10-708 PGM (Spring 2019): Homework 3

Andrew ID: [your Andrew ID]
Name: [your first and last name]
Collaborators: [Andrew IDs of all collaborators, if any]

1 Variational Autoencoders (Lisa) [65 pts]

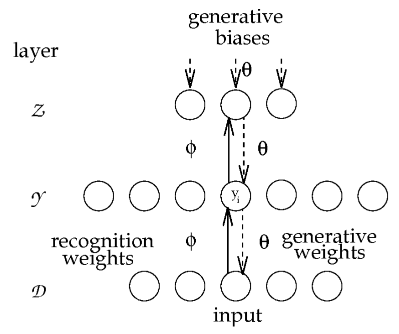

Figure 1: A **Helmholtz machine** [3, 1] contains two networks: (1) bottom-up "recognition" connections ϕ that convert the input data into representations in successive hidden layers, and (2) top-down "generative" connections θ that reconstruct the data from representation in one layer from the representation in the layer above.

The Helmholtz machine (Figure 1) is an architecture that can find hidden structure in data by learning a generative model of the data. Helmholtz machines are usually trained using unsupervised learning algorithms such as the classical **Wake-Sleep** algorithm [3] or the modern **Auto-Encoding Variational Bayes (AEVB)** [4], also known as variational autoencoder.

In this problem, you will (re-)derive and implement the Wake-Sleep and AEVB algorithms. The sections are organized as follows:

(3 pts) Section 1.1: Derivation of the **evidence lower bound objective (ELBO)**, which lowerbounds the data log-likelihood $\log p_\theta(\mathbf{x})$.

(6 pts) Section 1.2: Derivation of the **Wake-Sleep** algorithm, which alternates between the Wake phase and Sleep phase to optimize an estimate of ELBO.

(10 pts) Section 1.3: Derivation of **AEVB**, which optimizes a stochastic estimate of ELBO.

(3 pts) Section 1.4: Short-answer questions on Wake-Sleep and AEVB.

(8 pts) Section 1.5: Derivation of an **alternate lower bound** $\mathcal{L}_k(\mathbf{x})$ for the data log-likelihood, which will be used to evaluate trained models in the next section.

(35 pts) Section 1.6: Implementations and experiments on the MNIST handwritten digits dataset.

For all parts, assume that latent variables $\mathbf{z}$ are distributed according to a prior $p(\mathbf{z}) = N(0, I)$. The Helmholtz machine tries to learn the **recognition** parameters ϕ and **generative** parameters θ such that

$$q_\phi(\mathbf{z} \mid \mathbf{x}) \approx p_\theta(\mathbf{z} \mid x) \propto p_\theta(\mathbf{x}, \mathbf{z})$$

where:

- $q_\phi(\mathbf{z} \mid \mathbf{x})$ is the variational distribution approximating the posterior distribution $p_\theta(\mathbf{z} \mid \mathbf{x})$ for $\mathbf{z}$ given the evidence $\mathbf{x}$. Assume that q_ϕ is parameterized by a Gaussian, i.e., $q_\phi(\mathbf{z} \mid \mathbf{x}) = N(\mathbf{z}; \mu_\phi(\mathbf{x}), \Sigma_\phi(\mathbf{x}))$.
- $p_\theta(\mathbf{x}, \mathbf{z}) = p(\mathbf{z})p_\theta(\mathbf{x} \mid \mathbf{z})$ is the joint probability of $(\mathbf{x}, \mathbf{z})$, where $\mathbf{z} \sim p(\mathbf{z}) = N(0, I)$, and $\mathbf{x} \sim p_\theta(\mathbf{x} \mid \mathbf{z})$ is the likelihood.

Assume $\mathbf{x}$ are binary vectors. In other words, $p_{\boldsymbol{\theta}}(\mathbf{x} \mid \mathbf{z})$ can be modeled with a sigmoid belief net, so the likelihood is of the form $p_\theta(\mathbf{x}|\mathbf{z}) = \text{Bernoulli}(f_\theta(\mathbf{z}))$. Actually, the data points $\mathbf{x}$ in MNIST take values in $[0, 1]$ rather than $\{0, 1\}$, but the loss term $\mathbb{E}_q[p_\theta(\mathbf{x} \mid \mathbf{z})]$ still uses sigmoid cross-entropy loss, which is a common practice [2].

