6장. 순환 신경망

Wrap Up

1. 컨볼루션 신경망은 이미지 데이터에 특화되어 있지만, 음성 인식이나 비디오, 6장에서 다뤄볼 텍스트 데이터에도 사용됩니다.
2. **완전연결층은 공간 정보를 손실**하는 반면, **컨볼루션층은 공간정보를 유지**합니다.
3. 컨볼루션층은 컨볼루션 필터를 통해 이미지의 특징을 인식할 수 있게 됩니다. 또한, **컨볼루션 필터가 가지는 파라미터**는 이미지 필터와 다르게 직접 정의해주지 않고, **학습을 통해 조정**됩니다.
4. 컨볼루션층에서는 주요한 인자로서 **컨볼루션 필터 개수, 스트라이드 크기, 패딩 여부**를 사용하고, 풀링층에서는 주요한 인자로서 **커널 크기, 스트라이드 크기**를 사용합니다.
5. 컨볼루션층은 1x1 크기를 사용하여 최대한 공간정보를 손실하지 않도록 하며, 다운샘플링이 필요할 경우 최대 풀링층을 사용합니다.
6. model.summary(), plot_model() 함수는 모델 구조를 확인하기에 유용합니다.
7. **규제화 함수, 드롭아웃, 배치 정규화**는 과대적합을 방지할 뿐, 100% 해결해주지 않습니다.

Wrap Up

8. **데이터 증식 방법**은 다양한 데이터를 모델에 입력해 **더욱 견고한 모델**을 얻을 수 있도록 도와줍니다. 케라스에는 이를 위한 이미지 제네레이터가 준비되어 있습니다.
9. **전이 학습**은 **사전 학습된 가중치를 사용**하여 더욱 빠르게 향상된 성능을 얻을 수 있도록 도와줍니다. 케라스는 수많은 이미지 데이터로 구성된 ImageNet 데이터를 학습한 다양한 모델을 제공하고 있습니다.
10. **모델 상위층은 데이터의 구체적 특징**을 학습하며, **모델 하위층은 단순한 특징을 학습**합니다.
11. **텐서플로우 허브**는 우리가 원하는 모델을 쉽게 찾을 수 있도록 도와줍니다.

6장의 내용?

- 컨볼루션 신경망과 함께 양대산맥을 이루는 순환 신경망(RNN; Recurrent Neural Network)
  - 음성 인식, 기계 번역(네이버 파파고, 구글 번역기)
  - 주식, 온도, 매출과 같이 시간이 지남에 따라 변화하는 데이터를 활용하여 주식 종가, 날씨, 상점 매출 예측
  - 자연어 처리(NLP; Natural Language Preprocessing)

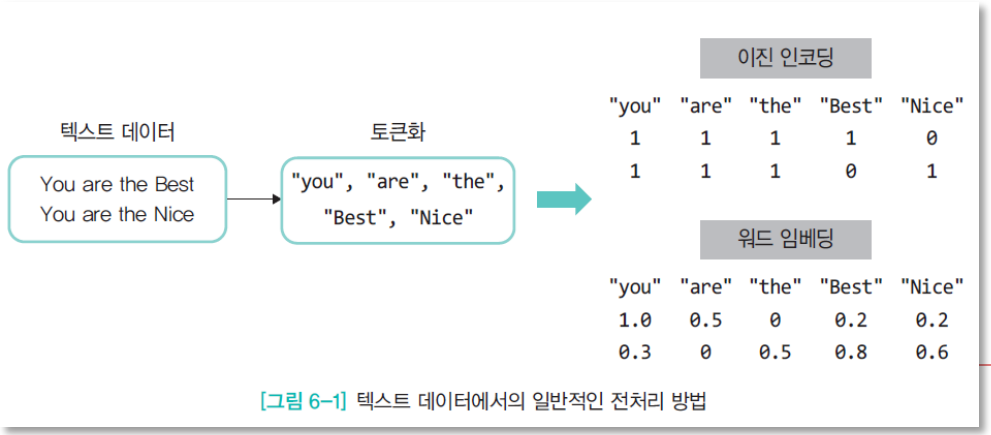
- 이번 장에서는 다음 내용을 다뤄봅니다.
  - Embedding, SimpleRNN, LSTM, Conv1D
  - IMDB, reuters, 주기를 포함한 시계열 데이터셋
  - BERT 간단히 알아보기

Embedding – 원리 이해하기

- Embedding층은 **수많은 단어(또는 데이터)를 벡터 형태로 표현**할 수 있어 텍스트 분류를 위해 사용하는 가장 기본에 해당하는 층
  - 사전 학습된 가중치를 불러와 사용할 수 있음(실습을 참고)
  - 토큰, 토큰화, 텍스트 분류에 사용되는 기본적 용어를 알고 사용해보자

- **토큰(Token)**
  - 문법적으로 더 이상 나눌 수 없는 언어 요소
  - 이를 수행하는 작업을 **토큰화(Tokenizer)**라고 함

- 텍스트 데이터를 신경망에 입력하기 위해서 일반적으로 토큰화 작업을 수행하고 정의된 토큰에 고유 인덱스를 부여한 뒤, 인코딩을 통해 적절한 형태로 바꿔주는 전처리 작업 과정을 거치게 됨
  - 원-핫 인코딩, 이진 인코딩
  - 워드 임베딩(Word Embedding)



[그림 6-1] 텍스트 데이터에서의 일반적인 전처리 방법

Embedding – 원리 이해하기

- 토큰화 작업을 수행해보자
  - tensorflow.keras.preprocessing.text 모듈에서 이를 위한 함수를 제공

- 데이터 준비

```
04  texts = ['You are the Best',
05           'You are the Nice']
```

- Tokenizer() 함수를 사용하여 토큰화 작업 수행
  - oov_token: 데이터에 나타나지 않은 단어를 전달된 단어로 교체
  - fit_on_texts() 함수를 통해 데이터에 적용하고, texts_to_sequences() 함수로 변환

```
07  tokenizer = Tokenizer(num_words = 10, oov_token = '<OOV>')
08  tokenizer.fit_on_texts(texts)
09
10  # 텍스트 데이터를 정수 인덱스 형태로 변환합니다.
11  sequences = tokenizer.texts_to_sequences(texts)
```

```
{'<OOV>': 1, 'you': 2, 'are': 3, 'the': 4, 'best': 5, 'nice': 6}
```

Embedding – 원리 이해하기

- 새로운 데이터 "You are the One"
  - 'One' 단어는 새로 등장했기 때문에, '<OOV>'로 대체됨

```
test sequences: [[2, 3, 4, 1]]
```

