DD2424 Assignment 3

Wei Wang
wewang@kth.se
Royal Institute of Technology

DD2424 Deep Learning in Data Science

Assignment Report

Introduction

The of object of this assignment is to upgrade the 2-layer networks into k-layer networks with multiple outputs to classify images (once again) from the CIFAR-10 dataset. In order to get better performance, batch normalization has to be incorporated into the k-layer network both for training and testing. After adding batch normalization, parameters searching is also required for getting better outcome.

Exercise

Exercise 1

Upgrade Assignment 2 code to train & test k-layer net-works
In order to successfully test k-layer networks. Here are several subfunctions I have to write:

1. ***[W,b,GDparams]= ParameterInit3(train_X,...,alpha)***
   Initialize almost all necessary parameter and generate W and b according to "layers" and "hidden_notes"

2. ***EvaluateClassifier3(X,W,b,GDparams,varargin)***
   Take the input data X and make the forward pass processing according to the W b. The intermediary score h, S $\hat{S}$ and h are recorded. Then [P,S,S_hat,h,M_bn,V_bn] are returned. (M_bn,V_bn are used for batch normalization)

3. ***ComputeGradients3(X,...,V_bn)***
   Compute the gradient of the cost function for using the gradient equations in the lectures notes 4.The snippet of code of computegradient3 is shown as Listing 1.Then I wrote a short script to compute the differences between analytic and the numerical gradients.As shown in Figure 2. I set the batch to be 20 ($Ttrain\_X = train\_X(:, 1:20); Ttrain\_Y = train\_Y(:, 1:20)$)and δ in ComputeGradsNumSlow to be $e^{-}6$. A 3-layers nets is used for analytic analysis,the hidden nodes of layer-2 is 50 and the hidden nodes of layer-3 is 30. The outcome of diff is shown in Figure 1. we can see that the larger of the layers has less difference with numerical gradients. The former the layer is, the larger the difference seems to be. I also calculated the 2-layers nets gradients. The batch size of data is 20 and the hidden nodes of layer-2 is 50. The outcome is shown is Figure 3. We can also notice that when nets get deeper, the analysis gradient is not so precise.

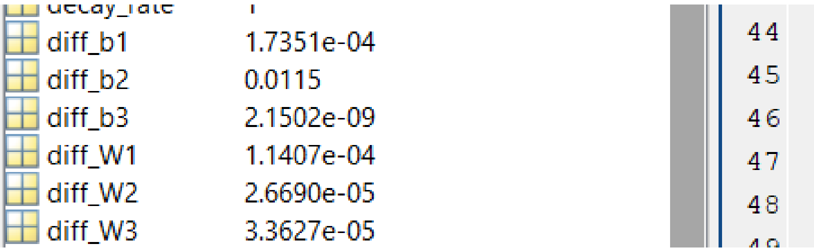

Figure 1: the diff of 3-layer nets without batch normalization

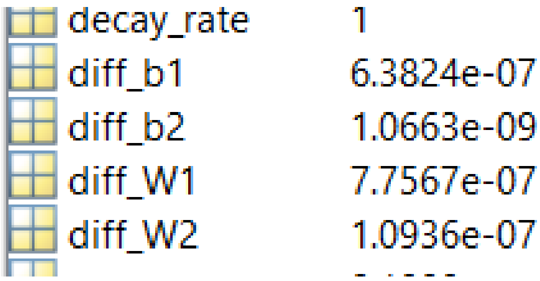

Figure 2: the diff of 2-layer nets without batch normalization

```
    for i = 1:N
        Yn = Y(:,i); % (K x 1)
        Pn = P(:,i); % (K x 1)
        Xn = X(:,i); % (d x 1)
    %   g = - Yn'/(Yn'*Pn)*(diag(Pn)-Pn*Pn');
        g = -(Yn-Pn)';
        % gradient L w.r.t b{L} = g
        for j = (L):-1:2
            grad_b{j} = grad_b{j} +g';
            grad_W{j} = grad_W{j} +g'*h{j-1}(:,i)';
            % update g
            % (1 x m)
            g = g*W{j};
            g = g*diag(S{j-1}(:,i)>0);
        end
        % (1 x m)--> (1 x m x m x m)
        grad_b{1} = grad_b{1} + g';
        grad_W{1} = grad_W{1} + g'*Xn';
    end
    % gradient J
    for i = 1:L
        grad_W{i} = (1/N)*grad_W{i}+2*lambda*W{i};
        grad_b{i} = (1/N)*grad_b{i};
    end
end
```

Listing 1: snippet ofComputeGradients3

4. ***MiniBatchGD3(X, Y,...) and Run3(trainP,validP,GDparams,W,b)***
MiniBatch3 and Run3 are upgraded from assignment 2. GDparams carries

many necessary parameters such as GDparams.IFbn to decide whether use batch normalization or not.

Exercise 2

Can I train a 3-layer network?

