Learning Feature Representations

Module 1 Homework

Oscar Carlsson
Jimmy Aronsson

November 29, 2020

In this document, we summarize our work on the first Homework.

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. More precisely, we remove the empirical mean μ from each training image and see whether the remaining noise $x - \mu$ can be modeled with a multivariate Gaussian distribution $\mathcal{N}(\mathbf{0}, \Sigma_\theta)$ with learned precision matrix $\Lambda_\theta = \Sigma_\theta^{-1}$. Synthetic images could then be created by either sampling noise from the model distribution and adding back the empirical mean,

$$x' = \mu + \epsilon, \qquad \epsilon \sim \mathcal{N}(\mathbf{0}, \Sigma_\theta),$$

or by using the precision matrix directly:

$$x' = \mu + A_\theta \epsilon, \qquad A_\theta = \Lambda_\theta^{-1/2}, \epsilon \sim \mathcal{N}(0, 1/100).$$

Two different methods have been used to estimate the precision matrix:

- Noise-contrastive estimation (NCE), and
- 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 shall gloss over the introductions and focus on those additional, important details which are not featured in the presentation.

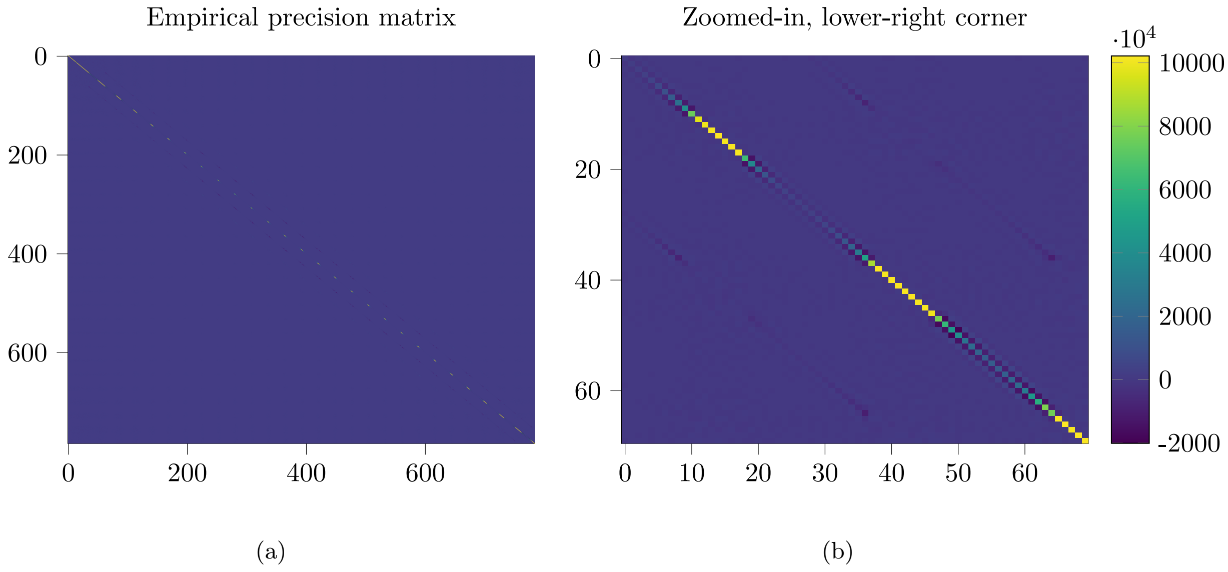


Figure 1: Empirical precision matrix for MNIST

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.

After choosing a suitable noise distribution p_n, we 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 ($z_i = 0$). Denote the resulting training examples by $x_1, \ldots, x_N$ and noise samples by $x'_1, \ldots, x'_M$, so that $M + N$ is the total number of coin flips. In NCE, we wish to 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

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

then we obtain the relatively simple expression

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

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

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

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. Two approaches were considered for keeping Λ_θ positive definite and retaining its sparse 4-/8-connected neighbourhood-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. Learning A_θ also turned out to be slower than the approach below.

