Encoding Gene Expression Using Deep Autoencoders for Expression Inference

Raju Bhukya
Department of Computer Science and Engineering, National Institute of Technology, India

Abstract: *Gene expression of an organism contains all the information that characterises its observable traits. Researchers have invested abundant time and money to quantitatively measure the expressions in laboratories. On account of such techniques being too expensive to be widely used, the correlation between expressions of certain genes was exploited to develop statistical solutions. Pioneered by the National Institutes of Health Library of Integrated Network-Based Cellular Signature (NIH LINCS) program, expression inference techniques has many improvements over the years. The Deep Learning for Gene expression (D-GEX) project by University of California, Irvine approached the problem from a machine learning perspective, leading to the development of a multi-layer feedforward neural network to infer target gene expressions from clinically measured landmark expressions. Still, the huge number of genes to be inferred from a limited set of known expressions vexed the researchers. Ignoring possible correlation between target genes, they partitioned the target genes randomly and built separate networks to infer their expressions. This paper proposes that the dimensionality of the target set can be virtually reduced using deep autoencoders. Feedforward networks will be used to predict the coded representation of target expressions. In spite of the reconstruction error of the autoencoder, overall prediction error on the microarray based Gene Expression Omnibus (GEO) dataset was reduced by 6.6%, compared to D-GEX. An improvement of 16.64% was obtained on cross platform normalized data obtained by combining the GEO dataset and an RNA-Seq based 1000G dataset.*

Keywords: *Deep autoencoder, gene expression, internal covariance shift, machine learning, MLP, PCA.*

Received March 5, 2019; accepted April 13, 2020
https://doi.org/10.34028/iajit/18/5/1

1. Introduction

The overview of an organism's cellular function can be obtained by collectively measuring the expression of its genes. Gene expression profiling is the area in molecular biology that performs this key functionality [9]. The expression of a gene is the quantity that describes the mapping from the organism's genotype to its phenotype. Proteins and RNAs are synthesized using this information. Several clinical techniques are employed to directly measure the expression of a gene. Deoxyribonucleic Acid (DNA) microarrays measure the relative activity of previously identified target genes [2]. Sequence based techniques like Ribonucleic Acid (RNA) -Seq provide information on the sequences of genes in addition to their expression level. Thus, gene expression can be viewed as a quantitative representation of an organism's genetic code. Regulation of gene expression is crucial for its overall development [1]. Expression is profiled under a variety of conditions such as diseases, drug intake and induced genetic mutations, to understand the inner workings of the organism [2].

The methods to measure the gene expression can be broadly classified under two categories, clinical and statistical. The Connectivity Map (CMap) project is an example of the clinical technique [14]. Even with the advent of latest technological innovations, whole genome expression profiling is too expensive to be used in an academic set up. The CMap project in the initial phase was able to perform just 564 genome wide expression profiles using the Affymetrix GeneChip microarrays [2]. On the contrary, statistical methods rely on inferring the expression of several genes using the directly measured expressions of certain carefully selected genes. These techniques rely on the high correlation between the thousands of genes in the human genome. The Library of Integrated Network-based Cellular Signatures (LINCS) program initiated by the National Institute of Health, USA employed this observation to formulate a cost effective alternative to whole genome expression profiling [17]. They used Principal Components Analysis (PCA) to discover that about 1000 genes explained close to 80% of the variance in the genome expression. The 978 relevant genes that constituted this set were termed as landmark genes. L1000 Luminex bead technology was used to measure the expression of this select set of genes [17]. The landmark expression signature was considered to accurately represent the cellular state at any given time. Using this information, the expressions of the rest were inferred by Linear Regression (LR). The unmeasured inferred genes were termed as target genes [2].

However, the relative number of the target genes (about 21000) with respect to the landmark genes (978) vexed the researchers. The LINCS program assumed linear relationship between the target and landmark gene expressions, and thus employed multi-task LR to predict the expressions of the target genes. Effectively, they had to build approximately 21000 models, each one fitted for a specific target gene, in order to profile the genome wide gene expression. The predicted values were compared with the expression profiles from the L1000 project. This linear model had the advantage of being highly scalable, but it failed to capture the non-linear relationships between some genes. Poor scalability of non-linear techniques like kernel machines steered the researchers away from such alternatives [2]. Researchers at the University of California, Irvine (UCI) decided that a machine learning approach would be ideal for expression profiling [2]. Their argument was that it would enhance the scalability and ensure better data representation. The complex hierarchical non linear relationships between genes were captured using deep learning [7, 8]. Deep learning, also called hierarchical learning uses multiple layers of abstraction to learn complex data representations [23]. The study at UCI culminated in the formulation of D-GEX, which is a multi-task multi-layer feed-forward neural network. The best performing model with 5 layers (1 input layer, 3 hidden layers of 9000 nodes each, 1 output layer) was reported to be 15% more accurate than the LR model [2].

