An Analysis of Optimization Algorithms on the JUAN La Salle Chatbot

Isobelle Linden C. Casado
De La Salle University - Dasmariñas
cic0030@dlsud.edu.ph

James Martin L. Rafols
De La Salle University - Dasmariñas
rjl0827@dlsud.edu.ph

David Allain R. Yabis
De La Salle University - Dasmariñas
ydr0111@dlsud.edu.

ABSTRACT

Machine learning and deep learning techniques enable people to predict data and patterns by analyzing datasets and applying neural network models and deep learning techniques to process these datasets. These models and techniques often employ the use of optimizing algorithms to improve accuracy and minimize loss, approaching an optimum that yields the least loss. Many optimizers have since then been developed from Stochastic Gradient Descent (SGD) to optimizers that apply a combination of SGD and momentum. Among these optimizers are Adam, Nadam, Adamax, Adagrad, and RMSProp. The study conducted experimentation utilizing the previously mentioned optimizers to the chatbot, JUAN La Salle, which employs a Feed-forward Neural Network to classify user input to reply with its appropriate response, enabling it to respond to different patterns in user input. The researchers ran all the optimizers with 5000 epochs and then recorded and averaged the loss functions of each optimizer. The runtime of each optimizer is also recorded. The researchers were able to determine the best optimizer by calculating the ratio for average loss over the runtime (seconds) for each optimizer among Adam, Nadam, Adamax, Adagrad, and RMSProp.

Keywords
Chatbot, optimizers, PyTorch , deep learning, feed forward neural networks

1. INTRODUCTION

Machine learning has grown and is still growing at a remarkable rate in this modern day and age. It has become one of the pinnacle of modern technology, from artificial intelligence found in your local chatbot to something as advanced as self-driving vehicles, machine learning has taken modern technology to a whole new level. Having said that, one of the main components of this is optimization. It is one of the driving forces of machine learning. The goal of learning algorithms is to create an optimized model and learn from it. The parameters of the objective function based on the data provided. The effectiveness and efficiency of numerical optimization methods have a significant impact on the popularization and implementation of machine learning models in the age of massive data. An optimizer is a function or algorithm that alters the characteristics of a neural network, such as its weights and learning rate. As a result, it aids in the reduction of total loss and the improvement of accuracy. It determines the value of the parameters (weights) that reduce error when mapping inputs to outputs. The accuracy of the deep learning model is greatly influenced by these optimization techniques or optimizers. They also have an impact on the model's speed training.

Through this, the researchers developed a chatbot utilizing natural language processing tools and machine learning, specifically the feedforward neural network and the ADAM optimizer. While ADAM has proven to be effective and useful for the chatbot, it still begs the question of whether this route is the best or not. In order to increase the performance and efficiency of machine learning systems, a number of useful optimization strategies were proposed.

The researchers aim to test and experiment with five different machine learning optimizers. Namely Adam, Nadam, Adamax, Adagrad, and RMSProp. These optimizers will be put to test with the same benchmark/control variables: hardware, epochs, batch size, learning rate, input, and neural network. Hardware also affects the training of the model. The hardware specifications used for this analysis comprises the following: a Ryzen 5 3600 CPU, 16GBs of DDR4

RAM, a GTX 1660 Super OC (GPU), and a standard PCIE Gen 3 SSD.

The main goals of this research is to identify the following:

1. Which optimizer produces the least amount of loss.
2. Which optimizer has the fastest runtime.
3. Among the five optimizers implemented, which has the lowest loss per second of runtime.

The optimizer that has the lowest loss per second will be the best suited optimizer for the JUAN La Salle Chatbot and its further development.

2. RELATED WORK AND TERMINOLOGY

2.1 Optimizers

The task of minimizing an objective function, the cost or loss function, parametrized by the model's parameters, is known as optimization in machine and deep learning. By computing gradients in an attempt to minimize the objective function, also known as the loss function, optimization algorithms define the foundation of a machine's ability to learn through experience. If the objective function is convex, any local minimum is a global minimum, and if the objective function is not convex, as in most deep learning issues, the goal of optimization techniques is to discover the lowest possible value of the objective function.