  - 'You', 'are', 'the'는 각각 2, 3, 4로 변환

- sequences_to_matrix() 함수를 사용하면 이진 형태로 인코딩된 결과를 얻을 수 있음

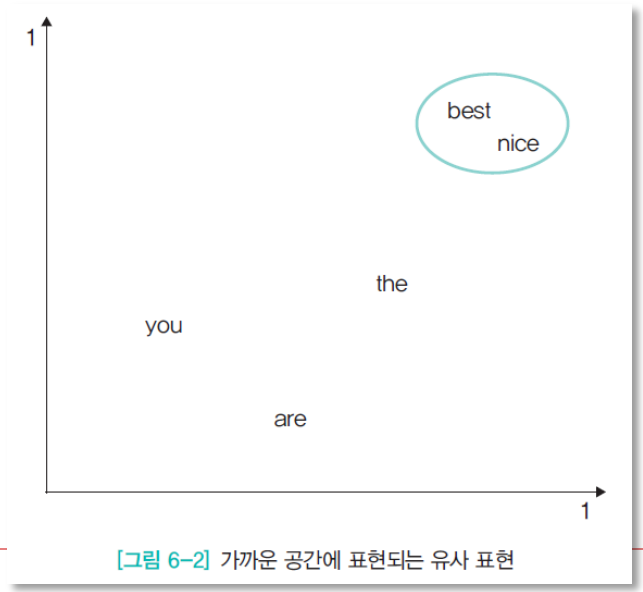
```
binary_vectors:
 [[0. 0. 1. 1. 1. 1. 0. 0. 0. 0.]
 [0. 0. 1. 1. 1. 0. 1. 0. 0. 0.]]
```

- sequences_to_matrix() 또는 to_categorical() 함수에서 얻을 수 있는 결과를 **희소 행렬(Sparse Matrix)**라고 표현하며, 이와 반대되는 개념을 **밀집 행렬(Dense Matrix)**라고 표현함

Embedding – 원리 이해하기

- 희소행렬
  - **존재하는 단어의 인덱스를 제외하고 전부 0으로 표현**
  - 고차원이며, 단어의 유사성(Similarity)를 표현할 수 없음
  - 행렬의 고차원으로 인해 불필요한 계산이 추가되며, 차원의 저주(Curse of Dimensionality)를 야기함

- 밀집행렬
  - **각 단어의 관계를 실수로 표현**하며, 저차원에 해당
  - 행렬에 속해있는 실숫값은 0과 1로 직접 지정해주는 희소행렬과 다르게 데이터를 기반으로 조정
  - 유사한 의미를 가지는 단어는 비슷한 공간에 표현(매핑)될 것



[그림 6-2] 가까운 공간에 표현되는 유사 표현

Embedding – 데이터 살펴보기

- Embedding층을 활용하여 IMDB 데이터셋 문제를 해결해보자
  - 위에서 배운 전처리 과정이 전부 수행된 채로 제공됨

- 데이터 다운로드

[함께 해봐요] **데이터셋 다운받기** use_embedding_layer.ipynb

```
01 from tensorflow.keras.datasets import imdb
02
03 num_words = 10000
04 (X_train, y_train), (X_test, y_test) = imdb.load_data(num_words=num_words)
```

  - num_words 인자를 통해 사용할 단어의 개수를 조절(여기서는 10,000개만 사용)
  - 학습 데이터와 테스트 데이터는 5:5 비율로 나뉘어서 제공

- 데이터 확인
  - 데이터에서 확인할 수 있는 숫자는 빈번하게 사용되는 정도를 나타냄
  - 레이블 → **1(긍정), 0(부정)**

```
[1, 14, 22, 16, 43, 530, 973, 1622, 1385, 65, 458, 4468, 66, 3941, 4, 173, 36,
256, 5, 25, 100, 43, 838, 112, 50, 670, 2, 9, 35, 480, 284, 5, 150, 4, 172,
... 생략 ...
-------
1
```

Embedding – 데이터 살펴보기

- 가장 빈번하게 사용되는 세 가지 단어 출력해보기
  - the, and, a
  - 포함? 제외? 고민해볼 수 있음
  - 포함시킨다면 어떤 방법으로 포함시킬 것인가 등

[함께 해봐요] IMDB 데이터셋에서 가장 빈번하게 사용되는 세 개의 단어

```
01  imdb_get_word_index = {}
02
03  for key, value in imdb.get_word_index().items():
04      imdb_get_word_index[value] = key
05
06  for i in range(1, 4):
07      print('{} 번째로 가장 많이 쓰인 단어 = {}'.format(i, imdb_get_word_index[i]))
```

```
1 번째로 가장 많이 쓰인 단어 = the
2 번째로 가장 많이 쓰인 단어 = and
3 번째로 가장 많이 쓰인 단어 = a
```

- **데이터의 길이가 전부 동일하도록 조정**해주기 위해 pad_sequences() 함수를 사용
  - 지정해준 길이보다 짧은 경우 0으로 채워넣음(zero padding), 긴 경우는 잘라냄

```
07  pad_X_train = pad_sequences(X_train, maxlen=max_len, padding = 'pre')
08  pad_X_test = pad_sequences(X_test, maxlen=max_len, padding = 'pre')
```

Embedding – 모델 구성하기

- Embedding층은 모델의 **첫 번째 층으로만 사용할 수 있으며, 주로 순환 신경망과 연결하여 사용**
  - **(batch_size, sequence_length)** 형태를 입력으로 받으며, **(batch_size, sequence_length, output_dim)** 형태를 출력

```
04  model = Sequential()
05  # 이 층은 모델의 제일 첫 번째 층으로만 사용할 수 있습니다.
06  # Flatten층을 사용하기 위해 input_length를 전달합니다.
07  model.add(Embedding(input_dim = num_words, output_dim = 32,
                        input_length = max_len))
```

  - input_dim(학습 데이터에서 사용한 단어의 개수), output_dim(임베딩 벡터 크기)
  - input_length 인자는 순환 신경망과 연결할 경우엔 사용하지 않음

Embedding – 모델 학습하고 평가하기

[함께 해봐요] **모델 학습시키기** use_embedding_layer.ipynb

```
01  history = model.fit(pad_X_train, y_train, batch_size = 32, epochs = 30,
                        validation_split = 0.2)
```

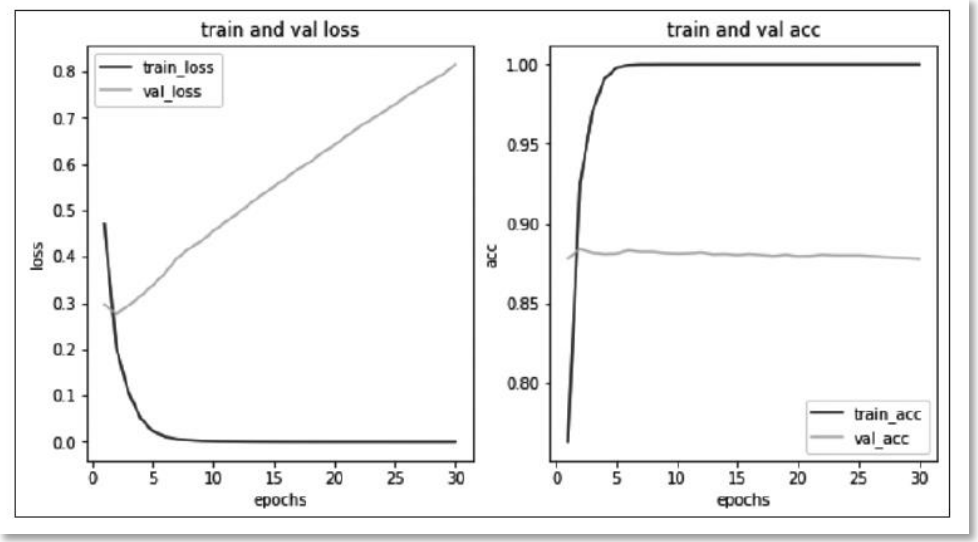
- validation_split 인자 사용
  - 학습 데이터의 끝에서 해당 비율만큼 떼어내어 검증 데이터셋으로 활용
  - 무작위로 20% 비율만큼 뽑아오는 것이 아닌 **단순하게 끝에서 떼어낸다**는 점을 주의