The D-GEX project has a handful of drawbacks. Some are general to any deep neural network, while the high dimensionality of the human genome dataset poses a significant computational inefficiency [26]. Theoretically, deeper networks are able to capture complex non-linear relationships within the data. But as the depth of the network grows, it becomes increasingly difficult to train the network because the errors propagated back towards the lower layers keeps on decreasing [3]. This is known as the vanishing gradient problem in deep neural networks [22]. Thus there is little or no change in weights of the first few layers. Hence, search for better results in deeper networks is hindered by this bottleneck. The dimensions the human genome expression data is 978x21290 (978 landmarks versus 21290 targets) [23]. Large number of features results in extremely slow training [13]. In addition to that, it is much harder to find a good solution when the number of features is large. This particular hiccup is known as the curse of dimensionality. High dimensional datasets are prone to become too sparse, which means that two randomly chosen instances are highly likely to be very far away from each other [3]. Previously unseen instances may diverge from the instances that the network was trained on and thus predictions will be inaccurate. Having greater dimensions ushers in the additional risk of overfitting the data [6]. Ideally, large number of training instances can account for the high dimensionality, but increasing the sample size is not easy in this particular context of expression signatures [3]. Coming to the specifics of the D-GEX model, the hidden units are activated using the hyperbolic tangent function [2]. With its mean at 0, hyperbolic tangent is a better non linear activation function than logistic function. But the function saturates to +1 or -1 for higher positive and negative inputs respectively. Effectively, the derivatives are pulled down to zero and the vanishing gradient problem kicks in [3]. Another key drawback of D-GEX was that training was done separately for randomly segregated batches of target genes due to hardware limitations [2]. This in turn increases the overall error. Also, by partitioning the target genes randomly, they completely ignored possible correlation between them.

Here, we present an alternative way to profile human genome expression. Since the key bottleneck is the large number of target genes, we propose that a deep autoencoder can be used to encode the target expression into a concise representation [11]. A modified multi-layer feed-forward neural network, that keeps the vanishing gradient problem at bay, is used to perform the actual prediction by regression. In order to ascertain the performance improvement of this encoded model over D-GEX, we compared the prediction errors of both on the cross-quantiled data from the Microarray based Gene Expression Omnibus (GEO) data and RNA-Seq based 1000G data. Additionally, the model was trained on the original GEO data to efficiently predict entire target profiles (21290 target genes) using a single network.

2. Proposed Model

Similar to the D-GEX approach, gene expression inference is treated as a supervised learning problem. The learning scenario is akin to that of multi-task regression. Expression profiles that were generated by the LINCS program are used to train and test models built for the expression inference.

2.1. Dataset

Gene Expression Omnibus (GEO) is a publicly available database repository of high throughput gene expression data and hybridization arrays, chips, microarrays [23]. The GEO expression data was made publicly available by the LINCS program. It was curated by the Broad Institute from the publicly available GEO database. It consists of 112634 gene expression profiles from the Affymetrix microarray platform. The 978 landmark genes and 21290 target genes of the human genome are represented by 22268 probes per profile. The expression levels are normalized into the range 4-15 [2].

A supplementary Illumina RNA-Seq platform dataset, titled as 1000 Genomes (1000G) RNA-Seq expression data is also used for training. This dataset is not derived from the Affymetrix Microarray platform and thus is used for testing the cross platform viability of the models developed. The expression levels of 462 profiles of lymphoblastoid cell line samples measured in RPKM format based on Gencode V12 annotations constitute 1000G [2].

Since the two datasets that are used for training are derived from different platforms, there may be probes that are present in one but not in the other. One Gencode probe may include multiple probes of the Microarray platform [2]. This means that not all genes covered in the GEO dataset is covered in the 1000G dataset. Using the gene ID and name information from the two datasets, probes that are common to both platforms were identified and the rest are pruned off. After pruning, we are left with 943 landmark genes and 9520 target genes, which exactly matched the feature set that was used to train D-GEX. To bring the two datasets in the same range, they are quantile normalized with standard normal distribution as reference [2]. This normalized GEO Expression data is used for training the models. The normalized 1000G dataset is set aside for testing and inferring the cross platform viability of the predictions. The original GEO dataset is used to build a complete model that can predict the entire target signature from the landmark expression.

2.2. Encoding Phase Using Auto Encoder

Applying Principal Components Analysis (PCA) exclusively for the target gene set, it was identified that significant correlation does exist between the target genes [2]. PCA projected the 21290 target genes into approximately 500 features as shown in Figure 1, with almost 99% of the variance retained. The reconstruction error was not that significant, either.

