Homework 5

10-405/10-605: Machine Learning with Large Datasets

Due Wednesday, April 12th at 11:59 PM Eastern Time

Instructions: Submit your solutions via Gradescope, *with your solution to each subproblem on a separate page*, i.e., following the template below.

Note that Homework 5 consists of two parts: this written assignment, and a programming assignment. Remember to fill out the collaboration section found at the end of this homework as per the course policy.

10-405 Students are required to complete the following:

1. Written Section 1.1 *[20 points]*
2. Written Section 1.3 *[20 points]*
3. All of the Programming (Section 2) *[40 points]*

Your homework score will be calculated as a percentage over the maximum points you can earn, which is 80.

10-605 Students are required to complete **all** sections of the homework. Your homework score will be calculated as a percentage over the maximum points you can earn, which is 100.

Programming: The programming in this homework **is** autograded so make sure that you upload your notebooks to the Programming submission slot and do not add any additional code boxes.

1 Written Section [60 Points]

1.1 Fast randomized linear algebra (20 pts)

Consider the following dataset:

i	x_i	y_i
1	-2	-1
2	-1	-1
3	1	1
4	2	1

We would like to fit the model $y_i = wx_i$ to this dataset by linear regression. (For this question we will use plain regression, with no regularization.) For the tasks below, you may use any desired combination of computer-assisted and manual calculation, but you should include your code if any in your answer.

(a) *[4 points]* Recall that the statistical leverage scores are the squared row norms of the matrix U from the SVD of the data matrix $X = USV^T$. (Here we use the compact version of the SVD, so that the number of columns of U is the rank of X.) What are the leverage scores of the four datapoints?

(b) *[6 points]* Suppose that we take a leverage-based sample of size 1, and happen to get the set of indices $\{3\}$. Use the Nyström method to find a vector of predictions $(\hat{y}_1, \hat{y}_2, \hat{y}_3, \hat{y}_4)$.

(c) *[4 points]* Now instead we would like to use random projection to solve this regression problem. Suppose that the sampled projection matrix turns out to be

$$P = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \end{pmatrix} \text{diag}(-1, 1, 1, 1)$$

What value of w do we get by regressing $Py \approx PXw$?

(d) *[6 points]* Compute the true optimal value of w and compare to the approximations you got in the previous two parts. Consider what would have happened if we had used a different set of Nyström points in part (b), or a different vector of random signs in part (c); would the result be the same?

1.2 Adam Can Fail Badly (20 pts) **10-605 Students Only**

1.2.1 Motivation

The Adam optimization algorithm is one of the most popular methods for neural network training [The paper can be accessed here: https://arxiv.org/abs/1412.6980]. It is implemented in tensorflow and pytorch as well.

To analyze the convergence of Adam, we will use the following online learning framework. We are given an arbitrary, unknown sequence of cost functions: $f_1(\theta), f_2(\theta), \ldots, f_T(\theta)$. For example, $f_t(\theta) = \|y_t - g_\theta(x_t)\|^2$ could be the loss on the (x_t, y_t) training instance. Here g_θ is our model that we want to train using our training data. Our goal is to iteratively minimize the empirical risk, $J_T(\theta) = \sum_{t=1}^{T} f_t(\theta)$ over a set of feasible points $\theta \in \mathcal{F}$. Let θ^*, denote the optimal point, that is,

$$\theta^* = \arg\min_{\theta \in \mathcal{F}} \sum_{t=1}^{T} f_t(\theta)$$

To measure the performance of Adam, we will use the following quantity, which is called the regret after T iterations:

$$R(T) = \sum_{t=1}^{T} [f_t(\theta_t) - f_t(\theta^*)]$$

Note that $R(T) \geq 0$, and if $\theta_t = \theta^*$ for all $t \in \{1, \ldots, T\}$, then $R(T) = 0$. In the original Adam paper, it has been proved (Corollary 4.2.) that under some conditions the average regret of Adam is converging to zero,

$$\lim_{T \to \infty} \frac{R(T)}{T} = 0.$$

In this exercise, your goal is to prove that this claim is incorrect. We will create a simple example, that satisfies the conditions of Corollary 4.2. Here each of the functions f_t is a simple 1D convex function on domain $\mathcal{F} = [-1, 1]$, and we will see that in this example

$$\liminf_{T \to \infty} \frac{R(T)}{T} > 0.$$

What's more, let

$$\theta_{BAD} = \arg\max_{\theta \in \mathcal{F}} \sum_{t=1}^{T} f_t(\theta),$$

the maximum point of our objective instead of the minimum [θ_{BAD} is the worst possible point in the feasible set $\mathcal{F}$]. We will also see that on this particular example instead of minimizing the objective function, Adam will visit θ_{BAD} infinite many times as $T \to \infty$.

