

# Properties of Autoencoders

Justin Johnson  
jcjohns@stanford.edu

Bharath Ramsundar  
rbharath@stanford.edu

## 1 Introduction

In recent years a variety of deep learning algorithms (including deep belief networks [4, 9] and deep neural networks, [5, 6]) have risen to prominence. The types of networks that are typically used in these applications are complicated, and studying their theoretical properties is difficult. In this project, we take first steps towards greater theoretical understanding of deep learning by surveying the prior literature and performing empirical studies.

We focus our work on a simple building-block of complicated deep networks: the sparse autoencoder. Past approaches have utilized features learned from unsupervised pretraining of deep neural networks in order to improve performance on classification tasks [6]. The sparse autoencoder is a fundamental building block of this unsupervised pretraining process. A single-layer autoencoder is a much simpler object than an entire deep network, but even this relatively simple object has not been well-studied theoretically. In this project we aim to explore different varieties of autoencoders and understand why they work.

## 2 Sparse Autoencoder

A sparse autoencoder is a neural network with a single hidden layer. Autoencoders attempt to learn nontrivial representations of the identity function on the training examples. These representations are consequently used as input features for classifiers.

Autoencoders are parameterized by choice of activation function  $f$ , choice of sparsity penalty  $\phi$ , and choice of regularization  $\psi$ . Neural net weights  $W, b$  are learned by minimizing an objective function  $J$  dependent on these choices:

$$J(W, b) = \frac{1}{m} \sum_{i=1}^m \ell(h_{W, b}(x^{(i)}), x^{(i)}) + \lambda \psi(W, b) + \beta \sum_{j=1}^p \phi(\hat{\rho}_j)$$

The first term is the  $\ell^2$  reconstruction error of the training data. The second term is  $\psi$ -regularization of the weight vectors, and the third term enforces a  $\phi$ -sparsity constraint on the mean activation of each hidden layer neuron over the training set. Further details are given in Appendix A.

## 3 Theoretical Underpinnings

In this section we survey certain theoretical properties of neural nets and autoencoders. We start with a general property of neural nets, the universal approximation result [3], which proves that arbitrary continuous functions may be uniformly approximated by single-layer nets. We continue to discuss a NP-Hardness result [1], which establishes the complexity of training even a single-layer net. We end by discussing a surprising result which shows that, in the absence of sparsity and regularization, autoencoders are simply a form of PCA [2].

### 3.1 Universal Approximation

Neural networks with one hidden layer are capable of uniformly approximating arbitrary continuous functions [3]. More precisely, finite linear combinations of the form

$$\sum_{j=1}^N \alpha_j \sigma(w_j^T x + b_j),$$

(where  $\sigma$  is a “sigmoidal” function, *i.e.*,  $\lim_{x \rightarrow \infty} \sigma(x) = 1$  and  $\lim_{x \rightarrow -\infty} \sigma(x) = 0$ ) can uniformly approximate any continuous function on  $\mathbb{R}^n$ . This result follows from basic functional analysis and from injectivity of the Fourier transform [11]. Further arguments based on the Stone-Weierstrass theorem or Wiener’s Tauberian theorem allow the result to be extended to more general classes of activation functions such as  $\sigma = \sin(nt)$  or  $\sigma = \exp(nt)$  [11].

Note that these approximation results don’t apply directly to autoencoders; autoencoders are distinct from general neural networks, since they attempt only to learn the identity. However, the universal approximation result motivates the belief that autoencoders can learn highly nontrivial, data-dependent representations of the identity.

### 3.2 NP-Hardness

Training a neural network classifier with  $n$ -dimensional inputs and only 3 hidden units is NP-hard when  $\sigma$  is a Heaviside step function [1]. The proof shows that there exist some sets of training data  $X$  such that learning a correct classifier on the data is combinatorially intractable. This result emphasizes the fact that training of a general neural network may be infeasible. However, the hardness result holds only for a specific class of network, and as the

next section will show, certain types of neural networks may be efficiently learned.

