lions@epfl

**EE-556: MATHEMATICS OF DATA: FROM THEORY TO COMPUTATION**
**LABORATORY FOR INFORMATION AND INFERENCE SYSTEMS**
**FALL 2019**

EPFL


INSTRUCTOR
PROF. VOLKAN CEVHER

HEAD TA
THOMAS SANCHEZ


# HOMEWORK EXERCISE-2 (FOR LECTURE 7)

This homework covers Lecture 7. First, we will study minimax problems in the context of GANs. You will implement a simple GAN setup with linear generator and dual variable. Secondly, you will implement two versions of stochastic gradient descent, Adam, Adagrad, and RMSProp, and you will use them to train a neural network to solve the image classification task. We will use one of the most popular deep learning frameworks "PyTorch".

We will provide some basic code upon which you will 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 need. Read the `README.md` file for further required python modules.

## 1 Minimax problems and GANs

Consider the function $f : \mathbb{R}^2 \to \mathbb{R}$ where $f(x, y) = xy$

1. (10 points) Find the first order stationary points, and classify them as local minimum, local maximum or saddle point according to the Hessian.

2. (10 points) Show that $(x^\star, y^\star) = (0, 0)$ is a solution to the minimax problem $\min_x \max_y f(x, y)$. This means that $f(x^\star, y^\star) \geq f(x^\star, y)$ 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 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 $(x_0, y_0) \neq (0, 0)$ diverges, for any $\gamma > 0$. Find the rate at which its distance to the origin grows.

In the context of GANs, suppose the true distribution is a multivariate normal on $\mathbb{R}^2$ (with mean and covariance matrix as specified in the code), and the noise distribution is a standard normal on $\mathbb{R}^2$. Your generator and dual variable classes are defined as

$$\mathcal{G} := \{g : g(z) = Wz + b\}, \qquad \mathcal{F} := \{f : f(x) = v^T x\} \tag{1}$$

For a matrix $W \in \mathbb{R}^{2\times 2}$ and vectors $b, v \in \mathbb{R}^2$.

1. (10 points) Suppose the space $\mathbb{R}^2$ is equipped with the $\ell_2$-norm $\|(x, y)\|_2^2 = x^2 + y^2$. For any $f \in \mathcal{F}$ compute its Lipschitz constant with respect to this norm. Describe the set of functions in $\mathcal{F}$ whose Lipschitz constant is at most 1.

2. (10 points) Implement a function enforcing the 1-Lipschitz constraint of the dual variable and the generator and dual variable functions (in `variables.py`). Then implement an stochastic estimate of the objective function of the minimax game (in `trainer.py`):

$$\min_{g\in\mathcal{G}} \max_{f\in\mathcal{F}} \boldsymbol{E}[f(X) - f(g(Z)))] \tag{2}$$

   where $X$ has the true distribution, and $Z$ has the noise distribution. In order to implement the stochastic estimate you will use samples from such distributions. Use the methods available for the class `torch.distributions.Distribution` from the PyTorch module.

3. (25 points) complete the missing functions in `trainer.py`, You should implement alternating and simultaneous stochastic gradient ascent/descent updates. More specifically

$$f_{k+1} = f_k + \gamma \mathrm{SG}_f(f_k, g_k), \qquad g_{k+1} = g_k - \gamma \mathrm{SG}_g(f_k, g_k) \qquad \text{(simultaneous)} \tag{3}$$

$$f_{k+1} = f_k + \gamma \mathrm{SG}_f(f_k, g_k), \qquad g_{k+1} = g_k - \gamma \mathrm{SG}_g(f_{k+1}, g_k) \qquad \text{(alternating)} \tag{4}$$

Where SG is the stochastic gradient oracle.

in the file `optim.py` we provide a modification of PyTorch's SGD optimizer, which allows for *negative* learning rates (stepsize). This is only a trick to be able to perform stochastic gradient descent for the generator, and stochastic gradient ascent for the dual variable. PyTorch's `Optimizer.step()` method always performs a step in the direction of the negative gradient; by allowing both positive and negative learning rates we can effectively switch between gradient descent or ascent.

Run both methods using the script `train.py` passing the option `--training_mode simultaneous` or `--training_mode alternating`. Include the generated plots in your report. Comment on your findings.

