Experiments with Stacked Denoising Autoencoders

Owen Jow OWEN@ENG.UCSD.EDU
University of California, San Diego

Abstract

In this project, I run experiments pertaining to stacked denoising autoencoders (SDAEs). To facilitate comparison, I begin by reproducing several of the results from Vincent et al. (2010). Next, I perform some more independent investigations, related to using denoising as an improved criterion for variational autoencoders, varying the amount of noise applied at different parts of the autoencoders, characterizing the latent/embedding space learned by autoencoders, and training/evaluating denoising autoencoders with more interesting architectures on more challenging datasets than the ones used in the original paper.

1. Introduction

The original papers on denoising autoencoders (Vincent et al., 2008) and stacked denoising autoencoders (Vincent et al., 2010) execute a significant number of experiments in order to characterize their methods. In the following sections, I reproduce a few of those experiments and try running a few more. More specifically, I experiment with denoising criteria for variational autoencoders, trainable/learned noise injection, manifold learning, different and more modern architectures, and different/more challenging datasets.

First, let's start with a refresher in the form of familiar experiments...

2. Experiments from Vincent et al. (2010)

For credibility, I mention here that I was able to reproduce many of the results of the original stacked denoising autoencoders paper (Vincent et al., 2010). As an example, here are the first-layer weights for autoencoders and denoising autoencoders trained on Olshausen's sparse coding dataset and on MNIST. Both map to a 120-dimensional feature space.

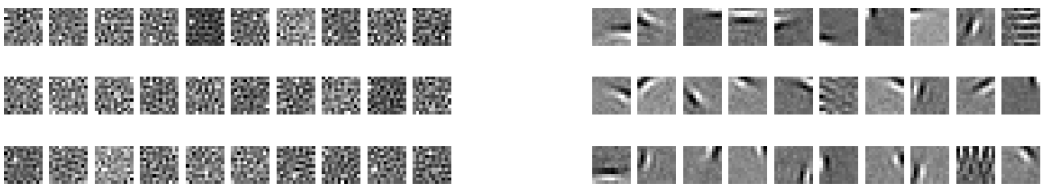

Figure 1: Visualizations of first-layer weights for a non-stacked autoencoder trained on 12×12 patches from the Olshausen dataset. Left: low-structure filters learned by a regular autoencoder (no input noise involved). Right: Gabor-like filters learned by a denoising autoencoder (using additive isotropic Gaussian noise with a standard deviation of 0.4).

Code for this project (incl. instructions to reproduce the results) can be found at `https://github.com/ohjay/sdae`.

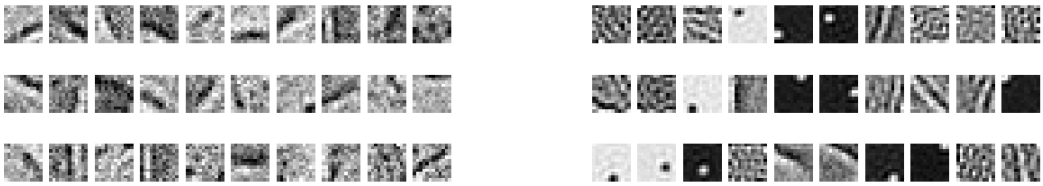

Figure 2: Visualizations of first-layer weights for a non-stacked denoising autoencoder trained on 12×12 patches from the Olshausen dataset with two different types of noise. Left: 25% salt-and-pepper noise. Right: 65% zero masking noise.

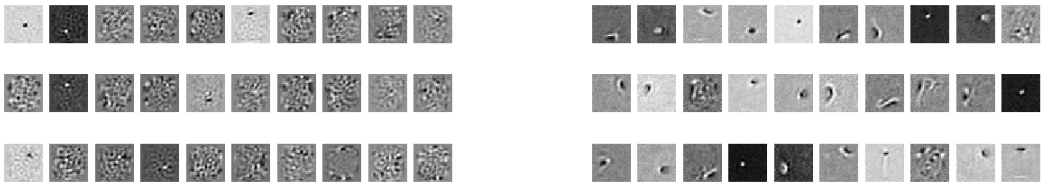

Figure 3: Visualizations of first-layer weights for a non-stacked autoencoder trained on MNIST. Left: filters learned by a regular autoencoder, which lack structure. Right: filters learned by a denoising autoencoder (once again using additive isotropic Gaussian noise with a standard deviation of 0.4), which resemble stroke and blot detectors.

