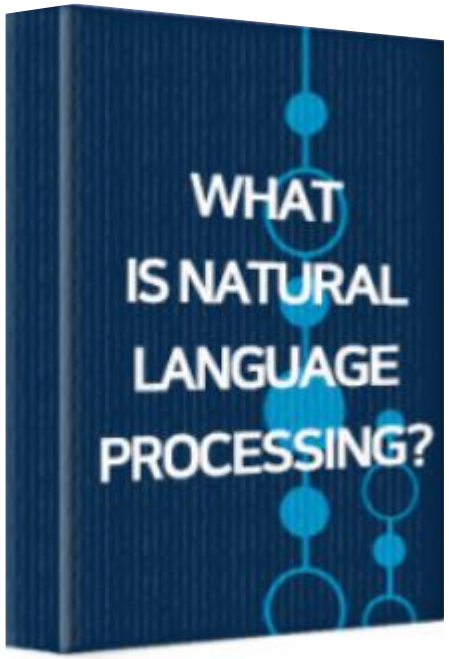
Tensorflow를 활용한 딥러닝 자연어 처리 입문. 5강

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



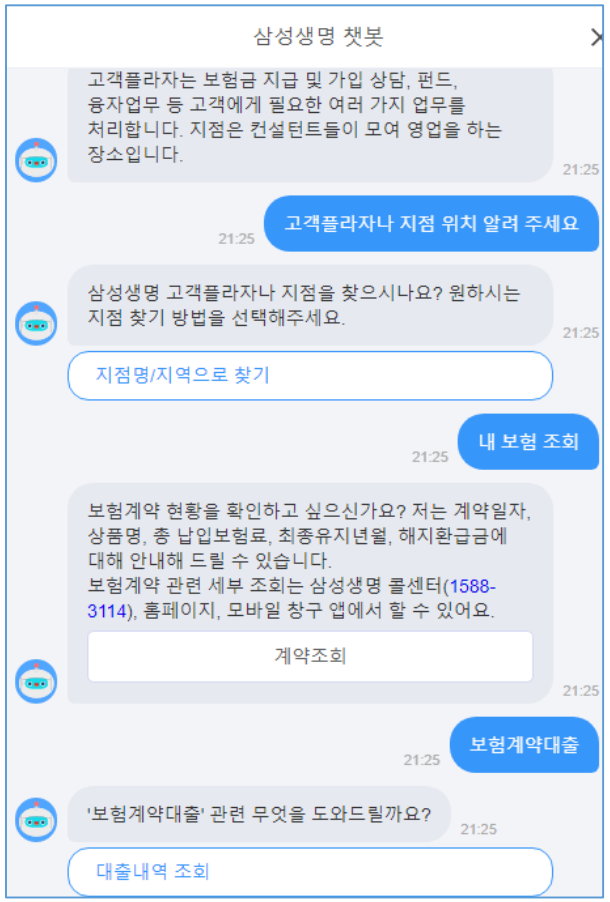
참고 자료 : https://wikidocs.net/book/2155

Chatbot

Chatbot Example

스케터랩의 챗봇 '이루다'

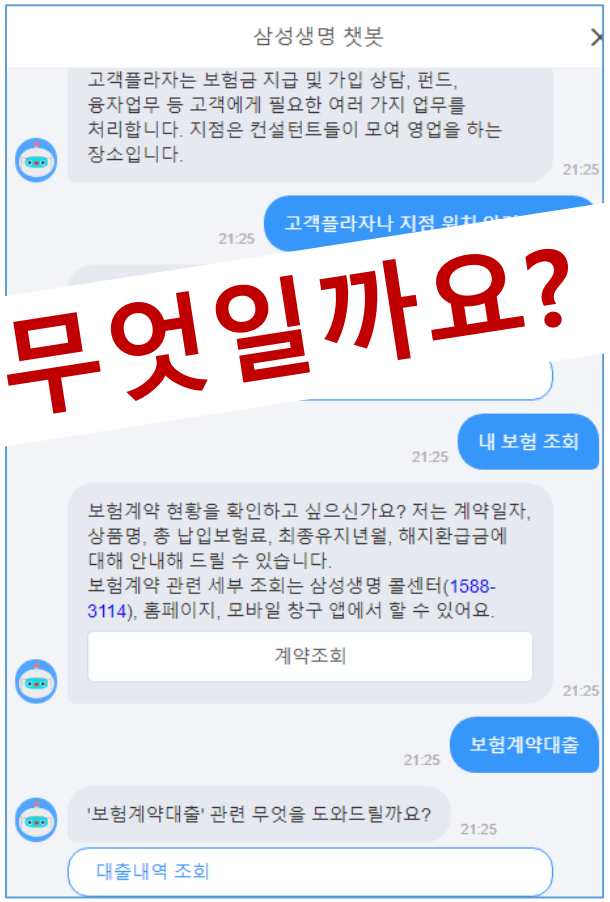
삼성생명 챗봇 '따봇'



Chatbot Example

스케터랩의 챗봇 '이루다'

삼성생명 챗봇 '따봇'

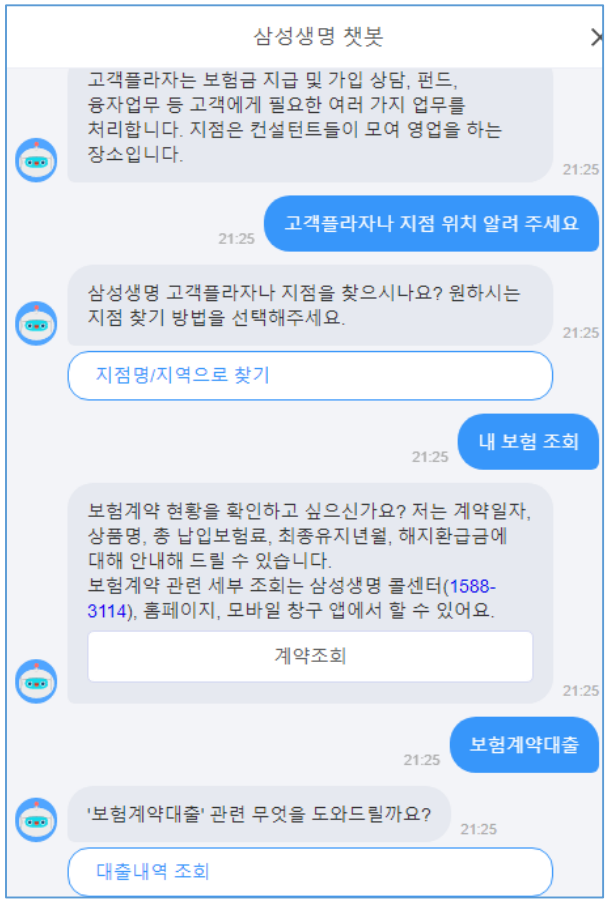


이 두 챗봇의 차이는 무엇일까요?

Chatbot Example

스케터랩의 챗봇 '이루다'

삼성생명 챗봇 '따봇'



Closed Domain Vs. Open-Domain

Closed-Domain Chatbot : 키워드와 인텐트를 기반으로 특정 업무를 수행.

Ex) 삼성생명 챗봇 '따봇', 카카오미니, 연말정산 Q&A봇

Open-Domain Chatbot : 어떤 토픽이든 상관없이 사람과 대화를 나눈다.

Ex) 심심이, 드림이, 이루다

Closed Domain Vs. Open-Domain

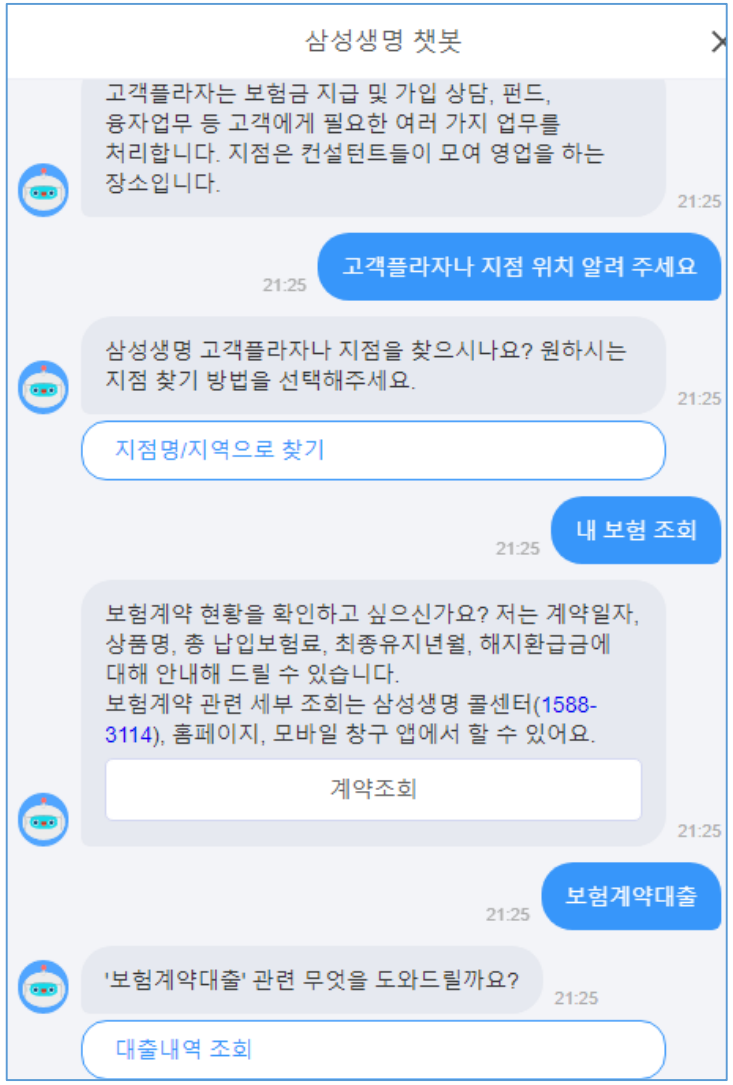
Closed-Domain Chatbot : 키워드와 인텐트를 기반으로 특정 업무를 수행.

Ex) 삼성생명 챗봇 '따봇', 카카오미니, 연말정산 Q&A봇

Open-Domain Chatbot : 어떤 토픽이든 상관없이 사람과 대화를 나눈다.

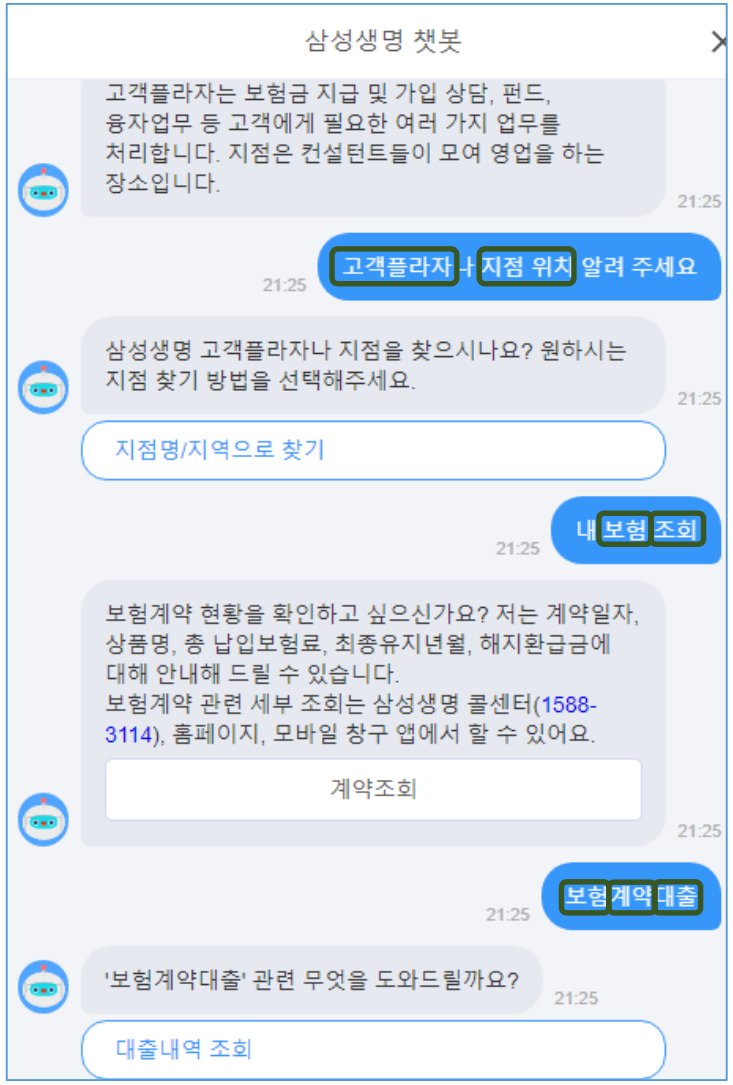
Ex) 심심이, 드림이, 이루다

Closed Domain Chatbot



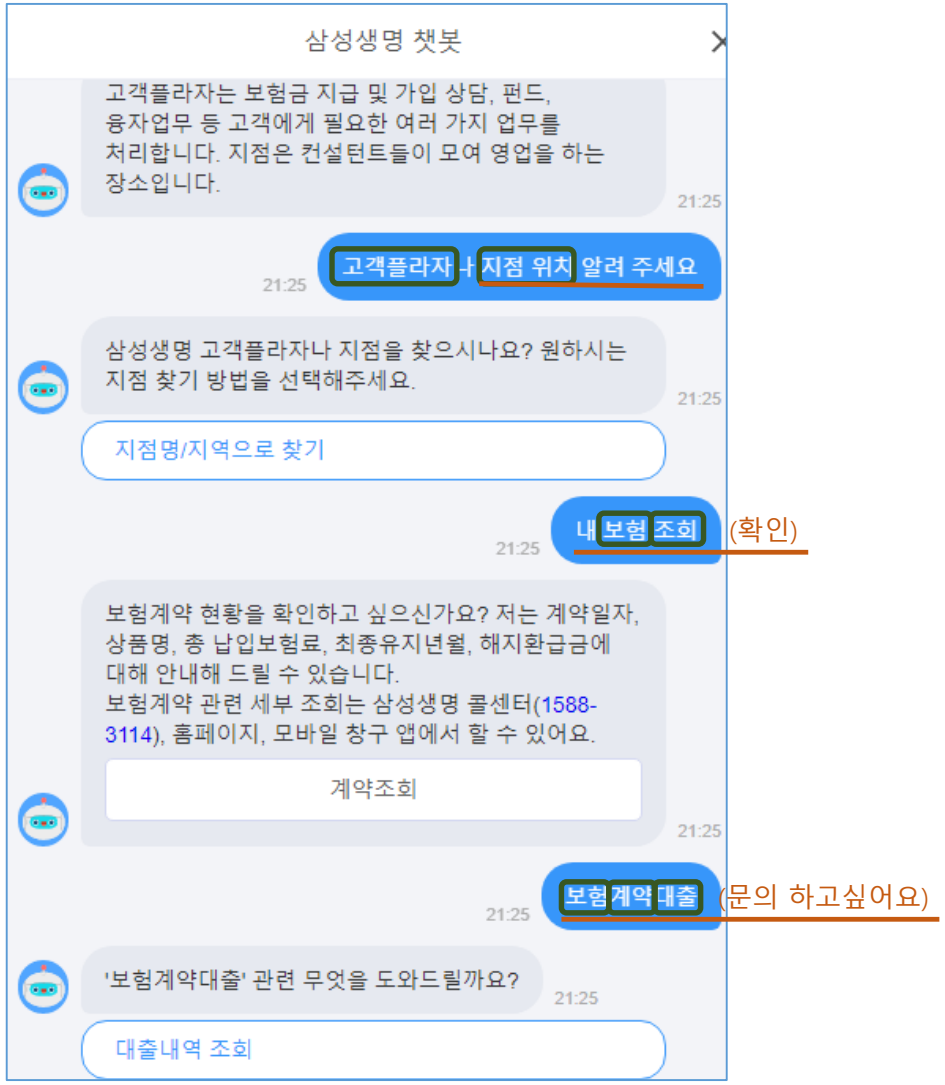
Closed-Domain Chatbot :
키워드와 인텐트를 기반으로
특정 업무를 수행.

Closed Domain Chatbot



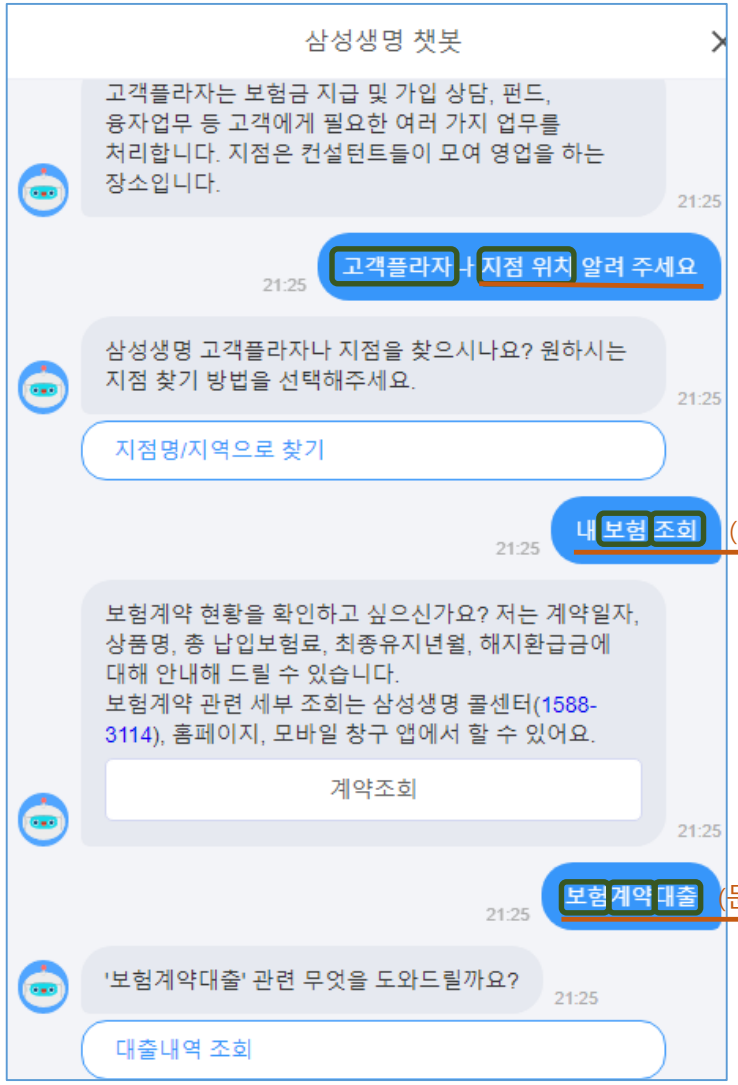
Closed-Domain Chatbot :
키워드와 인텐트를 기반으로
특정 업무를 수행.

Closed Domain Chatbot



Closed-Domain Chatbot :
키워드와 인텐트를 기반으로
특정 업무를 수행.

Closed Domain Chatbot



Closed-Domain Chatbot :
키워드와 인텐트를 기반으로
특정 업무를 수행.

Named Entity Recognition (NER)

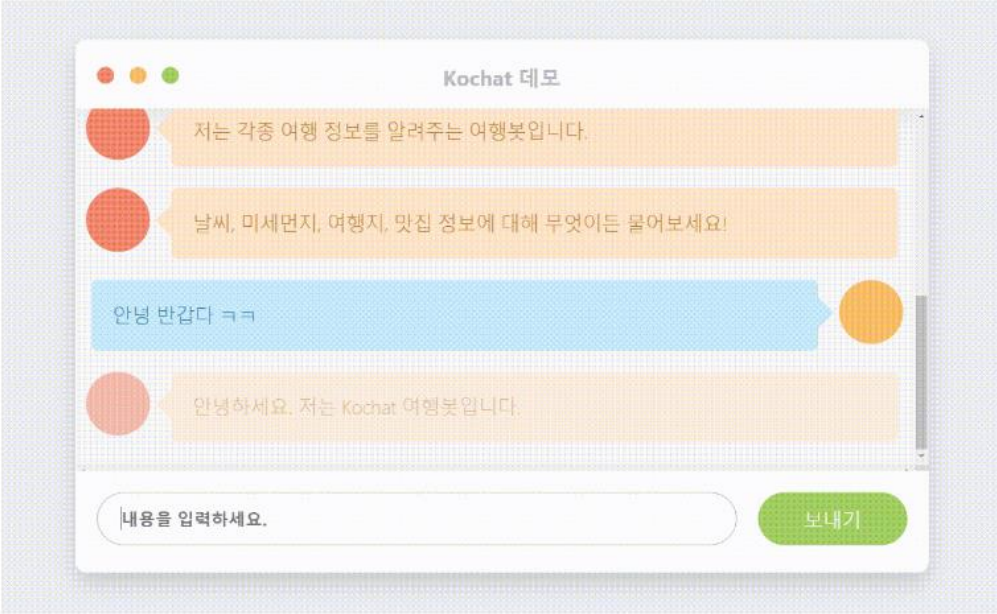
Intent Classification

KoChat

- 딥 러닝 모델을 내장하고 있어 NLP를 몰라도 챗봇을 만들 수 있도록 만든 프레임워크

Kochat
오픈소스 한국어 챗봇 프레임워크

https://github.com/gusdnd852/kochat

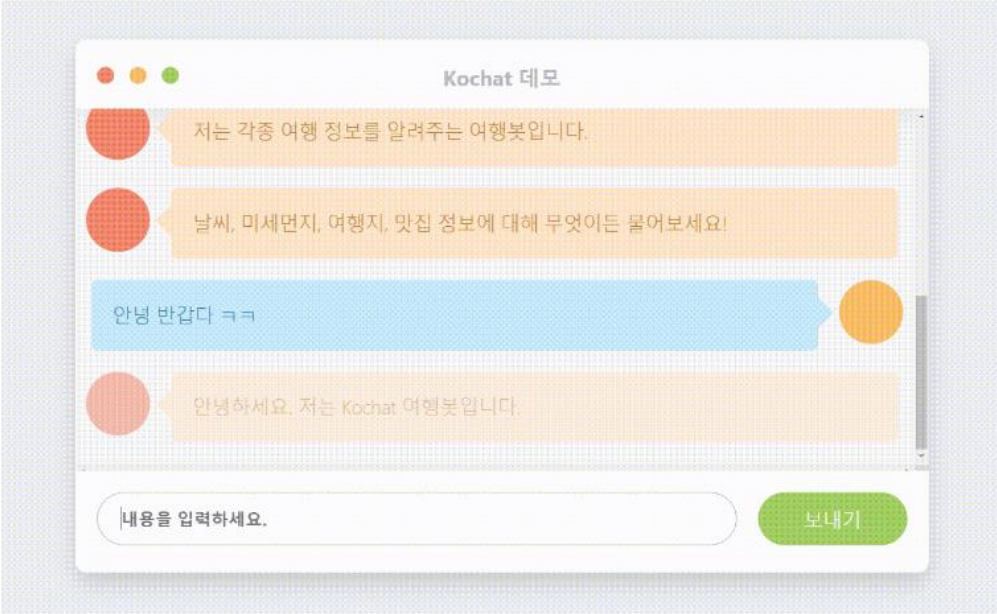


KoChat

- **딥 러닝 모델을 내장**하고 있어 **NLP를 몰라도 챗봇을 만들 수 있도록** 만든 프레임워크

Kochat
오픈소스 한국어 챗봇 프레임워크

https://github.com/gusdnd852/kochat



KoChat

- **딥 러닝 모델을 내장**하고 있어 **NLP를 몰라도 챗봇을 만들 수 있도록** 만든 프레임워크

```python
# 1. 데이터셋 객체 생성
dataset = Dataset(ood=True)

# 2. 임베딩 프로세서 생성
emb = GensimEmbedder(model=embed.FastText())

# 3. 의도(Intent) 분류기 생성
clf = DistanceClassifier(
    model=intent.CNN(dataset.intent_dict),
    loss=CosFace(dataset.intent_dict)
)

# 4. 개체명(Named Entity) 인식기 생성
rcn = EntityRecognizer(
    model=entity.LSTM(dataset.entity_dict),
    loss=CRFLoss(dataset.entity_dict)
)

# 5. 딥러닝 챗봇 RESTful API 학습 & 빌드
kochat = KochatApi(
    dataset=dataset,
    embed_processor=(emb, True),
    intent_classifier=(clf, True),
    entity_recognizer=(rcn, True),
    scenarios=[
        weather, dust, travel, restaurant
    ]
)
```

FastText (Pre-trained Word Embedding)

KoChat

- **딥 러닝 모델을 내장**하고 있어 **NLP를 몰라도 챗봇을 만들 수 있도록** 만든 프레임워크

```python
# 1. 데이터셋 객체 생성
dataset = Dataset(ood=True)

# 2. 임베딩 프로세서 생성
emb = GensimEmbedder(model=embed.FastText())

# 3. 의도(Intent) 분류기 생성
clf = DistanceClassifier(
    model=intent.CNN(dataset.intent_dict),
    loss=CosFace(dataset.intent_dict)
)

# 4. 개체명(Named Entity) 인식기 생성
rcn = EntityRecognizer(
    model=entity.LSTM(dataset.entity_dict),
    loss=CRFLoss(dataset.entity_dict)
)

# 5. 딥러닝 챗봇 RESTful API 학습 & 빌드
kochat = KochatApi(
    dataset=dataset,
    embed_processor=(emb, True),
    intent_classifier=(clf, True),
    entity_recognizer=(rcn, True),
    scenarios=[
        weather, dust, travel, restaurant
    ]
)
```

FastText (Pre-trained Word Embedding)