2-layers without batch normalization:

Data_batch_1.mat is used for training and test_batch.mat is used for test. Using the parameters in assignment2 by running 20 epochs. The test accuracy is around 0.44 after 20 epochs showing that the upgrading succeed.

```
#################### epoch= 17 #######################
train loss=1.429881 Validation loss = 1.671364
train acc=0.530700  Validation acc = 0.437600
#################### epoch= 18 #######################
train loss=1.419637 Validation loss = 1.670553
train acc=0.533800  Validation acc = 0.438700
#################### epoch= 19 #######################
train loss=1.410002 Validation loss = 1.669741
train acc=0.537300  Validation acc = 0.441700
#################### epoch= 20 #######################
train loss=1.401479 Validation loss = 1.669319
train acc=0.541100  Validation acc = 0.444300
```

Listing 2: 2-layer network without bn running outcome

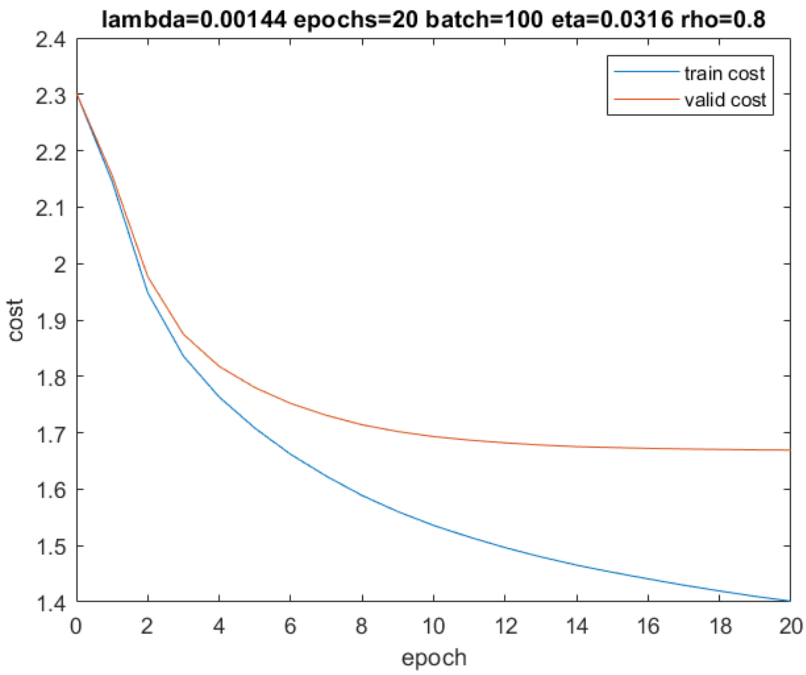

Figure 3: training curve of 2-layers network without bn

3-layers without batch normalization:

When adding 1 more layer to the network, the network seems stop learning using the same parameters as the previous one. When eta is increased, training becomes

