`max_of_k` Heuristic Estimator

Gabriel Wu

January, 2024

Code can be found here.

1 Overview

We're given a model M that takes in k integers $x_1, \ldots, x_k$ (each in the range $[n] = \{1, \ldots, n\}$) and outputs a probability distribution over $[n]$. It has been trained to output the maximum of its inputs. The *soft accuracy* of the model is the expected probability it places on the correct answer on a random input. For simplicity, I define the input distribution as all $n!/(n-k)!$ inputs in which all x_i are distinct.

We are interested in finding a heuristic estimator $\mathbb{G}$ that given the proper argument π outputs a reasonable estimate for the soft accuracy of M. I've found it conceptually confusing to distinguish $\mathbb{G}$ from π in this setting,[1] so I'll just treat them as one joint object for now.

2 Model Architecture

The model is a one-layer attention-only transformer with no bias (except in positional encoding). It has a single attention head and hidden dimension d.

The model comes with the following matrices:

Name	Dimension	Description
$\mathbf{P}$	$k \times d$	Positional encoding
$\mathbf{E}$	$n \times d$	Embedding matrix
$\mathbf{Q}$	$d \times d$	Query matrix
$\mathbf{K}$	$d \times d$	Key matrix
$\mathbf{V}$	$d \times d$	Value matrix
$\mathbf{O}$	$d \times d$	Output matrix
$\mathbf{U}$	$d \times n$	Unembedding matrix

The output of the model is defined as follows.

[1] In some sense, my π can be thought of as $O(1)$ information outlining to $\mathbb{G}$ how to set up the algebraic rewrite, possibly followed by walking $\mathbb{G}$ through all of the computations. Notably, this is different from Jacob's setting where π has the clean interpretation of a specific set of directions upon which to do covariance propagation.

1. Let $\mathsf{input} \in \mathbb{R}^{k \times n}$ be the one-hot encoding of the input
2. $\mathsf{res} \leftarrow \mathsf{input} \cdot \mathbf{E} + \mathbf{P}$ (dimension $k \times d$)
3. $\mathsf{attn\_presoft} \leftarrow \mathsf{res} \cdot \mathbf{Q} \cdot \mathbf{K}^T \cdot \mathsf{res}^T$ (dimension $k \times k$)
4. $\mathsf{attn\_prob} \leftarrow \text{masked\_softmax}(\mathsf{attn\_presoft})$ (dimension $k \times k$)
5. $\mathsf{attn} \leftarrow \mathsf{attn\_prob} \cdot \mathsf{res} \cdot \mathbf{V} \cdot \mathbf{O}$ (dimension $k \times d$)
6. $\mathsf{logits} \leftarrow (\mathsf{res} + \mathsf{attn}) \cdot \mathbf{U}$ (dimension $k \times n$)
7. $\mathsf{output} \leftarrow \text{softmax}(\mathsf{logits}_{k,1}, \ldots, \mathsf{logits}_{k,n})$ (dimension n)
8. return output

Note that after step 2, only the values at k-th token position end up mattering for the final output.

The particular model I worked with had $k = 10$, $n = 64$, and $d = 32$. When doing any asymptotic analysis of the algorithm, I will treat $k = o(n)$ and $n = \Theta(d)$.

3 Heuristic Estimator

3.1 Outline

Let the input be $x_1, \ldots, x_k$, where each $x_i \in [n]$. Let $m = \text{argmax}_i x_i \in [k]$ be the token position that achieves the maximum value.

The heuristic estimator actually produces n different estimates: the soft accuracy of the model conditional on the correct answer being v, for every $v \in [n]$. Call these estimates $A_1, \ldots, A_n$. Its overall estimate of the soft accuracy of the model is $\sum_{v=1}^{n} A_i \Pr[x_m = v]$.

