Chapter09.
Vectorizing Text & RNN

작성자 : 김진성

목차

1. 텍스트 벡터화(자연어 전처리)

- 텍스트 벡터화(Vectorizing Text)란?
  - ✓ 텍스트를 숫자형 벡터로 변환하는 전처리 과정
  - ✓ 딥러닝 모델은 숫자형만 처리
- 방법
  - ✓ 텍스트 → 단어 → 단어 벡터 변환
  - ✓ 텍스트 → 문자 → 문자 벡터 변환
  - ✓ 텍스트 → N-gram(단어나 문자 그룹) → N-gram 벡터 변환
  - ❖ 토큰(token) : 텍스트를 나누는 단위(단어, 문자, N-gram)
- 벡터 변환 방법(토큰에 숫자형 벡터 연결 방법)
  1) 원-핫 인코딩(희소행렬)
  2) 단어 임베딩(토큰 임베딩)

N-gram : 연속된 단어나 문자의 그룹 단위(텍스트에서 단어나 문자를 하나씩 이동하면서 추출)

1) 원핫인코딩(희소행렬)

2) 단어 임베딩(토큰 임베딩)

Text

"The cat sat one the mat."

Tokens

"the", "cat", "sat", "one", "the", "mat"

Vector encoding of the Tokens

"the" [[0., 1., 0., 0., 0., 0., 0., 0.],
"cat", [0., 0., 1., 0., 0., 0., 0., 0.],
"sat" [0., 0., 0., 1., 0., 0., 0., 0.],
"one" [0., 0., 0., 0., 1., 0., 0., 0.],
"the" [0., 1., 0., 0., 0., 0., 0., 0.],
"mat" [0., 0., 0., 0., 0., 1., 0., 0.]]

Vector encoding of the Tokens

"the"	"cat"	"sat"	"one"	"the"	"mat"
0.0	0.0	0.4	0.0	0.0	1.0
0.5	1.0	0.5	0.2	0.5	0.5
1.0	0.2	1.0	1.0	1.0	0.0

2. 원-핫 인코딩(one-hot encoding)

- ✓ 텍스트를 구성하는 모든 단어에 고유한 정수 인덱스 부여
- ✓ 문서 단어 행렬에서 해당 단어가 출현하면 1, 출현하지 않으면 0으로 표시되는 희소행렬(sparse matrix)

Keras 사용 단어 수준 원-핫 인코딩

```
samples = ['The cat sat on the mat.', 'The dog ate my homework.']
token = Tokenizer()
token.fit_on_texts(samples) # 단어 인덱스 구축
word = token.word_index # 단어 인덱스 객체 : 단어에 고유한 정수 인덱스 부여
Print(word) # { ' 단어 ' : 고유숫자} : 고유숫자 : 단어 출현 순서대로 할당
#{ ' the ' : 1, ' cat ' : 2, ' sat ' : 3, ' on ' : 4, ' mat ' : 5, ' dog ' : 6, ' ate ' : 7, ' my ' : 8, ' homework ' : 9}

# 각 단어 -> 정수 인덱스 변환
sequences = token.texts_to_sequences(samples)
print(sequences)  # [[1, 2, 3, 4, 1, 5], [1, 6, 7, 8, 9]]

# ont-hot encoding(sparse matrix)
one_hot = token.texts_to_matrix(samples, mode='binary')
print(one_hot)
'''
[[0. 1. 1. 1. 1. 1. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 1. 1. 1. 1.]] '''
```

sparse matrix

```
     the cat sat on mat dog ate my  homework
[[0. 1.   1.  1.  1.  1.  0.  0.  0.    0.]
 [0. 1.   0.  0.  0.  0.  1.  1.  1.    1.]]
```

