Parsimonious Computing: A Minority Training Regime for Effective Prediction in Large Microarray Expression Data Sets

1st Shailesh Sridhar
Department of Computer Science
PES University
Bangalore, India
shailesh.sridhar@gmail.com

2nd Snehanshu Saha
Department of Computer Science and Information Systems and APPCAIR
BITS Pilani K K Birla Goa Campus
Goa, India
snehanshu.saha@ieee.org

3rd Azhar Shaikh
Department of Electronics and Communication
PES University
Bangalore, India
azhar199865@gmail.com

4th Rahul Yedida
Department of Computer Science
North Carolina State University
Raleigh, USA
ryedida@ncsu.edu

5th Sriparna Saha
Department of Computer Science and Engineering
IIT Patna
Patna, India
sriparna.saha@gmail.com

***Abstract*—Rigorous mathematical investigation of learning rates used in back-propagation in shallow neural networks has become a necessity. This is because experimental evidence needs to be endorsed by a theoretical background. Such theory may be helpful in reducing the volume of experimental effort to accomplish desired results. We leveraged the functional property of Mean Square Error, which is Lipschitz continuous to compute learning rate in shallow neural networks. We claim that our approach reduces tuning efforts, especially when a significant corpus of data has to be handled. We achieve remarkable improvement in saving computational cost while surpassing prediction accuracy reported in literature. The learning rate, proposed here, is the inverse of the Lipschitz constant. The work results in a novel method for carrying out gene expression inference on large microarray data sets with a shallow architecture constrained by limited computing resources. A combination of random sub-sampling of the dataset, an adaptive Lipschitz constant inspired learning rate and a new activation function, A-ReLU helped accomplish the results reported in the paper.**

***Index Terms*—Lipschitz Constant, Adapative Learning, microarray expression data, A-Relu, Mean Square Loss, Mean Absolute Error.**

I. Introduction

Gene expression patterns are studied by microbiologists extensively to determine the genetic behaviour of cells. This is conventionally done via gene expression profiling, which is used to obtain and examine cell behaviour patterns in various scenarios such as drug treatment and disease. The CMap (connectivity Map) project was launched with the intent of creating a reference collection of patterns [1]. There are approximately 22,000 genes across the entire human genome. However, it has been discovered that most of them are highly correlated. Accordingly, regulatory and target genes have been identified. Researchers from the LINCS program, analyzing the CMap data found that around 80% of the data can be captured using a set of 1000 carefully selected genes. This inspired the development of the L1000 Luminex bead technology, which is able to measure the expression profiles of these approximately 1000 genes, called landmark genes at a significantly lower cost [2]. Using this data, the expression profiles of the other roughly 21,000 "target genes" can be inferred computationally. This computational inference is, however, challenging. Several techniques have been employed in the past for similar problems, such as Linear regression used by the LINCS researchers [3] and Kernel machines [4].

In recent years, due to the rapid growth of deep learning and neural networks, there have been several attempts to apply deep learning approaches to this problem. A recent significant work is D-GEX, a multi-task, multi-layer feedforward neural network, employing deep learning techniques such as dropout and momentum [5]. The authors compared the performance of this model to other machine learning methods such as k-NN regression and linear regression and show that deep learning achieves more accurate target gene predictions for gene expression. The best results of this model were obtained when an overwhelming 27,000 neurons were used across three hidden layers.

In contrast, we present a neural network that uses a substantially smaller number of neurons in a single hidden layer. It is reasonable to believe that the reduction in computing units will limit the representational power of the network. In order to counter this limitation, we employ an adaptive learning rate based on the Lipschitz constant and an activation function(A-ReLU) [6] that is more suitable for the given task. Drawing inspiration from the D-GEX paper, we utilize the microarray expression data from the GEO [7] project.

A. Motivation & Contribution

A gene expression profile consists of thousands of genes. Even a set of landmark genes consists of several hundreds. While deep learning architectures with several thousands of neurons are able to satisfactorily solve such problems, they are computationally intensive and prohibitively expensive. The authors of D-GEX [5] used an NVIDIA GTX TITAN Z Graphics card with dual GPUs to train. Such computing infrastructure is scarce and orthogonal to the philosophy of Parsimonious Computing, propagated in our work. Our goal is to show that a much smaller neural architecture can be used for the same task without suffering from unacceptable prediction error and can be trained with much smaller computing infrastructure requirements. The following theoretical contributions help in solving the gene inference problem with minimal computing infrastructure, called Parsimonious Computing model proposed in our work.

a) **Lipschitz constant adaptive learning rate**
The magnitude of the learning rate plays a huge role in determining the time a neural network will take to converge. Too small a learning rate results in a very small rate of convergence and possible premature convergence at a local minima, while a learning rate which is too large may not be able to converge at all. To counter this, adaptive learning rate schedulers have been proposed, so that the learning rate is adequately large when away from the global minima, and appropriately small when the network is close to it. Here we propose a learning rate scheduler which is calculated from the following formula:

$$\boxed{\max_{i,j}\left\|\frac{\partial E}{\partial w_{ij}^{[L]}}\right\| = \frac{1}{m}\left(K_a + \|\mathbf{y}\|\right)K_z} \tag{1}$$

where,

- K_a denotes the maximum of the activation in the final layer of the network
- K_z denotes the maximum of the activation in the penultimate layer
- y denotes the target values
- m is the number of training examples

b) **A-ReLU as an activation function**
The original D-GEX network utilizes tanh as an activation function. Though quite effective, activation functions such as tanh and sigmoid suffer from the vanishing gradient problem as the gradient is squashed between a small range of values. ReLU overcomes this issue by providing a constant gradient value for all inputs. However, ReLU has issues of its own, as it is non-differential at 0. In this work, we use A-ReLU as an activation function, a relatively new activation function defined by the equation

$$\begin{cases} kx^n & x \geq 0 \\ 0 & x < 0 \end{cases}$$

We show experimentally that it performs better than other classical activation functions in this regression task. We claim to have established the following major items.