1.1 Evidence Lower Bound Objective (ELBO)

Suppose we want to learn a directed latent variable model (Figure 2) that is able represent a complex distribution $p(\mathbf{x})$ over the data in the following form:

$$p_{\boldsymbol{\theta}}(\mathbf{x}) = \int p_{\boldsymbol{\theta}}(\mathbf{x} \mid \mathbf{z})p(\mathbf{z})d\mathbf{z} \tag{1}$$

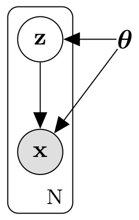

Figure 2: The latent variable model in Problem 1.1.

Suppose we want to approximate the posterior distribution $p_{\boldsymbol{\theta}}(\mathbf{z} \mid \mathbf{x})$ using some variational distribution $q_{\boldsymbol{\phi}}(\mathbf{z} \mid \mathbf{x})$. A tractable way to learn this model is to optimize the **evidence lower bound objective (ELBO)**, also known as the variational lower bound, defined as follows:

$$\begin{aligned}\mathcal{L}(\mathbf{x}) &= \mathbb{E}_{\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}, \mathbf{z}) - \log q_\phi(\mathbf{z} \mid \mathbf{x})] \\ &= \int_z q_\phi(\mathbf{z} \mid \mathbf{x}) \log \frac{p_\theta(\mathbf{x}, \mathbf{z})}{q_\phi(\mathbf{z} \mid \mathbf{x})} d\mathbf{z} \,.\end{aligned}$$

(3 pts) For a single data point $\mathbf{x}^{(i)}$, prove that

$$\log p_\theta(\mathbf{x}^{(i)}) \geq \mathcal{L}(\mathbf{x}^{(i)}).$$

Solution

The above result shows that, for iid data points $\mathbf{x} = \{\mathbf{x}^{(i)}\}_{i=1}^{N}$,

$$\log p_\theta(\mathbf{x}) \overset{\text{iid}}{=} \sum_{i=1}^{N} \log p_\theta(\mathbf{x}^{(i)}) \geq \sum_{i=1}^{N} \mathcal{L}(\mathbf{x}^{(i)}) = \mathcal{L}(\mathbf{x})$$

which gives the ELBO $\mathcal{L}(\mathbf{x})$ on the data log-likelihood $\log p_\theta(\mathbf{x})$.

1.2 Wake-Sleep Algorithm

In this section, we will derive the optimization objectives for the Wake-Sleep algorithm, which decomposes the optimization procedure into two phases:

- **Wake**-phase: Given recognition weights ϕ, we activate the recognition process and update the generative weights θ to increase the probability that they would reconstruct the correct activity vector in the layer below.
- **Sleep**-phase: Given generative weights θ, we activate the generative process and update the recognition weights ϕ to increase the probability that they would produce the correct activity vector in the layer above. Since it has generated the instance, it knows the true underlying causes, and therefore has available the target values for the hidden units that are required to train the bottom-up weights ϕ.

1.2.1 Wake-phase

The Wake-phase fixes the recognition weights ϕ and optimizes a Monte Carlo estimate of ELBO w.r.t. the generative weights θ.

(3 pts) Given N iid data points $\mathbf{x} = \{\mathbf{x}^{(i)}\}_{i=1}^{N}$, show that

$$\theta^* := \arg\max_\theta \mathcal{L}(\mathbf{x}) = \arg\max_\theta \sum_{i=1}^{N} \mathbb{E}_{\mathbf{z} \sim q_\phi(\mathbf{z} \mid \mathbf{x}^{(i)})} \log p_\theta(\mathbf{x}^{(i)} \mid \mathbf{z}) \tag{2}$$

which gives the Wake-phase objective.

Solution

Wake-phase Pseudocode: Given N iid data points $\mathbf{x} = \{\mathbf{x}^{(i)}\}_{i=1}^{N}$, do the following for each $i \in [N]$:

1. Feed $\mathbf{x}^{(i)}$ into the recognition network to get $\mu_\phi(\mathbf{x}^{(i)})$ and $\Sigma_\phi(\mathbf{x}^{(i)})$.
2. Draw L samples $\mathbf{z}_1^{(i)}, \ldots, \mathbf{z}_L^{(i)} \sim q_\phi(\mathbf{z} \mid \mathbf{x}) = N(\mathbf{z}; \mu_\phi(\mathbf{x}), \Sigma_\phi(\mathbf{x}))$.
3. For each $l \in [L]$, feed $\mathbf{z}_l^{(i)}$ into the generative network to get $p_\theta(\mathbf{x} \mid \mathbf{z}_l^{(i)})$.

