Practical Introduction to Neural Networks
Homework 4
Due: Wednesday June 12, 2019

Late Submissions will not be accepted

Overview

Before starting with this homework please make sure that you have TensorFlow and Jupyter installed on your laptop. In addition, you will need a **GPU** for training CNNs. If you don't have available GPU, you could use the Compute Engine of Google Cloud Platform. Please check the setup guide on canvas. For each problem in this homework we have provided a Jupyter notebook template with which you can get started.

Templates are available at the course Github Repository:
`https://github.com/shlizee/PracticalIntroductionNN`

Please submit Jupyter notebooks with your code (and outputs) by the due date to your Github repository and place a link to your repository in the canvas assignment. For problem 1 and 2, submit a PDF file in the canvas assignment.

Problem 1: Loss function for Variational-Autoencoder (VAE)

We discussed in class that formulating a loss function for VAE directly, i.e. finding the data likelihood given prior $p_\theta(z)$ and conditional distribution $p_\theta(x|z)$ (represented by a decoder network), leads to intractable integral

$$p_\theta(x) = \int p_\theta(z)\ p_\theta(x|z)\ dz$$

Thereby the authors of [1] suggest to estimate the (log) likelihood of the generated data through additional network (an encoder network $q_\phi(z|x)$)

$$\log p_\theta(x) = \mathbb{E}_{z\sim q_\phi(z|x)}[\log p_\theta(x)].$$

and derive a loss function $\mathcal{L}(x, \theta, \phi)$.

Write step-by-step derivation of the loss $\mathcal{L}(x, \theta, \phi)$ for VAE, explain each step and assumptions in them, the final terms which compose the loss function and explain why not all the terms could be estimated.

Problem 2: Optimal $D(\mathbf{x})$ for LSGAN

We showed in class that the the minimax objective for training GANs can be formulated as follows:

$$\min_G \max_D V_{\text{GAN}}(D, G) = \mathbb{E}_{\mathbf{x}\sim\mathbf{p}_{\text{data}}(\mathbf{x})}\left[\log D(\mathbf{x})\right] + \mathbb{E}_{\mathbf{z}\sim\mathbf{p}_{\mathbf{z}}(\mathbf{z})}\left[\log\left(1 - D(G(\mathbf{z}))\right)\right].$$

From it we can derive the optimal discriminator D as

$$D^*_G(\mathbf{x}) = \frac{\mathbf{p_{data}}(\mathbf{x})}{\mathbf{p_{data}}(\mathbf{x}) + \mathbf{p_{model}}(\mathbf{x})},$$

where $\mathbf{p_{model}}(\mathbf{x})$ denotes the distribution of samples from the generator. Regular GANs view D as as a binary classifier and thus adopt the sigmoid loss function which saturates quickly. Since D gradients computed for the log loss are being used to train G, the combination of sigmoid and log could lead to vanishing gradients for updates of G when the samples are far from the real data but end up on the correct side of the decision boundary.

To remedy this problem, the Least Squares GAN (LSGAN) proposes to use L_2 loss instead. LSGAN also uses the *a*-*b*-*c* coding scheme, where **a** is the discriminator value for **fake** data and **b** is the discriminator value for **real** data, and **c** is the value that the generator would like the discriminator to assign to generated samples. Then the objective functions for LSGAN can be defined as follows:

$$\begin{aligned}
\min_D V_{\text{LSGAN}}(D) &= \frac{1}{2}\mathbb{E}_{\mathbf{x}\sim\mathbf{p}_{\text{data}}(\mathbf{x})}\left[(D(\mathbf{x}) - \mathbf{b})^{\mathbf{2}}\right] + \frac{1}{2}\mathbb{E}_{\mathbf{z}\sim\mathbf{p_z(z)}}\left[(D(G(\mathbf{z})) - \mathbf{a})^{\mathbf{2}}\right] \\
\min_G V_{\text{LSGAN}}(G) &= \frac{1}{2}\mathbb{E}_{\mathbf{z}\sim\mathbf{p_z(z)}}\left[(D(G(\mathbf{z})) - \mathbf{c})^{\mathbf{2}}\right].
\end{aligned}$$

The authors show that the values of a, b, c can be assigned to any values solving $b - c = 1, b - a = 2$ equations, in particular $(a = -1, b = 1, c = 0)$. You can find more details in the paper [2].

Given these new objective functions, derive the optimal discriminator D for a fixed G. Discuss the difference between the LSGAN and GAN D optimal values.

Problem 3: Implement GANs in TensorFlow

In this problem, you will implement three types of GAN (Vanilla GAN, LSGAN, DCGAN) models which generate novel images that resemble a set of training images. Since GANs are notoriously finicky with hyperparameters, and also require many training epochs, we will be working with a simple dataset: MNIST. For details, check **HW4_template.ipynb**.

Vanilla GAN

You will implement functions of discriminator, generator, and loss function of GAN. Then will use the Adam Optimizer to train your model.

Least-Square GAN

As discussed in Problem 2, LSGAN is a more stable alternative to the vanilla GAN. You will implement the new loss function for LSGAN and retrain a new model.

Deep Convolutional GAN

In this example, you are asked to implement DCGAN which uses a convolutional network for both the discriminator and the generator. Please read carefully the guidelines described in the template to set up the architecture and then train the model.

References

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

[2] Xudong Mao, Qing Li, Haoran Xie, Raymond YK Lau, Zhen Wang, and Stephen Paul Smolley. Least squares generative adversarial networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2794–2802, 2017.