BLG 506E Computer Vision 2022-2023 Spring Assignment II

Resul Dagdanov - 511211135

***Abstract*— This assignment focuses on implementing and using multiclass Support Vector Machine (SVM) and Softmax linear classifiers for image classification is the focus of this assignment. The development and use of these classifiers, practice utilizing vectorized gradient code, and evaluation of the classifiers' performance using numeric gradient checking are the main goals of this assignment. There are two key components to the assignment. We put the SVM and Softmax classifiers into practice and use them to categorize the CIFAR10 public dataset in the first section. Implementing vectorized gradient code, evaluating it against naive implementations, and utilizing numeric gradient testing to verify the code's accuracy is part of the second portion of the assignment. We have developed code in the linear classifier.py file and adhere to the guidelines in the linear classifier.ipynb that is given a notebook in order to complete the assignment. We have received instructions for testing and evaluating the classifiers' performance as well as guidance on how to incorporate them in the notebook. This assignment's overall goal is to increase your knowledge of and practical proficiency in creating and using linear classifiers for image classification tasks.**

***Index Terms*— Image Classifier, Softmax Classifier, SVM, Linear Classifier**

I. Introduction

Linear classifiers are a popular class of algorithms for solving classification problems in machine learning. They are simple yet powerful methods that work by finding a linear boundary between classes in feature space. This boundary is used to make predictions for new, unseen data. Linear classifiers have been widely studied and applied in many areas, including computer vision [1], natural language processing [2], and bioinformatics [3]. Some of the most well-known linear classifiers include the perception [4], logistic regression [5], Support Vector Machines (SVMs) [6], and Softmax regression. Overall, linear classifiers are an important and widely used class of algorithms in machine learning, with numerous applications and research directions.

The CIFAR10 dataset is a popular benchmark dataset for image classification tasks in machine learning, which contains 60,000 color images divided into 10 classes, each having 6,000 images. The image size of the photos in the dataset is relatively modest at 32x32 pixels, compared to other similar datasets like ImageNet. The dataset comprises ten classes, including car, truck, boat, cat, deer, dog, frog, horse, ship, and airplane, with an equal number of images in each class in the training and test sets. The training set contains 50,000 images, while the test set contains 10,000 images. Due to the small image size and significant similarities between some classes, such as dogs and cats or ships and airplanes, the CIFAR10 dataset poses a significant challenge for classification tasks. The dataset has been widely used to evaluate the performance of various machine learning and deep learning models and has played a crucial role in the development of state-of-the-art computer vision methods.

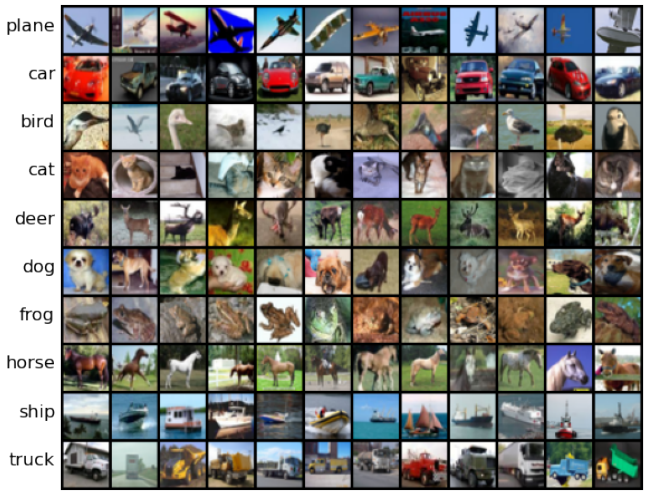

Fig. 1. CIFAR10 dataset example class label images [7].

A well-liked approach for resolving multiclass classification problems is the Softmax linear classifier, commonly referred to as Softmax regression. By employing the Softmax function to convert the output of a linear classifier into a probability distribution across the classes, it is a generalization of logistic regression that can handle more than two classes. The Softmax function returns a probability distribution across the classes after receiving a vector of real-valued scores as input. The exponential function of a class's score divided by the total of all classes' exponential scores is the definition of the Softmax function. The loss function used by the Softmax classifier is commonly the cross-entropy loss [8], which assesses the similarity between the predicted probability distribution and the actual labels. The Softmax classifier learns a weight matrix that maps the input features to the scores for each class. The Softmax classifier can be enhanced with regularization, dropout, or other methods to avoid overfitting. It can be trained using gradient descent or other optimization algorithms. The Softmax classifier is a vital part of deep learning architectures like neural networks and has been extensively employed in a variety of applications, including image classification, speech recognition, natural language processing, and recommendation systems.