Text Classification Using CNN

KoChat

- **딥 러닝 모델을 내장**하고 있어 **NLP를 몰라도 챗봇을 만들 수 있도록** 만든 프레임워크

```python
# 1. 데이터셋 객체 생성
dataset = Dataset(ood=True)

# 2. 임베딩 프로세서 생성
emb = GensimEmbedder(model=embed.FastText())

# 3. 의도(Intent) 분류기 생성
clf = DistanceClassifier(
    model=intent.CNN(dataset.intent_dict),
    loss=CosFace(dataset.intent_dict)
)

# 4. 개체명(Named Entity) 인식기 생성
rcn = EntityRecognizer(
    model=entity.LSTM(dataset.entity_dict),
    loss=CRFLoss(dataset.entity_dict)
)

# 5. 딥러닝 챗봇 RESTful API 학습 & 빌드
kochat = KochatApi(
    dataset=dataset,
    embed_processor=(emb, True),
    intent_classifier=(clf, True),
    entity_recognizer=(rcn, True),
    scenarios=[
        weather, dust, travel, restaurant
    ]
)
```

FastText (Pre-trained Word Embedding)

Text Classification Using CNN

LSTM

CRF

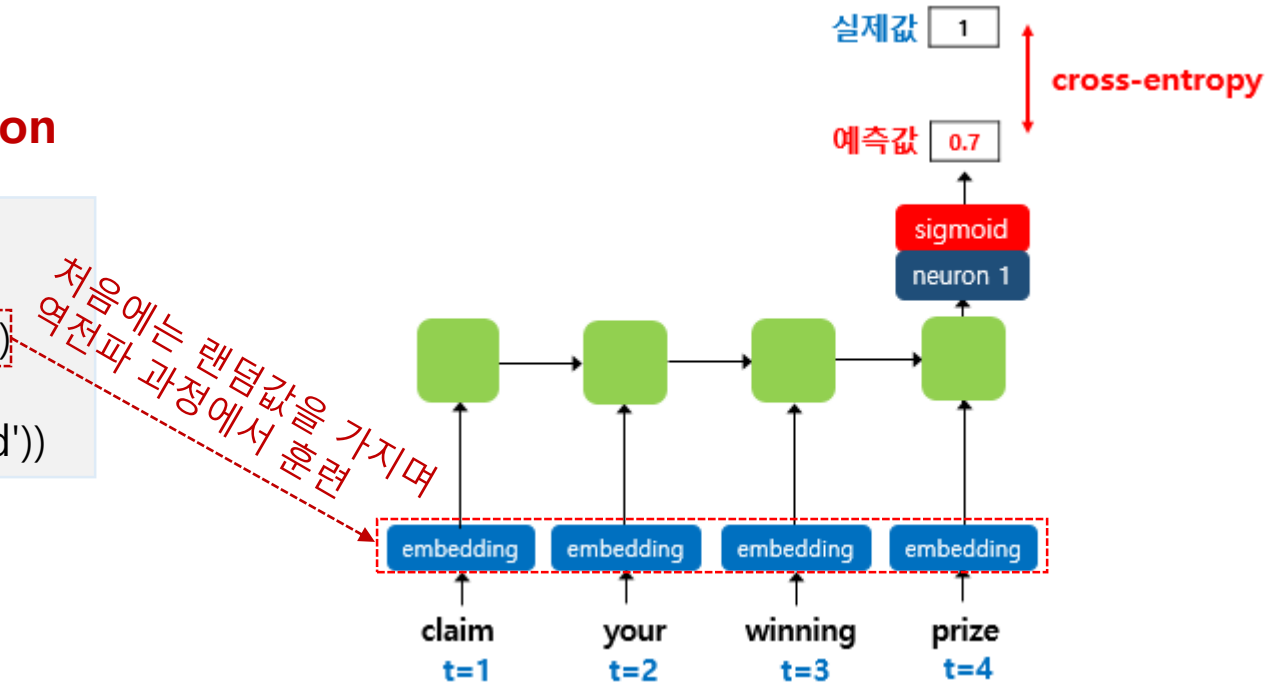
Pre-trained Word Embedding

Embedding layer (랜덤 초기화 방법)

케라스, 파이토치 등의 딥 러닝 프레임워크에서는 임베딩 층(embedding layer)을 제공.
초기에 모든 단어의 임베딩 벡터값은 랜덤 초기화 되며, 모델이 역전파하는 과정에서 학습된다.

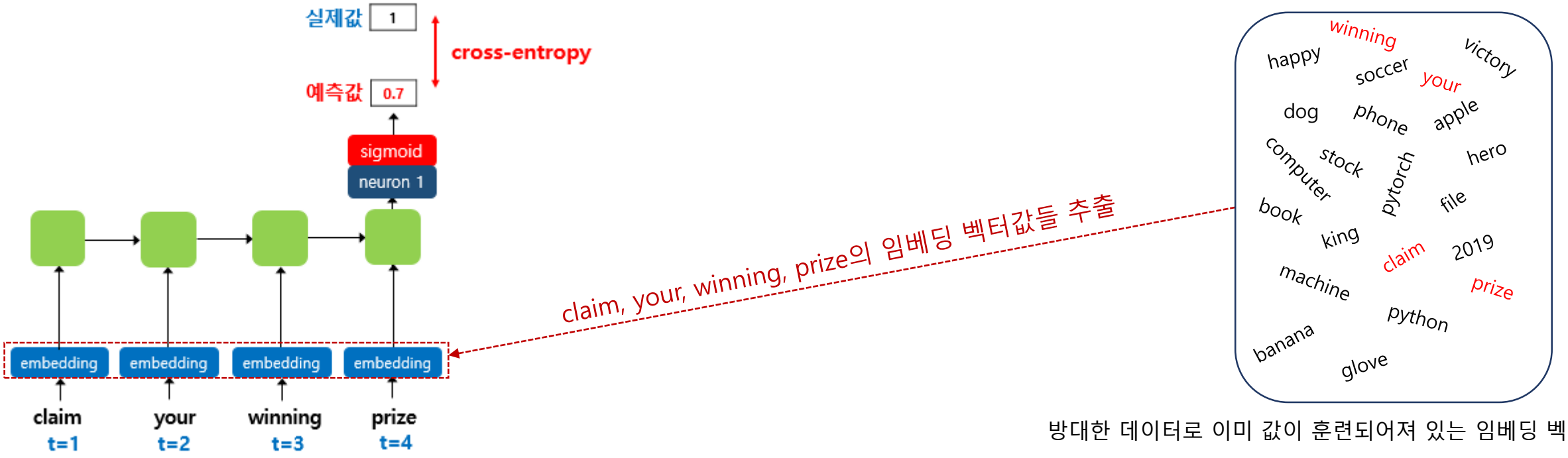
Text Classification Implemententation

```
model = Sequential()
# 임베딩 벡터의 차원은 256
model.add(Embedding(vocab_size, 256))
model.add(LSTM(256))
model.add(Dense(1, activation='sigmoid'))
```



Pre-trained Word Embedding

- 방대한 양의 텍스트 데이터로 이미 훈련되어져 있는 임베딩 벡터값들을 갖고와서 모델의 입력으로 사용하는 것이 모델의 성능을 높이는 시도.
- 이때 이 임베딩 벡터들은 Word2Vec, FastText, GloVe 등의 알고리즘으로 훈련된 벡터들.

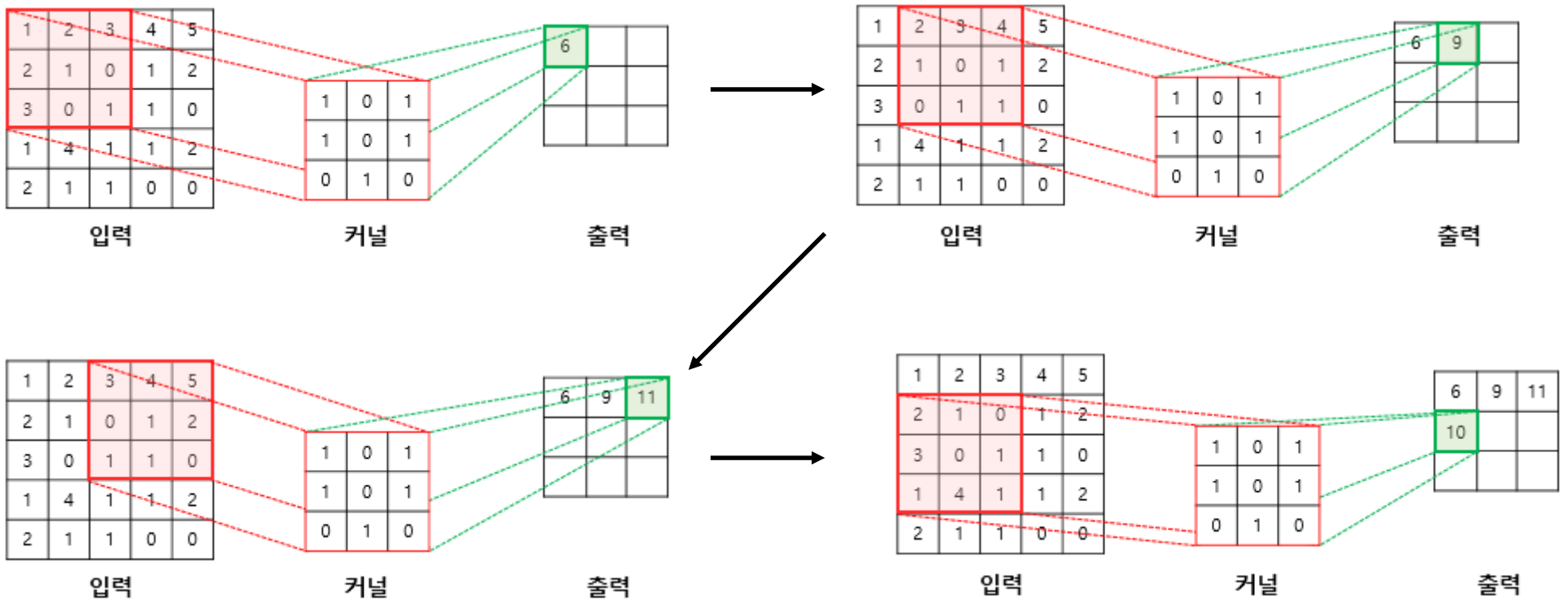


방대한 데이터로 이미 값이 훈련되어져 있는 임베딩 벡터들

1D CNN

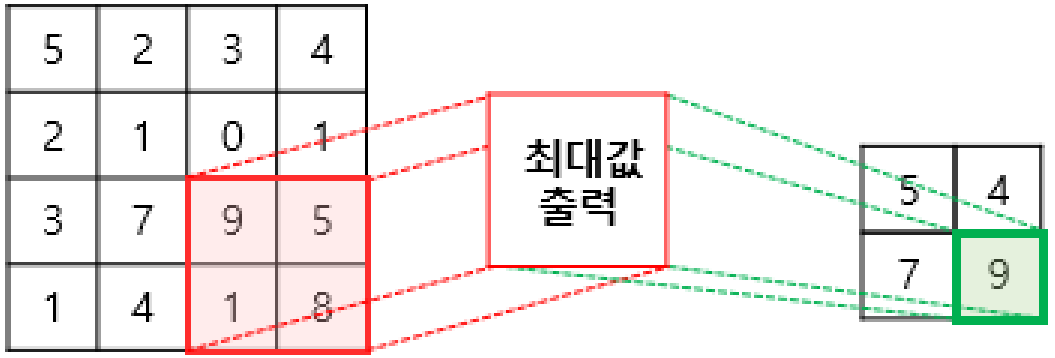
2D Convolution (in Image Processing)

- 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

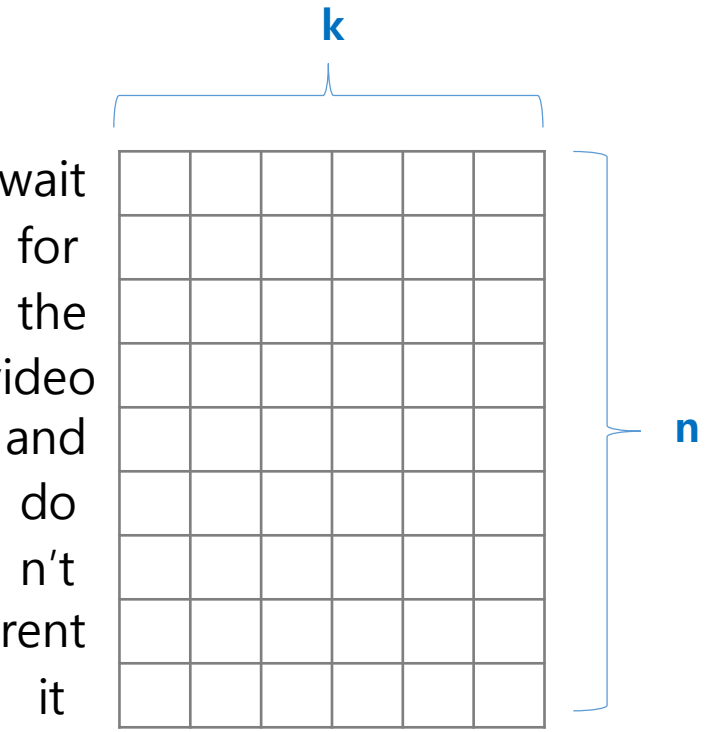
CNN for Text Classification

- **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

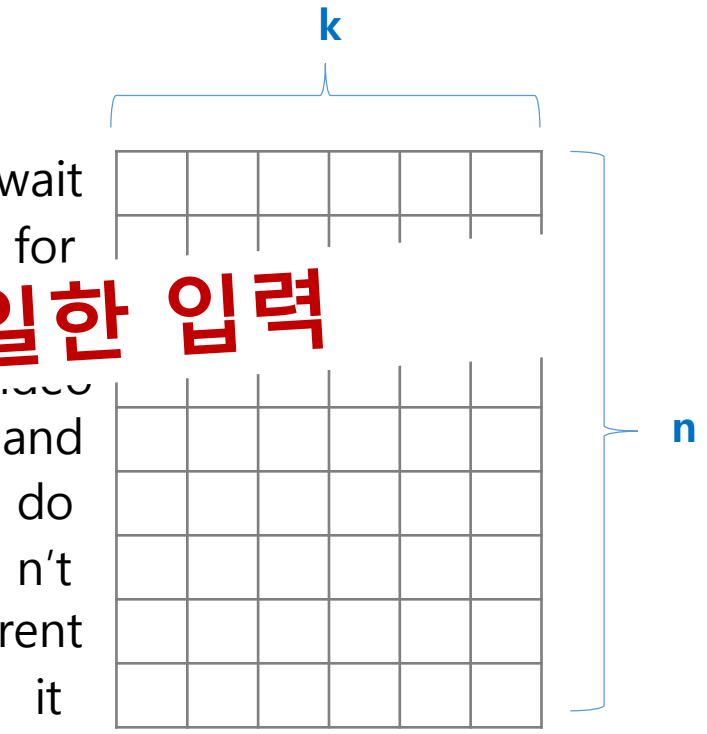
- **CNN의 입력은 문장 또는 문서의 각 단어가 Embedding을 거치고 난 후의 문장 테이블.**
- **n을 문장 길이, k를 임베딩 벡터의 차원이라고 하였을 때, CNN의 입력은 다음과 같이 n × k 행렬.**

입력 문장

wait for the video and

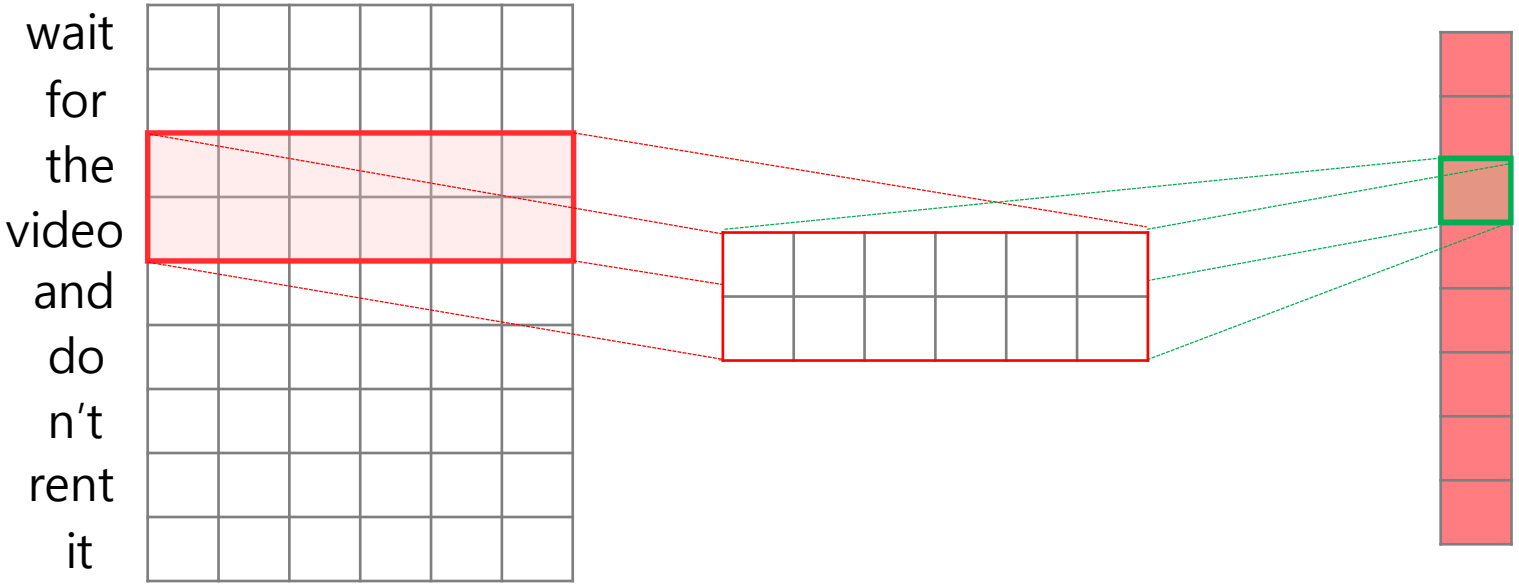
Tokenization /

사실 이는 RNN과 동일한 입력



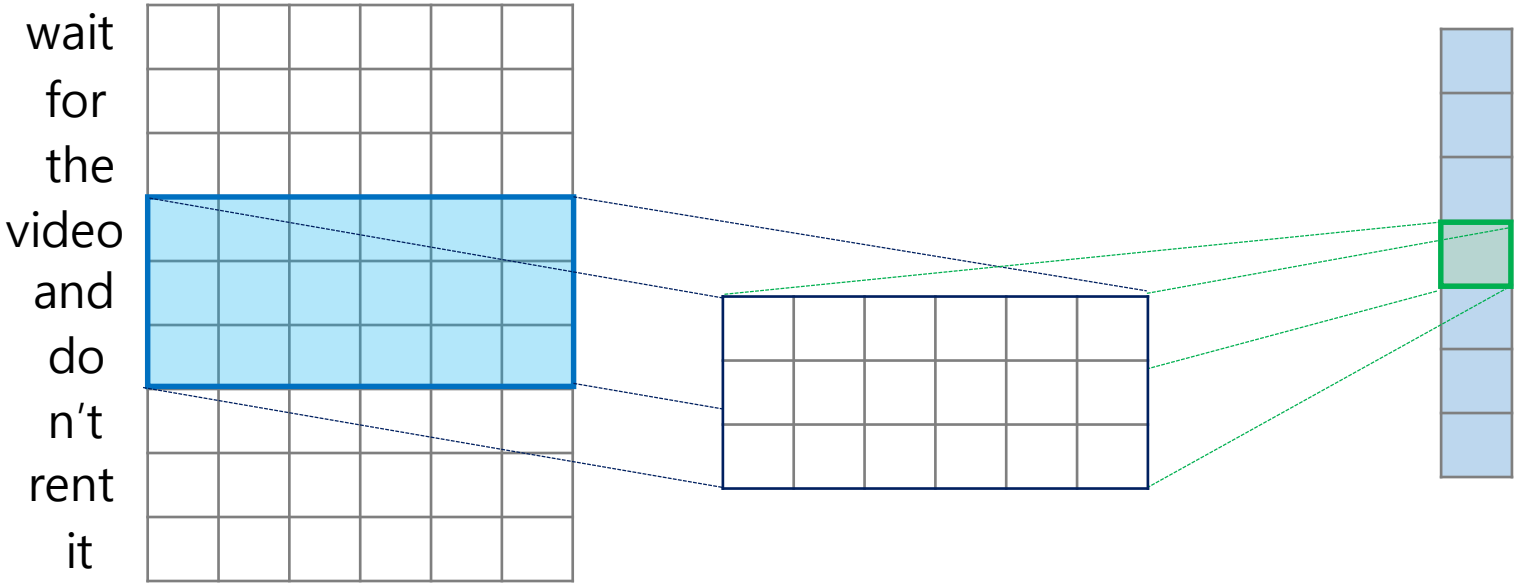
1D Convolution (in NLP)

- 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)

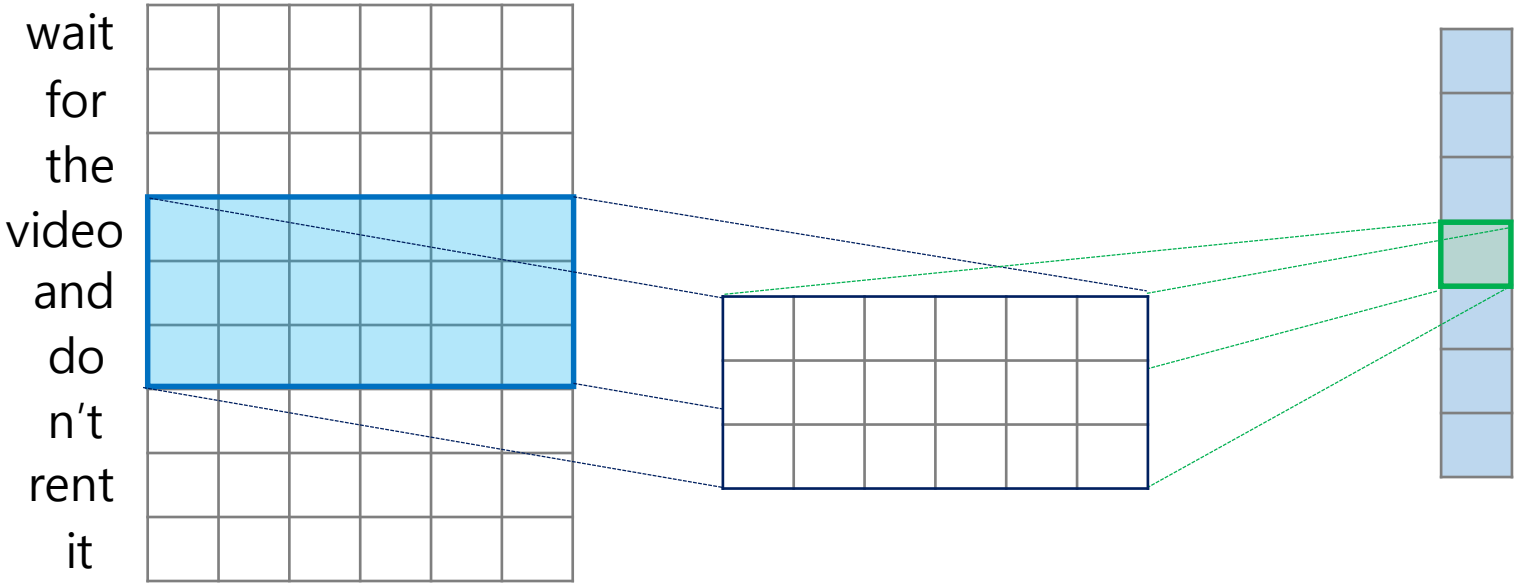
- 오직 한 방향으로만 윈도우를 슬라이딩하므로 1D라고 한다.
- 커널(kernel) 또는 필터(filter)라는 행렬로 임베딩 행렬의 가장 맨 위부터 가장 맨 아래까지 (↓)
  순차적으로 훑으며 겹쳐지는 부분의 임베딩 행렬과 커널의 원소의 값을 곱해서 모두 더한 값을 출력으로 한다.



커널의 크기가 커질수록, 한 번에 좀 더 많은 단어를 참고한다.

