Sentiment Analysis based on Convolutional Neural Networks

Fan Liu, Jin Zhang

Simon Fraser University

1 Introduction

With the increasing ubiquity of review sites like Yelp and Rotten Tomatoes, there are more and more personal comments left on the public Internet. Sentiment analysis, which uses natural language processing (NLP) and machine learning techniques, becomes an important method to classify rapidly increasing subjective opinionated data. In this project, we conduct a sentiment analysis using one-layer Convolutional Neural Networks (CNNs) to develop a sentence classification model. We focus on one-layer CNNs because it has strong empirical performance and comparative simple computation.

To conduct sentiment analysis of semantical words, the words should be converted to word vectors firstly. Word vectors are words in the sentences that are projected from a sparse, 1-of-V encoding (here V is the vocabulary size) onto a lower dimensional vector space. They are feature extractors that encode semantic features of words in their dimensions, and are essential parts in the development of a sentence classification model [1]. In this project, we employ GloVe pre-trained word vectors. This model development is based on a training set with 6,000 sentences, and each sentence is labeled as 1 (positive) or 0 (negative). The main tasks in our model development using CNNs include specifying a model architecture and setting accompanying hyperparameters, where the accompanying hyperparameters include the filter size, the number of filters, the cross-validation fold, the activation function, the pooling strategy and the regularization terms. We experimented several groups of accompanying hyperparameters to raise the accuracy of sentence classification.

2 Model

Our CNN architecture construction starts with the initialization of word vector representations. We take advantage of the result from Global Vectors for Word Representation (GloVe) [2], which is an unsupervised learning algorithm for obtaining vector representation for words. We use the pre-trained GloVe word vectors

where dimensionality of each word is 300. Moreover, in case of the input sentence is with inappropriate size or includes unknown words, we also create two special symbols and add them to the word vectors (details in the section Dataset). Since there are totally 14,099 words in vocabulary of the training set (14,097 words and 2 special symbols, details in the section Dataset), the word vectors are represented as a $14,099 \times 300$ cell and each row for a word. In this way, a sentence with n words can be convert to a $n \times 300$ word vector cell.

We use eighteen filters in the CNN, six of them with filter size 3, six of them with filter size 4 and six of them with filter size 5. Each filter has same width as the dimensionality of word vectors. The weights $\boldsymbol{w}$ in the filters are initialized with the value generated from a normal distribution with mean 0 and variance 0.1.

For each sentence in the training set, the six filters are applied to extract the sentence features. Suppose a sentence has n words (denoted as $\boldsymbol{x}_1$ to $\boldsymbol{x}_n$), for the convolution operation involves a filter with filter size h and weights $\boldsymbol{w}$, the filter is applied to a window of h words in the sentence to calculate feature $\boldsymbol{c}$. For example, for a window of words $\boldsymbol{x}_i$ to $\boldsymbol{x}_{i+h-1}$, the function to calculate $\boldsymbol{c}_i = \boldsymbol{w} \times \boldsymbol{x}_{i:i+h-1} + \boldsymbol{b}$, where $\boldsymbol{b}$ is a bias term. In this way, the sentence generates features $\boldsymbol{c}_1, \boldsymbol{c}_2, ..., \boldsymbol{c}_{n-h+1}$ [1]. Then we apply ReLU pooling operation over these features to capture the most import feature. This process is replicated for all eighteen filters and finally we obtain eighteen features for a sentence. At the end, these features are concatenated and passed to softmax layer to calculate the probability distribution over the two labels. After the predicted label of sentence is determined, we compare it with the true label of this sentence and compute the loss of convolution operation. With the loss, we apply back propagation to compute the derivatives of parameters in the word vector cell, convolution layer and softmax layer, and hence to update these parameters after each sentence. The above procedure is replicated for all sentences in the training set employing the stochastic gradient descent. To control overfitting problem, the dropout and a constraint on l_2-norm of the weights for regularization are used. Dropout is applied on the penultimate layer, which drops out some hidden units during forward propagation, in order to prevent co-adaption of hidden-units. In addition, we constrain l_2-norms of the weights vector before the output layer, by rescaling $\boldsymbol{W}_{out}$ to the value of 3 if the 2-norm of $\boldsymbol{W}_{out}$ is greater than 3 when updating $\boldsymbol{W}_{out}$.

To increase the accuracy on training set and testing set, we iterate our CNN learning process 10 times until the accuracy reach convergence.