- Detailed proof of Lipschitz Adaptive Learning Rate (LALR) for Mean Square Loss (MSE) in training a shallow neural network.
- A novel activation function, A-ReLU to demonstrate better performance compared to established activation units
- Propose Parsimonious Computing, a philosophy that advocates practice of frugality in computing resources, by leveraging deep mathematical insights; in particular, a combination of the above two items.
- Provide empirical evidence that the effect of Lipschitz learning is equivalent to dropping one hidden layer of 9000 neurons in the deep learning architecture used in the base paper [5], while maintaining comparable prediction accuracy. In other words, a shallow network of 2 hidden layers of 9000 neurons, powered by Parsimonious Computing model matches the performance of a deep neural net with 3 hidden layers of 9000 neurons, twice as large as our shallow neural network. Table I provides evidence in support of this claim.

II. Literature Survey

The field of micro-array bioinformatics has witnessed tremendous growth with a large focus on gene expression profiling. Gene expression profiling was used to identify biomarkers for Parkinson's disease in [8]. Molecular information obtained from Micro-Array data is leveraged to predict if a tumour is cancerous or not in a supervised setting. The same information is used to discover new types of tumours in an unsupervised setting in [9]. A comprehensive comparison of applying different machine learning techniques like SVM, RBF multi-layer perceptron, Random Forest etc. and feature selection methods like chi-squared and CSF for classification on micro-array gene expression data was given in [10]. Correlation between gene profiles is exploited by gene regulatory networks [11] to identify landmark and target genes. This paved the way for research in building computational inference models to predict target gene profiles using landmark genes. A computational model based on linear regression was proposed by [3], but was not complex enough to capture intrinsic non-linearity present within micro-array data. The number of independent regulatory modules between the landmark and target genes are inferred from the rank of their connectivity matrix and then is used as a low rank regularisation constraint to formulate gene expression inference as a multi-target linear regression problem in [4]. Recent emergence of deep learning methods has led to its application as a diagnostic tool for gene expression analysis. DeepCC [12], a deep learning framework for cancer molecular sub-type classification, outperforms all traditional machine learning methods on 13 independent data sets based on Affymetrix platforms. A 3 layer feed-forward neural network named D-GEX is used to predict gene expression values of target genes in [5]. This method outperforms all previous methods by a large margin and is used as a baseline for our work.

A fundamental problem faced by neural networks is that of finding optimal values for its learning rate. Recent works agree that a non-monotonic learning rate scheduling system would offer faster convergence [13], [14]. Of late, there has been some development in finding novel ways to adaptively change the learning rate of a neural network. The results have theoretical, intuitive and empirical support and rely on non-monotonic scheduling of the learning rate. The method we use here, also yields a non-monotonic learning rate, but does not follow any predefined shape. Contrary to the trend of using deeper neural networks for regressing gene expression values, we exploit wider shallow neural networks. We claim these are as effective as their deep counterparts with the added benefit of cheap training due to the reduction in number of parameters.

The remainder of the paper is organized in the following manner: Section III describes data and and its handling by constrained computing resources. This is followed by Parsimonious Computing in section IV, the motivation behind tackling such a corpus of data in a cost effective way. Section IV, thus, describes the methods which help accomplish Parsimonious Computing goals, in other words make cheap training a feasible option. Section V discusses experimental settings and interventions, followed by detailed results in section VI.

III. Datasets

GEO (Gene Expression Omnibus) is a public functional genomics data repository from which The Broad Institute produced the GEO expression data. The data set consists of 129,158 gene expression profiles or samples obtained from the Affymetrix microarray platform. Each profile is associated with 22,268 gene probes, Out of which 978 are landmark genes and the remaining 21,290 are target genes whose values are to be predicted. Authors in the base paper [5] processed the data into a simpler and easier to handle format, by implementing first quantile normalization of the data into a numerical range between 4 and 15, followed by removing any duplicates. The final, ready-to-use data has 111,009 profiles.

The GEO dataset is enormous, with just the training data consuming more than 3GB of space. Given that all training was carried out using a single GPU with 4GB ram, we used random subsampling along with the 50:50 split employed by the D-GEX authors [5].

In order to represent the entire GEO-tr dataset appropriately while ensuring that the computing infrastructure utilized can handle the scale of data used, the network is trained on six random sub-samples of the data set, each containing 20,000 data points or slightly less than 1/5th of it. A representative MAE was obtained by averaging the MAE obtained across each individual subsample. This average is considered so that any disparity among sub-samples could be taken into account.

IV. Parsimonious Computing: Our contribution in Cheap Training

In this section we discuss the methods used in our model that enable us to achieve results very close to those reported by the D-GEX method [5] but without requiring the same amount of compute power. All experiments were carried out on a laptop with an Intel i5 processor and an NVIDIA GTX-1050Ti graphics card. The GPU utilized has 4GB RAM associated with it.