Finally, use SGD to maximize

$$\max_\theta \sum_{i=1}^{N} \frac{1}{L} \sum_{l=1}^{L} \log p_\theta(\mathbf{x}^{(i)} \mid \mathbf{z}_l^{(i)}) \tag{3}$$

This gives a Monte Carlo estimate of the Wake-phase objective in Eq. (2).

1.2.2 Sleep-phase

The Sleep phase fixes the generative weights θ and updates the recognition weights ϕ. It is generally intractable to directly minimize the KL-divergence term in $\mathcal{L}(\mathbf{x})$ w.r.t. ϕ:

$$\arg\min_{\phi} D_{\mathrm{KL}}\left[q_\phi(\mathbf{z} \mid \mathbf{x}) \parallel p_\theta(\mathbf{z} \mid \mathbf{x})\right] = \arg\min_{\phi} \int_{\mathbf{z}} q_\phi(\mathbf{z} \mid \mathbf{x}) \log \frac{q_\phi(\mathbf{z} \mid \mathbf{x})}{p_\theta(\mathbf{z} \mid \mathbf{x})}\, d\mathbf{z}$$

So instead, the Sleep phase minimizes the KL divergence the wrong way round,

$$\arg\min_{\phi} D_{\mathrm{KL}}\left[p_\theta(\mathbf{z} \mid \mathbf{x}) \parallel q_\phi(\mathbf{z} \mid \mathbf{x})\right].$$

(3 pts) Suppose we sample $\mathbf{z} \sim p(\mathbf{z}) = N(0, I)$, then sample $\mathbf{x} \sim p_\theta(\mathbf{x} \mid \mathbf{z})$. Show that

$$\phi^* := \arg\min_{\phi} D_{\mathrm{KL}}\left[p_\theta(\mathbf{z} \mid \mathbf{x}) \parallel q_\phi(\mathbf{z} \mid \mathbf{x})\right] = \arg\max_{\phi} \mathbb{E}_{p_\theta(\mathbf{x},\mathbf{z})}\left[\log q_\phi(\mathbf{z} \mid \mathbf{x})\right] \tag{4}$$

which gives the Sleep-phase objective.

Solution

Sleep-phase Pseudocode: Let $L \in \mathbb{N}$ be a sample size hyperparameter. For each $l \in [L]$, do the following:

1. Draw $\mathbf{z}^l \sim N(0, I)$.
2. Sample $\mathbf{x}^l$ from the generative network $p_\theta(\mathbf{x} \mid \mathbf{z}^l) = \text{Bernoulli}(f_\theta(\mathbf{z}^l))$.
3. Feed $\mathbf{x}^l$ into the recognition network to get $\mu(\mathbf{x}^l)$ and $\Sigma(\mathbf{x}^l)$.
4. Compute $q_\phi(\mathbf{z}^l \mid \mathbf{x}^l) = N(\mathbf{z}^l; \mu(\mathbf{x}^l), \Sigma(\mathbf{x}^l))$.

Finally, do SGD to maximize

$$\max_{\phi} \frac{1}{L} \sum_{l=1}^{L} \log q_\phi(\mathbf{z}^l \mid \mathbf{x}^l) \tag{5}$$

This gives a Monte Carlo estimate of the Sleep-phase objective in Eq. (4).

1.3 Autoencoding Variational Bayes (AEVB)

In this section, you will derive the optimization procedure for Auto-Encoding Variational Bayes (AEVB). Unlike Wake-Sleep, AEVB avoids the two-stage optimization procedure and instead optimizes a stochastic estimate of ELBO directly w.r.t. to parameters θ of the generative model (generation network) and parameters ϕ of the variational distribution (recognition network).