3 Dataset

The dataset for this project are 6,000 labeled sentences about movie reviews. The records in the given file train.txt include the sentence ID, the content of the sentence and the label of the sentence, which are separated by "::". Each sentence is labeled as 1 (positive) or 0 (negative). The average sentence length is 20.

Our model development begin with the tokenization of sentence, which allow us to get the word list of all words appeared in the training dataset. We also add two new symbols into the world list. The first one is <PAD>, which is used to make up the shortage of word if the length of a sentence is smaller than the minimum filter size;

Description	Values
word vectors	GloVe
filter size	(3, 4, 5)
filter number	2
activation function	ReLU
pooling	1-max pooling
dropout rate	0.5
l_2-norm constraint	3

Table 1. Baseline experiment configuration.

the second one is <UNK>, which is used to represent an unknown word if it appears in the validation dataset. The vocabulary size of our dataset is 14099, and 13426 of them are present in the set of pre-trained word vectors.

4 Results and discussion

4.1 Baseline result

For the baseline CNN experiment, we modify the previous configuration and hyperparameters [1, 3] slightly, using GloVe as the input word vectors, (2, 3, 4) as the filter size, and 2 as the filter number for each filter size, which are adjusted to our dataset size. The parameters are described in Table 1. The experiments were performed under 5-fold cross validation, as the given dataset contains 6000 sentences and testing dataset contains 1000 sentences. The accuracy on validation dataset of the baseline experiment is 74.07%.

With the baseline performance obtained, we now try to optimize our CNN model by considering the effect of various architecture configurations and parameters. Experiments were performed by varying the parameter of interest, while keeping all other parameters constant. Each experiment validation accuracy was obtained from the mean of 5-fold cross validation results.

4.2 Word Embedding

We started with initializing the word vectors with random values, which were trained on the training dataset and modified during back propagation. We then tried word vector representations from pre-trained GloVe models to improve performance in the absence of a large training set. GloVe vectors were trained on 6 billion tokens, with dimensionality of 50, 100, 200, or 300 and trained on 840 billion tokens with dimensionality of 300. The pre-trained GloVe word vectors were learned for each sentence. Results are reported in Table 2.

The validation accuracy increases from 66.12% when using random 300-dimensional word vectors to 74.05% with 300-dimensional GloVe word embedding. Comparing results among experiments with GloVe, better performance is achieved with greater word vector dimensionality. However, whether the GloVe vectors were trained from 6 billion tokens or 840 billion tokens does not make an obvious difference on the result.

Word embedding	Accuracy
random.300d	66.12%
GloVe.6B.50d	67.35%
GloVe.6B.100d	71.77%
GloVe.6B.200d	73.28%
GloVe.6B.300d	74.05%
GloVe.840B.300d	74.07%

Table 2. Accuracies using different word vectors. Random.300d means the word vectors are randomly initialized and the dimension for a word is 300. GloVe.6B means the word vectors were pre-trained on 6 billion tokens while GloVe.840B means 840 billion tokens.

4.3 Number of filter for each filter size

We then investigate the effect of the number of filter for each filter size, while holding other parameters constant, where the filter size is (2, 3, 4). We tried the filter number of 2, 4, 6, 8, 10, 20, 30 and 50, and the results are reported in Figure 1.

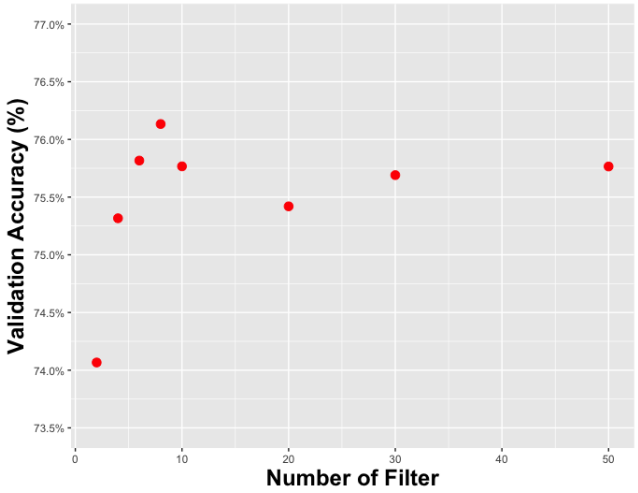

Fig. 1. Accuracy results with different numbers of filter.

