ECE549 / CS543 Computer Vision: Assignment 0

Nickel Liang
zuodong2@illinois.edu
February 2021

1. **Calculus Review [10 pts].**

The $\mathrm{softmax}$ function is a commonly-used operator which turns a vector into a valid probability distribution, i.e. non-negative and sums to 1.

For vector $\mathbf{z} = (z_1, z_2, \ldots, z_k) \in \mathbb{R}^k$, the output $\mathbf{y} = \mathrm{softmax}(\mathbf{z}) \in \mathbb{R}^k$, and its i-th element is defined as

$$y_i = \mathrm{softmax}(\mathbf{z})_i = \frac{\exp(z_i)}{\sum_{j=1}^{k} \exp(z_j)} \tag{1}$$

Answer the following questions about the $\mathrm{softmax}$ function. Show the calculation steps (as applicable) to get full credit.

(a) **Shift Invariance [3 pts]** Verify that $\mathrm{softmax}(\mathbf{z})$ is invariant to constant shifting on $\mathbf{z}$, i.e. $\mathrm{softmax}(\mathbf{z}) = \mathrm{softmax}(\mathbf{z} - C\mathbf{1})$ where $C \in \mathbb{R}$ and $\mathbf{1}$ is the all-one vector. The $\mathrm{softmax}(\mathbf{z} - \max_j z_j)$ trick is used in deep learning packages to avoid numerical overflow.

For the i-th element in $\mathrm{softmax}$ function, we have

$$\mathrm{softmax}(\mathbf{z} - C\mathbf{1})_i = \frac{e^{z_i - C}}{\sum_{j=1}^{k} e^{z_j - C}} = \frac{e^{z_i} e^{-C}}{\sum_{j=1}^{k} e^{z_j} e^{-C}} = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}} = \mathrm{softmax}(\mathbf{z})_i \tag{2}$$

Hence $\mathrm{softmax}(\mathbf{z})$ is invariant to constant shifting on $\mathbf{z}$.

(b) **Derivative [3 pts]** Let $y_i = \mathrm{softmax}(\mathbf{z})_i, 1 \le i \le k$. Compute the derivative $\frac{\partial y_i}{\partial z_j}$ for any i, j. Your result should be as simple as possible, and may contain elements of $\mathbf{y}$ and/or $\mathbf{z}$.

We can apply the quotient rule of derivatives to calculate this partial derivative. When $i = j$, we have

$$\frac{\partial y_i}{\partial z_j} = \frac{e^{z_i} \sum_{j=1}^{k} e^{z_j} - e^{z_j} e^{z_i}}{(\sum_{j=1}^{k} e^{z_j})^2} = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}} \frac{\sum_{j=1}^{k} e^{z_j} - e^{z_j}}{\sum_{j=1}^{k} e^{z_j}} = y_i(1 - y_j) \tag{3}$$

When $i \neq j$, we have

$$\frac{\partial y_i}{\partial z_j} = \frac{0 - e^{z_j} e^{z_i}}{(\sum_{j=1}^{k} e^{z_j})^2} = -\frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}} \frac{e^{z_j}}{\sum_{j=1}^{k} e^{z_j}} = -y_i y_j = y_i(0 - y_j) \tag{4}$$

We can combine these two conditions with a Kronecker delta function δ_{ij}, and the final form would be

$$\frac{\partial y_i}{\partial z_j} = y_i(\delta_{ij} - y_j) \tag{5}$$

(c) **Chain Rule [4 pts]** Consider $\mathbf{z}$ to be the output of a linear transformation $\mathbf{z} = W^\top \mathbf{x} + \mathbf{u}$, where vector $\mathbf{x} \in \mathbb{R}^d$, matrix $W \in \mathbb{R}^{d \times k}$, and vector $\mathbf{u} \in \mathbb{R}^k$. Denote $\{\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_k\}$ as the columns of W. Let $\mathbf{y} = \mathrm{softmax}(\mathbf{z})$. Compute $\frac{\partial y_i}{\partial \mathbf{x}}$, $\frac{\partial y_i}{\partial \mathbf{w}_j}$, and $\frac{\partial y_i}{\partial \mathbf{u}}$. (Hint: You may reuse (1.2) and apply the chain rule. Vector derivatives: $\frac{\partial(\mathbf{a}\cdot\mathbf{b})}{\partial \mathbf{a}} = \mathbf{b}$, $\frac{\partial(\mathbf{a}\cdot\mathbf{b})}{\partial \mathbf{b}} = \mathbf{a}$.)

