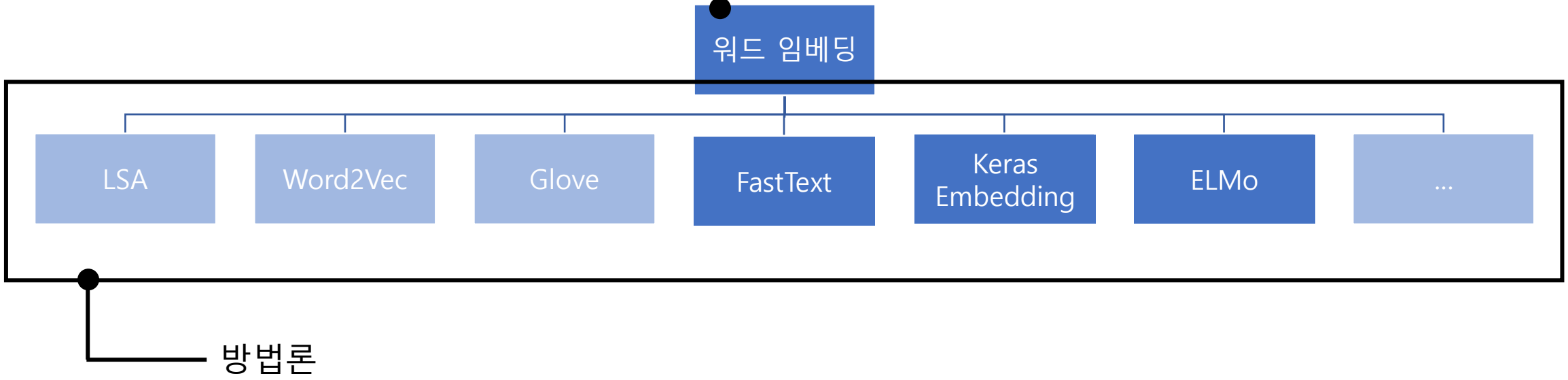
워드 임베딩
(Word Embedding) II

초급 13조
배수현, 변재경

목차

워드 임베딩

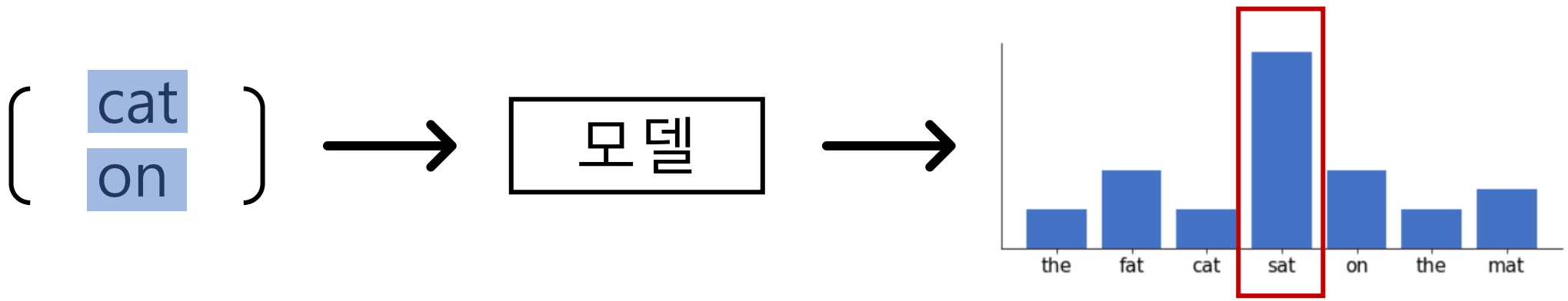
- 단어를 밀집 벡터(dense vector)의 형태로 표현하는 방법
- 워드 임베딩 과정을 통해 나온 결과 밀집 벡터 : 임베딩 벡터



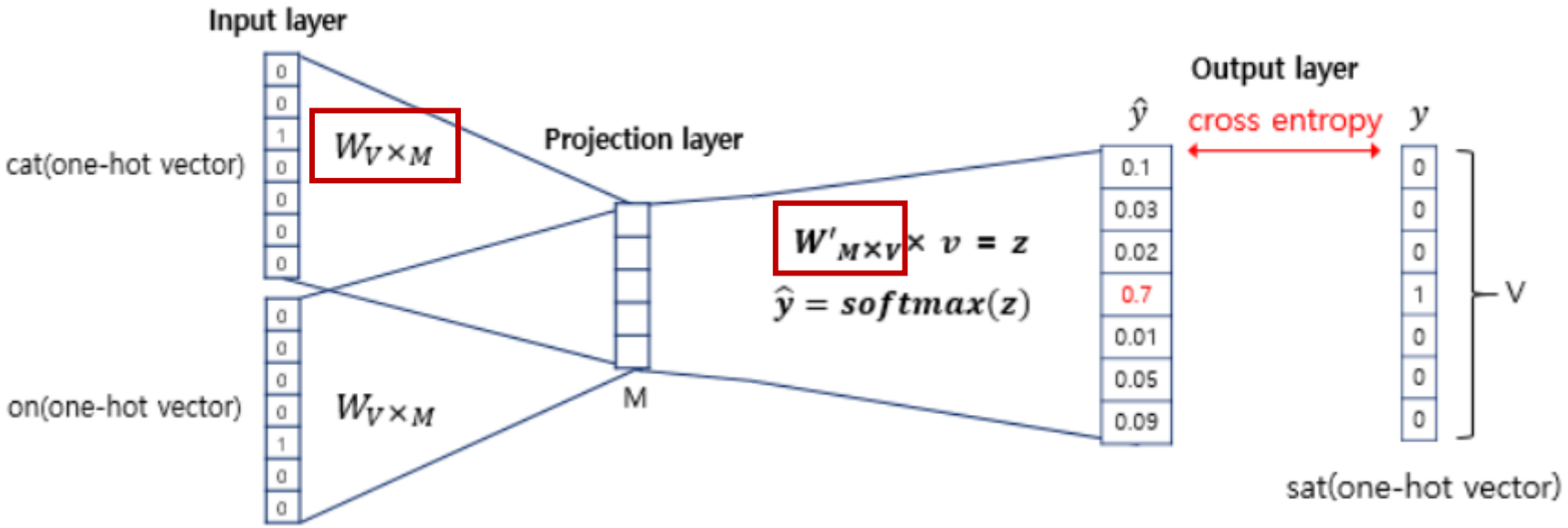
패스트텍스트(FastText)

