CMPT 741 – DATA MINING

Project - Sentiment Analysis with Convolutional Neural Network

Contents

In this report, we will discuss our approach, experiment setup and findings on using deep learning to perform Sentiment Analysis: predict whether a movie comment is positive or negative. Section 1 describes our convolution neural network architecture. Section 2 explains the dataset and experiment setup. In section 3, we will discuss experiment results. Section 4 summaries our findings and lesson learnt.

1. Model

Our deep learning model follows the Convolution Neural Network described by Yoon K. 2014 [1] and further explained by Denny B. 2015 [2]. The below figures are extracted from the Jiaxi. 2016 project revisit slides [3]:

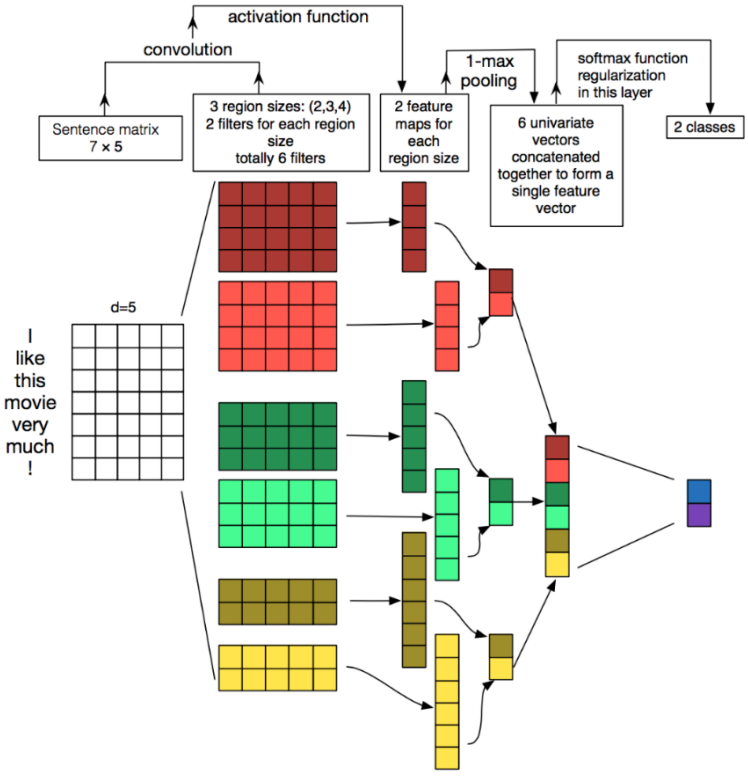


Figure 1.1 – Convolution Neural Network Model for NLP

Key steps to develop our convolutional neural network include:

- Data Preparation:
  - read in a dataset of sentences and get their corresponding matrix representation
  - split data in train/validation set
- Train Model
  - Set model parameters and initialize weights
  - Forward pass through convolution, pooling, concatenation, output and calculate softmax loss
  - Backward pass to calculate gradients for model weights and word vectors
- Evaluate Model
  - Apply model on validation set to get average validation accuracy and validation loss

1.1. Matrix Representation of Sentence:

If we can represent each word as a d-length vector, each sentence of N words would be represented as a Nxd matrix. The word vectors can be randomly initialized based on our vocabulary or obtained from pre-trained word embedding such as Word2Vec or GloVe [4]. In addition to matrix representation, we need to include a few special words to support convolution:

- Padding word <*PAD*>: A sentence should have minimum length of filter size so that convolution filter can be applied. For any sentence with length less than filter size, we pad the sentence with <PAD> word so that it meets minimum length required.
- Unknown word <*UNK*>: Test set might contain words that are not in the vocabulary built from training set. Vector representation for those words are the same as <UNK> word.

The word vectors can be updated using gradients during backpropagation.

1.2. Convolution layer and Activation function:

For each matrix represent a sentence, we can apply a convolution filter and apply a Rectified Linear Unit (ReLU) to extract features. The ReLU activation function is used instead of Sigmoid so that we can avoid vanishing gradient problem when training our network. The filter weights are randomly initialized and updated using gradients during backpropagation.

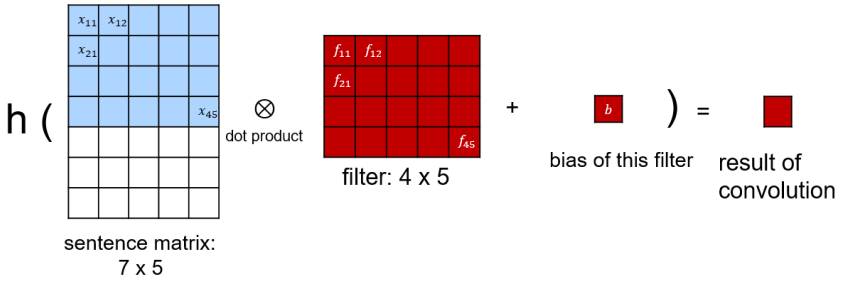


Figure 1.2.1 – Convolution Filter and Activation Function

1.3. Pooling layer:

Pooling layer helps extract the important features. In our network, we use 1-max pooling to get the most prominent feature.