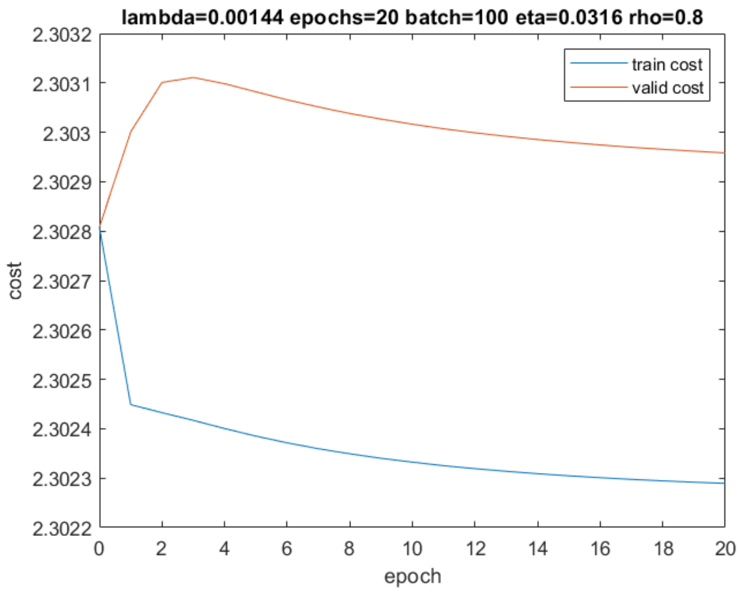

Figure 4: training curve of 3-layers network without bn

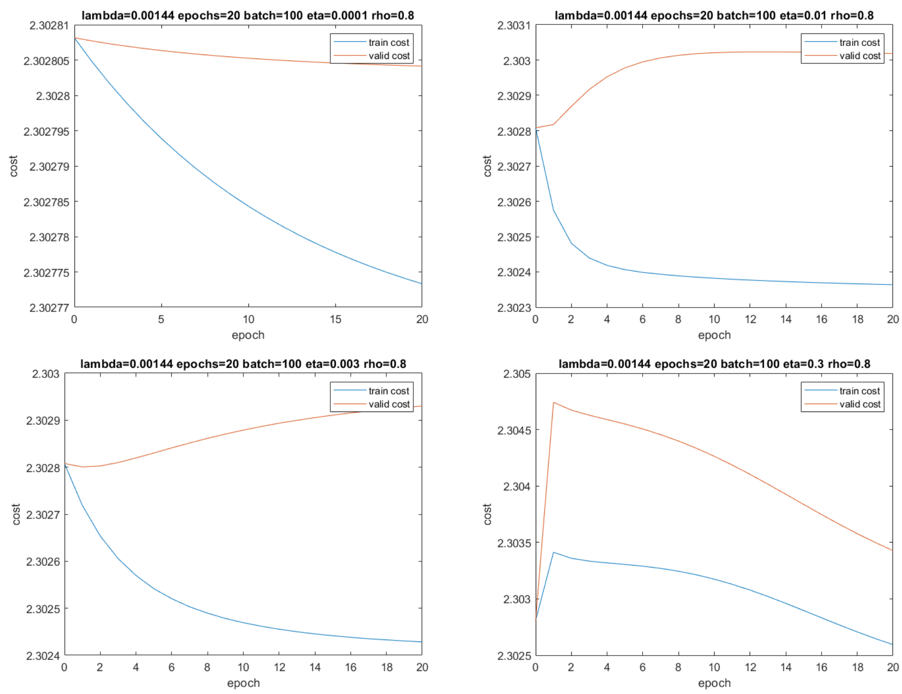

Figure 5: training curve of 3-layers network without bn

more unstable.When eta is decreased, the curve became more stable but the loss can hardly get smaller. I got the conclusion that **when using bigger eta, the network which is without BN will be more unstable and hard for training.** After parameter searching, a better outcome is shown in Figure7. we can notice the curve is not stable.

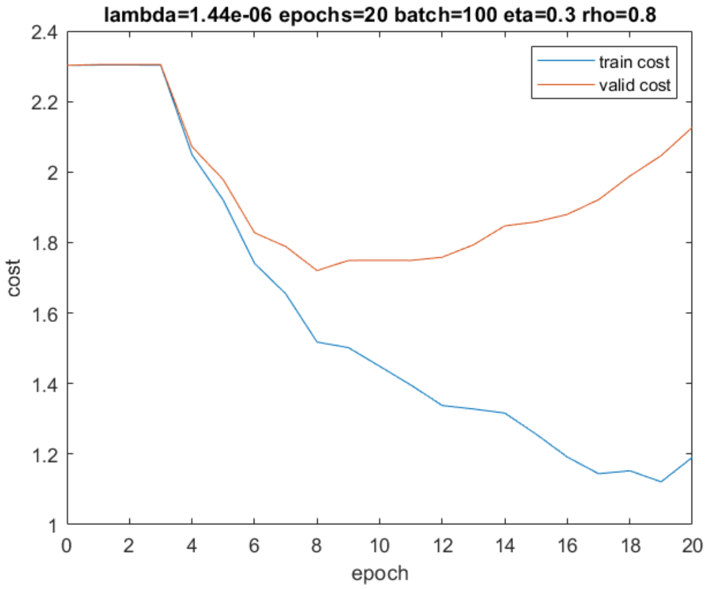

Figure 6: A better training curve of 3-layers network without bn

```
#################### epoch= 17 #######################
train loss=1.144251 Validation loss = 1.921561
train acc=0.605100  Validation acc = 0.399400
#################### epoch= 18 #######################
train loss=1.152620 Validation loss = 1.988506
train acc=0.597000  Validation acc = 0.391600
#################### epoch= 19 #######################
train loss=1.121293 Validation loss = 2.046439
train acc=0.610200  Validation acc = 0.390900
#################### epoch= 20 #######################
train loss=1.190905 Validation loss = 2.126961
train acc=0.594200  Validation acc = 0.379600
```

Listing 3: 3-layer network without bn running outcome

Exercise 3

1. ***[s_bn,mean_scores,var_scores] = BatchNormalization(scores,varargin)***
   I wrote the Batchnormalization() forward pass function according to equations (11) -(17) in Lab3 instruction. Is is assumed that the layer means and variances are computed from the mini-batch data sent into the function while training,

and the un-normalized scores are normalized by known pre-computed means and variances while testing. I use the **varargin** cell structure to control this. The Batchnormalization( ) is shown below:

```
 function [s_bn,mean_scores,var_scores] = BatchNormalization(scores,varargin)
 if numel(varargin) == 2
     eps = 1e-6;
     mean_scores = varargin{1};
     var_scores  = varargin{2};
     s_bn = diag(var_scores+eps)^(-0.5) *(scores-repmat(mean_scores,1,size(scores,2)));
 else
     eps = 1e-6;
     n = size(scores,2);
     mean_scores = mean(scores,2);
     var_scores = var(scores, 0, 2);
     var_scores = var_scores *(n-1)/n;
     s_bn = diag(var_scores+eps)^(-0.5) *(scores-repmat(mean_scores,1,size(scores,2)));
 end
end
```

