

Exercise 1

- Convert 8-bit data to $[0, 1]$ range and add per-pixel Gaussian noise $\epsilon \sim \mathcal{N}(0, 1/100)$.
- Model image patches as Gaussian

$$p_{\theta}(x) = \frac{1}{Z} e^{-\frac{1}{2}(x-\mu)^T \Lambda (x-\mu)}.$$

- Subtract the empirical mean, leading to

$$p_{\theta}(x) = \frac{1}{Z} e^{-\frac{1}{2}x^T \Lambda x}.$$

- Λ has a 2D Laplacian structure: 4-connected neighbouring pixels are correlated.
- Estimate Λ via
 - NCE (explain your choice of $p_n(x')$).
 - cNCE (explain your choice of $p_n(x'|x)$) or score matching (coin flip).
- Use SGD (or RMSProp or ADAM) for gradient-based optimization
- Visualize samples from p_{θ} (with $A = \Lambda^{-1/2} = \Sigma^{1/2}$)

$$x' \leftarrow \mu + A\epsilon, \quad \epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I}).$$

- NCE: How close is the estimate of $\log Z$ to $\frac{D}{2} \log(2\pi) + \frac{1}{2} \log |\Sigma|$? ($D = 28^2$)
- Bonus exercise: Rerun with 8-connected neighbourhood assumption

Noise-contrastive estimation (NCE)

NCE casts the estimation of distribution parameters (in our case, the precision matrix Λ_{θ}) as a supervised learning problem. It also jointly estimates the unknown partition function Z_{θ} , and it does all this by applying logarithmic PSR to estimate parameters of a binary random variable.

Start by considering a batch of MNIST training images x_1, x_2, x_3, \dots , which we assume are drawn from an unknown data distribution p_d . Also consider a fully known noise distribution $p_n(x')$; we could try a multidimensional Gaussian $\mathcal{N}(0, \Sigma_n)$, maybe with diagonal Σ_n ? Perhaps $\Sigma_n = \epsilon I$?

Next, randomly let $z \in \{0, 1\}$ decide whether to draw a sample from p_d (i.e., drawing a random training image x_i) or a sample from the noise distribution p_n . That is,

$$p_{d,n}(x|z=0) = p_d(x), \quad p_{d,n}(x|z=1) = p_n(x).$$

Further suppose that $p_z(z=0) = \eta$ and $p_z(z=1) = 1 - \eta$, where $\eta \in [0, 1]$ is a hyperparameter. Bayes rule combined with some formal symbol pushing yields the posterior distribution

$$p_{d,n}(z|x) = \frac{(1-z)p_d(x) + zp_n(x)}{p_d(x) + zp_n(x)}, \quad \left(\eta = \frac{1}{1+\nu} \right)$$

Given a sample x , we can thus calculate the probability $p_{d,n}(0|x)$ that x was drawn from the data distribution p_d , and the probability $p_{d,n}(1|x) = 1 - p_{d,n}(0|x)$ that x was drawn from the noise distribution p_n .

Posterior induced by true data distribution p_d :

$$p_{d,n}(z|x) = \frac{(1-z)p_d(x) + \nu z p_n(x)}{p_d(x) + \nu p_n(x)}$$

Posterior induced by model distribution p_θ :

$$p_{\theta,n}(z|x) = \frac{(1-z)p_\theta(x) + \nu z p_n(x)}{p_\theta(x) + \nu p_n(x)}$$

NCE uses logarithmic PSR to align these two posterior distributions. In order to do this, we need to form a set of training data $\{(x_i, z_i)\}$ where $z_i \sim p(z)$ and

$$x_i \sim \begin{cases} p_d(x) & \text{if } z = 0 \\ p_n(x) & \text{if } z = 1 \end{cases}$$

This should be very easy to implement in code:

1. Draw $z \in \{0, 1\}$ many times, from a Bernoulli distribution with probability $p(z = 0) = \eta$. Let N be the number of times you drew $z = 0$ and let M be the number of times you drew $z = 1$. (You can probably draw all these $N + M$ times using a single command, in one single line, you don't need a for-loop.)
2. For each time you drew $z = 0$, pick a random training example. For each time you drew $z = 1$, sample from the fully known noise distribution $p_n(x)$.
3. Store a list of the pairs (x_i, z_i) .

