Expert Systems With Applications 63 (2016) 1–7

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa

Deep learning with adaptive learning rate using laplacian score

B. Chandra*, Rajesh K. Sharma

Indian Institute of Technology Delhi, New Delhi, India

ARTICLE INFO

Article history:
Received 28 September 2015
Revised 12 May 2016
Accepted 13 May 2016
Available online 16 May 2016

Keywords:
Adaptive learning rate
Deep learning
Gradient descent
Laplacian score

ABSTRACT

An attempt has been made to improve the performance of Deep Learning with Multilayer Perceptron (MLP). Tuning the learning rate or finding an optimum learning rate in MLP is a major challenge. Depending on the value of the learning rate, classification accuracy can vary drastically. This issue has been taken as a challenge in this paper. In this paper, a new approach has been proposed to combine adaptive learning rate in conjunction with the concept of Laplacian score for varying the weights. Learning rate is taken as a function of parameter which itself is updated on the basis of error gradient by forming mini-batches. Laplacian score of the neuron is further used for updating the incoming weights. This removes the bottleneck involved in finding the optimum value for the learning rate in Deep Learning by using MLP. It is observed on benchmark datasets that this approach leads to increase in classification accuracy as compared to the existing benchmark levels achieved by the well known methods of deep learning.

1. Introduction

Deep learning has achieved the state-of-the-art results in the field of Machine Learning such as computer vision and speech recognition. The field of Deep learning started with the development of Deep Belief Network (Hinton, Osindero, & Teh, 2006). Deep Belief Network uses unsupervised pre-training and supervised fine tuning. Based on the same concept of unsupervised pre-training, several algorithms have been developed such as Stacked Denoising autoencoder (Vincent, Larochelle, Lajoie, Bengio, & Manzagol, 2010) and Contractive auto-encoders (Rifai, Vincent, Muller, Glorot, & Bengio, 2011).

With the development of Rectified Linear Network (Glorot, Bordes, & Bengio, 2011), it has been possible to get superior performance in deep learning without unsupervised pre training. Deep learning is prone to over-fitting since it deals with training a large number of parameters. In order to prevent Deep MLP from over-fitting, several regularization conditions have been proposed such as adding noise to the input (Vincent et al., 2010), using probabilistic activation functions (Hinton et al., 2006), dropout (Hinton, Srivastava, Krizhevsky, Sutskever, & Salakhutdinov, 2012) and making the activation values sparse (Glorot et al., 2011). In dropout, the input and hidden neurons are masked with certain probability during each iteration. Dropout has been shown to improve the performance of different types of neural networks (Dahl, Sainath & Hinton, 2013; Goodfellow, Warde-Farley, Mirza, Courville, & Bengio, 2013; Hinton et al., 2012).

Max-out Network has been developed (Goodfellow et al., 2013) for Deep learning without unsupervised pre-training. Both Max-out Network and Rectified Linear Network use dropout to prevent overtraining. Though there is a marked improvement in the performance by using Rectified Linear Network and Max-out Network, there is a limitation due to the involvement of hyper-parameters.

When deep learning is used for classifying images for intelligent systems, an appropriate learning rate has to be chosen for each of the layers which is a difficult task. Moreover, intelligent systems require high precision in terms of accuracy.

Multilayer Perceptron (MLP) has been applied successfully for the detection of medical fraud. Neural Network has been used for intrusion detection in which an expert system module has been included which analyses the output of neural network and relates it for intrusion detection. However, there is another important area where credit card frauds have been detected by using neural network (Patidar & Sharma, 2011; Raj & Portia, 2011, March). Neural Network has also been used to control an autonomous surface vehicle where the vehicle dynamics are unknown and suffer from uncertainties (Pan, Lai, Yang, & Wu, 2013).

MLP has been used to predict the utility of online reviews (Lee & Choeh, 2014). Details of the product and characteristics of the tests contained in the reviews are used as input to MLP in order to make prediction about usefulness of the reviews. Another application in expert systems includes forecasting of movie revenues during the pre-production (Ghiassi, Lio, & Moon, 2015). A model has been developed by using MLP to forecast the total movie revenue during the pre-production phase itself. MLP uses pre-release

* Corresponding author at: Room MZ 149, Indian Institute of Technology Delhi, India.
E-mail addresses: bchandra104@yahoo.co.in (B. Chandra), justrks@gmail.com (R.K. Sharma).

http://dx.doi.org/10.1016/j.eswa.2016.05.022
0957-4174/

advertisement expenditure, runtime, seasonality of tentative release date and production budget to make accurate prediction about the movie revenue.

In business domain, Neural Network approach has been successfully applied to predict the outcome of price negotiation (Moosmayer, Chong, Liu, & Schuppar, 2013). The price negotiation process of companies has been analyzed by predicting the final negotiated price between the seller and buyer by using Neural Network where sellers' reservation price, target price and initial offer price are used to make the prediction. Another application of Neural network in business domain is for the prediction of bankruptcy (Iturriaga & Sanz, 2015) where Neural network is used to make three year ahead prediction of bankruptcy risk on the basis of information about bank's loans portfolio, default rate, capital composition, liquidity etc.

