Novel Lossy Compression Algorithms with Stacked Autoencoders

Anand Atreya and Daniel O'Shea

{aatreya, djoshea}@stanford.edu

11 December 2009

1. Introduction

1.1. Lossy compression

Lossy compression is a strategy to reduce the size of data while maintaining the majority of its useful or meaningful information. Though some data is lost in the compression process, lossy compression algorithms try to remove data that is unlikely to be considered important by a human consumer. Given this goal, today's lossy compression techniques are carefully designed to model human perception as closely as possible in order to maximize the amount of valuable retained information for a given data compression level. The popular JPEG image compression algorithm, for example, is reliant upon the observation that human visual perception is far more sensitive to low-frequency spatial variation than high-frequency spatial variation. Similarly, the MP3 audio compression algorithm uses carefully-tuned psychoacoustic models to make inferences about which components of a given audio stream are most perceived by a human listener. These perceptual coding methods exhibit quite impressive compression results, but require a great deal of work among psychologists, computer scientists, and engineers, and are ineffective or even detrimental when applied in situations for which they were not designed. Using JPEG to compress line art, for example, often yields undesirable results.

It is thus valuable to be able to easily develop a compression method that is naturally tuned to a particular type of data. This can be accomplished by extracting statistical regularities in the data and then using this information to compress the data. We demonstrate that one can efficiently compress data through the careful application of deep-belief networks while maintaining the data's most meaningful characteristics.

1.2. Deep learning

Deep-belief networks are a relatively new approach to neural networks that are unique in having more than one hidden layer (stacked autoencoders) [1]. These networks are typically trained one layer at a time, with a set of sparsity constraints to minimize the number of connections between nodes. Training networks in this way allows one to develop hierarchical representations while ensuring that individual layers represent meaningful components of the information. With images of human faces, for example, a deep-belief network may contain a first layer composed of various edges, a second layer that uses edges to form face parts, and a third layer that uses face parts to form entire face representations. The components generated at each layer are referred to as basis functions. Due to the sparsity constraints, these basis functions develop such that they maximize the amount of information that can be represented through a minimal combination of bases.

Since the training method for these networks is designed to encode data using very little information, we believe that the encoded representation is naturally suited for lossy compression applications.

2. Approach

2.1. Autoencoder Training

We used autoencoders, as implemented in the Stanford Deep Network library, to perform nonlinear dimensionality reduction on inputs from a given dataset, and to encode the reduced representation of each input into a compressed version of the original. Our approach combines lossy encoding via stacked autoencoders, lossy vector quantization via k-means, and lossless Huffman encoding.

The autoencoder is designed to represent each input patch as a vector of activation values at each node. From these activation values, the decoding stage forms a reconstructed version of the original input patch. Initially, these reconstructed values are used to train the autoencoder to minimize the error between input and the reconstructed output, while a parameter is used to ensure sparsity of the activation vector. When images are to be compressed, the algorithm feeds the input image into the nonlinear encoding layer and outputs a representation of the activation vector. This activation vector is then further compressed. The decom-

pression algorithm feeds the decompressed activation values into the decoding stage of the autoencoder and returns the reconstruction. Additional autoencoders may be nested recursively within the encoding and decoding layers of an outer autoencoder to apply this methodology to the stacked autoencoder case. We analyze the results of these scenarios below.

In order to generate reconstructions with adequate fidelity, we explored appropriate parameter values for the network architecture and training algorithm, such as number of nodes, target sparsity, learning rate, etc. We focused our attention primarily on the target sparsity parameter, exploring a range of 10 logarithmically spaced values from 0.005 to 0.5 during the training procedure. For the two-autoencoder case, we performed a grid search over the same range of sparsity values for both the outer and inner autoencoder.

2.2. Vector Quantization via k-means

The next task is to efficiently encode the activation values on each node into a stream of bits that will comprise the compressed input. One approach to this problem is simply to serialize each activation value directly as a floating point number. However, this is very data inefficient. Thus, we have chosen to use k-means clustering to find an ideal set of quantization values for the real-valued activations.

