# Nonlinear Extensions of Reconstruction ICA

Apaar Sadhwani and Apoorv Gupta
*CS229 Project Report, Fall 2011*

*Abstract*— **In a recent paper [1] it was observed that unsupervised feature learning with overcomplete features could be achieved using linear autoencoders (named Reconstruction Independent Component Analysis). This algorithm has been shown to outperform other well-known algorithms by penalizing the lack of diversity (or orthogonality) amongst features. In our project, we wish to extend and improve this algorithm to include other non-linearities.**

**In this project we have considered three unsupervised learning algorithms: (a) Sparse Autoencoder (b) Reconstruction ICA (RICA), a linear autoencoder proposed in [1], and (c) Non-linear RICA, a proposed extension of RICA for capturing non-linearities in feature detection.**

**Our research indicates that exploring non-linear extensions of RICA holds good promise; preliminary results with *hyperbolic tangent* function on the MNIST dataset showed impressive accuracy (comparable with sparse autoencoders), robustness, and required a fraction of the computational effort.**

## I. Motivation

In many problems of interest - text, video, imaging, or speech - a fundamental problem is that of feature recognition. Hard coded (or, hand engineered) features have long been used, and continue to provide us with great accuracy. However, hard coding features requires much experience and the performance with unexplored domains remains uncertain. To overcome these obstacles, we use unsupervised learning, wherein we try to identify features that occur in a sparse fashion across our data. Mathematically, this amounts to trying to re-construct a high dimensional vector by passing it through a map with sparsity constraints. The sparsity constraints force retention of only *important details*, that is, the required *features*.

Given our high level strategy for feature extraction, it is important to note that while data contains only a sparse set of features in a particular instance, the number of such features can be potentially many. In line with this observation, the restriction that the number of features be less than the dimensionality of data seems unnatural; this occurs in many algorithms, and RICA provides a way of circumventing this by penalizing degeneracy, instead of imposing it as a hard constraint. However, with its current formulation, RICA captures only linear features. Through our study, we explore the benefits of extending it to non-linear features.

## II. Introduction

In this paper, we compare the performance of three unsupervised learning algorithms on accuracy and time. The first of these algorithms is Sparse Autoencoder, which is a feed-forward neural network. Here we consider the special case of a neural network with three layers - an input layer, a hidden layer, and an output layer. The hidden layer automatically enforces sparsity constraints since we have a small number of these. In addition, we enforce penalty on the average activations of the hidden layer, which is especially useful for the case of overcomplete features. This is further explained in Section III. The next algorithm we consider is RICA, as explained in IV. With its "soft" orthonormality constraint, overcompleteness is accommodated with no extra effort. Sparsity is however enforced as a separate penalty term in the objective function. The performance obtained depends considerably on the nature of the penalty term, and we consider both $L_1$ and $L_2$ penalties. This is further explained in Section IV, and one may refer to the original paper [1] for the underlying theory and details.

In the end, we propose a non-linear extension to RICA. The non-linear extension we test with is only for *encoding* the features, while the decoding function remains linear. This function is hyperbolic tangent, and our choice was based on its ubiquitous use as a basis function throughout the machine learning literature. We explain this procedure in greater detail in Section V.

It is important to note that all three algorithms can be viewed under the same umbrella - Sparse Autoencoders use sigmoid function for encoding/decoding, RICA uses linear function for encoding/decoding, and Non-linear RICA uses *tanh* for encoding and linear for decoding. We test the three algorithms for different number of hidden features, both above and below the overcompleteness threshold. The training and testing is performed on the MNIST dataset of handwritten digits, which is obtained from IX. The results and conclusions are discussed in Section VI.

## III. Sparse Autoencoders

### A. Introduction

Suppose we have unlabeled training examples set $\{x^{(1)}, x^{(2)}, \ldots\}$. An autoencoder neural network is an ***unsupervised learning algorithm*** that applies backpropagation, setting the target values to be equal to the inputs. i.e., it uses $y^{(i)} = x^{(i)}$. The autoencoder tries to learn a function $h_{W,b}(x) \approx x$.

