Efficient Encoding Using Deep Neural Networks

Chaitanya Ryali **Gautam Nallamala** **William Fedus** **Yashodhara Prabhuzantye**

Abstract

Deep neural networks have been used to efficiently encode high-dimensional data into low-dimensional representations. In this report, we attempt to reproduce the results of Hinton and Salakhutdinov [**?**]. We use Restricted Boltzmann machines to pre-train, and standard backpropagation to fine-tune a deep neural network to show that such a network can efficiently encode images of handwritten digits. We also construct another deep autoencoder using stacked autoencoders and compare the performance of the two autoencoders.

1 Introduction

In the last decade, neural networks that can efficiently encode natural images have been found [**?**]. The breakthrough, in part, was due to the observation that deep neural networks that are pre-trained, unsupervised, to the distribution of inputs and fine tuned by supervision, can exhibit much better performance in representation and classification. One particular unsupervised pretraining procedure, the restricted Boltzmann machine (RBM), exhibits a striking similarity to a technique from physics - the renormalization group - used to describe the theory of phase transitions. Particularly, the renormalization group method is a scaling process, used to integrate out degrees of freedom of a system and preserves statistics, due to inherent scale invariance. In fact, a direct mapping between RBMs and the renormalization group has been found, and it has been shown that a deep belief network (DBN) trained using RBMs can be used to efficiently represent a system of 2D Ising spins close to the critical temperature [**?**]. In our report, we set out to understand why natural images can be efficiently encoded by deep neural networks by exploring the connection with physics.

As a first step, we attempted to implement our own autoencoders that can reproduce the results of the paper by Hinton and Salakhutdinov [**?**], specifically on images from the MNIST database. The MNIST database contains labeled handwritten digits and is used as a standard benchmark to test classification performance. As in the paper, using our own code, we build a deep autoencoder that is pretrained with RBMs and fine-tuned with backpropagation. We show that our deep autoencoder can efficiently encode the handwritten images and we highlight the differences between pure pre-training and pre-training combined with fine-tuning. To understand the differences between an RBM based auto encoder and other autoencoders, we also build another kind of deep autoencoder greedily pretrained using stacked autoencoders and again fine-tuned with backpropagation. In the last few years, it has been discovered that pretraining is not necessary to train deep neural networks. These deep neural networks, trained directly via backpropagation, use a rectified linear activation unit (relu) instead of a sigmoid or a tanh function. We attemped to build such a network, however we did not succeed in training the network; it appears that the training requires greater computational power and data sets that are much larger than the MNIST data set.

2 Restricted Boltzmann Machines

2.1 Overview

An RBM is a stochastic neural network which learns a probability distribution over its set of inputs. RBMs are Boltzmann machines subject to the constraint that their neurons must form a bipartite

graph. This restriction allows for efficient training using gradient-based contrastive divergence. As a simple example, we can consider our training set to be binary vectors, where an instance of an image has been mapped into a binary format. The visible units of the RBM ($\boldsymbol{v}$) will correspond to the pixels of the image and the hidden units ($\boldsymbol{h}$) represent the feature detectors as seen in Figure 1.

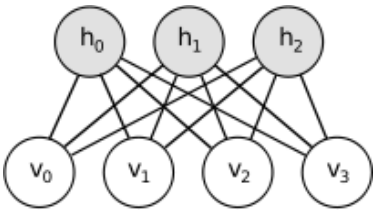

Figure 1: Simple schematic of a Restricted Boltzmann Machine with four visible nodes and three hidden nodes.

For an RBM, we may define an energy of a particular state configuration $(\boldsymbol{v}, \boldsymbol{h})$ to be Equation 1

$$E(\boldsymbol{v}, \boldsymbol{h}) = -\sum_{i \in \text{visible}} a_i v_i - \sum_{j \in \text{hidden}} b_j h_j - \sum_{i,j} v_i h_j w_{ij} \tag{1}$$

where v_i, h_j are binary states of visible unit i and hidden j, respectively, and a_i, b_j are the biases to those nodes and w_{ij} is the weight between them. In direct correspondence to classical statistical physics, with an energy defined for the system, we may now assign a probability to every possible pair of visible and hidden nodes according to Equation 2

