Homework 4

Due Date: May 23, 2018

Notes

- I worked on video parts. So, I won't hand-in *main_hw4_audio.m* file.
- Please refer to attached *main_hw4_video.m*, *mlp_feedforward.m*, *mlp_backprop.m, weight_update.m, softmax_regression.m, feature_extraction* files.

IMPORTANT: 본 과제에 필요한 mat 파일들을 저장하는 과정에서 해동라운지로부터 개인에게 할당된 용량을 초과하는 일이 많이 있었습니다. (processed 폴더의 용량만 500MB 가 넘어가는 상황이었습니다.) 이에, 본 과제에서 요구되는 일정 부분들을 수행하지 못한 점을 양해 부탁드립니다. 그에 대해 제가 취한 조치들을 아래에 적어 두었습니다.

- I also made changes on *load_data.m*, *preprocess_data_label.m*, to reduce memory by not processing on audio data.
- I could not make *model_1000_nll_0.73_acc_0.20.mat* files and etc, because My computer cannot hold such large memory. -> For such reason, I could not fit 'best model' for my test case, but used 'last model' (on 50000th epoch) to test my program.

I. Code Completion

- STEP 0.

I modified *main_hw4_video.m* to change hyper-variables.

```
%% STEP 0. Some Hyperparameters
clear all; clc; close all; rng(0);
student_id = 20160042;
your_name = 'Inyong Koo';
fprintf('[Homework 4-2 Video] name : %s, student id : %d \n', your_name, student_id )

opt.pooling_type = 'mean'; % Pooling type, should be either 'max' or 'mean'
opt.train_model = 'sae'; % Train model, should be either 'sae' or 'pca'
opt.hidden_size = 512; % Dimension of hidden layer of 2-layers MLP
opt.batch_size = 50; % Number of data in mini-batch
opt.init_std = 0.01; % Standard deviation of gaussian distribution where initial weight values are sampled
opt.learning_rate = 0.02; % Learning rate to be used in stochatic gradient decsent
opt.total_iteration = 50000; % Total number of mini-batch updates
opt.check_valid_freq = 1000; % How often you will check validation error
opt.print_train_freq = 100; % How often train error will be printed

nll_save = zeros(opt.total_iteration, 1);
valid_nll_save = zeros(opt.total_iteration/opt.check_valid_freq, 2);
```

I altered learning_rate by 0.04, 0.06 and so on, and found out that larger learning rate causes overfitting.

I could have changed the hidden_size and other variables, but I decided to **fix learning rate by 0.02** to compare different models. I added *opt.train_model* to modify train models.

- STEP 1.

I modified *main_hw4_video.m* to change number of files we're using.

```
%% STEP 1. Pre-processing raw data --- [DO NOT MODIFY]
data_dir = '../data/';
processed_dir = '../data/processed/';
model_dir = sprintf('trained_model_%s_%s/', 'video', opt.pooling_type);

if ~exist(processed_dir, 'dir')
    mkdir(processed_dir);
end
if ~exist(model_dir, 'dir')
    mkdir(model_dir);
end
if ~(exist(fullfile(processed_dir, 'video_processed.mat'), 'file') && ...
        exist(fullfile(processed_dir, 'data_label.mat'), 'file'))
    preprocess_data_label(data_dir, processed_dir);
end

load(fullfile(processed_dir, 'video_processed.mat'), 'video_processed');
train_video = video_processed{1}(1:34,1);   % Modify train set number here
valid_video = video_processed{2}(1:5,1);    % Modify valid set number here
test_video = video_processed{3}(1:6,1);     % Modify test set number here
clear video_processed;

load(fullfile(processed_dir, 'data_label.mat'), 'label');
train_label = label{1}(1:34,:);             % Modify train set number here
valid_label = label{2}(1:5,:);              % Modify valid set number here
test_label = label{3}(1:6,:);               % Modify test set number here
clear label;

fprintf('Video data and labels loaded\n');
```

Given data has 850 train set, 50 validation set, and 150 test set, but we could not use all those files since we are limited on time and space. *EE476_Homework4_guideline.pdf* indicates that we may use (17, 1, 3) sets, but I chose to use (34, 5, 6) sets.

- STEP 2.

On Step 2, I extracted features from given sets by altering codes of HW 3.

```
%% STEP 2. Extract feature from pretrained CNNs --- [COMPLETE CODE FOR THIS STEP]
% Replaces the content of cell arrays
% 1) train_video 2) valid_video 3) test_video
% using CNNs you trained in Homework 3
% Structure of cell arrays and above should not be changed
% Data in each cell should have
% (feature map height) X (feature map width) X (feature map number) X (number of time steps)
if strcmp(opt.train_model,'sae')
    load('20160042_task3_sae.mat');
else
    load('20160042_task3_pca.mat');
end

train_video = feature_extraction(train_video, params, opt.train_model);
valid_video = feature_extraction(valid_video, params, opt.train_model);
test_video = feature_extraction(test_video, params, opt.train_model);
fprintf('Feature extraction done\n');
```

Please refer to *feature_extraction.m* file. Note that I used pretrained CNN data, by loading the results of HW3.

