# LALR: Theoretical and Experimental validation of Lipschitz Adaptive Learning Rate in Regression and Neural Networks


1st Snehanshu Saha
*CSIS and APPCAIR*
*BITS Pilani K K Birla Goa Campus*
Goa, India
snehanshu.saha@ieee.org

2nd Tejas Prashanth
*Department of Computer Science*
*PES University*
Bangalore, India
tejuprash@gmail.com

3rd Suraj Aralihalli
*Department of Computer Science*
*PES University*
Bangalore, India
suraj.ara16@gmail.com

4th Sumedh Basarkod
*Department of Computer Science*
*PES University*
Bangalore, India
sumedhpb8@gmail.com

5th T.S.B. Sudarshan
*Department of Computer Science*
*PES University*
Bangalore, India
sudarshan@pes.edu

6th Soma S Dhavala
*Founder*
*ML Square*
Bangalore, India
soma@mlsquare.org



***Abstract*—We propose a theoretical framework for an adaptive learning rate policy for the Mean Absolute Error loss function and Quantile loss function and evaluate its effectiveness for regression tasks. The framework is based on the theory of Lipschitz continuity, specifically utilizing the relationship between learning rate and Lipschitz constant of the loss function. Based on experimentation, we have found that the adaptive learning rate policy enables up to 20x faster convergence compared to a constant learning rate policy.**



***Index Terms*—Adaptive learning rate, Lipschitz constant, Mean Absolute Error**


## I. Introduction

Gradient descent based optimization algorithms, such as Stochastic gradient descent, Adam [1], have been widely used in the field of deep learning. Gradient descent based learning involves updation of weights of a neural network by back propagation of gradients in order to lower the error [2]. In addition, the training process is a challenging task due to the large number of hyper parameters that require tuning. Among the various hyper-parameters, learning rate is a key factor that influences the speed of convergence of a neural network. Large values of learning rate can hinder convergence and instead may lead to the divergence of the optimization algorithm [3]. Conventionally, learning rate is manually chosen in order to control the rate of convergence. In addition, methods have been developed in order to decay the learning rate over time or use a non-monotonic learning rate scheduler in order for the optimization algorithm to converge faster [4]. However, making the learning rate adaptive, is an ongoing research field.

Neural networks are widely used for a variety of classification and regression based tasks. Regression is implemented in neural networks as a supervised learning problem and is capable of handling complex non-linear dependencies [5]. A multitude of factors have contributed to the phenomenal success of (deep) neural networks (DNNs). They include, but are not limited to ease of access to big data sets, affordable computing, plug-and-play deep learning frameworks and auto differentiation frameworks. The Back propagation algorithm and auto differentiation frameworks have greatly simplified the process of fitting DNNs – it is no longer necessary to write inference software from scratch. Instead, a practitioner or an analyst can simply specify the model, and call the supplied optimization technique, thus abstracting the complexity of the process. While the success of deep neural networks in Computer Vision and Natural Language Processing is transformational, DNNs still suffer from a variety of problems. Some of the criticisms against DNNs are that they can make confident mistakes, due to their opaque nature with respect to the explainability [6], [7]. Making DNNs both reliable and explainable are the new research frontiers. In this paper, we argue that Quantile Regression (QR) can be used to provide prediction intervals, which is one of the ways of quantifying uncertainty [8]. In the context of DNNs, QR models can be fit by minimizing the Check Loss function. Moreover, the number of independent regression models that are fit is equivalent to the number of quantiles desired. While one may look for a more efficient architecture to fit all the quantiles simultaneously, fitting multiple independent regression models is a simple alternative. If one can speed-up convergence, then the process of fitting all quantiles independently can be done in nearly the same amount of time that a typical regression model takes to provide point estimates. In this regard, we investigate Adaptive Learning Rates for Check Loss, whose special case is the Mean Absolute Error that estimates conditional medians.

The paper is organized in the following manner. Section


978-1-7281-6926-2/20/$31.00 ©2020 IEEE

II provides the related work in the field of convergence of gradient descent. Section III describes the motivation and the contributions of the paper. Section IV provides a theoretical background on Lipschitz continuity and its applications in neural networks. Section V provides details on the derivation of Lipschitz constant for Mean Absolute Error and Quantile loss. Section VI and VII provide the experimental results for the theoretical framework. Section VIII provides a conclusion and future areas of research for this work.

## II. Related work

Various approaches have been attempted in order to obtain faster convergence. Many of these approaches involve a theoretical study of gradient descent based optimization algorithms. For example, Hardt et. al. [9] worked on the theoretical proof for stability of Stochastic Gradient Descent(SGD). The relationship between generalisation error and stability was identified and a stability measure was defined. Similarly, Kuzborskij et. al. [10] studied the data dependent stability of stochastic gradient descent. The data-dependent notion of algorithmic stability was established and used to employ generalisation bounds. In addition, generalisation bounds were computed based on data-dependent distribution and initialisation of SGD. Development of novel optimization algorithms is another type of solution worthy of mention. For example, Adam [1] optimization algorithm enabled deep learning models to achieve faster convergence. The core idea behind Adam was to combine the usage of momentum-based gradient descent and RMSProp [11].

