Evaluation of Genetic Algorithms and Evolutionary Strategy for Training DNN Models

Afnan Gurung
700038669

Abstract

In Deep Learning, rapid advancements have led to different training methods for training deep neural networks (DNNs). Therefore, this project aims to evaluate the performance of two similar methods regarding training DNNs: Genetic Algorithms and Evolutionary Strategies. I have chosen these two methods in specific as both are optimisation methods that derive ideas from nature to fine-tune computing performance.

This project's main objective is to conduct a comparative assessment between the two methods of training a DNN model for a common task. By utilising the benefits of each algorithm, I aim to deduce which method provides better performance in completing the common task. The results gained from this research project can be used in future studies to provide valuable insight to anyone considering using these optimisation methods in training their DNN model.

I certify that all material in this dissertation which is not my own work has been identified.

Table of Contents

1. Introduction

In recent years, the rapid developments in the field of machine learning have led to massive surges of innovation, specifically in terms of deep learning with the use of Deep Neural Networks (DNN) [1]. These networks consist of a complex neural architecture [2] in order to contribute to a multitude of different fields such as image recognition [3] and natural language processing [4].

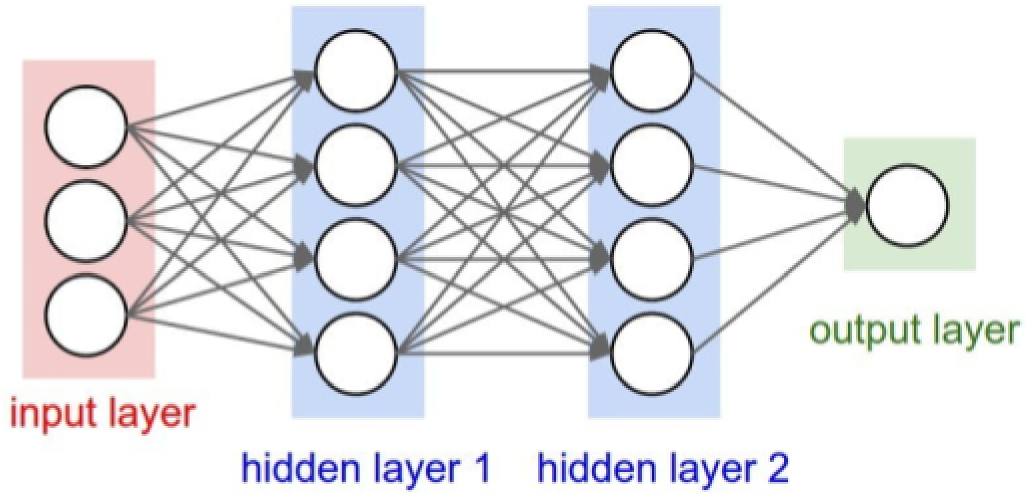

Figure 1: A Deep Neural Network Model [38]

As shown in the figure above, DNN models consist of an input layer, an output layer and multiple hidden layers between them. The input layer as the name suggests takes input data and stores it in the nodes of the layer [27]. Each node is then calculated into a sum using a weight along with a bias added to the final sum. The resulting sum is then fed to an activation function [28] in the hidden layer which then determines which neuron to activate. This is how deep learning models are able to compute complex non-linear relationships. Based on the calculations and computation of function in the preceding layers, an output is achieved via prediction in the output layer [29]. Based on how accurate the predicted output was to the actual output, values such as the weights and bias are then modified in the next iteration to try and achieve an output that is even closer to the actual desired output [30].

DNN is still a very new field that is being constantly researched in order to find new techniques to improve its efficiency. However, these optimisation techniques often require tweaking a large array of hyperparameters which is very computationally expensive and laborious [5].

One of these optimisation techniques that is currently being used is the use of nature-inspired computational methods [6] which utilise the principles of nature such as natural selection or evolution. Two examples are Genetic Algorithms (GAs) [7] and Evolutionary Strategies (ES) [8]. These techniques use natural principles such as genetic operations of crossovers, mutation or selection. The use of these makes these techniques have very great adaptability towards optimisation problems, making them very suitable for optimising the vast hyperparameter space [9] of DNNs.

Due to the recency of development in the field of DNNs as a whole can be considered scarce compared to other pre-existing fields. Therefore, this proposed project will aim to change that by conducting a thorough evaluation of Genetic Algorithms against Evolutionary Strategies. Each optimisation method's effectiveness in training DNN models will be tested whilst also evaluating the benefits and disadvantages of each.

The results from this project can be used to further our understanding of these optimisation techniques and also figure out potential practical applications for these methods. The data acquired can also be further utilised to provide valuable insight for future researchers who are looking into optimising DNN models.

In the following sections, the background along with related work regarding DNNs will be explored. Additionally, a project specification will be created, detailing all the requirements of the project: functional and non-functional. By the conclusion of the proposed project, I aim to provide an insightful analysis of the use of Genetic Algorithms and Evolutionary Strategies in the training of deep neural networks.

2. Literature Review and Objectives

2.1. Summary of literature review

2.1.1. Background

Deep Neural Networks have been a key aspect in terms of making technological advancements in the field of machine learning and artificial intelligence as a whole [10]. Thus, improving the efficiency [11] in training these DNN models has been a crucial point of interest for many researchers. In this section, pre-existing research regarding the optimisation of these networks will be discussed in detail. In particular, the use of the optimisation methods, Genetic Algorithms and Evolutionary Strategies. Additionally, deep learning as a whole has seen a surge in research and demand [12] compared to other fields in artificial intelligence.

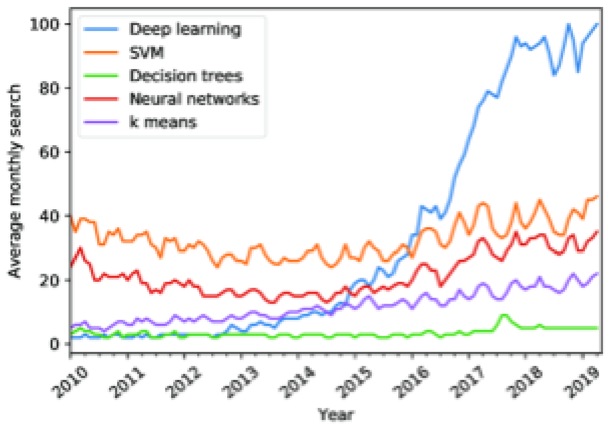

Figure 2: Google search trend showing increased attention in deep learning [39]

Figure 2 shows that over the recent years, deep learning has been getting a huge surge in terms of Google searches on the internet. This is due to the benefits deep learning has over traditional machine learning. One main benefit is that DNN models have great versatility in comparison to alternatives which makes them better for a wide range of applications with varying complexities [32]. Another reason for this surge is that there have also been major advancements in terms of computer hardware. This resolves one of the main drawbacks of deep learning which is the requirements of computational power as a DNN model has to process a lot of data. With the recent advancements, specifically in GPUs, training DNN models take a lot less time than before due to the increased number of cores in modern GPUs [31].

