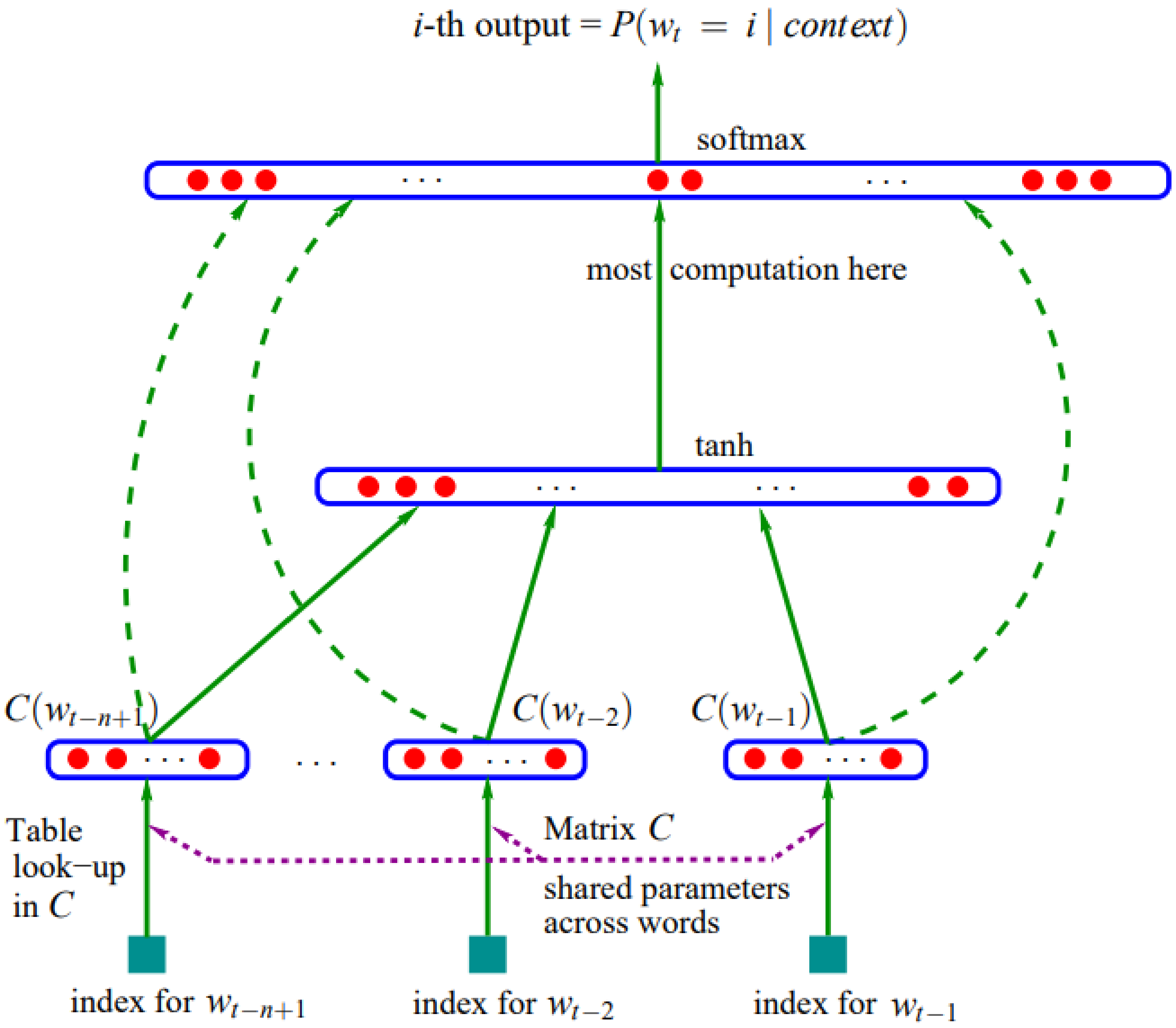
Recurrent Neural Network Based Language Model
+
Character-Aware Neural Language Model

AI LAB
임영수

Statistical Language Model : Language domain에 제한적이다.
(몇가지 가정을 전제로 한다!)
N-gram statistic model(가장 널리 쓰이는 모델)조차도 가정이 쪼금 필요하다.

여태까지 등장한 model들은 특정 training data에서만 잘 동작하고, 복잡하며,
N-gram에 비해 tiny improvement만 있었다 + 잘 쓰지도 않는다

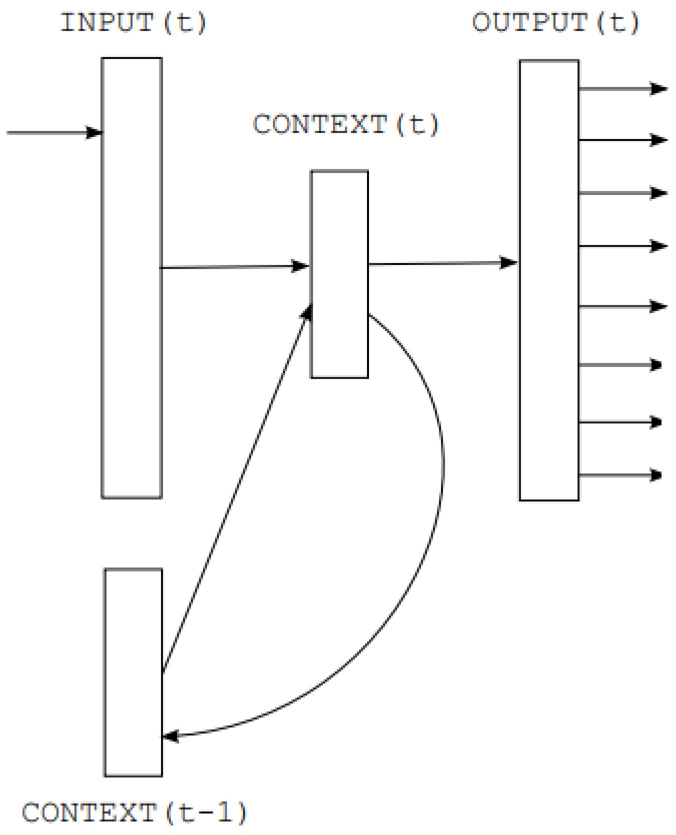


Bengio's Model
(Feedforward Neural Network
with **fixed length** context)

-> 5~10번째 전의 단어들만으로
다음 단어를 판단한다

-> context가 가변적이지 못하다!

