iDeepLe: deep learning in a flash

Conference Paper · May 2018

DOI: 10.1117/12.2304418

1 author:

Amirhessam Tahmassebi
Florida State University
26 PUBLICATIONS 52 CITATIONS

SEE PROFILE

iDeepLe: Deep Learning in a Flash

Amirhessam Tahmassebi

Department of Scientific Computing, Florida State University, Tallahassee, Florida, USA

ABSTRACT

Emerging as one of the most contemporary machine learning techniques, deep learning has shown success in areas such as image classification, speech recognition, and even playing games through the use of hierarchical architecture which includes many layers of non-linear information. In this paper, a powerful deep learning pipeline, intelligent deep learning (iDeepLe) is proposed for both regression and classification tasks. iDeepLe is written in Python with the help of various API libraries such as Keras, TensorFlow, and Scikit-Learn. The core idea of the pipeline is inspired by the sequential modeling with considering numerous layers of neurons to build the deep architecture. Each layer in the sequential deep model can perform independently as a module with minimum finitudes and does not limit the performance of the other layers. iDeepLe has the ability of employing grid search, random search, and Bayesian optimization to tune the most significant predictor input variables and hyper-parameters in the deep model via adaptive learning rate optimization algorithms for both accuracy and complexity, while simultaneously solving the unknown parameters of the regression or the classification model. The parallel pipeline of iDeepLe has the capacity to handle big data problems using Apache Spark, Apache Arrow, High Performance Computing (HPC) and GPU-enabled machines as well. In this paper, to show the importance of the optimization in deep learning, an exhaustive study of the impact of hyper-parameters in a simple and a deep model using optimization algorithms with adaptive learning rate was carried out.

Keywords: Deep Learning, Deep Neural Networks, Machine Learning, Classification, Regression, Optimization.

1. INTRODUCTION

Creation of the first computational model based on artificial neural networks (ANN) with application to artificial intelligence (AI) might date back to a model built in 1943, which was inspired from biology to simulate how the brain works.[1] Neurons in the perceptual system represent features of the sensory input. The brain has a deep architecture and learns to extract many layers of features, where features in one layer represent combinations of simpler features in the layer below and so on. This is referred to as feature hierarchy. Based on this idea, several architectures for the topology of the networks such as layers of neurons with fully/sparse connected hidden layers were proposed.[2] The essential questions to ask are: How can the weights that connect the neurons in each layer be adjusted? How many parameters should we find and how much data is necessary to train or test the network? What would be the most efficient strategy for optimizing the hyper-parameters? What optimization algorithm can speed up the optimization process of the hyper-parameters? What parallel architecture or cluster-computing framework can be employed along with the deep neural networks to handle big data problems? Before 2006 there was lack of computational equipment to train or test deep neural networks and shallow neural networks were the best choice to be employed as a model. In 2006 Hinton et al.[3] proposed Deep Belief Networks (DBN) based on Restricted Boltzmann Machines (RBM). Additionally, in 2007 Bengio et al.[4] created a greedy layer-wise training of deep neural networks. This would be the beginning of the revolution in deep neural networks and a breakthrough in learning deep architectures using different perspectives of optimization. Deep learning models involve optimization from the first day of appearance of neural networks. For example, Vapnik and Cortes[5] proposed Support Vector Machines (SVM) just by employing a smart optimization algorithm in implementing the single layer perceptron. SVM along with different kernel functions such as linear, sigmoid, radial basis function, and polynomials is still one of the best methods implemented in machine learning.[5] This highlights

E-mail: atahmassebi@fsu.edu
URL: www.amirhessam.com, www.iDeepLe.com

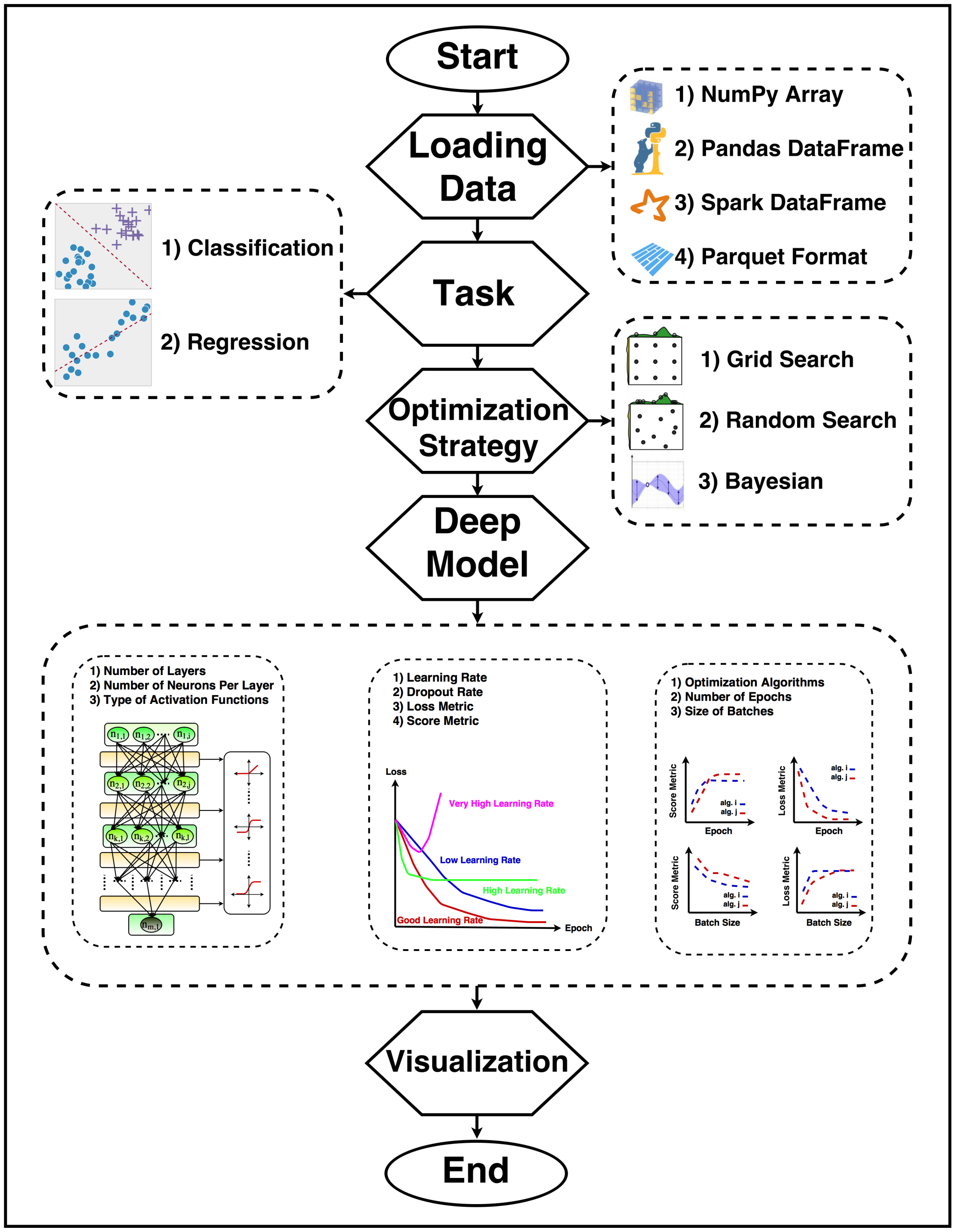

Figure 1. An illustration of iDeepLe pipeline.

the importance of the optimization algorithms in development of the deep neural networks topologies. Deep learning explores complicated structures especially in big data problems with the help of the backpropagation optimization algorithm to compute the hyper-parameters involved in the network.[2,6]

Emerging as one of the most contemporary machine learning techniques, deep learning has shown success in areas such as image classification, speech recognition, and even playing games through the use of hierarchical architecture which includes many layers of non-linear information.[7–9] Availability of huge amounts of training data, powerful computational infrastructure, and advances in academia could be named as the three main bedrocks of recent deep learning success. This encourages the development of deep learning models to be applied to real-world problems. Importantly, we live in an era where we have sufficient computational equipment and cutting-edge technologies that allow us to better optimize the hyper-parameters involved in deep neural networks. However, developing new deep models for classification or regression tasks to solve real-world problems still demands robust optimization techniques. It should be noted that, at this point, there is no consensus on choosing the right optimization method based on the network topology and architecture.[2,7] Additionally, the optimization strategies are getting more popular as well.[10–17]

In this paper, a powerful deep learning pipeline, intelligent deep learning (iDeepLe) is proposed for both regression and classification models. The core idea of the proposed pipeline is inspired by the sequential modeling and it has the ability of employing various optimization strategies including grid search, random search, and Bayesian optimization along with adaptive learning rate optimization algorithms for hyper-parameters optimization. The parallel pipeline of the proposed model has the potential to handle big data problems using Apache Spark, High Performance Computing (HPC) and GPU-enabled machines.

