20 Fall ESC Week5

ConvNets for NLP

6조 김채형 백채빈 서경덕 이규민 조유림

NO	contents
1	Intro to CNNs
2	Simple CNN for Sentence Classification
3	CNN potpourri
4	Deep CNN for Sentence Classification
5	Quasi-recurrent Neural Networks

01

Intro to CNN

01

CNN for NLP 개념

등장배경

RNN의 문제점

문제점 01

cannot capture phrase
without prefix context

문제점 02.

최종 벡터에 마지막 단어가
너무 많이 포함됨

문제점 03

소프트맥스 함수가
보통 마지막에 포함됨

결론은 CNN이 답이다

특정 길이의 모든 가능한 word subsequence에 대하여 벡터를 계산하자!

분류 문제(감성 분석, 스팸 탐지, 주제 분류 등)에 주로 활용

01

CNN for NLP 개념

Convolution Neural Network & Convolution

Convolution Neural Network란?

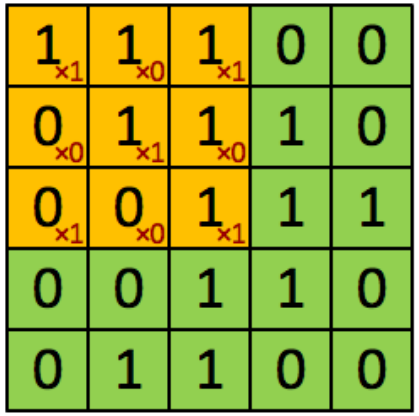
Window가 sliding하며 convolution 연산이 수행되는 과정을 포함한 네트워크

Convolution이란?

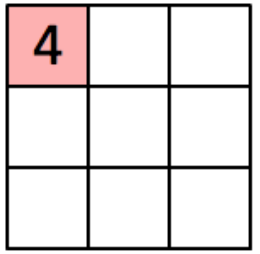
필터를 움직이면서 weight를 곱한 후 더하는 "가중합(Weighted Sum)" 연산

1d discrete convolution

$$(f * g)[n] = \sum_{m=-M}^{M} f[n-m]g[m].$$



Image



Convolved Feature

01

CNN for NLP 개념

input

NLP에서의 CNN의 input은 문장 혹은 문서 전체의 matrix 형태

- matrix의 각 행은 하나의 token (이때 token은 주로 단어 혹은 문자)
- 즉 matrix의 각 행은 각각의 단어 벡터 (임베딩 벡터 혹은 원-핫 벡터)

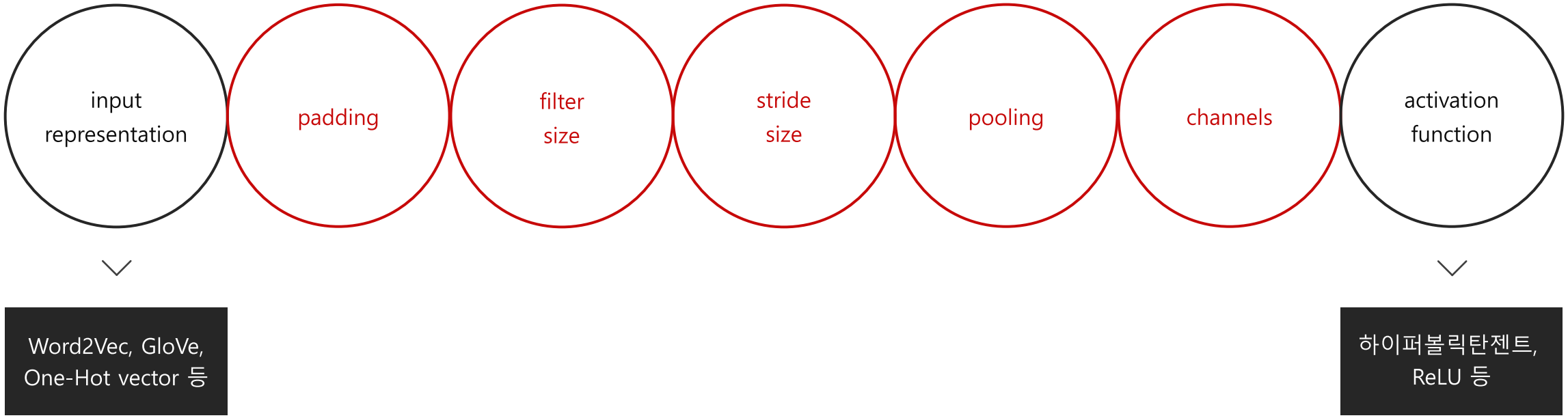
- 예를 들어 단어 벡터의 차원이 100이고, 단어의 수가 10이라면
- matrix 즉 input은 10 x 100 크기를 가짐

tentative	0.2	0.1	−0.3	0.4
deal	0.5	0.2	−0.3	−0.1
reached	−0.1	−0.3	−0.2	0.4
to	0.3	−0.3	0.1	0.1
keep	0.2	−0.3	0.4	0.2
government	0.1	0.2	−0.1	−0.1
open	−0.4	−0.4	0.2	0.3

02

CNN Hyperparameter

종류



02

CNN Hyperparameter

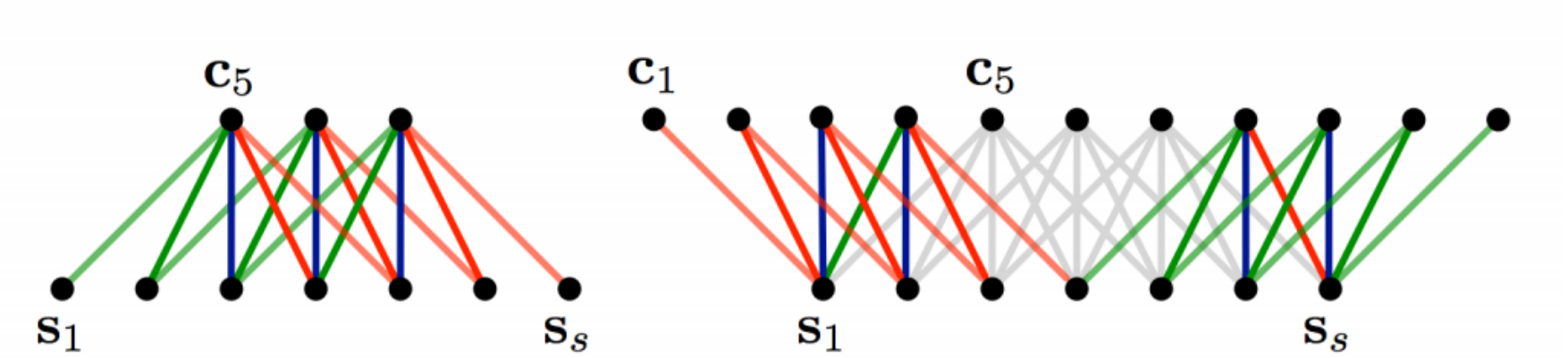
padding

padding 역할

- 가장자리에 추가로 모든 원소가 0을 갖도록 만들어 주어
- matrix의 모든 요소에 대하여 convolution 수행 가능하도록 함

padding 결과 output size

- input에 비해 좀 더 크거나 동일한 크기의 output size
- output size : n_out = (n_in + 2 * n_padding − n_filter) + 1



Narrow Convolution
zero padding 사용하지 않은 것
output size = (7-5)+1=3

Wide Convolution
zero padding 사용한 것
output size = (7+2*4-5)+1=11

02

CNN Hyperparameter

filter size / stride size

–

filter
size

필터의 넓이
= matrix의 넓이
= 단어 벡터의 차원(dimension)

필터의 높이 (region size)
: 보통 2~5개의 단어를 sliding

–

stride
size

각 단계에서
필터를 얼마만큼 움직여서
convolution을 수행할 것인지를
결정하는 것

stride size가 클수록
- 필터가 더 적게 사용됨
- output size가 작아짐
- RNN과 비슷한 효과
- (tree 구조)

02

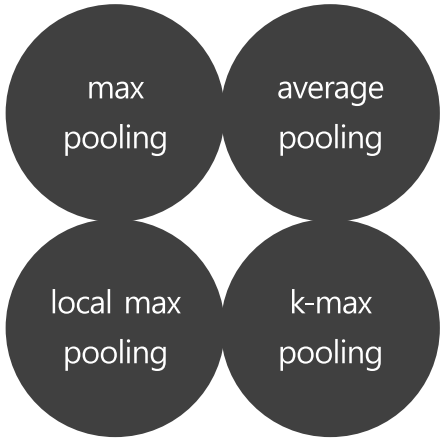
CNN Hyperparameter

pooling

pooling

주로
convolution layer
다음에 pooling layer
적용

pooling 종류



pooling 사용 이유

- 고정된 크기의 output을 만들 수 있음
- 각각 길이가 다른 문장들을 input으로 넣을 수 있게 됨
- 크기(dimension)가 줄어도 중요한 정보는 모두 보존

02

CNN Hyperparameter

channel

channel은 input이 무엇인지에 따라 개념이 달라질 수 있음!

