EECS 182 Deep Neural Networks
Fall 2023 Anant Sahai

Homework 10

This homework is due on November 11, at 10:59PM.

1. Read a Research Paper: FaceNet

In class, you have learnt how self-supervised learning can be used to learn useful representations from large datasets without labels (e.g., learning features from ImageNet). While these features may inherently pick out some notion of "similarity" between different images in the dataset, they are not incentivized to *cluster* different data points based on any interesting similarity measure.

The paper "FaceNet: A Unified Embedding for Face Recognition and Clustering" explores how we can view task of face recognition through the lens of self-supervised (or to be more accurate, *slightly* supervised) learning.

Personally I found this blog post to help a lot more than reading the paper. The two figures are particularly helpful.

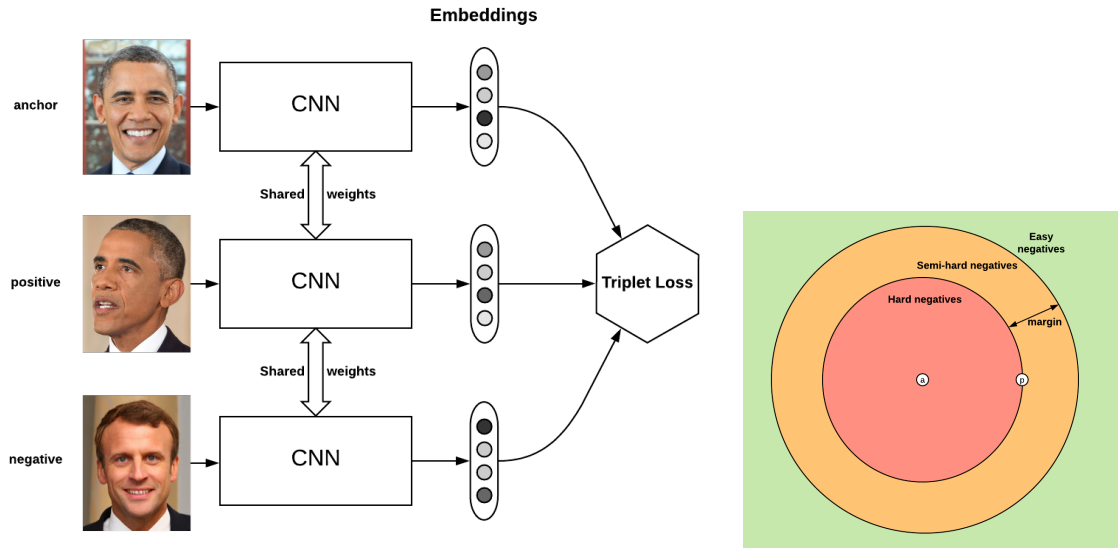


Figure 1: Figures from blog post.

Read the paper and answer the questions below.

(a) **What are the two neural network architectures considered by the authors?**

(b) **Briefly describe the *triplet loss* and how it differs from a typical supervised learning objective.**

(c) **What is the challenge with generating all possible triplets? Briefly describe how the authors address this challenge.**

(d) **How many parameters and floating point operations (FLOPs) do the authors use for their neural network? How does this compare to a ResNet-50?** Read Table 1 of the original paper[1] on ResNet to find out the FLOPs of ResNet-50. Its parameter count can be found by searching online.

[1] He, Kaiming, et al. "Deep residual learning for image recognition." (2016).

(e) **What do the authors mean by *semi-hard* negatives?**

(f) **What are harmonic embeddings?**

(g) **How does the performance vary with embedding dimensionality?**

(h) **How does the performance vary with increasing amounts of training data?**

(i) Briefly share your favorite *emergent* property/result of the learned behavior with a triplet loss from the paper.

(j) **Which approach taken by the authors interested you the most? Why?** ($\approx$ 100 words)

2. Coding Question: Visualizing Attention

Please run the cells in the Visualizing_BERT.ipynb notebook, then answer the questions below.

