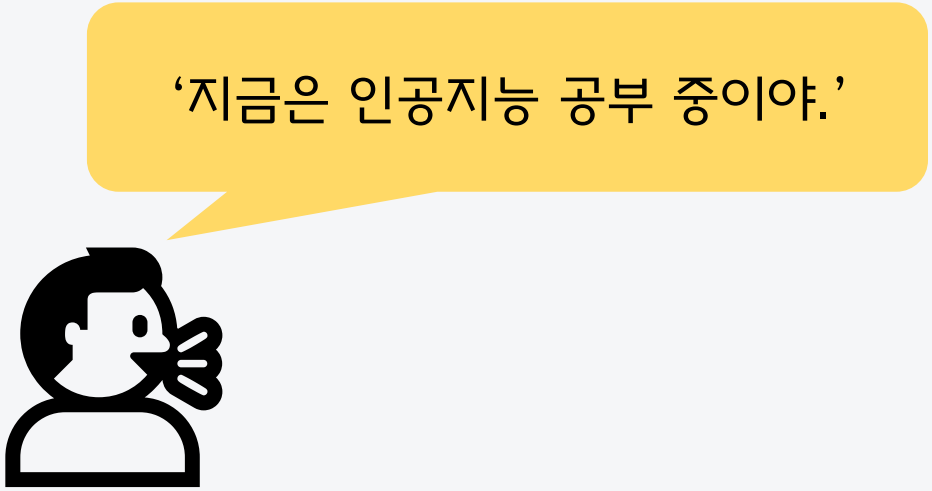

RNN

목차

01

자연어

1. 이론

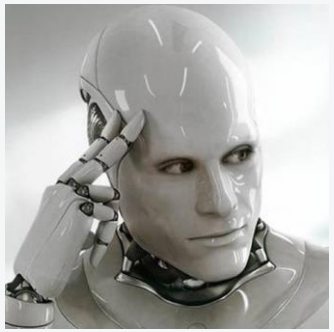


자연어 처리 : 음성이나 텍스트를 컴퓨터가 인식하고 처리하는 것

1. 이론

전처리 과정 필요!

‘지금은 인공지능 공부 중이야.’



1-1. 텍스트 토큰화

토큰

‘지금은/ 인공지능/ 공부/ 중이야.’

토큰화

입력된 텍스트를
잘게 나누는 과정

토큰화1

```
text = '지금은 인공지능 공부 중이야'
result = text_to_word_sequence(text)
print(result)
```

```
['지금은', '인공지능', '공부', '중이야']
```

text_to_word_sequence( )
: 단어 단위로 나눠주는 함수

1-1. 텍스트 토큰화

토큰화2 Tokenizer( ) : 텍스트 전처리 함수

* document_count, word_docs, word_index

```
from tensorflow.keras.preprocessing.text import Tokenizer

docs = ['먼저 텍스트의 각 단어를 나누어 토큰화 합니다.',
        '텍스트의 단어로 토큰화 해야 딥러닝에서 인식됩니다.',
        '토큰화 한 결과는 딥러닝에서 사용 할 수 있습니다.',
       ]

token = Tokenizer()
token.fit_on_texts(docs)
print(token.word_counts)
```

```
OrderedDict([('먼저', 1), ('텍스트의', 2), ('각', 1), ('단어를', 1), ('나누어', 1), ('토큰화', 3), ('합니다',
1), ('단어로', 1), ('해야', 1), ('딥러닝에서', 2), ('인식됩니다', 1), ('한', 1), ('결과는', 1), ('사용', 1),
('할', 1), ('수', 1), ('있습니다', 1)])
```

1-2. 원-핫 인코딩

0인덱스	지금은	인공지능	공부	중이야
0	0	0	0	0

```
from tensorflow.keras.preprocessing.text import Tokenizer

text = "지금은 인공지능 공부 중이야"
token = Tokenizer()
token.fit_on_texts([text])
print(token.word_index)
```

```
{'지금은': 1, '인공지능': 2, '공부': 3, '중이야': 4}
```