In the next step, we will use primes and different subscripts to distinguish between training examples $x_i \sim p_d(x)$ and noise samples $x'_j \sim p_n(x)$. Note that expressions such as $p_d(x'_j)$ and $p_n(x_i)$ do make sense; we can always evaluate the probability $p_n(x_i)$ of drawing the training example x_i from the noise distribution p_n and vice versa.

The goal now is to minimize the NCE objective

$$\begin{aligned} J(\theta) &\propto \mathbb{E}_{x \sim p_d} \left[\log \frac{p_\theta(x)}{p_\theta(x) + \nu p_n(x)} \right] + \nu \mathbb{E}_{x \sim p_n} \left[\log \frac{\nu p_n(x)}{p_\theta(x) + \nu p_n(x)} \right] \\ &\approx \frac{1}{N} \sum_{i=1}^N \log \frac{p_\theta(x_i)}{p_\theta(x_i) + \nu p_n(x_i)} + \frac{\nu}{M} \sum_{j=1}^M \log \frac{\nu p_n(x'_j)}{p_\theta(x'_j) + \nu p_n(x'_j)}, \end{aligned}$$

with $x_i \sim p_d$ and $x'_j \sim p_n$. Now suppose that

$$\begin{aligned} p_\theta(x) &= \frac{1}{Z_\theta} e^{-\frac{1}{2}(x-\mu)^T \Lambda (x-\mu)} =: \frac{1}{Z_\theta} e^{f_\theta(x)}, \\ p_n(x) &= \frac{1}{Z_n} e^{-\frac{1}{2}(x-\mu_n)^T \Lambda_n (x-\mu_n)} =: \frac{1}{Z_n} e^{f_n(x)}. \end{aligned}$$

Then we get

$$\begin{aligned}\log p_\theta(x) &= f_\theta(x) - \log Z_\theta = -\frac{1}{2}(x - \mu)^T \Lambda(x - \mu) - \log Z_\theta, \\ \log p_n(x) &= f_n(x) - \log Z_n = -\frac{1}{2}(x - \mu_n)^T \Lambda_n(x - \mu_n) - \log Z_n,\end{aligned}$$

hence¹

$$\begin{aligned}J(\theta) &\approx \frac{1}{N} \sum_{i=1}^N \left(f_\theta(x_i) - \log Z_\theta - \log(p_\theta(x_i) + \nu p_n(x_i)) \right) + \\ &\quad + \frac{\nu}{M} \sum_{j=1}^M \left(f_n(x'_j) - \log Z_n - \log(p_\theta(x'_j) + \nu p_n(x'_j)) + \log \nu \right) \\ &= \frac{1}{N} \sum_{i=1}^N f_\theta(x_i) + \frac{\nu}{M} \sum_{j=1}^M f_n(x'_j) - \log Z_\theta - \nu \log Z_n + \nu \log \nu - \\ &\quad - \frac{1}{N} \sum_{i=1}^N \log(p_\theta(x_i) + \nu p_n(x_i)) - \frac{\nu}{M} \sum_{j=1}^M \log(p_\theta(x'_j) + \nu p_n(x'_j))\end{aligned}$$

We can also rewrite the last terms by noting that

$$\begin{aligned}\log(p_\theta(x) + \nu p_n(x)) &= \log \left(\frac{1}{Z_\theta} e^{f_\theta(x)} + \frac{\nu}{Z_n} e^{f_n(x)} \right) \\ &= \log \left(\left(\frac{Z_n}{Z_\theta} e^{f_\theta(x) - f_n(x)} + \nu \right) \left(\frac{1}{Z_n} e^{f_n(x)} \right) \right) \\ &= \log \left(\frac{Z_n}{Z_\theta} e^{f_\theta(x) - f_n(x)} + \nu \right) + f_n(x) - \log Z_n,\end{aligned} \tag{1}$$

¹I write $J(\theta) \approx$, but what I mean is that $J(\theta)$ is proportional to something which is approximately equal to ...

