Gumbel-Max, Gumbel-Softmax and Straight-Through

Beier Zhu

1 Motivation

Deep networks with discrete latent variables are hard to train because back-propagation cannot pass through non-differentiable layers. This note introduces the Gumbel-Max, Gumbel-Softmax estimators and their straight-through variant, which use the reparameterization trick to provide differentiable gradients for **categorical variables**.

Consider a random variable y whose distribution depends on parameter θ and loss function $f(y)$. The objective is to minimize the expected loss $\mathcal{L}(\theta) = \mathbb{E}_{y\sim\mathbb{P}_\theta}[f(y)]$ via gradient descent, which requires to estimate $\nabla_\theta \mathbb{E}_{y\sim\mathbb{P}_\theta}[f(y)]$. For distributions that are reparameterizable, we can compute the sample y as a deterministic function of the parameter θ and an independent random variable z, so that $y = g(\theta, z)$. The path-wise gradients from f to θ can be computed without encountering any stochastic nodes:

$$\frac{\partial}{\partial\theta}\mathbb{E}_{y\sim\mathbb{P}_\theta}[f(y)] = \frac{\partial}{\partial\theta}\mathbb{E}_z[f(g(\theta, z))] = \mathbb{E}_z\left[\frac{\partial f}{\partial g}\frac{\partial g}{\partial\theta}\right] \tag{1}$$

E.g., the Gaussian distribution $y \sim \mathcal{N}(\mu(\theta), \sigma(\theta))$ can be written as $y = \mu(\theta) + \epsilon\sigma(\theta),$ where $\epsilon \sim \mathcal{N}(0, 1)$, making it easy to compute $\frac{\partial y}{\partial\mu}\frac{\partial\mu(\theta)}{\partial\theta} = \nabla_\theta\mu(\theta)$ and $\frac{\partial y}{\partial\sigma}\frac{\partial\sigma(\theta)}{\partial\theta} = \epsilon\cdot\nabla_\theta\sigma(\theta)$.

2 Preliminaries

Definition 1 (Gumbel Distribution)**.** *Given mode μ and scale $\beta > 0$ and defining $z = \frac{x-\mu}{\beta}$, the CDF and PDF of* $\mathrm{Gumbel}(\mu, \beta)$ *are*

$$F(x; \mu, \beta) = \exp(-\exp(-z)), \quad f(x; \mu, \beta) = \frac{1}{\beta}\exp(-z - \exp(-z)) \tag{2}$$

The standard Gumbel distribution $\mathrm{Gumbel}(0, 1)$ *is*

$$F(x) = \exp(-\exp(-x)), \quad f(x) = \exp(-x - \exp(-x)) \tag{3}$$

3 Gumbel-Max Sampling

Goal: sample from a categorical distribution parameterized by:

$$\mathbb{P}(k) = \frac{1}{Z}\exp(x_k), \quad \text{where } Z = \sum_{k=1}^{K}\exp(x_k) \tag{4}$$

The Gumbel-max trick samples from $\mathbb{P}(k)$ by adding Gumbel noise to each x_k and then taking the $\arg\max$:

$$y = \arg\max_{k\in[K]} x_k + z_k, \text{ where } z_1, ..., z_K \sim \mathrm{Gumbel}(0, 1)^K. \tag{5}$$

Proof. Let $r_k = x_k + z_k$, it is straightforward that $r_k \sim \mathrm{Gumbel}(x_k, 1)$. Suppose that the k-th Gumbel variable r_k exceeds others. Then the probability of such event is

$$\mathbb{P}(k \text{ is largest}|r_k, \{x_{k'}\}_{k'=1}^K) = \prod_{k'\neq k} F(r_k; x'_k, 1) = \prod_{k'\neq k}\exp(-\exp(-r_k + x_{k'})) \tag{6}$$

