MIDDLE EAST TECHNICAL UNIVERSITY

SEMESTER I EXAMINATION 2024-2025

CENG 403 – Deep Learning - Self-Attention & Transformers (University Sources) - ANSWERED

Question 1. Mathematical Foundations of Self-Attention (25 marks)

Based on transformer research papers and university deep learning courses.

(a) Define the self-attention mechanism mathematically. For a sequence of input vectors $X = [x_1, x_2, ..., x_n]$ where $x_i \in \mathbb{R}^d$, derive the complete attention formula including: (10 marks)

- Query, Key, Value transformations
- Attention weight computation
- Output aggregation
- Scaling factor justification

Answer: Complete mathematical formulation of self-attention with QKV transformations, scaled dot-product computation, and output aggregation.

Mathematical Definition of Self-Attention:

1. Input Processing: Given input sequence $X = [x_1, x_2, ..., x_n]$ where $x_i \in \mathbb{R}^d$

2. Query, Key, Value Transformations:

$$Q = XW_Q \quad \text{where } W_Q \in \mathbb{R}^{d \times d_k} \tag{1}$$

$$K = XW_K \quad \text{where } W_K \in \mathbb{R}^{d \times d_k} \tag{2}$$

$$V = XW_V \quad \text{where } W_V \in \mathbb{R}^{d \times d_v} \tag{3}$$

3. Attention Weight Computation:

$$e_{ij} = \frac{q_i^T k_j}{\sqrt{d_k}} \quad \text{(scaled similarity)} \tag{4}$$

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{n} \exp(e_{ik})} \quad \text{(softmax normalization)} \tag{5}$$

4. Output Aggregation:

$$z_i = \sum_{j=1}^{n} \alpha_{ij} v_j$$

5. Complete Matrix Form:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) V$$

Scaling Factor Justification:

Statistical Analysis:

- For random vectors $q, k \sim \mathcal{N}(0, I)$ in $\mathbb{R}^{d_k}$
- Dot product: $q^T k = \sum_{i=1}^{d_k} q_i k_i$
- Expected value: $\mathbb{E}[q^T k] = 0$
- Variance: $\text{Var}[q^T k] = d_k$
- Standard deviation: $\sigma[q^T k] = \sqrt{d_k}$

Why Scaling is Critical:

- Large $d_k \to$ large dot products $\to$ saturated softmax
- Saturated softmax $\to$ tiny gradients $\to$ poor learning
- Scaling by $\frac{1}{\sqrt{d_k}}$ normalizes variance to 1
- Maintains stable gradients across different model sizes

(b) Prove that the attention weights α_{ij} sum to 1 for each query position i. Show that $\sum_{j=1}^{n} \alpha_{ij} = 1$. (5 marks)

Answer: Softmax normalization guarantees attention weights sum to 1 by construction.

Proof that $\sum_{j=1}^{n} \alpha_{ij} = 1$:

Given: Attention weights are computed using softmax:

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{n} \exp(e_{ik})}$$

To Prove: $\sum_{j=1}^{n} \alpha_{ij} = 1$ for any fixed i

Proof:

$$\begin{aligned}
\sum_{j=1}^{n} \alpha_{ij} &= \sum_{j=1}^{n} \frac{\exp(e_{ij})}{\sum_{k=1}^{n} \exp(e_{ik})} && (6)\\
&= \frac{1}{\sum_{k=1}^{n} \exp(e_{ik})} \sum_{j=1}^{n} \exp(e_{ij}) && (7)\\
&= \frac{\sum_{j=1}^{n} \exp(e_{ij})}{\sum_{k=1}^{n} \exp(e_{ik})} && (8)\\
&= \frac{\sum_{j=1}^{n} \exp(e_{ij})}{\sum_{j=1}^{n} \exp(e_{ij})} \quad \text{(relabeling } k \text{ as } j\text{)} && (9)\\
&= 1 && (10)
\end{aligned}$$

Key Insight: Softmax is designed to produce a probability distribution, ensuring all weights sum to 1 while preserving relative magnitudes of similarity scores.

Implications:

- Each output z_i is a convex combination of value vectors
- Attention weights represent a probability distribution over positions
- This property enables interpretability of attention patterns

(c) Analyze the computational and space complexity of self-attention for sequence length n and embedding dimension d. Compare with RNN complexity. (6 marks)

Answer: Self-attention has $O(n^2 d)$ time complexity vs RNN's $O(nd^2)$, with different trade-offs for different sequence lengths.