The accuracy increases from 74.06% for filter number of 2 to 76.13% for filter number of 8, but does not change much when using greater filter number. We know that the greater filter number means greater number of parameters, leading to the the model more susceptible to overfitting although regularization and a dropout layer are used. Taking this into consideration, we choose 6 as the filter number for our optimal model.

4.4 Filter size

We run some more experiments to choose the optimal filter size. We started with experiments using only one filter size, 1, 3, 5, 7, 9, and as usual the number of filter is set to 2. We also tried the combination of different filter sizes, which are (2, 3, 4), (3, 4, 5), (4, 5, 6), (5, 6, 7). The results are reported in Table 3.

Filter size	Accuracy
1	70.02%
3	71.08%
5	71.44%
7	71.48%
9	68.85%
(2, 3, 4)	74.07%
(3, 4, 5)	74.28%
(4, 5, 6)	74.21%
(5, 6, 7)	74.17%

Table 3. Accuracy results using different filter size.

For the experiments with single filter size, we can see that the difference of accuracy among filter size of 3, 5, 7 is small. However, during the learning process, the validation accuracy for filter size of 7 oscillates a lot, which means filter size greater than 7 is not suitable for our dataset. The filter size (3, 4, 5) is selected for the optimal model.

4.5 Regularization

In our baseline model, we use two common regularization strategies, adding dropout layer and l_2-norm constraints. We run experiments to explore the effect of the dropout rate by setting it to 0.0, 0.1, 0.3, 0.5, 0.7 and 0.9, while the l_2-norm constraint is set to 3. The results are illustrated in Figure 2.

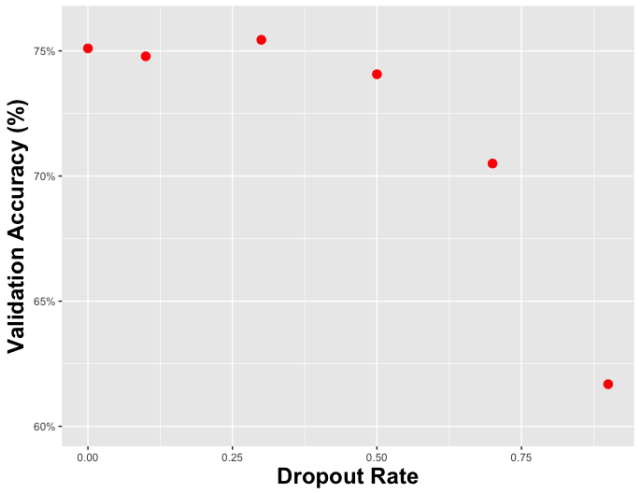

Fig. 2. Accuracy results with different dropout rates.

When the rate is set to the value of 0.0 or 0.1, a good validation accuracy can be reached during the learning process, but the training accuracy increases fast and over fitting occurs within 10 iterations. When the rate is set to

0.7 or 0.9, although over fitting is controlled, the validation accuracy is not good. 0.3 or 0.5 is a reasonable choice of the dropout rate, considering the tradeoff between controlling overfitting and accuracy.

5 Conclusion

We have conducted sentiment analysis of semantical words based on CNNs. To obtain an optimal model, experiments were performed to choose the best parameters for the given dataset. The best model is built with three filter sizes (3, 4, 5), each of which has 6 filters, using 1-max pooling and ReLU activation function, setting l_2-norm constraint to 3 and dropout rate to 0.5 for regularization. The accuracy for training set is around 88%, and the accuracy result of our model during the demo is 75.6%.

For the regularization, although using dropout keeps the training accuracy from reaching 98% within 10 iterations, the validation accuracy does not increase obviously. We can study the dropout function deeply to make sure the usage of the dropout strategy is correct. For future study, we can further optimize the model by changing the learning rate during the gradient descent, and investigate the effect of different activation functions.

References

[1] Kim, Y., 2014. "Convolutional neural networks for sentence classification". *CoRR,* **abs/1408.5882**.

[2] Mikolov, T., Chen, K., Corrado, G., and Dean, J., 2013. "Efficient estimation of word representations in vector space". *CoRR,* **abs/1301.3781**.

[3] Zhang, Y., and Wallace, B. C., 2015. "A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification". *CoRR,* **abs/1510.03820**.

