Deep Learning and Artificial Intelligence Mini-Project Report

Jack Lloyd

Department of Mathematical Sciences

Easter 2024

Lecturer: Dr. James Liley

1 Part I

1.1 A Summary of the Paper

In the first part of this report, we will analyse the paper 'Beyond the Edge of Stability via Two-step Gradient Updates' (Chen and Bruna, 2022)[1]. The overall focus of the paper is to analyse the behaviour of gradient descent, a well known optimisation method for finding local minima of differentiable multivariate functions. The specific focus is on the behaviour of the convergence of gradient descent. When solving an optimisation problem, we often look to minimise a loss function $f(\boldsymbol{x})$. Gradient descent seeks to find a parameter $\boldsymbol{x}^*$ which minimises $f(\boldsymbol{x})$ by iterating over

$$\boldsymbol{x}^{(t+1)} = \boldsymbol{x}^{(t)} - \eta \nabla_{\boldsymbol{x}} f(\boldsymbol{x}) \tag{1.1}$$

until convergence, where η is the learning rate or 'step size' hyperparameter. The learning rate controls the size of the steps when moving towards the minimum of the loss function. Choosing the correct learning rate is arguably the most difficult part of performing gradient descent. When adjusting the learning rate, the value of $\lambda(\boldsymbol{x}) := \lambda_{\max}[{\nabla_{\boldsymbol{x}}}^2 f(\boldsymbol{x})]$ (the maximum eigenvalue of the training loss Hessian) is measured since this indicates the curvature of the loss function. This value is commonly referred to as sharpness. It has been shown that $\sup_{\boldsymbol{x}}(\lambda(\boldsymbol{x})) < 2/\eta$ guarantees convergence of gradient descent. As the sharpness increases towards the threshold value $2/\eta$, it continues to rise steadily. However when it passes this threshold, gradient descent starts to become unstable and the iterations start to oscillate with increasing magnitude along the direction of greatest curvature (Cohen et al. 2020) [2]. It would be expected that the gradient descent then diverges, however this is not the case. It is shown by (Cohen et al. 2020) [2] that the sharpness hovers at or slightly above the threshold $2/\eta$ whilst the training loss consistently decreases over long periods of time. This region is known as the 'Edge of the Stability' and the analysis of this unstable convergence is the focus of Chen and Bruna's paper.

A particular area of attention is on the behaviour of fixed point two-step updates around local minima of the loss function. Say we denote the update of gradient descent on the function f as $F_\eta(x)$ with learning rate η. We say there exists a two period stable oscillation if there exists x such that $F_\eta(F_\eta(x))(= F_\eta^2(x)) = x$ for which x is not a minima of f. Such x is referred to as a fixed point of $F_\eta^2(x)$. Chen and Bruna (2022) [1] considers two step stable oscillations around the edge of stability in order to learn how gradient descent behaves when the sharpness passes the threshold $2/\eta$. This paper begins by developing conditions that one dimensional loss functions must obey to guarantee that there is a fixed point x_0 close to a minima which is a fixed point of $F_\eta^2(x)$, in which η is beyond the edge of stability. Chen and Bruna [1] then provide further ideas which allow this condition to still hold for more complex loss functions, further generalising the theory to provide similar results for matrix factorisation and two dimensional problems. The generalisation of the ideas into matrix factorisation and more complex settings provides exciting potential to be utilised in machine learning and neural network settings, in which a guarantee of convergence will vastly improve training. A future avenue of research is investigating the behaviour of gradient descent on loss functions of higher dimensions. If similar patterns occur with high-dimensional functions, this will vastly change time taken for complex models to train and learn patterns in big data. This means that more complex training algorithms can be deployed in the same time, allowing for more accurate predictions.

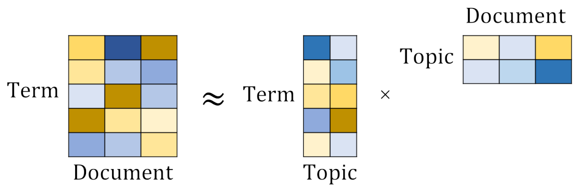


Figure 1.1: The Process of Matrix Decomposition for Topic Modelling Visualised. Source: Sou et al. (2016) [4]