RNN을 써보자!
RNN Language Model



$$x(t) = w(t) + s(t-1) \quad (1)$$

$$s_j(t) = f\left(\sum_i x_i(t) u_{ji}\right) \quad (2)$$

$$y_k(t) = g\left(\sum_j s_j(t) v_{kj}\right) \quad (3)$$

where $f(z)$ is sigmoid activation function:

$$f(z) = \frac{1}{1+e^{-z}} \quad (4)$$

and $g(z)$ is softmax function:

$$g(z_m) = \frac{e^{z_m}}{\sum_k e^{z_k}} \quad (5)$$

$$\mathrm{error}(t) = \mathrm{desired}(t) - y(t) \quad (6)$$

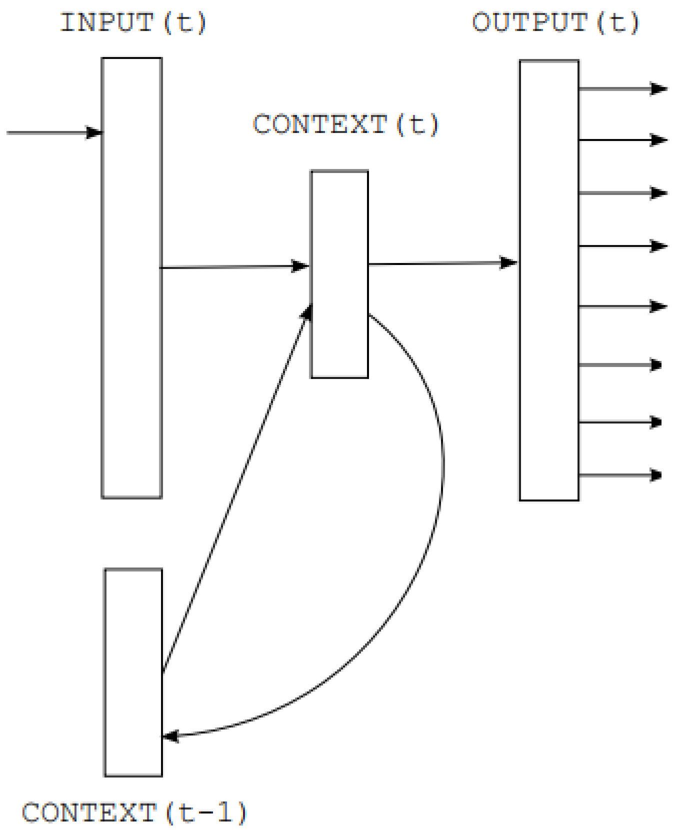
x – input layer (1-of-N encoding vector)
s – hidden layer (or context or state)
y – output layer
w – word vector

w의 size – vocabulary size(3만~20만)
s의 size – 30~500
(data 크기에 영향을 받는다)

Weight – standard backpropagation

Parameter의 개수가 줄어듬
(FNN – layer/hidden size, context-length
RNN – hidden layer size 하나만!)

Hidden layer가 컸는데도
Overtrain이 잘 안됨!(띠용)



$$x(t) = w(t) + s(t-1) \quad (1)$$

$$s_j(t) = f\left(\sum_i x_i(t)u_{ji}\right) \quad (2)$$

$$y_k(t) = g\left(\sum_j s_j(t)v_{kj}\right) \quad (3)$$

where $f(z)$ is sigmoid activation function:

$$f(z) = \frac{1}{1+e^{-z}} \quad (4)$$

and $g(z)$ is softmax function:

$$g(z_m) = \frac{e^{z_m}}{\sum_k e^{z_k}} \quad (5)$$

$$\mathrm{error}(t) = \mathrm{desired}(t) - y(t) \quad (6)$$

Statistical Model에서는
test data에서의 model update를 하지 않았다
(사람이름 등등에서 문제 발생!)

-> **Dynamic model**
Long term memory가 전체 synapse 안에 상주한다
(test pharse에서도 학습이 된다!)

Traning phase에서는 매 epoch마다 update
Dynamic model은 test data를 이용해 1번만!

전혀 Optimal하지 않음
근데 효과가 좋음;

Word-probabilities

$$P(w_i(t+1)|w(t), s(t-1)) = \begin{cases} \frac{y_{rare}(t)}{C_{rare}} & \text{if } w_i(t+1) \text{ is rare,} \\ y_i(t) & \text{otherwise} \end{cases} \quad (7)$$

Optimization(성능을 올려보자)

Special Rare Token!
-training data의 단어들 사이에서 rare한 단어들을 하나의 special rare token으로 merge
(rare의 기준은 threshold)

C_{rare} : threshold를 못 넘은 단어의 개수

모든 rare 단어들의 확률은
Uniform distribution

그래서 잘됨?