Listing 4: snippet Batchnormalization

2. ***EvaluateClassifier3(X,W,b,GDparams,varargin)***
   The Evaluateclassifier3( ) is also changed according to Batchnormalization( ):

```
for i = 1:L-1
    if IFbn
        if numel(varargin) == 2
            mean_score = varargin{1}{i};
            var_score  = varargin{2}{i};
            S_hat{i} = S{i};
            [S{i},M_bn{i},V_bn{i}]= BatchNormalization(S{i},mean_score,var_score);
        else
            S_hat{i} = S{i};
            [S{i},M_bn{i},V_bn{i}]= BatchNormalization(S{i});
        end
    end
    h{i} = max(0,S{i});
    S{i+1}= W{i+1}*h{i}+repmat(b{i+1},1,n);
end
```

Listing 5: snippet Batchnormalization

3. ***ComputeGradients3(X,h,S,S_hat,Y, P, W,...,V_bn)*** I wrote the computeGradient3( ) into two parts: one is the pipeline without batch normalization, the other is the pipeline totally following the back-prop algorithm in Lab3

instruction. Then I used the parameter **Ifbn** to switch. The script is shown below:

```
if IFbn
      g   = cell(N,1);
      % calculate gk
      for i = 1:N
          Yn = Y(:,i); % (K x 1)
          Pn = P(:,i); % (K x 1)
%            Xn = X(:,i); % (d x 1)
          g{i} = - Yn'/(Yn'*Pn)*(diag(Pn)-Pn*Pn');
%            g{i} = -(Yn-Pn)';
          % gradient L w.r.t b{L} = g
          grad_b{L} = grad_b{L} +g{i}';
          grad_W{L} = grad_W{L} +g{i}'*h{L-1}(:,i)';
      end
      % get grad_bk grad_wk
          grad_b{L} = grad_b{L}/N;
          grad_W{L} = grad_W{L}/N+2*lambda*W{L};
      % propagate to previous layers
      for i = 1:N
          g{i} = g{i}*W{L};
          g{i} = g{i}*diag(S{L-1}(:,i)>0);
      end
      % bn
      for i = L-1:-1:2
            g = BatchNormBackPass(g,S_hat{i},M_bn{i},V_bn{i});
            for j = 1:N
                grad_b{i} = grad_b{i} + g{j}';
                grad_W{i} = grad_W{i} + g{j}'*h{i-1}(:,j)';
            end
            grad_b{i} = grad_b{i}/N;
            grad_W{i} = grad_W{i}/N+2*lambda*W{i};

            for m = 1:N
                g{m} = g{m}*W{i};
                g{m} = g{m}*diag(S{i-1}(:,m)>0);
            end
      end
      g = BatchNormBackPass(g,S_hat{1},M_bn{1},V_bn{1});
      for j = 1:N
          grad_b{1} = grad_b{1} + g{j}';
          grad_W{1} = grad_W{1} + g{j}'*X(:,j)';
      end
      grad_b{1} = grad_b{1}/N;
      grad_W{1} = grad_W{1}/N+2*lambda*W{1};
else
     ... back-prop without BN

end
```

Listing 6: snippet of ComputeGradients3

4. ***g = BatchNormBackPass(g, S, mu, var)***

I wrote the BatchNormBackpass following the equation in the last page of Lecture note 4.

5. ***MiniBatchGD3(X, Y, GDparams, W, b,varargin)***

   In MiniBatch( ), I add exponential moving average for batch means and variances.

```
for j=1:fix(N/n_batch)
        j_start = (j-1)*n_batch + 1;
        j_end = j*n_batch;
        inds = j_start:j_end;
        Xbatch = Xtrain(:, inds);
        Ybatch = Ytrain(:, inds);
        [P,S,S_hat,h,M_bn,V_bn] = EvaluateClassifier3(Xbatch,Wstar,
    bstar,GDparams);
        if GDparams.IFbn
            if numel(varargin) == 0 && j == 1
                M_av = M_bn;
                V_av = V_bn;
            else
                for i = 1:L-1
                    M_av{i} = GDparams.alpha*M_av{i} +(1-GDparams.
    alpha)*M_bn{i};
                    V_av{i} = GDparams.alpha*V_av{i} +(1-GDparams.
    alpha)*V_bn{i};
                end
            end

            [grad_W,grad_b] = ComputeGradients3(Xbatch,h,S,S_hat,
    Ybatch, P, Wstar, GDparams,M_bn,V_bn);
        else
            [grad_W,grad_b] = ComputeGradients3(Xbatch,h,S,S_hat,
    Ybatch, P, Wstar, GDparams);
        end
```

Listing 7: snippet MiniBatchGD3

   The M_av and V_av were initialized to be M_bn and V_bn at the very first epoch.

Then I computed the gradient both in numerical calculation and analytic calculation. The batch of input data(Ttrain_X = train_X(:,1:20)) is 20. In the outcome. we can see the difference of W3 is the smallest.

Task

1. State how you checked your analytic gradient computations and whether you think that your gradient computations are bug free for your k-layer network with batch normalization.

The pipeline of back-prop of BN is based on the instruction equation 19-26. In order to make sure there is bug free, I calculate the gradient difference between numerical calculation and analytic calculation.The difference showing that my gradient computations are bug free.

