Homework 2: Self-Attention and Transformers — Solutions

CS 388: Natural Language Processing • Fall 2026

One-hot embeddings in $\mathbb{R}^3$, maximum length $L = 5$, query taken from the final position, and $\hat{y}(x) = \arg\max_\ell o_\ell$ with ties counted as wrong. In Questions 1 and 2 we have $W_K = W_V = I_3$, so that $s_i = (W_Q)_{ab}$ where a indexes the final symbol x_n and b indexes the attended symbol x_i. In every matrix below, rows and columns are indexed `A`, `B`, `C` in that order, and useful constants are $e^2 \approx 7.389$ and $e^4 \approx 54.598$.

Students are given the decomposition $o_\ell = \sum_{i:x_i=\ell} \alpha_i$ (each coordinate of o is the total attention mass on positions holding that symbol) and the fact that o is therefore a probability distribution over Σ. Both are used constantly below; the second is what lets the readout in 2(c) get away without a bias vector.

A note on grading. Many different matrices work. What matters is (i) that the score gap between the "target" symbol and the others is large enough for the *worst* sequence of length 5, not just for the short examples, and (ii) that the student checks the count-heavy cases. A construction with a gap that works on `AAB` but fails on `AAAAB` should not receive full credit.

1. Question 1

(a) A matrix W_Q capturing R1 and R2.

Solution. Give an attended `B` a score of 2 and an attended `A` a score of 0, from either query row. With no $\sqrt{d}$ in the way, the entries of W_Q *are* the scores:

$$W_Q = \begin{array}{c|ccc} & \texttt{A} & \texttt{B} & \texttt{C} \\ \hline \texttt{A} & 0 & 2 & 0 \\ \texttt{B} & 0 & 2 & 0 \\ \texttt{C} & 0 & 0 & 0 \end{array}$$

Row `C` and column `C` are never used here (no `C` occurs, and $x_n \neq$ `C`), so they may be anything; 0 is fine. Rows `A` and `B` are identical, so the last symbol does not affect the scores – only *what* is attended to does. The scores are therefore $s_i = 2$ if $x_i =$ `B` and $s_i = 0$ if $x_i =$ `A`.

Why 2 is large enough. By the decomposition given in the setup, $o_{\texttt{B}}/o_{\texttt{A}} = n_{\texttt{B}}e^2/n_{\texttt{A}}$, where $n_{\texttt{A}}, n_{\texttt{B}}$ are the symbol counts. R2 holds iff this exceeds 1, i.e. iff $2 > \ln(n_{\texttt{A}}/n_{\texttt{B}})$. The hardest case allowed by $L = 5$ is a single `B` among four `A`'s, which needs $2 > \ln 4 \approx 1.386$ ✓. Concretely, for `AAAAB`,

$$o_{\texttt{A}} = \tfrac{4}{4+e^2} \approx 0.351, \qquad o_{\texttt{B}} = \tfrac{e^2}{4+e^2} \approx 0.649,$$

so `B` still wins. R1 is immediate: with no `B` present, $o_{\texttt{B}} = 0$ and $o_{\texttt{A}} = 1$. Any gap $\Delta > \ln 4 \approx 1.386$ works, i.e. any entry larger than that in the `B` column. A gap of, say, 1 would be *wrong*: for `AAAAB` it gives $o_{\texttt{B}} \propto e \approx 2.72 < 4$.

(b) Verification on `AAA`, `AAB`, and `ABA`.

Solution. Write $Z = \sum_j \exp(s_j)$.

x	$(s_1, \ldots, s_n)$	Z	α	$o = (o_{\texttt{A}}, o_{\texttt{B}}, o_{\texttt{C}})$	$\hat{y}(x)$
`AAA`	$(0, 0, 0)$	3	$(\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$	(1, 0, 0)	`A` (R1 ✓)
`AAB`	$(0, 0, 2)$	$2 + e^2 \approx 9.389$	(0.107, 0.107, 0.787)	(0.213, 0.787, 0)	`B` (R2 ✓)
`ABA`	$(0, 2, 0)$	$2 + e^2 \approx 9.389$	(0.107, 0.787, 0.107)	(0.213, 0.787, 0)	`B` (R2 ✓)

AAB and ABA give the same o: attention is a sum over positions, so only the *multiset* of symbols matters (this is the fact Question 3 turns into a limitation).