1D Convolution (in NLP)

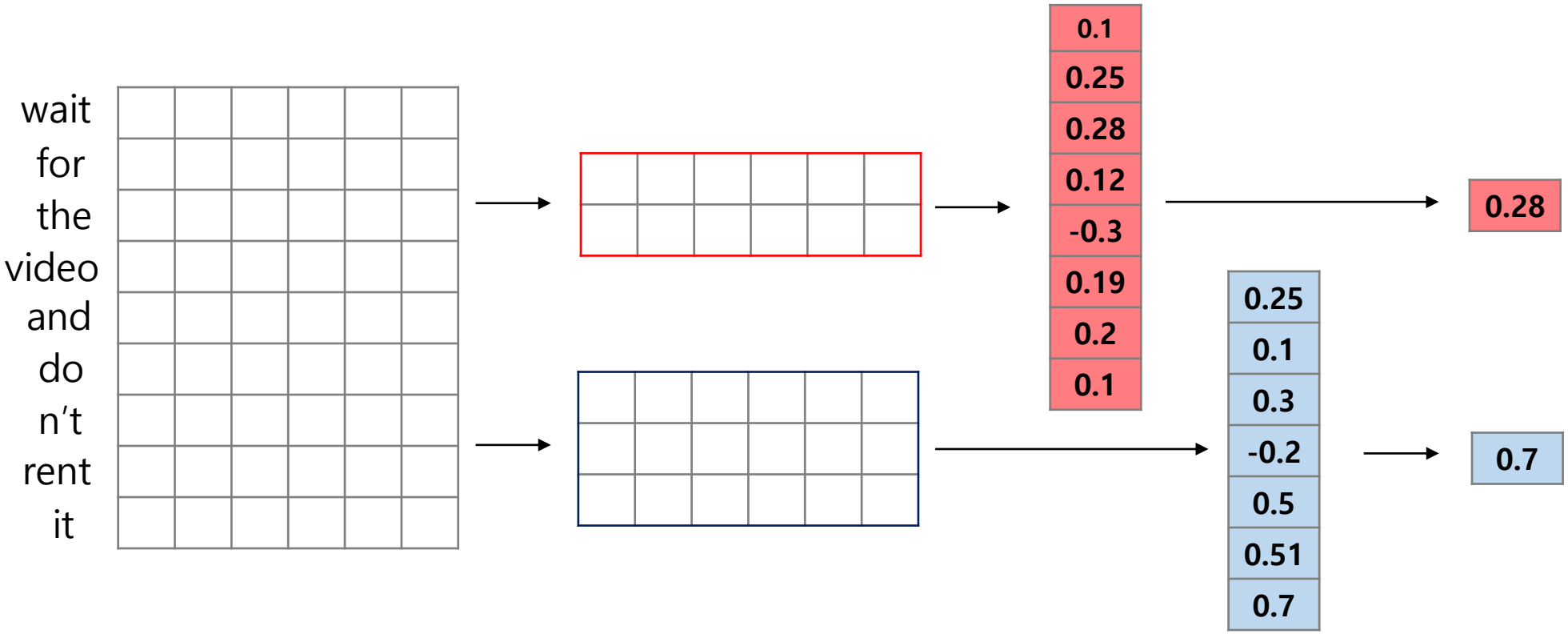
- 오직 한 방향으로만 윈도우를 슬라이딩하므로 1D라고 한다.
- 커널(kernel) 또는 필터(filter)라는 행렬로 임베딩 행렬의 가장 맨 위부터 가장 맨 아래까지 (↓)
  순차적으로 훑으며 겹쳐지는 부분의 임베딩 행렬과 커널의 원소의 값을 곱해서 모두 더한 값을 출력으로 한다.



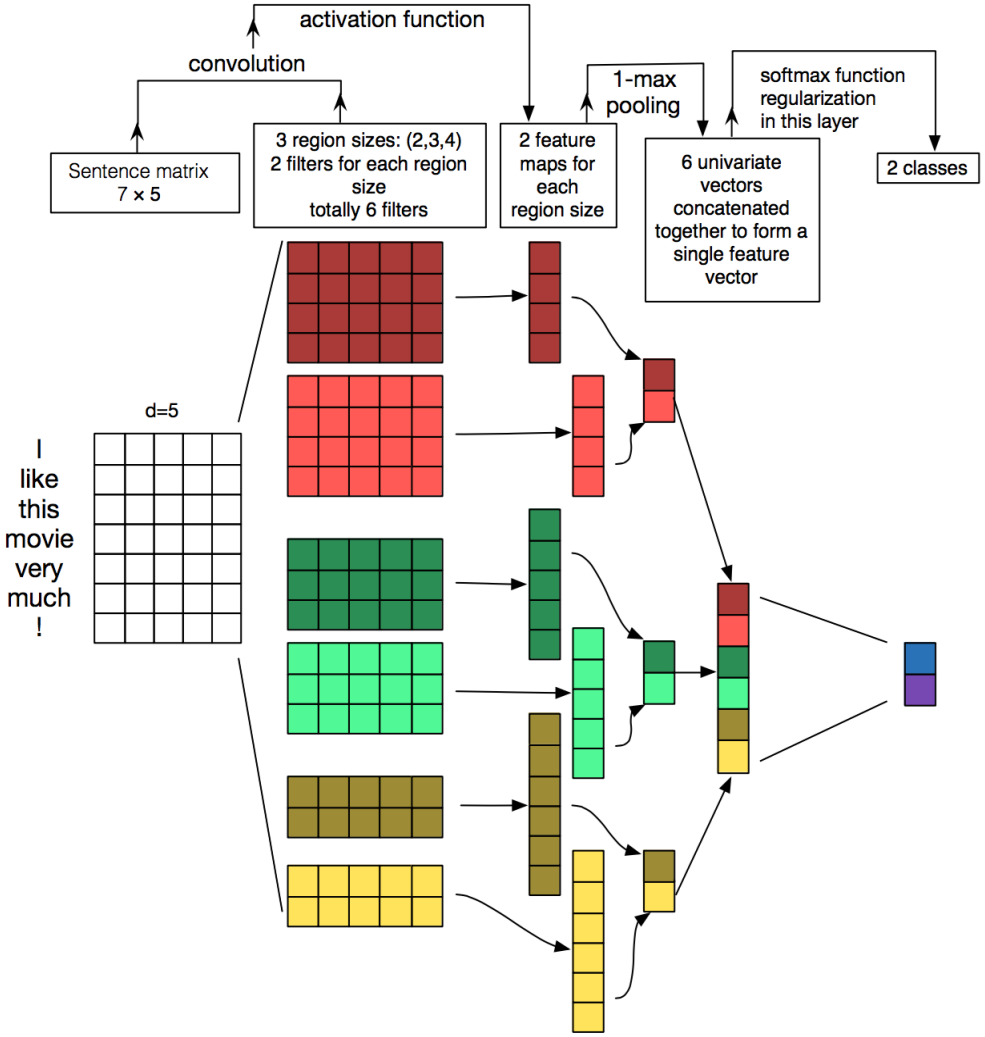
커널의 크기가 커질수록, 참고하는 n-gram의 n이 커진다.

Max-pooling

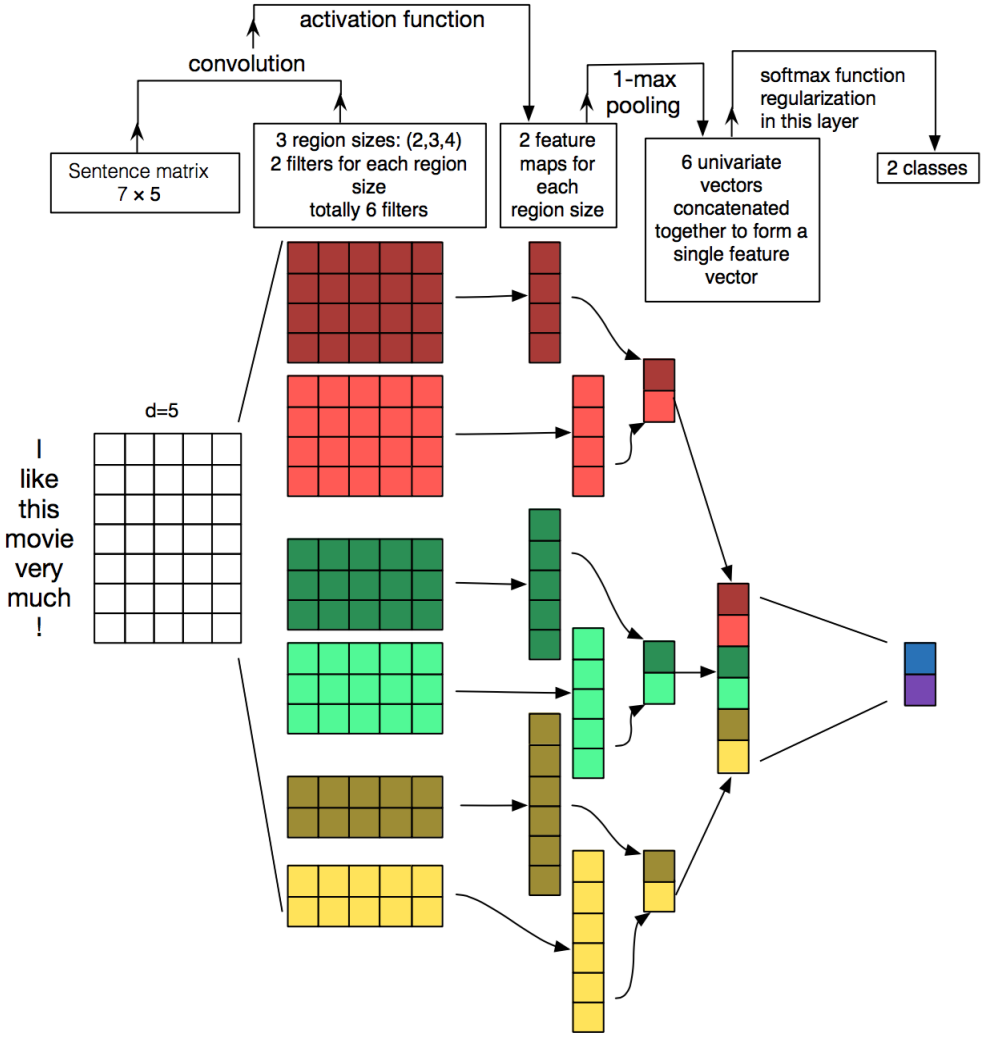
- 추출한 벡터 중에서 가장 중요한 특성만을 뽑아내기 위해서 수행한다. (벡터 -> 스칼라값)
- 커널의 크기에 따라서 합성곱 결과로 얻은 벡터의 길이는 다를 수 있지만 이와 상관없이 수행할 수 있는 연산이다.



1D CNN Overview



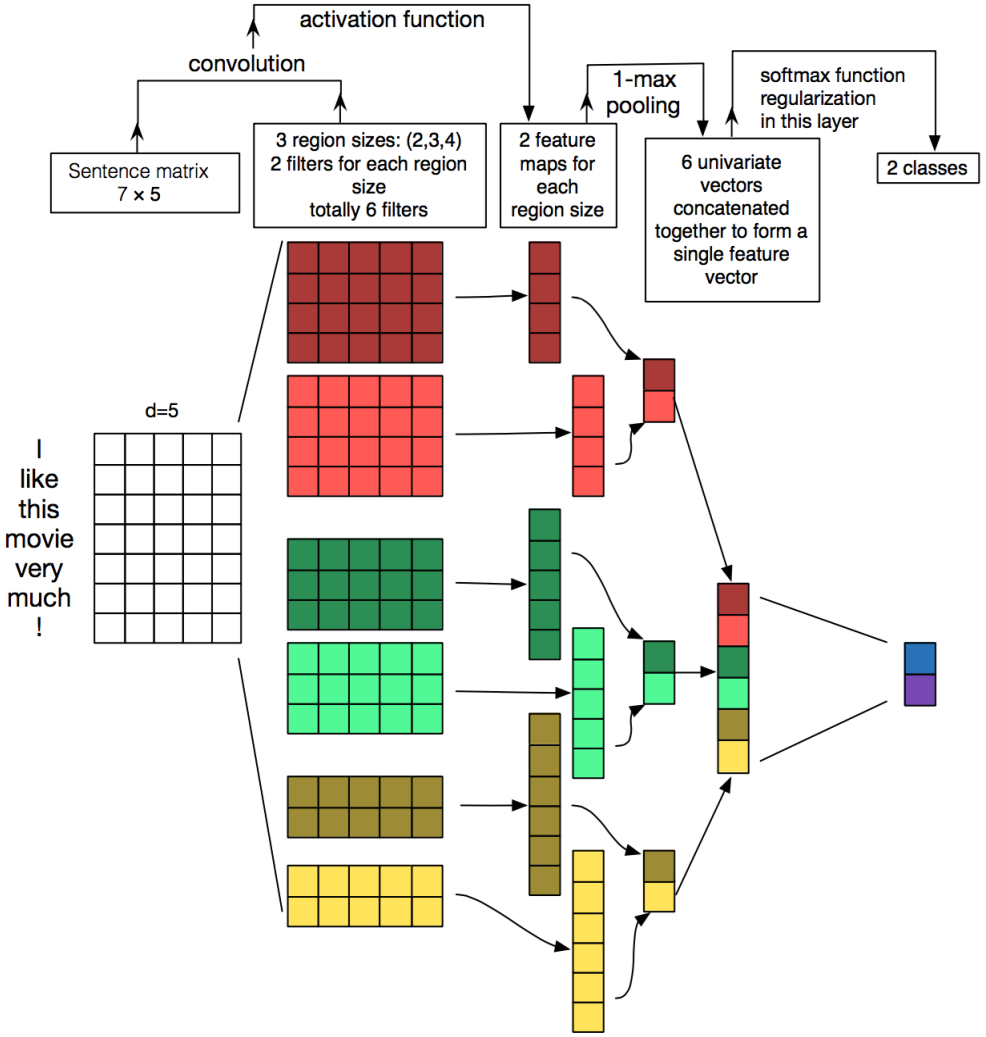
1D CNN Overview



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 :

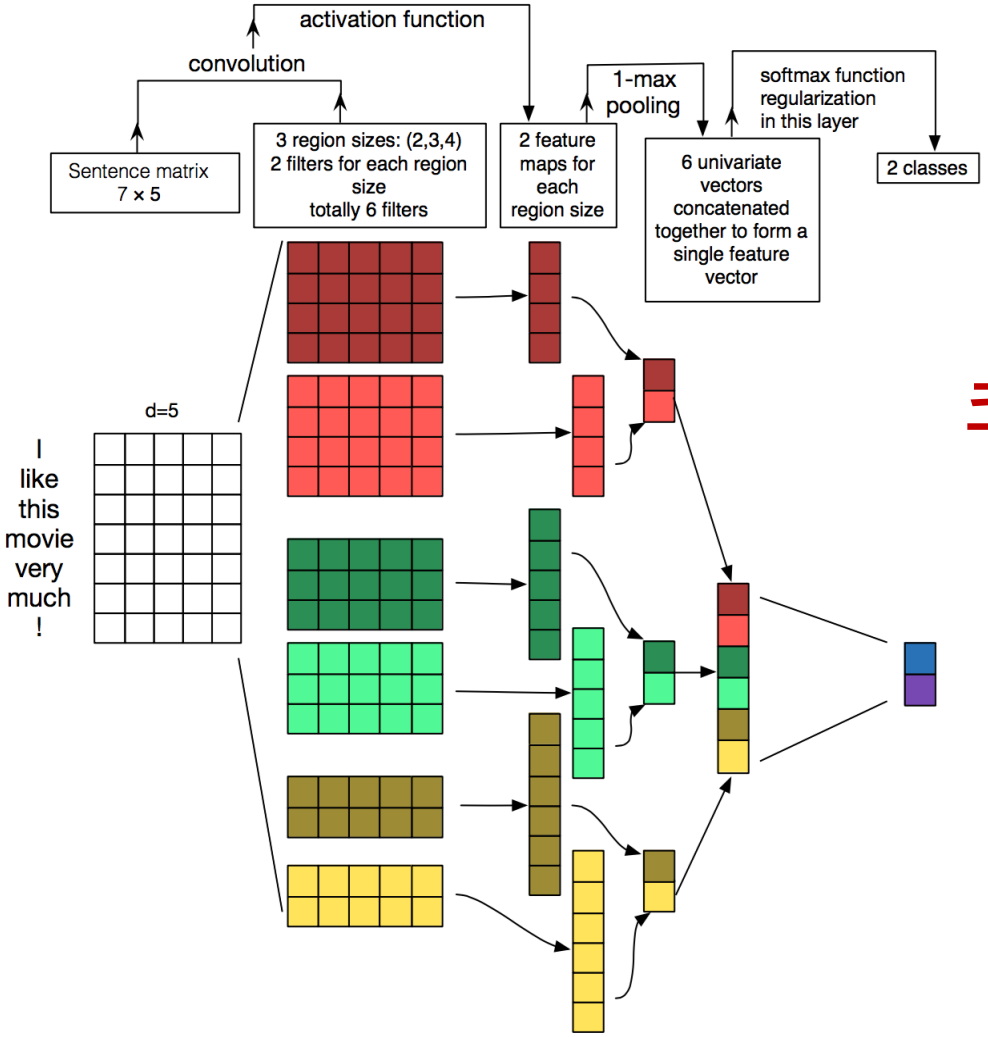
1D CNN Overview



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

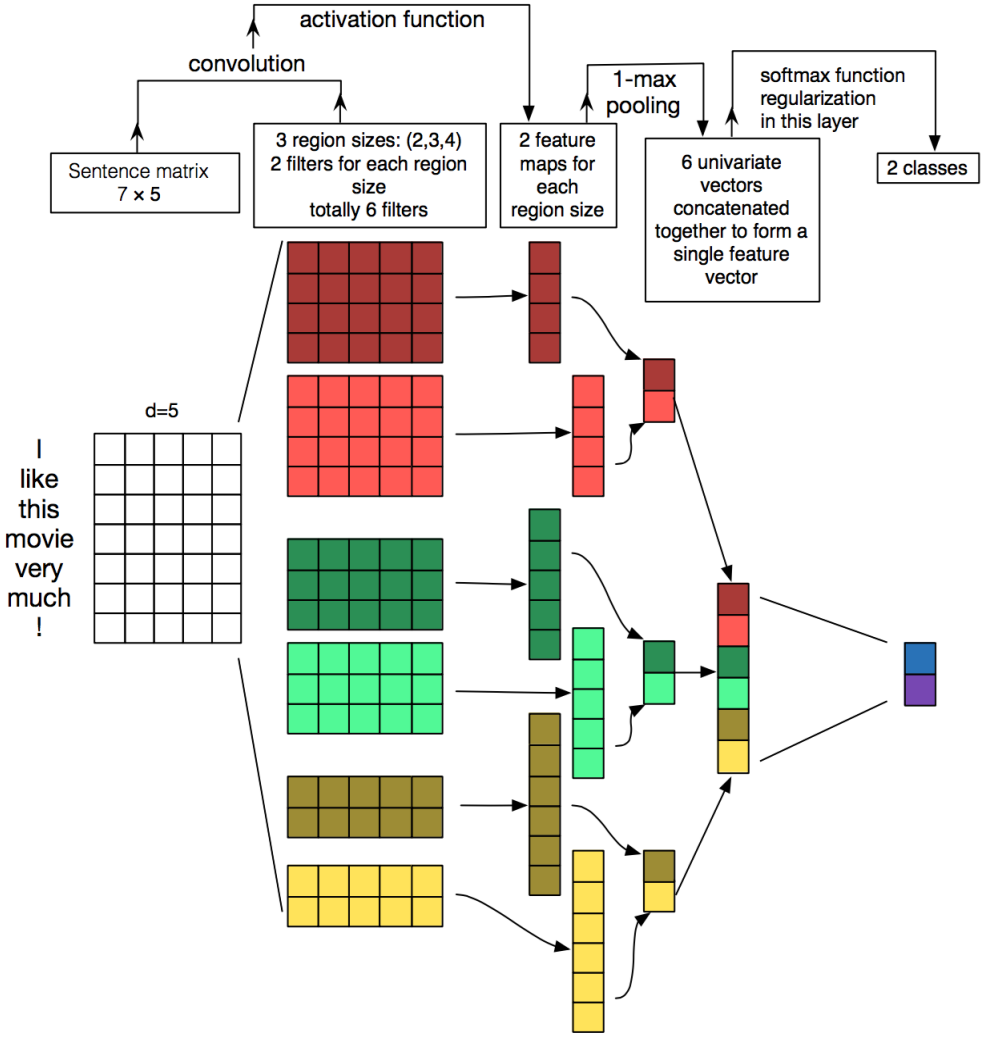
Embedding layer의 파라미터 : 알 수 없음
Conv1D의 파라미터 : 96개 (풀이 미공유)
Dense의 파라미터 : 14개 (풀이 미공유)

1D CNN Overview



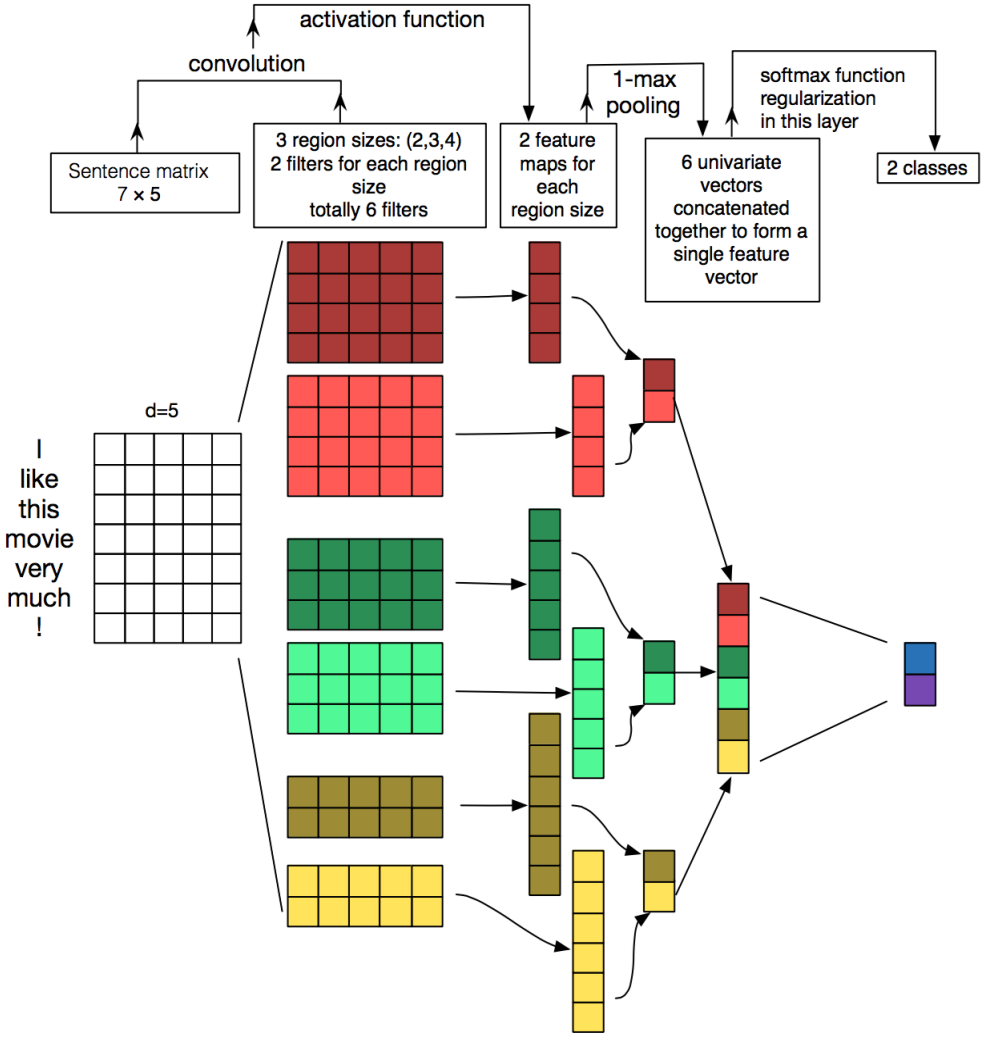
코드로 작성하면 어떻게 작성할까요?