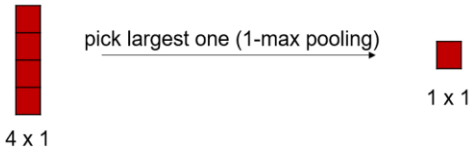
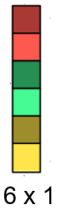


Figure 1.3.1 – Pooling Layer

1.4. Concatenation layer:

After getting most prominent feature by each filter, we combine all these features to obtain our feature set for each sentence. This is our penultimate layer.



1.5. Dropout and Output layer

With the penultimate layer containing feature set, we can perform different classification task. In this case, we connect it to the output layer, which has two nodes representing two classes (positive, negative). The output weights are initialized randomly and updated using gradients during backpropagation.

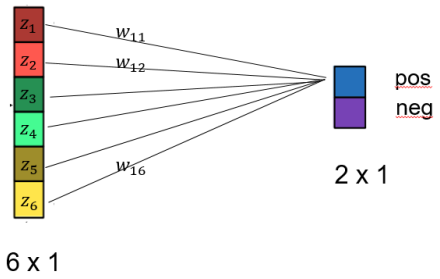


Figure 1.5.1 – Output layer

To reduce overfitting, we employ dropout layer right after the penultimate layer. Dropout prevents co-adaptation of hidden nodes by randomly dropping out a proportion of the hidden nodes during forward and back propagation, i.e some nodes are randomly masked and gradients are only backpropagated through the unmasked nodes.

1.6. Softmax Loss

Based on the output layer, we can calculate SoftMax loss and use the value to back-propagate and calculate gradients for filter weights, output weights and input word vectors.

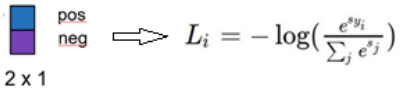


Figure 1.6.1 – Softmax loss based on output layer

2. Dataset and Experiment Setup

2.1. Dataset:

The training dataset contains 6000 movie comments with positive (1) or negative (0) label. We will use 5-fold cross validation and get 80/20 split for train/validation.

2.2. Experiment Setup

We train the model using **stochastic gradient descent**, i.e updating weights after each sentence, for 20 epochs and record the best validation accuracy. We include an early stop condition when the difference between the mean absolute value of validation softmax losses in two iterations is less than a threshold (0.0001). With **5-fold cross validation**, we take the average of 5 best validation accuracy as benchmark for our model comparison and parameter selection.

We define the convolution neural network with word vector randomly initialized as our **Random model**. Table 2.2.1 shows the list of parameters and their list of values that we experiment with this model. Ideally, we should benchmark our model across all permutations of different parameter values. However, for efficiency purpose, we only benchmark our model with incremental approach, i.e change one parameter at a time.

Parameter	List of values
Learning Rate	0.001, 0.01, 0.1
Word Vector length	5, 50, 100, 200, 300
Filter Size	[2,3,4], [2,3,4,5], [2,3,4,5,6]
Number of filters	2, 4, 6, 8, 10
Dropout Rate	0, 0.5

Table 2.2.1 – Model Parameters

In addition to the above **Random model**, we also experiment further with word embedding GLoVE, which contains word vectors pre-trained using Wikipedia 2014 text corpus of 400K vocabularies.

- **Static model**: We simply initialize word representation vectors using GLoVE but do not update these word vectors using backpropagation.
- **Non-static model**: We initialize word representation vectors using GLoVE and enable the training of word vectors using backpropagation through the same loop as training model weights
- **Continuous model**: We initialize word representation vectors using GLoVE and keep re-training the word vectors using an additional loop. We want to see if further improvement can be achieved, since learning rate for model parameters and learning rate for word vectors might be different.

3. Experiment Result

We start our experiment with the below initial parameters, and proceed to change one parameter at a time.

Learning Rate	Word Vector dimension	Filter Size	Number of Filter	Dropout
0.01	5	[2,3,4]	2	No dropout

3.1 Learning Rate:

We try varying the learning rate: 0.1, 0.01 and 0.001. The average validation accuracy figures are 0.627500, 0.633167, 0.505000 correspondingly.

If we take a loser look at each iteration, we can see that learning rate 0.001 is too slow and probably more prone to local minima. Learning rate 0.1 is too large, leading to a lot of oscillation. Hence, out of these three learning rates, we choose **learning rate 0.01** for our model.