2.1.2. Challenges Of Optimising DNNs

Training Deep Neural Networks is a very difficult and time-consuming process [13]. This is due to any improvements in the performance of the model often requiring fine-tuning of hyperparameters such as learning rates, batch sizes and many more [14]. This is a very computationally expensive and laborious process especially when dealing with a large-scale operation [15].

2.1.3. Gradient-based Optimisation Algorithms

Gradient-based algorithms are the most common type of optimisation algorithms [16] that are used in DNN training due to their computational efficiency. Weights are initialised and used to compute a gradient against an objective function. The gradient computed is used to decide how the weights should be adjusted in the next iteration to minimise the cost function. This is repeated until the termination criterion is met. However, this process becomes very slow as the datasets increase in size due to the increasing amounts of computational power and memory needed. To solve this problem, a few solutions were made in the form of alternative versions of gradient descent algorithms such as stochastic gradient descent (SGD) and its many variants [17, 18].

Stochastic Gradient Descent is an iterative algorithm used for optimisation problems. The implementation intends to improve upon gradient descent algorithms in terms of computing speed and memory usage [33]. The algorithm works by dividing the data set into smaller subsets known as batches. A batch is randomly selected to compute the gradient of the cost function. The parameters for the model are then updated in a way that would reduce the cost function. These steps are repeated in each iteration until a termination criterion is met [34].

Unlike gradient descent algorithms, SGDs utilise randomness in the form of batches to add noise to the computation. This helps solve one of the main problems of the standard gradient descent where the algorithm can often get stuck into local minima [35]. The use of batches each iteration also results in only subsets of the dataset being used at a given time which results in better computational efficiency compared to gradient descent, specifically as the size of datasets increases [36].

However, these algorithms, as the name suggests need a gradient which might not be possible in every problem which is where alternatives such as evolution algorithms [19] are looked at.

2.1.4. Genetic Algorithms

Genetic Algorithm (GA) is an optimisation technique that uses nature-inspired computing in the form of natural selection and genetics. The algorithm represents data in the form of individuals in a population [20]. The population is then modified in the form of genetic operations such as crossover, mutation and selection to produce individuals with a better fit for the solution of a given problem. The algorithm consists of employing the natural principle of natural selection with the main idea being 'the survival of the fittest' with the fittest in this context meaning individuals with a better fit to the solution. GAs have been adapted to be used in the optimisation of hyperparameters [21] along with being able to explore complex hyperparameter spaces that gradient-based alternatives would usually not be able to. A common application is the use of genetic algorithms in Efficient Neural Architecture Search (ENAS) Algorithms [22] to find optimal network architectures. A disadvantage of this algorithm is the use of many hyperparameters such as crossover, mutation and selection [23]. As discussed earlier, the optimisation of these is a very computationally expensive process.

2.1.5. Evolutionary Strategy

Evolutionary Strategy (ES) is another type of evolutionary algorithm that utilises nature-inspired computing [24]. Whilst genetic algorithms focus on evolving a population, the evolutionary strategy focuses on optimising the distribution of hyperparameters. Unlike genetic algorithms, data is represented usually in the form of vectors making the algorithm more suited for larger-scale operations [25] that require continuous search spaces. This allows evolution strategies to have much better scalability than genetic algorithms, making them better for larger-scale projects. However, much like GAs, evolutionary strategies also rely on the tuning of hyperparameters which is a computationally expensive process and can greatly affect the performance of a DNN model.

2.2. Objectives

2.2.1. Comparative Studies

Genetic Algorithms and Evolutionary Strategies are both examples of evolutionary algorithms. Both algorithms utilise natural principles in their implementations albeit with some differences. Whilst both utilise the idea of initialising a population, ES uses real-value vectors whilst GAs use binary strings or a set of parameters. This results in ES being more naturally suited for continuous problems whereas, with GAs, a different encoding scheme might be required altogether.

Similarly, the design of GAs means that it is naturally going to require more iterations on average than an ES model due to the use of multiple generations in acquiring a better-fit solution. This results in better convergence speeds for ES models in comparison to GAs on average.

As discussed previously, both algorithms have their strengths and weaknesses, with one outperforming the other being dependent on the problem at hand [26].

In conclusion, the pre-existing literature shows that both algorithms have shown the potential to be utilised as an optimisation method in the training of DNNs whilst also being an alternative to gradient-based optimisation algorithms. However, there are very few comparative studies conducted that directly compare the two algorithms as shown in the literature review above. Furthermore, the studies also focused on using one specific scenario to obtain results which in some cases could heavily favour one algorithm over the other.

This proposed project intends to change that by conducting experiments to evaluate their performance whilst factoring in the strengths and weaknesses of each algorithm. The result gained from this project will further bolster the understanding of the two algorithms whilst providing more valuable insight into the topic of deep learning as a whole.

3. Project Specification

3.1. Hypothesis

The proposed experiment will test Genetic Algorithms (GA) and Evolutionary Strategies (ES) in the training of DNNs. the results should indicate that these algorithms should result in improved performance in terms of efficiency and convergence speed in comparison to traditional methods. Evolutionary strategy algorithms should also outperform genetic algorithms as the experiment increases in scale. However, both methods should show competitive performance compared to gradient-based alternatives but still have the increased robustness of evolutionary algorithms.

3.2. Functional Requirements

Implementation

To implement both algorithms, I will be using Python for both algorithms to reduce the number of variables in this experiment. Specifically, I will be using the popular framework called PyTorch. It is a popular machine-learning framework that uses the Torch library. It was developed very recently by Meta AI and is preferred by developers for its ease of use and intuitive API. I will also be using additional libraries to help with this process such as PyGAD and DEAP which are both Python libraries that help with the implementation of evolutionary algorithms.

Initialisation

Both algorithms will use randomness to ensure that the initial population has true diversity each time the population is initialised. True diversity will prevent the algorithms from converging to solutions that are suboptimal and will also reduce bias that can occur with different methods of initialisation.

Fitness function
The function will need to be fully customisable to allow a user to change the configuration. This is due to differing levels of value in performance metrics for different tasks i.e. in Image Classification, accuracy is highly prioritised. Other researchers will also be able to contribute further research into this project by modifying the fitness function and recording what effect it had in terms of the performance and accuracy of the model.

Visualisation
Data acquired from the experiment will be represented in a user-friendly way rather than left as raw data. To achieve this, I will be using matplotlib to visually represent the test data in the form of graphs. This will allow data acquired from the experiments to be represented clearly in a visual format which will make it easier to under for people who are less knowledgeable in this field, improving the accessibility of the findings from this project.