1D CNN Overview




```
max_len = 7
vocab_size = 20 # 임의 선정
embedding_dim = 5
num_filters = 2

model_input = Input(shape = (max_len,))
z = Embedding(vocab_size, embedding_dim, input_length = max_len, name="embedding")(model_input)

conv_blocks = []

for sz in [2, 3, 4]:
    conv = Conv1D(filters = num_filters,
                          kernel_size = sz,
                          padding = "valid",
                          activation = "relu",
                          strides = 1)(z)
    conv = GlobalMaxPooling1D()(conv)
    conv = Flatten()(conv)
    conv_blocks.append(conv)

z = Concatenate()(conv_blocks) if len(conv_blocks) > 1 else conv_blocks[0]
model_output = Dense(2, activation="sigmoid")(z)

model = Model(model_input, model_output)
model.compile(loss="binary_crossentropy", optimizer="adam", metrics=["acc"])
model.summary()
```

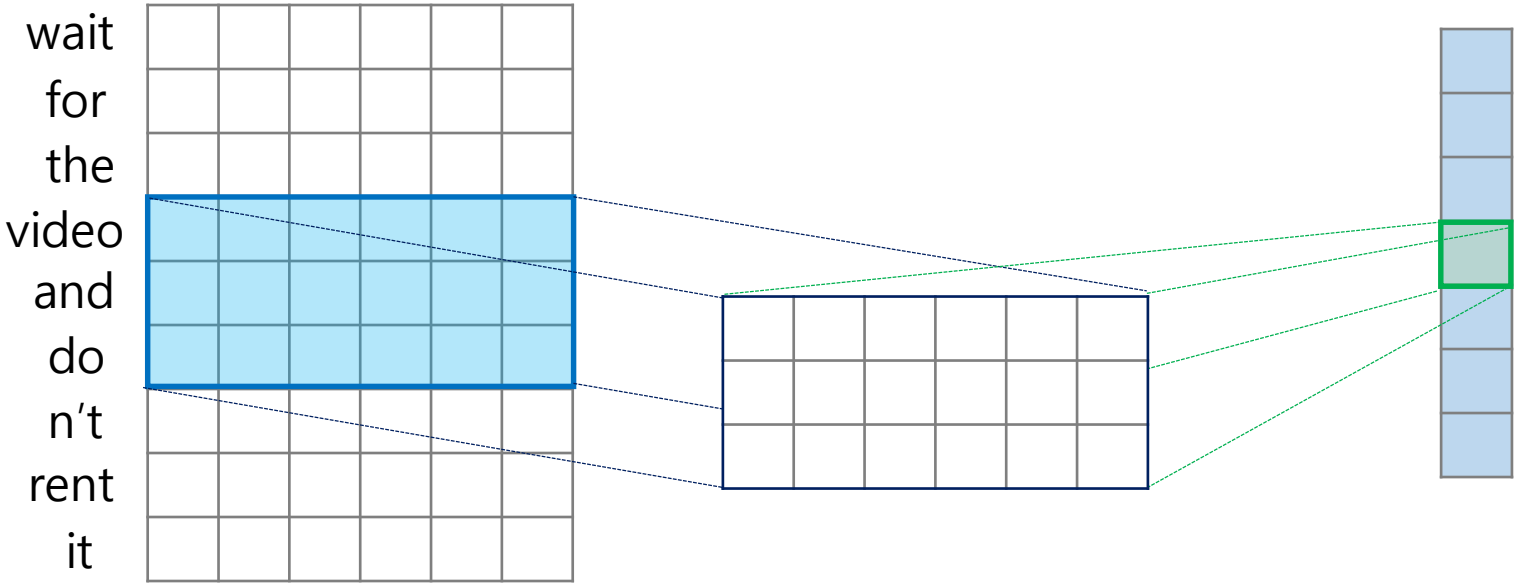
1D CNN Overview




```
Model: "model"
_________________________________________________________________________________________________
Layer (type)                 Output Shape              Param #   Connected to
=================================================================================================
input_5 (InputLayer)         [(None, 7)]               0
_________________________________________________________________________________________________
embedding (Embedding)        (None, 7, 5)              100       input_5[0][0]
_________________________________________________________________________________________________
conv1d_4 (Conv1D)            (None, 6, 2)              22        embedding[0][0]
_________________________________________________________________________________________________
conv1d_5 (Conv1D)            (None, 5, 2)              32        embedding[0][0]
_________________________________________________________________________________________________
conv1d_6 (Conv1D)            (None, 4, 2)              42        embedding[0][0]
_________________________________________________________________________________________________
global_max_pooling1d_4 (GlobalM (None, 2)              0         conv1d_4[0][0]
_________________________________________________________________________________________________
global_max_pooling1d_5 (GlobalM (None, 2)              0         conv1d_5[0][0]
_________________________________________________________________________________________________
global_max_pooling1d_6 (GlobalM (None, 2)              0         conv1d_6[0][0]
_________________________________________________________________________________________________
flatten_4 (Flatten)          (None, 2)                 0         global_max_pooling1d_4[0][0]
_________________________________________________________________________________________________
flatten_5 (Flatten)          (None, 2)                 0         global_max_pooling1d_5[0][0]
_________________________________________________________________________________________________
flatten_6 (Flatten)          (None, 2)                 0         global_max_pooling1d_6[0][0]
_________________________________________________________________________________________________
concatenate_1 (Concatenate)  (None, 6)                 0         flatten_4[0][0]
                                                                 flatten_5[0][0]
                                                                 flatten_6[0][0]
_________________________________________________________________________________________________
dense (Dense)                (None, 2)                 14        concatenate_1[0][0]
=================================================================================================
Total params: 210
Trainable params: 210
Non-trainable params: 0
_________________________________________________________________________________________________
```

Word-level 1D Convolution (in NLP)

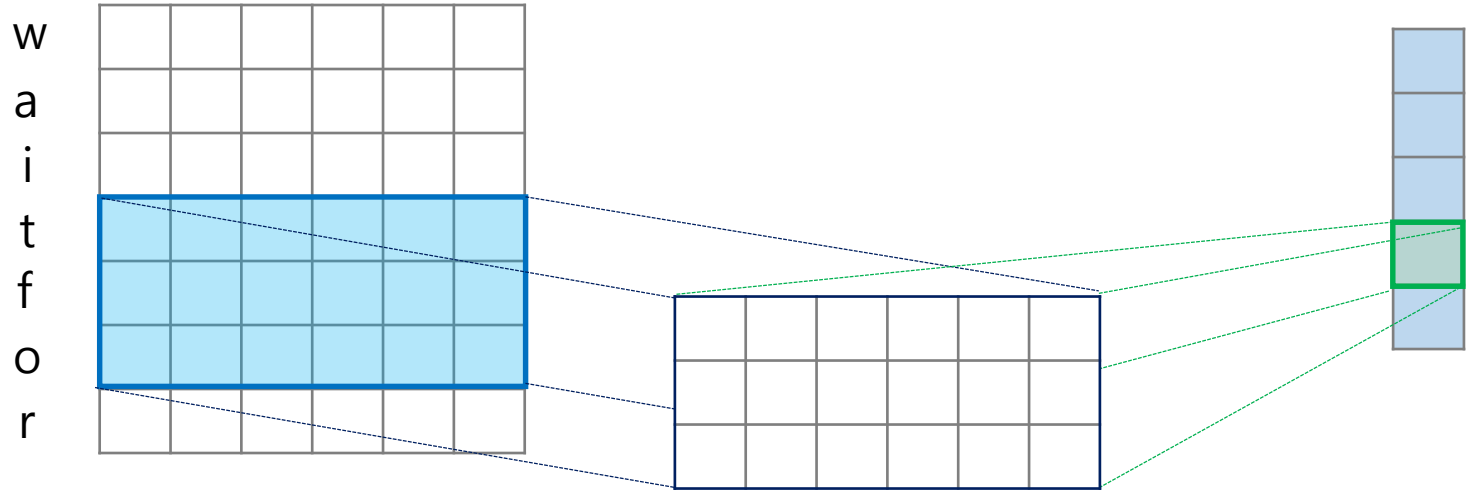
- 오직 한 방향으로만 윈도우를 슬라이딩하므로 1D라고 한다.
- 커널(kernel) 또는 필터(filter)라는 행렬로 임베딩 행렬의 가장 맨 위부터 가장 맨 아래까지 (↓)
  순차적으로 훑으며 겹쳐지는 부분의 임베딩 행렬과 커널의 원소의 값을 곱해서 모두 더한 값을 출력으로 한다.



커널의 크기가 커질수록, 참고하는 n-gram의 n이 커진다.

Char-level 1D Convolution (in NLP)

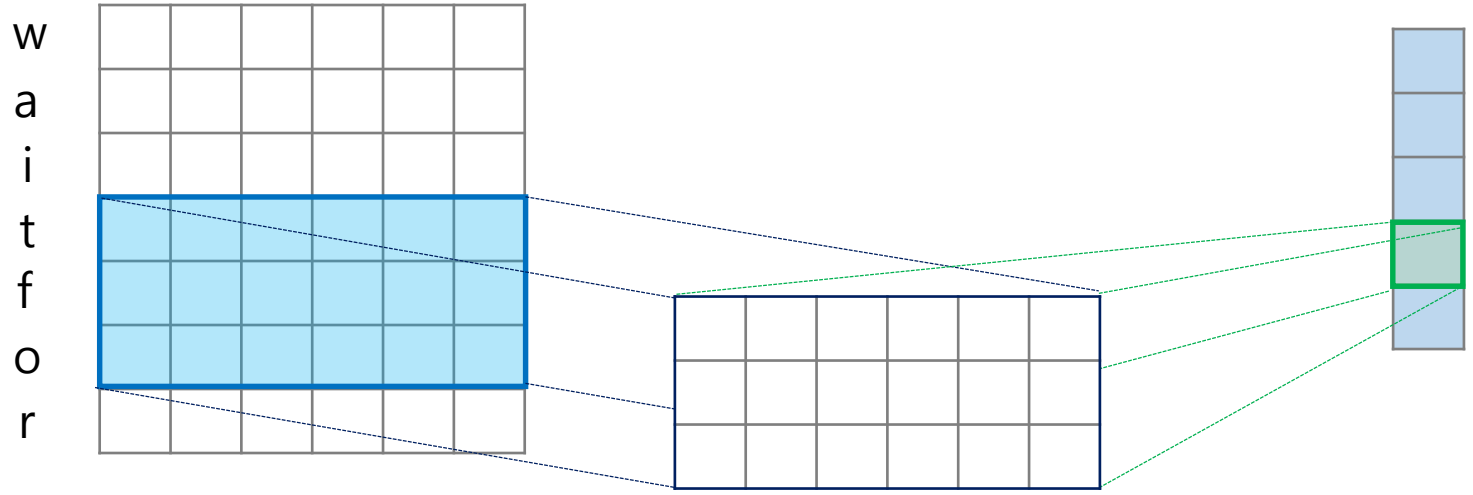
- 단어 단위 CNN이 아닌 Character 단위로 CNN을 수행한다.
- 이제 Character로 임베딩 행렬을 만든다.



OOV에 강건한 모델을 만들 수 있다.

Char-level 1D Convolution (in NLP)

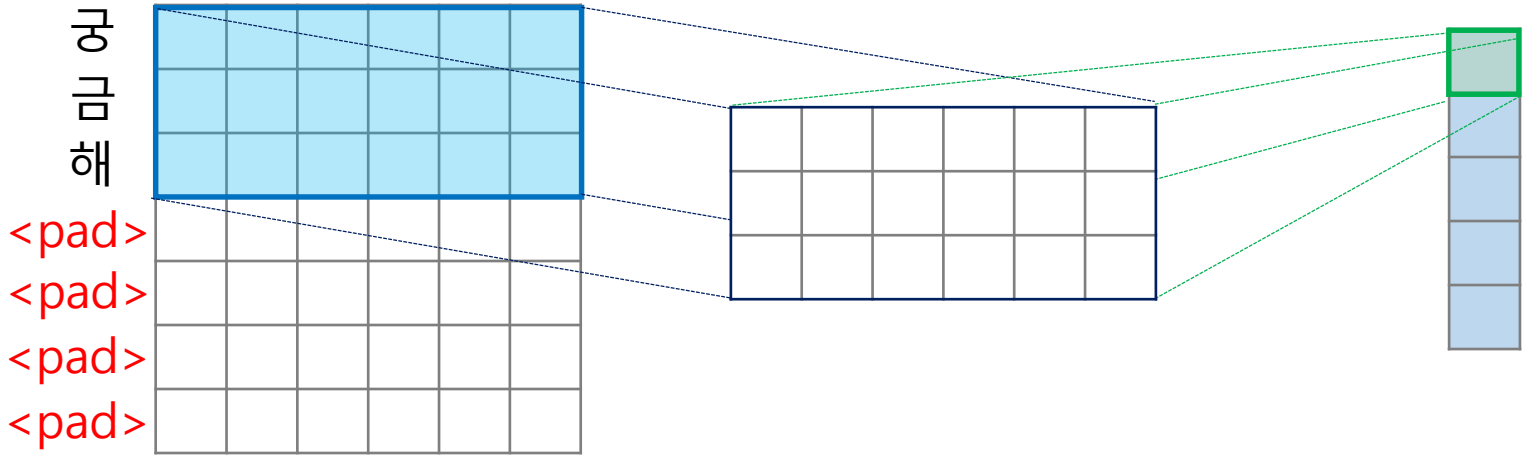
- 단어 단위 CNN이 아닌 Character 단위로 CNN을 수행한다.
- 이제 Character로 임베딩 행렬을 만든다.



OOV에 강건한 모델을 만들 수 있다.

음절-level 1D Convolution (in NLP)

- 단어 단위 CNN이 아닌 음절 단위로 CNN을 수행한다.
- 이제 음절로 임베딩 행렬을 만든다.

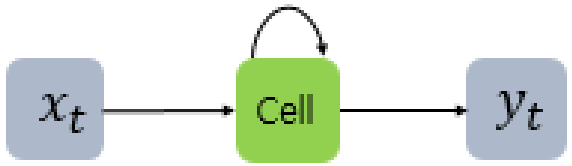


OOV에 강건한 모델을 만들 수 있다.

RNN

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

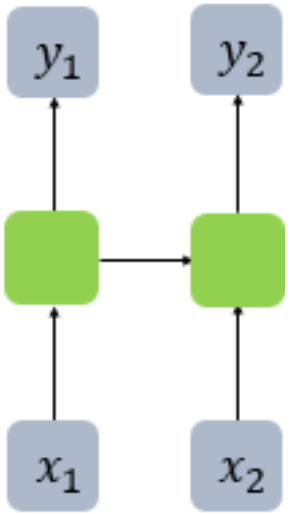


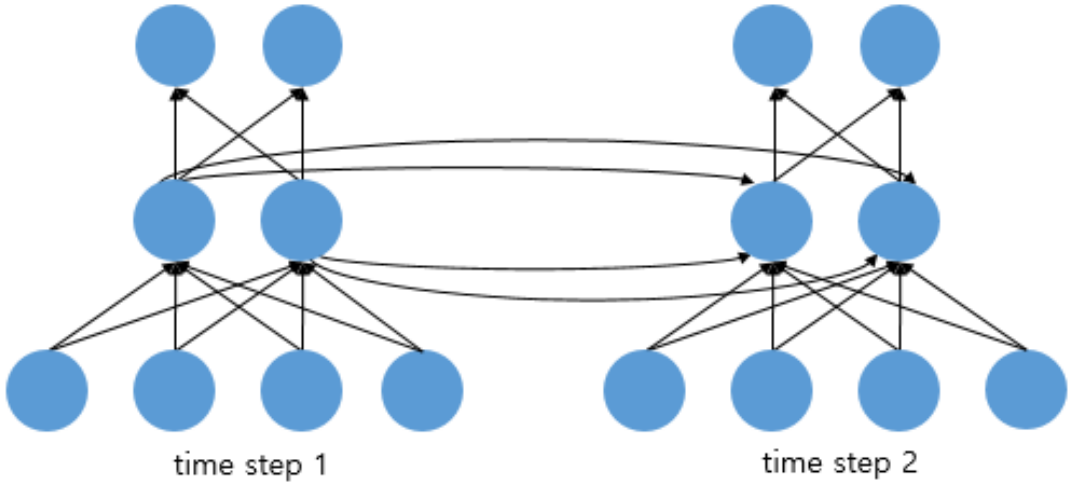
Recurrent Neural Networks have loops.

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

Recurrent Neural Network 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.

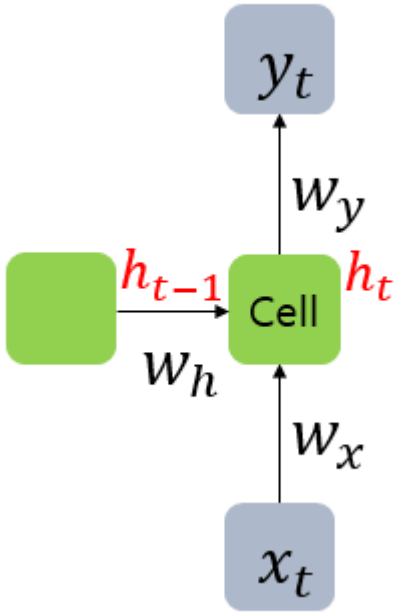




unrolled Recurrent Neural Network

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$(W_h h_t + b)$

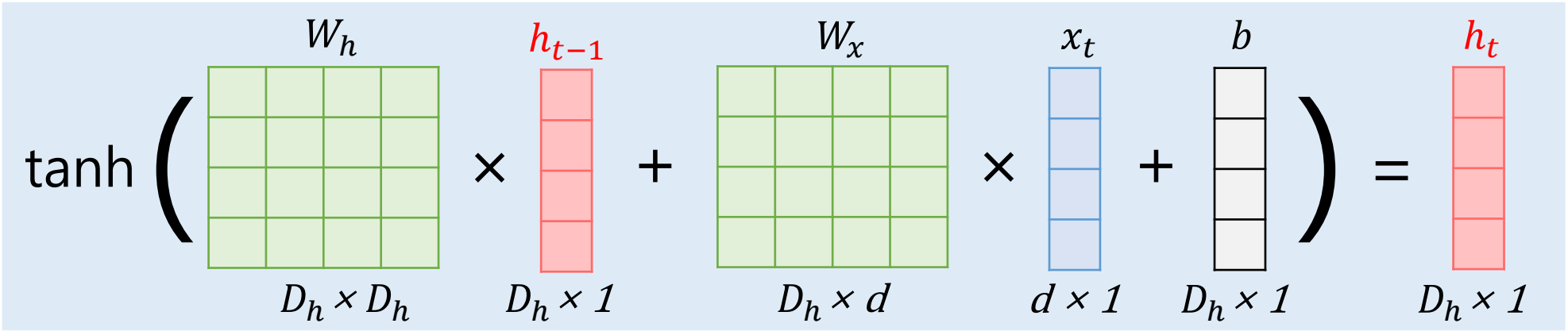
은닉층 : $h_t = tanh(W_h h_{t-1} + W_x x_t + b)$

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.

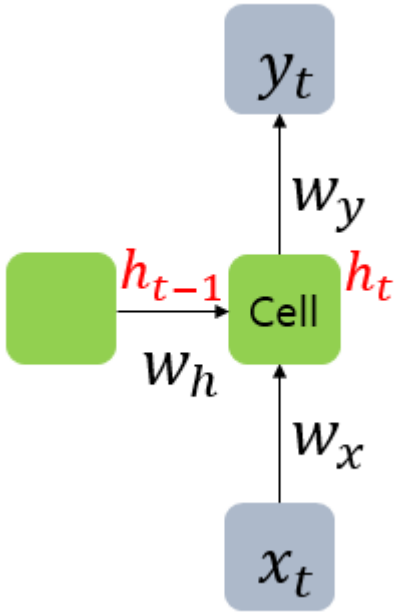
은닉층 : $h_t = tanh(W_h h_{t-1} + W_x x_t + b)$



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

파라미터 추정하기

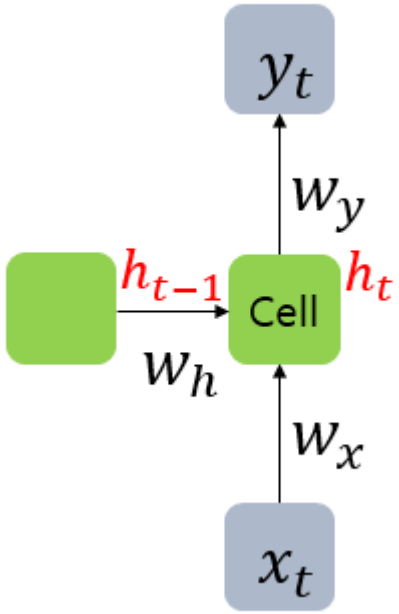
- 단어 집합(Vocabulary)의 크기가 5,000이고 임베딩 벡터의 차원이 100이다.
- Simple RNN을 사용하며 hidden state의 크기는 128이다.
- 은닉층은 단 1개만 존재한다.
- 이진 분류를 사용하며 활성화 함수는 시그모이드 함수를 사용한다.



이때, 파라미터의 총 개수는?

파라미터 추정하기

- 단어 집합(Vocabulary)의 크기가 5,000이고 임베딩 벡터의 차원이 100이다.
- Simple RNN을 사용하며 hidden state의 크기는 128이다.
- 은닉층은 단 1개만 존재한다.
- 이진 분류를 사용하며 활성화 함수는 시그모이드 함수를 사용한다.

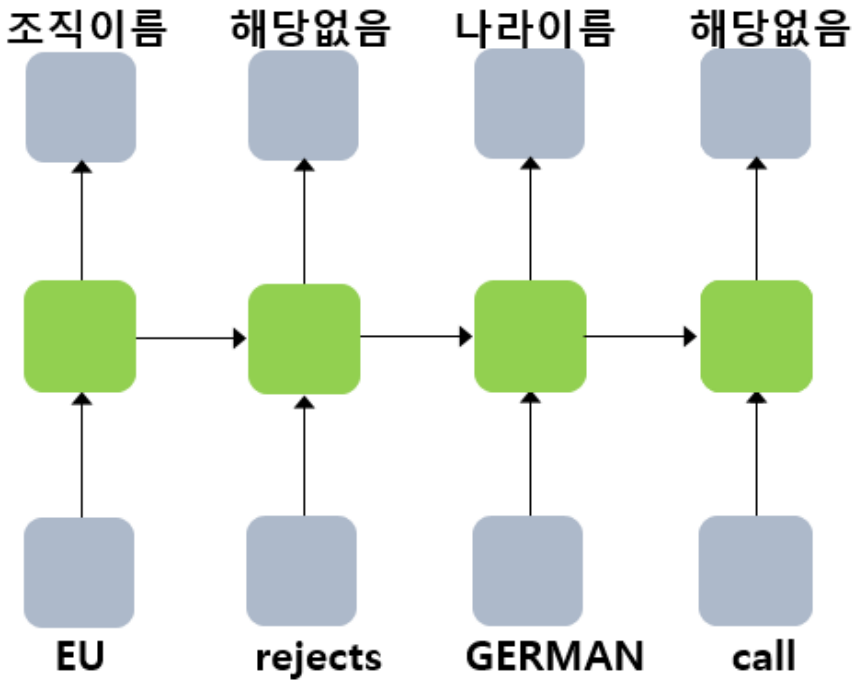


이때, 파라미터의 총 개수는?

Embedding layer의 파라미터 : 500,000 (풀이 미공유)
RNN의 파라미터 : 29,312 (풀이 미공유)
Dense의 파라미터 : 129 (풀이 미공유)

many-to-many RNN

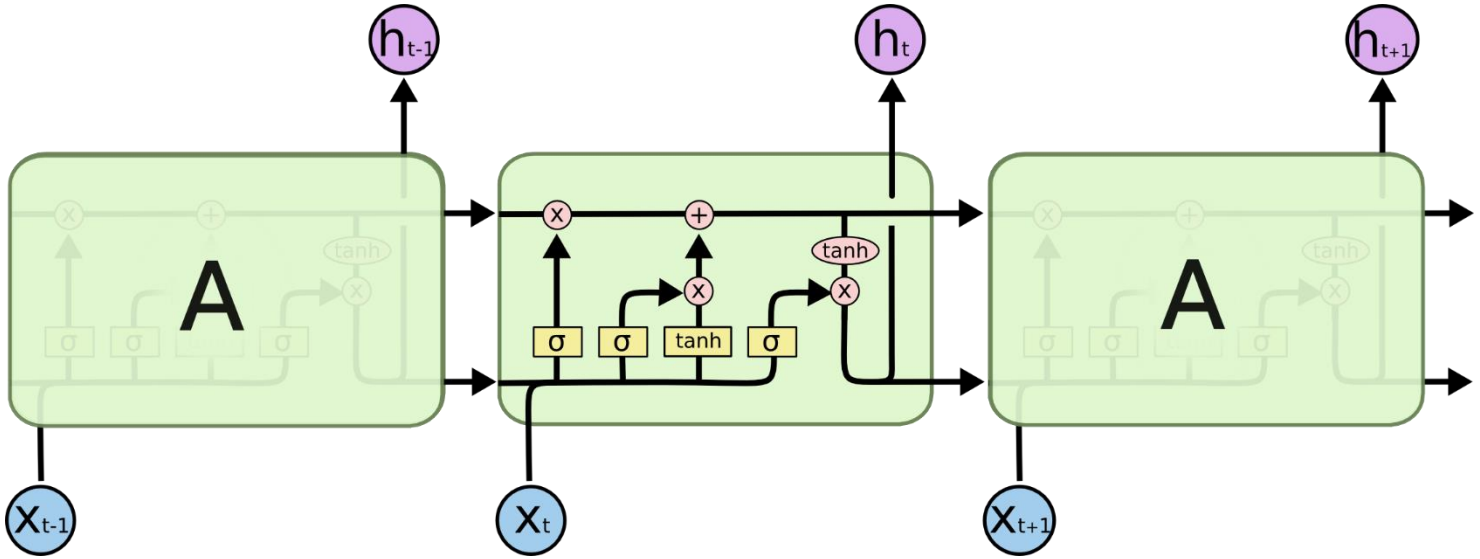
다-대-다는 각 단어에 레이블을 달아주는 시퀀스 레이블링 작업에 사용된다.
시퀀스 레이블링 작업의 대표적인 예로 개체명 인식이 있다.



Sequence Labeling example : Named Entity Recognition

Long Short-Term Memory, LSTM

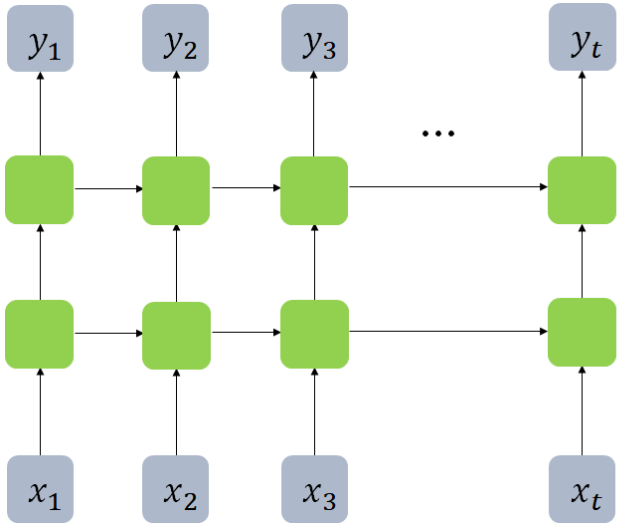
- 기존 RNN(바닐라 RNN)의 장기 의존성 문제를 개선하여 기억력을 높인 RNN의 명칭.
- 앞으로 나오는 설명에서 RNN을 사용한다고 하면 기본적으로 LSTM(또는 GRU)를 사용한다고 가정한다.