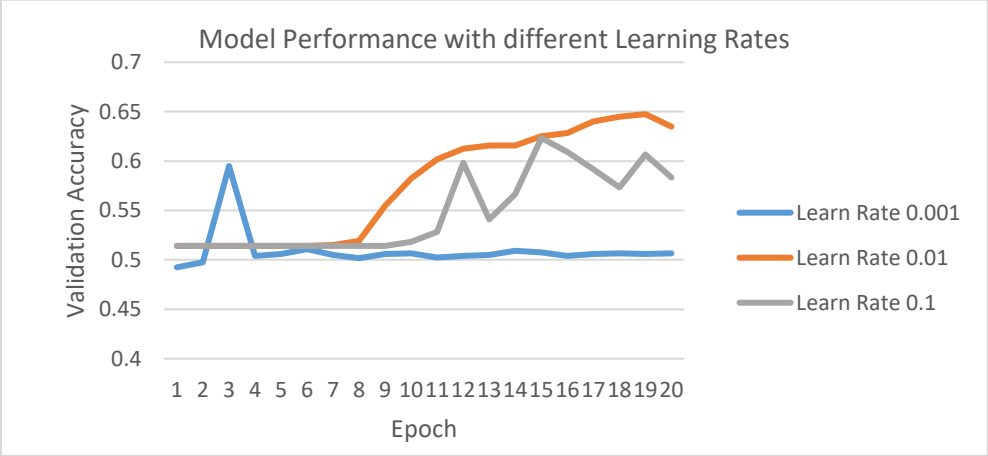


Figure 3.1.1 – Validation Accuracy in each epoch

3.2. Word Vector Dimension:

Table 3.2.1 shows validation accuracy achieved for different word vector dimensions:

Word vector dimension (d)	5	50	100	200	300
Validation Accuracy	0.633167	0.646000	0.657833	0.656667	0.655833

Table 3.2.1 – Validation Accuracy by Word vector dimension

There is little difference among word vector dimensions 100, 200 and 300. We choose the dimension with highest validation accuracy **d = 100** for our model.

3.3. Filter Size:

Table 3.3.1 shows validation accuracy achieved for different filter sizes:

Filter Size	[2,3,4]	[2,3,4,5]	[2,3,4,5,6]
Validation Accuracy	0.657833	0.658833	0.654500

Table 3.3.1 – Validation Accuracy by Filter Sizes

It makes little difference when we vary the filter size. We set the filter size with highest validation accuracy **[2,3,4,5]** for our model.

3.4. Number of Filters:

Table 3.4.1 shows validation accuracy for different number of filters per filter size: 0.674333

Number of filters	2	4	6	8	10
Validation Accuracy	0.658833	0.663500	0.679500	0.672000	0.668000

Table 3.4.1 – Validation Accuracy by Number of filters per filter size

6 filters per filter size give the best validation accuracy. We choose this value for our model.

3.5. Dropout Rate:

With no dropout, the model has validation accuracy of 0.679500. With dropout rate = 0.5, we have validation accuracy of 0.674333. Using dropout does not seem to make a difference, so we **disable dropout** for our model.

3.6. Word Embedding GLoVE:

Table 3.6.1 summaries the performance of different model variations:

Model	**Validation Accuracy**
Random (no GLoVE)	0.679500
Static (use GLoVE, do not train word vectors)	0.609000
Non-Static (use GLoVE, train word vectors)	0.723500
Continuous (use GLoVE, keep training word vectors	0.770800

Table 3.6.1 – Performance of different model variation

The Static model has validation accuracy of 0.609000. Clearly, the performance is much worse without training the word vectors. The Non-static model has validation accuracy of 0.723500, which is a significant improvement compared to model using randomly initialized word vector.

Since word vectors, which affect model inputs, and model parameters (filter weights, out put weights) have different roles in model performance, they might require different learning. In Continuous model, we keep training the word vectors by introducing another loop, as described below:

Initialize word vectors T using GLoVE
retrain_T = 30;
for retrain = 1:retrain_T
    randomly initialize filters and output weights
    for each training record
        update fitlers/weights and word vectors T through forward-backpropagation
    end for
end for

In each retrain iteration, word vectors T is not reinitialized but instead, kept re-training. Although this approach is more time consuming, it helps increase the validation accuracy to 0.770800, which is the best result we achieve so far. This is probably because training word vectors requires more updates or a different learning rate to adapt to our classification tasks.

4. Summary

We have described our experiment with convolutional neural network for sentiment analysis. Our best model (Continuous model) achieves 0.7708 validation accuracy and has the following parameters:

Learning Rate	Word Vector dimension	Filter Size	Number of Filter	Dropout
0.01	100	[2,3,4,5]	6	No Dropout

Our experiment shows that pre-trained word embedding such as GLoVE plays an important role in sentiment classification. The result also shows keep training the word embedding help improve the model performance significantly. However, this approach is time consuming and we should explore how to accelerate the training for word vectors as next step.