For any v, the estimate A_v is the result of the following process (all distributions are conditional on the event $x_m = v$):

1. Condition on x_k. Estimate the distributions of $\mathsf{attn\_presoft}_{k,m}$ and $\mathsf{attn\_presoft}_{k,i \sim [k] \setminus \{m\}}$.
2. Still conditioning on x_k, estimate the distributions of $\mathsf{attn\_prob}_{k,m}$ and $\mathsf{attn\_prob}_{k,i \sim [k] \setminus \{m\}}$. Then marginalize over all choices of x_k.
3. For each $\ell \in [n]$, estimate the distribution of $\mathsf{logits}_{k,\ell}$.
4. Estimate the distribution of output_v. The mean of this distribution is A_v.

Steps 1 and 2 are described in section 3.3, and steps 3 and 4 are described in section 3.4.

3.2 Bin Propagation

Throughout the argument I need to track the distributions of real-valued random variables. I do this by "binning" a distribution into b equally-probable bins and storing the mean of each bin. For example when $b = 10$ (which is what I normally use) I keep track of 10 values: the mean of the i-th decile of the distribution for each i from 1 to 10.

There are straightforward algorithms for applying functions to binned distributions and taking mixtures.

- If I have the binned distribution of a random variable X and want to estimate the binned distribution of $f(X)$, I simply apply f to the mean of each bin and sort the resulting list.
- If I have the binned distributions of X and Y and want to estimate the binned distribution of a bivariate function $f(X, Y)$, I first assume X and Y are independent, then compute the b^2 evaluations of f on all pairs of bin means. I sort this list of evaluations and group them into b new bins (each containing b points), storing their means.
- If I want to estimate the mixture distribution of X w.p. p and Y w.p. $1-p$, I generate a weighted list of length $2b$, sort the list, then greedily generate a binned distribution from that.
- Finally, I often have the binned distribution of X and wish to estimate the binned distribution of $X_1 + X_2 + \cdots + X_n$, where each X_i is drawn i.i.d. from the distribution of X (for example, this comes up in the denominator of the softmax operation). Instead of doing n iterative convolutions (this would take time $O(nb^2 \log b)$), I use the "square and multiply" trick, taking $O(b^2 \log b \log n)$ time.[2]

Note that mean propagation is equivalent to bin propagation when $b = 1$.

3.3 Estimating Attention

Let $t_i = \mathsf{input}_i \in \mathbb{R}^n$ be the one-hot encoding of $x_i \in [n]$. For any vector $X \in \mathbb{R}^a$, let $\text{stack}_b(X) \in \mathbb{R}^{b \times a}$ be the matrix formed by b copies of X stacked on top of each other.

Notice that we can write

$$\begin{aligned}
\mathsf{attn_presoft}_{k,i} &= (t_k \cdot \mathbf{E} + \mathbf{P}_k) \cdot \mathbf{Q} \cdot \mathbf{K}^T \cdot (t_i \cdot \mathbf{E} + \mathbf{P}_i)^T \\
&= t_k \cdot (\mathbf{E} + \text{stack}_n(\mathbf{P}_k))\mathbf{Q}\mathbf{K}^T\mathbf{E}^T \cdot t_i^T + t_k \cdot (\mathbf{E} + \text{stack}_n(\mathbf{P}_k))\mathbf{Q}\mathbf{K}^T(\mathbf{P}_i)^T \\
&= \mathsf{EQKE}_{x_k,x_i} + \mathsf{EQKP}_{x_k,i}
\end{aligned}$$

where we've defined the matrices

$$\begin{aligned}
\mathsf{EQKE} &:= (\mathbf{E} + \text{stack}_n(\mathbf{P}_k))\mathbf{Q}\mathbf{K}^T\mathbf{E}^T \\
\mathsf{EQKP} &:= (\mathbf{E} + \text{stack}_n(\mathbf{P}_k))\mathbf{Q}\mathbf{K}^T\mathbf{P}^T.
\end{aligned}$$

[2]Equivalently, decompose the sum into a balanced binary tree instead of a daisy chain. The $\log b$ factor comes from sorting.

Thus, after precomputing these two matrix products (which takes $O(n^3)$), we can exactly calculate $\mathsf{attn\_presoft}_{k,i}$ in constant time.