2. DEEP LEARNING MODEL

As of November 2017, over 200,000 users employed Keras by itself in their neural network models. Netflix, Uber, and Yelp are few examples of platforms, which built with Keras.[18] In addition to this, based on `arXiv.org` server, TensorFlow has the first rank in terms of mentions in scientific papers. Therefore, the combination of Keras and TensorFlow brings more power to the pipeline. In this way, the proposed pipeline would give the users more degrees of freedom in terms of choosing different layers, different optimizers, different backend supports, customized functions for loss metrics, score metrics, and visualization. Figure 1 presents the work flow of the iDeepLe's pipeline. As the first stage of the work flow, the pipeline begins with loading data. Based on the volume and dimensionality of the data, four different libraries including NumPy, Pandas, Spark, and Parquet can be used. As shown, after loading data level, the goal of the deep learning model in terms of regression or classification should be determined. The Scikit-Learn wrappers built on top of Keras can be employed along with different sequential core layers including densely connected networks, convolutional neural networks in one, two, and three dimensions with the ability of pooling, padding, dropping out, flattening, and reshaping for both regression and classification tasks. iDeepLe is also prepared with three optimization strategies such as grid search, random search, and Bayesian optimization to tune the hyper-parameters of the deep model. This level of the work flow can be done for the topology of the network including, number of layers, number of neurons per layer, type of activation functions per layer, learning rate, drop out rate, loss metric, score metric, number of epochs, size of batches, and the type of optimization algorithm. As the final stage, the results can be visualized to illustrate the performance of the optimized deep learning model according to the employed database.

3. BIG DATA APPROACH

The "V-V-V" or "3Vs" concept: 1) Volume, 2) Variety, and 3) Velocity introduces an arduous stage of analyzing problems with huge volume of a variety of data in high velocity. In an abstract statement of the rise of big data based on Moore's law, "world's data doubling every year". Deep neural networks can be the best choice to overcome this challenge. However, this process needs further infrastructures such parallel algorithms, high performance computing, and cloud computing.[19] As the structure of the deep models gets deeper and wider, the number of calculations increases dramatically. This brings the idea of employing computational solutions in addition to employing high-end computational equipment. In this regard, Apache Spark and Apache Arrow, which are fast and general engines for large-scale data processing were chosen to expedite the running process of the proposed pipeline.[20,21] Spark began life in 2009 as a project within the AMPLab at the University of California, Berkeley*. Spark engines can run programs 100x faster than Hadoop MapReduce in memory and 10x faster on disk. Since the proposed pipeline was written in Python, PySpark was used to implement the Spark context. Figure 2 depicts the data flow in PySpark in details. PySpark to create a Spark context used Py4J

$*$ https://amplab.cs.berkeley.edu/

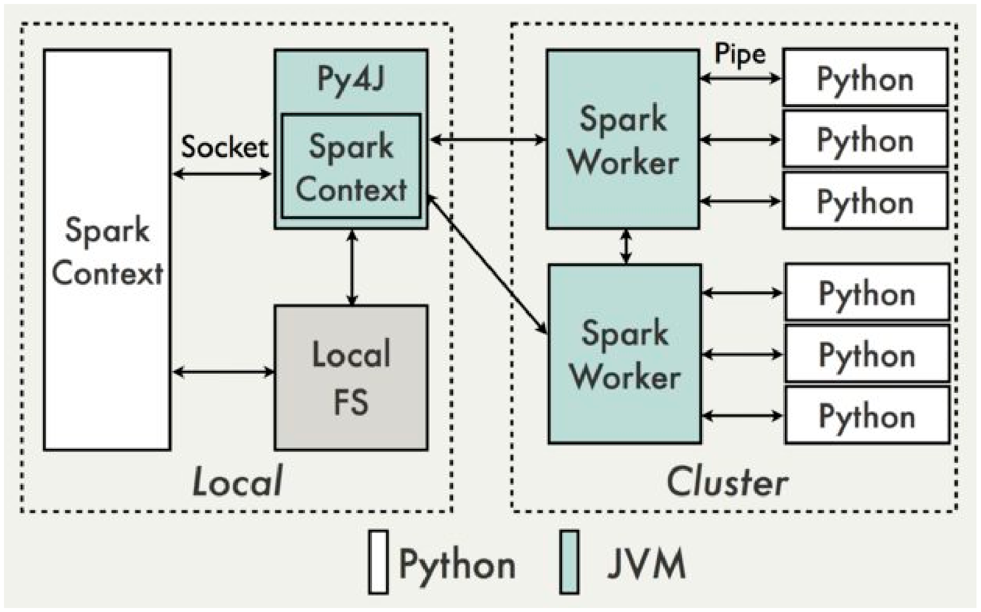

Figure 2. An illustration of PySpark data flow. The Python environments were shaded in white and the Java environments were shaded in blue**.

which is a BSD licensed Java collection method that enables Java programs to call back Python objects and create a Java Virtual Machine (JVM) which is an abstract computing machine that enables a computer to run a Java program. In addition to this, Spark is compatible with most of the Python libraries such as Pandas. In fact, the codes can be written using Pandas API and converted into Spark format. This process can also be done while loading the data into pipeline by choosing Parquet or Spark formats in schema structure other than common formats such as Comma Separated Values (CSV).

4. OPTIMIZATION

Deep learning models involve optimization in many contexts. The essential idea of optimization is finding the best weights w at each layer of the topology of the deep neural networks with the hope that these weights decrease the cost on the entire training set.[7] In contrast to the traditional optimization methods in which the optimization of the pure objective function is the direct goal, finding some weights w that minimize the cost function and reduce the expected generalization error respectively in deep model is indirect. Additional difference between optimizing the machine learning tasks and pure optimization is that the true distribution of the training data set in machine learning is unknown. To overcome this challenge, the true distribution of the training data was substituted with the empirical distribution defined by the training data and then the optimization algorithms were applied.[7,22] Optimization by itself is an arduous and time-consuming problem and that becomes more challenging for difficulties in convex problems such as ill-conditioning, local minima, saddle points, and steep cliff and non-convex problems including deep neural networks.[23] In this paper, to consider the previously mentioned challenges, various optimization strategies including grid-search and advanced strategy such as Bayesian optimization along with several optimization algorithms including basic algorithms and algorithms with adaptive learning rates for the training phase of the deep models were employed.

4.1 Optimization Strategy

As discussed, optimization by itself is a difficult and time-consuming problem. Therefore, the strategy to optimize the problem can dramatically change the training time. However, before tackling the problem, the optimization strategy according to the tuning process of hyper-parameters including the number of layers, the number of

** http://www.apache.org/

neurons per layer, activation functions, weight penalty, learning rate, and momentum should be determined. Tuning the hyper-parameter values is one of the most common reasons for not using neural networks in general. In other words, lots of skills require to set the hyper-parameters and without any prior knowledge about the hyper-parameters, the user could get completely stuck using wrong values as hyper-parameters and nothing would work. Thus, in this pipeline three different optimization strategies including grid search, random search, and Bayesian optimization were proposed to tune hyper-parameters. Grid search provides an exhaustive search over specified parameter values for the deep model. All the possible combinations of the specified hyper-parameters will be checked which demands huge amount of time and powerful computing equipment such as HPC and GPU-enabled machines. That is why sometimes random search would be a suitable alternative to grid search. In contrast to grid search, in random search not all parameter values would be tried out, but rather a fixed number of parameter settings will be sampled from the specified distributions. This is quite interesting since not all the alternative values in each list for hyper-parameters play an important role in the outcomes. Therefore, by random sampling, the most important hyper-parameters can be determined and the other hyper-parameters settings can be fixed the same as before. In this way, the same results would not be replicated anymore and the learning slope will be positive. To overcome over-fitting depending on the size of the data k-folds cross-validation can be employed.[24] Employing machine learning to predict what combinations are likely to work well could help to rescue from the huge computational time. It requires to predict the regions of the hyper-parameter space that might give better outcomes. It also requires to predict how well a new combination will do and model the uncertainty of that prediction using Gaussian Process models.[25] Gaussian processes (GPs) provide a principled, practical, and probabilistic approach in machine learning. GPs simply have an essential assumption that similar inputs give similar outputs. This simple and weak prior are actually very sensible for the effects of hyper-parameters. GPs are able to learn for each input dimension what the appropriate scale is for measuring similarity. GPs predict a Gaussian distribution of values rather than just predicting a single value. Bayesian optimization[26, 27] is a constrained global optimization approach built upon Bayesian inference and Gaussian process models to find the maximum value of an unknown function in the most efficient ways (less iterations). Algorithm 1 presents the details of how Bayesian optimization uses the prior and evidence to define a posterior distribution over the space of functions.