1.2 A Potential Real World Application: Enhancing Natural Language Processors

Chen and Bruna's paper [1] explores the behaviour of gradient descent, and specifically the performance of gradient descent with a large enough learning rate to cross the edge of stability, as defined in Section 1.1. A potential real world application of the results proposed in this paper is enhancing natural language processors (NLPs). Section 6 discusses the existence of a 2 period orbit in matrix factorisation problems, which are prevalent in the area of topic modelling in NLPs. Topic modelling essentially is an algorithm to output the topic(s) that best describe a section of text. Wang and Zhiang (2023) [3] propose the matrix factorisation in the following way: say we have N documents with K topics and a vocabulary of V words, we aim to learn a topic-document matrix $\boldsymbol{W}$, which contains the probabilities of a certain topic being included in a certain topic, and a word-topic matrix $\boldsymbol{C}$, which contains the probabilities of a word being associated to a topic, by decomposing a document-word matrix $\boldsymbol{D}$ (see Figure 1.1). Note that in Observation 1, it assumes that all matrices are of the same dimensions, however in Theorem 6 this assumption is relaxed. We want to find $\boldsymbol{W}$ and $\boldsymbol{C}$ which minimises the following:

$$||\boldsymbol{D} - \boldsymbol{CW}||_F{}^2 \quad \text{subject to} \quad \boldsymbol{C}, \boldsymbol{W} \geq \boldsymbol{0}.$$

We can then use gradient descent to find an approximate solution to the problem above, if we treat the expression as a loss function we look to minimise. If we can guarantee that the gradient descent will enter a period-2 orbit around a minima, then this can improve computation costs and time of finding a minima. This avoids NLPs having to set smaller learning rates to ensure that the gradient does not explode. When we consider this problem in a high-dimensional setting, such as in recommendation systems or search engines attempting to provide results given certain demands of topics, if each matrix factorisation problem can guarantee a decomposition, then this will greatly improve computation speed and will minimise a cost function which will be proportional to the number of interations/runs of the gradient descent algorithm.

1.3 A Use Case of Theorem 1

In this section we will focus on Theorem 1. The general idea of the theorem is that if one can find a one dimensional differentiable function $f(x)$ which obeys certain conditions based on the value of the derivatives at a minima $\bar{x}$ of $f(x)$, then there is a range of points near the minima for which every point is a fixed point of $F_\eta^2(x)$, where $F_\eta(x)$ is the gradient descent update function:

$$\boldsymbol{x}^{(t+1)} = F_\eta(x^{(t)}) = x^{(t)} - \eta f'(x^{(t)}) \ ,$$

with $f'(x)$ being the first derivative of $f(x)$. Let us take $f(x) = (x^2 - \frac{1}{4})^2$ for example. It is clear this function has minima at $\bar{x} = \pm\frac{1}{2}$; we will focus on $\bar{x} = \frac{1}{2}$. We list the first four derivatives of the function which we require to check the relevant conditions.

$$f'(x) = 4x(x^2 - \frac{1}{4}), \quad f''(x) = 4(x^2 - \frac{1}{4}) + 8x^2, \quad f^{(3)}(x) = 24x, \quad f^{(4)}(x) = 24 \ .$$

We require $f^{(3)}(\bar{x}) \neq 0$ and $3[f^{(3)}(\bar{x})]^2 - f''(\bar{x})f^{(4)}(\bar{x}) > 0$. Note, in this case, $f^{(3)}(\frac{1}{2}) = 12$ and $3[f^{(3)}(\frac{1}{2})]^2 - f''(\frac{1}{2})f^{(4)}(\frac{1}{2}) = 384$. Hence both conditions are satisfied. Theorem 1.1 then claims there exists an ϵ with sufficiently small $|\epsilon|$ and $\epsilon f^{(3)}(\frac{1}{2}) > 0$ such that: for any $x_0 \in [\frac{1}{2} - \epsilon, \frac{1}{2}]$, there exists a learning rate η such that x_0 is a fixed point of $F_\eta^2(x)$, and

$$\frac{2}{f''(\bar{x})} < \eta < \frac{2}{f''(\bar{x}) - \epsilon f^{(3)}(\bar{x})} \ .$$

Let us take $\epsilon = \frac{1}{100}$, which obeys both conditions required, and $\eta = \frac{1251}{1250} = 1.0008$. This η lies in the required bound, as

$$\frac{2}{f''(\frac{1}{2})} = 1 \quad \text{and} \quad \frac{2}{f''(\frac{1}{2}) - \epsilon f^{(3)}(\frac{1}{2})} = \frac{2}{2 - 0.12} \approx 1.0638 \ ,$$

and $1 < 1.0008 < 1.0638$. We then have for all $x_0 \in [0.49, 0.5]$, that $F_\eta(F_\eta(x_0)) = x_0$. In other words, every x_0 is a fixed point of $F_\eta^2(x)$.

