Learning Feature Representations

Module 1 Homework

Oscar Carlsson
Jimmy Aronsson

November 30, 2020

Exercise 1

In this first exercise, we attempt to model the (unknown) distribution of MNIST images, $x \sim p_d$, and hopefully generate synthetic images that look realistic. In more detail, we start by removing the average MNIST image $\mu \in \mathbb{R}^{28\times 28}$ from each training image x, and we then explore whether the remaining *MNIST noise* data $x - \mu$ can be modeled using a multivariate Gaussian $\mathcal{N}(\mathbf{0}, \Sigma_\theta)$ with a learned precision matrix $\Lambda_\theta = \Sigma_\theta^{-1}$. One could then create synthetic images that resemble real MNIST images. Two different methods have been considered for estimating Λ_θ:

- Noise-contrastive estimation (NCE),
- Score matching (SM).

Seeing as the underlying ideas and general analysis of these methods have already been discussed by Christopher Zach in his `presentation_part1`, we gloss over the introduction of each method and focus on those additional, problem-specific details which are not featured in the presentation.

Before jumping into the analysis and numerics, however, we must admit that we have not achieved good numerical results in this assignment. First and foremost, we suffered problems with infinite gradients when training NCE, and we never managed to solve this problem. Score matching did work and our sampled images do have digit-like patterns, however, they are not very convincing. These problems continued in Exercise 2 where ZCA whitening produced apparent noise, even when using the empirical covariance matrix, and the learned filters v_j do not have clear features.

In summary, our numerical results are quite disappointing - even more so considering how much time we have spent on this assignment. We have almost certainly made some bad design choices, bad initializations, and perhaps we misunderstood some parts of the theory. That being said, it has genuinely been fun and we have learned a whole lot from the process.

NCE

As explained in the presentation by Zach, noise-contrastive estimation (NCE) casts the estimation of distribution parameters as a supervised learning problem. This effectively means teaching the model distribution p_θ to distinguish between real data $x \sim p_d$ and noise data $x' \sim p_n$, where the noise distribution should be similar enough to the data distribution for this classification problem to be challenging; we want the model distribution p_θ to learn the most essential properties of p_d.

For the noise distribution, we chose a multivariate Gaussian $\mathcal{N}(\mathbf{0}, \Sigma_n)$ whose covariance matrix Σ_n coincides with the empirical covariance matrix for the MNIST noise data.

We then construct a data set by flipping a weighted coin $z \sim \text{Bern}(1-\eta)$ multiple times and letting each result $z_i \in \{0,1\}$ decide whether to sample x_i from the data distribution ($z_i = 1$) or from the noise distribution ($z_i = 0$). The resulting training examples are denoted by $x_1, \ldots, x_N$ and the noise samples by $x'_1, \ldots, x'_M$, so that $M + N$ is the total number of coin flips. In NCE, we use these samples to minimize the loss function

$$J(\theta) \propto \mathbb{E}_{x \sim p_d}\left[\log \frac{p_\theta(x)}{p_\theta(x) + \nu p_n(x)}\right] + \nu \mathbb{E}_{x \sim p_n}\left[\log \frac{\nu p_n(x)}{p_\theta(x) + \nu p_n(x)}\right], \tag{1}$$

where $\eta = \frac{1}{1+\nu}$. We approximate the right-hand side of equation (1) using the empirical estimate

$$\frac{1}{N}\sum_{i=1}^{N} \log \frac{p_\theta(x_i)}{p_\theta(x_i) + \nu p_n(x_i)} + \frac{\nu}{M}\sum_{j=1}^{M} \log \frac{\nu p_n(x'_j)}{p_\theta(x'_j) + \nu p_n(x'_j)},$$

which we simplify by rewriting both terms in the following way:

$$\begin{aligned} \log \frac{p_\theta(x)}{p_\theta(x) + \nu p_n(x)} &= -\log\left(1 + \nu \frac{p_n(x)}{p_\theta(x)}\right), \\ \log \frac{\nu p_n(x)}{p_\theta(x) + \nu p_n(x)} &= -\log\left(1 + \frac{1}{\nu}\frac{p_\theta(x)}{p_n(x)}\right). \end{aligned}$$

If we now insert the relative probability

$$w(x) = \frac{p_n(x)}{p_\theta(x)} = \sqrt{\frac{|\Lambda_n|}{|\Lambda_\theta|}} \exp\left(-\frac{1}{2} x^T (\Lambda_n - \Lambda_\theta)\, x\right),$$

then we obtain the relatively simple expression

$$J(\theta) \underset{\sim}{\propto} -\frac{1}{N}\sum_{i=1}^{N} \log(\nu w(x_i) + 1) - \frac{\nu}{M}\sum_{j=1}^{M} \log\left(\frac{1}{\nu w(x'_j)} + 1\right). \tag{2}$$

