Imperial College London Department of Earth Science and Engineering MSc in Applied Computational Science and Engineering

Independent Research Project

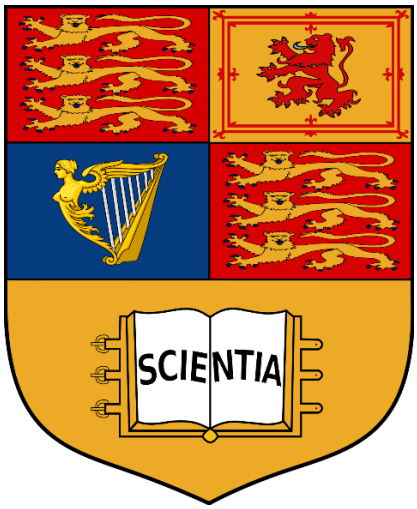
Final Report

Convolutional Loss Functions in Deep Learning

by
Ranran Tao

ranran.tao20@imperial.ac.uk
GitHub login: acse-rt120

Supervisor: Dr Lluis Guasch
Deborah Pelacani Cruz



August 2021

https://github.com/acse-2020/acse2020-acse9-finalreport-acse-rt120

ACKNOWLEDGEMENT

Firstly, I would like to thank my supervisors: Dr Lluis Guasch and Deborah Pelacani Cruz. Thanks to their patient and professional guidance in both video meetings and written emails. Also thanks to the ACSE teaching team for the organisation of this independent research project and the study group tutor and members who gave support and encouragement every week. Finally, thanks to my families and friends who support my study during this unusual pandemic.

ABSTRACT

The adaptive loss function with Wiener filter in deep learning is an approach to overcome cycle-skipping problems caused by Mean Squared Error loss function, which is often used in networks with autoencoder but ignores the consideration of the continuity of pixels, leading to the limitation of the recovery process of blurry images from autoencoders. The adaptive loss builds convolution-based instead of pixel-wise loss, thus achieving better results than traditional Mean Squared Error loss. However, the computation difficulty and time efficiency of the adaptive loss with Wiener filter still needs to be improved. This report provides optimisation strategies on the calculation process of the adaptive loss function. Findings on using torch.roll to optimize the speed of the calculation and findings on making 2D convolution instead of 1D to further improve the computing speed while enhancing the accuracy of the output image from the autoencoder are all presented with details in the report. Specifically, using torch.roll in generating Toeplitz matrix and keeping the input image in 2D without vectorising will give an advanced optimisation strategy for adaptive loss functions with Wiener filters in deep learning. Tests and results comparisons in this report also proved the success and advantages of the optimisation.

Key words: Optimisation, Deep Learning, Adaptive Loss Function, Wiener Filter, Autoencoder

1 Introduction

1.1 Background

A loss function plays an important role in deep learning. It provides an evaluation on how well the models predict the dataset by comparing predicted and actual data. It takes different forms, ranging from L2 norm to cross-entropy. The loss function will give a larger number if the predictions are not accurate enough, while it will also give a smaller number if the predictions are good. So, when optimisations are applied to improve the models, the loss function is a helpful evaluation to show if the optimisations are recommended or not.

In order to gain accurate prediction and a small number from loss function, the learning process is also important, where learnt features are at the heart of it. Sometimes, features which need to be learnt by the network are of large size, which improves the computational difficulties. So, an autoencoder is helpful since it provides an optimal way to initially compress the data and then reconstruct the data back from the encoded data to decoded data which are close to the original ones (Gogoi and Begum, 2017). An autoencoder saves as much information as possible while using as little data as possible, making it easier to complete different tasks.

A MSE (Mean Squared Error) loss function is usually used in the network with autoencoder. However, the MSE loss has some shortcomings, including the lack of consideration of the continuity of pixels, which limits the process of the recovery of blurry images from autoencoders (Xiong et al., 2020). In order to overcome these problems, the Wiener filter can be applied to keep the output and input data more similar, because it makes convolution-based instead of pixel-wise loss that could in theory incorporate more information from the encoding-decoding process for the optimisation.