hence

$$\begin{aligned}
J(\theta) &\approx \frac{1}{N} \sum_{i=1}^N f_\theta(x_i) + \frac{\nu}{M} \sum_{j=1}^M f_n(x'_j) - \log Z_\theta - \nu \log Z_n + \nu \log \nu - \\
&\quad - \frac{1}{N} \sum_{i=1}^N \log(p_\theta(x_i) + \nu p_n(x_i)) - \frac{\nu}{M} \sum_{j=1}^M \log(p_\theta(x'_j) + \nu p_n(x'_j)) \\
&= \frac{1}{N} \sum_{i=1}^N f_\theta(x_i) + \frac{\nu}{M} \sum_{j=1}^M f_n(x'_j) - \log Z_\theta - \nu \log Z_n + \nu \log \nu - \\
&\quad - \frac{1}{N} \sum_{i=1}^N \left(\log \left(\frac{Z_n}{Z_\theta} e^{f_\theta(x_i) - f_n(x_i)} + \nu \right) + f_n(x_i) - \log Z_n \right) - \\
&\quad - \frac{\nu}{M} \sum_{j=1}^M \left(\log \left(\frac{Z_n}{Z_\theta} e^{f_\theta(x'_j) - f_n(x'_j)} + \nu \right) + f_n(x'_j) - \log Z_n \right) \\
&= \frac{1}{N} \sum_{i=1}^N [f_\theta(x_i) - f_n(x_i)] + \log(Z_n/Z_\theta) + \nu \log \nu - \\
&\quad - \frac{1}{N} \sum_{i=1}^N \log \left(\frac{Z_n}{Z_\theta} e^{f_\theta(x_i) - f_n(x_i)} + \nu \right) - \frac{\nu}{M} \sum_{j=1}^M \log \left(\frac{Z_n}{Z_\theta} e^{f_\theta(x'_j) - f_n(x'_j)} + \nu \right)
\end{aligned}$$

We can also rewrite this in another way, by factorizing out p_θ instead of p_n in equation (1). Then we should get something similar to

$$\begin{aligned}
J(\theta) &\approx \frac{\nu}{M} \sum_{j=1}^M [f_n(x'_j) - f_\theta(x'_j)] - \nu \log(\nu Z_\theta/Z_n) - \\
&\quad - \frac{1}{N} \sum_{i=1}^N \log \left(\nu \frac{Z_\theta}{Z_n} e^{f_n(x_i) - f_\theta(x_i)} + 1 \right) - \frac{\nu}{M} \sum_{j=1}^M \log \left(\nu \frac{Z_\theta}{Z_n} e^{f_n(x'_j) - f_\theta(x'_j)} + 1 \right)
\end{aligned}$$

I spent a very small amount of time trying to see if these sums of logarithms simplify, using the additive law of logarithms, but they don't seem to. I also tried calculating the average of these two approximations of $J(\theta)$ to see if that would simplify the logarithms, but that didn't seem to give me anything either.

Now suppose that we have normalized so that $p_\theta \sim \mathcal{N}(0, \Sigma_\theta)$ and $p_n \sim \mathcal{N}(0, \Sigma_n)$ with $\Lambda = \Sigma^{-1}$. Using the identity $\det(\Sigma) = 1/\det(\Lambda)$, the above approximation of $J(\theta)$ becomes

$$J(\theta) \approx \frac{\nu}{M} \left(\sum_{j=1}^M x_j'^T (\Lambda_n - \Lambda_\theta) x_j' \right) - \frac{\nu}{2} \log \left(\nu^2 \frac{\det \Lambda_n}{\det \Lambda_\theta} \right) -$$

$$- \frac{1}{N} \sum_{i=1}^N \log \left(\nu \sqrt{\frac{\det \Lambda_n}{\det \Lambda_\theta}} \exp(x_i^T (\Lambda_n - \Lambda_\theta) x_i) + 1 \right) -$$

$$- \frac{\nu}{M} \sum_{j=1}^M \log \left(\nu \sqrt{\frac{\det \Lambda_n}{\det \Lambda_\theta}} \exp(x_j'^T (\Lambda_n - \Lambda_\theta) x_j') + 1 \right)$$