For the answer of this question, I use denominator layout to show the result. If numerator layout is preferred, apply an additional transpose to the result.

We have

$$y_i = \frac{e^{\mathbf{z_i}}}{\sum_{j=1}^{k} e^{\mathbf{z_j}}} = \frac{e^{\mathbf{w_i}^\top \mathbf{x} + u_i}}{\sum_{j=1}^{k} e^{\mathbf{w_j}^\top \mathbf{x} + u_j}} \tag{6}$$

and

$$\frac{\partial \mathbf{w_j}^\top \mathbf{x}}{\partial \mathbf{x}} = \frac{\partial \mathbf{x}^\top \mathbf{w_j}}{\partial \mathbf{x}} = \mathbf{w_j} \tag{7}$$

Hence for $\frac{\partial y_i}{\partial \mathbf{x}}$ we have

$$\frac{\partial y_i}{\partial \mathbf{x}} = \frac{e^{\mathbf{z_i}}\mathbf{w_i}\sum_{j=1}^{k} e^{\mathbf{z_j}} - \sum_{j=1}^{k} e^{\mathbf{z_j}}\mathbf{w_j}e^{\mathbf{z_i}}}{(\sum_{j=1}^{k} e^{\mathbf{z_j}})^2} = \frac{e^{\mathbf{z_i}}}{\sum_{j=1}^{k} e^{\mathbf{z_j}}}(\mathbf{w_i} - \mathbf{w_j}) \tag{8}$$

$$= y_i(\mathbf{w_i} - \mathbf{w_j}) \tag{9}$$

For $\frac{\partial y_i}{\partial \mathbf{w}_j}$, we need to consider two conditions. When $i = j$, we have

$$\frac{\partial y_i}{\partial \mathbf{w_j}} = \frac{(e^{\mathbf{z_i}}\mathbf{x})\sum_{j=1}^{k} e^{\mathbf{z_j}} - (\sum_{j=1}^{k} e^{\mathbf{z_j}}\mathbf{x})e^{\mathbf{z_i}}}{(\sum_{j=1}^{k} e^{\mathbf{z_j}})^2} = \frac{e^{\mathbf{z_i}}}{\sum_{j=1}^{k} e^{\mathbf{z_j}}}(1 \cdot \mathbf{x} - \mathbf{x}) \tag{10}$$

$$= y_i(1 \cdot \mathbf{x} - \mathbf{x}) \tag{11}$$

When $i \neq j$, we have

$$\frac{\partial y_i}{\partial \mathbf{w_j}} = \frac{0 - (\sum_{j=1}^{k} e^{\mathbf{z_j}}\mathbf{x})e^{\mathbf{z_i}}}{(\sum_{j=1}^{k} e^{\mathbf{z_j}})^2} = \frac{e^{\mathbf{z_i}}}{\sum_{j=1}^{k} e^{\mathbf{z_j}}}(0 \cdot \mathbf{x} - \mathbf{x}) \tag{12}$$

$$= y_i(0 \cdot \mathbf{x} - \mathbf{x}) \tag{13}$$

Using Kronecker delta function δ_{ij} to further simplify these two conditions, we have

$$\frac{\partial y_i}{\partial \mathbf{w_j}} = y_i(\delta_{ij} - 1)\mathbf{x} \tag{14}$$