3.3. Non-functional Requirements

Performance
The proposed system should be able to produce results within a reasonable time frame. This will require the system to be efficient when dealing with larger data sets.

Scalability
The proposed system should be able to scale with larger DNNs and datasets without having any noticeable issues.

Robustness
The results acquired from this experiment should be able to be reproduced in another system. This ensures that the data acquired can be considered reliable for use in scientific research.

Compatibility
The entire setup should run flawlessly across multiple popular operating systems such as Windows, MacOS or Linux.

Error handling
In case of errors, the proposed system should be able to output errors to the terminal. This will make it easier for users to troubleshoot the problems even if they aren't as familiar with the language or the framework as the lines causing errors will be output straight to the terminal.

Documentation
A readme file should be included with the project to give concise and clear instructions to anybody looking to replicate results obtained from this experiment

Usability
The proposed system needs to be set up in a way so that it is easy for a user to easily set up and run their experiments with their chosen configuration. There will be clear concise

instructions included in the readme file to help the user set up the model in a straightforward manner even if they have never worked with the framework before.

3.4. Evaluation Criteria

To evaluate and compare the two algorithms, I will be testing the algorithms in the form of time series forecasting tasks. Specifically, stock market prediction. This will allow for clear evaluation metrics such as average model accuracy in terms of how accurate the model's predicted price is to the actual stock price as well as rating the performance based on convergence speed. Additionally, individual parameters will also be looked at to identify their effects on the performance of a model. For instance, different population sizes will be used to see what effect they have on the overall performance of each model.

For this project, I will be utilising historical stock data in this experiment to allow for easy comparisons for model accuracy. I will be acquiring this data using datasets from Yahoo Finance. This is because it will allow me to use historical stock data of a stock as a dataset for training the algorithms. I can then evaluate the performance of each model based on the accuracy of the predicted stock price to the actual stock price as well as compare the convergence speeds of both models.

3.5. Project Risks and Challenges

3.5.1. Risks

Hardware failure

Both algorithms are computationally intensive in terms of training DNNs. In worst-case scenarios, this could lead to potential hardware failure which would increase the training time for these models severely, affecting the number of experiments that can be run in the given timeframe.

3.5.2. Challenges

Hyperparameter tuning

Both algorithms use hyperparameters to function properly [37]. The use of suboptimal values for these hyperparameters can have a severe effect on the overall performance of the model. Additionally, finding optimal values for these hyperparameters can be very challenging. Therefore, a lot of testing will need to be run in order to try and find optimal values for the experiments.

Convergence traps

Although both algorithms have randomness to reduce the likelihood of getting stuck in local optima [40]. Therefore, multiple runs with the same parameters will be run in order to get an average of the results obtained. This will help identify anomalies in the data which were caused by the algorithm getting stuck, thus improving the accuracy of the experiment's findings.

Bias

Evolutionary algorithms as a whole can be prone to bias affecting the results obtained [41]. The use of randomness in initialising the population will help reduce bias but both implementations will also need to be designed whilst keeping the idea of removing bias in mind.

Computational Power

Machine learning in general is a very costly field, both in terms of, time cost and computational power. The use of a dedicated graphics processing unit will essentially be mandatory for any researcher looking to replicate or expand on the findings of this project. This is due to GPUs possessing more cores than CPUs which helps them speed up the training time of these models immensely.

Exploration vs Exploitation

The algorithms will need to balance exploration and exploitation to allow for optimal performance. Particularly, genetic algorithms are more susceptible to getting stuck in local optima whilst exploring [42].

3.6. Conclusion

This project aims to highlight the value evolutionary algorithms can provide for training DNN models in comparison to more traditional methods like SGD. It will also reveal the drawbacks of using evolutionary algorithms such as hyperparameter tuning. Despite these drawbacks, the project's result will highlight the fact that evolutionary algorithms can be a competitive alternative to gradient descent algorithms for training DNN models. The results should show that the algorithms will outperform one another depending on the scenario but always offer competitive performance in comparison to gradient descent algorithms.

To conclude, the result should be valuable research data that future researchers can utilise in their studies to further refine pre-existing algorithms like GA and ES or even come up with new algorithms that intend to solve the drawbacks of these algorithms. Ultimately, this research should help provide a step forward in fully utilising the potential of evolutionary algorithms in training DNNs and maybe even come up with new methods that utilise aspects from both gradient descent and evolutionary algorithms to come up with a hybrid.

4. Development

4.1. Stock Data API

To acquire test data for this experiment, the library yfinance was used to make API calls to download historical stock data from Yahoo! Finance directly [42]. This decision was made as yfinance is a well-established library in Python with a lot of pre-existing support. Additionally, yfinance is free of charge whereas other stock data APIs operate on a subscription-based model. Therefore, the usage of a free API will further make my research data more accessible to any researchers looking to replicate or develop upon my work. This was also the main reason for opting to download historical stock data directly from the website as opposed to shipping the system with a CSV file of stock data as this helps reduce the number of variables in the testing environment as it could be possible for a future researcher to accidentally tamper with a CSV file or have some data become corrupt during download. In addition, this will further help meet one of my project requirements which is to ensure that my system is as robust as possible. Directly loading the data into the code will ensure that the same dataset is utilised throughout different systems.

Furthermore, the yfinance allows future researchers to conduct further experiments with varied configurations very easily. Currently, for this project, the configuration is set to use the entire historical stock data of NASDAQ with a 70-15-15 split, 70% for training, and 15% for validation and testing. This split was chosen as it is very balanced, allowing for a robust training time and unbiased evaluations to occur [43].

4.2. Data preprocessing

As stated in the specification, this project aims to showcase the potential of evolutionary algorithms in DNN training, thus the historical stock data will be heavily filtered. For the project in particular, only the stock's closing value each day will be used. This is due to the stock market being very volatile during the day when trading is occurring as the value of stock is never constant when the market is open. Therefore, using the closing value of the stock for the day ensures there are no unwanted variances in the test data otherwise, it would be difficult to make accurate performance comparisons between the methods due to too many factors.

4.3. Models

4.3.1. LSTM

The LSTM model will be developed using the PyTorch library in Python [44]. This is because the library already has a well-established and active community which will help in troubleshooting any problems that occur during development [45]. Additionally, Pytorch is highly intuitive and has a very easy-to-understand syntax which will further improve the accessibility of this system as not every researcher will be adept in the field of LSTMs or time series-based problems in particular.

Although there are alternatives such as TensorFlow, PyTorch has a much bigger community when it comes to research-based projects and is considered easier to pick up which will further improve the accessibility of my research to any new researchers interested in this field. The

LSTM model developed will be used in combination with an SGD optimiser to obtain a set of baseline results.

Furthermore, Pytorch provides a flexible and dynamic architecture for machine learning models. This would make it very easy for me to add additional features to the system such as dropout layers whereas it would be much more complex to implement it from scratch without the use of such a library. Additionally, PyTorch has a built-in optimiser function. This function will be utilised to use a stochastic gradient descent algorithm to obtain a set of benchmark results from training.

