ESE 546, FALL 2022

HOMEWORK 1

RUI JIANG [RJIANG6@SEAS.UPENN.EDU]

Solution 1 (Problem 1.a, Time spent: 1 hour)**.**

Report 1.a: The slack variable formulation allows some data points to violate the margin constraint. To minimize these violations while still maximizing the margin, we add a penalty term to the objective function.

One common formulation is:

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}\|\theta\|^2 + C\sum_{i=1}^{n}\xi_i \\ \text{subject to} \quad & y_i(\theta^T x_i + \theta_0) \geq 1 - \xi_i, \quad \forall i = 1, \ldots, n \\ & \xi_i \geq 0, \quad \forall i = 1, \ldots, n \end{aligned}$$

where $\sum_{i=1}^{n}\xi_i$ is the penalty and C contraolls the trades off betweeen margin size and violation penalty.

One example is $C = \frac{1}{n}$, then we are minimize $\frac{1}{2}\|\theta\|^2 + \frac{1}{n}\sum_{i=1}^{n}\xi_i$. $\frac{1}{n}\sum_{i=1}^{n}\xi_i$ is a normalized penalty by the number of samples.

Solution 2 (Problem 1.b, Time spent: 0.5 hour)**.**

Part (1.b): The support samples in this SVM are the training data points that lie closest to the decision bounday in an SVM. They are critical points which define the boundaries for SVM. My SVM model has roughtly 37% of training data are support samples.

Solution 3 (Problem 1.c, Time spent: 1 hour)**.**
Part (1.c): The `svm.SVC` in sklearn use One-vs-One (OvO) method in default. To build a K class classifier, it build $\frac{K(K-1)}{2}$ classifier. For MNIST (10 classes), it builds 45 classifiers. For example, it's 0 vs 1, 0 vs 2, ... 8 vs 9. For each image, all 45 classifiers run on this image. The class with the hightest total number of votes wins.

The following is the alternative method:
One-vs-All method:

(1) We train one classifier per class. Each classifier is trained to distinguish that specific class from all other classes combined.
Classifier 1: Is it a "0"?
Classifier 2: Is it a "1"?
...
(2) We will have 10 classifiers.
(3) Run 10 classifiers for each image.
(4) In stead of looking at Yes or No vote, we will look at the raw distance from the hyperplane.
(5) The classifier with most confident to its class is the winner.

Solution 4 (Time spent: 1 hour)**.**
Part (1.d):

1. What does the parameter named "shrinking" in svm.SVC do?
The "shrinking" parameter is a boolean (default=`True`) that specify whether to use the shrinking heuristic. Based on the document, if the number of iterations is large, then shrinking heuristic can shorten the training time. However, if it loosely solve the optimization problem the code without using shrinking may be much faster. . Because it will temporarily remove training points that are unlikely to become support vectors. This can speed up the training.

2. Explain what optimization algorithm is used to fit the SVM in scikit-learn.
Scikit-learn's svm.SVC is based on LIBSVM, which uses the Sequential Minimal Optimization (SMO) algorithm. SMO solves the SVM dual optimization problem by breaking it into a series of very small subproblems involving only two Lagrange multipliers at a time. Each subproblem has a closed-form solution, making the algorithm efficient and numerically stable. SMO is well-suited for kernelized SVMs, where the optimization problem is convex but high-dimensional.

3. Why does fitting SVMs require a decent amount of RAM?
Fitting an SVM is memory-intensive because a grid between every data point and every other data point need to be created. For N samples, this matrix has dimensions $N \times N$. If there are 1000 samples, then the matrix has $1,000,000$ dimensions.

4. What do the parameters C and γ do? What are their default values?
C controls the trade-off between maximizing the margin and minimizing the classification error on the training data. A lower C encourages a wider margin (smoother decision boundary) but allows more misclassifications. A higher C penalizes misclassifications heavily, leading to a complex boundary that tries to classify all training points correctly.

Solution 5 (Time spent: 1 hour)**.**

Part (1.e): The ratio of the number of suport samples to the total number of training samples is 37.66%.

Solution 6 (Problem 1.f,Time spent: 1 hour)**.**

Part (1.f):

Validation Accuracy: 0.9545

Validation Error: 0.0455

The Confusion matrix is (row is real label, column is predicted label):

	0	1	2	3	4	5	6	7	8	9
0	196	0	0	0	0	0	1	1	2	0
1	0	198	0	0	0	1	0	0	0	1
2	1	0	191	2	2	0	1	3	0	0
3	0	1	2	189	0	3	0	2	3	0
4	0	0	1	0	187	0	5	0	0	7
5	0	0	1	2	0	192	2	1	1	1
6	0	0	0	0	0	0	200	0	0	0
7	0	1	1	0	3	0	0	193	0	2
8	1	0	0	3	1	2	0	1	189	3
9	1	1	0	3	8	2	0	10	1	174

The observed pattern is:

1. The diagonal are the right prediction, which is pretty significant.

2. Number 6 is the most accurate. The shape of 6 is more different compare to others

3. The wrong prediction is not symatic. For exaomple 9 could be misclassified as 7 for 10 times, but 7 is misclassified as 9 for only 2 times. The reason could be the edge detection is not clear for 7 and 9.

