

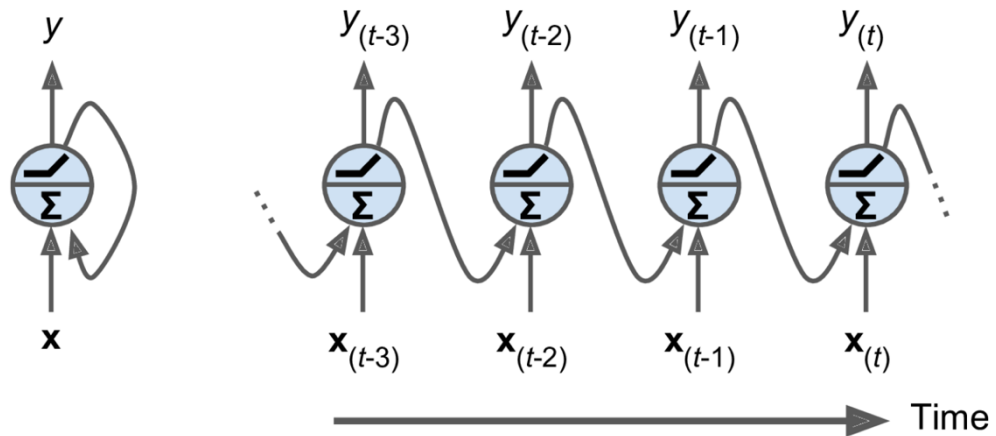
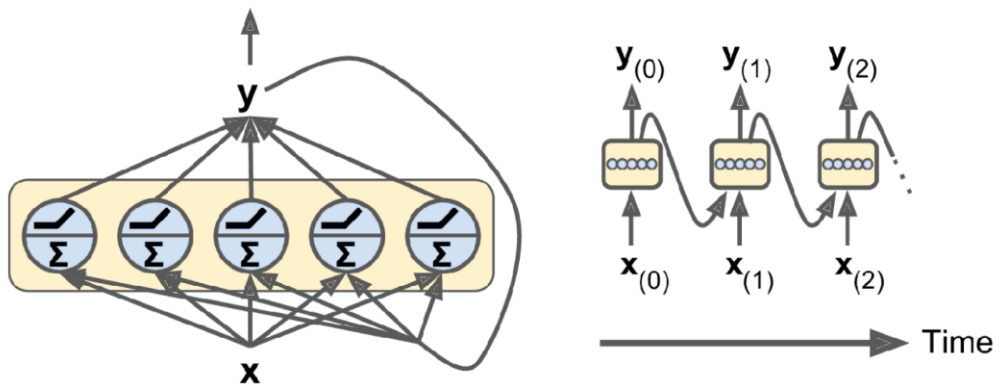


# TAVE 서기

서기 내용			
서기 일자	21.11.20	서기	서가을
주제	순환 신경망 RNN		
시간	20:30 - 22:30	장소	Zoom 미팅
스터디 인원	고성호, 권기호, 이아현, 서가을 : 시작		
			
스터디 인원	고성호, 권기호, 이아현, 서가을 : 종료		
			
내용			
배운 내용	Chapter15 순환 신경망 RNN		
	15.1 순환 뉴런과 순환 층		



- 왼쪽이 가장 간단한 뉴런 한 개로 구성된 RNN 구조
- 오른쪽은 이를 시간으로 표현한 것.
- 각 time step t마다 이전 step의 출력을 입력으로 받음
- 첫 step은 이전 출력이 없으므로 0으로 설정

