HOMEWORK
David F. Gleich
October 28, 2025

Homework 4

Please answer the following questions in complete sentences in a clearly prepared manuscript and submit the solution by the due date on Gradescope (Due morning of Nov 6.)

Remember that this is a graduate class. There may be elements of the problem statements that require you to fill in appropriate assumptions. You are also responsible for determining what evidence to include. An answer alone is rarely sufficient, but neither is an overly verbose description required. Use your judgement to focus your discussion on the most interesting pieces. The answer to "should I include 'something' in my solution?" will almost always be: Yes, if you think it helps support your answer.

Problem 0: Homework checklist

- Please identify anyone, whether or not they are in the class, with whom you discussed your homework. This problem is worth 1 point, but on a multiplicative scale.
- Make sure you have included your source-code and prepared your solution according to the most recent Piazza note on homework submissions.

Problem 1: Flop counts

1. Let $\boldsymbol{E}$ be a matrix of the form

$$\boldsymbol{E} = \begin{bmatrix} \boldsymbol{D} & \mathbf{u} \\ \mathbf{v}^T & \alpha \end{bmatrix}$$

   where $\boldsymbol{D}$ is a diagonal matrix. Compute a flop count for computing $\boldsymbol{E}^{-1}\boldsymbol{A}$ where $\boldsymbol{E}$ is $n \times n$ (so $\boldsymbol{D}$ is $n-1 \times n-1$) and $\boldsymbol{A}$ is $n \times c$. You may lose points if your count is not $O(nc)$.

2. Consider computing trace$(\boldsymbol{A}^T\boldsymbol{B})$ where $\boldsymbol{A}$ and $\boldsymbol{B}$ are in compressed sparse column format and have the same dimensions. Suppose we have access to a fused multiply add operation. This is an operation that takes three inputs α, γ, β and computes
   $\theta \leftarrow \alpha * \gamma + \beta$ in a single operation. Explain how to use this operation when computing the trace and compute how much such operations you need along with any other floating point operations (additions, multiplications, divisions, etc.)

Problem 2: Inner-products are backwards stable.

1. Find a proof that computing $\mathbf{x}^T\mathbf{y}$ is backwards stable. Explain this proof in enough detail for a classmate to understand it without having read the document. This could take up to a page to give enough detail.

2. Show that computing a matrix-vector product $\mathbf{y} = \boldsymbol{A}\mathbf{x}$ is backwards stable.

Problem 3: Accurate summation

Consider a list of n numbers. For simplicity, assume that all numbers are positive so you don't have to write a lot of absolute values.

1. Show that the following algorithm is backwards stable.

```
function mysum(x::Vector{Float64})
  s = zero(Float64)
  for i=1:length(x)
    s += x[i]
  end
  return s
end
```

   Which requires showing that $\text{mysum}(\mathbf{x}) = \sum \hat{x}_i$ where $\|\hat{\mathbf{x}} - \mathbf{x}\|/\|\mathbf{x}\| \leq C_n\varepsilon$ where ε is the unit-roundoff for Float64. (You may want to solve problem 2 first.)

2. Consider adding three positive numbers together a, b, c. Describe how to compute $s = a + b + c$ with the greatest accuracy.

3. Use the results of part 2 to describe a way to permute the input $\mathbf{x}$ to `mysum` to attain the greatest accuracy. Find an input vector $\mathbf{x}$ where this new ordering gives a measurable change in the floating point accuracy as determined by the number of correct digits in the mantissa. (Hint, this means you should know the true sum of your vector so that you can identify it's best floating point representation.)

4. Lookup the Kahan summation algorithm and implement it to sum a vector. Compare the accuracy with what you found in part 3.

Problem 4: Quadratic equations

Read through the stack exchange post on solving quadratic equations. https://math.stackexchange.com/questions/311382/solving-a-quadratic-equation-with-precision-when-using-floating-point-variables

This suggests a number of approaches to compute the roots of a quadratic equation through closed form solutions.

An alternative approach is to use an iterative algorithm to estimate that root of an equation. In this case, we can use a simple bisection approach, which works quite nicely for finding the root.

