15장 RNN과 CNN 활용: 순차데이터 처리

감사의 글

자료를 공개한 저자 오렐리앙 제롱과 강의자료를 지원한 한빛아카데미에게 진심어린 감사를 전합니다.

소개

- 시계열 데이터 분석 활용
  - 주식가격 예측
  - 차의 이동경로 예측

- 임의의 길이를 갖는 순차데이터(시퀀스) 처리 가능
  - 문장, 문서, 오디오 샘플 등

- 자동번역, 받아쓰기 등 자연어처리(NLP) 분야에 매우 유용함.

주요 내용

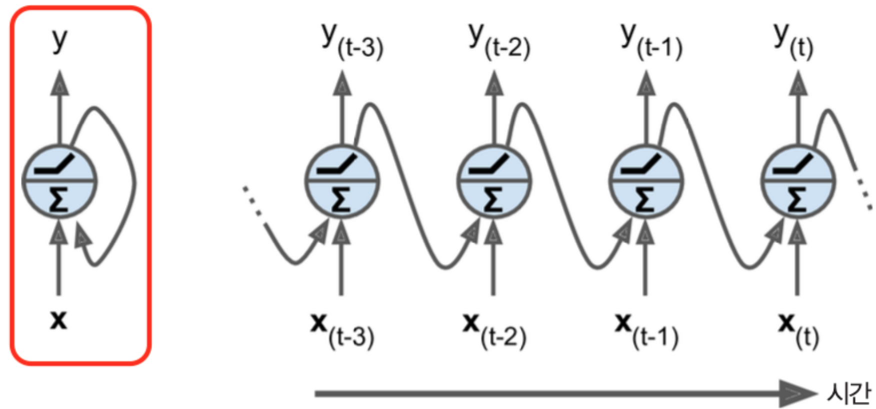
- 순환신경망(RNN) 기본개념 및 역전파 네트워크 훈련 방법 소개
- 시계열 예측 수행
- RNN의 주요 난제 설명
    - 불안정한 그레이디언트 다루기: 순환 드롭아웃 및 순환층 정규화
    - 매우 제한적인 단기기억 문제 다루기: LSTM과 GRU 셀 활용

순환뉴런과 순환층

- 지금까지 살펴본 신경망 모델은 아래층에서 위층으로 학습결과를 전달하는 **전방향전달** (feedforward) 방식 사용
- 순환신경망(recurrent neural network, RNN)도 비슷하게 작동
- 차이점: 후방향으로 전달하여 학습결과를 순환시키는 기능 활용

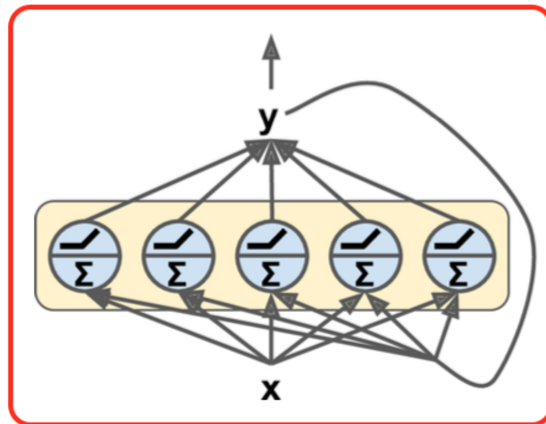
순환뉴런과 타입스텝

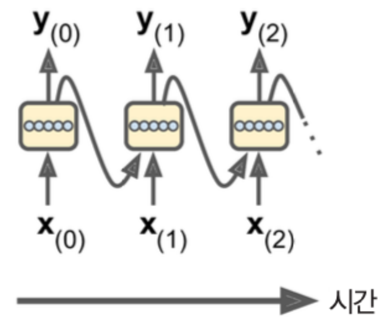
- 순환뉴런: 입력을 받아 출력을 만든 후 자신에게도 출력결과를 전달



순환층

- 순환뉴런으로 구성된 층





하나의 샘플에 대한 순환층의 출력

- 가중치 행렬
  - $\mathbf{W}_x$: 현재 타임스텝의 입력에 대한 연결가중치 행렬
  - $\mathbf{W}_y$: 이전 타입스텝의 출력에 대한 연결가중치 행렬
  - 타임스텝에 의존하지 않음.

$$\mathbf{y}_{(t)} = \phi\left(\mathbf{W}_x{}^T \mathbf{x}_{(t)} + \mathbf{W}_y{}^T \mathbf{y}_{(t-1)} + \mathbf{b}\right)$$

배치에 대한 순환층의 출력

$$\mathbf{Y}_{(t)} = \phi\left(\mathbf{X}_{(t)}\mathbf{W}_x + \mathbf{Y}_{(t-1)}\mathbf{W}_y + \mathbf{b}\right)$$

$$= \phi\left(\left[\mathbf{X}_{(t)} \ \ \mathbf{Y}_{(t-1)}\right]\mathbf{W} + \mathbf{b}\right) \quad \text{여기에서} \quad \mathbf{W} = \begin{bmatrix} \mathbf{W}_x \\ \mathbf{W}_y \end{bmatrix}$$

