, pp. 818–833. DOI: 10.1007/978-3-319-10590-1_53.

[18] Christian Szegedy, Wei Liu, Yangqing Jia, et al. “Going Deeper With Convolutions”. In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015.

[19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, et al. “Deep Residual Learning for Image Recognition”. In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.

[20] Jie Hu, Li Shen, and Gang Sun. “Squeeze-and-Excitation Networks”. In: *arXiv* (Sept. 5, 2017). arXiv: http://arxiv.org/abs/1709.01507v2 [cs.CV].

[21] Gul Varol, Ivan Laptev, and Cordelia Schmid. “Long-Term Temporal Convolutions for Action Recognition”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 40.6 (2017), pp. 1510–1517. DOI: 10.1109/tpami.2017.2712608.

[22] Pichao Wang, Wanqing Li, Zhimin Gao, et al. "Scene Flow to Action Map: A New Representation for RGB-D Based Action Recognition with Convolutional Neural Networks". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2017. DOI: `10.1109/cvpr.2017.52`.

[23] Gunnar Johansson. "Visual perception of biological motion and a model for its analysis". In: *Perception & Psychophysics* 14.2 (1973), pp. 201–211. DOI: `10.3758/bf03212378`.

[24] Earnest Paul Ijjina and C Krishna Mohan. "Human Action Recognition Based on MOCAP Information Using Convolution Neural Networks". In: *2014 13th International Conference on Machine Learning and Applications (ICMLA)*. IEEE. 2014, pp. 159–164.

[25] Earnest Paul Ijjina and C Krishna Mohan. "Human action recognition based on motion capture information using fuzzy convolution neural networks". In: *2015 Eighth International Conference on Advances in Pattern Recognition (ICAPR)*. IEEE. 2015, pp. 1–6.

[26] Michael Edwards and Xianghua Xie. "Graph-Based CNN for Human Action Recognition from 3D Pose". In: *Workshop on Deep Learning in Irregular Domains at the 28th British Machine Vision Conference (BMVC 2017)*. 2017.

[27] Michael M. Bronstein, Joan Bruna, Yann LeCun, et al. "Geometric Deep Learning: Going beyond Euclidean data". In: *IEEE Signal Processing Magazine* 34.4 (2017), pp. 18–42. DOI: `10.1109/msp.2017.2693418`.

[28] Proclus. *A Commentary on the First Book of Euclid's Elements*. Ed. by Glenn R Morrow. Princeton University Press, 1992.

[29] Martin Gardner. *Fads and Fallacies in the Name of Science*. Vol. 394. Courier Corporation, 1957.

[30] Ronald Newbold Bracewell. *The Fourier Transform and Its Applications*. McGraw-Hill New York, 1986.

[31] Michael Edwards and Xianghua Xie. "Graph Based Convolutional Neural Network". In: *ArXiv* (Sept. 28, 2016). arXiv: `http://arxiv.org/abs/1609.08965v1 [cs.CV]`.

[32] Mikael Henaff, Joan Bruna, and Yann LeCun. "Deep Convolutional Networks on Graph-Structured Data". In: *ArXiv* (June 16, 2015). arXiv: `http://arxiv.org/abs/1506.05163v1 [cs.LG]`.

[33] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. "Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering". In: *Proceedings of the 30th International Conference on Neural Information Processing Systems*. NIPS'16. Barcelona, Spain: Curran Associates Inc., 2016, pp. 3844–3852. ISBN: 978-1-5108-3881-9. URL: `http://dl.acm.org/citation.cfm?id=3157382.3157527`.

[34] Ron Levie, Federico Monti, Xavier Bresson, et al. "CayleyNets: Graph Convolutional Neural Networks with Complex Rational Spectral Filters". In: *ArXiv* (May 22, 2017). arXiv: `http://arxiv.org/abs/1705.07664v1 [cs.LG]`.

[35] Federico Monti, Michael M. Bronstein, and Xavier Bresson. "Geometric Matrix Completion with Recurrent Multi-Graph Neural Networks". In: *ArXiv* (Apr. 22, 2017). arXiv: `http://arxiv.org/abs/1704.06803v1 [cs.LG]`.

[36] Bing Yu, Haoteng Yin, and Zhanxing Zhu. "Spatio-Temporal Graph Convolutional Networks: A Deep Learning Framework for Traffic Forecasting". In: *arXiv* (Sept. 14, 2017). DOI: `10.24963/ijcai.2018/505`. arXiv: `http://arxiv.org/abs/1709.04875v4 [cs.LG]`.

[37] David Duvenaud, Dougal Maclaurin, Jorge Aguilera-Iparraguirre, et al. "Convolutional Networks on Graphs for Learning Molecular Fingerprints". In: *Proceedings of the 28th International Conference on Neural Information Processing Systems -Volume 2*. NIPS'15. Montreal, Canada: MIT Press, 2015, pp. 2224–2232. URL: `http://dl.acm.org/citation.cfm?id=2969442.2969488`.