We found that $w(x)$ is typically very small in practice, hence the sum $(\nu w)^{-1} + 1$ is dominated by its first term. Its logarithm can thus be approximated by the numerically more stable expression

$$\log\left(\frac{1}{\nu w(x)} + 1\right) \approx -\log \nu w(x) = \frac{1}{2} x^T (\Lambda_n - \Lambda_\theta) x - \frac{1}{2}\log\left(\nu^2 \frac{|\Lambda_n|}{|\Lambda_\theta|}\right).$$

It would also be possible to remove the first sum in equation (2), since $\log(\nu w(x) + 1) \approx \log 1$. We decided to keep it, however, because it didn't cause computational problems and we didn't want our estimate to be independent of the real training data. Thus, our final estimate is

$$\boxed{J(\theta) \underset{\sim}{\propto} -\frac{\nu}{2}\log\left(\nu^2 \frac{|\Lambda_n|}{|\Lambda_\theta|}\right) - \frac{1}{N}\sum_{i=1}^{N} \log(\nu w(x_i) + 1) + \frac{\nu}{2M}\sum_{j=1}^{M} {x'_j}^T (\Lambda_n - \Lambda_\theta) x'_j}$$

We also obtained an expression for the gradient $\nabla J(\theta)$ in terms of the precision matrix Λ_θ, though we found this expression rather bulky and difficult to handle. Instead, we used `tf.GradientTape` to compute the gradient and update Λ_θ. Three approaches were considered to keep Λ_θ symmetric positive definite and retain its sparse 4-/8-connected structure after each epoch:

1. Writing the precision matrix as $\Lambda_\theta = (A_\theta^T A_\theta) \cdot M$ for a learned matrix A_θ and a predefined masking matrix $M \in \{0,1\}^{28\times 28}$ that is applied element-wise, enforcing the neighbourhood structure by killing undesired matrix elements.

   The matrix product $A_\theta^T A_\theta$ is guaranteed to be symmetric positive definite whenever A_θ is invertible, which any square matrix almost surely is. Combined with the fact that element-wise products of positive definite matrices is again positive definite, we hoped this would prove that Λ_θ is symmetric positive definite. Unfortunately, we eventually realized that our masking matrix is not positive definite, so we cannot guarantee that Λ_θ is, either.

2. Forcing a symmetric gradient by throwing away its lower triangular part and replacing it with the transpose of its upper triangular part. We also apply the aforementioned masking matrix M to force the neighbourhood structure on the gradient. This ensures that Λ_θ is symmetric and retains its neighbourhood structure for all epochs. On the other hand, we still cannot guarantee that Λ_θ remains positive definite.

3. Using the eigendecomposition $\Lambda_\theta = U^T DU$ or, alternatively, $\Lambda_\theta = U^T DU \cdot M$, where the diagonal matrix D contains the eigenvalues of Λ_θ. This would produce a symmetric matrix that can be kept positive definite by changing the values of negative eigenvalues.

None of these approaches worked well in practice. As explained above, we encountered infinite gradients - a problem we were unable to solve. We suspect the problem is caused by $w(x)$ taking such extreme values, which is why we tried approximating the second logarithm in equation (2) to avoid computing a large exponential, but to no avail. The extremity of $w(x)$ also causes the loss function to be essentially independent of the MNIST data - a serious problem on its own.

A natural solution to both of these problems seems to be: Improving the initialization of both precision matrices Λ_θ and Λ_n. We tried this, and we also tried scaling down both determinants $|\Lambda_\theta|$ and $|\Lambda_n|$ by the same constant factor to improve numeric stability, but the problem persisted.

We recognize that we may have fallen into the trap discussed in `presentation_part1`, slide 51: That we want to model properties of p_d, not of p_n, and that it's easy to have a "good" solution if p_θ simply detects features in noise. Indeed, we are explicitly warned against expressions of the form

$$\log p_\theta(x) = -\sum_k \log(1 + \exp(\cdots)),$$

which is precisely the kind of expressions we obtain in equation (2). We did attempt to naïvely change the sign in $J(\theta)$ to see what would happen, but the problem prevailed.

