Problem

Given a sequence of integers, determine whether the sequence has any pair of numbers which are consecutive.

Example

Input: [1 6 1 8 2]
Output: [1 0 1 0 1]

Explanation: In the input, 1 and 2 are consecutive so the output is 1 for each instance of 1 and 2 in the input. Neither 5, 7, nor 9 are present in the input so 6 and 8 are marked as 0.

Setup

Each element in the input vocabulary is a separate token. For example, '251' represents a single token. For a particular model, the size of the vocabulary is fixed.

The input is fed to a single layer transformer encoder with bidirectional, single-head attention and no feedforward layer. The output dimension of this attention layer is set to 1. The single logit per position is fed into a sigmoid to predict the output at the same position with cross entropy loss.

We will refer to the following quantities in analysis:

- d_{vocab}: vocab size
- d_{head}: head dimension
- d_{model}: embedding dimension
- seq_{length}: length of input sequence
- q_i: query vector for position i
- k_i: key vector for position i
- v_i: value vector for position i

We track two metrics besides loss:

- acc_{soft}: frequency of output tokens that are correct.
- acc_{hard}: frequency of entire inputs that are correct.

Overview

I had an initial hypothesis for how the transformer might solve this problem which turned out to be completely wrong–the transformer had a far more elegant, efficient design!

Analysis

Hypothesis 1a: If $d_{head} \geq d_{vocab}$, the attention head will learn the following scheme:

$q_i = [0 \dots 101 \dots 0]$, where the 1 is in position $i-1$ if $i > 1$ and $i+1$ if $i < d_{vocab} - 1$
$k_i = [0 \dots 1 \dots 0]$, where the 1 is in position i.
$v_i = [0 \dots 1 \dots 0]$, where the 1 is in position i

Note that there's plenty of freedom in the construction of v_i since its only constraint is that it should uniquely represent i. In this particular problem statement, we can be even looser since a position only needs to return the existence of the neighbor and not the neighbors value.

Now an input value x will only attend to positions where the input value is $x-1$ and $x+1$. This attention scheme serves as an OR condition to read in a value vector when either $x-1$ OR $x+1$ is in the input.

Hypothesis 1b: If $d_{head} < d_{vocab}$, the network does not have enough dimensions to encode the input in the scheme proposed in hypothesis 1a, so it must sacrifice precision. Previous research has shown that superposition does not occur without sparsity and a nonlinearity, so the network will fail to achieve 100% accuracy.

Observation 1: The network cannot solve the problem even for $d_{head} \geq d_{vocab}$!

We train with $d_{vocab} = 16$, $d_{head} = 32$, $d_{model} = 128$, and $seq_{length} = 5$. We run across 4 random seeds and choose the best training run. The training parameters are provided in the Appendix.

The best model achieves $soft_{acc} = 85\%$ and $hard_{acc} = 46\%$.

We proposed a theoretical construction of the network in Hypothesis 1a that should solve this problem, so why can't the network learn this solution?

Hypothesis 1c: The issue is that every token has to attend to *something* even when there are no consecutive values in the input. We can fix this by introducing a dummy token in the input that tokens can attend to.

Observation 2: By adding a dummy token to the start of each sequence, the network can solve the problem with 100% accuracy.

We train with the exact same parameters above, and in all four seeds, the network now achieves $soft_{acc} = 100\%$ and $hard_{acc} = 100\%$.

While we did expect the network should learn the solution after the dummy token modification, the following observation demonstrates that the mechanism Hypothesis 1a proposed is wrong.

