Text Classification with CNN

TAs: Jiho Kim and Kijong Han

Advisor: Key-Sun Choi

Semantic Web Research Center

KAIST

Contents

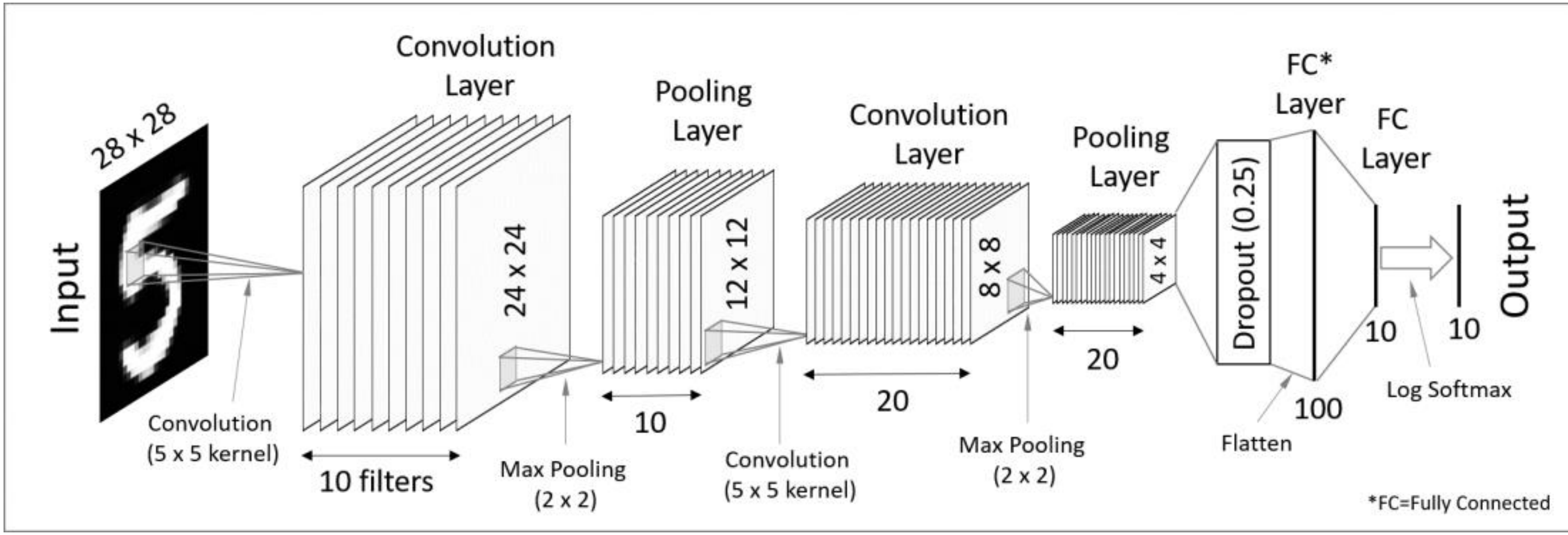
- Brief Introduction to Convolutional Neural Network(CNN)
- Implementing Text Classifciation with CNN using Tensorflow

Tensorflow Tutorial

- https://github.com/hunkim/DeepLearningZeroToAll

Introduction

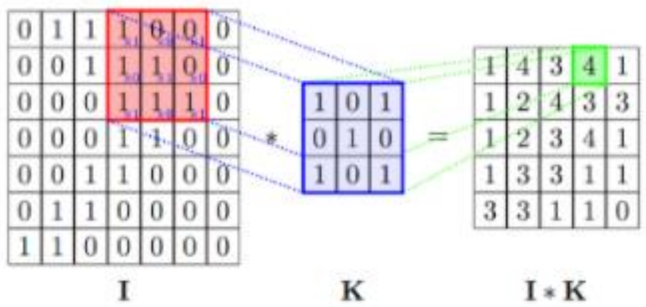
Convolutional Neural Networks



A Convolutional Neural Network (CNN) is comprised of one or more convolutional layers (often with a pooling step) and then followed by one or more fully connected layers as in a standard multilayer neural network.

The architecture of a CNN is designed to **take advantage of the 2D structure of an input such as image**

Convolution Layer



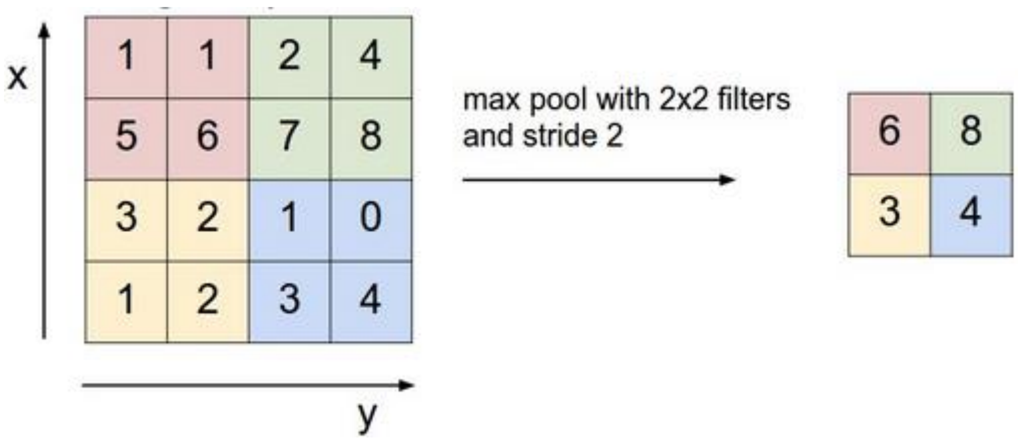
Overlaying the kernel(filter) on top of the input in all possible ways, and recording the **sum** of **elementwise products** between the input and the kernel(filter):

