Using Deep Autoencoders to Learn From Undersampled Data

Elias Rubin

A Junior Paper
presented to the faculty
of Princeton University
in candidacy for the degree
of Bachelor of the Arts

Recommended for acceptance
by the Department of
Astrophysical Sciences

Advisers: Robert Lupton & Peter Melchior

May 3, 2016

I hereby declare that I am the sole author of this thesis.

I authorize Princeton University to lend this thesis to other institutions or individuals for the purpose of scholarly research.

Elias Rubin

I further authorize Princeton University to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

Elias Rubin

ABSTRACT

We investigate the prospect of using deep autoencoders – a type of artificial neural network – to reconstruct undersampled spectrograph data. Our deep rectifying networks are able to learn a sparse, disentangled code from synthetic undersampled data, and create faithful reconstructions of novel inputs. We also discuss some considerations for the design and training of deep networks. Code and additional visualizations are available at `https://github.com/efrubin/SpringJP-Autoencoders`.

1. Introduction

Artificial neural networks have seen a recent resurgence in artificial intelligence and machine learning. This popularity can be attributed to their use in *deep learning* – so named because deep networks consist of multiple layers of artificial neurons. Each layer of the network can be thought of as applying a function to the current representation of the data (Goodfellow et al. 2016). In general, the computation sequence can be expressed as a directed acyclic graph of function applications, as shown in Figure 1. In the special case of a feedforward network, there are no feedback effects; that is, the output of the later layers is never returned to earlier layers. Deep learning models have already shown great promise in such diverse areas as computer vision, natural language processing, heuristic game-playing, and others. The aim of this paper is to explore a relatively simple application of one kind of network – a deep autoencoder – to a synthetic dataset of undersampled point-spread function data.

The plan of this paper is as follows. In section 2, we review the general principles behind using autoencoders for dimensionality reduction and learning function representations. In

section 3, we discuss the astronomical problems our methods can be applied to, as well as the current favored solutions for these problems. In section 4, we discuss various considerations for network design and training, including choice of network structure, activation function, and training strategy. In section 5, we present qualitative and quantitative analysis of the success of our model for recovering and reconstructing data. Finally, in section 6, we discuss our findings and propose directions for future work.

Our models are built on top of *Keras* (Chollet 2015), an open-source, back-end agnostic deep learning framework that currently supports both Theano and TensorFlow.

2. Literature Review

It should be noted that the neural networks used in machine learning are not intended to replicate the models of the brain developed by neuroscientists, but that does not stop designers of these networks from drawing on neuroscientific insights. Models of energy usage in the brain (Attwell & Laughlin 2001) drawing upon by fMRI imaging (Shulman & Rothman 1998) suggest that the brain uses sparse, distributed codes to represent information. In the context of a multilayer network, a distributed code means that information can be represented by a path through the network, instead of simply the activation of a given neuron.

Learning codes can be a powerful way of discovering structure in data. The notion of dimensionality reduction supposes that there is some redundancy in a data set, that is, the structure of some D dimensional input can actually be expressed in L dimensions, where $L < D$. This structure can then be used to make inferences. The basic idea is this: given N samples of D-dimensional input, create a representation of the contributing latent variables in L dimensions, also known as a code. Then, from this L dimensional representation,

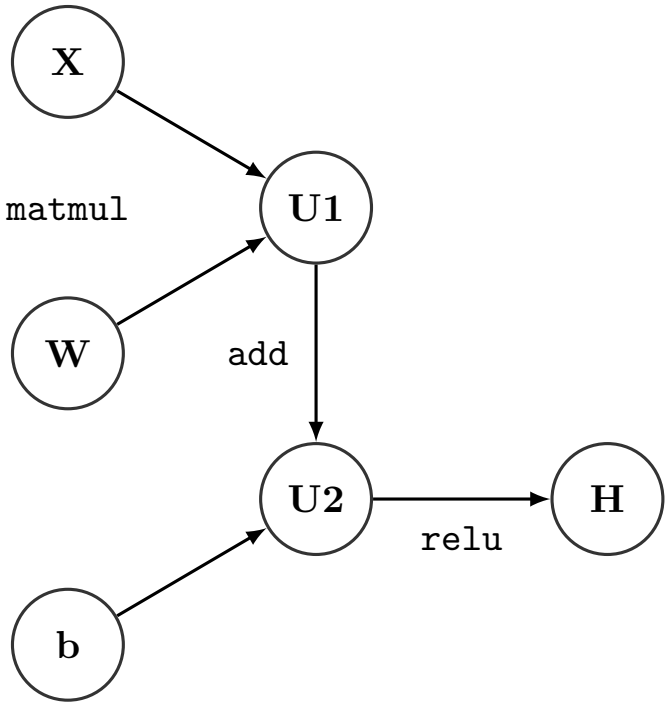


Fig. 1.— The computation behind a single layer of a feedforward network. Here, **X** is an $N \times d$ matrix of N d-dimensional samples, **W** is a matrix of weights, **b** is a vector of biases, **U1** and **U2** are intermediate layers, and **H** is a hidden layer.

reconstruct the D dimensional input. An autoencoder is a feed-forward, heirarchical neural network that attempts to learn this representation/reconstruction pattern. It does so in two parts: an *encoder*, which learns to generate the latent representation from the input, and a *decoder*, which learns to reconstruct the original input given the latent representation (Goodfellow et al. 2016).

