Optimizing Deep Belief Networks for Small Data Sets

Ara Jermakyan, *UCSD*

Abstract

With the explosion of Deep Learning, huge datasets and computational resources are called to push the limits of Machine Learning and Artificial Intelligence. However, some datasets are small in size, and thus special care must be given to these datasets. Deep Belief Networks are generative models that create data compared with their discriminatory counterparts which simply classify data. Deep Belief Networks, while outdated, can still be shown to perform well compared with many methods. However Deep Belief Networks, like other Deep Learning architectures, are dependent on data. Optimizing these networks for smaller datasets can prove to bring a new dimension to common techniques and algorithms. We will showcase our attempts to improve the performance of a Deep Belief Network for small datasets.

1 Introduction

Machine Learning has come a long way in the last 50 years, specifically in the fields of Deep Learning. From the introduction of the perceptron to the recent Convolutional Neural Networks (CNN), Deep Learning techniques are slowly evolving over time. Recent advances in computational resources including Graphics Processing Units (GPU's), as well as in optimization techniques and numerical methods have allowed previously unsolvable problems to take precedent today.

The main differentiator between Deep Learning compared with other machine learning techniques is the use of "hidden layers." Instead of finding a correlation between the outputs with the inputs of the system, the architecture will detect latent features over and over again until the output is determined. For example, when detecting numbers from an image, hidden layers can take pixel input, find patterns, and then correlate the output with these patterns, rather than directly correlating pixel locations with the output.

Most deep learning algorithms utilize a discriminative approach, modeling a posterior $p(y|x)$, and ultimately classify y based off of x directly [2]. A generative model on the other hand models the joint probability $p(x, y)$. For classification of $p(y|x)$, Bayes' theorem can be utilized on the joint probability [2]. While generative models are used to create new data, discriminative models tend to outperform their generative counterparts when it comes to classification.

On the contrary, Geoffrey Hinton discovered a way to utilize a generative model to ultimately beat discriminative models utilizing Deep Belief Networks (DBN) [3]. Instead of training solely through backward-propagation, DBN's train each layer at a time in a pre-training phase, and then apply backpropagation for the actual classification. The idea here is that before someone can classify images based off of their raw input right away, a person should first learn what images are, and what features and patterns are typically constituted in these images, before classifying them [1].

These systems build off of having plenty of data to work with. However, some datasets simply do not have as much data needed to drive these algorithms. Decoupling the DBN from this data constraint can showcase its applicability for these datasets. This paper attempts to build on top of Hinton's DBN's by attempting to optimize DBN's for smaller datasets. The Support Vector Machine (SVM) is used as a baseline for our algorithms, since it is one of the best performing discriminate models [1], and was used in Hinton's original paper to showcase DBN's success.

The organization of this paper is as follows: in Section 2, we describe background information needed for DBN's. In Section 3, we define the experimental setup and the architectural decisions made for testing. In section 4, we describe the results obtained for the different experiments and its comparison to SVM. In section 5 we conclude and discuss the possibility of future work.

2 Background

2.1 Restricted Boltzmann Machine

A Boltzmann Machine (BM) is essentially input neurons interconnected with other hidden units in order to determine the latent features (represented by the hidden units). Optimizing this system depends on minimizing the overall energy function of the system, as well as minimizing the reconstruction error from the hidden units. This is done by randomly sampling the units, and update the weights after the system achieves equilibrium. Due to the many connections, this ends up taking too long to find a global minimum [6].

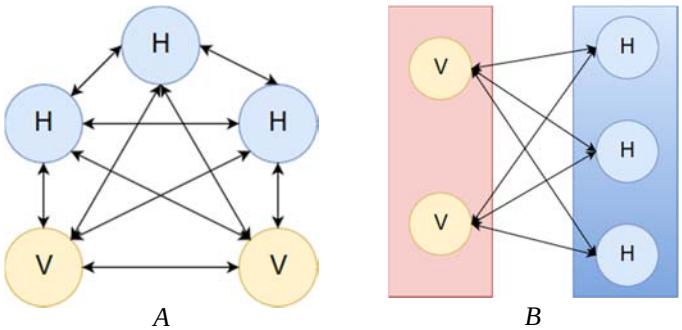

Figure 2.1. A) showcases a Boltzmann Machine where each element is interconnected. B) showcases a Restricted Boltzman Boltzman Machine where the elements are split into two categories and are not connected to each other.