CV에서는
- 컬러 이미지를 input으로 할 때 RGB 3개의 값을 각각 하나의 channel로 가짐

NLP에서는
- Word2Vec으로 임베딩 한 벡터들이 모인 matrix를 하나의 channel로, GloVe로 임베딩 한 벡터들이 모인 matrix를 또 하나의 channel로 가짐
- 같은 문장에 대해 서로 다른 언어로 표현한 것을 각 channel로 가짐

CNN Hyperparameter

A 1D convolution for text

tentative	0.2	0.1	−0.3	0.4
deal	0.5	0.2	−0.3	−0.1
reached	−0.1	−0.3	−0.2	0.4
to	0.3	−0.3	0.1	0.1
keep	0.2	−0.3	0.4	0.2
government	0.1	0.2	−0.1	−0.1
open	−0.4	−0.4	0.2	0.3

t,d,r	−1.0
d,r,t	−0.5
r,t,k	−3.6
t,k,g	−0.2
k,g,o	0.3

Apply a **filter** (or **kernel**) of size 3

3	1	2	−3
−1	2	1	−3
1	1	−1	1

CNN Hyperparameter

1D convolution for text with padding

Ø	0.0	0.0	0.0	0.0
tentative	0.2	0.1	−0.3	0.4
deal	0.5	0.2	−0.3	−0.1
reached	−0.1	−0.3	−0.2	0.4
to	0.3	−0.3	0.1	0.1
keep	0.2	−0.3	0.4	0.2
government	0.1	0.2	−0.1	−0.1
open	−0.4	−0.4	0.2	0.3
Ø	0.0	0.0	0.0	0.0

Ø,t,d	−0.6
t,d,r	−1.0
d,r,t	−0.5
r,t,k	−3.6
t,k,g	−0.2
k,g,o	0.3
g,o,Ø	−0.5

Apply a **filter** (or **kernel**) of size 3

3	1	2	−3
−1	2	1	−3
1	1	−1	1

CNN Hyperparameter

3 filters 1D convolution with padding = 1

Ø	0.0	0.0	0.0	0.0
tentative	0.2	0.1	−0.3	0.4
deal	0.5	0.2	−0.3	−0.1
reached	−0.1	−0.3	−0.2	0.4
to	0.3	−0.3	0.1	0.1
keep	0.2	−0.3	0.4	0.2
government	0.1	0.2	−0.1	−0.1
open	−0.4	−0.4	0.2	0.3
Ø	0.0	0.0	0.0	0.0

Ø,t,d	−0.6	0.2	1.4
t,d,r	−1.0	1.6	−1.0
d,r,t	−0.5	−0.1	0.8
r,t,k	−3.6	0.3	0.3
t,k,g	−0.2	0.1	1.2
k,g,o	0.3	0.6	0.9
g,o,Ø	−0.5	−0.9	0.1

Apply 3 **filters** of size 3

3	1	2	−3
−1	2	1	−3
1	1	−1	1

1	0	0	1
1	0	−1	−1
0	1	0	1

1	−1	2	−1
1	0	−1	3
0	2	2	1

Could also use (zero)

padding = 2

Also called “wide convolution”

CNN Hyperparameter

conv1d, padded with max pooling over time

Ø	0.0	0.0	0.0	0.0
tentative	0.2	0.1	−0.3	0.4
deal	0.5	0.2	−0.3	−0.1
reached	−0.1	−0.3	−0.2	0.4
to	0.3	−0.3	0.1	0.1
keep	0.2	−0.3	0.4	0.2
government	0.1	0.2	−0.1	−0.1
open	−0.4	−0.4	0.2	0.3
Ø	0.0	0.0	0.0	0.0

Ø,t,d	−0.6	0.2	1.4
t,d,r	−1.0	1.6	−1.0
d,r,t	−0.5	−0.1	0.8
r,t,k	−3.6	0.3	0.3
t,k,g	−0.2	0.1	1.2
k,g,o	0.3	0.6	0.9
g,o,Ø	−0.5	−0.9	0.1

max p	0.3	1.6	1.4

Apply 3 **filters** of size 3

3	1	2	−3
−1	2	1	−3
1	1	−1	1

1	0	0	1
1	0	−1	−1
0	1	0	1

1	−1	2	−1
1	0	−1	3
0	2	2	1

CNN Hyperparameter

conv1d, padded with ave pooling over time

ø	0.0	0.0	0.0	0.0
tentative	0.2	0.1	−0.3	0.4
deal	0.5	0.2	−0.3	−0.1
reached	−0.1	−0.3	−0.2	0.4
to	0.3	−0.3	0.1	0.1
keep	0.2	−0.3	0.4	0.2
government	0.1	0.2	−0.1	−0.1
open	−0.4	−0.4	0.2	0.3
ø	0.0	0.0	0.0	0.0

ø,t,d	−0.6	0.2	1.4
t,d,r	−1.0	1.6	−1.0
d,r,t	−0.5	−0.1	0.8
r,t,k	−3.6	0.3	0.3
t,k,g	−0.2	0.1	1.2
k,g,o	0.3	0.6	0.9
g,o,ø	−0.5	−0.9	0.1
ave p	−0.87	0.26	0.53

Apply 3 **filters** of size 3

	3	1	2	−3
	−1	2	1	−3
14	1	1	−1	1

1	0	0	1
1	0	−1	−1
0	1	0	1

1	−1	2	−1
1	0	−1	3
0	2	2	1

CNN Hyperparameter

stride = 2

Ø	0.0	0.0	0.0	0.0
tentative	0.2	0.1	−0.3	0.4
deal	0.5	0.2	−0.3	−0.1
reached	−0.1	−0.3	−0.2	0.4
to	0.3	−0.3	0.1	0.1
keep	0.2	−0.3	0.4	0.2
government	0.1	0.2	−0.1	−0.1
open	−0.4	−0.4	0.2	0.3
Ø	0.0	0.0	0.0	0.0

Ø,t,d	−0.6	0.2	1.4
d,r,t	−0.5	−0.1	0.8
t,k,g	−0.2	0.1	1.2
g,o,Ø	−0.5	−0.9	0.1

Apply 3 **filters** of size 3

	3	1	2	−3
	−1	2	1	−3
16	1	1	−1	1

1	0	0	1
1	0	−1	−1
0	1	0	1

1	−1	2	−1
1	0	−1	3
0	2	2	1

CNN Hyperparameter

local max pool, stride = 2

Ø	0.0	0.0	0.0	0.0
tentative	0.2	0.1	−0.3	0.4
deal	0.5	0.2	−0.3	−0.1
reached	−0.1	−0.3	−0.2	0.4
to	0.3	−0.3	0.1	0.1
keep	0.2	−0.3	0.4	0.2
government	0.1	0.2	−0.1	−0.1
open	−0.4	−0.4	0.2	0.3
Ø	0.0	0.0	0.0	0.0

Apply 3 **filters** of size 3

3	1	2	−3
−1	2	1	−3
1	1	−1	1

1	0	0	1
1	0	−1	−1
0	1	0	1

1	−1	2	−1
1	0	−1	3
0	2	2	1

Ø,t,d	−0.6	0.2	1.4
t,d,r	−1.0	1.6	−1.0
d,r,t	−0.5	−0.1	0.8
r,t,k	−3.6	0.3	0.3
t,k,g	−0.2	0.1	1.2
k,g,o	0.3	0.6	0.9
g,o,Ø	−0.5	−0.9	0.1
Ø	−Inf	−Inf	−Inf

Ø,t,d,r	−0.6	1.6	1.4
d,r,t,k	−0.5	0.3	0.8
t,k,g,o	0.3	0.6	1.2
g,o,Ø,Ø	−0.5	−0.9	0.1

CNN Hyperparameter

conv1d, k-max pooling over time, k = 2

Ø	0.0	0.0	0.0	0.0
tentative	0.2	0.1	−0.3	0.4
deal	0.5	0.2	−0.3	−0.1
reached	−0.1	−0.3	−0.2	0.4
to	0.3	−0.3	0.1	0.1
keep	0.2	−0.3	0.4	0.2
government	0.1	0.2	−0.1	−0.1
open	−0.4	−0.4	0.2	0.3
Ø	0.0	0.0	0.0	0.0

Ø,t,d	−0.6	0.2	1.4
t,d,r	−1.0	1.6	−1.0
d,r,t	−0.5	−0.1	0.8
r,t,k	−3.6	0.3	0.3
t,k,g	−0.2	0.1	1.2
k,g,o	0.3	0.6	0.9
g,o,Ø	−0.5	−0.9	0.1

2-max p	−0.2	1.6	1.4
	0.3	0.6	1.2

Apply 3 **filters** of size 3

3	1	2	−3
−1	2	1	−3
1	1	−1	1

1	0	0	1
1	0	−1	−1
0	1	0	1

1	−1	2	−1
1	0	−1	3
0	2	2	1

CNN Hyperparameter

dilation = 2

Ø	0.0	0.0	0.0	0.0
tentative	0.2	0.1	−0.3	0.4
deal	0.5	0.2	−0.3	−0.1
reached	−0.1	−0.3	−0.2	0.4
to	0.3	−0.3	0.1	0.1
keep	0.2	−0.3	0.4	0.2
government	0.1	0.2	−0.1	−0.1
open	−0.4	−0.4	0.2	0.3
Ø	0.0	0.0	0.0	0.0

Apply 3 **filters** of size 3

3	1	2	−3
−1	2	1	−3
1	1	−1	1

1	0	0	1
1	0	−1	−1
0	1	0	1

1	−1	2	−1
1	0	−1	3
0	2	2	1

Ø,t,d	−0.6	0.2	1.4
t,d,r	−1.0	1.6	−1.0
d,r,t	−0.5	−0.1	0.8
r,t,k	−3.6	0.3	0.3
t,k,g	−0.2	0.1	1.2
k,g,o	0.3	0.6	0.9
g,o,Ø	−0.5	−0.9	0.1

