# 텐서플로와 머신러닝으로 시작하는 자연어처리

Lecture 3
2019. 06. 15

# Text Classification

# Text Classification

- What is Text Classification?
- Why do we need Text Classification?
- Text Classification Dataset
- How do we evaluate Text Classification Model?
- How does Text Classification work?
- Classification Papers
    - Convolutional neural networks for sentence classification
    - Character-level Convolutional Networks for Text Classification

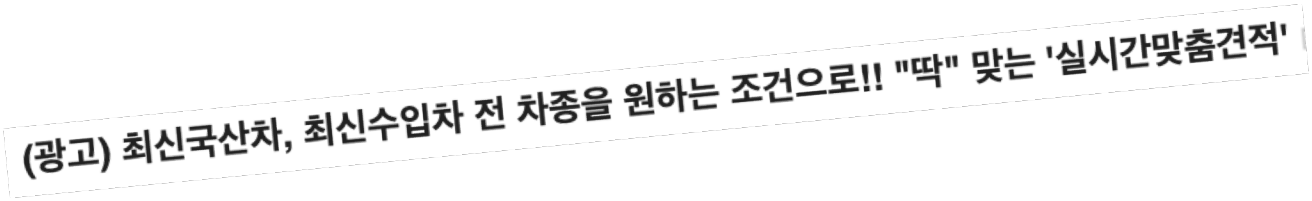
# What is Text Classification

# Text Classification

- 다양한 Natural Language Processing Task 중 가장 기본이 되는 Task
- 주어진 Source Text를 Pre-Defined된 Class들로 분류함.

# Text Classification

- 다양한 Natural Language Processing Task 중 가장 기본이 되는 Task
- 주어진 Source Text를 Pre-Defined된 Class들로 분류함.



# Text Classification

- 다양한 Natural Language Processing Task 중 가장 기본이 되는 Task
- 주어진 Source Text를 Pre-Defined된 Class들로 분류함.

# Text Classification

- 다양한 Natural Language Processing Task 중 가장 기본이 되는 Task
- 주어진 Source Text를 **Pre-Defined된 Class**들로 분류함.



  - **2** Class Classification: Binary Classification
  - **N** Class Classification: Multiclass Classification
    or Multilabel Classification

# Why do we need Text Classification

# Why do we need Text Classification

## Spam Filtering

주어진 Text를 Spam인지 아닌지 분류(Binary Classification)

# Why do we need Text Classification

## Spam Filtering

주어진 Text를 Spam인지 아닌지 분류(Binary Classification)

Input Text

- (광고)★월간 십일절★ 장바구니 7천원 쿠폰 + T멤버십~22%혜택
- 시스템옴므, 왕겜, 지오다노, 앤드지, 헨리코튼, 닥터자르트, 직구
- 안녕하세요 미팅 관련 연락드립니다!

# Why do we need Text Classification

## Spam Filtering

주어진 Text를 Spam인지 아닌지 분류(Binary Classification)

Input Text

- (광고)★월간 십일절★ 장바구니 7천원 쿠폰 + T멤버십~22%혜택 **=> Spam**
- 시스템옴므, 왕겜, 지오다노, 앤드지, 헨리코튼, 닥터자르트, 직구 **=> Spam**
- 안녕하세요 미팅 관련 연락드립니다! **=> Not**

# Why do we need Text Classification

## Sentiment Analysis

주어진 Text를 주어진 감정을 분류 (Binary or Multiclass Classification)

# Why do we need Text Classification

## Sentiment Analysis

주어진 Text를 주어진 감정을 분류 (Binary or Multiclass Classification)

긍정 or 부정

# Why do we need Text Classification

## Sentiment Analysis

주어진 Text를 주어진 감정을 분류 (Binary or Multiclass Classification)

- 긍정, 중립, 부정
- 아주 나쁨, 나쁨, 보통, 좋음, 아주 좋음

# Why do we need Text Classification

## Sentiment Analysis

주어진 Text를 주어진 감정을 분류 (Binary or Multiclass Classification)

Input Text

- 소재는 좋지만 감독의 연출이 말아먹은 좋은예 ⇒ 부정
- 퇴근후 영화보면서 야식에 혼술하는거 완전 세상행복한건데 왜케 불행하게 묘사해놨냐 ⇒ 부정
- 눈물날뻔했다.. 마블의 끝이며 시작 ⇒ 긍정

# Why do we need Text Classification

## Sentiment Analysis

주어진 Text를 주어진 감정을 분류 (Binary or Multiclass Classification)

https://app.monkeylearn.com/main/classifiers/cl_pi3C7JiL/

# Why do we need Text Classification

## Language Detection

주어진 Text의 Language를 Classification

https://app.monkeylearn.com/main/classifiers/cl_Vay9jh28/

# Why do we need Text Classification

## Application

- SNS Monitoring
- Brand Monitoring
- Intent Classification in Dialogue System
- Voice of Customer Classification

# Text Classification Dataset

# Text Classification Dataset

1. Sentiment Analysis
    - IMDb Dataset
    - SST-2 Dataset
    - SST-5 Dataset
    - Yelp Review Dataset
2. Text Classification
    - AG News Corpus
    - DBpedia Ontology Dataset

# Text Classification Dataset

1. Sentiment Analysis
    - IMDb Dataset
    - SST-2 Dataset
    - SST-5 Dataset
    - Yelp Review Dataset
2. Text Classification
    - AG News Corpus
    - DBpedia Ontology Dataset

**IMDb Dataset**

- Binary Classification Dataset(긍정 or 부정)
- Internet Movie Database (IMDb) 사이트의 Review들을 평점에 따라 Labeling함
    - 7점 이상 => 긍정
    - 4점 이하 => 부정
- 전체 50,000개

# Text Classification Dataset

1. Sentiment Analysis
    - IMDb Dataset
    - SST-2 Dataset
    - SST-1 Dataset
    - Yelp Review Dataset
2. Text Classification
    - AG News Corpus
    - DBpedia Ontology Dataset

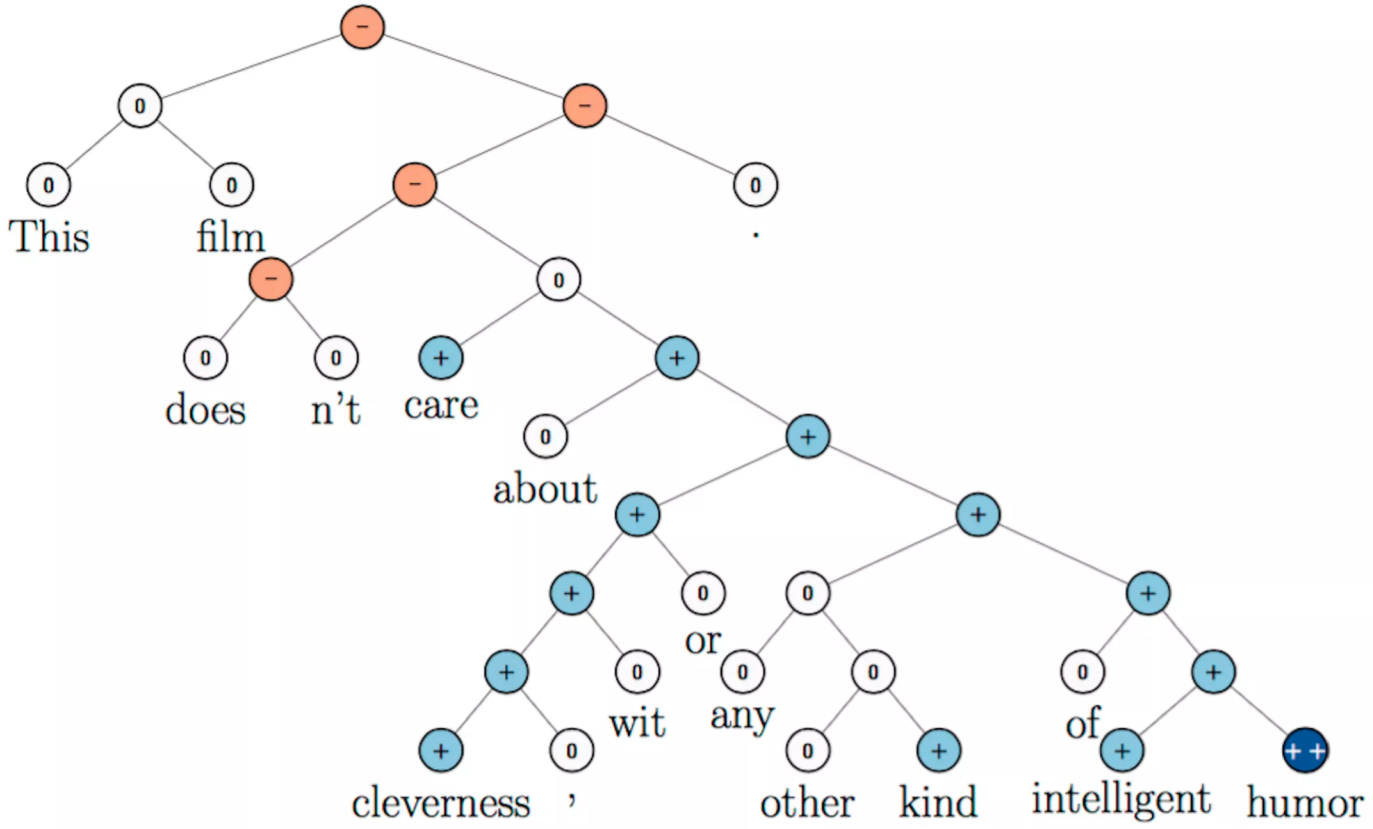
**SST Dataset**

- Stanford Sentiment Treebank
- Sentence가 Tree구조로 되어있는 Dataset
- 영화 리뷰 text로 구성
- SST-2
    - Binary Class (긍정, 부정)
    - 56,400개
- SST-1
    - Multi Class (5 단계)
    - 94,200개

# Text Classification Dataset

1. Sentiment Analysis
   - IMDb Dataset
   - SST-2 Dataset
   - SST-5 Dataset
   - Yelp Review Dataset
2. Text Classification
   - AG News Corpus
   - DBpedia Ontology Dataset

**SST Dataset**



# Text Classification Dataset

1. Sentiment Analysis
    - IMDb Dataset
    - SST-2 Dataset
    - SST-5 Dataset
    - Yelp Review Dataset
2. Text Classification
    - AG News Corpus
    - DBpedia Ontology Dataset

**Yelp Review Dataset**

- 크라우드 소싱 리뷰 포럼인 Yelp의 리뷰로 구성
- 전체 500,000개
- SST Dataset과 마찬가지로 2가지 종류로 라벨링
    - Binary Class
    - 5 Class

# Text Classification Dataset

1. Sentiment Analysis
    - IMDb Dataset
    - SST-2 Dataset
    - SST-5 Dataset
    - Yelp Review Dataset
2. Text Classification
    - AG News Corpus
    - DBpedia Ontology Dataset

**AG News Corpus**

- AG News Corpus로 구축된 데이터셋
- 30,000개의 Training Data
- Class 당 1,900개의 Test Data
- 총 4개의 Class
    - World
    - Sports
    - Business
    - Sci/Tech

# Text Classification Dataset

1. Sentiment Analysis
   - IMDb Dataset
   - SST-2 Dataset
   - SST-5 Dataset
   - Yelp Review Dataset
2. Text Classification
   - AG News Corpus
   - DBpedia Ontology Dataset

**DBpedia Ontology Dataset**

- DBpedia의 title과 content로 구성
- 전체 14개의 Class
  - e.x) Company, Educational, …
- 560,000개의 Training Data
- 각 Class마다 70,000개의 Test Data

