KU KONKUK UNIVERSITY

Text Representation

자연어처리연구실

이산 표현 (Discrete Representation)

- 원-핫 인코딩 (one-hot encoding)
  - 단어를 벡터로 표현하는 가장 간단한 방법
  - 단어 사전(dictionary)을 구성하고 해당 단어를 1로, 그 밖의 단어는 0으로 표현
    - 사전: 개, 고양이, 늑대, 사자, 송어, 잉어
    - 단어: 개 [1, 0, 0, 0, 0, 0], 고양이 [0, 1, 0, 0, 0, 0], 늑대 [0, 0, 1, 0, 0, 0]
- 원-핫 인코딩의 한계
  - 대용량 메모리 필요
    - 사전의 크기: 음성 인식 20K, 구문 분석: 50K, 대용량 사전: 500K, 구글 1T 말뭉치: 13M
  - 유사성 비교 불가능
    - Sim(개, 늑대) = AND([1, 0, 0, 0, 0, 0], [0, 0, 1, 0, 0, 0]) = 0
    - Sim(개, 잉어) = AND([1, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 1]) = 0

개=늑대=잉어??

Edited by Harksoo Kim

분산 표현 (Distributed Representation)

- 분산 표현
  - 단어를 문맥에 기반하여 표현하는 방법
  - 비슷한 문맥에서 등장하는 단어는 비슷한 의미를 가질 것이라는 가정에서 출발

신종 **코로나** 바이러스 집단 감염증이 일상생활을 통해 지속 확산되고 있다.

부산 수영구 댄스 동호회발 신종 **코로나** 바이러스 집단 감염이 목욕탕으로 퍼지는 등 확산되고 있다.

코로나? 신종, 바이러스, 집단, 감염, 확산, 일상, 생활, ..., 목욕탕

프랑스는 지금까지 북부도시 릴에서 2명의 **메르스** 바이러스 감염환자가 발생했다.

메르스? 바이러스, 감염, 프랑스, 지금, ..., 환자, 발생

Sim(코로나, 메르스) → AND(코로나, 메르스) ≠0

Edited by Harksoo Kim

공기 행렬 (Co-Occurrence Matrix)

[예문] I like deep learning. I like NLP. I enjoy flying.

Window size: 1

counts	I	like	enjoy	deep	learning	NLP	flying	.
I	0	2	1	0	0	0	0	0
like	2	0	0	1	0	1	0	0
enjoy	1	0	0	0	0	0	1	0
deep	0	1	0	0	1	0	0	0
learning	0	0	0	1	0	0	0	1
NLP	0	1	0	0	0	0	0	1
flying	0	0	1	0	0	0	0	1
.	0	0	0	0	1	1	1	0

Sim(NLP, learning) = ?

그림 출처: Kira Radinsky 교수 강의자료

Edited by Harksoo Kim

코사인 유사도 (Cosine Similarity)

- 코사인 유사도
  - 길이로 정규화된 내적을 바탕으로 두 벡터 사이의 유사도를 측정하는 척도

두 벡터 요소의 구성이 비슷하면 큰 값을 가짐

$$\vec{X} \cdot \vec{Y} = |\vec{X}||\vec{Y}| \cos(\theta)$$

$$\cos(\vec{X}, \vec{Y}) = \frac{\vec{X} \cdot \vec{Y}}{|\vec{X}||\vec{Y}|}$$

두 벡터가 직각: 0
두 벡터가 동일: 1

$$\vec{X} \cdot \vec{Y} = x_1 y_1 + x_2 y_2 + \cdots + x_n y_n$$

$$|\vec{X}| = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}$$

$$|\vec{Y}| = \sqrt{y_1^2 + y_2^2 + \cdots + y_n^2}$$

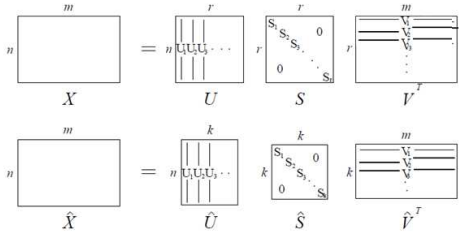
cos(NLP, learning) = 1/2 = 0.5

```
import torch

NLP = torch.FloatTensor([0,1,0,0,0,0,0,1])
learning = torch.FloatTensor([0,0,0,1,0,0,0,1])

print(torch.cosine_similarity(NLP, learning, dim=0))

tensor(0.5000)
```


