Week2

CH15 - RNN과 CNN을 사용해 시퀀스 처리하기

RNN

- 어느 정도 미래를 예측할 수 있는 네트워크
- 고정 길이 입력 X
- 임의 길이를 가진 시퀀스
- 문장, 문서, 오디오 샘플
- NLP에 유용(자동 번역, 스피치 투 텍스트)

15.1 순환 뉴런과 순환층

- 피드포워드 신경망과 비슷하지만 뒤쪽으로 순환하는 연결 有
- 입력 → 출력 (자기 자신에게도 출력을 보냄)
- 타임 스텝마다 $x_{(t)}$와 이전 타임 스텝 출력인 $y_{(t-1)}$을 입력으로 받음

  첫 번째는 이전 출력이 없기 때문에 0으로 설정

→ 시간에 따라 네트워크를 펼쳤다

15.1.1 메모리 셀

순환 뉴런의 출력 → 일종의 **메모리** 형태 (이전 타임 스템의 모든 입력에 대한 함수이므로)

메모리 셀: 타임 스텝에 걸쳐 어떤 상태를 보존하는 신경망의 구성 요소

셀의 상태 = $h_{(t)}$

- h는 hidden을 의미
- 타임 스텝의 입력과 이전 스텝의 상태에 대한 함수 → $h_{(t)} = f(h_{(t-1)}, x_{(t)})$
- 출력 $y_{(t)}$와 같을 수도 다를 수도 있음

15.1.2 입력과 출력 시퀀스

- **시퀀스-투-시퀀스 네트워크**

  입력 시퀀스를 받아 출력 시퀀스 생성

  ex. 시계열 데이터 예측에 유용
- **시퀀스-투-벡터 네트워크**

입력 시퀀스 주입 → 마지막을 제외한 모든 출력 무시

ex. 영화 리뷰의 연속된 단어에서 감성 점수 출력

- **벡터-투-시퀀스 네트워트**

  하나의 입력 벡터 반복 주입 → 하나의 시퀀스 출력

  ex. 이지미를 입력해 이미지에 대한 캡션을 출력

- **인코더-디코더**

  인코더: 시퀀스-투-벡터 네트워크

  디코더: 벡터-투-시퀀스 네트워트

  → 인코더 뒤에 디코더를 연결 (ex. 한 언어의 문장을 다른 언어로 번역)

15.2 RNN 훈련하기

타임 스템으로 네트워크를 펼치고 보통의 역전파 사용

→ **BPTT** (BackPropagation Through Time)

15.3 시계열 예측하기

시계열: 타임 스텝마다 하나 이상의 값을 가진 시퀀스

- **단변량 시계열**

  타임 스텝마다 하나의 값(ex. 시간당 사용자 접속자수)

- **다변량 시계열**

  타임 스텝마다 여러 값(ex. 회사의 재정 안정성 연구 - 회사의 수입, 부채 등등)

과거 데이터에서 누락된 값 예측 → **값 대체**

15.3.1 기준 성능

1. 기준 성능 준비

   각 시계열의 마지막 값을 그대로 예측 → **순진한 예측**

2. 완전 연결 네트워크 사용

   1차원 특성 배열 기대 → Flatten 층 추가

15.3.2 간단한 RNN 구현하기

```
model = keras.models.Sequential([
    keras.layers.SimpleRNN(1, input_shape=[None, 1])
])
#tanh 함수 사용
#mse = 0.014 -> 선형모델 보다 안 좋은 결과
```

<트렌드와 계절성>

- 가중 이동 평균
- 자동 회귀 누적 이동 평균

→ **차분**: 시계열에서 트렌드와 계절성을 삭제해야 한다 (최종 예측을 할 때 다시 더함)

→ 성능을 향상시킬 수 있는 방법이지만 RNN에서는 딱히 필요 없음...