REFERENCES

1. Yoon K. 2014, *Convolutional Neural Networks for Sentence Classification* [link]
2. Denny B. 2015, *Understanding Convolutional Neural Network for NLP* [link]
3. Jiaxi. 2016, *Project Revisit* [link]
4. Jeffrey P., Richard S., and Christopher D. 2014, *Global Vector for Word Representation* [link]

APPENDIX – MATLAB CODE

List of Matlab files in the project:

- **read_data.m**: code provided by Jiaxi to read input and build vocabulary
- **vector_reprensentation.m**: code to read GLoVE word embedding. It requires GLoVE files for corresponding word vector dimensions of 50, 100, 200 and 300:

  glove.6B.50d.txt
  glove.6B.100d.txt
  glove.6B.200d.txt
  glove.6B.300d.txt

  These files can be downloaded and extracted from GLoVE website [link].
- **train_model_experiment.m**: code with cross validation to experiment with different model parameters
- **train_model_final.m**: code to train final model using chosen model parameters and keep retrain word vectors
- **apply_model.m**: apply the final model on a test set and write output to a file.

read_data.m

```
function [data, wordMap] = read_data()
% CMPT-741 example code for: reading training data and building vocabulary.
% NOTE: reading testing data is similar, but no need to build the vocabulary.
%
% return:
%       data(cell), 1st column -> sentence id, 2nd column -> words, 3rd column -> label
%       wordMap(Map), contains all words and their index, get word index by calling wordMap(word)

headLine = true;
separater = '::';

words = [];
data = cell(6000, 3);

fid = fopen('train.txt', 'r');
line = fgets(fid);

ind = 1;
while ischar(line)
   if headLine
      line = fgets(fid);
      headLine = false;
   end
   attrs = strsplit(line, separater);
   sid = str2double(attrs{1});

   s = attrs{2};
   w = strsplit(s);
   words = [words w];

   y = str2double(attrs{3});
```

```
    % save data
    data{ind, 1} = sid;
    data{ind, 2} = w;
    data{ind, 3} = y;

    % read next line
    line = fgets(fid);
    ind = ind + 1;
end
words = unique(words);
wordMap = containers.Map(words, 1:length(words));
fprintf('finish loading data and vocabulary\n');
```

vector_reprensentation.m

```
function glove = vector_representation(d)
% Read vector representation of word from pre-trained word vector GloVe
%
% return:
%       word vector GloVe

headLine = true;
fid = fopen(strcat('glove.6B.',num2str(d),'d.txt'), 'r');
line = fgets(fid);
separater = ' ';
attrs = strsplit(line, separater);
word = attrs{1};
vector = attrs(2:length(attrs));

glove = containers.Map;
ind = 1;
while ischar(line)
    if headLine
        line = fgets(fid);
        headLine = false;
    end

    attrs = strsplit(line, separater);
    word = attrs{1};
    vector = attrs(2:length(attrs));

    % save data
    glove(word) = vector;

    % read next line
    line = fgets(fid);
    ind = ind + 1;
end
fprintf('finish loading word vector GloVe\n');
```

