HOMEWORK
David F. Gleich
August 27, 2018

Homework 1

Please answer the following questions in complete sentences in a clearly prepared manuscript and submit the solution by the due date on Blackboard (around Sunday, September 2nd, 2018.)

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: Operations

Compute the following by hand or using Julia. The vector $\mathbf{e} = \begin{bmatrix}1 & \dots & 1\end{bmatrix}^T$ (i.e. the all ones vector).

1. $\begin{bmatrix}1 & 1 & 2\\ 3 & 5 & 8\\ 13 & 21 & 34\end{bmatrix}\begin{bmatrix}1 & -2 & 3\\ -4 & 5 & -6\\ 7 & -8 & 9\end{bmatrix} = ?$
2. $\mathbf{x} =$ `ones(1000,1)` $\mathbf{y} =$ `1:1000` $\mathbf{x}^T\mathbf{y} = ?$

   (Optional extra question – worth no points – who is always credited with discovering a very efficient way to compute this as a child?)

   Numpy users $\mathbf{x} =$ `ones((1000,1))` $\mathbf{y} =$ `arange(1.,1001.)[:, newaxis]'`
3. $\mathbf{x} = \begin{bmatrix}1.5 & 2 & -3\end{bmatrix}^T$. (Assume $\mathbf{e}$ is 4×1.)
   $\mathbf{e}\mathbf{x}^T = ?$
   $\mathbf{x}\mathbf{e}^T = ?$
4. $\mathbf{x} = \begin{bmatrix}-5 & 4 & 2\end{bmatrix}^T$. (Assume $\mathbf{e}_i$ is 3×1.) $\mathbf{e}_1\mathbf{x}^T = ?$
   $\mathbf{x}\mathbf{e}_3^T = ?$

Problem 2: A proof

Let $\boldsymbol{A}$ and $\boldsymbol{C}$ be invertible matrices. We'll prove that the inverse of $\begin{bmatrix} \boldsymbol{A} & \boldsymbol{B} \\ 0 & \boldsymbol{C} \end{bmatrix}$ is easy to determine!

1. Show that the inverse of
$$\begin{bmatrix} 1 & a \\ 0 & 1 \end{bmatrix}$$
is
$$\begin{bmatrix} 1 & -a \\ 0 & 1 \end{bmatrix}.$$

2. Now, show that the inverse of
$$\begin{bmatrix} \boldsymbol{I} & \boldsymbol{A} \\ 0 & \boldsymbol{I} \end{bmatrix}$$
is
$$\begin{bmatrix} \boldsymbol{I} & -\boldsymbol{A} \\ 0 & \boldsymbol{I} \end{bmatrix}.$$

3. Recall that for general $\boldsymbol{A}$ and $\boldsymbol{B}$, not those in the problem!, $(\boldsymbol{AB})^{-1} = \boldsymbol{B}^{-1}\boldsymbol{A}^{-1}$. Use this fact, and the result of problem 2.2 to determine the inverse to $\begin{bmatrix} \boldsymbol{A} & \boldsymbol{B} \\ 0 & \boldsymbol{C} \end{bmatrix}$ when $\boldsymbol{A}$ and $\boldsymbol{C}$ are invertible. Alternatively, give the inverse of $\begin{bmatrix} \boldsymbol{A} & \boldsymbol{B} \\ 0 & \boldsymbol{C} \end{bmatrix}$. Hint: think about diagonal matrices!

Problem 3: Simplifying a matrix expression

The following derivation occured when I was working on a proof with a student that we recently used in a research paper. We have an expression:

$$\mathbf{q}(\mathbf{x}) = \text{Diag}((\boldsymbol{I} - \boldsymbol{H})\mathbf{x}) \cdot (\boldsymbol{I} + \boldsymbol{H})\mathbf{x}$$

where $\boldsymbol{H}$ is a square, non-negative matrix and

$$\text{Diag}(\mathbf{z}) = \begin{bmatrix} z_1 & 0 & \dots & 0 \\ 0 & z_2 & \dots & 0 \\ & & \ddots & \\ 0 & \dots & 0 & z_n \end{bmatrix}$$

(which is a diagonal matrix where $\mathbf{z}$ is on the diagonal). We needed to show that:

$$\mathbf{y}^T\mathbf{q}(\mathbf{x}) = \mathbf{x}^T\boldsymbol{C}\mathbf{x}$$

for some matrix $\boldsymbol{C}$ that depends on $\mathbf{y}$. Your goal in this problem is to work out an expression for $\boldsymbol{C}$.

This entire problem can be done element-wise, but the proof and results are fairly simple if you embrace matrix notations.

1. (Very easy.) As a warm-up, show that if $\boldsymbol{F}$ and $\boldsymbol{G}$ are square diagonal matrices, then $\boldsymbol{FG} = \boldsymbol{GF}$.

2. Show that $\mathbf{q}(\mathbf{x}) = \text{Diag}(\mathbf{x})\mathbf{x} - \text{Diag}(\boldsymbol{H}\mathbf{x})\boldsymbol{H}\mathbf{x}$. (Note that you need to use some properties of diagonal matrices in order to show this.)

