# Deep Learning Project 3: Denoising Autoencoder

Elliot Greenlee

December 1, 2017

**Abstract**

Previous work on learning deep models has shown a problem with the training difficulty of purely supervised learning, but an unsupervised solution exists. Mapping inputs to an intermediate representation constrained to certain properties as an initial step can boost supervised performance, and stacks of these autoencoder layers can be used to initialize deep architectures. In this project the denoising constraint is applied to train an autoencoder on image data from CIFAR-10 under various training optimizations. The unsupervised autoencoder representation is compared to a supervised representation produced by LeNet-5 for classification by a support vector machine. Additionally, autoencoder stacking is used to initialize the weights of LeNet-5 for faster and more robust training before fine-tuning.

## 1 Introduction

Machine learning methods fall into two groups: supervised and unsupervised. In supervised learning, some discriminative task is replicated through learning, based on the data which includes individual class labels. In the case of convolutional neural networks, features are learned that support the supervised task. In unsupervised learning, a representation task is performed to discover the hidden trends in the data, without access to class labels. Representation performance is based on the ability to eliminate redundant variables. In competitive tasks, supervised learning has a lower absolute asymptotic error, but unsupervised learning will approach its lowest asymptotic error more quickly [10].

Many unsupervised methods use input reconstruction to produce good representations which eliminate redundant variables and preserve useful ones. Some methods stochastically reconstruct the input from the representation to approximate density, while others apply desirable constraints to the representation like low dimensionality or sparsity. These representations are used as more suitable input to supervised learning. Principal component analysis and autoencoders are two examples of this unsupervised structure [11].

Basic autoencoders convert high-dimensional data to low-dimensional data using a multilayer neural network with a small middle layer. The encoder network performs the reduction to a lower dimensional representation, while the decoder network reconstructs the input from the representation. Backpropagation is used to train these networks, minimizing the error between the original input and the reconstruction. Each layer can be thought of as features that capture correlations between elements of the prior layer; this produces low dimensional condensed features [4].

Constraints other than low dimensionality are also desirable for the representation. One such constraint is robustness to corruptions in the input, since a good representation should capture stable structure. In the denoising autoencoder, training is performed using corrupted inputs. These corrupted inputs are passed to the encoder to create a representation, and then the decoder recreates the original input. This corruption can be masking, Gaussian noise, salt-and-pepper noise, or another method [14]. In this project, a denoising autoencoder is implemented using various corruption functions, activation functions, loss functions, and learning approaches, and then used to denoise the CIFAR10 dataset before classification by LeNet-5. However, denoising is not the real purpose of this autoencoder model.

Instead, a denoising autoencoder can be used to produce representations of the data suitable for input into a supervised model, similar to dimensionality reduction processes. These representations capture interesting features robust to small invariances of the input, and are more suited for future supervised learning tasks, based on the hypothesis that features of $X$ that help to capture $P(X)$

also help to capture $P(Y|X)$. Although these representations have a higher lowest error compared to supervised learning, they improve training speed. In this project, this comparison is made using the features learned by LeNet-5 and a denoising autoencoder applied to a support vector machine. DAE paper

This similarity between unsupervised and supervised learning in the task of finding weights to produce features means that the unsupervised autoencoder can be used to initialize the weights of a supervised neural network method nearer to a better local minimum of generalization error than random. This produces much lower valleys for gradient descent to optimize in a discriminative task. These weights are initialized by stacking layers of denoising autoencoders pretained on the unsupervised reconstruction task. Each layer maps to the same number of hidden input and output nodes as each pair of layers in a desired supervised network structure. From this initialization, fine-tuning in the discriminative takes much less time than fully discriminative training [sDAE:2010]. In this project, LeNet-5 is initialized for classification of CIFAR10 using a stacked denoising autoencoder pretrained to denoise CIFAR10 images.

## 2 Related Works

The advanced autoencoder networks today began with work on Restricted Boltzmann Machines. An RBM is a two layer neural network, restricted so that no intra-layer communication takes place. Additionally, instead of deterministic units like sigmoid, stochastic units with a distribution like Gaussian are used. On the forward pass, inputs $x$ are used to make predictions about node activations $a$ based on weights $w$ as $p(a|x;w)$. On the reverse pass, the probability of inputs $x$ given activations $a$ is estimated as $p(x|a;w)$. The Kullback Leibler divergence is used to measure the distance between the reconstructed probability distribution and the ground truth, and the weights are adjusted to minimize reconstruction error, eventually producing weights that come to represent the structure of the input.