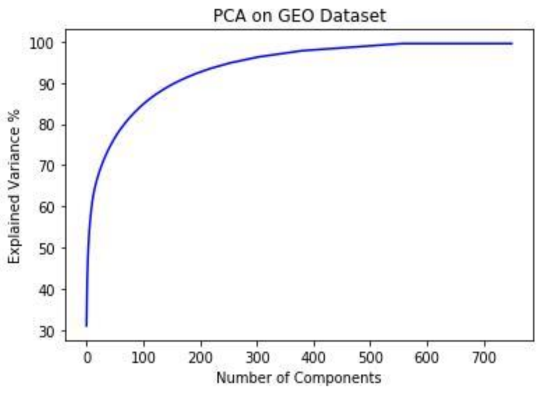

Figure 1. Principal components analysis on target expressions.

Thus, the target expression can effectively be represented using much less number of features, without compromising on the accuracy. PCA is used solely to validate our attempt to encode the target signature. The projections made by PCA are based on an assumption of linearity and thus cannot represent the complexity of correlation patterns [12, 25]. Instead, autoencoders are employed to form complex nonlinear encodings of the target expressions [21].

Autoencoders are artificial neural networks capable of learning efficient coding of the input data without any supervision [27]. Basically, an auto encoder learns to copy its inputs to its outputs. Effectively, the auto encoder learns the identity function, under certain constraints [3]. The constraints are defined by the architecture of the network. The number of neurons in the output layer must be equal to that of the number of inputs. By limiting the number of nodes in the hidden layer to a value less than the number of nodes in the input and output layers, the auto encoder learns concise representations of input data. Auto encoders learn by reducing the error in reconstructing the input data [27]. An auto encoder typically consists of two phases-an encoder phase and a decoder phase. The encoder phase learns the coding and the decoder phase reconstructs the inputs from the learnt coding. Stacking auto encoders one above another creates a deep auto encoder which can learn much more complex coding [3]. The cardinal principle of a deep auto encoder is that it is symmetrical about its central hidden layer. A technique called tying weights, where the weights of the encoder phase are reused for the decoder phase in the transposed form can simplify the training process [3].

We separately trained two auto encoders-one to encode the whole target set (21290 genes) and another to encode the target genes in cross-platform normalized dataset. The ideal length of the coding was experimentally determined to be 2000. This choice accounts for the non-linearity which was not considered by PCA. For ease of training, deep architectures with 3 hidden layers were selected. Different number of nodes were tried for the first and third layer (5000, 6000, 8000), with the innermost encoding layer consisting of 2000 neurons [2].

2.3. Using the Coded Representation

In order to test the usability and effectiveness of the coding, a multilayer feed forward network is constructed and trained on the landmark expressions and the target coding. Except the input layer nodes, each node in this predictor network is a neuron with a nonlinear activation function. Such networks are widely identified as Multi-Layer Perceptrons (MLP). Formally, an MLP can be considered as a non-linear transformation $f : R^M \rightarrow R^N$, where M is the size of the input vector and N is the size of the output vector [2]. Deep networks are formed by introducing more layers in between the input and output layers.

Two MLP networks were constructed-one each for the coding learnt by the two auto encoder models. The performance of these networks on both datasets was compared with the D-GEX model. A variety of changes were made to the MLP network of D-GEX.

The following subsections describe the nature and necessity of the alterations.

1) Activation: the hyperbolic tangent function used in D-GEX can capture the non-linearity, but it is prone to the vanishing gradients problem since it saturates to ±1 for high/low input values [2]. The middle ground between the linear activation and purely non-linear activation is attained using Restricted Linear Units (ReLU) [16]. But the derivative of ReLU is pulled to 0 for negative values. The Exponential Linear Unit (ELU) was derived as an alternative to ReLU in neural networks [5]. The exponential linear function is defined as

$$f(x)=\begin{cases} x & if\ x>0. \\ \alpha(e^{x}-1) & if\ x\leq 0. \end{cases} \tag{1}$$

ELUs are preferred widely since they are not vulnerable to the vanishing gradient problem. This is because they are continuous everywhere, even at the origin, as depicted in Figure 2.

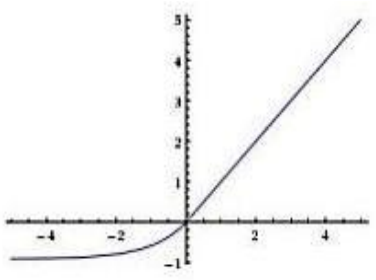

Figure 2. Exponential linear function.