- 항상 결과를 확인하고, 학습 과정을 기록하는 것을 습관화할 것

[함께 해봐요] **모델 평가하기** use_embedding_layer.ipynb

```
01  model.evaluate(pad_X_test, y_test)
```

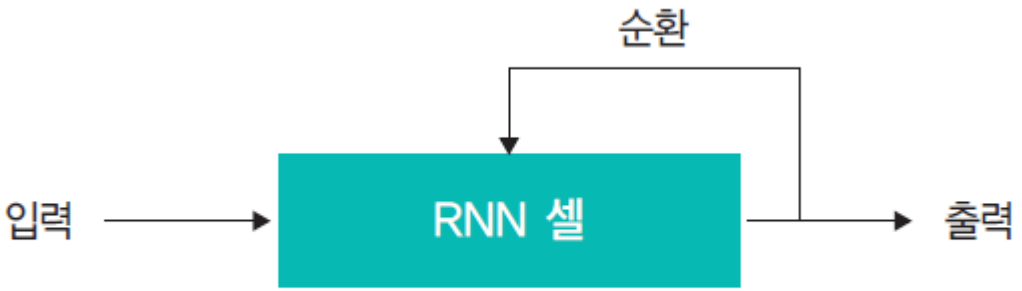
```
[0.8114458246806264, 0.87172]
```



RNN – 원리 이해하기

- Embedding층은 데이터의 표현을 학습하여 데이터 사전을 구축하는 것으로 쉽게 이해할 수 있음
  - 유사 단어를 비슷한 공간에 매핑하지만, **시퀀스 데이터의 중요한 특성인 순서와 맥락을 고려하지 않음**
  - "Dense vs Conv" 차이를 떠올려보자

- 순환 신경망은 완전연결층, 컨볼루션 신경망의 반대되는 개념으로 설명할 수 있음
  - 완전연결층과 컨볼루션 신경망은 **피드 포워드 네트워크(feed-forward network)**
  - 피드 포워드 네트워크는 출력값이 오직 마지막 층인 출력층을 향함
  - 하지만 순환 신경망은 출력값이 출력층을 향하면서도 동시에 **현재층의 다음 값으로 사용**

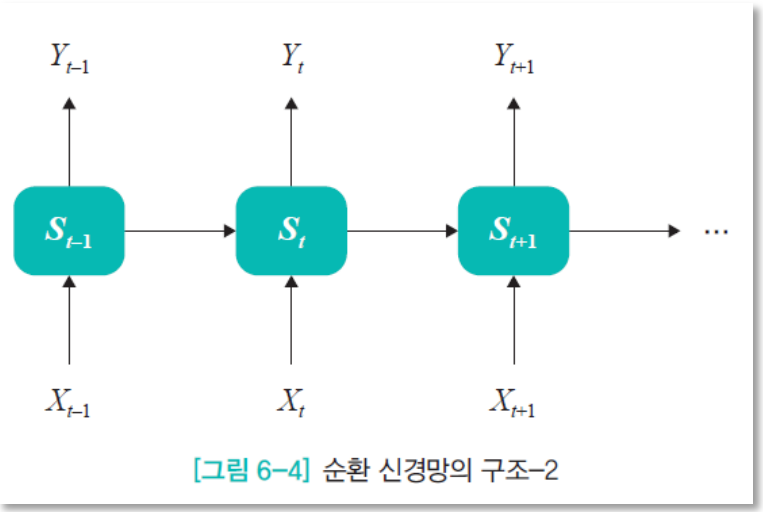


[그림 6-3] 순환 신경망의 구조-1

RNN – 원리 이해하기

- RNN 셀
  - 순환 신경망의 노드가 출력값을 반환하는 동시에 이전 상태(state)를 기억하는 메모리 역할을 수행
  - **은닉 상태(hidden state)**

- 다음 그림에서 x는 입력, y는 출력, t는 현재 시점을 의미
- 의사코드에서 output_t가 state_t의 값을 변환시키는 것을 확인
  - state_t는 activation_func(활성화 함수)에서 사용되고 있음



