Homework 5 Programming Part 1

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

Due Monday, November 18th at 11:59 PM Eastern Time

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

IMPORTANT: Please use the provided template. If you do not follow the template (i.e. there is some misalignment), your assignment may not be graded correctly by our AI assisted grader and there will be a **2% penalty** (e.g., if the homework is out of 100 points, 2 points will be deducted from your final score).

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

All 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. The grading breakdown is as follows:

1. Written Section, due Wednesday, Nov. 20 at 11:59pm ET *[40 points]*
2. Programming Part 1, due Monday, Nov. 18 at 11:59pm ET *[50 points]*
3. Programming Part 2, due Wednesday, Nov. 20 at 11:59pm ET *[10 points]*

1 Programming Part 1 [50 Points]

1.1 Introduction - Zero Shot Hyperparameter Tuning

Llama2 70B cost an estimated $3.9 million to train, GPT-4 an estimated $78 million to train, and Gemeni Ultra an estimated $191 million. (Source: https://aiindex.stanford.edu/report/, Chapter 1, Figure 3). If training a machine learning model from scratch just once costs this much, how can we perform hyperparameter tuning?

In this homework we will consider the problem of "zero-shot" hyperparameter tuning – selecting hyperparemeters for a model which we only have sufficient budget to train to completion once. Traditional hyperparemeter tuning methods such as grid search or random search rely on being able to train several models to convergence and selecting the model with the best validation performance. This paradigm fails in the context of training massive foundation models on very large datasets, as these models can only feasibly be trained from scratch a very limited number of times.

1.2 Background

1.2.1 GPT-2 Class Models

In 2019, OpenAI released Generative Pre-Trained Transformer 2 (GPT 2), trained on a dataset containing 8 million web-pages with the objective of predicting the next word of a given sentence. The GPT-2 class of models are:

- GPT-2 Tiny (unofficial, 30 million parameters)
- GPT-2 Small (124 million parameters)
- GPT-2 Medium (350 million parameters)
- GPT-2 Large (774 million parameters)
- GPT-2 Extra-Large (1.5 billion parameters)

The sizes of these models scale the relative performance and computation costs. The GPT-2 class models are commonly used for tasks such as text-generation and question-answering, which you will be putting to the test in this assignment while you balance between model performance and computational cost.

1.2.2 HellaSwag Evaluation

HellaSwag is a popular LLM benchmark used to determine if LLM's have reached commonsense levels of inference. The questions in this benchmark are trivial for humans ($> 95\%$ accuracy), but LLM's have a much harder time of conducting inference that seem like common sense to humans. At the time, OpenAI's GPT-2 model struggled with solving such questions, reaching an accuracy of roughly 30%, thus one of the goals of this homework is to increase the HellaSwag accuracy of your models through the strategies that you will be devising.

1.3 Instructions

1.3.1 Overview

Your goal in this homework is to find the "max learning rate" hyperparameter which achieves minimal validation loss and maximal HellaSwag accuracy for a fixed training objective. The training objective is to pretrain from scratch a GPT-2 class model (GPT2-Tiny, depth=6, channels=384, heads=64, 31M parameters) on 10 billion tokens of training data from the FineWeb dataset.

Unfortunately, you will be constrained to only have $\sim$ \$40 [1]: sufficient budget to complete an entire pretraining run approximately twice. A naive strategy might be to simply train the model twice with two different max learning rates and choose the candidate max learning rate with better validation performance. It is your job to instead design a methodology to use this limited budget to evaluate a much larger set of candidate max learning rates ($\gg 2$).

The approach we recommend in this homework is the following: first, scale down some aspect of the training process. Then, perform hyperparameter tuning in this scaled-down setting. Finally, in the "transfer" stage, use this information to create an estimation of the optimal hyperparameters in the full-scale model. The following are all properties of the training process that might be scaled down:

1. The number of training iterations
2. The training dataset size
3. The model size (total number of parameters)
   (a) The model depth (number of layers)
   (b) The number of attention heads
   (c) The embedding dimension

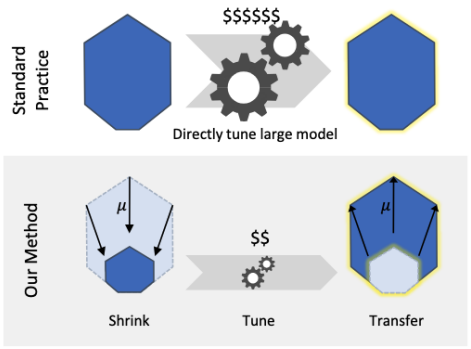


Figure 1: A visualization of using scaling to perform single-shot hyperparameter tuning. Source: Yang et al.