### 3.3 Equivalence with PCA

An autoencoder that does not include regularization  $\psi$  or sparsity  $\phi$  learns a feature representation that is closely related to the principal components of the training data [2]. In the absence of  $\phi$  or  $\psi$ , the cost vector is simply the  $\ell^2$  difference between the training data and autoencoder output. Using the fact that singular value decomposition provides the optimal rank  $p$  approximation to a  $n$  dimensional vector, we may then directly solve for the weight and bias vectors.

The interesting consequence is that the nonlinearity of the autoencoder, dependent on choice of activation  $f$ , is meaningless without appropriate choice of regularizer or sparsity. This result motivates our in-depth empirical study of the relations between the activation and sparsity functions in the next section.

## 4 Experiments

### 4.1 Equivalence with PCA

We first studied the empirical relationship between PCA and autoencoders with various activation functions. In order to remain consistent with the analysis in [2] we set  $\lambda = \beta = 0$  to omit the regularization and sparsity penalty terms. For each choice of activation function we trained an autoencoder using 2000 images from the MNIST [7] training set.

Once an autoencoder has been trained, we can treat the activations of the hidden units on any input as a feature transform of that input. In the language of Appendix A, this feature transform is  $a^{(2)}(x) = f(W^{(1)}x + b^{(1)})$ . We can visualize this feature transform by viewing each row of  $W^{(1)}$  as a linear filter against which the input  $x$  is matched.

For each activation function considered, the feature transform of the corresponding trained autoencoder was used to train a softmax classifier to recognize digits. Each classifier was evaluated on both the training set and a test set consisting of 2000 additional MNIST images.

We also computed the first 200 principal components of the training data. Projecting any input  $x$  onto the first 200 principal components of the training data gives a feature transform of the input  $x$ . We used this feature transform to train a softmax classifier.

Table 1 shows a subset of the features learned by each autoencoder, a subset of the first 200 principal components of the data, and the performance of each trained softmax classifier on the training and testing sets. Qualitatively, some of the features learned by the autoencoders look similar to the principal components of the training data. The sigmoid, identity, and sinusoidal activation functions give rise to feature transforms that per-

form very similarly to the PCA feature transform. The feature transform learned from the arctangent activation function appears to be underfitting the training data; this could indicate that the underlying autoencoder was not fully trained.

### 4.2 Generalized Sparse Autoencoders

The theoretical results of [2] and the empirical results of the previous section suggest that autoencoders cannot learn novel feature transforms unless the optimization objective is modified. To this end, we trained autoencoders using a variety of activation functions and sparsity constraint functions. We use the regularizer  $\psi(W, b) = \frac{1}{2}\|W^{(1)}\|_F + \frac{1}{2}\|W^{(2)}\|_F$  where  $\|A\|_F$  is the Frobenius norm.

As in the previous section, the feature transform of each trained autoencoder is used to train a softmax classifier. The results of these experiments are found in Table 2. In all cases we take  $\rho = 0.1$ .

The autoencoders trained with the empty sparsity penalty  $\phi(\hat{\rho}_j) = 0$  are essentially the same as the autoencoders from the previous section with the addition of regularization on  $W$ . The learned features are similar, but the softmax classifier enjoys lower generalization error.

The  $\ell^1$  sparsity penalty  $\phi(\hat{\rho}_j) = |\hat{\rho}_j - \rho|$  gives rise to classifiers that perform worse than a softmax classifier trained only on the pixel values of the training set. We did not experiment extensively with different values of the parameters  $\lambda, \beta$ , and  $\rho$ ; it is possible that a better choice of parameters would result in better classifiers in these cases.

For all but the identity activation function, the  $\ell^2$  sparsity penalty  $\phi(\hat{\rho}_j) = (\hat{\rho}_j - \rho)^2$  and the KL sparsity penalty give rise to classifiers that significantly outperform all other classifiers considered.

## 5 Discussion

Our experimentation so far has led us to focus on the sparsity constraint of the autoencoder. Our results demonstrate that either sigmoidal or trigonometric activation functions are necessary to achieve performant behavior. Moreover, KL and  $\ell^2$  sparsity seem to perform better than  $\ell^1$  or nonexistent sparsity. Future work might consider more rigorous mathematical justification for guiding the choice of sparsity function.