Word2Vec

- 단어를 **밀집벡터**로 표현하는 방법 중 하나
- **추론 기반** 기법
  - The fat cat ? on the mat

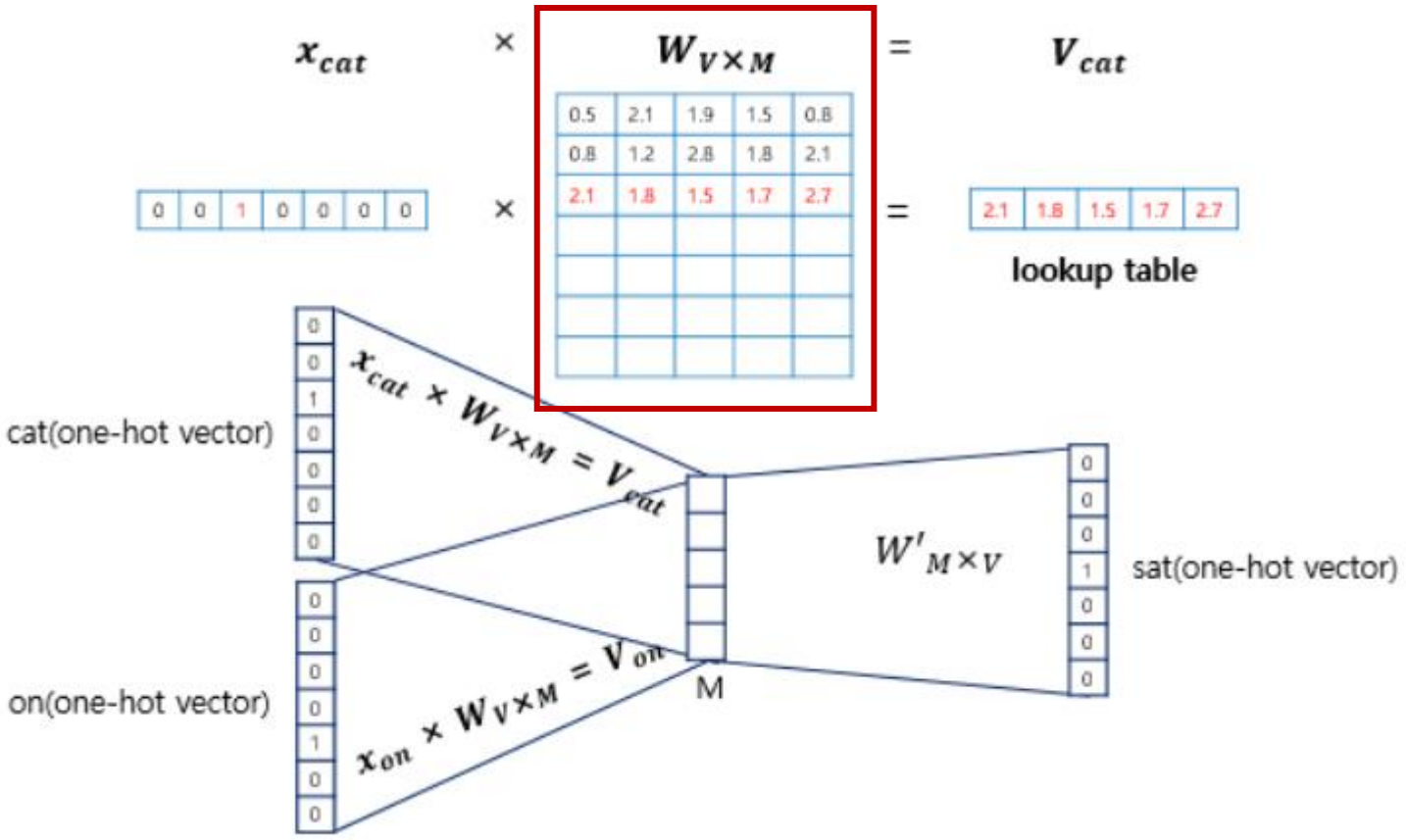


패스트텍스트(FastText)



https://wikidocs.net/22660

패스트텍스트(FastText)



https://wikidocs.net/22660

패스트텍스트(FastText)

- 페이스북에서 개발
- Word2Vec의 확장
- Word2Vec와 달리 내부 단어(subword)를 고려하여 학습

Subword (n_gram=3_gram)

apple

“<ap”, “app”, “ppl”, “ple”, “le>”, “<apple>"

special subword

패스트텍스트(FastText)

- Word2Vec dictionary

단어	원-핫 인코딩
I	[1 0 0 0]
like	[0 1 0 0]
delicious	[0 0 1 0]
apple	[0 0 0 1]

단어	임베딩 벡터
I	[0.2 0.7 2.1]
like	[0.3 1.1 1.2]
delicious	[1.2 3.1 3.1]
apple	[1.1 2.2 1.4]

패스트텍스트(FastText)

- 패스트텍스트(FastText) dictionary

[“I”, “like”, “delicious”, “apple”]
Subword (n_gram=3_gram)
like
“<li”, “lik”, “ike”, “ke>”, “<like>"
delicious
“<de”, “del”, “eli”, “lic”, “ici”, “cio“, “iou”, “ous”, “us>”, “<delicious>"
apple
“<ap”, “app”, “ppl”, “ple”, “le>”, “<apple>"

패스트텍스트(FastText)

- 패스트텍스트(FastText) dictionary

단어	원-핫 인코딩
<li	[1 0 0 0 0 0 0 0 ..]
lik	[0 1 0 0 0 0 0 0 ..]
ike	[0 0 1 0 0 0 0 0 ..]
ke>	[0 0 0 1 0 0 0 0 ..]
<like>	[0 0 0 0 0 1 0 0 ..]
<de	[0 0 0 0 0 0 1 0 ..]
del...	[0 0 0 0 0 0 0 1 ..]

단어	임베딩 벡터
<li	[-0.16 0.11 -0.06]
lik	[-0.13 0.05 -0.07]
ike	[-0.00 -0.02 -0.05]
ke>	[-0.02 -0.01 0.01]
<like>	[-0.08 0.00 -0.02]
<de	[-0.05 0.14 -0.03]
del...	[-0.11 0.17 0.04]

패스트텍스트(FastText)

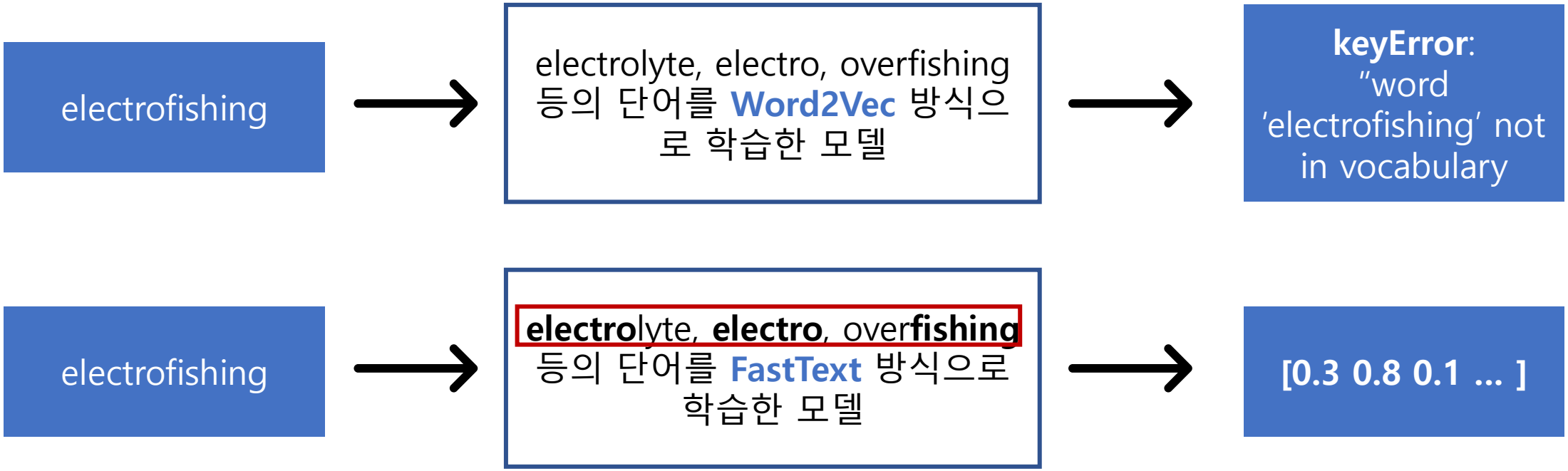
- 모르는 단어(Out Of Vocabulary, OOV)에 대한 대응

```
from gensim.models import FastText
sentences = [["I", "like", "delicious", "apple"]]

model = FastText(sentences, size=3, min_n=1, max_n=3, min_count=1)

print(model.wv['del'])
print(model.wv['ke>'])
```

```
[-0.11116432 0.17243099 0.04147036]
[-0.02032711 -0.01019276 0.0149658]
```

