Multi-layer auto-encoders
a CS221 artificial intelligence project

Charles Celerier
Computer Science
Stanford
cceleri@cs.stanford.edu

Bill Chickering
Computer Science
Stanford
chickering@cs.stanford.edu

August 18, 2013

This is a short introduction.

1 Introduction

A fundamental challenge to supervised machine learning is that of feature selection. There exist many strategies for determining a set of variables derived from an original representation to accurately classify domain data. Ideally, the selected variables minimize redundant and/or irrelevant information and thereby reduce the risk of overfitting. Historically, domain experts and computer scientists have laboriously handcrafted feature sets for a variety of machine learning problems. These manually engineered features suffer from the requirements of domain expertise and a great deal of work, and therefore, might not scale well. Moreover, such feature sets can inadvertently ignore important structure in the data. These deficiencies can be addressed via unsupervised feature learning using dimension reduction techniques such as principal component anaysis (PCA) or factor analysis. Recently it was found that K-means clustering offers another straightforward method for determining an efficacious feature space (e.g. the CS221 Visual Cortex programming assigment). In this report we document a project in which unsupervised feature learning is performed using autoencoder neural networks to discover feature spaces that can be utilized by a supervised learning algorithm to effectively classify images of handwritten digits.

Our subject is the MNIST database of handwritten digits[1], which includes 50,000 training and 10,000 testing images, each a 28x28 grid of real numbers between 0 and 1. It is worth mentioning that this dataset has been employed before to demonstrate feature learning with autoencoders. Indeed, the pioneering paper by Hinton *et al.***??** that introduced the technique upon which our project is based used this dataset among others. We chose to work with the MNIST data because of its large size, simple format, and obvious classifications thereby allowing us to focus on technique.

2 Method

An autoencoder is a special type of neural network consisting of an input layer, an output layer, and one or more hidden layers. An autoencoder aims to approximate the identity function as closely as possible, reproducing the signals of its input layer at its output layer. By placing constraints on the network, input signals must be compressed and decompressed using optimized encodings in order to approximately reproduce the input. Such encodings can then serve as feature sets in a supervis ed learning problem. The challenge is training such autoencoders in order to discover the optimal e ncoding for a particular domain space.

Until recently, the standard approach to training multilayer feedforward neural networks, such as autoencoders, is to use a variation of the backpropagation algorithm. We describe this algorithm later in this report. For now, we merely point out a key feature of the backpropagation algorithm: it attempts to optimize all weights and biases of a neural network simultaneously. In the case of large networks containing multiple layers and 1000+ nodes, backpropagation following random initialization of the parametes can fail due to

[1] http://yann.lecun.com/exdb/mnist/

local minima in the objective function**??**. A technique pioneered by Hinton *et al.***??** largely overcomes this challenge by "pretraining" individual restricted Boltzmann machines (RBMs), which can then be "unrolled" to form a deep, multilayer autoencoder with weights and biases that are close to optimal prior to running a backpropagation algorithm.

An RBM may be represented as a complete bipartite graph where one set corresponds to "visible" units ν_i and the other to "hidden" units h_j. We work with binary RBMs in which each node is either on or off. A weight w_{ij} is associated with each edge that connects a visible unit ν_i to a hidden unit h_j. Given the state of all visible units, the probability that a hidden unit is in the "on" state is given by $\sigma\left(b_j + \sum_i \nu_i W_{ij}\right)$, where $\sigma(x)$ is the logistic function $1/(1+\exp(-x))$ and b_j is the bias associated with hidden unit j. By coupling the visible and hidden units in this way, an RBM models a particular statistical ensemble of unit configurations, which in our case are images. The technique of Hinton *et al.* "pretrains" an RBM so that it approximately models a training set.

A detailed description of the technique of Hinton *et al.* is beyond the scope of this report; we refer the interested reader to the original papers referenced here as well as to our provided source code. The basic idea is to model our data as a physical system. We may then borrow some statistical machinery from physics to help pretrain our RBMs. Perhaps the most fundamental of these is the *free energy* ascribed to each training image, given by

$$E(\nu, \mathbf{h}) = -\sum_{\mathbf{i} \in \text{visible}} \mathbf{b_i}\nu_\mathbf{i} - \sum_{\mathbf{j} \in \text{hidden}} \mathbf{b_j h_j} - \sum_{\mathbf{i,j}} \nu_\mathbf{i} \mathbf{h_j} \mathbf{w_{ij}}. \tag{1}$$

As per the Maxwell-Boltzmann distribution, the probability of a particular image is proportional to $\exp\left(-E(\nu, \mathbf{h})\right)$. Using stochastic gradient descent to maximize the log-likelihood of our training images, along with an approximation technique known as *contrastive divergence* that allows us estimate an expectation over all possible unit configurations under the probability distribution modeled by an RBM, we iteratively update the biases and weights of an RBM. Once a RBM is optimized in this manner, it is straighforward to create an autoencoder with a single hidden layer by adding an output layer that simply decodes the states of the hidden layer. To construct an autoencoder with three hidden layers, we pretrain an additional RBM using the output (i.e. hidden layer) of the first RBM as its visible layer. Importantly, the hidden layer of the second RBM must consist of fewer nodes than the hidden layer of the first RBM. In this way, multiple RBMs, each with fewer hidden nodes than the one before it, can be unrolled to form a pretrained deep neural network autoencoder.