1-2. 원-핫 인코딩

```
from tensorflow.keras.utils import to_categorical

token = Tokenizer()
token.fit_on_texts([text])
x = token.texts_to_sequences([text])
print("인덱스로만 채워진 배열")
print(x)
print("₩n")
word_size = len(token.word_index)+1
x = to_categorical(x, num_classes=word_size)
print(x)
```

```
인덱스로만 채워진 배열
[[1, 2, 3, 4]]


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

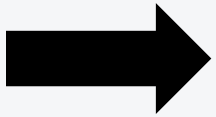


```
[[[0. 1. 0. 0. 0.]      지금은
  [0. 0. 1. 0. 0.]      인공지능
  [0. 0. 0. 1. 0.]      공부
  [0. 0. 0. 0. 1.]]]    중이야
```

1-3. 단어 임베딩

‘지금은 인공지능 공부 중이야.’

1문장



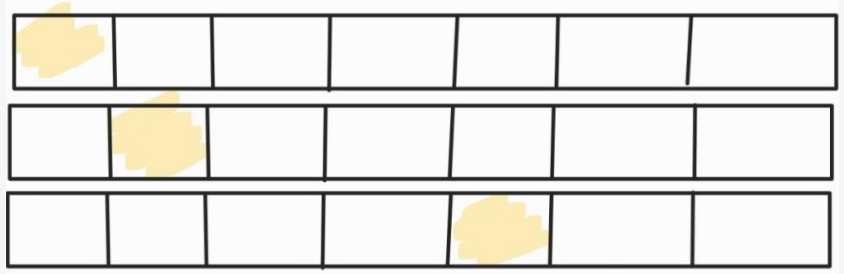
```
[[[0. 1. 0. 0. 0.]
  [0. 0. 1. 0. 0.]
  [0. 0. 0. 1. 0.]
  [0. 0. 0. 0. 1.]]]
```

4개

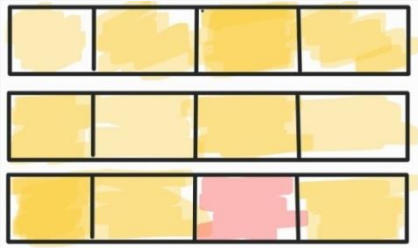
공간 낭비 발생!

1-3. 단어 임베딩

원-핫 인코딩



단어 임베딩



단어 간의 유사도 계산

Embedding( )

Embedding(입력될 총 단어 수, 크기)

+ Embedding(입력될 총 단어 수, 크기, 한 번에 넣을 개수)

1-4. 패딩

‘지금은/ 인공지능/ 공부/ 중이야.’

4개

‘안녕’

1개

Padding : 데이터의 길이를 동일하게 해주는 작업

pad_sequence( )

부족하면 0 넣어서 채워주고, 많으면 잘라서 같은 길이로 맞춤

1-4. 패딩

```
from tensorflow.keras.preprocessing.text import Tokenizer

docs = ['지금은 인공지능 공부 중이야', '안녕']

