ISBN: 978-3-95450-223-3 ISSN: 2673-7000 doi:10.18429/JACoW-NAPAC2019-WEPLE07

TRANSFER MATRIX CLASSIFICATION WITH ARTIFICIAL NEURAL NETWORK[*]

Y. Sun[†], ANL, Argonne, IL 60439, USA

Abstract

Standard neural network algorithms are developed for classification and regression applications. In this paper, some details of the neural network algorithms are presented on several optimization process. Artificial neural network is trained to classify multi-class transfer matrix of different types of particle accelerator components. It is shown that with a fully-connected feedforward neural network, it is possible to get high accuracy of 99% on training data, validation data and test data. Some hyperparameters are scanned for better performance. Different stochastic gradient descent algorithms are also benchmarked.

INTRODUCTION

Deep learning with artificial neural networks (ANN) has been successfully applied in many areas, such as image recognitions, natural language processing, internet advertising and automatic driving. An artificial neural network code has been developed in the programming language Python [1]. The modules/functions of this code include the following: initialization of parameters **W** and **b**; Forward and backward propagation with several popular activation functions; cost functions (such as mean square error, softmax, cross-entropy); optimization algorithms [2] (gradient descent and stochastic gradient descent algorithms, including *adam* [3]); variational autoencoders and so on.

In the following sections, the artificial neural network training are discussed and optimized, to classify multi-class transfer matrix of different types of particle accelerator components.

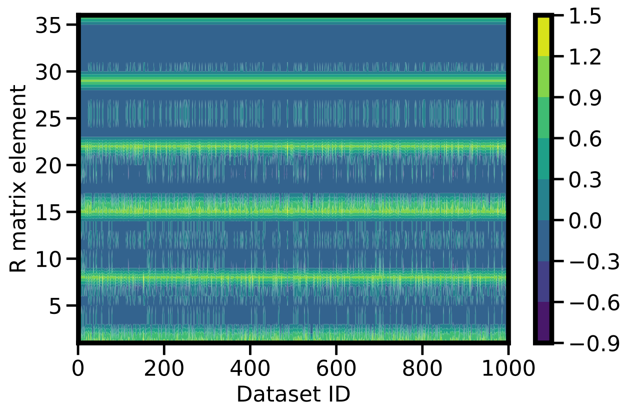

Figure 1: Input training data, size of 36 by 1000.

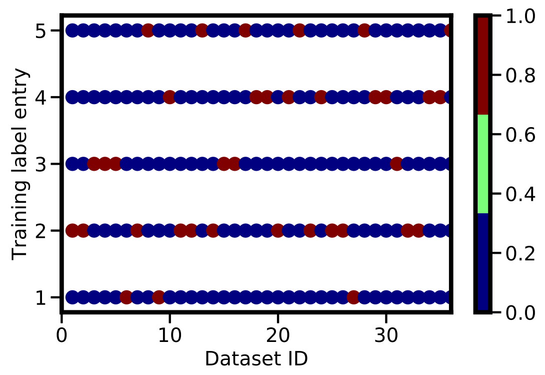

Figure 2: Output training data (label), size of 5 by 1000 (showing first 35).

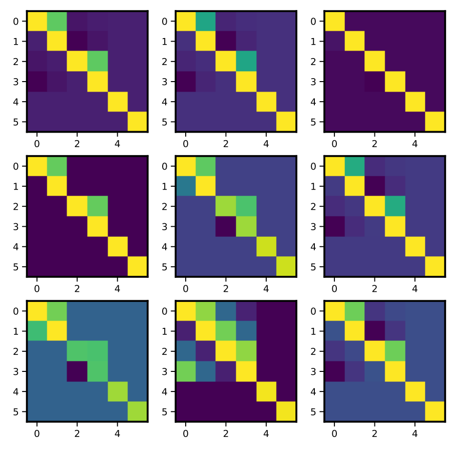

Figure 3: Selected samples of input training data, in the format of 6 by 6 matrix.