train_model_experiment.m

```
%% CMPT-741 template code for: sentiment analysis base on Convolutional Neural Network
% author: llbui
% date: 27 Oct for release this code

clear; clc;

%% Section 1: preparation before training

% section 1.1 read file 'train.txt', load data and vocabulary by using function read_data()
[data,wordMap]=read_data;
wordMap('<PAD>') = length(wordMap) + 1;
wordMap('<UNK>') = length(wordMap) + 1;

% section 1.2 seperate dataset to training and validation sets
min_length = 5;
train_length = length(data) * 0.8;
validation_length = length(data) * 0.2;

data_indice = cell(length(data),2);
for i=1:length(data)
   sentence = data{i,2};
   sentence_length = length(sentence);
   % pad sentence shorter than filter length
   if sentence_length < min_length
      sentence2 = cell(1, min_length);
      for j=1:sentence_length
         sentence2{j} = sentence{j};
      end
      for j=(sentence_length+1):min_length
         sentence2{j} = '<PAD>';
      end
      sentence = sentence2;
   end
   label = data{i,3};

   indices = zeros(size(sentence));
   for j=1:length(indices)
      indices(j) = wordMap(sentence{j});
   end
   data_indice{i,1} = indices;

   if label == 1
      label = 1;
   else
      label = 2;
   end
   data_indice{i,2} = label;
end

data_indice = data_indice(randperm(length(data_indice)),:);
```

```
% init word embeding
d = 100;
use_glove = 1;

if use_glove == 1
   if (d ~= 50) && (d ~= 100) && (d ~= 200) && (d ~= 300)
      fprintf('GloVe can only be used with word dimension 50, 100, 200, 300. Proceeding without
Glove...\n');
      use_glove = 0;
   else
      fprintf('Loading Glove...\n');
      glove = vector_representation(d);
   end
end
if use_glove == 1
   % read word vector from GloVe word embedding
   wordMap_key = wordMap.keys();
   T = zeros(length(wordMap),d);
   for i = 1:length(wordMap_key)
      if isKey(glove,wordMap_key{i})
         T(wordMap(wordMap_key{i}),:) = str2double(glove(wordMap_key{i}));
      else
         T(wordMap(wordMap_key{i}),:) = normrnd(0,0.1,[1,d]);
      end
   end
   clear glove;
else
   % initialize word vector
   T = normrnd(0,0.1,[length(wordMap),d]);
end
T_original = T; %store pre-trained values of word vectors

% k - fold cross validation:
fold = 5;
overall_accuracy = 0;
for k=1:fold
   data_validation = data_indice(validation_length*(k-1)+1:validation_length*k,:);
   data_train = vertcat(data_indice(1:validation_length*(k-
1),:),data_indice(validation_length*k+1:end,:));
   T = T_original;

   % store best model
   best_accuracy = 0;
   best_w_out = 0;
   best_b_out = 0;
   best_w_conv = 0;
   best_b_conv = 0;
   best_T = T;

   % keep retrain T since it gives better result
   retrain_T = 1;
   for retrain = 1:retrain_T

      % ----------init filter
```

```
filter_size = [2,3,4,5];
n_filter = 6;

w_conv = cell(length(filter_size), 1);
b_conv = cell(length(filter_size), 1);

for i=1:length(filter_size)
   w_conv{i} = normrnd(0, 0.1, [filter_size(i), d, 1, n_filter]);
   b_conv{i} = zeros(n_filter, 1);
end

% init output layer
total_filters = length(filter_size) * n_filter;
n_class = 2;
w_out = normrnd(0, 0.1, [total_filters, n_class]);
b_out = zeros(n_class, 1);

% learning parameters
step_size = 1e-2; % learning rate
reg = 0.5; % regularization with dropout rate
max_iter = 20;
threshold = 0.0001; % early stop threshold
prev_loss = 0;

for iter=1:max_iter
   preds = zeros(train_length,1);
   actuals = zeros(train_length,1);

%% Section 2: training
% Note:
% you may need the resouces [2-4] in the project description.
% you may need the follow MatConvNet functions:
%       vl_nnconv(), vl_nnpool(), vl_nnrelu(), vl_nnconcat(), and vl_nnloss()

   % for each example in train.txt do
   for ind = 1:train_length

      % section 2.1 forward propagation and compute the loss
      %get sentence matrix
      word_indices = data_train{ind,1};
      y = data_train{ind, 2};
      X = T(word_indices,:);

      pool_res = cell(1, length(filter_size));
      cache = cell(2, length(filter_size));
      for i = 1: length(filter_size)
         %convolution operation
         conv = vl_nnconv(X, w_conv{i}, b_conv{i});

         %activation reLu
         relu = vl_nnrelu(conv);

         % 1-max pooling
         sizes = size(conv);
```

```
            pool = vl_nnpool(relu, [sizes(1), 1]);

            % keep values for back-propagate
            cache{2,i} = relu;
            cache{1,i} = conv;
            pool_res{i} = pool;
        end