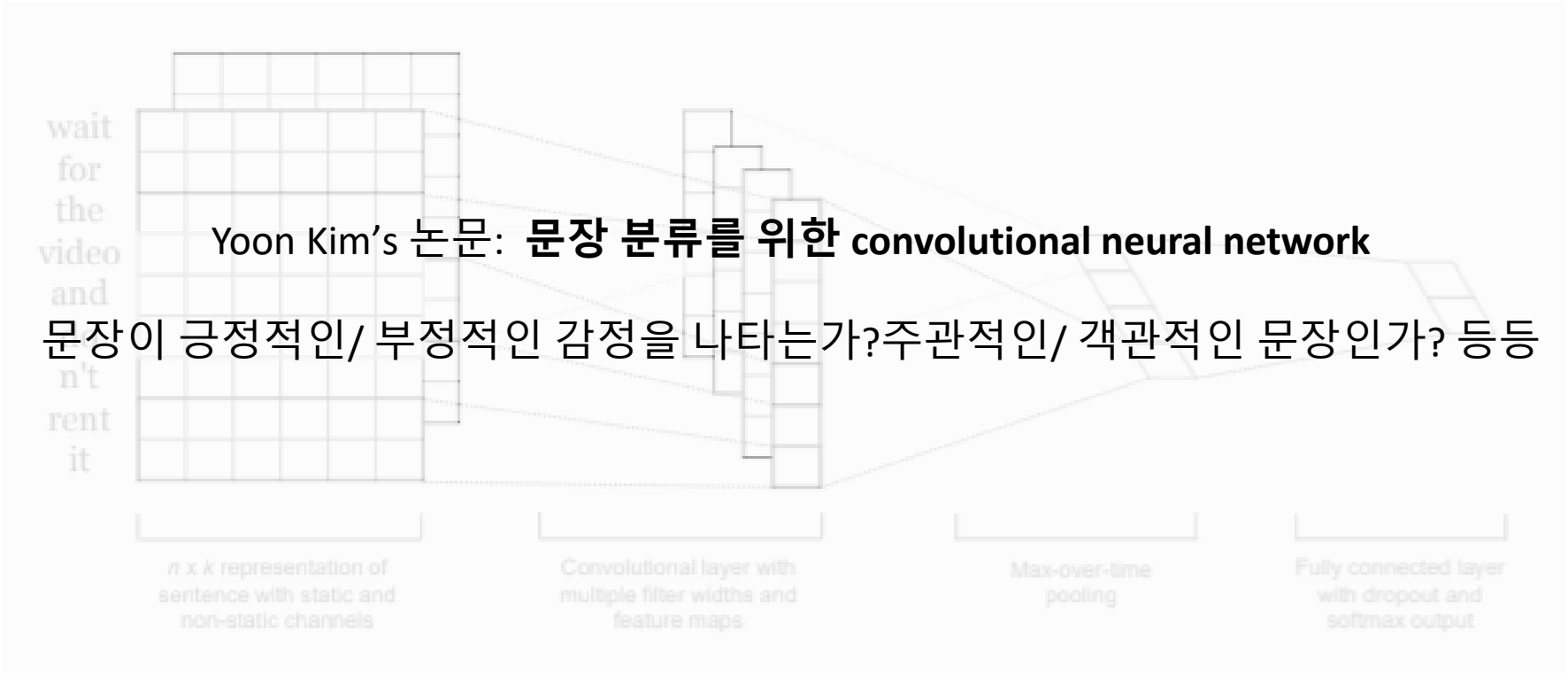
1,3,5	0.3	0.0
2,4,6		
3,5,7		

2	3	1
1	−1	−1
3	1	0

1	3	1
1	−1	−1
3	1	−1

02

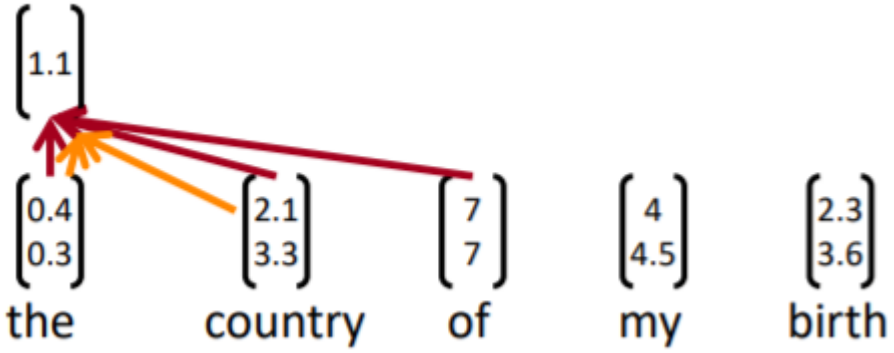
Simple CNN for Sentence Classification



Yoon Kim’s 논문: **문장 분류를 위한 convolutional neural network**

문장이 긍정적인/ 부정적인 감정을 나타는가?주관적인/ 객관적인 문장인가? 등등

Input vector & filter



Word vectors: $\mathbf{x}_i \in \mathbb{R}^k$

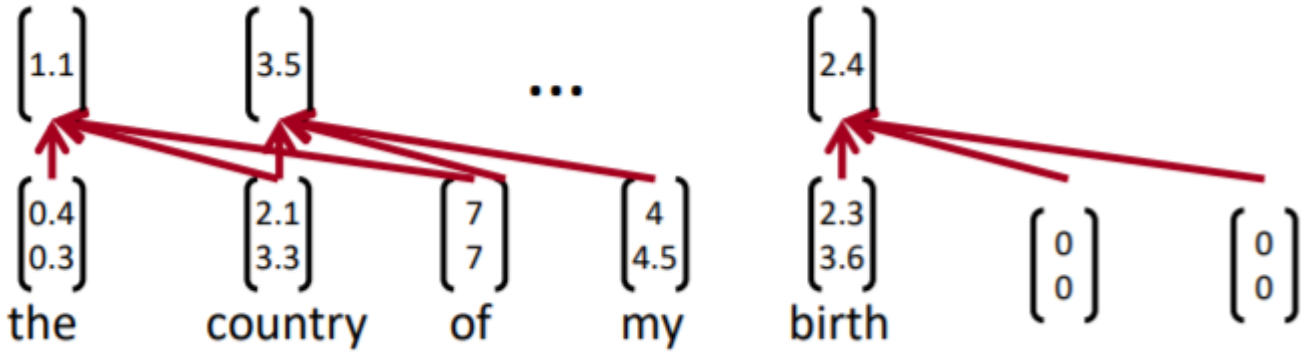
Sentence: $\mathbf{x}_{1:n} = \mathbf{x}_1 \oplus x_2 \oplus \cdots \oplus \mathbf{x}_n$

Concatenation of words in range: $\mathbf{x}_{i:i+j}$

Convolutional filter: $\mathbf{w} \in \mathbb{R}^{hk}$

문장은 n개의 단어 벡터들을 concatenate한 것
필터의 너비는 단어 벡터의 크기에 의해 결정 → k
필터 사이즈는 window 수에 따라서 결정 → h
(주황색- size 2 빨간색 - size3)

Padding

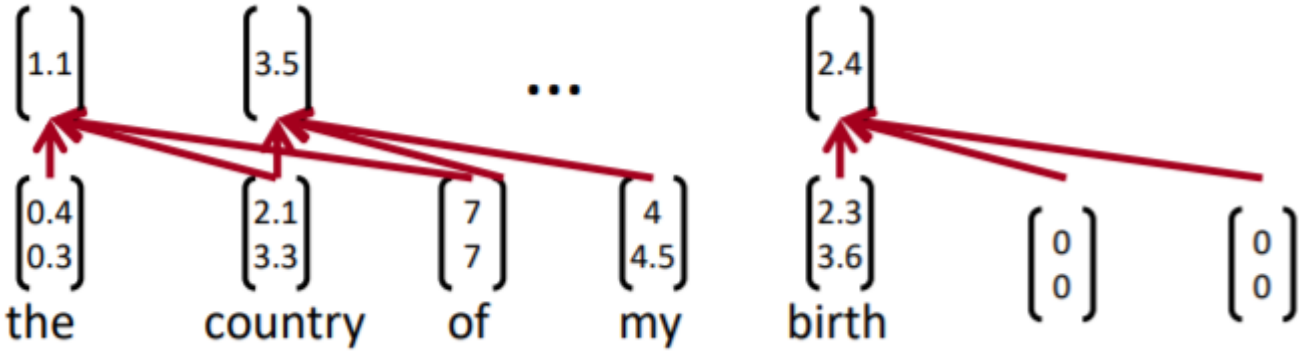


이미지 분류와 달리 패딩을 양쪽에 넣지 않고 오른쪽 끝에만 넣는다

Feature map

sub-sentence 벡터와 필터와의 내적 결과물

$$c_i = f(\mathbf{w}^T \mathbf{x}_{i:i+h-1} + b)$$



n = 문장에 들어가는 단어 수 h = kernel size

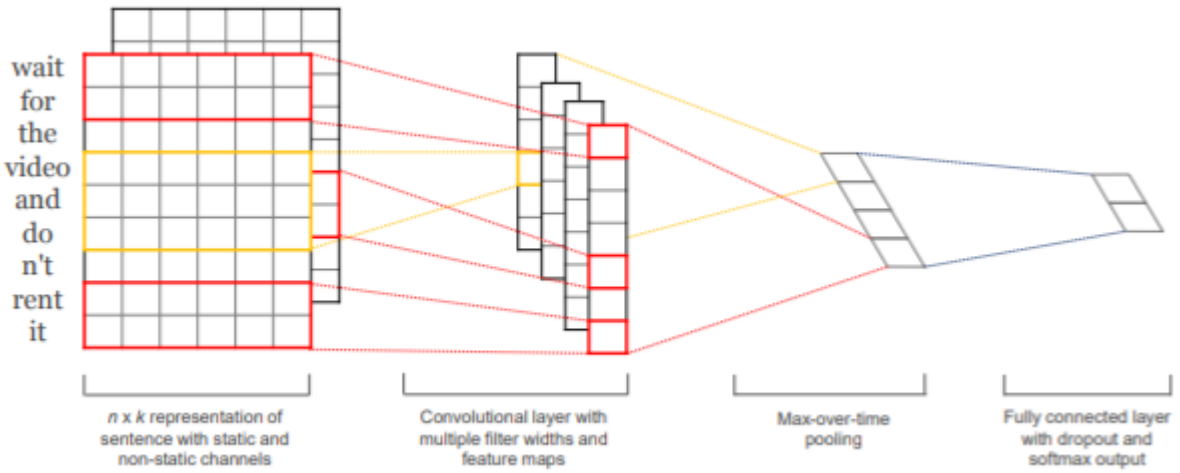
총 n-h+1개의 feature map이 만들어진다

$$\mathbf{c} = [c_1, c_2, \ldots, c_{n-h+1}] \in \mathbb{R}^{n-h+1}$$

Max pooling layer

capture most important activation

$$\mathbf{c} = [c_1, c_2, \ldots, c_{n-h+1}] \qquad \hat{c} = \max\{\mathbf{c}\}$$



