# Istanbul Technical University- Spring 2017
# BLG 506E COMPUTER VISION
# Homework 2
# Linear Classifier

Aydin Ayanzadeh
student number: 504161503
ayanzadeh17@itu.edu.tr
a.ayanzadeh@gmail.com

March 25, 2018

**Q1:**

Linear classifier is a simple but not strong method that can be used for image classification. Its not strong because as can be inferred from it's name it can do well only if the fed dataset be linearly separable. What linear classifier does is getting input datapoints (X) and multiplying it with a randomly generated weight matrix (W), result is a matrix that contains scores of different classes for all datapoints. Highest class score for a datapoint means that the datapoint is belong to that particular class. But because in first iteration we fill the weight matrix with randomly generated numbers ,results wont be promising. So, we need to somehow change the weight matrix to get better results.
Generally we have two type of loss that are very common, SVM loss and Softmax loss. At this point, linear classifier tries to minimize the loss with finding best wights. This can be done using back propagation which means finding gradient of weights. Strictly speaking, these gradients show effect of each element of W matrix on loss function. So, we use gradient to update the wights. Iteratively doing these steps will provide us final weight matrix which can be used to classify new input data. Two loss functions, SVM and Softmax will be presented in this paragraph. What SVM loss function dose simply is insuring that correct class score is higher than the scores of incorrect classes by a constant delta value. For this aim, loss of each input datapoint is being calculated using following formula:

**Multiclass Support Vector Machine loss (SVM)** is zero when the result of score function for correct class of an image has a score higher than all other classes by a given margin $\Delta$. The equation 1 calculate the loss for ith image in training set. Total loss is the average of the loss of all images in the training set.

$$L_i = \sum_{j \neq y_i} max(0, s_j - s_{y_i} + \Delta) \tag{1}$$

In **Softmax classifier** scores are considered as the unnormalized log probabilities of classes which are normalized using equation 2.

$$P(y_i|x_i; W) = \frac{e^{f_{y_i}}}{\Sigma_j e^{f_j}} \tag{2}$$

Where $f_{y_i}$ is the score for correct class and $f_j$ is the score for $jth$ class. **cross-entropy loss** of image i is calculated using equation 3. Similar to **SVM**, Total loss is calculated by averaging the loss of all images in train dataset. After training, classifier predict the label of test image as the label for class with the highest probability.

$$L_i = -log\left(\frac{e^{f_{y_i}}}{\Sigma_j e^{f_j}}\right) \tag{3}$$

In order to have $W$ parameters that would generalize well we must add a regularization term to loss function to penalize large values of parameters and is in favor of more diffused weights. In other words, the best set of parameters are those which take into account all part of the images. Thus, we have equation 4 as total loss.

$$L = \underbrace{\frac{1}{N}\Sigma_i L_i}_{\text{data loss}} + \underbrace{\lambda R(W)}_{\text{regularization loss}} \tag{4}$$

Where $\lambda$ is a hyper-parameter, namely, regularization strength, which is tunned by cross-validation.
Assume we have scores $[45, 40, 12]$ and the correct class is the first one.
Calculating loss using **SVM** and $\Delta = 1$:

$$L = max(0, 40 - 45 + 1) + max(0, 12 - 45 + 1) = 0$$

Calculating loss using **cross-entroppy**:

$$L = -log(\frac{e^{45}}{e^{45}+e^{40}+e^{12}}) = 0.0067$$

As you can see **SVM** loss is zero if the score of correct class is greater than other classes by a given margin, however **Softmax** loss is never zero because loss always is getting better and correct class probability ($p_1 = 0.9933$) is increasing and incorrect class probabilities ($p_2 = 0.006692$, $p_3 = 4 \times 10^{-15}$) are decreasing.
**5.1)**
Partial derivative of equation 1 with respect to row of $W$ related to correct class is shown in equation 5.

$$\nabla_{w_{y_i}} L_i = -\left(\sum_{j \neq y_i} 1(w_j^T x_i - w_{y_i}^T x_i + \Delta > 0)\right) x_i \tag{5}$$