(a) Attention in GPT: Run part a of the notebook and generate the corresponding visualizations

i. **What similarities and differences do you notice in the visualizations between the examples in this part?** Explore the queries, keys, and values to identify any interesting patterns associated with the attention mechanism.

ii. **How does attention differ between the different layers of the GPT model? Do you notice that the tokens are attending to different tokens as we go through the layers of the network?**

(b) BERT pays attention: Run part b of the notebook and generate the corresponding visualizations.

i. Look at different layers of the BERT model in the visualizations of part (b) and identify different patterns associated with the attention mechanism. Explore the queries, keys, and values to further inform your answer. **For instance, do you notice that any particular type of tokens are attended to at a given timestep?**

ii. **Do you spot any differences between how attention works in GPT vs. BERT? Think about how the model architectures are different.**

iii. For the example with syntactically similar but definitionally different sentences, look through the different layers of the two BERT networks associated with sentence a and sentence b, and take a look at the queries, keys, and values associated with the different tokens. **Do you notice any differences in the embeddings learned for the two sentences that are essentially identical in structure but different in meaning?**

iv. **For the pre-training related examples, do you notice BERT's bi-directionality in play? Do you think pre-training the BERT helped it learn better representations?**

(c) BERT has multiple heads!: Run part c of the notebook and generate the corresponding visualizations.

i. **Do you notice different features being learned throughout the different attention heads of BERT? Why do you think this might be?**

ii. **Can you identify any of the different features that the different attention heads are focusing on?**

(d) Visualizing untrained attention weights

i. **What differences do you notice in the attention patterns between the randomly initialized and trained BERT models?**

ii. **What are some words or tokens that you would expect strong attention between? What might you guess about the gradients of this attention head for those words?**

(e) **Were you able to identify interesting patterns in the visualizations?** If yes, please share some examples (describe in text or paste a screenshot). If not, feel free to use this space for your frustrations.

3. Vision Transformer

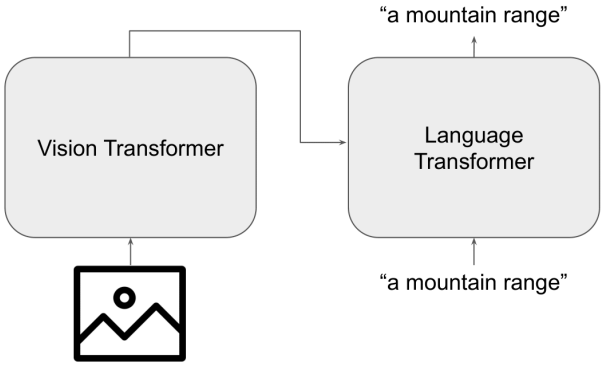


Figure 2: Image captioning model

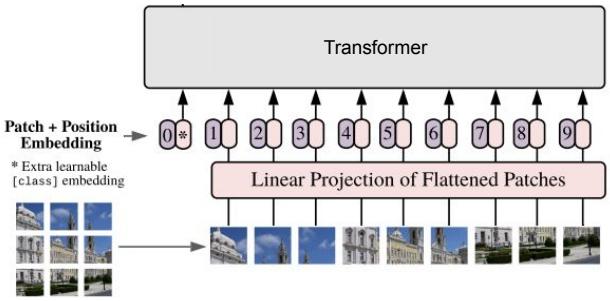


Figure 3: Vision Transformer

You are building a model to perform image captioning. As shown in Figure 2, the model consists of a vision transformer which takes in images and a language transformer which outputs captions. The language transformer will use cross-attention to access the representation of the image.

(a) For each transformer, state whether it is more appropriate to use a transformer encoder (a transformer with no masking except to handle padding) or decoder (a transformer with autoregressive self-attention masking) and why.

Vision transformer?

◯ Encoder-style transformer

◯ Decoder-style transformer

Reason:

Language transformer?

◯ Encoder-style transformer

◯ Decoder-style transformer

Reason:

(b) A standard language transformer for captioning problems alternates between layers with cross-attention between visual and language features and layers with self-attention among language features. Let's say we modify the language transformer to have a single layer which performs both attention operations at once. The grid below shows the attention mask for this operation. (For now, assume the vision transformer only outputs 3 image tokens called <ENC1>, <ENC2>, and <ENC3>. <SOS> is the start token, and <PAD> is a padding token.)

