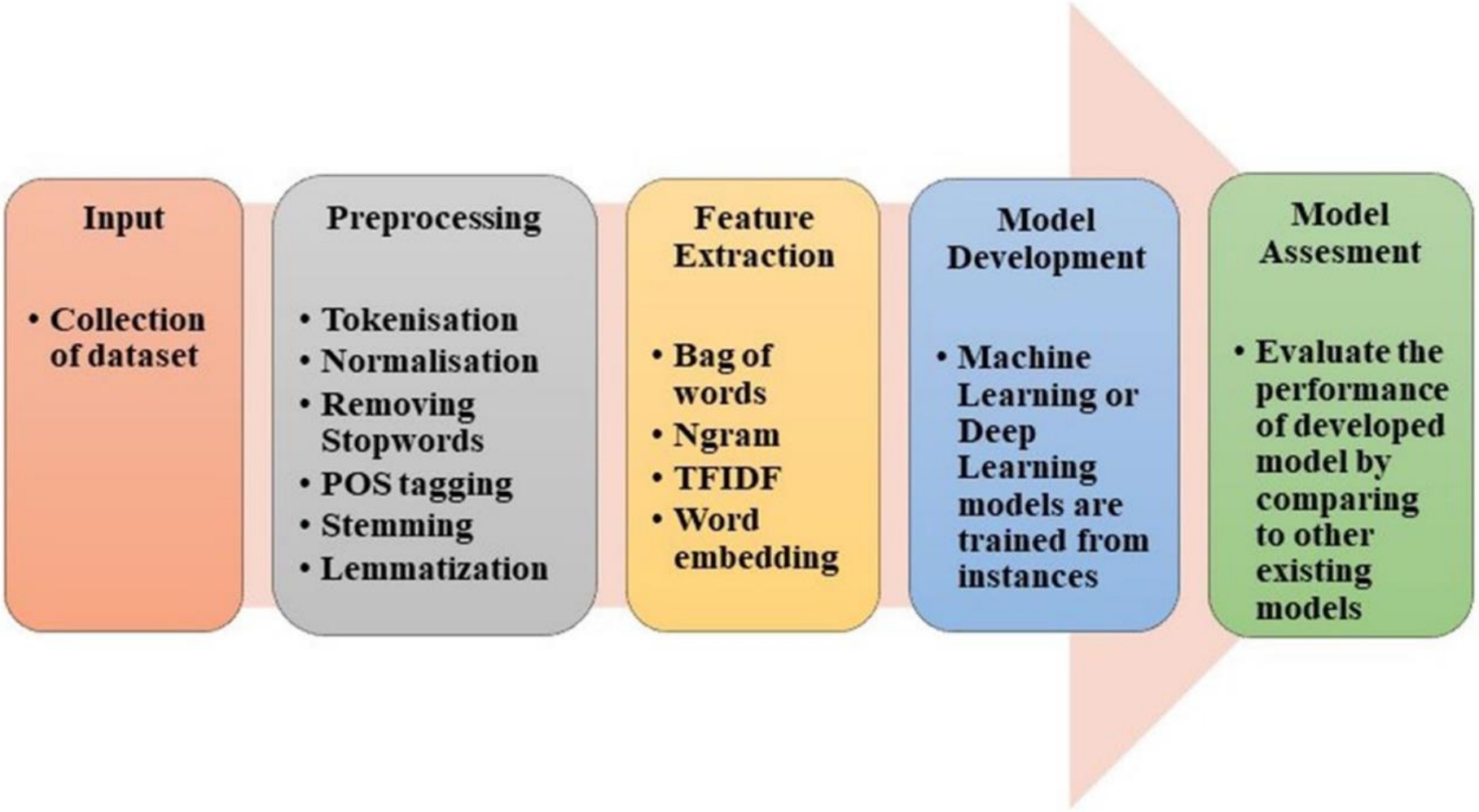
특징 추출 – 단어 표현

감성 분석 절차


Input
• Collection of dataset
Preprocessing
• Tokenisation
• Normalisation
• Removing Stopwords
• POS tagging
• Stemming
• Lemmatization
Feature Extraction
• Bag of words
• Ngram
• TFIDF
• Word embedding
Model Development
• Machine Learning or Deep Learning models are trained from instances
Model Assesment
• Evaluate the performance of developed model by comparing to other existing models

Feature Extraction – 단어 표현

- **텍스트를 컴퓨터가 이해하고, 효율적으로 처리하게 하기 위해서는 컴퓨터가 이해할 수 있도록 텍스트를 적절히 숫자로 변환해야 함**
  - 단어를 표현하는 방법에 따라 자연어 처리의 성능이 크게 달라짐
  - 단어를 수치화하여 벡터로 표현 → 특징 벡터 (Feature Vector)