And with respect to other classes is shown in equation 7.

$$\nabla_{w_j} L_i = 1(w_j^T x_i - w_{y_i}^T x_i + \Delta > 0) x_i \tag{6}$$

**In the naive Method** of SVM we have traverse over all training datasets one by one and calculate their score for that datapoint by multiplying pixel values of the datapoint with the weight of given matrix. For calculating the margins of data point we have to employ another for this task. IN another word, In terms of naive approach, what it does is iterating over all training datapoints one by one and calculate the score for that datapoint by multiplying pixel values of the datapoint with weight matrix. Then, code has another for loop which iterates over scores of the datapoint and calculates the margin. If margin be greater than zero for aclass, margin will be added to the loss. Also for margins grater than zero, according to formula of calculating analytical gradient, pixel values of current datapoint will be added to corresponding column of the class -which second for loop is iterating over it- and will be subtracted from column of current datapoints class in gradient matrix. After iterating over all datapoints well have a loss which is a sum over all training examples and also dw which is sum over all training examples too. So by dividing the loss and dw by number of trains we get average loss and gradient. Finally code adds the regularization term to loss where regularization parameter is 0.5.

**In the vectorized version** of SVM loss: is so faster than naive approach due to lack of using for loops.

About the margin due to vectorized implementation of the code, we do not need to have allocate new value for margin so we save all of the steps in the Scores. we have add margin to over loss when our margin is positive(otherwise, we do not do anything). At the end, we

divide the loss and dW by number of training and add the regularization term derivative to get the total loss.
1. We calculate the scores of classes for all the training samples using a dot product of X and W.
2. Specify the correct class scores.
3. Calculate the margins.
4.for preventing the adding the delta=1 to the loss, we allocate zero for correct classes of margin.
Sum up positive margins,Then adding the multiplication of transpose of$X_s uband\ y_m ask$ gives us the gradient matrix which is sum over all training examples. Final gradient matrix is calculated by dividing all of elements of dw by number of training examples and adding the regularization term.

In calculating the gradient matrix we again use $margin_p ositive_i dx$. First it multiple the X matrix with $positive_i dx$ and get a dw which needs some modification. Then it calculates a $sub_m ask$ matrix by adding all of positive margins along axis one. That is used to implement following formula:

$$\nabla_{w_j} L_i = 1(w_j^T x_i - w_{y_i}^T x_i + \Delta > 0)x_i \tag{7}$$

Then adding the multiplication of transpose of X$_s ubandy_m askgivesusthegradientmatrixwhichissumov$
and transpose of X, and sum the result to partial derivatives matrix, divide it by number of training and add regularization to have derivatives matrix.

**5.2**
1. We randomly choose a mini-batch of images. 2. For training, inside a for loops we calculate the loss and gradients, save the loss history and update the weights using gradient and learning rate until. We specify the number of iteration. 3. To predict, after learning the weights, we dot product the test image with wights to calculate the score functions related to each class. Afterwards, we assign the label corresponding to the class with highest score. After writing the LinearSVM.predict function and evaluate the performance on both the training and validation set we will reach the 0.364653 accuracy on the training and 0.381000 validation set.

we have some Hyper-parameters that can be very effectve in increasing the validation accuracy.(Number of iteration assumed 3000), Here, after tuning for finding the best regularization strqenght and learning rate. (lr 1.000000e-07 reg 2.375000e+04) which achieved **0.393000 percent** validation accuracy.

After cross-validation we can visualize the learned weights. As you can see in Figure 3, each row of learned weights matrices look like a template for corresponding class.

After cross-validation we have the final model which is used to predict the class of test images. Using the learned model we managed to achieve **0.3750 accuracy on test set**.

