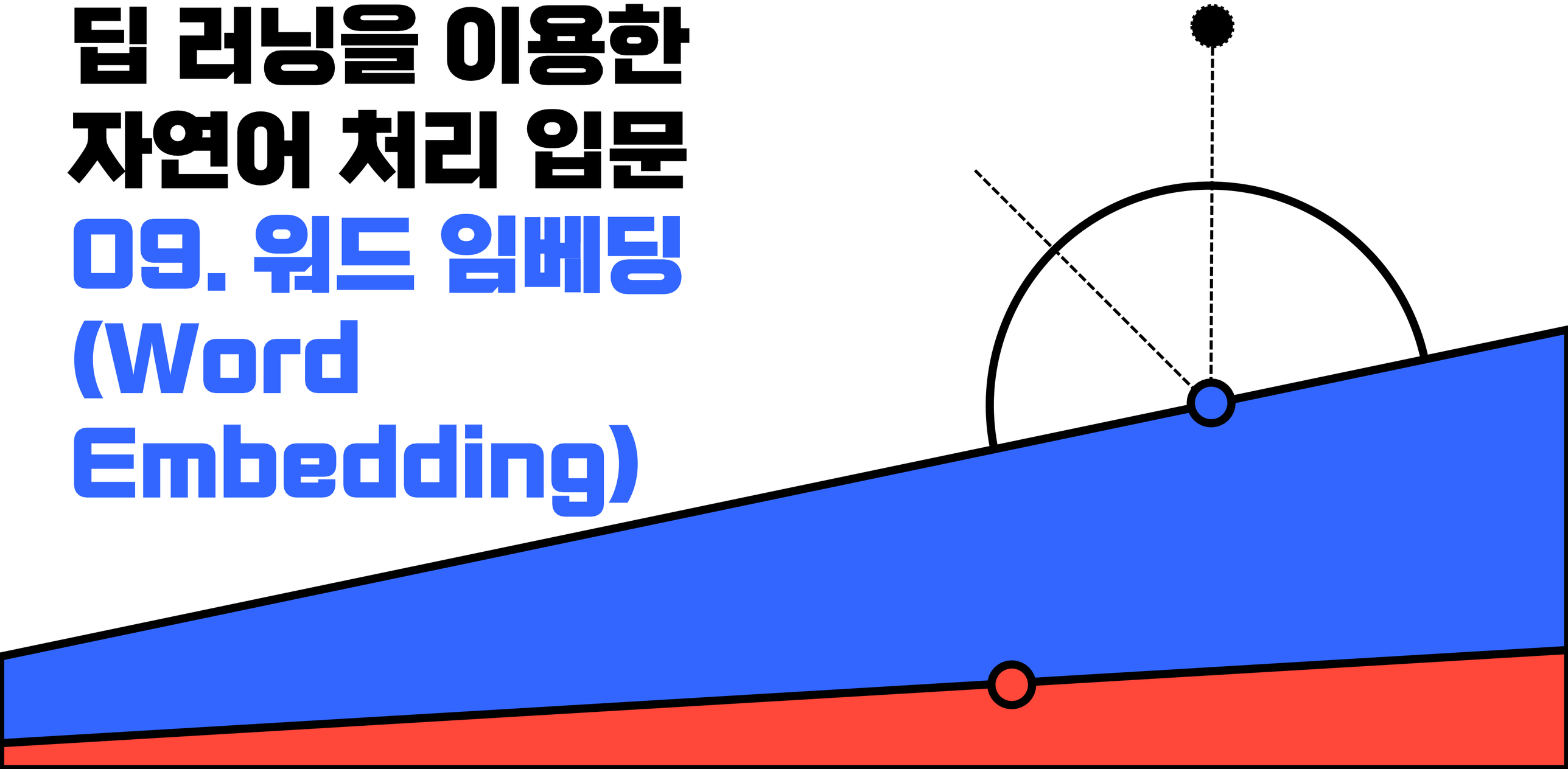
딥 러닝을 이용한 자연어 처리 입문

09. 워드 임베딩 (Word Embedding)



09-01 워드 임베딩(Word Embedding)

1. 희소 표현 (Sparse Representation):

원-핫 인코딩으로 표현

"I": [1, 0, 0, 0, 0]
"love": [0, 1, 0, 0, 0]
"cats": [0, 0, 1, 0, 0]

09-01 워드 임베딩(Word Embedding)

2. 밀집 표현 (Dense Representation):

고차원의 실수 벡터로 표현
"I", "love", "cats"를 각각 3차원의 벡터로 표현

"I": [0.2, 0.4, 0.1]
"love": [0.6, 0.8, 0.3]
"cats": [0.9, 0.5, 0.7]

모든 값이 0 또는 1이 아닌 실수값을 가지며,
각 차원은 단어의 의미적 특성을 반영

> 워드 임베딩은 밀집 표현과 관련이 있음
단어를 고차원의 실수 벡터로 표현하여 의미 정보를 담고
단어 간의 유사성을 반영

09-02 워드투벡터(Word2Vec)

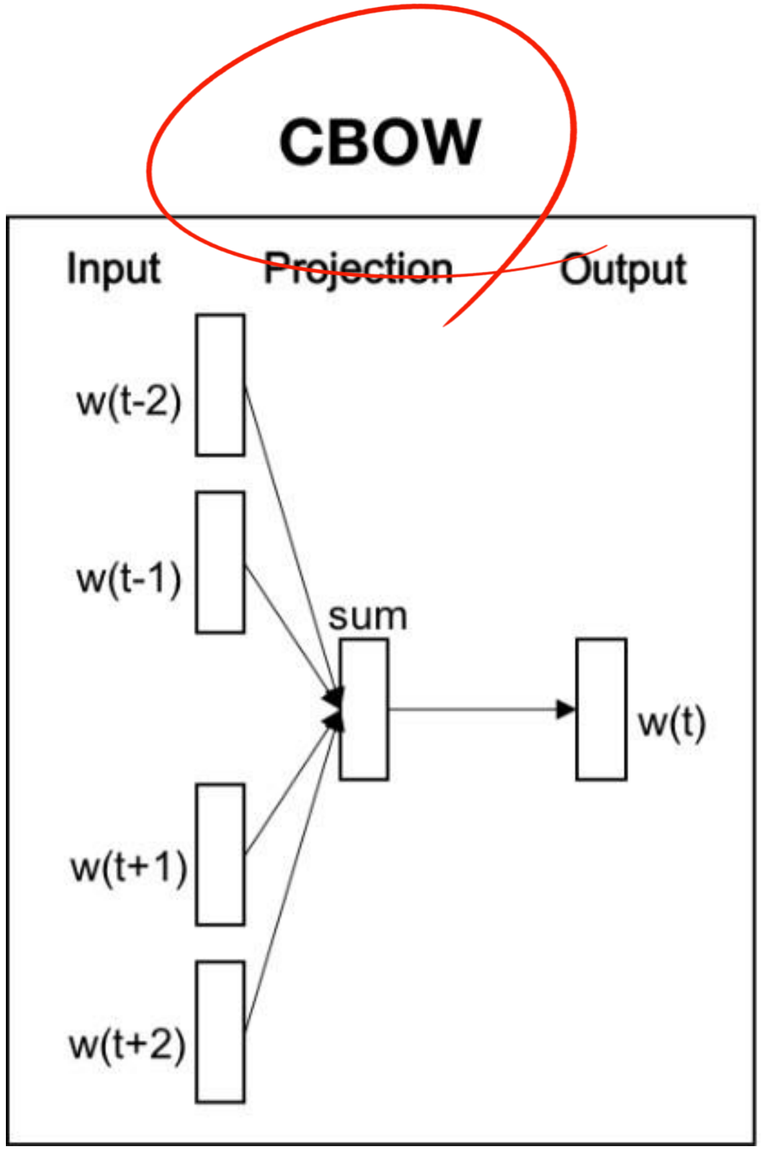
3. 분산 표현 (Distributed Representation):
주변 단어와의 관계를 고려