Once pretrained, an autoencoder can be fine-tuned using backprogation without fear of local minima in our objective function (i.e. the mean squared error). As with the RBM pretraining technique, the details of this classic algorithm for training neural networks are beyond this report's scope and are well documented elsewhere (an example implementation is also provided within our source code). The essence of this algorithm is to propagate the discrepancy between the autoencoder's output and its input back through the neural network, adjusting its parameters along the way.

These autoencoders essentially function by compressing and decompressing the input image. The compression algorithm used has been optimized for the given training data. In addition to providing a domain-optimized compression algorithm, we can also use the dimensionally reduced representation of the data as a feature space. By removing the decoders such that the innermost hidden layer with the fewest nodes becomes the output layer, we construct a feature encoder. To explore the utility of such learned feature spaces, we conducted several supervised learning experiments in which we construct naive Bayes classifiers and compare their error rates.

3 Results

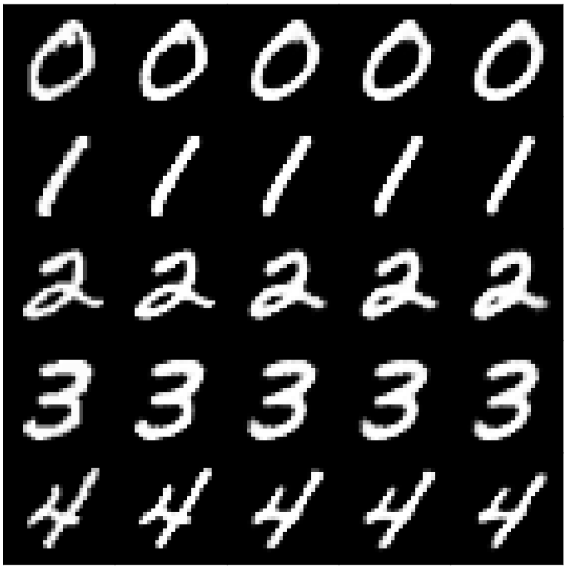
Here's an equation:

$$\mathcal{C} = \frac{\sum_{q \in Q} \sum_{i=1}^{L_q} \mathbb{1}\left\{click\ @\ i \wedge i \leq 16\right\}}{\sum_{q \in Q} \sum_{i=1}^{L_q} \mathbb{1}\left\{i \leq 16\right\}}, \tag{2}$$

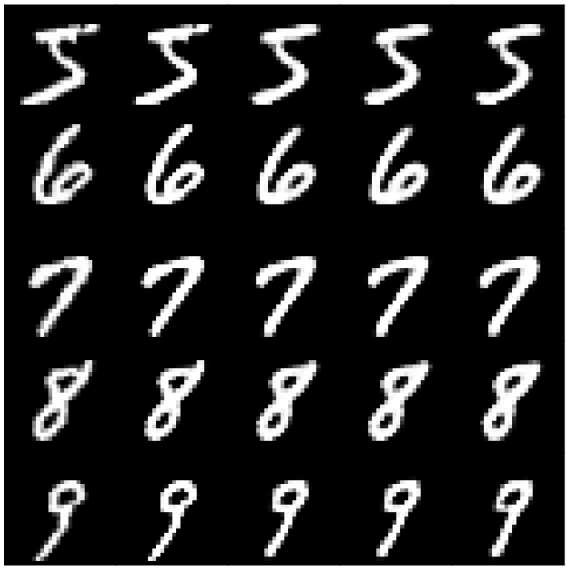
4 Conclusion

References

[1] *Restricted Boltzmann Machines (RBM)*, `http://deeplearning.net/tutorial/rbm.html`.



(a) Encoding of digits $0-4$ in each layer



(b) Encoding of digits $5-9$ in each layer

[2] G. E. Hinton and R. R. Salakhutdinov, *Reducing the dimensionality of data with neural networks*, Science **313** (2006), no. 5786, 504–507.

[3] N. Kambhatla and T. Leen., *Dimension reduction by local principal component analysis*, Neural Computation **9** (1997), no. 7, 1493–1516.

[4] Raul Rojas, *Neural networks - a systematic introduction*, `http://page.mi.fu-berlin.de/rojas/neural/chapter/K7.pdf`, 1996.