- **단어의 특징 벡터 추출 방법**
  - 카운트 기반 벡터화
  - Word Embedding

카운트 기반 벡터화

- **단어의 빈도수에 기반하여 단어의 특징 벡터를 만드는 방법**

- **정보 검색이나 텍스트 마이닝 분야에서 주로 사용**
  - 통계적인 접근 방법을 통해 여러 문서로 이루어진 텍스트 데이터가 있을 때
    - 어떤 단어가 특정 문서 내에서 얼마나 중요한 것인지를 나타내거나,
    - 문서의 핵심어 추출,
    - 검색 엔진에서 검색 결과의 순위 결정,
    - 문서들 간의 유사도를 구하는 등의 용도

    로 사용할 수 있음

- **카운트 기반 벡터화 방법**
  - BoW (Bag of Words)
  - DTM (Document-Term Matrix, 문서 단어 행렬)
  - TF-IDF (Term Frequency – Inverse Document Frequency)
  - 정수 인코딩 (Integer Encoding)

BOW (Bag of Word)

- **단어들의 순서는 전혀 고려하지 않고, 단어들의 출현 빈도(frequency)에만 집중하는 텍스트 데이터의 수치화 표현 방법**
  - 단어 출현 빈도가 높을수록 중요한 단어로 다루어짐
  - BoW 생성 방법
    - (1) 각 단어에 고유한 정수 인덱스를 부여 # 단어 집합 생성.
    - (2) 각 인덱스의 위치에 단어 토큰의 등장 횟수를 기록한 벡터 생성
  - 사이킷런의 CountVectorizer 클래스로 BoW 생성
    - from sklearn.feature_extraction.text import CountVectorizer
    - vector = CountVectorizer()
    - vector.fit_transform(corpus)

DTM (Document-Term Matrix)

- **다수의 문서에서 등장하는 각 단어들의 빈도를 행렬로 표현한 것**
  - 문서별 단어의 빈도를 정리
  - 문서 단어 행렬(DTM, Document-Term matrix)을 구성
    - 문서 d에 등장한 단어 t의 횟수는 tf(t,d)로 표현

	그래서	데이터	분석	…	이다	한다
doc#1	13	20	16	…	65	71
doc#2	11	15	32	…	69	81

그림 13-1 카운트 기반 벡터화의 DTM 예: tf("데이터", doc#1) = 20

  - BoW와 다른 표현 방법이 아니라 BoW 표현을 다수의 문서에 대해서 행렬로 표현하고 부르는 용어

TF-IDF 기반 벡터화

- Term Frequency – Inverse Document Frequency
  - 특정 문서에 많이 나타나는 단어는 해당 문서의 단어 벡터에 가중치를 높임
  - 모든 문서에 많이 나타나는 단어는 범용적으로 사용하는 단어로 취급하여 가중치를 낮추는 방식

- d에 등장한 단어 t의 TF-IDF

$$tf-idf(t, d) = tf(t,d) \times idf(t,d)$$

  - (역 문서 빈도)idf(t,d)
  - 는 : 전체 문서의 개수
  - df(d,t)는 단어 t가 포함된 문서 d의 개수

$$idf(t, d) = log\frac{n_d}{1+df(d, t)}$$

	그래서	데이터	분석	…	이다	한다
doc#1	0.12	0.52	0.42	…	0.19	0.20
doc#2	0.13	0.48	0.67	…	0.18	0.22

그림 13-2 TF-IDF 기반 벡터화의 DTM 예: tf-idf("데이터", doc#1) = 0.52

N-gram

- n개의 연속적인 단어 나열을 의미
- 코퍼스에서 n개의 단어 뭉치 단위로 끊어서 이를 하나의 토큰으로 간주함
- An adorable little boy is spreading smiles
  - unigrams : an, adorable, little, boy, is, spreading, smiles
  - bigrams : an adorable, adorable little, little boy, boy is, is spreading, spreading smiles
  - trigrams : an adorable little, adorable little boy, little boy is, boy is spreading, is spreading smiles

Integer Encoding

- **빈도수가 높은 순서대로 차례로 낮은 숫자부터 정수를 부여하여 특징 벡터 생성**
  - 데이터 셋의 단어에 대한 빈도수에 기반하여 빈도수가 높은 순서부터 indexing
  - 문장의 단어를 해당 index로 변환하여 특징 벡터 생성
  - 단어 수만큼 차원이 높아지므로, 빈도수가 높은 단어만 사용
  - 저빈도로 index가 할당되지 않는 단어들은 index에 포함된 단어 수 + 1 할당