Solution 7 (Problem 1.g, Time spent: 1 hour)**.**

Part (1.g): The parameter I used are:

```
params = {
    'C': [0.1, 1, 10],
    'kernel': ['linear', 'rbf'],
    'gamma': ['scale', 'auto']
}
```

The best hyperparameters are:

- C: 10
- gamma: scale
- kernel: rbf

$C = 10$ means the model will give high penalty for misclassifications.

The gamma is $scale$ because it can help to prevent overfitting or underfitting.

The kernel selected is rbf, which means the data cannot use a liniearly plane to classify, which meakes sense because the MINIST data is a complex data set with non-linear pattern.

Solution 8 (Problem 1.h, Time spent: 1 hour)**.**
Part (1.h): The following is the original image, filter image, Gabor kernel real and imaginary kernel after calling $gabor_kernel()$.

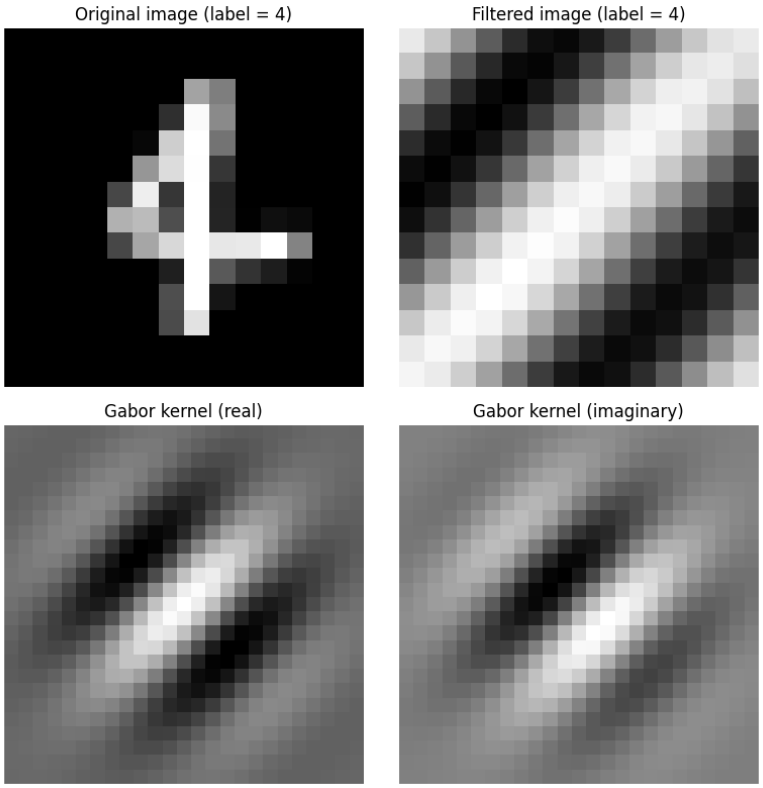

FIGURE 1. The example image with the real and imaginary coefficients and filtered image

The following is the plot of 8 pairs of real and imaginary coefficients from the 16 filters.
Gabor Filter SVM Pipeline:

(1) **Data Preparation:** Subsampled 1,000 images (100 per class), split 50/50 into 500 train / 500 validation.
(2) **Filter Bank:** 16 Gabor filters using frequencies [0.05, 0.25], thetas [$\pi/4$, $\pi/2$, $3\pi/4$, π], bandwidths [0.1, 1].
(3) **Feature Extraction:** Applied all 16 filters to each 14×14 image, extracting real coefficients $\rightarrow$ 3136 features per image (16×196).
(4) **Preprocessing:** Applied StandardScaler to normalize features then used PCA to reduce dimensionality, retaining the most informative features while reducing computational cost.
(5) **SVM Training:** GridSearchCV with RBF kernel, C $\in$ [0.1, 1, 10], gamma $\in$ [scale, auto], 5-fold CV.

