Homework 1 Solutions

January 20, 2022

Problem 1 [$6 \times$ (3 points analytical + 1 point numerical)]

See the **supplementary Julia notebook** for example numerical validations.

a) $f(x) = (x^T x)^4$: The result is a scalar, which makes it easier to apply the chain rule without worrying about commuting things: $df = 4(x^T x)^3 d(x^T x) = 4(x^T x)^3 (dx^T x + x^T dx) = 8(x^T x)^3 x^T dx$, and as in class we can also write this as a gradient column vector $\nabla f = 8(x^T x)^3 x$, such that $df = (\nabla f)^T dx$.

b) $f(x) = \cos(x^T Ax)$: again applying the chain rule, and using the derivative of $x^T Ax$ from class, we immediately get $\nabla f = -\sin(x^T Ax)(A + A^T)x$

c) $f(A) = \operatorname{trace}(A^4)$: here, we have to be a little careful with the product rule for $d(A^4) = dA\, A^3 + A\, dA\, A^2 + A^2 dA\, A + A^3\, dA$ because A and dA do not commute. $df = \operatorname{trace} d(A^4)$ (by linearity of trace), which $= \operatorname{trace}(dA\, A^3) + \operatorname{trace}(A\, dA\, A^2) + \operatorname{trace}(A^2 dA\, A) + \operatorname{trace}(A^3 dA)$, again by linearity. But by the cyclic property of the trace we have $\operatorname{trace}(A\, dA\, A^2) = \operatorname{trace}(A^3 dA)$ etc., and hence $df = 4 \operatorname{trace}(A^3 dA)$. Equivalently, from class we can write $\nabla f = 4(A^3)^T$.

d) $f(A) = A^4$: From above, $df = dA\, A^3 + A\, dA\, A^2 + A^2 dA\, A + A^3\, dA$, which is a linear operator acting on dA.

e) $f(A) = \theta^T A$: this is simply $df = \theta^T dA$, which is a linear operator on dA. (Easy, right? But this problem reportedly drastically confused 6.036 students who weren't used to thinking of derivatives as linearization.)

f) $f(x) = \sin.(x)$ (elementwise sine): The derivative is simply $df = \cos.(x).*dx$, where $.*$ denotes the elementwise "Hadamard" product as in Julia or Matlab.

More generally, we can easily derive the rule that $d(g.(x)) = g'.(x) .* dx$ for any scalar function g applied elementwise. One way to see it is to write it out component-wise:

$$d(g.(x)) = d\begin{pmatrix} g(x_1) \\ g(x_2) \\ \vdots \\ g(x_n) \end{pmatrix} = \begin{pmatrix} g'(x_1)dx_1 \\ g'(x_2)dx_2 \\ \vdots \\ g'(x_n)dx_n \end{pmatrix} = g'.(x) .* dx\,.$$

Elementwise functions are somewhat alien to ordinary linear algebra (among other things, they are basis-dependent), but are common in many fields from machine learning ("activation" functions are applied elementwise to a layer of a neural network) to data science, and once we write out the rules we see that they are quite simple.

To write this as a Jacobian matrix, realize that the elementwise product $a .* dx$ is equivalent to multiplying dx on the left by a *diagonal matrix* whose diagonal entries are the components of the vector a. This matrix is denoted $\operatorname{diagm}(a)$ in Julia, and $\operatorname{diag}(a)$ in Matlab and Numpy, so we could therefore see that our Jacobian matrix is $\operatorname{diagm}(g'.(x))$, or

$$f'(x) = \operatorname{diagm}(\cos.(x)) = \begin{pmatrix} \cos(x_1) & & & \\ & \cos(x_2) & & \\ & & \ddots & \\ & & & \cos(x_n) \end{pmatrix}$$

in this case.

Problem 2 (15+5 points)

part (a):

We have $dL = d[(x_N(p) - y_0)^T(x_N(p) - y_0)] = 2(x_N(p) - y_0)^T dx_N$ (similar to our derivative $d(x^T x) = 2x^T dx$ in class), and differentiating the recurrence for each k, we also have $dx_k = \frac{\partial f_k}{\partial p}dp + \frac{\partial f_k}{\partial x_{k-1}}dx_{k-1}$ as in class. Putting these two together, we have:

$$\begin{aligned} dL &= 2(x_N(p) - y_0)^T \left(\frac{\partial f_N}{\partial p}dp + \frac{\partial f_N}{\partial x_{N-1}}dx_{N-1} \right) \\ &= 2(x_N(p) - y_0)^T \left(\frac{\partial f_N}{\partial p}dp + \frac{\partial f_N}{\partial x_{N-1}} \left(\frac{\partial f_{N-1}}{\partial p}dp + \frac{\partial f_{N-1}}{\partial x_{N-2}}dx_{N-2} \right) \right) \\ &= 2(x_N(p) - y_0)^T \left(\frac{\partial f_N}{\partial p}dp + \frac{\partial f_N}{\partial x_{N-1}} \left(\frac{\partial f_{N-1}}{\partial p}dp + \frac{\partial f_{N-1}}{\partial x_{N-2}} \left(\frac{\partial f_{N-2}}{\partial p}dp + \frac{\partial f_{N-2}}{\partial x_{N-3}} (\cdots) \right) \right) \right) \end{aligned}$$