2. Forcing a symmetric gradient by throwing away its lower triangular part and replacing it with the transpose of its upper triangular part. We then applied the previously mentioned 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.

We ended up choosing the latter approach.[1]

Insert figures of samples, loss, etc

[1] We later realized Λ_θ can also be modeled through its eigendecomposition, which would make it easy to ensure positive (semi)definiteness by flipping the signs of negative eigenvalues. Due to time constraints and different priorities, however, we never tried implementing this approach.

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

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

and start training. Gradients were again computed with `tf.GradientTape`, and the symmetry and neighbourhood structure was enforced in the same way as for NCE.

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 $\mathcal{N}(\bar{\mu}, \Sigma_\theta)$ using the empirical mean $\bar{\mu}$ and the learned covariance matrix $\Sigma_\theta = \Lambda_\theta^{-1}$. (2) Following the instructions in the assignment by setting

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

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 2-5. 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 6-7 show the learned precision matrix in the cases of 4-/8-connected structure, and finally, Figure 8 illustrates the loss in both cases.

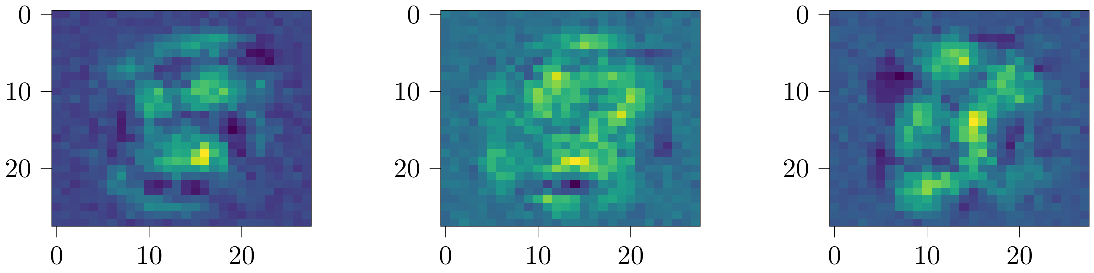


Figure 2: Samples drawn from $\mathcal{N}(\bar{\mu}, \Sigma_\theta)$ with 4-connected precision matrix (SM).

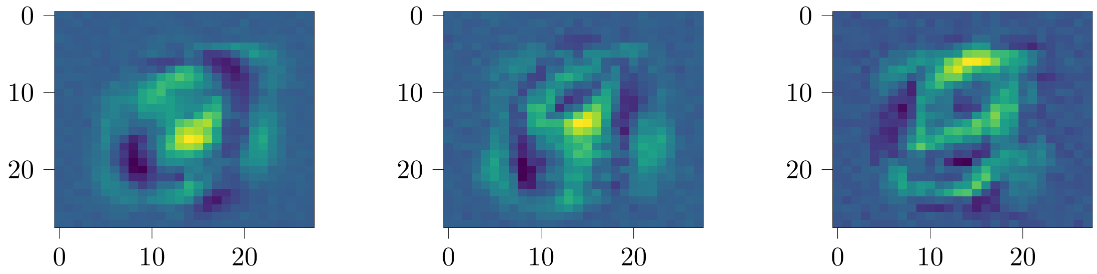


Figure 3: Samples drawn from $\mathcal{N}(\bar{\mu}, \Sigma_\theta)$ with 8-connected precision matrix (SM).

