lions@epfl

EE-556: MATHEMATICS OF DATA: FROM THEORY TO COMPUTATION
LABORATORY FOR INFORMATION AND INFERENCE SYSTEMS
FALL 2021

EPFL

INSTRUCTOR
PROF. VOLKAN CEVHER

HEAD TA
FABIAN LATORRE &ALI KAVIS

HOMEWORK EXERCISE-2 (FOR LECTURE 7)

This homework covers Lectures 7-10. Please take a look at the material for the context and notation. You have three weeks to complete the homework.

The homework requires you to study minimax problems in the context of Generative Adversarial Networks (GANs). Since the state-of-the-art GAN training is computationally expensive, you will implement a simple GAN with linear generator and a dual variable. Then, you will implement two variants of the stochastic gradient descent, AMSGrad and RMSProp. You will use these algorithms to train a neural network to solve the image classification task. For concrete demonstrations, we will use one of the most popular deep learning frameworks, called "PyTorch". These problems should expose you the fundamentals of GAN training at a basic level.

For the homework, we provide some basic code upon which you need to build functionality. You should only add code in the sections marked with the `TODO` comment, but feel free to modify other parts of the code if you really need to. Please read the `README.md` file for further required python modules.

1 Minimax problems and GANs

1.1 Theoretical recap: stationary points and convergence in minmax games

Consider a stylized function $f : \mathbb{R}^2 \to \mathbb{R}$, such that we have $f(x, y) = (x - 1)(y - 1)$.

1. (10 points) Write down the first-order stationary points of f, and classify them as local minimum, local maximum, or saddle point by inspecting its Hessian.

2. (10 points) Show that $(x^\star, y^\star) = (1, 1)$ is a solution to the minimax problem $\min_x \max_y f(x, y)$. You can quantify the solution by using the following saddle point inequality: $f(x^\star, y^\star) \geq f(x^\star, y)$ and $f(x^\star, y^\star) \leq f(x, y^\star)$, for all x, y.

3. (20 points) One possible attempt at finding this solution via iterative first-order methods is to perform gradient updates on the variables x and y. More precisely for $\gamma > 0$ consider the simultaneous gradient descent/ascent updates

$$x_{k+1} = x_k - \gamma \nabla_x f(x_k, y_k), \qquad y_{k+1} = y_k + \gamma \nabla_y f(x_k, y_k)$$

   Show that the sequence of iterates $\{x_k, y_k\}_{k=0}^{\infty}$ starting from any point $(x_0, y_0) \neq (1, 1)$ diverges, for any $\gamma > 0$. Find the rate at which the distance of the sequence $\{x_k, y_k\}$ to the origin grows as the number of iterations k increases.

1.2 Practice: GAN

Please review Lecture 10 for the notation and the setup in this exercise. As a basic example for the implementation of GANs, we will try to learn the density of a mixture of Gaussian (MoG) distribution from empirical samples using the Wasserstein GAN (WGAN) framework. MoG distribution is multi-modal, and hence, we can visualize multiple "modes" of the distribution.

Both your generator neural network $h_{\mathbf{x}}$ and dual variable neural network $\mathrm{d}_{\mathbf{y}}$ classes are defined as single hidden layer networks:

$$\mathcal{H} := \{h : h_{\mathbf{x}}(\omega) = X_2 \,\mathrm{relu}(X_1 \omega + x_1) + x_2\}, \qquad \mathcal{D} := \{\mathrm{d} : \mathrm{d}_{\mathbf{y}}(\mathbf{a}) = Y_2 \,\mathrm{relu}(Y_1 \mathbf{a} + y_1) + y_2\}, \tag{1}$$

where $\mathbf{x} = [x_1; x_2; X_1; X_2]$ are the "generator" parameters, and $\mathbf{y} = [y_1; y_2; Y_1; Y_2]$ are the "dual" or the "discriminator" parameters. The dimensions of these parameters will be apparent from the context as well as the base code provided along with the homework.