A. Lipschitz Adaptive Learning Rate

Recently, there has been a lot of work on finding novel ways to adaptively change the learning rate. These have both theoretical [15] and intuitive, empirical [13], [14] backing. These works rely on non-monotonic scheduling of the learning rate. Authors in [13] argue for cyclical learning rates. Our proposed method also yields a non-monotonic learning rate, but does not follow any predefined shape. we propose a novel theoretical framework to compute large, adaptive learning rates for use in gradient-based optimization algorithms. We start with a presentation of the theoretical framework and the motivation behind it, and then derive the mathematical formulas to compute the learning rate on each epoch.

Our results show that, compared to standard choices of learning rates, our approach converges quicker and achieves better results. Our approach exploits functional properties of the loss function, and only makes two minimal assumptions on the loss function: it must be Lipschitz continuous [16] and (at least) once differentiable. Commonly used loss functions satisfy both these properties. We argue that the use of Lipschitz constants to determine learning rate greatly improves convergence in comparison with standard learning rate choices. We present empirical evidence of our claims in the results section. This is a departure from the approach of manually tuning learning rates.

1) Theoretical Framework: For a function, the Lipschitz constant is the least positive constant L such that

$$\|f(\mathbf{w}_1) - f(\mathbf{w}_2)\| \leq L\,\|\mathbf{w}_1 - \mathbf{w}_2\| \tag{2}$$

for all $\mathbf{w}_1$, $\mathbf{w}_2$ in the domain of f. From the mean-value theorem for scalar fields, for any $\mathbf{w}_1, \mathbf{w}_2$, there exists $\mathbf{v}$ such that

$$\begin{aligned}\|f(\mathbf{w}_1) - f(\mathbf{w}_2)\| &= \|\nabla_{\mathbf{w}} f(\mathbf{v})\|\|\mathbf{w}_1 - \mathbf{w}_2\| \\ &\leq \sup_{\mathbf{v}}\|\nabla_{\mathbf{w}} f(\mathbf{v})\|\|\mathbf{w}_1 - \mathbf{w}_2\|\end{aligned}$$

Thus, $\sup_{\mathbf{v}}\|\nabla_{\mathbf{w}} f(\mathbf{v})\|$ is such an L. Since L is the least such constant,

$$L \leq \sup_{\mathbf{v}}\|\nabla_{\mathbf{w}} f(\mathbf{v})\| \tag{3}$$

In this paper, we use $\max\|\nabla_{\mathbf{w}} f\|$ to derive the Lipschitz constants. Our approach makes the minimal assumption that the functions are Lipschitz continuous and differentiable up to first order only [1]. Because the gradient of these loss functions

[1]Note this is a weaker condition than assuming the gradient of the function being Lipschitz continuous. We exploit merely the boundedness of the gradient.

is used in gradient descent, these conditions are guaranteed to be satisfied.

By setting $\alpha = \frac{1}{L}$, we have $\Delta\mathbf{w} \leq 1$, constraining the change in the weights. We stress here that we are not computing the Lipschitz constants of the *gradients* of the loss functions, but of the losses themselves. Therefore, our approach merely assumes the loss is L-Lipschitz, and not β-smooth. We argue that the boundedness of the effective weight changes makes it optimal to set the learning rate to the reciprocal of the Lipschitz constant. This claim, while rather bold, is supported by our experimental results.

2) Significance of Lipschitz constant (LC): The Lipschitz constant (LC) has found a variety of uses in computing and applications. The central condition to the existence and uniqueness of solutions to first order system of differential equations of the form $y'(t) = f(t, y(t)$ is LC of f. The existence of LC guarantees contraction and eventually a fixed point i.e. solution to the above system [17] and saves the trouble of computing an analytical solution to the system above. Finding an LC is equivalent to to the fact that the function, f possesses Lipschitz continuity. Given, $f : R \to R$, there exists an L such that

$$\|f(\mathbf{x}) - f(\mathbf{y})\| \leq L\,\|\mathbf{x} - \mathbf{y}\|$$

Consequently,

$$\frac{\|f(\mathbf{x}) - f(\mathbf{y})\|}{\|\mathbf{x} - \mathbf{y}\|} \leq L$$

This is equivalent to stating that computing a LC of a function (loss function, in our case) is identical to computing the maximum of the derivative of f, via the Mean Value theorem. For loss functions which are differentiable, we can easily compute LCs and therefore find the bound on the derivatives to be used in deep neural network training. Mean Square Loss (MSE) satisfy the conditions of differentiability and hence Lipschitz continuity. We compute the LC of MSE enabling us to arrive at adaptive learning rate formulation, Lipschitz Adaptive Learning rate (LALR).

3) Deriving the Lipschitz constant for neural networks: For a neural network that uses the sigmoid, (or A-ReLU), or softmax activations, it is easily shown that the gradients get smaller towards the earlier layers in backpropagation. Because of this, the gradients at the last layer are the maximum among all the gradients computed during backpropagation. If $w_{ij}^{[l]}$ is the weight from node i to node j at layer l, and if L is the number of layers, then

$$\max_{h,k}\left\|\frac{\partial E}{\partial w_{hk}^{[L]}}\right\| \geq \left\|\frac{\partial E}{\partial w_{ij}^{[l]}}\right\| \forall\; l, i, j \tag{4}$$

4) Least-squares cost function: For the least squares cost function, we will compute the Lipschitz constant for linear regression where the output is continuous. We will then prove the equivalence of the general result with regression in neural networks and derive the former as a special case of the latter.