As discussed in class, since we have a single scalar output and many inputs p, the trick is to *evaluate left-to-right* so that we are always doing vector–matrix multiplications and *never* matrix–matrix multiplications. This is straightforward computationally, but writing it out formally as a recurrence relation involves a little thought. We have to keep track of the row vector that we accumulate, starting from the left-most row vector $2(x_N(p) - y_0)^T$. Let's give this a name:

$$\boxed{z_N^T = 2(x_N(p) - y_0)^T}.$$

Then, let's define subsequent elements of z_k by grouping vector–matrix products in dL:

$$\begin{aligned} dL &= z_N^T \left(\frac{\partial f_N}{\partial p}dp + \frac{\partial f_N}{\partial x_{N-1}} \underbrace{(\cdots)}_{dx_{N-1}} \right) \\ &= z_N^T \frac{\partial f_N}{\partial p}dp + \underbrace{z_N^T \frac{\partial f_N}{\partial x_{N-1}}}_{z_{N-1}} \underbrace{\left(\frac{\partial f_{N-1}}{\partial p}dp + \frac{\partial f_{N-1}}{\partial x_{N-2}} \underbrace{(\cdots)}_{dx_{N-2}} \right)}_{dx_{N-1}} \\ &= z_N^T \frac{\partial f_N}{\partial p}dp + z_{N-1}^T \frac{\partial f_{N-1}}{\partial p}dp + \underbrace{z_{N-1}^T \frac{\partial f_{N-1}}{\partial x_{N-2}}}_{z_{N-2}} \underbrace{(\cdots)}_{dx_{N-2}} \\ &= \cdots = \underbrace{\left(\sum_{k=1}^{N} z_k^T \frac{\partial f_k}{\partial p} \right)}_{(\nabla L)^T} dp\,, \end{aligned}$$

where we have defined a (linear!) recurrence:

$$\boxed{z_{k-1}^T = z_k^T \frac{\partial f_k}{\partial x_{k-1}}}.$$

Each step of this "backwards" (**descending** k) recurrence ("backpropagation", "reverse-mode"), starting with z_N and then computing z_{N-1} and so on, only involves a vector–matrix multiply, so it is fast. We then have the gradient:

$$\boxed{\nabla L = \sum_{k=1}^{N} \left(\frac{\partial f_k}{\partial p} \right)^T z_k}.$$

which again involves N matrix–vector multiplications.

part (b):

In general, if $\partial f_k/\partial x_{k-1}$ and/or $\partial f_k/\partial p$ are sparse (mostly zeros), the vector–matrix and matrix–vector multiplications above (respectively) can be simplified because you only need to multiply by the nonzero parts of the matrix.

In particular, the problem asks you what happens if f_k only depends upon the k-th block of parameters p_k. Suppose $x_k \in \mathbb{R}^{m_k}$ (the size of the NN "layer"), so that f_k has m_k outputs. Then we obtain a sparse $\partial f_k/\partial p$ with the following $m_k \times Nn$ form:

$$\begin{pmatrix} 0 & 0 & \cdots & \underbrace{\frac{\partial f_k}{\partial p_k}}_{m_k \times n} & 0 & 0 & \cdots & 0 \end{pmatrix},$$

where each "0" represents an $m_k \times n$ *block* of zeros, and the only nonzero block is $\partial f_k/\partial p_k$, which appears in the k-th column block.

In consequence, we gain efficiency in the computation of ∇L above by skipping the multiplications by these zero blocks. Furthermore, in the ∇L summation, most of the elements of each vector in the sum is zero, and we can skip the addition of zeros. Finally, we obtain the simplification:

$$\boxed{\nabla L = \begin{pmatrix} \left(\frac{\partial f_1}{\partial p_1}\right)^T z_1 \\ \left(\frac{\partial f_2}{\partial p_2}\right)^T z_2 \\ \vdots \\ \left(\frac{\partial f_N}{\partial p_N}\right)^T z_N \end{pmatrix}}.$$

Problem 3 (5+10+5+5 points)

a) $f(v) = \int_0^1 \sin(v(x))dx$, so

$$df = f'(v)[dv] = f(v+dv) - f(v) = \int_0^1 (\sin(v+dv) - \sin(v))\, dx = \boxed{\int_0^1 \cos(v(x))\, dv(x)\, dx},$$

which is a linear operator acting on perturbation *functions* $dv(x)$ (or δv, dropping terms beyond first order).