**6. Softmax implementation(Additional part:)** If we want to create a timeline of

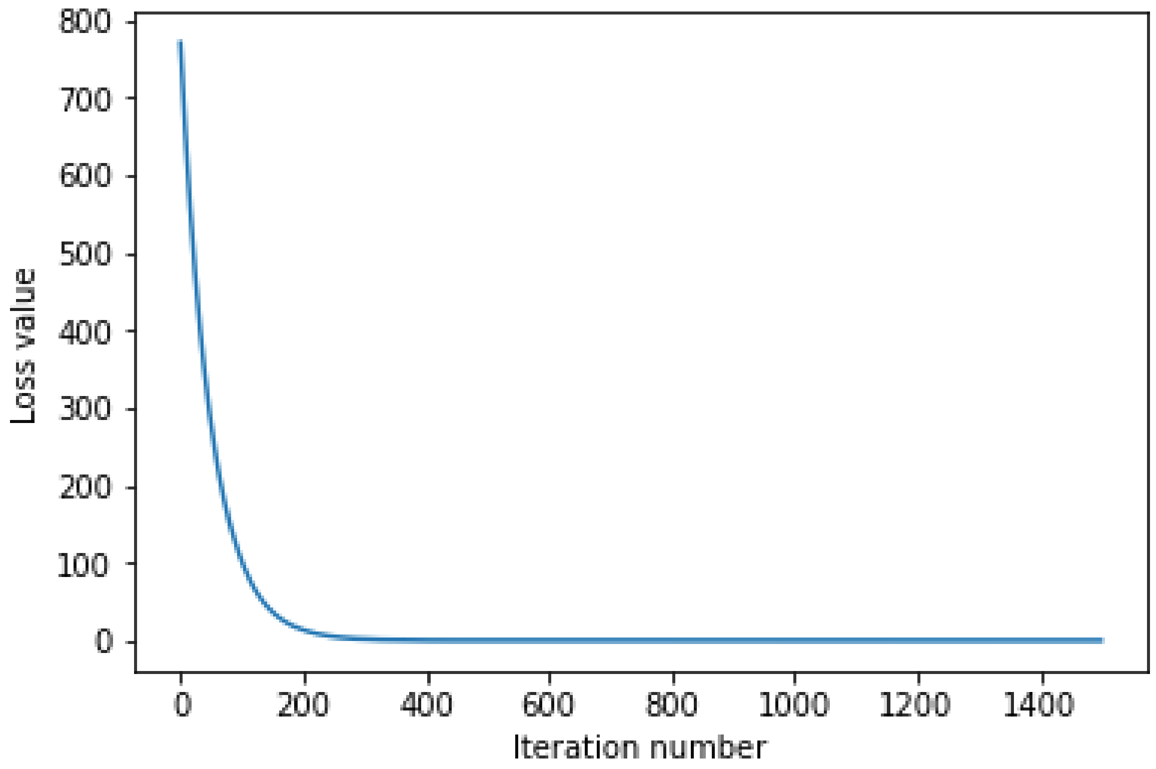

Figure 2: Result of tuning learning rate and regularization strength.

implementation for this part we should doing the following task:

1.implement a fully-vectorized loss function for the Softmax classifier
2.check your implementation with numerical gradient
3.tune the learning rate and regularization strength
4.optimize the loss function with SGD
5. visualization of the final learned weights