In addition to selecting the aspects of model training to scale down, a challenge you will face with this approach is the "transfer" stage. Namely, how do we form a prediction about the relationship between the optimal hyperparameters in a scaled-down setting vs. the full-scale setting? For example, suppose that you are investigating the scaling behavior of some parameter of the training process, Y. (as discussed, this might be the number of training iterations, the model size, or the training dataset size, etc.) For the "transfer" stage, you will want to form a hypothesis about the relationship between Y and a hyperparameter of choice α.

The following might be hypotheses that you explore:

1. No scaling. Assume that hyperparameters transfer between training processes of different scales. In other words, assume that the same hyperparameter values that are optimal in a small-scale training setting will be optimal for a large-scale training run.
2. Functional scaling. Assume that $\alpha = f(Y)$ for some function f. You might explore one of the following functional families:
   (a) Sublinear scaling

(b) Linear scaling
(c) Polynomial scaling
(d) Exponential scaling
(e) etc.

1.3.2 Deadlines and Deliverables

This homework will be split into two parts:

1. Part 1. Experimental overview and experiments. Deliverables:
   (a) Completed experiential design.
   (b) Completed cost analysis.
   (c) Completed hyperparameter tuning table.
   (d) Completed empirical questions.
   (e) Final selection of "max learning rate" hyperparameter.
2. Part 2. Final pre-training run. Deliverables:
   (a) Final HellaSwag accuracy
   (b) Final validation loss
   (c) Pre-trained model weights

1.3.3 Grading Criteria

This homework will be run in a competition format. You will need to submit your final learning rate and total hyperparameter tuning expenditure to a public leaderboard on Gradescope. Your final submission of model weights will be used to evaluate HellaSwag accuracy and validation loss on a held-out section of the data. Students who achieve top scores on HellaSwag accuracy and minimize validation loss will be invited to present the strategy they used in front of the class during lecture.

In other words, winners will not be awarded extra credit but will be awarded bragging rights ☺.

1.3.4 Total Budget

We ask you not to use more than the following budget in performing your hyperparameter tuning (Part 1). **Part 2 will require a fixed budget of $15-$20. Please leave $20 in your account or you will be unable to complete this homework.**

1. Part 1: $20 [1]
2. Part 2: $20

In order to avoid accidentally exhausting your entire budget with a single training run, it is **mandatory** to perform a cost analysis of each experiment in advance. This entails doing the following:

1. Run your experiment for just 20 steps. The first (0th) step is an anomaly and is much slower due to memory loading. Get a worst-case estimate from steps 10-19, just to be safe. Record which step in the range step 10 to step 19 took the longest, and how long it took (in milliseconds).

[1] If you have additional AWS credits, you may use all but the last $20 in Programming Part 1 (as mentioned in lecture). However, please leave enough to complete Programming Part 2 ($20)

2. Use the formula below to estimate the runtime of the full experiment in hours:

$$\texttt{Runtime (hours)} = \frac{\texttt{time per step (ms)} * \texttt{number of steps}}{3.6 \times 10^6 \texttt{ ms per hour}}$$

3. Multiply this value by the cost of running an AWS g5.xlarge per hour ($1.1066/hour at time of writing, but please round to $1.50/hour to be safe).

1.4 Getting Started

1.4.1 Requesting GPU Access from AWS

Please check your AWS vCPU Quota for "All G and VT Instances" as well as "Running On-Demand G and VT Instances". This should say at least "8." If you have not yet completed your AWS Quota request, please see the appendix for details on how to do so.

1.4.2 Launching your AWS Resources

The following is a high level overview of what you will need to do to setup your AWS instance and begin running experiments. For a much more detailed guide, see the appendix and recitation materials.

