Word Embedding - Word2Vec

for NLP (Natural Language Processing: 자연어처리)

Word representation

단어집(Vocabularies) = [a, aaron, …, zuru, <UNK>] e.g. |V| = 10,000

- 신경망에 **텍스트**를 입력 할 수 있습니까? ⇒ NO !
- 신경망에 **숫자**를 입력 할 수 있습니까? ⇒ YES !
- **One-Hot-Encoding** 방법 : 모두에 0 을, 단어 위치에만 1 을 할당한 벡터 사용

Man (5391) O_{5391} $\begin{bmatrix}0\\0\\0\\.\\0\\1\\0\\.\\0\\0\end{bmatrix}$

Woman (9853) O_{9853} $\begin{bmatrix}0\\0\\0\\0\\0\\.\\0\\1\\.\\0\end{bmatrix}$

King (4914) O_{4914} $\begin{bmatrix}0\\0\\0\\1\\0\\.\\.\\.\\.\\0\end{bmatrix}$

Queen (7157) O_{7157} $\begin{bmatrix}0\\0\\0\\0\\0\\.\\.\\1\\.\\0\end{bmatrix}$

Apple (456) O_{456} $\begin{bmatrix}0\\.\\1\\.\\0\\0\\0\\0\\0\\0\end{bmatrix}$

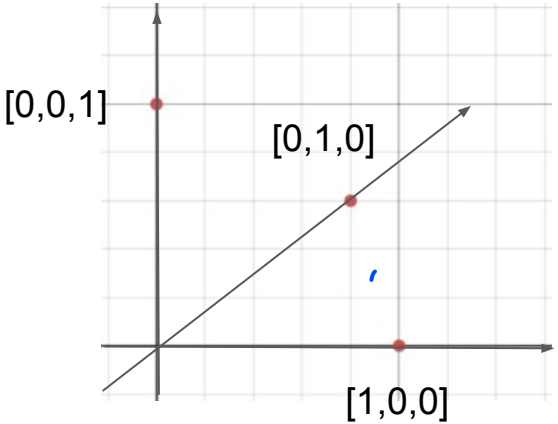
Orange (6257) O_{6257} $\begin{bmatrix}0\\0\\0\\.\\0\\0\\1\\0\\.\\0\end{bmatrix}$

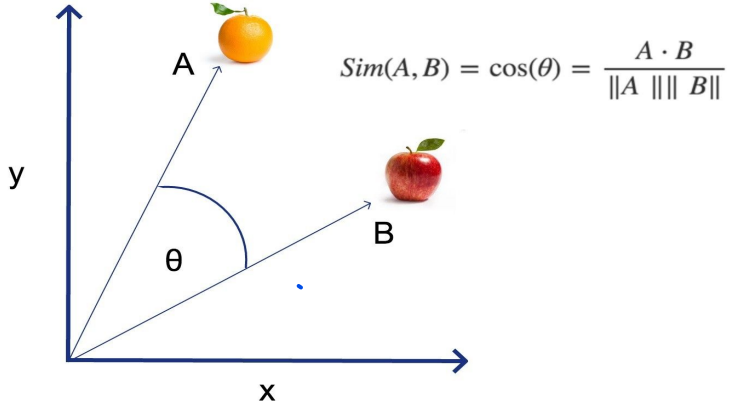
Word representation

- 문장관련 응용에서는 **단어의 유사성 정보**를 포함할 필요가 있다.

  I want a glass of orange _ juice _ . ⇒ I want a glass of apple __ ??? __ .

- One-Hot-Encoding 에는 **유사성(similarity)** 정보가 없다. ⇒ **단점**
  - 부호화 된 **벡터들 간의 거리가 모두 같다.**
  - 부호화 된 벡터들 간의 각도가 90° 여서, **코사인(cos) 유사도** 또한 0 이다.





One-Hot-Encoding 의 대안

word-embedding : 특징적인 표현(featurized representation)

	Man (5391)	Woman (9853)	King (4914)	Queen (7157)	Apple (456)	Orange (6257)
Gender	-1	1	-0.95	0.97	0.00	0.01
Royal	0.01	0.02	0.93	0.95	-0.01	0.00
Age	0.03	0.02	0.7	0.69	0.03	-0.02
Food	0.04	0.01	0.02	0.01	0.95	0.97
...						
Size,Cost, ...						
	e_{5391}	e_{9853}	e_{4914}	e_{7157}	e_{456}	e_{6257}

단어수 > 300

특징들의 가중치 값

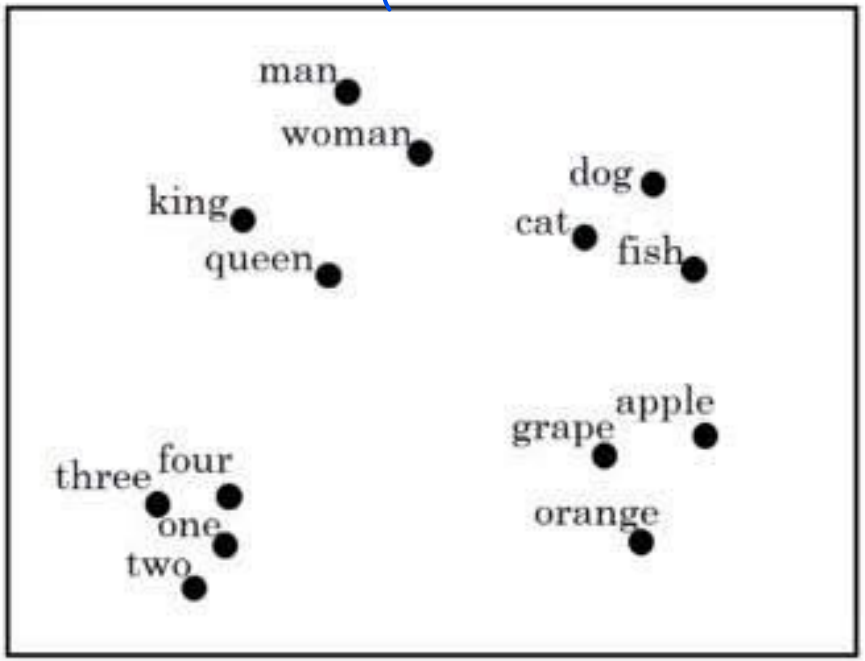
Visualizing word-embedding using T-SNE (다차원 자료 시각화 알고리즘)

시각화를 할 수 있게 해줌

n-D (eg. 300-D)

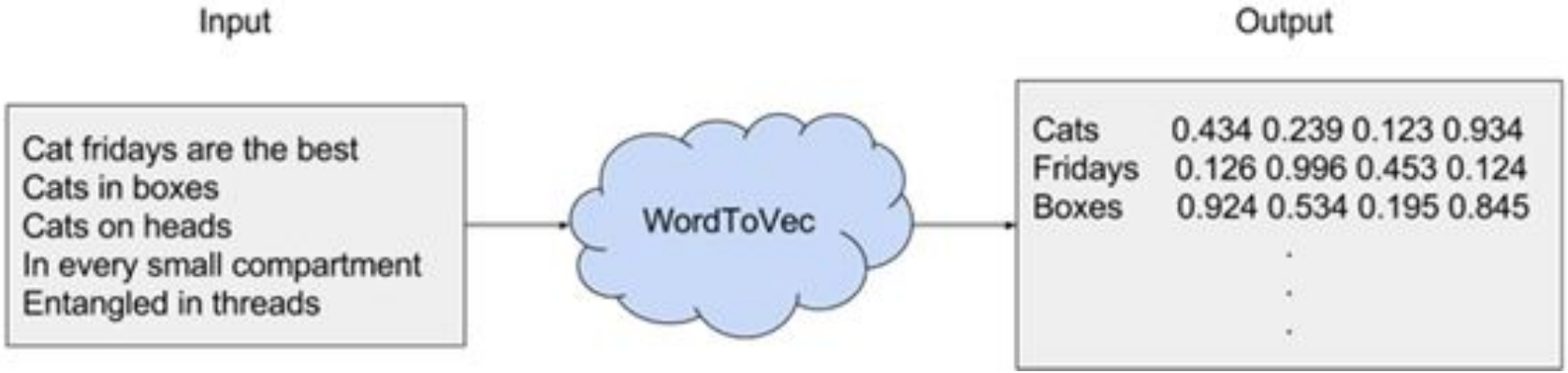
⇩

2-D



Word embeddings : Word2Vec 방법 1

Word2Vec 는 아래 그림과 같이 텍스트 코퍼스(corpus, 말뭉치)를 입력으로 받아들이고 각 단어에 대한 벡터 표현을 출력하는 알고리즘입니다.