Algorithm 1: Bayesian Optimization

```
  Input: Training data $S$
  Output: Posterior distribution
1 for $i \in \{1, 2, \ldots, n\}$ do
2     Find $x_i$ by optimizing the acquisition function over the $GP : x_i = argmaxu(x|S_{1:i-1})$;
3     Sample the objective function: $y = f(x_i) + \epsilon_i$;
4     Augment the data $S_{1:i} = \{S_{1:i-1}, (x_i, y_i)\}$;
5     Update the GP;
6 end
```

4.2 Optimization Algorithm

Optimization is a broad and fundamental field of mathematics and in order to harness it to the deep learning ends, it needs to be narrowed down. Employing gradient descent and its variants as the optimization algorithms for deep learning and other machine learning tasks is probably the most frequent choice among users. The most popular variant is mini-batch gradient descent which is also called stochastic gradient descent (SGD). SGD performs an update for every mini-batch of size m training instances as shown in Algorithm 2. SGD is the typical algorithm for training deep neural networks. The ability of adapting the learning rate based on the data characteristics at each iteration brings another variant of gradient based optimization algorithms, Adaptive Gradient (Adagrad). As shown in Algorithm 3, Adagrad assigns lower learning rates to the repetitive input variables and also higher learning rates to less repetitive input variables. This would help to detect rare but possible input variables in regression or classification tasks. This ability of Adagrad can be revealed in any problem with sparse training data.[7,28,29] Although the learning rate in Adagrad fails to converge to zero for

Algorithm 2: SGD

```
Input: Training data $S$, learning rate $\eta$, weights $w$
Output: Updated weights $w$
1  $w \leftarrow w_0$;
2  while stopping criterion is not met do
3      Randomly shuffle the training data $S$ ;
4      Sample a minibatch of size $m$:$\{(x^{(1)}, y^{(1)}), \dots, (x^{(m)}, y^{(m)})\} \in S$ ;
5      for $i \in \{1, \dots, m\}$ do
6          $\hat{G} \leftarrow \frac{\partial}{\partial w_i} cost(w, (x^{(i)}, y^{(i)}))$; Gradient calculation
7      end
8      $w \leftarrow w - \eta \hat{G}$;
9  end
```

Algorithm 3: Adagrad

```
Input: Training data $S$, learning rate $\eta$, weights $w$, fuzz factor $\epsilon$, learning rate decay over each update $r$
Output: Updated weights $w$
1  $\epsilon \leftarrow \epsilon_0 \approx 10^{-8}$;
2  $w \leftarrow w_0$;
3  $r \leftarrow 0$;
4  while stopping criterion is not met do
5      Randomly shuffle the training data $S$ ;
6      Sample a minibatch of size $m$:$\{(x^{(1)}, y^{(1)}), \dots, (x^{(m)}, y^{(m)})\} \in S$ ;
7      for $i \in \{1, \dots, m\}$ do
8          $\hat{G} \leftarrow \frac{\partial}{\partial w_i} cost(w, (x^{(i)}, y^{(i)}))$; Gradient calculation
9      end
10     $r \leftarrow r + \hat{G} \odot \hat{G}$;
11     $w \leftarrow w - \frac{\eta}{\epsilon + \sqrt{r}} \odot \hat{G}$;
12 end
```

Algorithm 4: Adadelta

```
Input: Training data $S$, learning rate $\eta$, weights $w$, decay rate $\rho$, fuzz factor $\epsilon$
Output: Updated weights $w$
1  $\rho \leftarrow \rho_0$;
2  $\epsilon \leftarrow \epsilon_0 \approx 10^{-8}$;
3  $w \leftarrow w_0$;
4  $E[\hat{G}^2]_{t=0} \leftarrow 0$;
5  $E[\Delta w^2]_{t=0} \leftarrow 0$;
6  for $t \in \{1, \dots, T\}$ do
7      Randomly shuffle the training data $S$ ;
8      Sample a minibatch of size $m$:$\{(x^{(1)}, y^{(1)}), \dots, (x^{(m)}, y^{(m)})\} \in S$ ;
9      for $i \in \{1, \dots, m\}$ do
10         $\hat{G}_t \leftarrow \frac{\partial}{\partial w_i} cost(w_t, (x^{(i)}, y^{(i)}))$; Gradient calculation
11     end
12     $E[\hat{G}^2]_t \leftarrow \rho E[\hat{G}^2]_{t-1} + (1 - \rho)\hat{G}_t^2$;
13     $\Delta w_t \leftarrow -\frac{\sqrt{E[\Delta w^2]_{t-1} + \epsilon}}{\sqrt{E[\hat{G}^2]_t + \epsilon}} \hat{G}_t$;
14     $E[\Delta w^2]_t \leftarrow \rho E[\Delta w^2]_{t-1} + (1 + \rho)\Delta w^2{}_t$ ;
15     $w_{t+1} \leftarrow w_t + \Delta w_t$;
16 end
```

Algorithm 5: RMSprop

```
Input: Training data $S$, learning rate $\eta$, weights $w$, decay rate $\rho$, fuzz factor $\epsilon$, learning rate decay over
       each update $r$
Output: Updated weights $w$
1  $\rho \leftarrow \rho_0$;
2  $\epsilon \leftarrow \epsilon_0 \approx 10^{-8}$;
3  $w \leftarrow w_0$;
4  $r \leftarrow 0$;
5  while stopping criterion is not met do
6      Randomly shuffle the training data $S$ ;
7      Sample a minibatch of size $m$:$\{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\} \in S$ ;
8      for $i \in \{1, \ldots, m\}$ do
9          $\hat{G} \leftarrow \frac{\partial}{\partial w_i} cost(w, (x^{(i)}, y^{(i)}))$; Gradient calculation
10     end
11     $r \leftarrow \rho r + (1 - \rho)\hat{G} \odot \hat{G}$;
12     $w \leftarrow w - \frac{\eta}{\sqrt{\epsilon + r}} \odot \hat{G}$;
13 end
```