1. (10 points) Fill in the missing code in the file `variables.py` implementing the two neural networks and the spectral normalization (Myato 2018) method that enforces a Lipschitz constraint. For this you have to do the following:

   (a) Create the weights and any auxilliary variables you might need for the Spectral Normalization method (**Hint:** Think carefully which variables need to be updated via autograd and hence need to be implemented as `torch.nn.Modulules` or `torch.nn.Parameters` and which variables you will need to use `.detach()` and manual updates)

(b) Fill in the `enforce_lipschitz` function which performs Spectral Normalization as presented in Myato 2018

Then, implement a stochastic estimate of the objective function of the minimax game (in `trainer.py`):

$$\min_{\mathbf{x}\in\mathcal{X}} \max_{\mathbf{y}\in\mathcal{Y}} \boldsymbol{E}[\mathrm{d}_{\mathbf{y}}(\mathbf{a})] - \boldsymbol{E}[\mathrm{d}_{\mathbf{y}}(h_{\mathbf{x}}(\omega))] = \min_{h\in\mathcal{H}} \max_{\mathrm{d}\in\mathcal{F}} \boldsymbol{E}[\mathrm{d}(\mathbf{a})] - \boldsymbol{E}[\mathrm{d}(h(\omega))], \tag{2}$$

where $\mathbf{a} \sim \hat{\mu}_n$ is distributed with respect to the empirical estimate $\hat{\mu}_n$ of the true distribution $\hat{\mu}^\natural$ (i.e., it is distributed uniformly on a set of samples drawn from the true distribution), and $\omega \sim \mathrm{p}_\Omega$ is the noise distribution that we push-forward with the generator neural network $h_{\mathbf{x}}(\omega) \sim h_{\mathbf{x}}\sharp\mathrm{p}_\Omega$. The stochastic estimate that you should implement simply corresponds to an average over finite samples drawn from the target and noise distribution. Use the methods available for the class `torch.distributions.Distribution` from the PyTorch module.

Remark: The `enforce_lipschitz` does not need to return a value. It should modify the parameters $\mathbf{y}$ of the dual network $\mathrm{d}_{\mathbf{y}}$ in situ. In order to do this we recommend directly modifying any tensor's values by rewriting their data attribute e.g., `y.data = y.data + 1`. This avoids PyTorch's automatic tracking of operations for automatic differentiation, which *might* cause issues.

2. (10 points) Compare and contrast Spectral Normalization with Spectral *Regularization* (mentioned in the paper) and also compare and constrast with other methods of Lipschitz constraints used in WGAN training, namely Weight Clipping and Gradient penalty. Write an explanation of the differences.

3. (25 points) Complete the missing functions in `trainer.py`, You should implement an alternating gradient ascent/descent update, training the generator 1 time for every 5 dual updates. More specifically, you will implement the conceptual algorithm below using the parameters of the neural networks

$$\mathrm{d}^{k+1} = \mathrm{clip}(\mathrm{d}^k + \gamma \mathrm{SG}_{\mathrm{d}}(\mathrm{d}^k, h^k k)), \qquad \text{(always)} \tag{3}$$

$$h^{k+1} = h^k - \gamma \mathrm{SG}_h(\mathrm{d}^{k+1}, h^k) \qquad (\text{if } k \bmod 5 = 0), \tag{4}$$

where SG is the stochastic gradient oracle. To perform the optimization, use 2 separate instances of Adam with $(\beta_0, \beta_1) = (0.0, 0.9)$ instead of the simpler SGD steps used for illustrative purposes above.

After filling in your code, run the script `train.py`. Include the generated plots in your report and comment on your findings.

2 Optimizers of Neural Network

The goal of this exercise is to implement different optimizers for a handwritten digit classifier using the well-known MNIST dataset. This dataset has 60000 training images, and 10000 test images. Each image is of size 28×28 pixels, and shows a digit from 0 to 9.

(a) General guideline: Complete the codes marked with *TODO* in the *optimizers.py*.

(b) The implementation of mini-batch SGD and SGD with HB momentum are provided as examples.

