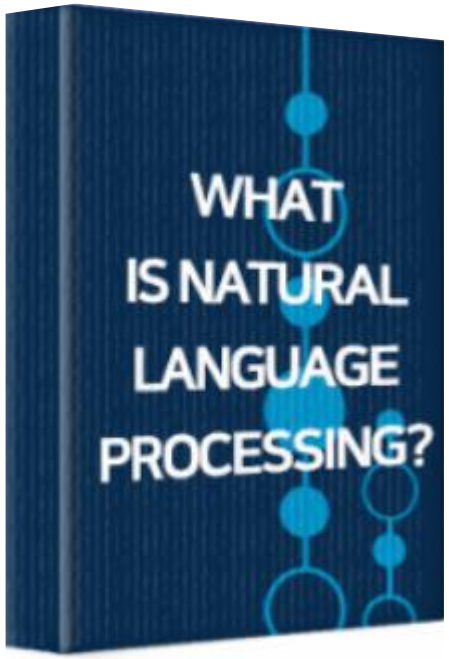
Tensorflow를 활용한 딥러닝 자연어 처리 입문. 4강

앞으로 배우게 될 내용

- Text preprocessing for NLP & Language Model
- Basic Tensorflow & Vectorization
- Word Embedding (Word2Vec, FastText, GloVe)
- **Text Classification (using RNN & CNN)**
- Chatbot with Deep Learning
- Sequence to Sequence
- Attention Mechanism
- Transformer & BERT



참고 자료 : https://wikidocs.net/book/2155

Text Classification

Text Classification

주어진 텍스트가 이미 정의된 카테고리 중 어디로 분류되어야 하는지를 예측하는 작업

자연어 처리 작업 중 가장 수요가 높으면서 기본적인 태스크

Text Classification

주어진 텍스트가 이미 정의된 카테고리 중 어디로 분류되어야 하는지를 예측하는 작업

자연어 처리 작업 중 가장 수요가 높으면서 기본적인 태스크

실제로 쓰이는 예제

Text Classification

주어진 텍스트가 이미 정의된 카테고리 중 어디로 분류되어야 하는지를 예측하는 작업
자연어 처리 작업 중 가장 수요가 높으면서 기본적인 태스크

실제로 쓰이는 예제

- 언론사의 뉴스들을 자동으로 IT, 정치, 문화 등의 카테고리로 분류하는 뉴스 분류기
- 스팸 메일이나 문자를 자동으로 스팸으로 판단하는 스팸 분류기
- 뉴스 데이터로부터 어떤 종목 뉴스인지 판별하여 해당 종목 게시판으로 분류
- 사용자가 남긴 리뷰로부터 긍정인지 부정인지 감성 분류
- 챗봇은 사용자가 입력한 질의로부터 '환불', '주문', '건의' 등인지 분류하고 이에 맞는 대응을 한다.

Text Classification

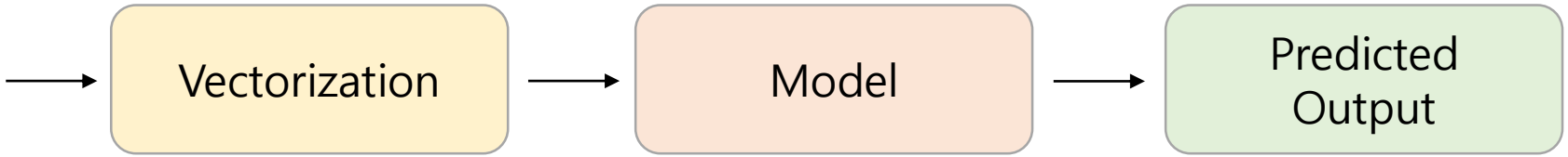
- 인공 신경망을 이용한 텍스트 분류 과정을 그림으로 도식화하면 아래와 같다.

문서1 : me free lottery

문서2 : free get free you

문서3 : you free scholarship

문서4 : free to contact me



Text Classification

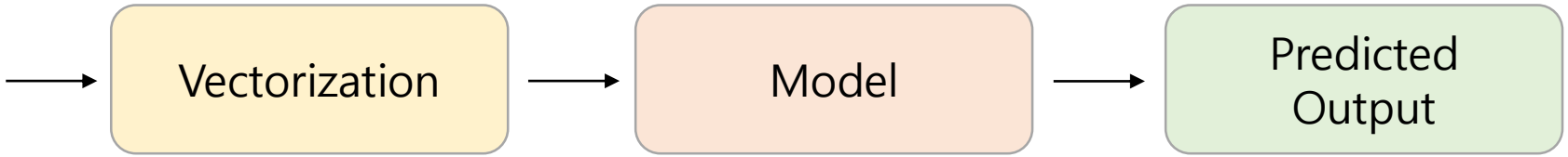
- 인공 신경망을 이용한 텍스트 분류 과정을 그림으로 도식화하면 아래와 같다.
- 분류는 크게 **이진 분류(Binary Classification)**과 **다중 클래스 분류(multiclass classification)**가 있다.

문서1 : me free lottery

문서2 : free get free you

문서3 : you free scholarship

문서4 : free to contact me



Text Classification

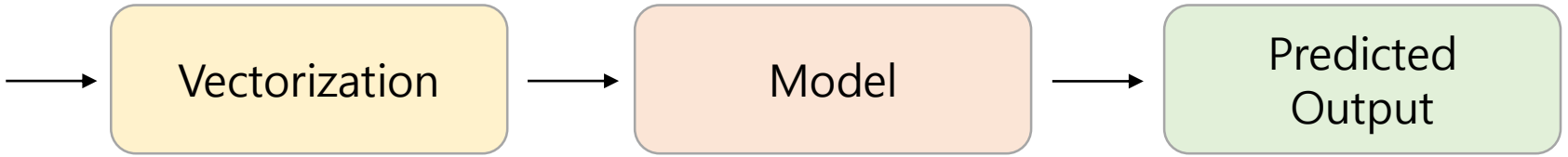
- 인공 신경망을 이용한 텍스트 분류 과정을 그림으로 도식화하면 아래와 같다.
- 분류는 크게 **이진 분류(Binary Classification)**과 **다중 클래스 분류(multiclass classification)**가 있다.

문서1 : me free lottery

문서2 : free get free you

문서3 : you free scholarship

문서4 : free to contact me



1) DTM, TF-IDF
2) Word Embedding
3) Document Embedding

Text Classification

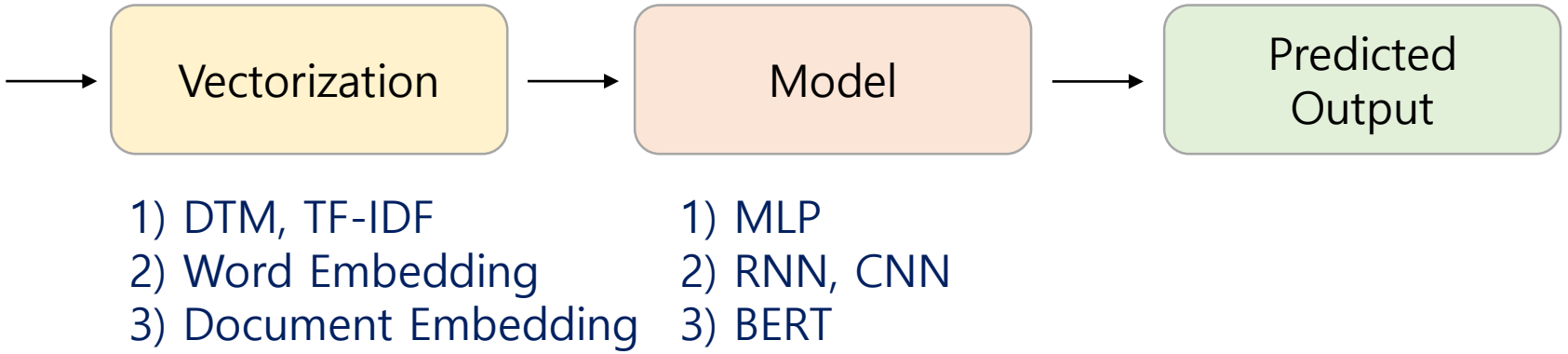
- 인공 신경망을 이용한 텍스트 분류 과정을 그림으로 도식화하면 아래와 같다.
- 분류는 크게 **이진 분류(Binary Classification)**과 **다중 클래스 분류(multiclass classification)**가 있다.

문서1 : me free lottery

문서2 : free get free you

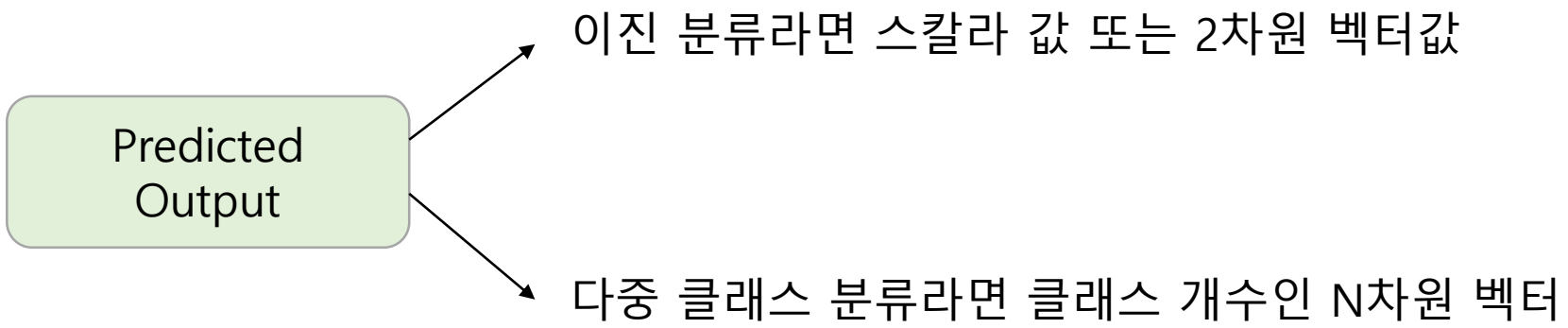
문서3 : you free scholarship

문서4 : free to contact me



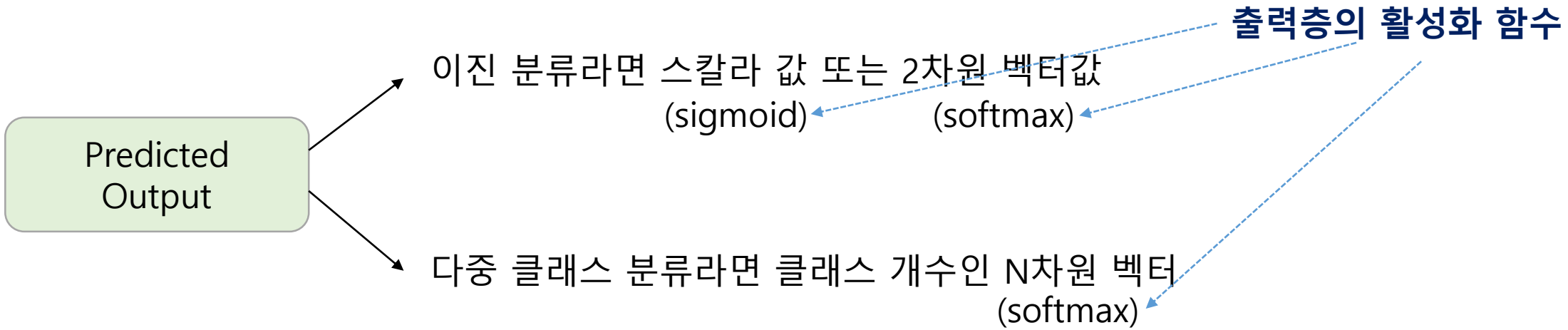
Text Classification

- 인공 신경망을 이용한 텍스트 분류 과정을 그림으로 도식화하면 아래와 같다.
- 분류는 크게 **이진 분류(Binary Classification)**과 **다중 클래스 분류(multiclass classification)**가 있다.
- 클래스 수가 **2개**면 이진 분류라고 하며, **3개 이상**이면 다중클래스 분류



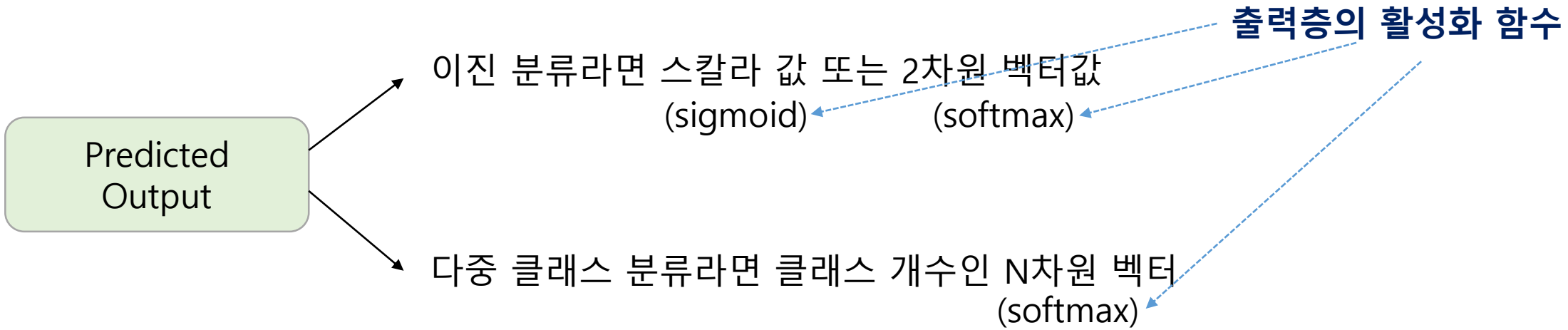
Text Classification

- 인공 신경망을 이용한 텍스트 분류 과정을 그림으로 도식화하면 아래와 같다.
- 분류는 크게 **이진 분류(Binary Classification)**과 **다중 클래스 분류(multiclass classification)**가 있다.
- 클래스 수가 **2개**면 이진 분류라고 하며, **3개 이상**이면 다중클래스 분류



Text Classification

- 인공 신경망을 이용한 텍스트 분류 과정을 그림으로 도식화하면 아래와 같다.
- 분류는 크게 **이진 분류(Binary Classification)**과 **다중 클래스 분류(multiclass classification)**가 있다.
- 클래스 수가 **2개**면 이진 분류라고 하며, **3개 이상**이면 다중클래스 분류



지난 시간 내용(2강, 3강) 간단히 복습!

Text Classification using MLP

2주차에 배운 내용을 복습해봅시다.
각각의 문서를 다음과 같이 고정 길이를 가지는 벡터로 변환합니다.

문서1 : me free lottery
문서2 : free get free you
문서3 : you free scholarship
문서4 : free to contact me

Bag of Words →

Document #1 Vector

Document Term Matrix

	contact	free	get	lottery	me	scholarship	to	you
0	0	1	0	1	1	0	0	0
1	0	2	1	0	0	0	0	1
2	0	1	0	0	0	1	0	1
3	1	1	0	0	1	0	1	0

Vocabulary size

Text Classification using MLP

2주차에 배운 내용을 복습해봅시다.
각각의 문서를 다음과 같이 고정 길이를 가지는 벡터로 변환합니다.

문서1 : me free lottery
문서2 : free get free you
문서3 : you free scholarship
문서4 : free to contact me

Bag of Words

Document #1 Vector

Document Term Matrix

	contact	free	get	lottery	me	scholarship	to	you
0	0	1	0	1	1	0	0	0
1	0	2	1	0	0	0	0	1
2	0	1	0	0	0	1	0	1
3	1	1	0	0	1	0	1	0

Vocabulary size

label
1
1
0
0

실제값

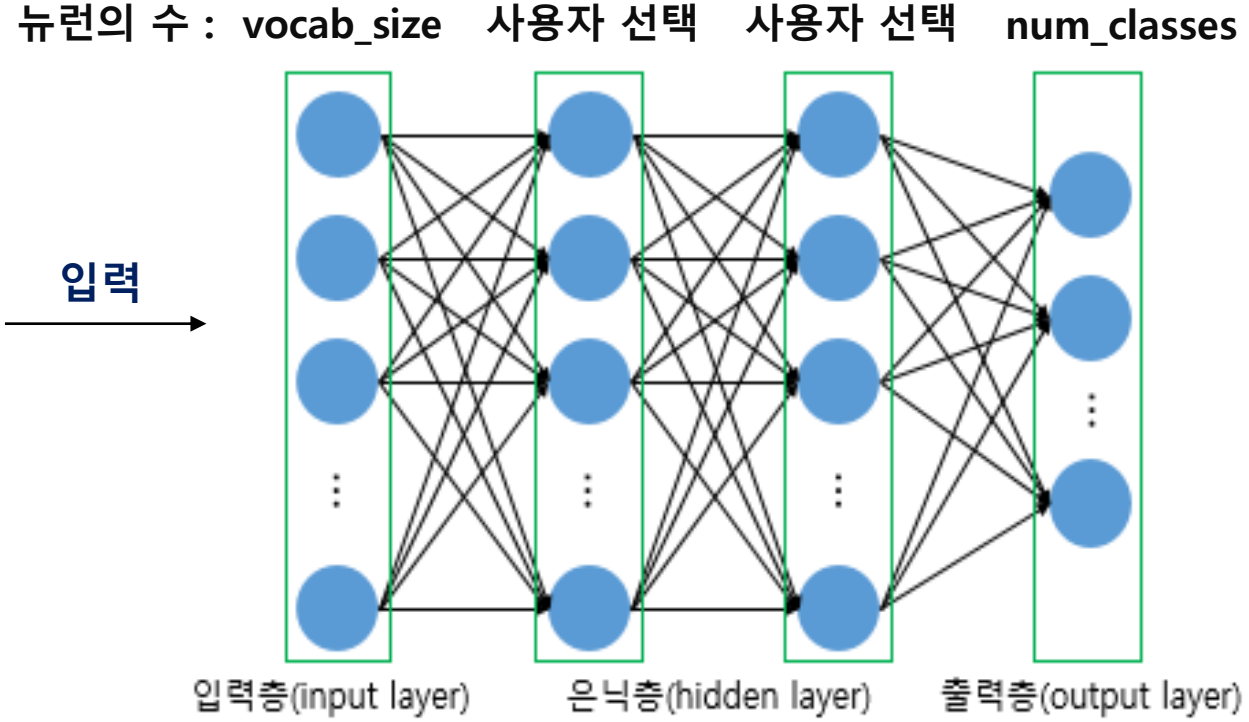
Text Classification using MLP

각각의 문서를 다음과 같이 고정 길이를 가지는 벡터로 변환합니다.
Vocabulary size를 입력층의 뉴런 수로 가지는 MLP에 문서 벡터를 입력으로 사용.

vocab size

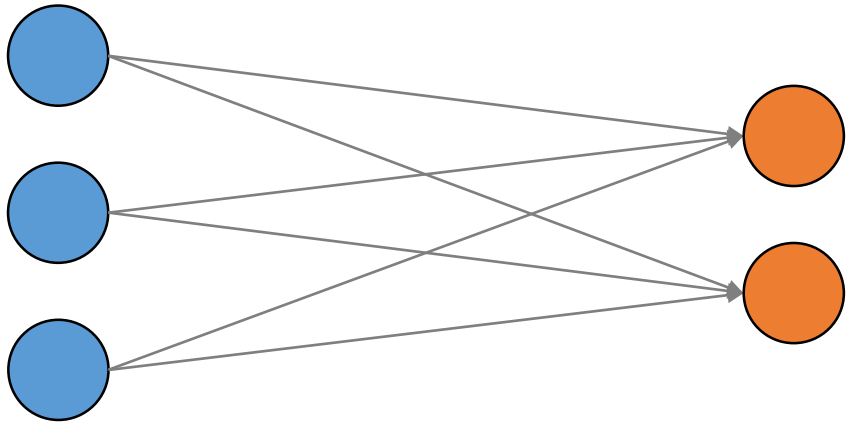
	0	1	2	3
contact	0	1	2	3
free	0	0	1	1
get	1	0	1	0
lottery	0	2	1	1
me	1	0	0	0
scholarship	0	0	0	1
to	0	0	0	0
you	0	1	1	0

입력



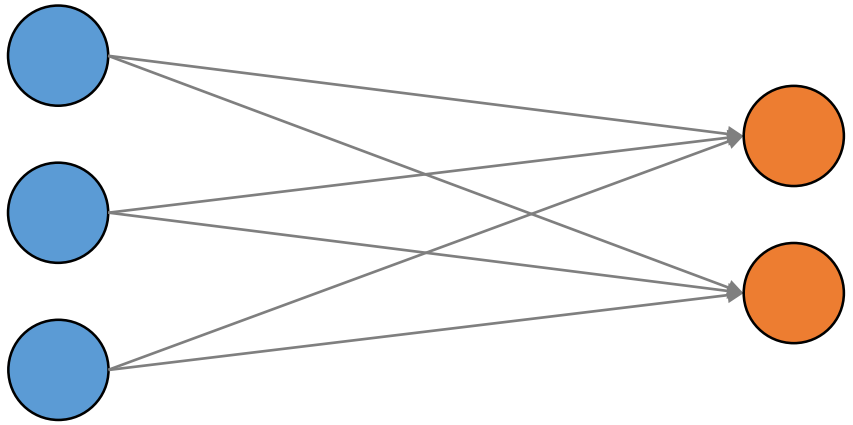
Feed Forward Neural Network

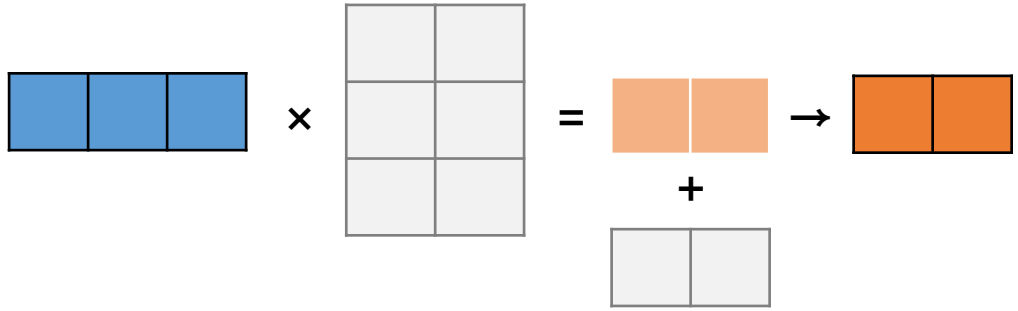
신경망을 행렬과 벡터 연산으로 이해할 수 있다면, 신경망을 이해하기가 매우 용이하다.



Feed Forward Neural Network

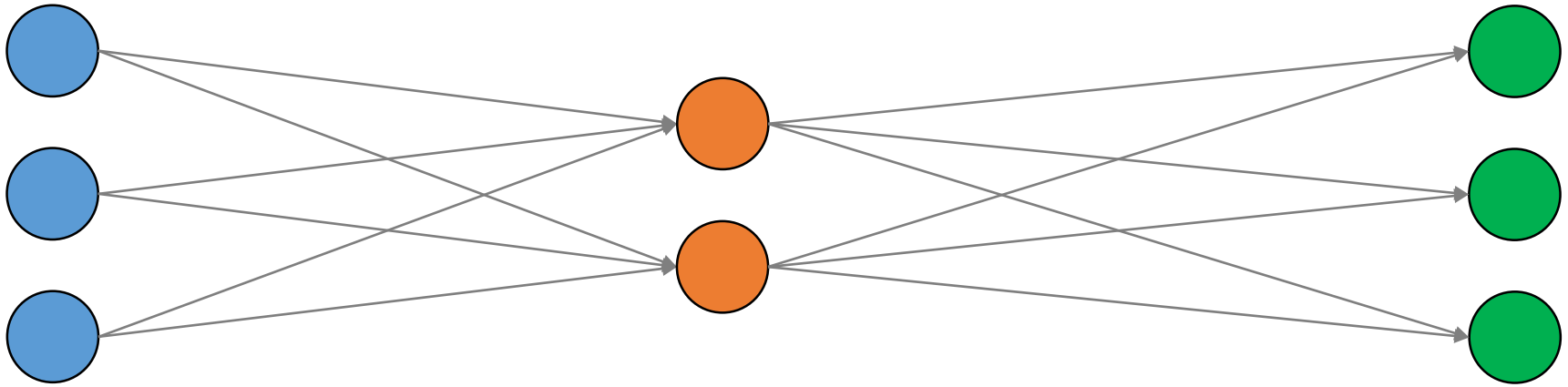
신경망을 행렬과 벡터 연산으로 이해할 수 있다면, 신경망을 이해하기가 매우 용이하다.





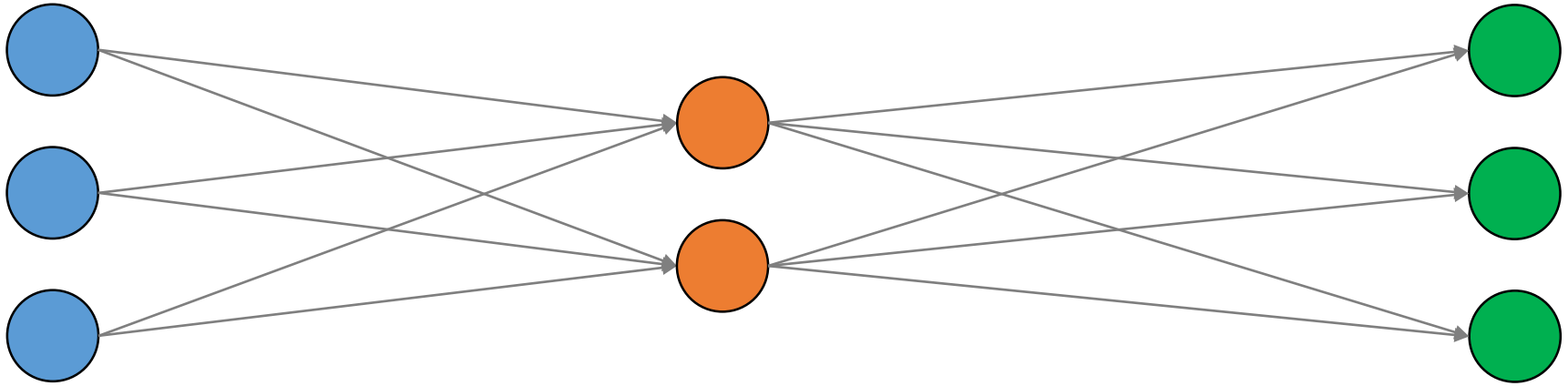
Feed Forward Neural Network

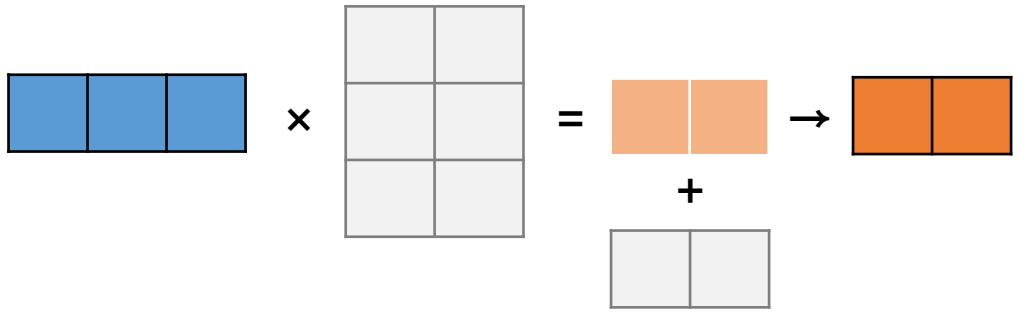
신경망을 행렬과 벡터 연산으로 이해할 수 있다면, 신경망을 이해하기가 매우 용이하다.



Feed Forward Neural Network

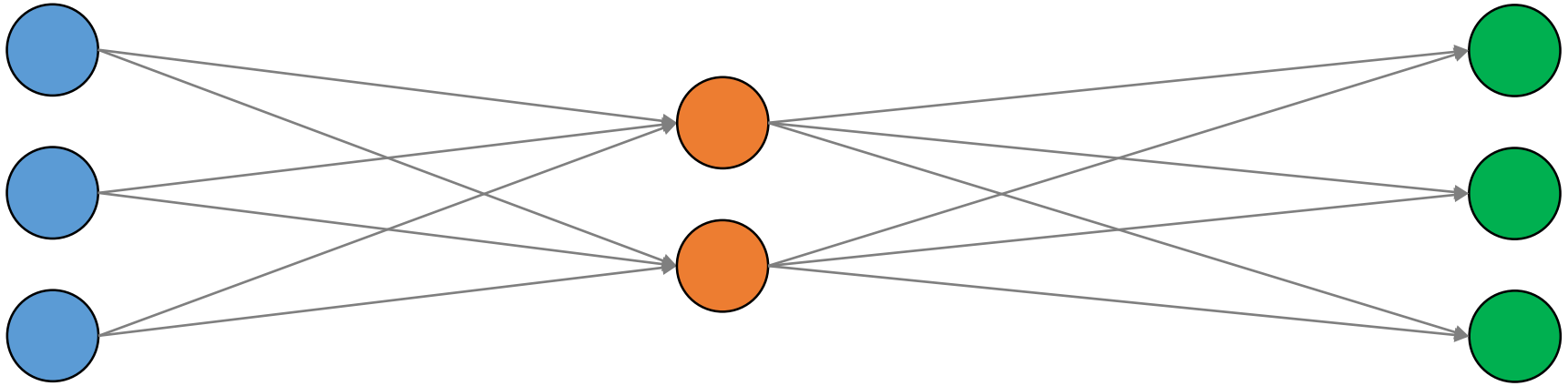
신경망을 행렬과 벡터 연산으로 이해할 수 있다면, 신경망을 이해하기가 매우 용이하다.

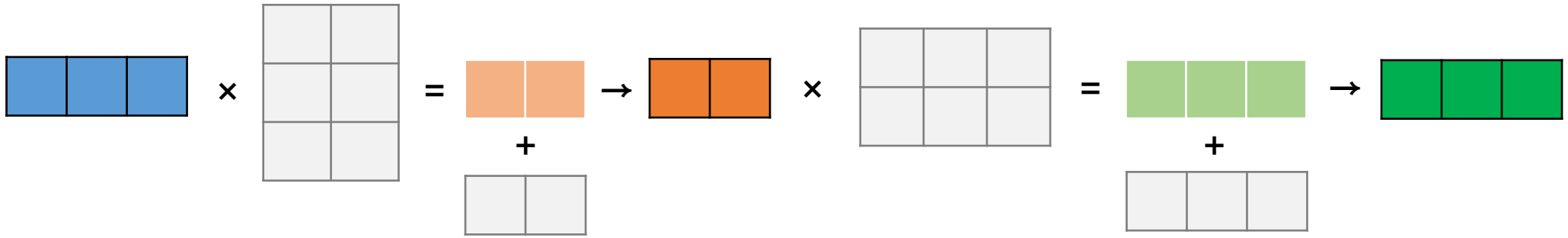




Feed Forward Neural Network

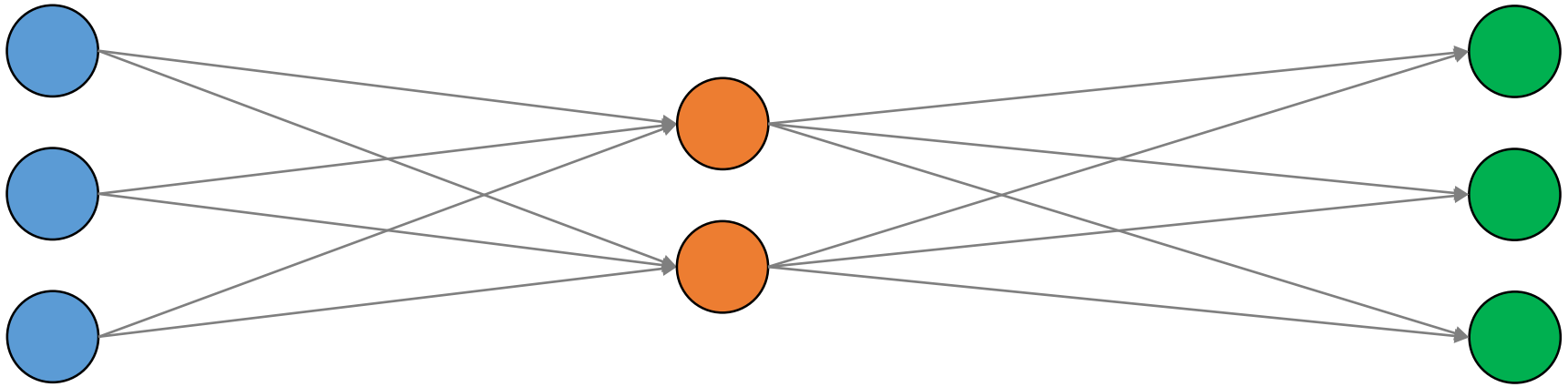
신경망을 행렬과 벡터 연산으로 이해할 수 있다면, 신경망을 이해하기가 매우 용이하다.

