Improving Neural Network Learning by Modifying the Output Target Function

Mijail Gomez, *Student Member, IEEE*,

***Abstract*—Neural Networks have been known to be difficult to train. In this paper an alternative method of training neural network is shown. The method consist of modifying the target output function such that noise, present in the input, is used to regularize the learning parameters. We apply this modification to both supervised and representation learning problems. For the supervised learning problem, we compared with the standard method of training neural networks and SVM. For representation learning we compare with PCA, RBMs, and Denoising Auto-Encoder. Simulated data was used to conduct the experiments.**

***Index Terms*—Neural Networks, Regularization, Noise Regularization, Representation Learning**

I. Introduction

NEURAL Networks have an old history, dating back to the 1940's. In recent years there has been a spike in neural network research, specifically in Deep Neural Networks. Motivated by this surge of excitement I chose to learn more about neural networks.

An important property of neural networks, that lays down the ground work, follows from the Cybenko-Hornik-Funahashi Theorem: [3] [4] [6] [5]

Theorem 1 (Universal Approximation) Let f be continuous, non-constant, bounded and monotone increasing. For $\epsilon > 0$, there is an integer N and c_i, θ_i, w_{ij} such that

$$\bar{f}(x_1, \ldots, x_n) = \sum_{i=1}^{N} c_i \phi \left(\sum_{j=1}^{n} w_{ij} x_j - \theta_i \right)$$

satisfies, $\max_{\mathbf{x} \in \chi} \left| f(\mathbf{x}) - \bar{f}(\mathbf{x}) \right| < \varepsilon$.

The theorem states that a multi-layer neural network can approximate any continuous function f to any accuracy requiring any number of hidden nodes. This motivates the use of multi-layer neural networks when trying to learn complex models.

A drawback in implementing multi-layer neural networks is that it can converge to a local minimum. And it also might take a while to converge compared to other learning algorithms. In this paper we explore a modification to neural network and we applied it to a problem in detecting meteors.

M. Gomez is with the Department of Electrical Engineering, University of Illinois Urbana-Champaign. gomez19@illinois.edu

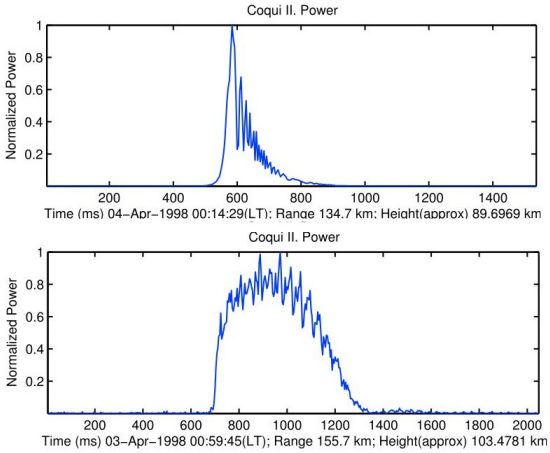

Fig. 1: Top plot shows an underdense meteor echo, bottom plot shows an overdense meteor echo

In a radar detection setting we wish to classify between two different types of meteors echoes. The time series in figure 1 shows the magnitude of two meteor echoes, an underdense echo and an overdense echoe. From this problem setting a method to train neural networks and to obtain representations was motivated. We know a-priori that the two meteor echoes will on average have distinctive features (i.e., larger bandwith, decay rate). From this information we can create a model for each class. This model can then be used as a label to train a neural network.

There have been various modifications to supervised and representation learning algorithms. The structure of the paper is as follows. In a supervised learning setting, we first give an overview of the theory behind noise regularization. Then we show how by modifying the target function we can achieve noise regularization and a well behaved error function. The for learning representations we go over some well-known algorithms; we go over auto-encoder, denoising auto-encoder, and RBMs. And we show that our modification can also be used to find representations in data. We conclude with a series of experiments to show how our implementation improves both in supervised learning and in finding representations.