# How do we evaluate Text Classification Model

# How do we evaluate Text Classification Model

**Error**

실제 값과는 다르게 예측을 한 경우

# How do we evaluate Text Classification Model

**Error**

실제 값과는 다르게 예측을 한 경우

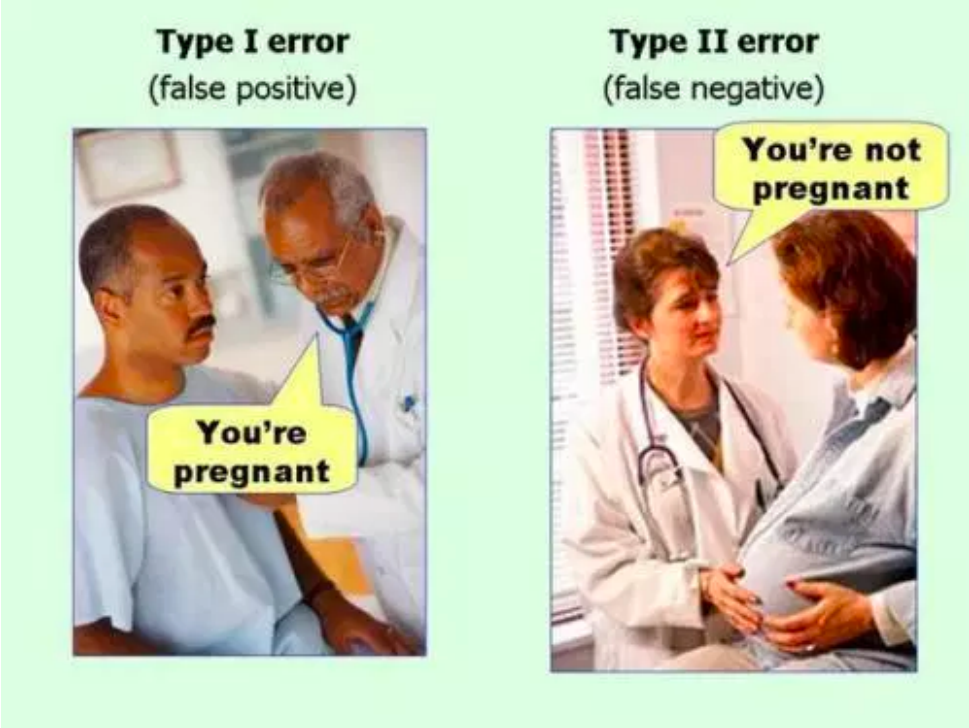
1. 긍정을 부정으로 예측한 경우
2. 부정을 긍정으로 예측한 경우

# How do we evaluate Text Classification Model

**Error**

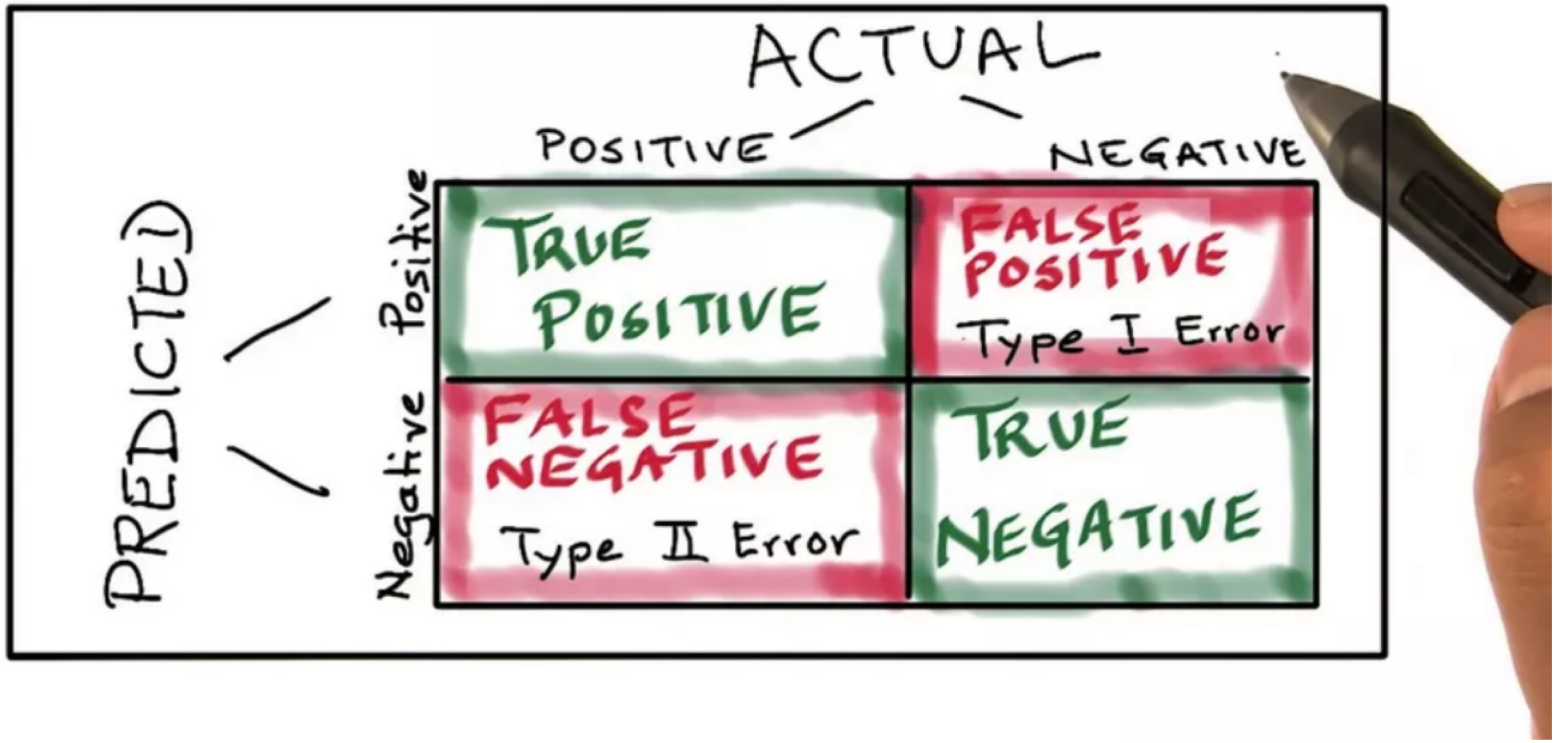
실제 값과는 다르게 예측을 한 경우

1. 긍정을 부정으로 예측한 경우
2. 부정을 긍정으로 예측한 경우



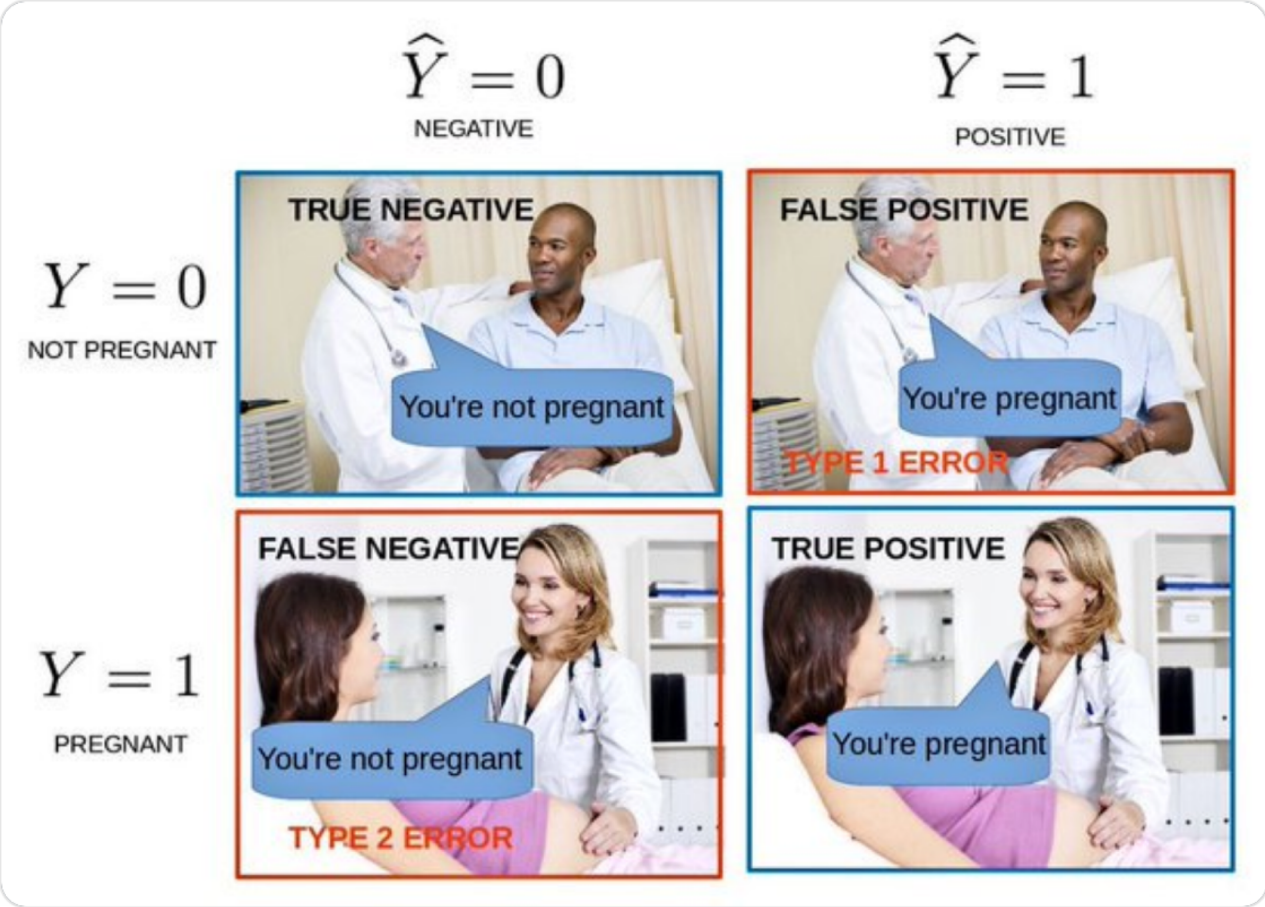
# How do we evaluate Text Classification Model

## Confusion Matrix



# How do we evaluate Text Classification Model

## Confusion Matrix



# Classification Metric

1. Accuracy

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

$$= \frac{T}{T + N}$$

전체적인 Classifier의 성능

| | | Actual: Positive | Actual: Negative |
|---|---|---|---|
| Predicted | Positive | TP | FP |
| Predicted | Negative | FN | TN |

# Classification Metric

2. Error Rate

$$Error\ Rate = \frac{FN + FP}{TP + TN + FP + FN}$$

$$= \frac{F}{T + N}$$

| | | Actual: Positive | Actual: Negative |
|---|---|---|---|
| Predicted | Positive | TP | FP |
| Predicted | Negative | FN | TN |

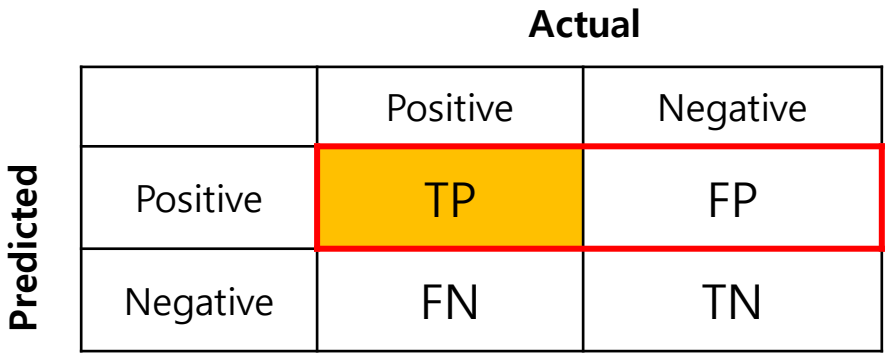
Classification error

# Classification Metric

3. Precision

$$Precision = \frac{TP}{TP + FP}$$

Classifier가 Positive로 예측한 값 중 실제 Positive 비율



# Classification Metric

4. Sensitivity (or Recall or True Positive Rate, TPR)

$$Recall = \frac{TP}{TP + FN}$$

실제 Positive인 값을 얼마나 잘 분류하는지 측정하는 척도

| | | Actual: Positive | Actual: Negative |
|---|---|---|---|
| Predicted | Positive | TP | FP |
| | Negative | FN | TN |

# Classification Metric

5. Specificity (or True Negative Rate, TNR)

$$Specificity = \frac{TN}{TN + FP}$$

실제 Negative인 값을 얼마나 잘 분류하는지 측정하는 척도

| | | Actual: Positive | Actual: Negative |
|---|---|---|---|
| Predicted | Positive | TP | FP |
| Predicted | Negative | FN | TN |

# Classification Metric

6. F1 Score

$$F_1 = \left( \frac{recall^{-1} + precision^{-1}}{2} \right)^{-1} = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

Recall과 Precision의 조화 평균(Harmonic mean)

(조화 평균: 역수의 평균의 역수)

# Classification Metric

6. F1 Score

$$F_1 = \left( \frac{recall^{-1} + precision^{-1}}{2} \right)^{-1} = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

Recall과 Precision의 조화 평균(Harmonic mean) → Imbalanced Data 성능 측정 시 유용

(조화 평균: 역수의 평균의 역수)

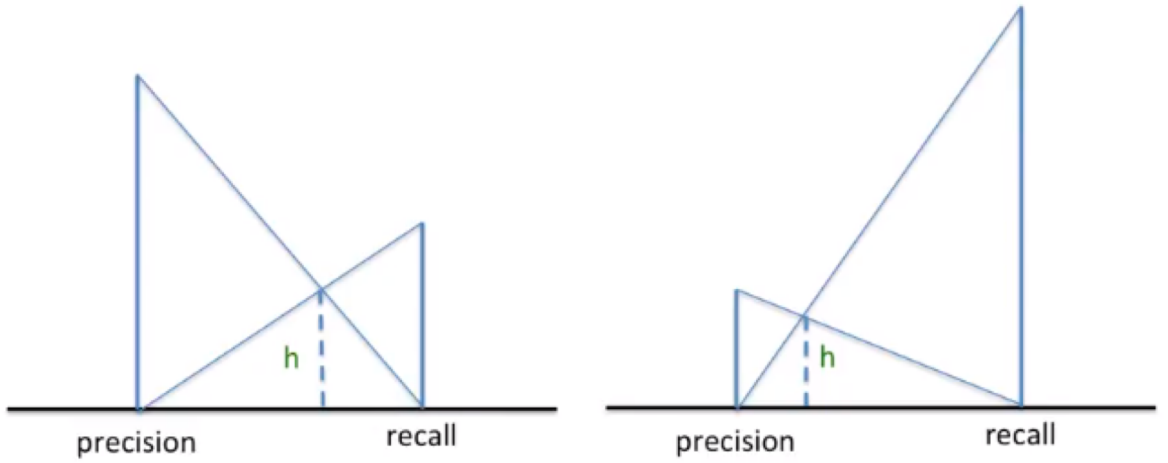
# Classification Metric

## 6. F1 Score

$$F_1 = \left( \frac{recall^{-1} + precision^{-1}}{2} \right)^{-1} = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

Recall과 Precision의 조화 평균(Harmonic mean) → Imbalanced Data 성능 측정 시 유용

(조화 평균: 역수의 평균의 역수)



