بسم الله الرحمن الرحیم

فهرست

تمرین اول و دوم شبکه های عصبی

محسن رجائی ۹۴۰۹۸۴۴

ابتدا ساختار شبکه ساخته شده را شرح می دهیم(این نکته قابل توجه است که پیاده سازی انجام شده به صورت پویا می باشد و تعداد لایه های شبکه و تعداد نورون های هر لایه، تعداد ورودی ها و خروجی ها و سایر پارامتر ها قبل از اجرای یادگیری شبکه از کاربر در ورودی دریافت می گردد، و بوسیله این کد می توان شبکه را با هر تعداد لایه دلخواه و هر تعداد نورون مورد نیاز ایجاد کرد، اما در اینجا برای اینکه بتوان این شبکه را شرح داد از یک مدل خاص و خیلی ساده شده استفاده می کنیم).

تئوری یادگیری

تئوری یادگیری ما در تصویر زیر خلاصه شده است:

$$E^p = \frac{1}{2}(D^p - M(Z^p, W))^2, \qquad E_{train} = \frac{1}{P}\sum_{p=1} E^p$$

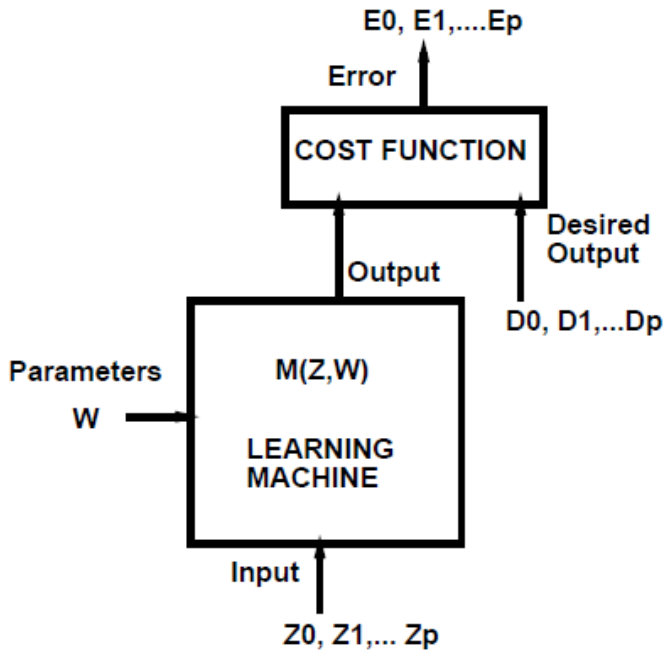


Fig. 1. Gradient-based learning machine.

بعد از بدست آمدن مقدار خطا با روش `Backpropagation` مقادیر `W` یا وزن ها را تغییر می دهیم تا مقدار خطای بدست آمده مینیمم گردد.

ساختار شبکه:

مدل ما ۴ لایه دارد:

لایه اول ورودی ها

لایه دوم، در واقع لایه مخفی اول

لایه سوم و لایه مخفی دیگر (دوم)

لایه چهارم و در واقع لایه خروجی که ۱۰ نورون در این لایه قرار دارد .

همچنین هر لایه، بجز لایه آخر(خروجی) دارای یک نورون `Bias` است.

چون ساختار شبکه در هنگام اجرا مشخص میشود(ورودی ها از کاربر دریافت میگردند)، و برای اینکه درک مفهوم انجام کار ساده تر باشد، شکل زیر را به عنوان شبکه مورد استفاده در نظر بگیرید(این ساختار ساده شده ساختار اصلی است).

میتوانید تصویر شبکه را در صفحه بعد ببینید.

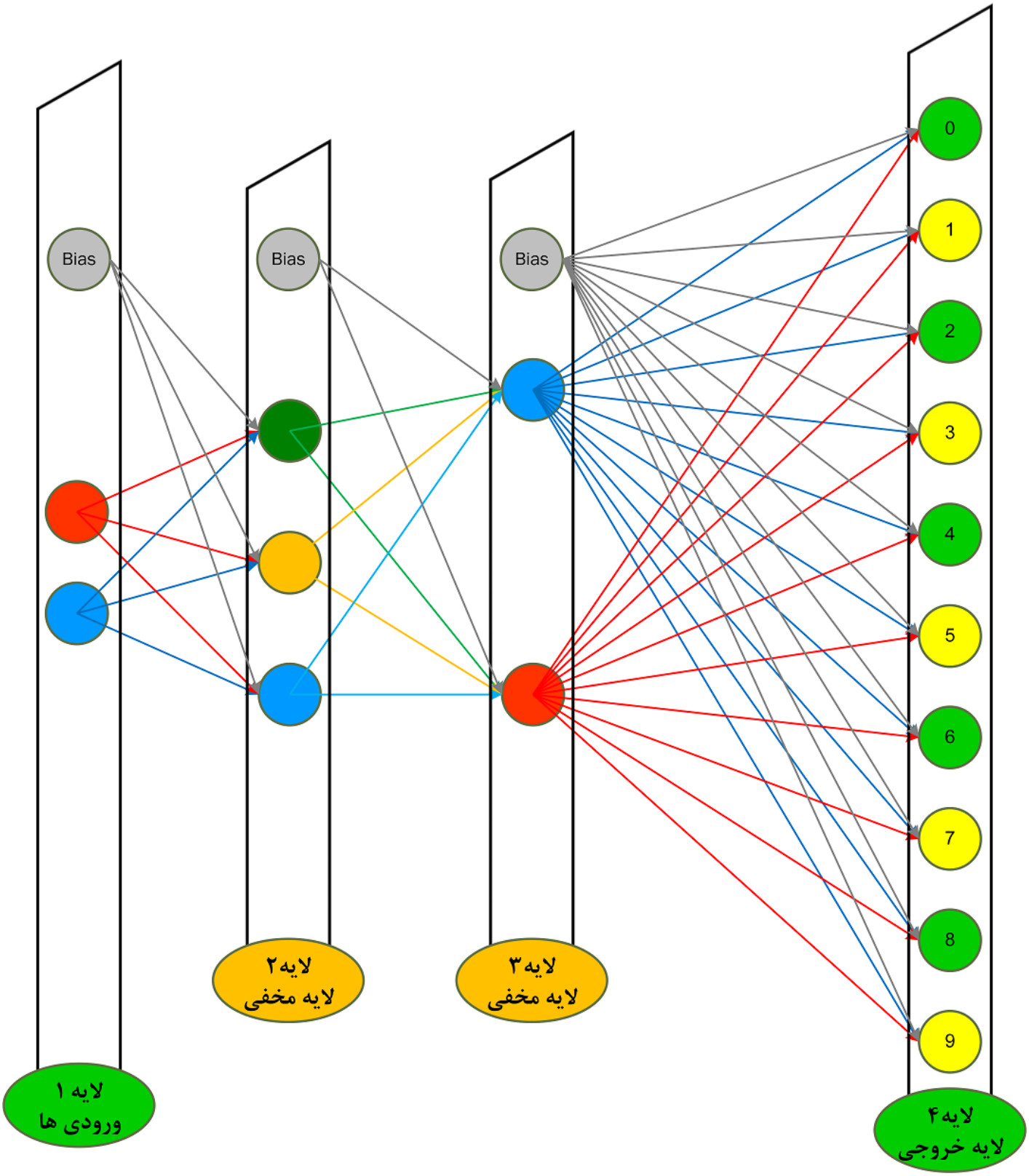
ساختار یک شبکه ۴ لایه ای که دارای ۷۸۴ ورودی و ۱۰ خروجی و ۲ لایه مخفی می باشد به صورت زیر می باشد:

MATLAB Variable: layer
Nov 23, 2015

فیلدها

	Size	wts	z	a	delta	DW	bias	MSE	delta_bias
1	2	[]	[]	*2x50000 do...*	[]	[]	[]	[]	[]
2	3	[0.0755,0.09...	[-0.0895,-0....	[0.4776,0.47...	[0,0,0]	[0,0,0;0,0,0;...	1	[]	0
3	2	[0.1116,-0.0...	[0.1835,-0.0...	[0.5457,0.49...	[0,0]	[0,0;0,0;0,0;...	1	[]	0
4	10	*3x10 double*	[0.1992,0.04...	[0.5496,0.51...	[0,0,0,0,0,0,...	*3x10 double*	1	*10x50000 do..*	0

تصویری از ساختار ساده شده شبکه طراحی شده:



ما از یک ساختار در متلب به شرح زیر برای تعریف لایه ها استفاده کردیم:

```
layer =
struct('Size',[],'wts',[],'z',[],'a',[],'delta',[],'DW',[],'bias',[],
'MSE',[],'delta_W',[],'delta_bias',[],'big_delta',[],'big_delta_bias',
[],'delta_W_last',[]);
```

که هر لایه توسط دستور `layer(i)` مشخص میشود.

فیلد `layer(i).Size` مشخص کننده تعداد نورون ها در لایه `i` ام می باشد.

فیلد `layer(i).wts` مشخص کننده وزن های ورودی به این لایه می باشد، در واقع وزن های ورودی از لایه `i-1` به لایه`i` را مشخص می کند. این فیلد یک ماتریس دوبعدی است که در لایه اول این ماتریس وجود خارجی ندارد(بدلیل اینکه هیچ وزنی در لایه اول که همان ورودی ها هستند معنا ندارد)و اندازه آن توسط رابطه زیر مشخص میشود:

`layer(c).wts.Size =[layer(i-1).Size+1,layer(i).Size]`

فیلد `layer(i).z` مشخص کننده مجموع وزن های ورودی به لایه `i` ضرب در ورودی های مربوط به نورون `j` در لایه `i` می باشد.(`z=WX`)

فیلد `layer(i).a` مشخص کننده اعمال تابع سیگموید بر روی مقدار `layer(i).z` می باشد.(`a=sigmoid(layer(i).z)`)

فیلد `layer(i).delta` نیز مشخص کننده دلتا در لایه `i` برای پیاده سازی الگوریتم `Backpropagation` می باشد.

فیلد `MSE` هم نمایش دهنده مجموع مربعات خطا می باشد.

فیلد `delta_bias` و `delta_W` به ترتیب برای آپدیت کردن وزن های مربوط به بایاس و وزن های هر لایه بکار می روند.

فیلدهای `'big_delta'` , `DW` , `'big_delta_bias'` نگهدارنده وزن ها برای استفاده در روش آموزش دسته ای مورد استفاده قرار می گیرد.

فیلد `'delta_W_last'` برای اضافه کردن `Momentum` به کار می رود.

در اینجا تصویری از ساختار مربوط به یک`MLP` را با ۴لایه می بینید:

Fields	Size	wts	z	a	delta	DW	bias	MSE	delta_W	delta_bias	big_delta	big_delta_bias	delta_W_last
1	784	[]	[]	784x1000 do...	[]	[]	[]	[]	[]	[]	[]	[]	[]
2	25	785x25 dou...	1x25 double	1x25 double	1x25 double	785x25 dou...	1	[]	[]	0	785x25 double	1x25 double	785x25 double
3	10	26x10 double	[-5.9156,-5....	[0.0027,0.00...	[0.0027,0.00...	26x10 double	1	10x1000 do...	[]	0	26x10 double	[-2.8662e+03,-1.7...	26x10 double
4				activation's									

تعداد نورون های هر لایه

وزن های لایه قبل به این لایه

ورودی های درخواستی که باید توسط کاربرمقدار دهی گردند:(ورودی ها باید توسط کاربر در پاسخ به سولات زیر وارد شوند)

MR_MLP:

```
Enter the learning method(1 for Batch method or 0 for Online method)>>
Enter the learning rate value >>
Enter the momentum value >>
Enter the regularization value >>
Enter the validation check value >>
Enter the iteration value >>
Enter the number of samples >>
Enter the number of layers >>
…
Enter numbers of INPUTS (number of neuron in first layer) >>
Enter numbers of OUTPUTS (number of neuron in last layer) >>
```

شیوه انجام کار:

- Load MNIST Dataset
- Receive parameter from user in input before start learning
- Create MLP Layer’s

ایجاد ساختار شبکه عصبی و مقدار دهی اولیه تصادفی به وزن ها

مقدار دهی اولیه به پارامتر هایی مثل:

نرخ یادگیری، تعدا لایه ها و تعدا نورون های هر لایه، ضریب منتظم سازی، تعداد خروجی ها و …

به دلیل اینکه خود کد به اندازه کافی واضح هست در ادامه خلاصه کد feed forward و آموزش شبکه عصبی قرار داده می شود:

برا ی اینکه فرمت کد بهم نریزد و نمایش خوبی داشته باشد تصویر کد خلاصه را قرار داده ام و کد خلاصه شده و تصویر آن را نیز پیوست کرده ام.

Feed Forward & Train & Test.m

Feed Forward & Train & Test.png

```
%% ========== Forward : Computing Delta's : Update Weights =============
for epoch=1:iteration % forward and update weight's in number of  iterations
    delta_W=zeros();
    delta_theta=zeros();
    for num_in=1:samples
        %% ========== Forward =============
        for c=2:L
            for i=1:layer(c).Size
                % calaulate sum(XW)
                elm_sum = zeros();
                elm_sum = (layer(c-1).a(:))'*(layer(c).wts(1:end-1,i))+elm_sum;
                % add bias
                elm_sum = (layer(c).bias*layer(c).wts(end,i))+elm_sum;
                % calculate activation & z
                layer(c).z(1,i)=(elm_sum);
                layer(c).a(1,i) = sigmoid(layer(c).z(1,i));

            end
        end
        %%
        % ========== Computing MSE  =============
        layer(L).MSE(:,num_in) = (layer(L).a - target).^2;
        % ========== Computing Delta's  =============
        layer(L).delta = (layer(L).a - target);

        % Compute Other Delta's
        hl=L-1;
        while(hl>1)
            layer(hl).delta = layer(hl+1).delta * (layer(hl+1).wts(1:end-1,:))' .* ...
                layer(hl).a .* (1-layer(hl).a); % or sigmoidGradient(layer(hl).z)
            hl=hl-1;
        end

        %% ==========  Update Weights =============
        up_ind=L;
        while(up_ind>1)
            if parameter.method == 1 % batch method
                %% BIG DELTA Weights Batch for INPUT's(L(1))
                layer(up_ind).big_delta = -parameter.learning_rate.*((layer(up_ind-1).a * layer(up_ind).delta)) +...
                    layer(up_ind).big_delta;

                %% BIG DELTA BIAS's Batch for INPUT's(L(1))
                layer(up_ind).big_delta_bias = -parameter.learning_rate .* layer(up_ind).delta + ...
                    layer(up_ind).big_delta_bias;

            else % online method
                %% BIG DELTA Weights ONLINE
                delta_W = -parameter.learning_rate.* (layer(up_ind-1).a' * layer(up_ind).delta) ;
                delta_theta = -parameter.learning_rate .* layer(up_ind).delta ;

                %%Update Weight's layer i or update weights from i-1 to i
                % Update Weight's
                layer(up_ind).wts(1:end-1,:) =  layer(up_ind).wts(1:end-1,:) +...
                    delta_W + (parameter.alfa  * layer(up_ind).delta_W_last(1:end-1,:)) -...
                    parameter.lambda * parameter.learning_rate * (layer(up_ind).wts(1:end-1,:));
                layer(up_ind).delta_W_last(1:end-1,:) = delta_W;
                % Update Bias
                layer(up_ind).wts(end,:) = layer(up_ind).wts(end,:) +...
                    delta_theta + (parameter.alfa  * layer(up_ind).delta_W_last(end,:));
                layer(up_ind).delta_W_last(end,:)  = delta_theta;

            end % end if
            up_ind = up_ind-1;
        end
```

```
        %% Overall Batch
        if parameter.method == 1 % batch method
            for i=2:L
                % Update Weight's
                layer(i).wts(1:end-1,:) = (1/samples) *layer(i).big_delta(1:end-1,:) - ...
                    parameter.lambda * parameter.learning_rate * (layer(i).wts(1:end-1,:)) + ...
                    (parameter.alfa  *  layer(i).delta_W_last(1:end-1,:));
                %% Last DELTA W Weights
                layer(i).delta_W_last(1:end-1,:) = layer(i).wts(1:end-1,:);

                % Update Bias
                layer(i).wts(end,:) = (1/samples)*layer(i).big_delta_bias -...
                    (parameter.alfa  *  layer(i).delta_W_last(end,:));
                %               % Last DELTA W Weights
                layer(i).delta_W_last(end,:) = layer(i).wts(end,:);


            end
        end
        %%
        %% TEST & Accuracy & MSE
        Train or Test = 0;% accuracy on Train Data
        Accuracy_Train(epoch) = Test(layer,samples,Train_or_Test,L,labels,images)/samples;
        %
        Train_or_Test = 1;% accuracy on Test Data
        Accuracy_Test(epoch) = Test(layer,10000,Train_or_Test,L,tlabels,timages)/10000;
        %
        Train_or_Test = 2;% accuracy on Validation Data
        Accuracy_Validation(epoch) = Test(layer,10000,Train_or_Test,L,validation_labels,...
            validation_images)/10000;

        %% Compute Overall MSE
        MSE(epoch) = (sum(layer(L).MSE(:))^0.5) / samples;

        %% Validation check
        if epoch == 1
            validation = Accuracy_Validation(epoch);
        end

        if check_validation(1,validation_check) == 1
            break;
        elseif Accuracy_Validation(epoch)<=validation
            check_validation(1,index)=1;
            index = index+1;
        else
            validation = Accuracy_Validation(epoch);
        end
        %%
    end
end
```

لازم به ذکراست که تابع خطای مورد استفاده MSE می باشد.

تذکر: بعد از اینکه به تعداد"validation_check " بار مقدار خطای ارزیابی کاهش نیافت ما برنامه را خاتمه می دهیم و نتایج را اعلام می کنیم. به همین دلیل دربعضی از نمودار هایی که در ادامه در پاسخ به سوالات مشاهده خواهید کرد مشاهده می کنید که در یک لحظه مقادیر به صفر رسیده اند که این یعنی مقدار عدم تطابق رشد دقت بر روی داده های آموزشی و داده های ارزیابی به مقدار "validation_check " رسیده است و برنامه خاتمه یافته است.