II. Supervised Learning

A. Neural Networks and Regularization

The basic neural network model consist of a series of functional transformations. Let us consider a two layer neural network. The zeroth layer (input layer) takes the observations as the input, the first layer (hidden layer) consist a weighted linear transformation of the inputs, and the

third layer (output layer) is a weighted linear transformation of the hidden nodes. More generally, the set of equations for multi-layer neural network is as follows. Layer l computes an output vector $\mathbf{z}^l$ using the output $\mathbf{z}^{l-1}$ of the previous layer, starting with the n-th observation $\mathbf{z}^0 = \mathbf{x^n}$,

$$\mathbf{z}^l = h(\mathbf{W}^l \mathbf{z}^{l-1}),\ l = (1, \ldots, L)$$

where the bias has been included with the input and the hidden nodes, $\mathbf{W}^l$ is the set of weight coefficients we wish to learn. The function $h(\cdot)$ depends on the application, typical functions are *sigmoid*, tanh, and the identity.[7]

In order to learn the weight parameter we assume we have a training set consisting of of the set of tuples $\{(\mathbf{x}^1, \mathbf{t}^1), \ldots, (\mathbf{x}^N, \mathbf{t}^N)\}$, and we wish to minimize the error function

$$E^n = \|\mathbf{t}^n - f(\boldsymbol{W}, \mathbf{x}^n)\|_2^2.$$

The backpropagation algorithm is usually used to learn the weight parameters. The algorithm consist of two phases forward propagation and backward propagation. In the backward propagation the weights are updated using gradient descent, assuming the activation functions are differentiable. Newton and Quasi-Newton methods are used to speed up the convergence rate (BFGS, conjugate gradient, etc.). [**?**]

Neural network models are well suited for large data sets, since their capacity can easily be increased by adding more layers or more units in each layer. However, big networks with millions or billions of parameters can easily over-fit even the largest of data sets. A well known method reduce over-fitting is regularization.

Regularization is the process of incorporating additional information to prevent over-fitting. This extra information is usually of the form of a penalty for complexity, such as restrictions for smoothness or bounds on the vector space norm. To reduce over-fitting, adding an l_2 penalty on the network weights is one simple but effective approach. That is,

$$\begin{aligned}\tilde{E}^n &= \|\mathbf{t}^n - f(\boldsymbol{W}, \mathbf{x}^n)\|_2^2 + \frac{\lambda}{2}\|\boldsymbol{W}\|_2^2 \\ &= E^n + \frac{\lambda}{2}\|\mathbf{W}\|_2^2\end{aligned}$$

where λ is called the *regularization parameter*. The first term enforces closeness to the data while the second smoothness. [8]

Early stopping is an alternative to regularization, which basically controls the effective complexity of a network. A more recent method is dropout, during forward propagation half of the activation functions in each layer are deleted the error is backpropagated only through the remaining activation functions. [16]

B. Training Neural Network With Noise

It was shown, in [1], the addition of noise in a neural network in some cases leads to improvements in generalization. A brief derivation will be shown. Let the noise on the input vector be described by the random vector $\boldsymbol{\eta}$. The error function when training with noise can then be written as follows

$$\tilde{E} = \int\int\int \sum_k (y_k(\mathbf{x} + \boldsymbol{\eta}) - t_k)^2 p(t_k|\mathbf{x}) p(\mathbf{x}) \tilde{p}(\boldsymbol{\eta}) d\mathbf{x} dt_k d\boldsymbol{\eta},$$

where $\tilde{p}(\boldsymbol{\eta})$ denotes the distribution function of the noise. Assuming the noise amplitude is small, we can expand the network output as a Taylor series in powers of $\boldsymbol{\eta}$ to give

$$y_k(\mathbf{x}+\boldsymbol{\eta}) = y_k(\mathbf{x}) + \sum_i \eta_i \frac{\partial y_k}{\partial x_i}\Big|_{\boldsymbol{\eta}=0} + \sum_i \sum_j \eta_i \eta_j \frac{\partial^2 y_k}{\partial x_i \partial x_j}\Big|_{\boldsymbol{\eta}=0} + \mathcal{O}(\boldsymbol{\eta}^3).$$

Assuming the noise is Additive White Gaussian Noise (AWGN), we have

$$\int \eta_i \tilde{p}(\boldsymbol{\eta}) d\boldsymbol{\eta} = 0$$
$$\int \eta_i \eta_j \tilde{p}(\boldsymbol{\eta}) d\boldsymbol{\eta} = \sigma^2 \delta_{ij}$$

where σ^2 is proportional to the variance of the noise. Thus integrating over the noise we get,

$$\tilde{E} = E + \sigma^2 E^R$$

where E is the standard sum-of-squares, and the extra term E^R is given by

$$E^R = \int\int \sum_k \sum_i \left(\left(\frac{\partial y_k}{\partial x_i} \right)^2 + \frac{1}{2}(y_k(\mathbf{x}) - t_k) \frac{\partial^2 y_k}{\partial x_i^2} \right) p(t_k|\mathbf{x}) p(\mathbf{x}) d\mathbf{x} dt_k.$$

provided the noise amplitude is small, so that the higher order terms in the Taylor series expansion is valid.

