Homework 6 Written Assignment

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

Due Wednesday, April 28th 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 6 consists of two parts: this written assignment, and a programming assignment. The written part is worth **50%** of your total HW6 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.

As we consider more hyperparameters, the space we search over to find good hyperparameter configurations grows very quickly. In the first two questions, we will explore just how quickly this space grows. We will then study a simple example in the third question to see how early stopping can help speed up random search.

1 Written Section (50 pts)

1.1 Curse of Dimensionality (15 pts)

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

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

In words, every coordinate of every point in our search space S is within ϵ of a point in our set E. The ϵ-covering number is the size of the smallest such set that provides an ϵ-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 ϵ-coverage of S. The ϵ-covering number is 10. The questions below will ask you to generalize this idea.

(a) *[5 points]* Find the size of a set needed to provide ϵ-coverage (ϵ-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 ϵ.

(b) *[5 points]* Find the ϵ-covering number of $S = [a_1, b_1] \times \ldots \times [a_d, b_d]$, 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 ϵ.

(c) *[5 points]* With respect to your answer in 1.1 (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?

1.2 Grid versus Random Search (15 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) *[5 points]* Imagine that we consider q hyperparameter configurations, which are enough to provide ϵ-coverage for grid search. How many different models will we train? What is this as a percent 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) *[5 points]* Now we 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 an epsilon cover of S). Which search strategy - grid or random search, should we employ? Why?

1.3 Benefits of Early Stopping (20 pts)

Consider two configurations, and imagine we plot validation error as a function of number of iterations. This validation error is noisy since it's based on partially trained models. However, let's assume that these models eventually converge to some fixed validation errors, which implies that there exists some *envelope of uncertainty* bounding this noise. For the ith configuration, we will define this envelope of uncertainty by an upper bound $\gamma_{i,k}^{+}$ and a lower bound $\gamma_{i,k}^{-}$ that is a function of k, the number of iterations run. Moreover, we assume that the width of the envelope is monotonically non-increasing, i.e., $\gamma_{i,k}^{+} - \gamma_{i,k}^{-} \leq \gamma_{i,k-1}^{+} - \gamma_{i,k-1}^{-}$. See Figure 1 below for a visualization.

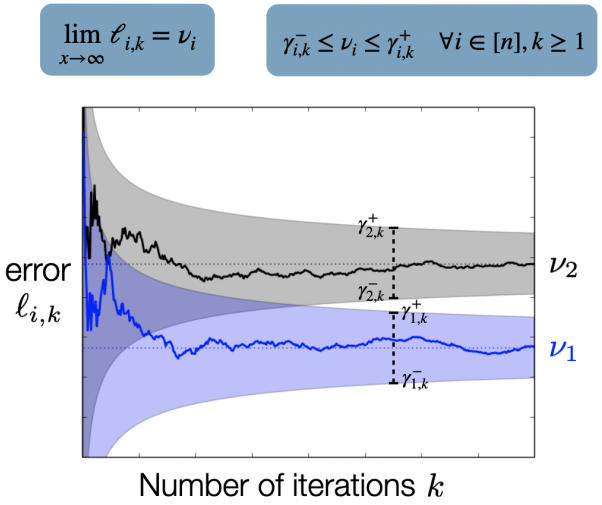

Figure 1: A visualization of the envelopes of uncertainty around the true convergent validation errors at each training iteration.

(a) *[5 points]* What is the range of possible values for configuration 1 validation error after iteration $k = 2$?

(b) *[5 points]* Now imagine that after training each configuration at iteration k, we gain access to $\gamma_{i,k}^{+}$ and $\gamma_{i,k}^{-}$ [Note: This is not a realistic assumption, but it is helpful in understanding how early-stopping works]. What is the earliest iteration at which we, with certainty, can know that configuration 1 is better than configuration 2?

(c) *[5 points]* Next, imagine the we have N configurations we're considering, assuming without loss of generality that the first configuration is the best one. Once again, we will assume that we have access to $\gamma_{i,k}^{+}$ and $\gamma_{i,k}^{-}$. Our goal is to find an efficient allocation of resources across these N configurations, i.e., to choose how many iterations to run each configuration such that we identity the best configuration with the fewest number of total iterations. Consider a uniform allocation strategy, as is done with vanilla random search, in which we are required to allocate the same number of iterations to all configurations. What is our required total budget in order to, with certainty, be able to identify that configuration 1 is the best?

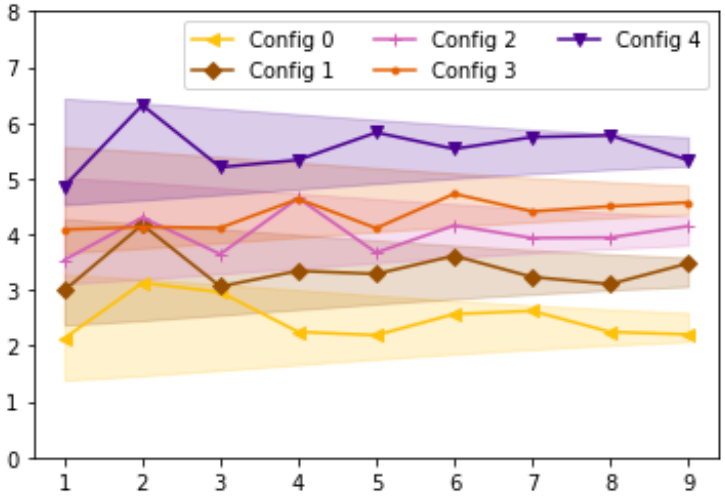

Figure 2: Validation loss as a function of iteration number for five configurations. Note that the shaded regions are the associated envelopes of uncertainty.

