6장 텍스트와 시퀀스를 위한 딥러닝

박채원, 윤예준, 최혜원

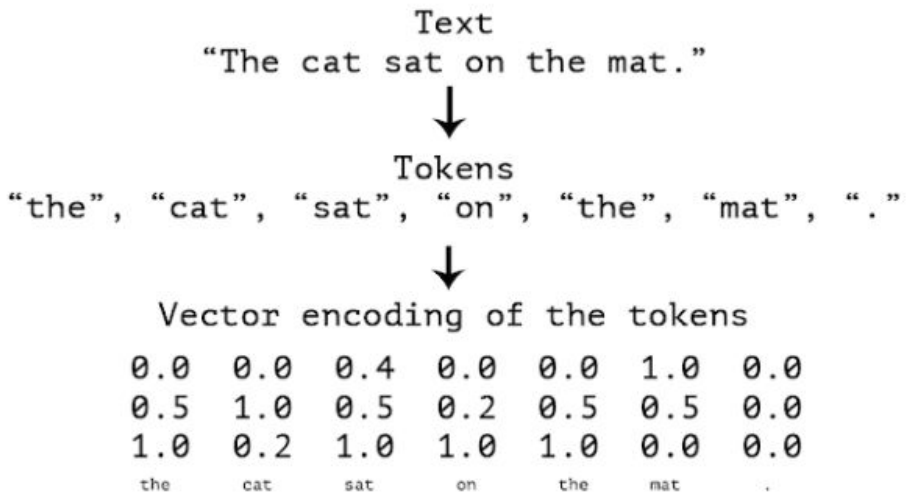
Before

1. 시퀀스 데이터를 처리하는 딥러닝 모델
   - 순환 신경망
   - 1D 컨브넷
2.

   ex) 문서 분류나 시계열 분류-글의 주제나 책의 저자 식별.
   ex)시계열 비교-두 문서나 두 주식 가격이 얼마나 밀접하게 관련 있는지 추정
   ex)시퀀스-투-시퀀스 학습-영어 문장을 프랑스어로 변환
   ex)감성분석 - 트윗이나 영화 리뷰 긍/부정 분류
   ex)시계열 예측- 어떤 지역의 최근 날씨 데이터가 주어졌을 때 향후 날씨 예측.

6.1 텍스트 데이터 다루기

- 시퀀스 처리용 딥러닝 모델 - 문서분류, 감성분석, 저자식별, 질문응답
- 컴퓨터 비전 : 픽셀 / 자연어처리 : 단어, 문장, 문단
- 텍스트 벡터화(단어, 문자, n-gram)



Unigrams:

"The" "purpose" "of" "this" "study" "was" "to" "examine" "the" "incidence" "of" "breast" "cancer" "with" "triple" "negative" "phenotype".

Bigrams:

"The purpose" "of this" "study was" "to examine" "the incidence" "of breast" "cancer with" "triple negative" phenotype.

"purpose of" "this study" "was to" "examine the" "incidence of" "breast cancer" "with triple" "negative phenotype".

6.1 텍스트 데이터 다루기

1. 단어와 문자의 **원-핫 인코딩**
   - 모든 단어에 고유한 정수 인덱스(i)를 부여하고 이를 크기가 N(어휘사전의 크기)인 이진 벡터로 변환.
   - i 번째 원소만 1, 나머지는 0

```python
import numpy as np

samples = ['The cat sat on the mat.', 'The dog ate my homework.']
token_index = {} #dictionary
for sample in samples:
    for word in sample.split():
        if word not in token_index:
            token_index[word] = len(token_index) + 1
max_length = 10
print(token_index)
results = np.zeros(shape=(len(samples),
                          max_length,
                          max(token_index.values()) + 1
                          ))
for i, sample in enumerate(samples):
    for j, word in list(enumerate(sample.split()))[:max_length]:
        index = token_index.get(word)
        results[i, j, index] = 1
```

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

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

```
{'The': 1, 'cat': 2, 'sat': 3, 'on': 4, 'the': 5, 'mat.': 6, 'dog': 7, 'ate': 8, 'my': 9, 'homework.': 10}
```

```python
import string

samples =  ['The cat sat on the mat.', 'The dog ate my homework.']
characters = string.printable  출력가능한모든아스키문자
token_index = dict(zip(characters, range(1, len(characters)+1)))

max_length = 50

results = np.zeros((len(samples), max_length, max(token_index.values()) + 1))
for i, sample in enumerate(samples):
    for j, character in enumerate(sample):
        index = token_index.get(character)
        results[i, j, index] = 1
print(token_index)
print(results.shape)
print(results)
```

```
{'0': 1, '1': 2, '2': 3, '3': 4, '4': 5, '5': 6, '6': 7, '7': 8, '8': 9, '9': 10, 'a': 11, 'b': 12, 'c': 13, 'd': 14, 'e': 15, 'f': 16, 'g': 17, 'h': 18, 'i': 19,
(2, 50, 101)
[[[0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  ...
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]]

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

6.1 텍스트 데이터 다루기

케라스를 이용한 원-핫 인코딩

```
#빈도가 높은 1000개의 단어만 선택하도록
from keras.preprocessing.text import Tokenizer

samples = ['The cat sat on the mat.', 'The dog ate my homework.']

tokenizer = Tokenizer(num_words = 1000)
tokenizer.fit_on_texts(samples)

sequences = tokenizer.texts_to_sequences(samples)

one_hot_results = tokenizer.texts_to_matrix(samples, mode='binary')

word_index = tokenizer.word_index
print('%s개의 고유한 토큰을 찾았습니다.' %len(word_index))
```

9개의 고유한 토큰을 찾았습니다.

6.1 텍스트 데이터 다루기

원-핫 해싱(hashing)-토큰의 수가 너무 커서 모두 다루기 어려울 때

단점 : 해시충돌 발생 가능성

```
#one-hot hashing
dimensionality = 1000
max_length = 1000
results = np.zeros((len(samples), max_length, dimensionality))
print(results.shape)
for i, sample in enumerate(samples):
    for j, word in list(enumerate(sample.split()))[:max_length]:
        index = abs(hash(word)) % dimensionality
        results[i, j, index] = 1
print(results)
```

Hash 값 출력->

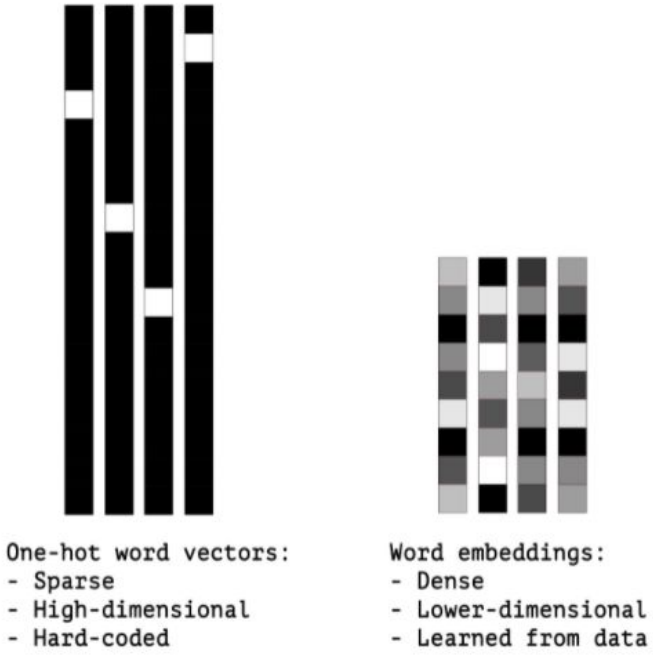
```
-365337694250988253
2806440617842615289
4694832480488194465
-625288842643541692
8363611396915874874
5088647705162010833
-365337694250988253
3092896201862689518
-2978621971514307076
7626054940765646618
-1334737204159395989
```

```
(2, 1000, 1000)
[[[0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  ...
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]]

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

Hash :
https://kadensungbincho.tistory.com/23

6.1 텍스트 데이터 다루기

2. **단어 임베딩** 사용하기
   - 원-핫 인코딩 : 고차원, sparse(희소)함(대부분 0)
   - 단어 임베딩 : 저차원, 실수형 벡터(밀집)
     a. 관심 대상인 문제에서 단어 임베딩 학습
     b. 사전 훈련된 단어 임베딩 로드

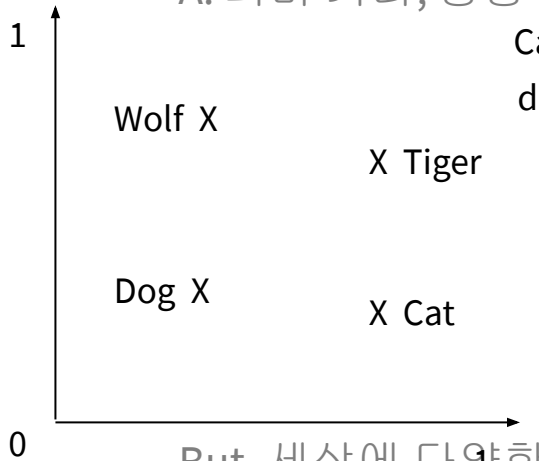


6.1 텍스트 데이터 다루기

2. **단어 임베딩** 사용하기
   a. **Embedding 층을 사용하여 단어 임베딩 학습하기**

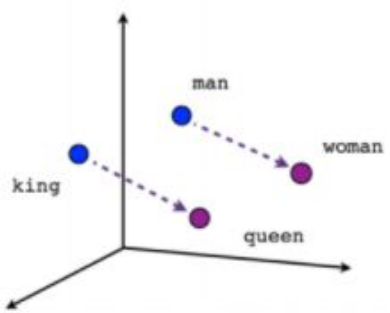
   Q. 랜덤 매칭 -> 임베딩 공간이 구조적이지 않다. (accurate, exact : 비슷한 의미, 다른 임베딩)