I did find things to be more sensitive to noise type, learning rate, and embedding dimensionality than I had anticipated. For the most part, I seemed to get the cleanest results (first-layer filter-wise) with $\sigma = 0.4$ Gaussian noise. Accordingly, I adopted $\sigma = 0.4$ Gaussian noise as a sort of de facto standard for noise type/level in later parts of this project.

For evaluation of learned representation efficacy, I set up four variants of MNIST to use for classification: MNIST, MNIST `rot` (random rotations), MNIST `bg-rand` (random backgrounds), and MNIST `bg-rand-rot` (you can probably guess). The original paper did not use `MNIST bg-rand-rot`, which I believe would have been the most challenging variant. As a supervised fine-tuning procedure, I attached a (fresh) two-layer dense module to the encoder of (a) a regular autoencoder and (b) a denoising autoencoder and trained the whole thing on each classification task. The resulting test accuracies are reported in Table 1. As was shown in the original paper, the denoising autoencoder outperforms the regular autoencoder on all fronts in terms of learning useful representations for classification.

Figure 4: Examples of images from the MNIST variant datasets. Left: MNIST `rot`. Middle: MNIST `bg-rand`. Right: MNIST `bg-rand-rot`.

Finally, in Figure 5, I show MNIST samples generated using the top-down Bernoulli sampling method described in section 7 of Vincent et al. (2010).

Table 1: Comparison between the downstream classification performance of a pretrained regular autoencoder and a pretrained denoising autoencoder on different MNIST datasets (after the corresponding encoder outputs are each hooked up to a two-layer dense classification module). Naturally, the autoencoders share the same architecture: both are stacked with two encoder and decoder blocks (a *block* being a dense layer and a sigmoid). In all cases, the autoencoder was pretrained on vanilla MNIST. Mainly this was due to time constraints, but I also noticed that the classification performance dropped when the autoencoder was trained on MNIST with random backgrounds, intuitively because it doesn't make sense to try to recover the underlying structure of a background which is just noise. Nevertheless, there are probably some easy performance boosts to be gained from pretraining the autoencoders for `rot` and `bg-rand-rot` classification on MNIST `rot` data, as the data distributions would align more closely between pretraining and supervised fine-tuning.

Dataset	"No-Pretraining" Test Acc.	SAE Test Acc.	SDAE Test Acc.
MNIST	0.9792	0.9807	0.9831
MNIST `rot`	0.9217	0.9243	0.9448
MNIST `bg-rand`	0.8685	0.9001	0.9323
MNIST `bg-rand-rot`	0.7210	0.7953	0.8622

3. Denoising Variational Autoencoder

Variational autoencoders (Kingma and Welling, 2013) are a well-known modern-day variant of autoencoders which have spawned state-of-the-art results in a wide range of application spaces, including sketches (Ha and Eck, 2017) and high-quality image generation (Razavi et al., 2019). Similarly to a standard autoencoder, a variational autoencoder consists of an encoder and a decoder and is trained to reconstruct its inputs. However, it enforces via KL regularization that the latent space reflect a particular prior distribution, mapping the input into the mean and the standard deviation of that multivariate distribution and then sampling from it in a differentiable fashion (by means of the reparameterization trick) to obtain an actual latent vector. Later, latent vectors can be sampled freely from this distribution and decoded, making a VAE an actual generative model.

Normally, variational encoders are trained to reconstruct the inputs without modification. In this case, the noise comes in at the embedding level, when sampling using the reparameterization trick. Building off of the insights of denoising autoencoders, there is a natural question to be asked: is it also beneficial with VAEs to train for a denoising criterion instead of a simple reconstruction one? (VAEs maximize a variational lower bound which consists of a negative KL divergence and a reconstruction accuracy.)

