

# Homework 5

## 10-605/10-805: Machine Learning with Large Datasets

**Due Wednesday, November 15th 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. The written part is worth **60%** of your total HW5 grade. The programming part makes up the remaining **40%**.

**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 [60 Points]

## 1.1 BackProp (20 pts)

Suppose that we create a deep neural network by using  $L$  fully connected hidden layers (without considering input and output layers) with ReLU activation and with bias term, and we use the squared loss for the final layer.

- (a) [5 points] Write down the expression for inference for this neural network (i.e., the forward pass). Please use the following notation:

- $b_l$  for bias term of layer  $l$
- $\mathbf{W}_l$  for the weights of layer  $l$
- $\mathbf{o}_l$  for the output of layer  $l$
- $\sigma_l$  for activation function
- $h_l$  for the output of the linear layer  $l$  (before activation function)

(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  $H^{(0)} = H^{(1)} = \dots = H^{(L)} = 1$ , where  $H^{(0)}$  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 multiplication or the nonlinearities (ignore the bias term addition for both cases)?

- (c) *[5 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  $x$ ?
- (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 (ignore the bias term addition for both cases)? Note that we make the same assumption as 1(b),  $H^{(0)} = H^{(1)} = \dots = H^{(L)} = 1$ . You should calculate the cost of running backprop to compute the gradient with respect to a single training example  $x$ .

## 1.2 Curse of Dimensionality (20 pts)

We define the  $\epsilon$ -cover of a search space  $S$  as a subset  $E \subset S$ :

$$E = \{x \in \mathbb{R}^n \mid \forall y \in S, \exists x \text{ s.t. } \|y - x\|_\infty \leq \epsilon\}$$

In words, every coordinate of every point in our search space  $S$  is within  $\epsilon$  of a point in our set  $E$ . The  $\epsilon$ -covering number is the size of the smallest such set that provides an  $\epsilon$ -cover of  $S$ .

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

Let's work through a concrete example to make the concept clearer: let's say we want to tune the learning rate for gradient descent on a logistic regression model by considering learning rates in the range of  $[1, 2]$ . We want to have  $\epsilon = .05$ -coverage of our search space  $S = [1, 2]$ . Then  $E = \{1.05, 1.15, 1.25, 1.35, 1.45, 1.55, 1.65, 1.75, 1.85, 1.95\}$ . We see that any point in  $[1, 2]$  is within 0.05 of a point in  $E$ . So  $E$  provides  $\epsilon$ -coverage of  $S$ . The  $\epsilon$ -covering number is 10. The questions below will ask you to generalize this idea.

- (a) *[6 points]* Find the size of a set needed to provide  $\epsilon$ -coverage ( $\epsilon$ -covering number) of  $S = [0, 1] \times [0, 2]$ , the Cartesian product of two intervals. Note that we want this to hold for a generic  $\epsilon$ .

- (b) *[6 points]* Find the  $\epsilon$ -covering number of  $S = [a_1, b_1] \times \dots \times [a_d, b_d]$ , the Cartesian product of  $d$  arbitrary closed intervals. Specifically, write down an expression  $N(\epsilon, S)$  as a function of  $a_i$ ,  $b_i$ , and  $\epsilon$ .

- (c) *[8 points]* With respect to your answer in 1.2 (b), intuitively, how is the covering number related to the volume of  $S$ ? Moreover, on what order does the covering number grow with respect to the dimension  $d$ ? What does this say about the volume of  $S$  as a function of dimensionality?

### 1.3 Grid versus Random Search (20 pts)

There are two basic strategies commonly employed in hyperparameter search: grid search and random search. Grid search is defined as choosing an independent set of values to try for each hyperparameter and the configurations are the Cartesian product of these sets. Random search chooses a 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* ( $h_1$ ) of them has any impact on the model, while the rest have *no* effect on model performance. For simplicity, assume that for a fixed value of  $h_1$  that our training procedure will return the exact same trained model regardless of the values of the other hyperparameters.

- (a) [8 points] Imagine that we consider  $q$  hyperparameter configurations, which are enough to provide  $\epsilon$ -coverage for grid search. How many distinct models will this training procedure produce? What is this as a fraction of the total number of configurations?
  
  
  
  
  
  
  
  
  
  
- (b) [5 points] Alternatively, if we consider  $q$  random configurations as part of random search, then what percent of the configurations will cause a change to the model? Why?
  
  
  
  
  
  
  
  
  
  
- (c) [7 points] Now let's change our point of view: instead of desiring  $\epsilon$ -coverage, we have a fixed budget of  $B \ll |E|$  configurations to try (where  $|E|$  is an epsilon cover of  $S$ ). Which search strategy—grid or random search, should we employ? Why?

## 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:

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 2.10. [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 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:

### 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:  $L_{\text{content}}^l(p, x) = \sum_{i,j} (F_{ij}^l(x) - P_{ij}^l(p))^2$
2. `gram_matrix()`  
The gram matrix,  $G_{ij}^l$ , 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:  $E_l = \sum_{i,j} (G_{ij}^l - A_{ij}^l)^2$

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 more emphasis 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 lecture on October 27: <https://drive.google.com/file/d/1jmqI0gSs-qrDwxSxvIfpe0-A09jJtZNd/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.).