Natural Language Processing: Assignment 6

Mohammad Taha Majlesi - 810101504

Fall 2022

Abstract

This document provides a detailed, section-by-section walkthrough of Assignment 6, focusing on the mechanics, complexity, and theoretical underpinnings of the Multi-Head Self-Attention mechanism. Each concept is expanded with in-depth explanations to build a strong foundational understanding, moving from parameter counting and computational analysis to the profound connection between self-attention and convolution. The material is presented in a clear, structured format suitable for deep learning practitioners and students.

Contents

1 The Architecture of Multi-Head Self-Attention

1.1 Core Concept: The Intuition Behind Self-Attention

At its heart, self-attention is a mechanism that allows a model to weigh the importance of different elements in an input sequence when processing a specific element. For instance, when processing the word "it" in the sentence "The cat sat on the mat because it was tired," self-attention helps the model understand that "it" refers to "the cat" and not "the mat."

This is achieved through three key components derived from each input element (e.g., a word vector or an image pixel):

- **Query (Q):** A representation of the current element, acting as a "question" about what other elements are relevant to it.
- **Key (K):** A representation of each element in the sequence that can be "matched" against a query. The dot product of a Query and a Key determines their compatibility or "attention score."
- **Value (V):** A representation of each element that contains the actual information to be aggregated.

The output for a given element is a weighted sum of all Value vectors in the sequence, where the weights are the attention scores calculated from the Query-Key interactions. "Multi-head" attention simply means running this process in parallel multiple times with different learned transformations, allowing the model to focus on different types of relationships simultaneously.

1.2 Calculating the Number of Parameters (Question a.1)

To understand a model's capacity and memory footprint, we must count its learnable parameters. Let's break down the components of a multi-head attention block to count them precisely.

1.2.1 Component Dimensions

First, we define the key dimensions involved:

- N_h: The number of parallel attention heads.
- D_i: The dimension of each input vector (e.g., the embedding size).
- D_{hid}: The internal dimension for Query and Key vectors within each head (often denoted d_k).
- D_o: The internal dimension for Value vectors within each head (often denoted d_v).
- D_{out}: The final output dimension of the entire block.

1.2.2 Step 1: Parameters Within Each Head

For each head $h \in [N_h]$, the input vector is transformed into Query, Key, and Value vectors using independent weight matrices.

- **Query Matrix ($W_q^{(h)}$):** Transforms an input of size D_i to a Query of size D_{hid}. Its shape is $\mathbb{R}^{D_i \times D_{hid}}$, containing $D_i \cdot D_{hid}$ parameters.
- **Key Matrix ($W_k^{(h)}$):** Transforms an input of size D_i to a Key of size D_{hid}. Its shape is $\mathbb{R}^{D_i \times D_{hid}}$, containing $D_i \cdot D_{hid}$ parameters.
- **Value Matrix ($W_v^{(h)}$):** Transforms an input of size D_i to a Value of size D_o. Its shape is $\mathbb{R}^{D_i \times D_o}$, containing $D_i \cdot D_o$ parameters.

The total number of parameters for a *single head* is the sum of these:

$$\text{Params}_{\text{head}} = (D_i \cdot D_{hid}) + (D_i \cdot D_{hid}) + (D_i \cdot D_o) = D_i \cdot (2D_{hid} + D_o)$$

1.2.3 Step 2: Total Parameters for All Heads

Since there are N_h independent heads, each with its own set of matrices, we multiply the per-head parameter count by N_h:

$$\text{Params}_{\text{all\_heads}} = \boxed{N_h \cdot D_i \cdot (2D_{hid} + D_o)}$$

1.2.4 Step 3: Parameters for the Output Projection

The outputs of all N_h heads (each of dimension D_o) are concatenated into a single large vector of size $N_h \cdot D_o$. This vector is then projected back to the desired final output dimension, D_{out}.

- **Output Matrix (W_{out}):** This matrix takes the concatenated vector and produces the final output. Its shape is $\mathbb{R}^{(N_h D_o) \times D_{out}}$, containing $(N_h \cdot D_o) \cdot D_{out}$ parameters.
- **Output Bias (b_{out}):** A bias vector is typically added after the final matrix multiplication. It has a size equal to the output dimension, D_{out}, adding D_{out} parameters.

Total parameters for the output layer: $N_h \cdot D_o \cdot D_{out} + D_{out}$.

1.2.5 Step 4: Grand Total

The total number of parameters in the entire multi-head self-attention module is the sum of the head parameters and the output projection parameters:

$$\text{Total Parameters} = \underbrace{N_h \cdot D_i(2D_{hid} + D_o)}_{\text{All Heads}} + \underbrace{N_h \cdot D_o \cdot D_{out}}_{\text{Output Matrix}} + \underbrace{D_{out}}_{\text{Output Bias}}$$

This formula is critical for designing models that fit within specific hardware or memory constraints.