شبکه مورد استفاده

شبکه غالب مورد استفاده ما برای آزمایش ها ۳ لایه دارد:

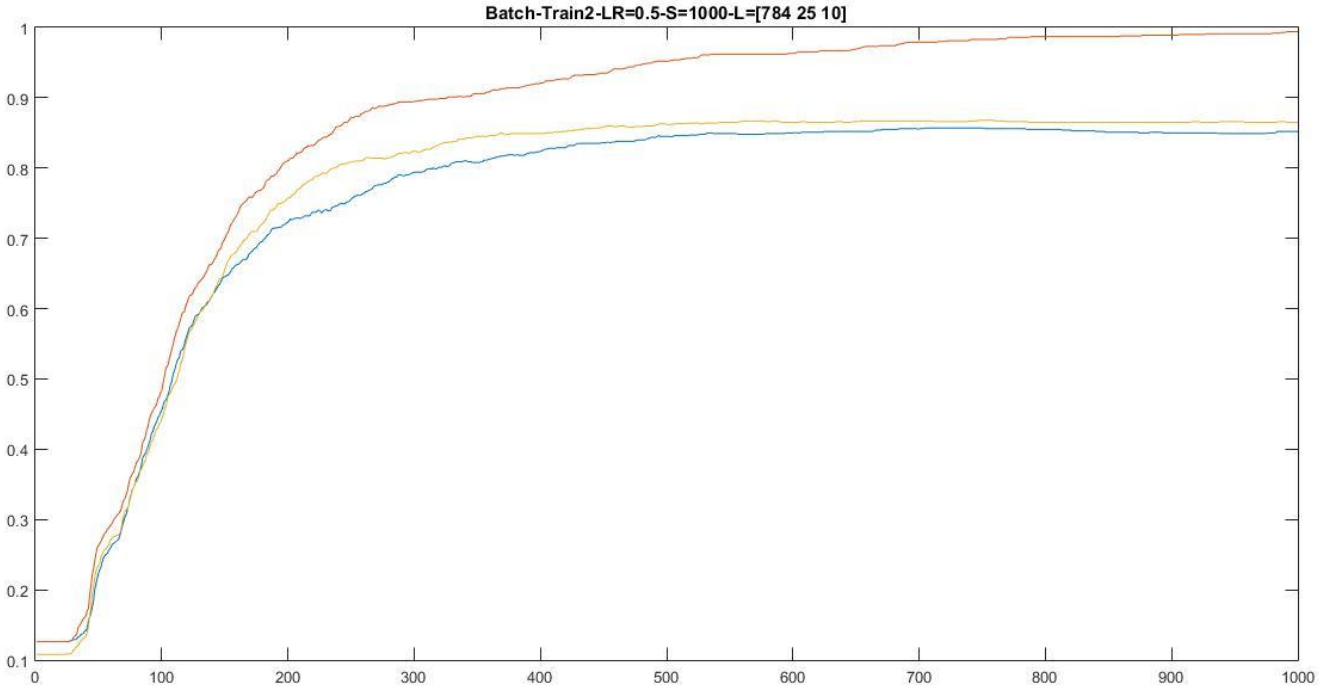
لایه ورودی با ۷۸۴ ورودی(نورون)

لایه پنهان با ۲۵ نورون

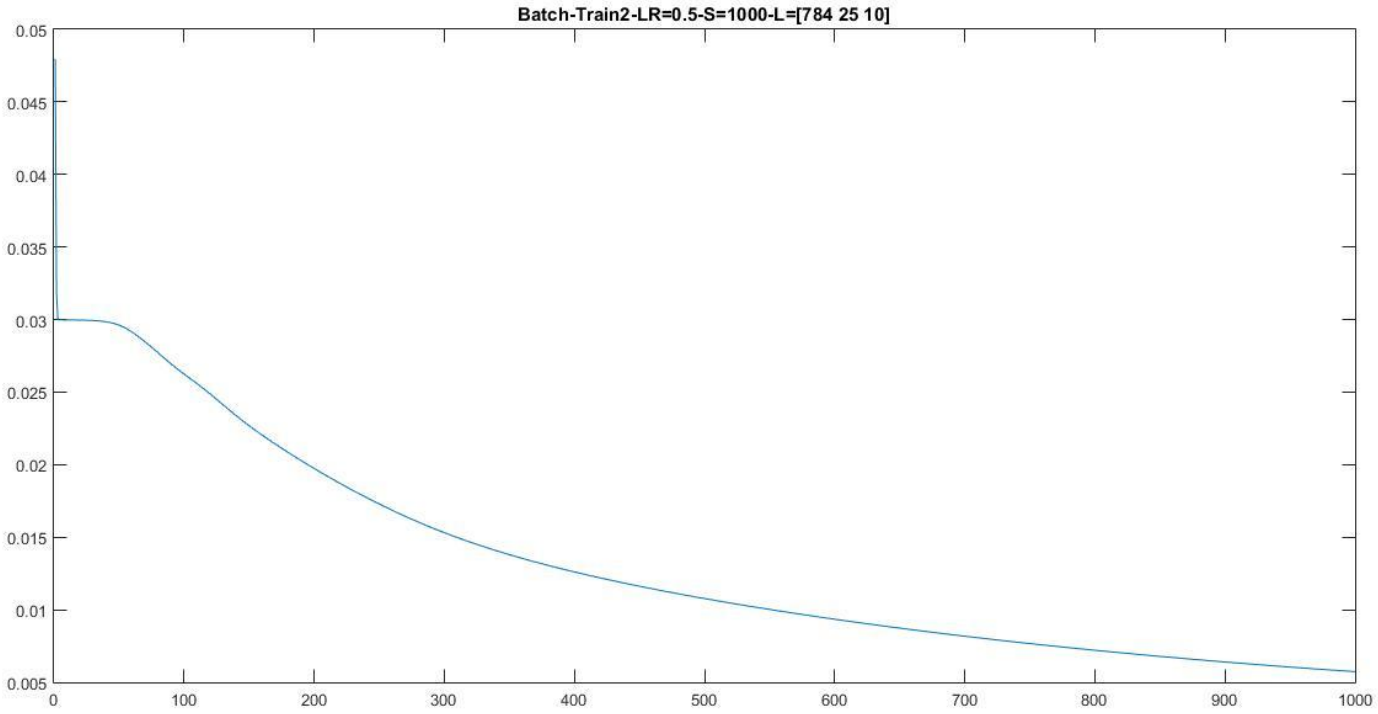
لایه خروجی با ۱۰ نورون

یادگیری با روش دسته ای:

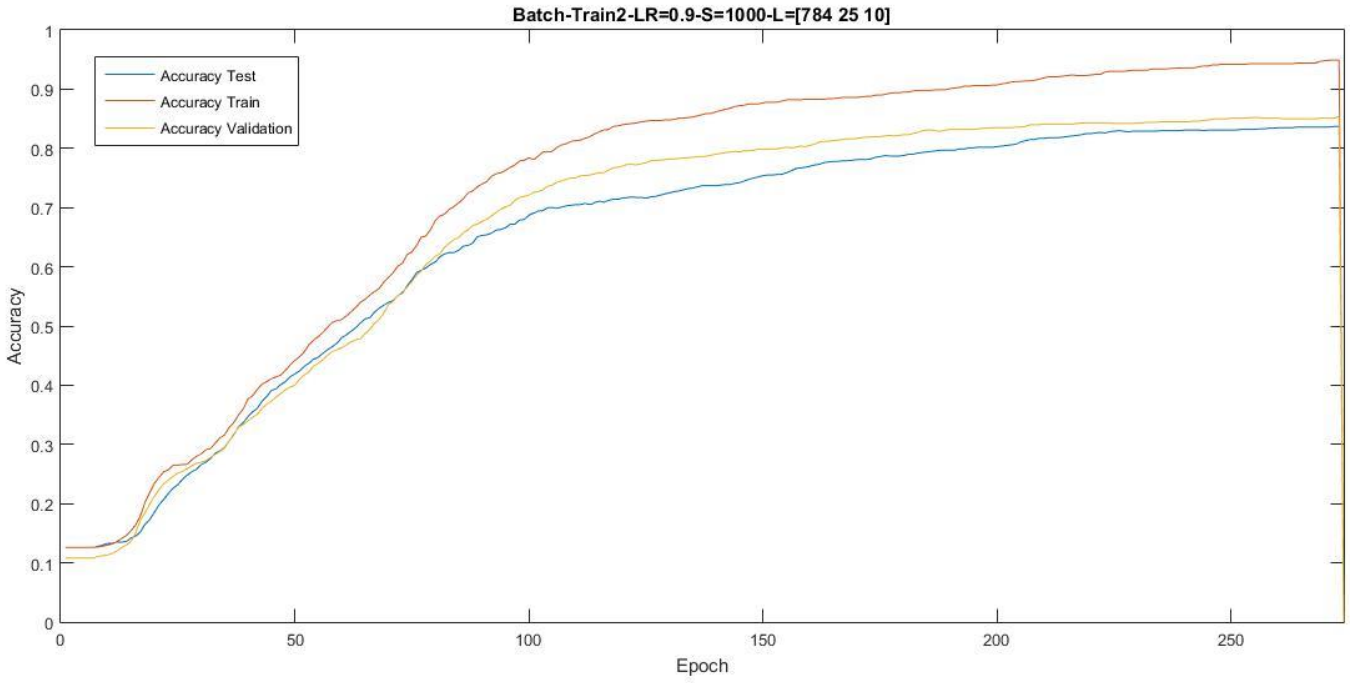
نمودار دقت بر حسب epoch بدست امده در یادگیری با روش دسته ای:(در ۱۰۰۰ epoch)



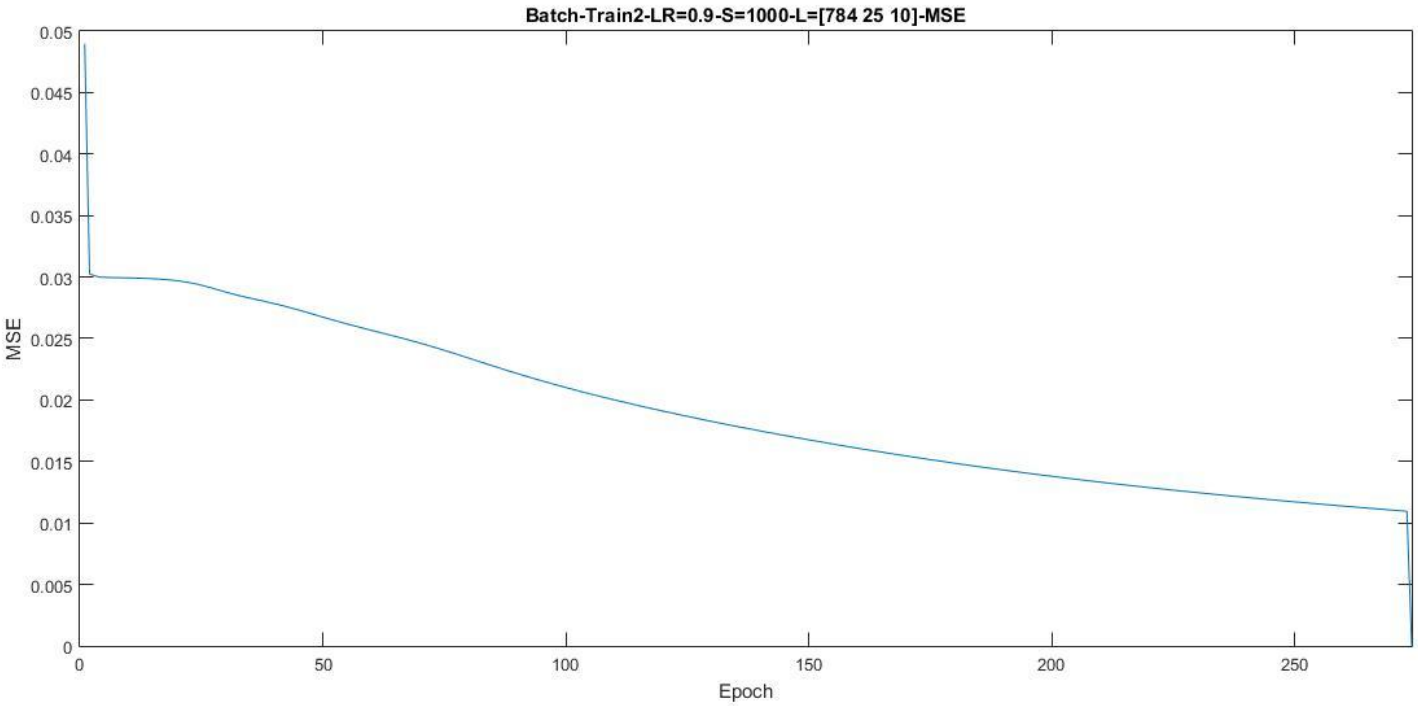
نمودار MSE بر حسب epoch:



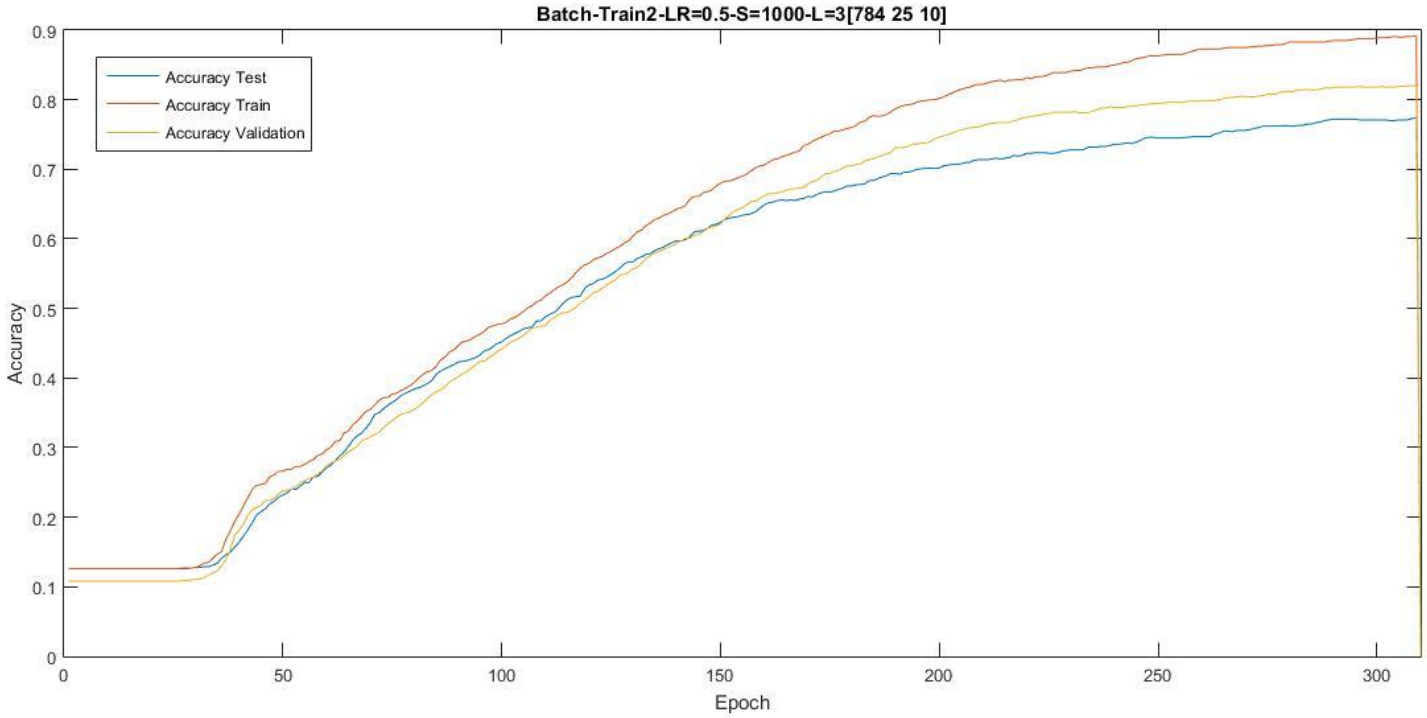
نمودار دقت آزمایش دوم:(در حدود ۲۵۰ epoch)



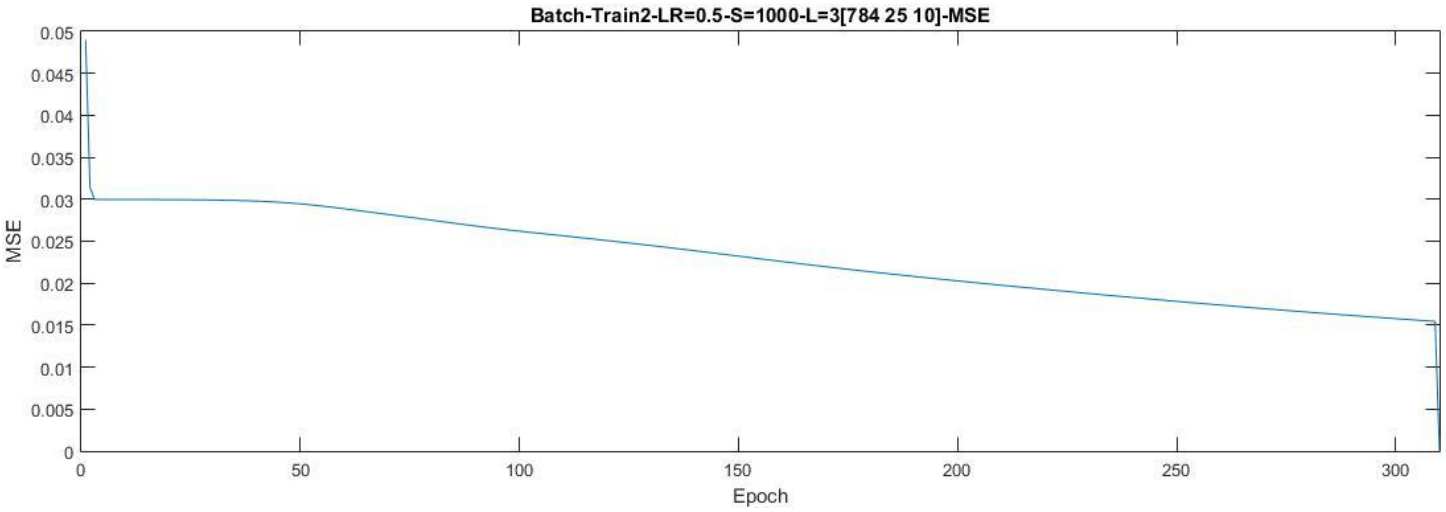
نمودار MSE مربوط به آزمایش دوم:



نمودار دقت آزمایش سوم:(در حدود ۳۰۰epoch)



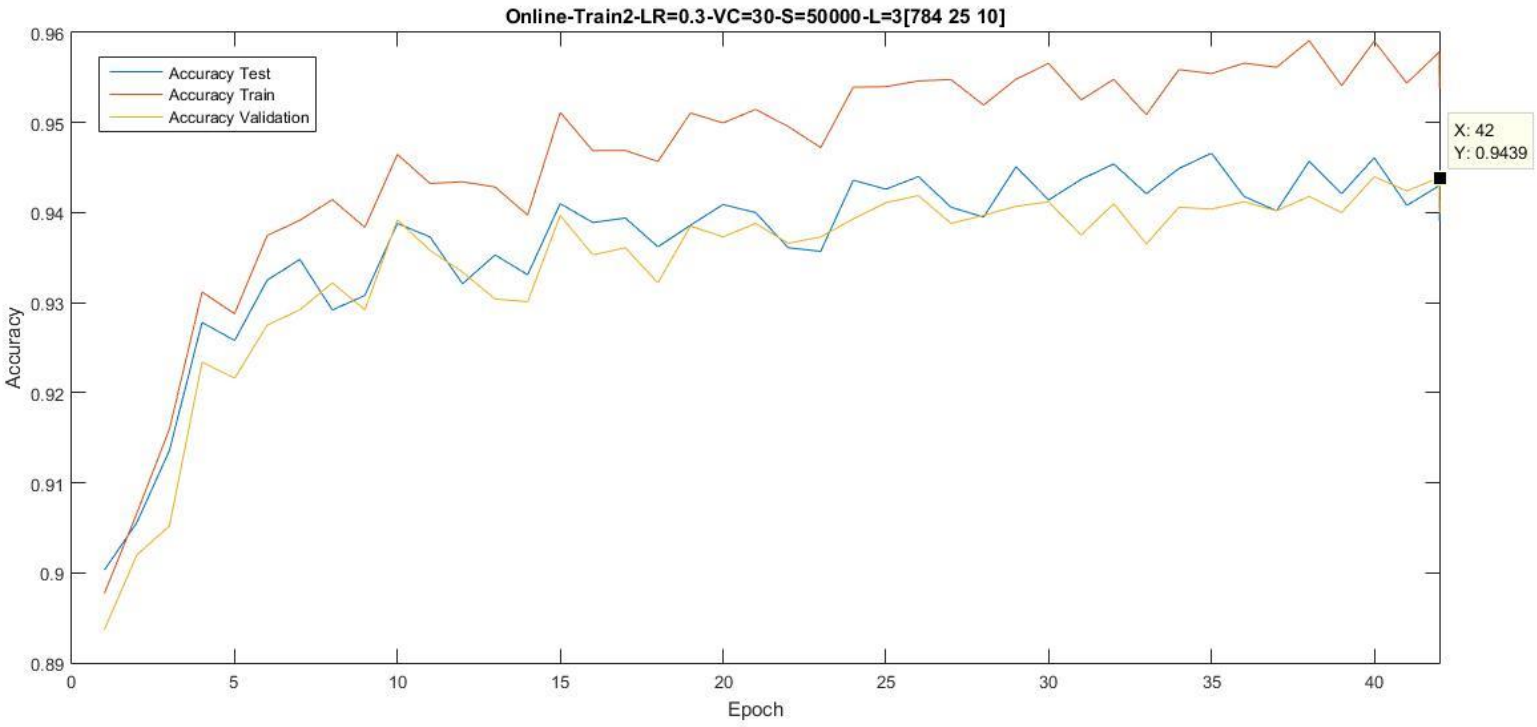
نمودار MSE آزمایش سوم:



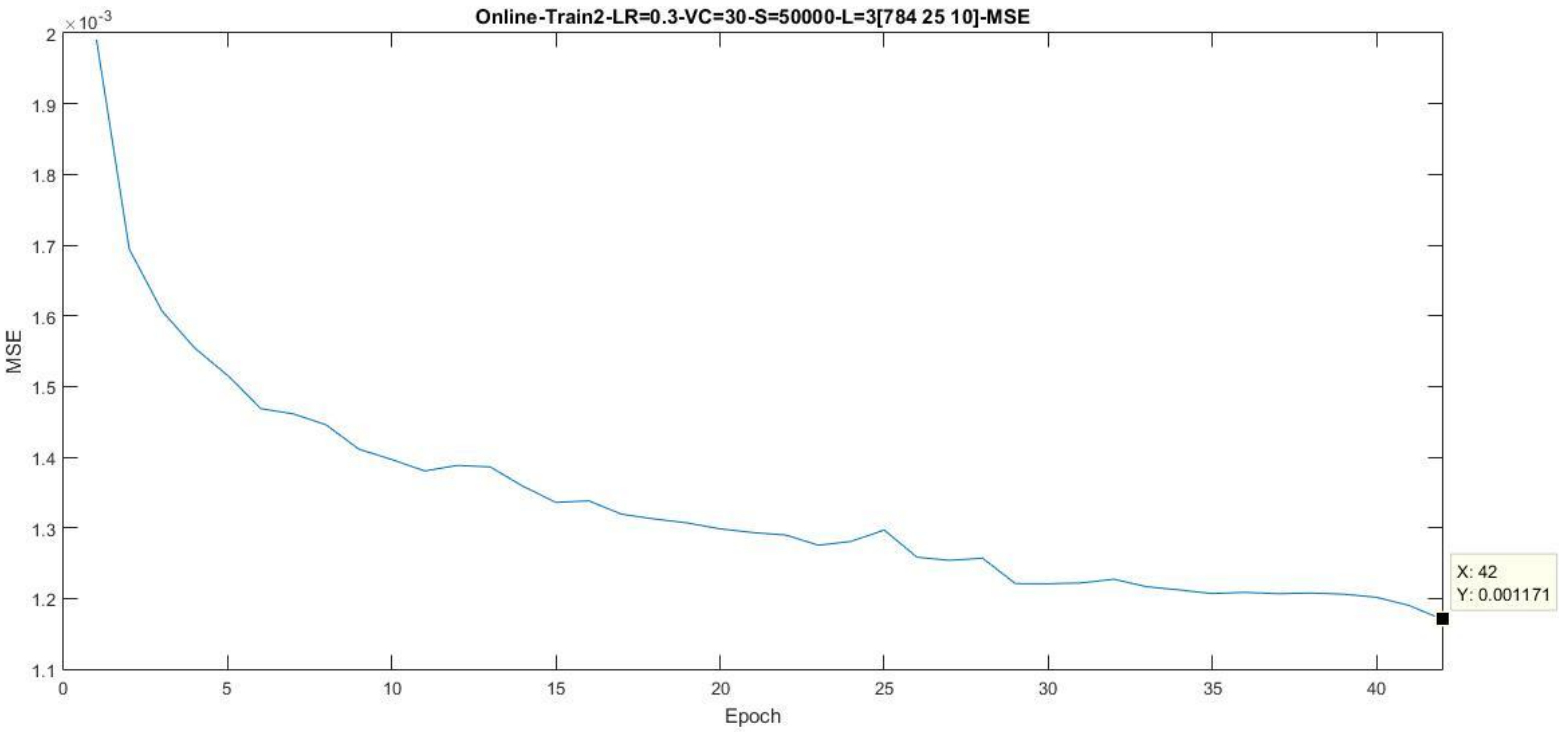
یادگیر ی با روش Online:

تست اول:

نمودار دقت بر حسب epoch:

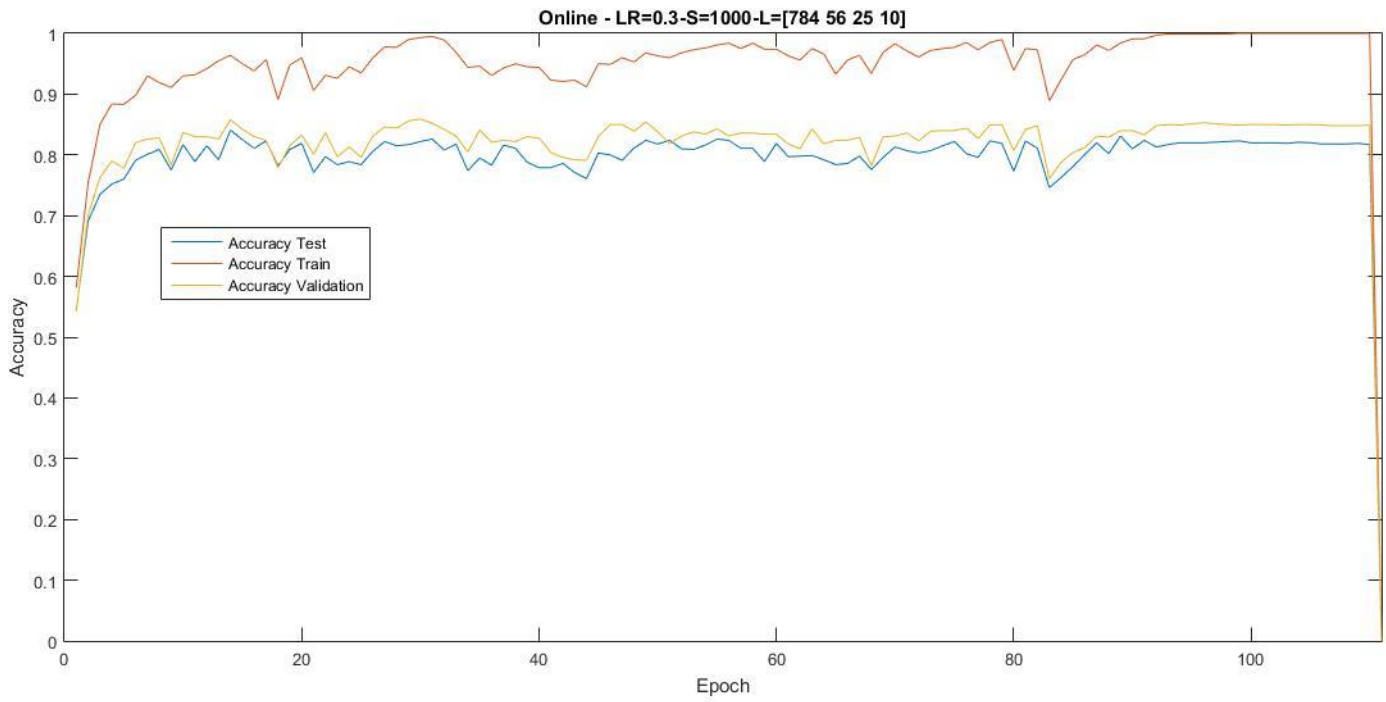


نمودار MSE بر حسب epoch:

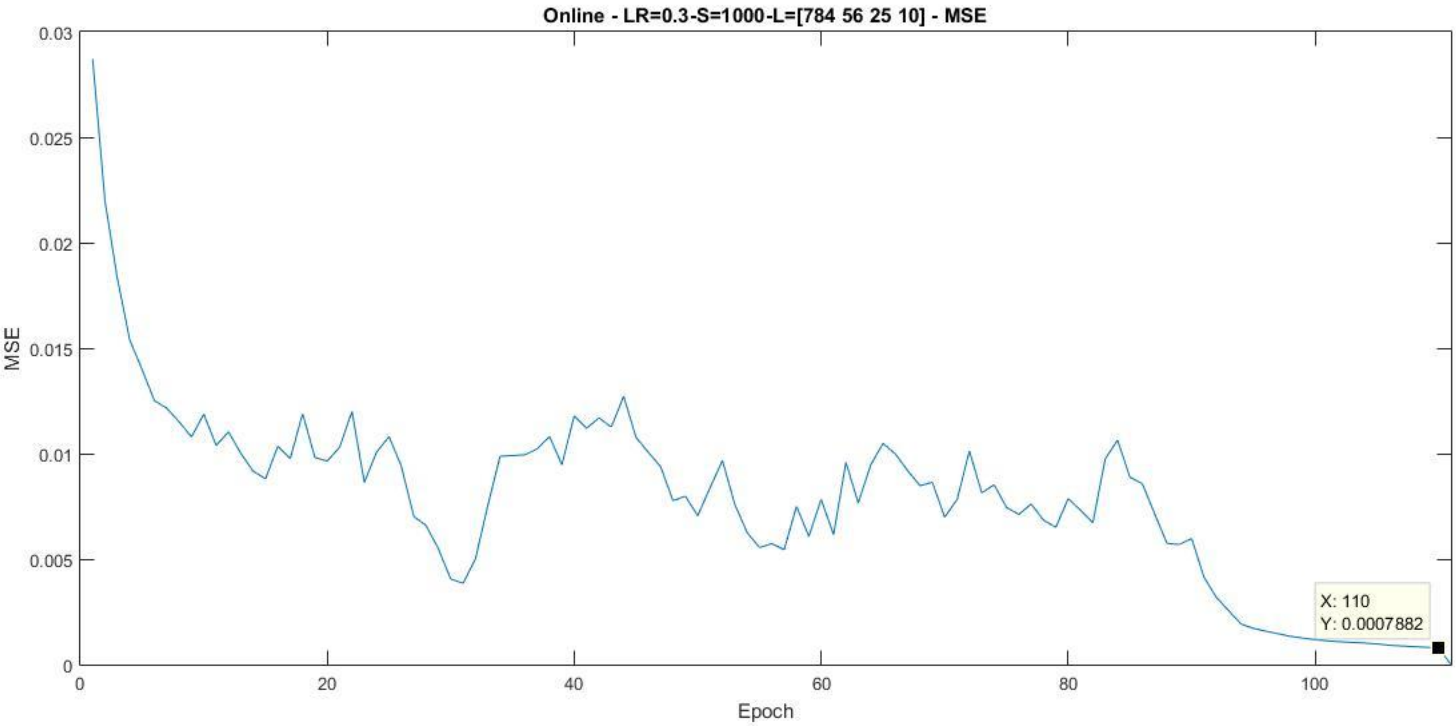


تست دوم:

نمودار دقت بر حسب epoch:



نمودار MSE بر حسبepoch:

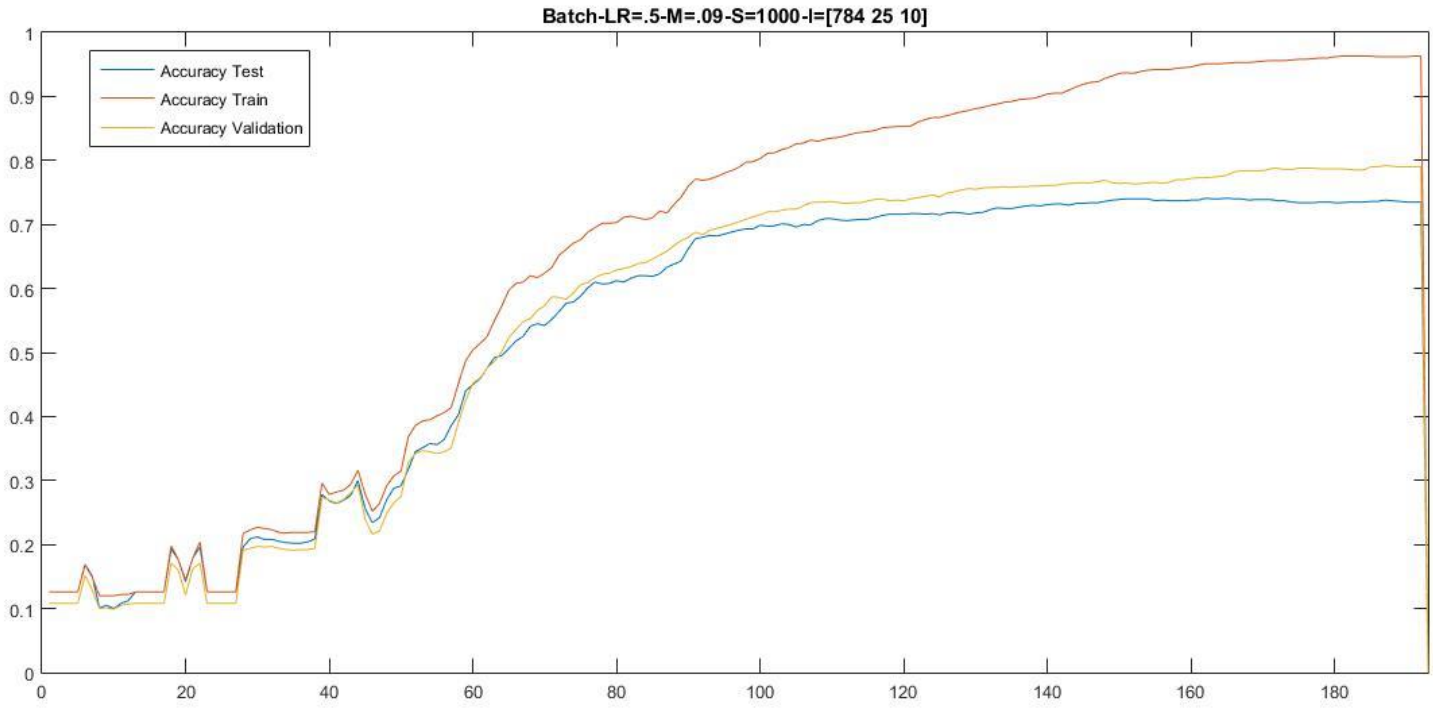


سرعت همگرایی این روش خیلی نسبت به روش دسته ای بهتر است،همانطور که مشاهده می کنید، در epoch اول ما دقت ۶۰ درصد را بر روی داده ها داریم، اما در روش دسته ای در epoch اول دقت ما در حدود ۱۲ درصد می باشد و بعد از epoch شماره ۹۰ به دقت ۱۰۰ درصد بر روی نمونه های آموزشی دست پیدا کرده کرده ایم و در روش دسته ای از epoch شماره ۳۰۰ به ۹۰ درصد دقت بر روی نمونه های آموزشی دست پیدا کرده ایم و همچنین در epoch شماره ۱۰۰۰ به دقت تقریبی ۱۰۰ درصد بر روی نمونه های آموزشی دست پیدا کرده ایم.

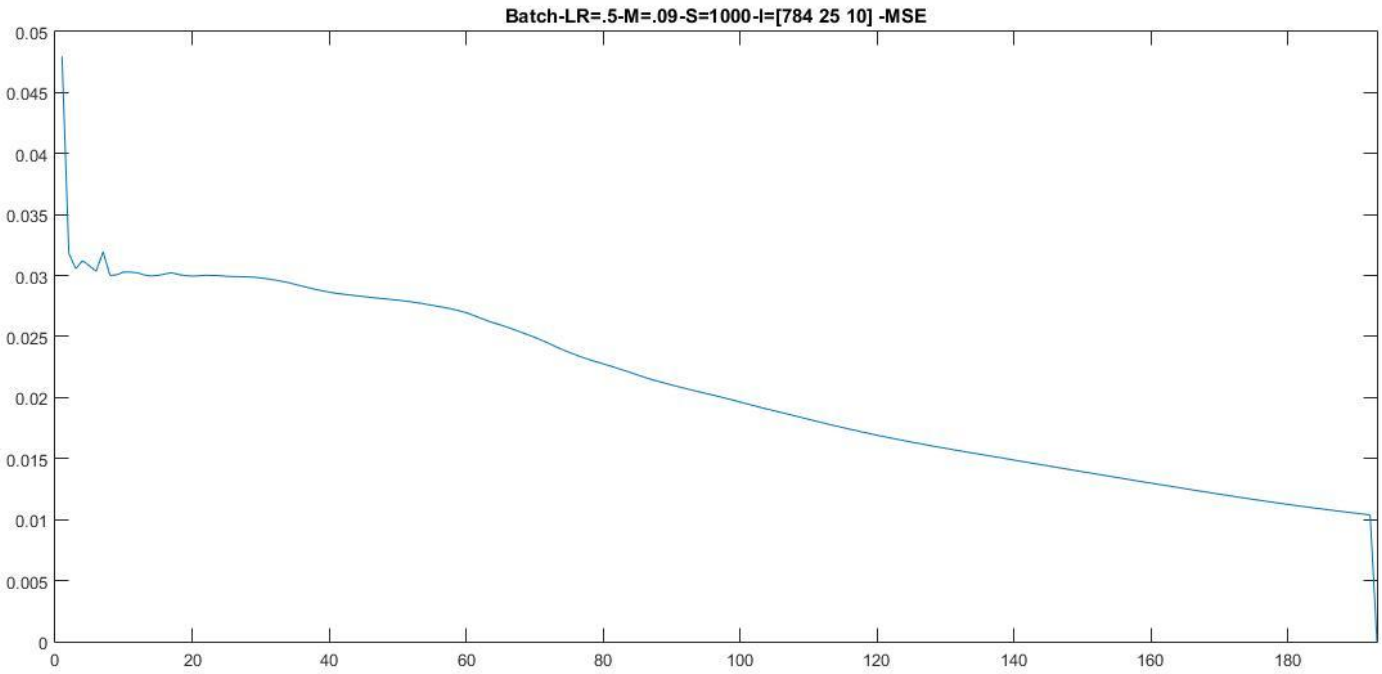
دقت بدست آمده در هر دو روش برروی نمونه های آموزشی به ۱۰۰ میرسد و در هر دو روش بر روی داده های تست و ارزیابی به حدود ۸۰درصد میرسد و تقریبا هر دو روش به دقت یکسانی دست پیدا کرده اند.(دقت بدست آمده در هر دو روش تقریبا یکسان می باشد)..

روش Momentum

نمدار دقت بر حسب epoch:



نمودار MSE بر حسب epoch:



همانطور که مشاهده می کنید دقت بدست آمده با روش Momentum چندان تفاوتی با روش دسته ای بدون استفاده از این ویژگی ندارد و تقریبا دقت در هردو روش یکسان است، اما سرعت همگرایی روش دسته ای با Momentum خیلی بهتر از روش دسته ای بدون استفاده از این ویژگی است و ما در اینجا بعد از epoch شماره ۸۰به دقت تقریبی ۱۰۰ درصد در نمونه های آموزشی دست پیدا میکنیم، که در روش دسته ای بدون استفاده از این ویژگی در epoch شماره ۱۰۰۰ به دقت ۱۰۰ درصد بر روی نمونه های آموزشی دست پیدا کردیم.