Convolution Filter act as retrieving specific features form local area of input. (such as contour for image)

Many Filters are used so that various type of feature can be extracted

Convolution Filters are learned automatically in network architecture of CNN, not defined by human.

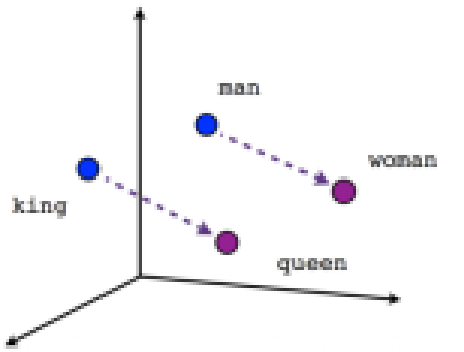
Pooling Layer



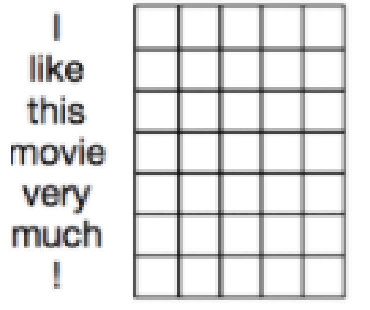
- Down Sampling
    - Reducing Operation time and memory
    - Could prevent overfitting
    - Types of Pooling
        - Max Pooling
            - Select only maximum Value
        - Average Pooling
            - …

Natural Language Text → 2D Matrix

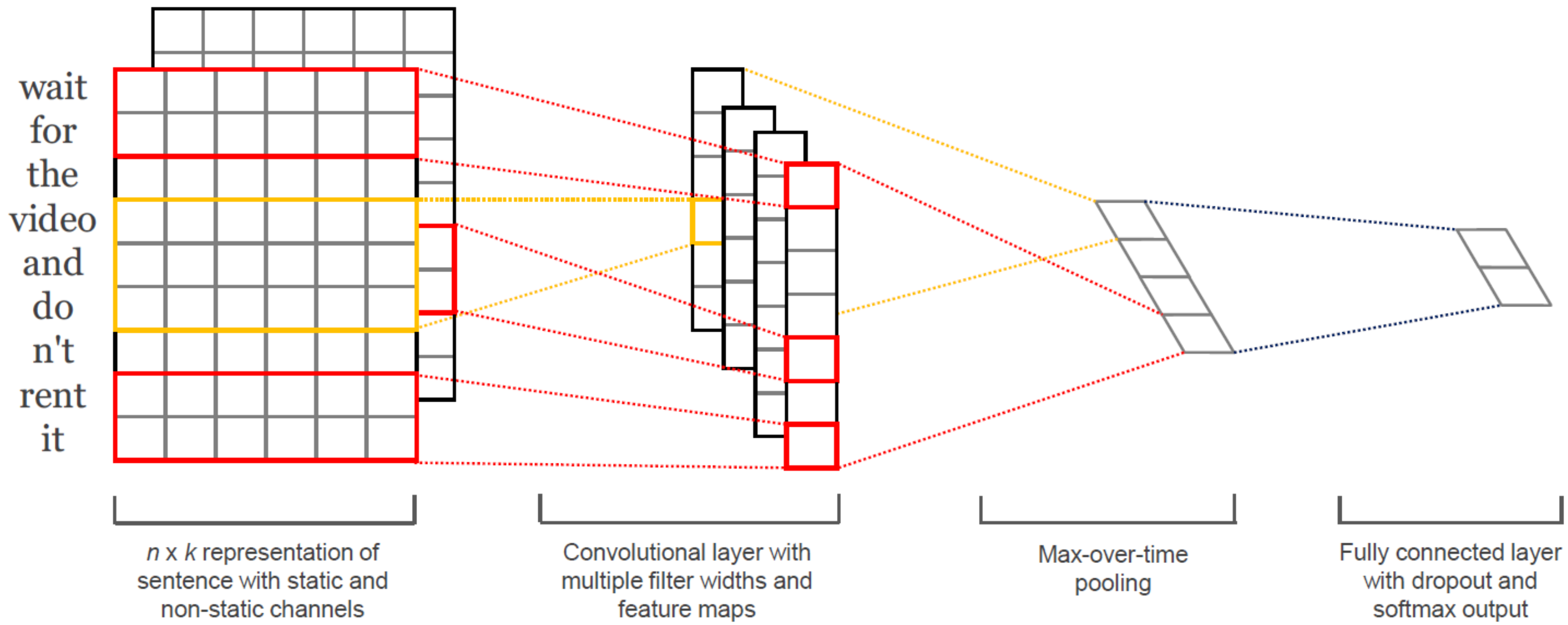
- Word can be represented as semantically-meaningful dense vectors.
  (word2vec, Glove, fasttext)



- Sentence or Document can be represented as matrix by concatenating word vectors.