As it happens, this question has already been asked and answered, again by the Université de Montréal (Im et al., 2015). The idea is to create a *denoising* variational autoencoder (DVAE) by training with noise at the input level just like the denoising autoencoders of old, and the verdict is that it is indeed helpful for improving performance contingent on choosing

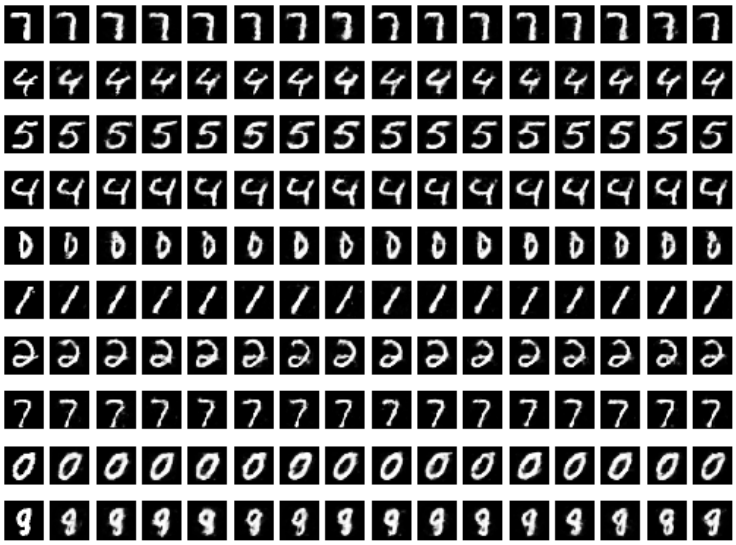

Figure 5: SDAE-generated MNIST samples. The leftmost image in each row is the original; all of the images to the right of it are corresponding randomly-generated samples. For the images of fours, the decoder removes the extra part jutting out to the right. It also tends to straighten out the top of the five. On the bottom it goes back and forth between an eight and a nine, presumably due to the "leg" of the original adding some ambiguity.

a good noise distribution. I corroborate this notion by training my own DVAE on MNIST `bg-rand-rot` and showing in Table 5 that the denoising VAE provides improvement over the vanilla VAE. I also train a two-level SDVAE (which is my term for a stacked version of a DVAE) in the same layer-by-layer fashion as described in Vincent et al. (2010), and find (again in Table 5) that stacked DVAE does about the same as DVAE (maybe slightly better; admittedly I did not perform much of an architecture or hyperparameter search).

Note: the DVAE's variational lower bound, maximized w/r/t θ and ϕ, is

$$\begin{aligned}\mathbb{E}_{\tilde{q}_\phi(\mathbf{z}|\mathbf{x})}\left[\log \frac{p_\theta(\mathbf{x},\mathbf{z})}{q_\phi(\mathbf{z}|\tilde{\mathbf{x}})}\right] &= \mathbb{E}_{\tilde{q}_\phi(\mathbf{z}|\mathbf{x})}\left[\log p_\theta(\mathbf{x}|\mathbf{z})\right] - \mathbb{E}_{\tilde{q}_\phi(\mathbf{z}|\mathbf{x})}\left[\frac{p(\mathbf{z})}{q_\phi(\mathbf{z}|\tilde{\mathbf{x}})}\right] \\ &= \mathbb{E}_{\tilde{q}_\phi(\mathbf{z}|\mathbf{x})}\left[\log p_\theta(\mathbf{x}|\mathbf{z})\right] - \mathbb{KL}(q_\phi(\mathbf{z}|\tilde{\mathbf{x}})||p(\mathbf{z}))\end{aligned}$$

where $p_\theta(\mathbf{x}|\mathbf{z})$ is represented by the decoder (generative) network, $q_\phi(\mathbf{z}|\tilde{\mathbf{x}})$ is represented by the encoder (inference) network, and $p(\mathbf{z})$ is the prior over $\mathbf{z}$.

