EECS 545 Machine Learning - Homework 6

Due date: 11:55pm, Tuesday 4/19/2022

Reminder: While you are encouraged to discuss problems in a small group (up to 5 people), you should write your own solutions and source codes independently. In case you worked in a group for the homework, please include the names of your collaborators in the homework submission. This time you will work with the **py** files (not **ipynb** files) we provided for programming questions. Your answers should be as concise and clear as possible. Please address all questions to `http://piazza.com/class#winter2022/eecs545` with a reference to the specific question in the subject line (E.g. RE: Homework 6, Q2(a)).

Submission Instruction: You should submit both **solution** and **source code**. We may inspect your source code submission visually and run the code to check the results. Please be aware your points can be deducted if you don't follow the instructions listed below:

- Submit **solution** to **Gradescope**
  - Your solution should contain your answers to **all** questions (typed or hand-written). **Please match pages to your answers on Gradescope**, as we reserve the right to not grade submissions without matched pages.
  - Your solution should be self-contained and self-explanatory, regardless of the source code submission.
  - Hand-writings that are very difficult to read may lead to some penalties. If you prefer hand-writing, please use scanners or mobile scanning apps that provide some correction for shading and projective transformations for better legibility; you should not just take photos and submit them.
- Submit **source code** to **Gradescope**
  - Source code should contain your codes to programming questions.
  - To receive full credit your code must run and terminate without error in the autograder. Your code must output all important information to stdout.
  - Your submission to gradescope should contain all your code (either ZIP file or individual upload), as listed below. Do not use a different name for the code or the autograder may not find your submission.
    - `q1_cvae.py`
    - `q2_gan.py`
    - `q3_dqn.py`

Please **do not include** the data folder (the MNIST data files which are automatically downloaded when the program runs) in the source code submission as they are large.

Source Code Instruction: Your source code should run under an environment with Python 3.6+. Please install the packages in our starter code so that they could be correctly imported.

Please do not use any other library unless otherwise instructed or change the directory structure. You should be able to load the data and execute your code on someone else's computer with the environment described above. Note, the outputs of your source code must match with your solution.

1 [35 points] Conditional Variational Autoencoders

In this problem, you will implement a conditional variational autoencoder (CVAE) from [1] and train it on the MNIST dataset.

(a) **[10 points]** Derive the variational lower bound of a conditional variational autoencoder. Show that:

$$\begin{aligned}\log p_\theta(\mathbf{x}|\mathbf{y}) &\geq \mathcal{L}(\theta,\phi;\mathbf{x},\mathbf{y}) \\ &= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x},\mathbf{y})}\left[\log p_\theta(\mathbf{x}|\mathbf{z},\mathbf{y})\right] - D_{KL}\left(q_\phi(\mathbf{z}|\mathbf{x},\mathbf{y})\,\|p_\theta(\mathbf{z}|\mathbf{y})\right), \qquad (1)\end{aligned}$$

where $\mathbf{x}$ is a binary vector of dimension d, $\mathbf{y}$ is a one-hot vector of dimension c defining a class, $\mathbf{z}$ is a vector of dimension m sampled from the posterior distribution $q_\phi(\mathbf{z}|\mathbf{x},\mathbf{y})$. The posterior distribution is modeled by a neural network of parameters ϕ. The generative distribution $p_\theta(\mathbf{x}|\mathbf{y})$ is modeled by another neural network of parameters θ. Similar to the VAE that we learned in the class, we assume the conditional independence on the componenets of $\mathbf{z}$: i.e., $q_\phi(\mathbf{z}|\mathbf{x},\mathbf{y}) = \prod_{j=1}^m q_\phi(z_j|\mathbf{x},\mathbf{y})$, and $p_\theta(\mathbf{z}|\mathbf{y}) = \prod_{j=1}^m p_\theta(z_j|\mathbf{y})$.

