Intro:

This is an attempt to interact with deepseek on non-coding parts of homework 9. The purpose of this study is to better understand how to prompt/interact with LLMs more effectively and LLM's capability of solving real life reasoning/math related problems with few-shot prompting. The specific model I interacted with was DeepSeek-V3.2. I used Deepseek's web ui to interact with the model. I will focus on how different prompting methods (or modes of reasoning) affect a model's one-shot correctness of the problems, and how to improve its accuracy without providing more in context examples. Note that only important parts of the full conversations are included here (for the sake of length), for full conversation, please refer to the links:

Conversation traces:

https://chat.deepseek.com/share/jkzqnyn7j8say9v7jc
https://chat.deepseek.com/share/838vauzbwa2g0ynfby
https://chat.deepseek.com/share/pp0exea4mnmt36qfqu

Annotated Trace:

For deepseek webui, there is an option called "DeepThink", with explanation (Think before responding to solve reasoning problems). To better understand its effect, I started the interaction on problem 1 with this feature off, and my prompt is simply telling the model that this is a homework problem from my deep learning class and that I need the model to solve it and clearly present the final answer for each part:

Here is a homework problem for my deeplearning course. Please solve it and clearly present the final answer for each subparts:
Suppose q, k ∈ R
d
are two random vectors with q, k iid∼ N(μ, σ2
I), where μ ∈ R
d
and σ ∈ R
+. In other
words, each component qi of q is drawn from a normal distribution with mean μ and standard deviation σ,
and the same is true for k.
(a) Define E[q
T k] in terms of μ, σ and d.
(b) Considering a practical case where μ = 0 and σ = 1, define Var(q
T k) in terms of d.
(c) Continue to use the setting in part (b), where μ = 0 and σ = 1. Let
s be the scaling factor on the dot
product. Suppose we want E[
q
T k
s
] to be 0, and Var(
q
T k
s
) to be σ = 1. What should s be in terms of d?

Model answer:

Alright, let's go step-by-step.

Part (a)

We have $q, k \in \mathbb{R}^d$, $q_i \overset{\text{iid}}{\sim} \mathcal{N}(\mu, \sigma^2)$, $k_i \overset{\text{iid}}{\sim} \mathcal{N}(\mu, \sigma^2)$, and q independent of k.

We want $\mathbb{E}[q^T k]$.

$$q^T k = \sum_{i=1}^{d} q_i k_i$$

$$\mathbb{E}[q_i k_i] = \mathbb{E}[q_i]\mathbb{E}[k_i] = \mu \cdot \mu = \mu^2$$

Therefore:

$$\mathbb{E}[q^T k] = \sum_{i=1}^{d} \mu^2 = d\mu^2$$

Answer for (a):

$$\boxed{d\mu^2}$$

Part (b)

Now $\mu = 0, \sigma = 1$.

So $q_i, k_i \sim \mathcal{N}(0,1)$, i.i.d., independent between q and k.

We want $\mathrm{Var}(q^T k)$.

$$q^T k = \sum_{i=1}^{d} q_i k_i$$

Since q_i and k_i are independent with mean 0:

$$\mathrm{Cov}(q_i k_i, q_j k_j) = 0 \quad \text{for } i \neq j$$

because $\mathbb{E}[q_i k_i q_j k_j] = \mathbb{E}[q_i k_i]\mathbb{E}[q_j k_j] = 0 \cdot 0 = 0$.

So:

$$\mathrm{Var}(q^T k) = \sum_{i=1}^{d} \mathrm{Var}(q_i k_i)$$

$$\mathrm{Var}(q_i k_i) = \mathbb{E}[q_i^2 k_i^2] - (\mathbb{E}[q_i k_i])^2$$

$$\mathbb{E}[q_i k_i] = 0$$

$$\mathbb{E}[q_i^2 k_i^2] = \mathbb{E}[q_i^2]\mathbb{E}[k_i^2] = (1)(1) = 1$$

So $\mathrm{Var}(q_i k_i) = 1 - 0 = 1$.

Thus:

$$\mathrm{Var}(q^T k) = d \cdot 1 = d$$

Answer for (b):

$$\boxed{d}$$

Part (c)

We still have $\mu = 0, \sigma = 1$ for the components of q, k.

Let scaled dot product be $\frac{q^T k}{s}$.

We want:

1. $\mathbb{E}\left[\frac{q^T k}{s}\right] = 0$
2. $\mathrm{Var}\left(\frac{q^T k}{s}\right) = 1$

From (a) with $\mu = 0$, $\mathbb{E}[q^T k] = 0$, so condition 1 is automatically satisfied for any s.