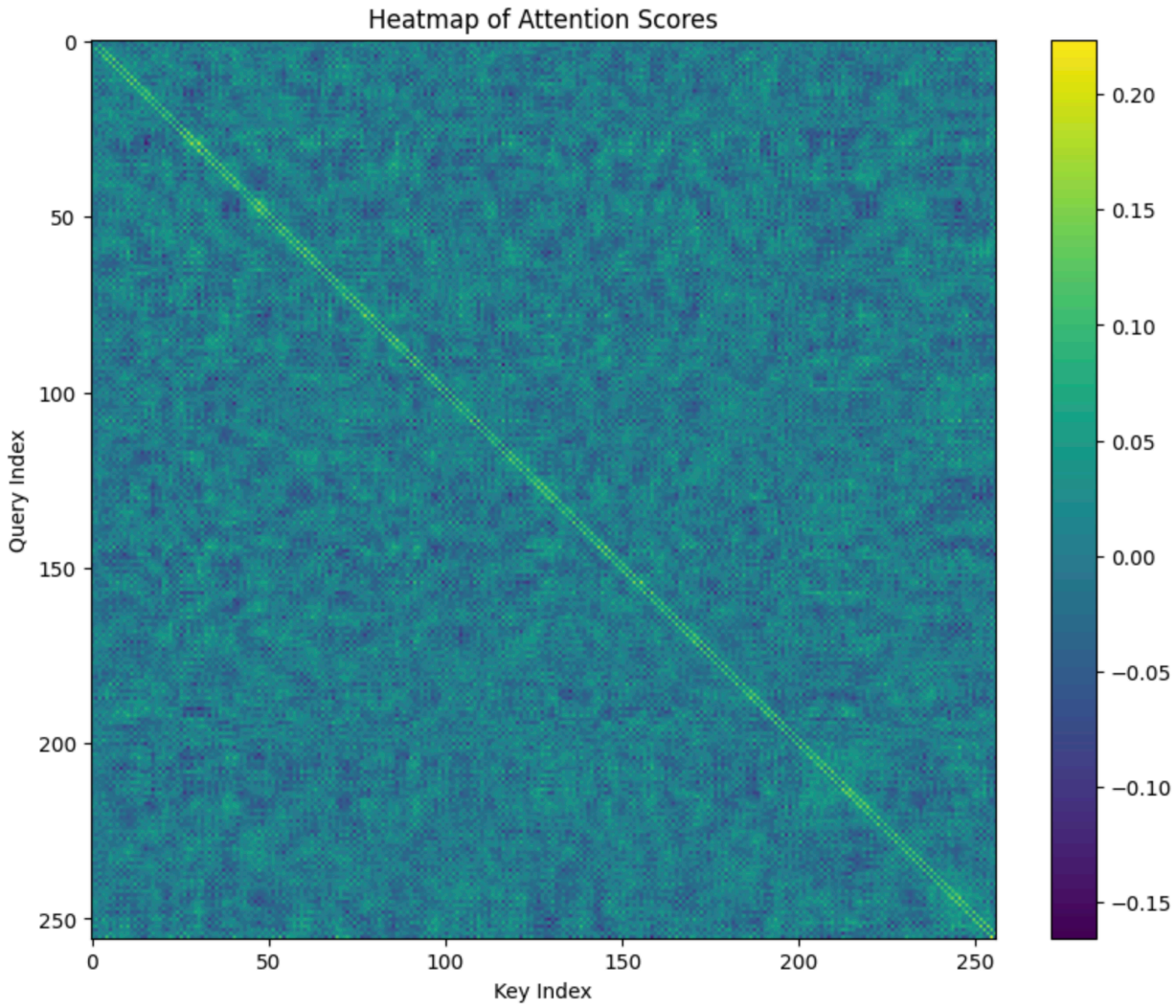
Observation 3: The network can solve the problem even when $d_{head} < d_{vocab}$, contradicting Hypothesis 1b.

We train with $d_{vocab} = 256$, $d_{head} = 32$, $d_{model} = 128$, and $seq_{length} = 5$. Due to instability in training, we train across 8 seeds for 4000 epochs and continue training the model with lowest loss for 30000 epochs.

The network achieves $soft_{acc} = 100\%$ and $hard_{acc} = 100\%$. Note the network achieved 100% accuracy quickly but we continue training since it is easier to inspect a lower loss model.