## III. Motivation & contribution

Statistical Software such as `SAS`,`STATA`, and many libraries in `R`, provide uncertainty estimates, along with predictions, in terms of standard errors and p-values. Bayesian counterparts produce credible intervals. The classical Frequentist estimation techniques either rely on second-order optimization techniques to produce confidence intervals or rely on special modeling assumptions. The Bayesian analogues require access to efficient, cheap, posterior samples, based on which any functional of the random variables can be computed. The statistics community lays enormous importance on producing such inference summaries. Unfortunately, Machine Learning tools rarely provide such reports, as they are primarily concerned with prediction tasks alone. In the context of DNNs, the problem is more pronounced, as the current Deep Learning landscape is still evolving with respect to providing such uncertainty estimates. For instance, L-BFGS technique is still experimental in `pytorch`. Developing generic, rich inference techniques is one way to make Deep Learning credible. Another avenue is to consider uncertainty quantification as primary inference goal problem that can work with the almost ubiquitous Back Propagation and Stochastic Gradient Descent. One such method is Quantile Regression(QR). QR is well-known in the field of econometrics, and has been introduced to the ML community fairly recently [8]. The benefit of QR applied in the context of DNNs is that no new inferential algorithms are required to fit them – one only needs Check Loss, also called as Quantile loss. It is desirable to look for methods to accelerate the convergence by exploiting the Check loss function structure, which can support the workhorse inference techniques.

We propose an adaptive learning rate scheme for training neural networks with Mean Absolute Error(MAE) and Check loss as the loss functions. Noting that MAE is a special case of Check loss, an adaptive learning rate scheme for Check loss is also proposed. Unlike mean squared error(MSE), MAE as a loss function is robust to outliers since it relies on the absolute value of errors instead of the square of error [12]. In addition, since MAE is primarily applied for regression tasks, the adaptive learning rate is derived for regression based problems in neural networks. We contribute to the following problems

1) Theoretical framework for computation of Lipschitz adaptive learning rate (LALR) for MAE loss function in single and multi label multivariate regression models, including Quantile regression
2) Compute LALR for Check loss function using neural networks
3) Evaluate the effectiveness of the framework against regression based data sets

## IV. Theoretical background

### A. Notation

We use the following notation

1) m indicates the batch size and n indicates the number of predicted output values in multivariate regression.
2) $(x^i, y^i)$ refers to a single training example. $(x^i_j, y^i_j)$ refers to a particular output value for a single training example.
3) A superscript of $l$, such as $a^{[l]}$, denotes the layer number and a superscript of L denotes the last layer, unless specified otherwise. For instance, $y^{[L]}$ denotes the actual output of the last layer according to the training data.
4) $a^{[l]}$ is used to represent the activation at a particular layer, with a subscript indicating the activation of a particular neuron in that layer. For example, $a^{[l]}_j$ represents the activation value of the $j^{th}$ neuron in layer $l$. The same notation is followed for $y^{[l]}$, which indicates the correct output at a particular layer.
5) $W^{[l]}_{ij}$ denotes the weights from neuron $i$ in layer $l-1$ and neuron $j$ in layer $l$.

### B. Lipschitz continuity

A function $f(x)$ is said to be Lipschitz continuous in its domain if there exists a constant $k$ in its domain such that for every pair of points, the absolute value of the slope between those points is not greater than $k$. The minimum value of $k$ is known as the Lipschitz constant. Mathematically, the Lipschitz constant for a function $f$,that depends on $x$,is expressed as $\|f(x_1) - f(x_2)\| \leq k \|x_1 - x_2\|$ where $k$ is the Lipschitz constant. Since MAE is Lipschitz continuous in its domain, there exists a Lipschitz constant $k$ in its domain. Since the

mean value theorem holds good, the supremum of the gradient, sup $\|\nabla f(x)\|$, exists and the supremum of the gradient is one such Lipschitz constant.

By computing the Lipschitz constant of the loss function, $max\,\|\nabla_w f\|$, one can constrain the change in weights in the weight update rule to $\triangle w \leq 1$ by setting the learning rate to be equal to the reciprocal of the Lipschitz constant.

$$\mathbf{w} = \mathbf{w} - \eta.\nabla_w f$$

where $\eta = \frac{1}{max\|\nabla_w f\|}$. This particular choice of LALR, under the assumption that gradients cannot change arbitrarily fast, ensures a convex quadratic upper bound, minimized by the descent step. It is fairly straightforward to show (via Taylor series expansion of $f$), $f \in C^2$ that $f(w^{k+1}) \leq f(w^k) - \frac{1}{2L}\left\|\nabla f(w^k)\right\|^2$. This implies that Gradient descent decreases $f$ if $\eta = 1/L$ where $L$ is the Lipschitz constant.

### C. Lipschitz constant in neural network

In a neural network, the gradients are smaller in the earlier layers than in the last layer. Consequently, the following relation holds true [9],

$$\max_{ij}\left\|\frac{\partial E}{\partial w_{ij}^{[L]}}\right\| \geq \left\|\frac{\partial E}{\partial w_{ij}^{[l]}}\right\| \; \forall l, i, j$$

Hence, the maximum value of the gradient in the neural network can be found using the maximum value of gradient in the last layer.