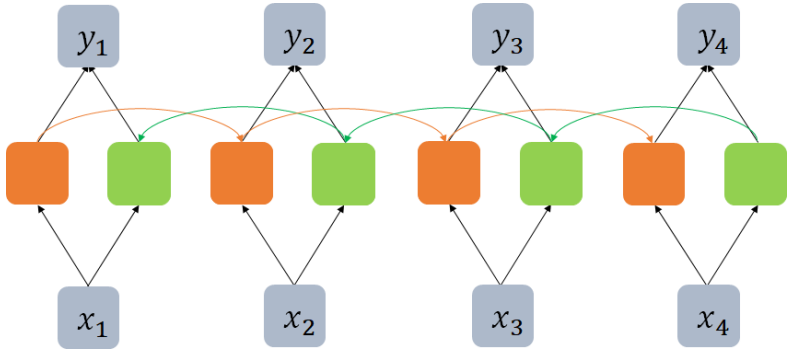
LSTM(Long Short-Term Memory)

Deep Bidirectional Recurrent Neural Networks

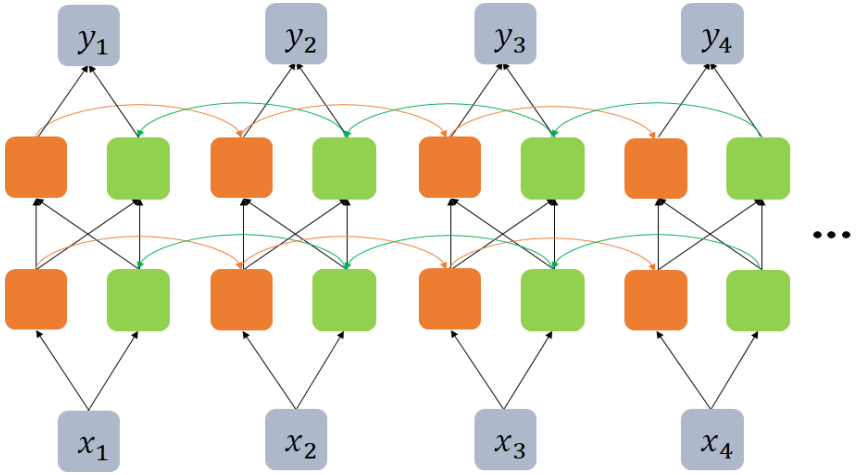
RNN의 은닉층을 높이거나, 역방향으로 입력을 참고하는 RNN을 추가하여 양방향으로 만들 수도 있다.
또는 양방향 RNN의 은닉층을 추가하여 깊은 양방향 RNN을 만들 수도 있다.



Deep RNN



Bidirectional RNN

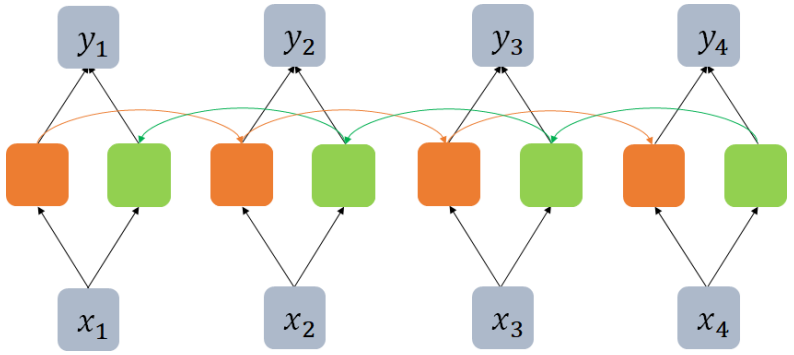


Deep RNN

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

Bidirectional Recurrent Neural Networks

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



Bidirectional LSTM

빈 칸 채우기 문제

Exercise is very effective at [ ] belly fat.

1) Reducing
2) increasing
3) multiplying

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

Named Entity Recognition

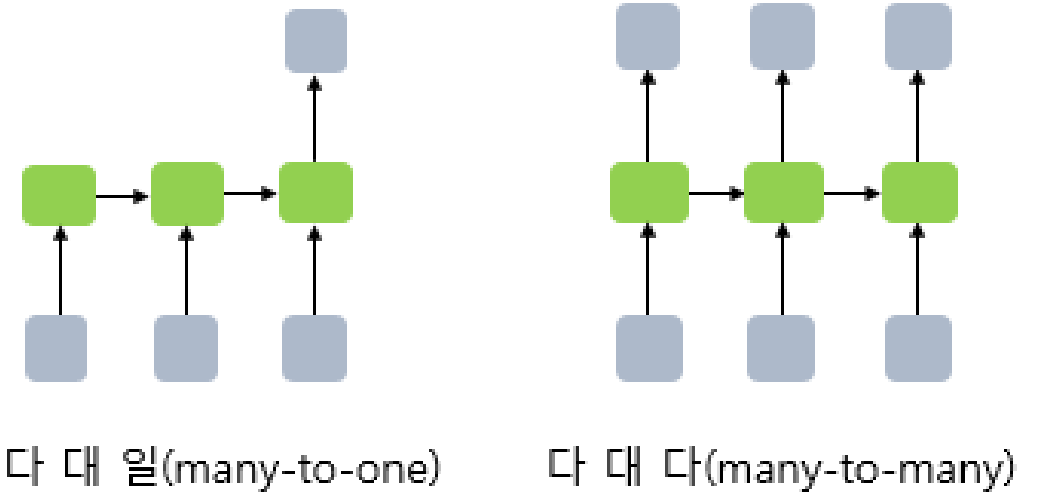
Sequence Labeling

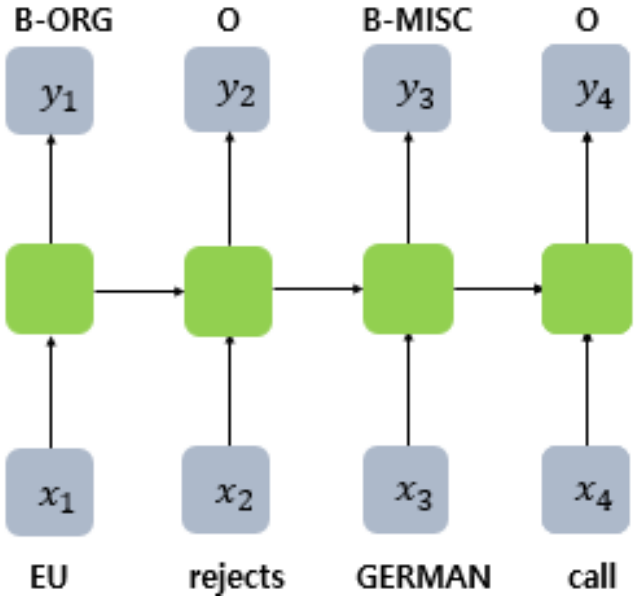
- 시퀀스 레이블링이란 $[x_1, x_2, x_3, ..., x_n]$에 대해서 $[y_1, y_2, y_3, ..., y_n]$을 각각 부여하는 작업을 말한다.
- 개체명 인식은 대표적인 시퀀스 레이블링 태스크에 속한다.

-	X_train	y_train	길이
0	['EU', 'rejects', 'German', 'call', 'to', 'boycott', 'British', 'lamb']	['B-ORG', 'O', 'B-MISC', 'O', 'O', 'O', 'B-MISC', 'O']	8
1	['peter', 'blackburn']	['B-PER', 'I-PER']	2
2	['brussels', '1996-08-22']	['B-LOC', 'O']	2
3	['The', 'European', 'Commission']	['O', 'B-ORG', 'I-ORG']	3

Sequence Labeling using RNN

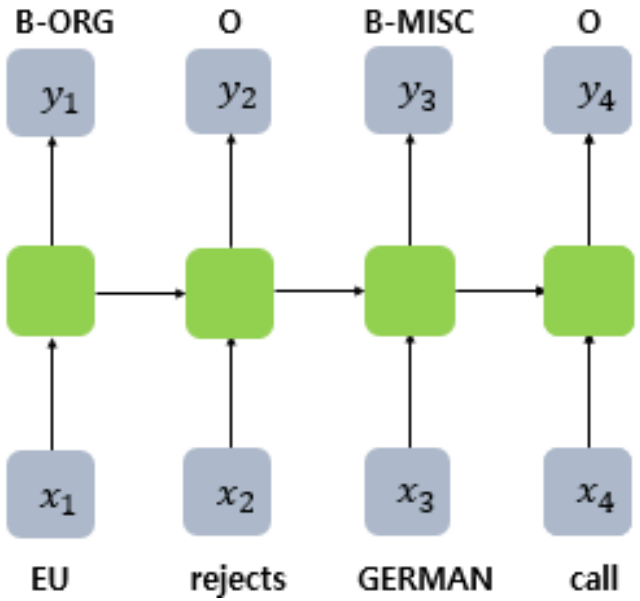
- 시퀀스 레이블링이란 $[x_1, x_2, x_3, ..., x_n]$에 대해서 $[y_1, y_2, y_3, ..., y_n]$을 각각 부여하는 작업을 말한다.
- 개체명 인식은 대표적인 시퀀스 레이블링 태스크에 속한다.
- RNN의 Many-to-Many를 문제를 풀 경우에는 return_sequences=True를 해준다.





Sequence Labeling using RNN

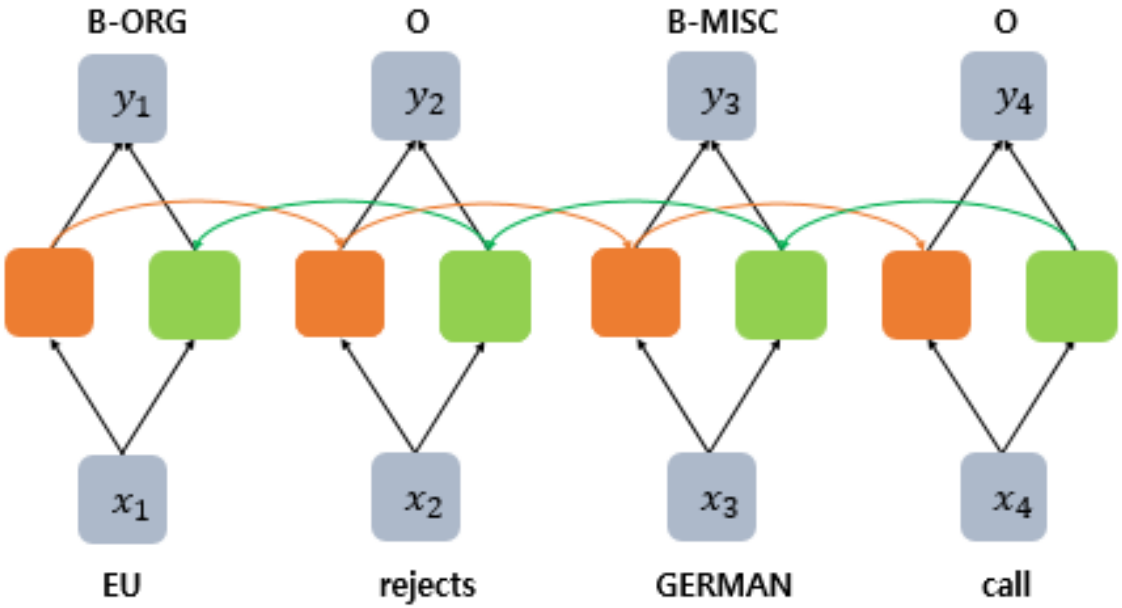
- RNN의 Many-to-Many를 문제를 풀 경우에는 return_sequences=True를 해준다.
- 이 경우 모든 timesteps에 대해서 예측한다.




```
model = Sequential()
model.add(Embedding(vocab_size, 256))
model.add(LSTM(hidden_size, return_sequences=True))
model.add(Dense(classes, activation='softmax'))
```

Sequence Labeling using RNN

- RNN의 Many-to-Many를 문제를 풀 경우에는 return_sequences=True를 해준다.
- 앞, 뒤 문맥을 참고해서 예측이 필요한 시퀀스 레이블링의 경우 양방향 LSTM(BiLSTM)을 사용한다.



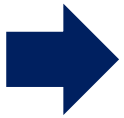

```
model = Sequential()
model.add(Embedding(vocab_size, 256))
model.add(Bidirectional(LSTM(hidden_size, return_sequences=True)))
model.add(Dense(classes, activation='softmax'))
```

Named Entity Recognition

- 개체명 인식은 비정형 텍스트의 개체명 언급을 인명, 단체, 장소, 수치등 정의된 분류로 분류하는 작업.
- 영어 자연어 처리 패키지 NLTK를 이용한 개체명 인식 결과를 예제로 확인해봅시다.

토큰화와 품사 태깅을 수행하는 코드(전처리)

```
from nltk import word_tokenize, pos_tag, ne_chunk

sentence = "James is working at Disney in London"
sentence = pos_tag(word_tokenize(sentence))
print(sentence)
```

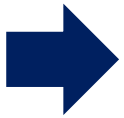


Named Entity Recognition

- 개체명 인식은 비정형 텍스트의 개체명 언급을 인명, 단체, 장소, 수치등 정의된 분류로 분류하는 작업.
- 영어 자연어 처리 패키지 NLTK를 이용한 개체명 인식 결과를 예제로 확인해봅시다.

토큰화와 품사 태깅을 수행하는 코드(전처리)

```
from nltk import word_tokenize, pos_tag, ne_chunk

sentence = "James is working at Disney in London"
sentence = pos_tag(word_tokenize(sentence))
print(sentence)
```



```
[('James', 'NNP'), ('is', 'VBZ'), ('working', 'VBG'), ('at', 'IN'), ('Disney', 'NNP'), ('in', 'IN'),
('London', 'NNP')]
```

Named Entity Recognition

- 개체명 인식은 비정형 텍스트의 개체명 언급을 인명, 단체, 장소, 수치등 정의된 분류로 분류하는 작업.
- 영어 자연어 처리 패키지 NLTK를 이용한 개체명 인식 결과를 예제로 확인해봅시다.

토큰화와 품사 태깅 결과를 입력으로 개체명 인식

```
sentence = ne_chunk(sentence)
print(sentence)
```

Named Entity Recognition

- 개체명 인식은 비정형 텍스트의 개체명 언급을 인명, 단체, 장소, 수치등 정의된 분류로 분류하는 작업.
- 영어 자연어 처리 패키지 NLTK를 이용한 개체명 인식 결과를 예제로 확인해봅시다.

토큰화와 품사 태깅 결과를 입력으로 개체명 인식

```
sentence = ne_chunk(sentence)
print(sentence)
```

```
(S
(PERSON James/NNP)
is/VBZ
working/VBG
at/IN
(ORGANIZATION Disney/NNP)
in/IN
(GPE London/NNP))
```

BIO Tagging (or IOB Tagging)

- 개체명 인식 데이터에서 주로 사용하는 태깅 방법.
- B는 Begin의 약자로 개체명이 시작되는 부분
- I는 Inside의 약자로 개체명의 내부 부분
- O는 Outside의 약자로 개체명이 아닌 부분을 의미

영화에 대한 개체명을 태깅한다고 하면?

BIO Tagging (or IOB Tagging)

- 개체명 인식 데이터에서 주로 사용하는 태깅 방법.
- B는 Begin의 약자로 개체명이 시작되는 부분
- I는 Inside의 약자로 개체명의 내부 부분
- O는 Outside의 약자로 개체명이 아닌 부분을 의미

영화에 대한 개체명을 태깅한다고 하면?

```
해 B
리 I
포 I
터 I
보 O
러 O
가 O
자 O
```

BIO Tagging (or IOB Tagging)

- 개체명 인식 데이터에서 주로 사용하는 태깅 방법.
- B는 Begin의 약자로 개체명이 시작되는 부분
- I는 Inside의 약자로 개체명의 내부 부분
- O는 Outside의 약자로 개체명이 아닌 부분을 의미

영화와 극장에 대한 개체명을 태깅한다고 하면?

```
해 B
리 I
포 I
터 I
보 O
러 O
가 O
자 O
```

BIO Tagging (or IOB Tagging)

- 개체명 인식 데이터에서 주로 사용하는 태깅 방법.
- B는 Begin의 약자로 개체명이 시작되는 부분
- I는 Inside의 약자로 개체명의 내부 부분
- O는 Outside의 약자로 개체명이 아닌 부분을 의미

영화와 극장에 대한 개체명을 태깅한다고 하면?

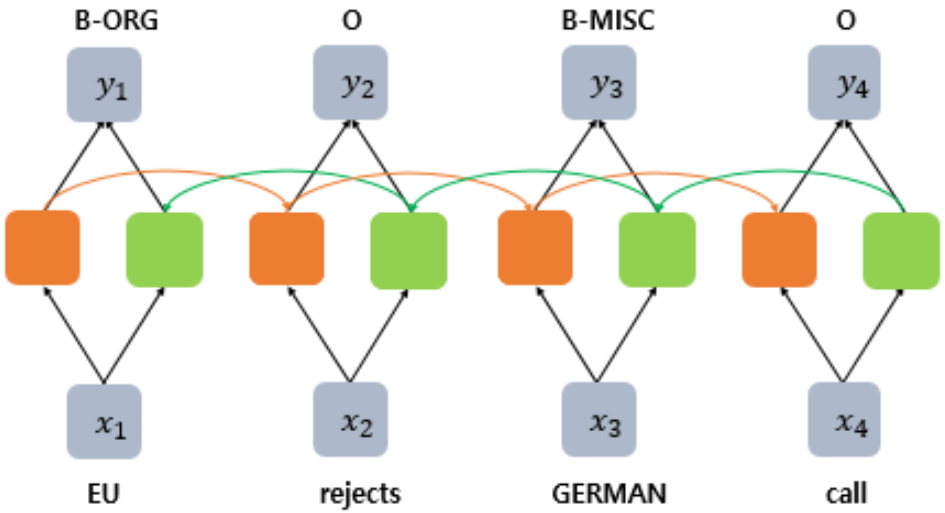
```
해 B
리 I
포 I
터 I
보 O
러 O
가 O
자 O
```

```
해 B-movie
리 I-movie
포 I-movie
터 I-movie
보 O
러 O
메 B-theater
가 I-theater
박 I-theater
스 I-theater
가 O
자 O
```

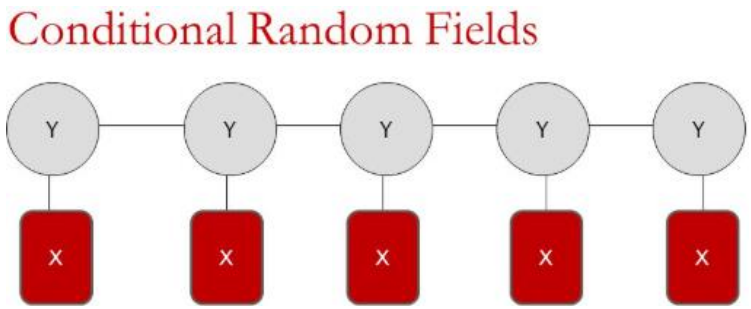
Sequence Labeling Models

- BiLSTM과 CRF 모두 별도로 시퀀스 레이블링을 위해서 사용할 수 있는 독립적인 모델.

Bidirectional LSTM, BiLSTM



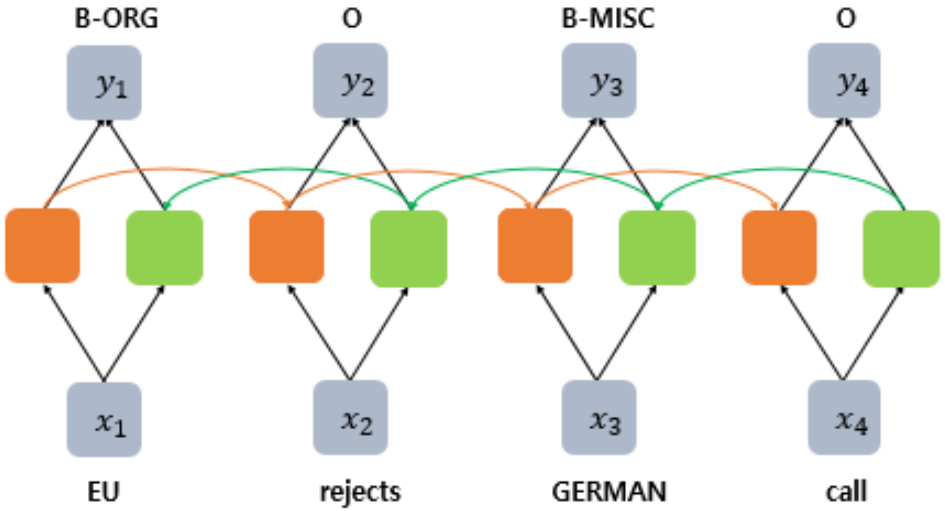
Conditional Random Fields, CRF



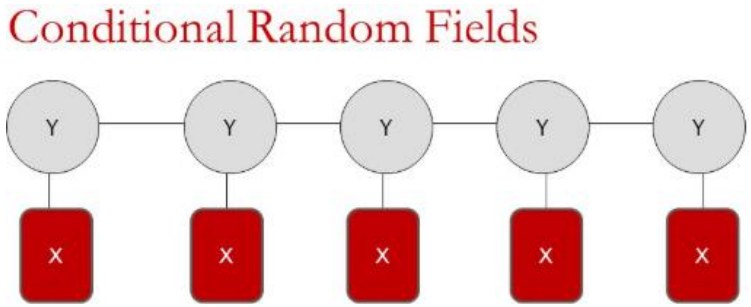
Sequence Labeling Models

- BiLSTM과 CRF 모두 별도로 시퀀스 레이블링을 위해서 사용할 수 있는 독립적인 모델.

Bidirectional LSTM, BiLSTM



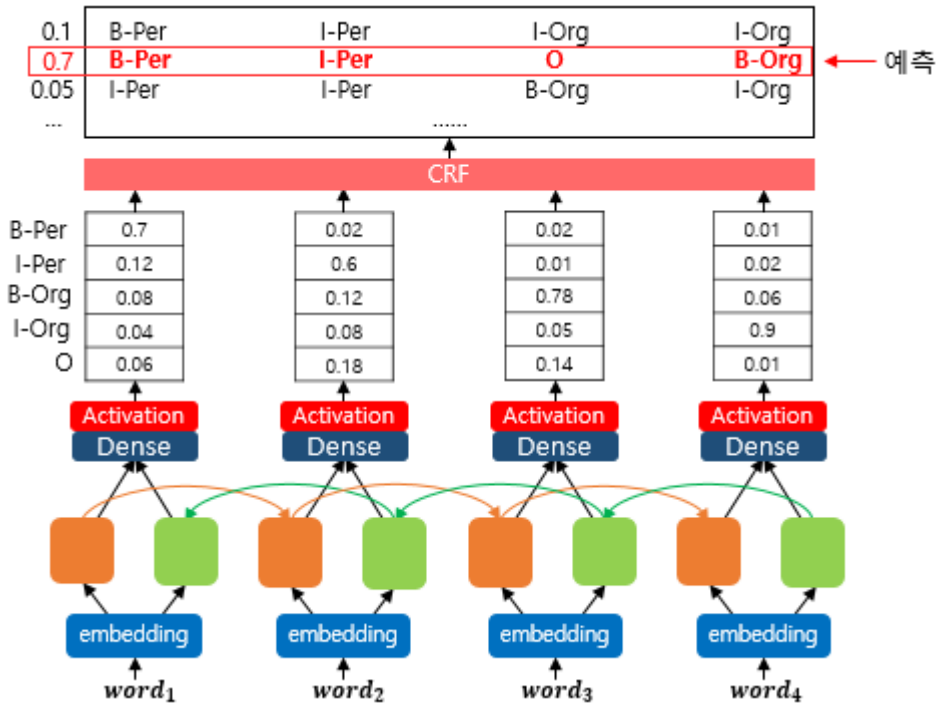
Conditional Random Fields, CRF



이 두 모델을 합치면? BiLSTM-CRF

Bidirectional LSTM-CRF for sequence Labeling

- BiLSTM과 CRF를 사용하여 시퀀스 레이블링을 수행하는 모델을 제안.



Bidirectional LSTM-CRF Models for Sequence Tagging

Zhiheng Huang
Baidu research
huangzhiheng@baidu.com

Wei Xu
Baidu research
xuwei06@baidu.com

Kai Yu
Baidu research
yukai@baidu.com

Abstract

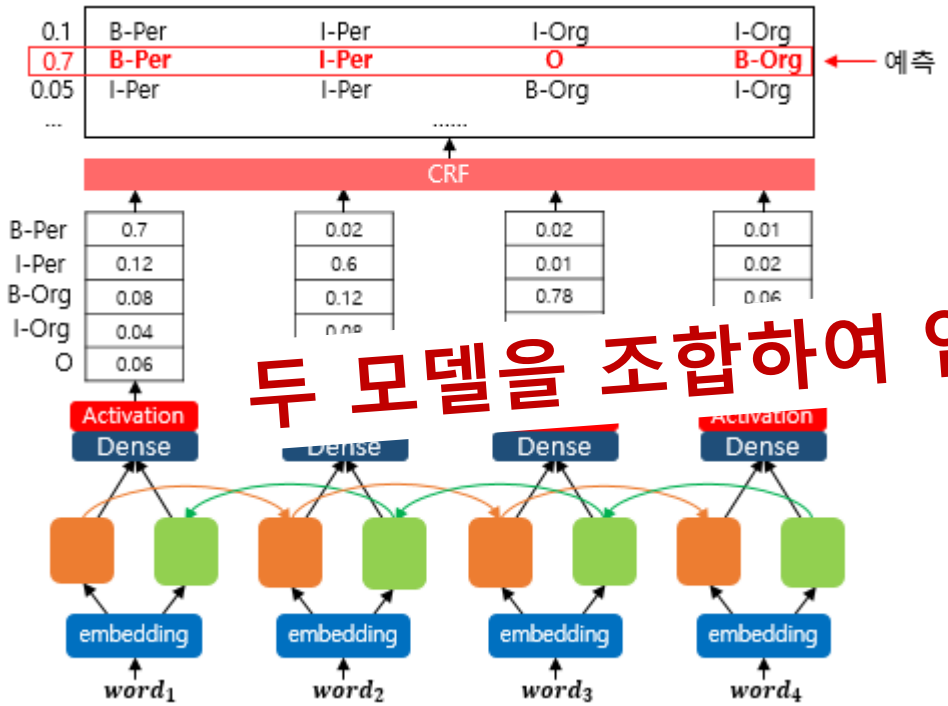
In this paper, we propose a variety of Long Short-Term Memory (LSTM) based models for sequence tagging. These models include LSTM networks, bidirectional LSTM (BI-LSTM) networks, LSTM with a Conditional Random Field (CRF) layer (LSTM-CRF) and bidirectional LSTM with a CRF layer (BI-LSTM-CRF). Our work is the first to apply a bidirectional LSTM CRF (denoted as BI-LSTM-CRF) model to NLP benchmark sequence tagging data sets. We show that the BI-LSTM-CRF model can efficiently use both past and future input features thanks to a bidirectional LSTM component. It can also use sentence level tag information thanks to a CRF layer. The BI-LSTM-CRF model can produce state of the art (or close to) accuracy on POS, chunking and NER data sets. In addition, it is robust and has less dependence on word embedding as compared to previous observations.