Our quantization algorithm for encoding the activation vector works as follows. Once the autoencoder has been trained using a specific sparsity parameter, we load the trained network and sample the activation vectors for N input images. We then run k-means clustering on each node independently, assigning each of the N values into k clusters. We then encode each element of an input image's activation vector using the codebook (i.e. cluster means) computed for each node. This imposes an upper bound of 2^k bits per node. The size of the compressed representation k can be tuned as a quality or bitrate parameter similar to parameters tunable in mainstream lossy compression algorithms. On each trained autoencoder, we performed vector quantization using logarithmically spaced k values from 2 through 256. For $k = 256$, distortion of the activation vectors due to lossy quantization is minimal. For $k = 2$, the amount of distortion and the degradation of reconstruction quality incurred depends on the sparsity value used to train the network.

2.3. Huffman Encoding

While vector quantization attempts to find the quantization codebook to minimize squared error distortion, cluster assignments need not be uniform, especially when activation vectors are sparse. We therefore compute the entropy of the cluster assignment histogram, which yields the number of bits needed on average to encode the quantized activation vector for images drawn from the training set.

3. Results

3.1. Role of Sparsity and k Parameters

For high sparsity values, the autoencoder lightly compresses the inputs. In this regime, reconstruction error is sensitive to the k parameter used in vector quantization. Lower k values significantly distort the activation vectors.

For low sparsity values, the autoencoder heavily compresses the inputs. The reconstruction error is insensitive to k. Activation vectors may be compressed down to binary (k=2) vectors with no substantial loss in quality. These binary vectors have low entropy, allowing large compression ratios.

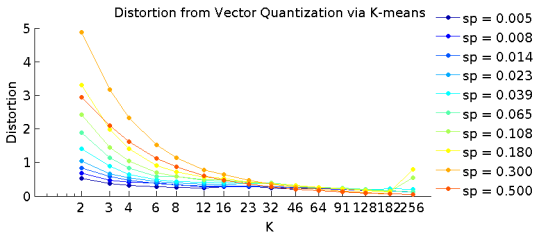


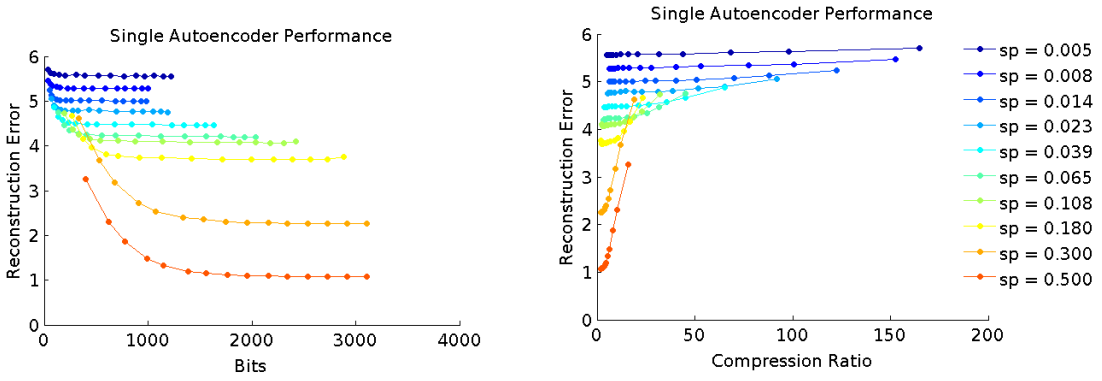
Figure 2: Amount of quantization distortion (L2 norm between raw and quantized activation values) induced by k-means vector quantization.

3.2. Two Autoencoder Compression

We examined the benefit on effective compression of the use of a nested set of two autoencoders. For each autoencoder, we independently varied the sparsity parameter as described above. We found that only the case where the outer autoencoder has a sparsity of 0.5 actually contributed to the "frontier" of achievable compression, shown in the final curves in Figure 3.