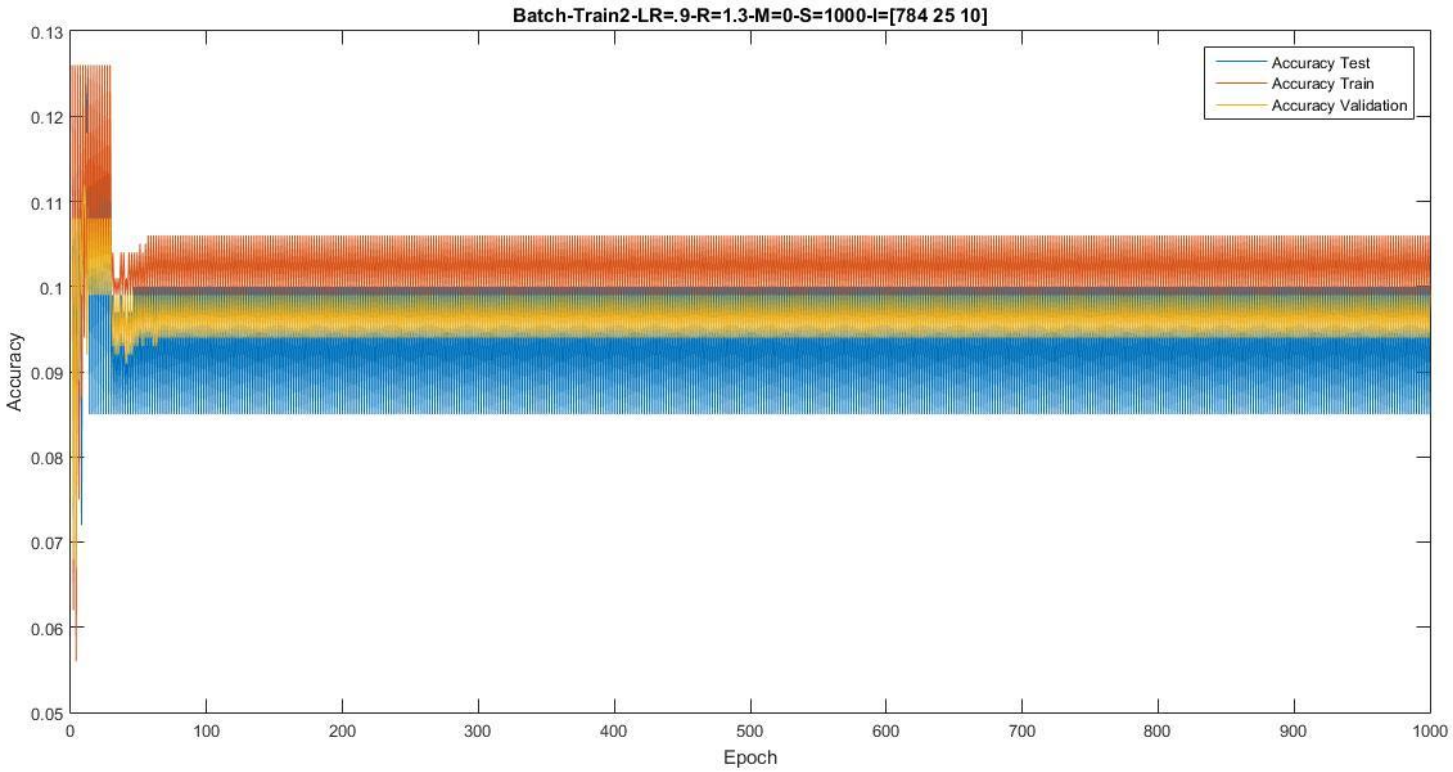
دقت ها یکسان و سرعت همگرایی روش دسته ای با Momentum خیلی بهتر از روش دسته ای بدون استفاده از این ویژگی می باشد.

روش Regularization

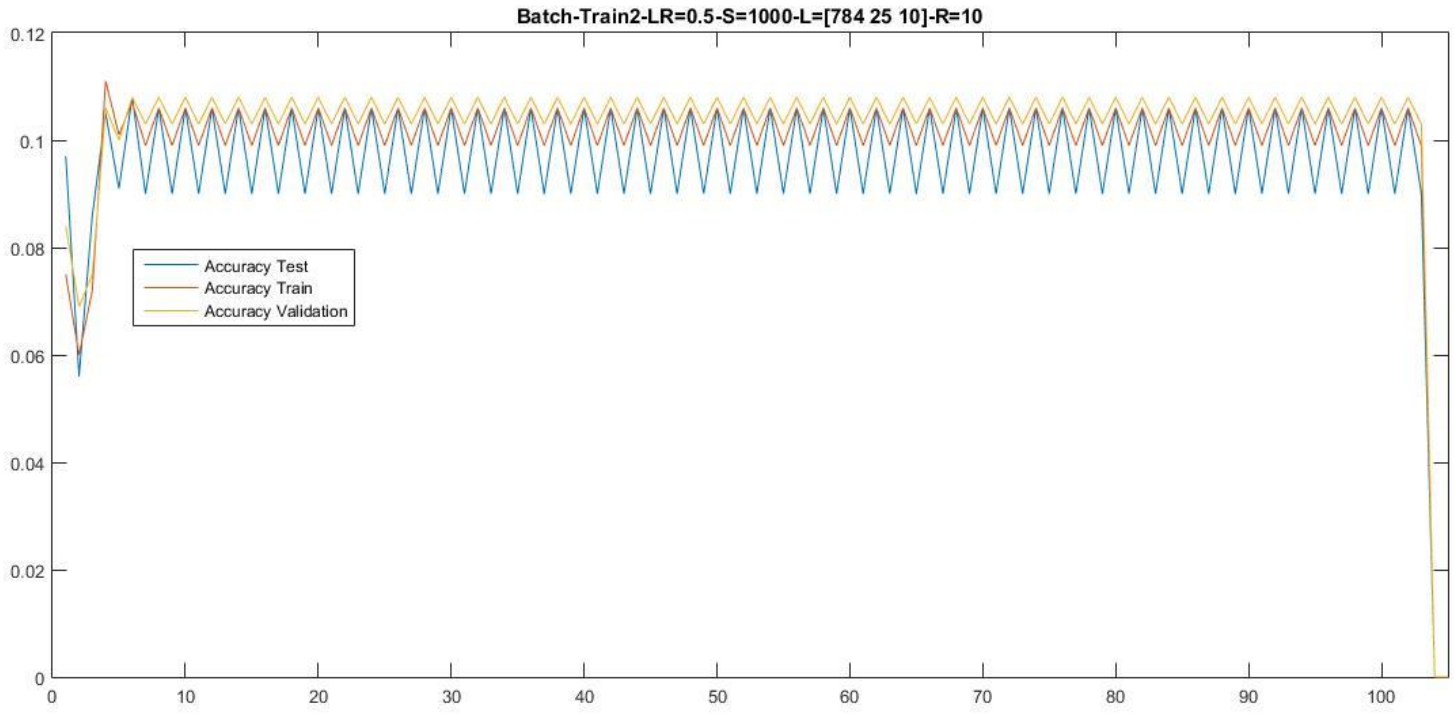
نمودا دقت بر حسب epoch تست اول با مقدار Regularization زیاد:

توجه داشته باشید که در پیاده سازی من R به صورت "lambda*learning_rate" می باشد.

R=1.3و learning_rate=0.9 یعنی ضریب منتظم سازی برابر با learning_rate*lambda یعنی (0.9)*(1.3) می باشد.

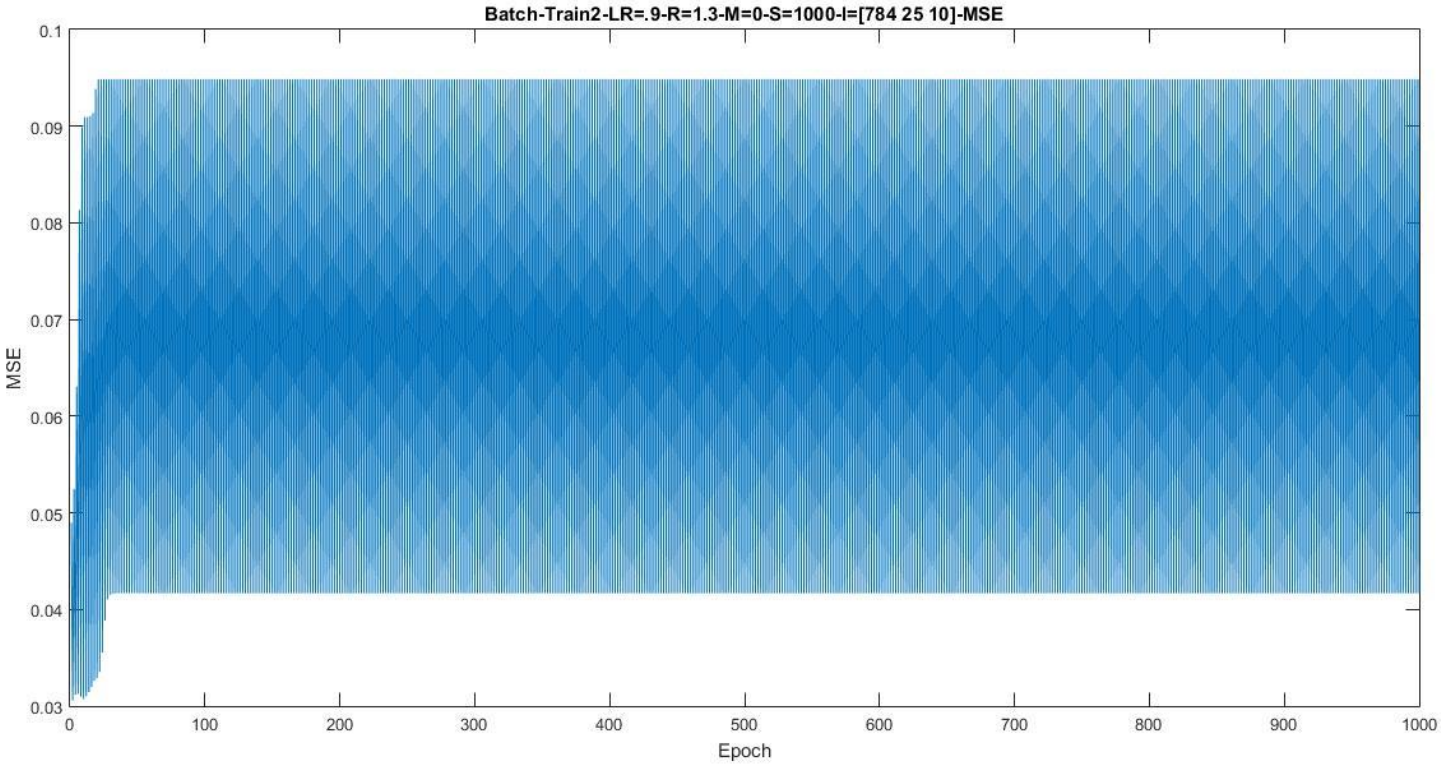


مقدار R=10

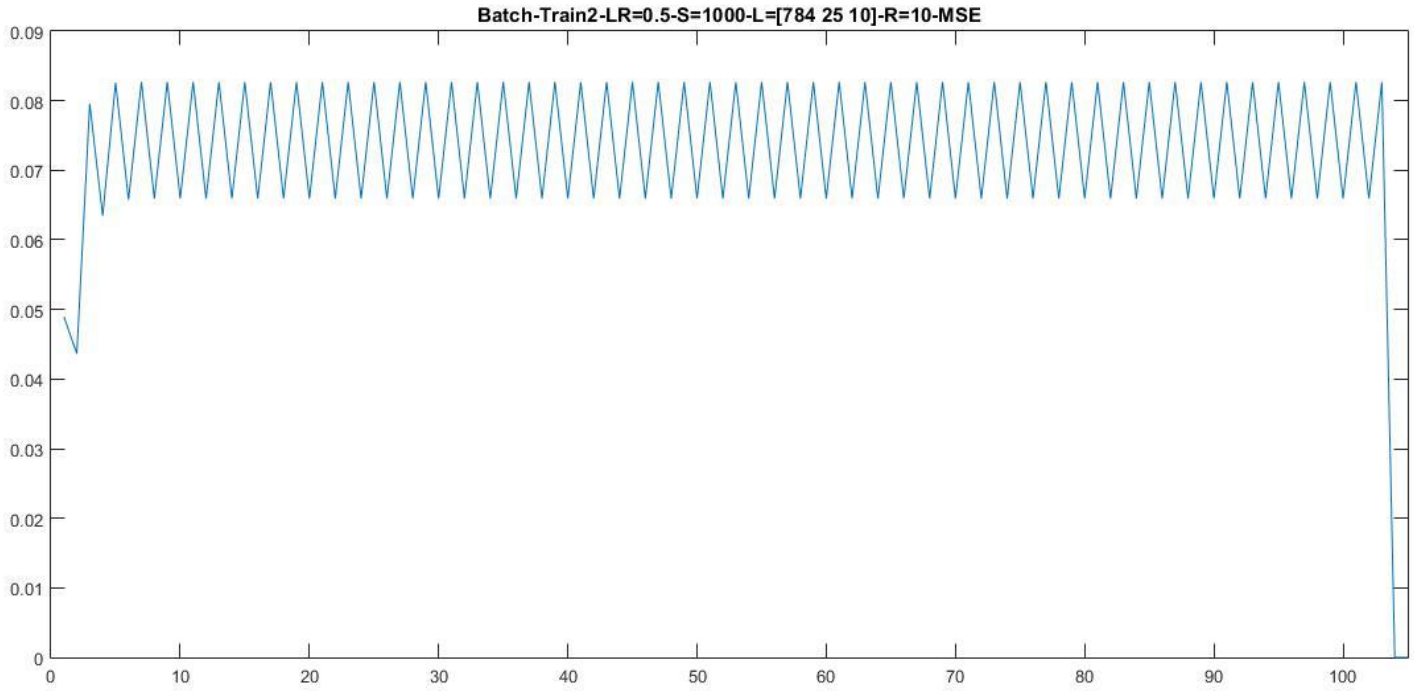


نمودا MSE بر حسب epoch تست اول با مقدار Regularization زیاد:

R=1.3

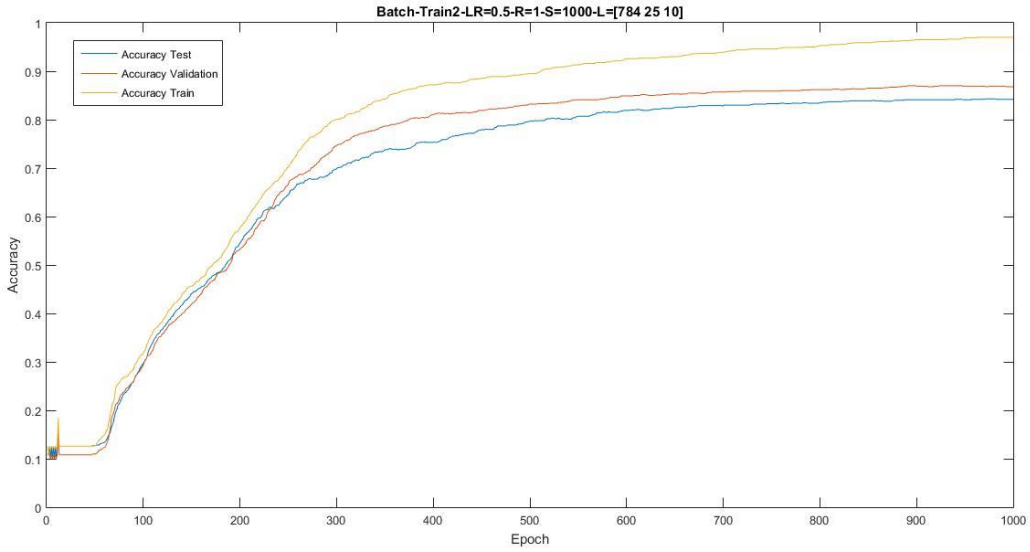


R=10



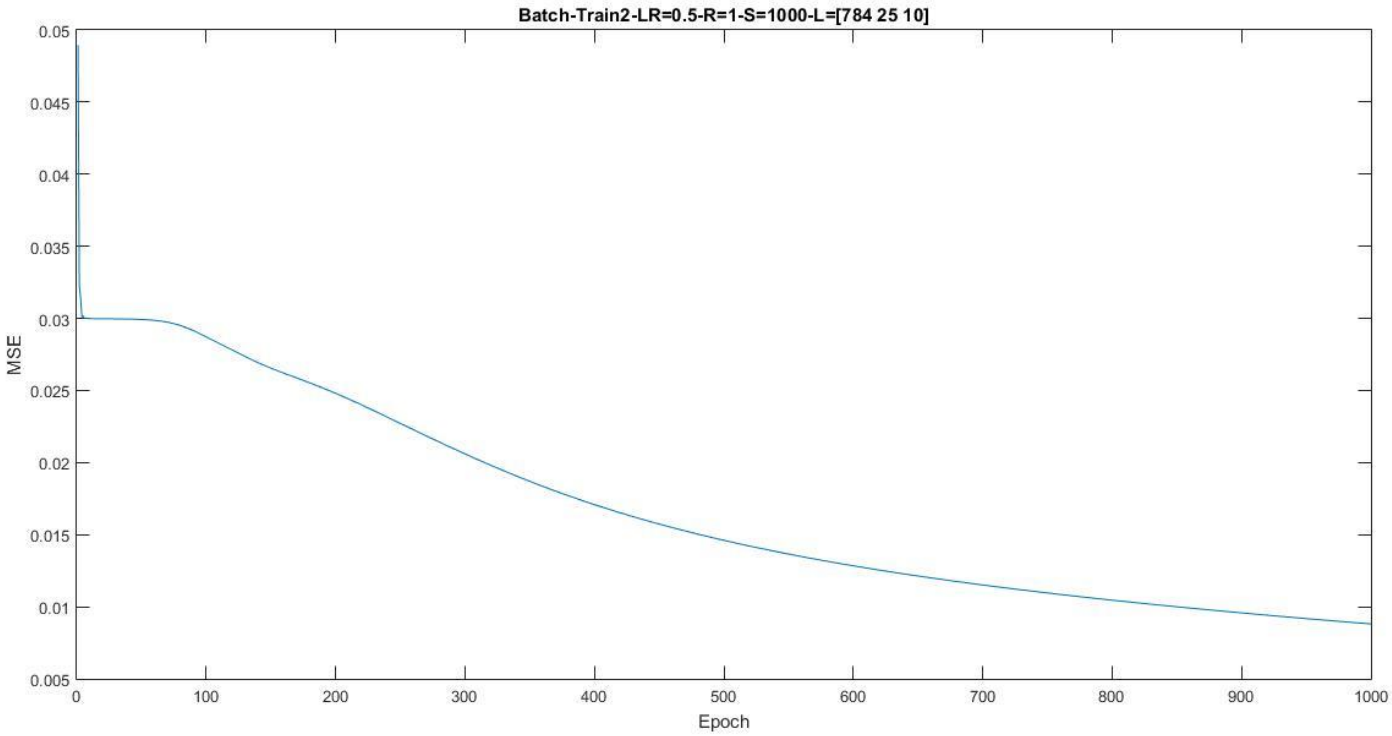
نمودار دقت بر حسب epoch در تست دوم با مقدار Regularization کمتر:

learning_rate=0.5 و R=1.0



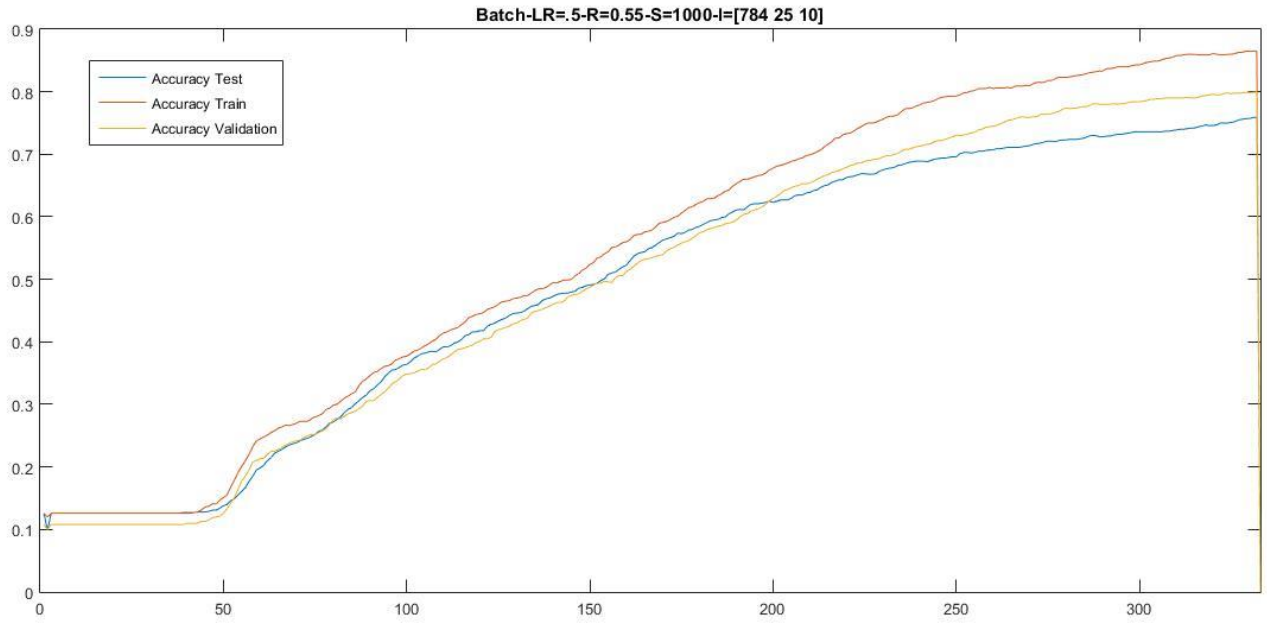
نمودار MSE بر حسب epoch در تست دوم با مقدار Regularization کمتر:

learning_rate=0.5 و R=1.0



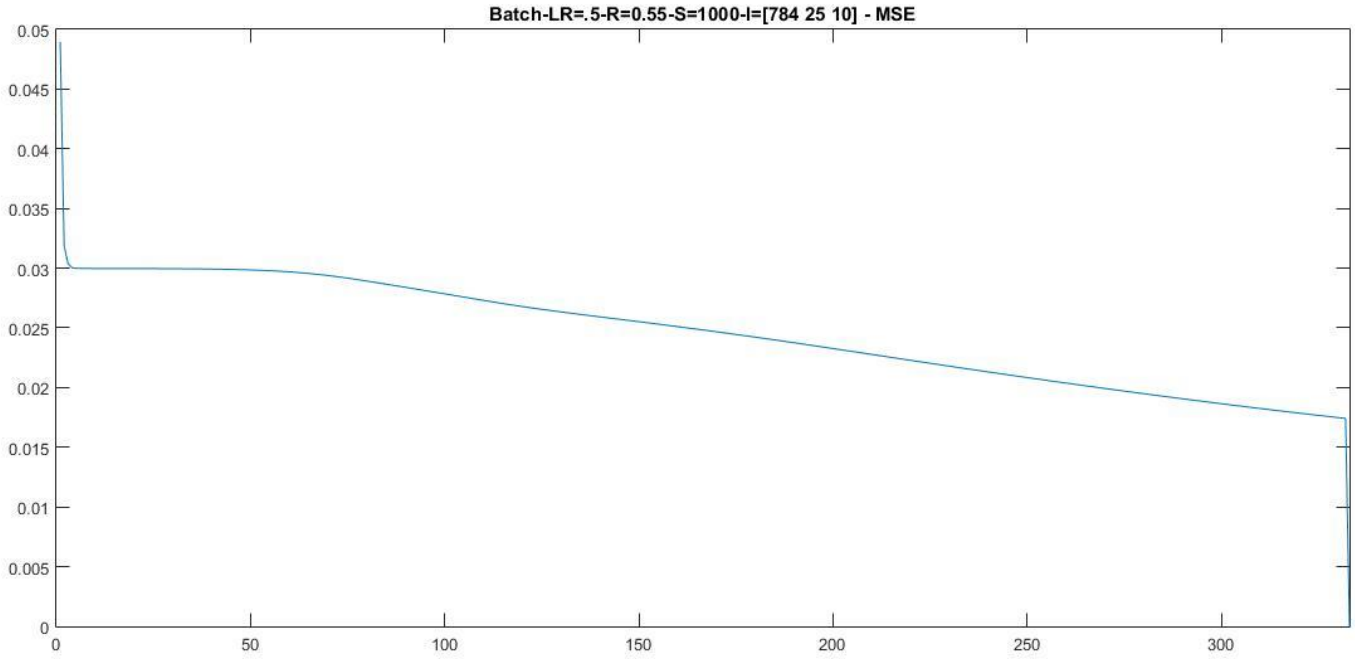
نمودار دقت بر حسب epoch در تست سوم با مقدار Regularization کمتر:

learning_rate=0.5 و R=0.55



نمودار MSE بر حسب epoch در تست سوم با مقدار Regularization کمتر:

learning_rate=0.5 و R=0.55



همانطور که در نمودار ها مشاهده می کنید ضریب منتظم سازی زیاد از رشد وزن ها به مقدار زیادی می کاهد و سعی می کند مقدار وزن ها را به صفر برساند و در نمودار مشاهده می کنید که تقریبا دقت در امتداد یک خط راست در حدود ۱۰ درصد نگاه داشته شده است، در واقع حرکت زیگزاگی دقت نمایانگر این است که شبکه سعی داشته که دقت را بهبود ببخشد و یک تغییر در جهت افزایش دقت بوجود می آورد و تا حدی دقت رشد پیدا میکند اما همین مقدار تغییر در جهت مخالف بوسیله منتظم سازی اعمال می گردد و حرکت زیگزاگی حول یک خط راست را در محدوده دقت کم ایجاد میکند.

اگر ضریب منتظم سازی زیاد باشد سعی میکند وزن ها را کم کرده و در واقع نمودار را به صورت یک خط صاف در آورد که به معنی `under fit` می باشد.

اگر خیلی هم کم باشد بدلیل اینکه نمیتواند تاثیر گذار باشد می توان از اثر آن چشم پوشی کرد و در اینجا `over fit` اتفاق می افتد.

اگر مقدار معقولی برای ضریب منتظم سازی انتخاب شود، باعث میشود از رشد بی رویه برازش یعنی از `fit`شدن به طور کامل با داده های آموزشی جلوگیری گردد و تا حدودی از `over fit` اجتناب گردد.

منتظم سازی تاثیر چندانی در سرعت همگرایی ندارد، اما باعث میشود رشد دقت بر روی نمونه های آموزشی کمتر گردد.

Neural Network toolbox

در این قسمت ابتدا یک داده ها را فراخوانی کردیم و سپس یک شبکه `feed forward` با یک لایه پنهان ایجاد کردیم، و بعد از اینکه توابع لجستیک هر خروجی را و همچنین روش یادگیری و سایر پارامتر ها را مشخص کردیم با دستور `train` به آموزش شبکه پرداختیم، در ادامه تصاویر کد مربوط به این کار قرار داده میشود:

فراخوانی داد ها و تجمیع انها در یک ماتریس:

```
%% ============ Load data  ========
clear all;
clc;
Load_Data;
validation_images = images(:,IND);
validation_labels = labels(IND);

% images(:,:) = [];
% labels(IND) = [];
total_images = zeros(784,70000);
total_images(:,1:end-10000)= images;
total_images(:,60001:end)= timages;
total_labels = zeros(70000,1);
total_labels(1:60000,:) = labels;
total_labels(60001:end,:) = tlabels;

% images(:,IND) = [];
% labels(IND) = [];
```

مقدار دهی اولیه به پارامتر ها:(روش آموزش، اندازه نورون های و لایه های شبکه و غیره)

```
%% ======  Initializing...
vector_target = Target_to_Vector(total_labels);
% Inputs + Targets
inputs = total_images;
targets = vector_target;
% Choose a Logestic Function
TF={'logsig','logsig'};
% Choose a Training Function
%For a list of all training functions type: help nntrain
%'trainlm' is usually fastest(Levenberg-Marquardt)
%'trainbr' takes longer but may be better for challenging problems.
%'trainscg' uses less memory. Suitable in low memory situations.
% net.trainFcn = 'trainlm';
BTF = 'trainb';% batch training
% Create a feed forward neural network with hidden Layer Size 15
hiddenLayerSize = 15;
mlp = newff(inputs,targets,hiddenLayerSize,TF,BTF);
```

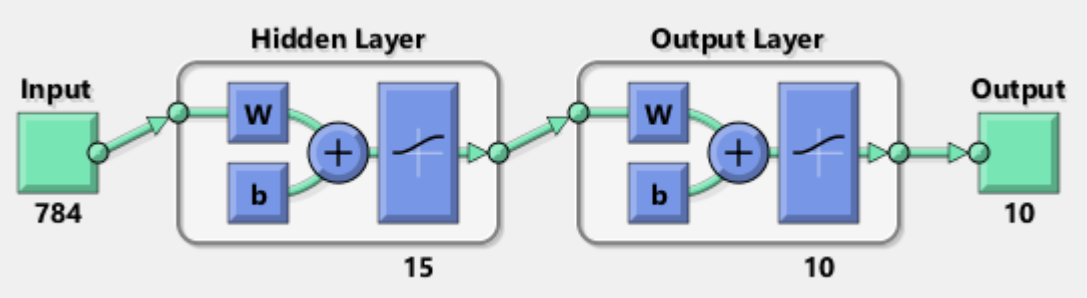
مشخص کردن داده های آموزشی و ارزیابی و تست و تابع هزینه و ...

```
%% Set Training, Validation & Test Data with Performance Function & Plots
% Setup Division of Data for Training, Validation, Testing
% For a list of all data division functions type: help nndivide
mlp.divideFcn = 'dividerand';  % Divide data randomly
mlp.divideMode = 'sample';  % Divide up every sample
mlp.divideParam.trainRatio = 7143/10000;
mlp.divideParam.valRatio = 1428/10000;
mlp.divideParam.testRatio = 1428/10000;

% Choose a Performance Function
% For a list of all performance functions type: help nnperformance
mlp.performFcn = 'mse';  % Mean squared error
% Choose Plot Functions
% For a list of all plot functions type: help nnplot
mlp.plotFcns = {'plotperform','ploterrhist','plotregression','plotfit'};

mlp.trainParam.epochs=1000;% number of epoch's
% net.trainParam.goal=1e-8;% maximum goal for stop rule
mlp.trainParam.max_fail=20;% maximum failof training and validation set
view(mlp);% shows the MLP
```

آموزش و تست شبکه:

```
%% Train the Network
[mlp,tr] = train(mlp,inputs,targets);
%% Test the Network
y = mlp(inputs);
e = gsubtract(targets,y);
performance = perform(mlp,targets,y);
tind = vec2ind(targets);
yind = vec2ind(y);
percentErrors = sum(tind ~= yind)/numel(tind);
```

شبکه طراحی شده به صورت زیر می باشد:



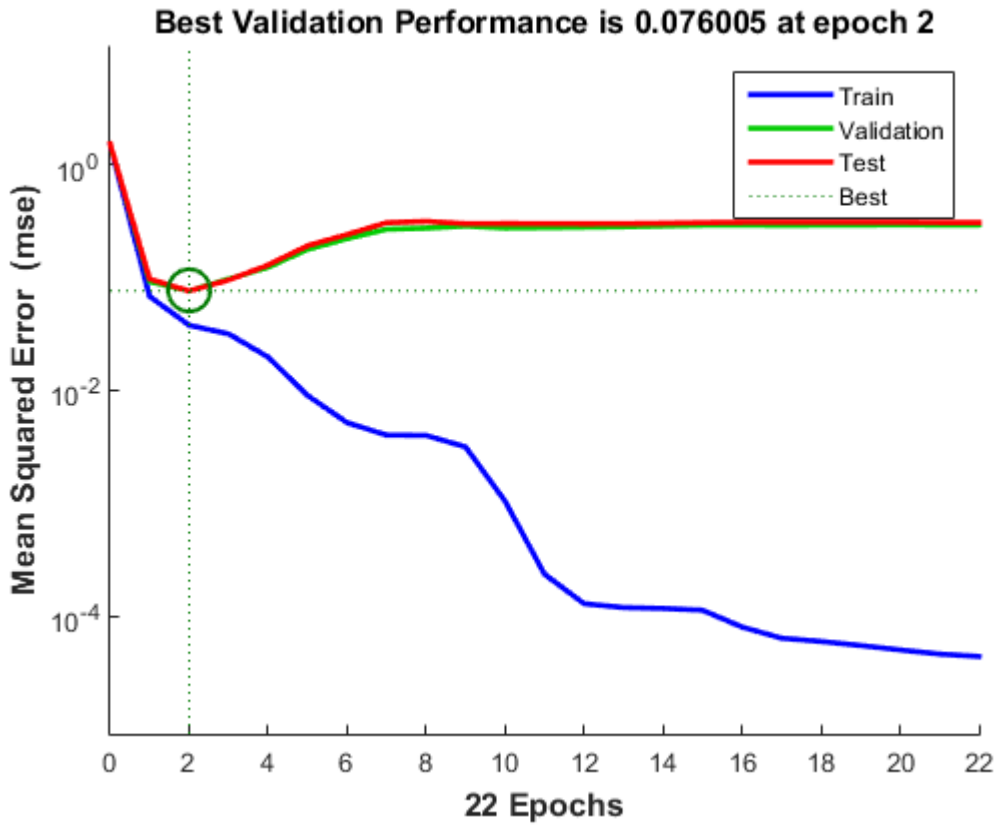
و نتایج تست شبکه به صورت زیر می باشد:

Algorithms

Data Division:	Random	(dividerand)
Training:	Batch Weight/Bias Rule	(trainb)
Performance:	Mean Squared Error	(mse)
Calculations:	MATLAB	

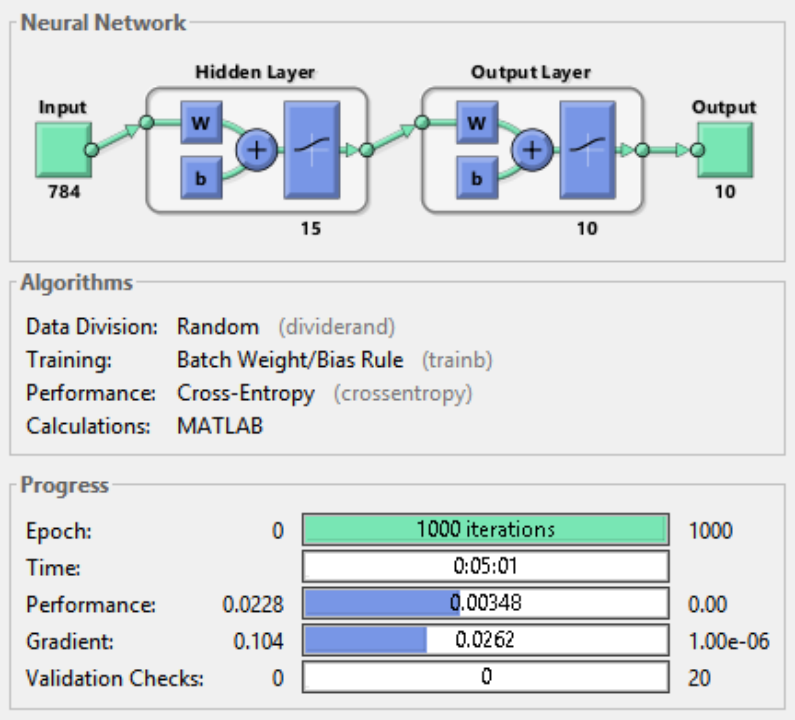
Progress

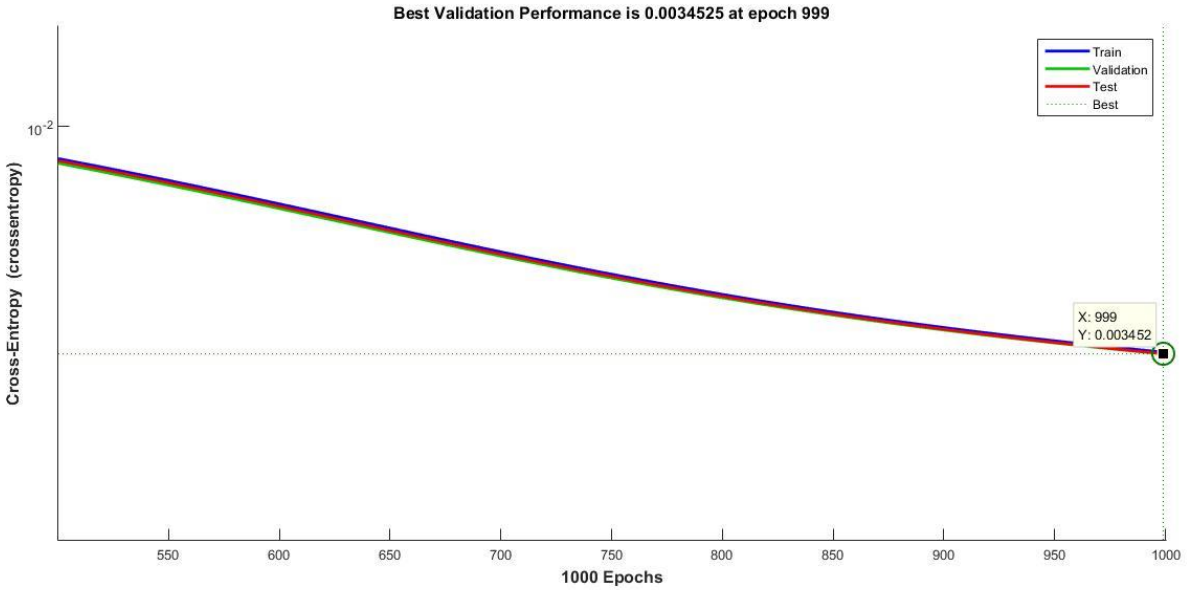
Epoch:	0	1000 iterations	1000
Time:		0:04:45	
Performance:	0.644	0.293	0.00
Gradient:	1.13	0.0666	1.00e-06
Validation Checks:	0	0	20



تغییر تابع هزینه

اگر تابع هزینه را از MSE به Cross Entropy تغییر دهیم نتایج زیر را خواهیم داشت:





که مشاهده میکنیم تنها با تغییر این تابع مقدار کارایی ما به طرز چشم گیری افزایش می یابد. و لذا اگر بسته به نیاز بهترین تابع ارزیابی را انتخاب کنیم نتیجه مناسبتری خواهیم گرفت.