Vanilla Minibatch SGD
Input: learning rate γ
1. initialize θ_0 **2. For** t = 0, 1,... ,N-1: obtain the minibatch gradient $\hat{\mathbf{g}}_t$ update $\theta_{t+1} \leftarrow \theta_t - \gamma\hat{\mathbf{g}}_t$

Minibatch SGD with Momentum
Input: learning rate γ, momentum ρ
1. initialize θ_0, $\mathbf{m_0} \leftarrow \mathbf{0}$ **2. For** t = 0, 1,... ,N-1: obtain the minibatch gradient $\hat{\mathbf{g}}_t$ update $\mathbf{m}_{t+1} \leftarrow \rho\mathbf{m}_t + \hat{\mathbf{g}}_t$ update $\theta_{t+1} \leftarrow \theta_t - \gamma\mathbf{m}_{t+1}$

(c) Implement of following optimizers:

(a) (10 points) Implement RMSProp

RMSProp
Input: global learning rate γ, damping coefficient δ, decaying parameter τ
1. initialize θ_0, $\mathbf{r} \leftarrow \mathbf{0}$ **2. For** t = 0, 1,... ,N-1: obtain the minibatch gradient $\hat{\mathbf{g}}_t$ update $\mathbf{r} \leftarrow \tau\mathbf{r} + (1-\tau)\hat{\mathbf{g}}_t \odot \hat{\mathbf{g}}_t$ update $\theta_{t+1} \leftarrow \theta_t - \frac{\gamma}{\delta + \sqrt{\mathbf{r}}}\hat{\mathbf{g}}_t$

where $\odot$ is the element-wise multiplication between two matrices.

(b) (10 points) Implement AMSGrad method

AMSGrad
Input: global learning rate γ, damping coefficient δ, first order decaying parameter β_1, second order decaying parameter β_2
1. initialize θ_0, $\mathbf{m}_1 \leftarrow \mathbf{0}$, $\mathbf{m}_2 \leftarrow \mathbf{0}\mathbf{m}_2^{max}$ **2. For** t = 0, 1,... ,N-1: obtain the minibatch gradient $\hat{\mathbf{g}}_t$ update $\mathbf{m}_1 \leftarrow \beta_1\mathbf{m}_1 + (1-\beta_1)\hat{\mathbf{g}}_t$ update $\mathbf{m}_2 \leftarrow \beta_2\mathbf{m}_2 + (1-\beta_2)\hat{\mathbf{g}}_t \odot \hat{\mathbf{g}}_t$ correct bias $\hat{\mathbf{m}}_1 \leftarrow \frac{\mathbf{m}_1}{1-\beta_1^{t+1}}$ $\hat{\mathbf{m}}_2 \leftarrow \frac{\mathbf{m}_2}{1-\beta_2^{t+1}}$ update $\mathbf{m}_2^{max} = \max(\mathbf{m}_2^{max}, \hat{\mathbf{m}}_2)$ update $\theta_{t+1} \leftarrow \theta_t - \gamma\frac{\hat{\mathbf{m}}_1}{\delta + \sqrt{\mathbf{m}_2^{\hat{max}}}}$

(d) (20 points) Set the learning rate to 0.5, 10^{-2}, and 10^{-5}. Run the neural network for 15 epochs, repeat it for 3 times and compare the averaged accuracies. Plot the obtained training loss for different optimizers. State how the performance of the adaptive learning-rate methods versus SGD and SGD with momentum methods is compared.

3 Guidelines for the preparation and the submission of the homework

Work on your own. Do not copy or distribute your code to other students in the class. Do not reuse any other code related to this homework. Here are few warnings and suggestions for you to prepare and submit your homework.

- This homework is due at 23:59, 5th December, 2021
- Submit your work before the due date. Late submissions are not allowed and you will get 0 point from this homework if you submit it after the deadline.
- Your final report should include detailed answers and it needs to be submitted in PDF format.
- The PDF file can be a scan or a photo. Make sure that it is legible.
- The results of your simulations should be presented in the final report with clear explanation and comparison evaluation.
- We provide Pytorch scripts that you can use to implement the algorithms, but you can implement them from scratch using any other convenient programming tool (in this case, you should also write the codes to time your algorithm and to evaluate their efficiency by plotting necessary graphs).
- Even if you use the Pytorch scripts that we provide, you are responsible for the entire code you submit. Apart from completing the missing parts in the scripts, you might need to change some written parts and parameters as well, depending on your vision for the implementation.
- The code should be well-documented and should work properly. Make sure that your code runs without errors. If the code you submit does not run, you will not be able to get any credits from the related exercises.
- Compress your code and your final report into a single ZIP file, name it as `ee556_2021_hw2_NameSurname.zip`, and submit it through the moodle page of the course.