Table 1: *Performance of models on WSJ DEV set when increasing size of training data.*

Model	# words	PPL	WER
KN5 LM	200K	336	16.4
KN5 LM + RNN 90/2	200K	271	15.4
KN5 LM	1M	287	15.1
KN5 LM + RNN 90/2	1M	225	14.0
KN5 LM	6.4M	221	13.5
KN5 LM + RNN 250/5	6.4M	156	11.7

Table 2: *Comparison of various configurations of RNN LMs and combinations with backoff models while using 6.4M words in training data (WSJ DEV).*

	PPL		WER	
Model	RNN	RNN+KN	RNN	RNN+KN
KN5 - baseline	-	221	-	13.5
RNN 60/20	229	186	13.2	12.6
RNN 90/10	202	173	12.8	12.2
RNN 250/5	173	155	12.3	11.7
RNN 250/2	176	156	12.0	11.9
RNN 400/10	171	152	12.5	12.1
3xRNN static	151	143	11.6	11.3
3xRNN dynamic	128	121	11.3	11.1

ㅇㅇ 잘됨

(PPL – perplexity (낮을수록좋다)
WER – Word Error Rate)

KN(Kneser-Ney smoothed n-gram)

Data의 수가 더 적었음에도 기존 모델보다 성능이 더 좋더라!

특수한 가정이 필요하지 않아서 쉽게 적용할 수 있다!

Voca 사이즈가 클수록 성능이 좋아진다!

짱짱

But!

Word-embedding(대표적으론 One-hot vector)에는 단어 정보가 담기지 않아요!
-> 빈도수가 적은 단어에 대한 PPL이 높아짐(성능 떡락)
-> 형태소가 많은 언어에서 특히 문제!