# Classification Metric

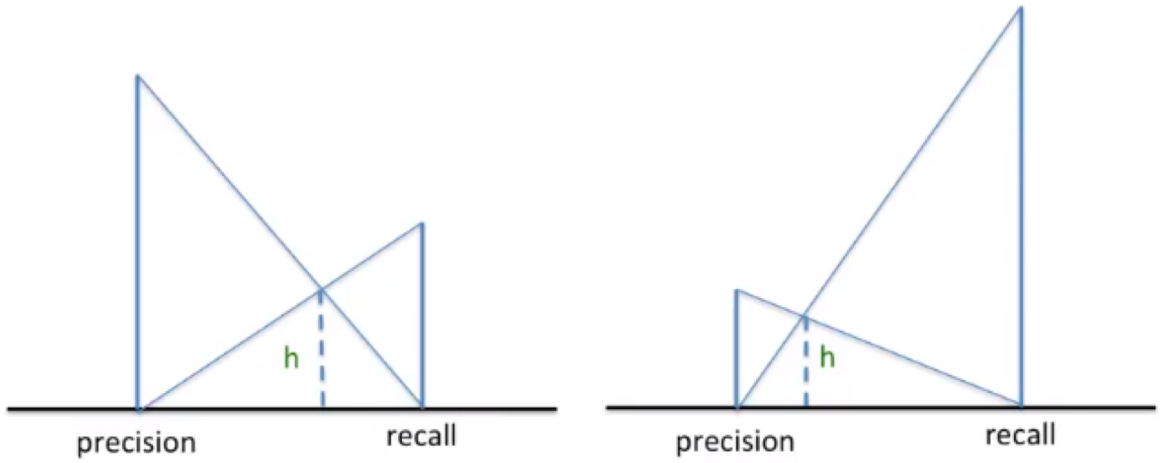
6. F1 Score

$$F_1 = \left( \frac{recall^{-1} + precision^{-1}}{2} \right)^{-1} = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

Recall과 Precision의 조화 평균(Harmonic mean) → Imbalanced Data 성능 측정 시 유용

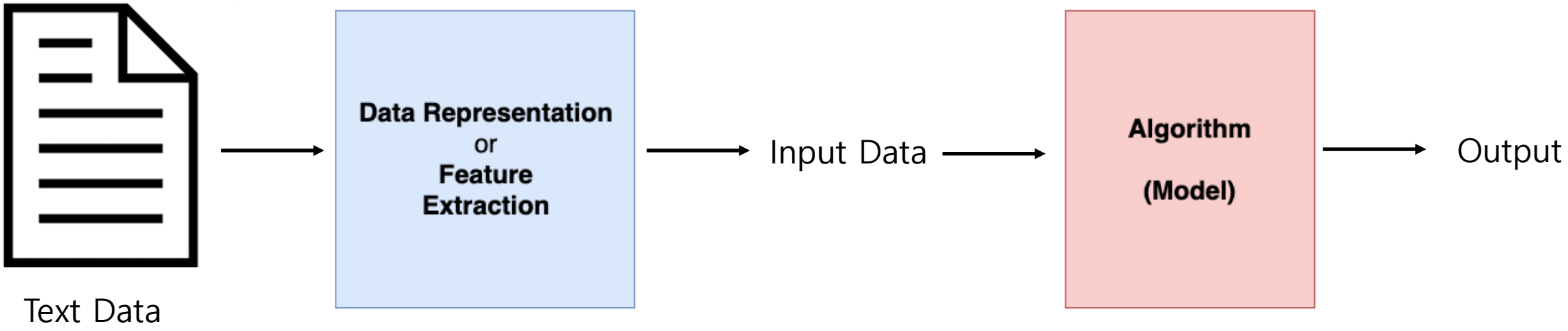
(조화 평균: 역수의 평균의 역수)

큰 값에 Penalty

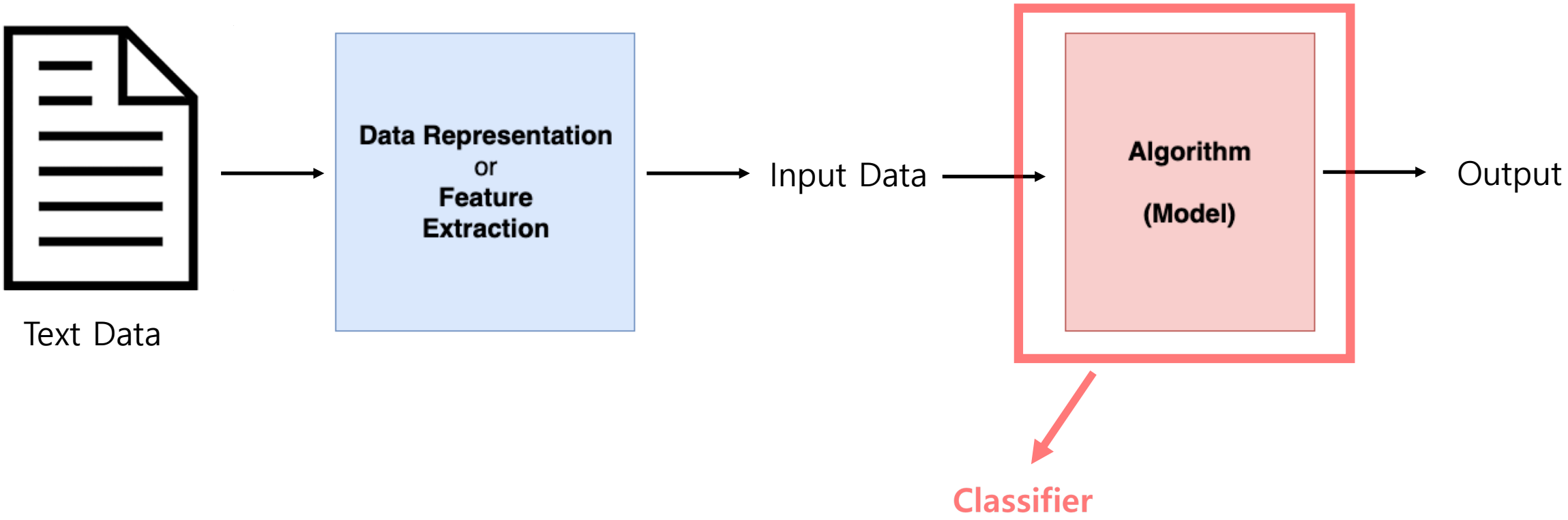


# How does Classification work

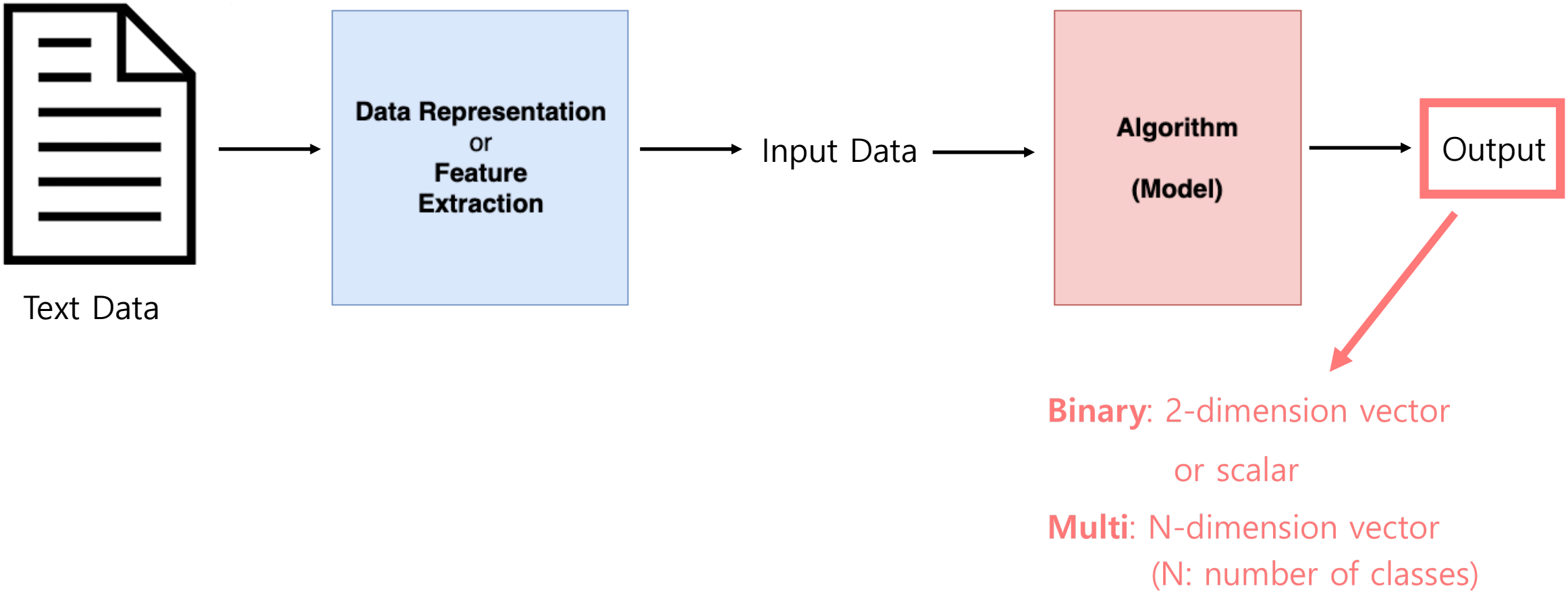
# How does Classification work



# How does Classification work



# How does Classification work



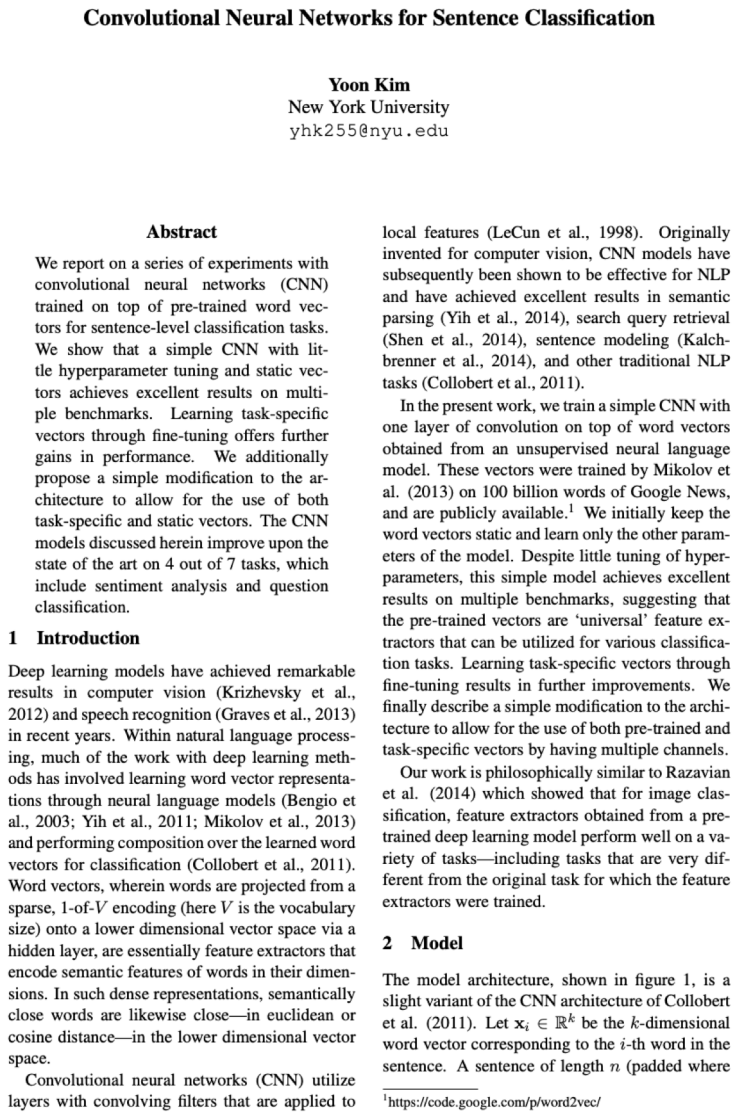
# Classification Papers

# Classification Papers

1. Convolutional neural networks for sentence classification
2. Character-level Convolutional Networks for Text Classification

# Convolutional neural networks for sentence classification

- Yoon kim(2014)
- 매우 간단한 모델임에도 괜찮은 성능을 보임
- Embedding vector사용에 대해 여러가지 실험을 진행
- Classification Task의 총 6개 Dataset에 대해서 진행



**Convolutional Neural Networks for Sentence Classification**

**Yoon Kim**
New York University
yhk255@nyu.edu

**Abstract**

We report on a series of experiments with convolutional neural networks (CNN) trained on top of pre-trained word vectors for sentence-level classification tasks. We show that a simple CNN with little hyperparameter tuning and static vectors achieves excellent results on multiple benchmarks. Learning task-specific vectors through fine-tuning offers further gains in performance. We additionally propose a simple modification to the architecture to allow for the use of both task-specific and static vectors. The CNN models discussed herein improve upon the state of the art on 4 out of 7 tasks, which include sentiment analysis and question classification.

**1 Introduction**

Deep learning models have achieved remarkable results in computer vision (Krizhevsky et al., 2012) and speech recognition (Graves et al., 2013) in recent years. Within natural language processing, much of the work with deep learning methods has involved learning word vector representations through neural language models (Bengio et al., 2003; Yih et al., 2011; Mikolov et al., 2013) and performing composition over the learned word vectors for classification (Collobert et al., 2011). Word vectors, wherein words are projected from a sparse, 1-of-$V$ encoding (here $V$ is the vocabulary size) onto a lower dimensional vector space via a hidden layer, are essentially feature extractors that encode semantic features of words in their dimensions. In such dense representations, semantically close words are likewise close—in euclidean or cosine distance—in the lower dimensional vector space.

Convolutional neural networks (CNN) utilize layers with convolving filters that are applied to local features (LeCun et al., 1998). Originally invented for computer vision, CNN models have subsequently been shown to be effective for NLP and have achieved excellent results in semantic parsing (Yih et al., 2014), search query retrieval (Shen et al., 2014), sentence modeling (Kalchbrenner et al., 2014), and other traditional NLP tasks (Collobert et al., 2011).

