Assignment 1 – Optional Part

1. Improvements

As is suggested in the instruction, the matlab code in naïve version is modified using the following tricks respectively:

1) Use data_batch_1.mat to Use data_batch_5.mat as training data, and decrease the size of validation set to 1000.

2) Train for a longer time and use the validation set to find the best point of epoch to stop training the network (early stopping).

3) Decay the learning rate by a factor 0.97 after each epoch.

4) Augment the training data by adding a random jitter to each image in the mini-batch before doing the forward and backward pass.

5) Apply Bootstrap Aggregating (Bagging) to ensemble diverse high-variance, low-bias classifiers.

6) Apply Adaboost algorithm to ensemble diverse low-variance, high-bias classifiers.

The test performances of the above improved version methods are evaluated respectively. It appears that method 1), 3) and 5) can all bring out a significant improvement while the others generate a minor one. Method 1) and 3) could enhance test accuracy by about 0.7%, and 5) can guarantee a around 38.5% accuracy. Thus, applying Bagging could bring the largest gains from my experimental results.

Method 6) is a sequential ensemble learning algorithm but it doesn't result in a good performance as expected due to the limitation of training data size. It is quite computationally demanding when the training data is expanded into 20000 (I will try it again after I get my PDC account and password. Maybe it will behave better with more training data).

2. SVM Multi-class Loss

Instead of using the cross-entropy loss, the SVM multi-class loss function is applied in this case, which is shown below:

$$l_{SVM} = \frac{1}{N}\sum_{i}\sum_{j\neq y_i}\left[\max\left(0, f(x_i;W)_j - f(x_i;W)_{y_i} + \Delta\right)\right] + lambda \cdot \sum_{k}\sum_{l} W_{k,l}^2$$

Here, N is the number of training examples, $lambda$ controls the amount of regularization, and Δ denotes the margin of SVM. To train the network, we need to calculate the gradient of the loss function with respect to the matrix W. Note here we take a bias to represent the two parameters W, b in the naïve implementation as one. The training data will be expanded into the dimension of $3073\times N$ (add a row of constant 1 at last) and W will become 10×3073in this case.

For a single training sample (x_i, y_i), the analytical gradient with respect to weight w_j is:

$$\frac{\partial l_i}{\partial w_j} = \begin{cases} -\left(\sum_{j\neq y_i} ind\left(w_j^T x_i - w_{y_i}^T x_i + \Delta > 0\right)\right)\cdot x_i, j = y_i \\ ind\left(w_j^T x_i - w_{y_i}^T x_i + \Delta > 0\right)\cdot x_i, j \neq y_i \end{cases}$$

According to the equation above, we can rewrite it in the form of matrix. The corresponding matlab code is shown below:

```
function grad_W = ComputeGradients(X, Y, s, W, delta, lambda)
% DENOTE d as the dimensionality of each image, N as the number of images
%        K as the number of label kinds
% INPUT     - X:                (d+1)*N
%           - Y:                K*N
%           - s:                K*N
%           - W:                K*(d+1)
%           - delta:            1*1
%           - lambda:           1*1
% OUTPUT    - grad_W:           K*(d+1)

grad_W = zeros(size(W));
sc = repmat(sum(s.*Y), size(s, 1), 1);
margin = s - sc + delta;

% flag represents if margin is above 0
flag = zeros(size(s));
flag(find(margin > 0)) = 1;
flag(find(Y == 1)) = -1;

for i = 1 : size(X, 2)
    Xi = X(:, i);
    fi = flag(:, i);

    % obtain gradient matrix for each training sample
    gi = repmat(Xi', size(W, 1), 1);
    gi(find(fi == 0), :) = 0;
    gi(find(fi == -1), :) = -length(find(fi == 1))*gi(find(fi == -1), :);

    % accumulate gradients
    grad_W = grad_W + gi;
end

% add regularization term
grad_W = 2*lambda*W + grad_W/size(X, 2);

end
```

After the gradient is computed analytically, I run several experiments and the results with different parameter settings are presented below:

1) lambda = 0, n_epochs = 100, n_batch = 100, eta = 0.001

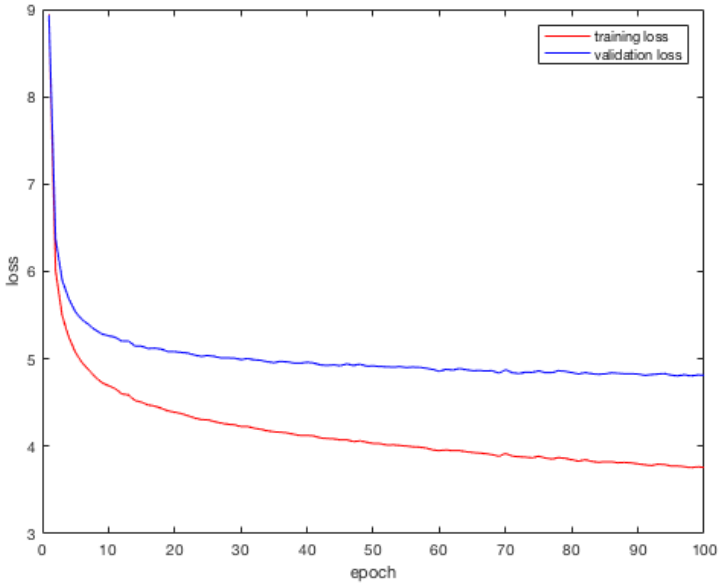

Fig. 1: Total loss function on the training data and validation data after each epoch of the mini-batch gradient descent algorithm (lambda = 0, n_epochs = 100, n_batch = 100, eta = 0.001)

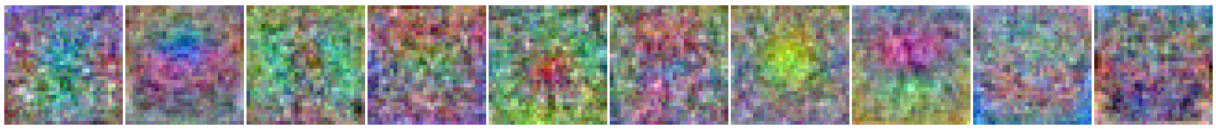

Fig. 2: The learnt W matrix visualized as class template images (lambda = 0, n_epochs = 100, n_batch = 100, eta = 0.001)

2) lambda = 0, n_epochs = 100, n_batch = 100, eta = 0.01

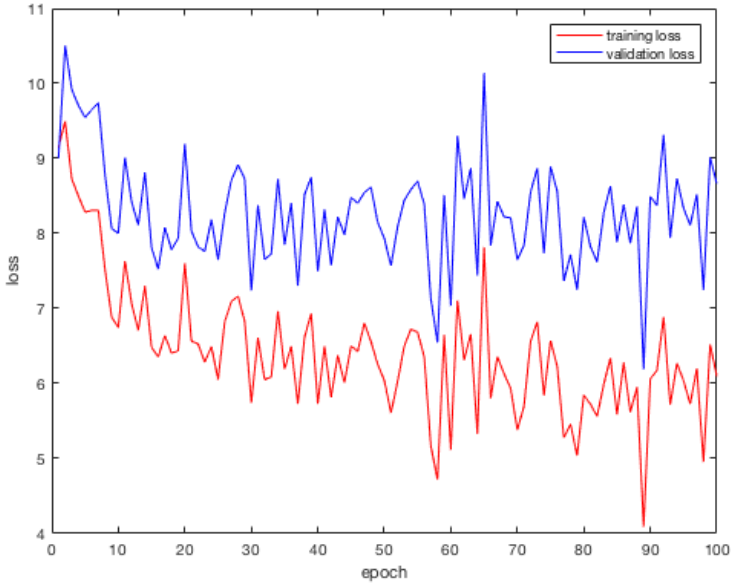

Fig. 3: Total loss function on the training data and validation data after each epoch of the mini-batch gradient descent algorithm (lambda = 0, n_epochs = 100, n_batch = 100, eta = 0.01)

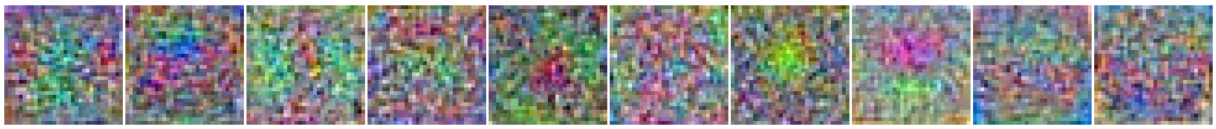

Fig. 4: The learnt W matrix visualized as class template images (lambda = 0, n_epochs = 100, n_batch = 100, eta = 0.01)