token = Tokenizer()
token.fit_on_texts(docs)
print(token.word_index)
```

```
{'지금은': 1, '인공지능': 2, '공부': 3, '중이야': 4, '안녕': 5}
```

```
x = token.texts_to_sequences(docs)
print(x)
```

```
[[1, 2, 3, 4], [5]]
```

1-4. 패딩

```python
from tensorflow.keras.preprocessing.sequence import pad_sequences

padded_x = pad_sequences(x, 4)
print("₩n패딩 결과:₩n", padded_x)
```

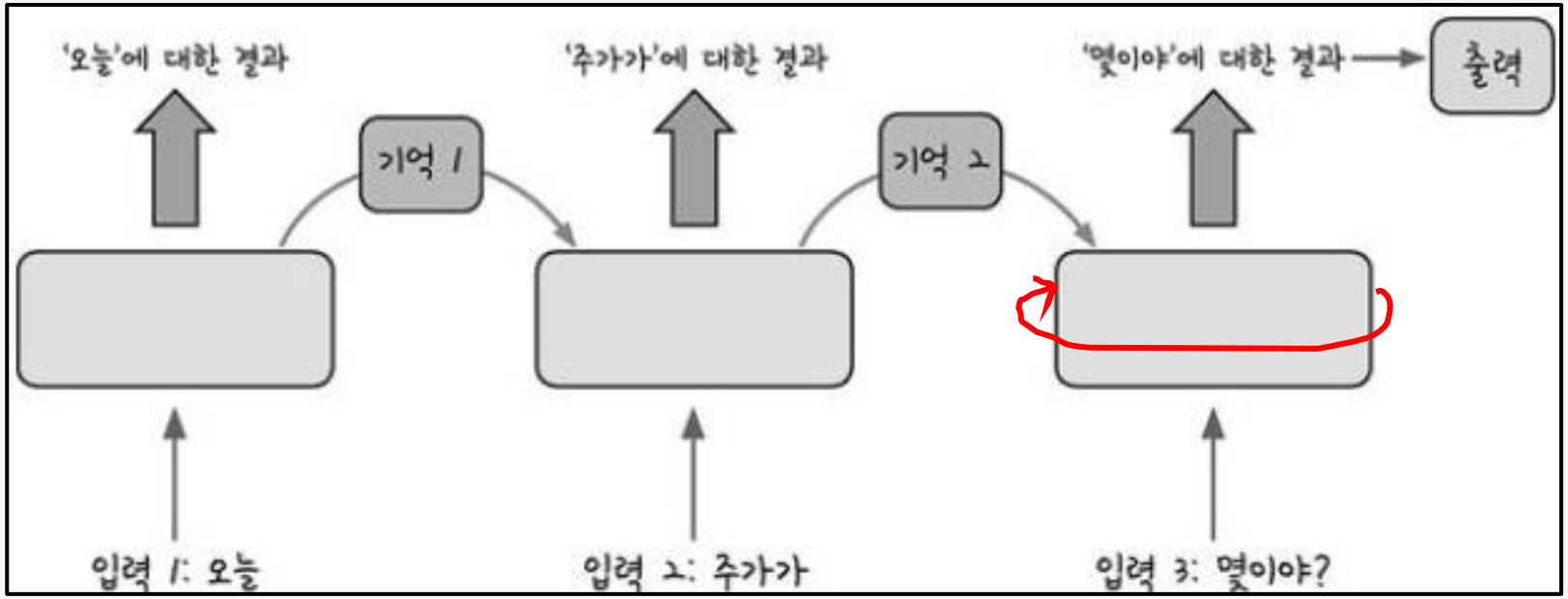
```
패딩 결과:
 [[1 2 3 4]
 [0 0 0 5]]
```

RNN

2-1. RNN

여러 개의 데이터가 순서대로 입력됐을 때 앞서 입력 받은
데이터를 잠시 기억해 놓는 방법

‘오늘 주가가 몇이야?’

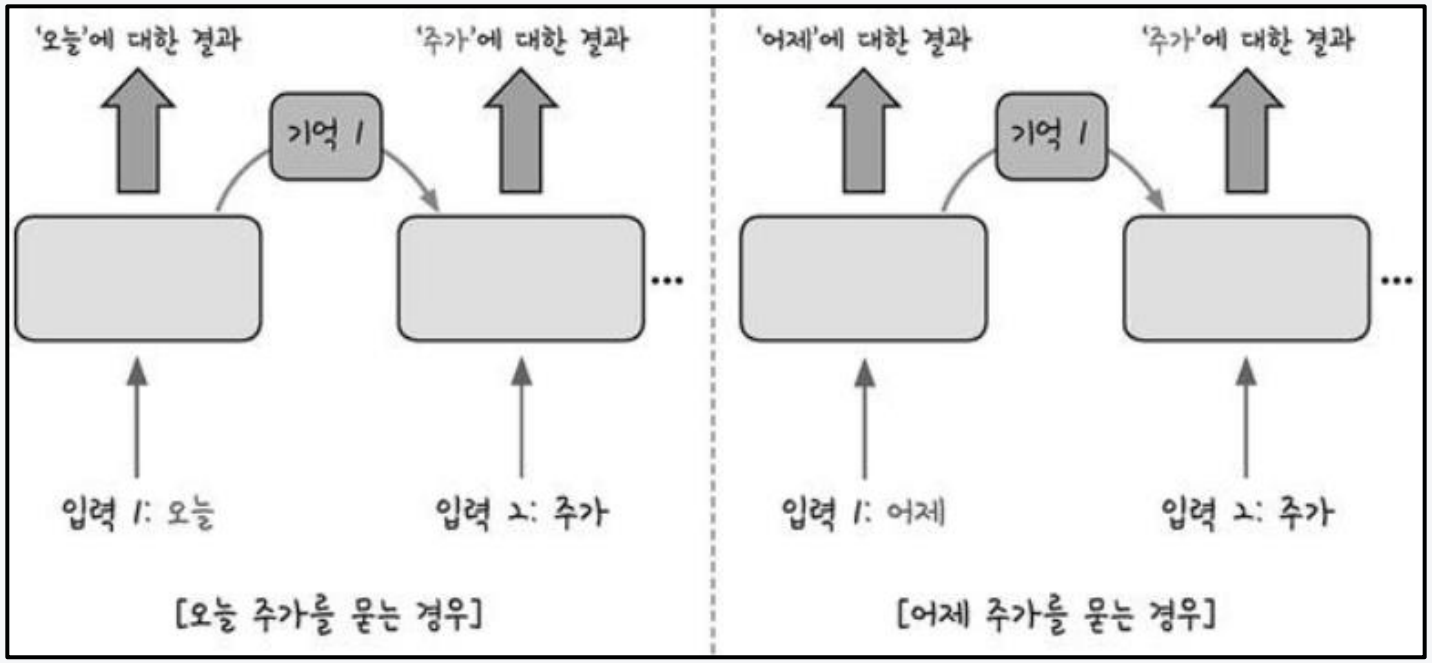


2-1. RNN