### B. Model Formation and Framework

We use the following neural network defined by the following equations and framework. Let $n_l$ denote the number of layers in our network. Our neural network has parameters $(W,b) = (W(1), b(1), W(2), b(2))$, where we write ${W_{ij}}^{(l)}$ to denote the parameter (or weight) associated with the connection between unit $j$ in layer $l$, and unit $i$ in layer $l+1$. Also,

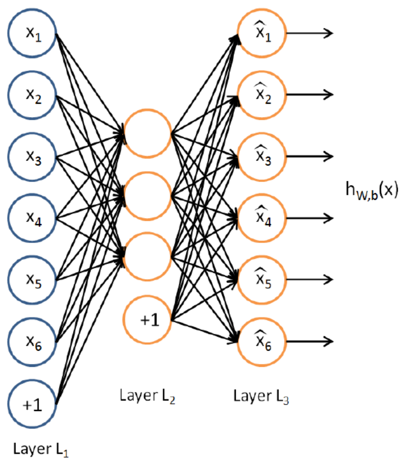

Fig. 1. Autoencoder

${b_i}^{(l)}$ is the bias associated with unit $i$ in layer $l+1$. We will write ${a_i}^{(l)}$ to denote the activation (or output value) of unit $i$ in layer $l$. Given a fixed setting of the parameters $\{W,b\}$, our neural network defines a hypothesis $h_{W,b}(x)$ that tries to reconstruct the original input. Specifically, the computation that this neural network represents is given by:

$$a_1{}^{(2)} = f(W_{11}{}^{(1)}x_1 + W_{12}{}^{(1)}x_2 + W_{13}{}^{(1)}x_3 + b_1{}^{(1)})$$
$$a_2{}^{(2)} = f(W_{21}{}^{(1)}x_1 + W_{22}{}^{(1)}x_2 + W_{23}{}^{(1)}x_3 + b_2{}^{(1)})$$
$$a_3{}^{(2)} = f(W_{31}{}^{(1)}x_1 + W_{32}{}^{(1)}x_2 + W_{33}{}^{(1)}x_3 + b_3{}^{(1)})$$

$$\{h_{W,b}(x)\}_i = a_i{}^{(3)} = f(W_{i1}{}^{(2)}a_1{}^{(2)} + W_{i2}{}^{(2)}a_2{}^{(2)} + W_{i3}{}^{(2)}a_3{}^{(2)} + b_i{}^{(2)})$$

In the sequel, we also let $z_i{}^{(2)} = \sum_{j=1}^{n} W_{ij}{}^{(1)}x_j + b_i^{(1)}$ denote the total weighted sum of inputs to unit $i$ in layer $l$, so that $a_i^{(l)} = f(z_i^{(l)})$. Also:

$$z^{(l+1)} = W^{(l)}a^{(l)} + b^{(l)} \tag{1}$$
$$a^{(l+1)} = f(z^{(l+1)}) \tag{2}$$

### C. Backpropagation algorithm

Given a training set of $m$ examples, we then define the overall cost function to be:

$$J(W,b) = \Big[\frac{1}{m}\sum_{i=1}^{m} J(W,b;x^{(i)})\Big] + \frac{\lambda}{2}\sum_{l=1}^{n_l-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(W_{ji}^{(l)})^2$$
$$= \Big[\frac{1}{m}\sum_{i=1}^{m}\Big(\frac{1}{2}\Big|\Big|h_{W,b}(x^{(i)}) - x^{(i)}\Big|\Big|^2\Big)\Big] + \frac{\lambda}{2}\sum_{l=1}^{n_l-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(W_{ji}^{(l)})^2 \tag{3}$$

The second term is a regularization term (also called a weight decay term) that tends to decrease the magnitude of the weights, and helps prevent overfitting.The ***weight decay parameter*** $\lambda$ controls the relative importance of the two terms. The goal is to minimize $J(W,b)$ as a function of $W$ and $b$ using the following back propagation algorithm:

1. Perform a feedforward pass, computing the activations $({a_i}^{(l)}$ for layers $L_2$, $L_3$, up to the output layer $L_{n_l}$, using Equations (1-2).
2. For the output layer (layer $n_l$), set
$$\delta^{(n_l)} = -(x - a^{(n_l)}) \bullet f'(z^{(n)}) \tag{4}$$
where $\bullet$ denotes the element-wise product operator.
3. For $l = n_l - 1, n_l - 2, n_l - 3, \ldots, 2$,set
$$\delta^{(l)} = ((W^l)^T\delta^{(l+1)}) \bullet f'(z^{(l)}) \tag{5}$$
4. Compute the desired partial derivatives:
$$\nabla_{W^{(l)}}J(W,b;x,y) = \delta^{(l+1)}(a^{(l)})^T$$
$$\nabla_{b^{(l)}}J(W,b;x,y) = \delta^{(l+1)} \tag{6}$$
5. Calculate $J_{sparse}(W,b)$ using:
$$J_{sparse}(W,b) = J(W,b) + \beta\sum_{j=1}^{s_2} KL(\rho\|\hat{\rho}_j) \tag{7}$$
where $\rho$, $\hat{\rho}_j$ (sparsity constraint) and $KL(\rho\|\hat{\rho}_j)$ is derived in [3]. Here we update the parameters as follows:
$$W^{(l)} := W^{(l)} - \alpha\Big[\Big(\frac{1}{m}\Delta W^{(l)}\Big) + \lambda W^{(l)}\Big] \tag{8}$$
$$b^{(l)} := b^{(l)} - \alpha\Big[\frac{1}{m}\Delta b^{(l)}\Big] \tag{9}$$
where $\alpha$ is the ***learning rate***.
6. Train neural network using unconstrained optimizer L-BFGS to find $W^* = argmin_W J(W,b)$.
7. Use $\sigma(WX)$, $X$ ={training data, test data} to find the training & test features.
8. Calculate the test and training error using softmax regression.

So, we observe the sparse auto-encoders have the following unconstrained optimization problem:

$$min_W\Big[\frac{\lambda}{m}\sum_{i=1}^{m}\|\sigma(W^T\sigma(Wx^{(i)}+b)) - x^{(i)}\|_2^2 + \beta\sum_{j=1}^{s_2}KL(\rho\|\hat{\rho}_j)\tag{10}$$

where $\sigma(x) = \frac{1}{1+\exp(-x)}$. As, we will explain later in section V, this forms our basis for motivation for modifying the optimization function of RICA.

### D. Results

By forming the image formed by the pixel intensity values, we can understand what feature a hidden unit is looking for. Each square in the figure below shows the input image $x$ that maximally activates one of hidden units. Observation is that different hidden units have learned to detect edges at different positions and orientations in the image. We run the proposed algorithm of Sparse auto-encoder on our dataset and get the following features.

This demostrates that the sparse auto-encoder algorithm learns a set of edge detectors (which are like pen strokes for MNIST dataset). In the figure below, we show that as we increase the number of hidden units in the sparse auto-encoder, the test accuracy starts increasing. At the same time, the train accuracy reaches 100% which might be an indication of overfitting.

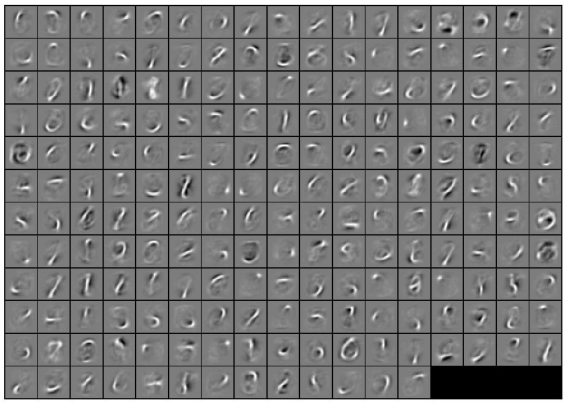

Fig. 2. Output Visualization (200 hidden units)

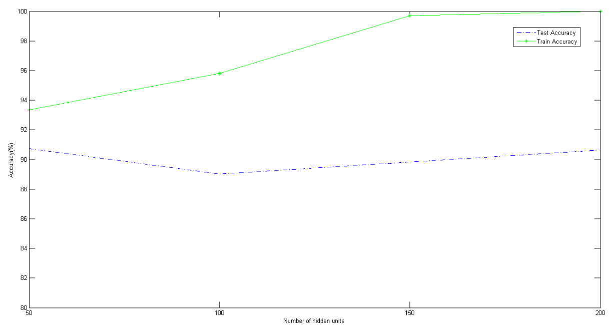

Fig. 3. Test and Training accuracy for sparse auto-encoder with different hidden units

## IV. Reconstruction ICA

### A. Motivation

Sparsity has been shown to work well for learning feature representations that are robust for object recognition. As discussed in section III, sparse autocoders is one of the many ways. Other methods include ICA nad ISA (Independent Subspace Analysis) refer paper. But the standard ICA has two major drawbacks:

- *Overcomplete Features*: The ICA is not easy to train in presence of overcomplete features (when the number of features $\gg$ dimensionality of input data). Autoencoders on the other hand work well in case of overcomplete features.
- *Whitening*: Standard ICA is sensitive to whitening (a preprocessing step that decorrelated the input data, and can not always be computed exactly for high dimensional data.