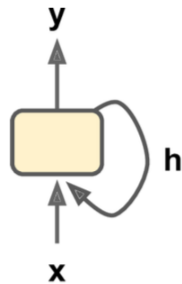
(메모리) 셀

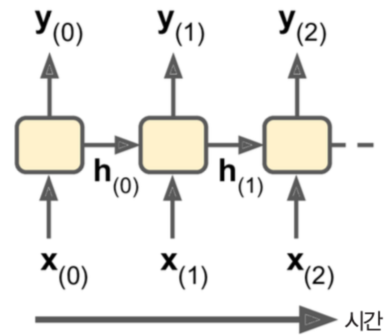
- 메모리 셀: 타임스텝에 걸쳐서 특정 상태를 보존하는 신경망의 구성요소

- 예제
    - 하나의 순환뉴런 또는 순환뉴런층
        - 10 스텝 정도의 짧은 패턴을 학습하는 기본적인 셀
    - LSTM 셀, GRU 셀
        - 장단기 기억 셀

- 상태 $\mathbf{h}_{(t)}$: 타임스텝 t 에서의 셀의 상태

$$\mathbf{h}_{(t)} = f(\mathbf{h}_{(t-1)}, \mathbf{x}_{(t)})$$





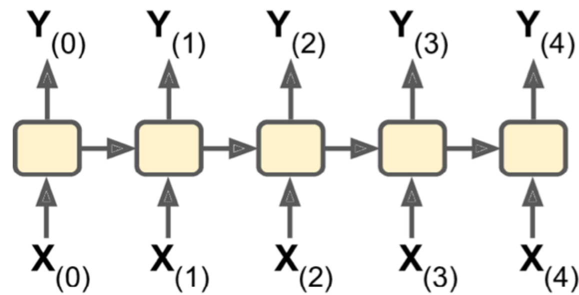
입력/출력 순차데이터

- 목적에 따라 입력/출력 데이터의 형식을 벡터 또는 순차데이터(시퀀스)로 지정할 수 있음.

seq-to-seq

- 입력: 순차데이터
- 출력: 순차데이터

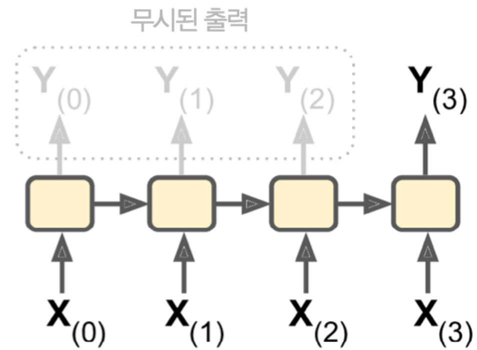
- 예제: 주식가격 시계열 데이터 활용에 유용
    - 최근 N일 동안의 주식가격을 입력하면 각 입력값보다 하루 이후의 가격을 출력



seq-to-vector

- 입력: 순차데이터
- 출력: 벡터

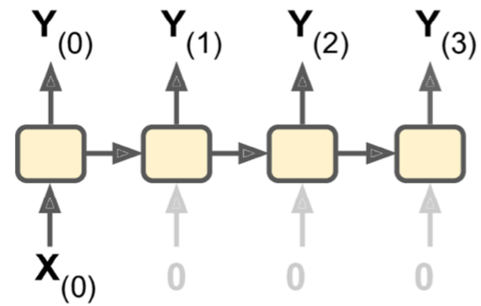
- 예제: 영화리뷰
  - 영화리뷰에 있는 연속된 단어를 주입하면 감성점수 출력(-1에서 1사이)



vector-to-seq

- 입력: 벡터
- 출력: 순차데이터

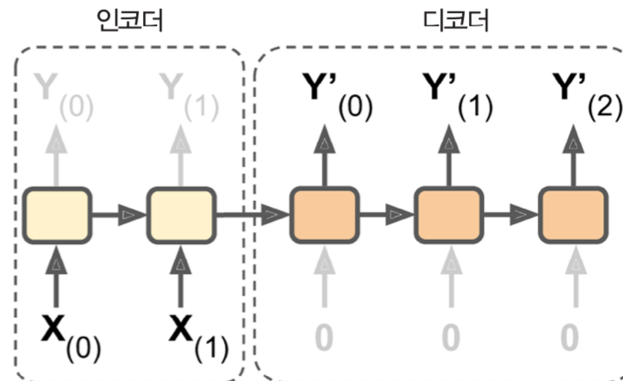
- 예제: 이미지를 입력하면 이미지 캡션 출력



encoder-decoder

- 인코더와 디코더의 조합을 사용하는 이중단계모델
  - 인코더: seq-to-vector 신경망
  - 디코더: vector-to-seq 신경망