2. Question 2

(a) No single W_Q captures R1–R4.

Solution. Both CA and BCA end in A, so both read their scores off row A of W_Q. Write

$$s_{\mathtt{A}} = (W_Q)_{\mathtt{AA}}, \qquad s_{\mathtt{B}} = (W_Q)_{\mathtt{AB}}, \qquad s_{\mathtt{C}} = (W_Q)_{\mathtt{AC}}$$

for the three scores available in that row. These three numbers are the *same* for both sequences. Using the decomposition from the setup, and writing Z for the (positive) normalizer of each sequence:

x	rule	requirement	inequality
CA	R3 (no B, has C)	$o_{\mathtt{A}} > o_{\mathtt{C}}$	$\frac{1}{Z}e^{s_{\mathtt{A}}} > \frac{1}{Z}e^{s_{\mathtt{C}}} \iff s_{\mathtt{A}} > s_{\mathtt{C}}$
BCA	R4 (has B, has C)	$o_{\mathtt{C}} > o_{\mathtt{A}}$	$\frac{1}{Z}e^{s_{\mathtt{C}}} > \frac{1}{Z}e^{s_{\mathtt{A}}} \iff s_{\mathtt{C}} > s_{\mathtt{A}}$

Both sequences contain exactly one A and exactly one C, so the counts multiplying the exponentials are 1 in all four places and cancel; the normalizers cancel within each row. We are left with $s_{\mathtt{A}} > s_{\mathtt{C}}$ and $s_{\mathtt{C}} > s_{\mathtt{A}}$, which is impossible. (The inequalities are strict because a tie counts as a wrong answer; note also that $s_{\mathtt{B}}$ never enters, since the argument only compares the A and C coordinates.) ■

(b) Why this happens.

Solution. Each score s_i depends only on two things: the final symbol x_n, which selects the row of W_Q, and the symbol x_i at the attended position. It does *not* depend on what else is in the sequence. So by the decomposition above, the ratio

$$\frac{o_{\mathtt{A}}}{o_{\mathtt{C}}} = \frac{n_{\mathtt{A}}\, e^{s_{\mathtt{A}}}}{n_{\mathtt{C}}\, e^{s_{\mathtt{C}}}}$$

depends only on the counts of A and C and on fixed scores – the shared softmax normalizer cancels. Inserting a B changes the normalizer but leaves this ratio untouched, so it can never flip the A vs. C comparison. But R3 and R4 demand exactly that flip: whether C beats A is supposed to depend on the presence of a *third* symbol. A single head scores each key independently, so it can weigh symbols against each other but cannot represent a *conjunction* of two separate features.

(c) Two heads that do capture R1–R4.

Solution. *The heads.* Head 1 gives an attended B a score of 4 and everything else 0; head 2 does the same for C. Every row is identical, so the construction does not care what the final symbol is:

$$W_Q^{(1)} = \begin{array}{c|ccc} & \mathtt{A} & \mathtt{B} & \mathtt{C} \\ \hline \mathtt{A} & 0 & 4 & 0 \\ \mathtt{B} & 0 & 4 & 0 \\ \mathtt{C} & 0 & 4 & 0 \end{array} \qquad W_Q^{(2)} = \begin{array}{c|ccc} & \mathtt{A} & \mathtt{B} & \mathtt{C} \\ \hline \mathtt{A} & 0 & 0 & 4 \\ \mathtt{B} & 0 & 0 & 4 \\ \mathtt{C} & 0 & 0 & 4 \end{array}$$

The two coordinates of interest are $h_1 = o_{\mathtt{B}}^{(1)}$ and $h_2 = o_{\mathtt{C}}^{(2)}$.

