Introduction

This matlab script will focus on image classification for CIFAR-10. Four different networks of different complexity will be investigated. You can preload the trained network model into matlab or train them with 'Network X' section below. Run section 'Plot training progress' to view the main experimental results discussed in the report. To get classification results of the final model run section 'Clf errors for training, validation, and test set'

Contents

Data loading and Pre-processing

- The LOADCIFAR function loads the data to a specific format.
- Each columns is a 3072 element long vector representing a 32×\times×32 pixels RGB image

```
%Load the dataset
[xTrain, tTrain, xValid, tValid, xTest, tTest] = LoadCIFAR(1);


%Centralise the data, use the mean of the training data
xValid = xValid - repmat(mean(xTrain,2),1,size(xValid,2));
xTest = xTest - repmat(mean(xTrain,2),1,size(xTest,2));
xTrain = xTrain - mean(xTrain,2);

batch_size = 100;
eta = 0.1;

error('Preload trained models into matlab or train them with sections below. Note training
 takes time. e.g net 4 takes 2 hours')
```

```
ans =

        32          32           3       50000



Error using Fully_connected_networks_script (line 25)
Preload trained models into matlab or train them with sections below. Note training takes
time. e.g net 4 takes 2 hours
```

Network 1

- No hidden layers

```
Weights_Net1 = {normrnd(0,1/(sqrt(size(xTrain,1))),[10,size(xTrain,1)]);};
Neurons_Net1 = {zeros(10,1);};
Thresholds_Net1 = {zeros(10,1);} ;


weights_epochs_net1 = cell(20,1);
thresholds_epochs_net1 = cell(20,1);

for epoch=1:20  % Loop through 20 epochs
    disp(epoch)

    %Save the weights in each epoch
    weights_epochs_net1{epoch} = Weights_Net1;
    thresholds_epochs_net1{epoch} = Thresholds_Net1;

    %Shuffle the data at start of epoch
    shuffle_index = randperm(size(xTrain,2));
    xTrain = xTrain(:,shuffle_index);
    tTrain = tTrain(:,shuffle_index);

    for i=1:batch_size:size(xTrain,2)  %For each epoch
        Min_Batch_indexes = i:i+batch_size-1;
        [Weights_Net1,Thresholds_Net1]  = MinBatchGradientDescent(Weights_Net1, Neurons_Ne
t1, Thresholds_Net1, xTrain, tTrain, Min_Batch_indexes,eta);
    end


end

disp('Done training net 1')
```

Network 2

- One hidden layer with 10 neurons

```
Weights_Net2 = {normrnd(0,1/(sqrt(size(xTrain,1))),[10,size(xTrain,1)]);
                normrnd(0,1/sqrt(10),[10,10]);};

Neurons_Net2 = {zeros(10,1);
                zeros(10,1);};

Thresholds_Net2 = {zeros(10,1);
                   zeros(10,1);};



%Training
weights_epochs_net2 = cell(20,1);
thresholds_epochs_net2 = cell(20,1);
for epoch=1:20  % Loop through 20 epochs
    disp(epoch)


    %Save the weights in each epoch
    weights_epochs_net2{epoch} = Weights_Net2;
    thresholds_epochs_net2{epoch} = Thresholds_Net2;
```

```
    %Shuffle the data at start of epoch
    shuffle_index = randperm(size(xTrain,2));
    xTrain = xTrain(:,shuffle_index);
    tTrain = tTrain(:,shuffle_index);
    for i=1:batch_size:size(xTrain,2)  %For each epoch
        Min_Batch_indexes = i:i+batch_size-1;
        [Weights_Net2,Thresholds_Net2]  = MinBatchGradientDescent(Weights_Net2, Neurons_Ne
t2, Thresholds_Net2, xTrain, tTrain, Min_Batch_indexes,eta);
    end

end


disp('Done training net 2')
```