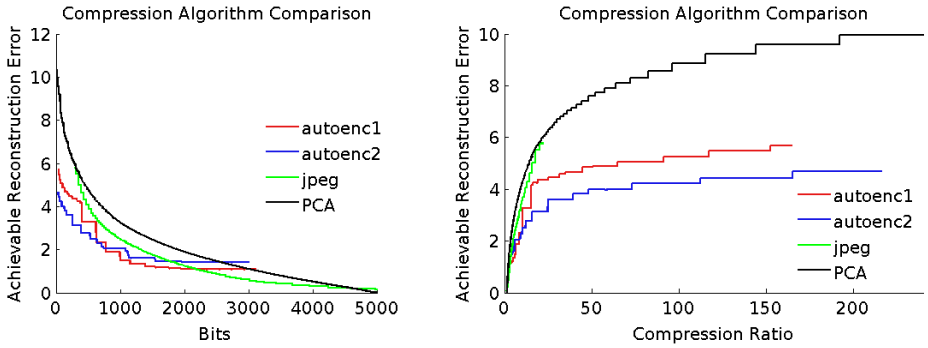
3.3. Comparison with JPEG and PCA

We compared our [autoencoder + vector quantization + Huffman encoding] scheme with two other common algorithms: JPEG and PCA. We found that the autoencoder compression algorithm outperforms JPEG in the high compression, reduced quality regime. This likely reflects the autoencoder's ability to learn the statistics of the class of input images.



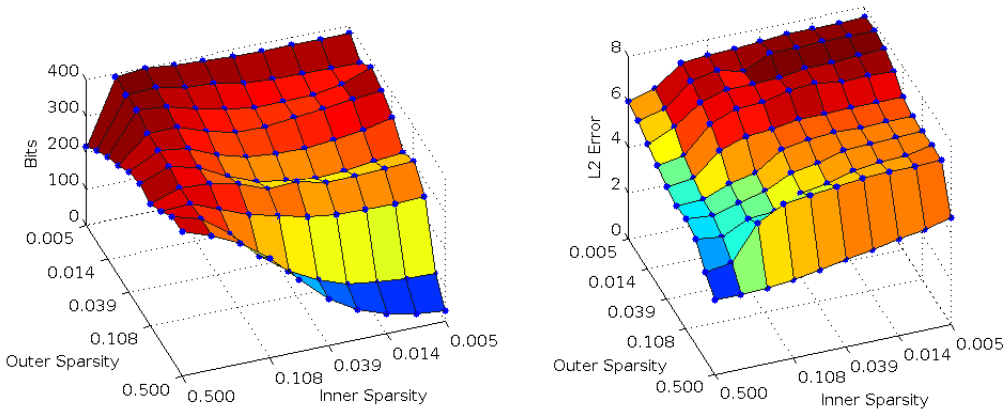
(a) Bits (b) Compression Ratio

Figure 1: Reconstruction error and size of compressed representation as (a) bit count and (b) compression ratio. Each color represents a autoencoder trained with a particular sparsity parameter. Each point within each curve represents a particular k value used in vector quantization.



(a) Bits (b) Compression Ratio

Figure 3: Comparison of compression curves for a single autoencoder, two stacked autoencoders, JPEG, and PCA. Plotted is the minimum attainable reconstruction error (L2 norm) given the constraint of (a) maximum average bits used, or (b) minimum compression ratio attained.



(a) Minimum Bits at $k = 2$ (b) Reconstruction Error at $k = 2$

Figure 4: Results of grid search over outer and inner autoencoder sparsity parameters for two autoencoder compression. (a) Bits used in compressed representation using $k = 2$ for vector quantization. (b) Reconstruction error attained using $k = 2$ for vector quantization.

For PCA, the optimal linear lossy encoding, we found that the autoencoder algorithm consistently outperforms at each reconstruction quality it achieves. This likely reflects the nonlinearity present in the encoding/decoding layers.

