Deep Multilayered Models for Classification Based on Stochastic Optimization

Dominic Klusek
Computer Science
College of Staten Island
Staten Island, New York
klusekdominic@gmail.com

***Abstract*—Development of efficient neural network classifier topology involves preprocessing of training data to reduce noise and discrepancies, utilizing an abstract number of hidden layers, choice of correct activation functions etc. The goal of this research is to study the differences of training Deep Learning Multilayer Perceptron (DLMP) with backpropagation when using stochastic gradient-based optimization methods as stochastic gradient descent and Adam for classification of multiclass real datasets with big dimensions. K-fold cross validation is used to split training and testing data in order to compare the performance of each model with the respective set of parameters. Experiments shown here include MNIST, Poker, and Shuttle datasets, and loss and accuracy metrics were compiled in a table for comparison.**

Keywords—neural network, multilayered perceptron, deep learning neural networks, gradient based learning, activation functions, K-fold cross validation

I. Introduction

Neural Networks (NNs) are systems patterned after the operation of neurons in the human brain. There are two major categories of NNs: shallow networks and Deep Learning Neural Networks (DLNNs). While shallow NNs usually have a single hidden layer between the input and output layer of the network, DLNNs have multiple hidden layers between the input and output layers of the network and different approaches to training. Hidden layers of DLNNs create abstractions with different depth which are then used to learn the desired generalized representation. The latter allows them to build more accurate representation in learning massive datasets. The attractiveness of DLNNs comes from their flexibility to incorporate prior knowledge by modifying connection weights between layers and their capabilities to solve classification and regression problems when dealing with huge datasets [1].

A significant amount of research in the last 10 years has been focused on improving the theoretical understanding of NNs with Deep Learning (DL), the benefits of unsupervised training on raw data as well as on combination of supervised and unsupervised training. DLMLP topologies require varying the number of hidden layers and their neurons, the development of specific activation and loss functions, batch processing, normalization, and implementation of different gradient methods. Software packages as Matlab, Python and Python libraries (Keras, Theano, TensorFlow, Cognitive Toolkit (CNTK)) were developed to allow researchers to focus their efforts on main issues as DLNNs topology, choice of parameters and learning approaches.

Research partially funded by CUNY College of Staten Island Mini Technology Grant as well as CUNY Undergraduate Research Stipend. Thank you to my mentor Dr. N. Gueorguieva, CUNY College of Staten Island for her assistance in conducting this research.

II. Multilayer Perceptron and Training

A. Multilayer Perceptron

Multilayered Perceptron (MLP) is a NN composed of an input layer, an output layer, and an arbitrary number of hidden layers in between. Layers include an arbitrary number of neurons each of which takes a weighted sum of processed data from a lower layer, applies an activation function to the summation result, and then brings the outcome to the next layer.

Weights and biases applied to connections between hidden layers affect the impact of individual features on the result of the NN [1]. Choosing an optimal DLMLP topology assumes an appropriate choice of hidden layers number, types of their activation functions and training algorithm, as well as optional regularizer to apply penalties on layer parameters and activities during training [2].

B. Deep Learning with Feedforward Neural Networks

Feedforward NNs are unidirectional NN systems which feed data from the input layer, then to sequential hidden layers before finally outputting data through the output layer. Nevertheless that Feed-Forward NN with DL propose a new initialization strategy: use a series of single layer networks which do not suffer from vanishing/exploding gradients in order to find the initial parameters for a deep MLP, they lack the efficiency of DLMLP with backpropagation. Because of this backpropagation methods were developed to modify weights and biases based on training data and an error function [3].

When backpropagation is utilized into the topology of DLMLP, the classification performance for real datasets significantly improves.

Modular representation of DLMLP with the two passes (forward and backward) are shown in Fig. 1 and Fig. 2, where each module is an object/function $\boldsymbol{a} = h(\boldsymbol{x}, \boldsymbol{\theta})$ which receives as an argument an input $\boldsymbol{x}$, contains trainable

parameters ($\boldsymbol{\theta}$) and returns the output $\boldsymbol{a}$ based on the respective activation function $h(\boldsymbol{x},\boldsymbol{\theta})$.

