Deep Generative Models
Homework Set 1 (Sol)

Mohammad Parsa Dini

October 12, 2024

Problem 1

A. We want to do MLE parameter estimation for a linear autoregressive model $AR(p)$. The $AR(p)$ is described by: $\forall t : y_t = \sum_{i=1}^{p} \phi_i y_{t-i} + \epsilon_t$ where ϵ_t's are identically and independently from $\mathcal{N}(0, \sigma^2)$ distribution. We assume that we y_0 is known and fixed, so $p(y_0) = 1$. By chain rule we have:

$$p(y_0, ..., y_n) = \prod_{t=1}^{n} p(y_t|y_{t-1}, ..., y_1) = \prod_{t=p+1}^{n} p(y_t|y_{t-1}, ..., y_{t-p}) = \prod_{t=1}^{n} \mathcal{N}(\sum_{i=1}^{p} \phi_i y_{t-i}, \sigma^2)$$

This is due to the fact that $\forall k : \mathbb{E}[y_k|y_{k-1}, ..., y_{k-p}] = \sum_{i=1}^{p} \phi_i y_{k-i}$ and $\text{Var}(y_k|y_{k-1}, ..., y_{k-p}) = \sigma^2$. Therefore:

$$p(y_0, ..., y_n) = \prod_{t=p+1}^{n} (\frac{1}{2\pi\sigma^2})^{\frac{1}{2}} \exp(-\frac{1}{2\sigma^2}(y_t - \sum_{i=1}^{p} \phi_i y_{t-i})^2)$$

Now we derive the Log-Likelihood function which is(let $\theta = (\sigma, \{\phi_i\}_{i=1}^{p})$:

$$L(\theta) = -\log p(y_0, ..., y_n) = \frac{n-p}{2} \log(2\pi\sigma^2) + \frac{1}{2\sigma^2} \sum_{t=p+1}^{n} (y_t - \sum_{i=1}^{p} \phi_i y_{t-i})^2$$

B. Now in order to maximize the log-likelihood function, we differentiate $L(\theta)$ with respect to σ and ϕ_i's which implies that:

$$\frac{\partial L(\theta)}{\partial \sigma} = (n-p)\frac{1}{\sigma} - \frac{1}{\sigma^3} \sum_{t=p+1}^{n} (y_t - \sum_{i=1}^{p} \phi_i y_{t-i})^2 \Rightarrow \sigma = \sqrt{\frac{1}{n-p} \sum_{t=p+1}^{n} (y_t - \sum_{i=1}^{p} \phi_i y_{t-i})^2}$$

$$\forall j \in \{1, 2, .., p\} : \frac{\partial L(\theta)}{\partial \phi_j} = \frac{y_{t-j}}{\sigma^2} \sum_{t=p+1}^{n} (y_t - \sum_{i=1}^{p} \phi_i y_{t-i}) = 0 \Rightarrow (y_{t-j} = 0) \vee (\sum_{t=p+1}^{n} (y_t - \sum_{i=1}^{p} \phi_i y_{t-i}) = 0)$$

which clearly can be solved for ϕ_i's by having $\{y_0, .., y_n\}$ values.

Problem 2

A. In an autoregressive model we are given a sequence of continuous random variables $x = (X_1, .., X_T)$, and each x_t given $x_1, .., x_{t-1}$ comes from $\mathcal{N}(f_\theta(x_1, ..., x_{t-1}), \sigma^2)$. We wish to get the likelihood function:

$$L(\theta) = \log p(x_1, ..., x_n) = \log(p(x_1) \prod_{k=2}^{T} p(x_k|x_1, .., x_{k-1})) = \log(\prod_{k=1}^{T} \mathcal{N}(f_\theta(x_1, ..., x_{k-1}), \sigma^2))$$

$$L(\theta) = \log \prod_{k=1}^{T} (\frac{1}{2\pi\sigma^2})^{\frac{1}{2}} exp(-\frac{(x_k - f_\theta(x_1, ..., x_{k-1}))^2}{2\sigma^2}) = -\frac{T}{2}\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_{k=1}^{T}(x_k - f_\theta(x_1, ..., x_{k-1}))^2$$