Network3

- One hidden layer with 50 neurons

```
Weights_Net3 = {normrnd(0,1/(sqrt(size(xTrain,1))),[50, size(xTrain,1)]);
                normrnd(0,1/sqrt(50),[10,50]);};

Neurons_Net3 = {zeros(50,1);
                zeros(10,1);};

Thresholds_Net3 = {zeros(50,1);
                   zeros(10,1);};


%Training
weights_epochs_net3 = cell(20,1);
thresholds_epochs_net3 = cell(20,1);
for epoch=1:20  % Loop through 20 epochs
    disp(epoch)

    %Save the weights in each epoch
    weights_epochs_net3{epoch} = Weights_Net3;
    thresholds_epochs_net3{epoch} = Thresholds_Net3;

    %Shuffle the data at start of epoch
    shuffle_index = randperm(size(xTrain,2));
    xTrain = xTrain(:,shuffle_index);
    tTrain = tTrain(:,shuffle_index);
    for i=1:batch_size:size(xTrain,2)  %For each epoch
        Min_Batch_indexes = i:i+batch_size-1;
        [Weights_Net3,Thresholds_Net3]  = MinBatchGradientDescent(Weights_Net3, Neurons_Ne
t3, Thresholds_Net3, xTrain, tTrain, Min_Batch_indexes,eta);
    end
end

disp('Done training net 3')
```

Network 4

- Two hidden layers with 50 neurons each

```
Weights_Net4 = {normrnd(0,1/(sqrt(size(xTrain,1))),[50,size(xTrain,1)]);
```

```
                normrnd(0,1/(sqrt(50)),[50,50]);
                normrnd(0,1/(sqrt(50)),[10,50]);};

Neurons_Net4 = {zeros(50,1);
                zeros(50,1);
                zeros(10,1);};

Thresholds_Net4 = {zeros(50,1);
                   zeros(50,1);
                   zeros(10,1);};

weights_epochs_net4 = cell(20,1);
thresholds_epochs_net4 = cell(20,1);


for epoch=1:20  % Loop through 20 epochs
    disp(epoch)

    %Save the weights in each epoch
    weights_epochs_net4{epoch} = Weights_Net4;
    thresholds_epochs_net4{epoch} = Thresholds_Net4;

    %Shuffle the data at start of epoch
    shuffle_index = randperm(size(xTrain,2));
    xTrain = xTrain(:,shuffle_index);
    tTrain = tTrain(:,shuffle_index);

    for i=1:batch_size:size(xTrain,2)  %For each epoch
        Min_Batch_indexes = i:i+batch_size-1;
        [Weights_Net4,Thresholds_Net4]  = MinBatchGradientDescent(Weights_Net4, Neurons_Ne
t4, Thresholds_Net4, xTrain, tTrain, Min_Batch_indexes,eta);
    end
end

disp('Done training net 4')
```

Plot the training progress

```
[net1_clfError_train_epochs ,net1_clfError_val_epochs] = plotClfError(weights_epochs_net1,
 thresholds_epochs_net1, Neurons_Net1, xTrain, tTrain, xValid, tValid, 'Net1', 'r');
hold on
[net2_clfError_train_epochs ,net2_clfError_val_epochs] = plotClfError(weights_epochs_net2,
 thresholds_epochs_net2, Neurons_Net2, xTrain, tTrain, xValid, tValid, 'Net2', 'g');
hold on
[net3_clfError_train_epochs ,net3_clfError_val_epochs] =plotClfError(weights_epochs_net3,
thresholds_epochs_net3, Neurons_Net3, xTrain, tTrain, xValid, tValid, 'Net3', 'm');
hold on
[net4_clfError_train_epochs ,net4_clfError_val_epochs] = plotClfError(weights_epochs_net4,
 thresholds_epochs_net4, Neurons_Net4, xTrain, tTrain, xValid, tValid, 'Net4', 'k');
legend('Train error - Net1', 'Validation error- Net1', 'Train error - Net2', 'Validation e
rror - Net2', 'Train error - Net3', ...
       'Validation error - Net3', 'Training error - Net4', 'Validation error - Net4' )

title('Training')
xlabel('Epochs')
ylabel('classification error')
```