5) Linear regression: We have,

$$g(\mathbf{w}) = \frac{1}{2m}\sum_{i=1}^{m}\left(\mathbf{x}^{(i)}\mathbf{w} - y^{(i)}\right)^2$$

Thus,

$$\begin{aligned} g(\mathbf{w}) - g(\mathbf{v}) &= \frac{1}{2m}\sum_{i=1}^{m}\left(\mathbf{x}^{(i)}\mathbf{w} - y^{(i)}\right)^2 - \left(\mathbf{x}^{(i)}\mathbf{v} - y^{(i)}\right)^2 \\ &= \frac{1}{2m}\sum_{i=1}^{m}\left(\mathbf{x}^{(i)}(\mathbf{w}+\mathbf{v}) - 2y^{(i)}\right)\left(\mathbf{x}^{(i)}(\mathbf{w}-\mathbf{v})\right) \\ &= \frac{1}{2m}\sum_{i=1}^{m}\left((\mathbf{w}+\mathbf{v})^T\mathbf{x}^{(i)T} - 2y^{(i)}\right)\left(\mathbf{x}^{(i)}(\mathbf{w}-\mathbf{v})\right) \\ &= \frac{1}{2m}\sum_{i=1}^{m}\left((\mathbf{w}+\mathbf{v})^T\mathbf{x}^{(i)T}\mathbf{x}^{(i)} - 2y^{(i)}\mathbf{x}^{(i)}\right)(\mathbf{w}-\mathbf{v}) \end{aligned}$$

The penultimate step is obtained by observing that $(\mathbf{w} + \mathbf{v})^T\mathbf{x}^{(i)T}$ is a real number, whose transpose is itself.

At this point, we take the norm on both sides, and then assume that $\mathbf{w}$ and $\mathbf{v}$ are bounded such that $\|\mathbf{w}\|, \|\mathbf{v}\| \leq K$. Taking norm on both sides,

$$\boxed{\frac{\|g(\mathbf{w}) - g(\mathbf{v})\|}{\|\mathbf{w} - \mathbf{v}\|} \leq \frac{K}{m}\left\|\mathbf{X}^T\mathbf{X}\right\| + \frac{1}{m}\left\|\mathbf{y}^T\mathbf{X}\right\|}$$

We are forced to use separate norms because the matrix subtraction $2K\mathbf{X}^T\mathbf{X} - 2\mathbf{y}^T\mathbf{X}$ cannot be performed. The RHS here is the Lipschitz constant. Note that the Lipschitz constant changes if the cost function is considered with a factor other than $\frac{1}{2m}$.

6) Regression with neural networks: Let the loss be given by

$$E(\mathbf{a}^{[L]}) = \frac{1}{2m}\left(\mathbf{a}^{[L]} - \mathbf{y}\right)^2 \tag{5}$$

where the vectors contain the values for each training example. Then we have,

$$\begin{aligned} E(\mathbf{b}^{[L]}) - E(\mathbf{a}^{[L]}) &= \frac{1}{2m}\left(\left(\mathbf{b}^{[L]} - \mathbf{y}\right)^2 - \left(\mathbf{a}^{[L]} - \mathbf{y}\right)^2\right) \\ &= \frac{1}{2m}\left(\mathbf{b}^{[L]} + \mathbf{a}^{[L]} - 2\mathbf{y}\right)\left(\mathbf{b}^{[L]} - \mathbf{a}^{[L]}\right) \end{aligned}$$

This gives us,

$$\begin{aligned} \frac{\|E(\mathbf{b}^{[L]}) - E(\mathbf{a}^{[L]})\|}{\|\mathbf{b}^{[L]} - \mathbf{a}^{[L]}\|} &= \frac{1}{2m}\|\mathbf{b}^{[L]} + \mathbf{a}^{[L]} - 2\mathbf{y}\| \\ &\leq \frac{1}{m}\left(K_a + \|\mathbf{y}\|\right) \end{aligned} \tag{6}$$

where K_a is the upper bound of $\|\mathbf{a}\|$ and $\|\mathbf{b}\|$. A reasonable choice of norm is the 2-norm.

By equation (13) (please see the subsection below, Equivalence of the constants), the second term on the right side of the equation is the derivative of the activation with respect to its parameter. Notice that if the activation is sigmoid or softmax, then it is necessarily less than 1; if it is ReLU type, it is either 0 or 1. Therefore, to find the maximum, we assume that the network is comprised solely of ReLU type activations, and the maximum of this is 1.

From (13) and 'equivalence of constants' calculations, we obtain

$$\boxed{\max_{i,j}\left\| \frac{\partial E}{\partial w_{ij}^{[L]}} \right\| = \frac{1}{m}\left(K_a + \|\mathbf{y}\|\right) K_z} \tag{7}$$

The Learning Rate to be used is hence a reciprocal of this calculated value. For example, in one of our experiments, after initialization, the values of the constants turned out to be the following:
K_z = 983.88; K_a = 142.86 & $y = 4329.24$. Substituting these values in the equation and scaling down by multiplying by a factor of 0.3, we arrive at a learning rate of approximately 1.36×10^{-4}. The scale-down factor is intuitive and considered "on-the-fly" to mitigate a likely exploding gradient problem by A-ReLU.