Problems with Co-Occurrence Vectors

- 차원의 저주 (Curse of dimensionality)
  - 차원이 증가하면서 학습데이터의 수가 차원의 수보다 적어서 성능이 저하되는 현상 → 희소 데이터 문제 (sparse data problem)
- 특이값 분해 (SVD; Singular Value Decomposition)
  - 행렬을 특정한 구조로 분해하는 방식

SVD (Singular Value Decomposition)

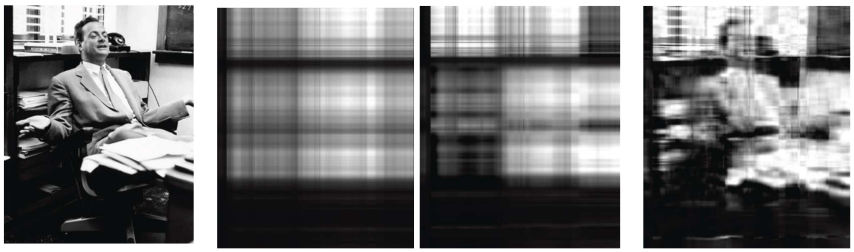


$\hat{X}$ is the best rank k approximation to X, in terms of least squares

그림 출처: Kira Radinsky 교수 강의자료

Effect of SVD

Original | 1st Singular | 1st & 2nd Singular | .....



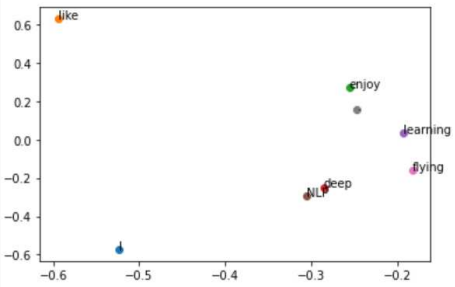
$R_i = c_i \cdot SV_1$ $R_i = c_i \cdot SV_1 + b_i \cdot SV_2$

그림 출처: Kira Radinsky 교수 강의자료

Simple SVD Word Vectors in Python

[예문] I like deep learning. I like NLP. I enjoy flying.

```
import numpy as np
import matplotlib.pyplot as plt

dic= ["I", "like", "enjoy", "deep", "learning", "NLP", "flying", "." ]
X = np.array([[0,2,1,0,0,0,0,0],
              [2,0,0,1,0,1,0,0],
              [1,0,0,0,0,0,1,0],
              [0,1,0,0,1,0,0,0],
              [0,0,0,1,0,0,0,1],
              [0,1,0,0,0,0,0,1],
              [0,0,1,0,0,0,0,1],
              [0,0,0,0,1,1,1,0]])
U, S, Vt =np.linalg.svd(X, full_matrices=False)

for i in range(len(dic)):
  plt.scatter(U[i,0], U[i,1])
  plt.text(U[i,0], U[i,1], dic[i])

plt.show()
```



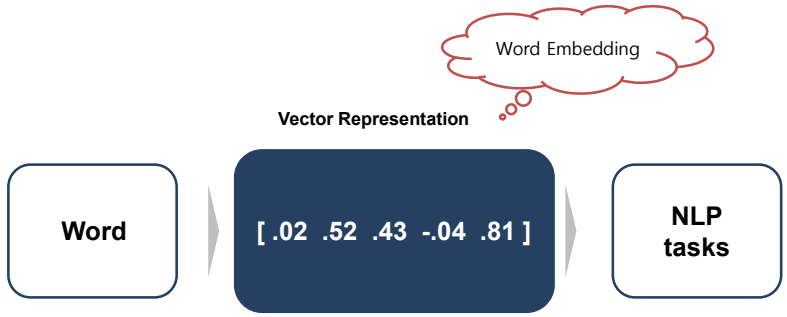
참고: Kira Radinsky 교수 강의자료

From SVD To Word2Vec

- SVD의 문제점
  - 계산에 너무 오랜 시간이 소요됨
    - n*m 행렬 계산 → $O(mn^2)$
  - 유연성이 떨어짐
    - 새로운 단어나 문서가 추가될 경우에 SVD를 처음부터 다시 수행
- 해결 방안
  - Learning representations by back-propagation errors (Rumelhart et al., 1986)
  - Neural probabilistic language model (Bengio et al., 2003)
  - NLP from Scratch (Collobert & Weston,2008)
  - Word2Vec (Mikolov et al., 2013)
    - Instead of capturing co-occurrence counts directly: Predict surrounding words of every word

Distributed Representation (Again)