We also noticed that training using the back-propagation algorithm [12] was quite slow compared to the PCA algorithm. The difference arises from the fact that PCA exploits highly-tuned linear-algebraic procedures rather than the gradient descent of back-propagation. However, as we have previously seen, autoencoders (in the absence of sparsity and regularization) reduce to PCA. This result motivates a search for a suitable ‘‘sparse encoder PCA’’ algorithm which retains the

efficiency of PCA while inheriting the flexibility of the sparse autoencoder. Some prior work on sparse coding [8] and sparse low rank PCA [13] may provide directions for this investigation.

## A Generalized Sparse Autoencoders

In this section we describe a general framework for sparse autoencoders. This is a generalization of the sparse autoencoder presented in [10]. An autoencoder is a neural network with a single hidden layer that attempts to learn the identity function  $\mathbb{R}^n \rightarrow \mathbb{R}^n$ . Let the hidden layer have  $p$  units, and let  $f : \mathbb{R} \rightarrow \mathbb{R}$  be a differentiable activation function. The neural network is parameterized by terms weights  $W^{(1)} \in \mathbb{R}^{p \times n}$  and  $W^{(2)} \in \mathbb{R}^{n \times p}$  and bias terms  $b^{(1)} \in \mathbb{R}^p$  and  $b^{(2)} \in \mathbb{R}^n$ . Given weight and bias terms, the prediction on an input  $x \in \mathbb{R}^n$  is

$$h_{W,b} = f(W^{(2)} f(W^{(1)} x + b^{(1)}) + b^{(2)})$$

Given training examples  $x^{(1)}, \dots, x^{(m)} \in \mathbb{R}^n$  the objective function for the generalized sparse autoencoder is

$$J(W, b) = \frac{1}{m} \sum_{i=1}^m \ell(h_{W,b}(x^{(i)}), x^{(i)}) + \lambda \psi(W, b) + \beta \sum_{j=1}^p \phi(\hat{\rho}_j)$$

where we define

$$\hat{\rho}_j = \frac{1}{m} \sum_{i=1}^m f\left((W_j^{(1)})^T x^{(i)} + b_j^{(1)}\right)$$

to be the average activation of the  $j$ th hidden unit over the training set; here  $(W_j^{(1)})^T$  is the  $j$ th row of the matrix  $W^{(1)}$ . In the sparse autoencoder objective function,  $\ell : \mathbb{R}^n \times \mathbb{R}^n \rightarrow \mathbb{R}$  is a loss function that encourages the autoencoder to have low reconstruction error. The function  $\psi$  is a regularizer for the weight and bias terms and the parameter  $\lambda \in \mathbb{R}$  controls the relative importance of the regularization. Frequently we define  $\psi(W, b) = \frac{1}{2} \|W^{(1)}\|_F^2 + \frac{1}{2} \|W^{(2)}\|_F^2$  where  $\|A\|_F$  is the Frobenius norm. The function  $\phi : \mathbb{R} \rightarrow \mathbb{R}$  is the sparsity penalty, and the parameter  $\beta \in \mathbb{R}$  controls the relative importance of the sparsity penalty.

We train a generalized sparse autoencoder using gradient descent. For the remainder of the discussion we assume that  $\ell(x, y) = \frac{1}{2} \|x - y\|^2$ ; this allows us to use the standard backpropagation algorithm to compute the gradient of the reconstruction error term. For notational convenience, for  $x \in \mathbb{R}^n$  define

$$\begin{aligned} z^{(2)}(x) &= W^{(1)} x + b^{(1)} & a^{(2)}(x) &= f(z^{(2)}(x)) \\ z^{(3)}(x) &= W^{(2)} a^{(2)}(x) + b^{(2)} & a^{(3)}(x) &= f(z^{(3)}(x)) \end{aligned}$$