R. Dagdanov is with Department of Aeronautical and Astonautical Engineering, Istanbul Technical University, Turkey dagdanov21 at itu.edu.tr

Similarly, another class of supervised learning techniques known as Support Vector Machines [6] can be applied to both classification and regression applications. The foundation of SVMs is the idea of locating the hyperplane with the biggest margin of separation between the data points of various classes. The gap between the nearest data points from each class and the hyperplane is referred to as the margin. Through the application of kernel functions, SVMs aim to maximize the margin while reducing the classification error. They are particularly good at handling high-dimensional data and nonlinear borders. The basic linear SVM generates a decision boundary by solving a convex optimization problem, with the goal of maximizing the margin while guaranteeing that the training data are accurately classified. SVMs use kernel functions to transfer the input data into a high-dimensional feature space where a linear decision boundary can be created in the case of nonlinear boundaries. The polynomial kernel, radial basis function (RBF) kernel, and sigmoid kernel are well-known kernel functions. SVMs are superior to conventional classification algorithms in a number of ways, including as high accuracy, robustness to outliers, and the capacity to handle large and sparse data sets. Additionally, they can be expanded to tackle multiclass classification issues and integrated with other strategies, including regularization and ensemble approaches, to perform better. SVMs are still a hot topic of research in machine learning despite being widely employed in a variety of applications, including image classification.

II. Methodology

In this section, a detailed inspection of a linear classifier algorithm is carried out with related equations. It is possible to formulate the linear classifier as follows:

- Initialize the weight vector w and the bias term b with small random values.
- For each input feature vector x, compute the scores for all classes using the dot product between w and x, and add the bias term b.
- Compute the loss function based on the scores and the true labels. For SVM, use the hinge loss, and for Softmax, use the cross-entropy loss.
- Add a regularization term to the loss function to prevent overfitting. L2 regularization is commonly used, which penalizes large weight values.
- Compute the gradients of the loss function with respect to the weights and biases using the chain rule.
- Update the weights and biases using gradient descent or a variant of it, such as stochastic gradient descent (SGD) or mini-batch SGD. The learning rate determines the size of the update step.
- Repeat the above steps for multiple epochs or until the loss function converges.
- To make a prediction for a new input feature vector x, compute the scores using the learned weights and biases, and choose the class with the highest score as the predicted class.
- Evaluate the performance of the model on a separate test set using metrics such as accuracy, precision, recall, and F1 score.

Given a training dataset of N feature vectors $x_1, x_2, ..., x_N$, with corresponding true labels $y_1, y_2, ..., y_N$, where $x_i \in \mathbb{R}^D$ and $y_i \in 1, 2, ..., K$ for a K-class problem, we want to learn a weight vector $w \in \mathbb{R}^D$ and a bias term $b \in \mathbb{R}$ that can be used to predict the class labels of new, unseen feature vectors.

The linear classifier model computes the score for class k as:

$$s_k = {w_k}^T.x + b \tag{1}$$

where w_k is the weight vector for class k and x is the feature vector.

The Softmax classifier predicts the class probabilities as:

$$P(y_i = j \mid x_i) = \frac{e^{s_j}}{\sum_{k=1}^{K} e^{s_k}} \tag{2}$$

where s_k is the score for class k and the denominator ensures that the probabilities sum to 1 over all classes.

The SVM classifier predicts the class as:

$$y = argmax_k s_k \tag{3}$$

where s_k is the score for class k and $\arg\max$ returns the index of the maximum score.

To train the linear classifier, we minimize a loss function L that measures the difference between the predicted scores and the true labels. For the Softmax classifier, the cross-entropy loss is commonly used:

$$L = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{K} y_{ij}\log(P(y_{ij} = 1 \mid x_i)) \tag{4}$$

where y_i is the true label for feature vector x_i. For the SVM classifier, the hinge loss is commonly used:

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + \Delta) \tag{5}$$

where s_j is the score of class j, y_i is the true class label of the input x_i, and Δ is the margin, which is a hyperparameter that controls the distance between the scores of the correct class and the incorrect classes. The hinge loss is a measure of the difference between the predicted scores and the true scores, with a penalty applied only if the score of the incorrect class is greater than the score of the correct class by at least the margin. The goal is to minimize the sum of the hinge losses over the entire training dataset, which will improve the accuracy of the classifier.

To prevent overfitting, we add a regularization term to the loss function that penalizes large weight values. The L2 regularization term is commonly used:

$$R(W) = \sum_{k}\sum_{l} W_{k,l}^2 \tag{6}$$

where W is the weight matrix, and the sum is taken over all the elements of the matrix. The L2 regularization term adds a penalty proportional to the square of the magnitude of the

weight vector, which encourages the model to use smaller weights. The hyperparameter λ controls the strength of the regularization and is usually set by cross-validation.