Score Matching

When given the choice between cNCE and score matching, we figured the latter would be more interesting, it being a fundamentally different approach than NCE. Fortunately, score matching also turned out to be easy to implement because the relevant analysis had already been excellently performed in the presentation. It allowed us to more or less directly implement the loss function

$$J(\mu, \Lambda_\theta) = \int \frac{1}{2}\|\nabla_x \log p_\theta(x)\|^2 + \Delta \log p_\theta(x) \approx \frac{1}{2N}\sum_{i=1}^{N} \|\Lambda_\theta(x_i - \mu)\|^2 - \operatorname{tr}(\Lambda_\theta),$$

and start training. Gradients were again computed with `tf.GradientTape`, and the symmetry and neighbourhood structure was enforced by manipulating the gradient as in NCE approach 2.

To get a better idea of how well our learned distribution p_θ approximates the data distribution p_d, we have created synthetic images in two different ways: (1) Sampling straight from a multivariate Gaussian $\mathcal{N}(\mu, \Sigma_\theta)$ using the empirical mean μ and the learned covariance matrix $\Sigma_\theta = \Lambda_\theta^{-1}$. (2) Following the instructions in the assignment by setting

$$x \leftarrow \mu + \Sigma_\theta^{1/2} \varepsilon,$$

where $\varepsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. Combined with the choice of 4-/8-connected structure, this makes 4 different kinds of samples, which are shown in Figures 1-4. An immediate observation is that 8-connected structure seems to produce better samples, which is reasonable considering it is more general than 4-connected structure and can better approximate the data distribution. That being said, while many samples contain digit-like shapes, they do not resemble MNIST images.

Figures 5-7 show the empirical and learned precision matrices with 4-/8-connected structure; Figure 9 illustrates losses.

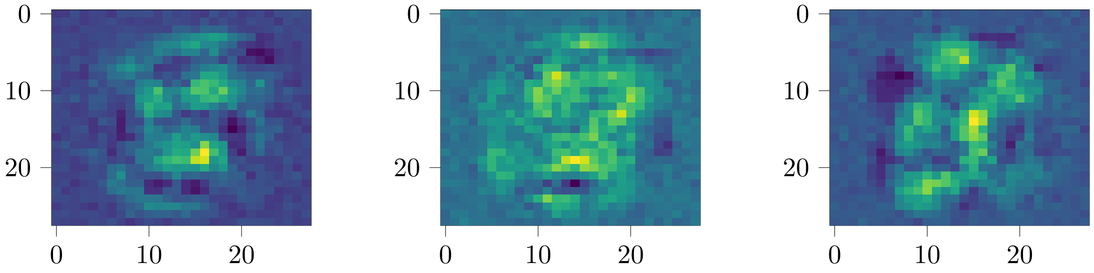


Figure 1: Samples drawn from $\mathcal{N}(\mu, \Sigma_\theta)$ with 4-connected precision matrix (SM).

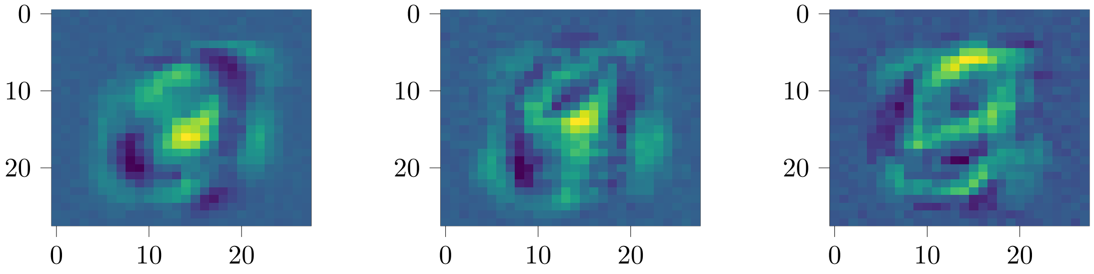


Figure 2: Samples drawn from $\mathcal{N}(\mu, \Sigma_\theta)$ with 8-connected precision matrix (SM).

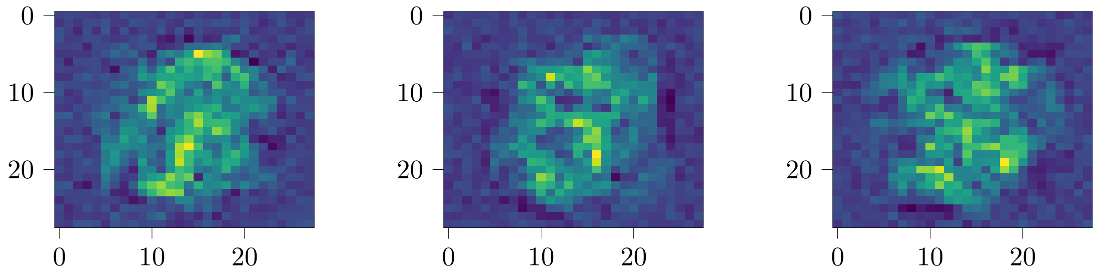


Figure 3: Samples $x \leftarrow \mu + \Sigma_{\theta}^{1/2}\varepsilon$ with $\varepsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and 4-connected precision matrix (SM).