   A. 의미 거리, 방향 반영하여 임베딩



Cat -> tiger/ dog-> wolf : 애완동물 -> 야생동물

dog-> cat / wolf-> tiger : 개과-> 고양이과



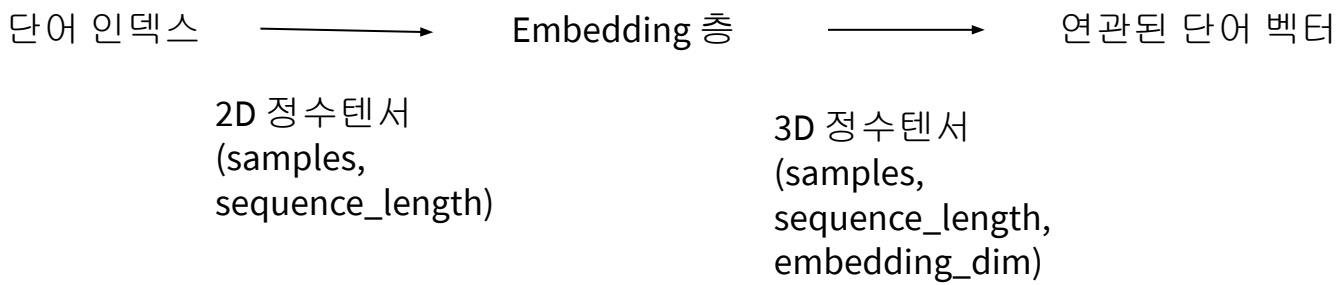
Male-Female

But, 세상에 다양한 언어, 특정문화, 환경 => 문제, 상황에 맞게 임베딩 best

6.1 텍스트 데이터 다루기

2. **단어 임베딩** 사용하기
   a. **Embedding 층을 사용하여 단어 임베딩 학습하기**
   - 일반화된 임베딩 공간은 존재하지 않음.
   - 새로운 작업에는 새로운 임베딩을 학습해야.-> Keras이용

```
1   from keras.layers import Embedding
2   embedding_layer = Embedding(1000, 64)
```



6.1 텍스트 데이터 다루기

2. **단어 임베딩** 사용하기
   a. **Embedding 층을 사용하여 단어 임베딩 학습하기**
   -IMDB에 적용

```
1   from keras.datasets import imdb
2   from keras import preprocessing
3   max_feature = 10000 단어 수
4   maxlen = 20   리뷰에서 20개가 넘는 단어는 버림
5
6   (x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_feature)
7
8   x_train = preprocessing.sequence.pad_sequences(x_train, maxlen=maxlen)
9   x_test = preprocessing.sequence.pad_sequences(x_test, maxlen=maxlen)
10
```

6.1 텍스트

2. 단어 임베딩

   a. **Embedd**

   -IMDB에

```
1  from keras.models import Sequen
2  from keras.layers import Flatte
3
4  model = Sequential()
5  model.add(Embedding(10000, 8, i
6
7  model.add(Flatten())
8
9  model.add(Dense(1, activation='
10 model.compile(optimizer='rmspro
11 model.summary()
12
13 history=model.fit(x_train, y_tr
14                  epochs=10,
15                  batch_size =
16                  validation_sp
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_3 (Embedding)      (None, 20, 8)             80000
_________________________________________________________________
flatten (Flatten)            (None, 160)               0
_________________________________________________________________
dense (Dense)                (None, 1)                 161
=================================================================
Total params: 80,161
Trainable params: 80,161
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
625/625 [==============================] - 14s 2ms/step - loss: 0.6838 - acc: 0.5680 - val_loss: 0.6094 - val_acc: 0.6994
Epoch 2/10
625/625 [==============================] - 1s 1ms/step - loss: 0.5605 - acc: 0.7467 - val_loss: 0.5206 - val_acc: 0.7280
Epoch 3/10
625/625 [==============================] - 1s 1ms/step - loss: 0.4654 - acc: 0.7847 - val_loss: 0.4985 - val_acc: 0.7460
Epoch 4/10
625/625 [==============================] - 1s 2ms/step - loss: 0.4217 - acc: 0.8095 - val_loss: 0.4947 - val_acc: 0.7476
Epoch 5/10
625/625 [==============================] - 1s 1ms/step - loss: 0.3954 - acc: 0.8260 - val_loss: 0.4942 - val_acc: 0.7528
Epoch 6/10
625/625 [==============================] - 1s 1ms/step - loss: 0.3721 - acc: 0.8366 - val_loss: 0.4973 - val_acc: 0.7544
Epoch 7/10
625/625 [==============================] - 1s 1ms/step - loss: 0.3527 - acc: 0.8461 - val_loss: 0.5011 - val_acc: 0.7556
Epoch 8/10
625/625 [==============================] - 1s 1ms/step - loss: 0.3398 - acc: 0.8560 - val_loss: 0.5078 - val_acc: 0.7546
Epoch 9/10
625/625 [==============================] - 1s 2ms/step - loss: 0.3243 - acc: 0.8629 - val_loss: 0.5134 - val_acc: 0.7520
Epoch 10/10
625/625 [==============================] - 1s 1ms/step - loss: 0.3115 - acc: 0.8700 - val_loss: 0.5204 - val_acc: 0.7536
```

6.1 텍스트 데이터 다루기

2. **단어 임베딩** 사용하기
    b. **사전 훈련된 단어 임베딩 사용하기**
    ex)자연어 처리할 경우-일반적 특성이 요구됨
    - Word2vec(성별, 같은 구체적인 의미있는 속성을 캐치)
    - GloVe(단어의 동시출현 통계를 기록한 행렬을 분해하는 기법)

6.1 텍스트 데이터 다루기

3. 모든 내용 적용 : 원본텍스트에서 단어 임베딩까지
   a. 원본 **IMDB** 텍스트 내려받기

```
import os

imdb_dir = 'drive/MyDrive/datasets/aclImdb'
train_dir = os.path.join(imdb_dir, 'train')
labels = []
texts = []

for label_type in ['neg', 'pos']:
    dir_name = os.path.join(train_dir, label_type)
    for fname in os.listdir(dir_name):
        if fname[-4:] == '.txt':
            f = open(os.path.join(dir_name, fname), encoding='utf8')
            texts.append(f.read())
            f.close()
            if label_type == 'neg':
                labels.append(0)
            else:
                labels.append(1)
```

6.1 텍스트 데이터 다루기

3. 모든 내용 적용 : 원본텍스트에서 단어 임베딩까지
   b. 데이터 토큰화

```
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
import numpy as np

maxlen = 100
training_samples = 200
validation_samples = 10000
max_words = 10000

tokenizer = Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(texts)
sequences = tokenizer.texts_to_sequences(texts)

word_index = tokenizer.word_index
print('%s개의 고유한 토큰을 찾았습니다. ' %len(word_index))
```

```
data = pad_sequences(sequences, maxlen=maxlen)
labels = np.asarray(labels)
print('데이터 텐서의 크기: ', data.shape)
print('레이블 텐서의 크기: ', labels.shape)

indices = np.arange(data.shape[0])
np.random.shuffle(indices)
data = data[indices]
labels = labels[indices]

x_train = data[:training_samples]
y_train = labels[:training_samples]
x_val = data[training_samples: training_samples + validation_samples]
y_val = labels[training_samples: training_samples + validation_samples]
```

6.1 텍스트 데이터 다루기

3. 모든 내용 적용 : 원본텍스트에서 단어 임베딩까지
   b. **GloVe** 단어 임베딩 내려받기+임베딩 전처리

```
glove_dir = './datasets/'

embeddings_index = {}
f = open(os.path.join(glove_dir, 'glove.6B.100d.txt'),
encoding='utf8')
for line in f:
    values = line.split()
    word = values[0]
    coefs = np.asarray(values[1:], dtype='float32')
    embedding_index[word] = coefs
f.close()

print('%s개의 단어 벡터를 찾았습니다. ' % len(embedding_index))
```

```
1  embedding_dim = 100
2
3  embedding_matrix = np.zeros((max_words, embedding_dim))
4  for word, i in word_index.items():
5      if i < max_words:
6          embedding_vector = embeddings_index.get(word)
7          if embedding_vector is not None:
8              embedding_matrix[i] = embedding_vector
```

6.1 텍스트 데이터 다루기

3. 모든 내용 적용 : 원본텍스트에서 단어 임베딩까지
   c. 모델 정의하기

```
1  from keras.models import Sequential
2  from keras.layers import Embedding, Flatten, Dense
3
4  model = Sequential()
5  model.add(Embedding(max_words, embedding_dim, input_length=maxlen))
6  model.add(Flatten())
7  model.add(Dense(32, activation='relu'))
8  model.add(Dense(1, activation='sigmoid'))
9  model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_4 (Embedding)      (None, 100, 100)          1000000
_________________________________________________________________
flatten_1 (Flatten)          (None, 10000)             0
_________________________________________________________________
dense_1 (Dense)              (None, 32)                320032
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 33
=================================================================
Total params: 1,320,065
Trainable params: 1,320,065
Non-trainable params: 0
_________________________________________________________________
```

6.1 텍스트 데이터 다루기

3. 모든 내용 적용 : 원본텍스트에서 단어 임베딩까지
   d. 모델에 **GloVe** 임베딩 로드하기 + 훈련과 평가

```
model.layers[0].set_weights([embedding_matrix])
```
가중치 설정