Neural Network has also been used to predict the direction of stock price changes on the basis of technical analysis, fundamental analysis and time series analysis (Dase & Pawar, 2010; de Oliveira, Nobre, & Zarate, 2013; Hadavandi, Shavandi, & Ghanbari, 2010; Kara, Boyacioglu, & Baykan, 2011).

In order to train the neural networks without using learning rate, Expectation Propagation (EP) method was proposed (Soudry, Hubara, & Meir, 2014). However, this method has several limitations. EP specifically uses stepwise activation functions and cannot be generalized in case of other types of neural networks. It has been quoted by the authors (Soudry et al., 2014) that EP is slower as compared to the standard back propagation algorithm.

In methods where learning rate is considered as a hyper-parameter, several DNNs are trained with different values of learning rate. The learning rate that has the least validation error is chosen as the optimum learning rate. This leads to high computation cost due to the training of several DNNs. In this paper, a novel method has been proposed to solve the problem of optimizing the learning rate hyper-parameter. Learning rate is taken as a function of another parameter which is updated along with the weights based on the error. In addition, the proposed method also uses the concept of Laplacian score (He, Cai, & Niyogi, 2005). Laplacian Score of the activation values of a particular layer gives a measure of relevance of the activation values of neurons in that layer. Since the activation value directly depends on the incoming weights, the value of incoming weights to the neurons with high Laplacian Scores are more significant than the weights corresponding to the neurons with low Laplacian Score. This follows that learning rate for incoming weight to neuron with higher Laplacian Score should be lower than the incoming weights to other neurons.

It has been shown on benchmark datasets that the proposed method achieves lower misclassification error than Rectified Linear network and Max-out network with dropout.

2. Overview of previous work

This section presents an overview of some of the existing techniques used in deep learning. Since the proposed work uses Deep MLP without pre-training, recent developments in deep learning without pre-training have been discussed.

Dropout technique is a regularization technique which efficiently prevents the network from overtraining. In dropout, the neurons in input and hidden layers are randomly masked with certain probability during each training iteration. Since the network obtained during an iteration can contain any combination of neurons, dropout removes the mutual dependence of neurons on each other, and hence increases the generalization capacity of the network. The Dropout technique was used to train Rectified Linear Network (Dahl, Sainath, & Hinton, 2013), which further improved the performance of Rectified Linear Network.

Maxout Network (Goodfellow et al., 2013) takes full advantage of dropout technique. For a network having I and J as the size of input and hidden layers, the Maxout Unit is given as follows.

$$f_j(x) = \underset{k\in[1,K]}{max}\ Z_{jk}$$

Where f_j denotes the activation of j^{th} hidden neuron and Z_{jk} is given by

$$Z_{jk} = \sum_{i=1}^{I} x_i W_{ij}^{(k)} + b_{jk}$$

Where $x \in R^I$ denotes the input, K is the number of linear units in Max-out unit, $W \in R^{IxJxk}$ and $b \in R^{JxK}$ denotes the weight and bias respectively. The Maxout activation is computed as maximum over K linear functions and hence can approximate any activation function for the large value of K.

Rectified Linear activation function (Glorot et al., 2011) is used in Deep MLP to remove the problem of vanishing gradient which occurs in the case of nonlinear activation functions such as sigmoid and tanh. Thus, it facilitates learning in supervised manner without unsupervised pre-training. The Rectified Linear activation function is given by

$$f(x) = max(0, x)$$

Where x is the net input to the neuron.

ADADELTA (Zeiler, 2012) is an adaptive learning rate algorithm which updates the learning rate of each parameter in a Deep MLP during training. The learning rate is updated by using the moving average of squares of weight updates and squares of gradients. Square root of the ratio between the two moving averages is taken as the learning rate. There are two hyper-parameters in ADADELTA, but the hyper-parameters have been shown to have no significant impact on the performance.

Based on gradient, hyper-parameter optimization method has been proposed (Maclaurin, Duvenaud, & Adams, 2015) to optimize hyper-parameters while training. The method has several limitations in the sense that for removing the exploding gradient problem, the learning rates are initialized to small values and their optimization is stopped when the error gradient begins to grow in magnitude. Hence, manual intervention is necessary for training of each hyper-parameter.

3. Proposed method

In Deep learning algorithms which use gradient descent, learning rate is considered as a hyper-parameter and is optimized on the basis of the least validation error. Magnitude of back propagated error derivative is less for lower layers as compared to the upper layers (Hochreiter, Bengio, Frasconi, & Schmidhuber, 2001). Hence, different layers should have different learning rates for efficient training. With increase in the depth of Neural Network, there is an exponential growth in the possible sets of learning rates. Search for optimal hyper-parameter requires training for many DNNs with different learning rates in each layer, and hence the optimization of learning rate is computationally expensive.

