Differientiable Sampling and Argmax

WIP, last updated: 2019.12.6

Introduction

Softmax is a commonly used function for turning an **unnormalized log probability** into a normalized probability (or **categorical distribution**).

$$\pi = \mathrm{softmax}(\mathbf{o}) = \frac{e^{\mathbf{o}}}{\sum_j e^{o_j}},$$
$$o_j \in (-\infty, +\infty)$$

Say $\mathbf{o}$ is the output of a neural network before softmax, we call $\mathbf{o}$ the **unnormalized log probability.**

After softmax, we usually **sample** from this categorical distribution, or taking an **argmax** function to select the index. However, one can notice that neither the **sampling** nor the **argmax** is **differientiable**.

Researchers have proposed several works to make this possible. I am going to discuss them here.

Sampling

I will introduce Gumbel Softmax [1611.01144], which have made the **sampling** procedure differentiable.

Gumbel Max

First, we need to introduce **Gumbel Max**. In short, Gumbel Max is a trick to use gumbel distribution to sample a categorical distribution.

Say we want to sample from a categorical distribution π.
The usual way of doing this is using π to separate $[0,1]$ into intervals, sampling from a uniform distribution $\mathrm{U} \sim [0,1]$, and see where it locates.

The Gumbel Max trick provides an alternative way of doing this. It use **Reparameterization Trick** to avoid the stochastic node during backpropagation.

$$y = \arg\max_i(o_i + g_i)$$

where $g_i \sim \mathrm{Gumbel}(0,1)$, which can be sampled by $-\log(-\log(\mathrm{Uniform}[0,1]))$.
We can prove that y is distributed according to π.

Prove that

$y = \arg\max_i(o_i + g_i)$, where $g_i \sim \mathrm{Gumbel}(0,1)$
which can be sampled by $-\log(-\log(\mathrm{Uniform}[0,1]))$
is distributed with $\pi = \mathrm{softmax}(o_i) = \frac{e^{o_i}}{\sum je^{o_j}}$

Prerequisites

Gumbel Distribution (param by location ****μ, and scale $\beta > 0$) (wikipedia)
CDF: $F(x;\mu,\beta) = e^{-e^{(x-\mu)/\beta}}$
PDF: $f(x;\mu,\beta) = \frac{1}{\beta}e^{-(z+e^{-z})}, z = \frac{x-\mu}{\beta}$

Mean: $\mathrm{E}(X)=\mu+\gamma\beta, \gamma\approx 0.5772$is the Euler–Mascheroni constant.
Quantile Function: $Q(p)=\mu-\beta\log(-\log(p))$(Quantile Function is used to sample random variables from a distribution given CDF, it is also called inverse CDF)
Proof

We actually want to prove that $\mathrm{Gumbel}(\mu=o_i,\beta=1)$ is distributed with $\pi_i=\frac{e^{o_i}}{\sum_j e^{o_j}}$.

We can find that $\mathrm{Gumbel}(\mu=o_i,\beta=1)$ has the following PDF and CDF

$$f(x;\mu,1)=e^{-(x-\mu)-e^{-(x-\mu)}} \quad (1)$$
$$F(x;\mu,1)=e^{-e^{-(x-\mu)}} \quad (2)$$

.Then, the probability that all other $\pi_{j\neq i}$ are less than π_i is:

$$\Pr(\pi_i \text{ is the largest}|\pi_i,\{o_j\})=\prod_{j\neq i}e^{-e^{-(\pi_i-o_j)}}$$

We know the marginal distribution over π_i and we are able to integrate it out to find the overall probability: ($p(x)=\int_y p(x,y)dy=\int_y p(x|y)p(y)dy$)

$$\Pr(i \text{ is largest}|\{o_j\})=\int e^{-(\pi_i-o_i)-e^{-(\pi_i-o_i)}}\times\prod_{j\neq i}e^{-e^{-(\pi_i-o_j)}}\,\mathrm{d}\pi_i \quad (3)$$
$$=\int e^{-\pi_i+o_i-e^{-\pi_i}\sum_j e^{o_j}}\,\mathrm{d}\pi_i \quad (4)$$
$$=\frac{e^{o_i}}{\sum_j e^{o_j}} \quad (5)$$

which is exactly a softmax probablity. QED.