1.3 The Mathematical Formulation of Self-Attention (Question a.2)

Let's formalize the self-attention calculation for a sequence of inputs $Z \in \mathbb{R}^{N \times D_i}$, where N is the sequence length (e.g., $W \times H$ for an image) and D_i is the input feature dimension. We denote the vector for the q-th element as $Z_{q,:}$.

The output for this q-th element, $O_{q,:}$, is calculated as follows:

1. **Score Calculation:** For the query $Z_{q,:}$, we compute a score against every key $Z_{k,:}$ in the sequence. For a single head (we omit the (h) superscript for clarity):

$$\text{score}(q, k) = (Z_{q,:} W_q) \cdot (Z_{k,:} W_k)^T$$

   This dot product results in a scalar value measuring the compatibility between query q and key k. Doing this for all k gives us a vector of scores for query q.

2. **Normalization with Softmax:** The scores are scaled (often by $\sqrt{D_{hid}}$) and then normalized using the softmax function. This converts the raw scores into a probability distribution of attention weights, ensuring they are all positive and sum to 1.

$$\alpha_{q,k} = \text{softmax}(\text{score}(q, k)) = \frac{\exp(\text{score}(q, k))}{\sum_{j=1}^{N} \exp(\text{score}(q, j))}$$

   The weight $\alpha_{q,k}$ represents how much attention the q-th element should pay to the k-th element.

3. **Weighted Aggregation of Values:** The final output for the q-th element is the weighted sum of all Value vectors in the sequence, using the attention weights $\alpha_{q,k}$.

$$O_{q,:} = \sum_{k=1}^{N} \alpha_{q,k} (Z_{k,:} W_v)$$

In matrix form for the entire sequence, this is elegantly expressed as:

$$\text{Attention}(Z) = \text{softmax}\left(\frac{Z W_q (Z W_k)^T}{\sqrt{d_k}}\right)(Z W_v)$$

The equations in the assignment are a specific instance of this general formulation.

2 Computational Complexity Analysis

Computational complexity, expressed using Big O notation, tells us how the runtime and memory requirements of an algorithm scale with the size of the input. For self-attention, the primary input variables are the sequence length (N) and the feature dimensions (D_x, D_p).

2.1 Complexity with Absolute Positional Encoding (Question b.2)

Absolute positional encoding adds information about an element's position directly to its input vector. This affects the calculation of the attention score matrix, $A \in \mathbb{R}^{N\times N}$. The calculation of each entry $A_{q,k}$ involves terms like $Z_{q,:}W_q W_k^T Z_{k,:}^T$.

Let's analyze the cost step-by-step, assuming $D_i = D_{hid} = D_o = D_x$ for simplicity.

1. **Cost of the Attention Score Matrix ($A^{absolute}$):**
   - To compute one entry $A_{q,k}$, we need to perform vector-matrix multiplications.
   - The cost of $Z_{q,:}W_q$ (a $(1 \times D_x)$ vector multiplied by a $(D_x \times D_x)$ matrix) is $O(D_x^2)$.
   - The expression for one entry involves two such multiplications and a dot product, so the cost is dominated by $O(D_x^2)$.
   - Since the matrix A has $N \times N = N^2$ entries, the total cost to compute all attention scores is $O(N^2 D_x^2)$.
2. **Cost of Softmax:**
   - Applying softmax requires normalizing each of the N rows. This is proportional to the size of the matrix, so the cost is $O(N^2)$.
3. **Cost of Value Aggregation:**
   - This involves multiplying the attention weight matrix ($\hat{A} \in \mathbb{R}^{N\times N}$) by the transformed value matrix ($ZW_v \in \mathbb{R}^{N\times D_x}$).
   - The cost of this matrix multiplication is $O(N^2 D_x)$.

Final Complexity (Absolute): Summing the dominant terms, the total complexity is:

$$O(\underbrace{N^2 D_x^2}_{\text{Scores}} + \underbrace{N^2}_{\text{Softmax}} + \underbrace{N^2 D_x}_{\text{Aggregation}}) \approx O(N^2 D_x^2 + N^2 D_x)$$

The quadratic dependence on both sequence length (N^2) and feature dimension (D_x^2) makes this very computationally expensive.

2.2 Complexity with Relative Positional Encoding (Question b.2)

Relative positional encoding computes attention scores based on the pairwise distance between elements, rather than their absolute positions. This simplifies the score calculation significantly. The score $A_{q,k}$ can often be computed with a complexity that depends only on the dimension of the positional encoding, D_p.