In the present work, we train a simple CNN with one layer of convolution on top of word vectors obtained from an unsupervised neural language model. These vectors were trained by Mikolov et al. (2013) on 100 billion words of Google News, and are publicly available.[1] We initially keep the word vectors static and learn only the other parameters of the model. Despite little tuning of hyperparameters, this simple model achieves excellent results on multiple benchmarks, suggesting that the pre-trained vectors are 'universal' feature extractors that can be utilized for various classification tasks. Learning task-specific vectors through fine-tuning results in further improvements. We finally describe a simple modification to the architecture to allow for the use of both pre-trained and task-specific vectors by having multiple channels.

Our work is philosophically similar to Razavian et al. (2014) which showed that for image classification, feature extractors obtained from a pre-trained deep learning model perform well on a variety of tasks—including tasks that are very different from the original task for which the feature extractors were trained.

**2 Model**

The model architecture, shown in figure 1, is a slight variant of the CNN architecture of Collobert et al. (2011). Let $\mathbf{x}_i \in \mathbb{R}^k$ be the $k$-dimensional word vector corresponding to the $i$-th word in the sentence. A sentence of length $n$ (padded where

[1] https://code.google.com/p/word2vec/

# Convolutional neural networks for sentence classification

## Model Architecture

# Convolutional neural networks for sentence classification

## Model Architecture

### 2 Model

The model architecture, shown in figure 1, is a slight variant of the CNN architecture of Collobert et al. (2011). Let $\mathbf{x}_i \in \mathbb{R}^k$ be the $k$-dimensional word vector corresponding to the $i$-th word in the sentence. A sentence of length $n$ (padded where necessary) is represented as

$$\mathbf{x}_{1:n} = \mathbf{x}_1 \oplus \mathbf{x}_2 \oplus \ldots \oplus \mathbf{x}_n, \quad (1)$$

where $\oplus$ is the concatenation operator. In general, let $\mathbf{x}_{i:i+j}$ refer to the concatenation of words $\mathbf{x}_i, \mathbf{x}_{i+1}, \ldots, \mathbf{x}_{i+j}$. A convolution operation involves a *filter* $\mathbf{w} \in \mathbb{R}^{hk}$, which is applied to a window of $h$ words to produce a new feature. For example, a feature $c_i$ is generated from a window of words $\mathbf{x}_{i:i+h-1}$ by

$$c_i = f(\mathbf{w} \cdot \mathbf{x}_{i:i+h-1} + b). \quad (2)$$

Here $b \in \mathbb{R}$ is a bias term and $f$ is a non-linear function such as the hyperbolic tangent. This filter is applied to each possible window of words in the sentence $\{\mathbf{x}_{1:h}, \mathbf{x}_{2:h+1}, \ldots, \mathbf{x}_{n-h+1:n}\}$ to produce a *feature map*

$$\mathbf{c} = [c_1, c_2, \ldots, c_{n-h+1}], \quad (3)$$

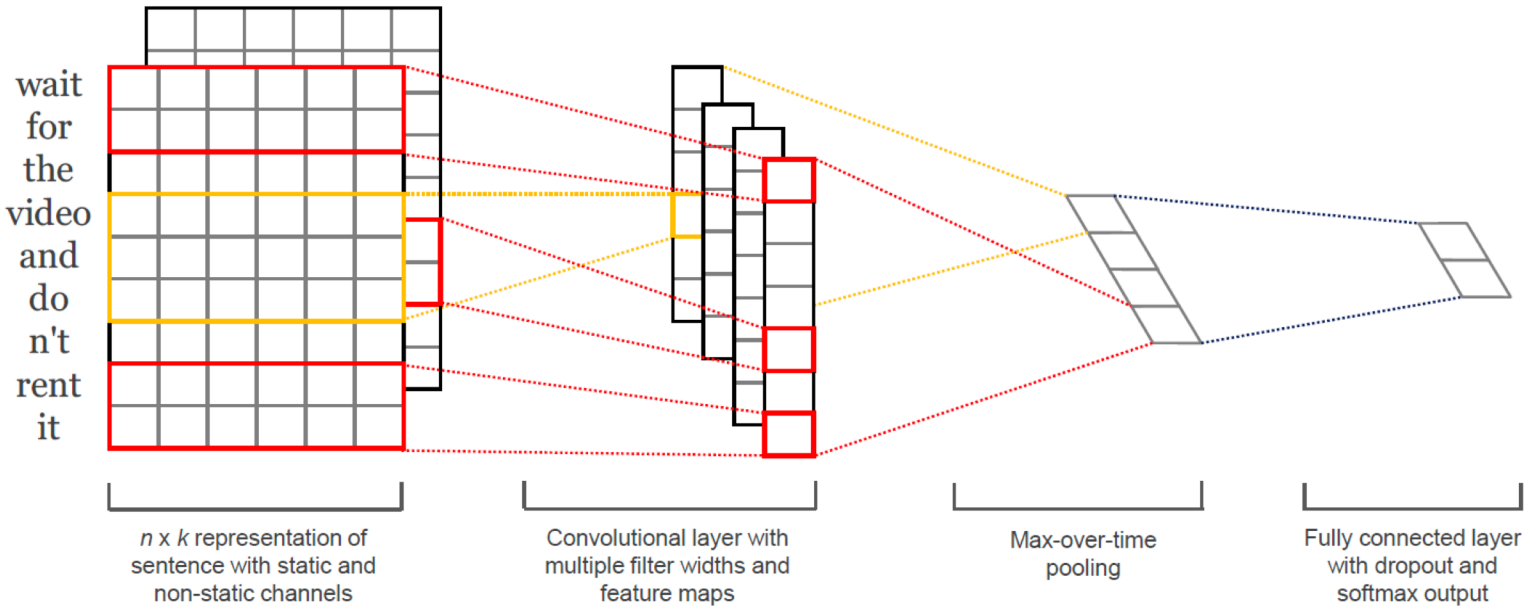
with $\mathbf{c} \in \mathbb{R}^{n-h+1}$. We then apply a max-over-time pooling operation (Collobert et al., 2011) over the feature map and take the maximum value $\hat{c} = \max\{\mathbf{c}\}$ as the feature corresponding to this particular filter. The idea is to capture the most important feature—one with the highest value—for each feature map. This pooling scheme naturally deals with variable sentence lengths.

We have described the process by which *one* feature is extracted from *one* filter. The model uses multiple filters (with varying window sizes) to obtain multiple features. These features form the penultimate layer and are passed to a fully connected softmax layer whose output is the probability distribution over labels.

In one of the model variants, we experiment with having two 'channels' of word vectors—one that is kept static throughout training and one that is fine-tuned via backpropagation (section 3.2).[2] In the multichannel architecture, illustrated in figure 1, each filter is applied to both channels and the results are added to calculate $c_i$ in equation (2). The model is otherwise equivalent to the single channel architecture.

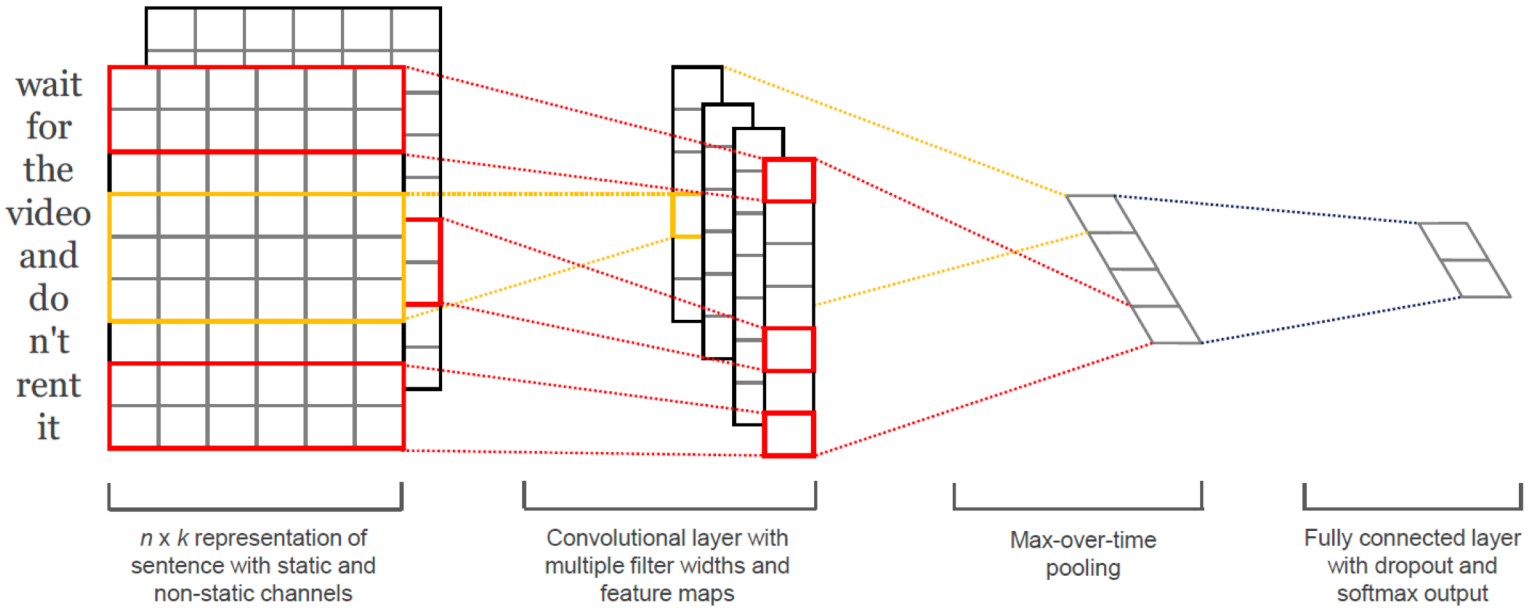
# Convolutional neural networks for sentence classification