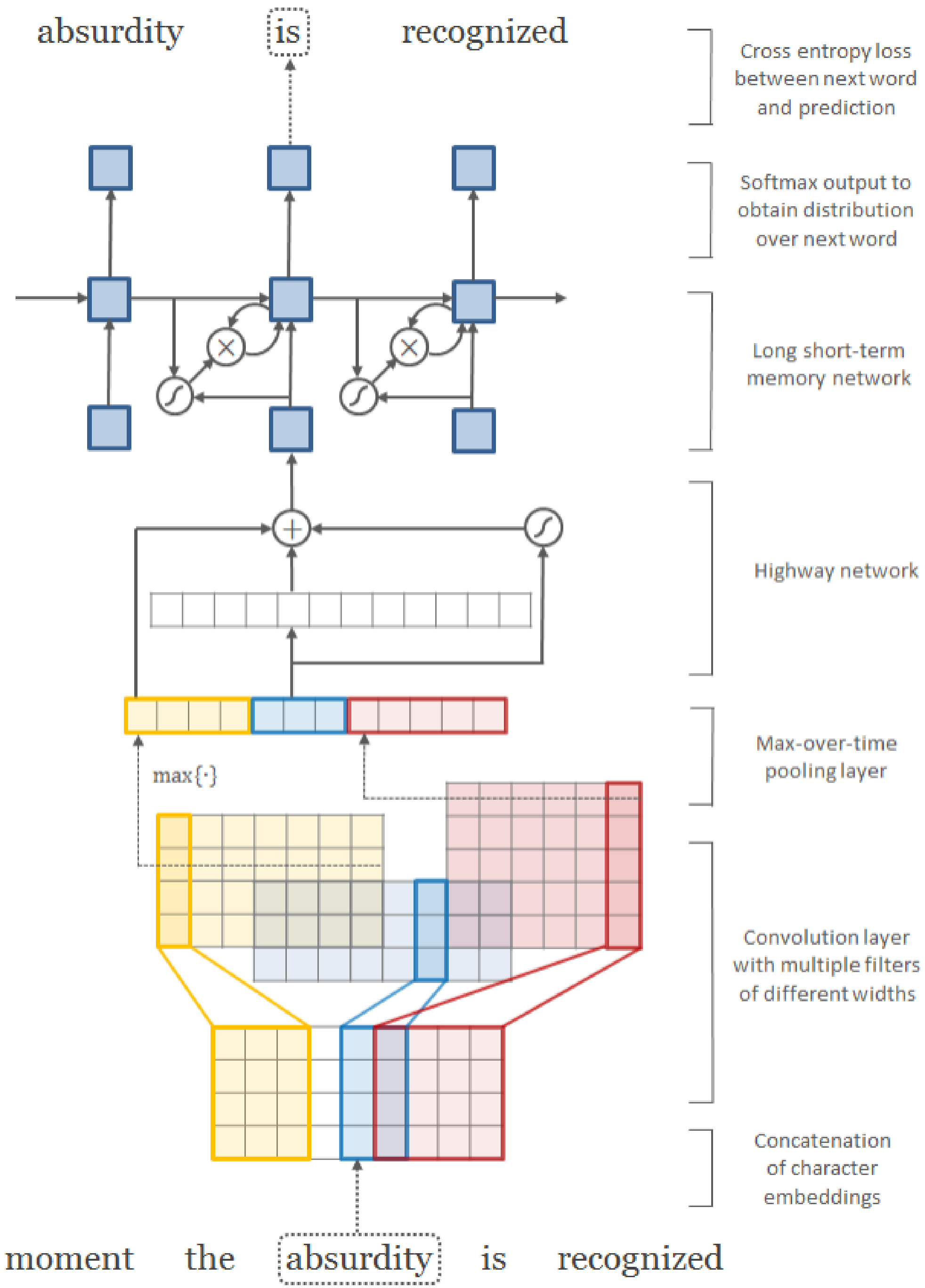
Character-Aware
Neural Language Model

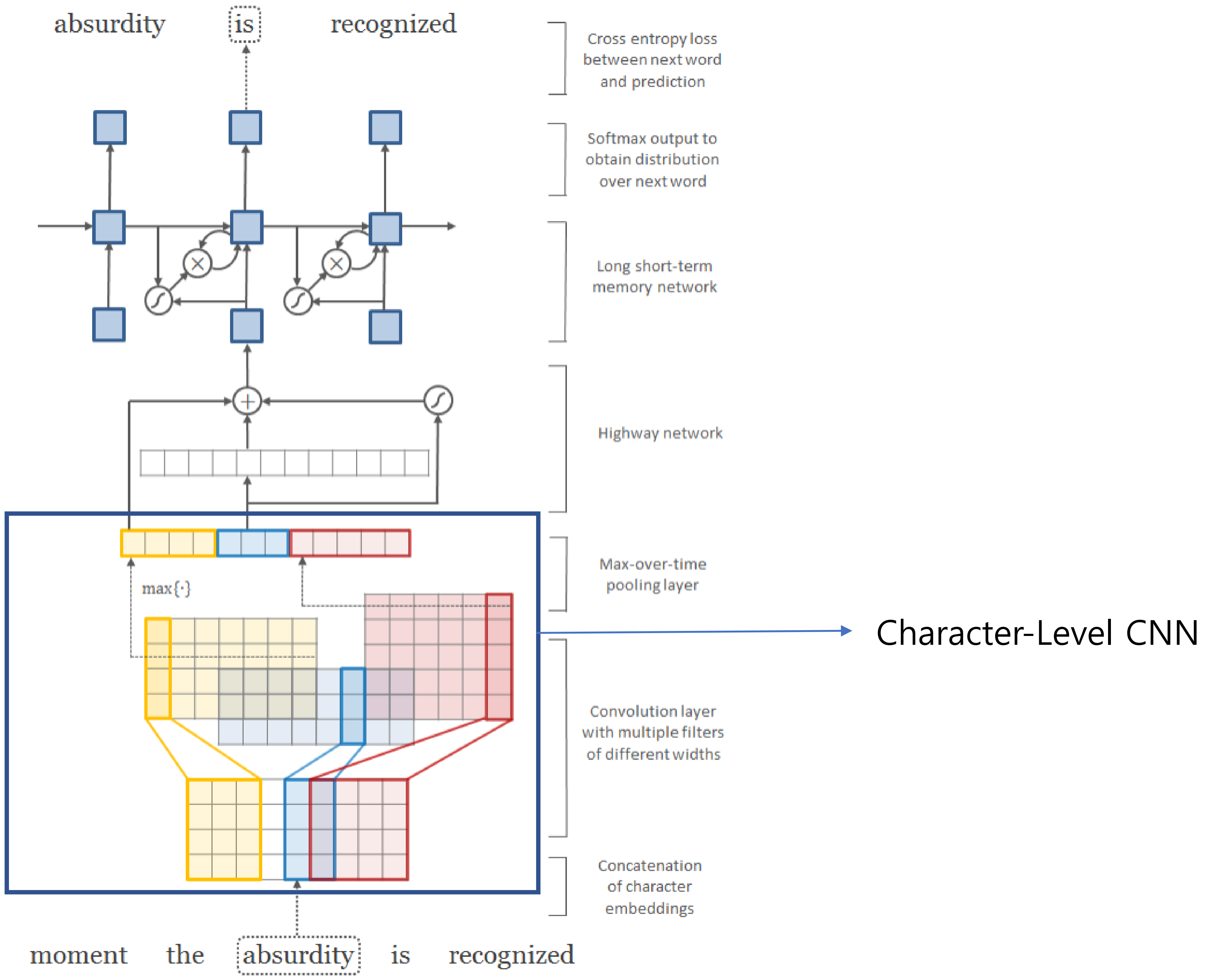
선요약 후설명

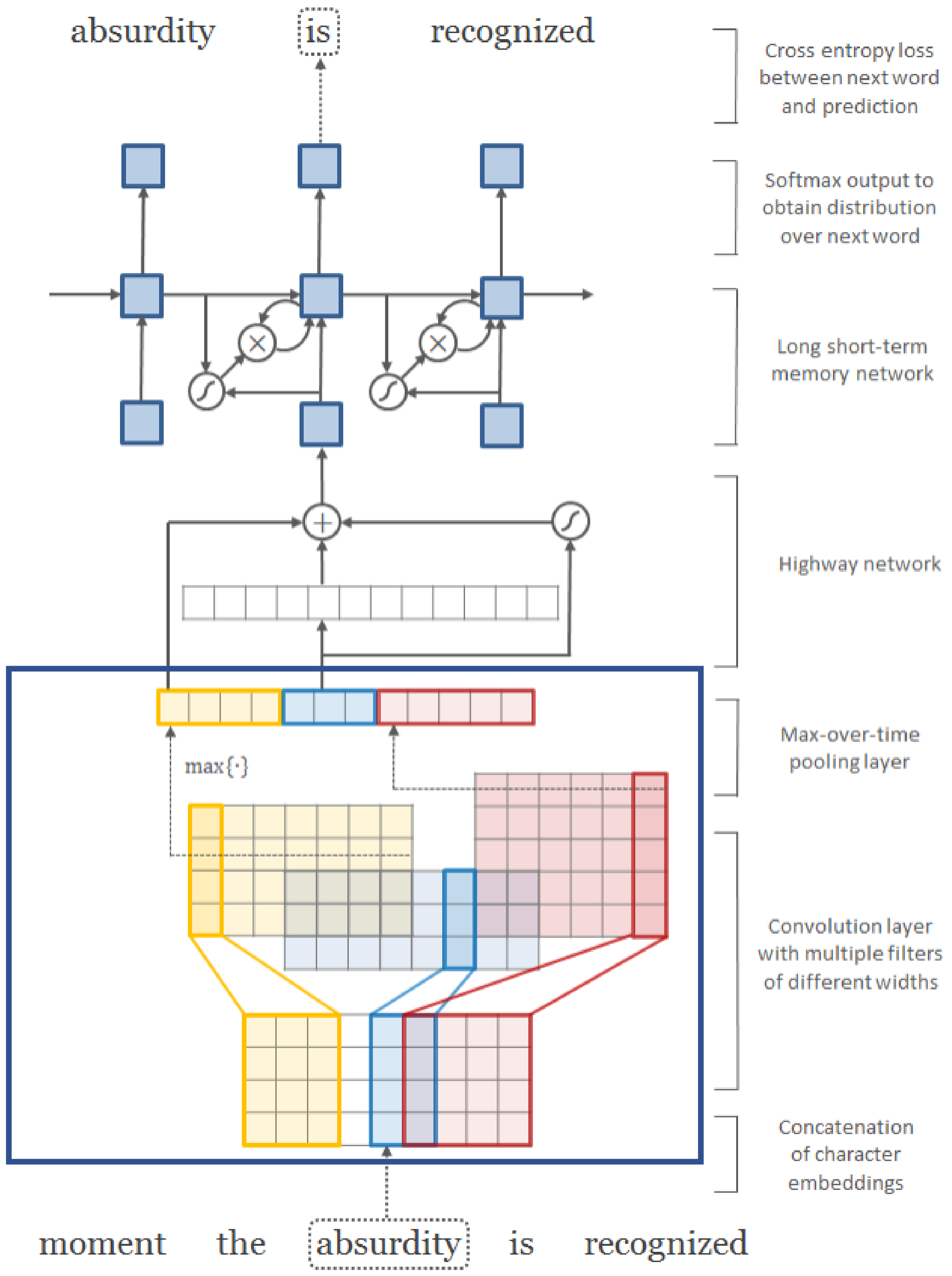
charCNN을 통한 subword information 학습을 시도해보았다!