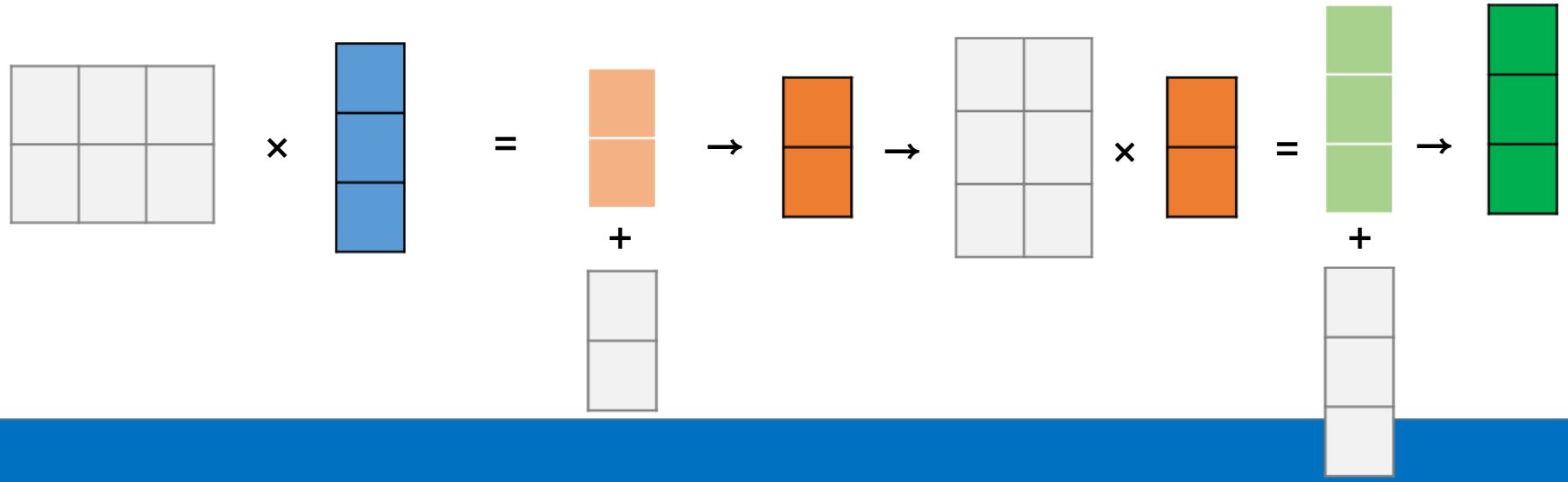




Feed Forward Neural Network

신경망을 행렬과 벡터 연산으로 이해할 수 있다면, 신경망을 이해하기가 매우 용이하다.

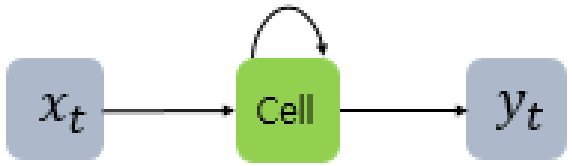




Recurrent Neural Network

Recurrent Neural Network

연속적인 시퀀스를 처리하기 위한 신경망
사람은 글을 읽을 때, 이전 단어들에 대한 이해를 바탕으로 다음 단어를 이해한다.
기존의 MLP에 비해서 RNN은 이러한 이슈를 다루며, 내부에 정보를 지속하는 루프로 구성된 신경망

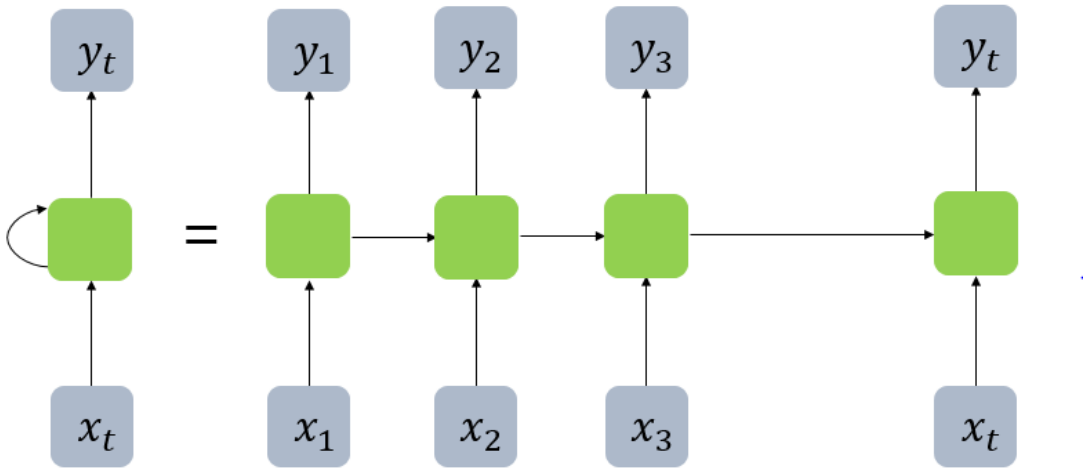


Recurrent Neural Networks have loops.

위 다이어그램에서 x_t 는 현재의 입력, y_t 는 과거와 현재의 정보를 반영한 출력.

Recurrent Neural Network

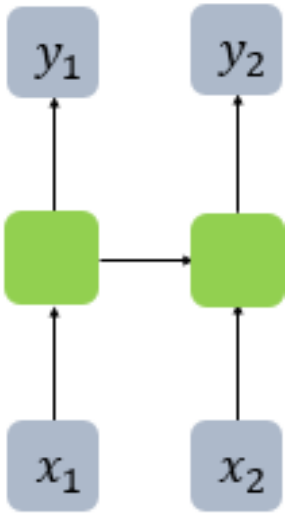
- RNN은 입력의 길이만큼 신경망이 펼쳐진다(unrolled).
- 이때, 입력받는 각 순간을 시점(time step)이라고 한다.
- x_t 는 t시점의 입력을 의미한다. 예를 들어 x_2는 두번째 시점의 입력을 의미한다.

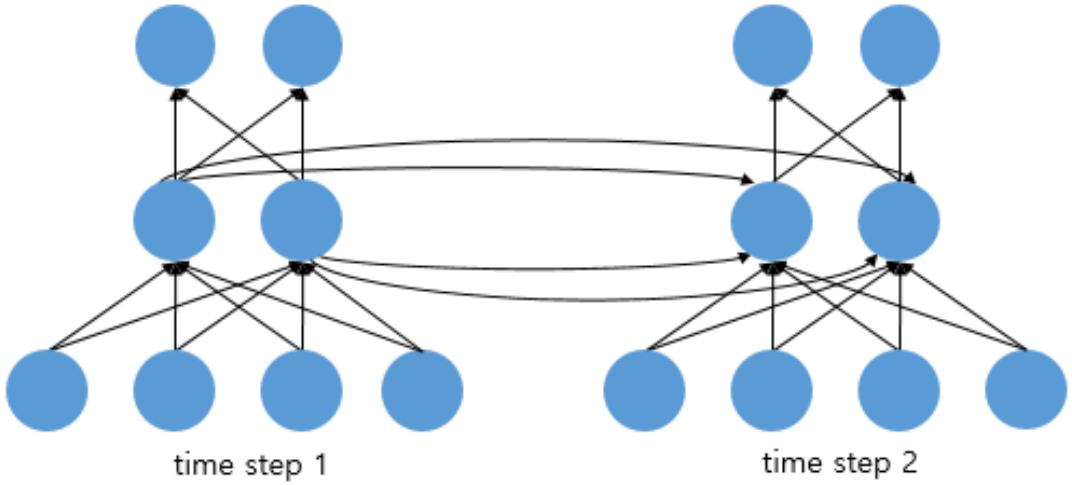


unrolled Recurrent Neural Network

Recurrent Neural Network Vs. FFNN

RNN은 피드 포워드 신경망에 시점(time step)이라는 개념을 도입한 것과 같다.
RNN의 입력과, 출력은 모두 기본적으로 벡터 단위를 가정한다.
아래의 두 그림은 보편적인 RNN 그림과 FFNN을 통해 RNN을 표현한 경우를 보여준다.

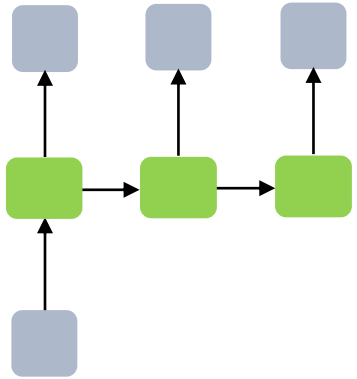




unrolled Recurrent Neural Network

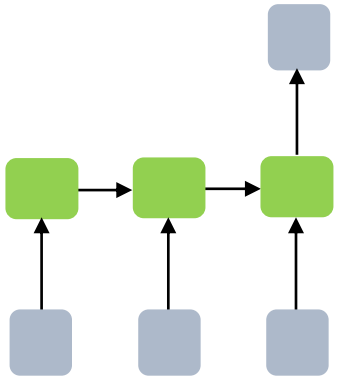
Types of Recurrent Neural Network

RNN은 설계하기 나름이지만 대표적으로 아래와 같은 유형이 있다.
자연어 처리에서 각 시점(time step)의 입력은 주로 단어 벡터 또는 형태소(한국어) 벡터가 된다.



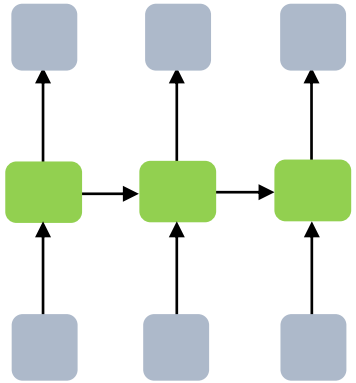
one-to-many

Ex) Image Captioning



many-to-one

Ex) Text Classification

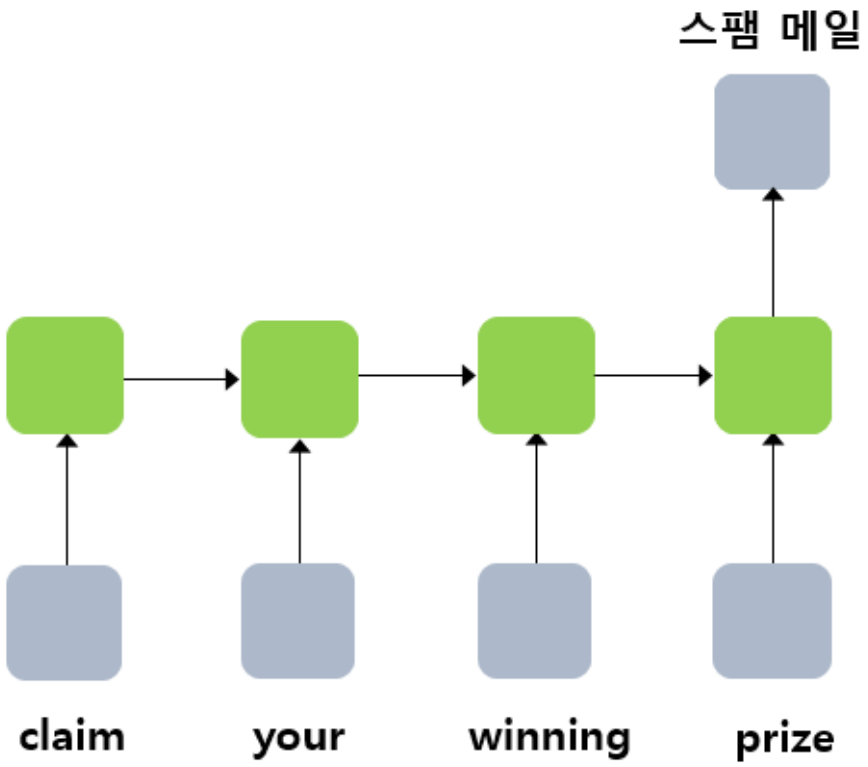


many-to-many

Ex) Named Entity Recognition

many-to-one RNN

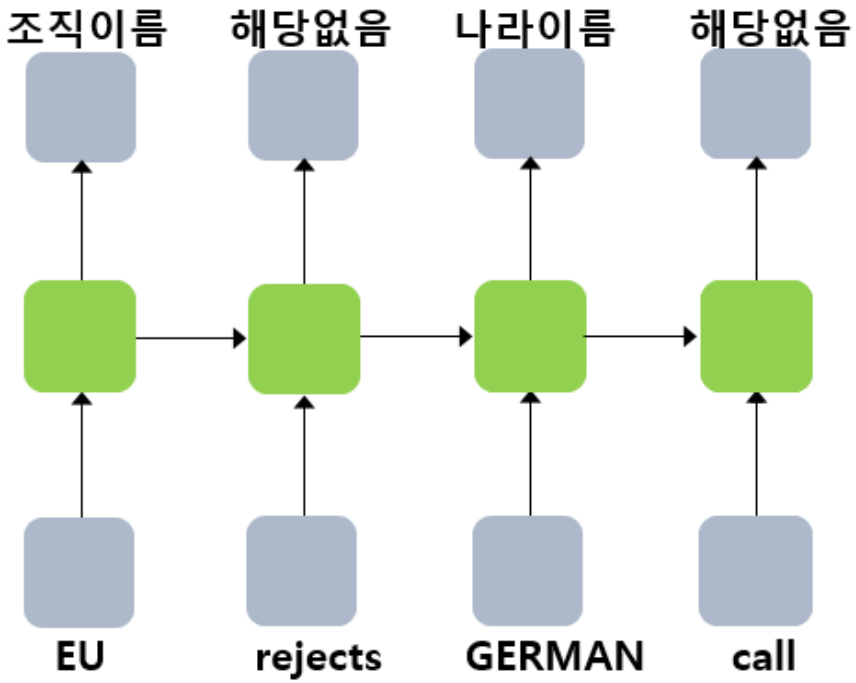
문서의 단어들을 순차적으로 입력받아 해당 문서의 유형을 판단하는 텍스트 분류에 사용될 수 있다.



Text Classification example : spam detector

many-to-many RNN

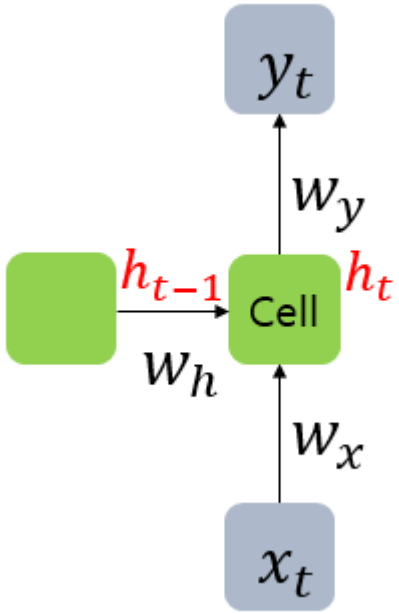
다-대-다는 각 단어에 레이블을 달아주는 시퀀스 레이블링 작업에 사용된다.
시퀀스 레이블링 작업의 대표적인 예로 개체명 인식이 있다.



Sequence Labeling example : Named Entity Recognition

Basic architecture of RNN

- 은닉층에 있는 RNN의 처리 단위를 셀(cell)이라고 부르며, 셀의 출력을 은닉 상태(hidden state)라고 한다.
- RNN은 시점(time step)에 따라서 입력을 받는데 현재 시점의 hidden state인 h_t연산을 위해 직전 시점의 hidden state인 h_{t-1}를 입력받는다. 이게 RNN이 과거의 정보를 기억하고 있는 비결이다.

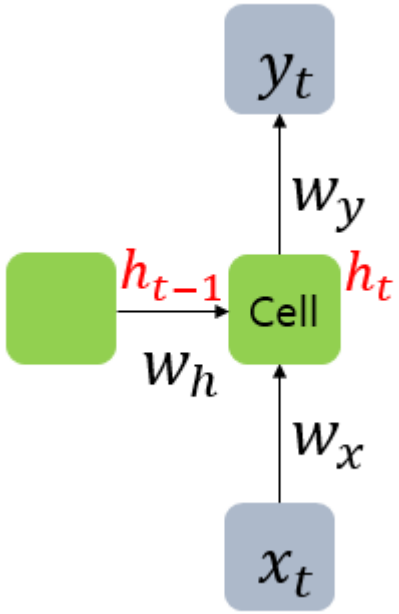


출력층 : y_t= 어떤 활성화 함수$(W_y h_t + b)$

은닉층 : $h_t = tanh(W_h h_{t-1} + W_x x_t + b)$

Basic architecture of RNN

- 은닉층에 있는 RNN의 처리 단위를 셀(cell)이라고 부르며, 셀의 출력을 은닉 상태(hidden state)라고 한다.
- RNN은 시점(time step)에 따라서 입력을 받는데 현재 시점의 hidden state인 h_t연산을 위해 직전 시점의 hidden state인 h_{t-1}를 입력받는다. 이게 RNN이 과거의 정보를 기억하고 있는 비결이다.

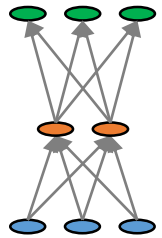


출력층 : $y_t = \text{어떤 활성화 함수}(W_y h_t + b)$

은닉층 : $h_t = tanh(W_h h_{t-1} + W_x x_t + b)$

출력층 : $y_t = softmax(W_y h_t + b)$

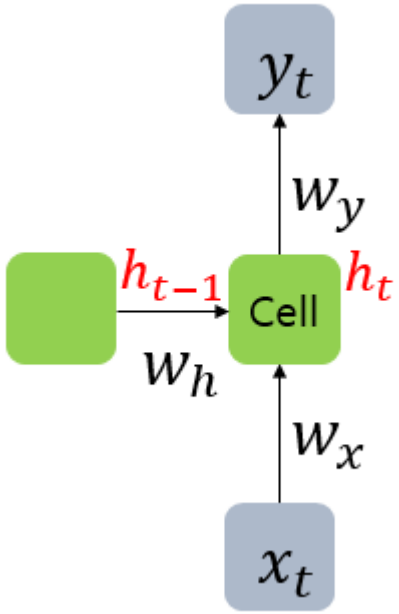
은닉층 : $h_t = tanh(W_x x_t + b)$



입력 : x_t

Basic architecture of RNN

- 은닉층에 있는 RNN의 처리 단위를 셀(cell)이라고 부르며, 셀의 출력을 은닉 상태(hidden state)라고 한다.
- RNN은 시점(time step)에 따라서 입력을 받는데 현재 시점의 hidden state인 h_t연산을 위해 직전 시점의 hidden state인 h_{t-1}를 입력받는다. 이게 RNN이 과거의 정보를 기억하고 있는 비결이다.



출력층 : y_t= 어떤 활성화 함수$(W_y h_t + b)$

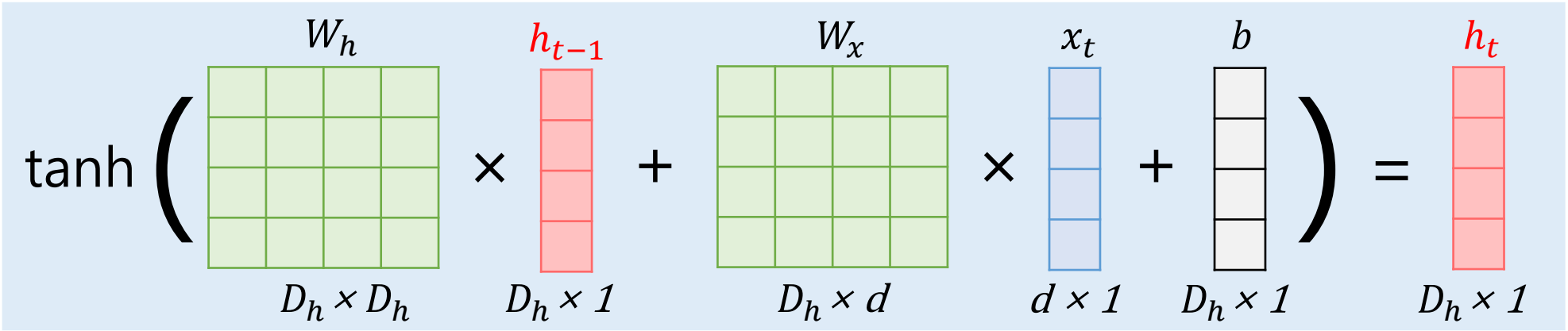
은닉층 : $h_t = tanh(W_h h_{t-1} + W_x x_t + b)$

이 과정을 좀 더 자세히 볼까요?

Basic architecture of RNN : Matrix Calculation

- 은닉층에 있는 RNN의 처리 단위를 셀(cell)이라고 부르며, 셀의 출력을 은닉 상태(hidden state)라고 한다.
- RNN은 시점(time step)에 따라서 입력을 받는데 현재 시점의 hidden state인 h_t연산을 위해 직전 시점의 hidden state인 h_{t-1}를 입력받는다. 이게 RNN이 과거의 정보를 기억하고 있는 비결이다.

은닉층 : $h_t = tanh(W_h h_{t-1} + W_x x_t + b)$

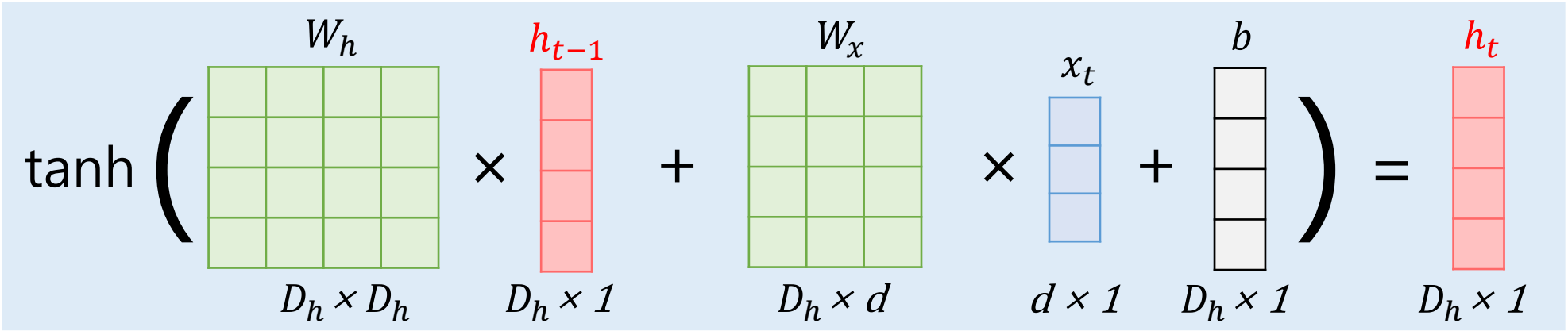


d : t time step의 단어 벡터의 차원
D_h : hidden state의 size (RNN의 주요 하이퍼 파라미터)

Basic architecture of RNN : Matrix Calculation

- 은닉층에 있는 RNN의 처리 단위를 셀(cell)이라고 부르며, 셀의 출력을 은닉 상태(hidden state)라고 한다.
- RNN은 시점(time step)에 따라서 입력을 받는데 현재 시점의 hidden state인 h_t연산을 위해 직전 시점의 hidden state인 h_{t-1}를 입력받는다. 이게 RNN이 과거의 정보를 기억하고 있는 비결이다.

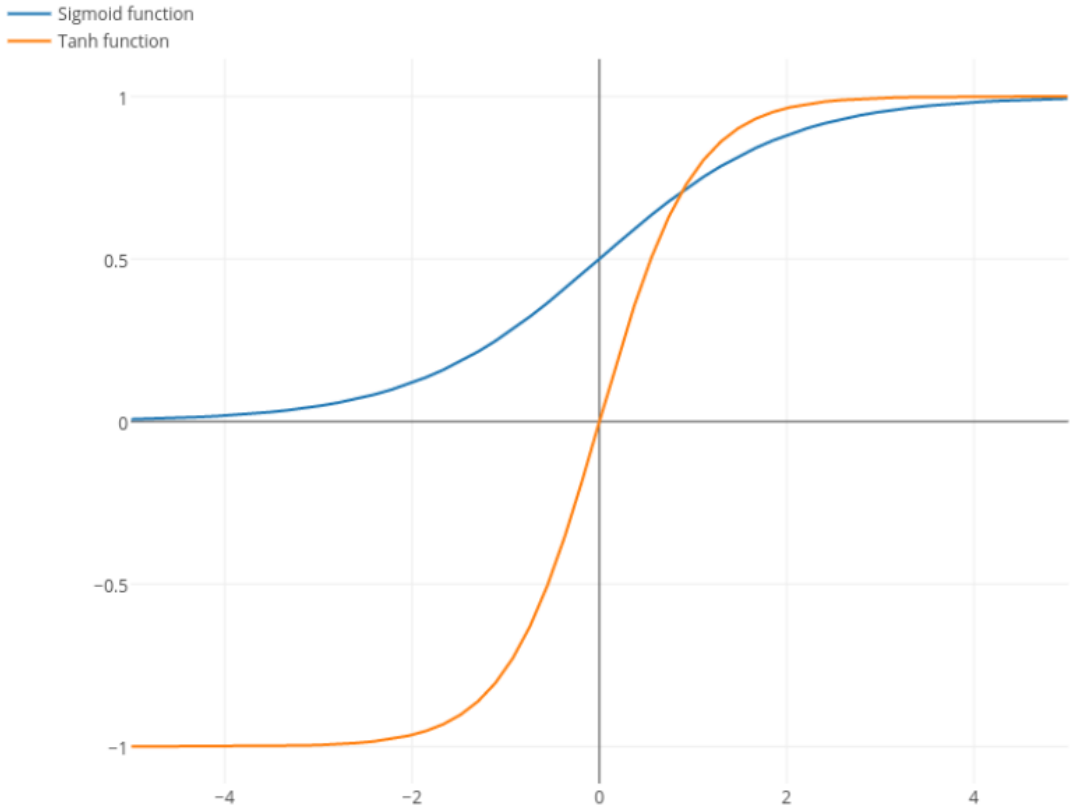
은닉층 : $h_t = tanh(W_h h_{t-1} + W_x x_t + b)$



d : t time step의 단어 벡터의 차원
D_h : hidden state의 size (RNN의 주요 하이퍼 파라미터)

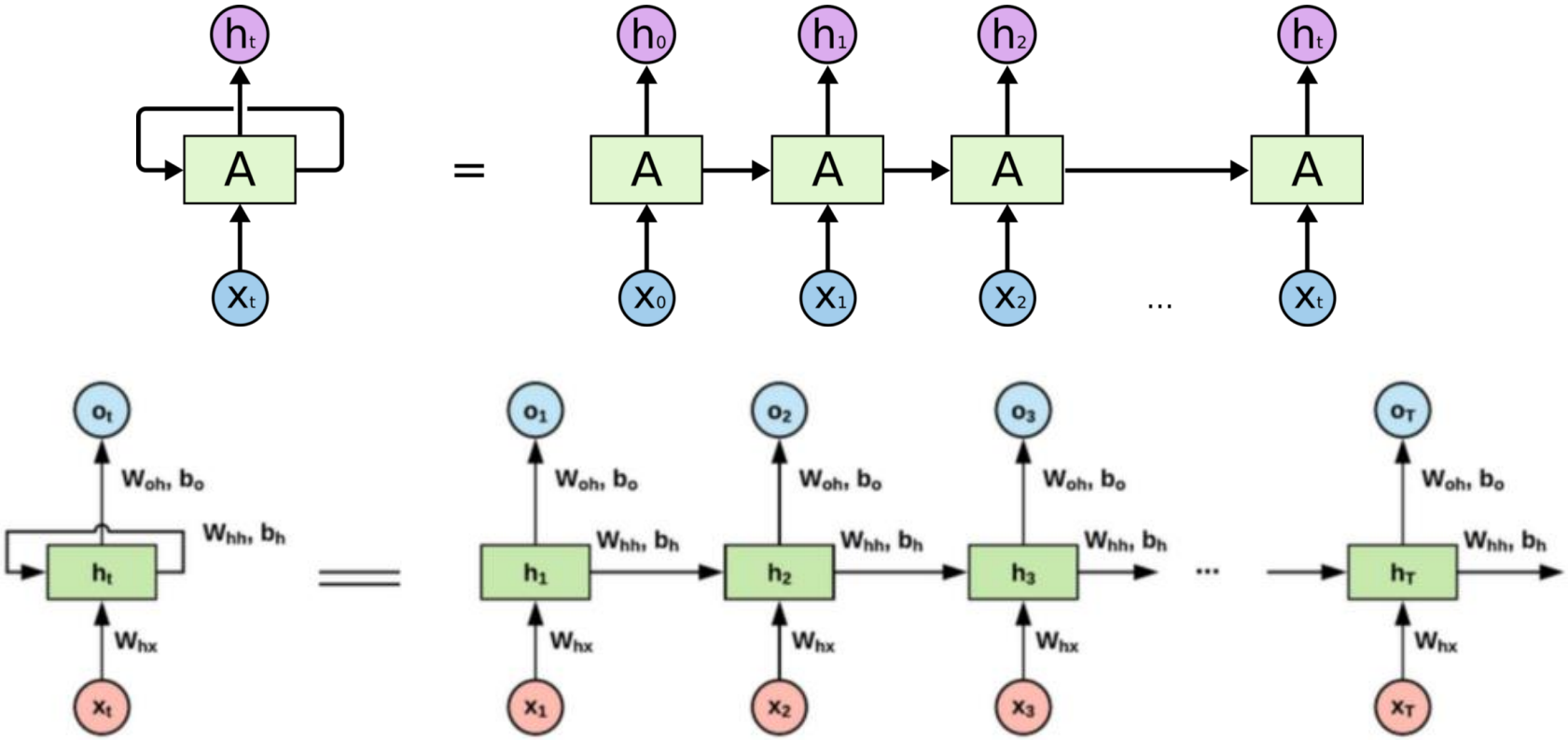
하이퍼볼릭탄젠트 함수

- 하이퍼볼릭탄젠트 함수는 시그모이드 함수와 달리 -1 ~ 1의 범위 값으로 값을 반환하는 함수이다.
- 반환값의 범위가 시그모이드 함수보다 크므로 일반적으로 은닉층에서는 시그모이드 함수보다 더 잘 동작한다.



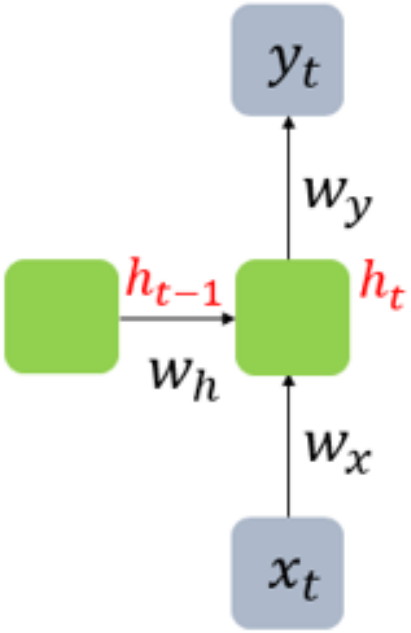
Basic architecture of RNN

- 아래의 두 RNN의 그림은 같은 그림일까요? 다른 그림일까요?
- 같은 아키텍처를 표현한 걸까요? 아니면 다른 아키텍처를 표현한 걸까요?