(3 pts) For a given data point $\mathbf{x}^{(i)}$, show that ELBO can be rewritten as

$$\mathcal{L}(\mathbf{x}^{(i)}) = -D_{\mathrm{KL}}\left[q_\phi(\mathbf{z} \mid \mathbf{x}^{(i)}) \parallel p(\mathbf{z})\right] + \mathbb{E}_{\mathbf{z} \sim q_\phi(\mathbf{z} \mid \mathbf{x}^{(i)})}[\log p_\theta(\mathbf{x}^{(i)} \mid \mathbf{z})]. \tag{6}$$

Solution

Equation (6) gives a stochastic estimator for ELBO:

$$\tilde{\mathcal{L}}(\mathbf{x}^{(i)}) = -D_{\mathrm{KL}}\left[q_\phi(\mathbf{z} \mid \mathbf{x}^{(i)}) \parallel p(\mathbf{z})\right] + \frac{1}{L}\sum_{l=1}^{L}[\log p_\theta(\mathbf{x}^{(i)} \mid \mathbf{z}^{(i,l)})] \tag{7}$$

where $\{\mathbf{z}^{(i,l)}\}_{l=1}^{L}$ are sampled from $q_\phi(\mathbf{z} \mid \mathbf{x}^{(i)})$. The AEVB algorithm optimizes this stochastic estimate of ELBO using a Monte Carlo gradient estimate.

In order to optimize the AEVB objective in Eq. (7) efficiently, we use a **reparameterization trick** to rewrite $\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\cdot]$ such that the Monte Carlo estimate of the expectation is differentiable w.r.t. ϕ. More specifically, we reparameterize the latent variable

$$\mathbf{z} \sim q_\phi(\mathbf{z} \mid \mathbf{x}^{(i)}) = N(\mathbf{z} \mid \mu_\phi(\mathbf{x}^{(i)}), \Sigma^2_\phi(\mathbf{x}^{(i)}))$$

as a deterministic function of the input $\mathbf{x}^{(i)}$ and an auxiliary noise variable ϵ:

$$\mathbf{z} = \mu_\phi(\mathbf{x}^{(i)}) + \Sigma_\phi(\mathbf{x}^{(i)}) \odot \epsilon \qquad\qquad \epsilon \sim N(0, I) \tag{8}$$

where $\odot$ signifies an element-wise product, and $\Sigma_\phi(\mathbf{x}^{(i)})$ is a vector of the same size as z.

(4 pts) Using this reparameterization, show that the AEVB objective in Eq. (7) can be rewritten as

$$\tilde{\mathcal{L}}(\mathbf{x}^{(i)}) = \frac{1}{2}\sum_{j=1}^{J}\left(1 + \log(\Sigma^2_{(i),j}) - \mu^2_{(i),j} - \Sigma^2_{(i),j}\right) + \frac{1}{L}\sum_{l=1}^{L}\log p_\theta(\mathbf{x}^{(i)} \mid \mathbf{z}^{(i,l)}). \tag{9}$$

where $\mu_{(i)} := \mu_\phi(\mathbf{x}^{(i)})$ and $\Sigma_{(i)} := \Sigma_\phi(\mathbf{x}^{(i)})$.

Solution

The AEVB optimization procedure works as follows:

1. For each $l \in [L]$, draw $\epsilon^{(l)} \sim N(0, I)$, and compute $\mathbf{z}^{(i,l)} := \mu_{(i)} + \Sigma_{(i)} \odot \epsilon^{(l)}$.
2. Optimize the AEVB objective in Eq. (9) w.r.t. μ, Σ, and θ.

(3 pt) Derive the gradients of the AEVB objective in Eq. (9) w.r.t. $\mu_{(i),j}$, $\Sigma_{(i),j}$, and θ. (For the gradient w.r.t. θ, you can leave the answer in terms of $p_\theta(\mathbf{x}^{(i)} \mid \mathbf{z}^{(i,l)})$.)

Solution

1.4 Short-answer Questions

(1 pts) Wake-Sleep requires a concurrent optimization of two objective functions, which together do not correspond to the optimization of (a bound of) the marginal likelihood. There is no guarantee that optimizing the Wake-Sleep objectives leads to a decrease in the free energy because:

(Choose 0-2 of the following choices)

A. The sleep phase trains the recognition model to invert the generative model for input vectors that are distributed according to the generative model rather than according to the real data.

B. The sleep phase learning does not follow the correct gradient.

Solution

(1 pt) Between Wake-Sleep and AEVB, which algorithm(s) can be applied to models with discrete latent variables?

