CS 15-851: Algorithms for Big Data Spring 2025

Lecture 11 Part 2 — April 10, 2025

Prof. David Woodruff *Scribe: Nairit Sarkar*

1 Recent Efficiency Improvements to Transformers

1.1 Autoregressive Language Model

An Autoregressive Language Model (ALM) takes in a sequence of tokens and produces a new token. We can view tokens as words and an example of an ALM is ChatGPT.

The general setup is this: we have an *input prompt* (sequence of tokens). Then, the input prompt can be efficiently converted into a *vector embedding* by converting each token into a d-dimensional vector. So, if we have n tokens in the input prompt and the vector embedding has dimension 128, our vector embedding is a $n \times 128$-dimensional matrix. We treat the dimension of the vector embedding (128 in this case) as a constant. Then, the vector embedding is fed into a *transformer* which returns an output embedding. Finally, the output embedding is converted back into tokens to generate an output.

A transformer is a very powerful component in the autoregressive model with applications in audio, video, and text processing. In a transformer, we have four layers: a normalization layer, the self-attention layer, another normalization layer, and a feedforward layer. We restrict our focus to the self-attention layer since all other layers take linear time.

Given an input embedding $X \in \mathbb{R}^{n \times d}$, the *self-attention layer* works as follows: we store and learn $W^Q, W^K, W^V \in \mathbb{R}^{d \times d}$. Then we let

$$\begin{aligned} Q &= X \cdot W^Q \in \mathbb{R}^{n \times d} \\ K &= X \cdot W^K \in \mathbb{R}^{n \times d} \\ V &= X \cdot W^V \in \mathbb{R}^{n \times d} \\ A &= \exp(QK^\top) \in \mathbb{R}^{n \times n} \\ D_{i,i} &= \sum_{j=1}^{n} A_{ij} \end{aligned}$$

where D is diagonal and return $D^{-1}AV \in \mathbb{R}^{n \times d}$. We define $Att(Q, K, V) = D^{-1}AV = \text{softmax}(QK^\top)V$.

The *softmax* operation is done by rows: for a given row $r = [r_1, r_2, \ldots, r_n]$, we have

$$\text{softmax}(r) = \left[\frac{\exp(r_1)}{\sum \exp(r_i)}, \frac{\exp(r_2)}{\sum \exp(r_i)}, \ldots, \frac{\exp(r_n)}{\sum \exp(r_i)}\right].$$

Intuitively, the rows of X are tokens and $X_i \cdot W^Q$ is the *query* for token X_i (where X_i is a row of matrix X). Then, the softmax operation on row i generates a probability distribution for the keys

most likely to *attend* to query i. We multiply this matrix by W^V to update the embeddings of each token.

Unfortunately, computing $Att(Q, K, V) = D^{-1}AV$ requires $O(n^2)$ time causing the self-attention layer to be the bottleneck when n is large. Given how powerful transformers are, we aim to optimize this computation.

Specifically, we can compute Q, K, V in $O(n)$ time (since we view d as a constant). However, we need $O(n^2)$ time to compute the following:

- $A = \exp(QK^\top)$,
- D when given A, and
- $D^{-1}AV$ when given A, D, V.

Furthermore, we need $O(n^2)$ time to compute a high quality $O(1/poly(n))$-approximation of $Att(Q, K, V)$:

Theorem 1 (Alman & Song 23'). *If $d \in O(\log n)$, $B = \Theta(\sqrt{\log n})$, and $Q, K, V \in [-B, B]^{n \times d}$ then assuming the Strong Exponential Time Hypothesis, it is impossible to approximate $Att(Q, K, V)$ up to $1/poly(n)$ additive error in subquadratic time $O(n^{2-\Omega(1)})$.*

This is not satisfactory in practice so we must make some assumptions. One possible way is to assume that the large entries are near the diagonal. However, this is a strong assumption.