- **Keras의 Integer Enoding**
  - from tensorflow.keras.preprocessing.text import Tokenizer
  - tokenizer = Tokenizer(num_words = vocab_size + 1)
  - tokenizer.fit_on_texts(dataset)
  - Tokenizer.texts_to_sequence(dataset)

패딩(Padding)

- 문서들의 feature vector를 행렬로 처리하기 위하여 각 문장의 feature vector의 차원을 맞추어 주는 작업
  - 길이가 짧은 문장은 숫자 0을 채워서 길이를 맞춤

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

- Keras 전처리 도구로 패딩하기
  - from tensorflow.keras.preprocessing.sequence import pad_sequences
  - pad_sequence(encoded_set, maxlen=)

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

워드 임베딩 (Word Embedding)

- **단어를 밀집 벡터(dense vector)의 형태로 표현하는 방법**
  - 인공 신경망 학습을 통해서 단어를 벡터화 –〉 임베딩 벡터(embedding vector)
  - 단어의 의미를 여러 차원에 분산
  - Word2Vec, FastText, Glove, Elmo 등

- **원-한 인코딩 (one-hot encoding)**
  - 단어에 해당하는 index 값만 1이고, 나머지는 0으로 표현되는 벡터 –〉 희소 벡터 (sparse vector)
  - 단어 개수에 따라 벡터의 차원이 커짐
  - 단어의 의미를 표현하지 못함

	원-핫 벡터	임베딩 벡터
차원	고차원(단어 집합의 크기)	저차원
다른 표현	희소 벡터의 일종	밀집 벡터의 일종
표현 방법	수동	훈련 데이터로부터 학습함
값의 타입	1과 0	실수

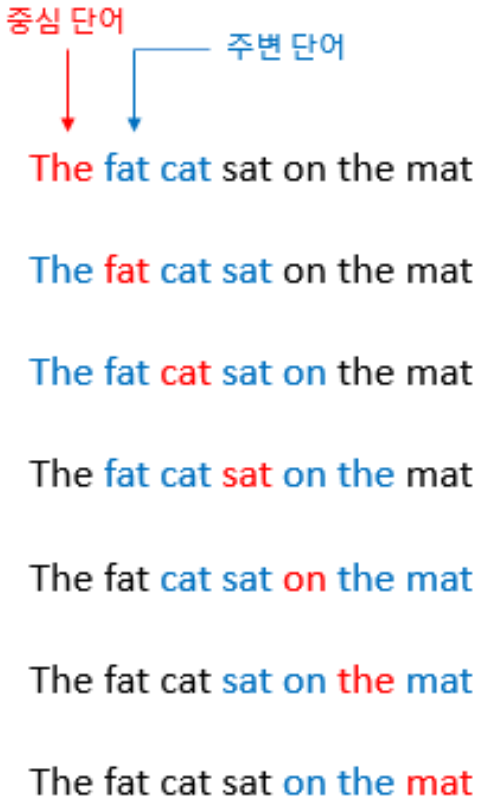
Word2Vec

- **각 단어 벡터가 단어 간 유사도를 반영한 값을 갖도록 학습**
  - 위치가 근접한 데이터를 유사도가 높은 벡터를 만들어준다는 점에서 착안된 아이디어
    - Work2Vec 단어 연산 : http://w.elnn.kr/search/
      - 한국 – 서울 + 도쿄 = 일본
      - 박찬호 – 야구 + 축구 = 호나우두
  - 임베딩 벡터가 윈도우 크기 내에서만 주변 단어를 고려하기 때문에 코퍼스의 전체적인 통계 정보를 반영하지 못한다는 단점

- **학습 방법**
  - CBOW (Continuous Bag of Words)
    - 주변에 있는 단어(context word)들을 입력으로 중간에 있는 단어(center word)들을 예측하는 방법
  - Skip-Gram
    - 중간에 있는 단어(center word)들을 입력으로 주변 단어(context word)들을 예측하는 방법

CBOW (Continuous Bag of Words)

- **주변에 있는 단어(context word)들을 입력으로 중간에 있는 단어(center word)들을 예측하는 방법**
  - 슬라이딩 윈도우 (sliding window)를 통해 학습 데이터 셋 생성
    - 윈도우(window) : 주변 단어 범위를 지정, 중심 단어 앞과 뒤 각각의 수
  - 예시 : window = 2



