THE HONG KONG UNIVERSITY OF SCIENCE & TECHNOLOGY
Machine Learning
Homework 3 Solutions

Due Date: See course website

Submission is to be made via Canvas by 11:00pm on the due date.

Question 1: The input of a convolutional layer has shape $27 \times 27 \times 256$ (width, height, depth). The layer uses 384 3×3 filters applied at stride 1 with no zero padding. What is the shape of the output of the layer? How many parameters are there? How many float multiplication operations it will take to compute the net inputs of the all the output units?

Solution: The shape of the output layer is $W_2 \times H_2 \times D_2$ where

$$\begin{aligned} W_2 &= (W_1 - F + 2P)/S + 1 = (27 - 3 + 0)/1 + 1 = 25 \\ H_2 &= (H_1 - F + 2P)/S + 1 = (27 - 3 + 0)/1 + 1 = 25 \\ D_2 &= 384. \end{aligned}$$

The number of parameters is

$$(FFD_1 + 1)K = (3 \times 3 \times 256 + 1)384 = 885,120.$$

The number of float multiplication ops it takes to compute the net input of each output unit is FFD_1. The number of output units is $W_2H_2D_2$. Hence, the total number of ops is:

$$(3 \times 3 \times 256) \times (25 \times 25 \times 384) = 552,960,000$$

Question 2: What is batch normalization? What is layer normalization? What are their pros and cons?

Solution: Batch normalization is a technique to stabilize the learning process of deep neural network. To be specific, each batch of data could possess different distribution which causes internal covariate shift. Batch normalization normalizes each batch of data in order to avoid the problem by using scaling factor and shifting factor, and in turn, results in stabilized learning process as well as faster convergence. However, there are some cons in this technique. First is the dependency on the size of batch. Since batch normalization is manipulating data of each batch, the data in batch decides the value of scaling and shifting factor. Therefore, careful choice of the batch size is crucial, and of course, when batch size is 1, batch normalization cannot be applied. In addition, at training phase, the scaling and shifting factor have to be identified and kept, since the values will be used in testing phase as well.

Batch normalization is not suitable for RNN because it destroys sequential dependencies, which are important for NLP. It also requires a lot of memory as statistics for each time step have to be computed and stored. Furthermore, if given a longer sentence in testing than in training data, it is not possible to apply batch normalization considering the absent of statistics.

Layer normalization is related to batch normalization, but different in that it normalizes the inputs across features. Since, layer normalization normalizes inputs across features, any number of batch size could be used. In addition, past experiments show promising results on Recurrent Neural Network models with layer normalization.

Question 3 In an LSMT cell, $\mathbf{h}^{(t)}$ is computed from $\mathbf{h}^{(t-1)}$ and $\mathbf{x}^{(t)}$ using the following formulae:

$$\begin{aligned} \mathbf{f}_t &= \sigma(\mathbf{W}_f\mathbf{x}^{(t)} + \mathbf{U}_f\mathbf{h}^{(t-1)} + \mathbf{b}_f) \\ \mathbf{i}_t &= \sigma(\mathbf{W}_i\mathbf{x}^{(t)} + \mathbf{U}_i\mathbf{h}^{(t-1)} + \mathbf{b}_i) \\ \mathbf{o}_t &= \sigma(\mathbf{W}_q\mathbf{x}^{(t)} + \mathbf{U}_q\mathbf{h}^{(t-1)} + \mathbf{b}_q) \\ \mathbf{c}_t &= \mathbf{f}_t \otimes \mathbf{c}_{t-1} + \mathbf{i}_t \otimes \tanh(\mathbf{U}\mathbf{x}^{(t)} + \mathbf{W}\mathbf{h}^{(t-1)} + \mathbf{b}) \\ \mathbf{h}^{(t)} &= \mathbf{o}_t \otimes \tanh(\mathbf{c}_t) \end{aligned}$$

(a) Intuitively, what are the functions of the forget gate $\mathbf{f}_t$ and the input gate $\mathbf{i}_t$ do? Answer briefly.