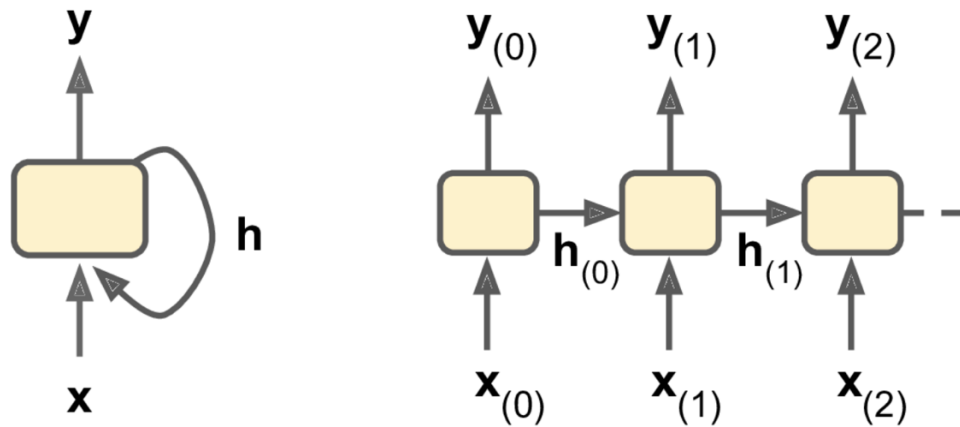


- 왼쪽은 순환 뉴런으로 된 것, 오른쪽은 time step으로 펼친 것
- time step t마다 입력벡터  $x(t)$ 와 출력벡터  $y(t-1)$ 을 받음
- 각 순환 뉴런은 두 개의 가중치 가짐
- 입력  $x(t)$ 를 위한 가중치
- 출력  $y(t-1)$ 을 위한 가중치
- $$Y_t = \phi(X_t W_x + Y_{t-1} W_y + b) = \phi\left(\begin{bmatrix} X_t & Y_{t-1} \end{bmatrix} W + b\right)$$
- (하나의 sample에 대한 전체의 출력 벡터)

#### - 메모리 셀

- Time step에 걸쳐 어떤 상태를 보존하는 신경망의 구조
- (순환뉴런의 출력은 모든 입력에 대한 함수이므로 메모리와 같다)

- 하나의 순환 뉴런 또는 순환 뉴런의 층의 짧은 패턴만 학습할 수 있음.



- 셀의 상태를 나타내는 함수:  $h_t = f(h_{t-1}, x_t)$
- 기본적으로 이전 step의 함수를 받았지만, 복잡한 경우 (hidden state가 있는 경우)에는 다를 수 있음.

#### - 입력과 출력 시퀀스

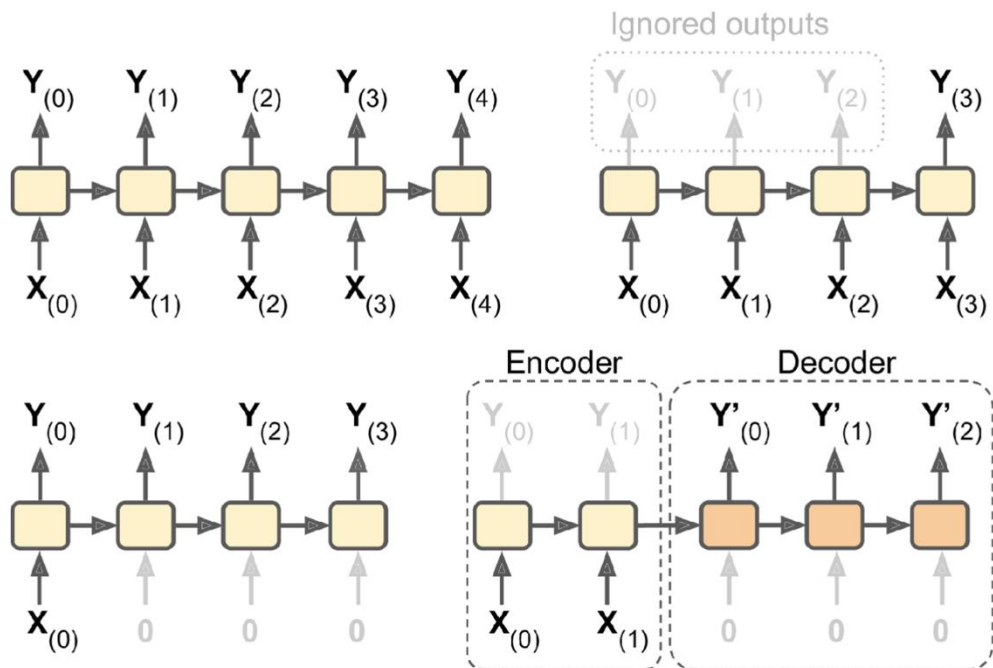


Figure 15-4. Seq-to-seq (top left), seq-to-vector (top right), vector-to-seq (bottom left), and Encoder-Decoder (bottom right) networks

- 좌측 위, 우측 위, 좌측 아래, 우측 아래 순으로 진행
- Sequence to sequence (좌측 위)