Now, condition on a particular value of x_k (and recall that we're also already conditioning on $x_m = v$). We can analytically compute the distribution of m (there are only two cases: $x_k < x_m$ and $x_k = x_m$), which allows us to compute the (binned) distribution of $\mathsf{attn\_presoft}_{k,m}$. We also wish to compute the distribution, over a random choice of $i \neq m$, of $\mathsf{attn\_presoft}_{k,i}$. We know that x_i is distributed uniformly over $[x_m - 1]$, so the distribution of the first term (EQKE_{x_k,x_i}) can be computed[3]. We can also analytically compute the distribution of i (again, $x_k < x_m$ and $x_k = x_m$ are handled separately), giving us the distribution of the second term $\mathsf{EQKP}_{x_k,i}$. Convolving these two terms gives us the distribution of $\mathsf{attn\_presoft}_{k,i\sim[k]\setminus\{m\}}$. All of the distributions in this step are exact, modulo binning error.

Using the distributions of $\mathsf{attn\_presoft}_{k,m}$ and $\mathsf{attn\_presoft}_{k,i\sim[k]\setminus\{m\}}$, we next compute the distributions of $\mathsf{attn\_prob}_{k,m}$ and $\mathsf{attn\_prob}_{k,i\sim[k]\setminus\{m\}}$. This is the first time we use the presumption of independence. We have:

$$\mathsf{attn\_prob}_{k,m} = \frac{\exp(\mathsf{attn\_presoft}_{k,m})}{\exp(\mathsf{attn\_presoft}_{k,m}) + \sum_{i\neq m}\exp(\mathsf{attn\_presoft}_{k,i})}.$$

We approximate this as the distribution of

$$\frac{X}{X + \sum_{i=1}^{k-1} Y_i},$$

where all X, Y_i are independent, X is distributed like $\exp(\mathsf{attn\_presoft}_{k,m})$, and each Y_i is distributed like $\exp(\mathsf{attn\_presoft}_{k,i\sim[k]\setminus\{m\}})$. Computationally, this involves a sum of $k-1$ i.i.d. variables (which takes $O(\log k)$ pairwise convolutions) followed by applying the bivariate function $f(a,b) = \frac{a}{a+b}$. We handle $\mathsf{attn\_prob}_{k,i\sim[k]\setminus\{m\}}$ similarly: it can be expressed as $\frac{Z}{X+Z+\sum_{i=1}^{k-2} Y_i}$.

We now have approximations for the distributions of $\mathsf{attn\_prob}_{k,m}$ and $\mathsf{attn\_prob}_{k,i\sim[k]\setminus\{m\}}$, conditional on x_m and x_k. We marginalize over all choices of x_k by taking a weighted mixture of these distributions, with weights given by $\Pr[x_k = v' | x_m = v]$ (which we can compute exactly). The final result of this phase is the distributions of $\mathsf{attn\_prob}_{k,m}$ and $\mathsf{attn\_prob}_{k,i\sim[k]\setminus\{m\}}$ conditional on $x_m = v$ and nothing else.

[3]Naively, computing this distribution takes $O(n)$ time because it would require iterating through $x_m - 1$ matrix entries. We speed this up by expressing the distribution $\mathsf{EQKE}_{x_k,[x]}$ as a mixture of $\mathsf{EQKE}_{x_k,[x-1]}$ and (the constant) $\mathsf{EQKE}_{x_k,x}$, then building them up cumulatively. This gives us amortized constant-time access to the distribution (ignoring binning overhead).