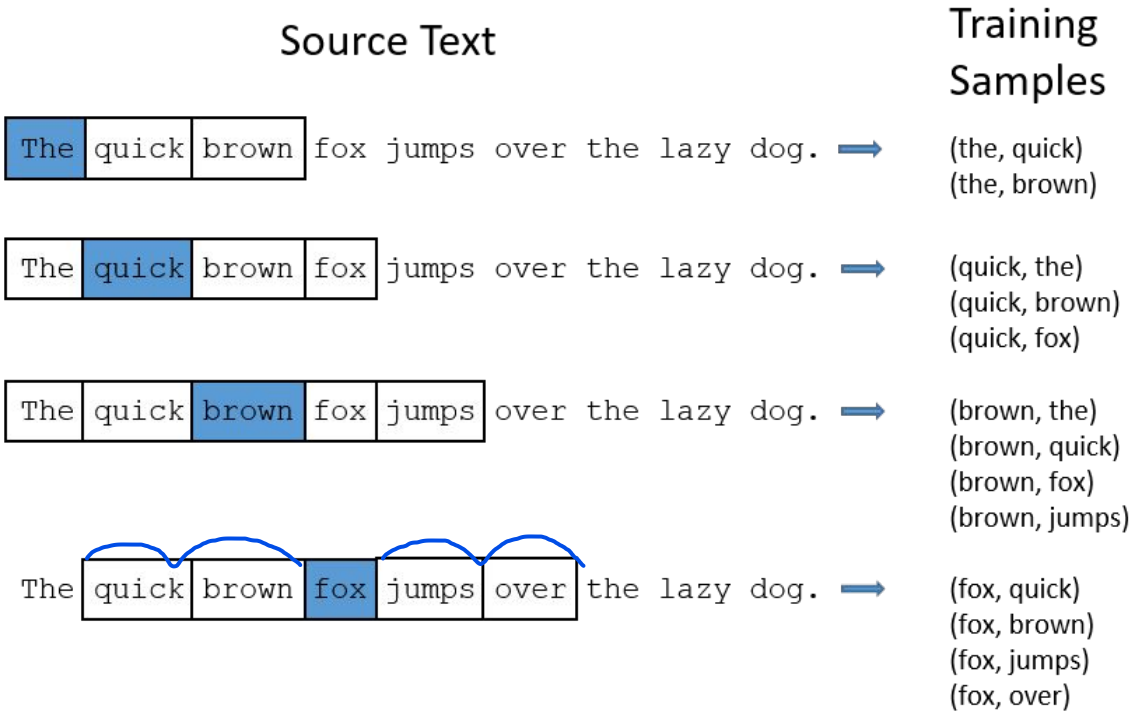


Word2Vec data generation (skip-gram)

파랑 단어의 앞 뒤를 관련성이 있다고 봄

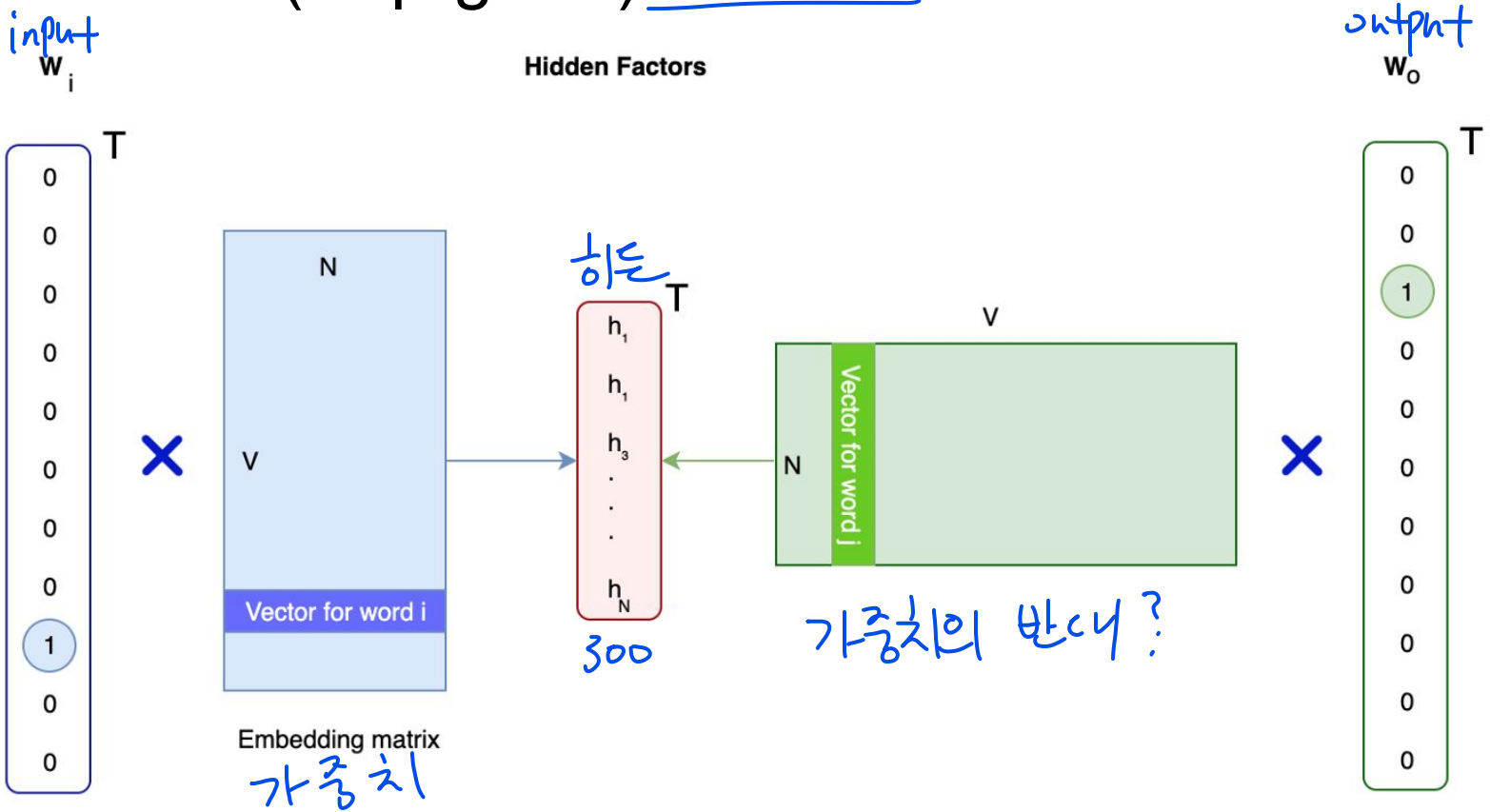
Skip-gram :

주어진 단어와 가장 관련성이 높은 단어를 찾는데 사용되는 비지도 학습 기술. 주어진 목표 단어에 대한 문맥 (이웃하는) 단어를 예측하는데 사용됩니다.