This expression honestly doesn't look too bad, we can definitely compute every component. Computing the gradient of this approximation is fairly straightforward (and can likely be automated with Gradient Tape, so the following computations may be unnecessary). Before computing the derivatives, recall that the derivative of a parameterized matrix is defined by computing the derivative of each component:

$$\left(\frac{\partial \Lambda_\theta}{\partial \theta} \right)_{ij} := \frac{\partial (\Lambda_\theta)_{ij}}{\partial \theta},$$

where we have pretended that θ is a single variable. In practice, θ is an array containing multiple parameters θ_k , and the corresponding expressions for derivatives with respect to θ_k are obtained by simply adding the subscript k at the appropriate places:

$$\left(\frac{\partial \Lambda_\theta}{\partial \theta_k} \right)_{ij} = \frac{\partial \Lambda_{ij}}{\partial \theta_k}.$$

For notational convenience we shall also suppress the subscript when computing $\frac{\partial J}{\partial \theta_k}$, because we can just put the subscript back afterwards. We find that

$$\frac{\partial J}{\partial \theta} \approx - \frac{\nu}{M} \left(\sum_{j=1}^M x_j'^T \frac{\partial \Lambda_\theta}{\partial \theta} x_j' \right) + \frac{\nu}{2 \det \Lambda_\theta} \frac{\partial \det \Lambda_\theta}{\partial \theta} -$$

$$- \frac{1}{N} \sum_{i=1}^N \frac{\nu \sqrt{\det \Lambda_n} \left[-\frac{1}{2} (\det \Lambda_\theta)^{-3/2} \frac{\partial \det \Lambda_\theta}{\partial \theta} - (\det \Lambda_\theta)^{-1/2} (x_i^T \frac{\partial \Lambda_\theta}{\partial \theta} x_i) \right] \exp(x_i^T (\Lambda_n - \Lambda_\theta) x_i)}{\nu \sqrt{\frac{\det \Lambda_n}{\det \Lambda_\theta}} \exp(x_i^T (\Lambda_n - \Lambda_\theta) x_i) + 1} -$$

$$- \frac{\nu}{M} \sum_{j=1}^M \frac{\nu \sqrt{\det \Lambda_n} \left[-\frac{1}{2} (\det \Lambda_\theta)^{-3/2} \frac{\partial \det \Lambda_\theta}{\partial \theta} - (\det \Lambda_\theta)^{-1/2} (x_j'^T \frac{\partial \Lambda_\theta}{\partial \theta} x_j') \right] \exp(x_j'^T (\Lambda_n - \Lambda_\theta) x_j')}{\nu \sqrt{\frac{\det \Lambda_n}{\det \Lambda_\theta}} \exp(x_j'^T (\Lambda_n - \Lambda_\theta) x_j') + 1},$$

which can be greatly simplified by cancelling factors from both numerators and denominators:

$$\begin{aligned}
\frac{\partial J}{\partial \theta} \approx & -\frac{\nu}{M} \left(\sum_{j=1}^M x_j'^T \frac{\partial \Lambda_\theta}{\partial \theta} x_j' \right) + \frac{\nu}{2 \det \Lambda_\theta} \frac{\partial \det \Lambda_\theta}{\partial \theta} + \\
& + \frac{\nu}{N} \sum_{i=1}^N \frac{\frac{1}{2 \det \Lambda_\theta} \frac{\partial \det \Lambda_\theta}{\partial \theta} + x_i^T \frac{\partial \Lambda_\theta}{\partial \theta} x_i}{\nu + \sqrt{\frac{\det \Lambda_\theta}{\det \Lambda_n}} \exp(-x_i^T (\Lambda_n - \Lambda_\theta) x_i)} + \\
& + \frac{\nu^2}{M} \sum_{j=1}^M \frac{\frac{1}{2 \det \Lambda_\theta} \frac{\partial \det \Lambda_\theta}{\partial \theta} + x_j'^T \frac{\partial \Lambda_\theta}{\partial \theta} x_j'}{\nu + \sqrt{\frac{\det \Lambda_\theta}{\det \Lambda_n}} \exp(-x_j'^T (\Lambda_n - \Lambda_\theta) x_j')}
\end{aligned}$$