        % concatenate
        z = vl_nnconcat(pool_res, 3);

        % dropout layer for regularization
        %vl_nndropout(z, 'rate', reg);
        %[z2,MASK] = vl_nndropout(reshape(z, [total_filters,1]));

        % compute output layer
        o = vl_nnconv(reshape(z, [total_filters,1]), reshape(w_out,[total_filters,1,1,2]), b_out);
        actuals(i) = y;
        [~, preds(i)] = max(o);

        % compute loss
        loss = vl_nnloss(o, y);

        % section 2.2 backward propagation and compute the derivatives
        do = vl_nnloss(o, y, 1);
        [dz,dw_out, db_out] = vl_nnconv(reshape(z, [total_filters,1]),
reshape(w_out,[total_filters,1,1,2]), b_out,do);
        %dz = vl_nndropout(z, dz2, 'mask', MASK);
        dpool_res = vl_nnconcat(pool_res, 3, reshape(dz, [1,1,total_filters]));
        cache_conv = cell(3, length(filter_size));
        for i = 1: length(filter_size)
            dpool = dpool_res{i};
            sizes = size(cache{1,i});
            drelu = vl_nnpool(cache{2,i}, [sizes(1), 1], dpool);
            dconv = vl_nnrelu(cache{1,i}, drelu);
            [dx, dw_conv, db_conv] = vl_nnconv(X, w_conv{i}, b_conv{i}, dconv);

            % keep value for  parameters update
            cache_conv{1,i} = dx;
            cache_conv{2,i} = dw_conv;
            cache_conv{3,i} = db_conv;
        end

        % section 2.3 update the parameters
        % update filter and word vector
        for i=1:length(filter_size)
            X = X - step_size*cache_conv{1,i};
            w_conv{i} = w_conv{i} - step_size*cache_conv{2,i};
            b_conv{i} = b_conv{i} - step_size*cache_conv{3,i};
        end
        for i=1:length(word_indices)
            T(word_indices(i),:) = X(i,:);
        end
```

```
        % update output layer
        w_out = w_out - step_size*reshape(dw_out,[total_filters, n_class]);
        b_out = b_out - step_size*db_out;
    end
%% Section 3: evaluate prediction
    % approximate train accuracy. This does not rerun for training set
    accuracy = length(find(preds==actuals))/train_length;
    fprintf('iter: %d - train accuracy: %f, ', iter, accuracy);

    % validation accuracy
    total_loss = 0;
    correct = 0;
    for ind = 1:validation_length
        %get sentence matrix
        word_indices = data_validation{ind,1};
        y = data_validation{ind, 2};
        X =T(word_indices,:);

        pool_res = cell(1, length(filter_size));
        cache = cell(2, length(filter_size));
        for i = 1: length(filter_size)
            %convolution operation
            conv = vl_nnconv(X, w_conv{i}, b_conv{i});

            %activation reLu
            relu = vl_nnrelu(conv);

            % 1-max pooling
            sizes = size(conv);
            pool = vl_nnpool(relu, [sizes(1), 1]);

            % keep values for back-propagate
            cache{2,i} = relu;
            cache{1,i} = conv;
            pool_res{i} = pool;
        end

        % concatenate
        z = vl_nnconcat(pool_res, 3);

        % dropout layer for regularization
        %vl_nndropout(z, 'rate', reg);
        %[z2,MASK] = vl_nndropout(reshape(z, [total_filters,1]));

        % ouput layer
        o = vl_nnconv(reshape(z, [total_filters,1]), reshape(w_out,[total_filters,1,1,2]), b_out);
        [~,pred] = max(o);
        if pred == y
            correct = correct + 1;
        end

        % loss
        loss = vl_nnloss(o, y);
        total_loss = total_loss + abs(loss);
```

```
            end

            accuracy = correct/validation_length;
            total_loss = total_loss/validation_length;
            fprintf('validation accuracy: %f , validation loss: %f\n', accuracy, total_loss);
            if accuracy > best_accuracy
               best_accuracy = accuracy;
               best_w_out = w_out;
               best_b_out = b_out;
               best_w_conv = w_conv;
               best_b_conv = b_conv;
               best_T = T;
            end

            % stop if loss difference is less than threshold
            if abs(total_loss - prev_loss) < threshold
               fprintf('Early stop... \n');
               prev_loss = 0;
               break;
            end
            prev_loss = total_loss;
         end
         fprintf('k=%d: best validation accuracy: %f \n', k, best_accuracy);
      end

      overall_accuracy = overall_accuracy + best_accuracy;
   end
   overall_accuracy = overall_accuracy/fold;
   fprintf('Overall validation accuracy: %f \n', overall_accuracy)
```