This adaptive loss using Wiener filter is a method defining the loss as the multiplication of a penalty function T and a Wiener filter w, taking place of the original loss which uses a L2-norm pixel-wise difference. The filter from this method is used to match the observed trace and predicted trace from the autoencoder (Warner and Guasch, 2016).

1.2 Theory

According to Warner and Guasch (2016), in the 1D adaptive waveform inversion case, the convolution is about two 1D vectors a and b, where a is the input / predicted and b is the output / observed data, they are both vectorised. In order to make predicted and observed data more closed with each other during evolution, a function

$$g = \frac{1}{2}||Aw - b||^2 \tag{1}$$

should be minimized, where A is a Toeplitz matrix made from the input data a for 1D convolution and w is the Wiener filter that is calculated in the following form:

$$w = (A^T A)^{-1} A^T b \tag{2}$$

An objective function f is then needed to force filter w to a delta function at zero lag:

$$f = \frac{1}{2}\frac{||Tw||^2}{||w||^2} \tag{3}$$

Where T is a penalty function which weights the coefficients of filter w. This process gives a new form of full waveform inversion and can avoid cycle skipping caused by the conventional way, where the traditional objective function is usually defined as:

$$f = \frac{1}{2}||a-b||^2 \tag{4}$$

in which cycle skipping might occur if the initial model gives predicted data which differ the observed data by more than half a cycle, leading to erroneous updates to the model. Cycle skipping is usually a major challenge when performing full waveform inversion and the conventional least squared loss function as in equation (4), which works on the residuals of the waveforms, has a limitation in avoiding the cycle skipping and model updates (Yao et al., 2019). Thus, it is necessary to apply the above adaptive loss function with Wiener filter to overcome the limitation by changing the pixel-wise loss into the convolution-based loss.

The delta function that the filter is expected to finally become in the training process can also be a test for the correctness of the filter when the input and output are kept the same. A delta function can be a Dirac delta function if the domain is continuous, which is also called unit impulse symbol or it can be a Kronecker delta function if the domain is discrete (Bracewell, 1986). It is always in a shape that all the values are zeros over the real numbers except at zero, and the integral of this function over the entire range equals to one (Arfken and Weber, 2000). Therefore, if a filter is forced by the penalty function to become a delta function successfully, the predicted data will be trained to be more and more close to the observed data, which makes the autoencoder produce a more accurate output.

The Toeplitz matrix mentioned above is in a discrete convolution operation, which can be constructed in an operation in the following way:

$$y = h * x = \begin{bmatrix} h_1 & 0 & \cdots & 0 & 0 \\ h_2 & h_1 & \cdots & \vdots & \vdots \\ h_3 & h_2 & \cdots & 0 & 0 \\ \vdots & h_3 & \cdots & h_1 & 0 \\ h_{m-1} & \vdots & \ddots & h_2 & h_1 \\ h_m & h_{m-1} & \ddots & \vdots & h_2 \\ 0 & h_m & \ddots & h_{m-2} & \vdots \\ 0 & 0 & \cdots & h_{m-1} & h_{m-2} \\ \vdots & \vdots & \vdots & h_m & h_{m-1} \\ 0 & 0 & \vdots & \ldots & h_m \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} \tag{5}$$

Where h matrix is the Toeplitz matrix in a discrete convolution of h and x (Gray, 2006). Zeros should be appended to help produce a full convolution instead of a partial convolution. This way is often called zero padding, which is frequently used in spectrum analysis (Smith, 2007).