15.3.3 심층 RNN

```
model = keras.models.Sequential([
    keras.layers.SimpleRNN(20, return_sequences=True, input_shape=[None, 1]),
    keras.layers.SimpleRNN(20, return_sequences=True),
    keras.layers.SimpleRNN(1)
])
```

```
#mse = 0.003 (선형모델보다 좋은)

#마지막 층은 이상적이지 X -> dense 층으로 변경

model = keras.models.Sequential([
    keras.layers.SimpleRNN(20, return_sequences=True, input_shape=[None, 1]),
    keras.layers.SimpleRNN(20),
    keras.layers.Dense(1)
])
```

15.3.4 여러 타임 스텝 앞을 예측하기

1. 이미 훈련된 모델을 사용

   다음 값을 예측 → 이 값을 입력으로 추가

```
#오차 누적
series = generate_time_series(1, n_steps + 10)
X_new, Y_new = series[:, :n_steps], series[:, n_steps:]
X = X_new
for step_ahead in range(10):
    y_pred_one = model.predict(X[:, step_ahead:])[:, np.newaxis, :]
    X = np.concatenate([X, y_pred_one], axis=1)

Y_pred = X[:, n_steps:]
```

→ 효율성 ↓, 성능도 나쁨

2. RNN을 훈련

   한번에 값을 예측하는 것

```
n_steps = 50
series = generate_time_series(10000, n_steps + 10)
X_train, Y_train = series[:7000, :n_steps], series[:7000, -10:, 0]
X_valid, Y_valid = series[7000:9000, :n_steps], series[7000:9000, -10:, 0]
X_test, Y_test = series[9000:, :n_steps], series[9000:, -10:, 0]

model = keras.models.Sequential([
    keras.layers.SimpleRNN(20, return_sequences=True, input_shape=[None, 1]),
    keras.layers.SimpleRNN(20),
    keras.layers.Dense(10)
])

#시퀀스 투 벡터 -> 시퀀스 투 시퀀스
#타깃 시퀀스
#인과 모델

Y = np.empty((10000, n_steps, 10))
for step_ahead in range(1, 10 + 1):
    Y[..., step_ahead - 1] = series[..., step_ahead:step_ahead + n_steps, 0]
Y_train = Y[:7000]
Y_valid = Y[7000:9000]
Y_test = Y[9000:]

model = keras.models.Sequential([
    keras.layers.SimpleRNN(20, return_sequences=True, input_shape=[None, 1]),
    keras.layers.SimpleRNN(20, return_sequences=True),
    keras.layers.TimeDistributed(keras.layers.Dense(10)) #모든 타임 스텝에서 출력을 Dense 층에 적용
])
```