From (b), $\mathrm{Var}(q^T k) = d$.

So:

$$\mathrm{Var}\left(\frac{q^T k}{s}\right) = \frac{d}{s^2}$$

We set:

$$\frac{d}{s^2} = 1 \quad \Rightarrow \quad s^2 = d \quad \Rightarrow \quad s = \sqrt{d}$$

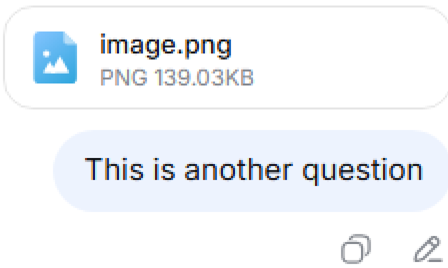
(We take positive scaling factor.)

Answer for (c):

$$\boxed{\sqrt{d}}$$

Note that the model was able to get all three subparts correct with only one one-shot attempt. Another thing to note is that although I did not turn on the “DeepThink” feature, nor did I explicitly prompt the model to explain its reasoning (like a chain of thought style), the model nonetheless still explained its thought process for solving the problem. I believe this can be attributed to two reasons: when the model is trained/finetuned, it has been trained toward a “reasoning” style where it likes to explain its reasoning when solving complex mathy problems; and that in the prompt I mentioned this is a course homework, so the model understands that I probably need more than just the final answer.

I therefore continue with the next question:



Model answer:

Let's go step-by-step.

Part (a)

We have:

Keys $K \in \mathbb{R}^{3\times4}$:

$$K = \begin{pmatrix} 1 & 0 & 5 & 0 \\ 2 & 3 & 0 & 0 \\ 0 & 4 & 0 & 1 \end{pmatrix}$$

Values $V \in \mathbb{R}^{3\times4}$:

$$V = \begin{pmatrix} 2 & 1 & 0 & 1 \\ 0 & 4 & -1 & 0 \\ 1 & 3 & 4 & -1 \end{pmatrix}$$

Query $q \in \mathbb{R}^3$:

$$q = \begin{pmatrix} 1 \\ 1 \\ 2 \end{pmatrix}$$

Step 1: Compute similarity scores

Similarity = dot product of q with each column of K:

- Column 1: $1 \cdot 1 + 1 \cdot 2 + 2 \cdot 0 = 1 + 2 + 0 = 3$
- Column 2: $1 \cdot 0 + 1 \cdot 3 + 2 \cdot 4 = 0 + 3 + 8 = 11$
- Column 3: $1 \cdot 5 + 1 \cdot 0 + 2 \cdot 0 = 5 + 0 + 0 = 5$
- Column 4: $1 \cdot 0 + 1 \cdot 0 + 2 \cdot 1 = 0 + 0 + 2 = 2$

Scores: $[3, 11, 5, 2]$

Step 2: Argmax

Argmax picks index with highest score.

Max score = 11 → index = 2 (if we index from 1: column 2; from 0: column 1? Let's clarify indexing.)

Actually, standard matrix columns:
Column 1 → score 3
Column 2 → score 11
Column 3 → score 5
Column 4 → score 2

So argmax index = 2 (1-based indexing) or index 1 (0-based indexing).

Step 3: Output

Output = the **value** at that same column index from V:

V's column 2: $[1, 4, 3]^T$.

So output vector = $\begin{pmatrix} 1 \\ 4 \\ 3 \end{pmatrix}$.

Answer for (a):

$$\boxed{\begin{pmatrix} 1 \\ 4 \\ 3 \end{pmatrix}}$$

Part (b)

In **argmax attention**, the attention weights are a one-hot vector: 1 for the highest score, 0 for others.

Gradient flow problem:

- In **softmax attention**, small changes in scores change the weights smoothly → gradients flow back to queries and keys through all score entries.
- In **argmax attention**, the output is discontinuous with respect to the scores: unless the argmax index changes, gradients w.r.t. scores are zero; when it changes, the gradient is infinite or undefined in a discrete jump.

Thus, during backpropagation, the gradient of the loss with respect to the similarity scores is zero almost everywhere. This means:

- The model cannot learn to adjust queries and keys to improve scores slightly — no learning signal for "this key was almost the best."
- Training would fail with gradient-based methods because argmax is not differentiable.

Answer for (b):

$$\boxed{\text{Argmax breaks gradient flow, making training impossible with gradient descent.}}$$

Note that because it is hard to extract matrix from the problem and input into model, so here I utilized the webui's native OCR ability. By only providing a screenshot of the problem, the web ui first extracts text from the screenshot and feeds the text into the model. As we can see, the extraction was pretty accurate and the model was able to get the correct answer one-shot.