Another approach is to use low-rank approximation. Suppose we had some algorithm that could take K, Q and produce $LR \approx softmax(QK^T)$ where $L \in n \times k, R \in k \times n$ with $k << n$. Then, we could compute $\text{softmax}(QK^T)V \approx LRV$ in nkd time since each multiplication takes nkd time. The challenge here is computing L, R efficiently.

1.2 Hyperattention

We split the computation of $Att(Q, K, V)$ into two steps:

- Approximate
$$D_{i,i} = \sum_{j \in [n]} A_{i,j} = \sum_{j \in [n]} \exp(\langle q_i, k_j \rangle)$$
- Approximate AV.

Let's start with the second. One approach is sketching: $AV \approx ASS^\top V$ where $S \in \mathbb{R}^{m \times n}$ is a sketching matrix. However, A is a dense matrix so computing AS requires $O(n^2)$ time. We can fix this by sampling. Specifically, we sample row i with probability $\propto \|v_i\|_2^2$. This is lower-variance than random sampling and gives $m = \text{srank}(\text{softmax}(QK^\top) \cdot d$.

Now let's address the first. One general approach we can use is splitting up the sum $D_{i,i}$ into "heavy" and "light" elements, explicitly computing the heavy sum, and estimating the light sum. This does better than naive random sampling since most of the variance will be in the heavy elements and we compute that exactly.

Theorem 2 (Informal)**.** *If the maximum squared column norm in softmax$(QK^\top)$ is $\frac{1}{n^{1-o(1)}}$ and the ratio of max and min row sums in $A = exp(QK^\top)$ after removing heavy elements is $n^{o(1)}$ then Att can be computed in $O(dn^{1+o(1)})$ time with*

$$\left\|softmax(QK^\top)V - Att\right\|_{op} \leq \epsilon \left\|softmax(QK^\top)\right\|_{op} \|V\|_{op}$$

The column norm bound is non-trivial - it allows for entries as large as $\frac{1}{n^{1/2-o(1)}}$ in softmax$(QK^\top)$. Estimating the contribution of light elements is non-trivial as well.

Let's see why the column norm bound is needed. If $\exp(QK^\top)$ has the first column all n and at most two n's in any row and all other entries 1, i.e.

$$\exp(QK^\top) = \begin{bmatrix} n & 1 & n & 1 & \dots & 1 \\ n & 1 & 1 & 1 & \dots & n \\ n & \vdots & \vdots & \vdots & \ddots & \vdots \\ n & 1 & 1 & n & \dots & 1 \end{bmatrix},$$

then note that $D_{i,i} = 2n$ or $3n$ so the norm of the first column is at least $(1/3)^2 \cdot n \in O(n)$. So the column norm bound condition does not hold.

Then, if $V = [1, 0, \dots, 0]$ (i.e. first unit vector), $\left\|\text{softmax}(QK^\top)\right\|_{op}^2 \approx n$, $\|V\|_2^2 = 1$, and $\left\|\text{softmax}(QK^\top)V - Att\right\|_{op}^2 \leq n/10$.

This has hardness $n^{2-o(1)}$ and bounded column norms avoid hard instances like these.

1.2.1 Finding Heavy Elements in Practice

One practical approach to finding heavy elements is using a permutation algorithm to bring the heavy elements near the main diagonal. Note that if $A_{ij} = \langle q_i, k_j \rangle$ is large, q_i and k_j are directionally similar in $\mathbb{R}^n$. The *Hamming sorted LSH* provides a hash function $\mathcal{H} : \mathbb{R}^d \to [B]$ such that $\Pr_{\mathcal{H}}[\mathcal{H}(q) = \mathcal{H}(k)]$ is roughly proportional to $\langle q, k \rangle$. Furthermore, the buckets are ordered in such a way that geometrically adjacent buckets have consecutive buckets [23', Han, Jayaram, Karbasi, Mirrokni, Woodruff, Zandieh].

Specifically, SimHash normalizes the points onto the hypersphere, and if we have 2^v buckets, we can create a binary representation of a point by choosing v hyperplanes and assigning the ith bit to be 1 or 0 depending on whether the point is above or below the ith hyperplane. SortLSH then creates buckets such that close bit strings are in the same bucket. Finally, we can use our hash to create a permutation matrix that brings heavy entries near the diagonal.

