EECS 182 Deep Neural Networks
Fall 2025 Anant Sahai and Gireeja Ranade

Homework 11

This homework is due on November 18, at 10:59PM.

1. LoRA

A common strategy for adapting a large pre-trained model to a new task is to update only a subset of its parameters, keeping the rest frozen. Low-Rank Adaptation (LoRA) offers a more flexible approach to this idea. In this problem, we focus on a single weight matrix W with m rows and ℓ columns, where W_0 is the pre-trained value. During LoRA-based training, W is replaced by $W_0 + AB$, where W_0 remains frozen and only A and B are learnable. Here, A is an $m \times k$ matrix and B is a $k \times \ell$ matrix, typically with $k \ll \min(m, \ell)$.

(a) Suppose you are using LoRA to adapt a pretrained deep neural net to a new task and observe that the model is "underfitting" the training data. **What would you try to adjust in the LoRA to get better performance?**

(b) Suppose both A and B are initialized to all zeros. **Why will this cause problems for LoRA-based finetuning?**

Remember, this is going to be trained using SGD-style updates over a training set with a loss function.

(c) Consider the following pseudocode for LoRA initialization:

```
A = torch.nn.Parameter(torch.empty(m, k))
B = torch.nn.Parameter(torch.empty(k, l))
torch.nn.init.xavier_uniform_(A)
torch.nn.init.xavier_uniform_(B)
```

Why might LoRA fine-tuning not work well with this initialization?

(d) **How much memory is required to store the LoRA adaptation weights (A and B)?** Assume we are using floats (4 bytes per real number) and give your answer in bytes. **How does this compare to storing a single full-rank adaptation matrix?**

2. A Brief Introduction to Transformer Interpretability

Modern neural networks, particularly large language models like Transformers, achieve remarkable performance but are often treated as "black boxes". Mechanistic interpretability is a research field dedicated to reverse-engineering the specific circuits these models learn. Instead of just observing their input-output behavior, we aim to understand the internal mechanisms and computations that lead to their capabilities. This problem explores a mathematical framework proposed by Elhage et al. that treats Transformers as a collection of interacting "circuits" built from attention and MLP layers.

The Residual Stream: A Central Communication Channel

A core concept in this framework is the **residual stream**. At any given layer l, the residual stream, denoted $\mathbf{X}^l$, is a sequence of vectors representing the state of the computation for each token in the input. Crucially,

the Transformer, like all modern post-ResNet architectures, is based on residual connections, meaning the output of each layer is *added* to the stream from the previous layer. For a layer l containing a component (e.g., an attention head) with function $f^{(l)}$, the update rule is:

$$\mathbf{X}^{l+1} = \mathbf{X}^l + f^{(l)}(\mathbf{X}^l)$$

The final output of an L-layer model is therefore the sum of the initial embedding and the outputs of all layers:

$$\mathbf{X}^{\text{final}} = \mathbf{X}^0 + \sum_{l=0}^{L-1} f^{(l)}(\mathbf{X}^l)$$

This additive structure is fundamental. It allows us to view the residual stream as a central communication bus or channel where different components read information from and additively write their results back to. This reframing is key to analyzing the function of individual components in relative isolation.

Throughout this problem, we refer to the following variables and their shapes.

Variable	Shape	Description
$\mathbf{T}$	$\mathbb{R}^{n_{\text{vocab}} \times n_{\text{context}}}$	Matrix of one-hot encoded input tokens. Each column is a token.
$\mathbf{X}^l$	$\mathbb{R}^{d_{\text{model}} \times n_{\text{context}}}$	"Residual stream" or "embedding" vectors of model at layer l. Each column is a vector for a token.
$\mathbf{W}_E$	$\mathbb{R}^{d_{\text{model}} \times n_{\text{vocab}}}$	Token embedding matrix. Maps one-hot vectors to the residual stream.
$\mathbf{W}_U$	$\mathbb{R}^{n_{\text{vocab}} \times d_{\text{model}}}$	Token unembedding matrix. Maps final residual stream vectors to logits.
$\mathbf{W}_Q^h, \mathbf{W}_K^h, \mathbf{W}_V^h$	$\mathbb{R}^{d_{\text{head}} \times d_{\text{model}}}$	Query, Key, and Value weight matrices for attention head h.
$\mathbf{W}_O^h$	$\mathbb{R}^{d_{\text{model}} \times d_{\text{head}}}$	Output weight matrix for attention head h. This weight matrix left multiplies the output from attention head h and projects it into the dimension of the residual stream.
$\mathbf{A}^h$	$\mathbb{R}^{n_{\text{context}} \times n_{\text{context}}}$	The attention pattern matrix (post-softmax) for attention head h.
$\mathbf{W}_{OV}^h$	$\mathbb{R}^{d_{\text{model}} \times d_{\text{model}}}$	The Output-Value matrix for attention head h, defined as $\mathbf{W}_{OV}^h = \mathbf{W}_O^h \mathbf{W}_V^h$.
$\mathbf{W}_{QK}^h$	$\mathbb{R}^{d_{\text{model}} \times d_{\text{model}}}$	The Query-Key matrix for attention head h, defined as $\mathbf{W}_{QK}^h = \mathbf{W}_Q^{hT} \mathbf{W}_K^h$.