Finally, one of the stated requirements was to ensure the performance of the system is good and can handle large datasets. Pytorch allows for this to happen as it offers efficient GPU acceleration as an option when training models. This will significantly speed up the system as the use of GPUs over CPUs is the primary factor when it comes to training times for machine learning models.

4.3.2. Evolutionary Algorithms

Both of the evolutionary algorithms will be developed using the DEAP library [46]. Similar to PyTorch, DEAP is a library with extensive documentation and an active community. It is also the most popular library when it comes to implementing evolutionary algorithms in Python.

Additionally, DEAP comes with parallelisation support built in which will allow for further improvements in terms of system performance as it will allow the system the utilise multiple threads to speed up training time via the use of concurrency. In addition, DEAP is also highly customisable in almost every aspect. For example, DEAP allows the user to choose which crossover or mutation function they want to use or if they want to use their function that was not included in the library. This will allow the system code to be very efficient as it will drastically reduce the number of lines but will also allow for custom functions to be used if the default functions do not satisfy any requirements for the experiment.

Both algorithms will utilise the main LSTM model to test a set of potential learning rates and find the most optimal learning rate for the model by comparing their performances.

4.4. Visualisation

Another requirement for this project was to visualise the results of the experiments [47]. To achieve this, the library matplotlib will be utilised to read results from the experiments and output them in the form of graphs, primarily training and validation loss graphs. This is a key requirement for this project as it will further enhance the accessibility of the obtained results due to images being much easier to understand and interpret than raw data even if you do not have expertise in the field.

4.5. Overall System

The entire system as a whole will be developed in Python only. This is once again because Python is the most popular language of choice in the world, meaning it is fairly likely for any potential future researchers to be familiar with the syntax of the system and be able to interpret it easily. The main alternative to this would be C, however, C is much more complex compared to Python in terms of readability and ease of use. This is due to the Python syntax being very intuitive and easy to understand even if you are not familiar with Python.

Both of my evolutionary algorithms will also use the same configuration to ensure that the tests are kept fair. For example, parameters like population size will be kept the same along with mutation rate, etc. Both algorithms will also use the same random function to initialise their population as it was one of my functional requirements to ensure that the population was initialised using true randomness which will be achieved using the random.uniform() function from the numpy package.

In addition, due to Python's popularity, it has a much bigger community and overall support than C does. This makes it much easier to find documentation, troubleshooting support and third-party modules that would help speed up development time for the system.

4.6. Evaluation

To effectively evaluate the performance of both evolutionary algorithms on DNN training, the closing value of NASDAQ's stock will be evaluated and compared to the values obtained by the model with each method. Specifically, the training and validation losses will be compared against each other. Additionally, to ensure that the predicted values are still close to the real values, the average MSE will also be recorded for each run.

5. Experiments & Evaluation

5.1. Testing Methodology

To evaluate the performance of both evolutionary algorithms, training and validation losses of each method when training the model will be compared as opposed to comparing actual closing values against predicted closing values. This is because comparing losses provides more insight into the data than just comparing the values. For example, plotting training validation loss curves will allow us to see whether the model is overfitting or underfitting whilst also being able to gauge how well it is generalising.

As for how the evolutionary algorithms will be utilised, they will be used to optimise the hyperparameters of the model. For the sake of this project, only the learning rate will be optimised. This is because the learning rate is one of the biggest factors that affect the overall performance of a machine-learning model. Additionally, only optimising one hyperparameter will make testing a lot easier as it will reduce the overall number of variables at hand, making it much easier to analyse the benefits of using evolutionary algorithms.

Furthermore, these algorithms will also be tested against a benchmark learning rate that will be obtained via preliminary testing through human trial and error. This will be used to gauge the difference in performance that can be achieved via evolutionary algorithms in retrospect to human means. All three methodologies will also keep a set of controlled variables in their configuration. This is to ensure that any performance gained or lost will be caused by the optimisation of the learning rate and nothing else. In addition, the average MSE will also be recorded for each run, this is because MSE is the average difference between predicted and actual values for the closing stock value in this experiment. This will allow us to continue comparing training and validation losses for each method whilst ensuring that the model is still predicting values that are close enough to the actual closing value of the stock.

Finally, like any reliable experiment, the tests will be run multiple times to ensure that the results obtained are reliable and most importantly, reproducible to a certain degree. For this project, I will run each of the three methods, a total of 10 times each to ensure that no anomalies are polluting the data. If there are any obvious anomalies present in any of the runs, I will disregard that run and complete an additional test instead.

5.2. Configuration & Controls

As discussed in the prior section, both algorithms will be tested against a benchmark. This benchmark will be obtained by performing some preliminary testing. This testing will also be used to test several different values for some of the variables that can be adjusted until optimal values are found. These values will be kept consistent throughout each testing methodology to ensure improved performance across the board whilst also keeping the experiments fair.

The first of these hyperparameters to be tested was the batch size. Multiple batch sizes were tested including the default value of 16, 32, 64 and 8. When comparing training loss curves for these, it showed that batch sizes greater than 16 resulted in poor generalisation of the data as the validation loss was much higher than the training loss. Batch sizes of 8 and 16 showed good generalisation of the data but batch size of 8 was significantly slower compared to a batch size of 16 whilst not showing much improvements in terms of generalisation, therefore, the control for batch size will be set to 16 for all three methodologies.

5.2.1. Lookback period

Lookback periods in LSTMs tell the model how many days to look back during training. A larger historical context will potentially result in improved model performance as it will directly increase training time, giving more time for the model to learn more trends but also increasing the risk of model overfitting due to an increase in parameters.

Lookback period values of 7, 10, 20 and 30 were tested. At the lookback period of 30, the model started to show signs of overfitting and also started showing signs of diminishing returns in terms of improvement in accuracy. Therefore, the experiments will be completed on a lookback period of 20 days.

5.2.2. Architecture size

Finding an optimal configuration for the architecture size can be very complex due to multiple parameters being present, thus, each parameter will be tested on its own to see their impact on model performance. The first parameter, input size will not need to be modified as for this experiment, the model will only ever take in 1 type of input which is the stock's closing value.

The second parameter, hidden layer size will be tested with a series of different values, primarily in base 2 so values of 64, 128, 256, 512 and 1024 will be tested. Additionally, to counter the issue of overfitting, regularisation techniques will be utilised by adding dropout to the model during this phase. Further testing showed that a dropout rate of 0.2 was sufficient to prevent overfitting in all tested values with a hidden layer size of 512 showing the best results when comparing model accuracy to training times as hidden layer size 1024 still showed improvements but the training times were significantly longer.