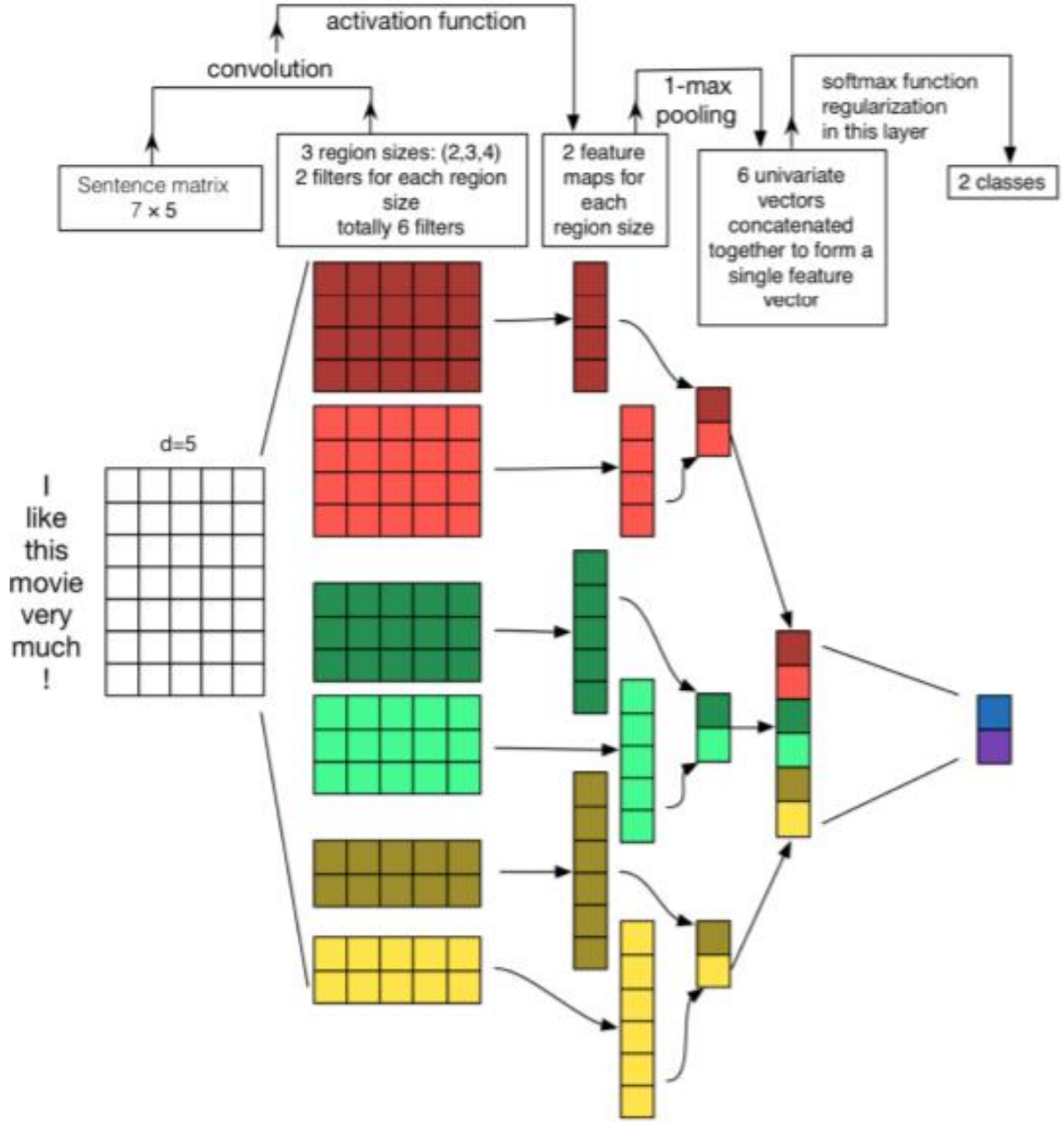
다양한 filter weights와 다른 window size를 적용 → 여러개의 feature map

각 feature map에 대해 max pooling 진행

03

Multi-channel input idea

fine-tuning한 것, 안한 것 모두를 이용해 최고의 조합을 만들어 냄


activation function
convolution
1-max pooling
softmax function regularization in this layer
Sentence matrix 7 × 5
3 region sizes: (2,3,4) 2 filters for each region size totally 6 filters
2 feature maps for each region size
6 univariate vectors concatenated together to form a single feature vector
2 classes
d=5
I
like
this
movie
very
much
!

final feature vector

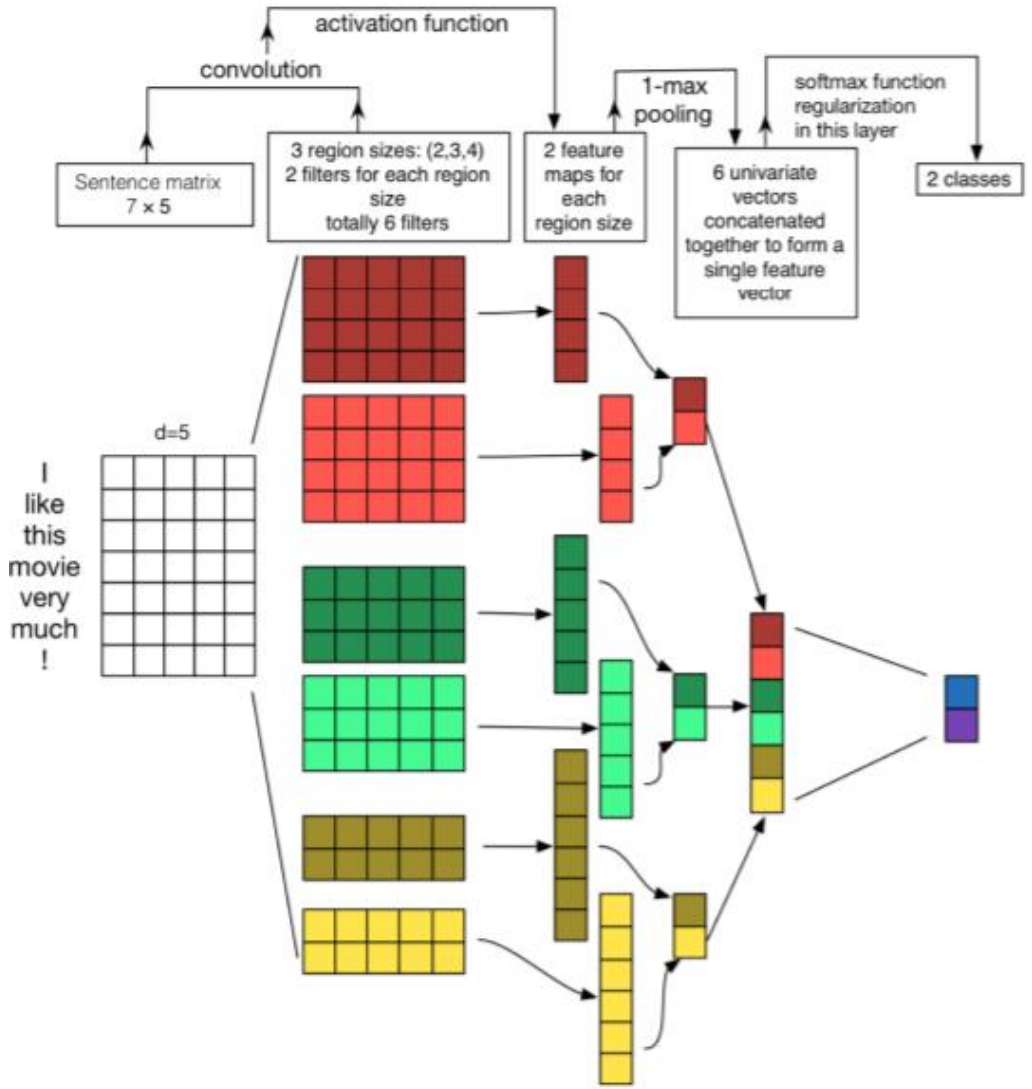
$$\mathbf{z} = [\hat{c}_1, \ldots, \hat{c}_m]$$

m개의 필터를 사용 → m개의 max-pooling 값들을 concatenate한 형태

Classification

$$y = softmax\left(W^{(S)}z + b\right)$$

final feature vector와 weight 연산
→ 소프트맥스 함수에 적용

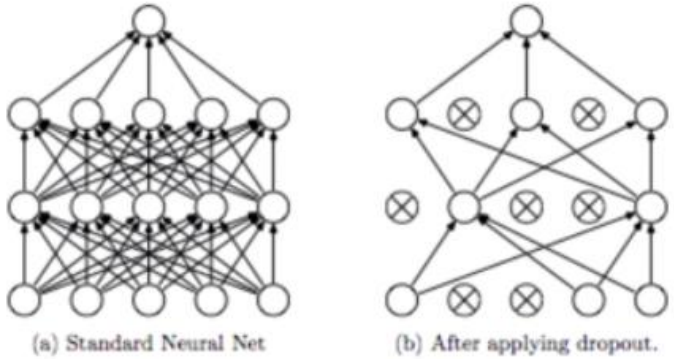


Regularization

Dropout

$$y = softmax\left(W^{(S)}(r \circ z) + b\right)$$

bern(p)를 따르는 vector r을 내적해서
몇 개의 노드를 없애는 것
→ 오버피팅 방지



(a) Standard Neural Net (b) After applying dropout.