ELUs have become the norm for non linear activation in recent times. The inner layers of our MLPs are formed using neurons with ELU activation. The output layer neurons are linear units since the output values are continuous [2].

2) Initialization: the vanishing gradient problem can be alleviated by the combination of a good activation function and initialization strategy [3]. Initializing weights randomly is a not preferred in deep neural networks because it can direct the cost function to local minima. An alternative is to sample from a normal distribution so that the total variance of the inputs of each layer and the total variance of the outputs are similar [15]. This ensures that the gradients would have the same variance before and after flowing through a layer. The mean and standard deviation is computed from the fan-in and fan-out of each. The He Initialization scheme for the Exponential Linear function samples from a normal distribution with mean 0 and standard deviation,

$$\sigma=\sqrt{\frac{4}{n_{input}+n_{output}}} \tag{2}$$

Or an uniform distribution between -r and +r where

$$r=\sqrt{\frac{12}{n_{input}+n_{output}}} \tag{3}$$

This variance scaling initialization scheme is used for the lower layers of our MLP. Since the network is used for regression, the initial values of output layer is sampled from a uniform distribution in the range [-1x10^{-4}, 1x10^{-4}] [2].

3) Optimization: weights in neural networks are updated by the standard back-propagation algorithm [4]. A variety of optimization techniques helps to improve the training speed. Gradient descent is the most common technique for optimization [3]. Momentum optimization method was derived as a way to accelerate the simple gradient descent optimization [3]. Gradient descent simply updates the weights by subtracting the gradient of the cost function $J(\theta)$ multiplied by the learning rate [3].

The key idea in Momentum optimization is to give sufficient weight to the previous gradients and not just the current derivative. Each epoch adds the local gradient to the momentum vector m and it updates the weights using this parameter [18].

$$m=\beta m+\alpha\frac{\partial}{\partial\theta}(J(\theta)), \tag{4}$$

Where β is momentum, and

$$\theta=\theta-m \tag{5}$$

Nesterov Accelerated Gradient (NAG) is a small variant of the momentum optimization, which has been empirically observed to be faster than the momentum technique [2]. The key idea is to measure the gradient at a position slightly ahead in the direction of the momentum, instead of measuring it locally [3].

$$\begin{gathered} m=\beta m+\alpha\frac{\partial}{\partial\theta}(J(\theta+\beta m)) \\ \theta=\theta-m \end{gathered} \tag{6}$$

With this slight modification, the convergence rate of the cost function is sufficiently improved [3]. Optimisation of both the MLP networks is done using NAG with the hyperparameter momentum set to 0.9.

4) Regularization: a bane of neural networks is their tendency to overfit the data on which they were trained [22]. Some techniques that are adopted while training to prevent this problem is collectively known as regularization techniques. Simple techniques like L1 and L2 regularization imposes certain restrictions on the range of values that the weights of the network can take. The most widely used regularization technique is dropout [3]. The approach is very simple - at every training step, every neuron (except the output neurons) is temporarily dropped out of the network with a probability p. The dropped neurons will be completely ignored during this training step, but

may become active in the next one [16]. Here, the hyperparameter p (called dropout rate) controls the number of neurons dropped out in each step. Dropout prevents the formation of small cliques of interdependent neurons by forcing them to interact with every other neuron at some training step [3]. A good generalized model is obtained as a result.

Another crude form of regularization is performed by pre-empting training when the performance on a validation set (not part of the training set) keeps on diminishing for several consecutive epochs [24]. This technique is termed as *early stopping*. The models are backed up at regular intervals and when a better model is not obtained for k epochs, training is terminated and the last saved model is restored for testing and further analysis [3]. Both early stopping and dropout regularization are used together while training the MLP networks. Here, k was set as 25 and a dropout rate of 10% (or a keep rate of 90%) was chosen.

5) Hidden Layer Configuration: there is no universal consensus on the ideal number of hidden layers and the number of neurons per hidden layer to be used while constructing a good neural network. A global solution is non-existent since the number varies from one application to another. Nevertheless, researchers have come up with certain workarounds where a quasi-ideal hidden layer configuration is deduced from known parameters. For instance, irrespective of the nature of the data, the number of neurons per hidden layer can be derived using the cardinality of the dataset and the sample size. One widely accepted observation is that a standard 2-layer feed-forward neural network with $\sqrt{(m+2)N} + 2\sqrt{\frac{N}{m+2}}$ neurons in the first hidden layer and $m\sqrt{(\frac{N}{m+2})}$ neurons in the second hidden layer can represent N distinct input samples with any desired precision [24]. Here, m is the number of output neurons. For our MLP, such a 2 hidden layer model consisting of 11850 and 11826 nodes respectively is adopted (our chosen m=2000 and N≈70000, the number of training samples).