Both constraints in the standard ICA arise due to the hard orthonormality constraints requiring the features to be orthogonal i.e. $WW^T = \mathbf{I}$, which is used to prevent degenerate solution in the feature matrix $W$. However, this condition of orthonormality cannot be met if we have overcompleteness.

In [1], the authors have proposed Reconstruction ICA(RICA), which is a modification to the standard ICA. In RICA, the orthonormality constraint is replaced with a linear reconstruction penalty(jut like the one we defined above for sparse auto-encoders in section III). This also makes it possible to use unconstrained solver L-BFGS for the optimization of the cost function. The reconstruction penalty can also be applied across all receptive fields and hence prevents degenerate features. Also, as RICA is based on foundations similar to auto-encoders, it is less sensitive to whitening.

### B. Model Formation

Given the ***unlabeled data*** $\{x^{(i)}\}_{i=1}^{m}, x^{(i)} \in R^n$, regular ICA is defined as the following optimization problem:

$$minimize_W \sum_{i=1}^{m}\sum_{j=1}^{k} g(W_j x^{(i)}), \text{subject to } WW^T = \mathbf{I} \tag{11}$$

RICA on the other hand produces the following unconstrained optimization problem:

$$minimize_W \left[\frac{\lambda}{m}\sum_{i=1}^{m} \|W^T W x^{(i)} - x^{(i)}\|_2^2 + \sum_{i=1}^{m}\sum_{j=1}^{k} g(W_j x^{(i)})\right] \tag{12}$$

where $g$ is a nonlinear convex function, e.g., smooth $L_1$ penalty: $g(.) = \log(\cosh(.))$ as it mimics the $L_1$ norm $\|.\|_1$. We also construct and analyze $L_2$ penalty in later section. $W$ is the weight matrix $W \in R^{k\times n}$ where $k$ is the number of components (features) and $W_j$ is one row (feature) in W. Observe that RICA is tied-weighted, i.e. $W$ is the encoding weights matrix and $W^T$ is the decoding weights matrix.(i.e. the encoding step is $Wx^{(i)}$ and the decoding step is $W^T(Wx^{(i)})$.The aim is to minimize the following cost function:

$$J(W) = \left[\frac{\lambda}{m}\sum_{i=1}^{m} \|W^T f(Wx^{(i)}) - x^{(i)}\|_2^2 + \sum_{i=1}^{m}\sum_{j=1}^{k} g(f(W_j x^{(i)}))\right]$$

$$J(W) = \frac{\lambda}{m} H(W) + G(W) \tag{13}$$

where:

$$H(W) = (W^T WX - X)^T (W^T WX - X) \tag{14}$$

$$G(W) = g(W_j X) \tag{15}$$

The activation function considered in case of RICA is proposed to be a linear: $\{Wx^{(i)}\}$. In the next section, we propose a non-linear extension of RICA which consider a range of non-linear functions $f(Wx^{(i)})$ instead of $(Wx^{(i)})$ and try to compare its performance with our implementation of RICA in terms of time and accuracy.