Solution

(1 pts) (True or False) Wake-Sleep and AEVB have the same computational complexity per datapoint.

Solution

AEVB is an elegant way to link directed graphical models to neural networks, and is theoretically appeasing because we optimise a (stochastic estimate of the) bound on the likelihood of the data. If the approximations made while performing variational bayes are valid, the training algorithm is guaranteed to increase the likelihood of the generative model. Moreover, there is a clear and recognized way to evaluate the quality of the model using the log-likelihood (either estimated by importance sampling or lower-bounded).

For i.i.d. datasets with continuous latent variables per datapoint, posterior inference for AEVB can be made especially efficient by fitting an approximate inference model (also called a recognition model) to the intractable posterior using the proposed lower bound estimator.

1.5 An Alternate Lower Bound on the Log-Likelihood

To compare trained models, we could simply look at the values of the lower bound. However, the bound could be loose and hence the numbers could be misleading. Here, we derive and prove a tighter approximation of the lower bound on the marginal likelihood, defined as follows:

$$\mathcal{L}_k(\mathbf{x}) = \mathbb{E}_{\mathbf{z}^{(1)},\ldots,\mathbf{z}^{(k)} \sim q_\phi(\mathbf{z}|\mathbf{x})} \left[\log \frac{1}{k} \sum_{i=1}^{k} \frac{p_\theta(\mathbf{x}, \mathbf{z}^{(i)})}{q_\phi(\mathbf{z}^{(i)} \mid \mathbf{x})} \right]. \tag{10}$$

(4 pts) Prove that $\log p(\mathbf{x}) \geq \mathcal{L}_k(\mathbf{x})$ for any $k \in \mathbb{N}$. (*Hint*: Use Jensen's inequality.)

Solution

(4 pts) Prove that $\mathcal{L}_{k+1}(\mathbf{x}) \geq \mathcal{L}_k(\mathbf{x})$ for any $k \in \mathbb{N}$. You can use the following lemma without proof:

Lemma: Let $I_k \subset [k+1] := \{1, \ldots, k+1\}$ with $|I_k| = k$ be a uniformly distributed subset of distinct indices from $[k+1]$. Then for any sequence of numbers $a_1, \ldots, a_{k+1}$,

$$\mathbb{E}_{I_k} \left[\frac{\sum_{i \in I_k} a_i}{k} \right] = \frac{\sum_{i=1}^{k+1} a_i}{k+1} \tag{11}$$

Solution

The above two results show that

$$\log p(\mathbf{x}) \geq \mathcal{L}_{k+1}(\mathbf{x}) \geq \mathcal{L}_k(\mathbf{x}).$$

However, the above inequalities do not guarantee $\mathcal{L}_k(\mathbf{x}) \to \log p(\mathbf{x})$ when $k \to \infty$. (The proof is left as an exercise to the reader. Or you can come to my office hours.)

1.6 Experiments (35 pts)

We provide a Jupyter notebook which already contains a working implementation of AEVB:

`https://colab.research.google.com/drive/1nmPXgoNLUKxj-VTwB0l0956Swiv5jrVF`

Please follow the instructions in the Colab notebook. All provided code will run as is, but you will also need to complete the following TODO's:

1. Implement the Wake-Sleep algorithm by modifying the provided AEVB code.
2. Implement the lower bound metric $\mathcal{L}_{100}$ for $k = 100$ as defined in Eq. (10).
3. Submit your modified .ipynb notebook on Gradescope.

You will use these implementations to complete the remaining sections.

1.6.1 Training

Train both Wake-Sleep and AEVB on the MNIST dataset for 100 epochs (using batch size 100.

If your Wake-Sleep implementation does not seem to learn a reasonable representation after 100 epochs, you can train it for longer than 100 epochs. If you show that the Wake-Sleep training losses and $\mathcal{L}_{100}^{\text{test}}$ continue to decrease past 100 epochs, and it yields better visualization results, then it would show that your Wake-Sleep implementation is working, and you will still get full points. Report the number of epochs used to train your Wake-Sleep model.

If you encounter `nan`'s even after adjusting your hyperparameters, the problem may be your loss implementation that causes numerical instability issues.