3.4 Estimating Logits

The next step is to estimate the distribution of $\mathsf{logits}_{k,\ell}$ for every $\ell \in [n]$. We have:

$$\begin{aligned}
\mathsf{logits}_{k,\ell} &= [(\mathsf{attn} + \mathsf{res}) \cdot \mathbf{U}]_{k,\ell} \\
&= [\mathsf{attn_prob} \cdot \mathsf{res} \cdot \mathbf{V} \cdot \mathbf{O} \cdot \mathbf{U} + (\mathsf{input} \cdot \mathbf{E} + \mathbf{P}) \cdot \mathbf{U}]_{k,\ell} \\
&= (\mathsf{attn_prob} \cdot (\mathsf{input} \cdot \mathbf{E} + \mathbf{P}) \cdot \mathbf{VOU})_{k,\ell} + (\mathsf{input} \cdot \mathbf{EU})_{k,\ell} + (\mathbf{PU})_{k,\ell} \\
&= (\mathsf{attn_prob} \cdot (\mathsf{input} \cdot \mathbf{EVOU} + \mathbf{PVOU}))_{k,\ell} + \mathsf{EU}_{x_k,\ell} + \mathsf{PU}_{k,\ell} \\
&= \sum_{i=1}^{k} (\mathsf{attn_prob}_{k,i} \cdot (\mathsf{EVOU}_{x_i,\ell} + \mathsf{PVOU}_{i,\ell})) + \mathsf{EU}_{x_k,\ell} + \mathsf{PU}_{k,\ell} \\
&= \mathsf{attn_prob}_{k,m} \cdot (\mathsf{EVOU}_{x_m,\ell} + \mathsf{PVOU}_{m,\ell}) + \\
&\quad \sum_{i \in [k] \setminus \{m\}} (\mathsf{attn_prob}_{k,i} \cdot (\mathsf{EVOU}_{x_i,\ell} + \mathsf{PVOU}_{i,\ell})) + \mathsf{EU}_{x_k,\ell} + \mathsf{PU}_{k,\ell}
\end{aligned}$$

where we define the matrices $\mathsf{EU} = \mathbf{EU}$, $\mathsf{PU} = \mathbf{PU}$, $\mathsf{EVOU} = \mathbf{EVOU}$, and $\mathsf{PVOU} = \mathbf{PVOU}$. These matrix products can be precomputed in $O(n^3)$ time.

To approximate the distribution of $\mathsf{logits}_{k,\ell}$, we treat all four terms above as independent. The last term – $\mathsf{PU}_{k,\ell}$ – is a constant. The distribution of the third term – $\mathsf{EU}_{x_k,\ell}$ – can be exactly computed analytically in amortized constant time, ignoring binning overhead[4]. The first term can be approximated as the product of two independent distributions, the first of which we obtained in section 3.3, and the latter which can be calculated exactly[5].

The second term is treated as the $(k-1)$-wise convolution of the distribution of

$$\mathsf{attn\_prob}_{k,i \sim [k] \setminus m} \cdot (\mathsf{EVOU}_{x_i,\ell} + \mathsf{PVOU}_{i,\ell}).$$

To compute this distribution, we treat the two factors as independent. We have the distribution of the first factor from section 3.3. The distribution of the second factor can be computed exactly (up to binning error): the two terms are independent, and both are uniform over some matrix entries[6].

Once we have an estimate for the distribution of $\mathsf{logits}_{k,\ell}$ for every $\ell \in [n]$, we can estimate the distribution of the final probability the model places on x_m using the same approach as in section 3.3. The mean of this distribution is A_v.

4 Time Complexity

For a given choice of $x_m = v$, the time taken in section 3.3 is dominated by the $O(n \log k)$ pairwise convolutions. In section 3.4, we also require $O(n \log k + \log n)$ pairwise convolutions. This gives $O(n \log k \cdot b^2 \log b)$ for every choice of x_m, and so $O(n^2 \log k \cdot b^2 \log b)$ in total. Taking into account the $O(n^3)$ precomputation of matrix products, we get $\boxed{O(n^3 + n^2 \log k \cdot b^2 \log b)}$.

[4] The exact distribution of $\mathsf{EU}_{x_k,\ell}$ is $\mathsf{EU}_{x_m,\ell}$ with probability $1/k$, otherwise uniformly distributed among $\mathsf{EU}_{[x_m-1],\ell}$. The latter distribution can be built up cumulatively.