- 주식 가격 같은 시계열 데이터 예측에 유용  
(최근 5일치 주식 넣으면 내일을 포함한 5일치 주식 데이터 출력)
- 입력 시퀀스를 받아 출력 시퀀스를 만들 수 있음
- Sequence to vector (우측 위)
- 입력 sequence를 주입하고 마지막 제외 모든 출력을 무시
- Ex. 영화리뷰: 연속된 단어를 주입하면 네트워크는 감성 점수를 출력하는 모델
- Vector to sequence (좌측 아래)
- 각 time step에서 하나의 입력 벡터를 반복해서 네트워크에 주입, 하나의 sequence 출력.
- 이미지를 입력하여 이미지 캡션을 출력하는 경우 사용 가능
- Encoder to Decoder
- Encoder(sequence to vector) 뒤에 Decoder(vector to sequence)네트워크 연결
- 한 단어의 문장을 네트워크에 주입하면 vector로 만들고 Decoder가 이 vector를 다른 언어의 문장으로 Decoding.
- Sequence to sequence보다 번역 성능이 좋음
- 문장의 마지막 단어가 번역의 첫 번째에 영향을 줄 수 있음.
- 번역하기 전 전체 문장이 주입될 때까지 기다리지 않아도 됨.

### - RNN 훈련하기

발표자의 개인사정으로 다음 시간에 발표 예정

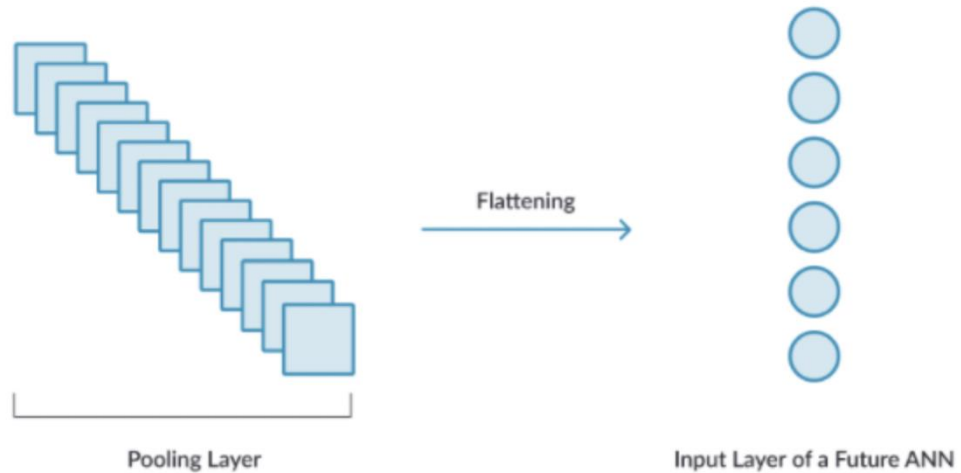
## 15.3 시계열 예측하기

- 시계열: 모든 time step마다 하나 이상의 값을 가진 sequence
- 단변량 시계열: time step마다 하나의 값
- 다변량 시계열 time step마다 여러 개의 값
- 값 대체: 과거 데이터에서 누락된 값을 예측

```
def generate_time_series(batch_size, n_steps):
    freq1, freq2, offsets1, offsets2 = np.random.rand(4, batch_size, 1)
    time = np.linspace(0, 1, n_steps) ## 수평축 간격 만들기
    series = 0.5 * np.sin((time - offsets1) * (freq1 * 10 + 10)) # 웨이브 1
    series += 0.2 * np.sin((time - offsets2) * (freq2 * 20 + 20)) # + 웨이브 2
    series += 0.1 * (np.random.rand(batch_size, n_steps) - 0.5) # + 잡음
    return series[:, :, np.newaxis].astype(np.float32)
```



- 시계열 값의 선형 조합으로 예측하기 위해 선형 회귀 모델 사용
- MSE 손실, Adam 옵티마이저로 컴파일 후 20 에포크 동안 훈련 후 검증세트에서 평가
- 이 경우 오차 0.004
- 이 네트워크는 입력마다 1차원 특성 배열을 기대하므로 flatten 층 추가



- Flatten layer: 추출된 주요 특징을 전결합층에 전달하기 위해 1차원 자료로 바꿔주는 layer
- 이미지 형태의 데이터를 배열 형태로 flatten
- Batch size에 영향을 주지 않음