RBMs can be stacked to form the layers of Deep Belief Networks. The first layer is trained on the inputs to produce a representation that can be thought of as low level features. These features are then input into the second layer to produce higher level features. Over many layers, deep hierarchical features are learned using only unsupervised training. From there fine tuning training can be applied for individual supervised tasks [5].

A simpler unsupervised method using standard neural networks with deterministic units produces similar performance, and in fact can be shown to be theoretically equivalent to RBMs and DBNs. This autoencoder structure comprised of an encoder to turn inputs into a representation, and a decoder to turn the representations back to approximations of the input could simply learn the identity function, except for constraints put on the representation to produce good results that preserve important internal structure and discard irrelevant variabilities. These constraints include lower dimensionality [4], sparsity [11], and robustness to input corruptions [14]. Each of these autoencoders can be pretrained and stacked in order to initialize the weights of a deep architecture. The most popular of these is the stacked denoising autoencoder [15], which is implemented in this project.

Since the invention of the stacked denoising autoencoder in 2010, there have been multiple improvements and alterations to the representation constraints. The contractive autoencoder obtains robustness analytically rather than stochastically, as in the denoising case. This is done by adding a term to the cost function which penalizes the representations sensitivity to the training data as the magnitude of the first derivative [12]. The variational autoencoder constrains representations to a Gaussian distribution. By modeling the true probability distribution as simple distribution that is easy to evaluate, it can be formulated as an optimization problem [8].

There are multiple version of the variational autoencoder framework, including using an importance sampling strategy to increase the model's flexibility [2], transforming a simple density to a complex one using invertible transformations [6], and generating approximate posterior samples through non-linear mappings of latent inputs [13], but the current state of the art is the Ladder VAE [7]. In VAE, the inference and generative distributions are computed separately, but this creates problems when the generative distribution is highly variable. In LVAE two passes are used: an upward pass that computes the approximate likelihood contributions and a stochastic downward pass that recursively approximates the posterior and generative distributions.

Despite the best efforts in unsupervised autoencoder development, supervised methods, specifically convolutional neural networks, have still produced the most accurate results. Additionally,

this power has come without complicated internal structures or math. This has led to an abundance of interest that has led the development trend towards CNNs rather than autoencoders.

# 3 Results

## 3.1 Data

The data for this report is CIFAR-10, originally collected by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton[9]. It is a set of 32x32 pixel RGB images of ten different classifications: four modes of transportation and six animals. There are 60,000 instances divided into 50,000 training samples, 5000 from each class, and 10,000 testing samples, 1000 from each class. Each of the ten classes is completely mutually exclusive. The highest accuracy reported [1] on the database is 96.53% [3].

## 3.2 Task 1

The denoising autoencoder structure has many hyperparamters to vary in order to achieve the lowest reconstruction error. First, the loss function is chosen between cross entropy loss and mean squared loss. Using an initial structure of 256 components, 10% masking noise, sigmoid activation, gradient descent, a 0.001 learning rate, 3000 epochs, and 25 samples in a batch, no visual discernible difference was observed in the reconstructed results. Mean squared error was chosen for the rest of the results due to its more intuitive calculation and error range.

Three activation functions, sigmoid, hyperbolic tangent, and relu, were evaluated using the same structure as the previous experiment using mean squared error. Relu achieved an accuracy of 0.099 compared to 0.160 from hyperbolic tangent and 0.204 from the sigmoid function. Relu is chosen as the future activation function.

Four optimization methods, gradient descent, adaptive gradients, momentum, and Adam were evaluated using the same structure as the previous experiment with relu as the activation function. Instead of stopping at a given epoch, instead iteration was halted when after 100 epochs the error had decreased by less than 0.00005. The results of this experiment can be seen in table 1. Although all accuracies are comparable, Adam arrived at the accuracy in only 200 epochs, and is chosen as the as the future optimization method.

| Optimization | Gradient Descent | Adaptive Gradients | Momentum | Adam |
|---|---|---|---|---|
| Error | 0.099 | 0.094 | 0.094 | 0.104 |
| Epochs | 3000 | 2500 | 3000 | 200 |

Table 1: Epochs until after 100 epochs the error decreased by less than 0.00005, and the resulting mean squared error for various optimization strategies.

Learning rate was evaluated using the same structure as the previous experiment with Adam as the optimization method, and stopping after 500 epochs. The results are found in table 2. The lowest error result is chosen for future evaluation, at a learning rate of 0.00001.

| Learning Rate | 0.1 | 0.03 | 0.01 | 0.003 | 0.001 | 0.0003 | 0.0001 | 0.00003 | 0.00001 |
|---|---|---|---|---|---|---|---|---|---|
| Error | 0.539 | 0.497 | 0.224 | 0.196 | 0.103 | 0.088 | 0.079 | 0.078 | 0.077 |

Table 2: Mean squared error for various learning rates.