TRANSFER MATRIX CLASSIFICATION

In a particle accelerator, there are several common components which are listed below: quadrupole magnets, skew quadrupole magnets, transverse gradient dipoles, dipole fringe fields, and drift spaces. A neural network model is established to classify these 5 types of components. The linear transfer matrix of a quadrupole magnet is shown below [4].

$$R_Q = \begin{pmatrix} \cos kL & \frac{\sin kL}{k} & 0 & 0 & 0 & 0 \\ -k\sin kL & \cos kL & 0 & 0 & 0 & 0 \\ 0 & 0 & \cosh kL & \frac{\sinh kL}{k} & 0 & 0 \\ 0 & 0 & k\sinh kL & \cosh kL & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \tag{1}$$

The transfer matrix of the other four types of components ($R_{skewquad}$, R_{dipole}, R_{fringe}, R_{drift}) are not shown here.

[*] Work supported by the U.S. Department of Energy, Office of Science, Office of Basic Energy Sciences, under Contract No. DE-AC02-06CH11357.
[†] yisun@anl.gov

ISBN: 978-3-95450-223-3 ISSN: 2673-7000 doi:10.18429/JACoW-NAPAC2019-WEPLE07

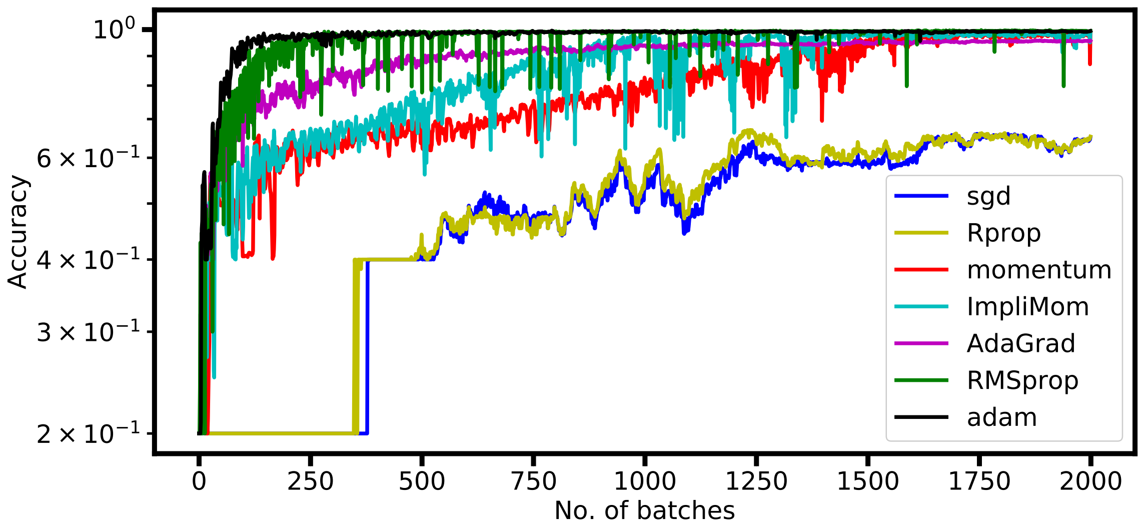

Figure 4: Benchmark of different stochastic gradient descent optimization algorithms on training data accuracy.

This is a 5-class classification problems, where the training input (first layer of ANN) and training label (last layer of ANN) datasets are shown in Figs. 1 and 2. In Fig. 1, color code is value of transfer matrix entry. It is noted that the 6 by 6 transfer matrix is reshaped into a 1 by 36 vector. There are 200 samples for each class and 1000 samples in total. Several samples of input training data are shown in Fig. 3, in the format of 6 by 6 matrix. It is observed that most of the diagonal entries are close to 1, and there are a lot of zero entries (sparse matrix).

