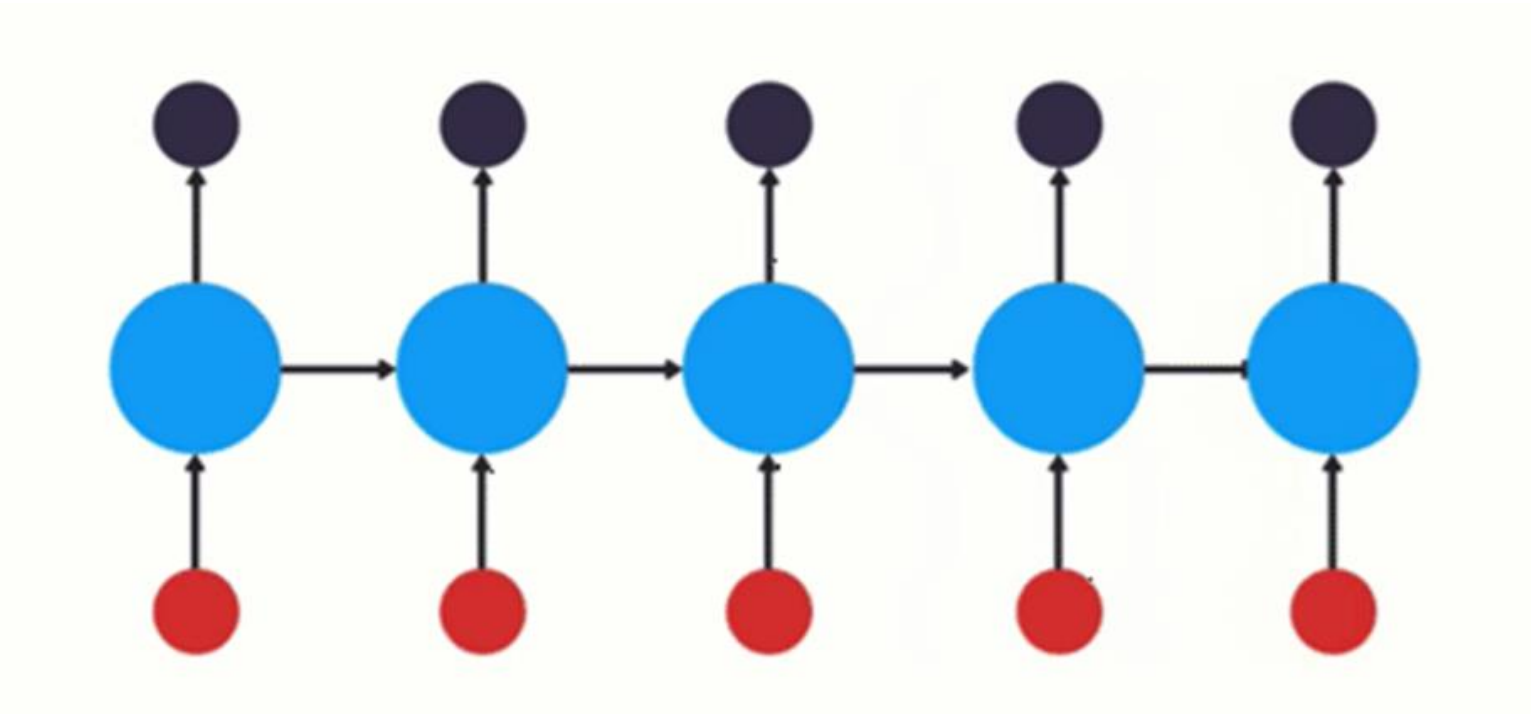
RNN을 사용해 시퀀스 처리하기



박 경 규

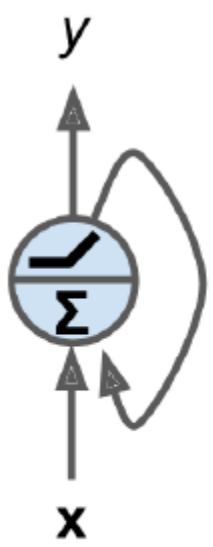
본 챕터 전체 내용의 출처는 도서 "핸즈온 머신러닝 2판(Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd Edition)"입니다.