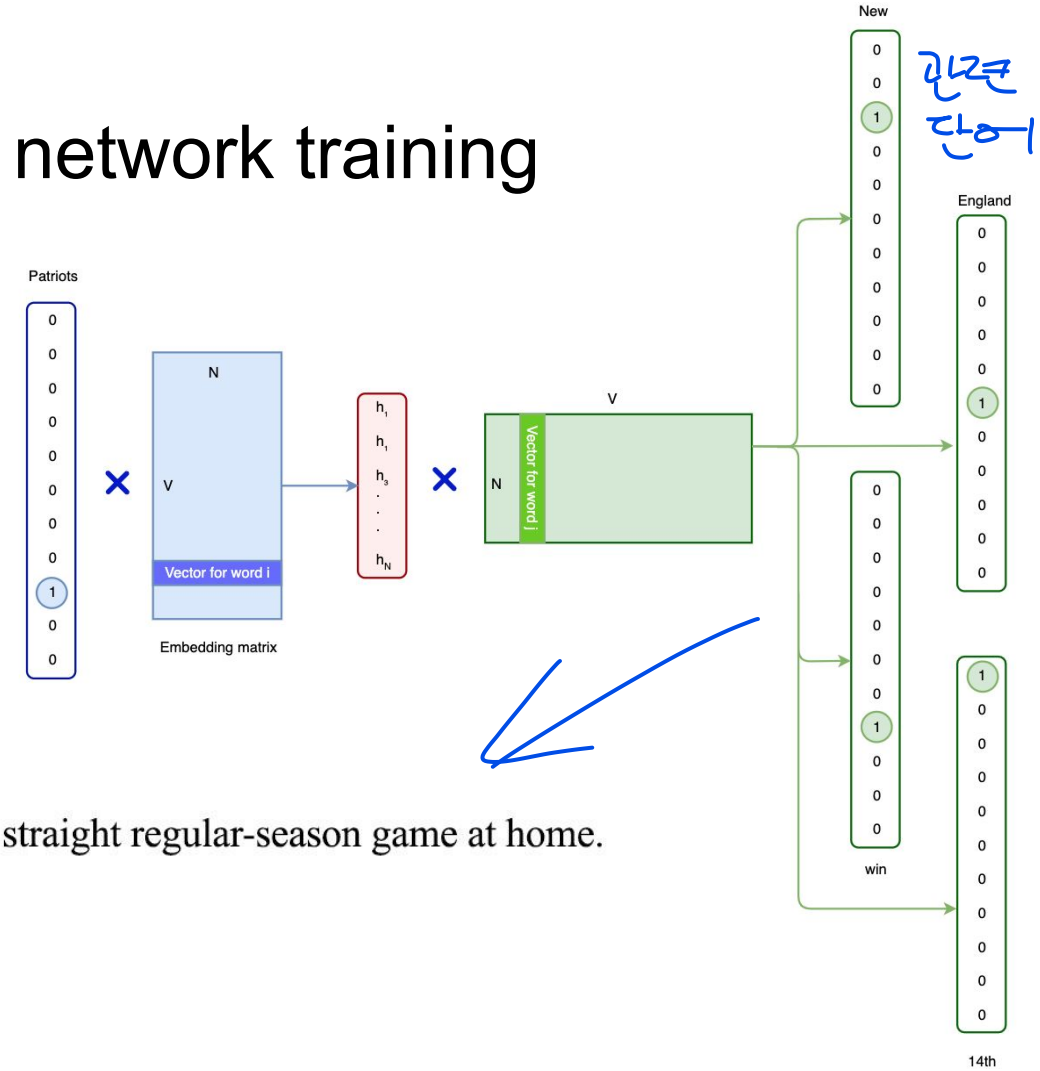
뉴런은 아님, 활성화 함수 없음

Word2Vec (skip-gram) network structure



Word2Vec (skip-gram) network training

단어가 주어지면 텍스트 문맥에서 주변 단어를 예측하는 네트워크: 5-그램 모델 (5 개의 연속 단어)을 사용한다고 가정하면, “Patriots”라는 단어가 주어지면 다음과 같은 훈련 데이터로 이웃 단어를 예측할 수 있습니다.



New England Patriots win 14th straight regular-season game at home.

Patriots → New
Patriots → England
Patriots → win
Patriots → 14th

Word embeddings

- Word2Vec pre-trained model 훈련이 오래걸리므로 이걸 쓰자
  - Google 뉴스 데이터세트 (약 1000 억 문장)의 일부에 대해 사전 훈련된 벡터를 제공하고 있습니다. 이 모델에는 300 만 개의 단어와 구에 대한 300 차원 벡터가 포함되어 있습니다. 크기는 약 1.5 GB. (GoogleNews-vectors-negative300.bin.gz)
  - 64-bit Python 에서, 약 6.7 GB 크기의 행렬을 사용한다.
    3 million words * 300 features * 8 bytes/feature = ~6.7GB

(Ex 1) Word2vec pre-trained model demo

```
import numpy as np
from sklearn.manifold import TSNE
import matplotlib.pyplot as plt
```

Download the word vectors

```
import gensim.downloader as api
word2vec_model = api.load('word2vec-google-news-300')
```

vector representation of a word

```
word2vec_model["beautiful"]
```

```
array([-0.01831055,  0.05566406, -0.01153564,  0.07275391,  0.15136719,
       -0.06176758,  0.20605469, -0.15332031, -0.05908203,  0.22851562,
…
       -0.09765625,  0.09619141, -0.09960938, -0.00285339, -0.03637695,
        0.15429688,  0.06152344, -0.34570312,  0.11083984,  0.03344727],
      dtype=float32)
```

word vectors understand the meanings of words

```
word2vec_model.most_similar("girl")
```

girl 과 유사한 단어 찾아줘

```
[ ('boy', 0.8543272018432617), ('teenage_girl', 0.7927976250648499), ('woman', 0.7494640946388245),
 ('teenager', 0.717249870300293), ('schoolgirl', 0.7075953483581543),
 ('teenaged_girl', 0.6650916337966919), ('daughter', 0.6489864587783813),
 ('mother', 0.6478164196014404), ('toddler', 0.6473966836929321), ('girls', 0.6154742240905762)]
```

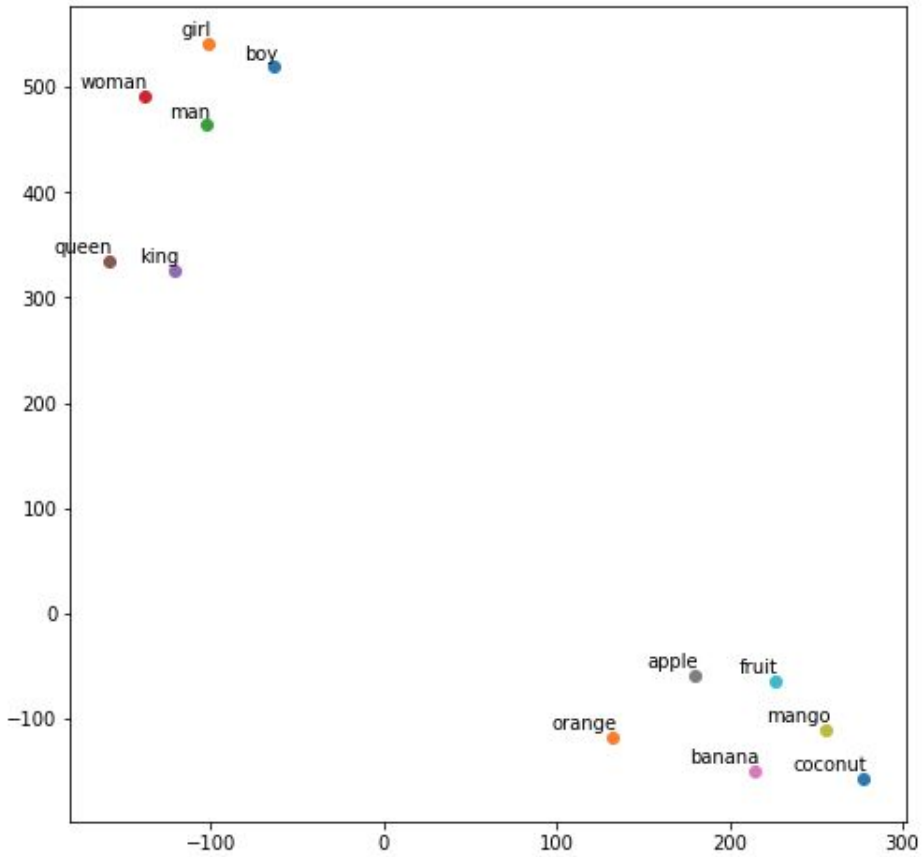
queen - girl + boy = king

```
word2vec_model.most_similar(positive=['boy', 'queen'], negative=['girl'], topn=1)
```