3) lambda = 0.1, n_epochs = 100, n_batch = 100, eta = 0.001

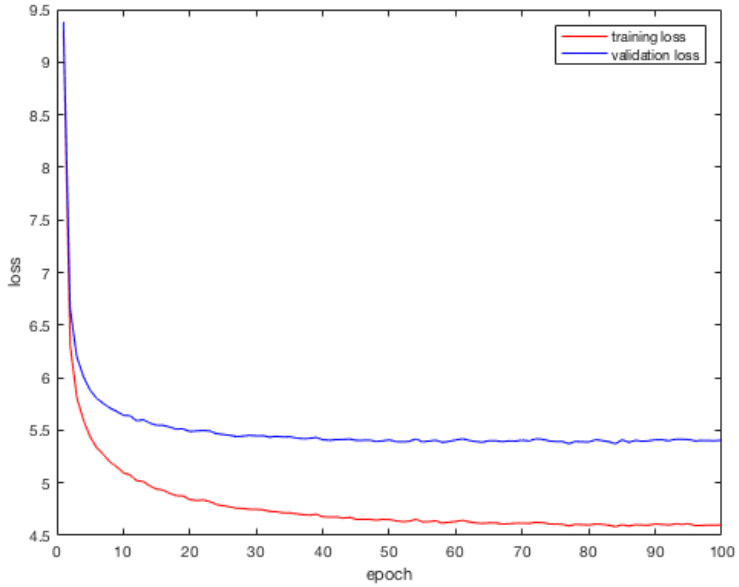

Fig. 5: Total loss function on the training data and validation data after each epoch of the mini-batch gradient descent algorithm (lambda = 0.1, n_epochs = 100, n_batch = 100, eta = 0.001)

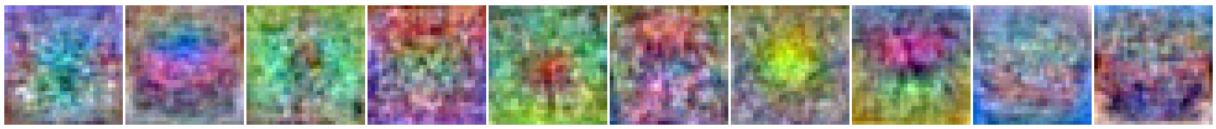

Fig. 6: The learnt W matrix visualized as class template images (lambda = 0.1, n_epochs = 100, n_batch = 100, eta = 0.001)

4) lambda = 1, n_epochs = 100, n_batch = 100, eta = 0.001

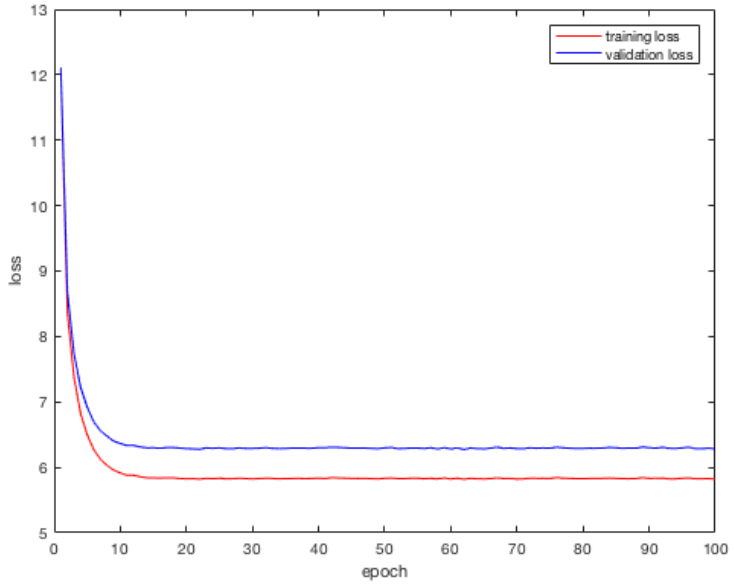

Fig. 7: Total loss function on the training data and validation data after each epoch of the mini-batch gradient descent algorithm (lambda = 1, n_epochs = 100, n_batch = 100, eta = 0.001)

Fig. 8: The learnt W matrix visualized as class template images (lambda = 1, n_epochs = 100, n_batch = 100, eta = 0.001)

Tab. 1: Training and test accuracy for different parameter settings

lambda	n_epochs	n_batch	eta	training accuracy (%)	test accuracy (%)
0	100	100	0.001	42.52	36.19
0	100	100	0.01	29.64	24.71
0.1	100	100	0.001	39.15	35.07
1	100	100	0.001	32.71	31.86

The results above show that using SVM loss function doesn't improve the test accuracy compared to using the cross-entropy loss. And the conclusions about the learning rate and the amount of regularization should also hold in this case. Besides, obvious fluctuations still appear after some epochs, which accounts for a much smaller learning rate here.