This expression can be simplified further, using Jacobi's formula:

$$\frac{\partial \det \Lambda_\theta}{\partial \theta} = \text{tr} \left(\text{adj}(\Lambda_\theta) \frac{\partial \Lambda_\theta}{\partial \theta} \right),$$

where $\text{adj}(\Lambda_\theta)$ is the adjugate of Λ_θ . Since Λ_θ is invertible, $\text{adj}(\Lambda_\theta) = (\det \Lambda_\theta) \Lambda_\theta^{-1} = (\det \Lambda_\theta) \Sigma_\theta$, so after reintroducing the subscript k we get

$$\begin{aligned}
\frac{\partial J(\theta)}{\partial \theta_k} \approx & -\frac{\nu}{M} \left(\sum_{j=1}^M x_j'^T \frac{\partial \Lambda_\theta}{\partial \theta_k} x_j' \right) + \frac{\nu}{2} \text{tr} \left(\Sigma_\theta \frac{\partial \Lambda_\theta}{\partial \theta_k} \right) + \\
& + \frac{\nu}{N} \sum_{i=1}^N \frac{\frac{1}{2} \text{tr} \left(\Sigma_\theta \frac{\partial \Lambda_\theta}{\partial \theta_k} \right) + x_i^T \frac{\partial \Lambda_\theta}{\partial \theta_k} x_i}{\nu + \sqrt{\frac{\det \Lambda_\theta}{\det \Lambda_n}} \exp(-x_i^T (\Lambda_n - \Lambda_\theta) x_i)} + \\
& + \frac{\nu^2}{M} \sum_{j=1}^M \frac{\frac{1}{2} \text{tr} \left(\Sigma_\theta \frac{\partial \Lambda_\theta}{\partial \theta_k} \right) + x_j'^T \frac{\partial \Lambda_\theta}{\partial \theta_k} x_j'}{\nu + \sqrt{\frac{\det \Lambda_\theta}{\det \Lambda_n}} \exp(-x_j'^T (\Lambda_n - \Lambda_\theta) x_j')}
\end{aligned}$$

This expression is bulky, but if we can compute $\frac{\partial \Lambda_\theta}{\partial \theta_k}$ and $\Sigma_\theta = \Lambda_\theta^{-1}$ efficiently, then the whole derivative should be relatively efficient to compute. Indeed, the expression mainly contains determinants of known matrices, matrix products of known matrices, and scalar products. Of course, we need to compute this derivative for each component θ_k but we can save resources by only computing the k -independent weights

$$w(x) = \sqrt{\frac{\det \Lambda_n}{\det \Lambda_\theta}} \exp(x^T (\Lambda_n - \Lambda_\theta) x)$$

for $x \in \{x_1, \dots, x_N, x_1', \dots, x_M'\}$ once each time we update θ (that is, once per epoch). We can then reuse these pre-computed weights when computing each derivative $\frac{\partial J(\theta)}{\partial \theta_k}$, which then has the simpler expression

$$\begin{aligned}
\frac{\partial J(\theta)}{\partial \theta_k} \approx & \frac{\nu}{N} \sum_{i=1}^N \frac{w(x_i)}{\nu w(x_i) + 1} \left(x_i^T \frac{\partial \Lambda_\theta}{\partial \theta_k} x_i \right) - \\
& - \frac{\nu}{M} \sum_{j=1}^M \left(\frac{w(x'_j)}{\nu w(x'_j) + 1} - 1 \right) \left(x'_j{}^T \frac{\partial \Lambda_\theta}{\partial \theta_k} x'_j \right) + \\
& + \frac{1}{2} \text{tr} \left(\Sigma_\theta \frac{\partial \Lambda_\theta}{\partial \theta_k} \right) \left(\nu + \frac{1}{N} \sum_{i=1}^N \frac{w(x_i)}{\nu w(x_i) + 1} + \frac{\nu}{M} \sum_{j=1}^M \frac{w(x'_j)}{\nu w(x'_j) + 1} \right)
\end{aligned}$$

Many parts of this expression are independent of k and can be pre-computed. Moreover, it would be nice if we could rewrite $\Sigma_\theta \frac{\partial \Lambda_\theta}{\partial \theta_k}$ as the derivative $\frac{\partial M_\theta}{\partial \theta_k}$ of some matrix M_θ , because then we could use linearity to rewrite the trace term as

$$\frac{\partial \text{tr}(M)}{\partial \theta_k},$$

but I don't know if such a matrix M exists. It might also be computationally cheaper to undo our earlier application of Jacobi's formula and, instead of the trace, compute

$$\frac{1}{\det(\Lambda_\theta)} \frac{\partial \det(\Lambda_\theta)}{\partial \theta_k},$$

but I doubt it; the trace should be cheaper as we need to compute $\frac{\partial \Lambda_\theta}{\partial \theta_k}$ anyway.