순환 신경망

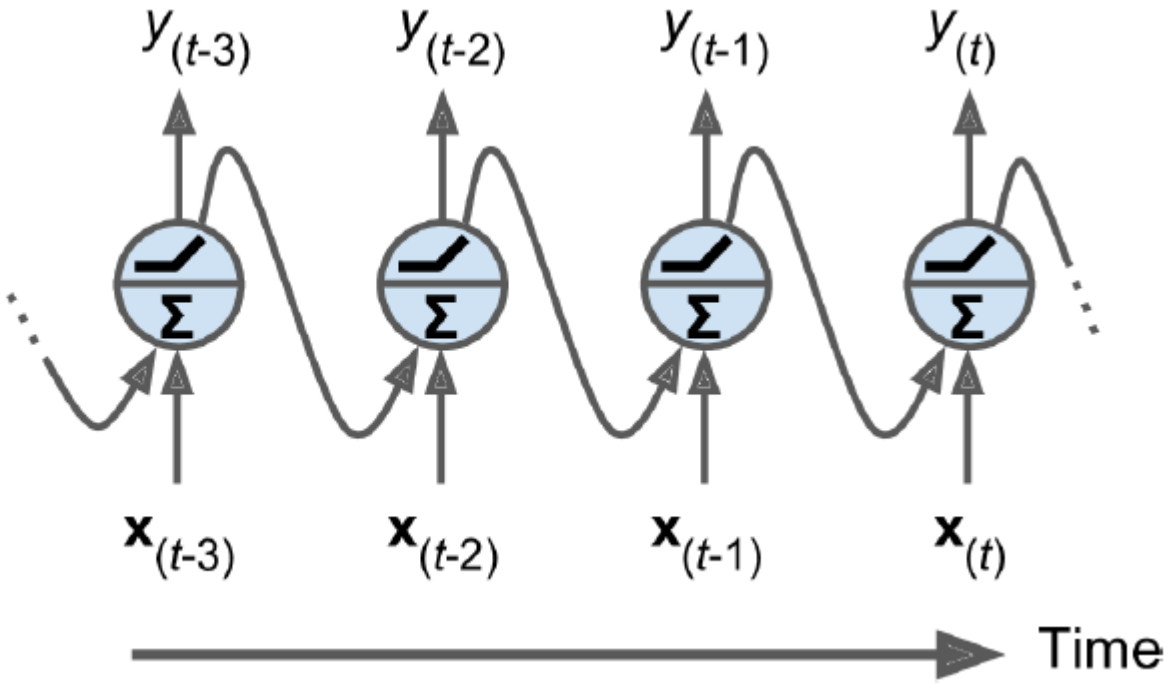
순환신경망은 고정 길이 입력이 아닌 임의 길이를 가진 시퀀스를 다룰 수 있습니다.

- 순환신경망은 시계열 데이터를 분석해서 미래값을 예측하고 문장, 오디오를 입력으로 받아 자동번역, 자연어처리에 유용합니다.
- 순환신경망은 피드포워드 신경망과 비슷하지만 뒤쪽으로 순환하는 연결도 있다는 점이 다릅니다.