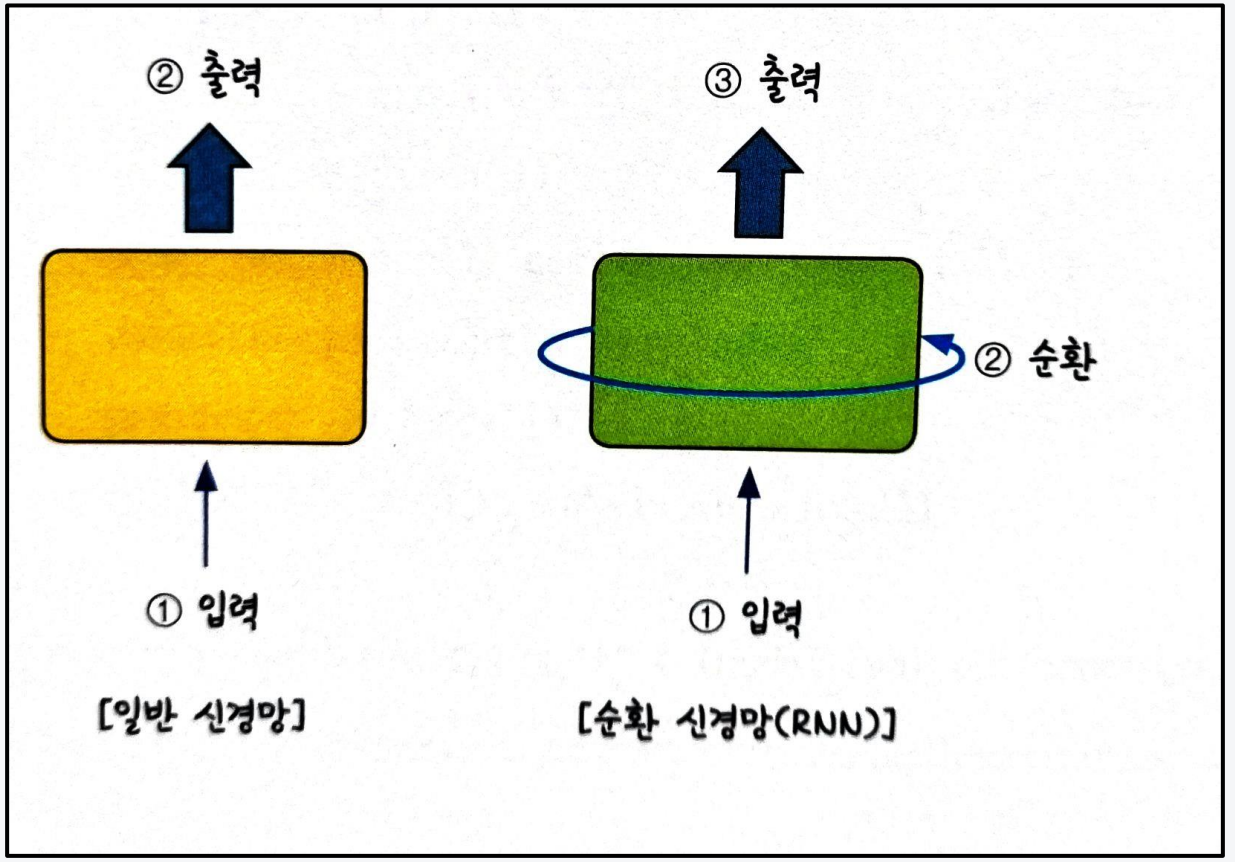
왜 ? 비슷한 두 문장이 입력되었을 때 차이 구별하여 반영 가능

'오늘 주가가 몇이야'

'내일 주가가 몇이야'

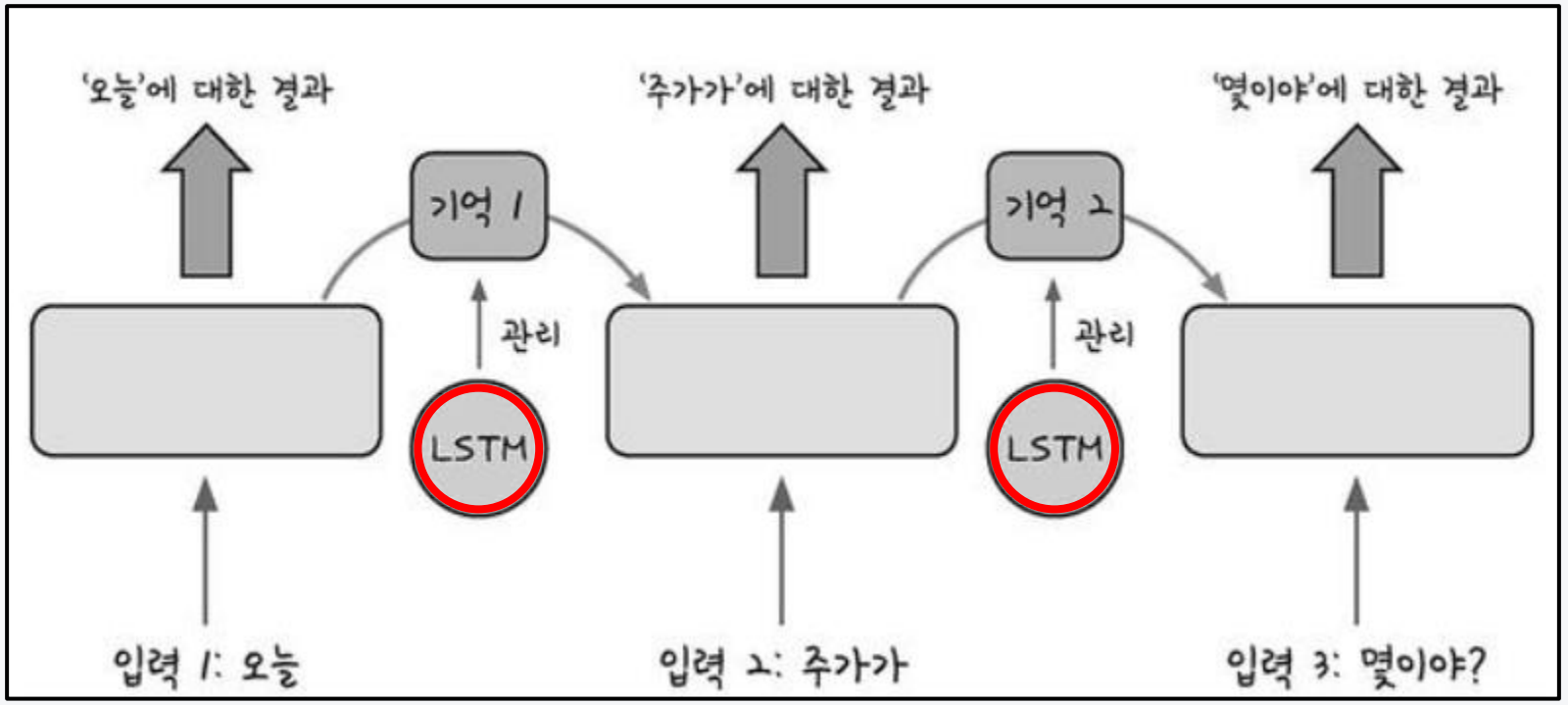


2-1. RNN



한 층 안에서 반복을 많이 해서 일반 신경망보다 기울기 소실 문제 많이 발생

2-2. LSTM



반복되기 직전에 다음 층으로 기억된 값을 넘길지 안 넘길지를 관리하는 단계가 추가

03

구현

3-1. 모델 구현

