Special Participation A: CS182 Homework 11

Hiya Shah

December 7, 2025

overall thoughts:
- performed well on proof-based Q's
- performed poorly on formula derivations, especially for recent papers that may not be in its training data
- also performed poorly on the GPU calculations

Question 1: LoRA

USER PROMPT:

Please start by solving #1. (Parts a, b, c, d)

RESPONSE:

Here is the solution for Question #1 from the Homework 11 PDF.

(in an earlier prompt I asked the model to output the problem statement before solving the question. I also asked it to go part by part to avoid context rot.

Problem Statement

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 $k \times l$ matrix, typically with $k \ll \min(m, l)$.

(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?

I found it interesting that the model generally did not hallucinate on the problem statement at all.

correct

Solutions and Explanations

Part (a) Explanation: Underfitting occurs when the model lacks the capacity (complexity) to capture the underlying patterns in the training data. In the context of LoRA, the capacity of the adaptation is directly constrained by the rank k. A small k limits the update matrix $\Delta W = AB$ to a low-rank subspace, restricting the types of changes that can be made to the pre-trained weights. To allow the model to learn more complex adaptations and fit the data better, you need to increase the expressiveness of the update.
Answer: You should **increase the rank** k. Increasing k adds more trainable parameters and increases the capacity of the adaptation, allowing the model to fit the training data more closely.

Part (b) Explanation: The weight update is given by $W = W_0 + AB$. During backpropagation, we compute gradients with respect to A and B. By the chain rule (ignoring the loss function derivative term for a moment), the gradients for the matrices generally take the form:

- Gradient w.r.t A: $\frac{\partial L}{\partial A} \approx \frac{\partial L}{\partial W} B^T$
- Gradient w.r.t B: $\frac{\partial L}{\partial B} \approx A^T \frac{\partial L}{\partial W}$

If both A and B are initialized to zero ($A = 0, B = 0$):

1. The forward pass is effectively just the pre-trained model ($W_0 + 0$), which is fine.
2. However, in the backward pass, the gradient for A depends on B (which is 0), resulting in $\frac{\partial L}{\partial A} = 0$.
3. Similarly, the gradient for B depends on A (which is 0), resulting in $\frac{\partial L}{\partial B} = 0$.

Since the gradients are zero, the parameters A and B will never be updated by SGD, and the model will remain at the initial state (W_0) forever.
Answer: Initializing both to zero causes the **gradients for both A and B to be zero** (vanishing gradients). Since the gradient of A depends on B and vice versa, if both are zero, no learning signal can backpropagate to updated the weights, and the model stays frozen at W_0.

correct, but it doesn't mention when either A or B are 0 (the typical initializ. approach)

Part (c) Explanation: The goal of LoRA is to start fine-tuning from the pre-trained model's performance. Ideally, at step 0, the model should be equivalent to the pre-trained model, implying we want the update AB to be zero initially. Standard LoRA initialization typically sets A to random Gaussian values and B to zero (or vice versa), ensuring that $AB = 0$. If you use Xavier (random) initialization for **both** A and B, the product AB will be a non-zero random matrix. This means:

$$W_{initial} = W_0 + \text{RandomNoise}$$

This immediately perturbs the well-tuned pre-trained weights W_0 with random noise. This can drastically degrade the model's initial performance (increasing loss) and may lead to training instability or optimization difficulties, as the model has to "unlearn" this noise before it can improve.
Answer: This initialization results in $AB \neq 0$, adding random noise to the pre-trained weights W_0 right at the start. This destroys the benefit of starting from the optimal pre-trained state, likely causing a spike in loss and potential training instability.

Part (d) **Explanation:** We need to calculate the storage for matrices A and B:

- Matrix A has dimensions $m \times k$.
- Matrix B has dimensions $k \times l$.
- Total elements $= mk + kl = k(m + l)$.
- Using 4-byte floats, total bytes $= 4 \times k(m + l)$.

A full-rank adaptation matrix (updating W directly) would have dimensions $m \times l$:

- Total elements $= ml$.
- Total bytes $= 4ml$.