### 15.3.2 간단한 RNN 구현하기

```
np.random.seed(42)
tf.random.set_seed(42)

##가장 간단하게 만들 수 있는 RNN
model = keras.models.Sequential([
    keras.layers.SimpleRNN(1, input_shape=[None, 1])
])
## 순환 신경망은 타임 스텝의 길이와 상관 없이 모두 처리 가능
## 입력 시퀀스의 길이를 지정할 필요가 없다 => None
optimizer = keras.optimizers.Adam(learning_rate=0.005)
model.compile(loss="mse", optimizer=optimizer)
history = model.fit(X_train, y_train, epochs=20,
                    validation_data=(X_valid, y_valid))
```

- 케라스로 RNN구현하는 법
- 순환 신경망은 어떤 길이의 time step도 처리할 수 있기 때문에 입력 sequence의 길이를 지정할 필요가 없으므로 None으로 설정할 것



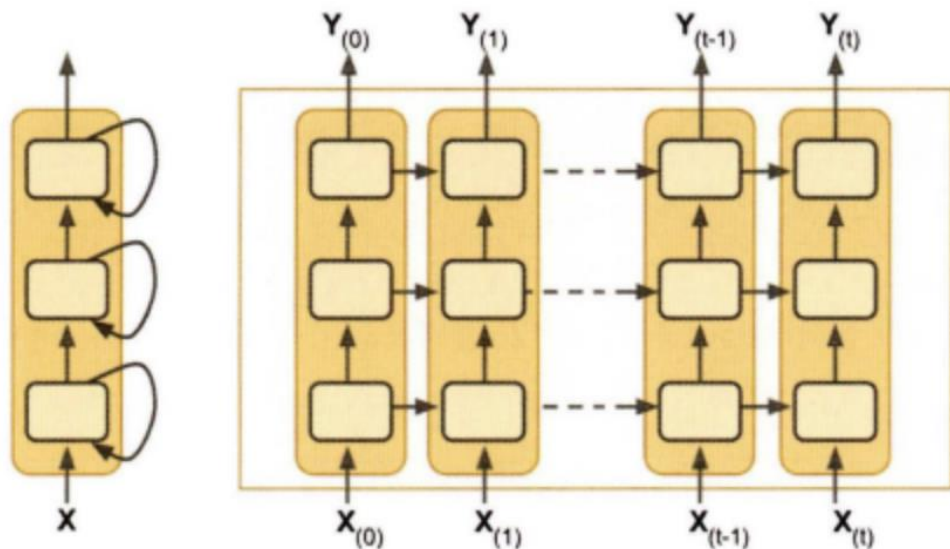
```
model.evaluate(X_valid, y_valid)
```

63/63 [=====] - 1s 8ms/step - loss: 0.0109  
0.010881561785936356

- 순진한 예측보다 성능이 낮지만 간단한 선형모델 앞지르지 못함
- 기본 RNN은 순환 뉴런은 입력과 은닉 차원마다 하나 이상의 파라미터를 가지고 편향이 있음
- [트렌드와 계절성]
- 시계열에서 트렌드 삭제 필요 : 매달 10% 성장하는 웹사이트의 접속 사용자 수
- 시계열에서 계절성 삭제 필요 : 여름에 잘 팔리는 선크림 판매량
- 차분 : 매 타임 스텝의 값과 작년도 값의 차이를 계산하여 시계열에서 계절성을 삭제
- RNN을 사용할 때는 이런 작업들이 모두 필요 없음

### 15.3.3 심층 RNN

- RNN은 여러 층으로 쌓는 것이 일반적 (심층 RNN)



- 왼쪽: 심층 RNN, 오른쪽: time step으로 펼친 모습

```

np.random.seed(42)
tf.random.set_seed(42)

## 세 개의 Simple RNN층을 쌓은 것
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)
])
## 마지막 층을 제외하고는 모든 순환 층에서 return_sequences=True로 설정
## 설정 하지 않으면 마지막 타임 스텝의 출력만 담은 2D 배열이 출력되고
## 다음 순환 층이 3D 형태로 시퀀스를 받지 못하기 때문에 작동하지 못함

model.compile(loss="mse", optimizer="adam")
history = model.fit(X_train, y_train, epochs=20,
                    validation_data=(X_valid, y_valid))