Stochastic Gradient Descent (SGD)

Stochastic Gradient Descent (SGD) is a variant of vanilla gradient descent that trains on individual data elements in contrast to vanilla gradient descent's method of training the entire dataset. In terms of computation and performance, SGD outperforms vanilla gradient descent. SGD randomly or stochastically selects a data instance from the training set and then updates the weight on gradients for that data instance, uniformly selecting a random data instance from the dataset and using that instance's gradient of the model loss. Although in practice, SGD updates model parameters on mini-batches of data, which is similar to vanilla gradient descent. This method takes advantage of the parallelization capability of Graphics Processing Units (GPUs), in which the model function can process multiple data sets. This method takes advantage of the ability of Graphics Processing Units (GPUs) of parallelization, in which the model function can process multiple data instances at the same time, allowing for faster training time with minimal noise.

This makes SGD work significantly superior to vanilla gradient descent because of noisy parameter updates in SGD. More specifically, SGD updates the parameters after forward-prop on every data instance, allowing the parameters to bounce around which reduces the probability of getting stuck in a local minimum, also known as annealing, which is a common problem in vanilla gradient descent.

SGD + Momentum

Momentum is a technique for accelerating SGD in the desired direction while dampening oscillations. It accomplishes this by adding a portion of the previous time step's update vector to the current update vector. Using momentum is essentially pushing a ball down a slope. As the ball rolls downhill, it gathers momentum and accelerates (until it achieves its terminal velocity, which is 1 if there is no air resistance). The momentum term grows for dimensions whose gradients point in the same directions and decreases for dimensions whose gradients change directions for our parameter updates. Faster convergence and less oscillation is acquired as a result of this.

Adam

One improvement from Momentum + SGD is Adaptive Moment Estimation or also known as Adam. Adam, like Adadelta and RMSprop, preserves an exponentially decaying average of previous squared gradients vt , but he also keeps an exponentially decaying average of past gradients mt. Adam acts like a heavy ball with friction, preferring flat minima on the error surface, whereas momentum behaves like a ball traveling down a slope. The decaying averages of past and past squared gradients, mt and vt , are computed in the following way: RMSProp is a derivation of AdaGrad, a stochastic gradient descent algorithm that has different learning rates for each of its parameters (variables). The major difference RMSProp has with AdaGrad is that the gradient gt is calculated by an

exponentially decaying average, instead of the sum of its gradients.

Nadam

The Nesterov-accelerated Adaptive Moment Estimation, or Nadam, algorithm adds Nesterov's Accelerated Gradient (NAG) or Nesterov momentum, which is a better type of momentum, to the Adaptive Movement Estimation (Adam) optimization process. It's an Adam extension that uses NAG momentum rather than classical momentum.

Nadam improves performance by using decaying step size (alpha) and initial moment (mu) hyperparameters. For the sake of simplicity, this aspect will be ignored and assume constant values for the time being.

Adagrad

Adaptive Gradient Algorithm (Adagad) is a gradient-based optimization that adapts its learning rate component-wise to the parameters, incorporating knowledge of past observations. It makes larger updates for parameters with infrequent features and smaller updates on frequent features, making it particularly excel at dealing with sparse data such as for NLP or image recognition. Thus, Adagad has no need for manually tuning the learning rate and is not very sensitive to the size of the master step. Convergence is also faster and more reliable than with vanilla SGD with unequal weight scaling.

RMSProp

Root Mean Squared Propagation (RMSProp) is an extension of gradient descent and Adagrad. It uses a decaying average of partial gradients in adapting step size for each parameter, allowing the algorithm to simply forget early gradients to focus on the most recent observations of partial gradients during the search, essentially overcoming Adagrad's limitations.

PyTorch

PyTorch is a Deep Learning tensor library based on Torch's functionalities with a Python API for easy integration into Python workflows. It uses dynamic computation graphs making it unique to other Deep Learning frameworks such as TensorFlow and Keras. It also allows users to test portions of the code in real-time instead of running the whole code to check just one part of the code.