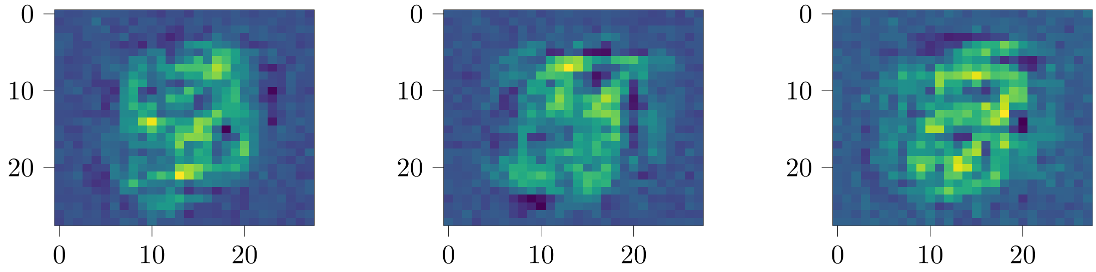
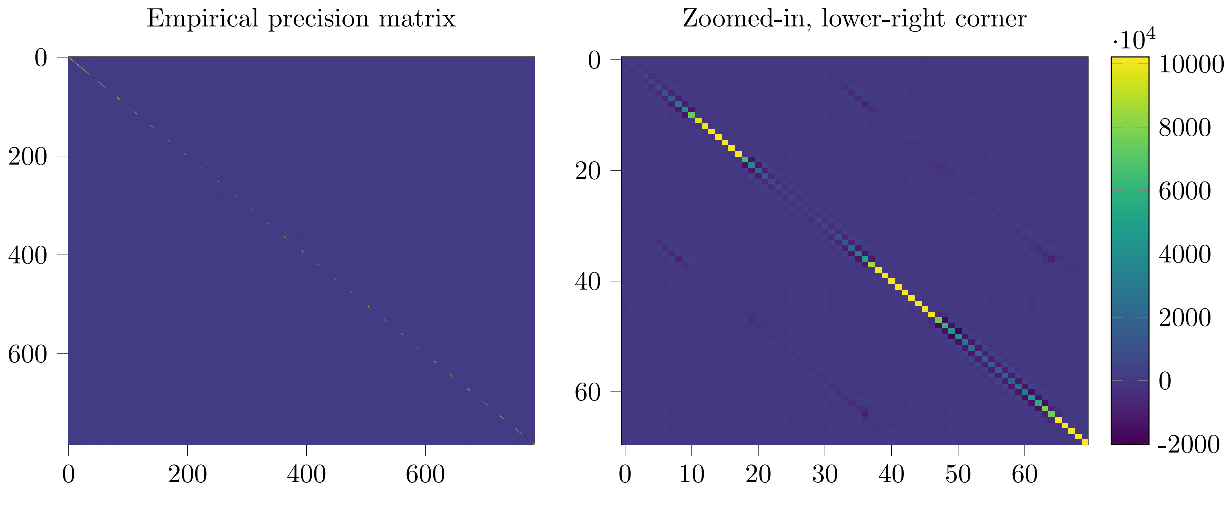