Architecture



Convolution Filters retrieve features from phrase (N consecutive words, N-gram)

Use various size of filters(various N consecutive words)

Not linguistically or cognitively plausible but very fast and robust.

Yoon Kim. Convolutional Neural Networks for Sentence Classification. EMNLP(2014)

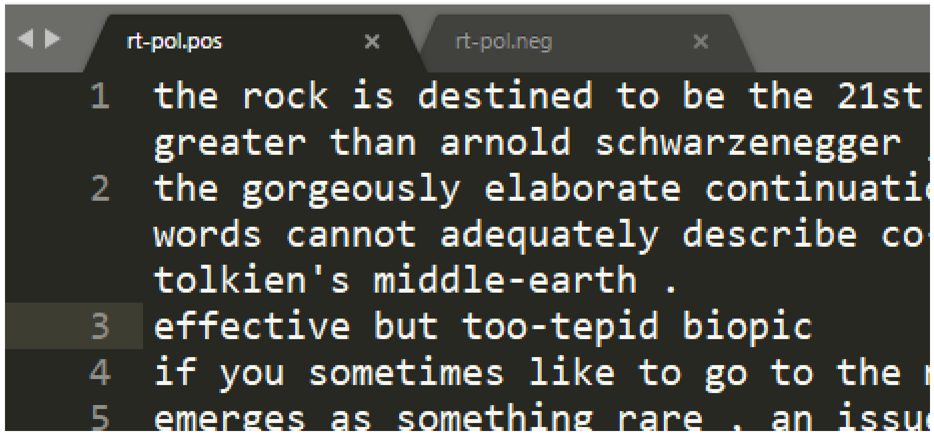
Goal

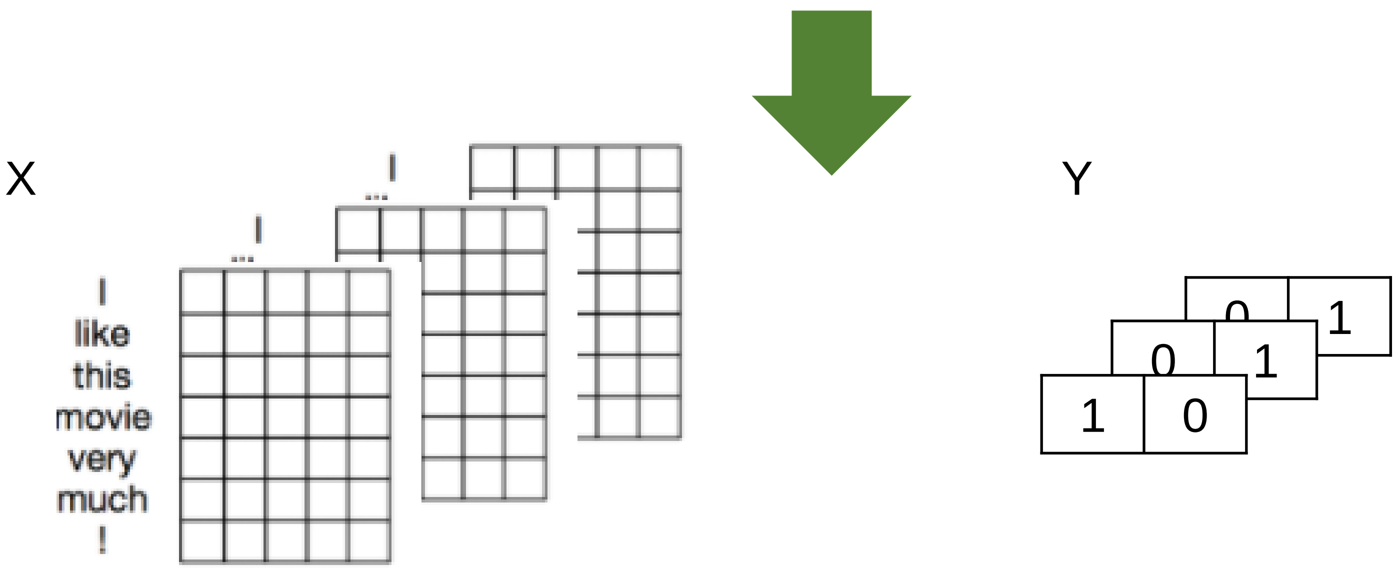
- Implement Movie Review Text Classification with CNN using Tensorflow.