- 원-핫 인코딩 특징

✓ one-hot 벡터들은 표현하고자 하는 단어의 인덱스 값만 1이고, 나머지 인덱스에는 전부 0으로 표현되는 벡터 표현 방법

✓ 벡터 또는 행렬(matrix)의 값이 대부분 0으로 표현되는 방법을 희소 행렬(sparse matrix)이라고 함

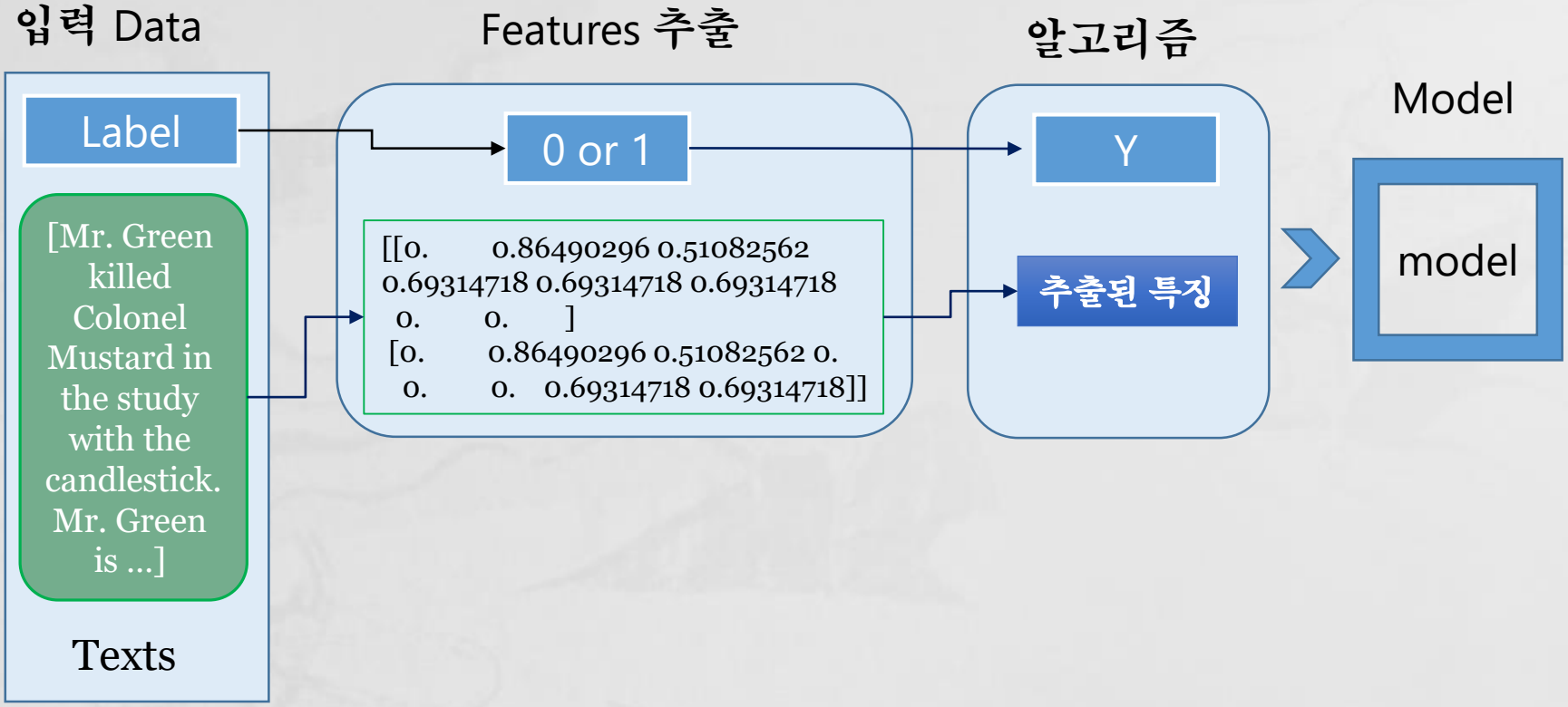
✓ 예)) 단어가 10,000개 있고 '강아지'란 단어의 인덱스는 5였다면 one-hot 벡터의 표현은 다음과 같다.

강아지 벡터 = [0 0 0 0 1 0 0 0 0 0 0 0 ... 0] # 1 뒤의 0의 수는 9,995개

✓ 단어가 많은 경우 고차원(단어수=차원수)

✓ 행렬의 많은 값이 0이 되면서 공간적 낭비

3. 지도학습을 위한 특징 추출



- 텍스트로부터 특징 추출 방법

```
binary = tokenizer.texts_to_matrix(texts=texts, mode='binary')
print(binary) # 단어 출현여부
'''
[[0. 1. 1. 1. 1. 1. 0. 0.]
 [0. 1. 1. 0. 0. 0. 1. 1.]]
'''
count = tokenizer.texts_to_matrix(texts=texts, mode='count')
print(count) # 단어 카운트
'''
[[0. 2. 1. 1. 1. 1. 0. 0.]
 [0. 2. 1. 0. 0. 0. 1. 1.]]
'''
freq = tokenizer.texts_to_matrix(texts=texts, mode='freq')
print(freq)
'''
[[0.         0.33333333 0.2  0.2  0.2  0.2  0.   0.        ]
 [0.         0.4        0.2  0.   0.   0.   0.2  0.2       ]]
'''
tfidf = tokenizer.texts_to_matrix(texts=texts, mode='tfidf')
print(tfidf)
'''
[[0.         0.86490296 0.51082562 0.69314718 0.69314718 0.69314718
  0.         0.         ]
 [0.         0.86490296 0.51082562 0.         0.         0.
  0.69314718 0.69314718]]
'''
```

- 'tfidf' 가중치 적용 희소행렬 활용 예

스팸(spam) 메일 분류기 생성을 위해서 model에 입력할 text 자료를 문서 대비 단어의 출현 비율로 가중치를 적용하여 희소행렬을 만들고, 이를 model의 입력으로 이용한다.

