CSCI 145 Problem 20: The Differentiable Dictionary

July 25, 2026

Submission Instructions

Please upload your work to Gradescope by **11:59pm on DATE**.

- Turn in a single PDF of *handwritten* solutions. Typed (including LaTeX) solutions will not be accepted.
- Problems are grouped into three-week **cycles**. Once per cycle, you will meet with the grader and present the solution to *one* problem from that cycle, selected at random when you arrive — so bring your work and be ready to explain any of them.

Problem 20: The Differentiable Dictionary

You have programmed dictionaries since your first week of Python: `d = {"france": "paris", "japan": "tokyo"}`. Lecture treated attention weights as something gradient descent discovers; this problem runs the arrow the other way. Instead of analyzing weights someone else trained, *you* will choose $\mathbf{W}_Q$, $\mathbf{W}_K$, and $\mathbf{W}_V$ by hand and program an attention layer to *be* a dictionary. The mechanism that makes the lookup exact is the very saturation lecture worked to avoid.

Throughout, the vocabulary has m token types, and token type t has the **one-hot embedding** $\mathbf{e}_t \in \mathbb{R}^m$: a 1 in coordinate t and zeros elsewhere, so distinct types have orthonormal embeddings. (In lecture's notation this is the embedding matrix $\mathbf{E} = \mathbf{I}_m$, so the embedding dimension is $d = m$.) The input is a sequence of types $t_1, \ldots, t_n \in \{1, \ldots, m\}$ with embeddings $\mathbf{x}_i = \mathbf{e}_{t_i}$, and as in lecture, $\mathbf{q}_i = \mathbf{W}_Q^\top \mathbf{x}_i$, $\mathbf{k}_i = \mathbf{W}_K^\top \mathbf{x}_i$, and $\mathbf{v}_i = \mathbf{W}_V^\top \mathbf{x}_i$. In place of lecture's $1/\sqrt{d_k}$, we scale the scores by a knob $\beta > 0$ that we control:

$$a_{ij} = \frac{e^{\beta \langle \mathbf{q}_i, \mathbf{k}_j \rangle}}{\sum_{l=1}^{n} e^{\beta \langle \mathbf{q}_i, \mathbf{k}_l \rangle}}, \qquad \mathbf{o}_i = \sum_{j=1}^{n} a_{ij}\, \mathbf{v}_j.$$

Finally, fix a **matching** π: a permutation of $\{1, \ldots, m\}$ pairing each token type with the type it should look up (π(france) = paris, say).

Part A: Programming the Scores

Choose $\mathbf{W}_K = \mathbf{I}_m$ and let $\mathbf{W}_Q \in \mathbb{R}^{m \times m}$ be the matrix whose t-th row is $\mathbf{e}_{\pi(t)}^\top$. Show that with this choice $\mathbf{q}_i = \mathbf{e}_{\pi(t_i)}$ and $\mathbf{k}_j = \mathbf{e}_{t_j}$, and conclude that for *any* input sequence the scores are:

$$\langle \mathbf{q}_i, \mathbf{k}_j \rangle \;=\; \begin{cases} 1 & \text{if } t_j = \pi(t_i), \\ 0 & \text{otherwise.} \end{cases}$$

In words, every query scores 1 exactly on its matched key and 0 everywhere else.

Now make it concrete: with vocabulary $(1, 2, 3, 4) = (\text{france}, \text{paris}, \text{japan}, \text{tokyo})$ and the matching π that swaps $1 \leftrightarrow 2$ and $3 \leftrightarrow 4$, write out $\mathbf{W}_Q$ as a 4×4 matrix and compute the 4×4 score matrix $[\langle \mathbf{q}_i, \mathbf{k}_j \rangle]_{i,j}$ for the input sequence (france, paris, japan, tokyo).