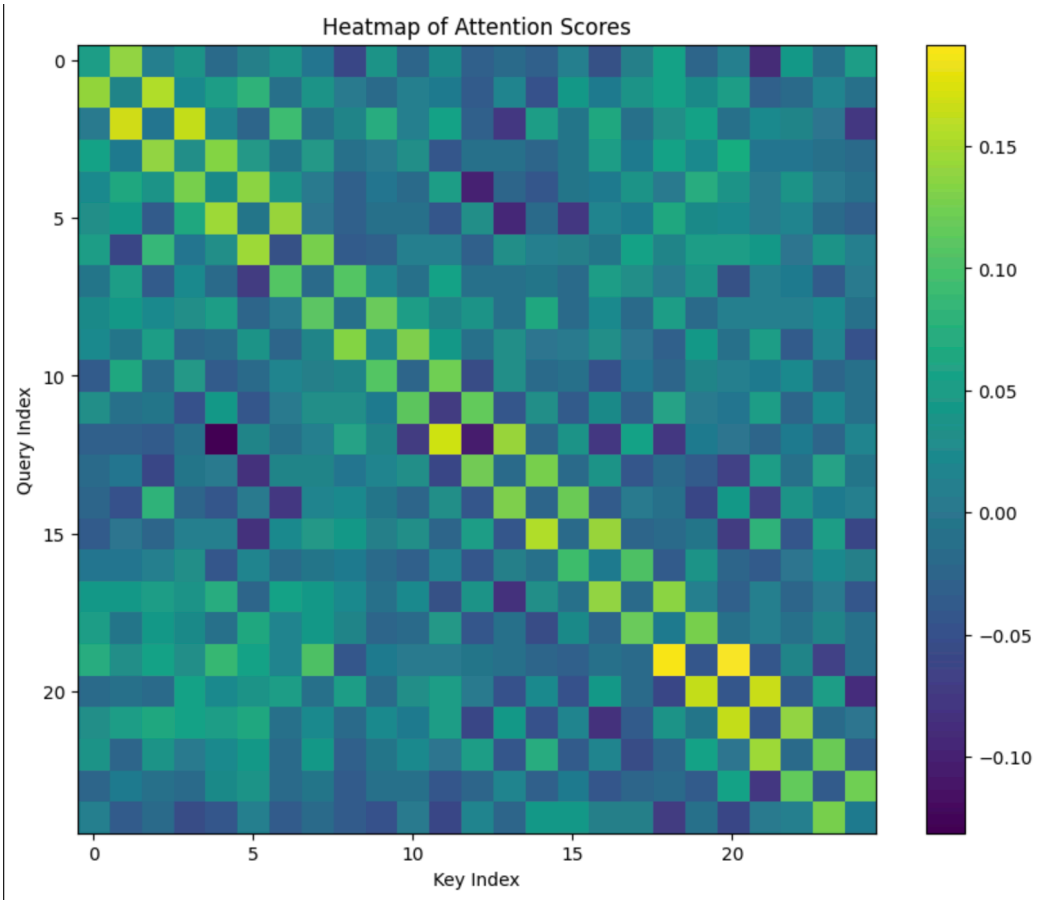
Indeed the network has found a solution, but how? Hypothesis 1a only works when $d_{head} d_{vocab}$ but here d_{vocab} is eight times as large as d_{head}.

We can visualize the attention scores across the query and key vectors to get some intuition.



There are two bright stripes in the attention heatmap demonstrate that each of the query vectors attend strongly to the key vectors of its two neighbors as we would expect. However, the patchiness of the rest of the matrix is odd.

Here is the same attention matrix but zoomed in on the region [125:150, 125:150].



It seems as though there's a checkerboard pattern where each query vector attends more to key vectors of the opposite parity. This insight leads us to propose a new scheme.

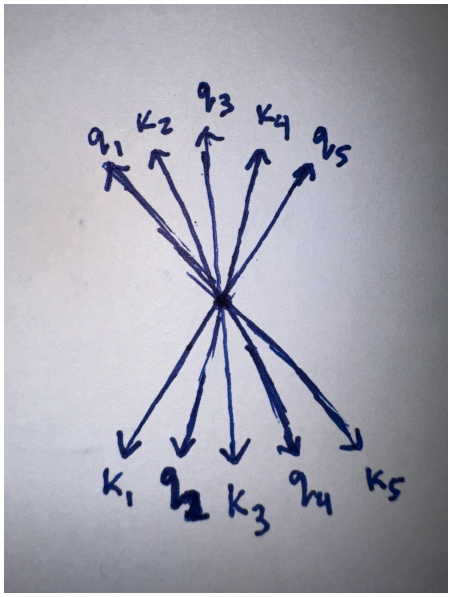
Hypothesis 2: The attention head can use a cyclic scheme to solve the problem even if $d_{head} < d_{vocab}$. The key intuition is that we can decompose two separate sets of constraints we need the key and query vectors to satisfy. Here are the two sequences:

$$q_1, k_2, q_3, k_4, \ldots, q_{d_{vocab}-1}, k_{d_{vocab}}$$
$$k_1, q_2, k_3, q_4, \ldots, k_{d_{vocab}-1}, q_{d_{vocab}}$$

Sequence 1 has odd query vectors and even key vectors. Sequence 2 has even query vectors and odd key vectors. The vectors within the same sequence must have high dot product with each other if they are neighbors. The vectors across sequences must have a dot product of zero.