Proceeding to the next question, because the question has code in its prompt, I again screenshot the problem and utilized the webui's native OCR capability:

image.png
PNG 153.49KB

Here is another problem

(a)

`self.W_k = nn.Linear(d_model, ___(a)___)`

In multi-head attention, each head projects the input to `d_k` dimensions, and there are `num_heads` of them, so total output size = `num_heads * d_k = d_model` (since `d_k = d_model // num_heads`).

So `W_k` should output `d_model` to be split into heads.

$$\boxed{d\_model}$$

(b)

`self.W_v = nn.Linear(___(b)___, d_model)`

Similarly, `W_v` takes in `d_model` and outputs `d_model` (then split into heads).

$$\boxed{d\_model}$$

(c)

`self.W_o = nn.Linear(d_model, ___(c)___)`

The final output linear layer takes concatenated head outputs (size `d_model`) and projects to `d_model` again usually.

$$\boxed{d\_model}$$

(d)

`attn_scores = torch.matmul(___(d)___, K.transpose(-2, -1))`

This is `Q @ K^T`, so:

$$\boxed{Q}$$

(e)

`attn_probs = torch.___(e)___(attn_scores, dim=-1)`

Softmax along the last dimension (over keys).

$$\boxed{softmax}$$

(f)

`output = torch.matmul(attn_probs, ___(f)___)`

Multiply attention probabilities by `V`.

$$\boxed{V}$$

(g)

In `split_heads`:
`return x.view(batch_size, seq_length, ___(g)___, self.d_k).transpose(1, 2)`

We reshape `(batch_size, seq_length, d_model)` to `(batch_size, seq_length, num_heads, d_k)` then transpose to `(batch_size, num_heads, seq_length, d_k)`.

So `(g)` = `num_heads`.

Again, the model was able to solve the problem correctly in one try without any further prompting. The style it answered the question is also consistent with previous questions: a step by step solution with explanations. Note that one would expect a model to easily solve this type of question because this question simply asks for a vanilla implementation of the attention mechanism. Such a concept should be abundant in model's training data and thus the model doesn't necessarily need to "reason", it is simply completing something that it already "memorized".

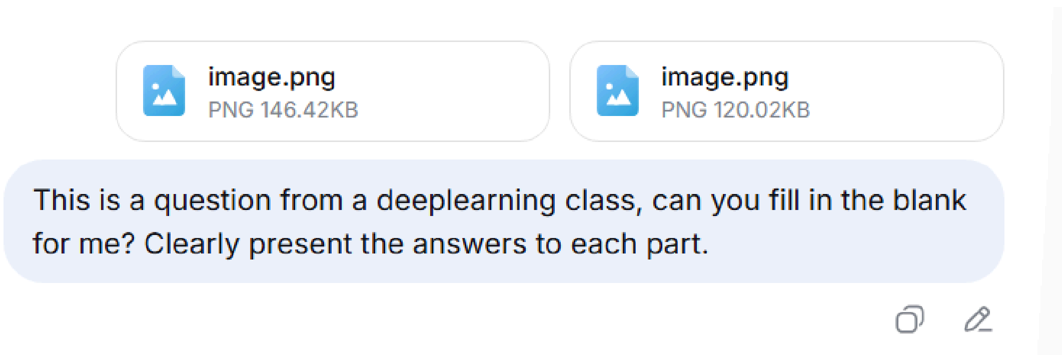
For the next part of the problem, because the problem is too long, I broke it down into two separate screenshots and uploaded them to the ui. The first image is the example provided by the question of a MHA implementation, whereas the second image is code for multi query attention (some modifications from the MHA example provided) with blanks. Without any information (I did not include the part that tells the model the specific place holders like ___A___ are blanks to fill with options), I directly prompted the model to fill in the blanks. I did this because I expected the model to be able to tell places like ___A____ are blanks to fill. I also did not provide the options for the model to choose because I expect the model to know the answers without the options. However, something interesting happened:

> We are given **Multi-Query Attention** pseudocode, where **all heads share the same keys and values**.
>
> The shapes given in the comments:
>
> - `prev_K`: `[b, n, k]` (shared across heads)
> - `prev_V`: `[b, n, v]` (shared across heads)
> - `W_q`: shape `A` — query projection weights (per head)
> - `W_k`: shape `B` — key projection weights (shared)
> - `W_v`: shape `[d, v]` — value projection weights (shared)
> - `W_o`: shape `[h, v, d]` — output projection weights
>
> ---
>
> **Final answers:**
>
> **C:** `torch.einsum("bd,hdq->bhq", x, W_q)`
> **D:** `torch.einsum("bd,dk->bk", x, W_k)`
> **E:** `torch.einsum("bd,dv->bv", x, W_v)`