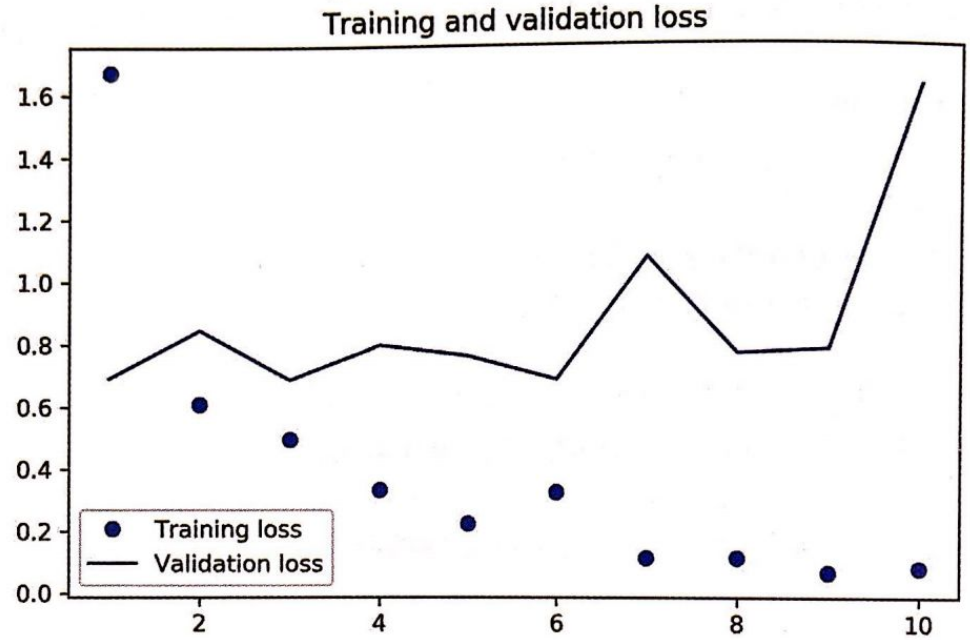
```
model.layers[0].trainable = False
```
Trainable = false로 만드는 이유 : 훈련하는 동안 사전훈련된 부분이 업데이트 되어 정보를 잃는 것을 막기 위해

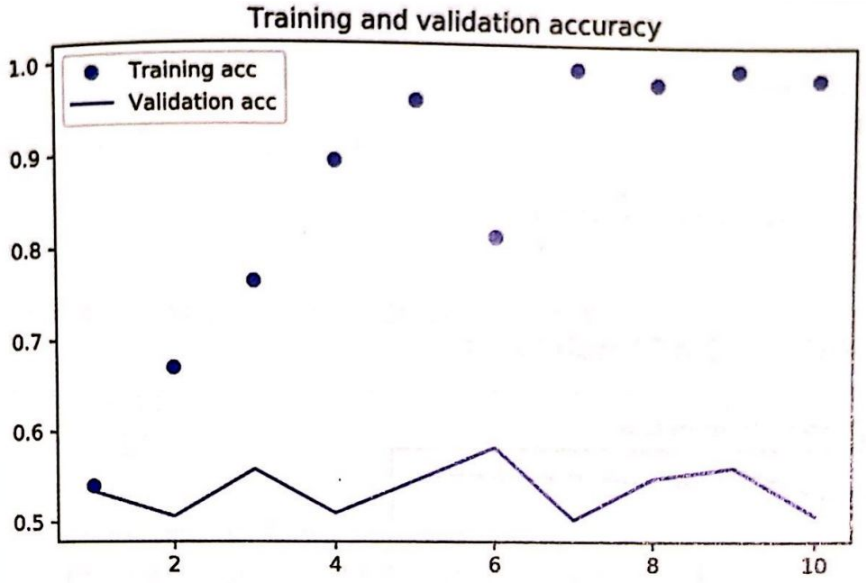
```
model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrices=['acc'])
history = model.fit(x_train, y_train,
                    epochs = 10,
                    batch_size = 32,
                    validation_data=(x_val, y_val))
model.save_weights('pre_trained_glove_model.h5')
```

모델 훈련

6.1 텍스트 데이터 다루기

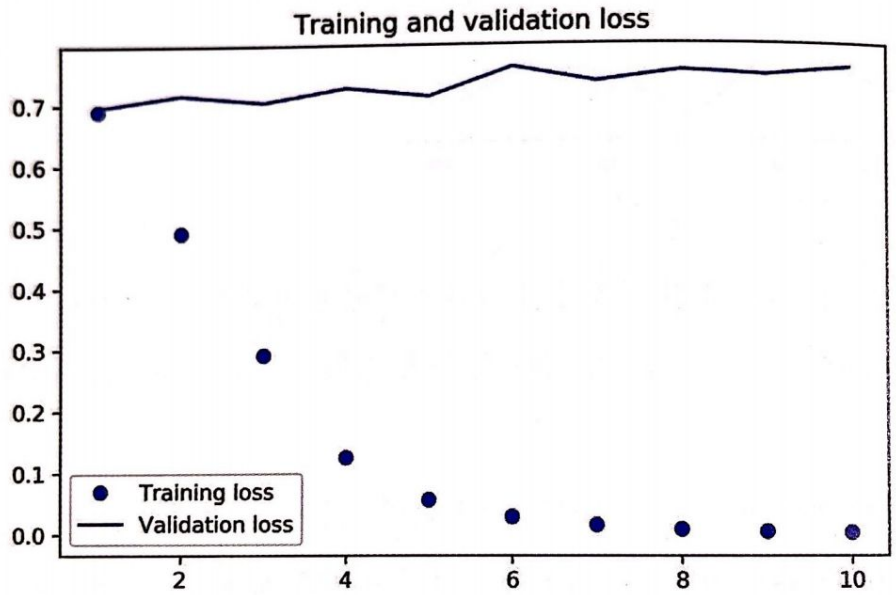
3. 모든 내용 적용 : 원본텍스트에서 단어 임베딩까지
   e. 결과 그래프

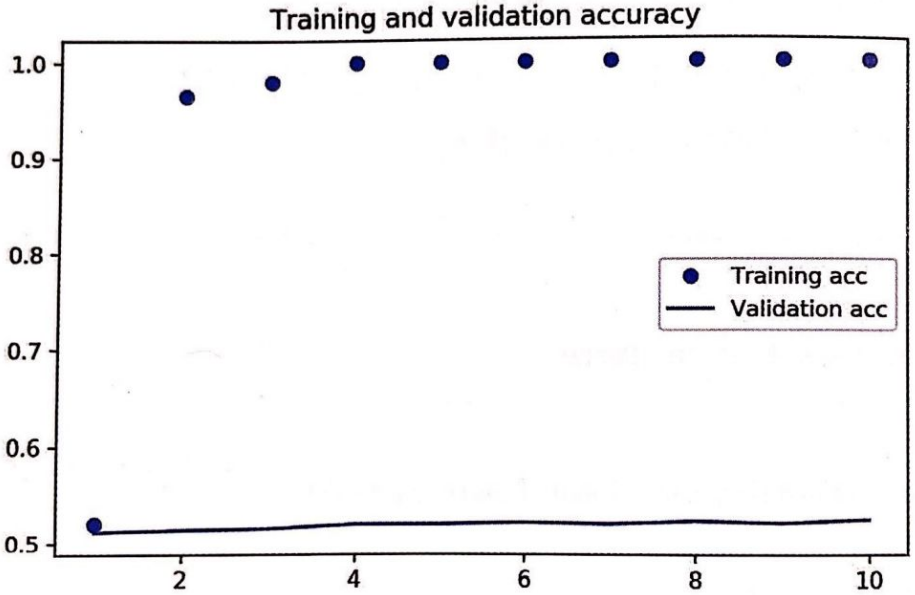




6.1 텍스트 데이터 다루기

3. 모든 내용 적용 : 원본텍스트에서 단어 임베딩까지
   f. 결과 개선 시도 - 사전훈련된 단어 임베딩 사용 x
      -> 해당 작업에 특화된 입력토큰의 임베딩 학습.





6.1 텍스트 데이터 다루기

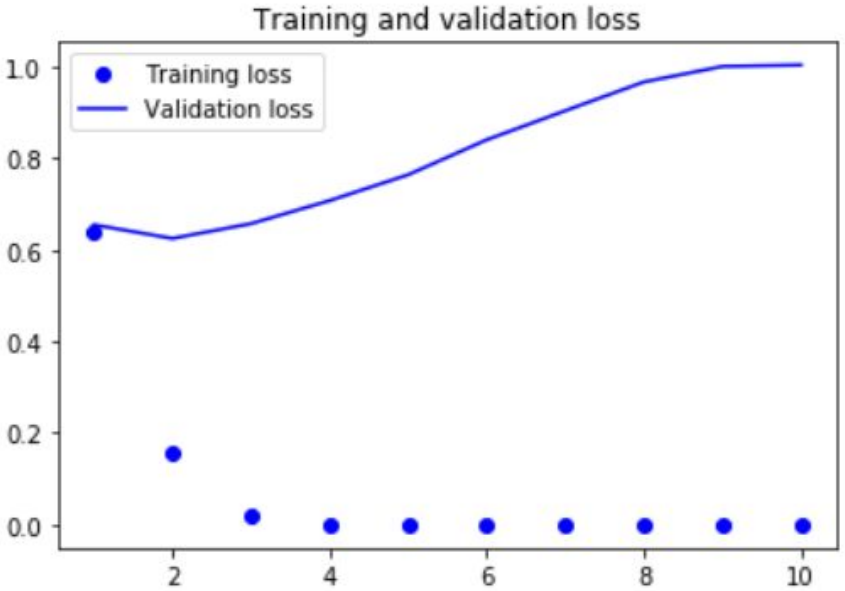
In [17]:
```
training_samples = 2000
x_train = data[:training_samples]
y_train = labels[:training_samples]
x_val = data[training_samples: training_samples + validation_samples]
y_val = labels[training_samples: training_samples + validation_samples]
```