■ 순환뉴런

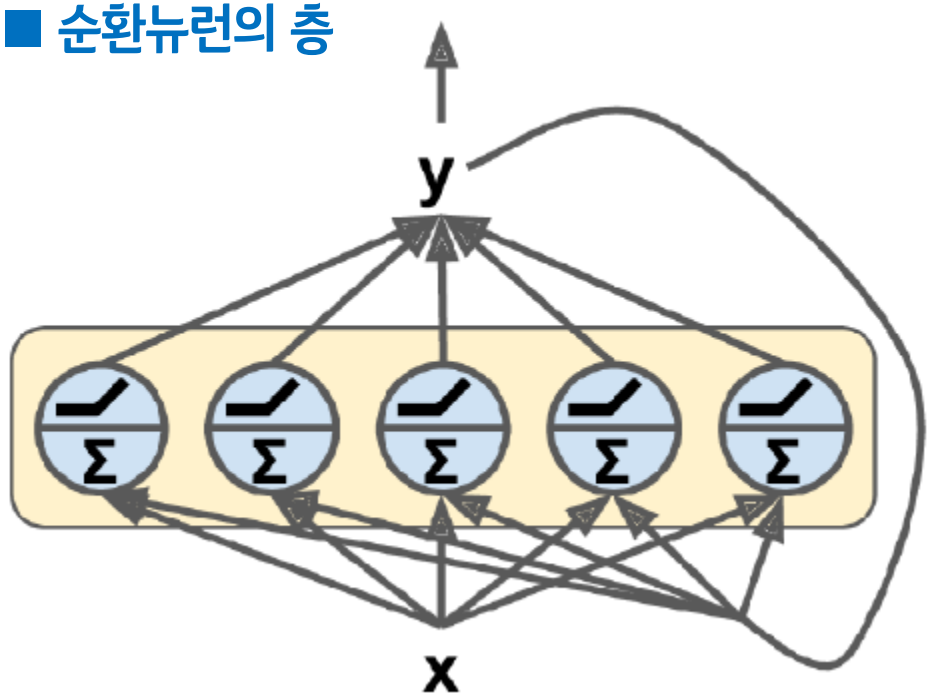


■ 순환뉴런을 타임 스텝으로 펼친 모습

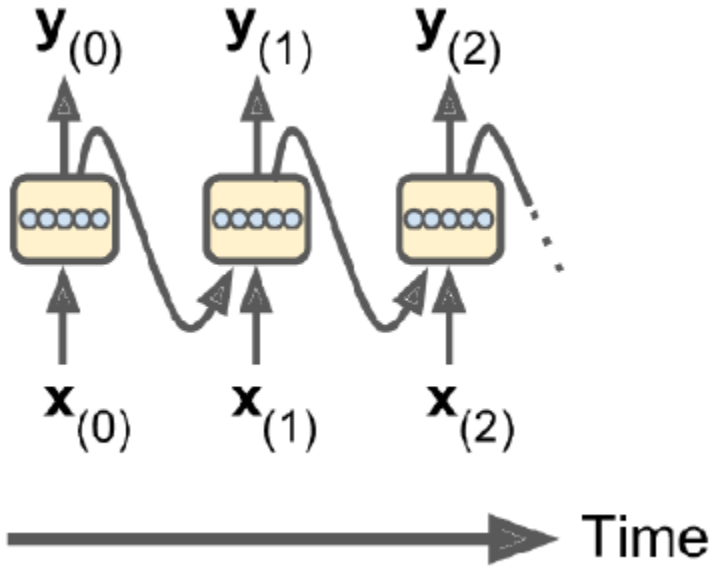


순환 신경망

■ 순환뉴런의 층



■ 타임 스텝으로 펼친 모습



■ 하나의 샘플에 대한 순환 층의 출력

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

■ 미니배치에 있는 전체 샘플에 대한 순환 뉴런 층의 출력

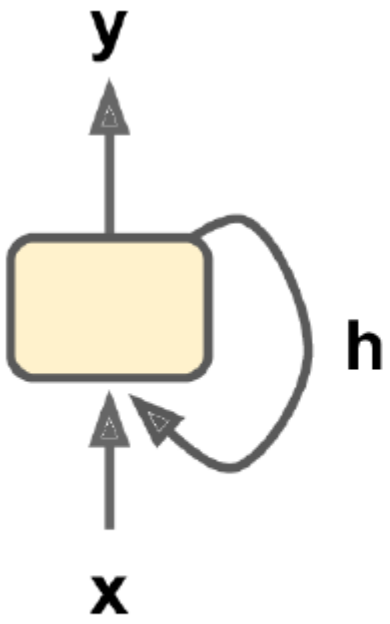
$$\mathbf{Y}_{(t)} = \phi\left(\mathbf{X}_{(t)}\mathbf{W}_x + \mathbf{Y}_{(t-1)}\mathbf{W}_y + \mathbf{b}\right)$$

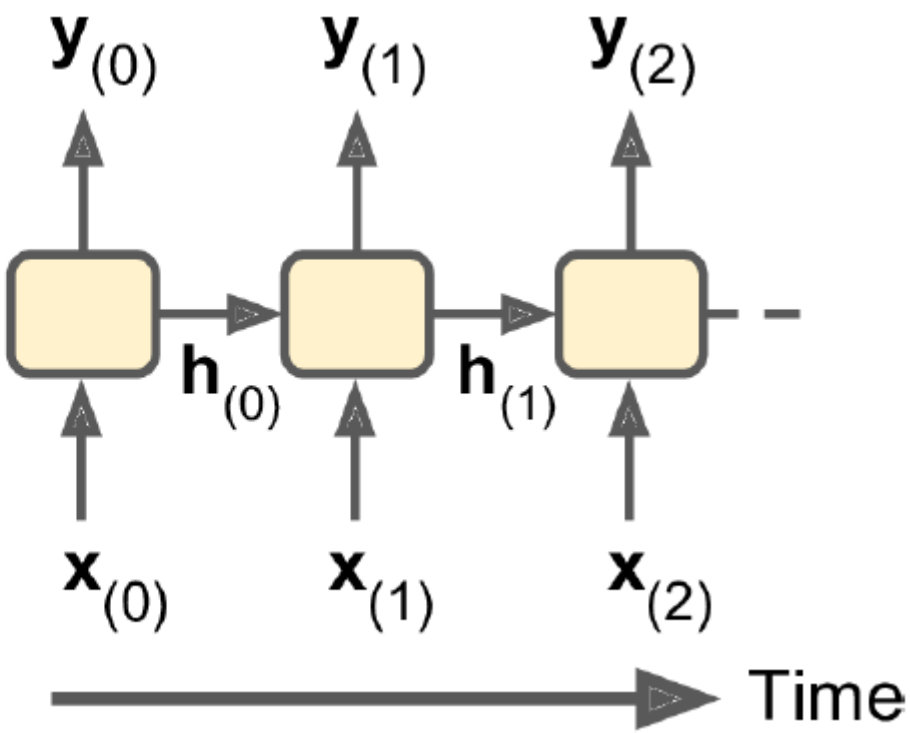