## V. Non Linear Extension of RICA

### A. Model Formation

In this section, we propose our formulation of extending RICA to incorporate non-linear functions. Hereafter, we refer to our algorithm as NLE-RICA. The main motivation behind NLE-RICA is to form an intermediate function that

TABLE I
THE FOLLOWING ALGORITHM IS USED FOR LINEAR RICA:

```
1.Initialize $\lambda = \{0.1, 0.9\}$
2.Using back propagation algorithm, calculate $\{a^{(2)}, a^{(3)}, \delta^{(1)}, \delta^{(1)}, \delta^{(1)}\}$
3.Calculate $\nabla_W H(W) = 2(WW^TWXX^T + WXX^TW^TW - 2WXX^T)$
4.Calculate $\nabla_W G(W) = tanh(WX)X^T$
5.Calculate $\nabla_W J(W) = \frac{\lambda}{m}(\nabla_W H(W)) + \nabla_W G(W)\}$ and $J(W)$
6.Use the unconstrained optimizer L-BFGS to find $W^* = argmin_W J(W)$
7.Use $f(WX)$, $X$ ={training data, test data} to find the training & test features.
8.Calculate the test and training error using softmax regression.
```

mimics linear function in case of RICA and the non-linear function(sigmoid) in case of sparse auto-encoders. For the same purpose, we consider the following functional form for $f(Wx^{(i)})$:

- $f(x) = x$ (Linear RICA)
- $f(x) = 0.4x + 0.6\tanh(x)$ (Mixed NLE-RICA I)
- $f(x) = 0.8x + 0.2\tanh(x)$ (Mixed NLE-RICA II)
- $f(x) = \tanh(x)$ (Pure NLE-RICA)

The reason for including $\tanh(x)$ in the NLE-RICA is its property of mimicking the shape of sigmoid functions. Thus, while making a transition from Linear RICA model

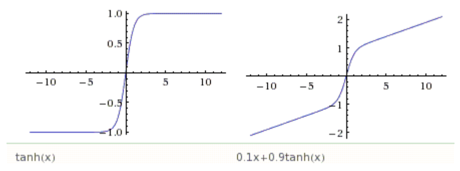

Fig. 4. Graph for $\tanh(x)$ and 0.1x + 0.9tanh($x$)

to Pure NLE-RICA model we incorporate some linearity as well. These models are referred to as mixed NLE-RICA I and Mixed NLE-RICA II models. Hence, NLE-RICA is defined as the following optimization problem:

$$min_W \Big[\frac{\lambda}{m}\sum_{i=1}^{m} \|W^T f(Wx^{(i)}) - x^{(i)}\|_2^2 + \sum_{i=1}^{m}\sum_{j=1}^{k} g(f(W_j x^{(i)}))\Big] \tag{16}$$

We observe that in NLE-RICA, the encoding is done through $f(Wx^{(i)})$, but the decoding is done through $W^T f(Wx^{(i)})$. This is different from sparse auto-encoder where you had sigmoid function being used for encoding (from input layer to hidden layer) and decoding (from hidden layer to output layer). This also differs from RICA in the sense that the encoding is carried out by $f(Wx^{(i)})$ instead of $(Wx^{(i)})$. Also, observe the change in the $L_1$ penalty term in the optimization problem. Let's define the conventions before discussing the algorithm.

$$J(W) = \Big[\frac{\lambda}{m}\sum_{i=1}^{m} \|W^T f(Wx^{(i)}) - x^{(i)}\|_2^2 + \sum_{i=1}^{m}\sum_{j=1}^{k} g(f(W_j x^{(i)}))\Big]$$

$$J(W) = \frac{\lambda}{m}H(W) + G(W) \tag{17}$$

where:

$$H(W) = (W^T f(WX) - X)^T (W^T f(WX) - X) \tag{18}$$

$$G(W) = g(f(W_j X)) \tag{19}$$

and $g(.) = \log(\cosh(.))$ and $f(.) = \gamma(.) + (1-\gamma)(.)$ where $\gamma \in \{0, 0.4, 0.8, 1\}$. Hence:

$$\nabla_W J(W) = \frac{\lambda}{m}\nabla_W H(W) + \nabla_W G(W) \tag{20}$$

where we calculate $\nabla_W H(W)$ using the back-propagation algorithm defined in section III and $\nabla_W G(W) = \tanh(f(WX))(\gamma + (1-\gamma)(1-(\tanh(WX))^2))X^T$

$$\nabla_W J(W) = \frac{\lambda}{m}(\nabla_W H(W) + \nabla_{W^T} H(W)) + \nabla_W G(W) \tag{21}$$

$$== \frac{\lambda}{m}(\delta^{(2)} a^{(1)^T} + a^{(2)} \delta^{(3)^T}) + \nabla_W G(W) \tag{22}$$