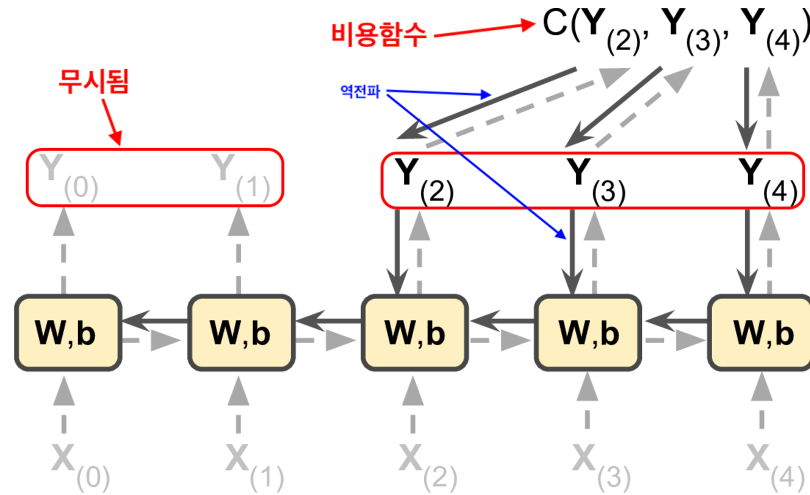
- 예제: 언어 번역
  - seq-to-seq 모델보다 성능 좋음.
  - 이유: "한국말은 끝까지 들어야 한다" 와 동일한 이유이며, 하나의 입력문장이 끝날 때까지 기다린 후 그 결과를 번역하는 과정에 적합함. (16장 참조)



RNN 훈련

- BPTT(backpropagation through time) 전략: 타임스텝을 통과하는 역전파 전략

- 가중치 $\mathbf{W}$와 편향 $\mathbf{b}$가 타임스텝에 의존하지 않기에 역전파가 모든 관련된 타임스텝에 대해 일관되게 작동함.



시계열 예측

단변량 시계열

- 타임스텝마다 하나의 값을 가지는 순차데이터
- 예제
  - 웹사이트의 시간당 접속자 수
  - 도시의 날짜별 온도

다변량 시계열

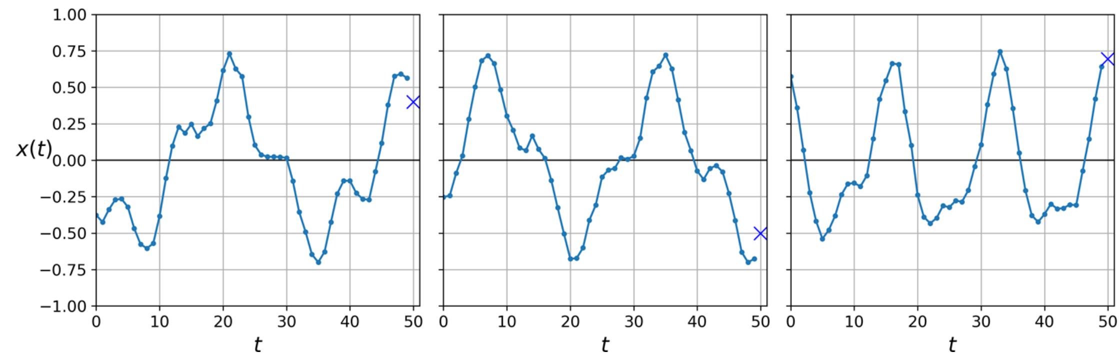
- 타임스텝마다 여러 개의 값을 가지는 순차데이터
- 예제
  - 회사의 수입, 부채 등 여러 지표를 사용한 기업의 분기별 재정안정성

활용법 두 가지

- 예측: 미래의 값 추정하기
- 결측치 대체: 과거 데이터에서 누락된 값 추정하기

예제

- 아래 그림: 3개의 단변량 시계열
  - 타임스텝: 50개
  - 목표: x 로 표시된 값 예측하기



- 위와 같은 시계열 데이터 10,000개를 훈련 데이터로 생성하여 순환신경망(RNN) 학습법을 살펴보고자 함.

훈련 시계열 데이터 준비

- 아래 함수를 이용하여 시계열 데이터 생성

- 아래 함수의 반환값의 모양: (배치 크기, 타임스텝, 1)
    - 마지막 값이 1인 이유: 단변량 시계열 데이터를 생성하기 때문

```
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)
    series = 0.5 * np.sin((time - offsets1) * (freq1 * 10 + 10))  #   파도 곡선 1
    series += 0.2 * np.sin((time - offsets2) * (freq2 * 20 + 20)) # + 파도 속선 2
    series += 0.1 * (np.random.rand(batch_size, n_steps) - 0.5)   # + 잡음
    return series[..., np.newaxis].astype(np.float32)
```

- 길이가 50인 1만 개의 시계열 데이터를 생성하여 각각 7천 개 2천 개, 천 개 크기의 훈련, 검증, 테스트 세트 생성
  - 예를 들어 `X_train`과 `t_train`의 모양은 각각 `(7000, 50, 1)`과 `(7000, 1)` 임.

```
n_steps = 50
series = generate_time_series(10000, n_steps + 1)
X_train, y_train = series[:7000, :n_steps], series[:7000, -1]
X_valid, y_valid = series[7000:9000, :n_steps], series[7000:9000, -1]
X_test, y_test = series[9000:, :n_steps], series[9000:, -1]
```

