The log-sum-exp trick

Christian Berrig

August 8, 2023

0 The problem

In numerical calculations, there are limitations to how many bits can be allocated to a number of a specific type, which means that for very small numbers and very large numbers, there are simply not enough bits to be allocated to represent such a number. In case of the large numbers, this is called an overflow problem (as one would need to take in use the "text bit" outside the allocation, whereby the number has "overflown") and for very small numbers an underflow problem.

If such an number is the final result of an computation, you are out of luck; then the number cannot be represented numerically, unless more bits are allocated to the number. However, if this is an intermediate result, the $LogSumExp$ (LSE) function or LSE-trick can be used to bypass an underflow or overflow problem, to yield the final result, which is in range of the number allocation.

In the following I will use the Gibbs-Boltzmann distribution as an example, as the goal here is to represent probabilities p_i from normalising collections of exponential functions, s.t.

$$p_i = \frac{\exp\{x_i\}}{\sum_{j=1}^{N} \exp\{x_j\}} \quad , \quad \forall i \in \{1, ..., N\}$$

1 The solution / why the trick works

Using the property of the logarithm:

$$\begin{aligned} p_i &= \frac{\exp\{x_i\}}{\sum_{j=1}^{N} \exp\{x_j\}} \\ \log p_i &= x_i - \log \sum_{j=1}^{N} \exp\{x_j\} \\ \log p_i &= x_i - \mathrm{LSE}\left(x_1, ..., x_N\right) \\ p_i &= \exp\{x_i - \mathrm{LSE}\left(x_1, ..., x_N\right)\} \end{aligned}$$

where

$$\mathrm{LSE}\left(x_1, ..., x_N\right) = \mathrm{LSE}\left(\boldsymbol{x}\right) = \log \sum_{j=1}^{N} \exp\{x_j\}$$

is the LogSumExp function.

This is only a matter or rewriting, but as we are still summing over the original exponential arguments, how does this help? This is where the trick enters: we factor out a common exponential factor, with the exponent equaling the largest exponent of the original arguments. That is:

$$\begin{aligned}
\mathrm{LSE}\left(x_1, ..., x_N\right) =& \log \sum_{j=1}^{N} \exp\{x_j\} \\
=& \log \exp\{c\} \sum_{j=1}^{N} \exp\{x_j - c\} \\
=& c + \log \sum_{j=1}^{N} \exp\{x_j - c\} \\
=& c + \mathrm{LSE}\left(x_1 - c, ..., x_N - c\right)
\end{aligned}$$

where $c = \max\left\{x_1, ..., x_N\right\}$

Now we are suddenly dealing with arguments of the exponentials that are all translated by the same constant ensuring that the exponentials themselves are *much* smaller than the original exponentials.

So applying this to the find the probability from above we can now write the full expression as:

$$\begin{aligned}
p_i =& \frac{\exp\{x_i\}}{\sum_{j=1}^{N} \exp\{x_j\}} \\
=& \exp\{x_i - \mathrm{LSE}\left(x_1, ..., x_N\right)\} \\
=& \exp\{x_i - c - \mathrm{LSE}\left(x_1 - c, ..., x_N - c\right)\}
\end{aligned}$$

which should be able to evaluate numerically.

Finally, before giving an example in code, note that we are not doing anything fancy here: in fact we are only reducing the original expression by a common factor:

$$\begin{aligned}
p_i =& \frac{\exp\{x_i\}}{\sum_{j=1}^{N} \exp\{x_j\}} \\
=& \frac{\exp\{x_i - c\}}{\sum_{j=1}^{N} \exp\{x_j - c\}} \\
=& \exp\{x_i - c - \mathrm{LSE}\left(x_1 - c, ..., x_N - c\right)\}
\end{aligned}$$

We went though all of the above for the purpose of not having to deal with exponentials of large numbers, so of cause you could start by subtracting the common factor (the max arg.) first and use the usual representation as well, however the LSE is an elegant way of functionalizing the problem, which makes the code more readable.

2 Coding examples

```
import numpy as np

xs = np.array([1000, 1000, 1000])

# Calculating p values the naive way will cause an overflow for
    large values in xs
p = lambda xs: np.exp(xs)/np.sum(np.exp(xs))

# Note that according to the arguments, the mapping p should give
    equal probability measure to each of the entries (as they are
    equanl in size).

def logsumexp(x):
    c = x.max()
    return c + np.log(np.sum(np.exp(x - c)))

# These two will not cause overflow
print(np.exp(xs - logsumexp(xs)))
print(p(xs - max(xs)))

# This will cause an overflow
print(p(xs))
```