The computation of E^R involves second order derivatives, which is computationally expensive. The complexity can be reduced from the realization that the optimum solution will be of the form, $y_k^{min} = \langle t_k|\mathbf{x}\rangle + \mathcal{O}(\sigma^2)$. The regularizing term then simplifies,

$$E^R \approx \int \sum_k \|\nabla y_k(\mathbf{x})\|_2^2 \, p(\mathbf{x}) d\mathbf{x}$$

where the t_k variables have been integrated out. The second equation follows from fact that the second term vanishes, at the minimum of the total error, with order proportional to σ^2,

Now, consider a neural network constructed with only linear activation functions. The regularizing term, for small noise, will be proportional to the l_2 norm of the weights. That is,

$$\begin{aligned}\tilde{E} &= E + \sigma^2 E^R \\ &= E + \sigma^2 \|\mathbf{W}\|_2^2\end{aligned}$$

This has the form of a regularization term added to the usual sum-of-squares error, with the coefficient of the regularizer determined by the noise variance σ^2.

C. Training Neural Network With Modified Target Function

It was shown in the previous section that adding noise to the input of a neural network could improve performance by regularizing; with the restriction that the noise be small in amplitude. The problem with this concept is that most of the signals we deal with in the real world inherently contain noise. Therefore, the constraint, on small noise, is not met and the performance will not improve.

We propose the following idea: *Use the noiseless version of the input signal as the target function of the neural network*. Intuitively the performance should improve because we are feeding the neural netowrk with more information. Also, the error function is no longer an average of 1-dimensional errors, but D-dimensional errors for each observation (D is the dimension of the input data). And the error function is minimizing the difference between the signal approximation produced by the neural network and the noiseless version of the input signal.

If the data has some structure. For example, the meteor echo is a time series. And we know before hand how the two types of meteor echoes behaves. We are able to estimate the noiseless version of the meteor echo. One way to estimate the noiseless signal can be by curve fitting or taking the average of the meteor echo observation for each of our classes. This estimate of our noiseless meteor echo, can be used as the Target label to our neural network. This, intuitively should provide more information to the neural network to learn. We shall show this by analyzing the error function.

Consider a two layer neural network, using the mean square error as a loss function, we are trying to minimize the following cost function:

$$E^n = \left\| t^n - h(\mathbf{W}^2 h(\mathbf{W}^1 \mathbf{x}^n)) \right\|_2^2$$

In binary classifcation the target consist of one node, $t \in \{-1, 1\}$. If learning K different classes then $\mathbf{t} \in \{-1, 1\}^K$; classification is made by picking the output node with the highest magnitude. Now, let us change the binary target function to a noiseless version of the input for each class. We will restrict the activation functions to be linear i.e., identity. We will also assume that the noise is additive and uncorrelated to the input data. Let our observed instance be $\tilde{\mathbf{x}}^n$ and our estimate of the noiseless observation is $\mathbf{x}^n$. Then, using the corresponding estimate of the signal as target, our error function becomes

$$\begin{aligned} E^n &= \left\| \mathbf{t}^n - \mathbf{W}^2\mathbf{W}^1\tilde{\mathbf{x}}^n \right\|_2^2 \\ &= \left\| \mathbf{x}^n - \mathbf{W}^2\mathbf{W}^1(\mathbf{x}^n + \boldsymbol{\eta}^n) \right\|_2^2 \\ &= \left\| \mathbf{x}^n - \mathbf{W}^2\mathbf{W}^1\mathbf{x}^n - \mathbf{W}^2\mathbf{W}^1\boldsymbol{\eta}^n) \right\|_2^2 \\ &= \|\mathbf{x}^n - \mathbf{W}\mathbf{x}^n\|_2^2 + \|\mathbf{W}\boldsymbol{\eta}^n)\|_2^2 - 2(\mathbf{x}^n - \mathbf{W}\mathbf{x}^n)^T(\mathbf{W}\boldsymbol{\eta}^n) \end{aligned}$$