- Dataset
  - MR movie review dataset.
  - 10,662 review text, 2classes(positive,negative)

Implementation

Part1. Prepare Data





Load libraries

```python
import tensorflow as tf
import numpy as np
from gensim.models.word2vec import Word2Vec
```

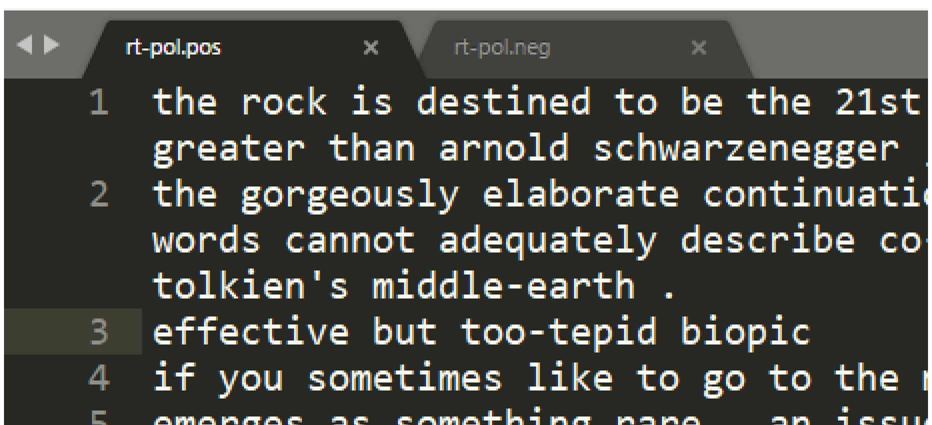
Load data from files

```
print("Loading data...")
# Load data from files
positive_examples = list(open("./data/rt-pol.pos", "r", encoding='utf-8').readlines())
positive_examples = [s.strip() for s in positive_examples]
negative_examples = list(open("./data/rt-pol.neg", "r", encoding='utf-8').readlines())
negative_examples = [s.strip() for s in negative_examples]
x_text = positive_examples + negative_examples
```

x_text = [“the rock is destined to be the …”, “the gorgeously … ”, …. “..”]

Movie review data file (rt-pol.pos, rt-pol.neg)

- One review sentence per one line.



Generate Labels

```
# Generate labels
positive_labels = [[0, 1] for _ in positive_examples]
negative_labels = [[1, 0] for _ in negative_examples]
y = np.concatenate([positive_labels, negative_labels], 0)
```

y = [[0,1],[0,1], …… [1,0], [1,0]]

list of 2-dimensional vector represent class(pos,neg) of each data.

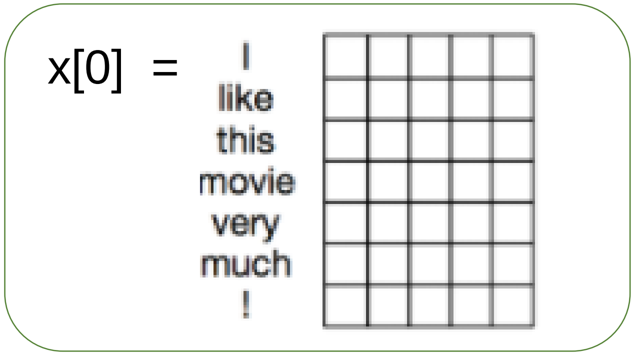
Load pre-trained word embedding

```python
# Load pre-trained word embedding
w2vec_model =  Word2Vec.load('./data/model-brown-vectors.bin')
embedding_size = w2vec_model.vector_size
```

$$\text{w2vec\_model[‘apple’]} = [\overbrace{-0.502,\ 0.236,\ \ldots\ ,\ -0.268,\ 0.463}^{\text{embedding\_size}}]$$

Text → matrix

```python
# Convert text to matrix consist of word embedding
sequence_length = max([len(x.split(" ")) for x in x_text])
x = np.zeros((len(x_text), sequence_length, embedding_size), dtype=float)
for i,xi in enumerate(x_text):
    tokens = xi.split(' ')
    for j,token in enumerate(tokens):
        if token in w2vec_model:
            x[i, j] = w2vec_model[token]
```



- Set unknown word as zero vectors

- x.shape = (data_size, sequence_length, embedding_size)

Number of data.

