CSCI 5922 problem set 4

Jake Krol

March 2025

1 Q1

1.1 Q1a: architectural similarities between transformers and RNNs

- Both transformers and RNNs can support **multiple input** vectors, such as n objects in a sequence.
- Both transformers and RNNs can support **multiple output** vectors, such as m objects in a sequence.

1.2 Q1b: architectural differences between transformers and RNNs

- **Transformers do not always have recurrent connections** between hidden layers. While, **RNNs always have recurrent connections** between hidden layers.
- In **transformers, input feature vectors** are converted into **three different representations** (involving three distinct, learned weight matrices): key, query, and value. Key, query, and value vectors create context vectors **encoding the attention** with respect to other inputs at a given time step. While in **RNNs**, **input feature vectors** only undergo **one transformation** before (involving one learned weight matrix) computing the current hidden state. And in RNNs, there is **no attention architectural component**.

1.3 Q1c: architectural similarities between transformers and MLPs

- Both transformers and MLPs have **at least one hidden state** which is a learned representation of input data. And, hidden states involve a **non-linear transformation**.
- Both transformers and MLPs **hidden state dimensions** are arbitrarily dependent on the **hyperparameter** set by the model designer.

1.4 Q1d: architectural differences between transformers and MLPs

- In **transformers, each hidden state** can have an associated **output**. While in **MLPs**, there exists only **one output layer** at the end of the model.
- In **MLPs**, **hidden states** h_t are computed solely by a **transformation of the previous state** h_{t-1}. However, **transformer hidden states** are **not always directly** a **function** of **other hidden states**. Instead in self-attention, transformer hidden states are derived by the value vector of the current input and a context vector derived by the similarity score between the current input's query vector and all other inputs' key vectors.

2 Q2

2.1 Q2a

2.1.1 Q2ai

The role of $\sqrt{d_k}$, which represents the dimension of both query and key vectors in a transformer, in the attention function is to **scale down the dot products** of query and key vectors before applying the softmax function.

The motivation for scaling down dot products is to reduce the potential numerical instability since the softmax function's terms grow exponentially with the dot product scores of queries and keys.

$$softmax(\mathbf{x}) = \frac{e^{x_i}}{\sum_j^n e^{x_j}}$$

In attention, $\mathbf{x}$ houses attention scores of row-vectors in QK^T, so the softmax denominator could become exceptionally large, if d_k is large. These **large values could produce numerically instability**, and empirically it has been found that the $\sqrt{d_k}$ normalization of attention scores improves training. For instance, the output softmax vector could be saturated with values close many values close to 0 and one value close to 1, due to the exponential nature of terms. This would essentially cause hard-attention with respect to a single other token. Yet, in soft-attention it is often desirable to consider all other to some extent.

2.1.2 Q2aii

As previously mentioned, QK^T are attention scores, and applying softmax row-wise to QK^T yields attention weight vectors with elements that sum to 1. Intuitively, each row-vector in QK^T has the attention weights for all other tokens, j, with respect to a single token i. Furthermore, the output range of softmax elements being $\in [0, 1]$ enables interpreting weights as the probability of other

tokens, j, being influential to the current token, i. So, the **role of softmax** is to **coerce attention score vectors to a probability distribution**.

2.2 Q2b

$softmax(\frac{QK^T}{\sqrt{d_k}})$ can be thought of as an attention matrix because each **row-vector houses the attention weights for all other tokens, j, with respect to the current token row, i.** This matrix encodes attention across tokens.

2.3 Q2c

The motivation to use multi-headed attention is to **capture different levels of relationships among tokens**. For example, in natural language processing there may be separate concepts present in the same written context, and two attention heads could quantify how words relate to each other under concept one and concept two, separately.