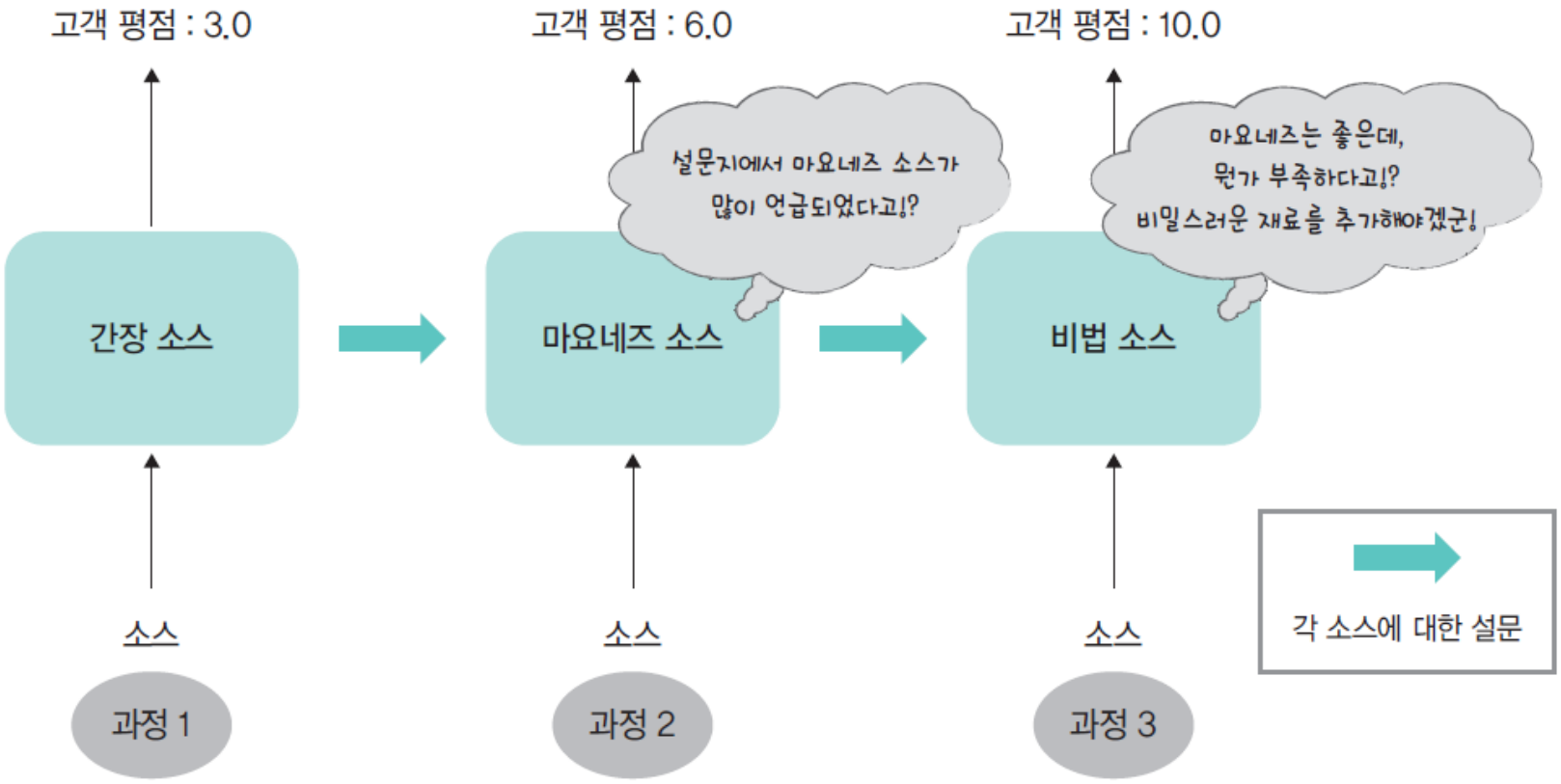
[그림 6-4] 순환 신경망의 구조-2

예시: 순환 신경망을 표현한 의사코드

```
state_t = 0  # 초기 상태

# 각 시점에 해당하는 입력을 반복합니다.
for input_t in input_sequence:
    # 입력과 은닉상태를 활성화 함수에 통과시킵니다.
    output_t = activation_func(input_t, state_t)
    # 출력값은 다음 시점을 위한 은닉 상태가 됩니다.
    state_t = output_t
```

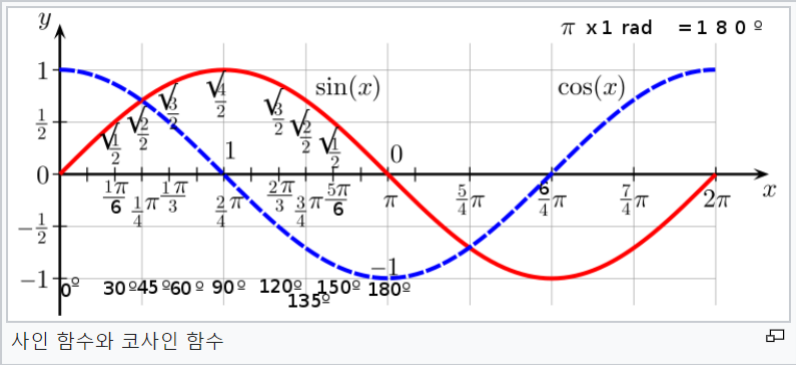
RNN – 원리 이해하기



[그림 6–5] 순환 신경망의 예

RNN – 데이터 살펴보기

- 주기를 가진 함수: sin, cos함수
  - 시계열 데이터 형태와 유사
  - 예제를 직접 만들어보자

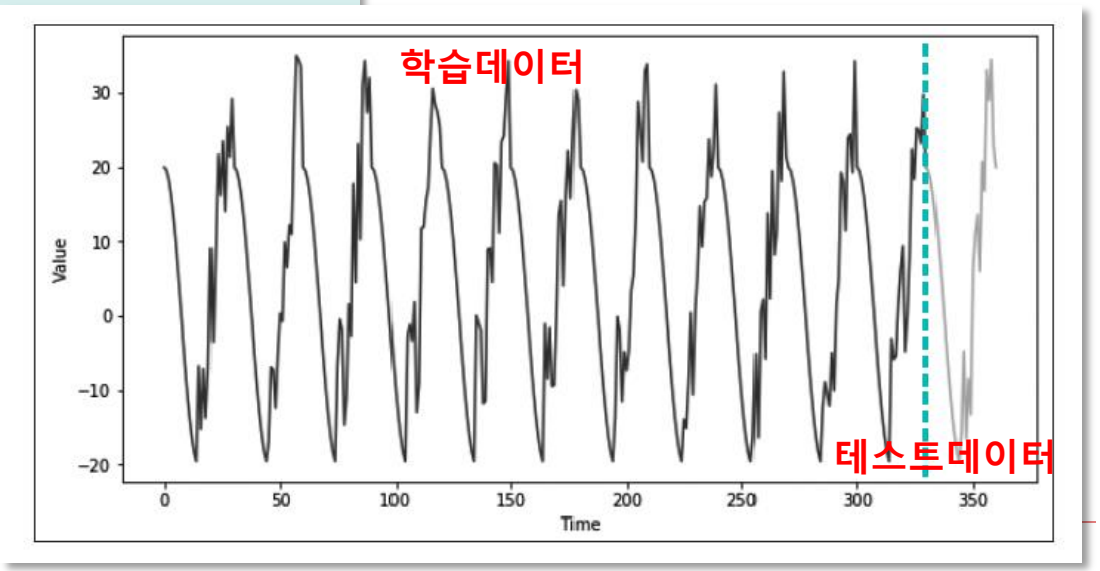


사인 함수와 코사인 함수

출처 : https://ko.wikipedia.org/wiki/%EC%82%BC%EA%B0%81%ED%95%A8%EC%88%98

- 360일 데이터, 주기는 한달(30일)

```
06  time = np.arange(30 * 12 + 1)
07  month_time = (time % 30) / 30
08  time_series = 20 * np.where(month_time < 0.5,
09                              np.cos(2 * np.pi * month_
10                              np.cos(2 * np.pi * month_
```



RNN – 데이터 살펴보기

- 시간적 순서가 존재하도록 make_sequence() 함수를 정의

```
04      for i in range(len(time_series)):
05          x = time_series[i:(i + n)]
06          if (i + n) < len(time_series):
07              x_train.append(x)
08              y_train.append(time_series[i + n])
```

  - 위 함수를 사용하면 다음과 같은 결과를 얻을 수 있음

```
[ 1  2  3  4  5  6  7  8  9 10] | 11
[ 2  3  4  5  6  7  8  9 10 11] | 12
[ 3  4  5  6  7  8  9 10 11 12] | 13
```

  - 이를 통해 우리가 구성한 모델은 시간에 관련한 패턴을 학습할 수 있음
  - make_sequence() 함수에서 시퀀스 길이를 결정하는 파라미터 n은 하이퍼파라미터로 학습 성능에 영향을 줌

```
01  def make_sequence(time_series, n):
```