To remove the problem of optimizing learning rate, a new method has been proposed which removes the drawbacks of previous methods such as Expectation Propagation (EP) (Soudry et al., 2014). EP has a drawback that it is derived specifically for stepwise activation function and cannot be generalized in case of other types of activation functions. In addition, training of EP is two to five times slower as compared to the standard back propagation.

An attempt has been made to develop a new hyper parameter free adaptive learning algorithm for Deep MLP. The proposed method tries to avoid the extensive search for finding the optimum learning rate hyper-parameter in MLP. Learning rate is composed of

a function of parameter α, which is termed as Learning Parameter. Learning parameter is updated in the same way as the weights of Deep MLP. In the proposed method, two functions, namely Sigmoid and Exponential, are considered as given below.

$$f_1\left(\alpha^{(l)}\right) = 1/\left(1 + exp\left(-\alpha^{(l)}\right)\right) \tag{1}$$

$$f_2\left(\alpha^{(l)}\right) = \exp\left(-\alpha^{(l)^2}\right) \tag{2}$$

Where $\alpha^{(l)}$ denotes the learning parameter for the layer l.

In order to determine the magnitude of change in weights corresponding to each neuron in each layer (except the input layer), the concept of Laplacian score has been used to update the learning rate. Laplacian score is used to measure the relevance of each neuron in each layer.

The output from each layer is obtained during the forward propagation for a mini-batch. For any particular layer, the Laplacian score of neurons is computed by using the output of those neurons for the instances in the mini-batch. Relevance of neurons depends on the Laplacian score of output, and the output depends on the incoming weights. Hence, high Laplacian score indicates that the incoming weights to the neuron are already trained to produce relevant output and any further modification might degrade their performance. To prevent the weights from changing, the learning rate for the incoming weights should be less for higher Laplacian score. For incorporating the effect of Laplacian score on learning rate, a factor of (1-Laplacian score) is included in the learning rate. Hence, for weights connecting neurons in layer $(l-1)$ to j^{th} neuron in layer l, the effective learning rate is given by

$$\eta_j^{(l)} = \left(1 - L_j^{(l)}\right)\ f\left(\alpha^{(l)}\right) \tag{3}$$

Where $L_j^{(l)}$ denotes the Laplacian score of the output from j^{th} neuron in layer l and $\alpha^{(l)}$ denotes the learning parameter for weight values connecting neurons in layer $(l-1)$ to neurons in layer l. f denotes one of the functions given by (1) and (2).

The concept of Dropout has been used to prevent overtraining. Evaluation of Laplacian Score is discussed in the following subsection.

3.1. Computation of Laplacian Score

Computation of Laplacian Score is described in details as follows.

Let there be 4 neurons in a layer and feed-forward step is carried out by using a mini-batch of size 3. Let the activation values at the neurons in a particular level be denoted by O_{ji} where j denotes the index of neuron and i denotes the index of instance in the minibatch. The activation values can be represented in matrix form as follows.

$$activation\ values = \begin{bmatrix} O_{11}O_{12}O_{13} \\ O_{21}O_{22}O_{23} \\ O_{31}O_{32}O_{33} \\ O_{41}O_{42}O_{43} \end{bmatrix}$$

Find the Laplacian score of neurons' output as follows.

Construct the Nearest Neighbor graph G with 3 nodes where i^{th} node corresponds to the i^{th} output instance (i.e., i^{th} column of the activation matrix)

$$G(i,j) = \begin{cases} 1 & \text{if } i^{th} \text{ sample} \in \text{Nearest neighbour of} j^{th} \text{ sample} \\ & OR\ j^{th} \text{ sample} \in \text{Nearest neighbour of } i^{th} \text{ sample} \\ 0 & otherwise \end{cases}$$

Using the Nearest Neighbor graph G, construct the weight matrix M as follows.

$$M(i,j) = \begin{cases} exp(-((O_{1i}-O_{1j})^2 + (O_{2i}-O_{2j})^2 \\ +(O_{3i}-O_{3j})^2 + (O_{4i}-O_{4j})^2)/\sigma) & if\ G(i,j) = 1 \\ 0 & if\ G(i,j) = 0 \end{cases}$$

Where σ is a constant.

For output from r^{th} neuron, define $O_r = [O_{r1}, O_{r2}, O_{r3}]^T$ where the superscript T denotes transpose.

Define $I = [1,\ 1,\ 1]^T$ and a diagonal matrix D such that $D(i,i) = \sum_i M(i,j)$.

Compute graph Laplacian matrix L as $L = D - M$.

Compute the transformed output as follows.

$$\widetilde{O}_r = O_r - \frac{O_r^T DI}{I^T DI} I$$

The Laplacian score of r^{th} neuron is computed as.

$$L_r = \frac{\widetilde{O}_r{}^T L \widetilde{O}_r}{\widetilde{O}_r{}^T D \widetilde{O}_r}$$

Normalize Laplacian scores to the range [0, 1] as

$$L_i = (L_i - min(L))/(max(L) - min(L))\ \forall\ i = 1,\ 2,\ 3,\ 4 \tag{3}$$

Importance of each neuron is established by the higher value of Laplacian score of its output.

