BLG 506E COMPUTER VISION

ASSIGNMENT 2

Linear Classifier

Student No: 504221511

Name & Surname: Enis Teper

1. **Download assignment 1 from Stanford's CS231n: http://cs231n.github.io/assignments2019/assignment1/ You will be following svm.ipynb and softmax.ipynb notebooks.**

Since I have downloaded the code base from previous homework, I did not download it again. The code base eliminates the workload of writing code from scratch for required algorithms. It just leaves empty most important cases to let us improve ourselves.

2. **Download and extract CIFAR-10 dataset using get_datasets.sh script under cs231n/datasets . (If you downloaded the dataset for the first assignment, you can use symbolic links or arrange paths to save disk space.)**

Because of I have downloaded the dataset from previous homework I did not download it again. The dataset consists 60,000 32x32x3 colored RGB images of 10 classes. In image classification the more image the model generalizes the classes. Because of this as much as high data should be used. Of course, balance of the classes, some image processing problems like background clutter, deformation or intraclass variation situations should be considered.

3. **Start notebook with jupyter-notebook command. In svm.ipynb, run library/parameter setups, load CIFAR10 dataset, visualize it, split data as training-validation-test set and do preprocessing.**

CIFAR-10 dataset is loaded thanks to pre-defined functions. Training data size is 50,000 and test data size is 10,000. It is divided as 49,000 image to training 1,000 to validation and 1,000 to test and 500 to development.

In order to center the data mean of all train images are subtracted from all images. Additionally, bias column is inserted to let the SVM function focus on only weights.

4. **Implement svm_loss_naive function in cs231n/classifiers/linear_svm.py . Run gradient check cell and answer inline question 1.**

In linear_svm.py svm_loss_naive function is implemented by changing upper for loop to eliminate the workload of running 2 for loop for twice. In for loop, each score is calculated by multiplying the input with weights. To calculate loss, svm loss function which is $\sum_{j \neq i} \boldsymbol{max}(\boldsymbol{0}, \boldsymbol{s_j} - \boldsymbol{s_i} + \boldsymbol{1})$ is used. The function subtracts current score from correct class score when the j is not equal to i. Thus, if the calculated margin is above zero it is added to total loss and gradients are updated with respect to input.

Finally, loss is scaled by dividing it to train number and loss from the regularization loss which is $\lambda * \sum_k \sum_l W_{k,l}^2$ is added to total loss. Similar to loss gradient vector is scaled by train number and the loss obtained from derivation of L2 regularization formula $2 * \lambda * W_{k,l}$ is added for each relevant gradient.

From gradient check we can see that our analytical gradient calculation results are very similar to our numerical gradient calculation. Since numerical gradient calculation obtains gradients from each point one by one, it is so slow. That is why we use analytical gradient but to check is our results are trustable or not we implement gradcheck to compare them. Since relative errors are too low we can say that our approach is okay.

Inline Question 1 It is possible that once in a while a dimension in the gradcheck will not match exactly. What could such a discrepancy be caused by? Is it a reason for concern? What is a simple example in one dimension where a gradient check could fail? How would change the margin affect of the frequency of this happening? *Hint: the SVM loss function is not strictly speaking differentiable*

Yes, gradcheck may not match exactly in some specific situations. It is mainly caused by the loss function of $max(0, L_i)$ since the function is not strictly differentiable. Here L_i represents the difference of scores of correct classes and incorrect classes. When L_i is 0 at the $max(0, L_i)$ it fails to gradient check at that point. Yet, because of the gradient calculation of remaining rows optimization of the model may still work properly. Changing margin may not eliminate the problem because the reason of this situation is caused by the loss function of $max(0, L_i)$.

5. Implement svm_loss_vectorized function. Run gradient check cell and compare the results (loss difference and times) to naive approach.

Even though analytical gradient approach is much more fast than the numerical gradient calculation it can be still more fast by using vectorized loss function of svm. With this approach, instead of calculating scores one by on with respect to train number all scores at obtained at once.

Thanks to numpy library it enables us to multiply weights with multiple inputs, so by multiplying development (train) data (500, 3073) with weights (3073, 10) we obtain scores for each class with size of (500, 10). Then correct class scores are obtained from the (500, 10) result array as (500, 1). Reason of obtaining correct class scores to calculating the gradients to calculate the total loss and gradients. At following step margin is calculated with respect to relevant rows and correct class scores. Then, correct class score losses are assigned 0 to eliminate additional loss which is already not meant to be calculated. To finalize total loss, it is scaled and regularization result is added to it. With the purpose of calculating gradients, a temporary result matrix is defined. With this matrix, all svm loss function results that above zero is assigned as 1 and correct classes are assigned as 0. As a final step, sum of ones are subtracted from correct class at each row. Gradients are calculated by multiplying transpose of input matrix which is (3073, 500) with temporary result matrix (500, 10). Lastly, gradient vector is scaled by development (train) number and loss from derivative of L2 loss is added.