On the surface this does not sound so different from the family of linear dimensionality reduction methods, each of which also attempt to capture latent representations. The most widely-used linear method is principal components analysis (PCA), which attempts to learn the (orthogonal) directions of maximal variance in a dataset. However, PCA does not do well when there are significant non-linearities in the data. For instance, if an important feature is actually related to a combination of two or more input variables then PCA will not able to learn that feature. (Ivezic et al. 2014).

Autoencoding networks, however, are significantly more powerful. They can be constructed arbitrarily deeply and can incorporate non-linear activation functions to learn non-linear features of data (Hinton & Salakhutdinov 2006), and can use many hidden layers as well as powerful techniques for achieving stronger performance, including convolution, pooling, and dropout.

When each data point is a real vector $\mathbf{x}$, we can view the encoder portion of the network as

$$\mathbf{h} = f(\mathbf{x}); \mathbb{R}^D \rightarrow \mathbb{R}^L \tag{2.1a}$$

and the decoder portion of the network as

$$\hat{\mathbf{x}} = g(\mathbf{h}); \mathbb{R}^L \rightarrow \mathbb{R}^D. \tag{2.1b}$$

Thus $\hat{\mathbf{x}} = g(f(\mathbf{x}))$. Our goal is to learn the functions f and g, but we also want to capture latent structure in the data. A danger associated with an autoencoder that is too powerful

is that it may learn the identity, that is, it may simply map $\mathbf{x} = \hat{\mathbf{x}}$ everywhere (Goodfellow et al. 2016). We can confront this by encouraging our representation to be sparse, either by employing explicit regularization or in our choice of activation unit. A consequence of encouraging sparsity is that a sparse network will have less capacity than a dense network with the same number of neurons; this can result in better generalization, but also in undercomplete representations. The number and dimensionality of the hidden layers in a network remain hyperparameters to be optimized. If the intrinsic dimensionality of the data is known, then the innermost layer should have no fewer dimensions.

The choice of hidden units is another important consideration for designing autoencoders. Nonlinear hidden units allow the network to learn nonlinear functions. The hidden units are determined by the activation function used after each layer. Generally the choice of hidden units are consistent throughout a network, but there is no reason that they must be. A common choice of hidden unit in current work is the rectifying linear unit (rectifier or relu) which works as

$$\text{rect}(z) = \max(0, z). \tag{2.2}$$

The advantages of rectifiers are discussed in Glorot et al. (2011), but the main idea is that they better promote sparsity than other common choices of activation functions such as the hyperbolic tangent or logistic sigmoid activations, because they zero out many activations. When each layer's weights are initialized uniformly near zero, around half of the hidden units are outputted as zero. This contrasts with hyperbolic tangent units that will be negative when the weight is negative, and sigmoid units that gradually tend towards zero, as shown in Figure 2. The networks we investigate primarily use linear units for input and output, which we believe is a reasonable choice because sigmoid or hyperbolic units squash the representation too much to create plausible reconstructions, and because using rectifying units for input could lead to the network disregarding some of the input data.

When the input is multidimensional (either in spatial extent like image data or temporal extent like audio data), networks can improve their generalization performance by employing one or more convolutional layers. The use of convolutional kernels for smoothing or feature detection is well studied in machine learning and signal processing (e.g. Perona & Malik 1990), but prior to the advent of convolutional networks these kernels had to be engineered by hand. Convolutional networks instead learn feature detecting kernels from the data. Although we have so far only investigated one-dimensional input, we suspect that convolutional layers will play an important role when generalizing this problem to higher dimensions.

The use of stacked autoencoders for recovering structure in undersampled signals has been studied by Mousavi et al. (2015) for image data. It shows some promise for this application, both because of the low computational complexity at query time (compared to more expensive optimization-based methods) and because of the network's ability to learn nonlinear represenations. However, the training and evaluation in Mousavi et al. (2015) is done in a supervised way, with well-sampled signals used as ground truth. Our approach is unsupervised and attempts to learn the underlying structure from the undersampled signals.

3. Astronomical Motivation

Given a constant number of pixels, there is an inherent tradeoff between field-of-view and resolution. Sampling an image to the seeing of a telescope over the desired field of view can be prohibitively expensive, so images are often undersampled, that is, they are sampled below the critical Nyquist sample spacing. This necessitates techniques to restore the signal lost via undersampling. These techniques are especially important for space telescopes, which face a dual challenge with their camera instrumentation. First, the