In the standard backpropagation algorithm we define

$$\begin{aligned} \delta^{(3)}(x) &= -(x - a^{(3)}(x)) \odot f'(z^{(3)}(x)) \\ \delta^{(2)}(x) &= \left( (W^{(2)})^T \delta^{(3)}(x) \right) \odot f'(z^{(2)}(x)) \end{aligned}$$

where  $\odot$  is componentwise multiplication of vectors and  $f'$  is applied componentwise. Then the gradients of the reconstruction terms are given by

$$\begin{aligned} \nabla_{W^{(k)}} \ell(h_{W,b}(x), x) &= \delta^{(k+1)}(x) a^{(k)}(x)^T \\ \nabla_{b^{(k)}} \ell(h_{W,b}(x), x) &= \delta^{(k+1)}(x) \end{aligned}$$

where  $k \in \{1, 2\}$  and we define  $a^{(1)}(x) = x$ .

Computing the gradient of  $\psi(W, b)$  is typically straightforward; for example when

$$\psi(W, b) = \frac{1}{2} \|W^{(1)}\|_F^2 + \frac{1}{2} \|W^{(2)}\|_F^2$$

then  $\nabla_{W^{(k)}} \psi(W, b) = W^{(k)}$  and  $\nabla_{b^{(k)}} \psi(W, b) = 0$  for  $k \in \{1, 2\}$ .

It remains to compute the gradient for the sparsity term. Clearly  $\nabla \phi(\hat{\rho}_j) = \phi'(\hat{\rho}_j) \nabla \hat{\rho}_j$  and  $\nabla_{b^{(2)}} \hat{\rho}_j = 0$  and  $\nabla_{W^{(2)}} \hat{\rho}_j = 0$  so we only need to compute  $\nabla_{b^{(1)}} \hat{\rho}_j$  and  $\nabla_{W^{(1)}} \hat{\rho}_j$ . A bit of arithmetic shows that

$$\begin{aligned} \frac{\partial \hat{\rho}_j}{\partial b_k^{(1)}} &= \begin{cases} \frac{1}{m} \sum_{i=1}^m f' \left( (W_j^{(1)})^T x^{(i)} + b_j^{(1)} \right) & j = k \\ 0 & \text{otherwise} \end{cases} \\ \nabla_{W_k^{(1)}} \hat{\rho}_j &= \begin{cases} \frac{1}{m} \sum_{i=1}^m f' \left( (W_j^{(1)})^T x^{(i)} + b_j^{(1)} \right) x^{(i)} & j = k \\ 0 & \text{otherwise} \end{cases} \end{aligned}$$

A bit more arithmetic shows that if we modify the standard backpropagation algorithm and by setting

$$\delta_i^{(2)}(x) = \left( \sum_{j=1}^p W_{ji}^{(2)} \delta_j^{(3)}(x) + \beta \phi'(\hat{\rho}_i) \right) f'(z_i^{(2)}(x))$$

the the gradient of the sparse autoencoder objective function is simply

$$\begin{aligned} \nabla_{b^{(k)}} J(W, b) &= \frac{1}{m} \sum_{i=1}^m \delta^{(k+1)}(x^{(i)}) + \lambda \nabla_{b^{(k)}} \psi(W, b) \\ \nabla_{W^{(k)}} J(W, b) &= \frac{1}{m} \sum_{i=1}^m \delta^{(k+1)}(x^{(i)}) a^{(k)}(x^{(i)})^T + \lambda \nabla_{W^{(k)}} \psi(W, b) \end{aligned}$$

The objective function can then be minimized using a gradient-descent method such as [14].

## References

- [1] Avrim L Blum and Ronald L Rivest. Training a 3-node neural network is NP-complete. *Neural Networks*, 5(1):117–127, 1992.
- [2] Hervé Bourlard and Yves Kamp. Auto-association by multilayer perceptrons and singular value decomposition. *Biological cybernetics*, 59(4):291–294, 1988.
- [3] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems (MCSS)*, 2(4):303–314, 1989.