TABLE II
THE FOLLOWING ALGORITHM IS USED FOR NLE-RICA:

```
1.Initialize $\lambda = \{0.1, 0.9\}$ and $\gamma \in \{0, 0.4, 0.8\}$
2.Using back propagation algorithm, calculate $\{a^{(2)}, a^{(3)}, \delta^{(1)}, \delta^{(2)}, \delta^{(3)}\}$
3.Calculate $\{\nabla_W H(W), \nabla_{W^T} H(W), \nabla_W G(W)\}$
4.Calculate $\nabla_W J(W) = \frac{\lambda}{m}(\nabla_W H(W) + \nabla_{W^T} H(W)) + \nabla_W G(W)\}$ and $J(W)$
5.Use the unconstrained optimizer L-BFGS to find $W^* = argmin_W J(W)$
6.Use $f(WX)$, $X$ ={training data, test data} to find the training & test features.
7.Calculate the test and training error using softmax regression.
```

## VI. RESULTS

### A. Effect of $\lambda$

$\lambda$ seems to play an important role in the calculate of cost for both linear RICA and NLE-RICA. Very small values of $\lambda$ (like those in auto-encoders) provide inadequate weight to the error term of $H(W)$. For our study, we chose $\lambda = \{0.1, 0.9\}$. The results for the two values are summarized in the graph below.(Figure 5)

It is clear from the graph that we cannot treat a value of $\lambda \in [0.1, 1]$ to be superior than the other. But a small value of $\lambda$ can clearly be ignored.

### B. Effect of number of hidden units

Choosing an optimal number of hidden units ($k$) is specifically important for the RICA algorithms to work well. A low number of hidden units make poor prediction giving high test error. Whereas, high number of units leads to may be an indication of over-fitting (making 0% training error). We run our algorithms for $k = \{200, 400, 700, 800, 900\}$ and get the following results(Figure 6):

### C. Change to $L_2$ norm for reconstruction penalty function

Instead of $g(.) = \log(\cosh(.))$ (the usual $L_1$ norm), we now take $g(.) = \|.\|^2$ ($L_2$ norm) as the reconstruction penalty. We apply this reconstruction on the four RICA functions and obtain the following errors($k$=900, $\lambda$ =0.9) (Table III):

As we can see, the $L_2$ penalty also provides good test and training accuracy. But, the tradeoff present is slower running time.

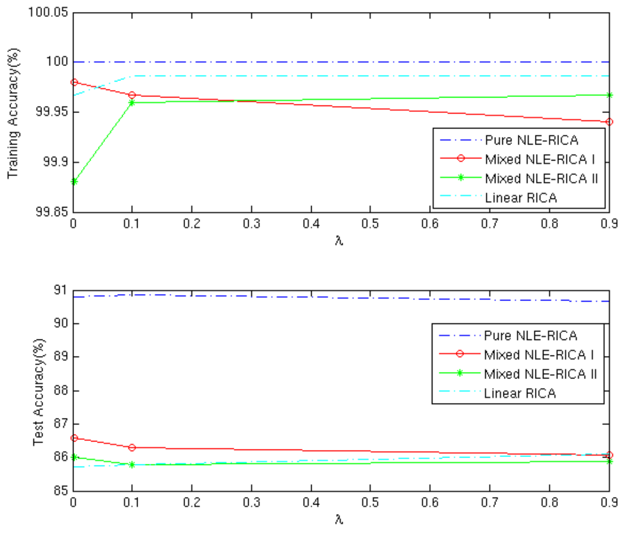

Fig. 5. Training(Top) and Test Accuracy(Bottom) for different values of $\lambda$, k=900, 500 iterations

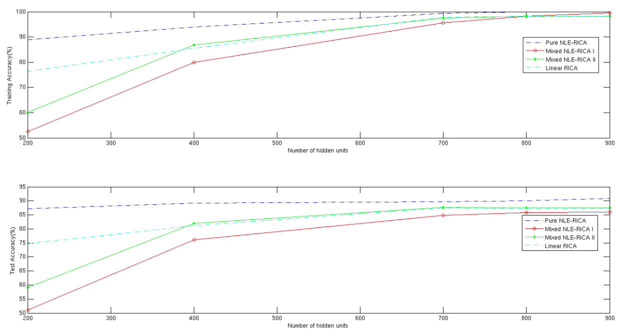

Fig. 6. Training(Top) and Test Accuracy(Bottom) for different number of hidden units ($\lambda$ =0.9, 500 iterations)