### D. Quantile regression

The most common loss function used in regression is the Mean Squared Error(MSE). It can be shown that, minimizing MSE is equivalent is maximizing the log-likelihood under the Gaussian noise assumption i.e. $y_i = f(x_i) + \epsilon_i; \epsilon_i = N(0, \sigma^2)$. Consequently, we get $E[y|x] = f(x)$, where $E[.]$ is the expectation operator. It means that, the minimization of MSE leads to the conditional mean. However, it is known that mean, as a measure of location, is not robust to outliers, due to which median is preferred in such cases. It is also known that, median is the minimization of MAE. In addition, generalization of the MAE is the Check loss, which is given by: $L_\tau(e) = (\tau - I(e < 0))e$. Similar to the way in which MSE is shown to maximize the likelihood under Gaussian noise assumption, Check loss can be shown to maximize the log-likelihood under Asymmetric Laplace noise(ALD) assumption: $y_i = f(x_i) + \epsilon; \epsilon = ALD(0, 1, \tau)$ & $ALD(y; \mu, \sigma, \tau) \equiv \frac{\tau(1-\tau)}{\sigma}\exp(-\rho_\tau(\frac{y-\mu}{\sigma}))$. Thereafter, it can be shown that $P(y \leq \mu) = \tau$ so that the predicted value can be interpreted as the corresponding conditional quantile. By fitting multiple quantiles, a prediction interval of required coverage can be constructed.

## V. Mathematical derivation

### A. Multiple regression using neural networks

The following section provides the derivation for the Lipschitz constant for the Mean Absolute Error as the loss function. Assuming one output variable, MAE is given by $E(a^{[L]}, y) = \frac{1}{m}\sum_{i=1}^{m}|a^{(i)[L]} - y^{(i)}|$ where $m$ is the batch size.

Consider a subset of a batch of $m$ training examples $(\mathbf{x}^{(i)}, y^{(i)})$, say $m_1$, which represents the training examples for which $a^{(i)[L]} > y^{(i)}$. Similarly, let $m_2$ be the training examples in the batch for which $a^{(i)[L]} < y^{(i)}$.

$$E(a^{[L]}, y) = \frac{1}{m}\sum_{\substack{i=1 \\ (\mathbf{x}^{(i)}, y^{(i)}) \in m_1}}^{m}\left(a^{(i)[L]} - y^{(i)}\right) + \frac{1}{m}\sum_{\substack{i=1 \\ (\mathbf{x}^{(i)}, y^{(i)}) \in m_2}}^{m}\left(y^{(i)} - a^{(i)[L]}\right)$$

Let $a^{[L]}$ and $b^{[L]}$ be two sets of predicted values for output for two different sets of weight matrices of the neural network. Then, $E(a^L, y) - E(b^L, y) = \frac{1}{m}\sum_{i=1}^{m}|a^{(i)[L]} - y^{(i)}| - |b^{(i)[L]} - y^{(i)}|$ . The equation can be elaborated by considering four different cases based on the values of $\boldsymbol{a^L - y}$ and $\boldsymbol{b^L - y}$ . Representing the equation in the form of vectors of dimensions $m \times 1$, each case represents a subset of values, with only those components of the vectors activated that match each case. We simplify the equation as follows[1]:

Case 1: $(x_i, y_i) \in (m_1, n_1)$ that satisfy the conditions $\boldsymbol{(a^L - y)_{m_1,n_1} > 0}$ and $\boldsymbol{(b^L - y)_{m_1,n_1} > 0}$

$$\frac{1}{m}((\boldsymbol{a^L - y}) - (\boldsymbol{b^L - y}))_{m1,n1} = \frac{1}{m}(\boldsymbol{a^L - b^L})_{m1,n1} \quad (1)$$

Similarly, the equation is simplified for the remaining cases and an inequality expression is established, as done in Case 1. Since $\boldsymbol{a^L}$ and $\boldsymbol{b^L}$ are mutually exclusive among the four cases, adding equations for each of the four cases yields the original $\boldsymbol{a^L}$ and $\boldsymbol{b^L}$. Hence,

$$\boldsymbol{E(a^L) - E(b^L)} \leq \frac{1}{m}\boldsymbol{(a^L - b^L)} \quad (2)$$

After applying L1-norm,

$$\frac{||\boldsymbol{E(a^L) - E(b^l)}||}{||\boldsymbol{(a^L - b^L)}||} \leq \frac{1}{m} \quad (3)$$

Considering the following backpropagation equation,

$$\max_{ij}|\frac{\partial E}{\partial w_{ij}^{[L]}}| \leq \max_{ij}|\frac{\partial E}{\partial a_j^{[L]}}|.\max_{ij}|\frac{\partial a_j^{[L]}}{\partial z_j^{[L]}}|.\max_{ij}|\frac{\partial z_j^{[L]}}{\partial w_{ij}^{[L]}}|$$

$$\max_{ij}|\frac{\partial E}{\partial w_{ij}^{[L]}}| \leq \max_{ij}|\frac{\partial E}{\partial a_j^{[L]}}|.\max_{ij}|\frac{\partial a_j^{[L]}}{\partial z_j^{[L]}}|.\max_{j}|a_j^{[L-1]}|$$

$\max_{ij}|\frac{\partial a_j^{[L]}}{\partial z_j^{[L]}}|$ can be considered to be 1 if the final layer activation function is ReLU for the regression model. Let

[1]Note: Although MAE is not twice differentiable, the functions obtained in each of the four cases are twice differentiable, $f \in C^2$, thus abiding by the assumption behind the proof described in Section IVB

$K_z$=$\max_j |a_j^{[L-1]}|$; then, $\max_{ij} |\frac{\partial E}{\partial w_{ij}^{[L]}}| \leq \frac{K_z}{m}$. Hence, the Lipschitz constant is equal to

$$\frac{K_z}{m} \tag{4}$$

### *B. Multi-label Multivariate regression using neural networks*

The following section provides the derivation for the Lipschitz constant for the Mean Absolute Error as the loss function for multivariate regression. The MAE is: $E(a^L, y) = \frac{1}{mn} \sum_{i=1}^{m} \sum_{j=1}^{n} |a_j^{(i)[L]} - y_j^{(i)}|$, where $m$ is the batch size and $n$ is the number of labels.