(i) One axis on this grid represents sequence embeddings used to make the queries, and the other axis represents sequence embeddings used to make the keys. **Which is which?**

- ◯ Each column creates a query, each row creates a key and a value
- ◯ Each column creates a key and a value, each row creates a query
- ◯ Each column creates a query and a value, each row creates a key
- ◯ Each column creates a key, each row creates a query and a value

(ii) **Mark X in some of the blank cells in the grid to illustrate the attention masks.** (A X marked cell is masked out, a blank cell is not.)

	<SOS>	a	mountain	range	<PAD>
<SOS>					
a					
mountain					
range					
<PAD>					
<ENC1>					
<ENC2>					
<ENC3>					

(c) As shown in Figure 3, a vision transformer encodes images by splitting them into patches. Patches get separately encoded into vectors, flattened into a sequence, and then passed into the transformer. What is the Big-O the time complexity of the vision transformer attention operation? Assume you have an image of size $H \times W$ and patches of size $P \times P$. Only consider the time of the attention operation, not the time to produce queries, keys, and values. Queries, keys, and values are all numerical vectors of dimension D.

(d) Some recent papers have reduced the complexity of vision transformer attention by segmenting an image into windows, as shown in Figure 4.

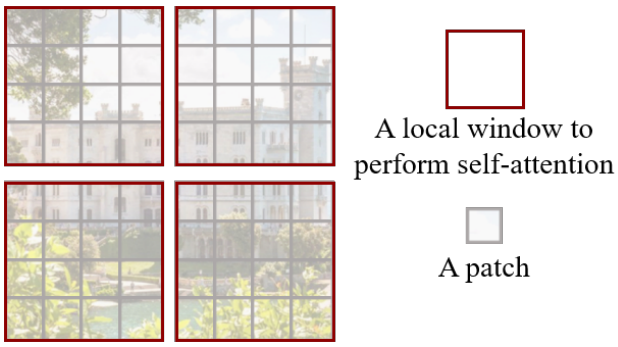


Figure 4: Vision transformer attention with windows

Patches only attend to other patches within the same window. **hat is the big-O runtime complexity of the attention operation after this modification?** Assume each window consists of K by K patches.

4. Coding: Masked Auto-Encoding

Please follow the instructions in this notebook. You will implement Vision Transformer and Masked Autoencoder in PyTorch.

If everything is implemented correctly, running the whole notebook once takes about 1 hour in total.

5. Coding Question: Summarization (Part I)

Please follow the instructions in this notebook. You will implement a Transformer using fundamental building blocks in PyTorch. You'll apply the Transformer encoder-decoder model to a sequence-to-sequence NLP task: document summarization. Refer to the Attention is All You Need paper for details on the model architecture. Once you finished with the notebook,

- Answer the following questions in your submission of the written assignment:

(a) **Please submit the screenshots of the training loss and the validation loss displayed on Tensorboard.**

6. Fermi Estimation for Large-scale Deep Learning Models

Fermi estimation is a technique for estimating quantities through rough approximations and educated guesses. Named after physicist Enrico Fermi, this method involves breaking down a problem into simpler, more manageable parts, making reasonable assumptions, and using simple arithmetic to arrive at a good enough answer.

In this question, you will be walked through a simple example of Fermi estimation of a deep learning model. Specifically, we will try to estimate the design parameters of a hypothetical GPT-5 with 1 trillion parameters.

This question is perhaps a bit reading-heavy, but the calculations are very simple, and I hope you find the lesson interesting.

(a) (**GPT-like AGI**) This is **not a question**, but some reading material to get you into the mood for Fermi estimation. We estimate the number of parameters necessary for achieving human-level Artificial General Intelligence, under two assumptions: that a GPT-like architecture is enough; that it requires only matching the human brain in the "numbers".