They are near-binary indicators. If B does not occur, $h_1 = 0$ *exactly*, since the sum defining it is empty. If it does, then with $n_B \geq 1$ and at most 4 other positions,

$$h_1 = \frac{n_B e^4}{n_B e^4 + (n - n_B)} \geq \frac{e^4}{e^4 + 4} = \frac{54.598}{58.598} \approx 0.932 > \tfrac{3}{4},$$

the worst case being one B among four non-B's. So $h_1 \in \{0\} \cup [0.932, 1]$, and likewise h_2. Any score gap $\lambda > \ln 12 \approx 2.485$ would do; $\lambda = 4$ is comfortable. Neither indicator can equal 1 exactly: the softmax gives every position strictly positive weight, so as long as some non-B symbol is present, $h_1 < 1$. Saturation is all we can ask for, which is why the readout below is built with margins rather than exact values.

The readout. We want logits $z_A = 1 - 2h_1$, $z_B = h_1 - 2h_2$, and $z_C = h_1 + 2h_2 - 2.5$. There is no bias vector, but we do not need one: each head's output sums to 1, so the constant 1 is available as $o_A^{(1)} + o_B^{(1)} + o_C^{(1)}$. Substituting that for the constants gives

$W_O =$

	A	B	C
$o_A^{(1)}$	1	0	−2.5
$o_B^{(1)}$	−1	1	−1.5
$o_C^{(1)}$	1	0	−2.5
$o_A^{(2)}$	0	0	0
$o_B^{(2)}$	0	0	0
$o_C^{(2)}$	0	−2	2

For instance column C reads $z_C = -2.5\, o_A^{(1)} - 1.5\, o_B^{(1)} - 2.5\, o_C^{(1)} + 2\, o_C^{(2)} = h_1 + 2h_2 - 2.5$, using $o_A^{(1)} + o_B^{(1)} + o_C^{(1)} = 1$.

Verification on all four cases, using $h \in \{0\} \cup [0.932, 1]$ and one worked example each:

rule	example x	z_A	z_B	z_C	$\hat{y}(x)$
R1 ($h_1 = 0,\ h_2 = 0$)	AAA	1	0	−2.5	A ✓
R2 ($h_1 \geq 0.932,\ h_2 = 0$)	AB	−0.964	0.982	−1.518	B ✓
R3 ($h_1 = 0,\ h_2 \geq 0.932$)	AC	1	−1.964	−0.536	A ✓
R4 ($h_1, h_2 \geq 0.932$)	AAABC	−0.863	−0.932	0.295	C ✓

The bounds hold in general, not just on these examples: in R2, $z_B = h_1 \geq 0.932$ while $z_A = 1 - 2h_1 \leq -0.863$ and $z_C = h_1 - 2.5 \leq -1.5$; in R3, $z_A = 1$ while $z_C = 2h_2 - 2.5 \leq -0.5$ and $z_B = -2h_2 \leq -1.86$; in R4, $z_C = h_1 + 2h_2 - 2.5 \geq 0.295$ while $z_A \leq -0.863$ and $z_B = h_1 - 2h_2 \leq 1 - 1.86 = -0.863$.

The saturation bound is what makes this work for *every* sequence of length ≤ 5. With a gap of only $\lambda = 2$, the sequence AAABC gives $h_1 = h_2 = e^2/(e^2 + 4) \approx 0.649$, hence $z_A \approx -0.298$ and $z_C \approx -0.554$: the model answers A and violates R4.

(d) Why no non-linearity is needed.