1. **Cost of the Attention Score Matrix ($A^{relative}$):**
    - The calculation for a single entry $A_{q,k}$ (like the inner product $v^T r_s$ in the assignment) now only costs $O(D_p)$.
    - The total cost to compute the entire $N \times N$ score matrix becomes $O(N^2 D_p)$.
2. **Cost of Softmax and Value Aggregation:**
    - These steps are identical to the absolute encoding case, with costs of $O(N^2)$ and $O(N^2 D_x)$ respectively.

Final Complexity (Relative): Summing the dominant terms:

$$O(\underbrace{N^2 D_p}_{\text{Scores}} + \underbrace{N^2}_{\text{Softmax}} + \underbrace{N^2 D_x}_{\text{Aggregation}}) \approx O(N^2 D_p + N^2 D_x)$$

Comparison: If we assume that $D_p \ll D_x$ (which is typical), relative encoding provides a substantial computational saving. The complexity is still quadratic in sequence length (N^2), but it is now only linear in the feature dimension (D_x) instead of quadratic. This is a key reason for the popularity of relative attention mechanisms in modern Transformer architectures.

3 The Equivalence of Self-Attention and Convolution

This section explores one of the most insightful theoretical results about self-attention: under certain conditions, it can perfectly simulate a standard convolution operation. This demonstrates that convolution can be viewed as a specialized, less flexible subset of what self-attention can do.

3.1 High-Level Intuition

- **Convolution:** Uses a small, static kernel (filter) that is slid across the input. The kernel's weights are independent of the input data (translation-invariant) and it only considers a local neighborhood.
- **Self-Attention:** Has a global receptive field (it can look at any other element) and its "weights" (the attention scores) are dynamically computed based on the input data itself.

The proof shows how the highly flexible, dynamic mechanism of attention can be constrained to behave like the rigid, static mechanism of convolution.

3.2 The Formal Proof of Equivalence (Question c.2)

The proof hinges on a critical assumption: we force the softmax function to act as a "one-hot" selector. For each head h and each query q, the softmax output is 1 for a single key k and 0 for all others. Crucially, this selected key is determined only by a fixed spatial offset from q, defined by a function $f(h)$ that maps a head to an offset vector Δ.

Assumption: $\text{softmax}(A^{(h)}_{q,k}) = 1$ if $k = q - f(h)$, and 0 otherwise.

Let's start with the multi-head self-attention formula for a single output pixel q:

$$\text{Attention}(Z)_{q,:} = \sum_{h=1}^{N_h} \left(\sum_{k=1}^{N} \text{softmax}(A^{(h)}_{q,k}) Z_{k,:} \right) W_v^{(h)} + b_{out}$$

1. **Apply the one-hot assumption.** The inner sum over k collapses, as only the term where $k = q - f(h)$ survives:

$$= \sum_{h=1}^{N_h} Z_{q-f(h),:} W_v^{(h)} + b_{out}$$

2. **Introduce the shift vector Δ.** Let Δ_K be the set of all possible spatial shifts in a convolutional kernel (e.g., {(-1,-1), (-1,0), ..., (1,1)}). The function $f : [N_h] \to \Delta_K$ creates a one-to-one mapping between each attention head and a unique shift in the kernel. We can therefore switch the summation from being over heads (h) to being over shifts ($\Delta \in \Delta_K$). We find the corresponding head using the inverse map $h = f^{-1}(\Delta)$.

$$= \sum_{\Delta \in \Delta_K} Z_{q-\Delta,:} W_v^{(f^{-1}(\Delta))} + b_{out}$$

(Note: Using $q + \Delta$ or $q - \Delta$ is a matter of convention, as the set of shifts Δ_K is typically symmetric).

3. **Define the Convolutional Filter.** Now, let's define a convolutional filter, W^{conv}, where the weights for a specific spatial offset Δ are exactly equal to the value matrix of the corresponding attention head:

$$W^{\text{conv}}_{\Delta,:,:} := W_v^{(f^{-1}(\Delta))}$$

   Substituting this definition into our equation gives:

$$= \sum_{\Delta \in \Delta_K} Z_{q-\Delta,:} W^{\text{conv}}_{\Delta,:,:} + b_{out}$$

This final expression is the mathematical definition of a discrete convolution. We have successfully shown that multi-head self-attention can replicate a convolution.