Comparing the two:

$$\text{Ratio} = \frac{4k(m+l)}{4ml} = k\left(\frac{1}{l} + \frac{1}{m}\right)$$

Since $k \ll m$ and $k \ll l$, the LoRA memory requirement is significantly smaller.
Answer:

- **Memory required:** $4k(m + l)$ **bytes** (or $4(mk + kl)$ bytes). ✓ correct
- **Comparison:** This is significantly **less memory** than a single full-rank adaptation matrix (which requires $4ml$ bytes), as $k \ll \min(m, l)$.

USER PROMPT:
For part b, how are gradients for A and B usually initialized instead? ✓ I decided to further prompt it to give me typical initialization instead

RESPONSE:

For standard LoRA (Low-Rank Adaptation), the matrices A and B are usually initialized as follows:

- **Matrix A:** Initialized with **random Gaussian values** (often using Kaiming/He initialization).
- **Matrix B:** Initialized to **exact zeros**.

Why is this done?

1. **Preserves Pre-trained Behavior (Solves Part C):** The update rule is $W = W_0 + AB$. By setting $B = 0$, the product AB becomes a zero matrix initially.

$$W = W_0 + A \cdot 0 = W_0$$

This ensures the model starts fine-tuning *exactly* as the pre-trained model, without any added noise that might degrade performance in the first few steps.

2. **Enables Learning (Solves Part B):** This setup avoids the "vanishing gradient" problem seen when both are zero.

- **Gradient for B:** The gradient $\frac{\partial L}{\partial B}$ depends on A (specifically $\approx A^T \frac{\partial L}{\partial W}$). Since A is **random** (non-zero), the gradient for B is non-zero.
- **Result:** B will move away from zero in the very first update step. Once B is non-zero, the gradient for A (which depends on B) becomes non-zero, and both matrices begin to learn effectively.

Question 2: Transformer Interpretability

USER PROMPT:

please solve question 2 part a

RESPONSE:

Here is the solution for **Question 2, Part (a)** from the Homework 11 PDF.

Problem Statement

2. A Brief Introduction to Transformer Interpretability (a) The Simplest Transformer (A Zero-Layer Model)

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

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.

Solutions and Explanations

Part (a)(i) Answer:

$$L = W_U W_E T$$

✓ correct, but I would like it to show the 2-step method of arriving

Explanation:

1. **Input:** We start with the one-hot token matrix T of shape $(n_{vocab} \times n_{context})$.
2. **Embedding:** The embedding matrix W_E has shape $(d_{model} \times n_{vocab})$. To get the residual stream X^0, we multiply the embedding matrix by the token matrix: $X^0 = W_E T$.
3. **Unembedding:** In a zero-layer model, the "final" residual stream is simply the initial embedding ($X^{final} = X^0$). The unembedding matrix W_U has shape $(n_{vocab} \times d_{model})$. To get the logits L, we project the residual stream back to the vocabulary size: $L = W_U X^{final} = W_U(W_E T)$.

Part (a)(ii) Answer:

somehow the model associated this term even though the solutions didn't mention it

- **Algorithm:** This model implements a **Bigram Model** (or a lookup table). It learns the statistical likelihood of the next token based *solely* on the current token, without any context from the rest of the sequence. Specifically, the matrix product $W_U W_E$ results in an $(n_{vocab} \times n_{vocab})$ matrix where the entry (i, j) represents the "score" or probability of token i appearing given token j is the input.
- **Dependency:** The prediction for the token at position t depends **only on the input token at position** t. Because there are no attention heads or mixing layers, there is no mechanism for information to move between different positions (tokens) in the sequence.

USER PROMPT:

please solve question 2 part b

RESPONSE:

Here is the solution for **Question 2, Part (b)** from the Homework 11 PDF.

Problem Statement

In the original Vaswani et al. paper... 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. **(i)** Show that these formulations are equivalent. Specifically, demonstrate how W_O^H can be constructed from the individual W_O^h matrices to make the two expressions for H_{out} identical. **(ii)** What is an advantage and disadvantage of each view?