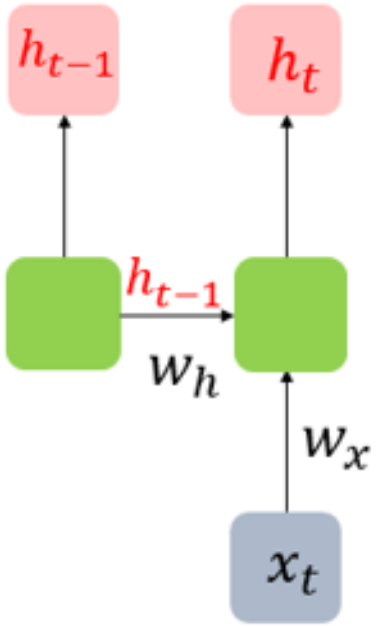


Basic architecture of RNN

- 인터넷에는 주로 두 가지 종류의 RNN 그림이 있다. 이를 혼동하면 안 된다.
- 두 그림은 엄연히 다른 출력을 표현하고 있다.



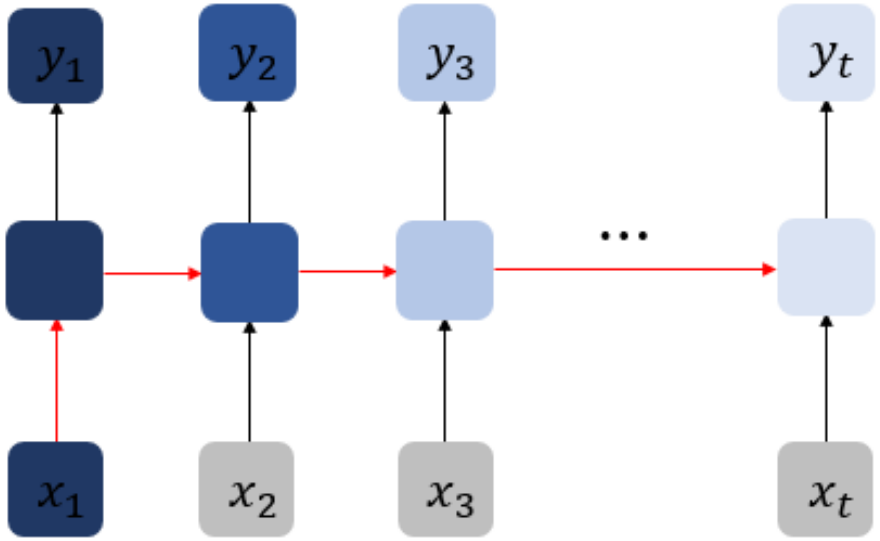
(a) 입력층, 은닉층, 출력층



(b) 입력층, 은닉층까지만 그린 그림

Long-Term Dependency Ploblem

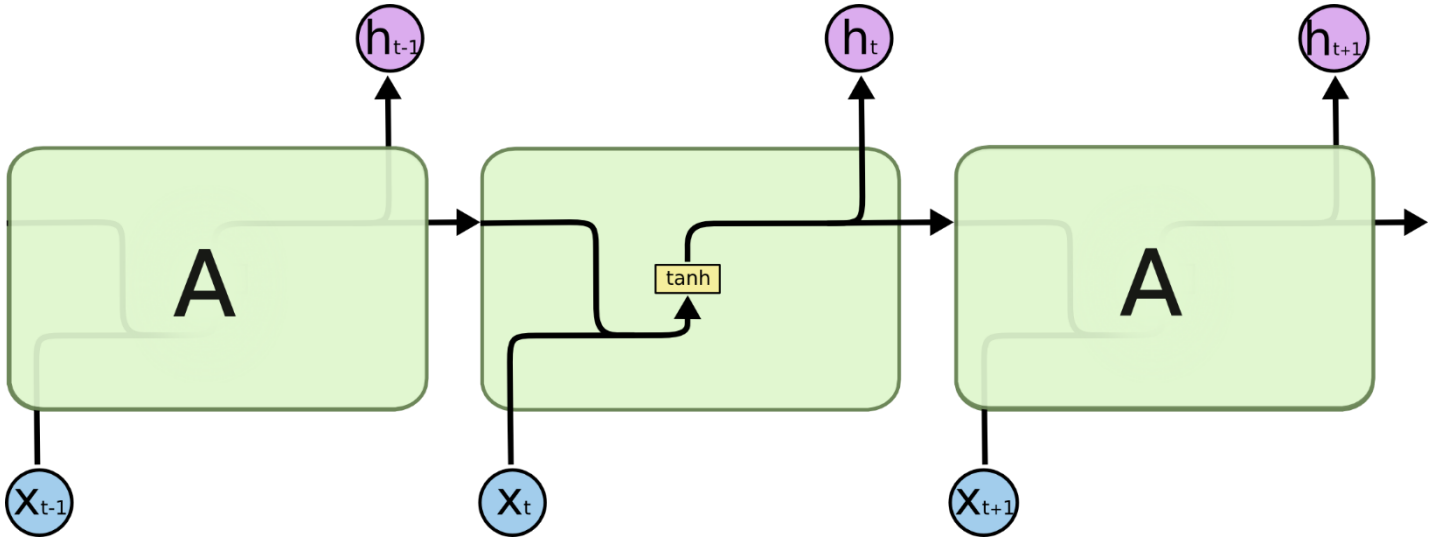
- 기존의 RNN은 시점이 길어지면서 앞에 있던 정보가 소실되는 '장기 의존성 문제'를 갖고 있다.
- x_1 의 정보량을 짙은 남색으로 표현하였을 때 뒤로 갈수록 점점 소실되는 것을 색의 얕아짐으로 표현하였다.
- 이는 RNN의 고질적인 문제이다.



Long-Term Dependency Ploblem

Long Short-Term Memory, LSTM

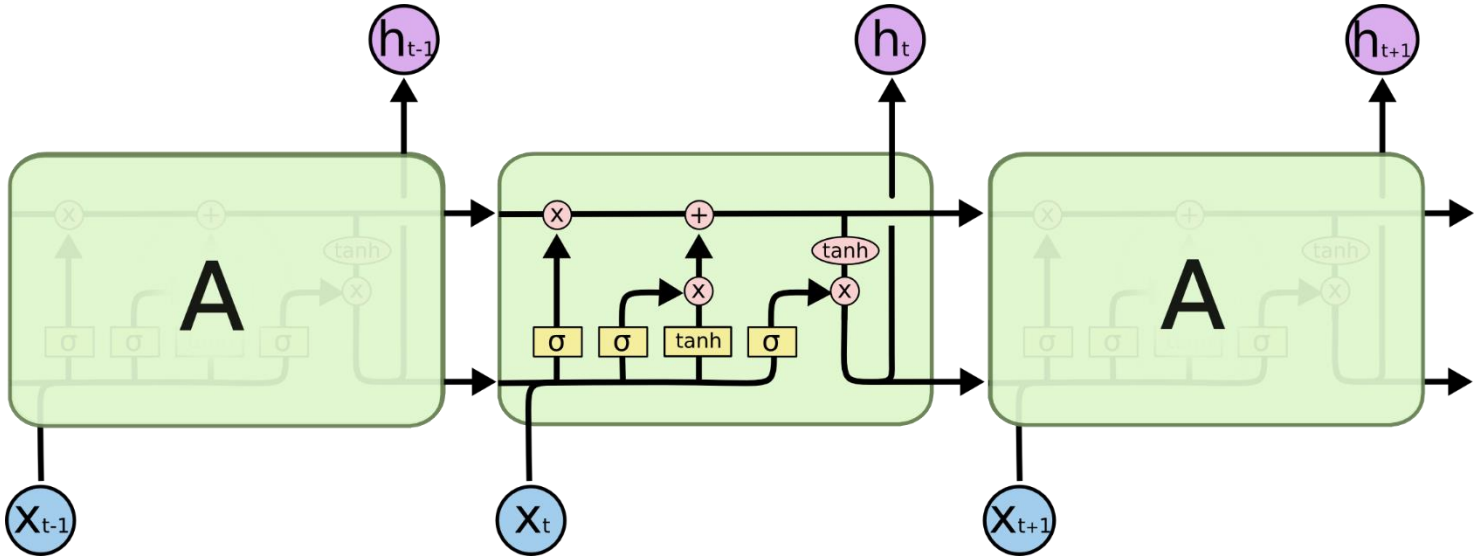
- 기존 RNN(바닐라 RNN)의 장기 의존성 문제를 개선하여 기억력을 높인 RNN의 명칭.
- 앞으로 나오는 설명에서 RNN을 사용한다고 하면 기본적으로 LSTM(또는 GRU)를 사용한다고 가정한다.



Vanilla RNN

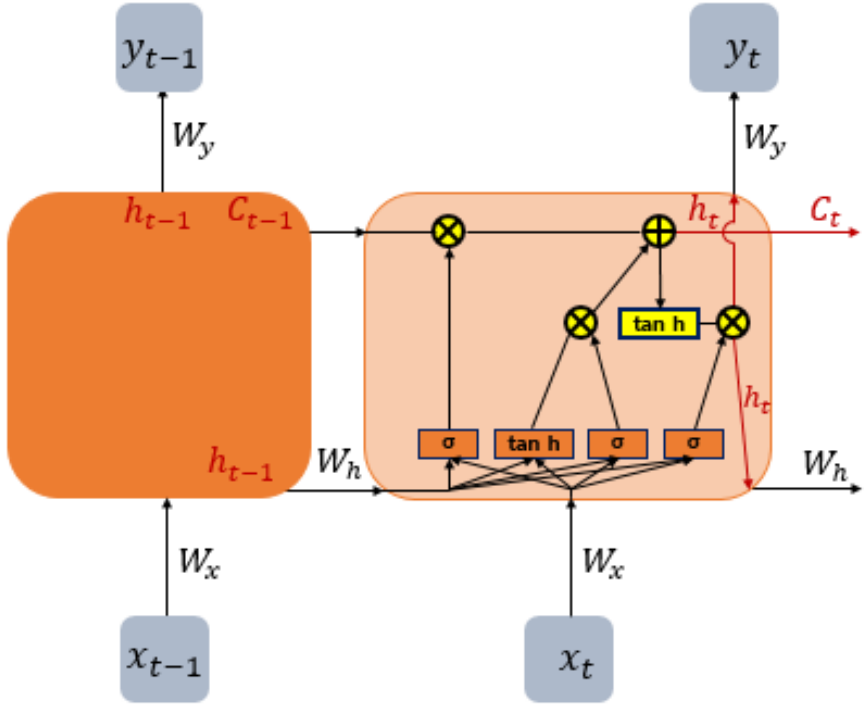
Long Short-Term Memory, LSTM

- 기존 RNN(바닐라 RNN)의 장기 의존성 문제를 개선하여 기억력을 높인 RNN의 명칭.
- 앞으로 나오는 설명에서 RNN을 사용한다고 하면 기본적으로 LSTM(또는 GRU)를 사용한다고 가정한다.



LSTM(Long Short-Term Memory)

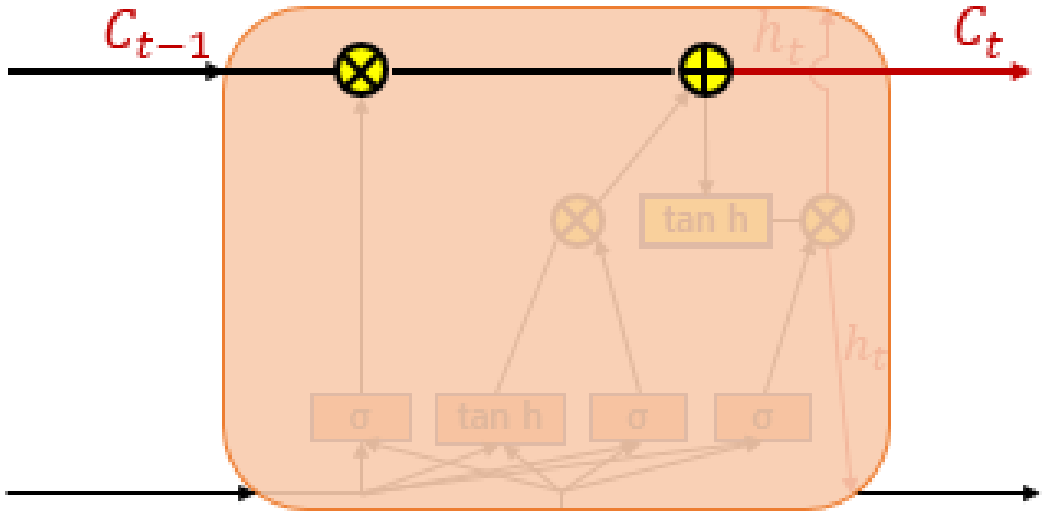
Long Short-Term Memory, LSTM



- 이하 식에서 σ는 시그모이드 함수를 의미합니다.
- 이하 식에서 tanh는 하이퍼볼릭탄젠트 함수를 의미합니다.
- $W_{xi}, W_{xg}, W_{xf}, W_{xo}$는 x_t와 함께 각 게이트에서 사용되는 4개의 가중치입니다.
- $W_{hi}, W_{hg}, W_{hf}, W_{ho}$는 h_{t-1}와 함께 각 게이트에서 사용되는 4개의 가중치입니다.
- b_i, b_g, b_f, b_o는 각 게이트에서 사용되는 4개의 편향입니다.

Long Short-Term Memory, LSTM

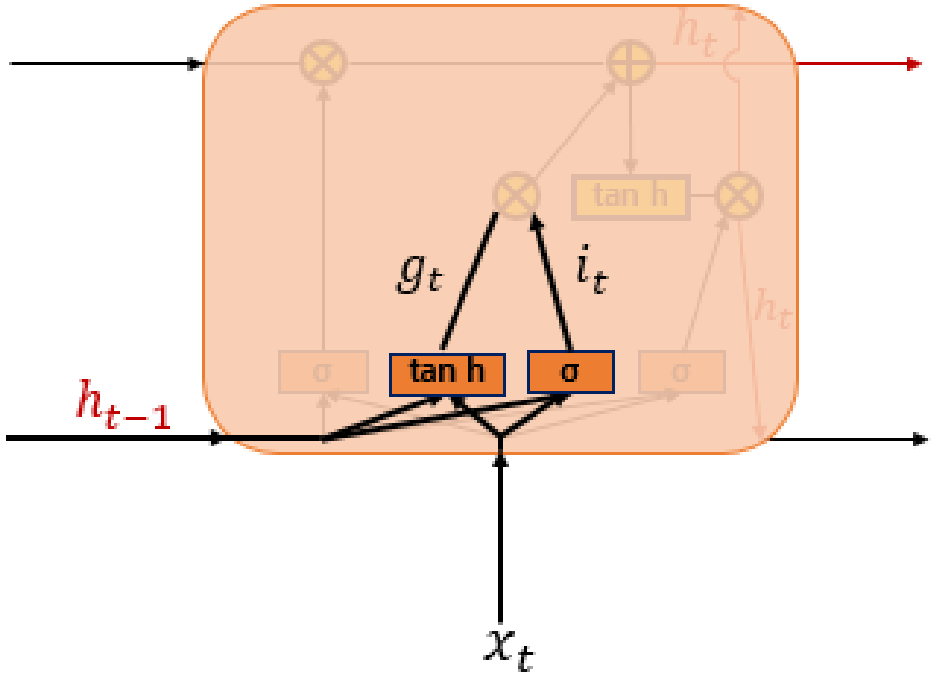
- LSTM의 핵심은 RNN에서는 없었던 Cell state이다.
- 이전에 배운 hidden state처럼 이전 시점의 cell state는 다음 시점의 cell state의 입력이 된다.
- cell state에 gate라는 구조를 통해서 정보를 더하거나 빼는 등의 통제를 한다.



Cell State

입력 게이트(Input Gate)

- 입력 게이트는 현재 정보를 기억하기 위한 게이트입니다.
- 시그모이드 함수를 지나 0과 1 사이의 값 and 하이퍼볼릭탄젠트 함수를 지나 -1과 1사이의 값이 두 개의 값
- 이 두 가지 값을 가지고 Cell state에서 이번에 선택된 기억할 값을 정한다.

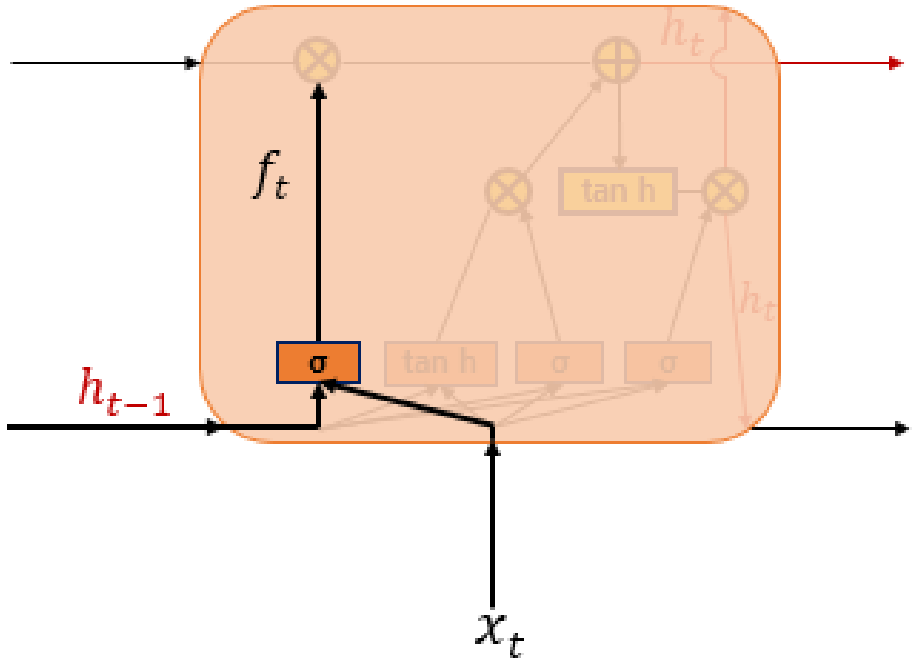


Input Gate

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i)$$
$$g_t = tanh(W_{xg}x_t + W_{hg}h_{t-1} + b_g)$$

삭제 게이트(Forget Gate)

- 삭제 게이트는 기억을 삭제하기 위한 게이트이다.
- 시그모이드 함수를 지나 0과 1 사이의 값이 나온다.
- 0에 가까울수록 정보가 많이 삭제된 것이며, 1에 가까울수록 정보를 온전히 기억한 셈이다.

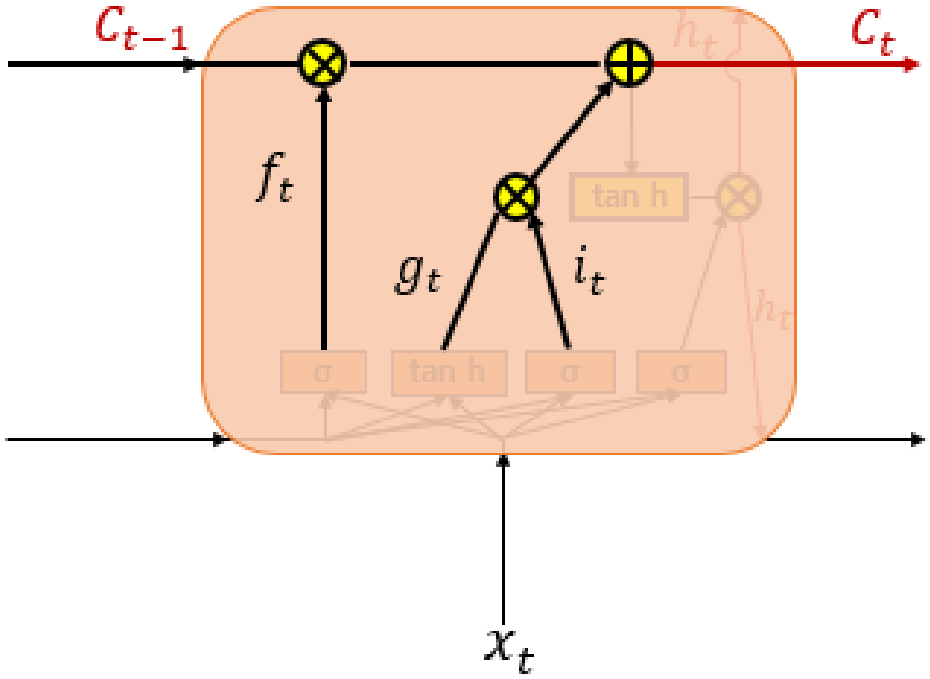


Forget Gate

$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + b_f)$$

셀 상태(Cell state) or 장기 상태

- 삭제 게이트에서 일부 기억을 소실.
- 입력 게이트의 i_t와 g_t를 가지고 elementiwise product를 수행 : 이번에 기억할 값.
- 두 값을 더한다.

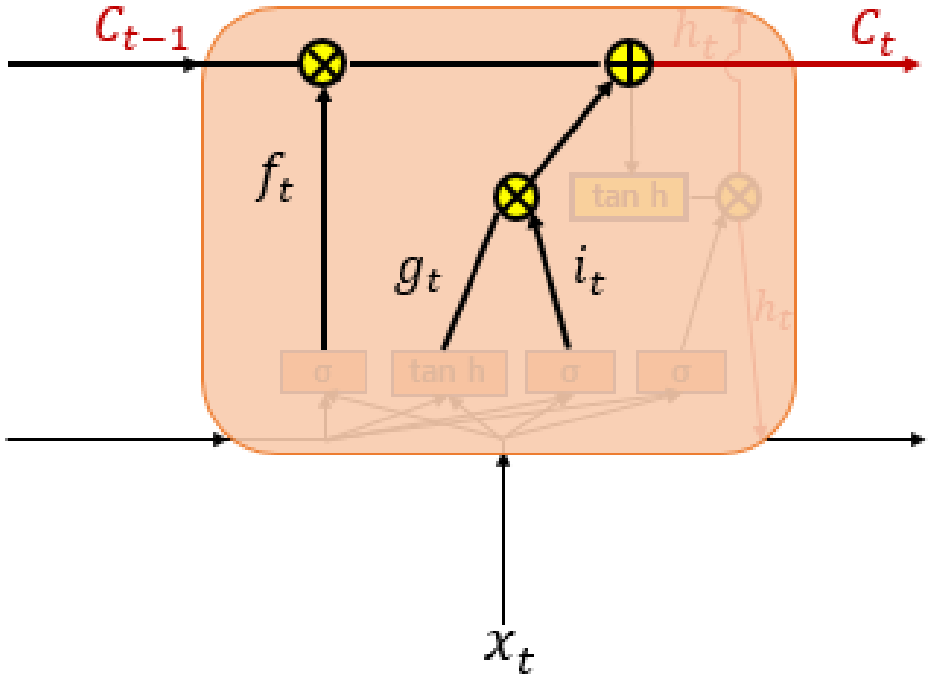


Cell State

$$C_t = f_t \circ C_{t-1} + i_t \circ g_t$$

셀 상태(Cell state) or 장기 상태

- 삭제 게이트에서 일부 기억을 소실.
- 입력 게이트의 i_t와 g_t를 가지고 elementiwise product를 수행 : 이번에 기억할 값.
- 두 값을 더한다.



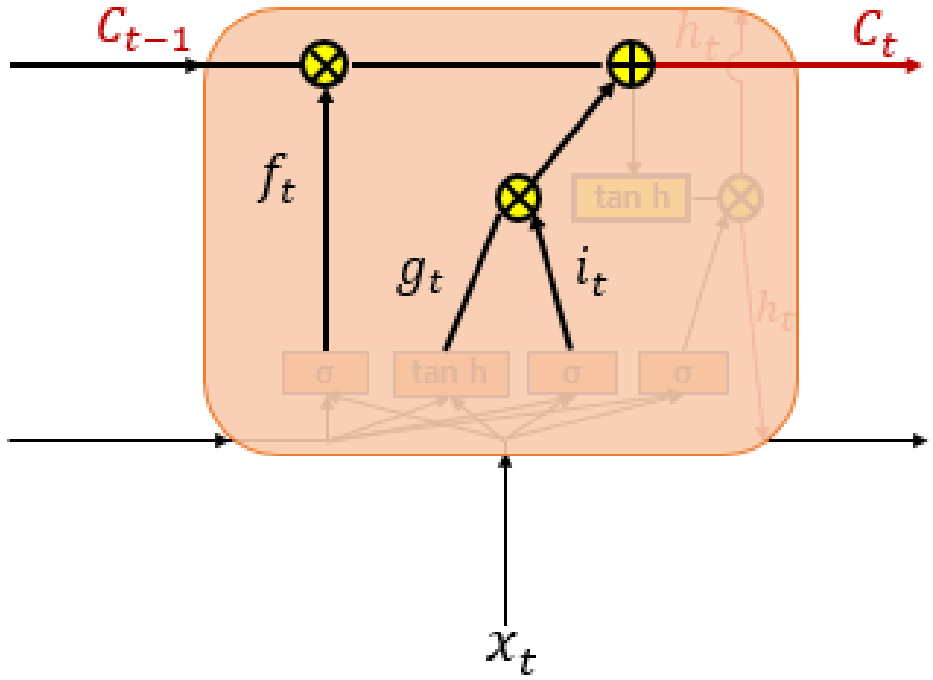
Cell State

$$C_t = f_t \circ C_{t-1} + i_t \circ g_t$$

삭제 게이트와 입력 게이트의 영향력을 생각해봅시다.

셀 상태(Cell state) or 장기 상태

- 삭제 게이트에서 일부 기억을 소실.
- 입력 게이트의 i_t와 g_t를 가지고 elementiwise product를 수행 : 이번에 기억할 값.
- 두 값을 더한다.



Cell State

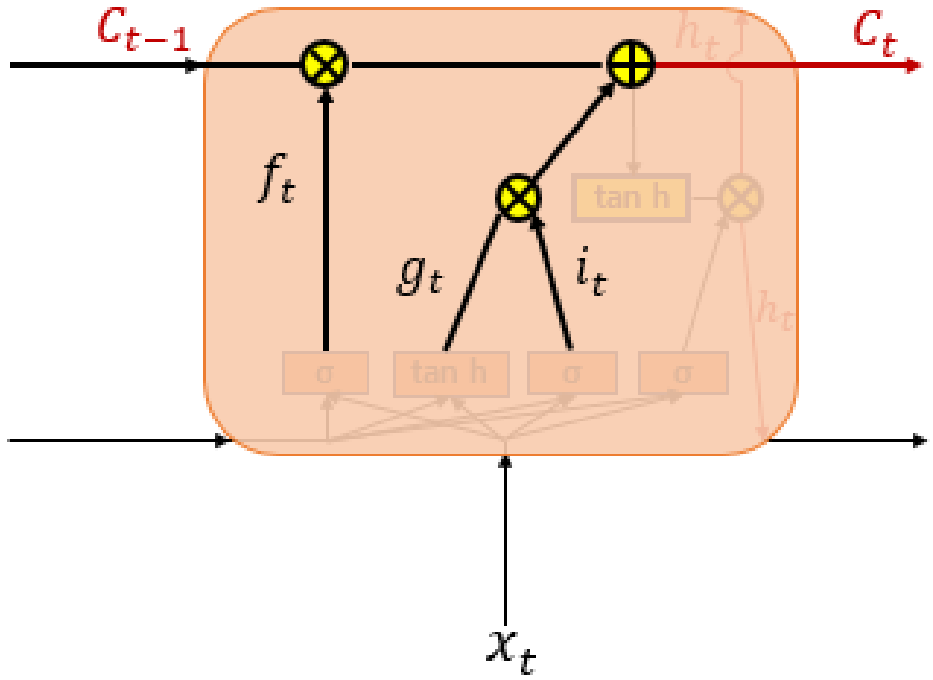
$$C_t = f_t \circ C_{t-1} + i_t \circ g_t$$

f_t가 0이 된다면 이전 시점의 Cell state값이 0이 된다.
오직 입력 게이트만이 현재 시점의 Cell state값을 결정한다.

이는 삭제 게이트가 닫히고, 입력 게이트만 열린 상태를 의미.

셀 상태(Cell state) or 장기 상태

- 삭제 게이트에서 일부 기억을 소실.
- 입력 게이트의 i_t와 g_t를 가지고 elementiwise product를 수행 : 이번에 기억할 값.
- 두 값을 더한다.



Cell State

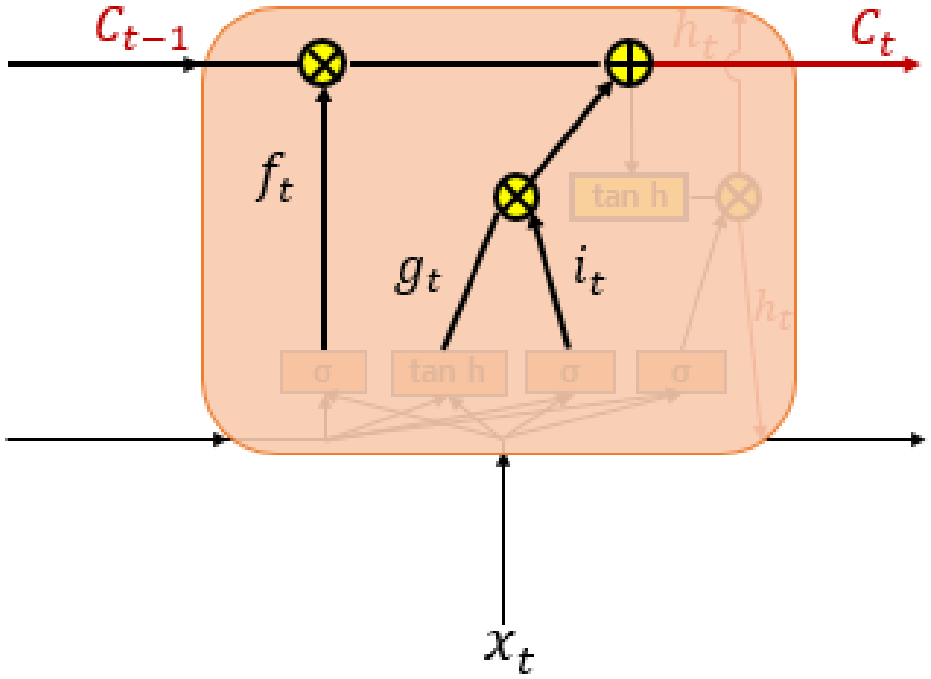
$$C_t = f_t \circ C_{t-1} + i_t \circ g_t$$

i_t가 0이 된다면 이전 시점의 Cell state값으로
현재 시점의 Cell state값을 결정한다.

이는 입력 게이트가 닫히고, 삭제 게이트만 열린 상태를 의미.

셀 상태(Cell state) or 장기 상태

- 삭제 게이트에서 일부 기억을 소실.
- 입력 게이트의 i_t와 g_t를 가지고 elementiwise product를 수행 : 이번에 기억할 값.
- 두 값을 더한다.



Cell State

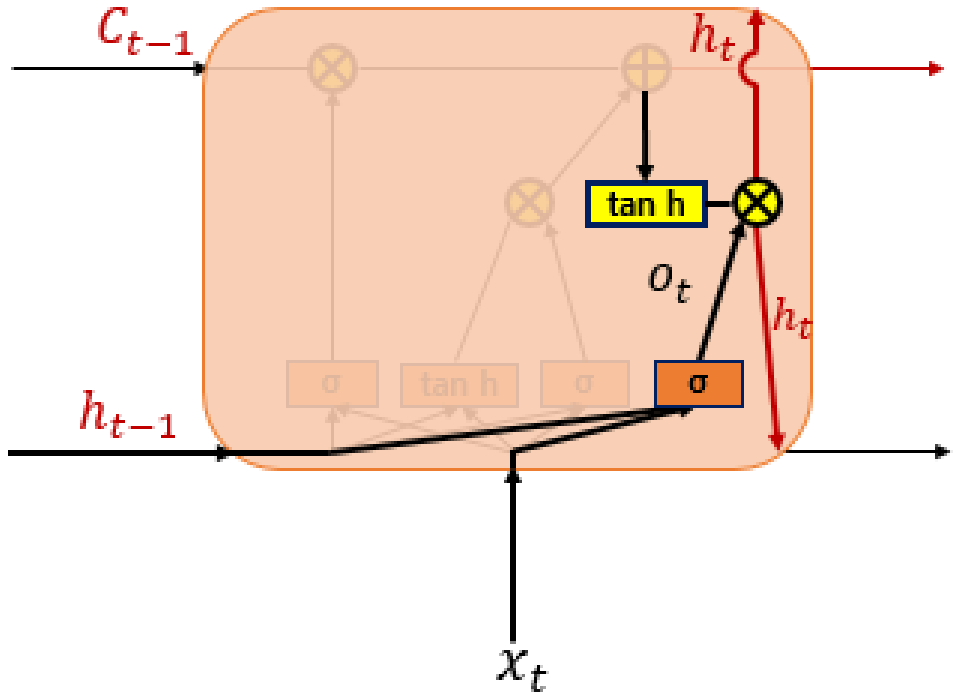
$$C_t = f_t \circ C_{t-1} + i_t \circ g_t$$

삭제 게이트는
이전 시점의 입력을 얼마나 반영할지를 결정한다.