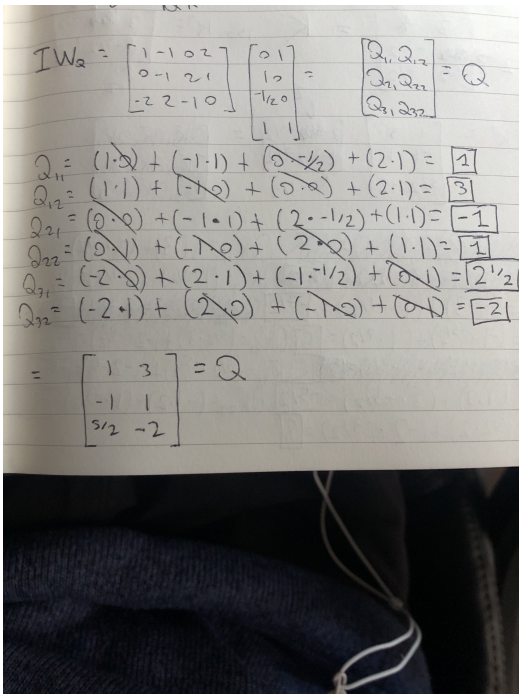
2.4 Q2d

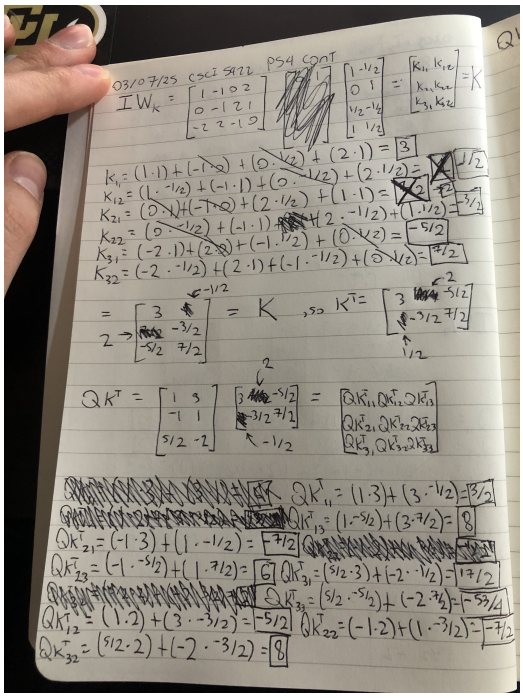
The key difference between transformer encoders and decoders that **encoders only see the input**, while **decoders see encoder representations and previous decoder representations**. A **real-world use case of encoder transformers** is representation learning of word tokens in natural language processing. The dense vectors learned by the encoder transformer could be used to understand semantic similarity of words.

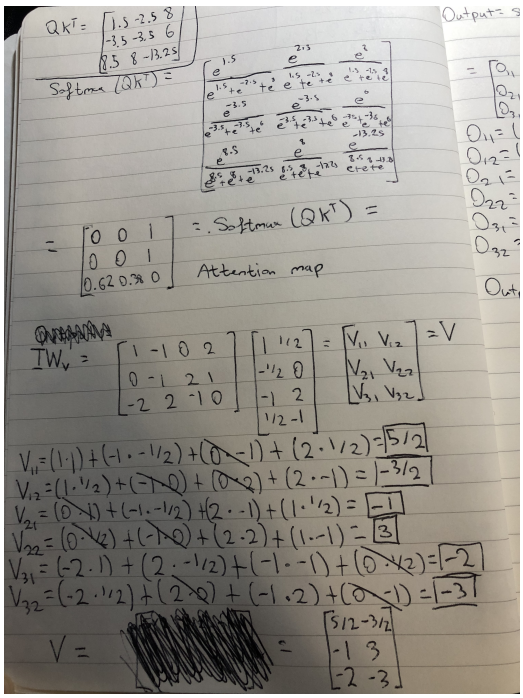
Meanwhile, the decoder has both a representation and an associated output that is, during training, compared with target values to compute loss. Also, in the "Attention is all you need" transformer architecture, the **encoder has only self-attention**, while the **decoder has both masked self-attention and cross-attention** to the encoder's hidden states. **A real world use case of decoder transformers** is for natural language generation, such as predicting the next word given some previous context.