	type	text
0	ham	우리나라 대한민국, 우리나라 만세
1	spam	비아그라 500GRAM 정력 최고!
2	ham	나는 대한민국 사람
3	spam	보험료 15000원에 평생 보장 마감 임박
4	ham	나는 홍길동

DTM 테이블(TF-IDF) 가중치 적용

자연어 숫자 벡터

```
[[ 0.         0.         0.33939315 0.         0.42066906 0.         0.
   0.         0.         0.84133812 0.         0.         0.         0.  0.         0.        ]
 [ 0.5        0.         0.         0.         0.         0.         0.
   0.5        0.         0.         0.         0.         0.5        0.5  0.         0.        ]
 [ 0.         0.53177225 0.53177225 0.         0.         0.         0.
   0.         0.659118   0.         0.         0.         0.         0.  0.         0.        ]
 [ 0.         0.         0.         0.40824829 0.         0.40824829
   0.40824829 0.         0.         0.         0.40824829 0.40824829
   0.         0.         0.40824829 0.         ]
 [ 0.         0.62791376 0.         0.         0.         0.         0.
   0.         0.         0.         0.         0.         0.         0.  0.         0.77828292]]
```

one_hot_encoding = token.texts_to_matrix(texts, mode='tfidf')

4. Word embedding

- ✓ 단어를 밀집 표현으로 변환하는 방법
- ✓ 밀집 단어 벡터를 사용하며, 저차원의 실수형 벡터
- ✓ 적은 차원으로 더 많은 정보를 저장하는 밀집행렬(Dense matrix)
- ✓ 밀집 벡터를 단어 임베딩 과정을 통해 나온 결과라고 하여 임베딩 벡터(embedding vector)
- ✓ 벡터의 차원이 조밀해졌다고 하여 밀집벡터(dense vector)

- 단어 임베딩 특징
  - ✓ 벡터의 차원을 단어 집합의 크기로 표현하지 않고, 사용자가 설정한 값(64, 128, 256, … 1024)으로 단어 벡터의 차원이 결정된다.
  - ✓ 이 과정에서 0과 1의 값이 실수값으로 된다.
  - ✓ 희소벡터 예) 10,000개 단어에서 '강아지' 단어의 인덱스가 5인 경우
    강아지 벡터 = [0 0 0 0 1 0 0 0 0 0 0 0 … 0] # 벡터 차원 : 10000
  - ✓ 밀집벡터 예) 사용자가 밀집 표현의 차원을 128로 설정한다면, 모든 단어의 벡터 표현의 차원은 128로 바뀌면서 벡터 값은 실수가 된다.
    강아지 벡터 = [0.2 1.8 1.1 -2.1 1.1 2.8 … 중략 …] # 벡터 차원 :128
  - ❖ 밀집벡터의 값은 딥러닝의 최적화 알고리즘으로 학습되어 만들어짐