- Create a S3 bucket. This will be used to store your training logs.
- Create an IAM role and user (save your access key and secret key somewhere safe!). This is required to save your training logs to an S3 bucket.
- Create a g5.xlarge instance using 'ami-03f759bf03d179ab0' and providing the IAM role.

1.4.3 Setting up your AWS credentials

- Type `aws configure` into the command line, and pass in your access and secret key when prompted. Make sure the region matches that of your EC2 instance (likely `us-east-1`).

1.4.4 Getting the training code

- Clone the github repo: `git clone https://github.com/10605/llm.c`
- Compile the model: `make`

1.4.5 Getting the FineWeb10B dataset

- Create a 25GiB volume with snap-0246e798d0a57d4e0
- Attach this volume to your AWS instance
- Mount the volume in the directory `llm.c/data`

1.4.6 Running an Experiment

Now you are ready to run your own experiment! You can do so with the following command:

```
./train_gpt2cu \
    -o my_first_experiment \
    -x 20
```

1.5 Hyperparameters and Flags

Description	Parameter	Default Value
learning rate	−l	6e-4
output log dir	−o	`NULL, no logging`
max_steps of optimization to run	−x	-1, 1 epoch steps (20k)

Table 1: Flags which you will absolutely need to use in this homework

Description	Parameter	Default Value
model depth	−1d	6
model channels	−1c	384
model number of heads	−1h	6
write optimization checkpoints every how many steps?	−n	0 (don't)
resume optimization found inside output log dir? (0=restart/overwrite, 1=resume/append)	−y	0, restart
input .bin filename used to continue training a model	−e	

Table 2: Flags which might be very useful, depending upon your strategy

Description	Parameter	Default Value
train data filename pattern	−id	`data/fineweb_train_*.bin`
val data filename pattern	−iv	`data/fineweb_val_*.bin`
tokenizer file path	−it	`data/gpt2_tokenizer.bin`
hellaswag file path	−ih	`data/hellaswag_val.bin`
(per-GPU, micro) batch size B	−b	64

Table 3: Flags which may be useful for solving errors

Description	Parameter	Default Value
learning rate scheduler ("cosine", "linear", "constant", or "wsd")	−k	"cosine"
learning rate	−l	6e-4
learning rate warmup iterations	−u	700
learning rate decay: final fraction, at end of training	−q	0.0
weight decay	−c	0.1
total desired batch size	−d	524288

Table 4: Flags which you could play with, but we don't recommend modifying

Description	Parameter	Default Value
val_loss_every, how often we evaluate val loss	−v	250
sample_every, how often we inference the model	−s	20000
hellaswag eval run?	−h	1

Table 5: Flags which you almost certainly won't need to change

1.6 Starter Strategies

These strategies should give you an idea of where to start and what we are expecting for your experimental design.

WARNING: It is not a good idea to exactly implement any one of these strategies. You will need to make some tweaks and adjustments to get the strategy working. This section is only meant to set expectations and give a good place to start.

1. **Aggressive Greedy Algorithm**
   Experimental scaling: Number of training iterations
   Transfer hypothesis: No scaling
   "Aggressive Greedy" practitioners say: 'this model likely won't converge to a global optimum no matter what, so let's turn it up to 11 and set our learning rate as high as possible!! High learning rates update the model faster!'

   In this strategy, you will find the learning rate that minimizes model loss after just a small number of epochs (50-100).

   Experimental overview: Start with an unreasonably high learning rate. Train the full-sized model for 10 epochs with "warm-start" (the -u flag) disabled. If training does not diverge (in other words, the loss does not start going up) double the learning rate until it diverges. Can this learning rate alpha_divergent. Then, train models for 10 epochs with learning rates that are fractions of alpha_divergent. You may want to perform several iterations to hone in on what the best scaling rate for your fractional values is.

   For the final training run, use the learning rate that performed best from these experiments.