Let's estimate how many layers the hypothetical GPT model should have. When prompted with a question, the first answer comes to mind in about a second, and it is generally a good one, if not the best. Perhaps slow deliberation is nothing but stringing together a long chain of snap judgments, guided by snap judgments about snap judgments.

The characteristic time-scale of a brain is 0.01 seconds – the fastest brain wave, gamma wave, is 100 Hz. This indicates that the brain takes on the order of 100 steps to make a snap decision. This is the "hundred-step-rule" of Jerome Feldman.[2]

This corresponds very well with the largest model of GPT-3, which has 96 layers.

How many parameters would such a model require? The brain has 10^{15} synapses. It's unclear how precise each synapse is, but one estimate states that the hippocampal synapse has a precision of about 5 bits[3], which can be stored within a 16-bit floating point number, with room to spare.

Assuming that, we expect an AGI GPT to have 10^{15} (1000 trillion) parameters. This homework question does not scale all the way up to 1000 trillion parameters, but only up to 1 trillion parameters. You can do the same calculations for the 1000 trillion parameter model, however.

[2] Feldman, Jerome A., and Dana H. Ballard. "Connectionist models and their properties." Cognitive science 6.3 (1982): 205-254.
[3] Bartol Jr, Thomas M., et al. "Nanoconnectomic upper bound on the variability of synaptic plasticity." elife 4 (2015): e10778.

(b) (**Chinchilla Scaling Law**) The paper "Training Compute-Optimal Large Language Models" (2022) reported a series of training runs on language models, trained by Google DeepMind researchers. Each training run is characterized by four numbers:

- L: the final loss (negative log-likelihood per token) achieved by the trained model.
- N: the number of parameters in the model.
- D: training dataset size, measured in tokens.
- C: training compute cost, measured in FLOP.

After training a few hundred models, they obtained a large dataset of (L, N, D, C), and they fitted a statistical law of the form

$$L = \frac{A}{N^\alpha} + \frac{B}{D^\beta} + L_0,$$

where the parameters are

$$\alpha = 0.34, \beta = 0.28, A = 406.4, B = 410.7, L_0 = 1.69.$$

They also estimated that the cost of training compute C is proportional to ND. This is understandable, because each token must flow through the entire model and "hit" each parameter once, incurring a fixed number of floating point operations. They estimated that it takes 6 FLOPs per parameter per token. That is,

$$C = C_0 ND, \quad C_0 = 6$$

Given the assumptions, for each fixed computing budget C, we can solve for the optimal D and N, which is usually referred to as "Chinchilla optimal" training:

$$\begin{cases} \min_{N,D} L = \frac{A}{N^\alpha} + \frac{B}{D^\beta} + L_0 \\ \text{such that } C_0 ND = C \end{cases}$$

Solve the above equations symbolically to find N_{opt}, D_{opt} as a function of $C, C_0, \alpha, \beta, A, B$. Then, plug in the numerical values of the parameters, to find a numerical expression for N_{opt}, D_{opt} as a function of C.

(c) In the same paper, they also performed a *direct* statistical fitting, to find the optimal N, D for a given C, without going through the intermediate steps above. This gives a slightly different result (only slightly different – as you would know after solving the previous problem):

$$N_{opt}(C) = 0.1C^{0.5}; \quad D_{opt}(C) = 1.7C^{0.5}.$$

For the rest of the question, we will use **these equations** as the Chinchilla scaling laws. **Do not** use the equations you derived for the previous problem.

Suppose we decide that our next AI should have 1 trillion ($N = 10^{12}$) parameters, and we use Chinchilla scaling laws. **How much compute would it cost to train, and how many tokens would its training dataset have?**

(d) (**Dataset size**) Assuming each English word cost about 1.4 tokens, **how many English words would 10 trillion tokens be?** Assuming each page has 400 words, and each book has 300 pages, **how many books is that?** To put it into context, look up the size of Library of Congress, and Google Books, and **compare with the number we just calculated.**

(e) (**Memory requirement**) Typically, a deep learning model has 16-bit floating point parameters. Modern systems sometimes use lower precision (e.g. 8-bit) floating point numbers to save space, but generally it is necessary to use at least 16-bit during training, and the model is converted to lower precision after training ("post-training quantization").