Notice that instead of recognizing placeholders like ___A___ to be blanks to fill, the model falsely treats these placeholders as actual variables. This caused the model to not fully answer the question. Although correct answers for A and B are mentioned by the model, it did not treat AB the same as CDE. To further study the reasons for this, I propose two hypotheses:

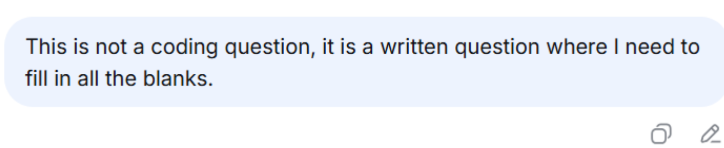
1. Because the input was text extracted from the image, there may be inaccurate extraction and thus misled the model to think A/B are different from C/D/E.
1. The model did not have enough reasoning ability to recognize the blanks from variables.

For the first hypothesis, because the model and OCR feature are closed sourced, there is no good way to actually test it. One thing I could do to test this is by providing the model with a human verified correct version of the extracted text, to make sure the model receives the correct text. However, this hypothesis is likely not true because C/D/E are no different from A/B in the image. If the OCR can correctly extract C/D/E, then it can probably also extract OCR A/B correctly. I thus want to focus on testing hypothesis 2 — will increasing the model's reasoning ability help the model to identify the correct answers to fill?
To test this, I will test the same thing on the same model, except this time with the "DeepThink" feature on. The DeepThink feature is supposed to be an enhancement on model's reasoning ability, and to make sure past context does not interfere with model's answer, I started a new chat, input same prompt:



However, the model still gives only C,D,E as the final answers. This result is not enough to help us determine whether the hypothesis is correct or not. We still don't know how much this "DeepThink" feature improved model reasoning ability, thus we can't rule out the possibility that if the model has further enhanced reasoning ability, it will be able to identify all the blanks.

Upon further analysis, I found out that the screenshot to model was in the form of python code. And the difference between A/B and C/D/E is that A/B are in the function signature, while C/D/E are in the function body. When the model is asked to "fill in the blanks", it assumes the blanks are in the function body — that's how coding works. The model assumes this is a coding question and only fills in the blanks as if this is a coding question. With this realization, I provided more context to the model and it was able to correctly fill in all the blanks:



Based on the Multi-Query Attention (MQA) pseudocode and the comparison with standard Multi-Head Attention (MHA), the blanks can be filled as follows. In MQA, keys and values are shared across all heads, which reduces memory access and improves efficiency. The queries remain per-head, similar to MHA.

- **Blank A**: Shape of `W_q`
  Since queries are per-head, `W_q` should have shape `[h, d, q]`. This allows projecting the input to queries for each head.
- **Blank B**: Shape of `W_k`
  Since keys are shared across heads, `W_k` should have shape `[d, k]`. This projects the input to keys without a head dimension.
- **Blank C**: Expression for `q` (query projection)
  The query projection uses `torch.einsum("bd,hdq->bhq", x, W_q)`. This computes queries for each head, resulting in a tensor of shape `[b, h, q]`.
- **Blank D**: Expression for `k` (key projection)
  The key projection uses `torch.einsum("bd,dk->bk", x, W_k)`. This computes keys shared across heads, resulting in a tensor of shape `[b, k]`.
- **Blank E**: Expression for `v` (value projection)
  The value projection uses `torch.einsum("bd,dv->bv", x, W_v)`. This computes values shared across heads, resulting in a tensor of shape `[b, v]`.

Note that the model only correctly identifies all the blanks with the DeepThink feature. Without strong reasoning ability, the model still is not able to recognize all blanks:

This is not a coding question, it is a written question where I need to fill in all the blanks.

4 / 4

Got it — you need the filled-in blanks for the **Multi-Query Attention** pseudocode in a written homework answer format.

C: `torch.einsum("bd,hdq->bhq", x, W_q)`
D: `torch.einsum("bd,dk->bk", x, W_k)`
E: `torch.einsum("bd,dv->bv", x, W_v)`

Explanation:

- For **C**, queries are computed per head, so `W_q` has shape `[h, d, q]`, and the einsum maps `[b, d]` and `[h, d, q]` → `[b, h, q]`.
- For **D**, keys are shared across heads, so `W_k` has shape `[d, k]`, and the einsum maps `[b, d]` and `[d, k]` → `[b, k]`.
- For **E**, values are also shared, so `W_v` has shape `[d, v]`, and the einsum maps `[b, d]` and `[d, v]` → `[b, v]`.