In order to maximize the likelihood function, one must minimize the term $\sum_{k=1}^{T}(x_k - f_\theta(x_1, ..., x_{k-1}))^2$. In other words we have an MSE loss function as $l(\theta) = \sum_{k=1}^{T}(x_k - f_\theta(x_1, ..., x_{k-1})^2$. Thus, in order to train the neural network $f\theta$, we must compute the gradient of the loss. Therefore, we get:

$$\nabla_\theta L(\theta) = \frac{-1}{\sigma^2}\sum_{k=1}^{T}(x_k - f_\theta(x_1, ..., x_{k-1}))$$

So in order to find θ, we can use an optimizer like SGD to update weights θ each step($\theta_{t+1} \rightarrow \eta\nabla_\theta L(\theta_t)$.

B. Generating a new sequence $(x'_1, ..., x'_T)$ can't be processed in parallel due to sequential dependency. Since for each variable x'_k (that $x'_k|x'_1, ..., x'_{k-1} \sim \mathcal{N}(f_\theta(x'_1, ..., x'_{k-1}), \sigma^2))$), the previous variables play their part by setting the mean of the normal distribution: $\mathbb{E}x_t = f_\theta(x_1, ..., x_{k-1})$. Thus, for generating each variable, all the variables prior to that one, must be generated first.

C. Firstly, we take this as a lemma that for independent gaussian distributions $p_\theta \sim \mathcal{N}(\mu_1, \sigma^2)$ and $q_\Phi \sim \mathcal{N}(\mu_2, \sigma^2)$, then their KL-divergence will be $D_{KL}(p_\theta||q_\Phi) = \frac{(\mu_1 - \mu_2)^2}{2\sigma^2}$. Using chain rule we obtain:

$$D_{KL}(p_\theta||q_\Phi) = \mathbb{E}_{x\sim p_\theta}[\log(\frac{p_\theta(x)}{q_\Phi(x)}] = \mathbb{E}_{x\sim p_\theta}[\log(\frac{p_\theta(x_1)\prod_{k=2}^{T} p_\theta(x_k|x_1, .., x_{k-1}))}{q_\Phi(x_1)\prod_{k=2}^{T} q_\Phi(x_k|x_1, .., x_{k-1})}]$$

$$D_{KL}(p_\theta||q_\Phi) = \mathbb{E}_{x\sim p_\theta}\log[\frac{p_\theta(x_1)}{q_\Phi(x_1)}] + \mathbb{E}_{x\sim p_\theta}\sum_{k=2}^{T}\log(\frac{p_\theta(x_k|x_1, .., x_{k-1})}{q_\Phi(x_k|x_1, .., x_{k-1})}$$

$$D_{KL}(p_\theta||q_\Phi) = \mathbb{E}_{x_1\sim p_\theta} D_{KL}(p_\theta(x_1)||q_\Phi(x_1)) + \sum_{k=2}^{T}\mathbb{E}_{x_1,...,x_{k-1}\sim p_\theta} D_{KL}(p_\theta(x_k|x_1, ..., x_{k-1})||q_\Phi(x_k|x_1, ..., x_{k-1})$$

Now, by using the lemma we mentioned earlier we get(note that $\mathbb{E}_{x_k\sim p_\theta}[x_k|x_1, ..., x_{k-1}] = f_\theta(x_1, ..., x_{k-1})$ and also $\mathbb{E}_{x_k\sim q_\Phi}[x_k|x_1, ..., x_{k-1}] = g_\Phi(x_1, ..., x_{k-1})$:

$$D_{KL}(p_\theta||q_\Phi) = \mathbb{E}_{x_1,...,x_{k-1}\sim p_\theta}\frac{1}{2\sigma^2}\sum_{k=1}^{T}(f_\theta(x_1, ..., x_{k-1}) - g_\Phi(x_1, ..., x_{k-1}))^2$$

