ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal
Regular Issue, Vol. 9 N. 2 (2020), 79-90
eISSN: 2255-2863
DOI: https://dx.doi.org/10.14201/ADCAIJ2020927990

Enhancing Performance of a Deep Neural Network: A Comparative Analysis of Optimization Algorithms

Noor Fatima

Department of Computer Science, Aligarh Muslim University, Aligarh, India
nfatima111@myamu.ac.in

KEYWORD

Adadelta; Adagrad; Adam; Adamax; Deep Learning; Neural Networks; Nadam; Optimization algorithms; RMSprop; SGD

ABSTRACT

Adopting the most suitable optimization algorithm (optimizer) for a Neural Network Model is among the most important ventures in Deep Learning and all classes of Neural Networks. It's a case of trial and error experimentation. In this paper, we will experiment with seven of the most popular optimization algorithms namely: sgd, rmsprop, adagrad, adadelta, adam, adamax and nadam on four unrelated datasets discretely, to conclude which one dispenses the best accuracy, efficiency and performance to our deep neural network. This work will provide insightful analysis to a data scientist in choosing the best optimizer while modelling their deep neural network.

1. Introduction

As we advance in the field of Deep Learning and devise diverse models, we realize how important the role optimizers have to play in the learning process. The scope of experimentation arises when we come across choosing the best optimizer for our task. We perceived a necessity to determine which optimizer bestows the best results. Optimizers or Optimization algorithms are used in training a model. They are one of the two parameters required compulsorily to compile a model (the other being loss function i.e., the way of measuring the validity of our predictions). They update the weight parameters to minimize the error function E(x). During training optimizers shape and adapt our model into its most efficient and usable form by playing and futzing with the weights. The error function is a mathematical function relying on the model's internal learnable parameters used to compute the target values/ dependent variable (y) from the set of predictors/ matrix of features (x) present in the model. Those internal parameters play a very significant role in efficiently and effectively training a model

and processing accurate results. Therefore, we use multiple optimization approaches and algorithms to estimate optimum values of the model's parameters which influence its learning process and output.

Keeping in mind our objective i.e. finding the optimum model prediction accuracy, we incorporate optimizers beginning with Stochastic Gradient Descent (SGD). Followed by Root Mean Square Propagation (RMSprop), Adaptive Gradient Algorithm (Adagrad), Adadelta, Adaptive Moment Estimation (Adam): known to be the most suitable for almost all types of neural network models, Adamax and Nesterov-accelerated Adaptive Moment Estimation (Nadam). Without going much into their theoretical aspects and formulations, we will be using the aforesaid in our experimental work to ascertain the best Optimizer with respect to its training and validation accuracies.

In this paper, we will be using four Deep Neural Network (DNN) Models. First: I developed for the classification of various Indian rivers based on their elemental toxicity. Second: a survey model of students willing to attend a workshop. Third: a self-made dataset of undergraduate students of an Institution who pursued Masters from the same University. Finally, the famous Titanic dataset for standardization of our results. The root DNN model is capable of classifying almost any type of data with a little adjustment of parameters. We will experiment the effect of various optimizers on these four models and deduce which one provides maximum training and validation accuracy. Lastly, we compare the performance of all the concerned optimizers by plotting graphs of analysis for all the four datasets under study.

2. State of the Art

Most of the previous works on Deep Neural Networks have incorporated SGD, RMSprop and Adam particularly in their respective models for prediction or classification. In order to equitably determine which one provides the best performance and how, we wish to compare the working of other popular optimizers as well namely: Adagrad, Adadelta, Adamax and Nadam. Much of the prevalent research has been performed in the way to optimize the already available optimization algorithms but not enough emphasis is laid on why we prefer one over the other by integrating them actually in a model. Although similar works have been performed both for Convolutional and Deep Neural Networks but no such work involving comparison of optimizers by using their accuracy matrices was observed.

My work gained inspiration from the latest and most related paper by authors Dami Choi et al. (2020). The authors experimentally demonstrated that the hyperparameter search space may be the single most important factor explaining the rankings obtained by recent empirical comparisons in the literature. They found the accepted adaptive gradient algorithms never underperform momentum or gradient. Our work will eventually prove to be consistent with the claims of this paper.