where $\mathbb{W} = \mathbf{W}^2\mathbf{W}^1$. Assuming that the input and the noise are uncorrelated, then the third term will average out. Then we have the following,

$$E = \frac{1}{N}\sum_{n=1}^{N} \|\mathbf{x}^n - \boldsymbol{W}\mathbf{x}^n\|_2^2 + \sigma^2 \|\boldsymbol{W}\|_2^2$$

where σ^2 is proportional to the variance of the noise.

Notice that the noise acts like a regularizing parameter. A similar expression was obtain in the previous section, but no extra noise was added to obtain this regularizing term. Another key difference is in the error term. It is minimizing the difference between a clean noiseless signal and the corresponding signal representation produced by the neural network.

III. Representation Learning

An official definition for representation learning has not been set yet. Many authors have their own perspective of what it really means. Here we use two definitions, that is applicable to our problem:

1. "A good representation is one that disentangles the underlying factors of variation."[11]
2. "A good representation is one that can be obtained robustly from a corrupted input and that will be useful for recovering the corresponding clean input."[13]

A. Auto-Encoder and Denoising Auto-Encoder

Let us begin by defining a feature extracting function in a specific parametrized closed form. This function, that we will denote f_θ, is called the *encoder* and allows the straight forward and efficient computation of a features vector $h = f_\theta(x)$ from an input x. Another closed form parametrized function g_θ, called the *decoder*, maps from feature space back into input space, producing a *reconstruction* $r = g_\theta(h)$. Auto-encoders are parametrized through their encoder and decoder and are trained using a different training principle. The set of parameters θ of the encoder and decoder are learned simultaneously on the attempting to achieve the lowest possible *reconstruction error* $E(x, r)$ – a measure of the discrepancy between x and its reconstruction – on average over a training set,[11][9]

$$\begin{aligned} E^n(\tilde{\mathbf{x}}, \mathbf{r}) &= \|\tilde{\mathbf{x}}^n - \mathbf{r}\|_2^2 \\ &= \|\tilde{\mathbf{x}}^n - \mathbf{g}_\theta(\mathbf{f}_\theta(\tilde{\mathbf{x}}^n))\|_2^2. \end{aligned}$$

Minimization of the reconstruction error is usually carried by gradient descent or its variants. The minimization of the reconstruction error aims to capture the structure of the data generating distribution. The trivial solution, learning the identity function, is a possibility when the number of hidden nodes is larger than the dimensionality of the input data vector. *regularized auto-encoders*, a particular form of regularization consists in constraining the code to have a low dimension, and this what the classical auto-encoder. Specifically if the activation functions are purely linear then the auto-encoder essentially becomes PCA (Principal Component Analysis). Using linear features limits the representations that can be learned. Also, they cannot be stacked to form deeper, more abstract

representations since the composition of linear operations yields another linear operation. A practical advantage of training auto-enconder variants is that they define a simple tractable optimization objective that can be used to monitor progress.

The denoising auto-encoder is a stochastic version of the auto-encoder. By adding perturbations to the input it forces the neural network to find hidden representations not available in the train data set. That is, the learner must capture the structure of the input distribution in order to optimally undo the effect of the corruption process. This also enforces the network to discover more robust features and prevent it from simply learning the identity. [12][13]

B. Restricted Boltzmann Machines

Restricted Boltzmann Machines(RBM) is a stochastic neural network. It is likely the most popular subclass of Boltzmann machine. The RBM forms a bipartite graph with the visible and the hidden nodes forming two layer of vertices in the graph (and no connection between units from the same layer). The conditional distribution over the hidden units given the visible units factorize to give:

$$P(h|x) = \prod_i P(h_i|x),$$

where $P(h_i = 1|x) = \text{sigmoid}\left(\sum_j W_{ij}x_j + c_i\right)$. The conditional distribution over the visible units given the hidden units factorize to give:

