7. 순환신경망
(Recurrent Neural Network)

2021 2학기 실감 응용 인공지능

고병철

7.1 RNN의 이해

AI Course

- **단일 이미지에서 객체 인식 : CNN 적합**
- **동영상에서 객체 움직임 인식 : RNN 적합**

- **자연어, 동영상, 음악, 주가 차트 .. 공통점?**
  - 시간적 순서를 가지는 시퀀스(Sequence) 기반 데이터
  - 연속적인 시계열(time series) 데이터

  - 시간 순서를 기반으로 데이터들의 상관관계를 파악하고, 이것을 기반으로 현재 및 과거 데이터를 통해서 미래에 발생 될 값을 예측

7.1 RNN의 이해

AI Course

- 기존 신경망 구조 : 모든 입력과 출력이 각각 독립적이라 가정했음
  시간에 따른 입출력 간의 관계는 고려되지 않았음

  - **“우와, 청소 잘했다” vs. “제길, 청소 잘했다”**

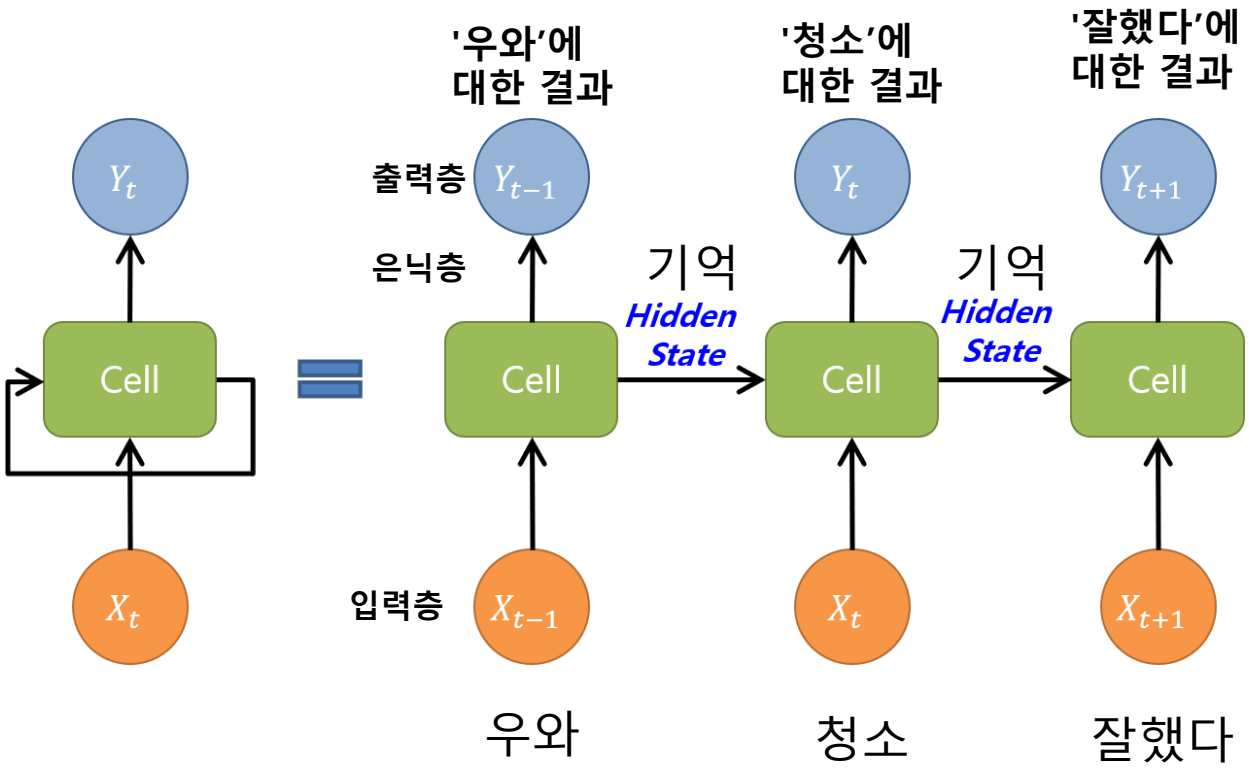
  “청소 잘했다”의 뜻은?

- **RNN(Recurrent Neural Network), 순환 신경망**
  - 이전의 정보가 현재 입력에 대한 결과를 유추할 때 사용되어 더 나은 결과를 예측하는 것을 도울 수 있음
  - 이러한 시퀀스 데이터를 모델링하기 위해 소개된 딥러닝 알고리즘
  - RNN활용의 예: 구글 번역기

7.1 RNN의 이해

AI Course

- RNN(Recurrent Neural Network), 순환 신경망

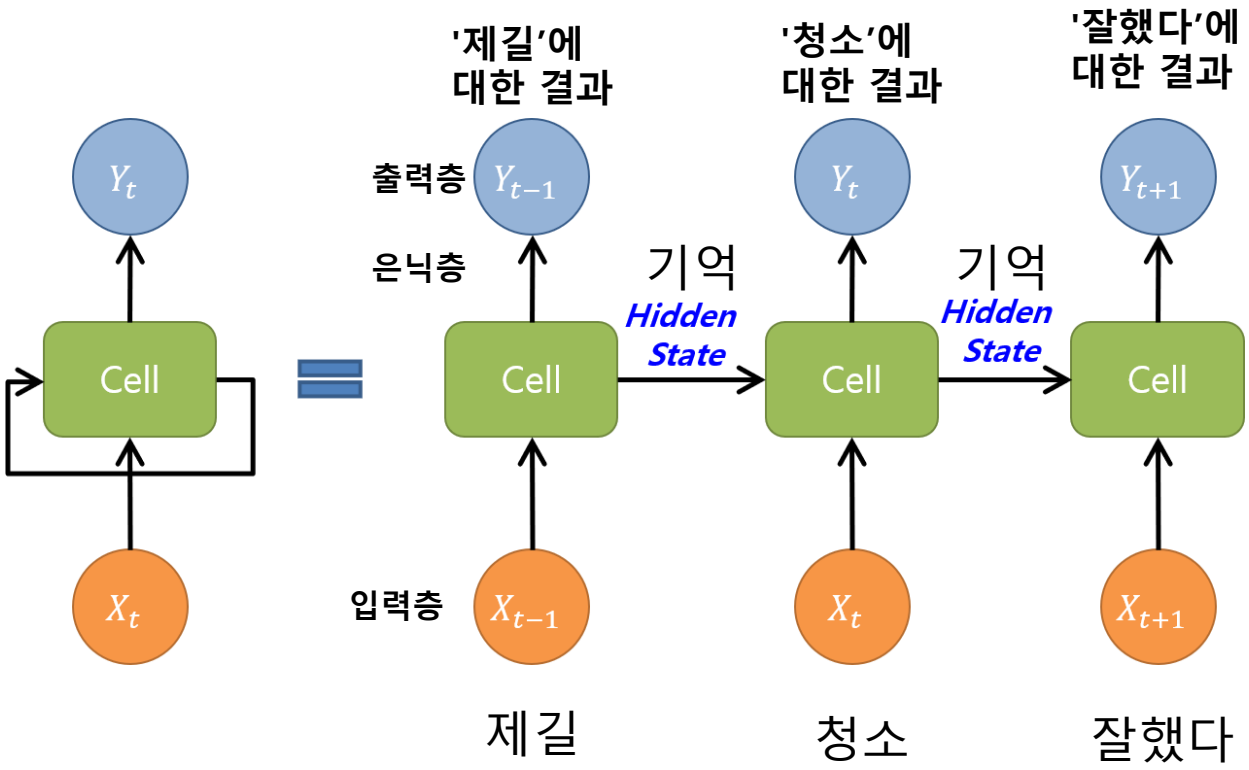


긍정적으로 '잘했다' 라는 의미로 출력

7.1 RNN의 이해

AI Course

- RNN(Recurrent Neural Network), 순환 신경망

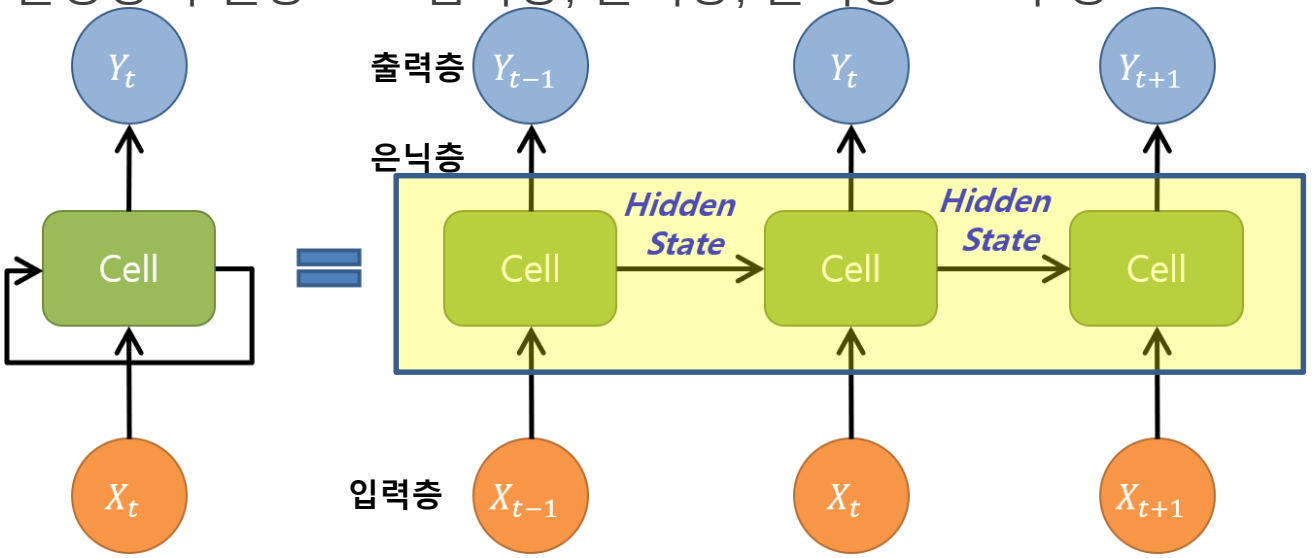


부정적으로 '못해다' 라는 의미로 출력

7.1 RNN

AI Course

- **RNN(순환신경망; Recurrent Neural Network)**
  - RNN도 신경망의 일종으로 입력층, 은닉층, 출력층으로 구성



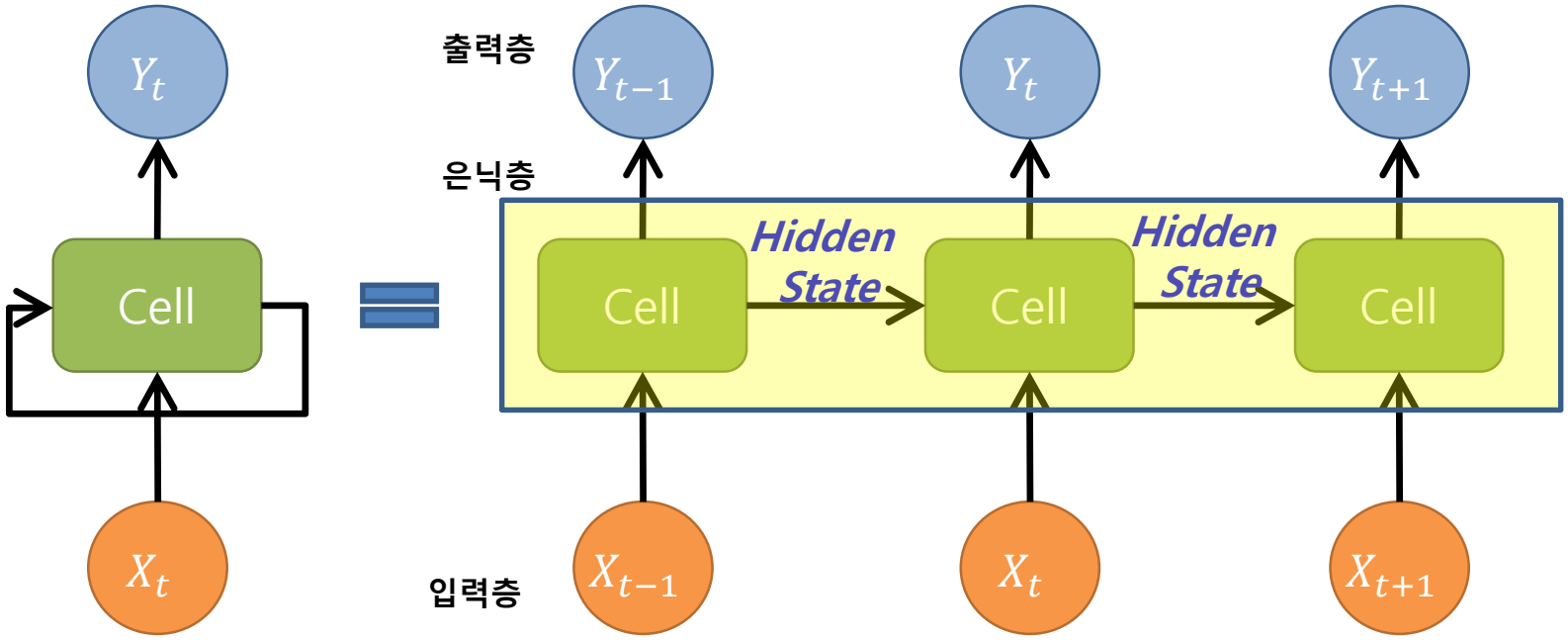
  - 은닉층에서는 하나 이상의 순환적 구조를 가지는 계층으로 구성
    - 이 은닉층을 RNN에서는 메모리 셀(memory cell) 또는 셀(cell)이라고 호칭
    - RNN은 현재 입력된 정보와 이전 정보를 함께 고려하여 현재의 결과를 예측
    - 현재의 정보는 다시 다음 입력에 대한 결과를 예측할 때 사용

  - 각 셀의 은닉 상태(hidden state,기억)에서 각 시퀀스에서의 정보들을 기억하고 있다가 다음 셀에 전달.
  - 전달된 이전 셀의 상태는 현재 셀에서 결과를 예측하기 위한 연산에서 사용

7.1 RNN

AI Course

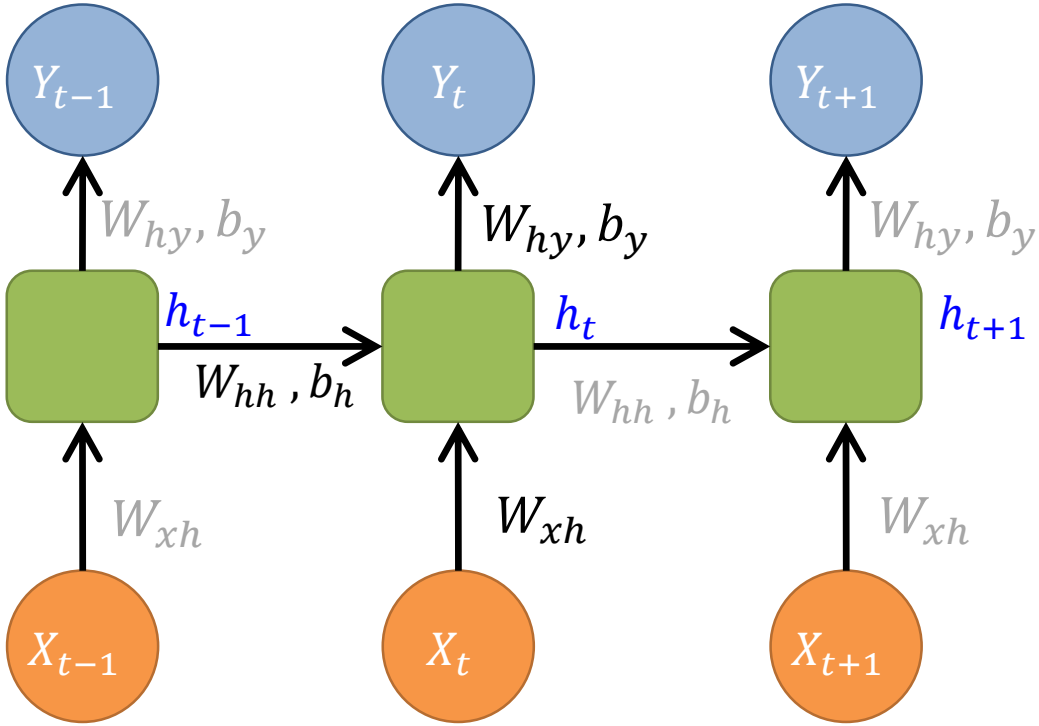
- **RNN(순환신경망; Recurrent Neural Network)**
  - 은닉 계층 안에 하나 이상의 순환 계층을 갖는 신경망
  - RNN의 입력은 순서가 있는 데이터를 순서대로 입력해야 하며, 네트워크를 통해서 순서대로 출력을 얻게 됨
  - 순서가 있는 데이터는 소리, 언어, 날씨, 주가 등의 데이터 처럼 시간의 변화에 함께 변화하면서 그 영향을 받는 데이터를 의미함



7.1 RNN

AI Course

- **RNN(순환신경망; Recurrent Neural Network)**



$$h_t = \tanh(X_t \cdot W_{xh} + h_{t-1} \cdot W_{hh} + b_h) \quad (7.1)$$

$$Y_t = f(h_t \cdot W_{hy} + b_y) \quad (7.2)$$

1. 시간 t 에서의 입력: X_t
   시간 t 에서의 출력: Y_t
   은닉 상태: h_t
   입력층과 셀간의 가중치: W_{xh}
   셀 간의 가중치: W_{hh}
   셀과 출력노드 가중치: W_{hy}
   셀 바이어스: b_h
   출력 노드의 바이어스: b_y