(b) Why do we use the sigmoid function for $\mathbf{f}_t$ and $\mathbf{i}_t$, but tanh for the memory cell $\mathbf{c}_t$ and the output $\mathbf{h}^{(t)}$? Answer briefly.

Solution: (a) The forget gate $\mathbf{f}_t$ determines which components of the previous state $\mathbf{c}_{t-1}$ and how much of them to remember/forget. The input gate $\mathbf{i}_t$ determines which components of the input from $\mathbf{h}^{(t-1)}$ and $\mathbf{x}^{(t)}$ and how much of them should go into the current state.

(b) The sigmoid function is used for $\mathbf{f}_t$ and $\mathbf{i}_t$ so that their values are likely to be close to 0 or 1, and hence mimicking the close and open of gates. The tanh function is used for $\mathbf{c}_t$ and $\mathbf{h}^{(t)}$ so that strong gradient signals can be backpropagated from $\mathbf{h}^{(t)}$ to $\mathbf{c}_t$, and then to $\mathbf{h}^{(t-1)}$.

Question 4 BERT input representation is the sum of token embedding, segment embedding, and positional embedding. Briefly explain why positional embedding is introduced in the architecture.

Solution: In BERT, unlike AutoRegressive approach, input tokens are fed into the architecture at once. Without positional embedding, the input representation of a token holds no information about its position in a sentence, in other words, each token has only one input representation (assuming same segment embedding). By adding positional embedding to input representation, each token can have different representations according to its position, and holds positional information.

Question 5 (optional): Let $p(\mathbf{x}, \mathbf{z})$ and $q(\mathbf{z})$ be two probability distributions. Show that

$$\log p(\mathbf{x}) \geq E_{\mathbf{z}\sim q(\mathbf{z})} \log p(\mathbf{x}|\mathbf{z}) - \mathcal{D}_{KL}(q(\mathbf{z})||p(\mathbf{z}))$$

where $\mathcal{D}_{KL}(q(\mathbf{z})|p(\mathbf{z})) = E_{\mathbf{z}\sim q(\mathbf{z})} \log q(\mathbf{z}) - E_{\mathbf{z}\sim q(\mathbf{z})} \log p(\mathbf{z})$.

Note that the RHS of the inequality is known as the **variational lower bound** of $\log p(\mathbf{x})$, or the **evidence lower bound (ELBO)**. Another way to write the inequality is as follows:

$$\log p(\mathbf{x}) \geq E_{\mathbf{z}\sim q(\mathbf{z})} \log p(\mathbf{x}, \mathbf{z}) - E_{\mathbf{z}\sim q(\mathbf{z})} \log q(\mathbf{z}) = E_{\mathbf{z}\sim q(\mathbf{z})} \log p(\mathbf{x}, \mathbf{z}) + H(q).$$

Reflect on how the variational lower bound is used variational autoencoder. You can do this by pointing out what are the two distributions.

Solution: Consider $\mathcal{D}_{KL}(q(\mathbf{z})|p(\mathbf{z}|\mathbf{x}))$:

$$\begin{aligned}
\mathcal{D}_{KL}(q(\mathbf{z})|p(\mathbf{z}|\mathbf{x})) &= E_{\mathbf{z}\sim q(\mathbf{z})} \log q(\mathbf{z}) - E_{\mathbf{z}\sim q(\mathbf{z})} \log p(\mathbf{z}|\mathbf{x}) \\
&= E_{\mathbf{z}\sim q(\mathbf{z})} \log q(\mathbf{z}) - E_{\mathbf{z}\sim q(\mathbf{z})} \log \frac{p(\mathbf{x}|\mathbf{z})p(\mathbf{z})}{p(\mathbf{x})} \\
&= E_{\mathbf{z}\sim q(\mathbf{z})} \log p(\mathbf{x}) - E_{\mathbf{z}\sim q(\mathbf{z})} \log p(\mathbf{x}|\mathbf{z}) + E_{\mathbf{z}\sim q(\mathbf{z})} \log q(\mathbf{z}) - E_{\mathbf{z}\sim q(\mathbf{z})} \log p(\mathbf{z}) \\
&= E_{\mathbf{z}\sim q(\mathbf{z})} \log p(\mathbf{x}) - E_{\mathbf{z}\sim q(\mathbf{z})} \log p(\mathbf{x}|\mathbf{z}) + \mathcal{D}_{KL}(q(\mathbf{z})|p(\mathbf{z})).
\end{aligned}$$

The equation follows from the fact that $\mathcal{D}_{KL}(q(\mathbf{z})|p(\mathbf{z}|\mathbf{x})) \geq 0$.