Finally, the number of layers parameter was tested at values of 1, 2, 3 and 4. During testing, it was found that stacking layers of 2 showed a decent improvement in model accuracy with further increases in layers showing diminishing returns. Additionally, at a stacked layer count of 4, the model started showing signs of a vanishing gradient problem occurring, therefore, a stacked layer of 2 will be utilised for the experiments.

With this set of tests completed, it was decided that the model architecture used for the LSTM architecture will consist of input size 1, a hidden layer count of 512 and a total of 2 stacked layers. This model will be kept consistent throughout the entire experiment.

5.2.3. Learning rate

Arguably one of the most important hyperparameters for determining model performance. More extensive testing will be done to find the optimal learning rate. Initially, the default value for the learning rate will be tested and compared against smaller learning rates to see the performance difference.

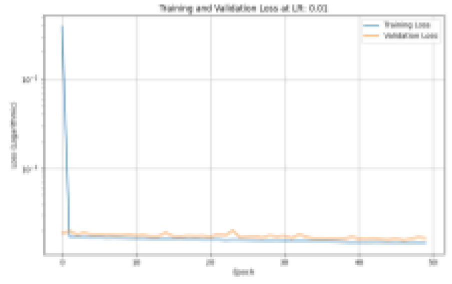

Figure 3: Loss curve from training at 0.01

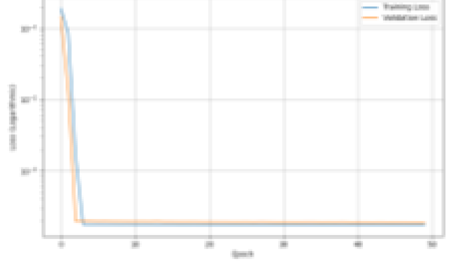

Figure 4: Loss curve from training at 0.001

Figures 3 and 4 show the results from training the model. The graphs show a steep drop in loss, suggesting that the model is prematurely converging into a suboptimal solution. Therefore, more testing is required at lower learning rates

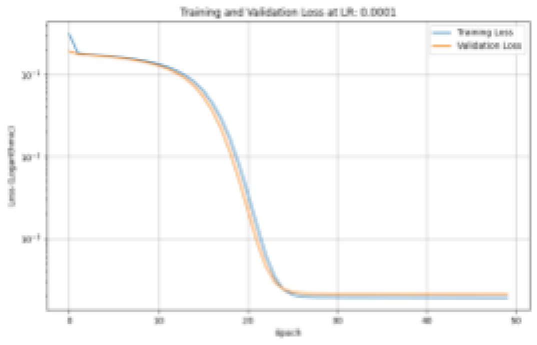

Figure 5: Loss curve form training at 0.0001

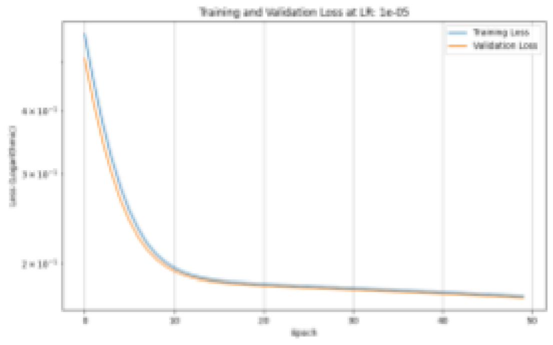

Figure 6: Loss curve from training at 0.00001

Figures 5 and 6 show the results from further testing. Figure 5 in particular still shows the model finding a suboptimal solution during epoch 0-10 also known as a local minimum. However, due to the lower learning rate, the model is eventually able to continue converging to a better solution. Figure 6 is starting to get closer to an optimal learning rate as the convergence is getting smoother however, the model is continuing to converge to a suboptimal solution as the model performance is still plateauing very early into the training cycle.

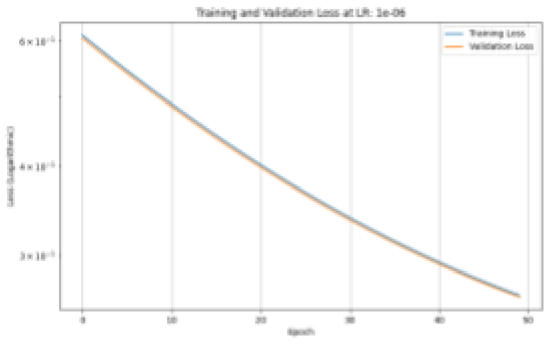

Figure 7: Loss curve from training at 0.000001

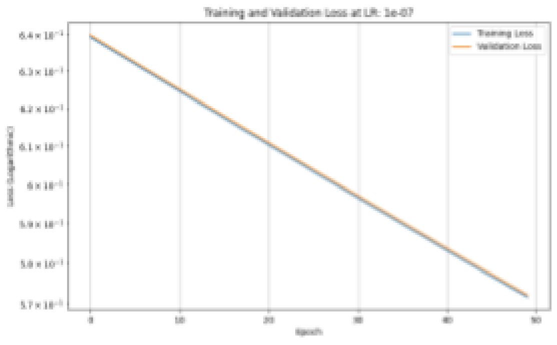

Figure 8: Loss curve from training at 0.0000001

Figure 7 shows good convergence behaviour from the model as it continuously converges into an optimal solution but does not stop converging before training finishes which suggests that more epochs are required. Figure 8 shows a similar story however, the convergence speed shown by the graph is almost linear along with the fact that the validation loss is consistently greater than the training loss. This suggests that the LR has gotten too low to the point that the model has started to show signs of overfitting now, thus, for the experiment's benchmark we will opt to use an LR of 0.000001 instead.

Furthermore, this data will be further used to optimise other aspects of the experiment as now that we know that the optimal learning rate is somewhere in the range of 0.000001 to 0.0000001 we can use this as the range for the random population generation for the evolutionary algorithms.

5.2.4. Number of epochs

As discussed earlier, the model was unable to converge to a solution due to running out of epochs suggesting the default value of 50 was too low. Further testing will be done where the epochs will be increased by 50 each run until the model stops converging.

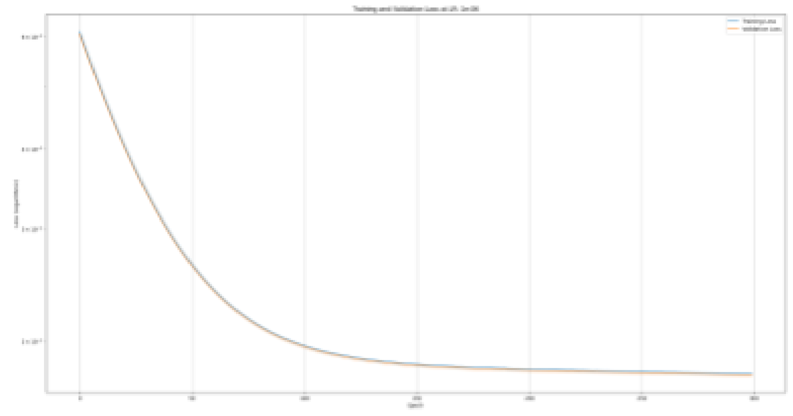

Figure 9: Epochs increased to 300

Figure 9 shows the training results after reaching 300 epochs. The model shows consistent convergence but begins to stagnate at around 300 epochs suggesting an optimal solution has been reached and that further increasing the number of epochs would not show a significant improvement in model performance. Therefore, 300 epochs will be used throughout this experiment as a control.