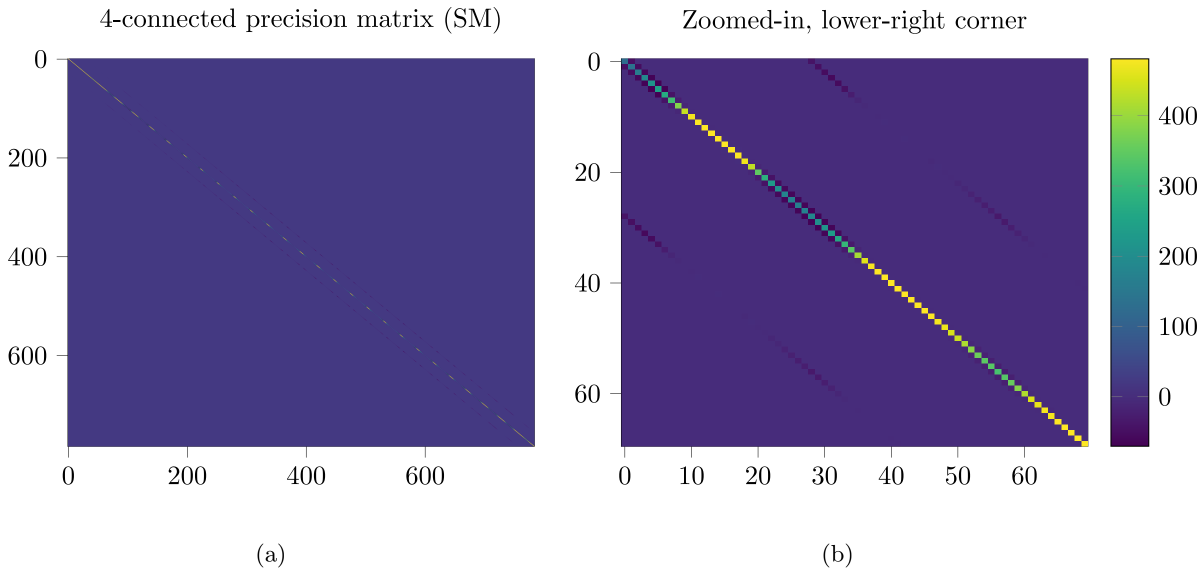
Figure 4: Samples $x \leftarrow \mu + \Sigma_{\theta}^{1/2}\varepsilon$ with $\varepsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and 8-connected precision matrix (SM).



(a)

(b)

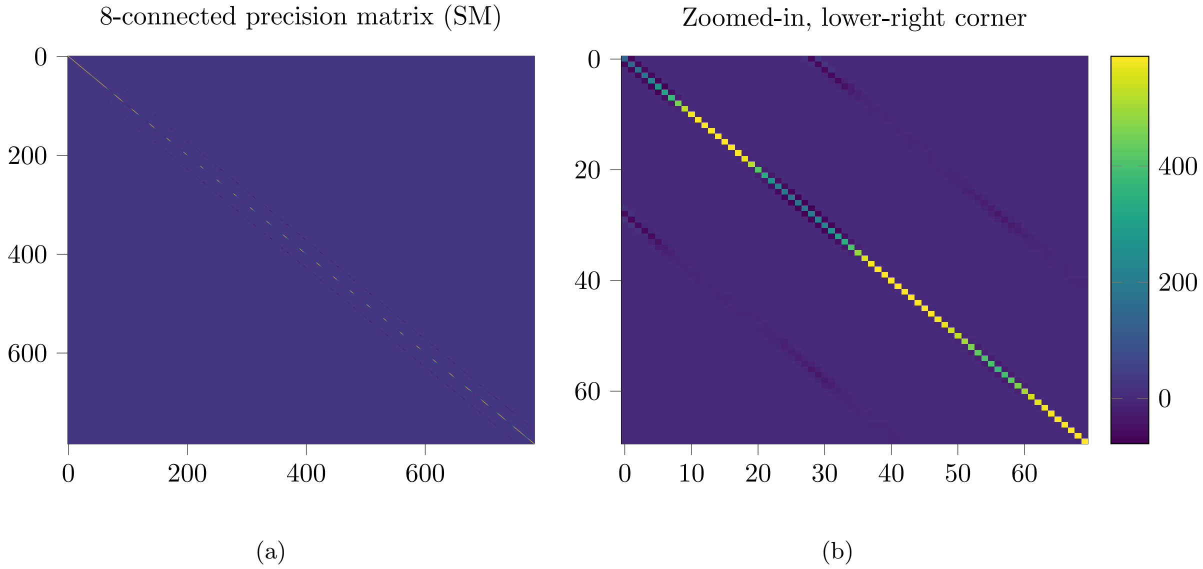
Figure 5: Empirical precision matrix for MNIST



(a)

(b)