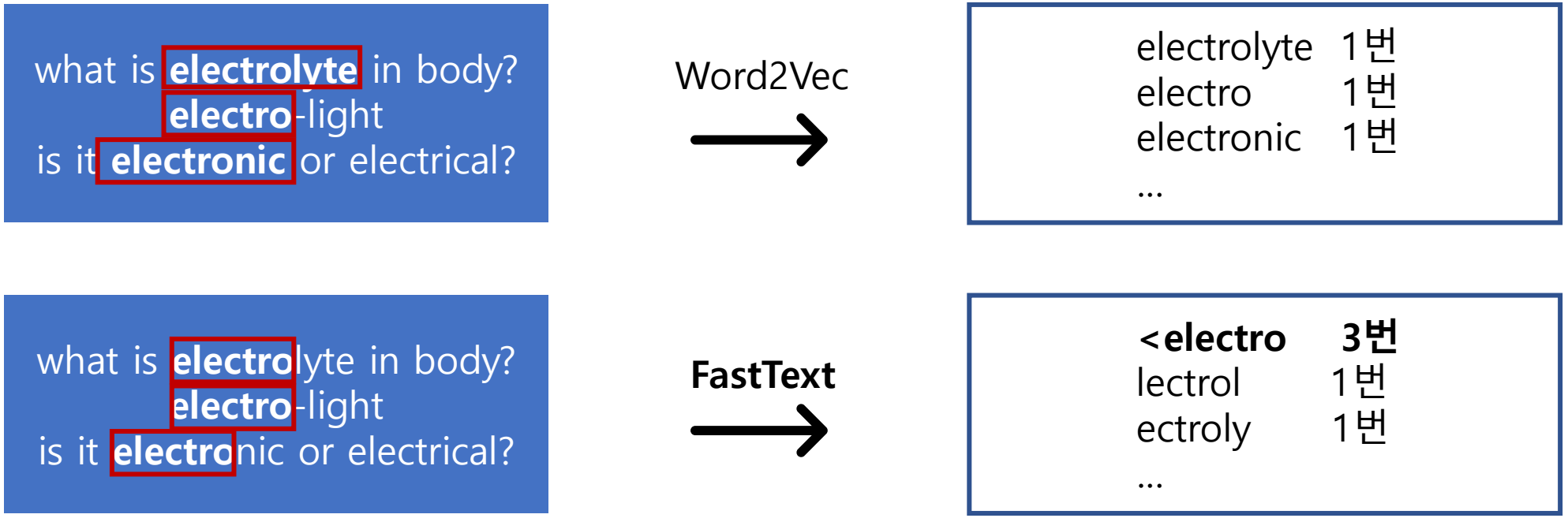
패스트텍스트(FastText)

- 모르는 단어(Out Of Vocabulary, OOV)에 대한 대응



패스트텍스트(FastText)

- 단어 집합 내 빈도 수가 적었던 단어(Rare Word)에 대한 대응



패스트텍스트(FastText)

- 한글 패스트텍스트(FastText)

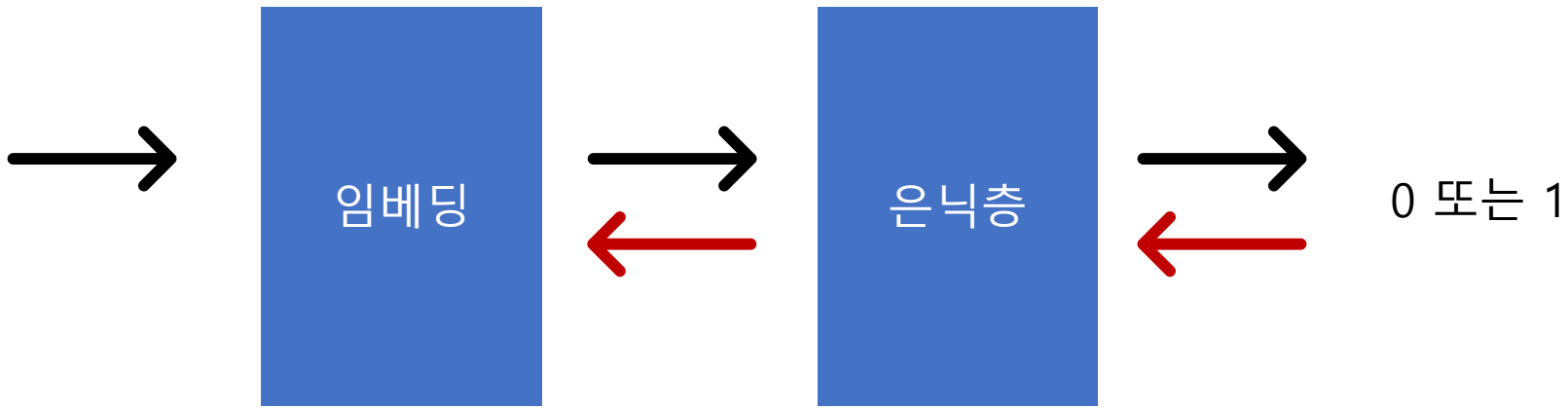
자연어
(1) 음절 단위
“<자연“, “자연어“, “연어처“, “어처리“, “처리>“, “<자연어처리>”

(2) 자모 단위
“<ㅈ ㅏ“, “ㅈ ㅏ_”, “ ㅏ_ㅇ”, … “ㄹ ㅣ >“, " <자연어처리>"

Keras Embedding()

- 긍정 / 부정 이진 분류

```
[[ 1  2 3 4]
 [ 5  6 0 0]
 [ 7  8 0 0]
 [ 9 10 0 0]
 [11 12 0 0]
 [13  0 0 0]
 [14 15 0 0]]
```



Keras Embedding()

- 긍정 / 부정 이진 분류

```
# 학습 데이터
sentences = ['nice great best amazing',
             'stop lies',
             'pitiful nerd',
             'excellent work',
             'supreme quality',
             'bad',
             'highly respectable']

y_train   = [1, 0, 0, 1, 1, 0, 1]  # 긍정은 1, 부정은 0
```

Keras Embedding()

- 긍정 / 부정 이진 분류

```
# token화
t = Tokenizer()
t.fit_on_texts(sentences)
vocab_size = len(t.word_index) + 1

print(vocab_size)

16
```

Keras Embedding()