$$p(\boldsymbol{v}, \boldsymbol{h}) = \frac{e^{-E(\boldsymbol{v}, \boldsymbol{h})}}{Z} \tag{2}$$

where $Z = \sum_{\boldsymbol{v}, \boldsymbol{h}} e^{-E(\boldsymbol{v}, \boldsymbol{h})}$ is called our partition function which simply normalizes the probabilities to unity. With this formulation, we can increase the probability that the system assigns to an image in our training set by adjusting the parameters $\boldsymbol{a}, \boldsymbol{b}, w$. By Equation 2, we see that this is mathematically equivalent to *minimizing* the energy of input states. To achieve this, we may use a simple gradient ascent procedure where we seek w_{ij} that (approximately) maximizes the log-likelihood of the visible nodes as seen in Equation 3.

$$w_{ij} = w_{ij} + \epsilon \frac{\partial \log p(\boldsymbol{v})}{\partial w_{ij}} \tag{3}$$

$$= w_{ij} + \epsilon \left(\langle v_i h_j \rangle_{\text{data}} - \langle v_i h_j \rangle_{\text{model}} \right) \tag{4}$$

where ϵ is the learning rate, $\langle v_i h_j \rangle_{\text{data}}$ is the expectation under the data distribution and $\langle v_i h_j \rangle_{\text{model}}$ is the expectation under the model distribution. A simpler update rule can be derived for the biases. The unbiased sample of $\langle v_i h_j \rangle_{\text{data}}$ may be simply found by randomly selecting a training image $\boldsymbol{v}$ and then setting binary state h_j to 1 according to

$$P(h_j = 1 | \boldsymbol{v}) = \sigma(b_j + \sum_i v_i w_{ij}) \tag{5}$$

where $\sigma(x)$ is the standard logistic sigmoid function. Now with $\boldsymbol{h}$ set, we may get a reconstruction of the input via

$$P(v_i = 1 | \boldsymbol{h}) = \sigma(a_i + \sum_j h_j w_{ij}) \tag{6}$$

2.2 Pretraining via Contrastive Divergence

To calculate correlations within our model $\langle v_i h_j \rangle_{\text{model}}$,, we may use block Gibbs sampling. As described earlier in Equations 5 and 6, we sample hidden units given a randomly set visible configuration v, and then reconstruct the input given the previously obtained hidden configuration h. The process is repeated for n steps (Figure 2).

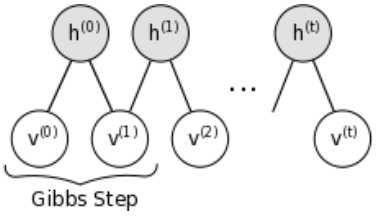

Figure 2: A graphical representation of Gibbs sampling where $v^{(n)}$ and $h^{(n)}$ refer to the set of *all* the visible and hidden nodes at the n-th step of the Markov chain, respectively.

This procedure is guaranteed to give accurate samples drawn from $p(\boldsymbol{v}, \boldsymbol{h})$ given $n \to \infty$. However, the required expectation $\langle v_i h_j \rangle_{\text{model}}$ converges slowly. Therefore, we use a contrastive divergence algorithm [**?**] that speeds up the convergence. The contrastive divergence algorithm CD_n performs Gibbs sampling with n steps by instead starting with a visible configuration drawn from the training set. In our implementation (as in [**?**]), we use CD_1. Specifically, we first drive the hidden units using a training vector. The hidden units are then set stochastically to binary values based on Equation 5. These stochastic binary values for the hidden states are used to drive reconstructions of the visible units with deterministic values in $[0, 1]$. The product $\langle v_i h_j \rangle_{\text{model}}$ is then formed from deterministic hidden units driven from the reconstructed visible units. Additionally, instead of updating the weights for each single example, we choose to update the weights over mini-batches, in accordance with the recommendations outlined by Hinton [**?**].