3.2. Weight update rule

The forward pass equations during the training are given as follows. Input for every hidden layer and output layer is computed as

$$N_j^{(l)} = \sum_{i=1}^{S^{(l-1)}} W_{ij}^{(l)} \left(O_i^{(l-1)} d_i^{(l-1)}\right) + b_j^{(l)}\ \forall\ j = 1\ to\ S^{(l)} \tag{4}$$

Where,

$N_i^{(l)}$ - Input to the i^{th} neuron in layer l.

$S^{(l)}$ - Size of layer l.

$W_{ij}^{(l)}$ - Weight connecting i^{th} neuron in layer $(l-1)$ to j^{th} neuron in layer l.

$O_i^{(l)}$ - Output of i^{th} neuron in layer l. $O_i^{(0)}$ being same as x_i, the i^{th} input feature.

$d_i^{(l)}$ - Dropout mask of i^{th} neuron in layer l, being randomly generated according to the following rule.

$$d_j^{(l)} = Binomial(n = 1,\ p = 1 - dropout)\ \forall\ j = 1\ to\ S^{(l)}$$

For all hidden layers, Rectified Linear activation function is applied on $N_j^{(l)}$, given by (4), to obtain hidden activation which is given by.

$$O_j^{(l)} = max\left(0, N_j^{(l)}\right)\ \forall\ j = 1\ to\ S^{(l)} \tag{5}$$

Only for output layer, sigmoid activation function is used.

Cross entropy error is used and is given as follows.

$$E = -\sum_{j=1}^{S^{(H)}} \left(Y_j\ log\left(O_j^{(H)}\right) + \left(1 - Y_j\right)\ log\left(1 - O_j^{(H)}\right)\right) \tag{6}$$

Where Y denotes the class label and H denotes the total number of layers (excluding input layer). The neural network is trained to minimize the training error denoted by (6).

For every layer starting from output to first hidden layer, change in weight is given by

$$W_{ri}^{(l)} = W_{ri}^{(l)} - \eta_i^{(l)} \frac{\partial E}{\partial W_{ri}^{(l)}} \tag{7}$$

Where,

$$\frac{\partial E}{\partial W_{ri}^{(l)}} = -O_r^{(l-1)} d_r^{(l-1)} \delta_i^{(l)} \ \forall\ r = 1\ to\ S^{(l-1)},\ \ \forall\ i = 1\ to\ S^{(l)}$$

For all the layers except output layer, $\delta^{(l)}$ is given by

$$\delta_i^{(l)} = \begin{cases} d_i^{(H-l-1)} \sum_{j=1}^{S^{(H-l)}} W_{ij}^{(H-l)} \delta_j^{(l-1)} & if\ N_i^{(H-l-1)} > 0 \\ & \forall\ i = 1\ to\ S^{(H-l-1)} \\ 0 & otherwise \end{cases}$$

For output layer, $\delta^{(H)}$ is given by

$$\delta_j^{(H)} = \left(O_j^{(H)} - Y_j\right) \ \forall\ j = 1\ to\ S^{(H)}$$

Learning rate of the incoming weights for i^{th} neuron in layer l is given by.

$$\eta_i^{(l)} = \left(1 - L_i^{(l)}\right) f\left(\alpha^{(l)}\right) \ \forall\ i = 1\ to\ S^{(l)} \tag{8}$$

Where $\alpha^{(l)}$ denotes the Learning parameter for incoming weights to layer l, and $L_i^{(l)}$ is the Laplacian score of i^{th} neuron in layer l (See (3) in Subsection 3.1).

Weight update equation for bias is given by

$$b_j^{(l)} = b_j^{(l)} - \zeta_j^{(l)} \frac{\partial E}{\partial b_j^{(l)}} \tag{9}$$

Where,

$$\frac{\partial E}{\partial b_j^{(l)}} = -\delta_j^{(H-l+1)} \ \forall j = 1\ to\ S^{(l)}$$

Learning rate of the incoming bias to j^{th} neuron in layer l is given by.

$$\zeta_j^{(l)} = \left(1 - L_j^{(l)}\right) f\left(\beta^{(l)}\right) \ \forall j = 1\ to\ S^{(l)} \tag{10}$$

Where $\beta^{(l)}$ denotes the Learning parameter for incoming bias to layer l.