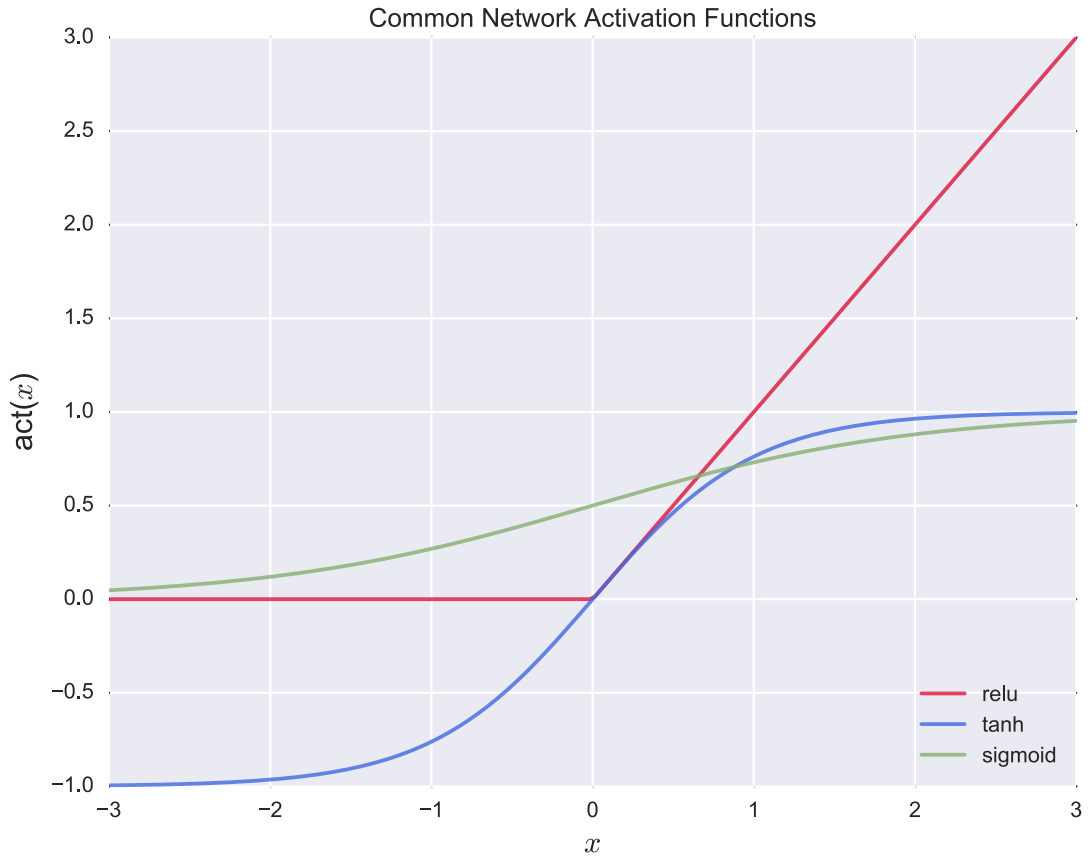


Fig. 2.— A visualization of three common choices of activation function. Rectifiers act in a linear way if they are activated, and are zero otherwise. Hyperbolic tangent and logistic sigmoid units both squash more extreme values into a limited range, but when the weights are uniformly initialized but near zero, twice as many neurons are active when compared to rectifiers.

telescope's optics are diffraction-limited rather than limited by atmospheric effects, so the camera becomes much more of a limiting factor. Second, the camera technology used in space telescopes is not easily replaceable or upgradeable, and the existing instrumentation severely undersamples the point-spread function (PSF) of the optics. For the Hubble telescope's wide-field camera, the width of a pixel is about equal to the full width half maximum (FWHM) of the optics in the near-infrared, and greatly exceeds the FWHM in the blue (Fruchter & Hook 2002).

Most techniques used in astronomy for reconstruction are linear, including the widely-used Drizzle algorithm (Fruchter & Hook 2002), which uses dithering to more fully sample the PSF. Dithering is a technique of spatially offsetting the telescope, and dithering by less than one pixel can sample the PSF at multiple spatial locations. An advantage of the Drizzle algorithm compared to Fourier analysis or linear algebra based approaches is that it allows the use of arbitrarily spaced samples, rather than exposures that require certain relative offsets. This advantage is shared by the Imcom algorithm of Rowe et al. (2011), which further improves upon Drizzle by allowing finer control over the PSF in the output image. However, both the Imcom and Drizzle algorithms do not do well when certain inputs are not well-known in advance, specifically, the PSF at the position of each pixel center. When the spatial variation of the PSF is not exactly known, using these linear reconstruction methods is not very feasible (Rowe et al. 2011).

The specific motivation behind this project comes from the Prime Focus Spectrograph (PFS), an instrument planned for the Subaru 8.2 meter telescope. Raw spectral data is noisy. An image contains contributions from the instrument's PSF and spectral lines from the night sky as well as the spectral lines of the object of interest. Controlling for the contributions of the instrument and the night sky is critical or else they dominate the image. This challenge is compounded by the fact that the PSF is undersampled. Furthermore, the

PSF varies over the physical extent of the instrument, as well as over the wavelength range sampled, via an unknown function.

The main objective is to learn a model of the PSF from the undersampled data such that we can query the model at an arbitrary point on the detector and wavelength to find a value to subtract from an image. We can first consider this problem in one dimension, and consider two parts in one dimension: First, training a network to recover the values of a function given the undersampled input data. Second, training a network to recover a well-sampled but spatially varying function. We focus primarily on the undersampled problem in this paper, but suspect that similar methods can be leveraged for the second problem and combined together for a useful solution. The current state of the art solution would follow Lupton et al. (2001) and use an upsampling algorithm to deal with the undersampled data combined with PCA for the spatial variation over the domain.

