Made by Sung Jae Hyuk
GitHub: @okaybody10
E-mail: okaybody10@korea.ac.kr

Homework Assignment 1
Coding Guidelines

2023-01-25

Introduction

Hello everyone! My name is Jae Hyuk Sung, an author of this coding guidelines. I am glad to be able to write this guideline report.
I will introduce the task and the file you need to implement for each question. Since each question has lots of difficult parts, especially those that require mathematical knowledge, I will provide guidelines for solving them.
Since this report is written in LaTeX, it is written **only in English** for readability.
Also, you don't have to fill out the inline questions.
If you have any questions or comments, please feel free to let me know. Here is my contact information.

E-mail:	okaybody10@korea.ac.kr
KakaoTalk:	okaybody10
Instagram:	@okaybody5

Follow is welcome. *^^*

Q1. k-Nearest Neighbor classifier

Implement files

- *k_nearest_neighbor.py*
- *knn.ipynb*

k_nearest_neighbor.py

You need to implement a total of 4 functions: 3 functions depending on the number of iterations and 1 function predicting the actual label.
It is a bit easier for you to implement a function that uses two loops and one loop.
But, you have to construct mathematical formulas when coding with loops to implement the no-loop function.
No loop function should take less than 0.1 second.
Hint: KNN implies that you need to express a matrix representation of the L2 distances.
Let $x = \begin{bmatrix} x_1 & x_2 & \cdots & x_t \end{bmatrix}^T$ is training set, and $y = \begin{bmatrix} y_1 & y_2 & \cdots & y_n \end{bmatrix}^T$ is object set, and distance matrix A has $n \times t$ size. (It also means $A[i, j]$ implies the distance with y_j and x_j.)
In other words, we can rewrite matrix A as follows:

$$A = \begin{bmatrix} \sqrt{y_0^2 - 2y_0x_0 + x_0^2} & \cdots & \sqrt{y_0^2 - 2y_0x_t + x_t^2} \\ \sqrt{y_1^2 - 2y_1x_0 + x_0^2} & \cdots & \sqrt{y_1^2 - 2y_1x_t + x_t^2} \\ \vdots & \ddots & \vdots \\ \sqrt{y_n^2 - 2y_nx_0 + x_0^2} & \cdots & \sqrt{y_n^2 - 2y_nx_t + x_t^2} \end{bmatrix}$$

For convenience, we define the element-wise function $f(W)$ which returns the square root of elements.
Then, there exists a matrix B which satisfies $A = f(B)$.
Also, we can express B with another element-wise function, broadcasting, and Hadamard product.

Finally, you need to implement "predict_lables", which returns the **index** of largest score.
Note that I emphasize the index! You don't need to check the sorted array.

KNN.ipynb

Fortunately, functions in KNN are not difficult! You need to split data for cross-validation and perform k-fold cross-validation. *Hint*: Search for array_split function in NumPy package, and perform K-fold CV with three loops.

Q2. Support Vector Machine(SVM)

Implement files

- *linear_svm.py*
- *linear_classifier.py*
- *svm.ipynb*

linear_svm.py

You need to implement two function, naive version and vectorized version.
In “navie”function, there is already implemented to calculate loss, and you have to implement to get gradient.
To calculate gradient, you need to modify the code that computes the loss **a little bit**.
Next, you need to implement the function of the vectorized.
Naive function should take less than 0.05, vectorized function should take less than 0.007.

Hint: In vectorized function, you have to formulate the mathematical formula.
First, you have to extract the ansewr probability, and then subtracte them.
Note that answer matrix only dependes on first index of X, score matrix.
To resolve this problem, let us think of a binary matrix that actas as accept and reject.
In other words, if binary matrix's element(especially ith column and jth row, (i, j)) has value 1, then we “accept”to pass (i, j) value.
In the same manner, we can define reject.
For example, let A is binary matrix, and X is score matrix. More specifially,

$$A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix}, \qquad X = \begin{bmatrix} 0.4 & 1 & 2.5 \\ 3.4 & 1 & 0.2 \\ -0.4 & -1 & 0 \end{bmatrix}$$

Since $A[0, 0] = 1$, we accept to pass value of $X[0, 0]$. But, $A[0, 2] = 0$, so we reject to pass value of $X[0, 2]$.
As a result,

$$A \odot X = \begin{bmatrix} 0.4 & 0 & 0 \\ 0 & 1 & 0.2 \\ 0 & 0 & 0 \end{bmatrix}$$

Please don't forget the L2 regularization term! Regularization term should be reflected in loss and gradient.

linear_classifier.py

In this file, you need to implement (minibatch)stochastic gradient descent(a.k.a. SGD).
More specifically, you need to draw as many datasets as there are minibatches from the entire dataset.
Hint: Search for choice function in NumPy package, and perform gradient descent.

Finally, you have to predict labels, and it is more easier than others.

svm.ipynb

You have to perform hyperparameter tuning on the validation set.
Note that you can't use test set, so when you do tuning, you can only judge accuracy from validation set.

Q3. Softmax

Implement files

- *softmax.py*
- *softmax.ipynb*

Important property of softmax function

For convenience, let us define $f(\mathbf{x})$ is softmax function, where $\mathbf{x}$ is vector with dimension k, and c is scalar. *i.e.* $c \in \mathbb{R}$
Then, softmax function has **"translation invariant"** property. More specifically,

$$f(\mathbf{x} + c) = f(\mathbf{x})$$

If you want to solve a problem without the vanishing/exploding problem, you should use this property properly.

softmax.py

As before, you need to implement two functions in this file.
First, on navie function, you can complete function using only loop.
Also, if you use outer product of vector, you can easily get the gradient.
Next is vectorized function.
If you use the sum properly in a matrix, you can easily get loss.
Note that you don't have to check all probabilities! Since you have answer, you can get loss by checking only answer probability. *Hint*: let p is score vector (Not probability vector) and y is one-hot encoding answer vector, and loss is $\mathcal{L}$.
Then,

$$\frac{\partial \mathcal{L}}{\partial p} = p - y$$

softmax.ipynb

If you have implemented the above *svm.ipynb* well, you can easily do this part of tuning the hyperparameter.

Q4. Two-Layer Neural Network

Implement file

- *layers.py*
- *two_layer_net.ipynb*

layers.py

This is core of assignment 1!
You need to complete 4 function, and copy & paste 2 function. (SVM loss and softmax loss)
In assignment 1, you only use affine layer and ReLU activation function.
Affine layer is simple linear function! In other words, $y = xW + b$ and x is input, W is weight, b is bias.
If you use trace trick, you can also get gradient easily.
It is easy to implement ReLU if you use boolean indexing.