(12 pts) For both Wake-Sleep and AEVB, plot $\mathcal{L}_{100}^{\text{test}}$ vs. the epoch number. To save computation time, you can evaluate $\mathcal{L}_{100}$ every 10 epochs. Which algorithm converged faster? (Grading: 2 points for Wake-Sleep. You can still get 10 points without implementing Wake-Sleep, if you implement $\mathcal{L}_k$ correctly. For each algorithm, you can also get 3 extra credit points for plotting $\mathcal{L}_{100}^{\text{train}}$ vs. the epoch number.)

Solution

(8 pts) For Wake-Sleep, also plot the training losses for the wake-phase and sleep-phase vs. the epoch number. Specify the hyperparameters used, such as the learning rates for Wake-phase and Sleep-phase, and the sample size used to compute the Monte Carlo estimate of the gradient for the Sleep objective.

Solution

1.6.2 Reconstructed Images

Next, we provide code that samples 100 MNIST images from the test set, uses the recognition network to map them to latent space, then applies the generator network to reconstruct the images.

(5 pts) Run this code to visualize these reconstruction samples $\tilde{\mathbf{x}}^{(1)}, \ldots, \tilde{\mathbf{x}}^{(100)}$ on a 10×10 tile grid. Also visualize the original MNIST images $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(100)}$ on a 10×10 tile grid. Briefly compare the results for Wake-Sleep vs. AEVB. (Grading: 2 points for Wake-Sleep, 2 points for AEVB, 1 point for comparing the two.)

Solution

1.6.3 Latent Space Visualization (Part 1)

Since we have specifically chosen the latent space to be 2-dimensional, now we can easily visualize the learned latent manifold of digits. We provide code that samples 5000 MNIST images from the test set, and visualize their latent representations as a scatter plot, where colors of the points correspond to the digit labels.

(5 pts) Run this code to visualize the latent space scatterplot. Briefly compare the results for Wake-Sleep vs. AEVB. (Grading: 2 points for Wake-Sleep, 2 points for AEVB, 1 point for comparing the two.)

Solution

1.6.4 Latent Space Visualization (Part 2)

Finally, we provide code that uses the generator network to plot reconstructions at the positions in the latent space for which they have been generated.

(5 pts) Run this code to visualize the latent space reconstructions. Briefly compare the results for Wake-Sleep vs. AEVB. (Grading: 2 points for Wake-Sleep, 2 points for AEVB, 1 point for comparing the two.)

Solution

2 Dissecting GANs (Hao) [35 pts]

We have covered Generative Adversarial Networks (GANs) in lectures. In this problem, we will rethink a few building blocks of GANs. We will start with some simple proofs about the original GANs (section 2.1). Then we will investigate the loss function of GANs (section 2.2), and extend it from two players to three players (section 2.3). Lastly, we will compare VAEs with GANs on a few statements.

2.1 Warm-up: Optimality of GANs [9 pts]

Let us first introduce a few notations before we can start. In a typical GAN framework, we are interested in learning a generator distribution p_g over training data $\boldsymbol{x}$. To parametrize p_g, we introduce the generator G, which is usually instantiated as a differentiable neural network. G maps input noise variables $\boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z})$ to data space as $G(\boldsymbol{z}; \theta_g) \sim p_g(\boldsymbol{x})$, where $p_{\boldsymbol{z}}(\boldsymbol{z})$ is a prior of $\boldsymbol{z}$. In order to train G (i.e. its parameters θ_g), we define a second neural network $D(\boldsymbol{x}; \theta_d)$, which takes a sample $\boldsymbol{x}$ as input, and outputs the probability that $\boldsymbol{x}$ comes from the real data distribution $p_d(\boldsymbol{x})$ rather than $p_g(\boldsymbol{x})$. The training is conducted as follows: D is trained to correctly classify samples from the training data as real, and samples from G as fake; simultaneously, G is trained to generate samples as real as possible in order to fool D. In a mathematical form, we represent the training as a minimax game with value function $V(G, D)$:

$$\min_G \max_D V(D, G) = \mathbb{E}_{\boldsymbol{x} \sim p_d(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]. \tag{12}$$

Assuming both G and D have infinite capacity (i.e. under non-parametric assumptions).