Length of word vectors

Max length(# of words) of sentence

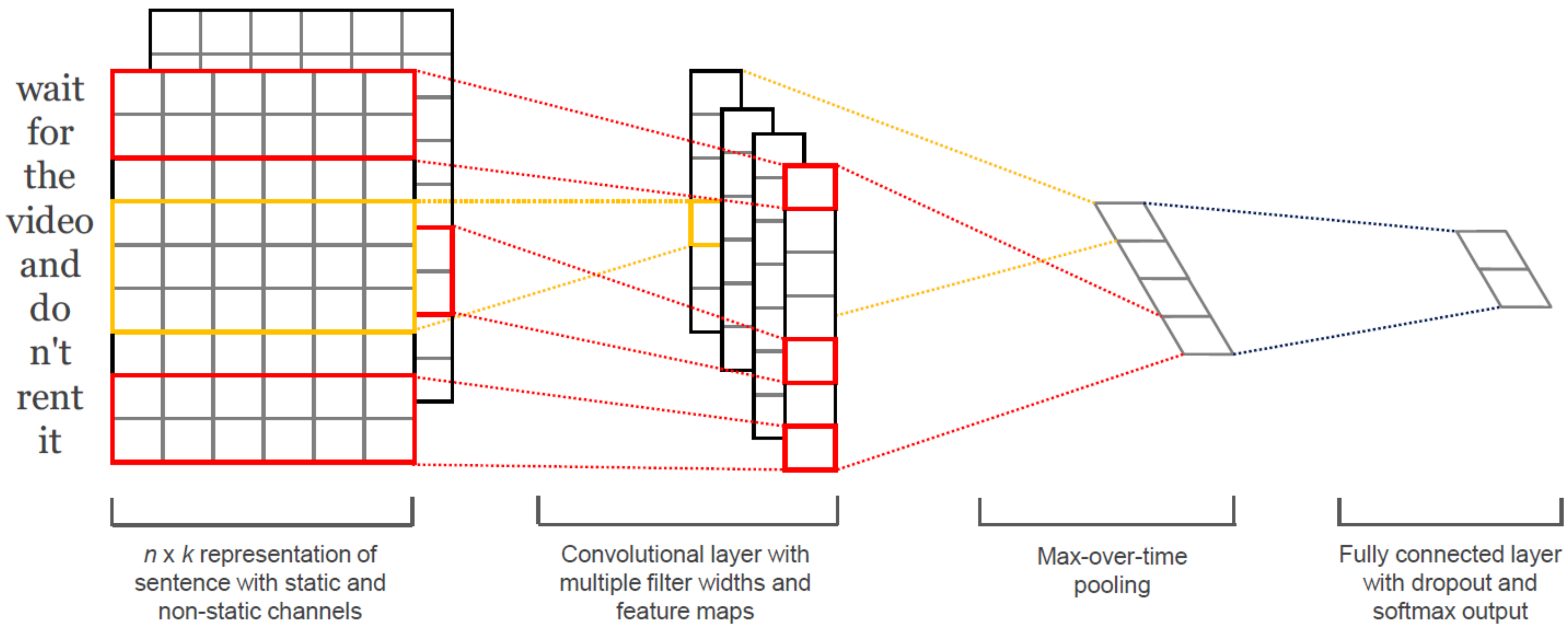
Shuffle and Split Data

```
# Randomly shuffle data
np.random.seed(10)
shuffle_indices = np.random.permutation(np.arange(len(y)))
x_shuffled = x[shuffle_indices]
y_shuffled = y[shuffle_indices]

# Split train/test set
test_sample_index = -1 * int(0.1 * float(len(y)))
x_train, x_test = x_shuffled[:test_sample_index], x_shuffled[test_sample_index:]
y_train, y_test = y_shuffled[:test_sample_index], y_shuffled[test_sample_index:]
```

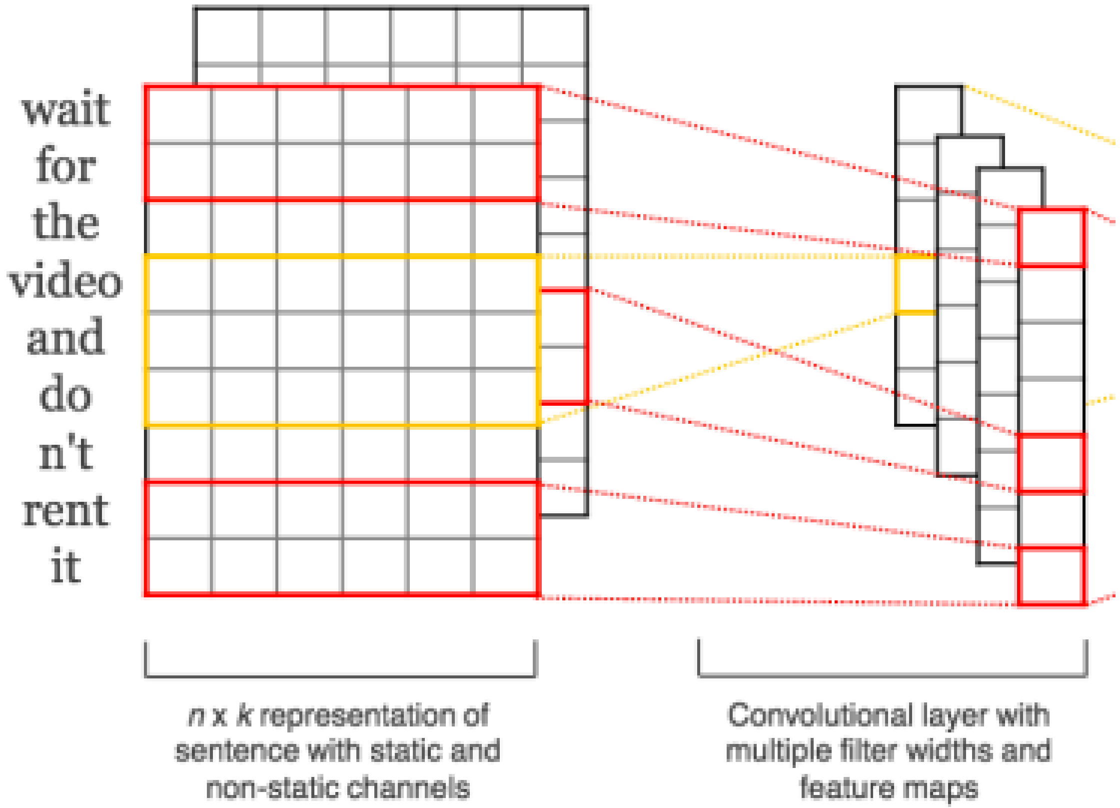
- Randomly shuffle data
- Split data into Train:Test = 90% : 10%