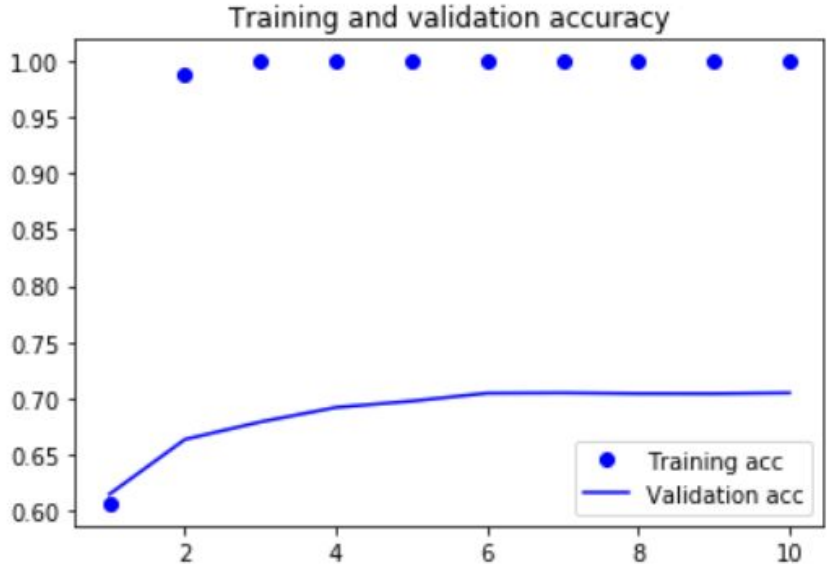
In [18]:
```
history = model.fit(x_train, y_train,
                    epochs=10,
                    batch_size=32,
                    validation_data=(x_val, y_val))
```

```
Train on 2000 samples, validate on 10000 samples
Epoch 1/10
2000/2000 [==============================] - 0s 181us/step - loss: 0.6383 - acc: 0.6060 - val_loss: 0.6543 - val_acc: 0.6142
Epoch 2/10
2000/2000 [==============================] - 0s 177us/step - loss: 0.1578 - acc: 0.9880 - val_loss: 0.6246 - val_acc: 0.6631
Epoch 3/10
2000/2000 [==============================] - 0s 169us/step - loss: 0.0191 - acc: 0.9995 - val_loss: 0.6568 - val_acc: 0.6787
Epoch 4/10
2000/2000 [==============================] - 0s 168us/step - loss: 0.0016 - acc: 1.0000 - val_loss: 0.7071 - val_acc: 0.6916
Epoch 5/10
2000/2000 [==============================] - 0s 167us/step - loss: 1.2005e-04 - acc: 1.0000 - val_loss: 0.7635 - val_acc: 0.6972
Epoch 6/10
2000/2000 [==============================] - 0s 174us/step - loss: 8.1684e-06 - acc: 1.0000 - val_loss: 0.8398 - val_acc: 0.7043
Epoch 7/10
2000/2000 [==============================] - 0s 172us/step - loss: 7.4167e-07 - acc: 1.0000 - val_loss: 0.9032 - val_acc: 0.7046
Epoch 8/10
2000/2000 [==============================] - 0s 171us/step - loss: 1.7394e-07 - acc: 1.0000 - val_loss: 0.9660 - val_acc: 0.7040
Epoch 9/10
2000/2000 [==============================] - 0s 177us/step - loss: 1.1575e-07 - acc: 1.0000 - val_loss: 0.9998 - val_acc: 0.7039
Epoch 10/10
2000/2000 [==============================] - 0s 173us/step - loss: 1.1111e-07 - acc: 1.0000 - val_loss: 1.0030 - val_acc: 0.7046
```

6.1 텍스트 데이터 다루기

3. 모든 내용 적용 : 원본텍스트에서 단어 임베딩까지
   f. 결과 개선 시도 1 - 사전훈련된 단어 임베딩 사용 x
      -> 해당 작업에 특화된 입력토큰의 임베딩 학습.





6.1 텍스트 데이터 다루기

3. 모든 내용 적용 : 원본텍스트에서 단어 임베딩까지
    g. 테스트 데이터 토큰화 -> 모델 평가

```
import os

labels = []
texts = []

for label_type in ['neg', 'pos']:
    dir_name = os.path.join(train_dir, label_type)
    for fname in sorted(os.listdir(dir_name)):
        if fname[-4:] == '.txt':
            f = open(os.path.join(dir_name, fname), encoding='utf8')
            texts.append(f.read())
            f.close()
            if label_type == 'neg':
                labels.append(0)
            else:
                labels.append(1)
sequences = tokenizer.texts_to_sequences(texts)
x_test = pad_sequences(sequences, maxlen=maxlen)
y_test = np.asarray(labels)
```

```
model.load_weights('pre_trained_glove_model.h5')
model.evaluate(x_test, y_test)
```

```
25000/25000 [==============================] - 0s 19us/step
[1.6611276818060874, 0.50412]
```

6.1 텍스트 데이터 다루기

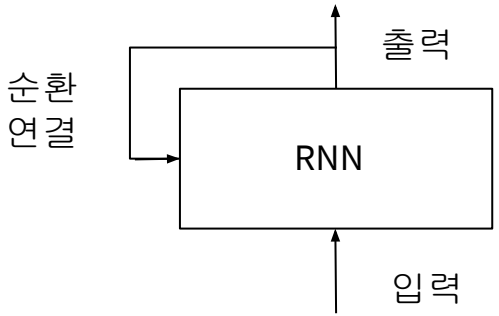
4. **모든 내용 적용 : 원본텍스트에서 단어 임베딩까지**
   - 원본 텍스트를 신경망이 처리할 수 있는 형태로 변환 (인코딩)
   - 케라스 모델에 embedding 층을 추가하여 어떤 작업에 특화된 토큰 임베딩을 학습
   - 데이터가 부족한 자연어 처리 문제에서 사전 훈련된 단어 임베딩을 사용하여 성능 향상을 꾀함.

6.2 순환 신경망 이해하기

- 앞에서 본 완전 연결 네트워크나 컨브넷과 같은 신경망들은 메모리가 존재하지 않았음. 그러므로 전체 시퀀스를 주입해주어야 했다.
- 하지만 이번장에서 볼 수 있는 순환 신경망(RNN)은 시퀀스의 원소를 순회하면서 지금까지 처리한 정보를 '상태'에 저장한다.
- 즉 네트워크에 하나의 입력을 주입하면 시퀀스의 원소를 차례대로 방문한다.
- 각 타임스텝 t에서 현재상태와 입력을 연결하여 출력을 계산한다.
- 이 출력을 다음 스텝의 상태로 설정.

Y = activation(dot(state_t,U)+dot(input_t,W)+b)

순환 네트워크 : 루프를 가진 네트워크 ▶



6.2 순환 신경망 이해하기

```
import numpy as np

timesteps = 100
input_features = 32
output_features  = 64

inputs = np.random.random((timesteps, input_features))

state_t = np.zeros((output_features,))

W = np.random.random((output_features, input_features))
U = np.random.random((output_features, input_features))
b = np.random.random((output_features, ))

successive_outputs = []
for input_t in inputs:
  output_t = np.tanh(np.dot(W, input_t) + np.dot(U, state_t) + b)

  successive_outputs.append(output_t)

  state_t = output_t

final_output_sequence = np.stack(successive_outputs, axis=0)
```

간단한 RNN의 정방향 계산을 넘파이로 구현한 것.

입력과 현재 상태 연결

네트워크 상태 업데이트

6.2 순환 신경망 이해하기

1. 케라스의 순환층

앞서 간단하게 구현한 과정이 케라스의 SimpleRNN 층에 해당.

한가지 다른점은 하나의 시퀀스가 아닌 시퀀스 배치를 처리한다.

즉 (batch_size, timesteps, input_features) 의 3차원 텐서를 입력을 받는다.

또한 두가지 모드로 실행가능하다. 1) 각 타임스텝의 출력을 모은 전체 시퀀스를 반환(3차원 텐서) 하거나 2)입력시퀀스에 대한 마지막 출력만(2차원 텐서) 반환.
디폴트로는 마지막 출력만 반환해주고 매개변수에 return_sequences = True 를 넣어주면 전체 상태 시퀀스를 반환한다.

