CSE6243: Advanced Machine Learning **Fall 2023**

Lecture 19: Differentiable Algorithm II: Top-K/Sorting Layer

Lecturer: Bo Dai *Scribes: Zachary Aristei*

Note: *LaTeX template courtesy of UC Berkeley EECS Department.*

Disclaimer: *These notes have not been subjected to the usual scrutiny reserved for formal publications. They may be distributed outside this class only with the permission of the Instructor.*

19.1 Recap and Overview

So far, this class has covered the following differentiable models in previous lectures:

1. Auto-Regressive Model as Differentiable Gibbs Sampler (Covered in Module 1)
2. Graph Neural Network (GNN) and Value Iteration Network (VIN) as Differentiable Dynamic Programming (Covered in Lecture 18)
3. Differentiable Searching **(Covered Today)**
4. Differentiable Convex Optimization (Dropped due to time constraints).

19.2 New Content

19.2.1 Motivating Example

The following optimization problem serves as a motivating example for why Differentiable Searching becomes useful in a Machine Learning context:

$$P(y_1, \cdots, y_t) = \Pi_{t=1}^{T}(P(y_t|y_{<t}) \tag{19.1}$$

This joint distribution is often optimized in the context of current state-of-the-art autoregressive and Large Language Models. Let $Y := [y_1, \cdots, y_t]^T$. In order to generate a sentence from a LLM model trained on this joint word distribution, it is necessary to maximize the joint distribution above using the following objective:

$$\max_{Y} \log P(Y) = \max_{Y} \sum_{t=1}^{T} \log P(y_t|y_{<t}) \tag{19.2}$$

For an simpler example with 2 words, this can be represented as:

$$\max_{y_1,y_2} \log P(y_1) + \log P(y_2|y_1)$$

In order to maximize this distribution, there's no efficient dynamic programming approach that's available that eliminates the need to compute a joint distribution conditioned on each word of the sequence. Hence we will find that there exists a need to differentiably approximate a search procedure to search for the top-k elements that maximize the joint distribution.

19.2.2 Search Heuristics

19.2.2.1 Greedy Search

In order to maximize this distribution, we will employ a greedy search for the `argmax` of each token position in the distribution.

Let $\mathcal{Y} := \{y \mid y \in \mathbb{N}\},\ |\mathcal{Y}| = n$ represent a finite set of discrete output tokens that are possible outputs to the model.

Algorithm 1 Greedy Search

```
$Y \leftarrow \varnothing$
$t \leftarrow 1$
**while** $t \leq T$ **do**
    $Y[t] \leftarrow \text{argmax}\,[\log P(Y[t] \mid Y)]$
**end while**
**return** $Y$
```

This algorithm has a critical issue. The inputs are generated entirely conditioned on the distribution of the earlier positioned tokens that get generated. This may not arrive at the maximum distribution because the earlier tokens are maximized from distributions that are not conditioned on the optimal tokens later in the sequence. Thus in order to resolve this issue, one must maximize conditioned on all the permutations of tokens possible, which is an intractable problem.

19.2.2.2 Beam Search

Proposal: Instead of greedily finding the top 1 choice at each step, search for the top k choices at each step, conditioned on all of the top k maximized joint distributions from the previous step.

When this algorithm is complete, $\bar{Y} = [Y^*_{T1}, \cdots, Y^*_{Tk}]^T$ represents sequences associated with the top k joint distributions of length T that have been found through the traversal of the sequence-space.

Algorithm 2 Beam Search

```
$\bar{Y} : |\bar{Y}| = k$
$\bar{Y} \leftarrow \varnothing$
$t \leftarrow 1$
**while** $t \leq T$ **do**
    $\mathbf{Y} \in \mathbb{R}^{k \times k}$
    **while** $i := 1, \ldots, k$ **do**
        $\mathbf{Y}[i] \leftarrow \text{argmax}_{top\ k}\,[\log P(Y[t] \mid \bar{Y}_i)]$
    **end while**
    $\bar{Y} \leftarrow \text{argmax}_{top\ k\ entries}\,\mathbf{Y}$
**end while**
**return** $\bar{Y}$
```

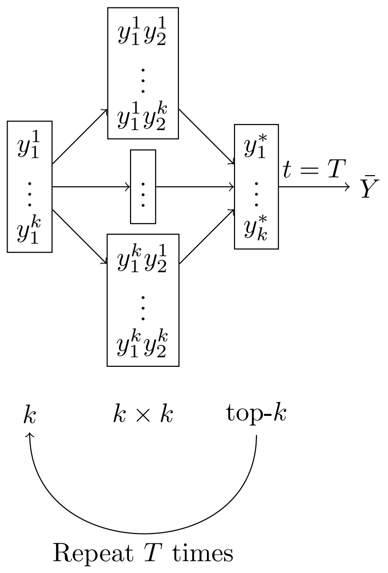

Figure 19.1: Beam Search Diagram

Beam search represents the de-facto state-of-the-art algorithm for parsing the permutations of tokens in search for the optimized combination for the joint probability. However, it is worth noting that this algorithm still only yields approximation, as searching through all $T \times |\mathcal{Y}|$ sequences for the Top-k is an NP-hard problem.

