Deep Recurrent Neural Networks for Sequential Phenotype Prediction in Genomics

Farhad Pouladi
Faculty of Applied Sciences
of Communications,Tehran, Iran
UMTS Project Partners bv. (SPAA05),
Waalwijk, Netherlands
pouladi@ictfaculty.ir

Hojjat Salehinejad
Farnoud Data Communication Company,
Kerman, Iran
hojjat.salehinejad@uoit.net

Amir Mohammad Gilani
Mobile Communication Company
of Iran (MCI), Tehran, Iran
am.gilani@mci.ir

***Abstract*—In analyzing of modern biological data, we are often dealing with ill-posed problems and missing data, mostly due to high dimensionality and multicollinearity of the dataset. In this paper, we have proposed a system based on matrix factorization (MF) and deep recurrent neural networks (DRNNs) for genotype imputation and phenotype sequences prediction. In order to model the long-term dependencies of phenotype data, the new Recurrent Linear Units (ReLU) learning strategy is utilized for the first time. The proposed model is implemented for parallel processing on central processing units (CPUs) and graphic processing units (GPUs). Performance of the proposed model is compared with other training algorithms for learning long-term dependencies as well as the sparse partial least square (SPLS) method on a set of genotype and phenotype data with 604 samples, 1980 single-nucleotide polymorphisms (SNPs), and two traits. The results demonstrate performance of the ReLU training algorithm in learning long-term dependencies in RNNs.**

***Index Terms*—genotype imputation; phenotype prediction; recurrent neural networks; sequence learning;**

I. Introduction

Machine learning is a practical approach to deal with real world challenges such as to model neck pain and motor training induced plasticity [10], [12]. Such methods have been widely used to solve difficult optimization problems [6], [7]. Opposition based learning is one example which has achieved successful results in medical image processing and optimization problems [25], [20], [11]. The genome-wide association (GWA) studies have discovered many convincingly replicated associations for complex human diseases using high-throughput single-nucleotide polymorphism (SNP) genotypes [1], [2]. The genotype imputation has been used for fine-map associations and facilitates the combination of results across studies [1]. The issue of missing genotype data and its imputation implies creating individualistic genotype data [2]. Impact of even small amounts of missing data on a multi-SNP analysis is of great importance for the complex diseases research [2]. There are several programs such as BEAGLE [3], MaCH [4], and IMPUTE2 [5], which provide imputation capability of untyped variants.

The sparse partial least squares (SPLS) and least absolute shrinkage and selection operator (LASSO) methods are well-known for simultaneous dimension reduction and variable selection [8], [9]. The LASSO is a shrinkage and selection method for linear regression, which attempts to minimize an error function. This function is typically the sum of squared errors with a bound on the sum of the absolute values of the coefficients [9]. The partial least squares (PLS) regression is used as an alternative approach to the ordinary least squares (OLS) regression method [8]. The SPLS method is the sparse version of PLS method, which simultaneously works to achieve good predictive performance and variable selection by producing sparse linear combinations of the original predictors [8].

In general, matrix factorization is a technique to decompose a matrix for multivariate data into two matrices with F latent features [15]. Many matrix factorization techniques have been proposed to increase its performance, such as non-negative [16], sparse [17], non-linear [14], and kernel-based approaches [13]. A kernel non-negative matrix factorization method is proposed for feature extraction and classification of micro-array data in [13]. Performance evaluation of this method for eight different gene samples has showed better performance over linear as well as other well-known kernel-based matrix factorization approaches.