원-핫 벡터 vs 단어 임베딩

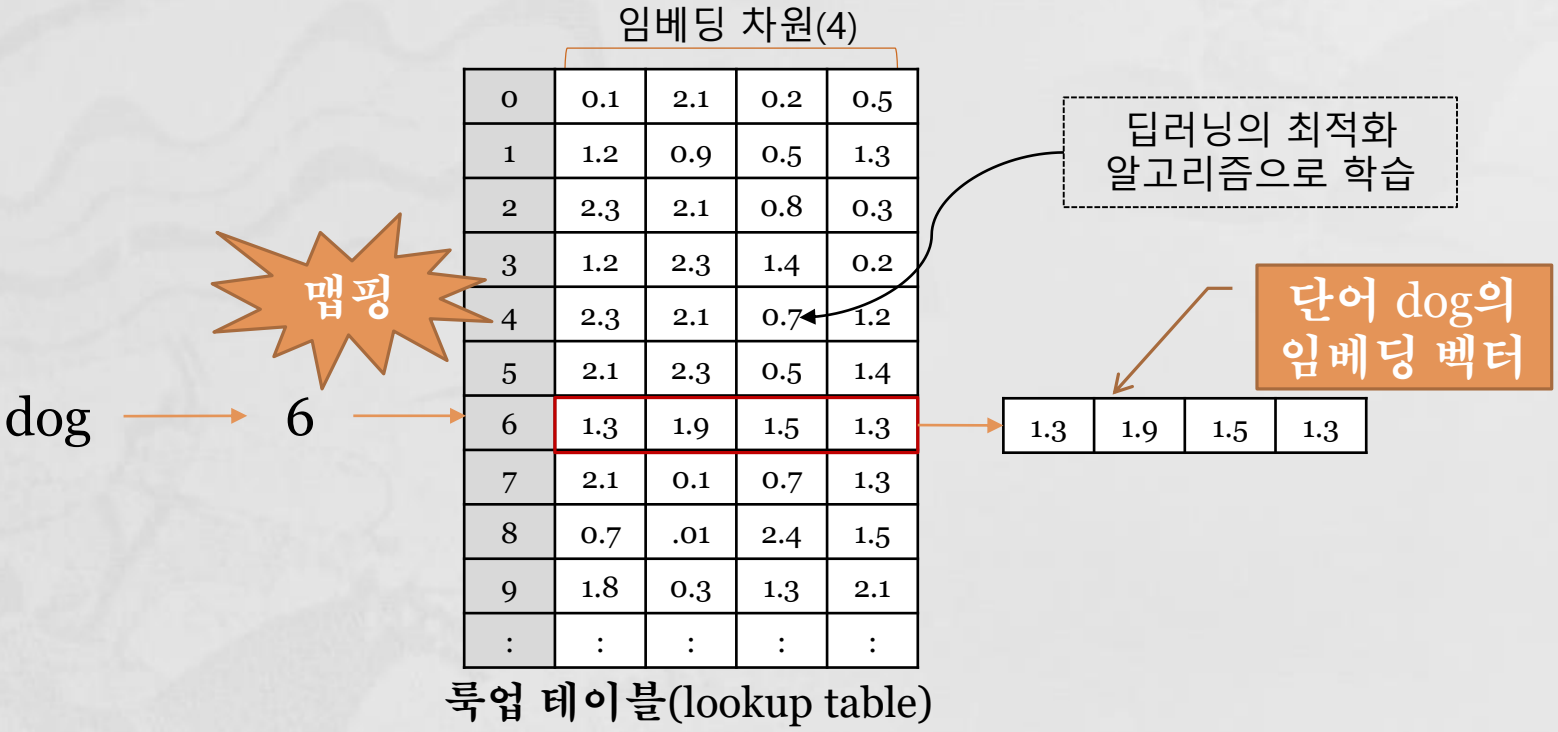
-	원-핫 벡터	임베딩 벡터
차원	고차원(단어 집합의 크기)	저차원
다른 표현	희소 벡터의 일종	밀집 벡터의 일종
표현 방법	수동	훈련 데이터로부터 학습함
값의 타입	1과 0	실수

- 단어 임베딩 처리 과정

임베딩 층 입력으로 사용될 입력 시퀀스의 각 단어들은 모두 정수 인덱스 변환

{ ' the ' : 1, ' cat ' : 2, ' sat ' : 3, ' on ' : 4, ' mat ' : 5, ' dog ' : 6, ' ate ' : 7, ' my ' : 8, ' homework ' : 9, ... }

단어 → 단어 고유 정수 → 임베딩 층 → 임베딩 벡터



Keras 사용 임베딩 층 객체 생성 예

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

# 전체 단어 크기
vocab_size = len(word_index)+1
print('단어 크기: {}'.format((vocab_size))) # 단어 크기: 8630

# 임베딩 차원 : # 32, 64, 128, ...1024차원(단어가 많은 경우)
embedding_dim = 32

# 전체 문장에서 가장 긴 단어 길이
maxlen = max(len(l) for l in x_data) # 171

# 공급 data : sequences
x_data = sequence.pad_sequences(x_data, maxlen=maxlen)
x_data.shape # (5574, 171)

Model = Sequential()
# 임베딩 층
model.add(Embedding(vocab_size, embedding_dim, input_length=maxlen))
# vocab_size=8630 : 문장에서 출현한 전체 단어 크기
# embedding_dim=32 : 임베딩 벡터 차원(길이)
# input_length=maxlen : 임베딩 층으로 입력되는 sequence(정수 인덱스) 길이
```

임베딩 벡터 차원

전체 단어 크기

	0	1	2	3	...	31
0	01	21	02	05	...	12
1	12	09	05	13	...	21
2	23	21	08	03	...	31
3	12	23	14	02	...	21
4	23	21	07	12	...	02
5	21	23	05	14	...	03
:	:	:	:	:	...	:
8629	18	03	13	21	...	24

룩업 테이블(lookup table)

임베딩 층에 sequence 데이터 공급 과정

단어 → 단어 고유 정수 → 임베딩 층 → 임베딩 벡터

정수 인덱스 행렬(sequences)

```
[[  0,   0,   0, ...,  56, 3986,  135],
 [  0,   0,   0, ...,  441,    6, 1766],
 [  0,   0,   0, ..., 2713,  367, 2714],
   ...,
 [  0,   0,   0, ..., 8627,  231, 8628],
 [  0,   0,   0, ...,  198,   12,   47],
 [  0,   0,   0, ...,    1,   40,  242]]
```

(5574, 171)

임베딩 벡터 차원

	0	1	2	3	...	31
0	01	21	02	05	...	12
1	12	09	05	13	...	21
2	23	21	08	03	...	31
56	12	23	14	02	...	21
:	23	21	07	12	...	02
135	23	08	07	23	...	12
:	:	:	;	:	:	:
3986	21	23	05	14	...	03
:	:	:	:	:	...	:
8629	18	03	13	21	...	24

01	21	02	05	...	12
12	23	14	02	...	21
23	08	07	23	...	12
21	23	05	14	...	03

단어

룩업 테이블(lookup table)

❖ 정수 인덱스를 룩업 테이블(밀집 행렬)로 매핑하여 정수와 연관된 벡터를 찾는다.

원-핫 벡터 vs 단어 임베딩 벡터

- 전체 문장 : 5574, 전체 단어 크기 : 8630

원-핫 벡터(희소행렬)