6) Learning rate: the hyperparameter learning rate controls the step size in gradient descent [28]. Learning rate of the MLP is initialized to $5\text{x}10^{-4}$. For optimal learning, this value is programmatically tuned using a decay rate of 0.9 until it reaches a minimum of $1\text{x}10^{-5}$. A small subset of the training set is used for this tuning procedure.

2.4. Selecting Best Model

The cross platform normalized dataset is partitioned into 3 disjoint sets-train, validation and test. The dataset is initially divided into 20 clusters. From each cluster, a maximum of 4500 expression profiles are written onto *train* and 700 onto *validation*. The remnants of each cluster, if any, are added to *test*. The MLP has 943 nodes in the input layer and 2000 nodes in the output layer.

The same approach is used to partition the original GEO dataset into *GEO-train*, *GEO-val* and *GEO-test*. This MLP has 978 input nodes and 2000 output nodes. In each case, the performance on the validation set is monitored every 5 epochs. Training will be stopped early if the latest model performs worse than the model 25 epochs prior to it. The model with the least validation error is the best performing model. The performance is measured using the metric Mean Absolute Error (MAE) [2, 10]. For n samples,

$$MAE = \frac{\sum_{i=1}^{n} | y_i - \overline{y_i} |}{N}, \tag{7}$$

Where y_i s are the actual values and $\overline{yi}$ s are the predicted values [2, 10]. Different models are compared based on their MAE on the respective test sets. The 1000G dataset is used as a test set for the MLP trained on the quantile normalized data.

3. Experimental Results

In order to measure the effectiveness of this proposed approach, many networks are trained and tested. The first phase finds the ideal autoencoder configuration, subject to the hardware context. The second phase compares the prediction accuracy of our MLP model with the D-GEX model. All the models were trained and tested using the Tensorflow library in Python on an i7-3770 CPU@ 3.40GHz x8 [2].

3.1. Autoencoder Phase

In order to empirically determine the ideal number of nodes in the 3-hidden layer autoencoders, each of the candidate autoencoder architectures were trained for 150 epochs and saved after their reconstruction errors were recorded. From Table 1, it is evident that the 6000x2000x6000 configuration produced the best 3-layer encoding. It can also be observed that as the coding length increased, the reconstruction error also deceased. But, as the difference in nodes between the first layer and coding layer increase, the reconstruction error starts to grow slightly upwards. It is noted that as number of nodes in the coding layer increases the train and test time is gradually increasing. Attempts to increase the number of neurons in the coding layer further were stalled due to lack of primary memory. The encoded target expressions were written onto disk for training the prediction networks.

Table 1. Reconstruction error of autoencoders.

Number of nodes in layers 1 and 3	Number of nodes in coding layer	Reconstruction error	Training time (min)	Testing time (min)
5000	500	0.7837	20	4
6000	500	0.6554	23	5.3
8000	500	0.6632	26	5.4
5000	1000	0.6935	30	6
6000	1000	0.5912	33	6.5
8000	1000	0.6023	38	7.3
5000	2000	0.5346	40	8
6000	2000	0.4478	46	9.1
8000	2000	0.4671	54	11

3.2. MLP Phase

Training of the MLP networks were done for a maximum of 250 epochs. The test sets were run through the saved MLP models to obtain the predicted target encodings, which were decoded using the saved auto encoder models. The final decoded target expressions were used to compute the MAE of the models.

3.3. Performance of Models on Cross Platform Normalized Data

Two 3-layer feed forward networks with 9000 nodes in each layer were trained on *train* for 250 epochs, one for target genes 0-4760 and another for genes 4760-9520. The parameters were directly adapted from the best performing D-GEX model [2]. Activation of lower layers was done using hyperbolic tangent function. Output units had linear activation. Momentum technique was used for optimization. Dropout was performed with 10% dropout rate. MAE of both the networks was combined to produce the total error rate for genes 0-9520, which is described in Table 2. The performance of this D-GEX model was recorded and saved to compare with our models.

Then, an MLP network was constructed to predict the encoded target expressions. The activation function for lower layers was the exponential linear function, and linear function for the output layer. Early stopping regularization was used alongside dropout with 10% dropout rate. Optimization was done using NAG technique. The number of nodes in the first two hidden layers was 11850 and 11826, respectively [19]. This post-encoding prediction network is titled E-MLP.

Table 2. Comparison of models.

Model	Error on *validation*	Error on *test*
D-GEX	1.1342	1.2927
E-MLP	0.9833	**1.0775**

The decoded target predictions were compared to the original target expressions to derive the final error rate mentioned in Table 2. The encoded prediction model showcased 16.64% improvement in test error over D-GEX.