The number of components in the encoding was evaluated using the same structure as the previous experiment with a learning rate of 0.00001. The results are found in table 3. The lowest error result is chosen for future evaluations, using 1024 components at the error limit.

| Components | 8 | 16 | 32 | 64 | 128 | 256 | 512 | 1024 |
|---|---|---|---|---|---|---|---|---|
| Error | 0.163 | 0.144 | 0.125 | 0.107 | 0.091 | 0.077 | 0.070 | 0.069 |

Table 3: Mean squared error for various numbers of components in the encoding layer.

The number of samples per batch was evaluated using the same structure as the previous experiment with 256 components in the encoding and 50 epochs. The results are found in table 4. In this case, weights are updated after each batch, but the entire training dataset is iterated through per epoch. A balance between runtime and accuracy is chosen at a batch size of 16.

| Samples Per Batch | 2 | 4 | 8 | 16 | 32 | 64 | 128 | 256 | 512 |
|---|---|---|---|---|---|---|---|---|---|
| Error | 0.078 | 0.079 | 0.080 | 0.083 | 0.087 | 0.092 | 0.099 | 0.110 | 0.127 |
| Runtime (mm:ss) | 33:22 | 18:44 | 10:26 | 6.58 | 5.55 | 4.54 | 4.37 | 4.15 | 4.04 |

Table 4: Mean squared error and runtime in minutes and seconds for various samples per batch.

Using this optimized model and hyperparamters, the convergence over the number of epochs was investigated, resulting in the graph shown in figure 1. Convergence is achieved around epoch 60.

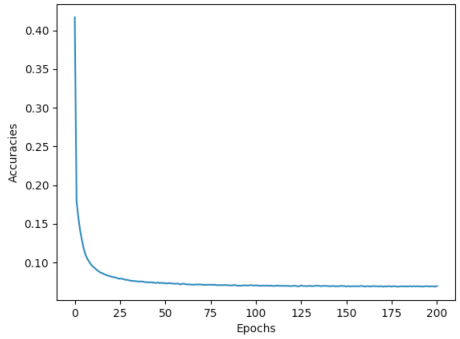


Figure 1: Testing accuracy vs. the current epoch for the optimized model.

Finally, various noise addition patterns were attempted at various strengths. The results can be found in table 5.

| Noise Strength | 0.1 | 0.2 | 0.3 |
|---|---|---|---|
| Masking | 0.073 | 0.128 | 0.193 |
| Salt and Pepper | 0.055 | 0.075 | 0.095 |
| Gaussian | 0.224 | 0.223 | 0.224 |

Table 5: Mean squared error for noise addition strategies and strength percentages.

The lowest error achieved by the autoencoder was 0.055, using 1024 components in the representation, relu activation, a learning rate of 0.0001, adam optimization, 60 epochs, 10% salt and pepper noise, and 16 samples per batch. Figure 2 shows an input test image of a frog and the reconstructed image.

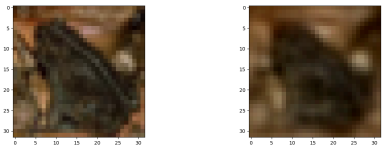

Figure 2: Picture of the original input front image from the test dataset, and the reconstructed version after encoding.

### 3.3 Task 2

The difference in accuracy is compared between LeNet-5 trained and tested on two separate datasets. The first is the original CIFAR10 dataset. In this case, after 2000 epochs an accuracy of 55.1% is achieved. The second dataset is the reconstructed CIFAR10 dataset, encoded and then decoded by a denoising autoencoder with the optimal structure from task 1. This achieved an

accuracy of 25% after 2000 epochs. This result confirms that while the representation is designed for maximum reconstruction accuracy, some data is lost in the process that could be used for classification.

Next, the difference is accuracy is compared between an SVM trained and tested on two separate feature sets. The first are the features extracted by the LeNet-5 convolutions after 2000 epochs and an accuracy of 55.1% as above. In this case, an accuracy of 60.1% is achieved, 5% higher than the fully connected layers of the LeNet-5 architecture. The second are the features encoded by a denoising autoencoder with the optimal structure from task 1. This achieved an accuracy of 49.6%. This observed result confirms that the features extracted for the task of classification are better suited for that task than an unsupervised representation.