C. Activation Functions

Activation functions are one of the main parameters of DLNNs that allow them to adapt to nonlinear problems. They emulate the charging/discharging of neurons in the brain and allow for feature selection. When it comes to classification problems the preferable activation functions for the output layer are Sigmoid and Softmax; Tanh is not suggested as much due to the problem of gradient saturation. For hidden layers Relu and its different modifications are widely used as they are computationally cheaper when compared to other nonlinear activation functions. These activation functions add computational complexity to DLNNs. However, they also add flexibility and generalization which makes them effective for solving nonlinear problems.

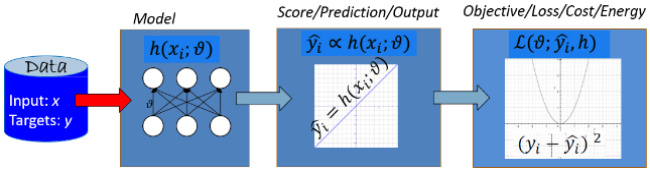

Fig.1 Modular representation of DLMLP forward pass

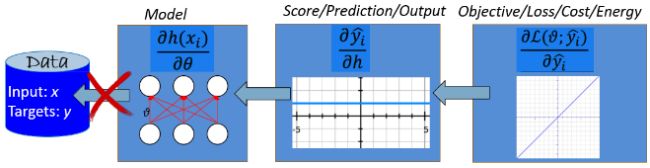

Fig. 2. Modular representation of DLMLP backward pass

D. Backpropagation and Learning

Initial values for weights and biases must be optimized to minimize overall error for classification. To minimize error, stochastic gradient-based methods are utilized to modify weights and biases in the direction of the gradient.

SGD without Nesterov momentum optimization algorithm calculates the gradient w.r.t. each of the variables of the loss function, and then utilizes the calculated error to modify each system of weights by utilizing the gradient of the error function and updating weights and biases based on equations (1) and (2).

$$\boldsymbol{w} = \boldsymbol{w} - \eta \nabla \boldsymbol{L}_i(\boldsymbol{b}) \tag{1}$$

$$\boldsymbol{b} = \boldsymbol{b} - \eta \nabla \boldsymbol{L}_i(\boldsymbol{b}) \tag{2}$$

where $\boldsymbol{w}$ and $\boldsymbol{b}$ are bias vectors, η is the learning rate, and $\boldsymbol{L}_i$ is the loss function at iteration i. Controlling the amount of change to weights and biases is done by modifying the learning rate. Too large value of a learning rate could result in never reaching the true minimum error, and too small of a learning rate could result in longer training times.

Adam optimization algorithm is another stochastic gradient based optimization method that computes individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients [4] expressed with Equations (3-5).

$$\boldsymbol{m}_t \leftarrow \beta_1 \cdot \boldsymbol{m}_{t-1} + (1-\beta_1) \cdot \boldsymbol{g}_t^1 \tag{3}$$

$$\boldsymbol{v}_t \leftarrow \beta_2 \cdot \boldsymbol{v}_{t-1} + (1-\beta_2) \cdot \boldsymbol{g}_t^2 \tag{4}$$

$$\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \alpha \cdot \boldsymbol{m} / (\sqrt{\boldsymbol{v}_t} + \varepsilon) \tag{5}$$

α is the step size, β_1 and β_2 are exponential decay rates for the moment estimates, $\boldsymbol{m}$ is the first moment estimate at step t, $\boldsymbol{v}$ is the second moment estimate at step t, $\boldsymbol{\theta}$ is the vector of parameters, and $\boldsymbol{g}$ is the gradient w.r.t. stochastic objective at step t. Decay rates β_1 and β_2 are values between [0, 1] where optimal recommended values for β_1 and β_2 should be close to 1 [5].