RNN 성능 비교

- RNN의 성능을 타 방식과 비교하기 위해 두 가지 방식의 성능을 먼저 계산해 보고자 함.
    - 순진한 예측
    - 선형회귀 예측

순진한 예측

- 시계열의 마지막 값을 예측값으로 사용

```
y_pred = X_valid[:, -1]
```

- 평균제곱오차(MSE)가 0.02 정도로 매우 우수한 것처럼 보임

선형회귀 예측

- 완전연결 네트워크를 사용하여 선형회귀 예측 실행
    - 입력값을 1차원 배열로 변환해서 사용함.

```
model = keras.models.Sequential([
    keras.layers.Flatten(input_shape=[50, 1]),
    keras.layers.Dense(1)
    ])

model.compile(loss="mse", optimizer="adam")
```

- 검증 세트에 대한 평균제곱오차(MSE)가 0.0041 정도로 훨씬 우수함

가장 단순한 RNN 모델 예측 성능

- 한 개의 순환뉴런을 갖는 `SimpleRNN` 순환층 한 개로 구성된 RNN 모델
    - 입력값의 크기, 즉, 타임스텝의 크기가 임의이어도 되기에 입력값의 모양의 첫째 항목이 `None`으로 지정됨.

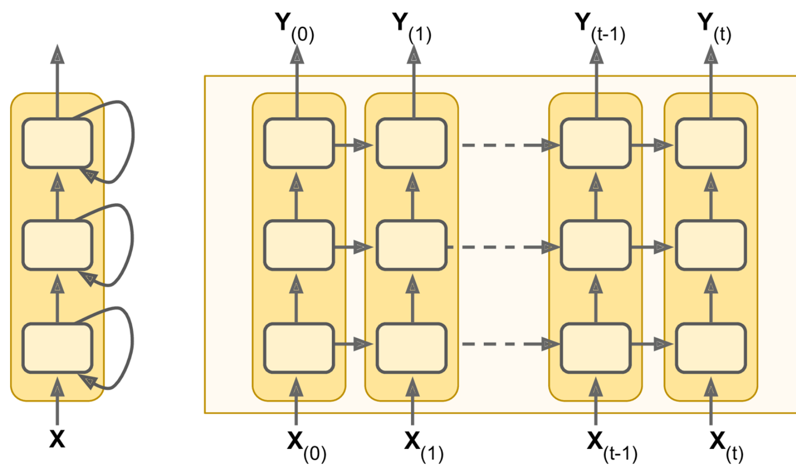
```
model = keras.models.Sequential([
    keras.layers.SimpleRNN(1, input_shape=[None, 1])
])

optimizer = keras.optimizers.Adam(lr=0.005)
model.compile(loss="mse", optimizer=optimizer)
```

- `SimpleRNN` 층의 특징
    - 활성화 함수: `tanh`
    - 반환값: 최종 출력만 리턴하는 게 기본. 모든 타임스텝에서 출력하도록 하려면 `return_sequences=True`로 설정.
    - 모든 시계열 샘플에 데이터를 동시에 처리.

- 검증 세트에 대한 평균제곱오차(MSE)가 0.0109 정도로 선형회귀 예측보다 좋지 않음.
- 하지만 선형회귀 모델은 51개의 파라미터를 사용하는 반면에 위 RNN 모델은 단 세 개의 파라미터만 사용함.
- 선형회귀 모델의 파라미터
    - 50 개의 입력값에 대한 가중치 + 편향
- 위 RNN 모델의 파라미터
    - 순환층에 사용된 순환뉴런의 수를 지정하는 파라미터 + 입력값에 대한 1개의 가중치 + 편향

심층 RNN 모델 예측 성능

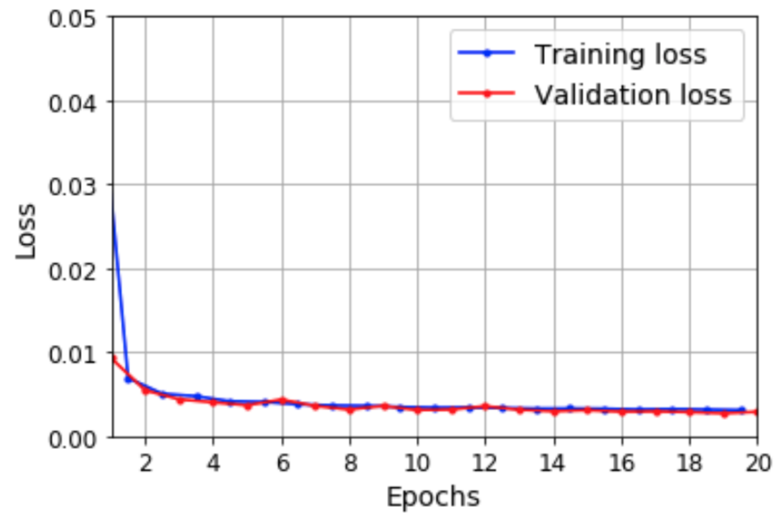
- 셀을 여러 층으로 쌓은 RNN



- 마지막 층을 제외한 순환층은 `return_sequences=True`을 반드시 사용해야 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)
])