Solutions and Explanations

Part (b)(i): Proof of Equivalence **Answer:** We can construct the large output matrix W_O^H by **concatenating the individual head output matrices W_O^h horizontally**:

$$W_O^H = [W_O^1 \quad W_O^2 \quad \cdots \quad W_O^H]$$

Explanation: Using the concatenation formula $H_{out} = W_O^H R^H$:

$$H_{out} = [W_O^1 \quad W_O^2 \quad \cdots \quad W_O^H] \begin{bmatrix} r^1 \\ r^2 \\ \vdots \\ r^H \end{bmatrix}$$

According to the rules of block matrix multiplication:

$$H_{out} = W_O^1 r^1 + W_O^2 r^2 + \cdots + W_O^H r^H = \sum_{h=1}^{H} W_O^h r^h$$

This is exactly the expression for the "additive and independent" view.

this was perfectly solved and fully matched the solutions

Part (b)(ii): Advantages and Disadvantages Concatenation View (Original Vaswani et al.):

- **Advantage: Computational Efficiency.** Concatenating operations allows the computer to perform one large matrix multiplication (GEMM) rather than many small ones. GPUs are highly optimized for large matrix operations.
- **Disadvantage: Obscures Interpretability.** It treats the layer as a single monolithic transformation, making it harder to analyze the specific role or "circuit" of a specific head.

Additive / Independent View (Elhage et al.):

- **Advantage: Interpretability.** It explicitly models the Residual Stream as a communication channel where independent agents (heads) read and write information in parallel.
- **Disadvantage: Inefficient Implementation.** Writing code that loops over heads or performs H separate small matrix multiplications is generally slower on modern hardware.

added disadvantages that weren't in the solution. is also a bit too verbose.

USER PROMPT:

please solve question 2 part c

RESPONSE:

Here is the solution for **Question 2, Part (c)** from the Homework 11 PDF.

Problem Statement

(i) Derive the expression for S_{ij} as a bilinear form involving X_i and X_j and a single "virtual" weight matrix W_{QK}. Explicitly define W_{QK} in terms of W_Q and W_K. **(ii)** Consider a toy scenario where $d_{Model} = 3$. Interpret W_{QK} for: (1) Identity, (2) Single non-zero diagonal, (3) Off-diagonal, (4) Negative Identity.

Solutions and Explanations

Part (c)(i) Answer:

$$S_{ij} = X_i^T W_{QK} X_j$$

$$W_{QK} = W_Q^T W_K$$

Derivation: $S_{ij} = Q_i^T K_j = (W_Q X_i)^T (W_K X_j) = X_i^T W_Q^T W_K X_j$.

Part (c)(ii) 1. $W_{QK} = I$ (Identity Matrix)

- **Math:** $S_{ij} = X_i \cdot X_j$.
- **Interpretation:** The head looks for **similarity**. It attends to tokens X_j that are most similar (parallel) to the current query token X_i.

2. W_{QK} has 1 at $(0, 0)$ and 0 elsewhere

- **Math:** $S_{ij} = x_{i,1} \cdot x_{j,1}$.
- **Interpretation:** The score depends **only on the first dimension**. The head looks for tokens where the first feature matches. Information in the 2nd and 3rd dimensions is ignored.

3. Off-Diagonal / Interaction Case

- **Interpretation:** This represents a **cross-feature interaction**. If W_{QK} has a 1 at index (u, v), the head looks for tokens where the **Key's v-th feature** matches the **Query's u-th feature**.

4. $W_{QK} = -I$ (Negative Identity)

- **Math:** $S_{ij} = -(X_i \cdot X_j)$.
- **Interpretation:** The head looks for **anti-similarity** (vectors pointing in opposite directions).
- **Effect on Softmax:** It **suppresses attention to similar tokens** and focuses on dissimilar ones (since similar tokens produce large negative scores which softmax drives to zero).

USER PROMPT:

please solve question 2 part d

RESPONSE:

Here is the solution for **Question 2, Part (d)** from the Homework 11 PDF.