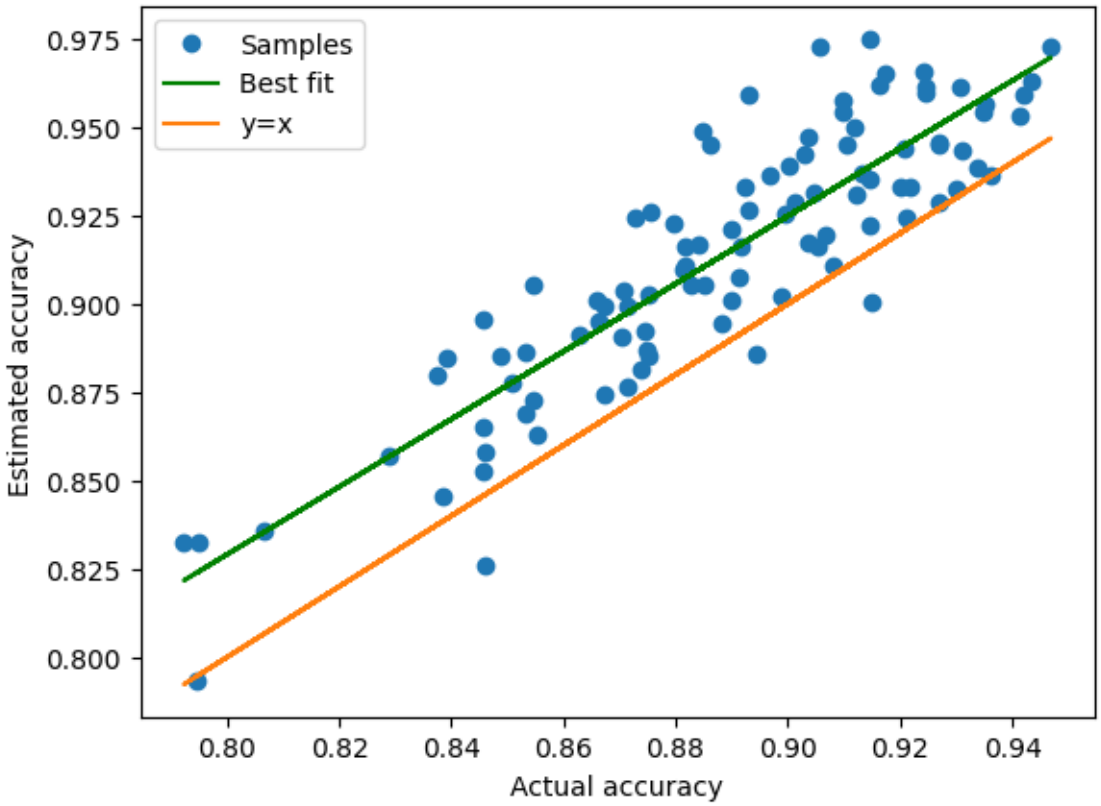
[5] Note $m \sim \mathrm{Unif}([k])$, so the distribution of $\mathsf{PVOU}_{m,\ell}$ is easy to calculate. $\mathsf{EVOU}_{x_m,\ell}$ is a constant.

[6] $\mathsf{EVOU}_{x_i,\ell}$ is distributed like $\mathsf{EVOU}_{[x_m-1],\ell}$, which can be built up cumulatively.

5 Empirical Results

Unless otherwised stated, I use $b = 10$ bins in my estimates. It takes about 3 seconds for the estimate to run on my computer.