It can be observed that the reconstruction error of 0.4478 can be reduced substantially by training a deeper auto encoder, which would further pull down the overall predictive error. Figure 3 show that our model has clear superiority over D-GEX in terms of the validation errors.

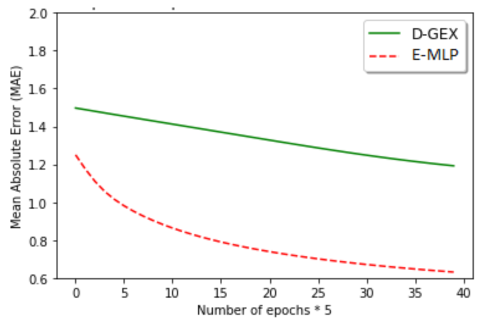

Figure 3. Comparison of validation errors.

The expressions of each target gene predicted by D-GEX and our encoded MLP were compared, which is represented in Figure 4. Our model produced better predictions for 99.87% of the target genes.

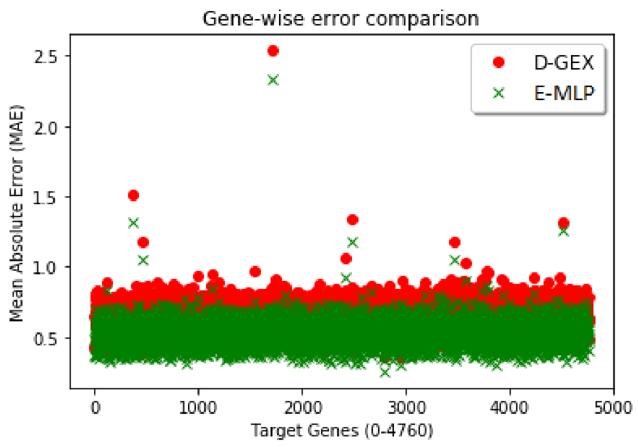

Figure 4. Comparison of gene-wise errors.

3.4. Performance on 1000G Dataset

Both the models were tested on the 1000G dataset, and the results are described in Table 3. Encoding the target before prediction boosted prediction accuracy over D-GEX by 49.23%.

Table 3. Predictive errors on 1000G.

Model	Genes (0-4760)	Genes (4760-9520)	Total
D-GEX	0.7756	0.7757	1.5513
E-MLP	0.3794	0.4082	0.7876

From Figure 5. it is clearly observable that the predictive error on 1000G by D-GEX was almost halved by encoding, although the models were trained on the normalized version of GEO dataset. This is an evidence of the cross platform viability of the E-MLP model.

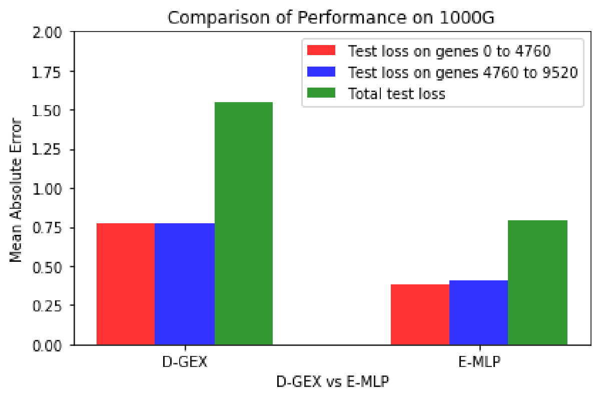

Figure 5. Testing models on 1000G.

3.5. Performance of Models on GEO Data

Similar to the E-MLP model for the cross platform normalized data, another MLP was constructed and trained for the entire GEO dataset with 978 landmark genes and 21290 target genes. An autoencoder with 3 hidden layers of 8000, 2000 and 8000 nodes respectively was used to encode the target expressions. The internal layer configuration was chosen on the basis of our observation that sudden decrease in the number of nodes per successive layers led to large reconstruction error. Deeper models were not viable due to limited main memory. The parameters for the predictive MLP network were directly adapted from the previous model. The hidden layer configuration of this MLP is identical to the 11850x11826 configuration that was used in E-MLP. Since the output nodes have been significantly reduced by the encoding phase, there was no need to look beyond 2-hidden layer feed-forward networks.

For the consolidated model that performs genome-wide microarray based profiling, the MAE on the test set was observed to be 1.8484. The principal contributor to this quantity was the reconstruction error of the autoencoder, which was close to 1.81. In order to compare the performance with the existing D-GEX model, the target gene set was divided into 5 disjoint sets of 4258 genes each. 5 MLP networks with an input layer consisting layer of 978 nodes, 3 hidden layers of 9000 nodes and an output layer of 4258 genes were constructed and trained on each of these target subsets.