Note that the superscript h is omitted when context is clear about which attention head we are referring to.

(a) **The Simplest Transformer (A Zero-Layer Model)**

i. Given a matrix of one-hot input tokens $\mathbf{T} \in \mathbb{R}^{n_{\text{vocab}} \times n_{\text{context}}}$, write the mathematical expression for the final logits, $\mathbf{L} \in \mathbb{R}^{n_{\text{vocab}} \times n_{\text{context}}}$. Your expression should be in terms of $\mathbf{T}$, the token embedding matrix $\mathbf{W}_E$, and the unembedding matrix $\mathbf{W}_U$.

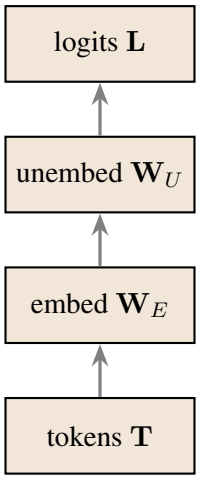


Figure 1: The Simplest Transformer: A Zero-Layer Model

ii. In simple terms, what algorithm does this zero-layer model implement? What information does the prediction for the token at position t depend on? This reveals the baseline functionality of the Transformer architecture before any contextual processing is introduced.

(b) **Multi-Head Attention: Concatenation vs. Addition**

In the original Vaswani et al. paper on transformers, a multi-head attention layer is described differently than our "independent circuits" view. There, the outputs of all heads are concatenated and then multiplied by a single large output matrix. This question asks you to prove this is equivalent to an "additive and independent" view.

Consider an attention layer with H heads. Let the value computation output for head h be $\mathbf{r}^h \in \mathbb{R}^{d_{\text{head}} \times n_{\text{context}}}$. In the "concatenation" view, these are stacked vertically to form a matrix $\mathbf{R}^H \in \mathbb{R}^{d_{\text{head}} \cdot H \times n_{\text{context}}}$ and multiplied by a single large output matrix $\mathbf{W}_O^H \in \mathbb{R}^{d_{\text{model}} \times d_{\text{head}} \cdot H}$, and the final output is $\mathbf{H} = \mathbf{W}_O^H \mathbf{R}^H$. Typically, the dimension $d_{\text{head}} \cdot H$ is equal to d_{model} where $H = d_{\text{model}}/d_{\text{head}}$. In the "additive and independent" view, each head h has its own output matrix, $\mathbf{W}_O^h \in \mathbb{R}^{d_{\text{model}} \times d_{\text{head}}}$, and the total output is $\mathbf{H} = \sum_{h=1}^{H} \mathbf{W}_O^h \mathbf{r}^h$.

(i) Show that these formulations are equivalent. Specifically, demonstrate how $\mathbf{W}_O^H$ can be constructed from the individual $\mathbf{W}_O^h$ matrices to make the two expressions for $\mathbf{H}$ identical.

(ii) What is an advantage and disadvantage of each view?

For the following parts, we consider a Transformer with a single attention layer with H heads and no normalizations as shown in Figure 2.

(c) **The QK Circuit: Determining Attention Patterns**

An attention head can conceptually be split into two independent operations: one that decides *where to look* (the QK circuit) and one that decides *what information to move* (the OV circuit). Let's first analyze the "where to look" mechanism.

The pre-softmax attention score from a "query" token at position i to a "key" token at position j is computed as:

$$\mathbf{S}_{ij} = \mathbf{Q}_i^T \mathbf{K}_j$$

where $\mathbf{Q}_i = \mathbf{W}_Q \mathbf{X}_i$ and $\mathbf{K}_j = \mathbf{W}_K \mathbf{X}_j$ are the query and key vectors.