7) Equivalence of the constants: The equivalence of the above two formulas is easy to see by understanding the terms of (7). Let us define

$$K_z = \max_j \left\| a_j^{[L-1]} \right\| \tag{8}$$

In any layer, we have the computations

$$z^{[l]} = W^{[l]T} a^{[l-1]} + b^{[l]} \tag{9}$$

$$a^{[l]} = g(z^{[l]}) \tag{10}$$

$$a^{[0]} = X \tag{11}$$

Thus, the gradient with respect to any weight in the last layer is computed via the chain rule as follows.

$$\begin{aligned} \frac{\partial E}{\partial w_{ij}^{[L]}} &= \frac{\partial E}{\partial a_j^{[L]}} \cdot \frac{\partial a_j^{[L]}}{\partial z_j^{[L]}} \cdot \frac{\partial z_j^{[L]}}{\partial w_{ij}^{[L]}} \\ &= \frac{\partial E}{\partial a_j^{[L]}} \cdot \frac{\partial a_j^{[L]}}{\partial z_j^{[L]}} \cdot a_i^{[L-1]} \end{aligned} \tag{12}$$

This gives us

$$\max_{i,j}\left| \frac{\partial E}{\partial w_{ij}^{[L]}} \right| \le \max_j \left| \frac{\partial E}{\partial a_j^{[L]}} \right| \cdot \max_j \left| \frac{\partial a_j^{[L]}}{\partial z_j^{[L]}} \right| \cdot \max_j \left| a_j^{[L-1]} \right| \tag{13}$$

Because a linear regression model can be thought of as a neural network with no hidden layers and a linear activation, and from (11), we have,

$$\mathbf{a}^{[L-1]} = \mathbf{a}^0 = \mathbf{X}$$

and therefore

$$K_z = \max_j \left\| a_j^{[L-1]} \right\| = \|\mathbf{X}\| \tag{14}$$

Next, observe that K_a is the upper bound of the final layer activations. For a linear regression model, we have the "activations" as the outputs: $\hat{\mathbf{y}} = \mathbf{W}^T\mathbf{X}$. Using the assumption that $\|\mathbf{W}\|$ has an upper bound K, we obtain

$$K_a = \max\left\| \mathbf{a}^{[L]} \right\| = \max\left\| \mathbf{W}^T\mathbf{X} \right\| = \max\|\mathbf{W}\| \cdot \|\mathbf{X}\| = K\|\mathbf{X}\| \tag{15}$$

Substituting (14) and (15) in (7), we obtain

$$\begin{aligned} \max_{i,j}\left\| \frac{\partial E}{\partial w_{ij}^{[L]}} \right\| &= \frac{1}{m}\left(K_a + \|\mathbf{y}\|\right) K_z \\ &= \frac{1}{m}\left(K\|\mathbf{X}\| + \|\mathbf{y}\|\right)\|\mathbf{X}\| \\ &= \frac{K}{m}\left\|\mathbf{X}^T\mathbf{X}\right\| + \frac{1}{m}\left\|\mathbf{y}^T\mathbf{X}\right\| \end{aligned}$$

B. A-ReLU

We have employed the use of the activation function A-ReLU [6] in our experiments. A-ReLU is a continuous and differentiable approximation of the ReLU Activation function, proven to be differentiable at 0 too, thereby alleviating the problem of undefined gradients in the neighbourhood of 0 faced by ReLU. Moreover, it is straightforward to show that

- A-ReLU does not suffer from local minima problems
- A-ReLU does not admit of saddle points
- A-ReLU admits of a fixed point (easy to show by virtue of Intermediate Value Theorem) thereby ensuring optima.
- The exploding gradient is easy to control

What needs to be considered is the lack of tuning efforts activation functions usually need. This was accomplished using approximation techniques and Hausdorff distance. ReLU is an offshoot of SBAF [6]. Let us consider the activation function, SBAF:

$$y = \frac{1}{1 + kx^{\alpha}(1-x)^{1-\alpha}} \tag{16}$$

α + β =1 where $\alpha > 0$ and $\beta > 0$. We show the k, α values for which SBAF approximates to A-ReLU activation function. $k = 1, \alpha = 1$; SBAF becomes $\frac{1}{1+x}$ which upon binomial expansion (restricting to first order expansion assuming $0 < x < 1$) yields $y = 1 - x = 1 -$ ReLU. Approximate ReLu (A-ReLu) is motivated by the fact it is a least square approximation of ReLu such that AReLu is continuous and differentiable at $x = 0$ unlike ReLu and could also be derived from the generic family of activation functions detailed in [6]. Therefore, least square optimization is the way forward to determine the optimal parameters of A-ReLU. This is presented below.
Consider the function $f(x) = kx^n$. We know that the ReLU activation function is $y = \max(0, x)$. We need to approximate the values n and k such that $f(x)$ approximates to the ReLU activation function over a fixed interval. Define, Relative error $= \frac{||f(x)-y||}{||y||}$. Let the minimum tolerable error be $\epsilon < 10^{-3}$.Thus, assuming a error threshold,

$$\begin{aligned} \frac{||f(x) - y||}{||y||} &<= \epsilon < 10^{-3} \\ ||f(x) - y|| &\le 10^{-3}||y|| \\ ||f(x)|| &\le ||y||(10^{-3} + 1) \\ ||f(x)|| &\le 1.001||y|| \end{aligned}$$