The MAEs of all 5 models were computed and is shown in Table 4. Since the models are independent of each other, the overall error of the D-GEX model on the GEO dataset is computed as the sum of errors on all the models.

Table 4. MAE on 5 disjoint sets of target genes.

Set 1	Set 2	Set 3	Set 4	Set 5
0.3873	0.4032	0.3982	0.3964	0.3862

The overall error of the D-GEX model is thus 1.9713. This translates to an accuracy improvement of 6.6%. Although deeper encoders can produce better representations with minimum reconstruction error, hardware limitations hampered our attempts to increase the depth of the autoencoders. A deeper autoencoder can pull down the reconstruction error and thus improve the overall prediction accuracy.

4. Conclusions and Future Work

Gene expression profiling is an effective cost efficient method that characterises cellular state under various biological conditions. Direct measurement using laboratorial techniques like the Luminex Bead Technology is too expensive [15]. The D-GEX project successfully found an alternative using a deep learning solution. But the large number of target genes was a bottleneck in D-GEX, which was countered by building separate models for disjoint sets of target genes. Here, we have presented a statistical approach to profile thousands of genes at once using a combination of two neural network architectures. An auto encoder overcomes the redundancy by encoding the target expression into a concise representation. This enabled the construction of a single trained MLP to profile complete genome expressions. The MLP architecture of D-GEX is adopted for the second phase and modifications are made in the choice of the activation function, optimization technique, regularization strategy and hidden layer configuration. The modified encoded model resulted in a 16.64% improvement in performance on the normalized GEO data. The mean absolute error on the RNA-Seq based 1000G data is lesser than that of the test error on GEO dataset. This lends proof to the cross platform viability of the model. Additionally, a performance improvement of 6.6% was obtained on the original GEO data.

The percentage improvement is to be considered solely as a justification for the efficiency introduced by the encoding stage. In the current implementation, the depth of the autoencoder is severely restricted due to hardware limitations. Architectures with more number of hidden layers with a gradual decrease in the number of neurons in each successive layer of the encoder phase can further reduce the reconstruction error which would result in better encoding [20]. Training using multiple GPU cores could accelerate training and the additional memory would enable us to accommodate deeper architectures [2]. Using GPUs, deep networks could be efficiently trained using a greedy layer-wise approach. The change made to the distribution of the inputs of each layer as the parameters of the previous layers are updated is termed as Internal Covariance Shift (ICS). Batch Normalization (BN) addresses this problem [20]. This technique involves an additional normalization step in each layer, resulting in slower training in deep networks. Using BN for shallow networks is thus redundant. When the autoencoder is made deeper, using BN would be essential to prevent the vanishing gradient problem.

References

[1] Amilpur S. and Bhukya R., "EDeepSSP: Explainable Deep Neural Networks for Exact Splice Sites Prediction," *Journal of Bioinformatics and Computational Biology*, vol. 18, no. 04, 2020.

[2] Arel I., Rose D., and Karnowski T., "Deep Machine Learning-A New Frontier in Artificial Intelligence Research," *IEEE Computational Intelligence Magazine*, vol. 5, no. 4, pp. 13-18, 2010.

[3] Bansal M., Belcastro V., Ambesi-Impiombato A., and Di Bernardo D., "How to Infer Gene Networks from Expression Profiles," *Molecular Systems Biology*, vol. 3, no. 1, pp. 1-10, 2007.

[4] Baldi P., "Autoencoders, Unsupervised Learning, and Deep Architectures," *in Proceedings of ICML Workshop on Unsupervised and Transfer Learning*, Washington, pp. 37-50, 2012.

[5] Baldi P. and Sadowski P., "Understanding Dropout," *Advances in Neural Information Processing Systems*, vol. 26, pp. 2814-2822, 2013.

[6] Bengio Y., "Learning Deep Architectures for AI," *Foundations and Trends® in Machine Learning*, vol. 2, no. 1, pp. 1-127, 2009.

[7] Bhukya R. and Ashok A., "Gene Expression Prediction Using Deep Neural Networks," *The International Arab Journal of Information Technology*, vol. 17, no. 3, pp. 422-431, 2020.

[8] Bhukya R. and Sumit D., "Referential DNA Data Compression Using Hadoop Map Reduce Framework Using Deep Neural Networks," *The International Arab Journal of Information Technology*, vol. 17, no. 2, pp. 207-214, 2020.

[9] Caruana R., Lawrence S., and Giles L., "Overfitting in Neural Nets: Backpropagation, Conjugate Gradient, and Early Stopping," *in Proceedings of 13th International Conference on Neural Information Processing Systems*, Denver, pp. 402-408, 2000.