Integrating over the condition z_k yields the marginal distribution

$$
\begin{aligned}
&\mathbb{P}(k \text{ is largest}|\{x_{k'}\}_{k'=1}^K) && (7)\\
&= \int \mathbb{P}(k \text{ is largest}|r_k, \{x_{k'}\}_{k'=1}^K) f(r_k; x_r, 1) \mathrm{d}r_k && (8)\\
&= \int \prod_{k' \neq k} \exp\{-\exp(-r_k + x_{k'})\} \exp\{-r_k + x_k - \exp(-r_k + x_k)\} \mathrm{d}r_k && (9)\\
&= \int \exp\{-\sum_{k' \neq k} \exp(-r_k + x_{k'}) - r_k + x_k - \exp(-r_k + x_k)\} \mathrm{d}r_k && (10)\\
&= \exp(x_k) \int \exp\{-r_k - \exp(-r_k) \sum_{k'} \exp(x_{k'})\} \mathrm{d}r_k && (11)\\
&= \frac{1}{R} \exp(x_k) && (12)
\end{aligned}
$$

Here, we denote $\frac{1}{R} := \int \exp\{-r_k - \exp(-r_k) \sum_{k'} \exp(x_{k'})\} \mathrm{d}r_k$ which is constant for all k. By definition, we have

$$
\sum_{k=1}^K \mathbb{P}(k \text{ is largest}|\{x_{k'}\}_{k'=1}^K) = \frac{\sum_{k=1}^K \exp(x_k)}{R} = 1. \tag{13}
$$

Therefore, we have

$$
\mathbb{P}(k \text{ is largest}|\{x_{k'}\}_{k'=1}^K) = \frac{\exp(x_k)}{\sum_{k'=1}^K \exp(x_{k'})}, \tag{14}
$$

which is exactly the softmax probability. $\square$

4 Gumbel-Softmax Sampling

Because $\arg\max$ is non-differentiable, it cannot be used directly to train neural networks. Gumbel-Softmax use the softmax function as a continuous approximation to $\arg\max$ to generate K-dimensional simplex $\mathbf{y} \in \Delta^{K-1}$

$$
\mathbf{y}_k = \frac{\exp\{(x_k + z_k)/\tau\}}{\sum_{k'=1}^K \exp\{(x_{k'} + z_{k'})/\tau\}}. \tag{15}
$$

The density of the Gumbel-Softmax distribution becomes identical to the categorical distribution $\mathbb{P}(k)$ when $\tau \to 0$. While Gumbel-Softmax samples are differentiable, they are not identical to samples from the corresponding categorical distribution for non-zero temperature. For learning, there is a tradeoff between small temperatures, where samples are close to one-hot but the variance of the gradients is large, and large temperatures, where samples are smooth but the variance of the gradients is small. **In practice, we start at a high temperature and anneal to a small but non-zero temperature.**

5 Straight-Through

Continuous relaxations of one-hot vectors are suitable for problems such as learning hidden representations and sequence modeling. For scenarios in which we are constrained to sampling discrete values (e.g. from a discrete action space for reinforcement learning, or quantized compression), we discretize y using $\arg\max$ but use our continuous approximation in the backward pass by approximating

$$
y = \mathsf{sg}[\arg\max_{k \in [K]}[x_k + z_k] - \mathbf{y}] + \mathbf{y}, \tag{16}
$$

where $\mathsf{sg}[]$ is the stop gradient operator (which can be implemented as `.detach()` in Pytorch). In the forward pass, the output is $y = \arg\max_k[x_k + z_k]$. In the backward pass, the gradient is the smoothed $\nabla \mathbf{y}$.

6 Application: Neural Networks Sparsification

Let $\mathcal{D} = \{(\mathbf{x}_1, \mathbf{y}_1), \cdots, (\mathbf{x}_N, \mathbf{y}_N)\}$ be a dataset consists of N i.i.d. samples, $\mathbf{w} \in \mathbb{R}^n$ be the weights of a neural network. We denote $\mathbf{m} \in \{0,1\}^n$ as the mask of the weights: $m_i = 0$ means the weight w_i is pruned and otherwise w_i is kept. The problem of training sparse neural networks can be formulated as

$$\min_{\mathbf{w},\mathbf{m}} \mathcal{L}(\mathbf{w}, \mathbf{m}) = \frac{1}{N} \sum_{i=1}^{N} \ell(h(\mathbf{x}_i; \mathbf{w} \circ \mathbf{m}), \mathbf{y}_i)$$
$$\textit{s.t. } \|\mathbf{m}\|_0 \leq K \text{ and } \mathbf{m} \in \{0,1\}^n, \tag{17}$$