3. Why Optimizers?

Optimization Algorithms or Optimizers constitute to be a key ingredient in enhancing the performance of a neural network. They tune the hyperparameters of a model according to their design in a conventional manner. *Hyperparameters* which influence the conduct of an optimizer like learning rate controls its *update rule*, which in turn defines the optimizer itself. Any two optimizers can be distinguished by the combination of their hyperparameters and update rule. Particularly for the case

of Deep Learning models, we supply a loss function (can be assumed as the objective function to be targeted for our problem of optimization) to our model during training, which along with the optimization algorithm, is the mandatory parameter for our model's compilation. The role of an optimizer is to play with the weights and learning rate of the nodes of our model under the training process, such that it successfully minimizes the loss function. Summarizing, the fundamental goal of an optimizer is to reduce the training error. Now we shall briefly discuss (regarding) the Optimizers we are going to employ in our experimentation.

A. *Stochastic Gradient Descent (SGD)*

Stochastic Gradient Descent or SGD is a variant of the most basic optimization Algorithm known as Gradient (or slope of a function) Descent. The latter being too simple to be used in Deep Learning, has major applications in Linear Regression, Classification, Backpropagation, etc. with the advantage of ease in computability and implementation. Beginning from an initial value, it aims to reach the lowest amount of the cost function using down stepping or iterations. It can be used with almost all learning algorithms. Stochastic (means random) Gradient Descent, on the other hand selects a few random samples rather than the entire dataset for every single iteration, thus largely improving the speed of estimation. It uses only one static learning rate for the course of the complete training phase for all parameters.

$$SGD\ update\ rule \text{ by Robbins \& Monro, 1951: } (H_i, \beta_i)$$

Contemplate a differentiable loss function: ∂*, of a neural network over an entire dataset as,* $\partial: \mathbb{R}_d \rightarrow \mathbb{R}$ *having* $\nabla\partial(\alpha)$ *as its first partial derivative. The gradient on mini-batch* $\alpha 0 \in \mathbb{R}_d$*, initializes our algorithm with its hyperparameter being a learning rate schedule* $\beta: \mathbb{N} \rightarrow (0,\infty)$*;* H_i*=history;* ∂*=loss function.*

$$\alpha_{i+1} = \alpha_i - \beta_i \nabla\partial(\alpha_i) \qquad \text{- (1)}$$

B. *Root Mean Square Propagation (RMSprop)*

The Root Mean Square Propagation or RMSprop has a very interesting fact associated with it, i.e. although it being so popular, it's an optimizer that was never published. Geoff Hinton, the father of backpropagation proposed it in his online course on Neural Networks for machine learning. RMSprop and Adadelta both came into the picture simultaneously but independently with the objective to cope with the vanishing learning rates of Adagrad. RMSprop is a gradient based optimizer which instead of considering the learning rate as a hyperparameter, utilizes an adaptive learning rate that changes over time.

$$RMSprop\ update\ rule\ by \text{ Tieleman \& Hinton, 2012: } (H_i, \beta_i, \gamma, \rho, \varepsilon)$$

m,u $\in$ *moving averages ;* $\gamma \in [0,\infty]$ *= a momentum parameter;* ε *is a smoothing term to prevent division by zero.*

$$u_0 = 1, m_0 = 0$$
$$u_{i+1} = \rho u_i + (1-\rho)\nabla\partial(\alpha_i)^2$$
$$m_{i+1} = \gamma m_i + \frac{\beta_i}{\sqrt{u_{i+1}+\varepsilon}}\nabla\partial(\alpha_i)$$

$$\alpha_{i+1} = \alpha_i - m_{i+1} \quad \text{- (2)}$$

C. Adaptive Gradient Algorithm (Adagrad)

Adaptive Gradient Algorithm (Adagrad) is very similar to stochastic gradient descent algorithm but unlikely uses adaptive gradients to improve robustness as shown by Dean et al. One of the main advantages of Adagrad is to obliterate the manually tuning of its learning rate, while its most significant flaw being the aggregation of squared gradients in the denominator. During training, with every added term being positive, the aggregate sum builds up, causing the learning rate to decline gradually becoming imperceptible, and thus our algorithm loses its ability to obtain more knowledge. Adadelta aims to win over this shortcoming.