$$P(x|h) = \prod_i P(x_i|h),$$

where $P(x_i = 1|h) = \text{sigmoid}\left(\sum_j W_{ij}h_i + b_j\right)$. The conditional factorization property of the RBM immediately implies that most inference we would like to make are readily tractable. RBM training consist of maximizing the *likelihood of the training data*. With N training examples, the log likelihood is given by:

$$\sum_{n=1}^{N} \log P(x^n;\theta) = \sum_{n=1}^{N} \log \sum_{h\in\{0,1\}^{d^h}} P(x^n, h;\theta)$$

where h is the hidden nodes, and d^h is the number of hidden units. Gradient descent is generally used to train RBMs. The derivative of the log likelihood,

$$\frac{\partial}{\partial\theta_i}\sum_{n=1}^{N}\log p(x^{(n)} = -\sum_{n=1}^{N}\mathbb{E}_{p(h|x^{(n)})}\left[\frac{\partial}{\partial\theta_i}\mathcal{E}_\theta^{BM}(x^{(n)},h)\right] + \sum_{n=1}^{N}\mathbb{E}_{p(x,h)}\left[\frac{\partial}{\partial\theta_i}\mathcal{E}_\theta^{BM}(x^{(n)},h)\right]$$

where $\mathcal{E}_\theta^{BM} = x^T Wh - b^T x - c^T h$ is the enery function describing the network structure. The RBM conditional independence property implies that the expectation over the data set is readily tractable. The Expectation over the hidden units can be approximated using Gibbs sampling. A common technique is to use Contrastive Divergence, which basically runs Gibbs sampling for only a few iterations.[11][15][14]

C. Training Auto-Encoder With Modified Target Output

The main idea behind the denoising auto-encoder is to force the neural network to find hidden representations. This is achieved by adding perturbations to the input, such as noise. What if our data already is perturbed or has noise inherently? Then adding more noise would not help in finding hidden representations. Adding more noise might actually make it hard to find any representations.

Recall the expression obtained when modifying the output target function,

$$E = \frac{1}{N}\sum_{n=1}^{N}\|\mathbf{x}^n - \boldsymbol{W}\mathbf{x}^n\|_2^2 + \sigma^2\|\boldsymbol{W}\|_2^2$$

where σ^2 is proportional to variance of the noise. The first term is an auto-encoder but with noiseless inputs. The second term is the regularizing term induced by the noise present in the original input. The regularization term prevents the neural network form learning the Identity. The advantage over a normal auto-encoder is that we are learning from a noiseless input and the noise present in the original signal only acts as a regularizing term. Second, compared to the denoising aut-encoder we use the perturbations already present in the input signal to find the hidden representation in the datum.

IV. Experiments

A. Simulated Meteor Echo and Model

We will simulate the problem of classifying between under-dense and over-dense meteor echoes. Over-dense trails have a plasma frequency that exceeds that probing frequency. This means that compared to under-dense echoes they have a longer lifetime. For each type the duration of the echo is consistent. The amplitudes can vary between echoes and is correlated to the decay rate. Therefore, a classifying algorithm is achievable and can be used to detect when either of these meteor echoes is present. Figure 1 from [2], shows the two types of meteor echoes. From the real echoes we can simulate the over-dense and under-dense meteor with the following equations,

$$f_1(t) = A_1\exp(-\tau_1(t-t_1))\mathrm{u}(t-t_0) + \eta$$
$$f_2(t) = A_2\exp(-\tau_2(t-t_1))\mathrm{u}(t-t_1) + (\mathrm{u}(t-t_2) - \mathrm{u}(t-t_1)) + \eta$$

where η is additive white Gaussian noise, $A_i, i \in \{1,2\}$ will model the different meteor amplitudes, $\tau_i, i \in \{1,2\}$ models the different decays rates, and $t_i \in \{1,2\}$ will model jitter noise (random fluctuations in receive time). Figure 2, shows various instances of the simulated echoes along the corresponding noiseless version.

When training the neural network with the modified output target function, the input will consist of several

instances of the signal plus noise as shown in Figure 2. And the output target function will be the noiseless version of those instances, i.e., the blue and orange signals in Figure 2.