2. 입력에 대한 h_t를 계산.
   - 수식 7.1 이용

3. 시간 t의 출력 :
   - 수식 7.2 이용

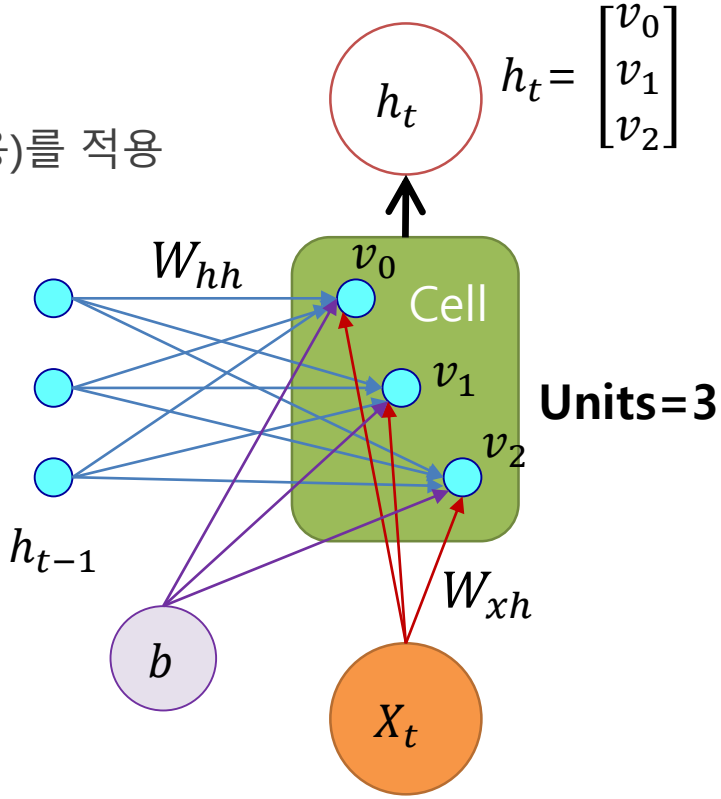
모든 시간대에서 사용되는 파라미터는 동일한 Weight, Bias 등을 공유

7.1 RNN

AI Course

- **RNN의 셀 안에도 은닉노드(unit)가 존재한다~~!**
  - **Units** : RNN 신경망에 존재하는 노드 개수
  - 셀에서 은닉노드가 3개 존재한다면
    - 입력 X_t 와 3개의 은닉노드들의 가중치들 값을 곱 계산 $X_t \cdot W_{xh}$
    - 이전 셀의 은닉노드와 현재 셀의 은닉노드들 간의 가중치 값들과 이전 셀의 은닉상태 값의 곱 계산 $h_{t-1} \cdot W_{hh}$
    - 두 가지 값과 바이어스를 더한 결과를 활성화 함수 (RNN에서는 tanh를 주로 사용)를 적용

$$h_t = \tanh(X_t \cdot W_{xh} + h_{t-1} \cdot W_{hh} + b_h)$$



7.1.1 RNN 구현

AI Course

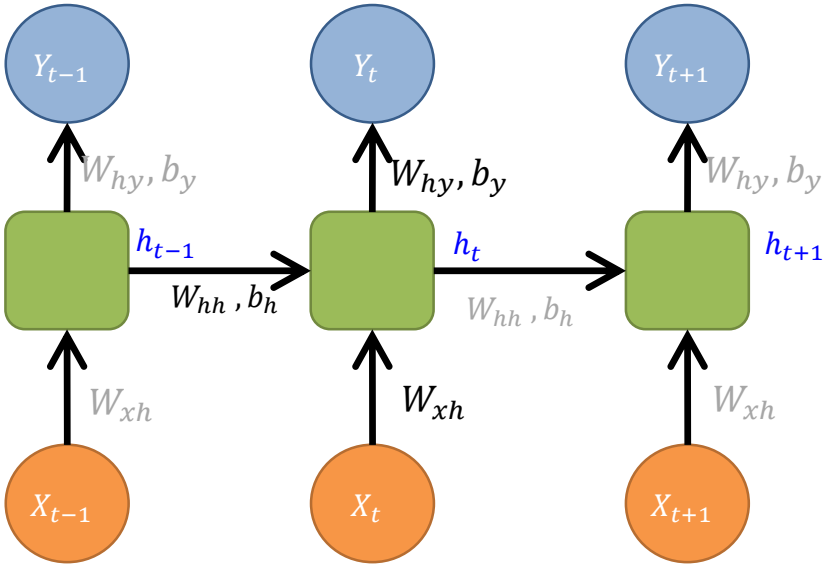
- **텐서플로우 케라스에서는 RNN에 대한 함수를 제공**
  - 함수 내부적으로 RNN 연산들을 모두 처리해주고 있어 간단하게 RNN을 사용해 볼 수 있음
  - 텐서플로우 케라스에서 제공하는 RNN 함수는 tensorflow.keras.layers.SimpleRNN

- **SimpleRNN 함수 알아보기**
  - RNN은 입력부터 최종 출력까지 얻을 수 있는 하나의 완성된 신경망을 표현
  - 케라스 SimpleRNN 함수에서는 X_t를 입력하게 되면 이전 은닉 상태 h_{t-1}을 고려하여 현재셀의 은닉 상태 h_t만 출력.
  - 최종적인 출력값 Y 를 얻고 싶다면 레이어를 추가해야만 해야함.

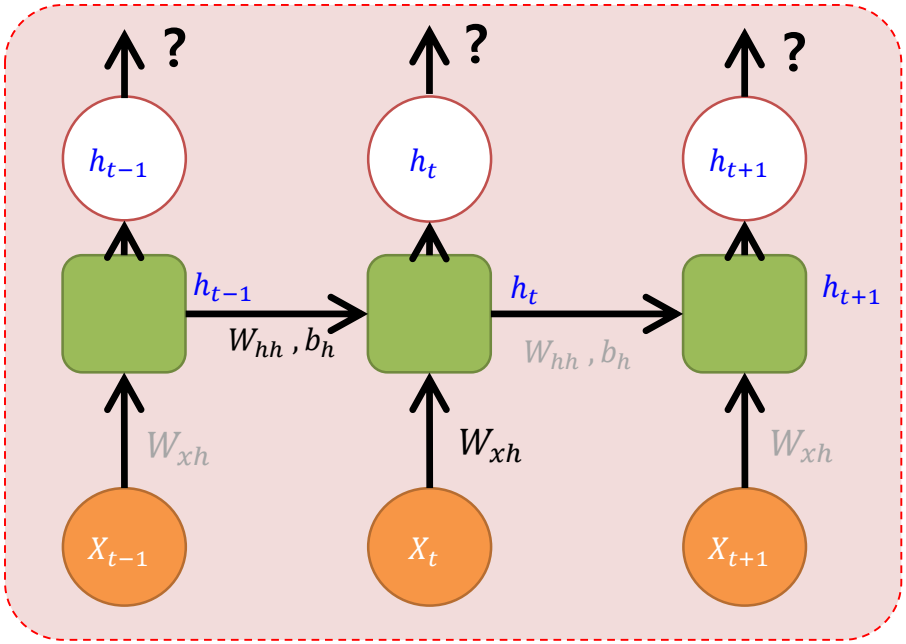
7.1.1 RNN 구현

AI Course

- **케라스 RNN 함수의 표현**
  - 케라스 RNN함수는 우리가 앞에서 봤는 RNN을 기반한 하나의 완성된 신경망을 지원하는 것이 아니라 cell의 출력인 은닉 상태 h만 표현함
  - 따라서 출력층 대한 설정을 해야함



일반 RNN을 표현한 그림
입력층부터 출력층까지 하나의 완성된 신경망을 표현

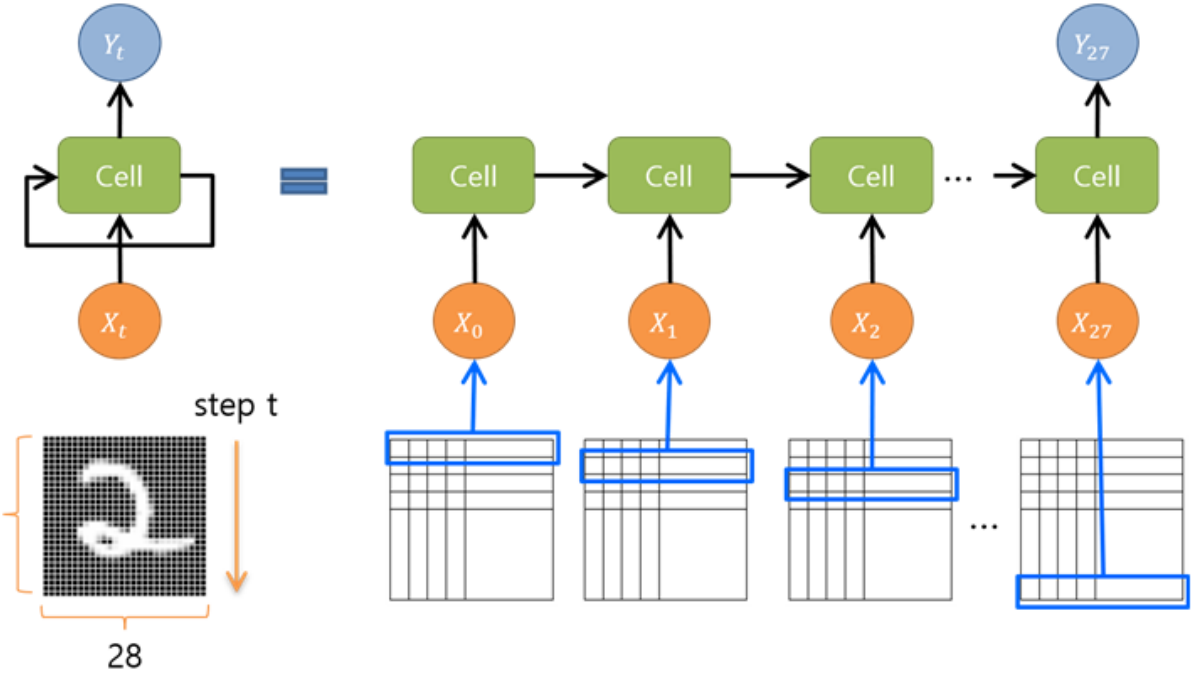


케라스 RNN함수의 그림
입력층과 은닉층만을 표현함

7.1.1 RNN 구현

AI Course

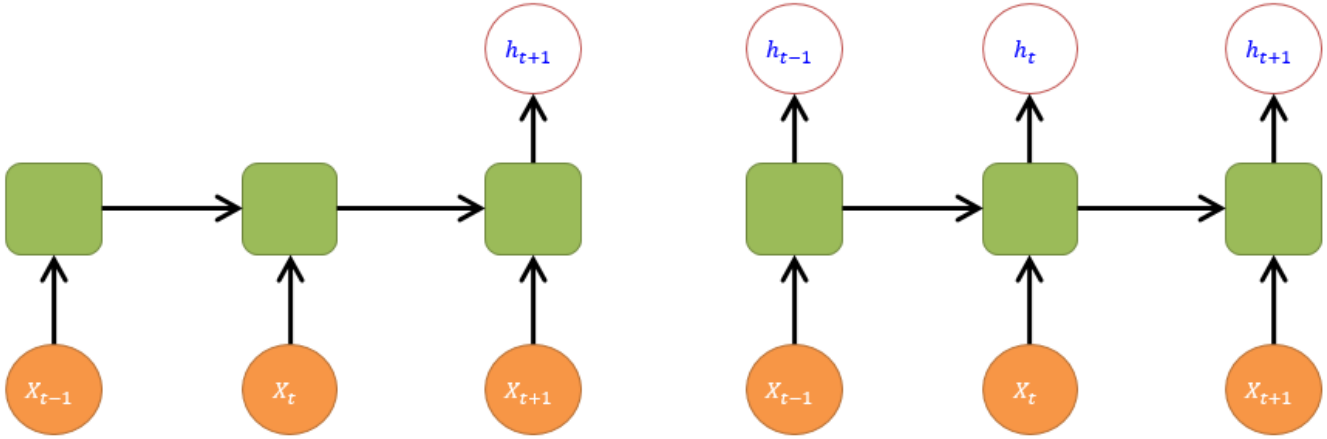
- **시퀀스 데이터를 모델링하기 위해 RNN을 구성**
  - MNIST 데이터셋을 시퀀스 데이터로 변환하여 RNN를 구성
  - MNSIT 데이터를 구성하고 있는 이미지는 행(row)이 28 픽셀, 열(column) 28 픽셀
    - 이미지를 시퀀스 데이터로 재구성하기 위해서 각 타임 스텝마다 한 행씩 RNN에 입력
    - 총 스텝 수는 28이며, 한 스텝마다 입력되는 입력 벡터의 크기는 28



7.1.1 RNN 구현

AI Course

- **시퀀스 데이터를 모델링하기 위해 RNN을 구성**
  - MNIST 데이터셋을 시퀀스 데이터로 변환하여 RNN를 구성
  - RNN은 각 타임 스텝마다 를 추정할 수도 있고, 마지막 스텝에서 최종 결과를 추정할 수도 있음.

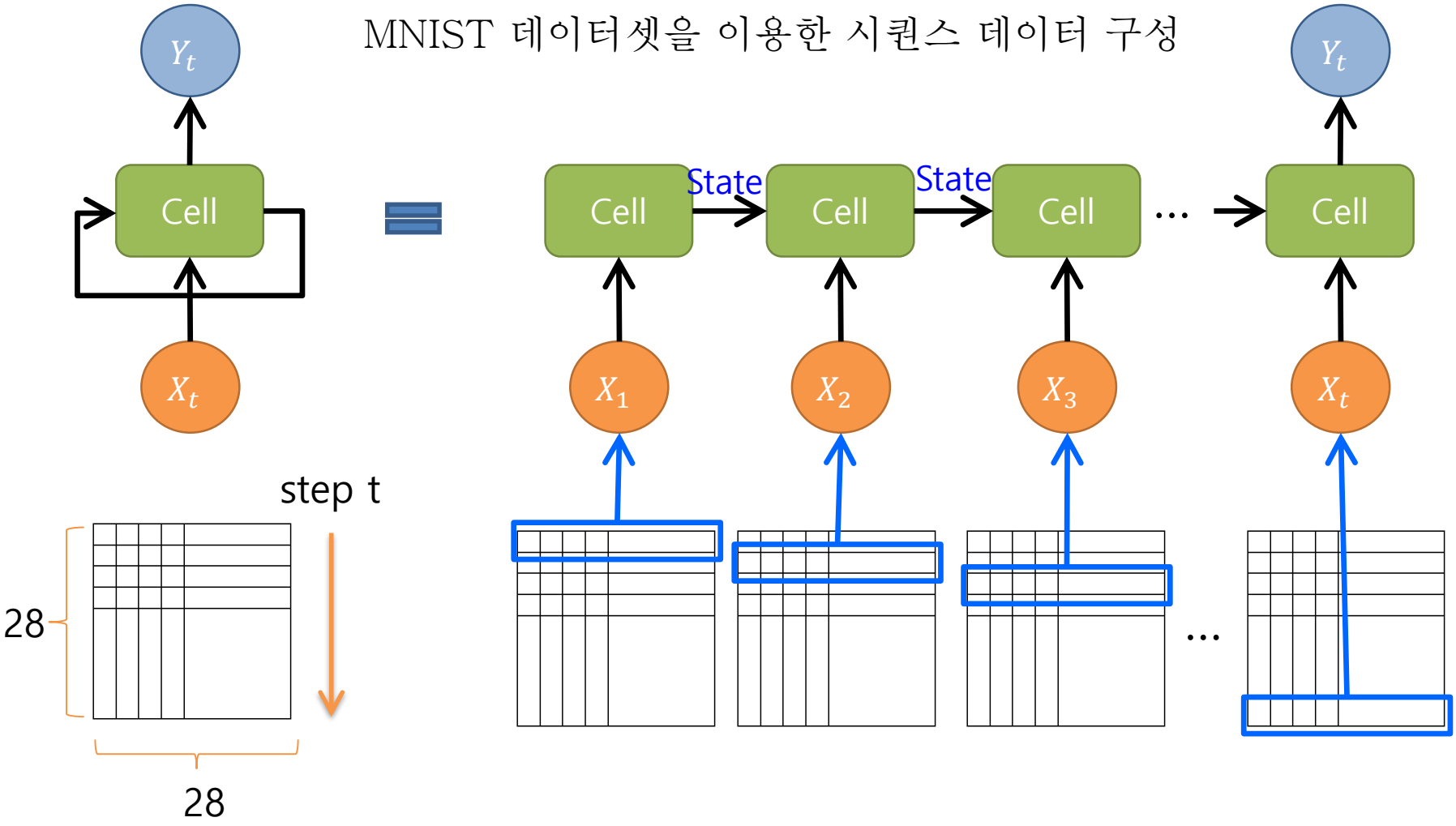


  - 이번 실습에서는 28회의 매 타임 스텝마다 RNN으로 값을 입력하고 마지막 스텝에서 최종적인 Y값을 추정하도록 네트워크를 구성
  - 네트워크의 최종 출력은 숫자 0~9의 10개 클래스에 대한 확률값이 도출되도록 전체 네트워크를 구성.