1.2.2 The Adam algorithm

Algorithm 1 below provides a generic adaptive framework that encapsulates many popular adaptive methods. Note the algorithm is still abstract because the "averaging" functions ϕ_t and ψ_t have not been specified. Here $\phi_t : \mathcal{F}^t \to \mathbb{R}^d$ and $\psi_t : \mathcal{F}^t \to \mathcal{S}_+^d$. Here $\mathcal{S}_+^d$ denotes the set of $d \times d$ positive semidefinite matrices. For ease of exposition, we refer to α_t as step size and $\alpha_t V_t^{-1/2}$ as learning rate of the algorithm and furthermore, restrict ourselves to diagonal variants of adaptive methods encapsulated by Algorithm 1 where $V_t = \text{diag}(v_t)$. These methods are called "adaptive", because they can adaptively change their learning rates based on the values of the gradients in previous iterations. We first observe that the standard stochastic gradient algorithm falls in this framework as a special case by using:

$$\phi_t(g_1, \ldots, g_t) = g_t \quad \text{and} \quad \psi_t(g_1, \ldots, g_t) = \mathbb{I}, \tag{SGD}$$

Algorithm 1 Generic Adaptive Method Setup

Input: $x_1 \in \mathcal{F}$, step size $\{\alpha_t > 0\}_{t=1}^T$, sequence of functions $\{\phi_t, \psi_t\}_{t=1}^T$
for $t = 1$ **to** T **do**
  $g_t = \nabla f_t(x_t)$
  $m_t = \phi_t(g_1, \ldots, g_t)$ and $V_t = \psi_t(g_1, \ldots, g_t)$
  $\hat{x}_{t+1} = x_t - \alpha_t m_t / \sqrt{V_t}$
  $x_{t+1} = \Pi_{\mathcal{F}, \sqrt{V_t}}(\hat{x}_{t+1})$
end for

and $\alpha_t = \alpha/\sqrt{t}$ for all $t \in [T]$.

A particularly popular variant of Adam uses the following averaging functions:

$$\phi_t(g_1, \ldots, g_t) = (1-\beta_1)\sum_{i=1}^{t} \beta_1^{t-i} g_i \quad \text{and} \quad \psi_t(g_1, \ldots, g_t) = (1-\beta_2)\text{diag}(\sum_{i=1}^{t} \beta_2^{t-i} g_i^2), \tag{Adam}$$

for some $\beta_1, \beta_2 \in [0, 1)$. This update can alternatively be stated by the following simple recursion:

$$m_{t,i} = \beta_1 m_{t-1,i} + (1-\beta_1) g_{t,i} \text{ and } v_{t,i} = \beta_2 v_{t-1,i} + (1-\beta_2) g_{t,i}^2 \tag{1}$$

and $m_{0,i} = 0$ and $v_{0,i} = 0$ for all $i \in [d]$. and $t \in [T]$. A value of $\beta_1 = 0.9$ and $\beta_2 = 0.999$ is typically recommended in practice. We note the additional projection operation in Algorithm 1. When $\mathcal{F} = \mathbb{R}^d$, the projection operation is an identity operation. This projection operation is needed to make sure the algorithm generates feasible points: $x_t \in \mathcal{F}$.

Without loss of generality, assume that the initial point is $\theta_1 = 1$.

1.2.3 An example where Adam fails

We consider the setting where f_t are linear functions and $\mathcal{F} = [-1, 1]$. In particular, we define the following function sequence:

$$f_t(x) = \begin{cases} Cx, & \text{for } t \bmod 3 = 1 \\ -x, & \text{otherwise,} \end{cases}$$

where $C > 2$, and $x \in \mathcal{F} = [-1, 1]$.

Questions:

(a) *[1 points]* What is $\nabla f_t(x)$?

(b) *[1 points]* What is the minimum value of f_t?

(c) *[2 points]* What is the objective function $J(x) = \sum_{t=1}^{T} f_t$? [Write this as a function of T,C,and x]

(d) *[2 points]* What is the location $x^* \in [-1, 1]$ where the empirical risk $J(x) = \sum_{t=1}^{T} f_t(x)$ is minimal?