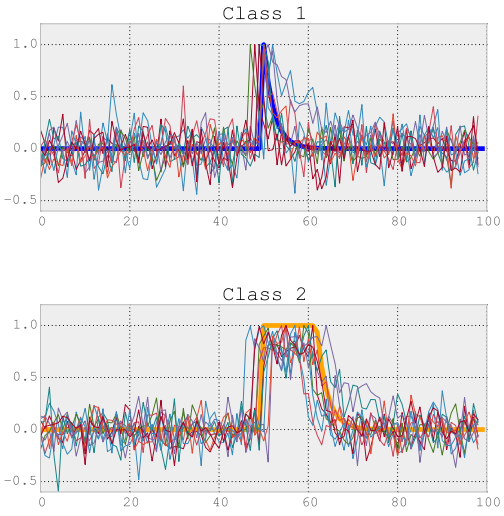

Fig. 2: Simulated Meteor Echo. Model (blue and orange), and Observations

B. Supervised Learning

We construct a two layer neural network with 16 hidden nodes. We training with the simulated data with binary labels and the modified output target function. First we will analyze how these two methods learn. One way to achieve this is to look at the weights learned. In Figure 3, we show the weights learned and the convergence rate for both binary target and modified target functions.

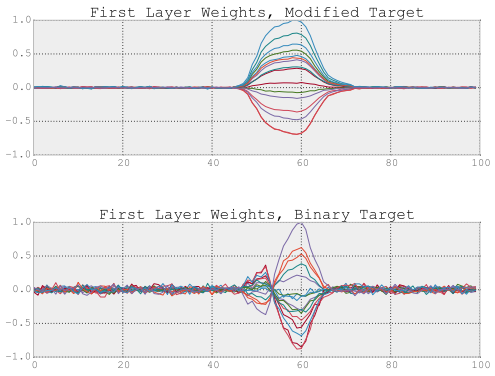

Fig. 3: Weights learned, two layer neural network with 16 hidden nodes

We can see that the weights learned are significantly different. The weights for the modified output target function are simpler and less noisier. This is very important when trying to implement deeper neural networks, since over-fitting is a problem.

To compare in terms of computational cost, we look at the convergence rate. The convergence rate for a neural network is how many iterations it needed to hit the minimum in the error function. From 4 we can see that the modified output target function drastically outperforms binary target labels. We see an improvement of more than half. This means that by modifying the output target function we have achieved a better behaved error function.

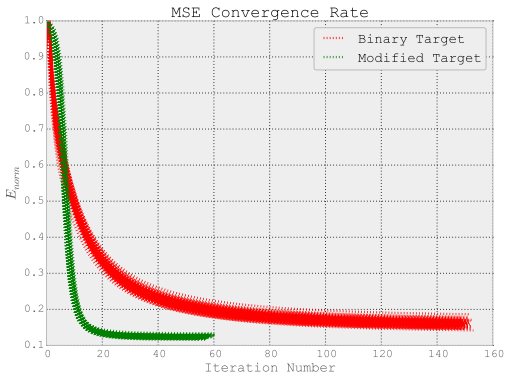

Fig. 4: Convergence rate, two layer neural network with 16 hidden nodes

Now, to test it as a classifier we compare error rates. Table I and II show the error rates. We use SVM, with raidal basis kernel, as our reference point. For a two layer neural network with 16 hidden nodes, our implementation is superior to the normal implementation of a neural network (binary target). Another improvement is in implementing deep neural networks. If we increase the number of layers we see that the implementation of binary target starts failing and does not converge which leads to large standard deviations. Our implementation does not suffer, actually it improves the error rate getting closer to SVM.

Method	**error rate ± std**
NN Binary Target (16)	3.46 ± 0.404
NN Modified Target (16)	2.37 ± 0.313
SVM (RBF kernel)	1.17 ± 0.268

TABLE I

Method	**error rate ± std**
NN Binary Target ($600 \times 2 \times 600$)	8.65 ± 12.401
NN Modified Target ($600 \times 2 \times 600$)	1.51 ± 0.321
SVM (RBF kernel)	1.16 ± 0.256

TABLE II

C. Representation Learning

We compared versus PCA, RBM, and denoising autoencoder. The set up is as follows: For PCA we used the

weights from first two principles components with the highest eigenvalues. In RBM we constrained the number of hidden units to be two. Similarly for the auto-encoder, denoising auto-encoder, and our implementation we constrained the number of hidden nodes to be two. Then the weights learned were used to project a new instance of our datum to a two-dimensional space.