4. (10 points) Show that given two distributions $\mu$ and $\nu$, if $\mathcal{F}$ is the class of functions defined in (1), it holds that:

$$\max_{f \in \mathcal{F}} \boldsymbol{E}_{X\sim\mu}[f(X)] - \boldsymbol{E}_{Y\sim\nu}[f[Y]] \geq 0 \tag{5}$$

5. (15 points) Show that given two distributions $\mu$ and $\nu$ on $\mathbb{R}^2$, if their first moments coincide i.e.,

$$\boldsymbol{E}_{(x_1,x_2)\sim\mu}[x_1] = \boldsymbol{E}_{(x_1,x_2)\sim\nu}[x_1], \qquad \boldsymbol{E}_{(x_1,x_2)\sim\mu}[x_2] = \boldsymbol{E}_{(x_1,x_2)\sim\nu}[x_2] \tag{6}$$

then, if $\mathcal{F}$ is the class of functions defined in (1), it holds that:

$$\max_{f \in \mathcal{F}} \boldsymbol{E}_{X\sim\mu}[f(X)] - \boldsymbol{E}_{Y\sim\nu}[f[Y]] = 0 \tag{7}$$

Why is this a possible explanation to the observed behaviour of our GANs example?

## 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 $28 \times 28$ 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 $\gamma$ |
| **1.** initialize $\theta_0$<br>**2.** **For** t = 0, 1,... ,N-1:<br>obtain the minibatch gradient $\hat{\mathbf{g}}_t$<br>update $\theta_{t+1} \leftarrow \theta_t - \gamma\hat{\mathbf{g}}_t$ |

| **Minibatch SGD with Momentum** |
|---|
| **Input:** learning rate $\gamma$, momentum $\rho$ |
| **1.** initialize $\theta_0$, $\mathbf{m_0} \leftarrow \mathbf{0}$<br>**2. For** t = 0, 1,... ,N-1:<br>obtain the minibatch gradient $\hat{\mathbf{g}}_t$<br>update $\mathbf{m}_{t+1} \leftarrow \rho\mathbf{m}_t + \hat{\mathbf{g}}_t$<br>update $\theta_{t+1} \leftarrow \theta_t - \gamma\mathbf{m}_{t+1}$ |

(c) Implement of following optimizers:

(a) (10 points) Implement AdaGrad method

| **AdaGrad** |
|---|
| **Input:** global learning rate $\gamma$, damping coefficient $\delta$ |
| **1.** initialize $\theta_0$, $\mathbf{r} \leftarrow \mathbf{0}$<br>**2. For** t = 0, 1,... ,N-1:<br>obtain the minibatch gradient $\hat{\mathbf{g}}_t$<br>update $\mathbf{r} \leftarrow \mathbf{r} + \hat{\mathbf{g}}_t \odot \hat{\mathbf{g}}_t$<br>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 RMSProp

| **RMSProp** |
|---|
| **Input:** global learning rate $\gamma$, damping coefficient $\delta$, decaying parameter $\tau$ |
| **1.** initialize $\theta_0$, $\mathbf{r} \leftarrow \mathbf{0}$<br>**2. For** t = 0, 1,... ,N-1:<br>obtain the minibatch gradient $\hat{\mathbf{g}}_t$<br>update $\mathbf{r} \leftarrow \tau\mathbf{r} + (1-\tau)\hat{\mathbf{g}}_t \odot \hat{\mathbf{g}}_t$<br>update $\theta_{t+1} \leftarrow \theta_t - \frac{\gamma}{\delta + \sqrt{\mathbf{r}}}\hat{\mathbf{g}}_t$ |

(c) (10 points) Implement Adam method

| **Adam** |
|---|
| **Input:** global learning rate $\gamma$, damping coefficient $\delta$, first order decaying paramter $\beta_1$, second order decaying parameter $\beta_2$ |
| **1.** initialize $\theta_0$, $\mathbf{m}_1 \leftarrow \mathbf{0}$, $\mathbf{m}_2 \leftarrow \mathbf{0}$<br>**2. For** t = 0, 1,... ,N-1:<br>obtain the minibatch gradient $\hat{\mathbf{g}}_t$<br>update $\mathbf{m}_1 \leftarrow \beta_1\mathbf{m}_1 + (1-\beta_1)\hat{\mathbf{g}}_t$<br>update $\mathbf{m}_2 \leftarrow \beta_2\mathbf{m}_2 + (1-\beta_2)\hat{\mathbf{g}}_t \odot \hat{\mathbf{g}}_t$<br>correct bias $\hat{\mathbf{m}}_1 \leftarrow \frac{\mathbf{m}_1}{1-\beta_1^{t+1}}$ $\hat{\mathbf{m}}_2 \leftarrow \frac{\mathbf{m}_2}{1-\beta_2^{t+1}}$<br>update $\theta_{t+1} \leftarrow \theta_t - \gamma\frac{\hat{\mathbf{m}}_1}{\delta + \sqrt{\hat{\mathbf{m}}_2}}$ |

(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 obtained average 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 4:00PM, 15 November, 2019
- 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.
- Questions of 0 points are for self study. You do not need to answer them in the report.
- 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 eligible.
- 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 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_2019_hw2_NameSurname.zip`, and submit it through the moodle page of the course.