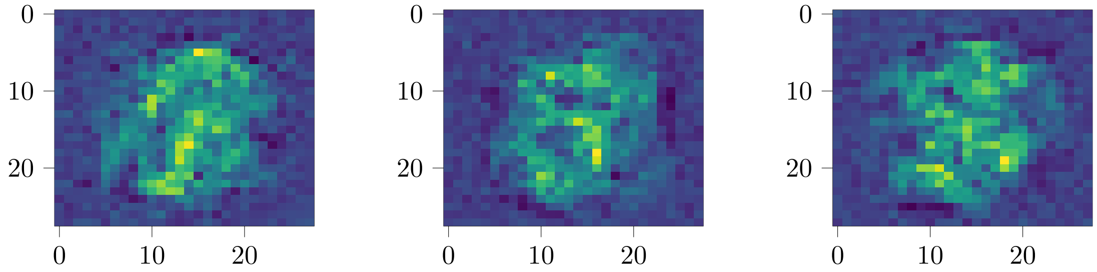


Figure 4: Samples $x \leftarrow \bar{\mu} + \Sigma_\theta^{1/2}\varepsilon$ with $\varepsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and 4-connected precision matrix (SM).

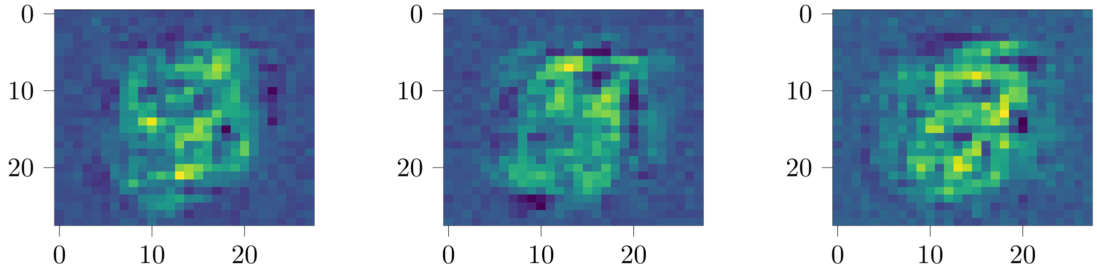
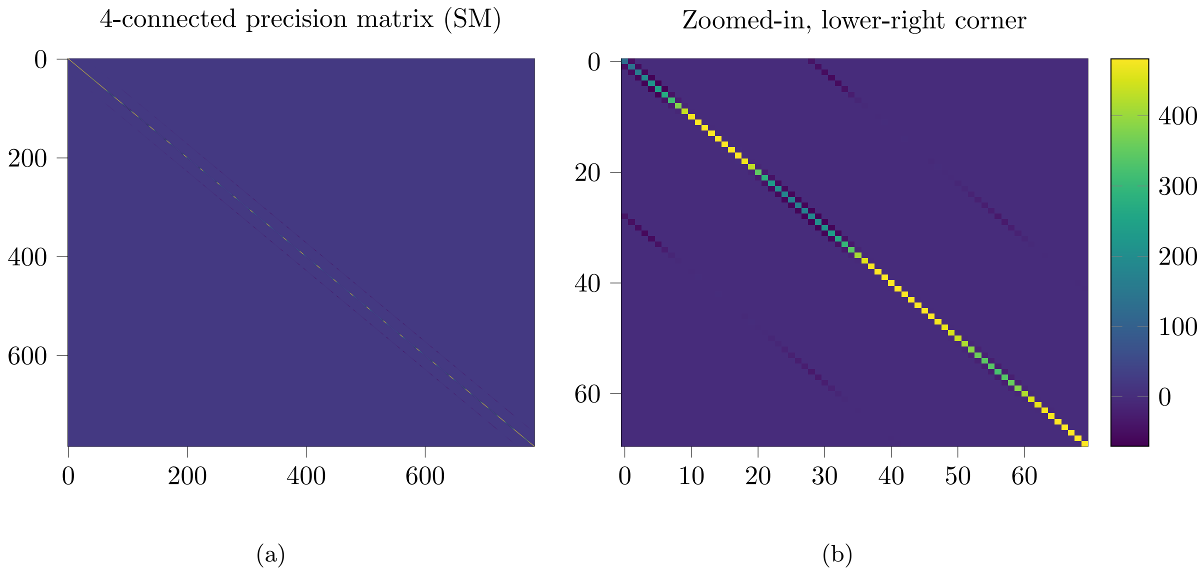