```
[[ 0,  0,  0,  ...,  1,  1, 1],
 [ 0,  0,  0,  ...,  1,  1, 1],
 [ 0,  0,  0,  ...,  1,  1, 1],
 [ 0,  0,  0,  ...,  1,  1, 1],
 [ 0,  0,  0,  ...,  1,  1, 1],
 ...,
 [ 0,  0,  0,  ...,  1,  1, 1],
 [ 0,  0,  0,  ...,  1,  1, 1],
 [ 0,  0,  0,  ...,  1,  1, 1],
 [ 0,  0,  0,  ...,  1,  1, 1],
 [ 0,  0,  0,  ...,  1,  1, 1]]
```

(5574 x 8630)=48,103,620

단어 임베딩 벡터(밀집행렬)

	0	1	2	3	...	31
0	01	21	02	05	...	12
1	12	09	05	13	...	21
2	23	21	08	03	...	31
3	12	23	14	02	...	21
4	23	21	07	12	...	02
5	23	08	07	23	...	12
:	:	:	;	:	:	:
3986	21	23	05	14	...	03
:	:	:	:	:	...	:
8629	18	03	13	21	...	24

(8630 x 32)= 276,160

단어수 = 행

5. RNN 개요

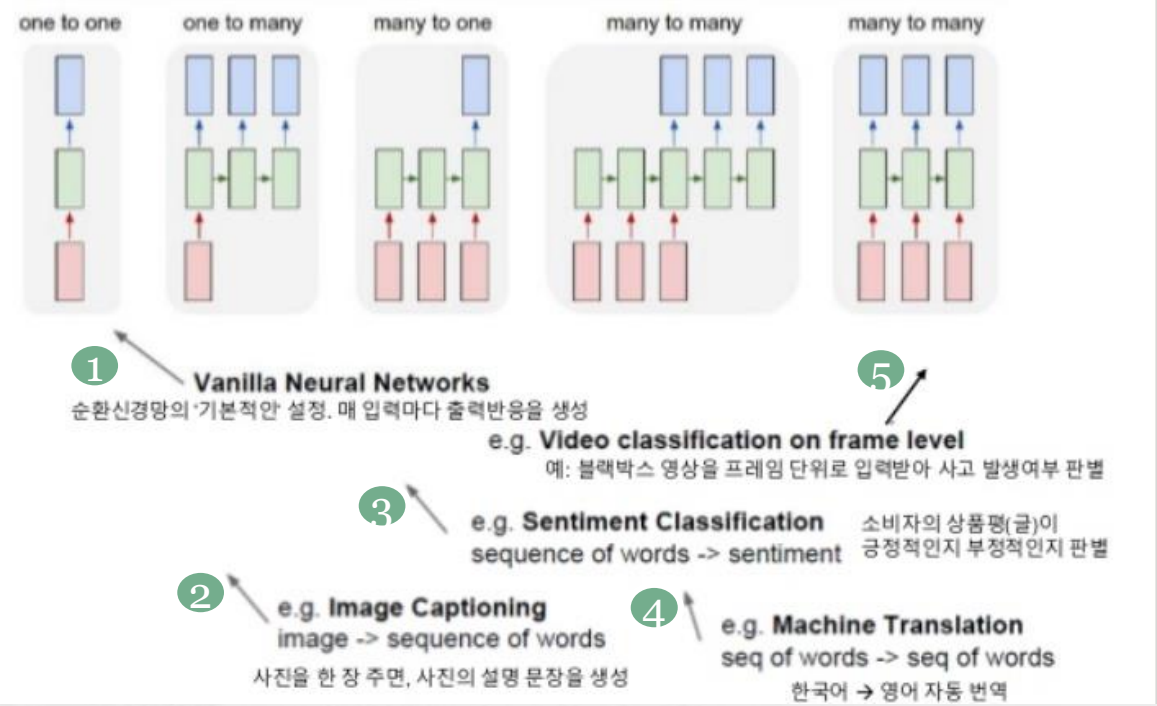
❖ 순환신경망(Recurrent Neural Network : RNN)

✓ 순서가 있는 데이터(text, audio, video)의 패턴을 인식하기 위한 Neural Network

✓ 은닉층 노드 들이 일정한 방향을 가지고 직접연결(directed cycle)된 인공신경망

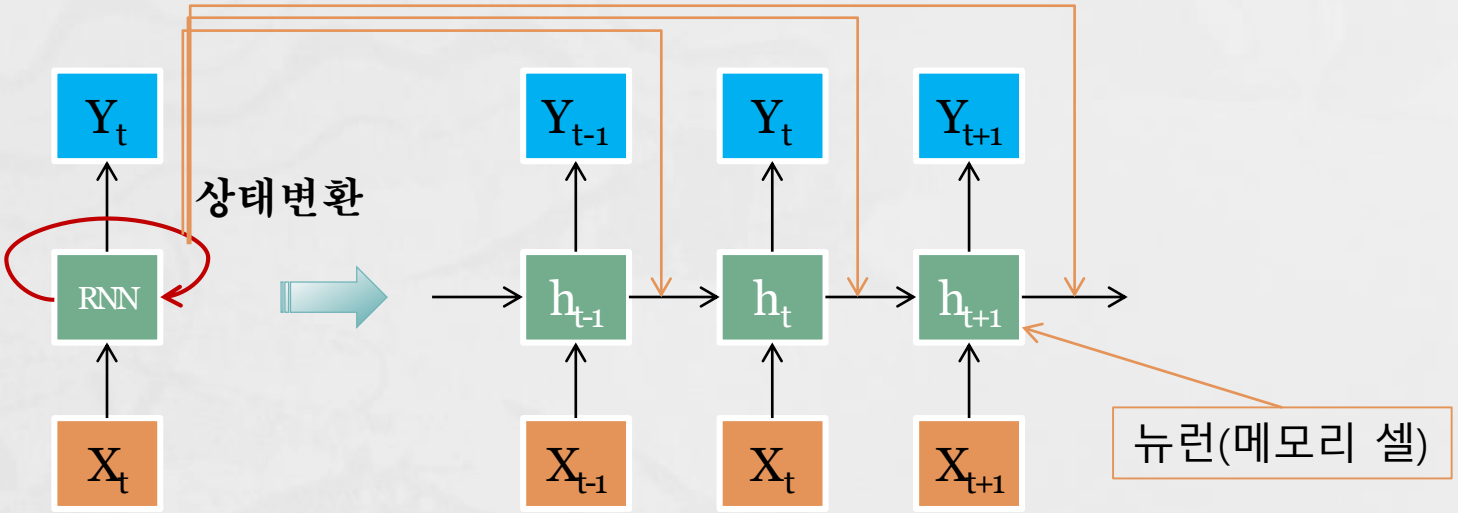
✓ CNN과 더불어 최근 각광 받고 있는 알고리즘

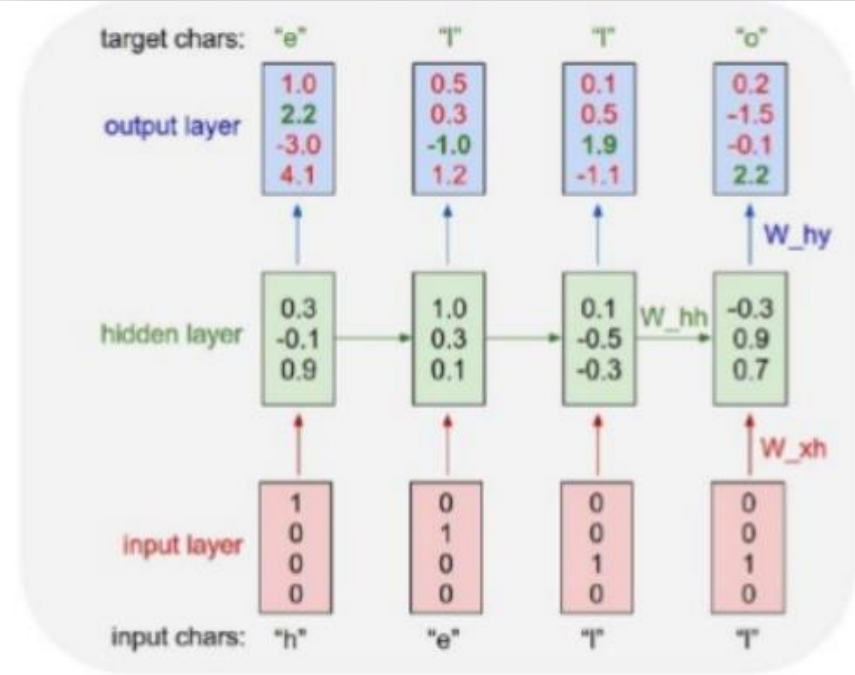
RNN 모델 사례



❖ 순환신경망 계산식

- 활성화 함수는 주로 hyperbolic tangent 함수 사용
- 계산식 : $h_t = \tanh(W*(h_{t-1}, x_t) + b)$