- STEP 3, 4-0.

I did not modify STEP 3 and 4-0. Please refer to attached *global_pooling.m* (Given)

- STEP 4.

I implemented *mlp_feedforward.m* as following.

```
function [mlp, output] = mlp_feedforward(mlp, input, type)
    % input : input data to current mlp layer
    % output : output feature of mlp layer
    % mlp.weight : concatenation of weight and bias

    if ~(strcmp(type, 'fc') || strcmp(type, 'relu'))
        error('Layer type should be either fc (fully connected) or relu (ReLU)');
    end

    %% Complete codes below
    if strcmp(type, 'fc')
        output = input * mlp.weight + repmat(mlp.bias, size(input,1), 1);
        mlp.input = input;
    else
        output = max(0, input);   %reLU
        mlp.active = output;
    end
end
```

I followed simple (y = W*x + b) model for fully-connected network.

- STEP 5.

I implemented *softmax_regression.m* as following.

```
function [nll, softmax, err] = softmax_regression(input, label)
    % input : input data to current mlp layer with size (batch_size) X (feature_dim)
    % label : label of input data (batch_size) X 10
    % nll : cross-entropy error
    % err : error to be back-propagated to 2-layer mlp

    %% Complete codes below
    % softmax function output
    softmax = (exp(input') ./ sum(exp(input)'))';

    % Calculate cross entropy error using output from softmax
    yhat = 1./(1+exp(-softmax));
    err = - (label .* log(yhat) + (1 - label) .* log(1-yhat));
    nll = sum(sum(err))/10;
end
```

Cross-entropy loss function is as following.

$$J(\mathrm{w}) = -\frac{1}{N}\sum_{n=1}^{N}[y_n \log \hat{y}_n + (1-y_n)\log(1-\hat{y}_n)] \quad \text{where} \quad \hat{y} = 1/(1+\mathrm{e}^{-x}).$$

- STEP 6-1.

I implemented *mlp_backprop.m* as following.

```
function [mlp, err_out] = mlp_backprop(mlp, err_in, type)
    % input : input data to current mlp layer
    % output : output feature of mlp layer

    if ~(strcmp(type, 'fc') || strcmp(type, 'relu'))
        error('Layer type should be either fc (fully connected) or relu (ReLU)');
    end

    %% Complete codes below
    if strcmp(type, 'fc')
        err_out = - (mlp.weight * err_in');
        mlp.error = err_in;
    else
        err_out = abs((mlp.active .* err_in'));    % derivative reLU
    end
end
```

- STEP 6-2.

I implemented *weight_update.m* as following.

```
function mlp = weight_update(mlp, learning_rate, batch_size)
    % Complete codes below
    weight_gradient = mlp.input' * mlp.error;
    bias_gradient = sum(mlp.error, 1);
    mlp.weight = mlp.weight - learning_rate * weight_gradient / batch_size.
    mlp.bias = mlp.bias - learning_rate * bias_gradient / batch_size;
end
```

II. TASKS

Under condition Learning_rate = 0.02, (#Train = 34 / #Valid = 5/ #Test = 6),

I altered global pooling method (max or mean), and CNN model (SAE or PCA), and observed the training result.

After learning is finished, I used following code lines to visualize the result.

```
figure();
plot(nll_save);
title('Train data - Cross entropy error (nll)');

figure();
plot(valid_nll_save(:,1));
title('Validation data - Cross entropy error (nll)');

figure();
plot(valid_nll_save(:,2));
title('Validation data - accuracy');
```

Also, I uncommented last lines of *main_hw4_video.m* to get the Test result.

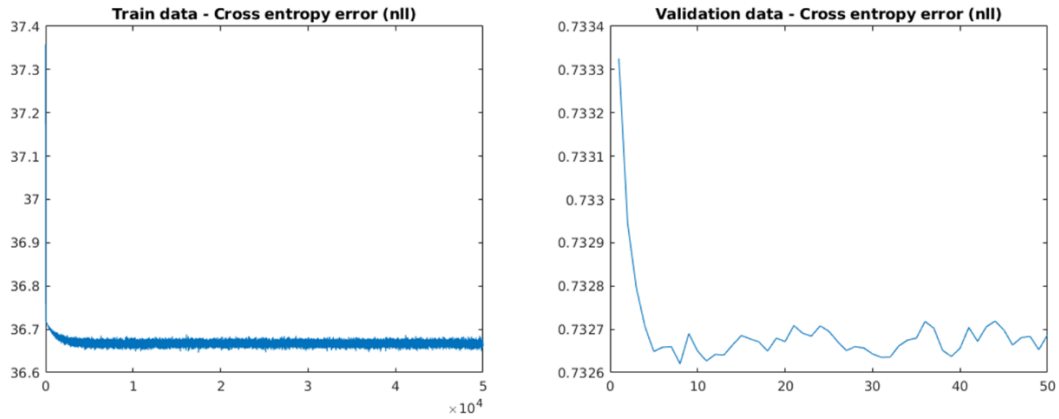
```
%% EVALUATION ON TEST DATA WITH CHOSEN MODEL
% Use test code below to evaluate your best model with test data

[test_nll, test_acc] = evaluate(test_video, test_label, {mlp, relu});

fprintf('Test, Error = %.2f, Accuracy = %.2f%%\n', ...
        test_nll, 100 * test_acc)
```