An RBM speeds up the algorithm in two ways. First, the visible units and the hidden units are arranged into a bipartite graph; in other words, the only connections that exist are between visible and hidden. This not only removes the connections between each layer, but allows for multiple units to be updated in parallel at the same time. Second, utilizing an algorithm known as Contrastive Divergence, the system will only calculate *k* steps in the Markov Chain instead of waiting reach full convergence. This speeds up learning under the assumption that the optimal model will be found after enough epochs [3].

2.2 Contrastive Divergence

Contrastive Divergence (CD) is mathematically driven, but can be described on a high-level as a "learning" and "un-learning" phase. In the learning phase, the hidden units learn from the visible units, and ultimately represent the latent features of the input data. The unlearning phase then attempts to generate fantasies from these latent variables. This can be thought of as attempting to prevent the weights from over-fitting [3].

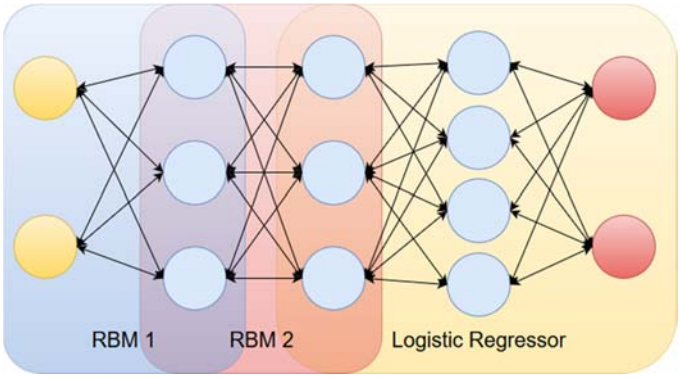

Figure 2.2. A deep belief network consisting of two RBM layers and a Logistic Regressor

2.3 Deep Belief Networks

A DBN is a Multi-Layer-Perceptron (MLP) network where each layer is an RBM as shown in figure 2.2. As mentioned earlier, a generative model simply generates new data. Thus, on top of the RBM's, an extra logistic regression layer is added for classification. At first glance, this is a regular MLP used for backpropagation learning. However, instead, the DBN learns in two stages.

First the RBM stack pre-trains, and learns its features one layer at a time. In other words, the model must first understand what the data is, and what patterns are important within this data. After pre-training, the RBM then connects to the logistic regressor and fine-tunes its parameters to better classify the images. This is simply learning how to classify the data based on the patterns that were learned rather than purely on the results.

3 Implementation

3.1 Framework and Resources

The experiments that were conducted were done using the Theano framework, a tensor-based library used for deep learning applications in python. This framework was chosen due to its small resource footprint on laptops as well as the author's proficiency in the framework. The code utilized to implement the RBM and DBN originated from https://deeplearning.net. The experiments updated the code and tweaked the hyperparameters to ensure a better performance.

For the comparison to SVM, sklearn library was utilized due to is simplicity in running, as well as the author's proficiency in that framework.

Lastly, the experiments were run on a local laptop utilizing a Geoforce 940MX GPU due to resources available to the author.

3.2 Data

We will use the MNIST data set for our experiments. To ensure comparable tests, we will use the same random seed in each experiment, and shuffle the data. We will ensure that there is equal representation from each of the ten classes. Data samples from 1000-5000 where chosen due to maximize time for learning from the algorithm.