A sparse matrix factorization method has been proposed for tumor classification using gene expression data, [17]. In this approach, the gens are selected using a sparse matrix factorization method and then the features are extracted to be fed into a support vector machine (SVM) for tumor samples classification. It is reported that the performance results have been improved versus the non-sparse matrix factorization techniques. The artificial neural network (ANN) is another successful machine learning approach for prediction and classification applications [29]. As an example, a feed-forward ANNs model and a Bayesian approach are utilized to impute missing genotype data of SNPs in [2]. Sequence modelling is one of the most areas in machine learning. This is due to the fact that a large class of phenomenal and data around us is made of sequences of data with particular patterns. Some examples are retail data, speed recognition, natural language modelling, music generation and genotype data for medical applications. With the great practical advances in deep learning, this state-of-the-art machine learning technique is the key for many problems in science and engineering. Recurrent neural network (RNN), due to its recurrent connections is considered

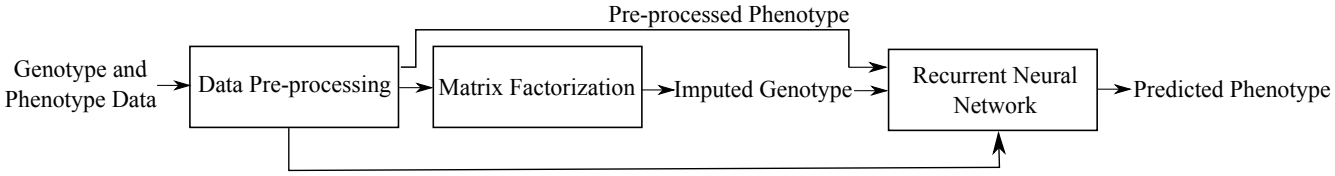


Fig. 1: The proposed system model.

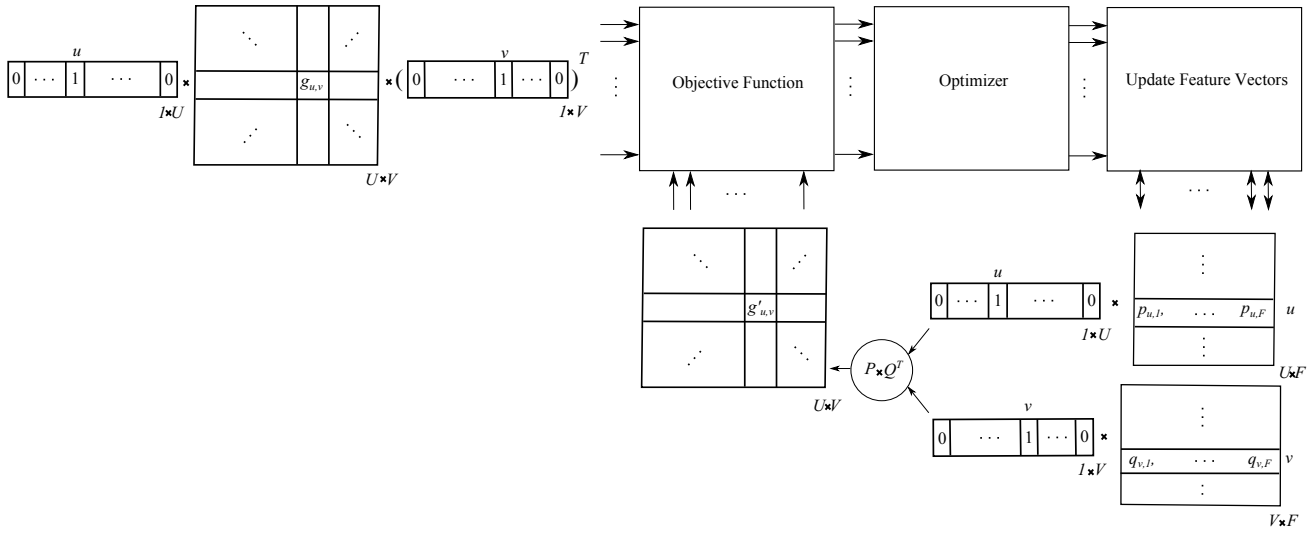


Fig. 2: The matrix factorization model.

as a subcategory of deep learning methods. This powerful model is capable of learning temporal patters and in sequential data. The power of RNN arises from its hidden state, which works as the "memory" of system to remember past important features for the future decision makings. The hidden state is consisted of high-dimensional non-linear dynamics which enables modelling any phenomena, if trained well [31], [30].