To conclude, all three testing methodologies will use the following configuration. Batch size at 16, lookback period at 20, architecture size of (1, 512, 2), and number of epochs at 300. For the baseline benchmark, the learning rate will be set at 0.000001. All three methods will be repeated 10 times to make sure that no outliers are present to skew the data as well.

5.3. Results

5.3.1. Stochastic Gradient Descent

Run	Training Loss	Validation Loss	Avg MSE	Runtime

1	0.178	0.177	0.175	950
2	0.178	0.177	0.175	953
3	0.176	0.176	0.174	972
4	0.178	0.177	0.175	962
5	0.178	0.177	0.175	944
6	0.175	0.176	0.176	966
7	0.176	0.175	0.173	954
8	0.174	0.172	0.171	967
9	0.177	0.176	0.174	973
10	0.177	0.176	0.174	978
AVERAGED	0.177	0.176	0.174	962

Table 1 shows the results after training the model 10 times. These results will be used as a benchmark to calculate the improvement in model performance.

5.3.2. Genetic Algorithm

Run	Training Loss	Validation Loss	Avg MSE	Runtime	LR(10^-6)
1	0.0899	0.0890	0.0877	1558	4.59
2	0.0757	0.0748	0.0736	1542	5
3	0.0991	0.0981	0.0967	1556	4.24
4	0.0822	0.0813	0.0801	1555	5
5	0.0567	0.0577	0.0567	1611	5
6	0.116	0.116	0.114	1613	3.91
7	0.0864	0.0856	0.0842	1551	4.72
8	0.0983	0.0975	0.0961	1417	4.43
9	0.117	0.116	0.114	1562	4.01

10	0.0861	0.0852	0.0839	1593	5
AVERAGED	0.0907	0.0901	0.0887	1556	4.59

Table 2 shows the results of using a genetic algorithm function to optimise the learning rate before training. On average, training loss improved by 48.7% whilst validation loss improved by 49.6%. However, the runtime has increased by 61.8% compared to the benchmark.

5.3.3. Evolutionary Strategy

Run	Training Loss	Validation Loss	Avg MSE	Runtime	LR(10^-6)
1	0.0832	0.0823	0.0811	1731	4.94
2	0.0759	0.0752	0.0739	1749	4.95
3	0.0749	0.0741	0.0729	1743	4.76
4	0.0697	0.0688	0.0677	1739	4.92
5	0.0776	0.0769	0.0757	1740	4.83
6	0.0866	0.0858	0.0845	1756	4.80
7	0.0737	0.0729	0.0717	1750	4.90
8	0.0840	0.0831	0.0818	1729	4.80
9	0.0764	0.0756	0.0744	1769	4.89
10	0.0650	0.0641	0.0630	1748	4.91
AVERAGED	0.0767	0.0759	0.0747	1746	4.87

Table 3 shows the results of using an evolutionary strategy function to optimise the learning before training. On average, training loss improved by 56.7% compared to the benchmark and 15.4% better than using a genetic algorithm. Validation loss had a similar story with a 56.9% increase against the benchmark and a 15.7% increase against the genetic algorithm method. However, the runtime performance has continued to get worse with an 81.5% increase compared to the benchmark and a 12.2% increase in comparison to the genetic algorithm.

5.4. Evaluation

As shown by the table of results, both the genetic algorithm and evolutionary strategy showed clear improvement in model accuracy when compared to the benchmark of stochastic gradient descent optimisation with a static learning rate. As for why this is the case, it becomes apparent when looking at the loss curves for both methods.

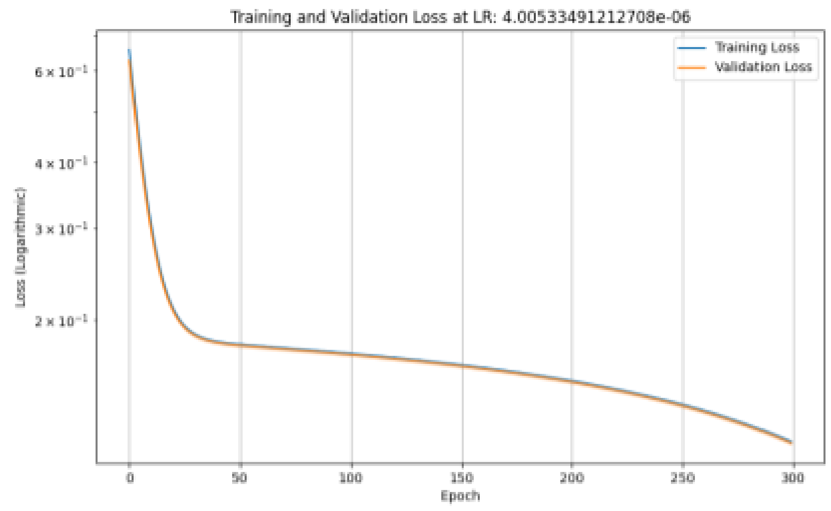

Figure 10: Loss curve when using the genetic algorithm for training

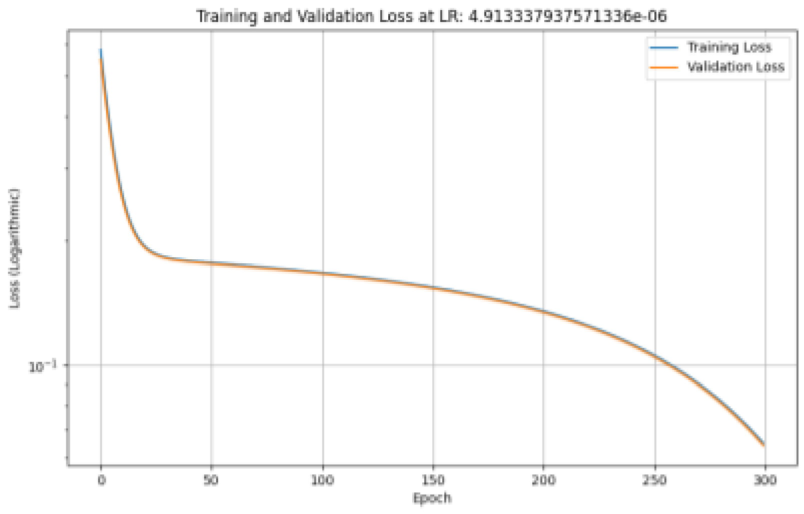

Figure 11: Loss curve when using the evolutionary strategy method

Both Figures 10 and 11 show that the model was getting stuck in a saddle point around epochs 40-60 but both of them were able to escape the step out of the local minimum and escape it [50]. This allowed both methods to obtain better model accuracy as they were able to continue converging after escaping the saddle point. However, with the baseline benchmark, this was not the case as what most likely happened was that the benchmark learning rate was too small which meant that it could not escape the saddle point due to not being able to make big enough steps.