Algorithm 6: Adam

```
Input: Training data $S$, learning rate $\eta$, weights $w$, fuzz factor $\epsilon$, learning rates decay over each update $r_1$
       and $r_2$, exponential decay rates $\beta_1$ and $\beta_2$
Output: Updated weights $w$
1  $\epsilon \leftarrow \epsilon_0 \approx 10^{-8}$;
2  $w \leftarrow w_0$;
3  $r_1 \leftarrow 0$;
4  $r_2 \leftarrow 0$;
5  $t \leftarrow 0$;
6  while stopping criterion is not met do
7      Randomly shuffle the training data $S$ ;
8      Sample a minibatch of size $m$:$\{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\} \in S$ ;
9      for $i \in \{1, \ldots, m\}$ do
10         $\hat{G} \leftarrow \frac{\partial}{\partial w_i} cost(w, (x^{(i)}, y^{(i)}))$; Gradient calculation
11         $t \leftarrow t + 1$;
12     end
13     $r_1 \leftarrow \beta_1 r_1 + (1 - \beta_1)\hat{G}$;
14     $r_2 \leftarrow \beta_2 r_2 + (1 - \beta_2)\hat{G} \odot \hat{G}$;
15     $\hat{r_1} \leftarrow \frac{r_1}{1 - \beta_1^t}$;
16     $\hat{r_2} \leftarrow \frac{r_2}{1 - \beta_2^t}$;
17     $w \leftarrow w - \eta \frac{\hat{r_1}}{\epsilon + \sqrt{\hat{r_2}}}$;
18 end
```

```
Algorithm 7: Adamax
  Input: Training data $S$, learning rate $\eta$, weights $w$, fuzz factor $\epsilon$, learning rate decay over each update $r$,
         exponentially weighted infinity norm $u$, exponential decay rates $\beta_1$ and $\beta_2$
  Output: Updated weights $w$
1  $\epsilon \leftarrow \epsilon_0 \approx 10^{-8}$;
2  $w \leftarrow w_0$;
3  $r \leftarrow 0$;
4  $u \leftarrow 0$;
5  $t \leftarrow 0$;
6  while stopping criterion is not met do
7      Randomly shuffle the training data $S$ ;
8      Sample a minibatch of size $m$:$\{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\} \in S$ ;
9      for $i \in \{1, \ldots, m\}$ do
10         $\hat{G}_t \leftarrow \frac{\partial}{\partial w_i} cost(w_t, (x^{(i)}, y^{(i)}))$; Gradient calculation
11         $t \leftarrow t + 1$;
12     end
13     $r_t \leftarrow \beta_1 r_{t-1} + (1 - \beta_1)\hat{G}_t$;
14     $u_t \leftarrow \max(\beta_2 u_{t-1}, |\hat{G}_t|)$;
15     $w_t \leftarrow w_{t-1} - \frac{\eta r_t}{(1-\beta_1^t)u_t}$;
16 end
```

```
Algorithm 8: Nadam
  Input: Training data $S$, learning rate $\eta$, weights $w$, fuzz factor $\epsilon$, learning rates decay over each update $r_1$
         and $r_2$, momentum decay rate $\gamma$, exponential decay rates $\beta_1$ and $\beta_2$
  Output: Updated weights $w$
1  $\epsilon \leftarrow \epsilon_0 \approx 10^{-8}$;
2  $w \leftarrow w_0$;
3  $t \leftarrow 0$;
4  $r_1 \leftarrow 0$;
5  $r_2 \leftarrow 0$;
6  while stopping criterion is not met do
7      Randomly shuffle the training data $S$ ;
8      Sample a minibatch of size $m$:$\{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\} \in S$ ;
9      for $i \in \{1, \ldots, m\}$ do
10         $\hat{G}_t \leftarrow \frac{\partial}{\partial w_i} cost(w_t, (x^{(i)}, y^{(i)}))$; Gradient calculation
11         $t \leftarrow t + 1$;
12     end
13     $r_{1t} \leftarrow \beta_1 r_{1t-1} + (1 - \beta_1)\hat{G}_t$;
14     $\hat{r_1}_t \leftarrow \frac{r_{1t}}{1-\beta_1^t}$;
15     $w_{t+1} \leftarrow w_t - \frac{\eta}{\epsilon + \sqrt{r_{2t}}}(\beta_1 r_{1t} + \frac{1-\beta_1}{1-\beta_1^t}\hat{G}_t)$;
16 end
```

long iterations, Adaptive Delta (Adadelta) solves this problem based on two main ideas as shown in Algorithm 4: (1) scaling the learning rate to avoid observing discontinuity in the learning progress by restricting the past gradients for a fixed size instead of incorporating the whole historical gradient information, (2) employing an acceleration term such as momentum[30] to process the first idea.[31] Similar to the Adadelta algorithm, a modified version of the Adagrad algorithm that divides the learning rate by an exponentially decaying average of squared gradients, Root Mean Square Propagation (RMSprop) is presented as shown in Algorithm 5. RMSprop would

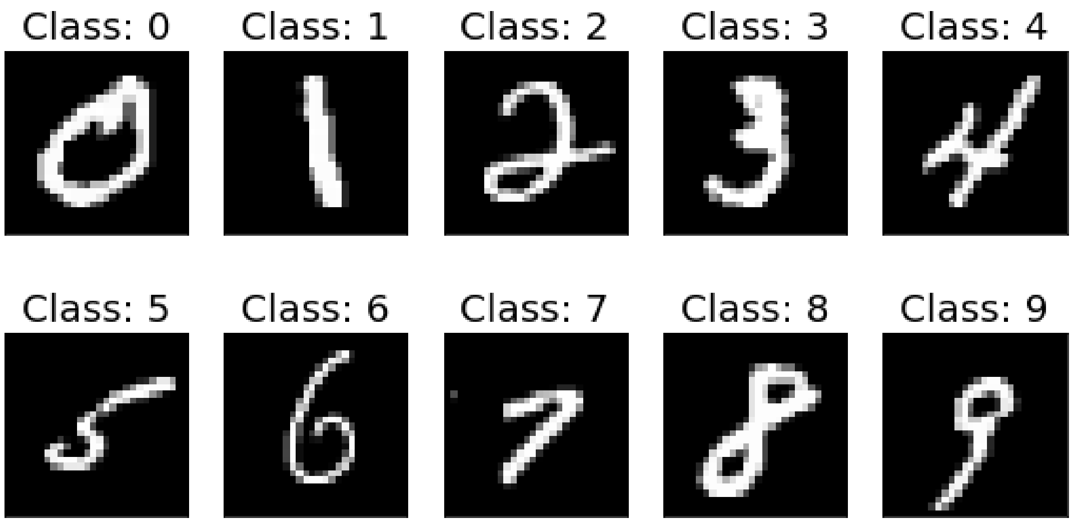

Figure 3. An illustration of the classes in the MNIST database.

outperform Adagrad in non-convex problems due to the learning rate shrinkage of the Adagrad algorithm. Moreover, as a generalization of the Adagrad algorithm by calculating and updating some statistics such as the first and the second moments of historical gradients at each iteration, Adaptive Moment Estimation (Adam) is presented as shown in Algorithm 6. Adam requires a little memory to process and it is suitable for noisy and big data problems in terms of both dimension and volume. Additionally, by changing the updating norm of the weights from the L_2 norm of the previous and current gradients to the L_∞ norm, Adaptive Moment Estimation based on the Infinity Norm (Adamax) is presented as shown in Algorithm 7. As a variant of the Adam algorithm, Nesterov Adaptive Moment Estimation (Nadam) is presented by incorporating Nesterov[32] momentum[30] to accelerate the learning process by summing up the exponential decay of the moving average of the past and current gradients as shown in Algorithm 8. Nadam combines the opposite signs of gradients in directions of high curvature with higher speed to damp the fluctuations which is suitable for problems with noisy gradients or gradients with high curvature in particular.[7,33] Furthermore, a combination of the Nesterov momentum and the Adamax algorithm leads us to another variant of Adam, referred to as the Nadamax algorithm.[34,35] Tahmassebi et al.[2] were shown the performance of the optimization algorithms discussed in section 4.2 in details for solving a real-world problem.

Table 1. Parameter settings of the simple and the deep model.

	Simple Model	**Deep Model**
Number of Input Neurons	784	784
Number of Output Neurons	10	10
Number of Hidden Layers	1	4
Array of Hidden Neurons	{100}	{512, 256, 128, 64 }
Hidden Activation Function	Sigmoid	ReLU
Output Activation Function	Softmax	Softmax
Total Number of Parameters	79,510	575,050

”With four parameters I can fit an elephant, and with five I can make him wiggle his trunk (John von Neumann).”

5. RESULTS & DISCUSSION

To test out the performance of the proposed pipeline in terms of the introduced optimization strategies and optimization algorithms, a famous classification problem was employed. In this regard, the MNIST [***] database (Modified National Institute of Standards and Technology database) which is a large database of handwritten digits $(0, 1, 2, \cdots, 9)$ of size 28×28 pixels was used. It is one of the most common databases to train deep learning models and can be a great benchmark to compare the optimization strategies and optimization algorithms. The training set of the database has $60,000$ exemplars and the testing set has $10,000$ exemplars.

Two different neural networks models were designed to be applied to the MNIST database: 1) simple model, 2) deep model. The details of the designed models are presented in Table 1. In this section, the impact of various hyper-parameters in the performance of the simple model was investigated. Additionally, the performance of the deep model using different hyper-parameters was investigated as well.