4. Methods

4.1. Synthetic Data

Our synthetic dataset is based off a simple model of a PSFs. Each sample is a normalized Gaussian, centered at some x_c evaluated at some n pixels ranging from -1 to 1, such that

$$I(x, x_c, \sigma) = \frac{\exp \frac{((x-x_c)/\sigma)^2}{2}}{\Delta x \sum_0^n \exp \frac{((-1+(\Delta x n)-x_c)/\sigma)^2}{2}} \tag{4.1}$$

where the normalization is done in such a way because the PSF is evaluated discretely and over a finite range.

As we generally train on relatively small datasets (between 500 and 5000 training samples), and as the network performance is at least somewhat sensitive to the input data, we use consistent random seeds for data generation and for splitting the data into training

and validation sets. We evaluate the performance of our networks on the validation set both qualitatively and quantitatively: qualitatively by the visual appearance of the fit including the degree to which there are "obviously wrong" features, and quantitatively via the χ^2 values, both for a given x and as a spatial average across the network.

4.2. Network Architecture

The autoencoder is a simple feed-forward network using a stacked encoder and decoder structure (see Fig 3). We ran several experiments with a variety of network architectures, to varying results. It turns out that there are several important considerations to make when building a network, including the depth, the number of neurons in the innermost layer, and whether or not the network is overcomplete or undercomplete. When comparing network designs we have made an effort to control for these effects - for example by addding additional layers with the same dimensionality to give an undercomplete network the same depth as an overcomplete one.

There is some theoretical basis supporting the use of deep networks. Montúfar et al. (2014) examine feedforward rectifying networks with n_0 input units and L hidden layers of width $n \geq n_0$ and find the following lower bound on the number of linear regions the network can compute.

$$\Omega\Big(\big(\frac{n}{n_0}\big)^{(L-1)n_0} n^{n_0}\Big). \tag{4.2}$$

Because our networks attempt to find a low dimensional representation, they do not fulfill the criteria that $n \geq n_0$ for all layers, but there is also strong empirical support behind deep networks (see, e.g Glorot et al. 2011; Hinton et al. 2012; Kulkarni et al. 2015) even when the individual layers have fewer nodes than the input nodes. However, there are trade-offs associated with using very deep architectures. The networks are hard to optimize because the networks have a large number of parameters, the gradients of which may vary

substantially across the network unless initialized correctly. Furthermore, the networks are powerful enough that they may capture a lot of noise in the training set, leading to poor generalization performance (Hinton et al. 2012). Using regularization can help reduce overfitting, but at the cost of also reducing the ability of the model to capture a complex function. Enforcing layer-wise regularization on the network was not a high priority in our work because the small dimensionality of the innermost layer and the sparsity-inducing rectifying units already serve as network-scale regularizers.

The number of neurons in the innermost layer also appears to have an effect on the rate at which gradients propagate through the network. Our experiments on data with only 1 latent variable show a decreased rate of convergence with respect to number of training epochs in a network with one neuron in the innnermost layer compared to networks with three or five neurons in the innermost layer. However, the networks with five central neurons did not show markedly different convergence compared to those with three. We generally choose to use a network with three central nodes.

The network shown in Figure 3 is an overcomplete network which means that at least one of the hidden layers has greater dimensionality than the input layer. We investigate both overcomplete and undercomplete networks.

4.3. Network Training

Training deep networks continues to be an area of active investigation. One property that has become desirable recently is that of disentanglement – that is, training a network in such a way that only one property is varied over a given batch (Kulkarni et al. 2015). Other considerations include the choice of initial weights, the pretraining scheme, the loss function, and the training algorithm itself. Each of these factors can affect the network's