7.1.1 RNN 구현

AI Course

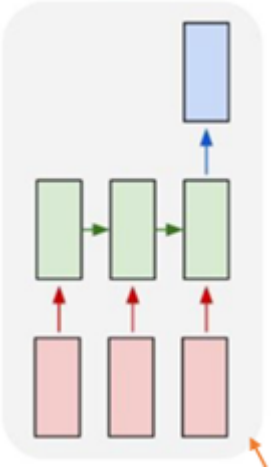
- **MNIST를 이용한 RNN 구현**



7.1.1 RNN 구현

AI Course

- **MNIST를 이용한 RNN 구현**
  - RNN 구현을 위한 TensorFlow 코드 구성 순서



이 형태의 RNN 구성

7.1.1 RNN 구현

AI Course

- **MNIST 를 이용한 RNN 구현**
  - 먼저 필요한 라이브러리 및 모듈을 가져오자.
  - 코드 상에서 필요한 라이브러리 및 특정 모듈을 모두 임포트해보자.

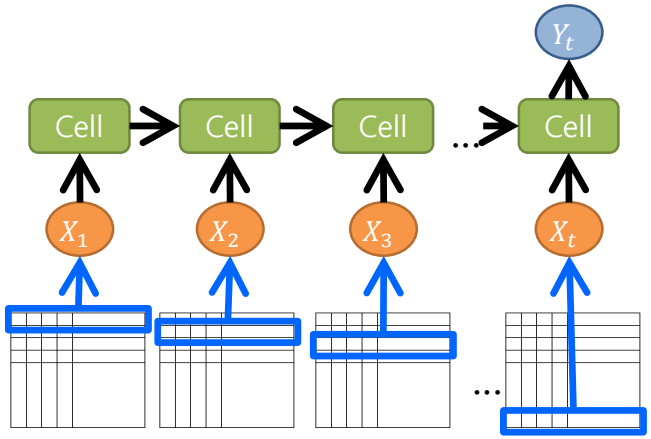
```
# 라이브러리 및 모듈 임포트
import tensorflow as tf
import numpy as np
import matplotlib.pylab as plt
from tensorflow.keras.datasets.mnist import load_data
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import SimpleRNN, Dense
from tensorflow.keras.optimizers import Adam
```

7.1.1 RNN 구현

AI Course

- **데이터 구성**
  - 입력 벡터 : (nb_samples, timesteps, input_dim)
  - MNIST 데이터 : 28x28 image => (nb_samples, 28 height, 28 width)
    - MNIST의 기존 데이터 구성이 (nb_samples, height, width) 이므로 reshape 필요 없음 (*이미지의 높이를 타임 스텝 수로 사용*)

```
(train_images, train_labels), (test_images, test_labels) = tf.keras.datasets.mnist.load_data()

# train_image's shape : (nb_samples, img_height, img_width)이므로 RNN타입과 동일
# Input data of RNN : (nb_samples, timesteps, input_dim)


# 픽셀값 0~1로 정규화
train_images, test_images = train_images / 255.0, test_images / 255.0
```



7.1.1 RNN 구현

AI Course

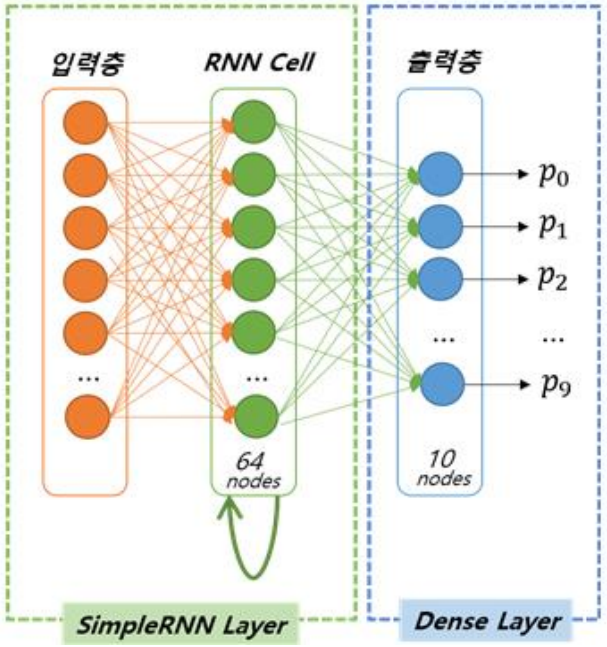
- **데이터 구성**
  - 입력 데이터에 대한 레이블 : (nb_sample, class_size)
  - One-and-hot encoding을 이용하여 표현

```
from tensorflow.keras.utils import to_categorical

one_hot_train_labels = to_categorical(train_labels, 10)
one_hot_test_labels = to_categorical(test_labels, 10)
```

7.1.1 RNN 구현

AI Course

- **모델 구성**
  - RNN layer (Cell node(unit): 64)
  - Fully Connected layer (Node: 10, Activation: Softmax)
  - 구현
    - Tensorflow.keras.layers.SimpleRNN() 함수 사용
    - RNN cell의 노드(unit) 수 : 64개
    - 마지막 RNN의 cell에서만 hidden값 출력
    - RNN의 Input_shape : (28, 28)
    - RNN에서 Output되는 결과 : (batch_size, 10)
    - 다음 Fully Connected layer는 벡터를 입력 받아야함.
      RNN에서 Output되는 결과가 (batch_size, 10) 이므로 즉, 데이터 1개의 크기는 벡터이므로 그 결과 FC layer로 바로 입력 가능
    - Fully Connected layer : Dense layer 사용



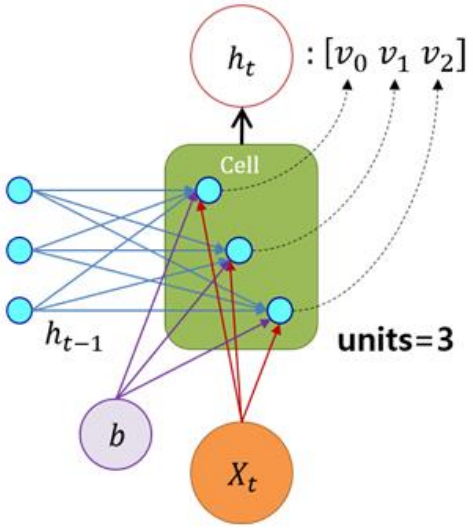

```
model = tf.keras.models.Sequential()

model.add(tf.keras.layers.SimpleRNN(units=64, input_shape = (28, 28),
return_sequences=False))
model.add(tf.keras.layers.Dense(10, activation='softmax'))
```

7.1.1 RNN 구현

AI Course

- **케라스 RNN함수의 파라미터**
  - RNN 네트워크 구성을 위한 코드는 매우 간략
  - RNN 레이어는 SimpleRNN 함수를 살펴보기
    - **① 'units' 매개변수**: 셀에 존재하고 있는 은닉 노드 수
    - RNN에서 출력값의 차원은 셀을 구성하고 있는 노드 수와 동일
    - 예를 들어, 3개 값을 가지는 은닉 상태 값을 표현하고 싶다면, 'units=3'으로 설정하여 셀의 은닉 노드 수를 3개로 구성

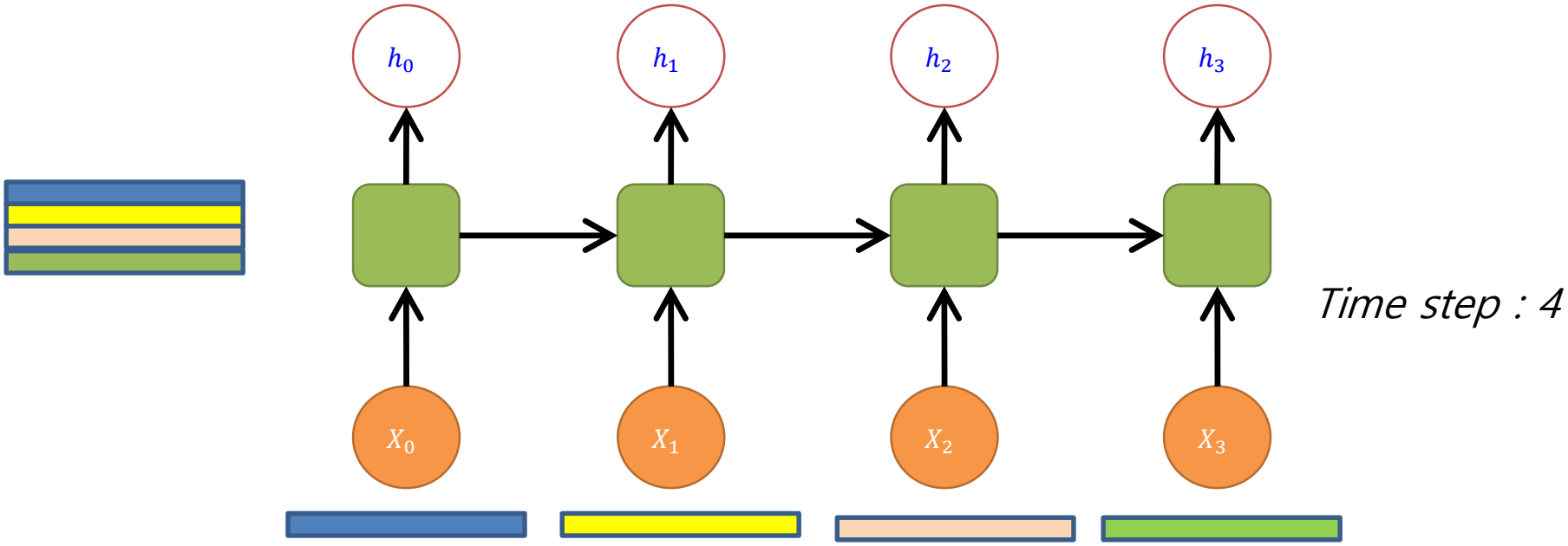
    - 셀의 노드 수를 3개로 설정한 경우
      - 출력 를 계산하는 과정에서 필요한 입력값, 이전 은닉 상태값, 바이어스, 가중치들의 관계를 그림으로 표현
      - 노드 간의 연결 화살표마다 가중치가 존재.



7.1.1 RNN 구현

AI Course

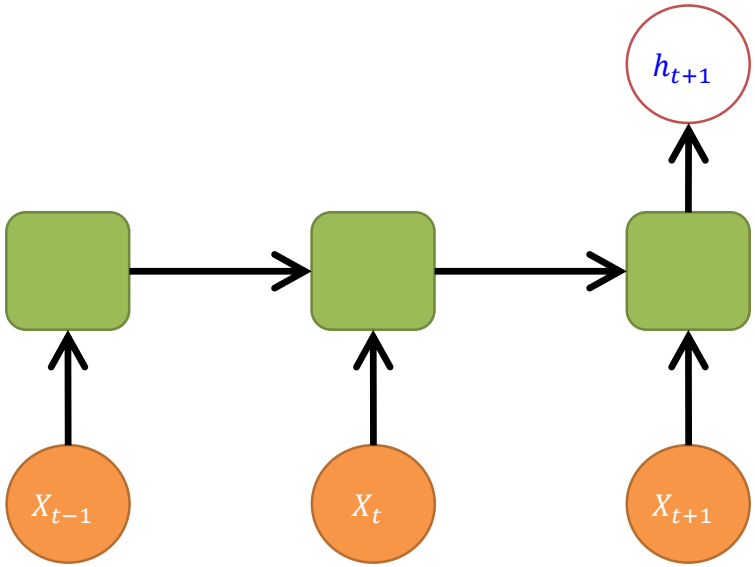
- 케라스 RNN함수의 파라미터
  - ②Input_shape :
    - RNN이 첫번째 layer일 경우 input_shape 파라미터를 지정해야 함
    - input_shape=(timesteps, input_dim)
      - Timesteps : 입력 시퀀스의 길이, 입력데이터로 몇 개를 순차적으로 입력할 것인지를 나타냄
      - Input_dim : 입력 데이터의 벡터 크기



7.1.1 RNN 구현

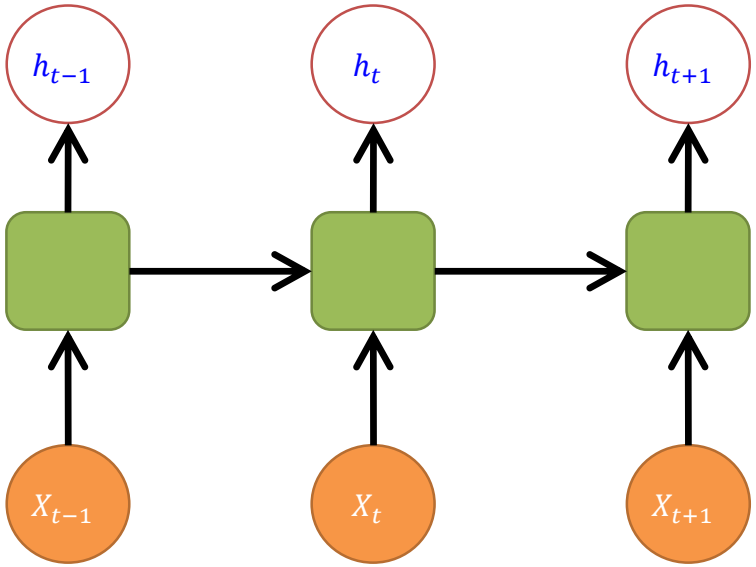
AI Course

- **케라스 RNN함수의 파라미터**
  - **③return_sequences** : RNN 계산 과정에 있는 각 time step에서의 hidden state를 출력할 것인지에 대한 값을 의미



Return_sequences=False

Network's output shape :
2D tensor(batch_size, output_dim)



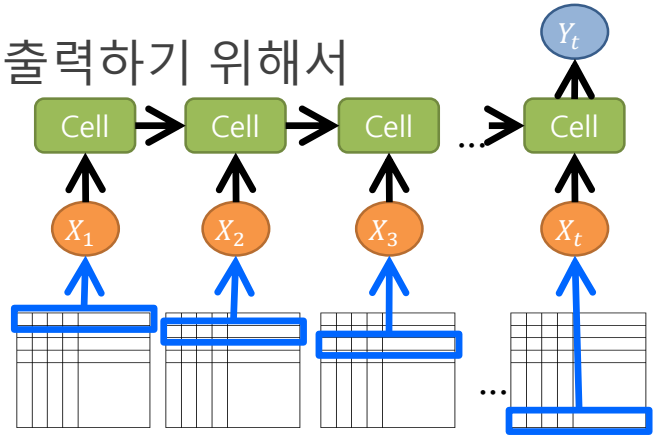
Return_sequences=True

Network's output shape :
3D tensor(batch_size, timesteps, output_dim)

7.1.1 RNN 구현

AI Course

- **모델 구성**
  - RNN layer (Cell node: 64)
  - Fully Connected layer (Node: 10, Activation: Softmax)
  - 구현
    - RNN cell은 마지막 time step에서 결과 출력하기 위해서 **return_sequences = False**




```
model = Sequential()
model.add(SimpleRNN(units=64, input_shape = (28,
                                             28),return_sequences=False))
model.add(Dense(10, activation='softmax'))
```

7.1.1 RNN 구현

AI Course

- **학습 과정 설정**
  - 최적화 함수 : Adam(learning_rate : 0.001)
  - 손실 함수 : categorical_crossentropy
  - 평가 함수 : accuracy

```
model.compile(optimizer=tf.optimizers.Adam(learning_rate=0.001)
, loss='categorical_crossentropy', metrics=['accuracy'])
```

- **학습**
  - Epoch : 5
  - Batch_size : 32

```
model.summary()
history = model.fit(train_images, one_hot_train_labels, epochs=
5, batch_size=32)
```

7.1.1 RNN 구현

AI Course

- 모델 평가

```
print("\n============test results==========")
labels=model.predict(test_images)
print("\n Accuracy: %.4f" % (model.evaluate(test_images,
                                    one_hot_test_labels)[1]))

fig = plt.figure()
for i in range(10):
    subplot = fig.add_subplot(2, 5, i+1)
    subplot.set_xticks([])
    subplot.set_yticks([])
    subplot.set_title('%d' % np.argmax(labels[i]))
    subplot.imshow(test_images[i],
                   cmap=plt.cm.gray_r)
plt.show()

print("====================================")
```

7.1.1 RNN 구현

AI Course

- 전체 코드 (연습 7-1)

```
import tensorflow as tf
import numpy as np
import matplotlib.pylab as plt
from tensorflow.keras.datasets.mnist import load_data
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import SimpleRNN, Dense
from tensorflow.keras.optimizers import Adam

# MNIST 데이터셋을 로드하여 준비
(train_images, train_labels), (test_images, test_labels) = load_data()

# shape of train_image & test_image : (nb_samples, image_height, image_width)
# Input data of RNN : (nb_samples, timesteps, input_dim)

# 픽셀값 0~1로 정규화
train_images, test_images = train_images / 255.0, test_images / 255.0

# 데이터 레이블 one hot 코드 변경
one_hot_train_labels = to_categorical(train_labels, 10)
one_hot_test_labels = to_categorical(test_labels, 10)

# 모델 구성
model = Sequential()
model.add(SimpleRNN(units=64,input_shape = (28, 28),return_sequences=False))
model.add(Dense(10, activation='softmax'))
```

7.1.1 RNN 구현

AI Course

- 전체 코드

```
# 모델 학습을 위한 최적화 함수 및 손실 함수 등 설정
model.compile(optimizer=Adam(learning_rate=0.001),
              loss='categorical_crossentropy', metrics=['accuracy'])

# 모델 구조 출력
model.summary()

# 모델 학습 수행
history = model.fit(train_images, one_hot_train_labels, epochs=5,batch_size=32)

# 모델 평가
print("₩n=============test results==========")
labels=model.predict(test_images)
print("₩n Accuracy: %.4f" % (model.evaluate(test_images,
                                            one_hot_test_labels)[1]))

# 결과 이미지로 출력
fig = plt.figure()
for i in range(10):
    subplot = fig.add_subplot(2, 5, i+1)
    subplot.set_xticks([])
    subplot.set_yticks([])
    subplot.set_title('%d' % np.argmax(labels[i]))
    subplot.imshow(test_images[i],
                   cmap=plt.cm.gray_r)
plt.show()

print("==================================")
```