It might very well be possible to construct a tensor $\nabla_\theta \Lambda_\theta$, with matrix-valued components $\frac{\partial \Lambda_\theta}{\partial \theta_k}$, and compute $\nabla J(\theta)$ in one go. That depends on how capable Tensorflow / PyTorch / ... is.

Remark: If $\frac{\partial \Sigma_\theta}{\partial \theta_k}$ for some reason is easier to compute than $\frac{\partial \Lambda_\theta}{\partial \theta_k}$, then we can rewrite the above expression using the equalities

$$\frac{\partial \Lambda_\theta}{\partial \theta_k} = -\Lambda_\theta \frac{\partial \Sigma_\theta}{\partial \theta_k} \Lambda_\theta, \quad \Sigma_\theta \frac{\partial \Lambda_\theta}{\partial \theta_k} = -\frac{\partial \Sigma_\theta}{\partial \theta_k} \Lambda_\theta.$$

Implementation steps for NCE:

Step 0. Load MNIST minibatch $\{x_1, x_2, x_3, \dots\}$ which has been preprocessed, e.g. the empirical mean of the entire MNIST dataset should be subtracted from each image. Next, define the model distribution $p_\theta \sim \mathcal{N}(0, \Sigma_\theta)$ for some initial weights $\theta \in \mathbb{R}^K$, and a noise distribution $p_n \sim \mathcal{N}(0, \Sigma_n)$. Then choose a hyperparameter $\eta \in [0, 1]$ for the distribution $p_z(z=0) = \eta$ and compute $\nu = \frac{1}{\eta} - 1$.

Step 1. Draw $z \sim p_z$ a predetermined number of times. (This number is a hyperparameter?)

- Each time $z = 0$, draw a random training example x_i from the minibatch.
- Each time $z = 1$, draw x'_j from the noise distribution p_n .

Let N be the number of times you drew $z = 0$, and M the number of times you drew $z = 1$.

Step 2. Use the samples obtained above to compute the weights

$$w(x) = \sqrt{\frac{\det \Lambda_n}{\det \Lambda_\theta}} \exp(x^T (\Lambda_n - \Lambda_\theta) x)$$

for all samples $x \in \{x_1, \dots, x_N, x'_1, \dots, x'_M\}$.

Step 3. Compute the NCE objective²

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

Step 4. Minimize the NCE objective, either using Gradient Tape or using the relation

$$\begin{aligned} \frac{\partial J(\theta)}{\partial \theta_k} \approx & \frac{\nu}{N} \sum_{i=1}^N \frac{w(x_i)}{\nu w(x_i) + 1} \left(x_i^T \frac{\partial \Lambda_\theta}{\partial \theta_k} x_i \right) - \\ & - \frac{\nu}{M} \sum_{j=1}^M \left(\frac{w(x'_j)}{\nu w(x'_j) + 1} - 1 \right) \left(x_j'^T \frac{\partial \Lambda_\theta}{\partial \theta_k} x'_j \right) + \\ & + \frac{1}{2} \text{tr} \left(\Sigma_\theta \frac{\partial \Lambda_\theta}{\partial \theta_k} \right) \left(\nu + \frac{1}{N} \sum_{i=1}^N \frac{w(x_i)}{\nu w(x_i) + 1} + \frac{\nu}{M} \sum_{j=1}^M \frac{w(x'_j)}{\nu w(x'_j) + 1} \right) \end{aligned}$$

for each parameter θ_k . It might be possible to construct a tensor $\nabla_\theta \Lambda_\theta$ so the full gradient $\nabla J(\theta)$ can be computed in one go. Regardless, I hope we can use Gradient Tape to compute each derivative $\frac{\partial \Lambda_\theta}{\partial \theta_k}$ because I don't know what to do otherwise.

²The approximation sign is not really correct. It is an approximation of an expectation *proportional to* $J(\theta)$.