## Model Architecture



# Convolutional neural networks for sentence classification

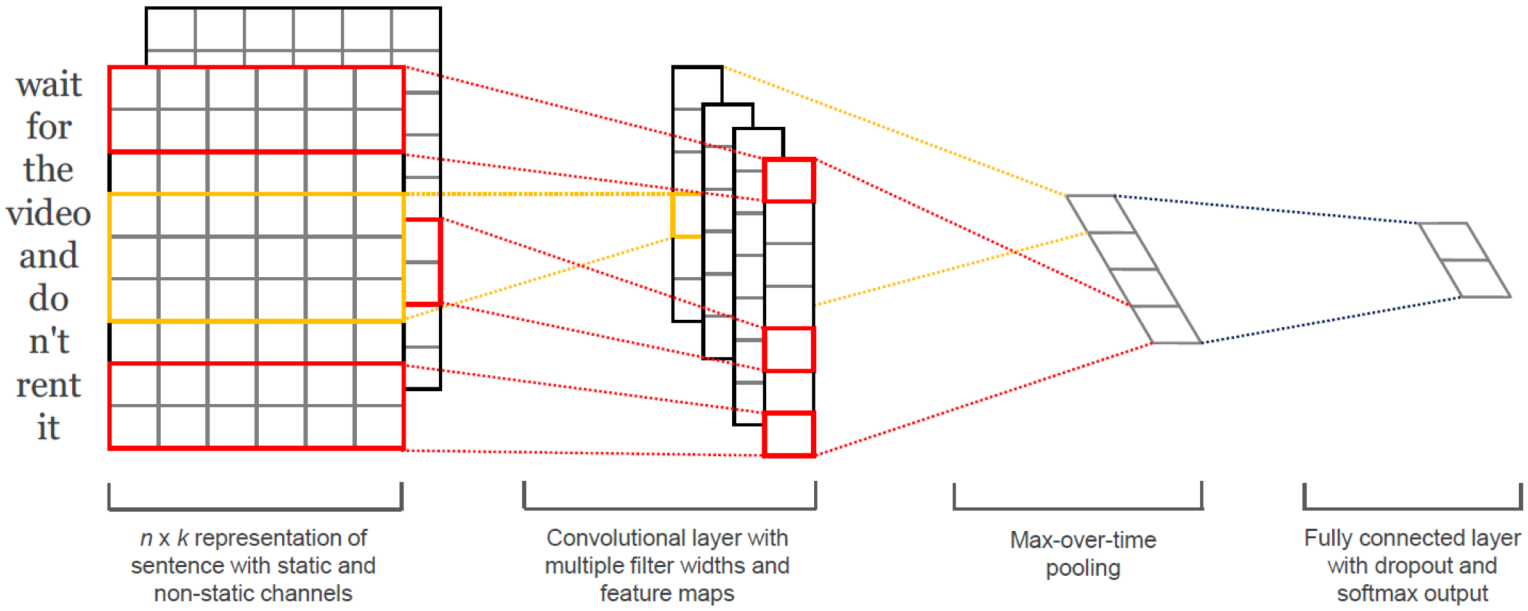
## Model Architecture




- $x_i \in R^k$: $k$-dimensional word vector

# Convolutional neural networks for sentence classification

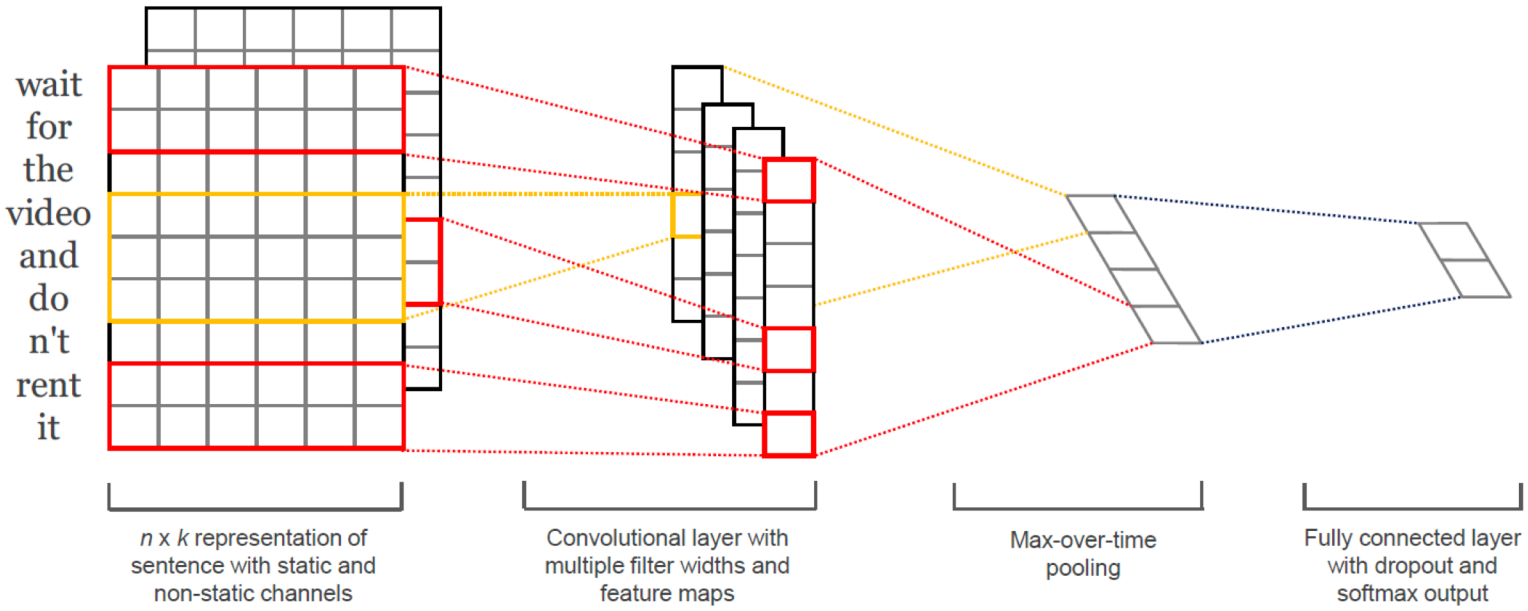
## Model Architecture




- $x_i \in R^k$: $k$-dimensional word vector
- $x_{1:n} = x_1 \oplus x_2 \oplus \ldots \oplus x_n$ (input sentence)

# Convolutional neural networks for sentence classification

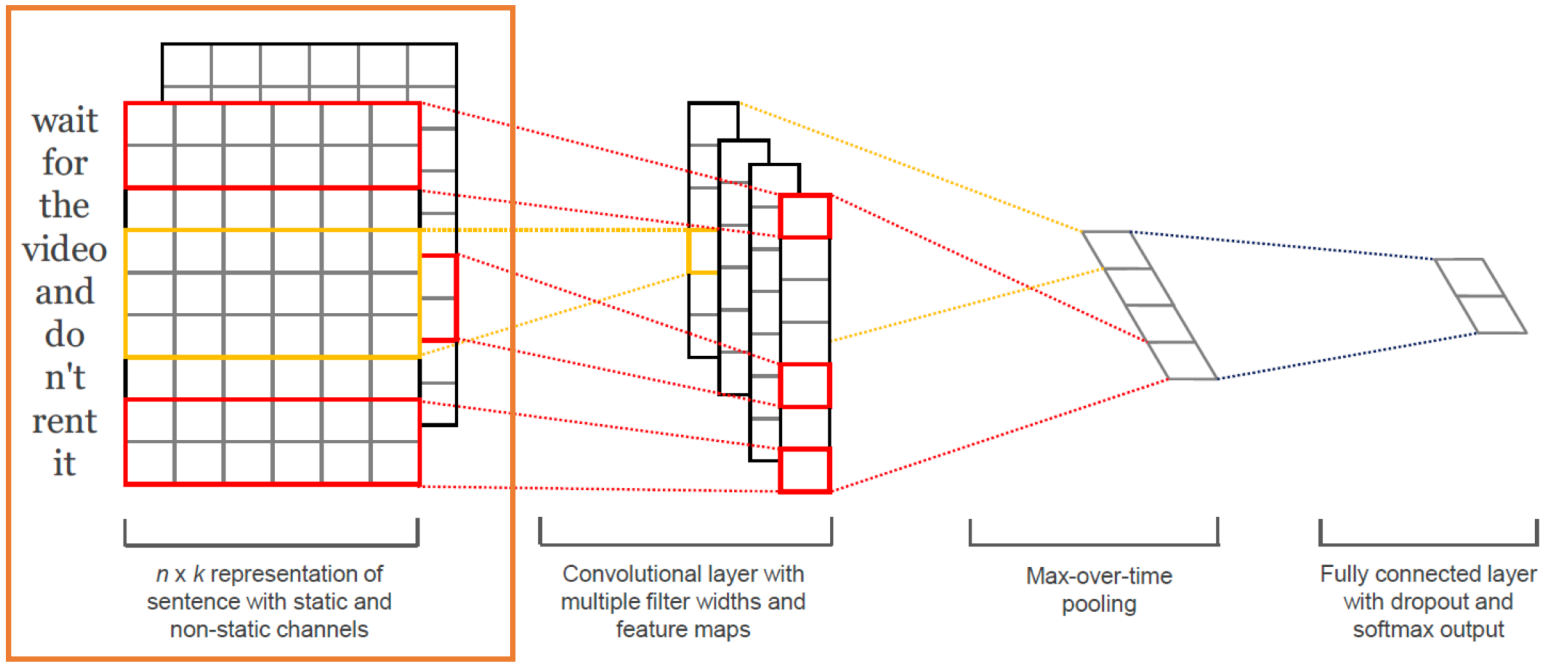
## Model Architecture




- $x_i \in R^k$: $k$-dimensional word vector
- $x_{1:n} = x_1 \oplus x_2 \oplus \ldots \oplus x_n$ (input sentence) ⇒ padded where necessary

# Convolutional neural networks for sentence classification

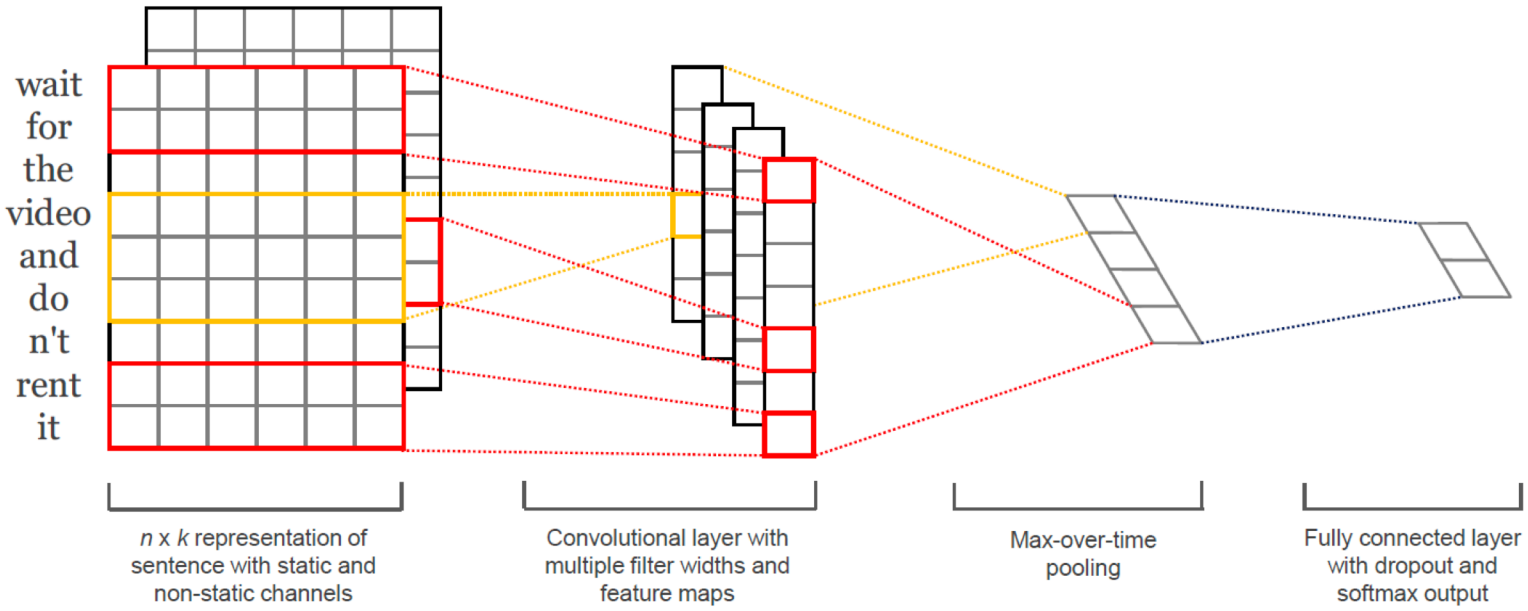
## Model Architecture




- $x_i \in R^k$: $k$-dimensional word vector
- $x_{1:n} = x_1 \oplus x_2 \oplus \ldots \oplus x_n$ (input sentence) ⇒ padded where necessary

# Convolutional neural networks for sentence classification

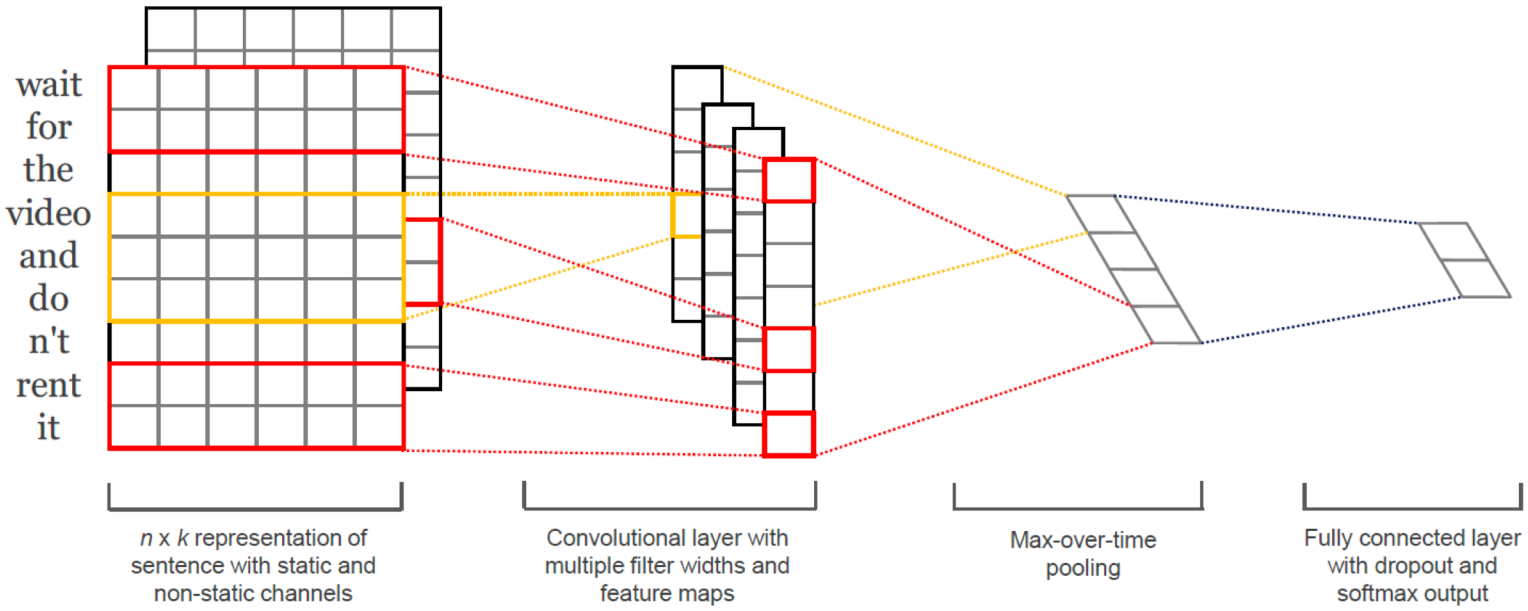
## Model Architecture




- $x_i \in R^k$: $k$-dimensional word vector
- $x_{1:n} = x_1 \oplus x_2 \oplus \ldots \oplus x_n$ (input sentence) ⇒ padded where necessary
- $c_i = f(w \cdot x_{i:i+h-1} + b)$, where w and b are convolutional filter parameter