Part2. Build Network



Yoon Kim. Convolutional Neural Networks for Sentence Classification. EMNLP(2014)

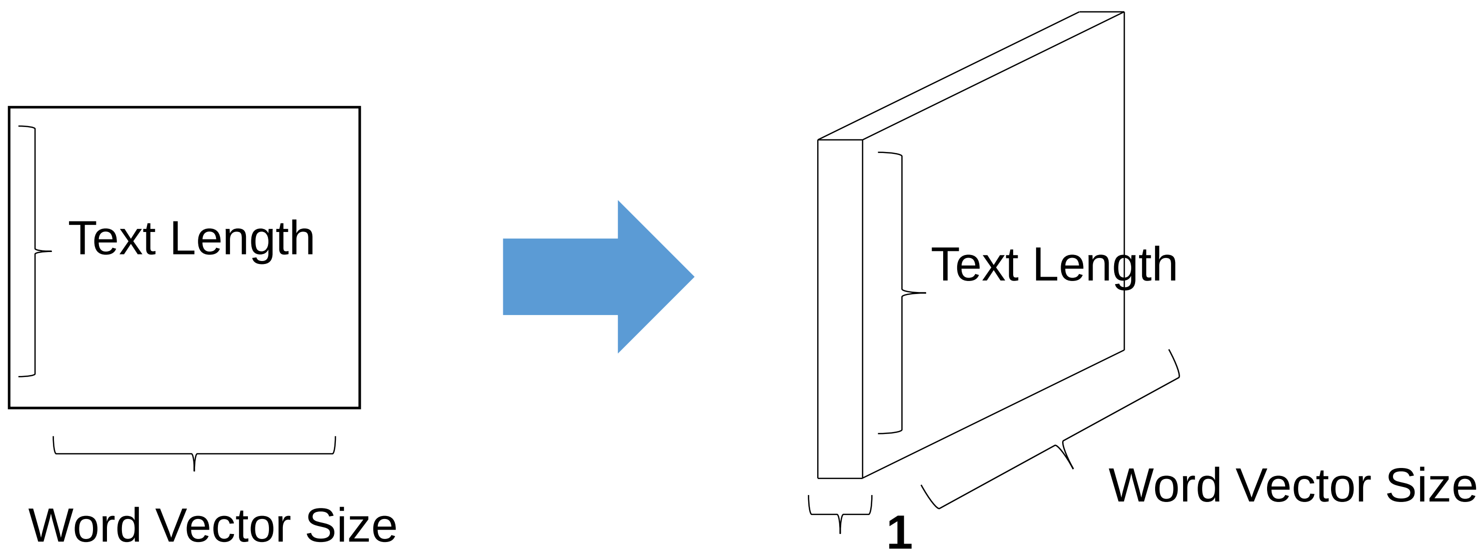
Convolution Layer



*2D Data -> 3D Data

- 1 Text : [Text Length * Word Vector Size] → 2D
- 1 Image : [Height * Width * Channel] → 3D   (Channel : RGB, etc..)

