Depth Image Dimension Reduction Using Deep Belief Networks

Isma Hadji* and Akshay Jain**
Department of Electrical and Computer Engineering
University of Missouri
19 Eng. Building West, Columbia, MO, 65211
Email: *ih9p5@mail.missouri.edu, **akshay.jain@mail.missouri.edu

***Abstract*—Nowadays 3D images are becoming a widely used tool in many computer vision tasks, such as scene segmentation or object recognition. A common problem when using 3D images, is that often the size of the corresponding cloud of points is very big and therefore hard to process, making the subsequent algorithms computationally expensive. In this work, we address this problem by introducing a pre-processing step in order to reduce the number of points needed to represent an object in 3D by reducing the number of pixels in the depth image. In order to achieve this we use state-of-the-art techniques for data dimension reduction; Deep Belief Networks based on RBM greedy layer wise pre-training. In this work we demonstrate that the role of the structure of the network is not as important as the underlying methods used to learn the parameters of the network. Most importantly, we prove in this work the strategic role of the fine tuning and the RBM pre-training steps in several aspects.**

I. Introduction

Data dimension reduction is a very important topic in the field of machine learning. The choice of a subset of features can be very helpful both in terms of data handling as well as any subsequent task for which this data is to be used. Some applications for which reduced dimension data can be used include, but are not limited to, data clustering and classification. In fact, the features used in such algorithms can have a tremendous impact on their results. For example, using different features or dimension can highly affect the results of a classifierIn addition to that, reducing the dimensionality of the feature space can ultimately help to reduce the time complexity for various tasks. All these reasons make the problem of dimension reduction a very hot topic that is being thoroughly explored by the machine learning community.

Feature dimension reduction techniques can be classified into 3 distinct categories. Traditionally, the primary choices that have been used are algorithms that reduce connectivity between data. PCA and ICA are the most well known and widely used algorithms falling in this category. While the former attempts to correlate features, the later goes further by maximizing independence between them.

The second category of approaches for dimension reduction are graph based approaches. ISOMAPs, Local Linear Embedding (LLE) are such techniques. The common idea to most of the methods falling in this category is to reduce dimension while learning the manifold on which it falls and trying to keep the same shape in lower dimensional spaces.

Finally in the last category, we find probabilistic approaches. The most prominent work based on probabilistic approaches are based on the principle of Auto-encoders. The recent Deep Belief networks (DBN) are, undoubtedly, the most effective method for achieving this task. Although, many algorithms have been developed for training DBN, in this work we will focus on one of the most successful implementations [1] that uses Restricted Boltzman Machines (RBM) for initializing the parameters of each level in the Deep network. The targeted task in this case is the compression of depth images into smaller codes, that could be used later on for categorizing and recognizing the objects present in each image. Ultimately, the goal of this work, is to study the various aspects of DBN while encoding the input depth images in such a way as to keep the most information about what object they represent, hence, being able to recover the shape of the object from the smaller code.

II. Background and Related Work

Deep belief networks for data dimension reduction gained popularity with the introduction of a Greedy layer wise training step by Hinton in 2006 [1], that treats each layer as a Restricted Boltzman Machine. Ever since, many techniques for pre-training the layers of the Deep network have been proposed. These techniques can be divided into 2 main paradigms; stochastic Restricted Boltzman Machines and deterministic auto-encoders.

Dual layer RBM's provide a simple and efficient way to learn the features of data. However, it requires the visible units to be binary valued which is not the case in many real world applications. Therefore, a lot of work done on RBM's is focused towards approaches to use real valued inputs. [1] scales RBM's input vectors to values between [0,1]. These vectors are then used as probabilities to calculate the binary features. [2] presents an extension to the approach proposed in [1] to modify the energy function of the RBM and changing the range of input values. They modify the conditional density of the units of one layer given the other layer of RBM as a truncated exponential and a quadratic exponential term. They also provide a method to allow the hidden unit values to be non binary. [3]presents an approach called Noisy Rectified Linear Units (NReLu). They modify there previous work [4] to represent each hidden neuron with an infinite number of

binary hidden units with same weights but different biases. This is found to be better for recognizing objects because it learns the variation in intensity better than binary units.