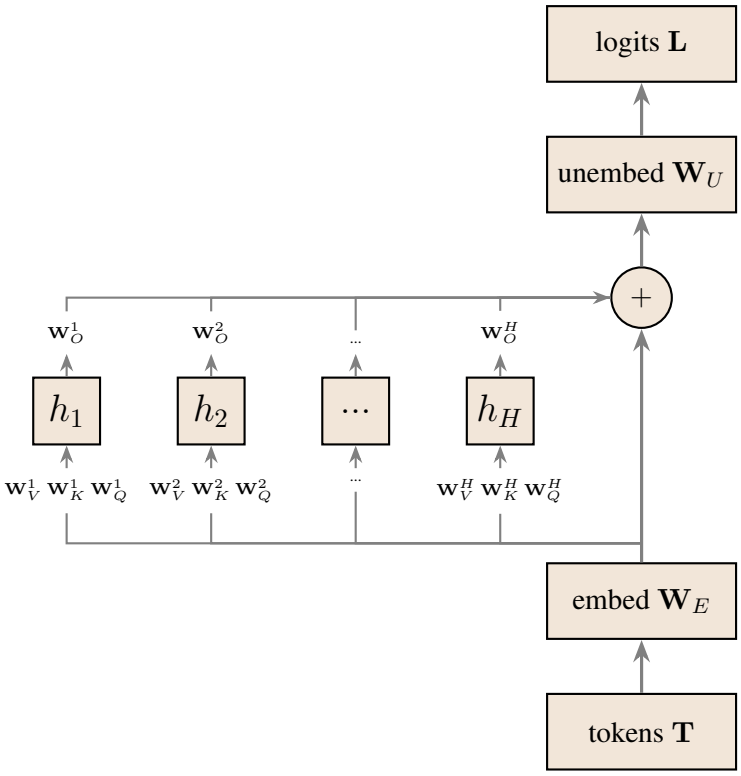


Figure 2: A Transformer With A Single Attention Layer with No Normalizations

(i) Starting from the definitions above, derive the expression for $\mathbf{S}_{ij}$ as a bilinear form involving the residual stream vectors $\mathbf{X}_i$ and $\mathbf{X}_j$, and a single "virtual" weight matrix $\mathbf{W}_{QK}$. Explicitly define $\mathbf{W}_{QK}$ in terms of $\mathbf{W}_Q$ and $\mathbf{W}_K$.

(ii) The matrix $\mathbf{W}_{QK}$ can be interpreted as defining the "question" the attention head asks to determine which tokens to attend to. Consider a toy scenario where $d_{\text{model}} = 3$.

(1) If $\mathbf{W}_{QK} = \mathbf{I} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$, what kind of relationship between $\mathbf{X}_i$ and $\mathbf{X}_j$ would lead to a high attention score $\mathbf{S}_{ij} = \mathbf{X}_i^T \mathbf{W}_{QK} \mathbf{X}_j$? Describe in words what this head "looks for."

(2) If $\mathbf{W}_{QK} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$, what kind of relationship would lead to a high attention score? What happens to information in the second and third dimensions?

(3) If $\mathbf{W}_{QK} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix}$, what pattern does this head look for? How does this differ from the identity case?

(4) If $\mathbf{W}_{QK} = -\mathbf{I} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix}$, what pattern does this head look for? What is the effect of this on the attention pattern after softmax?

(d) **The OV Circuit: Reading and Writing Information**

Now let's analyze the information-moving part of the attention head. Once the attention pattern $\mathbf{A}^h$ is determined (by the QK circuit), the head reads information from attended tokens and writes it to the destination. This is governed by the OV circuit. Note that the output of head h is:

$$h(\mathbf{X}^0) = \mathbf{W}_O^h \mathbf{W}_V^h \mathbf{X}^0 \mathbf{A}^{hT} = \mathbf{W}_{OV}^h \mathbf{X}^0 \mathbf{A}^{hT}$$

where $\mathbf{A}^h$ is the attention pattern (post-softmax) determined by the QK circuit.