model.compile(loss="mse", optimizer="adam")
```

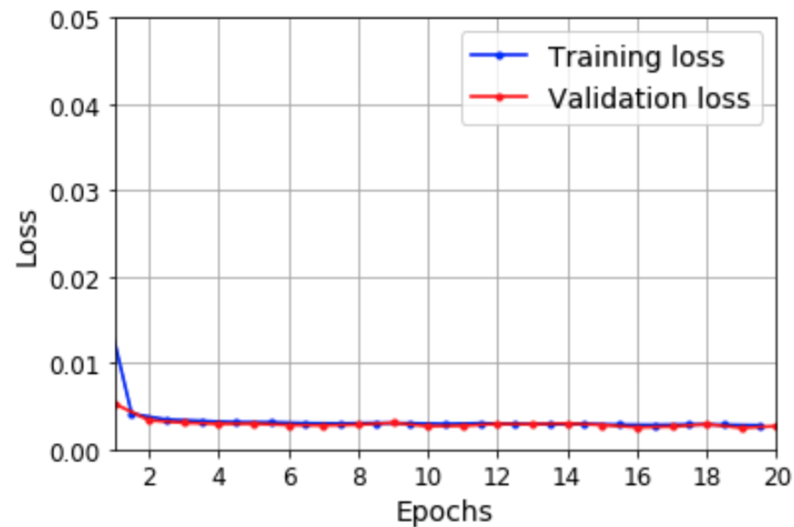
- 검증 세트에 대한 평균제곱오차(MSE)가 0.0029 정도로 선형회귀 예측보다 두 배 정도 좋아졌음.



- 하지만 위 모델은 아래 두 가지 관점에서 개선될 수 있음.
  - 첫째, 단변량 시계열 데이터를 사용하기에 마지막 순환층이 하나의 순환뉴런으로 구성되어야 함. 이는 전달되는 (은닉)상태가 하나의 숫자에 불과함을 의미하며 많은 정보를 담고 있지 않음. 따라서 분명이 이전 층의 (은닉) 정보가 많이 활용될 것임.
  - 둘째, `SimpleRNN` 층의 활성화 함수가 `tanh`로 고정이어서 다른 활성화 함수를 사용 불가.

- 위 문제를 해결하기 위해 마지막 출력 층을 밀집층으로 바꾸고, 활성화 함수도 원하는 대로 설정할 수 있음.
  - 성능은 비슷하고 훈련속도는 빨라짐.

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

model.compile(loss="mse", optimizer="adam")
```



여러 타임스텝 미래 예측하기

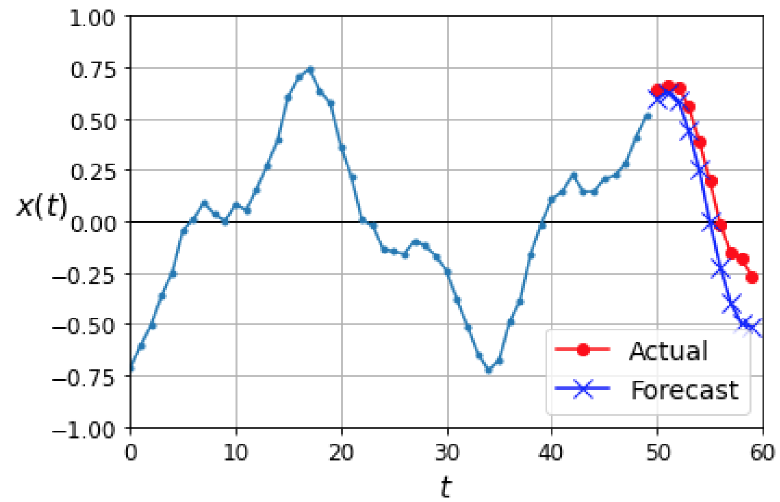
- 지금까지 한 타임스텝 미래만 예측하였음.
- 여러 타임스텝 미래를 예측하려면 단순히 타깃(레이블) 값만 지정된 타임스텝 앞의 것으로 설정하면 됨.
- 하지만 다음 타임스텝 여러 개의 값을 예측하려면? 다음 두 가지 방식 활용 가능
    - 한 타임스텝 예측을 원하는 만큼 반복하기
    - 여러 타임스텝의 값을 동시에 예측하기

한 타임스텝 예측 반복

- 이미 훈련된 모델을 이용하여 한 타임스텝을 예측
- 예측 결과를 추가한 후 동일한 과정 반복

```
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)
```

- 미래 예측이 반복될 수록 정확도 떨어지지만 경우에 따라 잘 작동할 수 있음.



여러 타임스텝 값 동시 예측 방법 1

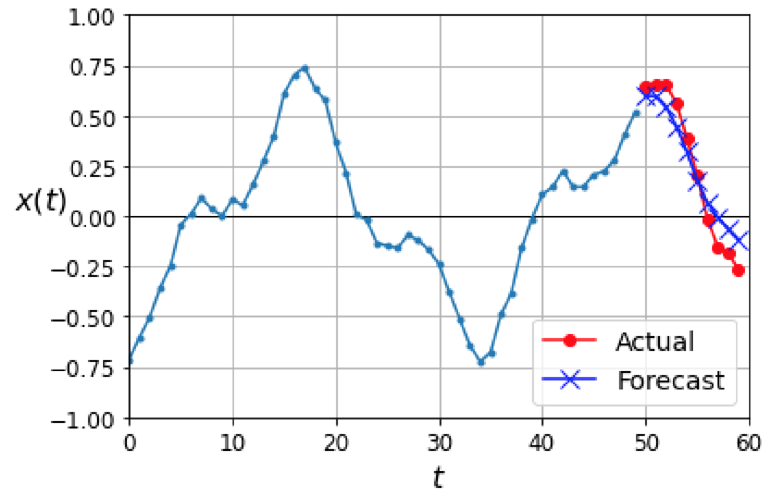
- 마지막 예측 과정에서 미래 타임스텝 10개의 값을 예측하는 것으로 변경
- 여전히 seq-to-vector 모델 사용.

```
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]
```

- 이후 모델 활용은 동일함.