As mentioned in the above sections, several different stochastic gradient descent optimization algorithms [2] are implemented in this neural network code, including stochastic gradient descent (*sgd*), resilient backpropagation (*Rprop*), momentum and implicit momentum, and adaptive methods with running average of first and second moments of gradient [2]. Figure 4 shows the benchmark of these different stochastic gradient descent optimization algorithms on training data accuracy. It is observed that method *adam* seems to have overall best performance. In the following sections *adam* is the default optimization method.

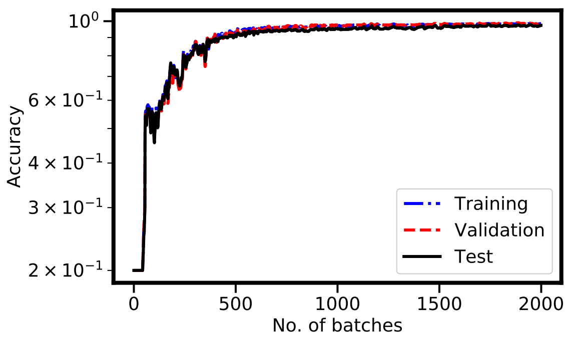

Figure 5: Accuracy as a function of number of batches, for training/validation/test datasets.

For validation and test datasets, they each are composed of 500 samples where there are 100 samples for each class. Using a neural network with three hidden layers and a small number of neurons on each layer, it is possible to achieve high accuracy for training/validation/test datasets. As shown in Fig. 5, the accuracy is around 99% after training with 1000-2000 mini batches.

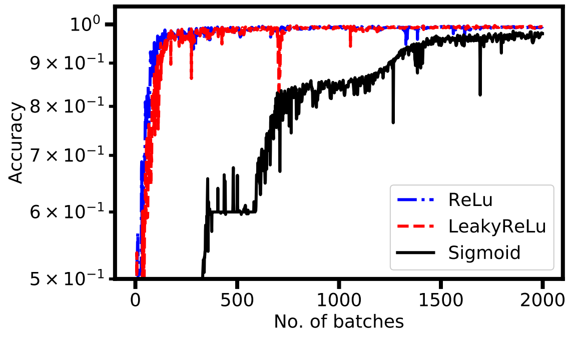

Figure 6: Comparison of different activations in hidden layers.

The default activation function for the hidden layers is rectified linear unit function (Relu). It is also compared with the other two activations, a sigmoid function, and the leaky Relu function where there is some residual gradients for the negative part. It is observed from Fig. 6 that Relu and leaky Relu share similar performance, while sigmoid function converges slower due to the vanishing gradient issue.

SCAN ON HYPERPARAMETERS

In this section, scan results on some hyperparameters of the neural network model are discussed. By default the learning rate is 0.001 and the mini batch size is 32.

A one dimentional scan on training data size shows that data size of 300 seems to be enough to achieve high accura-

ISBN: 978-3-95450-223-3 ISSN: 2673-7000 doi:10.18429/JACoW-NAPAC2019-WEPLE07

cies for all three datasets (training, validation and test data). The best accuracies found are 0.997, 0.998, 0.986 for training, validation and test datasets. It makes sense for the test dataset to have the lowest accuracy, as it is not involved in the training process. The results are shown in Fig. 7.

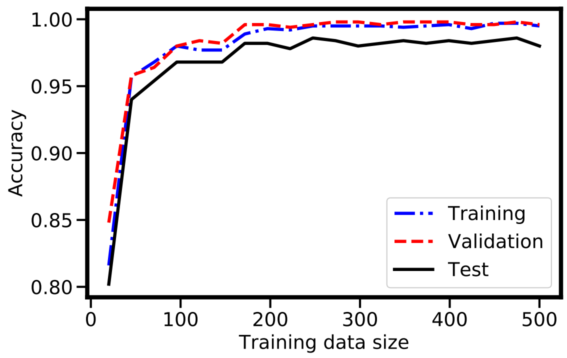

Figure 7: One dimentional scan on training data size, showing the accuracies for training, validation and test datasets.

