Exercise 1 - Maximum Likelihood Estimation Refresher

Assume you are given datapoints $(\mathbf{x}_i)_{i=1}^N$ with $\mathbf{x}_i \in \mathbb{R}^n$ coming from a Gaussian distribution. Derive the maximum likelihood estimator of its mean.

Solution

First, let's quickly remember that the maximum likelihood estimator (MLE) of a probability distribution from dataapoints $\mathbf{x}_1, \dots, \mathbf{x}_N$ is given by

$$\hat{\theta}_{\text{MLE}} = \arg\max_{\theta \in \Theta} \prod_{i=1}^{N} f(\mathbf{x}_i|\theta),$$

where f is the probability density function of the considered probability distribution family, θ are the parameters of the distribution, and Θ is the parameter space (a set containing all possible parameters). The product on the right hand side is also known as the likelihood function. In practice, we usually work with the log-likelihood function instead. Because the logarithm is monotonously increasing, the resulting estimators are the same, i.e.

$$\hat{\theta}_{\text{MLE}} = \arg\max_{\theta \in \Theta} \prod_{i=1}^{N} f(\mathbf{x}_i|\theta), = \arg\max_{\theta \in \Theta} \sum_{i=1}^{N} \log\left(f(\mathbf{x}_i|\theta)\right).$$

Second, let's remember that the probability density function of a multivariate Gaussian distribution is given by

$$f(\mathbf{x}_i|\mu, \Sigma) = \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x}_i - \mu)^\top \Sigma^{-1}(\mathbf{x}_i - \mu)\right)$$

with parameters $\theta = (\mu, \Sigma)$, where $\mu \in \mathbb{R}^n$ is the mean vector and $\Sigma \in \mathbb{R}^{n \times n}$ is the covariance matrix. Moreover, the notation $|\Sigma|$ denotes the determinant of Σ.

Our goal is to maximize the log-likelihood function which in this case is

$$\begin{aligned} l(\mu, \Sigma|\mathbf{x}_1, \dots, \mathbf{x}_N) &:= \sum_{i=1}^{N} \log f(\mathbf{x}_i|\mu, \Sigma) \\ &= \sum_{i=1}^{N} \left(-\frac{n}{2}\log(2\pi) - \frac{1}{2}\log|\Sigma| - \frac{1}{2}(\mathbf{x}_i - \mu)^\top \Sigma^{-1}(\mathbf{x}_i - \mu)\right). \end{aligned}$$

For obtaining the MLE of μ, we can simply take the gradient of l with respect to μ and set it to zero resulting in:

$$\nabla_\mu l(\mu, \Sigma|\mathbf{x}_1, \dots, \mathbf{x}_N) = \sum_{i=1}^{N} \Sigma^{-1}(\mathbf{x}_i - \mu) = 0$$

Since Σ is a covariance matrix, it is positive definite. Thus, we can multiply both sides of the equation by Σ and obtain

$$\begin{aligned} 0 &= N\mu - \sum_{i=1}^{N} \mathbf{x}_i, \\ \Rightarrow \hat{\mu}_{\text{MLE}} &= \frac{1}{N}\sum_{i=1}^{N} \mathbf{x}_i. \end{aligned}$$

Conveniently, Σ disappears and thus we do not have to worry about it.

Exercise 2 - More Gradients

You are an ML Engineer at Googlezon where you are working on an internal ML framework called TorchsorFlow. You are tasked with implementing a new layer known as BatchNormalization. The idea of this layer is as follows:

During training, consider the outputs of the previous layer $\mathbf{a}_i = (a_i^{(1)}, \dots, a_i^{(N)})$ for each element $i \in \{1, \dots, M\}$ of the input batch. Compute the mean μ_j and variance σ_j^2 over each input dimension j. Use the resulting statistics to normalize the output of the previous layer. Finally, rescale the resulting vector with a learned constant γ and shift it by another learned constant β.

(a) Write down the mathematical expression for the BatchNormalization layer. What are its learnable parameters?

(b) Compute the gradient of the loss $\mathcal{L}$ with respect to the input of the BatchNormalization $\mathbf{a}_i$ layer.

(c) At test time, the batch size is usually 1. So, it is not meaningful (or even possible) to compute mean / variance. How would you implement a layer like this?

Solution

For the purpose of batch normalization, we can consider each output neuron individually. Thus, we will simplify our notation and write a_i, y_i, … instead of $a_i^{(j)}$, $y_i^{(j)}$, … respectively.