(i) **[Understanding the Attention-Weighted Average]**
Before proceeding, let's clarify what $\mathbf{X}^0\mathbf{A}^T$ computes. Recall that:

- $\mathbf{X}^0 \in \mathbb{R}^{d_{\text{model}} \times n_{\text{context}}}$ where the j-th column $\mathbf{X}_j^0$ is the residual stream vector for the token at position j.
- $\mathbf{A} \in \mathbb{R}^{n_{\text{context}} \times n_{\text{context}}}$ is the attention pattern where $\mathbf{A}_{ij}$ is the attention weight from destination position i to source position j.
- Therefore, $\mathbf{A}^T \in \mathbb{R}^{n_{\text{context}} \times n_{\text{context}}}$ where $(\mathbf{A}^T)_{ji} = \mathbf{A}_{ij}$.

Show that the i-th column of $\mathbf{X}^0\mathbf{A}^T$ is an attention-weighted average of the source token vectors. Specifically, show:

$$(\mathbf{X}^0\mathbf{A}^T)_i = \sum_{j=1}^{n_{\text{context}}} \mathbf{A}_{ij}\mathbf{X}_j^0$$

(ii) Write the full expression for the final residual stream, $\mathbf{X}^{\text{final}}$, in terms of the initial stream $\mathbf{X}^0$ and the heads' weight matrices.

(iii) The update vector added to the residual stream at destination position t by head i is the t-th column of the head's output, denoted $h_i(\mathbf{X}^0)_t$. Prove that this update vector must lie within the column space of $\mathbf{W}_{OV}^i$. That is, show that $h_i(\mathbf{X}^0)_t \in \text{Col}(\mathbf{W}_{OV}^i)$. This demonstrates that the head can only write to a low-dimensional subspace of the d_{model}-dimensional residual stream.

(e) **Formalizing Read and Write Subspaces via SVD**

Important: Separating the QK and OV Circuits
In this analysis, we focus exclusively on the **OV circuit** $\mathbf{W}_{OV}$, which characterizes *what* information is moved and *how* it is transformed.
Recall from part (d) that the full output of an attention head is:

$$h(\mathbf{X}^0) = \mathbf{W}_{OV}\mathbf{X}^0\mathbf{A}^T$$

This can be decomposed into two independent operations:

- **QK Circuit** (analyzed in part c): Computes the attention pattern $\mathbf{A}$, which determines *which* source tokens to attend to at each destination position.
- **OV Circuit** (analyzed here): The linear transformation $\mathbf{W}_{OV}$ applied to source tokens, determining *what* information is extracted and *where* it is written.

In our SVD analysis below, when we write $\mathbf{W}_{OV}\mathbf{X}_{\text{src}}$, we are analyzing what happens to information from a *single* source token. The attention pattern then determines which source tokens this transformation is applied to and with what weights. These two circuits operate *independently*: you can understand what the OV circuit does to each source token without knowing which tokens will be attended to.

In part (d), we showed that each attention head can only write to the column space of $\mathbf{W}_{OV}$, a low-dimensional subspace. But this raises deeper questions: *which* directions in the source token does

the head read from? And *which* directions in the destination does it write to? The Singular Value Decomposition (SVD) provides a complete answer, revealing that attention heads are specialized communication channels between specific subspaces.

(i) Prove that $\text{rank}(\mathbf{W}_{OV}) \leq d_{\text{head}}$. Why does this imply that $\mathbf{W}_{OV}$ is a low-rank matrix (in typical transformer architectures)?

(ii) Because $\mathbf{W}_{OV}$ is low-rank, its SVD has a special structure. Let $\mathbf{W}_{OV} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$ be the (compact) SVD, where $\mathbf{U} \in \mathbb{R}^{d_{\text{model}} \times r}$, $\mathbf{\Sigma} \in \mathbb{R}^{r \times r}$, and $\mathbf{V} \in \mathbb{R}^{d_{\text{model}} \times r}$, with $r = \text{rank}(\mathbf{W}_{OV}) \leq d_{\text{head}}$. Consider a source token with residual stream vector $\mathbf{X}_{\text{src}} \in \mathbb{R}^{d_{\text{model}}}$. Show that the output of the OV circuit can be decomposed as:

$$\mathbf{W}_{OV}\mathbf{X}_{\text{src}} = \sum_{k=1}^{r} \sigma_k (\mathbf{V}_k^T \mathbf{X}_{\text{src}}) \mathbf{U}_k \tag{1}$$

where $\mathbf{U}_k$ and $\mathbf{V}_k$ are the k-th columns of $\mathbf{U}$ and $\mathbf{V}$, respectively, and σ_k is the k-th singular value.