```
seed = 0
numpy.random.seed(seed)
tf.random.set_seed(3)


(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=5000)


x_train = sequence.pad_sequences(x_train, maxlen=100)
x_test = sequence.pad_sequences(x_test, maxlen=100)


model = Sequential()
model.add(Embedding(5000, 100))

model.add(Dropout(0.5))

model.add(Conv1D(64, 5, padding='valid', activation='relu',strides=1))

model.add(MaxPooling1D(pool_size=4))

model.add(LSTM(55))

model.add(Dense(1))
model.add(Activation('sigmoid'))
model.summary()
```

3-1. 모델 구현

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_4 (Embedding)      (None, None, 100)         500000
_________________________________________________________________
dropout_2 (Dropout)          (None, None, 100)         0
_________________________________________________________________
conv1d_2 (Conv1D)            (None, None, 64)          32064
_________________________________________________________________
max_pooling1d_2 (MaxPooling1 (None, None, 64)          0
_________________________________________________________________
lstm_4 (LSTM)                (None, 55)                26400
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 56
_________________________________________________________________
activation_2 (Activation)    (None, 1)                 0
=================================================================
Total params: 558,520
Trainable params: 558,520
Non-trainable params: 0
```

```
history = model.fit(x_train, y_train, batch_size=100, epochs=20, validation_data=(x_test, y_test))

print("₩n Test Accuracy: %.4f" % (model.evaluate(x_test, y_test)[1]))
```

```
Test Accuracy: 0.8549
```


끝