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

model.compile(loss="mse", optimizer="adam")
```

- 성능 비교(MSE)
  - 순진한 예측: 약 0.227
  - 선형회귀 예측: 약 0.0188
  - 심층 RNN 예측: 약 0.0094



여러 타임스텝 값 동시 예측 방법 2

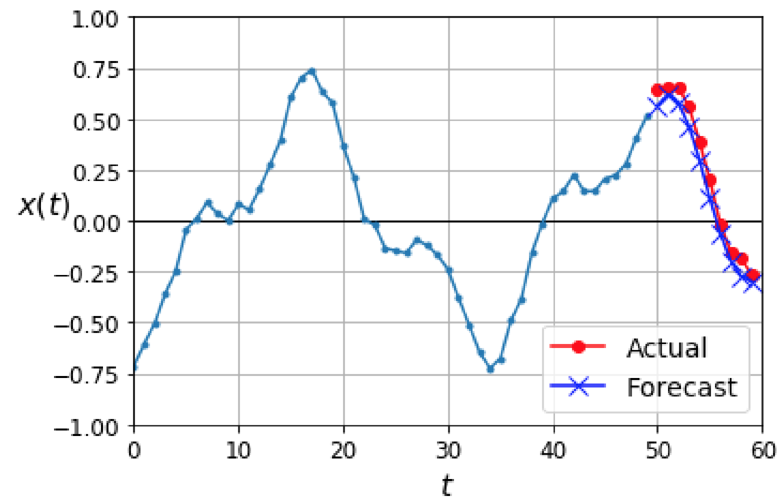
- 전체 타임스텝에서 미래 타임스텝 10개의 값을 예측하도록 설정

```python
n_steps = 50
series = generate_time_series(10000, n_steps + 10)
X_train = series[:7000, :n_steps]
X_valid = series[7000:9000, :n_steps]
X_test = series[9000:, :n_steps]

# 각 타임스텝에 대한 타깃값을 10개씩 설정
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:]
```

- 모델 설정은 seq-to-seq 모델 형식으로 변경
    - 모든 순환층에서 `return_sequences=True` 사용
    - 모든 타입스텝에서의 출력값을 처리하기 위해 출력층에서 `TimeDistributed` 클래스 활용
        - 하지만 `keras.layers.Dense(10)`라고 써도 됨.
    - MSE는 그냥 마지막 10번째 값만 대상으로 측정하도록 하려면 새로 정의해야 함.

```python
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))
])

def last_time_step_mse(Y_true, Y_pred):
    return keras.metrics.mean_squared_error(Y_true[:, -1], Y_pred[:, -1])

model.compile(loss="mse", optimizer=keras.optimizers.Adam(lr=0.01), metrics=[last
_time_step_mse])
```

- 성능(MSE): 약 0.0065



여러 타임스텝 값 동시 예측 방법 3

- 방법2을 방법1과 혼합하기
- 10개씩 여러 번 반복 예측하면 단기 예측에 잘 활용할 수 있음. (16장 참조)

MC Dropout 활용

- 각 메모리셀에서 MC Dropout 층 추가
    - 입력값과 은닉 상태 일부를 드롭아웃시키기
- 예측을 여러번 실행 후 평균값과 표준편차 계산하여 **오차 막대(error bars)** 구하여 활용

긴 시퀀스 다루기

- 심층 RNN을 훈련시킬 때 심층신경망 훈련 과정에서 발생하는 문제가 일반적으로 동일하게 발생함.

- 문제 1: 그레이디언트 소실과 폭주, 긴 훈련시간 또는 불안정한 훈련

- 문제 2: 긴 시퀀스를 처리할 때 입력의 앞 부분을 조금씩 잊혀지기

불안정한 그레이디언트 문제 다루기

- 좋은 가중치 초기화, 빠른 옵티마이저, 드롭아웃 등 활용
- 각 순환층 이전에 사용하는 배치정규화는 별 도움되지 않음.
- 반면에 **층 정규화**(layer normalization)는 어느정도 도움 됨.

층 정규화

- 2016년에 소개됨
- 배치 차원이 아니라 특성 차원의 정규화 실행
- 현재 케라스에서 지원되지 않지만 직접 구현 가능

- 예를 들어, 선형 연산 후와 활성화 함수 이전에 층 정규화를 수행하려면 아래 클래스 활용 가능