3. Write an expression of $\boldsymbol{C}$ in terms of $\mathbf{y}$ using the simplified version from part 2.

Problem 4: Deep neural nets

This problem will be more difficult if you haven't used Julia or Matlab before, so get started early! It's designed to teach you about writing `for` loops to construct a matrix operation for a particular task.

Deep neural networks for image recognition are a new technology that demonstrates impressive results. In this problem, we will build matrices that let us simulate what would happen for a random deep neural net.

A deep neural network is a sequence of matrix-vector products and non-linear functions. Let $\mathbf{x}$ represent the input to the neural net. Then a deep net computes a function such as:

$$\text{deepnet}(\mathbf{x}) = f(\boldsymbol{W}_k \cdots f(\boldsymbol{W}_3 f(\boldsymbol{W}_2 f(\boldsymbol{W}_1 \mathbf{x}))) \cdots)$$

where f is a non-linear function and $\boldsymbol{W}_i$ is a weight matrix where $\boldsymbol{W}_i$ has fewer rows that columns. (So the output of $\boldsymbol{W}_i \mathbf{x}$ is a *smaller* vector than the input.) These are called *deep* neural networks because the number of layers k is usually large (hundreds). A popular choice for f is the function:

$$f(x) = \begin{cases} x & x > 0 \\ 0 & x \leq 0 \end{cases}$$

which is called the ReLU (rectified linear unit). (These units model a "neuron" that "activates" whenever x is positive and is non-active when x is non-positive.) Also, the function could vary with the layer. Other choices involved in deep neural net architecture are: * how many layers (or how many functions and weight matrices)? * what is the shape of a weight matrix? We don't wish to get into too many of the details here. We are going to have you evaluate a simple neural network architecture.

Given an input image as a 32×32 pixel image, where each pixel is a real-valued number between 0 and 1, we want to simulate the following neural network architecture:

$$\text{ournet}(\mathbf{x}) = f(\boldsymbol{W}_3 f(\boldsymbol{W}_2 f(\boldsymbol{W}_1 \mathbf{x})))$$

where

- $\boldsymbol{W}_1$ is a 256×1024 matrix that is a rudimentary edge-detector, the result of $f(\boldsymbol{W}_1 \mathbf{x})$ should be large if there is an edge (see below for how to do this);
- f is a ReLU unit;
- $\boldsymbol{W}_2$ is another 64×256 matrix which is the same type of edge detector.
- $\boldsymbol{W}_3$ is a 1×64 matrix that simply sums the output of all the inputs. (Hence, $\boldsymbol{W}_3 = \mathbf{e}^T$ where $\mathbf{e} \in \mathbb{R}^{64}$.)

So the challenge here is to build $\boldsymbol{W}_1$ and $\boldsymbol{W}_2$. As mentioned, these are extremely crude edge-detectors. (There are much better things you can do, but I wanted to keep this fairly simple.)

We'll illustrate the construction of our edge detector with a 4×4 image. Suppose this input is represented by a matrix:

$$\text{input} = \begin{bmatrix} x_1 & x_5 & x_9 & x_{13} \\ x_2 & x_6 & x_{10} & x_{14} \\ x_3 & x_7 & x_{11} & x_{15} \\ x_4 & x_8 & x_{12} & x_{16} \end{bmatrix}.$$

What we want to do is output a 2×2 result:

$$\text{output} = \begin{bmatrix} y_1 & y_3 \\ y_2 & y_4 \end{bmatrix}$$

where

$$y_1 = x_1 + x_6 - x_5 - x_2$$
$$y_2 = x_3 + x_8 - x_7 - x_4$$
$$y_3 = x_9 + x_{14} - x_{13} - x_{10}$$
$$y_4 = x_{11} + x_{16} - x_{15} - x_{12}$$

This particular formula comes from applying a 2×2 computational stencil:

$$\begin{bmatrix} + & - \\ - & + \end{bmatrix}$$

to each 2×2 region of the input image to reduce it to a single number. (As mentioned a few times, if this number is positive, it means there is an edge or high-contrast region somewhere inside that 2×2 block, but there are much better ways to solve this problem! This is really just a simple example.)

Now, I described the input and output in terms of *matrices* above. However, the deep neural network architecture works with a *vector* input and produces a *vector* output. So we have to rework this a little bit.

1. Note that the input and output matrices have a linear ordering $x_1, \ldots, x_{16}$ and $y_1, \ldots, y_4$. Let $\mathbf{x} \in \mathbb{R}^{16}$ and $\mathbf{y} \in \mathbb{R}^4$. Write down the matrix $\boldsymbol{A}$ such that $\mathbf{y} = \boldsymbol{A}\mathbf{x}$.