Self-Attention Complexity Analysis:

Time Complexity:

- QKV projections: $O(nd^2)$ (3 matrix multiplications)
- Attention scores: QK^T requires $O(n^2 d)$ operations
- Softmax: $O(n^2)$ operations

- Output computation: $O(n^2 d)$ operations
- **Total:** $O(n^2 d + nd^2)$

Space Complexity:

- QKV matrices: $O(nd)$ each
- Attention matrix: $O(n^2)$
- **Total:** $O(n^2 + nd)$

RNN Complexity Analysis:

Time Complexity:

- Per time step: $O(d^2)$ (hidden-to-hidden transformation)
- Total for sequence: $O(nd^2)$
- Sequential dependency prevents parallelization

Space Complexity:

- Hidden states: $O(nd)$ (if storing all for backprop)
- Or $O(d)$ if not storing intermediate states

Comparison and Trade-offs:

When $n << d$ (short sequences, large embeddings):

- Self-attention: $O(nd^2)$ dominates
- RNN: $O(nd^2)$
- Similar complexity, but self-attention allows parallelization

When $n >> d$ (long sequences, small embeddings):

- Self-attention: $O(n^2 d)$ becomes prohibitive
- RNN: $O(nd^2)$ remains manageable
- RNN may be more efficient for very long sequences

Practical Implications:

- Self-attention excels with parallel hardware (GPUs)

- RNNs better for extremely long sequences
- Memory requirements can be limiting factor for self-attention

(d) Explain why self-attention is permutation invariant and how positional encoding addresses this limitation. (4 marks)

Answer: Self-attention treats input as a set, losing order information. Positional encoding injects position-specific signals to restore sequence order awareness.

Permutation Invariance in Self-Attention:

Why It Occurs:

- Attention weights depend only on content similarity: $\alpha_{ij} \propto \exp(q_i^T k_j)$
- No inherent notion of position in the computation
- Swapping positions i and j doesn't change the final representations
- Mathematical proof: $f(\pi(X)) = \pi(f(X))$ for any permutation π

Problem Illustration:

- "Cat chased dog" vs "Dog chased cat"
- Both would produce identical embeddings without positional information
- Critical semantic differences lost

Positional Encoding Solution:

Basic Approach:

$$\text{Input} = \text{Token Embedding} + \text{Positional Encoding}$$

How It Works:

- Each position gets unique encoding vector
- Combined with content embeddings before attention
- Attention now sees both content and position information
- Different positions produce different representations

Types of Positional Encoding:

- **Learned:** Position-specific parameters trained end-to-end
- **Sinusoidal:** Fixed trigonometric functions with different frequencies
- **Relative:** Encoding relative distances between positions

Effectiveness: Position encoding breaks permutation invariance while preserving the parallel processing benefits of self-attention.

Question 2. Multi-Head Attention Architecture (30 marks)
Based on "Attention Is All You Need" and related transformer literature.

(a) Design a multi-head attention mechanism with $h = 8$ heads for input dimension $d_{model} = 512$. Calculate: (12 marks)

- Dimension of each head: $d_k = d_v = ?$
- Total number of parameters in all projection matrices
- Memory requirements for storing attention matrices
- Computational complexity compared to single-head attention

Answer: Multi-head attention with 8 heads and 512-dimensional embeddings, showing parameter count and complexity analysis.

Multi-Head Attention Design for $h = 8$, $d_{model} = 512$:

1. Dimension of Each Head:

$$d_k = d_v = \frac{d_{model}}{h} = \frac{512}{8} = 64$$

Rationale: Equal split ensures total dimension remains d_{model} after concatenation.

2. Parameter Count Calculation:

Per Head Parameters:

- $W_Q^{(i)} \in \mathbb{R}^{512 \times 64}$: 32,768 parameters
- $W_K^{(i)} \in \mathbb{R}^{512 \times 64}$: 32,768 parameters
- $W_V^{(i)} \in \mathbb{R}^{512 \times 64}$: 32,768 parameters
- Total per head: $3 \times 32,768 = 98,304$ parameters

All Heads: $8 \times 98,304 = 786,432$ parameters

Output Projection: $W_O \in \mathbb{R}^{512 \times 512}$: 262,144 parameters

Total Parameters: $786,432 + 262,144 = 1,048,576$ parameters

3. Memory Requirements (for sequence length n):

Per Head:

- Q, K, V matrices: $3 \times n \times 64$ values
- Attention matrix: $n \times n$ values
- Output: $n \times 64$ values

All Heads:

- QKV storage: $8 \times 3 \times n \times 64 = 1,536n$ values
- Attention matrices: $8 \times n^2 = 8n^2$ values
- Head outputs: $8 \times n \times 64 = 512n$ values
- **Total:** $8n^2 + 2,048n$ **values**