train_model_final.m

```
%% CMPT-741 template code for: sentiment analysis base on Convolutional Neural Network
% author: llbui
% date: 27 Oct for release this code

clear; clc;

%% Section 1: preparation before training

% section 1.1 read file 'train.txt', load data and vocabulary by using function read_data()
[data,wordMap]=read_data;
wordMap('<PAD>') = length(wordMap) + 1;
wordMap('<UNK>') = length(wordMap) + 1;

% section 1.2 seperate dataset to training and validation sets
min_length = 5;
train_length = length(data) * 0.8;
validation_length = length(data) - train_length;

data_indice = cell(length(data),2);
for i=1:length(data)
   sentence = data{i,2};
   sentence_length = length(sentence);
```

```
    % pad sentence shorter than filter length
    if sentence_length < min_length
        sentence2 = cell(1, min_length);
        for j=1:sentence_length
            sentence2{j} = sentence{j};
        end
        for j=(sentence_length+1):min_length
            sentence2{j} = '<PAD>';
        end
        sentence = sentence2;
    end
    label = data{i,3};

    indices = zeros(size(sentence));
    for j=1:length(indices)
        indices(j) = wordMap(sentence{j});
    end
    data_indice{i,1} = indices;

    if label == 1
        label = 1;
    else
        label = 2;
    end
    data_indice{i,2} = label;
end

data_indice = data_indice(randperm(length(data_indice)),:);
data_train = data_indice(1:train_length,:);
data_validation = data_indice(train_length+1:end,:);

% init word embeding
d = 100;
use_glove = 1;

if use_glove == 1
    if (d ~= 50) && (d ~= 100) && (d ~= 200) && (d ~= 300)
        fprintf('GloVe can only be used with word dimension 50, 100, 200, 300. Proceeding without
Glove...\n');
        use_glove = 0;
    else
        fprintf('Loading Glove...\n');
        glove = vector_representation(d);
    end
end
if use_glove == 1
    % read word vector from GloVe word embedding
    wordMap_key = wordMap.keys();
    T = zeros(length(wordMap),d);
    for i = 1:length(wordMap_key)
        if isKey(glove,wordMap_key{i})
            T(wordMap(wordMap_key{i}),:) = str2double(glove(wordMap_key{i}));
        else
            T(wordMap(wordMap_key{i}),:) = normrnd(0,0.1,[1,d]);
```

```
        end
    end
    clear glove;
else
    % initialize word vector
    T = normrnd(0,0.1,[length(wordMap),d]);
end

% store best model
best_accuracy = 0;
best_w_out = 0;
best_b_out = 0;
best_w_conv = 0;
best_b_conv = 0;
best_T = 0;

retrain_T = 10;
for retrain = 1:retrain_T
    % init filter
    filter_size = [2,3,4,5];
    n_filter = 6;

    w_conv = cell(length(filter_size), 1);
    b_conv = cell(length(filter_size), 1);

    for i=1:length(filter_size)
        w_conv{i} = normrnd(0, 0.1, [filter_size(i), d, 1, n_filter]);
        b_conv{i} = zeros(n_filter, 1);
    end

    % init output layer
    total_filters = length(filter_size) * n_filter;
    n_class = 2;
    w_out = normrnd(0, 0.1, [total_filters, n_class]);
    b_out = zeros(n_class, 1);

    % learning parameters
    step_size = 1e-2; % learning rate
    reg = 1e-1; % regularization
    max_iter = 20;
    threshold = 0.0001; % early stop threshold
    prev_loss = 0;

    for iter=1:max_iter
        preds = zeros(train_length,1);
        actuals = zeros(train_length,1);

    %% Section 2: training
    % Note:
    % you may need the resouces [2-4] in the project description.
    % you may need the follow MatConvNet functions:
    %       vl_nnconv(), vl_nnpool(), vl_nnrelu(), vl_nnconcat(), and vl_nnloss()

        % for each example in train.txt do
```

```
for ind = 1:train_length

    % section 2.1 forward propagation and compute the loss
    %get sentence matrix
    word_indices = data_train{ind,1};
    y = data_train{ind, 2};
    X = T(word_indices,:);

    pool_res = cell(1, length(filter_size));
    cache = cell(2, length(filter_size));
    for i = 1: length(filter_size)
      %convolution operation
      conv = vl_nnconv(X, w_conv{i}, b_conv{i});

      %activation reLu
      relu = vl_nnrelu(conv);

      % 1-max pooling
      sizes = size(conv);
      pool = vl_nnpool(relu, [sizes(1), 1]);

      % keep values for back-propagate
      cache{2,i} = relu;
      cache{1,i} = conv;
      pool_res{i} = pool;
    end

    % concatenate
    z = vl_nnconcat(pool_res, 3);


    % compute output layer
    o = vl_nnconv(reshape(z, [total_filters,1]), reshape(w_out,[total_filters,1,1,2]), b_out);
    actuals(i) = y;
    [~, preds(i)] = max(o);

    % compute loss
    loss = vl_nnloss(o, y);

    % section 2.2 backward propagation and compute the derivatives
    do = vl_nnloss(o, y, 1);
    [dz,dw_out, db_out] = vl_nnconv(reshape(z, [total_filters,1]),
reshape(w_out,[total_filters,1,1,2]), b_out,do);
    dpool_res = vl_nnconcat(pool_res, 3, reshape(dz, [1,1,total_filters]));
    cache_conv = cell(3, length(filter_size));
    for i = 1: length(filter_size)
      dpool = dpool_res{i};
      sizes = size(cache{1,i});
      drelu = vl_nnpool(cache{2,i}, [sizes(1), 1], dpool);
      dconv = vl_nnrelu(cache{1,i}, drelu);
      [dx, dw_conv, db_conv] = vl_nnconv(X, w_conv{i}, b_conv{i}, dconv);

      % keep value for  parameters update
      cache_conv{1,i} = dx;
```

```
            cache_conv{2,i} = dw_conv;
            cache_conv{3,i} = db_conv;
        end

        % section 2.3 update the parameters
        % update filter and word vector
        for i=1:length(filter_size)
            X = X - step_size*cache_conv{1,i};
            w_conv{i} = w_conv{i} - step_size*cache_conv{2,i};
            b_conv{i} = b_conv{i} - step_size*cache_conv{3,i};
        end
        for i=1:length(word_indices)
            T(word_indices(i),:) = X(i,:);
        end

        % update output layer
        w_out = w_out - step_size*reshape(dw_out,[total_filters, n_class]);
        b_out = b_out - step_size*db_out;
    end
%% Section 3: evaluate prediction
    % approximate train accuracy. This does not rerun for training set
    accuracy = length(find(preds==actuals))/train_length;
    fprintf('iter: %d - train accuracy: %f, ', iter, accuracy);

    % validation accuracy
    total_loss = 0;
    correct = 0;
    for ind = 1:validation_length
        %get sentence matrix
        word_indices = data_validation{ind,1};
        y = data_validation{ind, 2};
        X =T(word_indices,:);

        pool_res = cell(1, length(filter_size));
        cache = cell(2, length(filter_size));
        for i = 1: length(filter_size)
            %convolution operation
            conv = vl_nnconv(X, w_conv{i}, b_conv{i});

            %activation reLu
            relu = vl_nnrelu(conv);

            % 1-max pooling
            sizes = size(conv);
            pool = vl_nnpool(relu, [sizes(1), 1]);

            % keep values for back-propagate
            cache{2,i} = relu;
            cache{1,i} = conv;
            pool_res{i} = pool;
        end

        % concatenate
        z = vl_nnconcat(pool_res, 3);
```

```
            % ouput layer
            o = vl_nnconv(reshape(z, [total_filters,1]), reshape(w_out,[total_filters,1,1,2]), b_out);
            [~,pred] = max(o);
            if pred == y
                correct = correct + 1;
            end

            % loss
            loss = vl_nnloss(o, y);
            total_loss = total_loss + abs(loss);

        end

        accuracy = correct/validation_length;
        total_loss = total_loss/validation_length;
        fprintf('validation accuracy: %f , validation loss: %f\n', accuracy, total_loss);
        if accuracy > best_accuracy
            best_accuracy = accuracy;
            best_w_out = w_out;
            best_b_out = b_out;
            best_w_conv = w_conv;
            best_b_conv = b_conv;
            best_T = T;
        end

        % stop if loss difference is less than threshold
        if abs(total_loss - prev_loss) < threshold
            fprintf('Early stop... \n');
            prev_loss = 0;
            break;
        end
        prev_loss = total_loss;
    end
    fprintf('Retrain %d: best validation accuracy: %f \n', retrain, best_accuracy);
end
```

