DD2424 Deep Learning in Data Science

Name: Sri Janani Rengarajan *Programme: Embedded Systems*

Assignment 1: Training and testing of a one layer network with multiple outputs

The aim of the assignment is to classify images into ten classes using a single layer network. Input to the classifier is the vector **x,** which contains the information about the image. Output of the classifier is a probability vector **p**, for each possible class. The parameters of the classifier are **W** and **b.** These parameters are learned by minimizing the cross-entropy loss of the classifier. The method that is implemented to learn these parameters is called **mini batch gradient descent**.

Steps involved in implementing mini batch gradient descent method:

- Implementation of the classifier
- Computation of gradients for the implemented classifier

1. Classifier Implementation:
   The classifier is implemented as follows:

```
function P = EvaluateClassifier(X,W,b)
 s = W*X + b;
 exp_s = exp(s);
 sum_s = sum(exp_s);
 P = exp_s ./ sum_s;
end
```

2. Gradient computation:
   Two methods where used for gradient computation, they are as follows
   Method 1:

```
function [grad_W, grad_b] = ComputeGradients(X, Y, P, W, lambda)
 N = size(X,2);
 I = ones(N,1);
 G = -(Y-P);
 grad_W = ((G*X')/N) + (2*lambda*W);
 grad_b = (G*I)/N;
end
```

   Method 2:

```
function [grad_W, grad_b] = ComputeGradientsl(X, Y, P, W, lambda)
grad_W = zeros(size(W));
grad_b = zeros(size(W, 1), 1);

for j = 1 : size(X, 2)
    Pj = P(:, j);
    Yj = Y(:, j);
    Xj = X(:, j);
    g = -Yj'*(diag(Pj) - Pj*Pj')/(Yj'*Pj);
    grad_b = grad_b + g';
    grad_W = grad_W + g'*Xj';
end
grad_b = grad_b/size(X, 2);
grad_W = 2*lambda*W + grad_W/size(X, 2);
end
```

Both the methods were compared with `ComputeGradsNumSlow()` for different values of Lambda (regularization term) and mini batch size. The results are presented in Table 1.

Error computation:

```
P = EvaluateClassifier(Xt(:,100), W, b);
[ngW,ngb] = ComputeGradsNumSlow(Xt(:,100), Yt(:,100),W, b, lambda, h);
[agW,agb] = ComputeGradients(Xt(:,100), Yt(:,100), P, W, lambda);
[agWl,agbl] = ComputeGradientsl(Xt(:,100), Yt(:,100), P, W, lambda);
error_b = max(abs(agb-ngb)./max(0,abs(agb)+abs(ngb)));
error_W = max(max(max(abs(agW-ngW))./max(0,abs(agW)+abs(ngW))));
error_Wl = max(max(max(abs(agWl-ngW))./max(0,abs(agWl)+abs(ngW))));
error_bl = max(abs(agbl-ngb)./max(0,abs(agbl)+abs(ngb)));
diff_b = max(abs(agb-agbl));
[diff_W, ~] = max(max(abs(agW-agWl)));
```

Lambda, Mini batch size	Method-1 Error grad_b	Method-2 Error grad_b	Method- 1 Error grad_W	Method-2 Error grad_W
0 , 1	1.2007e-09	1.2007e-09	1.5097e-06	1.5097e-06
0,100	5.5433e-10	5.5433e-10	5.4911e-07	5.4911e-07
0.1,100	1.8479e-09	1.8479e-09	5.7392e-06	5.7392e-06
1,100	2.3439e-09	2.3439e-09	2.7486e-05	2.7486e-05

Table 1. Gradient computation error between analytical and numerical methods

Lambda, mini batch size	Difference between Method 1&2 in grad_W computation	Difference between Method 1&2 in grad_b computation
0 , 1	4.4408e-16	1.1102e-16
0,100	1.1102e-16	2.7756e-17
0.1,100	1.1102e-16	2.7756e-17
1,100	1.1102e-16	2.7756e-17

Table 2. Gradient computation error between the two analytical methods

From tables 1 and 2, it can be understood that the analytical methods perform as good as the numerical method. Of the two analytical methods proposed above, the first one is chosen as it is recommended as an efficient way to compute gradients in Matlab (Ref. Lecture 3).

Results:

Accuracy obtained during training, validation and testing for different values of learning rate (η) and regularization (λ) are presented in table 3.