where $h(\cdot; \mathbf{w} \circ \mathbf{m})$ is the pruned network with $\circ$ being the element-wise product, and $\ell(\cdot, \cdot)$ is the loss function, *e.g.*, squared loss for regression and cross-entropy loss for classification, and K is the is the model size we want to reduce the network to. However, since the objective is discrete with respect to the mask $\mathbf{m}$, thus such problem is hard to solve. Instead, we view each component of mask $\mathbf{m}$ as a binary random variable and reparameterize Problem. (17) with respect to the distribution of this random variable. Specifically, we view m_i as a Bernoulli random variable with probability s_i to be 1 and $1 - s_i$ to be 0, that is $m_i \sim \mathrm{Bern}(s_i)$, where $s_i \in [0, 1]$. Assuming the variables m_i are independent, then the distribution of $\mathbf{m}$ and the expectation of its L_0 norm are

$$\mathbb{P}(\mathbf{m} \mid \mathbf{s}) = \prod_{i=1}^{n} s_i^{m_i} (1 - s_i)^{(1-m_i)} \tag{18}$$

$$\mathbb{E}_{\mathbf{m}}[\|\mathbf{m}\|_0] = \sum_{i=1}^{n} s_i = \mathbf{1}^\top \mathbf{s}. \tag{19}$$

Therefore, problem (17) can be relaxed into the following formulation

$$\min_{\mathbf{w},\mathbf{m}} \mathbb{E}_{\mathbf{m}}[\mathcal{L}(\mathbf{w}, \mathbf{m})] \tag{20}$$

$$\textit{s.t. } \mathbf{1}^\top \mathbf{s} \leq K \text{ and } \mathbf{s} \in [0,1]^n \tag{21}$$

Loss computation. Eq. (20) can be viewed as a special case of standard Gumbel-Softmax reparameterization when the categorical distribution has only two classes–that is, when the categorical distribution degenerates into a Bernoulli.

In Gumbel-Softmax with two classes, for a categorical distribution with logits $\{x_0, x_1\}$ and Gumbel noise $z_0, z_1 \sim \mathrm{Gumbel}(0,1)^2$, a differentiable sample is obtain as in Eq. (15)

$$m = y_1 = \frac{\exp\{(x_1 + z_1)/\tau\}}{\exp\{(x_0 + z_0)/\tau\} + \exp\{(x_1 + z_1)/\tau\}} \tag{22}$$

$$= \frac{1}{1 + \exp\{-[(x_1 - x_0) + (z_1 - z_0)]/\tau\}} \tag{23}$$

$$= \sigma\left(\frac{(x_1 - x_0) + (z_1 - z_0)}{\tau}\right) \tag{24}$$

By setting $x_1 = \log s$ and $x_0 = \log(1 - s)$, we obtain

$$m = \sigma\left(\frac{\log \frac{s}{1-s} + z_1 - z_0}{\tau}\right), \ z_0, z_1 \sim \mathrm{Gumbel}(0,1)^2, \tag{25}$$

where $\sigma(x) = \frac{1}{1+\exp(-x)}$. Note that the difference between two Gumbel noises follows Logistic distribution, *i.e.*, $z_1 - z_0 \sim \mathrm{Logistic}(0, 1)$. Let $\epsilon \sim \mathrm{Logistic}(0, 1)$, Eq. (25) can be written as

$$m = \sigma\left(\frac{\log \frac{s}{1-s} + \epsilon}{\tau}\right), \ \epsilon \sim \mathrm{Logistic}(0,1) \tag{26}$$

Applying to all components of $\mathbf{m}$, Eq. (20) becomes:

$$\min_{\mathbf{w},\mathbf{s}} \mathbb{E}_{\boldsymbol{\epsilon}}\left[\mathcal{L}\left(\mathbf{w}, \sigma\left(\frac{\log \frac{\mathbf{s}}{1-\mathbf{s}} + \boldsymbol{\epsilon}}{\tau}\right)\right)\right] \tag{27}$$