apply_model.m

```
%% Predict sentiments
% author: llbui
% date: 27 Oct for release this code

%% Section 1: read test data
% section 1.1 read file 'sample_test.txt' and load data
fileName = 'sample_test.txt';
headLine = true;
separater = '::';

data_test = cell(1000, 2);

fid = fopen(fileName, 'r');
line = fgets(fid);
```

```
ind = 1;
while ischar(line)
    if headLine
        line = fgets(fid);
        headLine = false;
    end
    attrs = strsplit(line, separater);
    sid = str2double(attrs{1});

    s = attrs{2};
    w = strsplit(s);

    % save data
    data_test{ind, 1} = sid;
    data_test{ind, 2} = w;

    % read next line
    line = fgets(fid);
    ind = ind + 1;
end

% section 1.2 handle data
min_length = 5;
data_test_indices = cell(length(data_test));

for i=1:length(data_test)
    sentence = data_test{i,2};
    sentence_length = length(sentence);
    % pad sentence shorter than filter length
    if sentence_length < min_length
        sentence2 = cell(1, min_length);
        for j=1:sentence_length
            sentence2{j} = sentence{j};
        end
        for j=(sentence_length+1):min_length
            sentence2{j} = '<PAD>';
        end
        sentence = sentence2;
    end

    indices = zeros(size(sentence));
    for j=1:length(indices)
        if isKey(wordMap,sentence{j})
            indices(j) = wordMap(sentence{j});
        else
            indices(j) = wordMap('<UNK>');
        end
    end
    data_test_indices{i} = indices;
end

%% Section 2: apply model
output = cell(length(data_test),2);
for ind = 1:length(data_test)
```

```
    %get sentence matrix
    word_indices = data_test_indices{ind,1};
    X =best_T(word_indices,:);

    pool_res = cell(1, length(filter_size));
    cache = cell(2, length(filter_size));
    for i = 1: length(filter_size)
        %convolution operation
        conv = vl_nnconv(X, best_w_conv{i}, best_b_conv{i});

        %activation reLu
        relu = vl_nnrelu(conv);

        % 1-max pooling
        sizes = size(conv);
        pool = vl_nnpool(relu, [sizes(1), 1]);

        % keep values for back-propagate
        cache{2,i} = relu;
        cache{1,i} = conv;
        pool_res{i} = pool;
    end

    % concatenate
    z = vl_nnconcat(pool_res, 3);

    % ouput layer
    o = vl_nnconv(reshape(z, [total_filters,1]), reshape(best_w_out,[total_filters,1,1,2]), best_b_out);
    [~,pred] = max(o);

    % output
    output{ind,1} = data_test{ind,1};
    if pred == 1
        output{ind,2} = 1;
    else
        output{ind,2} = 0;
    end
end

%% Section 3: write output to file
textHeader = 'id::label'; % header
fid = fopen(strcat('submission_',fileName),'w');
fprintf(fid,'%s\n',textHeader);
for ind = 1:length(output)
    fprintf(fid,'%i::%i\n',output{ind,1},output{ind,2});
end
fclose(fid);
```