3.3 Assumptions

- Using the RBM and DBN implementations, our hidden unit weights are initialized according to [4]'s specifications, which are dependent on our sigmoid activation function.
- The hidden unit and visible unit biases in each RBM layer is initialized to 0.
- The pre-training phase is constrained to 100 epochs for each layer, while fine-tuning is constrained to 1000 epochs.
- The CD algorithm will converge over the many epochs, and convergence at each step is not needed.
- Since we are running for many epochs, we are not interested in finding the best performance in the shortest time possible. Therefore the learning rates were held constant and utilized stochastic gradient descent, with batch-size of 10
- The fine-tuning backpropagation algorithm has the option of stopping early if overfitting is detected. Pre-training does not have this check in place, and thus manual observations will be made after the experiments run. The overfitting will be detected by looking at the differences of global energies between the training and validation data and ensuring it does not grow over time [5].

- The Logistic Regressor connected to the RBM stack will contain 500 hidden units
- SVM will run with the same data samples used for the DBN, using a regularization value of 10 and 100. This gave the best results based on the experiments.

3.4 Experiments

Given the previous assumptions, we will discuss our experiments. Our goal is 1) to have the DBN perform better than the SVM utilizing the same dataset and 2) optimize the DBN to obtain the best possible results. We will first obtain a baseline from the SVM in our early experiments. Then we will attempt to minimize the error rate and get it as close to 1.37%, which is our best result when running a DBN on the entire dataset.

We ran each experiment multiple times took the average of the results. Thus our experiments altered the following:

- Sample Sizes for training data
- Sample Sizes for validation and test data
- Pre-training learning rate
- Fine-tuning learning late
- Number of hidden layers
- Number of units within each layer
- Number of iterations (k) for Contrastive Divergence

4 RESULTS

4.1 Varying Training Samples

The first set of tests revolved around varying training sample size (1000, 2000, 3000, 4000, 5000). For the SVM, it performed best when its regularization parameter was 10 and 100. As seen in figure 4.1, both SVM and DBN performed better with increasing sample size. Thus our baseline for 5000 samples based on SVM is 8.5%. The DBN clearly outperforms the SVM.

4.2 Varying Validation and Test Samples

In these set of tests, we varied the validation and test set sample sizes (500, 1000, 1500, 2000). As seen in Figure 4.2, the error rate rose with increasing sample size. This may be attributed to the lack of training examples, and thus larger test sets will introduce new data that the model has never seen. We kept these sets to 1000 samples each, since it performed the best consistently.

4.3 Pre-Learning Rate

In these set of tests, we altered different learning rates for pre-training of our DBN (0.5, 0.1, 0.07, 0.04, 0.01). As shown in Figure 4.3, the model performed best when the learning rates were within the range of 0.01-0.1. From our data, it seemed that 0.04 performed the best and was the most consistent.

We used the test error to make our decisions here. A better indicator of this value may lie in analyzing the validation set error at the end of pre-training. More research will be needed to be done here.

4.4 Varying Number of Hidden Units in Each Layer

In these tests, we constrained our tests to use one or two RBM layers. For one layer, we varied the hidden layer size (500, 1000, 1500, 2000). As shown in Figure 4.4, the test performance was best when a hidden layer had a 1000 or more hidden units. There does not seem to be a dramatic improvement between using 1000 units or 2000 units for a single layer. This could imply that some saturation or over-fitting of the features is occurring [7].

For testing our second layer (1000, 1500, 2000), the first layer was constrained to 1000 units. As shown in Figure 4.5, no conclusion can be drawn. However, this seems to imply that the layer size needs to be constrained: having too few units means that not enough features are being captured, whereas having too many units saturates the learning process.

4.5 Varying Number of Hidden Units in Each Layer

Since we could not reach a conclusion in the last batch of tests, we decided to test the number of layers with both 100 units and 200 unties, as shown in Figure 4.6 and 4.7. For 1000 units, it seems that three layers are optimal. However, for 2000 units, 1 and 4 layers perform the best. Again, not much conclusion can be drawn from this analysis. However, one can assume that having less hidden units will require more layers compared with having layers with more units.

The results found in 4.6 and 4.7 are similar to Theis et. al's tests in that adding more layers does not improve the result, and sometimes does much worse. Therefore, this further confirms that having some sort of compromise between the number of layer and layer size is required [6, 7].

Regardless, the system with 2000 units, performed much better than the system with 1000 units. Due to computational resources and time, the rest of the tests were carried with three layers with 1000 hidden units each.