- 긍정 / 부정 이진 분류

```
['nice great best amazing',
 'stop lies',
 'pitiful nerd',
 'excellent work',
 'supreme quality',
 'bad',
 'highly respectable']
```

```
# 정수 인코딩
X_encoded = t.texts_to_sequences(sentences)
print(X_encoded)
```

```
[[1, 2, 3, 4], [5, 6], [7, 8], [9, 10], [11, 12], [13], [14, 15]]
```

```
# 가장 긴 문장의 길이 구하기
max_len = max(len(l) for l in X_encoded)
print(max_len)
```

```
4
```

Keras Embedding()

- 긍정 / 부정 이진 분류

```
# 가장 긴 문장의 길이로 맞추기
X_train = pad_sequences(X_encoded, maxlen=max_len, padding='post')
y_train = np.array(y_train)
print(X_train)
```

```
[[ 1  2 3 4]
 [ 5  6 0 0]
 [ 7  8 0 0]
 [ 9 10 0 0]
 [11 12 0 0]
 [13  0 0 0]
 [14 15 0 0]]
```

Keras Embedding()

- 긍정 / 부정 이진 분류

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Embedding, Flatten

model = Sequential()
model.add(Embedding(vocab_size, 4, input_length=max_len))
model.add(Flatten())
model.add(Dense(1, activation='sigmoid'))
```

Keras Embedding()

- 임베딩 벡터들의 값은 학습 과정에서 다른 가중치들과 함께 학습

```
from tensorflow.keras.layers import Embedding

v = Embedding(20000, 128, input_length=max_len)
```

  - **vocab_size** = 20000
    텍스트 데이터의 전체 단어 집합의 크기.
  - **output_dim** = 128
    워드 임베딩 후의 임베딩 벡터의 차원.
  - **input_length** = 50
    입력 시퀀스의 길이. 만약 각 텍스트 데이터의 각 샘플의 길이가 500 개의 단어로 구성되어 있다면, 이 값은 500.

Keras Embedding()

- 사전 학습된 Embedding()
  - [실습 코드] https://wikidocs.net/33793
  - [Glove] https://blog.keras.io/using-pre-trained-word-embeddings-in-a-keras-model.html
  - [Word2Vec] https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Embedding, Flatten, Input

model = Sequential()
e = Embedding(vocab_size, 300, weights=[embedding_matrix],
              input_length=max_len, trainable=False)
model.add(e)
model.add(Flatten())
model.add(Dense(1, activation='sigmoid'))
```

ELMo

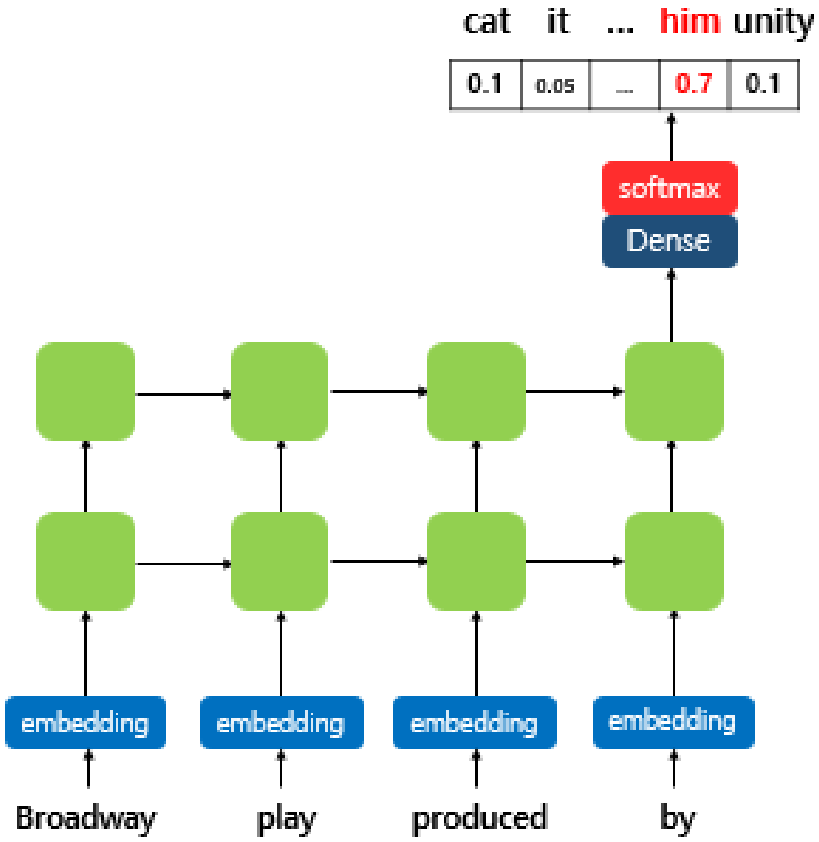
- ELMo: Embeddings from Language Models
- 사전 훈련된 언어 모델(Pre-trained language model)을 사용

pre-trained
word representation

deep contextualized
word representation

ELMo

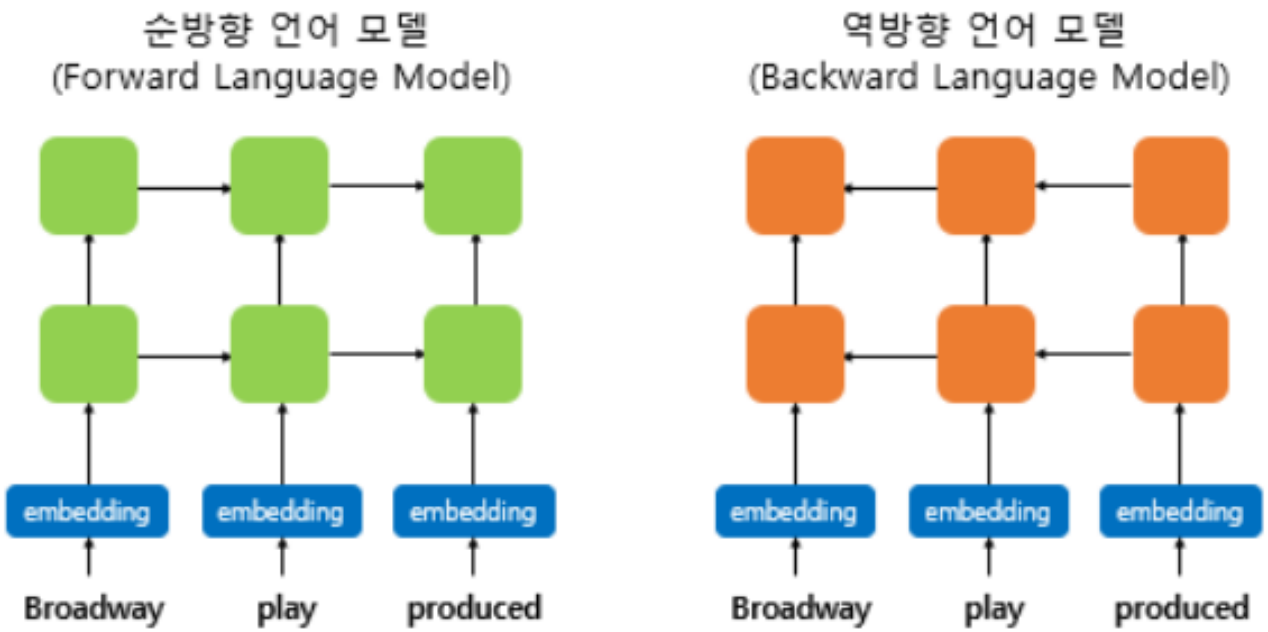
- biLM(Bidirectional Language Model)의 사전 훈련