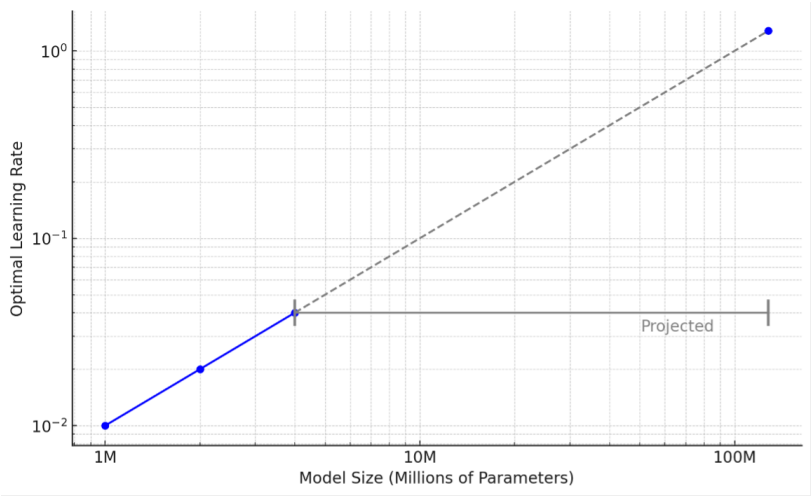
2. **Early Stopping**
   Experimental scaling: Number of training iterations
   Transfer hypothesis: No scaling
   The "early stopping" crowd doesn't believe in late bloomers. They hold the hypothesis that the model which performs best after n training epochs will continue to outperform all others when trained for 10n epochs (or 100n epochs for that matter).

   In this strategy, you will randomly sample learning rates and train for a fixed fraction of the total number of training epochs (for example, 1/10). The learning rate which performs best from this cohort will be selected for the full training run.

   Experimental overview: Use grid or random search to perturb the starter learning rate (6e-4) in a small range above and below to generate 10 trial learning rates. Then, train the full-sized model for 100M tokens. For the final 1T token training run, select the learning rate that had the lowest validation loss from the 10 candidate models.

3. **Heuristic Model-Scaling Projection**
   Experimental scaling: Model size
   Transfer hypothesis: Functional scaling
   The "Heuristic Model-Scaling Projectionists" like to build mathematical models. In this strategy, you will train several tiny models of various scales to convergence. The goal is to fit a trend line to model scale vs optimal learning rate.

   "Heuristic Model-Scaling Projectionists" aim to build a plot like this:



Experimental overview: Define models with the following hyperparameters:

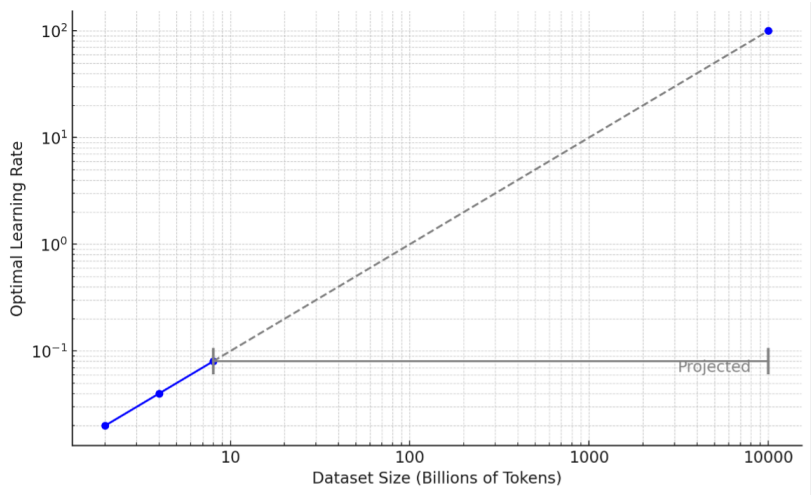
- Tiniest
  - Depth - 1
  - Channels - 64
  - Heads - 1
- Tinier
  - Depth - 2
  - Channels - 128
  - Heads - 2
- Tiny
  - Depth - 3
  - Channels - 192
  - Heads - 3

Use random search to train each model 10 times with a random hyperparameter alpha. Create a plot of model size vs. optimal learning rate, and extrapolate to the target model size of 30M paramters.

Select the learning rate projected to be optimal for a model of 30M parameters for your final training run.