$$Adagrad\ update\ rule\ by\ \text{Ruder}: (H_{i,}\beta_i, \varepsilon, G_i)$$

G_i is a diagonal matrix of which each element is the sum of the squares of gradient up to time step i, $\odot$= *Matrix Multiplication between* G_i *and* $\nabla\partial(\alpha_i)$.

$$\alpha_{i+1} = \alpha_i - \frac{\beta_i}{\sqrt{G_i+\varepsilon}} \odot \nabla\partial(\alpha_i) \quad \text{- (3)}$$

D. AdaDelta

AdaDelta is another member of the family set of stochastic gradient descent algorithms and is more like a modified successor to Adagrad. Here we have the default learning rate completely eliminated from the update rule, hence there is no need to set its default value. Also, it dispenses adaptive techniques for hyperparameter tuning and is also visibly sturdy to noisy gradient details. It imposes limits on the aggregate past gradients by setting a fixed size.

E. Adaptive Moment Estimation (Adam)

Adaptive Moment Estimation or Adam is the most popular of all the optimizers. It's closely related to RMSprop and Adagrad. Known for its high efficiency, versatility and faster convergence adam uses the L2 norm or Euclidean norm for optimization. Like RMSprop, it involves the squared gradients for scaling the learning rate while it taking the benefit of momentum using gradient's moving average instead of the gradient itself like Stochastic Gradient Descent with momentum. The name Adam is derived from adaptive moment estimation, indicating that it computes individual learning rates for different parameters, it uses estimations of first and second moments of gradient to adapt the learning rate for each weight of the network.

$$Adam\ update\ rule\ by\ Kingma, D.P.\ and\ Ba, J\text{: } (H_i, \theta_i, \sigma_1, \sigma_2, \varepsilon)$$

$$u_0 = 1, m_0 = 0$$
$$m_{i+1} = \sigma_1 m_i + (1-\sigma_1)\nabla\,\partial(\alpha_i) u_{i+1} = \sigma_2 u_i + (1-\sigma_2)\nabla\,\partial(\alpha_i)^2$$

$$(\sigma = hyper\ parameters\ introduced\ by\ the\ author\ as\ \beta)$$

$$n_{i+1} = \frac{\sqrt{1-\sigma_2^{i+1}}}{1-\sigma_1^{i+1}}$$

$$\alpha_{i+1} = \alpha_i - \theta_i \frac{m_{i+1}}{\sqrt{u_{i+1}}+\varepsilon} n_{i+1} \qquad \text{- (4)}$$

F. Adamax

Adamax was introduced along with adam in the same paper. It may be called as a variant of Adam optimizer that uses infinity or max norm for optimization. Data which is traditionally noisy in terms of gradient updates (ex: dataset having multiple outliers) will make Adamax outdo Adam.

G. Nesterov-accelerated Adaptive Moment Estimation (Nadam)

Nadam is a combination of Adam and NAG (Nesterov Accelerated Gradient). It uses Nesterov to update the gradient one step ahead, so it gets its name Nesterov- accelerated Adaptive Moment Estimation. Its mostly used for noisy or high curvature gradients. The learning process is accelerated by summing up the exponential fall of the moving averages for the previous and present gradient.

$$Nadam\ update\ rule\ by\ Dozat, 2016\text{: } (H_i, \theta_i, \sigma_1, \sigma_2, \varepsilon)$$

$$u_0 = 1, m_0 = 0$$
$$m_{i+1} = \sigma_1 m_i + (1-\sigma_1)\nabla\,\partial(\alpha_i)$$
$$u_{i+1} = \sigma_2 u_i + (1-\sigma_2)\nabla\,\partial(\alpha_i)^2$$
$$n_{i+1} = \frac{\sqrt{1-\sigma_2^{i+1}}}{1-\sigma_1^{i+1}}$$

$$\alpha_{i+1} = \alpha_i - \theta_i \frac{\sigma_1 m_{i+1} + (1-\sigma_1)\nabla\,\partial(\alpha_i)}{\sqrt{u_{i+1}}+\varepsilon} n_{i+1} \qquad \text{- (5)}$$