A Toeplitz matrix is often applied to model systems with shift invariant properties. The matrix structure itself has evident feature of shift invariance. Since Wiener filter is a technique that filters a signal with noise by using Linear Time Invariant (LTI) filtering (Reddy and Jayaraman, 2019), a Toeplitz matrix is a suitable choice when it comes to Wiener filer/LTI system modelling

(Reddi, 1984). Toeplitz matrices in 1D cases as above in (5) are well-known and classical in many applications, while Toeplitz matrices in 2D cases are also actively studied in recent years (Sakhnovich, 2017). The calculation of an equation involving a Toeplitz matrix as above usually uses Levinson recursion. Likewise, Levinson recursion is also applied to 2D cases in a similar way by regarding block Toeplitz matrices as matrix elements in a Toeplitz matrix (Musicus, 1988). A simple block matrix is in the form of the following:

$$A = \begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1N} \\ A_{21} & A_{22} & \cdots & A_{2N} \\ \vdots & \vdots & \vdots & \vdots \\ A_{M1} & A_{M2} & \vdots & A_{MN} \end{bmatrix} \tag{6}$$

Where A_{ij} are matrices. A is defined as a block Toeplitz matrix if the matrix structure of A, with respect to all the sub-matrices inside, is a Toeplitz. Moreover, A is a doubly blocked Toeplitz matrix if every submatrix A_{ij} is also in the form of a Toeplitz matrix (Araujo et al., 2020). After a doubly blocked Toeplitz matrix is generated in this way, the rest of the calculation of the 2D adaptive loss function with Wiener filter can be made in a much same way as it is in 1D case.

1.3 Motivation

This adaptive loss with Wiener filter has proved advantages compared to traditional MSE method (Warner and Guasch, 2016). However, it is not efficient enough when it comes to computation, particularly when calculating the Toeplitz matrix. There is still space for further optimisation in computation in 1D calculation and new experiment in 2D calculation. Therefore, this project is an optimisation of 1D and 2D Wiener filter calculation in deep networks. The description of methodology and code details will be delivered in Software Description and code metadata section, followed by the implementation and tests. The results and evaluation will be discussed in results and comparisons section. Finally, an overall conclusion of the project and future work will be in the last section.

2 Software Description

This section firstly summarises the implementation strategies and test methods in the methodology part and then describes the code details in the following code metadata part. The project was coded in Python and operated in google colab using gpu. Images from MNIST dataset were chosen as input data for research.

2.1 Methodology

Build an autoencoder for the encoding and decoding process of the training network. The main work should focus on changing the traditional MSE loss function into adaptive loss function and then optimise it. Firstly consider the calculation of the loss function in 1D, which requires both input data and output data being vectorised into 1D. Then build the Toeplitz matrix of the input data using the way stated in equation (5) and make the zero padding of the output data for calculation. After the Toeplitz matrix is built and the zero padding is finished, generate a Wiener filter using the Toeplitz matrix and zero padded output data as described in equation

(2), which makes convolution-based instead of pixel-wise loss that could in theory incorporate more information from the encoding-decoding process for the optimisation. A penalty function T is also needed to form an adaptive loss shown in equation (3). The penalty function is usually in a shape which equals one everywhere but zero when it is close to the middle point, because this form of penalty function can weight the coefficients of filter, which means penalizing the filter coefficients at large lags and penalty becomes larger when lag is larger, thus helping the filter to be a delta function by a multiplication operation as in equation (3). Once the filter becomes a delta function, it means that the input and output data are kept the same. Try different ways, using torch.roll and not using torch.roll, to create the Toeplitz matrix, compare their performance in time consuming and training loss convergence after training and then select the best way as an optimisation of 1D calculation. Then, extend the calculation into a more complex 2D version, exploring a new way to reduce the cost of computing and increase accuracy since 2D input data considers more spatial information. Keep the shape of input data without vectorising, which is 2D. Then implement in the similar way as above when it is 1D but calculate a larger Toeplitz matrix which is doubly blocked in the way described in the theory section above.

More specifically, when making the zero padding and Toeplitz matrix, the size of the Toeplitz matrix should be dependent on the length of the filter. In 1D case, if the filter is a vectorised data with length m, then the size of the corresponding Toeplitz matrix should be ($2m$-1, m). Meanwhile, the output data should also be zero padded, which should be a vectorised data of length $2m$-1. In 2D case, if the filter is a vector of length n^2, then the Toeplitz matrix for 2D convolution is of the size $((2n-1)^2, n^2)$. The vectorised output data should be of length $(2n-1)^2$. They should be zero padded and should be padded at the centre of zeros by different trials. Each submatrix is a small Toeplitz made from each row of the 2D input matrix of size n by n, which means the number of the rows of the input matrix is n, so there will be n different small Toeplitz matrices as elements in the large doubly blocked Toeplitz matrix.