Solution. Because the non-linearity has already happened, inside the heads. The hard part of R1–R4 is turning "does a B appear *anywhere* in the sequence" into a number, and that is what each head's softmax does: it is a non-linear function of the input that saturates to an indicator. Once the two heads have produced h_1 and h_2, the remaining job is a map from (approximately) $\{0, 1\}^2$ to three labels, and that map is linearly separable. The three decision regions

$$\text{A} : h_1 \approx 0, \qquad \text{B} : h_1 \approx 1,\ h_2 \approx 0, \qquad \text{C} : h_1 \approx 1,\ h_2 \approx 1$$

are convex and pairwise separable by straight lines, so a three-way linear classifier can carve them out. The usual "conjunctions need a non-linearity" intuition comes from XOR, where the two positive corners lie on a diagonal and no single line separates them; AND is not like that. What we could *not* do is compute the conjunction with one head and a linear readout (part 2(a)) – multi-head attention is doing the feature construction that an MLP would otherwise have to do.

3. Question 3

(a) Can attention, as defined so far, capture R5?

Solution. No. The output $o = \sum_i \alpha_i v_i$ is a sum over positions, and both the sum and the softmax normalizer are unchanged if we permute the positions. The only positional information the model has is which symbol sits at the final position, because that is where the query comes from. So `CAA` and `ACA`, which have the same multiset of symbols $\{\texttt{A}, \texttt{A}, \texttt{C}\}$ *and* the same final symbol, produce an identical q, an identical collection of (k_i, v_i) pairs, and therefore an identical o – yet R5 wants `C` for the first and `A` for the second. This holds for any W_Q, W_K, W_V, and extra heads do not help: each head is individually permutation-invariant, so any readout of their concatenation is too.

(b) Fixing it with an integer positional encoding.

Solution. Take $\tilde{e}_i = [\, e(x_i),\, i \,]$ and make the query a constant vector that points in the *negative* position direction, so that earlier positions score higher. Let

$$W_Q = \begin{pmatrix} 0 & 0 & 0 & -2 \\ 0 & 0 & 0 & -2 \\ 0 & 0 & 0 & -2 \\ 0 & 0 & 0 & 0 \end{pmatrix}, \qquad W_K = I_4 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \qquad W_V = I_4.$$

The first three rows of W_Q each contribute -2 to the last coordinate and exactly one of them is selected by the one-hot part of $\tilde{e}_n$; the fourth row is 0 so the index n itself contributes nothing. Hence $q = \tilde{e}_n W_Q = [0, 0, 0, -2]$ no matter what the last symbol is or how long the sequence is. With $W_K = I_4$ we get $k_i = [\, e(x_i),\, i \,]$, so the one-hot part of k_i is killed by the zeros in q and

$$s_i \;=\; q\, k_i^\top \;=\; -2i, \qquad \alpha_i \;=\; \frac{e^{-2i}}{\sum_{j=1}^{n} e^{-2j}}.$$

The scores depend only on position, so attention is a fixed, steeply decaying profile. For $n = 5$:

i	1	2	3	4	5
s_i	-2	-4	-6	-8	-10
α_i	0.865	0.117	0.016	0.002	0.000

With $W_V = I_4$ the values are the embeddings themselves, so by the same decomposition as before the first three coordinates of o are $o_\ell = \sum_{i:x_i=\ell} \alpha_i$, and position 1 contributes $\alpha_1 \approx 0.865$ to the coordinate of x_1. The fourth coordinate of o is $\sum_i \alpha_i\, i$ and is simply ignored.

Why "more than half". The mass not on position 1 is $1 - \alpha_1$, and it is the most that any *other* coordinate could possibly collect. So if $\alpha_1 > \frac{1}{2} > 1 - \alpha_1$, the coordinate of x_1 is strictly the largest no matter how the remaining symbols are arranged, and $\hat{y}(x) = x_1$ — which is exactly R5 on the alphabet $\{\texttt{A}, \texttt{C}\}$.

Why -2 is large enough. Dividing through by e^{-2},

$$\alpha_1 = \frac{1}{1+e^{-2}+e^{-4}+e^{-6}+e^{-8}} = \frac{1}{1.1565} \approx 0.865 > \tfrac{1}{2}.\checkmark$$

In general a gap of λ per position needs $\sum_{k=1}^{L-1} e^{-\lambda k} < 1$, for which $\lambda > \ln(L-1) = \ln 4 \approx 1.386$ is sufficient; $\lambda = 2$ clears it.