```

- tf. Keras로 심층 RNN을 구현하려면 순환 층을 쌓으면 됨.
- 모든 층에서 return\_sequences = True 설정 해야 함.

```
model.evaluate(X_valid, y_valid)
```

```

63/63 [=====] - 1s 19ms/step - loss: 0.0029
0.002910560928285122

```

- 선형모델의 성능 앞지름
- 하나의 time step에서 다음 time step으로 가기 위해 다음 순환 층의 은닉상태 사용 -> 마지막 층의 은닉 상태는 크게 필요하지 않음.
- SimpleRNN은 기본적으로 활성화함수 tanh 함수여서 예측된 값이 -1~1 사이 값
- 출력층을 Dense로 바꾸면 더 빠르고 정확, 원하는 활성화함수 선택 가능
- Dense층으로 바꾸는 법: 두 번째 순환층에서 return\_sequences = True를 제거
- 출력층을 Dense로 바꾸는 경우가 많음.

#### 15.3.4 여러 타임 스텝 앞을 예측하기

1. 이미 훈련된 모델을 사용하여 다음 값 예측한 다음, 이 값을 입력으로 추가하는 방법

2. RNN을 훈련하여 다음 값 10개를 한번에 예측하는 방법

[첫 번째 방법]



```

np.random.seed(43) # 42는 훈련 세트에 있는 첫 번째 시리즈를 반환하기 때문에 다른 값으로 지정합니다

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:]

```

- 하나씩 다음 10개의 값을 예측
- 다음 10개의 값을 예측하기 위해 9개의 time step을 가진 sequence 생성
- 다음 스텝에 대한 예측은 보통 더 미래의 타임 스텝에 대한 예측보다 정확함(오차가 누적되므로)
- 한 번에 하나의 미래 스텝을 예측하기 위해 RNN을 사용하는 것보다 나옴

[두 번째 방법]

동시에 다음 10개의 값을 모두 예측

- 모든 타임 스텝에서 RNN 출력에 대한 항이 손실에 포함 됨 => 더 많은 오차 그래디언트가 모델로 흐름. 시간에 따라서만 흐름 필요가 없음.
- 훈련이 안정적, 훈련 속도가 높아짐

## 15.4 긴 시퀀스 다루기

- 긴 시퀀스로 RNN 훈련하려면 많은 스텝이 필요 -> 펼친 RNN이 매우 깊은 네트워크 됨.
- 이러한 모델에서 생기는 문제점
  - Gradient vanishing/ gradient exploding
  - 학습시간 증가
  - 긴 시퀀스를 처리할 때 앞부분을 잊어버림

### 15.4.1 불안정한 그래디언트 문제와 싸우기

- 기존의 network에 적용했던 것 처럼 weight initialization, optimizer, dropout 측면에서 다양하게 생각해볼 수 있음
- 수렴하지 않는 ReLU함수를 사용하면 가중치 업데이트를 위한 출력이 증가하다가 폭주할 수 있다.

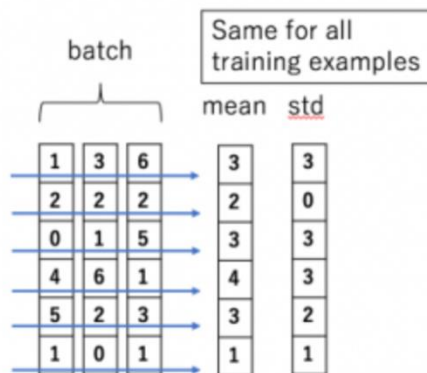
### [해결방법]

- Learning rate를 줄이거나 activation function을 바꿈
- 텐서보드에 gradient 찍어보고 불안하면 gradient clipping 해보기
- Batch normalization 시도

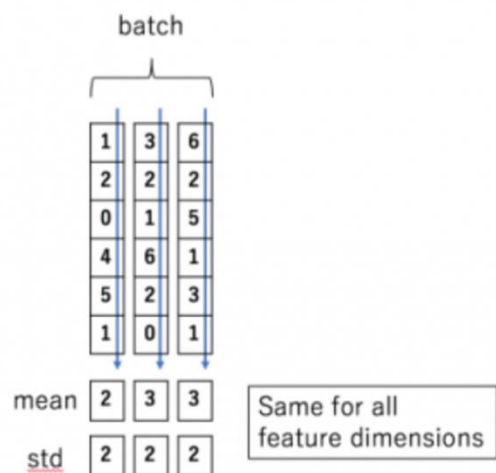
(값의 범위의 차이를 왜곡시키지 않고 데이터셋을 공통 스케일로 변경하는 것

- RNN 효과적으로 사용하지 못할 수 있음.

