ROBOTICS AND REINFORCEMENT LEARNING

DEEP LEARNING: SPARSE AUTOENCODERS

November 26, 2014

INTRODUCTION

Deep learning describes a category of algorithms that use a pipeline of neural networks to learn a hierarchy of features, typically for classification tasks. Over the past 5 to 10 years, deep learning based algorithms have been shown to be a very effective way to process sensory information and have been used in a number record breaking algorithms against competitive datasets such as TMINST (for sound) and ImageNet (for vision). Models of this kind have free parameters

in the millions, and the challenge of deep learning methods is the challenge of understanding complex and evolving computational systems. A number of advancements in training along with the computational advantages of the GPU have made deep learning tractable where 10 years ago these kinds of networks were too big to run and too difficult to train efficiently. My goal in these deep learning articles is to briefly overview recent advancements in deep learning that have lead to a resurgence of interest in these kinds of information processing systems one at a time. You should be able to find other posts on this topic here. I will assume a basic familiarity with neural networks is known.

SPARSE AUTOENCODERS

Typically in a supervised learning task, a neural network with many layers (deep) is initialized with random weights from a Gaussian distribution. Performing back-propagation directly on a freshly initialized network will tend to be very slow and get stuck in poor local minima as the loss function is highly pathological when parameterized by millions of free variables with complex dependencies. This leads to deep neural networks that take a long time to train and yeild poor results. In his highly influential work on deep networks, Hilton showed that if you pretrain each layer of the network in an unsupervised manner to learn a sparsified representation before the

classification task the learning problem was greatly reduced (Hilton et al., 2006).

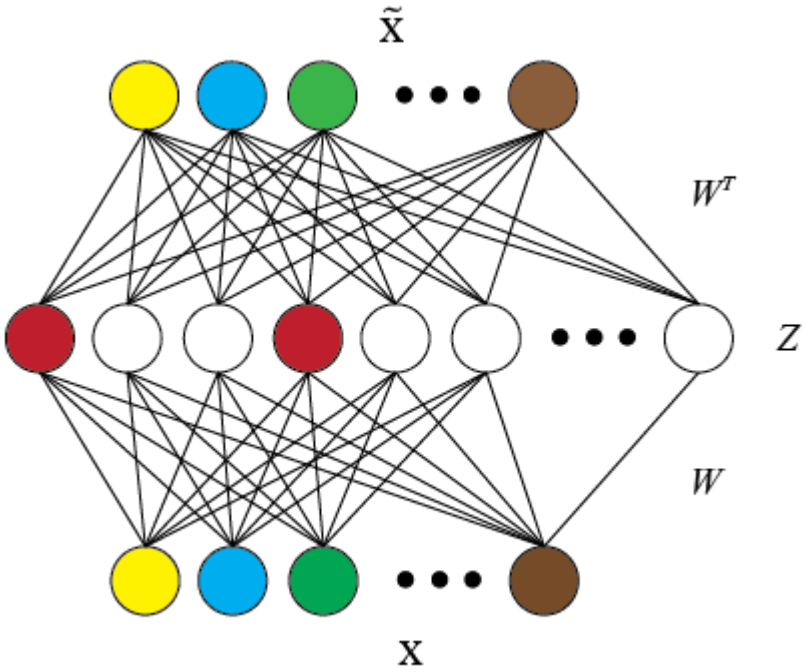
An autoencoder takes as input a feature vector $\mathbf{x}$ and learns a code dictionary that changes the raw input from one representation (presumably an inefficient one) to another. Sparse autoencoders are a type of autoencoder with a sparsity enforcer that directs a single layer network to learn a code dictionary that minimizes reconstruction error while restricting the number of code-words required for reconstruction. In fact, we can think of the task of classification as a kind of sparsifying algorithm that reduces the input to a single class value that minimizes prediction error.

The simplest sparse autoencoder consists of a single hidden layer, h, that is connected to the input vector $\mathbf{x}$ by a weight matrix W forming the encoding step. The hidden layer then outputs to a reconstruction vector $\widetilde{\mathbf{x}}$, using a tied weight matrix $W^T$to form the decoder (although the matrix could be untied separating the encoder from the decoder). The activation function is f and $\mathbf{b}$ is the typical bias term.

$$\mathbf{z} = f(W\mathbf{x} + \mathbf{b})$$
$$\widetilde{\mathbf{x}} = f(W^T\mathbf{z} + \mathbf{b}')$$

Learning occurs via backpropogation on the reconstruction error.

$$\min ||\mathbf{x} - \widetilde{\mathbf{x}}||_2^2$$

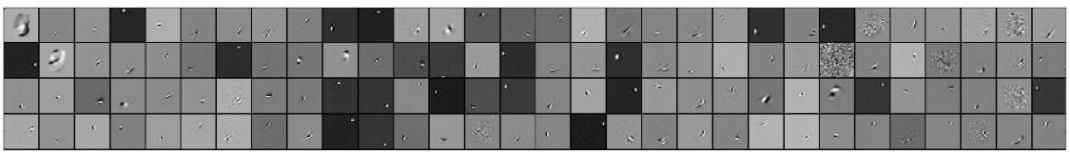


A diagram of a sparse autoencoder network. The input vector $\mathbf{x}$ is converted to a sparse representation on the hidden layer as $\mathbf{z}$ and then reconstructed as $\tilde{\mathbf{x}}$.