4. Training and Validation Accuracy

The dataset used to train a model is known as the training set, whereas which is said to evaluate the model's performance is known as the validation set. I have used 70% of the dataset for training and the remaining 30% for validation in my work. Accuracy is one of the metrics to measure the performance

of our model. It is the percentage estimate of the correct classifications or predictions on the test data. When we apply our model on the training set, how accurately it's able to ascertain, predict or classify the output according to the complication defines the training accuracy of the model. Similarly, validation accuracy is the accuracy of the model on its validation dataset.

5. Learning Curves in Model Performance Evaluation

A learning curve of a model can be defined as a plot of a model's learning performance over time or understanding. Learning curves of model performance on the train and validation datasets can be used to evaluate whether it is an over, under or a well-fit model. It can be used to assess whether the train or validation datasets stand as a representative of the problem dominion or not. Thus, they can be used to measure the relative performance of different models to ascertain the best one graphically.

6. Datasets

For conducting a fair comparative analysis, four datasets were chosen. The first three: *Toxicity, Workshop, Masters,* was created of 200, 400 and 600 instances, respectively. The fourth one being the famous *Titanic* dataset having data of 800 passengers. The first dataset *Toxicity*, was devised based on the data available in "Status of Trace & Toxic Metals in Indian Rivers", Ministry of Jal Shakti Dept. of Water Resources released in August 2019. I employed the concentrations of Arsenic (As), Cadmium (Cd), Nickel (Ni), Iron (Fe), Lead (Pb) and Zinc (Zn) corresponding to *200* Indian Rivers as my matrix of features, while keeping Toxicity as the dependent variable, judging a river to be lowly (Low or 0) or highly (High or 1) toxic. Out of the whole dataset we specify 140 as training and 60 as testing/validation dataset by keeping the validation split attribute to be 0.3. The second dataset *Workshop*, included the data of *400* students of Department of Computer Science, Aligarh Muslim University, selected at random. I used it in discerning the probability of a student attending the recently held IoT Workshop. The parameters of assessment included age, gender, semester, enrolment to any club and hostler/ day scholar. The third is the *Masters* dataset, which contains data of *600* undergraduate students to understand their post-graduation pursuance trend from the same University by considering their residential details, gender, age, etc. The fourth and last being the standard *Titanic* dataset which assesses the aspects of passengers who boarded the deck and managed to survive or not. Out of the total 888 people on the list, *800* were chosen as per our requirement. The datasets will be addressed directly by these names throughout the course of this paper.

7. Optimizing the DNN Model

We build a simple Sequential Deep Neural Network with about six fully connected layers to classify our data. We fixed the following parameters, such as the loss function as 'binary crossentropy', activation of first five deep layers as 'relu' and 'sigmoid' as the activation of the output layer. Also, we randomly initialize the weights to numbers close to zero uniformly. Further, we train all the models for a thousand epochs with a batch size 8 for each optimization algorithm: SGD, RMSprop, Adagrad, Adadelta, Adam, Adamax and Nadam, taken one at a time. Finally, we analyze graphically, which one provides best training and validation accuracy for our all four datasets.

8. Experimentation and Analysis

A. *Stochastic Gradient Descent (SGD)*

The first optimizer we chose is Stochastic Gradient Descent (SGD). The accuracy plots exhibit the following trends: during the entire training period, both the training and validation accuracies remain almost unchanged that are: 83.57% and 80% for *toxicity*, 71.43% and 61.67% for *workshop*, 59.52% and 62.78% for *masters*, 59.46% and 65% for *titanic,* respectively as shown in fig. 1.