4. **Heuristic Data-Scaling Projection**
   Experimental scaling: Dataset size
   Transfer hypothesis: Functional scaling
   The "Heuristic Dataset-Scaling Projectionists" mostly see eye-to-eye with the "Heuristic Model-Scaling Projections", they just disagree on what should be scaled. These folks think that model scaling dynamics are more difficult to predict than data scaling dynamics. In this strategy, you will train several full-sized models on a single pass over datasets of various scales. The goal is to fit a trend line to dataset scale vs. optimal learning rate.

   "Heuristic Dataset-Scaling Projectionists" aim to build a plot like this:



Experimental overview: In this strategy, you will work with the following provided datasets:

Define a model with the same architecture as the final 30M parameter model.

- Depth 6
- Channels 384
- Heads 6

Now, define the following datasets as fractions of the real Fineweb10B dataset (this can be achieved using the -x flag):

- ∼26M tokens (50 steps with effective batch size 524288)
- ∼52M tokens (100 steps with effective batch size 524288)
- ∼105M tokens (200 steps with effective batch size 524288)

Use random search to train the model 10 times on each dataset with a random hyperparameter alpha. Create a plot of dataset size vs. optimal learning rate, and extrapolate to a dataset of 1B tokens.

Select the learning rate projected to be optimal for a model trained on a dataset of 10B tokens (19k steps) for your final training run.

5. **Hyperband**
   Experimental scaling: Number of training iterations
   Transfer hypothesis: No scaling
   As discussed in lecture, Hyperband is a variant of early stopping that adds successive "halving" to evaluate more candidate models using the same computational budget.

   Experimental overview: find the HyperBand table below.

	S=5		S=4		S=3		S=2		S=1		S=0	
i	n_i	r_i	n_i	r_i	n_i	r_i	n_i	r_i	n_i	r_i	n_i	r_i
0	243	2	98	6	41	20	18	61	9	185	6	555
1	81	6	32	20	13	61	6	185	3	555		
2	27	20	10	61	4	185	2	555				
3	9	61	3	185	1	555						
5	3	185	1	555								
6	1	555										

   WARNING: This method requires evaluating a very large number of candidate models. You will therefore likely need to write some infrastructure code which can assist you to repeatedly schedule and save models.

   Reference: Hyperband: A Novel Bandit-Based Approach

6. **Hybrid**

   In this method, you will create a novel approach by mixing-and-matching several of the above approaches, scaling methods, and transfer methods. For example, you might combine HyperBand with Heuristic Model-Scaling Projection.

7. **To the Literature!**

   The "to the literature!" crowd knows that it is wise to stand on the shoulders of giants. They won't consider attempting a problem until they at least understand the state of the art.

   In this strategy, you will first perform a cursory literature review to identify the state-of-the-art method for single-shot hyperparameter tuning. This strategy will then be adopted and applied to the task in this homework.

   Here are some publications to get you started:

   - Language Models are Unsupervised Multitask Learners
   - Training Compute-Optimal Large Language Models
   - Tuning Large Neural Networks via Zero-Shot Hyperparameter Transfer

8. **None of the above**

   You are encouraged to apply your ingenuity and creativity to this problem and develop your own strategy for "zero-shot" hyperparameter tuning! We ask only that you do not exceed the outlined budget for your hyperparameter tuning strategy (in other words, please do not brute force this problem on a cluster if you have access to one).

1.7 FAQs

Q. Do really need to implement ALL of these experimental strategies?? That is too much work!
A. No. Please pick just ONE strategy. We gave a recommendation for a wide array of strategies because this topic is still an open research question and we hope that we can all learn together as a class which method works best.

Q. I don't really care for open-ended assignments! Can you please just tell me which strategy to use?
A. Sounds good! Select from strategies 1-4 (at random if necessary). These first four strategies are the most straightforward and are all fine choices.

Q. I don't really like structure! Can you please just let me do my own thing?
A. Sure! Please follow strategy "None of the above" which allows you the freedom to do as you please. We ask only that you make an honest effort at designing and implementing a strategy to find an optimal hyperparameter in a zero-shot fashion.

Q. I like a little but of structure, but please don't just throw me into the deep end! What should I do then?
A. Read over all of the strategies and see if any of them seem suitable to you. Pick whichever seems best to you, fill in the missing details needed and then (optionally) make some modifications to make a hybrid strategy or to improve the strategy.