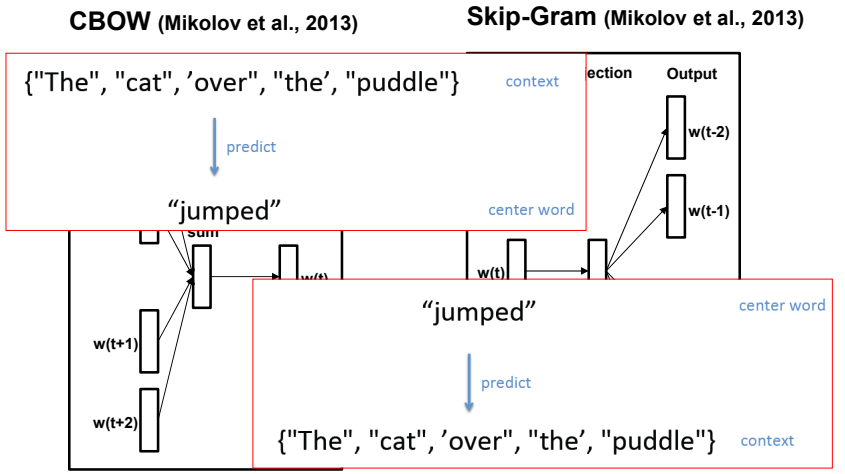
고차원 one-hot 벡터 → 저차원 실수 벡터
기본 아이디어: 비슷한 문맥에서 나온 단어는 비슷한 뜻을 가짐!



- **Word2vec (Mikolov, 2013)**
- **Glove (Pennington, et al., 2014)**

그림 출처: NLP 겨울학교(오혜연 교수)

Word2Vec



How will it work?

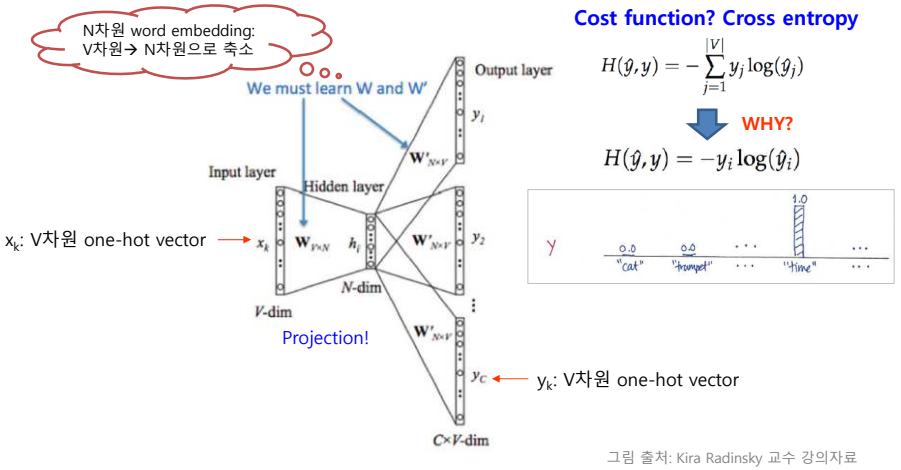


그림 출처: Kira Radinsky 교수 강의자료

SoftMax

Cost function? Cross entropy

$$H(\hat{y}, y) = -\sum_{j=1}^{|V|} y_j \log(\hat{y}_j)$$

WHY?

$$H(\hat{y}, y) = -y_i \log(\hat{y}_i)$$

Softmax: Linear regression generalization to multi-class

$$h_\theta(x) = \begin{bmatrix} P(y=1|x;\theta) \\ P(y=2|x;\theta) \\ \vdots \\ P(y=K|x;\theta) \end{bmatrix} = \frac{1}{\sum_{j=1}^{K} \exp(\theta^{(j)\top} x)} \begin{bmatrix} \exp(\theta^{(1)\top} x) \\ \exp(\theta^{(2)\top} x) \\ \vdots \\ \exp(\theta^{(K)\top} x) \end{bmatrix}$$

Output: a K-dimensional vector (whose elements sum to 1)

확률로 만들려면?
V만큼의 계산 필요!

- Hierarchical SoftMax
- Negative Sampling
- Sub-Sampling Frequent Words

GloVe (Global Vectors)

Main Insight: Ratio of co-occurrence probabilities can encode meaning

Prediction → Probability

→ **Use global information (co-occurrence over corpus) while learning word vectors**

	x = solid	x = gas	x = water	x = random
$P(x\|\text{ice})$	large	small	large	small
$P(x\|\text{steam})$	small	large	large	small
$\frac{P(x\|\text{ice})}{P(x\|\text{steam})}$	large	small	~1	~1

Solid가 문맥으로 주어졌을 때, ice와 steam의 비율이 크도록 학습!