네트워크의 표현력 증가를 위해 여러개의 순환층을 쌓을때는 중간층들이 전체 출력 시퀀스를 반환하도록 설정해야한다.

```
model.add(SimpleRNN(32, return_squences=True))
```

6.2 순환 신경망 이해하기

- IMDB 영화 리뷰 분류 문제에 적용해보기.

```
from keras.layers import Dense,Embedding, SimpleRNN
from keras.models import *

model = Sequential()
model.add(Embedding(max_features, 32))
model.add(SimpleRNN(32))
model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])
history=model.fit(input_train, y_train,
                  epochs=10,
                  batch_size=128,
                  validation_split=0.2)
```

IMDB 데이터 셋 로드

↓

순환 네트워크 생성

↓

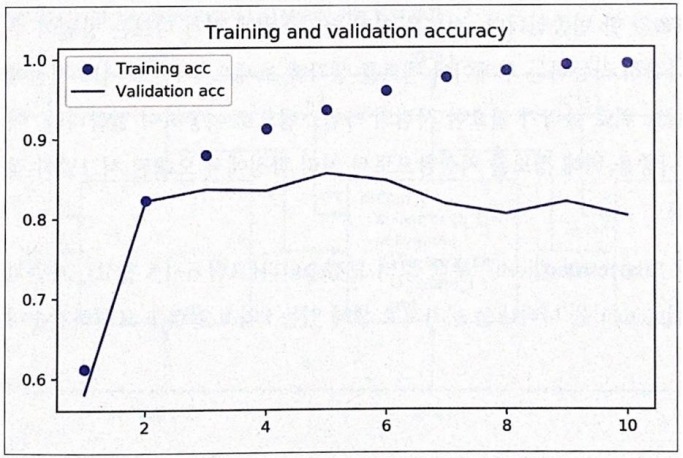
순환 네트워크 훈련

↓

훈련&검증 손실, 정확도 확인

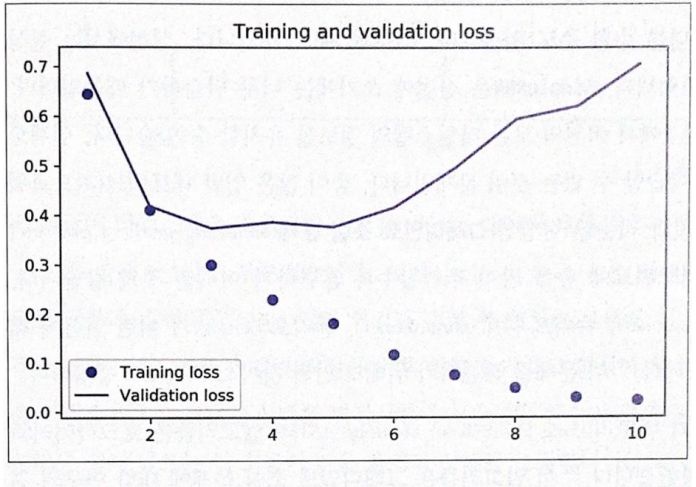
6.2 순환 신경망 이해하기

그림 6-12 SimpleRNN을 사용한 IMDB 문제의 훈련 정확도와 검증 정확도



정확도 비교

그림 6-11 SimpleRNN을 사용한 IMDB 문제의 훈련 손실과 검증 손실



손실 비교

6.2 순환 신경망 이해하기

2. LSTM과 GRU 층 이해하기

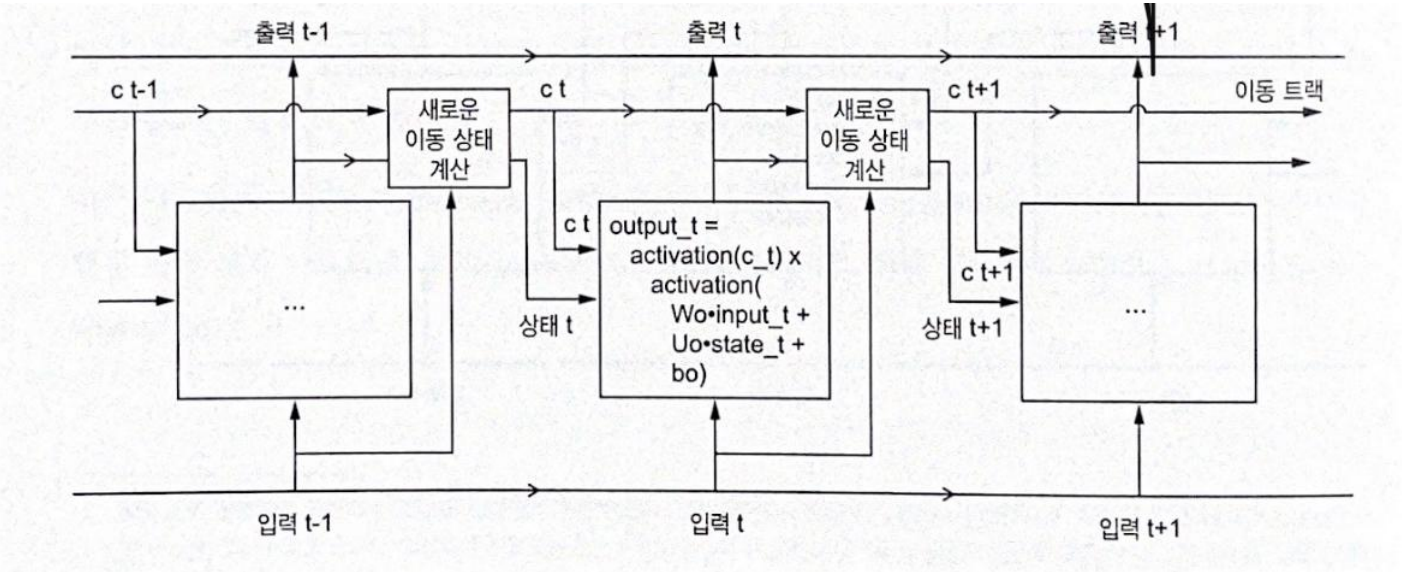
SimpleRNN은 너무 단순하기때문에 실전에선 다른 순환층인 LSTM과 GRU를 사용한다.

SimpleRNN은 실제론 긴 시간에 걸친 의존성은 학습할 수 없는 것이 문제다. 이 때문에 **그래디언트 소실문제**가 발생한다.

이를 해결하기 위해 장-단기 알고리즘이 개발되었다.

LSTM은 SimpleRNN에 정보를 여러 타임스텝에 걸쳐 나르는 방법이 추가 된다.

6.2 순환 신경망 이해하기



LSTM 구조

6.2 순환 신경망 이해하기

```
output_t = activation(c_t)*activation(dot(input_t, Wo)+dot(state_t, Uo)+bo)

i_t = activation(dot(state_t, Ui)+dot(input_t, Wi)+bi)
f_t = activation(dot(state_t, Uf)+dot(input_t, Wf)+bf)
k_t = activation(dot(state_t, Uk)+dot(input_t, Wk)+bk)

c_t+1 = i_t*k_t + c_t*f_t
```

새로운 이동상태 계산

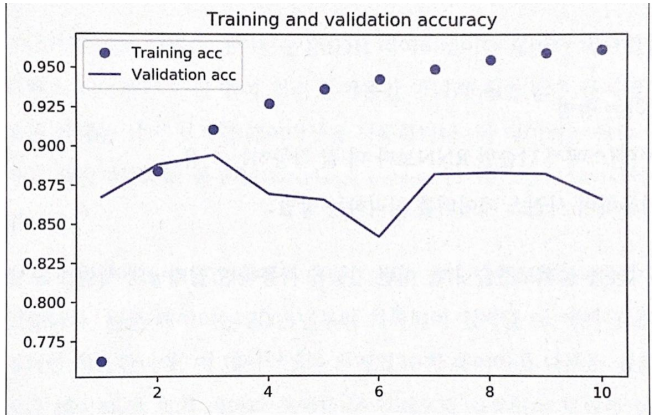
- 3개의 다른 변환이 연관
- 자신만의 가중치 행렬을 가짐

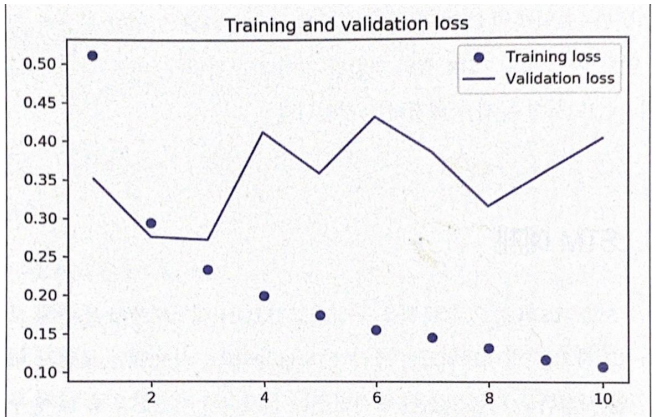
6.2 순환 신경망 이해하기

3. 케라스를 사용한 LSTM 예제

```
from keras.layers import Dense,Embedding, LSTM
from keras.models import *

model = Sequential()
model.add(Embedding(max_features, 32))
model.add(LSTM(32))
model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])
history=model.fit(input_train, y_train,
                  epochs=10,
                  batch_size=128,
                  validation_split=0.2)
```

LSTM을 이용한 IMDB 데이터 예측





6.2 순환 신경망 이해하기

- Simple RNN보다는 LSTM이 더 나은 성능을 보여줌.
- 3장에서 사용한 완전연결 네트워크보다 적은 데이터를 사용하고 더 나은 성능
- 하지만 많은 계산을 한것 치고 크게 좋은 결과는 아님.
- 이는 튜닝이 이루어지지 않았고 규제가 없기 때문.
- 사실은 LSTM이 잘하는 분야가 아님.
- 이러한 문제엔 완전연결네트워크가 더 나음.
- LSTM은 질문-응답, 기계 번역과 같은 복잡한 자연어 처리 문제에 좋은 성능을 보여줌

6.3 순환 신경망의 고급 사용법

순환 드롭아웃(recurrent dropout)

스태킹 순환 층(stacking recurrent layer)

양방향 순환 층(bidirectional recurrent layer)