The total loss function for the linear classifier is then:

$$L = \frac{1}{N}\sum_i L_i + \lambda R(W) \tag{7}$$

where N is the number of training examples. The goal is to minimize the total loss function over the weights W, which will improve the generalization performance of the classifier.

To train the model, we use gradient descent to update the weights and biases in the opposite direction of the gradients of the loss function:

$$W_{t+1} = W_t - \eta \nabla_{W_t} L \tag{8}$$

$$b_{t+1} = b_t - \eta \nabla_{b_t} L \tag{9}$$

where W_t and b_t are the weights and biases at iteration t, η is the learning rate, and $\nabla_{W_t} L$ and $\nabla_{b_t} L$ are the gradients of the loss function with respect to the weights and biases, respectively. The gradients can be computed using the chain rule of calculus:

$$\nabla_{W_t} L = \frac{1}{N}\sum_i \nabla_{W_t} L_i + 2\lambda W_t \tag{10}$$

$$\nabla_{b_t} L = \frac{1}{N}\sum_i \nabla_{b_t} L_i \tag{11}$$

where $\nabla_{W_t} L_i$ and $\nabla_{b_t} L_i$ are the gradients of the loss function with respect to the weights and biases for the i-th example in the training set. The factor of 2 in the L2 regularization term arises from the derivative of the square function. By iteratively updating the weights and biases using gradient descent, we can minimize the loss function and improve the performance of the model on the training set.

III. Experiments

A. Linear SVM Classifier

Initially, we implemented a fully-vectorized loss function for the Support Vector Machine (SVM) classifier. We also implemented the fully-vectorized expression for its analytic gradient and checked our implementation using a numerical gradient. Then, we used a validation set to tune the learning rate and regularization strength. We optimized the loss function with Stochastic Gradient Descent (SGD) to minimize the loss and obtain the optimal weights and biases for the classifier. Finally, we visualized the final learned weights to gain insights into the classifier's decision-making process. By completing these tasks, we were able to gain a better understanding of the SVM classifier and its performance on the CIFAR10 dataset. We were also able to gain experience in implementing and optimizing machine learning models using vectorized code and numerical gradient checking.

In the first experiment, we tested the naive version of SVM loss implemented in the linear-classifier.py file. The provided code for the loss part of the svm-loss-naive function was used for this task. The expected loss value was 9.000197. After running the code, we obtained a loss value of **9.000869**, which slightly deviated from the expected value. The small difference could be due to variations in computer precision or other factors. Nonetheless, we can conclude that the implementation of the SVM loss function is correct since the obtained loss is in the same order of magnitude as the expected value.

We have performed numeric gradient checking on the gradients of our SVM loss. We have observed relative errors less than 1e-5 as expected in Table I.

TABLE I

NUMERIC AND ANALYTIC GRADIENTS WITH RELATIVE ERROR WITH NAIVE SVM LOSS IMPLEMENTATION.

Numerical	Analytic	Relative Error
0.031599	2.022362	9.692307e-01
0.111444	7.132441	9.692308e-01
0.011204	0.717044	9.692308e-01
-0.046128	-2.952163	9.692308e-01
0.071948	4.604700	9.692308e-01
0.025688	1.644042	9.692308e-01
0.185388	11.864857	9.692308e-01
-0.021740	-1.391379	9.692308e-01
-0.159613	-10.215223	9.692308e-01
0.092690	5.932135	9.692308e-01

After turning on the regularization for naive SVM loss implementation, the results are shown in Table II.

TABLE II

NUMERIC AND ANALYTIC GRADIENTS WITH RELATIVE ERROR WITH NAIVE SVM LOSS IMPLEMENTATION WITH REGULARIZATION.

Sample	**Numerical**	**Analytic**	**Relative Error**
1	0.124849	2.115612	8.885506e-01
2	0.168915	7.189911	9.540920e-01
3	0.148752	0.854592	7.034875e-01
4	-0.024936	-2.930972	9.831279e-01
5	-0.008570	4.524181	1.000000e+00
6	-0.103155	1.515200	1.000000e+00
7	-0.335573	11.343895	1.000000e+00
8	-0.222176	-1.591814	7.550417e-01
9	0.681163	-9.374447	1.000000e+00
10	-0.004089	5.835356	1.000000e+00

These two tables: a) Table I and b) Table II were naive SVM loss implementation results.

In the code materials, we have implemented svm-loss-vectorized function which calculates SVM loss in a vectorized format without using any explicit loops. It is expected that the speed and the performance between non-vectorized and vectorized SVM loss implementations should be in a range of 15-120x speed-up with vectorized calculations. We have calculated SVM naive loss as **9.002106** in **201.37 ms** and SVM vectorized loss as **9.002106** in **2.52 ms**. These results show that we have achieved the speed-up of **79.95X** in vectorized format compared to naive implementation.