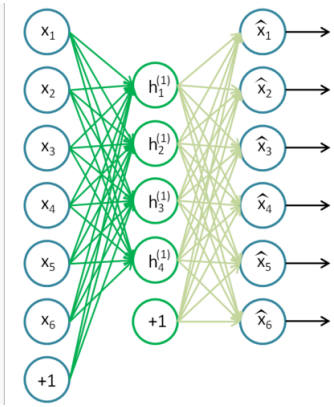

Figure 3: An example of an autoenccoder.

3 Autoencoders and Stacked Autoencoders

Using stacked RBMs is one way of unsupervised pre-training. Another method of unsupervised pre-training is to use stacked autoencoders.

3.1 Autoencoders

An autoencoder neural network is an unsupervised learning algorithm that applies backpropagation, setting the target values to be equal to the inputs, i.e $t^{(i)} = x^{(i)}$. The output of the network is a reconstruction of the input, i.e $y^{(i)} = \hat{x}^{(i)}$. The cost function is

$$J(W,b) = \frac{1}{2}\sum_{i=1}^{m}||x^{(i)} - \hat{x}^{(i)}||^2 + \frac{\lambda}{2}\sum_{l=1}^{n_{l-1}}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(W_{ji}^{(l)})^2,$$

where the first term is just the sum squared error of reconstruction and the second is a weight decay term. The network learns a compressed representation of the data, when it's forced to reconstruct the input from activations of a hidden layer with a smaller number of dimensions than the input dimension, see fig [3] for an illustration. This kind of autoencoder (which is a non linear method), with a single hidden layer usually learns something similar to PCA, a linear dimensionality reduction method. As in [**?**], we can enforce sparsity constraints upon the activations of the hidden layer of the autoencoder.

3.2 Stacked Autoencoders

Much like stacking RBMs, we stack autoencoders by feeding the activations of the hidden layer to the hidden layer of the next autoencoder, i.e we treat the activations of the previous hidden layer as the input to the next autoencoder, allowing us to learn higher-order features, and hopefully a better representation of the data, see fig [4] for an illustration.

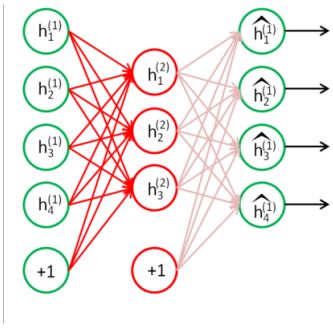

Figure 4: Greedy layer-wise training of an autoencoder.

4 Implementation and Results

We use the MNIST data set, which is readily available and may be downloaded from http://yann.lecun.com/exdb/mnist/. The data set consists of 60,000 28x28 labeled hand-written digits. It has been shown that a simple softmax regression on MNIST images encoded using the deep encoder as constructed below gives state-of-the-art classification performance. In this report, however, we will focus on reconstructing the images independent of the labels. The inputs and outputs are the 784 dimensional vectors drawn from raw MNIST images. Our goal is to reduce the dimensionality of these images to 30 dimensions. The details of our implementations and results obtained are given below:

4.1 Deep autoencoder using RBMs

We require the use of an encoding multilayer network to reduce the original high-dimensional data into a low-dimensional representation and a decoder multilayer network to recover the original high-dimensional input. A schematic of the network architecture is depicted in Figure 5

The neural network is composed of a stack of RBMs with each having one layer of feature detectors. The weights of a single layer are tuned one layer a time with the output of a lower layer used as the input to the deeper layers. Optimization of the weights of deep nonlinear autoencoders is difficult. If the weights are pretrained, however, the weights are sufficiently close to an optimum and gradient-based optimization works well.

We use our own RBM pretraining and backpropagation code written from scratch. The network architecture was 784-500-200-30, which was then unrolled as in Figure 5. The RBM training proce-