Table 2: Comparison between the downstream classification performance of a pretrained regular VAE and a pretrained denoising VAE on MNIST `bg-rand-rot`. I use the latent mean vector as the encoded representation. SDVAE refers to a stacked denoising variational autoencoder, which is trained layer-wise just like a stacked denoising autoencoder.

Dataset	VAE Test Acc.	DVAE Test Acc.	SDVAE Test Acc.
MNIST `bg-rand-rot`	0.7482	0.7943	0.7962

Finally, in Figure 6 I show a comparison of generated samples using a VAE and DVAE by varying the first two dimensions of a 20D latent space (keeping the other 18 dimensions fixed, after an initial random sampling for each of them).

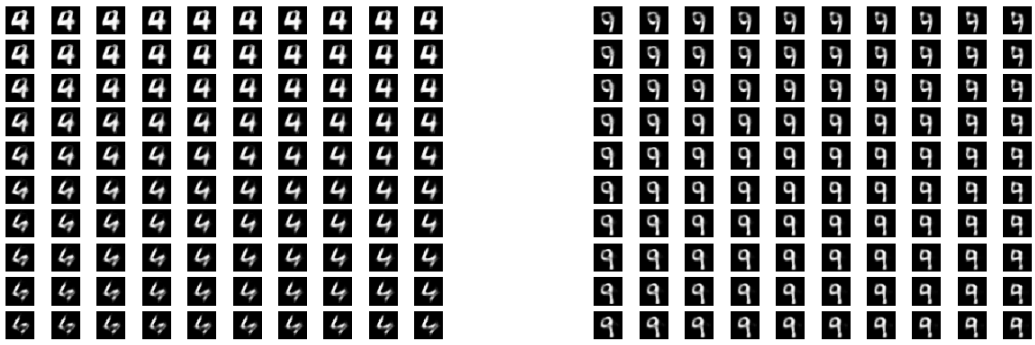

Figure 6: Comparison of generated samples using a VAE (left) and a DVAE (right). I didn't notice any meaningful differences between VAEs and DVAEs in terms of sample generation.

4. Noise Transformers

Vincent et al. (2010) mentions in its "future work" section the possibility of having the corruption process be "parameterized and learnt directly from the data," which would avoid having to hardcode or hand-engineer the noise distributions. I decided to give this a try. Inspired by spatial transformer networks (Jaderberg et al., 2015), I created a noise transformer module to add noise to the input of each layer of the stacked denoising autoencoder, and to learn the best corruption process to apply based on the data. The noise transformer is a simple fully-connected network which predicts a global Gaussian standard deviation σ to use during noise generation for the entire input. It applies the noise using the same process as the reparameterization trick for VAEs, with an external ϵ map serving as the source of randomness in order for the actual noise prediction to remain differentiable.

At first, I tried predicting a noise value for each component of the input, but found that the noise transformer was learning to cut corners by applying heavy noise to the background and leaving the middle part (i.e. the actual number part of an MNIST image) relatively untouched.

The modification to the loss function is simple, and is motivated by the idea of providing a reward for denoising at a higher noise level:

$$\texttt{loss} = \texttt{reconstructionLoss} - \lambda \cdot \texttt{mean}(\sigma)$$

where λ is a hyperparameter meant to control the balance between emphasizing accuracy and difficulty. Note that this loss function is only used to optimize the noise transformer. The denoising autoencoder uses the same objective as before. The hope is that the noise transformer serves as a sort of "coach" which wants to see the denoising autoencoder succeed at reconstruction, but also wants to keep things difficult.

Unfortunately, I found the noise transformer exceedingly difficult to train. Unless the value of λ was chosen very precisely via grid search, the predicted global σ would saturate at the extreme values (0 or 1 assuming a sigmoid activation function). Usually σ would go to 0 in order to make life as easy as possible for the autoencoder. In order to get around this, I decided to clamp the minimum value of the noise to a certain value (usually 0.4) and choose λ as best I could. The result was that the denoising autoencoder would at worst end up with the noise level it was already using (0.4), but mostly the noise transformer would try to squeeze a little more noise into the input wherever it could. I think of it as a kind of fine-tuning process, or a way to avoid some hyperparameter tuning once a baseline noise level has been chosen – the motivation being that it tends to be beneficial to denoise at a higher noise level if possible.