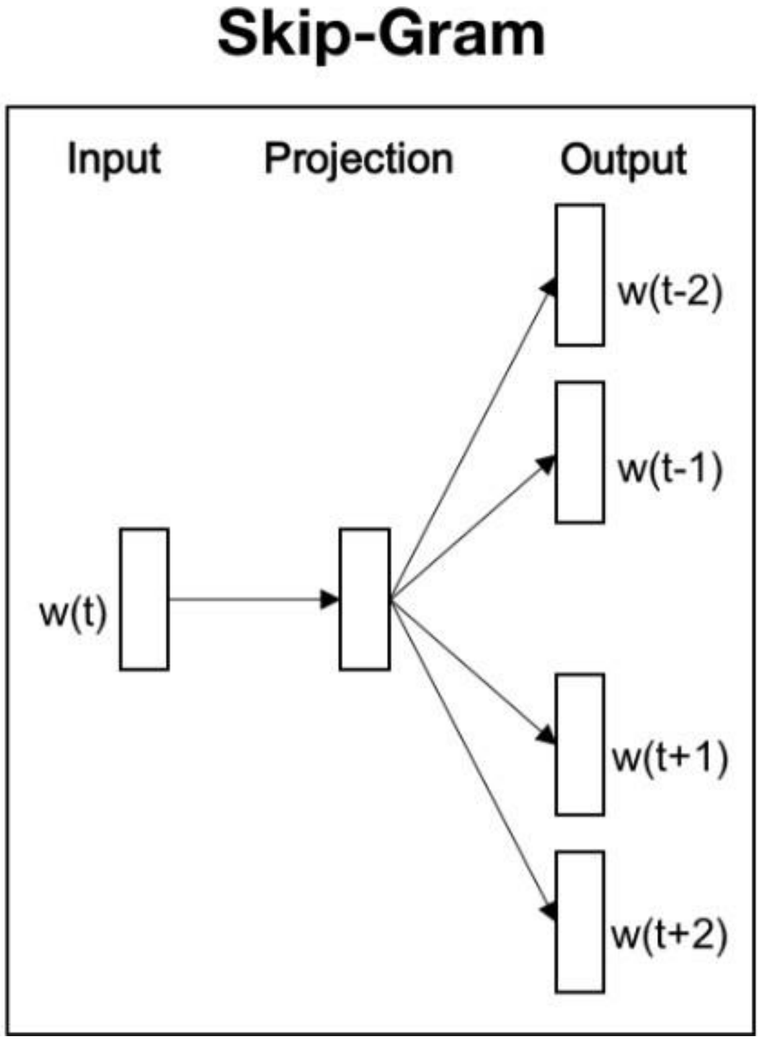
"I": [0.2, -0.1, 0.8]
"love": [0.7, 0.3, -0.5]
"cats": [-0.4, 0.9, 0.2]

단어의 의미 정보를 벡터의 여러 차원에 분산하여 표현
의미적으로 유사한 단어는 벡터 공간 상에서 가까이 위치

> 워드투벡터(Word2Vec)는
워드 임베딩을 생성하기 위한 특정한 알고리즘
분산 표현을 학습하는 방법

09-02 워드투벡터(Word2Vec)





09-02 워드투벡터(Word2Vec)

CBOW(Continuous Bag of Words)