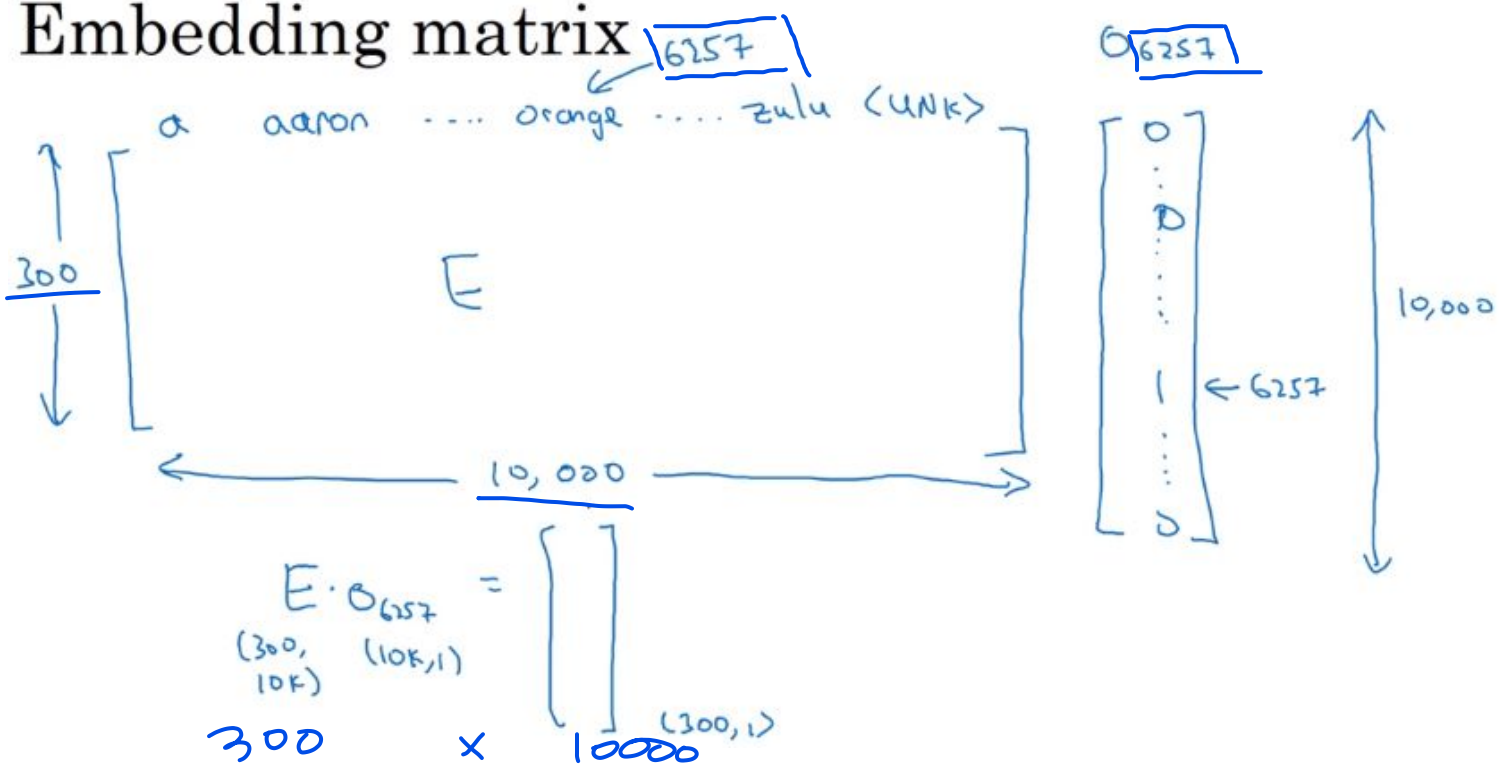
```
[('king', 0.7298422455787659)]
```

```
vocab = ["boy", "girl", "man", "woman", "king", "queen", "banana", "apple", "mango", "fruit",
"coconut", "orange"]

def tsne_plot(model):
    labels = []
    wordvecs = []
    for word in vocab:
        wordvecs.append(model[word])
        labels.append(word)

    tsne_model = TSNE(perplexity=3, n_components=2, init='pca', random_state=42)
    coordinates = tsne_model.fit_transform(wordvecs)
```

```
    x = []
    y = []

    for value in coordinates:
        x.append(value[0])
        y.append(value[1])

    plt.figure(figsize=(8,8))
    for i in range(len(x)):
        plt.scatter(x[i],y[i])
        plt.annotate(labels[i], xy=(x[i], y[i]),
                     xytext=(2, 2),
                     textcoords='offset points',
                     ha='right', va='bottom')

    plt.show()

tsne_plot(word2vec_model)
```



Embedding matrix

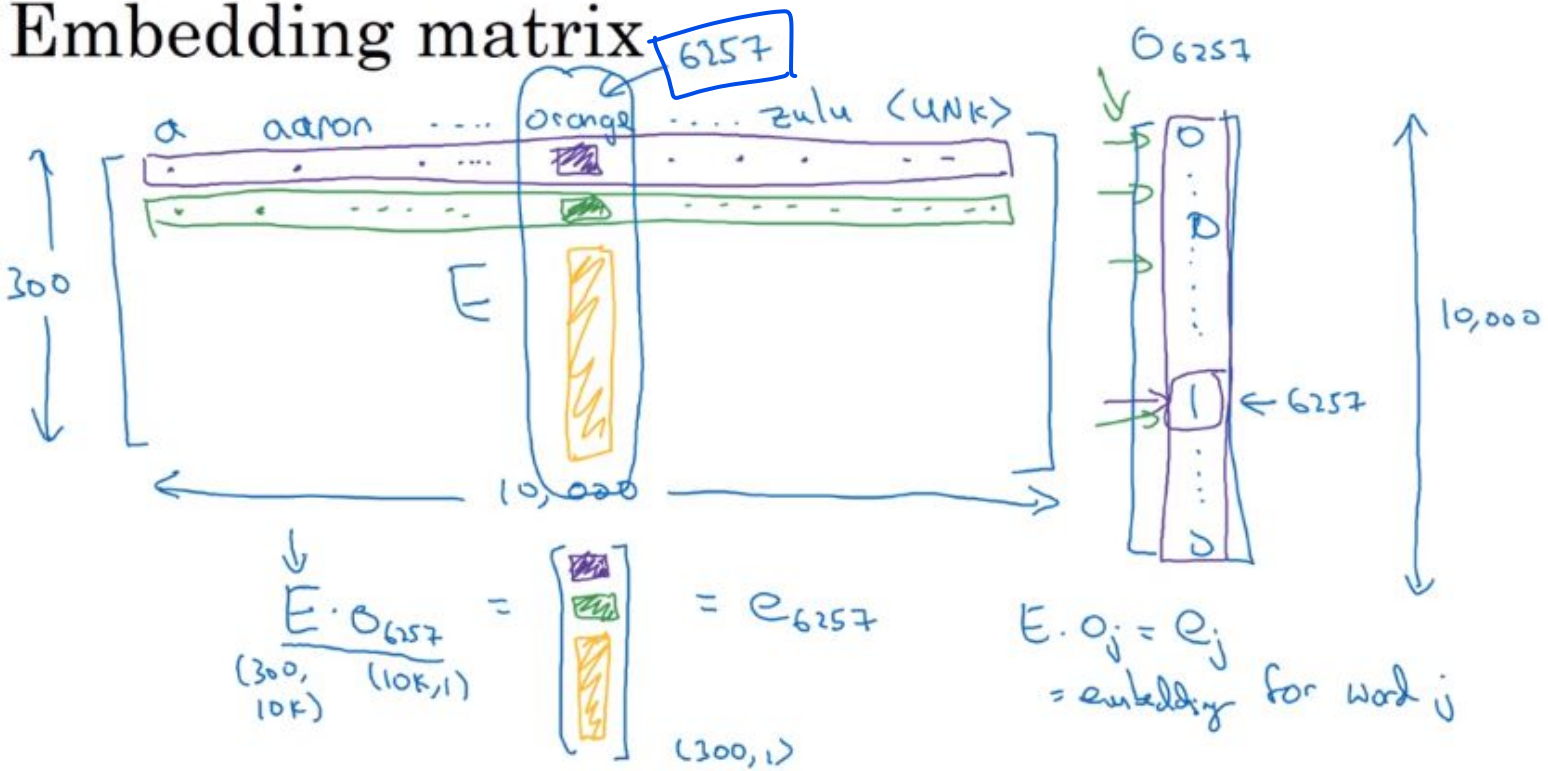


E = n_embedding features × n_vocabulary size sized matrix.