Given that each parameter is a 16-bit floating point number, how much memory does it cost to store a model with 1 billion parameters? How about our hypothetical GPT-5, which has 1 trillion parameters? How many A100 GPU (VRAM = 40 GB) would be required to contain the full GPT-5 model?

(f) (**Memory cost**)

This table[4] gives the price per megabyte of different storage technology, in price per megabyte (2010 dollars), up to 2018.

Year	Memory (DRAM)	Flash/SSD	Hard disk
~1955	$411,000,000		$6,230
1970	$734,000.00		$260.00
1990	$148.20		$5.45
2003	$0.09	$0.305	$0.00132
2010	$0.019	$0.00244	$0.000073
2018	$0.0059	$0.00015	$0.000020

The same costs *relative* to the cost of a hard disk in ~2018:

Year	Memory	Flash/SSD	Hard disk
~1955	20,500,000,000,000		312,000,000
1970	36,700,000,000		13,000,000
1990	7,400,000		270,000
2003	4,100	15,200	6.6
2010	950	122	3.6
2018	295	7.5	1

Suppose long-term memory storage can last 1 year before being replaced. How much money does it cost per year to store a 1 trillion parameter model on an SSD, the most expensive form of long-term storage? How much money does it cost to store a 1 trillion parameter model on DRAM memory? Use 2018 prices.

(g) (**Memory bandwidth and latency**) While the memory itself is cheap, moving the data requires expensive high-bandwidth wiring. Indeed, the memory bandwidth between the DRAM (or "VRAM" for "Video RAM") and the little processors on the GPU is a main bottleneck on how good the GPU can perform.

During a single forward pass of a model, the parameters of the model are loaded from the DRAM of the GPU into the fast cache memories, then pushed through the thousands of computing processors on the GPU.

A100 GPU has a memory bandwidth of 1.6 TB/s.

[4] Source: Storage 2: Cache model – CS 61 2018.

What is the minimal latency, in seconds, for the GPU to perform a single forward pass through our hypothetical GPT-5 model with 1 trillion parameters? How many tokens can it output (autoregressively) in one minute? How about GPT-3 (175 billion parameters)?

Note: Since we are just trying to compute an order-of-magnitude estimate, let's assume for the problem that the model fits onto a single DRAM on a single GPU. You can also ignore the need to read/write model activations and optimizer states.

There are some ways to improve latency. For example, the model can be parallelized over several GPUs, which effectively increases the memory bandwidth. For example, "tensor parallelism" splits each layer into several GPUs.

There is also "pipeline parallelism", which splits the model into layers. The first few layers go into one GPU, the next few go into another, and so on. This does NOT decrease latency, but it does increase throughput.

The fundamental bottleneck in an autoregressive model like GPT is that, by design, they have to start a sentence without knowing how it will end. That is, they have to generate the first token before generating the next one, and so on. This can never be parallelized (except by egrageous hacks like speculative decoding).

One reason Transformers dominated over RNN is that training and inferring an RNN *both* must be done one-token-at-a-time. For Transformers, training can be done in parallel over the entire string. Inferring however still cannot be parallelized.

Parallization tends to be a deeply troublesome business – parallel programming is generally deeply troublesome.[5] However, there is a very simple method to improve thoroughput: batch mode. For example, GPT-5 might be able to run 1 million conversations in parallel, outputting one token for all conversations per forward pass. This trick works until the batch size is so large that the activations due to tokens takes up about as much memory bandwidth as the model parameters.

Concretely, we can estimate what is a good batch size for GPT-3 175B. It has 96 attention layers, each with 96x 128-dimension heads. It is typically run with 16-bit precision floating point numebrs. **For a single token, how many megabytes would all the floating point activations cost?**

The model itself has 175 billion 16-bit floating point parameters, taking up about 350 GB. **How many tokens do we need to put into a batch, before the activations occupy the same amount of memory as the model parameters?**

(h) (**Training cost**) How much money does compute cost? We can work through an example using the current standard computing hardware: Nvidia A100 GPU (40 GB VRAM version).