Reference: https://lips.cs.princeton.edu/the-gumbel-max-trick-for-discrete-distributions/****

Gumbel Softmax

Notice that there is still an argmax in Gumbel Max, which still makes it indifferentiable. Therefore, we use a softmax function to approximate this argmax procedure.

$$\mathbf{y} = \frac{e^{(o_i+g_i)/\tau}}{\sum_j e^{(o_j+g_j)/\tau}}$$

where $\tau \in (0, \infty)$ is a temparature hyperparameter.

We note that the output of Gumbel Softmax function here is a vector which sum to 1, which somewhat looks like a one-hot vector (but it's not).
So by far, this does not actually replace the argmax function.

To actually get a pure one-hot vector, we need to use a **Straight-Through (ST) Gumbel Trick**.
Let's directly see an implementation of Gumbel Softmax in PyTorch
(We use the hard mode, soft mode does not get a pure one-hot vector).

```python
def gumbel_softmax(logits, tau=1, hard=False, eps=1e-10, dim=-1):
    # type: (Tensor, float, bool, float, int) -> Tensor
    r"""
    Samples from the Gumbel-Softmax distribution (`Link 1`_  `Link 2`_) a

    Args:
      logits: `[..., num_features]` unnormalized log probabilities
      tau: non-negative scalar temperature
      hard: if ``True``, the returned samples will be discretized as one-
            but will be differentiated as if it is the soft sample in aut
      dim (int): A dimension along which softmax will be computed. Defaul

    Returns:
      Sampled tensor of same shape as `logits` from the Gumbel-Softmax di
      If ``hard=True``, the returned samples will be one-hot, otherwise t
      be probability distributions that sum to 1 across `dim`.

    .. note::
      This function is here for legacy reasons, may be removed from nn.Fu

    .. note::
      The main trick for `hard` is to do  `y_hard - y_soft.detach() + y_s

      It achieves two things:
      - makes the output value exactly one-hot
      (since we add then subtract y_soft value)
      - makes the gradient equal to y_soft gradient
      (since we strip all other gradients)

    Examples::
        >>> logits = torch.randn(20, 32)
        >>> # Sample soft categorical using reparametrization trick:
        >>> F.gumbel_softmax(logits, tau=1, hard=False)
        >>> # Sample hard categorical using "Straight-through" trick:
        >>> F.gumbel_softmax(logits, tau=1, hard=True)
```

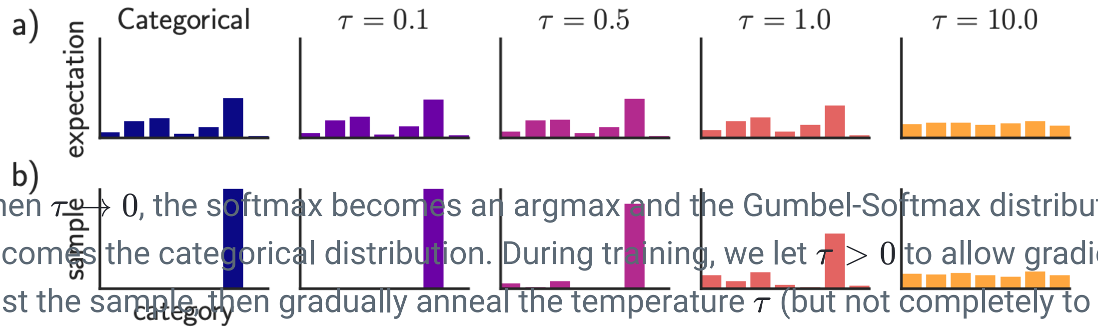
```python
    .. _Link 1:
        https://arxiv.org/abs/1611.00712
    .. _Link 2:
        https://arxiv.org/abs/1611.01144
    """
    if eps != 1e-10:
        warnings.warn("`eps` parameter is deprecated and has no effect.")

    gumbels = -torch.empty_like(logits).exponential_().log()  # ~Gumbel(0
    gumbels = (logits + gumbels) / tau  # ~Gumbel(logits,tau)
    y_so

    if hard:
        # Straight through.
        index = y_soft.max(dim, keepdim=True)[1]
        y_hard = torch.zeros_like(logits).scatter_(dim, index, 1.0)
        ret = y_hard - y_soft.detach() + y_soft
    else:
        # Reparametrization trick.
        ret = y_soft
    return ret
```

When fowarding, the code use an argmax to get an actual one-hot vector.