# Convolutional neural networks for sentence classification

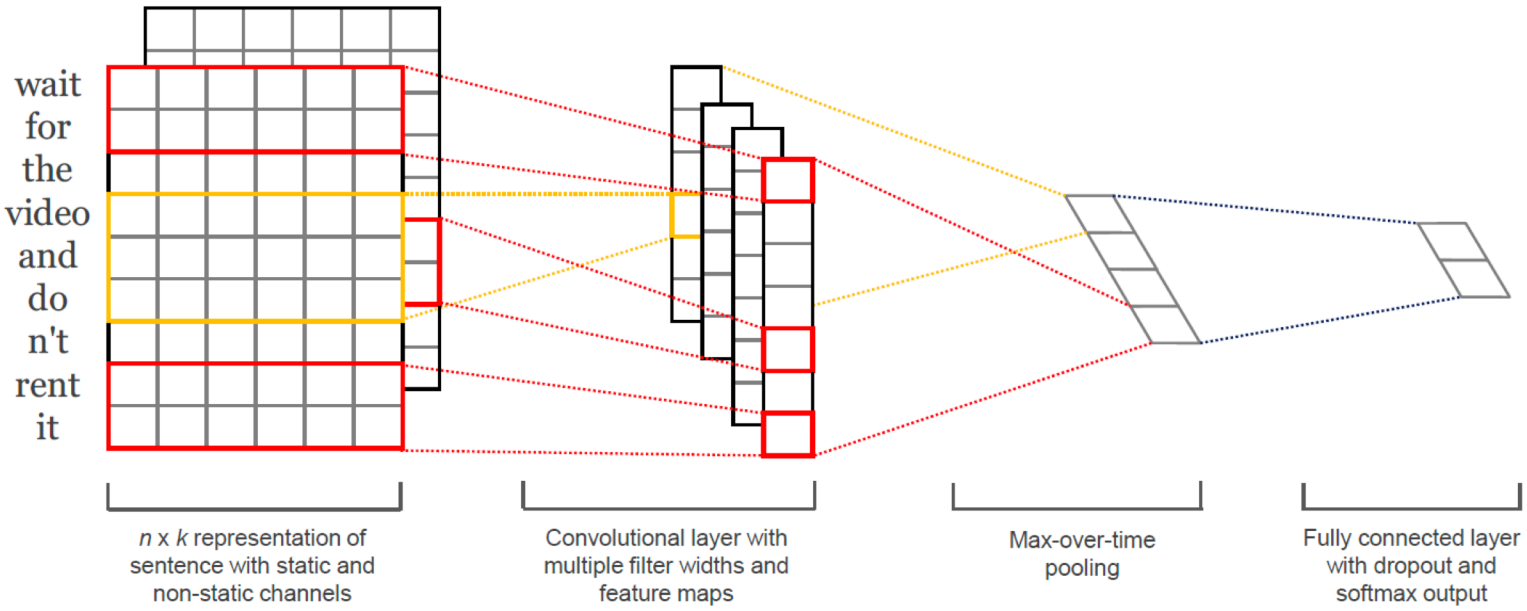
## Model Architecture




- $x_i \in R^k$: $k$-dimensional word vector
- $x_{1:n} = x_1 \oplus x_2 \oplus \ldots \oplus x_n$ (input sentence) ⇒ padded where necessary
- $c_i = f(w \cdot x_{i:i+h-1} + b)$, where w and b are convolutional filter parameter
- $\mathbf{c} = [c_1, c_2, \ldots, c_{n-h+1}]$

# Convolutional neural networks for sentence classification

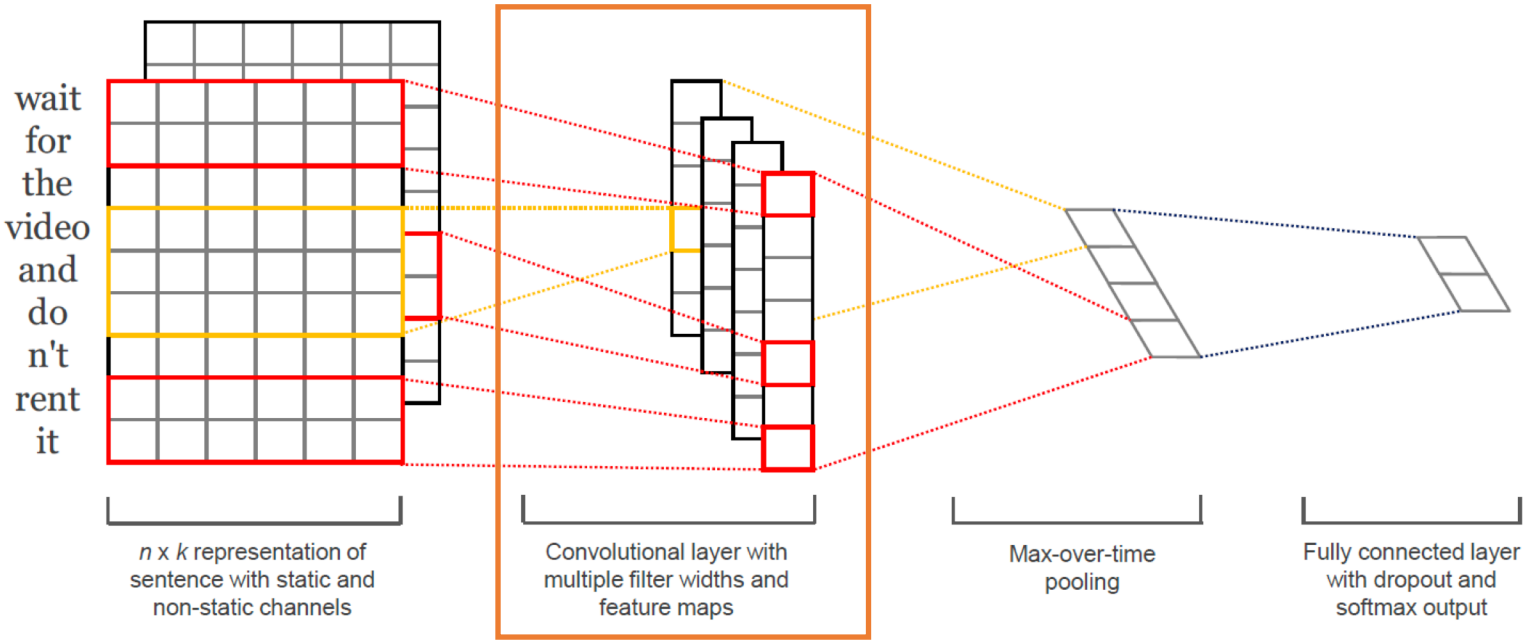
## Model Architecture




- $x_i \in R^k$: $k$-dimensional word vector
- $x_{1:n} = x_1 \oplus x_2 \oplus \ldots \oplus x_n$ (input sentence) ⇒ padded where necessary
- $\boldsymbol{c_i} = f(w \cdot x_{i:i+h-1} + b)$, where w and b are convolutional filter parameter
- $\mathbf{c} = [c_1, c_2, \ldots, c_{n-h+1}]$

# Convolutional neural networks for sentence classification

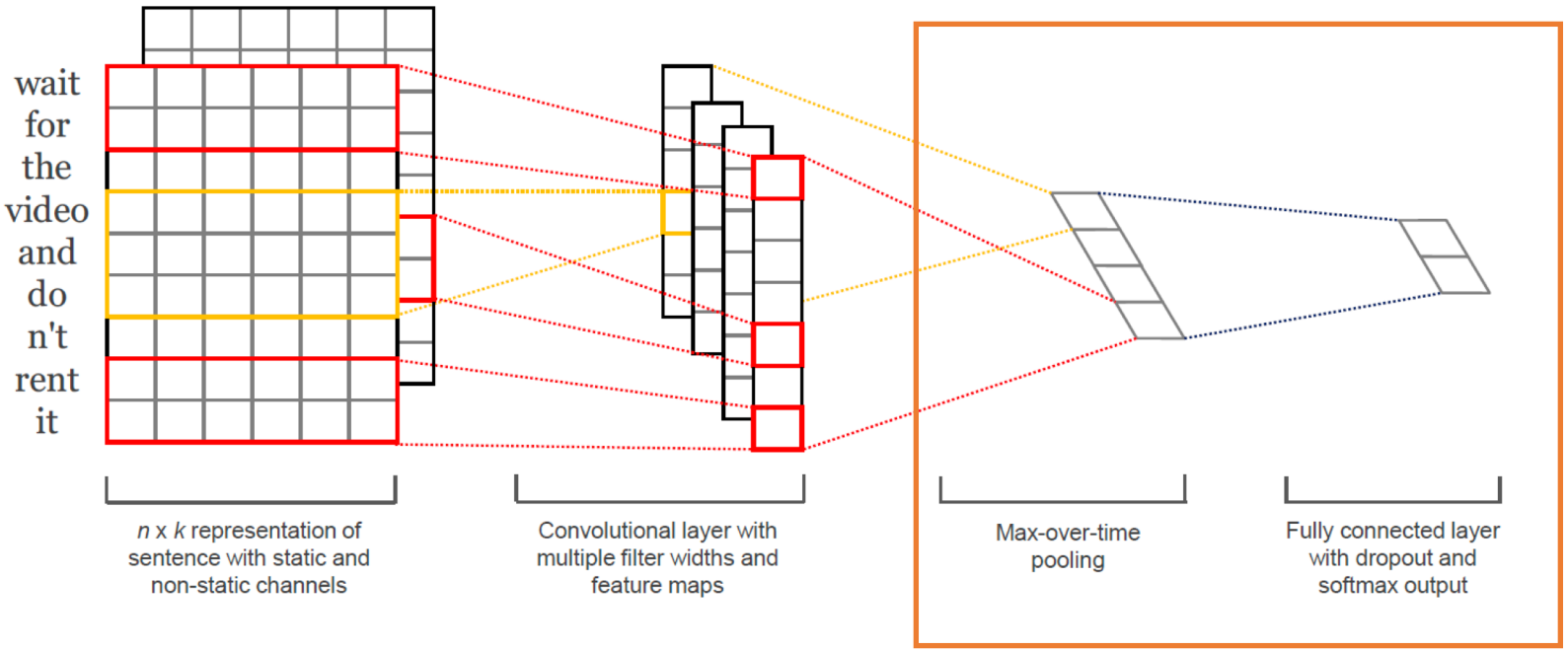
## Model Architecture




- $x_i \in R^k$: $k$-dimensional word vector
- $x_{1:n} = x_1 \oplus x_2 \oplus \ldots \oplus x_n$ (input sentence) ⇒ padded where necessary
- $c_i = f(w \cdot x_{i:i+h-1} + b)$, where w and b are convolutional filter parameter
- $\mathbf{c} = [c_1, c_2, \ldots, c_{n-h+1}]$

# Convolutional neural networks for sentence classification

## Model Architecture




max-pooling & fully connected layer

- $x_i \in R^k$: $k$-dimensional word vector
- $x_{1:n} = x_1 \oplus x_2 \oplus \ldots \oplus x_n$ (input sentence) ⇒ padded where necessary
- $c_i = f(w \cdot x_{i:i+h-1} + b)$, where w and b are convolutional filter parameter
- $\mathbf{c} = [c_1, c_2, \ldots, c_{n-h+1}]$

# Convolutional neural networks for sentence classification

## Regularization

# Convolutional neural networks for sentence classification

## Regularization

### 2.1 Regularization

For regularization we employ dropout on the penultimate layer with a constraint on $l_2$-norms of the weight vectors (Hinton et al., 2012). Dropout prevents co-adaptation of hidden units by randomly dropping out—i.e., setting to zero—a proportion $p$ of the hidden units during foward-backpropagation. That is, given the penultimate layer $\mathbf{z} = [\hat{c}_1, \ldots, \hat{c}_m]$ (note that here we have $m$ filters), instead of using

$$y = \mathbf{w} \cdot \mathbf{z} + b \tag{4}$$

for output unit $y$ in forward propagation, dropout uses

$$y = \mathbf{w} \cdot (\mathbf{z} \circ \mathbf{r}) + b, \tag{5}$$

where $\circ$ is the element-wise multiplication operator and $\mathbf{r} \in \mathbb{R}^m$ is a 'masking' vector of Bernoulli random variables with probability $p$ of being 1. Gradients are backpropagated only through the unmasked units. At test time, the learned weight vectors are scaled by $p$ such that $\hat{\mathbf{w}} = p\mathbf{w}$, and $\hat{\mathbf{w}}$ is used (without dropout) to score unseen sentences. We additionally constrain $l_2$-norms of the weight vectors by rescaling $\mathbf{w}$ to have $||\mathbf{w}||_2 = s$ whenever $||\mathbf{w}||_2 > s$ after a gradient descent step.

# Convolutional neural networks for sentence classification

## Regularization

- Dropout
- Clipping weight by L2-norm

# Convolutional neural networks for sentence classification

## Regularization

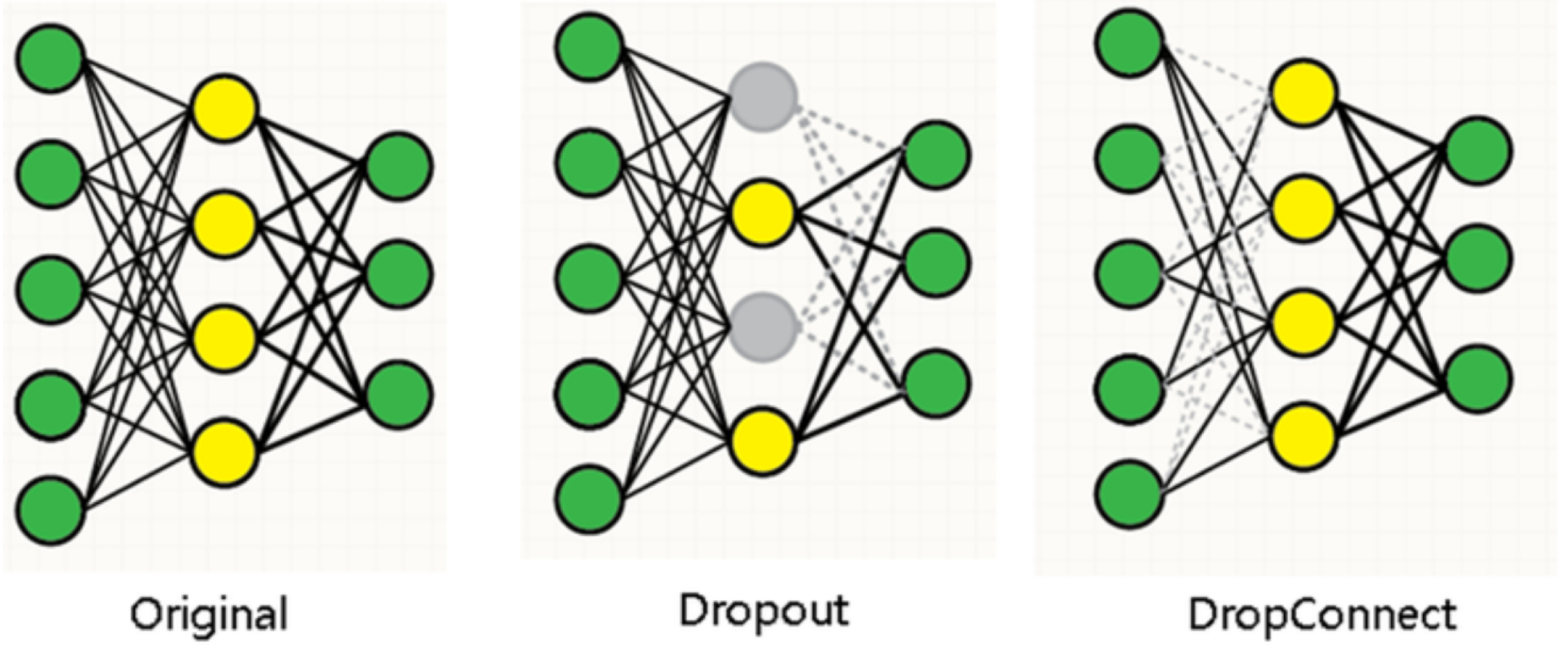
- **Dropout**
- Clipping weight by L2-norm