The second approach for pre-training the layers of a deep network is to consider each one of them as a small auto-encoder. The most prominent methods falling in this category include Sparse Auto-Encoders [5], Denoising [6] and Contractive Auto-Encoders [7], to state a few. The basic idea behind this type of Auto-Encoders is to to learn an underlying network that is insensitive to changes in the input while ultimately reducing dimensionality. Sparse Auto-Encoders achieve this goal by introducing a sparsifying function to the front end of the decoder. The role of this function is to transform the code to a sparse vector by making the output of the activation function closer to zero. Denoising auto-encoder follow the same general framework but in this case the goal is to make the network insensitive to noisy data. Therefore, in this case, changes are applied to the input directly by adding noise to the input and modifying the learner so as to correct the effect of the corruption. The so trained layers are stacked and further trained in a similar manner to RBM based approaches [8]. Similarly, contractive Auto-Encoders (CAE) share the same idea of making the Deep Network less prone to changes in the input. However, CAEs do not alter the input but modify the objective function instead. The main difference in this case is adding a penalty whenever small changes in the input cause relevant changes in the learned features. Although, these are the main techniques used for building Deep networks, several other methods have been proposed in the related literature such as the more recent; saturating Auto-Encoders proposed in [9] and transforming Auto-Encoders in [10]. We refer the reader to the survey proposed by Bengio et al in [11] for a more exhaustive description.

In this work, we use the approach presented in [1] because of its simplicity to learn the features from high dimesnional input vectors, as well as its widely accepted effectiveness.

III. Deep Belief Networks

Deep Belief Networks are multi-layer Neural networks, characterized by the presence of many hidden layers. DBNs are usually viewed as a probabilistic way to encode the input data and model relationships between features. In that sense, DBNs can be used as Auto-Encoders in which the network is made of 2 main parts, an encoding part, that generates the reduced dimensions data – the code, and the decoding part, that reconstructs the data to its original dimensions. The main idea in using DBN for data dimension reduction is to minimize the reconstruction error between the 2 parts of the network. A usual choice for the reconstruction error is the mean squared error. However, because of the probabilistic character of DBN, cross-entropy as defined in eq1 is a very common metric as well. In eq 1 $x_k$is the actual input, while $z_k$is the reconstructed data at the output.

$$E(x,z) = \sum_{k=1}^{d} x_k log(z_k) + (1-x_k) log(1-z_k) \quad (1)$$

Having many layers, is a suitable characteristic in DBN for encoding data to lower dimensional feature space going from one layer to the next one. In fact, this is what makes DBN an attractive choice for data dimension reduction. However, the presence of many layers, implies a large amount of parameters to learn and the traditional back-propagation is not efficient in this case without a good initialization of the weights. For this reason, a pre-processing step that trains each layer separately is often involved. Restricted Boltzman Machines, is a simple and robust way used to pre-train the layers of a DBN. To find the initial weights each layer of the DBN is treated as a separate RBM. The initial high dimesnional input vector is used as visible layer while the hidden layer forms the features. Since, there is only one actual visible layer, the output of first RBM is used as the visible layer for the second layer and so on.

RBMs encode the energy between the two layers using their joint configuration as given by 2

$$E(v,h) = -\sum_{i=1}^{M} b_i v_i - \sum_{j=1}^{N} b_j h_j - \sum_{i,j} v_i h_j w_{ij} \quad (2)$$

where v_i ,h_j are the visible and hidden units respectively, $b_i$and c_j are the bias for the visible and hidden units respectively. w_{ij} is the weight connecting unit$i$ and j. M denotes the number of visible units, while N are the number of hidden units. 2 can be used to derive 3 and 4 which gives the probability of one layer given other, where $\sigma(x)$is equal to $1/(1+exp(-x))$.

$$p(h_i = 1|v) = \sigma(\sum_{j=1}^{N} w_{ij} v_j + c_j) \quad (3)$$

$$p(v_i = 1|h) = \sigma(\sum_{i=1}^{M} w_{ij} h_i + b_i) \quad (4)$$

For our application of reducing depth image dimension, the input units are gray scale pixels. To incorporate these real values to RBM's, each pixel is scaled between [0,1].For all the RBM layers except the top one. Next, 3 is used to generate the hidden units given the visible units. The hidden units are converted back to binary values based on the probability calculated by 3. Then, 4 is used to re-calculate the visible units given hidden ones this time. The hidden units are then updated again using the new visible units. It is worth noting, that for the top most RBM layer, a linear activation function is used, since we are looking for real gray scale image pixel values. The update of the weights are calculated by equation 5

$$\Delta w_{ij}^{t} = \varepsilon((VH')_{data} - (VH')_{recons}) + p\Delta w_{ij}^{t-1} \quad (5)$$

where V and H are the visible and hidden state vectors. This equation takes into account the difference in the transitions from visible state to the hidden state in the two updates. ε is called the learning rate. Essentially, a learning rate that is too big can make the lgorithm unstab;e, wile a learning rate that is too small increases the chances of falling in a local minima while slowing down the convergence process at the same time. The symbol p denotes the momentum [12]. It controls how much does the change in weight in the previous iteration affect the the weights in the current one. This acts as a regularization parameter that that penalizes situations that can cause over-fitting.