Since $f(x) = kx^n$ approximates the positive half (i.e., when $x > 0$) of the ReLU activation function, $y = max(x, 0)$, the value of y when $x > 0$ can be written as: $||y|| = ||x||$. Using this value in the error calculation, we rewrite the error approximation as,

$$||kx^n|| \leq 1.001||x||$$
$$||kx^{n-1}|| \leq 1.001$$

The above is an optimization problem i.e. $\min \left\|kx^{n-1}\right\|$ subject to the constraints $k > 0, n > 1, -10 < x < 10$. We obtain the following bounds on k and n:

$$0 < k < 1; 1 < n < 2$$

Therefore, we obtain the following continuous approximation of ReLU:

$$\begin{cases} kx^n & x \geq 0 \\ 0 & x < 0 \end{cases}$$

where $0 < k < 1, 1 < n < 2, -10 < x < 10$. More precisely, the approximation to the order of 10^{-3} is $k = 0.54, n = 1.3$. We used the parameter values k, α in the ballpark range while training the network. This ballpark range agrees with the Mathematical notion of $\epsilon-$ neighborhood of the theoretically computed values.

V. Experiments And Results

The variables in our experiments are the type of activation function used, the number of hidden layer neurons, the number of epochs and the type of learning rate policy used. For each experiment, we choose one activation function from a pool of four (Sigmoid, Tanh, ReLU, A-ReLU), either 500 or 1500 or 3000 hidden neurons, one among fixed learning rate policy, the Decay Factor based Scheduler employed in [5] and the Lipschitz learning rate policy for 200 epochs. In [5], the authors used a pair of bounds on the higher and lower values of the learning rate (5×10^{-4}) and (1×10^{-5}). We use the same pair of bounds in our training. We only use a single hidden layer in most experiments as opposed to upto 3 (with upto 9000 hidden neurons in each layer) used in the D-GEX(D) architecture [5] which uses a decay factor based scheduler to vary the learning rate, and report a MAE which is very close to the one reported by the base paper authors. Hence our proposed model can effectively perform as well as the deep model but with a much smaller number of parameters.Due to the shallowness of the model used, it can be trained on low compute hardware, as shown in our experiments. We train our model for 100 iterations every epoch with a batch size of 200 on a single Nvidia 1050Ti GPU. From here onwards we will refer to the models using a Decay based LR(Base paper), Lipschitz Adaptive LR(Our proposed model) and fixed LR(for comparison) as D-GEX(D), D-GEX(L) and D-GEX(F). All experiments carried out for the aforementioned models and results obtained are without dropout.

We trained our modified D-GEX(L) architecture on GEO-tr using random sub-sampling and tested on the GEO-te data. The tables below indicate the MAE and standard deviation between subsamples, obtained by each model.

A. Performance

Our best performing model uses the Lipschitz Adaptive Learning Rate along with the A-ReLU activation function, and is trained for 200 epochs. The results are summarized in Tables I,II,III and IV. For the A-Relu activation function, we use $k = 0.6$ and $n = 1.2$. The learning rate in the fixed LR experiments is set to 5×10^{-7}. The starting value for the Lipschitz Adaptive LR depended on the random initialization of the network weights.

TABLE I
MAE Comparison: Our Model trained on Google Colab Vs D-GEX: Both models are trained on entire corpus of training data. Our model does better than D-GEX with constrained infrastructure (One hidden layer less)

Training on the Entire data set	
Model	MAE
D-GEX(D) 3-layer architecture 9000 × 3	0.3240
D-GEX(L)(A-ReLU) 2-layer architecture 9000 × 2	0.3213

TABLE II
Comparison of the Smallest D-GEX(D) architecture Trained on Entire Dataset and D-GEX(L) Trained on Random subsamples(Less Than 1/5 Size: D-GEX(D) performs marginally better but held the advantage of training over the entire data set.)

Epochs	D-GEX(D) (3000)	D-GEX(L) (500)	D-GEX(L) (1500)
100	–	0.380709	0.367214
200	0.3421	0.378030	0.364621

TABLE III
Comparison Of MAE on D-GEX(L) Vs D-GEX(D): Both models are trained on identical subsamples: Our model D-GEX(L) performs better.

Training on sub-sampled datasets				
Epochs	D-GEX(D) (500)	D-GEX(D) (1500)	D-GEX(L) (500)	D-GEX(L) (1500)
100	0.421837	0.399483	0.380709	**0.367214**
200	0.388720	0.396381	0.378030	**0.364621**
$0.0001 \leq \sigma \leq 0.0007$				

TABLE IV
Mean Absolute Error when training D-GEX(D): Trained on Subsamples

MAE after training with D-GEX(D) with different choices of activations					
Number Of Neurons	Number of Epochs	Sigmoid	Tanh	ReLU	A-ReLU
500	200	0.438009	0.396381	**0.390081**	0.394212
1500	200	0.389272	0.388720	0.381064	0.378030
$0.00008 \leq \sigma \leq 0.0003$					

TABLE V

MEAN ABSOLUTE ERROR WHEN TRAINING D-GEX(L): TRAINED ON SUBSAMPLES

MAE after training D-GEX(L) with different choices of activations					
Number Of Neurons	Number of Epochs	Sigmoid	Tanh	ReLU	A-ReLU
500	200	0.389381	0.388309	0.381064	0.378030
1500	200	0.374043	0.376105	0.368724	**0.364621**
$0.0001 \leq \sigma \leq 0.0006$					