The same comparison between SVM naive implementation and vectorized implementation could be conducted for calculating the gradients of the obtained loss values. SVM naive gradient calculation result took **225.01 ms** while SVM vectorized gradient calculation result too **1.95 ms** making

a speed-up of **115.49X** with gradient result difference of 2.35e+03; which is not a big difference.

After finishing implementations of SVM loss and gradient calculation functions, we conducted training with vectorized SVM implementation code on the CIFAR10 dataset. The results of this training are shown in Table III.

TABLE III
SVM CLASSIFIER TRAINING LOSS.

Iteration	Loss
0 / 1500	9.000784
100 / 1500	9.000764
200 / 1500	9.000777
300 / 1500	9.000768
400 / 1500	9.000779
500 / 1500	9.000771
600 / 1500	9.000771
700 / 1500	9.000769
800 / 1500	9.000771
900 / 1500	9.000771
1000 / 1500	9.000771
1100 / 1500	9.000789
1200 / 1500	9.000787
1300 / 1500	9.000769
1400 / 1500	9.000777

The learning rate is priorly chosen as $3e-100$ and the regularization strength is $2.5e4$. The total training took about 1.905891 seconds. As expected, the loss is observed to be below 9.000777 value.

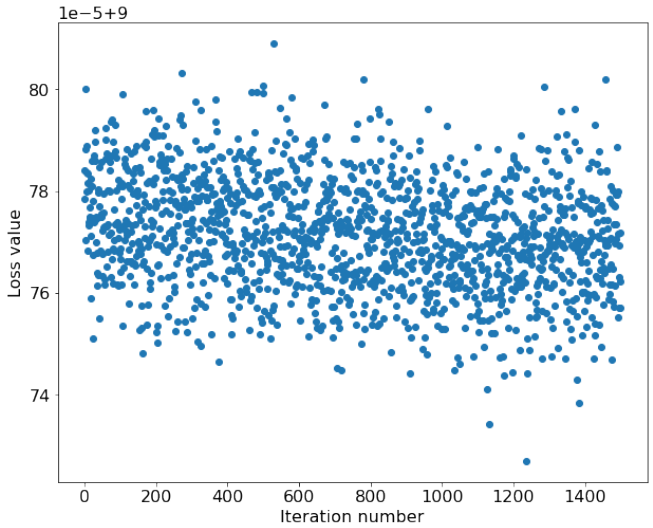

Fig. 2. SVM Training Iteration Loss.

Table IV shows that training and validation set accuracies. It is expected to see an accuracy of less than 20%.

TABLE IV
SVM CLASSIFIER TRAINING ACCURACY RESULTS ON CIFAR10.

Training Accuracy	Validation Accuracy
9.35%	9.11%

Initially, the performance of our model was below our goal. However, we took steps to improve it by modularizing the functions and implementing them as LinearSVM.

To further improve the model, we used the validation set to tune the hyperparameters, which included the regularization strength and learning rate. We experimented with different ranges of values for these parameters to find the best combinations.

To successfully complete the assignment, the best model found through cross-validation should have achieved an accuracy of at least 37% on the validation set. Fortunately, we were able to beat this goal with our best model achieving over 38.1% validation accuracy with **39.11%**.

TABLE V
SELECTING DIFFERENT HYPERPARAMETERS DURING SVM TRAINING.

Epoch	Learning Rate	Regularization Strength
1	5.0e-04	1.0e-04
2	5.0e-04	5.0e-04
3	5.0e-04	1.0e-03
4	5.0e-04	5.0e-03
5	5.0e-04	1.0e-02
6	1.0e-03	1.0e-04
7	1.0e-03	5.0e-04
8	1.0e-03	1.0e-03
9	1.0e-03	5.0e-03
10	1.0e-03	1.0e-02
11	5.0e-03	1.0e-04
12	5.0e-03	5.0e-04
13	5.0e-03	1.0e-03
14	5.0e-03	5.0e-03
15	5.0e-03	1.0e-02
16	1.0e-02	1.0e-04
17	1.0e-02	5.0e-04
18	1.0e-02	1.0e-03
19	1.0e-02	5.0e-03
20	1.0e-02	1.0e-02
21	5.0e-02	1.0e-04
22	5.0e-02	5.0e-04
23	5.0e-02	1.0e-03
24	5.0e-02	5.0e-03
25	5.0e-02	1.0e-02

We have obtained this result with cross-validation hyperparameter tuning. For the hyperparameters, we have selected:

1) learning rates: 5e-4, 1e-3, 5e-3, 1e-2, 5e-2
2) regularization strengths: 1e-4, 5e-4, 1e-3, 5e-3, 1e-2