Lambda	eta	Training Accuracy	Validation Accuracy	Testing accuracy
0	0.1	40.22	27.94	27.23
0	0.001	45.76	38.46	39.08
0.1	0.001	44.62	38.82	39.46
1	0.001	40.03	36.32	37.38

Table 3. Accuracy results for different values of learning rate (η) and regularization (λ)

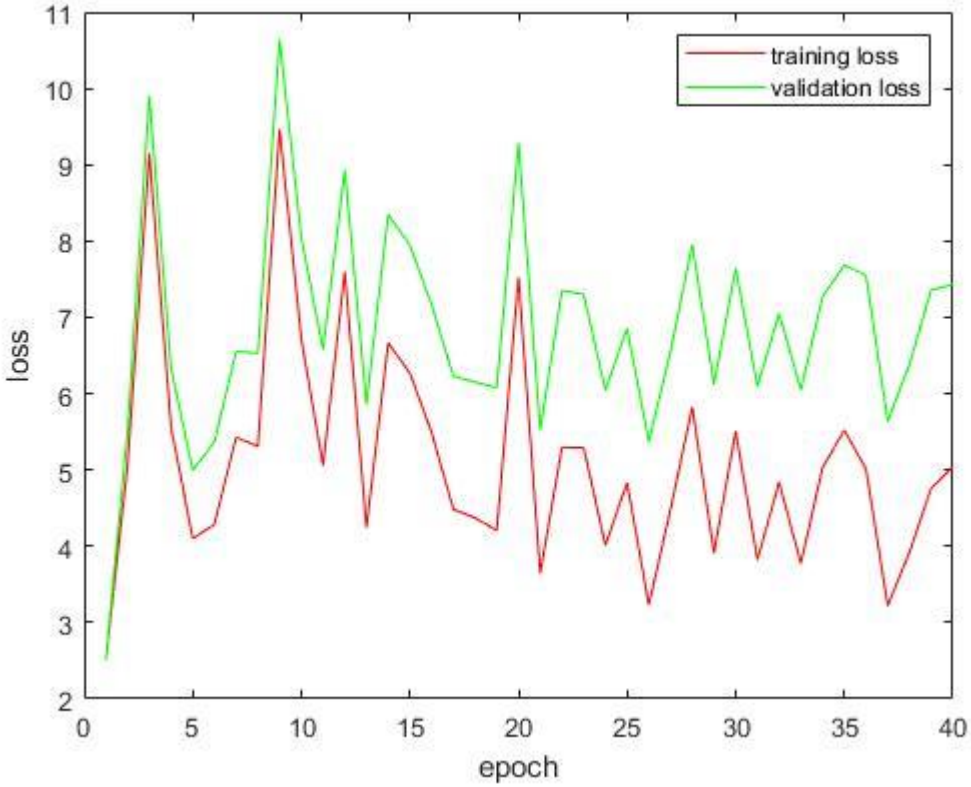

Figure 1. The graph of the training and validation loss computed after every epoch, for the following parameter settings: n_batch=100, eta=0.1, n_epochs=40 and lambda=0.

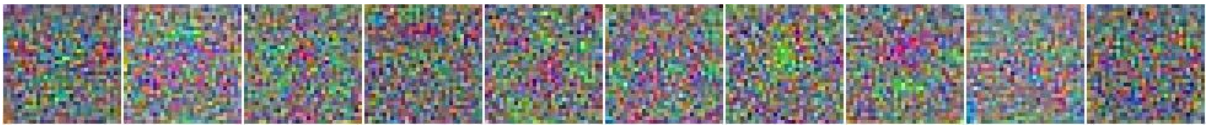

Figure 2. The learnt W matrix visualized as class template images, for the following parameter settings: n_batch=100, eta=0.1, n_epochs=40 and lambda=0.