In VAE, the target distribution is $p_\theta(\mathbf{x}^{(i)}|\mathbf{z})$ and the variational distribution is $q(\mathbf{z}|\mathbf{x}^{(i)})$. The first version of the inequality is used.

Question 6: Suppose two random variables x and z are related by the following equation:

$$z = x^2 + 2x + \frac{x^2}{2}\epsilon,$$

where ϵ is a random variable that follows the normal distribution $\mathcal{N}(0, 1)$, i.e.,

$$p(\epsilon) = \frac{1}{\sqrt{2\pi}} e^{\frac{-\epsilon^2}{2}}.$$

What is the density function of the conditional distribution $p(z|x)$?

(Note that the purpose of this problem is to help you understand reparameterization.)

Solution: Since ϵ is Gaussian, z is also Gaussian. Its means is $\mu_z = x^2 + 2x$, and its variance is $\sigma_z^2 = x^4/4$. Hence, its density function is:

$$p(z|x) = \frac{1}{\sqrt{2\pi}x^2/2} e^{-\frac{(z-(x^2+2x))^2}{x^4/2}}$$

Question 7: What are the main differences between variational autoencoder (VAE) and generative adversarial network (GAN) in terms their functionalities and the ways they operate?

Solution: Both VAE and GAN take a collection $\{\mathbf{x}^{(i)}\}$ of unlabeled data as input and assume the data are generated from a random latent vector $\mathbf{z}$. VAE learns a conditional distribution $p(\mathbf{x}|\mathbf{z})$ while GAN learns a deterministic function $\mathbf{x} = g(\mathbf{z})$.

Both VAE and GAN represent the relationship between $\mathbf{x}$ and $\mathbf{z}$ using a deep neural network, which is called the generative model (aka decode in VAE). VAE learns the parameters of the generative network by minimizing the KL divergence between the real data distribution and the model distribution. The objective in intractable. So, an encoder network is introduced to provide a lower bound. The parameters of the encoder and the decoder are trained simultaneously.

GAN learns the parameters of the generative network by minimizing the Jensen-Shannon divergence between the real data distribution and the model distribution. The objective in intractable. So, a discriminator network is introduced to provide an approximation. The parameters of the generator and the discriminator are trained alternately.

Question 8: What are the objective functions for the discriminator and the generator in GAN? What are the objective functions for the critic and generator in WGAN?

Solution: The discriminator in GAN maximizing:

$$V(G, D) = E_{\mathbf{x}\sim p_r(\mathbf{x})}[\log D(\mathbf{x})] + E_{\mathbf{x}\sim p_g(\mathbf{x})}[\log(1 - D(\mathbf{x}))],$$

p_r is the real data distribution, p_g is the generator distribution, and $D(\mathbf{x})$ is the probability that $\mathbf{x}$ is a real example.

Theoretically, the generator of GAN should minimize $E_{\mathbf{x}\sim p_g(\mathbf{x})}[\log(1 - D(\mathbf{x}))]$. However, this leads to unstable training. In practice, it minimizes the following function instead:

$$-E_{\mathbf{x}\sim p_g(\mathbf{x})}[\log D(\mathbf{x})].$$

The critic of WGAN maximizes the following function:

$$L_c = E_{\mathbf{x}\sim p_r}[f(\mathbf{x})] - E_{\mathbf{x}\sim p_g}[f(\mathbf{x})],$$

where $f(\mathbf{x})$ is the critic function represented by a neural network and the weights of the network are restricted to $[-c, c]$. The generator of WGAN minimizes:

$$-E_{\mathbf{x}\sim p_g(\mathbf{x})}[\log f^*(\mathbf{x})],$$

f^* is the critic function learning by the critic.