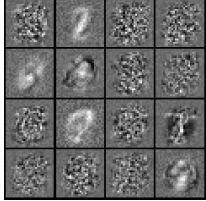
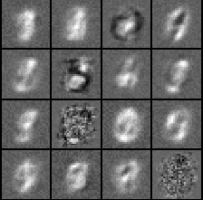
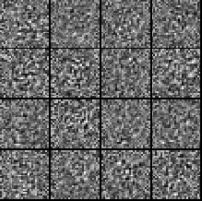
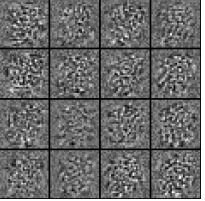
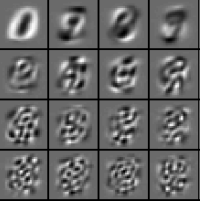
| Activation function $f(z)$ | $1/(1 + e^{-z})$  | $1/2 + \tan^{-1}(z)/\pi$  | $z$  | $1/2 + \sin(z)/2$   | PCA   |
|----------------------------|---|---|--|---|---|
| Learned features           |  |  |  |  |  |
| Training accuracy          | 99.95%  | 90.50%  | 99.95%   | 99.95%  | 100.00%   |
| Testing accuracy           | 82.00%  | 80.05%  | 82.00%   | 79.60%  | 82.30%  |

Table 1: Single layer autoencoders with no regularization or sparsity penalty were trained on 2000 MNIST images and the extracted feature transform was used to train a softmax classifier. In all cases we learned a 200-dimensional feature transform; a small subset of these features are shown. We compare with a softmax classifier trained on PCA features extracted from the same training data. In all cases the trained feature transform and classifier are tested on a different set of 2000 MNIST images. For reference, a softmax classifier trained directly from the pixel values of the training data achieves a training accuracy of 100.00% and a testing accuracy of 82.90%.

- [4] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.
- [5] A. Krizhevsky, I. Sutskever, and G. Hinton. ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*, pages 1106–1114, 2012.
- [6] Quoc V Le, Rajat Monga, Matthieu Devin, Greg Corrado, Kai Chen, Marc’Aurelio Ranzato, Jeff Dean, and Andrew Y Ng. Building high-level features using large scale unsupervised learning. *arXiv preprint arXiv:1112.6209*, 2011.
- [7] Yann LeCun and Corinna Cortes. The MNIST database of handwritten digits, 1998.
- [8] Honglak Lee, Alexis Battle, Rajat Raina, and Andrew Y Ng. Efficient sparse coding algorithms. *Advances in neural information processing systems*, 19:801, 2007.
- [9] Honglak Lee, Roger Grosse, Rajesh Ranganath, and Andrew Y Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 609–616. ACM, 2009.
- [10] Andrew Ng, Jiquan Ngiam, Chuan Yu Foo, Yifan Mai, and Caroline Suen. UFLDL tutorial. [http://ufldl.stanford.edu/wiki/index.php/UFLDL\\_Tutorial](http://ufldl.stanford.edu/wiki/index.php/UFLDL_Tutorial).
- [11] Walter Rudin. Functional analysis. international series in pure and applied mathematics, 1991.
- [12] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Cognitive modeling*, 1:213, 2002.
- [13] Haipeng Shen and Jianhua Z Huang. Sparse principal component analysis via regularized low rank matrix approximation. *Journal of multivariate analysis*, 99(6):1015–1034, 2008.
- [14] Ciyou Zhu, Richard H Byrd, Peihuang Lu, and Jorge Nocedal. Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound-constrained optimization. *ACM Transactions on Mathematical Software (TOMS)*, 23(4):550–560, 1997.

