CSE8803/CX4803 - Homework 2 - Spring 2022

Please submit your answers to Questions 1-3 in one PDF file, and everything that is required to submit for Question 3 in a .zip file.

1 CNN basics [10 pts]

We have covered several parameters of convolution layers in the lecture, e.g., the number of filters and the size of filters. In this problem, you will see another two important convolution layers, i.e., *padding* and *stride*.

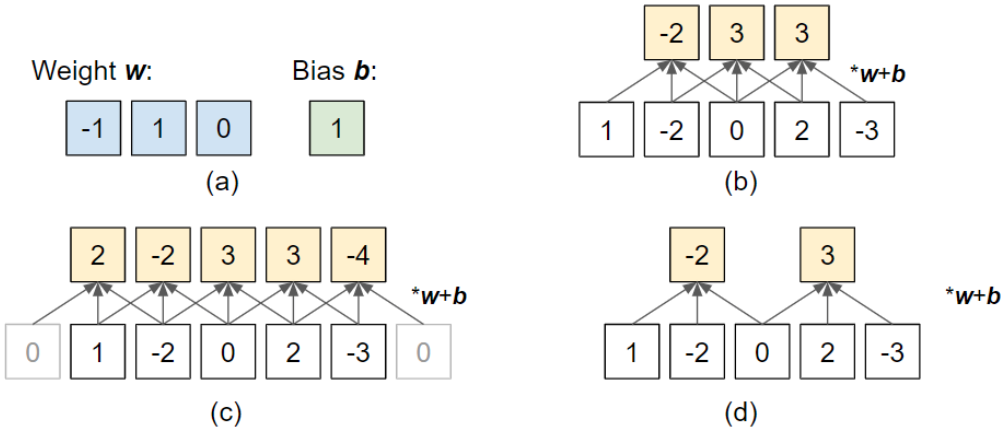

Figure 1: 1D CNN.

Let us first consider 1D convolution as an example. Assume we have an 1-channel filter with a weight $\mathbf{w} = [-1, 1, 0]$ and a bias $b = 1$ and an 1D input $\mathbf{x} = [1, -2, 0, 2, -3]$ (Fig. 1(a)). It is easy to show that using the convolution operation introduced in the lecture, we will have an output $\mathbf{y} = [-2, 3, 3]$ (Fig. 1(b)).

Padding is often used to control the size of output. For example, it is common that we want the output of a convolution layer to have the same dimension as the input. In our above example, we can pad a '0' on each side of the input and then apply the convolution operation (Fig. 1(c)). The output will be $\mathbf{y} = [2, -2, 3, 3, -4]$, with a same length as the input.

Stride is another parameter used to control the step size when sliding the filter over the input. The default value of stride is 1. We can increase the value, say, to 2, which means in every step we will shift the filter by 2 positions. See Fig. 1(d) for an example.

Now, generalize the above ideas to 2D cases. Given the following filter (weight $\mathbf{w}$ and a bias $b = 1$) and input ($\mathbf{x}$),

$$\text{Filter: } \mathbf{w} = \begin{pmatrix} 1 & 0 & -1 \\ 2 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \quad b = 1$$

$$\text{Input: } \mathbf{x} = \begin{pmatrix} 3 & 2 & 0 & 4 & 3 \\ 0 & 2 & 3 & 4 & 1 \\ 1 & 1 & 4 & 2 & 2 \\ 1 & 2 & 3 & 1 & 4 \\ 0 & 4 & 3 & 3 & 0 \end{pmatrix},$$

we now consider different convolutions:

(1) [4 pts] Compute the standard convolution result $\mathbf{y}$ (i.e., without padding and with a stride size of 1).

(2) [4 pts] To produce a convolution output with the same size as the input, we can apply a padding to $\mathbf{x}$. It is easy to see by a extra column or row of 0's on each side of matrix $\mathbf{x}$, we can obtain a same-size output $\mathbf{y}$. Please compute the result $\mathbf{y}$ after applying the convolution with padding.

(3) [2 pts] Compute the convolution result after applying a convolution with stride size of 2.