Computation of first and second moment are done using equations (3) and (4), afterwards bias values are corrected for these estimates (not shown). Afterwards the update rule in equation (5) is utilized to update DLMLP parameters. Adaptive learning rates for different parameters from first and second moment estimates reduce noise of data in datasets.

III. Validation of Experiment Results

A. Validation after Training

After training a validation step should be performed on data that was not used during training. The validation step gives the final accuracy of the DLMLP after training and is imperative to prove its effectiveness after the training. However, performing only a singular test can lead to an unmitigated bias that could occur due to the order of data in the dataset.

B. K-Fold Cross Validation

K-fold cross validation is a method to remove unmitigated bias by separating a dataset into K splits. $K-1$ splits are utilized during training, and the remaining split is used in the validation step. This is performed K times and then the average performance metrics (final accuracy, error, and accuracy during training) give the overall performance of DLMLPs, thus, neutralizing a large portion of unmitigated bias and giving the general performance on data such as in the current dataset.

IV. Experimentation

A. Dataset Information

Datasets are standard datasets from the University of Irvine (UCI) [6] machine learning repository, and LibSVM dataset repository [7]. The details of the three datasets presented in this research MNIST, Shuttle, and Poker datasets are given in Table I below.

Table I. *Table of dataset information*

Dataset	Number of Classes	Number of Examples	Number of Features
MNIST	10	60,000	780
Shuttle	7	43,500	9
Poker	10	25,000	10

MNIST dataset contains 60,000 patterns where each one has of 780 attributes. The latter are decimal numbers which represent pixels in an 28x28 image. Class labels consist of handwritten digits which are coded as follows: 0: 0, 1: 1, 2:2, 3:3, 4:4, 5:5, 6: 6, 7: 7, 8: 8, 9:9.

Shuttle dataset contains 43,500 training samples with 9 attributes all of which are numerical representations of readings from instrument readings. Class labels consist of system states which are coded as follows: 1 Rad Flow, 2 Fpv Close, 3 Fpv Open, 4 High, 5 Bypass, 6 Bpv Close7 Bpv Open.

Poker dataset contains 25,000 poker hand training patterns, where each hand consists of 5 cards, and each card is described using it suit and number. Class labels consist of various hand types and are coded as follows: 0: Nothing in hand; not a recognized poker hand, 1: One pair; one pair of equal ranks, 2: Two pairs; two pairs of equal ranks, 3: Three of a kind; three equal ranks, 4: Straight; five cards, sequentially ranked with no gaps, 5: Flush; five cards with the same suit, 6: Full house; pair + different rank three of a kind, 7: Four of a kind; four equal ranks, 8: Straight flush; straight + flush, and 9: Royal flush; {Ace, King, Queen, Jack, Ten} + flush.

B. Initialization and Activation Functions

Weights were initialized using the Glorot Normal algorithm and bias terms were initialized to zero for all datasets.

For experiments we focus on Relu, Sigmoid, Softmax, and Tanh activation functions.

C. Tested Loss Functions

For the MNIST dataset class labels were converted to binary matrices, and categorical cross entropy loss function was implemented, because utilizing binary crossentropy resulted in poor variance, and no noticeable change was observed in terms of accuracy or loss ratings. For the Poker and Shuttle datasets, binary crossentropy loss function was utilized because it was sufficient for testing of different DLMLP topologies and resulted in distinct changes in accuracy and loss ratings.

D. Experimental Details

Experiments were designed to study the effect of using different number of hidden layers, neurons, batch size as well as to implement different sets of activation functions, and stochastic optimizers.

The number of hidden layers varies between 1-3 hidden layers; batch size takes values of 128, 1024, and the entire length of the training set to observe the impact it had on training time and convergence. The average performance frame of the current DLMLP topology with backpropagation was obtained for each test case by utilizing 10-fold cross validation. SGD and Adam were tested to compare the performance of the original stochastic gradient optimizer to a newer predecessor. Parameters for SGD were set to α = 0.01, decay = 0.0, and momentum = 0.0. The values of Adam parameters were established to α = 0.001, β_1 = 0.9, β_2 = 0.999, $\varepsilon = 1^{-7}$, and decay = 0.0.