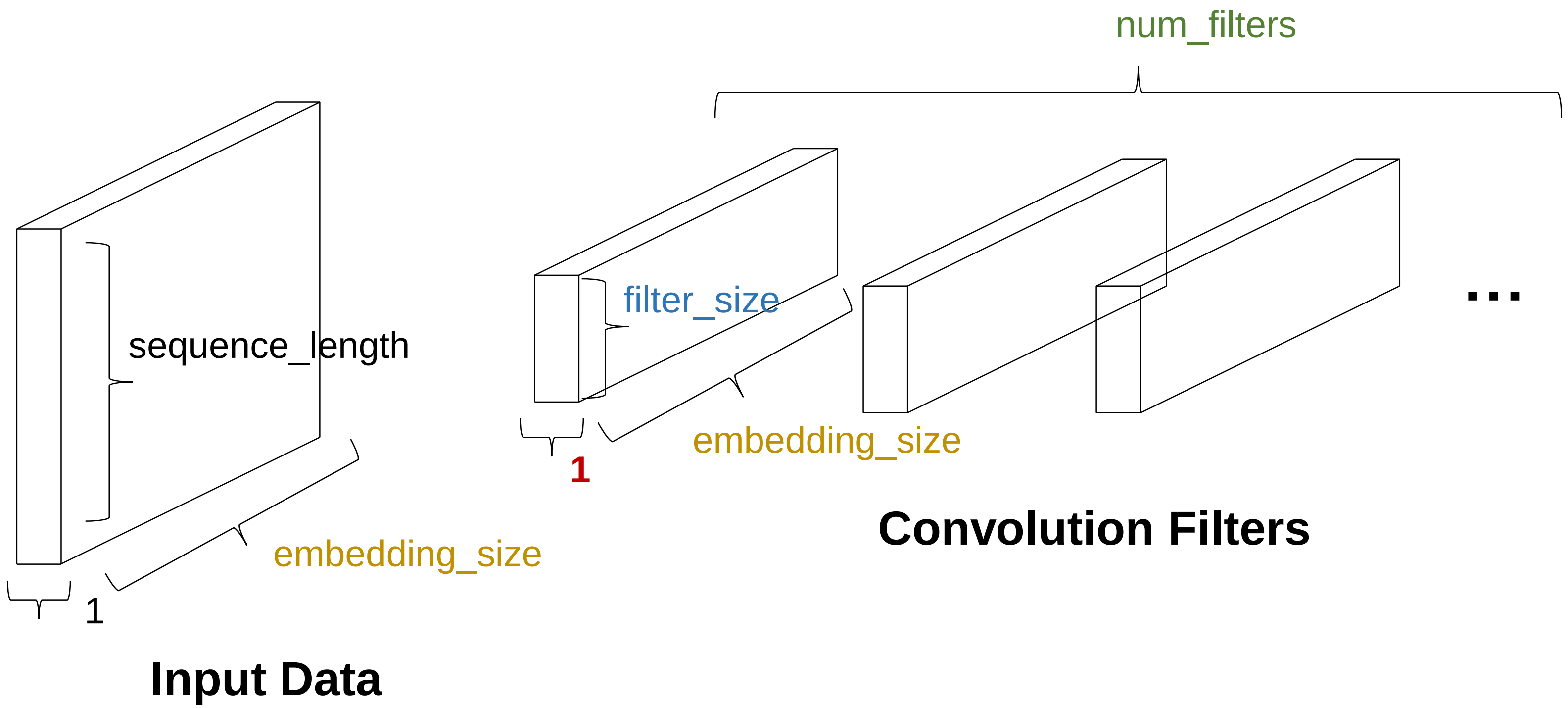
How to change text data into 3D?
→ Just expand dimension of data to have 1 channel like black and white image




```
extended_input_x = tf.expand_dims(input_x, -1)
```

Convolution Layer

```
filter_shape = [filter_size, embedding_size, 1, num_filters]
W_c = tf.Variable(tf.truncated_normal(filter_shape, stddev=0.1), name="W_c")
conv = tf.nn.conv2d(
    extended_input_x, W_c, strides=[1, 1, 1, 1], padding="VALID", name="conv")
```



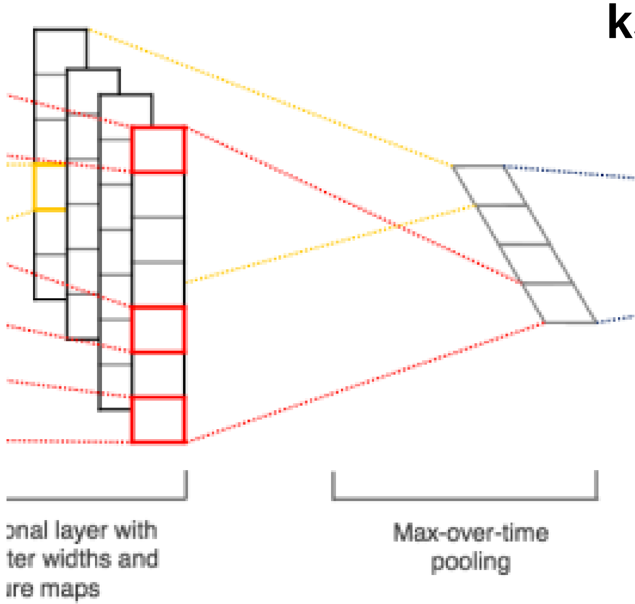
Max Pooling Layer

```
    pooled = tf.nn.max_pool(
        h, ksize=[1, sequence_length - filter_size + 1, 1, 1],
        strides=[1, 1, 1, 1],
        padding='VALID',
        name="pool")
```