중심 단어 / 주변 단어

The fat cat sat on the mat

The fat cat sat on the mat

The fat cat sat on the mat

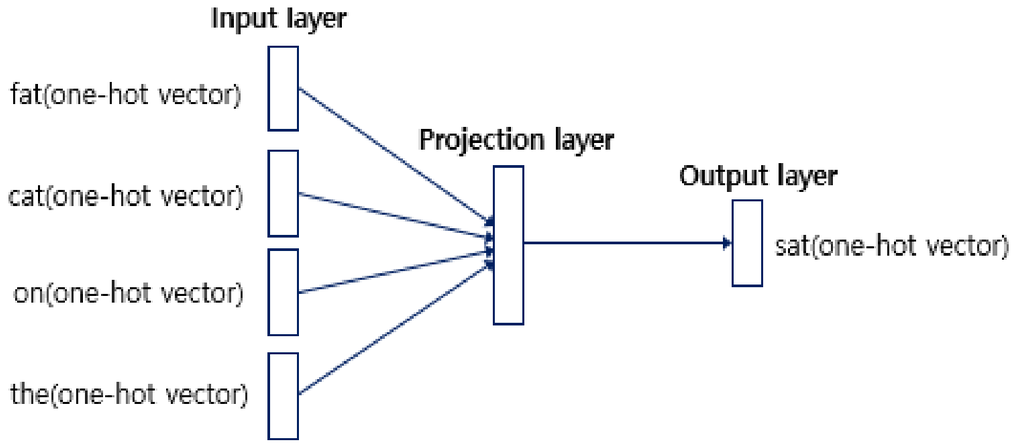
The fat cat sat on the mat

The fat cat sat on the mat

The fat cat sat on the mat

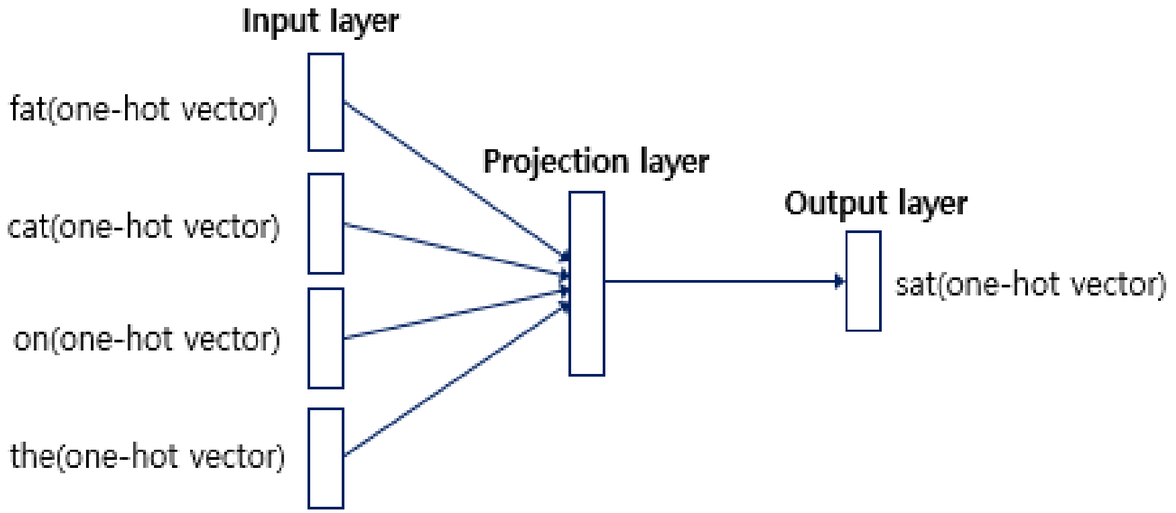
The fat cat sat on the mat

중심 단어	주변 단어
[1, 0, 0, 0, 0, 0, 0]	[0, 1, 0, 0, 0, 0, 0], [0, 0, 1, 0, 0, 0, 0]
[0, 1, 0, 0, 0, 0, 0]	[1, 0, 0, 0, 0, 0, 0], [0, 0, 1, 0, 0, 0, 0], [0, 0, 0, 1, 0, 0, 0]
[0, 0, 1, 0, 0, 0, 0]	[1, 0, 0, 0, 0, 0, 0], [0, 1, 0, 0, 0, 0, 0], [0, 0, 0, 1, 0, 0, 0], [0, 0, 0, 0, 1, 0, 0]
[0, 0, 0, 1, 0, 0, 0]	[0, 1, 0, 0, 0, 0, 0], [0, 0, 1, 0, 0, 0, 0], [0, 0, 0, 0, 1, 0, 0], [0, 0, 0, 0, 0, 1, 0]
[0, 0, 0, 0, 1, 0, 0]	[0, 0, 1, 0, 0, 0, 0], [0, 0, 0, 1, 0, 0, 0], [0, 0, 0, 0, 0, 1, 0], [0, 0, 0, 0, 0, 0, 1]
[0, 0, 0, 0, 0, 1, 0]	[0, 0, 0, 1, 0, 0, 0], [0, 0, 0, 0, 1, 0, 0], [0, 0, 0, 0, 0, 0, 1]
[0, 0, 0, 0, 0, 0, 1]	[0, 0, 0, 0, 1, 0, 0], [0, 0, 0, 0, 0, 1, 0]



09-02 워드투벡터(Word2Vec)

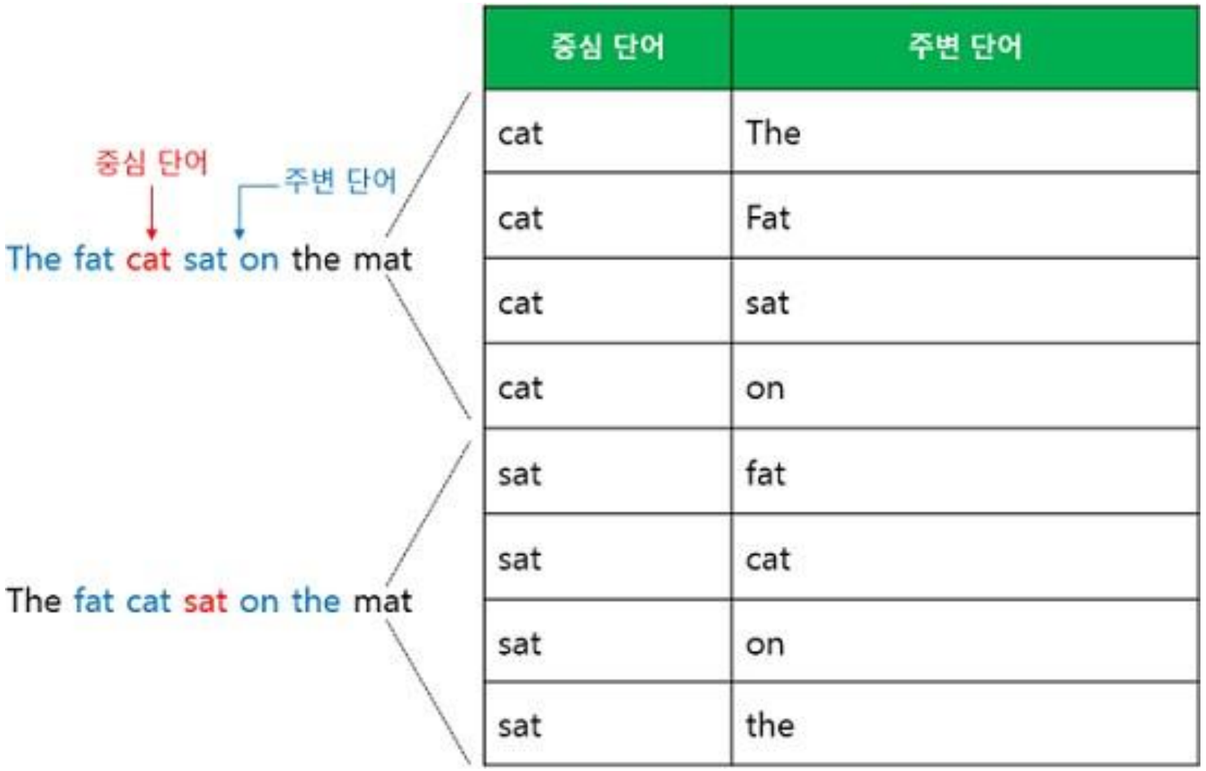
CBOW(Continuous Bag of Words)



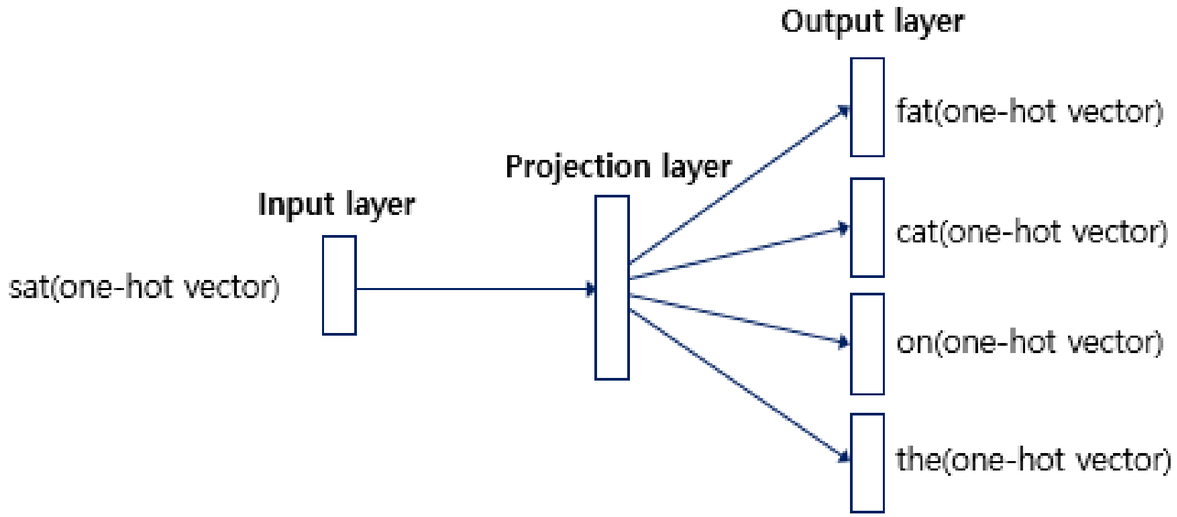
입력 데이터 준비 → 원-핫 인코딩 → 임베딩 → 평균화 → 출력 계층 통과 → 손실 함수와 학습