Solution:

(1)

$$\mathbf{y} = \begin{pmatrix} 10 & 8 & 10 \\ 4 & 6 & 17 \\ 5 & 10 & 10 \end{pmatrix}$$

(2)

$$\mathbf{y} = \begin{pmatrix} 6 & 12 & 9 & 6 & 12 \\ 0 & 10 & 8 & 10 & 14 \\ 2 & 4 & 6 & 17 & 11 \\ 5 & 5 & 10 & 10 & 9 \\ -1 & 3 & 13 & 9 & 8 \end{pmatrix}$$

(3)

$$\mathbf{y} = \begin{pmatrix} 10 & 10 \\ 5 & 10 \end{pmatrix}$$

2 Theory: conditional log-likelihood and Mean Squared Error [10 pts]

You have seen in our lectures that very often the Mean Squared Error (MSE) is used as an objective function to minimize when training a model. In this exercise, you will see that there is a connection between the MSE criterion and the maximum likelihood criterion through a linear regression model.

Recall that with a linear regression model, we aim to find parameters $\mathbf{w}$, such that for a vector $\mathbf{x} \in \mathbf{R}^n$, the output $\hat{y} = \mathbf{w}^T\mathbf{x}$ is a good prediction for y.

One common objective function used to fit this model is to find $\mathbf{w}$ such that the MSE between y and $\hat{y}$ is minimized. That is, we want to perform

$$\min \frac{1}{m} \sum_{i=1}^{m} \|y_i - \hat{y}_i\|^2 \tag{1}$$

Where m is the number of datapoints (samples). Now let's look at the regression problem from a probabilistic point of view, and we now want to learn a probability distribution such that the probability of generating y from $\mathbf{x}$ is maximized. We define the conditional probability distribution $p(y|\mathbf{x})$ to be a Gaussian distribution with predicted mean as $\hat{y}$ and variance σ^2, that is

$$p(y|\mathbf{x}) = \mathcal{N}(y; \hat{y}, \sigma^2)$$

We want to maximize the conditional log likelihood, that is

$$\max \sum_{i=1}^{m} \log p(y_i|\mathbf{x}_i; \boldsymbol{\theta}) \tag{2}$$

Show that Equations 1 and 2 lead to the same location of the optimum given that σ is a constant.

Solution:
Take Gaussian distribution into the log likelihood function:

$$\begin{aligned}
&\sum_{i=1}^{m} \log p(\mathbf{y}_i|\mathbf{x}_i, \theta) \\
=&\sum_{i=1}^{m} \log \mathcal{N}(\mathbf{y}_i; \mathbf{w}^T\mathbf{x}_i, \sigma^2) \\
=&-c - \sum_{i=1}^{m} \frac{(y_i - \mathbf{w}^T\mathbf{x}_i)^T(y_i - \mathbf{w}^T\mathbf{x}_i)}{2\sigma^2}
\end{aligned} \tag{3}$$

(c is a constant irrelevant to parameter $\mathbf{w}$). And maximizing the log likelihood function above equals to minimizing MSE:

$$\begin{aligned}
&\min_{\mathbf{w}} \sum_{i=1}^{m} \frac{(y_i - \mathbf{w}^T\mathbf{x}_i)^T(y_i - \mathbf{w}^T\mathbf{x}_i)}{2\sigma^2} \\
=&\min_{\mathbf{w}} \frac{1}{m} \sum_{i=1}^{m} \|y_i - \mathbf{w}^T\mathbf{x}_i\|_2^2
\end{aligned} \tag{4}$$

3 Theory: Principal Component Analysis [15 pts]

In the lecture, we mentioned that PCA found a set of orthogonal principal components that maximize the variance preserved in the latent embedding. The principal components can also be derived with a different objective which is to minimize the mean square error(MSE) between the input data and the data reconstructed from the principle components. Denoting the input high dimensional data $\mathbf{X}$ by $\{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_N\} \in \mathbf{R}^d$, the first principle component by $\mathbf{u}$, and the projection of sample $\mathbf{x}_i$ along $\mathbf{u}$ by $\mathbf{x}'_i$, we aim to find the $\mathbf{u}$ that minimizes the MSE between the $\mathbf{x}_i$ and $\mathbf{x}'_i$. That is, we need to solve:

$$\arg\min_{\mathbf{u}} \frac{1}{N} \sum_{i=1}^{N} \|\mathbf{x}'_i - \mathbf{x}_i\|^2 \tag{5}$$

where $\mathbf{u}^T\mathbf{u} = 1$.