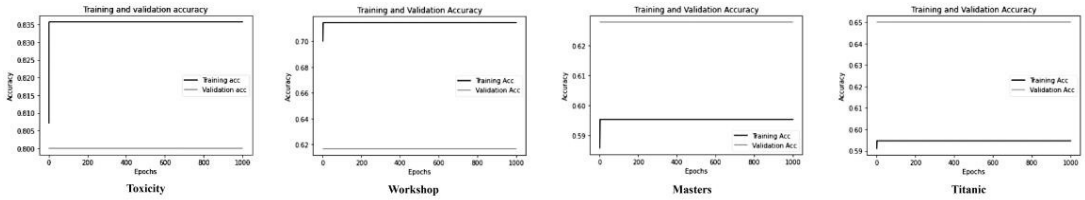

Fig. 1. Training and Validation Accuracies corresponding to SGD Algorithm.

B. *Root Mean Square Propagation (RMSprop)*

Now, we take Root Mean Square Propagation (RMSprop). The trends (refer to fig. 2) change rapidly across the entire training period roughly for all four datasets with particular emphasis on *toxicity,* where the training accuracy reaches from 81% to 92.14%. In contrast, validation goes from 80% to 85% in 1000 epochs. The remaining three go like, 71.43% and 61.67% for *workshop*, 89.05% and 80.56% for masters, 88.04% and 79.58% for *titanic*. We infer receiving a relatively good accuracy percentage for the models using this optimizer.

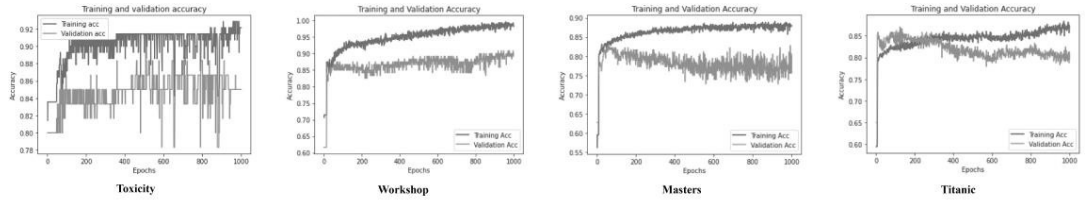

Fig. 2. Training and Validation Accuracies corresponding to RMSprop Algorithm.

C. *Adaptive Gradient Algorithm (Adagrad)*

Next, we take the Adaptive Gradient Algorithm (Adagrad) as shown in fig.3. Training accuracy is the only updating attribute that goes from 77% to 83.57%, while the validation accuracy 80% stands invariable in the first dataset *toxicity*. The *workshop* dataset shows deviation all along the training

process and finally settles at 87.50% and 83.33% accuracies. *Masters* has stagnant values of 59.52% and 62.78%. Finally, *titanic* shows an interesting trend movement resulting in 81.25% and 84.58%.

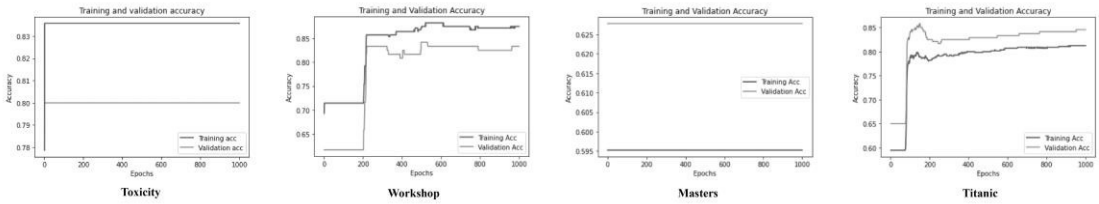

Fig. 3. Training and Validation Accuracies corresponding to Adagrad Algorithm.

D. Adadelta

We now test Adadelta, the successor of Adagrad also known for slower though better local minima. For *toxicity,* we find only the training accuracy growing and that just from 82% to 83.57% during the complete expanse of training. The validation accuracy remains 80% throughout. The same trend continues for the rest three datasets of, 71.43% and 61.67% for *workshop*, 59.52% and 62.78% for *masters*, 59.46% and 65% for *titanic* as shown in fig. 4.