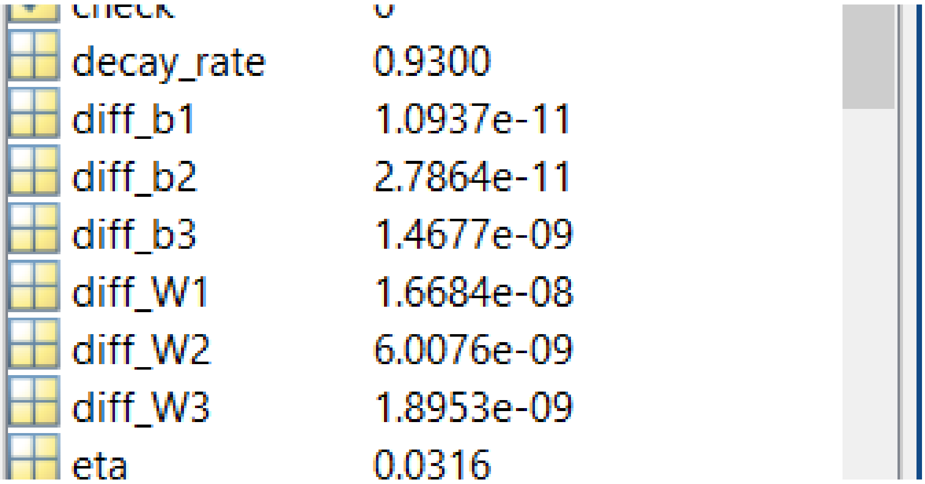

Figure 7: The gradient differences of 3-layers network with BN

2. Include graphs of the evolution of the loss function when you tried to train your 3-layer network without batch normalization and with batch normalization.

1. The parameters of 3-layers without BN:
   when I use the parameters searched in assignment2, the 3-layers without BN can hardly converge.
   rng_number=40; n_epochs=20; eta=0.0316; lambda=1.46e-4; rho=0.88; decay_rate=0.93
   The training curve is shown in Figure 8.

```
################### epoch= 17 ######################
train loss=2.302298 Validation loss = 2.302970
train acc=0.103200  Validation acc = 0.101000
################### epoch= 18 ######################
train loss=2.302295 Validation loss = 2.302966
train acc=0.103200  Validation acc = 0.101000
```

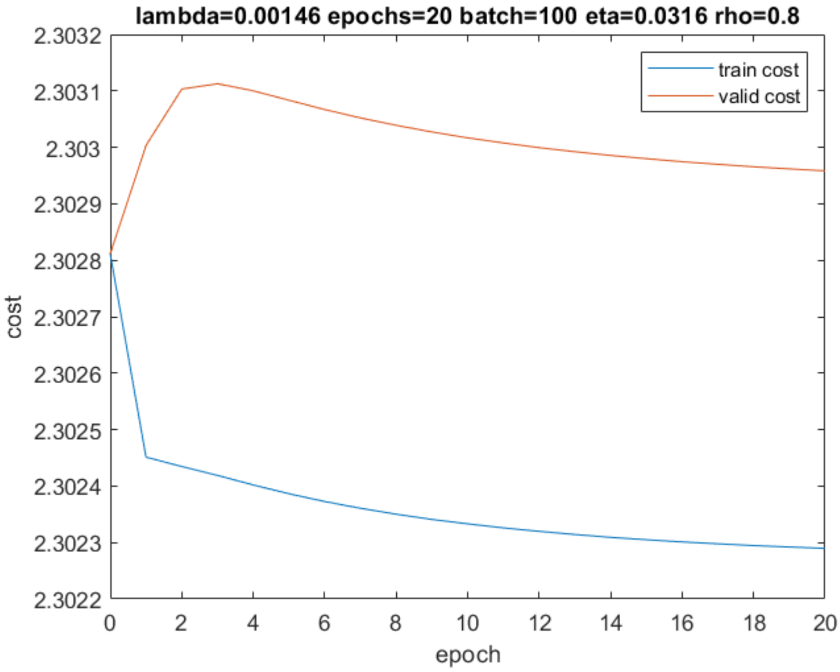

Figure 8: The training curve of 3-layers network without BN

```
#################### epoch= 19 #######################
train loss=2.302292 Validation loss = 2.302962
train acc=0.103200  Validation acc = 0.101000
#################### epoch= 20 #######################
train loss=2.302290 Validation loss = 2.302958
train acc=0.103200  Validation acc = 0.101000
```

Listing 8: training loss without BN

After a rough search, the training curve of 3-layer network without BN is shown below Figure 9. The loss is shown in Listing 9.I changed the parameters to be : *rng_number=40; n_epochs=20; eta=0.0316; lambda=1.46e-6; rho=0.8; decay_rate=0.93*

```
#################### epoch= 17 #######################
train loss=1.144251 Validation loss = 1.921561
train acc=0.605100  Validation acc = 0.399400
#################### epoch= 18 #######################
train loss=1.152620 Validation loss = 1.988506
train acc=0.597000  Validation acc = 0.391600
#################### epoch= 19 #######################
train loss=1.121293 Validation loss = 2.046439
train acc=0.610200  Validation acc = 0.390900
#################### epoch= 20 #######################
train loss=1.190905 Validation loss = 2.126961
train acc=0.594200  Validation acc = 0.379600
```

