

Graduate Descent

- [Atom](#)
- [About](#)
- [Archive](#)

Exp-normalize trick

Feb 11, 2014

This trick is the very close cousin of the infamous log-sum-exp trick ([scipy.misc.logsumexp](#)).

Supposed you'd like to evaluate a probability distribution $\boldsymbol{\pi}$ parametrized by a vector $\mathbf{x} \in \mathbb{R}^n$ as follows:

$$\pi_i = \frac{\exp(x_i)}{\sum_{j=1}^n \exp(x_j)}$$

The exp-normalize trick leverages the following identity to avoid numerical overflow. For any $b \in \mathbb{R}$,

$$\pi_i = \frac{\exp(x_i - b) \exp(b)}{\sum_{j=1}^n \exp(x_j - b) \exp(b)} = \frac{\exp(x_i - b)}{\sum_{j=1}^n \exp(x_j - b)}$$

In other words, the $\boldsymbol{\pi}$ is shift-invariant. A reasonable choice is $b = \max_{i=1}^n x_i$. With this choice, overflow due to \exp is impossible—the largest number exponentiated after shifting is 0.

The naive implementation is terrible when there are large numbers!

```
>>> x = np.array([1, -10, 1000])
>>> np.exp(x) / np.exp(x).sum()
RuntimeWarning: overflow encountered in exp
RuntimeWarning: invalid value encountered in true_divide
Out[4]: array([ 0.,  0., nan])
```

The exp-normalize trick avoid this common problem.

```
def exp_normalize(x):
    b = x.max()
    y = np.exp(x - b)
    return y / y.sum()
```

```
>>> exp_normalize(x)
array([0., 0., 1.])
```

Log-sum-exp for computing the log-distribution

$$\log \pi_i = x_i - \text{logsumexp}(\mathbf{x})$$

where

$$\text{logsumexp}(\mathbf{x}) = b + \log \sum_{j=1}^n \exp(x_j - b)$$

Typically with the same choice for b as above.

Exp-normalize v. log-sum-exp

Exp-normalize is the gradient of log-sum-exp. So you probably need to know both tricks!

If what you want to remain in log-space, that is, compute $\log(\boldsymbol{\pi})$, you should use `logsumexp`. However, if $\boldsymbol{\pi}$ is your goal, then exp-normalize trick is for you! Since it avoids additional calls to `exp`, which would be required if using log-sum-exp and more importantly exp-normalize is more numerically stable!

Numerically stable sigmoid function

The sigmoid function can be computed with the exp-normalize trick in order to avoid numerical overflow. In the case of $\text{sigmoid}(x)$, we have a distribution with unnormalized log probabilities $[x, 0]$, where we are only interested in the probability of the first event. From the exp-normalize identity, we know that the distributions $[x, 0]$ and $[0, -x]$ are equivalent (to see why, plug in $b = \max(0, x)$). This is why sigmoid is often expressed in one of two equivalent ways:

$$\text{sigmoid}(x) = 1/(1 + \exp(-x)) = \exp(x)/(\exp(x) + 1)$$

Interestingly, each version covers an extreme case: $x = \infty$ and $x = -\infty$, respectively. Below is some python code which implements the trick:

```
def sigmoid(x):
    "Numerically stable sigmoid function."
    if x >= 0:
        z = exp(-x)
        return 1 / (1 + z)
    else:
        # if x is less than zero then z will be small, denom can't be
        # zero because it's 1+z.
        z = exp(x)
        return z / (1 + z)
```

Closing remarks: The exp-normalize distribution is also known as a [Gibbs measure](#) (sometimes called a Boltzmann distribution) when it is augmented with a temperature parameter. Exp-normalize is often called "softmax," which is unfortunate because log-sum-exp is *also* called "softmax." However, unlike exp-normalize, it *earned* the name because it is actually a soft version of the max function, where as exp-normalize is closer to "soft argmax." Nonetheless, most people still call exp-normalize "softmax."

Posted by Tim Vieira Feb 11, 2014 [misc numerical](#)

Comments

2 Comments Graduate Descent**1** Login ▾

♥ Recommend 8 t Tweet f Share

Sort by Best ▾



Join the discussion...

LOG IN WITH

OR SIGN UP WITH DISQUS (?)



Name

**Jae Duk Seo** • 3 months ago

amazing trick!!!

^ | ▾ • Reply • Share >

**Mihir Gore** • a year ago

Good catch!

^ | ▾ • Reply • Share >

ALSO ON GRADUATE DESCENT

How to test gradient implementations

5 comments • a year ago

Tim Vieira — I am aware of the trick. I wrote about it in an earlier post! ...**Importance Sampling**

1 comment • 4 years ago

Adam — The link to great notes on variance reduction is down, but web archive to the ...**Dimensional analysis of gradient ascent**

2 comments • 2 years ago

Tim Vieira — Adaptive step size methods, e.g. Adagrad, Adadelta, and Adam, are generic ...**Gradient-based Hyperparameter ...**

4 comments • 2 years ago

Tim Vieira — The method I've described in this post treats the inner optimization algorithm as ...

✉ Subscribe d Add Disqus to your siteAdd DisqusAdd

🔒 Disqus' Privacy PolicyPrivacy PolicyPrivacy

Recent Posts

- [Black-box optimization](#)
- [Backprop is not just the chain rule](#)

- [Estimating means in a finite universe](#)
- [How to test gradient implementations](#)
- [Counterfactual reasoning and learning from logged data](#)

Tags

[optimization](#), [calculus](#), [automatic-differentiation](#), [sampling](#), [statistics](#), [reservoir-sampling](#), [testing](#), [counterfactual-reasoning](#), [importance-sampling](#), [machine-learning](#), [datastructures](#), [algorithms](#), [rant](#), [Gumbel](#), [decision-making](#), [hyperparameter-optimization](#), [misc](#), [numerical](#), [crf](#), [deep-learning](#), [structured-prediction](#), [visualization](#)

Copyright © 2014–2018 Tim Vieira — Powered by [Pelican](#)