4. Discussion

We have presented an alternative approach to image compression that exploits the statistics of the data to achieve better compression than standard algorithms like JPEG and PCA. This improvement in performance, however, comes at the cost of loss of generality. Specifically, in order to benefit from the improvement in compression ratio for a given reproduction quality, our algorithm requires that one first train the algorithm offline on a large collection of images from the same domain, and then later apply that algorithm to compress images from the same domain. While this approach can be expensive, it is likely preferable in cases where one has a large amount of similar data to compress and would benefit from a better compression ratio at the cost of increased compression time.

In this work, we focused exclusively on the MNIST dataset. Although we tested the algorithm on examples that were not seen during training (with identical performance, so data is not shown), we have not yet explored the results of attempting to compress images drawn from a different domain than that on which the network was trained. We expect that performance would suffer significantly in this case, since the primary reason that our algorithm performs well–that it is able to exploit statistics of the domain data–no longer applies.

4.1. Domain Generality

Using the ngvideo datasets and pools of image patches drawn from the Caltech 101 dataset, we can explore the feasibility of training a domain-general image compression algorithm. We already have shown that the weight patches trained on the ngvideo dataset extract general image features, such as oriented edges, that are likely features present in large classes of image domains. Using general datasets such as these, we can compare our trained compression algorithm to other common lossy compression algorithms. It is as yet unclear whether we can hope to outperform existing lossy compression algorithms like JPEG in the sense of achieving better reconstruction quality for a given bitrate for images drawn from any domain.

Nevertheless, the advantage of using a machine learning approach is that we can easily tailor our compression algorithm to perform optimally on a specific training dataset, as we have done on MNIST. Using a dataset extracted from images from a certain domain, e.g. face images, handwritten characters, natural landscapes, etc., we have been able to train a compression algorithm to exploit statistical regularities in images characteristic of that particular domain. Because algorithms like JPEG are handtuned for compressing images in a general sense, we have shown that we can outperform JPEG for a specific domain of images, since those images possess strong statistical regularities that JPEG is unable to exploit.

4.2. Integrating human "error" models

The compression method described above relies upon the fact that autoencoders are trained to minimize the sum of squared error between the original data and its reconstruction. However, this is not necessarily the same error function that a human observer would use to evaluate the quality of a reconstruction. While JPEG, MP3, etc. were designed to maximize the amount of useful information to a human observer, this method takes a more indirect approach in trying to make the reconstructed output match the input in as close a mathematical sense as possible. Since the mathematical error function is not necessarily the same as a human's, we would like to be able to incorporate what humans value in the reconstruction.

The most direct way to accomplish this would be to simply change the error function used by the autoencoder. This would require changing the gradient descent portion of the training algorithms. For example, the underlying principle of JPEG image compression is that humans value low frequency information in images but tend not to notice alterations to high frequency content. We could model this by computing a least squared error function on blurred versions the input and reconstruction. However, changing the error function would require us to ensure that the new optimization problem remains convex or at least tractable, and may require fine-tuning of network architecture and training parameters. While this approach may prove successful, it would be advantageous to integrate human quality assessment into the compression system in a less tedious way.

Another simple approach is to manipulate the inputs to the network directly so that the autoencoder would attempt to reproduce the manipulated versions of the original input. For example, we could low-pass filter all input images before feeding them to the autoencoder training algorithm.

Alternatively, we could add noise to higher frequencies in the input images and ask the autoencoder to reproduce the original images, without the added noise.

Such a network is referred to as a denoising autoencoder. Both of these strategies share the common theme of asking the network to learn to extract features that lie predominantly in the range of frequencies that humans value most in the reconstruction while ignoring features that depend on high frequency content. The reasoning behind this is that once the network uses these human-valued features to reconstruct the input, the compressed representation of node activation values will selectively encode information that humans value most, improving the quality of the reconstruction from a human perspective.

5. References

1. G. E. Hinton and R. R. Salakhutdinov. Reducing the Dimensionality of Data with Neural Networks. (28 July 2006). Science 313 (5786), 504.

2. Ian J. Goodfellow, Quoc V. Le, Andrew M. Saxe, Honglak Lee and Andrew Y. Ng. Measuring invariances in deep networks. NIPS 2009.