Pytorch has two main features. Much like NumPy, PyTorch implements Tensor Computation with Graphical Processing Unit (GPU) acceleration support. It also has automatic differentiation for creating and training deep neural networks.

Feed-forward Neural Network

Feed-forward neural networks, also known as multilayer perceptrons (MLP), are one of the most popular and commonly used neural network models due to their practical applications. The feed-forward neural network is biologically inspired in that it mimics the characteristics of actual neurons in brains, making calculations through a synthetic network of neurons known as artificial neural networks. Formally, a feed-forward neural network is a classification algorithm. It is made up of a number of basic neuron-like processing units that are arranged in layers, with each layer's units coupled to those in the previous layer. Each link may have a different strength or weight, therefore they are not all equal. The weights on these connections represent a network's knowledge. The units of a neural network are frequently referred to as nodes.

Generally, there are three layers in a feedforward neural network: the input layer, the hidden layer, and the output layer. Data enters through neurons in the input layer and passes through each layer in the network, finally arriving at the output. Each layer consists of its own types of neurons; the input layer consists of only the inputs to the network, following a hidden layer that consists of a number of neurons (hidden units) in parallel. Through each neuron, a weighted summation of the inputs is performed which then passes to the activation function (the neuron function).

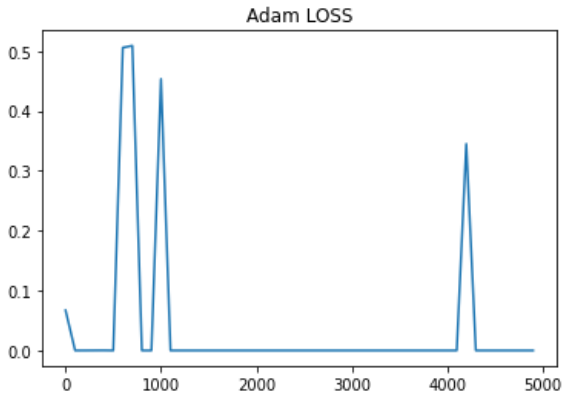
3. EXPERIMENTAL RESULTS

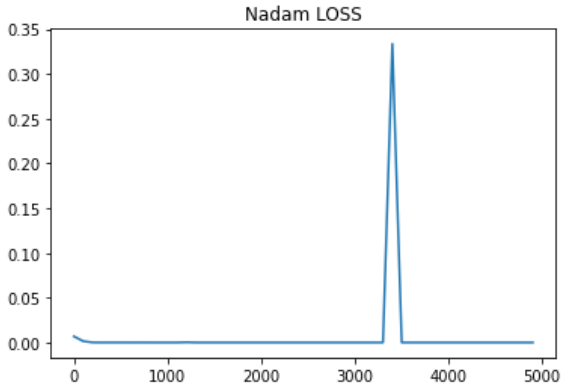
The researchers limited the number of optimizers to those which are similar to the Adam optimizer. Thus Nadam, Adamax, Adagrad, and RMSProp are chosen along with Adam.

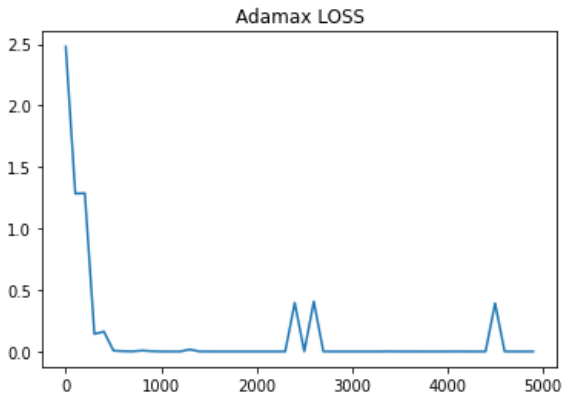
The experiment took a day of coding and training. The experiment rig took a heavy toll with multiple crashes despite having sufficient resources. The researchers