(Lafferty et al., 2001). Convolutional network based models (Collobert et al., 2011) have been recently proposed to tackle sequence tagging problem. We denote such a model as *Conv-CRF* as it consists of a convolutional network and a CRF layer on the output (the term of *sentence level log-likelihood (SSL)* was used in the original paper). The Conv-CRF model has generated promising results on sequence tagging tasks. In speech language understanding community, recurrent neural network (Mesnil et al., 2013; Yao et al., 2014) and convolutional nets (Xu and Sarikaya, 2013) based models have been recently proposed. Other relevant work includes (Graves et al., 2005; Graves et al., 2013) which proposed a bidirectional recurrent neural network for speech recognition.

In this paper, we propose a variety of neural network based models to sequence tagging task. These models include LSTM networks, bidirectional LSTM networks (BI-LSTM), LSTM networks with a CRF layer (LSTM-CRF), and bidirectional LSTM networks with a CRF layer (BI-LSTM-CRF). Our contributions can be summarized as follows. 1) We systematically com-

Bidirectional LSTM-CRF for sequence Labeling

- BiLSTM과 CRF를 사용하여 시퀀스 레이블링을 수행하는 모델을 제안.



Bidirectional LSTM-CRF Models for Sequence Tagging

Zhiheng Huang
Baidu research
huangzhiheng@baidu.com

Wei Xu
Baidu research
xuwei06@baidu.com

Kai Yu
Baidu research
yukai@baidu.com

Abstract

In this paper, we propose a variety of Long Short-Term Memory (LSTM) based models for sequence tagging. These models include LSTM networks, bidirectional LSTM networks (BI-LSTM), LSTM networks with a CRF layer (LSTM-CRF), and bidirectional LSTM networks with a CRF layer (BI-LSTM-CRF). Our contributions can be summarized as follows. 1) We systematically compare the performance of these models on POS, chunking and NER data sets. 2) Our work is the first to apply a bidirectional LSTM CRF (denoted as BI-LSTM-CRF) model to NLP benchmark sequence tagging data sets. We show that the BI-LSTM-CRF model can efficiently use both past and future input features thanks to a bidirectional LSTM component. It can also use sentence level tag information thanks to a CRF layer. The BI-LSTM-CRF model can produce state of the art (or close to) accuracy on POS, chunking and NER data sets. In addition, it is robust and has less dependence on word embedding as compared to previous observations.

(Lafferty et al., 2001). Convolutional network based models (Collobert et al., 2011) have been recently proposed to tackle sequence tagging problem. We denote such a model as *Conv-CRF* as it consists of

tagging tasks. In speech language understanding community, recurrent neural network (Mesnil et al., 2013; Yao et al., 2014) and convolutional nets (Xu and Sarikaya, 2013) based models have been recently proposed. Other relevant work includes (Graves et al., 2005; Graves et al., 2013) which proposed a bidirectional recurrent neural network for speech recognition.

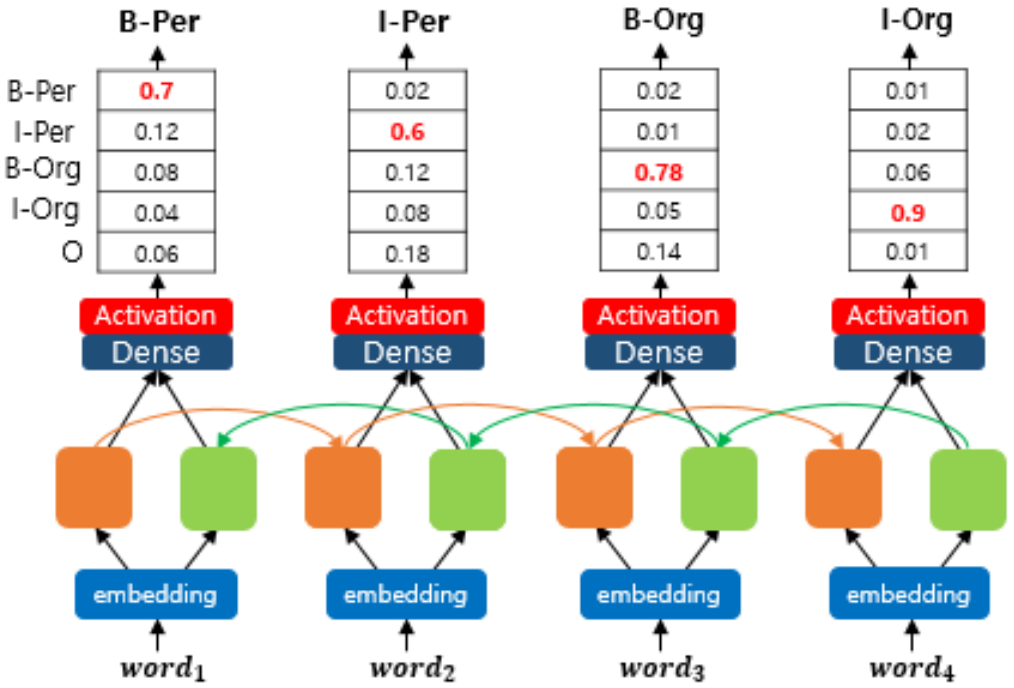
In this paper, we propose a variety of neural network based models to sequence tagging task. These models include LSTM networks, bidirectional LSTM networks (BI-LSTM), LSTM networks with a CRF layer (LSTM-CRF), and bidirectional LSTM networks with a CRF layer (BI-LSTM-CRF). Our contributions can be summarized as follows. 1) We systematically com-

두 모델을 조합하여 얻을 수 있는 이점은 무엇일까요?

Bidirectional LSTM-CRF for sequence Labeling

사람(Person), 조직(Organization) 두 가지를 태깅하는 모델이라고 가정해보자.
이 경우 필요한 태깅은 다음과 같다.

B-Per, I-Per, B-Org, I-Org, O

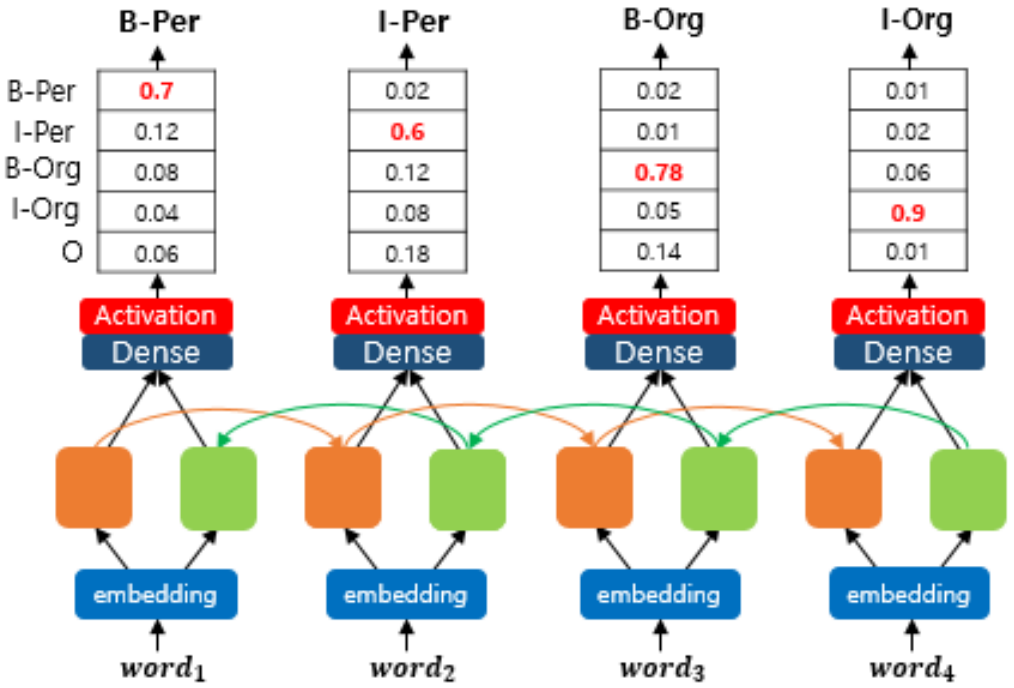


- 일반적인 BiLSTM 모델은 각 단어를 벡터로 입력받고, 모델의 출력층에서 활성화 함수를 통해 개체명을 예측한다.

Bidirectional LSTM-CRF for sequence Labeling

사람(Person), 조직(Organization) 두 가지를 태깅하는 모델이라고 가정해보자.
이 경우 필요한 태깅은 다음과 같다.

B-Per, I-Per, B-Org, I-Org, O



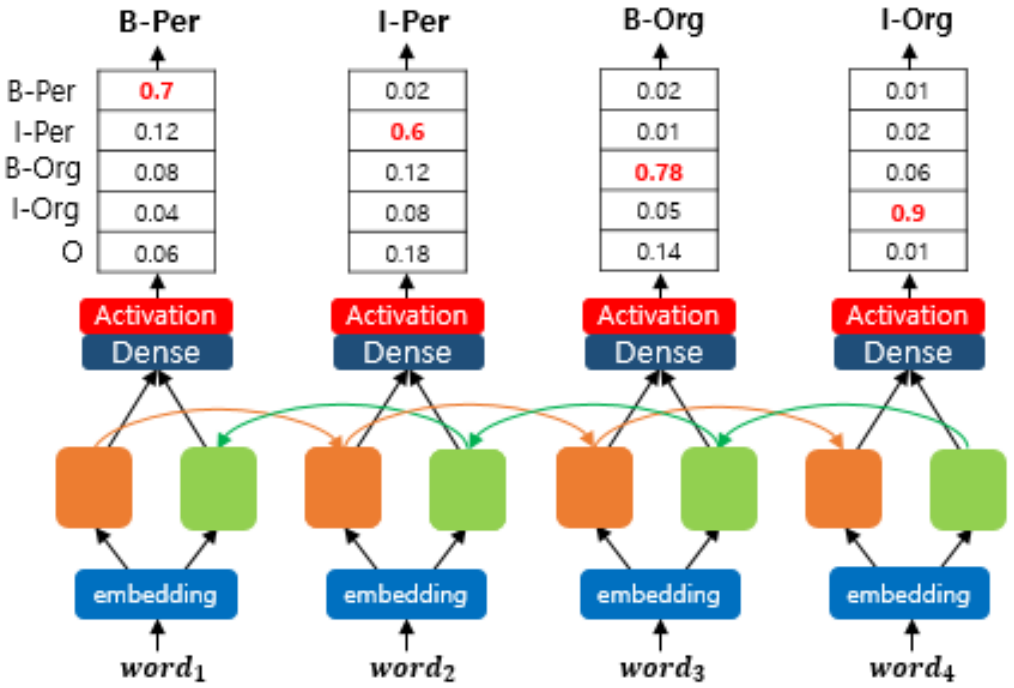
- 일반적인 BiLSTM 모델은 각 단어를 벡터로 입력받고, 모델의 출력층에서 활성화 함수를 통해 개체명을 예측한다.

현재 이 BiLSTM은 제대로 예측하고 있나요?

Bidirectional LSTM-CRF for sequence Labeling

사람(Person), 조직(Organization) 두 가지를 태깅하는 모델이라고 가정해보자.
이 경우 필요한 태깅은 다음과 같다.

B-Per, I-Per, B-Org, I-Org, O



- 일반적인 BiLSTM 모델은 각 단어를 벡터로 입력받고, 모델의 출력층에서 활성화 함수를 통해 개체명을 예측한다.

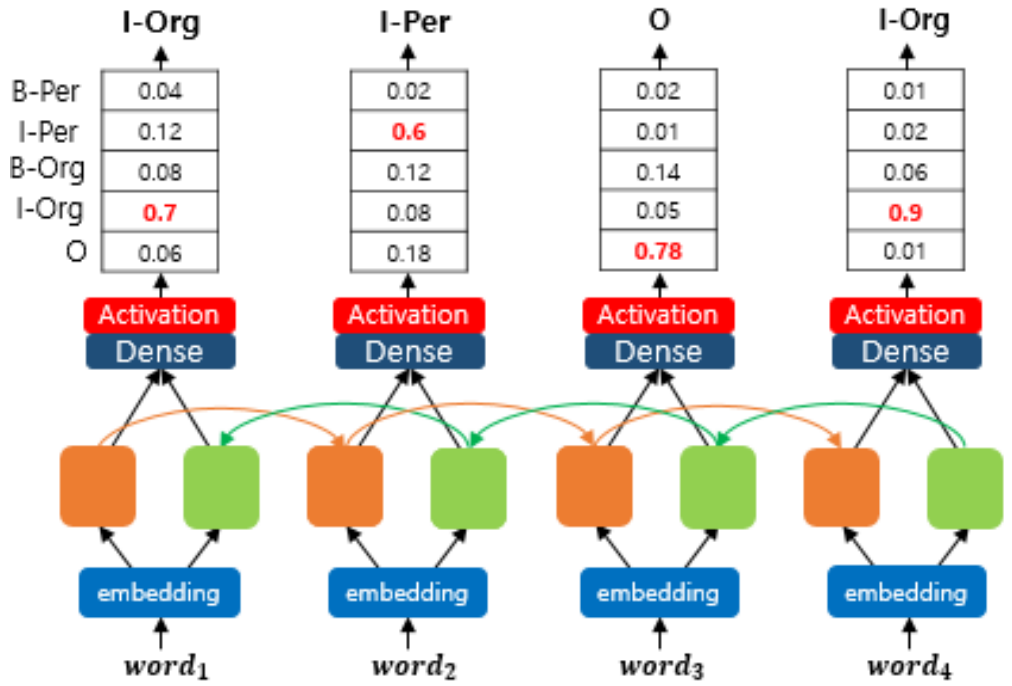
현재 이 BiLSTM은 제대로 예측하고 있나요?

입력 단어가 어떤 단어인지를 보여주지 않고있으므로 알 수 없음.

Bidirectional LSTM-CRF for sequence Labeling

사람(Person), 조직(Organization) 두 가지를 태깅하는 모델이라고 가정해보자.
이 경우 필요한 태깅은 다음과 같다.

B-Per, I-Per, B-Org, I-Org, O



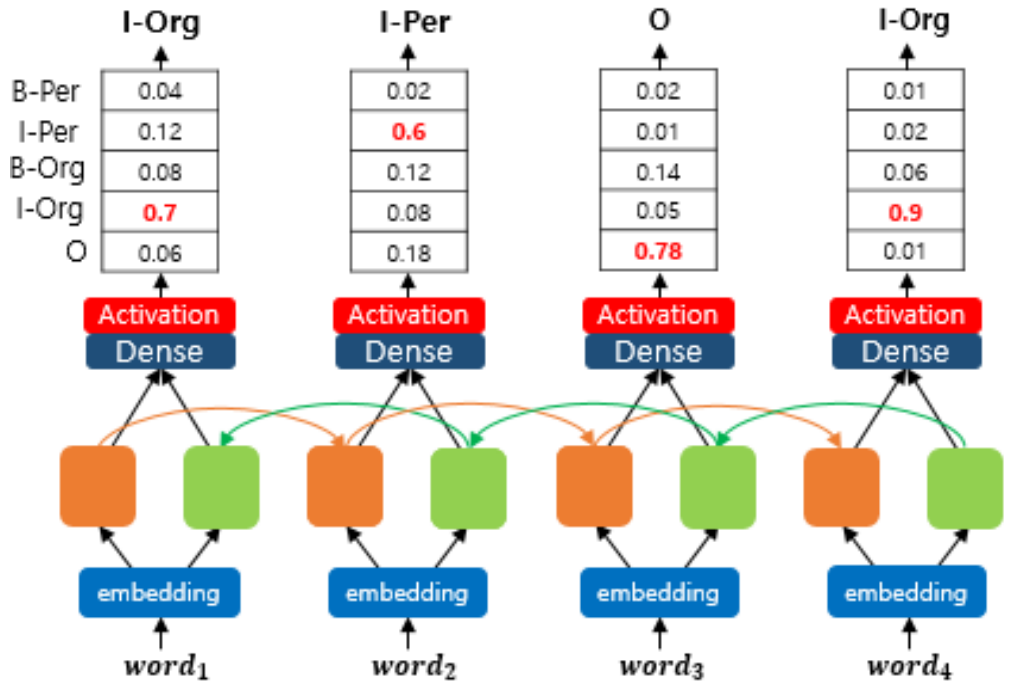
- 일반적인 BiLSTM 모델은 각 단어를 벡터로 입력받고, 모델의 출력층에서 활성화 함수를 통해 개체명을 예측한다.

현재 이 BiLSTM은 제대로 예측하고 있나요?

Bidirectional LSTM-CRF for sequence Labeling

사람(Person), 조직(Organization) 두 가지를 태깅하는 모델이라고 가정해보자.
이 경우 필요한 태깅은 다음과 같다.

B-Per, I-Per, B-Org, I-Org, O



- 일반적인 BiLSTM 모델은 각 단어를 벡터로 입력받고, 모델의 출력층에서 활성화 함수를 통해 개체명을 예측한다.

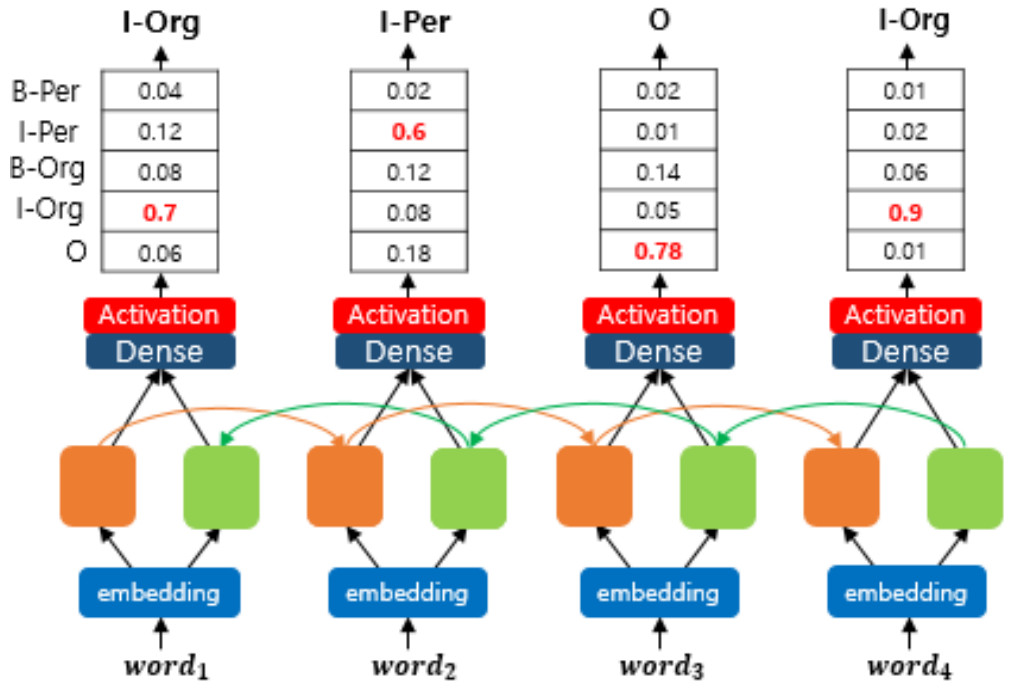
현재 이 BiLSTM은 제대로 예측하고 있나요?

입력 단어와 상관없이
잘못된 예측을 하고 있음.

Bidirectional LSTM-CRF for sequence Labeling

사람(Person), 조직(Organization) 두 가지를 태깅하는 모델이라고 가정해보자.
이 경우 필요한 태깅은 다음과 같다.

B-Per, I-Per, B-Org, I-Org, O



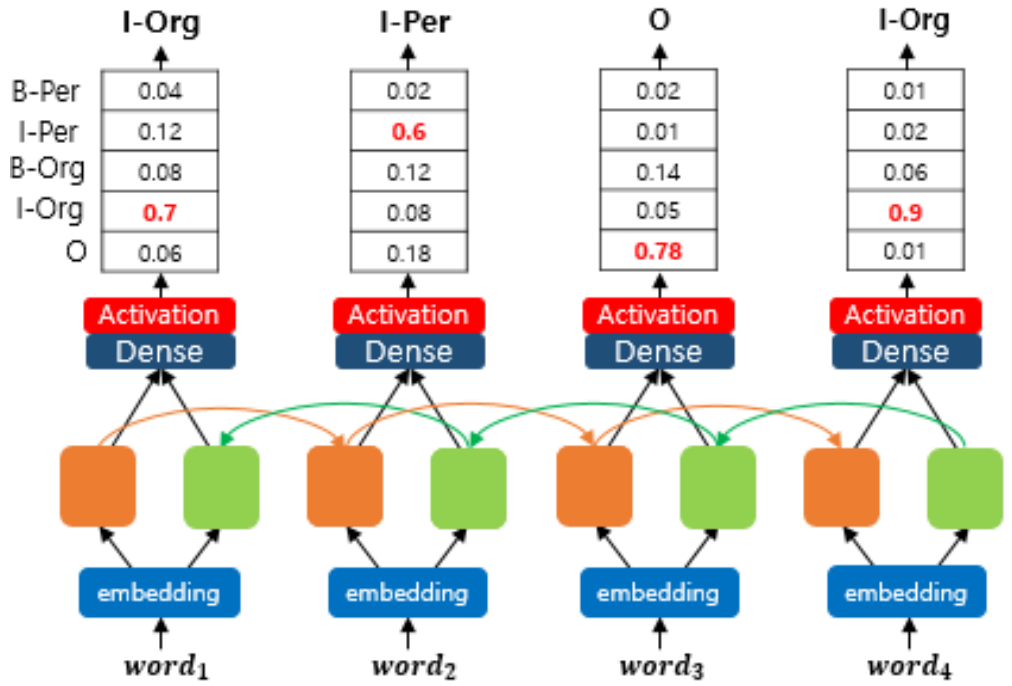
아래의 모든 BIO 태깅 규칙을 위반

- 첫번째 단어의 레이블에서 I가 등장할 수 없습니다.
- I-??은 반드시 B-?? 뒤에서 등장해야 합니다.
  (Ex) I-Per과 I-Org 둘 다 위반.)
- O 뒤에는 I가 등장할 수 없습니다.

Bidirectional LSTM-CRF for sequence Labeling

사람(Person), 조직(Organization) 두 가지를 태깅하는 모델이라고 가정해보자.
이 경우 필요한 태깅은 다음과 같다.

B-Per, I-Per, B-Org, I-Org, O



아래의 모든 BIO 태깅 규칙을 위반

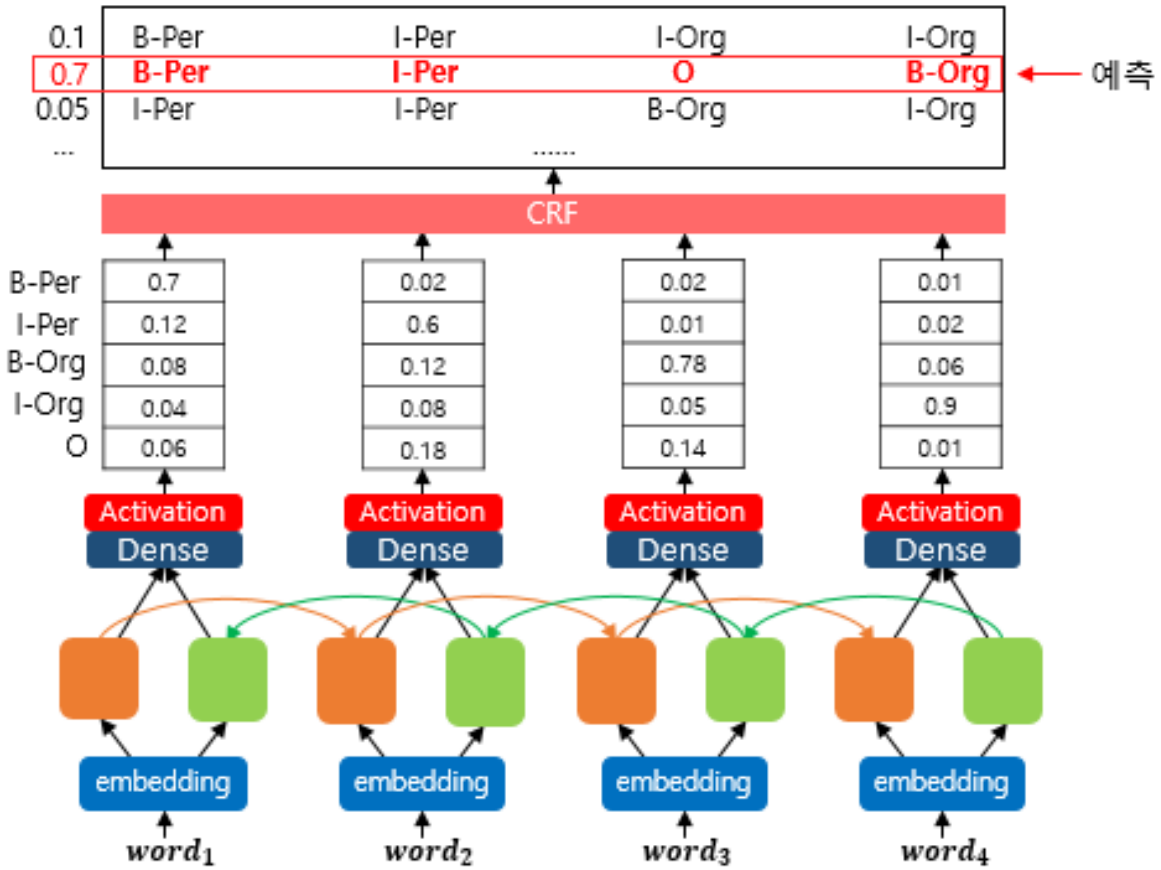
- 첫번째 단어의 레이블에서 I가 등장할 수 없습니다.
- I-??은 반드시 B-?? 뒤에서 등장해야 합니다.
  (Ex) I-Per과 I-Org 둘 다 위반.)
- O 뒤에는 I가 등장할 수 없습니다.