그랬더니 parameter가 더 적어도 다른 모델보다 더 좋더라!

짱짱


absurdity
is
recognized
Cross entropy loss between next word and prediction
Softmax output to obtain distribution over next word
Long short-term memory network
Highway network
Max-over-time pooling layer
max{·}
Convolution layer with multiple filters of different widths
Concatenation of character embeddings
moment
the
absurdity
is
recognized


absurdity
is
recognized
Cross entropy loss between next word and prediction
Softmax output to obtain distribution over next word
Long short-term memory network
Highway network
Max-over-time pooling layer
max{·}
Convolution layer with multiple filters of different widths
Concatenation of character embeddings
moment
the
absurdity
is
recognized
Character-Level CNN



Character-Level CNN

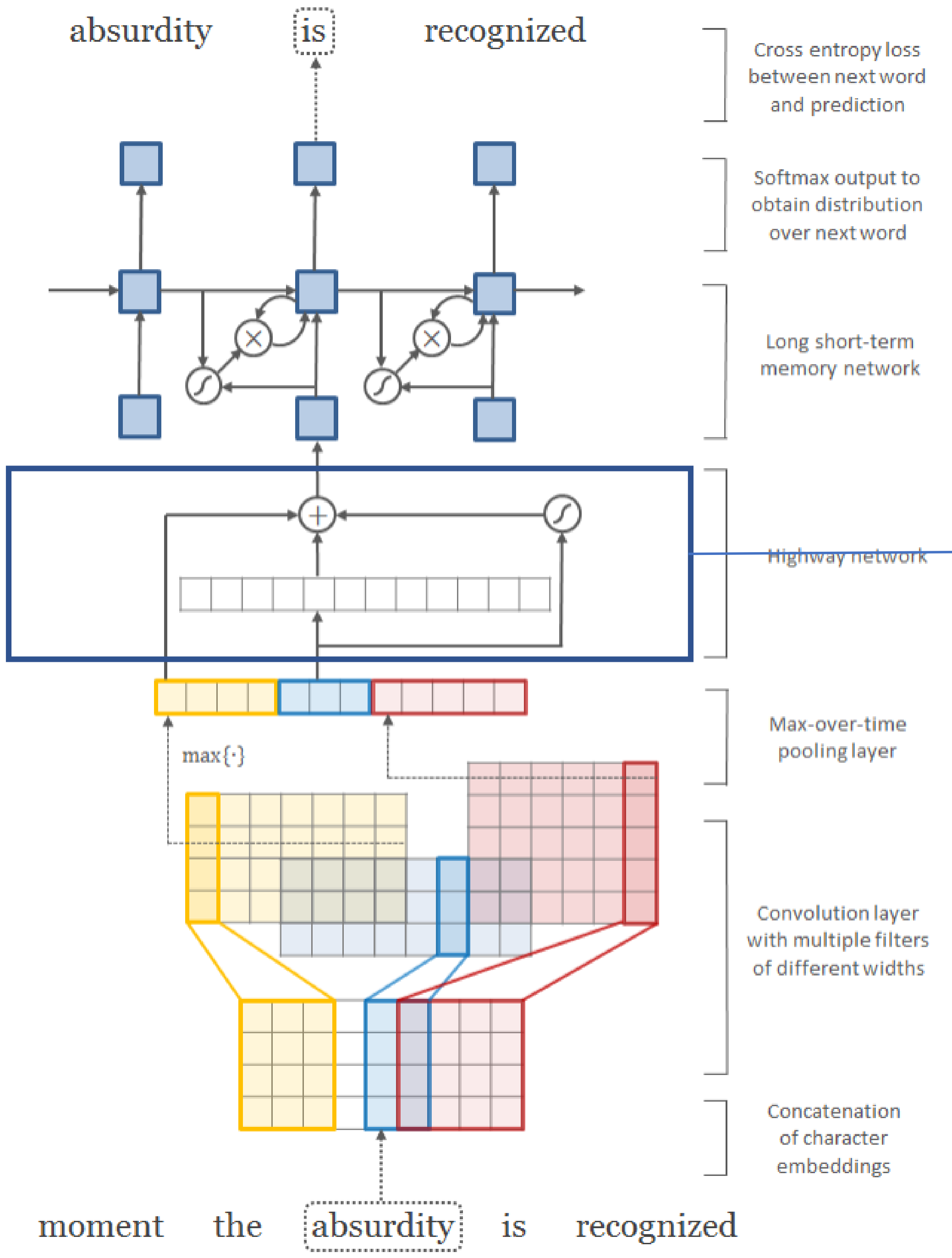
단어(char)로 이루어진 word를 CNN에 넣는다!
(word = sequence of character)