(e) *[2 points]* Prove that the location where the empirical risk $J(x) = \sum_{t=1}^{T} f_t(x)$ is maximal is $x_{BAD} = 1$

Consider the execution of Adam algorithm for this sequence of functions with

$$C = 3,\ \beta_1 = 0,\ \beta_2 = \frac{1}{1+C^2} = \frac{1}{10}, \alpha = \frac{1}{2},\ \text{and}\ \alpha_t = \frac{\alpha}{\sqrt{t}} = \frac{1}{2\sqrt{t}}$$

Assume that the initial point is $x_1 = 1$. We can observe that the conditions on the parameters required for the convergence of Adam in Corollary 4.2. are satisfied.

Our main claim is that for iterates $\{x_t\}_{t=1}^{\infty}$ arising from the updates of ADAM, we have $x_t > 0$ for all $t \in \mathbb{N}$ and furthermore, $x_{3t+1} = x_{BAD} = 1$ for all $t \in \mathbb{N} \cup \{0\}$, that is x_t visits $x_{BAD} = 1$ infinite many times, and will never get arbitrary close to the optimal $x^* < 0$!

For proving this, we resort to the principle of mathematical induction. Since $x_1 = 1$, both the aforementioned conditions hold for the base case. Suppose for some $t \in \mathbb{N} \cup \{0\}$, we have $x_i > 0$ for all $i \in [3t+1]$ and $x_{3t+1} = 1$. Using the Adam update rules, Equation (1), our aim is to prove that i) $0 < x_{3t+2} < 1$, ii) $0 < x_{3t+3}$, and iii) $x_{3t+4} = 1$. Your next task will be to prove the first part. The second and third parts can be proven similarly, and those proofs are not required in this exercise.

(f) *[12 points]* Suppose for some $t \in \mathbb{N} \cup \{0\}$, we have $x_i > 0$ for all $i \in [3t+1]$ and $x_{3t+1} = 1$. Prove that $0 < x_{3t+2} < 1$

1.3 Grid versus Random Search (20 pts)

We define an L_∞ ϵ-cover of a set S as a subset $E \subset S$ that satisfies the following property:

$$\forall y \in S, \exists x \in E \text{ s.t. } \|y - x\|_\infty \leq \epsilon$$

In words, every coordinate of every point in set S is within ϵ of a point in our cover E. The ϵ-covering number of S is the cardinality of the smallest ϵ-cover of S.

$$N(\epsilon, S) = \min\{|E| : E \text{ is an } \epsilon\text{-cover of } S\}$$

There are two basic strategies commonly employed in hyperparameter search: grid search and random search. In grid search, we choose a set of values to try for each hyperparameter, often equally spaced between a min and max value. The configurations are the Cartesian product of these sets, and we search through them in sequential order, like nested `for` loops. Random search instead chooses a distribution for each hyperparameter, often uniform between a min and max value. It then samples a new independent random value for each hyperparameter at each configuration.

Imagine we are in the (extreme) case where we have d hyperparameters all in the range $[0, 1]$, but only *one* of them (h_1) has any impact on the model, while the rest have *no* effect on the model. For simplicity, assume that for a fixed value of h_1 our training procedure will return the exact same trained model every time, regardless of the values of the other hyperparameters; and changing h_1 will always result in a distinct model.

For grid search, we will pick an even grid with k elements, for some positive integer k (see part (a) below). For random search, we will use the uniform distribution $U(0, 1)$.

(a) *[4 points]* An even grid is one where the points are evenly spaced, so that the farthest we can get from any point is exactly ϵ before we reach either the end of the interval or the region where another point is closer. If we want k points in an even grid, what will be the value of ϵ, and what will be the values of the grid points?

(b) *[4 points]* Imagine that we run grid search with the even grid defined in the previous part, which is just fine enough to provide ϵ-coverage. How many distinct hyperparameter settings will we try? How many distinct models will this training procedure produce? The ratio of these two numbers is the fraction of times we see a new model; what is this fraction?

(c) *[5 points]* Alternatively, if we run random search, then what fraction of the configurations will give us a new model? Why?

(d) *[7 points]* Now let's change our point of view: instead of desiring ϵ-coverage, we have a fixed budget of $B \ll |E|$ configurations to try (where $|E|$ is the size of an ϵ-cover of S). Which search strategy—grid or random search—should we employ? Why?