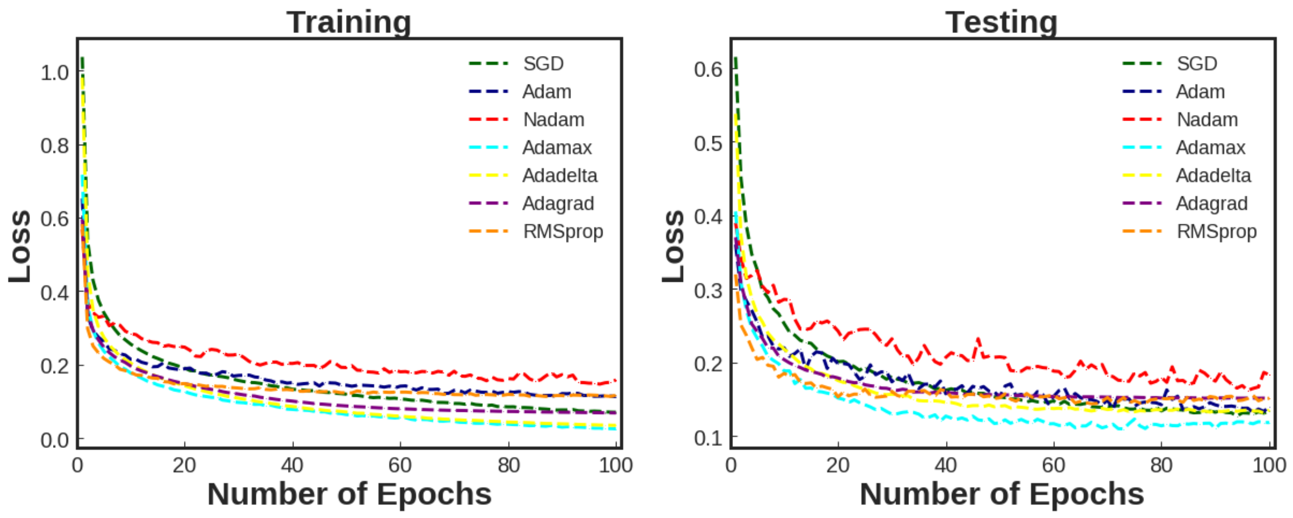

Figure 4. The performance of the optimization algorithms on the simple model: Loss evolution.

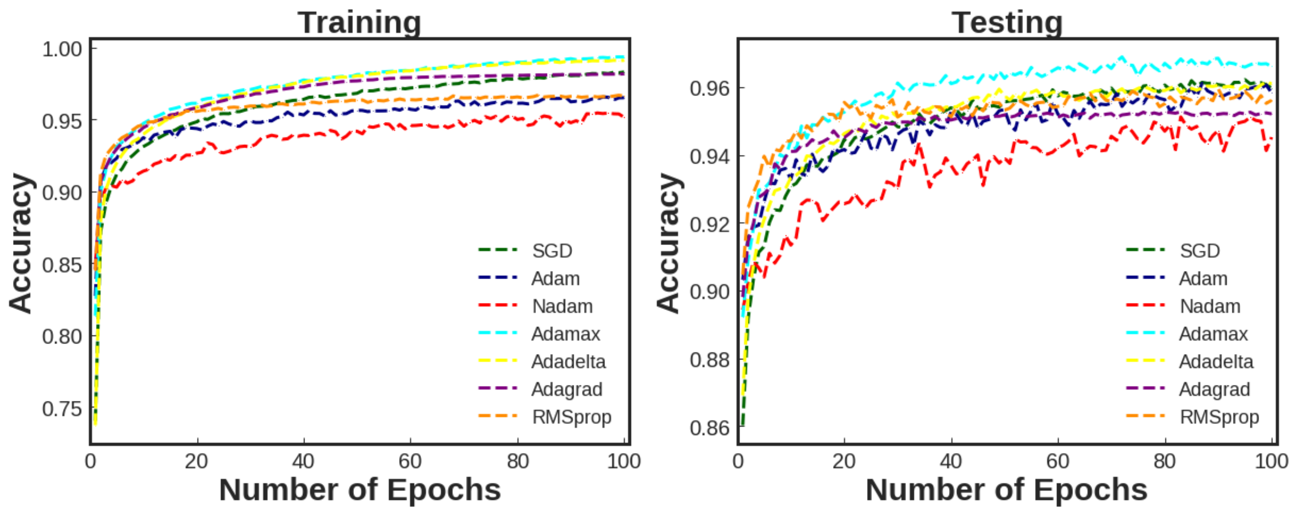

Figure 5. The performance of the optimization algorithms on the simple model: Classification accuracy.

$* * *$ http://yann.lecun.com/exdb/mnist/

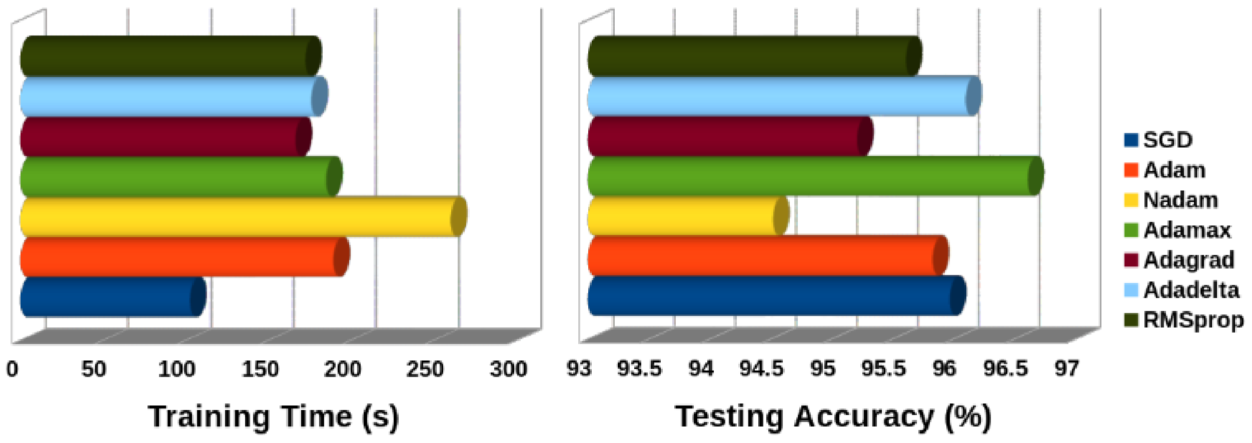

Figure 6. The performance of the optimization algorithms on the simple model: Training time vs testing accuracy.

As the first task for the simple model, an exhaustive comparison of the classification accuracy and loss evolution through 100 epochs employing all the optimization algorithms previously discussed in section 4 was presented. Figure 4 illustrates an exhaustive comparison of the loss evolution through the epochs with employing various optimization algorithms using the MNIST database with a batch size of 128 and categorical cross entropy as loss function for both training and testing sets. Adamax has shown the best performance among the optimization algorithms with a classification accuracy value of 96.5% on the testing set. Moreover, the Adamax algorithm has the least loss value in both training and testing process. However, the Nadam algorithm has the highest loss value among all of the employed algorithms in both training and testing processes. These results are in agreement with the results that are presented in Figure 5. Figure 5 shows an exhaustive comparison of the classification accuracy evolution through the epochs employing various optimization algorithms using the MNIST database with a batch size of 128 and categorical cross entropy as loss function for both training and testing sets. The Adamax and the Nadam algorithms with values of 96.61% and 94.52% have shown the highest and the lowest values for classification accuracy. It should be noted that the deviation between the highest and the lowest classification accuracies is roughly 2%. However, this deviation in the performance of the deep model would be dramatically different due to the stability of the model based on different optimization algorithms. Additionally, the wall-clock time for the training process using the discussed algorithms is presented in Figure 6. Nadam has the most and SGD the least training time (102 seconds). It should always be a balance between the amount of time needs to be spent for running a model and the desired accuracy. For instance, as seen in Figure 6, the SGD has the least computational run-time with a value of 95.97% accuracy which is quite the same as the Adamax performance with training time value of 184.97 seconds.

Figure 7 presents an exhaustive comparison of three loss metrics including mean-absolute-error (MAE), mean-square-error (MSE), and categorical cross entropy (CCE) using three different learning rates (η) 0.1, 0.01, and 0.001 through 100 epochs employing Adamax optimization algorithm using the MNIST database with a batch size of 128 for both training and testing sets. As seen, the evolution of the loss metrics are getting smoother as the learning rates got smaller. In addition to this, Figure 8 illustrates an exhaustive comparison of the training time and the testing accuracy using three loss metrics including MAE, MSE, and CCE using three different learning rates (η) 0.1, 0.01, and 0.001 through 100 epochs employing Adamax optimization algorithm using the MNIST database with a batch size of 128 for both training and testing sets. As seen, employing larger learning rates requires much more time for the training process. This amount of time has not necessary been spent to get better results as seen in Figure 7. In addition to this, as the learning rate got smaller, the numbers of step size were increased. Therefore, the training and testing evolution plots got smoother in comparison to larger learning rates as shown in Figure 7.