(b) **[10 points]** Derive the analytical solution to the KL-divergence between two Gaussian distributions $D_{KL}\left(q_\phi(\mathbf{z}|\mathbf{x},\mathbf{y})\,\|p_\theta(\mathbf{z}|\mathbf{y})\right)$. Let us assume that $p_\theta(\mathbf{z}|\mathbf{y}) \sim \mathcal{N}(\mathbf{0},\mathbf{I})$ and show that:

$$D_{KL}\left(q_\phi(\mathbf{z}|\mathbf{x},\mathbf{y})\,\|p_\theta(\mathbf{z}|\mathbf{y})\right) = -\frac{1}{2}\sum_{j=1}^{m}\left(1+\log\left(\sigma_j^2\right)-\mu_j^2-\sigma_j^2\right), \qquad (2)$$

where μ_j and σ_j are the outputs of the neural network that estimates the parameters of the posterior distribution $q_\phi(\mathbf{z}|\mathbf{x},\mathbf{y})$.

You can assume without proof that

$$D_{KL}\left(q_\phi(\mathbf{z}|\mathbf{x},\mathbf{y})\,\|p_\theta(\mathbf{z}|\mathbf{y})\right) = \sum_{j=1}^{m} D_{KL}\left(q_\phi(z_j|\mathbf{x},\mathbf{y})\,\|p_\theta(z_j|\mathbf{y})\right).$$

This is a consequence of conditional independence of the components of $\mathbf{z}$.

(c) **[15 points]** Fill in code for CVAE network as a `nn.Module` class called `CVAE` in the starter code `q1_cvae.py`:

- Implement the `recognition_model` function $q_\phi(\mathbf{z}|\mathbf{x},\mathbf{y})$.
- Implement the `generative_model` function $p_\theta(\mathbf{x}|\mathbf{z},\mathbf{y})$.
- Implement the `forward` function by inferring the Gaussian parameters using the recognition model, sampling a latent variable using the reparametrization trick and generating the data using the generative model.
- Implement the variational lowerbound `loss_function` $\mathcal{L}(\theta,\phi;\mathbf{x},\mathbf{y})$.
- Train the CVAE and visualize the generated image for each class (i.e., 10 images).
- Repeat the image generation three times with different random noise. Submit 3 x 10 array of subplots showing all the generated images, where the images in the same row are generated from the same random noise, and images in the same column are generated from the the same class label.

If trained successfully, you should be able to sample images $\mathbf{x}$ that reflect the given label $\mathbf{y}$ given the noise vector $\mathbf{z}$.

2 [30 points] Generative Adversarial Networks

This problem asks you to implement generative adversarial networks and train it on MNIST dataset. Specifically, you will implement the Deep Convolutional Generative Adversarial Networks (DCGAN) [2]. In the generative adversarial networks formulation, we have a generator network G that takes in random vector $\mathbf{z}$ and a discriminator network D that takes in an input image $\mathbf{x}$. The parameters of G and D are optimized via the adversarial objective:

$$\min_G \max_D \quad \mathbb{E}_{\mathbf{x}\sim p_{data}}\Big[\log D(\mathbf{x})\Big] + \mathbb{E}_{\mathbf{z}\sim p(\mathbf{z})}\Big[\log(1 - D(G(\mathbf{z})))\Big], \tag{3}$$

In practice, we alternate between training D and G where we first train G to maximize:

$$\mathbb{E}_{\mathbf{z}\sim p(\mathbf{z})}\Big[\log D(G(\mathbf{z}))\Big], \tag{4}$$

followed by training D to maximize:

$$\mathbb{E}_{\mathbf{x}\sim p_{data}}\Big[\log D(\mathbf{x})\Big] + \mathbb{E}_{\mathbf{z}\sim p(\mathbf{z})}\Big[\log(1 - D(G(\mathbf{z})))\Big], \tag{5}$$

Therefore, the two separate optimizations make up one full training step.