Your task for this problem is to implement a bisection algorithm to return all the solutions of $ax^2 + bx + c = 0$ when $c \neq 0$.

```
""" Return all the solutions to ax^2 + bx + c. It is acceptable to return
NaN instead of a root as well. """
function roots(a::Float32,b::Float32,c::Float32)
end
```

The input to this method is Float32 so you can compare to higher-accuracy solutions with Float64 and to elucidate some of the issues that arise with slightly lower-precision.

Compare the accuracy of this procedure to the methods suggested on the stack exchange page and explain your results. Note that you may need to look for extremal inputs. In this case, Float32 is handy because there are only 4 billion inputs for each input value a, b, c. This is still too many to test all combinations. But there are only two choices for the roots, which greatly reduces the space.

Problem 5: Condition numbers

Consider the following computations. Discuss if they are well-conditioned or ill-conditioned. If the answer depends on the types of input, please provide some rough guidance. (e.g. for subtraction, it's ill-conditioned if the numbers are close by)

1. The entropy of a probability distribution is $H(\mathbf{p}) = -\sum_{i=1}^{n} p_i \log p_i$ where $0 < p_i < 1$. Compute the condition number of the entropy function.
2. A matrix vector product $\mathbf{y} = \boldsymbol{A}^T\mathbf{x}$.
3. Evaluating a neural network layer $\mathbf{y} = f(\boldsymbol{Q}^T\mathbf{x})$ where the elements are $y_i = f(w_i^T x)$ and f the soft-plus function $\log(1+e^x)$ *and* $\boldsymbol{Q}$ *is an orthogonal matrix*

Problem 6: Experience with the SVD

Produce the analytic SVDs of the following matrices. (That is, no numerical approximations, but feel free to let Julia give you a good guess!). It's important to think about these questions because I may give similar questions on a midterm or final and you'll be expected to remember these. It's also handy to think about how to construct these, even though we haven't seen any algorithms yet. You should be able to work them all out directly from the definition.

1. $\begin{bmatrix} 0 & -3 \\ 0 & 0 \end{bmatrix}$
2. $\begin{bmatrix} -5 & 0 \\ 2 & 0 \end{bmatrix}$
3. $\begin{bmatrix} 1 & -2 \\ 2 & -4 \\ 0 & 0 \end{bmatrix}$
4. $\begin{bmatrix} 2 & 0 \\ 0 & 5 \end{bmatrix}$

Problem 7: Backwards stability

1. Let $f(x) = \sqrt{x}$. Suppose you have an algorithm where $\texttt{myf}(x) = \sqrt{x+\mu}$ where μ is the machine precision. Is $\texttt{myf}(x)$ backwards stable?
2. Suppose that you have a fancy implementation of $\sqrt{x}$ and you compute $\texttt{mysqrt}(0.1\mu) = -1 \cdot 10^{-16}\mu$. Is this a backwards stable implementation?

Problem 8: Condition numbers

Show that

$$\frac{1}{\kappa(\boldsymbol{A})}$$

measures the relative distance from $\boldsymbol{A}$ to the space of singular matrices. That is

$$\frac{1}{\kappa(\boldsymbol{A})} = \text{ smallest } \frac{\|\boldsymbol{D}\|}{\|\boldsymbol{A}\|} \text{ such that } \boldsymbol{A} + \boldsymbol{D} \text{ is singular.}$$

Everything here involves the matrix 2-norm.

Problem 9: More experience with the SVD

In this problem, we are going to compute an SVD of data derived from a transformer-based function. Specifically, we are going to use a function underlying the small gpt2 transformer. (This is an ancestor of the recently released GPT-5).

The things you need to know about transformers is we give them an input as a sequence of token values. The tokens are numbered between 0 and $50,256$. These represent parts of words. The output is a matrix of low-dimensional embeddings where each row is a d dimensional vectors ($d = 768$) associated with each input token. (If you've seen transformers formally before, we are looking at the GPT-2 function *before* the final vocabulary transformation as this keeps the data more manageable.)

