

LEARNING FEATURE REPRESENTATIONS

MODULE 1 HOMEWORK

OSCAR CARLSSON

JIMMY ARONSSON

NOVEMBER 28, 2020

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

Exercise 1

In the first exercise, we fit a multivariate Gaussian to MNIST patches'

Noise distribution $\mathcal{N}(\mathbf{0}, \Sigma_n)$ Model distribution $\mathcal{N}(\mathbf{0}, \Sigma_\theta)$

NCE

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.

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

We approximate the right-hand side of equation (1) using the empirical estimate

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

where $x_i \sim p_d$ are training examples and $x'_j \sim p_n$ are drawn from the noise distribution. Next, we simplify the above expression above by rewriting both terms in the following way:

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

and then insert the relative probability

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

to obtain the relatively simple expression

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

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

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

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 . 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 $\Lambda_\theta = (A_\theta^T A_\theta) \circ M$ for some learned, invertible matrix A_θ and a predefined masking matrix M to enforce the neighbourhood structure - here, \circ denotes the Hadamard product (i.e. element-wise multiplication).

The product $A^T A$ is guaranteed to be symmetric positive definite whenever A is invertible, and it can be shown that the Hadamard product of two positive definite matrices is again positive definite, so we hoped this would be sufficient to guarantee that Λ_θ is symmetric positive definite. Unfortunately, we eventually realized that our masking matrix is not positive definite, so we could not obtain such a guarantee on Λ_θ . Learning A_θ also turned out to be slower than the following approach, so we ditched this one.

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, though we still cannot guarantee it being positive definite.

We later realized that 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. But we didn't have time to implement this change.

Insert figures of samples, loss, etc

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

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

and start training. Gradients were again computed with `Gradient Tape` and enforced symmetry.

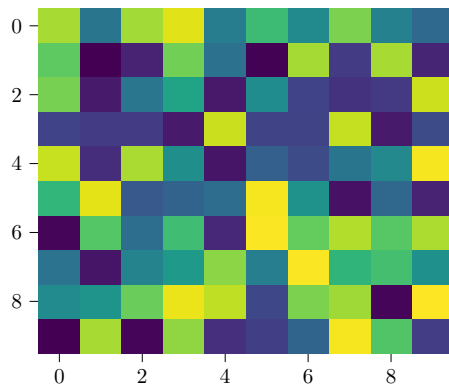


Figure 1: Test image and caption

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 ullamcor-
per vestibulum turpis. Pellentesque cursus luctus mauris.