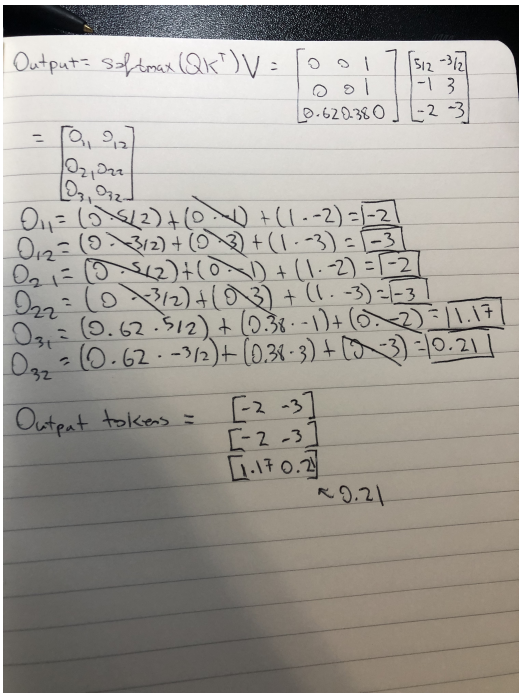
3 Q3

3.1 Q3a and Q3b

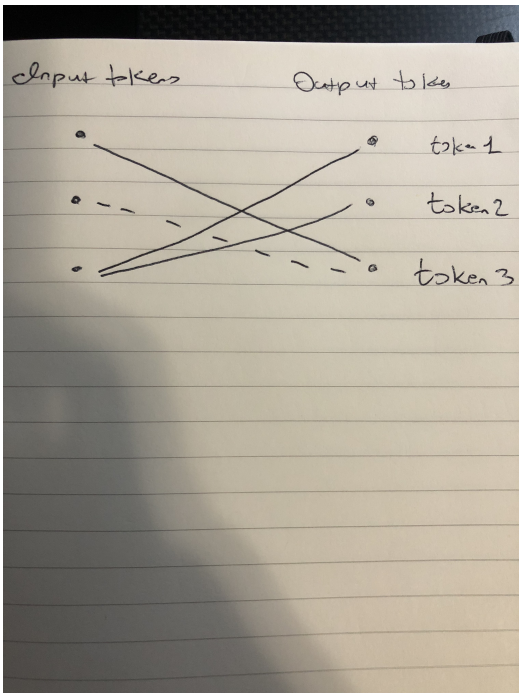



03/07/25 CSCI 5922 PS4 COAT
IW_K =
$K_{11} = (1\cdot1) + (-1\cdot-1/2) + (0\cdot1/2) + (2\cdot1/2) = 3$
$K_{12} = (1\cdot-1/2) + (-1\cdot1) + (0\cdot1/2) + (2\cdot1/2) = -1/2$
$K_{21} = (0\cdot1) + (-1\cdot-1/2) + (2\cdot1/2) + (1\cdot1) = 3/2$
$K_{22} = (0\cdot-1/2) + (-1\cdot1) + (2\cdot-1/2) + (1\cdot1/2) = -5/2$
$K_{31} = (-2\cdot1) + (2\cdot-1/2) + (-1\cdot-1/2) + (0\cdot1/2) = -7/2$
$K_{32} = (-2\cdot-1/2) + (2\cdot1) + (-1\cdot-1/2) + (0\cdot1/2) = 3$
K^T
QK^T =
$QK^T_{11} = (1\cdot3) + (3\cdot-1/2) = 3/2$
$QK^T_{13} = (1\cdot-5/2) + (3\cdot-7/2) = 8$
$QK^T_{21} = (-1\cdot3) + (1\cdot-1/2) = -7/2$
$QK^T_{23} = (-1\cdot-5/2) + (1\cdot-7/2) = -6$
$QK^T_{31} = (5/2\cdot3) + (-2\cdot-1/2) = 17/2$
$QK^T_{33} = (5/2\cdot-5/2) + (-2\cdot-7/2) = -53/4$
$QK^T_{12} = (1\cdot2) + (3\cdot-3/2) = -5/2$
$QK^T_{22} = (-1\cdot2) + (1\cdot-3/2) = -7/2$
$QK^T_{32} = (5/2\cdot2) + (-2\cdot-3/2) = 8$


QK^T =
Softmax (QK^T) =
= Softmax (QK^T) =
Attention map
Output =
IW_V =
$V_{11} = (1\cdot1) + (-1\cdot-1/2) + (0\cdot-1) + (2\cdot1/2) = 5/2$
$V_{12} = (1\cdot1/2) + (-1\cdot0) + (0\cdot2) + (2\cdot-1) = -3/2$
$V_{21} = (0\cdot1) + (-1\cdot-1/2) + (2\cdot-1) + (1\cdot1/2) = -1$
$V_{22} = (0\cdot1/2) + (-1\cdot0) + (2\cdot2) + (1\cdot-1) = 3$
$V_{31} = (-2\cdot1) + (2\cdot-1/2) + (-1\cdot-1) + (0\cdot1/2) = -2$
$V_{32} = (-2\cdot1/2) + (2\cdot0) + (-1\cdot2) + (0\cdot-1) = -3$
V =


Output = softmax(QK^T)V = [0 0 1; 0 0 1; 0.62 0.38 0] [5/2 -3/2; -1 3; -2 -3]
= [O11 O12; O21 O22; O31 O32]
O11 = (0 · 5/2) + (0 · -1) + (1 · -2) = -2
O12 = (0 · -3/2) + (0 · 3) + (1 · -3) = -3
O21 = (0 · 5/2) + (0 · -1) + (1 · -2) = -2
O22 = (0 · -3/2) + (0 · 3) + (1 · -3) = -3
O31 = (0.62 · 5/2) + (0.38 · -1) + (0 · -2) = 1.17
O32 = (0.62 · -3/2) + (0.38 · 3) + (0 · -3) = 0.21
Output tokens = [-2 -3; -2 -3; 1.17 0.21]
≈ 0.21

3.2 Q3c



3.3 Q3d

- Output token 1 attends to input token 3.
- Output token 2 attends to input token 3.
- Output token 3 attends to input token 1 and, to a lesser extent, attends to input token 2.

  A pattern I see is that none of the output tokens attend highly to input tokens with the same positional encoding.