RNN – 모델 구성하기

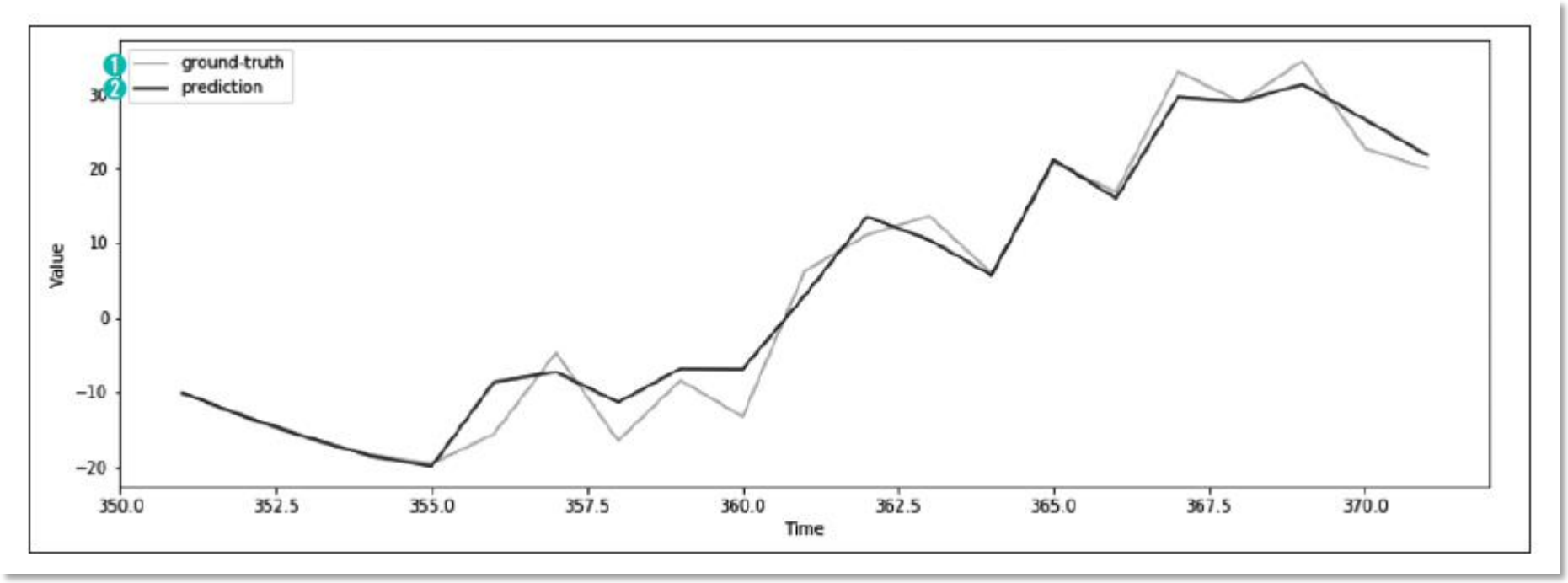
- SimpleRNN층 사용
  - **(batch_size, timesteps, input_dim)** 형태를 입력으로 받으며, **(batch_size, units)** 형태를 출력
  - timesteps는 하나의 샘플에 포함된 정보의 개수를 의미
  - 입력 형태가 Embedding층의 출력 형태와 동일한 것을 확인

```
04  model = Sequential()
05  # SimpleRNN층을 첫 번째 층으로 사용하는 경우,
06  # 반드시 input_shape를 명시해주어야 합니다.
07  model.add(SimpleRNN(units = 32, activation = 'tanh', input_shape = (n, 1)))
```

  - input_shape = (n, 1), n은 timesteps를 1은 데이터 특성 개수를 의미
  - 활성화 함수는 주로 tanh 또는 relu를 사용하며, 기본값은 tanh

RNN – 모델 학습하고 평가하기

- 일부 맞추지 못한 결과가 보이지만, 납득할 수준
  - 텍스트 전처리와 Embedding층이 추가됩니다.



RNN – 여러 개 쌓아보기(실습)

- SimpleRNN층을 여러 개 연결하기 위해서는 추가 인자를 설정
  - **recurrent_sequence**
  - True일 경우, RNN 셀의 전체 상태를 반환
  - 드롭아웃 사용을 위해 **dropout, recurrent_dropout** 인자를 사용
  - 제공되는 코드를 통해 IMDB 데이터셋에 적용해보세요

```
16  model = Sequential()
17  model.add(Embedding(input_dim = num_words, output_dim = 32))
18  # 새로운 인자 세 개가 사용되었습니다.
19  model.add(SimpleRNN(32, return_sequences = True, dropout = 0.15,
                        recurrent_dropout = 0.15))
20  model.add(SimpleRNN(32))
```

LSTM – 원리 이해하기

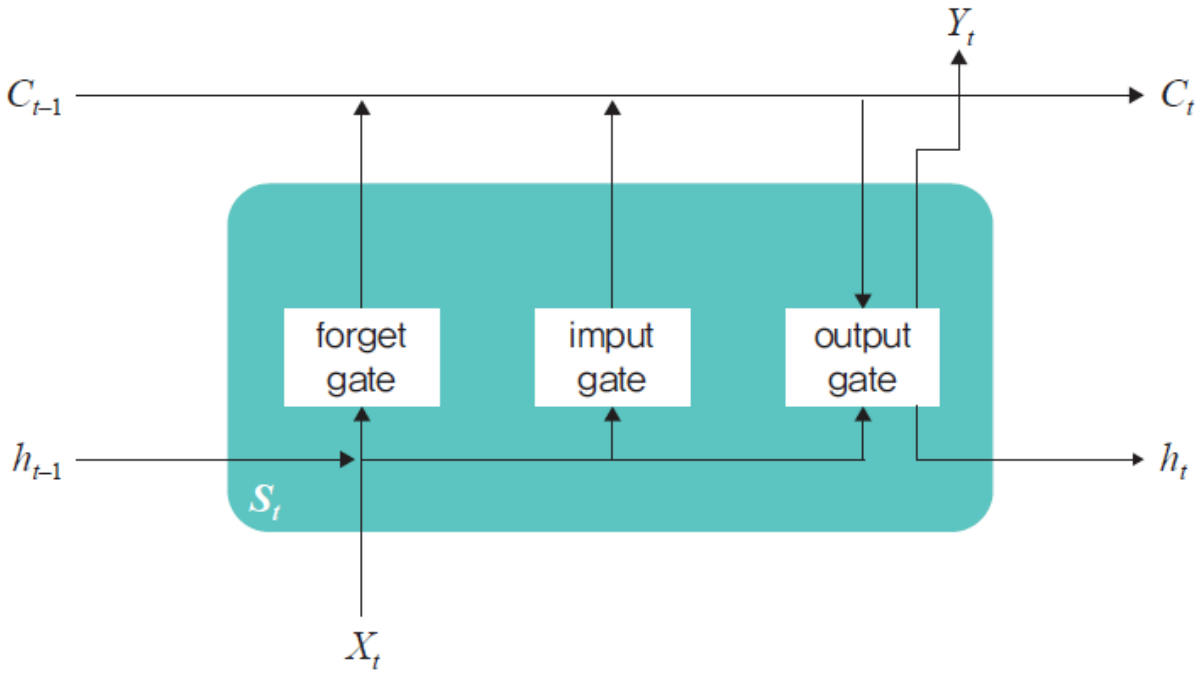
- 단순한 순환 신경망의 문제점
  - '시점이 흐를수록 지속해서 기억하지 못한다'
  - **그래디언트 손실 문제**
  - **이러한 문제는 chatGPT와 같은 LLM(Large Language Model)에서도 아직 근본적으로 해결되지 않았음**

- 이를 해결하기 위해 1997년 고안된 방법
  - **LSTM(Long Short-Term Memory)**
  - Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. Neural computation, 9(8), 1735-1780.