Consider $\boldsymbol{a^L}$, the predicted output of the neural network for multivariate regression, to be a flattened form of the conventional matrix of size m $\times$ n. Let the output be of size $(m \cdot n) \times 1$. Consider **y** to be a flattened vector of size $(m \cdot n) \times 1$ . Consider a subset of a batch of $m$ training examples $(x_j^{(i)}, y_j^{(i)})$, say $mn_1$, which represents the corresponding output units for the training examples for which $a^{(i)[L]} > y^{(i)}$. Similarly, let $mn_2$ be the corresponding output units in the batch for which $a^{(i)[L]} < y^{(i)}$.

$$E(a^{[L]}, y) = \frac{1}{mn} \sum_{\substack{i=1 \\ (x_j^{(i)}, y_j^{(i)}) \in mn_1}}^{mn} \left(a^{(i)[L]} - y^{(i)}\right) + \frac{1}{mn} \sum_{\substack{i=1 \\ (x_j^{(i)}, y_j^{(i)}) \in mn_2}}^{mn} \left(y^{(i)} - a^{(i)[L]}\right)$$

Let $a^{[L]}$ and $b^{[L]}$ be two sets of predicted values for output for two different sets of weight matrices of the neural network. Then, $E(a^L, y) - E(b^L, y) = \frac{1}{mn} \sum_{i=1}^{mn} |a^{(i)[L]} - y^{(i)}| - |b^{(i)[L]} - y^{(i)}|$. The proof is carried out in the same manner, as described in Section VA. The Lipschitz constant is obtained to be

$$\frac{K_z}{mn} \tag{5}$$

### *C. Check loss using neural networks*

$$\rho_\tau(x) = \begin{cases} x\tau & \text{if } x \geq 0 \\ -x(1-\tau), & \text{otherwise} \end{cases}$$

Without loss of generality, assume that $x_1 < x_2$

Case-1: $0 < x_1 < x_2$

$$\implies \frac{|\rho_\tau(x_2) - \rho_\tau(x_1)|}{|x_2 - x_1|} \leq \tau$$

Case-2: $x_1 < 0 < x_2$

$$\implies \frac{|\rho_\tau(x_2) - \rho_\tau(x_1)|}{|x_2 - x_1|} \leq \tau$$

Case-3: $x_1 < x_2 < 0$

$$\implies \frac{|\rho_\tau(x_2) - \rho_\tau(x_1)|}{|x_2 - x_1|} \leq (1-\tau)$$

$\therefore L_{\rho_\tau(.)} = \max(\tau, 1-\tau) \because \tau \in [0, 1]$

Hence, the Lipschitz constant is

$$\frac{K_z * \max(\tau, 1-\tau)}{m} \tag{6}$$

## VI. Experimentation

In order to test the effectiveness of the adaptive learning rate for mean absolute error and check loss, the scheme is tested against the commonly used datasets that consist of regression based modelling[2].

1) California Housing Dataset- This dataset consists of 20 443 samples and 9 features that can be used to predict the mean housing price in a particular locality. The features include location of the house, number of rooms within that particular locality, age of houses, among many others.
2) Boston Housing Dataset- The dataset consists of 13 features, which are used to predict the median house value in a particular locality. Some of the features include average number of rooms per house and crime rate.
3) Energy Efficiency Dataset- The dataset consists of 8 features, which are used to predict heating and cooling load requirements of buildings. The dataset is used to perform multivariate regression.

In order to quantitatively compare the effectiveness of the constant learning rate and adaptive learning rate schemes, two methods of comparison are used

1) **Number of epochs**- A threshold value $T_L$ is chosen and the number of epochs required for the constant and adaptive learning rate schemes to reach the threshold is measured. This is to measure which method leads to *faster convergence*
2) **Performance**- The model is trained and the loss value after a fixed number of epochs is compared for the constant and adaptive learning rate schemes. This is to determine which method results in a model with *higher accuracy* in its predictions.

It is important to note that during the comparison, the *same initial weights are used for both the schemes*. Also, a constant learning rate of 0.1 was used.

### *A. Implementation details*

The entire framework is implemented in Keras, which uses Tensorflow as its backend. Keras allows for easy and fast prototyping. The flexibility provided by Keras to add dense layers and dropout layers with minimum ease made it an obvious choice to experiment with the regression datasets. Depending on the dataset, the underlying architecture such as the number of dense layers, number of neurons in each layer and dropout layers is chosen accordingly. Learning rate is calculated at the beginning of each epoch based on the mathematical expression derived. Learning rate is easily

[2]All datasets used are open-source and obtained from scikit-learn and University of California, Irvine Machine Learning Repository(https://archive.ics.uci.edu/ml/index.php)

integrated in the pipeline using the callbacks functionality in Keras.