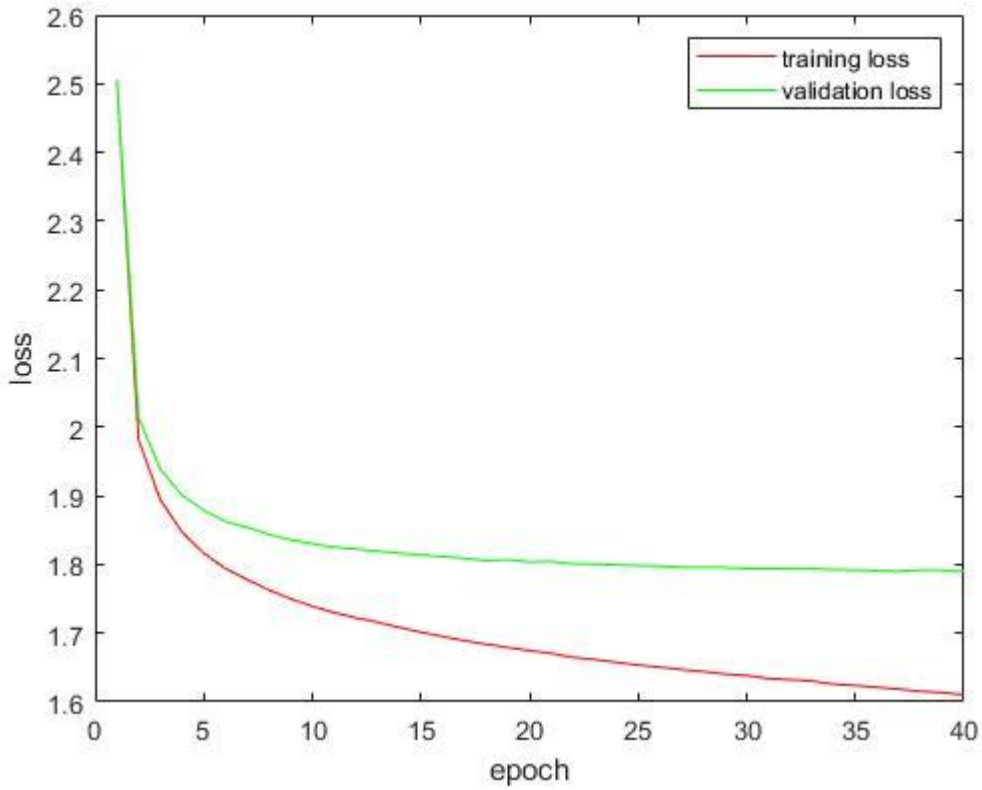

Figure 3. The graph of the training and validation loss computed after every epoch, for the following parameter settings: n_batch=100, eta=0.001, n_epochs=40 and lambda=0.

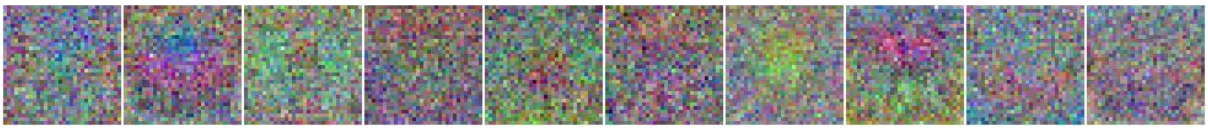

Figure 4. The learnt W matrix visualized as class template images, for the following parameter settings: n_batch=100, eta=0.001, n_epochs=40 and lambda=0.

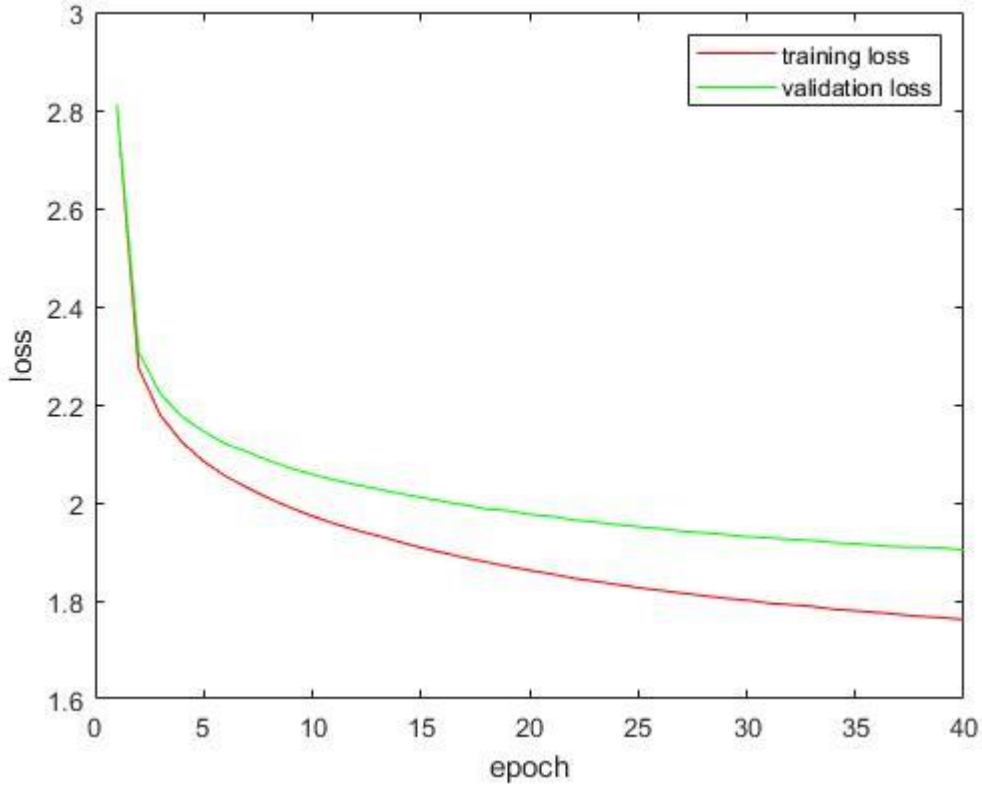

Figure 5. The graph of the training and validation loss computed after every epoch, for the following parameter settings: n_batch=100, eta=0.001, n_epochs=40 and lambda=0.1