The importance of this theorem is that it gives readers an initial class of functions that when we perform gradient descent on the function to find a minima, we can guarantee that the gradient descent will oscillate around the edge of stability. This means that when performing gradient descent on functions of this form, users can guarantee that the gradient descent algorithm will not diverge. This guarantee means that users can choose larger learning rates, leading to faster convergence. This will improve both computation time and cost, which are key considerations in neural network training.

1.4 Discussion of Theory Proposed

In this section we will discuss Proposition 2, Observation 1 and Figure 3 of Chen and Bruna's [1] paper, highlighting their importance in the context of the paper.

1.4.1 Proposition 2

Proposition 2 is a natural progression from Proposition 1, which states given a squared loss function $f(x) = (g(x) - y)^2$, if $g(x)$ obeys certain conditions (which can be derived from the conditions stated in Theorem 1) then the loss function will have a fixed point of $F_\eta^2(x)$ around $x = \bar{x}$. Proposition 2 follows on from this stating if we have two functions $p(x), q(y)$ at points $x = \bar{x}$ and $y = p(\bar{x})$ both satisfying the conditions in Proposition 1, then we can say $q(p(x))$ also satisfies the conditions to have a fixed point of $F_\eta^2(x)$ at $\bar{x}$. This proposition is important as it allows users to easily compose more complex functions whilst guaranteeing they have a fixed point in gradient descent.

In the relevance of the paper, this advancement is interesting as it gives us a whole new group of functions that behave around the edge of stability. In neural networks, we compose activation functions as we calculate the output of neurons in each layer. In a deep neural network, the overall output of the neural network will be an extremely complex function created by many layers of different activation functions composed with each other. Direct analysis of the fixed point behaviour of the entire network would be extremely difficult and time consuming. Proposition 2 allows us to break the analysis down into the analysis of individual activation functions using Proposition 1, which is much easier. This means if we know the behaviour of all individual activation functions within a neural network, then we know the fixed point behaviour of the entire network. Proposition 2 is extremely useful as it allows users to quickly draw convergence criteria of large networks, which means training a network can be much more time and cost efficient.

1.4.2 Observation 1

Observation 1 considers the problem of matrix factorisation, where we aim to decompose a target matrix $\boldsymbol{C}$ into the product of two matrices $\boldsymbol{Y}$ and $\boldsymbol{Z}$. The solution to this problem can be rephrased into minimising the loss function $L(\boldsymbol{Y}, \boldsymbol{Z}) = \frac{1}{2}||\boldsymbol{Y}\boldsymbol{Z}^T - \boldsymbol{C}||^2{}_F$. Observation 1 states that, given a learning rate η, if the two leading eigenvalues of $\boldsymbol{C}$ satisfy certain conditions that there exists a possible initialisation of $\boldsymbol{Y}$ and $\boldsymbol{Z}$ which leads gradient descent to converge to a period 2 orbit. This is important as it shows that a period 2 orbit of gradient descent can still exist in a more complicated, higher dimensional context. This observation is extremely importance as it uses similar ideas to those used in Section 4 on 1D functions and provides a result for n-dimensional matrices. Matrix factorisation is a large area of focus in machine learning. If we can guarantee convergence to solutions of a matrix factorisation problem this can make algorithms which aim to make use of matrix factorisation cost and time efficient. For example, matrix factorisation can be used in the compression of deep neural networks (DNNs). In DNNs, dimension reduction of the weight matrices is required since the initial cost is often large due to the vast set of parameters (Cai et al. 2023) [5].

1.4.3 Figure 3

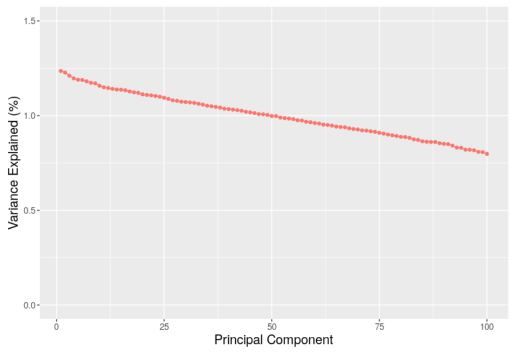
Figure 3 shows two plots of the trajectory of gradient descent as it attempts to converge to the minima of a 2D loss function $L(x) = \frac{1}{2}(1-xy)^2$. It focuses on work from Damian et al. (2023) [6] which proposes to track gradient descent's trajectory projected onto the manifold $\mathcal{M} = \{\theta : \lambda(\theta) < \frac{2}{\eta}, L(\theta) \cdot u(\theta) = 0\}$, where $\lambda(\theta)$ and $u(\theta)$ are the leading eigenvalue and eigenvector of the Hessian of the loss function respectively. Note that this manifold concerns values of θ which do not cross the edge of stability.