2 Programming Section [40 Points]

2.1 Introduction

The goal of this assignment is to gain familiarity with deep learning in TensorFlow. There are two parts:

In **Part 1**, you will implement *neural style transfer* in TensorFlow. Neural style transfer will involve building a system that can take in two images (one content image and one style image), and output a third image whose content is closest to the content of the content image while the style matches the style of the style image.

In **Part 2**, you will implement several of the optimization methods discussed in lecture including Gradient Descent, SGD, AdaGrad, and Adam, and train a linear model with them using Tensorflow 2.11. [*Note: you can't use tf.compat.v1.train.Optimizer or tf.keras.optimizers in the final evaluation.*]

2.2 Logistics

We provide the code template for this assignment in *two* Jupyter notebooks. Follow the instructions in the notebooks and implement the missing parts marked with '`<FILL_IN>`' or '`# YOUR CODE HERE`'. Most of the '`<FILL_IN>/YOUR CODE HERE`' sections can be implemented in just a few lines of code.

2.3 Getting lab files

You can obtain the notebooks `hw5_part1.ipynb` and `hw5_part2.ipynb` after downloading and unzipping the `Source code (zip)` from the course website.

To run these notebooks, you can upload the notebooks to your Google drive and open them with Google Colaboratory (Colab).

2.4 Preparing for submission

We provide several public tests via `assert` in the notebook. You may want to pass all those tests before submitting your homework. You can individually submit a notebook for debugging but **make sure to submit the notebook for your final submission to receive full credit.**

In order to enable auto-grading, please do not change any function signatures (e.g., function name, parameters, etc) or delete any cells. If you do delete any of the provided cells (even if you re-add them), the autograder will fail to grade your homework. If you do this, you will need to re-download the homework files and fill in your answers again and resubmit.

2.5 Submission

1. Download the notebooks from Colab to your local computer by going to `File -> Download .ipynb` and submit them to the corresponding Gradescope grader.

2. Submit the completed notebooks via Gradescope.

2.6 Part A: Neural Style Transfer in Tensorflow

In this part you will implement style transfer based on the paper: "A Neural Algorithm of Artistic Style": https://arxiv.org/pdf/1508.06576.pdf.

Basically, we will build a system that can take in two images (one input content image and one input style image), and output another image whose content is closest to the content of the content image while style is closest to the style of the style image.

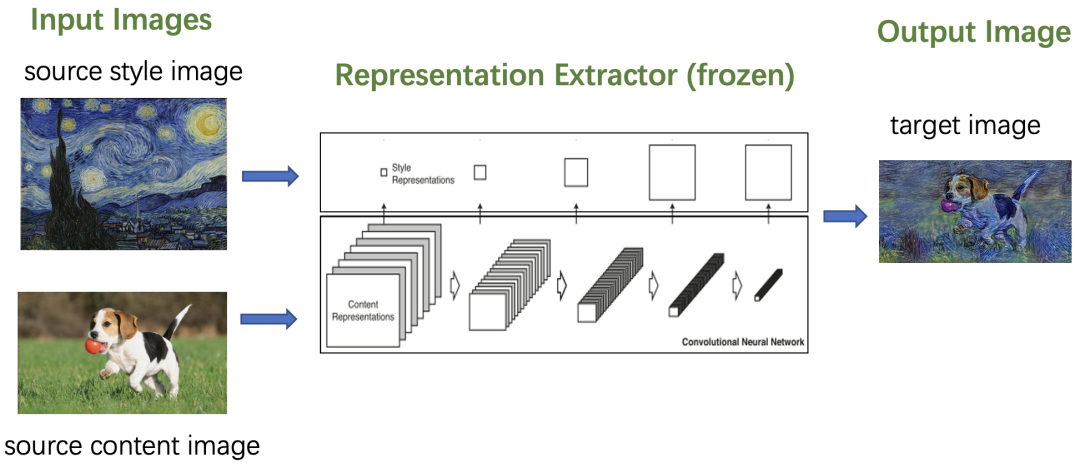

Figure 1: Overview of style transfer.

As depicted in Fig. 1, neural style transfer is able to generate an image whose content is close to the content image (a dog running on the grass with a ball in its mouth), but painted in the style of the style image (Van Gogh's "The Starry Night"). As the name suggests, it transfers the "style" of one image to another one. With this technique, we can obtain for an arbitrary image, its equivalent in different artwork styles. Towards this end, we consider two aspects of an image's representation, content representation and style representation. We make the assumption that we could have a pre-trained image encoder as a representation extractor that is able to disentangle content information and style information of an image. We also assume some intermediate layers capture content representation and some capture style representation. And the target image is expected to be close to the source content image in terms of the content representation, and close to the source style image in terms of the style representation.