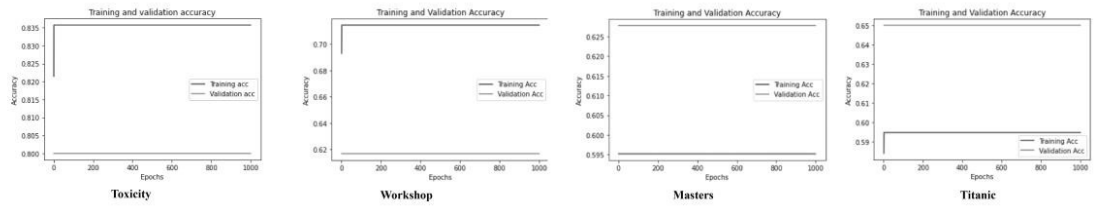

Fig. 4. Training and Validation Accuracies corresponding to Adadelta Algorithm.

E. Adaptive Moment Estimation (Adam)

Adaptive Moment Estimation (Adam) showed the most engrossing and fluctuating trends. Elaborating *toxicity*, the training accuracy started from 78% increased to a local minimum 92%, retreated to 80s, repeated this behavior in a zigzag fashion, finally at the end of 1000 epochs we obtained 92.86% training and 85% validation accuracies. Rest we have, 99.98% and 88.33% for *workshop*, 89.76% and 74.44% for *masters*, 89.11% and 78.75% for *titanic*. We observe in fig. 5 Adam provides higher accuracies than the previous optimum obtained from RMSprop in all the models.

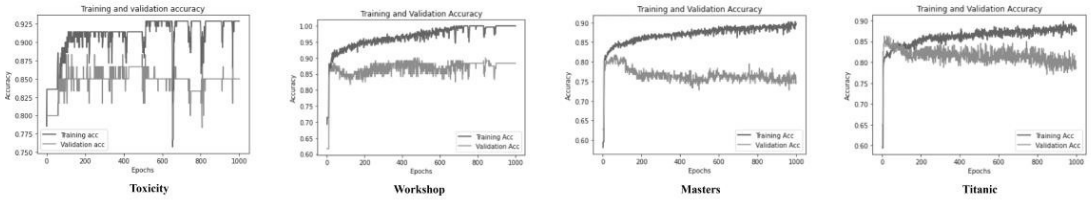

Fig. 5. Training and Validation Accuracies corresponding to Adam Algorithm.

F. Adamax

Adamax (refer to fig.6) concluded with both the training and validation accuracies remaining fixed at 83.57% and 80% respectively in *toxicity*, 71.42% and 61.67% in *workshop*, 59.46 and 65% in *titanic*. However, it demonstrated variability in *masters* with accuracies finally settling at 87.38% and 76.11%.

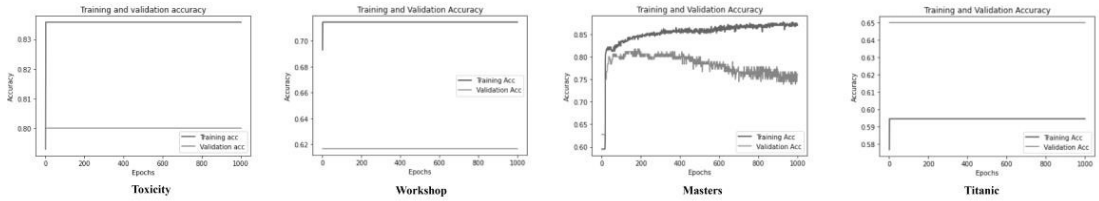

Fig. 6. Training and Validation Accuracies corresponding to Adamax Algorithm.

G. Nesterov-accelerated Adaptive Moment Estimation (Nadam)

Lastly, we use Nesterov-accelerated Adaptive Moment Estimation (Nadam). We obtain 83.57% and 80% for *toxicity*, 71.43% and 61.67% for *workshop*, 88.1% and 80% for *masters*, 87.14% and 81.67% for *titanic*, respectively as in fig.7.

