단어 임베딩(word embedding) 이해하기

학습 내용

- 단어 임베딩 기본 개념 이해
- IMDB 영화 리뷰 데이터 셋을 활용하여 단어 임베딩 실습해 보기

목차

01. 단어 임베딩(word embedding)의 이해

목차로 이동하기

단어 임베딩 이해하기

- 단어와 벡터를 연관짓는 강력하고 인기 있는 또 다른 방법 중의 하나는 단어 임베딩이다.
  - 단어 임베딩은 **밀집 단어 벡터를 사용**하는 것.
  - 단어 임베딩은 언어를 **기하학적 공간에 매핑**하는 것.
  - 잘 구축된 임베딩 공간에서는 동의어가 **비슷한 단어 벡터로 임베딩**된다.

Onehot vs 단어 임베딩 비교

- A. 고차원 저차원
  - 원핫 인코딩 기법으로 만든 벡터는 대부분 0으로 채워지는 고차원이다.
  - 단어 임베딩은 저차원 기법이다.
  - 원핫인코딩은 단어 사전이 2만개의 토큰으로 이루어져 있다면 20,000차원의 벡터를 사용
  - 보통 단어 임베딩은 **256, 512, 1024차원의 단어 임베딩**을 사용.
- B. 원-핫 인코딩이나 해싱으로 얻은 단어 표현은 희소하고 고차원이며, 수동으로 인코딩된다.
- C. 단어 임베딩은 조밀하고 **비교적 저차원이며 데이터로부터 학습**

02. 단어 임베딩 만드는 두가지 방법

목차로 이동하기

- 신경망을 구축하고 학습하는 방법처럼 **단어 벡터를 학습**하기
- **사전에 훈련**된 단어 임베딩을 로드하기(pretrained word embedding)

- 구글의 토마스 미코로프 word2vec 알고리즘(https://code.google.com/archive/p/word2vec (https://code.google.com/archive/p/word2vec))
- 스탠포드 대학교 : GloVe(https://nlp.stanford.edu/projects/glove (https://nlp.stanford.edu/projects/glove))

2-1 Embedding 층을 사용하여 단어 임베딩 학습하기

- 단어와 밀집 벡터를 연관 짓는 가장 간단한 방법은 **랜덤하게 벡터를 선택**하는 것.
- 이 방식의 문제점은 임베딩 공간이 구조적이지 않다는 것.
  - 예를 들어, accurate와 exact단어는 대부분 문장에서 비슷한 의미로 사용. 단, 다른 임베딩을 갖는다.
  - 심층 신경망이 이런 임의의 구조적이지 않은 임베딩 공간을 이해하기는 어렵다.

03. 해결 제안

목차로 이동하기

- 문제는 **사람의 언어를 완전히 매핑**시킬 수 있는 **이상적인 단어 임베딩 공간**을 만드는 것이다. 이런 공간이 있을까?

- 아마도 가능하겠지만, 완벽하게는 어려울 수 있다.
- 다만, 최근에는 많은 발전을 이루었다.

04. IMDB 데이터를 이용한 Embedding 실습

목차로 이동하기

- Embedding 층(특정 단어를 나타내는) 정수 인덱스를 밀집 벡터로 매핑하는 딕셔너리로 이해
- 정수를 입력받아, 내부 딕셔너리에서 **이 정수와 연관된 벡터를 찾아 반환**

In [1]:

```
from keras.layers import Embedding

# Embedding 층은 적어도 두 개의 매개변수를 사용.
# 가능한 토큰의 개수(여기서는 1,000으로 단어 인덱스 최댓값 + 1입니다)와
# 출력되는 임베딩 차원(여기서는 64)입니다
embedding_layer = Embedding(1000, 64)

print( type(embedding_layer), embedding_layer)
```

```
<class 'keras.layers.core.embedding.Embedding'> <keras.layers.core.embedding.Embeddi
ng object at 0x0000026039F4D520>
```

Embedding층의 입력

- (samples, 임베딩차원, sequence_length)

    - samples : 샘플수
    - sequence_length : 시퀀스 길이 (단순히 길이)
    - 정수 텐서를 입력으로 받음. 2D텐서
- 여기서 sequence_length가 작은 길이의 시퀀스는 **0으로 패딩**되고, **긴 시퀀스는 잘리게** 됩니다.

Embedding층의 출력

- (samples, sqeuence_length, 임베딩 차원)
    - samples : 샘플수
    - sequence_length : 시퀀스 길이
    - embedding_dimensionality : 임베딩 차원
    - 출력은 3D 텐서가 된다.

Embedding층의 객체

- 가중치는 다른 층과 마찬가지로 랜덤하게 초기화
- **신경망의 학습을 통해 점차 조정**되어진다.
    - 훈련이 끝나면 임베딩 공간은 **특정 문제에 특화된 구조**를 많이 갖는다.