#### Batch Normalization



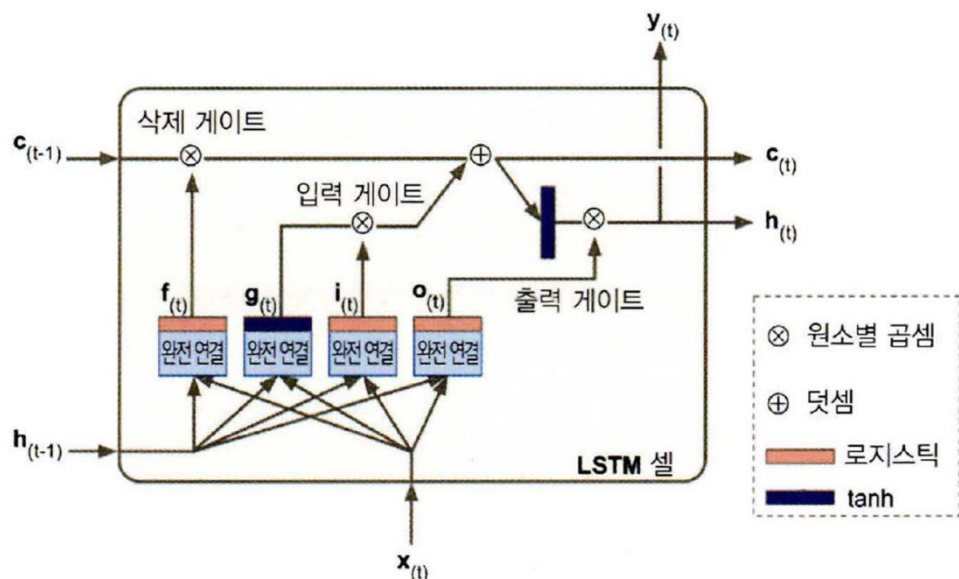
#### Layer Normalization



- RNN에 잘 맞는 정규화는 normalization이라고 함.

### 15.4.2 단기 기억 문제 해결하기

#### 1. LSTM



- 네트워크가 장기기억에서 어떤 것을 저장할지 정하는 것이 핵심.
- 여기서  $h_{(t)}$ 를 단기기억,  $c_{(t)}$ 를 장기기억이라고 봄.

## 2. GRU

LSTM 의 간소화

$$z_{(t)} = \sigma\left(W_{xz}^T \mathbf{x}_{(t)} + W_{hz}^T \mathbf{h}_{(t-1)} + \mathbf{b}_z\right)$$

$$r_{(t)} = \sigma\left(W_{xr}^T \mathbf{x}_{(t)} + W_{hr}^T \mathbf{h}_{(t-1)} + \mathbf{b}_r\right)$$

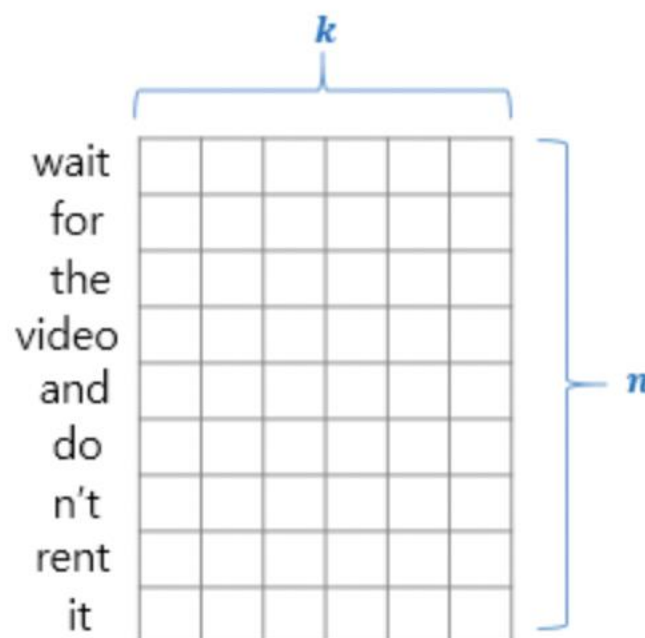
$$\mathbf{g}_{(t)} = \tanh\left(W_{xg}^T \mathbf{x}_{(t)} + W_{hg}^T (r_{(t)} \otimes \mathbf{h}_{(t-1)}) + \mathbf{b}_g\right)$$

$$\mathbf{h}_{(t)} = z_{(t)} \otimes \mathbf{h}_{(t-1)} + (1 - z_{(t)}) \otimes \mathbf{g}_{(t)}$$