Problem Statement

(i) Show that the i-th column of $X_0 A^T$ is an attention-weighted average of the source token vectors: $(X_0 A^T)_i = \sum_{j=1}^{n_{context}} A_{ij} X_0^j$. **(ii)** Write the full expression for the final residual stream, X_{final}. **(iii)** Prove that the update vector $h_i(X_0)_t$ must lie within the column space of W_{OV}^i.

Solutions and Explanations

Part (d)(i) **Proof:** The element at row r and column i of $X_0 A^T$ is:

$$Z_{r,i} = \sum_k (X_0)_{r,k} (A^T)_{k,i} = \sum_k (X_0)_{r,k} A_{i,k}$$

The i-th column is the vector formed by these elements for all r:

$$(X_0 A^T)_i = \sum_{k=1}^{n_{context}} A_{i,k} X_0^k$$

Renaming k to j:

$$(X_0 A^T)_i = \sum_{j=1}^{n_{context}} A_{ij} X_0^j$$

Part (d)(ii) Answer:

$$X_{final} = X_0 + \sum_{h=1}^{H} W_{OV}^h X_0 (A^h)^T$$

Part (d)(iii) Proof: The update vector is the t-th column of $h_i(X_0) = W_{OV}^i(X_0(A^i)^T)$. Let v_t be the t-th column of the weighted-sum matrix $X_0(A^i)^T$. Then the update vector is $W_{OV}^i v_t$. By definition, Ax is a linear combination of the columns of A. Thus, $W_{OV}^i v_t$ lies in the span (column space) of W_{OV}^i.

USER PROMPT:

please solve question 2 part e

RESPONSE:

these proofs largely follow the solutions

Here is the solution for **Question 2, Part (e)** from the Homework 11 PDF.

Problem Statement

(i) Prove that $\text{rank}(W_{OV}) \leq d_{head}$. **(ii)** Show that $W_{OV}X_{src} = \sum_{k=1}^{r} \sigma_k (V_k^T X_{src}) U_k$. **(iii)** Define the read/write subspaces and what happens to orthogonal information. **(iv)** Explain the connection to W_V. **(v)** Synthesis: Explain how the SVD characterizes information movement.

Solutions and Explanations

Part (e)(i) Proof: $W_{OV} = W_O W_V$. The rank of a product is limited by the min rank of its factors: $\text{rank}(W_{OV}) \leq \min(\text{rank}(W_O), \text{rank}(W_V)) \leq d_{head}$. This implies it is low-rank because typically $d_{head} \ll d_{model}$.

Part (e)(ii) Derivation: $W_{OV}X_{src} = (U\Sigma V^T)X_{src} = U\Sigma(V^T X_{src})$. This expands to the sum $\sum_{k=1}^{r} \sigma_k (V_k^T X_{src}) U_k$.

Part (e)(iii)

1. **Read Subspace:** Span of V (Row space). Directions in input the head can "see".
2. **Write Subspace:** Span of U (Column space). Directions in output the head writes to.
3. **Orthogonal Info:** It is zeroed out (multiplied by 0). The head cannot process it.

Part (e)(iv) The read subspace is the subspace preserved by the projection W_V. Since W_V maps to $\mathbb{R}^{d_{head}}$, it can preserve at most d_{head} dimensions. Thus the read subspace rank is naturally bounded by d_{head}.

Part (e)(v) Synthesis:

- **V (Sensor):** Reads the input by projecting onto specific features.
- **Σ (Amplifier):** Determines the gain/strength of the transfer.
- **U (Writer):** Writes the result as a feature vector to the destination.

The attention pattern A acts as a router/volume knob determining which source tokens apply this transformation.

Question 5: Fermi Estimation

USER PROMPT:
please solve question 5 part a

RESPONSE:

Here is the solution for **Question 5, Part (a)** from the Homework 11 PDF.

Answer: This part of the problem does not require a solution.