Specifically, we utilize a pre-trained VGG model and keep its parameters frozen. We define the layers of interest within the model: content layers and style layers. For a given image, we refer to the output features of the content layers within the model with this image as the input, as the content representations of the image (w.r.t. the model). Like-wise, we define for an image its style representation (w.r.t. the model) as the gram matrix of the output features of the style layers within the model with this image as the input. For example, for a 5-layer perceptron, if we select the content layers to be the second and fourth layer, then the content representation of an image would be the output features from the second and fourth layer. The training objective is to find an output image that could minimize the combination of content loss and style loss, where content loss is the distance between the content representation of the input content image and the output image, and style loss is the distance between the style representation of the input style image and the output image. The only trainable tensor is the output image, which could be optimized via gradient descent.

There are 5 parts of the assignment:

2.6.1 Visualize data

The first part is written for you.

2.6.2 Prepare the data

The second part has the following functions:

1. `load_and_process_img()`: Load and process the image at the given path.
2. `deprocess_img()`: Perform inverse processing on the input image

2.6.3 Creating the model

The third part is to create the model:

1. Define content and style representations (we need to define the content layers and style layers)
2. `model_VGG()`: Builds the model (Use `tf.keras.applications.vgg19.VGG19` to get `style_outputs` and `content_outputs`)

2.6.4 Loss functions

The fourth part is to define and create the loss functions:

1. `compute_content_loss()`
   Use `tf.reduce_mean` and implement the formula: $L^l_{\text{content}}(p,x) = \sum_{i,j}(F^l_{ij}(x) - P^l_{ij}(p))^2$
2. `gram_matrix()`
   The gram matrix, G^l_{ij}, is the inner product between the vectorized feature map i and j in layer l. Remember to divide the `gram_matrix` by `input_tensor.shape[0]`
3. `compute_style_loss()`
   We need to implement the formula: $E_l = \sum_{i,j}(G^l_{ij} - A^l_{ij})^2$

Now that we have defined the loss functions, we need to develop the following functions:

1. `compute_features()`
   This function has following arguments:
   `model`: The model that we are using.
   `content_path`: The path to the content image.
   `style_path`: The path to the style image.
   Returns: The style features and the content features.
2. `calculate_loss()`
   Returns: `total_loss`, `style_loss`, `content_loss`
   We will give more emphasis to deep layers. For example, the weights for `block1conv1` can be 1, for `block2conv1` it can be 2, and so on `weight_per_style_layer = [1.0,2.0,3.0,4.0,5.0]`

2.6.5 Optimization loop

The final part is to optimize the loop so that we can get the best stylized image.

2.7 Part B: Optimization Methods

In this portion of the homework you will implement several common optimizers. You will first build a simple linear regression model and train it by calling existing optimizers provided by tf.keras. Then you will implement and evaluate your own optimizers. Finally you will have a better understanding of the mechanism of different optimizers and their effectiveness on simple models. Detailed instructions are included in the assignment notebook.

There are 4 optimizers you need to implement:

- Gradient Descent: update weights using all samples to reach the minimum of the loss function.
- Stochastic Gradient Descent: update weights using one sample at a time to reach the minimum of the loss function.
- AdaGrad: decrease the step size for coordinates with high gradient magnitudes with a cheap approximation of the hessian.
- Adam: combine momentum (move further in the correct direction and less in the wrong direction) and RMSprop (take a moving average of the coordinates of the gradient to put moreemphasis on the current gradients).

The instructions in the notebook will guide you to implement each of the optimizers. To understand the theory behind them, we encourage you to read the update rule provided in the notebook, as well as the details provided in lecture 18.

3 Collaboration Questions

1. (a) Did you receive any help whatsoever from anyone in solving this assignment?

   (b) If you answered 'yes', give full details (e.g. "Jane Doe explained to me what is asked in Question 3.4")

2. (a) Did you give any help whatsoever to anyone in solving this assignment?

   (b) If you answered 'yes', give full details (e.g. "I pointed Joe Smith to section 2.3 since he didn't know how to proceed with Question 2")

3. (a) Did you find or come across code that implements any part of this assignment?

   (b) If you answered 'yes', give full details (book & page, URL & location within the page, etc.).