(iii) Based on the decomposition above, define precisely:
   (a) The **read subspace**: Which directions in $\mathbf{X}_{\text{src}}$ can the head extract information from?
   (b) The **write subspace**: Which directions in the destination residual stream can the head write to?
   (c) What happens to information in $\mathbf{X}_{\text{src}}$ that is orthogonal to the read subspace?

   Hint: Think about which matrix "touches" the input first (this determines what can be read) and which matrix determines the form of the output (this determines what can be written).

(iv) **[Connecting to the Value Projection]** The value vector computed by the head is $\mathbf{v} = \mathbf{W}_V \mathbf{X}_{\text{src}}$, which lives in $\mathbb{R}^{d_{\text{head}}}$. Explain in 2-3 sentences how the read subspace $\text{span}(\mathbf{V}_1, \ldots, \mathbf{V}_r)$ relates to what $\mathbf{W}_V$ can "see" in the source token. Why is it natural that the read subspace dimension is at most d_{head}?

(v) **[Synthesis]** The paper describes attention heads as moving information "from the residual stream of one token to another." In 2-3 sentences, explain how the SVD $\mathbf{W}_{OV} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$ provides a complete characterization of this information movement. What role does each component ($\mathbf{V}$, $\mathbf{\Sigma}$, $\mathbf{U}$) play, and how does this relate to the attention pattern $\mathbf{A}$?

3. Coding Question: Transformer Interpretability

Please follow the instructions in this notebook. You will implement a single attention head and then create an induction copy head by combining a previous token head with a copying head.

This question was adapted from a past interview question by Anthropic.

4. Scaling Laws of Batch Size

A common question in neural network training is, how should I select my hyperparameters? While a proper hyperparameter sweep will always provide the best answer, sweeps can become impractical especially at larger network sizes.

In this homework question, we will derive a simple scaling law for the optimal learning rate under varying batch sizes. Complete this notebook, and answer the following questions in your submission:

(a) **For linear regression using SGD, attach curves of learning rate vs MSE loss for all the considered batch sizes, and curve showing relationship between batch size and optimal learning rate. What function does this resemble?**

(b) **Attach the same curves but for an MLP instead of linear regression. How do the scaling laws differ?**

(c) **Finally, show the same curves when using the Adam optimizer instead of SGD. Does the scaling for learning rate change when using Adam?**

5. 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-6 with 100 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.[1]

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[2], 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 100 trillion parameters. You can do the same calculations for the 1000 trillion parameter model, however.

(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

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

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

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 100 trillion ($N = 10^{14}$) 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 1000 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-6, which has 1 trillion parameters? How many H200 GPUs (VRAM $\approx$ 100 GB) would be required to contain the full GPT-6 model?

(f) (**Memory cost**)

This table[3] gives the price per megabyte of different storage technology, in price per megabyte, up to 2025.

Year	Memory (DRAM)	Flash/SSD	Hard disk
~1955	$613,000,000		$9,290
1970	$1,090,000		$388
1990	$221		$8.13
2003	$0.134	$0.455	$0.00194
2010	$0.0283	$0.00358	$0.000108
2025	$0.0040	$0.00005	$0.000039

The same costs *relative* to the cost of a hard disk in ~2025:

Year	Memory (DRAM)	Flash/SSD	Hard disk
~1955	15,700,000,000,000		238,000,000
1970	28,000,000,000		9,950,000
1990	5,670,000		208,000
2003	3,440	11,600	49.7
2010	727	91.8	2.79
2025	102	1.28	1.00

Suppose long-term memory storage can last 1 year before being replaced. How much money does it cost per year to store a 100 trillion parameter model on an SSD, the most expensive form of long-term storage? How much money does it cost to store a 100 trillion parameter model on DRAM memory? Use 2025 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.

H200 GPU has a memory bandwidth of 4.8 TB/s.

What is the minimal latency, in seconds, for the GPU to perform a single forward pass through our hypothetical GPT-6 model with 100 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.

[3] Source: Storage 2: Cache model – CS 61 2025.

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 can be a subtle business. However, there is a very simple method to improve thoroughput: batch mode. For example, GPT-6 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 numbers. **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 H200 GPU (100 GB VRAM version).