In this paper, we are proposing a new model for missing genotype imputation and phenotype prediction using matrix factorization and RNNs. In this model, a simple but efficient matrix factorization method is used for missing genotype imputation. Then, the imputed genotypes are used along the sequence of available phenotype data to train our RNN with the recently developed ReLU learning approach. In order to evaluate performance of the ReLU approach in learning long-term dependencies in phenotype data, it is compared with the LSTM-RNN and SRNN approaches.

In the next section, the data structure of the dataset used for the experiments is described. In Section III, the methodologies based on the matrix factorization technique and DRNN is discussed in detail. The experimental results as well are comparative analysis are provided in Section IV. Finally, the paper is concluded in Section V and some guidelines are further developments are provided.

II. Data Structure

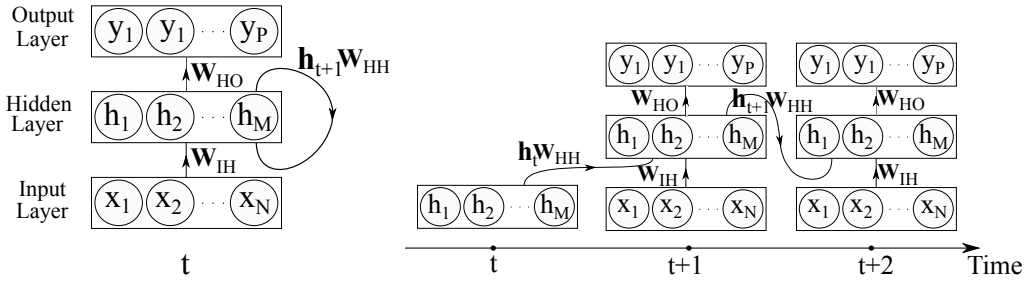
For the experiments, we are using a set of data provided for our research by Afzalipour research hospital. The genotype data contains genotypes of 1980 SNPs for 604 observations and the phenotype data provides measurements of two phenotype, called trait 1 and trait 2. Out of 1980 SNPs provided in the genotype data, 5% contain missing genotypes. The percentage of observations with missing genotypes for each SNP varies from 1 to 25. For each trait, 30 randomly selected observations have missing values.

III. Methodology

In order to deal with the missing genotype and phenotype problem, we are utilizing the matrix factorization (MF) and RNNs techniques to fit prediction models as in Figure 1. To do so, after data pre-processing, the genotype dataset with missing values is imported into the MF system to predict the missing genotype values. By having the estimated genotype dataset and corresponding phenotypes, the RNN is utilized in a supervised manner to train a network model for prediction of phenotypes, based on the known genotype-phenotype pairs. Each stage is described in details in the following subsections.

A. Data Pre-processing

In general, the SNP genotypes (AA, BB, AB, or Null) are denoted with integer numbers for computational purposes, however, some programs may be able to work with this AA/AB/BB format directly. The data pre-processing step is an opportunity to clean data, remove noise, and translate the genotype data and indicate/distinguish missing values from the available data.



(a) Fodled SRNN. (b) Unfolded SRNN through time.

Fig. 3: A simple recurrent neural network (SRNN) and its unfolded structure through time. To keep the figure simple, biases are not shown.

B. Matrix Factorization Model

The proposed MF structure for genotype data imputation is presented in Figure 2. In this model we consider U number of samples and V number of SNPs. Therefore, the genotype data is structured as a $U \times V$ matrix, called $G_{U\times V}$. The objective of MF technique is to estimate two matrices, $P_{U\times F}$ and $Q_{V\times F}$ with F latent features, such that their product $G^{'}_{U\times V}$ estimates $G_{U\times V}$ as:

$$G \approx G^{'} = P \times Q^{T}, \tag{1}$$

where each element of the genotype matrix $G^{'}$ is computed by using the dot product such as:

$$g^{'}_{u,v} = \sum_{f=1}^{F} p_{u,f} q_{f,v}. \tag{2}$$

In order to find the best values for the matrices P and Q, we need to minimize the objective function which describes the difference between the G and $G^{'}$ genotype matrices [15]. To do so, the gradient descent algorithm is utilized as the optimizer in Figure 2 to update the feature matrices P and Q iteratively.