7.1.1 RNN 구현

AI Course

- 결과 화면

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11493376/11490434 [==============================] - 0s 0us/step
[[0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]]
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
simple_rnn (SimpleRNN)       (None, 64)                5952
_________________________________________________________________
dense (Dense)                (None, 10)                650
=================================================================
Total params: 6,602
Trainable params: 6,602
Non-trainable params: 0
_________________________________________________________________
```

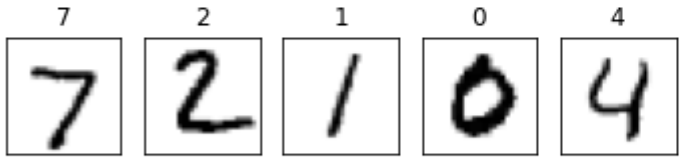
{28x64}+{64}+{64+64}
=1792+64+4096 =5952

7.1.1 RNN 구현

AI Course

- 결과 화면

```
----------------------------------------------------------------
Epoch 1/5
1875/1875 [==============================] - 11s 6ms/step - loss: 0.5374 - accuracy: 0.8364
Epoch 2/5
1875/1875 [==============================] - 11s 6ms/step - loss: 0.2781 - accuracy: 0.9200
Epoch 3/5
1875/1875 [==============================] - 11s 6ms/step - loss: 0.2305 - accuracy: 0.9341
Epoch 4/5
1875/1875 [==============================] - 11s 6ms/step - loss: 0.2053 - accuracy: 0.9416
Epoch 5/5
1875/1875 [==============================] - 11s 6ms/step - loss: 0.1860 - accuracy: 0.9471

=============test results==========
313/313 [==============================] - 1s 3ms/step - loss: 0.2004 - accuracy: 0.9437

 Accuracy: 0.9437
```

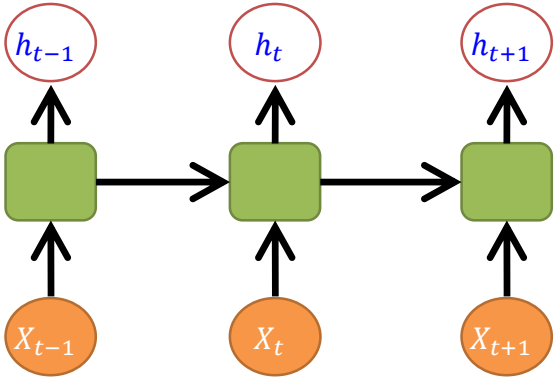



```
==================================
```

7.1.1 RNN 구현

AI Course

- RNN 각 cell에서 hidden 값 출력하기
  - return_sequences = True




```
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.SimpleRNN(units=64, return_sequences =
True,input_shape = (28, 28)))
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(32, activation='relu'))
model.add(tf.keras.layers.Dense(10, activation='softmax'))
```

7.1.1 RNN 구현

AI Course

- **RNN 각 cell에서 hidden 값 출력하기**
  - return_sequence = True
    - Return_sequence = True를 사용하였을 경우,
      RNN의 출력 shape : (Batch_size, time_steps, output_dims)
    - 뒤의 FC layer에 입력하기 위해서는 각 스텝 별 벡터를 1개 벡터로 변경해야함. 즉, FC layer 전에 Flatten()해야함
    - (Batch_size, **time_steps, output_dims**)

      => (Batch_size, **time_steps*output_dim**)

```
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.SimpleRNN(units=64, return_sequences
 = True,input_shape = (28, 28)))
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(32, activation='relu'))
model.add(tf.keras.layers.Dense(10, activation='softmax'))
```

7.1.1 RNN 구현

AI Course

- **RNN 각 cell에서 hidden 값 출력하기**
  - 실행 결과

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
simple_rnn_15 (SimpleRNN)    (None, 28, 64)            5952
_________________________________________________________________
flatten_2 (Flatten)          (None, 1792)              0
_________________________________________________________________
dense_24 (Dense)             (None, 32)                57376
_________________________________________________________________
dense_25 (Dense)             (None, 10)                330
=================================================================
Total params: 63,658
Trainable params: 63,658
Non-trainable params: 0
_________________________________________________________________
Epoch 1/5
1875/1875 [==============================] - 13s 7ms/step - loss: 0.2422 - accuracy: 0.9270
Epoch 2/5
1875/1875 [==============================] - 13s 7ms/step - loss: 0.1140 - accuracy: 0.9661
Epoch 3/5
1875/1875 [==============================] - 13s 7ms/step - loss: 0.0895 - accuracy: 0.9726
Epoch 4/5
1875/1875 [==============================] - 14s 7ms/step - loss: 0.0738 - accuracy: 0.9770
Epoch 5/5
1875/1875 [==============================] - 14s 7ms/step - loss: 0.0636 - accuracy: 0.9804

==========test results==========
313/313 [==============================] - 1s 3ms/step - loss: 0.0824 - accuracy: 0.9772

Accuracy: 0.9772
```

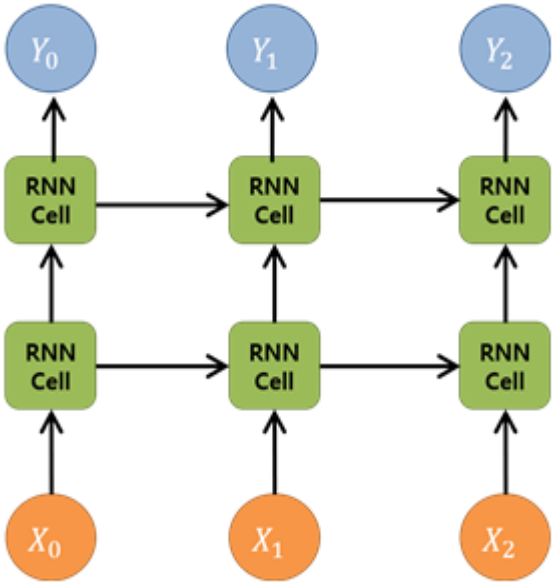
28x64

1792x32+bias

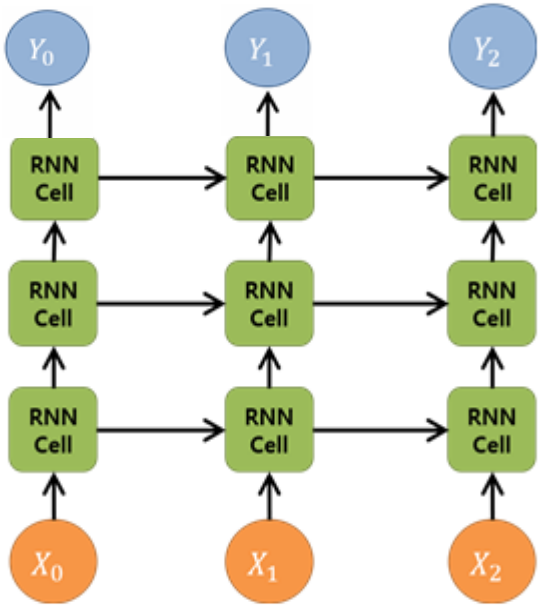
7.1.2 심층RNN 구현하기

AI Course

- **심층 순환 신경망 (Deep Recurrent Neural Network)**
  - RNN을 여러 개의 은닉층을 가지도록 구성한 신경망
  - 심층 RNN에서 첫 번째 RNN 레이어는 입력 데이터가 입력되며, 마지막 RNN 레이어에서는 원하는 시점에서의 RNN 셀에서만 최종값을 출력
  - 심층 RNN을 구성하는 중간 RNN 레이어에서는 이전 RNN 레이어에서 출력된 값을 다음 RNN 레이어의 입력으로 전달해야 하므로 모든 시퀀스마다 출력값을 가져야 함.



< 2개의 RNN 레이어로 구성된 네트워크 >

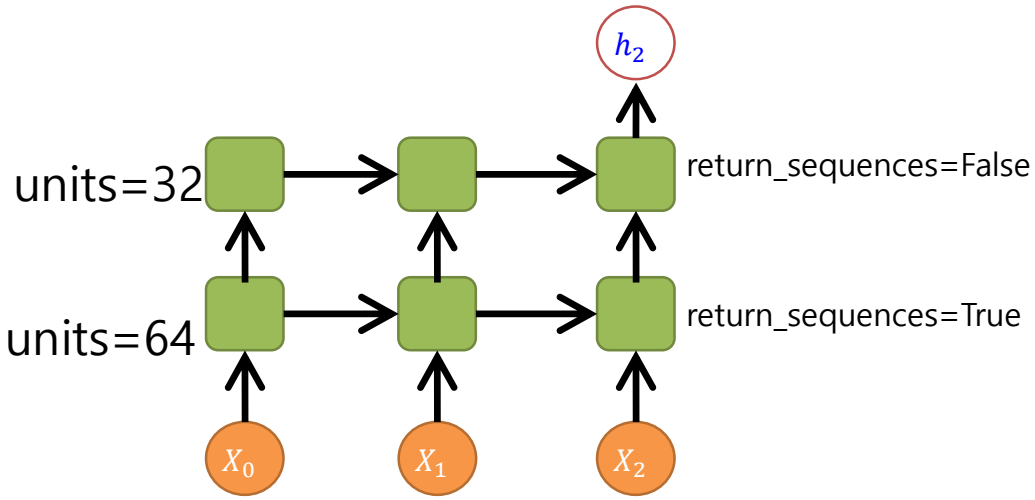


< 3개의 RNN 레이어로 구성된 네트워크 >

7.1.2 심층RNN 구현하기

AI Course

- **깊은 순환 신경망 (Deep Recurrent Neural Network)**
  - RNN을 여러 개의 은닉층을 가지도록 구성한 신경망



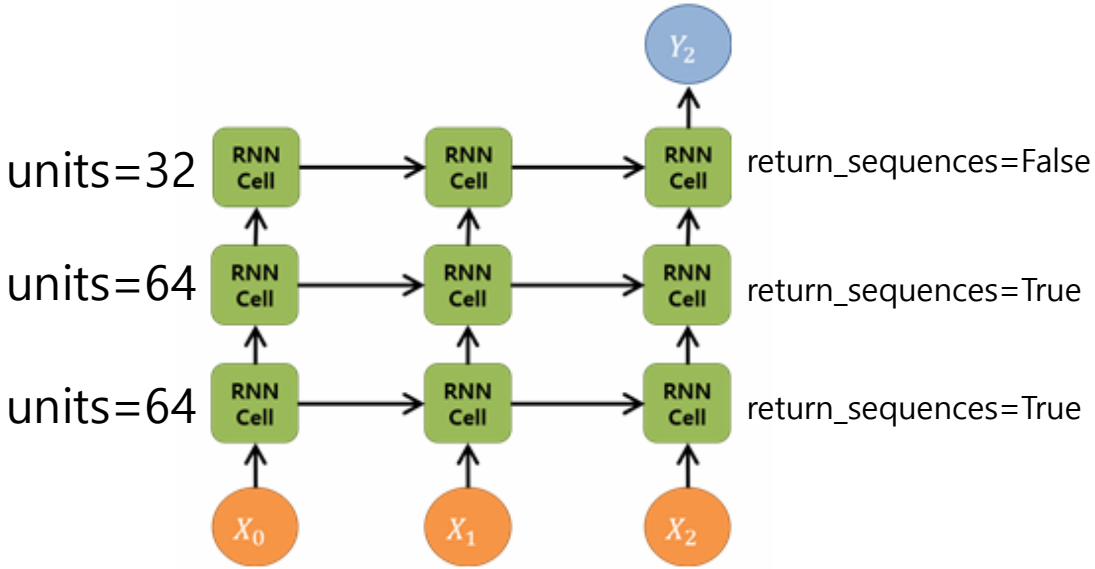

```
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.SimpleRNN(units=64, return_sequences=True,
input_shape = (28, 28)))
model.add(tf.keras.layers.SimpleRNN(units=32))
model.add(tf.keras.layers.Dense(10, activation='softmax'))
```

- 첫번째 RNN 레이어의 출력을 두번째 RNN 레이어의 입력으로 사용해야하므로, 첫번째 RNN 레이어에서는 각 시퀀스마다 결과를 output 하도록 해야함 (return_sequences=True)

7.1.2 심층RNN 구현하기

AI Course

- **깊은 순환 신경망 (Deep Recurrent Neural Network)**
  - RNN을 여러 개의 은닉층을 가지도록 구성한 신경망




```
model = Sequential()
model.add(SimpleRNN(units=64, return_sequences=True, input_shape = (28, 28)))
model.add(SimpleRNN(units=64, return_sequences=True))
model.add(SimpleRNN(units=32))
model.add(Dense(10, activation='softmax'))
```

7.1.2 심층RNN 구현하기

AI Course

- **깊은 순환 신경망 (Deep Recurrent Neural Network)**
  - RNN을 여러 개의 은닉층을 가지도록 구성한 신경망

연습 (7-2)