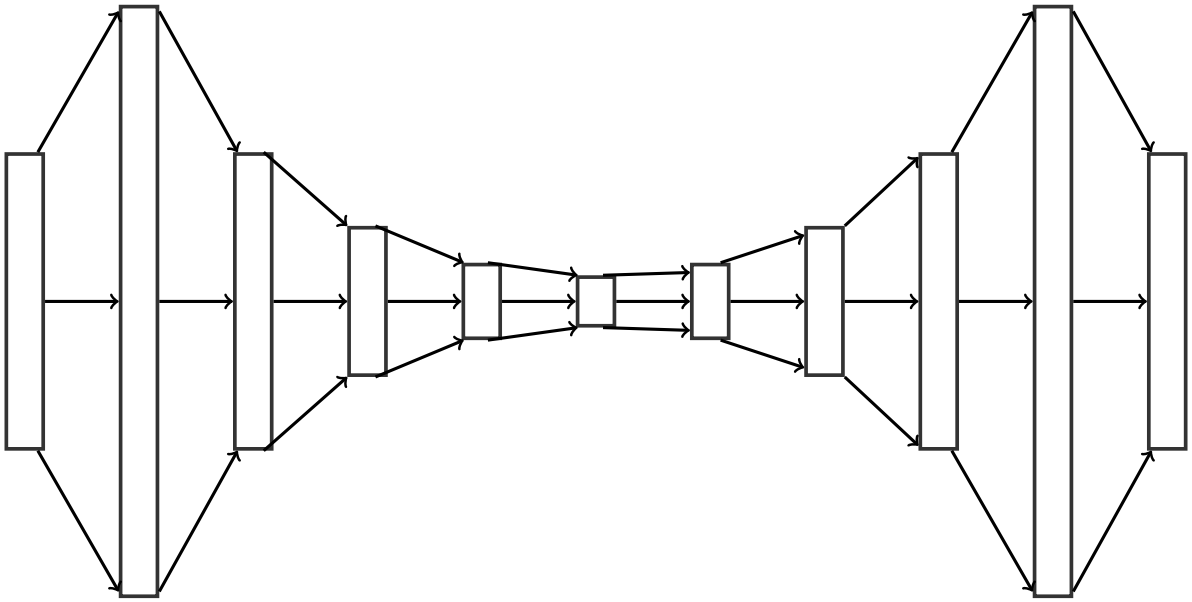


Fig. 3.— The structure of a simple feedforward autoencoder. This autoencoder uses an overcomplete layer which has twice the number of input nodes as the first layer. This is just like our network `T` which has a 20-40-20-15-7-3 structure.

overall performance.

As deep networks have a large search space, it is not feasible for training algorithms to find global optima, so it is important to initialize the network so that it can find a reasonable local optimum. For example, we know that our point-spread function should be positive, so we can initialize the input and output layers with positive random weights. The weights for the hidden layers are chosen following Glorot & Bengio (2010), and use the following heuristic:

$$W_{ij} \sim \mathcal{N}\big[-\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}}\big] \tag{4.3}$$

where n is the input dimension of the layer. When combined with rectifying units, this means that on average half of the neurons from a given layer will produce hard zeros.

Hinton & Salakhutdinov (2006) introduced the idea of initializating each layer of a deep network using unsupervised pretraining. Their pretraining method is to construct multiple two-layer binary restricted Boltzmann machines (RBMs), and to feed the output of each RBM to the next as the input (what they call the visible unit). They then use the weights of each RBM to initialize the layers of the autoencoder, initially setting the weights in the decoder layer as the transpose of the corresponding weights in the encoder layer. The loss function that Hinton & Salakhutdinov (2006) target is an energy function of the visible and hidden units $\mathbf{v}$ and $\mathbf{h}$ given by

$$E(\mathbf{v}, \mathbf{h}) = -\sum_{i \in visible} b_i v_i - \sum_{j \in hidden} b_j h_j - \sum_{i,j} v_i h_j w_{ij} \tag{4.4}$$

where b is the corresponding bias and $\mathbf{W}$ is a weight matrix. Our pretraining scheme is somewhat different. We instead use a series of autoencoders, each with a single hidden layer. Concretely, suppose we had a network with the dimensions 20-10-5-3 and a design matrix $\mathbf{X_{train}}$. We would construct three autoencoders, a 20-10-20 autoencoder, a 10-5-10 autoencoder, and a 5-3-5 autoencoder. Each layer in each of these autoencoders is then initialized by the chosen initialization function. For example, in the 20-10-20 layer, each

of the 20 neuron layers are initialized with positive uniform initialized, and the 10 neuron layer is initialized with the Glorot normal initialization. The 20-10-20 autoencoder is then trained on $\mathbf{X_{train}}$ (our datasets are small enough that we train on the whole dataset rather than using batch training), targeting the mean-squared error of the reconstruction, for a specified number of training epochs, where a training epoch is a pass over the entire dataset. To generate the input for the 10-5-10 autoencoder, we create a new design matrix $\mathbf{H_1}$ which is the result of the hidden layer from the trained 20-10-20 network. After training all of the autoencoders, we would create an unrolled autoencoder, where the weights in the encoder and decoder are taken from the corresponding weights in their respective single-layer encoder and decoder. The full network is then trained on $\mathbf{X_{train}}$, using stochastic gradient descent and again targeting the mean-squared error of the reconstruction.

Glorot et al. (2011) suggest that layer-wise unsupervised pretraining may not be neccessary for deep rectifying networks, as long as the randomly chosen initial weights are reasonable. However, they evaluate networks for supervised classification tasks on labeled data. These networks are trained using the cross-entropy loss between the predicted labels and ground truth. Hinton & Salakhutdinov (2006) report that attepting to use a deep autoencoder with sigmoid activations without pretraining results in a network that just reproduces the mean of the input data. Our experiments with using pretraining each layer show a modest increase in the speed of convergence versus not pretraining at all. It does appear that pretraining is not strictly necessary for rectifying networks, even for unsupervised autoencoders.

5. Results and Discussion

Our investigation primarily focuses on building an autoencoder which is capable of learning the upsampling task. We consider a number of model architectures and attempt

to quantify their performance via reconstruction quality and convergence speed. We also consider the effects of how dramatic the undersampling is on reconstruction quality. The default width of our simulated PSF is $\sigma = 2$ px, which is used unless otherwise specified.

We begin by qualitatively discussing which factors seem to have produced the most effective networks, measured by the visual quality of the reconstruction over the validation set. One important factor is the initialization function. Without a good choice of the initialization, the network learns a poor representation of the data because gradient descent is not powerful enough to move out of poor local minima. One common outcome of poor initialization is that the network learns an average of the training set and outputs the average whenever it is queried. We found this was the case when using constant initialization functions, but the initialization function makes a difference even when the choice is reasonable. Using a uniformly random initialization function centered at zero for the input and output layers was less effective than using a uniformly random function with a floor at zero. We suspect this is because the positively uniform initialization takes advantage of the fact that we know the PSF is positive.