constrain l2 norms of weight vectors

If $\|W_{c\cdot}^{(S)}\| > s$

then rescale it so that: $\|W_{c\cdot}^{(S)}\| = s$

weight vector의 크기를 제한하는 것

03

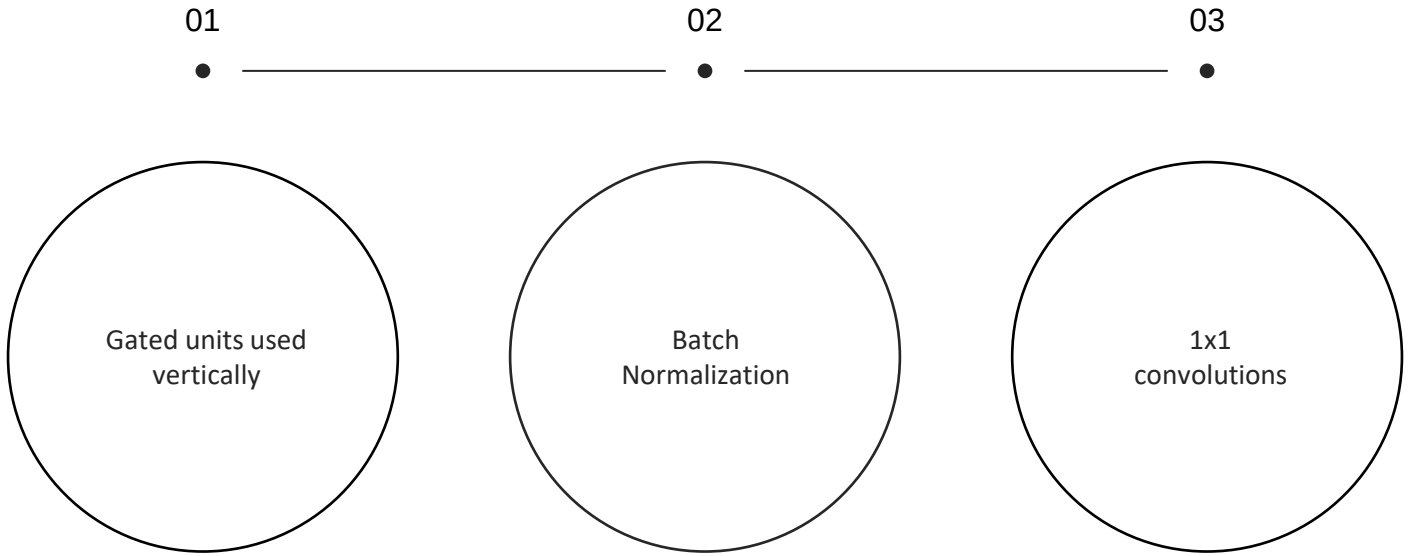
실제 사용한 hyperparameter

Nonlinearity	ReLU
Window filter sizes	h = 3, 4, 5
Each filter size has 100 feature maps	
Dropout	p = 0.5
L2 constraint s	s = 3
Mini batch size for SGD training	50
Word vectors	pre-trained with word2vec
	k = 300

Model comparison

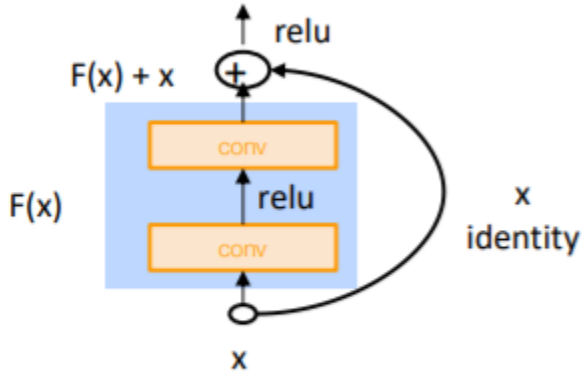
Bag of vectors	**Window Model**
- 간단한 분류문제를 해결하는데 사용하는 모델 - 각 단어 벡터의 평균을 넣는 방식 - Relu 활성화 함수를 넣으면 성능이 좋아짐	- NER POS 등 - NER은 문장 안에서 해당 단어가 location, person, organization 등 어떤 개체명에 해당하는지 분류하는 방법론 - 예를 들어 '디카프리오가 나온 영화를 틀어줘' 문장에서 디카프리오를 사람으로 인식하는 것을 목표로 함
CNNs	**Recurrent Neural Networks**
- 분류에 좋은 성능을 보임 - 문장의 길이가 짧은 경우에 zero padding 이 필요	- 왼쪽에서 오른쪽 방향으로 문장의 모든 단어들을 훑는 방식 - sequence tagging/ 문장에서 다음에 올 단어 예측 등에서 좋은 성능을 보임 - CNN에 비해 속도가 느림

복잡한 Convolutional architecture를 살펴보기 전에..
몇 가지 개념을 보고 가자!

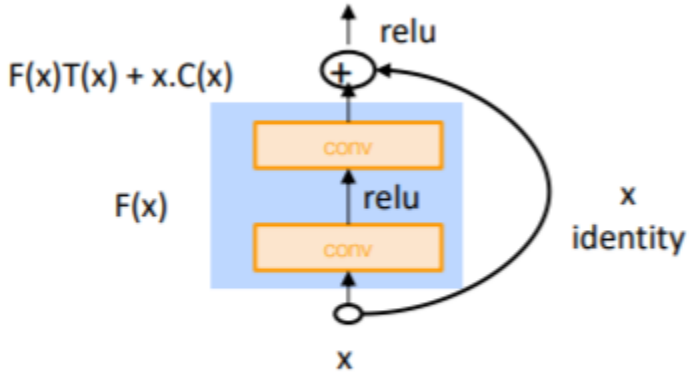


Gated units used vertically

LSTM에서 살펴본 Gate → 모델에서 자주 사용하는 일반적인 아이디어를 의미
(ex. Input Gate, Forget Gate 등등)
그 외에도 수직적 구조의 Gate가 존재한다!

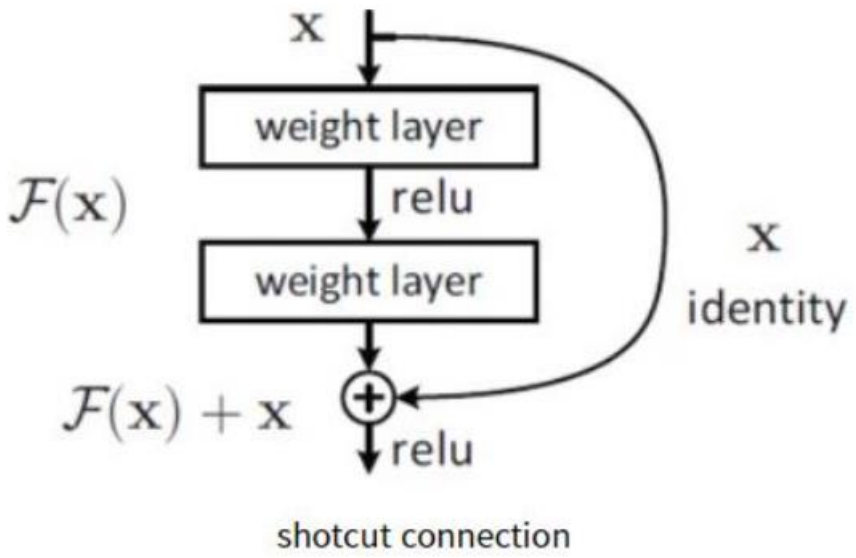


Residual block
(He et al. ECCV 2016)



Highway block
(Srivistava et al. NeurIPS 2015)

Residual block의 중요한 특징들 복습하고 가자!

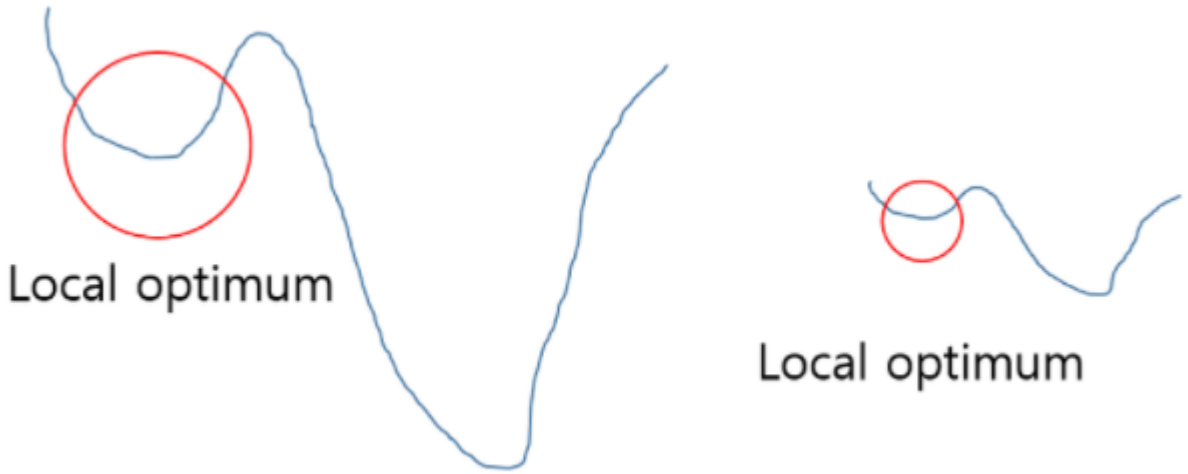


1. 다음 layer로 갈 수 있도록 지름길을 만들어 준 것!
2. F(x)+x을 최소화하는 방식으로 학습한다. 미분 값이 1이 최소치로 보장되기 때문에 gradient vanishing 문제를 해결해 준다
3. F(X) + x 더하기 연산을 수행할 때 차원을 맞춰줘야 한다 → downsample 사용(x의 차원을 F(x)에 맞추어 바꿔주는 작업)