For $\frac{\partial y_i}{\partial \mathbf{u}}$, we first obtain $\frac{\partial u_i}{\partial \mathbf{u}}$

$$\frac{\partial u_i}{\partial \mathbf{u}} = \mathbf{e_i} \tag{15}$$

where $\mathbf{e_i}$ is a standard basis column vector, whose i-th term is 1 while all other terms are 0. Hence we have

$$\frac{\partial y_i}{\partial \mathbf{u}} = \frac{(e^{\mathbf{z_i}}\mathbf{e_i})\sum_{j=1}^{k} e^{\mathbf{z_j}} - (\sum_{j=1}^{k} e^{\mathbf{z_j}}\mathbf{e_j})\mathbf{e^{z_j}}}{(\sum_{j=1}^{k} e^{\mathbf{z_j}})^2} = \frac{e^{\mathbf{e_i}}}{\sum_{j=1}^{k} e^{\mathbf{z_j}}}(\mathbf{e_i} - \mathbf{e_j}) \tag{16}$$

$$= y_i(\mathbf{e_i} - \mathbf{e_j}) \tag{17}$$

2. **Linear Algebra Review [10 pts].** Answer the following questions about matrices. Show the calculation steps (as applicable) to get full credit.

   (a) **Matrix Multiplication[2 pts]** Let $V = \begin{bmatrix} -\frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix}$. Compute $V \begin{bmatrix} 1 \\ 0 \end{bmatrix}$, $V \begin{bmatrix} 0 \\ 1 \end{bmatrix}$. What does matrix multiplication Vx do to x?

$$V \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix}, V \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} -\frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} \end{bmatrix} \tag{18}$$

   This matrix multiplication will rotate vector x 135 degrees counter clockwise.

   (b) **Matrix Transpose[2 pts]** Verify that $V^{-1} = V^\top$. What does $V^\top x$ do?

$$V^{-1} = \frac{1}{\frac{1}{2} + \frac{1}{2}} \begin{bmatrix} -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix} = \begin{bmatrix} -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix} \tag{19}$$

$$V^\top = \begin{bmatrix} -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix} \tag{20}$$

   Hence we have $V^{-1} = V^\top$. $V^\top x$ will rotate vector x 135 degrees clockwise.

   (c) **Diagonal Matrix[2 pts]** Let $\Sigma = \begin{bmatrix} \sqrt{8} & 0 \\ 0 & 2 \end{bmatrix}$. Compute $\Sigma V^\top x$ where $x = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ \frac{1}{\sqrt{2}} \end{bmatrix}, \begin{bmatrix} -\frac{1}{\sqrt{2}} \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ -\frac{1}{\sqrt{2}} \end{bmatrix}$ respectively. These are 4 corners of a square. What is the shape of the result points?

   We first calculate $\Sigma V^\top$

$$\Sigma V^\top = \begin{bmatrix} -2 & 2 \\ -\sqrt{2} & -\sqrt{2} \end{bmatrix} \tag{21}$$

$$\Sigma V^\top \begin{bmatrix} \frac{1}{\sqrt{2}} \\ 0 \end{bmatrix} = \begin{bmatrix} -\sqrt{2} \\ -1 \end{bmatrix} \tag{22}$$

$$\Sigma V^\top \begin{bmatrix} 0 \\ \frac{1}{\sqrt{2}} \end{bmatrix} = \begin{bmatrix} \sqrt{2} \\ -1 \end{bmatrix} \tag{23}$$

$$\Sigma V^\top \begin{bmatrix} -\frac{1}{\sqrt{2}} \\ 0 \end{bmatrix} = \begin{bmatrix} \sqrt{2} \\ 1 \end{bmatrix} \tag{24}$$

$$\Sigma V^\top \begin{bmatrix} 0 \\ -\frac{1}{\sqrt{2}} \end{bmatrix} = \begin{bmatrix} -\sqrt{2} \\ 1 \end{bmatrix} \tag{25}$$

   The resulting points is a rectangle.

   (d) **Matrix Multiplication II [2 pts]** Let $U = \begin{bmatrix} -\frac{\sqrt{3}}{2} & \frac{1}{2} \\ -\frac{1}{2} & -\frac{\sqrt{3}}{2} \end{bmatrix}$. What does Ux do? (Rotation matrix wiki)

   Ux will rotate vector x 120 degrees clockwise.