A word can be identified by an index. o_w = some **one-hot vector** for that word.

E x o_w = n_embedding features x 1 sized matrix = e_w = **embedding** for word `w`.

Embedding matrix



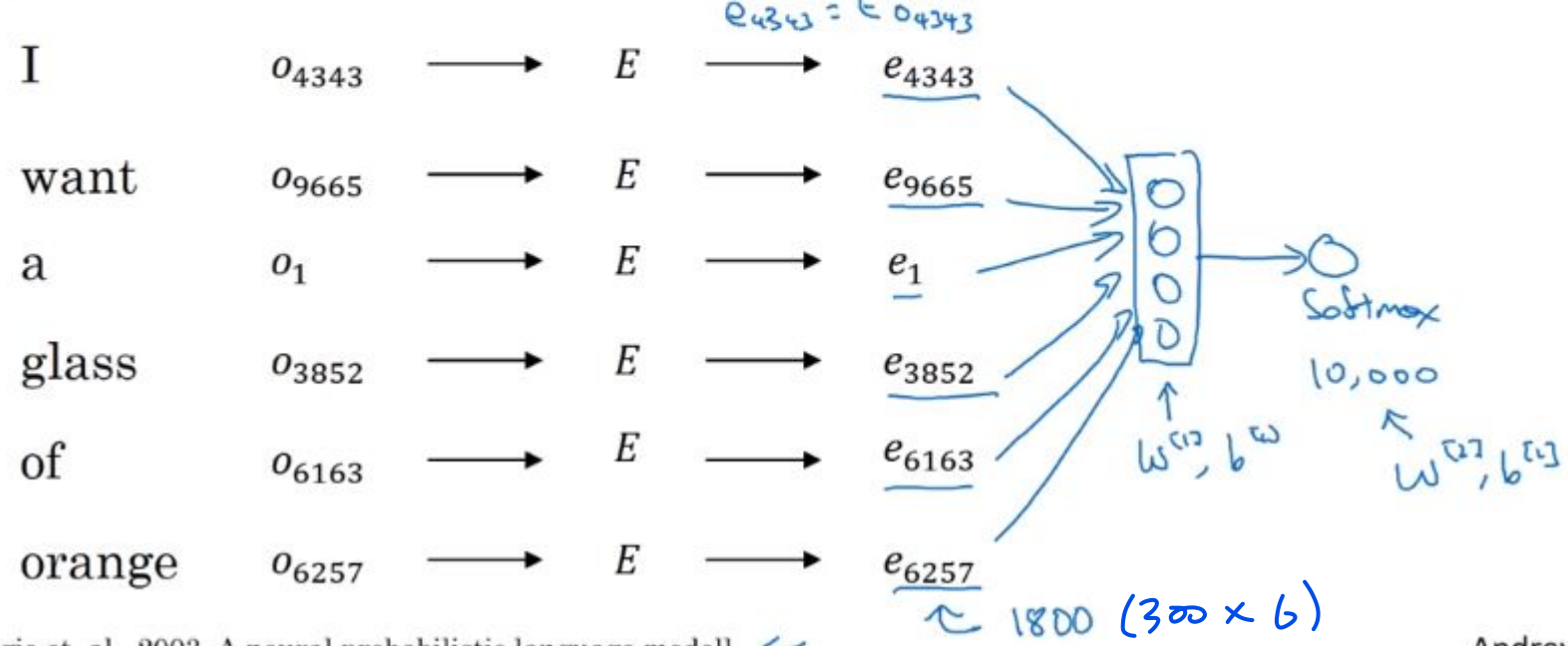
인어

Neural language model

4

I	want	a	glass	of	orange	juice.
4343	9665	1	3852	6163	6257	

$e_{4343} = E\, o_{4343}$

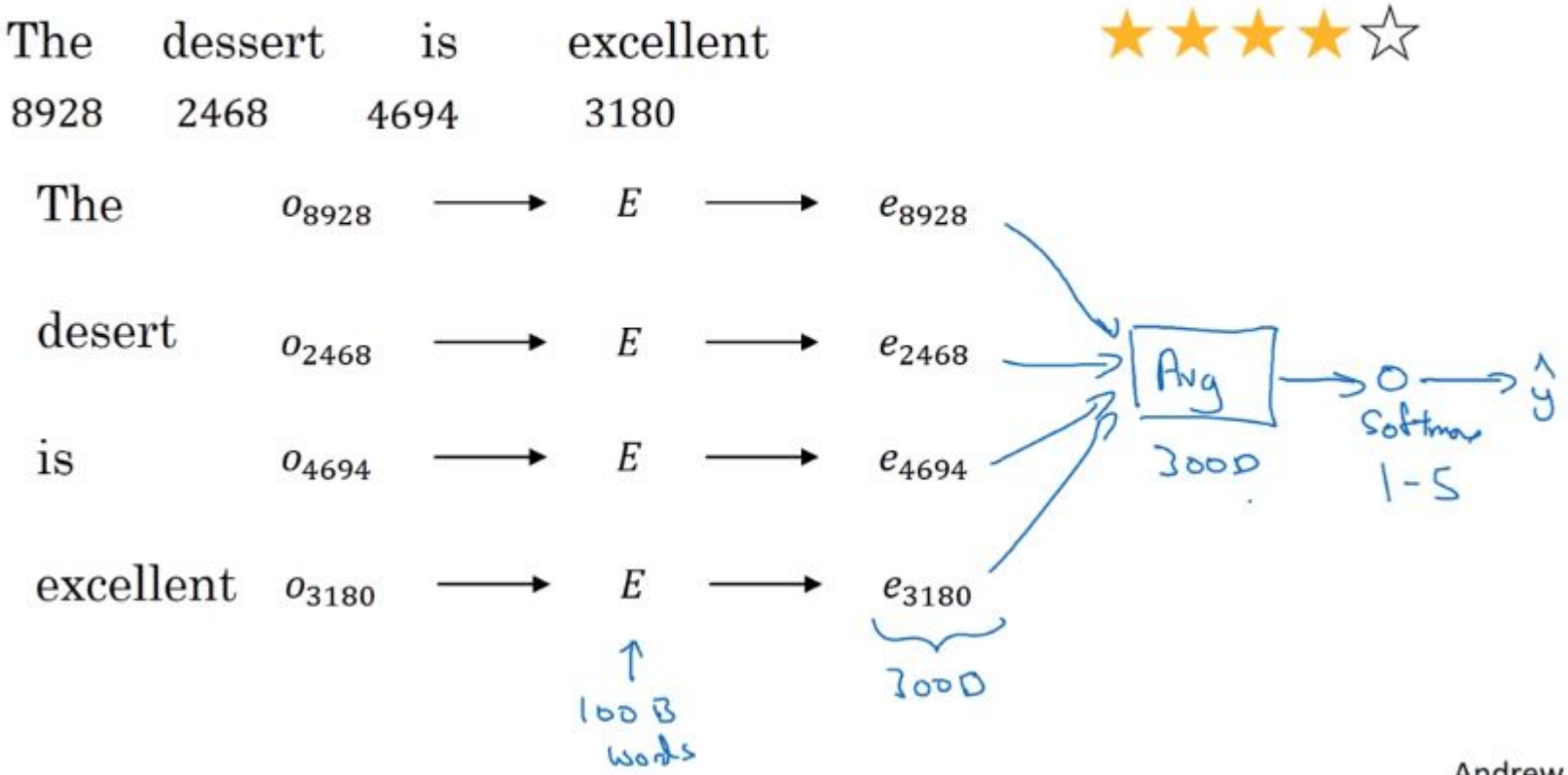


[Bengio et. al., 2003, A neural probabilistic language model]