In addition, both methods showed an increased runtime when compared to the benchmark but this was expected due to how the system has been set up. This is due to both evolutionary algorithms using a smaller version of the main training loop to test the population for the best solution. What this effectively means is that both of the methods are using each member of the population to train the model for a smaller number of epochs which in this experiment was set to 50. This meant that it would be physically impossible for either of the algorithms to have a lower runtime than the benchmark.

As for the runtime performance difference that was observed between the genetic algorithm and evolutionary strategy method, it can be determined that it was caused by how each of the algorithms work. Unlike GAs, ES has a higher emphasis on using mutation to obtain optimal solutions whereas GAs focus more on the use of the crossover function hence why there was roughly a 12% difference in runtime performance. Additionally, ES favouring mutation rate is also the reason why it was able to achieve a higher level of model accuracy than GA at an average of around 15% but at the downside of being more computationally expensive.

6. Critical Assessment

The primary objective of this project was to highlight the potential usage of evolutionary algorithms in the field of machine learning. I chose this project specifically as I was interested in evolutionary algorithms from my studies in Year 2 and wondered why their usage was so niche in the field of machine learning as my literature review had found that the use cases for evolutionary algorithms are very niche currently. However, as shown by my test data, evolutionary algorithms are more than viable to be utilised in machine learning compared to just using standard optimisers like SGD as there is always potential for human error as shown in the preliminary testing as the results I obtained did not show any signs of a potential saddle point until I used evolutionary algorithms and compared the results.

Python and PyTorch were a very good choice as coming into this project I had no experience working with machine learning code but the simple syntax and extensive documentation made development much easier than if I had chosen an alternative such as creating my own LSTM model from scratch. The use of Python also allowed me to use third-party libraries to reduce the overall lines of code present in my system whilst also allowing me to further optimise my code as I was able to implement parallelisation into my functions to improve my overall system performance.

However, I believe I could have handled the development phase much better as a lot of my development time was spent further researching machine learning topics as I was still learning the intricacies of machine learning during this phase. I think I would have benefitted from having a roadmap/plan as it would give me clear targets to work towards each week whilst also allowing me to plan ahead of time.

The most time-consuming part was implementing the use of evolutionary algorithms to optimise the hyperparameters. This is due to the lack of documentation and research in this field as mentioned previously, there was very little data available regarding this particular use case. This led to me essentially having to spend most of my designing a way to take a subset of my main model and use it to test all the members of the population which took much longer than expected due to needing time to debug and perform logic checks.

Despite this drawback, I still firmly believe this research project as a whole to be a success as looking back to my design, I proved my hypothesis successfully whilst also meeting most of the requirements except for performance and termination criteria. I did not bother implementing the termination criteria requirement as I deemed it unnecessary for my set of experiments but if needed I could easily meet this requirement by utilising early stopping in my code. As for the performance requirement, it is the one requirement that I could not meet due to how the test system has been implemented I have optimised it as much as possible but the performance is never going to be competitive to traditional methods due to the implementation.

7. Conclusion & Future Research

In conclusion, the main aim of this project was to showcase the possibility of using evolutionary algorithms for training DNN models. I believe this has been more than achieved as my test data showed on average a 50% increase in terms of model accuracy, showcasing the potential to use evolutionary algorithms instead of only utilising traditional optimisers. There was also a significant increase in runtime due to the increased computational costs, however, I believe this should not be seen as a significant drawback for using evolutionary algorithms in this scenario as model training time will always largely be dependent on the computational power available as it does not matter how optimised your code is if you do not have the hardware to run the model. Fortunately, computer hardware is improving at an exponential rate year by year, with leading GPU manufacturer NVIDIA increasing their computing power by 1000x in 8 years which even surpasses Moore's Law [48, 49]. Therefore, I fully believe that evolutionary algorithms will see an increase in usage in machine learning as hardware continues to improve evolutionary algorithms allow for automation of finding optimal learning rate which is a very big positive as it reduces human involvement in training, reducing the possibilities for human errors.

As for future research, I would aim to further develop my findings by exploring the effects on model accuracy and performance when changing multiple hyperparameters as I stated in my development I opted to optimise only one hyperparameter to showcase the potential of evolutionary algorithms. Now that this has been achieved, the next logical step would be to test the results of optimising other hyperparameters. Furthermore, I would also like to adapt the methods so that they can be utilised to train different types of models i.e. ResNet Model for Image Classification Tasks. This would allow more insight into my test data as we can then compare the performance of the algorithms across a wider range of tasks rather than just time series forecasting.

8. References

[1] Langley, P. (2011). The changing science of machine learning. *Machine learning*, *82*(3), 275-279.

[2] Liu, W., Wang, Z., Liu, X., Zeng, N., Liu, Y., & Alsaadi, F. E. (2017). A survey of deep neural network architectures and their applications. *Neurocomputing*, *234*, 11-26.

[3] Wu, M., & Chen, L. (2015, November). Image recognition based on deep learning. In *2015 Chinese automation congress (CAC)* (pp. 542-546). IEEE.

[4] Chowdhary, K. R. (2020). *Fundamentals of artificial intelligence* (pp. 603-49). New Delhi:: Springer India.

[5] Sze, V., Chen, Y. H., Yang, T. J., & Emer, J. S. (2017). Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE*, *105*(12), 2295-2329.

[6] Siddique, N., & Adeli, H. (2015). Nature inspired computing: an overview and some future directions. *Cognitive computation*, *7*, 706-714.

[7] Forrest, S. (1996). Genetic algorithms. *ACM computing surveys (CSUR)*, *28*(1), 77-80.

[8] Beyer, H. G., & Schwefel, H. P. (2002). Evolution strategies–a comprehensive introduction. *Natural computing*, *1*, 3-52.

[9] Feurer, M., & Hutter, F. (2019). Hyperparameter optimization. *Automated machine learning: Methods, systems, challenges*, 3-33.

[10] Sharma, N., Sharma, R., & Jindal, N. (2021). Machine learning and deep learning applications-a vision. *Global Transitions Proceedings*, *2*(1), 24-28.

[11] Chen, X., Liu, X., Gales, M. J., & Woodland, P. C. (2015, April). Improving the training and evaluation efficiency of recurrent neural network language models. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (pp. 5401-5405). IEEE.