(d) *[5 points]* Finally, imagine we are considering the following $N = 5$ configurations shown in Figure 2, and that we want to compare the efficiency of the following two strategies:

(a) "uniform allocation": As defined in part (c) above, we uniformly allocate the same number of training iterations on each of the configurations;

(b) "adaptive allocation": We incrementally perform iterations on each configuration, decide whether to discard some of the configurations, and then repeat the process.

For both strategies, assume as before that after training the ith configuration at iteration k, we gain access to the associated envelope of uncertainty defined by $\gamma^{+}_{i,k}$ and $\gamma^{-}_{i,k}$ and visualized in Figure 2.

Under these assumptions, how many configuration-iterations will we have run to identify the best configuration for each strategy (please fill out the table below)? How does this adaptive allocation approach compare to the uniform allocation strategy?

Note: The total number of configuration-iterations is lower-bounded by 5 (running each configuration for one iteration and stopping) and 45 (running all configurations for all 9 iterations). Moreover, we will accept some minor variations to our expected final answers given that slight visual ambiguity of the shaded regions of uncertainty in Figure 2.

Configuration	**Iterations with Uniform Allocation**	**Iterations with Adaptive Allocation**
0		
1		
2		
3		
4		
sum		

2 Programming Section (50 pts)

2.1 Introduction

In this homework you will explore methods for neural network pruning as part of a class-wide competition. In particular, you will attempt to compress a simple neural network which has 592,933 parameters in total. It can achieve about 63% test accuracy in a 5-class classification task after training for 50 epochs with the default parameters provided in the notebook. To help you start, we provide a simple example of manipulating the weights, assigning the new weights to the model, and evaluating how that affects the accuracy in the notebook. You can explore various pruning techniques as you like. You might also want to read some references e.g. this or this.

2.2 Logistics

We will provide a notebook for you for this part which provides data loading helper functions, defines the model architecture, and provides some simple functionalities to save and load model weights for you to start with. To download the notebook for this part, go to this link. Make sure to use your andrew id. This homework is done on Colab.

2.2.1 Getting the data

Download the compressed training and validation data from this link using your andrew email ID. Upload it to Colab by going to the `Files` tab in the sidebar and clicking on the upload button. The first few cells in the notebook untar and load the training and validation sets. We will use a hidden test set on gradescope to test the final performance of your model. To avoid uploading the dataset everytime the runtime disconnects, you might want to upload the dataset to drive and mount it from colab.

2.2.2 Training and pruning the network

You need to first train a network in the notebook until convergence. Starting from this pre-trained full model, you need to explore different strategies to prune the network (i.e., setting some weights to zero) without degrading the accuracy. Your output should be a pre-trained model with the '.h5' extension with the same architecture as the neural network we define in the notebook. Changing the architecture or designing a new model architecture yourself is not allowed and will cause the autograder to fail. It might be helpful to save the weights of your model before proceeding with the pruning methods in order to save progress. You can achieve this through `model.get_weights()` and `model.set_weights(saved_weights)`.

2.2.3 Submission

You do not need to submit the notebook, but instead, have to submit the final pruned model with the name **"my_model_weights.h5"**. In the notebook, we specify how to download the model weights to your local work space. You will then upload the model weights to Gradescope for autograding. You can submit an unlimited number of times, and the final ranking will be determined by the last submission.

2.2.4 Evaluation

Each submission will receive a score, which is a function of the accuracy of the test data and the sparsity level of the model. We assume there is an efficient way to encode the indices of sparse matrices, therefore do not take into account the storage overhead for sparse matrix formats. In particular, the autograder uses the following functions to calculate the score of a model:

if accuracy > 0.6 and sparsity > 0:

score = (accuracy + num_zero_weights / total_parameters) / 2

else:

score = 0

This evaluation metric encourages to achieve a good trade off between accuracy and sparsity (the ratio between the number of zero weights and the number of total parameters) while ensuring a reasonable accuracy (accuracy > 0.6). You must obtain an accuracy higher than 0.6 and a total **score higher than 0.36** to receive full credit (though we encourage you to search for even higher scores as part of the competition!)

2.3 The competition

We will maintain a leaderboard on gradescope and plan to give bonus points to the top 5 submissions in this competition. In such competitions there is always a risk of data leakage but we expect that you will only use the provided train/validation data to train your model. At the end of the competition, we will ask the top 5 participants to submit their code and we will manually check to make sure that they used only the provided train/validation data. Note that you can't use any network pruning libraries in Tensorflow and we will manually verify the top 5 submissions to make sure that's not the case.

We hope that the competition is a fun learning experience for those who want to dive deep into model compression.

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.).