The validation methods aim to test that the filter in loss function is correct. Two main ways can be made to do the validation. One way is plotting the images of the filter and check whether it is a delta function or not. The other way is more direct, which is convolving the filter with a random image and check whether a same image will be obtained.

2.2 Code Metadata

This project was built using python language on google colab platform with runtime type GPU. Libraries like torch, torchvision are used for doing tensor computations in deep learning with GPU acceleration. Library like sklearn helps to build machine learning models and read data. Library like matplotlib is for image visualization. Library like numpy is a tool for computing and array operations (Stančin and Jović, 2019). In the github repository (link on top of each page), codes for implementation and tests were uploaded in .ipynb files. Detailed user instructions are in README files.

https://github.com/acse-2020/acse2020-acse9-finalreport-acse-rt120

3 Implementation, Tests and Results

3.1 Implementation and Tests

This section shows the implementation and advantages of optimised adaptive loss function with Wiener filter in both 1D and 2D situation. One image from MNIST dataset was chosen as the input data for training. Two different methods of 1D calculation were implemented. An important difference between them is that the first method did not use torch.roll to build the Toeplitz matrix, the second method used torch.roll:

Table1: Different 1D Methods

First Method	Second Method
`for i in range(h):` `  A[i:i+h, i] = x[:]`	`for j in range(lw):` `  A_T[j] = torch.roll(x, j)`

Where h and lw are the length of the filter and A is the Toeplitz matrix made from the input data, A_T is the transpose of the Toeplitz matrix because torch.roll rolls tensors horizontally while the Toeplitz matrix rolls matrix elements vertically. x in the first method is the original vectorised input data while x in the second method is the vectorised input data with zero padding. The rest of the codes are in the github repository. Validation tests for filters were then made for checking the correctness. As stated in methodology, if the filter becomes a delta function after the training process, the predicted data from the autoencoder will be trained to be more and more close to the observed data, which means that the implementation will be as expected. Thus, a simple way to check the filter without the need for training process is making the input data the same as the output data, then plot the filter to test if it is a delta function. Also, if the output image is still the same with the input image after the convolution operation of the Toeplitz matrix and the filer, then it means the filter is correct. The following figure1 is the printed test result using the same input and output data. The filter can be proved correct because it is a delta function on the right side. Also, on the left side, it can be seen that a same image was obtained after convolving the filter with a random image:

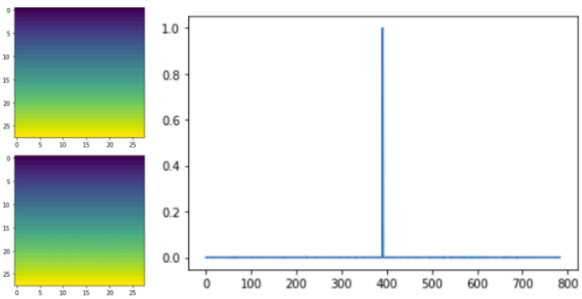


Figure1: Filter test with same input and output

During implementation, different penalty functions T were tried to multiply the filter to generate the adaptive loss function. A smooth gaussian function was firstly implemented and then a sharper function with zero in the middle but ones at everywhere else was also made. Both shapes aim to penalize the filter at lager lags by the multiplication operation mentioned in the

theory section. The following chart presents these different penalty function trials with the smooth gaussian function on the left and the sharp function on the right:

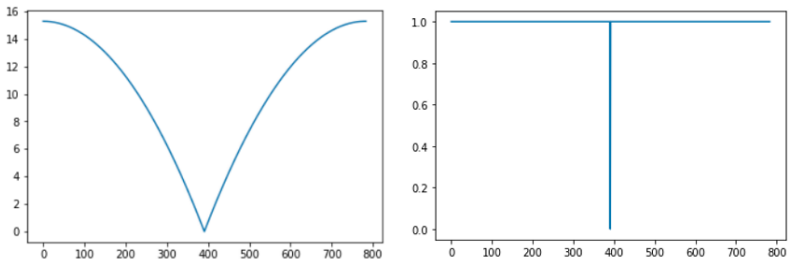


Figure2: Different penalty functions

A way of choosing which penalty function to reach a better result for this adaptive loss is to multiply the filter with both of the penalty functions respectively and then train the network and observe the plotted filter and the printed output image. If the filter is forced to be more like a delta function after the training with the same number of epochs and if the training loss value is smaller than the other one, then the corresponding penalty function will be considered more suitable for the adaptive loss.