data, height, width, channel

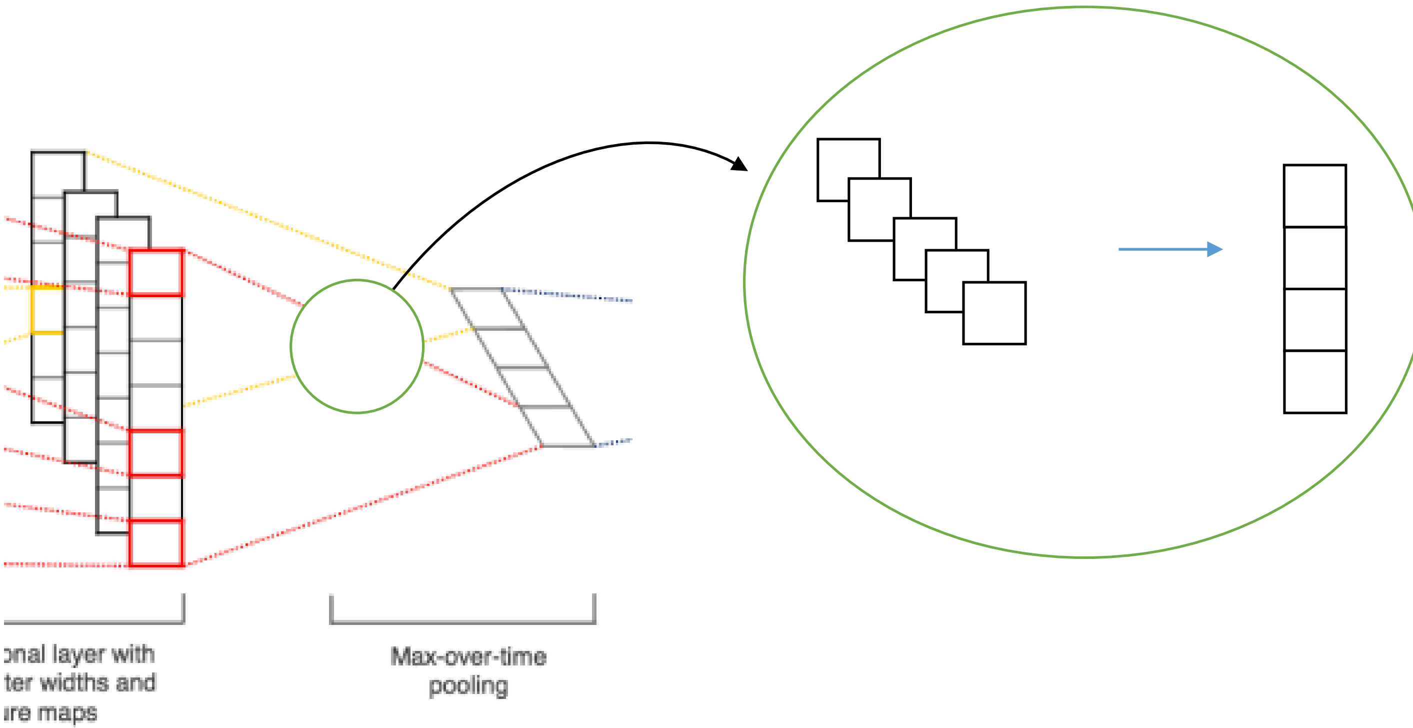
ksize = Units of max pooling area

sequence_length
– filter_size + 1



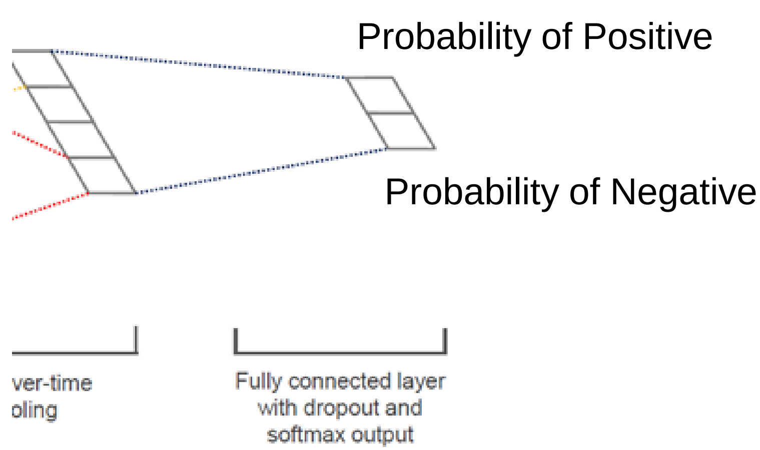
Flatten and Dropout

```python
# Flatten and Dropout
num_filters_total = num_filters * len(filter_sizes)
h_pool = tf.concat(pooled_outputs, 3)
h_pool_flat = tf.reshape(h_pool, [-1, num_filters_total])
h_drop = tf.nn.dropout(h_pool_flat, dropout_keep_prob)
```



Fully Connected Layer with Softmax Output

```python
# Final Fully Connected Layer
W_f = tf.get_variable(
                "W_f",
                shape=[num_filters_total, num_classes],
                initializer=tf.contrib.layers.xavier_initializer())
b_f = tf.Variable(tf.constant(0.1, shape=[num_classes]), name="b_f")
scores = tf.nn.xw_plus_b(h_drop, W_f, b_f, name="scores")
predictions = tf.argmax(scores, 1, name="predictions")
```



Loss, Accuracy, Optimizer

```
#loss
loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=scores, labels=input_y))

#accuracy
correct_predictions = tf.equal(predictions, tf.argmax(input_y, 1))
accuracy = tf.reduce_mean(tf.cast(correct_predictions, "float"))

#optimizer
optimizer = tf.train.AdamOptimizer(1e-3).minimize(loss)
```

Output of ‘correct_predictions’ node

[True,True,False, …., False, True]

True for right prediction, False for wrong prediction.