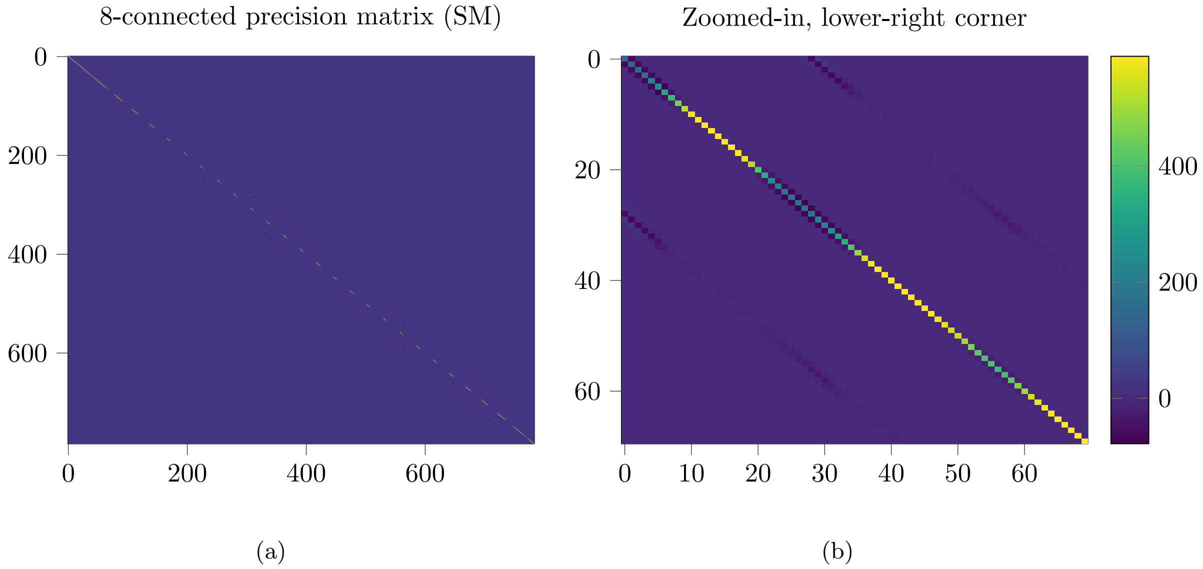
Figure 5: Samples $x \leftarrow \bar{\mu} + \Sigma_\theta^{1/2}\varepsilon$ with $\varepsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and 8-connected precision matrix (SM).



(a)

(b)

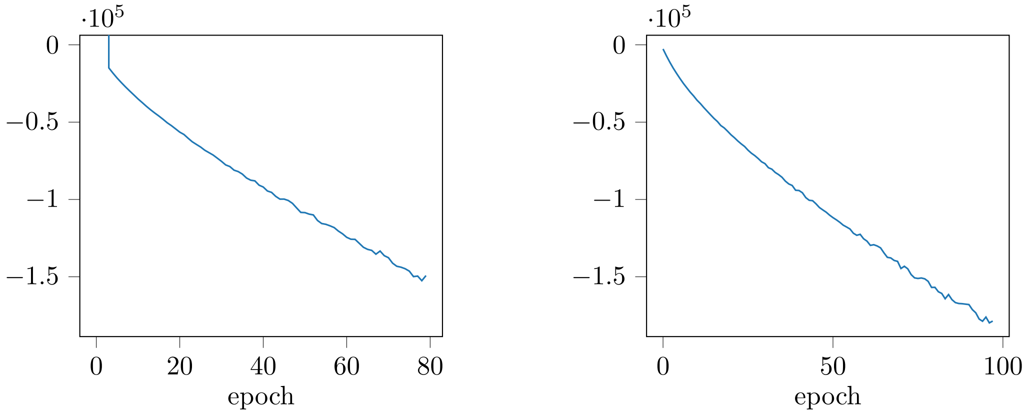
Figure 6: Learned precision matrix Λ_θ using SM, with 4-connected neighbourhood structure.



(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 step 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. See `create_image_patches.py` for details.

Next, we computed a constrained Gaussian representing the above data Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.