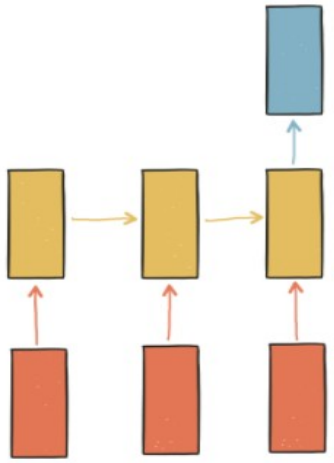
# Session 5

# RNN 알고리즘

# 순환 신경망(Recurrent Neural Network, RNN)

➢ **순환 신경망(RNN·Recurrent Neural Network)**

- 입력과 출력을 시퀀스 단위로 처리하는 시퀀스 모델
- 앞에 나온 데이터를 바탕으로 뒤에 나올 정보를 추론하는데 최적화된 인공지능(AI) 알고리즘
- RNN이 뉴럴 네트워크와 다른 점 : '기억'(다른 말로 hidden state)을 갖고 있다
- 네트워크의 기억은 지금까지의 입력 데이터를 요약한 정보 입니다
- 새로운 입력이 들어 올 때마다 네트워크는 자신의 기억을 조금씩 수정합니다.
- 입력을 모두 처리하고 난 후 네트워크에게 남겨진 기억은 시퀀스 전체를 요약하는 정보가 됩니다.



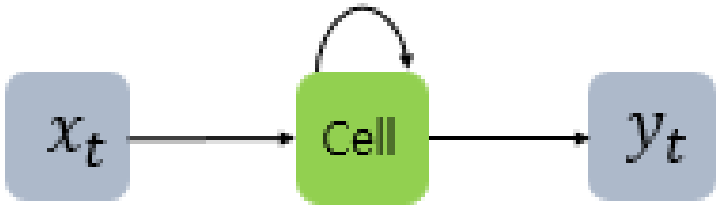
첫번째 입력이 들어오면 첫번째 기억이 만들어집니다.
두번째 입력이 들어오면 기존의 기억과 새로운 입력을 참고하여 새 기억을 만듭니다.
입력의 길이만큼 이 과정을 얼마든지 반복할 수 있습니다.
각각의 기억은 그때까지의 입력을 요약해서 갖고 있는 정보입니다.
RNN은 이 요약된 정보를 바탕으로 출력을 만들어 냅니다.

음악은 음계들의 시퀀스, 동영상은 이미지의 시퀀스, 에세이는 단어들의 시퀀스.
시퀀스의 길이는 가변적
기존의 뉴럴 네트워크 알고리즘은 이미지처럼 고정된 크기의 입력을 다루는 데는 탁월하지만, 가변적인 크기의 데이터를 모델링하기에는 적합하지 않습니다

# 순환 신경망(Recurrent Neural Network, RNN)

## ➢ 순환 신경망(Recurrent Neural Network, RNN)

- 반복적이고 순차적인 데이터(Sequential data)학습에 특화된 인공신경망
- RNN은 은닉층의 노드에서 활성화 함수를 통해 나온 결과값을 출력층 방향으로도 보내는 피드 포워드 신경망(Feed Forward Neural Network)이며, 다시 은닉층 노드의 다음 계산의 입력으로 보내는 내부 순환구조가 들어있다
- 순환구조를 이용하여 과거의 학습을 Weight를 통해 현재 학습에 반영
- 히든 노드가 방향을 가진 엣지로 연결돼 순환구조를 이루는(directed cycle) 인공신경망
- 지속적이고 반복적이며 순차적인 데이터학습의 한계를 해결한 알고리즘
- 시간에 종속
- 시계열 데이터를 활용하므로 자연어(인간 사이에서 쓰는 언어), 주가, 날씨 등 앞뒤가 바뀌면 안 되는 정보를 처리하는데 적합하다