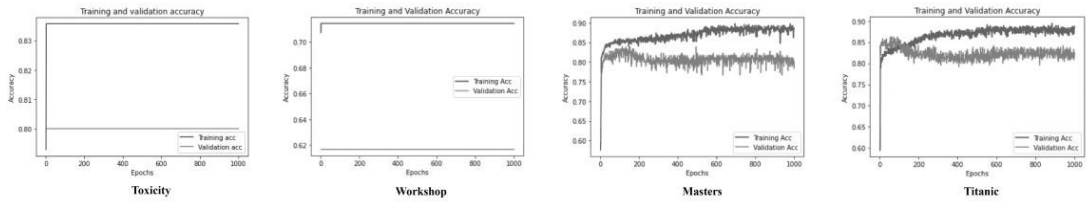

Fig. 7. Training and Validation Accuracies corresponding to Nadam Algorithm.

9. Results and Discussion

We discovered various trends in the above experimentation, and thus analyzed that Adam and RMSprop Algorithms provide optimum training and validation accuracy. From the first dataset *Toxicity*, adam stands as the best optimizer with 92.86% training and 85% validation accuracy, followed by rmsprop, even on a dataset comprising of as low as 200 real-life data samples. Nevertheless, rest of the optimizers settled on 83.57% training and 80% validation accuracy unanimously. For the second dataset *Workshop,* comprising of 400 instances, we observe sgd, rmsprop, nadam performing way poorly than expected. In contrast, adagrad performs unpredictably better, with adam being the foremost again. The dataset *Masters* showed rmsprop closely competing with adam and even performing better on the validation dataset followed by adamax. Finally, in dataset *Titanic* we discover, our four out of the seven optimizers trying to outshine the other by giving excellent outcomes, which is reasonable as the dataset has grown to 800 data instances.

Here Adam succeeds accompanied by the fair playing rmsprop, nadam, adagrad algorithms. The following fig. 8 contains a graphical comparison of all the optimization algorithms employed in each dataset based on the aforementioned parameters.

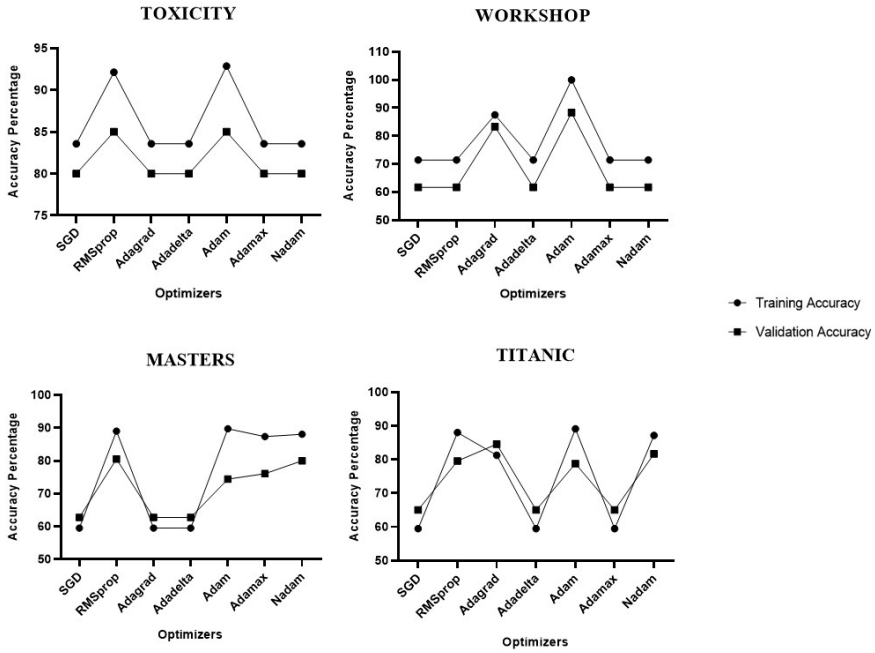

Fig. 8. Accuracy Analysis Plots: Graphical comparison of all the Optimization Algorithms employed, on the basis of their Training and Validation Accuracies.

10. Conclusion

Studying the graphs and analyzing the trends we conclude, 'Adam' Optimization Algorithm works best with all the four Deep Neural Network Models, under all circumstances and is thus practically able to work with any classification model resulting in the best accuracy. While RMSprop was found to be a fine choice in three out of the four datasets, it proved to be the second winner. Also from our experimentation, we noticed Optimizers, SGD and Adadelta, fail to provide satisfactory results in any of the four models. Hence, they are the least recommended optimization algorithms for a supervised deep learning model.