Main Insight of GloVe

- Ratio of co-occurrence probabilities can encode meaning!

How can we capture this behavior in the word vector space?

Log-bilinear model: $w_i \cdot w_j = \log P(i|j)$

Vector differences: $w_x \cdot (w_a - w_b) = \log \frac{P(x|a)}{P(x|b)}$

x=solid, a=ice, b=steam

Think: a = "ice", b = "steam"

The training objective of GloVe is to **learn word vectors such that their dot product equals the logarithm of the words' probability of co-occurrence**.

Object Function of GloVe

그림 출처: Kira Radinsky 교수 강의자료

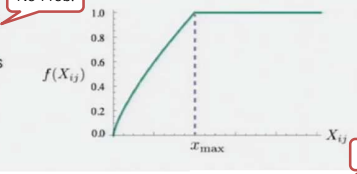
The log-bilinear model

$$J = \frac{1}{2} \sum_{ij} f(P_{ij}) \left(w_i \cdot \tilde{w}_j - \log P_{ij}\right)^2$$

$$J = \sum_{i,j=1}^{V} f\left(X_{ij}\right) \left(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij}\right)^2$$

No Prob.

- X - cooccurrence matrix
- w - word vectors
- $\tilde{w}$ - context word vectors
- b - bias
- $\tilde{b}$ - bias



3/4 100

$$f(x) = \begin{cases} (x/x_{\max})^\alpha & \text{if } x < x_{\max} \\ 1 & \text{otherwise} \end{cases}$$

Fast training, scalable to huge corpora,
Good performance even with small corpus, and small vectors

From One-hot Rep. to Distributed Rep.

- fastText (by facebook ← word2vec by google)
  - 부분 단어(subword)로 학습하여 노이즈에 강함
- GloVe (by stanford)
  - 동시 등장 확률을 함께 학습

사전 만들기

실습 코드 다운로드: https://github.com/KUNLP/Lecture

Mecab 인스톨

```
!git clone https://github.com/SOMJANG/Mecab-ko-for-Google-Colab.git
%cd Mecab-ko-for-Google-Colab
!bash install_mecab-ko_on_colab190912.sh
```

실행

```
texts = ["죽는 날까지 하늘을 우러러 한 점 부끄럼이 없기를",
         "잎새에 이는 바람에도 나는 괴로워 했다.",
         "별을 노래하는 마음으로 모든 죽어 가는 것을 사랑해야지",
         "그리고 나한테 주어진 길을 걸어가야겠다.",
         "오늘 밤에도 별이 바람에 스치운다."]

(indexes, words) = make_dic(texts)

print(word2index(indexes,'하늘'), word2index(indexes,'학교'))
print(index2word(words,4), index2word(words,0))
```

```
사전 크기: 24
4 0
하늘 unk
```


사전 만들기

make_dic 함수

```
from konlpy.tag import Mecab
from nltk import FreqDist
import numpy as np

def make_dic(texts):

  words = []
  m = Mecab()

  for snt in texts:
    res = m.pos(snt)
    for (lex,pos) in res:
      if pos[0] == 'N' or pos[0] == 'V':
        words.append(lex)

  # 단어의 빈도수 계산
  dic = FreqDist(np.hstack(words))
  print('사전 크기: {0:d}'.format(len(dic)))

  # 인덱스를 저장할 변수 초기화
  indexes = {}
  words = {}

  # 단어에 고유 번호(인덱스) 부여
  for num, word in enumerate(dic):
    idx = num+2
    indexes[word]= idx
    words[idx] = word

  # 문장 길이 정규화에 사용한 패딩 인덱스
  indexes['pad'] = 1
  words[1] = 'pad'
  # 사전에 없는 단어에 대한 인덱스
  indexes['unk'] = 0
  words[0] = 'unk'

  return (indexes, words)

def word2index(indexes, word):
  idx = indexes[word] if word in indexes else indexes['unk']
  return idx

def index2word(words, index):
  w = words[index] if index in words else -1
  return w
```