https://wikidocs.net/33930

ELMo

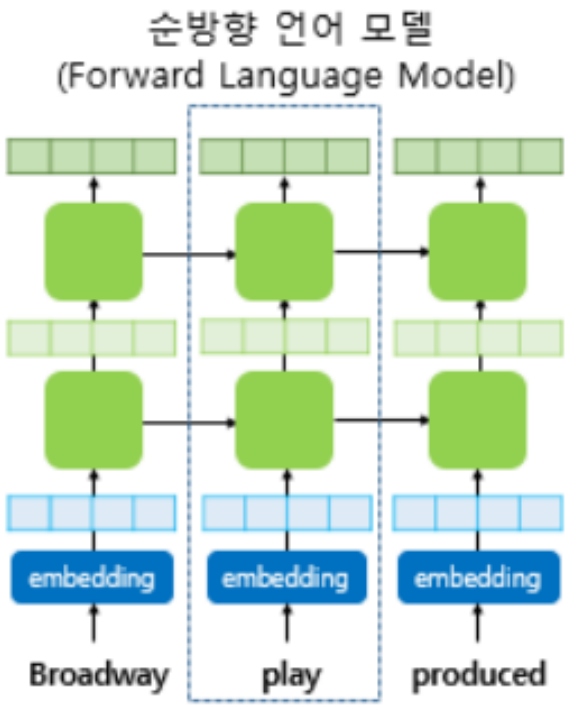
- biLM(Bidirectional Language Model)의 사전 훈련