Since learning parameter is varying at each iteration, the iteration number t is also included in weight and bias update rules are given by (7) and (9), which are redefined as follows.

$$W_{ri}^{(l,\ t+1)} = W_{ri}^{(l,t)} - \eta_i^{(l,t)} \frac{\partial E^{(t)}}{\partial W_{ri}^{(l,t)}} \tag{11}$$

$$b_j^{(l,t+1)} = b_j^{(l,t)} - \zeta_j^{(l,t)} \frac{\partial E^{(t)}}{\partial b_j^{(l,t)}} \tag{12}$$

Using chain rule on (6), (8) and (11), the derivative of error with respect to learning parameters for weight in layer l is computed as follows.

$$\frac{\partial E^{(t+1)}}{\partial \alpha^{(l,t)}} = \sum_{ij} \left(\frac{\partial E^{(t+1)}}{\partial W_{ij}^{(l,t+1)}}\right)\left(\frac{\partial W_{ij}^{(l,t+1)}}{\partial \eta_j^{(l,t)}}\right)\left(\frac{\partial \eta_j^{(l,t)}}{\partial \alpha^{(l,t)}}\right)$$

$$\Rightarrow \frac{\partial E^{(t+1)}}{\partial \alpha^{(l,t)}} = \frac{\partial f\left(\alpha^{(l,t)}\right)}{\partial \alpha^{(l,t)}} \sum_{ij} \left(1 - L_i^{(l,t)}\right)\left(\frac{\partial E^{(t+1)}}{\partial W_{ij}^{(l,t+1)}}\right)\left(\frac{\partial E^{(t)}}{\partial W_{ij}^{(l,t)}}\right)$$

$$\alpha^{(l,t+1)} = \alpha^{(l,t)} - \frac{\partial E^{(t+1)}}{\partial \alpha^{(l,t)}} \tag{13}$$

Similarly, using (6), (10) and (12), the updated equation for learning parameter β for bias is given below.

$$\frac{\partial E^{(t+1)}}{\partial \beta^{(l,t)}} = f'\left(\beta^{(l,t)}\right) \sum_{ij} \left(1 - L_i^{(l,t)}\right)\left(\frac{\partial E^{(t+1)}}{\partial b_j^{(l,t+1)}}\right)\left(\frac{\partial E^{(t)}}{\partial b_j^{(l,t)}}\right)$$

$$\beta^{(l,t+1)} = \beta^{(l,t)} - \frac{\partial E^{(t+1)}}{\partial \beta^{(l,t)}} \tag{14}$$

The learning parameters α and β are updated by using (13) and (14). Learning rates are computed on the basis of updated values of learning parameters by using (8) and (10). Finally, weights and bias are updated by using new values of learning rates.

Forward pass equations during the testing phase are same as the equations used during training phase except that the dropout mask is not used during training, and the output of each hidden layer is multiplied by the term $(1 - dropout)$.

For each layer, the incoming input is given as

$$N_j^{(l)} = \sum_{i=1}^{S^{(l-1)}} W_{ij}^{(l)} \left((1 - dropout) O_i^{(l-1)}\right) + b_j^{(l)} \ \forall\ j = 1\ to\ S^{(l)}$$

Rectified linear activation function, as given by (5), is used for the hidden layers, and sigmoid activation function is used for the output layer.

The final prediction of Deep MLP is given by

$$predicted\ class = \underset{j}{\operatorname{argmax}}\, O_j^{(H)}$$

3.3. Algorithm

The Complete Algorithm for proposed method is given below.

Input: Training and testing datasets. Initial random values of weights and bias.
For each mini-batch in training dataset. // Training
  For each layer l from 1 to H //Forward pass
    For each neuron j from 1 to $S^{(l)}$
      Compute input $N_j^{(l)}$ and output $O_j^{(l)}$.
      Compute Laplacian score $L_j^{(l)}$.
  End
End
For each layer l from H to 1 // Backward pass
  For j from 1 to $S^{(l)}$ // gradient computation
    For i from 1 to $S^{(l-1)}$
      Compute error gradient $\frac{\partial E}{\partial W_{ij}^{(l)}}$
    End
    Compute error gradient $\frac{\partial E}{\partial b_j^{(l)}}$
  End
  Update the learning parameters $\alpha^{(l)}$ and $\beta^{(l)}$.
  // update learning parameters
  For each neuron j from 1 to $S^{(l)}$ // compute learning rates
  Compute learning rates $\eta_j^{(l)}$ and $\zeta_j^{(l)}$.
  End
  For j from 1 to $S^{(l)}$ // update weights and bias
    For i from 1 to $S^{(l-1)}$
    Update the weight $W_{ij}^{(l)}$ using learning rate $\eta_j^{(l)}$
    End
    Update the bias $b_j^{(l)}$ using learning rate $\zeta_j^{(l)}$
  End
 End
End
For each sample s in testing dataset. // Testing
 For each layer l from 1 to H // forward pass
  For each neuron j from 1 to $S^{(l)}$
   Compute input $N_j^{(l)}$ and output $O_j^{(l)}$.
  End
End
Compute misclassification error as

$$E(s) = \begin{cases} 0 & \text{if } argmax_j\ O_j^{(H)} = argmax_j\ Y_j \\ 1 & \text{otherwise} \end{cases}$$

End
Compute misclassification error percentage as
 $E_{percentage} = 100 * \sum_s E(s)$ / Total number of samples in testing dataset
Return $E_{percentage}$

Table 1
Description of datasets.