Preceding Figure 3 it is proposed that for $L(x) = \frac{1}{2}(1-xy)^2$, such a manifold only exists when $\eta < 1$. Figure 3a shows the trajectory of GD when $\eta = 1.08$, hence a manifold does not exist. Importantly, it shows that the trajectory gets stuck oscillating between two values, either side of the line $xy = 1$, which provides the set of solutions which minimise $L(\theta)$. When the learning rate is set to $\eta = 0.95 < 1$, such a manifold does exist and is of the form $\mathcal{M} = \{(x, y) : xy = 1, x + y < \sqrt{78/19}\}$ (Proposition 6). Note that the manifold is a subset of the set of solutions which minimise $L(\theta)$. In this case, Figure 3b shows that the trajectory converges to a value on the manifold, moving from a sharper region to a flatter region. This is important to readers since it shows if we set the learning rate to satisfy the conditions for a manifold to exist, then if we project the gradient descent trajectory onto the manifold, we can guarantee that the trajectory will converge to a minima whilst moving from a sharper region to a flatter region.

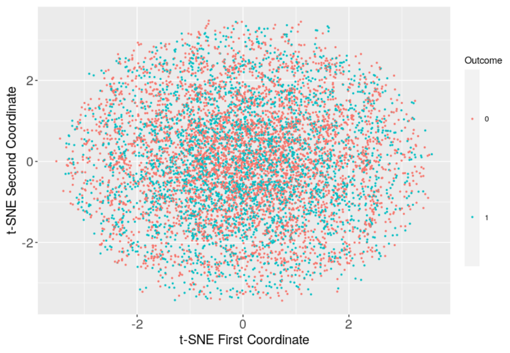
2 Part II

2.1 Data Analysis

Before applying machine learning methods to our data to create a decision rule, we attempted to perform dimensionality reduction, considering our data is of high dimensions. We first performed principal component analysis (PCA), which aims to find variables which explain a large proportion of the variance in the dataset. However, the PCA did not yield significant results since each of the 100 principal components explained an approximately equal amount of the total variance (see Figure 2.1a). We also performed t-distributed stochastic neighbour embedding (t-SNE), a non-linear dimensionality reduction technique which embeds high-dimensional data for visualisation in a low dimensional space. Figure 2.1b displays the results of performing t-SNE on our dataset. The lack of distinct clusters shows that we cannot identify patterns in the datasets which correspond to the same outcome.



(a) Scree Plot to Show the Proportion of Total Variance Explained By Each Principal Component



(b) t-SNE scatterplot displaying 2D embedded points

Figure 2.1: Plots to show performance of PCA (a) and t-SNE (b)

We began by splitting up `x_test` into two separate datasets: a training set and a test set. We used the 80-20 rule, where we randomly sampled (without replacement) 80% of the data and set this subset equal to our training data set. The rest of the data was left to become our test set. We did this so that we can train our different models on the test set and then gauge how our model predicts the output using the test set. For ease of working, we also separated the `ID` column, the predictor values and the output column. This allows us to fit models in `R`. We attempted multiple different methods using different machine learning techniques to perform binary classification before comparing them and deciding on the best decision rule.

2.2 Method 1: Logistic Regression

Given the 100 `X` values, we want to predict an output of 0 or 1. We began with a simple method: logistic regression. Given an input, logistic regression will produce a probability value of the output being (in this case) a 1. Say $p(X) = p(Y = 1|X)$. After fitting our logistic regression model to the test set, we then used the model to predict the `Y` values in the training set given the predictor values . We then compared our predicted values to our true values, which gave us an accuracy of 63.9%.

2.3 Method 2: Random Forests and Boosting

Classification trees are a common classification algorithm in machine learning. However, due to the high-dimensions of our dataframe, `R` struggled to fit a simple classification tree using the `tree`. Hence, we used random forests to help combat this issue. The idea behind random forests is that we take a bootstrap resample of the training data, build a classification tree but at each split randomly select m out of p total predictors as split candidates and select the optimal split within the selected predictors. We repeat this building n trees and then averaging over the trees. This allows us to combat the issue of dimensionality as we work with smaller subsets of predictors.

Fouodo et al. (2023) [7] highlight that the two hyperparameters which have the largest effect on the performance of the `randomForest` algorithm are `mtry.prop`, the proportion of variables we randomly select to be split candidates, and `sample.fraction`, the proportion of the data to bootstrap sample. We looped through possible values of `mtry.prop` and `sample.fraction` to see what effect they had on the error rate of our random forest, ultimately deciding on `mtry.prop` $= 0.1$ and `sample.fraction` ≈ 0.6. Using this random forest, we obtained a prediction accuracy of around 60%. We then attempted to fit a boosted tree to the data, which sequentially builds an ensemble of shallow trees with each tree learning and improving on the previous one, in the hope that sequential learning would allow for the model to learn complex relationships in the dataset. However, the performance was extremely similar to that of random forests.