filter의 개수는 보통 [100,1000] 사이

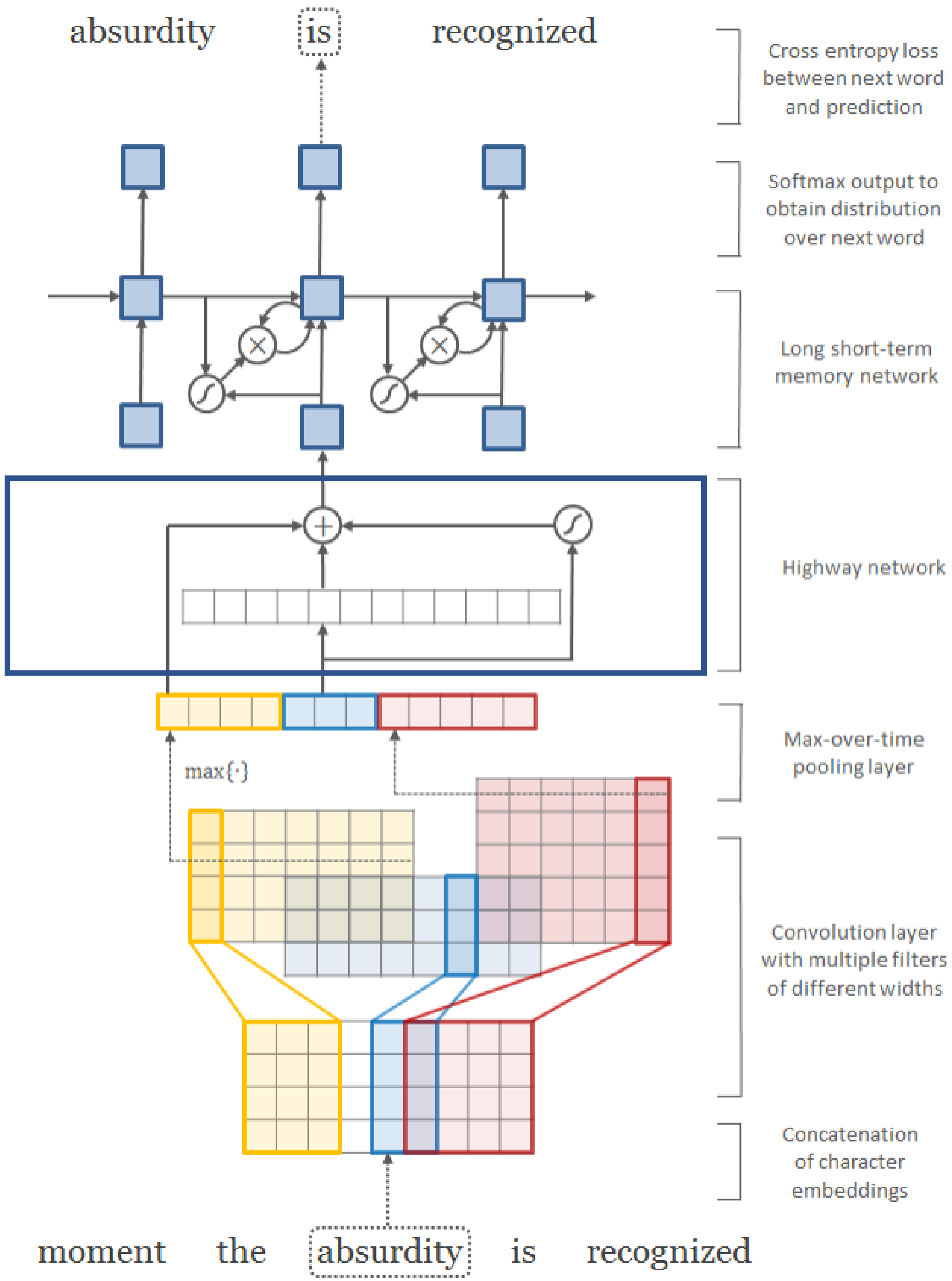
filtering 자체가 n-gram의 역할을 한다

결과물을 max-over-time pooling

-> Word embedding같은 효과!