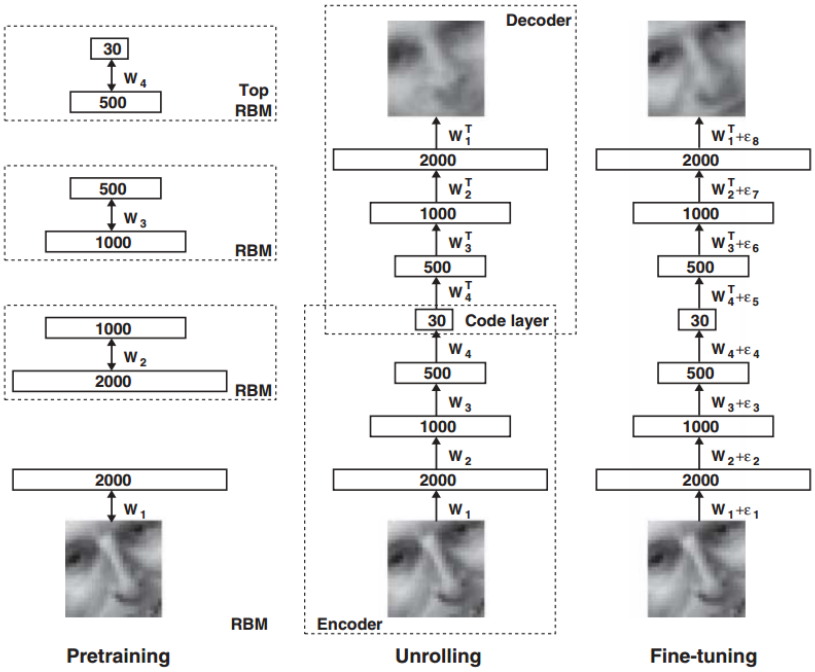

Figure 5: The proposed network consists of a series of stacked restricted Boltzmann machines (left). The RBMs are unrolled after training (center), giving a pretrained deep encoder-decoder network. The deep network is then fine-tuned by backpropagation (right). Figure taken from [**?**].

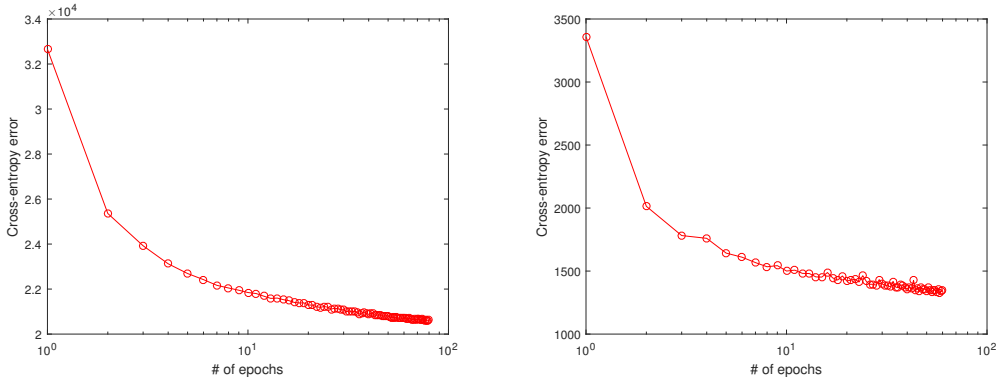

Figure 6: Reconstruction error. Left: Cross entropy error during the training of the first layer's RBM. Right: Cross entropy error during backpropagation.

dure is as described in Section 2.2. We (approximately) optimize the log-likelihood by performing stochastic gradient ascent on the weights and biases of the network. The weights were initialized to mean zero normal distribution with 0.1 deviation and the biases were initialized to zero. The positive and negative gradients in Equation 3 are averaged over minibatches of size 100. The SGD consists of 100 epochs over the entire data set with momentum changed from 0.5 to 0.9 after 60 epochs. The learning rate and weight decay (for L^2 regularization) were fixed at 0.1 and 0.0002 respectively. The RBM of the final layer is special - the visible units are reconstructed using linear activation functions instead of a sigmoid. The training procedure is therefore slightly different - the activations of the hidden units formed by linear combinations of the visible units in the first step of CD_n are peppered with standard normal noise. We learned that this linear layer is crucial; when we replaced the linear unit by a sigmoid activation unit the reconstruction performance decreased sharply. Because of the unbounded activations in this layer, the learning rate was lowered to 0.001.

Figure 7: Comparison between digits reconstructed from a 30-dimensional encoding. From top to bottom: Original, RBM + backpropagation, RBM only and PCA.