Batch normalization

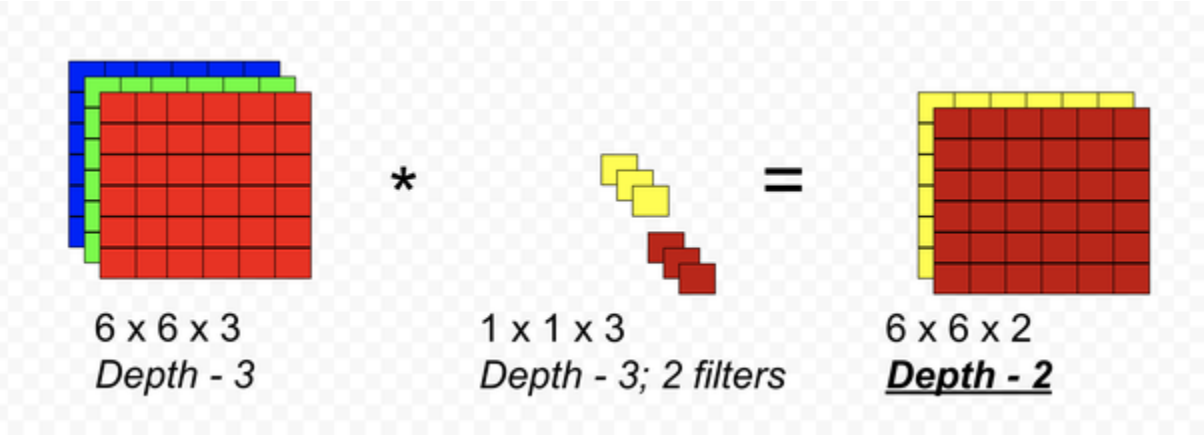
활성화 함수를 통과하기 이전에 평균 0과 unit variance로 정규화하는 과정

1. Gradient vanishing/ Gradient exploding 문제 해결 → 각 output을 정규화하여 training 과정 자체를 안정화하자
2. 내부 공변량 문제 해결 → 네트워크의 각 층을 지날 때 마다 데이터의 분포가 바뀌는 현상
3. Local optimum에 빠지는 가능성을 줄여줌 → 아래 그림에서 local minimum 근처에 있을 때 global minimum 지점을 찾지 못하고 머물러 있는 경우가 발생 → 정규화를 통해 local minimum에 빠질 가능성을 낮춰 줌



(좌) Normalization 적용 전 / (우) Normalization 적용 후

1x1 convolutions



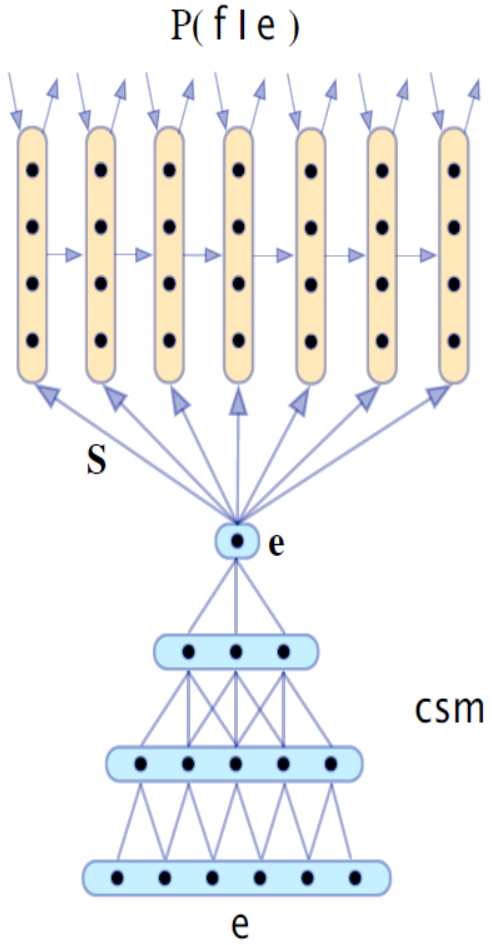
1. 같은 위치의 각 채널의 값으로 FC linear layer를 만들 수 있음
2. 필터의 수를 조정해서 채널의 수를 조절할 수 있다(3개의 channel → 2개의 channel)
3. 적은 수의 파라미터를 통해서 뉴럴 네트워크의 layer를 추가할 수 있다
   (FC layer는 많은 수의 파라미터를 필요로 함) → 연산량 절감
4. 연산 후에 활성화 함수 RELU를 사용하면 추가로 non-linearity 얻을 수 있음

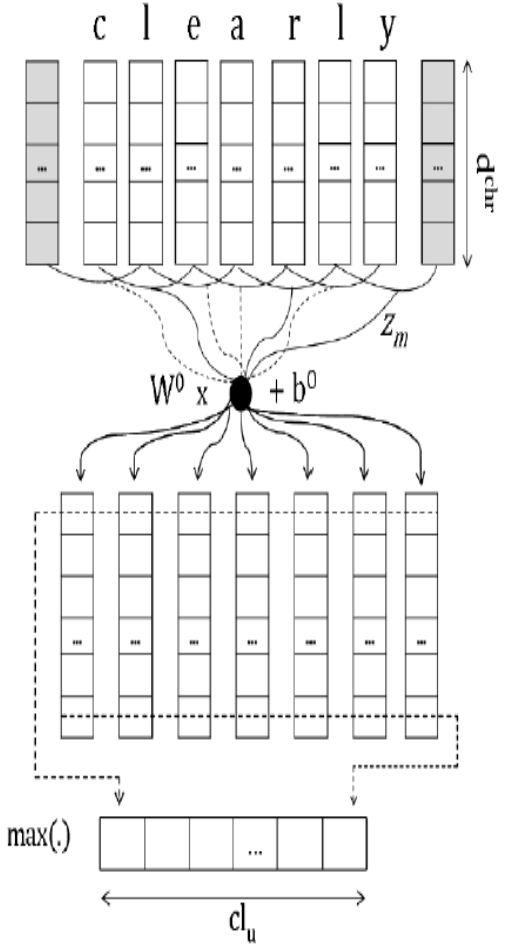
03

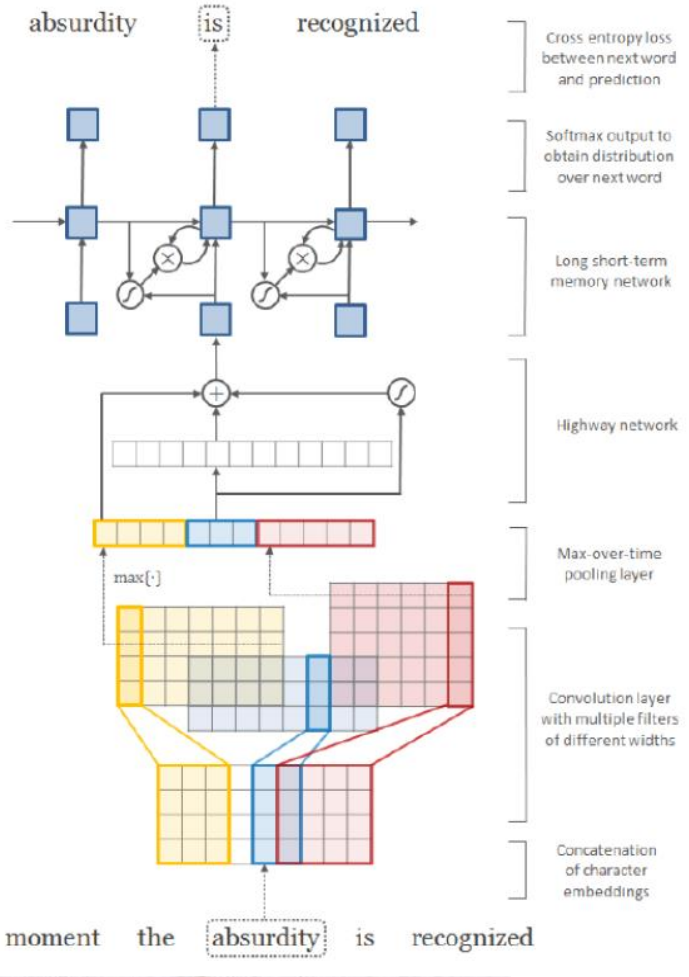
CNN Potpourri

CNN potpourri

- "Recurrent Continuous Translation Models" Kalchbrenner and Blunsom (2013)
- Learning Character-level Representations for Part-of-Speech Tagging Dos Santos and Zadrozny (2014)
- Character-Aware Neural Language Models (Kim, Jernite, Sontag, and Rush 2015)