In short, I roughly determined a lower bound for the noise as 0.4, and then let the noise transformer amp it up as appropriate at each level. The results were some slightly improved performance on MNIST `bg-rand` classification.

Table 3: Fixed Gaussian $\sigma = 0.4$ noise vs. learned, per-input Gaussian σ noise. A better comparison might be to use a fixed Gaussian whose σ is equal to the mean of the learned Gaussian σ across the dataset.

Dataset	SDAE ($\sigma = 0.4$) Test Acc.	SDAE (Learned σ) Test Acc.
MNIST `bg-rand`	0.9323	0.9390

5. Manifold Learning

5.1 t-SNE Visualization of Latent Space

We can use t-SNE (van der Maaten and Hinton, 2008) to visualize the 1000D latent space such that distances between points in the high-dimensional space are represented in 2D. This can serve as a justification or reinforcement of the usefulness of denoising autoencoders in producing learned representations for classification. The t-SNE plots for MNIST embeddings using either (a) a regular autoencoder or (b) a denoising autoencoder are shown in Figure 7. In order to generate the points in embedded space, I pass examples of each class through the stacked encoder.

The plots seem to be consistent with my expectations about the similarities/distances between different digits. For example, 4 and 9 are somewhat confounded because their written forms often resemble each other. 3, 5, and 8 are also close together.

But the big takeaway is that the denoising autoencoder appears to cluster and separate the images for different classes a little more neatly relative to the regular, non-denoising autoencoder. This can obviously lead to an easier time during a later classification phase.

5.2 Toy Manifold Recovery Experiment

I also attempted a toy experiment in order to see if the denoising autoencoder could learn to express a simple 1D data manifold in terms of a low-dimensional embedding. To create a 1D data manifold, I simply interpolated between two images, one a dandelion and the

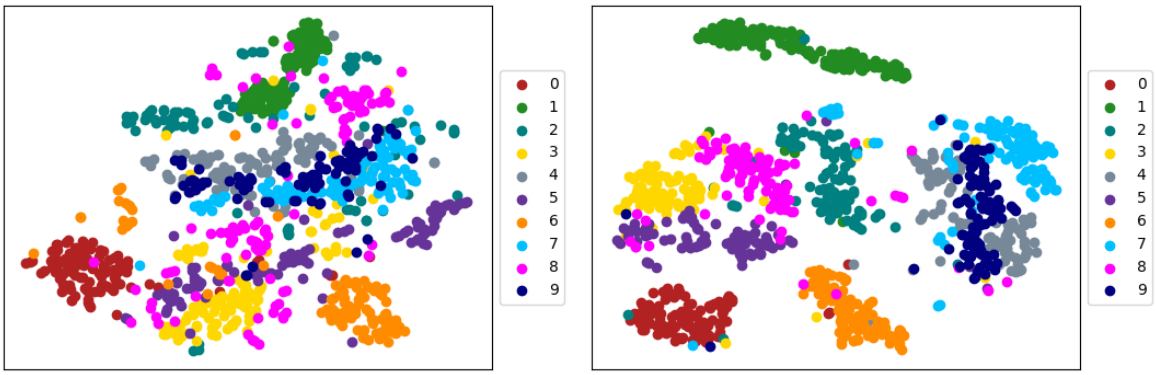

Figure 7: t-SNE visualization of the MNIST latent space for a regular autoencoder (left) and a denoising autoencoder (right). The two autoencoders share the same architecture; the only difference is that one was trained using the denoising criterion and the other was not. Empirically, the denoising autoencoder's latent space seems to be better organized with respect to maximizing inter-class variance, signifying that the latent space is metrically well-conditioned for the downstream classification problem.

other its black background (see Figure 8), at various steps. Effectively I created a dataset of 6400 images of the dandelion at different brightness states.

