STATS 232A Project 5: Generator and descriptor

1 Generator: real inference

The model has the following form:

$$Y = f(Z; W) + \epsilon \tag{1}$$

$$Z \sim N(0, I_d),\ \epsilon \sim \mathrm{N}(0, \sigma^2 I_D),\ d < D. \tag{2}$$

$f(Z; W)$ maps latent factors into image Y, where W collects all the connection weights and bias terms of the ConvNet.

Adopting the language of the EM algorithm, the complete data model is given by

$$\log p(Y, Z; W) = \log[p(Z)p(Y|Z, W)] \tag{3}$$

$$= -\frac{1}{2\sigma^2}||Y - f(Z; W)||^2 - \frac{1}{2}||Z||^2 + \text{const.} \tag{4}$$

The observed-data model is obtained by intergrating out Z: $p(Y; W) = \int p(Z)p(Y|Z, W)dZ$. The posterior distribution of Z is given by $p(Z|Y, W) = p(Y, Z; W)/p(Y; W) \propto p(Z)p(Y|Z, W)$ as a function of Z.

We want to minimize the observed-data log-likelihood, which is $L(W) = \sum_{i=1}^{n} \log p(Y_i; W) = \sum_{i=1}^{n} \log \int p(Y_i, Z_i; W)dZ_i$. The gradient of $L(W)$ can be calculated according to the following well-known fact that underlies the EM algorithm:

$$\frac{\partial}{\partial W} \log p(Y; W) = \frac{1}{P(Y; W)} \frac{\partial}{\partial W} \int p(Y, Z; W)dZ \tag{5}$$

$$= \mathrm{E}_{p(Z|Y,W)}[\frac{\partial}{\partial W} \log p(Y, Z; W)]. \tag{6}$$

The expectation with respect to $p(Z|Y, W)$ can be approximated by drawing samples from $p(Z|Y, W)$ and then compute the Monte Carlo average.

The Langevin dynamics for sampling $Z \sim p(Z|Y, W)$ iterates

$$Z_{\tau+1} = Z_\tau + \delta U_\tau + \frac{\delta^2}{2}[\frac{1}{\sigma^2}(Y - f(Z_\tau; W))\frac{\partial}{\partial Z} f(Z_\tau; W) - Z_\tau], \tag{7}$$

where τ denotes the time step for the Langevin sampling, δ is the step size, and U_τ denotes a random vector that follows $\mathrm{N}(0, I_d)$.

The stochastic gradient algorithm can be used for learning, where in each iteration, for each Z_i, only a single copy of Z_i is sampled from $p(Z_i|Y_i, W)$ by running a finite number of steps of Langevin dynamics starting from the current value of Z_i, i.e., the warm start. With $\{Z_i\}$ sampled in this manner, we can update the parameter W based on the gradient $L'(W)$, whose Monte Carlo approximation is:

$$
\begin{aligned}
L'(W) &\approx \sum_{i=1}^{n} \frac{\partial}{\partial W} \log p(Y_i, Z_i; W) && (8)\\
&= -\sum_{i=1}^{n} \frac{\partial}{\partial W} \frac{1}{2\sigma^2} ||Y_i - f(Z_i; W)||^2 && (9)\\
&= \sum_{i=1}^{n} \frac{1}{\sigma^2} (Y_i - f(Z_i; W)) \frac{\partial}{\partial W} f(Z_i; W). && (10)
\end{aligned}
$$

Algorithm 1 describes the details of the learning and sampling algorithm.

Algorithm 1 Generator: real inference

Input:
(1) training examples $\{Y_i, i = 1, ..., n\}$,
(2) number of Langevin steps l,
(3) number of learning iterations T.

Output:
(1) learned parameters W,
(2) inferred latent factors $\{Z_i, i = 1, ..., n\}$.

Let $t \leftarrow 0$, initialize W.
Initialize Z_i, for $i = 1, ..., n$.
repeat
    Inference step: For each i, run l steps of of Langevin dynamics to sample $Z_i \sim p(Z_i|Y_i, W)$ with warm start, i.e., starting from the current Z_i, each step follows equation 7.
    Learning step: Update $W \leftarrow W + \gamma_t L'(W)$, where $L'(W)$ is computed according to equation 10, with learning rate γ_t.
    Let $t \leftarrow t + 1$.
until $t = T$

1.1 TO DO

For the lion-tiger category, learn a model with 2-dim latent factor vector. Fill the blank part of `./GenNet/GenNet.py`. **Show**:

(1) Reconstructed images of training images, using the inferred z from training images.

(2) Randomly generated images, using randomly sampled z.

(3) Generated images with linearly interpolated latent factors from $(-2, 2)$ to $(-2, 2)$. For example, you inperlolate 8 points from $(-2, 2)$ for each dimension of z. Then you will get a 8×8 panel of images. You should be able to seee that tigers slight change to lion.

(4) Plot of loss over iteration.