### B. Choosing the threshold

A regression model is constructed using Ordinary Least Squares in order to obtain the threshold. The intuition behind using regression models is to obtain estimates of convergence points that can be obtained using a neural network.

## VII. Results

### A. Mean Absolute Error

Table I summarizes the results obtained for the three datasets

TABLE I: Mean Absolute Error: Number of epochs

| Dataset | Threshold | Epochs (constant) | Epochs (adaptive) |
|---|---|---|---|
| California Housing | 0.371 | 1409 | 114 |
| Boston Housing | 0.257 | 946 | 346 |
| Energy Efficiency | 0.229 | 1950 | 85 |

*Note: For Boston Housing and Energy Efficiency datasets, alternate methods of threshold calculations are used as stated below*

1) Find the minimum loss value from the constant learning rate scheme
2) Set the threshold as the above mentioned loss value

Heuristic 1: Alternative method for threshold calculation

Tables II and III compare the loss values after a fixed number of iterations of training between the constant and adaptive learning rate schemes. It is important to note that the adaptive learning rate automatically decreases over time for all the data sets, as proved mathematically in section V.

TABLE II: Mean Absolute Error: Loss value

| Dataset | Epochs | Loss(constant) | Loss(adaptive) |
|---|---|---|---|
| California Housing | 2500 | $0.3630 \pm 0.0049$ | $0.3474 \pm 0.0031$ |
| Boston Housing | 1000 | $0.2621 \pm 0.0015$ | $0.2480 \pm 0.0021$ |
| Energy Efficiency | 2000 | $0.2282 \pm 0.0020$ | $0.1631 \pm 0.0042$ |

TABLE III: Mean Absolute Error: Validation Loss value

| Dataset | Epochs | Validation Loss(constant) | Validation Loss(adaptive) |
|---|---|---|---|
| California Housing | 2500 | $0.3832 \pm 0.0053$ | $0.3686 \pm 0.0038$ |
| Boston Housing | 1000 | $0.3118 \pm 0.0096$ | $0.3134 \pm 0.0132$ |
| Energy Efficiency | 2000 | $0.2537 \pm 0.0037$ | $0.1785 \pm 0.0032$ |

*1) California Housing Dataset:* The architecture used for the California Housing dataset is described Table IV. As noted in Figure 2, the learning rate starts at a large value of 6.78 and decreases exponentially before saturating at a value of 0.55. As depicted in Figure 1, the adaptive learning rate policy converges 10 times faster than the constant learning rate based model.

TABLE IV: California Housing: Configuration of hyperparameters

| Hyperparameter | Value |
|---|---|
| Feature Scaling technique | Standardization |
| Batch size | 256 |
| Activation function | ReLU activation in the hidden layers<br>SoftSign activation in the last layer |
| Optimization algorithm | Mini-batch Gradient Descent |
| Number of hidden layers | 2 |
| Number of hidden neurons | [20,15] |
| Number of output units | 1 |

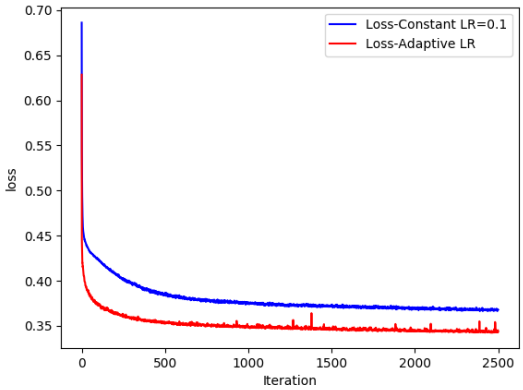



Fig. 1: California Housing: Training loss over time

*2) Energy Efficiency dataset:* Since the model predicts two output variables,the Lipschitz constant derived for multivariate regression in Section VB is used to compute the adaptive learning rate. The architecture of the neural network consists of a single hidden layer with 50 neurons. The model is trained using Gradient Descent with a batch size of 64. The remaining hyperparameters are the same as the ones described in Table IV. Figure 3 shows that the loss due to the adaptive learning rate decreases faster than that due to the constant learning rate. Moreover, although the learning rate starts at a high value 5.87, it decreases rapidly and eventually leads to faster convergence.

*3) Boston Housing Dataset:* The dataset is used to perform regression in order to predict housing prices in Boston. When implemented using a neural network, the architecture is described in Table V. Although the adaptive learning rate scheme converges twice as fast as the constant learning rate policy, the performance improvement observed is relatively smaller compared to the improvement in the other datasets.