Figure 9 presents an exhaustive comparison of three loss metrics including MAE, MSE, and CCE using three different batch sizes 16, 64, and 256 through 100 epochs employing Adamax optimization algorithms with

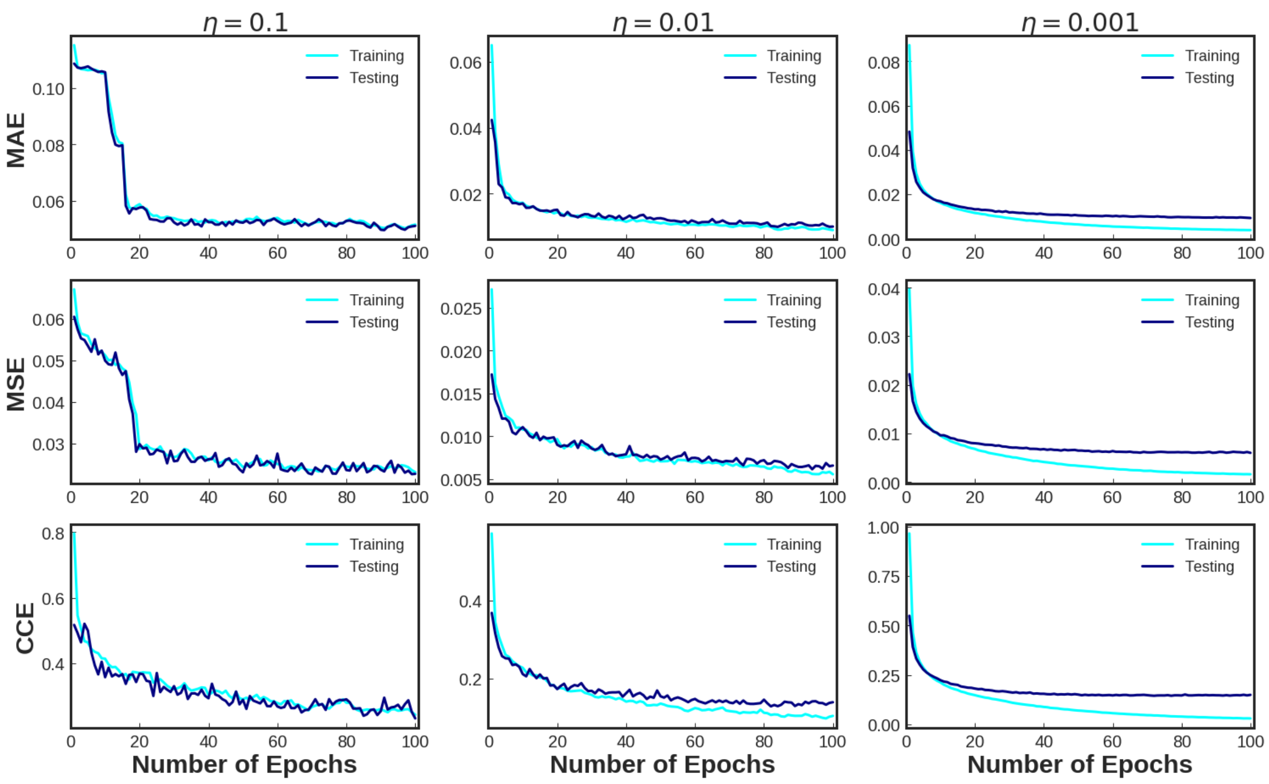

Figure 7. The impact of learning rate on loss evolution of the simple model.

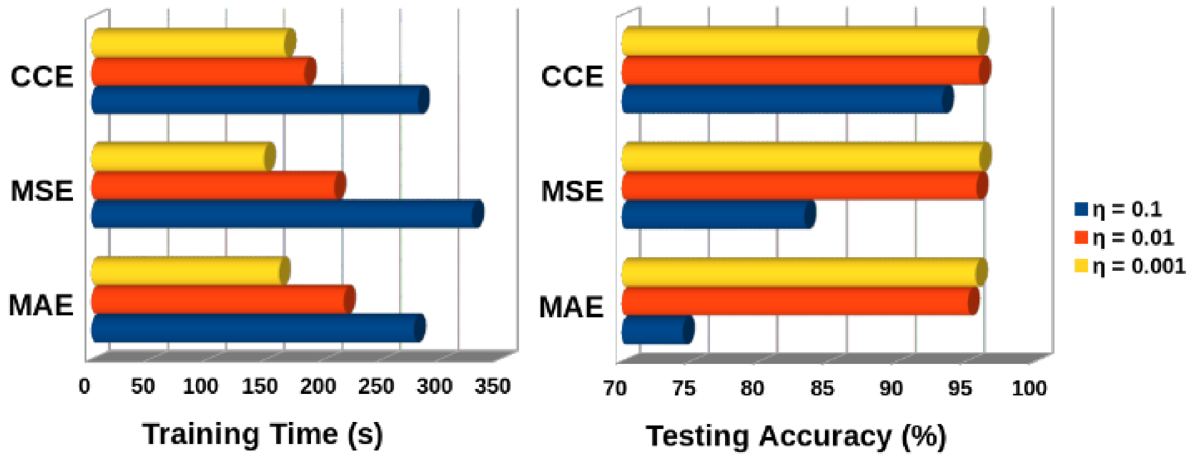

Figure 8. The impact of learning rate on loss evolution of the simple model: Training time vs testing accuracy.

a learning rate of 0.001 using the MNIST database for both training and testing sets. Furthermore, Figure 10 depicts an exhaustive comparison of training time and testing accuracy using three loss metrics including MAE, MSE, and CCE using three different batch sizes 16, 64, and 256 through 100 epochs employing Adamax optimization algorithms with a learning rate of 0.001 using the MNIST database for both training and testing sets. As seen employing a small batch size requires more training time with better results. In other words, it is

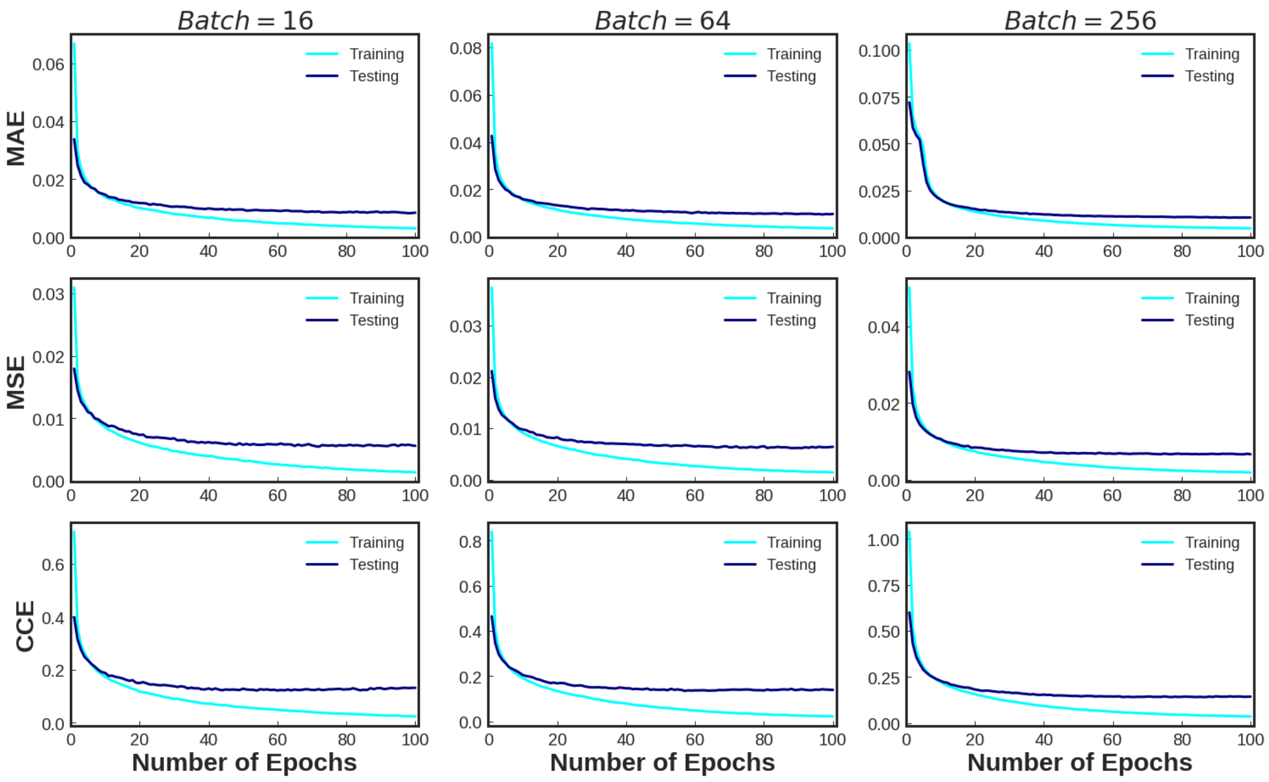

Figure 9. The impact of batch size on loss evolution of the simple model.