The best validation accuracy of the SVM from the search is:

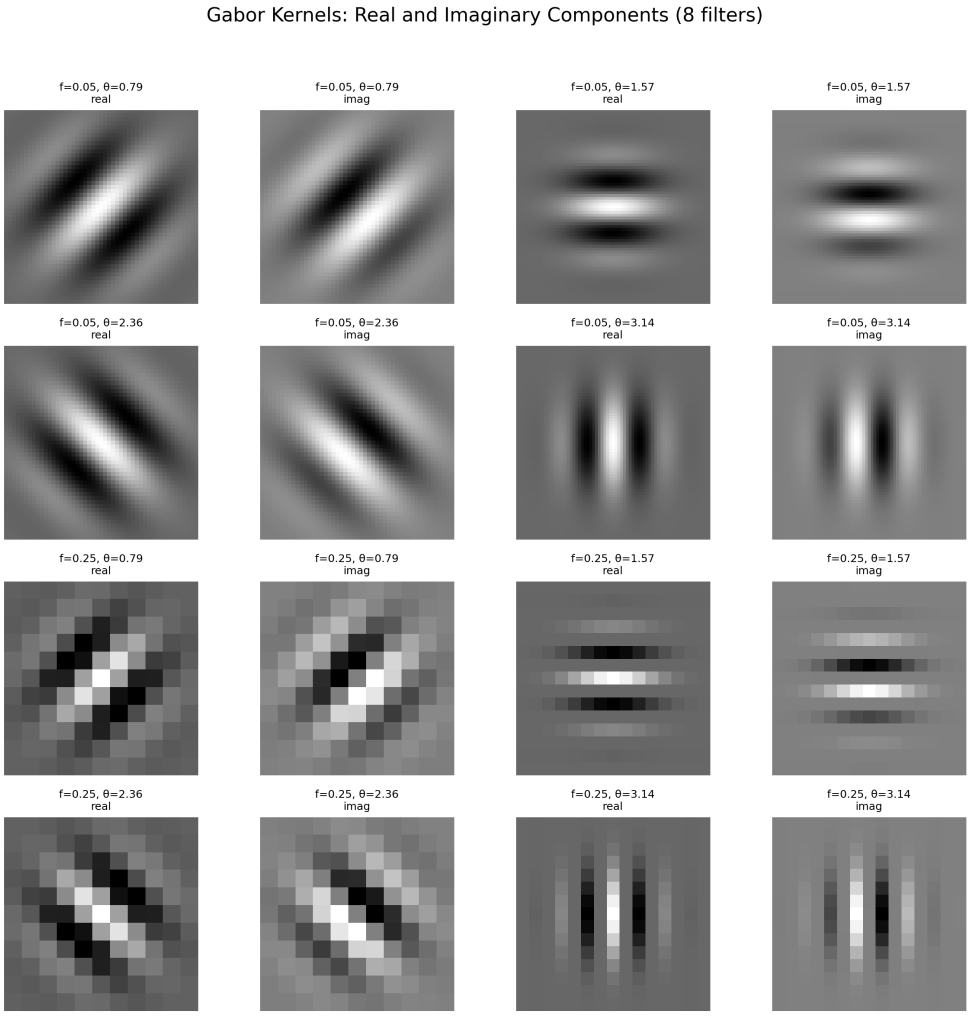

FIGURE 2. 8 pairs of real and imaginary coefficients from the 16 Gabor filters

- PCA components: 50
- Explained variance ratio: 0.9963
- Best SVM parameters: 'C': 10, 'gamma': 'scale', 'kernel': 'rbf'
- Validation Accuracy: 0.8380
- Validation Error: 0.1620

The SVM search running against the original training data and test data, the best SVM parameters are the same: 'C': 10, 'gamma': 'scale', 'kernel': 'rbf'

Key point: Gabor features allow good classification with fewer samples (500 vs 8000) because they extract texture/edge information that is more discriminative than raw pixels.

Solution 9 (Part 2 Jensen's Inequality, Time spent: 2 hour)**.**

Let X be a random variable with $\mu = \mathbb{E}[X]$ and let $\varphi : \mathbb{R} \to \mathbb{R}$ be convex and finite. We prove that

$$\mathbb{E}[\varphi(X)] \geq \varphi(\mu).$$

Finite-support proof (as allowed). Assume X takes values in a finite set $\{x_1, \ldots, x_n\}$ with probabilities $p_i = \mathbb{P}(X = x_i)$, where $p_i \geq 0$ and $\sum_{i=1}^n p_i = 1$. Then

$$\mu = \mathbb{E}[X] = \sum_{i=1}^n p_i x_i, \qquad \mathbb{E}[\varphi(X)] = \sum_{i=1}^n p_i \, \varphi(x_i).$$

A function φ is convex iff for any weights $(p_i)_{i=1}^n$ summing to 1,

$$\varphi\left(\sum_{i=1}^n p_i x_i\right) \leq \sum_{i=1}^n p_i \, \varphi(x_i).$$

Applying this with the above (p_i) gives

$$\varphi(\mu) = \varphi\left(\sum_{i=1}^n p_i x_i\right) \leq \sum_{i=1}^n p_i \, \varphi(x_i) = \mathbb{E}[\varphi(X)],$$

which is exactly Jensen's inequality.

