

Problem Set 5

Due Friday October 30, 10 am

Comments

- This covers material in Unit 8.
- It's due at 10 am (Pacific) on Friday Oct. 30, both submitted as a PDF to Pensive as well as committed to your GitHub repository.
- As usual, your electronic solution should be in the form of an Quarto file named `ps5.qmd`, with Python code chunks. If you want to initially work in a Jupyter notebook, that is fine, but you'll need to run `quarto convert file.ipynb` to generate a qmd file before rendering to PDF and submitting.

Problems

1. Computation on a GPU often uses single precision (32 bits) floating point numbers or even half-precision (16 bits) (as part of “mixed” precision calculation), using the same form as the 64-bit representation we've been discussing, but with fewer bits. Answer the following questions for the **half-precision case**.
 - a. With half precision, discuss how one might allocate your “budget” of 16 bits in terms of precision and magnitude of numbers. There is no right answer; just discuss the tradeoff and decide on an allocation that seems reasonable to you.
 - b. Using that specific allocation, what is the largest number (approximately) that can be represented (i.e., when overflow would occur). You should report that number in both base 2 and base 10.
 - c. Using that specific allocation, what is the (exact) gap between numbers that can be represented in this representation, in base 2.
 - d. Based on your answer to (c), what is the approximate number of digits of precision in base 10?
 - e. Now look at `numpy.float16` and figure out what allocation it uses compared to what you came up with. You should do this by looking at `np.info(np.float16)` and inferring the allocation, not by asking AI or Google. Please don't do this until you've worked out parts (a)-(d). You won't learn as much if you don't work through it yourself first. Finish by showing some carefully chosen examples of `numpy.float16` numbers that demonstrate the range and precision are as indicated by what you see with `np.info(np.float16)`.
2. Consider the following estimates of the variance of a set of numbers. The results depend on whether the magnitude of the numbers is large or small. You can assume that for a vector w ,

$\text{var}(w)$ is calculated as $\sum_{i=1}^n (w_i - \bar{w})^2 / (n - 1)$.

```
import numpy as np
rng = np.random.default_rng(seed = 1)
def dg(x, form = '.20f'):
    print(format(x, form))

z = rng.normal(size = 100)
x = z + 1e12
## Calculate the empirical variances
dg(np.var(z))
dg(np.var(x))
```

```
0.72514887009499828796
0.72514631554484365594
```

Explain why these two estimates agree to only a small number of decimal places and which of the two is the more accurate answer, when mathematically the variance of $\mathbf{z}$ and the variance of $\mathbf{x}$ are exactly the same (since $\mathbf{x}$ is just the addition of a constant to $\mathbf{z}$). How many digits of accuracy do you expect in the less accurate of the two?

3. Consider the following, in which we run into problems when trying to calculate on a computer. Suppose I want to calculate a predictive density for new data (e.g., in a model comparison in a Bayesian context):

$$f(y^*|y, x) = \int f(y^*|y, x, \theta) \pi(\theta|y, x) d\theta = E_{\theta|y, x} f(y^*|y, x, \theta).$$

Here $\pi(\theta|y, x)$ is the posterior distribution (the distribution of the parameter, θ , given the data, y , and predictors, x). All of θ , y , and x will generally be vectors.

If we have a set of samples for θ from the posterior distribution, $\theta_j \sim \pi(\theta|y, x)$, $j = 1, \dots, m$, we can estimate that quantity for a vector of conditionally IID observations using a Monte Carlo estimate of the expectation:

$$f(y^*|y, x) \approx \frac{1}{m} \sum_{j=1}^m \prod_{i=1}^n f(y_i^*|y, x, \theta_j).$$

- a. Explain why I should calculate the product in the equation above on the log scale. What is likely to happen if I just try to calculate it directly?
- b. Here's a re-expression, using the log scale for the inner quantity,

$$\frac{1}{m} \sum_{j=1}^m \exp \sum_{i=1}^n \log f(y_i^*|y, x, \theta_j),$$

which can be re-expressed as

$$\frac{1}{m} \sum_{j=1}^m \exp(v_j)$$

where

$$v_j = \sum_{i=1}^n \log f(y_i^* | y, x, \theta_j).$$

Thus we have

$$\log f(y^* | y, x) \approx \log \left(\frac{1}{m} \sum_{j=1}^m \exp(v_j) \right).$$

What is likely to happen when I try to exponentiate v_j ? Figure out how you could calculate this log predictive density without running into that problem.