Once the RBM pre-training is done, the parameters learnt on the layer-wise basis, can now be used as initial parameters to train the whole network, using traditional back-propagation (BP) algorithm. Although the RBM pre-training plays an important role in initializing the network, BP remains a very important fine-tuning step. In fact, it puts together the contribution of each layer and interconnects the different layers, and since a good initialization is obtained from RBM, BP can now be applied without much loss of information especially in the last stages where the decay in the propagated error doesn't affect the results too much in this case.

Given that the core of the algorithm is reducing the error using gradient descent, the back-propagation stage can affect the results in different ways. In additon to the effect of the learning rate and momentum that are used in a similar manner to the RBM stage,the number of iterations plays a critical role in over or under fitting the data. Therefore, one should seek a good trade-off between the two when implememnting this part of the algorithm Ulitimately ths should decrese the reconstruction error while maintaining a good generalization capability for the DBN.

IV. Experimental Results and Discussion

As was highlighted in section III the Deep Belief Networks based on RBM layer wise pre-training depends on many parameters. For this reason, we are testing, in this section, the effect of each component on the overall data dimension reduction results. This is specifically illustrated for the application of depth image compression. In order to do this we are treating one component at a time in what we will call a coarse to fine testing scheme. We first start by testing the role of the structure of the network both in terms of depth and number of neurons per layer. Then, we focus on parameters governing the convergence of the objective functions. To perform these tests we use the auto encoder code available at [13]which is a modified version of the code provided by [1].

The dataset used, is a subset of the RGB-D dataset [14]containing 5 different objects seen from 3 different positions and 120 different views. Two subsets with 1235 depth images each are randomly created, where one is used for training and the other for testing. Since, the images contain objects of the same size, the original depth images have been resized to be all 50×50 images.

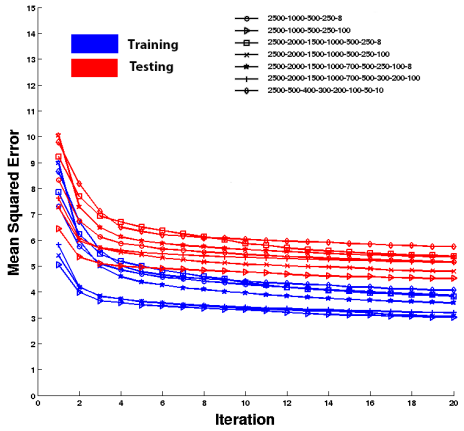

Figure 1. The reconstruction error after the fine tuning stage versus the number of BP iterations for different configurations of network structures

A. *Effect of structure of the DBN*

In this experiment, we address the effect of varying the structure of the DBN by changing the number of layers as well as the number of neurons in each layer. We test various layer configurations and evaluate the results based on the cross-entropy based, reconstruction error on the training as well as testing dataset. To this end, we test by changing the depth of the DBN while keeping the code size same and vice versa. We also test by varying the number of neurons for constant depth and code sizes. Figure 1 summarizes the results obtained. We observe that increasing the depth of the DBN decreases the performance for both training and testing data, though not by much. We also verify that increasing the code size results in better performance, since the images are compressed by a smaller ratio. However, it is worth noting that we get comparable errors for code size 8 and 100, provided the decrease in number of neurons in the initial stages of the network is not too big. Finally, we observe that a large drop in the number of neurons in successive layers produces high reconstruction error. This is more evident when the first hidden layer has a lot less number of neurons than the input layer, since the first few layers are the ones that count more on the initialization given that the backpropagation error tends to fade away in these layers and therefore the fine tunining doesn't help much. Putting all the results together, we note that all the tested network configuration have errors within a small interval, which suggests that minor variations in the structure of the DBN does not have a significant effect on the results. Following this, we choose the network configuration $2500 - 2000 - 1500 - 1000 - 500 - 250 - 100$ for rest of the experiments.