In addition, the computational training time per epoch is larger for the adaptive learning rate based models due to the calculation of the adaptive learning rate. However, due to its faster convergence, LALR based models exhibit a smaller overall training time. For example, LALR based models trained on the Boston Housing dataset take up to 1.7 times longer to train per epoch compared to constant learning

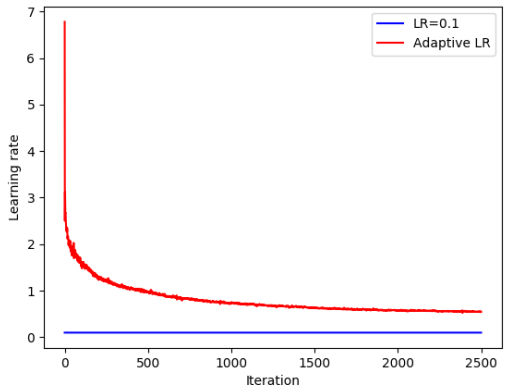



Fig. 2: California Housing: Learning rate over time

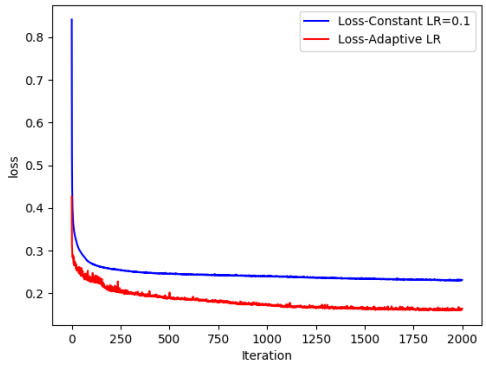



Fig. 3: Energy Efficiency: Training loss over time

rate based models, but depict a reduction in the overall training time by upto 2 times. Similarly, LALR based models trained on the Energy Efficiency dataset and the California Housing dataset depict a reduction in training time by 6 times and 8 times respectively.

Although the network architectures considered are simplistic, experiments with deeper architectures depict no further improvement in the training and validation loss obtained. For example, when tested on a 15 hidden layer network, as shown in Figure 5, the adaptive learning rate

TABLE V: Boston Housing: Configuration of hyperparameters

| Hyperparameter | Value |
|---|---|
| Feature Scaling technique | Standardization |
| Batch size | 8 |
| Activation function | ReLU activation in the hidden layer<br>SoftSign activation in the last layer |
| Optimization algorithm | Mini-batch Gradient Descent |
| Number of hidden layers | 1 |
| Number of hidden neurons | [20] |
| Number of output units | 1 |

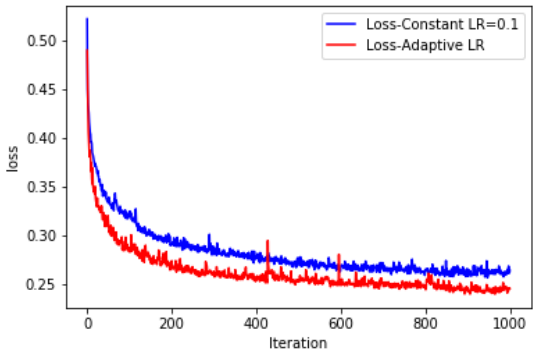



Fig. 4: Boston Housing: Training loss over time

based model exhibits a 2.5x faster convergence than the constant learning rate based model for the California Housing Dataset. The 15-hidden layer network consists of 100 neurons in the first hidden layer and 50 hidden neurons in each subsequent hidden layer. The hidden activation unit used is LeakyReLU with a small slope of 0.3 on the negative side. In addition, a dropout of 10% is used in each hidden unit. The remaining hyper-parameters are the same as shown in Table IV.

### *B. Quantile regression*

Firstly, quantile regression is implemented using neural networks in Keras and it is trained using a constant learning rate. The neural network consists of a single perceptron with zero hidden layers and a linear activation unit in its output layer. The network is trained using mini-batch gradient descent with a batch size of 64. The performance of the model is then compared against a baseline implementation in `statsmodel` [13]. Table VI provides a summary of the results. The results indicate that the performance of neural network based learning is comparable to statistical based learning.

Secondly, the efficacy of neural network based quantile regression is analyzed using a synthetically generated dataset. The dataset consists of a single feature and a single output and it is depicted in the following equation: $y = f(x) + \epsilon$

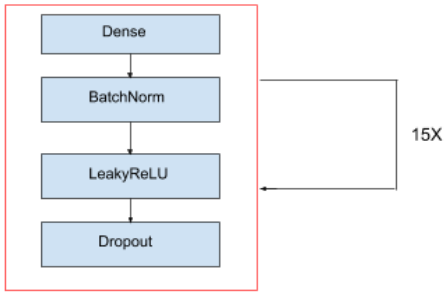



Fig. 5: 15 hidden layer network

TABLE VI: Comparison of quantile regression with a baseline implementation(AIC: Akaike Information Criteria)

| Dataset | Quantiles | | | | | |
|---|---|---|---|---|---|---|
| | 5th | | 50th | | 95th | |
| | AIC | AIC baseline | AIC | AIC baseline | AIC | AIC baseline |
| California Housing | 1278.98 | 1279.05 | 6180.87 | 6178.84 | 2298.53 | 2296.95 |
| Boston Housing | 53.9094 | 51.8603 | 146.3936 | 145.3413 | 73.1767 | 73.2916 |

TABLE VII: Comparison of theoretical and predicted quantiles

| Theoretical quantile | Predicted quantile |
|---|---|
| 0.05 | 0.07 |
| 0.30 | 0.296 |
| 0.50 | 0.457 |
| 0.70 | 0.732 |
| 0.95 | 0.959 |

where $\epsilon \sim$ N(0,$\sigma(x)^2$); $\sigma(x)$ = 0.1 exp(1-x) [8]. Various quantiles are chosen and individual neural networks are trained with the Check loss function for each quantile. The neural network consists of 2 hidden layers, with 10 and 5 neurons respectively. The hidden activation units employ the softplus activation function and a linear activation unit in the output layer. The network is trained using mini-batch gradient descent for 3000 iterations with a batch size of 64. The results are summarized in Table VII. Hence, neural networks are effective in modelling heteroscedastic data for quantile regression.