(e) **Geometric Interpretation[2 pts]** Compute $A = U\Sigma V^T$. From the above questions, we can see a geometric interpretation of Ax: (1) V^T first rotates point x, (2) Σ rescales it along the coordinate axes, (3) then U rotates it again. Now consider a general squared matrix $B \in \mathbb{R}^{n\times n}$. How would you obtain a similar geometric interpretation for Bx?

As computed in part c, we have

$$\Sigma V^\top = \begin{bmatrix} -2 & 2 \\ -\sqrt{2} & -\sqrt{2} \end{bmatrix} \tag{26}$$

$$\tag{27}$$

Hence

$$A = U\Sigma V^\top = \begin{bmatrix} \sqrt{3} - \frac{\sqrt{2}}{2} & -\sqrt{3} - \frac{\sqrt{2}}{2} \\ 1 + \frac{\sqrt{6}}{2} & -1 + \frac{\sqrt{6}}{2} \end{bmatrix} \tag{28}$$

For a general squared matrix B, we can perform singular value decomposition (SVD) on it and extract matrix U, Σ, and $V^\top$. Matrix $V^\top$ will first rotate matrix x, then Σ will rescale x, then U will rotate x again.

3. **Un-shredding Images [30 pts]** We accidentally shredded some images! In this problem, we will recover the original images from the corresponding shreds. Along the way we will learn how to deal with and process images in Python. Example input / output is shown below.

   (a) **Combine [5 pts]** We will start simple, and work with images where the shredder simply divided the image into vertical strips. These are prefixed with $simple\_$ in the $shredded-images$ folder. Each folder contains the shreds (as individual files) for that particular image.

   Our first task is to take these strips and concatenate them (in any order) to produce a complete image. Without proper ordering of the strips, this image won't quite look correct, but that's ok for now. We will tackle that in the next part. For now, save this weird composite image. Include the generated image into your report.

(b) **Re-order [10 pts]** As you noticed, without correctly ordering the strips, the images didn't quite look correct. Our goal now is to identify the correct ordering for these strips. We will adopt a very simple greedy algorithm. For each pair of strips, we will compute how well does strip 1 go immediately left of strip 2. We will do this my measuring the similarity between the last column of pixels in strip 1 and the first column of pixels in strip 2. You can measure similarity using (negative of) the *sum of squared differences* which is simply the L2 norm of the pixel difference.

Once you have computed the similarity between all pairs of strips, you can follow a greedy algorithm to composite the image together. Start with a strip and place the strip that is most compatible with it on the left and the right side, and so on. Compose the strips together into an image using this simple greedy algorithm.

Save the generated composite images for all the $simple\_$ shreds from the $shredded - images$ folder, and include them in your report. Also include a brief description of your implemented solution, focusing especially on the more non-trivial or interesting parts of the solution.

I implemented this part using a simple greedy algorithm. I will go over the images array, take two shreds, and compute the similarities between the first and last column of each shred by calculating the L2 norm of the difference. The smaller L2 norm is, the more similar two shreds are, and hence they are more likely to sit next to each other. In each iteration, I pick the global minimum L2 norm and merge the corresponding

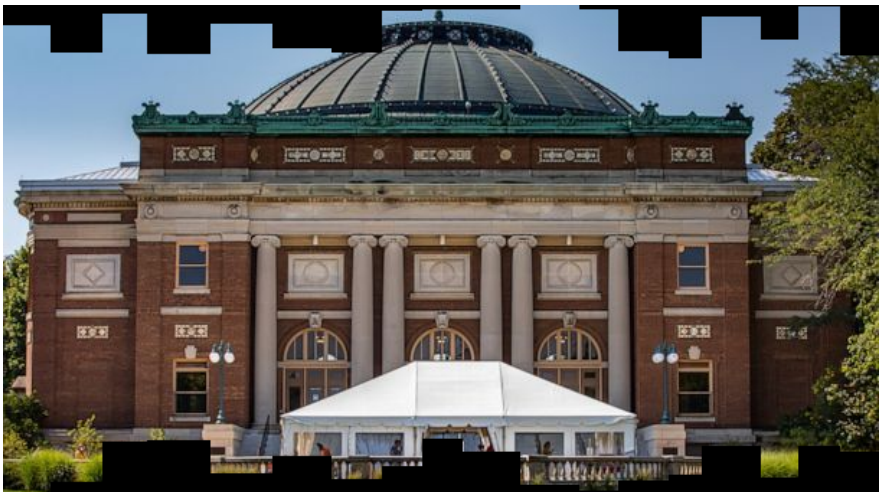
images. Then I remove the original shreds from the images array and put the merged image back to the images array. I will get a merged image when the images array have only one element left.