- 어떤 시간(t)의 뉴런(h_t)에는 그 이전 시간(t-1)에 생성된 뉴런(h_{t-1})의 상태가 주입되어 만들어지고, 그 뉴런(h_t)은 다시 그 이후 시간(t+1)에 입력되는 순환구조
- 은닉노드(뉴런)를 메모리 셀(memory cell) or 셀(cell)이라고 부름
- 셀(cell) : 특정 시점에서 상태 데이터(state data)를 저장하는 역할





학습 단계

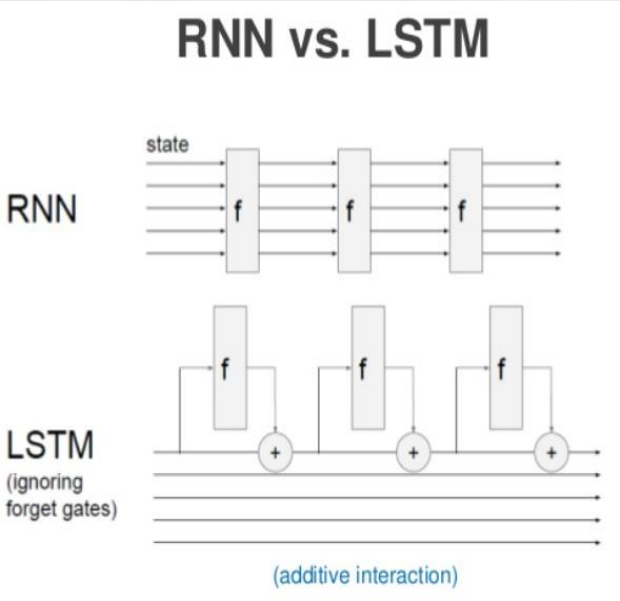
- 출력: 각 글자별 '확신' 정도
- 녹색 값이 높고, 빨간 색 값이 낮도록 매개변수 W_*를 조정(학습)

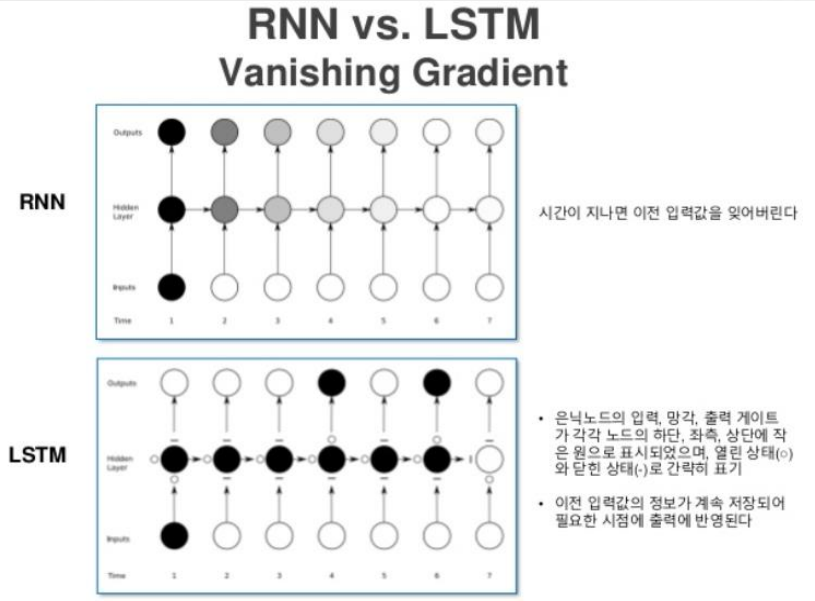
테스트(추론) 단계

- 한 글자를 입력
- 글자별 확률값이 출력됨
- 이 확률분포에서 글자를 하나 샘플링
- 선택한 글자를 다음 입력으로 사용

장단기 메모리(LSTM : Long Short Term Memory)

✓ 기본순환신경망의 확장





6. Keras RNN model

1) 입력 data 생성(데이터 전처리)