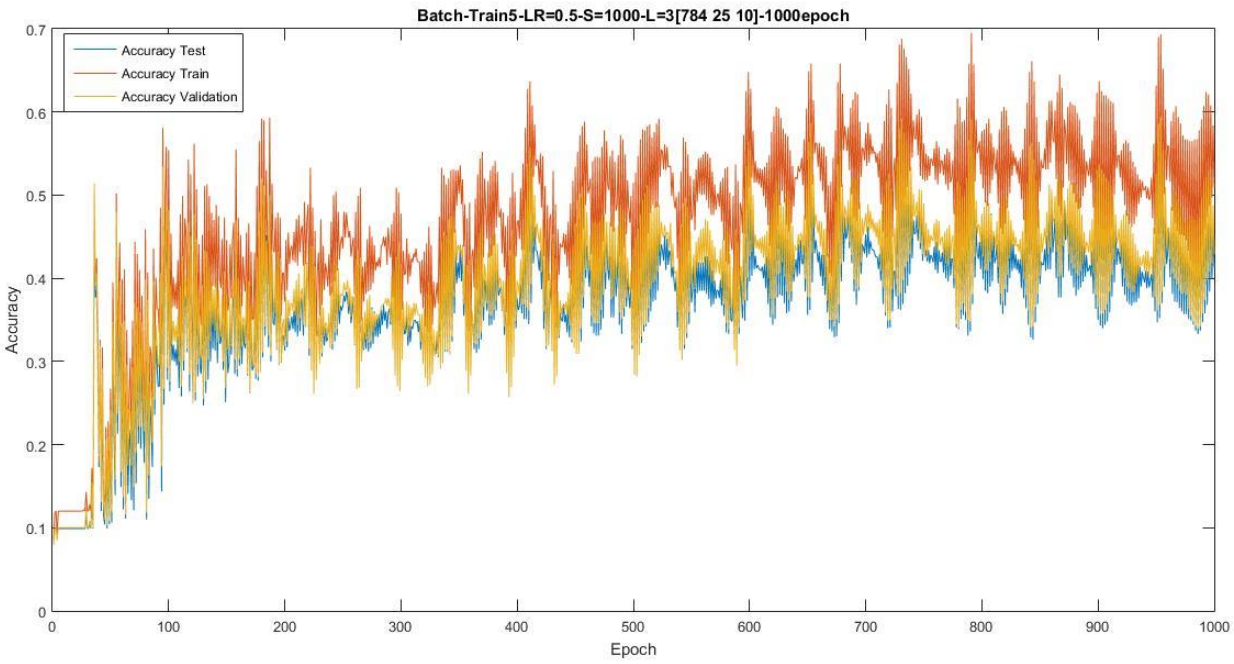
اگر خروجی شبکه را نگاه کنیم دید بهتری بدست می آوریم:

Field	Value
trainFcn	'trainb'
trainParam	*1x1 struct*
performFcn	'crossentropy'
performParam	*1x1 struct*
derivFcn	'defaultderiv'
divideFcn	'dividerand'
divideMode	'sample'
divideParam	*1x1 struct*
trainInd	*1x50006 double*
valInd	*1x9997 double*
testInd	*1x9997 double*
stop	'User stop.'
num_epochs	242
trainMask	*1x1 cell*
valMask	*1x1 cell*
testMask	*1x1 cell*
best_epoch	241
goal	0
states	*1x6 cell*
epoch	*1x243 double*
time	*1x243 double*
perf	*1x243 double*
vperf	*1x243 double*
tperf	*1x243 double*
val_fail	*1x243 double*
best_perf	0.0290
best_vperf	0.0291
best_tperf	0.0290

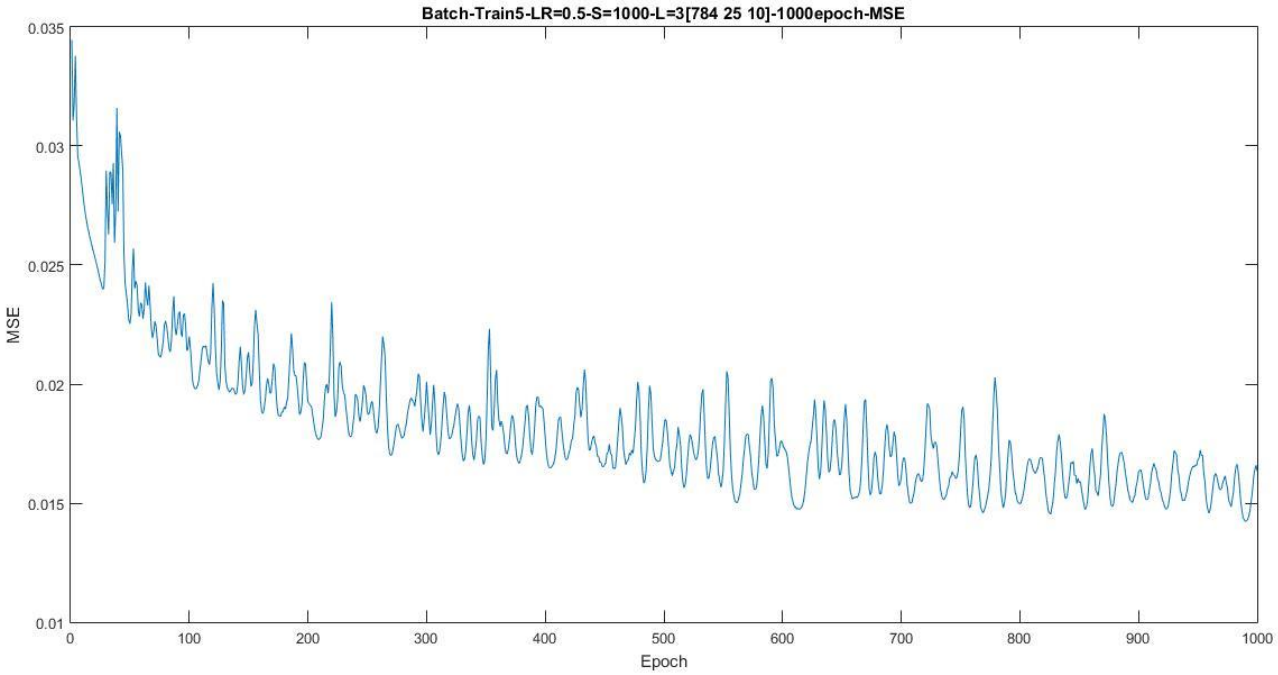
تغییر تابع لجستیک مورد استفاده در روش دسته ای در شبکه پیاده سازی شده:

تابع مورد استفاده را از سیگموید به `tanh` تغییر دادم و نتایج به صورت زیر می باشند:

نمودار دقت بر حسب epoch:



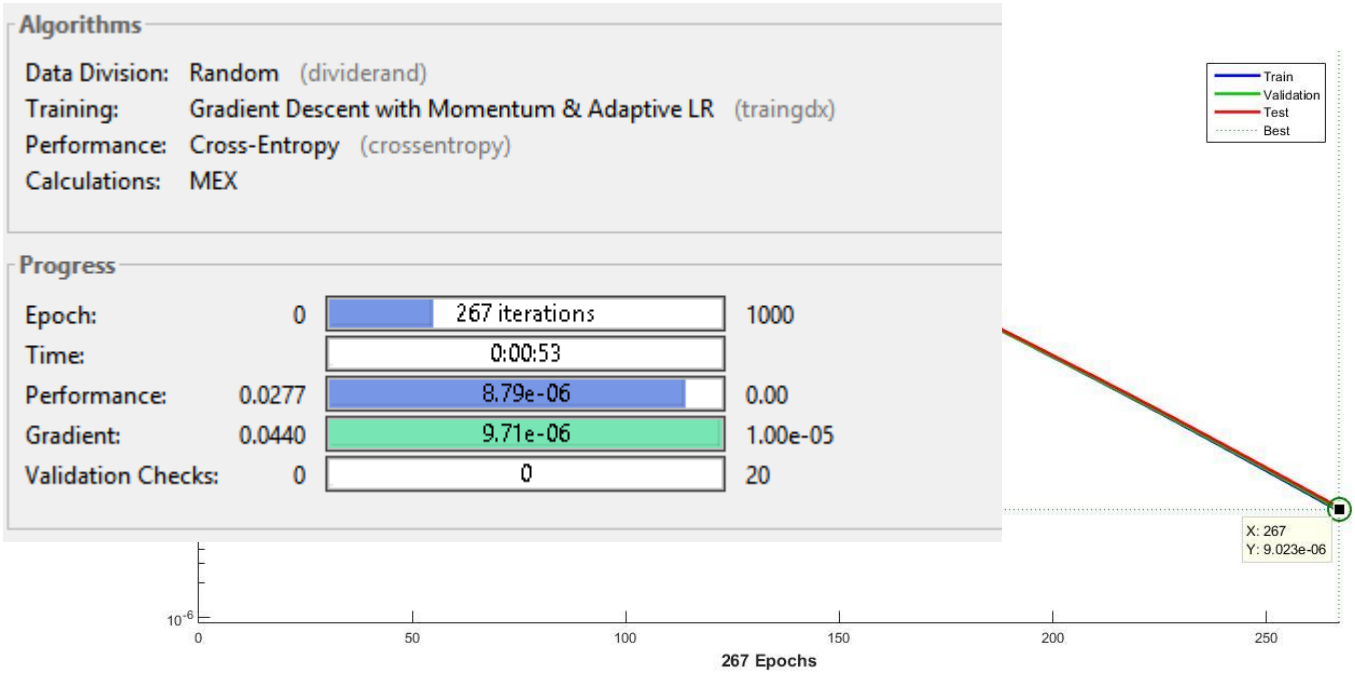
نمودار MSE بر حسب epoch:



تغییر روش آموزش

با تغییر روش آموزش نیز نتایج بهتری می گیریم:

روش آموزش 'traingdx' هست یعنی نزول گردایانی با Momentum و نرخ یادگیری وفق پذیر



trainFcn	'traingdx'
trainParam	*1x1 struct*
performFcn	'crossentropy'
performParam	*1x1 struct*
derivFcn	'defaultderiv'
divideFcn	'dividerand'
divideMode	'sample'
divideParam	*1x1 struct*
trainInd	*1x50006 double*
valInd	*1x9997 double*
testInd	*1x9997 double*
stop	'Minimum gradient r...
num_epochs	267
trainMask	*1x1 cell*
valMask	*1x1 cell*
testMask	*1x1 cell*
best_epoch	267
goal	0
states	*1x8 cell*
epoch	*1x268 double*
time	*1x268 double*
perf	*1x268 double*
vperf	*1x268 double*
tperf	*1x268 double*
gradient	*1x268 double*
val_fail	*1x268 double*
lr	*1x268 double*
best_perf	8.7886e-06
best_vperf	9.0234e-06
best_tperf	9.4531e-06

روش نزول گرادیانی و Momentum

برای آموزش شبکه به روش نزول گرادیانی با Momentum باید روش آموزش را `'traingdm'` قرار دهیم.

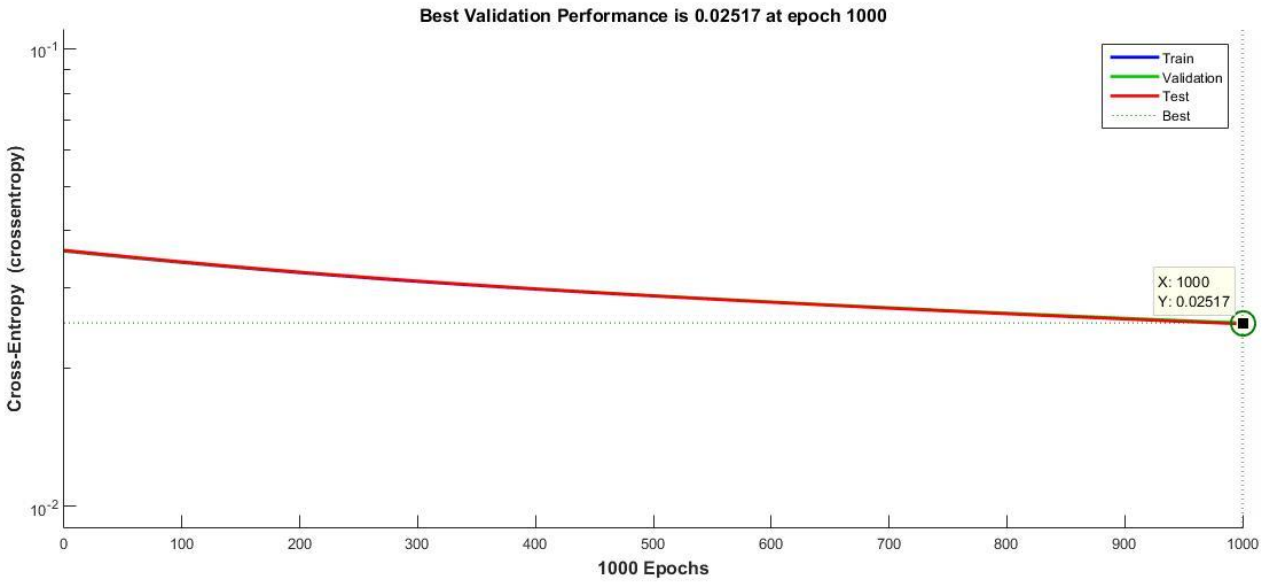
خروجی شبکه:

Algorithms

Data Division:	Random	(dividerand)
Training:	Gradient Descent with Momentum	(traingdm)
Performance:	Cross-Entropy	(crossentropy)
Calculations:	MEX	

Progress

Epoch:	0	1000 iterations	1000
Time:		0:03:34	
Performance:	0.0361	0.0251	0.00
Gradient:	0.0464	0.0240	1.00e-05
Validation Checks:	0	0	20



trainFcn	'traingdm'
trainParam	*1x1 struct*
performFcn	'crossentropy'
performParam	*1x1 struct*
derivFcn	'defaultderiv'
divideFcn	'dividerand'
divideMode	'sample'
divideParam	*1x1 struct*
trainInd	*1x50006 double*
valInd	*1x9997 double*
testInd	*1x9997 double*
stop	'Maximum epoch rea...
num_epochs	1000
trainMask	*1x1 cell*
valMask	*1x1 cell*
testMask	*1x1 cell*
best_epoch	1000
goal	0
states	*1x7 cell*
epoch	*1x1001 double*
time	*1x1001 double*
perf	*1x1001 double*
vperf	*1x1001 double*
tperf	*1x1001 double*
gradient	*1x1001 double*
val_fail	*1x1001 double*
best_perf	0.0251
best_vperf	0.0252
best_tperf	0.0250

به دلیل اینکه شبکه طراحی شده برای تمرین یک در هنگام اجرا تعدا لایه ها و سایر عوامل آن مشخص میشود، لذا اثر تغییر تعدا نورون ها و تعداد لایه ها را در روی شبکه طراحی شده برای تمرین یک آزمایش می کنم:

افزایش تعداد لایه ها در روش آنلاین:

شبکه ای با ۵ لایه:

تعداد نورون های لایه اول یا ورودی ها برابر ۷۸۴

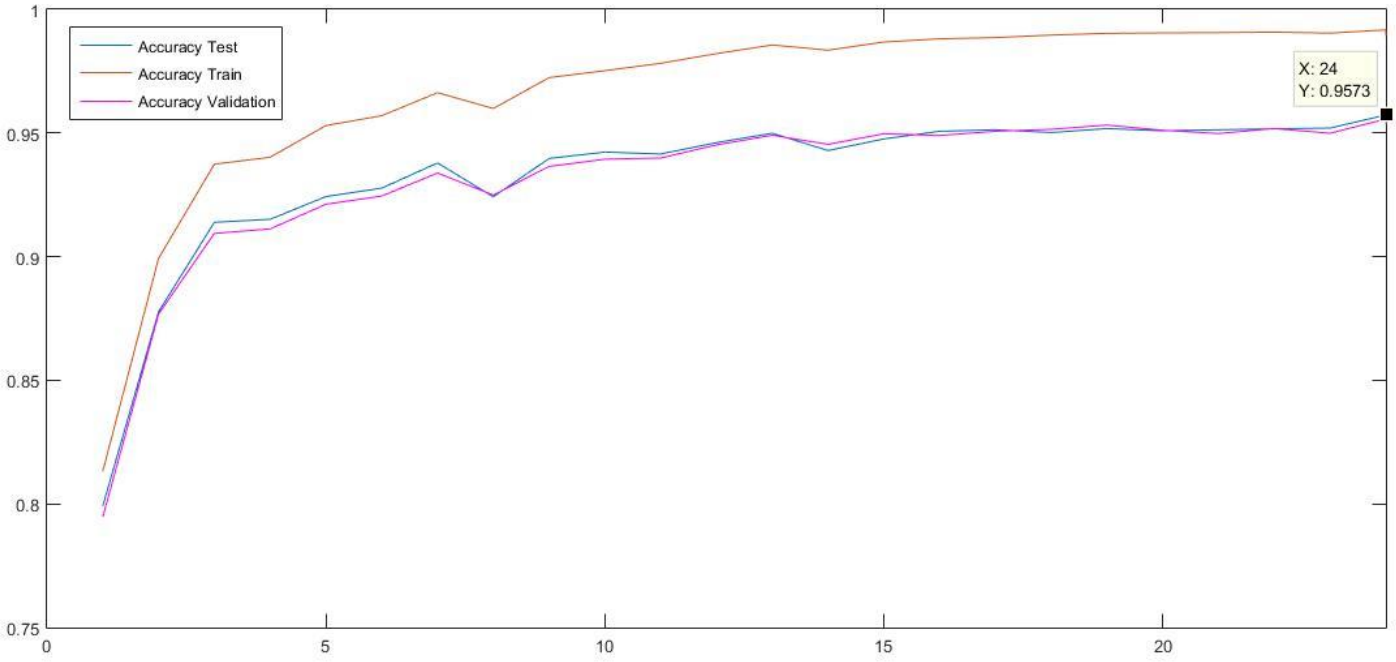
تعداد نورون های لایه دوم ۳۶۲

تعداد نورون های لایه سوم ۱۸۳

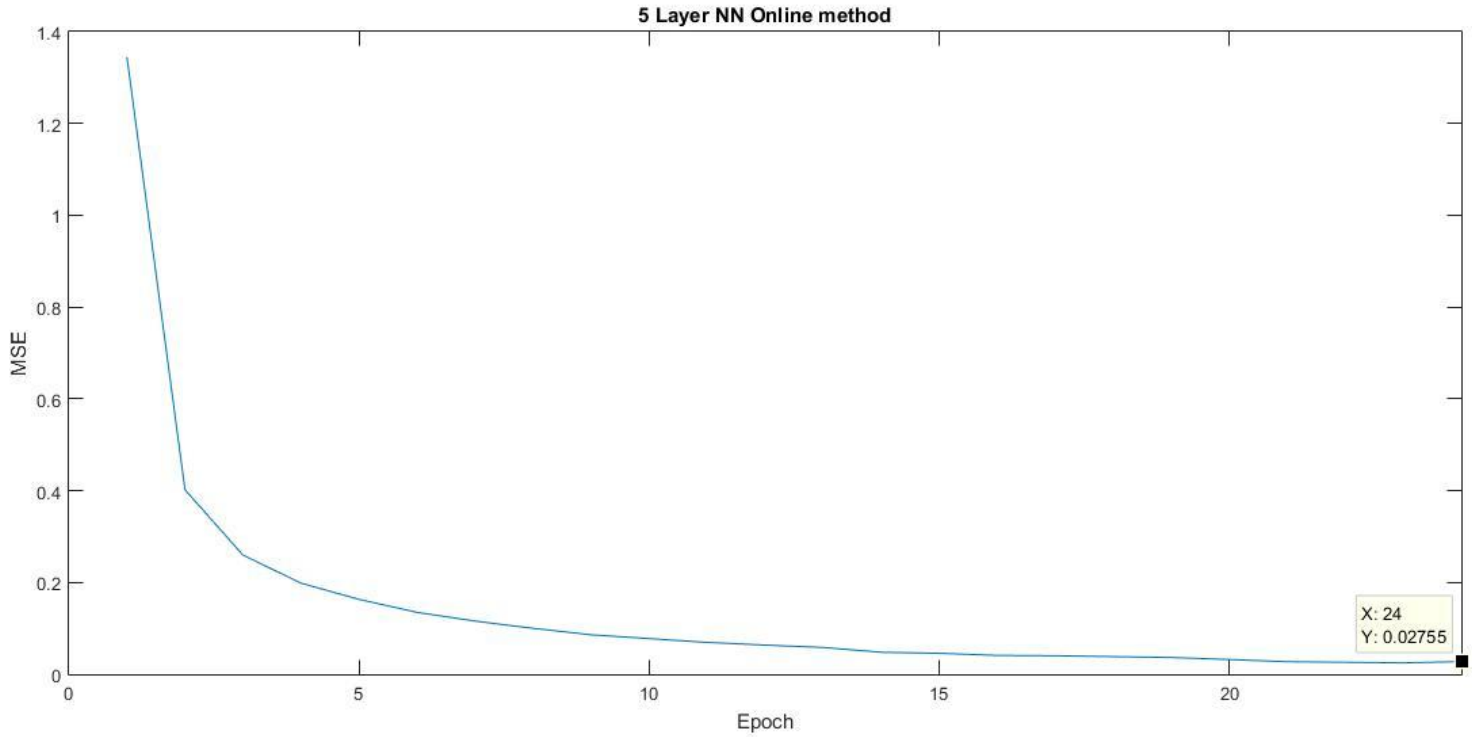
تعداد نورون های لایه چهارم ۶۸

تعداد نورون های لایه پنجم(خروجی) ۱۰

نمودار دقت بر حسب epoch:



نمودار MSE بر حسب epoch:



مشاهده میکنیم که با افزایش تعداد لایه ها در epoch اول دقت کمتر از شبکه ای با تعداد لایه کمتر می باشد اما در epoch های بعدی بیشتر همگرا می گردد و در تعداد epoch کمتری به دقت مورد نظر دست می یابد، اما در نظر بگیرید که باید تعدا نمونه ها زیاد باشد و همچنین سرعت آموزش این شبکه به مراتب کمتر از شبکه ای با لایه های کمتر می باشد و همچنین امکان Over train شدن نیز با افزایش تعداد لایه ها افزایش می یابد و همچنین ممکن است شبکه به خوبی جواب ندهد یعنی توابع مورد استفاده به گونه ای هستند که نمیتوانند پاسخگوی این نیاز باشند و باید با توابعی بهتر جایگزین گردند.