이 규칙 자체를 학습할 수 있도록 해보자!

Bidirectional LSTM-CRF for sequence Labeling

- BiLSTM-CRF 모델은 BILSTM 위에 CRF 층을 추가한다.
- CRF 층은 학습 과정에서 태깅 시퀀스의 가장 적절한 조합을 찾도록 훈련된다.

B-Per, I-Per, B-Org, I-Org, O



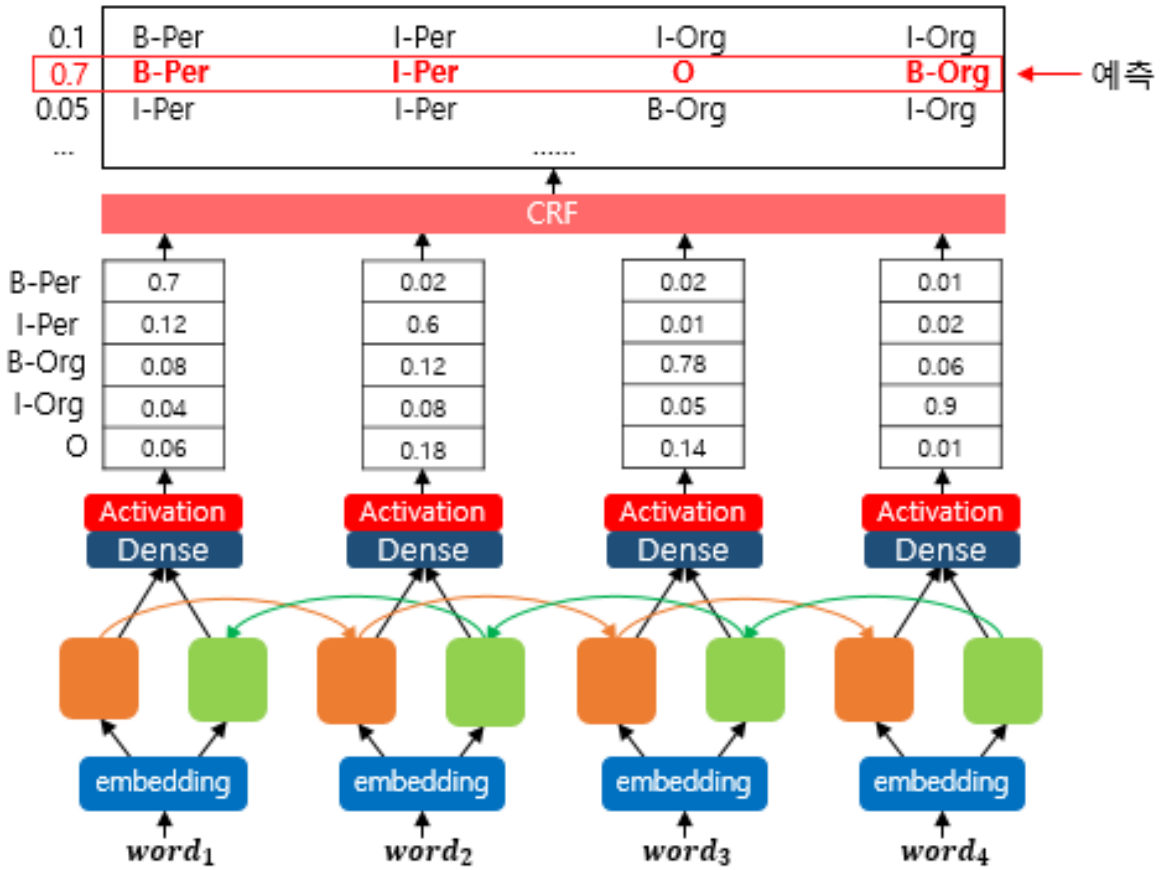
CRF 층은 학습 과정에서 아래의 규칙들을 학습한다.

- 첫번째 단어의 레이블에서 I가 등장할 수 없습니다.
- 개체명은 반드시 B로 시작하고, I에서 끝납니다.
- O 뒤에는 I가 등장할 수 없습니다.

Bidirectional LSTM-CRF for sequence Labeling

- BiLSTM-CRF 모델은 BILSTM 위에 CRF 층을 추가한다.
- CRF 층은 학습 과정에서 태깅 시퀀스의 가장 적절한 조합을 찾도록 훈련된다.

B-Per, I-Per, B-Org, I-Org, O



CRF 층은 학습 과정에서 아래의 규칙들을 학습한다.

- 첫번째 단어의 레이블에서 I가 등장할 수 없습니다.
- 개체명은 반드시 B로 시작하고, I에서 끝납니다.
- O 뒤에는 I가 등장할 수 없습니다.

BiLSTM에서 발생가능한 규칙 위반을 보정하는 효과

Bidirectional LSTM-CNN for sequence Labeling

- BiLSTM과 CNN을 조합하여 탄생한 모델.

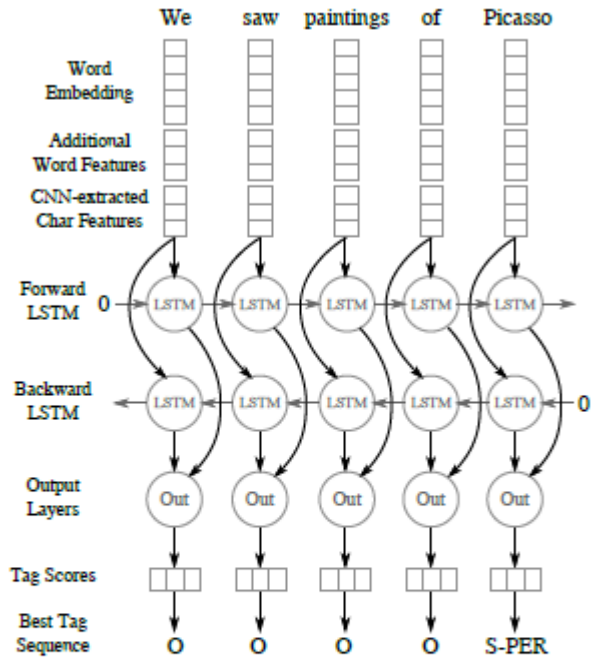


Figure 1: The (unrolled) BLSTM for tagging named entities. Multiple tables look up word-level feature vectors. The CNN (Figure 2) extracts a fixed length feature vector from character-level features. For each word, these vectors are concatenated and fed to the BLSTM network and then to the output layers (Figure 3).

Bidirectional LSTM-CNN for sequence Labeling

- BiLSTM과 CNN을 조합하여 탄생한 모델.

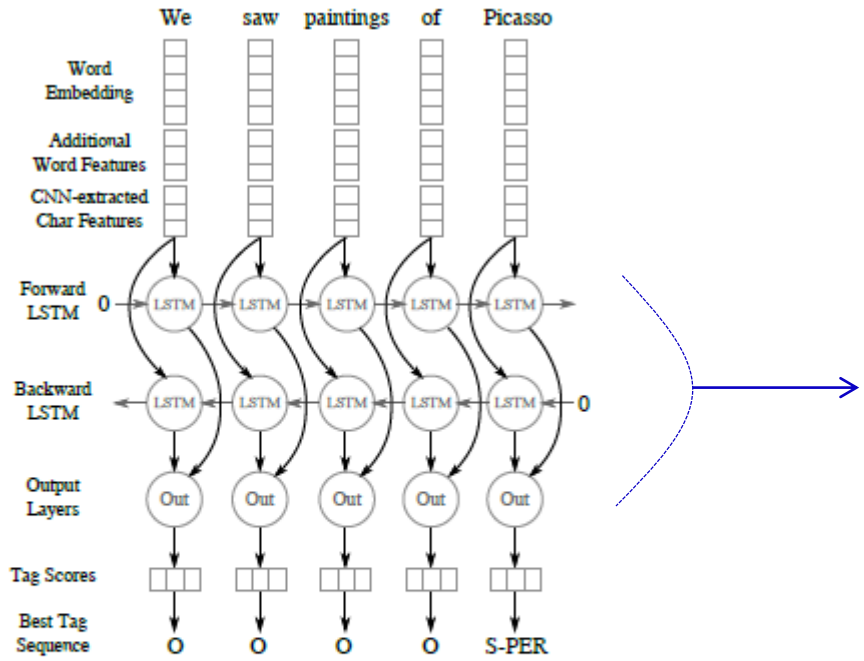


Figure 1: The (unrolled) BLSTM for tagging named entities. Multiple tables look up word-level feature vectors. The CNN (Figure 2) extracts a fixed length feature vector from character-level features. For each word, these vectors are concatenated and fed to the BLSTM network and then to the output layers (Figure 3).

Bidirectional LSTM-CNN for sequence Labeling

- BiLSTM과 CNN을 조합하여 탄생한 모델.

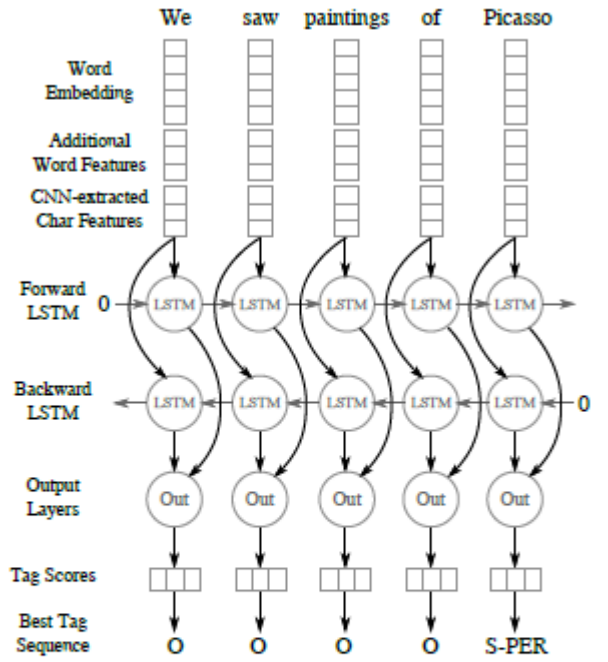
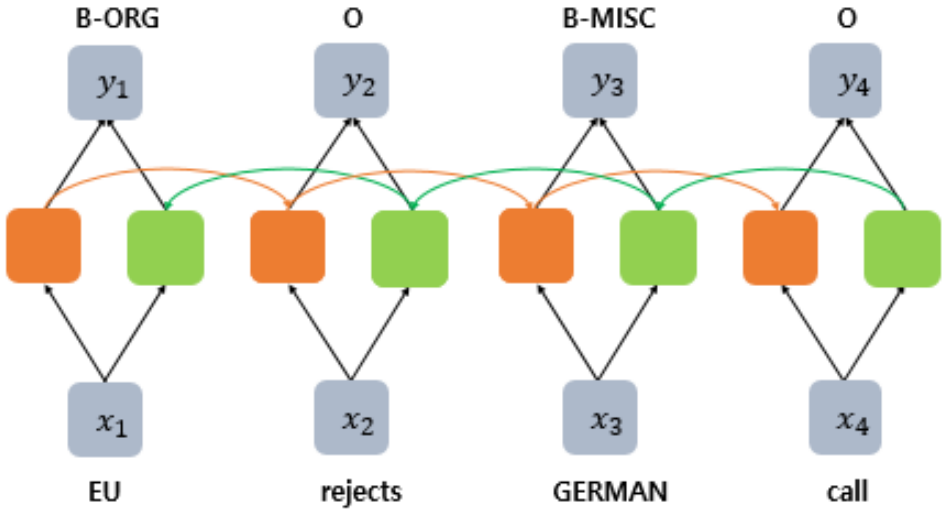


Figure 1: The (unrolled) BLSTM for tagging named entities. Multiple tables look up word-level feature vectors. The CNN (Figure 2) extracts a fixed length feature vector from character-level features. For each word, these vectors are concatenated and fed to the BLSTM network and then to the output layers (Figure 3).

Bidirectional LSTM, BiLSTM



Bidirectional LSTM-CNN for sequence Labeling

- BiLSTM과 CNN을 조합하여 탄생한 모델.

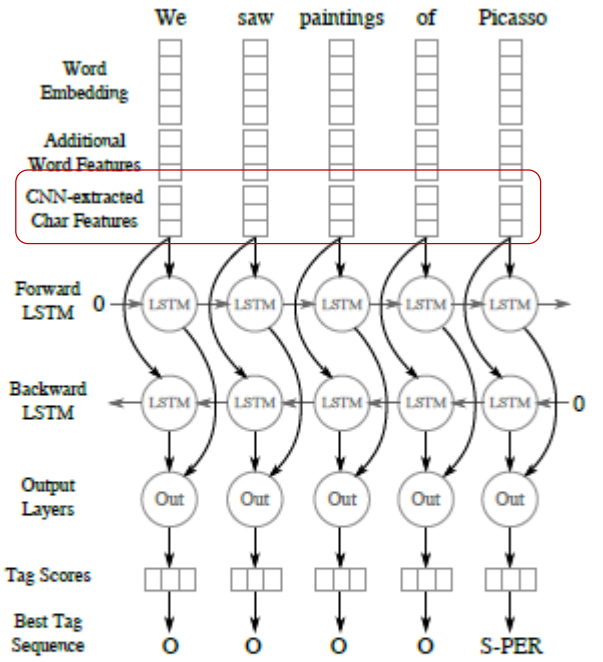
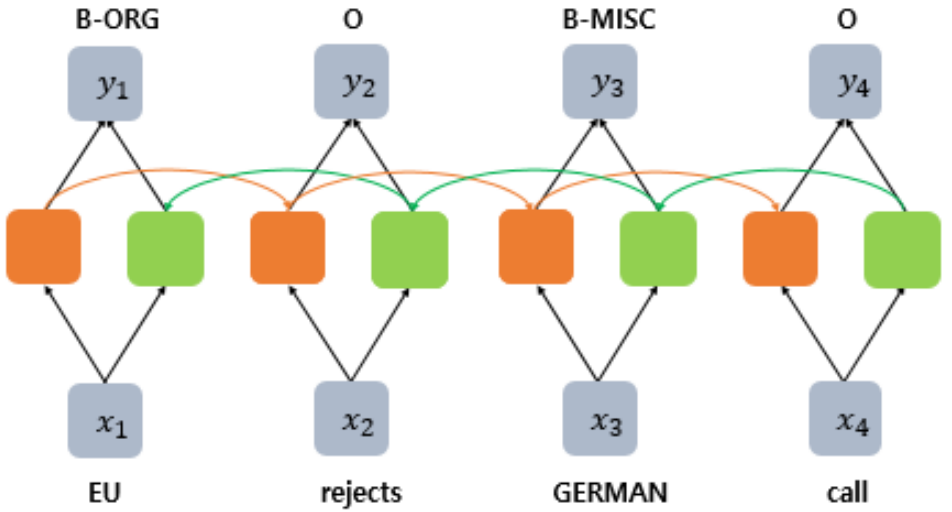


Figure 1: The (unrolled) BLSTM for tagging named entities. Multiple tables look up word-level feature vectors. The CNN (Figure 2) extracts a fixed length feature vector from character-level features. For each word, these vectors are concatenated and fed to the BLSTM network and then to the output layers (Figure 3).

Bidirectional LSTM, BiLSTM



Bidirectional LSTM-CNN for sequence Labeling

- BiLSTM과 CNN을 조합하여 탄생한 모델.

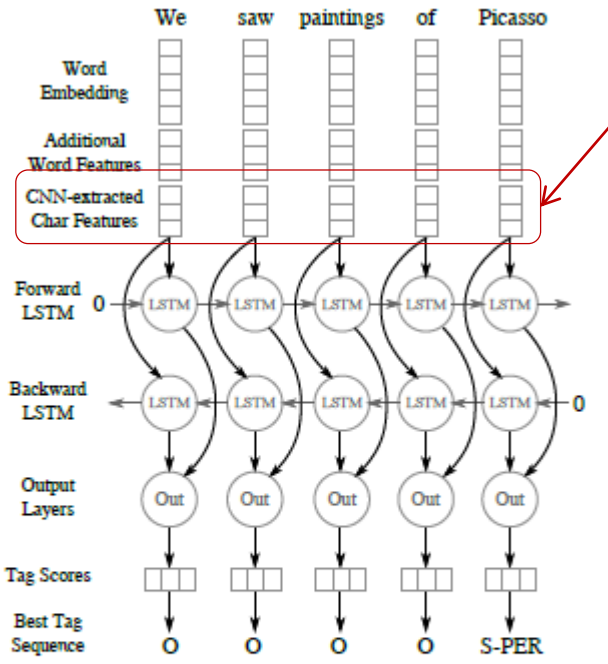
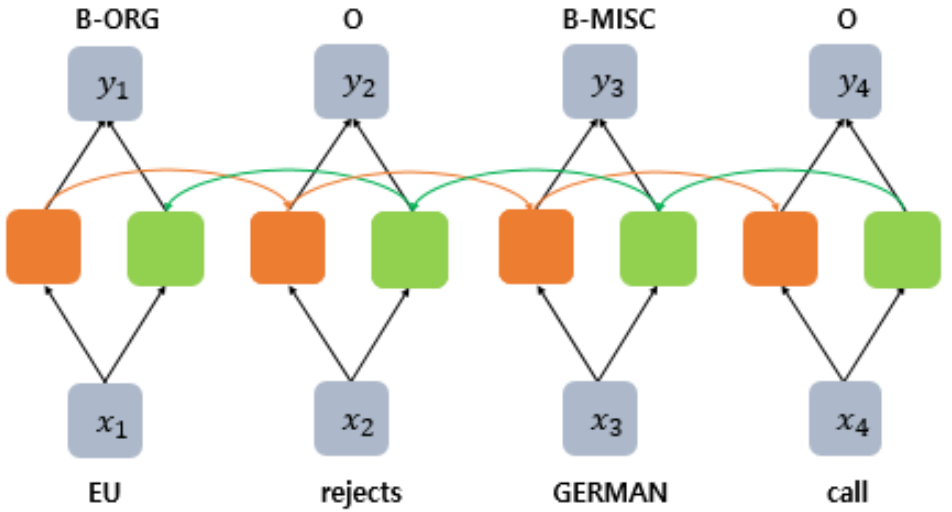


Figure 1: The (unrolled) BLSTM for tagging named entities. Multiple tables look up word-level feature vectors. The CNN (Figure 2) extracts a fixed length feature vector from character-level features. For each word, these vectors are concatenated and fed to the BLSTM network and then to the output layers (Figure 3).

입력에 CNN-extracted Char Features를 추가

Bidirectional LSTM, BiLSTM



Bidirectional LSTM-CNN for sequence Labeling

- BiLSTM과 CNN을 조합하여 탄생한 모델.

입력에 CNN-extracted Char Features를 추가

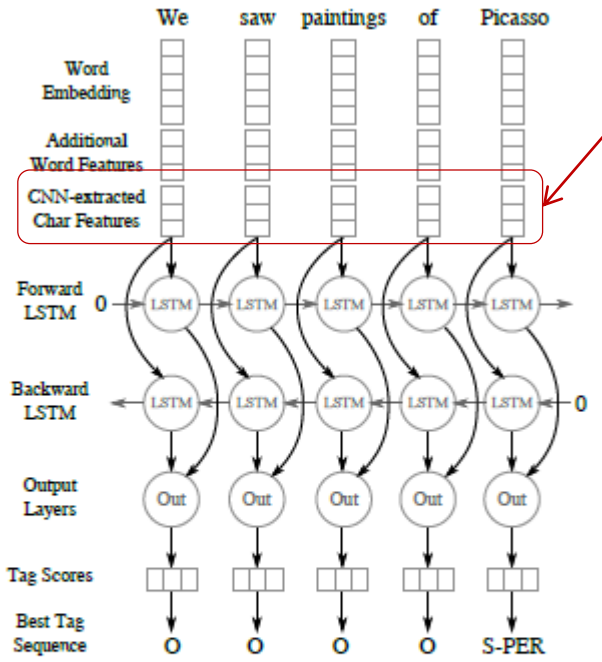


Figure 1: The (unrolled) BLSTM for tagging named entities. Multiple tables look up word-level feature vectors. The CNN (Figure 2) extracts a fixed length feature vector from character-level features. For each word, these vectors are concatenated and fed to the BLSTM network and then to the output layers (Figure 3).

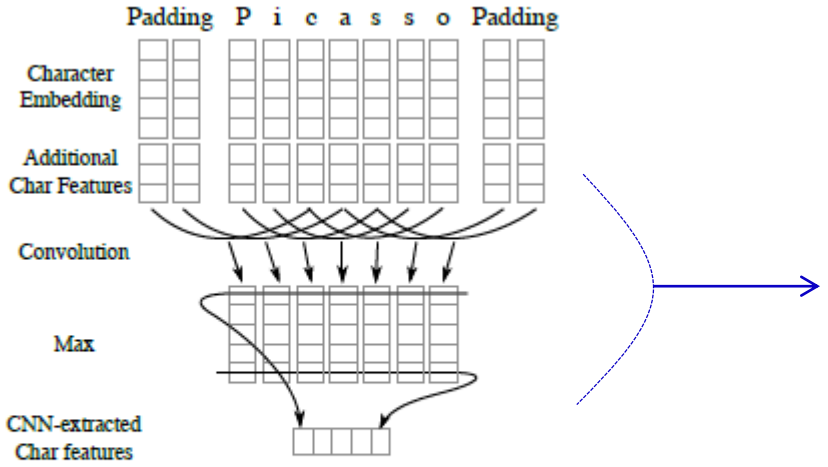


Figure 2: The convolutional neural network extracts character features from each word. The character embedding and (optionally) the character type feature vector are computed through lookup tables. Then, they are concatenated and passed into the CNN.

Bidirectional LSTM-CNN for sequence Labeling

- BiLSTM과 CNN을 조합하여 탄생한 모델.

입력에 CNN-extracted Char Features를 추가

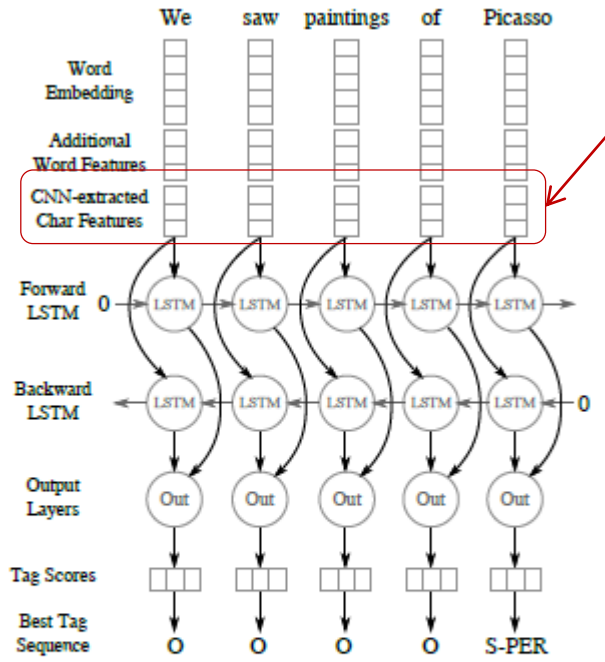


Figure 1: The (unrolled) BLSTM for tagging named entities. Multiple tables look up word-level feature vectors. The CNN (Figure 2) extracts a fixed length feature vector from character-level features. For each word, these vectors are concatenated and fed to the BLSTM network and then to the output layers (Figure 3).

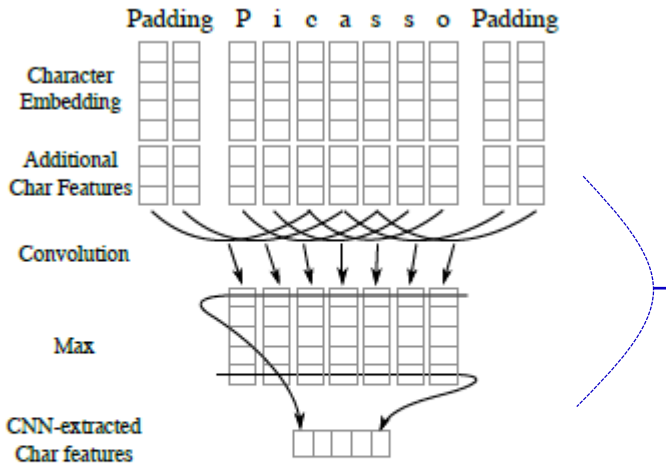
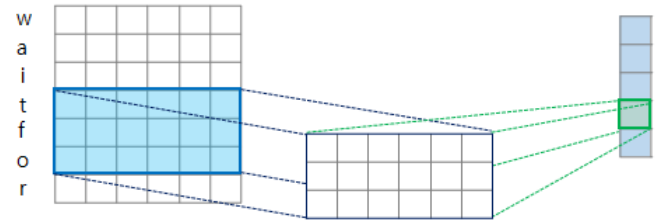


Figure 2: The convolutional neural network extracts character features from each word. The character embedding and (optionally) the character type feature vector are computed through lookup tables. Then, they are concatenated and passed into the CNN.