6.3 순환 신경망의 고급 사용법

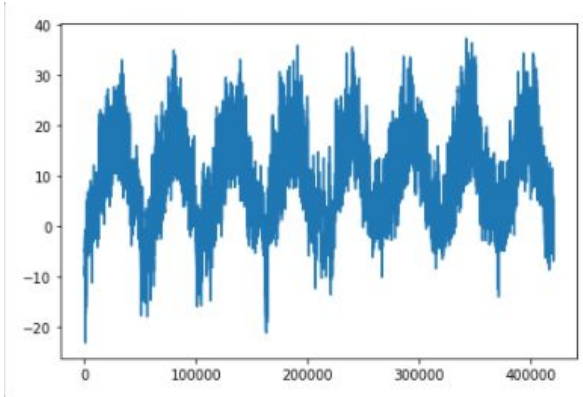
```
import os

data_dir = './datasets/jena_climate/'
fname = os.path.join(data_dir, 'jena_climate_2009_2016.csv')

f = open(fname)
data = f.read()
f.close()

lines = data.split('#n')
header = lines[0].split(',')
lines = lines[1:]

print(header)
print(len(lines))
```

```
import numpy as np

float_data = np.zeros((len(lines), len(header) - 1))
for i, line in enumerate(lines):
    values = [float(x) for x in line.split(',')[1:]]
    float_data[i, :] = values
```

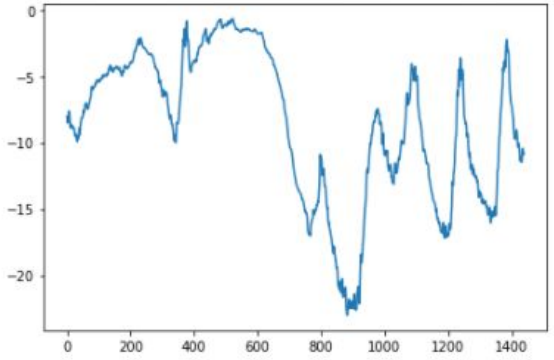
6.3 순환 신경망의 고급 사용법

1. 기온 예측 문제

```
1 temp = float_data[:, 1]  # 온도(섭씨)
2 plt.plot(range(len(temp)), temp)
3 plt.show()
```




```
1 plt.plot(range(1440), temp[:1440])
2 plt.show()
```



6.3 순환 신경망의 고급 사용법

```
mean = float_data[:200000].mean(axis=0)
float_data -= mean
std = float_data[:200000].std(axis=0)
float_data /= std
```

- data
- lookback
- delay
- min_index, max_index
- shuffle
- batch_size
- step

```
def generator(data, lookback, delay, min_index, max_index,
              shuffle=False, batch_size=128, step=6):
    if max_index is None:
        max_index = len(data) - delay - 1
    i = min_index + lookback
    while 1:
        if shuffle:
            rows = np.random.randint(
                min_index + lookback, max_index, size=batch_size)
        else:
            if i + batch_size >= max_index:
                i = min_index + lookback
            rows = np.arange(i, min(i + batch_size, max_index))
            i += len(rows)

        samples = np.zeros((len(rows),
                            lookback // step,
                            data.shape[-1]))
        targets = np.zeros((len(rows),))
        for j, row in enumerate(rows):
            indices = range(rows[j] - lookback, rows[j], step)
            samples[j] = data[indices]
            targets[j] = data[rows[j] + delay][1]
        yield samples, targets
```

6.3 순환 신경망의 고급 사용법

- train_gen
- val_gen
- test_gen

```
lookback = 1440
step = 6
delay = 144
batch_size = 128

train_gen = generator(float_data,
                      lookback=lookback,
                      delay=delay,
                      min_index=0,
                      max_index=200000,
                      shuffle=True,
                      step=step,
                      batch_size=batch_size)
val_gen = generator(float_data,
                    lookback=lookback,
                    delay=delay,
                    min_index=200001,
                    max_index=300000,
                    step=step,
                    batch_size=batch_size)
test_gen = generator(float_data,
                     lookback=lookback,
                     delay=delay,
                     min_index=300001,
                     max_index=None,
                     step=step,
                     batch_size=batch_size)

# 전체 검증 세트를 순회하기 위해 val_gen에서 추출할 횟수
val_steps = (300000 - 200001 - lookback) // batch_size

# 전체 테스트 세트를 순회하기 위해 test_gen에서 추출할 횟수
test_steps = (len(float_data) - 300001 - lookback) // batch_size
```

6.3 순환 신경망의 고급 사용법

평균 절댓값 오차 평가

```
def evaluate_naive_method():
    batch_maes = []
    for step in range(val_steps):
        samples, targets = next(val_gen)
        preds = samples[:, -1, 1]
        mae = np.mean(np.abs(preds - targets))
        batch_maes.append(mae)
    print(np.mean(batch_maes))

evaluate_naive_method()
```

```
0.2897359729905486
```

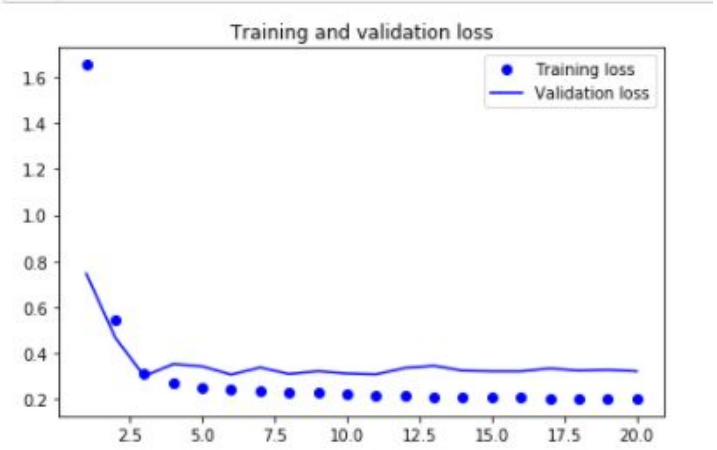
```
0.29 * std[1]
```

```
2.5672247338393395
```

6.3 순환 신경망의 고급 사용법

기본적인 머신 러닝 모델 구현

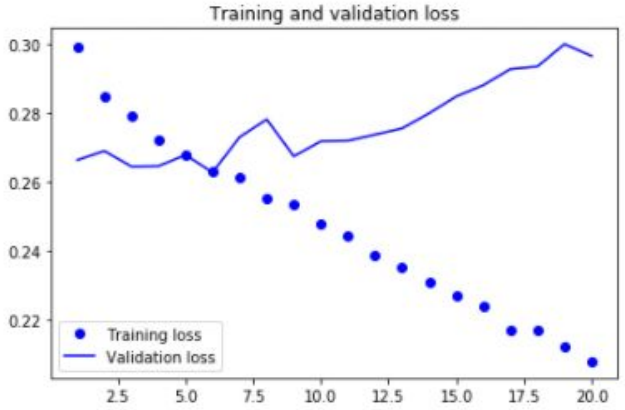
```
from keras.models import Sequential
from keras import layers
from keras.optimizers import RMSprop

model = Sequential()
model.add(layers.Flatten(input_shape=(lookback // step, float_data.shape[-1])))
model.add(layers.Dense(32, activation='relu'))
model.add(layers.Dense(1))

model.compile(optimizer=RMSprop(), loss='mae')
history = model.fit_generator(train_gen,
                              steps_per_epoch=500,
                              epochs=20,
                              validation_data=val_gen,
                              validation_steps=val_steps)
```



6.3 순환 신경망의 고급 사용법

순환 신경망 모델

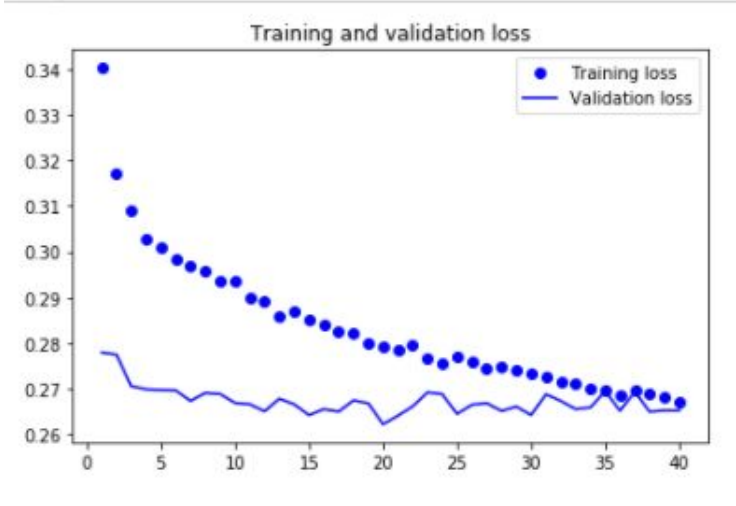
```
from keras.models import Sequential
from keras import layers
from keras.optimizers import RMSprop

model = Sequential()
model.add(layers.GRU(32, input_shape=(None, float_data.shape[-1])))
model.add(layers.Dense(1))

model.compile(optimizer=RMSprop(), loss='mae')
history = model.fit_generator(train_gen,
                              steps_per_epoch=500,
                              epochs=20,
                              validation_data=val_gen,
                              validation_steps=val_steps)
```