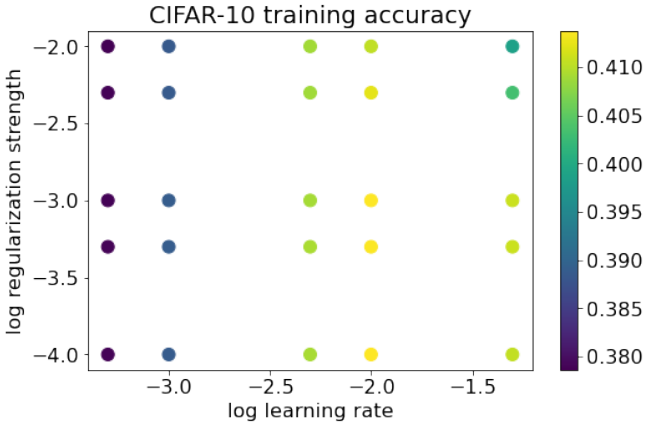

Fig. 3. SVM Training Accuracy with SVM Loss.

Table V and Table VI show which learning rate and regularization strengths are selected during SVM training

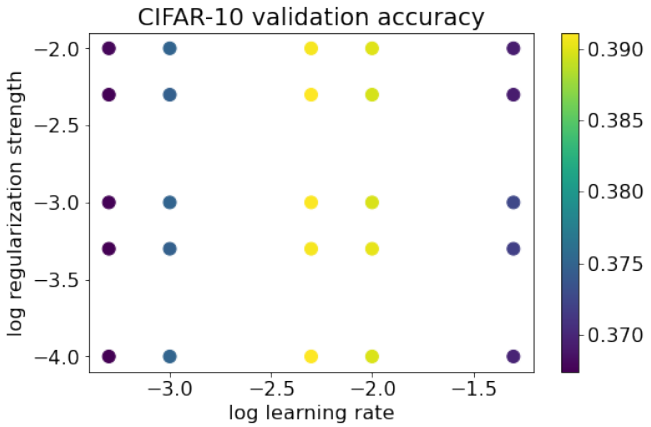

Fig. 4. SVM Validation Accuracy with SVM Loss.

Fig. 5. Linear SVM Classifier Model Learned Weight Visualization for Each Class in CIFAR10 Dataset.

epochs. Specifically, Table VI shows which hyperparameter resulted in which training and validation accuracy.

The CIFAR10 dataset SVM training and validation accuracy results could be visualized in Figure 3 and Figure 4, respectively.

TABLE VI
TRAINING AND VALIDATION ACCURACY OF SVM TRAINING WITH DIFFERENT HYPERPARAMETERS.

Hyperparameters		Accuracy	
Learning Rate	**Regularization**	**Train**	**Validation**
5.0×10^{-4}	1.0×10^{-4}	0.378900	0.367400
5.0×10^{-4}	5.0×10^{-4}	0.378875	0.367400
5.0×10^{-4}	1.0×10^{-3}	0.378775	0.367500
5.0×10^{-4}	5.0×10^{-3}	0.378825	0.367500
5.0×10^{-4}	1.0×10^{-2}	0.378600	0.367800
1.0×10^{-3}	1.0×10^{-4}	0.388650	0.375000
1.0×10^{-3}	5.0×10^{-4}	0.388825	0.374700
1.0×10^{-3}	1.0×10^{-3}	0.388825	0.374900
1.0×10^{-3}	5.0×10^{-3}	0.388500	0.374600
1.0×10^{-3}	1.0×10^{-2}	0.388450	0.374900
5.0×10^{-3}	1.0×10^{-4}	0.409200	0.391100
5.0×10^{-3}	5.0×10^{-4}	0.409300	0.390800
5.0×10^{-3}	1.0×10^{-3}	0.409100	0.391000
5.0×10^{-3}	5.0×10^{-3}	0.408750	0.391100
5.0×10^{-3}	1.0×10^{-2}	0.408875	0.390800
1.0×10^{-2}	1.0×10^{-4}	0.413725	0.389800
1.0×10^{-2}	5.0×10^{-4}	0.413575	0.390200

With the hyperparameter tuning, we have obtained the CIFAR10 test set accuracy of **39.28%** with linear SVM classifier implementation. The chosen hyperparameters are:

- learning rate : 5.0×10^{-3}
- regularization : 1.0×10^{-4}

We can visualize the SVM learned weights for each class as shown in Figure 5.

B. Linear Softmax Classifier

In section, we have aimed to implement a Softmax classifier and optimize its loss function using stochastic gradient descent (SGD). Similar to the Support Vector Machine (SVM) task, we have started by implementing a fully-vectorized loss function for the Softmax classifier and then its analytic gradient expression. We have also checked our implementation using a numerical gradient and tuned the learning rate and regularization strength using a validation set. We have optimized the loss function with SGD and visualized the final learned weights. To begin with, we have implemented the naive Softmax loss function with nested loops in the softmax-loss-naive function.