Listing 9: training loss without BN

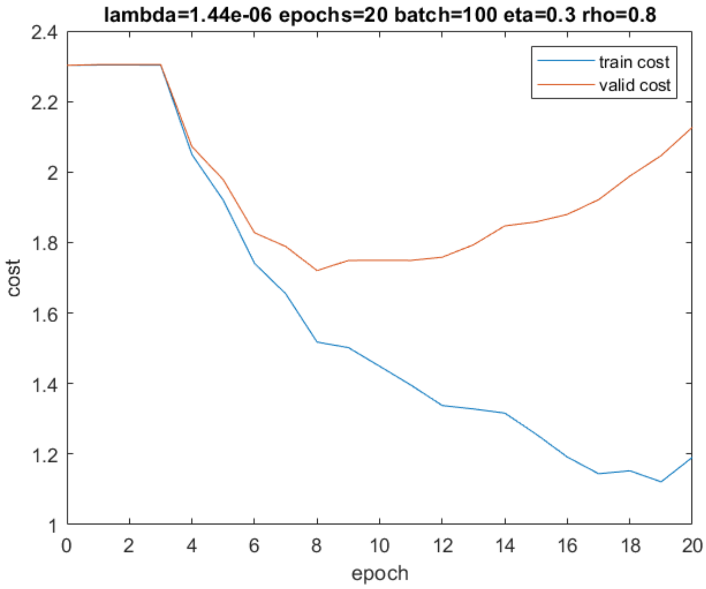

Figure 9: The training curve of 3-layers network without BN

2. The parameters of 3-layers with BN:
   The parameter is the same as assignment2. Is is clearly shown that using BN **can accelerate the convergence speed and make the training more stable.**Also, the outcome is better than the network without BN.
   rng_number=40; n_epochs=20; eta=0.0316; lambda=1.46e-4; rho=0.88; decay_rate=0.93

```
#################### epoch= 17 ######################
train loss=1.167175 Validation loss = 1.977223
train acc=0.647600  Validation acc = 0.416300
#################### epoch= 18 ######################
train loss=1.179972 Validation loss = 2.004503
train acc=0.641900  Validation acc = 0.409200
#################### epoch= 19 ######################
train loss=1.187421 Validation loss = 2.046314
train acc=0.643500  Validation acc = 0.407200
#################### epoch= 20 ######################
train loss=1.134107 Validation loss = 2.053281
train acc=0.665400  Validation acc = 0.412800
```

Listing 10: training loss with BN

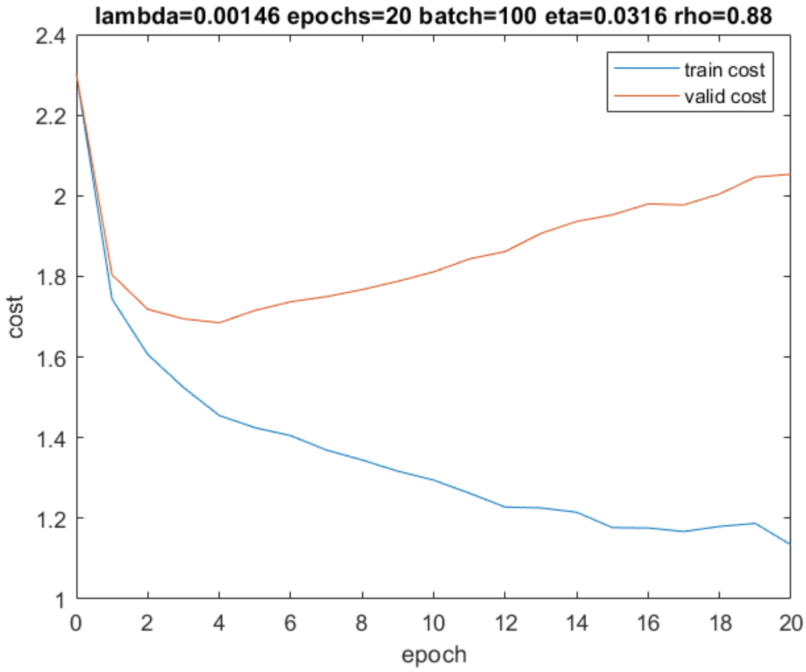

Figure 10: The gradient differences of 3-layers network with BN

3.State the range of the values you searched for lambda and eta, the number of epochs used for training during the ne search, and the hyper-parameter settings for your best performing 3-layer network you trained with batch normalization. Also state the test accuracy achieved by network.