Part B: The Sharpness Knob

Suppose token i's matched type appears *exactly once* in the sequence, at position $j^\star$. Using Part A's scores, derive the attention weights in closed form:

$$a_{ij^\star} = \frac{e^{\beta}}{e^{\beta} + (n-1)}, \qquad a_{ij} = \frac{1}{e^{\beta} + (n-1)} \quad \text{for } j \neq j^\star.$$

Evaluate the weight on the match at $\beta = 0$, at $\beta = \ln 27$ with $n = 4$, and in the limit $\beta \to \infty$, and describe the three regimes in words: uniform blur, soft lookup, exact (hard) lookup. This is the class demo's saturation, now working *for* us: lecture divided by $\sqrt{d_k}$ precisely to keep attention out of the regime this problem dials it into. One caution, in one sentence: using the softmax Jacobian derived in class, what happens to the gradient of this layer as $\beta \to \infty$, and why does that make a perfectly programmed dictionary perfectly untrainable?

Part C: Returning the Value

A dictionary must return something. Store a value vector $\mathbf{u}_t \in \mathbb{R}^{d_v}$ for each token type by choosing $\mathbf{W}_V \in \mathbb{R}^{m \times d_v}$ with t-th row $\mathbf{u}_t^\top$, so that $\mathbf{v}_j = \mathbf{u}_{t_j}$. Show that in the setting of Part B the output has the closed form:

$$\mathbf{o}_i = \frac{e^\beta \, \mathbf{u}_{\pi(t_i)} + \sum_{j \neq j^\star} \mathbf{u}_{t_j}}{e^\beta + (n-1)} \xrightarrow{\beta \to \infty} \mathbf{u}_{\pi(t_i)}.$$

So the output converges to the matched token's stored value. In the concrete instance of Part A with $\mathbf{u}_t = \mathbf{e}_t$, what vector does the query of "france" return in the limit? Conclude in one sentence: attention *is* a soft lookup table, and hard lookup is its $\beta \to \infty$ (fully saturated) limit.

Part D*: Two Layers That Complete a Pattern

Real transformers chain lookups. Suppose a first attention layer has already produced, at every position i, the *concatenated* embedding

$$\tilde{\mathbf{x}}_i = \begin{bmatrix} \mathbf{e}_{t_i} \\ \mathbf{e}_{t_{i-1}} \end{bmatrix} \in \mathbb{R}^{2m},$$

the current token stacked on top of the *previous* token (take $\mathbf{e}_{t_0} = \mathbf{0}$ at the first position). Building this "previous-token head" requires knowing which token is previous, i.e., positional information; it arrives two lectures from now, so assume it here. Design a second attention layer over the $\tilde{\mathbf{x}}_i$: give block matrices $\mathbf{W}_Q, \mathbf{W}_K \in \mathbb{R}^{2m \times m}$ and $\mathbf{W}_V \in \mathbb{R}^{2m \times m}$, built from $\mathbf{I}_m$ and $\mathbf{0}$, so that (i) the query of position i is $\mathbf{e}_{t_i}$, (ii) the key of position j is $\mathbf{e}_{t_{j-1}}$, and (iii) the value of position j is $\mathbf{e}_{t_j}$. Explain in a sentence which positions i attends to as $\beta \to \infty$, then trace the circuit on the sequence (A, B, A): show the final token attends to position 2 and outputs $\mathbf{e}_B$. The circuit has looked up *what followed A last time*, so it predicts that B comes next, completing the pattern $A\,B\,A \to B$. This two-layer circuit is called an **induction head**, and interpretability researchers have found exactly this structure, learned from data, inside real trained language models.

Where this goes next. You just programmed a transformer component with pencil and paper, no gradient descent required: soft lookup is what attention is, with β interpolating between a blurry average and an exact dictionary. The knob you dialed has an official name: β is an inverse *temperature*, and it returns next lecture when we control how confidently a transformer samples its next token. Part D leans on the one thing self-attention cannot supply by itself, position, which is the course's structure thread and the subject of the lecture after next; induction heads built this way are the leading candidate for how large language models learn patterns from their own context window.