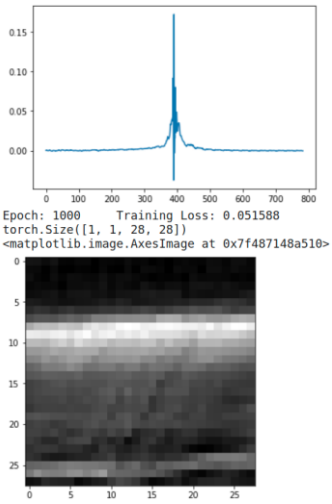


Figure3: Training – smooth gaussian as penalty function

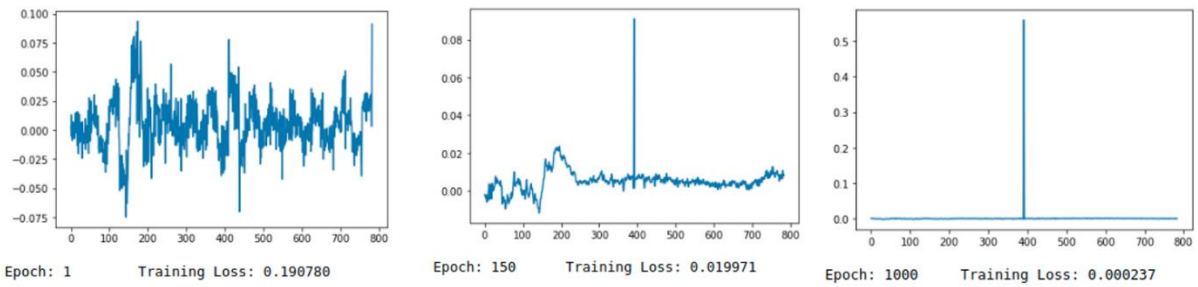


Figure4: Training – sharp T function as penalty function

Above figure3 is the training result when a smooth gaussian penalty function in figure2 was

implemented, figure4 is the training result when a sharp penalty function was implemented. It is obvious that a smooth gaussian function did not perform well enough as the filter is not an exact delta function after 1000 epochs training and the output image is blurry. On the contrary, the sharper penalty function T produces satisfying test results. Figure4 recorded three different conditions as the number of epochs increases, which are the changing shapes of the filter evolution when epochs are 1, 150 and 1000 respectively. It can be observed clearly the changing process of the filter from a random shape into a delta function during network training. Also, the training loss values of two situations vary differently, the loss value of the loss function using a sharp T function is much smaller, only 0.000237 compared to 0.051588. Thus, a sharp penalty function in figure1 was proved more suitable and was chosen for implementation.

An extension of the research to 2D version was then made for further optimisation, which means keeping the dimension of the input data without vectorising. 2D version is expected to obtain more information during autoencoder process because a 2D input matrix keeps the original column data without squeezing it into 1D and thus 2D is able to include more spatial information than vectorised 1D input data, which helps to produce more accurate results compared to 1D methods. It should also have a potential to improve the speed of calculation. A new larger doubly blocked Toeplitz matrix was built in the way stated in methodology. The form of the Toeplitz matrix can be seen more clearly when the input data has a smaller size than a MNIST image. Thus, for the purpose of checking the correctness of the new Toeplitz matrix, set the size of the input data as a 3 by 3 matrix: [[1, 2, 3], [4, 5, 6], [7, 8, 9]], the corresponding Toeplitz matrix formed by the codes as the following:

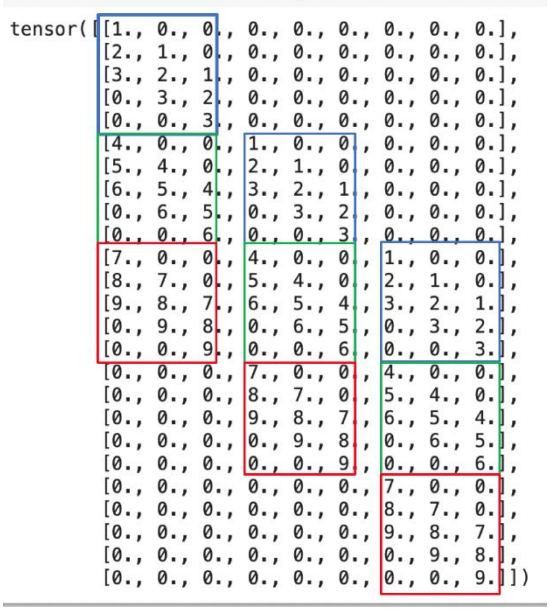


Figure5: Toeplitz Matrix formed by 2D input data

The above Toeplitz matrix was an output from the code uploaded in the github repository. It matches the description of the Toeplitz matrix made by a 2D matrix from methodology and theory section. Sub-matrices with blue, green and red frames are small Toeplitz matrices formed by three different rows of the 2D input data respectively. The whole large matrix is also a Toeplitz matrix by considering its matrix elements as 3 by 3 sub-matrices. Thus, the codes

https://github.com/acse-2020/acse2020-acse9-finalreport-acse-rt120

for Toeplitz matrix in 2D version are correct and can be safely used for a MNIST image which has a larger size. A similar delta function test in the way 1D did was also made for 2D implementation and the filter proved correct.

The output images of all the three methods after the training with 1000 epochs are shown below in figure6. Each column represents a different method. The top image of each column is the output from the autoencoder and the bottom image of each column is the original input before being put into the autoencoder.

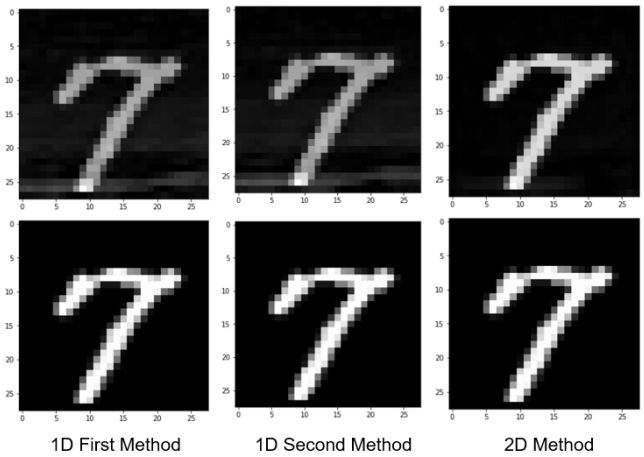


Figure6: Output Images

3.2 Results and Comparisons

Comparisons of the performances of different 1D and 2D methods were made after the completion of all the implementation. Analysis is shown in this section from two main aspects: time efficiency and convergence rate.

As for the time efficiency, the exact running time costed by each method was recorded in the following table2. The number of epochs was set as the same number, which was 1000. The training data was one image from MNIST dataset. From records in table2, 1D second method using torch.roll costs the least amount of time, which is only 28 seconds. 2D method costs the similar amount of time but a little slower. The most time consuming way is 1D first method without using torch.roll, which costs 5 seconds more than the most efficient 1D second method. Therefore, 1D second method and 2D method have a similar advantage in time efficiency compared to 1D first method.