4.6 Fine-tuning Learning Rates

As shown in Figure 4.8, we tried varying different learning rates to optimize the backpropagation phase. Having a learning rate of 0.01-0.07 seems to give the most optimal results and thus we chose 0.07 for being the most consistent.

Just like choosing the pre-training learning rate, we based it off of the base error rate. It may be more correct to instead compare the validation error rates at the beginning and end of the fine-tuning stage for a better indicator for this hyper-parameter.

4.7 Varying K Values

For all the tests done so far, we assume that the CD algorithm will only run for one iteration. As mentioned previously, this helps speed up the training process at the cost of not fully

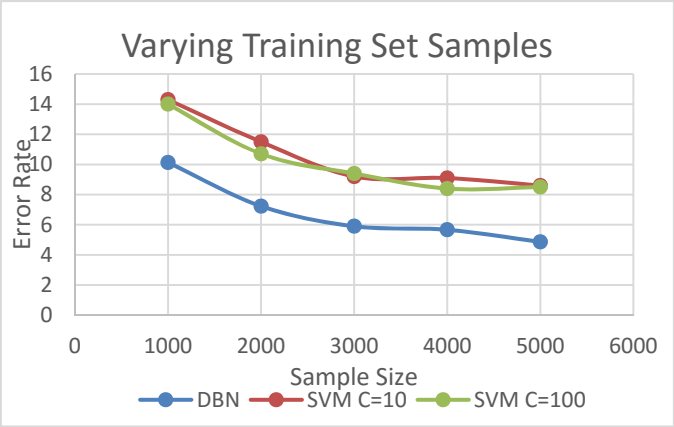

Figure 4.1

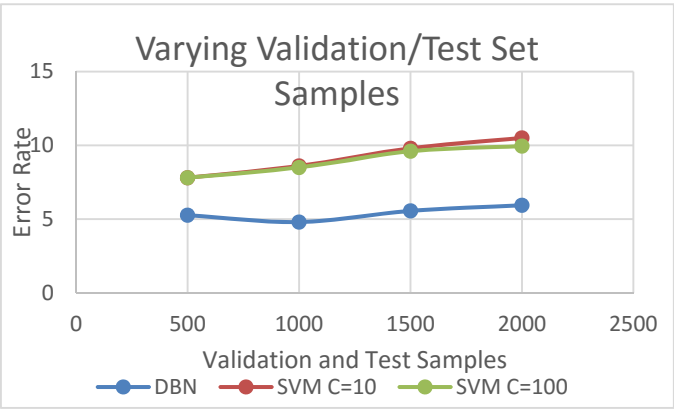

Figure 4.2

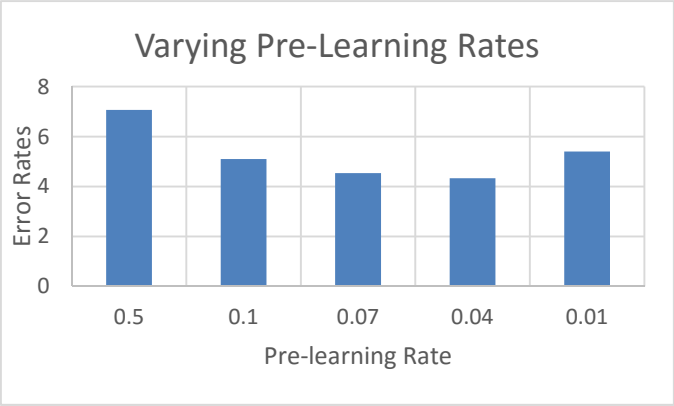

Figure 4.3

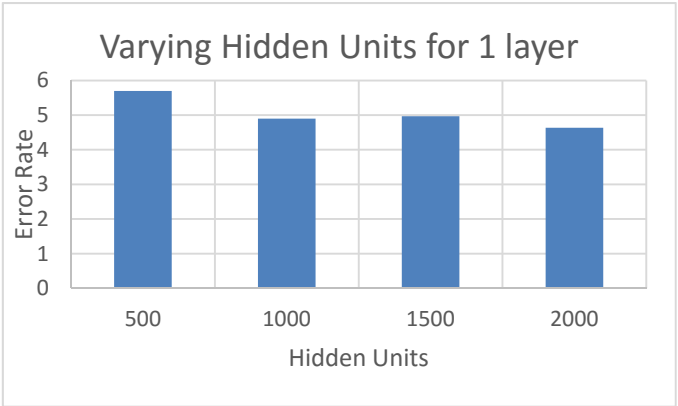

Figure 4.4

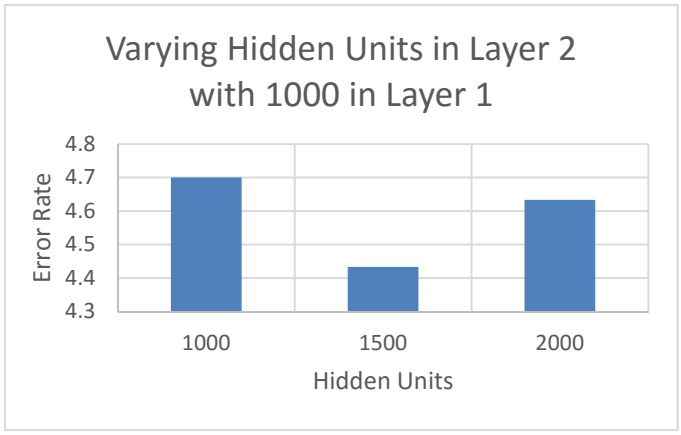

Figure 4.5

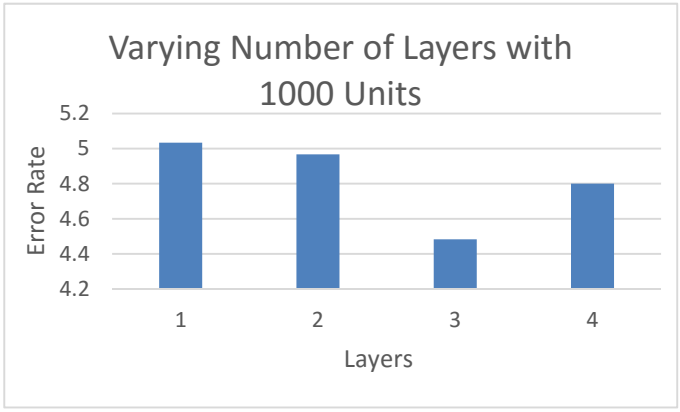

Figure 4.6

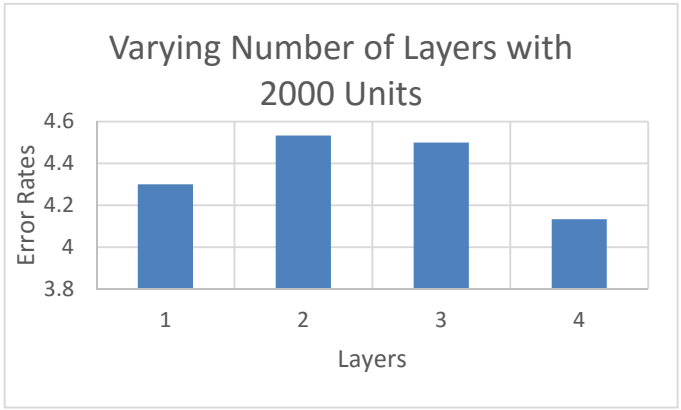

Figure 4.7

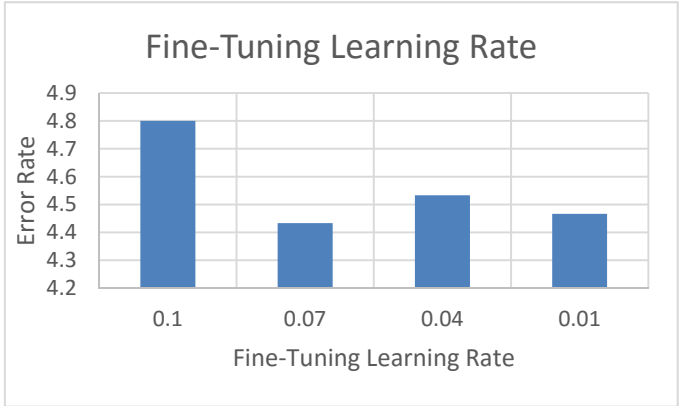

Figure 4.8

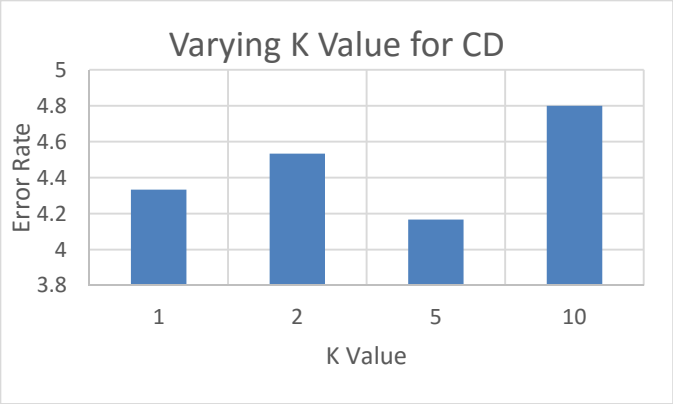

Figure 4.9

converging. Here we want to see how this trade-off translates to the learning process. While figure 4.9 does showcase a k-value of 5 performed best, there was no real discernable improvement that could be seen. Thus, the assumption made earlier that convergence will be achieved overtime seems to be true here.