First rough search:

I set the *search number = 100; rng_number=40; n_epochs=5;* $eta = e^{-4} - e^{1}$; $lambda = e^{-7} - e^{-1}$; *rho=0.9; decay_rate=0.95; train data = data_batch_1.mat; test data = test_batch.mat*

The top 3 graph is shown below

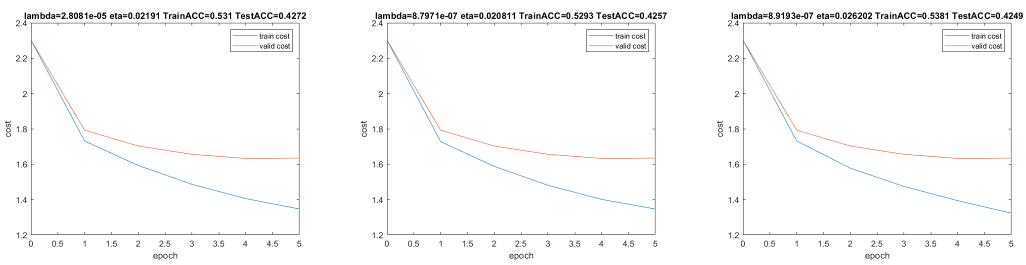

Figure 11: The Top-3 learning curve (rough search)

Table 1: Top16 coarse searching testing accuracy with 5 epochs

	eta	lambda	train accuracy	test accuracy
top1	0.021909942	2.81E-05	0.531	0.4272
top2	0.020810752	8.80E-07	0.5293	0.4257
top3	0.026202212	8.92E-07	0.5381	0.4249
top4	0.01653527	4.13E-06	0.5239	0.4237
top5	0.02430095	0.001323581	0.5321	0.4217
top6	0.103422378	0.000444255	0.5321	0.421
top7	0.007062768	7.66E-07	0.5137	0.4207
top8	0.01722279	9.63E-05	0.529	0.4204
top9	0.007056148	0.098428852	0.5121	0.4198
top10	0.055774277	0.024148088	0.5283	0.4195
top11	0.072874748	0.002220315	0.5396	0.4191
top12	0.043335998	0.000159417	0.53	0.4191
top13	0.045183708	0.000234025	0.5403	0.4189
top14	0.034854939	0.004073542	0.5193	0.4188
top15	0.209136611	0.000114808	0.5183	0.4187
top16	0.02837389	3.63E-06	0.52	0.418

Precise search:

I set the *search number = 100; rng_number=40; n_epochs=10; eta=* $0.009 - 0.035$*;* *lambda=*$e^{-7} - e^{-3}$*; rho=0.9; decay_rate=0.95; train data=data_batch_1.mat; test data = test_batch.mat*

The Best test accuracy at the second search is 0.435 (10 epoch)

Table 2: Top16 precise searching testing accuracy with 5 epochs

	eta	lambda	train accuracy	test accuracy
top1	0.009976828	2.24E-06	0.5944	0.4351
top2	0.015179735	1.76E-05	0.5934	0.4285
top3	0.022828666	1.68E-06	0.6111	0.428
top4	0.020718543	4.91E-05	0.5971	0.4277
top5	0.029637711	7.51E-07	0.6012	0.4268
top6	0.031951095	3.66E-05	0.6041	0.4267
top7	0.01832903	1.16E-06	0.5943	0.4267
top8	0.013022228	4.47E-07	0.5985	0.4264
top9	0.013763121	2.90E-06	0.5978	0.426
top10	0.022244822	6.03E-07	0.5875	0.4259
top11	0.017005871	1.15E-07	0.5895	0.4257
top12	0.021288386	3.99E-06	0.5997	0.4255
top13	0.017281191	1.59E-05	0.5861	0.4251
top14	0.027226389	9.58E-07	0.6132	0.4246
top15	0.016494211	7.45E-05	0.5926	0.4243
top16	0.034853878	6.75E-06	0.603	0.4235

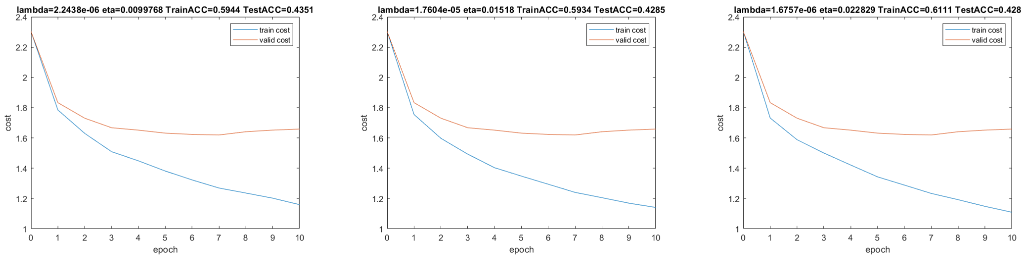

Figure 12: The Top-3 learning curve (precise search)

4.Plot the training and validation loss for your 2-layer network with batch normalization with 3 different learning rates (small, medium, high) for 10 epochs and make the same plots for a 2-layer network with no batch normalization.