6.3 순환 신경망의 고급 사용법

순환 드롭아웃

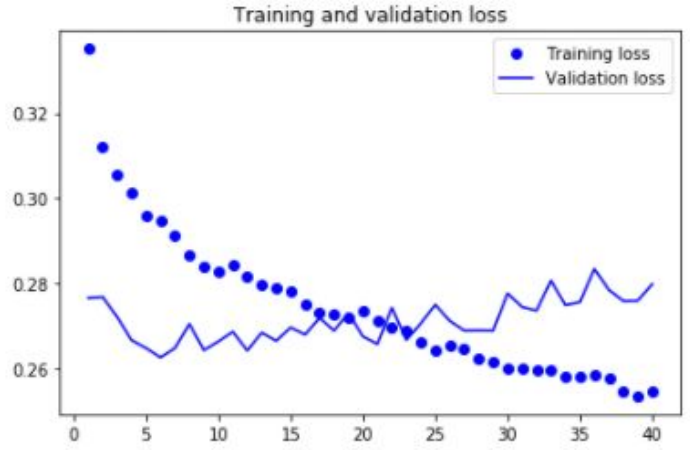
```
from keras.models import Sequential
from keras import layers
from keras.optimizers import RMSprop

model = Sequential()
model.add(layers.GRU(32,
                     dropout=0.2,
                     recurrent_dropout=0.2,
                     input_shape=(None, float_data.shape[-1])))
model.add(layers.Dense(1))

model.compile(optimizer=RMSprop(), loss='mae')
history = model.fit_generator(train_gen,
                              steps_per_epoch=500,
                              epochs=40,
                              validation_data=val_gen,
                              validation_steps=val_steps)
```



6.3 순환 신경망의 고급 사용법

스태킹 순환 층

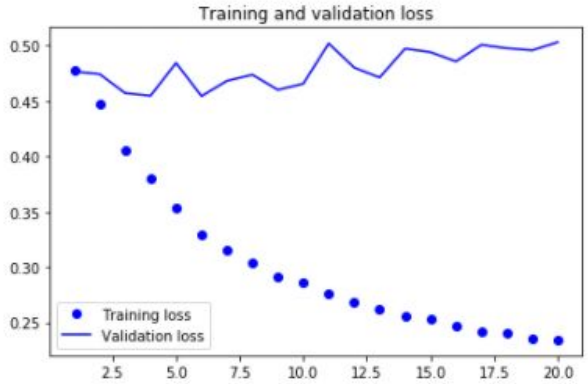
```
from keras.models import Sequential
from keras import layers
from keras.optimizers import RMSprop

model = Sequential()
model.add(layers.GRU(32,
                     dropout=0.1,
                     recurrent_dropout=0.5,
                     return_sequences=True,
                     input_shape=(None, float_data.shape[-1])))
model.add(layers.GRU(64, activation='relu',
                     dropout=0.1,
                     recurrent_dropout=0.5))
model.add(layers.Dense(1))

model.compile(optimizer=RMSprop(), loss='mae')
history = model.fit_generator(train_gen,
                              steps_per_epoch=500,
                              epochs=40,
                              validation_data=val_gen,
                              validation_steps=val_steps)
```



6.3 순환 신경망의 고급 사용법

```python
def reverse_order_generator(data, lookback, delay, min_index, max_index,
                            shuffle=False, batch_size=128, step=6):
    if max_index is None:
        max_index = len(data) - delay - 1
    i = min_index + lookback
    while 1:
        if shuffle:
            rows = np.random.randint(
                min_index + lookback, max_index, size=batch_size)
        else:
            if i + batch_size >= max_index:
                i = min_index + lookback
            rows = np.arange(i, min(i + batch_size, max_index))
            i += len(rows)

        samples = np.zeros((len(rows),
                            lookback // step,
                            data.shape[-1]))
        targets = np.zeros((len(rows),))
        for j, row in enumerate(rows):
            indices = range(rows[j] - lookback, rows[j], step)
            samples[j] = data[indices]
            targets[j] = data[rows[j] + delay][1]
        yield samples[:, ::-1, :], targets

train_gen_reverse = reverse_order_generator(
    float_data,
    lookback=lookback,
    delay=delay,
    min_index=0,
    max_index=200000,
    shuffle=True,
    step=step,
    batch_size=batch_size)
val_gen_reverse = reverse_order_generator(
    float_data,
    lookback=lookback,
    delay=delay,
    min_index=200001,
    max_index=300000,
    step=step,
    batch_size=batch_size)
```

```python
model = Sequential()
model.add(layers.GRU(32, input_shape=(None, float_data.shape[-1])))
model.add(layers.Dense(1))

model.compile(optimizer=RMSprop(), loss='mae')
history = model.fit_generator(train_gen_reverse,
                              steps_per_epoch=500,
                              epochs=20,
                              validation_data=val_gen_reverse,
                              validation_steps=val_steps)
```

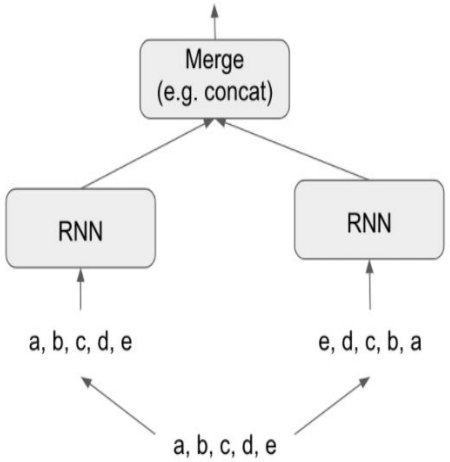


6.3 순환 신경망의 고급 사용법

```python
from keras.datasets import imdb
from keras.preprocessing import sequence
from keras import layers
from keras.models import Sequential

# 특성으로 사용할 단어의 수
max_features = 10000
# 사용할 텍스트의 길이(가장 빈번한 max_features 개의 단어만 사용합니다)
maxlen = 500

# 데이터 로드
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_features)

# 시퀀스를 뒤집습니다
x_train = [x[::-1] for x in x_train]
x_test = [x[::-1] for x in x_test]

# 시퀀스에 패딩을 추가합니다
x_train = sequence.pad_sequences(x_train, maxlen=maxlen)
x_test = sequence.pad_sequences(x_test, maxlen=maxlen)

model = Sequential()
model.add(layers.Embedding(max_features, 128))
model.add(layers.LSTM(32))
model.add(layers.Dense(1, activation='sigmoid'))

model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['acc'])
history = model.fit(x_train, y_train,
                    epochs=10,
                    batch_size=128,
                    validation_split=0.2)
```

```
Epoch 7/10
20000/20000 [==============================] - 72s 4ms/step - loss: 0.1534 - acc: 0.9475 - val_loss: 0.4477 - val_acc: 0.8694
Epoch 8/10
20000/20000 [==============================] - 72s 4ms/step - loss: 0.1457 - acc: 0.9517 - val_loss: 0.6034 - val_acc: 0.8502
Epoch 9/10
20000/20000 [==============================] - 72s 4ms/step - loss: 0.1322 - acc: 0.9569 - val_loss: 0.4100 - val_acc: 0.8664
Epoch 10/10
20000/20000 [==============================] - 72s 4ms/step - loss: 0.1213 - acc: 0.9593 - val_loss: 0.4392 - val_acc: 0.8764
```

6.3 순환 신경망의 고급 사용법




```
from keras import backend as K
K.clear_session()
```

```
model = Sequential()
model.add(layers.Embedding(max_features, 32))
model.add(layers.Bidirectional(layers.LSTM(32)))
model.add(layers.Dense(1, activation='sigmoid'))

model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])
history = model.fit(x_train, y_train, epochs=10, batch_size=128, validation_split=0.2)
```

```
Epoch 7/10
20000/20000 [==============================] - 134s 7ms/step - loss: 0.1689 - acc: 0.9427 - val_loss: 0.3866 - val_acc: 0.8578
Epoch 8/10
20000/20000 [==============================] - 134s 7ms/step - loss: 0.1531 - acc: 0.9471 - val_loss: 0.3366 - val_acc: 0.8754
Epoch 9/10
20000/20000 [==============================] - 134s 7ms/step - loss: 0.1416 - acc: 0.9522 - val_loss: 0.3549 - val_acc: 0.8642
Epoch 10/10
20000/20000 [==============================] - 134s 7ms/step - loss: 0.1336 - acc: 0.9556 - val_loss: 0.3664 - val_acc: 0.8770
```

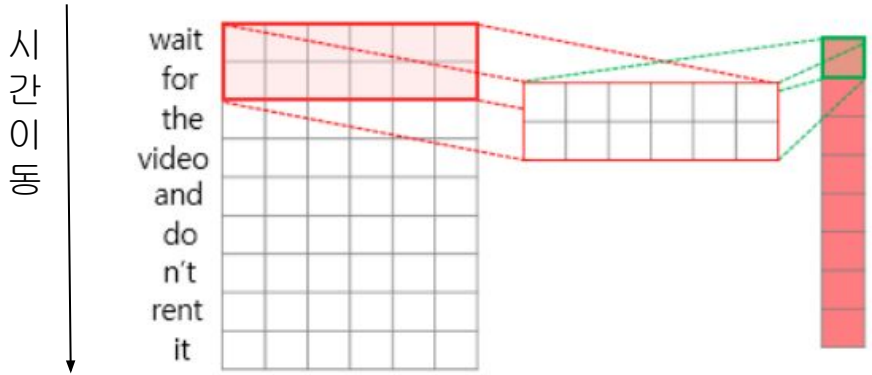
6.3 순환 신경망의 고급 사용법

```
from keras.models import Sequential
from keras import layers
from keras.optimizers import RMSprop

model = Sequential()
model.add(layers.Bidirectional(
    layers.GRU(32), input_shape=(None, float_data.shape[-1])))
model.add(layers.Dense(1))

model.compile(optimizer=RMSprop(), loss='mae')
history = model.fit_generator(train_gen,
                              steps_per_epoch=500,
                              epochs=40,
                              validation_data=val_gen,
                              validation_steps=val_steps)
```

```
Epoch 32/40
500/500 [==============================] - 195s 390ms/step - loss: 0.1482 - val_loss: 0.3286
Epoch 33/40
500/500 [==============================] - 195s 390ms/step - loss: 0.1451 - val_loss: 0.3310
Epoch 34/40
500/500 [==============================] - 195s 390ms/step - loss: 0.1445 - val_loss: 0.3332
Epoch 35/40
500/500 [==============================] - 195s 390ms/step - loss: 0.1429 - val_loss: 0.3343
Epoch 36/40
500/500 [==============================] - 195s 390ms/step - loss: 0.1422 - val_loss: 0.3372
Epoch 37/40
500/500 [==============================] - 195s 389ms/step - loss: 0.1407 - val_loss: 0.3314
Epoch 38/40
500/500 [==============================] - 195s 390ms/step - loss: 0.1394 - val_loss: 0.3357
Epoch 39/40
500/500 [==============================] - 195s 390ms/step - loss: 0.1371 - val_loss: 0.3326
Epoch 40/40
500/500 [==============================] - 195s 390ms/step - loss: 0.1363 - val_loss: 0.3369
```