And it uses `ret = y_hard - y_soft.detach() + y_soft`, `y_hard` has no grad, and by minusing `y_soft.detach()` and adding `y_soft`, it achieves a grad from `y_soft` without modifying the forwarding value.

So eventually, we are able to get a pure one-hot vector in forward pass, and a grad when back propagating, which **makes the sampling procedure differentiable**.

Finally, let's look at how τ affects the sampling procedure. The below image shows the sampling distribution (which is also called the Concrete Distribution [1611.00712]) and one random sample instance when using different hyperparameter τ.



> when $\tau \to 0$, the softmax becomes an argmax and the Gumbel-Softmax distribution becomes the categorical distribution. During training, we let $\tau > 0$ to allow gradients past the sample, then gradually anneal the temperature τ (but not completely to 0, as the gradients would blow up).
>
> from Eric Jang, https://blog.evjang.com/2016/11/tutorial-categorical-variational.html

Figure 1: The Gumbel-Softmax distribution interpolates between discrete one-hot-encoded categorical distributions and continuous categorical densities. (a) For low temperatures ($\tau = 0.1, \tau = 0.5$), the expected value of a Gumbel-Softmax random variable approaches the expected value of a categorical random variable with the same logits. As the temperature increases ($\tau = 1.0, \tau = 10.0$), the expected value converges to a uniform distribution over the categories. (b) Samples from Gumbel-Softmax distributions are identical to samples from a categorical distribution as $\tau \to 0$. At higher temperatures, Gumbel-Softmax samples are no longer one-hot, and become uniform as $\tau \to \infty$.

image from https://arxiv.org/abs/1611.01144

Argmax

How to make argmax differentiable?

> ⓘ Intuitively, the **Straight-Through Trick** is also applicable for softmax+argmax

Some have introduced the soft-argmax function. It doesn't actually makes it differentiable, but use a continuous function to approximate the softmax+argmax procedure.

$$\pi = \text{soft-argmax}(\mathbf{o}) = \frac{e^{\beta \mathbf{o}}}{\sum_j e^{\beta o_j}}$$

where β can be a large value to make π very much "look like" a one-hot vector.

Discussion

- Goal
  - **softmax + argmax** is used for classification, we only want the index with the highest probability.
  - **gumbel softmax + argmax** is used for sampling, we may want to sample an index not with the highest probability.
- Deterministic
  - **softmax + argmax** is deterministic. Get the index with the highest probablity.
  - **gumbel softmax + argmax** is stochastic. We need to sample from a gumbel distribution in the beginning.
- Output vector
  - **softmax** and **gumbel softmax** aboth output a vector sum to 1.
  - **softmax** outputs a *normalized probability distribution*.
  - **gumbel softmax** outputs a *sample* somewhat more similar to a one-hot vector. (can be controlled by τ)
- **Straight-Through Trick** can actually be applied to both **softmax + argmax** and **gumbel softmax + argmax**, which can make both of them differentiable. (?)

Reference

- Gumbel Softmax [1611.01144]
- Concrete Distribution (Gumbel Softmax Distribution) [1611.00712]
- Eric Jang official blog: https://blog.evjang.com/2016/11/tutorial-categorical-variational.html
- PyTorch Implementation of Gumbel Softmax: https://pytorch.org/docs/stable/nn.functional.html#torch.nn.functional.gumbel_softmax

- https://timvieira.github.io/blog/post/2014/07/31/gumbel-max-trick/
- https://lips.cs.princeton.edu/the-gumbel-max-trick-for-discrete-distributions/

Machine Learning - Previous
Meta Learning

Next - Notes
GAN theory