- $z_{(t)}$ 가 1을 출력하면 forget gate가 열리고 input gate가 닫히며,
- $z_{(t)}$ 가 0일 경우 반대로 forget gate가 닫히고 input gate가 열린다.

## 3. 1D합성곱 층을 사용해 시퀀스 처리하기

- 'wait for the video and don't rent it'이라는 문장을 토큰화, 패딩, 임베딩 층 거쳐 행렬로 변화



- 1D CNN커널의 너비는 문장 행렬에서의 임베딩 벡터의 차원과 동일하게 설정
- 그렇기 때문에 1D CNN에서는 높이 사이즈만을 명명하여 해당 커널의 사이즈라고 간주

### 15.5 연습문제

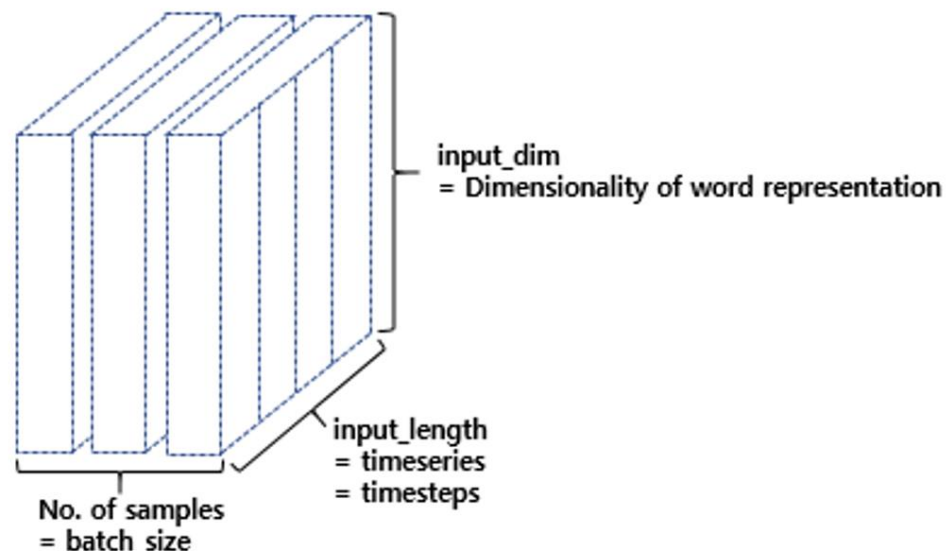
1. 시퀀스-투-시퀀스 RNN을 사용한 애플리케이션에는 어떤 것들이 있나요?

시퀀스-투-벡터 RNN과 벡터-투-시퀀스 RNN은 어떤가요?

- 시퀀스-투-시퀀스 RNN : 날씨 예측, 기계 번역, 비디오 캡션 생성, 스피치 투 텍스트(STT), 음악 생성, 노래의 화음 식별 등
- 시퀀스-투-벡터 RNN : 음악 샘플을 장르로 구분하기, 책 후기에 대한 감성 분석, 뇌에 심은 인공칩에서 읽은 데이터를 기반으로 실어증 환자가 생각하는 단어 예측하기 등
- 벡터-투-시퀀스 RNN : 이미지 캡션 생성, 현재 아티스트를 기반으로 음악 플레이리스트 생성, 일련의 파라미터를 기반으로 한 멜로디 생성, 사진 속에서 보행자 위치 찾기 등

2. RNN 층의 입력은 얼마나 많은 차원을 가지나요?

각 차원이 표현하는 것은 무엇인가요? 출력은 어떤가요?



RNN층의 입력 차원은 3차원

첫 번째: 배치차원 (배치크기)

두 번째: 시간 (타입 스텝의 개수)

세 번째: 타임 스텝마다 입력 (입력 특성의 개수)

RNN층의 출력 차원은 3차원

처음 두 개: 입력과 동일

마지막 : 뉴런 개수

3. 심층 시퀀스-투-시퀀스 RNN을 만든다면 어떤 RNN 층을

return\_sequences=True로 설정해야 하나요? 시퀀스-투-벡터 RNN은 어떤가요?