4. Computational Complexity vs Single-Head:

Single-Head Attention ($d_k = 512$):

- QKV projections: $O(n \times 512^2) = O(262,144n)$
- Attention computation: $O(n^2 \times 512)$

Multi-Head Attention (8 heads, $d_k = 64$ each):

- QKV projections: $O(8 \times n \times 512 \times 64) = O(262,144n)$ (same!)
- Attention computation: $O(8 \times n^2 \times 64) = O(512n^2)$ (same!)
- Output projection: $O(n \times 512^2) = O(262,144n)$

Key Insight: Multi-head attention has the same computational complexity as single-head attention but provides much richer representations through parallel attention patterns.

(b) Implement the multi-head attention algorithm in pseudocode. Include: (10 marks)

- Input preprocessing
- Parallel head computation
- Output concatenation and projection
- Masking for causal attention

Answer: Complete multi-head attention algorithm with masking support.

Multi-Head Attention Algorithm:

function MultiHeadAttention(X, $mask = None$)
Input: $X \in \mathbb{R}^{n \times d_{model}}$, optional mask
Output: $Z \in \mathbb{R}^{n \times d_{model}}$

// **Input Preprocessing**
$n, d_{model} \leftarrow$ shape(X)
$d_k \leftarrow d_{model}/h$

// **Initialize head outputs list**

head_outputs $\leftarrow$ []

// **Parallel Head Computation**
for $i = 1$ to h **do**
  // Project to Q, K, V for head i
  $Q^{(i)} \leftarrow X \cdot W_Q^{(i)}$ $\{\mathbb{R}^{n \times d_k}\}$
  $K^{(i)} \leftarrow X \cdot W_K^{(i)}$ $\{\mathbb{R}^{n \times d_k}\}$
  $V^{(i)} \leftarrow X \cdot W_V^{(i)}$ $\{\mathbb{R}^{n \times d_k}\}$

  // Compute scaled dot-product attention
  scores $\leftarrow \frac{Q^{(i)} \cdot K^{(i)T}}{\sqrt{d_k}}$ $\{\mathbb{R}^{n \times n}\}$

  // **Apply Masking (if provided)**
  if $mask \neq None$ **then**
    scores $\leftarrow$ scores + mask $\times (-\infty)$
  end if

  // Softmax normalization
  attn_weights $\leftarrow$ softmax(scores) {along last dim}

  // Weighted aggregation
  head_output $\leftarrow$ attn_weights $\cdot V^{(i)}$ $\{\mathbb{R}^{n \times d_k}\}$
  head_outputs.append(head_output)
end for

// **Output Concatenation and Projection**
concatenated $\leftarrow$ concat(head_outputs) $\{\mathbb{R}^{n \times d_{model}}\}$
$Z \leftarrow$ concatenated $\cdot W_O$ {Final projection}

return Z

Masking Types:

1. Causal Mask (for decoder):

- Upper triangular matrix with $-\infty$ values
- Prevents attention to future positions
- $\text{mask}[i, j] = -\infty$ if $j > i$, else 0

2. Padding Mask:

- Masks out padding tokens
- $\text{mask}[i, j] = -\infty$ if position j is padding, else 0

Implementation Notes:

- Can be vectorized for efficiency
- Gradient computation handled automatically by autodiff
- Memory optimization: compute attention heads sequentially if memory-constrained

(c) Analyze why multiple attention heads capture different types of relationships. Provide examples of what different heads might learn in language modeling. (8 marks)

Answer: Multiple heads learn specialized attention patterns capturing diverse linguistic relationships through different Q, K, V projections and training dynamics.

Why Multiple Heads Capture Different Relationships:

1. Different Parameter Initialization:

- Each head has independent $W_Q^{(i)}$, $W_K^{(i)}$, $W_V^{(i)}$ matrices
- Random initialization leads to different gradient flows
- Heads evolve to minimize different aspects of the loss

2. Representation Subspace Specialization:

- Each head projects embeddings to different d_k-dimensional subspace
- Different subspaces can capture orthogonal linguistic features
- Example: syntactic vs semantic subspaces

3. Training Dynamics and Competition:

- Heads compete to provide useful signals
- Natural specialization emerges to minimize redundancy
- Different heads find different optima in parameter space

Examples of Learned Attention Patterns:

Head 1 - Syntactic Dependencies:

- Subject-verb agreement: "The cats [MASK] running"
- High attention from "cats" to "[MASK]" for verb prediction
- Captures grammatical number agreement