As explicitly stated in the problem text ("This is not a question..."), Part (a) serves as context/reading material to establish the motivation for the subsequent calculations. It uses **Fermi estimation** to propose that a human-level AGI might require approximately 10^{15} **(1000 trillion) parameters**.

USER PROMPT:
please solve question 5 part b

RESPONSE:

Here is the solution for **Question 5, Part (b)** from the Homework 11 PDF.

Solutions and Explanations

1. Symbolic Derivation **Step 1:** From constraint $C = C_0ND$, write $D = \frac{C}{C_0N}$.
Step 2: Substitute into $L(N) = AN^{-\alpha} + B(\frac{C}{C_0})^{-\beta}N^{\beta} + L_0$.
Step 3: Differentiate w.r.t N and set to 0.

$$\alpha AN^{-\alpha-1} = \beta B(\frac{C}{C_0})^{-\beta}N^{\beta-1}$$

Solving for N:

$$N_{opt} = \left(\frac{\alpha A}{\beta B}\right)^{\frac{1}{\alpha+\beta}}\left(\frac{C}{C_0}\right)^{\frac{\beta}{\alpha+\beta}}$$

Step 4: Solve for D_{opt} using $D = C/(C_0N)$:

$$D_{opt} = \left(\frac{C}{C_0}\right)^{\frac{\alpha}{\alpha+\beta}} \dots$$

these do
not
match the
solutions,
and it did not
plug in
numerical values

2. Numerical Solution *(The user asked for the symbolic derivation, but note that Part (c) provides the simplified numerical laws to use moving forward.)*

USER PROMPT:
please solve question 5 part c

RESPONSE:

Here is the solution for **Question 5, Part (c)** from the Homework 11 PDF.

Problem Statement

Using Chinchilla laws $N = 0.1C^{0.5}$ and $D = 1.7C^{0.5}$, calculate compute cost and dataset size for a 100 trillion parameter model ($N = 10^{14}$).

Solution

1. Calculate Compute Cost (C):

$$N = 0.1C^{0.5} \implies 10N = C^{0.5} \implies C = 100N^2$$

$$C = 100 \times (10^{14})^2 = 100 \times 10^{28} = 10^{30} \text{ FLOPs}$$

2. Calculate Dataset Size (D):

$$D = 1.7C^{0.5} = 1.7(10N) = 17N$$

$$D = 17 \times 10^{14} = 1.7 \times 10^{15} \text{ tokens}$$

Answer:

- **Compute (C):** 10^{30} FLOPs.
- **Dataset (D):** 1.7×10^{15} tokens (1.7 quadrillion).

USER PROMPT:
please solve question 5 part d

RESPONSE:

Here is the solution for **Question 5, Part (d)** from the Homework 11 PDF.

Problem Statement

Assume 1.4 tokens/word, 400 words/page, 300 pages/book. How many words is 1000 trillion tokens? How many books? Compare to Library of Congress and Google Books.

Solution

1. Number of Words:

$$10^{15} \text{ tokens}/1.4 = 7.14 \times 10^{14} \text{ words}$$

(714 Trillion words).

2. Number of Books: Words per book $= 400 \times 300 = 120,000$.

$$\text{Books} = \frac{7.14 \times 10^{14}}{120,000} \approx 5.95 \times 10^{9} \text{ books}$$

(6 Billion books).

3. Comparison:

- **Library of Congress:** 32 million books. The dataset is **186x larger**.
- **Google Books:** 40 million scanned. The dataset is **150x larger**.

USER PROMPT:

please solve question 5 part e

RESPONSE:

Here is the solution for **Question 5, Part (e)** from the Homework 11 PDF.

Problem Statement

Given 16-bit parameters: 1. Memory for 1 billion params? 2. Memory for 1 trillion params? 3. How many H200 GPUs (100 GB VRAM) for the 1T model?

Solution

1. 1 Billion Params: $10^{9} \times 2 \text{ bytes} = 2$ GB.

2. 1 Trillion Params: $10^{12} \times 2 \text{ bytes} = 2$ TB.