- IMDB 데이터 셋
    - 총 5만개로 이루어진 긍정 부정의 리뷰
    - 이 데이터셋은 훈련 데이터 25,000개와 테스트 데이터 25,000개로 나뉘어 있고 각각 50%는 부정, 50%는 긍정 리뷰로 구성
    - 스탠포드 대학의 앤드류 마스(Andrew Maas)가 수집한 데이터 셋
- train_data는 여러개의 단어로 이루어진 리뷰. 리뷰 단어는 각각 매칭된 word index 값으로 이루어짐.
- train_labels는 1(긍정), 0(부정)이 됨.

In [10]:

```
from keras.datasets import imdb
from tensorflow.keras import preprocessing
```

In [11]:

```
# 특성으로 사용할 단어의 수
max_features = 10000

# 정수 리스트로 데이터를 로드합니다.
(X_train, y_train), (X_test, y_test) = imdb.load_data(num_words=max_features)
```

In [12]:

```
X_train.shape, y_train.shape, X_test.shape, y_test.shape
```

Out[12]:

```
((25000,), (25000,), (25000,), (25000,))
```

In [13]:

```
print(X_train[0])
print(y_train[0])
```

```
[1, 14, 22, 16, 43, 530, 973, 1622, 1385, 65, 458, 4468, 66, 3941, 4, 173, 36, 256,
5, 25, 100, 43, 838, 112, 50, 670, 2, 9, 35, 480, 284, 5, 150, 4, 172, 112, 167, 2,
336, 385, 39, 4, 172, 4536, 1111, 17, 546, 38, 13, 447, 4, 192, 50, 16, 6, 147, 202
5, 19, 14, 22, 4, 1920, 4613, 469, 4, 22, 71, 87, 12, 16, 43, 530, 38, 76, 15, 13, 1
247, 4, 22, 17, 515, 17, 12, 16, 626, 18, 2, 5, 62, 386, 12, 8, 316, 8, 106, 5, 4, 2
223, 5244, 16, 480, 66, 3785, 33, 4, 130, 12, 16, 38, 619, 5, 25, 124, 51, 36, 135,
48, 25, 1415, 33, 6, 22, 12, 215, 28, 77, 52, 5, 14, 407, 16, 82, 2, 8, 4, 107, 117,
5952, 15, 256, 4, 2, 7, 3766, 5, 723, 36, 71, 43, 530, 476, 26, 400, 317, 46, 7, 4,
2, 1029, 13, 104, 88, 4, 381, 15, 297, 98, 32, 2071, 56, 26, 141, 6, 194, 7486, 18,
4, 226, 22, 21, 134, 476, 26, 480, 5, 144, 30, 5535, 18, 51, 36, 28, 224, 92, 25, 10
4, 4, 226, 65, 16, 38, 1334, 88, 12, 16, 283, 5, 16, 4472, 113, 103, 32, 15, 16, 534
5, 19, 178, 32]
1
```

In [14]:

```
#  리뷰의 길이와 10개 단어(인덱스) 보기
len(X_train[0]), X_train[0][0:10]   # 단어가 218개 단어로 구성
```

Out[14]:

```
(218, [1, 14, 22, 16, 43, 530, 973, 1622, 1385, 65])
```

다양한 길이가 리뷰가 있을 것이다.

- 리뷰에서 맨 마지막 50개 단어를 얻고, 나머지는 버린다. 또는 길이가 짧다면 0으로 채운다.
- 시퀀스의 길이가 50개로 한다.
- 리스트 형태의 리뷰를 2D 정수 텐서로 변환 : preprocessing.sequence.pad_sequences

텍스트 데이터 전처리

In [15]:

```
# 리스트를 (samples, maxlen) 크기의 2D 정수 텐서로 변환합니다.
maxlen = 50

X_train_n = preprocessing.sequence.pad_sequences(X_train, maxlen=maxlen)
X_test_n = preprocessing.sequence.pad_sequences(X_test, maxlen=maxlen)
```

In [16]:

```
print("변경 전 : " , X_train.shape, X_test.shape)
print("변경 후 : ", X_train_n.shape, X_test_n.shape)
```

```
변경 전 :  (25000,) (25000,)
변경 후 :  (25000, 50) (25000, 50)
```

왜 1D 텐서를 2D텐서로 변경하는가?