The most important specifications are:

- Unit price: 30000 USD.
- Rental price: 1.50 USD/hr.
- Speed: 1.98 petaFLOP/s = 1.98e15 FLOP/s.
- Power: 0.7 kiloWatt.
- Memory bandwidth: 4800 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 H200-hour? If we were to buy 1 petaFLOP-day of compute with rented H200 GPU, how much would it cost?**

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 an H200? How much money does it cost to train our hypothetical GPT-6?

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 H200, we only get 0.594 petaFLOP/s, instead of 1.98 petaFLOP/s).[4] 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[5]:

- iPhone 1: 150 million USD.

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

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

- A typical 5 nm chip: 0.5 billion USD.
- Airbus A380: 18 billion USD.[6]
- 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-6. Is it on the order of a large commercial actor like Google, or a state actor like China?

OpenAI and Broadcom announce strategic collaboration to deploy 10 gigawatts of OpenAI-designed AI accelerators and OpenAI and NVIDIA announce strategic partnership to deploy 10 gigawatts of NVIDIA systems are just two 10 gigawatt data centers that OpenAI plans to use. In perfect conditions, each data center would be able to run $10 \times 10^9 / 0.7 \times 10^3 \approx 14$ million H200 chips.

The largest companies, like OpenAI have GPUs on the order of tens of millions of H200s. What is the wall clock hour of training GPT-6, assuming you have 30 million H200s, 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.

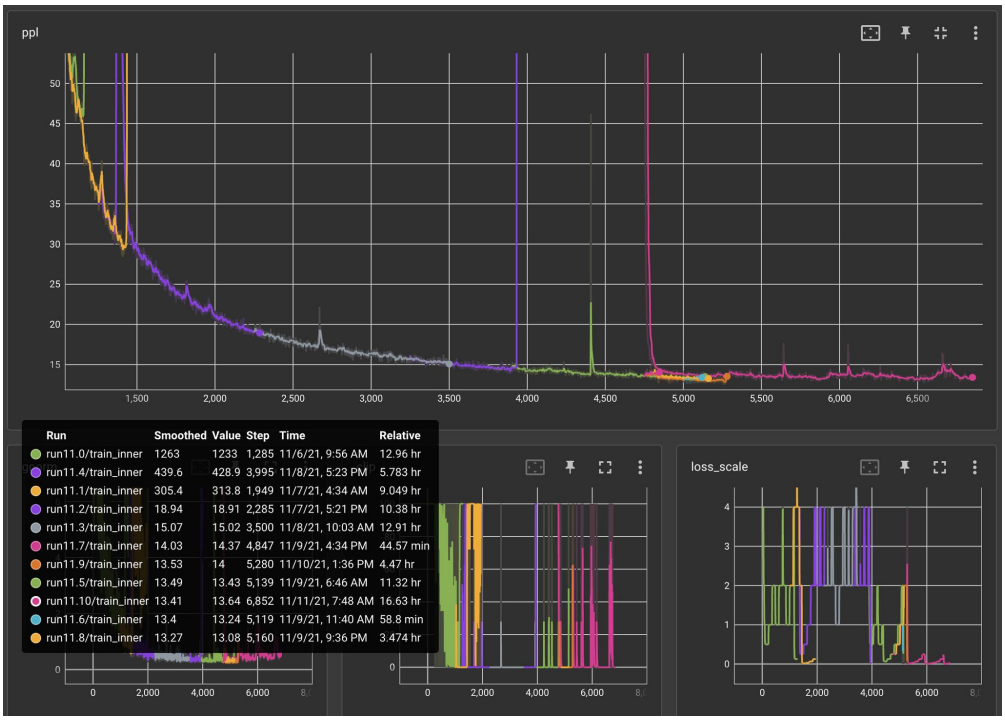
All together, we should expect the failures, restarts, deadends… to triple the cost at least, to ~1 billion USD.