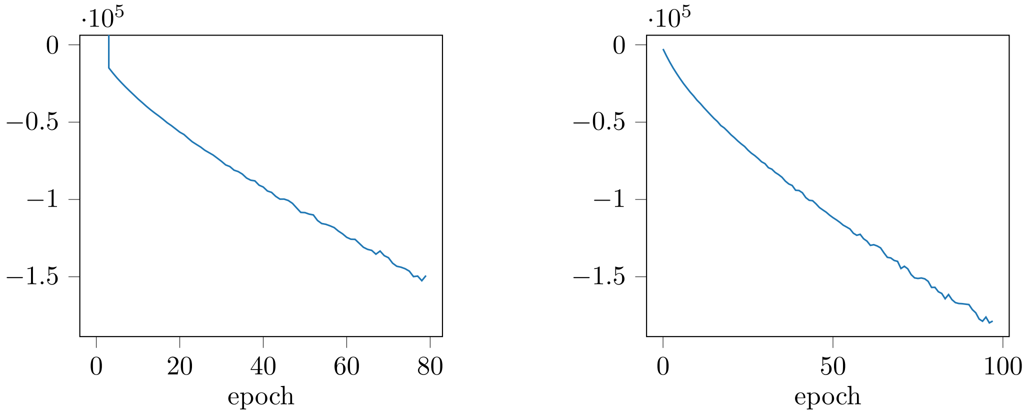
Figure 6: Learned precision matrix Λ_θ using SM, with 4-connected neighbourhood structure.



(a)

(b)

Figure 7: Learned precision matrix Λ_θ using SM, with 8-connected neighbourhood structure.



(a) 4-connected

(b) 8-connected

Figure 8: Loss (SM)

Exercise 2

The first task was to extract 50,000 image patches of resolution 28×28. We solved this problem by running the following loop: In each iteration, an image from the `Flickr30k` dataset is loaded, converted to grayscale, and split into multiple patches using the method `tf.image.extract_patches`. Two such patches are selected at random and saved, before moving on to the next iteration, and the program terminates after saving 50,000 patches.

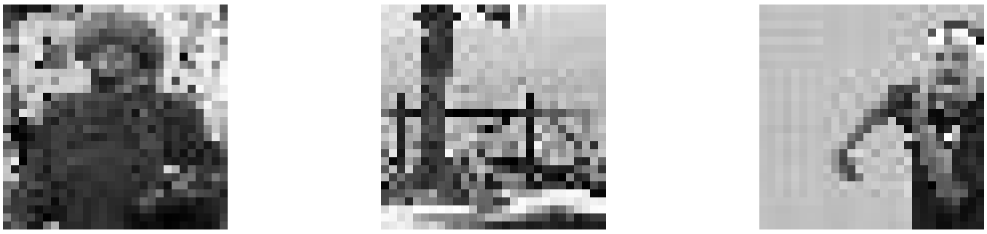
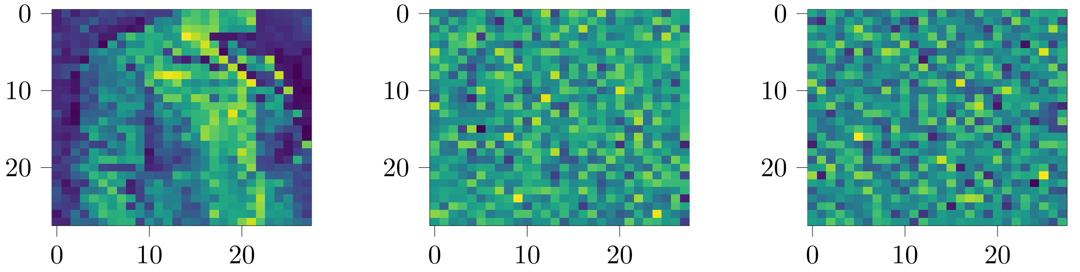


Figure 9: Examples of image patches.

Next, we used SM to compute a constrained Gaussian representing the above data. This allowed us to use the learned covariance $C = \Sigma_\theta = \Lambda_\theta^{-1}$ instead of the empirical covariance $C = \frac{1}{N-1} XX^T$ when performing ZCA whitening. Unfortunately, however, we must have done something wrong when whitening. Even the empirical covariance matrix produces apparent noise, despite us using ZCA rather than PCA whitening:

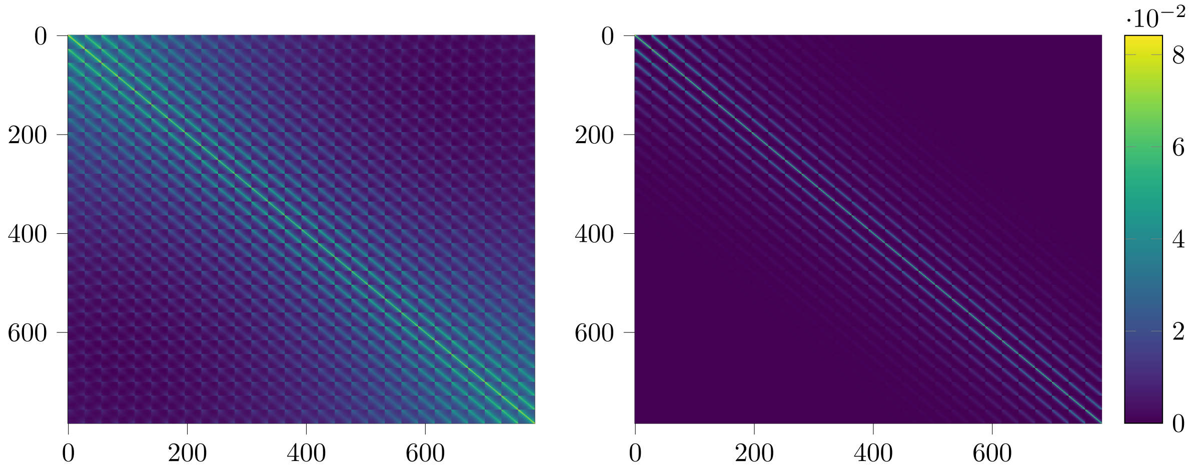


(a) Non-whitened

(b) Whitened with empirical cov.

(c) Whitened with learned cov.

Figure 10: ZCA whitening produces apparent noise, even when using the empirical cov.



(a) Empirical covariance matrix

(b) Learned covariance matrix

Figure 11: Empirical vs. learned covariance matrix, using 4-connected precision matrix.

After whitening, we trained two separate 2-layer deep energy models (DEM) of the form

$$\begin{aligned} \log p_\theta(x) &\doteq -\frac{1}{2\sigma^2}\|x\|^2 + b^T x + \sum_{k=1}^{K} S(w_k^T g_\theta(x) + c_k), \qquad \begin{pmatrix} S(u) = \log(1+e^u) \\ s(u) = \operatorname{sigmoid}(u) \end{pmatrix} \\ g_\theta(x) &= s(Vx) \qquad \text{single layer sigmoid NN} \end{aligned}$$

with whitened and non-whitened data x, respectively. This meant learning two different instances of the parameters V, W, b, c, with the following choice of hyperparameters:[1]

$$K = 64, \qquad V \in \mathbb{R}^{64\times 784}, \qquad \sigma = 1.$$

[1] Due to time constraints, we never attempted $\sigma = 0.1$.

The network was optimized using score matching, i.e. by minimizing the loss function estimate

$$J_{SM}(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left[\frac{1}{2}\|\nabla_x \log p_\theta(x_i)\|^2 + \Delta \log p_\theta(x_i)\right].$$

We can expand the loss function using

$$\frac{1}{2}\|\nabla_x \log p_\theta(x)\|^2 + \Delta \log p_\theta(x) = \sum_{l=1}^{784} \frac{1}{2}\left(\frac{\partial \log p_\theta(x)}{\partial x^l}\right)^2 + \frac{\partial^2 \log p_\theta(x)}{\partial {x^l}^2},$$

and obtain explicit expressions for the derivatives:

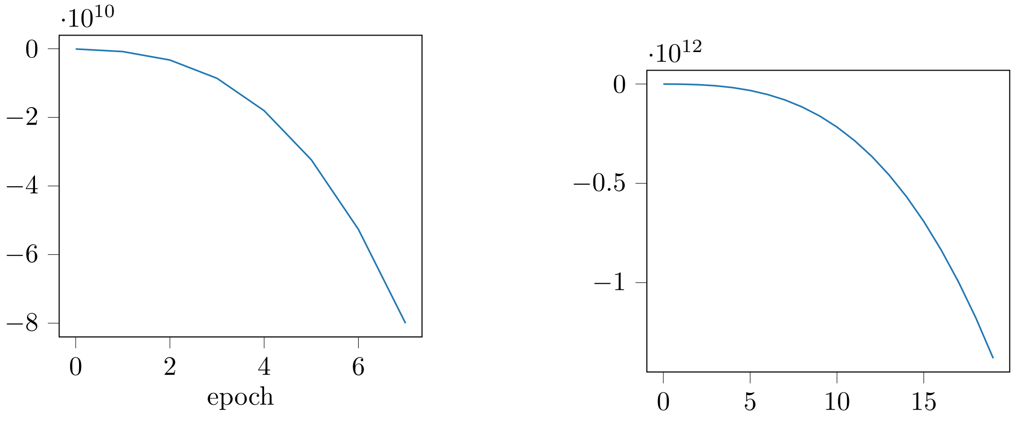
$$\begin{aligned}\frac{\partial \log p_\theta(x)}{\partial x^l} &= -\frac{x^l}{\sigma^2} + b^l + \sum_{k=1}^{K} s(w_k^T g_\theta(x) + c_k)\left(w_k^T \frac{\partial g_\theta(x)}{\partial x^l}\right)\\ &= -\frac{x^l}{\sigma^2} + b^l + \sum_{k=1}^{K} s(w_k^T g_\theta(x) + c_k)\left(w_{ki} \frac{\partial s\left(V_j^i x^j\right)}{\partial x^l}\right) \qquad \text{(Einstein notation)}\\ &= -\frac{x^l}{\sigma^2} + b^l + \sum_{k=1}^{K} s(w_k^T g_\theta(x) + c_k)\Big(w_{ki} s'(V_j^i x^j) V_l^i\Big),\end{aligned}$$