For the next two subparts, the model was able to one-shot correct answers:

3. Total computational complexity

Projections: $O(bd^2)$
Attention: $O(bnd)$

$$\boxed{O(bd^2 + bnd)}$$

This is the **same big-O** as MHA, but the constant factors for projections are smaller in MQA because key/value projections are shared.

4. Total memory access complexity

$$O(d^2) + O(bd) + O(bnd)$$

$$\boxed{O(d^2 + bd + bnd)}$$

This is the **same big-O** as MHA, but constants are smaller because k and v have size d/h in cache instead of d per head (MHA cache shape $[b, h, n, d/h]$ also gives $O(bnd)$ total, so same big-O).

For the last problem, I started by simply pasting the question prompt to the model and asking it one-shot the problem (answering all four subparts at the same time). The model was able to get b and c correct one shot. But the model got part a and part d wrong.

I started solving this by using a screenshot of the question instead of merely copy-pasting the text. When I naively copy the text, there are certain symbols/formulas that are somewhat malformed, and I want to make sure that doesn't affect the model's inference. Next, instead of giving all four subparts all together, I give each subpart one at a time. This is not based on anything, but is simply something I personally do and have found to be working for me while interacting with LLM's in the past. With these changes, the model was able to solve problem (d):

Final answer:

$$\boxed{O(ND^3)}$$

for computation and $O(ND^2 + D^3)$ for memory, with $P = (D+1)(D+2)/2$ and $M \approx D$.

However, it still fails to answer question (a):

Final answer:
The condition is:

$$\mathrm{sim}(Q_i, K_j) \geq 0 \quad \text{for all } i, j$$

and

$$\forall i,\ \exists j \text{ such that } \mathrm{sim}(Q_i, K_j) > 0$$

(or more simply: $\mathrm{sim}(Q_i, K_j) > 0$ for all i, j, which is the case for the exponential kernel in softmax).

The model isn't necessarily wrong, the answer is still correct, just that the model wasn't incorporating the case that the sim function value can be negative. I have tried a few methods and the model just seems to not mention the case where sim value can be negative. After closer inspection, I have made the following hypothesis: the model is not mentioning all cases because the question did not ask for the most "loose" condition. As a result, the model gives a tighter condition that not only satisfies the prompt but also ensures better numerical stability. With this hypothesis in mind, I changed the prompt a little bit: instead of asking for a condition, I ask for the "minimal" condition, and as expected, the model gives correct answer:

> Identify the MINIMAL conditions that needs to be met by the sim function to ensure that Vi
> in Equation 3
> remains finite (the denominator never reaches zero).

The denominator in Equation 3 is:

$$\sum_{j=1}^{N} \mathrm{sim}(Q_i, K_j)$$

For V_i' to remain finite, the denominator must be **nonzero** for all i.

The **minimal** condition is:

$$\sum_{j=1}^{N} \mathrm{sim}(Q_i, K_j) \neq 0 \quad \forall i \in \{1, \ldots, N\}$$

This means for each query Q_i, there must be **at least one key** K_j such that $\mathrm{sim}(Q_i, K_j) \neq 0$.
It does **not** require $\mathrm{sim}(q, k) \geq 0$ for all q, k, only that the sum over j is nonzero.

Final answer:

$$\boxed{\sum_{j=1}^{N} \mathrm{sim}(Q_i, K_j) \neq 0 \quad \forall i}$$

Summary:

To sum up, the model (DeepSeek-V3.2) is a quite strong model and is able to solve most of the non coding questions correctly (one-shot). For this specific problem set, two things matter the most for accuracy: 1. Model’s reasoning ability and 2. Correct prompt and context. Prompt and context are very important because if instructions and context are not explicitly told, the model will make certain assumptions, and thus give incorrect answers. For example in question 4, when not stated explicitly, the model thinks it is completing some code function instead of filling blanks for a written question, thus identifying the wrong blanks to fill. The model’s reasoning ability is also very important : for the previous example, even if the correct context and prompt is provided, the model still was not able to identify the correct blanks to fill without the DeepThink feature on. Together, these observations suggest that while DeepSeek-V3.2 is already quite capable on non-coding questions, its performance is highly sensitive to both how we phrase the task and model’s reasoning capabilities. Thus, carefully designing prompts and systematically leveraging features like DeepThink will be essential for reliably getting correct answers and understanding the model's core strengths and weaknesses.