البته در آزمایش هایی که انجام دادم اگر از یک حدی تعداد لایه ها افزایش یابد و یا تعداد نورون ها از یک تعدادی کم یا زیاد شوند شبکه را نمیتوان آموزش داد! که فکر میکنم به این دلیل باشد که یا ویژگی هایی زیادی را که چندان هم در تصمیم گیری ما موثر نیستند انتخاب می کنیم و یا اینکه کمتر از مقدار مورد نیاز برای تصمیم گیری ویزگی ها را انتخاب می کنیم.

تست افزایش تعداد لایه ها در روش دسته ای:

شبکه ای با ۴ لایه و آموزش به روش دسته ای:

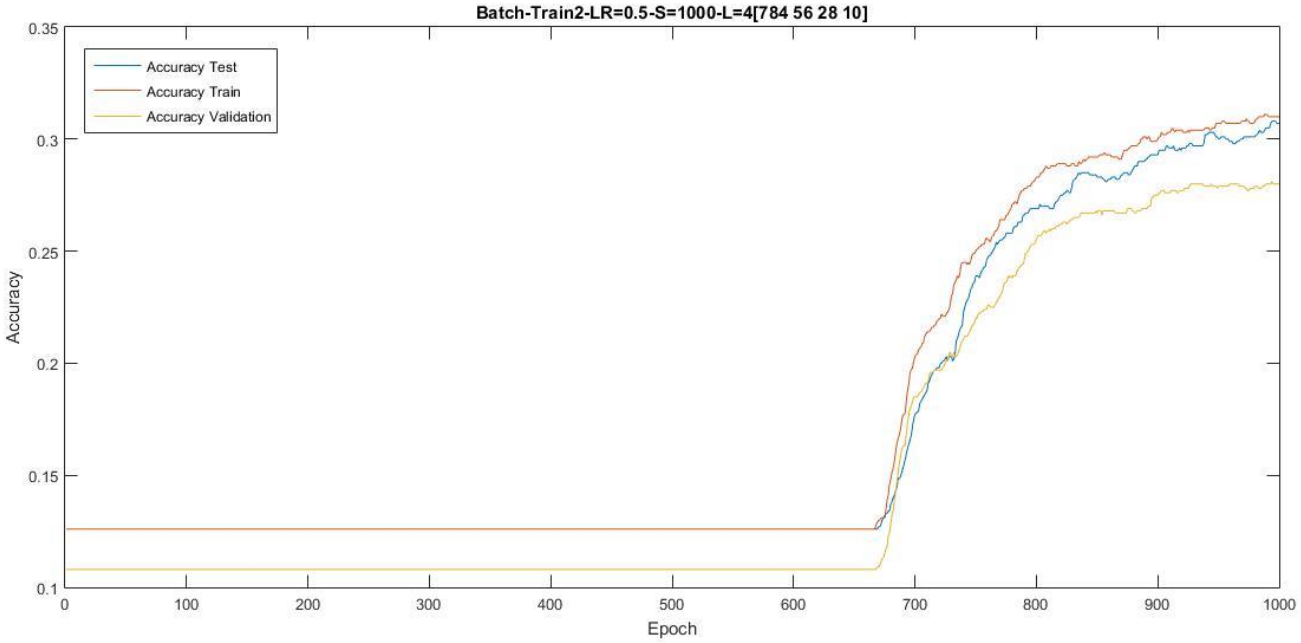
تعداد نورون های لایه اول یا ورودی ها برابر ۷۸۴

تعداد نورون های لایه دوم ۵۶

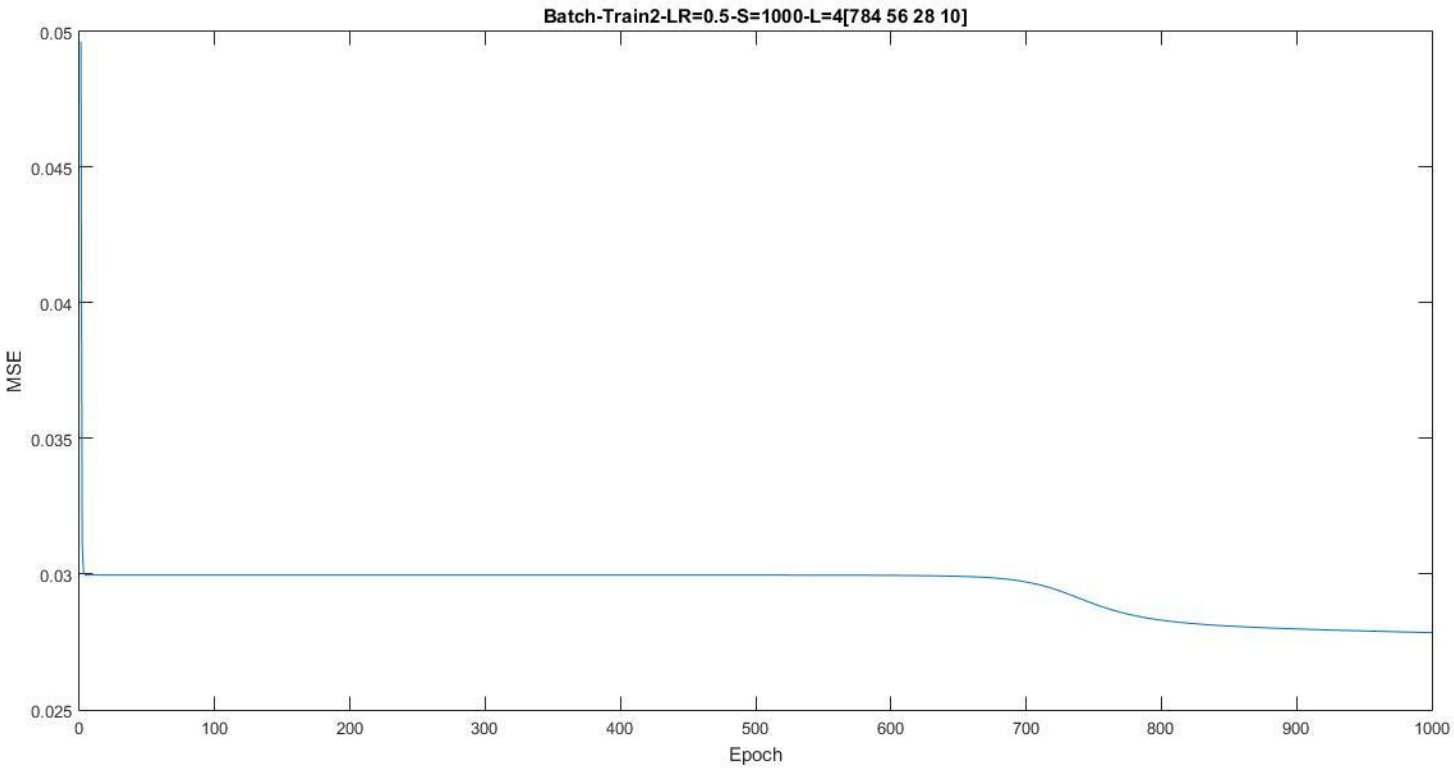
تعداد نورون های لایه سوم ۲۸

تعداد نورون های لایه چهارم(خروجی) ۱۰

نمودار دقت بر حسب epoch:



نمودار MSE بر حسب epoch:



در این روش سرعت همگرایی به شدت افت پیدا میکند!

و بدلیل اینکه زمان آموزش این شبکه خیلی زیاد می باشد، لذا شبکه را تا epoch۱۰۰۰ آموزش دادیم و می بینیم که تا حدود epoch شماره۷۰۰ دقت ثابت است و تغییری نمی کند، اما بعد از این epoch یکدفعه رشد دقت افزایش می یابد و تا epoch شماره ۱۰۰۰ که که من شبکه را تست کردم افزایش دقت ادامه داشته لذا می توان نتیجه گرفت که سرعت همگرایی کم میشود، دقت بهبود می یابد یا حداقل کمتر از شبکه ای با تعداد لایه کمتر نمیشود، البته همانطور که قبلا گفتم این افزایش و کاهش یک حد بالا و پایین دارد.

شبکه ای با ۷ لایه و روش دسته ای:

لایه ورودی ۷۸۴

لایه اول ۱۰

لایه دوم ۱۰

لایه سوم ۱۰

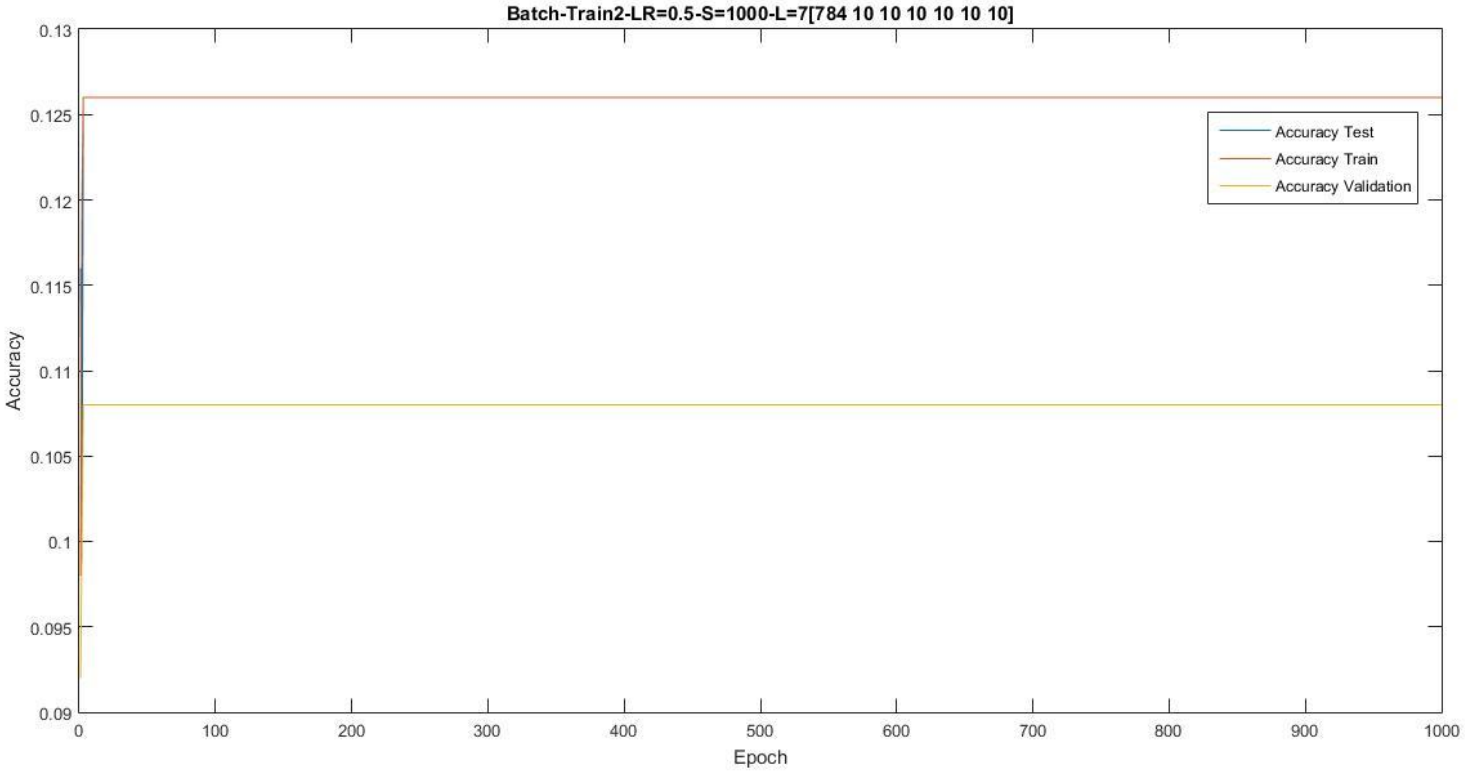
لایه چهارم ۱۰

لایه پنجم ۱۰

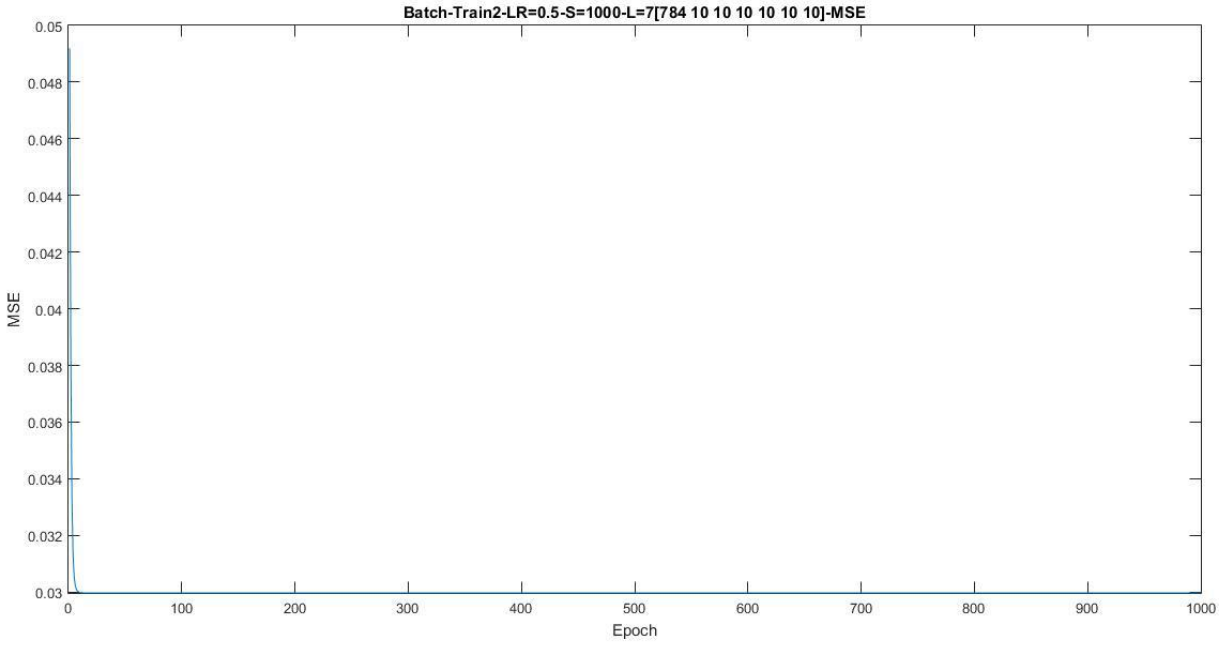
لایه ششم ۱۰

لایه هفتم ۱۰(خروجی)

نمودار دقت بر حسب epoch:



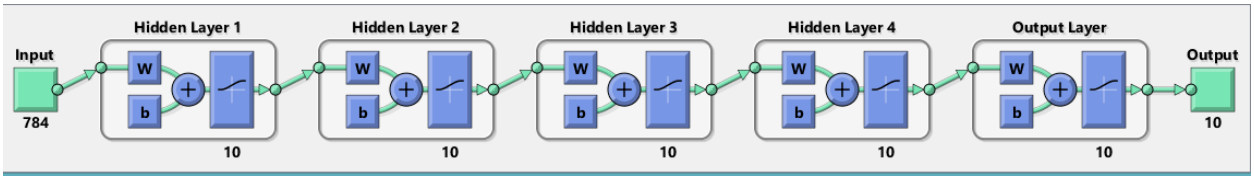
نمودار MSE بر حسب epoch:

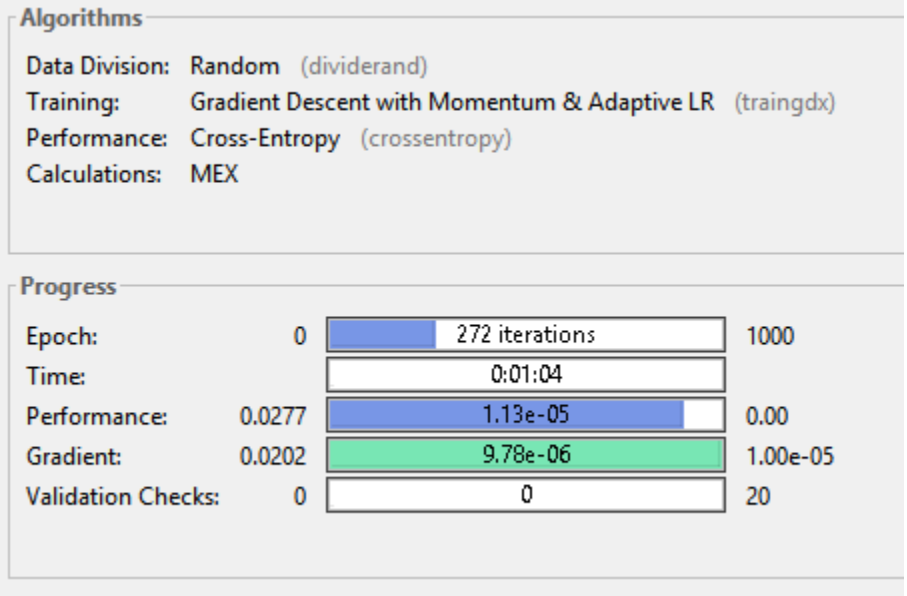


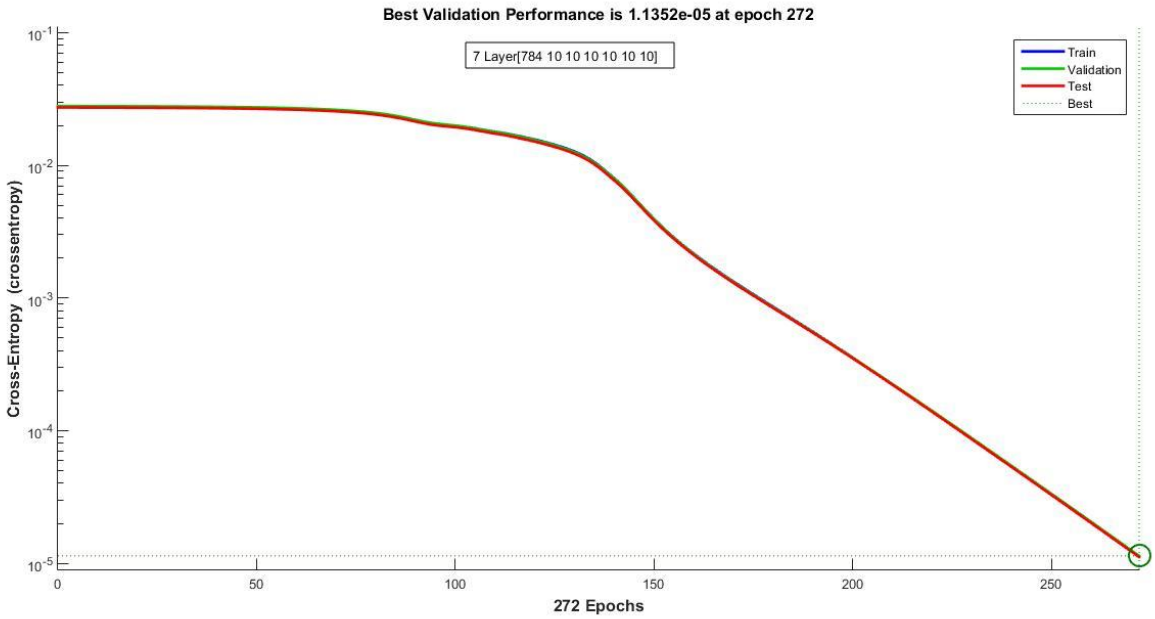
همانطور که مشاهده می کنید در این شبکه دقت از حدود ۱۰ درصد شروع شده و ثابت مانده است و به نظرم برای نتیجه گرفتن در این شبکه باید شبکه را حداقل تا `epoch 10000` آموزش داد!