1.8 Deliverables

Question 1: Running an Experiment

In this question, you will be running an example experiment and analyzing the results. The goal of this is to familiarize yourself with the workflow, and maybe give you some ideas for how you will implement your own experimental design.

You will train a GPT-2 model that has 1 transformer layer, 64 channels, and 1 attention head for 1000 epochs, with 2 different learning rates, visualize the results, and analyze what you find.

To run the experiment, you will type this into the command line:

```
./train_gpt2cu \
    -o "question_1_experiment_one"  \
    -1d 1 \
    -1c 64 \
    -1h 1 \
    -l  0.01 \
    -x 20
```

Before we run the full experiment, we want to estimate how long the experiment will take and how much it will cost. We will therefore train it for just 20 steps and see how long it takes per step!

(a) *[1 points]* Run this experiment for 20 steps. The first (0th) step is an anomaly and is much slower due to memory loading. We will get a worst-case estimate from steps 10-19, just to be safe. Write down which step in the range step 10 to step 19 took the longest, and how long it took (in milliseconds).

(b) *[1 points]* Use this time-per-step value to estimate how long it will take to run the entire 1000 steps.

(c) *[1 points]* Now estimate the cost of this experiment. Using the formula below, obtain the estimated total runtime of the experiment in hours. Multiply this value by the cost of running an AWS g5.xlarge ($1.1066/hour at time of writing, but please round to $1.50/hour to be safe)

$$\texttt{Runtime (hours)} = \frac{\texttt{time per step (ms)} * \texttt{number of steps}}{3.6 \times 10^6 \texttt{ ms per hour}}$$

Now, run the experiment for the full 1000 steps using the command below. Note that the `-x` flag (which specifies the total number of steps we train for) has been changed. Once it has finished (set a timer using your prediction!), you will need to upload the results to your S3 bucket that you created during the AWS setup for this homework.

```
./train_gpt2cu \
    -o "question_1_experiment_one"  \
    -1d 1 \
    -1c 64 \
    -1h 1 \
    -l  0.01 \
    -x 1000
```

To upload the results to the bucket, run the following in your command line, changing the bucket name to the one you created.

```
python upload_to_s3.py "your_s3_bucket_name" \
                       "question_1_experiment_one" \
                       "question_1_experiments"
```

The first argument is your bucket name. The second argument is the local path to the output files, which doubles as the experiment name (note that this is exactly the same as what we passed into the `-o` flag when running the experiment). The third argument specifies the 'run cluster'. Think of it as naming the collection of experiments to which this specific experiment belongs.

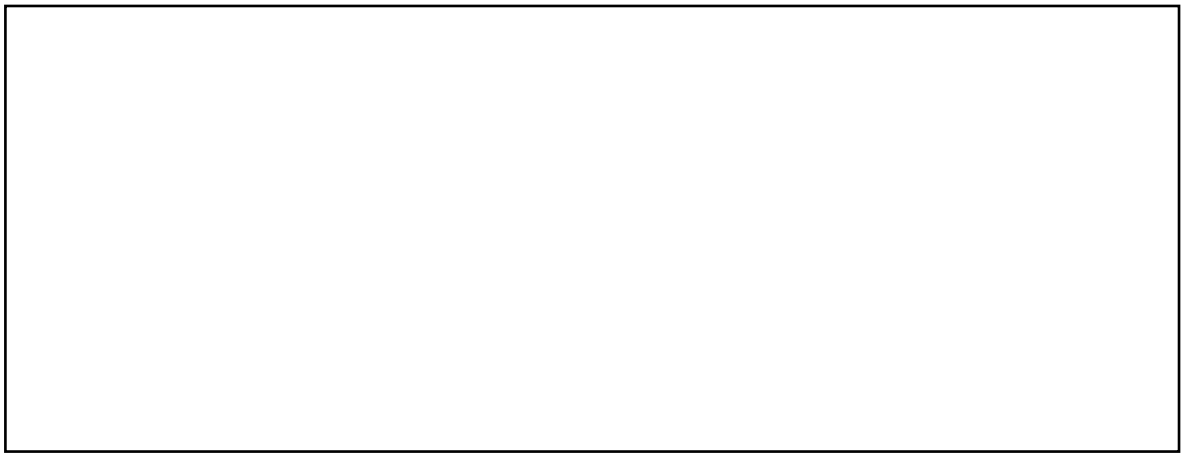
Once you have completed this. Do another run for 1000 steps, but this time change the learning rate using the `-l` flag to 0.02 from 0.01, and also change the `-o` flag to `"question_2_experiment_two"`. Once that has finished, upload those results to your S3 bucket by typing the following into your command line. Notice that we changed the second argument accordingly.

```
python upload_to_s3.py "your_s3_bucket_name" \
                       "question_1_experiment_two" \
                       "question_1_experiments"
```