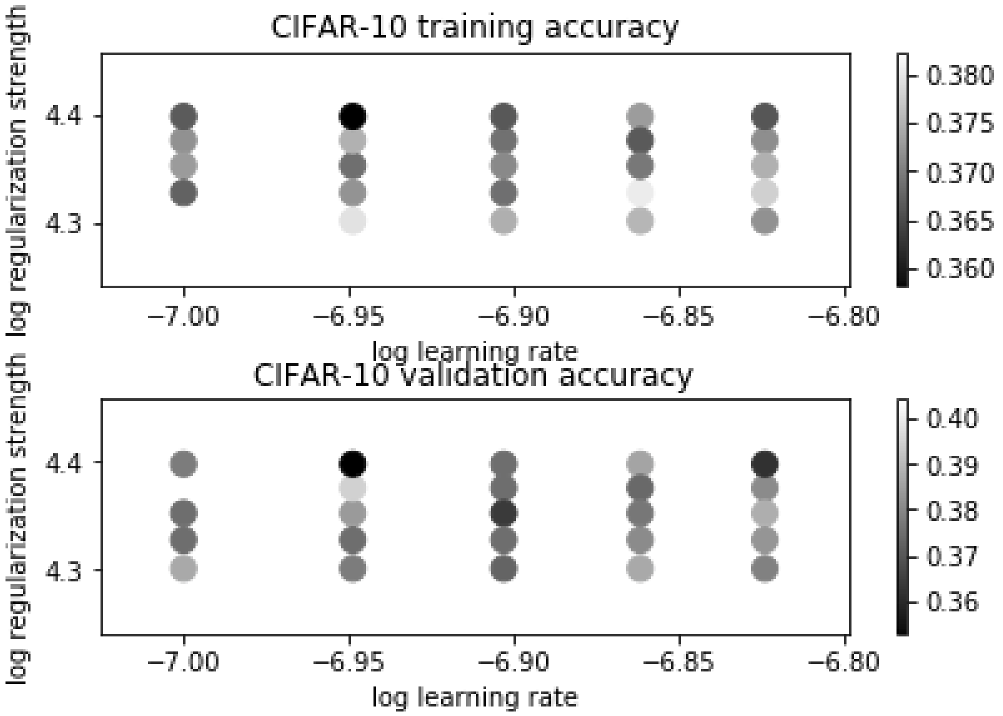

Figure 3: Result of tuning learning rate and regularization strength.