6. **Implement LinearClassifier.train function in cs231n/classifiers/linear_classifier.py . Train your svm and plot your loss. Implement LinearClassifier.predict function to measure the accuracy.**

As TODO suggests to use np.random.choice it is used with respect to input and batch size. Then, losses are calculated thanks to our vectorized svm loss function. Thanks to gradient descent loss decreases at each step. However, when the model reaches its maximum performance loss will not be able to decrease at some point. In our training it seems its struggling to decrease loss below 5. As can be seen from the plot which is given below at Figure 1, our loss significantly decreases with respect to our initial loss. This means our model much more generalizes the data and more successful.

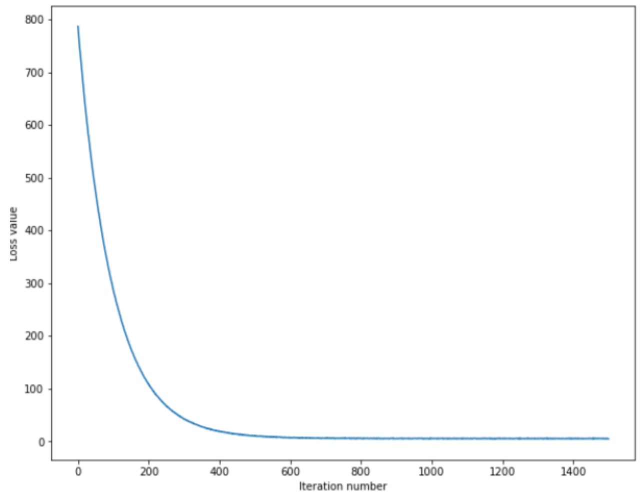

Figure 1: Loss Plot

Accuracy is a metric that shows how successful is our model. It checks how many of our predictions are true and it is calculated by $\frac{True\ Predictions}{Total\ Predictions}$. To calculate our predictions inputs are multiplied with weights and highest scored column of each row is obtained as predicted class.

7. Tune hyperparameters using validation set. Determine the best ones.

Hyperparameter tuning is too important to optimize our model. However, it requires lot of computational power and time to try each combination of hyperparameters. Even though, there are algorithms to tune our hyperparameters in efficient way we are using naive approach here. For this reason, I have implemented 4 different learning rates and 4 different regularization parameters since my computational power is not so high. My best validation accuracy was 0.391 with learning rate of 1e-7 and regularization parameter as 1e+4 it is visualized at **Figure 2**. By searching similar values to this learning rate and regularization parameter we may obtain better results.

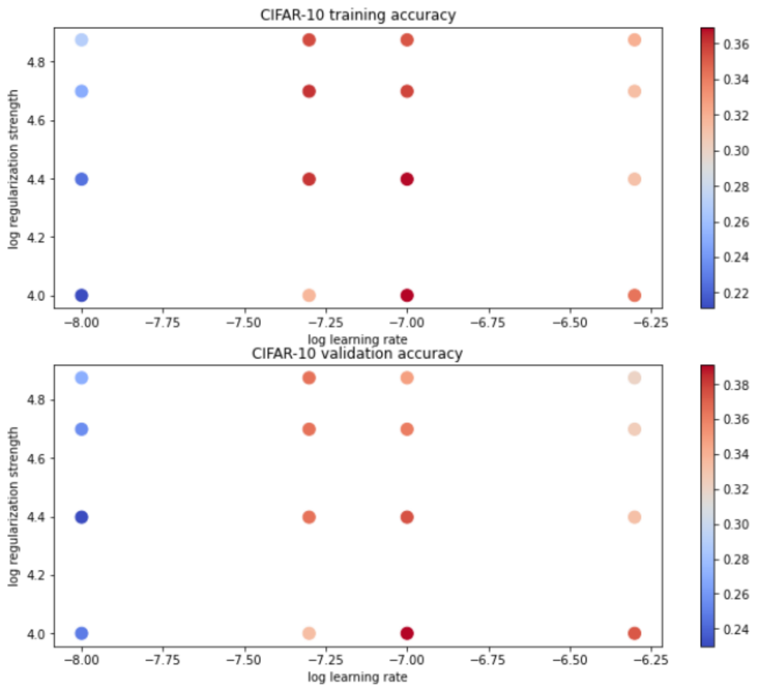

Figure 2: Hyperparameter Tuning Results

8. Visualize the learned weights. (Answer inline question 2)

Learned weights are visualized by reshaping as 32x32x3 and shown at Figure 3. It can be understanded from the images that weights consist templates of classes. However, because of we have trained our model with 500 images we cannot understand clearly that weights consists templates of classes.