The above procedure is illustrated in pseudocode as in Algorithm 1. As it is demonstrated, the parameters are set in the initialization step and random values in range $[a, b]$ are allocated to the feature matrices P and Q, [15]. Based on the availability of each genotype such as $G_{u,v} \neq \emptyset$ for all $\{u, v\} \in \{\{1, ..., U\}, \{1, ..., V\}\}$, the estimated genotype matrix $G^{'}_{u,v}$ is computed. The objective function is then formulated with respect to P and Q as:

$$min\ \ e(G, P, Q)^2 = \sum_{u=1}^{U}\sum_{v=1}^{V}(G_{u,v} - G^{'}_{u,v})^2 + \frac{\beta}{2}(\|P\|_F + \|Q\|_F) \tag{3}$$

where Forbenius norms of P and Q are used for regularization under control of parameter β to prevent over-fitting of model by penalizing it with extreme parameter values [15]. Normally β is set to some values in the range of 0.02, such that P and Q can approximate G without having to contain large numbers. The feature matrices of P and Q are updated as:

Algorithm 1 Matrix factorization

Initialization
$P \leftarrow rand([a, b])$
$Q \leftarrow rand([a, b])$
for each epoch **do**
  if $G_{u,v} \neq \emptyset\ \forall\ \{u, v\} \in \{\{1, ..., U\}, \{1, ..., V\}\}$ **then**
    $G^{'}_{u,v} \leftarrow P \times Q^{T}$
    $e^2 \leftarrow \sum_{u=1}^{U}\sum_{v=1}^{V}(G_{u,v} - G^{'}_{u,v})^2 + \frac{\beta}{2}(\|P\|_F + \|Q\|_F)$
    $P \leftarrow P - \alpha\frac{\partial e^2}{\partial P}$
    $Q \leftarrow Q - \alpha\frac{\partial e^2}{\partial Q}$

$$P^{updated} = P - \alpha\frac{\partial e^2}{\partial P} \tag{4}$$

and

$$Q^{updated} = Q - \alpha\frac{\partial e^2}{\partial Q} \tag{5}$$

respectively, where α represents the learning rate and is practically set to 0.0001.

C. Recurrent Neural Network with Rectified Linear Unit Model

The utilized RNN in the proposed model in Figure 1 is consisted of input, hidden, and output layers, where each layer is consisted of corresponding units. The input layer is consisted of N input units, where its inputs are defined as a sequence of vectors through time t such as $\{..., \mathbf{x}_{t-1}, \mathbf{x}_t, \mathbf{x}_{t+1}...\}$ where $\mathbf{x}_t = (x_1, x_2, ..., x_N)$. In a fully connected RNN, the inputs units are connected to hidden units in the hidden layer, where the connections are defined with a weight matrix $\mathbf{W}_{IH}$. The hidden layer is consisted of M hidden units $\mathbf{h}_t = (h_1, h_2, ..., h_M)$, which are connected to each other through time with recurrent connections. As it is demonstrated in Figure 3b, the hidden units are initiated before feeding the inputs. The hidden layer structure defines the state space or "memory" of the system, defined as

$$\mathbf{h}_t = f_H(\mathbf{W}_{IH}\mathbf{x}_t + \mathbf{W}_{HH}\mathbf{h}_{t-1} + \mathbf{b}_h) \tag{6}$$

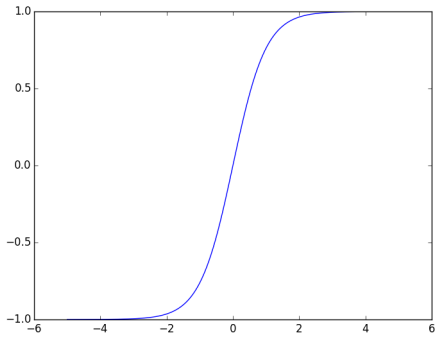


Fig. 4: The “tanh” activation function.