Highway Network



Highway Network

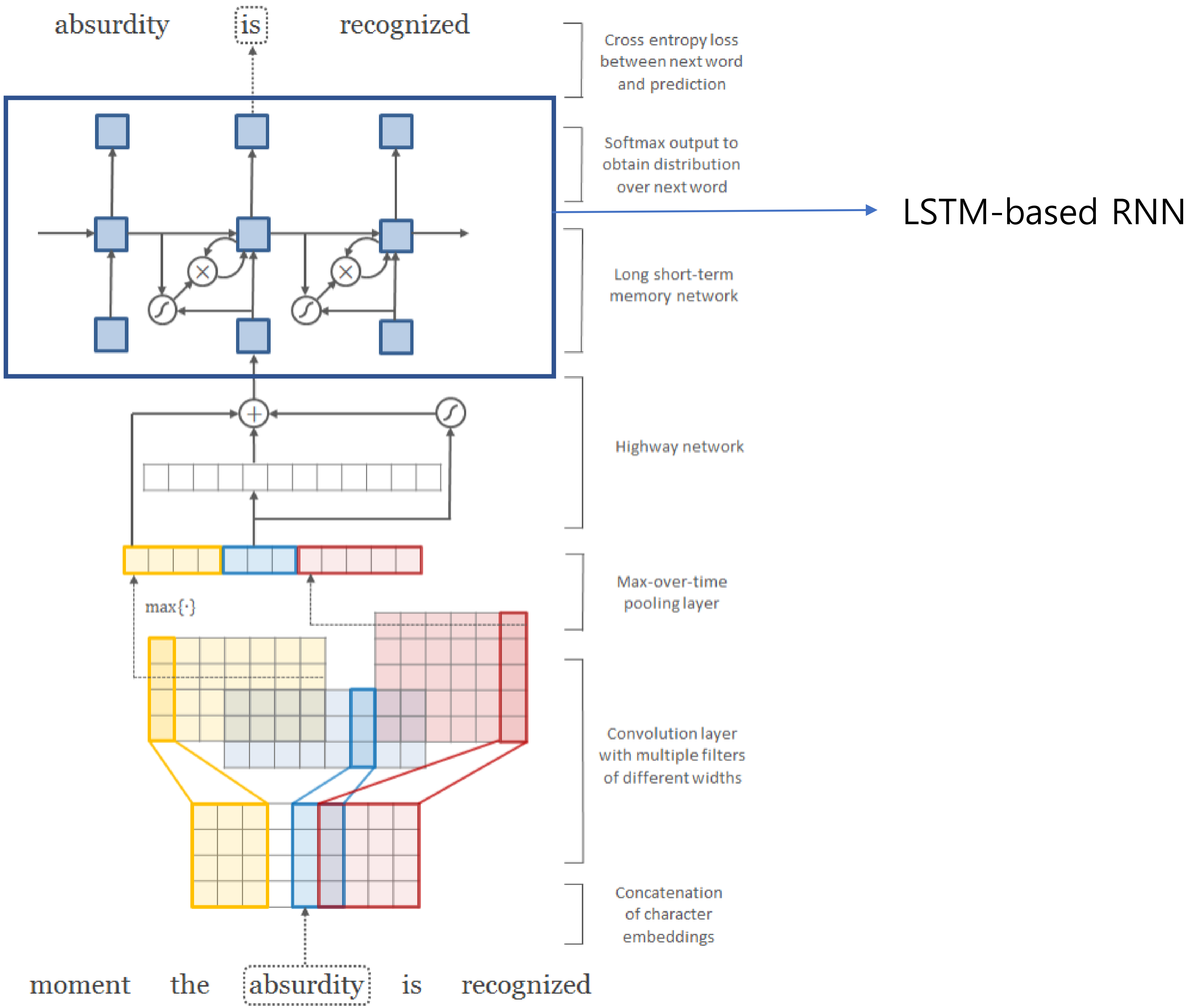
※Residual Network(ResNet)
모든 layer를 거치지 않고 일부 layer만을 거치는 방법

Model interaction에 사용
(원래는 MLP를 써봤는데 구리더라!)

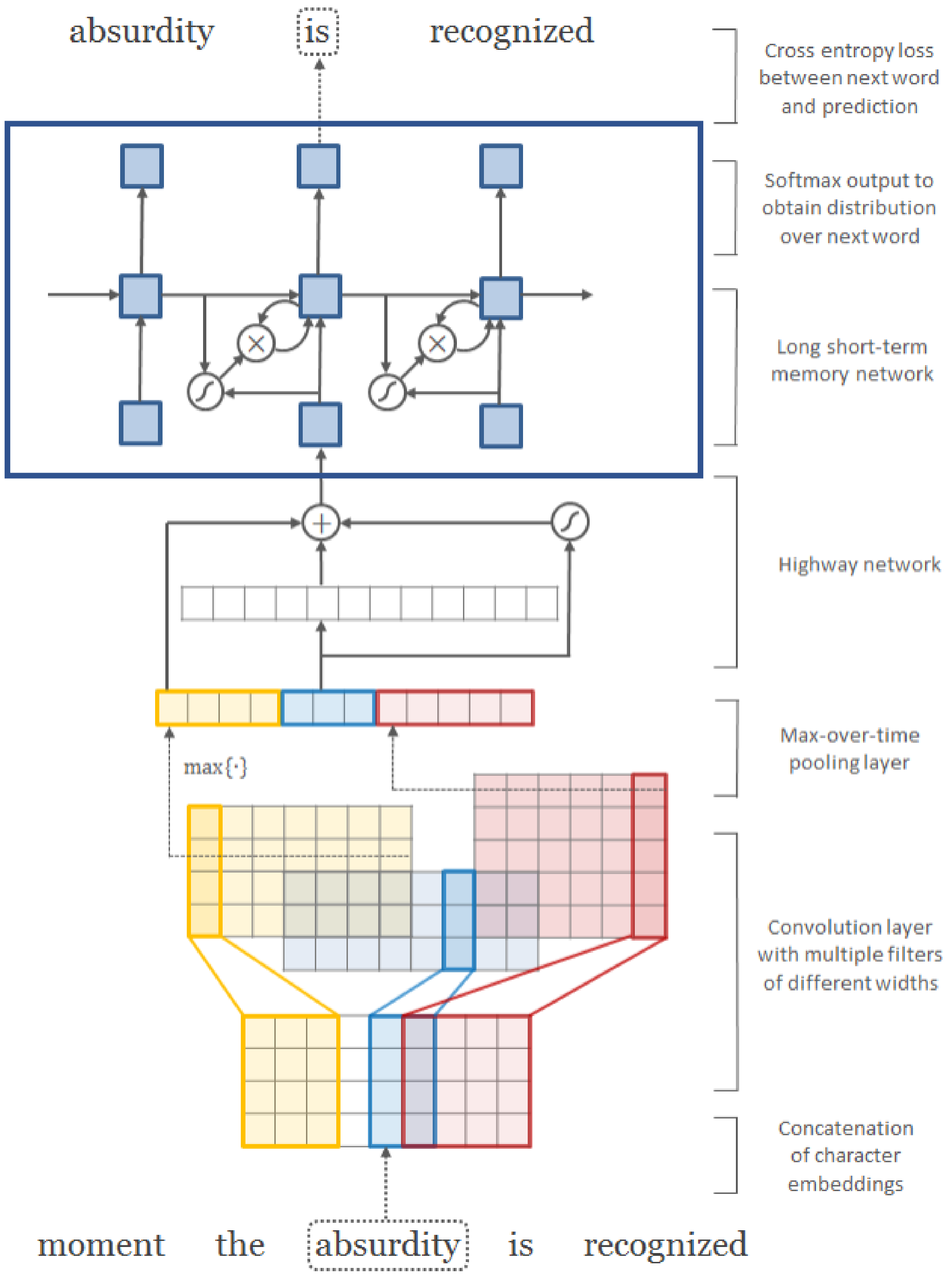
$$\mathbf{z} = \mathbf{t} \odot g(\mathbf{W}_H \mathbf{y} + \mathbf{b}_H) + (\mathbf{1} - \mathbf{t}) \odot \mathbf{y} \qquad (8)$$