[10] Chen Y., Li Y., Narayan R., Subramanian A., and Xie X., "Gene Expression Inference with Deep Learning," *Bioinformatics*, vol. 32, no. 12, pp. 1832-1839, 2016.

[11] Clevert D., Unterthiner T., and Hochreiter S., "Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUS)," arXiv preprint arXiv:1511.07289, pp. 1-14, 2015.

[12] Dasari C. and Bhukya R., "InterSSPP: Investigating Patterns Through Interpretable Deep Neural Networks for Accurate Splice Signal Prediction," *Chemometrics and Intelligent Laboratory Systems*, vol. 206, 2020.

[13] De Sousa C., "An Overview on Weight Initialization Methods for Feedforward Neural Networks," *in Proceedings of the International Joint Conference on Neural Networks*, Vancouver, pp. 52-59, 2016.

[14] Edgar R., Domrachev M., and Lash A., "Gene Expression Omnibus: NCBI Gene Expression and Hybridization Array Data Repository," *Nucleic Acids Research*, vol. 30, no. 1, pp. 207-210, 2002.

[15] Géron A., *Hands-On Machine Learning with Scikit-Learn and TensorFlow Concepts, Tools, and Techniques to Build Intelligent Systems*, O'Reilly Media, 2019.

[16] Glorot X. and Bengio Y., "Understanding the Difficulty of Training Deep Feedforward Neural Networks," *in Proceedings of the 13th International Conference on Artificial Intelligence and Statistics*, Sardinia, pp. 249-256, 2010.

[17] Glorot X., Bordes A., and Bengio Y., "Deep Sparse Rectifier Neural Networks," *in Proceedings of the 14th International Conference on Artificial Intelligence and Statistics*, FL, pp. 315-323, 2011.

[18] Huang G., "Learning Capability and Storage Capacity of Two-Hidden-Layer Feedforward Networks," *IEEE Transactions on Neural Networks*, vol. 14, no. 2, pp. 274-281, 2003.

[19] Ioffe S. and Szegedy C., "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," *in Proceedings of the 32nd International Conference on Machine Learning*, Lille, pp. 448-456, 2015.

[20] Lamb J., Crawford E., Peck D., Modell J., Blat I., Wrobel M., Lerner J., Brunet J., Subramanian A., Ross K., Reich M., Hieronymus H., Wei G., Armstrong S., Haggarty S., Clemons P., Wei R., Carr S., Lander E., and Golub T., "The Connectivity Map: Using Gene-Expression Signatures to Connect Small Molecules, Genes, And Disease," *Science*, vol. 313, no. 5795, pp. 1929-1935, 2006.

[21] Le Q., Ranzato M., Monga R., Devin M., Chen K., Corrado G., Dean J., and Ng A., "Building High-Level Features Using Large Scale Unsupervised Learning," *in Proceedings of the 29th International Conference on Machine Learning*, Scotland, pp. 8595-8598, 2011.

[22] Lecun Y., Bengio Y., and Hinton G., "Deep Learning," *Nature*, vol. 521, pp. 436-444, 2015.

[23] Lin C., Jain S., Kim H., and Bar-Joseph Z., "Using Neural Networks for Reducing the Dimensions of Single-Cell RNA-Seq Data," *Nucleic Acids Research*, vol. 45, no. 17, pp. 1-11, 2017.

[24] NIH LINCS Program. *http://lincsproject.org/* www.lincsproject.org, Last Visited, 2013.

[25] Pierson E. and Yau C., "ZIFA: Dimensionality Reduction for Zero-Inflated Single-Cell Gene Expression Analysis," *Genome Biology*, vol. 16, no. 1, 2015.

[26] Rumelhart E., Hinton E., and Williams J., "Learning Representations by Back-Propagating Errors," *Nature*, vol. 323, no. 6088, pp. 533-536, 1986.

[27] Senior A., Heigold G., Ranzato M., and Yang K., "An Empirical Study of Learning Rates in Deep Neural Networks for Speech Recognition," *in Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing*, Vancouver, pp. 6724-6728, 2013.

[28] Vincent P. and Larochelle H., "Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion Pierre-Antoine Manzagol," *Journal of Machine Learning Research*, vol. 11, pp. 3371- 3408, 2010.

Raju Bhukya has received his B.Tech in Computer Science and Engineering from Nagarjuna University in the year 2003, M.Tech degree in Computer Science and Engineering from Andhra University in the year 2005 and P.hD in Computer Science and Engineering from National Institute of Technology (NIT) Warangal in the year 2014. He is currently working as an Assistant Professor in the Department of Computer Science and Engineering in National Institute of Technology, Warangal, Telangana, India. He is currently working in the areas of Bio-Informatics and Data Mining.