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
simple_rnn (SimpleRNN)       (None, 28, 64)            5952
_________________________________________________________________
simple_rnn_1 (SimpleRNN)     (None, 28, 64)            8256
_________________________________________________________________
simple_rnn_2 (SimpleRNN)     (None, 32)                3104
_________________________________________________________________
dense (Dense)                (None, 10)                330
=================================================================
Total params: 17,642
Trainable params: 17,642
Non-trainable params: 0
```

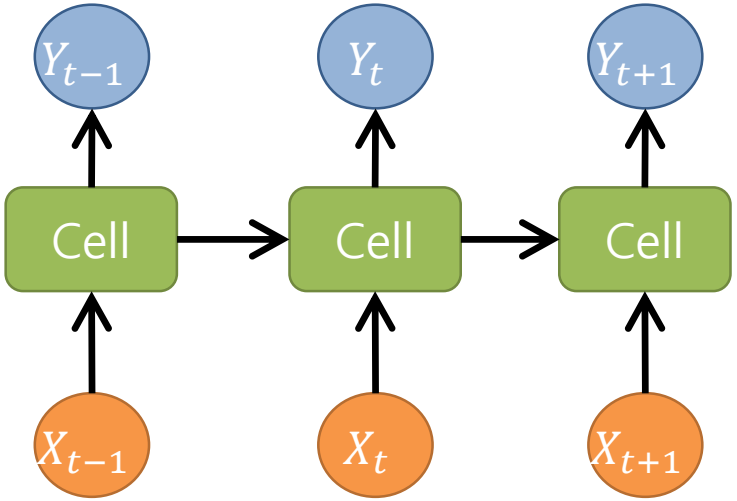
7.2 LSTM의 이해

AI Course

- **바닐라 RNN의 한계**
  - 짧은 sequence에 대해서만 효과를 보임
  - 고려하는 Time step이 길어질 수록 이전의 정보가 뒤쪽으로 충분히 전달되지 못하는 현상이 발생

  == 장기 의존성 문제(The problem of Long-Term Dependencies)

RNN은 Memento??

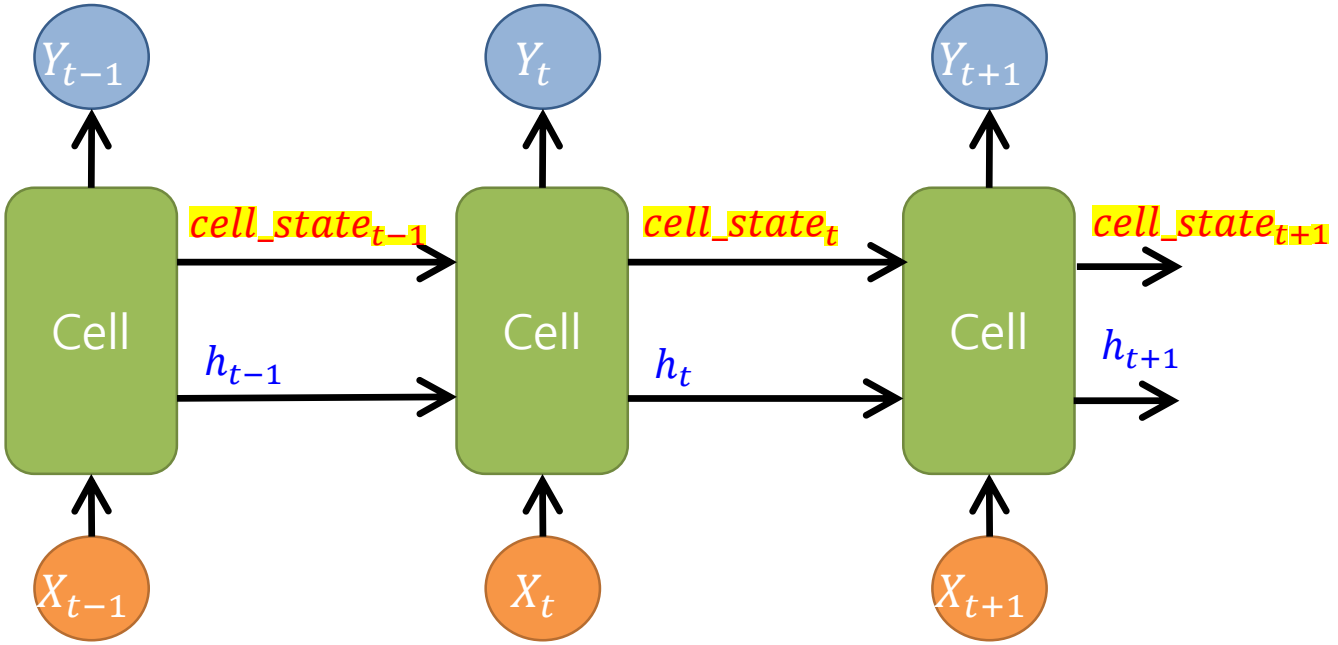


. "I grew up in France..... I speak fluent '*??*' "

7.2 LSTM의 이해

AI Course

- **장단기 메모리 (Long Short-Term Memory)**
  - RNN의 장기 의존성 문제를 개선할 있으며, 학습 또한 빠르게 수렴함
  - 시퀀스를 고려할 수 있는 딥러닝 알고리즘 중에 많이 사용
  - 장기 기억을 위해서 **셀 상태(cell state)를 추가**
  - 기존의 RNN의 은닉 상태에 셀 상태가 추가된 구조
    - **장기 기억 : 셀 상태, 단기 기억 : 은닉 상태**

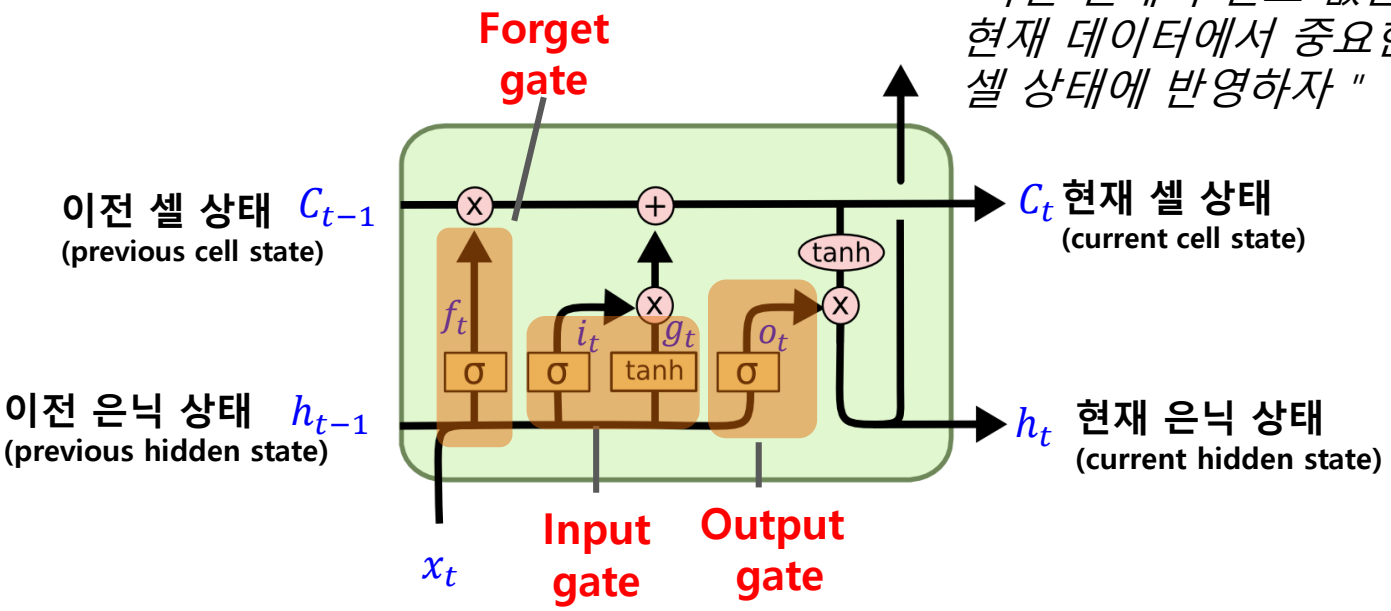


7.2 LSTM의 이해

AI Course

- **장단기 메모리 (Long Short-Term Memory)**
  - **3개의 게이트**로 구성
    - **삭제 게이트(forget gate)** : 이전 셀 상태 정보에서 삭제시킬 기억을 제어함
    - **입력 게이트(input gate)** : 현재 입력된 정보 중에서 어떤 것을 셀 상태에 저장할 것인지를 제어함
    - **출력 게이트(output gate)** : 셀 상태의 어느 부분을 읽어서 현재의 은닉 상태를 출력할 것인지를 제어

"이전 셀에서 필요 없는 것은 삭제하고 현재 데이터에서 중요한 것을 새롭게 셀 상태에 반영하자 "



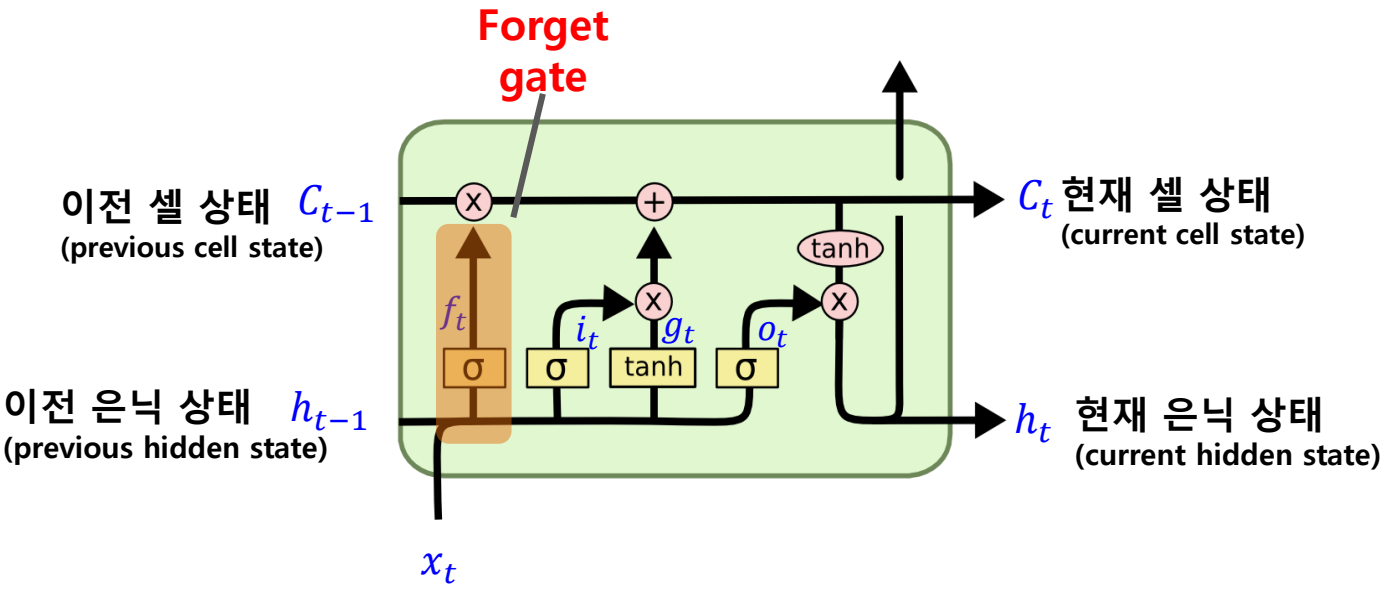
7.2 LSTM의 이해

AI Course

- **삭제 게이트 : 셀 상태에서 감쇠 및 삭제시킬 기억을 결정**
  - 이전 은닉 상태(h_{t-1})과 현재 입력(x_t)을 입력 받아서 시그모이드 함수(σ)를 통과시킴으로써 그 결과 f_t 는 0과 1 사이의 값을 가지게 되고, 원소별(element-wise) 곱셈연산에 의해서 셀 상태값을 제어
  - 만약 f_t 값이 1이라면 셀 상태의 모든 정보를 보존하게 되고, 그 값이 0이라면 셀 상태의 모든 정보는 삭제시킴

⊗ ⊕ 는 모두 element-wise연산

$$f_t = \sigma(W_{xf}^T \cdot x_t + W_{hf}^T \cdot h_{t-1} + b_f)$$



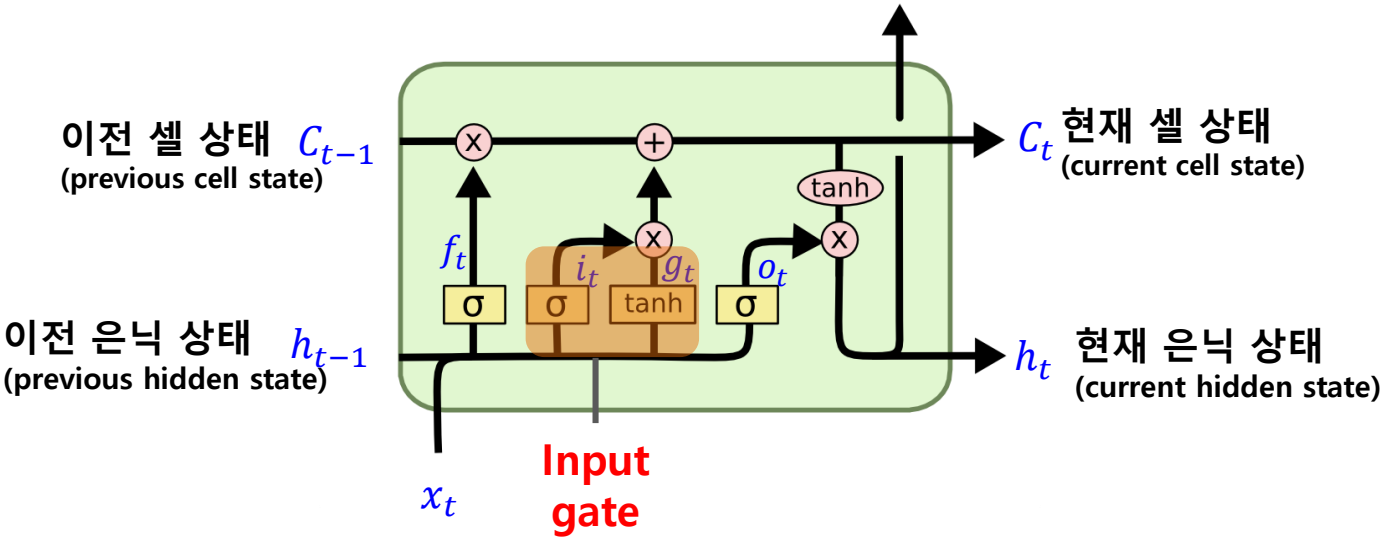
7.2 LSTM의 이해

AI Course

- **입력 게이트 : 현재 입력된 정보 중에서 어떤 것을 셀 상태에 저장할 것인지를 결정**
  - 이전 은닉 상태(h_{t-1})과 현재 입력(x_t)을 입력 받아서 시그모이드 함수(σ)와 tanh 함수를 각각 통과시킴으로써 i_t와 g_t를 계산함
  - g_t 는 이전 은닉 상태 (h_{t-1})와 현재 입력 (x_t)를 받아서 새로운 상태값을 생성하며, i_t 는 g_t 의 어느 부분이 기존의 셀 상태와 더해져야 하는지를 제어함

$$i_t = \sigma(W_{xi}^T \cdot x_t + W_{hi}^T \cdot h_{t-1} + b_i)$$

$$g_t = \tanh(W_{xg}^T \cdot x_t + W_{hg}^T \cdot h_{t-1} + b_g)$$

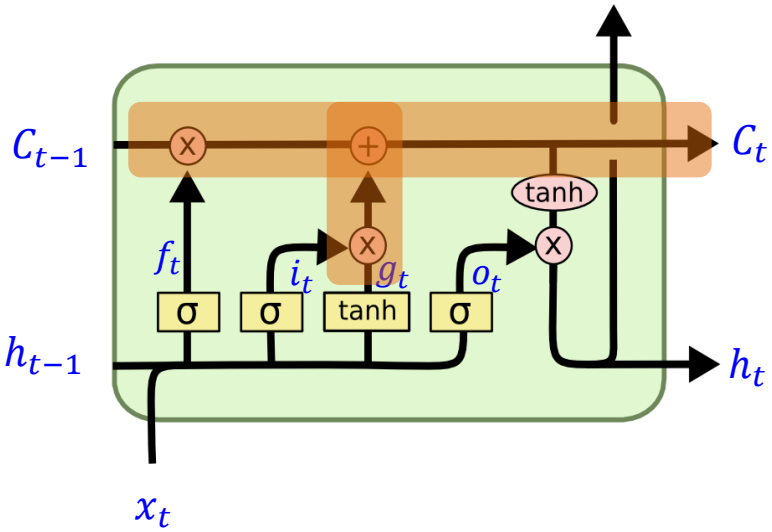


7.2 LSTM의 이해

AI Course

- **셀 상태 업데이트**
  - 이전 셀 상태에 삭제 게이트의 결과 f_t 를 원소별 곱셈을 수행해서 셀 상태 정보를 삭제시키고, 입력 게이트의 결과인 $i_t \otimes g_t$ 를 이용하여 셀 상태 C를 업데이트 시킴

$$C_t = f_t \otimes C_{t-1} + i_t \otimes g_t$$



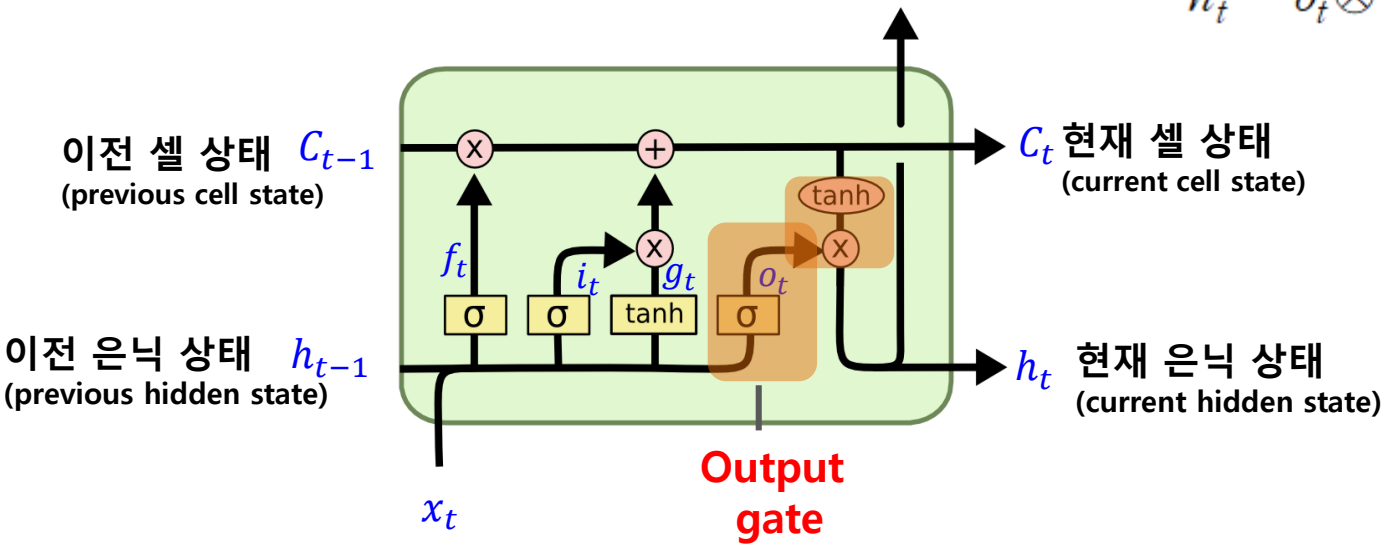
7.2 LSTM의 이해

AI Course

- **출력 게이트 : 업데이트된 셀 상태 를 기반으로 특정 부분을 읽어 현재의 은닉 상태 를 제어**
  - 먼저, 이전 은닉 상태(h_{t-1})과 현재 입력(x_t)을 입력 받아서 셀 상태 C_t 의 어느 부분을 출력으로 내보낼지를 o_t 에 저장하게 됨.
  - 그리고 셀 상태 를 tanh 함수를 거쳐 –1과 1 사이의 값으로 변경시킨 후에 o_t 와 원소별 곱셈을 수행하여 셀 상태 C_t 에서 o_t 에 설정된 특정한 부분만을 출력시키도록 하여 현재의 은닉 상태 h_t 를 결정하게 됨

$$o_t = \sigma(W_{xo}^T \cdot x_t + W_{ho}^T \cdot h_{t-1} + b_o)$$

$$h_t = o_t \otimes \tanh(C_t)$$



7.2.1 LSTM의 구현하기

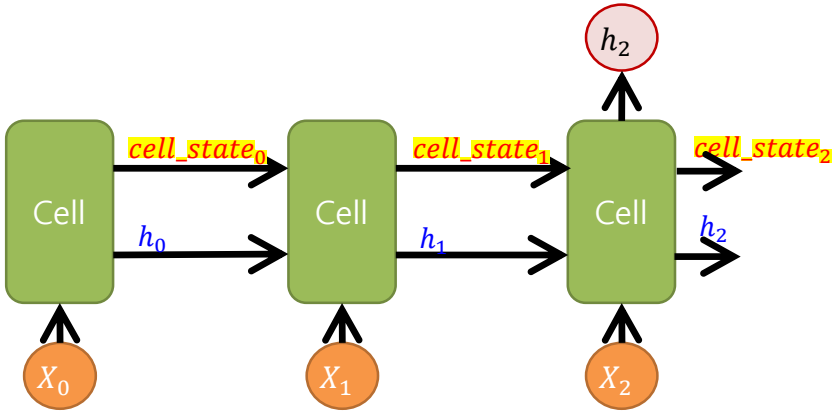
AI Course

- **LSTM 구현**
  - tensorflow.keras.layers.LSTM (node (unit): 64)
  - FC layer (node : 10, activation : softmax)

(연습 7-3)

LSTM에서 셀의 유닛수는 은닉상태와 셀상태 벡터의 크기