09-02 워드투벡터(Word2Vec)

Skip-gram



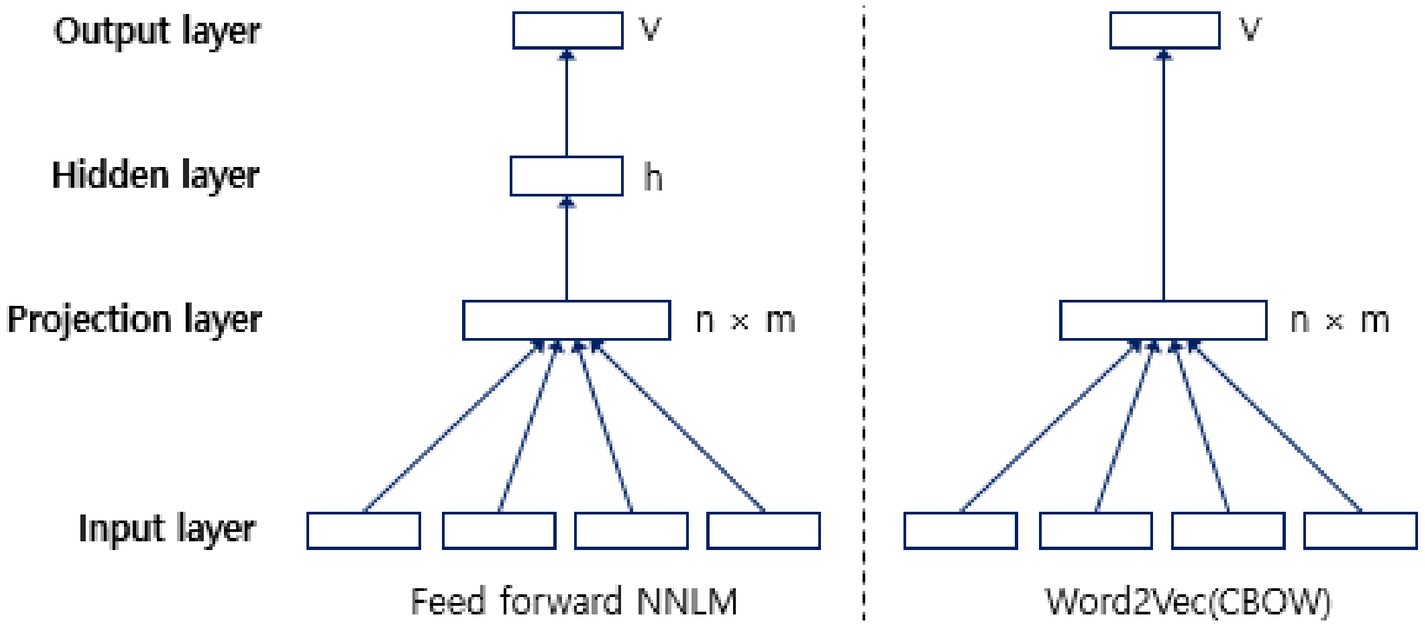
중심 단어	주변 단어
cat	The
cat	Fat
cat	sat
cat	on
sat	fat
sat	cat
sat	on
sat	the



09-02 워드투벡터(Word2Vec)

NNLM Vs. Word2Vec

- 예측대상
- 참고하는 단어 범위
- 구조



09-03 한국어 Word2Vec 실습

한국어 Word2Vec 만들기(네이버 영화 리뷰)

```
from gensim.models import Word2Vec

model = Word2Vec(sentences = tokenized_data, vector_size = 100, window = 5, min_count = 5, workers = 4, sg = 0)
```

```
# 완성된 임베딩 매트릭스의 크기 확인
model.wv.vectors.shape
```

```
(16477, 100)
```

```
print(model.wv.most_similar("최민식"))
```

```
[('한석규', 0.8789200782775879), ('안성기', 0.8757420778274536), ('김수현', 0.855679452419281), ('이민호', 0.8545516863822937), ('김명민', 0.8525030612945557), ('최민수', 0.8492398262023926), ('이성재', 0.8478372097015381), ('윤제문', 0.8470626473426819), ('김창완', 0.8456774950027466), ('이주승', 0.8442063927650452)]
```

```
print(model.wv.most_similar("히어로"))
```

```
[('슬래셔', 0.8747539520263672), ('느와르', 0.8666149377822876), ('무협', 0.8423701524734497), ('호러', 0.8372749090194702), ('물의', 0.8365858793258667), ('무비', 0.8260530233383179), ('물', 0.8197994232177734), ('홍콩', 0.8120777606964111), ('블록버스터', 0.8021541833877563), ('블랙', 0.7880141139030457)]
```

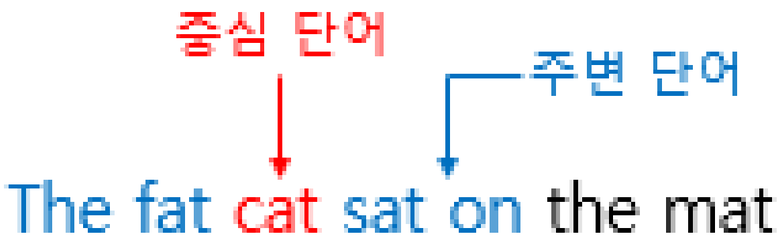
09-04 네거티브 샘플링을 이용한 Word2Vec 구현 (SGNS)

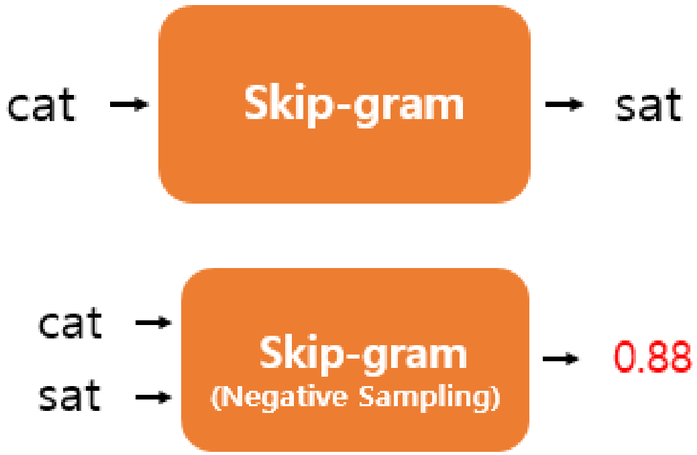
1. 네거티브 샘플링(Negative Sampling)