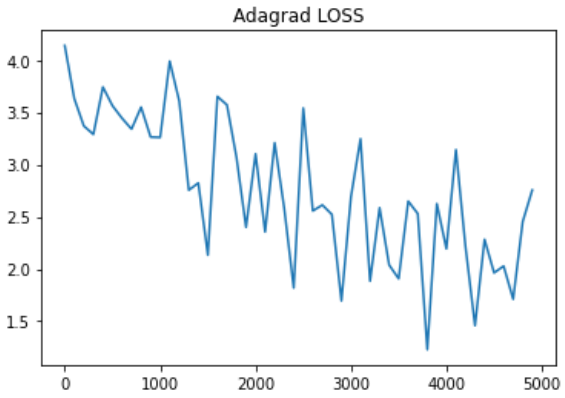
suspect that it might be because of background processes and/or because of incompatible software configurations.

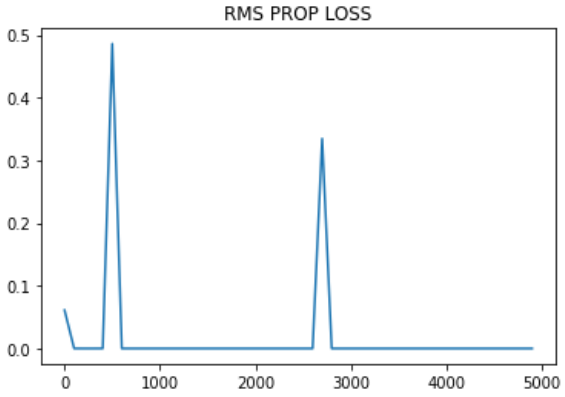
The experiment is conducted by running the same code underneath JUAN La Salle. The same neural network model and input was also implemented. All optimizers will be run with 5000 epochs. With that the researchers trained the data using the five (5) optimizers. The loss of Adam, Nadam, Adamax, Adagrad, and RMSProp, respectively, are in the following figures. Note that the y-axis represents the loss function, and the x-axis represents the number of epochs.



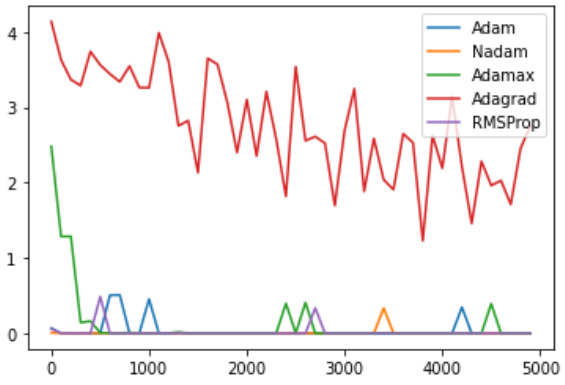








The following figure represents the plotted line graphs of each optimizer in one graph for simpler comparison. Note that it is more ideal the line graph approaches zero as this means that the optimizer is more accurate at this particular epoch.



In order to make a conclusion, the researchers averaged the loss functions of all five optimizers. The following figure describes the data gathered in arbitrary order. As can be observed, Adam has the lowest average loss at

0.006845, followed by RMSProp at 0.017641, average loss and Adagrad with the highest average loss at 2.763669.

	OPTIMIZERS	AVERAGE LOSS
0	Adam	0.037603
1	Nadam	0.006845
2	Adamax	0.131737
3	Adagrad	2.763669
4	RMSProp	0.017641

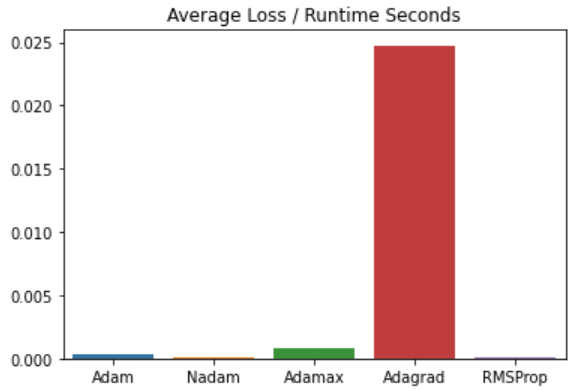
After calculating the loss, the runtime was also measured. Using the python *time* library, the training time was measured with real world units, in this case using seconds. The following figures are the results following the order in the previous figure. As can be observed in the figure, Adagrad scores the lowest at 111.67 runtime seconds, followed by RMSProp at 117.36 runtime seconds, and with Adamax at 157.80 runtime seconds. Note that Nadam is second the slowest at 136.19 runtime seconds.