The most important specifications are:

- Unit price: 15000 USD.
- Rental price: 2 USD/hr.
- Speed: 0.3 petaFLOP/s = 3e14 FLOP/s.
- Power: 0.3 kiloWatt.
- Memory bandwidth: 1600 GB/s.

In the literature about the largest AI models, the training cost is often reported in units of "petaFLOP-day", which is equal to 1 petaFLOP/second x 1 day. **How many FLOP is 1 petaFLOP-day? What is the equivalent number of A100-hour? If we were to buy 1 petaFLOP-day of compute with rented A100 GPU, how much would it cost?**

[5] I had to take CS 267 last semester. Never again...

The largest model of GPT-3 cost 3640 petaFLOP-days to train (according to Table D.1 of the report). How much would it cost if it were trained with A100? How much money does it cost to train our hypothetical GPT-5?

In reality, the GPU cannot be worked to their full speed, and we only use about 30% of its theoretical peak FLOP/sec (so for example, for A100, we only get 0.1 petaFLOP/s, instead of 0.3 petaFLOP/s).[6] For this question, we assume that the utilization rate is 100%.

Also, we are assuming the training happens in one go, without hardware failures, divergences, and other issues requiring restart at a checkpoint. There is not much published data from large-scale training, but the OPT-175B training run (described later) took 3 months to complete, but would have taken only 33 days if there were no need for the many restarts. This suggests that the restarts would increase the computing cost by 2x to 3x.

For context, here are the costs of *development* of various items[7]:

- iPhone 1: 150 million USD.
- A typical 5 nm chip: 0.5 billion USD.
- Airbus A380: 18 billion USD.[8]
- Three Gorges Dam: 250 billion CNY, or about 30 billion USD.
- Manhattan Project: 24 billion USD (2021 level)
- Apollo Program: 178 billion USD (2022 level)

Comment on the cost of our hypothetical GPT-5. Is it on the order of a large commercial actor like Google, or a state actor like China?

Microsoft announces new supercomputer, lays out vision for future AI work (2020):

> The supercomputer developed for OpenAI is a single system with more than 285,000 CPU cores, 10,000 GPUs and 400 gigabits per second of network connectivity for each GPU server. Compared with other machines listed on the TOP500 supercomputers in the world, it ranks in the top five, Microsoft says.

The largest companies, like Microsoft have GPU on the order of 10000 A100. What is the wall clock hour of training GPT-5, assuming you have 10000 A100, perfect utilization, and no interruptions?

(i) (**the difficulty of large-scale training**) This is **not a problem**, but some interesting reading of some "stories from the trenches".

Large models do end up diverging many times during training runs, requiring restarts. Sometimes the divergence is so bad that the whole training run must be started from scratch. Sometimes the convergence is too slow, requiring fixes to the optimizer and learning rate schedules, etc.

All together, we should expect the failures, restarts, deadends… to triple the cost at least, to ~1 billion USD.

In 2021, a team of 5 engineers from Meta trained a LLM with 175 billion parameters, in 3 months, using 1024 80GB A100 GPUs from. Excluding all the divergences, hardware failures, and other issues that caused lost progress, the final model would have taken about 33 days of continuous training.

They have kept journals during their training. This is now published at metaseq/projects/OPT/chronicles at main · facebookresearch/metaseq · GitHub. You can see how difficult it is to train a large model. Selected quotes:

> These notes cover ~90 restarts over the course of training the lineage of this current model (experiments 12.xx).

