Problem Set 1 CS585 - Minh Le Nguyen

Problem 1 Calculus Review

A. Shift Variance

$\mathrm{softmax}(\mathbf{z}) = \mathrm{softmax}(\mathbf{z} - C\mathbf{1})$ where $C \in \mathbb{R}$ and $\mathbf{1}$ is the all-one vector.

$softmax(z_i) = y_i = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}}$

$softmax(z - C1) = \frac{e^{z_i - C}}{\sum_{j=1}^{k} e^{z_j - C}} = \frac{\frac{e^{z_i}}{e^C}}{\sum_{j=1}^{k} \frac{e^{z_j}}{e^C}} = e^C \frac{\frac{e^{z_i}}{e^C}}{\sum_{j=1}^{k} e^{z_j}} = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}} = softmax(z)$

Thus $softmax(z)$ is invariant to constant shifting on z

B. Derivative

Let $\sum = \sum_{j=1}^{k} e^{z_j}$

$\frac{\partial y_i}{\partial z_j} = \frac{\partial \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}}}{\partial z_j} = \frac{(e^{z_i})' \sum - e^{z_i} \sum'}{\sum^2}$ (product rules for derivatives)

When $i \neq j$:

$\frac{(e^{z_i})' \sum + e^{z_i} \sum'}{\sum^2} = \frac{0 - e^{z_i} e^{z_j}}{\sum^2} = -\frac{e^{z_i}}{\sum} \frac{e^{z_j}}{\sum} = -y_i y_j$ (1)

When $i = j$:

$\frac{(e^{z_i})' \sum + e^{z_i} \sum'}{\sum^2} = \frac{e^{z_i} \sum - e^{z_i} e^{z_j}}{\sum^2} = -\frac{e^{z_i}}{\sum} \frac{\sum - e^{z_j}}{\sum} = y_i(1 - y_j)$ (2)

From (1) and (2) we can conclude that:

$$\frac{\partial y_i}{\partial z_j} = \begin{cases} -y_i y_j & \text{when } i \neq j \\ y_i(1 - y_j) & \text{when } i = j \end{cases} \tag{1}$$

C.Chain Rule

$z_j = W^T x + u$

Compute $\frac{\partial y_i}{\partial x}$:

$\frac{\partial z_j}{\partial x} = W^T$

$\frac{\partial y_i}{\partial z_j} . \frac{\partial z_j}{\partial x} = \frac{\partial y_i}{\partial x}$

$$\frac{\partial y_i}{\partial x} = \begin{cases} -y_i y_j W^T & \text{when } i \neq j \\ y_i(1 - y_j) W^T & \text{when } i = j \end{cases} \tag{2}$$

Compute $\frac{\partial y_i}{\partial w_j}$:

$\frac{\partial z_j}{\partial w_j} = x$

$\frac{\partial y_i}{\partial z_j} . \frac{\partial z_j}{\partial w_j} = \frac{\partial y_i}{\partial w_j}$

$$\frac{\partial y_i}{\partial w_j} = \begin{cases} -y_i y_j x & \text{when } i \neq j \\ y_i(1 - y_j) x & \text{when } i = j \end{cases} \tag{3}$$

Compute $\frac{\partial y_i}{\partial u}$:

$\frac{\partial z_j}{\partial u} = 1$

$\frac{\partial y_i}{\partial z_j} . \frac{\partial z_j}{\partial u} = \frac{\partial y_i}{\partial u}$

$$\frac{\partial y_i}{\partial u} = \begin{cases} -y_i y_j & \text{when } i \neq j \\ y_i(1-y_j) & \text{when } i = j \end{cases} \tag{4}$$

Problem 2 Linear Algebra Review

A. Matrix Multiplication

$V = \begin{bmatrix} -\frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix}$

$V \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} -\frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix}$

$V \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} -\frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} -\frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} \end{bmatrix}$

Vx rotates vector x counter clockwise 135 degree

B. Matrix Transpose

$V = \begin{bmatrix} -\frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix}$

$V.V^{-1} = I$ where I is the indentity matrix $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$

We want to find V^{-1}. Let $V^{-1} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$

Thus we get:

$\begin{bmatrix} -\frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} a & b \\ c & d \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$

$-\frac{1}{\sqrt{2}}a + -\frac{1}{\sqrt{2}}c = 1$

$\frac{1}{\sqrt{2}}a + -\frac{1}{\sqrt{2}}c = 0$

=> $c = -\frac{1}{\sqrt{2}}$ and $a = -\frac{1}{\sqrt{2}}$ (1)

$-\frac{1}{\sqrt{2}}b + -\frac{1}{\sqrt{2}}d = 0$

$\frac{1}{\sqrt{2}}b + -\frac{1}{\sqrt{2}}d = 1$

=> $d = -\frac{1}{\sqrt{2}}$ and $b = \frac{1}{\sqrt{2}}$ (2)

From (1) and (2):