	OPTIMIZERS	AVERAGE LOSS	RT SECS
0	Adam	0.037603	134.51
1	Nadam	0.006845	136.19
2	Adamax	0.131737	157.80
3	Adagrad	2.763669	111.67
4	RMSProp	0.017641	117.36

With the given data, the researchers calculated the best optimizer by solving the ratio for average loss by the runtime seconds of the optimizers. The succeeding findings are in the following figure. Nadam has the lowest average loss over runtime at 0.000050, followed by RMSProp at 0.000150 average loss over runtime, with Adagrad with the highest average loss over runtime at 0.024749.

	OPTIMIZERS	AVERAGE LOSS	RT SECS	AVE / RT
0	Adam	0.037603	134.51	0.000280
1	Nadam	0.006845	136.19	0.000050
2	Adamax	0.131737	157.80	0.000835
3	Adagrad	2.763669	111.67	0.024749
4	RMSProp	0.017641	117.36	0.000150

4. CONCLUSION AND FUTURE RESEARCH DIRECTIONS



Adagrad performs the least with an average loss over time of 0.024749. Making it the most unoptimized choice for JUAN La Salle. While Adamax and Adam came at 4th and 3rd respectively, with only a 0.1 difference. The optimizer that showed the best results is the Nadam optimizer. Nadam showed a 0.000050 rating, followed by RMSProp with a 0.000150, the lowest loss produced among the five (5) optimizers. Nadam, despite having a relatively mediocre runtime, showed the least amount of loss per second, making it the ideal optimizer for the JUAN La Salle Chatbot system.

Though Nadam yields the best results, the researchers recommend to try and explore other optimizers and experiment with them to see how the results would compare to the initial findings of the study. It is important to note that the study did not use epochs as a variable, and so the researchers also recommend adjusting the number of epochs for each optimizer in future studies. Different neural networks may also affect the results of the study as well as experimenting with different epochs. The researchers also recommend experimenting with other machine learning platforms such as tensorflow, sklearn and many more platforms as they may or may not prove to be better than the platform used in the study.

5. REFERENCES

[1] Shialing Sun. A survey of optimization methods from a machine learning ... Retrieved February 5, 2022 from https://arxiv.org/pdf/1906.06821.pdf

[2] Anon. 2021. A comprehensive guide on Deep learning optimizers. (October 2021). Retrieved February 5, 2022 from https://www.analyticsvidhya.com/blog/2021/10/a-comprehensive-guide-on-deep-learning-optimizers/#:~:text=An%20optimizer%20is%20a%20function,loss%20and%20improve%20the%20accuracy.

[3] Jason Brownlee. 2021. Gradient descent optimization with Nadam from scratch. (October 2021). Retrieved February 5, 2022 from https://machinelearningmastery.com/gradient-descent-optimization-with-nadam-from-scratch/#:~:text=Mini%2DCourse%20now!-,Nadam%20Optimization%20Algorithm,an%20improved%20type%20of%20momentum

[4] Anon. 2021. What is Adagrad? (June 2021). Retrieved February 5, 2022 from https://databricks.com/glossary/adagrad

[5] Senthilkumar, M. (2014, March 27). *Use of artificial neural networks (Anns) in colour measurement*. Colour Measurement. Retrieved January 24, 2022, from https://www.sciencedirect.com/science/article/pii/B9781845695590500042

[5] Simplilearn. 2021. What is pytorch, and how does it work?: Simplilearn. (December 2021). Retrieved February 5, 2022 from https://www.simplilearn.com/what-is-pytorch-article

[6] Kingma, D. P., & Ba, J. L. (2015). Adam: a Method for Stochastic Optimization. International Conference on Learning Representations, 1–13.