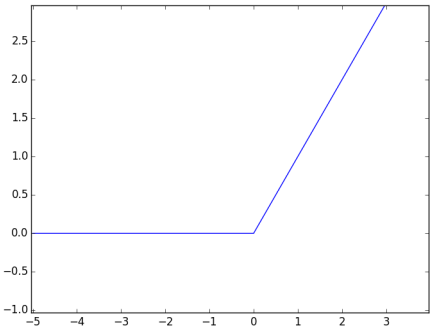


Fig. 5: The “rectified linear unit (ReLU)” activation function.

where $f_H(.)$ is the hidden layer activation function and $\mathbf{b}_h$ is the bias vector of the hidden units. The hidden units are connected to the output layer with weighted connections $\mathbf{W}_{HO}$. The output layer has P units such as $\mathbf{y}_t = (y_1, y_2, ..., y_P)$ which are estimated as

$$\mathbf{y}_t = f_O(\mathbf{W}_{HO}\mathbf{h}_t + \mathbf{b}_0) \tag{7}$$

where $f_O(.)$ is the output layer activations functions and $\mathbf{b}_0$ is the bias vector.

Learning long term dependencies in RNNs is a difficult task [30]. This is due to two major problems which are vanishing gradients and exploding gradients. The long-short-term memory (LSTM) method is one of the popular methods to overcome the vanishing gradient problem. A recent proposed method suggests that proper initialization of the RNN weights with rectified linear units has good performance in modeling long-range dependencies [30]. In this approach, the model is trained by utilizing back-propagation through time (BPTT) technique to compute the derivatives of error with respect to the weights. The reported performance analysis show that this method has comparable results in comparision to the LSTM method, with much less complexity.

In this model, each new hidden state vector is inherited from the previous hidden vector by copying its values, adding the effects of inputs, and finally, replacing negative state values by zero. In other words, this means that the recurrent weight matrix is initialized to an identity matrix and the biases are set to zero. This procedure is in fact replacing the ”tanh” activation function (Figure 4) with a rectified linear unit (ReLU)(Figure 5). The ReLU in fact is modelling the behaviour of LSTM. In LSTM, the gates are set in a way that there is no decay to model long-term dependencies. In ReLU, when the error derivatives for the hidden units are back-propagated through time they remain constant provided no extra error-derivatives are added [30].

IV. Experimental Results

The proposed model is implemented for parallel processing using Theano in Python [21], [18], [19]. In this section, we are presenting the performance result from comparison between simple RNN, LSTM, and ReLU training methods for phenotype sequence prediction. The ReLU method is then compared with the well-known sparse partial least square (SPLS) method.

TABLE I: Parameter setting for the experiments.

Parameter Name	Parameter Descritption	Parameter Value
α	Learning Rate	0.001
β	Regularization Control	0.02
N_{EP}	Number of Epochs	5000
U	Number of Samples	604
V	Number of SNPs	1980
F	Number of Features	400
eta_1	eta for Trait 1	0.2
eta_2	eta for Trait 2	0.2
K_1	K for Trait 1	3
K_2	K for Trait 2	4
$[a, b]$	Boundary values for P and Q	[0,1]

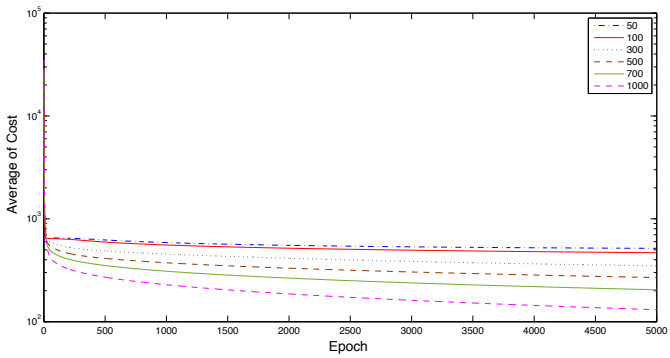
A. Parameter Setting

As it is described in Section II, it is assumed that the genotypes data is either missing or observed. Therefore, the observed genotypes are represented as $G_{u,v} \in \{0, 1, 2\}$ and the missing (null) data is represented as $G_{u,v} = 5$ for the experiments [1]. The genotype dataset $G_{U,V}$ is consisted of $V = 1980$ SNPs for $U = 604$ observations. Parameter setting for all the experiments are presented in Table I adapted from the literature, [14], [19], [29], unless a change is mentioned. As recommended in the literature, the 10-fold cross-validation is used to tune the parameters eta and K for the SPLS algorithm using the SPLS package in R programming language [**?**], [26], [27], [8]. The parameter tuning is conducted separately for each provided trait in the phenotype dataset for $1 \leq K \leq 10$. The optimal values are provided in Table I.

