Exercises
Deep Learning
Fall 2018

Machine Learning Institute
Dept. of Computer Science, ETH Zürich
Fernando Pérez-Cruz
Web http://www.da.inf.ethz.ch/teaching/2018/DeepLearning/

Series Monday, Dec 17, 2018
(Deep Learning, Exercise series 11)

Problem 1 (Generative Adversarial Networks):

Read the original GAN publication (you can click on the embedded hyperlinks in the pdf) and answer the following questions:

- Goodfellow et al. "Generative Adversarial Networks." arXiv:1406.2661 (2014).

(i) Describe the mode collapse problem of GANs (sometimes referred to as "the Helvetica scenario"). (ii) GANs are typically trained by taking a single (or several) update steps for the discriminator in the inner loop and a single (or several) update steps for the generator in the outer loop. Explain what would happen if we train the discriminator in the inner loop to optimality. (iii) GANs are mostly successful for generating images. What is the challenge of applying GANs to discrete data (e.g. text)? (iv) Name one advantage and one disadvantage of using GANs vs. VAEs.

Problem 2 (Generalizing GANs to any f-divergences):

In the original GAN paper the authors show that training GANs is related to approximate minimization of the symmetric Jensen-Shannon divergence. In this exercise, we will show that the principle of GANs is more general and we can extend it to arbitrary f-divergences.

Statistical divergences measure the difference between two given probability distributions, P and Q. A large class of different divergences, called f-divergences, are defined as

$$D_f(P||Q) = \int_x q(x) f(\frac{p(x)}{q(x)}) dx, \tag{1}$$

where P and Q possess an absolute continuous density function p and q with respect to a base measure dx defined on the domain X. The function $f : \mathbb{R}_+ \to \mathbb{R}$ is a convex, lower-semicontinuous function satisfying $f(1) = 0$. Different choices of f recover popular divergences. For example, $f(u) = -(u+1)\log\frac{1+u}{2} + u\log u$ recovers the Jensen-Shannon divergence.

- Derive a variational formulation of f-divergences (which can be estimated with samples from P and Q), i.e. show that $D_f(P||Q) \geq \sup_{T\in \mathrm{T}}(\mathbb{E}_{x\sim P}[T(x)] - \mathbb{E}_{x\sim Q}[f^*(T(x))])$, where T is an arbitrary function $T : X \to \mathbb{R}$ within some family of functions T. More precisely, use the fact that f can be written as the bi-conjugate $f(u) = \sup_{t\in \mathbf{dom}_{f^*}}\{tu - f^*(t)\}$, where $f^*(t) = \sup_{u\in \mathbf{dom}_f}\{tu - f(t)\}$ denotes the convex conjugate (also known as fenchel conjugate or fenchel dual). In other words, $f(u)$ is written as the conjugate of the conjugate.

Now let us see how the variational lower bound on the f-divergence $D_f(P||Q)$ can be used in order to estimate a generative model Q given a true distribution P. We use two neural networks, Q and T. Q is the generative model, which takes as input a random vector and outputs a sample. We parametrize Q through a vector θ and write Q_θ. T is the variational function, which takes as input a sample and returns a scalar. Similarly, we parametrize T using a vector ω and write T_ω.

- Rewrite the variational lower bound as $F(\theta, \omega)$ using the new notation.
- Describe how we can learn a generative model Q_θ by optimizing $F(\theta, \omega)$.

Respecting the domain $\mathbf{dom}f^*$ of the conjugate functions f^*, we can represent T_ω as $T_\omega = g_f(V_\omega(x))$, where $V_\omega : X \to \mathbf{R}$ without any range constraints on the output, and $g_f : \mathbf{R} \to \mathbf{dom}_{f^*}$ is an output activation function specific to the f-divergence used.

Name	Output activation g_f	dom_{f^*}	Conjugate $f^*(t)$	$f'(1)$
Total variation	$\frac{1}{2}\tanh(v)$	$-\frac{1}{2} \leq t \leq \frac{1}{2}$	t	0
Kullback-Leibler (KL)	v	$\mathbb{R}$	$\exp(t-1)$	1
Reverse KL	$-\exp(v)$	$\mathbb{R}_-$	$-1-\log(-t)$	-1
Pearson χ^2	v	$\mathbb{R}$	$\frac{1}{4}t^2+t$	0
Neyman χ^2	$1-\exp(v)$	$t<1$	$2-2\sqrt{1-t}$	0
Squared Hellinger	$1-\exp(v)$	$t<1$	$\frac{t}{1-t}$	0
Jeffrey	v	$\mathbb{R}$	$W(e^{1-t})+\frac{1}{W(e^{1-t})}+t-2$	0
Jensen-Shannon	$\log(2)-\log(1+\exp(-v))$	$t<\log(2)$	$-\log(2-\exp(t))$	0
Jensen-Shannon-weighted	$-\pi\log\pi-\log(1+\exp(-v))$	$t<-\pi\log\pi$	$(1-\pi)\log\frac{1-\pi}{1-\pi e^{t/\pi}}$	0
GAN	$-\log(1+\exp(-v))$	$\mathbb{R}_-$	$-\log(1-\exp(t))$	$-\log(2)$
α-div. ($\alpha<1, \alpha\neq 0$)	$\frac{1}{1-\alpha}-\log(1+\exp(-v))$	$t<\frac{1}{1-\alpha}$	$\frac{1}{\alpha}(t(\alpha-1)+1)^{\frac{\alpha}{\alpha-1}}-\frac{1}{\alpha}$	0
α-div. ($\alpha>1$)	v	$\mathbb{R}$	$\frac{1}{\alpha}(t(\alpha-1)+1)^{\frac{\alpha}{\alpha-1}}-\frac{1}{\alpha}$	0