04

Deep CNN for Sentence Classification

06

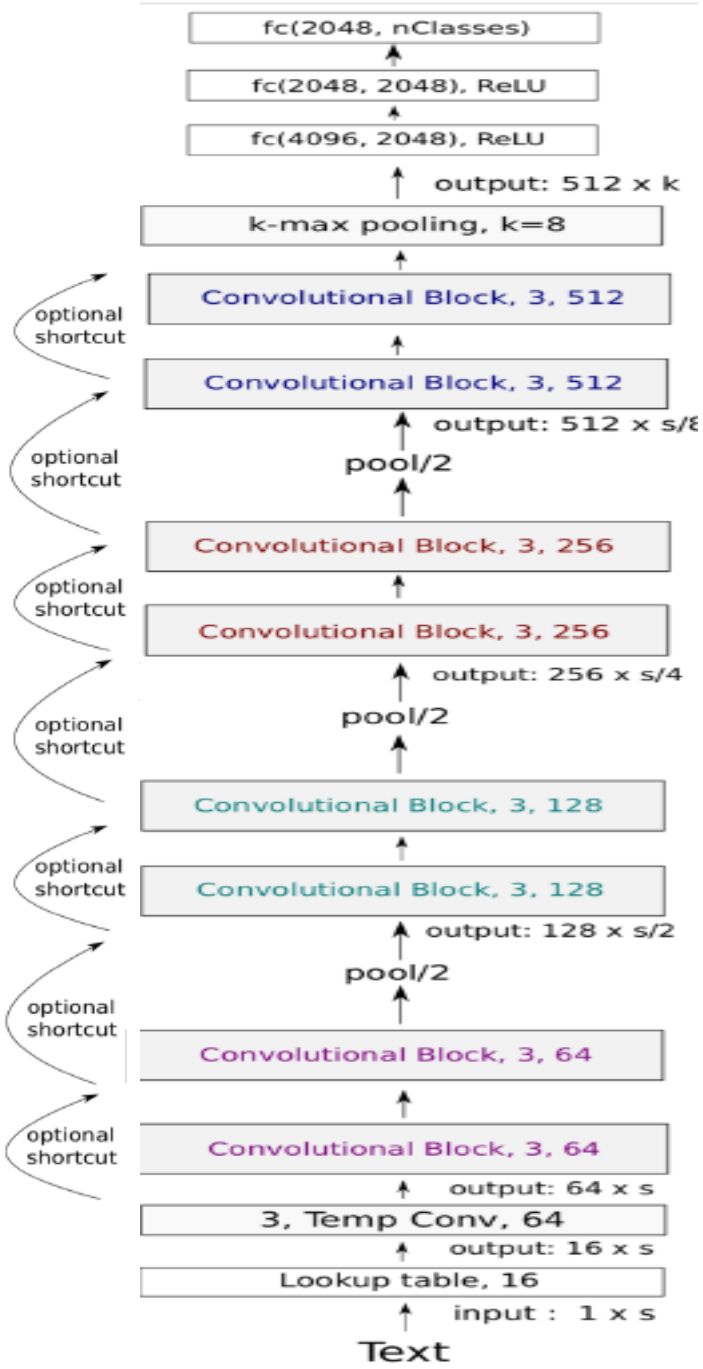
VD- CNN architecture

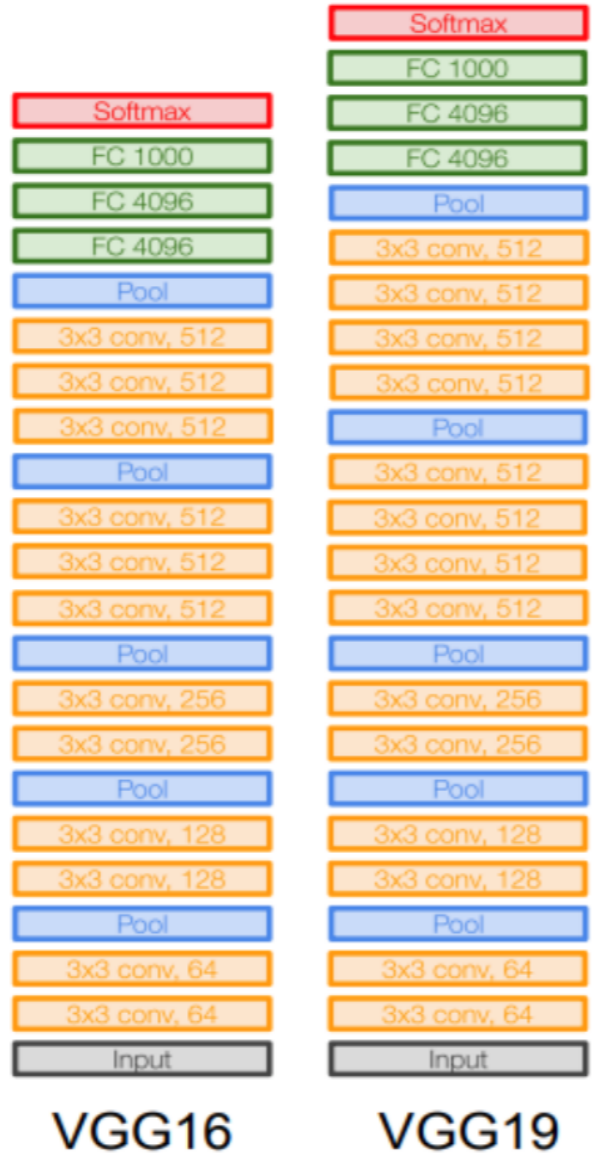
- 기존의 딥러닝 NLP이 아닌
VGGnet , ResNet 과 유사한 구조

- Input은 CNN에서 image size 맞춘 것
document size을 맞춰준다. ex) 1024 ch
i) Longer than document size
➔ Truncate
ii) Shorter than document size
➔ Padding
- Convolutional Block

- K-max Pooling

- Fully Connected Layers





06

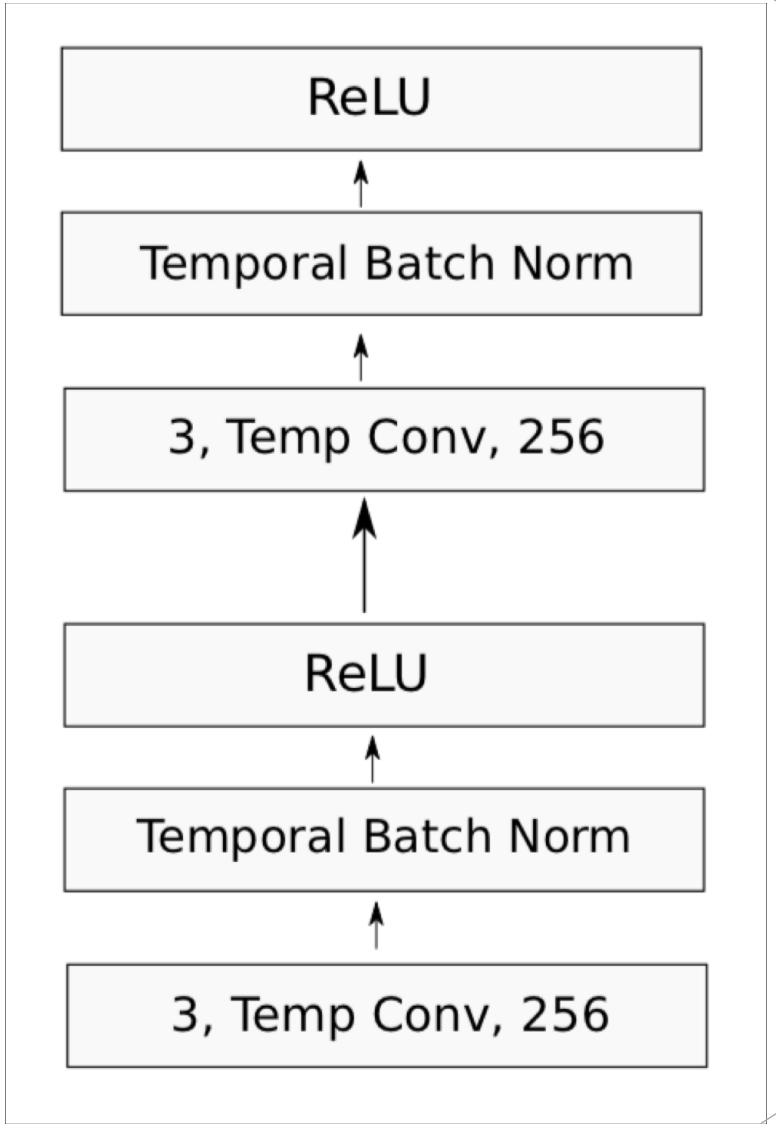
VD- CNN architecture

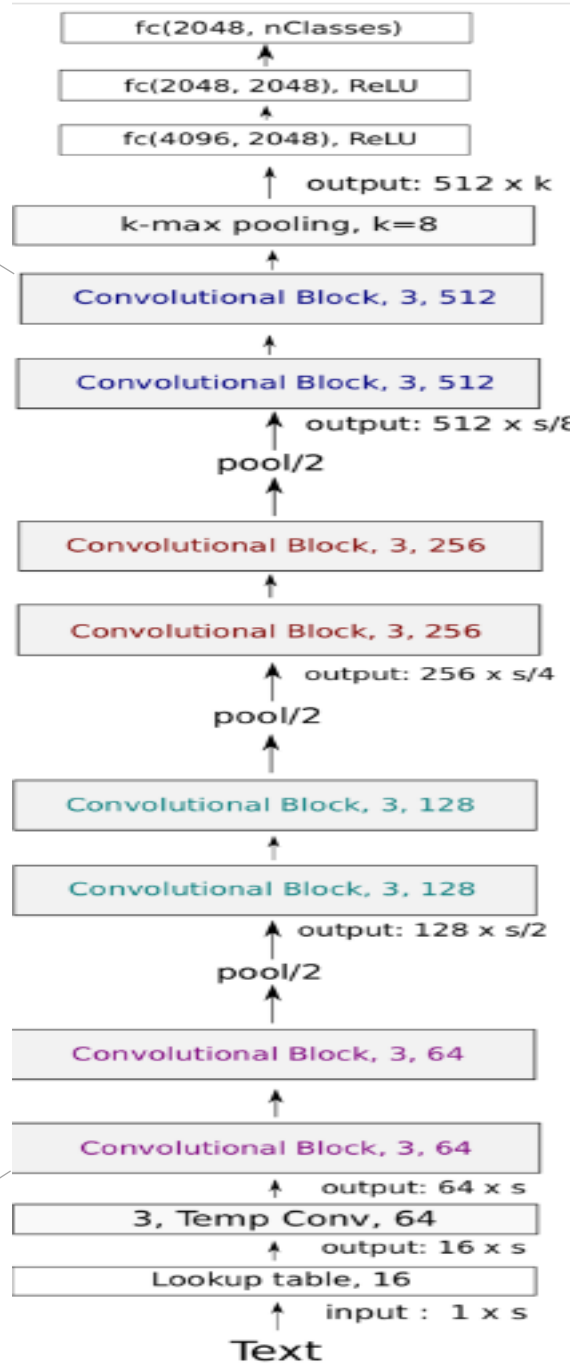
- Convolutional block

- Convolution Block
= 2 x (convolutional layer -> batch -> ReLU nonlinearity)

- Convolution size =3

- Padding (∵차원 보존)





Experiment with bigger dataset

- Deeper Networks → Better result

But, 막연히 layer를 늘린다고 좋은 성능을 주는 것은 아님

- Shrinking Method

→ MaxPooling 추천

- Overall Message

" We can build a good text classification system using ConvNets."