- 여전히 많은 사례에서 사용되고 있으며, 이를 변형한 여러 가지 방법이 존재

LSTM – 원리 이해하기

- LSTM의 핵심은 **정보를 여러 시점에 걸쳐 나르는 장치**가 추가되었다는 것
  - 그래디언트를 보존할 수 있어 그래디언트 손실 문제가 발생하지 않도록 도와줌
  - 아래 그림에서 C로 표현된 **'Cell State'**, h는 hidden state
  - 정보 나르는 것을 도와줄 세 개의 게이트: **forget gate, input gate, output gate**



[그림 6-6] LSTM의 구조

LSTM – 원리 이해하기

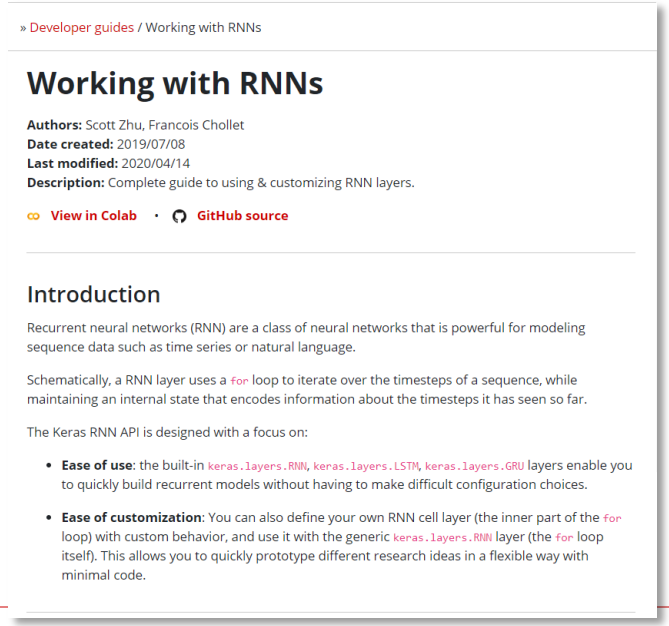
게이트	내용[12]	
forget_gate	수식	f_t = sigmoid(dot(x_t, Wf) + dot(h_{t-1}, Uf) + bf)[13]
	설명	'Cell state'가 나르는 정보 중, 관련 없는 정보를 시그모드 함수를 사용하여 제거합니다(f_t). 'Cell state'는 여기서 생성된 값과 원소별 곱을 통해 정보를 제거하게 됩니다. 곱은 아래 'input_gate'의 수식에서 볼 수 있습니다(f_t * C_{t-1}).
input_gate	수식	i_t = sigmoid(dot(x_t, Wi) + dot(h_{t-1}, Ui) + bi) C_t = f_t * C_{t-1} + i_t * tanh(dot(x_t, Wc) + dot(h_{t-1}, Uc) + bc)
	설명	input_gate는 두 가지 작업을 수행합니다. 첫 번째, 현재 시점의 정보(x_t)와 이전 시점의 상태(h_{t-1})에 시그모이드 함수를 활용하여 어떤 정보를 업데이트할지 결정합니다. 두 번째, 현재 시점의 정보와 이전 시점의 상태에 tanh 함수를 활용하여 새로운 정보를 만듭니다. 이 둘을 곱한 뒤, 'forget gate'를 통해 걸러진 정보와 더해져 현재 시점의 'Cell state'를 만들게 됩니다(C_t).
output_gate	수식	o_t = sigmoid(dot(x_t, Wo) + dot(h_{t-1}, Uo) + bo) h_t = o_t * tanh(C_t)
	설명	'output gate'는 출력값과 현재 시점의 상태 h_t를 만듭니다. h_t는 현재 시점의 정보와 이전 시점의 상태에 시그모이드 함수를 통과시켜 얻은 값과 tanh 함수를 통과한 'Cell state' 값을 곱해 만들어집니다. 또, h_t는 그림의 h_t와 동일하며, 결괏값을 만들기 위해 활성화 함수를 통과한 h_t는 그림의 y_t와 동일합니다.

LSTM – 원리 이해하기

- 신경망 공부를 시작했지만, 수학은 너무 어려워…
  - (우리에게 중요) 식을 전부 기억할 필요는 없음
  - 다만, 모든 연산이 'Cell State'를 중심으로 이루어진다는 것과 LSTM의 핵심적인 기능을 기억!

- LSTM은 워낙 유명한 모델이라 이를 잘 설명한 많은 글들이 있습니다.
  - 다른 표현들로 설명되어 있으니, 종합하여 나만의 것으로 만드세요.
  - 이해하고 사용하는 것과 이해하지 않고 사용하는 것은 엄청난 차이를 가져다 줍니다.

- 데이터를 다뤄봅시다!
  - 추가로, 케라스 공식 홈페이지에도 좋은 튜토리얼이 존재합니다
    - https://keras.io/guides/working_with_rnns/



LSTM – 데이터 살펴보기

- reuters 데이터셋
  - IMDB 데이터셋과 유사함
  - 총 11,258개의 뉴스 기사가 46개 카테고리로 이루어져 있는 다중 분류 문제

- 데이터 다운로드

[함께 해봐요] **reuters 데이터셋 다뤄보기** use_LSTM_layer.ipynb

```
01 from tensorflow.keras.datasets import reuters
02
03 num_words = 10000
04 (X_train, y_train), (X_test, y_test) = reuters.load_data(num_words=num_words)
05
06 print(X_train.shape, y_train.shape)
07 print(X_test.shape, y_test.shape)
```

```
(8982,) (8982,)
(2246,) (2246,)
```

- 문장 길이 동일하게 맞추기

```
05 pad_X_train = pad_sequences(X_train, maxlen=max_len)
06 pad_X_test = pad_sequences(X_test, maxlen=max_len)
```

LSTM – 모델 구성하기

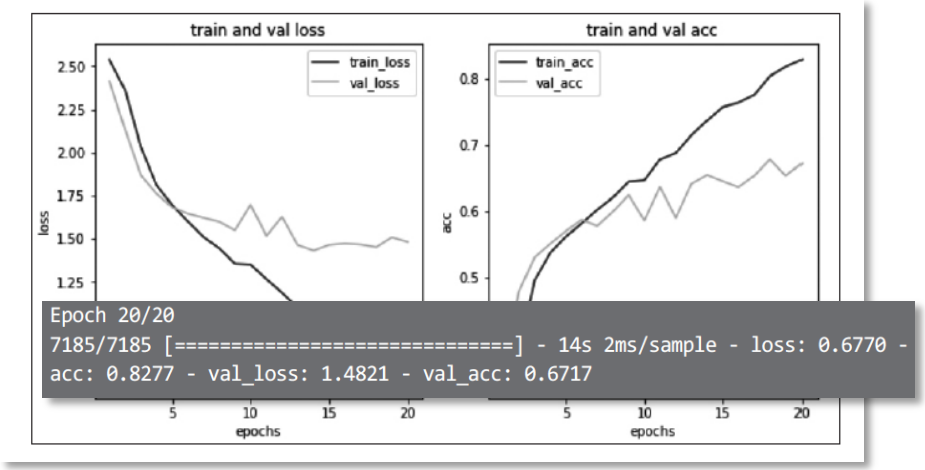
- LSTM은 설정할 수 있는 여러 가지 인자가 있지만, 특별한 상황이 아니라면 케라스가 제공하는 기본값 그대로를 사용해도 무방함
  - LSTM을 여러 개 쌓기 위해 이전처럼 return_sequences 인자를 True로 전달

```
04  model = Sequential()
05  model.add(Embedding(input_dim = num_words, output_dim = 64))
06  model.add(LSTM(64, return_sequences = True))
07  model.add(LSTM(32))
```