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:

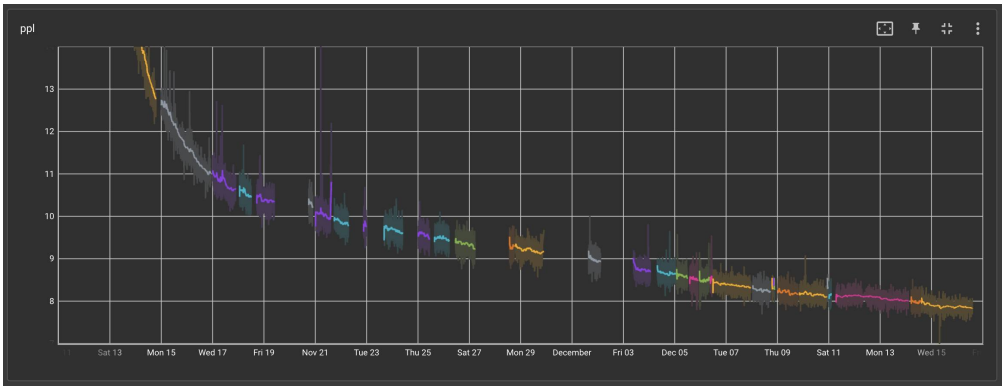
> These notes cover ~90 restarts over the course of training the lineage of this current model (experiments 12.xx).
>
> Found issues with the new dataset where perplexity was unreasonably low… After applying as much regex-ing as we could to salvage the dataset, we relaunched another set of experiments 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.

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



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.

If we estimate that GPT-5 has 1 trillion parameters, how many FLOPs would it take to infer on 1 million tokens? How much money would it cost if it were run with a H200?

The price offered by OpenAI is 120 USD per 1 million tokens. Assuming that GPT-5 cost 1 billion 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 1980 TFLOP/sec, as in a H200 GPU? Compare this to the actual value of 700 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.

When GPT-3 was trained, the standard chip was the NVIDIA A100, which was slower and much less energy-efficent. We estimate that GPT-3 training on A100 GPUs cost 6 million 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?**

6. Soft-Prompting Language Models

You are using a pretrained language model with prompting to answer math word problems. You are using chain-of-thought reasoning, a technique that induces the model to "show its work" before outputting a final answer.

Here is an example of how this works:

```
[prompt] Question: If you split a dozen apples evenly among yourself
and three friends, how many apples do you get? Answer: There are 12
apples, and the number of people is 3 + 1 = 4. Therefore, 12 / 4 = 3.
Final answer: 3\n
```

If we were doing hard prompting with a frozen language model, we would use a hand-designed [prompt] that is a set of tokens prepended to each question (for instance, the prompt might contain instructions for the task). At test time, you would pass the model the sequence and end after "Answer:" The language model will continue the sequence. You extract answers from the output sequence by parsing any tokens between the phrase "Final answer: " and the newline character "\n".

(a) Let's say you want to improve a frozen GPT model's performance on this task through soft prompting and training the soft prompt using a gradient-based method. This soft prompt consists of 5 vectors prepended to the sequence at the input — these bypass the standard layer of embedding tokens into vectors. (Note: we do not apply a soft prompt at other layers.) Imagine an input training sequence which looks like this:

```
["Tokens" 1-5: soft prompt] [Tokens 6-50: question]
```



```
[Tokens 51-70: chain of thought reasoning]
[Token 71: answer] [Token 72: newline]
[Tokens 73-100: padding].
```

We compute the loss by passing this sequence through a transformer model and computing the cross-entropy loss on the output predictions. If we want to train the soft-prompt to output correct reasoning and produce the correct answer, **which output tokens will be used to compute the loss?** (Remember that the target sequence is shifted over by 1 compared to the input sequence. So, for example, the answer token is position 71 in the input and position 70 in the target).

(b) Continuing the setup above, **how many parameters are being trained in this model?** You may write this in terms of the max sequence length S, the token embedding dimension E, the vocab size V, the hidden state size H, the number of layers L, and the attention query/key feature dimension D.

(c) **Mark each of the following statements as True or False and give a brief explanation.**

(i) If you are using an autoregressive GPT model as described in part (a), it's possible to precompute the representations at each layer for the indices corresponding to prompt tokens (i.e. compute them once for use in all different training points within a batch).

(ii) If you compare the validation-set performance of the *best possible* K-token hard prompt to the *best possible* K-vector soft prompt, the soft-prompt performance will always be equal or better.

(iii) If you are not constrained by computational cost, then fully finetuning the language model is always guaranteed to be a better choice than soft prompt tuning.

(iv) If you use a dataset of samples from Task A to do prompt tuning to generate a soft prompt which is only prepended to inputs of Task A, then performance on some other Task B with its own soft prompt might decrease due to catastrophic forgetting.