[38] Florian Dorfler and Francesco Bullo. "Kron Reduction of Graphs With Applications to Electrical Networks". In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 60.1 (2013), pp. 150–163. DOI: `10.1109/tcsi.2012.2215780`.

[39] Inderjit S. Dhillon, Yuqiang Guan, and Brian Kulis. "Weighted Graph Cuts without Eigenvectors: A Multilevel Approach". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 29.11 (2007), pp. 1944–1957. DOI: `10.1109/tpami.2007.1115`.

[40] Dorit Ron, Ilya Safro, and Achi Brandt. "Relaxation-Based Coarsening and Multiscale Graph Organization". In: *Multiscale Modeling & Simulation* 9.1 (2011), pp. 407–423. DOI: `10.1137/100791142`.

[41] Chun Wang, Shirui Pan, Guodong Long, et al. “MGAE: Marginalized Graph Autoencoder for Graph Clustering”. In: *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management - CIKM '17*. ACM Press, 2017. DOI: 10.1145/3132847.3132967.

[42] Or Litany, Alex Bronstein, Michael Bronstein, et al. “Deformable Shape Completion with Graph Convolutional Autoencoders”. In: *arXiv* (Dec. 1, 2017). arXiv: http://arxiv.org/abs/1712.00268v4 [cs.CV].

[43] Yiluan Guo, Hossein Nejati, and Ngai-Man Cheung. “Deep neural networks on graph signals for brain imaging analysis”. In: *arXiv* (May 13, 2017). arXiv: http://arxiv.org/abs/1705.04828v1 [cs.CV].

[44] Andrew P. Valentine and Jeannot Trampert. “Data space reduction, quality assessment and searching of seismograms: autoencoder networks for waveform data”. In: *Geophysical Journal International* 189.2 (2012), pp. 1183–1202. DOI: 10.1111/j.1365-246X.2012.05429.x.

[45] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing*. Second. Prentice Hall, 2002.

[46] D. I. Shuman, S. K. Narang, P. Frossard, et al. “The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains”. In: *IEEE Signal Processing Magazine* 30.3 (2013), pp. 83–98. DOI: 10.1109/msp.2012.2235192.

[47] Cédric Chevalier and Ilya Safro. “Comparison of Coarsening Schemes for Multilevel Graph Partitioning”. In: *Learning and Intelligent Optimization*. Ed. by Thomas Stützle. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 191–205.

[48] American Clinical Neurophysiology Society. *Technical Standard 1: Standard for Transferring Digital Neurophysiological Data Between Independent Computer Systems*. Feb. 2008. URL: https://www.acns.org/pdf/guidelines/Technical-Standard-1.pdf.

[49] Valer Jurcak, Daisuke Tsuzuki, and Ippeita Dan. “10/20, 10/10, and 10/5 systems revisited: Their validity as relative head-surface-based positioning systems”. In: *NeuroImage* 34.4 (2007), pp. 1600 –1611. ISSN: 1053-8119. DOI: https://doi.org/10.1016/j.neuroimage.2006.09.024. URL: http://www.sciencedirect.com/science/article/pii/S1053811906009724.

[50] S. Lopez, A. Gross, S. Yang, et al. “An analysis of two common reference points for EEGS”. In: *2016 IEEE Signal Processing in Medicine and Biology Symposium (SPMB)*. IEEE, 2016. DOI: 10.1109/spmb.2016.7846854.

[51] M Golmohammadi, V Shah, S Lopez, et al. "The TUH EEG Seizure Corpus". In: *Proceedings of the American Clinical Neurophysiology Society Annual Meeting.* 2017, p. 1.

[52] Ali Shoeb and John Guttag. "Application of Machine Learning to Epileptic Seizure Detection". In: *Proceedings of the 27th International Conference on International Conference on Machine Learning.* ICML'10. Haifa, Israel: Omnipress, 2010, pp. 975–982. ISBN: 978-1-60558-907-7. URL: `http://dl.acm.org/citation.cfm?id=3104322.3104446`.

[53] A. Harati, M. Golmohammadi, S. Lopez, et al. "Improved EEG Event Classification Using Differential Energy". In: *2015 IEEE Signal Processing in Medicine and Biology Symposium (SPMB).* IEEE, 2015. DOI: `10.1109/spmb.2015.7405421`.

[54] Liu Rui, Hossein Nejati, and Ngai-Man Cheung. "Dimensionality reduction of brain imaging data using graph signal processing". In: *2016 IEEE International Conference on Image Processing (ICIP).* IEEE, 2016. DOI: `10.1109/icip.2016.7532574`.

[55] Kevin Swersky Geoffrey Hinton Nitish Srivastava. *Neural Networks for Machine Learning: Overview of mini-batch gradient descent.* Lecture slides. 2014. URL: `http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf`.

[56] P. Charbonnier, L. Blanc-Feraud, G. Aubert, et al. "Two deterministic half-quadratic regularization algorithms for computed imaging". In: *Proceedings of 1st International Conference on Image Processing.* IEEE Comput. Soc. Press, 1994. DOI: `10.1109/icip.1994.413553`.