$$= \phi\left(\left[\mathbf{X}_{(t)} \quad \mathbf{Y}_{(t-1)}\right]\mathbf{W} + \mathbf{b}\right) \text{ with } \mathbf{W} = \begin{bmatrix}\mathbf{W}_x \\ \mathbf{W}_y\end{bmatrix}$$

메모리 셀

타임 스텝 t에서 순환 뉴런의 출력은 이전 타임 스텝의 모든 입력에 대한 함수이고, 타임스텝에 걸쳐서 상태를 보존하는 신경망의 구성 요소를 메모리 셀(혹은 간단히 셀) 이라고 합니다.

■ 셀의 히든 상태와 출력은 다를 수 있습니다.

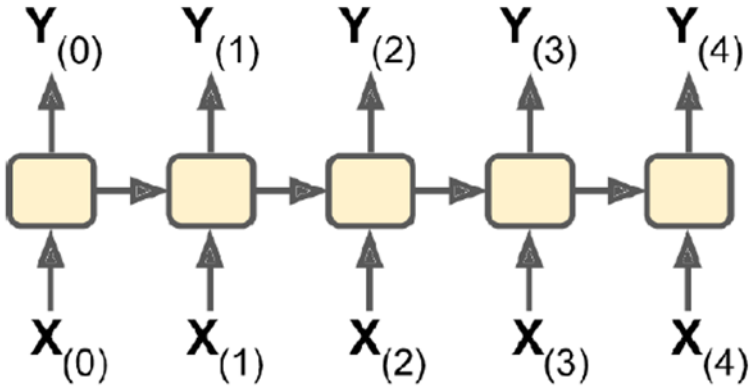




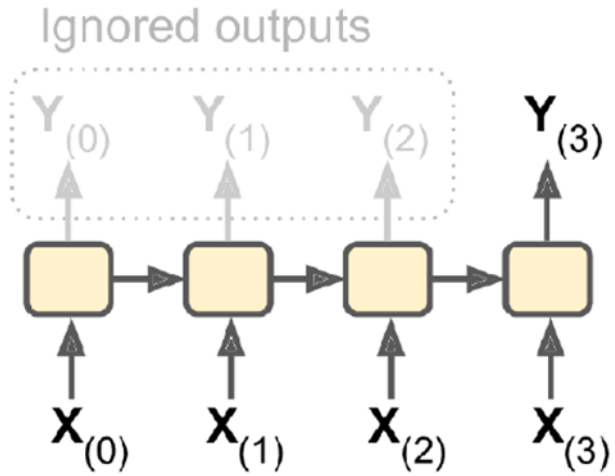
$$\mathbf{h}_{(t)} = f(\mathbf{h}_{(t-1)}, \mathbf{x}_{(t)})$$