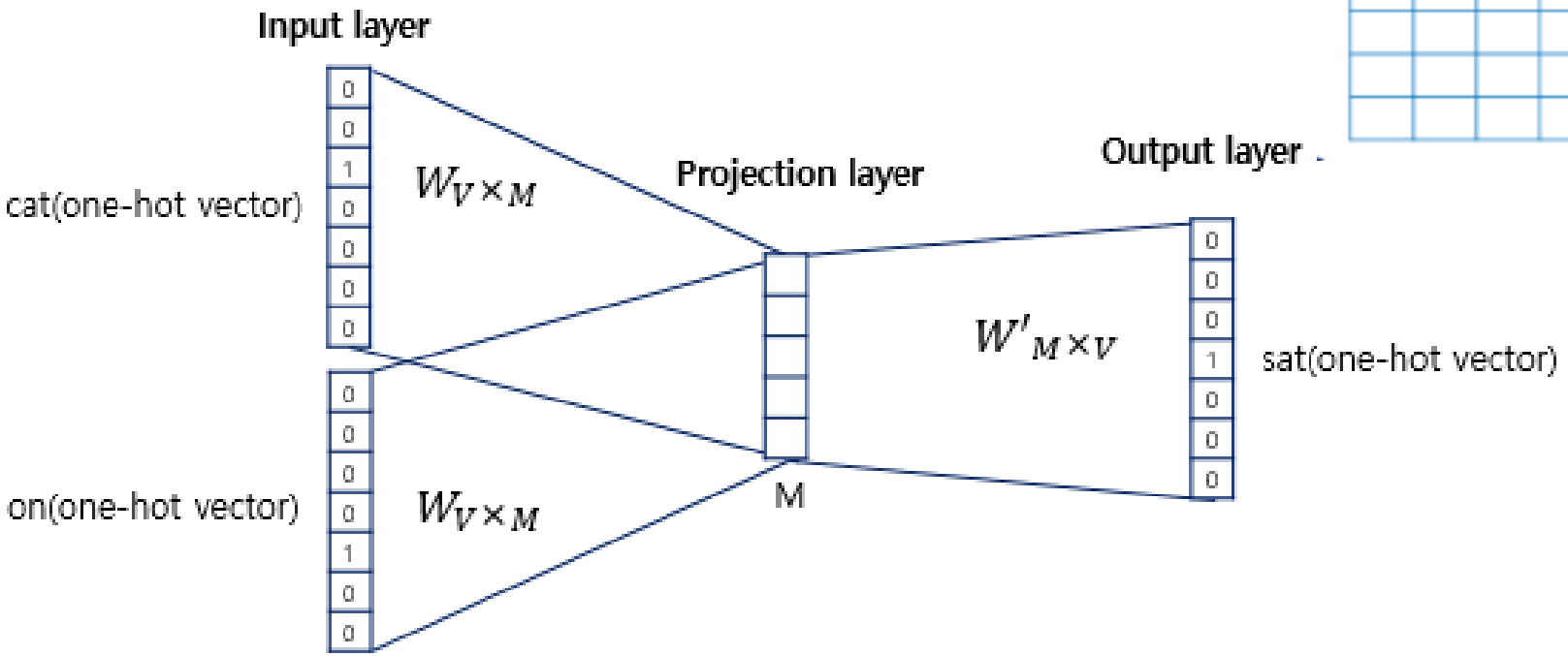
중심 단어	주변 단어
[1, 0, 0, 0, 0, 0, 0]	[0, 1, 0, 0, 0, 0, 0], [0, 0, 1, 0, 0, 0, 0]
[0, 1, 0, 0, 0, 0, 0]	[1, 0, 0, 0, 0, 0, 0], [0, 0, 1, 0, 0, 0, 0], [0, 0, 0, 1, 0, 0, 0]
[0, 0, 1, 0, 0, 0, 0]	[1, 0, 0, 0, 0, 0, 0], [0, 1, 0, 0, 0, 0, 0], [0, 0, 0, 1, 0, 0, 0], [0, 0, 0, 0, 1, 0, 0]
[0, 0, 0, 1, 0, 0,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

CBOW (Continuous Bag of Words)

- 입력값과 출력값은 one-hot-vector
- 은닉층이 1개인 shallow neural network 모델 -> 가중치 W, W' 학습
- 학습 후의 W의 각 행백터가 각 단어의 M차원의 임베딩 벡터가 됨