E. Experimental Results

All test results and respective parameters are stored in an output file to allow further analysis. Partial display of the results is shown in Table II.

Table II. *Average Accuracy according to output layer activation function*

A. For batch size 128 utilizing SGD				
Dataset	Relu	Sigmoid	Softmax	Tanh
MNIST	26.27 %	92.80 %	93.02 %	11.15 %
Poker	30.47 %	42.60 %	42.37 %	40.99 %
Shuttle	12.08 %	78.25 %	78.25 %	70.39 %
B. For batch size 1024 utilizing SGD				
Dataset	Relu	Sigmoid	Softmax	Tanh
MNIST	42.80 %	84.14 %	86.62 %	25.5 %
Poker	29.37 %	42.61 %	42.37 %	41.28 %
Shuttle	17.04 %	78.25 %	78.25 %	78.25 %
C. For batch size full test data utilizing SGD				
Dataset	Relu	Sigmoid	Softmax	Tanh
MNIST	27.02 %	11.54 %	14.72 %	12.69 %
Poker	36.43 %	44.27 %	42.37 %	40.06 %
Shuttle	26.81 %	75.69 %	78.25 %	78.25 %
D. For batch size 128 utilizing Adam				
Dataset	Relu	Sigmoid	Softmax	Tanh
MNIST	83.87%	97.72%	97.81%	12.97%
Poker	31.61%	42.61%	42.38%	41.00%
Shuttle	13.28%	78.25%	78.25%	67.79%
E. For batch size 1024 utilizing Adam				
Dataset	Relu	Sigmoid	Softmax	Tanh
MNIST	84.34%	97.16%	97.33%	11.42%
Poker	29.07%	42.62%	42.38%	41.14%
Shuttle	17.89%	78.25%	78.25%	67.79%
F. For batch size full test data utilizing Adam				
Dataset	Relu	Sigmoid	Softmax	Tanh
MNIST	43.38%	75.19%	80.74%	24.34%
Poker	35.47%	44.28%	42.38%	39.86%
Shuttle	23.99%	75.69%	78.25%	67.79%

V. CONCLUSION

The backpropagation procedure is not limited to DLNNs with feed-forward flows. It can be applied to networks of modules with any topology, as long as the connection graph is acyclic, i.e. no loops. DLMLP is regarded as a backbone of DLNN topologies when it comes to classification due to its flexibility as well as potentials for parallel computations. Like all NNs with DL, DLMLP with backpropagation has many hyper parameters that are modified during the training

in order to obtain an optimal performance metric and solution.

Analysis of results shown in Table 2 and Figures 3, 4, 5 allow us to make the following conclusions:

a) Sigmoid and Softmax are the optimal output layer activation functions for single and multi-binary output in solving classification problems.