```
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.LSTM(units=64,
input_shape = (28, 28)))
model.add(tf.keras.layers.Dense(10, activation='softmax'))
```



return_sequences=False

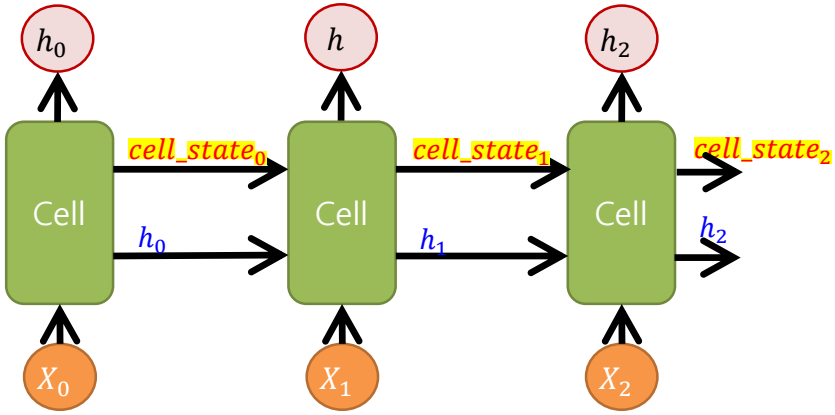
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 64)                23808
_________________________________________________________________
dense (Dense)                (None, 10)                650
=================================================================
Total params: 24,458
Trainable params: 24,458
Non-trainable params: 0
```

7.2.1 LSTM의 구현하기

AI Course

- **LSTM 구현**
  - tensorflow.keras.layers.LSTM (node : 64, return_sequence=True)
  - FC layer (node : 10, activation : softmax)

```
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.LSTM(units=64, return_sequences =
True,input_shape = (28, 28)))
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(10, activation='softmax'))
```



Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 28, 64)	23808
flatten (Flatten)	(None, 1792)	0
dense (Dense)	(None, 10)	17930

Total params: 41,738
Trainable params: 41,738
Non-trainable params: 0

7.2.1 LSTM의 구현하기

AI Course

- LSTM 결과 비교

return_sequences = False

```
Epoch 1/5
1875/1875 [==============================] - 21s 11ms/step - loss: 0.4162 - accuracy: 0.8680
Epoch 2/5
1875/1875 [==============================] - 21s 11ms/step - loss: 0.1403 - accuracy: 0.9581
Epoch 3/5
1875/1875 [==============================] - 20s 11ms/step - loss: 0.0993 - accuracy: 0.9703
Epoch 4/5
1875/1875 [==============================] - 20s 11ms/step - loss: 0.0752 - accuracy: 0.9772
Epoch 5/5
1875/1875 [==============================] - 21s 11ms/step - loss: 0.0619 - accuracy: 0.9812

==========test results==========
313/313 [==============================] - 1s 5ms/step - loss: 0.0730 - accuracy: 0.9772
```

return_sequences = True

```
Epoch 1/5
1875/1875 [==============================] - 19s 10ms/step - loss: 0.2340 - accuracy: 0.9275
Epoch 2/5
1875/1875 [==============================] - 19s 10ms/step - loss: 0.0788 - accuracy: 0.9753
Epoch 3/5
1875/1875 [==============================] - 19s 10ms/step - loss: 0.0542 - accuracy: 0.9830
Epoch 4/5
1875/1875 [==============================] - 19s 10ms/step - loss: 0.0425 - accuracy: 0.9866
Epoch 5/5
1875/1875 [==============================] - 19s 10ms/step - loss: 0.0362 - accuracy: 0.9881

==========test results==========
313/313 [==============================] - 1s 4ms/step - loss: 0.0363 - accuracy: 0.9878
```

7.3 GRU

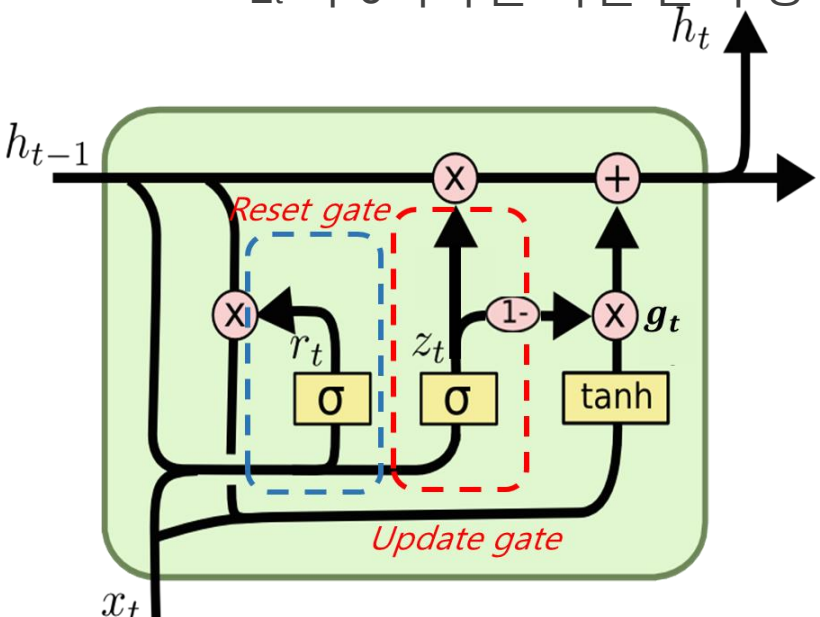
AI Course

- **Gated Recurrent Unit**
  - LSTM의 간소화된 버전
  - LSTM의 cell state를 없애고, 이를 hidden state에서 그 역할을 수행
  - 2개의 게이트로 구성
    - Reset gate(r) : 새로운 입력을 이전 기억과 어떻게 합할지를 정함
    - Update gate(z) : 이전 기억을 얼마만큼 기억할지를 정함

7.3 GRU

AI Course

- **Gated Recurrent Unit**
  - Reset gate(r_t) : 새로운 입력을 이전 기억과 어떻게 합할지를 정함
    - 이전 은닉 상태 h_{t-1} 의 어느 부분을 리셋시켜 출력시킬지를 제어하는 r_t 가 존재
  - Update gate(z_t) : 이전 기억을 얼마만큼 기억할지를 정함
    - LSTM 셀의 삭제 게이트와 입력 게이트의 역할을 모두 담당
    - z_t 가 1이라면 이전 은닉 상태를 유지하게 되고, 현재 스텝의 입력값은 제외
    - z_t 가 0이라면 이전 은닉 상태는 삭제되고, 현재 스텝의 입력값이 저장.



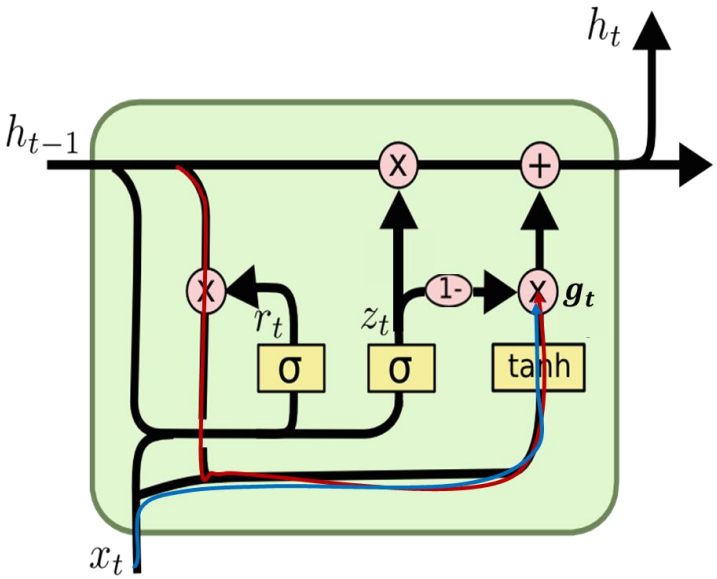
$$r_t = \sigma(W_{xr}^T \cdot x_t + W_{hr}^T \cdot h_{t-1} + b_r)$$

$$z_t = \sigma(W_{xz}^T \cdot x_t + W_{hz}^T \cdot h_{t-1} + b_z)$$

7.3 GRU

AI Course

- **Gated Recurrent Unit**
  - g_t: 후보 (candidate)라는 것으로 현시점의 정보 후보군을 계산
  - 가중치가 곱해진 입력값 ($W_{xg}^T \cdot x_t$) + 이전 은닉상태와 리셋 게이트 rt의 가중치곱 ($W_{hg}^T \cdot (r_t \otimes h_{t-1})$) + bias
    - 과거 은식상태의 정보를 그대로 이용하지 않고 리셋 게이트의 곱을 이용
  - 업데이트 게이트와 후보의 결과를 덧셈하여 t시간의 은닉상태를 계산
  - $z_t \otimes h_{t-1}$ 은 과거시간의 정보양을 결정, $(1-z_t) \otimes g_t$ 은 현재시간의 정보양을 결정



$$g_t = \sigma(W_{xg}^T \cdot x_t + W_{hg}^T \cdot (r_t \otimes h_{t-1}) + b_g)$$

$$h_t = z_t \otimes h_{t-1} + (1-z_t) \otimes g_t$$

7.3.1 GRU 구현하기

AI Course

- **Gated Recurrent Unit**
  - tf.keras.layers.GRU() 함수 사용

(연습 7-4)

```
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.GRU(units=64, input_shape = (28, 28)))
model.add(tf.keras.layers.Dense(10, activation='softmax'))
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
gru (GRU)                    (None, 64)                18048
_________________________________________________________________
dense_1 (Dense)              (None, 10)                650
=================================================================
Total params: 18,698
Trainable params: 18,698
Non-trainable params: 0
```

7.3.1 GRU

AI Course

- **Gated Recurrent Unit 결과**

(연습 7-4)

```
_________________________________________________________________
Epoch 1/5
1875/1875 [==============================] - 19s 10ms/step - loss: 0.4764 - accuracy: 0.8453
Epoch 2/5
1875/1875 [==============================] - 19s 10ms/step - loss: 0.1459 - accuracy: 0.9571
Epoch 3/5
1875/1875 [==============================] - 19s 10ms/step - loss: 0.1016 - accuracy: 0.9692
Epoch 4/5
1875/1875 [==============================] - 19s 10ms/step - loss: 0.0789 - accuracy: 0.9761
Epoch 5/5
1875/1875 [==============================] - 19s 10ms/step - loss: 0.0651 - accuracy: 0.9808

============test results==========
313/313 [==============================] - 1s 4ms/step - loss: 0.0679 - accuracy: 0.9786

 Accuracy: 0.9786
```

RNN vs LSTM vs GRU

AI Course

- 네트워크 사이즈 비교

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
simple_rnn (SimpleRNN)       (None, 64)                5952
_________________________________________________________________
dense (Dense)                (None, 10)                650
=================================================================
Total params: 6,602
Trainable params: 6,602
Non-trainable params: 0
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 64)                23808
_________________________________________________________________
dense (Dense)                (None, 10)                650
=================================================================
Total params: 24,458
Trainable params: 24,458
Non-trainable params: 0
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
gru (GRU)                    (None, 64)                18048
_________________________________________________________________
dense_1 (Dense)              (None, 10)                650
=================================================================
Total params: 18,698
Trainable params: 18,698
Non-trainable params: 0
```

7.4 양방향 (Bidirectional) 추론

AI Course

- **정방향 추론 vs. 양방향 추론**

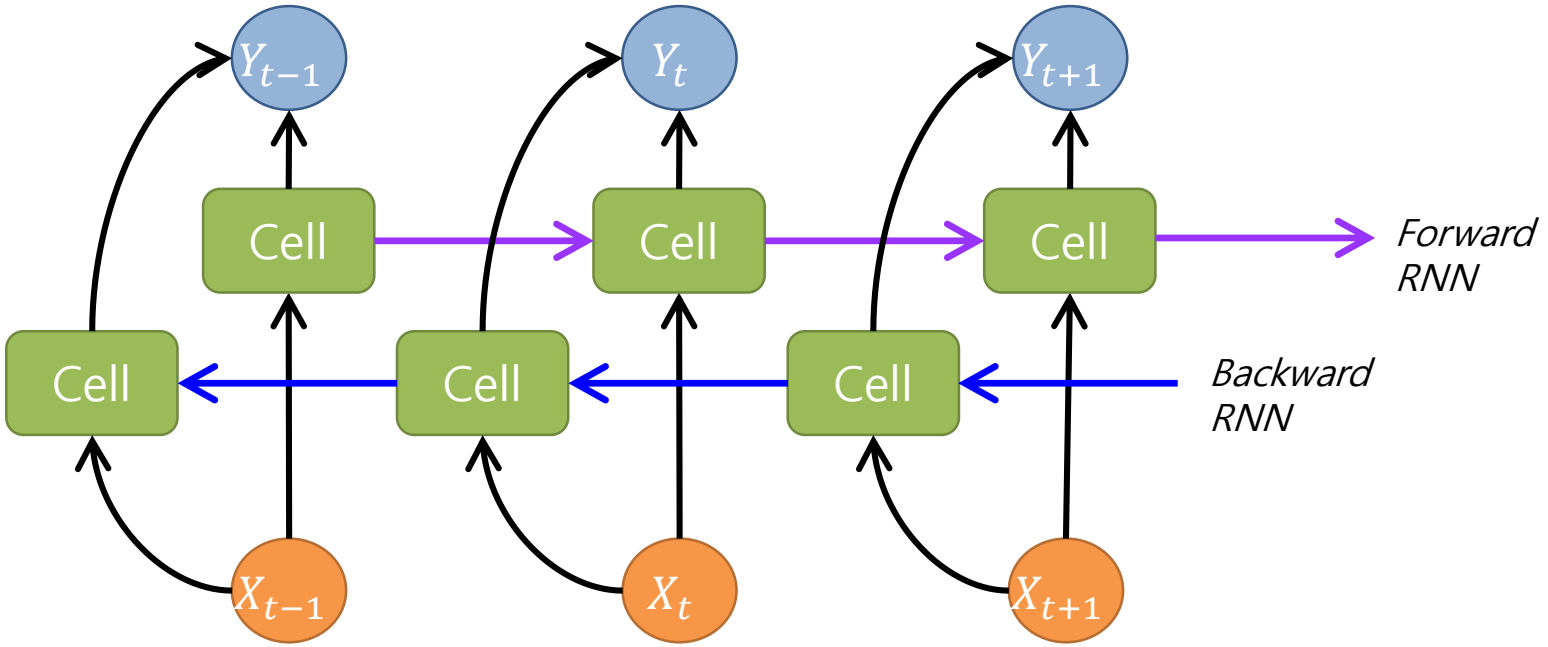
" 그 가방의 _____은 2만원이었다. "

  - 사용되는 데이터의 특성에 따라서 정방향 추론 못지않게 역방향 추론도 유의미한 결과를 낼 수 있음
  - 하지만, 일반적 RNN, LSTM 구조는 오직 정방향 데이터만을 처리함
  - 이를 개선하기 위해서 나온 개념이 **양방향(Bidirectional) 추론**이며, 이를 RNN과 LSTM에 적용할 수 있음

7.4 양방향 (Bidirectional) 추론

AI Course

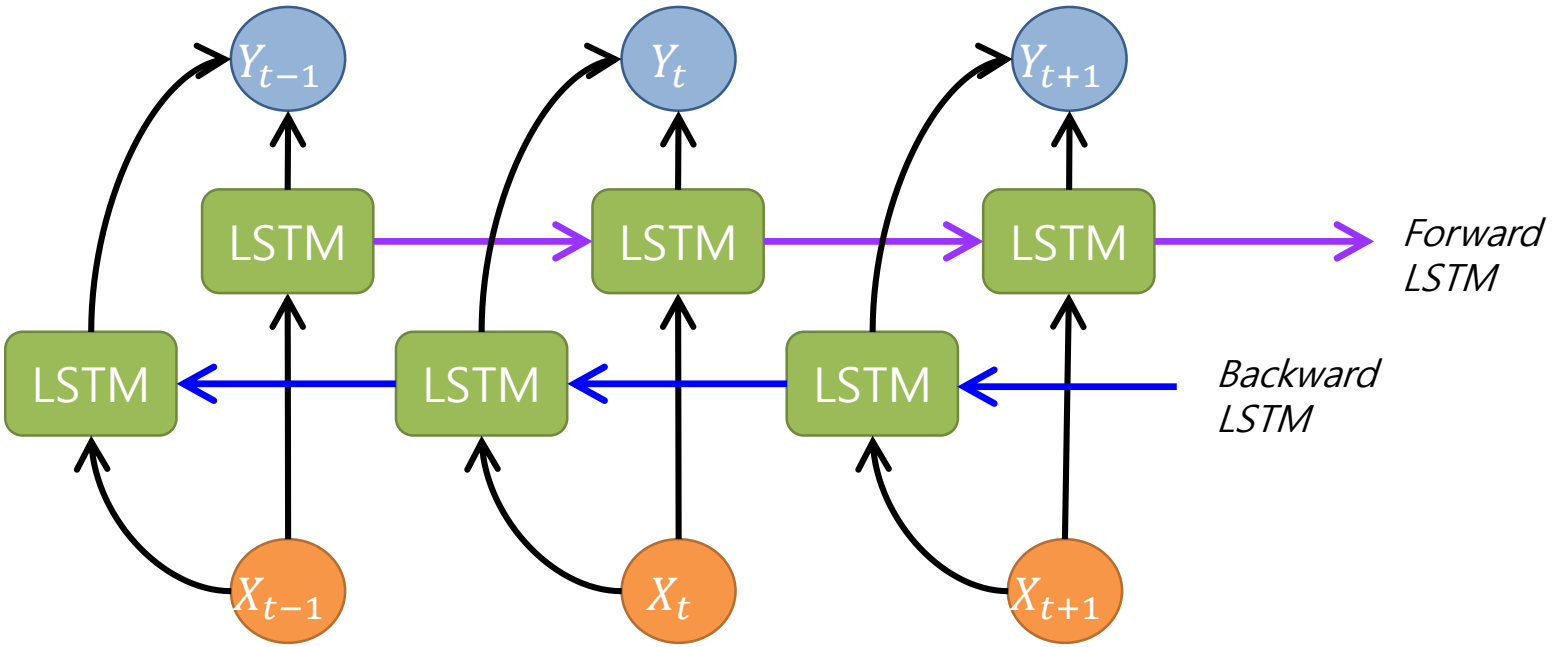
- **Bidirectional 추론**
  - 기존의 RNN, LSTM, GRU는 정방향(Forward)으로의 순서만을 고려했다면, 양방향 RNN은 역방향의 순서도 고려하는 모델임
  - 실제로는 두 개의 모델(정방향, 역방향)을 만들어 학습 결과를 합치는 인공 신경망 모델에 가까움



7.4 양방향 (Bidirectional) 추론

AI Course

- **Bidirectional LSTM**
  - 양뱡향 LSTM은 전방향에 대한 순서도 역방향의 순서도 고려하는 모델



7.4.1 양방향 (Bidirectional) 추론 구현

AI Course

- **Bidirectional RNN 구현**
  - tf.keras.layers.Bidirectional()함수에 RNN이나 LSTM 레이어를 전달하면 됨

from tensorflow.keras.layers import LSTM, Flatten, Dense, Bidirectional