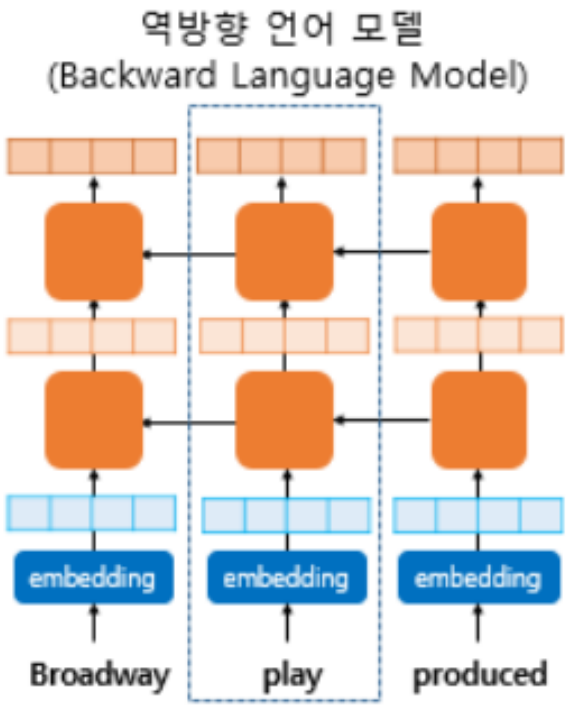


https://wikidocs.net/33930

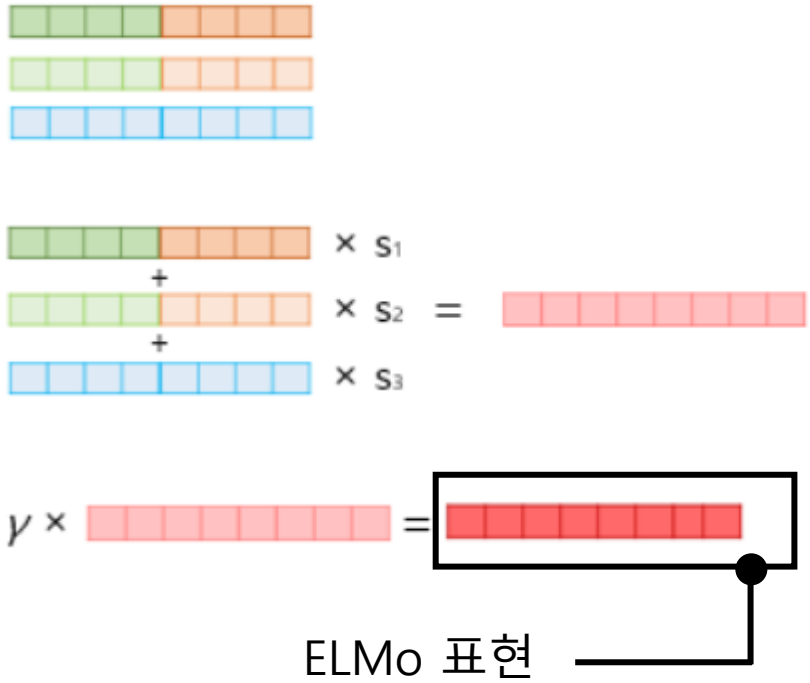
ELMo

- biLM(Bidirectional Language Model)의 활용



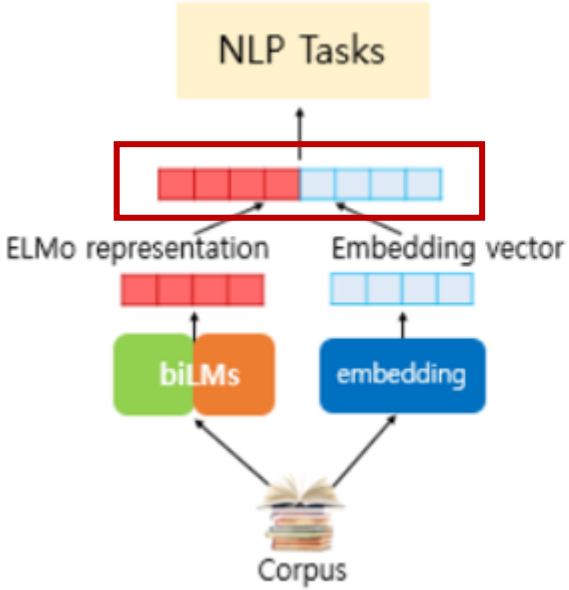


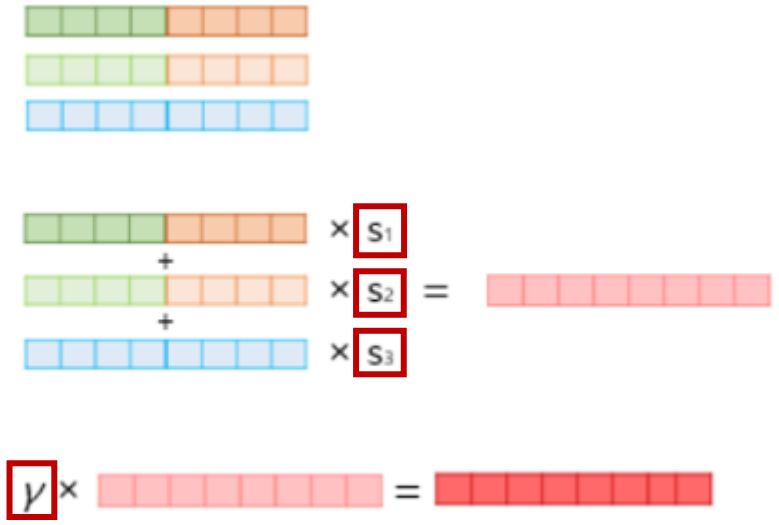
https://wikidocs.net/33930



ELMo

- NLP Task에서의 ELMo 활용

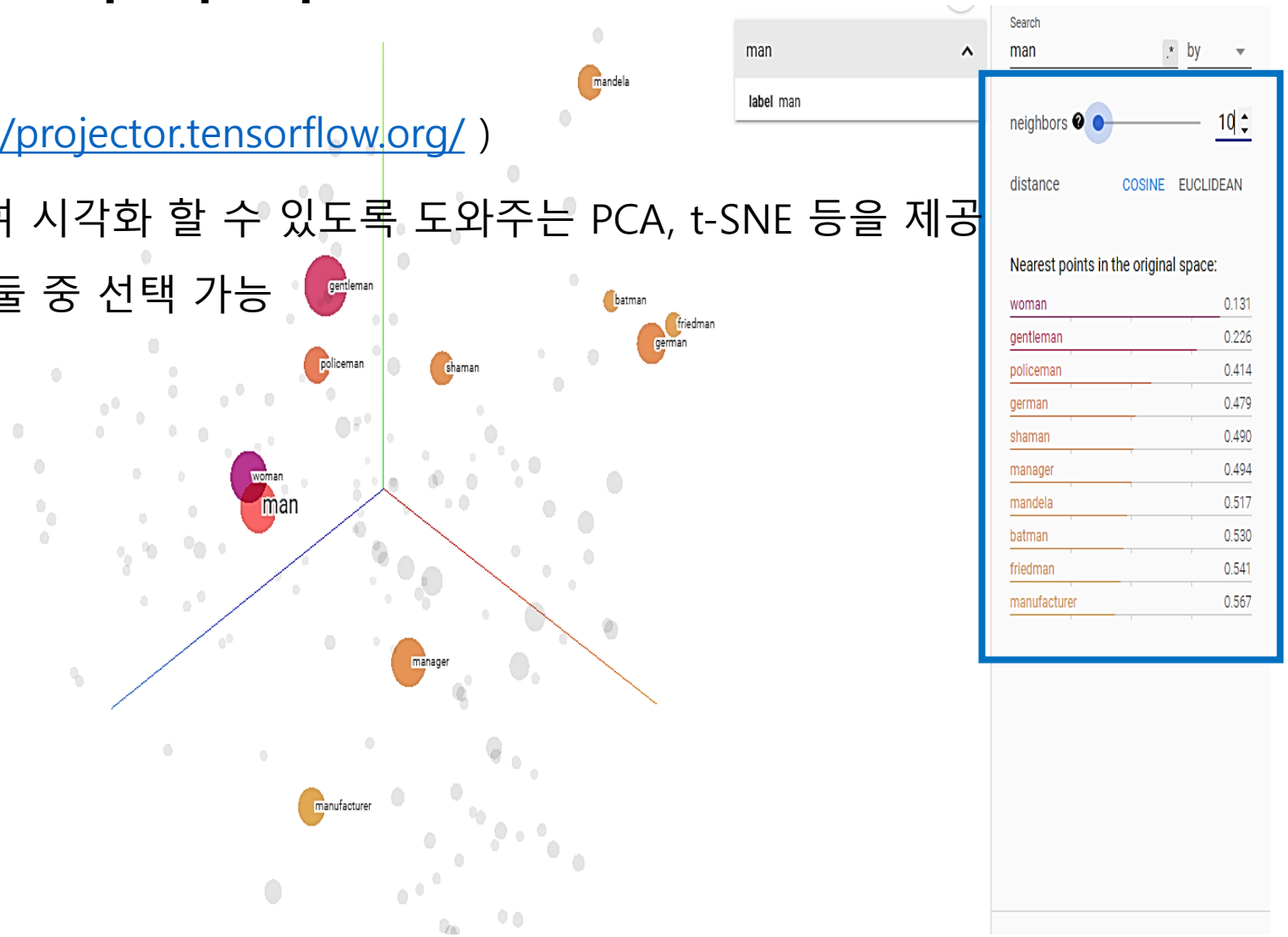




https://wikidocs.net/33930

임베딩 벡터의 시각화

- 구글의 임베딩 프로젝터 (https://projector.tensorflow.org/)
- 복잡한 데이터를 차원을 축소하여 시각화 할 수 있도록 도와주는 PCA, t-SNE 등을 제공
- distance : COSINE, EUCLIDEAN 둘 중 선택 가능



문서 벡터를 이용한 추천 시스템

1. 데이터 전처리
2. 문서 간 유사도 구하기
3. 추천 시스템 구현하기

GOAL :

입력 : 작품 제목

출력 : 입력의 작품과 가장 유사한 5개의 작품이 출력

1) 문서 데이터 전처리

(1) 전처리

```
Def _removeNonAscii(s):
    return  " ".join(I for I in s if  ord(i)<128)

Def make_lower_case(text):
    return text.lower()

Def remove_stop_words(text):
    text = text.split()
    stops = set(stopwords.words( " English " ))
    text = [w for w in text if not w in stops]
    text =  "  " .join(text)
    return text

def remove_html(text):
    html_pattern = re.compile('<.*?>')
    return html_pattern.sub(r'', text)

def remove_punctuation(text):
    tokenizer = RegexpTokenizer(r'[a-zA-Z]+')
    text = tokenizer.tokenize(text)
    text = " ".join(text)
    return text

df['cleaned'] = df['Desc'].apply(_removeNonAscii)
df['cleaned'] = df.cleaned.apply(make_lower_case)
df['cleaned'] = df.cleaned.apply(remove_stop_words)
df['cleaned'] = df.cleaned.apply(remove_punctuation)
df['cleaned'] = df.cleaned.apply(remove_html)
```

(2) Nan 제거

```
df['cleaned'].replace('', np.nan, inplace=True)
df = df[df['cleaned'].notna()]
```

(3) 토큰화를 수행

```
corpus = [] for words in df['cleaned']:
corpus.append(words.split())
```

Desc	Unnamed: 0.1	author	genre	image_link	rating	title	cleaned
We know that power is shifting: From West to E...	0.0	Moisés Naím	Business	https://i.gr-assets.com/images/S/compressed.ph...	3.63	The End of Power: From Boardrooms to Battlefie...	know power shifting west east north south pres...
Following the success of The Accidental Billio...	1.0	Blake J. Harris	Business	https://i.gr-assets.com/images/S/compressed.ph...	3.94	Console Wars: Sega, Nintendo, and the Battle t...	following success accidental billionaires mone...
How to tap the power of social software and ne...	2.0	Chris Brogan	Business	https://i.gr-assets.com/images/S/compressed.ph...	3.78	Trust Agents: Using the Web to Build Influence...	tap power social software networks build busin...
William J. Bernstein is an American financial ...	3.0	William J. Bernstein	Business	https://i.gr-assets.com/images/S/compressed.ph...	4.20	The Four Pillars of Investing	william j bernstein american financial theoris...
Amazing book. And I joined Steve Jobs and many...	4.0	Akio Morita	Business	https://i.gr-assets.com/images/S/compressed.ph...	4.05	Made in Japan: Akio Morita and Sony	amazing book joined steve jobs many akio morit...