Dataset	Training instances	Validation instances	Testing instance	Classes	Features
MNIST	50,000	10,000	10,000	10	784
Basic	10,000	2000	50,000	10	784
Rotation	10,000	2000	50,000	10	784
Background Random	10,000	2000	50,000	10	784
Background Images	10,000	2000	50,000	10	784
Rectangles	1000	200	50,000	2	784
Rectangles Images	10,000	2000	50,000	2	784
Convex	6000	2000	50,000	2	784

Table 2
Classification error (in percentage) and run time (in seconds) for rectified linear.

	Exponential		Sigmoid		Standard deep MLP	
Datasets	LPLS	LP	LPLS	LP	Optimum	ADADELTA
Background images	18.62 (7798)	19.46 (7273)	20.51 (8949)	20.71 (7227)	21.92 –	21.1 (13,525)
Basic	2.31 (7739)	2.4 (7144)	2.5 (7882)	2.66 (6413)	2.63 –	2.63 (10,557)
Rotation	7.61 (7843)	7.98 (7184)	8.29 (7035)	8.53 (6936)	8.68 –	8.17 (12,456)
Rectangles images	21.2 (7681)	22.5 (6703)	22.3 (8740)	23 (7554)	23.4 –	23.1 (11,087)
Rectangles	1.32 (816)	1.6 (719)	2.09 (843)	2.17 (714)	3.91 –	1.64 (1002)
Background random	10.81 (7749)	10.5 (6514)	10.47 (8913)	11.32 (7658)	11.75 –	15.2 (14,406)
Convex	19.58 (4881)	21.1 (3995)	20.11 (4851)	21.06 (3977)	22.7 –	22.4 (6505)
MNIST	1.08 (16,909)	1.09 (16,620)	1.08 (16,584)	1.06 (16,453)	1.09 –	1.16 (46,348)

4. Results

The performance of the proposed method has been compared with the standard Deep Multilayer Perceptron on several benchmark datasets (Larochelle, Erhan, Courville, Bergstra, & Bengio, 2007; LeCun, Bottou, Bengio, & Haffner, 1998). Description of the datasets is given in Table 1.

4.1. Implementation details

The results have been obtained by using Deep MLP with 3 hidden layers and 1000 neurons in each hidden layer. The mini-batch size is taken as 20 and the MLP is trained for 500 epochs by minimizing cross entropy error function using back propagation. Dropout is used to prevent the network from overtraining. Two different activation functions namely Rectified Linear and Max-out have been considered for hidden layer. The output units are composed of sigmoid activation function. Dropout probability is taken as hyper-parameter, and the optimum dropout probability is chosen from the set {0.1, 0.2, 0.3, 0.4, 0.5} on the basis of minimum validation error. In Deep MLP, it has been observed that the magnitude of back propagated error gradient is different in different layers. Hence, different learning parameters are taken for different layers which are updated on the basis of the steepest descent rule. Initial values of learning parameters α, β are taken as 1 for all the layers.

Two sets of results are obtained for the proposed method, first by using only learning parameter without any Laplacian score (LP) and second by using both learning parameter and the Laplacian score (LPLS). For both LP and LPLS, results are obtained for the two functions given by (1) and (2). Table 2 shows the results for various benchmark datasets when Rectified Linear is used as activation function in hidden layers and Table 3 shows the results when Max-out is used in hidden layers. The runtime is given in seconds (in parenthesis) in Tables 2 and 3. The results using (1) are mentioned under the heading 'Sigmoid', and the results for (2) are mentioned under 'Exponential'.

The proposed method is compared with two methods. In the first method, learning rate is optimized by searching over several values and choosing the learning rate that has the least validation error. The results are mentioned under the heading 'Optimum'. In the second method, adaptive learning rate ADADELTA is used. The hyper-parameter values for ADADELTA are taken to be $\epsilon = 1e-6$ and $\rho = 0.95$. Network configurations have been kept the same for the proposed method, and optimum dropout probability is chosen from the set {0.1, 0.2, 0.3, 0.4, 0.5}. The results for 'Optimum' are obtained by optimizing the learning rate hyper-parameter over the set {0.5, 0.05, 0.005, 0.0005}. For 'Optimum', learning rate is slowly decreased with the number of epochs by the factor $1/(1+\exp(currentepoch/100))$. This was executed for the initial values {0.5, 0.05, 0.005, 0.0005}. Time factor for 'Optimum' has not been provided since additional search time is involved for finding the optimum value of learning-rate. The classification error is also computed by using the Probabilistic Expectation Back propagation with Real weights (R-EBP-P) (Cheng, Soudry, Mao, & Lan, 2015). The number of hidden neurons is taken as 1000 and three hidden layers are considered. The optimum dropout probability is chosen from the set {0.1, 0.2, 0.3, 0.4, 0.5}, and training is performed for 500 epochs by using batch size of 20. The result of R-EBP-P for various benchmark datasets is shown in Table 4.