Now obviously, in order to generate a sample, we need to have all the prior samples generated due to the sequential dependencies. Furthermore, if x is going to be a high dimensional random vector($T \rightarrow \infty$), then getting $x_k|x_1, ..., x_k$ will be harder, since the neural network corresponding with x_k given $x_1, ..., x_{k-1}$ will be much complex. Additionally, taking expectations will be harder as k grows, since we need to integrate over joint distribution of k variables.
Now, instead of taking the expectation which is hard, we can use "Monte-Carlo" simulation. Since we know the distribution, we will generate a long sequence of data $\{x_k^i\}_{i=1}^M \sim p_\theta$ and $\{y_k^i\}_{i=1}^M \sim q_\Phi$ and then calculating the square of their average difference as below:

$$D_{KL}(p_\theta||q_\Phi) \approx \frac{1}{M}\sum_{m=1}^{M}\frac{1}{2\sigma^2}\sum_{k=1}^{T}(f_\theta(x_1^m, ..., x_{k-1}^m) - g_\Phi(x_1^m, ..., x_{k-1}^m))^2$$

D. Well, the AR models cannot capture long-term dependencies for a couple of reasons. Firstly, we will run out of storage quickly as we go further and also computations will be more complex(even if we use methods as above such as Monte-Carlo distribution).
Furthermore, "gradient vanishing" is another issue, as the network's depth increases, the sensitivity of

the loss function with respect to the early steps will grow smaller and as a result it can't capture those long-term dependencies.
To address this issue, we can use an RNN structure. Each RNN model maintains a hidden state that encapsulates past information. Rather than referencing the entire history of observations $p(x_k|x_1, \ldots, x_{k-1})$, it relies on a compressed version known as the hidden state $p(x_k|h_{k-1})$.

E. The degradation of the model could be caused by the fact that the model's structure was not good so that it couldn't capture long-term dependencies. Furthermore, given the sequential structure of the model for generation of x_k, if an error ocurres in prior samples, it could propagate and accumulate and wreck havoc on the future samples.

Problem 3

Since the π_c's must sum to 1 over c's, we can use the idea of the softmax to set them as($\theta = (W, b, v, u, d)$):

$$\pi_i^c = \frac{\exp(W_c^T h_i + b_i^c)}{\sum_{k=1}^{C} \exp(W_k^T h_i + b_k^c)}$$

and since $p(x_i|x_{1:i-1}; \theta) \sim \mathcal{N}(v_i^T h_i + b_i, e^{u_i^T h_i + d_i})$ and $p(x_i|x_{1:i-1}) \sim \sum_{c=1}^{C} \pi_i^c \mathcal{N}(\mu_i^c, \sigma_i^c)$, we will formulate μ_i^c and σ_i^c as;

$$\mu_i^c = (v_i^c)^T h_i + b_i^c$$
$$\sigma_i^c = \exp((u_i^c)^T h_i + d_i^c)$$

The number of parameters for π is $C \times (D + 1)$ since $\dim(\pi_i) = C$, $\dim(b_i) = C$ and $W \in \mathbb{R}^{C \times D}$. Likewise, the number of parameters of μ and σ will be $C \times (D + 1)$. At last, the total number of required parameter is $3C \times (D + 1)$.

1 Problem 4

A. if $x \sim \mathcal{N}(0, 2)$, then $\mathbb{E}[(x - 0)^2] = 2$ and $\mathbb{E}[x] = 0$. Thus these two will suggest that:

$$\mathbb{E}_{x \sim \mathcal{N}(0,2)}[x^2 + x + 1] = 2 + 0 + 1 = 3$$

B. We know that for a function of a random variable x: $\text{Var}(f(x)) = \mathbb{E}[f(x)^2] - (\mathbb{E}[f(x)])^2$ So replacing the expectation with "Monte-Carlo" will result in:

$$\text{Var(f(x))} \approx \frac{1}{M} \sum_{m=1}^{M} f(x_m)^2 - (\frac{1}{M} \sum_{m=1}^{M} f(x_m))^2$$

C.