A two dimentional scan is performed on grids of learning rates and number of total mini batches. The number of total mini batches is from 100 to 3000, while the learning rate is from 1×10^{-4} to 0.035. The figure of merit here is the average accuracy. As shown in Fig. 8, for this specific classification problem, the best learning rate is from 0.02 to 0.05, where the neural network learning converges in as few as 200 batches. When the learning rate is too large (larger than 0.05), the optimization is jumping around the optimum and it is difficult to converge. On the other hand, if the learning rate is small, it takes much more number of batches to move to the optimum point.

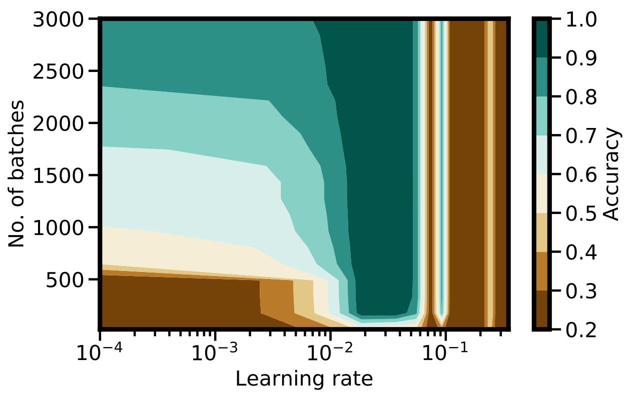

Figure 8: Two dimentional scan on learning rate and number of total mini batches.

Another two dimentional scan is performed on grids of mini batch size and initialization random seed number. The mini batch size is scanned from 1 to 512, while the initialization random seed number is from 0 to 20. It is found that the neural network performance is rarely dependent on the initialization random seed number. Also a mini batch size larger than 10 is preferred. See Fig. 9.

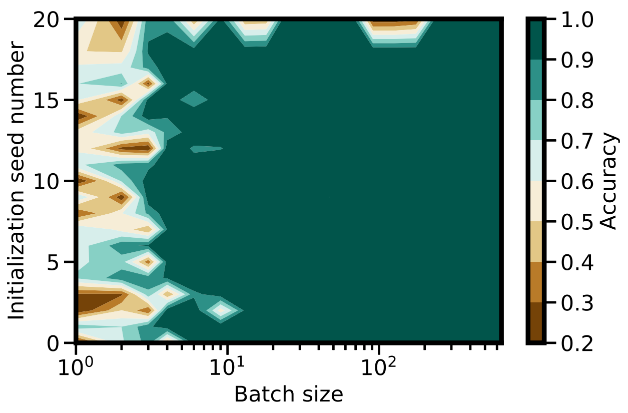

Figure 9: Two dimentional scan on mini batch size and initialization random seed number.

CONCLUSIONS

A basic neural network framework has been developed in Python. It is successfully applied to a 5-class classification of particle accelerator components transfer matrix, using a fully connected feedforward network. The best accuracy achieved is 99% for training, validation and test datasets. Different stochastic gradient descent optimization algorithms are benchmarked on this application, and it is confirmed that *adam* seems to be most robust. Several hyperparameters are scanned to improve the performance, such as learning rate, number of epochs, mini batch size and initialization random seed number.

ACKNOWLEDGMENT

The authors thank https://www.coursera.org/, and https://www.udemy.com/ for providing free online courses of neural networks.

REFERENCES

[1] Python Software Foundation, "Python Language Reference", `http://www.python.org`

[2] "Stochastic gradient descent algorithms", `https://en.wikipedia.org/wiki/StochasticGradientDescent`

[3] D. Kingma *et al.*, "Adam: A method for stochastic optimization", `arXiv:1412.6980`, 2014.

[4] *Handbook of Accelerator Physics and Engineering*, A. Chao and M. Tigner, Ed. Singapore, Singapore: World Scientific, 1999.