Word2Vec에서 학습 속도를 향상시키기 위해 전체 단어 집합 대신 일부 단어 집합을 사용하여 이진 분류 문제로 변환하는 기법

09-04 네거티브 샘플링을 이용한 Word2Vec 구현 (SGNS)

2. 네거티브 샘플링 Skip-Gram(Skip-Gram with Negative Sampling, SGNS)





입력과 레이블의 변화

입력1	입력2	레이블
cat	The	1
cat	fat	1
cat	sat	1
cat	on	1

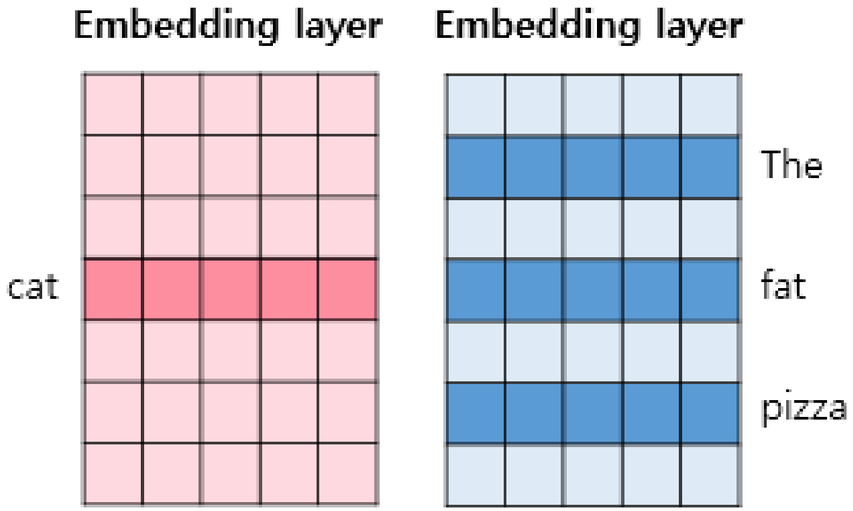
Negative Sampling

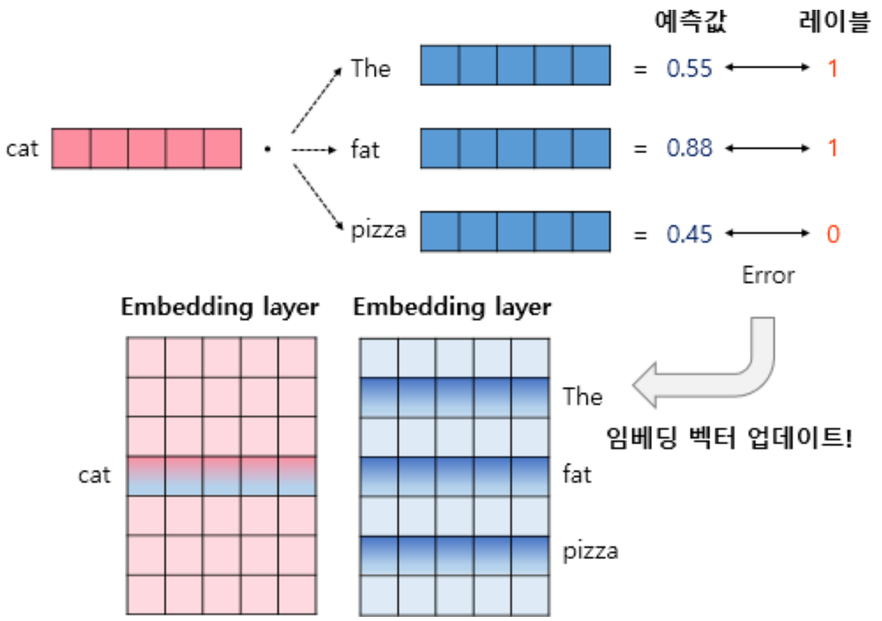
입력1	입력2	레이블
cat	The	1
cat	fat	1
cat	pizza	0
cat	computer	0
cat	sat	1
cat	on	1

단어 집합에서 랜덤으로 선택된 단어들을 레이블 0의 샘플로 추가.

09-04 네거티브 샘플링을 이용한 Word2Vec 구현 (SGNS)

2. 네거티브 샘플링 Skip-Gram(Skip-Gram with Negative Sampling, SGNS)





09-05 글로브(GloVe)

기존 방법론에 대한 비판

기존의 단어 임베딩 방법론들은
단어의 빈도 정보를 고려하지 않고,
맥락 단어를 이용하여 단어 간 유사성을 파악하는 데 어려움

이에 대한 비판으로 단어 빈도 정보의 중요성이 제기되었고,
글로브는 단어의 동시 등장 통계를 활용하여
의미적 관계를 더 잘 포착할 수 있는 벡터 표현을 제공

09-05 글로브(GloVe)

윈도우 기반 동시 등장 행렬(Window based Co-occurrence Matrix)

단어의 동시 등장 행렬은
전체 단어 집합을 행과 열로 구성하고,
윈도우 크기 내에서 **단어들이 함께 등장한 횟수**를
행렬의 **해당 위치에 표기**한 행렬

- I like deep learning
- I like NLP
- I enjoy flying

카운트	I	like	enjoy	deep	learning	NLP	flying
I	0	2	1	0	0	0	0
like	2	0	0	1	0	1	0
enjoy	1	0	0	0	0	0	1
deep	0	1	0	0	1	0	0
learning	0	0	0	1	0	0	0
NLP	0	1	0	0	0	0	0
flying	0	0	1	0	0	0	0

09-05 글로브(GloVe)

동시 등장 확률과 손실함수

GloVe의 아이디어:
임베딩 된 중심 단어와 주변 단어 벡터의 내적이
전체 코퍼스에서의 동시 등장 확률이 되도록 만드는 것

- X : 동시 등장 행렬(Co-occurrence Matrix)
- X_{ij} : 중심 단어 i가 등장했을 때 윈도우 내 주변 단어 j가 등장하는 횟수
- $X_i : \sum_j X_{ij}$: 동시 등장 행렬에서 i행의 값을 모두 더한 값
- $P_{ik} : P(k \mid i) = \frac{X_{ik}}{X_i}$: 중심 단어 i가 등장했을 때 윈도우 내 주변 단어 k가 등장할 확률
  Ex) P(solid I ice) = 단어 ice가 등장했을 때 단어 solid가 등장할 확률
- $\frac{P_{ik}}{P_{jk}}$: P_{ik}를 P_{jk}로 나눠준 값
  Ex) P(solid I ice) / P(solid I steam) = 8.9
- w_i : 중심 단어 i의 임베딩 벡터
- $\tilde{w}_k$: 주변 단어 k의 임베딩 벡터

$$Loss\ function = \sum_{m,n=1}^{V} f(X_{mn})(w_m^T \tilde{w}_n + b_m + \tilde{b_n} - logX_{mn})^2$$