5 Conclusion and Future Work

With the many tests done, there was only one true indicator of success: the training sample size. This is the case for both the SVM and DBN. When constraining our tests to 5000 samples, the best results always had a 4.0%-4.5% error rate. This is almost half that of the SVM. However, attempts to break the error rate to consistently give less than 4% was futile, as it seems the DBN always converges to this rate.

For the future we would like to implement the following:

- Addition of an energy-based overfitting detection mechanism into the pre-training process instead of manual inspection [5].
- Initialization of biases, specifically visible unit biases, as defined in [5].
- Initialization of weights from a Gaussian distribution rather than a uniform distribution [5].
- Addition of momentum-based gradient descent to speed up the training process [8].
- Addition of regularization methods [7].

We would also like to do further research in the following areas:

- Visualization methods for DBN's
- Configuration of DBN's core algorithms to better suit small datasets.
- A metric on which to analyze the effectiveness of pre-training.
- A metric on which to analyze the effectiveness of fine-tuning.
- Data shuffling methodology to ensure the data sample is representative of the set.
- Methodologies on handling small datasets.
- Comparison of hyper-parameters for small and large datasets.

In conclusion, DBN's seem to perform well when compared with discriminative methods such as SVM's. However, the DBN seems to converge to a certain range given sample size. The best result that we achieved with the full data-set was 1.37%. Our DBN using a tenth of the data was able to achieve 3.9% as its best result. We believe that more work can be done to understand the model better and achieve better results.

References

[1] Hinton, Geoffrey E. "To recognize shapes, first learn to generate images." *Progress in brain research* 165 (2007): 535-547.

[2] Ng, Andrew Y., and Michael I. Jordan. "On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes." *Advances in neural information processing systems* 2 (2002): 841-848.

[3] Hinton, G. E, Osindero, S., and Teh, Y. W. (2006). A fast learning algorithm for deep belief nets. Neural Computation, 18:1527-1554.

[4] Y. Bengio, X. Glorot, Understanding the difficulty of training deep feedforward neuralnetworks, AISTATS 2010

[5] Hinton, Geoffrey. "A practical guide to training restricted Boltzmann machines." *Momentum* 9.1 (2010): 926.

[6] Theis, Lucas, et al. "In all likelihood, deep belief is not enough." *Journal of Machine Learning Research* 12.Nov (2011): 3071-3096.

[7] Kim, Jae Won. "Classification with Deep Belief Networks."

[8] Sims, James Christopher. "An implementation of Deep Belief Networks using restricted Boltzmann machines in Clojure." (2016).