6.4 컨브넷을 사용한 시퀀스 처리

1. 시퀀스 데이터를 위한 1D 합성곱 이해하기
   부분 시퀀스를 추출하여 나머지에 동일한 합성곱을 적용
   특정 위치에서 학습한 패턴을 다른 위치에서 인식 가능
   ->1D 컨브넷에 (시간 이동에 대한)이동 불변성 제공



관련 논문 : Fully Character-Level Neural Machine Translation without Explicit Segmentation
https://direct.mit.edu/tacl/article/doi/10.1162/tacl_a_00067/43402/Fully-Character-Level-Neural-Machine-Translation

6.4 컨브넷을 사용한 시퀀스 처리

2. 시퀀스 데이터를 위한 1D 풀링
   2D 풀링과 동일 방법
   1D 패치를 추출-> 최댓값(최대풀링) / 평균값(평균풀링) 출력
3. 1D 컨브넷 구현

```
1  from keras.datasets import imdb
2  from keras.preprocessing import sequence
3  max_features = 10000
4  max_len = 500
5
6  print('데이터 로드 ...')
7  (x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_features)
8  print(len(x_train), '훈련 시퀀스')
9  print(len(x_test), '테스트 시퀀스')
10
11 print('시퀀스 패딩 (samples x time)')
12 x_train = sequence.pad_sequences(x_train, maxlen=max_len)
13 x_test = sequence.pad_sequences(x_test, maxlen=max_len)
14 print('x_train 크기: ', x_train.shape)
15 print('x_test  크기: ', x_test.shape)
```

```
데이터 로드 ...
<string>:6: VisibleDeprecationWarning:
/usr/local/lib/python3.7/dist-packages/
  x_train, y_train = np.array(xs[:idx])
/usr/local/lib/python3.7/dist-packages/
  x_test, y_test = np.array(xs[idx:]),
25000 훈련 시퀀스
25000 테스트 시퀀스
시퀀스 패딩 (samples x time)
x_train 크기:  (25000, 500)
x_test  크기:  (25000, 500)
```

6.4 컨브넷을 사용한 시퀀스 처리

3. 1D 컨브넷 구현

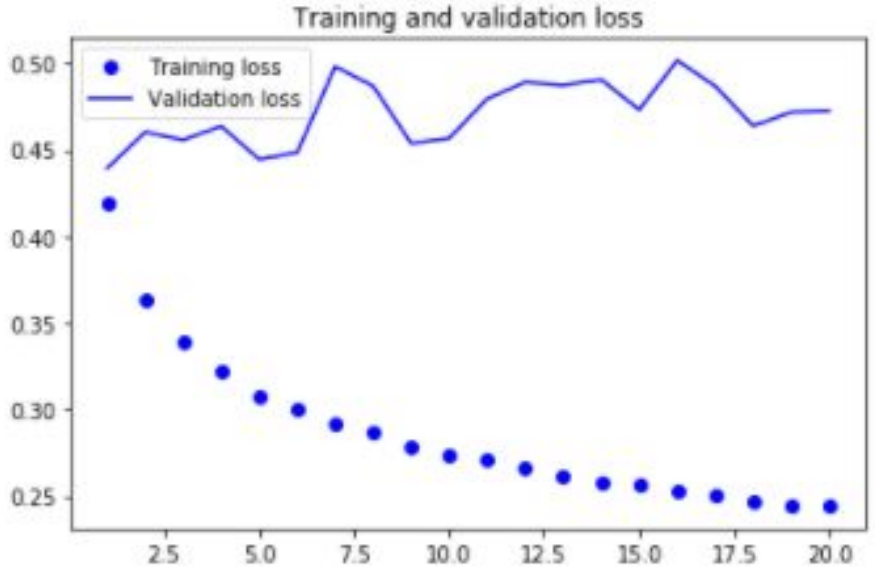
```
from keras.models import Sequential
from keras import  layers
from keras.optimizers import RMSprop

model = Sequential()
model.add(layers.Embedding(max_features, 128, input_length=max_len))
model.add(layers.Conv1D(32, 7, activation='relu'))
model.add(layers.MaxPooling1D(5))
model.add(layers.Conv1D(32, 7, activation='relu'))
model.add(layers.GlobalMaxPooling1D())
model.add(layers.Dense(1))

model.summary()
model.compile(optimizer = RMSprop(lr=1e-4),
              loss='binary_crossentropy',
              metrics=['acc'])
history=model.fit(x_train, y_train,
                  epochs=10,
                  batch_size=128,
                  validation_split=0.2)
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 500, 128)          1280000
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 494, 32)           28704
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 98, 32)            0
_________________________________________________________________
conv1d_3 (Conv1D)            (None, 92, 32)            7200
_________________________________________________________________
global_max_pooling1d_1 (Glob (None, 32)                0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 33
=================================================================
Total params: 1,315,937
Trainable params: 1,315,937
Non-trainable params: 0
```

6.4 컨브넷을 사용한 시퀀스 처리

4. CNN과 RNN을 연결하여 긴 시퀀스를 처리하기

```
from keras,models import Sequential
from keras import layers
from keras,optimizers import RMSprop

model = Sequential()
model,add(layers,Conv1D(32, 5, activation='relu',
                        input_shape=(None, float_data,shape[-1])))
model,add(layers,MaxPooling1D(3))
model,add(layers,Conv1D(32, 5, activation='relu'))
model,add(layers,MaxPooling1D(3))
model,add(layers,Conv1D(32, 5, activation='relu'))
model,add(layers,GlobalMaxPooling1D())
model,add(layers,Dense(1))

model,compile(optimizer=RMSprop(), loss='mae')
history = model,fit_generator(train_gen,
                              steps_per_epoch=500,
                              epochs=20,
                              validation_data=val_gen,
                              validation_steps=val_steps)
```



6.4 컨브넷을 사용한 시퀀스 처리

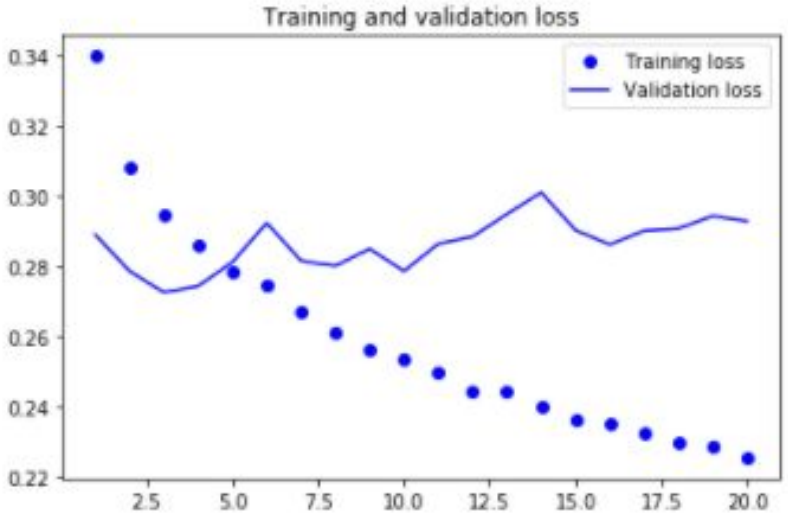
```
from keras.models import Sequential
from keras import layer
from keras.optimizers import RMSprop
model = Sequential()
model.add(layers.Conv1D(32, 5, activation='relu',
                        input_shape=(None, float_data.shape[-1])))
model.add(layers.MaxPooling1D(3))
model.add(layers.Conv1D(32, 5, activation='relu'))
model.add(layers.GRU(32, dropout=0.1, recurrent_dropout=0.5))
model.add(layers.Dense(1))

model.summary()

model.compile(optimizer=RMSprop(), loss='mae')
history = model.fit_generator(train_gen,
                              steps_per_epoch=500,
                              epochs=20,
                              validation_data=val_gen,
                              validation_steps=val_steps)
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_6 (Conv1D)            (None, None, 32)          2272
_________________________________________________________________
max_pooling1d_4 (MaxPooling1 (None, None, 32)          0
_________________________________________________________________
conv1d_7 (Conv1D)            (None, None, 32)          5152
_________________________________________________________________
gru_1 (GRU)                  (None, 32)                6240
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 33
=================================================================
Total params: 13,697
Trainable params: 13,697
Non-trainable params: 0
```

◀ 두개의 Conv1D 층 다음 GRU 층

6.4 컨브넷을 사용한 시퀀스 처리

훈련 손실 & 검증 손실 비교



- 규제가 있는 GRU 모델만큼 좋지는 않음.
- 하지만 훨씬 빨라서 데이터를 2배 더 많이 처리할 수 있음.