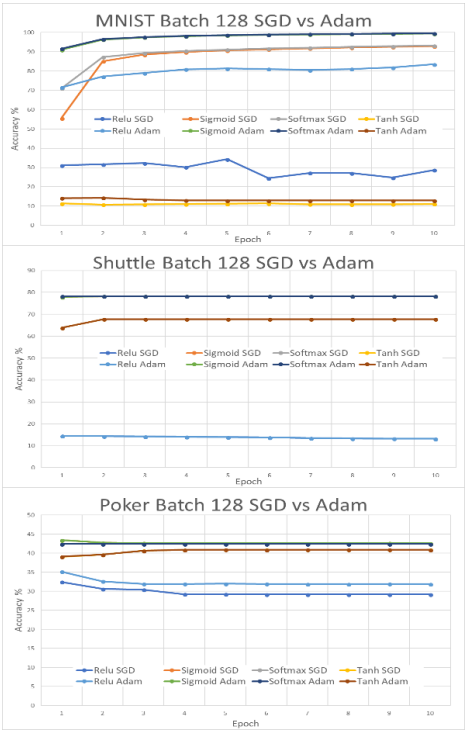

Fig 3. Graphical representation of results for batch size 128

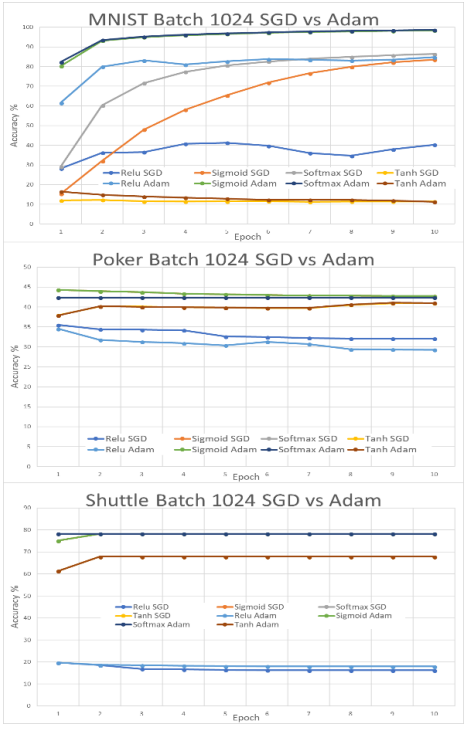

Fig 4. Graphical representation of results for batch size 1024

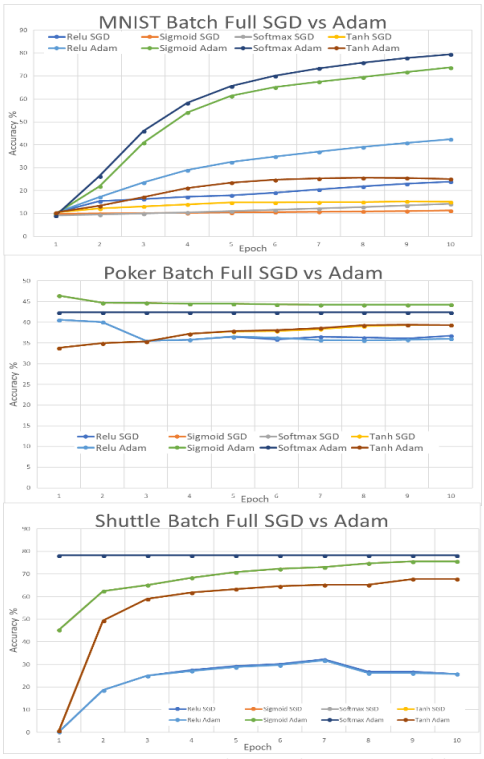

Fig 5. Graphical representation of results for batch size of full test data

b) In terms of stochastic optimization algorithms, the Adam demonstrates better peformance when compared to SGD without Nesterov momentum due to the variable learning rates.

c) Batch size has a large effect on the training of DLMLP in general, and though it is possible to reach similar accuracies with larger batch sizes, it would require a larger amount of training.

d) Initial choice of hyper parameters is ultimately up to the programmer, but the torough testing allows the finding of the optimal set.

References

[1] Yann LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11):2278-2324, November 1998.

[2] I. Goodfellow, Y. Bengio, and A. Courville, Deep learning. Cambridge, MA: MIT Press, 2017.

[3] D. Gupta and Dishashree, “Fundamentals of Deep Learning - Activation Functions and their use,” Analytics Vidhya, 23-Oct-2017. https://www.analyticsvidhya.com/blog/2017/10/fundamentals-deep-learning-activation-functions-when-to-use-them/.

[4] Ramchoun, Hassan, Mohammed Amine, Janati Idrissi, Youssef Ghanou, and Mohamed Ettaouil. "Multilayer Perceptron: Architecture Optimization and Training." International Journal of Interactive Multimedia and Artificial Intelligence 4, no. 1 (2016): 26. doi:10.9781/ijimai.2016.415.

[5] D. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization", *CoRR*, vol. 141269809, 2014.

[6] [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.

[7] https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/.