입력과 출력 시퀀스

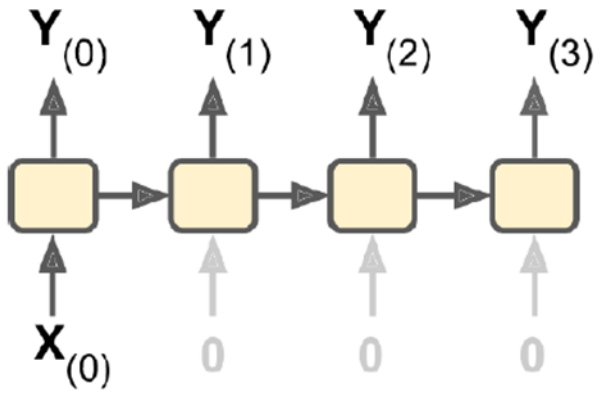
Seq-to-Seq



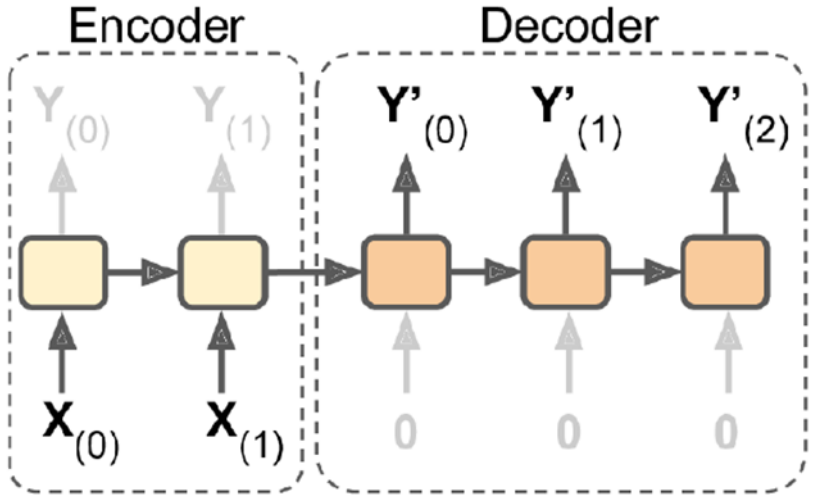
Seq-to-Vector



Vector-to-Seq

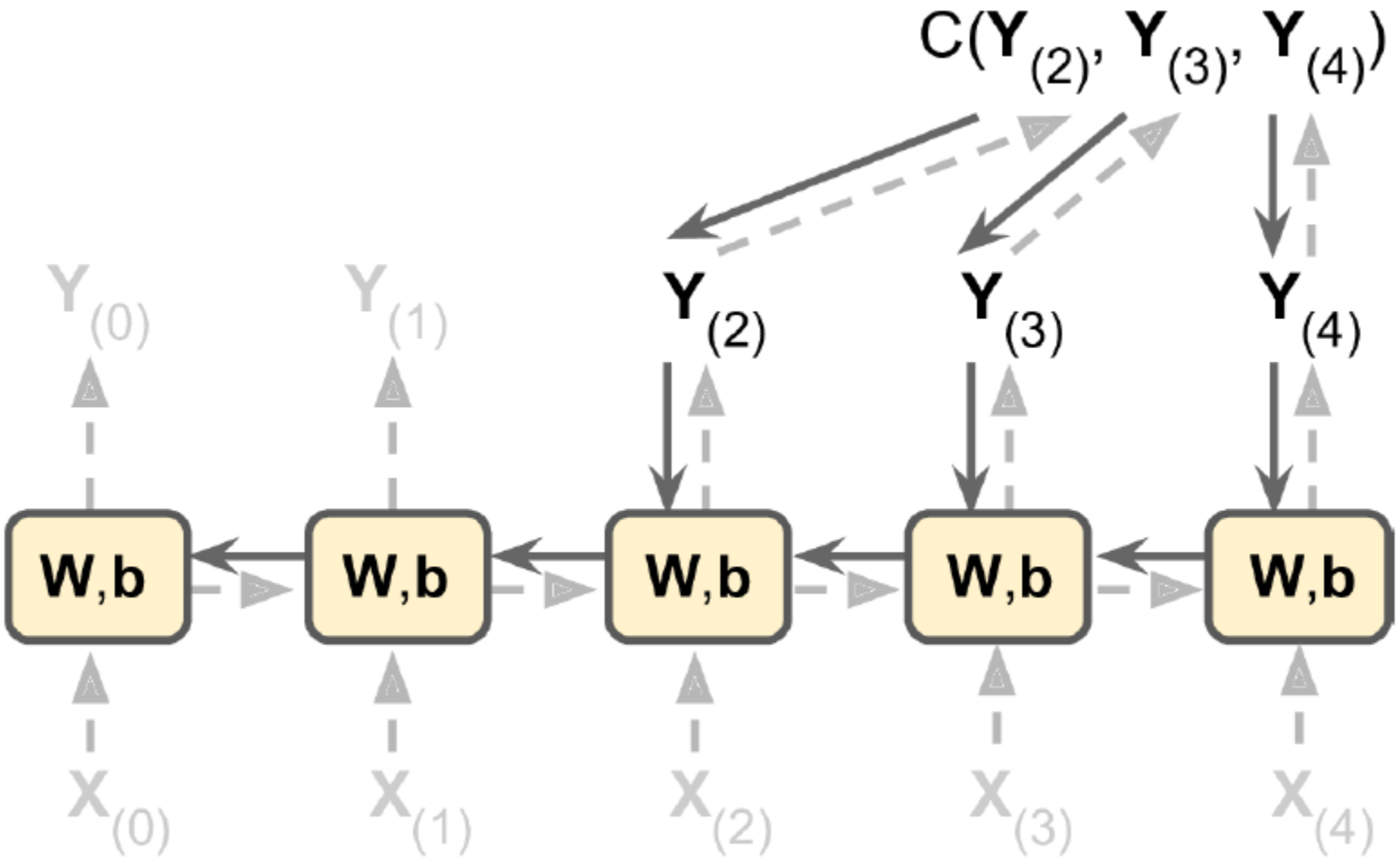


Encoder-Decoder



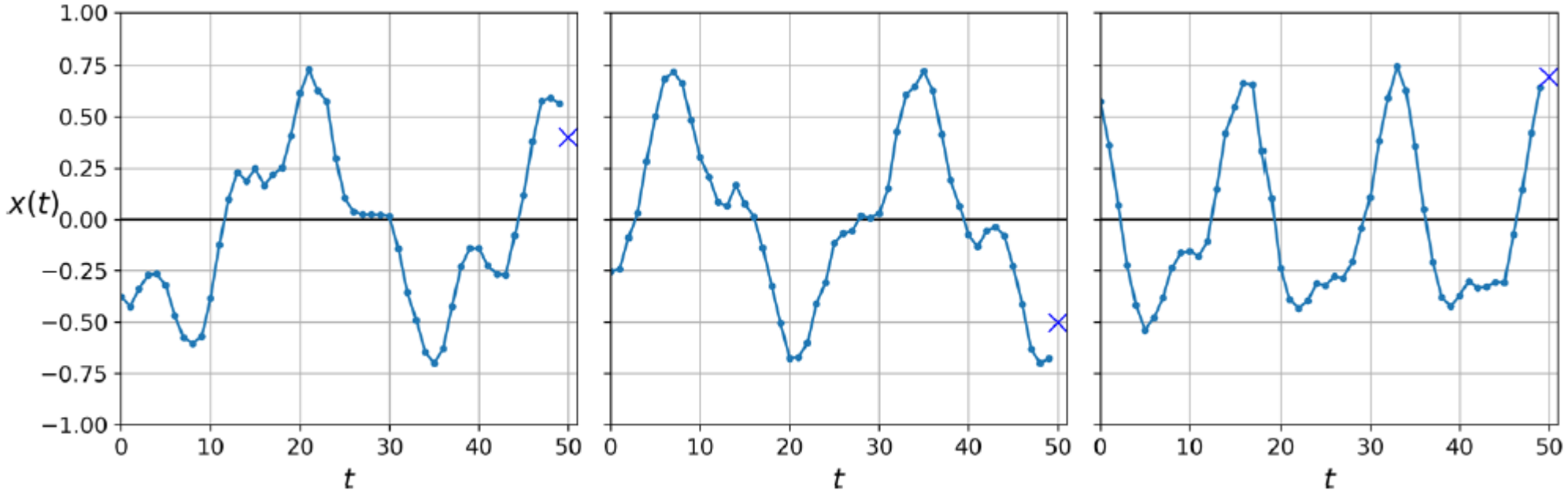
RNN 훈련하기

RNN 훈련하기 위한 기법은 타임 스텝으로 네트워크를 펼치고 역전파를 사용하는 것으로 BPTT(Back Propagation Through Time)라고 합니다.



시계열 예측하기

타입 스텝마다 하나 이상의 값을 시퀀스를 시계열이라고 하며 단변량 시계열, 다변량 시계열 데이터가 있습니다.



시계열 예측하기

■ 시계열 데이터 생성

```
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))  #   wave 1
    series += 0.2 * np.sin((time - offsets2) * (freq2 * 20 + 20)) # + wave 2
    series += 0.1 * (np.random.rand(batch_size, n_steps) - 0.5)   # + noise
    return series[..., np.newaxis].astype(np.float32)
```

■ 훈련 세트, 검증 세트, 테스트 세트 분리

```
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]
```