입력 게이트는
현재 시점의 입력을 얼마나 반영할지를 결정한다.

출력 게이트(Output Gate)

- 출력 게이트는 Hidden State를 연산하는 일에 쓰인다.
- Hidden State는 Cell State와 비교하여 단기 상태라고도 부른다.

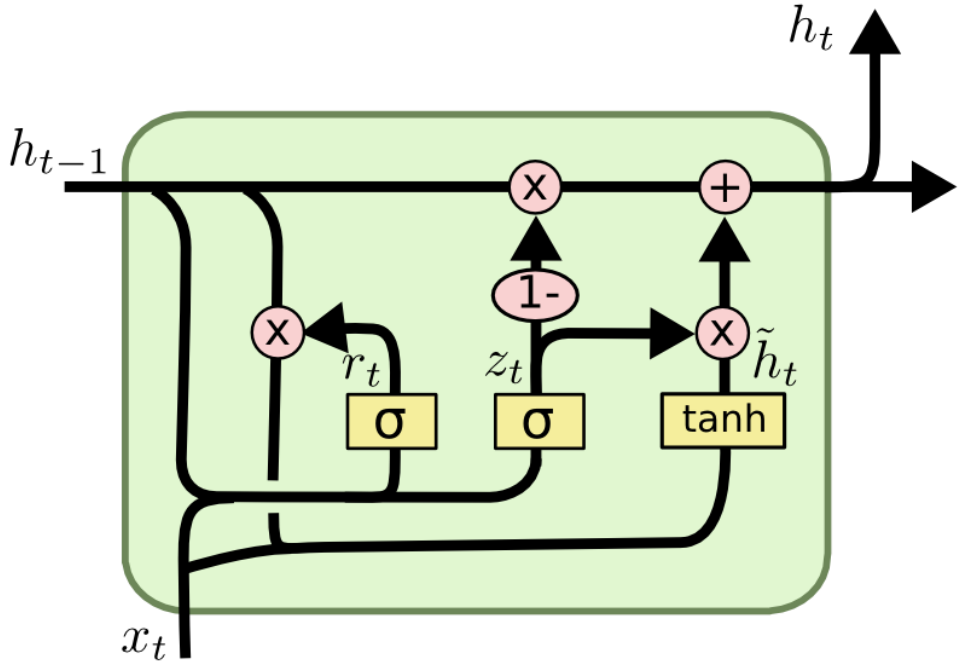


Output Gate / Hidden State

$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + b_o)$$
$$h_t = o_t \circ tanh(c_t)$$

Gated Recurrent Unit, GRU

- 뉴욕대학교 조경현 교수가 제안.
- LSTM과 마찬가지로 장기 의존성 문제에 비해 Vanilla RNN에 강건.
- 3개의 게이트가 있었던 LSTM과 달리 업데이트 게이트, 리셋 게이트로 게이트를 2개로 줄였다.



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

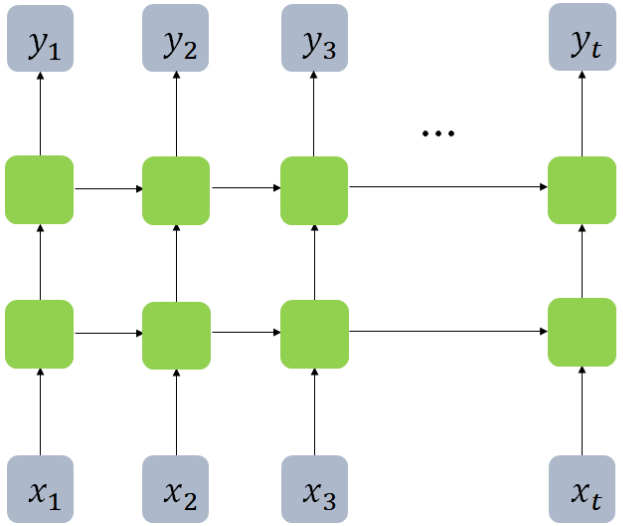
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

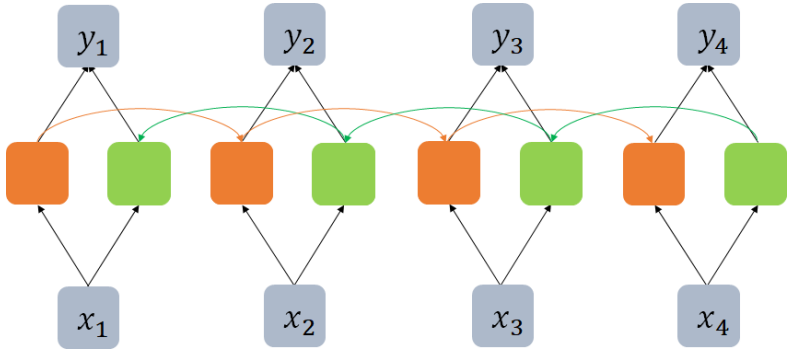
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Deep Bidirectional Recurrent Neural Networks

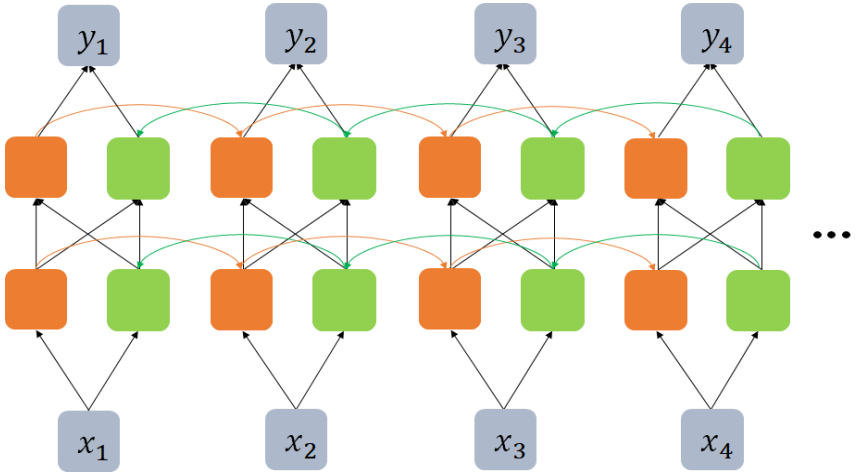
RNN의 은닉층을 높이거나, 역방향으로 입력을 참고하는 RNN을 추가하여 양방향으로 만들 수도 있다.
또는 양방향 RNN의 은닉층을 추가하여 깊은 양방향 RNN을 만들 수도 있다.



Deep RNN



Bidirectional RNN

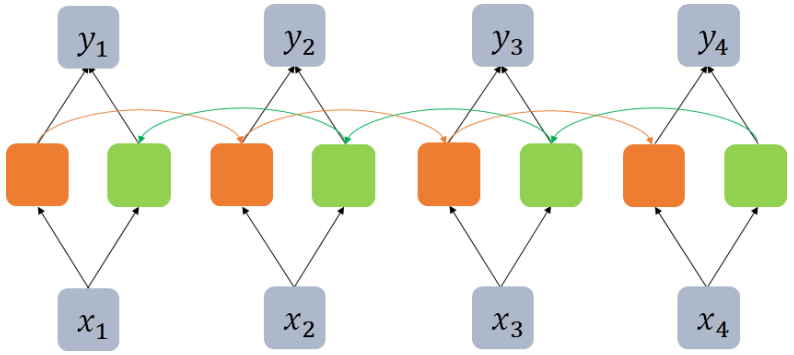


Deep RNN

Bidirectional RNN은 앞의 문맥뿐만 아니라 뒤의 문맥까지 참고할 수 있다는 이점을 가진다.

Bidirectional Recurrent Neural Networks

RNN의 은닉층을 높이거나, 역방향으로 입력을 참고하는 RNN을 추가하여 양방향으로 만들 수도 있다.
또는 양방향 RNN의 은닉층을 추가하여 깊은 양방향 RNN을 만들 수도 있다.



Bidirectional RNN

빈 칸 채우기 문제

Exercise is very effective at [ ] belly fat.

1) Reducing
2) increasing
3) multiplying

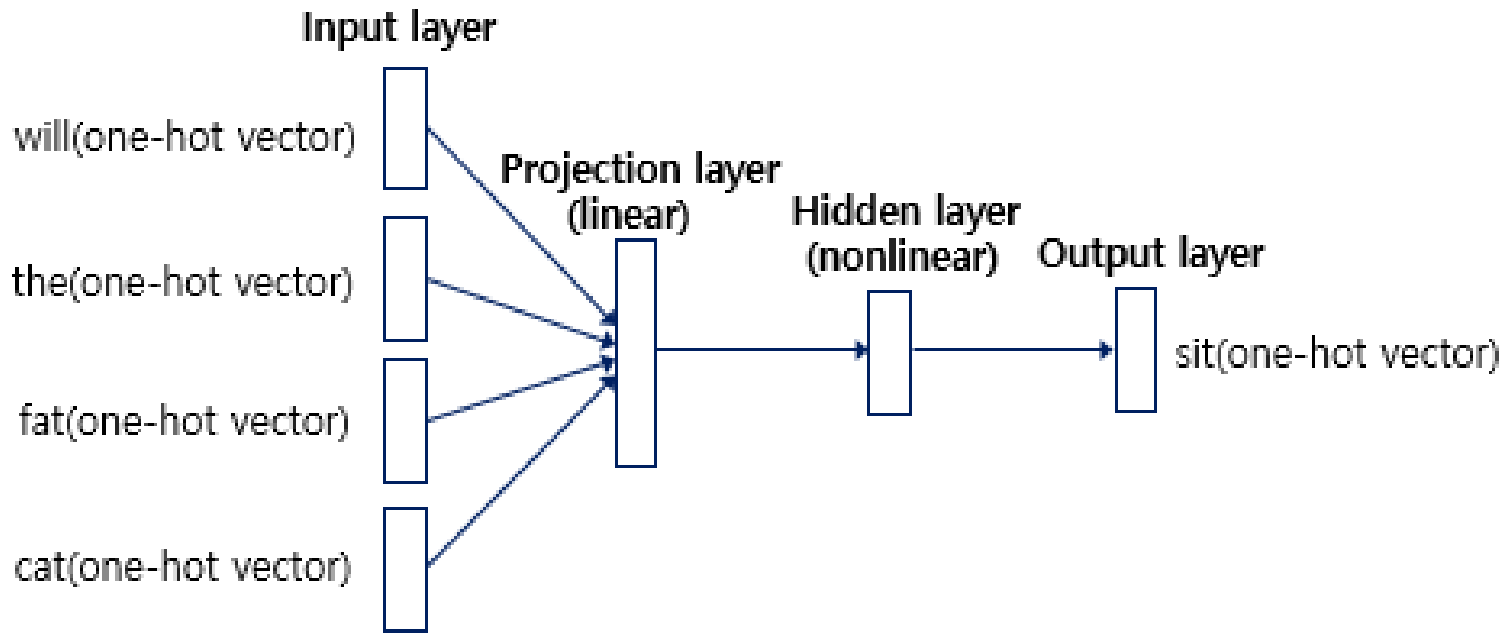
Bidirectional RNN은 앞의 문맥뿐만 아니라 뒤의 문맥까지 참고할 수 있다는 이점을 가진다.

RNN Language Model

NNLM(Neural Network Language Model)

피드 포워드 신경망으로 구현한 Language Model.

Language Model이므로 이전 단어들로부터 다음 단어를 예측한다.



NNLM(Neural Network Language Model)

NNLM은 n개의 이전 단어들로부터 n+1 단어를 예측하는 모델.
신경망의 입력은 원-핫 인코딩을 사용하여 얻은 원-핫 벡터로 한다.
다음과 같은 예문을 훈련한다고 해보자.

예문 : "what will the fat cat sit on"

원-핫 인코딩

what = [1, 0, 0, 0, 0, 0, 0]
will = [0, 1, 0, 0, 0, 0, 0]
the = [0, 0, 1, 0, 0, 0, 0]
fat = [0, 0, 0, 1, 0, 0, 0]
cat = [0, 0, 0, 0, 1, 0, 0]
sit = [0, 0, 0, 0, 0, 1, 0]
on = [0, 0, 0, 0, 0, 0, 1]

NNLM(Neural Network Language Model)

NNLM은 n개의 이전 단어들로부터 n+1 단어를 예측하는 모델.
신경망의 입력은 원-핫 인코딩을 사용하여 얻은 원-핫 벡터로 한다.
다음과 같은 예문을 훈련한다고 해보자.

예문 : "what will the fat cat sit on"

↓ **원-핫 인코딩**

what = [1, 0, 0, 0, 0, 0, 0]
will = [0, 1, 0, 0, 0, 0, 0]
the = [0, 0, 1, 0, 0, 0, 0]
fat = [0, 0, 0, 1, 0, 0, 0]
cat = [0, 0, 0, 0, 1, 0, 0]
sit = [0, 0, 0, 0, 0, 1, 0]
on = [0, 0, 0, 0, 0, 0, 1]

sit을 예측한다고 하면 n은 4이므로 will, the, fat, cat으로부터 sit을 예측한다.

NNLM(Neural Network Language Model)

NNLM은 n개의 이전 단어들로부터 n+1 단어를 예측하는 모델.
신경망의 입력은 원-핫 인코딩을 사용하여 얻은 원-핫 벡터로 한다.
다음과 같은 예문을 훈련한다고 해보자.

예문 : "what will the fat cat sit on"

↓ **원-핫 인코딩**

sit을 예측한다고 하면 n은 4이므로 will, the, fat, cat으로부터 sit을 예측한다.

what = [1, 0, 0, 0, 0, 0, 0]
will = [0, 1, 0, 0, 0, 0, 0]
the = [0, 0, 1, 0, 0, 0, 0]
fat = [0, 0, 0, 1, 0, 0, 0]
cat = [0, 0, 0, 0, 1, 0, 0]
sit = [0, 0, 0, 0, 0, 1, 0]
on = [0, 0, 0, 0, 0, 0, 1]

→

입력

will = [0, 1, 0, 0, 0, 0, 0]
the = [0, 0, 1, 0, 0, 0, 0]
fat = [0, 0, 0, 1, 0, 0, 0]
cat = [0, 0, 0, 0, 1, 0, 0]

NNLM(Neural Network Language Model)

NNLM은 n개의 이전 단어들로부터 n+1 단어를 예측하는 모델.
신경망의 입력은 원-핫 인코딩을 사용하여 얻은 원-핫 벡터로 한다.
다음과 같은 예문을 훈련한다고 해보자.

예문 : "what will the fat cat sit on"

원-핫 인코딩

sit을 예측한다고 하면 n은 4이므로 will, the, fat, cat으로부터 sit을 예측한다.

what = [1, 0, 0, 0, 0, 0, 0]
will = [0, 1, 0, 0, 0, 0, 0]
the = [0, 0, 1, 0, 0, 0, 0]
fat = [0, 0, 0, 1, 0, 0, 0]
cat = [0, 0, 0, 0, 1, 0, 0]
sit = [0, 0, 0, 0, 0, 1, 0]
on = [0, 0, 0, 0, 0, 0, 1]

입력

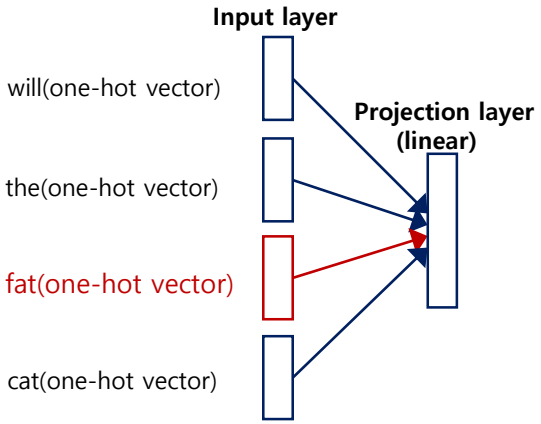
will = [0, 1, 0, 0, 0, 0, 0]
the = [0, 0, 1, 0, 0, 0, 0]
fat = [0, 0, 0, 1, 0, 0, 0]
cat = [0, 0, 0, 0, 1, 0, 0]

예측 대상(출력)

sit = [0, 0, 0, 0, 0, 1, 0]

Projection layer

- V차원의 원-핫 벡터는 V x M 크기의 행렬과 곱해져 M차원의 벡터를 얻는다.
- i번째 인덱스에 1의 값을 가지는 원-핫 벡터와 가중치 W 행렬의 곱은 W행렬의 i번째 행을 그대로 읽어오는 것과(lookup) 동일하다.
- 이 연산을 **lookup table**이라 한다.



$$x_{fat} \times W_{V \times M} = e_{fat}$$

0	0	0	1	0	0	0

×

0.5	2.1	1.9	1.5	0.8
0.8	1.2	2.8	1.8	2.1
0.1	0.8	1.2	0.9	0.7
2.1	1.8	1.5	1.7	2.7

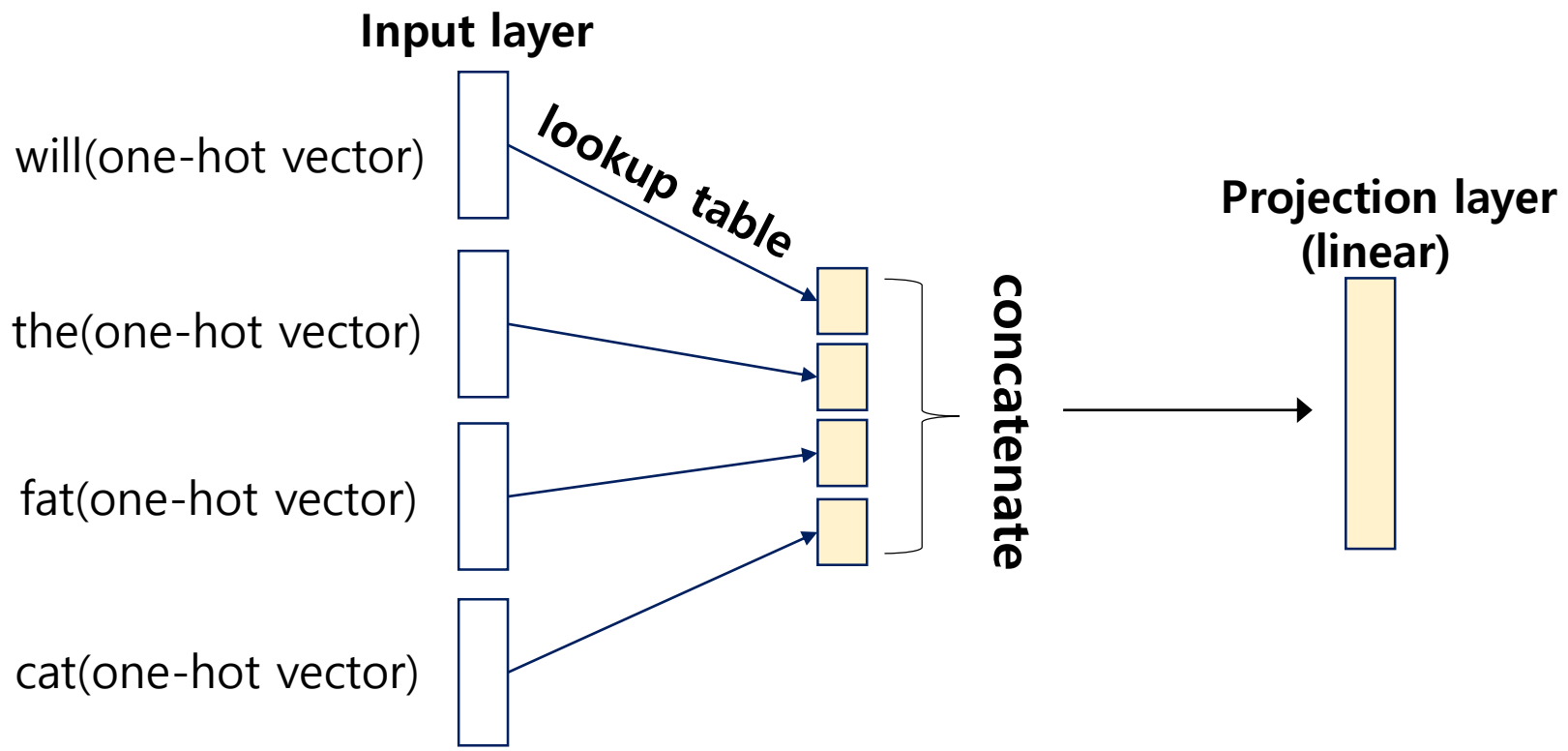
=

2.1	1.8	1.5	1.7	2.7

lookup table

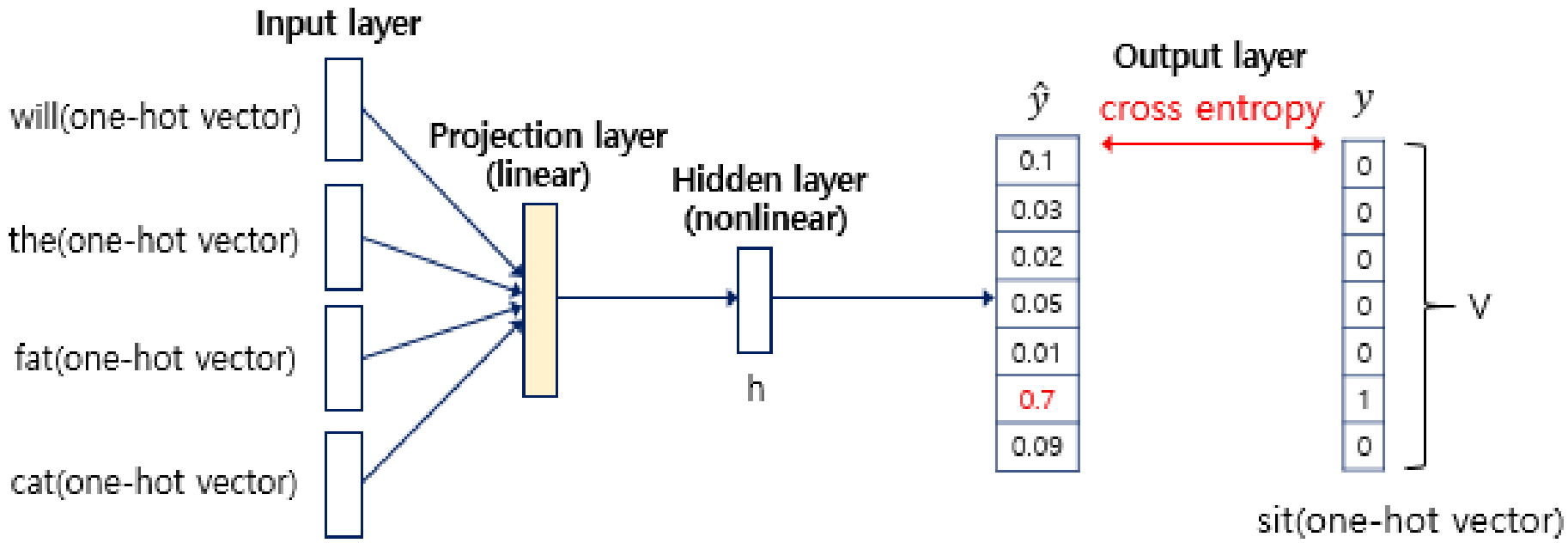
Projection layer

각 원-핫 벡터가 lookup table을 거치면 모두 concatenate된다.



Hidden layer and Output layer

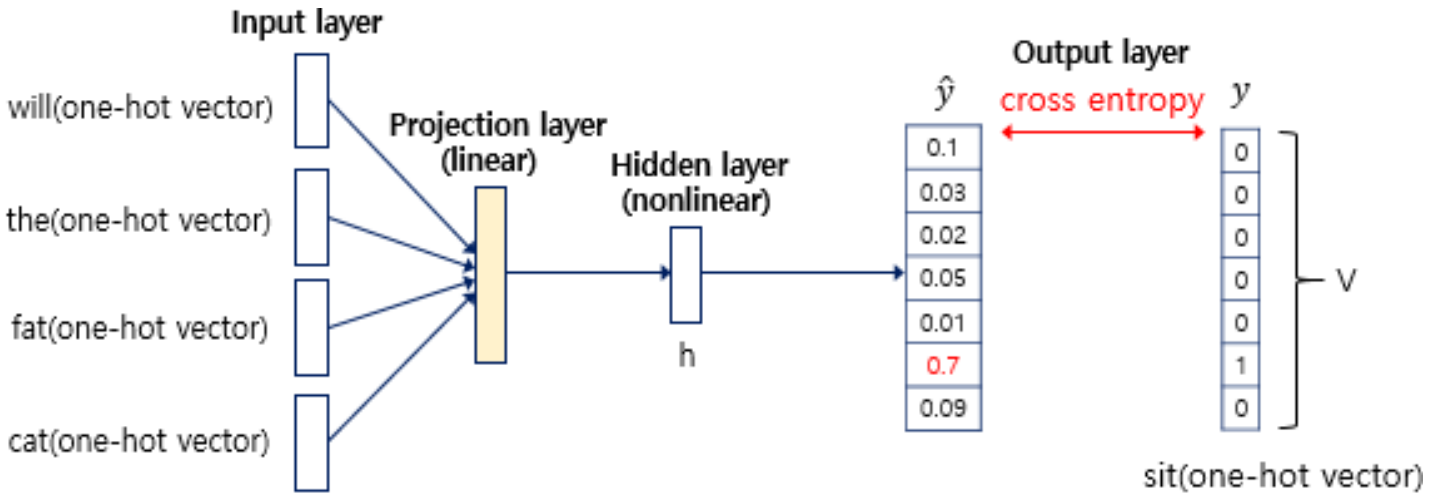
Projection layer 이후, 하나의 Hidden layer를 거치고 Output layer에서 실제값인 다음 단어 원-핫 벡터로부터 loss를 구하고 학습된다.
이 과정에서 Projection layer의 embedding table이 학습된다.



NNLM(Neural Network Language Model)

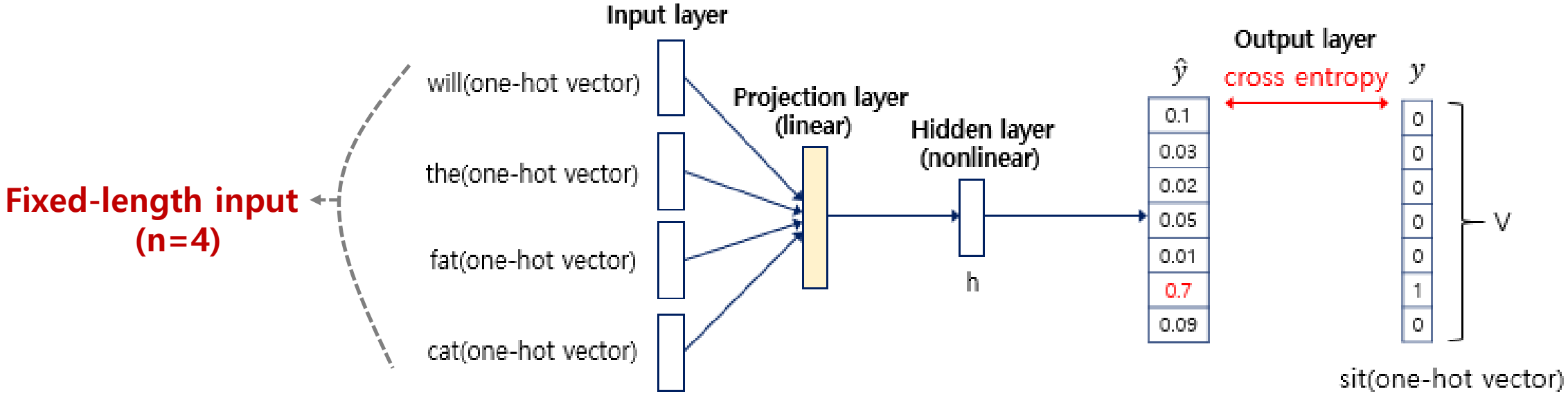
투사층의 가중치 행렬 W의 각 행은 각 단어와 맵핑되는 밀집 벡터(Dense Vector)이다.
훈련 과정에서 유사한 용도로 사용되는 단어들은 유사한 단어 벡터값을 가지게 된다.

ex) 만약 a cute dog를 학습했다면, a cute cat이 훈련 데이터에 없더라도 a cute cat을 생성 가능!



NNLM의 한계

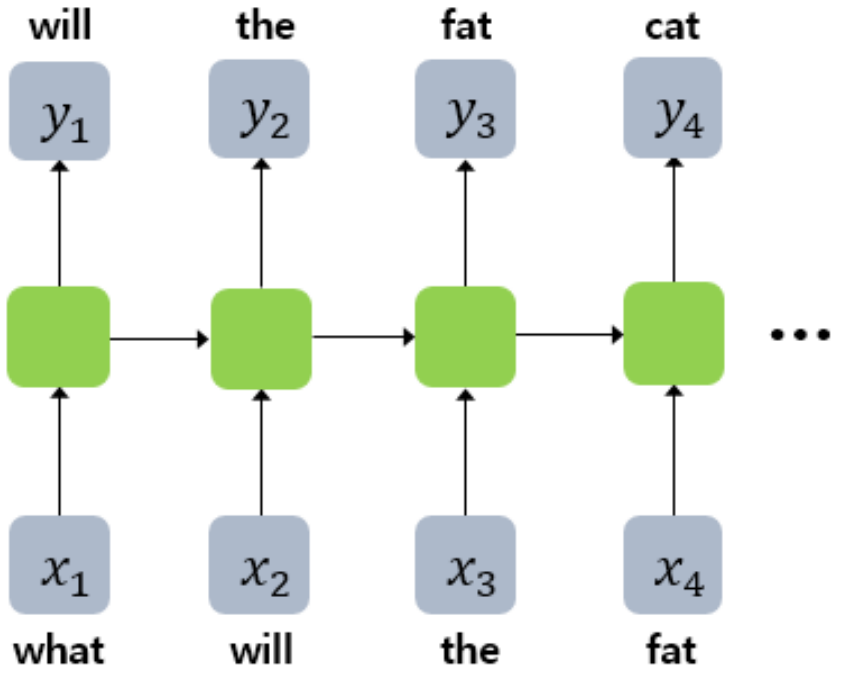
NNLM은 오직 n개의 고정된 길이의 입력을 받을 수 있다. (N-gram Language Model과 같음.)
문장의 길이가 길어지고, 앞의 문장에 중요한 정보가 있다면 이를 충분히 반영할 수 없다.



RNN Language Model

- RNN으로 구현한 언어 모델.
- Language Model이므로 이전 단어들로부터 다음 단어를 예측한다.
- 입력 길이가 고정되지 않는다.