As a mathematical function then, we have

$$f(S) : \text{length n token sequence} \to \mathbb{R}^{n \times d}.$$

Since the token sequence is one-hot encoded, that means we associate the vector e_i with token i. In which case the input is actually $\boldsymbol{X} = \mathbb{R}^{n \times T}$ where $T = 50{,}257$\$. (We increase by one since we are 1-indexed.)

The function is implemented in this file https://www.cs.purdue.edu/homes/dgleich/cs515-2025/homeworks/gpt2.jl (This includes a number of other things associated with the lecture in the textbook on Transformers that may be independently interesting, but is not required for this problem.) The function you want is `gpt2func` and you can call it like this.

```
include("gpt2.jl") # this will download the model if needed
gpt2func([19044, 45977, 389, 616, 2460, 290], gpt2model)
```

If you have an issue with the code, comment out the lines that use `_add_label` function as they use characters that are utf-8 encoded and for some reason the purdue cs server doesn't seem to want to send it in utf8 all the time.

This will return a 6×768 matrix from the final layer of the transformer. If you want to play around with `gpt2`, there

We are going to use the SVD to compare random text and with token sequences from Wikipedia.

In the file https://www.cs.purdue.edu/homes/dgleich/cs515-2025/homeworks/wikitext_tokens.txt you will find 10000 random sequences of tokens from Wikipedia of various length. You can read this input file in Julia with the function

```
function read_token_sequence(file="wikitext_tokens.txt")
  open(file, "r") do io
    tokens = Vector{Vector{Int}}()
    for line in eachline(io)
      if startswith(line, "#")
        continue
      end
      if length(strip(line)) == 0
        continue
      end
      vals = split(line, ",")
      toks = parse.(Int, vals)
      push!(tokens, toks)
    end
```

```
    return tokens
  end
end
tokens = read_token_sequence()
```

Given this result, we can get the matrix for the first token sequence like this

```
Y1 = gpt2func(tokens[1], gpt2model)
```

The idea is to to take the first 100 sequences of tokens from this file and compute the function $f(S)$ for each. This will give use 100 different output matrices $\boldsymbol{Y}_1, \ldots, \boldsymbol{Y}_{100}$. Then we vertically concatenate all the output matrices into a giant matrix

$$\boldsymbol{Y} = \begin{bmatrix} \boldsymbol{Y}_1 \\ \vdots \\ \boldsymbol{Y}_{100} \end{bmatrix}.$$

Let $\mathbf{s}_{\text{text}}$ be the singular values of $\boldsymbol{Y}$. (Hint, I get the largest singular value is 27738.82.) I used the function `svdvals` in Julia for this, you need not implement your own algorithm.

Next, we want to compare this to random token data. Suppose S is a length n token sequence we used above. Then we want to generate a length n token sequence where we draw token id's randomly from 0 to $50,256$ (the code uses zero based indices to make it easier to compare with Python).

If we generate a random sequence R to match the length of the first 100 sequences above, and then compute that function, we get 100 different output matrices $\boldsymbol{Z}_1, \ldots, \boldsymbol{Z}_{100}$ of exactly the same sizes. We assemble these into a giant matrix $\boldsymbol{Z}$ as before and then compute their singular values to get $\mathbf{s}_{\text{random}}$.

1. What do you notice when you compare $\mathbf{s}_{\text{text}}$ to $\mathbf{s}_{\text{random}}$?
2. Suppose we did this SVD computation for all 10,000 sequences, what computational challenges do we encounter? Can you overcome them? (You do not have to overcome them for full points, but this is feasible to do using what we have learned in class.)
3. The actual output from the transformer is usually a sequence of logits that can be rounded back to tokens. What this means is that we take the output $\boldsymbol{Y}$ that we get and multiply it by a $768 \times 50,257$ dimensional matrix $\boldsymbol{E}^T$. This is known as the token decoding. Give an procedure to compute the SVD of $\boldsymbol{Y}\boldsymbol{E}^T$ without actually computing the matrix $\boldsymbol{A} = \boldsymbol{Y}\boldsymbol{E}^T$ assuming that $\boldsymbol{Y}$ and $\boldsymbol{E}$ are full-rank. (Hint: use one or more QR factorizations.) You are only allowed to do an SVD of a matrix with 768 columns.