```
model = tf.keras.models.Sequential()
model.add(Bidirectional(SimpleRNN(units=64, return_sequences = True),input_shape = (28, 28)))
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(10, activation='softmax'))
```

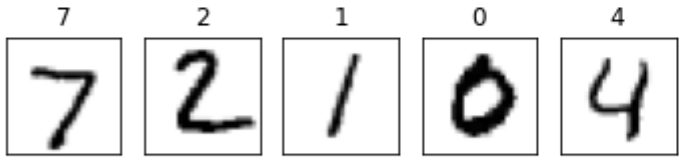
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
bidirectional_3 (Bidirection (None, 28, 128)           11904
_________________________________________________________________
flatten_2 (Flatten)          (None, 3584)              0
_________________________________________________________________
dense_3 (Dense)              (None, 10)                35850
=================================================================
Total params: 47,754
Trainable params: 47,754
Non-trainable params: 0
```

7.4.1 양방향 (Bidirectional) 추론 구현

AI Course

- **Bidirectional RNN 구현**

```
_________________________________________________________________
Epoch 1/5
1875/1875 [==============================] - 19s 10ms/step - loss: 0.2350 - accuracy: 0.9297
Epoch 2/5
1875/1875 [==============================] - 19s 10ms/step - loss: 0.1276 - accuracy: 0.9632
Epoch 3/5
1875/1875 [==============================] - 19s 10ms/step - loss: 0.1054 - accuracy: 0.9698
Epoch 4/5
1875/1875 [==============================] - 20s 11ms/step - loss: 0.0962 - accuracy: 0.9729
Epoch 5/5
1875/1875 [==============================] - 19s 10ms/step - loss: 0.0846 - accuracy: 0.9763

============test results==========
313/313 [==============================] - 1s 4ms/step - loss: 0.1076 - accuracy: 0.9718

 Accuracy: 0.9718
```




```
==================================
```

7.4.1 양방향 (Bidirectional) 추론 구현

AI Course

- **Bidirectional LSTM 구현**
  - Bi-RNN보다는 Bi-LSTM을 많이 사용함
  - tf.keras.layers.Bidirectional()함수에 RNN이나 LSTM 레이어를 전달하면 됨

```
model = tf.keras.models.Sequential()
model.add(Bidirectional(LSTM(units=64, return_sequences = True),
input_shape = (28, 28)))
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(10, activation='softmax'))
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
bidirectional_2 (Bidirection (None, 28, 128)           47616
_________________________________________________________________
flatten_1 (Flatten)          (None, 3584)              0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                35850
=================================================================
Total params: 83,466
Trainable params: 83,466
Non-trainable params: 0
```

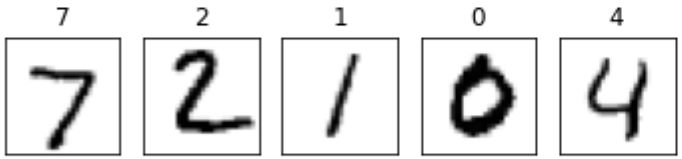
```
==========test results==========
313/313 [==============================] - 2s 8ms/step - loss: 0.0402 - accuracy: 0.9873

 Accuracy: 0.9873
```

7.4.1 양방향 (Bidirectional) 추론 구현

AI Course

- **Bidirectional LSTM 구현**

```
_________________________________________________________________
Epoch 1/5
1875/1875 [==============================] - 36s 19ms/step - loss: 0.2132 - accuracy: 0.9350
Epoch 2/5
1875/1875 [==============================] - 36s 19ms/step - loss: 0.0731 - accuracy: 0.9769
Epoch 3/5
1875/1875 [==============================] - 36s 19ms/step - loss: 0.0527 - accuracy: 0.9834
Epoch 4/5
1875/1875 [==============================] - 36s 19ms/step - loss: 0.0424 - accuracy: 0.9864
Epoch 5/5
1875/1875 [==============================] - 36s 19ms/step - loss: 0.0352 - accuracy: 0.9889

============test results==========
313/313 [==============================] - 2s 8ms/step - loss: 0.0402 - accuracy: 0.9873

 Accuracy: 0.9873
```

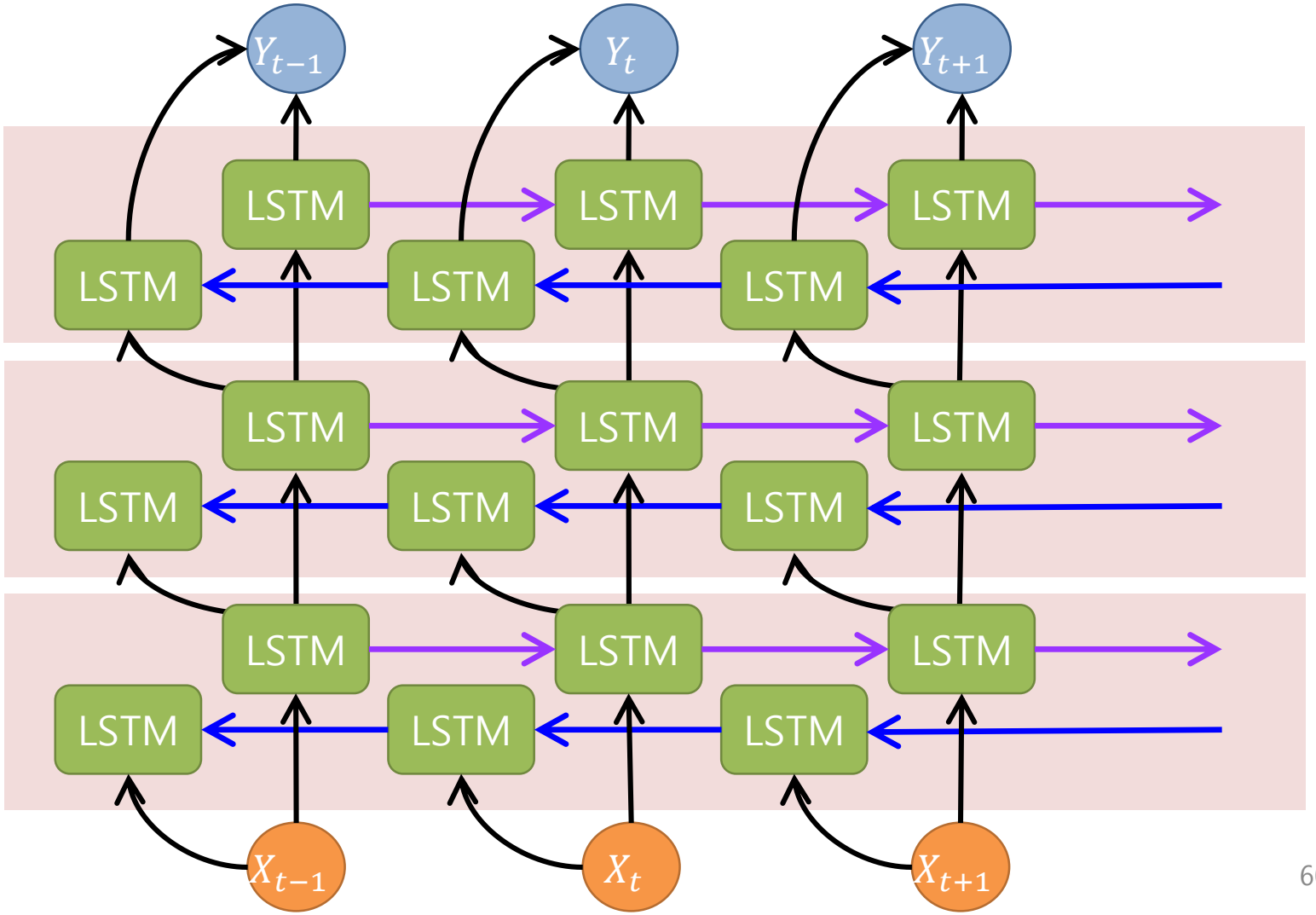



```
================================
```

7.4.1 양방향 (Bidirectional) 추론 구현

AI Course

- **Deep Bidirectional LSTM (Deep Bi-LSTM)**



7.4.1 양방향 (Bidirectional) 추론 구현

AI Course

- **Deep Bi-LSTM 구현 (연습 7-5)**

from tensorflow.keras.layers import LSTM, Flatten, Dense, Bidirectional

```
model = tf.keras.models.Sequential()
model.add(Bidirectional(LSTM(units=64, return_sequences = True),inpu
t_shape = (28, 28)))
model.add(Bidirectional(LSTM(units=64, return_sequences = True)))
model.add(Bidirectional(LSTM(units=64, return_sequences = True)))
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(10, activation='softmax'))
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
bidirectional_4 (Bidirection (None, 28, 128)           47616
_________________________________________________________________
bidirectional_5 (Bidirection (None, 28, 128)           98816
_________________________________________________________________
bidirectional_6 (Bidirection (None, 28, 128)           98816
_________________________________________________________________
flatten_3 (Flatten)          (None, 3584)              0
_________________________________________________________________
dense_4 (Dense)              (None, 10)                35850
=================================================================
```

7.4.1 양방향 (Bidirectional) 추론 구현

AI Course

- **Deep Bi-LSTM 구현**
  - 3개의 Bi-LSTM 레이어를 사용할 경우, One Bi-LSTM에 비해서 시간이 3배 이상으로 소요됨

3 Bi-LSTM

```
------------------------------------------------------------------
Epoch 1/5
1875/1875 [==============================] - 147s 78ms/step - loss: 0.2046 - accuracy: 0.9333
Epoch 2/5
1875/1875 [==============================] - 147s 78ms/step - loss: 0.0649 - accuracy: 0.9801
Epoch 3/5
1875/1875 [==============================] - 147s 78ms/step - loss: 0.0471 - accuracy: 0.9857
Epoch 4/5
1875/1875 [==============================] - 145s 77ms/step - loss: 0.0367 - accuracy: 0.9887
Epoch 5/5
1875/1875 [==============================] - 146s 78ms/step - loss: 0.0288 - accuracy: 0.9910

==========test results==========
313/313 [==============================] - 7s 22ms/step - loss: 0.0379 - accuracy: 0.9894

 Accuracy: 0.9894
```

Single Bi-LSTM

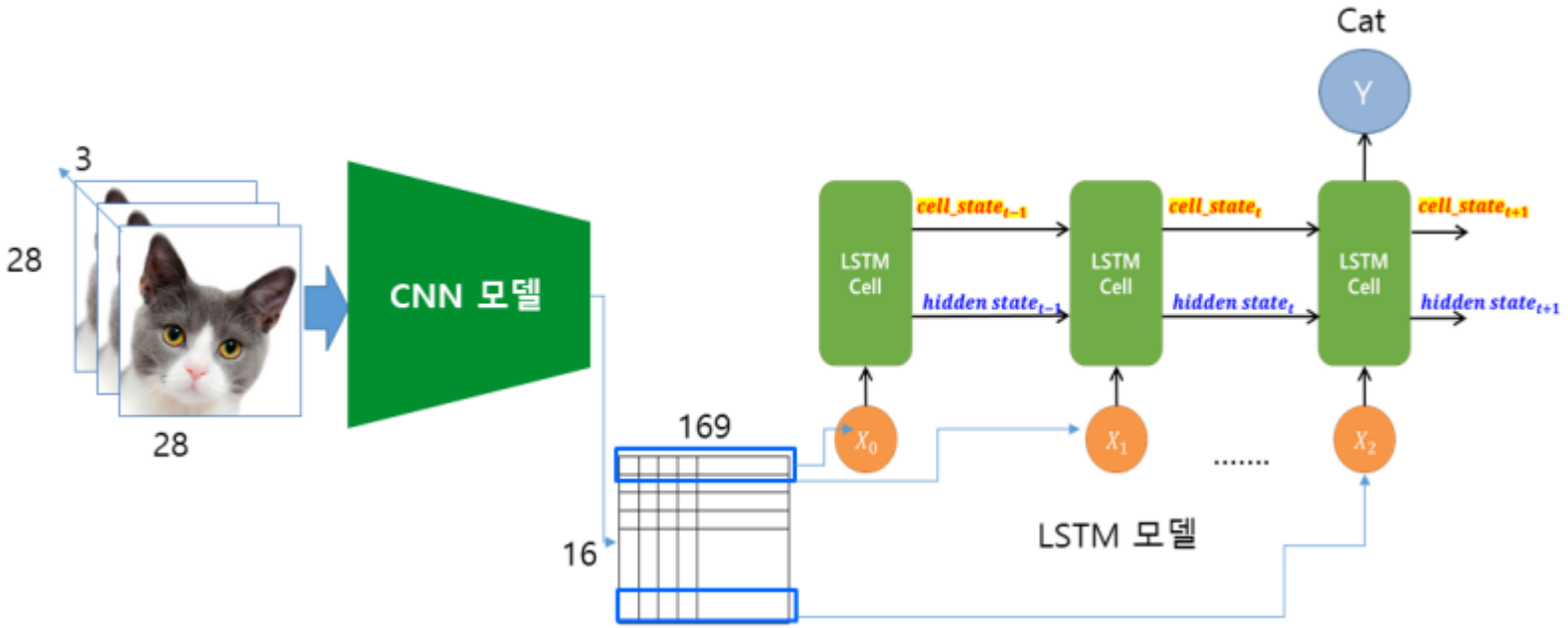
```
Epoch 1/5
1875/1875 [==============================] - 36s 19ms/step -
Epoch 2/5
1875/1875 [==============================] - 36s 19ms/step -
Epoch 3/5
1875/1875 [==============================] - 36s 19ms/step -
Epoch 4/5
1875/1875 [==============================] - 36s 19ms/step -
Epoch 5/5
1875/1875 [==============================] - 36s 19ms/step -
```

```
Accuracy: 0.9873
```

여기서 잠깐~

AI Course

- **RGB 컬러 이미지로 LSTM 학습시키기**
  - RGB 3채널로 구성된 컬러 이미지일 경우는 어떻게 될까?
  - 방법 1) 먼저 RGB 3채널을 1개의 채널로 변경해야 한다. 가장 쉬 운 방법은 RGB 이미지를 모두 평균하여 1개의 gray이미지를 만드는 방법
  - 방법 2) CNN을 통해 특징 맵을 추출하고 이것을 결합하여 RNN 등에 입력➔ 더 좋은 성능을 얻을 수 있다.