Show that the solution for $\mathbf{u}$ in Eq. 5 is the same as the solution in the following objective function:

$$\arg\max_{\mathbf{u}} \mathbf{u}^T\mathbf{\Sigma}\mathbf{u} \tag{6}$$

where $\mathbf{u}^T\mathbf{u} = 1$ and $\mathbf{\Sigma}$ is the covariance matrix of $\mathbf{X}$. Note that $\mathbf{x}' = (\mathbf{u}^T\mathbf{x}_i)\mathbf{u}$ and $\|\mathbf{y}\|^2 = \mathbf{y}^T\mathbf{y}$ for a vector $\mathbf{y}$.

Solution:

$$\begin{aligned}
&\frac{1}{N} \sum_{i=1}^{N} \|\mathbf{x}'_i - \mathbf{x}_i\|^2 \\
=&\frac{1}{N} \sum_{i=1}^{N} (\mathbf{x}'_i - \mathbf{x}_i)^T(\mathbf{x}'_i - \mathbf{x}_i) \\
=&\frac{1}{N} \sum_{i=1}^{N} (\mathbf{x}_i'^T\mathbf{x}'_i - \mathbf{x}_i'^T\mathbf{x}_i - \mathbf{x}_i^T\mathbf{x}'_i + \mathbf{x}_i^T\mathbf{x}_i) \\
=&\frac{1}{N} \sum_{i=1}^{N} (\|\mathbf{x}_i\|^2 + \mathbf{x}_i'^T\mathbf{x}'_i - \mathbf{x}_i'^T\mathbf{x}_i - \mathbf{x}_i^T\mathbf{x}'_i)
\end{aligned} \tag{7}$$

Apply $\mathbf{x}' = (\mathbf{u}^T\mathbf{x}_i)\mathbf{u}$ and also note that the transpose of a column vector times a column vector gives a scalar value which is the dot product of the two vectors (for example: $\mathbf{u}^T\mathbf{x}_i$), the equation above can be further transformed into (intermediate steps skipped):

$$\begin{aligned}
&\frac{1}{N}\sum_{i=1}^{N}(\|\mathbf{x}_i\|^2 - (\mathbf{u}^T\mathbf{x}_i)(\mathbf{x}_i^T\mathbf{u})) \\
=&\frac{1}{N}\sum_{i=1}^{N}\|\mathbf{x}_i\|^2 - \frac{1}{N}\sum_{i=1}^{N}(\mathbf{u}^T\mathbf{x}_i)(\mathbf{x}_i^T\mathbf{u}) \\
=&\frac{1}{N}\sum_{i=1}^{N}\|\mathbf{x}_i\|^2 - \mathbf{u}^T\frac{1}{N}(\sum_{i=1}^{N}\mathbf{x}_i\mathbf{x}_i^T)\mathbf{u} \\
=&\frac{1}{N}\sum_{i=1}^{N}\|\mathbf{x}_i\|^2 - \mathbf{u}^T\mathbf{\Sigma}\mathbf{u}
\end{aligned} \tag{8}$$

The first term in Eq. 8 does not depend on $\mathbf{u}$, so minimizing Eq. 8 is equivalent to maximizing the 2nd term in Eq. 8, which is the same objective as in Eq. 6.

4 Programming: Dimensionality reduction [15 pts]

Dimensionality reduction algorithms are important when we want to visualize the structure of high-dimensional data. In our lecture, we talked about one linear dimensionality reduction algorithm: "PCA", and two non-linear dimensionality reduction algorithms: "TSNE" and "UMAP". In this programming assignment, we will try to implement those algorithms on a real world dataset. We will use a human peripheral blood mononuclear cells(PBMCs) dataset [1], which is generated using single-cell RNA Sequencing technology.