(a) The forward pass is given by the following equations

$$\begin{aligned}
\mu_B &:= \frac{1}{M}\sum_{i=1}^{m} a_i, && \text{(mini-batch mean)} \\
\sigma_B^2 &:= \frac{1}{M}\sum_{i=1}^{M}(a_i - \mu_B)^2, && \text{(mini-batch variance)} \\
\hat{a}_i &:= \frac{a_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}, && \text{(normalize)} \\
y_i &:= BN_{\gamma,\beta}((a_i)_{i=1}^M) := \gamma\hat{a}_i + \beta. && \text{(scale and shift)}
\end{aligned}$$

The entire layer is defined as

$$BN(\mathbf{a}_1, \dots \mathbf{a}_M) = \left(BN_{\gamma^{(1)},\beta^{(1)}}((a_i^{(1)})_{i=1}^M), \dots, BN_{\gamma^{(N)},\beta^{(N)}}((a_i^{(N)})_{i=1}^M)\right)$$

where $\gamma^{(1)}, \dots, \gamma^{(N)}$ and $\beta^{(1)}, \dots, \beta^{(N)}$ are learnable parameters.

(b) The derivatives can be expressed using the chain rule where we obtain μ_B, σ_B, $\hat{a}_i$, and y_i during the forward pass while $\partial\mathcal{L}/\partial y_i$ is obtained from earlier steps of the backward pass. The remaining

derivatives are:

$$\frac{\partial \mathcal{L}}{\partial \hat{a}_i} = \frac{\partial \mathcal{L}}{y_i} \cdot \gamma,$$
$$\frac{\partial \mathcal{L}}{\partial \sigma_B^2} = \sum_{i=1}^{M} \frac{\partial \mathcal{L}}{\partial \hat{a}_i} \cdot (a_i - \mu_B) \cdot \frac{-1}{2}(\sigma_B^2 + \epsilon)^{-3/2},$$
$$\frac{\partial \mathcal{L}}{\partial \mu_B} = \left(\sum_{i=1}^{M} \frac{\partial \mathcal{L}}{\partial \hat{a}_i} \cdot \frac{-1}{\sqrt{\sigma_B^2 + \epsilon}} \right) + \frac{\partial \mathcal{L}}{\partial \sigma_B^2} \cdot \frac{\sum_{i=1}^{M} -2(a_i - \mu_B)}{M},$$
$$\frac{\partial \mathcal{L}}{\partial a_i} = \frac{\partial \mathcal{L}}{\partial \hat{a}_i} \cdot \frac{1}{\sqrt{\sigma_B^2 + \epsilon}} + \frac{\partial \mathcal{L}}{\partial \sigma_B^2} \cdot \frac{2(a_i - \mu_B)}{M} + \frac{\partial \mathcal{L}}{\partial \mu_B} \cdot \frac{1}{M},$$
$$\frac{\partial \mathcal{L}}{\partial \gamma} = \sum_{i=1}^{M} \frac{\partial \mathcal{L}}{\partial y_i} \cdot \hat{a}_i,$$
$$\frac{\partial \mathcal{L}}{\partial \beta} = \sum_{i=1}^{M} \frac{\partial \mathcal{L}}{\partial y_i}.$$

Here ϵ is a small constant which is added in practice to the variance to avoid division by zero. It is actually not part of the derivative.

Exercise 3 - Autodiff Modes

Consider the function $F(x) = f_3(f_2(f_1(x))$ and assume you also know the derivatives f_i' for all f_i.

(a) Apply the chain rule to express $F'(x)$ in terms of f_i's and f_i.

(b) Write down the pseudocode for computing $F'(x)$ using the forward mode and the reverse mode respectively. Assume all functions to be scalar functions of a scalar variable, i.e. $f_i : \mathbb{R} \to \mathbb{R}$.

(c) If you simply ask your interpreter / compiler to evaluate the expression in (a), will the computation be in forward mode, reverse mode, or neither of the modes? Why? You can assume that your interpreter / compiler does not do any caching or optimization and simply evaluates the expression from left to right. Does anything change if you assume that your interpreter caches results that have been computed before?

Solution

(a) For better readability we will write $(g \circ h)(x)$ for $g(h(x))$. By applying the chain rule, we obtain

$$\begin{aligned} F'(x) &= (f_3 \circ f_2 \circ f_1)'(x) \\ &= (f_2 \circ f_1)'(x) \cdot (f_3' \circ f_2 \circ f_1)(x) \\ &= f_1'(x) \cdot (f_2' \circ f_1)(x) \cdot (f_3' \circ f_2 \circ f_1)(x) \end{aligned}$$

(b) First, let's start with the forward mode

```
d = f1'(x)
v = f1(x)
d = f2'(x) * d
v = f2(v)
d = f3'(v) * d
```

Now, for the reverse mode, we first do a "forward pass" before computing gradients:

```
v1 = f1(x)
v2 = f2(v1)
d = f3'(v2)
d = d*f2'(v1)
d = d*f1'(x)
```

(c) Simply evaluating the expression in (a) is not in line with any of the modes. It also involves repeated computations because $f_1(x)$ will be computed twice. Now, if we allow for caching of ingtermediate results, this doubling of compugtaiton disappears. The order written above will then be in line with forward move automatic differentiation. However, this is specific to our example and in general not true.

Exercise 4 - GloVe Embeddings

Open the notebook presented in class and work through it by trying some of the ideas presented therein for different word combinations.