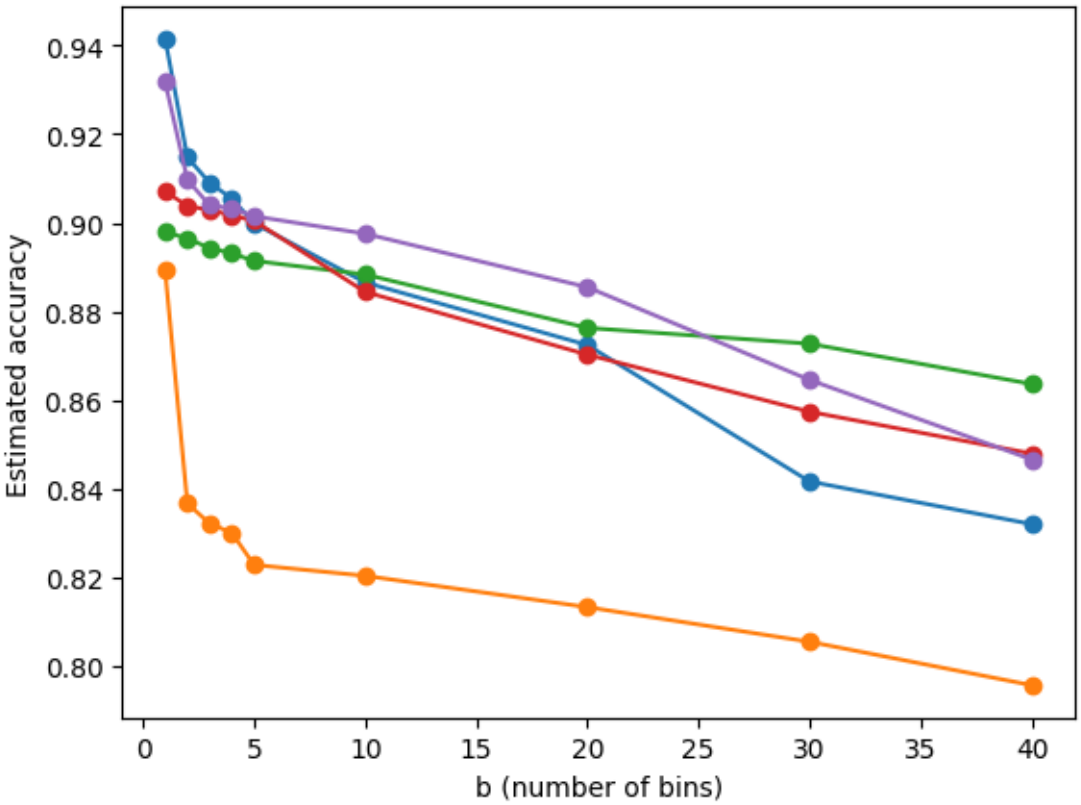
Let $\mathcal{D}$ be the distribution over models that you get by starting with an overtrained model M_0, then adding random Gaussian noise independently to every parameter ($\sigma = 0.1$). Here are 100 random samples from $\mathcal{D}$, plotting the true soft accuracy (empirically derived over 10^4 samples) against estimated soft accuracy:



The estimator tends to slightly overestimate the soft accuracy of the model: on average, the estimate is about 2.5 percentage points higher the ground truth. Overall, I'm somewhat satisfied with this estimator's performance – it's clearly doing something right. I'm not worried about a cherry-picked explanation or estimator because aren't many free-parameters to vary, and I didn't use the performance of the estimator to choose how to modify it.

On the overtrained model, which has a soft accuracy of 0.99990, the estimator outputs 0.99992.

Changing the number of bins also affects the estimate. More bins consistently results in a lower estimate. Here are the results of 5 different samples from $\mathcal{D}$ (each color represents different sample):



6 Open questions

Surprise accounting. Under our current understanding, we want the total suprise of $\mathbb{G}$ throughout processing the argument to be at most (log) the size of the search space of the coincidence (which in this case corresponds roughly to the number of paramaters of the model)[7]. I have not gone through all of the careful accounting to figure this out yet. I strongly suspect that this argument goes above our allotted suprise budget, and we'll have to figure out a way to use "naively predict the distribution of the result, then choose distribution and pay in the KL" steps to get it under budget, if it is possible at all.

Analog of Wick propagation. $\mathbb{G}(\mathbb{E}_{x\sim D}[M(x)]|\pi)$ is to $\mathbb{G}(M(x)|\pi)$ as covariance propagation is to Wick propagation as this estimator is to... what? It would be great if we had an analogous interpretation of what it means to apply this estimator to a specific input. This would allow for anomaly detection experiments.

[7]We'd also want to see how sensitive the model's behavior is to the exact setting of its parameters; if it's not sensitive then our surprise budget should decrease.