B. *Effect of the BP error function*

Next step in our coarse to fine testing scheme is addressing the effect of the reconstruction error type, used for fine tuning

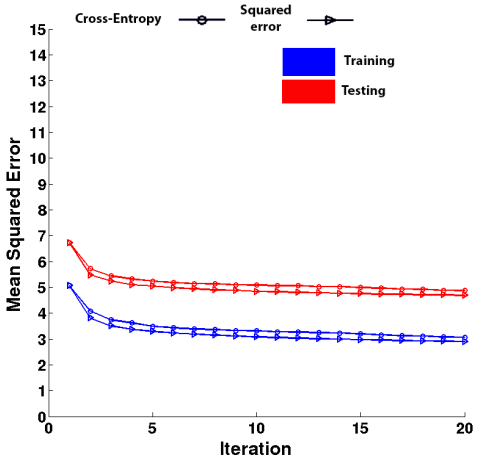

Figure 2. The overall back-propagation error versus the number of iterations using cross-entropy and mean squared error.

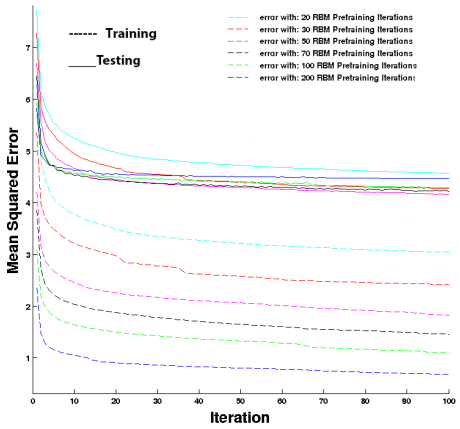

Figure 3. The reconstruction error after the fine tuning stage versus the number of BP iterations for different number of RBM iterations

the overall algorithm, on the behavior of the network. We consider modifying the objective function minimized by the Back-Propagation algorithm. As was introduced in III there are two main error types that could be minimized; the mean squared error and the cross-entropy. In this experiment we have used the best network configuration obtained from previous experiment and alternated between the two errors. Surprisingly, as can be seen from figure 2 , the mean squared error performance is better for both training and testing, although cross-entropy, being a stochastic approach, is usually more suited for this type of problems.

C. Overfitting/Underfitting due to pre training

In this experiment we test how much does the initial weights computed during the RBM pre-training effect the reconstruction error after fine tuning. In Figure 3 we report the final reconstruction error for both training and testing data for different sets of initial weights which are obtained by varying the number of RBM iterations from 20 to 200. It is observed that the RBM pre training stage is not able to find a good set of initial weights for iterations less than 50. However, the initial set of weights over-fit to the training data if the RBM runs for more than 70 iterations. Hence, this is a very important parameter which should be chosen according to the dataset. We choose 50 number of RBM iterations for all the further experiments.

D. Effect of the learning rate

A common problem to most Gradient Descent based approaches is the learning rate used to update the weights. The method adopted in this paper, relies on learning rate both for the RBM initialization as well as the back-propgataion fine tuning. In this experiment, we focus on the role of the learning rate for the initialization step. We vary this parameters from

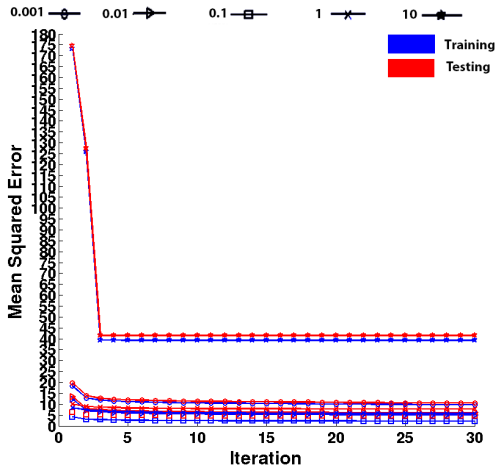

Figure 4. The overall back-propagation error versus the number of iterations using different learning rates

10^{-3} all the way to 10. Figure 4 confirms the role of the learning rate as was described in section III. In fact, for both, the high –10 and the low – 10^{-3} the algorithm fails and gets entrapped in local minima. Moreover, the high learning rate clearly is more harmful. The best learning rate appeared to be a mid value of 0.1, therefore confirming our initial intuition.

