Final corrected answers for Problem 1:

- **(a)** $\mu^T\mu$ (or $\|\mu\|^2$, or $\mu \cdot \mu$)
- **(b)** d
- **(c)** $\sqrt{d}$

Thanks again for the sharp catch — this is a really common trick in homework problems: they first give the general case with vector mean to make you think carefully, then specialize to μ=0 later. Easy to miss on a quick read!

Ready for Problem 2 (argmax attention) whenever you are!

> That's great! Next, let's take a look at problem 2. In this problem, we consider a different form of attention that is performed as an argmax, rather than a traditional softmax. How would we answer question 2 parts a and b?

Problem 2: Argmax Attention

Part (a) – Compute the actual output

We are doing **hard argmax attention** (no softmax, no temperature, no scaling mentioned → raw dot product).

Query:
$q = [1, 1, 2]^T$

Keys (columns):
$k_1 = [1, 2, 0]^T$
$k_2 = [0, 3, 4]^T$
$k_3 = [5, 0, 0]^T$
$k_4 = [0, 0, 1]^T$

Compute dot products q^Tk_i:

$q^Tk_1 = 1\cdot1 + 1\cdot2 + 2\cdot0 = 1 + 2 + 0 = \mathbf{3}$
$q^Tk_2 = 1\cdot0 + 1\cdot3 + 2\cdot4 = 0 + 3 + 8 = \mathbf{11}$
$q^Tk_3 = 1\cdot5 + 1\cdot0 + 2\cdot0 = 5 + 0 + 0 = \mathbf{5}$
$q^Tk_4 = 1\cdot0 + 1\cdot0 + 2\cdot1 = 0 + 0 + 2 = \mathbf{2}$

Scores: [3, 11, 5, 2] → the maximum is 11 at position 2 (second key).

Argmax therefore selects the **second value vector** only (one-hot at index 1 if 0-indexed).

Corresponding values (again columns):
$v_1 = [2, 0, 1]^T$
$v_2 = [1, 4, 3]^T$ ← this one is chosen
$v_3 = [0, -1, 4]^T$
$v_4 = [1, 0, -1]^T$

So the output of argmax attention is exactly v_2:

Answer (a):

$$\begin{bmatrix}1\\4\\3\end{bmatrix}$$

Part (b) – Why is argmax attention basically untrainable with gradient descent?

Even though argmax looks appealingly simple, it destroys gradient flow.

- The argmax operation is **almost everywhere discontinuous** and its gradient is **zero almost everywhere** (subgradient contains zero).
- For any fixed query q and keys $\{k_i\}$, small changes in the parameters that affect q or the k_i's will **not** change the identity of the winning key unless they cross the decision boundary.
- When the correct key is already the winner, the gradient of the loss with respect to all queries and keys is **exactly zero** (because moving them a tiny bit keeps the same key selected).
- Only on the razor-thin manifold where two keys are exactly tied do you get a non-zero subgradient, but that has measure zero and is never hit in practice with floating-point numbers.

Consequences:

- The model gets **no learning signal** telling it how to make queries or keys better when the current choice is already the best one.
- It cannot learn smooth, distributed representations or subtle similarities — it's forced into hard, discrete decisions with no gradient-based way to improve.
- In practice, training with argmax attention completely fails to converge to anything useful (people have tried variants like straight-through estimator or REINFORCE, but they're unstable and far worse than softmax).

Softmax fixes this by being **smooth and everywhere differentiable**, giving non-zero gradients to **all** keys (weighted by attention probability), so the model can gradually shift mass toward better keys and away from worse ones.

Answer (b) (short version you can write):

Argmax is not differentiable (its gradient is zero almost everywhere). When the current key is already the most similar, gradients through the attention scores are exactly zero, so the model receives no signal to improve queries or keys. Softmax, by contrast, is smooth and provides non-zero gradients to all keys proportional to their attention weights, enabling effective end-to-end training.

That's it for Problem 2!

Let me know when you're ready for Problem 3 (the PyTorch multi-head attention fill-in-the-blanks) — it's a long one but very mechanical.

↳ Explain straight-through estimator for argmax

↳ Gumbel-softmax for discrete attention