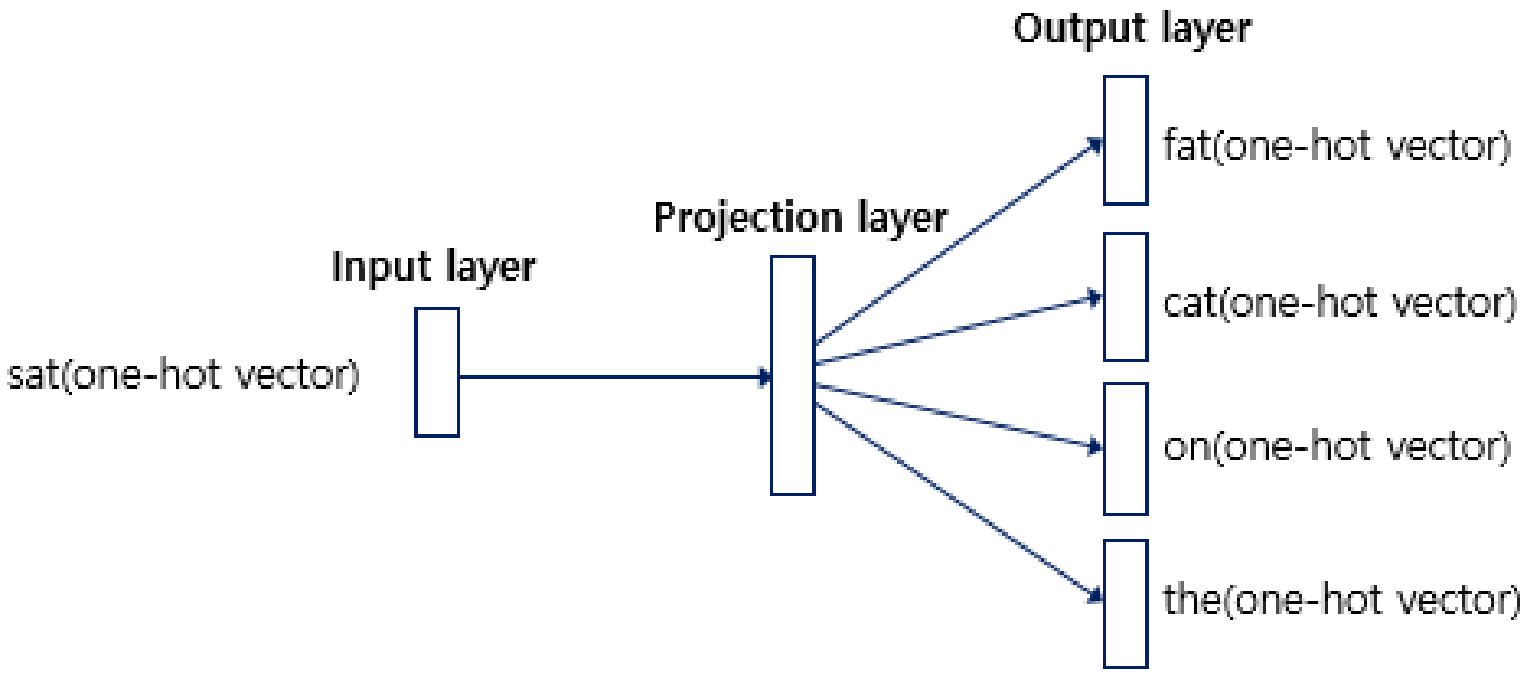
$W_{V\times M}$

0.5	2.1	1.9	1.5	0.8
0.8	1.2	2.8	1.8	2.1
2.1	1.8	1.5	1.7	2.7



Skip-gram

- 중간에 있는 단어(center word)들을 입력으로 주변 단어(context word)들을 예측하는 방법
  - 전반적으로 Skip-gram이 CBOW보다 성능이 좋다고 알려짐
  - 예시: The fat cat sat on the mat



사전 훈련된 워드 임베딩

- **이미 훈련되어져 있는 워드 임베딩을 가져와서 이를 임베딩 벡터로 사**
  - 훈련 데이터가 적은 상황인 경우 해당 문제에 특화된 것은 아니지만 보다 많은 훈련 데이터로 이미 Word2Vec이나 GloVe 등으로 학습되어져 있는 임베딩 벡터들을 사용하는 것이 성능의 개선을 가져올 수 있음

  - GloVe 다운로드 링크 : http://nlp.stanford.edu/data/glove.6B.zip
    - 파일의 embedding vector를 읽어서 저장
  - Word2Vec 다운로드 링크 : https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM
    - gensim.models.KeyedVectors.load_word2vec_format('GoogleNews-vectors-negative300.bin.gz', binary=True)

케라스 임베딩 층(Keras Embedding layer)

- **훈련 데이터의 단어들에 대해 워드 임베딩을 수행하는 도구 Embedding()을 제공**
  - Embedding()은 인공 신경망 구조 관점에서 임베딩 층(embedding layer)을 구현
  - v = Embedding(vocab_size, output_dim, input_length=input_length)