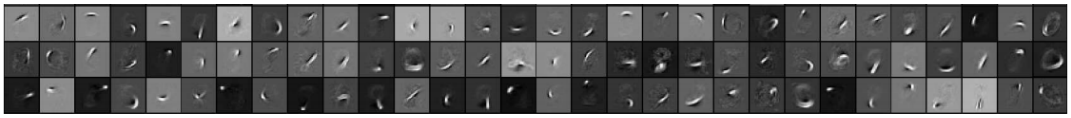
Now that we have the network setup, the next step is to add a sparsifying component that drives the vector $\mathbf{z}$ towards a sparse representation. There are many ways in the literature to accomplish this, but the one I find the simplest to implement as well as understand comes from k-Sparse Autoencoders (Makhzani et al., 2013). k-Sparse Autoencoders finds the k highest activations in $\mathbf{z}$ and zeros out the rest. The error is then backpropogated only through the k active nodes in h. For low levels of k (very sparse), k is scaled down incrementally over the course of training. A percise mathematical formulation can be found in their paper.

One of the nice things about this k-Sparse Autoencoders are they allow for a clear exploration on the effects of sparsity on a dataset in terms of percentage activation of the network. For example, the MINST dataset consists of 10,000 images of handwritten digits stored as 28x28 such as the 4 digit shown below.

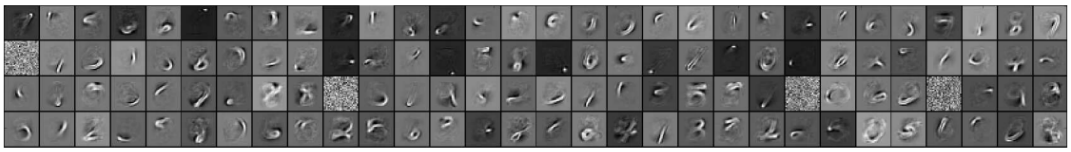
In their paper, Makhzani et al. explore different values of k on MINST. The results show that as the value of k decreases the network is forced to learn increasingly complete representations of each individual digit. For $k = 70$, the representation is over-complete and the autoencoder learns highly local features involving small stroke and blob detectors. For $k = 10$ the representation is so sparse that each node in the network can only represent features from a single digit and have become completely global. This makes sense intuitively because in order to reconstruct each handwritten digit from only 10 basis functions, each basis must closely resemble the full image.



(a) $k = 70$



(b) $k = 40$



(c) $k = 25$

(d) $k = 10$

This figure shows the effect of sparsity on the MINST dataset of handwritten digits. The k-Sparse Autoencoder trained on each example had 1000 hidden units. For high k-values the features learned are highly local while low k-values learned features that were global and specific. The best classification performance was achieved using $k = 40$. Figure taken from (Makhzani et al.,2013).

There are many other flavors of autoencoders. Denoising autoencoders were introduced by Bengio's group and operate by attempting to accurately reconstruct the input after a percentage of the data has been randomly removed (Vincent et al., 2008) . This forces the network to learn robust features that tend to generalize better. This idea later served as the basis for dropout. Stacking autoencoders so that each layer of the network learns an encoding of the layer below creates a network that can learn hierarchical features in an unsupervised manner (Vincent et al., 2010). It turns out that training stacked layers in this manner allows a deep network to learn a good representation incrementally instead of trying to train the whole network in ensemble from a random initialization of weights. A wonderful investigation into why pretraining with autoencoders yields better results can be found here.

REFERENCES

Hilton Geoffrey E., Stinchcombe Maxwell, and Teh Yee W. (2006) A fast learning algorithm for deep belief nets. Neural Comput., 18(7):1527-1554, July 2006

Makhzani Alireza, Frey Brendan (2013) preprint arXiv:1312.5663 [cs.LG]

Vincent P., Larochelle H., Bengio Y. and Manzagol P.A. (2008), Extracting and Composing Robust Features with Denoising Autoencoders, Proceedings of the Twenty-fifth International Conference on Machine Learning (ICML'08), pages 1096 - 1103, ACM.

Vincent P., Larochelle H., Lajoie I., Bengio Y., and Manzagol P.A. (2010), Stacked denoising autoencoders: learning useful representations in a deep network with local denoising criterion, J Mach. Learn. Res., vol. 11, no. 11, pp.3371-3408.

D. Erhan, A. Courville, Y. Bengio, and P. Vincent, "Why does unsupervised pre-training help deep learning?" in Proceedings of AISTATS 2010, vol. 9, May 2010, pp. 201–208.

In Robotics, Sensors Tags Deep Learning, AutoEncoders, Hessian Free

Share 9 Likes

← A Lightweight, Embeddabl... Self-Organizing Kinematics →

COMMENTS (2) Newest First Subscribe via e-mail

Preview POST COMMENT...

zah A year ago · 0 Likes

Hi, Thank you for your tutorial. Please correct some of orthographic errors like Hilton or MINST that are Hinton and MNIST respectively.

Q 2 years ago · 0 Likes

The author name of the first reference should be Hinton :)