Basically I was wondering if an autoencoder which mapped to a 1D space would be able to model the 1D distribution of the data similarly to a VAE. However, I found that there was almost no expressivity with a 1D embedded space – all of the samples generated over a linear space ended up looking pretty much the same. So I increased the embedding dimensionality to 2D and managed to generate some better-varying samples over that space. The generated samples are shown in Figure 8.

I think this shows that the autoencoder can barely start to model a simple latent distribution on its own. But it's not very good at it; overall expressivity is low.

6. Convolutional Autoencoders

The original stacked denoising autoencoders paper used dense layers. However, in the time since, there's been a trend toward convolutional architectures, especially where images or other volumetric data are concerned. In this section, I try using convolutional stacked denoising autoencoders which are, architecturally speaking, lightweight renditions of modern-day architectures. In other words, they follow an encoder-decoder scheme like the very popular U-Net (Ronneberger et al., 2015) except with a lot less layers (eight in total) and no skip connections so that the encoder and decoder can be used independently of each other. Following the notion that transposed convolutions can lead to checkerboard artifacts (Odena et al., 2016), I try to use upsampling and convolutions instead of transposed convolutions when possible.

I record the results of my experiments with convolutional autoencoders in Table 5. Note: I refer to convolutional (stacked) autoencoders as CAEs and convolutional (stacked) denoising autoencoders as CDAEs.

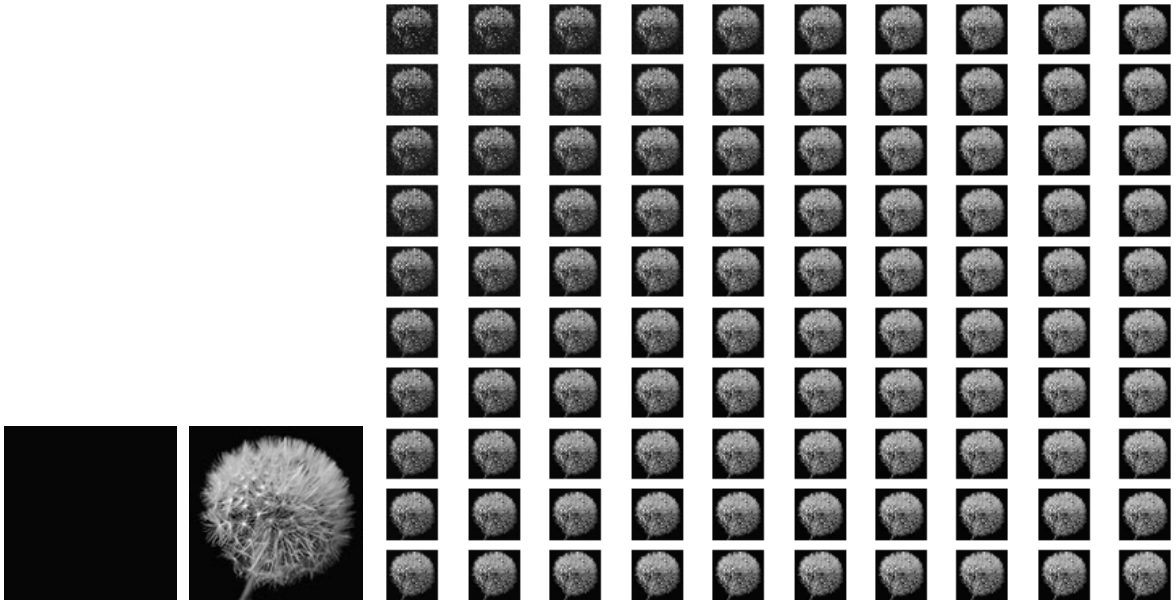

Figure 8: Samples generated from interpolated 2D latent vectors, over an artificially-created data distribution which lies on a 1D manifold between two points in high-dimensional space (left). The autoencoder did not seem to be able to recover the manifold very expressively in terms of the 2D latent space (nor did it fare any better with a 1D latent space, which I also tried), perhaps because the latent space isn't required to resemble any prior distribution as with VAEs. However, there is still some semblance of smoothness: the decoded dandelion images largely increase in brightness from the top left to the middle or bottom right.