Andrew Ng

감정

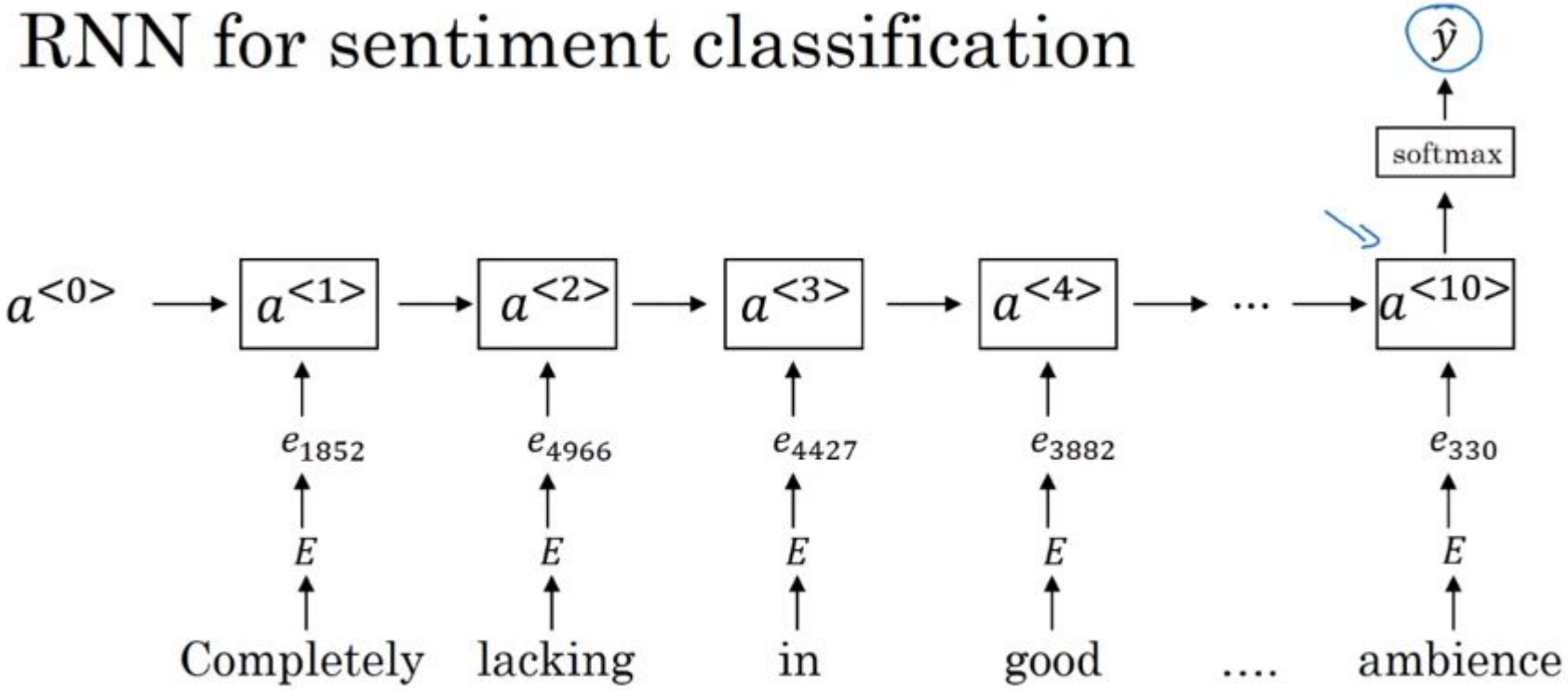
Simple sentiment classification model



Andrew Ng

사실 RNN이 더 자주 쓰임

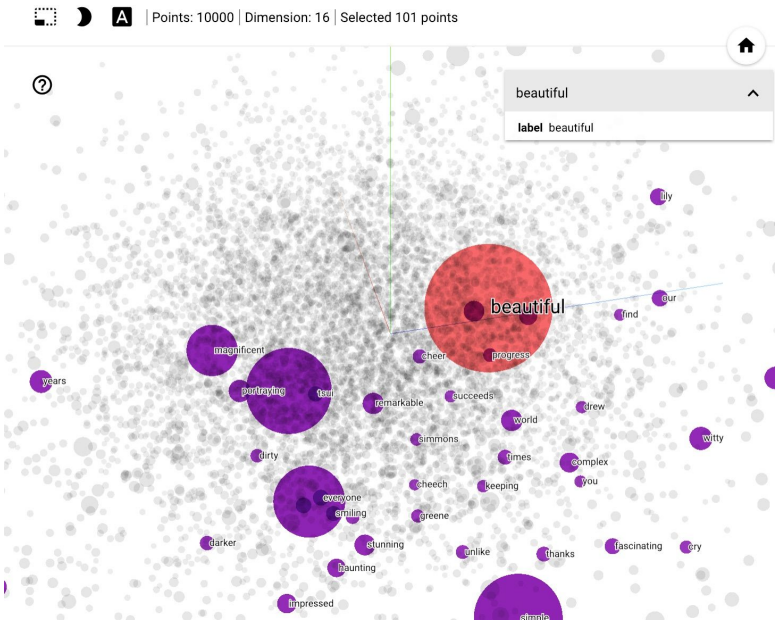
RNN for sentiment classification



many-to-one

(Ex 2 : Option) 단어 임베딩 - Practice with Colab

이 튜토리얼에서는 단어 임베딩을 소개합니다. 여기에는 작은 데이터 세트에서 단어 임베딩을 처음부터 학습하는 완전한 코드가 포함되어 있으며, Embedding Projector (아래 이미지 참조)를 사용하여 임베딩을 시각화 합니다.



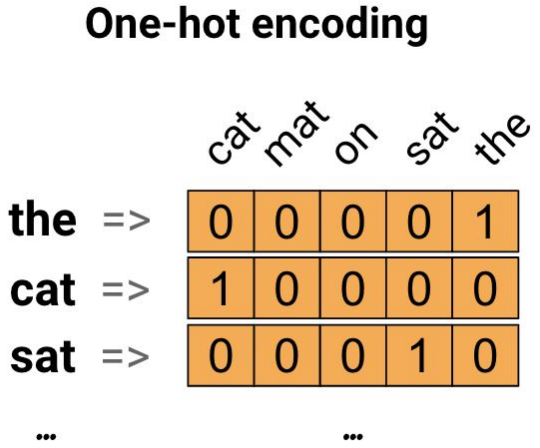
텍스트를 숫자로 표현

기계 학습 모델은 벡터(숫자 배열)를 입력으로 사용합니다. 텍스트로 작업 할 때 가장 먼저해야 할 일은 문자열을 모델로 보내기 전에 문자열을 숫자로 변환(또는 텍스트를 "벡터화")하는 전략을 제시해야 합니다. 이 섹션에서는 이를 위한 세 가지 전략을 살펴 보겠습니다.

- 원-핫 인코딩(One-hot encodings)
- 정수 인코딩 (Integer encoding)
- 단어 임베딩(Word embeddings)

원-핫 인코딩(One-hot encodings)

첫 번째 아이디어로서, 우리는 어휘에서 각 단어를 "일대일" 인코딩 할 수 있습니다. "The cat sat on the mat"라는 문장을 생각해보십시오. 이 문장의 어휘 (또는 독특한 단어)는 (cat, mat, on, sat, the) 입니다. 각 단어를 나타내기 위해 어휘의 개수와 길이가 같은 제로 벡터를 만든 다음, 단어에 해당하는 색인에 각 단어를 배치합니다. 이 방법은 다음 그림에 나와 있습니다.



문장의 인코딩을 포함하는 벡터를 만들기 위해 각 단어에 대한 원핫 벡터를 연결할 수 있습니다.

요점 :이 방법은 비효율적입니다. 원-핫 인코딩 된 벡터는 매우 희박합니다 (즉, 대부분의 인덱스는 0 임). 어휘에 10,000 단어가 있다고 상상해보십시오. 각 단어를 원-핫 인코딩하면 요소의 99.99%가 0인 벡터를 만듭니다.