1. [**20 points**] Fill in code for the DCGAN network in the `q2_gan.py`. Descriptions of what should be filled in is written as comments in the code itself.
   - Implement the `sample_noise` function.
   - Implement the `build_discriminator` function.
   - Implement the `build_generator` function.
   - Implement the `get_optimizer` function.
   - Implement the `bce_loss` function.
   - Using `bce_loss`, implement the `discriminator_loss` function.
   - Using `bce_loss`, implement the `generator_loss` function.
2. [**10 points**] Using the default hyper-parameters in the starter code (`NUM_TRAIN, NUM_VAL, NOISE_DIM, batch_size`), train the DCGAN network and report 6 plots of sample images at the (0, 250, 500, 1000, 2000, 3000)th iterations. Note, each plot should include 16 (4x4) sample images of digits.

If trained successfully, you should see improvements in the sample qualities as the training progresses.

References

[1] Kihyuk Sohn, Xinchen Yan, and Honglak Lee. Learning structured output representation using deep conditional generative models. In *NeurIPS*. 2015.

[2] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

3 [25 points] Deep Q-Network (DQN)

In this problem, you are asked to implement DQN algorithm[1] and train an agent to play OpenAI Gym CartPole task. We first encourage you to familiarize yourself with OpenAI Gym (`http://gym.openai.com/`) and the cartpole environment (`https://github.com/openai/gym/blob/master/gym/envs/classic_control/cartpole.py`) as we will be using them in this homework.

1. [**15 points**] Implement DQN algorithm in `q3_dqn.py`. Specifically,
   - Fill in the blank(s) in function `select_action` to implement ϵ-greedy action selection method.
   - Fill in the blank(s) in function `optimize` to train the deep Q network.
   - Fill in the blank(s) in function `main`.
   - Fill in the blank(s) in class `DQN`.
2. [**10 points**] Run the script to train the agent playing CartPole task and report the learning curve of the episode duration using the provided function `plot_durations` for the first 1000 episodes. While we provide a suggestion for an optional architecture DQN design (fc-relu-fc), you can change network architecture and hyper-parameters to make the agent perform better. It is expected to over achieve a reward greater than 300 for the 100 episodes-running average reward within 1000 episodes.

[1]You can find the detailed algorithm in the Nature paper: Mnih et al., "Human-level control through deep reinforcement learning", 2015.

4 [10 points] Policy Gradients

For this section, **we will grade the best scoring question.** If you do only one of the questions, we will grade only that question.

Recall the REINFORCE algorithm with objective

$$J(\theta) = \mathbb{E}_{\tau \sim p(\tau;\theta)}[r(\tau)]$$

Remember that the gradient of the objective function above can be approximated as follows:

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log\pi_\theta(a_t^i | s_t^i) r(\tau^i)$$

where the outer sum runs over different agent episodes $\tau^1, ..., \tau^N$ and $\tau^i = ((a_1^i, s_1^i), ..., (a_T^i, s_T^i))$ represents a single episode.

1. **[10 points]** Let $r(\tau) = \sum_{i=1}^{T} r_t$. In this case, the above gradient estimator becomes

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log\pi_\theta(a_t^i | s_t^i) \sum_{t'=1}^{T} r_{t'}^i$$

Show that the following is an unbiased estimate of the $\nabla_\theta J(\theta)$ above:

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log\pi_\theta(a_t^i | s_t^i) \sum_{t'=t}^{T} r_{t'}^i$$

That is, we can omit the rewards collected in the past while keeping the estimator unbiased. The new estimator has the advantage of having lower variance than the original estimator.

2. **[10 points]** Show that adding a state dependent baseline $b(s)$ does not introduce any bias in the estimator. i.e., Show that the following is an unbiased estimator of the gradient. Adding a baseline can further reduce the variance of the estimator.

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log\pi_\theta(a_t^i | s_t^i) \left(\left[\sum_{t'=t}^{T} r_{t'}^i \right] - b(s_t^i) \right)$$