15.4 긴 시퀀스 다루기

15.4.1 불안정한 그레이디언트 문제와 싸우기

- 가중치 초기화, 옵티마이저, 드롭아웃 등
- 수렴하지 않는 활성화 함수(ReLU) 도움 X

  경사하강법 → 출력이 폭주 or 그레디언트 자체가 폭주
- 배치 정규화 (딱히 효과 없음)
- **층 정규화**

특성 차원에 대해 정규화

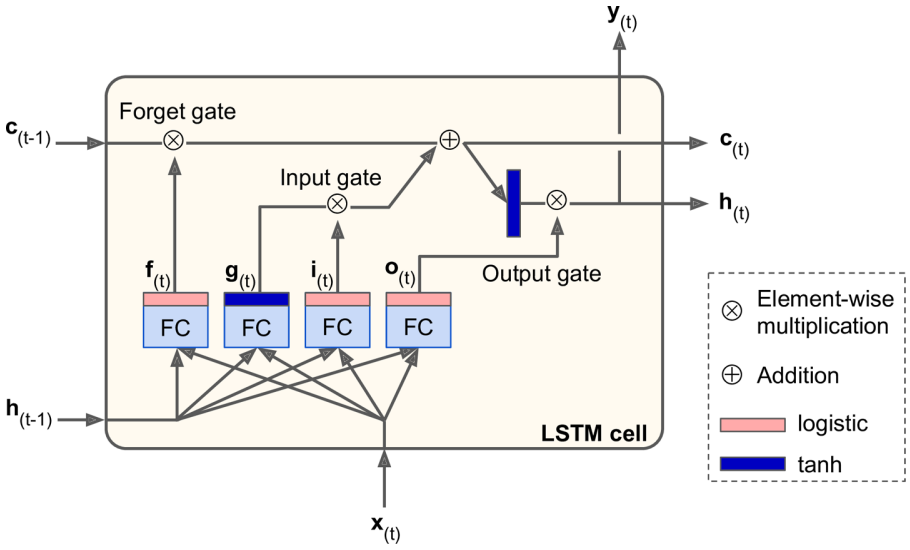
훈련과 테스트에서 동일하게 작동

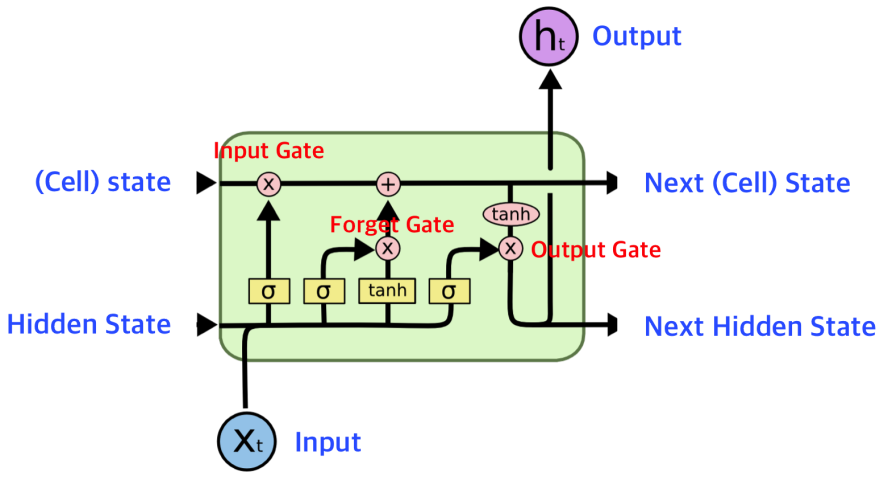
15.4.2 단기 기억 문제 해결하기

<LSTM 셀>

- 장단기 메모리
- 훈련이 빠르게 수렴하고 데이터에 있는 장기간의 의존성 감지
- LSTM 층 사용

```
model = keras.models.Sequential([
    keras.layers.LSTM(20, return_sequences=True, input_shape=[None, 1]),
    keras.layers.LSTM(20, return_sequences=True),
    keras.layers.TimeDistributed(keras.layers.Dense(10))
])
```





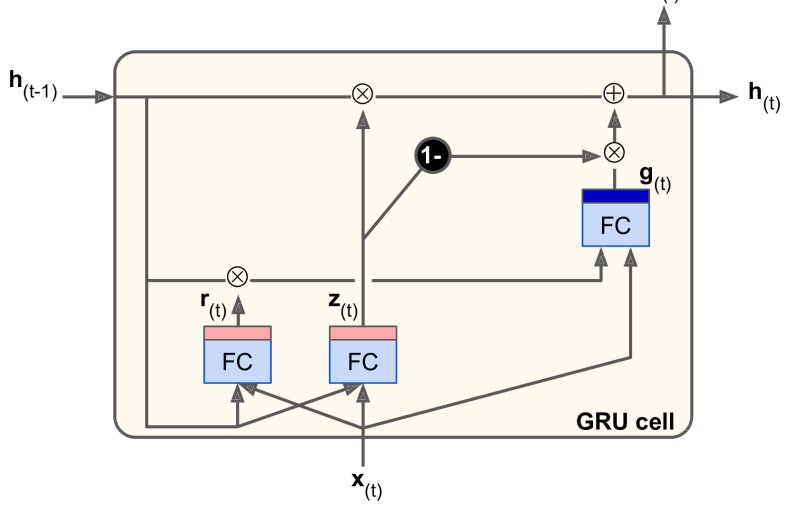
<핍홀 연결>

- 게이트 제어기에 장기 상태도 조금 노출 → 더 많은 문맥 감지
- LSTM 층이 핍홀 지원 X

  (`keras.layers.LSTMCell` 기반 또는 `tf.keras.experimental.PeepholeLSTMCell`층 사용)