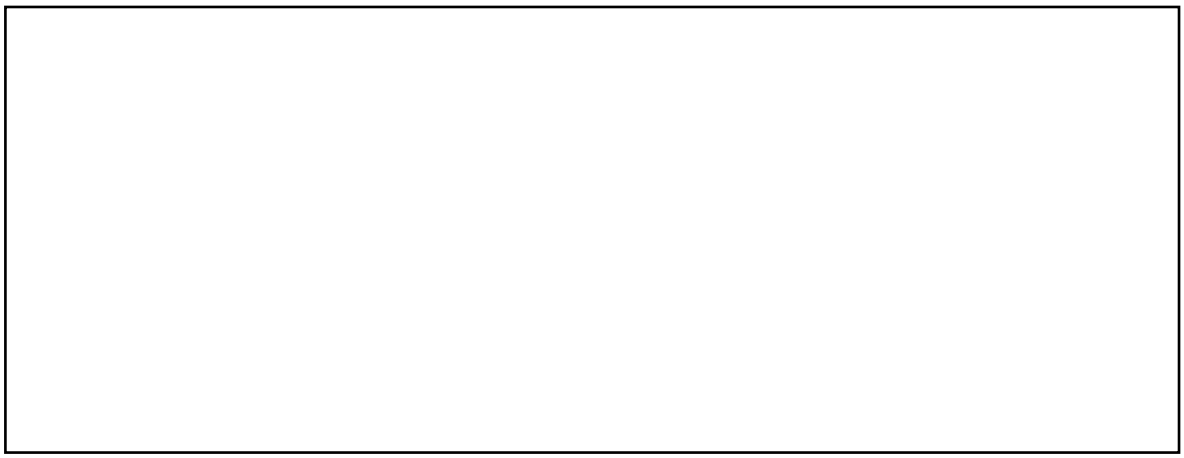
After having uploaded both experiments to your S3 bucket, you can now shut down the instance by running `sudo shutdown -h now`. This will close the instance and prevent you from being charged (Note that you will still be charged for the volume, but this amount is negligible compared to the instance.)

Now that you have uploaded your logs to the S3 bucket, it's time to visualize and interpret your results! Download and open the `vislog.ipynb` notebook, and fill in your AWS access key and AWS secret key that you got when creating the IAM user. Also fill in your AWS bucket. Then, follow the instructions in the notebook to make to make the plots and complete the following questions.

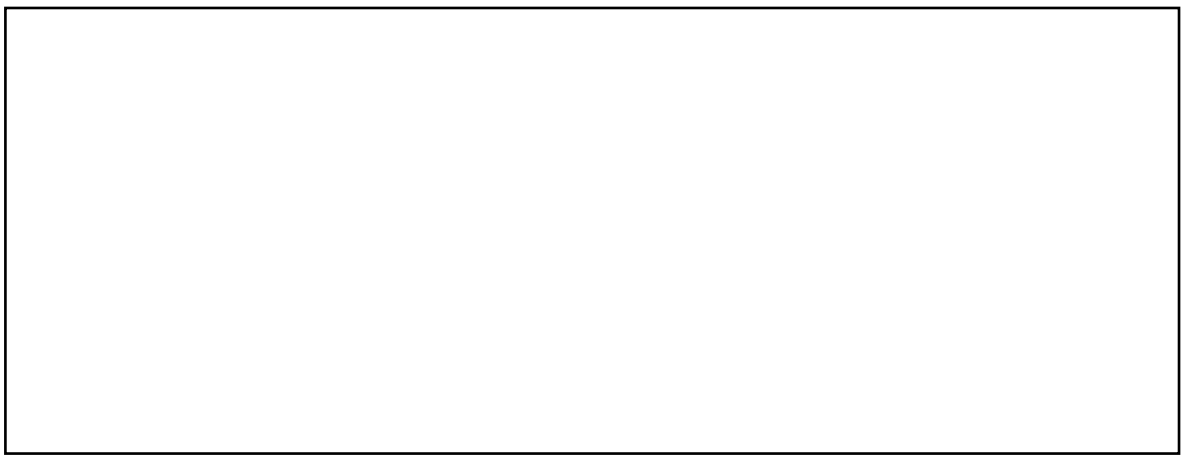
(d) *[1 points]* Make one plot that contains the training and validation loss for the first experiment you ran.