Algorithm 1 Neural Networks Sparsification

repeat
    Sample mini batch of data $\mathcal{B}$
    Sample I noises from Logistic distribution $\boldsymbol{\epsilon} \sim \text{Logistic}(0,1)^I$
    Gradient descent $\mathbf{w}, \mathbf{z} \leftarrow \mathbf{w}, \mathbf{s} - \eta \mathbf{g}(\mathcal{B})$, where η is the learning rate
    Projection to $\mathcal{C}$: $\mathbf{s} \leftarrow \mathsf{proj}_{\mathcal{C}}(\mathbf{z})$
until converge

Let $\{\boldsymbol{\epsilon}^{(i)}\}_{i=1}^{I}$ denote the I sampled Logistic noises, and $\mathcal{B}$ denote the sampled batch data $\{(\mathbf{x}_1, \mathbf{y}_1), ..., (\mathbf{x}_B, \mathbf{x}_B)\}$, using Monte-Carlo estimation of the expected gradient, we have

$$\mathbf{g}(\mathcal{B}) = \frac{1}{I} \sum_{i=1}^{I} \nabla_{\mathbf{w},\mathbf{s}} \mathcal{L}_{\mathcal{B}} \left(\mathbf{w}, \sigma \left(\frac{\log \frac{\mathbf{s}}{1-\mathbf{s}} + \boldsymbol{\epsilon}^{(i)}}{\tau} \right) \right) \tag{28}$$

Projected gradient descent. We denote the feasible region in Eq. (21) as

$$\mathcal{C} = \{\mathbf{s} | \mathbf{1}^\top \mathbf{s} \leq K \text{ and } \mathbf{s} \in [0,1]^n\}. \tag{29}$$

We aim to project a vector $\mathbf{z} \in \mathbb{R}^n$ onto the convex set $\mathcal{C}$. This corresponds to solving

$$\min_{\mathbf{s}} \frac{1}{2} \|\mathbf{s} - \mathbf{z}\|_2^2 \ \textit{s.t.}\ 0 \leq s_i \leq 1, \sum_i s_i \leq K. \tag{30}$$

We introduce the Lagrange multipliers: $\lambda \geq 0$, $\alpha_i \geq 0$ and $\beta_i \geq 0$. The Lagrangian is

$$\mathcal{L}(\mathbf{s}, \lambda, \boldsymbol{\alpha}, \boldsymbol{\beta}) = \frac{1}{2} \|\mathbf{s} - \mathbf{z}\|_2^2 + \lambda \left(\sum_i s_i - K \right) - \sum_i \alpha_i s_i + \sum_i \beta_i (s_i - 1) \tag{31}$$

The KKT conditions are:

$$s_i = z_i - \lambda + \alpha_i - \beta_i \tag{32}$$

$$\alpha_i s_i = 0, \beta_i (s_i - 1) = 0, \lambda (\sum_i s_i - K) = 0 \tag{33}$$

- if $s_i \in (0,1)$, then $\alpha_i = \beta_i = 0 \Rightarrow s_i = z_i - \lambda$.
- if $s_i = 0$, then $\alpha_i \geq 0, \beta_i = 0 \Rightarrow z_i - \lambda = s_i - \alpha_i + \beta_i \leq 0 \Rightarrow s_i = \max(0, z_i - \lambda)$.
- if $s_i = 1$, then $\alpha_i = 0, \beta_i \geq 0 \Rightarrow z_i - \lambda = s_i - \alpha_i + \beta_i = 1 + \beta_i \geq 1 \Rightarrow s_i = \min(1, z_i - \lambda)$

Combining all three conditions, we have $s_i(\lambda) = \mathsf{clip}(z_i - \lambda, 0, 1)$. Finally, from the complementary slackness of λ:

- **If the constraint is inactive** (*i.e.*, $\lambda = 0$): compute $s_i = \mathsf{clip}(z_i, 0, 1)$ and check whether $\sum_i s_i \leq K$. If satisfied, return $s_i = \mathsf{clip}(z_i, 0, 1)$.
- **Otherwise** (constraint active): since $\mathsf{clip}(z_i - \lambda, 0, 1)$ is a non-increasing function of λ, apply a bisection search to find λ^* such that $\sum_i \mathsf{clip}(z_i - \lambda^*, 0, 1) = K$, and return $s_i = \mathsf{clip}(z_i - \lambda^*, 0, 1)$.