Appendix A: Python code to construct word vectors for the vocabulary of the given dataset

```
import os
import numpy as np
from tempfile import TemporaryFile
BASE_DIR = '.'
GLOVE_DIR = BASE_DIR + '/glove.840B/'

embeddings_index = {}

f = open(os.path.join(GLOVE_DIR, 'glove.840B.300d.txt'))
for line in f:
    values = line.split()
    word = values[0]
    coefs = np.asarray(values[1:], dtype='float32')
    embeddings_index[word] = coefs
f.close()

print('Found %s word vectors.' % len(embeddings_index))

embedding_matrix = 0.1 * np.random.randn(14099, 300)

vocab = open(os.path.join(BASE_DIR, 'vocab.dat'))

j = 0
for line in vocab:
        ls = line.split(' ', 1)
        i = int(ls[0])
        word = ls[1]
        embedding_vector = embeddings_index.get(word.rstrip())
        if embedding_vector is not None:
                embedding_matrix[i-1] = embedding_vector
                j = j + 1

np.savetxt('wordvector_300_840B.txt', embedding_matrix)
```

Appendix B: Matlab training code

```
%% CMPT-741 course project: sentiment analysis base on Convolutional Neural
   Network
% author: Liu Fan, Jin Zhang
% date: Dec. 7, 2016

clear; clc;
rng(1234);
%% Section 1: preparation before training

% section 1.1 read file 'train.txt', load data and vocabulary by using function
   read_data()

% read data
[data, wordMap] = read_data();

% pad data to make sure the sentence is longer than the filter size
wordMap('<PAD>') = length(wordMap) + 1;
wordMap('<UNK>') = length(wordMap) + 1;

% save wordMap to
% fileID = fopen('vocab.dat','w');
% formatSpec = '%d %s\n';
% vocab = wordMap.keys();
% [nrows,ncols] = size(vocab);
% for col = 1:ncols
%     fprintf(fileID,formatSpec,wordMap(vocab{col}), vocab{col});
% end
% fclose(fileID);

% init embedding
d = 300;
total_words = length(wordMap);

% init filters
filter_size = [3,4,5];
n_filter = 6;
total_filters = length(filter_size) * n_filter;
n_class = 2;

% update the data to include <PAD> for short sentences and <UNK> according
% to filter size
for i = 1: length(data)
    sentence = data{i, 2};
    if length(sentence) < 5
        for k = length(sentence)+1: 5
            sentence{k} = '<PAD>';
```

```
        end
    end
    for j = 1: length(sentence)
        if (isKey(wordMap, sentence{j}) == 0)
            sentence{j} = '<UNK>';
        end
    end
    data{i, 2} = sentence;
end

% init gradient descent parameters
numIterations = 10;
rate = 0.01;

%% Section 2: training

% set parameter for 5-fold cross validation
fold = 5;
fold_size = length(data) / fold;

% cross validation
for fold_index = 0 : (fold - 1)
    fprintf('********************** (@_@) ********************** \n');
    fprintf('This is cross-validation with fold %i as validation set \n',
        fold_index+1);

    % separate data into training_data and validation_data
    training_data = [data(1 : fold_size*fold_index, :); data(fold_size*(fold_index
        +1)+1 : end, :)];
    validation_data = data(fold_size*fold_index+1 : fold_size*(fold_index+1), :);

    % init parameters
    T = importdata('wordvector_300_840B.txt');
    % T = normrnd(0, 0.1, [total_words, d]);
    W_conv = cell(length(filter_size), 1);
    B_conv = cell(length(filter_size), 1);

    for i = 1: length(filter_size)
        % get filter size
        f = filter_size(i);
        % init W with: FW x FH x FC x K
        W_conv{i} = normrnd(0, 0.1, [f, d, 1, n_filter]);
        B_conv{i} = zeros(n_filter, 1);
    end

    % init output layer
    W_out = normrnd(0, 0.1, [total_filters, n_class]);
    B_out = zeros(n_class, 1);
```

```
loss = cell(1, numIterations);
for t = 1: numIterations
    loss_t = 0;
    for i = 1: length(training_data)
        sentence = training_data{i, 2};

        % get sentence matrix
        % words_indexs = [wordMap('i'), wordMap('like'),
        % ..., wordMap('!')]
        word_indexs = zeros(1, length(sentence));
        for j = 1: length(sentence)
            word_indexs(j) = wordMap(sentence{j});
        end
        X = T(word_indexs, :);

        % section 2.1 forward propagation and compute the loss
        pool_res = cell(1, length(filter_size));
        cache = cell(2, length(filter_size));
        for k = 1: length(filter_size)
            % convolutional operation
            conv = vl_nnconv(X, W_conv{k}, B_conv{k});

            % apply activation function: relu
            relu = vl_nnrelu(conv);

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

            % important: keep these values for back-prop
            cache{2, k} = relu;
            cache{1, k} = conv;
            pool_res{k} = pool;
        end

        % concatenate
        z = vl_nnconcat(pool_res, length(filter_size));

        % compute loss
        % o: value of output layer
        % y: ground truth label (1 or 2)
        if training_data{i, 3} == 1
            y = 1;
        else y = 2;
        end
        [z_dropout, mask] = vl_nndropout(z, 'rate', 0.5);
        z_reshape = reshape(z_dropout, length(filter_size) * n_filter, 1);
```

```