09-05 글로브(GloVe)

```
from glove import Corpus, Glove

corpus = Corpus()

# 훈련 데이터로부터 GloVe에서 사용할 동시 등장 행렬 생성
corpus.fit(result, window=5)
glove = Glove(no_components=100, learning_rate=0.05)

# 학습에 이용할 쓰레드의 개수는 4로 설정, 에포크는 20.
glove.fit(corpus.matrix, epochs=20, no_threads=4, verbose=True)
glove.add_dictionary(corpus.dictionary)
```

```
print(glove.most_similar("university"))
```

```
[('harvard', 0.8690162017225468), ('cambridge', 0.8373272000675909), ('mit', 0.8288055170365777), ('stanford', 0.8212712738131419)]
```

09-06 패스트텍스트(FastText)

1. 내부 단어(subword)의 학습

```
# n = 3인 경우
<ap, app, ppl, ple, le>
```

```
# 특별 토큰
<apple>
```

```
# n = 3 ~ 6인 경우
<ap, app, ppl, ppl, le>, <app, appl, pple, ple>, <appl, pple>, ..., <apple>
```

```
apple = <ap + app + ppl + ppl + le> + <app + appl + pple + ple> + <appl + pple> + , ..., +<apple>
```

09-06 패스트텍스트(FastText)

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

데이터셋이 충분하다면 모르는 단어에 대해서도
다른 단어와의 유사도를 계산할 수 있음!

"birthplace"라는 단어를 학습하지 않았을때
다른 단어에서 "birth"와 "place"라는 작은 부분이 있었다면,
FastText는 "birthplace"의 벡터를 얻을 수 있음

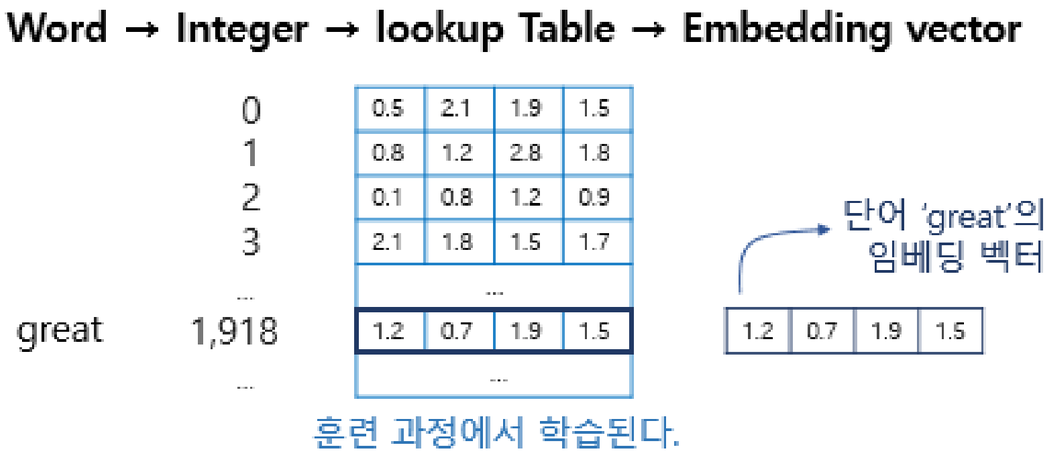
모르는 단어라도 비슷한 작은 부분을
가진 단어와의 유사도를 계산하여 적절한 처리

09-08 사전 훈련된 워드 임베딩 (Pre-trained Word Embedding)

1. 케라스 임베딩 층(Keras Embedding layer)

1) 임베딩 층은 룩업 테이블이다.

어떤 단어 → 단어에 부여된 고유한 정수값 → 임베딩 층 통과 → 밀집 벡터



09-08 사전 훈련된 워드 임베딩 (Pre-trained Word Embedding)

케라스 임베딩 층(Keras Embedding layer)

1) 임베딩 층은 룩업 테이블이다.

```
vocab_size = 20000
output_dim = 128
input_length = 500

v = Embedding(vocab_size, output_dim, input_length=input_length)
```

09-08 사전 훈련된 워드 임베딩 (Pre-trained Word Embedding)

케라스 임베딩 층(Keras Embedding layer)