[12] Kulin, M., Kazaz, T., De Poorter, E., & Moerman, I. (2021). A survey on machine learning-based performance improvement of wireless networks: PHY, MAC and network layer. *Electronics*, *10*(3), 318.

[13] Colbrook, M. J., Antun, V., & Hansen, A. C. (2022). The difficulty of computing stable and accurate neural networks: On the barriers of deep learning and Smale's 18th problem. *Proceedings of the National Academy of Sciences*, *119*(12), e2107151119.

[14] Hutter, F., Lücke, J., & Schmidt-Thieme, L. (2015). Beyond manual tuning of hyperparameters. *KI-Künstliche Intelligenz*, *29*, 329-337.

[15] Sun, P., Wen, Y., Han, R., Feng, W., & Yan, S. (2019). Gradientflow: Optimizing network performance for large-scale distributed dnn training. *IEEE Transactions on Big Data*, *8*(2), 495-507.

[16] Arora, J. S. (2006). Jan A. Snyman, Practical Mathematical Optimization: An introduction to basic optimization theory and classical and new gradient-based algorithms. *Structural and Multidisciplinary Optimization*, *31*(3), 249-249.

[17] Amari, S. I. (1993). Backpropagation and stochastic gradient descent method. *Neurocomputing*, *5*(4-5), 185-196.

[18] Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

[19] Bäck, T., & Schwefel, H. P. (1993). An overview of evolutionary algorithms for parameter optimization. *Evolutionary computation*, *1*(1), 1-23.

[20] Sivanandam, S. N., Deepa, S. N., Sivanandam, S. N., & Deepa, S. N. (2008). *Genetic algorithms* (pp. 15-37). Springer Berlin Heidelberg.

[21] Aszemi, N. M., & Dominic, P. D. D. (2019). Hyperparameter optimization in convolutional neural network using genetic algorithms. *International Journal of Advanced Computer Science and Applications*, *10*(6).

[22] Gottapu, R. D., & Dagli, C. H. (2020). Efficient architecture search for deep neural networks. *Procedia Computer Science*, *168*, 19-25.

[23] Abramson, D., & Abela, J. (1991). *A parallel genetic algorithm for solving the school timetabling problem* (pp. 1-11). Canberra, Australia: Division of Information Technology, CSIRO.

[24] Hansen, N., Arnold, D. V., & Auger, A. (2015). Evolution strategies. *Springer handbook of computational intelligence*, 871-898.
[25] Salimans, T., Ho, J., Chen, X., Sidor, S., & Sutskever, I. (2017). Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*.

[26] Tirumala, S. S. (2020). Evolving deep neural networks using coevolutionary algorithms with multi-population strategy. *Neural Computing and Applications*, *32*(16), 13051-13064.

[27] LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *nature*, *521*(7553), 436-444.

[28] Nwankpa, Chigozie, et al. "Activation functions: Comparison of trends in practice and research for deep learning." *arXiv preprint arXiv:1811.03378* (2018).

[29] Bengio, Y., Goodfellow, I., & Courville, A. (2017). *Deep learning* (Vol. 1). Cambridge, MA, USA: MIT press.

[30] Kelleher, J. D. (2019). *Deep learning*. MIT press.

[31] Chen, J. Y. (2009, December). GPU technology trends and future requirements. In *2009 IEEE International Electron Devices Meeting (IEDM)* (pp. 1-6). IEEE.

[32] Bialer, O., Garnett, N., & Tirer, T. (2019, May). Performance advantages of deep neural networks for angle of arrival estimation. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (pp. 3907-3911). IEEE.

[33] Amari, S. I. (1993). Backpropagation and stochastic gradient descent method. *Neurocomputing*, *5*(4-5), 185-196.

[34] Bottou, L. (2012). Stochastic gradient descent tricks. In *Neural Networks: Tricks of the Trade: Second Edition* (pp. 421-436). Berlin, Heidelberg: Springer Berlin Heidelberg.

[35] Bottou, L. (1991). Stochastic gradient learning in neural networks. *Proceedings of Neuro-Nımes*, *91*(8), 12.

[36] Bottou, L. (2010). Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010: 19th International Conference on Computational StatisticsParis France, August 22-27, 2010 Keynote, Invited and Contributed Papers* (pp. 177-186). Physica-Verlag HD.

[37] Bardenet, R., Brendel, M., Kégl, B., & Sebag, M. (2013, May). Collaborative hyperparameter tuning. In *International conference on machine learning* (pp. 199-207). PMLR.

[38] Figure 1.
https://s7280.pcdn.co/wp-content/uploads/2020/07/Two-or-more-hidden-layers-comprise-a-Deep-Neural-Network.png

[39] Figure 2.
https://www.researchgate.net/figure/Google-search-trend-showing-increased-attention-in-deep-learning-over-the-recent-years_fig3_348898137

[40] Nijssen, S., & Back, T. (2003). An analysis of the behavior of simplified evolutionary algorithms on trap functions. *IEEE Transactions on Evolutionary Computation*, *7*(1), 11-22.

[41] Raidl, G. R., & Gottlieb, J. (2005). Empirical analysis of locality, heritability and heuristic bias in evolutionary algorithms: A case study for the multidimensional knapsack problem. *Evolutionary computation*, *13*(4), 441-475.

[42] Alshahrani, A., Namazi, N. M., Abdouli, M., & Alqarni, M. A. (2017, October). Escaping the local optima trap caused by PSO by hybridization scheme for elongate the WSN's lifetime. In *2017 8th IEEE Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)* (pp. 121-127). IEEE.

[42] https://pypi.org/project/yfinance/

[43]
https://pub.towardsai.net/breaking-the-mold-challenging-the-common-split-for-training-validation-and-test-sets-in-machine-271fd405493d

[44] Sunny, M. A. I., Maswood, M. M. S., & Alharbi, A. G. (2020, October). Deep learning-based stock price prediction using LSTM and bi-directional LSTM model. In *2020 2nd novel intelligent and leading emerging sciences conference (NILES)* (pp. 87-92). IEEE.

[45] https://pytorch.org/docs/stable/index.html

[46] https://deap.readthedocs.io/en/master/

[47] https://matplotlib.org/

[48]
https://www.nextbigfuture.com/2024/03/nvidia-increased-compute-power-1000x-in-8-years-to-20-petaflops-in-the-blackwell-gpu.html#:~:text=the%20Blackwell%20GPU-,Nvidia%20Increased%20Compute%20Power%201000X%20in%208%20Years,Petaflops%20in%20the%20Blackwell%20GPU&text=Nvidia%20has%20increased%20compute%20power,of%20their%20innovation%20is%20increasing.

[49] https://www.investopedia.com/terms/m/mooreslaw.asp

[50]
https://medium.com/@sujathamudadla1213/what-is-saddlepoint-problem-in-machine-learning-5e4ddb81bd18