Clf errors for training, validation, and test set

```
[val ,idx] = min(cell2mat(net1_clfError_val_epochs));
net1_clfError_at_epoch = idx;
[net1_clfError_train] = ClfErr(weights_epochs_net1{idx}, thresholds_epochs_net1{idx}, Neur
ons_Net1, xTrain, tTrain);
[net1_clfError_val] = ClfErr(weights_epochs_net1{idx}, thresholds_epochs_net1{idx}, Neuron
s_Net1, xValid, tValid);
[net1_clfError_test] = ClfErr(weights_epochs_net1{idx}, thresholds_epochs_net1{idx}, Neuro
ns_Net1, xTest, tTest);
disp('Computed for net 1')


[val, idx] = min(cell2mat(net2_clfError_val_epochs));
net2_clfError_at_epoch = idx;
[net2_clfError_train] = ClfErr(weights_epochs_net2{idx}, thresholds_epochs_net2{idx}, Neur
ons_Net2, xTrain, tTrain);
[net2_clfError_val] = ClfErr(weights_epochs_net2{idx}, thresholds_epochs_net2{idx}, Neuron
s_Net2, xValid, tValid);
[net2_clfError_test] = ClfErr(weights_epochs_net2{idx}, thresholds_epochs_net2{idx}, Neuro
ns_Net2, xTest, tTest);
disp('Computed for net 2')



[val, idx] = min(cell2mat(net3_clfError_val_epochs));
net3_clfError_at_epoch = idx;
[net3_clfError_train] = ClfErr(weights_epochs_net3{idx}, thresholds_epochs_net3{idx}, Neur
ons_Net3, xTrain, tTrain);
[net3_clfError_val] = ClfErr(weights_epochs_net3{idx}, thresholds_epochs_net3{idx}, Neuron
s_Net3, xValid, tValid);
[net3_clfError_test] = ClfErr(weights_epochs_net3{idx}, thresholds_epochs_net3{idx}, Neuro
ns_Net3, xTest, tTest);
disp('Computed for net 3')




[val, idx] = min(cell2mat(net4_clfError_val_epochs));
net4_clfError_at_epoch = idx;
[net4_clfError_train] = ClfErr(weights_epochs_net4{idx}, thresholds_epochs_net4{idx}, Neur
ons_Net4, xTrain, tTrain);
[net4_clfError_val] = ClfErr(weights_epochs_net4{idx}, thresholds_epochs_net4{idx}, Neuron
s_Net4, xValid, tValid);
[net4_clfError_test] = ClfErr(weights_epochs_net4{idx}, thresholds_epochs_net4{idx}, Neuro
ns_Net4, xTest, tTest);
disp('Computed for net 4')
```

Helper Functions

```
function [ClfError_train_epochs, ClfError_val_epochs] = plotClfError(Weights_epoch, Thresh
hold_epoch, Neurons, trainData, trainLabel ,valData, valLabel, NetworkType, color);
    L = size(Neurons,1);

    %Classification error - Training
    ClfError_train_epochs = cell(20,1);
    for epoch=1:1:20
        predictions_train = cell(size(trainData,2),1);
        for i=1:1:size(trainData,2)
            %Propagate forward validation set
            %First layer
```

```
            Neurons{1}=sigmf(Weights_epoch{epoch}{1}*trainData(:,i) - Threshhold_epoch{epo
ch}{1}, [1,0]);
            for l=2:1:L
                Neurons{l} = sigmf(Weights_epoch{epoch}{l}*Neurons{l-1} - Threshhold_epoch
{epoch}{l}, [1,0]);
            end

            [val, idx] = max(Neurons{L});
            output = zeros(10,1);
            output(idx) = 1;
            predictions_train{i} = output;
        end
        ClfError = 0;
        for i=1:1:size(trainData,2)
            ClfError = ClfError + sum(abs(trainLabel(:,i) - predictions_train{i}));
        end
        ClfError_train_epochs{epoch} = ClfError/(2*size(trainData,2));
    end
    disp(strcat('Done computing clfError for Training-', NetworkType))


    %Classification error - Validation
    ClfError_val_epochs = cell(20,1);
    for epoch=1:1:20
        predictions_val = cell(size(valData,2),1);
        for i=1:1:size(valData,2)
            %Propagate forward validation set
            %First layer
            Neurons{1}=sigmf(Weights_epoch{epoch}{1}*valData(:,i) - Threshhold_epoch{epoch
}{1}, [1,0]);
            for l=2:1:L
                Neurons{l} = sigmf(Weights_epoch{epoch}{l}*Neurons{l-1} - Threshhold_epoch
{epoch}{l}, [1,0]);
            end
            [val, idx] = max(Neurons{L});
            output = zeros(10,1);
            output(idx) = 1;
            predictions_val{i} = output;
        end

        ClfError = 0;
        for i=1:1:size(valData,2)
            ClfError = ClfError + sum(abs(valLabel(:,i) - predictions_val{i}));
        end
        ClfError_val_epochs{epoch} = ClfError/(2*size(valData,2));
    end

    disp(strcat('Done computing clfError for Validation-', NetworkType))

    %Plotting
    x = 1:1:20;
    semilogy(x,cell2mat(ClfError_train_epochs),color)
    %plot(x,cell2mat(ClfError_train_epochs),color)
    hold on
    %plot(x,cell2mat(ClfError_val_epochs),strcat('--.',color))
    semilogy(x,cell2mat(ClfError_val_epochs),strcat('--.',color))
end
```

```
function [ClfError] = ClfErr(Weights, Threshhold, Neurons, data, labels ,valData);
    L = size(Neurons,1);
    predictions_train = cell(size(data,2),1);
    for i=1:1:size(data,2)
        %Propagate forward validation set
        %First layer
        Neurons{1}=sigmf(Weights{1}*data(:,i) - Threshhold{1}, [1,0]);
        for l=2:1:L
            Neurons{l} = sigmf(Weights{l}*Neurons{l-1} - Threshhold{l}, [1,0]);
        end

        [val, idx] = max(Neurons{L});
        output = zeros(10,1);
        output(idx) = 1;
        predictions_train{i} = output;
    end
    ClfError = 0;
    for i=1:1:size(data,2)
        ClfError = ClfError + sum(abs(labels(:,i) - predictions_train{i}));
    end
    ClfError = ClfError/(2*size(data,2));
end
```