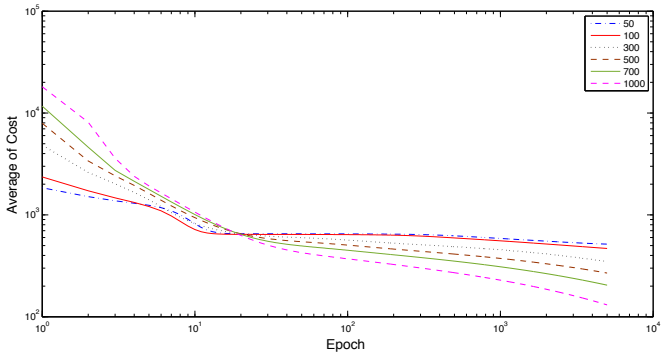
Due to small size of data samples, $\%80$ of available genotype and phenotype data is used for training the ANN, $\%10$ for validation, and $\%10$ for testing. Regarding the SPLS algorithm, $\%80$ of the provided data is considered as training data and $\%20$ as test data.

B. Simulation Results Analysis

In this subsection, performance results of the proposed method is presented and compared with the SPLS method for the described dataset in Section II. Performance of the methods is evaluated by measuring the correlation between the original



(a) Logarithmic view for average of costs.



(b) Logarithmic view for average of costs and epochs.

Fig. 6: Average of costs versus epochs for genotype imputation using matrix factorization technique with different number of latent features.

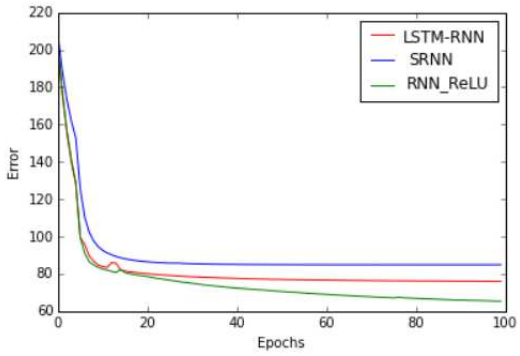


Fig. 7: Training error comparison between SRNN, LSTM-RNN, and ReLU-RNN.

genotype and phenotype values with the corresponding predicted values.

The average of cost for fitting the MF model for different epochs is presented in Figure 6. In order to have a deeper look in details, the figure is presented in linear and logarithmic scales. The measure to compute the average of cost is the mean-square error, which is the average of squared difference between targets and the output of the MF model. As the results show, the more number of latent features F results in less average of cost. In addition, the model is trying to fit

TABLE II: Performance results of the matrix factorization technique success for different latent features F.

F	Missing Genotype	Genotype Matrix Construction
50	%52.45	%71.47
100	%65.16	%76.11
300	%67.53	%88.11
500	%68.80	%95.72
700	%70.88	%96.91
1000	%72.48	%97.56

TABLE III: Performance results of the ReLU-RNN for successful phenotype traits prediction.

Trait	Mode	Original Genotype	Missing Genotype
1	Train	%82.75	%80.53
	Validation	%86.56	%80.22
	Test	%84.66	%80.25
2	Train	%80.62	%75.01
	Validation	%85.25	%80.54
	Test	%79.58	%78.29

TABLE IV: Performance results in terms of correlation between the predicted and original phenotype data for the sparse partial least squares (SPLS) method.

