**Probsparse Attention:**

**In traditional attention mechanism. In standard attention:**

* **Given a query Q, key K, and value V matrices**
* **Attention(Q, K, V) = softmax(QK^T / √d) \* V**

**Where d is the dimension of the key vectors.**

**A screenshot of a computer

Description automatically generated**

**A diagram of a mathematical equation

Description automatically generated**

**A diagram of a diagram

Description automatically generated**

**One of the Problem with Attention Mechanism is As sequence lengths grow, the quadratic complexity becomes a significant bottleneck. This is especially problematic for long sequences or large datasets.**

**ProbSparse attention aims to reduce this complexity by only computing attention for a subset of query-key pairs. It's based on the idea that not all attention connections are equally important.**

**ProbSparse attention introduces a sparsity measurement called "Top-u" attention sparsity. For each query, it only computes attention with the top-u keys that are most likely to have high attention scores.**

**The method assumes that the attention scores follow a certain probability distribution. Typically, this is assumed to be a long-tailed distribution like the power-law distribution.**

**Query Sparsity Measurement**

**For each query q\_i, we compute a measure M(q\_i) to determine its sparsity:**

**M(q\_i) = max(q\_i \* K^T) - mean(q\_i \* K^T)**

**Where max and mean are computed across all keys for a given query.**

**We select the top-u queries based on their M(q\_i) scores. Let's call this set of queries Q\_u.**

**For the selected queries Q\_u, we compute full attention:**

**A\_sparse = softmax(Q\_u \* K^T / √d) \* V**

**For the remaining queries, we use a simpler aggregation method, often just mean pooling:**

**A\_dense = mean(V)**

**The final output is a combination of A\_sparse and A\_dense, properly masked to ensure each query gets exactly one output vector.**

**So Overall,**

**Input could be:**

** Query Matrix (Q): Shape [batch\_size, seq\_length, d\_model]**

** Key Matrix (K): Shape [batch\_size, seq\_length, d\_model]**

** Value Matrix (V): Shape [batch\_size, seq\_length, d\_model]**

** Sparsity parameter (u): A scalar value determining how many queries will use full attention.**

**Where:**

* **batch\_size: Number of sequences in a batch**
* **seq\_length: Length of each sequence**
* **d\_model: Dimension of the model (size of each vector)**

**Process**

1. **Sparsity Measurement:** 
   * **For each query, compute a measure of how likely it is to produce high attention scores.**
   * **Select the top-u queries based on this measure.**
2. **Sparse Attention Computation:** 
   * **For the selected top-u queries:** 
     + **Compute full attention: softmax(Q\_selected \* K^T / √d) \* V**
   * **For the remaining queries:** 
     + **Use a simpler aggregation (often mean pooling of V)**
3. **Combine Results:** 
   * **Merge the results from full attention and simple aggregation.**

**Output**

* **Output Matrix: Shape [batch\_size, seq\_length, d\_model]** 
  + **Each row corresponds to an input query**
  + **For top-u queries: output is the result of full attention computation**
  + **For other queries: output is the result of simple aggregation.**

**Let's consider a simple example with a short sequence of words.**

**Input**

**Sequence: "The cat sat on the mat"**

* **Query (Q), Key (K), and Value (V) matrices are derived from this sequence**
* **Each word is represented by a vector of dimension 4**
* **Sparsity parameter u = 2 (we'll compute full attention for the top 2 queries)**

**Q, K, V shapes: [1, 6, 4] (1 batch, 6 words, 4-dimensional vectors)**

**Process**

1. **Sparsity Measurement:** 
   * **Compute the sparsity measure for each query (word)**
   * **Let's say we get these scores: "The": 0.5, "cat": 1.2, "sat": 0.8, "on": 0.3, "the": 0.4, "mat": 1.0**
2. **Select Top Queries:** 
   * **Top 2 scores: "cat" (1.2) and "mat" (1.0)**
3. **Attention Computation:** 
   * **For "cat" and "mat": Compute full attention with all words**
   * **For other words: Use mean pooling**

**Output**

**Output shape: [1, 6, 4] (same as input)**

* **"cat" and "mat": Outputs are weighted combinations of all word vectors, based on attention scores**
* **Other words: Outputs are the mean of all word vectors**

**Example output vectors (conceptually):**

* **"The": [mean vector]**
* **"cat": [weighted combination based on attention]**
* **"sat": [mean vector]**
* **"on": [mean vector]**
* **"the": [mean vector]**
* **"mat": [weighted combination based on attention]**

**Key Points**

1. **Only "cat" and "mat" got full attention computation**
2. **Other words used a simpler, less computationally expensive operation**
3. **The output maintains the original sequence length and structure**
4. **Computational savings increase with longer sequences and smaller u values**