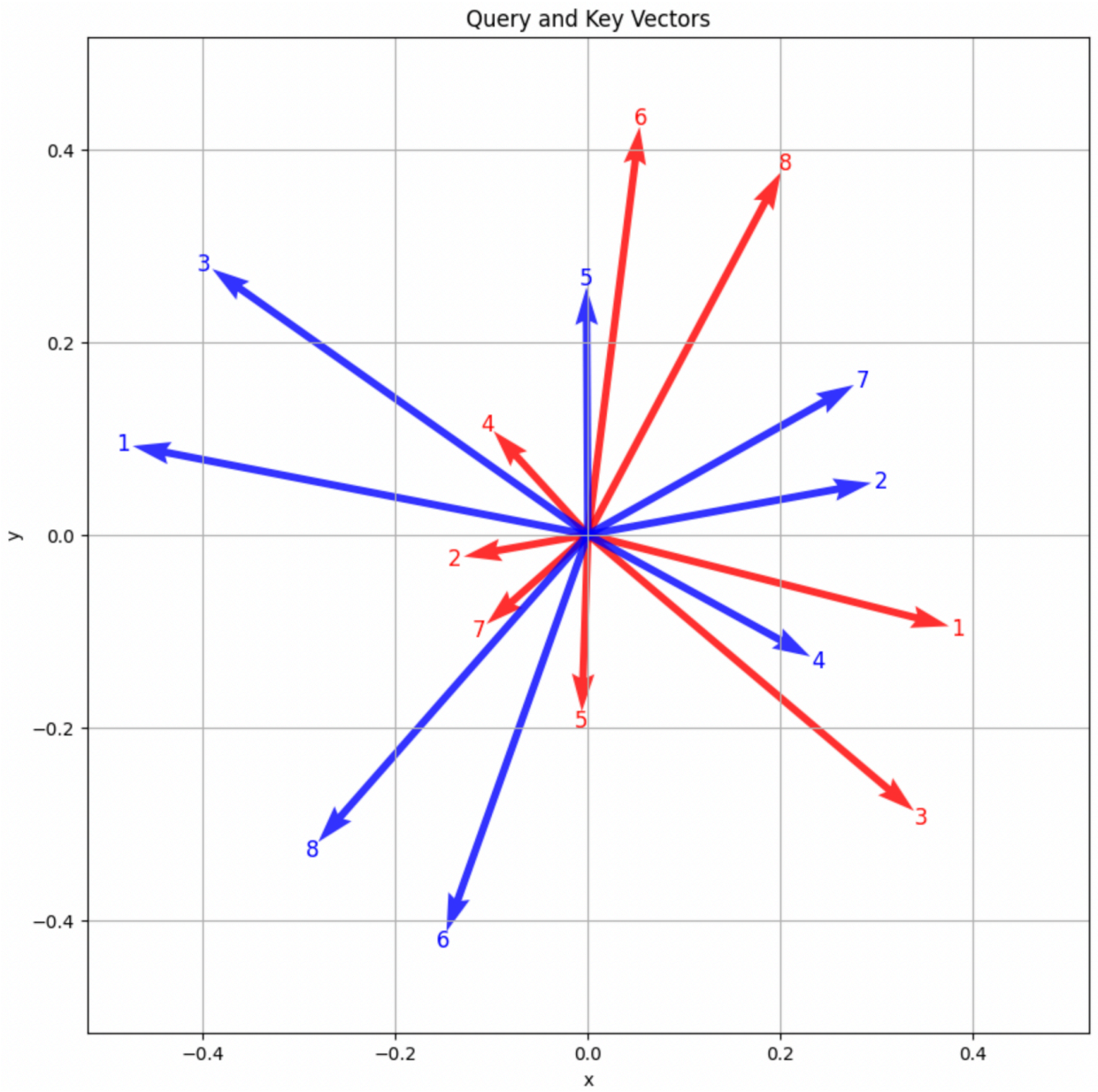
If this is satisfied, then every input element will attend to exactly its neighbors. We can actually satisfy this for arbitrary d_{vocab} for $d_{head} = 2$! The general idea is we lay out the vectors cyclically in exactly the order they appear in each sequence above. Since there are no constraints between the two sequences, we can put the two sets of vectors on opposite sides of the space.

We can illustrate this visually when $d_{head} = 2$ for $d_{vocab} = 5$.



Observation 4; Training a network with $d_{head} = 2$ and visualizing the query and key vectors yields a very similar scheme to Hypothesis 2!