$V^{-1} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} = \begin{bmatrix} -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix} = V^T$

$V^T x$ rotates x clockwise 135 degree

C. Diagonal Matrix

$V^T = \begin{bmatrix} -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix}$

$$\sum = \begin{bmatrix} \sqrt{8} & 0 \\ 0 & 2 \end{bmatrix}$$

$$x = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ \frac{1}{\sqrt{2}} \end{bmatrix}, \begin{bmatrix} -\frac{1}{\sqrt{2}} \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ -\frac{1}{\sqrt{2}} \end{bmatrix}$$

$$\sum V^T x_1 = \begin{bmatrix} \sqrt{8} & 0 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{2}} \\ 0 \end{bmatrix} = \begin{bmatrix} -\sqrt{2} \\ -1 \end{bmatrix}$$

$$\sum V^T x_2 = \begin{bmatrix} \sqrt{8} & 0 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} 0 \\ \frac{1}{\sqrt{2}} \end{bmatrix} = \begin{bmatrix} \sqrt{2} \\ -1 \end{bmatrix}$$

$$\sum V^T x_3 = \begin{bmatrix} \sqrt{8} & 0 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} -\frac{1}{\sqrt{2}} \\ 0 \end{bmatrix} = \begin{bmatrix} \sqrt{2} \\ 1 \end{bmatrix}$$

$$\sum V^T x_4 = \begin{bmatrix} \sqrt{8} & 0 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} 0 \\ -\frac{1}{\sqrt{2}} \end{bmatrix} = \begin{bmatrix} -\sqrt{2} \\ 1 \end{bmatrix}$$

It is a retangle, with height $= 2$, width $= 2\sqrt{2}$

D. Matrix Multiplication II

$$U = \begin{bmatrix} -\frac{\sqrt{3}}{2} & \frac{1}{2} \\ -\frac{1}{2} & -\frac{\sqrt{3}}{2} \end{bmatrix}$$

Ux rotates x clockwise with $\theta = 150$ degree

E. Geometric Interpretation

$$A = U\Sigma V^T = \begin{bmatrix} -\frac{\sqrt{3}}{2} & \frac{1}{2} \\ -\frac{1}{2} & -\frac{\sqrt{3}}{2} \end{bmatrix} \begin{bmatrix} \sqrt{8} & 0 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix} = \begin{bmatrix} \frac{2\sqrt{3}-\sqrt{2}}{2} & -\sqrt{3} - \frac{1}{\sqrt{2}} \\ 1 + \frac{\sqrt{3}}{\sqrt{2}} & -1 + \frac{\sqrt{3}}{\sqrt{2}} \end{bmatrix}$$

In general, the order of transformation operations (scaling, rotations, shifting) do not commute, i.e: $ABCx \neq CABx$

Thus, to get the similar geometric interpretation for Bx, where B is a general square matrix, we can make $B = A$.

However, in a 2 dimensional rotation and scaling, that is the rotation only rotate around 1 axis, the rotates can be added or subtracted.

Thus: V^T rotates x clockwise 135 degree

U rotates x clockwise 150 degree

Thus V^T and U rotates x 285 degree clockwise, or 75 degree counter clockwise.

Therefore, B can be the product of $\sum$ and $\begin{bmatrix} cos(75) & -sin(75) \\ sin(75) & cos(75) \end{bmatrix}$

Problem 3 Un-shredding Image

3.1

In [297…

```
from PIL import Image
import os, glob
import numpy as np
from IPython.display import display

def load_images_from_folder(folder):
    # first step is to get the list of all the files in the folder
    image_filenames = glob.glob(os.path.join(folder, '*.png'))
    # Now you should load each image into memory as a
    # numpy array
    images = []
    # ADD CODE HERE
```

```python
    image_filenames.remove('shredded-image\\simple_larry-roberts.png')
    for file in image_filenames:
        images.append(np.array(Image.open(file)))
    return images

images = load_images_from_folder('shredded-image')
simple_combined = Image.fromarray(np.hstack(images), 'RGB')
display(simple_combined)
```

3.2

In [284…

```python
# We'll begin by computing similarities between all image pairs
similarities = np.zeros((len(images), len(images)))
for i, ith_image in enumerate(images):
  for j, jth_image in enumerate(images):
    # Now we'll compute similarity by taking the right-most
    # column of the ith image, and the left-most column of the
    # jth image
    # ADD CODE HERE

    similarities[i, j] = np.sum((ith_image[:, ith_image.shape[1]-1].astype(np.int64) - jth_image[:,0].astype(np.int64))**2)
    #similarities[i,j] = 0
```