From Table 2, it can be observed that Sigmoid LP performs better than 'Optimum' on seven datasets and better than ADADELTA on five datasets. Exponential LP performs better than 'Optimum' on seven datasets and outperforms ADADELTA on all the datasets. This demonstrates the superiority of the proposed method for updating the learning parameter on the basis of the error derivative. For both Sigmoid and Exponential learning rate functions, the misclassification error is further decreased when the Laplacian score is

Table 3
Classification error (in percentage) and run time (in seconds) for max-out.

Datasets	Exponential		Sigmoid		Standarddeep MLP	
	LPLS	LP	LPLS	LP	Optimum	ADADELTA
Background images	20.73 (16,957)	21.39 (15,839)	21.42 (16,728)	21.50 (16,037)	24.76 –	23.62 (17,129)
Basic	3.42 (16,806)	3.45 (15,428)	3.52 (16,937)	3.61 (16,281)	5.70 –	4.05 (16,858)
Rotation	8.81 (16,384)	9.46 (15,372)	9.13 (15,948)	9.57 (15,198)	11.03 –	11.37 (17,037)
Rectangles images	24.6 (16,583)	24.82 (15,794)	25.03 (16,382)	25.89 (15,839)	27.15 –	28.2 (17,439)
Rectangles	1.54 (1536)	1.60 (1518)	1.58 (1523)	1.72 (1502)	2.81 –	1.83 (1647)
Background random	12.63 (16,768)	14.8 (15,848)	13.72 (16,829)	15.29 (15,631)	21.75 –	18.57 (17,693)
Convex	23.37 (10,483)	23.79 (9733)	25.32 (10,372)	25.71 (9829)	26.1 –	28.49 (11,753)
MNIST	1.25 (30,462)	1.29 (29,583)	1.31 (29,744)	1.34 (29,894)	1.37 –	1.46 (32,948)

Table 4
Classification error (in percentage) and run time (in seconds) for R-EBP-P.

Datasets	R-EBP-P
Background images	32.85 (27,483)
Basic	9.13 (26,938)
Rotation	15.38 (26,483)
Rectangles images	29.3 (26,931)
Rectangles	3.72 (2435)
Background random	25.73 (27,194)
Convex	36.48 (12,748)
MNIST	1.74 (56,181)

also incorporated in the learning rate computation. Sigmoid LPLS performs better than 'Optimum' on all the datasets, and performs better than ADADELTA on six datasets while Exponential LPLS outperforms both 'Optimum' and ADADELTA. From Table 3, it can be observed that both Sigmoid LP and Exponential LP outperform 'Optimum' as well as ADADELTA for each of the datasets. The misclassification error that is obtained by using LPLS is less than that of LP for both Exponential and Sigmoid. From Tables 2, 3 and 4, it can be observed that the proposed method outperforms R-EBP-P for all the datasets. On comparing the results in Tables 2 and 3, it can be found that the misclassification error is less when Rectified Linear activation function is used as compared to Max-out for all the methods and for each of the datasets. Hence, the proposed method for including Laplacian Score in the learning rate and updating the learning parameter is a promising concept which gives a suggestive mechanism for updating weights.

It can also be observed that with Exponential learning rate and using Rectified Linear activation function, LPLS performs better than all other methods for every dataset.

Time factor has been compared with ADADELTA since 'Optimum' needs additional search time as explained before. The ratio of runtimes of ADADELTA and runtimes of proposed methods lie in the range 1.1 to 2.8 for various datasets. Both LP and LPLS take less time compared to the adaptive learning rate ADADELTA. This is due to the fact that for updating the learning rate by using ADADELTA, several matrix operations are performed which leads to significant increase in runtime.

5. Conclusion

Deep Learning with adaptive learning rate has been attempted to enhance the classification power of Deep Learning since the choice of learning rate becomes a bottleneck in using MLP for Deep Learning. In contrast to the existing methods for updation of learning rate, the proposed approach uses an additional concept viz Laplacian score for understanding the significance of various neurons. It gives a measure of importance for each of the neurons which helps in improving the weight updation. Learning rate is taken as a function of a parameter and is adapted by using the error gradient as well as in combination with Laplacian score of output of the neurons for each mini-batch. Recently developed Rectified Linear activation function and Max-out have been used. The results illustrate the superiority of this approach over the widely known adaptive learning rate method ADADLTA and over the Deep Network where optimum learning rate is searched as a hyper-parameter.

The proposed method has a limitation that LPLS cannot be used in online mode since several samples are required for computing Laplacian score. On the other hand, LP can be used in online mode. Hence, the managerial insights based on the experimental outcomes would be to use 'Exponential LP' with Rectified Linear activation function when the data is available in streams. 'Exponential LPLS' with Rectified Linear activation function should be preferred for the datasets which are available in batches.

Future research suggestions – In the case of medical diagnosis, the classification of diseases is a difficult task since several symptoms may be common for various diseases. There may be fuzziness involved, therefore the use of fuzzy logic may be appropriate for diagnosis of deadly diseases like cancer by using deep learning. Hence, fuzzy concept can be incorporated in the proposed method.