Trait	Mode	Original Phenotype	Missing Phenotype
1	Train	%75.10	%69.52
	Test	%54.56	%51.53
2	Train	%80.53	%72.85
	Test	%69.82	%56.42

better that before in each epoch, however, after epoch 110 the progress is not very significant. The performance results for the success percentile in missing genotype data and success percentile in the whole genotype data construction is provided in Table II. There is a trade-off between the number of features and performance. This is due to the fact that increasing the number of features increases the computational complexity.

In Table III, performance of the ReLU-RNN for the genotype prediction is presented. As it is showed, the ReLU method has better training performance comparing to the results of the SPLS method in Table IV. The missing genotype represents the error for the test dataset in percentile while the original genotype represents the training error values in percentile. Since the training dataset has been seen by the model during training, it is reasonable to see that performance of the methods for the training dataset is better than the unseen test data.

In Figure 7, the training error of the SRNN, LSTM-RNN, and ReLU-RNN algorithms are compared. For better illustration, the first 100 training epochs are presented. As the results show, the SRNN algorithm has more training error than the LSTM-RNN method. This is while the ReLU-RNN approach has the least training error comparing to LSTM-RNN. At the early epochs, we see that the LSTM and ReLU are almost at the same training loss, however, the ReLU achieves less error in further epochs.

V. Conclusion and Future Works

In this paper, a novel model is proposed which utilizes matrix factorization and deep recurrent neural networks (DRNN) for genotype imputation and phenotype sequences prediction. Sine we are interested in keeping track of sequences with long-term dependencies in genomics, the state-of-the-art recited linear unit learning method is used.

The performance results show the with the ReLU methods has a better performance in training comparing to the LSTM-RNN and simple RNN methods. The ReLU learning methods also has less computational complexity comparing to the LSTM method. For future research, it is interesting to analyze other recent advances in deep learning for genotype-phenotype application; particularly that these algorithms are moving toward more simple designs which is suitable for big data application.

References

[1] J. Marchini and B. Howie, "Genotype imputation for genome-wide association studies," *Nature Reviews Genetics*, vol. 11, pp. 499-511, 2010.

[2] Y. V. Sun and S. LR. Kardia1, "Imputing missing genotypic data of single-nucleotide polymorphisms using neural networks," *European Journal of Human Genetics*, vol. 16, pp. 487-495, 2008.

[3] SR. Browning and BL. Browning, "Rapid and accurate haplotype phasing and missing-data inference for whole-genome association studies by use of localized haplotype clustering," *American journal of human genetics*, vol. 81, pp. 10841097, 2007.

[4] Y. Li, CJ. Willer, J. Ding, P. Scheet, and GR. Abecasis, "MaCH: using sequence and genotype data to estimate haplotypes and unobserved genotypes," *Genetic Epidemiology*, vol. 34, pp. 816834 2010.

[5] BN. Howie, P. Donnelly, and J. Marchini, "A flexible and accurate genotype imputation method for the next generation of genome-wide association studies," *PLoS genetics 5: e1000529*, 2009.

[6] H. Salehinejad and S. Talebi. "Dynamic fuzzy logic-ant colony system-based route selection system," *Applied Computational Intelligence and Soft Computing*, 2010.

[7] H. Salehinejad, et al. "Micro-differential evolution with vectorized random mutation factor," *Evolutionary Computation, 2014 IEEE Congress on (CEC)*, pp. 2055-2062, 2014.

[8] H. Chun and S. Kele, "Sparse partial least squares regression for simultaneous dimension reduction and variable selection," J R Stat Soc Series B Stat Methodol, vol. 72, no. 1, pp. 3-25, 2010.

[9] R. Tibshirani, "Regression shrinkage and selection via the lasso," J. Royal. Statist. Soc B., Vol. 58, No. 1, pp. 267-288, 1996.

[10] J. Baarbé, et al. "A novel protocol to investigate motor training-induced plasticity and sensorimotor integration in the cerebellum and motor cortex," *Journal of neurophysiology*, 111.4, pp. 715-721, 2014.

[11] H. Salehinejad, S. Rahnamayan, and H. R. Tizhoosh, "Type-II opposition-based differential evolution," *Evolutionary Computation, 2014 IEEE Congress on (CEC)*, pp. 1768-1775, 2014.