(1) 전처리 → (2) Nan 제거

cleaned
know power shifting west east north south pres...
following success accidental billionaires mone...
tap power social software networks build busin...
william j bernstein american financial theoris...
amazing book joined steve jobs many akio morit...
...
ralph roberts sus setenta aos tras la muerte d...
murder vicarage marks debut agatha christies u...
john wyndham published novel day triffids mode...
classic book revealed flannery o connor one or...
imbued every page frank mccourt s astounding h...

(3) 토큰화

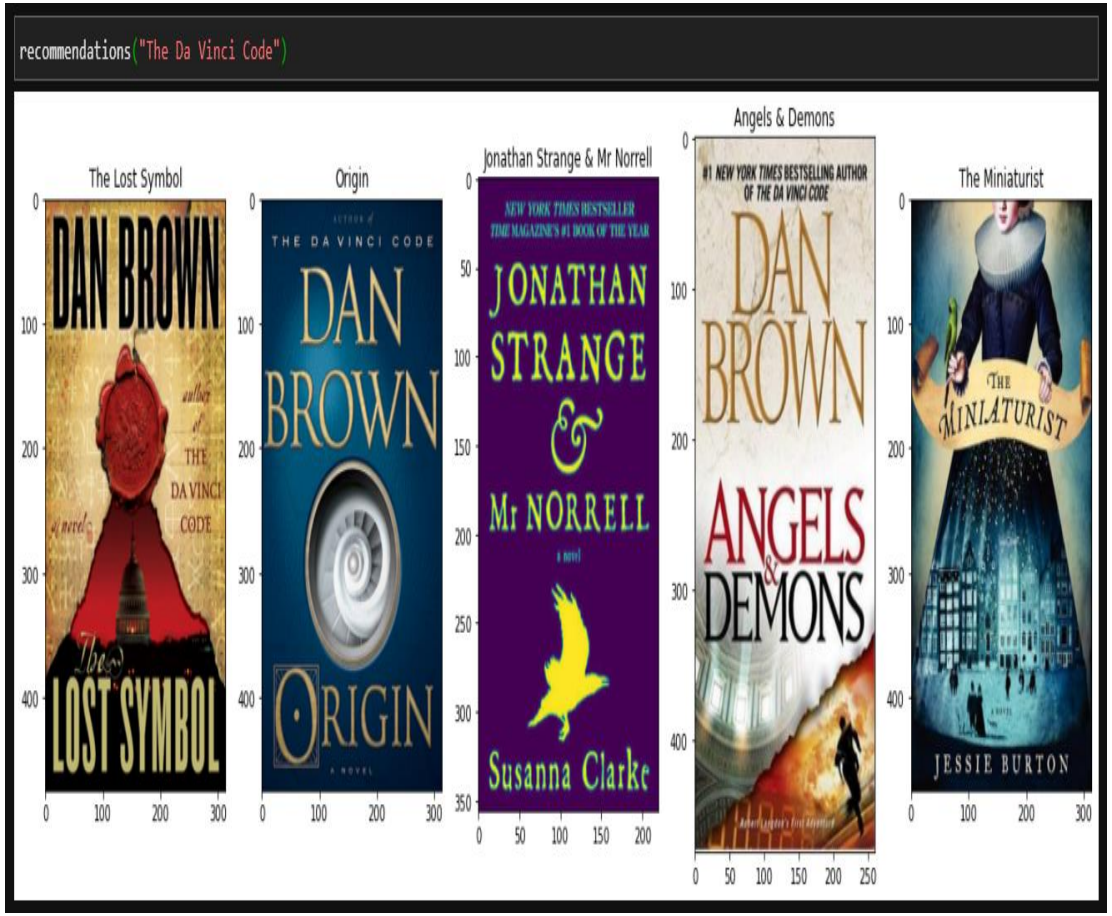
```
[['know',
  'power',
  'shifting',
  'west',
  'east',
  'north',
  'south',
  'presidential',
  'palaces',
  'public',
  'squares',
  'formidable',
  'corporate',
  'behemoths',
  'nimble',
  'startups',
  'and',
  'slowly',
  'surely',
  'men',
  'women',
  'power',
  'merely',
  'shifting',
  'dispersing',
```

2) 문서 간 유사도 구하기

```
# 사전 훈련되어있는 워드 임베딩 사용하기
urllib.request.urlretrieve("https://s3.amazonaws.com/dl4j-distribution/GoogleNews-vectors-negative300.bin.gz", \ filename="GoogleNews-vectors-
negative300.bin.gz")

word2vec_model = Word2Vec(size = 300, window=5, min_count = 2, workers = -1)
word2vec_model.build_vocab(corpus)
word2vec_model.intersect_word2vec_format('GoogleNews-vectors-negative300.bin.gz', lockf=1.0, binary=True)
word2vec_model.train(corpus, total_examples = word2vec_model.corpus_count, epochs = 15)


# 단어 벡터의 평균 구하기
def vectors(document_list):
    document_embedding_list = []

    # 각 문서에 대해서
    for line in document_list:
        doc2vec = None
        count = 0
        for word in line.split():
            if word in word2vec_model.wv.vocab:
                count += 1
                # 해당 문서에 있는 모든 단어들의 벡터값을 더한다.
                if doc2vec is None:
                    doc2vec = word2vec_model[word]
                else:
                    doc2vec = doc2vec + word2vec_model[word]

        if doc2vec is not None:
            # 단어 벡터를 모두 더한 벡터의 값을 문서 길이로 나눠준다.
            doc2vec = doc2vec / count
            document_embedding_list.append(doc2vec)

    # 각 문서에 대한 문서 벡터 리스트를 리턴
    return document_embedding_list
```

3) 추천시스템 구현하기

```
# 문서 벡터 간  코사인 유사도 계산하기
cosine_similarities = cosine_similarity(document_embedding_list,
document_embedding_list)
print('코사인 유사도 매트릭스의 크기 :',cosine_similarities.shape)

# 추천 함수
def recommendations(title):
    books = df[['title', 'image_link']]

    # 책의 제목을 입력하면 해당 제목의 인덱스를 리턴받아 idx에 저장.
    indices = pd.Series(df.index, index = df['title']).drop_duplicates()
    idx = indices[title]

    # 입력된 책과 줄거리(document embedding)가 유사한 책 5개 선정.
    sim_scores = list(enumerate(cosine_similarities[idx]))
    sim_scores = sorted(sim_scores, key = lambda x: x[1], reverse = True)
    sim_scores = sim_scores[1:6]

    # 가장 유사한 책 5권의 인덱스
    book_indices = [i[0] for i in sim_scores]

    # 전체 데이터프레임에서 해당 인덱스의 행만 추출. 5개의 행을 가진다.
    recommend = books.iloc[book_indices].reset_index(drop=True)

    fig = plt.figure(figsize=(20, 30))

    # 데이터프레임으로부터 순차적으로 이미지를 출력
    for index, row in recommend.iterrows():
        response = requests.get(row['image_link'])
        img = Image.open(BytesIO(response.content))
        fig.add_subplot(1, 5, index + 1)
        plt.imshow(img)
        plt.title(row['title'])
```