For the sake of sanity-checking, whether we have implemented the code correctly or not, we have run the Softmax classifier with a small random weight matrix and no regularization. It is expected to see the results near $log(10) = 2.3$. We have obtained a Softmax loss of **2.302797** and sanity check of **2.302585**. These two values are correct; so problem with the implementation.

We have performed numeric gradient checking on the gradients of our Softmax loss. We have observed relative errors less than 1e-5 as expected in Table VII.

TABLE VII
NUMERIC AND ANALYTIC GRADIENTS WITH RELATIVE ERROR WITH NAIVE SOFTMAX LOSS IMPLEMENTATION.

Numerical	Analytic	Relative Error
0.003046	0.003046	5.045984e-07
0.006309	0.006309	2.663255e-08
0.005390	0.005390	1.917530e-07
0.002580	0.002580	4.473513e-07
0.007512	0.007512	7.381182e-07
0.006417	0.006417	9.004100e-08
0.011390	0.011390	2.041099e-07
0.001821	0.001821	1.447045e-07
-0.014710	-0.014710	3.854999e-07
-0.005154	-0.005154	9.740901e-07

We have implemented softmax-loss-vectorized function which calculates Softmax loss in a vectorized format without using any explicit loops. It is expected that the speed and the performance between non-vectorized and vectorized Softmax loss implementations should be in a range of 15-120x speed-up with vectorized calculations. We have calculated Softmax naive loss as **2.302812** in **29.171705 s** and Softmax vectorized loss as **2.302812** in **2.064705 s**. These results show that we have achieved the speed-up of **14.13X** in vectorized format compared to naive implementation.

After turning on the regularization for naive Softmax loss implementation, the results are shown in Table VIII. It is expected that the relative error will be below $1e-5$. As shown in Table VIII, we have achieved that expectation.

TABLE VIII
NUMERIC AND ANALYTIC GRADIENTS WITH RELATIVE ERROR WITH NAIVE SOFTMAX LOSS IMPLEMENTATION WITH REGULARIZATION.

Numerical	Analytic	Relative Error
0.004915	0.004915	2.079737e-07
0.005887	0.005887	4.988083e-08
0.006309	0.006309	2.699530e-07
0.001580	0.001580	1.033739e-06
0.005839	0.005839	6.162031e-07
0.006800	0.006800	5.541226e-07
0.011466	0.011466	3.165576e-07
0.002314	0.002314	1.239312e-06
-0.016812	-0.016812	1.622449e-07
-0.006673	-0.006673	3.033818e-07

After finishing implementations of Softmax loss and gradient calculation functions, we conducted training with vectorized Softmax implementation code on the CIFAR10 dataset. The results of this training are shown in Table IX.

TABLE IX
SOFTMAX CLASSIFIER TRAINING LOSS.

Iteration	Loss
0 / 1500	2.303355
100 / 1500	2.303353
200 / 1500	2.303353
300 / 1500	2.303352
400 / 1500	2.303352
500 / 1500	2.303350
600 / 1500	2.303350
700 / 1500	2.303349
800 / 1500	2.303348
900 / 1500	2.303348
1000 / 1500	2.303347
1100 / 1500	2.303348
1200 / 1500	2.303347
1300 / 1500	2.303345
1400 / 1500	2.303345

Table X shows that training and validation set accuracies. It is expected to see an accuracy of less than 10%.

TABLE X
SOFTMAX CLASSIFIER TRAINING ACCURACY RESULTS ON CIFAR10.

Training Accuracy	Validation Accuracy
8.97%	8.69%

To further improve the model, we used the validation set to tune the hyperparameters, which included the regularization strength and learning rate. We experimented with different ranges of values for these parameters to find the best combinations.

To successfully complete the assignment, the best model found through cross-validation should have achieved an accuracy of at least 37% on the validation set. Fortunately, we were able to beat this goal with our best model achieving over 39.8% validation accuracy with **40.28%**.

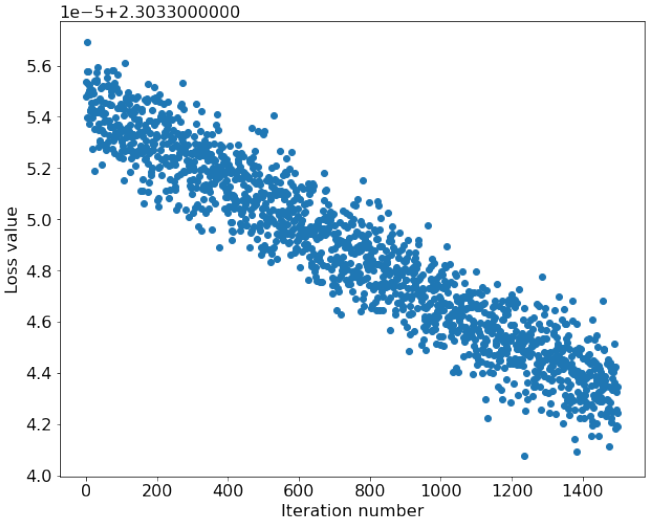

Fig. 6. Softmax Training Iteration Loss.