Due to the effectiveness of neural network based quantile regression, an adaptive learning rate policy for quantile loss is tested. To elaborate, the adaptive learning rate scheme is tested against a constant learning rate of 0.1 for the datasets mentioned in Section VI. Experiments are run independently for each chosen value of quantile and the results are summarized in Tables VIII IX, X. The threshold is calculated according to Heuristic 1 for all the datasets. Furthermore, the architecture and hyperparameters used for each of the datasets are identical to the ones used with Mean Absolute Error, as mentioned in Section VIIA, unless specified otherwise.

The adaptive learning rate for Check loss converges up to 13 times faster than the constant learning rate for the $5^{th}$ quantile and up to 10 times faster for the $95^{th}$ quantile. Figures 6a and 6b depict the training and validation loss for the California Housing dataset for the $5^{th}$ quantile. It is important to note that the model is able to achieve faster convergence without overfitting. Furthermore, the models for Energy Efficiency dataset also depict a similar trend with a speed up in convergence by 11 times. The architecture proposed in Table V for the Boston Housing dataset resulted in overfitting of the model. Consequently, the architecture is slightly modified to a single hidden layer network with 15 neurons. Moreover, the model is trained with a batch size of 256 for 1000 epochs. The remaining hyperparameters remain unchanged. Figures 6c and 6d depict a faster convergence by nearly 20 times, a significant increase in the speed of convergence compared to the other two datasets. In addition, the adaptive learning rate for the Boston Housing dataset starts at a large value of 6.43 and saturates at a value of 2.58. In contrast, the adaptive learning rate for the Energy Efficiency dataset starts at a relatively lower value of 3.23 and saturates at a value of 0.76.

TABLE VIII: Quantile loss: Comparison of iterations

| Dataset | Threshold | | Epochs(constant LR) | | Epochs(adaptive LR) | |
|---|---|---|---|---|---|---|
| | 5th | 95th | 5th | 95th | 5th | 95th |
| California Housing | 0.059 | 0.1659 | 995 | 1000 | 138 | 92 |
| Boston Housing | 0.0755 | 0.1605 | 1000 | 1000 | 47 | 42 |
| Energy Efficiency | 0.0406 | 0.1110 | 1977 | 1996 | 144 | 184 |

TABLE IX: Quantile loss: Loss value for various quantiles

| Dataset | Epochs | Loss(constant LR) | | Loss(adaptive LR) | |
|---|---|---|---|---|---|
| | | 5th | 95th | 5th | 95th |
| California Housing | 1000 | 0.0582 ± 0.0006 | 0.1659 ± 0.0004 | 0.0551 ± 0.0008 | 0.1599 ± 0.0004 |
| Boston Housing | 1000 | 0.0749 ± 0.0008 | 0.1597 ± 0.0007 | 0.0656 ± 0.0008 | 0.1439 ± 0.0015 |
| Energy Efficiency | 2000 | 0.0407 ± 0.0002 | 0.1107 ± 0.0002 | 0.0357 ± 0.0004 | 0.1062 ± 0.0002 |

TABLE X: Quantile validation loss: Validation Loss value for various quantiles

| Dataset | Epochs | Loss(constant LR) | | Loss(adaptive LR) | |
|---|---|---|---|---|---|
| | | 5th | 95th | 5th | 95th |
| California Housing | 1000 | 0.0616 ± 0.0007 | 0.1654 ± 0.0004 | 0.0595 ± 0.0009 | 0.1621 ± 0.0005 |
| Boston Housing | 1000 | 0.0744 ± 0.0009 | 0.1401 ± 0.0019 | 0.0695 ± 0.0030 | 0.1377 ± 0.0015 |
| Energy Efficiency[a] | 2000 | 0.0475 ± 0.0005 | 0.1130 ± 0.0006 | 0.0424 ± 0.0009 | 0.1081 ± 0.0003 |

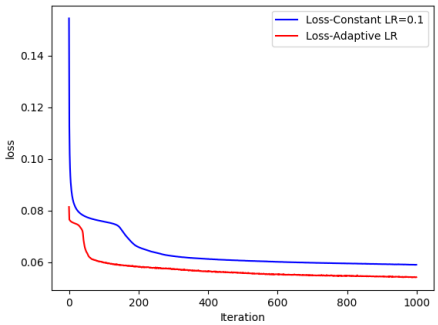
[a] Multiple Regression is used to predict a single output variable

Furthermore, as mentioned in Section VIIA, although additional computation is performed during every epoch in order to calculate the adaptive learning rate, the training time for LALR based models is lower than constant learning rate based models. For instance, LALR based models trained on the Energy Efficiency dataset depict an increase in training time per epoch by an average of 5 times, but exhibit an overall decrease in training time by 8 times. Similarly, for the Boston Housing dataset, LALR based models depict a decrease in the overall training time by up to 5 times although the computational time per epoch is 7 times higher.