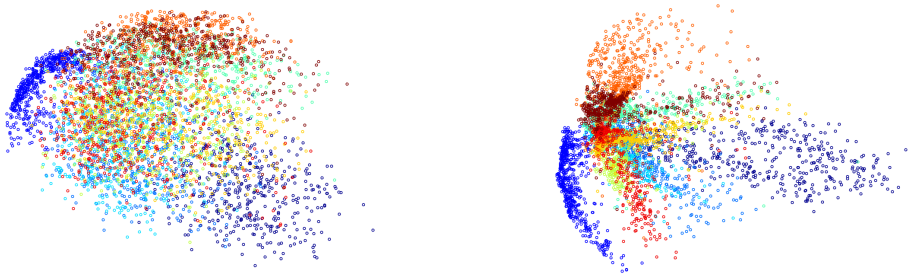

Figure 8: The two dimensional reduction of MNIST digits with each digits color-coded. Left: RBM pretraining only. Right: After fine-tuning.

After pretraining, the network was fine-tuned using standard backpropagation on a cross-entropy loss function. The weights were initialized from the RBM procedure. The hyperparameters (learning rate, momentum, weight decay) were similar to those used for training the nonlinear RBM layers. Stochastic gradient descent was performed for 150 epochs using minibatches of size 1000.

Figure 6 shows that the training procedure converges. The comparison between reconstructions of digits drawn randomly from the test set is shown in Figure 7. We observe that the difference between pure pre-training and pretraining with backpropagation is not very significant, but both perform significantly better than a simple principal component analysis. We further constructed a 784-500-200-2 encoder network that gives a two-dimensional representation for the originally 784 dimensional data. The results, shown in Figure 8, clearly show that the backpropagation helps in distinguishing the digits.

4.2 Deep autoencoder with RELU units

We use the same network architecture: 784-500-200-30, without depending on pre-training for weight initialization and directly train the network using back-propagation. We note extremely slow convergence as there was no significant decrease in the cost function even after 20 epochs. We have not succeeded in training this network.

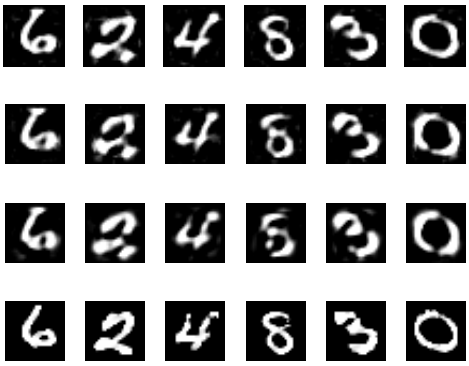

Figure 9: Reconstructions, from top to bottom: Deep Autoencoder (with pre-training using stacked autoencoders) 90 epochs, 30 epochs, PCA, original.

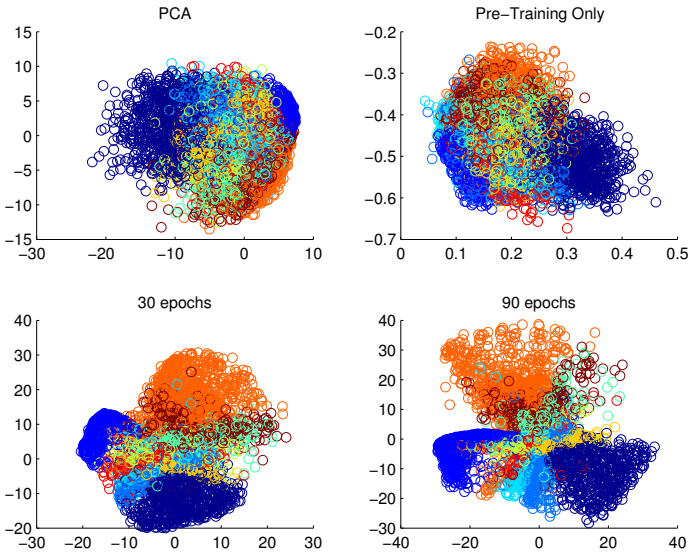

Figure 10: Scatter plots: Mapping to 2 dimensions using PCA, only pre training, after 30 epochs, after 90 epochs.