3.2.1 Mapping the Concepts

Multi-Head Self-Attention	**Convolution**
An attention head (h)	A specific position in the kernel (offset Δ)
The head's value matrix ($W_v^{(h)}$)	The filter weights at that kernel position (W_Δ^{conv})
Number of heads (N_h)	Size of the convolutional kernel (e.g., $3 \times 3 = 9$)

3.3 Extensions to Other Convolution Types

- **Dilated Convolution (Question e.2):** Yes, self-attention can easily simulate this. A dilated convolution applies a filter over a larger area by skipping pixels. This can be achieved by simply defining the head-to-offset map $f(h)$ to map to non-contiguous pixel locations (e.g., shifts of -2, 0, 2 instead of -1, 0, 1). The flexibility of attention makes this trivial.

- **Strided Convolution (Question e.3):** Yes, but it requires an additional step. A strided convolution reduces the output resolution by only computing outputs at specified intervals (the stride). The self-attention layer itself computes an output for every input pixel (a dense output). To simulate a stride, a subsequent downsampling or pooling layer must be applied to discard the outputs that would have been skipped.

4 Advanced Properties and Training Dynamics

4.1 Hard vs. Soft Attention: The Role of α (Question d.1)

The softmax function includes a temperature or scaling factor, often represented as α (or implicitly as $1/\sqrt{d_k}$). This proof explores what happens in the limit as $\alpha \to \infty$.

Goal: Prove that $\lim_{\alpha\to\infty} \text{softmax}(A_{q,:})_k = 1$ for one specific k and 0 for all others.

The attention score is given as $A_{q,j} = -\alpha(\dots)$. The key insight is that for any two scores $A_{q,j}$ and $A_{q,k}$, their difference is also multiplied by α. As α grows, even tiny differences in the base scores become enormous.

Let's analyze the softmax expression:

$$\text{softmax}(A_{q,:})_k = \frac{\exp(A_{q,k})}{\sum_j \exp(A_{q,j})}$$

Let's assume the score for our target key, $A_{q,k}$, is the highest score. We can rewrite the denominator:

$$= \frac{\exp(A_{q,k})}{\exp(A_{q,k}) + \sum_{j\neq k} \exp(A_{q,j})}$$

Divide the numerator and denominator by $\exp(A_{q,k})$:

$$= \frac{1}{1 + \sum_{j\neq k} \frac{\exp(A_{q,j})}{\exp(A_{q,k})}} = \frac{1}{1 + \sum_{j\neq k} \exp(A_{q,j} - A_{q,k})}$$

Since we assumed $A_{q,k}$ is the highest score, the term $(A_{q,j} - A_{q,k})$ is negative for all $j \neq k$. This difference is proportional to $-\alpha$. As $\alpha \to \infty$, the term $(A_{q,j} - A_{q,k}) \to -\infty$. Consequently, $\exp(\dots) \to 0$.

Therefore, in the limit:

$$\lim_{\alpha\to\infty} \text{softmax}(A_{q,:})_k = \frac{1}{1 + \sum_{j\neq k} 0} = \frac{1}{1} = 1$$

Intuition: A large α makes the softmax function "sharper" or less "soft." In the infinite limit, it becomes a "hard" attention mechanism, equivalent to an 'argmax', which selects only the single most relevant item and ignores all others. This is the mechanism that enables the convolution equivalence proof.

4.2 How Attention Heads Evolve During Training (Question 2)

The behavior of attention heads is not static; it evolves as the model learns from data.

- **At Initialization:** With random weights, attention patterns are chaotic and meaningless. Heads typically attend weakly to many pixels, often in a noisy, localized cloud around the query pixel.
- **Early Training (Lower Layers):** The layers closest to the input learn to perform fundamental feature extraction.
  - Some heads specialize into **local pattern detectors**, similar to the Gabor filters or edge detectors in the first layers of a CNN. They learn fixed, convolution-like patterns.

    - Other heads learn to become **context aggregators**, spreading their attention over a wider area to provide a blurred or downsampled view of the input, akin to an average pooling operation.

- **Late Training (Higher Layers):** Deeper layers operate on more abstract features. Their attention heads learn more complex, content-dependent relationships.
    - Heads may learn to attend to **semantically similar regions**, regardless of their spatial distance. For example, in an image with two dogs, a head might learn that query pixels on one dog should attend to key pixels on the other dog.
    - They can form **structural patterns**, like attending in straight lines or circles, effectively learning to perform specialized, long-range feature extraction that is difficult for small, local convolutions.

Conclusion: The training process showcases the flexibility of self-attention. The model learns that for many tasks, local, convolution-like operations are extremely useful, and it dedicates some of its "attentional capacity" (i.e., some of its heads) to simulating them. However, it also learns to use other heads for more dynamic, long-range, and content-aware interactions that are impossible for a fixed convolutional kernel to perform, giving it a significant advantage in modeling complex dependencies.