Table 4: The benefit of convolutional autoencoders.

Dataset	SDAE Test Acc.	CAE Test Acc.	CDAE Test Acc.
MNIST	0.9831	0.9925	0.9930
MNIST `bg-rand`	0.9323	0.9764	0.9807
MNIST `bg-rand-rot`	0.8622	0.9271	0.9483

As you can see, the convolutional autoencoders improve upon the dense autoencoders on all fronts. An intuitive explanation for the increased performance is that you don't need to look at the entire image in order to denoise; rather, you can do it based on some receptive field around each point in question. This is how many state-of-the-art denoising methods perform denoising, e.g. (Bako et al., 2017) with its per-pixel filtering kernels. Also, one motivation for masking noise is that the autoencoder has to learn to fill in missing values based on their neighbors and thus learn correlations between different structures in the data. Again, you only need to look at some neighborhood in order to do this. All of this points to convolution as being a more appropriate and efficient way to extract structure and meaningful relationships from the data, at least in this context.

7. Other Datasets

There are a lot more datasets now than there were in 2010. I tried using some more of them, e.g. CUB-200-2011 and CIFAR-10. The former contains approximately 10,000 images of birds (200 classes, ∼5000 training images) of varying resolutions which I cropped (according to the provided bounding boxes) and resized so that they were 128×128. The latter is like MNIST, except with 32×32 color images and more variety in the 10 object classes and 60,000 images. Both can be used for classification purposes similarly to MNIST.

Figure 9: Examples of images from the CUB-200-2011 (left) and CIFAR-10 (right) datasets.

Once again, I observe an increase in performance from using denoising autoencoders to provide embeddings for classification on these datasets. These accuracies are far from state of the art, but in fairness I am using very simplistic network architectures and haven't experimented much with the hyperparameters.

Table 5: Performance of convolutional autoencoders vs. convolutional denoising autoencoders in terms of representation learning for more challenging (/color) datasets.

Dataset	CAE Test Acc.	CDAE Test Acc.
CUB-200-2011	0.2193	0.2381
CIFAR-10	0.6395	0.6504

8. Other Avenues of Exploration

There were a lot of things I didn't get around to. I'll mention them here.

- A comparison to true modern, ultra-deep encoder-decoder architectures, especially for higher-resolution images like CUB-200-2011. Currently those reconstructions are pretty blurry.
- Investigate whether there's a place and a use for a denoising criterion in some very new work like VQ-VAE-2 (Razavi et al., 2019) which was released in June and uses VAEs with discrete codes and a learnt prior to generate impressively high-quality, high-resolution images.
- An empirical geometric/topological analysis in the vein of Fawzi et al. (2018) which further explores manifold learning and evaluates the connectedness of the latent space for embedded members of the same class. There might be room to investigate whether

the encoder maps natural inputs to a connected space as well. For example, if you interpolate between encoded natural images in embedded space, and decode each of them, do the decoded results also look natural? This is similar to what I was trying to determine in the toy manifold recovery experiment.

- This has likely been done and published, but a comparison between using denoising as an autoencoding criterion vs., say, colorization for learning good representations.
- Varying the noise spatially over the input based on variance. High-variance regions would receive more noise because it's more important to recover the structure there. Although I have doubts that this is any better than just using the "more noise" level everywhere in the first place.
- Increasing or decreasing the noise level of the course of the network, or over time. One motivation for this could be curriculum learning, where it usually helps during training to gradually ramp up difficulty. It might also be useful to use different types of noise at different layers; one could perform an empirical study of this.
- Evaluate the convolutional autoencoder on tasks other than classification. There are certain tasks for which a volumetric feature representation (or one that retains spatial dimensions) is better suited, like segmentation or optical flow. How do the autoencoder's embeddings fare for such applications? How does it generalize in terms of usefulness across different applications?
- What if you use a modern technique which is supposed to be better for *denoising*, like kernel prediction? Does better denoising equate to learning better representations?
- Evaluation of the need for layer-wise pretraining, as opposed to training the entire stacked encoder and the entire stacked decoder at once. How much does it help?
- Comparison of stacked denoising autoencoders versus autoencoders with dropout.
- Evaluation of the effect of the embedding size.
- More structured generalization experiments, in terms of the encoder providing good embeddings. What if you train on data from one distribution, and use it on data from a slightly different distribution? How does it do? This is similar to the notion of training on one MNIST variant and using that encoder for all of the others as well.