TABLE VI

MEAN ABSOLUTE ERROR WHEN TRAINING D-GEX(F):TRAINED ON SUBSAMPLES

MAE after training with D-GEX(F) with different choices of activations					
Number Of Neurons	Number of Epochs	Sigmoid	Tanh	ReLU	A-ReLU
500	200	0.422051	0.395568	0.387693	0.387512
1500	200	0.402797	0.380132	0.370924	0.370296
$0.00004 \leq \sigma \leq 0.0004$					

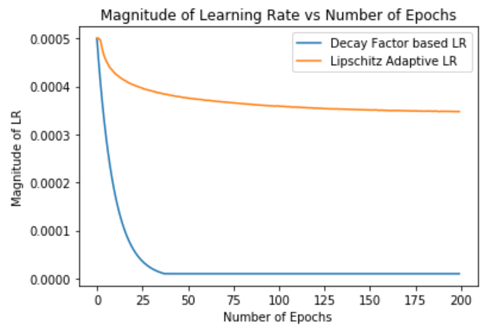

Fig. 1. Lipschitz Adaptive Learning Rate is always significantly higher, than the Decay Based Learning Rate employed in [5] allowing much faster convergence

As expected, reflecting the base paper, the larger among the two architectures used performed the best, as larger architectures generally allow for a richer representation of features.

1) The effect of Lipschitz adaptive learning rate: It is apparent that in all configurations, architectures employing the use of the LALR, (D-GEX(L)) converge better than those using a fixed learning rate. Using a Lipschitz adaptive scheduler allows the learning rate to be appropriately large away from the minima, allowing it to converge faster, and adequately small so as to not overshoot the minima when in its vicinity.

As seen in Figure 1, the learning rate while using the scheduler showcases a seemingly exponential reduction while training, and is significantly larger than the learning rate obtained while using the Decay Factor based scheduler used by the authors in [5], while not being too large. This allows a much faster convergence.

Given the difficulty of the problem, the Lipschitz adaptive learning rate allows much faster convergence and makes it possible to carry out training even with minimal computing infrastructure with a reasonable training time.

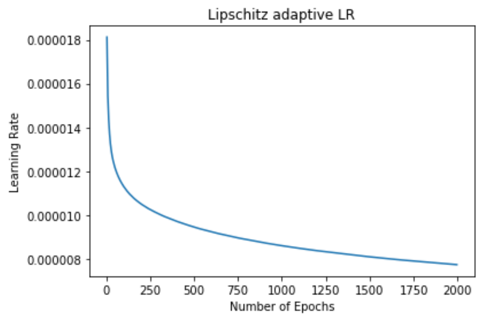

Fig. 2. Learning Rate using Lipschitz adaptive LR over 2000 epochs. The learning rate showcases an exponential decrease

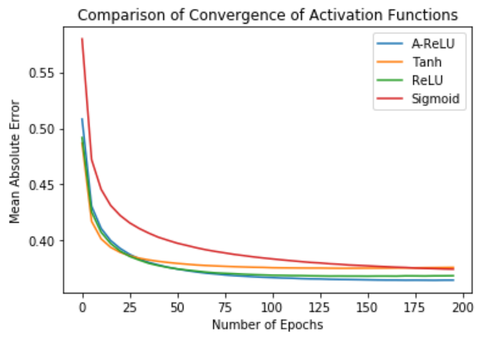

Fig. 3. MAE from 0 to 200 epochs

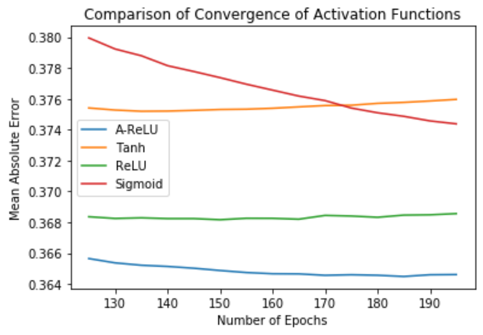

Fig. 4. MAE from 125 to 200 epochs

The two plots above indicate how the A-ReLU activation function allows the network to converge to smaller errors than other, more conventional activation functions in this task

2) The effect of using A-ReLU: Another interesting observation is the success of A-ReLU as an activation function when training in conjunction with Lipschitz scheduler. The original D-GEX paper trains the network for 200 epochs using the Tanh activation function. Comparing the performance of standard activation functions, and for the same number of epochs, we observe that A-ReLU performs the best.

VI. DISCUSSION AND CONCLUSION

We demonstrate that neural network learning of computational biology tasks like gene expression inference from huge Genomic data sets can be achieved with limited computational resources, specifically only one GPU with 4 GB RAM. We also present evidence that a combination of a shallow neural network and an adaptive learning rate achieve better performance than deeper nets.

We divide the GEO dataset into smaller random sub-samples in order to handle the massiveness of the training data. To ensure that no bias due to the exclusion of samples because of the random nature of sub-sampling, the reported results are obtained by averaging the MAEs corresponding to each one of six sub-samples. In order to accelerate training, an adaptive learning rate based on the Lipschitz constant has been utilized. The performance is also subjected to different activation functions to capture significant deviation, if any. A-ReLU is found to outperform several other, better known activation functions.