```
class LNSimpleRNNCell(keras.layers.Layer):
    def __init__(self, units, activation="tanh", **kwargs):
        super().__init__(**kwargs)
        self.state_size = units
        self.output_size = units
        self.simple_rnn_cell = keras.layers.SimpleRNNCell(units,
                                                          activation=None)
        self.layer_norm = LayerNormalization()
        self.activation = keras.activations.get(activation)
    def get_initial_state(self, inputs=None, batch_size=None, dtype=None):
        if inputs is not None:
            batch_size = tf.shape(inputs)[0]
            dtype = inputs.dtype
        return [tf.zeros([batch_size, self.state_size], dtype=dtype)]
    def call(self, inputs, states):
        outputs, new_states = self.simple_rnn_cell(inputs, states)
        norm_outputs = self.activation(self.layer_norm(outputs))
        return norm_outputs, [norm_outputs]
```

- 위 모델을 활용한 순환모델 생성은 기존과 동일함.

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

model.compile(loss="mse", optimizer="adam", metrics=[last_time_step_mse])
```

단기기억 문제 해결하기

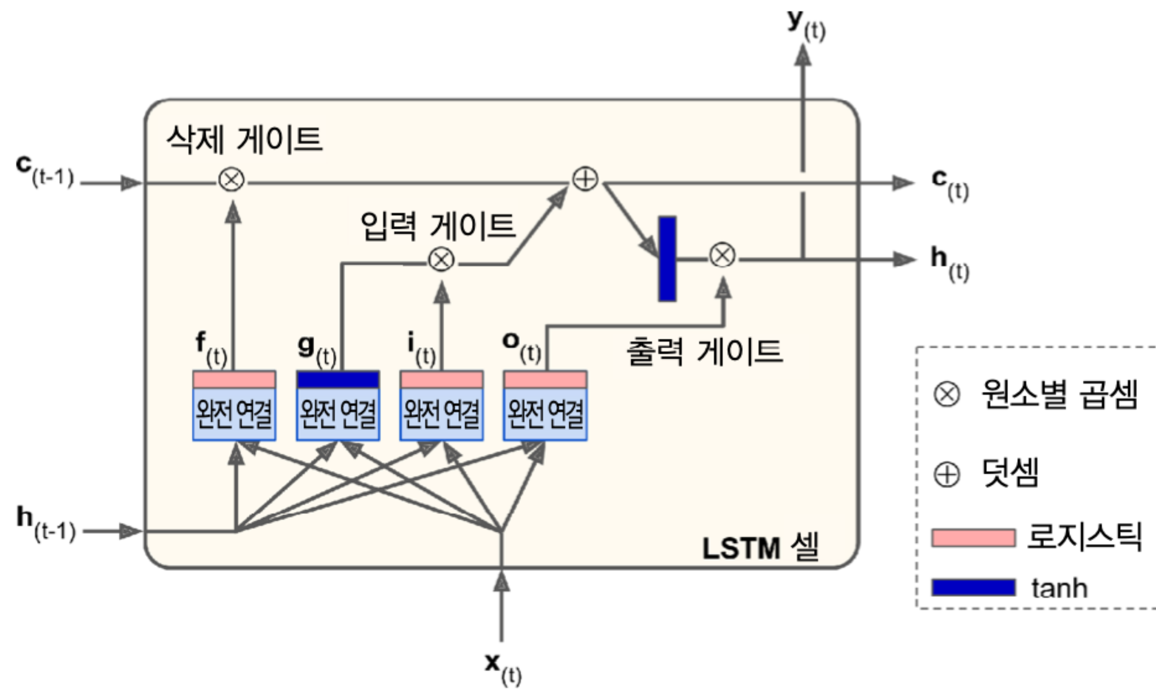
- 장기 메모리를 가진 셀 활용

LSTM 셀

- 1997년에 소개됨
- 활용법: `SimpleRNN` 층 대신 `LSTM` 층 사용하면 됨.

```
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))
    ])

model.compile(loss="mse", optimizer="adam", metrics=[last_time_step_mse])
```

- 작동방식

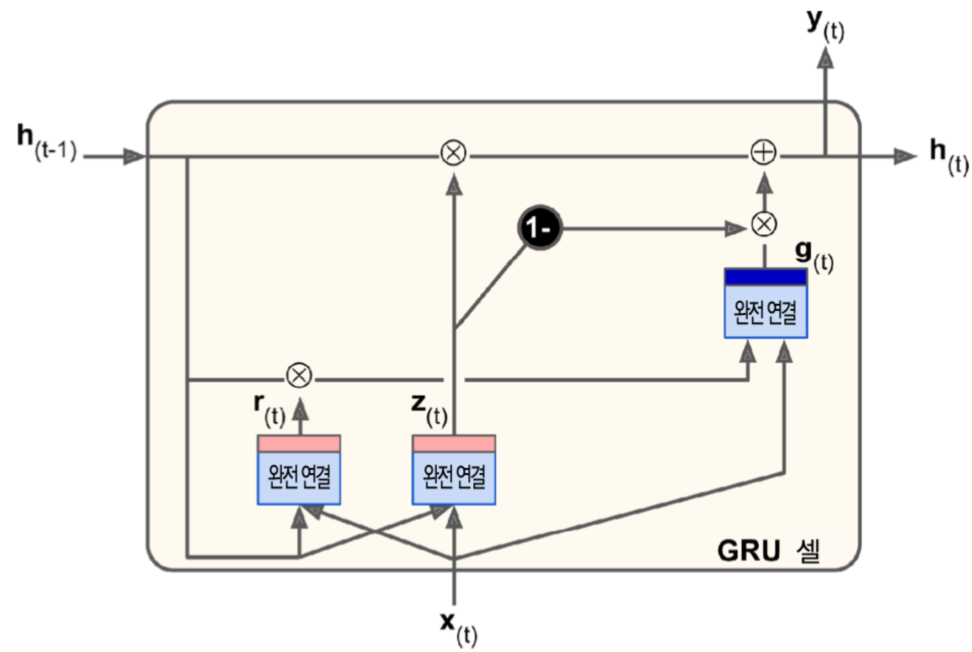


GRU 셀

- 2014년에 소개됨
- LSTM 셀의 간소화 버전. 성능 유사

```
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))
    ])

model.compile(loss="mse", optimizer="adam", metrics=[last_time_step_mse])
```