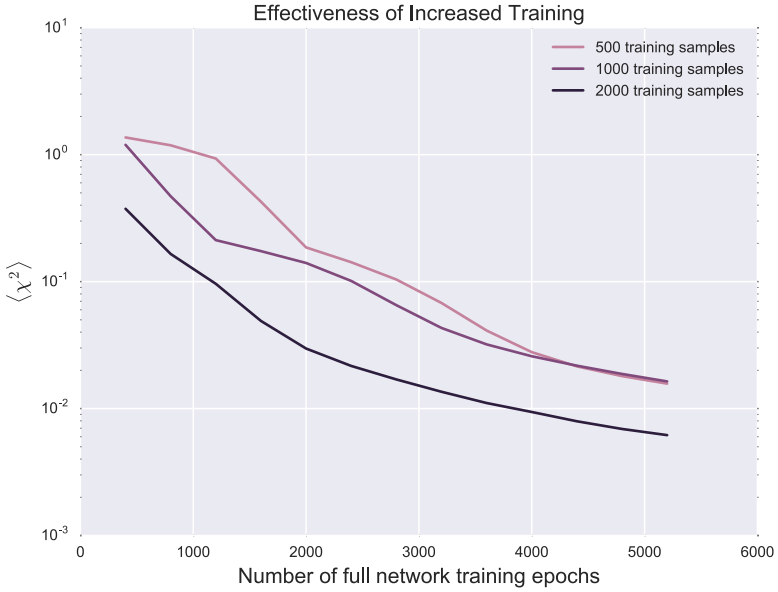
Another factor in both the speed of convergence and the generalization quality is the dimensionality of the innermost layer. Comparing rectifying networks of equal depth with five, three, and one central nodes showed the most rapid convergence and generalization quality for the network with three central nodes, despite the fact that not all of the neurons were always active. Although it is possible that gradients can propagate better through the layer with three neurons, it's not immediately clear that this occurs – one network with three central neurons we investigated only produced non-zero activations for one neuron anyways, as shown in Figure 7. It is possible that the networks with five neurons in the innermost layer are insufficiently regularized, which could explain why they struggle to generalize to the validation set given the same amount of training.

Increasing the size of the training set promotes convergence in a smaller number of epochs. However, it also appears to improve the asymptotic bound, as shown in Figure 4. Increasing the number of samples is also helpful when the width of the PSF is small. At a width of $\sigma = 0.5$ px, we were not able to obtain convergence with 500 training samples, but we were able to do so with 1000. This has implications for the undersampled case. $\sigma = 2$ px is actually reasonably well sampled. But it seems that the network needs a certain baseline amount of information in order to converge at all, so when the sampling rate is lower, the network needs more samples.

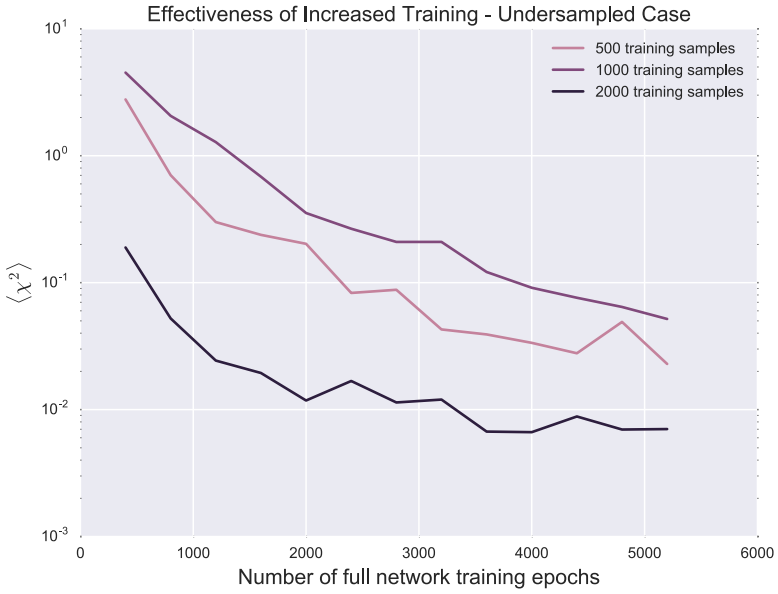
Because the variance among each curve in our sample is fully explained by x_c, our training scheme should produce a disentangled code where there is a clear correspondence between the code value and x_c. This does seem to occur, judging by the relationship shown in Figure 7. It should therefore be possible to query the decoder network at an arbitrary x_c; first by converting the x_c value to the code value (perhaps via interpolation) and then by decoding the code value.

6. Conclusions

In this work we investigate the use of deep autoencoders to learn representations of undersampled functions. We find that rectifying networks are capable of learning a sparse, disentangled code and generating reconstructions that are faithful to the original inputs. We also find that there are some nuances to training deep autoencoders, including the choice of initializations, the pretraining scheme or lack thereof, and the depth of the network. Future work extending into multiple dimensions and attempting to capture more complex latent representations will have to be more careful about the training scheme in order to produce a disentangled code. Transforming from the query space to the code space may also pose a challenge in general.