각 단어를 고유 정수로 인코딩 (integer encoding)

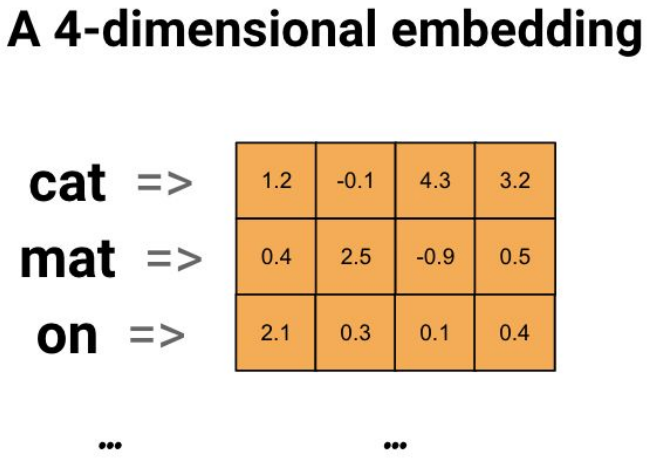
두 번째 방법은 고유한 숫자를 사용하여 각 단어를 인코딩하는 것입니다. 위의 예를 계속하면 1을 "cat"에, 2를 "mat"등에 지정할 수 있습니다. 그런 다음 "매트에 앉은 고양이"라는 문장을 [5, 1, 4, 3, 5, 2]와 같은 밀도가 높은 벡터로 인코딩 할 수 있습니다. 이 접근법은 효율적입니다. 희박한 벡터 대신 이제 모든 요소가 가득 찬 조밀한 벡터가 됩니다.

그러나 이 방법에는 두 가지 단점이 있습니다.

- 정수 인코딩은 임의적입니다 (단어 사이의 관계를 잡아내지 않습니다).
- 정수 인코딩은 모델이 해석하기 어려울 수 있습니다. 예를 들어 선형 분류기는 각 피처에 대한 단일 가중치를 학습합니다. 두 단어의 유사성과 인코딩의 유사성 사이에는 관계가 없으므로 이 특징-가중치 조합은 의미가 없습니다.

단어 임베딩(Word embeddings)

단어 임베딩을 사용하면 유사한 단어가 유사한 인코딩을 갖는 효율적이고 밀도가 높은 표현을 사용할 수 있습니다. 중요한 것은 이 인코딩을 직접 지정할 필요가 없습니다. 임베딩은 부동 소수점 값으로 구성된 밀도가 높은 벡터입니다 (벡터의 길이는 지정한 매개 변수입니다). 임베드에 대한 값을 수동으로 지정하는 대신 학습 가능한 매개 변수 (모델이 밀도가 높은 레이어의 가중치를 학습하는 것과 동일한 방식으로 학습 중에 모델이 학습한 가중치)입니다.



8 차원(소형 데이터 세트의 경우)부터, 큰 데이터 세트로 작업 할 때는 최대 1024 차원의 단어 임베딩을 하는 것이 일반적입니다. 더 높은 차원의 임베딩은 단어 사이의 세밀한 관계를 캡처 할 수 있지만 더 많은 데이터를 학습해야합니다.

왼쪽은 단어 임베딩을 위한 다이어그램입니다. 각 단어는 부동 소수점 값으로 구성된 4 차원 벡터로 표시됩니다. 임베딩을 생각하는 또 다른 방법은 "조회 테이블(lookup table)"입니다. 이러한 가중치를 학습 한 후에는 표에서 해당하는 밀도가 높은 벡터를 찾아 각 단어를 인코딩 할 수 있습니다.

Setup

```
%tensorflow_version 2.x
import tensorflow as tf

from tensorflow import keras
from tensorflow.keras import layers

import tensorflow_datasets as tfds
tfds.disable_progress_bar()

print(tf.__version__)
```

임베드 레이어 사용

Keras를 사용하면 단어 임베딩을 쉽게 사용할 수 있습니다. 임베딩 Embedding 레이어를 살펴 보겠습니다.

임베딩 레이어는 정수 인덱스(특정 단어를 나타냄)에서 밀도가 높은 벡터(그들의 임베딩)에 매핑되는 조회 테이블로 이해 될 수 있습니다. 임베딩의 차원(또는 너비)은 밀도 층의 뉴런 수를 실험하는 것과 같은 방식으로 문제에 대해 잘 작동하는지 확인할 수 있는 매개 변수입니다.

```
embedding_layer = layers.Embedding(1000, 5)
```

임베딩 레이어를 만들면 임베딩의 가중치가 다른 레이어와 마찬가지로 임의로 초기화됩니다. 훈련 중에는 역전파를 통해 점차 조정됩니다. 학습이 되면, 학습된 단어 임베딩들은 단어들 간의 유사성을 개략적으로 인코딩합니다 (모델이 학습한 특정 문제에 대해 학습되었으므로).

임베딩 레이어에 정수를 전달하면 결과는 각 정수를 임베딩 테이블의 벡터로 바꿉니다:

```
result = embedding_layer(tf.constant([1,2,3]))
result.numpy()
```

```
array([[-0.00910502,  0.00617131,  0.03805144,  0.01175289,  0.03235831],
       [ 0.0350986 ,  0.01272419, -0.01147569,  0.03842572, -0.01150563],
       [-0.00227773,  0.00467912,  0.02497515,  0.00768433, -0.04889264]],
      dtype=float32)
```

텍스트 또는 시퀀스 문제의 경우 임베딩 레이어는 (샘플, sequence_length)의 모양으로 정수로 구성된 2D 텐서가 됩니다. 여기서 각 항목은 정수 시퀀스입니다. 가변 길이의 시퀀스를 포함 할 수 있습니다. 모양 (32, 10) (길이 10인 32 시퀀스의 batch) 또는 (64, 15) (길이 15인 64 시퀀스의 batch)로 batch 위의 임베드 레이어에 공급할 수 있습니다.

반환된 텐서는 입력보다 축을 하나 더 가지며, 임베딩 벡터는 새로운 마지막 축을 따라 정렬됩니다. (2, 3) 인 입력 배치를 전달하면 출력은 (2, 3, N)이 됩니다

```
result = embedding_layer(tf.constant([[0,1,2],[3,4,5]]))
result.shape
```

일련의 시퀀스가 입력으로 주어지면, 임베딩 레이어는 (샘플, sequence_length, embedding_dimensionality) 모양의 3D 부동 소수점 텐서를 반환합니다. 이 가변 길이 시퀀스를 고정된 표현으로 변환하기 위해 다양한 표준 접근법이 있습니다. RNN, Attention 또는 Pooling 레이어를 Dense 레이어로 전달하기 전에 사용할 수 있습니다. 이 자습서는 가장 간단하기 때문에 풀링을 사용합니다. Text Classification with an RNN는 다음 단계로 좋습니다.

임베딩 학습(Learning embeddings from scratch)

이 자습서에서는 IMDB 영화 리뷰에 대한 감정 분류기를 훈련시킵니다. 이 과정에서 모델은 임베딩을 처음부터 학습합니다. 우리는 전처리 된 데이터 셋을 사용할 것입니다.

텍스트 데이터 세트를 처음부터 로드하려면 텍스트로드 자습서Loading text tutorial.를 참조하십시오.

```
(train_data, test_data), info = tfds.load(
    'imdb_reviews/subwords8k',
    split = (tfds.Split.TRAIN, tfds.Split.TEST),
    with_info=True, as_supervised=True)
```

Downloading and preparing dataset imdb_reviews/subwords8k/1.0.0 (download: 80.23 MiB, generated: Unknown size, total: 80.23 MiB) to /root/tensorflow_datasets/imdb_reviews/subwords8k/1.0.0...

Shuffling and writing examples to /root/tensorflow_datasets/imdb_reviews/subwords8k/1.0.0.incompleteQ75A3G/imdb_reviews-train.tfrecord

Shuffling and writing examples to /root/tensorflow_datasets/imdb_reviews/subwords8k/1.0.0.incompleteQ75A3G/imdb_reviews-test.tfrecord

Shuffling and writing examples to /root/tensorflow_datasets/imdb_reviews/subwords8k/1.0.0.incompleteQ75A3G/imdb_reviews-unsupervised.tfrecord

Dataset imdb_reviews downloaded and prepared to /root/tensorflow_datasets/imdb_reviews/subwords8k/1.0.0. Subsequent calls will reuse this data.