다빈치 코드는 작가 댄 브라운의 작품.
추천되는 작품들 또한 5개 중 3개가 댄 브라운의 작품들이 추천됨을 확인

워드 임베딩의 평균

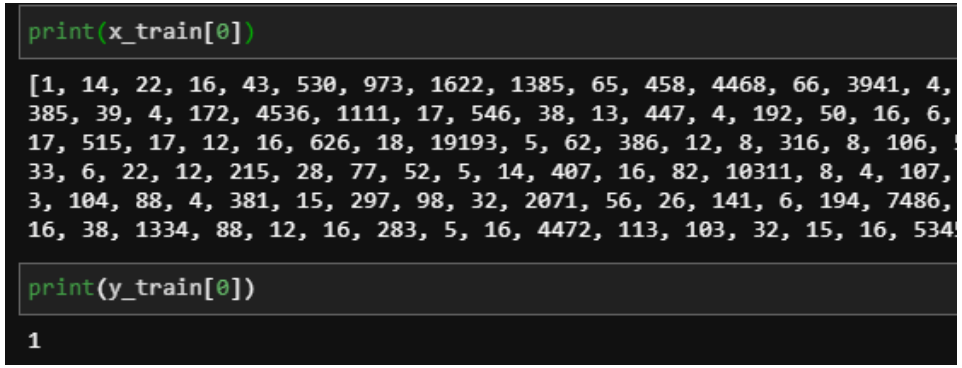
1. 데이터 전처리
2. 모델 설계하기

GOAL :
단어 벡터의 평균으로 분류 모델링하기

1) 문서 데이터 준비

```
# IMDB 영화 리뷰 데이터 불러오기
# 등장 빈도 순위로 20000번째에 해당하는 단어까지를 사용
vocab_size = 20000
(x_train, y_train), (x_test, y_test) =
imdb.load_data(num_words=vocab_size)
```

```
# 모든 리뷰의 길이를 400으로 맞추기 위한 패딩

max_len = 400

x_train = pad_sequences(x_train, maxlen=max_len)
x_test = pad_sequences(x_test, maxlen=max_len)
print('x_train의 크기(shape) :', x_train.shape)
print('x_test의 크기(shape) :', x_test.shape)
```



X : 토큰화와 정수 인코딩이라는 텍스트 전처리가 끝난 상태

Y : 긍정인 경우 1을 부정인 경우 0

-> 리뷰의 평균 길이보다 긴 400으로 패딩

2) 모델 설계하기

```
# 임베딩 벡터를 평균으로 사용하는 모델을 설계

model = Sequential()
model.add(Embedding(vocab_size, 50, input_length=max_len))
model.add(GlobalAveragePooling1D()) # 모든 단어 벡터의 평균을 구한다.
model.add(Dense(1, activation='sigmoid'))

es = EarlyStopping(monitor='val_loss', mode='min', verbose=1,
patience=4)
mc = ModelCheckpoint('embedding_average_model.h5',
monitor='val_acc', mode='max', verbose=1, save_best_only=True)

model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['acc']) model.fit(x_train, y_train, batch_size=32,
epochs=10, callbacks=[es, mc], validation_split=0.2)
```

GlobalAveragePooling1D()은 입력으로 들어오는 모든 벡터들의 평균을 구하는 역할.
Embedding() 다음에 **GlobalAveragePooling1D()**을 추가하면, 해당 문장의 모든 단어 벡터들의 평균 벡터를 구하게 됨.

별 다른 신경망을 추가하지 않고, 단어 벡터의 평균만으로도 88.74%라는 높은 정확도를 얻음.

```
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['acc'])
model.fit(x_train, y_train, batch_size=32, epochs=10, callbacks=[es, mc], validation_split=0.2)

Epoch 1/10
625/625 [==============================] - 10s 13ms/step - loss: 0.6729 - acc: 0.6390 - val_loss: 0.5404 - val_acc: 0.8106

Epoch 00001: val_acc improved from -inf to 0.81060, saving model to embedding_average_model.h5
Epoch 2/10
625/625 [==============================] - 8s 13ms/step - loss: 0.4901 - acc: 0.8405 - val_loss: 0.3978 - val_acc: 0.8628

Epoch 00002: val_acc improved from 0.81060 to 0.86280, saving model to embedding_average_model.h5
Epoch 3/10
625/625 [==============================] - 8s 13ms/step - loss: 0.3536 - acc: 0.8822 - val_loss: 0.3353 - val_acc: 0.8776

Epoch 00003: val_acc improved from 0.86280 to 0.87760, saving model to embedding_average_model.h5
Epoch 4/10
625/625 [==============================] - 8s 13ms/step - loss: 0.2846 - acc: 0.9043 - val_loss: 0.3101 - val_acc: 0.8774

Epoch 00004: val_acc did not improve from 0.87760
Epoch 5/10
625/625 [==============================] - 8s 13ms/step - loss: 0.2464 - acc: 0.9171 - val_loss: 0.2882 - val_acc: 0.8882

Epoch 00005: val_acc improved from 0.87760 to 0.88820, saving model to embedding_average_model.h5
Epoch 6/10
625/625 [==============================] - 8s 13ms/step - loss: 0.2097 - acc: 0.9309 - val_loss: 0.2784 - val_acc: 0.8914

Epoch 00006: val_acc improved from 0.88820 to 0.89140, saving model to embedding_average_model.h5
Epoch 7/10
625/625 [==============================] - 8s 12ms/step - loss: 0.1873 - acc: 0.9406 - val_loss: 0.2732 - val_acc: 0.8932

Epoch 00007: val_acc improved from 0.89140 to 0.89320, saving model to embedding_average_model.h5
Epoch 8/10
625/625 [==============================] - 8s 12ms/step - loss: 0.1655 - acc: 0.9466 - val_loss: 0.2715 - val_acc: 0.8938

Epoch 00008: val_acc improved from 0.89320 to 0.89380, saving model to embedding_average_model.h5
Epoch 9/10
625/625 [==============================] - 8s 12ms/step - loss: 0.1523 - acc: 0.9510 - val_loss: 0.2726 - val_acc: 0.8948

Epoch 00009: val_acc improved from 0.89380 to 0.89480, saving model to embedding_average_model.h5
Epoch 10/10
625/625 [==============================] - 8s 12ms/step - loss: 0.1342 - acc: 0.9585 - val_loss: 0.2743 - val_acc: 0.8948

Epoch 00010: val_acc did not improve from 0.89480
<tensorflow.python.keras.callbacks.History at 0x7f227b6d3ca0>
```

```
loaded_model = load_model('embedding_average_model.h5')
print("\n 테스트 정확도: %.4f" % (loaded_model.evaluate(x_test, y_test)[1]))

782/782 [==============================] - 4s 5ms/step - loss: 0.2837 - acc: 0.8874

 테스트 정확도: 0.8874
```

참고 자료

- [블로그] 딥 러닝을 이용한 자연어 처리 입문 https://wikidocs.net/book/2155
- [책] 밑바닥부터 시작하는 딥러닝 2
- [유튜브] Minsuk Heo 허민석 - ELMo (Deep contextualized word representations) https://www.youtube.com/watch?v=YZerhaFMPTw
- [유튜브] 고려대학교 산업경영공학부 DSBA 연구실 - 08-3: ELMo https://www.youtube.com/watch?v=zV8kIUwH32M

감사합니다.