- 케라스로 심층 시퀀스-투-시퀀스 RNN을 만들려면 모든 RNN 층에 설정해야 함
- 시퀀스-투-벡터 RNN을 만들려면 최상위 RNN층을 제외하고 설정해야 함

```
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)
])
```

4. 일자별 단변량 시계열 데이터를 가지고 다음 7일을 예측하려고 합니다.  
어떤 RNN 구조를 사용해야 하나요?

1. 시퀀스-투-벡터 RNN

- 최상위 RNN 층을 제외하고 모두 return\_sequences=True로 설정한 RNN 층을 쌓음
- 출력 RNN 층에는 뉴런 7개를 사용함
- 해당 모델을 시계열에 랜덤한 윈도우를 적용해 훈련
- 

-

2. 시퀀스-투-시퀀스 RNN

- 모든 RNN 층을 return\_sequences = True로 설정
- 해당 모델을 시계열에 랜덤한 윈도우를 적용해 훈련  
(입력과 타겟에 동일한 길이의 시퀀스 사용)

5. RNN을 훈련할 때 주요 문제는 무엇인가요? 어떻게 이를 처리할 수 있나요?

1. 불안정한 그래디언트

- 더 작은 학습률
- 하이퍼볼릭탄젠트와 같이 수렴하는 활성화 함수
- 그래디언트 클리핑
- 층 정규화
- 타임 스텝마다 드롭아웃

2. 제한적인 단기기억

- LSTM
- GRU

6. LSTM 셀의 구조를 그릴 수 있나요?

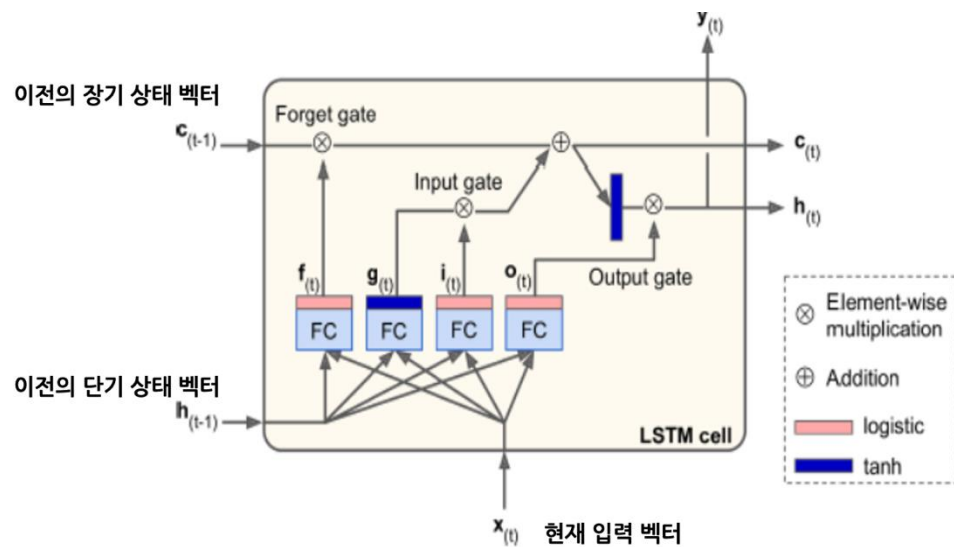


그림 15-9 LSTM 셀

7. 왜 RNN 안에 1D 합성곱 층을 사용해야 하나요?

합성곱 층은 순환 층이 아니므로 불안정한 그래디언트의 영향을 덜 받음.

8. 영상을 분류하기 위해 어떤 신경망 구조를 사용할 수 있나요?

1. 초당 한 프레임을 받아 각 프레임을 합성곱 신경망에 통과시킴
2. CNND의 출력 시퀀스를 시퀀스 투 벡터 RNN에 주입하고 마지막에 소프트맥스 통과시켜 모든 클래스에 대한 확률 구함.



	<p>출처 : <a href="https://wikidocs.net/35476">https://wikidocs.net/35476</a></p>
과제할 당	16.1 이아현 16.2 권기호 16.3 채원석 16.4 이문기 16.5 허주희 16.6 고성호
특이사 항	없음
비고	없음