The counterexample from (a) is now resolved. For `CAA` and `ACA` we have $n = 3$, so $\alpha \approx (0.867, 0.117, 0.016)$ and:

x	$o_{\texttt{A}}$	$o_{\texttt{C}}$	$\hat{y}(x)$
`CAA`	$0.117 + 0.016 = 0.133$	0.867	`C` ✓
`ACA`	$0.867 + 0.016 = 0.883$	0.117	`A` ✓

The two sequences are finally distinguishable, because the keys now carry position and the query asks a question about position rather than about identity.

4. Question 4

(a) The mistake.

Solution. In `forward`, the line that is supposed to recombine the heads reads

```
combined = head_outputs[:, 0]
```

`head_outputs` has shape `(B, n_heads, T, d_head)`, so indexing `[:, 0]` keeps head 0 and silently throws away heads $1, \ldots,$ `n_heads`-1. That line was supposed to *concatenate* all of the heads along the feature axis, rebuilding one `d_model`-dimensional vector per position. (The comment directly above it – "recombine the heads" – is the giveaway that this is the intended step.)

(b) What happens if it is not fixed.

Solution. There are two layers to this answer, and full credit requires the second one.

The immediate symptom. `combined` comes out with shape `(B, T, d_head)` instead of `(B, T, d_model)`, but `W_O` is a `Linear(d_model, d_model)`. The very next line therefore tries to multiply a $T \times$ `d_head` matrix against a `d_model` $\times$ `d_model` one. With the settings in `__main__` (`d_model` $= 8$, `n_heads` $= 2$, so `d_head` $= 4$) running the file prints a traceback ending in

```
RuntimeError:  mat1 and mat2 shapes cannot be multiplied (5x4 and 8x8)
```

where 5×4 is `(T, d_head)` and 8×8 is the weight of `W_O`. Worth noticing: if `n_heads` $= 1$ then `d_head` $=$ `d_model`, the line is a harmless no-op, and the layer is perfectly correct — which is exactly how a bug like this survives a single-head unit test and reaches production.

The real problem. Suppose someone made the error go away by resizing the output projection to `Linear(d_head, d_model)`. The code would then run silently on any number of heads, and the layer would be a single-head attention layer wearing a multi-head costume. `W_Q`, `W_K` and `W_V` would still compute `n_heads` separate sets of projections, and all of those attention distributions would still be computed, but only head 0 would ever reach the output. The parameters belonging to heads $1, \ldots$ would receive *zero gradient* and never train, and none of their attention patterns would influence anything downstream.

By Question 2(a) this is precisely fatal for the rules this assignment is about: a single head cannot capture R1–R4, because the ratio $o_{\texttt{A}}/o_{\texttt{C}}$ cannot be made to depend on whether a `B` is present. So the model would quietly lose the ability to represent a conjunction of two features — the one capability that motivates multi-head attention in the first place — while reporting no error at all, training happily, and simply converging to a worse solution. Silent capacity loss is much harder to catch than a crash.

(c) A one-line fix.

Solution. Replace the offending line with

```
combined = head_outputs.transpose(1, 2).reshape(B, T, self.d_model)
```

This transposes back to `(B, T, n_heads, d_head)` and then flattens the last two axes, which is exactly the per-position concatenation of the heads. Two details worth a remark when grading:

- `.reshape` is needed rather than `.view`. The `transpose` leaves the tensor non-contiguous, so `.view` would raise its own error; `.reshape` copies when it has to.
- The transpose is not optional. Reshaping `head_outputs` directly would interleave the heads with the time axis and scramble the sequence.

Equivalent one-liners are fine — for instance

```
combined = torch.cat([head_outputs[:, h] for h in range(self.n_heads)], dim=-1)
```

spells the concatenation out literally.