نمونه ای از افزایش تعداد لایه ها در شبکه بوسیله Toolbox:

شبکه ای با ۷ لایه و Cross Entropy:







trainFcn	'traingdx'
trainParam	*1x1 struct*
performFcn	'crossentropy'
performParam	*1x1 struct*
derivFcn	'defaultderiv'
divideFcn	'dividerand'
divideMode	'sample'
divideParam	*1x1 struct*
trainInd	*1x50006 double*
valInd	*1x9997 double*
testInd	*1x9997 double*
stop	'Minimum gradient r...
num_epochs	272
trainMask	*1x1 cell*
valMask	*1x1 cell*
testMask	*1x1 cell*
best_epoch	272
goal	0
states	*1x8 cell*
epoch	*1x273 double*
time	*1x273 double*
perf	*1x273 double*
vperf	*1x273 double*
tperf	*1x273 double*
gradient	*1x273 double*
val_fail	*1x273 double*
lr	*1x273 double*
best_perf	1.1255e-05
best_vperf	1.1352e-05
best_tperf	1.1141e-05

شبکه ای با ۱۵ لایه

و Cross

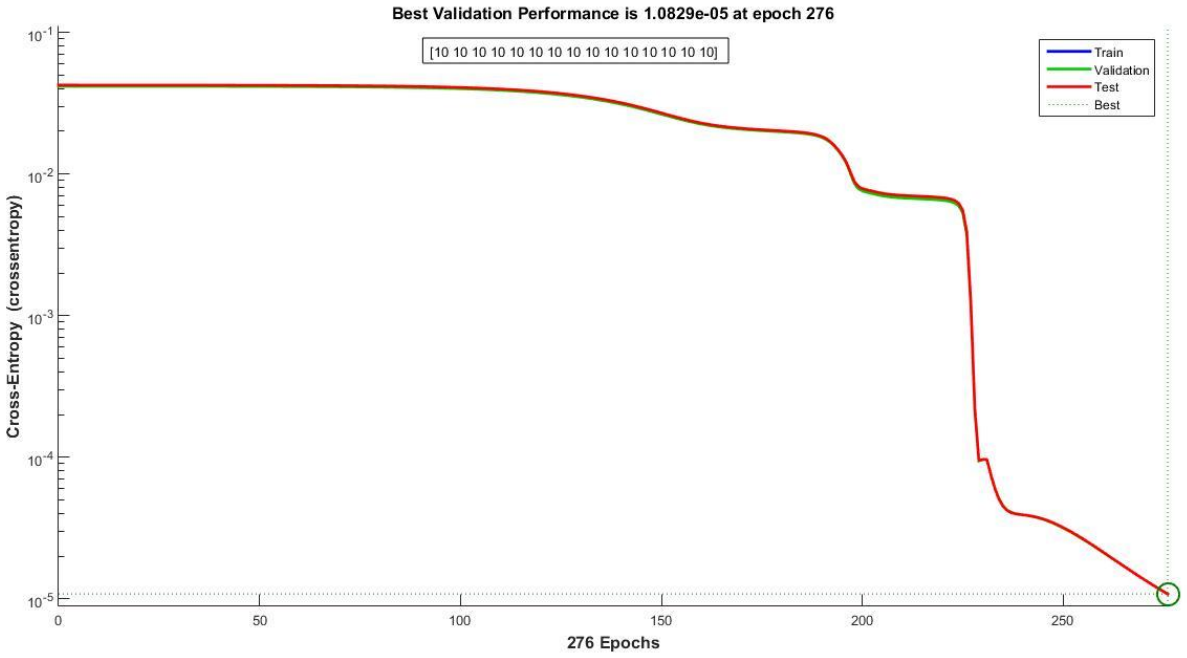
Entropy:

Algorithms

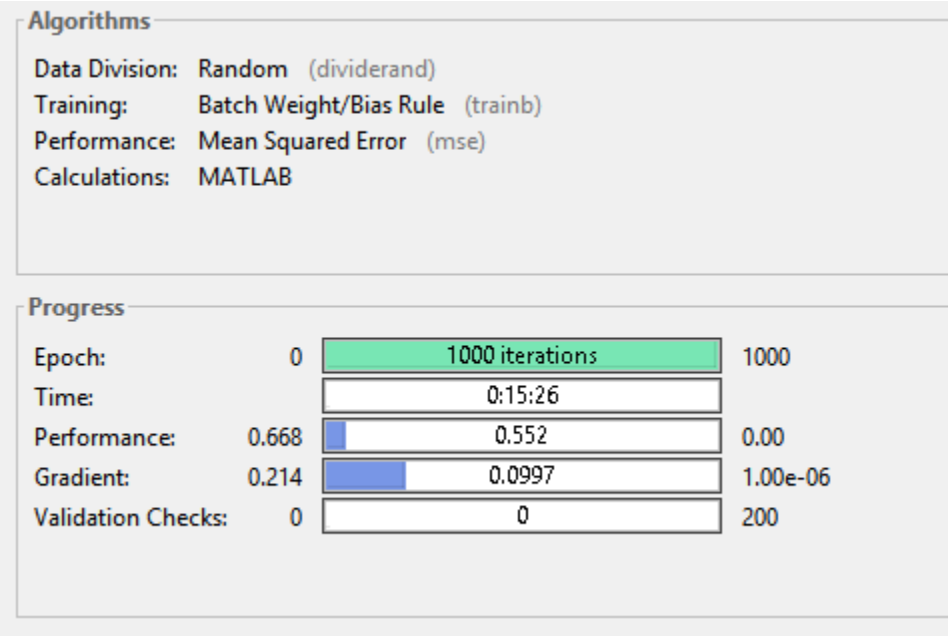
Data Division:	Random	(dividerand)
Training:	Gradient Descent with Momentum & Adaptive LR	(traingdx)
Performance:	Cross-Entropy	(crossentropy)
Calculations:	MEX	

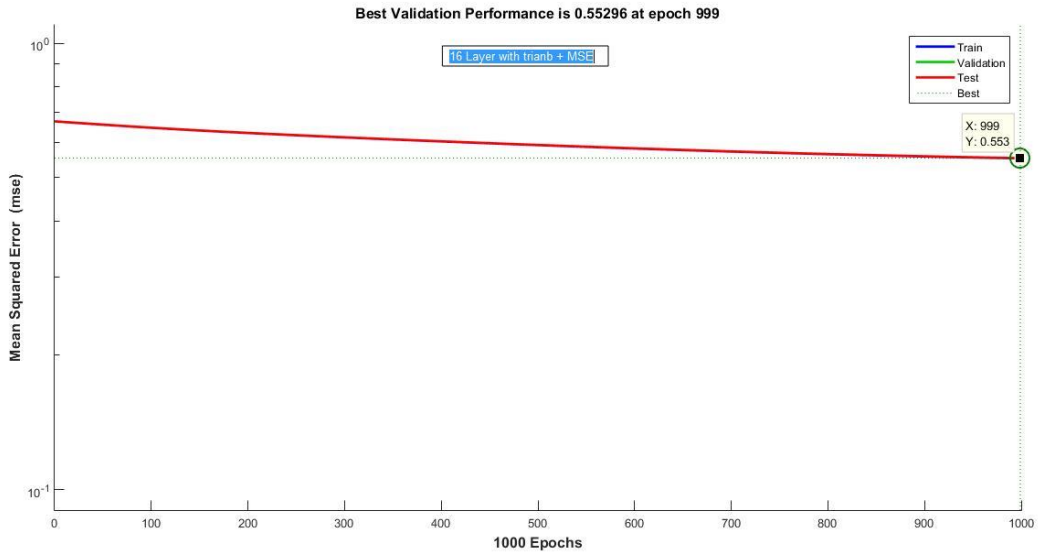
Progress

Epoch:	0	276 iterations	1000
Time:		0:02:46	
Performance:	0.0421	1.08e-05	0.00
Gradient:	0.00925	9.82e-06	1.00e-05
Validation Checks:	0	0	200



و در آخر شبکه ای با ۱۶ لایه وروش آموزش دسته ای و تابع خطای MSE:





trainFcn	'trainb'
trainParam	*1x1 struct*
performFcn	'mse'
performParam	*1x1 struct*
derivFcn	'defaultderiv'
divideFcn	'dividerand'
divideMode	'sample'
divideParam	*1x1 struct*
trainInd	*1x50006 double*
valInd	*1x9997 double*
testInd	*1x9997 double*
stop	'Maximum epoch rea...
num_epochs	1000
trainMask	*1x1 cell*
valMask	*1x1 cell*
testMask	*1x1 cell*
best_epoch	999
goal	0
states	*1x6 cell*
epoch	*1x1001 double*
time	*1x1001 double*
perf	*1x1001 double*
vperf	*1x1001 double*
tperf	*1x1001 double*
val_fail	*1x1001 double*
best_perf	0.5523
best_vperf	0.5530
best_tperf	0.5530

در toolbox ما با مشکل کمتری در افزایش لایه ها درگیر هستیم(منظور زمان آموزش هست و دقت)بدلیل اینکه این توابع با بهترین کارایی نوشته شده اند و مواردی را در نظر گرفته و بکار می برند که ما از آنها استفاده نمیکنیم، مثلا بجای استفاده از گردایان عددی از گردایان آماری استفاده می کنند که سرعت آموزش شبکه را خیلی افزایش می دهد و ...

و در آخر با پارامتر های زیر کد پیاده سازی شده رو تست کردم:

تست آخر

تعدا نمونه های آموزشی:۵۰۰۰۰

نرخ یادگیری:۰,۹

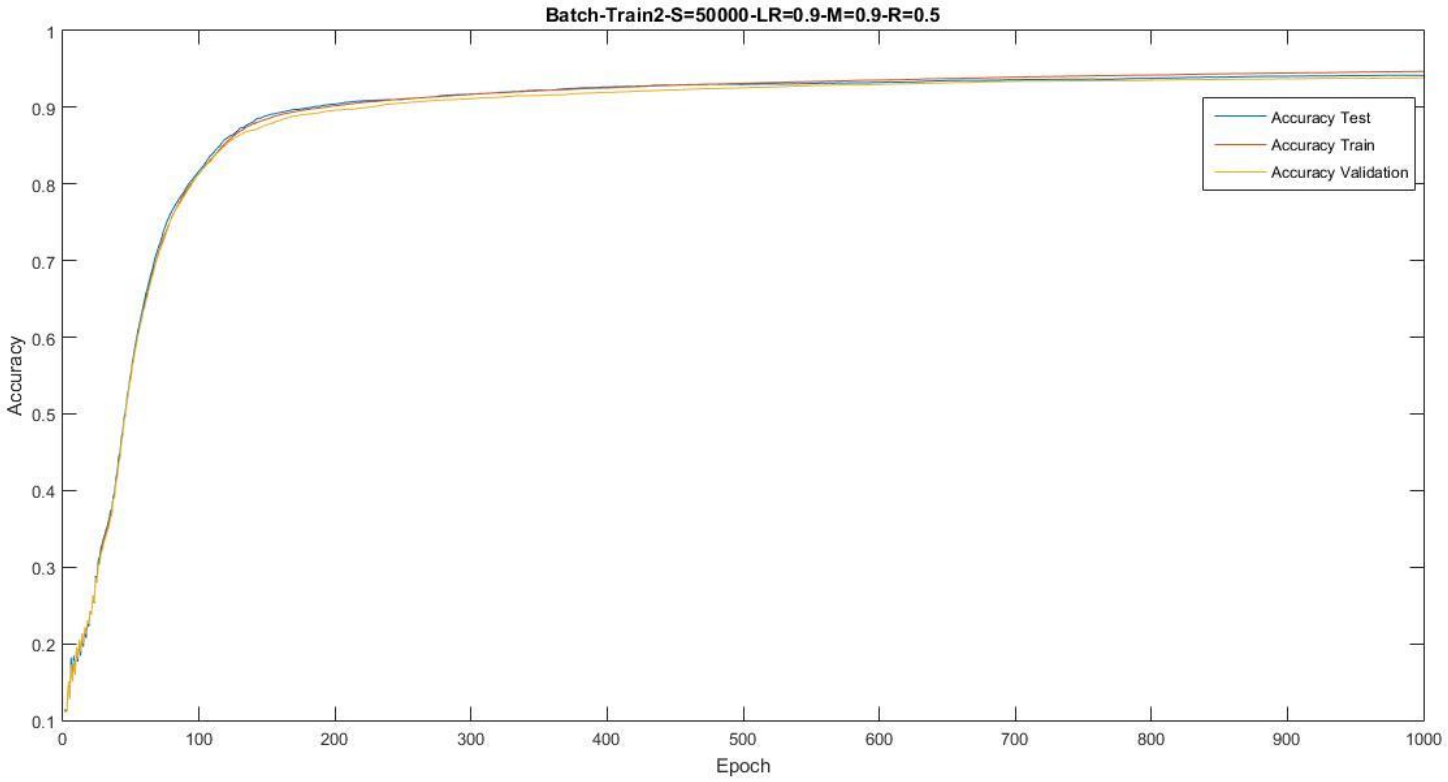
```
Regularization: 0.5
Momentum: 0.9
```

تعدا لایه ها:۳(۷۸۴ ورودی ،۲۵ نورون در لایه پنهان و ۱۰ نورن خروجی)

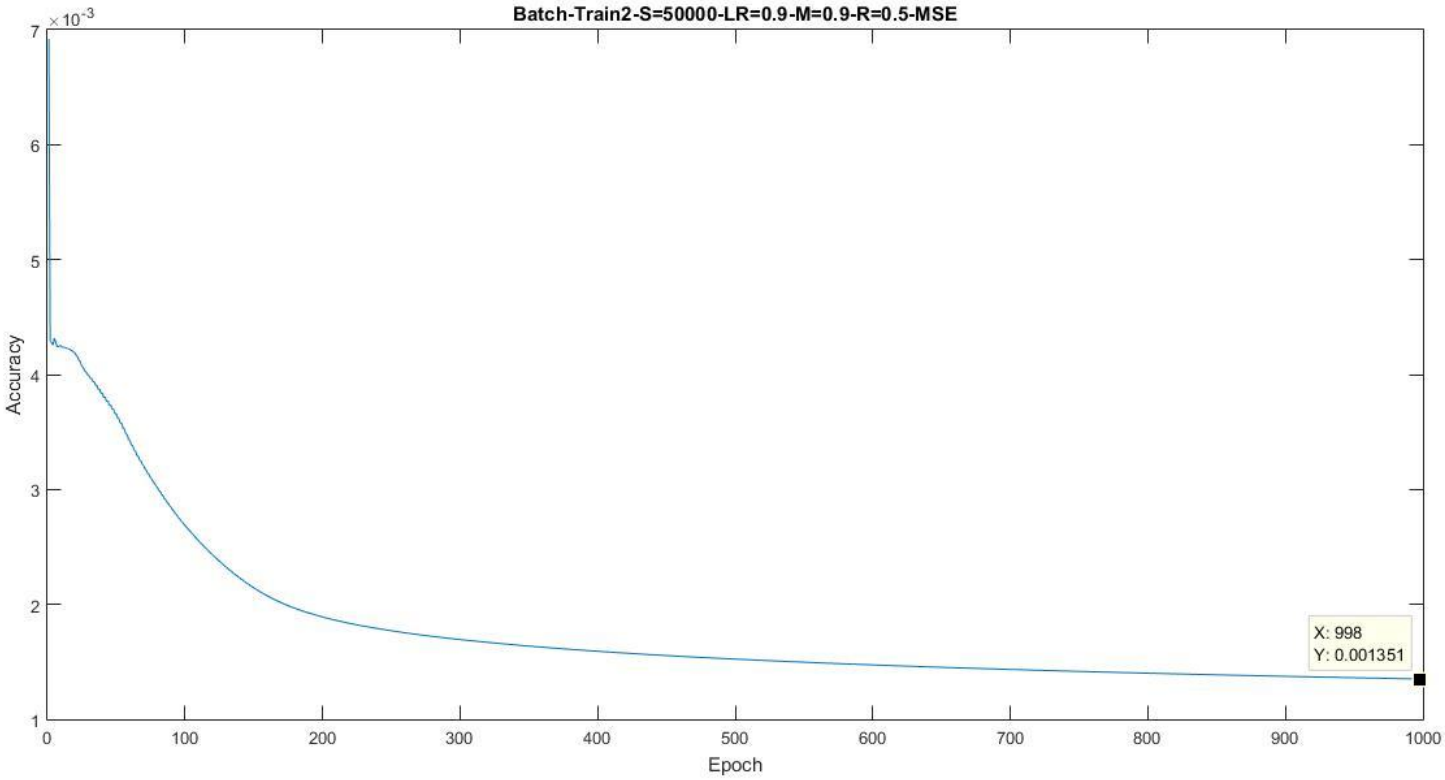
تعداد epoch:۱۰۰۰

زمان یادگیری (زمان اجرای کد در متلب):۴۵ ساعت

نمودار دقت بر حسب epoch:



نمودار MSE بر حسب epoch:



به لطف خداوند متعال نتیجه جالب توجه بدست آمده در این آزمایش رشد همزمان و تقریبا یکسان دقت در داده های آموزشی و تست و ازریابی هست. و در ۱۰۰۰ epoch مقدار دقت به ۹۴ درصد برای داده های آموزشی و تست و ارزیابی میرسد.