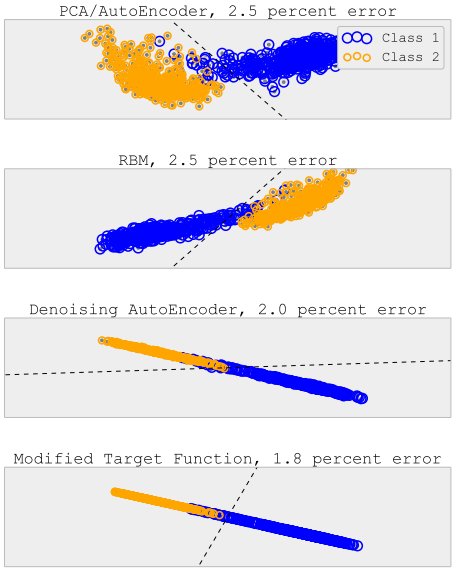

Fig. 5: 2 dimensional representations of the datum

It is clear the our method performs at least as good if not better than the denoising auto-encoder. Further test need to be made, were instead of using two hidden nodes we use dimensions larger than the input vector. This will force the neural network to find better representations.

V. Conclusion

In conclusion, we have shown that by modifying the output target function of a neural network we were able to improve performance both in supervised and representation learning. The results are not conclusive yet. First we need to apply this method to real world data. Second in representation learning, we have not used our method to its full potential. It is our belief that it can perform better at learning representation to train deep neural networks. This is of course, future work.

References

[1] C. M. Bishop, J. J. T. Avenue *Training with Noise is Equivalent to Tikhonov Regularization*, 1995
[2] S. Zhao, J. Urbina, L. Dyrud, R. Seal, *Multilayer detection and classification of specular and nonspecular meteor trails* , Radio Science, 46(6), 2011
[3] G. Cybenko, *Approximations by superpositions of sigmoidal functions*, Mathematics of Control, Signals, and Systems, 2 (4), 303-314, 1989
[4] K. Hornik, *Approximation Capabilities of Multilayer Feedforward Networks* , Neural Networks, 4(2), 251257, 1991
[5] S. Haykin, *Neural Networks: A Comprehensive Foundation*, Volume 2, Prentice Hall. ISBN 0-13-273350-1, 1998
[6] M. Hassoun, *Fundamentals of Artificial Neural Networks*, MIT Press, p. 48, 1995
[7] C. M. Bishop, *Pattern Recognition and Machine Learning*, 2006
[8] T. Hastie, R. Tibshirani, J. Friedman, *The Elements of Statistical Learning* (Second Edition), 2009
[9] Y. Bengio, *Learning Deep Architectures for AI. Foundations and Trends in Machine Learning* , (Vol. 2, pp. 1127), 2009
[10] G. Hinton, R. Salakhutdinov, *Deep Boltzmann Machines*, (2009)
[11] Y. Bengio, A. Courville, P. Vincent, *Representation Learning : A Review and New Perspectives*, 134, 2012
[12] P. Vincent, H. Larochelle, Y. Bengion, P.-A. Manzagol, *Extracting and composing robust features with denoising autoencoders*, Proceedings of the 25th international conference on Machine learning - ICML 08, 10961103, 2008
[13] P. Vincent, *Stacked Denoising Autoencoders : Learning Useful Representations in a Deep Network with a Local Denoising Criterion*, 11, 33713408, 2010
[14] Q. V. le, J. Ngiam, A. Coates, A. Lahiri, B. Prochnow, A. Y. Ng, *On optimization methods for deep learning*, 2011
[15] H. Schulz, M. Andreas, S. Behnke, *Investigating Convergence of Restricted Boltzmann Machine Learning*, 1-9, 2010
[16] G. Hinton, N. Srivastava, A. Krizhevsky, I. Stutskever, R. Salakhutdinov, *Improving neural networks by preventing co-adaptation of feature detectors*, (2012)
[17] H. Larochelle, Y. Bengio, P. Lamblin, *Exploring Strategies for Training Deep Neural Networks*, (2009)
[18] G. Hinton, R. Salakhutdinov, *Reducing the Dimensionality of Data with Neural Networks*, (2006)