2) 임베딩 층 사용하기

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

embedding_dim = 4

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

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['acc'])
model.fit(X_train, y_train, epochs=100, verbose=2)
```

09-09 엘모(Embeddings from Language Model, ELMo)

1. ELMo(Embeddings from Language Model)

문맥을 반영한 워드 임베딩 (Contextualized Word Embedding)

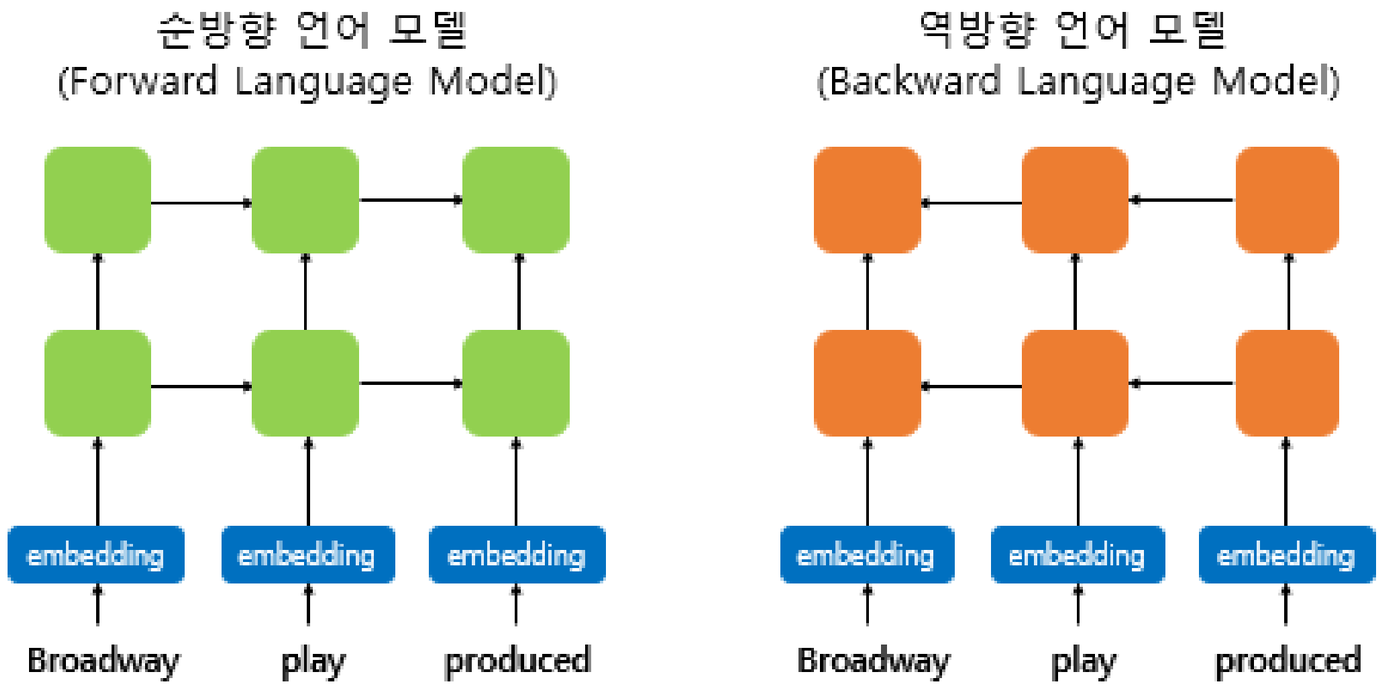
"Bank"

Bank Account(은행 계좌)

River Bank(강둑)

09-09 엘모(Embeddings from Language Model, ELMo)

2. biLM(Bidirectional Language Model)의 사전 훈련

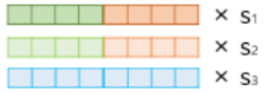


09-09 엘모(Embeddings from Language Model, ELMo)

3. biLM의 활용

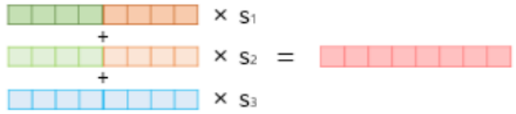
1) 각 층의 출력값을 연결(concatenate)한다.

2) 각 층의 출력값 별로 가중치를 준다.



이 가중치를 여기서는 s_1, s_2, s_3라고 합시다.

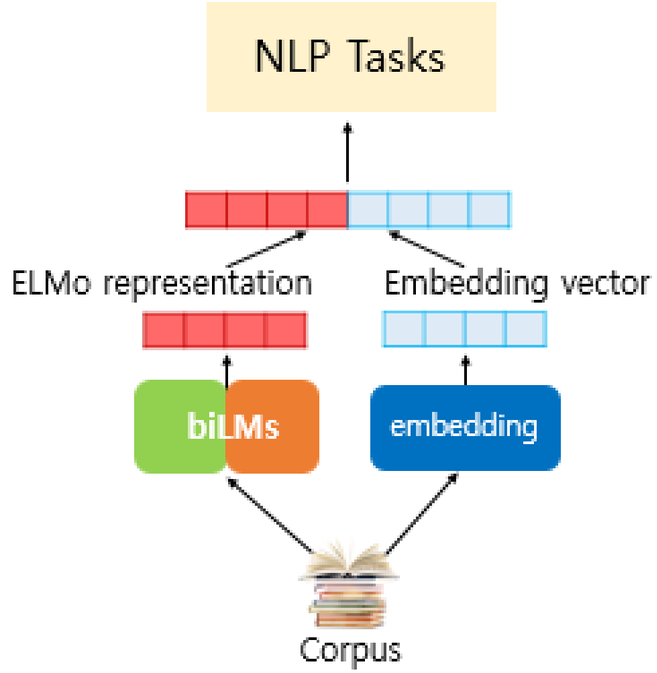
3) 각 층의 출력값을 모두 더한다.



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

4) 벡터의 크기를 결정하는 스칼라 매개변수를 곱한다.

이 스칼라 매개변수를 여기서는 γ이라고 합시다.



09-09 엘모(Embeddings from Language Model, ELMo)

4. ELMo 표현을 사용해서 스팸 메일 분류하기

```
def ELMoEmbedding(x):
    return elmo(tf.squeeze(tf.cast(x, tf.string)), as_dict=True, signature="default")["default"]
# 데이터의 이동이 케라스 → 텐서플로우 → 케라스가 되도록 하는 함수
```

```
from keras.models import Model
from keras.layers import Dense, Lambda, Input

input_text = Input(shape=(1,), dtype=tf.string)
embedding_layer = Lambda(ELMoEmbedding, output_shape=(1024, ))(input_text)
hidden_layer = Dense(256, activation='relu')(embedding_layer)
output_layer = Dense(1, activation='sigmoid')(hidden_layer)
model = Model(inputs=[input_text], outputs=output_layer)
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

09-09 엘모(Embeddings from Language Model, ELMo)

4. ELMo 표현을 사용해서 스팸 메일 분류하기

```
history = model.fit(X_train, y_train, epochs=1, batch_size=60)
```

```
Epoch 1/1
4457/4457 [==============================] - 1508s 338ms/step - loss: 0.1129 - acc: 0.9619
```

```
print("\n 테스트 정확도: %.4f" % (model.evaluate(X_test, y_test)[1]))
```

```
1115/1115 [==============================] - 381s 342ms/step
테스트 정확도: 0.9803
```

09-10 임베딩 벡터의 시각화 (Embedding Visualization)