Word2Vec 사용하기

학습 데이터 만들기

```
from konlpy.tag import Mecab
import numpy as np
from gensim.models import Word2Vec, KeyedVectors

texts = ["죽는 날까지 하늘을 우러러 한 점 부끄럼이 없기를",
         "잎새에 이는 바람에도 나는 괴로워 했다.",
         "별을 노래하는 마음으로 모든 죽어 가는 것을 사랑해야지",
         "그리고 나한테 주어진 길을 걸어가야겠다.",
         "오늘 밤에도 별이 바람에 스치운다."]

# Word2Vec 학습에 사용할 데이터 만들기

m = Mecab()
result = []

for sent in texts:
  tag = m.pos(sent)
  words = []
  for (lex,pos) in tag:
    if pos[0] == 'N':
      words.append(lex)
  result.append(words)

print(result,"₩n")
```

대용량의 말뭉치를 이용

명사류(체언)만 추출

```
[['날', '하늘', '점', '부끄럼'], ['잎새', '바람', '나'],
['별', '노래', '마음', '것', '사랑'], ['나', '길'], ['밤', '별', '바람']]
```

Word2Vec 사용하기

학습 및 평가

```
# Word2Vec 학습시키기
model = Word2Vec(sentences=result, size=10, window=1, min_count=1, workers=1, sg=0)

# 값 읽어오기
print(model.wv['하늘'])
# 유사한 단어 가져오기
print(model.wv.most_similar("하늘"),"₩n")

# Word2Vec 모델 저장하기
model.wv.save_word2vec_format('/gdrive/My Drive/colab/text_rep/test_w2v')

# Word2Vec 모델 로드하기
loaded_model = KeyedVectors.load_word2vec_format("/gdrive/My Drive/colab/text_rep/test_w2v")

# 값 읽어오기
print(loaded_model.wv['하늘'])
# 유사한 단어 가져오기
print(loaded_model.wv.most_similar("하늘"))
```

- size: 임베딩 차원
- window:컨텍스트 윈도우 크기
- min_count: 단어 최소 빈도 수 제한
- workers: 학습을 위한 프로세스 수
- sg: 0 (CBOW), 1 (Skip-gram)

```
[-0.00181364  0.01349637  0.0494625  -0.04351401 -0.0074907  -0.00855643
  0.01056736 -0.04086038  0.0170731   0.01116189]
[('날', 0.2550535202026367), ('사랑', 0.19656619429588318), ('바람', 0.18
[-0.00181364  0.01349637  0.0494625  -0.04351401 -0.0074907  -0.00855643
  0.01056736 -0.04086038  0.0170731   0.01116189]
[('날', 0.2550535202026367), ('사랑', 0.19656619429588318), ('바람', 0.18
```


Pre-trained Word2Vec

- 영어
  - 구글이 학습한 300만 단어 벡터
  - https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edit
- 다국어(한국어 포함)
  - https://github.com/Kyubyong/wordvectors

Language	ISO 639-1	Vector Size	Corpus Size	Vocabulary Size
Bengali (w) | Bengali (f)	bn	300	147M	10059
Catalan (w) | Catalan (f)	ca	300	967M	50013
Chinese (w) | Chinese (f)	zh	300	1G	50101
Danish (w) | Danish (f)	da	300	295M	30134
Korean (w) | Korean (f)	ko	200	339M	30185

GloVe 사용하기

GloVe 인스톨

```
# colab 환경에 glove 설치
! pip install glove-python-binary
```

학습 및 평가

```
# Co-Occurrence Matrix 생성
corpus = Corpus()
corpus.fit(result, window=1)

# GloVe 학습시키기
glove = Glove(no_components=10, learning_rate=0.05)
glove.fit(corpus.matrix, epochs=20, no_threads=1, verbose=True)
glove.add_dictionary(corpus.dictionary)

# 값 읽어오기
print(glove.word_vectors[glove.dictionary['하늘']])
# 유사한 단어 가져오기
print(glove.most_similar("하늘"),"₩n")
```

10차원 임베딩

```
# GloVe 모델 저장하기
glove.save('/gdrive/My Drive/colab/text_rep/test_glove')

# GloVe 모델 로드하기
loaded_model = glove.load("/gdrive/My Drive/colab/text_rep/test_glove")

# 값 읽어오기
print(loaded_model.word_vectors[glove.dictionary['하늘']])
# 유사한 단어 가져오기
print(loaded_model.most_similar("하늘"))
```

```
[ 0.02104036 -0.01404578 -0.04851466  0.01803728 -0.0152335
 -0.04581227 -0.04701197 -0.00781063 -0.01010458]
[('부끄럼', 0.6554835514510583), ('점', 0.5460033054824257)
[ 0.02104036 -0.01404578 -0.04851466  0.01803728 -0.0152335
 -0.04581227 -0.04701197 -0.00781063 -0.01010458]
[('부끄럼', 0.6554835514510583), ('점', 0.5460033054824257)
```