Char CNN



Bidirectional LSTM-CNN for sequence Labeling

- BiLSTM과 CNN을 조합하여 탄생한 모델.

입력에 CNN-extracted Char Features를 추가

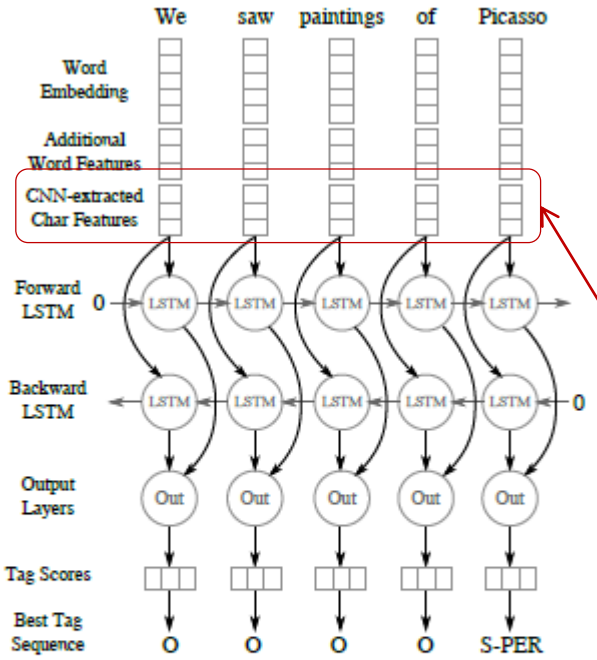


Figure 1: The (unrolled) BLSTM for tagging named entities. Multiple tables look up word-level feature vectors. The CNN (Figure 2) extracts a fixed length feature vector from character-level features. For each word, these vectors are concatenated and fed to the BLSTM network and then to the output layers (Figure 3).

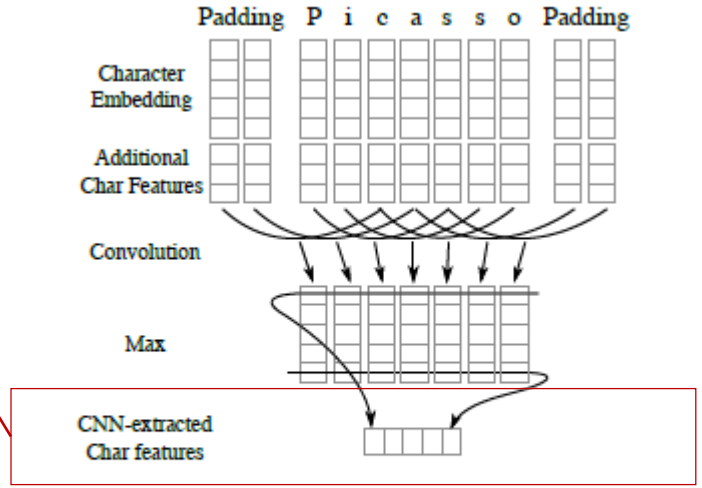


Figure 2: The convolutional neural network extracts character features from each word. The character embedding and (optionally) the character type feature vector are computed through lookup tables. Then, they are concatenated and passed into the CNN.

Bidirectional LSTM-CNN for sequence Labeling

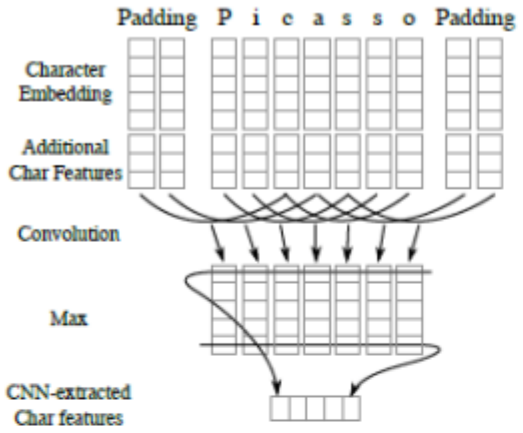


Figure 2: The convolutional neural network extracts character features from each word. The character embedding and (optionally) the character type feature vector are computed through lookup tables. Then, they are concatenated and passed into the CNN.

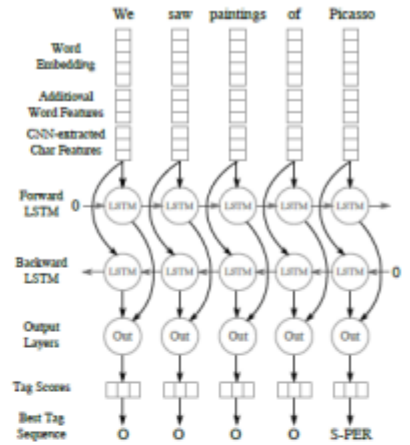


Figure 1: The (unrolled) BLSTM for tagging named entities. Multiple tables look up word-level feature vectors. The CNN (Figure 2) extracts a fixed length feature vector from character-level features. For each word, these vectors are concatenated and fed to the BLSTM network and then to the output layers (Figure 3).

Sequence Labeling

지금까지 BiLSTM + CRF와 BiLSTM + CNN 모델 두 가지를 배웠습니다.
여기서 추가적으로 또 어떤 방법들을 시도해볼 수 있을까요?

Sequence Labeling

지금까지 BiLSTM + CRF와 BiLSTM + CNN 모델 두 가지를 배웠습니다.
여기서 추가적으로 또 어떤 방법들을 시도해볼 수 있을까요?

- **Pre-trained Embedding**

  Ex) Word2Vec, FastText, GloVe

Sequence Labeling

지금까지 BiLSTM + CRF와 BiLSTM + CNN 모델 두 가지를 배웠습니다.
여기서 추가적으로 또 어떤 방법들을 시도해볼 수 있을까요?

- **Pre-trained Embedding**

  Ex) Word2Vec, FastText, GloVe

- **Contextual Embedding**

  Ex) ELMo, Context2Vec

Sequence Labeling

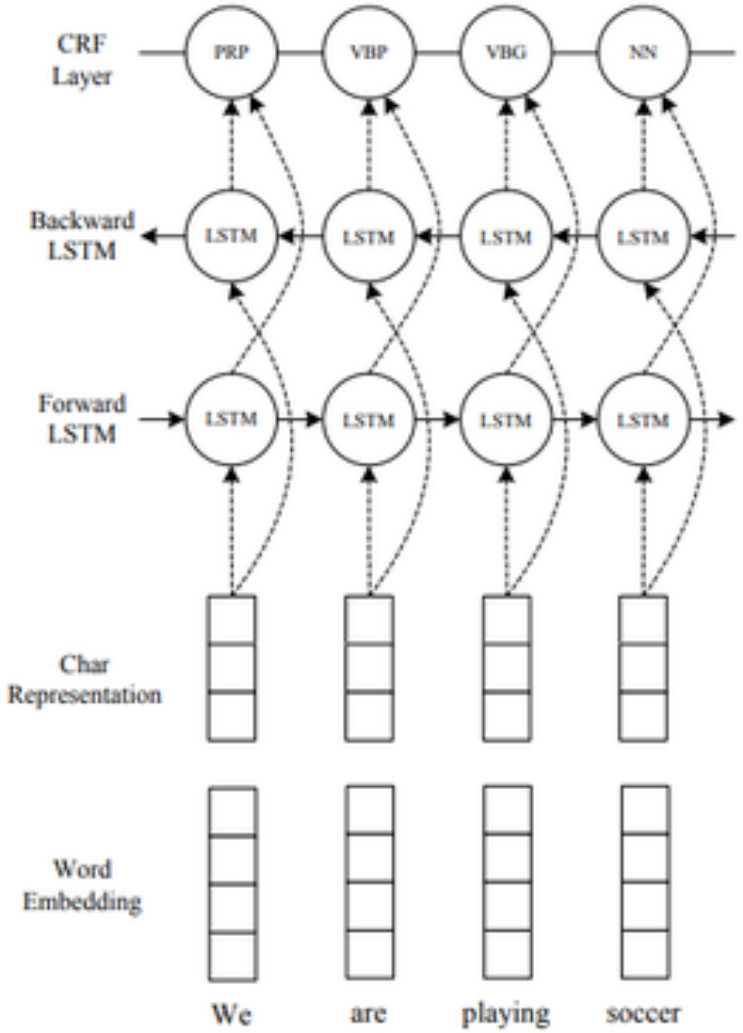
지금까지 BiLSTM + CRF와 BiLSTM + CNN 모델 두 가지를 배웠습니다.
여기서 추가적으로 또 어떤 방법들을 시도해볼 수 있을까요?

- **Pre-trained Embedding**

  Ex) Word2Vec, FastText, GloVe
- **Contextual Embedding**

  Ex) ELMo, Context2Vec
- **BiLSTM-CNN-CRF**

BiLSTM-CNN-CRF for Sequence Labeling



Sequence Labeling

지금까지 BiLSTM + CRF와 BiLSTM + CNN 모델 두 가지를 배웠습니다.
여기서 추가적으로 또 어떤 방법들을 시도해볼 수 있을까요?

- **Pre-trained Embedding**

  Ex) Word2Vec, FastText, GloVe
- **Contextual Embedding**

  Ex) ELMo, Context2Vec
- **BiLSTM-CNN-CRF**

Sequence Labeling

지금까지 BiLSTM + CRF와 BiLSTM + CNN 모델 두 가지를 배웠습니다.
여기서 추가적으로 또 어떤 방법들을 시도해볼 수 있을까요?

- **Pre-trained Embedding**
  Ex) Word2Vec, FastText, GloVe
- **Contextual Embedding**
  Ex) ELMo, Context2Vec
- **BiLSTM-CNN-CRF**
- **BERT ★ (8주차에 학습 예정)**

네이버 개체명 인식 대회 자료 Review

- 2018년 12월에 진행된 네이버-창원대 주최의 개체명 인식 대회
- Baseline Model은 Bidirectional RNN + CRF
- 우수한 모델들의 코드나 발표 자료가 게시되어져 있어서 리뷰하는 것을 권장드림.
- 단, 데이터는 어절 단위라서 실제 사용하기에는 무리가 있음.
- 평가 방법은 F1-Score를 사용.

발표 자료 보기

https://github.com/naver/nlp-challenge/blob/master/nlp-workshop.md#%EC%84%B8%EB%B6%80%EC%9D%BC%EC%A0%95

훈련 데이터 보기

https://raw.githubusercontent.com/naver/nlp-challenge/master/missions/ner/data/train/train_data

F1-SCORE

- 개체명인식에서는 그 어떤 개체도 아니라는 의미의 'O'라는 태깅이 존재한다.
- 그런데 전체 데이터에서 'O'는 대부분의 값을 차지하고 있다.
- 이로 인해 전부 'O'로 예측하더라도 정확도가 높아지는 상황이 발생한다.

true=['B-PER', 'I-PER', 'O', 'O', 'B-MISC', 'O','O','O','O','O','O','O','O','O','O','O','B-PER','I-PER','O','O','O','O','O','O','B-MISC','I-MISC','I-MISC','O','O','O','O','O','O','B-PER','I-PER','O','O','O','O','O']

predicted=['O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O']

위 경우의 정확도(Accuracy)는 몇 %일까요?

F1-SCORE

- 개체명인식에서는 그 어떤 개체도 아니라는 의미의 'O'라는 태깅이 존재한다.
- 그런데 전체 데이터에서 'O'는 대부분의 값을 차지하고 있다.
- 이로 인해 전부 'O'로 예측하더라도 정확도가 높아지는 상황이 발생한다.

true=['B-PER', 'I-PER', 'O', 'O', 'B-MISC', 'O','O','O','O','O','O','O','O','O','O','O','B-PER','I-PER','O','O','O','O','O','O','O','B-MISC','I-MISC','I-MISC','O','O','O','O','O','O','O','B-PER','I-PER','O','O','O','O','O']

predicted=['O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O']

위 경우의 정확도(Accuracy)는 몇 %일까요? **74%**

F1-SCORE

- 개체명인식에서는 그 어떤 개체도 아니라는 의미의 'O'라는 태깅이 존재한다.
- 그런데 전체 데이터에서 'O'는 대부분의 값을 차지하고 있다.
- 이로 인해 전부 'O'로 예측하더라도 정확도가 높아지는 상황이 발생한다.

true=['B-PER', 'I-PER', 'O', 'O', 'B-MISC', 'O','O','O','O','O','O','O','O','O','O','O','B-PER','I-PER','O','O','O','O','O','O','O','B-MISC','I-MISC','I-MISC','O','O','O','O','O','O','B-PER','I-PER','O','O','O','O','O']

predicted=['O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O']

위 경우의 정확도(Accuracy)는 몇 %일까요? **74%**

실제값에서도 대부분의 값이 'O'이기 때문에 그 어떤 개체도 찾지 못하였음에도 74%의 정확도를 얻습니다.

Confusion Matrix (Ex) Binary Classification)

		실제 정답	
		True	False
분류 결과	True	True Positive	False Positive
	False	False Negative	True Negative

- True Positive(TP) : 실제 True인 정답을 True라고 예측 (정답)
- False Positive(FP) : 실제 False인 정답을 True라고 예측 (오답)
- False Negative(FN) : 실제 True인 정답을 False라고 예측 (오답)
- True Negative(TN) : 실제 False인 정답을 False라고 예측 (정답)

Confusion Matrix (Ex) N=3)

	예측 클래스 0	예측 클래스 1	예측 클래스 2
정답 클래스 0	정답 클래스가 0, 예측 클래스가 0인 표본의 수	정답 클래스가 0, 예측 클래스가 1인 표본의 수	정답 클래스가 0, 예측 클래스가 2인 표본의 수
정답 클래스 1	정답 클래스가 1, 예측 클래스가 0인 표본의 수	정답 클래스가 1, 예측 클래스가 1인 표본의 수	정답 클래스가 1, 예측 클래스가 2인 표본의 수
정답 클래스 2	정답 클래스가 2, 예측 클래스가 0인 표본의 수	정답 클래스가 2, 예측 클래스가 1인 표본의 수	정답 클래스가 2, 예측 클래스가 2인 표본의 수

```
from sklearn.metrics import confusion_matrix

y_true = [2, 0, 2, 2, 0, 1]
y_pred = [0, 0, 2, 2, 0, 2]
```

```
confusion_matrix(y_true, y_pred)
```

Confusion Matrix (Ex) N=3)

	예측 클래스 0	예측 클래스 1	예측 클래스 2
정답 클래스 0	정답 클래스가 0, 예측 클래스가 0인 표본의 수	정답 클래스가 0, 예측 클래스가 1인 표본의 수	정답 클래스가 0, 예측 클래스가 2인 표본의 수
정답 클래스 1	정답 클래스가 1, 예측 클래스가 0인 표본의 수	정답 클래스가 1, 예측 클래스가 1인 표본의 수	정답 클래스가 1, 예측 클래스가 2인 표본의 수
정답 클래스 2	정답 클래스가 2, 예측 클래스가 0인 표본의 수	정답 클래스가 2, 예측 클래스가 1인 표본의 수	정답 클래스가 2, 예측 클래스가 2인 표본의 수

```
from sklearn.metrics import confusion_matrix

y_true = [2, 0, 2, 2, 0, 1]
y_pred = [0, 0, 2, 2, 0, 2]
```

```
confusion_matrix(y_true, y_pred)
```

```
array([[2, 0, 0],
       [0, 0, 1],
       [1, 0, 2]])
```

F1-Score

정밀도(Precision)

$$\text{precision} = \frac{TP}{TP + FP}$$

재현율(Recall)

$$\text{recall} = \frac{TP}{TP + FN}$$

- True Positive(TP) : 실제 True인 정답을 True라고 예측 (정답)
- False Positive(FP) : 실제 False인 정답을 True라고 예측 (오답)
- False Negative(FN) : 실제 True인 정답을 False라고 예측 (오답)
- True Negative(TN) : 실제 False인 정답을 False라고 예측 (정답)

F1-Score

정밀도(Precision)

$$\text{precision} = \frac{TP}{TP + FP}$$

재현율(Recall)

$$\text{recall} = \frac{TP}{TP + FN}$$

- True Positive(TP) : 실제 True인 정답을 True라고 예측 (정답)
- False Positive(FP) : 실제 False인 정답을 True라고 예측 (오답)
- False Negative(FN) : 실제 True인 정답을 False라고 예측 (오답)
- True Negative(TN) : 실제 False인 정답을 False라고 예측 (정답)

F-1 Score

$$F_1 = 2 \cdot \text{precision} \cdot \text{recall} \,/\, (\text{precision} + \text{recall})$$

F1-Score

```
from sklearn.metrics import classification_report

y_true = [0, 0, 0, 1, 1, 0, 0]
y_pred = [0, 0, 0, 0, 1, 1, 1]

print(classification_report(y_true, y_pred, target_names=['class 0', 'class 1']))
```

```
              precision    recall  f1-score   support

     class 0       0.75      0.60      0.67         5
     class 1       0.33      0.50      0.40         2

    accuracy                           0.57         7
   macro avg       0.54      0.55      0.53         7
weighted avg       0.63      0.57      0.59         7
```

- 0이라고 예측한 데이터의 75%만 실제로 0.
- 1이라고 예측한 데이터의 33%만 실제로 1.
- 실제 0인 데이터 중의 60%만 0으로 판별.
- 실제 1인 데이터 중의 50%만 1로 판별.

F1-Score

```
from sklearn.metrics import classification_report

y_true = [0, 0, 0, 1, 1, 0, 0]
y_pred = [0, 0, 0, 0, 1, 1, 1]

print(classification_report(y_true, y_pred, target_names=['class 0', 'class 1']))
```

```
              precision    recall  f1-score   support

     class 0       0.75      0.60      0.67         5
     class 1       0.33      0.50      0.40         2

    accuracy                           0.57         7
   macro avg       0.54      0.55      0.53         7
weighted avg       0.63      0.57      0.59         7
```

- 0이라고 예측한 데이터의 75%만 실제로 0.
- 1이라고 예측한 데이터의 33%만 실제로 1.
- 실제 0인 데이터 중의 60%만 0으로 판별.
- 실제 1인 데이터 중의 50%만 1로 판별.

macro: 단순평균

weighted: 각 클래스에 속하는
표본의 갯수로 가중평균

accuracy: 전체 학습데이터의 개수에서
각 클래스에서 자신의 클래스를
정확하게 맞춘 개수의 비율.

F1-Score (MultiClass Classification)

```
from sklearn.metrics import classification_report

y_true = [0, 0, 1, 1, 2, 2, 2]
y_pred = [0, 0, 1, 2, 2, 2, 1]
target_names = ['class 0', 'class 1', 'class 2']
print(classification_report(y_true, y_pred, target_names=target_names))
```

```
              precision    recall  f1-score   support

     class 0       1.00      1.00      1.00         2
     class 1       0.50      0.50      0.50         2
     class 2       0.67      0.67      0.67         3

    accuracy                           0.71         7
   macro avg       0.72      0.72      0.72         7
weighted avg       0.71      0.71      0.71         7
```

- 다중 클래스 분류의 경우에는 자기 자신을 양성, 다른 클래스를 음성으로 판단.

macro: 단순평균

weighted: 각 클래스에 속하는 표본의 갯수로 가중평균

accuracy: 전체 학습데이터의 개수에서 각 클래스에서 자신의 클래스를 정확하게 맞춘 개수의 비율.

Intent Classification

Text Classification

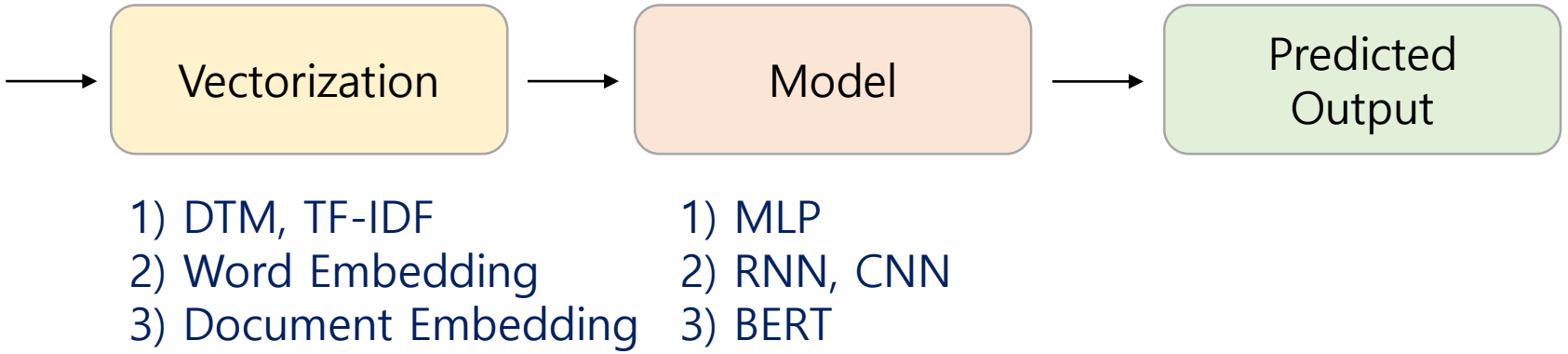
- 인공 신경망을 이용한 텍스트 분류 과정을 그림으로 도식화하면 아래와 같다.
- 분류는 크게 **이진 분류(Binary Classification)**과 **다중 클래스 분류(multiclass classification)**가 있다.

문서1 : me free lottery

문서2 : free get free you

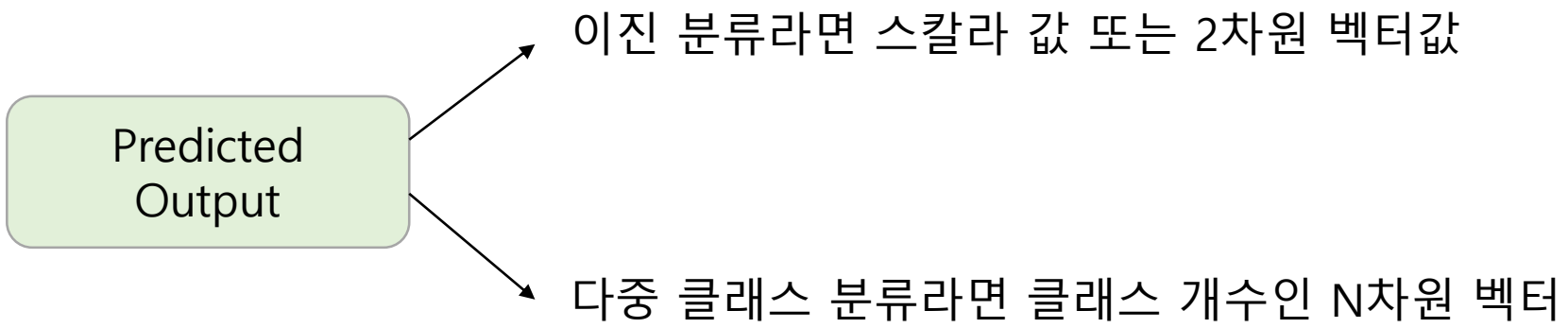
문서3 : you free scholarship

문서4 : free to contact me



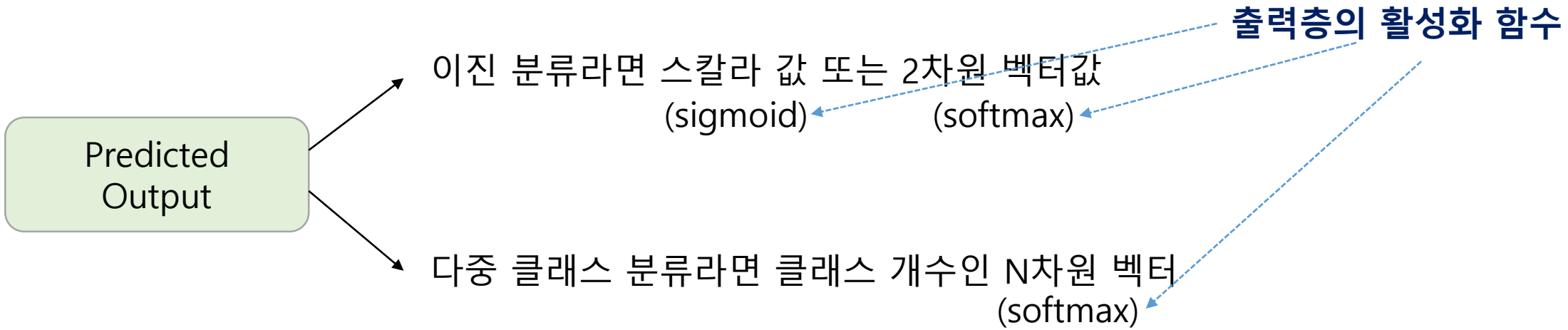
Text Classification

- 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



5주차 PJT

- 네이버-창원대학교에서 2018년에 개최한 개체명 인식 데이터셋 (한국어)
- 해당 데이터셋으로부터 BiLSTM-CRF 모델을 구현하고, F1 Score를 계산해보세요.
- BiLSTM-CRF 참고 링크 : https://wikidocs.net/34156
- 데이터 : https://raw.githubusercontent.com/naver/nlp-challenge/master/missions/ner/data/train/train_data
- 데이터 로드 방법 : https://colab.research.google.com/drive/1XfrnzhG_bj5fICCWiL2wzGNIL6ZL7vyx?usp=sharing#scrollTo=cJ_FWxbS6DmX