Single-cell RNA-Sequencing technology is a recently developed technology that is able to measure the mRNA abundance of thousands of genes in millions of individual cells, and the data generated from single-cell RNA-Sequencing technology is usually formulated as a high-dimensional count matrix(Fig.2). The count matrix is of the size (**ncells**, **ngenes**), where **ncells** and **ngenes** corresponds to number of cells and genes measured in the dataset. Considering each row (cell) of the matrix as an "observation", and each column

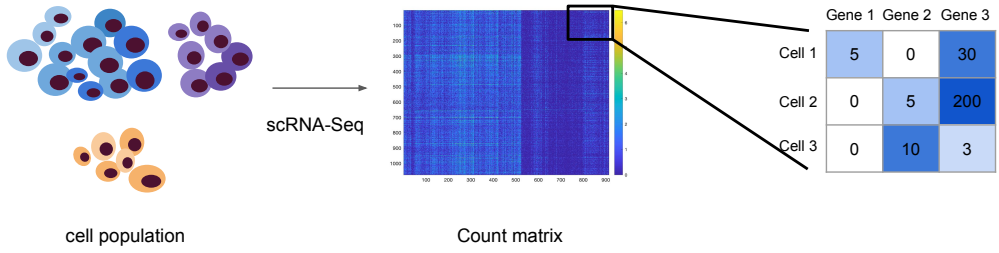

Figure 2: single-cell RNA Sequencing

(gene) as a feature, the matrix is no different from the high-dimensional dataset we usually meet in the machine learning field. Our PBMCs dataset includes a count matrix that measures 3000 genes in 3000 different cells.

You will use Python for this problem. Please make sure you familiarize yourself with Python and Numpy package before you start. Some reference materials:

- Python tutorial: https://cs4540-f18.github.io/notes/python-basics.
- Numpy tutorial: https://numpy.org/doc/stable/user/quickstart.html

And you may also need to install Python packages: **pandas**, **matplotlib**, **sklearn** and **umap-learn**. There are sufficient instruction materials for the installation of those packages online.

All the files needed for the programming are wrapped up into **hw2_coding.zip** on canvas, please make sure you download the files before you begin. **hw2_coding.zip** includes:

- **hw2_1.py**, the python script for sub-problem 1.
- **hw2_2.py**, the python script for sub-problem 2.
- **counts_PBMC.csv**, the count matrix of PBMC dataset.
- **celltypes_PBMC.txt**, the cell type annotation file of the PBMC dataset, use will need to use this file for a better visualization of the dataset.

You will need to:

1. Implement "PCA" algorithm from scratch using Numpy. Open **hw2_1.py** and read the instruction in the script, you will need to implement "PCA" in three different ways, fill in the blank part of three functions **pca**, **svd** and **pca_sklearn**. Run the script and see how the plot looks like. Submit the generated **pdf** and **npy** files.

2. Implement "TSNE" and "UMAP" using packages **sklearn** and **umap-learn**. Open **hw2_2.py** and read the instruction in the script. You will need to fill in the blank part of **tsne_op** and **umap_op**. Play with hyper-parameters "perplexity" in "TSNE", "n_neighbors" and "min_dist" in "UMAP". Compare the results generated with different hyper-parameters, briefly discuss how those hyper-parameters affect the final visualization results and find the set of hyper-parameters that can generate the best visualization(cell of different cell types are clearly separated). Submit the generated **pdf** file along with your discussion. (**Notes:** You can refer to some online materials when tuning the hyper-parameters. "TSNE" and "UMAP" is computationally expensive, people usually reduce the dimensions of the data to around $30 - 100$ using "PCA" before they feed the data into "TSNE" and "UMAP". Make sure you do that too with your implementation.)

Solution:
See hw2_1.sol.py and hw2_2.sol.py.

References

[1] Hyun Min Kang, Meena Subramaniam, Sasha Targ, Michelle Nguyen, Lenka Maliskova, Elizabeth McCarthy, Eunice Wan, Simon Wong, Lauren Byrnes, Cristina M Lanata, Rachel E Gate, Sara Mostafavi, Alexander Marson, Noah Zaitlen, Lindsey A Criswell, and Chun Jimmie Ye. Multiplexed droplet single-cell RNA-sequencing using natural genetic variation. *Nat. Biotechnol.*, 36(1):89–94, January 2018.