1.2.2 Causal Masking

The method described above supports causal masking: output embeddings only depend on past input embeddings so the causal masked version of this problem sets all entries above the main diagonal of $QK^\top$ to zero (before applying softmax). We can use a divide and conquer algorithm to find heavy elements in the causal masking variant: the top right submatrix has no heavy elements,

we can use the above method on the bottom left submatrix and call the recursive function on the other two submatrices. This relies on the assumption that submatrices will still satisfy our assumptions.

1.3 PolySketchFormer

Let $\text{sim}(q, k) \geq 0$ be an arbitrary function that measures similarity between query q and key k. Then, the *attention mechanism* with respect to sim is

$$o_j = \sum_{i \leq j} \frac{\text{sim}(q_j, k_i)}{\sum_{i' \leq j} \text{sim}(q_j, k_{i'})} v_i.$$

Suppose ϕ is a function such that $\text{sim}(q, k) = \langle \phi(q), \phi(k) \rangle$. If $Q' = \phi(Q)$ and $K' = \phi(K)$, the output is

$$D^{-1} \cdot LT(Q' \cdot (K')^\top) \cdot V.$$

Here, LT is the lower triangular part for the causal setting. This gives a linear time algorithm for computing $LT(A \cdot B^T) \cdot C$ since runtime depends on output dimension of ϕ. Unfortunately, there are no finite dimensional feature maps ϕ for softmax.

Performer (Choromanski et al.,) uses a finite-dimensional map ϕ to approximate the exponential. However, vectors with large norms require a larger output dimension.

Alternatively, we may consider arbitrary ϕ instead of first defining the sim function. We can consider $\phi(x) = elu(x) + 1$ (Katharopoulos et al. '20) or $\phi(x) = relu(x)$ but these result in worse model quality compared to softmax.

1.3.1 Polynomial Feature Map

Consider $\text{sim}(q, k) = \langle q, k \rangle^p$ where $p \geq 2$ is an even integer. We can achieve this by letting $\phi : x \to x^{\otimes p}$. If $x \in \mathbb{R}^h$, $x^{\otimes p} \in \mathbb{R}^{h^p}$ and $(x^{\otimes p})_{i_1, i_2, \ldots, i_p} = x_{i_1} \cdot \ldots \cdot x_{i_p}$. Then,

$$\langle \phi(q), \phi(k) \rangle = \langle q^{\otimes p}, k^{\otimes p} \rangle = \langle q, k \rangle^p$$

as desired.

This allows linear attention using polynomials: we can compute $Q^{\otimes p}$ and $K^{\otimes p}$ in order to compute

$$LT(Q^{\otimes p} \cdot (K^{\otimes p})^\top) \cdot V$$

in $O(nh^{p+1})$ time. Typically, $h = 64, 128, 256$ so this method is too expensive even for $p = 4$.

We can use sketching to approximate. Since $Q^{\otimes p}, K^{\otimes p}$ have a large number of columns, we wish to compute matrices Q', K' such that $Q^{\otimes p}(K^{\otimes p})^\top \approx Q' \cdot (K')^\top$. Ahle et al. '20 give a fast sketch called TensorSketch.

Unfortunately, this does not converge since the entries of $Q^{\otimes p}(K^{\otimes p})^\top$ are nonnegative whereas $Q' \cdot (K')^\top$ may have negative entries. The following theorem addresses this issue:

Theorem 3. *If Q', K' are sketches for degree $p/2$ then $Q'^{\otimes 2}, K'^{\otimes 2}$ is a sketch for degree p.*

Some other optimizations we may make:

- TensorSketch is a random sketch - we can instead treat the sketch as a learnable parameter
- When computing $LT(A \cdot B^\top) \cdot C$ use block multiplication and cumulative sums
- Compute diagonal blocks exactly as such blocks are sensitive to approximation