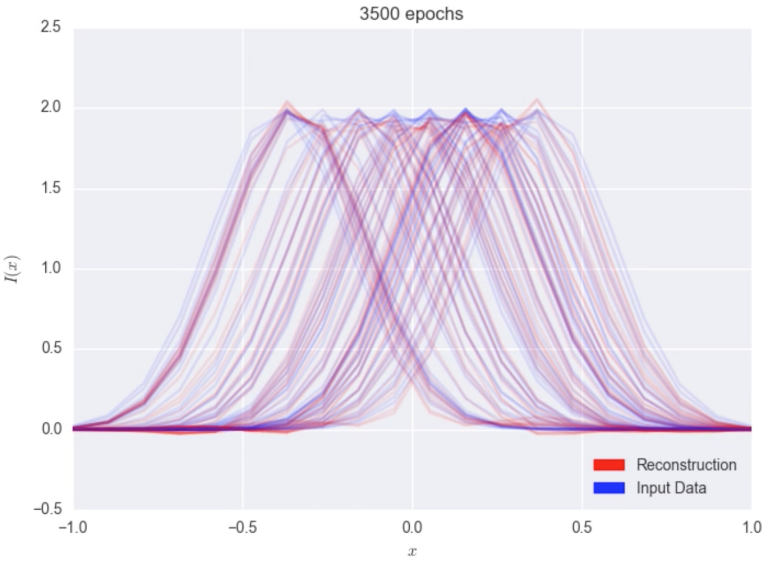


(a)

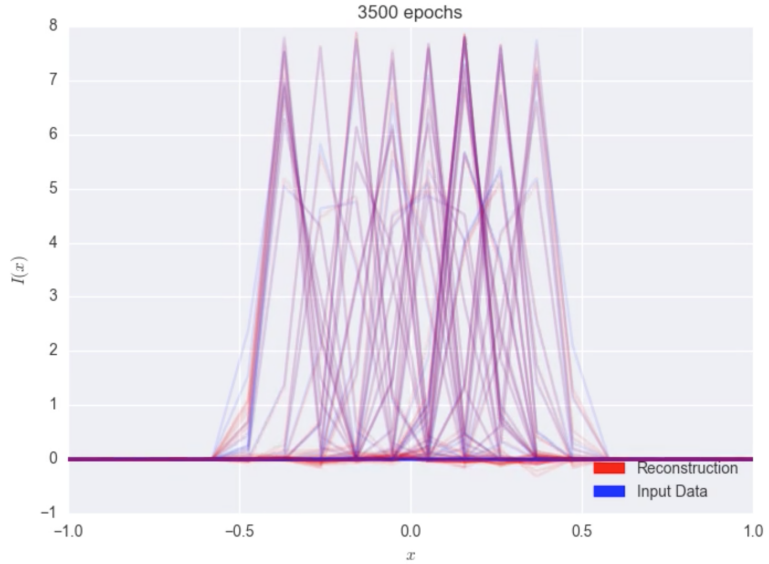


(b)

Fig. 4.— χ^2 versus training epoch colored by number of samples for a network with dimensions 20-40-20-15-7-3. (a) is for a well-sampled network with $\sigma = 2$ px. (b) is for an undersampled network with $\sigma = 1$ px.



(a)



(b)

Fig. 5.— A comparison of reconstructed PSFs versus input PSFs for the well-sampled (a) and undersampled (b) case after 3500 epochs of full-network training. Both use a training set of 500 curves with values of x_c between -0.4 and 0.4.

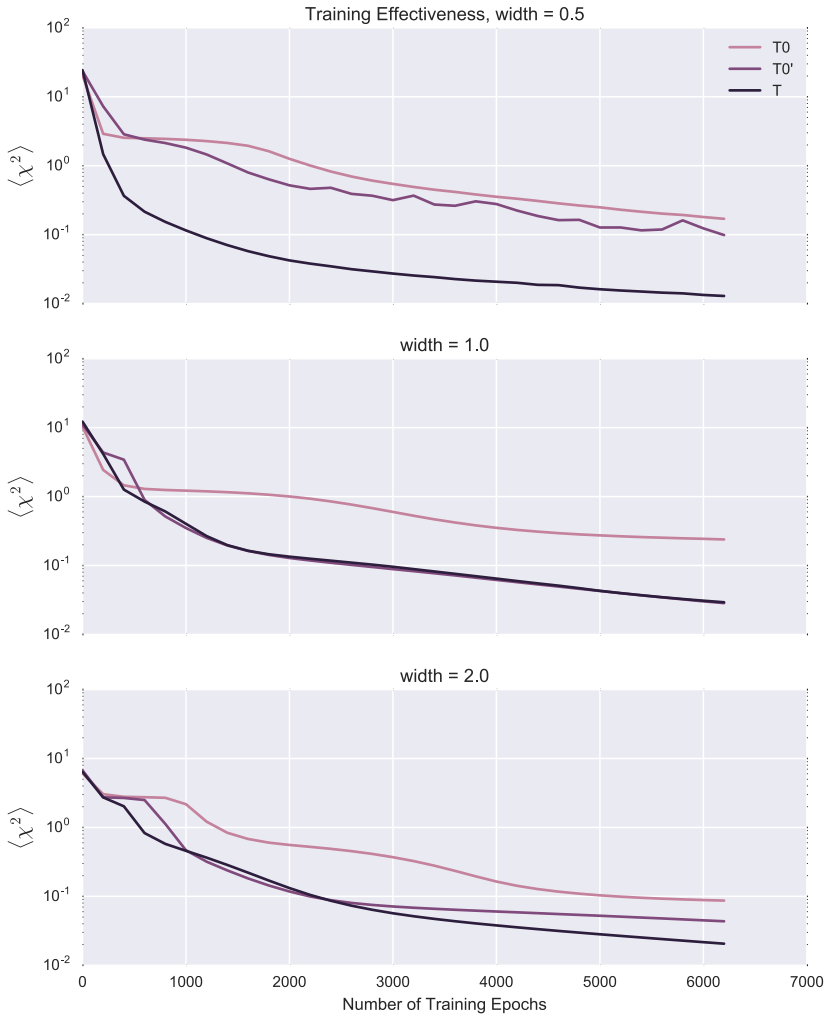
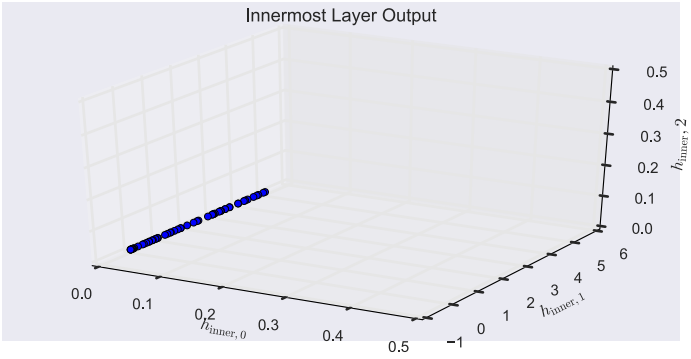