(c) **Align and Re-order [15 pts]** Next, we will tackle the case where our shredder also cut out the edges of some of the strips, so that they are no longer aligned vertically. These are prefixed with $hard\_$. We will now modify the strip similarity function from the previous part to compensate for this vertical misalignment.

We will vertically slide one strip relative to the other, in the range of 20% of the vertical dimension to find the best alignment, and compute similarity between the overlapping parts for different slide amounts. We will use the maximum similarity over these different slide amounts, as the similarity between the two strips. Note that you may need to experiment with similarity functions other than the sum of squared distances. One common alternative is zero mean normalized cross correlation, which is simply the dot product between the two pixel vectors after they have been normalized to have zero mean and unit norm.

Implement this strategy for measuring the similarity between the strips. Use this along with the greedy re-ordering strategy in the previous part, to reconstruct this harder set of shreds. You will also have to keep track of the slide amounts that led to the best match. You will need those when you stitch the shreds together.

Save the generated composite images for all the $hard\_$ shreds, and include them in your report. As before, include a brief description of your implemented solution, focusing especially on the more non-trivial or interesting parts of the solution. What design choices did you make, and how did they affect the quality of the result and the speed of computation? What are some artifacts and/or limitations of your implementation, and what are possible reasons for them?



[redacted]IN[redacted]
[redacted]JECT MRO[redacted]

Artificial Intelligence Group J
Vision Memo. No. 100.

1966 uly 7,

THE SUMR VISIMET ON PROJEC

Seour Paym pert

The summer vision project an atisusetempt to rs effectively in the constructionf a si o pagnificantummer worke our stem. The particular task was chosen rtly bpa caecause ita visual syrt of to sub-problems which will allow iividuandk ils to woregmented inn be syet pa[redacted]ate i[redacted]he constru[redacted]ton[redacted]sy o[redacted]stem compdently and ndepen[redacted]l [redacted]h to be a r enoug[redacted]

I implemented this section in a similar way as part 2 of this question. There are two major steps: the first step is to find the two most similar shreds and their offset, and the second step is to merge them correctly with top and bottom black space. In the first step, I still go over all possible combinations in image array. For each combination, I calculate the maximum Zero Mean Normalized Cross-Correlation among all possible offsets. Then, I pick two shreds that has the global maximum ZMNCC and proceed towards the second step. In the second step, I take two shreds and their relative offset value, and generate a combined image that has extra space filled with minus infinity, which helps me to identify which section is filled black and which section is valid data in a combined image. Then I remove two original shreds, and put this newly generated shreds back to the image array. The align and reorder process stops when there is only one image left in the image array.

There are a few drawbacks of my implementation. The first drawback is the run time. Since I always pick the global maximum ZMNCC value and I merge images when I find the maximum, I always need to re-calculate the ZMNCC value of the whole image array in each iteration. A more efficient implementation would be calculate and store all ZMNCC value, pick the maximum pair, record the order, then merge all images at the end.

Another drawback is that I failed to combine the text image correctly. This is largely due to the lack of feature between text shreds. Since the majority pixels in these shreds are close to white color, my ZMNCC implementation failed to find the most similar shreds sometimes. I tried a few different ZMNCC implementation, including calculate ZMNCC separately among all three color channels then combine them together, calculate ZMNCC of all three color channels together, and convert columns to gray scale before calculating ZMNCC value. I also tried different slide value between 10% to 20% of the taller image height. I think a possible fix for this issue is to use more columns on image edges to calculate the similarities between two shreds. Another possible fix is to use other similarity functions.