3. H200 GPUs Required:

$$\text{Num GPUs} = \frac{2000 \text{ GB}}{100 \text{ GB}} = 20 \text{ GPUs}$$

USER PROMPT:

please solve question 5 part f

RESPONSE:

Here is the solution for **Question 5, Part (f)** from the Homework 11 PDF.

Problem Statement

Cost to store 100 trillion parameter model for 1 year on SSD ($0.00005/MB) vs DRAM ($0.0040/MB).

Solution

Model Size: 10^{14} params $\times$ 2 bytes $= 200$ TB $= 2 \times 10^8$ MB.

SSD Cost:

$$2 \times 10^8 \times \$0.00005 = \$10,000$$

DRAM Cost:

$$2 \times 10^8 \times \$0.0040 = \$800,000$$

USER PROMPT:

please solve question 5 part g

RESPONSE:

Here is the solution for **Question 5, Part (g)** from the Homework 11 PDF.

Solution

1. GPT-6 (100T Params): Size = 200 TB. Bandwidth = 4.8 TB/s.

$$\text{Latency} = \frac{200}{4.8} \approx 41.7 \text{ seconds}$$

$$\text{Throughput} = 60/41.7 \approx 1.44 \text{ tokens/min}$$

2. GPT-3 (175B Params): Size = 0.35 TB.

$$\text{Latency} = \frac{0.35}{4.8} \approx 0.073 \text{ seconds (73ms)}$$

$$\text{Throughput} \approx 823 \text{ tokens/min}$$

3. Batch Size for GPT-3: Activation size per token $\approx$ 23.6 MB. Model Size = 350,000 MB.

$$B = \frac{350,000}{23.6} \approx 14,830$$

USER PROMPT:

please solve question 5 part h

RESPONSE:

Here is the solution for **Question 5, Part (h)** from the Homework 11 PDF.

Solution

1. **1 PF-Day:** $10^{15} \times 24 \times 3600 = 8.64 \times 10^{19}$ FLOPs.

2. **H200 Hours equivalent:** $24/1.98 \approx 12.12$ hours.

3. **Cost of 1 PF-Day:** $12.12 \times \$1.50 \approx \$18.18$.

4. **GPT-3 Cost (3640 PF-Days):** $3640 \times \$18.18 \approx \$66,182$.

5. **GPT-6 Cost (10^{30} FLOPs):** PF-Days $= 10^{30}/8.64 \times 10^{19} \approx 1.157 \times 10^{10}$. Cost $\approx$ 11.6 billion $\times \$18.18 \approx \$210$ Billion.

6. **Comparison:** More expensive than the Apollo Program ($178B).

194 days is not mentioned

USER PROMPT:

please solve question 5 part i

RESPONSE:

Here is the solution for **Question 5, Part (i)** from the Homework 11 PDF.

Answer: This part does not require a solution. It is a reading section describing how failures and restarts can triple the effective cost of training.

USER PROMPT:

please solve question 5 part j

RESPONSE:

Here is the solution for **Question 5, Part (j)** from the Homework 11 PDF.

Solution

1. **FLOPs for 1M tokens (1T param model):** $2 \times 10^{12} \times 10^{6} = 2 \times 10^{18}$ FLOPs.

2. **H200 Cost:** Time $= 2 \times 10^{18}/1.98 \times 10^{15} \approx 1010$ sec ≈ 0.28 hours. Cost $= 0.28 \times \$1.50 \approx \$0.42$.

3. **Tokens to recoup $1B:** Need $1B revenue at $120/1M tokens. Tokens $= (10^{9}/120) \times 10^{6} \approx 8.33$ Trillion tokens.

4. **Equivalent Essays:** 8.33 Trillion/1400 $\approx$ 5.95 Billion essays.

(actually 6.1 billion)

USER PROMPT:

please solve question 5 part k

RESPONSE:

Here is the solution for **Question 5, Part (k)** from the Homework 11 PDF.