We obtain similar results to the original D-GEX architecture [5] with a neural network containing a single hidden layer with 1500 neurons and trained on data, 1/5th the size of the original data set. This is reported in Table II. However, when the original D-GEX architecture and model was employed on smaller subsets of data, it is observed that our methods perform better (please see Table III). In order to establish the merit of our contribution further, we ran a final set of experiments on the entire training corpus, on Google COLAB, with a Tesla K80 GPU having 12 GB RAM. We show, in Table I, that we have again accomplished better performance in comparison with the base paper [5]. Apart from one,all the architectures used in this work consist of a single hidden layer, with a maximum of 1500 neurons. The largest architecture utilized in [5] was composed of three hidden layers of 9000 neurons each. As opposed to NVIDIA TITAN Z GPUs used in the base paper [5], all training in this work was done using an NVIDIA GeForce 1050Ti GPU possessing upto 4 GB RAM, on a laptop, except for one case which shows that our methods can be used at-scale.

The learning rate computed and used in training in this paper is truly adaptive and the term is not loosely used. The new learning rate at every iteration is computed by the formula presented in section IV. This is done without manual intervention and any sort of camouflaged supervision. Such Lipschitz adaptive learning rate is, of course, dependent on the choice of loss function and will vary based on the loss used in training and provided that the loss function is continuous and smooth up to the first order i.e. Lipschitz. The remarkable outcome of developing a LALR based training is evident in larger architectures as well; In Table I we demonstrate that, our network of 2 hidden layers of 9000 neurons beats the performance of a deeper neural net with 3 hidden layers of 9000 neurons each, hence performing better than a network with an entire additional layer. Furthermore, the best results produced in [5] uses a total of $27,000$ neurons whereas we have made use of only 18000 neurons in one experiment, and a maximum of 1500 neurons in all others. Our results are embellished with multiple runs on the data set with demonstrated insignificant standard deviation between MAEs computed for each run. Thus, we establish our claim of accomplishing impressive results via Parsimonious Computing. [2]

Acknowledgement

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

References

[1] Justin Lamb, Emily D Crawford, David Peck, Joshua W Modell, Irene C Blat, Matthew J Wrobel, Jim Lerner, Jean-Philippe Brunet, Aravind Subramanian, Kenneth N Ross, et al. The connectivity map: using gene-expression signatures to connect small molecules, genes, and disease. *science*, 313(5795):1929–1935, 2006.

[2] David Peck, Emily D Crawford, Kenneth N Ross, Kimberly Stegmaier, Todd R Golub, and Justin Lamb. A method for high-throughput gene expression signature analysis. *Genome biology*, 7(7):R61, 2006.

[3] Jeff Hasty, David McMillen, Farren Isaacs, and James J Collins. Computational studies of gene regulatory networks: in numero molecular biology. *Nature Reviews Genetics*, 2(4):268, 2001.

[4] Guibo Ye, Mengfan Tang, Jian-Feng Cai, Qing Nie, and Xiaohui Xie. Low-rank regularization for learning gene expression programs. *PloS one*, 8(12):e82146, 2013.

[5] Yifei Chen, Yi Li, Rajiv Narayan, Aravind Subramanian, and Xiaohui Xie. Gene expression inference with deep learning. *Bioinformatics*, 32(12):1832–1839, 2016.

[6] Snehanshu Saha, Nithin Nagaraj, Archana Mathur, and Rahul Yedida. Evolution of novel activation functions in neural network training with applications to classification of exoplanets. *CoRR*, abs/1906.01975, 2019.

[7] Ron Edgar, Michael Domrachev, and Alex E Lash. Gene expression omnibus: Ncbi gene expression and hybridization array data repository. *Nucleic acids research*, 30(1):207–210, 2002.

[8] Hongyu Diao, Xinxing Li, Sheng Hu, and Yunhui Liu. Gene expression profiling combined with bioinformatics analysis identify biomarkers for parkinson disease. *PloS one*, 7(12):e52319, 2012.

[9] Ainhoa Perez-Diez, Andrey Morgun, and Natalia Shulzhenko. Microarrays for cancer diagnosis and classification. In *Microarray technology and cancer gene profiling*, pages 74–85. Springer, 2007.

[10] Mehdi Pirooznia, Jack Y Yang, Mary Qu Yang, and Youping Deng. A comparative study of different machine learning methods on microarray gene expression data. *BMC genomics*, 9(1):S13, 2008.

[11] Mukesh Bansal, Vincenzo Belcastro, Alberto Ambesi-Impiombato, and Diego Di Bernardo. How to infer gene networks from expression profiles. *Molecular systems biology*, 3(1), 2007.

[12] Feng Gao, Wei Wang, Miaomiao Tan, Lina Zhu, Yuchen Zhang, Evelyn Fessler, Louis Vermeulen, and Xin Wang. Deepcc: a novel deep learning-based framework for cancer molecular subtype classification. *Oncogenesis*, 8(9):1–12, 2019.

[13] Leslie N Smith. Cyclical learning rates for training neural networks. In *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 464–472. IEEE, 2017.

[14] Leslie N Smith and Nicholay Topin. Super-convergence: Very fast training of neural networks using large learning rates. In *Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications*, volume 11006, page 1100612. International Society for Optics and Photonics, 2019.

[15] Sihyeon Seong, Yegang Lee, Youngwook Kee, Dongyoon Han, and Junmo Kim. Towards flatter loss surface via nonmonotonic learning rate scheduling. In *UAI*, pages 1020–1030, 2018.

[16] S. Saha. Differential equations: A structured approach. 2011.

[17] Stefan Banach. Sur les opérations dans les ensembles abstraits et leur application aux équations intégrales. *Fundamenta Mathematicae*, 3(1):133–181, 1922.

[2]Our code can be found at: *https://github.com/ShaileshSridhar2403/Parsimony-Computing-Gene-Expression-Inference-with-LALR*