$$\mathbf{t} = \sigma(\mathbf{W}_T \mathbf{y} + \mathbf{b}_T)$$

t = transform gate / (1-t) = carry gate
-> LSTM의 memory cell과 비슷하지 않은감?
-> 특정 dimension을 다음 network에
그대로 carry하는 역할을 한다



absurdity
is
recognized
Cross entropy loss between next word and prediction
Softmax output to obtain distribution over next word
LSTM-based RNN
Long short-term memory network
Highway network
Max-over-time pooling layer
max{·}
Convolution layer with multiple filters of different widths
Concatenation of character embeddings
moment
the
absurdity
is
recognized



요긴 그냥 특별할것없는 LSTM

Language Model이니까 당연히 RNN보단 LSTM

$$\Pr(w_{t+1} = j | w_{1:t}) = \frac{\exp(\mathbf{h}_t \cdot \mathbf{p}^j + q^j)}{\sum_{j' \in \mathcal{V}} \exp(\mathbf{h}_t \cdot \mathbf{p}^{j'} + q^{j'})} \quad (3)$$

Negative log-likelihood를 최소하는 방향으로 학습

$$NLL = -\sum_{t=1}^{T} \log \Pr(w_t | w_{1:t-1}) \quad (4)$$

그래서 잘됨?

	PPL	Size
LSTM-Word-Small	97.6	5 m
LSTM-Char-Small	92.3	5 m
LSTM-Word-Large	85.4	20 m
LSTM-Char-Large	78.9	19 m
KN-5 (Mikolov et al. 2012)	141.2	2 m
RNN† (Mikolov et al. 2012)	124.7	6 m
RNN-LDA† (Mikolov et al. 2012)	113.7	7 m
genCNN† (Wang et al. 2015)	116.4	8 m
FOFE-FNNLM† (Zhang et al. 2015)	108.0	6 m
Deep RNN (Pascanu et al. 2013)	107.5	6 m
Sum-Prod Net† (Cheng et al. 2014)	100.0	5 m
LSTM-1† (Zaremba et al. 2014)	82.7	20 m
LSTM-2† (Zaremba et al. 2014)	78.4	52 m

English Penn Treebank

		Small	Large
CNN	d	15	15
	w	$[1, 2, 3, 4, 5, 6]$	$[1, 2, 3, 4, 5, 6, 7]$
	h	$[25 \cdot w]$	$[\min\{200, 50 \cdot w\}]$
	f	tanh	tanh
Highway	l	1	2
	g	ReLU	ReLU
LSTM	l	2	2
	m	300	650

d = dimensionality of character embedding
w = filter width
h = number of filter matrix
f , g = 비선형 function
l = number of layers
m = number of hidden units

ㅇㅇ 잘됨

PPL – perplexity (낮을수록좋다)

KN(Kneser-Ney smoothed n-gram)

Word-level model들의 OOV들은
unknown token으로 replace되었음에도
우리꺼가 parameter 수(Size) 대비 성능이 쩐다!

	In Vocabulary					Out-of-Vocabulary		
	while	*his*	*you*	*richard*	*trading*	*computer-aided*	*misinformed*	*looooook*
LSTM-Word	*although*	*your*	*conservatives*	*jonathan*	*advertised*	–	–	–
	letting	*her*	*we*	*robert*	*advertising*	–	–	–
	though	*my*	*guys*	*neil*	*turnover*	–	–	–
	minute	*their*	*i*	*nancy*	*turnover*	–	–	–
LSTM-Char (before highway)	*chile*	*this*	*your*	*hard*	*heading*	*computer-guided*	*informed*	*look*
	whole	*hhs*	*young*	*rich*	*training*	*computerized*	*performed*	*cook*
	meanwhile	*is*	*four*	*richer*	*reading*	*disk-drive*	*transformed*	*looks*
	white	*has*	*youth*	*richter*	*leading*	*computer*	*inform*	*shook*
LSTM-Char (after highway)	*meanwhile*	*hhs*	*we*	*eduard*	*trade*	*computer-guided*	*informed*	*look*
	whole	*this*	*your*	*gerard*	*training*	*computer-driven*	*performed*	*looks*
	though	*their*	*doug*	*edward*	*traded*	*computerized*	*outperformed*	*looked*
	nevertheless	*your*	*i*	*carl*	*trader*	*computer*	*transformed*	*looking*

Word embedding 학습도 잘하더라!
OOV word도 학습하더라!
짱짱

charCNN을 쓰면 접두사/접미사/하이픈을 더 잘 구분한다

Corpus가 커질수록 성능도 더 좋아지더라

word embedding이랑 같이 쓰니까 성능이 더 안좋아지더라…?
(POS tagging이랑 NER이랑 같이쓴거는 성능 좋던데..?)
-> 우리쪽에선 쓸데없나봄

charCNN layer를 공유하다보니 GPU 사용효율성이 증가하더라

ㄲㅡㅌ