And these are the result.

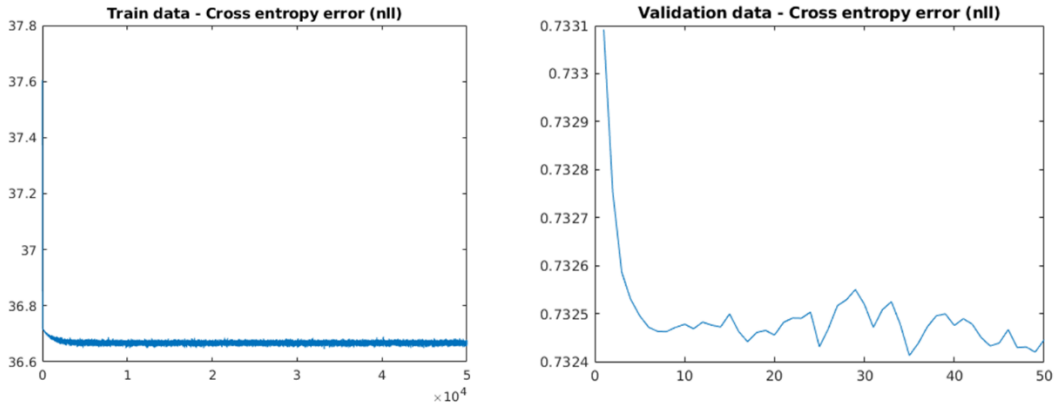
1. SAE CNN , Global Max pooling



Test, Error = 0.73, Accuracy = 16.67%

Validation accuracy was fixed on 20%.

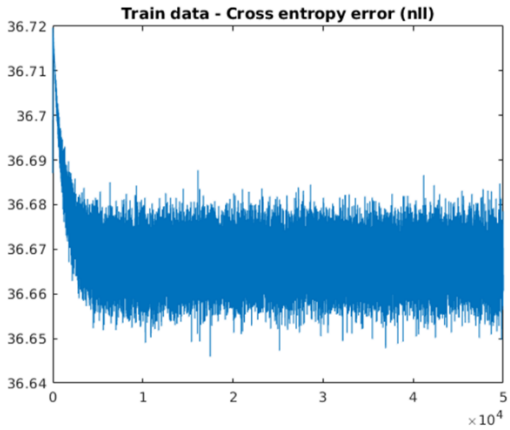
2. SAE CNN , Global Mean pooling

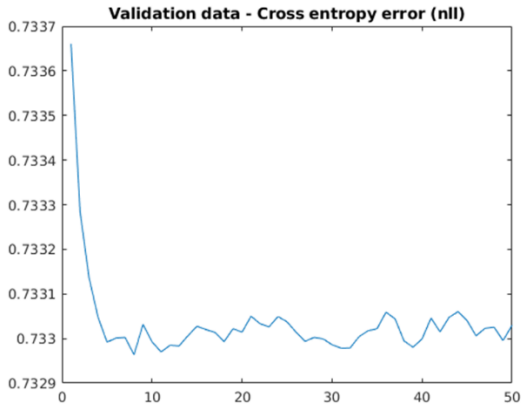


Test, Error = 0.73, Accuracy = 0%

Validation accuracy was fixed on 0%.

3. PCA CNN , Global Max pooling





Test, Error = 0.73, Accuracy = 16.67%

Validation accuracy was fixed on 0%.

I didn't do on PCA, Global Mean pooling model because it seemed unnecessary.

III. Evaluation

By number, it seems like SAE – Max pooling model is the best model. But I don't think accuracy is meaningful here, because We only had 6 cases. (16.67% = 1/6, 0% = 0/6). This logic also applies to validation accuracy too.

A meaningful observation we can have here is how cross entropy error decreases on train data. It seems like PCA model has met overfitting, while SAE model converged nicely.

I believe this model was not trained quite sufficiently, for I had to reduce the set size, and did not search for best hidden node number. I believe better environment and more trials can improve our model.