Comparison of Attention Mechanisms: From Vanilla to Linear Topological Masking

1 Simple Attention (Vanilla Softmax)

This is the standard attention mechanism found in the original Transformer. It treats the input as a set (or sequence) and is permutation invariant regarding graph structure.

Formulas and Mechanism

Given Query (Q), Key (K), and Value (V) matrices of size $N \times d$:

$$\text{Att}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d}}\right)V \tag{1}$$

What it concretely does:

- It computes a similarity score (dot product) between *every* pair of tokens.
- It normalizes these scores using Softmax so they sum to 1.
- It computes a weighted sum of Value vectors based on these scores.
- **Critique:** It is unaware of the graph topology. Node i attends to Node j purely based on feature similarity, ignoring whether they are neighbors or far apart.

Time Complexity Derivation

The bottleneck is the explicit computation of the attention matrix $A = QK^\top$.

1. Matrix multiplication $Q \times K^\top$: $(N \times d) \times (d \times N) \to (N \times N)$.
2. Complexity: $\mathcal{O}(N^2 d)$. Since d is usually small constant, we write $\mathcal{O}(N^2)$.
3. Space Complexity: We must store the $N \times N$ matrix in memory.

---

2 Softmax Attention with Topological Masking

This is the "Previous Solution" mentioned in the script. We explicitly inject graph structure into the attention scores.

Formulas and Mechanism

We calculate a topological mask $M(\mathcal{G})$ (e.g., using a power series of the adjacency matrix W):

$$M_\alpha(\mathcal{G}) := \sum_{k=0}^{\infty} \alpha_k W^k \tag{2}$$

We then inject this into the attention mechanism using the Hadamard (element-wise) product $\odot$:

$$\text{Att}_{\text{masked}} = \text{softmax}\left(\frac{QK^\top}{\sqrt{d}} + \log(M_\alpha)\right)V \quad \text{or} \quad D^{-1}\left(\exp\left(\frac{QK^\top}{\sqrt{d}}\right) \odot M_\alpha\right)V \tag{3}$$

What it concretely does:

- It forces the attention scores to respect the graph. If nodes i and j are far apart, $M_{ij} \approx 0$, effectively killing the attention between them.
- It introduces a strong structural inductive bias.

Time Complexity Derivation

Despite the benefits, the complexity remains quadratic.

1. We must still compute the full $N \times N$ attention matrix $QK^\top$.
2. We must also compute/store the dense $N \times N$ mask matrix M.
3. The element-wise product is performed on N^2 entries.
4. Total Complexity: $\mathcal{O}(N^2)$.

---

3 Linear Attention (Vanilla)

This is the optimization that solves the memory issue but loses the masking capability.

Formulas and Mechanism

We replace the Softmax kernel with a feature map $\phi(\cdot)$ (e.g., ReLU or Random Features). This allows us to use the associativity of matrix multiplication:

$$\text{Att}_{\text{linear}}(Q, K, V) = \phi(Q)\left(\phi(K)^\top V\right) \tag{4}$$

What it concretely does:

- Instead of computing $(Q \times K^\top) \times V$, it computes $Q \times (K^\top \times V)$.
- It avoids ever creating the $N \times N$ attention matrix.
- **Critique:** It effectively assumes an "all-to-all" attention (or a specific kernel approximation), but we cannot easily insert the topological mask M because $(A \times B) \odot M \neq A \times (B \odot M)$.

Time Complexity Derivation

1. First, compute $\phi(K)^\top V$: Dimensions are $(d \times N) \times (N \times d) \to (d \times d)$. Complexity: $\mathcal{O}(Nd^2)$.
2. Second, compute $\phi(Q) \times (\text{Result})$: Dimensions are $(N \times d) \times (d \times d) \to (N \times d)$. Complexity: $\mathcal{O}(Nd^2)$.
3. Total Complexity: $\mathcal{O}(N)$ (linear with respect to sequence length N).

---

4 Linear Attention with GRF Topological Masking

This is the paper's main contribution. It combines the $\mathcal{O}(N)$ speed of Linear Attention with the structural bias of Topological Masking.

Formulas and Mechanism

We decompose the mask M into Graph Random Features (GRFs) $\hat{\phi}_{\mathcal{G}}$ via random walks, such that $M_{ij} \approx \hat{\phi}_{\mathcal{G}}(v_i)^\top \hat{\phi}_{\mathcal{G}}(v_j)$. We merge semantic features $\phi(q)$ and graph features $\hat{\phi}_{\mathcal{G}}$ using the outer product:

$$\Phi_{\text{new}}(i) = \text{vec}\left(\phi(q_i) \otimes \hat{\phi}_{\mathcal{G}}(v_i)\right) \tag{5}$$

The attention becomes:

$$\text{Att}_{\text{GRF}} = \Phi_{\text{new}}(Q)\left(\Phi_{\text{new}}(K)^\top V\right) \tag{6}$$

What it concretely does:

- It creates a "super-feature" that contains both the token's content and its position in the graph (via random walk collision probability).
- It allows us to use the linear associativity trick again, because we "baked" the mask into the features themselves.

Time Complexity Derivation

At first glance, $\hat{\phi}_{\mathcal{G}}$ is size N, making the feature vector size $N \cdot d$, which would imply quadratic cost. However, **Theorem 3.2** saves us:

1. **Sparsity:** Because random walks halt with probability p_{halt}, the number of non-zero entries in $\hat{\phi}_{\mathcal{G}}$ is a constant $C \ll N$.
2. The sparse matrix multiplication cost depends on non-zero entries (NNZ).
3. Computing $\Phi_{\text{new}}(K)^\top V$: The number of operations is proportional to $N \times C \times d$.
4. Since C and d are constants relative to N, the complexity is $\mathcal{O}(N)$.

Summary of Differences

Method	Time Complexity	Uses Graph Topology?	Mask Implementation
1. Vanilla Softmax	$\mathcal{O}(N^2)$	No	None
2. Masked Softmax	$\mathcal{O}(N^2)$	**Yes**	Hadamard Product ($\odot$)
3. Vanilla Linear	$\mathcal{O}(N)$	No	None
4. **GRF Linear (Ours)**	$\mathcal{O}(N)$	**Yes**	**Feature Tensor Product ($\otimes$)**

Table 1: Comparison of attention mechanisms. The GRF method is the only one that achieves both linear complexity and topological awareness.