인코더(tfds.features.text.SubwordTextEncoder)를 가져와서 어휘를 빠르게 살펴보십시오.

어휘에서 "_"는 *공백을 나타냅니다. 전체 단어(*"_"로 끝나는 단어)와 큰 단어를 만드는 데 사용할 수 있는 부분 단어가 어떻게 어휘에 포함되어 있는지 확인하십시오.

```
encoder = info.features['text'].encoder
encoder.subwords[:20]
```

```
['the_', ', ', '. ', 'a_', 'and_', 'of_', 'to_', 's_', 'is_', 'br', 'in_', 'I_', 'that_',
 'this_', 'it_', ' /><', ' />', 'was_', 'The_', 'as_']
```

영화 리뷰는 길이가 다를 수 있습니다. padded_batch 메소드를 사용하여 리뷰 길이를 표준화합니다.

```
train_batches = train_data.shuffle(1000).padded_batch(10, padded_shapes=([None],[]))
test_batches = test_data.shuffle(1000).padded_batch(10, padded_shapes=([None],[]))
```

가져온 리뷰의 텍스트는 정수로 인코딩됩니다 (각 정수는 어휘의 특정 단어 또는 단어 부분을 나타냄).

배치가 가장 긴 예제에 채워지기 때문에 뒤따라 오는 0을 주의하십시오.

```
train_batch, train_labels = next(iter(train_batches))
train_batch.numpy()
```

```
array([[  19,  605,   25, ...,    0,    0,    0],
       [3654,  350, 1875, ...,    0,    0,    0],
       [ 691,    2,   12, ...,    0,    0,    0],
       ...,
       [  12,  321,   60, ...,    0,    0,    0],
       [8012, 6306, 7998, ...,    0,    0,    0],
       [  12, 2129,   22, ...,    0,    0,    0]])
```

간단한 모델 만들기

Keras Sequential API를 사용하여 모델을 정의합니다. 이 경우 "연속 단어 모음(Continuous bag of words)"스타일 모델입니다.

- 다음으로 Embedding 레이어는 정수로 인코딩 된 어휘를 사용하여 각 단어 인덱스에 대한 임베딩 벡터를 찾습니다. 이러한 벡터는 모델 학습으로 학습됩니다. 벡터는 출력 배열에 차원을 추가합니다. 결과 차수는 (배치, 순서, 임베딩) (batch, sequence, embedding)입니다.
- 다음으로 GlobalAveragePooling1D 레이어는 시퀀스 차원을 평균하여 각 예제에 대해 고정-길이 출력 벡터를 반환합니다. 이를 통해 모델은 가장 간단한 방법으로 가변 길이 입력을 처리 할 수 있습니다.
- 이 고정-길이 출력 벡터는 16 개의 숨겨진 단위로 완전히 연결된 (밀도) 레이어를 통해 파이프됩니다.
- 마지막 계층은 단일 출력 노드와 밀집되어 있습니다. 시그모이드 활성화 함수를 사용하면 이 값은 0과 1 사이의 부동 소수점이며 검토가 긍정적일 확률 (또는 신뢰 수준)을 나타냅니다.

주의 :이 모델은 마스킹을 사용하지 않으므로 제로 패딩이 입력의 일부로 사용되므로 패딩 길이가 출력에 영향을 줄 수 있습니다. 이 문제를 해결하려면 마스킹 및 패딩 안내서 masking and padding guide 를 참조하십시오.

```
embedding_dim=16

model = keras.Sequential([
  layers.Embedding(encoder.vocab_size, embedding_dim),
  layers.GlobalAveragePooling1D(),
  layers.Dense(16, activation='relu'),
  layers.Dense(1)
])

model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_3 (Embedding)      (None, None, 16)          130960
_________________________________________________________________
global_average_pooling1d_1 ( (None, 16)                0
_________________________________________________________________
dense_2 (Dense)              (None, 16)                272
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 17
=================================================================
Total params: 131,249
Trainable params: 131,249
Non-trainable params: 0
_________________________________________________________________
```

Compile and train the model

```
model.compile(optimizer='adam',
              loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
              metrics=['accuracy'])

history = model.fit(
    train_batches,
    epochs=10,
    validation_data=test_batches, validation_steps=20)
```

```
Epoch 1/10
2500/2500 [==============================] - 14s 6ms/step - loss: 0.5007 - accuracy: 0.7085 -
val_loss: 0.3909 - val_accuracy: 0.8550
Epoch 2/10
2500/2500 [==============================] - 14s 6ms/step - loss: 0.2816 - accuracy: 0.8836 -
val_loss: 0.4244 - val_accuracy: 0.8350
…
Epoch 10/10
2500/2500 [==============================] - 14s 6ms/step - loss: 0.1094 - accuracy: 0.9625 -
val_loss: 0.4692 - val_accuracy: 0.8550
```

이 방법을 사용하면 모델의 검증 정확도가 약 88%에 도달합니다 (모델이 과적합하고 훈련 정확도가 상당히 높아짐).

```
import matplotlib.pyplot as plt

history_dict = history.history

acc = history_dict['accuracy']
val_acc = history_dict['val_accuracy']
loss=history_dict['loss']
val_loss=history_dict['val_loss']

epochs = range(1, len(acc) + 1)

plt.figure(figsize=(12,9))
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

plt.figure(figsize=(12,9))
plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')
plt.ylim((0.5,1))
plt.show()
```

학습된 임베딩 꺼내보기

다음으로 훈련 중에 배운 단어를 꺼내어 봅시다. 이것은 (vocab_size, embedding-dimension)모양의 행렬입니다.

```
e = model.layers[0]
weights = e.get weights()[0]
print(weights.shape) # shape: (vocab_size, embedding_dim)
```

```
(8185, 16)
```

이제 가중치를 디스크에 씁니다. Embedding Projector 를 사용하기 위해 벡터 파일 (임베딩 포함)과 메타 데이터 파일 (단어 포함)의 두 가지 파일을 탭으로 구분 된 형식으로 업로드합니다.

```
import io

encoder = info.features['text'].encoder

out_v = io.open('vecs.tsv', 'w', encoding='utf-8')
out_m = io.open('meta.tsv', 'w', encoding='utf-8')

for num, word in enumerate(encoder.subwords):
  vec = weights[num+1] # skip 0, it's padding.
  out_m.write(word + "\n")
  out_v.write('\t'.join([str(x) for x in vec]) + "\n")
out_v.close()
out_m.close()
```

Colaboratory 에서 이 학습서를 실행중인 경우 다음 스니펫을 사용하여 이러한 파일을 로컬 시스템으로 다운로드하거나 파일 브라우저 (*보기-> 목차-> 파일 브라우저*)를 사용할 수 있습니다.

```
try:
 from google.colab import files
except ImportError:
  pass
else:
 files.download('vecs.tsv')
 files.download('meta.tsv')
```

임베딩을 시각화

임베딩을 시각화하기 위해 임베딩 프로젝터에 업로드합니다.
임베딩 프로젝터를 엽니다 (이것은 로컬 TensorBoard 인스턴스에서도 실행할 수 있습니다).

- "데이터로드"를 클릭하십시오.
- 위에서 만든 두 파일 vecs.tsv와 meta.tsv를 업로드합니다.

훈련한 임베딩이 표시됩니다. 가장 가까운 이웃을 찾기 위해 단어를 검색 할 수 있습니다. 예를 들어 "beautiful"을 검색해보십시오. "wonderful"과 같은 이웃을 볼 수 있습니다.

참고 : 임베드 레이어를 학습하기 전에 가중치가 무작위로 초기화 된 방식에 따라 결과가 약간 다를 수 있습니다

참고 : 실험적으로는 더 간단한 모델을 사용하여 해석하기 쉬운 임베딩을 생성 할 수 있습니다. Dense (16) 레이어를 삭제하고 모델을 다시 훈련시키며 임베딩을 다시 시각화하십시오.