We can visualize the query vectors (red) and key vectors (blue) in a 2d plane.



Notice how all odd query vectors are on one side *in order* and even query vectors are on the other side *in order*. The same is true for the key vectors.

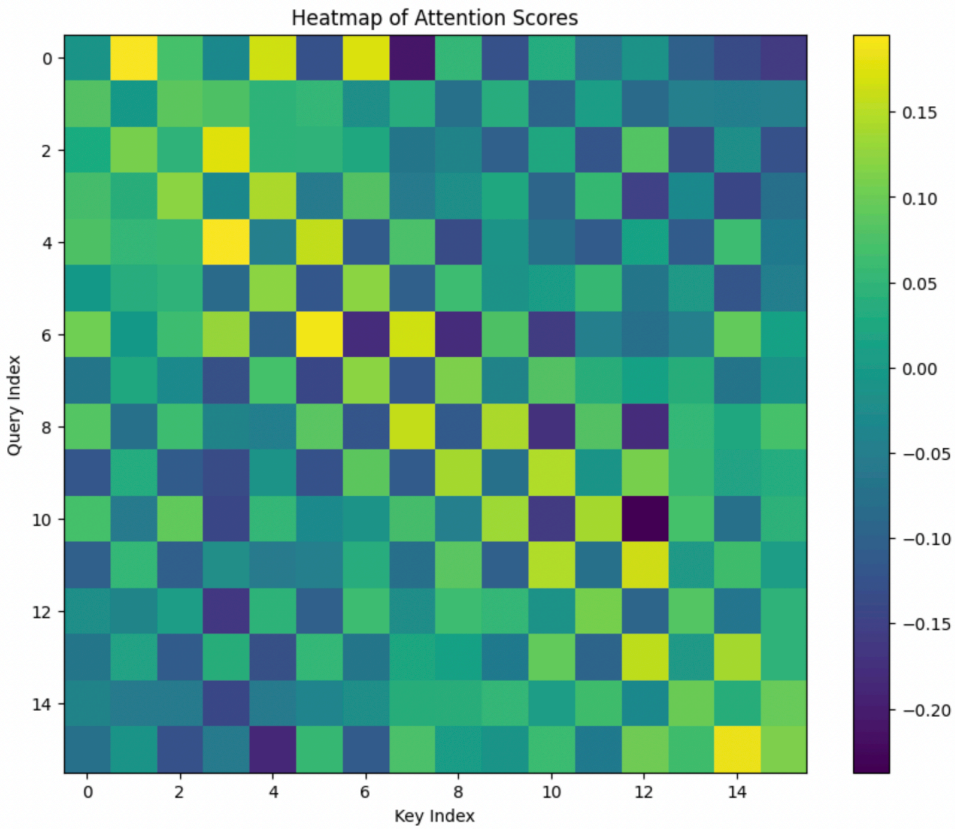
While the network's representation is very close to the theoretical solution outlined in Hypothesis 2, it is not exact for a few reasons: (1) network doesn't achieve exactly 0 loss at the end of training and (2) the network doesn't use constant length vectors so it has extra degrees of freedom.

Finally, we want some evidence that even with $d_{head} > 2$, a similar scheme takes place. While our evidence is not a conclusive proof, we provide a compelling observation.

Observation 5: The matrix of attention scores displays a checkerboard pattern even for higher dimensional spaces with $d_{head} > 2$.

Recall the decomposition of sequences of key and query vectors from Hypothesis 2. If this hypothesis is correct, we should expect q_1 to have much higher dot product to $k_2, k_4, k_6, \ldots$ than $k_1, k_3, k_5, \ldots$, since the even keys are in the same sequence as q_1. Similarly q_2 should have a higher dot product with the odd key vectors. That is, we should see a checkerboard pattern in the attention scores.

We saw this checkerboard pattern already in Observation 3, but we expect the pattern to be stronger on models with lower loss. It is easier to train the network at smaller dimensions so we demonstrate the effect more clearly here with $d_{vocab} = 16$ and $d_{head} = 4$.



We visualize the entire attention matrix here which clearly demonstrates the checkerboard pattern. Lower loss runs produce stronger checkerboard patterns, so we suspect that the smudges in the pattern can be fixed with even more tuning.

Appendix

The experiments used the following training parameters:

- Learning rate = 1e-4
- Adam optimizer with betas = [0.9, 0.999]
- ReduceLROnPlateau scheduler with patience = 100
- Batch size = 1000
- Loss function = cross entropy