E. Importance of pre training and fine tuning stage

The pre training step using RBM's is an effective way to calculate the initial weights for DBN's. However, fine tuning step is also very important to compute the final parameters. To test this hypothesis we use the pre training weights to reduce the dimensionality of the images and then reconstruct them without going through the fine tuning stage. Figure 5 compares the reconstruction of images by using just the pre training weights and the ones obtained after fine tuning. It is evident that the pre training weights produce noisier reconstructions

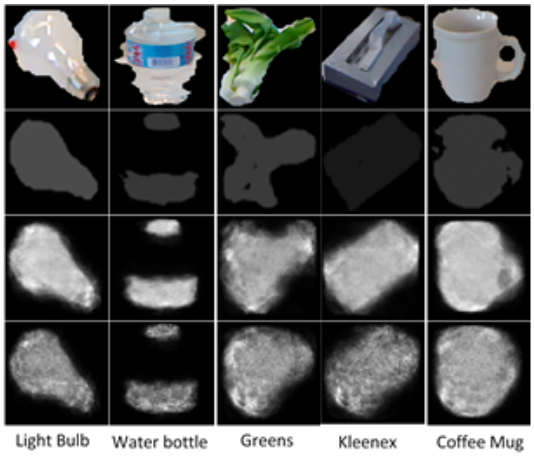

Figure 5. Reconstructed depth images. From top row to bottom, colored images of 5 objects, depth images of the objects, image reconstructed after the fine tuning stage, image reconstructed after the pre training stage

as compared to the fine tuning. Nevertheless, the shapes of the objects are clearly visible in the RBM pretraining weights case, proving there ability to provide good initializations for the network parameters.

V. Conclusion and Future Work

In this work, we compared different aspects of auto encoders and there pre trianing for reducing dimensionality of depth images. We conclude that the strcuture of the DBN, though important, does not affect the results significantly as long as a network with enough layers is selected. This favours the use of DBN for dimensionality reduction without requiring a very precise fine tuning of network structure. We also show that the number iterations used to obtain the pre training weights using RBM play an important role as it under-fits and over fits the model. Hence, this parameter should be carefully chosen for the application at hand. We also demonstrate that the choice of the error function for back propagation and learning rate of RBM also affects the DBN parameter calculation significantly and thus the reconstruction results. Selecting the correct learning rate is very important to avoid landing in local minimas. Finally, we perform a qualitative analysis to highlight the importance of both the RBM pre-trainig and fine tuning stages.

References

[1] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, Jul 2006.

[2] Y. Bengio, P. Lamblin, D. Popovici, H. Larochelle, U. D. MontrÃ©al, and M. QuÃ©bec, "Greedy layer-wise training of deep networks," in *In NIPS*. MIT Press, 2007.

[3] G. E. Hinton, "Rectified linear units improve restricted boltzmann machines vinod nair."

[4] Y. W. Teh and G. E. Hinton, "Rate-coded restricted boltzmann machines for face recognition," 2001.

[5] R. Marc, C. Poultney, S. Chopra, and Y. Lecun, "Efficient learning of sparse representations with an energy-based model," in *Advances in Neural Information Processing Systems (NIPS 2006*. MIT Press, 2006.

[6] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, "Extracting and composing robust features with denoising autoencoders," in *Proceedings of the 25th international conference on Machine learning*, ser. ICML '08, 2008, pp. 1096–1103.

[7] S. Rifai, P. Vincent, X. Muller, X. Glorot, and Y. Bengio, "Contractive auto-encoders: Explicit invariance during feature extraction," in *ICML*, 2011.

[8] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion," *J. Mach. Learn. Res.*, vol. 11, pp. 3371–3408, dec 2010.

[9] R. Goroshin and Y. LeCun, "Saturating auto-encoder," *CoRR*, vol. abs/1301.3577, 2013.

[10] G. Hinton, A. Krizhevsky, and S. Wang, "Transforming auto-encoders," in *Artificial Neural Networks and Machine Learning ICANN 2011*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2011, vol. 6791, pp. 44–51.

[11] Y. Bengio, A. C. Courville, and P. Vincent, "Unsupervised feature learning and deep learning: A review and new perspectives," *CoRR*, vol. abs/1206.5538, 2012.

[12] S. Haykin, *Neural Networks: A Comprehensive Foundation*, 2nd ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1998.

[13] S. Martin, "http://www.cs.otago.ac.nz/homepages/smartin/software.php."

[14] K. Lai, L. Bo, X. Ren, and D. Fox, "A large-scale hierarchical multi-view rgb-d object dataset."