[**3 pts**] Show that the optimization problem in Eq. 12 is equivalent with the following problem

$$\min_G C(G) = \mathbb{E}_{\boldsymbol{x} \sim p_d(\boldsymbol{x})} \left[\log \frac{p_d(\boldsymbol{x})}{p_d(\boldsymbol{x}) + p_g(\boldsymbol{x})} \right] + \mathbb{E}_{\boldsymbol{x} \sim p_g(\boldsymbol{x})} \left[\log \frac{p_g(\boldsymbol{x})}{p_d(\boldsymbol{x}) + p_g(\boldsymbol{x})} \right]. \tag{13}$$

Solution

[**2 pts**] Show that

$$C(G) = -\log(4) + 2 \cdot JSD(p_d || p_g),$$

where JSD is the Jensen-Shannon Divergence between two distributions.

Solution

[**1 pt**] What is the global minimum of the problem in Eq. 13? When will it be achieved?

Solution

The above results suggest that if we have perfect estimators of p_g with infinity capacity, we shall be able to match p_d using GANs. At that point, we say the minimax game in Eq. 12 reaches its *equilibrium*.

In practice, we instantiate D and G using neural networks, which are often highly non-convex, and optimize the NN parameters θ_g rather than p_g itself. The objective can be optimized via gradient-based methods – similar to VAE, a Monte Carlo estimate of the gradients can be obtained by sampling stochastic training batches (from $p_d(\boldsymbol{x})$ and $p_{\boldsymbol{z}}(\boldsymbol{z})$), feeding through the neural networks, and performing backpropagation. Early in learning, when G is poor, the optimization of G is problematic. Rather than training G to minimize $\log(1-D(G(z)))$, we can train G to maximize $\log D(G(z))$.

[3 pts] Explain what problem will be caused by a weak G and a strong D at training, and why maximizing $\log D(G(z))$ instead would be a better strategy.

Solution

2.2 Is the GAN Loss Really Good? [11 pts]

From the questions above, we noted that optimizing the GAN objective is essentially optimizing the JSD between the generator distribution p_g and the real data distribution p_d. But is JSD really a good choice?

Consider the following three distance measures between two distributions p and q[1]:

- The *Kullback-Leibler* (KL) divergence

$$KL(p||q) = \int \log\left(\frac{p(\boldsymbol{x})}{q(\boldsymbol{x})}\right)p(\boldsymbol{x})d\boldsymbol{x}.$$

- The JSD

$$JSD(p,q) = \frac{1}{2}KL(p||m) + \frac{1}{2}KL(q||m),$$

  where $m = \frac{1}{2}(p+q)$.

- The Earth-Mover (EM) distance:

$$W(p,q) = \inf_{\gamma\in\Gamma(p,q)} \mathbb{E}_{(x,y)\sim\gamma}\,||x-y||,$$

  where $\Gamma(p,q)$ represents the set of all joint distributions $\gamma(x,y)$ satisfying: $\forall\gamma(x,y)\in\Gamma(p,q)$, $\int_y \gamma(x,y)dy = p(x)$ and $\int_x \gamma(x,y)dx = q(y)$.

Now, consider the following two 2-dimensional distributions P and Q:

- $\forall(x,y)\sim P, x=0, y\sim \text{Uniform}(0,1)$.
- $\forall(x,y)\sim Q, x=\theta(0\le\theta\le 1), y\sim\text{Uniform}(0,1)$.

[5 pts] Calculate $KL(P||Q)$, $KL(Q||P)$, $JSD(P,Q)$ and $W(P,Q)$.

Solution

[4 pts] Construct another two distributions P,Q where $JSD(P,Q)$ is not differentiable with respect to the parameters of P or Q.

Solution

[2 pts] Use the above examples to discuss the potential drawbacks of using JSD in the GAN framework.

[1]Assuming both p and q are properly defined probability measures on a compact metric set and are absolutely continuous.

Solution

In fact, the EM distance has been introduced as a (perhaps) better alternative of the JSD distance in the orignial GAN objective.

2.3 The Third Player [10 pts]

A key disadvantage of GAN compared to VAE is that *it lacks the ability of inference*. In this section, we will take a probabilistic review of GANs, from which, we derive an enhanced framework so that it is able to infer the latent code of an observed $\boldsymbol{x}$.

