Introduction

This matlab script will focus on image classification for CIFAR-10. A network with 4 hidden layers with 20 neurons each will be constructed. The following code will plot the learning rate for each layer and check the effect known as 'vanishing gradient.' Preload the trained model into matlab and run section 'Plot learning rate' and 'Plot loss.' Or train the network from scratch by running section 'Construct network and training'

Contents

Dat 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(2);


%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 = 0.01;
nrEpochs = 100;

error('Preload the models into matlab or run the next section to construct the model from scratch')
```

```
ans =

          32           32            3        50000



Error using Vanishing_gradient_script (line 26)
Preload the models into matlab or run the next section to construct the model from scratch
```

Construct network and training

```
Weights = {normrnd(0,1/(sqrt(size(xTrain,1))),[20,size(xTrain,1)]);
                normrnd(0,1/(sqrt(20)),[20,20]);
                normrnd(0,1/(sqrt(20)),[20,20]);
                normrnd(0,1/(sqrt(20)),[20,20]);
```

```
                normrnd(0,1/(sqrt(20)),[10,20]);};

Neurons = {zeros(20,1);
                zeros(20,1);
                zeros(20,1);
                zeros(20,1);
                zeros(10,1);};

Thresholds = {zeros(20,1);
                   zeros(20,1);
                   zeros(20,1);
                   zeros(20,1);
                   zeros(10,1);};

weights_epochs = cell(nrEpochs,1);
thresholds_epochs = cell(nrEpochs,1);
deltaweights_epochs = cell(nrEpochs,1);
deltaThresh_epochs = cell(nrEpochs,1);

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

    %Save the weights in each epoch
    weights_epochs{epoch} = Weights;
    thresholds_epochs{epoch} = Thresholds;

    %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,Thresholds, Delta_weights, Delta_thresh]  = MinBatchGradientDescent(Weigh
ts, Neurons, Thresholds, xTrain, tTrain, Min_Batch_indexes,eta);
    end
    deltaweights_epochs{epoch} = Delta_weights;
    deltaThresh_epochs{epoch} = Delta_thresh;
end
```

Learning rate for each layer

```
 for layer=1:size(deltaThresh_epochs{1})
    %deltaweights_epochss{epoch}
    learning_speeds = cell(nrEpochs,1);
    for epoch=1:1:nrEpochs
        learning_speeds{epoch} = norm(deltaThresh_epochs{epoch}{layer});
    end
    x = 1:1:nrEpochs;
    %disp(size(x))
    %disp(size(learning_speeds{layer}))
    semilogy(x,cell2mat(learning_speeds))
    hold on
end



title('Vanishing gradient')
xlabel('Epochs')
ylabel('Learning speed')
```

```
legend('Layer 1', 'Layer 2', 'Layer 3', 'Layer 4', 'Layer 5')
hold off


%Plot energy function
L = size(Neurons,1);
```

Plotting the loss

```
loss_train_epochs = cell(nrEpochs,1);
for epoch=1:1:nrEpochs
    disp(epoch)
    predictions_train = cell(size(xTrain,2),1);
    for i=1:1:size(xTrain,2)
        %Propagate forward validation set
        %First layer
        Neurons{1}=sigmf(weights_epochs{epoch}{1}*xTrain(:,i) - thresholds_epochs{epoch}{1
}, [1,0]);
        for l=2:1:L
            Neurons{l} = sigmf(weights_epochs{epoch}{l}*Neurons{l-1} - thresholds_epochs{e
poch}{l}, [1,0]);
        end

        %[val, idx] = max(Neurons{L});
        %output = zeros(10,1);
        %output(idx) = 1;
        predictions_train{i} = Neurons{L};
    end
    loss = 0;
    for i=1:1:size(xTrain,2)
        loss = loss + sum((tTrain(:,i) - predictions_train{i}).^2);
    end
    loss_train_epochs{epoch} = loss/2;
end

x = 1:1:nrEpochs;
semilogy(x,cell2mat(loss_train_epochs))


title('Training progress')
xlabel('Epochs')
ylabel('Loss')
%plot(x,cell2mat(ClfError_train_epochs),color)
```

Functions

```
function plotClfError(weights_epochs, Threshhold_epoch, Neurons, xTrain, tTrain ,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(xTrain,2),1);
        for i=1:1:size(xTrain,2)
            %Propagate forward validation set
            %First layer
```

```
            Neurons{1}=sigmf(weights_epochs{epoch}{1}*xTrain(:,i) - Threshhold_epoch{epoch
}{1}, [1,0]);
            for l=2:1:L
                Neurons{l} = sigmf(weights_epochs{epoch}{l}*Neurons{l-1} - Threshhold_epoc
h{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(xTrain,2)
            ClfError = ClfError + sum(abs(tTrain(:,i) - predictions_train{i}));
        end
        ClfError_train_epochs{epoch} = ClfError/(2*size(xTrain,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_epochs{epoch}{1}*valData(:,i) - Threshhold_epoch{epoc
h}{1}, [1,0]);
            for l=2:1:L
                Neurons{l} = sigmf(weights_epochs{epoch}{l}*Neurons{l-1} - Threshhold_epoc
h{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
```

Stochastic gradient descent with min-batch

```
function [Weights, Threshholds, Delta_weights, Delta_Thresholds] = MinBatchGradientDescent
(Weights, Neurons, Threshholds, data,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
```

Published with MATLAB® R2017b