        o = vl_nnconv(z_reshape, reshape(W_out, length(filter_size) * n_filter
            , 1, 1, n_class), B_out);
        loss_t = loss_t + vl_nnloss(reshape(o, 1, 1, n_class, 1), y);

        % section 2.2 backward propagation and compute the derivatives
        dlossdo = vl_nnloss(reshape(o, 1, 1, n_class, 1), y, 1);
        [dlossdz, dlossdW_out, dlossdB_out] = ...
            vl_nnconv(z_reshape, reshape(W_out, length(filter_size) * n_filter
                , 1, 1, 2), B_out,dlossdo);

        dlossdpool_res = vl_nnconcat(pool_res, length(filter_size), ...
            reshape(dlossdz, 1, 1, length(filter_size) * n_filter));

        dlossdrelu = cell(1, length(filter_size));
        dlossdconv = cell(1, length(filter_size));
        dlossdX = cell(1, length(filter_size));
        dlossdW_conv = cell(1, length(filter_size));
        dlossdB_conv = cell(1, length(filter_size));

        for k = 1: length(filter_size)
            conv = cache{1, k};
            relu = cache{2, k};
            sizes = size(conv);
            dlossdrelu{k} = vl_nnpool(relu, [sizes(1), 1], dlossdpool_res{k});
            dlossdconv{k} = vl_nnrelu(conv, dlossdrelu{k});
            [dlossdX{k}, dlossdW_conv{k}, dlossdB_conv{k}] = ...
            vl_nnconv(X, W_conv{k}, B_conv{k},dlossdconv{k});
        end

        % section 2.3 update the parameters
        W_out = W_out - rate .* reshape(dlossdW_out, length(filter_size) *
            n_filter, 2);
        % set l2-norm constraint to 3
        if norm(W_out) > 3
            W_out = (W_out/norm(W_out, 2)*3);
        end
        B_out = B_out - rate .* dlossdB_out;
        for k = 1: length(filter_size)
            W_conv{k} = W_conv{k} - rate .* dlossdW_conv{k};
            B_conv{k} = B_conv{k} - rate .* dlossdB_conv{k};
            X = X - rate .* dlossdX{k};
        end
        for k = 1:length(sentence)
            index = word_indexs(k);
            T(index, :) = X(k,:);
        end
    end
```

```
% print loss for each epoch
fprintf('Epoch %d: loss: %.2d, ', t, loss_t);

loss{t} = loss_t;

% calculate accuracy on training data
error_train = 0;
for i = 1: length(training_data)
    sentence = training_data{i, 2};

    % get sentence matrix
    % words_indexs = [wordMap('i'), wordMap('like'),
    % ..., wordMap('!')]
    word_indexs = zeros(1, length(sentence));
    for j = 1: length(sentence)
        word_indexs(j) = wordMap(sentence{j});
    end
    X = T(word_indexs, :);

    % forward propagation and compute the loss
    pool_res = cell(1, length(filter_size));
    cache = cell(2, length(filter_size));
    for k = 1: length(filter_size)
        % convolutional operation
        conv = vl_nnconv(X, W_conv{k}, B_conv{k});

        % apply activation function: relu
        relu = vl_nnrelu(conv);

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

        % important: keep these values for back-prop
        cache{2, k} = relu;
        cache{1, k} = conv;
        pool_res{k} = pool;
    end

    % concatenate
    z = vl_nnconcat(pool_res, length(filter_size));

    % compute loss
    % o: value of output layer
    % y: ground truth label (1 or 2)
    z_reshape = reshape(z, 1, length(filter_size) * n_filter);
    o = reshape(z_reshape * W_out, 2, 1) + B_out;
    if o(1) > o(2)
```

```
        y = 1;
    else y = 0;
    end
    error_train = error_train + abs(y - training_data{i, 3});
end


% calculate accuracy on validation data
error_val = 0;
for i = 1: length(validation_data)
    sentence = validation_data{i, 2};
    % get sentence matrix
    % words_indexs = [wordMap('i'), wordMap('like'),
    % ..., wordMap('!')]
    word_indexs = zeros(1, length(sentence));
    for j = 1: length(sentence)
        word_indexs(j) = wordMap(sentence{j});
    end
    X = T(word_indexs, :);

    % forward propagation and compute the loss
    pool_res = cell(1, length(filter_size));
    cache = cell(2, length(filter_size));
    for k = 1: length(filter_size)
        % convolutional operation
        conv = vl_nnconv(X, W_conv{k}, B_conv{k});

        % apply activation function: relu
        relu = vl_nnrelu(conv);

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

        % important: keep these values for back-prop
        cache{2, k} = relu;
        cache{1, k} = conv;
        pool_res{k} = pool;
    end

    % concatenate
    z = vl_nnconcat(pool_res, length(filter_size));

    % compute loss
    % o: value of output layer
    % y: ground truth label (1 or 2)
    z_reshape = reshape(z, 1, length(filter_size) * n_filter);
    o = reshape(z_reshape * (W_out), 2, 1) + B_out;
```

```
            if o(1) > o(2)
                y = 1;
            else y = 0;
            end
            error_val = error_val + abs(y - validation_data{i, 3});
        end
        accuracy_train = 1 - error_train/length(training_data);
        accuracy_val = 1 - error_val/length(validation_data);
        fprintf('training accuracy: %f, validation accuracy: %f,\n',
           accuracy_train, accuracy_val);
    end
end
```