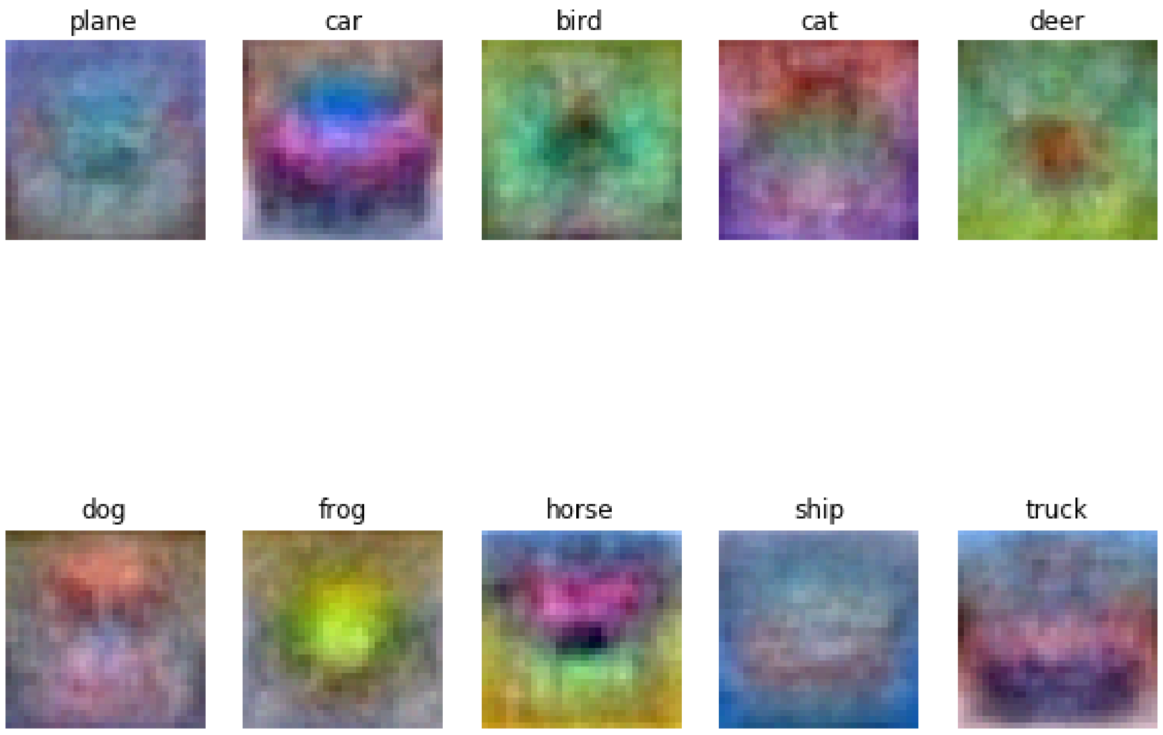

Figure 4: Visualization of learned weights for each class, Actually each of these are like a template for each class.
Inline Question2 Answer: This are the visualization of the learned weight for svm. Actually this are the average of the sum of the weight along samples in each class of the datasets. Actually these are the template for each claass, For example, if you analyze the template of the Frog you will be see that it has a one head with grish color actually this is a clue that the number of the frog with green-yellow skin are more common among the datasets.

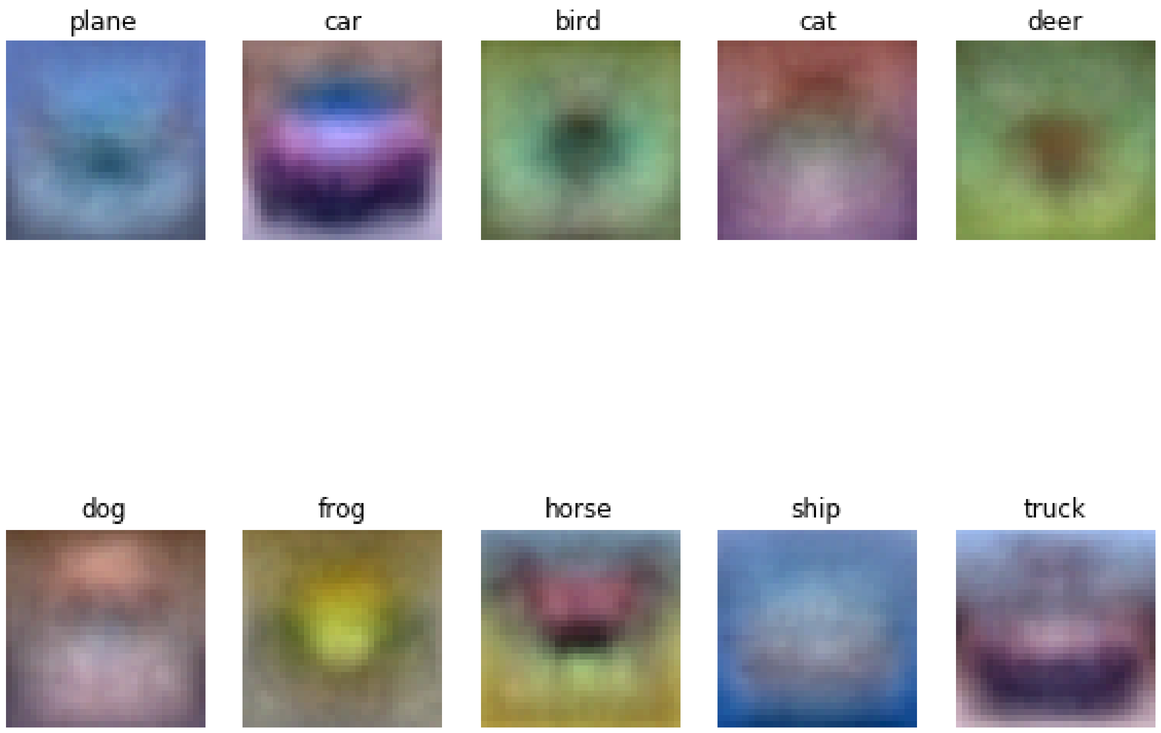

Figure 5: Visualization of learned weights for each class, Actually each of these are like a template for each class.

It turns out that performance of SVM and Softmax are not very different but because SVM only cares about satisfying the margin. Also SVM is not sensitive to details of individual scores. On the other hand Softmax cares about details of indevidual scores and can show the exact probability of incorrect classes. Furthermore, no mater which classifier we are using we add a regularization term to the final loss to better generalization.