- 작동방식



GRU 셀과 합성곱 층 혼용

- 1D 합성곱 층으로 입력값을 처리한 후에 GRU 셀 활용
- 2 이상의 보폭을 사용하여 1D 합성곱 층으로 시퀀스 길이를 줄이면서 핵심 정보만 GRU 셀에 전달.

- 예제: `kernel_size=4, strides=2` 활용

```
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
```

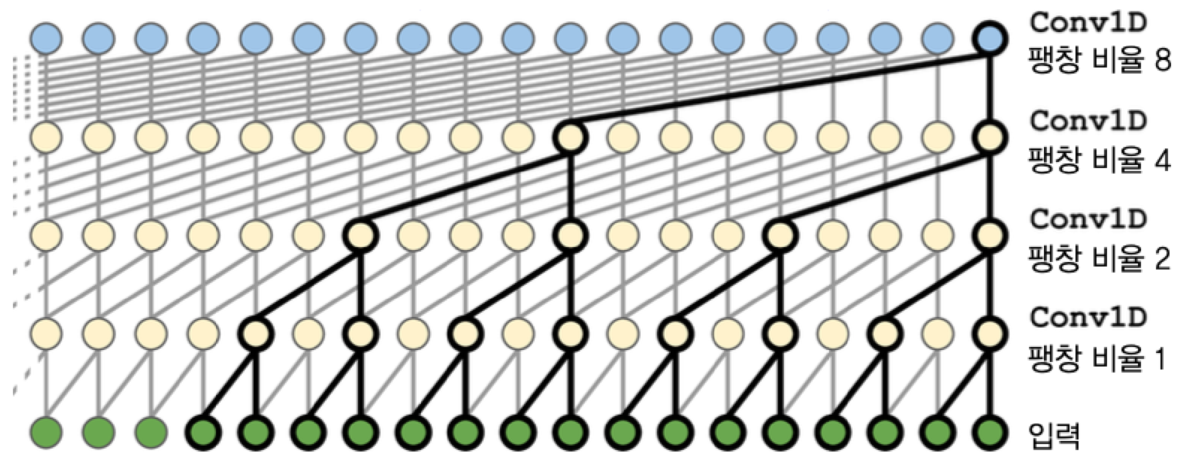
- 가장 좋은 성능 발휘

```
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))
    ])

model.compile(loss="mse", optimizer="adam", metrics=[last_time_step_mse])
```

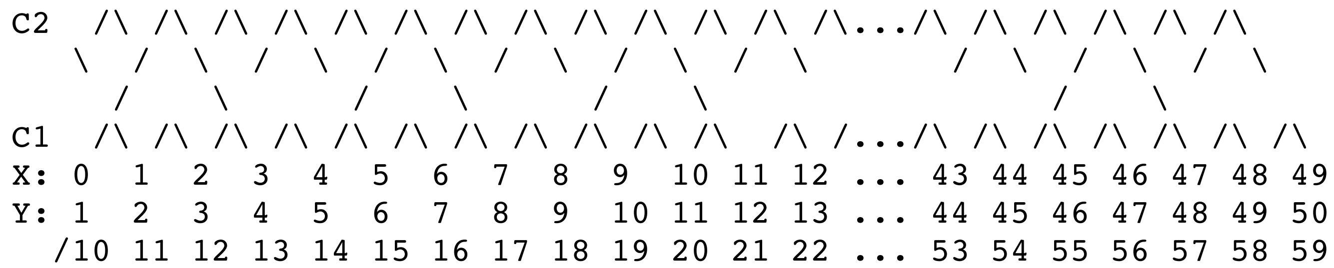
WAVENET

- 2016년에 소개됨
- 1D 합성곱 층을 쌓아 올림
- 네트워크 층마다 각 뉴런의 입력이 떨어져 있는 간격을 지정하는 **팽창 비율**(dilation rate)가 두 배로 커짐.

- 작동방식



- 예제



```python
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))
model.compile(loss="mse", optimizer="adam", metrics=[last_time_step_mse])
```

결론

- 마지막 두 모델이 가장 좋은 성능 제공
- WAVENET의 경우 텍스트-투-스피치 작업, 음악 오디오 샘플 생성 등 많은 오디오 문제에서 최상의 성능 발휘
    - 오디오 1초에 수만 개의 타임스텝이 있는 것을 고려할 때 대단한 성능임.
- LSTM 셀, GRU 셀보다 훨씬 긴 시퀀스 다룰 수 있음.