We have obtained this result with cross-validation hyperparameter tuning. For the hyperparameters, we have selected:

1) learning rates: 1e-5, 1e-4, 1e-3, 1e-2, 1e-1
2) regularization strengths: 1e-3, 1e-2, 1e-1, 1e0, 1e1

TABLE XI
TRAINING AND VALIDATION ACCURACY OF SOFTMAX TRAINING WITH DIFFERENT HYPERPARAMETERS.

Hyperparameters		Accuracy	
Learning Rate	**Regularization**	**Train**	**Validation**
1×10^{-5}	1×10^{-3}	0.2474	0.2489
1×10^{-5}	1×10^{-2}	0.2474	0.2489
1×10^{-5}	1×10^{-1}	0.2474	0.2489
1×10^{-5}	1	0.2475	0.2489
1×10^{-5}	10	0.2475	0.2487
1×10^{-4}	1×10^{-3}	0.2655	0.2657
1×10^{-4}	1×10^{-2}	0.2655	0.2657
1×10^{-4}	1×10^{-1}	0.2653	0.2656
1×10^{-4}	1	0.2640	0.2639
1×10^{-4}	10	0.2533	0.2537
1×10^{-3}	1×10^{-3}	0.3432	0.3364
1×10^{-3}	1×10^{-2}	0.3426	0.3362
1×10^{-3}	1×10^{-1}	0.3379	0.3322
1×10^{-3}	1	0.2961	0.2962
1×10^{-3}	10	0.2498	0.2507
1×10^{-2}	1×10^{-3}	0.4067	0.3903
1×10^{-2}	1×10^{-2}	0.4037	0.3855
1×10^{-2}	1×10^{-1}	0.3695	0.3598
1×10^{-2}	1	0.2935	0.2915
1×10^{-2}	10	0.2495	0.2472
1×10^{-1}	1×10^{-3}	0.4325	0.4028
1×10^{-1}	1×10^{-2}	0.4101	0.3907
1×10^{-1}	1×10^{-1}	0.3618	0.3535
1×10^{-1}	1	0.290425	0.2886
1×10^{-1}	1×10^{1}	0.0524	0.0519

With the hyperparameter tuning, we have obtained the CIFAR10 test set accuracy of **40.14%** with linear Softmax classifier implementation. The chosen hyperparameters are:

- learning rate : 1.0×10^{-1}
- regularization : 1.0×10^{-3}

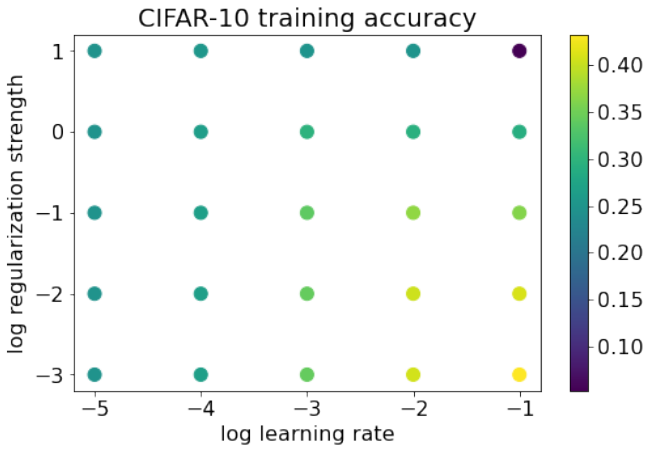

Fig. 7. Linear Softmax Classifier Training Accuracy with Softmax Loss.

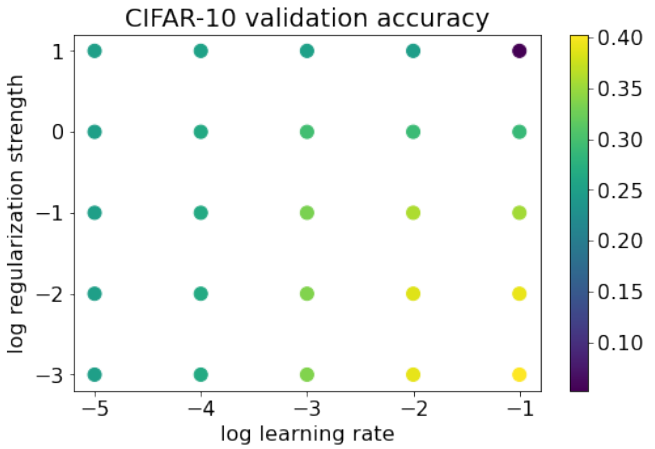

Fig. 8. Linear Softmax Classifier Validation Accuracy with Softmax Loss.