Ultimately, we conclude that our work stands in-line with the state of the art findings. Also we were able to reckon the finest and the least economical optimizers from our work.

11. References

Types of Optimization Algorithms used in Neural Networks https://towardsdatascience.com/types-of-optimization-algorithms-used-in-neural-networks-and-ways-to-optimize-gradient-95ae5d39529f

A Tour of Machine Learning Algorithms by Jason Brownlee on August 12 2019 in Machine Learning Algorithms https://machinelearningmastery.com/a-tour-of-machine-learning-algorithms/

Y Guo, Y Liu, A Oerlemans, S Lao, S Wu, MS Lew - Deep learning for visual understanding: A review. Neurocomputing, 2016 – Elsevier

A Shrestha, A Mahmood - Review of deep learning algorithms and architectures, IEEE Access, 2019 - ieeexplore.ieee.org

Xiaohan Ding, Guiguang Ding, Xiangxin Zhou, Yuchen Guo, Jungong Han, Ji Liu- Global Sparse Momentum SGD for Pruning Very Deep Neural Networks

Dept. of Water Resources, River Development and Ganga Rejuvenation; River Data Compilation-2 Directorate August, 2019 Central Water Commission, Ministry of Jal Shakti, New Delhi, India- http://cwc.gov.in/

S. Vani and T. V. M. Rao, An Experimental Approach towards the Performance Assessment of Various Optimizers on Convolutional Neural Network, 2019 3rd International Conference on Trends in Electronics and Informatics (ICOEI), Tirunelveli, India, 2019, pp. 331-336, doi: 10.1109/ICOEI.2019.8862686.

Timothy Dozat- Incorporating Nesterov Momentum into Adam

Robbins, H. and Monro, S. A stochastic approximation method. The Annals of Mathematical Statistics, 22(3): 400–407, 1951.

Tieleman, T. and Hinton, G. Lecture 6.5-RMSProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural networks for machine learning, 4(2):26–31, 2012.

Dozat, T. Incorporating Nesterov momentum into Adam. In ICLR Workshops, 2016.

Dami Choi, Christopher J. Shallue, Zachary Nado, Jaehoon Lee, Chris J. Maddison, George E. Dahl. On Empirical Comparisons of Optimizers for Deep Learning, 2020.

Kingma, D. P. and Ba, J. Adam: a method for stochastic optimization. In ICLR, 2015.

Dean, J., Corrado, G. S., Monga, R., Chen, K., Devin, M., Le, Q. V, Ng, A. Y. (2012). Large Scale Distributed Deep Networks. NIPS 2012: Neural Information Processing Systems, 1–11.

Zeiler, M. D. (2012). ADADELTA: An Adaptive Learning Rate Method. Retrieved from http://arxiv.org/abs/1212.5701

Wilson, A. C., Roelofs, R., Stern, M., Srebro, N., and Recht, B. The marginal value of adaptive gradient methods in machine learning. In Advances in Neural Information Processing Systems 30, pp. 4148–4158. Curran Associates, Inc., 2017.

Sebastian Ruder: An overview of gradient descent optimization algorithms

How to use Learning Curves to Diagnose Machine Learning Model Performance by Jason Brownlee on February 27, 2019 in Deep Learning Performance

Guoji Xu, Qin Chen, Jianhua Chen. "Prediction of Solitary Wave Forces on Coastal Bridge Decks Using Artificial Neural Networks", Journal of Bridge Engineering, 2018

"Image Analysis and Recognition", Springer Science and Business Media LLC, 2019

Amirhessam Tahmassebi, Amir H. Gandomi, Simon Fong, Anke Meyer-Baese, Simon Y. Foo. "Multi-stage optimization of a deep model: A case study on ground motion modeling", PLOS ONE, 2018

E. M. Dogo, O. J. Afolabi, N. I. Nwulu, B. Twala, C. O. Aigbavboa. "A Comparative Analysis of Gradient Descent-Based Optimization Algorithms on Convolutional Neural Networks", 2018 International Conference on Computational Techniques, Electronics and Mechanical Systems (CTEMS), 2018