시계열 예측하기

■ 기준 성능 - naive forecasting

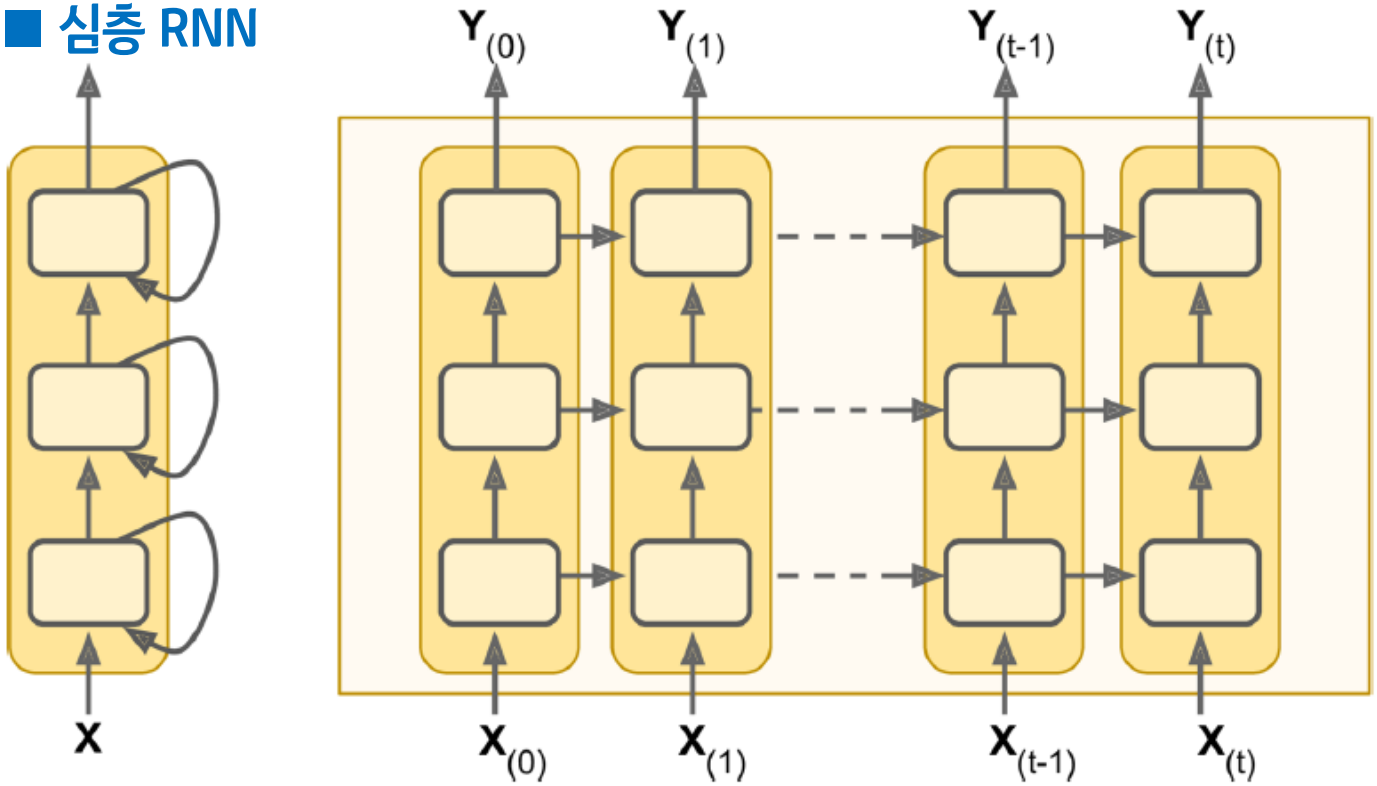
```
>>> y_pred = X_valid[:, -1]
>>> np.mean(keras.losses.mean_squared_error(y_valid, y_pred))
0.020211367
```

■ Simple RNN 구현

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

시계열 예측하기

■ 심층 RNN



■ 훈련 세트, 검증 세트, 테스트 세트 분리

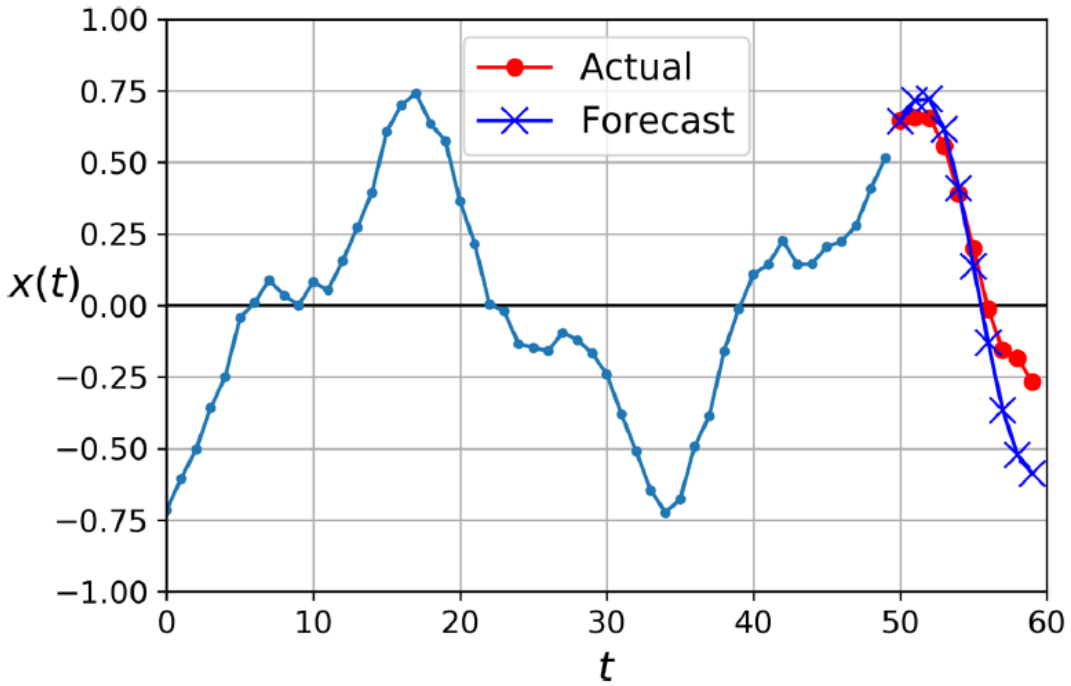
```
model = keras.models.Sequential([
    keras.layers.SimpleRNN(20, return_sequences=True, input_shape=[None, 1]),
    keras.layers.SimpleRNN(20, return_sequences=True),
    keras.layers.SimpleRNN(1)
])
```