9. Conclusion

Note: I have open-sourced my code, and aim to offer reproducibility for all results mentioned in this report. Currently, there are 50+ experiment files (i.e. executable scripts) saved in the `experiments` folder at `https://github.com/ohjay/sdae`. They should be good enough to produce pretty much all of the results described, although there might be some slight hyperparameter tweaks necessary (like reducing the learning rate).

To summarize: I have experienced the strengths of stacked denoising autoencoders in various contexts (for variational autoencoders, convolutional architectures, and datasets

other than MNIST). I have evaluated the metric quality of SDAEs' latent spaces, visualizing embeddings in 2D with t-SNE and observing a reasonable clustering of embedded images for different digits (even without supervision), which is slightly better distributed with denoising autoencoders than with non-denoising autoencoders. I have attempted to train network modules at each layer to apply noise to their inputs in an optimized way, and I have attempted to recover a simple manifold through the latent space in an experiment that didn't quite give me the results I hoped for. I have also sat in my chair and stared at a screen for a great many hours and now it is finally time to sleep. Thank you for reading!

References

Steve Bako, Thijs Vogels, Brian McWilliams, Mark Meyer, Jan Novák, Alex Harvill, Pradeep Sen, Tony DeRose, and Fabrice Rousselle. Kernel-predicting convolutional networks for denoising monte carlo renderings. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2017)*, 36(4):97:1–97:14, 2017. doi: 10.1145/3072959.3073708.

Alhussein Fawzi, Seyed-Mohsen Moosavi-Dezfooli, Pascal Frossard, and Stefano Soatto. Empirical study of the topology and geometry of deep networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

David Ha and Douglas Eck. A neural representation of sketch drawings. *CoRR*, abs/1704.03477, 2017. URL `http://arxiv.org/abs/1704.03477`.

Daniel Jiwoong Im, Sungjin Ahn, Roland Memisevic, and Yoshua Bengio. Denoising criterion for variational auto-encoding framework. *CoRR*, abs/1511.06406, 2015. URL `http://arxiv.org/abs/1511.06406`.

Max Jaderberg, Karen Simonyan, Andrew Zisserman, and Koray Kavukcuoglu. Spatial transformer networks. *CoRR*, abs/1506.02025, 2015. URL `http://arxiv.org/abs/1506.02025`.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2013. URL `http://arxiv.org/abs/1312.6114`. cite arxiv:1312.6114.

Augustus Odena, Vincent Dumoulin, and Chris Olah. Deconvolution and checkerboard artifacts. *Distill*, 2016. doi: 10.23915/distill.00003. URL `http://distill.pub/2016/deconv-checkerboard`.

Ali Razavi, Aaron van den Oord, and Oriol Vinyals. Generating diverse high-fidelity images with vq-vae-2, 2019.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597, 2015. URL `http://arxiv.org/abs/1505.04597`.

Aäron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. Neural discrete representation learning. *CoRR*, abs/1711.00937, 2017. URL `http://arxiv.org/abs/1711.00937`.

Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605, 2008. URL `http://www.jmlr.org/papers/v9/vandermaaten08a.html`.

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. ISBN 978-1-60558-205-4. doi: 10.1145/1390156.1390294. URL `http://doi.acm.org/10.1145/1390156.1390294`.

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.

Thijs Vogels, Fabrice Rousselle, Brian McWilliams, Gerhard Röthlin, Alex Harvill, David Adler, Mark Meyer, and Jan Novák. Denoising with kernel prediction and asymmetric loss functions. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2018)*, 37(4): 124:1–124:15, 2018. doi: 10.1145/3197517.3201388.