Table2: Time Costs Comparison (1000 epochs)

1D First Method	1D Second Method	2D Method
33 seconds	28 seconds	29 seconds

As for the convergence rate of the training loss values, the detailed training loss values with increasing number of epochs from 100 to 1000 were recorded for each method. The following chart shows the trends and features of the convergence of three methods.

https://github.com/acse-2020/acse2020-acse9-finalreport-acse-rt120

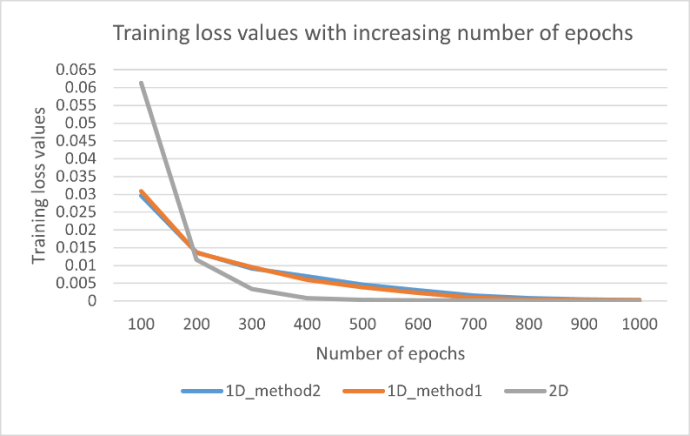


Figure7: Training Loss Values with increasing epochs

An obvious feature from above is that the 2D method performs the best convergence rate compared to the other two 1D methods and reaches a much smaller final loss value, while two 1D methods have similar slower convergence rates of loss values and reach similar larger final loss values when epochs are 1000. The final training loss is 0.000254 for 1D first method, 0.000215 for 1D second method when it is 1000 epochs. However, the final training loss for 2D method is 0.000070, which is much smaller than the other two. An more direct way to observe this training loss difference is from above figure6. The three output images in figure5 are images from three different methods after 1000 epochs training, it is obvious that the output image from 2D method is more close to the original input image compared to images from other methods. It can also be found in figure6 that 2D method can produce a similar loss value with two 1D methods when the number of epochs is only 500: the 2D training loss at this point is 0.000277.

Since 2D method can reach a similar effect with only half a number of epochs, a more detailed record can be made for further observation of the performances in both time consuming and training loss values in a same table. From the following table3, it can be seen that 2D method only spends 15 seconds (about half time amount which two 1D methods used) to give a similar training loss value, which means an image with similar accuracy.

Table3: Further Comparison

Methods	Time (seconds)	Training loss
Number of Epochs = 1000		
1D First Method	33	0.000254
1D Second Method	28	0.000215
2D Method	29	0.000070
Number of Epochs = 500		
1D First Method	16	0.004604
1D Second Method	14	0.003898
2D Method	15	0.000277

Therefore, from both time efficiency and convergence rate analysis, it shows that 1D second method optimised 1D first method in time efficiency but not convergence rate. So, using torch.roll when building the Toeplitz matrix is an initial optimisation to reduce computation difficulty. In 2D method, torch.roll was used to build the Toeplitz matrix as a way of optimisation in computing, making 2D consume time as little as the optimised 1D method. Furthermore, 2D method also makes a significant further improvement in convergence rate of training loss, which significantly optimises image accuracy too. Thus, 2D method gives the best optimisation after different comparisons and evaluations. Specifically, it means that using torch.roll in generating Toeplitz matrix and keeping the input image in 2D without vectorising will give an advanced optimisation strategy for adaptive loss functions with Wiener filters in deep learning.

4 Conclusion and Future Work

This paper provides optimisation strategies for the adaptive loss function with Wiener filter in deep learning. Based on original 1D Wiener filter calculation, it firstly proposed an optimisation on computing, using torch.roll when calculating the Toeplitz matrix for the convolution, which reduced the computation difficulty and cut the time consuming of the original 1D calculation. A further optimisation was then made to reduce more computation difficulty while increasing the accuracy of the output from the autoencoder. It is a method involving 2D calculation with torch.roll, which proved faster and having improved accuracy after the comparison with the optimised 1D method. All the implementations were tested and proved valid and correct, thus this research is able to demonstrate advanced benefits and the optimisation can be safely applied to deep networks with autoencoders when it comes to adaptive loss function using Wiener filter. The difficult part during the research was understanding the operations between the filter, input and output data and then deciding the way of zero padding and the size of the Toeplitz matrix for a fully convolved operation. The strength of this research is that it provides a proved optimisation strategy, a 2D Wiener filter calculation using torch.roll in deep network with autoencoder, which successfully improved the time efficiency and accuracy of the original Wiener filter calculation in the loss function. However, limitations still exist. The research is mainly focused on one image from one dataset. Future improvements can be made by increasing the batch size of each training, applying this optimisation strategy to a different dataset, try a deeper network and study on it, making the research more comprehensive and exploring more possible optimisations under different backgrounds and conditions.