Embedding Layer

- 임베딩 벡터를 사용하는 방법
  - 사전 학습된 벡터 사용
    - Word2Vec, GloVe 등을 읽어와서 사용하며 학습 과정에 업데이트 없음
  - 학습 과정에서 벡터 업데이트
    - 사전 학습된 벡터 또는 임의의 벡터를 만들고 학습 과정에서 업데이트 → 파인 튜닝 (fine-tuning)
    - 파이토치에서는 nn.Embedding()을 사용하여 구현

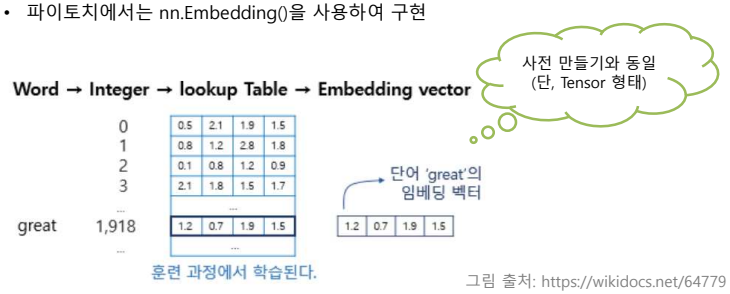


그림 출처: https://wikidocs.net/64779

Embedding Layer

사전 만들기

```
import torch
import torch.nn as nn

train_data = 'I like deep learning I like NLP I enjoy flying'

# 중복을 제거한 단어들의 집합인 단어 집합 생성.
word_set = set(train_data.split())

# 단어 집합의 각 단어에 고유한 정수 맵핑
vocab = {word: i+2 for i, word in enumerate(word_set)}
vocab['unk'] = 0
vocab['pad'] = 1
print(vocab)

# 임베딩 테이블: (단어수, 임베딩 사이즈)
embedding_table = torch.FloatTensor([[ 0.0,  0.0,  0.0],
                                     [ 0.0,  0.0,  0.0],
                                     [ 0.1,  0.8,  0.3],
                                     [ 0.7,  0.8,  0.2],
                                     [ 0.1,  0.8,  0.7],
                                     [ 0.9,  0.2,  0.1],
                                     [ 0.1,  0.1,  0.9],
                                     [ 0.2,  0.1,  0.7],
                                     [ 0.3,  0.1,  0.1]])
```

미등록어와 패딩을 위한 인덱스 부여

Word2Vec, GloVe, Random, ...

```
{'deep': 2, 'I': 3, 'flying': 4, 'learning': 5,
 'NLP': 6, 'enjoy': 7, 'like': 8, 'unk': 0, 'pad': 1}
```

Edited by Harksoo Kim

Embedding Layer

임베딩 층 만들기

```
# 입력 문장
input_snt = 'I like football'.split()

# 각 단어를 정수로 변환
idxes=[]

for word in input_snt:
  idx = vocab[word] if word in vocab else vocab['unk']
  idxes.append(idx)

idxes = torch.LongTensor(idxes)

# 입력 문장의임베딩가져오기: ['I', 'like', 'football (unk)']
lookup_result = embedding_table[idxes, :]
print(lookup_result)

# 임베딩 층 만들기
embedding_layer = nn.Embedding(num_embeddings=len(vocab), embedding_dim=3, padding_idx=1)
print(embedding_layer.weight)
```

```
tensor([[0.7000, 0.8000, 0.2000],
        [0.3000, 0.1000, 0.1000],
        [0.0000, 0.0000, 0.0000]])
Parameter containing:
tensor([[ 0.4472, -0.6269,  1.7040],
        [ 0.0000,  0.0000,  0.0000],
        [ 1.5586,  0.9990, -0.3825],
        [-0.9490,  0.0322,  1.6749],
        [-2.0015, -0.1524, -0.7792],
        [ 0.8227,  1.0647,  1.2288],
        [-1.6015, -1.4856,  0.2005],
        [ 0.5156,  0.1298, -1.1945],
        [-0.5552,  0.2856, -1.5769]], requires_grad=True)
```


Edited by Harksoo Kim

질의응답

Q&A

Homepage: http://nlp.konkuk.ac.kr
E-mail: nlpdrkim@konkuk.ac.kr

Edited by Harksoo Kim