(d) Suppose that you had a family of related tasks for which you want use a frozen GPT-style language model together with learned soft-prompting to give solutions for the task. Suppose that you have substantial training data for many examples of tasks from this family. **Describe how you would adapt a meta-learning approach like MAML for this situation?**

(HINT: This is a relatively open-ended question, but you need to think about what it is that you want to learn during meta-learning, how you will learn it, and how you will use what you have learned when faced with a previously unseen task from this family.)

7. TinyML - Quantization and Pruning.

(This question has been adapted with permission from MIT 6.S965 Fall 2022)

TinyML aims at addressing the need for efficient, low-latency, and localized machine learning solutions in the age of IoT and edge computing. It enables real-time decision-making and analytics on the device itself, ensuring faster response times, lower energy consumption, and improved data privacy.

To achieve these efficiency gains, techniques like quantization and pruning become critical. Quantization reduces the size of the model and the memory footprint by representing weights and activations with fewer bits, while pruning eliminates unimportant weights or neurons, further compressing the model.

(a) Please complete `pruning.ipynb`. Then answer the following questions.

i. In part 1 the histogram of weights is plotted. **What are the common characteristics of the weight distribution in the different layers?**

ii. **How do these characteristics help pruning?**

iii. After viewing the sensitivity curves, please answer the following questions. **What's the relationship between pruning sparsity and model accuracy? (i.e., does accuracy increase or decrease when sparsity becomes higher?)**

iv. **Do all the layers have the same sensitivity?**

v. **Which layer is the most sensitive to the pruning sparsity?**

vi. (Optional) After completing part 7 in the notebook, please answer the following questions. **Explain why removing 30 percent of channels roughly leads to 50 percent computation reduction.**

vii. (Optional) **Explain why the latency reduction ratio is slightly smaller than computation reduction.**

viii. (Optional) **What are the advantages and disadvantages of fine-grained pruning and channel pruning? You can discuss from the perspective of compression ratio, accuracy, latency, hardware support (*i.e.*, requiring specialized hardware accelerator), etc.**

ix. (Optional) **If you want to make your model run faster on a smartphone, which pruning method will you use? Why?**

(b) Please complete `quantization.ipynb`. Then answer the following questions.

i. After completing K-means Quantization, please answer the following questions. **If 4-bit k-means quantization is performed, how many unique colors will be rendered in the quantized tensor?**

ii. **If n-bit k-means quantization is performed, how many unique colors will be rendered in the quantized tensor?**

iii. After quantization aware training we see that even models that use 4 bit, or even 2 bit precision can still perform well. **Why do you think low precision quantization works at all?**

iv. (Optional) Please read through and complete up to question 4 in the notebook, then answer this question.

Recall that linear quantization can be represented as $r = S(q - Z)$. Linear quantization projects the floating point range $[fp_{min}, fp_{max}]$ to the quantized range $[quantized_{min}, quantized_{max}]$.

That is to say,

$$r_{\max} = S(q_{\max} - Z)$$
$$r_{\min} = S(q_{\min} - Z)$$

Substracting these two equations, we have,

$$S = r_{\max}/q_{\max}$$
$$S = (r_{\max} + r_{\min})/(q_{\max} + q_{\min})$$
$$S = (r_{\max} - r_{\min})/(q_{\max} - q_{\min})$$
$$S = r_{\max}/q_{\max} - r_{\min}/q_{\min}$$

Which of these is the correct result of subtracting the two equations?

v. (Optional) Once we determine the scaling factor S, we can directly use the relationship between $r_{\min}$ and $q_{\min}$ to calculate the zero point Z.

$$Z = \text{int}(\text{round}(r_{\min}/S - q_{\min})$$
$$Z = \text{int}(\text{round}(q_{\min} - r_{\min}/S))$$
$$Z = q_{\min} - r_{\min}/S$$
$$Z = r_{\min}/S - q_{\min}$$

Which of these are the correct zero point?

vi. (Optional) After finishing question 9 on the notebook, **please explain why there is no ReLU layer in the linear quantized model.**

vii. (Optional) After completing the notebook, **please compare the advantages and disadvantages of k-means-based quantization and linear quantization.** You can discuss from the perspective of accuracy, latency, hardware support, etc.

8. 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:

- Naman Jain.
- Anant Sahai.
- Patrick Mendoza.
- Sultan Daniels.
- Kevin Frans.
- Yuxi Liu.
- Olivia Watkins.
- Yujun Lin.
- Ji Lin.
- Zhijian Liu.
- Song Han.
- Liam Tan.
- Romil Bhardwaj.