3. Best Solution

It seems that some methods in section 1 can bump up performance, and I obtain my best prediction on test set by using an integration of these tricks. After several runs trying various methods and parameter settings, my best solution for classifying dataset CIFAR-10 is using Bootstrap Aggregating (Bagging) to generate 5 bootstrap replicates, each with a size of 10000, from the training set [batch_1, batch_2, batch_3, batch_4, batch_5]. And then use these bootstrap replicates to train the one-layer network. The final classifier is a weighted combination of the five classifiers with diverse initializations

(weight is given by the normalized test accuracy of each classifier). Since Bagging is a parallel ensemble learning for low-bias, high-variance classifiers, I could actually ignore early stopping and allow slight overfitting for these base classifiers in a practical sense. And about the learning rate, I choose to assign decayed initial values for the five base classifiers. As epoch increases, it is however better not to change the learning rate for each base classifier based on the experimental results. As for the loss function, I choose softmax + cross entropy loss instead of the SVM multi-class loss. The parameter setting is: lambda = 0, n_epochs = 100, n_batch = 100, eta = 0.01, decay = 0.97.

Best test accuracy: 40.42%

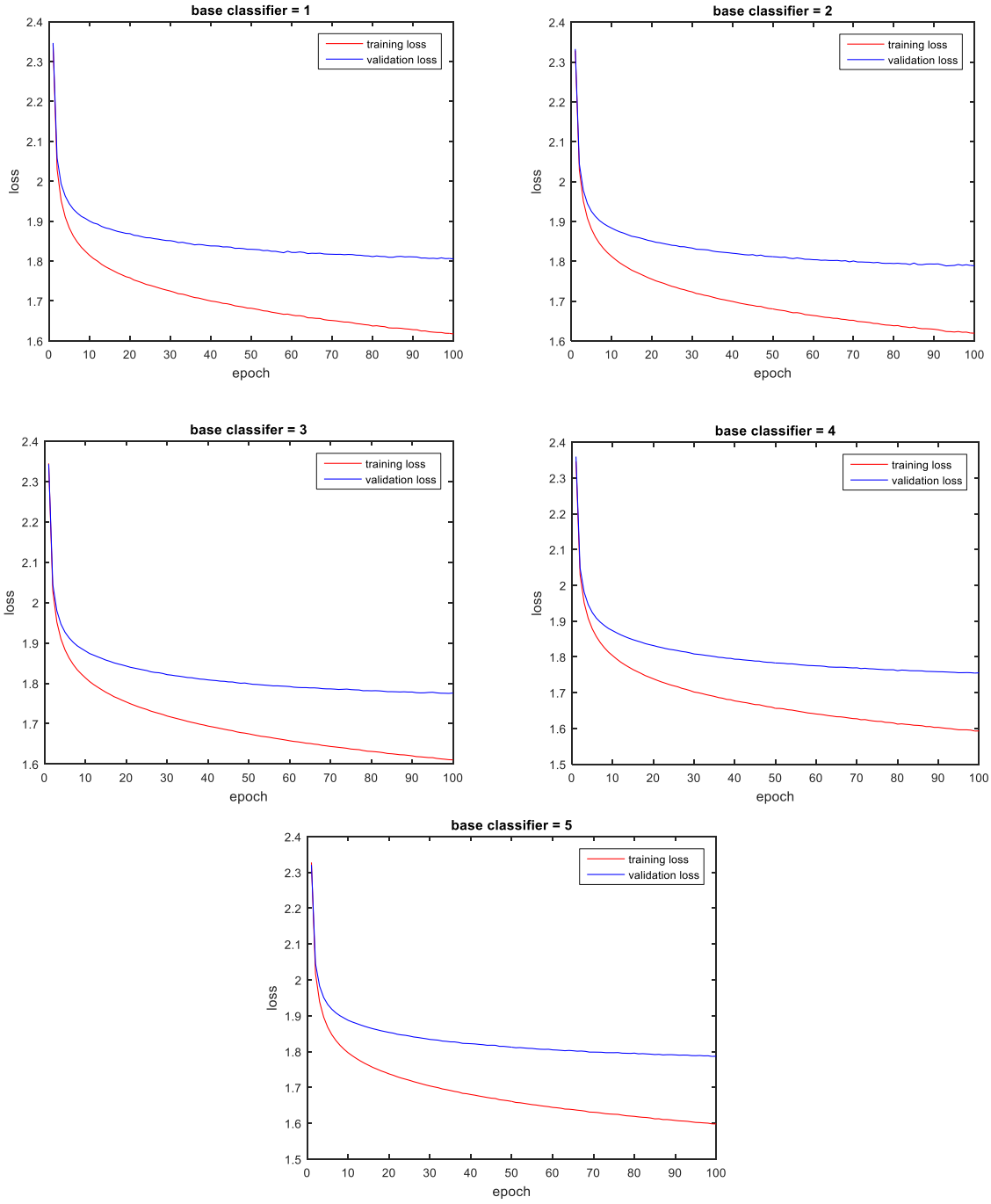

Fig. 9: Total loss function on the training data and validation data after each epoch of the mini-batch gradient descent algorithm for 5 diverse base classifiers