- (책 내용 참고) IMDB 데이터셋에서 사용 결과, 검증 데이터셋에서 더 높은 성능을 얻었음

LSTM – 모델 학습하고 평가하기

- retures 데이터셋 결과
  - 20 에폭, 검증 데이터셋의 정확도: 약 67%



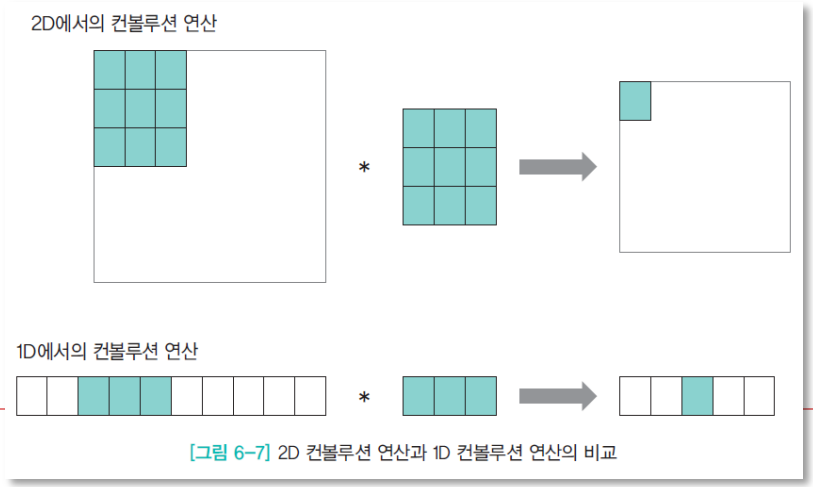
- LSTM뿐만 아니라 케라스가 제공하는 다양한 층을 사용해보세요
  - 과거의 정보뿐만 아니라 미래 정보까지 학습에 사용하는 **Bidirectional RNN**
  - LSTM을 변형한 **GRU**

Conv1D – 원리 이해하기

- 5장에서 지겹게 만나보았던 컨볼루션이 그리웠을거라고 생각합니다
  - 왜 또 나와서 우리를 괴롭힐까?
  - 시퀀스 데이터도 컨볼루션 방법을 통해 특징을 추출할 수 있습니다

- Conv1D층
  - 1차원 형태의 컨볼루션 필터를 가지는 1D 컨볼루션층
  - LSTM과 혼합하여 사용되는 등 순환 신경망과 함께 사용됨

- Conv2D: **2D 형태를 가진 컨볼루션 필터**로 컨볼루션 연산을 수행하여 **이미지 특징을 추출**
  Conv1D: **1D 형태를 가진 컨볼루션 필터**로 컨볼루션 연산을 수행하여 **시퀀스 데이터의 특징을 추출**



[그림 6-7] 2D 컨볼루션 연산과 1D 컨볼루션 연산의 비교

Conv1D – 원리 이해하기

- "나는 좋다"라는 문맥을 문장의 앞부분에서 인식했다면, 동일하게 뒷부분에서도 인식할 수 있음
  - 5장에서 설명한 이동 불변성이 기억나야 함

- 5장에서 사용한 최대 풀링층(Max Pooling), 여기서도 사용!
  - 차원만 바뀜
  - MaxPooling1D층
  - 코드상에서 MaxPooling1D, MaxPool1D는 동일한 층

Conv1D – 모델 구성 및 학습

- IMDB 데이터셋과 cos 데이터를 활용

- Conv1D층은 **(batch_size, timesteps, channels)** 형태를 입력으로 받으며, **(batch_size, timesteps, filters)** 형태를 출력

```
04  model = Sequential()
05  # 이 층은 모델의 제일 첫 번째 층으로만 사용할 수 있습니다.
06  # Flatten층을 사용하기 위해 input_length를 전달합니다.
07  model.add(Embedding(input_dim = num_words, output_dim = 32,
                        input_length = max_len))
08  model.add(Conv1D(32, 7, activation = 'relu'))
09  model.add(MaxPooling1D(7))
10  model.add(Conv1D(32, 5, activation = 'relu'))
11  model.add(MaxPooling1D(5))
12  model.add(GlobalMaxPooling1D())
```

Conv1D – 모델 구성 및 학습

이번에 사용해본 모델의 특징

- 컨볼루션 층하면 주로 3x3 필터 크기를 사용한다고 했고, 그렇게 사용해왔음.
  하지만 여기선 **비교적 큰 필터 크기인 5와 7을 사용**
  - 예를 들어, 문장 맥락 파악을 위해 많은 개수의 단어를 보면 맥락을 파악하기가 더 쉬울것
  - 그렇다고 해서 적은 개수의 단어를 보았을 때, 문장의 맥락을 파악하기 어렵다는 의미도 아니며, 많은 개수의 단어를 보았을 때 정확히 파악할 것이라는 보장은 하지 못함
  - **이를 해결하기 위해 [3, 5, 7] 또는 그 외의 크기를 사용하여 다양한 실험을 해보는 것을 추천**
    - **2D 컨볼루션층과 다르게 필터의 크기를 증가시킨다고 하여 모델의 크기가 급격히 증가하지 않기 때문**
  - **적절히 타협하여 크기 5를 사용**하거나, **여러 크기를 사용한 후에 Concatenate층을 사용하여 결과를 전부 병합하는 방법**도 존재
- **전역 최대 풀링층 사용**
  - GlobalMaxPooling1D층
  - 배치 차원을 제외하고 2차원 형태를 1차원 형태로 변환, Max → 차원별로 최댓값 특징을 사용
  - Flatten층을 사용해도 무방

Conv1D – 모델 구성 및 학습

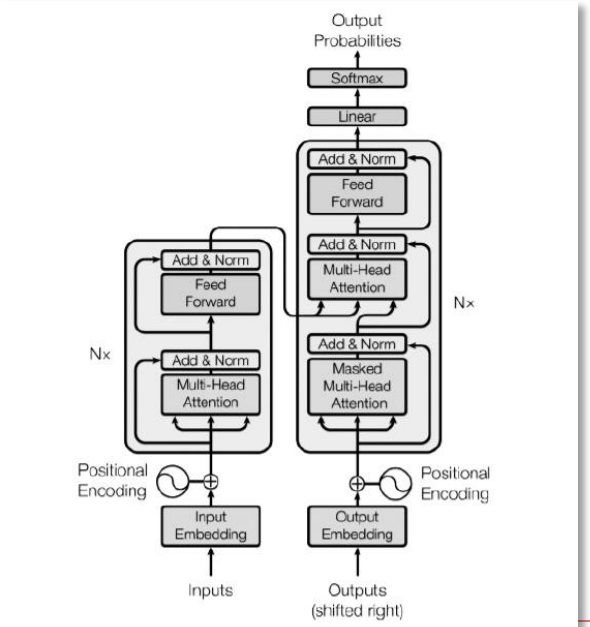
- Conv1D층과 LSTM층 혼합하여 사용하기
  - Conv-LSTM 모델

```
05  model = Sequential()
06  model.add(Conv1D(32, 3, activation = 'relu', input_shape = (10, 1)))
07  model.add(MaxPooling1D(2))
08  model.add(Conv1D(32, 3, activation = 'relu'))
09  # LSTM을 혼합하여 모델을 구성합니다.
10  model.add(LSTM(32, dropout = 0.2, recurrent_dropout = 0.2))
```