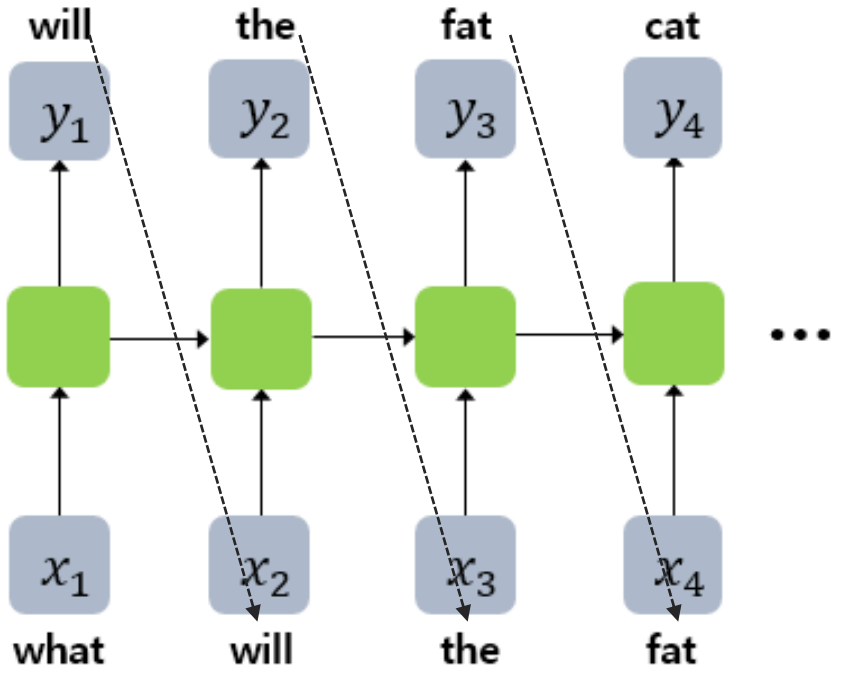
예문 : "what will the fat cat sit on"



RNN Language Model

- RNN으로 구현한 언어 모델.
- Language Model이므로 이전 단어들로부터 다음 단어를 예측한다.
- 입력 길이가 고정되지 않는다.

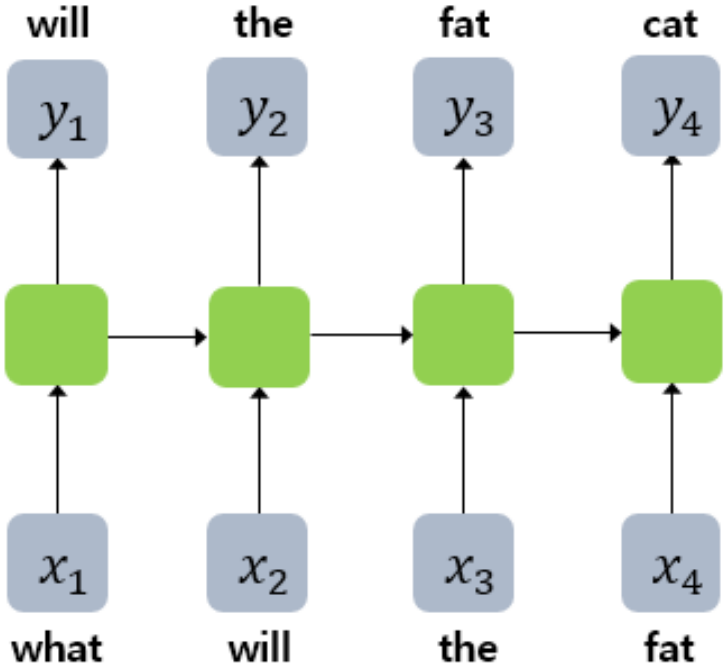
예문 : "what will the fat cat sit on"



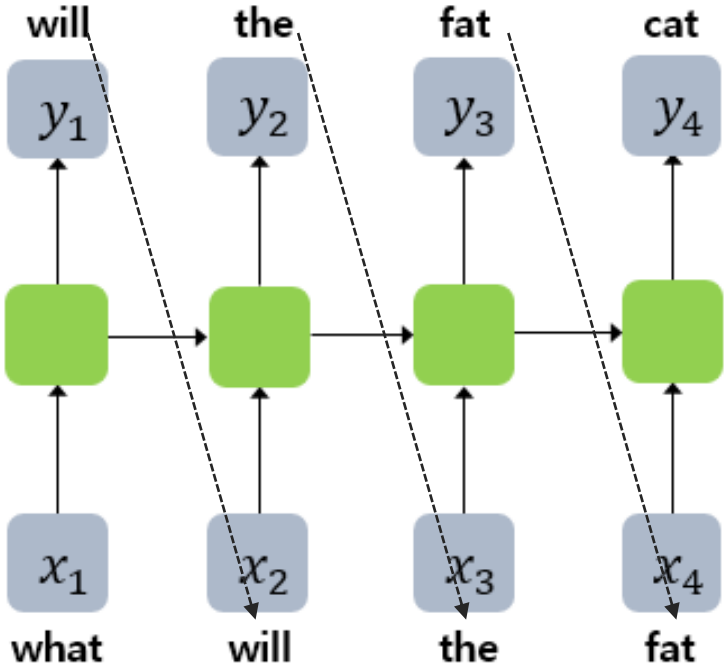
Teacher Forcing(교사 강요)

- 훈련 시에도 예측값을 입력으로 사용하면 훈련 시간이 지나치게 오래 걸린다.
- 훈련 시에는 실제값을 입력으로 사용한다.
- 훈련 시에는 what will the fat을 입력하면 will the fat cat이 예측되도록 훈련.

학습 과정

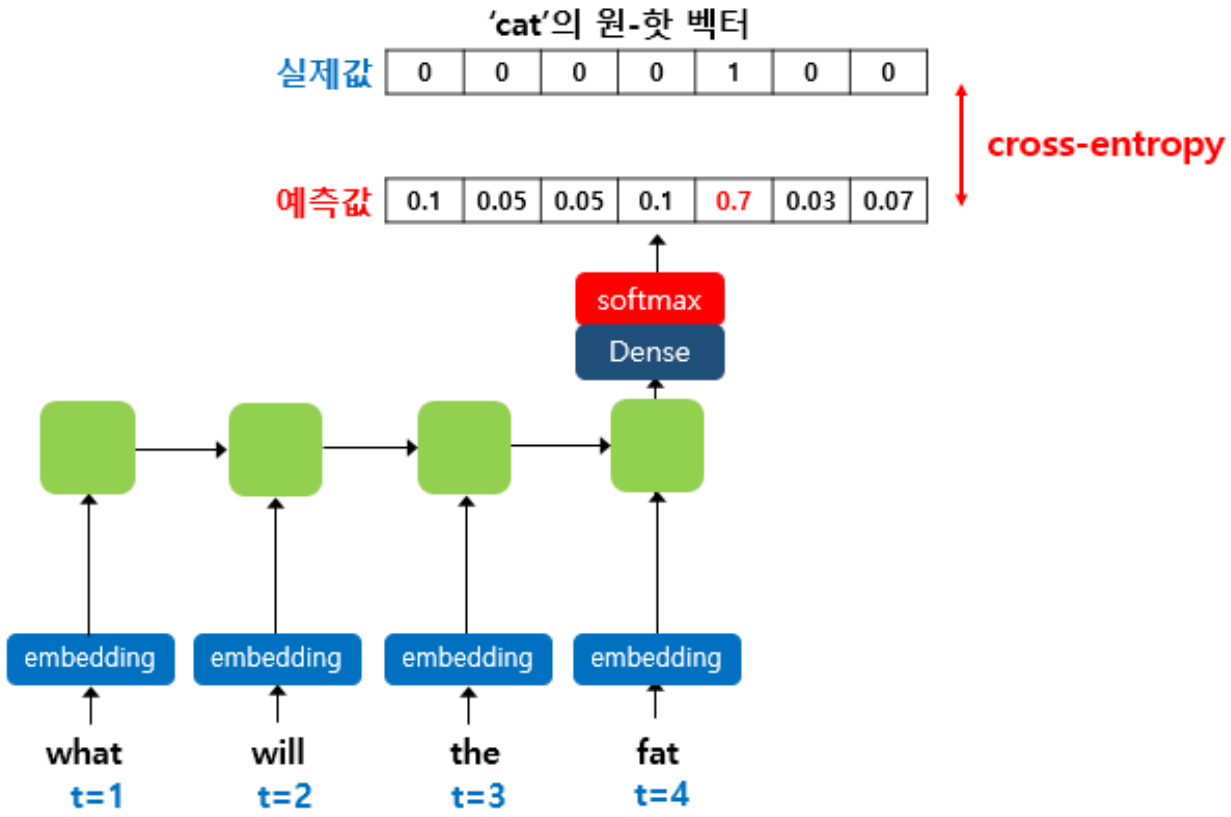


테스트 과정



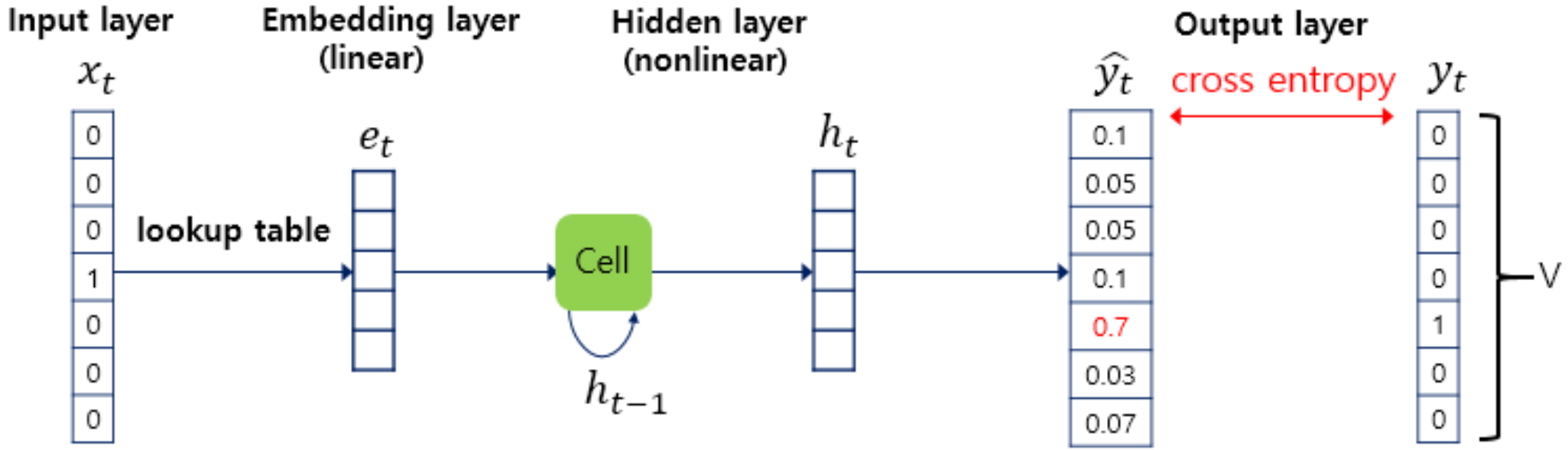
RNN Language Model

- 구체적인 아키텍처 : Embedding layer, Hidden layer, Output layer로 구성.
- Output layer에서는 Vocab size의 벡터로 다중 클래스 분류를 수행.



RNN Language Model

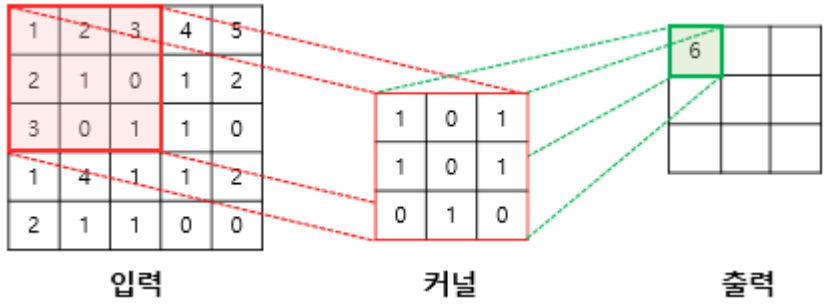
- 구체적인 아키텍처 : Embedding layer, Hidden layer, Output layer로 구성.
- Output layer에서는 Vocab size의 벡터로 다중 클래스 분류를 수행.



Convolutional Neural Network

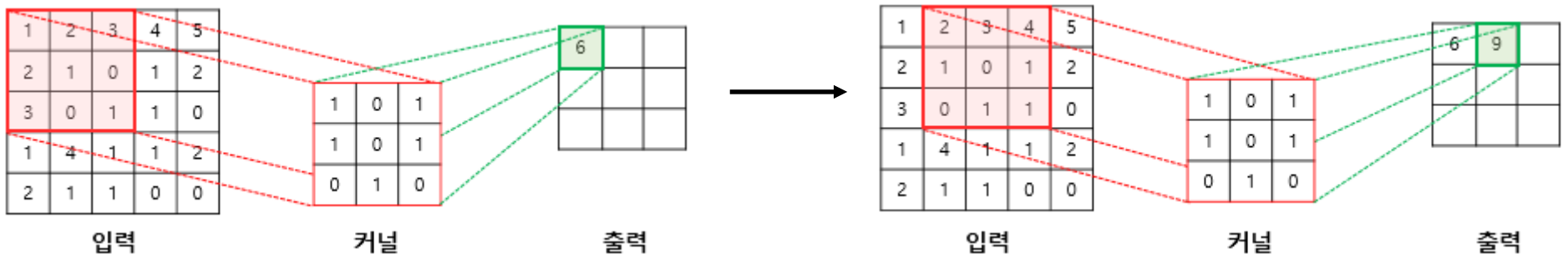
2D Convolution (in Image Processing)

- 합성곱 연산은 이미지의 특징을 추출하는 역할을 한다.
- 커널(kernel) 또는 필터(filter)라는 행렬로 이미지의 가장 왼쪽 위부터 가장 오른쪽 아래까지
  순차적으로 훑으며 겹쳐지는 부분의 각 이미지와 커널의 원소의 값을 곱해서 모두 더한 값을 출력으로 한다.



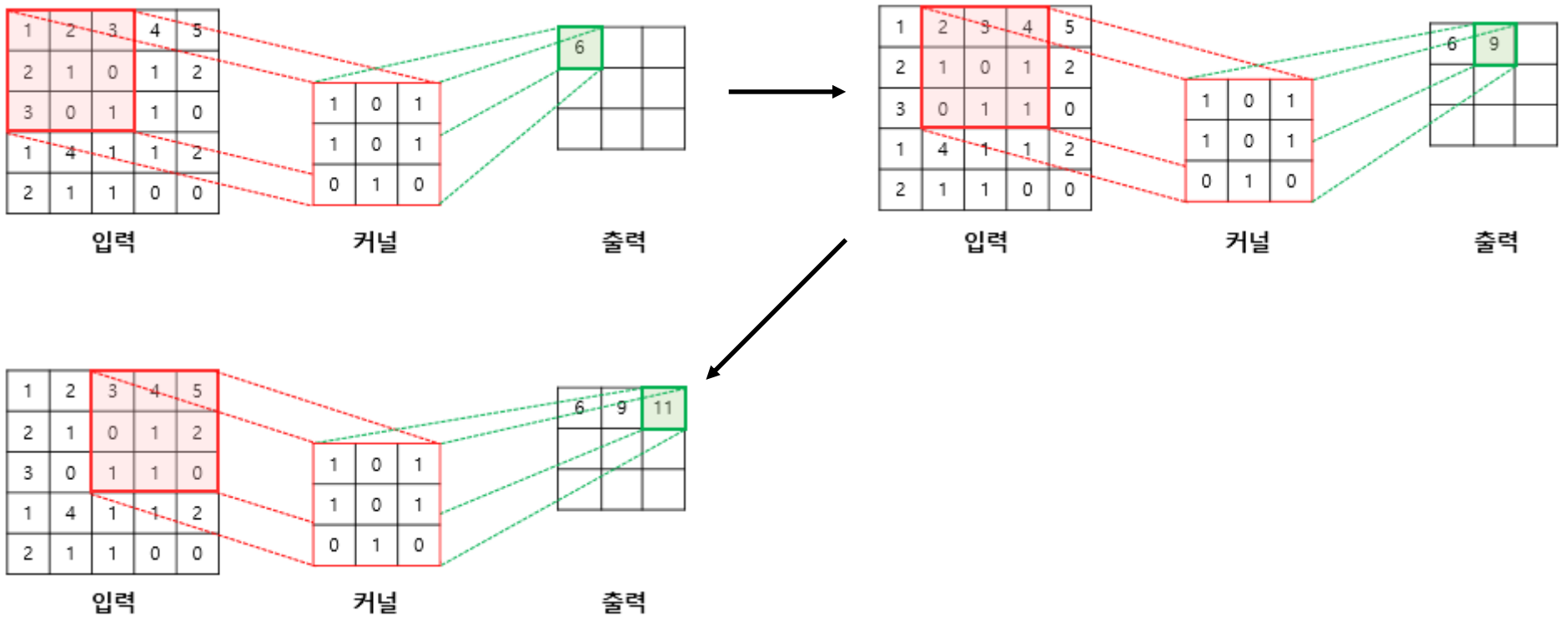
2D Convolution (in Image Processing)

- 합성곱 연산은 이미지의 특징을 추출하는 역할을 한다.
- 커널(kernel) 또는 필터(filter)라는 행렬로 이미지의 가장 왼쪽 위부터 가장 오른쪽 아래까지 순차적으로 훑으며 겹쳐지는 부분의 각 이미지와 커널의 원소의 값을 곱해서 모두 더한 값을 출력으로 한다.



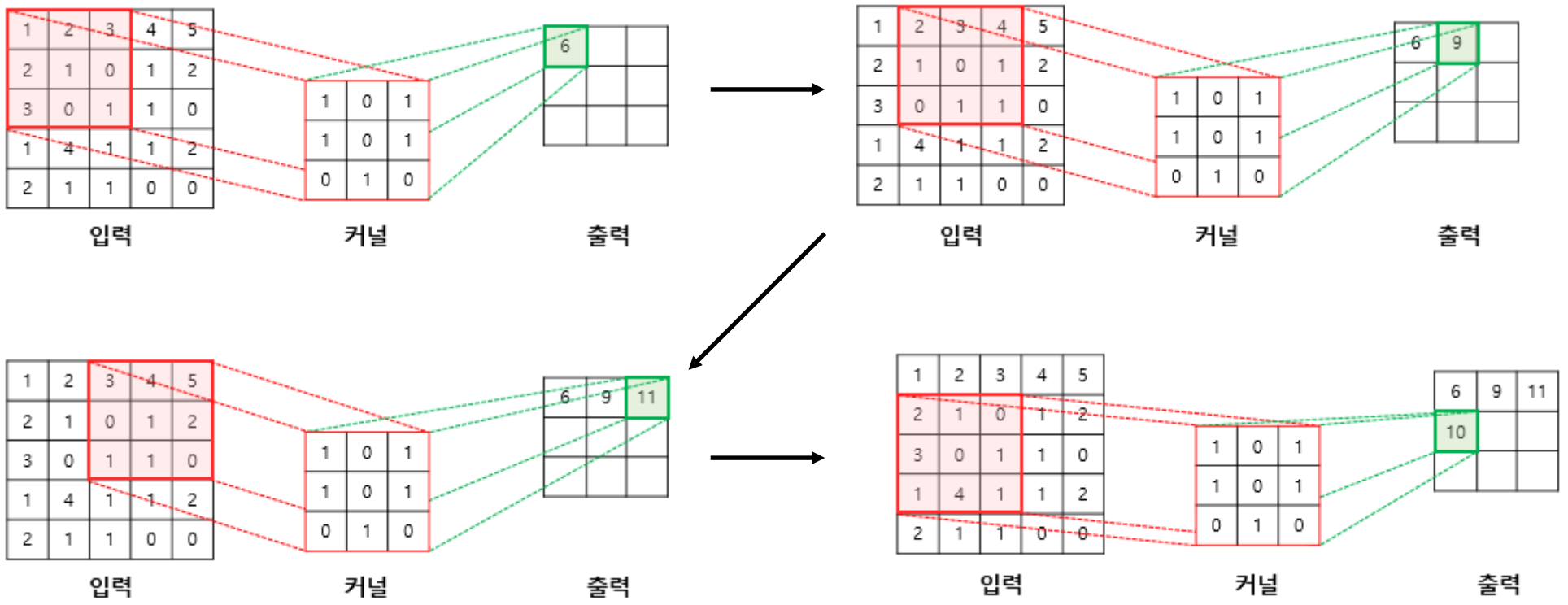
2D Convolution (in Image Processing)

- 합성곱 연산은 이미지의 특징을 추출하는 역할을 한다.
- 커널(kernel) 또는 필터(filter)라는 행렬로 이미지의 가장 왼쪽 위부터 가장 오른쪽 아래까지 순차적으로 훑으며 겹쳐지는 부분의 각 이미지와 커널의 원소의 값을 곱해서 모두 더한 값을 출력으로 한다.



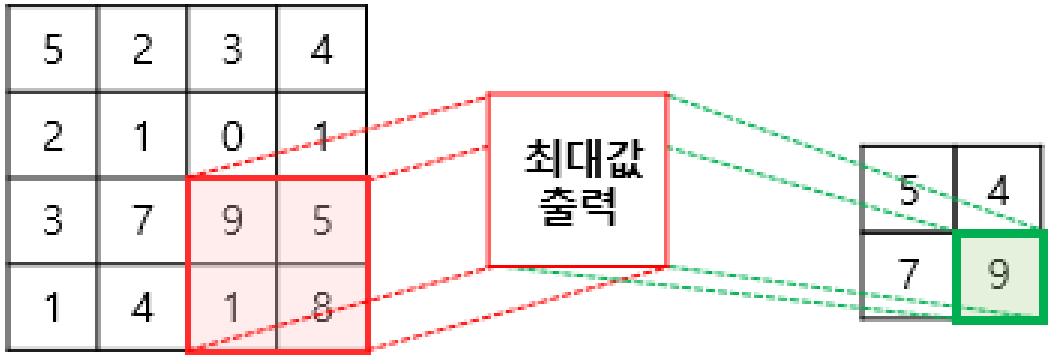
2D Convolution (in Image Processing)

- 합성곱 연산은 이미지의 특징을 추출하는 역할을 한다.
- 커널(kernel) 또는 필터(filter)라는 행렬로 이미지의 가장 왼쪽 위부터 가장 오른쪽 아래까지 순차적으로 훑으며 겹쳐지는 부분의 각 이미지와 커널의 원소의 값을 곱해서 모두 더한 값을 출력으로 한다.



2D Max-pooling (in Image Processing)

- 일반적으로 합성곱 층(합성곱 연산 + 활성화 함수) 다음에는 풀링 층을 추가하는 것이 일반적이다.
- 맥스 풀링은 커널과 겹치는 영역 안에서 최대값을 추출하는 방식으로 다운샘플링한다.
- 가장 중요한 특징(feature)를 추출하기 위해서 수행한다.



Stride = 2
2 × 2 크기 커널, 맥스 풀링

CNN for Text Classification

- Yoon Kim(2014)
- 매우 단순한 구조임에도 준수한 성능을 보인 모델.
- Convolutional layer를 오직 하나의 층만을 사용.

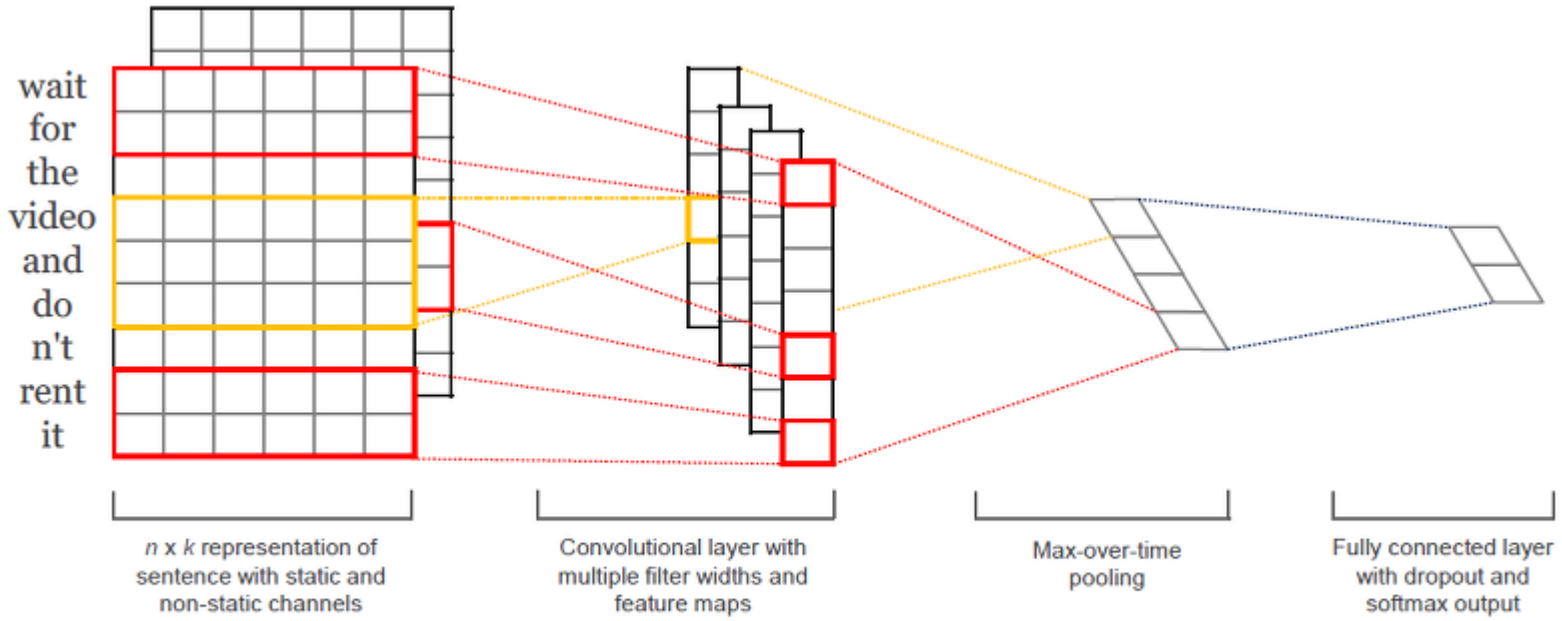


Figure 1: Model architecture with two channels for an example sentence.

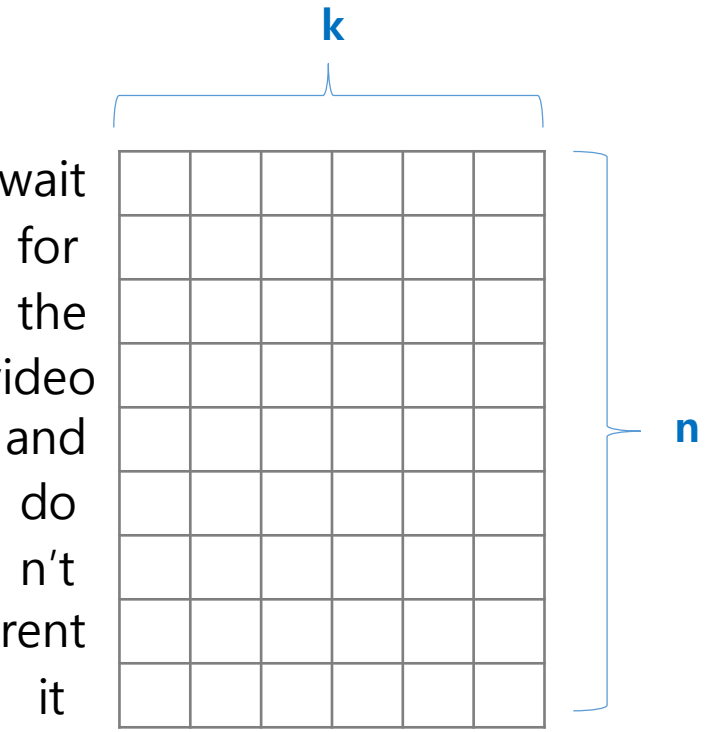
CNN for Text Classification

- **CNN의 입력은 문장 또는 문서의 각 단어가 Embedding을 거치고 난 후의 문장 테이블.**
- **n을 문장 길이, k를 임베딩 벡터의 차원이라고 하였을 때, CNN의 입력은 다음과 같이 n × k 행렬.**

입력 문장

wait for the video and don't rent it

Tokenization / Padding / Embedding



CNN for Text Classification

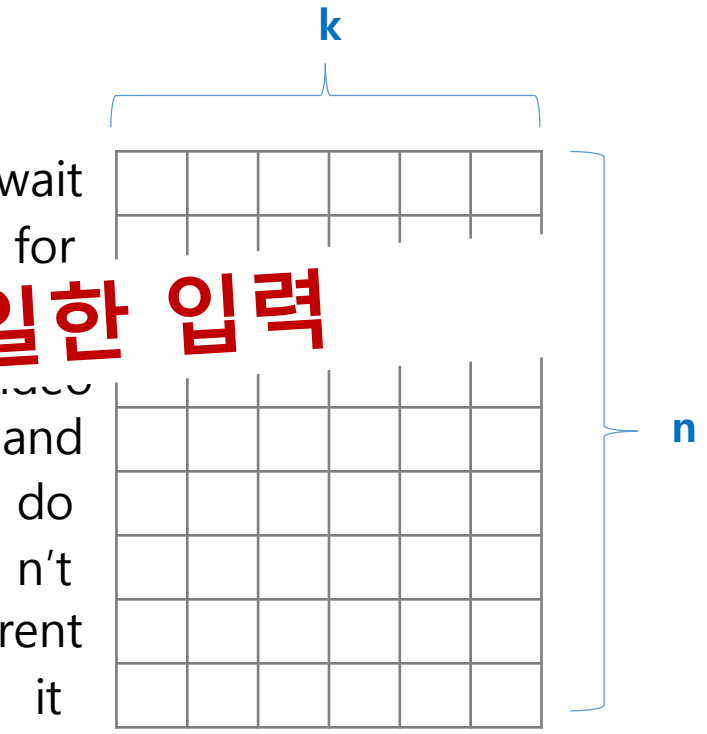
- **CNN의 입력은 문장 또는 문서의 각 단어가 Embedding을 거치고 난 후의 문장 테이블.**
- **n을 문장 길이, k를 임베딩 벡터의 차원이라고 하였을 때, CNN의 입력은 다음과 같이 n × k 행렬.**

입력 문장

wait for the video and

Tokenization /

사실 이는 RNN과 동일한 입력



CNN for Text Classification

n : 문장 또는 문서의 길이.

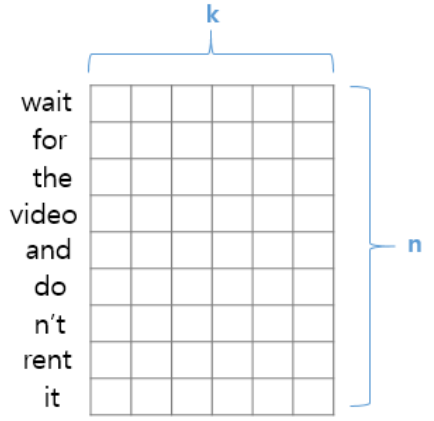
- **padding : n보다 길이가 짧은 문장은 제로 패딩, 길이가 긴 문장은 trim**

k : 임베딩 벡터의 차원.

- **embedding : 랜덤 초기화 후 학습 or Pre-trained Embedding(Word2Vec, GloVe 등)**
- **static : Pre-Trained Embedding을 추가 학습 시키면 non-static, 시키지 않으면 static**

그렇다면 임베딩 벡터를 사용할 때 선택할 수 있는 경우의 수는 세 가지!

1. **CNN–rand : 랜덤 초기화**
2. **CNN-static : pre-trained embedding 추가 훈련 x**
3. **CNN-non-static : pre-trained embedding 추가 훈련 o**



CNN for Text Classification

n : 문장 또는 문서의 길이.

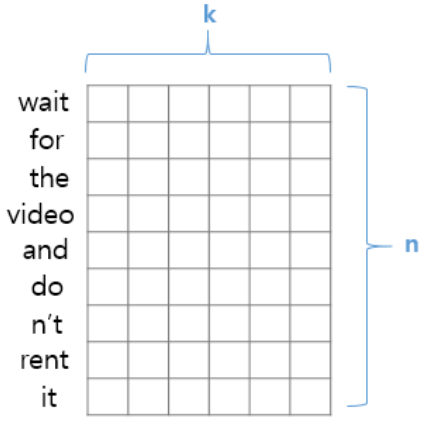
- **padding : n보다 길이가 짧은 문장은 제로 패딩, 길이가 긴 문장은 trim**

k : 임베딩 벡터의 차원.

- **embedding : 랜덤 초기화 후 학습 or Pre-trained Embedding(Word2Vec, GloVe 등)**
- **static : Pre-Trained Embedding을 추가 학습 시키면 non-static, 시키지 않으면 static**

그렇다면 임베딩 벡터를 사용할 때 선택할 수 있는 경우의 수는 네 가지!