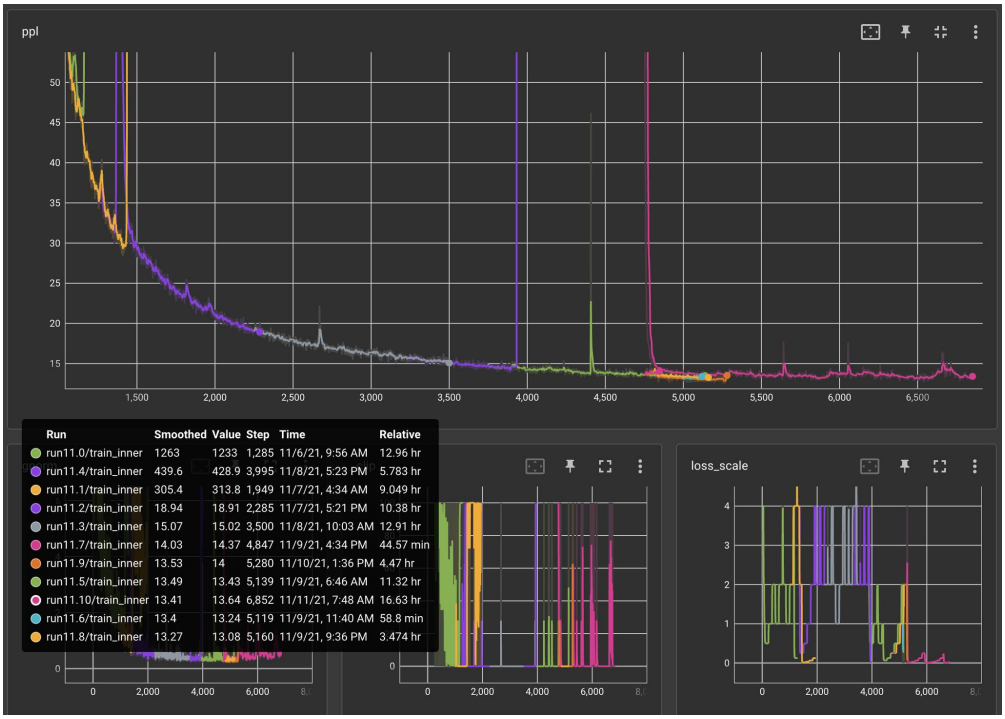
[6] The utilization rate of 30% is according to EpochAI.

[7] Sorry, not adjusted for inflation to the same year, but they are roughly in the range of 2000 – 2020 USD.

[8] Table 4.3 of Bowen, John T. The economic geography of air transportation: space, time, and the freedom of the sky. Routledge, 2010.

Found issues with the new dataset where perplexity was unreasonably low… After applying as much regex-ing as we could to salvage the dataset, we relaunched another set of experiments to test LPE (experiments 20-29) on the new dataset. We didn't have time to retrain a new BPE on the final dataset, so we fell back to using the GPT-2 BPE.

From experiment 11.4 onward, we saw grad norm explosions / loss explosions / nans after a couple hundred updates after each restart, along with extremely unstable loss scales that would drop to the point of massively underflowing. We started taking more and more drastic actions then, starting with increasing weight decay to 0.1, lowering Adam beta2 to 0.95, lowering LR again, until finally by experiment 11.10 we hot-swapped in ReLU and also switched to a more stable MHA calculation (noting that the x**3 term in GeLU might be a source of instability with FP16).



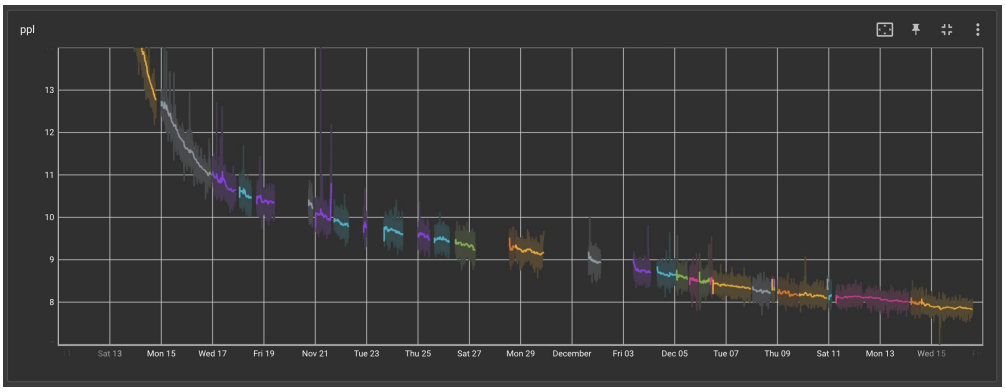
On November 11, we started training our 12.00 experiment with all of these changes, and since then, the only restarts we've had to make were all related to hardware issues (missing GPUs on instances, training randomly hanging after including a new node, ECC errors, partial checkpoint upload after hardware error, CUDA errors, NCCL errors, etc.).