(e) *[1 points]* Make one plot that contains the training and validation loss for the second experiment you ran.



(f) *[1 points]* Make one plot that contains the training loss for both experiments, using the learning rate as the label.



(g) *[1 points]* Make one plot that contains the validation loss for both experiments, using the learning rate as the label.

(h) *[1 points]* List how many parameters the model has, and the minimum training and validation loss for both learning rates.

(i) *[1 points]* Based on the results of these experiments, what other learning rates would you test if you had to find the optimal learning rate for this specific model and hyperparameters? Would you try larger or smaller learning rates?

Congratulations, you just completed an experiment! If you don't plan to continue working on your own experiments right now, please shut down your instance by running `sudo shutdown -h now` so you don't get charged!

Note: We recommend writing bash or shell scripts to automatically run experiments sequentially. This is not required to complete this homework, but it makes running experiments much more cost efficient and easier.

Question 2: Experimental overview

[5 points] Now, design your own strategy for solving the problem of zero-shot hyperparameter tuning. Read over the starter strategies in section 1.6. Fill in missing details and make modifications as needed. Describe in words your experimental overview. What strategies will you test? How will you evaluate if your methodology is working? What (if any) kind of iteration will you perform to adjust your strategy if necessary?

Question 3: Prediction

[5 points] Next plan your own experiments (10 mandatory experiments, more optional) in a similar way. Perform your time and cost analysis and fill in the hyperparameter tuning table below. What outcome do you expect to see from your experiments? How will you know if your experimental design is successful? How will you know if you need to revise your strategy and iterate again?

Question 4: Fill in your Experimental Table

[15 points points] Now that you have planned your experiments, please proceed to run your experiments and complete filling in the first 10 rows of the table below. 10 rows are required, but you may use the remainder as needed. Make sure not to mix up your items. Please be sure to fill in the validation loss and HellaSwag accuracy.

Question 5: Reflection

[2 points] Was your experimental design successful? Did your planned experiments result in the information you were looking for? (For example, if you performed random search, were the bounds of the random values selected appropriately?)

Question 6: Prediction

[2 points] Suppose we now doubled your budget. After completing your experiment, how would you modify your strategy to be more successful in the next round of testing?

Question 7: Select your Final Hyperparameter Value

(a) *[2.5 points]* Please report your total expenditure in dollars on this programming part 1 assignment.

(b) *[2.5 points]* On the basis of your experiments above, please select the learning rate you plan to use for the final full training run.

(c) *[1 points]* If you elected not to select the learning rate with the best performance in your experiments, please briefly motivate your selection below. For example, explain your transfer hypothesis.

Experiment number	Max learning rate	Dataset	Training epochs	Depth	Channels	Heads	Ramp up? (Y/N)	Learning rate decay target	Run time	Approx. cost	Final val loss	Final HellaSwag accuracy
1												
2												
3												
4												
5												
6												
7												
8												
9												
10												
11 (optional)												
12 (optional)												
13 (optional)												
14 (optional)												
15 (optional)												
16 (optional)												
17 (optional)												
18 (optional)												
19 (optional)												
20 (optional)												
21 (optional)												
22 (optional)												
23 (optional)												
24 (optional)												
25 (optional)												
26 (optional)												
27 (optional)												
28 (optional)												
29 (optional)												
30 (optional)												

2 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.).