Recall the generative process in a GAN: $\boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z}), x \sim p_g(\boldsymbol{x}|\boldsymbol{z})$, where $\boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z})$ is a prior and $p_g(\boldsymbol{x}|\boldsymbol{z})$ is the conditional distribution defined by G. Similar to VAE, we now introduce an inference network Q, which takes $\boldsymbol{x}$ as input, and infers its representation in the latent space. Q therefore parametrizes a distribution $p_q(\boldsymbol{z}|\boldsymbol{x})$ that approximates the posterior. Q, together with G, define two joint distributions from two different directions:

- $p_r(\boldsymbol{x}, \boldsymbol{z}) = p_d(\boldsymbol{x}) p_q(\boldsymbol{z}|\boldsymbol{x})$
- $p_f(\boldsymbol{x}, \boldsymbol{z}) = p_{\boldsymbol{z}}(\boldsymbol{z}) p_g(\boldsymbol{x}|\boldsymbol{z})$

In the original GAN, the discriminator D is designed to distinguish real samples $\boldsymbol{x} \sim p_d(\boldsymbol{x})$ from fake samples $\boldsymbol{x} \sim p_g(x|z)$, and G and D are trained until an equilibrium of the minimax game is achieved. With the third player Q introduced, we now enhance our D to distinguish pairs $(\boldsymbol{x}, \boldsymbol{z})$ from the distributions p_f and p_r, and we hope p_f and p_r will match each other at the equilibrium of the following minimax game adapted from Eq. 12:

$$\min_{G,Q} \max_{D} V(D, G, Q) = \mathbb{E}_{(\boldsymbol{x},\boldsymbol{z}) \sim p_r(\boldsymbol{x},\boldsymbol{z})}[\log D(\boldsymbol{x}, Q(\boldsymbol{x}))] + \mathbb{E}_{(\boldsymbol{x},\boldsymbol{z}) \sim p_f(\boldsymbol{x},\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z}), \boldsymbol{z}))]. \tag{14}$$

Intuitively, we can think the pairs $(\boldsymbol{x}, Q(\boldsymbol{x}))$ as real and $(G(\boldsymbol{z}), \boldsymbol{z})$ as fake, and a three player game is defined – the discriminator D is trained to distinguish fake pairs from real pairs while the generator G and the inferencer Q are jointly trained to fool D.

[**2 pts**] Given any G and Q, find the optimal discriminator of the minimax game in Eq. 14. Further, find the global optimum of the problem in Eq. 14 and the condition when it will be achieved.

Solution

[**6 pts**] Show that at the equilibrium of Eq. 14 where G and Q are optimal and deterministic, $G = Q^{-1}$ and $Q = G^{-1}$ almost everywhere.

Solution

[**2 pts**] Does the optimization of Eq. 14 have the similar problem as described in the last question in section 2.1. If yes, design a better strategy to address the problem. If not, explain why.

Solution

With the third player introduced, we showed that GAN is amendable to modeling joint distributions and supporting inference as VAE does, while still preserving nice theoretical properties.

2.4 Wrap-up: VAEs vs. GANs [5 pts]

Finally, let us wrap up this homework by looking at a few properties of VAEs and GANs. Below are a few statements about VAEs and GANs. Judge each one as true or false, and explain your answers in detail.

[1 pt] Both VAEs and GANs are trained by maximizing the likelihood of training data.

Solution

[1 pt] Both VAEs and GANs are trained by minimizing the divergence between the data distribution and the generator distribution.

Solution

[1 pt] Both VAEs and GANs define an explicit density function over the training data.

Solution

[1 pt] In image generation, VAEs tend to generate images that are sharper and of higher fidelity than GANs.

Solution

[1 pt] Both VAEs and GANs can be directly applied on modeling discrete data, such as text generation.

Solution

References

[1] Peter Dayan. Helmholtz machines and wake-sleep learning. *Handbook of Brain Theory and Neural Network. MIT Press, Cambridge, MA*, 44(0), 2000.

[2] Carl Doersch. Tutorial on variational autoencoders. *arXiv preprint arXiv:1606.05908*, 2016.

[3] Geoffrey E Hinton, Peter Dayan, Brendan J Frey, and Radford M Neal. The “wake-sleep” algorithm for unsupervised neural networks. *Science*, 268(5214):1158–1161, 1995.

[4] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.