2 Descriptor: real sampling

The descriptor model is as follows:

$$p_\theta(Y) = \frac{1}{Z(\theta)} \exp\left[f_\theta(Y)\right] p_0(Y), \tag{11}$$

where $p_0(Y)$ is the reference distribution such as Gaussian white noise

$$p_0(Y) \propto \exp\left(-\|Y\|^2/2\sigma^2\right) \tag{12}$$

The scoring function $f_\theta(Y)$ is defined by a bottom-up ConvNet whose parameters are denoted by θ. The normalizing constant $Z(\theta) = \int \exp\left[f_\theta(Y)\right] p_0(Y)dY$ is analytically intractable. The energy function is

$$\mathcal{E}_\theta(Y) = \frac{1}{2\sigma^2}\|Y\|^2 - f_\theta(Y). \tag{13}$$

$p_\theta(Y)$ is an exponential tilting of p_0.

Suppose we observe training examples $\{Y_i, i = 1, ..., n\}$ from an unknown data distribution $P_{\text{data}}(Y)$. The maximum likelihood learning seeks to maximize the log-likelihood function

$$L(\theta) = \frac{1}{n}\sum\nolimits_{i=1}^{n} \log p_\theta(Y_i). \tag{14}$$

If the sample size n is large, the maximum likelihood estimator minimizes the Kullback-Leibler divergence $\text{KL}(P_{\text{data}}\|p_\theta)$ from the data distribution P_{data} to the model distribution p_θ. The gradient of $L(\theta)$ is

$$L'(\theta) = \frac{1}{n}\sum\nolimits_{i=1}^{n} \frac{\partial}{\partial\theta} f_\theta(Y_i) - \text{E}_\theta\left[\frac{\partial}{\partial\theta} f_\theta(Y)\right], \tag{15}$$

where E_θ denotes the expectation with respect to $p_\theta(Y)$. The key to the above identity is that $\frac{\partial}{\partial\theta}\log Z(\theta) = \text{E}_\theta[\frac{\partial}{\partial\theta} f_\theta(Y)]$.

The expectation in equation (15) is analytically intractable and has to be approximated by MCMC, such as Langevin dynamics, which iterates the following step:

$$\begin{aligned} Y_{\tau+1} &= Y_\tau - \frac{\delta^2}{2}\frac{\partial}{\partial Y}\mathcal{E}_\theta(Y_\tau) + \delta U_\tau \\ &= Y_\tau - \frac{\delta^2}{2}\left[\frac{Y_\tau}{\sigma^2} - \frac{\partial}{\partial Y} f_\theta(Y_\tau)\right] + \delta U_\tau, \end{aligned} \tag{16}$$

where τ indexes the time steps of the Langevin dynamics, δ is the step size, and $U_\tau \sim \mathrm{N}(0, I)$ is Gaussian white noise. The Langevin dynamics relaxes Y_τ to a low energy region, while the noise term provides randomness and variability. A Metropolis-Hastings step may be added to correct for the finite step size δ. We can also use Hamiltonian Monte Carlo for sampling the generative ConvNet.

We can run $\tilde{n}$ parallel chains of Langevin dynamics according to (16) to obtain the synthesized examples $\{\tilde{Y}_i, i = 1, ..., \tilde{n}\}$. The Monte Carlo approximation to $L'(\theta)$ is

$$\begin{aligned} L'(\theta) &\approx \frac{1}{n}\sum\nolimits_{i=1}^{n} \frac{\partial}{\partial\theta} f_\theta(Y_i) - \frac{1}{\tilde{n}}\sum\nolimits_{i=1}^{\tilde{n}} \frac{\partial}{\partial\theta} f_\theta(\tilde{Y}_i) \\ &= \frac{\partial}{\partial\theta}\left[\frac{1}{\tilde{n}}\sum\nolimits_{i=1}^{\tilde{n}} \mathcal{E}_\theta(\tilde{Y}_i) - \frac{1}{n}\sum\nolimits_{i=1}^{n} \mathcal{E}_\theta(Y_i)\right], \end{aligned} \tag{17}$$

which is used to update θ.

To make Langevin sampling easier, we use mean images of training images as the sampling starting point. That is, we down-sampled each training image to a 1×1 patch, and up-sample this patch to the size of training image. We use cold start for Langevin sampling, i.e., at each iteration, we start sampling from mean images.

Algorithm 2 describes the details of the learning and sampling algorithm.

Algorithm 2 Descriptor: real sampling

Input:
(1) training examples $\{Y_i, i = 1, ..., n\}$,
(2) number of Langevin steps l,
(3) number of learning iterations T.

Output:
(1) estimated parameters θ,
(2) synthesized examples $\{\tilde{Y}_i, i = 1, ..., n\}$.

1: Let $t \leftarrow 0$, initialize θ.
2: **repeat**
3:    For $i = 1, ..., n$, initialize $\tilde{Y}_i$ to be the mean image of Y_i.
4:    Run l steps of Langevin dynamics to evolve $\tilde{Y}_i$, each step following equation (16).
5:    Update $\theta_{t+1} = \theta_t + \gamma_t L'(\theta_t)$, with step size γ_t, where $L'(\theta_t)$ is computed according to equation (17).
6:    Let $t \leftarrow t + 1$.
7: **until** $t = T$

2.1 TO DO

For the egret category, learn a descriptor model. Fill the blank part of `./DesNet/DesNet.py`.
Show:

(1) Synthesized images.
(2) Plot of training loss over iteration.

3 What to submit

Write a report to show your results. And zip the report with your code.