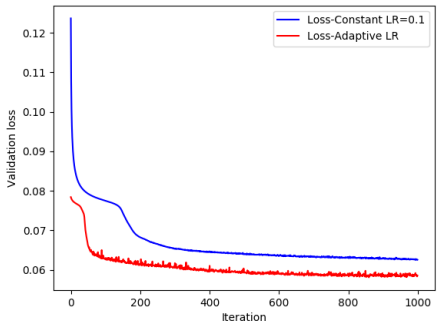
## VIII. Conclusion and Future Work

In summary, the paper proposes an adaptive learning rate scheme for regression based problems that utilize mean absolute error and check loss as loss functions. A theoretical framework for Lipschitz learning rate is derived for MAE and Check loss and its effectiveness is evaluated against commonly used regression based datasets.

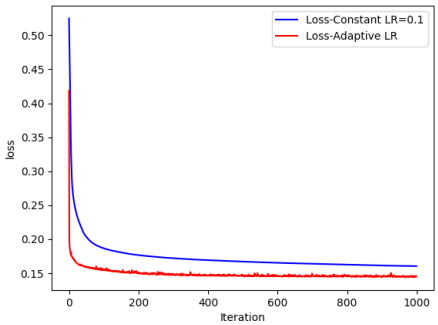
It is found that the adaptive learning rate policy performs better than the constant learning rate policy by a significant amount. For mean absolute error, the adaptive learning rate increases the speed of convergence by 5 to 20 times. Similarly, the adaptive learning rate for Check loss also achieves faster



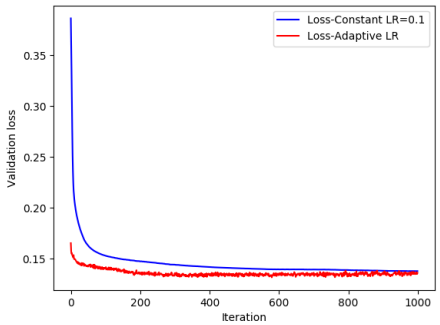

(a) $5^{th}$ quantile for California Housing: Training loss




(b) $5^{th}$ quantile for California Housing: Validation loss




(c) $95^{th}$ quantile for Boston Housing: Training loss




(d) $95^{th}$ quantile for Boston Housing: Validation loss

Fig. 6: Training and validation loss over time

convergence by nearly 20 times. It is important to note that even though the adaptive learning rate begins at a large value, faster convergence and better performance is achieved with Stochastic Gradient Descent, along with a natural decay in the learning rate. Hence, uncertainty in the prediction estimates of deep neural networks can be obtained with a smaller training time, thus increasing the reliability of the predictions.

An area of future work is to extend the theoretical framework for various other optimization algorithms such as momentum based gradient descent and Adam and evaluate its effectiveness. Furthermore, we also wish to explore an adaptive learning rate for Check loss that handles quantile crossing. Another area of future work is to analyze the relationship between Lipschitz Adaptive learning rate and activation units used in neural networks, primarily in the context of exploding gradients.

## ACKNOWLEDGEMENT

The authors would like to thank the Science and Engineering Research Board (SERB)-DST, Government of India for supporting this research (File SERB-EMR/ 2016/005687).

## REFERENCES

[1] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[2] R. Rojas, "The backpropagation algorithm," in *Neural networks*. Springer, 1996, pp. 149–182.

[3] Y. Bengio, "Practical recommendations for gradient-based training of deep architectures," in *Neural networks: Tricks of the trade*. Springer, 2012, pp. 437–478.

[4] S. Seong, Y. Lee, Y. Kee, D. Han, and J. Kim, "Towards flatter loss surface via nonmonotonic learning rate scheduling." in *UAI*, 2018, pp. 1020–1030.

[5] S. Lek, M. Delacoste, P. Baran, I. Dimopoulos, J. Lauga, and S. Aulagnier, "Application of neural networks to modelling nonlinear relationships in ecology," *Ecological Modelling*, vol. 90, no. 1, pp. 39 – 52, 1996. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0304380095001425

[6] A. Nguyen, J. Yosinski, and J. Clune, "Deep neural networks are easily fooled: High confidence predictions for unrecognizable images," *Computer Vision and Pattern Recognition (CVPR), 2015 IEEE Conference on*, 2015.

[7] C. Rudin, "Stop Explaining Black Box Machine Learning Models for High Stakes Decisions and Use Interpretable Models Instead," *arXiv e-prints*, 11 2018.

[8] I. Takeuchi, Q. V. Le, T. D. Sears, and A. J. Smola, "Nonparametric quantile estimation," *Journal of machine learning research*, vol. 7, no. Jul, pp. 1231–1264, 2006.

[9] M. Hardt, B. Recht, and Y. Singer, "Train faster, generalize better: Stability of stochastic gradient descent," *CoRR*, vol. abs/1509.01240, 2015. [Online]. Available: http://arxiv.org/abs/1509.01240

[10] I. Kuzborskij and C. H. Lampert, "Data-dependent stability of stochastic gradient descent," *arXiv preprint arXiv:1703.01678*, 2017.

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

[12] N. G. Reich, J. Lessler, K. Sakrejda, S. A. Lauer, S. Iamsirithaworn, and D. A. Cummings, "Case study in evaluating time series prediction models using the relative mean absolute error," *The American Statistician*, vol. 70, no. 3, pp. 285–292, 2016.

[13] S. Seabold and J. Perktold, "Statsmodels: Econometric and statistical modeling with python," in *Proceedings of the 9th Python in Science Conference*, vol. 57. Scipy, 2010, p. 61.