After a rough search(*search number* = *20*), I set the parameters to be: *rng_number=40; n_epochs=10; lambda*=$1.2617e^{-4}$*; rho=0.9; decay_rate=0.95; train data=data_batch_1.mat; test data = test_batch.mat*

1. ***eta = 0.3(high)***

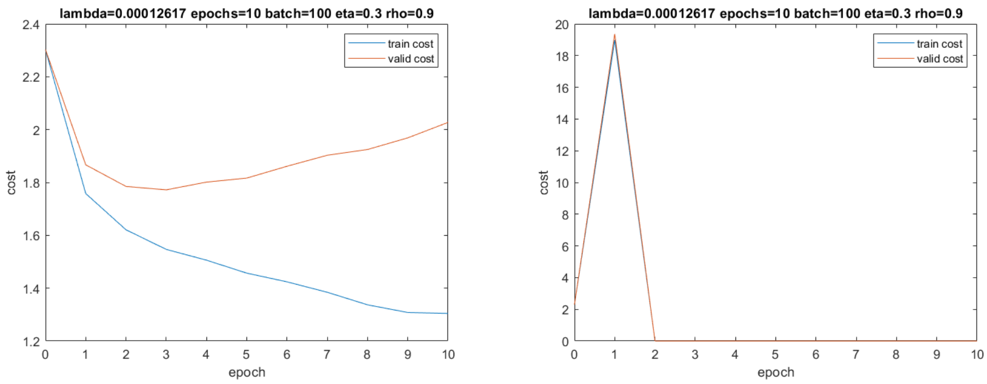

Figure 13: (high lr)Left:BN Right:without BN

```
% with BN
#################### epoch= 10 ######################
train loss=1.304410 Validation loss = 2.027296
train acc=0.589400  Validation acc = 0.395000
% without BN
bad parameter, too large
Elapsed time is 6.014476 seconds.
```

Listing 11: training loss

2. ***eta = 0.03(medium)***

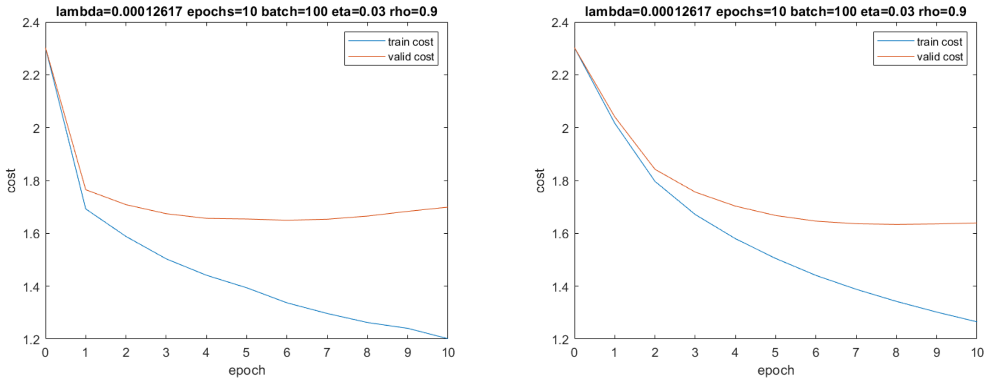

Figure 14: (medium lr),Left:BN Right:without BN

```
% with BN
#################### epoch= 10 #######################
train loss=1.201245 Validation loss = 1.699140
train acc=0.595900  Validation acc = 0.421600
% without BN
#################### epoch= 10 #######################
train loss=1.265019 Validation loss = 1.639208
train acc=0.567800  Validation acc = 0.432900
```

Listing 12: training loss

3. ***eta = 0.003(small)***

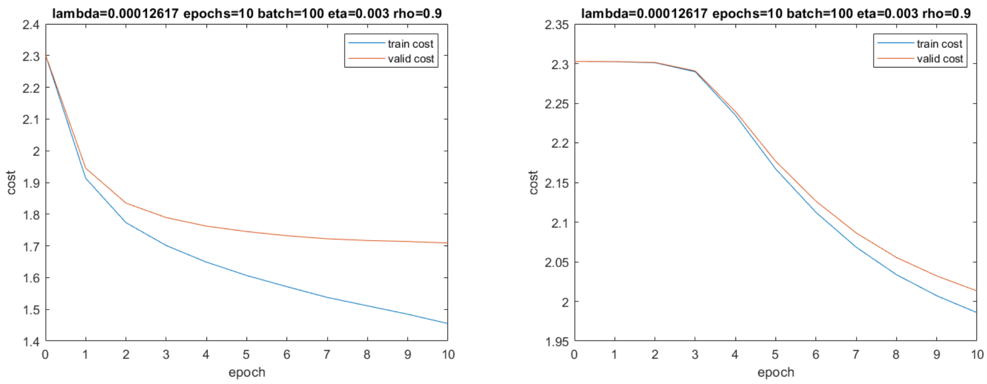

Figure 15: (small lr),Left:BN Right:without BN

```
% with BN
#################### epoch= 10 #######################
train loss=1.455016 Validation loss = 1.709162
train acc=0.514200  Validation acc = 0.399000
% without BN
#################### epoch= 10 #######################
train loss=1.985748 Validation loss = 2.013167
```

```
train acc=0.262800   Validation acc = 0.246800
```

Listing 13: training loss

Conclusion

From these comparison above, I can draw the conclusion that incorporating Batch Normalization can make the network training more stable and converge faster. It can adapt larger range of eta without collapse.