1. **CNN–rand : 랜덤 초기화**
2. **CNN-static : pre-trained embedding 추가 훈련 x**
3. **CNN-non-static : pre-trained embedding 추가 훈련 o**
4. **CNN-multichannel : 임베딩 행렬을 여러 개 사용. 예를 들어 2번 + 3번.**



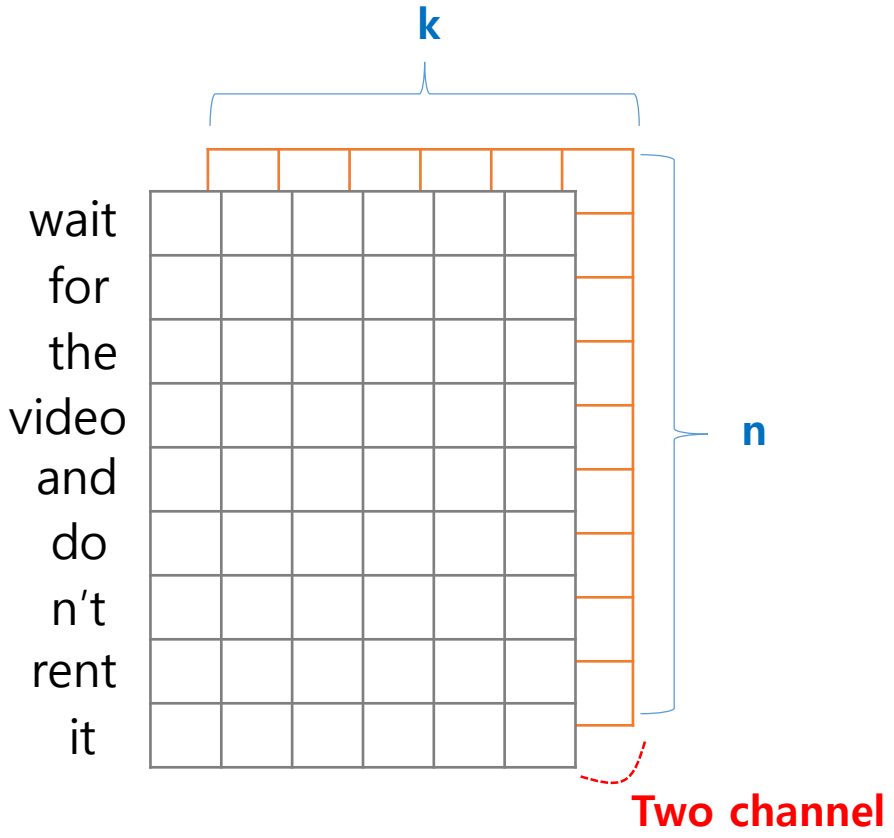
CNN-Multichannel

- CNN의 입력은 문장 또는 문서의 각 단어가 Embedding을 거치고 난 후의 문장 테이블.
- n을 문장 길이, k를 임베딩 벡터의 차원이라고 하였을 때, CNN의 입력은 다음과 같이 n × k 행렬.

문장

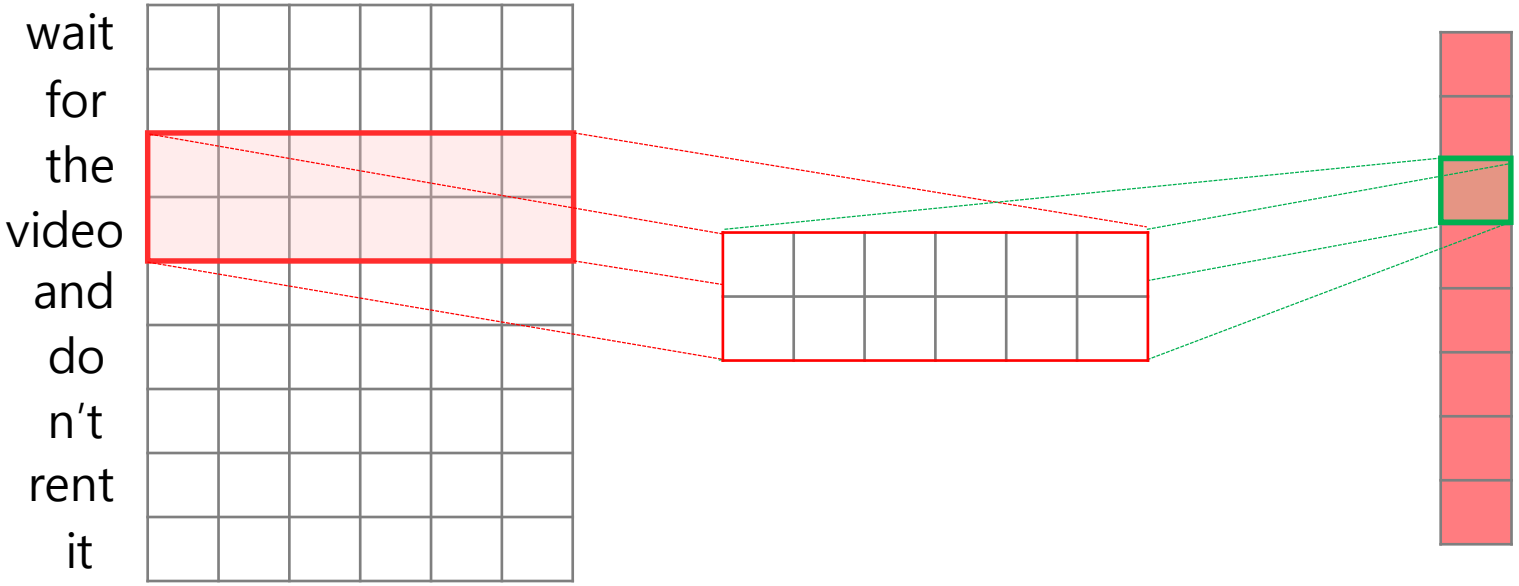
wait for the video and don't rent it

Tokenization / Padding / Embedding



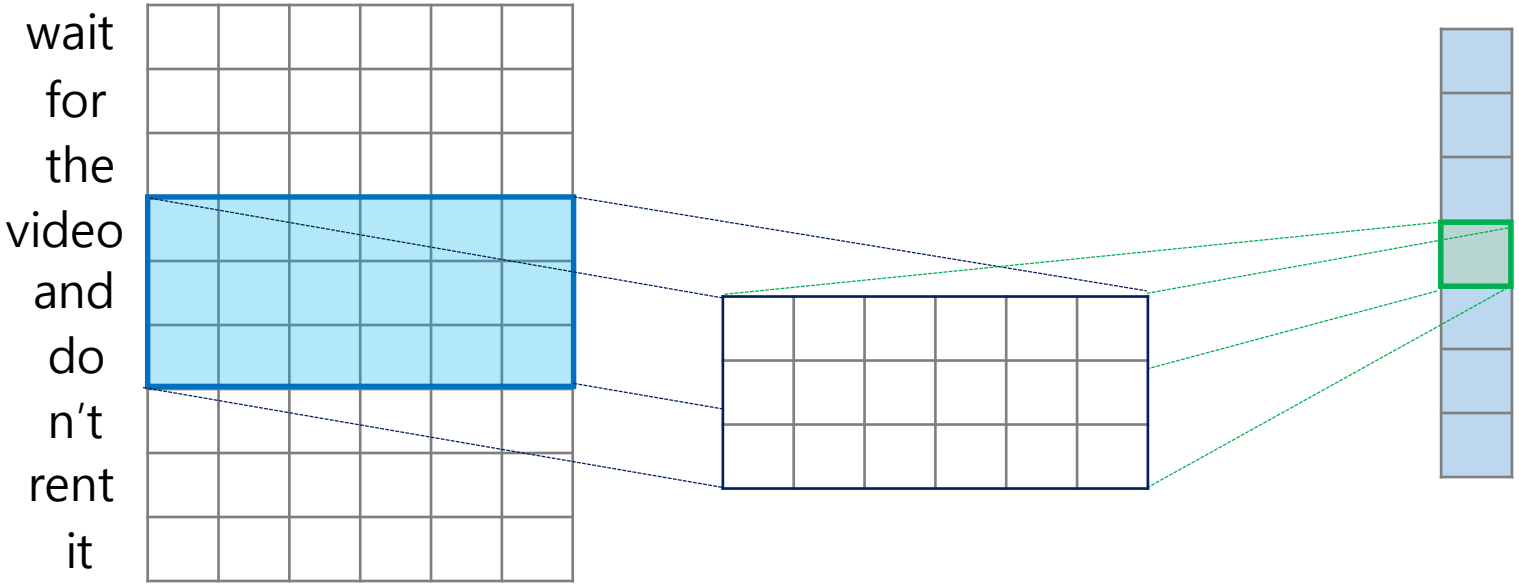
1D Convolution (in NLP)

- 오직 한 방향으로만 윈도우를 슬라이딩하므로 1D라고 한다.
- 커널(kernel) 또는 필터(filter)라는 행렬로 임베딩 행렬의 가장 맨 위부터 가장 맨 아래까지 (↓)
  순차적으로 훑으며 겹쳐지는 부분의 임베딩 행렬과 커널의 원소의 값을 곱해서 모두 더한 값을 출력으로 한다.



1D Convolution (in NLP)

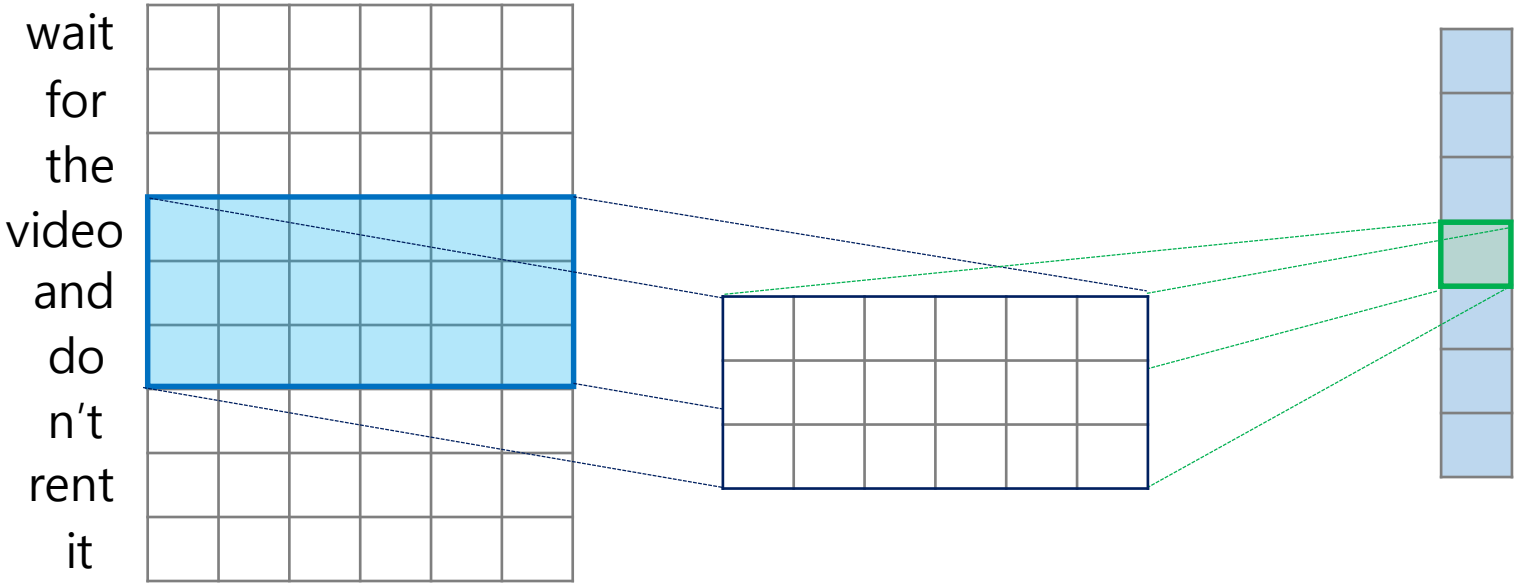
- 오직 한 방향으로만 윈도우를 슬라이딩하므로 1D라고 한다.
- 커널(kernel) 또는 필터(filter)라는 행렬로 임베딩 행렬의 가장 맨 위부터 가장 맨 아래까지 (↓)
  순차적으로 훑으며 겹쳐지는 부분의 임베딩 행렬과 커널의 원소의 값을 곱해서 모두 더한 값을 출력으로 한다.



커널의 크기가 커질수록, 한 번에 좀 더 많은 단어를 참고한다.

1D Convolution (in NLP)

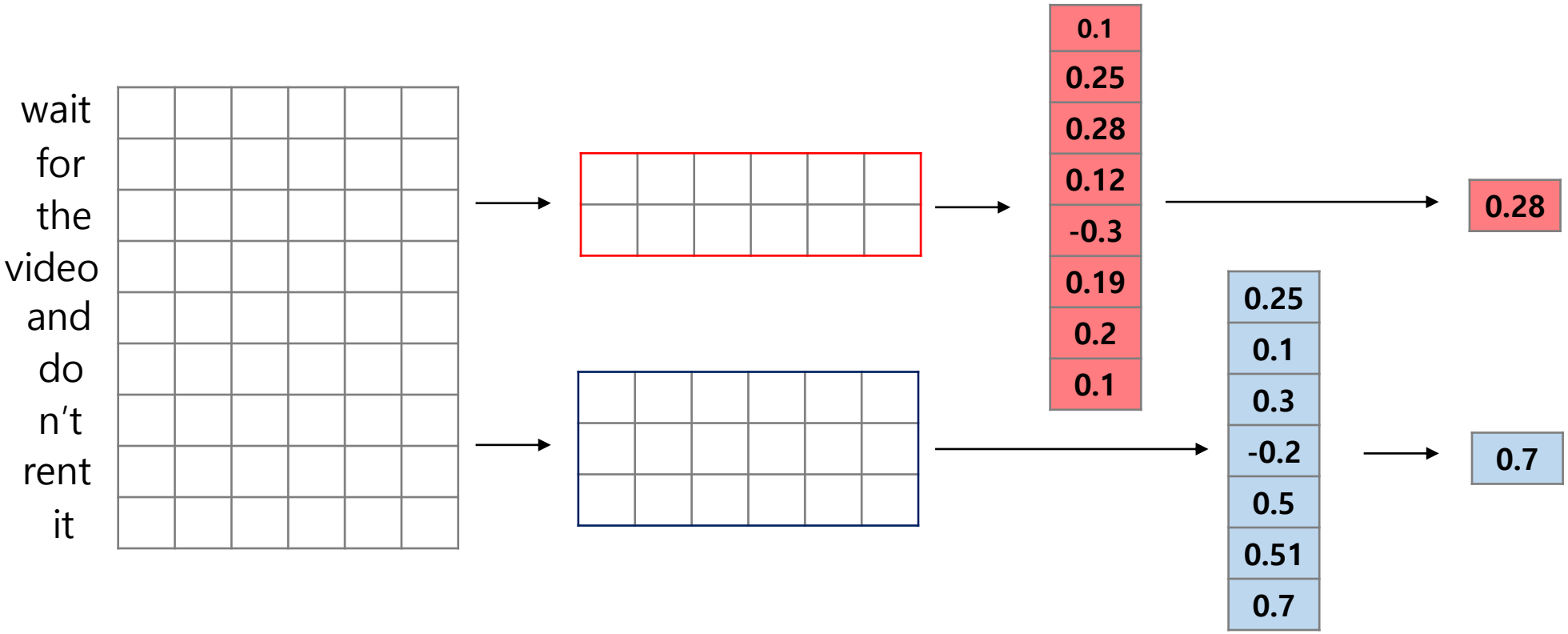
- 오직 한 방향으로만 윈도우를 슬라이딩하므로 1D라고 한다.
- 커널(kernel) 또는 필터(filter)라는 행렬로 임베딩 행렬의 가장 맨 위부터 가장 맨 아래까지 (↓)
  순차적으로 훑으며 겹쳐지는 부분의 임베딩 행렬과 커널의 원소의 값을 곱해서 모두 더한 값을 출력으로 한다.



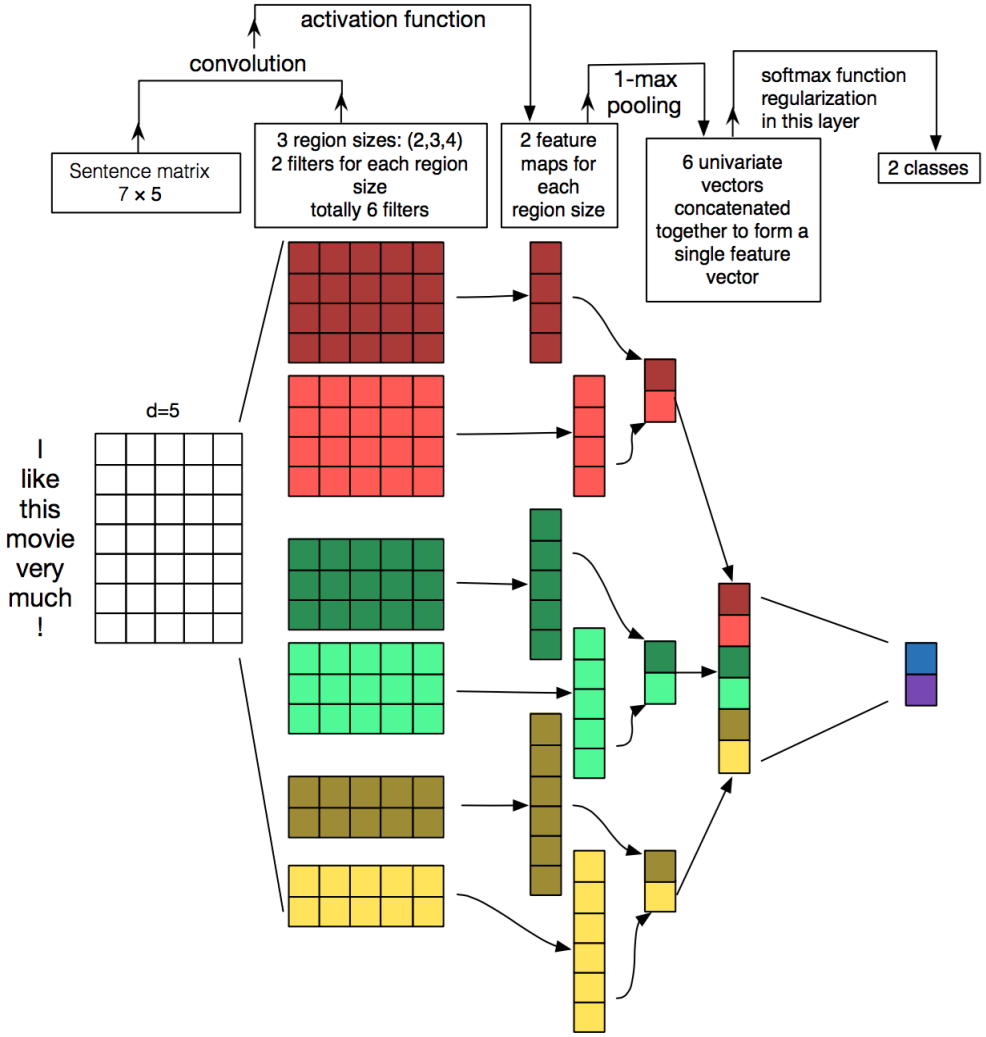
커널의 크기가 커질수록, 참고하는 n-gram의 n이 커진다.

Max-pooling

- 추출한 벡터 중에서 가장 중요한 특성만을 뽑아내기 위해서 수행한다. (벡터 -> 스칼라값)
- 커널의 크기에 따라서 합성곱 결과로 얻은 벡터의 길이는 다를 수 있지만 이와 상관없이 수행할 수 있는 연산이다.



1D CNN Overview



권장 PJT

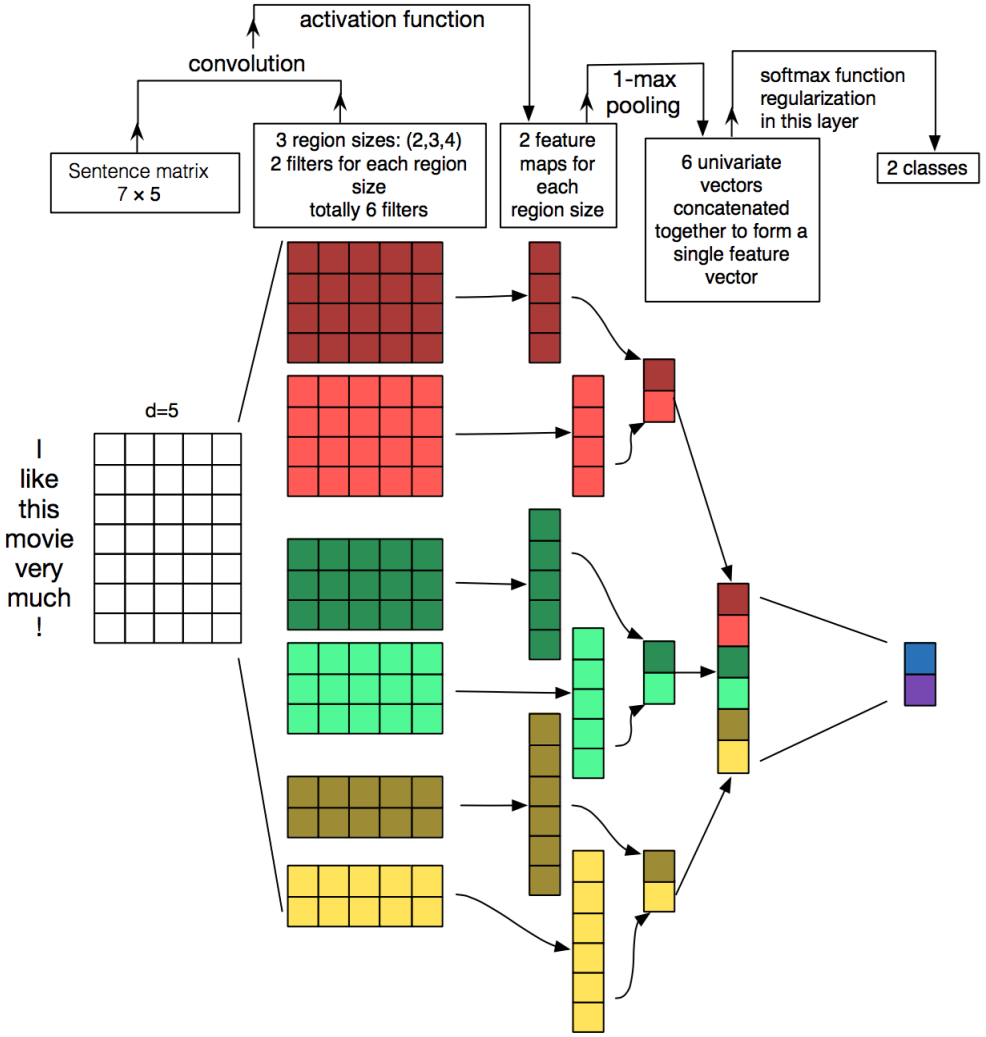
- 양방향 LSTM를 이용한 스팀 리뷰 감성 분석(한국어) : https://wikidocs.net/94748
- 1D CNN을 이용한 IMDB 리뷰 분류(영어) : https://wikidocs.net/80783

점검 Quiz - RNN

- 아래 전체 모델의 학습가능한 파라미터 수와 모델을 직접 작성해보세요.
- Sequential 모델로 작성 후 model.summary()를 하셔서 정답을 확인해보세요.

1. **Embedding을 사용하며, 단어 집합(Vocabulary)의 크기가 5,000이고 임베딩 벡터의 차원은 100입니다.**
2. **은닉층에서는 Simple RNN을 사용하며, 은닉 상태의 크기는 128입니다.**
3. **훈련에 사용하는 모든 샘플의 길이는 30으로 가정합니다.**
4. **이진 분류를 수행하는 모델로, 출력층의 뉴런은 1개로 시그모이드 함수를 사용합니다.**
5. **은닉층은 1개입니다.**

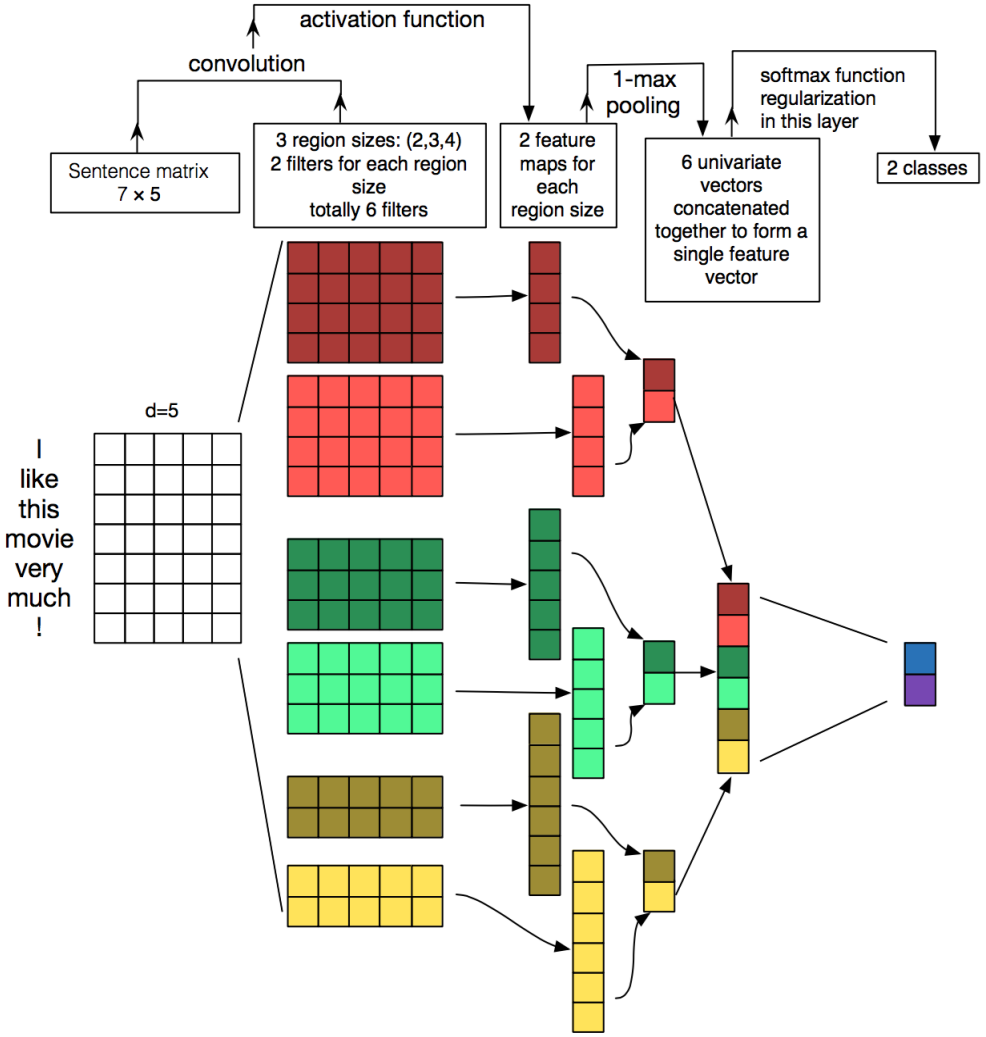
점검 Quiz - CNN



다음은 수많은 데이터 중 문장 1개에 대해서 이진 분류를 수행하고 있는 모델이다.
그림을 보고 해당 모델의 파라미터를 추측한다면 총 몇 개일까?

Embedding layer의 파라미터 : ?
Conv1D의 파라미터 : ?
Dense의 파라미터 : ?

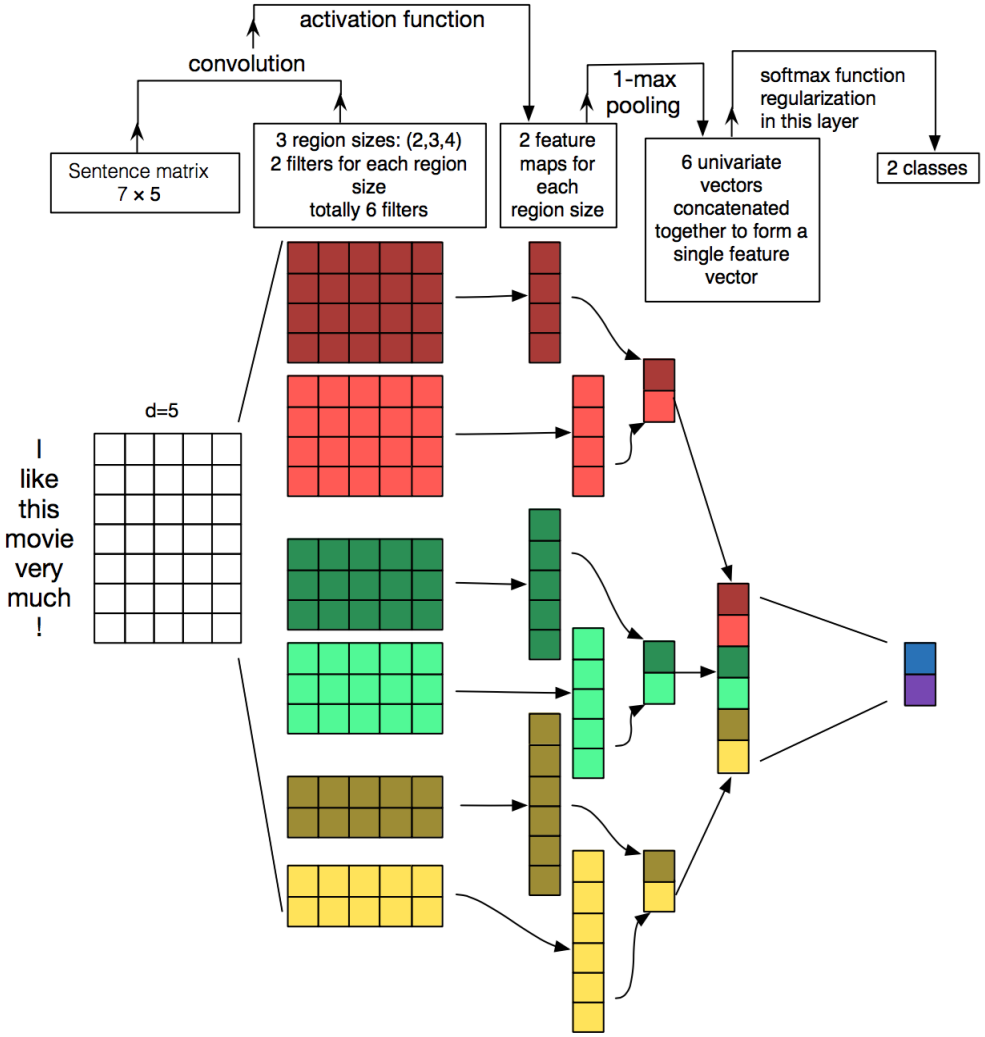
점검 Quiz - CNN



다음은 수많은 데이터 중 문장 1개에 대해서 이진 분류를 수행하고 있는 모델이다.
그림을 보고 해당 모델의 파라미터를 추측한다면 총 몇 개일까?

Embedding layer의 파라미터 : 알 수 없음
Conv1D의 파라미터 : 96개
Dense의 파라미터 : 14개

점검 Quiz - CNN



코드로 작성해보세요.

- 참고 링크 : https://wikidocs.net/85337

ELMo

엘모(ELMo) – Embeddings from Language Model

사전 훈련된 단어 표현(Pre-trained Word Representations)

- **텍스트 분류 등 다양한 NLU 모델의 핵심 컴포넌트이다.**

높은 퀄리티의 표현은 이상적으로 다의어를 모델링 할 수 있어야 한다.

엘모(ELMo) – Embeddings from Language Model

Example

GloVe mostly learns *sport*-related context

	Source	Nearest Neighbors
GloVe	play	playing, game, games, played, players, plays, player, Play, football, multiplayer
biLM	Chico Ruiz made a spectacular play on Alusik 's grounder {...}	Kieffer , the only junior in the group , was commended for his ability to hit in the clutch , as well as his all-round excellent play .
	Olivia De Havilland signed to do a Broadway play for Garson {...}	{...} they were actors who had been handed fat roles in a successful play , and had talent enough to fill the roles competently , with nice understatement .