2. We will want to run this on real images. Download

   - http://www.cs.purdue.edu/homes/dgleich/cs515-2018/homeworks/image1.png
   - http://www.cs.purdue.edu/homes/dgleich/cs515-2018/homeworks/image2.png
   - http://www.cs.purdue.edu/homes/dgleich/cs515-2018/homeworks/image3.png

   (Before you do the next step, you may have to add the following packages)

   ```
   using Pkg; Pkg.add(["Images","FileIO","ImageMagick"]) # on Linux
   using Pkg; Pkg.add(["Images","FileIO","QuartzImageIO"]) # on OSX
   ```

   Load these images in Julia and convert them into matrices.

   ```
   using FileIO, Images
   X1 = Float64.(load("image1.png"))
   X2 = Float64.(load("image2.png"))
   X3 = Float64.(load("image3.png"))
   ```

   Report the result of

   ```
   using LinearAlgebra
   tr(X1+X2+X3) # this computed the trace
   ```

   If you wish to look at the images (not required) then run

   ```
   colorview(Gray, X1)
   ```

   Alternatively, you can use

   ```
   using Plots
   pyplot()
   heatmap(X1,yflip=true, color=:gray)
   gui()
   ```

3. In what follows, we'll talk about two different types of indices. The image index of a pixel is a pair (i, j) that identifies a row and column for the pixel in the image. The vector index of a pixel is the index of that pixel in a

linear ordering of the image elements. For instance, in the sample from part 1, pixel (3,2) has linear index 7. Also, pixel (1,4) has index 13. Julia can help us built a map between pixel indices and linear or vector indices:

```
N = reshape(1:(4*4), 4, 4)
```

This creates the pixel index to linear index for the problem above because

```
N[1,4]
N[3,2]
```

return the appropriate entry.

In your own words, explain what the `reshape` operation does.

4. Now we need to construct the matrix $\boldsymbol{W}_i$ in order to apply the edge detector that we've build.
I'm giving you the following template, that I hope you can fill in. Feel free to construct $\boldsymbol{W}_1$ and $\boldsymbol{W}_2$ any way you choose, but the following should provide some guidance.

```
function crude_edge_detector(nin,nout)
  Nx = <fill in> # build a map using reshape that takes X indices to x
  Ny = <fill in>
  W = zeros(nout^2,nin^2)
  for i=1:nin
    for j=1:nin
      xi = <fill in>
      yj = <fill in>
      W[yj, xi] = <fill in>
    end
  end
  return W
end

W1 = crude_edge_detector(32,16)
W2 = crude_edge_detector(16,8)
```

Show the non-zero entries in the first row of $W2$ as well as the corresponding indices.

5. Now write a function to evaluate the neural net output on an image that we explained above. Note, your code should not recompute the edge detectors `W1` and `W2` each time, doing so will lose 1/3 the points on this question.

```
function net(x)
  <fill in multiple lines>
end
```

To call `net`, we need convert an image into a vector. You can use the `reshape` command to do this, or in Julia, you can use the `vec` command too.

Show the results of the following commands

```
@show net(vec(Float64.(load("image1.png"))))
@show net(vec(Float64.(load("image2.png"))))
@show net(vec(Float64.(load("image3.png"))))
```

(Hint, I get `net(vec(Float64.(load("image1.png")))) = 0.08235294117647171`)
The original images can be accessed from

- https://c1.staticflickr.com/8/7015/6554001581_3370ca8802_b.jpg
- https://c1.staticflickr.com/3/2880/33053003793_5840a879fb_b.jpg

- https://c1.staticflickr.com/5/4138/4814209463_10a00d0b2d_b.jpg

Do these results make sense?

6. Now suppose we change the edge detector to use the stencil

$$\begin{bmatrix} - & + \\ + & - \end{bmatrix}$$

instead. Show the output from the same neural net architecture using the new matrices W_1 and W_2.

7. The matrices $W1$ and $W2$ have very few entries compared to the number of zeros. This is a case where we could consider using sparse matrices instead of dense matrices. One efficient way of creating a sparse matrix in Julia is to produce a list of the elements that are non-zero, and then to use the matrix. Write the following function

```
using SparseArrays
function sparse_crude_edge_detector(nin,nout)
  Nx = <fill in> # build a map using reshape that takes X indices to x
  Ny = <fill in>
  nnz = <fill in> # this is the number of non-zeros
  I = zeros(Int, nnz) # the row index
  J = zeros(Int, nnz) # the column index
  V = zeros(nnz) # the value
  index = 1
  for i=1:nin
    for j=1:nin
      I[index] = <fill in>
      J[index] = <fill in>
      V[index] = <fill in>
      index += 1
    end
  end
  return sparse(I,J,V,nout^2,nin^2)
end
```

Write this function and make sure that the result is equivalent to what you got with your original function.

```
sW1 = sparse_crude_edge_detector(32,16)
W1 = crude_edge_detector(32,16)
sW1 == W1 # test for equality
```

8. Let `x = vec(Float64.(load("image1.png")))`. Then show the time to compute `W1*x` vs. `sW1*x`. (In Julia, this is `@time`). Repeat this a few times to make sure you have the correct time. Report the fastest time for each.