TABLE III

TEST AND TRAINING ERROR FOR $L_2$ RECONSTRUCTION PENALTY

| Model | Training Accuracy | Test Accuracy |
|---|---|---|
| Linear RICA | 94.79% | 83.13% |
| Mixel NLE-RICA I | 99.92% | 83.3% |
| Mixed NLE-RICA II | 99.73% | 84.21% |
| Pure NLE-RICA | 100% | 90.95% |

### D. *Effect of $\gamma$*

We check the test accuracy by varying the parameter $\gamma$ of our non-linear model $f(.) = \gamma(.) + (1-\gamma)\tanh(.)$. Here we use $\nabla_W G(W) = 2 * f(WX). * (\gamma + (1-\gamma)(1-(\tanh(WX))^2))X^T$ in the NLE-RICA algorithm.

As is evident from Figure 7, the test accuracy is decreasing as we move from $\gamma = 0$ (Pure NLE-RICA) to $\gamma = 1$ (Linear RICA). Thus, NLE-RICA can be more accurate than Linear RICA.

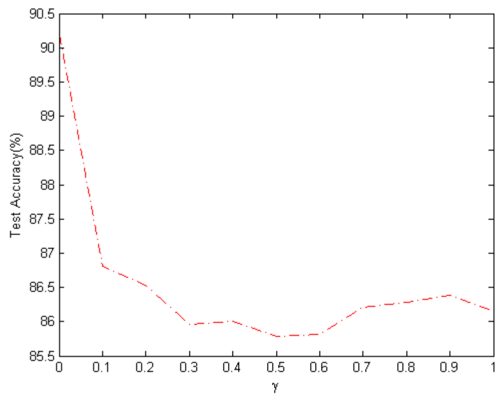

Fig. 7. Test Accuracy for different $\gamma$ ($\lambda$ =0.9,$k$=900,50 iterations)

## VII. IMPLEMENTATION

In the project, we run our algorithms on 30,000 training & 10,000 test samples from the MNIST dataset. The simulations were effected in the Stanford Corn computing environment with 8-core 2.7 GHz AMD Opteron (2384) processor, 32GB RAM, 10GB swap & 75 GB temp Disk, running Ubuntu GNU/Linux Operating System [4].

## VIII. CONCLUSIONS AND FUTURE WORK

The class of RICA algorithms, both linear and non-linear, provide a worthy alternative to Sparse Autoencoders, especially for scenarios where it becomes important to allow for overcomplete features. They compete well on accuracy of classification and consume much less computational effort. They are easy to code, seem quite robust in obtaining a consistent minima, and give very similar in-sampe and out-of-sample performance.

The non-linear extension seems a candidate worthy of exploring further - it beats the linear counterpart consistently. Future research should be directed towards experimenting with other functions, and also with deriving a more sound theoretical basis for their better performance.

Amongst the drawbacks, we found that the values of the hyperparameters ($\lambda$ and the number of hidden units) in RICA play a significant role in obtaining a reasonable solution. Investing time in developing a rule of thumb to guide a first time user would be very useful to more widespread use of this method.

Time permitting, we would have liked to test our results on other standard databases for our non-linear extension. Also, it was hard to compare the performance of Sparse Autoencoders with that of RICA (and its variants). As yet another step, we are quite curious and look forward to developing a theoretical framework to provide rules of thumb to guide the trade-offs between number of hidden units, linear or non-linear features, lambda, number of iterations, and lastly, the number of layers of encoding to obtain a sparse representation.

## IX. Acknowledgements

We would like to acknowledge and thank Mr. Quoc Le, PhD student at Stanford CS Department, for helping us with the project and providing us the MNIST dataset.

## References

[1] Q.V. Le, A. Karpenko, J. Ngiam, A.Y. Ng. (2011)"ICA with Reconstuction Cost for Efficient Overcomplete Feature Learning," NIPS.
[2] Q.V. Le, J. Ngiam, A. Coates, A. Lahiri, B. Prochnow, A.Y. Ng.(2011) "On optimization methods for deep learning", ICML
[3] Ng, A.,"CS294A Lecture Notes - Stanford University" .
[4] Unix Computing Environments, Stanford University.