BERT 가볍게 알아보기

- **BERT(Bidirectional Encoder Representations from Transformers)**
  - 이름이 매우 김, 그냥 버트라고 읽자
  - 자연어 처리 분야에서 최고 성능을 보여주었고, 활발하게 연구, 활용되고 있는 방법
  - 2018년 10월 구글에서 발표했으며, Transformer 형태를 이용
  - 25억 개의 Wikipedia 단어와 8억 개의 BooksCorpus 단어를 사용

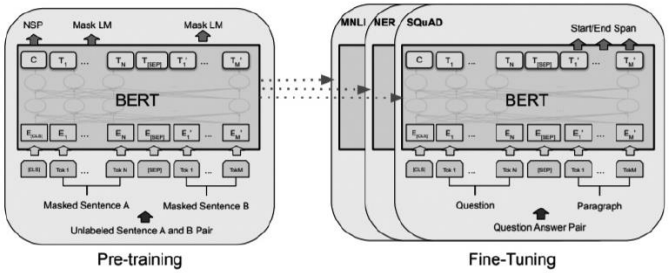
- BERT는 질문-대답 문제(QA; Question & Answering)를 다루는 SQuAD와 KorQuAD에 도전함과 동시에 상위권을 독식한 훌륭한 방법
  - SQuAD: Stanford Question Answering DataSet
  - KorQuAD: The Korean Question Answering DataSet



[그림 6-8] Transformer의 구조(그림 출처: 각주 19번의 논문)

BERT 가볍게 알아보기

- BERT가 주로 사용되기 이전의 자연어 처리 분야에서는 **사전 학습을 사용할 적당한 방법이 존재하지 않았음**
  - 사전 학습된 모델의 힘은 이미 5장에서 겪어보았죠?
  - BERT는 5장에서 사용한 모델 구조와 동일하게, 사전 학습된 BERT의 마지막 부분을 재정의하여 사용



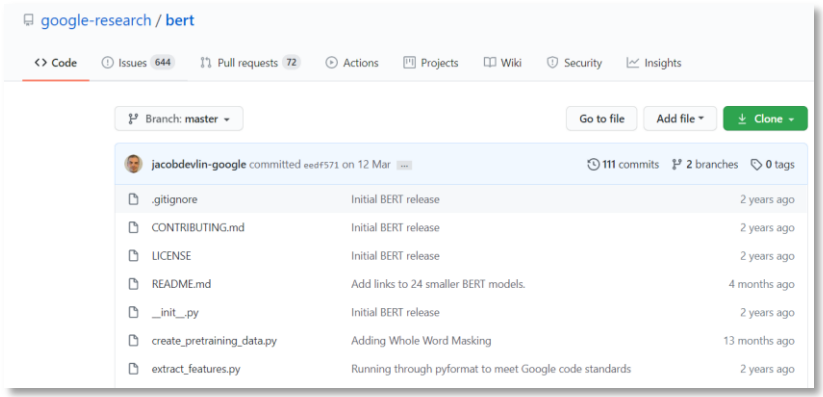
[그림 6-9] BERT는 사전 학습이 가능한 강력한 모델입니다.(그림 출처: 각주 16번의 논문)

  - 이를 활용하여 자연어 처리 연구에서 매우 탁월한 성능을 보여줌
  - 또한, **다양한 분야에서도 높은 성능**을 보여주면서 **활용성의 범위가 매우 넓다**는 것을 증명
    - 최근 개발된 모델도 이와 같은 특징을 보여주려고 매우 노력, ex) 일반인이 활용하기 어려여누 **GPT-3, Bard**
    - Q. 왜 일반인 활용이 어려울까? A. 구현은 가능. 그러나 돈이 매우 많이 필요!

- **한 분야에서 뛰어난 성능을 보여준 방법이 다른 분야에서도 뛰어난 성능을 보여줄 수 있다는 것은 딥러닝의 매~우 강력한 장점이다 ☞ 2023년에도 여전히 이를 증명하는 중(ing)**

BERT 가볍게 알아보기

- 자연어 처리 문제를 해결하고 싶다면, BERT 활용을 고민해보는 것은 어떨까요?
  - https://github.com/google-research/bert



- (추가) 이미지에 관심이 있어도, 자연어 처리에 관심이 있어도
  Transformer 개념은 꼭! 알아두기를 권장합니다.
  - 논문 제목:
    Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. In Advances in neural information processing systems (pp. 5998-6008).

요약 정리

1. 순환 신경망은 **시퀀스 또는 시계열 데이터 처리에 특화**되어 있습니다.
2. Embedding층은 **수많은 단어(또는 데이터)를 표현**할 수 있습니다. 항상 모델의 첫 번째 층으로만 사용할 수 있습니다.
3. Embedding층은 (batch_size, sequence_length) 형태를 입력으로 받으며, (batch_size, sequence_length, output_dim) 형태를 출력합니다.
4. Embedding층은 **단어의 관계와 맥락을 파악할 수 없습니다**. 이를 해결하기 위해 사용되는 것이 SimpleRNN층입니다. SimpleRNN층은 순환 신경망의 가장 기본적인 형태를 나타내며, 출력값의 업데이트를 위해 **이전 상태를 사용**합니다.
5. SimpleRNN층은 (batch_size, timesteps, input_dim)의 형태를 입력으로 받으며, (batch_size, units)를 출력합니다.
6. SimpleRNN층은 **그래디언트 손실 문제를 야기**합니다. 이를 해결하기 위해 고안된 것이 LSTM입니다. LSTM은 **과거의 정보를 나르는 'Cell State'**를 가지고 있으며, 정보를 제거 또는 제공하기 위한 input_gate, forget_gate, output_gate를 보유하고 있습니다.

요약 정리

7. Conv2D층을 통해 이미지 데이터의 특징을 추출할 수 있었다면, Conv1D층을 통해 시퀀스 데이터의 특징을 추출할 수 있습니다.
8. Conv1D층은 (batch_size, timesteps, channels) 형태를 입력으로 받으며, (batch_size, timesteps, filters) 형태를 출력합니다.
9. BERT는 자연어 처리 분야에서 최고의 성능을 달성한 모델입니다. 자연어 처리뿐만 아니라 다양한 분야에서 뛰어난 성능을 보여주고 있기 때문에 충분히 관심을 가져볼 만한 방법입니다.