시계열 예측하기

■ 여러 타임 스텝 앞을 예측하기 - 예측한 다음 이 값을 입력으로 추가

```
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:]
```



시계열 예측하기

■ 여러 타임 스텝 앞을 예측하기 - 다음 10개 값을 한 번에 예측

```
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)
])

Y_pred = model.predict(X_new)
```

시계열 예측하기

■ 모든 타임 스텝에서 다음 값 10개를 예측하도록 모델을 훈련

```
Y = np.empty((10000, n_steps, 10)) # each target is a sequence of 10D vectors
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))
])
```

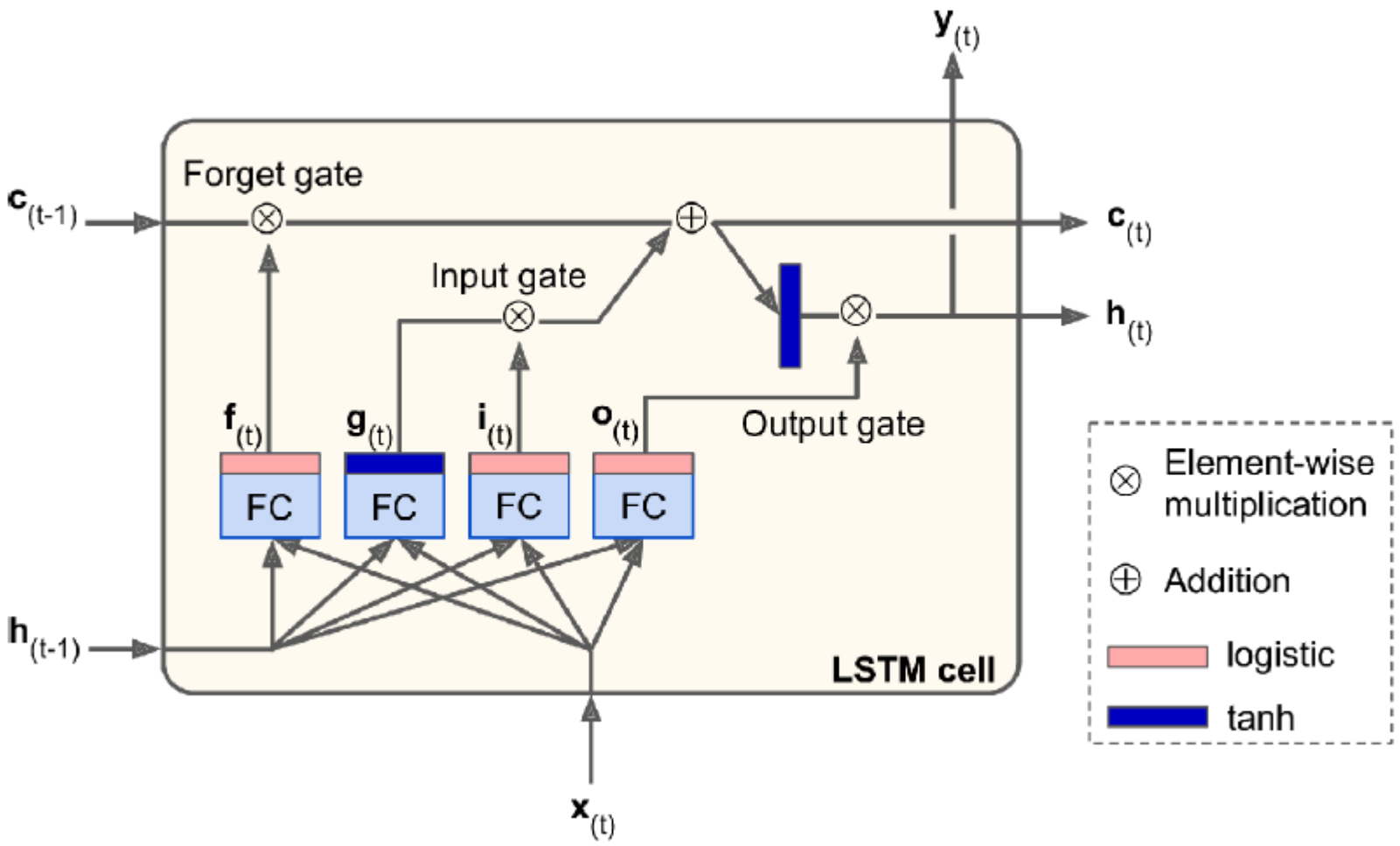
긴 시퀀스 다루기

■ 층 정규화

```
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 = keras.layers.LayerNormalization()
        self.activation = keras.activations.get(activation)
    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]
```

긴 시퀀스 다루기

■ LSTM(Long Short-Term Memory) 셀



$$\mathbf{i}_{(t)} = \sigma\left(\mathbf{W}_{xi}^{\top}\mathbf{x}_{(t)} + \mathbf{W}_{hi}^{\top}\mathbf{h}_{(t-1)} + \mathbf{b}_i\right)$$

$$\mathbf{f}_{(t)} = \sigma\left(\mathbf{W}_{xf}^{\top}\mathbf{x}_{(t)} + \mathbf{W}_{hf}^{\top}\mathbf{h}_{(t-1)} + \mathbf{b}_f\right)$$

$$\mathbf{o}_{(t)} = \sigma\left(\mathbf{W}_{xo}^{\top}\mathbf{x}_{(t)} + \mathbf{W}_{ho}^{\top}\mathbf{h}_{(t-1)} + \mathbf{b}_o\right)$$