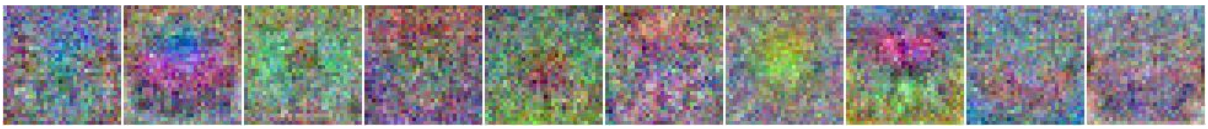

Figure 6. The learnt W matrix visualized as class template images, for the following parameter settings: n_batch=100, eta=0.001, n_epochs=40 and lambda=0.1

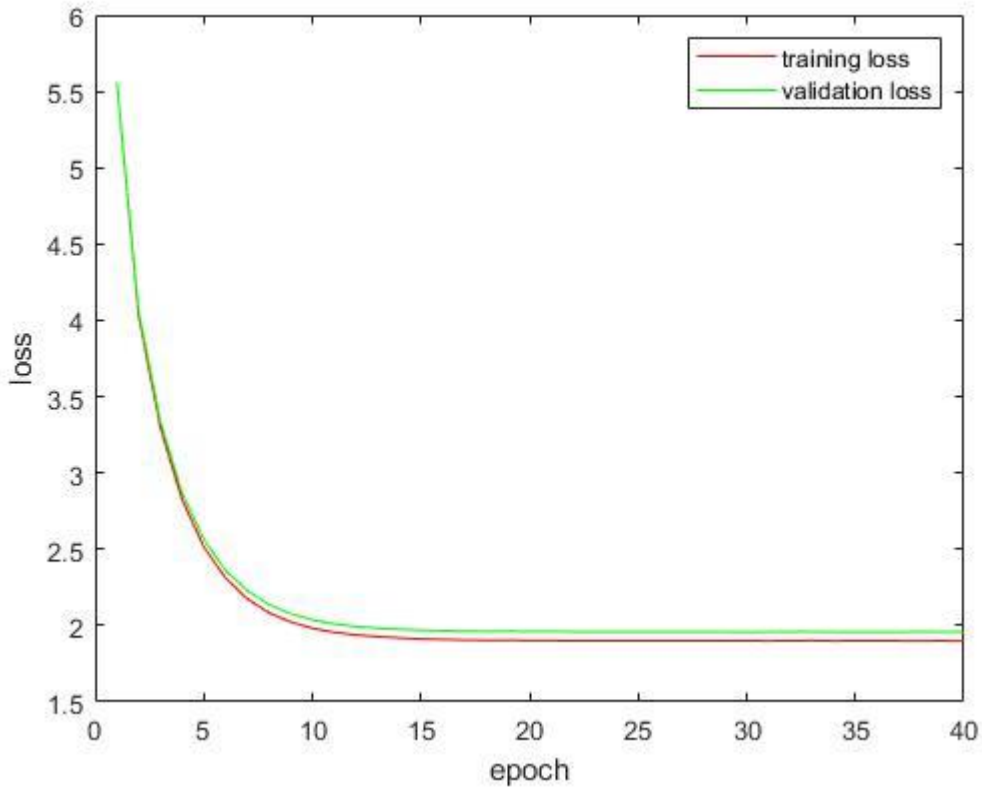

Figure 7. The graph of the training and validation loss computed after every epoch, for the following parameter settings: n_batch=100, eta=0.001, n_epochs=40 and lambda=1

Figure 8. The learnt W matrix visualized as class template images, for the following parameter settings: n_batch=100, eta=0.001, n_epochs=40 and lambda=1

Observations:

1. For a high value of learning rate (η), the convergence of W and b will be accelerated, which will cause the cost function to oscillate between epochs. This results in an unstable convergence.
2. As the regularization (λ) increases, the cost decreases. But a high value will result in a too optimistic cost function, causing the accuracy of the network to reduce.
3. Based on these observations, of the different tested values of λ and η, λ = 0.1 and η = 0.001 have good classification accuracy for the validation and testing phase.
4. For this implementation, the training data was randomly shuffled before each epoch which resulted in a better accuracy.