Table 4: Nearest neighbors to "play" using GloVe and the context embeddings from a biLM.

ELMo can distinguish the word sense based on the context

엘모(ELMo) – Embeddings from Language Model

Example

	Source	Nearest Neighbors
GloVe	play	playing, game, games, played, players, plays, player, Play, football, multiplayer
biLM	Chico Ruiz made a spectacular play on Alusik 's grounder {...}	Kieffer , the only junior in the group , was commended for his ability to hit in the clutch , as well as his all-round excellent play .
	Olivia De Havilland signed to do a Broadway play for Garson {...}	{...} they were actors who had been handed fat roles in a successful play , and had talent enough to fill the roles competently , with nice understatement .

Table 4: Nearest neighbors to "play" using GloVe and the context embeddings from a biLM.

GloVe mostly learns *sport*-related context

ELMo can distinguish the word sense based on the context

엘모(ELMo) – Embeddings from Language Model

ELMo

- ELMo에서 사용하는 벡터들은 기본적으로 방대한 데이터를 훈련시킨 Language Model로 훈련시킨 bidirectional LSTM에서 가지고 온다.
- 각 토큰 벡터는 전체 입력 문장으로부터 결정된다.

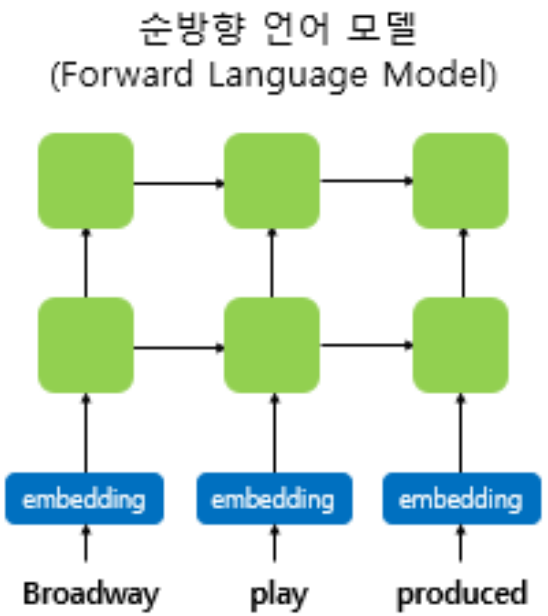
특징

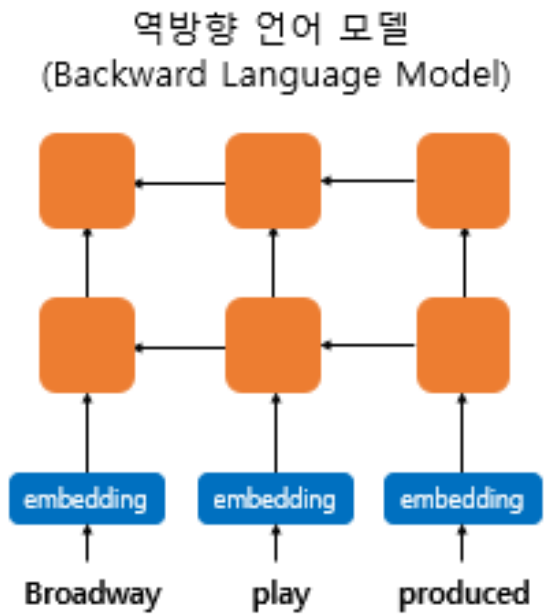
- ELMo에서 사용되는 biLM은 기본적으로 은닉층이 2개 이상인 깊은 신경망을 가정한다.
- ELMo representation은 기본적으로 biLM의 모든 내부 층들로부터 결정된다.
- 특정 토큰 위의 모든 내부 층들의 벡터가 선형 결합(linear combination)을 통해 표현을 얻는다.
- 위 방법은 Top layer의 LSTM의 표현만을 사용한 것보다 높은 성능을 보여준다.
- 각 층은 가지고 있는 의미가 다른데, 낮은 층에서는 구문 분석(Syntax analysis)에 유리한 정보를 갖고 있으며, 윗 층에서는 NLU에 가까운 정보를 가지고 있다. 이를 가중치로 조정한다.

엘모(ELMo) – Embeddings from Language Model

- 엘모는 직역하면 '언어 모델로부터 얻는 임베딩'
- 엘모는 BiLM(양방향 언어 모델)을 사용한다.
- 엘모는 순방향과 역방향으로 RNN 언어 모델을 따로 학습시킨다.

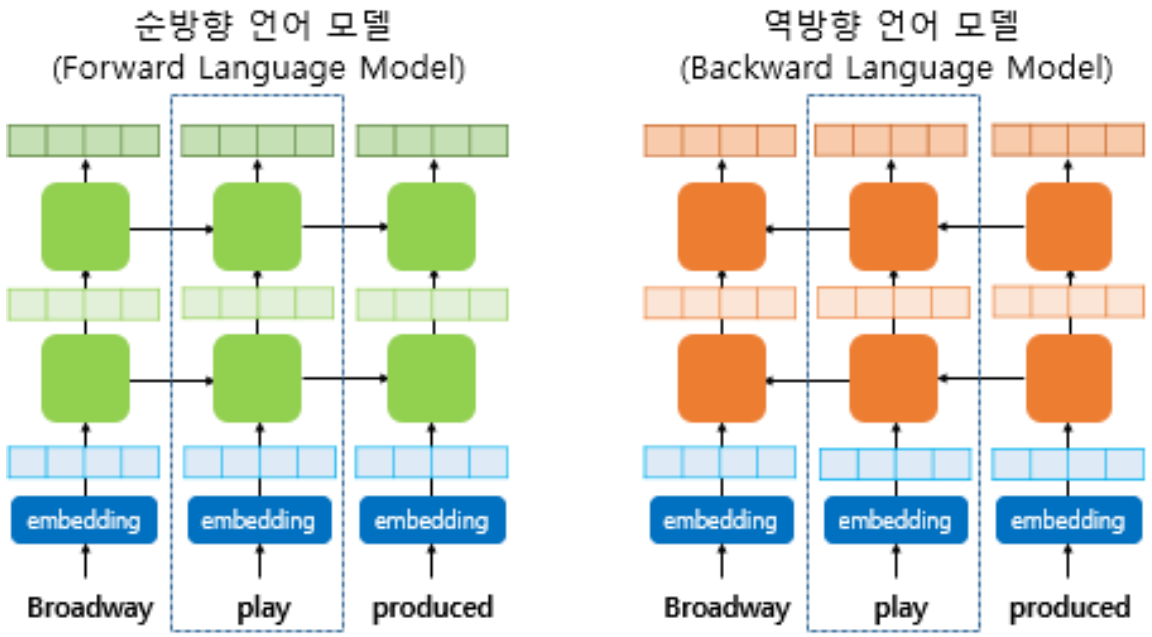
이 케릭터 이름이 엘모.





엘모(ELMo) – Embeddings from Language Model

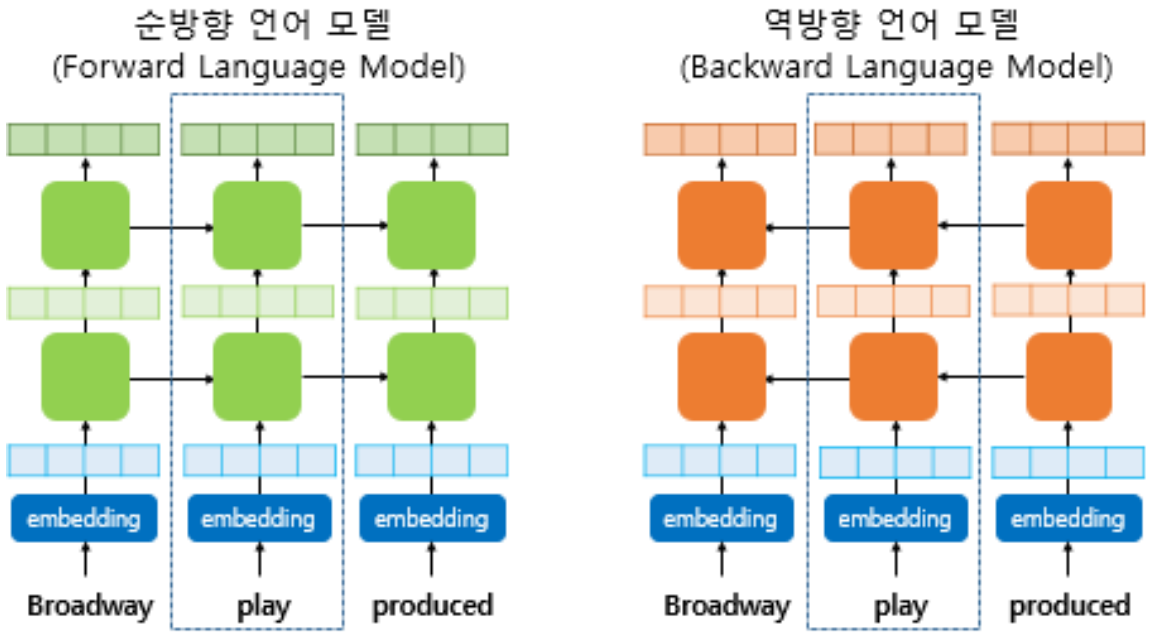
- 엘모는 BiLM(양방향 언어 모델)을 사용한다.
- 엘모는 순방향과 역방향으로 언어 모델을 따로 학습시킨다. 이것이 사전 학습(pre-training) 단계이다.
- 예를 들어 단어 'play'에 대한 임베딩을 얻는다고 가정해보자.



ELMo는 좌측 점선의 사각형 내부의
각 층의 결과값을 재료로 사용합니다.

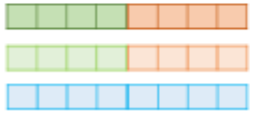
엘모(ELMo) – Embeddings from Language Model

- 예를 들어 단어 'play'에 대한 임베딩을 얻는다고 가정해보자.

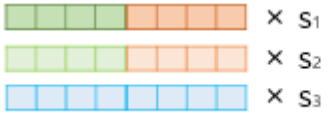


ELMo는 좌측 점선의 사각형 내부의
각 층의 결과값을 재료로 사용합니다.

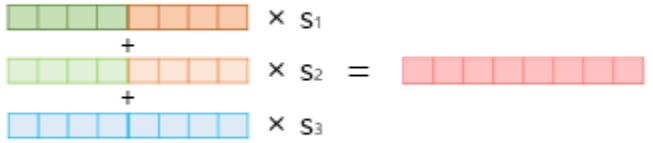
1) 각 층의 출력값을 연결(concatenate)한다.



2) 각 층의 출력값 별로 가중치를 준다.



3) 각 층의 출력값을 모두 더한다.

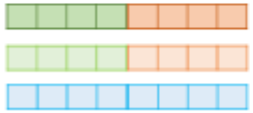


2)번과 3)번의 단계를 요약하여 가중합(Weighted Sum)을 한다고 할 수 있습니다.

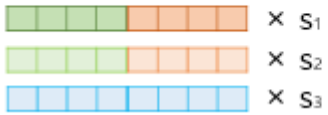
4) 벡터의 크기를 결정하는 스칼라 매개변수를 곱한다.

엘모(ELMo) – Embeddings from Language Model

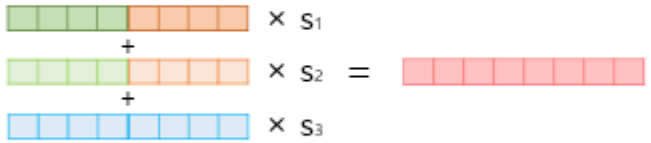
1) 각 층의 출력값을 연결(concatenate)한다.



2) 각 층의 출력값 별로 가중치를 준다.

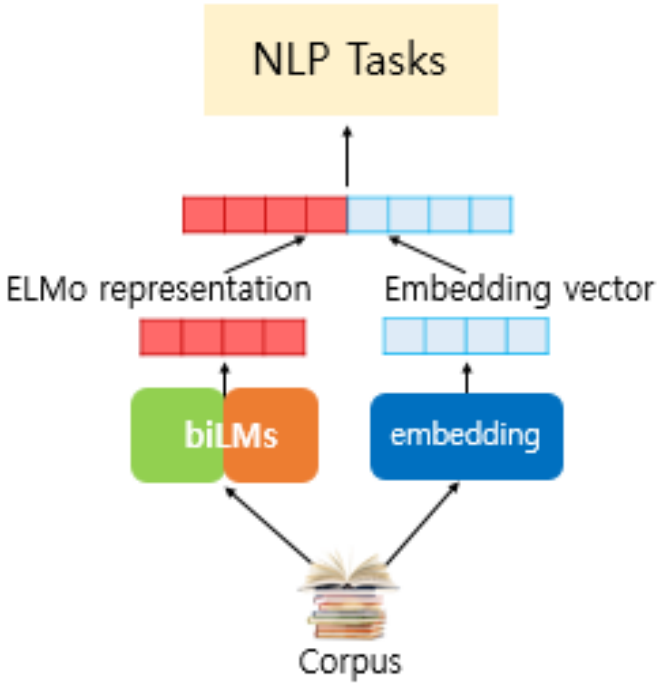


3) 각 층의 출력값을 모두 더한다.



2)번과 3)번의 단계를 요약하여 가중합(Weighted Sum)을 한다고 할 수 있습니다.

4) 벡터의 크기를 결정하는 스칼라 매개변수를 곱한다.



엘모(ELMo) – Embeddings from Language Model

- ELMo for downstream task

ELMo represents a word t_k as a linear combination of corresponding hidden layers (inc. its embedding)

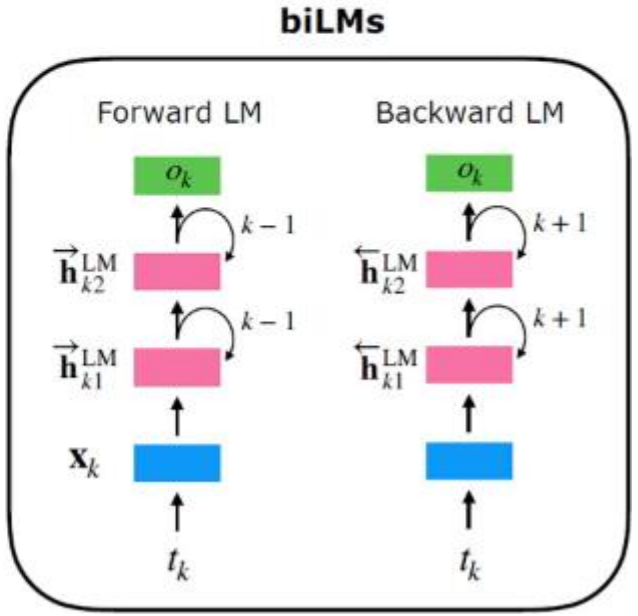
ELMo is a task specific representation. A down-stream task learns weighting parameters

$$\mathbf{ELMo}_k^{\text{task}} = \gamma^{\text{task}} \times \sum \left\{ s_2^{\text{task}} \times \mathbf{h}_{k2}^{\text{LM}},\ s_1^{\text{task}} \times \mathbf{h}_{k1}^{\text{LM}},\ s_0^{\text{task}} \times \mathbf{h}_{k0}^{\text{LM}} \right.$$

$$\left. ([\mathbf{x}_k ; \mathbf{x}_k]),\ [\overrightarrow{\mathbf{h}}_{kj}^{\text{LM}} ; \overleftarrow{\mathbf{h}}_{kj}^{\text{LM}}] \right\}$$

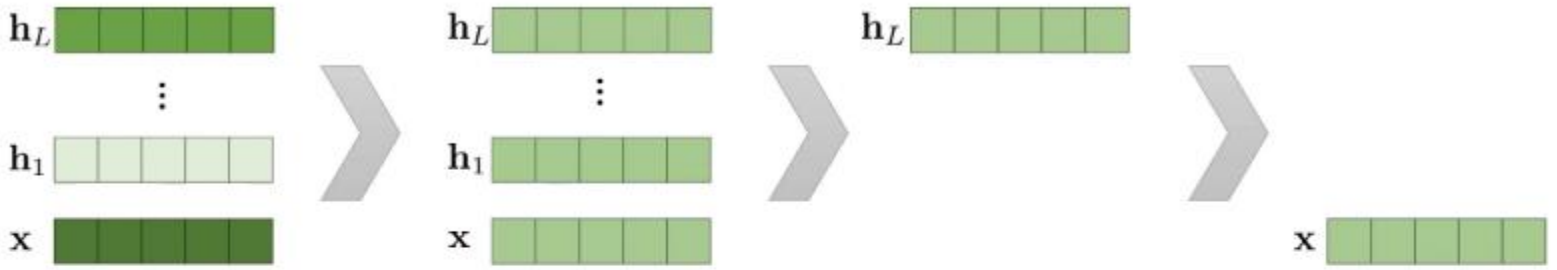
Concatenate hidden layers

Unlike usual word embeddings, ELMo is assigned to every *token* instead of a *type*

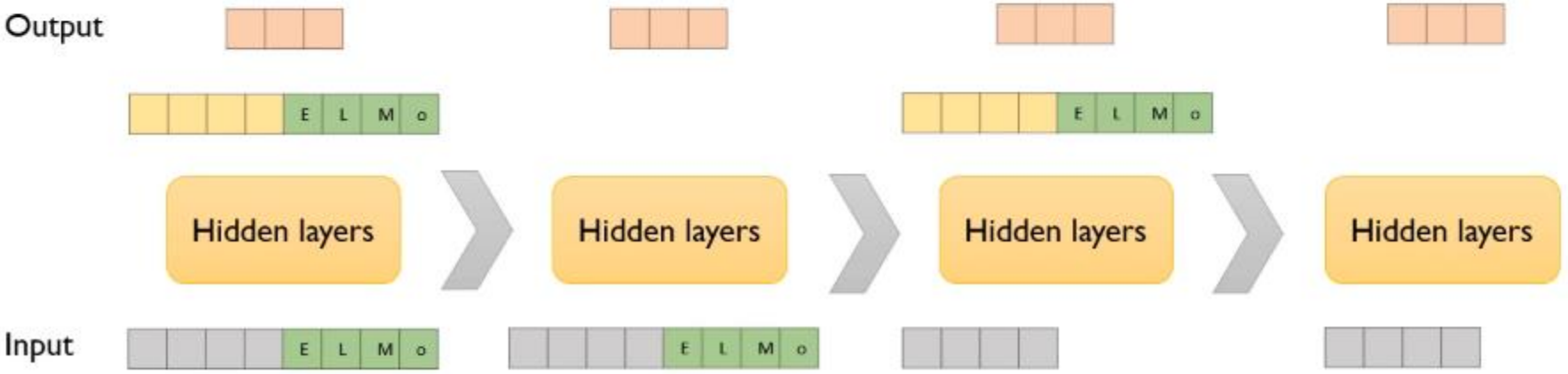


엘모(ELMo) – Embeddings from Language Model

- Analysis: Alternate layer weighting scheme



- Analysis: Where to include ELMo?



Character-level CNN for Text Classification

Character-level CNN for Text Classification

- Zhang et al., 2015
- 단어 레벨이 아닌 문자 레벨(Charater-level)의 접근.
- 총 70개의 Character를 사용. vocab_size = 70
- 26 알파벳, 10개의 숫자, 33개의 특수 문자

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

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.

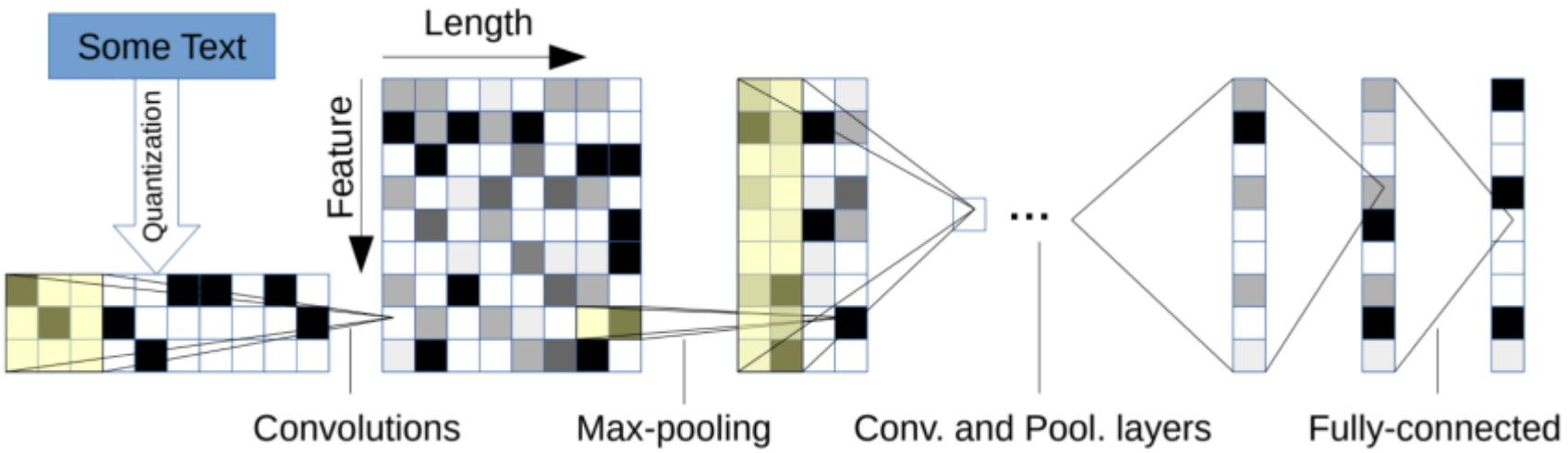
There are also related works that use character-level features for language processing. These include using character-level n-grams with linear classifiers [15], and incorporating character-level features to ConvNets [28] [29]. In particular, these ConvNet approaches use words as a basis, in which character-level 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

Character-level CNN for Text Classification

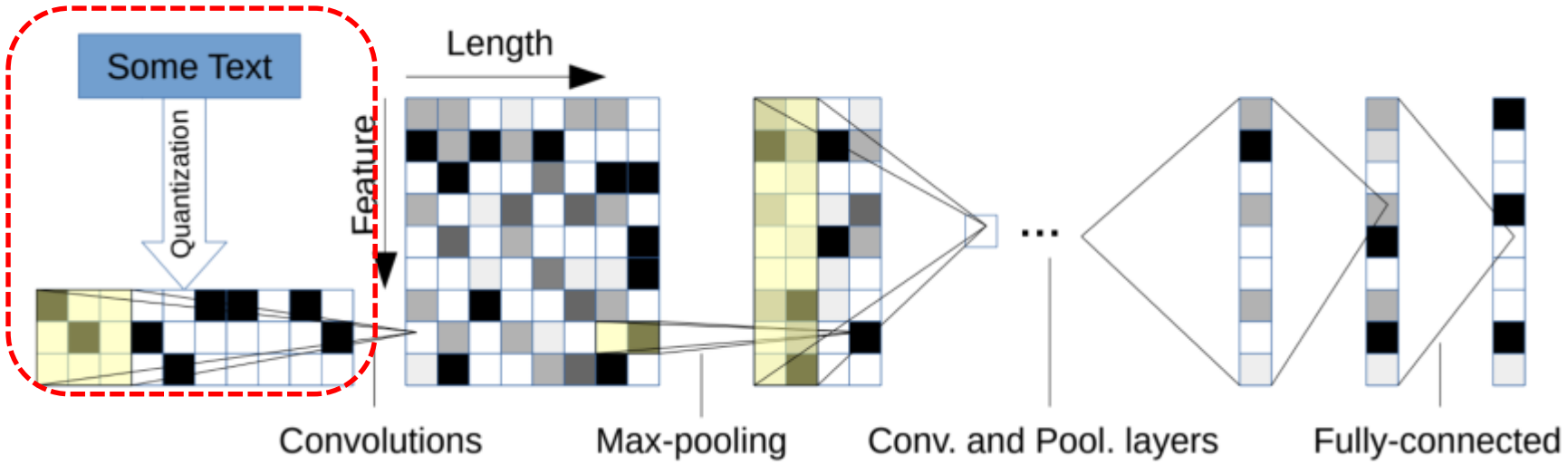
- Zhang et al., 2015
- 단어 레벨이 아닌 문자 레벨(Charater-level)의 접근.
- 총 70개의 Character를 사용
- 26 알파벳, 10개의 숫자, 33개의 특수 문자

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



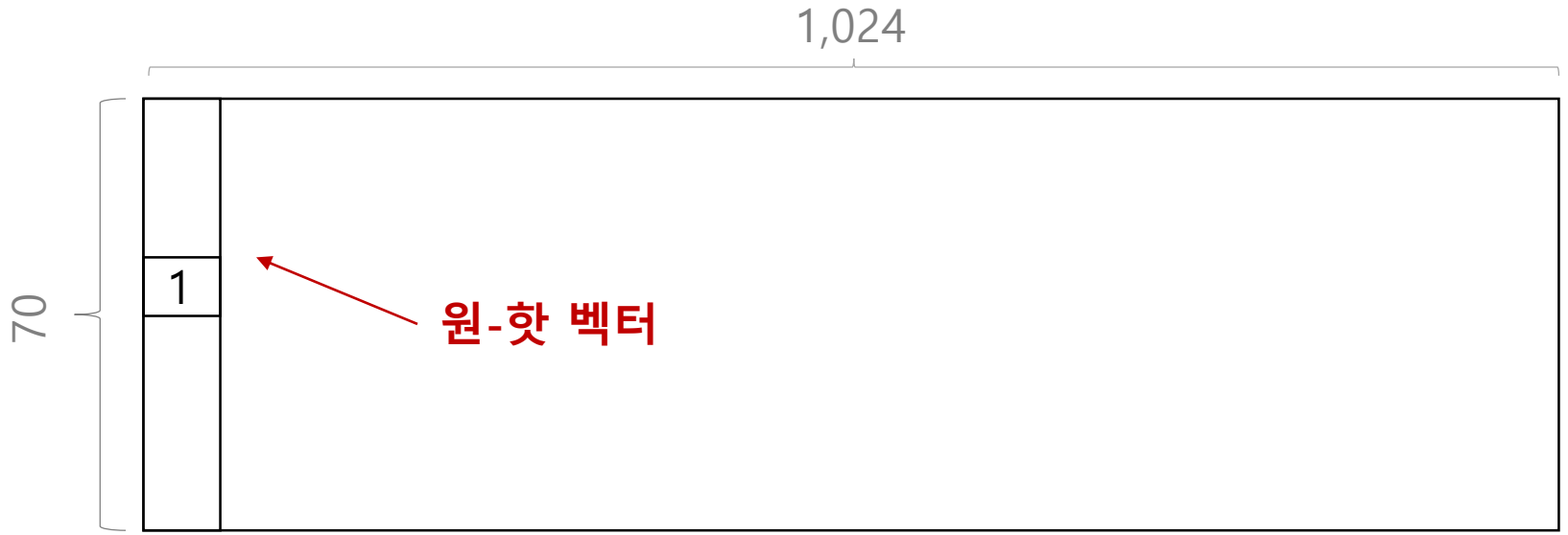
Character-level CNN for Text Classification

- 입력 행렬은 70 x 1,024의 크기를 가지는 행렬.
- 한 문장 또는 한 문서는 최대 1,024의 Character를 가질 수 있다고 가정.
- 각 열은 원-핫 벡터.
- Each column in an one-hot vector for the corresponding character (not distributed representation)



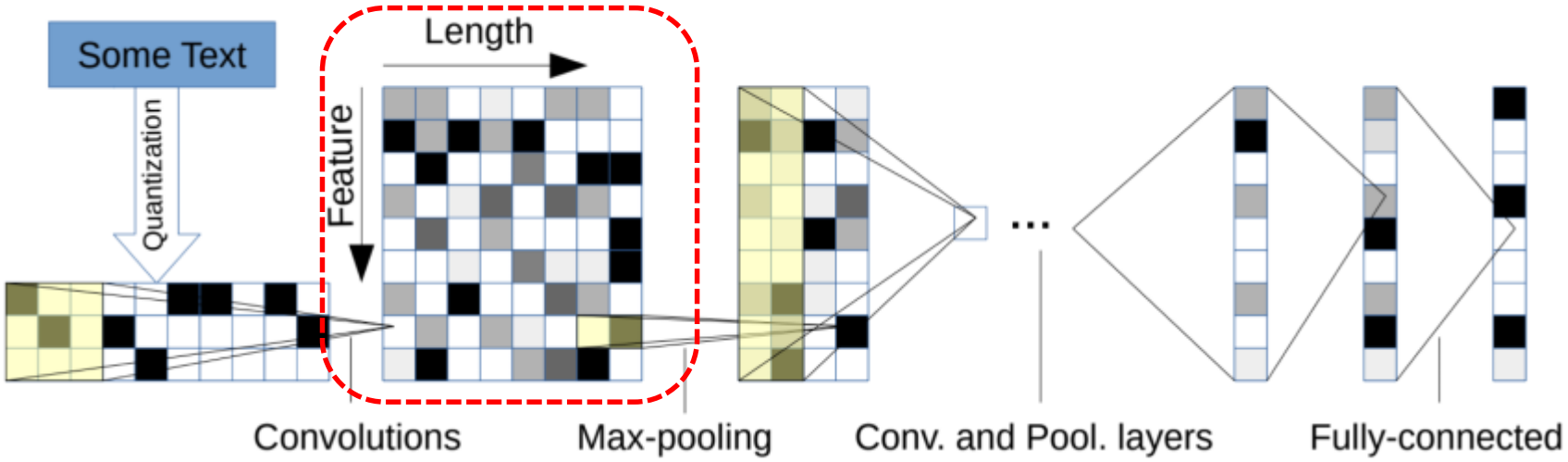
Character-level CNN for Text Classification

- 입력 행렬은 70 x 1,024의 크기를 가지는 행렬.
- 한 문장 또는 한 문서는 최대 1,024의 Character를 가질 수 있다고 가정.
- 각 열은 원-핫 벡터.
- Each column in an one-hot vector for the corresponding character (not distributed representation)



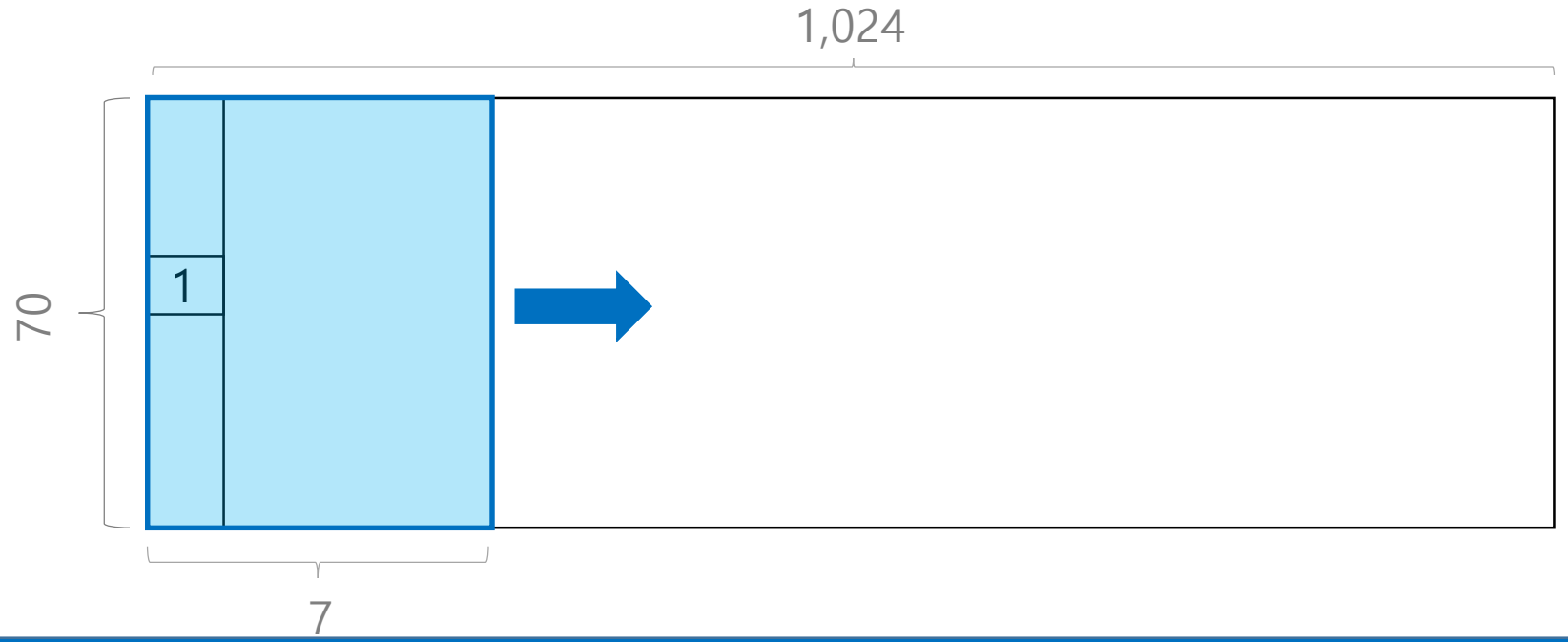
Character-level CNN for Text Classification

- Convolution 연산은 이번에도 1D로 이루어진다.
- 논문에서는 커널의 크기를 7로 두 번, 3으로 네 번 수행한다.
- 커널 크기가 7일 때를 가정한다면?