2.4 Method 3: Neural Networks

The third method we used to solve the binary classfication problem was fitting a neural network using the `keras` package. Before creating the neural network, we used the functions to_categorical() to one-hot encode the `Y` vectors into binary classification matrices and `array_reshape()` to shape the input `X` correctly. To begin, we fit a neural network with 5 layers, each 100 neurons wide, all with ReLu activation functions, with a single sigmoid output neuron. When using a neural network to perform a binary classification problem, it is common practise for the activation function of the output layer to be sigmoid. We judged performance based on prediction accuracy of our neural network on the test data set.

We experimented with altering the width and depth of the neural network, aiming first to overfit the data. Once we acheived a network which had overfit the data, we slowly reduced the depth and width of the network and dropout rate within the hidden layers until we achieved an optimal performance. We observed the effect of altering the activation functions in the network, using the ReLu, SeLu (scaled exponential linear unit), softplus and tanh functions. Despite ReLu being the common choice in deep learning models we noticed other activation functions consistently performed better. Using a wide network seemed to consistently overfit to the data so we dropped the width of the layers to 10 neurons wide to observe if there was a change in performance. This choice seemed to balance overfitting and underfitting well. We observed the effect on the depth of the network on performance, and concluded that 8 hidden layers seemed to perform best. We attempted to use dropout to combat any overfitting that occurred in our model. We observed that a dropout rate of 30% throughout the hidden layers was the optimal balance between underfitting the data and overfitting the data. Throughout the process of tuning our model, the prediction accuracy would always be between 60-64%, so it was difficult to conclude that a certain combination of parameters was clearly better than another.

The architecture of our final network we decided on consisted of 8 hidden layers each of 10 neurons wide, with SeLu activation functions throughout the network. The SeLu function is as such:

$$\mathrm{SeLu}(x) = \begin{cases} \lambda x \qquad \text{if} \quad x > 0 \\ \lambda\alpha(e^x - 1) \qquad \text{if} \quad x \leq 0 \end{cases}$$

We chose SeLu functions as they seemed to consistently perform better than ReLu, they learn quicker than ReLu and do not suffer from the vanishing gradient problem. Throughout the network we varied the dropout rate between either 25% or 30%. Our one output neuron had a sigmoid activation function, returning the probability that the input variables correspond to an output of a 1. Once trained on the training dataset, our neural network achieved a prediction accuracy of 64.7% on the test dataset, which was higher than any alternate architecture we trained.

2.5 Discussion of Methods

In this section we will compare the methods used and propose our chosen decision rule. Whilst logistic regression is extremely efficient, it struggles to recognise non-linear relationships between variables and is known to struggle modelling high-dimensional data. Both neural networks and random forests have the ability to model non-linear relationships between variables in high-dimensional data. However it is considered that neural networks have a higher potential to achieve a higher prediction accuracy and model considerably more complex relationships in the data. There are more parameters to tune in a neural network, which proved difficult to find a combination that successfully achieved a significantly higher prediction accuracy than the other two methods. Ultimately we decided to use a neural network for our final decision rule, considering the data seemed to contain extremely complex relationships which we hoped would be captured by an appropriate architecture.

Bibliography

[1] Lei Chen and Joan Brune. “Beyond the Edge of Stability via Two-Step Gradient Updates”. In: (2022).

[2] J. Cohen et al. “Gradient Descent on Neural Networks Typically Occurs at the Edge of Stability”. In: *International Conference on Learning Representations* (2020).

[3] J. Wang and X. Zhang. “Deep NMF Topic Modelling”. In: *Neurocomputing 515 157-173* (2023).

[4] S. Suh et al. “L-EnsNMF: Boosted Local Topic Discovery via Ensemble of Nonnegative Matrix Factorisation”. GitHub. URL: `https://sanghosuh.github.io/lens_nmf-icdm/#/`.

[5] G. Cai et al. “Low-Rank Matrix Factorization for Deep Neural Network Compression”. In: *Applied Sciences* (2023).

[6] Nichani Damian and Lee. “Self Stabilization: The Implicit Bias of Gradient Descent at the Edge of Stability”. In: *arXiv* (2023).

[7] C.J.K Fouodo et al. “Effect of Hyperparameters on Variable Selection in Random Forests”. In: *arXiv* (2023).