여기서 잠깐~

AI Course

- **RGB 컬러 이미지로 LSTM 학습시키기**
  - 예를 들어 cifar10 데이터 셋을 이용한다고 한다면 먼저 간단한 CNN을 구성.

```
# cifar10 데이터셋을 로드하여 준비
(train_images, train_labels), (test_images, test_labels) = tf.keras.da
tasets.cifar10.load_data()

# 픽셀값 0~1로 정규화
train_images = train_images.reshape((50000,32,32,3))
test_images = test_images.reshape((10000,32,32,3))
train_images, test_images = train_images / 255.0, test_images / 255.0
# 데이터 레이블 one hot 코드 변경
one_hot_train_labels = to_categorical(train_labels, 10)
one_hot_test_labels = to_categorical(test_labels, 10)

model = Sequential()
model.add(tf.keras.layers.Conv2D(32, (3, 3), activation='relu', stride
s=(1, 1), input_shape=(32, 32, 3)))
model.add(tf.keras.layers.MaxPooling2D((2, 2)))
model.add(tf.keras.layers.Conv2D(16, (3, 3), activation='relu'))
```

CNN 출력은 13x13크기의 특징맵 16개

여기서 잠깐~

AI Course

- **RGB 컬러 이미지로 LSTM 학습시키기**
  - 예를 들어 cifar10 데이터 셋을 이용한다고 한다면 먼저 간단한 CNN을 구성.

```
model.add(Reshape(target_shape = (16,13*13)))
model.add(LSTM(30, input_shape = (16,13*13), return_sequences = Fal
se))
model.add(Dense(10, activation='softmax'))
model.compile(optimizer=Adam(learning_rate=0.001),
              loss='categorical_crossentropy', metrics=['accuracy'])

model.summary()
history = model.fit(train_images, one_hot_train_labels, epochs=5, b
atch_size=32)
print("\n============test results==========")
labels=model.predict(test_images)
print("\n Accuracy: %.4f" % (model.evaluate(test_images,
                                           one_hot_test_labels)[1]
))
print("=================================")
```

return_sequences= True, Flatten() 이용하여 수정해 보자

실제 적용 사례

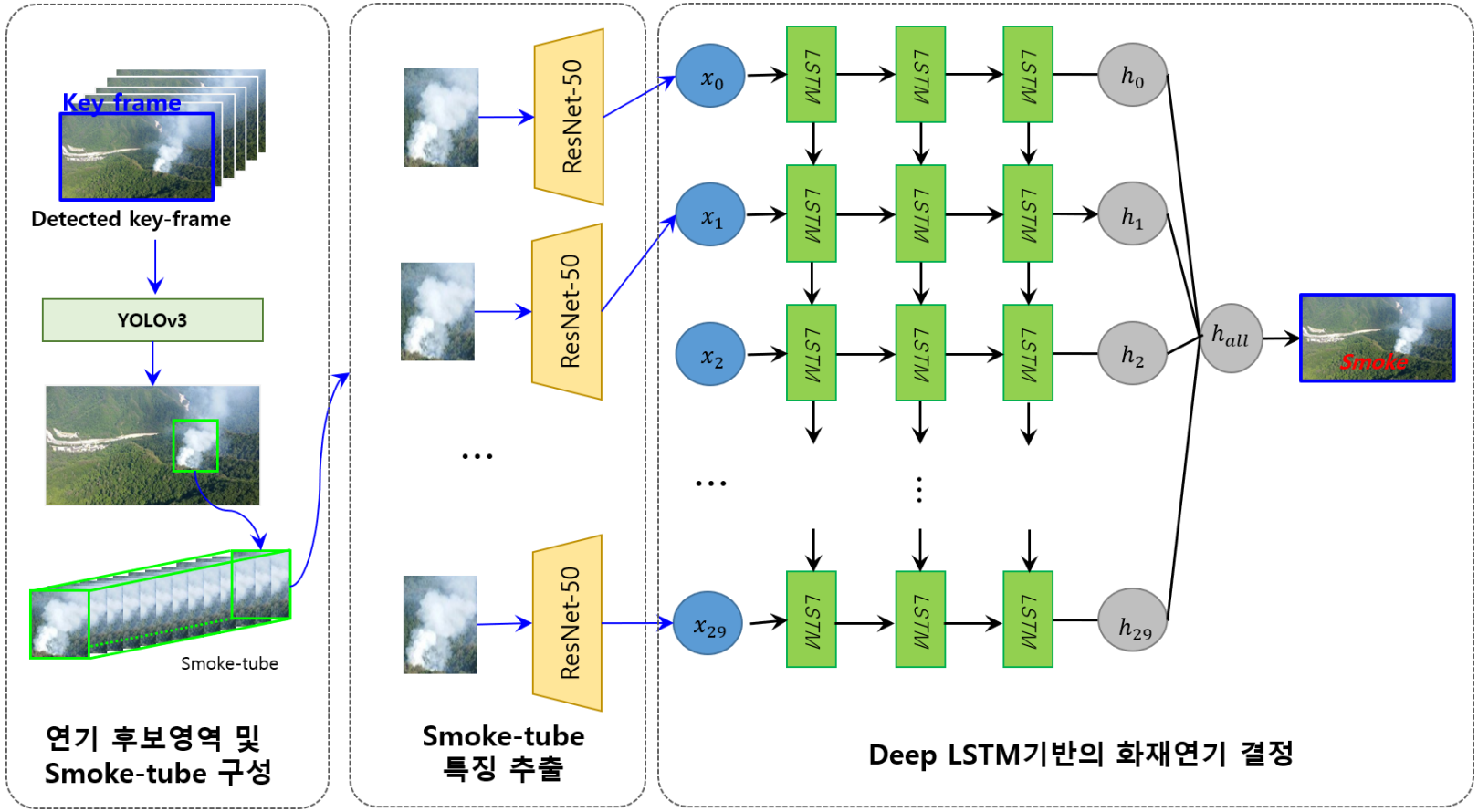
AI Course

- **실제 적용 방법**
  - 실제 RNN 계열의 알고리즘은 정지영상보다는 동영상에서 행동인식, 포즈인식, 표정인식, 객체추적, 화재감시 등 다수의 프레임을 분석하는 방법론에서 많이 적용되고 있음.
  - RNN, LSTM, GRU의 경우 입력으로 1차원 벡터 값을 입력 받으므로 2D 이미지를 입력받지 못함
  - 따라서 이미지 기반의 시퀀스 데이터를 LSTM에 적용할 때는 LSTM네트워크 앞에 또다른 특징추출 네트워크를 추가하여 특징을 1차원 벡터로 추출한 후에 그 추출된 벡터를 LSTM의 입력으로 사용함

실제 적용 사례

AI Course

- 실제 적용 방법

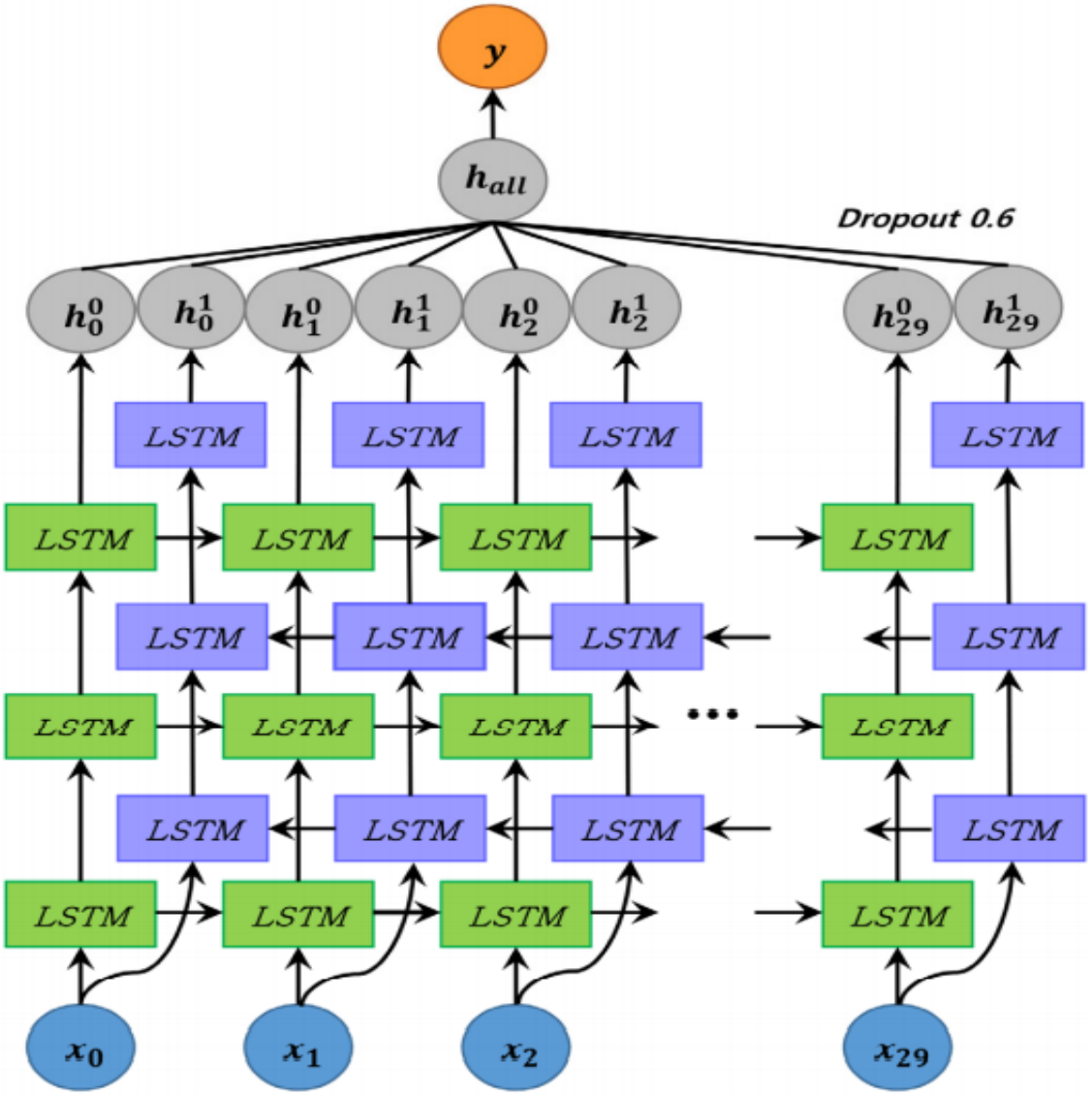


시퀀스 데이터(동영상) 분석을 통한 산불 연기 감지

실제 적용 사례

AI Course

- 산불 연기 검증을 위한 Deep Bidirectional LSTMs 구성도



실제 적용 사례

AI Course

- 표 7-1 6가지 RNN계열 모델에 대한 비교 실험 결과

Method YOLOv3+ResNet50 + (①~⑥)	Precision (%)	Recall (%)	F1-score (%)	TPR (%)	TNR (%)	Proc. Time (s/frame)
① one LSTM	88.85	84.08	86.40	84.08	85.20	0.143
② deep LSTMs	**86.41**	**89.51**	**87.93**	**89.51**	**80.26**	**0.165**
③ one Bi-LSTM	86.92	88.78	87.84	88.78	81.26	0.143
④ deep Bi-LSTM	89.85	84.32	87.00	84.32	86.65	0.155
⑤ one GRU	87.15	88.20	87.67	88.20	81.77	0.143
⑥ deep GRUs	87.46	86.90	87.18	86.90	82.52	0.153

* TPR: True Positive Rate, TNR: True Negative Rate

$$Precision = \frac{TP}{TP+FP},\ Recall = \frac{TP}{TP+FN},\ F1-score = 2\times\frac{Precision\times Recall}{Precision+Recall} \qquad TPR = \frac{TP}{TP+FN},\ TNR = \frac{TN}{TN+FP}$$

정리

AI Course

- 시간적 순서를 가지는 데이터를 시퀀스(sequence) 데이터 또는 시계열(time series) 데이터라고 한다..
- RNN(Recurrent Neural Network)은 현재 입력된 정보와 이전 정보를 함께 고려하여 현재의 결 과를 예측하며, 현재의 정보는 다시 다음 입력에 대한 결과를 예측할 때 사용된다.
- RNN의 은닉층에서는 하나 이상의 순환적 구조를 가지는 계층으로 구성되며, 이 은닉층을 RNN 에서는 메모리 셀(memory cell) 또는 셀(cell)이라고 부른다.
- 텐서플로우의 케라스에서 제공하는 RNN함수는 tensorflow.keras.layers.SimpleRNN으로 현재 시간에서의 입력Xt를 입력하게 되면 이전 은닉 상태 ht-1을 함께 고려하여 셀에서는 은닉 상 태 ht가 출력된다. 따라서, 최종적으로 네트워크에서 클래스에 따른 확률값을 획득하고 싶다면, RNN 뒤에 Fully Connected Layer 등을 추가시켜줘야 한다.
- tensorflow.keras.layers.SimpleRNN 함수의 units 매개변수는 셀에 존재하고 있는 은닉 노드의 수를 나타낸다. 신경망에서 출력값의 차원은 계층(Layer)을 구성하고 있는 노드 수와 동일하므 로 셀에 존재하고 있는 노드 수에 따라서 출력되는 은닉 상태 ht의 차원이 결정된다.
- tensorflow.keras.layers.SimpleRNN 함수에서 매 타임 스텝마다 은닉 상태를 출력시키고 싶다 면 return_sequences 매개변수를 ‘True’로 설정하면 된다.

정리

AI Course

- 고려해야하는 시퀀스 길이가 길어질수록 이전의 정보가 뒤쪽으로 충분히 전달되지 못하는 현상 이 RNN 모델에서 발생되며, 이 문제를 장기 의존성 문제(The problem of Long-term dependencies)라고 부른다.
- 장기 의존성 문제를 개선하기 위해서 장단기 메모리(Long Short-Term Memory, LSTM) 모델링 방법이 소개되었으며, 이 방법은 RNN과 비교하여 단기 기억은 물론 장기 기억을 고려할 수 있어 시퀀스 길이가 긴 과거 데이터를 고려하여 미래의 데이터를 예측할 수 있다.
- LSTM은 RNN의 은닉 상태(hidden state)에 셀 상태(cell state)가 추가된 구조를 가지며, 여기서 셀 상태가 장기 기억을 담당하게 된다.
- 텐서플로우 케라스에서 제공하는 LSTM 함수는 tensorflow.keras.layers.LSTM이다.
- GRU(Gated Recurrent Unit)는 LSTM의 간소화 버전으로 LSTM의 셀 상태를 제거하고 이 역할 을 은닉 상태에서 수행하도록 구성하였다.
- 텐서플로우 케라스에서 제공하는 GRU 함수는 tensorflow.keras.layers.GRU 이다.
- 양방향 추론은 이전의 상태만을 고려하기 보다는 이후의 상태도 함께 고려하여 도 좋은 결과를 도출하는 방법으로 시퀀스 데이터 모델링 방법들의 셀에 대하여 정방형 추론 및 역방향 추론을 결합한 구조를 가진다.

정리

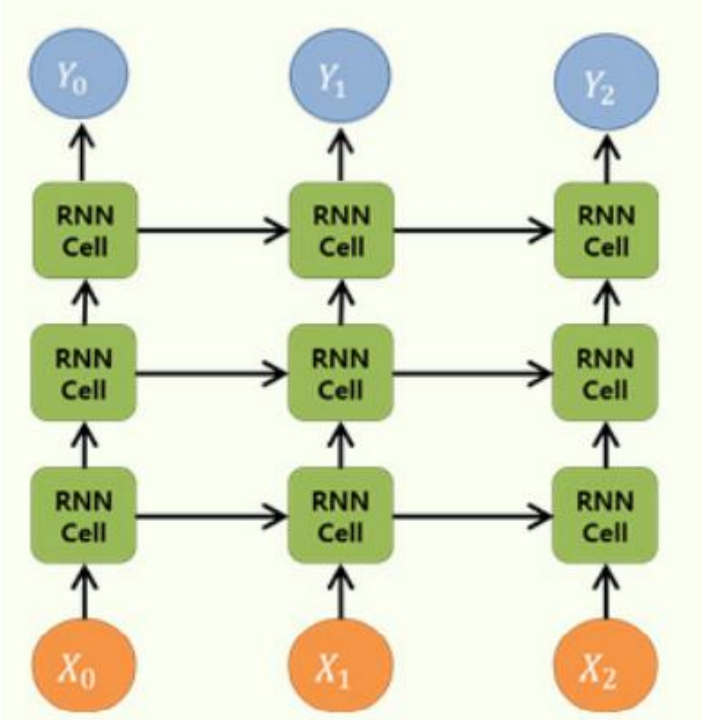
AI Course

- **RNN, LSTM, GRU를 여러 개 구성하여 심층신경망인 Deep RNN, Deep LSTM, Deep GRU를 생 성하여 시퀀스 모델링에 사용할 수도 있다.**

연습문제

AI Course

- 1. 연습 (7 1)에서 Unit 수를 24, 32, 64, 128개 사용하였을 때 성능을 비교하여 보자.
- 2. 아래 그림과 타임 스탬프가 3, 은닉 계층이 3, return_sequences가 True인 심층 RNN을 구성해서 MNIST 데이터에 대해 학습과 테스트를 해보자. 출력층은 32개의 노드와 10개의 노드를 갖는 2개의 층으로 구성하자. 각 계층의 unit 개수는 64, 64, 32로 한다



연습문제

AI Course

- **3. 연습 (7 5)를 3개의 LSTM으로 구성된 Deep LSTM을 사용하여 네트워크를 만들어 학습 및 테스트 를 진행하여 보고, 그 성능을 기존 LSTM 모델과 비교해보자. 또한, tensorflow.keras.layers.LSTM 함수는 드롭아웃 매개변수 'dropout'을 제공하고 있다. Deep LSTM에 드롭아웃을 적용하여 성능 개선이 있는지도 살펴보자.**
  - Hint)

```
model.add(LSTM~GRU(units=512,
        return_sequences=True,
        dropout=0.5,
        input_shape=(None, features_size,)))
```

- **4. 2번 문제에서 양방향 LSTM 네트워크로 구성하여 재학습 및 테스트를 진행하고, 그 성능을 비교 하여 보자.**

연습문제

AI Course

- 5. 연습 (7 4)의 GRU를 3개의 계층으로 늘이고 return_sequence = True로 설정하여 프로그램을 작 성해 보자. MNIST 데이터세트를 이용하여 학습을 하고 테스트 정확률이 이전 방법에 비해 얼마 나 변화하였는지 분석하여 보자.

- 6. 6장 연습문제에서 사용하였던 Fashion_MNIST 데이터세트를 시퀀스 데이터로 변경하여 클래 스로 분류할 수 있도록 모델을 생성하고 학습 및 테스트를 진행해보자. 우리가 배운 세가지 모델 인 RNN, LSTM, GRU을 각각 적용해 보고 어느 모델이 가장 성능이 좋은지 비교해 보자. · 클래스 종류 10개 : 'T shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat', 'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot'.

연습문제

AI Course

- 7. Fashion_MNIST 데이터세트를 이용하여 Bidirectional LSTM, Bidirectional GRU를 구현해 보고 Bidirectional RNN과의 성능을 비교해 보자. 또한 은닉계층을 3개로 늘렸을 경우의 성능도 측정 해보자.

- 8. '여기서 잠깐~!: RGB 컬러 이미지로 LSTM 학습시키기'에서 CNN과 LSTM의 구조를 변화시켜 가면서 성능을 향상시켜 보자. 동시에 RNN, GRU, Bidirectional 방법을 이용하여 성능의 차이를 비교분석해 보자.