and

$$\begin{aligned}\frac{\partial^2 \log p_\theta(x)}{\partial {x^l}^2} = -\frac{1}{\sigma^2} + \sum_{k=1}^{K} \Bigg[& s'(w_k^T g_\theta(x) + c_k)\Big(w_{ki} s'(V_j^i x^j) V_l^i\Big)^2 + \\ & + s(w_k^T g_\theta(x) + c_k)\left(w_{ki} s''\left(V_j^i x^j\right)\left(V_l^i\right)^2\right)\Bigg]\end{aligned}$$

These expressions are bulky but not difficult to implement

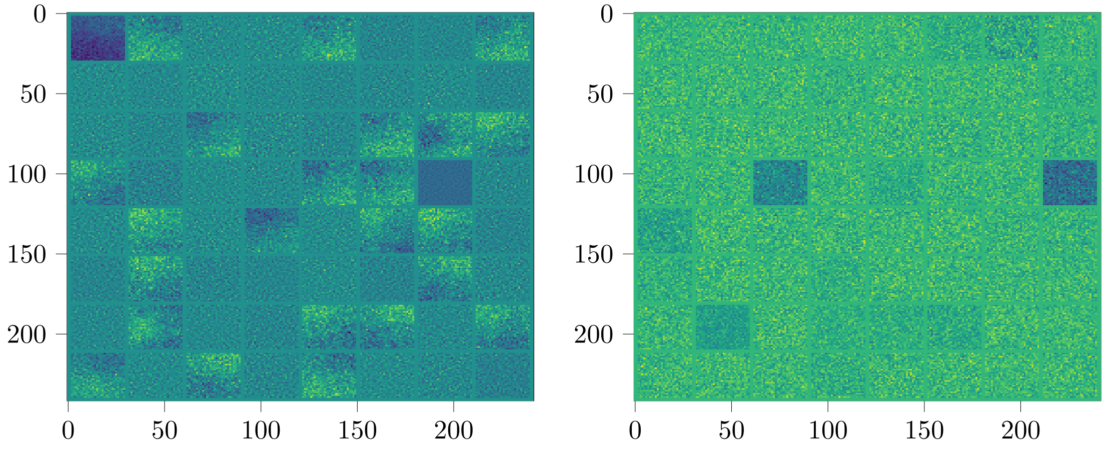
Figure 12 shows the loss for both models, i.e. using non-whitened and whitened data, respectively. Figure 13 illustrates the learned filters v_j for $j = 1, \ldots, 64$, though these mainly consist of noise.



(a) Non-whitened data, $\sigma = 1$.

(b) Whitened data, $\sigma = 1$.

Figure 12: DEM loss with and without whitening



(a) Non-whitened data, $\sigma = 1$.

(b) Whitened data, $\sigma = 1$.

Figure 13: Learned filters v_j of size 28×28, for $j = 1, \ldots, 64$.

Again, we appear to have made several poor design choices and other mistakes that causes the model to focus on the wrong things. Curiously, when comparing the probabilities of whitened data, hold-out natural patches, generated MNIST images, and true MNIST images, we *do* observe a clear distinction, but in completely the wrong direction:

As shown in Tables 1-2, both whitened and hold-out natural images are given extremely small probabilities, while generated and true MNIST images are significantly less improbable. This finding suggests that we have optimized our DEM *away* from the desired distribution.

As disappointing as these results are, we have invested much time and effort in this assignment and learned much from it. We feel it was worthwile.

	Whitened patch	Hold-out patch	Generated MNIST	True MNIST
mean	-232.7	-184.8	-26.7	-4.5
std	113.6	78.8	65.1	46.7

Table 1: Mean and std of $\log p_\theta(x)$ for samples of 10 randomly chosen images.

	Whitened patch	Hold-out patch	Generated MNIST	True MNIST
mean	-Inf	-215.3	-34.2	-17.3
std	NaN	132.7	55.3	58.0

Table 2: Mean and std of $\log p_\theta(x)$ for samples of 100 randomly chosen images.