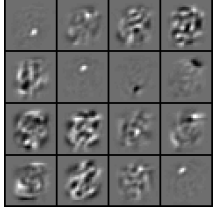
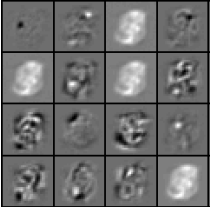
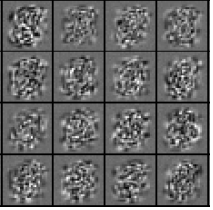
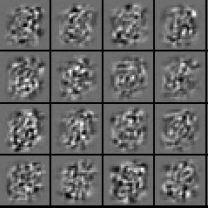
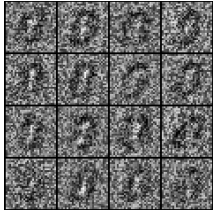
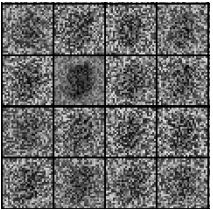
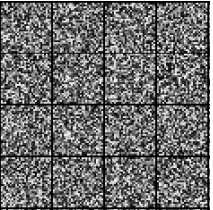
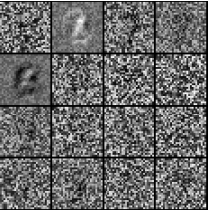
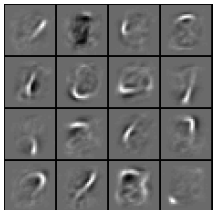
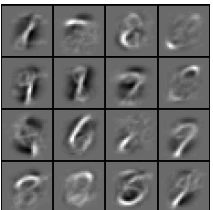
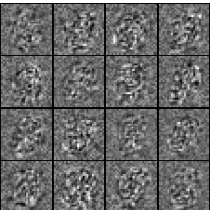
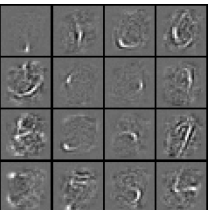
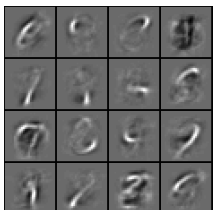
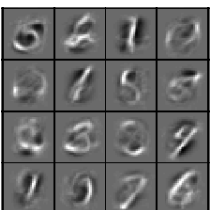
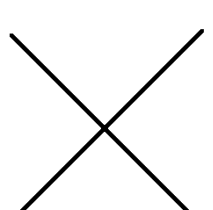
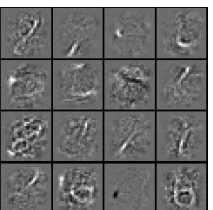
|                                       |                                   | Activation function $f(z)$  |   |  |   |
|---------------------------------------|-----------------------------------|---|---|--|---|
|                                       |                                   | $1/(1 + e^{-z})$  | $1/2 + \tan^{-1}(z)/\pi$  | $z$  | $1/2 + \sin(z)/2$   |
| Sparsity penalty $\phi(\hat{\rho}_j)$ | 0                                 | <br>98.75%<br>84.90%   | <br>98.90%<br>85.20%   | <br>100.00%<br>82.85%  | <br>99.90%<br>83.00%   |
|                                       | $ \hat{\rho}_j - \rho $           | <br>88.50%<br>80.45%   | <br>75.55%<br>67.45%   | <br>99.60%<br>79.85%   | <br>96.30%<br>79.90%   |
|                                       | $(\hat{\rho}_j - \rho)^2$         | <br>99.85%<br>91.30%  | <br>99.75%<br>90.45%  | <br>100.00%<br>83.30% | <br>99.85%<br>88.45%  |
|                                       | $KL(\rho \parallel \hat{\rho}_j)$ | <br>99.75%<br>90.30% | <br>99.50%<br>90.00% |                      | <br>99.70%<br>88.90% |

Table 2: We trained single layer autoencoders with  $\ell^2$  regularization and a variety of activation functions and sparsity constraints. In all cases we learned a 200-dimensional feature transform; for each case we show a small representative subset of these features. Each autoencoder was trained using 2000 MNIST images and in each case the extracted feature transform was used to train a softmax classifier. The final classifier was then tested on a different set of 2000 MNIST images. The accuracy on the training and test sets respectively are shown beneath each visualized feature transform. Note that the KL sparsity penalty is only defined when  $\hat{\rho}_j \in (0, 1)$ ; this is not true in general for  $f(z) = z$ .