We can visualize the Linear Softmax classifier learned weights for each class as shown in Figure 9.

IV. Accomplishments

The general list of accomplished tasks is enumerated below:

1) Implemented both SVM and Softmax linear classifier and tested on the CIFAR10 dataset.
2) Successfully trained the linear classifiers on the CIFAR10 training data.
3) Conducted hyperparameter tuning with cross-validation implementations.
4) Achieved better scores than minimum scores suggested.
5) Implemented all coding material with comments.
6) No procrastination is done.

V. Conclusion

In this project, we have implemented a fully-vectorized loss function for the SVM and Softmax classifiers. We have also derived the fully-vectorized expression for the analytic gradient and validated our implementations using numerical gradient checking.

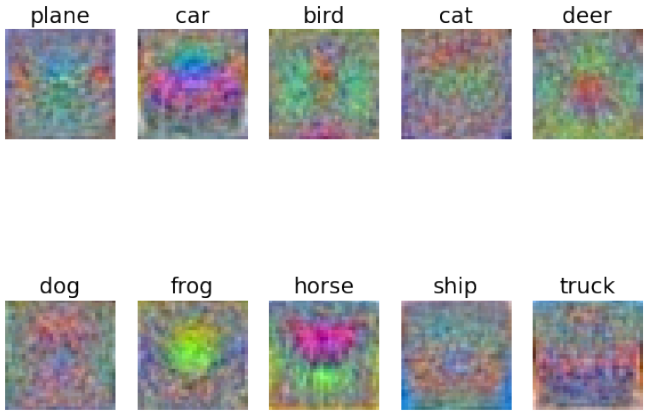

Fig. 9. Linear Softmax Classifier Model Learned Weight Visualization for Each Class in CIFAR10 Dataset.

We have used a validation set to tune the hyperparameters, including the learning rate and regularization strength, for both classifiers. We have optimized the loss functions using SGD, and we have visualized the final learned weights.

Overall, this project has provided us with a good understanding of how to implement and optimize linear classifiers such as SVM and Softmax. We have learned how to fine-tune the hyperparameters using a validation set and how to visualize the learned weights. These skills are valuable in the field of machine learning, where linear classifiers are widely used in many applications.

Acknowledgement

This work is conducted as an assignment of the "BLG 506E Computer Vision" graduate course lecture given by Prof. Hazım Kemal Ekenel.

References

[1] F. Perronnin, J. Sánchez, and T. Mensink, "Improving the fisher kernel for large-scale image classification," in *Computer Vision–ECCV 2010: 11th European Conference on Computer Vision, Heraklion, Crete, Greece, September 5-11, 2010, Proceedings, Part IV 11*. Springer, 2010, pp. 143–156.

[2] S. Alshahrani and E. Kapetanios, "Are deep learning approaches suitable for natural language processing?" in *Natural Language Processing and Information Systems: 21st International Conference on Applications of Natural Language to Information Systems, NLDB 2016, Salford, UK, June 22-24, 2016, Proceedings 21*. Springer, 2016, pp. 343–349.

[3] X. Chen, J. Tian, J. Cheng, and X. Yang, "Segmentation of fingerprint images using linear classifier," *EURASIP Journal on Advances in Signal Processing*, vol. 2004, pp. 1–15, 2004.

[4] S. I. Gallant *et al.*, "Perceptron-based learning algorithms," *IEEE Transactions on neural networks*, vol. 1, no. 2, pp. 179–191, 1990.

[5] D. G. Kleinbaum, K. Dietz, M. Gail, M. Klein, and M. Klein, *Logistic regression*. Springer, 2002.

[6] M. A. Hearst, S. T. Dumais, E. Osuna, J. Platt, and B. Scholkopf, "Support vector machines," *IEEE Intelligent Systems and their applications*, vol. 13, no. 4, pp. 18–28, 1998.

[7] Y. Abouelnaga, O. S. Ali, H. Rady, and M. Moustafa, "Cifar-10: Knn-based ensemble of classifiers," in *2016 International Conference on Computational Science and Computational Intelligence (CSCI)*. IEEE, 2016, pp. 1192–1195.

[8] S. Mannor, D. Peleg, and R. Rubinstein, "The cross entropy method for classification," in *Proceedings of the 22nd international conference on Machine learning*, 2005, pp. 561–568.