before)

$$y = \mathbf{w} \cdot \mathbf{z} + b$$

after)

$$y = \mathbf{w} \cdot (\mathbf{z} \circ \mathbf{r}) + b$$



# Convolutional neural networks for sentence classification

## Regularization

- **Dropout**
- Clipping weight by L2-norm

before)

$$y = \mathbf{w} \cdot \mathbf{z} + b$$

after)

$$y = \mathbf{w} \cdot (\mathbf{z} \circ \mathbf{r}) + b$$

$\mathbf{r} \in \mathrm{R}^m$: 'masking' vector of Bernoulli random variable with probability $p$ of being 1

# Convolutional neural networks for sentence classification

## Regularization

- **Dropout**
- Clipping weight by L2-norm

before)

$$y = \mathbf{w} \cdot \mathbf{z} + b$$

after)

$$y = \mathbf{w} \cdot (\mathbf{z} \circ \mathbf{r}) + b$$

| 0.3 |
|---|
| -0.4 |
| 0.5 |
| 1.2 |
| 0.04 |
| -0.2 |
| -1.4 |
| 0.84 |

$\mathbf{z}$

# Convolutional neural networks for sentence classification

## Regularization

- **Dropout**
- Clipping weight by L2-norm

before)

$$y = \mathbf{w} \cdot \mathbf{z} + b$$

after)

$$y = \mathbf{w} \cdot (\mathbf{z} \circ \mathbf{r}) + b$$

| $\mathbf{z}$ | $\mathbf{r}$ |
|---|---|
| 0.3 | 1 |
| -0.4 | 0 |
| 0.5 | 1 |
| 1.2 | 1 |
| 0.04 | 1 |
| -0.2 | 0 |
| -1.4 | 1 |
| 0.84 | 0 |

# Convolutional neural networks for sentence classification

## Regularization

- **Dropout**
- Clipping weight by L2-norm

before)

$$y = \mathbf{w} \cdot \mathbf{z} + b$$

after)

$$y = \mathbf{w} \cdot (\mathbf{z} \circ \mathbf{r}) + b$$

| $\mathbf{z}$ | | $\mathbf{r}$ |
|---|---|---|
| 0.3 | | 1 |
| -0.4 | | 0 |
| 0.5 | | 1 |
| 1.2 | | 1 |
| 0.04 | $\circ$ | 1 |
| -0.2 | | 0 |
| -1.4 | | 1 |
| 0.84 | | 0 |

# Convolutional neural networks for sentence classification

## Regularization

- **Dropout**
- Clipping weight by L2-norm

before)

$$y = \mathbf{w} \cdot \mathbf{z} + b$$

after)

$$y = \mathbf{w} \cdot (\mathbf{z} \circ \mathbf{r}) + b$$

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0.3 | | 1 | | 0.3 |
| -0.4 | | 0 | | 0 |
| 0.5 | | 1 | | 0.5 |
| 1.2 | | 1 | | 1.2 |
| 0.04 | ∘ | 1 | = | 0.04 |
| -0.2 | | 0 | | 0 |
| -1.4 | | 1 | | -1.4 |
| 0.84 | | 0 | | 0 |
| **z** | | **r** | | |

# Convolutional neural networks for sentence classification

## Regularization

- Dropout
- **Clipping weight by L2-norm**

# Convolutional neural networks for sentence classification

## Regularization

- Dropout
- **Clipping weight by L2-norm**

vector의 크기

- L1-norm
- L2-norm
- p-norm

# Convolutional neural networks for sentence classification

## Regularization

- Dropout
- **Clipping weight by L2-norm**

vector의 크기

- L1-norm $\|v\|_1 = |x_1 + x_2 + \cdots + x_n|$
- L2-norm
- p-norm

# Convolutional neural networks for sentence classification

## Regularization

- Dropout
- **Clipping weight by L2-norm**

vector의 크기

- L1-norm
- L2-norm
- p-norm

$$\|v\|_2 = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}$$

# Convolutional neural networks for sentence classification

## Regularization

- Dropout
- **Clipping weight by L2-norm**

vector의 크기

- L1-norm
- L2-norm
- p-norm

$$\|v\|_p = \sqrt[p]{x_1^p + x_2^p + \ ... + x_n^p}$$

# Convolutional neural networks for sentence classification

## Regularization

- Dropout
- **Clipping weight by L2-norm**

vector의 크기

- L1-norm
- L2-norm
- p-norm
- etc

$$|v| = 2|x_1| + \sqrt{3|x_2|^2 + \max(|x_3| + 2|x_4|)^2}$$

…

…

# Convolutional neural networks for sentence classification

## Regularization

- Dropout
- **Clipping weight by L2-norm**

$$w = \begin{cases} pw, & |w| \geq s \\ w, & |w| < s \end{cases}$$

$$(||pw||_2 = s)$$

# Convolutional neural networks for sentence classification

## Experimental Setup

# Convolutional neural networks for sentence classification

## Experimental Setup - Dataset

- MR – 영화 리뷰 긍정/부정 분류
- SST
- Subj – 주관적 또는 객관적 문장 분류
- TREC – 질문 분류 (6가지)
- CR – 제품 고객 리뷰 긍정/부정 분류
- MPQA – 의견 감정 분류

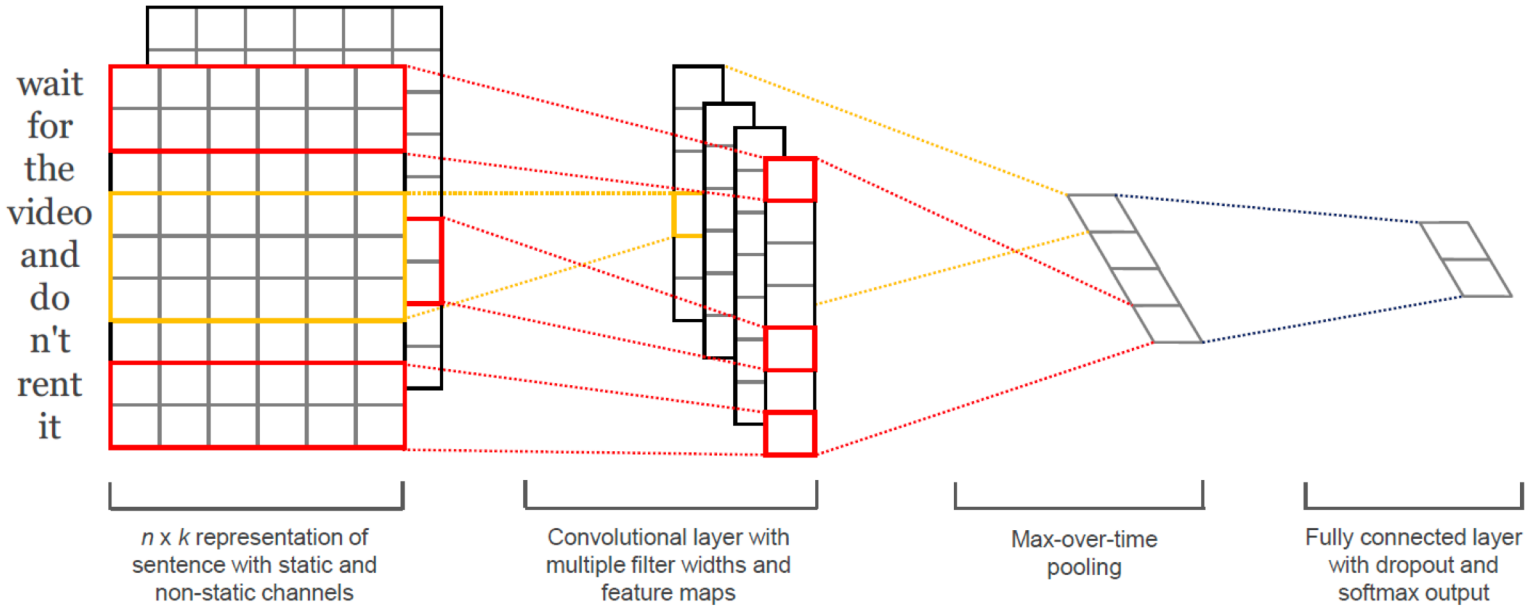
# Convolutional neural networks for sentence classification

## Experimental Setup - HyperParameters

- filter window size = 3, 4, 5
- 100 filter maps
- dropout rate = 0.5
- L2 constraint = 3
- batch size = 50

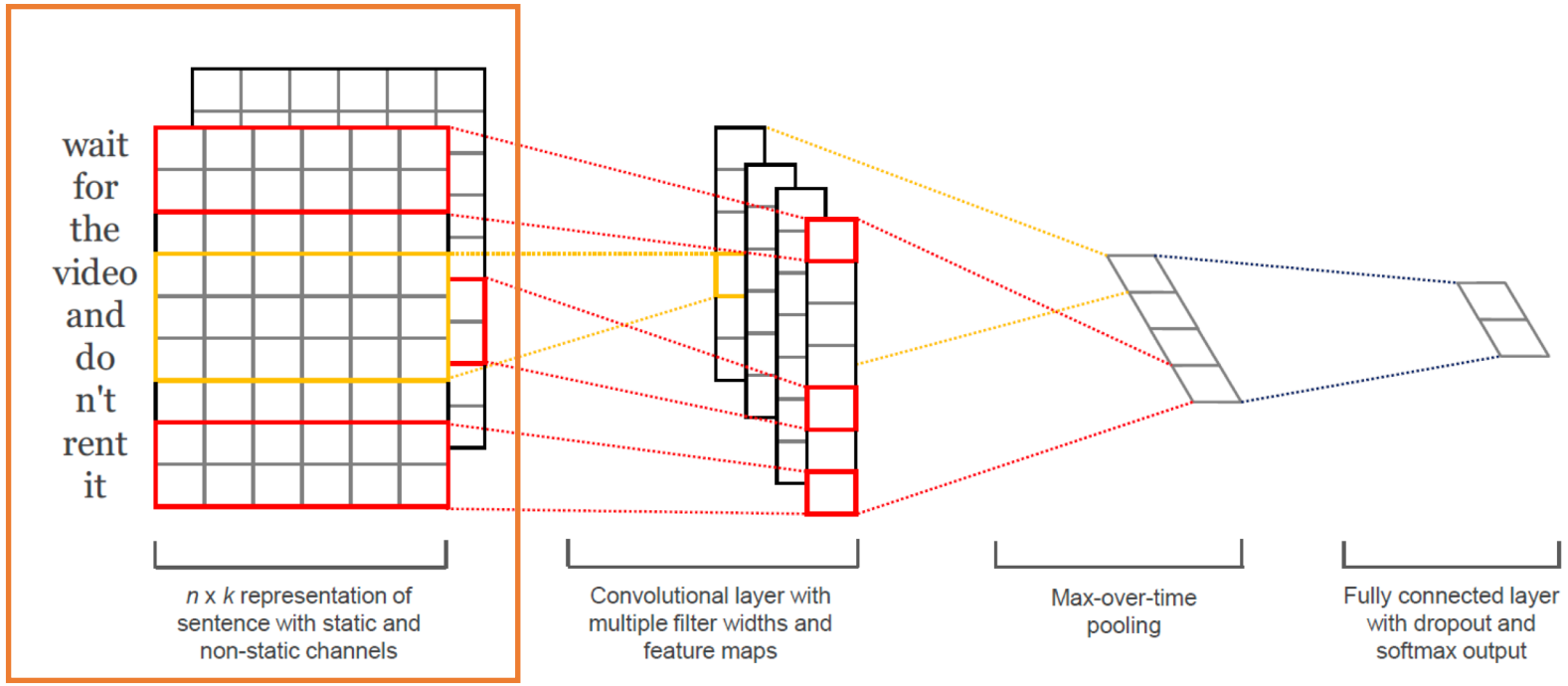
# Convolutional neural networks for sentence classification

## Experimental Setup - Pre-trained vector



# Convolutional neural networks for sentence classification

## Experimental Setup - Pre-trained vector



# Convolutional neural networks for sentence classification

## Experimental Setup - Pre-trained vector

- CNN-rand
- CNN-static
- CNN-non-static
- CNN-multichannel

# Convolutional neural networks for sentence classification

## Experimental Setup - Pre-trained vector

- CNN-rand ⟶ Randomly initialized
- CNN-static
- CNN-non-static
- CNN-multichannel