Replacement through the cloud interface can take hours for a single machine, and we started finding that more often than not we would end up getting the same bad machine again.

There were also issues with blob store when downloading 1.6TB of a single model checkpoint (992 files, each ~1.7GB) on restarts, at which point the downloads themselves would start hanging nondeterministically, which then delayed training recovery even further.

We managed to hit our top three record long runs of the experiment these past two weeks, lasting 1.5, 2.8, and 2 days each! If we were to look at only the runs that have contributed to pushing training further and plot training perplexity against wall clock time, we get the following [The breaks are due to the Thanksgiving holiday]:

(j) (**Inference cost**) Inference cost a lot less money than training, but it's still a substantial cost. For Transformer language models, it costs about 2 FLOPs per parameter to infer on one token.



Given that GPT-3.5 has 175 billion parameters, how many FLOPs would it take to infer on 1 million tokens? How much money would it cost if it were run with A100?

The price offered by OpenAI is 2 USD per 1 million tokens. Assuming that GPT-3.5 cost 10 million USD to develop and train, **how many tokens must be sold, just to recoup the cost?** Assuming each English word cost about 1.4 tokens, and each essay is 1000 words, **how many essays would it be equivalent to?**

(k) (**Energetic cost**) The Landauer limit states that the cost of erasing one bit of information is $E = k_B T \ln 2$, where k_B is the Boltzmann constant, and T is the temperature of the computing machinery. At room temperature, $T = 300K$, giving us $E = 3 \times 10^{-21} J$.

Now, one FLOP is a floating point operation, meaning that you start with two 32-bit objects and end up with a single 32-bit object. You start with 64 bits and end up with just 32 bits, and so you lose 32 bits. So by the Landauer limit, the minimal energetic cost is $32 k_B T \ln 2$.

Given this, what is the minimal energy required for performing one FLOP? What is the minimal power (in Watts) required to perform 300 TFLOP/sec, as in A100 GPU? Compared this to the actual value of 300 Watts.

(l) (**Environmental cost**) According to "Carbon emissions and large neural network training" (Patterson et al, 2021), the carbon emission of training GPT-3 is 552 tCO2. According to a 2021 poll of climate economists, 1 tCO2 emission should cost somewhere between 50 and 250 USD. Let's take their geometric average of 112 USD.

If we add all the tCO2 cost to the training of GPT-3, how much more expensive would it be? Compare that with its A100-GPU cost of training.

To put the number in another context, compare it with some typical American food. According to Our World in Data, it cost about 50 kg of CO2 emission per 1 kg of beef.

Assuming each burger ("quarter pounder") contains 1/4 pound (113 grams) of beef. **How many burgers would be equivalent to the CO2 emission of GPT-3?**

7. Homework Process and Study Group

Citing sources and collaborators are an important part of life, including being a student!
We also want to understand what resources you find helpful and how much time homework is taking, so we can change things in the future if possible.

(a) **What sources (if any) did you use as you worked through the homework?**

(b) **If you worked with someone on this homework, who did you work with?**
List names and student ID's. (In case of homework party, you can also just describe the group.)

(c) **Roughly how many total hours did you work on this homework?**

Contributors:

- Dhruv Shah.
- Shivam Singhal.
- Kevin Li.
- Olivia Watkins.
- Bryan Wu.
- Anant Sahai.
- Yuxi Liu.
- CS182 Staff from past semesters.
- Jake Austin.
- Linyuan Gong.
- Sheng Shen.
- Hao Liu.
- Allie Gu.