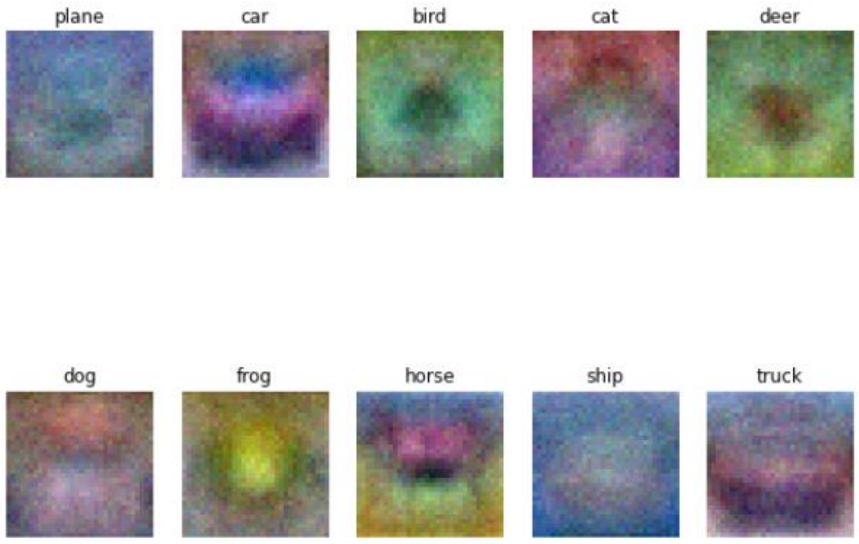

Figure 3: Weight Visualization

Inline question 2: Describe what your visualized SVM weights look like, and offer a brief explanation for why they look the way they do.

Visualized SVM weights represent templates of classes. These templates are optimized by iterating data and updating weights via gradient descent. Each template is stored as an column at weights of the SVM. From these templates most similar template to input gets the highest score because of the inner product between input image and the tempate. As can be seen, especially plane and ship has blue background which are representation of sky and sea. This means weights not only tries to match objects but also their backgrounds. However, these templates may vary by increasing data set or using subsets from the data.

9. Open softmax.ipynb . Load data.
Data is loaded and preprocesses that explained at 3rd question is done again.

10. Implement softmax_loss_naive function in cs231n/classifiers/softmax.py . Run gradient check cells and answer inline question 1.

Similar to SVM loss weights are multiplied with inputs and then to calculate the loss and gradients softmax loss functions which is $L_i = -\log(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}})$ is used. To implement this function exponentials of each value is calculated and correct class score represents the numerator and sum of all represent denominator. Thus, class probabilities are obtained. From these probabilities loss is calculated for each input. To calculate gradients -1 is applied to correct class to penalize it. Lastly, transpose of input is multiplied with result probabilities to calculate remaining gradients. Finally loss is scaled and loss from L2 regularization is added to total loss. Similarly, gradients are scaled with respect to training number and derivative of L2 regularization is added to gradients.

Similar to question 4 gradient check is done to check our analytical gradient approach is trustable or not by comparing it with numerical gradient calculation.

Inline Question 1 Why do we expect our loss to be close to -log(0.1)? Explain briefly.

Because of weights are initialized as uniform, all classes should have similar probability at start. Since the dataset has 10 classes the probability of each class should be 10% with initial weights which means $10 / 100 = 0.1$. Since our loss function is $L_i = -\log(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}})$ the result inside of the log should be equal to 0.1 because of uniformity. For this reason we assume that our loss close to be -log (0.1)

11. Implement softmax_loss_vectorized function. Run gradient check cell and compare the results (loss difference and times) to naive approach.

In vectorized version of softmax loss function inputs are multiplied with weights and (500, 10) array is obtained. Each row represents input and each column represents a class. Then exponential of each value is calculated for denominator. Afterwards, class probabilities are calculated of each row by division of correct class value to total value of that row. Then, loss is calculated by implementing minus log to these probabilities. Finally, it is scaled and with the addition of L2 Regularization loss total loss is calculated. For gradient calculation 1 is subtracted from correct classes for backpropagation and multiplying transpose of inputs with class probabilities gradients are calculated. At final, gradients are scaled and derivative of L2 Regularization is added to gradients.