Solution

1. **Min Energy:** $32 \times k_B T \ln 2 \approx 9.6 \times 10^{-20}$ Joules.
2. **Min Power:** $P = E \times 1.98 \times 10^{15} \approx 1.9 \times 10^{-4}$ Watts.
3. **Comparison:** Actual 700W is ~ 3.7 million times higher than the theoretical limit.

USER PROMPT:

please solve question 5 part l

RESPONSE:

Here is the solution for **Question 5, Part (l)** from the Homework 11 PDF.

Solution

1. **Carbon Cost:** 552 tons $\times$ \$112 $=$ \$61,824. This is $\sim 1\%$ of the \$6M training cost.
2. **Burgers:** 552,000 kg CO2$/(0.113 \times 50) \approx 97,700$ burgers.

correctly performs calc but does not analyze

for j-l, all Fermi estimates align with the solution values. **Verdict:** One-shot correct.

Question 6: Soft Prompting

USER PROMPT:

please solve question 6 part a

RESPONSE:

this answer is incorrect. the model may be experiencing degradation.

- tokens 1-4 : soft-prompt
- tokens 5-49 : reasoning
- tokens 70: answer
- token 71: newline
- tokens 72-100: padding

Here is the solution for **Question 6, Part (a)** from the Homework 11 PDF.

Problem Statement

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?**

Solution

Answer: The loss should be computed on the output predictions corresponding to:

1. The **chain of thought reasoning** (indices 51-70).
2. The **answer** (index 71).
3. The **newline/stop token** (index 72).

Specifically, we use the model outputs at indices **50 through 71**. We do **not** compute loss on the soft prompt (1-5) or the question (6-50).

USER PROMPT:

please solve question 6 part b

RESPONSE:

Here is the solution for **Question 6, Part (b)** from the Homework 11 PDF.

Problem Statement

How many parameters are being trained in this model?

Solution

Answer: $5 \times E$ (or $5 \times H$).

Explanation: The model weights are frozen. The only learnable parameters are the **5 soft prompt vectors** prepended to the input. Each vector has dimension E (embedding size). Total = $5 \times E$. correct

USER PROMPT:

please solve question 6 part c (parts i ii iii iv)

RESPONSE:

however, it doesn't note how it is done practically

all correct

Here is the solution for **Question 6, Part (c)** from the Homework 11 PDF.

Solutions

(i) True Explanation: In an autoregressive model, attention is masked. The prompt tokens (positions 1-5) only attend to themselves/preceding tokens. They do not see the specific question tokens that come later. Thus, their activations are constant across the batch and can be cached.

(ii) True Explanation: A soft prompt consists of continuous vectors. A hard prompt consists of discrete embeddings. The continuous space contains the discrete embeddings as a subset. Therefore, the optimal soft prompt is at least as good as the optimal hard prompt.

(iii) False Explanation: Full fine-tuning can overfit on small datasets or lead to catastrophic forgetting. Soft prompt tuning acts as a regularizer, sometimes yielding better generalization.

(iv) False Explanation: The model weights are frozen. Training a prompt for Task A only updates the prompt vectors, not the shared model weights. Therefore, Task B (using its own prompt and the shared frozen weights) is unaffected.

USER PROMPT:

please solve question 6 part d

RESPONSE:

Here is the solution for **Question 6, Part (d)** from the Homework 11 PDF.

Problem Statement

Describe how you would adapt a meta-learning approach like MAML for a family of tasks using a frozen GPT model and soft prompting.

Solution

To adapt MAML:

1. **What to Learn:** Learn the **initialization of the soft prompt vectors** (P_{init}).
2. **Inner Loop (Adaptation):** For a sampled task, start at P_{init}. Use the support set to perform gradient descent steps *on the prompt vectors only* to get P_i'.
3. **Outer Loop (Meta-Update):** Evaluate P_i' on the query set. Update P_{init} to minimize the query loss across tasks.

Inference: When faced with a new task, initialize with the meta-learned P_{init} and run a few gradient steps using the new task's data to adapt.

here, it is not as verbose; may be experiencing context rot. while it mentions GD, it doesn't mention SGD specifically. it also does not do variations on the basic MAML approach, which the solutions do have.