b) $g(v) = \int_0^1 \sqrt{1+v'(x)^2}dx$, so similarly expanding the integrand for $v'+dv'$ to first order in dv' (just a scalar Tayor expansion of $\sqrt{1+y^2}$ in y), we obtain the linearization $dg = g'(v)[dv] = \int_0^1 \frac{v'(x)}{\sqrt{1+v'(x)^2}} dv'(x)\, dx$. In order to get the integral in terms of dv, not dv', we integrate by parts:

$$dg = \left.\frac{v'(x)}{\sqrt{1+v'(x)^2}} dv\right|_0^1 - \int_0^1 \left(\frac{v'(x)}{\sqrt{1+v'(x)^2}}\right)' dv(x)dx\,,$$

where the first term is *zero* because $dv \in V$ vanishes on the endpoints. After a bit of high-school calculus to take the derivative and put the integrand over a common denominator, we obtain:

$$\boxed{dg = g'(v)[dv] = \int_0^1 \frac{-v''(x)}{(1+v'(x)^2)^{3/2}} dv(x)\, dx}.$$

c) To have $dg = 0$ for *any* dv, it is clear that we must have

$$\frac{v''(x)}{(1+v'(x)^2)^{3/2}} = 0$$

for $x \in [0,1]$ (almost everywhere) in the previous part. But this only happens if $\boxed{v'' = 0}$, a straight line! Furthermore, since we required $v(0) = v(1) = 0$, the only possible straight line is $\boxed{v(x) = 0}$. In this case $g(v) = 0$, which is clearly the **minimum** value of g since $g \geq 0$ by definition.

d) Geometrically, $g(v)$ is the **length** of the curve $v(x)$ (this $g(v)$ integral should be a familiar formula from 18.01!), and so its **minimum** occurs when $v(x)$ is a **straight line between the endpoints**.

This is a lot of effort just to find a solution that is a straight line, but is an example of a powerful technique from "**calculus of variations**": if $f(v) = \int F(v, v')dx$ for some integrand F, we linearize df and integrate by parts to obtain a linear functional $df = f'(v)[dv]$ as some integral of dv. Then by setting the integrand to zero, we obtain a second-order differential equation in v called the "Euler–Lagrange" equation, which can then be solved using 18.03 techniques (and beyond) and/or numerical methods to determine the v for which $f(v)$ is maximal/minimal.

Problem 4 (10 points)

For $Y(S) = A^T S A$, we have $dY = A^T dS A$, a linear operation on the input perturbation dS. If we write this out explicitly for 2×2 symmetric matrices and doing all of the multiplications by hand, we obtain:

$$\begin{aligned} dY &= \begin{pmatrix} dy_{11} & dy_{12} \\ dy_{12} & dy_{22} \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}^T \begin{pmatrix} ds_{11} & ds_{12} \\ ds_{12} & ds_{22} \end{pmatrix} \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \\ &= \begin{pmatrix} a_{11} & a_{21} \\ a_{12} & a_{22} \end{pmatrix} \begin{pmatrix} ds_{11}a_{11} + ds_{12}a_{21} & ds_{11}a_{12} + ds_{12}a_{22} \\ ds_{12}a_{11} + ds_{22}a_{21} & ds_{12}a_{12} + ds_{22}a_{22} \end{pmatrix} \\ &= \begin{pmatrix} a_{11}^2 ds_{11} + 2a_{21}a_{11}ds_{12} + a_{21}^2 ds_{22} & a_{11}a_{12}ds_{11} + (a_{11}a_{22} + a_{21}a_{12})ds_{12} + a_{21}a_{22}ds_{22} \\ \text{(symmetric)} & a_{12}^2 ds_{11} + 2a_{12}a_{22}ds_{12} + a_{22}^2 ds_{22} \end{pmatrix}. \end{aligned}$$

If we now write the inputs and outputs as 3-component vectors, we can then read off the entries of the Jacobian matrix (outputs=rows, inputs=columns!) from above:

$$\begin{pmatrix} dy_{11} \\ dy_{12} \\ dy_{22} \end{pmatrix} = \boxed{\begin{pmatrix} a_{11}^2 & 2a_{21}a_{11} & a_{21}^2 \\ a_{11}a_{12} & a_{11}a_{22} + a_{21}a_{12} & a_{22}a_{21} \\ a_{12}^2 & 2a_{12}a_{22} & a_{22}^2 \end{pmatrix}} \begin{pmatrix} ds_{11} \\ ds_{12} \\ ds_{22} \end{pmatrix},$$

where the boxed term is the desired Jacobian.