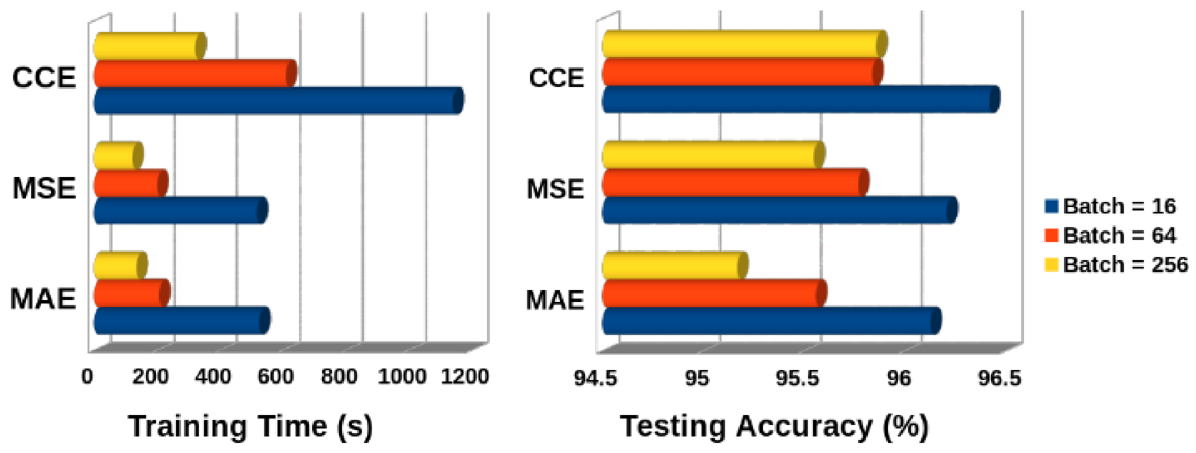

Figure 10. The impact of batch size on loss evolution of the simple model: Training time vs testing accuracy.

a trade-off between computational training time and testing accuracy. The training time and the size of batches have inverse correlation. As the size of batches increases, the training time decreases accordingly. Moreover, it should be noted that the CCE as the loss metric would be the best choice for simple models as shown in Figure 8 and Figure 10. Figure 11 shows an exhaustive comparison of the loss (MSE) and classification accuracy evolution through 100 epochs for both training and testing sets of the MNIST database using the deep model. In this part, SGD was employed as the optimization algorithm and the impact of the learning rate is depicted in Figure 11.

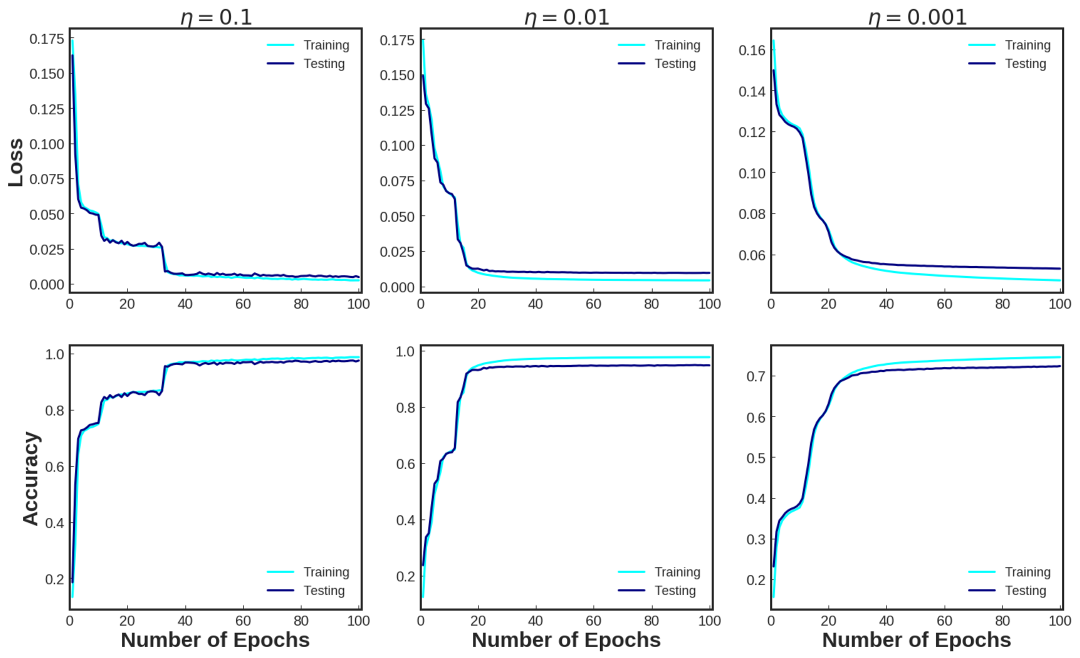

Figure 11. Impact of learning rate on the performance of the deep model: Loss vs accuracy.

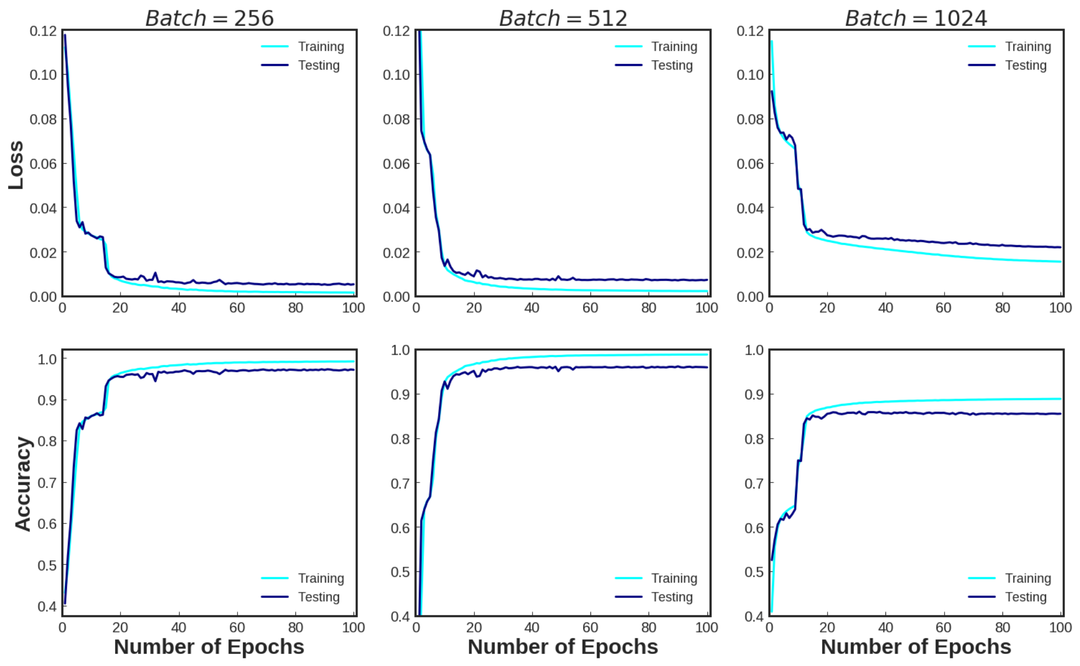

Figure 12. Impact of batch size on the performance of the deep model: Loss vs accuracy.

In addition to this, the impact of different batch sizes on the performance of the deep model was investigated. In contrast to the performance of the simple model, the deep model with a smaller learning rate showed the best performance with a classification accuracy of 97.50%. The question should be addressed here will be do we get the same results with employing the other types of optimization algorithms like the simple model? The answer is a "Big No". Employing another algorithm such as Adam with the same learning rate would end up with a over-fitted model with a classification accuracy of less than 20% on the testing set. In fact, as the structure of the networks gets more complex, the numbers of choices for hyper-parameters would get smaller accordingly. In other words, as the topology of the network gets deeper, the surface of the cost function gets more complicated and not all of the optimization algorithms can track all of the extrema points on the cost surface. That is why the optimization of the hyper-parameters in deep neural network is critical and important. Additionally, the strategy to optimize the hyper-parameters is more important. For example, for the deep model in this paper, there are $575,050$ parameters to optimize which will be a rough, tedious, and time-consuming process for an array of choices for each of the hyper-parameters. In addition to this, Keskar and Socher[36] have indicated that the optimization algorithms with adaptive learning rates have been found to generalize poorly compared to SGD algorithm.