Appendix C: Matlab demo code

```
clear; clc;

for num_file = 1: 3
    filter_size = [3, 4, 5];
    n_filter = 6;

    % read in test data
    headLine = true;
    separater = '::';
    load('parameter-345-86.mat')
    data = cell(1000, 2);
    inputfile = strcat('test',num2str(num_file),'.txt');
    fid = fopen(inputfile, '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{ind, 1} = sid;
        data{ind, 2} = w;

        % read next line
        line = fgets(fid);
        ind = ind + 1;
    end
    fprintf('finish loading evaluation_set %d\n', num_file);
    fclose(fid);

    outputfile = strcat('submission',num2str(num_file),'.txt');
    fileID = fopen(outputfile,'w');
    fprintf(fid, '%s::%s\n', 'id','label');
    formatSpec = '%d::%d\n';
    for i = 1: length(data)
        sentence = data{i, 2};
        if length(sentence) < 5
```

```
    for k = length(sentence)+1: 5
        sentence{k} = '<PAD>';
    end
end
% get sentence matrix
% words_indexs = [wordMap('i'), wordMap('like'),
% ..., wordMap('!')]
word_indexs = zeros(1, length(sentence));
for j = 1: length(sentence)
    if (isKey(wordMap, sentence{j}) == 0)
        sentence{j} = '<UNK>';
    end
    word_indexs(j) = wordMap(sentence{j});
end
X = T(word_indexs, :);

% forward propagation and compute the loss
pool_res = cell(1, length(filter_size));
cache = cell(2, length(filter_size));
for k = 1: length(filter_size)
    % convolutional operation
    conv = vl_nnconv(X, W_conv{k}, B_conv{k});

    % apply activation function: relu
    relu = vl_nnrelu(conv);

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

    % important: keep these values for back-prop
    cache{2, k} = relu;
    cache{1, k} = conv;
    pool_res{k} = pool;
end

% concatenate
z = vl_nnconcat(pool_res, length(filter_size));

% compute loss
% o: value of output layer
% y: ground truth label (1 or 2)
z_reshape = reshape(z, 1, length(filter_size) * n_filter);
o = reshape(z_reshape * (W_out), 2, 1) + B_out;
if o(1) > o(2)
    y = 1;
else y = 0;
end
```

```
        %save prediction to output
        fprintf(fileID,formatSpec,i,y);
    end
    fclose(fileID);
end
```