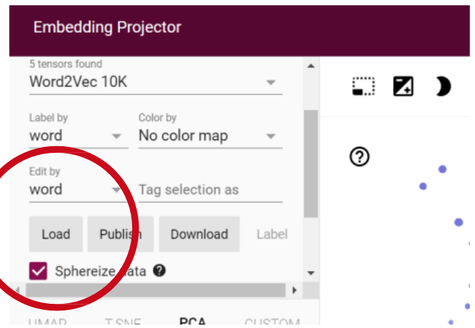
1. 워드 임베딩 모델로부터 2개의 tsv 파일 생성하기

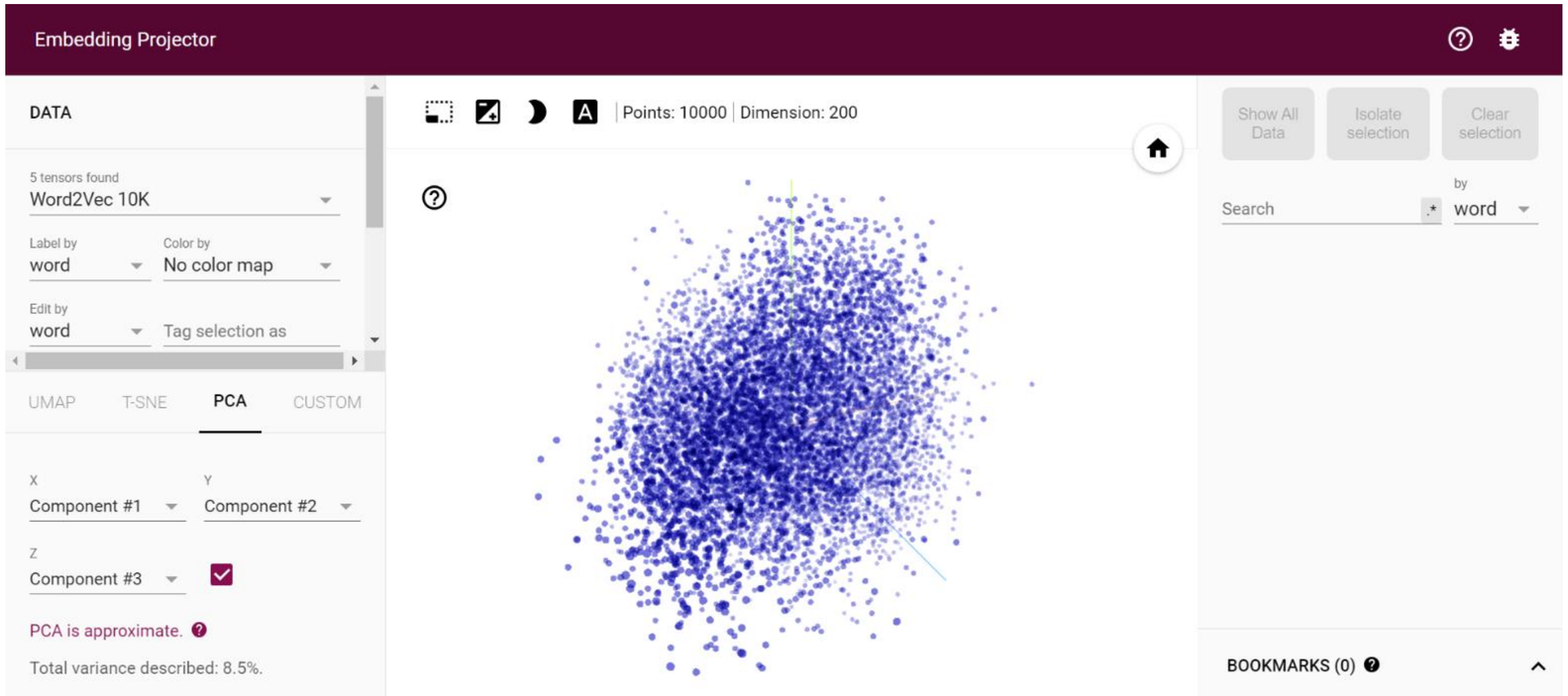
```
!python -m gensim.scripts.word2vec2tensor --input 모델이름 --output 모델이름
```

```
!python -m gensim.scripts.word2vec2tensor --input eng_w2v --output eng_w2v
```

09-10 임베딩 벡터의 시각화 (Embedding Visualization)

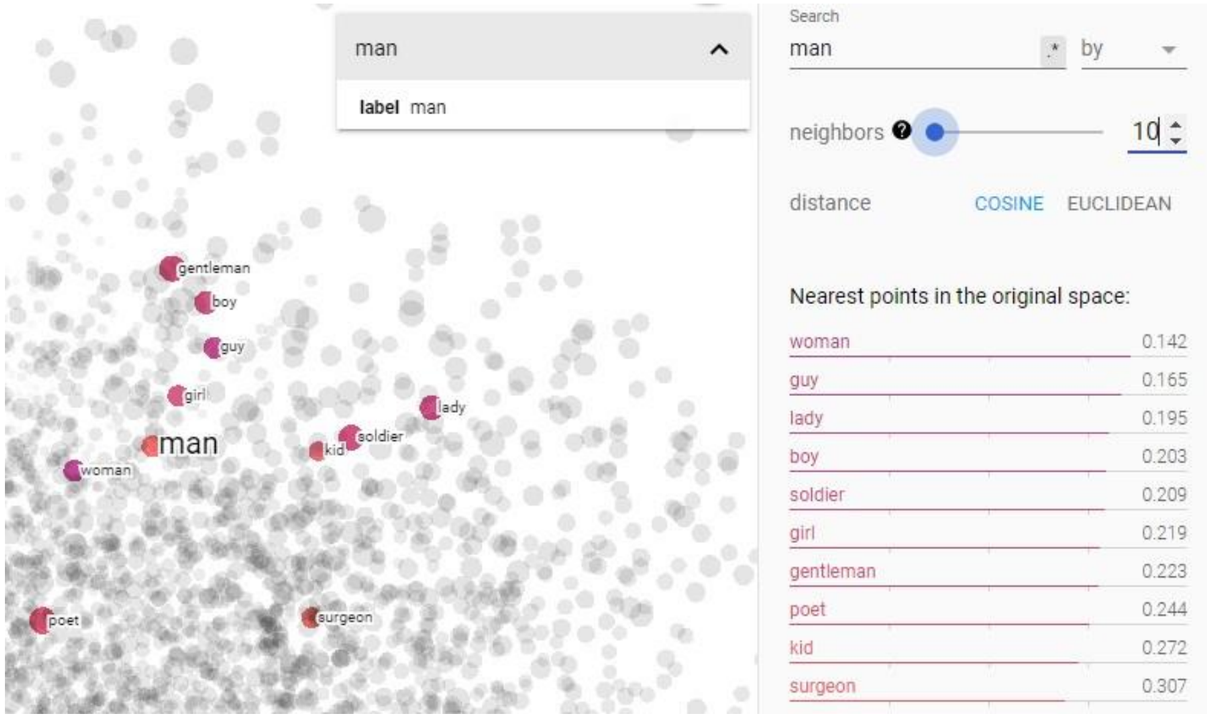
2. 임베딩 프로젝터를 사용하여 시각화하기





09-10 임베딩 벡터의 시각화 (Embedding Visualization)

2. 임베딩 프로젝터를 사용하여 시각화하기



09-11 문서 벡터를 이용한 추천 시스템 (Recommendation System using Document Embedding)

단어 벡터의 평균 구하기

```
def get_document_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
```

```
document_embedding_list = get_document_vectors(df['cleaned'])
print('문서 벡터의 수 :',len(document_embedding_list))
```

```
문서 벡터의 수 : 2381
```

09-11 문서 벡터를 이용한 추천 시스템 (Recommendation System using Document Embedding)

추천 시스템 구현하기

```
cosine_similarities = cosine_similarity(document_embedding_list, document_embedding_list)
print('코사인 유사도 매트릭스의 크기 :',cosine_similarities.shape)
```

```
코사인 유사도 매트릭스의 크기 : (2381, 2381)
```

```
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'])
```

09-11 문서 벡터를 이용한 추천 시스템 (Recommendation System using Document Embedding)

추천 시스템 구현하기

```
recommendations("The Da Vinci Code")
```

09-12 문서 임베딩 : 워드 임베딩의 평균

임베딩 벡터를 평균으로 사용하는 모델을 설계

```
from tensorflow.keras.models import Sequential, load_model
from tensorflow.keras.layers import Dense, Embedding, GlobalAveragePooling1D
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint

embedding_dim = 64

model = Sequential()
model.add(Embedding(vocab_size, embedding_dim))

# 모든 단어 벡터의 평균을 구한다.
model.add(GlobalAveragePooling1D())
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=T
rue)

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)
```

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

```
테스트 정확도: 0.8876
```