Corpus:	AG	Sogou	DBP.	Yelp P.	Yelp F.	Yah. A.	Amz. F.	Amz. P.
Method	n-TFIDF	n-TFIDF	n-TFIDF	ngrams	Conv	Conv+RNN	Conv	Conv
Author	[Zhang]	[Zhang]	[Zhang]	[Zhang]	[Zhang]	[Xiao]	[Zhang]	[Zhang]
Error	7.64	2.81	1.31	4.36	37.95*	28.26	40.43*	4.93*
[Yang]	-	-	-	-	-	24.2	36.4	-

Table 4: Best published results from previous work. Zhang et al. (2015) best results use a Thesaurus data augmentation technique (marked with an *). Yang et al. (2016)'s hierarchical methods is particularly

Depth	Pooling	AG	Sogou	DBP.	Yelp P.	Yelp F.	Yah. A.	Amz. F.	Amz. P.
9	Convolution	10.17	4.22	1.64	5.01	37.63	28.10	38.52	4.94
9	KMaxPooling	9.83	3.58	1.56	5.27	38.04	28.24	39.19	5.69
9	MaxPooling	9.17	3.70	1.35	4.88	36.73	27.60	37.95	4.70
17	Convolution	9.29	3.94	1.42	4.96	36.10	27.35	37.50	4.53
17	KMaxPooling	9.39	3.51	1.61	5.05	37.41	28.25	38.81	5.43
17	MaxPooling	8.88	3.54	1.40	4.50	36.07	27.51	37.39	4.41
29	Convolution	9.36	3.61	1.36	4.35	**35.28**	27.17	37.58	**4.28**
29	KMaxPooling	**8.67**	**3.18**	1.41	4.63	37.00	27.16	38.39	4.94
29	MaxPooling	8.73	3.36	**1.29**	**4.28**	35.74	**26.57**	**37.00**	4.31

Table 5: Testing error of our models on the 8 data sets. No data preprocessing or augmentation is used.

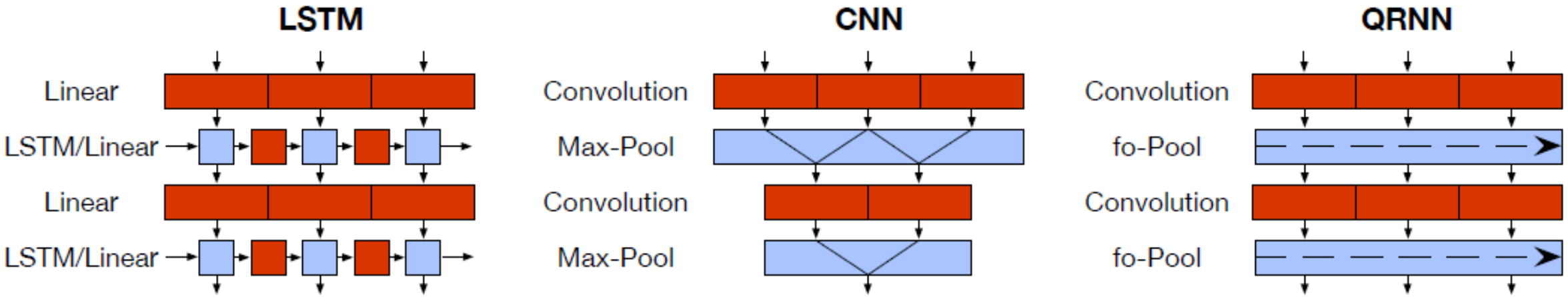
05

Quasi-recurrent Neural Networks

08

Quasi- Recurrent Neural Networks

- RNN & CNN의 장점을 섞어놓은 architecture



- RNN처럼 각 element의 순서에 의존
- CNN처럼 time step과 mini batch를 통한 병렬처리 가능

Quasi- Recurrent Neural Networks

- 시간에 따른 병렬처리를 위한 Convolution

$$\mathbf{z}_t = \tanh(\mathbf{W}_z^1 \mathbf{x}_{t-1} + \mathbf{W}_z^2 \mathbf{x}_t)$$
$$\mathbf{f}_t = \sigma(\mathbf{W}_f^1 \mathbf{x}_{t-1} + \mathbf{W}_f^2 \mathbf{x}_t)$$
$$\mathbf{o}_t = \sigma(\mathbf{W}_o^1 \mathbf{x}_{t-1} + \mathbf{W}_o^2 \mathbf{x}_t).$$

→

$$\mathbf{Z} = \tanh(\mathbf{W}_z * \mathbf{X})$$
$$\mathbf{F} = \sigma(\mathbf{W}_f * \mathbf{X})$$
$$\mathbf{O} = \sigma(\mathbf{W}_o * \mathbf{X}),$$

- 미래의 token을 예측해야 하므로, 미래의 정보에 의존하면 X

→ t 번째 time step에서는 t번째 element 이전만 볼 수 있도록 해야 함

Quasi- Recurrent Neural Networks

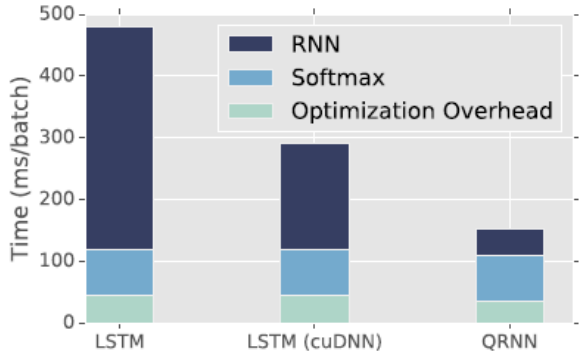
보통 더 빠르고 성능이 더 좋음

- Experiment

Model	Parameters	Validation	Test
LSTM (medium) (Zaremba et al., 2014)	20M	86.2	82.7
Variational LSTM (medium) (Gal & Ghahramani, 2016)	20M	81.9	79.7
LSTM with CharCNN embeddings (Kim et al., 2016)	19M	–	78.9
Zoneout + Variational LSTM (medium) (Merity et al., 2016)	20M	84.4	80.6
Our models			
LSTM (medium)	20M	85.7	82.0
QRNN (medium)	18M	82.9	79.9
QRNN + zoneout ($p = 0.1$) (medium)	18M	82.1	78.3

Language Model

Better



		Sequence length				
		32	64	128	256	512
Batch size	8	5.5x	8.8x	11.0x	12.4x	16.9x
	16	5.5x	6.7x	7.8x	8.3x	10.8x
	32	4.2x	4.5x	4.9x	4.9x	6.4x
	64	3.0x	3.0x	3.0x	3.0x	3.7x
	128	2.1x	1.9x	2.0x	2.0x	2.4x
	256	1.4x	1.4x	1.3x	1.3x	1.3x

Faster

Quasi- Recurrent Neural Networks

- Experiment

Model	Time / Epoch (s)	Test Acc (%)
NBSVM-bi (Wang & Manning, 2012)	–	91.2
2 layer sequential BoW CNN (Johnson & Zhang, 2014)	–	92.3
Ensemble of RNNs and NB-SVM (Mesnil et al., 2014)	–	92.6
2-layer LSTM (Longpre et al., 2016)	–	87.6
Residual 2-layer bi-LSTM (Longpre et al., 2016)	–	90.1
Our models		
Densely-connected 4-layer LSTM (cuDNN optimized)	480	90.9
Densely-connected 4-layer QRNN	150	91.4
Densely-connected 4-layer QRNN with $k = 4$	160	91.1

Table 1: Accuracy comparison on the IMDb binary sentiment classification task. All of our models use 256 units per layer; all layers other than the first layer, whose filter width may vary, use filter width $k = 2$. Train times are reported on a single NVIDIA K40 GPU. We exclude semi-supervised models that conduct additional training on the unlabeled portion of the dataset.

Sentiment Classification

Model	Train Time	BLEU (TED.tst2014)
Word-level LSTM w/attn (Ranzato et al., 2016)	–	20.2
Word-level CNN w/attn, input feeding (Wiseman & Rush, 2016)	–	24.0
Our models		
Char-level 4-layer LSTM	4.2 hrs/epoch	16.53
Char-level 4-layer QRNN with $k = 6$	1.0 hrs/epoch	19.41

Table 3: Translation performance, measured by BLEU, and train speed in hours per epoch, for the IWSLT German-English spoken language translation task. All models were trained on in-domain data only, and use negative log-likelihood as the training criterion. Our models were trained for 10 epochs. The QRNN model uses $k = 2$ for all layers other than the first encoder layer.

Character-level Neural Machine Translation

Quasi- Recurrent Neural Networks

- Limitation

1) Longer dependency를 가진 모델들에서 문제 발생
   character level LM에서는 잘 작동하지 않음
2) LSTM 만큼의 좋은 성능을 위해서는 보통 deeper network을 필요
   But, deeper network을 사용할 때도, 더 빠름
   효과적으로, depth를 true recurrence의 대안으로 사용함

발표 들어주셔서

감사합니다