[12] J. Daligadu, et al. "Alterations in cortical and cerebellar motor processing in subclinical neck pain patients following spinal manipulation," *Journal of manipulative and physiological therapeutics*, 36.8, pp. 527-537, 2013.

[13] Y. Li and A. Ngom, "A New Kernel Non-Negative Matrix Factorization and Its Application in Microarray Data Analysis," in *Proc. IEEE Symposium on Computational Intelligence in Bioinformatics and Computational Biology*, pp. 371-378, 2012.

[14] P. Binbin, L. Jianhuang, and C. Wen-Sheng, "Nonlinear nonnegative matrix factorization based on Mercer kernel construction," *Pattern Recognition*, vol. 44, Iss. 1011, pp. 2800-2810, 2011.

[15] R. Gribonval and K. Schnass, "Dictionary IdentificationSparse Matrix-Factorization via l_1-Minimization," *IEEE Transactions on Information Theory*, vol. 56, no. 7, pp. 3523-3539, 2010.

[16] D.D. Lee and S. Seung, "Learning the parts of objects by non-negative matrix factorization," *Nature*, vol. 401, pp. 788-791, 1999.

[17] C. Zheng and et. al., "Tumor Classification Based on Non-Negative Matrix Factorization Using Gene Expression Data," *IEEE Transactions on NanoBioscience*, vol. 10, pp. 86-93, 2011.

[18] E. Jones, E. Oliphant, P. Peterson, and et. al. "SciPy: Open Source Scientific Tools for Python," 2001, http://www.scipy.org/ [Online; accessed 2015-04-07]

[19] J. Bergstra, O. Breuleux, F. Bastien, and et. al., "Theano: A CPU and GPU Math Expression Compiler," in *Proceedings of the Python for Scientific Computing Conference (SciPy)*, Austin, TX, 2010.

[20] S. Rahnamayan, and H. R. Tizhoosh, "Image thresholding using micro opposition-based differential evolution (Micro-ODE)." In Evolutionary Computation, IEEE Congress on, pp. 1409-1416, 2008.

[21] Python Software Foundation. Python Language Reference, version 2.7. Available at http://www.python.org.

[22] MATLAB and Artificial Neural Networks Toolbox, Release 2012b, The MathWorks, Inc., Natick, Massachusetts, United States.

[23] Marquardt, D., "An Algorithm for Least-Squares Estimation of Nonlinear Parameters," *SIAM Journal on Applied Mathematics*, Vol. 11, No. 2, pp. 431441, June 1963.

[24] A. Ranganathan, "The Levenberg-Marquardt Algorithm," June 2004.

[25] S. Rahnamayan, H. R. Tizhoosh, and M. Salama, "Opposition-based differential evolution," Evolutionary Computation, IEEE Transactions on 12, no. 1, pp.64-79, 2008.

[26] D. Chung and et. al., "An Introduction to the 'spls' Package, Version 1.0," June 10, 2012.

[27] D. Chung and et. al., "Package spls," R programming language, February 20, 2015.

[28] F. Bastien, P. Lamblin, R. Pascanu, J. Bergstra, I. Goodfellow, A. Bergeron, N. Bouchard, D. Warde-Farley and Y. Bengio. ?Theano: new features and speed improvements?. NIPS 2012 deep learning workshop.

[29] S. Mahdavi-Jafari, H. Salehinejad, and S. Talebi. "A Pistachio Nuts Classification Technique: An ANN Based Signal Processing Scheme." In Computational Intelligence for Modelling Control and Automation, 2008 International Conference on, pp. 447-451. IEEE, 2008.

[30] Le, Q. V, Jaitly, N., & Hinton Google, G. E. "A Simple Way to Initialize Recurrent Networks of Rectified Linear Units," pp. 1?9, 2015, arXiv:1504.00941v2 [cs.NE] 7 Apr 2015.

[31] N. Boulanger-Lewandowski, Y. Bengio, and P. Vincent, "Modeling Temporal Dependencies in High-Dimensional Sequences: Application to Polyphonic Music Generation and Transcription," Appearing in Proc. ICML, 2012.