<GRU 셀>

- 게이트 순환 유닛




```
model = keras.models.Sequential([
    keras.layers.GRU(20, return_sequences=True, input_shape=[None, 1]),
    keras.layers.GRU(20, return_sequences=True),
    keras.layers.TimeDistributed(keras.layers.Dense(10))
])
```

- 제한적인 단기 기억

  100타임 스텝 이상의 시퀀스에서 장기 패턴 학습 어려움 - 입력 시퀀스를 짧게(1D 합성곱 층)

<1D 합성곱 층을 사용해 시퀀스 처리하기>

- 커널마다 1D 특성맵 출력
- 스트라이드 = 1 & 'same' 패딩 : 출력 시퀀스 길이 = 입력 시퀀스 길이
- 스트라이드 > 1 & 'valid' 패딩 : 출력 시퀀스 길이 < 입력 시퀀스 길이

```
1D conv layer with kernel size 4, stride 2, VALID padding:

              |-----2-----|     |-----5---...------|     |-----23----|
        |-----1-----|     |-----4-----|   ...      |-----22----|
  |-----0----|      |-----3-----|     |---...|-----21----|
X: 0  1  2  3  4  5  6  7  8  9  10 11 12 ... 42 43 44 45 46 47 48 49
Y: 1  2  3  4  5  6  7  8  9  10 11 12 13 ... 43 44 45 46 47 48 49 50
  /10 11 12 13 14 15 16 17 18 19 20 21 22 ... 52 53 54 55 56 57 58 59

Output:

X:     0/3   2/5   4/7   6/9   8/11 10/13 .../43 42/45 44/47 46/49
Y:     4/13  6/15  8/17 10/19 12/21 14/23 .../53 46/55 48/57 50/59
```

```
model = keras.models.Sequential([
    keras.layers.Conv1D(filters=20, kernel_size=4, strides=2, padding="valid",
                        input_shape=[None, 1]),
    keras.layers.GRU(20, return_sequences=True),
    keras.layers.GRU(20, return_sequences=True),
    keras.layers.TimeDistributed(keras.layers.Dense(10))
])
```

<WAVENET>

- 층마다 팽창 비율을 두 배로 늘리는 1D 합성곱 층 쌓기

```
C2  /\ /\ /\ /\ /\ /\ /\ /\ /\ /\ /\ /\ /\ /\ /\ /\...  /\ /\ /\ /\ /\ /\
   \  /  \  /  \  /  \  /  \  /  \  /  \  /  \  /  \       /  \  /  \  /  \
     /    \      /    \      /    \                 /    \
C1  /\ /\ /\ /\ /\ /\ /\ /\ /\ /\ /\  /\ /.../\ /\ /\ /\ /\ /\ /\ /\
X: 0  1  2  3  4  5  6  7  8  9  10 11 12 ... 43 44 45 46 47 48 49
Y: 1  2  3  4  5  6  7  8  9  10 11 12 13 ... 44 45 46 47 48 49 50
  /10 11 12 13 14 15 16 17 18 19 20 21 22 ... 53 54 55 56 57 58 59
```

```
model = keras.models.Sequential()
model.add(keras.layers.InputLayer(input_shape=[None, 1]))
for rate in (1, 2, 4, 8) * 2:
    model.add(keras.layers.Conv1D(filters=20, kernel_size=2, padding="causal",
                                  activation="relu", dilation_rate=rate))
model.add(keras.layers.Conv1D(filters=10, kernel_size=1))
```

15.5 연습문제