```
# 1. 토큰화 : 텍스트 - 토큰(단어) -> 단어 인덱스(고유한 정수 인덱스 부여)
token = Tokenizer() # 가장 빈도가 높은 4000개 단어 선택
token.fit_on_texts(texts) # 토큰 생성기에 문장 넣기
word_index = token.word_index
word_index # 단어 인덱스 -> {'단어' : 고유번호}
print('word length : ', len(word_index)) # word length :  8629

# 전체 단어 크기
vocab_size = len(word_index)+1
print('단어 집합의 크기: {}'.format((vocab_size)))
# 단어 집합의 크기: 8630

# 2. 텍스트 벡터화 : 텍스트(문장) -> 수치형 텐서 변환
sequences = token.texts_to_sequences(texts)

# 3. 입력 시퀀스 생성
x_data = sequence.pad_sequences(x_data, maxlen=maxlen)
x_data.shape # (5574, 171) # 171미만 길이는 0으로 채워짐
```

정수 인덱스 행렬(sequences)

```
[[  0,   0,   0, ...,  56, 3986,  135],
 [  0,   0,   0, ...,  441,    6, 1766],
 [  0,   0,   0, ..., 2713,  367, 2714],
  ...,
 [  0,   0,   0, ..., 8627,  231, 8628],
 [  0,   0,   0, ...,  198,   12,   47],
 [  0,   0,   0, ...,    1,   40,  242]]
```

(5574, 171)

2) Embedding 층

정수 인덱스를 입력 받아서 임베딩 벡터에서 해당 정수와 연관된 벡터를 찾아서 반환 (정수 인덱스 -> Embedding 층 -> 연관 단어 벡터)

```
model = Sequential()
model.add(Embedding(vocab_size, embedding_dim, input_length=maxlen))

#  vocab_size=8630 : 전체 단어 길이
#  embedding_dim=32 : embedding vector 길이(차원)
#  input_length = 171 : 입력 sequence 길이
```

정수 인덱스 행렬(sequences)

```
[[  0,   0,   0, ...,  56, 3986,  135],
 [  0,   0,   0, ...,  441,    6, 1766],
 [  0,   0,   0, ..., 2713,  367, 2714],
   ...,
 [  0,   0,   0, ..., 8627,  231, 8628],
 [  0,   0,   0, ...,  198,   12,   47],
 [  0,   0,   0, ...,    1,   40,  242]]
```

(5574, 171)

룩업 테이블(lookup table)

	0	1	2	3	...	31
0	01	21	02	05	...	12
1	12	09	05	13	...	21
2	23	21	08	03	...	31
56	12	23	14	02	...	21
:	23	21	07	12	...	02
135	23	08	07	23	...	12
:	:	:	;	:	:	:
3986	21	23	05	14	...	03
:	:	:	:	:	...	:
8629	18	03	13	21	...	24

연관 단어 벡터(32차원)

01	21	02	05	...	12
12	23	14	02	...	21
23	08	07	23	...	12
21	23	05	14	...	03

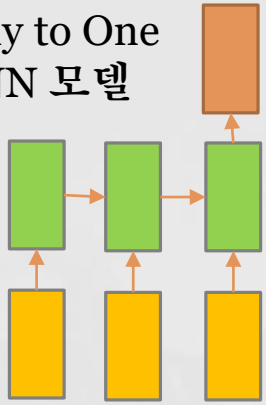
3) LSTM(RNN) 층

임베딩 층에서 반환 받은 32차원의 임베딩 벡터는 LSTM층에서 메모리 셀 32개 뉴런과 연산

LSTM input shape : [batch_size=?, timesteps=32, input_feature=171]

```
# LSTM(RNN)층
model.add(LSTM(32)) # LSTM layer(32 node)
# Fully connected 층 : 분류기
model.add(Dense(32, activation= ' relu ' )) # Affine layer(32 node)
model.add(Dense(1, activation='sigmoid')) # Output layer(1)
```

Layer (type)	Output Shape	Param #
embedding_3 (Embedding)	(None, 171, 32)	276160
lstm_2 (LSTM)	(None, 32)	8320
dense_6 (Dense)	(None, 32)	1056
dense_7 (Dense)	(None, 1)	33



Sequence length=171(입력 시퀀스 길이)
Input size = 32(Embedding vector 차원)
Hidden size = 32(뉴런 길이)

Spam 분류

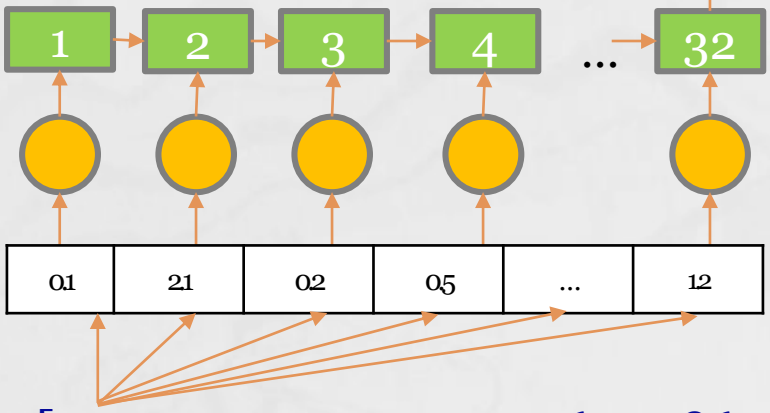
[[0.12109 0.87891]]

Output layer : Sigmoid(0~1 확률값)

Hidden layer : node(cell)=32

Input layer

Embedding vector(32차원 encoding)



[0, 0, 0, ..., 44, ... 56, 3986, 135] 정수 인덱스(Sequence length=171)

['go ... got amore wat'] 텍스트(자연어)