Figure 1: Recommended final layer activation functions and critical variational function level

- Rewrite $F(\theta,\omega)$ using V_ω and g_f.
- Recover the GAN objective by plugging in the correct output activation and conjugate functions from Figure 1.
- From the figure we can see that GAN is related to the Jensen-Shannon divergence through $D_{GAN} = 2D_{JS} - \log 4$. Discuss why training a generative model by minimizing the Jensen-Shannon divergence might give advantages over the standard maximum-likelihood training.

Problem 3 (Practical: GANs in Tensorflow):

In this exercise you will be implementing a GAN on the MNIST dataset. You will be given a template that you need to fill and adapt (See the jupyter notebook:*mnist-gan-template.ipynb* or the python script:*mnist-gan-template.py*). Our GAN comprises a generator function and a discriminator function. These functions are based on fully connected layers and their parameters are going to be optimized to respectively minimize and maximize the GAN value function using a gradient based algorithm.

More formally, a generator takes an input $z \in \mathbb{R}^d$ and transforms it into a sample $x \in \mathbb{R}^{d'}$ in the form $x = s(Wz+b)$, where s is a non-linear function such as a *tanh*. The variable W is the weight matrix and b is the bias. The latent vector z is typically sampled from a normal distribution i.e. $z \sim \mathbf{N}(0,1)$.

The discriminator is trained as a classifier to distinguish input samples i, which are either generated samples x or real samples x_r coming from our training set that represents p_{data}. The mapping through the discriminator follows a similar transformation: $o = s(W^{'}i + b^{'})$. The model parameters: $W, W^{'}, b, b^{'}$ are optimized by alternatively updating the discriminator and generator through maximizing and minimizing the GAN objective: $V(G,D) = \mathbf{E}_{x_r \sim p_{data}} D(x_r) + \mathbf{E}_z(1 - D(G(z))$. More concretely, the procedure for optimizing the parameters is as follows:

Algorithm 1 GAN algorithm

1: **for** number of training steps do **do**
2: Sample a batch of latent vectors $z \sim \mathbf{N}(0,1)$
3: Sample a batch of real images $x_r \sim p_{data}$
4: Update $W^{'}, b^{'}$ by ascending the stochastic gradient i.e. perform one gradient step of $\max_{W^{'},b^{'}} V(G,D) = \max_{W^{'},b^{'}} \mathbf{E}_{x_r \sim p_{data}} D(x_r) + \mathbf{E}_z(1 - D(G(z))$
5: Update W, b by descending the stochastic gradient i.e. perform one gradient step of $\min_{W,b} V(G,D) = \min_{W,b} \mathbf{E}_{x_r \sim p_{data}} D(x_r) + \mathbf{E}_z(1 - D(G(z))$

Your tasks are:

1. Create a generator and a discriminator of 2 hidden layers. In the template, you will have to fill the functions: *generator* and *discriminator*. The first hidden-layer will have 128 neurons. The dimensionality of the second layer depends on the output size. Hence, the second layer of the generator will be of size 784 and the size of the second discriminator layer is 1. For the first layers use *leaky_relu* as

an activation function. For the second generator/discriminator layers use as activation functions *tanh* and *sigmoid*, respectively.

2. Implement a GAN loss function and train your generative model. You can use the Adam Optimizer. By using the plot function given at the end of the template, display some generated images. (Hint: Use the *tf.train.AdamOptimizer* and set the learning rate to 0.0001 for both the generator and discriminator). Observe the progress of the sample quality throughout training.

3. You might have noticed that the GAN objective is subject to numerical instabilities due to the $\log$ as $\log 0$ is undefined. For this reason, you can either add a small ϵ to the input of every $\log$ or use the numerically stable Tensorflow function *tf.nn.sigmoid_cross_entropy_with_logits*. This function takes as input the logits (i.e. the output of the discriminator before applying the sigmoid activation). Train again your GAN. Display some generated images.

4. In practice, due to getting a stronger gradient signal, it is often preferred to use an alternative update rule for the generator, known as the non-saturating update rule, or simply the log-trick. Change the update of the generator from minimizing $\log\left(1 - D(G(z))\right)$ to maximizing $\log D(G(z))$ and generate samples. Analyze how the samples differ between the former and the latter update rule.

5. Tune the hyperparameters of the model to obtain better sample quality.

Optional: Change the fully connected layers to (de)convolutional layers in order to further strengthen the model.

Optional: Think about extending the previous unconditional framework to be able to do conditional generation based on the class.