Fig. 6.— A comparison of training effectiveness for three networks on 500 training samples with widths of $\sigma \in \{1 \text{ px}, 2 \text{ px}, 4 \text{ px}\}$ respectively. $\langle \chi^2 \rangle$ is computed over the validation set. The overcomplete network `T` seems to outperform the undercomplete networks when the width is especially small. `T0'` has the same depth as `T`, and their approximately similar performance when the width is not a limiting factor suggests that depth is also helpful.



(a)

(b)

Fig. 7.— The first subfigure shows that the innermost layer output only varies along one node and the other two are hard zeros. The second subfigure shows the non-zero values for the innermost layer versus the x_c corresponding to the input curve after 3000 epochs of training on 500 samples. The network has dimensions 20-40-20-15-7-3 and rectifying activation functions. Only the middle neuron in this layer is activated - the other neurons yield hard zeros everywhere.

We should also note that the networks in Kulkarni et al. (2015); Kingma & Welling (2013) have explicit stochastic layers where they parameterize a distribution. Extensions of this work should likely investigate using the code layer as the parameters of a distribution passed to the reconstruction layer. This may further improve generalization performance.

I extend thanks to my advisors Peter Melchior and Robert Lupton for their help and patience, as well as to my friends in the Astrophysics major and Cloister Inn, who did a remarkably good job of keeping me in high spirits.

REFERENCES

Attwell, D., & Laughlin, S. B. 2001, Journal of Cerebral Blood Flow & Metabolism, 21, 1133

Chollet, F. 2015, Keras, `https://github.com/fchollet/keras`

Fruchter, A., & Hook, R. 2002, Publications of the Astronomical Society of the Pacific, 114, 144

Glorot, X., & Bengio, Y. 2010, in International conference on artificial intelligence and statistics, 249–256

Glorot, X., Bordes, A., & Bengio, Y. 2011, in International Conference on Artificial Intelligence and Statistics, 315–323

Goodfellow, I., Bengio, Y., & Courville, A. 2016, book in preparation for MIT Press

Hinton, G., Deng, L., Yu, D., et al. 2012, Signal Processing Magazine, IEEE, 29, 82

Hinton, G. E., & Salakhutdinov, R. R. 2006, Science, 313, 504

Ivezic, Z., Connolly, A., VanderPlas, J., & Gray, A. 2014, Statistics, Data Mining, and Machine Learning in Astronomy: A Practical Python Guide for the Analysis of Survey Data, EBSCO ebook academic collection (Princeton University Press)

Kingma, D. P., & Welling, M. 2013, arXiv preprint arXiv:1312.6114

Kulkarni, T. D., Whitney, W. F., Kohli, P., & Tenenbaum, J. 2015, in Advances in Neural Information Processing Systems, 2530–2538

Lupton, R., Gunn, J. E., Ivezic, Z., et al. 2001, ASP Conf. Ser., 238, 269

Montúfar, G., Pascanu, R., Cho, K., & Bengio, Y. 2014, ArXiv e-prints, arXiv:1402.1869

Mousavi, A., Patel, A. B., & Baraniuk, R. G. 2015, ArXiv e-prints, arXiv:1508.04065

Perona, P., & Malik, J. 1990, Pattern Analysis and Machine Intelligence, IEEE Transactions on, 12, 629

Rowe, B., Hirata, C., & Rhodes, J. 2011, The Astrophysical Journal, 741, 46

Shulman, R. G., & Rothman, D. L. 1998, Proceedings of the National Academy of Sciences, 95, 11993