In [296…

```python
def greedy_merge(strips, compatibility):
    # ok, we want to merge images in order of compatibility
    # so we can begin by flattening the compatibility
    # array and then using argsort to get the index
    # of the most compatibile strips

    ## ADD CODE HERE
    ordering = np.argsort(np.reshape(compatibility, -1))

    # Now that we have our ordering, we need to keep track of
    # strips so we only select them once.  Let's keep track of
    # them in the "used_strips" variable
    used_strips = set()

    # OK, now we should iterate through our ordering and add
    # the most compatible strips until we have a single image
    merged_strips = [] # final image
    merged_left = None # left-most merged strip index
    merged_right = None # right-most merged strip index

    # we'll keep this going until all strips are used

    while len(used_strips) != len(strips):
    # we should always add at least one strip, so let's make sure
        num_used_start = len(used_strips)
```

```
        for next_item in ordering:
            # first we get its row and column index
            left_strip = next_item // len(strips) #get i
            right_strip = next_item % len(strips) #get j
            # skip if they're the same strip
            if left_strip == right_strip:
                continue

            # base case, no merged strips yet
            if merged_left is None:
                merged_strips = np.hstack((strips[left_strip], strips[right_strip]))
                merged_right = right_strip
                merged_left = left_strip
                used_strips.add(right_strip)
                used_strips.add(left_strip)
                continue

            # Check if you can add this to the left of merged_strips and merge it if
            # so. If you merge, you should update merged_left, used_strips,
            # then break out of the loop.

            if left_strip not in used_strips:
              # ADD CODE HERE
                if right_strip == merged_left:
                    merged_strips = np.hstack(( strips[left_strip],merged_strips))
                    merged_left = left_strip
                    used_strips.add(left_strip)
                    break

            # Check if you can add this to the left of merged_strips and merge it if
            # so. If you merge, you should update merged_right, used_strips,
            # then break out of the loop.
            if right_strip not in used_strips:
              # ADD CODE HERE
                if left_strip == merged_left:
                    merged_strips = np.hstack((merged_strips, strips[right_strip]))
                    used_strips.add(right_strip)
                    merged_right = right_strip
                    break

        assert num_used_start != len(used_strips)
    return merged_strips

ssd_images = greedy_merge(images, similarities)
ssd_combined = Image.fromarray(ssd_images, 'RGB')
display(ssd_combined)
```

3.3

In [295…

```
# We'll begin by computing similarities between all image pairs
similarities = np.zeros((len(images), len(images)))
```

```python
for i, ith_image in enumerate(images):
  for j, jth_image in enumerate(images):
    # Now we'll compute similarity by taking the right-most
    # column of the ith image, and the left-most column of the
    # jth image
    # ADD CODE HERE

    similarities[i, j] = np.sum((ith_image[:, ith_image.shape[1]-1].astype(np.int64) - jth_image[:,0].astype(np.int64))**2)
    #similarities[i,j] = 0
def greedy_merge(strips, compatibility):
    # ok, we want to merge images in order of compatibility
    # so we can begin by flattening the compatibility
    # array and then using argsort to get the index
    # of the most compatibile strips

    ## ADD CODE HERE
    ordering = np.argsort(np.reshape(compatibility, -1))

    # Now that we have our ordering, we need to keep track of
    # strips so we only select them once.  Let's keep track of
    # them in the "used_strips" variable
    used_strips = set()

    # OK, now we should iterate through our ordering and add
    # the most compatible strips until we have a single image
    merged_strips = [] # final image
    merged_left = None # left-most merged strip index
    merged_right = None # right-most merged strip index

    # we'll keep this going until all strips are used

    while len(used_strips) != len(strips):
    # we should always add at least one strip, so let's make sure
        num_used_start = len(used_strips)

        for next_item in ordering:
            # first we get its row and column index
            left_strip = next_item // len(strips) #get i
            right_strip = next_item % len(strips) #get j
            # skip if they're the same strip
            if left_strip == right_strip:
                continue

            # base case, no merged strips yet
            if merged_left is None:
                merged_strips = np.hstack((strips[left_strip], strips[right_strip]))
                merged_right = right_strip
                merged_left = left_strip
                used_strips.add(right_strip)
                used_strips.add(left_strip)
                continue

            # Check if you can add this to the left of merged_strips and merge it if
            # so. If you merge, you should update merged_left, used_strips,
            # then break out of the loop.

            if left_strip not in used_strips:
              # ADD CODE HERE
                if right_strip == merged_left:
                    merged_strips = np.hstack(( strips[left_strip],merged_strips))
                    merged_left = left_strip
                    used_strips.add(left_strip)
                    break

            # Check if you can add this to the left of merged_strips and merge it if
            # so. If you merge, you should update merged_right, used_strips,
            # then break out of the loop.
            if right_strip not in used_strips:
              # ADD CODE HERE
                if left_strip == merged_left:
                    merged_strips = np.hstack((merged_strips, strips[right_strip]))
                    used_strips.add(right_strip)
                    merged_right = right_strip
                    break

        assert num_used_start != len(used_strips)
    return merged_strips
```

```python
ssd_images = greedy_merge(images, similarities)
ssd_combined = Image.fromarray(ssd_images, 'RGB')
display(ssd_combined)
```

In [ ]:

```python

```

In [ ]:

```python

```