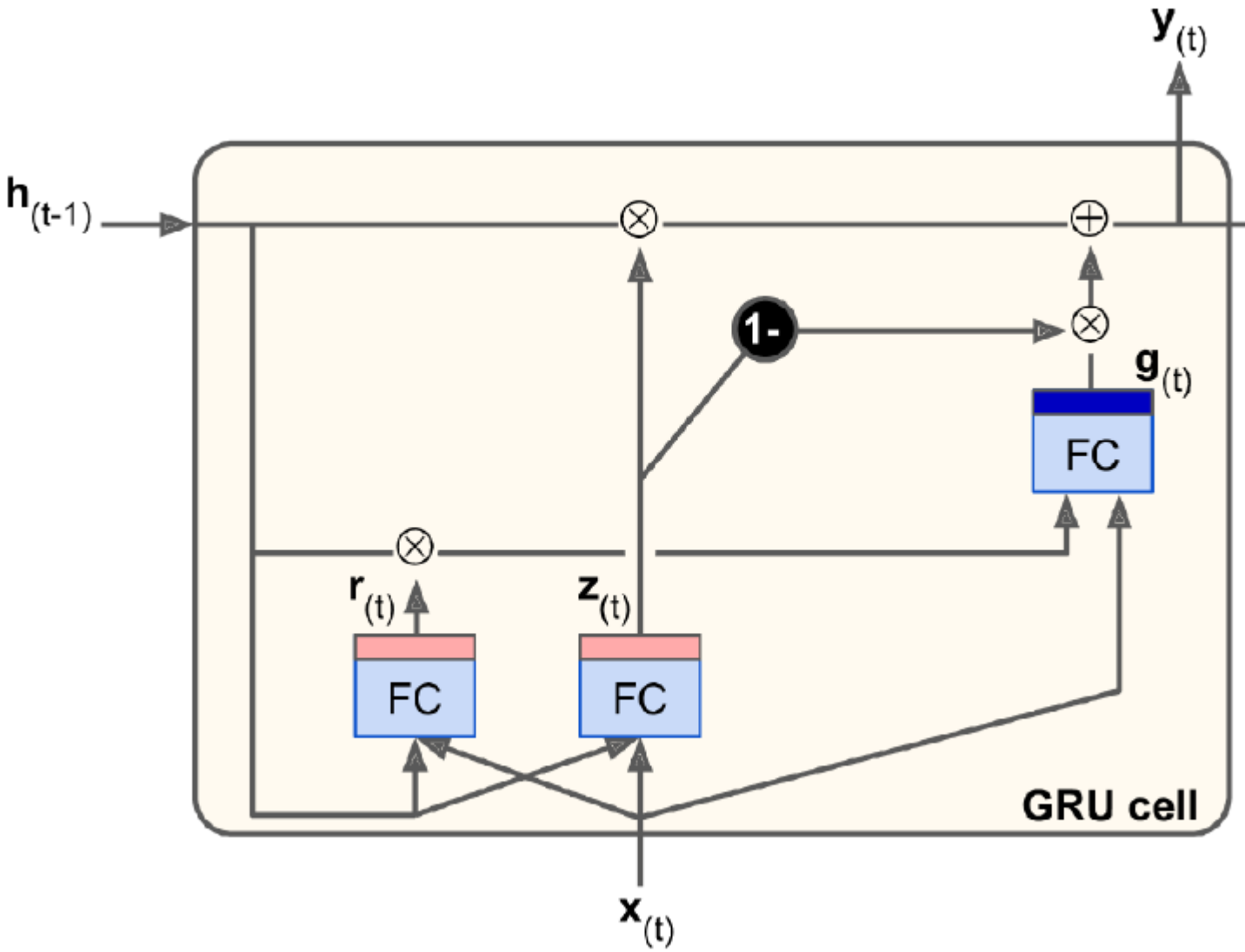
$$\mathbf{g}_{(t)} = \tanh\left(\mathbf{W}_{xg}^{\top}\mathbf{x}_{(t)} + \mathbf{W}_{hg}^{\top}\mathbf{h}_{(t-1)} + \mathbf{b}_g\right)$$

$$\mathbf{c}_{(t)} = \mathbf{f}_{(t)} \otimes \mathbf{c}_{(t-1)} + \mathbf{i}_{(t)} \otimes \mathbf{g}_{(t)}$$

$$\mathbf{y}_{(t)} = \mathbf{h}_{(t)} = \mathbf{o}_{(t)} \otimes \tanh\left(\mathbf{c}_{(t)}\right)$$

긴 시퀀스 다루기

■ GRU(Gated Recurrent Unit) 셀



$$\mathbf{z}_{(t)} = \sigma\left(\mathbf{W}_{xz}{}^{\top}\mathbf{x}_{(t)} + \mathbf{W}_{hz}{}^{\top}\mathbf{h}_{(t-1)} + \mathbf{b}_z\right)$$

$$\mathbf{r}_{(t)} = \sigma\left(\mathbf{W}_{xr}{}^{\top}\mathbf{x}_{(t)} + \mathbf{W}_{hr}{}^{\top}\mathbf{h}_{(t-1)} + \mathbf{b}_r\right)$$

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

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

Hands-on

Thank you

박경규, 010-7240-1128
kgpark88@gmail.com