https://github.com/acse-2020/acse2020-acse9-finalreport-acse-rt120

References

Araujo, A., Negrevergne, B., Chevaleyre, Y and Atif, J. (2020). On Lipschitz Regularization of Convolutional Layers using Toeplitz Matrix Theory. PSL, Universite Paris-Dauphine, ´ CNRS, LAMSADE, Paris, France. arXiv preprint arXiv :2006.08391

Arfken, G. B. and Weber, H. J. (2000). Mathematical Methods for Physicists (5th ed.), Boston, Massachusetts: Academic Press, ISBN 978-0-12-059825-0.

Bracewell, R. N. (1986). The Fourier Transform and Its Applications (2nd ed.), McGraw-Hill Book Company, New York.

Gogoi, M. and Begum, S. (2017). Image Classification Using Deep Autoencoders. 2017 IEEE International Conference on Computational Intelligence and Computing Research (ICCIC), (2017), pp. 1-5.

Gray, R. (2006). Toeplitz and Circulant Matrices: A review. Foundations and Trends® in Communications and Information Theory: Vol. 2: No. 3, pp 155-239, http://dx.doi.org/10.1561/0100000006

Musicus, B. R. (1988). Levinson and Fast Choleski Algorithms for Toeplitz and Almost Toeplitz Matrices. RLE TR No. 538, Massachusetts Institute of Technology, Cambridge, MA 02139 USA.

Reddi, S. S. (1984). Eigen Vector properties of Toeplitz matrices and their application to spectral analysis of time series, Signal Processing, Vol 7, North-Holland, pp.46-56, https://doi.org/10.1016/0165-1684(84)90023-9

Reddy, B. S. T. and Jayaraman, V. (2019). Application of Wiener Filter Making Signals Orthogonal. 2019 International Conference on Vision Towards Emerging Trends in Communication and Networking (ViTECoN), pp. 1-6, doi:10.1109/ViTECoN.2019.8899689.

Sakhnovich, A. (2017). Inversion of the Toeplitz-block Toeplitz matrices and the structure of the corresponding inverse matrices. Transactions of the American Mathematical Society 372 (2019), pp. 5547—5570, https://doi.org/10.1090/tran/7770

Smith, J. O. (2007). Mathematics of the Discrete Fourier Transform (DFT), with Audio Applications, Second Edition, e-book, assessed 17 August 2021, https://student.unsw.edu.au/citing-different-sources

Stančin, I. and Jović, A. (2019). An overview and comparison of free Python libraries for data mining and big data analysis. 2019 42nd International Convention on Information and

https://github.com/acse-2020/acse2020-acse9-finalreport-acse-rt120

Communication Technology, Electronics and Microelectronics (MIPRO), 2019, pp. 977-982, doi: 10.23919/MIPRO.2019.8757088.

Warner, M. and Guasch, L. (2016). Adaptive waveform inversion: Theory. Geophysics 81.6 (2016): pp.429-R445. DOI:10.1190/geo2015-0387.1

Xiong, Z., Lin, M., Lin, Z., Sun, T., Yang, G. and Wang, Z. (2020). Single image super-resolution via Image Quality Assessment-Guided Deep Learning Network. PLoS ONE 15(10): e0241313. https://doi.org/10.1371/journal.pone.0241313

Yao, G., Silva, N., Warner, M.,Wu, D. and Yang, C. (2019). Tackling cycle-skipping in full-waveform inversion with intermediate data. Geophysics 84(3):1-62 DOI: 10.1190/geo2018-0096.1