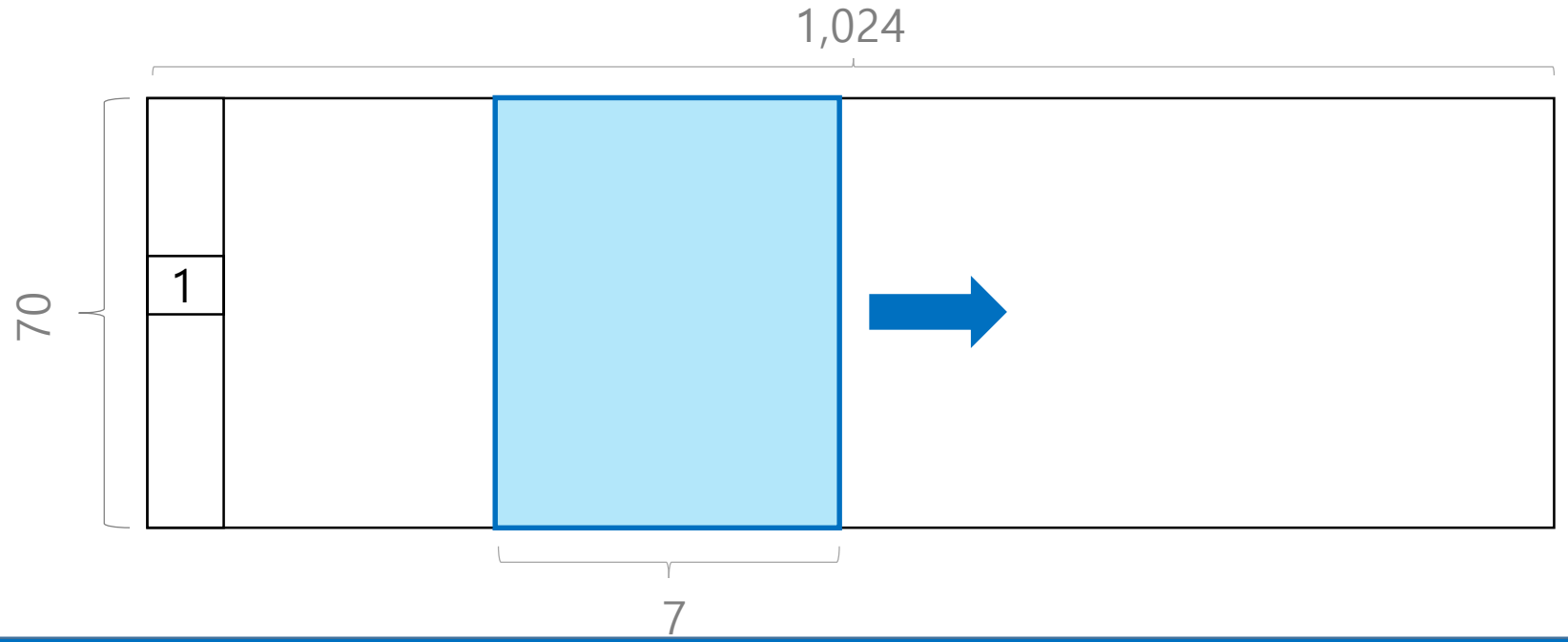
Character-level CNN for Text Classification

- **Convolution 연산은 이번에도 1D로 이루어진다.**
- **논문에서는 커널의 크기를 7로 두 번, 3으로 네 번 수행한다.**
- **커널 크기가 7일 때를 가정한다면?**



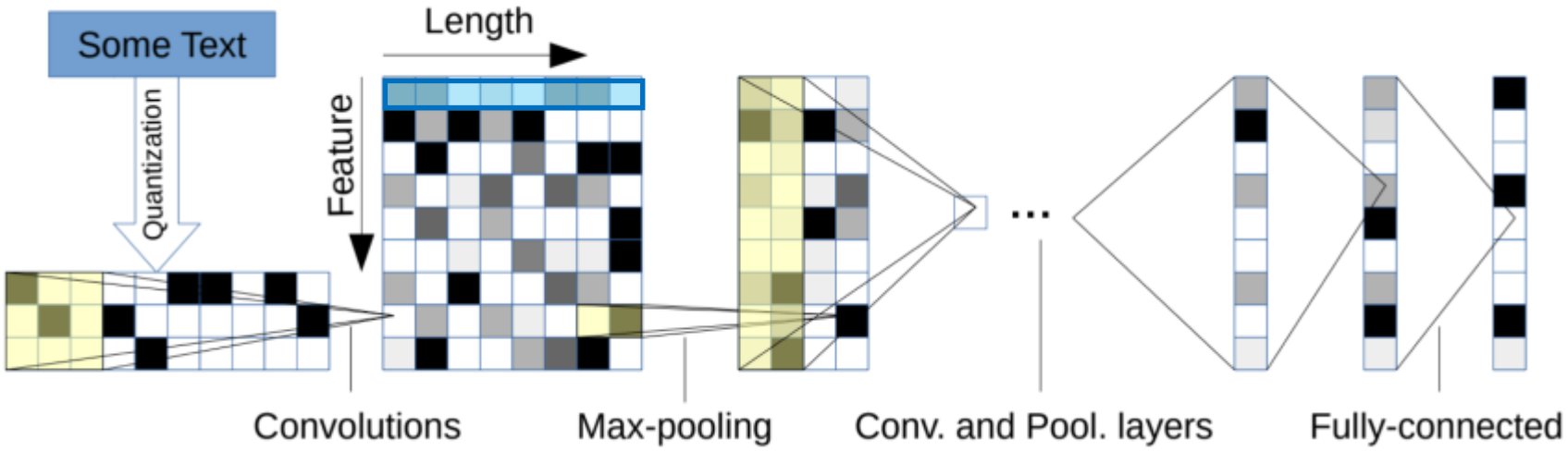
Character-level CNN for Text Classification

- Convolution 연산은 이번에도 1D로 이루어진다.
- 논문에서는 커널의 크기를 7로 두 번, 3으로 네 번 수행한다.
- 커널 크기가 7일 때를 가정한다면?



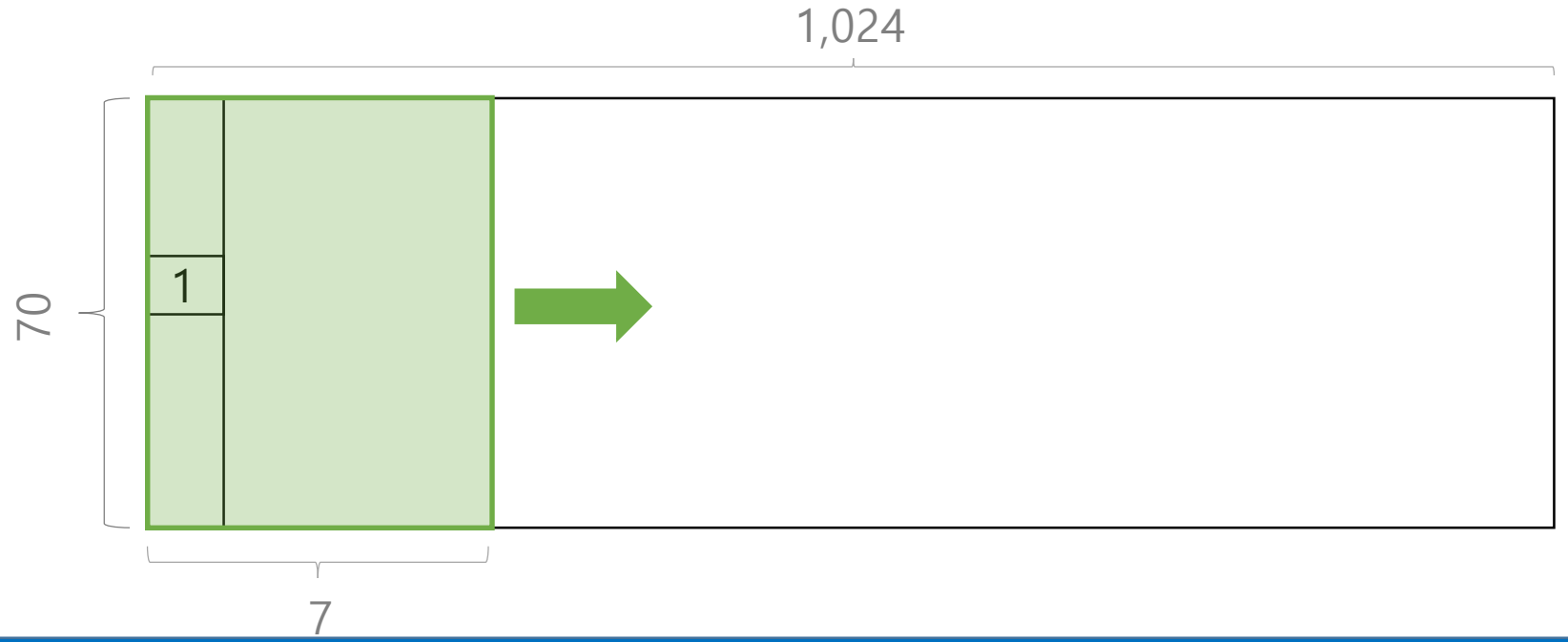
Character-level CNN for Text Classification

- Convolution 연산은 이번에도 1D로 이루어진다.
- 논문에서는 커널의 크기를 7로 두 번, 3으로 네 번 수행한다.
- 커널 크기가 7일 때를 가정한다면?



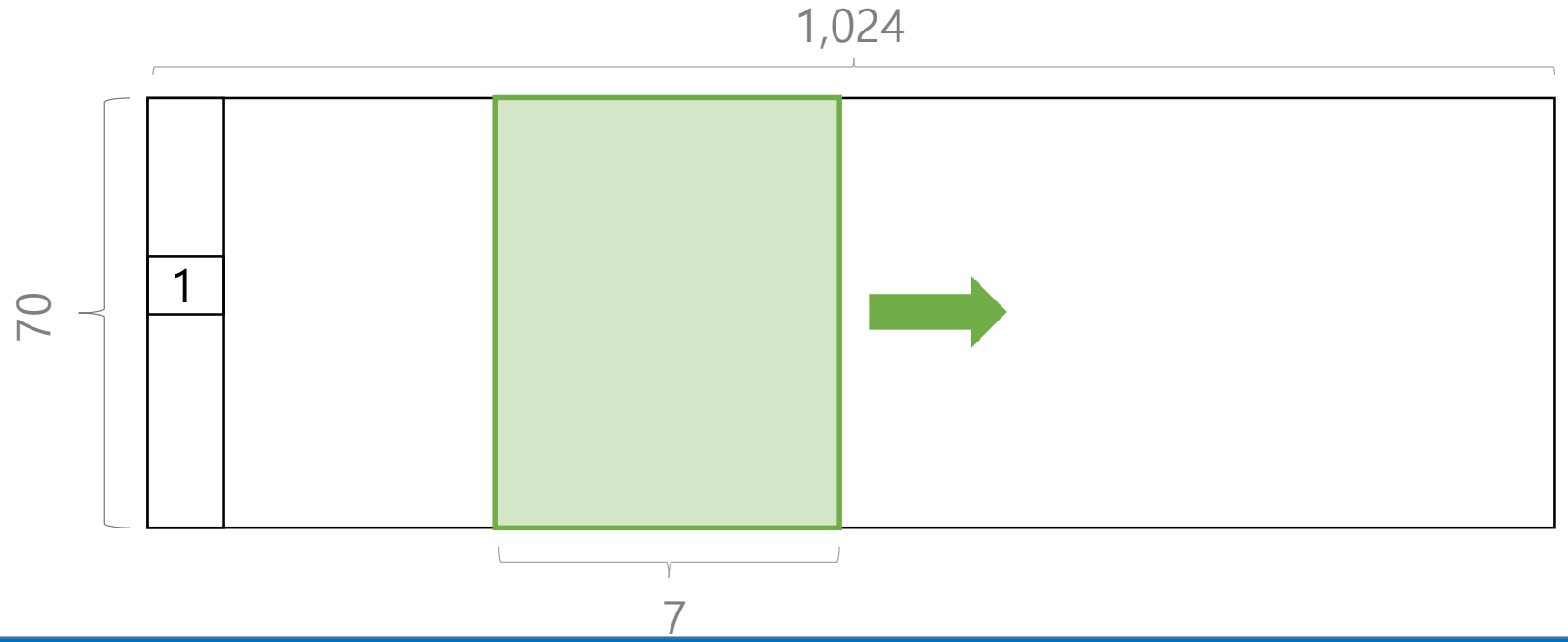
Character-level CNN for Text Classification

- Convolution 연산은 이번에도 1D로 이루어진다.
- 논문에서는 커널의 크기를 7로 두 번, 3으로 네 번 수행한다.
- 커널 크기가 7일 때를 가정한다면?



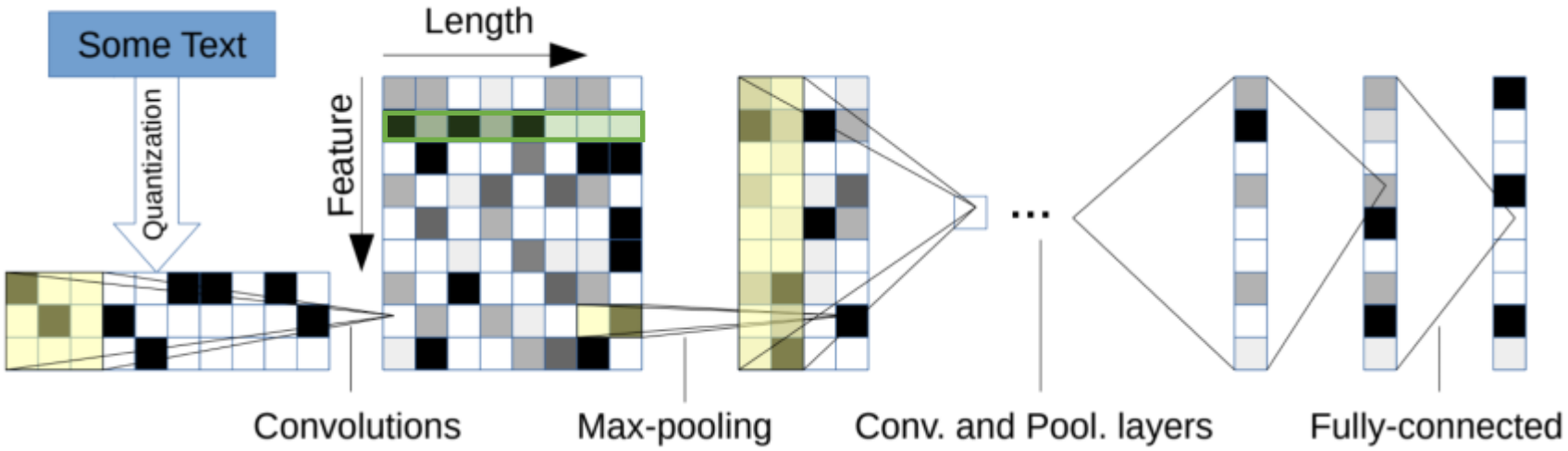
Character-level CNN for Text Classification

- Convolution 연산은 이번에도 1D로 이루어진다.
- 논문에서는 커널의 크기를 7로 두 번, 3으로 네 번 수행한다.
- 커널 크기가 7일 때를 가정한다면?



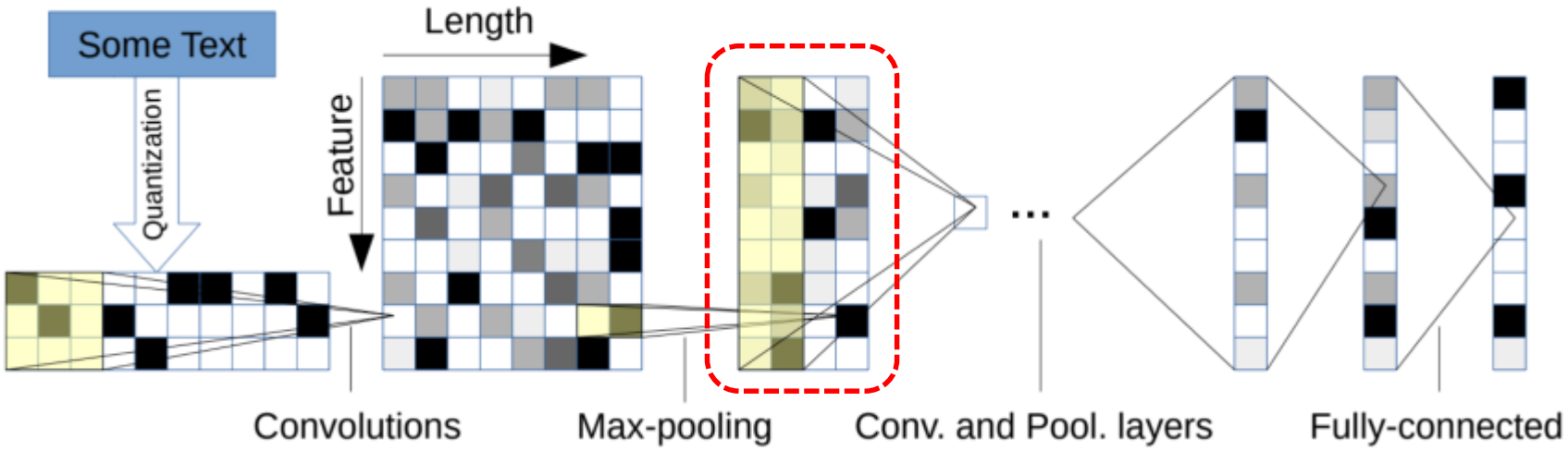
Character-level CNN for Text Classification

- Convolution 연산은 이번에도 1D로 이루어진다.
- 논문에서는 커널의 크기를 7로 두 번, 3으로 네 번 수행한다.
- 커널 크기가 7일 때를 가정한다면?



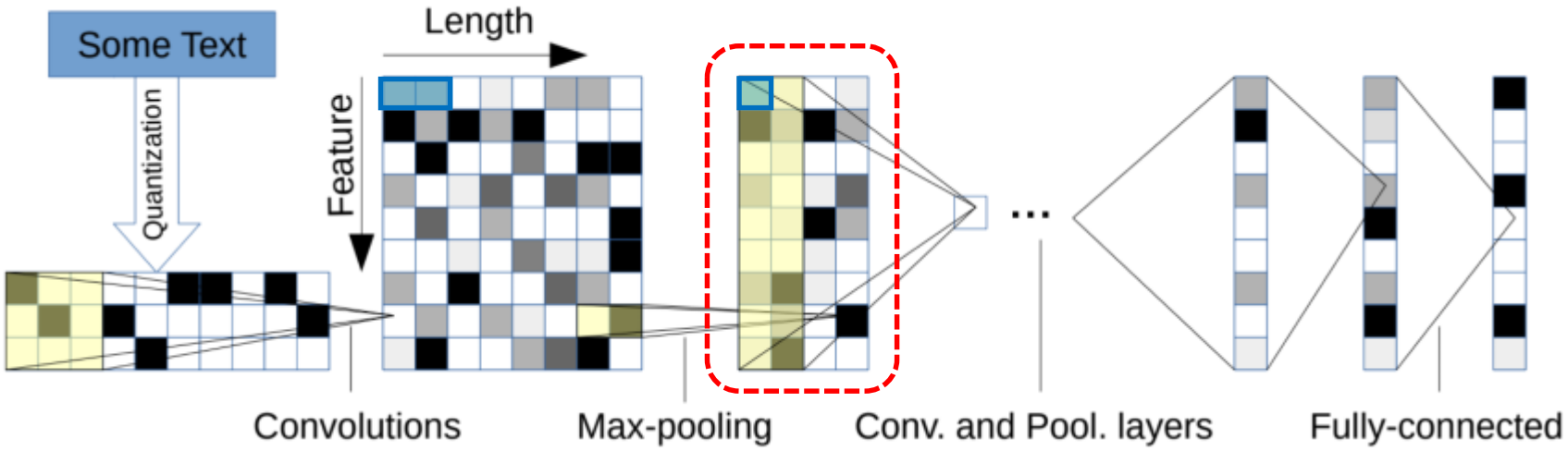
Character-level CNN for Text Classification

- 그림상으로보면 열의 크기가 줄어드는 것처럼 보인다.
- 실제로는 특정 사이즈의 크기 풀링 커널을
  특정 스트라이드를 수행하여 max-pooling을 수행한다.



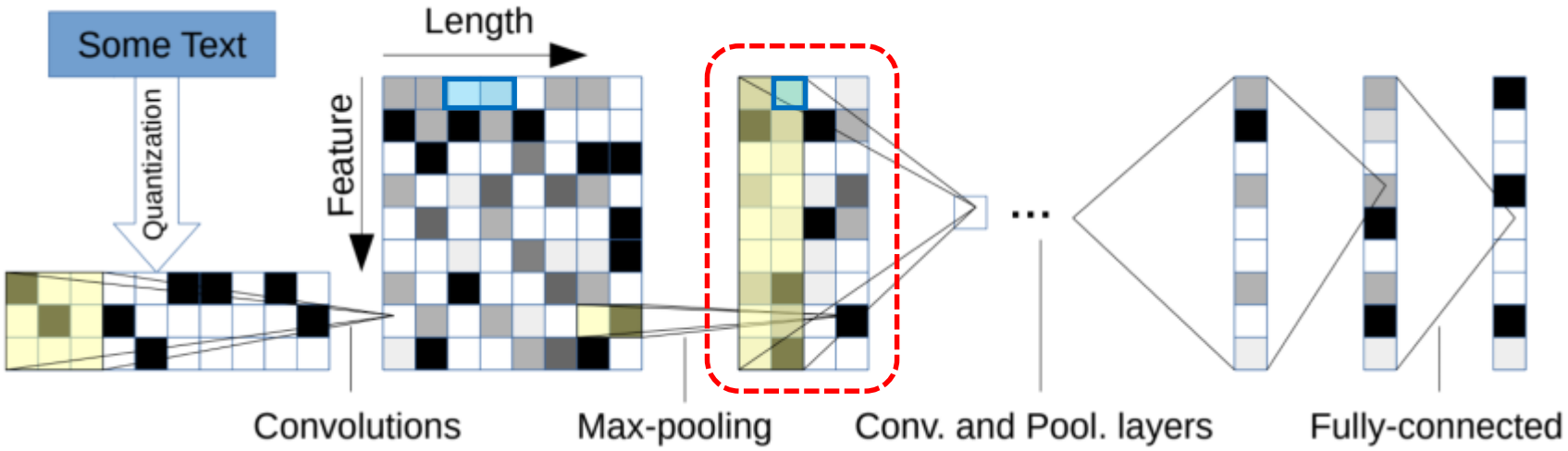
Character-level CNN for Text Classification

- 그림상으로보면 열의 크기가 줄어드는 것처럼 보인다.
- 실제로는 특정 사이즈의 크기 풀링 커널을
  특정 스트라이드를 수행하여 max-pooling을 수행한다.



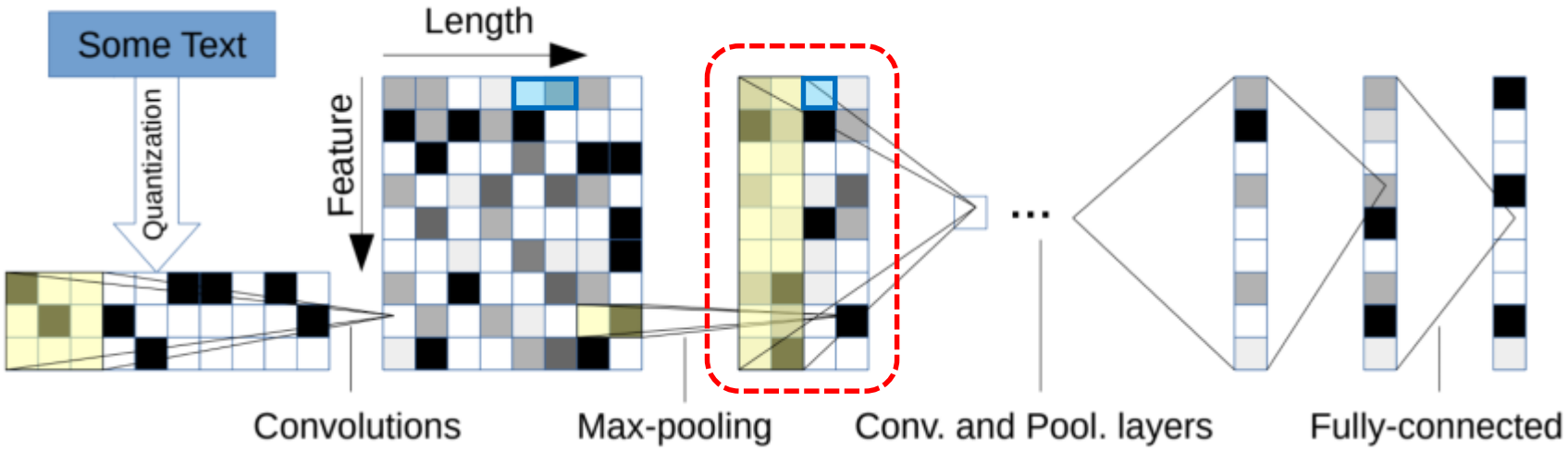
Character-level CNN for Text Classification

- 그림상으로보면 열의 크기가 줄어드는 것처럼 보인다.
- 실제로는 특정 사이즈의 크기 풀링 커널을
  특정 스트라이드를 수행하여 max-pooling을 수행한다.



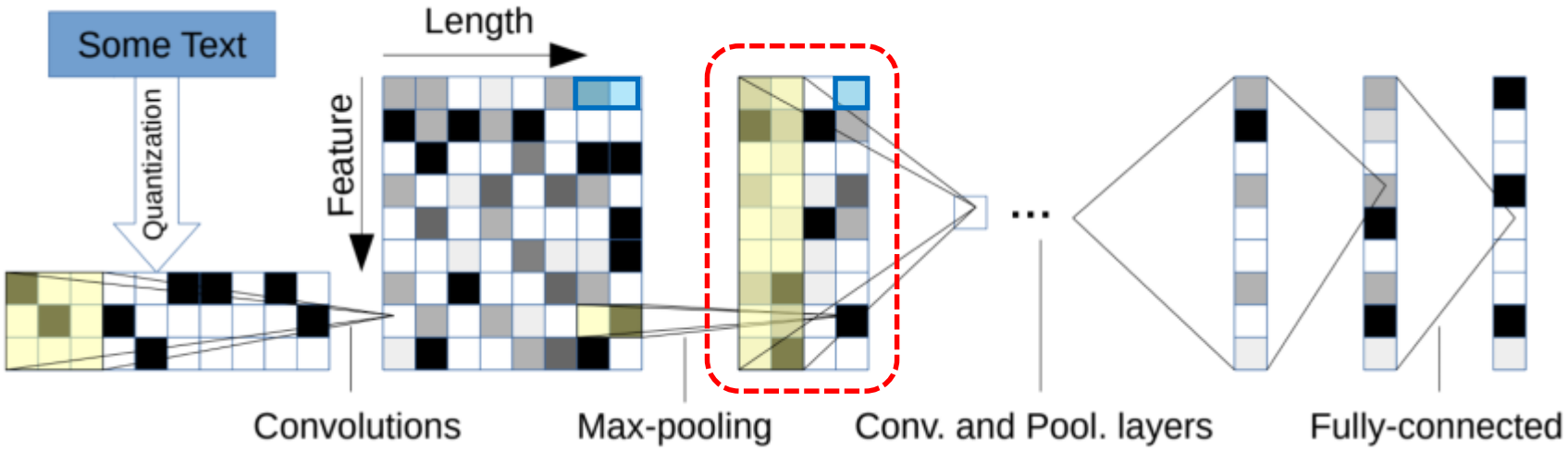
Character-level CNN for Text Classification

- 그림상으로보면 열의 크기가 줄어드는 것처럼 보인다.
- 실제로는 특정 사이즈의 크기 풀링 커널을
  특정 스트라이드를 수행하여 max-pooling을 수행한다.



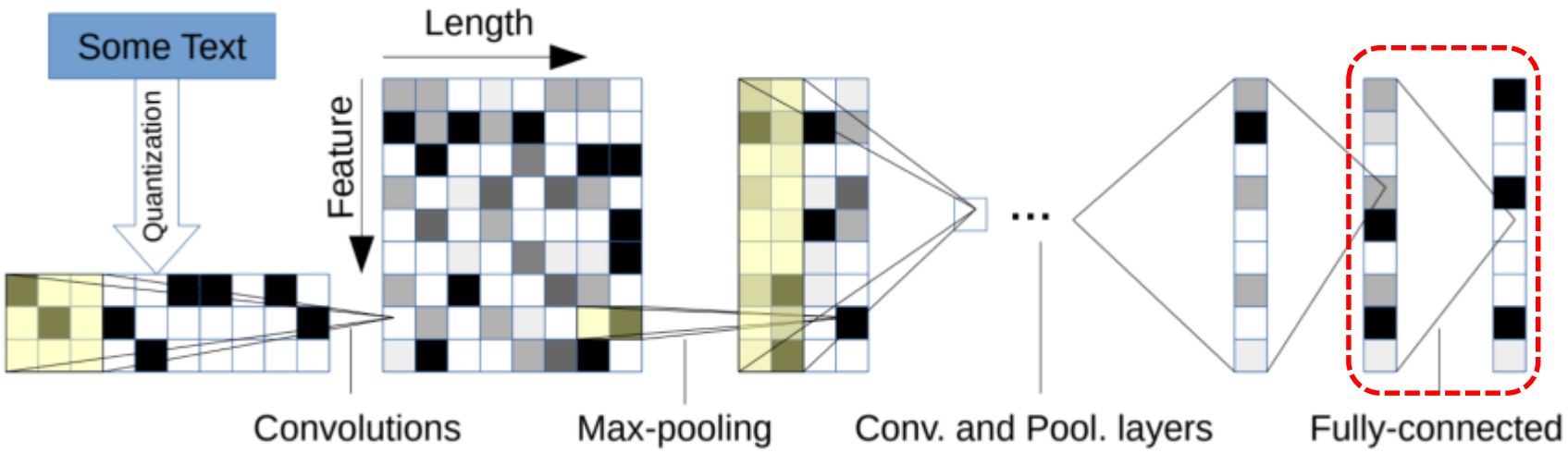
Character-level CNN for Text Classification

- 그림상으로보면 열의 크기가 줄어드는 것처럼 보인다.
- 실제로는 특정 사이즈의 크기 풀링 커널을
  특정 스트라이드를 수행하여 max-pooling을 수행한다.



Character-level CNN for Text Classification

- 출력층의 크기는 Task에 따라서 달라진다.
- Ex) 이진 분류 or 다중 클래스 분류



Character-level CNN for Text Classification

Charactel-level로 하였을 때 얻을 수 있는 이점

- Vocabulary size가 매우 감소. (알파벳, 숫자, 특수문자를 포함해도 100이 넘지 않음.0
- 오탈자, OOV, 신조어, 통신체 등에 강건해진다.