4.3 Deep autoencoder using stacked autoencoders

As discussed previously, we have pre-trained the network for weight initialization by greedy layer-wise optimization of weights using stacked autoencoders. We maintain the same network architecture: 784-500-200-30. We then fine-tune the parameters by back-propagation.(We used [**?**] for pre-training only.). The weight decay hyperparamter was set to 10^{-6} and the mini-batch hyperparameter was set to 100. The momentum hyper-parameter was set to $\mu = 0.5$.

We plot reconstructions of some samples for comparison in fig 9. We note that PCA and a deep auto encoder with only 30 epochs of training are comparable, although the autoencoder seems to be performing slightly better than PCA. As we fine tune the parameters of out network via back-propagation, we note the improved performance in reconstruction.

To visualize the learning and to note the effect of pre-training and pre-training with back-propagation, we also use 784-500-200-2 architecture to allow visualization via scatter plots, see

fig 10. We note the similarity between the PCA scatter plot and pre-training only scatter plot. We note better clustering of digits with 30 epochs of back propagation and the better reconstruction relative to PCA we noted in fig 9 makes sense in this context. We also note the improving clustering with more epochs, clearly the network hasn't converged yet.

5 Conclusions

Although the vanishing gradient problem has been solved for deep networks, pre-training can still be a useful tool. In literature, it has been found to improve accuracy and reduce training time. In this project, we tried to implement a deep autoencoder without pre-training using relu units to combat the vanishing gradient problem, but were unsuccessful in training the network. We believe this is due to extremely slow convergence due to poor initialization of weights. In training deep networks where it has been found unnecessary to perform pre-training are typically cases with very large data sets. Although, in general, pre-training via stacked autoencoders and via stacked RBMs are considered comparable, we note here in particular that we see excellent performance just from stacked RBMs and this shows improvement upon fine tuning via back-propagation as noted in figs 7 and 8. We note that the deep autoencoder performs much better than PCA. In fact, stacked RBMs without fine tuning is also seen to outperform PCA, which demonstrates the expressive power of stacked RBMs; we achieve good compression since we're able to estimate the probability distribution of the images well.

We also train a deep auto-encoder using stacked autoencoder pre-training method. We note that in this case, this method of pre-training isn't as effective as stacked RBMs, and isn't really much better than PCA, although the performance rapidly improves upon fine tuning via back-propagation, see fig 9. This improvement is once again seen in 10. We note that even when we don't have a very large training set, we are sometimes able to use pre-training to train deep neural networks and benefit from learning higher-order features, in addition to improving training time and accuracy.

In conclusion, we note that the important aspect of any encoding algorithm is scalability. The observation that deeper networks with many more parameters, as compared to PCA for example, can give better representations is non-trivial. The two-dimensional reductions shown in figures 8 and 10 show that there are indeed sub-spaces that capture the essential features of the original higher dimensional spaces. The decomposition to these nicer sub-spaces is nonlinear, as shown by the comparison to PCA. For PCA, we assume a linear decomposition and the measure of how much we lose during the decomposition is given by the loss in variability. We can therefore have good estimates of how many linear components we would require to capture features up to some threshold. For non-linear reductions, however, we are not aware of such a measure. It will be interesting to define such a measure of the variability captured by neural network representations. This will provide bounds on how much information we can capture in a network, the optimal number of parameters required to capture a certain amount of information and therefore quantify how scalable a neural network can be to larger dimensional data with highly non-linear features. We can further argue that the non-linearities are due to strong correlations in the data that extend beyond simple local correlations. Moreover, there is evidence that neural networks can indeed capture correlations within such data [**?**]. In order to understand the mechanism of deep networks, one method perhaps could be to generate data with complicated long-range correlations but whose structure is well-known, and try to understand how a deep network solves the problem of efficient encoding.

6 Contributions

Gautam wrote the code for RBM pre-training and backpropagation, and was helped by William. William and Chaitanya independently wrote the code for a deep autoencoder without pre-training and using relu units. Chaitanya wrote the code for fine-tuning via back propagation for the deep autoencoder pre-trained via stacked autoencoders. Gautam, William and Chaitanya wrote the report together.

bib