Solution 10 (Time spent: 20 hour)**.**

Problem 3.a:

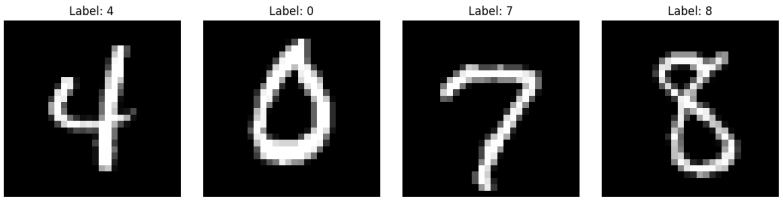

FIGURE 3. Randomly selected numbers

Solution 11 (Time spent: 3 hour)**.**
Problem (3b):

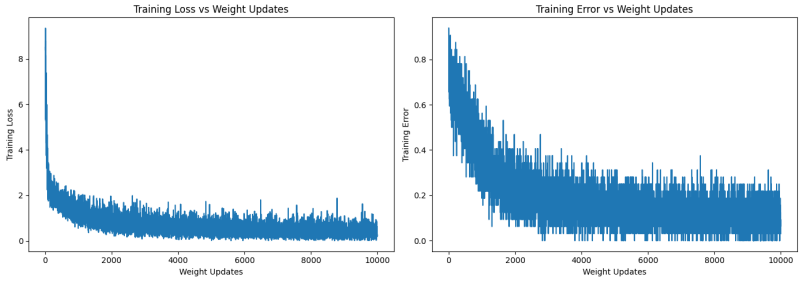

FIGURE 4. The plots of training loss and training error as a function of the number of weight updates

Both training loss and training error decreased as the number of weight updates increase. But after a certain number, the loss and training error stopped decrease.

Solution 12 (Time spent: 2 hour)**.**

Problem (3c): The validation loss and validation error also decreased as the number of weight

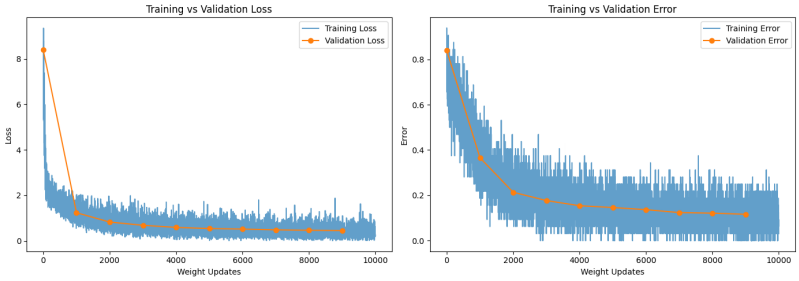

FIGURE 5. Plot the self NN validation loss and validation error as a function of the number of weight updates

updates increase. After 4000 weight updates, the decrease is not obvious anymore.

Solution 13 (Time spent: 1 hour)**.**

Problem (3d): The PyTorch NN enter the flat zone faster than my NN.

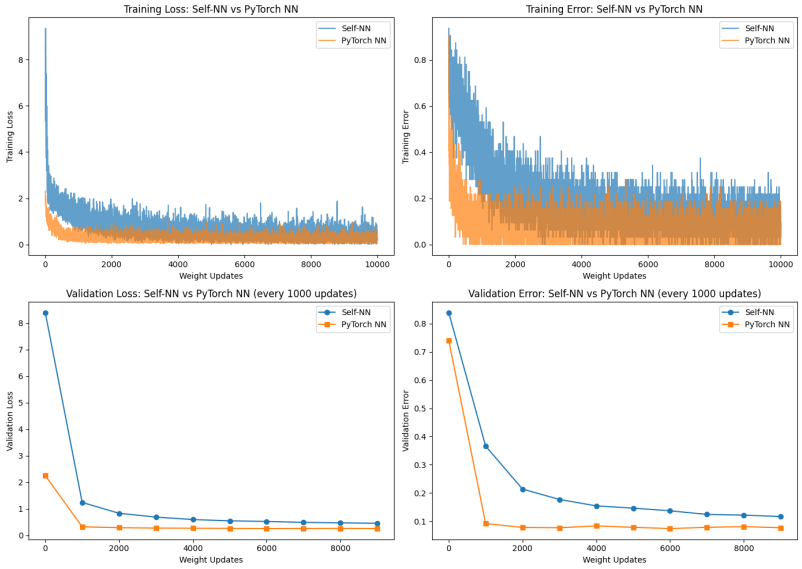

FIGURE 6. Comparing between my Neural Network and PyTorch Neural Network