12. Train your softmax classifier and tune hyperparameters to find the best ones. Answer inline question 2.

Since I explained hyperparameter tuning at question 7 I am not going to explain again to not to repeat myself. Same hyperparameters which are define for SVM is defined at here too. Thus we can compare results how hyperparameters behave on different losses. By training Softmax classifier our best accuracy was 0.324 which is lower than the SVM classifier. The reason may be the Softmax may require more data or higher learning rate. Because it obtained best result with 5e-7 which was highest learning rate. Also, regularization parameter for best validation accuracy was 5e4. Because of this regularization parameter was lower than 7.5e4 and higher than 2.5e4 we may assume it is okay for now. We may increase learning rate and get better results.

****Inline Question 2** - *True or False* Suppose the overall training loss is defined as the sum of the per-datapoint loss over all training examples. It is possible to add a new datapoint to a training set that would leave the SVM loss unchanged, but this is not the case with the Softmax classifier loss.**

Your Answer: True

Your Explanation: Since the SVM loss function is $max(0, s_j - s_{y_i} + 1)$ if this $s_{y_i} > s_j + 1$ condition is satisfied for all data SVM loss will not change. If the training loss of recent model is zero it may still continue as zero with new dataset. For this reason, SVM loss may remain unchanged with new dataset. To explain it more clearly, assume that we have scores of [2, 11, 9, 8, 5] and assuming 11 is the correct class each loss value will be zero of this input. The reason is $11 > 2 + 1, 11 > 9 + 1, 11 > 8 + 1, 11 > 5 + 1$ all conditions are satisfied so each loss will be 0. To implement more visualization to show why these losses will be zero is $max(0, 2 - 11 + 1), max(0, 9 - 11 + 1), max(0, 8 - 11 + 1), max(0, 5 - 11 + 1)$ so 0 will be maximum value for each implementation.

However, because of the loss function of Softmax which is $L_i = -\log(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}})$, it is not possible to obtain same loss of different datasets. The reason is, Softmax will have probabilities for each class and this probability is different for each input. Even though, all losses close to 0.001 it is will not be the same because of floating numbers. Assuming same input [2, 11, 9, 8, 5] if we want to find the loss we should get exponential of each value and sum them and exponential of 11 which is correct class will be divided to sum. Finally $-log()$ will be applied to calculate final loss. To visualize exponentials of each value is $[7.38905610e+00, 5.98741417e+04, 8.10308393e+03, 2.98095799e+03, 1.48413159e+02]$ and result loss is 0.17203930312807225. Let's assume our input array was $[2, 11, 9, 7, 5]$ for SVM loss it would not change anything but for Softmax loss our new loss is 0.14518467717594596. Thus, we can say that SVM may remain same with same loss but Softmax may not.

13. Visualize the learned weights.

Similar to question 8 weights are reshaped to 32x32x3 images and visualized. These weights represents templates of each class. However, because of training is done with low data size it cannot visualize clearly class templates.

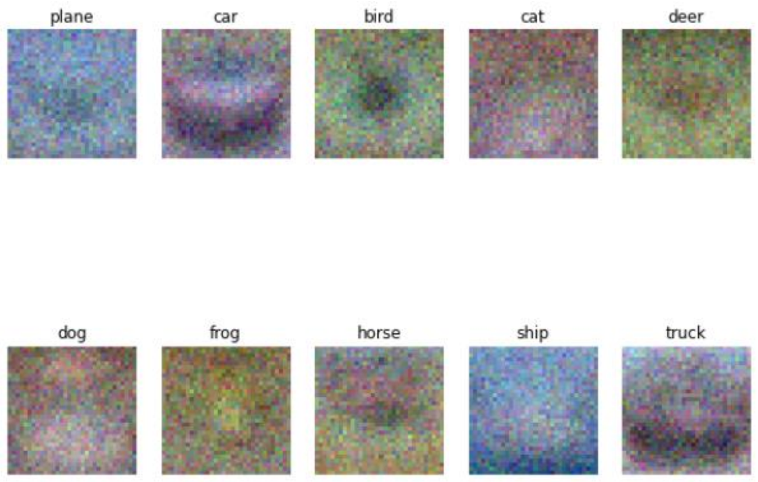

Figure 4: Visualized Weights of Softmax Classifier