Conflict of interest

The authors declare that there is no conflict of interest regarding the publication of this work.

Acknowledgements

The Council of Scientific and Industrial research (CSIR), India, is thankfully acknowledged for providing Research Fellowship to Rajesh Kumar Sharma.

References

Cheng, Z., Soudry, D., Mao, Z., & Lan, Z. (2015). Study on binary multilayer neural networks with ebp algorithm on image classification. arXiv preprint arXiv:1503.03562.

Dahl, G. E., Sainath, T. N., & Hinton, G. E. (2013, May). Improving deep neural networks for LVCSR using rectified linear units and dropout. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on* (pp. 8609–8613). IEEE.

Dase, R. K., & Pawar, D. D. (2010). Application of artificial neural network for stock market predictions: a review of literature. *International Journal of Machine Intelligence, 2*(2), 14–17.

de Oliveira, F. A., Nobre, C. N., & Zarate, L. E. (2013). Applying artificial neural networks to prediction of stock price and improvement of the directional prediction index–case study of petr4, Petrobras, Brazil. *Expert Systems with Applications, 40*(18), 7596–7606.

Ghiassi, M., Lio, D., & Moon, B. (2015). Pre-production forecasting of movie revenues with a dynamic artificial neural network. *Expert Systems with Applications, 42*(6), 3176–3193.

Glorot, X., Bordes, A., & Bengio, Y. (2011). Deep sparse rectifier neural networks. *International Conference on Artificial Intelligence and Statistics*, 315–323.

Goodfellow, I., Warde-Farley, D., Mirza, M., Courville, A., & Bengio, Y. (2013). Maxout networks. In *Proceedings of The 30th International Conference on Machine Learning* (pp. 1319–1327).

Hadavandi, E., Shavandi, H., & Ghanbari, A. (2010). Integration of genetic fuzzy systems and artificial neural networks for stock price forecasting. *Knowledge-Based Systems, 23*(8), 800–808.

He, X., Cai, D., & Niyogi, P. (2005). Laplacian score for feature selection. In *Advances in neural information processing systems* (pp. 507–514).

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

Hinton, G.E., Srivastava, N., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R.R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*.

Hochreiter, S., Bengio, Y., Frasconi, P., & Schmidhuber, J. (2001). Gradient flow in recurrent nets: the difficulty of learning long-term dependencies.

Iturriaga, F. J. L., & Sanz, I. P. (2015). Bankruptcy visualization and prediction using neural networks: a study of US commercial banks. *Expert Systems with Applications, 42*(6), 2857–2869.

Kara, Y., Boyacioglu, M. A., & Baykan, Ö. K. (2011). Predicting direction of stock price index movement using artificial neural networks and support vector machines: the sample of the istanbul stock exchange. *Expert systems with Applications, 38*(5), 5311–5319.

Larochelle, H., Erhan, D., Courville, A., Bergstra, J., & Bengio, Y. (2007, June). An empirical evaluation of deep architectures on problems with many factors of variation. In *Proceedings of the 24th international conference on Machine learning* (pp. 473–480). ACM.

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE, 86*(11), 2278–2324.

Lee, S., & Choeh, J. Y. (2014). Predicting the helpfulness of online reviews using multilayer perceptron neural networks. *Expert Systems with Applications, 41*(6), 3041–3046.

Maclaurin, D., Duvenaud, D., & Adams, R.P. (2015). Gradient-based Hyperparameter Optimization through Reversible Learning. *arXiv preprint arXiv:1502.03492*.

Moosmayer, D. C., Chong, A. Y. L., Liu, M. J., & Schuppar, B. (2013). A neural network approach to predicting price negotiation outcomes in business-to-business contexts. *Expert Systems with Applications, 40*(8), 3028–3035.

Pan, C. Z., Lai, X. Z., Yang, S. X., & Wu, M. (2013). An efficient neural network approach to tracking control of an autonomous surface vehicle with unknown dynamics. *Expert Systems with Applications, 40*(5), 1629–1635.

Patidar, R., & Sharma, L. (2011). Credit card fraud detection using neural network. *International Journal of Soft Computing and Engineering (IJSCE), 1*, 32–38.

Raj, S., & Portia, A. A. (2011, March). Analysis on credit card fraud detection methods. In *Computer, Communication and Electrical Technology (ICCCET), 2011 International Conference on* (pp. 152–156). IEEE.

Rifai, S., Vincent, P., Muller, X., Glorot, X., & Bengio, Y. (2011). Contractive auto-encoders: Explicit invariance during feature extraction. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)* (pp. 833–840).

Soudry, D., Hubara, I., & Meir, R. (2014). Expectation backpropagation: Parameter-free training of multilayer neural networks with continuous or discrete weights. In *Advances in Neural Information Processing Systems* (pp. 963–971).

Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., & Manzagol, P. A. (2010). Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *The Journal of Machine Learning Research, 11*, 3371–3408.

Zeiler, M.D. (2012). ADADELTA: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*.