### 3.4 Task 3

The difference in training performance between LeNet-5 initialized with random weights and weights generated by an unsupervised stacked denoising autoencoder were compared. The stacked denoising autoencoder used 10% masking corruption, normal weight initialization, a learning rate of 0.000002, 60 epochs, and a mini-batch size of 25 to achieve an error rate of 0.082 for the first layer and 0.023 for the second layer. The normal-initialized training weights achieved an overall accuracy of 55% using a learning rate of 0.00001, a batch size of 50, and 20% dropout. The autoencoder-initialized training weights achieved an overall accuracy of 61% using the same parameters as the optimal LeNet-5. As expected, the autoencoder contributed to a higher overall accuracy. The autoencoder weights also improved more rapidly during initial training. These results are shown in figures 3 and 4.

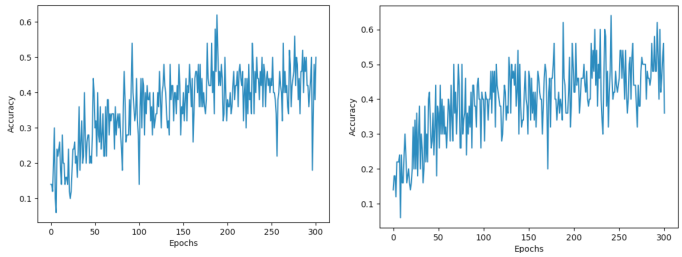


Figure 3: Training accuracy for the first 300 epochs. The normal weight initialization is shown on the left, and the autoencoder weight initialization is shown on the right.

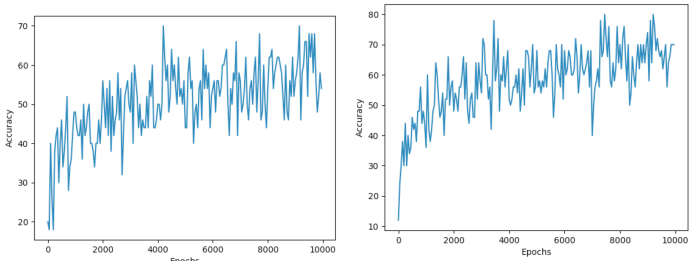


Figure 4: Training accuracy every 50 epochs for 10000 total epochs. The normal weight initialization is shown on the left, and the autoencoder weight initialization is shown on the right.

## 4 Summary

## References

[1] Rodrigo Benenson. who is the best in cifar-10? http://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html#43494641522d3130. Accessed: 2017-10-09.

[2] Yuri Burda, Roger B. Grosse, and Ruslan Salakhutdinov. Importance weighted autoencoders. *CoRR*, abs/1509.00519, 2015.

[3] Benjamin Graham. Fractional max-pooling. *CoRR*, abs/1412.6071, 2014.

[4] G E Hinton and R R Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, July 2006.

[5] Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural Comput.*, 18(7):1527–1554, July 2006.

[6] D. Jimenez Rezende and S. Mohamed. Variational Inference with Normalizing Flows. *ArXiv e-prints*, May 2015.

[7] C. Kaae Sønderby, T. Raiko, L. Maaløe, S. Kaae Sønderby, and O. Winther. Ladder Variational Autoencoders. *ArXiv e-prints*, February 2016.

[8] D. P Kingma and M. Welling. Auto-Encoding Variational Bayes. *ArXiv e-prints*, December 2013.

[9] Alex Krizhevsky. Learning multiple layers of features from tiny images. 2009. https://www.cs.toronto.edu/~kriz/cifar.html.

[10] Andrew Y. Ng and Michael I. Jordan. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems 14*, pages 841–848. MIT Press, 2002.

[11] Marc'Aurelio Ranzato, Y lan Boureau, and Yann L. Cun. Sparse feature learning for deep belief networks. In J. C. Platt, D. Koller, Y. Singer, and S. T. Roweis, editors, *Advances in Neural Information Processing Systems 20*, pages 1185–1192. Curran Associates, Inc., 2008.

[12] Salah Rifai, Pascal Vincent, Xavier Muller, Xavier Glorot, and Yoshua Bengio. Contracting auto-encoders: Explicit invariance during feature extraction. In *In Proceedings of the Twenty-eight International Conference on Machine Learning (ICML'11*, 2011.

[13] D. Tran, R. Ranganath, and D. M. Blei. The Variational Gaussian Process. *ArXiv e-prints*, November 2015.

[14] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pages 1096–1103, New York, NY, USA, 2008. ACM.

[15] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *J. Mach. Learn. Res.*, 11:3371–3408, December 2010.

# 5 Appendix

The LeNet-5 code for this project can be found on GitHub at https://github.com/LambentLight/LeNet-5. The SVM code for this project can be found on GitHub at https://github.com/LambentLight/cnn-features-svm. The denoising autoencoder and stacked denoising autoencoder code for this project can be found on GitHub at https://github.com/LambentLight/denoising_autoencoder. Please contact me with any questions or considerations.