Probabilistic loss approximation

Caio Corro

April 30, 2020

Let n be the length of input sentence, $\boldsymbol{s} \in \mathbb{R}^n$ and $\boldsymbol{e} \in \mathbb{R}^n$ be vectors of start and end scores, respectively, and $\boldsymbol{W} \in \mathbb{R}^{n\times n}$ be the matrix of bigram weights, i.e. $W_{i,j}$ is the score of word i being followed by word j. Let $\pi : [n] \to [n]$ denote a permutation, where $[n] = \{1 \dots n\}$. Let Π be the set of all permutations of length n.

The score of a permutation π is defined as:

$$f(\pi; \boldsymbol{s}, \boldsymbol{e}, \boldsymbol{W}) = s_{\pi(1)} + e_{\pi(n)} + \sum_{i=2}^{n} W_{\pi(i-1),\pi(i)}$$

The probability of a permutation π is defined as:

$$p_{\boldsymbol{s},\boldsymbol{e},\boldsymbol{W}}(\pi) = \frac{\exp(f(\pi; \boldsymbol{s}, \boldsymbol{e}, \boldsymbol{W}))}{\sum_{\pi' \in \Pi} \exp(f(\pi'; \boldsymbol{s}, \boldsymbol{e}, \boldsymbol{W}))} = \frac{\exp(f(\pi; \boldsymbol{s}, \boldsymbol{e}, \boldsymbol{W}))}{Z(\Pi; \boldsymbol{s}, \boldsymbol{e}, \boldsymbol{W})}$$

where $Z(\Pi; \boldsymbol{s}, \boldsymbol{e}, \boldsymbol{W})$ is the parition function. We will omit $\boldsymbol{s}$, $\boldsymbol{e}$ and $\boldsymbol{W}$ as it is clear from context.

The negative log-likelihood loss given the gold permutation $\hat{\pi}$ is defined as follows:

$$\mathcal{L}(\hat{\pi}) = -\log p(\hat{\pi}) = -\log \frac{\exp(f(\hat{\pi}))}{Z(\Pi)} = -f(\hat{\pi}) + \log Z(\Pi)$$

In general, computing $\log Z(\Pi)$ and its gradient is intractable (i.e. summing over all feasible solutions of the TSP). If Π has a special structure (i.e. a subset of all permutation), it can be tractable, i.e. with permutation computed via our dynamic program, we could compute $\log Z(\Pi)$ and its gradient with an equivalent of the forward-backward/inside-outside algorithm. However, even in this case this case be slow (as we need to compute it many times during training) and and computationally instable (e.g. computation can easily overflow).

Instead, we can note that the gradient of the partition function is:

$$\begin{aligned}
\nabla \log Z(\Pi) &= \frac{1}{Z(\Pi)} \nabla Z(\Pi) \\
&= \frac{1}{Z(\Pi)} \nabla \sum_{\pi' \in \Pi} \exp(f(\pi')) \\
&= \frac{1}{Z(\Pi)} \sum_{\pi' \in \Pi} \nabla \exp(f(\pi')) \\
&= \frac{1}{Z(\Pi)} \sum_{\pi' \in \Pi} \exp(f(\pi')) \nabla f(\pi') \\
&= \sum_{\pi' \in \Pi} \frac{\exp(f(\pi'))}{Z(\Pi)} \nabla f(\pi') \\
&= \sum_{\pi' \in \Pi} p(\pi') \nabla f(\pi') \\
&= \mathbb{E}_{\pi \sim p(\pi)} [\nabla f(\pi)] \\
&\simeq \frac{1}{k} \sum_{i=1}^{k} f(\pi^{(i)})
\end{aligned}$$

where we approximate the expectation via Monte-Carlo method with k samples $\pi^{(i)} \sim p(\pi)$. Note that we cannot sample from $p(\pi)$ efficiently, so we will just sample from another distribution for now before doing smarter (see the github issue for the different methods).

Note that here, we have an approximation of the gradient, but no approximation of the forward pass. To implement this in Pytorch, you first compute $\boldsymbol{s}$, $\boldsymbol{e}$ and $\boldsymbol{W}$. Then, you build a matrix $\tilde{\boldsymbol{s}}$ initialized at zero and:

- set $\tilde{\boldsymbol{s}}_{\hat{\pi}(1)} = 1$.
- for each sample $\pi^{(i)}$ you increment $\tilde{\boldsymbol{s}}_{\pi^{(i)}(1)}$ by $\frac{1}{k}$

and similarly for $\boldsymbol{e}$ and $\boldsymbol{W}$. Then you sum every cell of $\boldsymbol{s} \times \tilde{\boldsymbol{s}}$, $\boldsymbol{e} \times \tilde{\boldsymbol{e}}$ and $\boldsymbol{W} \times \tilde{\boldsymbol{W}}$ where $\times$ is the element wise multiplication, and call backward on this results. You can check that it will give the correct gradient estimation.