- Embedding() 층은 다음과 같이 입력(2D텐서)을 받는다.
- (samples, sequence_length)

In [17]:

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

- Embedding층의 출력
  - (samples, sqeuence_length, embedding_dimensionality)
  - (단어 인덱스, 시퀀스 길이, 임베딩 차원-출력)

모델 구축

In [18]:

```
maxlen
```

Out[18]:

50

In [19]:

```
model = Sequential()

# 나중에 임베딩된 입력을
# Flatten 층에서 펼치기 위해 Embedding 층에 input_length를 지정
# Embedding 층의 입력은 2D (samples, maxlen) 이다.
model.add(Embedding(10000, 8, input_length=maxlen))

# Embedding 층의 출력 크기는 (samples, maxlen, 8)가 됩니다.
```

In [20]:

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_1 (Embedding)     (None, 50, 8)             80000

=================================================================
Total params: 80,000
Trainable params: 80,000
Non-trainable params: 0
_________________________________________________________________
```

- 최대 길이 50개의 단어를 학습을 통해 8차원 임베딩 공간을 만든다는 것이다.

In [21]:

```
model.add(Flatten())
model.add(Dense(1, activation='sigmoid'))

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_1 (Embedding)     (None, 50, 8)             80000

 flatten (Flatten)           (None, 400)               0

 dense (Dense)               (None, 1)                 401

=================================================================
Total params: 80,401
Trainable params: 80,401
Non-trainable params: 0
_________________________________________________________________
```

In [22]:

```
%%time

# 분류기를 추가합니다.
model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['acc'])

history = model.fit(X_train_n, y_train,
                    epochs=10,
                    batch_size=32,
                    validation_split=0.2)
```

```
Epoch 1/10
625/625 [==============================] - 3s 3ms/step - loss: 0.6571 - acc: 0.6413
- val_loss: 0.5631 - val_acc: 0.7550
Epoch 2/10
625/625 [==============================] - 1s 2ms/step - loss: 0.4599 - acc: 0.8023
- val_loss: 0.4326 - val_acc: 0.8004
Epoch 3/10
625/625 [==============================] - 1s 2ms/step - loss: 0.3660 - acc: 0.8421
- val_loss: 0.4038 - val_acc: 0.8110
Epoch 4/10
625/625 [==============================] - 1s 2ms/step - loss: 0.3242 - acc: 0.8618
- val_loss: 0.3956 - val_acc: 0.8170
Epoch 5/10
625/625 [==============================] - 1s 2ms/step - loss: 0.2972 - acc: 0.8762
- val_loss: 0.3955 - val_acc: 0.8156
Epoch 6/10
625/625 [==============================] - 1s 2ms/step - loss: 0.2762 - acc: 0.8863
- val_loss: 0.4008 - val_acc: 0.8130
Epoch 7/10
625/625 [==============================] - 1s 2ms/step - loss: 0.2581 - acc: 0.8952
- val_loss: 0.4065 - val_acc: 0.8162
Epoch 8/10
625/625 [==============================] - 1s 2ms/step - loss: 0.2412 - acc: 0.9037
- val_loss: 0.4124 - val_acc: 0.8130
Epoch 9/10
625/625 [==============================] - 1s 2ms/step - loss: 0.2251 - acc: 0.9111
- val_loss: 0.4190 - val_acc: 0.8110
Epoch 10/10
625/625 [==============================] - 1s 2ms/step - loss: 0.2086 - acc: 0.9190
- val_loss: 0.4286 - val_acc: 0.8100
CPU times: total: 24.2 s
Wall time: 16 s
```

In [24]:

```
model.evaluate(X_test_n, y_test)
```

```
782/782 [==============================] - 2s 2ms/step - loss: 0.4256 - acc: 0.8188
```

Out[24]:

```
[0.4255502223968506, 0.8187599778175354]
```

- 학습 후, 테스트 데이터 평가 결과 정확도가 약 81%입니다.
- 리뷰 50개의 단어(시퀀스 길이)만 사용하여 좋은 결과를 얻었습니다.

- 임베딩 층을 펼쳐 하나의 Dense 층을 이용하여 학습수행. 입력 시퀀스 있는 각 단어를 독립적으로 다루었습니다.
- 단어 사이의 **관계나 문장 구조를 고려하지 않음**.
  - 해결책 : 각 시퀀스 전체를 고려한 특성이 학습되도록 **임베딩 층 위에 순환 층**이나 **1D 합성곱 층을 추가**하는 것이 좋다.

정리

- 토큰(단어 등)를 벡터로 변환하는 방법
  - 원핫 인코딩
  - 원-핫 해싱 : : 각 단어에 명시적으로 인덱스를 할당. 임의의 사이즈에 데이터를 매핑.
  - 단어 임베딩 사용
    - 케라스에서 Embedding을 이용하여 **일정단어를 일정 차원의 수로 단어를 벡터화** 시킨다.
    - 여기서의 **가중치는 학습**을 통해 **특정 데이터에 특정된 임베딩 공간**이 만들어진다.