”Now this is not the end. It is not even the beginning of the end. But it is, perhaps, the end of the beginning (Winston Churchill).”

6. CONCLUSIONS

A powerful deep learning pipeline, intelligent deep learning (iDeepLe) is proposed for both regression and classification tasks. iDeepLe is written in Python with the help of various API libraries such as Keras, TensorFlow, Scikit-Learn, and Apache Spark. iDeepLe has the ability of employing exhaustive grid search, random search, and Bayesian optimization to optimize the hyper-parameters in the deep model via adaptive learning rate optimization algorithms for both accuracy and complexity, while simultaneously solving the unknown parameters of the regression or the classification model. iDeepLe is shown to be a suitable tool to generate solid models for complex nonlinear systems. In this paper, an exhaustive investigation of the various hyper-parameters such as learning rate, loss function, score metric, batch size, and optimization algorithm employing two designed models: a simple and a complex neural networks topologies tested on MNIST database was carried out. This study highlights the key-points of hyper-parameters tunning in both simple and deep neural networks structures.

7. ACKNOWLEDGEMENTS

The author would like to thank Dr. Anke Meyer-Baese, Dr. Amir H. Gandomi, Dr. Katja Pinker-Domenig, and Dr. Gordon Erlebacher for their unlimited supports.

8. CONFLICTS OF INTEREST

The author declares that there is no conflict of interest regarding the publication of this article.

REFERENCES

[1] McCulloch, W. S. and Pitts, W., "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics* **5**(4), 115–133 (1943).

[2] Tahmassebi, A., Gandomi, A. H., Meyer-Baese, A., and Foo, S. Y., "Multi-stage optimization of deep model: A case study on ground motion modeling," *PloS one* (2018).

[3] Hinton, G. E., Osindero, S., and Teh, Y.-W., "A fast learning algorithm for deep belief nets," *Neural computation* **18**(7), 1527–1554 (2006).

[4] Bengio, Y., Lamblin, P., Popovici, D., and Larochelle, H., "Greedy layer-wise training of deep networks," in [*Advances in neural information processing systems*], 153–160 (2007).

[5] Cortes, C. and Vapnik, V., "Support-vector networks," *Machine learning* **20**(3), 273–297 (1995).

[6] LeCun, Y., Bengio, Y., and Hinton, G., "Deep learning," *Nature* **521**(7553), 436–444 (2015).

[7] Goodfellow, I., Bengio, Y., and Courville, A., [*Deep Learning*], MIT Press (2016).

[8] Krauss, C., Do, X. A., and Huck, N., "Deep neural networks, gradient-boosted trees, random forests: Statistical arbitrage on the s&p 500," *European Journal of Operational Research* **259**(2), 689–702 (2017).

[9] Gal, T. and Hanne, T., "Nonessential objectives within network approaches for mcdm," *European Journal of Operational Research* **168**(2), 584–592 (2006).

[10] Tahmassebi, A., Pinker-Domenig, K., Wengert, G., Lobbes, M., Stadlbauer, A., Romero, F. J., Morales, D. P., Castillo, E., Garcia, A., Botella, G., et al., "Dynamical graph theory networks techniques for the analysis of sparse connectivity networks in dementia," in [*Smart Biomedical and Physiological Sensor Technology XIV*], **10216**, 1021609, International Society for Optics and Photonics (2017).

[11] Tahmassebi, A., Pinker-Domenig, K., Wengert, G., Lobbes, M., Stadlbauer, A., Wildburger, N. C., Romero, F. J., Morales, D. P., Castillo, E., Garcia, A., et al., "The driving regulators of the connectivity protein network of brain malignancies," in [*Smart Biomedical and Physiological Sensor Technology XIV*], **10216**, 1021605, International Society for Optics and Photonics (2017).

[12] Tahmassebi, A., Gandomi, A. H., McCann, I., Schulte, M. H., Schmaal, L., Goudriaan, A. E., and Meyer-Baese, A., "An evolutionary approach for fmri big data classification," in [*Evolutionary Computation (CEC), 2017 IEEE Congress on*], 1029–1036, IEEE (2017).

[13] Tahmassebi, A., Gandomi, A. H., McCann, I., Schulte, M. H., Schmaal, L., Goudriaan, A. E., and Meyer-Baese, A., "fmri smoking cessation classification using genetic programming," in [*Workshop on Data Science meets Optimization*], (2017).

[14] Tahmassebi, A. and Gandomi, A. H., "Building energy consumption forecast using multi-objective genetic programming," *Measurement* (2018).

[15] Tahmassebi, A. and Gandomi, A. H., "Genetic programming based on error decomposition: A big data approach," in [*Genetic Programming Theory and Practice XIV*], Springer (2018).

[16] Tahmassebi, A., Amani, A. M., Pinker-Domenig, K., and Meyer-Baese, A., "Determining disease evolution driver nodes in dementia networks," in [*Medical Imaging 2018: Biomedical Applications in Molecular, Structural, and Functional Imaging*], **10578**, 1057829, International Society for Optics and Photonics (2018).

[17] Tahmassebi, A., Pinker-Domenig, K., Wengert, G., Helbich, T., Bago-Horvath, Z., and Meyer-Baese, A., "Determining the importance of parameters extracted from multi-parametric mri in the early prediction of the response to neo-adjuvant chemotherapy in breast cancer," in [*Medical Imaging 2018: Biomedical Applications in Molecular, Structural, and Functional Imaging*], **10578**, 1057810, International Society for Optics and Photonics (2018).

[18] Chollet, F. et al., "Keras," (2015).

[19] Tahmassebi, A., Gandomi, A. H., and Meyer-Bäse, A., "High performance gp-based approach for fmri big data classification," in [*Proceedings of the Practice and Experience in Advanced Research Computing 2017 on Sustainability, Success and Impact*], 57, ACM (2017).

[20] Meng, X., Bradley, J., Yavuz, B., Sparks, E., Venkataraman, S., Liu, D., Freeman, J., Tsai, D., Amde, M., Owen, S., et al., "Mllib: Machine learning in apache spark," *The Journal of Machine Learning Research* **17**(1), 1235–1241 (2016).

[21] Shoro, A. G. and Soomro, T. R., "Big data analysis: Apache spark perspective," *Global Journal of Computer Science and Technology* **15**(1) (2015).

[22] Bousquet, O. and Bottou, L., "The tradeoffs of large scale learning," in [*Advances in neural information processing systems*], 161–168 (2008).

[23] Fong, S., Deb, S., Hanne, T., and Li, J. L., "Eidetic wolf search algorithm with a global memory structure," *European Journal of Operational Research* **254**(1), 19–28 (2016).

[24] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E., "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research* **12**, 2825–2830 (2011).

[25] Rasmussen, C. E. and Williams, C. K., [*Gaussian processes for machine learning*], vol. 1, MIT press Cambridge (2006).

[26] Snoek, J., Larochelle, H., and Adams, R. P., "Practical bayesian optimization of machine learning algorithms," in [*Advances in neural information processing systems*], 2951–2959 (2012).

[27] Brochu, E., Cora, V. M., and De Freitas, N., "A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning," *arXiv preprint arXiv:1012.2599* (2010).

[28] Ruder, S., "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747* (2016).

[29] Duchi, J., Hazan, E., and Singer, Y., "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research* **12**(Jul), 2121–2159 (2011).

[30] Polyak, B. T., "Some methods of speeding up the convergence of iteration methods," *USSR Computational Mathematics and Mathematical Physics* **4**(5), 1–17 (1964).

[31] Zeiler, M. D., "Adadelta: an adaptive learning rate method," *arXiv preprint arXiv:1212.5701* (2012).

[32] Nesterov, Y., [*Introductory lectures on convex optimization: A basic course*], vol. 87, Springer Science & Business Media (2013).

[33] 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).

[34] Dozat, T., "Incorporating nesterov momentum into adam," (2016).

[35] Sutskever, I., Martens, J., Dahl, G., and Hinton, G., "On the importance of initialization and momentum in deep learning," in [*International conference on machine learning*], 1139–1147 (2013).

[36] Keskar, N. S. and Socher, R., "Improving generalization performance by switching from adam to sgd," *arXiv preprint arXiv:1712.07628* (2017).