Stochastic gradient descent with min-batch

```
function [Weights, Threshholds] = MinBatchGradientDescent(Weights, Neurons, Threshholds, d
ata,labels, MinBatchIndex, eta);
    %Initialize variables
    L =  size(Neurons,1); %Let L be the index of last layer
    Errors = cell(L,1);
    Delta_weights = cell(L,1);
    Delta_Thresholds = cell(L,1);

    %Initialize the errors
    for l=1:L
        Errors{l} = zeros(size(Neurons{l},1),1);
    end

    %Initialize the delta_thresholds
    for l=1:size(Neurons,1)
        Delta_Thresholds{l} = zeros(size(Neurons{l},1),1);
    end

    %Initialize the delta weights
    for l=1:size(Weights,1)
        Delta_weights{l} = zeros(size(Weights{l},1), size(Weights{l},2));
    end



    %Start min-batch
    for inputsIndex = MinBatchIndex
        inputs = data(:,inputsIndex);
```

```
        %Propagate forward
        Neurons{1} = sigmf(Weights{1}*inputs-Threshholds{1}, [1,0]);
        for l=2:L
            Neurons{l} = sigmf(Weights{l}*Neurons{l-1} - Threshholds{l}, [1,0]);
        end

        %Compute error at output layer
        if L == 1
            Locals_1 = Weights{1}*inputs-Threshholds{1};
            Errors{1} = (sigmf(Locals_1,[1,0])).*(1-(sigmf(Locals_1,[1,0]))).*(labels(:,in
putsIndex)-Neurons{1});
        else
            Locals_L = Weights{L}*Neurons{L-1}-Threshholds{L};
            Errors{L} = (sigmf(Locals_L,[1,0])).*(1-(sigmf(Locals_L, [1,0]))).*(labels(:,i
nputsIndex)-Neurons{L});
        end


        %Backpropgate the error (only when more than 1 layers)
        for l=L:-1:2
            if l==2
                Locals_layer_1 = Weights{l-1}*inputs-Threshholds{l-1};
                Errors{l-1} = Weights{l}'*Errors{l}.*(sigmf(Locals_layer_1 ,[1,0])).*(1-(s
igmf(Locals_layer_1, [1,0])));
            else
                Locals_l_Minus1 = Weights{l-1}*Neurons{l-2}-Threshholds{l-1};
                Errors{l-1} = Weights{l}'*Errors{l}.*(sigmf(Locals_l_Minus1 ,[1,0])).*(1-(
sigmf(Locals_l_Minus1, [1,0])));
            end
        end


        %Use error in each layer to compute delta weights/threshhold
        %First layer
        Delta_weights{1} = Delta_weights{1} + eta*Errors{1}*inputs';
        Delta_Thresholds{1} = Delta_Thresholds{1} - eta*Errors{1};

        %delta weights/thresh for all other layers
        for l=2:1:L
            Delta_weights{l} = Delta_weights{l} + eta*Errors{l}*Neurons{l-1}';
            Delta_Thresholds{l} = Delta_Thresholds{l} - eta*Errors{l};
        end
    end

    %Update weights and thresholds with delta weights/thresholds
    for l=1:1:L
        Weights{l} = Weights{l} + Delta_weights{l};
        Threshholds{l} = Threshholds{l} + Delta_Thresholds{l};
    end
end
```