19.2.3 Differentiating Top-K

19.2.3.1 Defining the Optimization Problem

When it comes to training models that rely on finding the top-k entries, another problem presents itself in that the `argmax` operator is not differentiable. Thus an approximation must be derived in order to be able to have a top-k model in which $\nabla_\theta L(\bar{Y})$ can be evaluated.

This approximator will be optimize the following objective:

$$\max_{\Gamma \geq 0} < \Gamma, C > \tag{19.3}$$

$$\text{s.t. } \Gamma \mathbf{1}_m = \mu \tag{19.4}$$

$$\Gamma^T \mathbf{1}_n = \nu \tag{19.5}$$

Proposition 19.1 *The solution to the above optimization problem can be formulated as the Top-K when the* *following is true:*

$$\mu := \frac{\mathbf{1}_n}{n} \in \mathbb{R}^n \tag{19.6}$$

$$\nu := [\frac{k}{n}, \frac{n-k}{n}]^T \in \mathbb{R}^2 \tag{19.7}$$

$$C \in \mathbb{R}^{n\times 2} \tag{19.8}$$

$$\forall i,\ C_{i1} = x_i^2,\ C_{i2} = (x_i - 1)^2 \tag{19.9}$$

Proof: We start by expanding the objective function:

$$\begin{aligned} < C, \Gamma > &= \sum_{i=1}^{n} x_i^2 \Gamma_{i1} + (x_i - 1)^2 \Gamma_{i2} \\ &= \sum_{i=1}^{n} x_i^2 (\Gamma_{i1} + \Gamma_{i2}) + \Gamma_{i2} - 2x_i \Gamma_{i1} \\ &= \frac{1}{n} \sum_{i=1}^{n} x_i^2 + \frac{n-k}{n} - 2\sum_{i=1}^{n} x_i \Gamma_{i2} \end{aligned}$$

The constraints can also be modified:

$$\begin{aligned} (19.4), (19.6) &\Rightarrow \Gamma \mathbf{1}_m = \frac{\mathbf{1}_n}{n} \\ &\Rightarrow \forall i,\ \Gamma_{i1} + \Gamma_{i2} = \frac{1}{n} \\ &\Rightarrow \forall (i,j),\ \Gamma_{ij} \geq 0 \end{aligned}$$

$$\begin{aligned} (19.5), (19.7) &\Rightarrow \Gamma^T \mathbf{1}_n = [\frac{k}{n}, \frac{n-k}{n}]^T \\ &\Rightarrow \sum_{i=1}^{n} \Gamma_{i1} = \frac{k}{n} \\ &\Rightarrow \sum_{i=1}^{n} \Gamma_{i2} = \frac{n-k}{n} \end{aligned}$$

Thus maximizing (19.3) is functionally equivalent optimizing this objective:

$$\min_{\Gamma \geq 0} \sum_{i=1}^{n} x_i \Gamma_{i2} \tag{19.10}$$

$$\text{s.t.} \sum_{i=1}^{n} \Gamma_{i2} = \frac{n-k}{n} \tag{19.11}$$

$$\Gamma_{i2} \leq \frac{1}{n}\ \forall i \tag{19.12}$$

Based on this new objective, we know that for optimum Γ^*:

$$\Gamma_{i1}^* = \begin{cases} \frac{1}{n} & i \text{ in top } k \\ 0 & \text{otherwise} \end{cases} \tag{19.13}$$

■

19.2.3.2 Finding the Optimal Γ^*

With the optimization problem defined, the loss function $\mathcal{L}(\Gamma, C)$ can be defined based on (19.3). However, this problem is not differentiable as-is. So, a regularization term (such as the cross-entropy $H(\Gamma)$) needs to be added to ensure the function remains smooth across the domain:

$$\mathcal{L}_\lambda(\Gamma, C) = \underset{\Gamma \geq 0}{\operatorname{argmax}} < \Gamma, C > + \lambda H(\Gamma) \tag{19.14}$$

By evaluating $\frac{\partial \mathcal{L}_\lambda}{\partial \Gamma}$, we can derive the gradient descent update for finding Γ^*, with the help of KKT conditions:

$$\textit{Let } f(\Gamma, C) := \Gamma^* = \min_{\Gamma} \mathcal{L}(\Gamma, C) + \lambda \Omega(\Gamma \mathbf{1}_n - \mu) + \eta \Omega(\Gamma^T \mathbf{1}_n - \nu) \tag{19.15}$$

$$\Gamma_{i+1} = \Gamma_i - \varepsilon_i \nabla_\Gamma f(\Gamma, C) \tag{19.16}$$

19.2.4 Key Takeaway

The approach used here for Top-K approximation can be generalized to many other non-differentiable problems. More specifically, the procedure outlined here involves:

1. Formulate the problem as a constrained optimization problem with an objective function that can be optimized using KKT conditions.
2. Add a regularization parameter such as Entropy or Expectation to enable enough smoothing to make the objective differentiable across the entire domain.