Homework 5

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

Due Wednesday, April 14th at 11:59:59 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. The written part is worth **50%** of your total HW5 grade. The programming part makes up the remaining 50%.

Submitting via Gradescope: When submitting on Gradescope, you must assign pages to each question correctly (it prompts you to do this after submitting your work). This significantly streamlines the grading process for the course staff. Failure to do this may result in a score of 0 for any questions that you didn't correctly assign pages to. It is also your responsibility to make sure that your scan/submission is legible so that we can grade it.

1 Written Section [50 Points]

1.1 Autodiff: Simple Neural Network (24 pts)

Consider a feedforward neural network, defined by the following composition of functions:

$$\mathbf{q} = W^{(1)}x + \mathbf{b^{(1)}} \tag{1}$$
$$\mathbf{h} = ReLU(\mathbf{q}) = \max(\mathbf{q}, 0) \tag{2}$$
$$\mathbf{p} = W^{(2)}\mathbf{h} + \mathbf{b^{(2)}} \tag{3}$$
$$\mathbf{L}(\mathbf{y}, \mathbf{p}) = (\mathbf{p} - \mathbf{y})^T(\mathbf{p} - \mathbf{y}) \tag{4}$$

Here $x \in \mathbb{R}^D$ is an input observation/data point, $W^{(1)} \in \mathbb{R}^{H^{(1)} \times D}$ is a set of weights corresponding to the first (input) layer, $\mathbf{b^{(1)}} \in \mathbb{R}^{H^{(1)}}$ is an offset term corresponding to the first layer, $W^{(2)} \in \mathbb{R}^{H^{(2)} \times H^{(1)}}$ is a set of weights corresponding to the second layer, and $\mathbf{b}^{(2)} \in \mathbb{R}^{H^{(2)}}$ is an offset term corresponding to the second layer. Note also that the max in step 2 is applied element-wise.

Our goal is to find the gradient of the loss, L, with respect to the weights W and b using backpropogation. We will first explore backpropagation on the simple neural network above, and then a more general ("deep") network in Problem 2. Note that you may need the Kronecker delta:

$$\delta_{ij} = \begin{cases} 0 & i \neq j \\ 1 & i = j \end{cases}$$

(a) *[10 points]* First derive the following five terms (Backward from loss to second hidden layer, from second hidden layer to output of first hidden layer and second set of weights):

$$\frac{\partial L}{\partial p_j}, \frac{\partial p_i}{\partial W^{(2)}_{j,k}}, \frac{\partial L}{\partial W^{(2)}_{j,k}}, \frac{\partial L}{\partial b^{(2)}_j}, \frac{\partial p_i}{\partial h_j}$$

Feedforward neural network:

$$\mathbf{q} = W^{(1)}x + \mathbf{b^{(1)}} \tag{1}$$
$$\mathbf{h} = ReLU(\mathbf{q}) = \max(\mathbf{q}, 0) \tag{2}$$
$$\mathbf{p} = W^{(2)}\mathbf{h} + \mathbf{b^{(2)}} \tag{3}$$
$$\mathbf{L}(\mathbf{y}, \mathbf{p}) = (\mathbf{p} - \mathbf{y})^T(\mathbf{p} - \mathbf{y}) \tag{4}$$

(b) *[6 points]* Now derive the following three terms (Backward from loss to first hidden layer and first preactivation):

$$\frac{\partial h_j}{\partial q_i}, \frac{\partial L}{\partial h_i}, \frac{\partial L}{\partial q_i}$$

Feedforward neural network:

$$\mathbf{q} = W^{(1)}x + \mathbf{b^{(1)}} \tag{1}$$
$$\mathbf{h} = ReLU(\mathbf{q}) = \max(\mathbf{q}, 0) \tag{2}$$
$$\mathbf{p} = W^{(2)}\mathbf{h} + \mathbf{b^{(2)}} \tag{3}$$
$$\mathbf{L}(\mathbf{y}, \mathbf{p}) = (\mathbf{p} - \mathbf{y})^T(\mathbf{p} - \mathbf{y}) \tag{4}$$

(c) *[8 points]* Finally, derive the following four terms, which are what we set out to calculate (backward from first preactivation to first set of weights, biases and loss to these values):

$$\frac{\partial q_i}{\partial W^{(1)}_{j,k}}, \frac{\partial q_i}{\partial b^{(1)}_j}, \frac{\partial L}{\partial W^{(1)}_{j,k}}, \frac{\partial L}{\partial b^{(1)}_i}$$

1.2 Autodiff: Deep Learning (26 pts)

Now, let's generalize Problem 1 to a larger neural network. Suppose that we create a deep network by using L fully connected hidden layers with ReLU activations, and we use the squared loss for the final layer (i.e., similar to the network in Problem 1, but with L hidden layers).

(a) *[8 points]* Write down the expression for inference for this neural network (i.e., the forward pass), remember to include bias term. (Hint: you may want to use a recurrence relation, similar to what we saw in lecture.)

(b) *[5 points]* Suppose that the sizes of the layers are $H^{(0)} = H^{(1)} = \cdots = H^{(L)} = 1$ ($H^{(0)}$ refers to the input layer). What is the computational cost of running inference on the deep neural network described in part (a)? In other words, how many numerical operations would be required to compute the forward pass? Which operation dominates in terms of number of operations: the matrix multiplies or the nonlinearities?

(c) *[8 points]* Now write down the expression for backpropagation on this deep neural network. That is, how would you compute the gradient of the network with respect to the weights for a training example x? (Hint: This should look very similar to what you have in Problem 1. You don't have to evaluate any of the terms and multiply it out).

(d) *[5 points]* What is the computational cost of running backpropagation on this deep neural network? In other words, how many numerical operations would be required to perform backpropagation? Which operation dominates in terms of number of operations: the matrix multiplies or the nonlinearities? Note that we make the same assumption as 2(b), $H^{(0)} = H^{(1)} = \cdots = H^{(L)} = 1$. You should calculate the cost of running backprop to compute the gradient with respect to a single training example x.

2 Programming Section [50 Points]

2.1 Introduction

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

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 on March 30th including Gradient Descent, SGD, AdaGrad, and Adam, and train a linear model with them using Tensorflow 2.0. [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. What you need to do is to 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 one or two 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 https://github.com/10605/released-hws/releases/tag/s21-hw5.

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

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 content image and one 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.

There are 5 parts of the assignment:

- **Visualize data**
- **Process the data**
- **Set up loss functions**
- **Create Model**
- **Optimize for loss function**

The first part (visualizing the data) is written for you.
The second part has following functions:

1. `load_and_process_img()`
2. `deprocess_img()`
3. Define content and style representations (we need to define the content layers and style layers)
4. Build the Model (Use `tf.keras.applications.vgg19.VGG19` to get `style_outputs` and `content_outputs`)

The third 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 etween 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 fourth part which consists offollowing 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]`

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 these notes: https://drive.google.com/file/d/1PA0ARElkcUZYuu4fPk_AbrFKF1fwsfI-/view.

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.).