# Convolutional neural networks for sentence classification

## Experimental Setup - Pre-trained vector

- CNN-rand
- CNN-static → Pre-trained by word2vec & word vector doesn't update during training
- CNN-non-static
- CNN-multichannel

# Convolutional neural networks for sentence classification

## Experimental Setup - Pre-trained vector

- CNN-rand
- CNN-static
- CNN-non-static → Pre-trained by word2vec & word vector update during training
- CNN-multichannel

# Convolutional neural networks for sentence classification

## Experimental Setup - Pre-trained vector

- CNN-rand
- CNN-static
- CNN-non-static
- CNN-multichannel ⟶ Static + Non-Static

# Convolutional neural networks for sentence classification

## Experimental Setup - Result

| Model | MR | SST-1 | SST-2 | Subj | TREC | CR | MPQA |
|---|---|---|---|---|---|---|---|
| CNN-rand | 76.1 | 45.0 | 82.7 | 89.6 | 91.2 | 79.8 | 83.4 |
| CNN-static | 81.0 | 45.5 | 86.8 | 93.0 | 92.8 | 84.7 | **89.6** |
| CNN-non-static | **81.5** | 48.0 | 87.2 | 93.4 | 93.6 | 84.3 | 89.5 |
| CNN-multichannel | 81.1 | 47.4 | **88.1** | 93.2 | 92.2 | **85.0** | 89.4 |
| RAE (Socher et al., 2011) | 77.7 | 43.2 | 82.4 | – | – | – | 86.4 |
| MV-RNN (Socher et al., 2012) | 79.0 | 44.4 | 82.9 | – | – | – | – |
| RNTN (Socher et al., 2013) | – | 45.7 | 85.4 | – | – | – | – |
| DCNN (Kalchbrenner et al., 2014) | – | 48.5 | 86.8 | – | 93.0 | – | – |
| Paragraph-Vec (Le and Mikolov, 2014) | – | **48.7** | 87.8 | – | – | – | – |
| CCAE (Hermann and Blunsom, 2013) | 77.8 | – | – | – | – | – | 87.2 |
| Sent-Parser (Dong et al., 2014) | 79.5 | – | – | – | – | – | 86.3 |
| NBSVM (Wang and Manning, 2012) | 79.4 | – | – | 93.2 | – | 81.8 | 86.3 |
| MNB (Wang and Manning, 2012) | 79.0 | – | – | **93.6** | – | 80.0 | 86.3 |
| G-Dropout (Wang and Manning, 2013) | 79.0 | – | – | 93.4 | – | 82.1 | 86.1 |
| F-Dropout (Wang and Manning, 2013) | 79.1 | – | – | **93.6** | – | 81.9 | 86.3 |
| Tree-CRF (Nakagawa et al., 2010) | 77.3 | – | – | – | – | 81.4 | 86.1 |
| CRF-PR (Yang and Cardie, 2014) | – | – | – | – | – | 82.7 | – |
| $SVM_S$ (Silva et al., 2011) | – | – | – | – | **95.0** | – | – |

# Convolutional neural networks for sentence classification

Static vs Non-Static

# Convolutional neural networks for sentence classification

Case by case !

Static vs Non-Static

# Convolutional neural networks for sentence classification

But generally

Static vs Non-Static

Small data

Enough data

# Character-level Convolutional Networks for Text Classification

- X Zhang et al. (2015)
- 기존의 방법들은 word 단위였던 것과 달리 문자(Character) 단위로 입력 값 구성

## Character-level Convolutional Networks for Text Classification*

**Xiang Zhang** **Junbo Zhao** **Yann LeCun**
Courant Institute of Mathematical Sciences, New York University
719 Broadway, 12th Floor, New York, NY 10003
{xiang, junbo.zhao, yann}@cs.nyu.edu

**Abstract**

This article offers an empirical exploration on the use of character-level convolutional networks (ConvNets) for text classification. We constructed several large-scale datasets to show that character-level convolutional networks could achieve state-of-the-art or competitive results. Comparisons are offered against traditional models such as bag of words, n-grams and their TFIDF variants, and deep learning models such as word-based ConvNets and recurrent neural networks.

### 1 Introduction

Text classification is a classic topic for natural language processing, in which one needs to assign predefined categories to free-text documents. The range of text classification research goes from designing the best features to choosing the best possible machine learning classifiers. To date, almost all techniques of text classification are based on words, in which simple statistics of some ordered word combinations (such as n-grams) usually perform the best [12].

On the other hand, many researchers have found convolutional networks (ConvNets) [17] [18] are useful in extracting information from raw signals, ranging from computer vision applications to speech recognition and others. In particular, time-delay networks used in the early days of deep learning research are essentially convolutional networks that model sequential data [1] [31].

In this article we explore treating text as a kind of raw signal at character level, and applying temporal (one-dimensional) ConvNets to it. For this article we only used a classification task as a way to exemplify ConvNets' ability to understand texts. Historically we know that ConvNets usually require large-scale datasets to work, therefore we also build several of them. An extensive set of comparisons is offered with traditional models and other deep learning models.

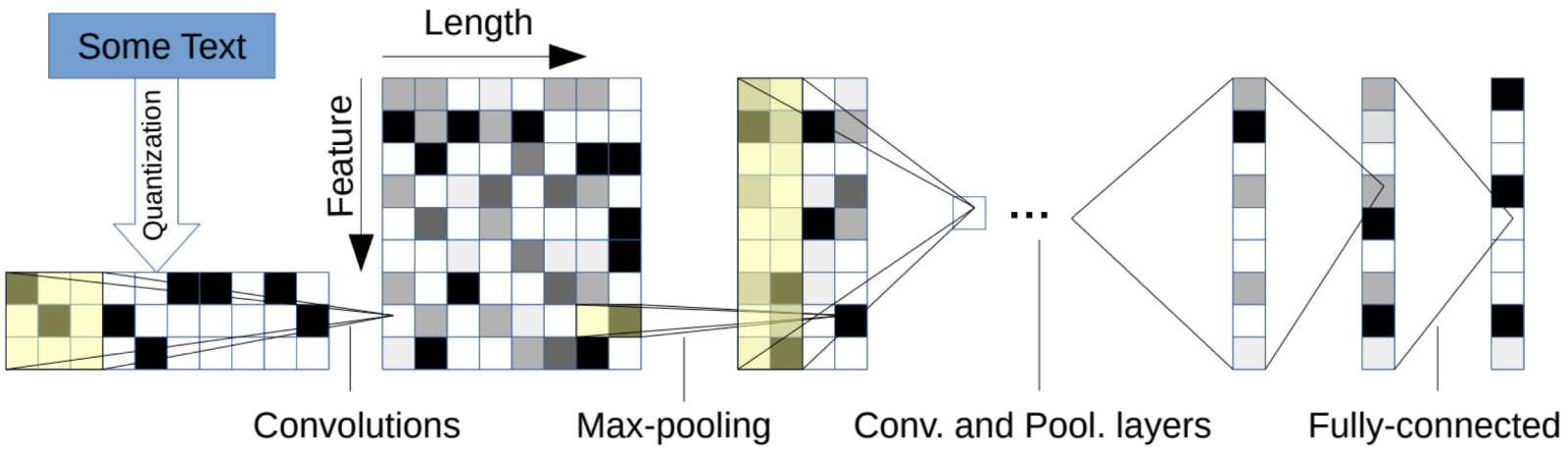
Applying convolutional networks to text classification or natural language processing at large was explored in literature. It has been shown that ConvNets can be directly applied to distributed [6] [16] or discrete [13] embedding of words, without any knowledge on the syntactic or semantic structures of a language. These approaches have been proven to be competitive to traditional models.

There are also related works that use character-level features for language processing. These include using character-level n-grams with linear classifiers [15], and incorporating character-level features to ConvNets [28] [29]. In particular, these ConvNet approaches use words as a basis, in which character-level features extracted at word [28] or word n-gram [29] level form a distributed representation. Improvements for part-of-speech tagging and information retrieval were observed.

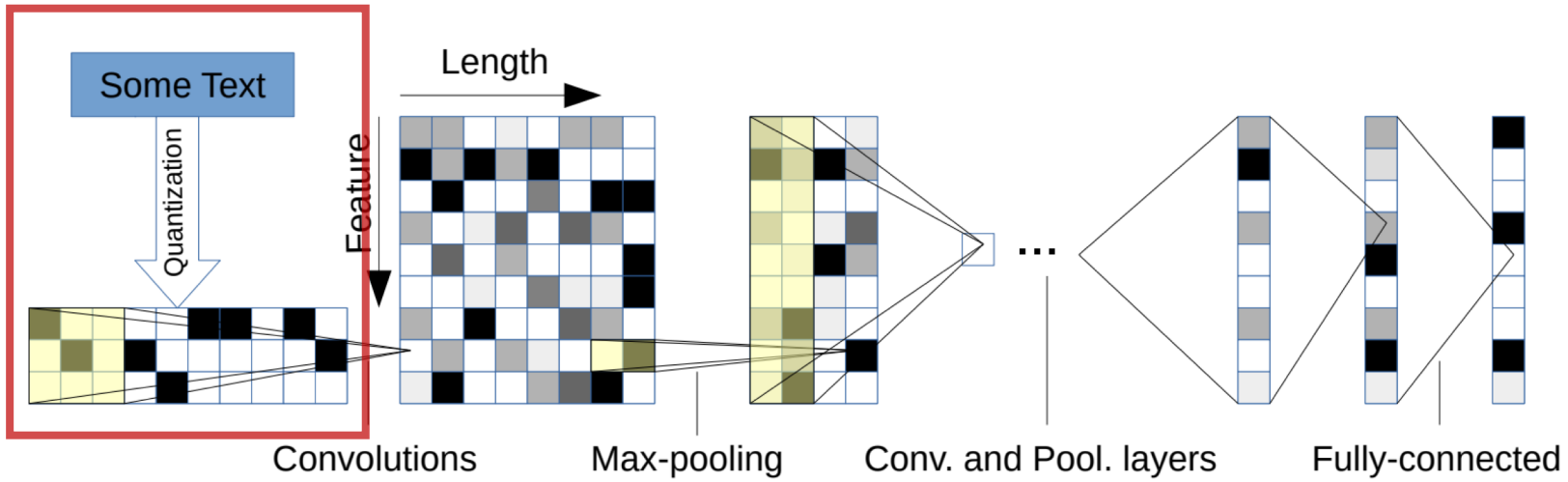
This article is the first to apply ConvNets only on characters. We show that when trained on large-scale datasets, deep ConvNets do not require the knowledge of words, in addition to the conclusion

*An early version of this work entitled "Text Understanding from Scratch" was posted in Feb 2015 as arXiv:1502.01710. The present paper has considerably more experimental results and a rewritten introduction.

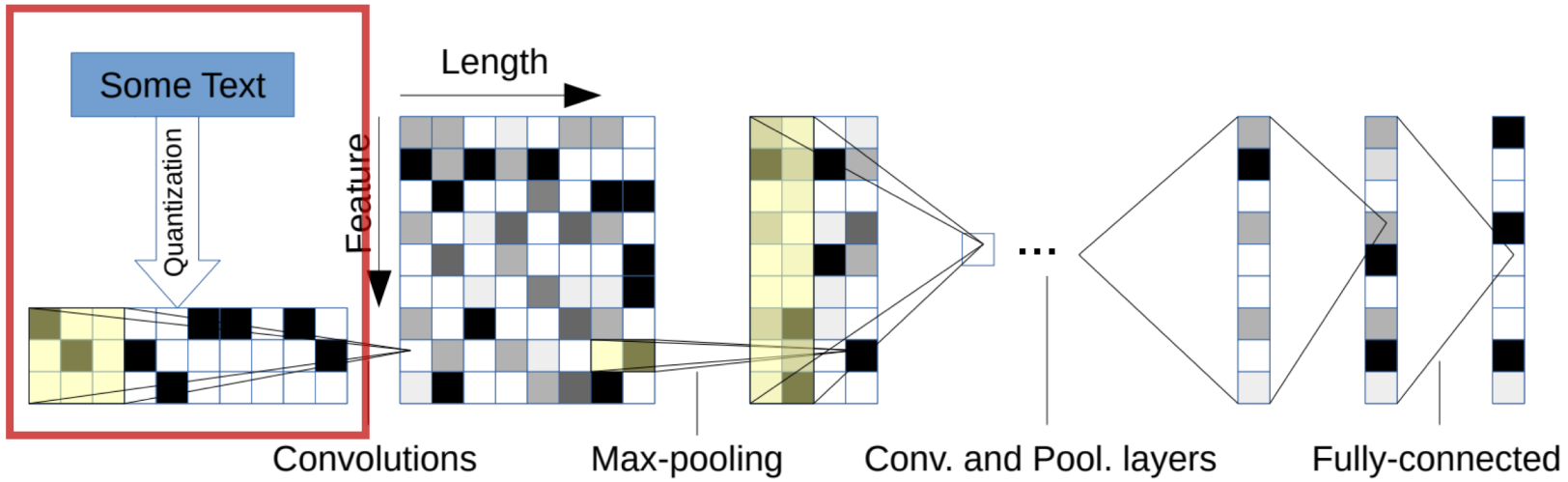
# Character-level Convolutional Networks for Text Classification



# Character-level Convolutional Networks for Text Classification



# Character-level Convolutional Networks for Text Classification




알파벳 : 26 숫자 : 10

```
abcdefghijklmnopqrstuvwxyz0123456789
-,;.!?:’’’/\|_@#$%ˆ&*˜‘+-=<>()[]{}
```

특수 : 34 총 : 70

Input Vocabulary

# Character-level Convolutional Networks for Text Classification

## Advantage of Character unit

- Small vocabulary size
- Good for misspelling
- Good for Unseen word
- Good for Informal text