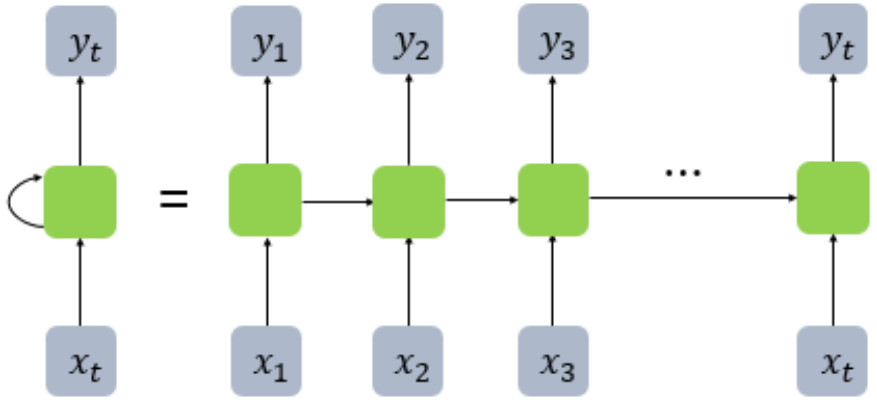

**셀(cell)** : 은닉층에서 활성화 함수를 통해 결과를 내보내는 역할을 하는 노드
메모리 셀 또는 RNN 셀은 이전의 값을 기억하려고 하는 일종의 메모리 역할을 수행

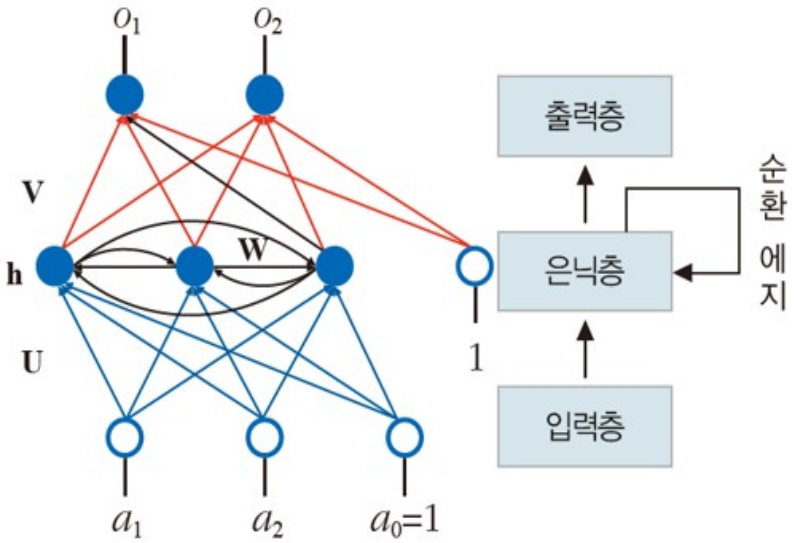
은닉층의 메모리 셀은 각각의 시점(time step)에서 바로 이전 시점에서의 은닉층의 메모리 셀에서 나온 값을 자신의 입력으로 사용하는 재귀적 활동을 합니다.

# 순환 신경망(Recurrent Neural Network, RNN)

## ➢ 순환 신경망(Recurrent Neural Network, RNN) 응용과 한계

- 번역
- 미래 예측(prediction 또는 forecasting) - 내일 주가, 내일 날씨, 기계의 고장, 풍속과 풍향 예측(풍력 발전기의 효율 향상), 농산물 가격/수요량 예측
- 음성 인식에 응용
- 생성 모델 - 이미지/비디오 자막 넣기
- 이미지/음악/춤 생성
- RNN 대화형 AI로 발전할 수 없는 RNN의 한계 : 처음에 입력했던 언어 정보가 뒤로 가면서 사라져 문장을 제대로 이해하기 어렵다



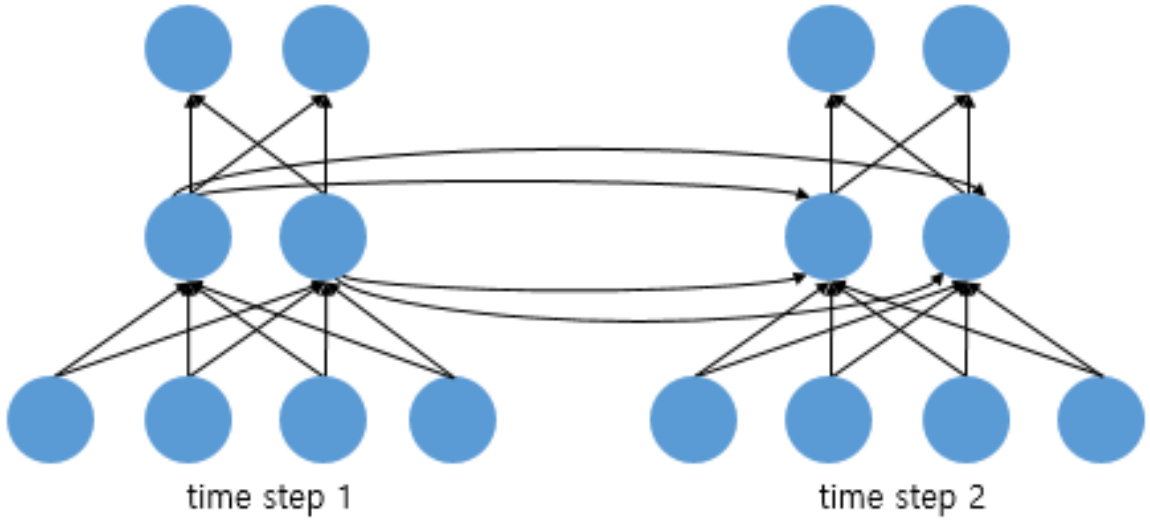




순환 신경망 가중치 집합 {U, V, W}

# 순환 신경망(Recurrent Neural Network, RNN)

## ➢ 순환 신경망(Recurrent Neural Network, RNN)

- 입력 , RNN Cell의 순환 구조, 출력 등으로 구성
- RNN에서는 입력층과 출력층에서는 각각 입력 벡터와 출력 벡터, 은닉층에서는 **은닉 상태**라는 표현을 사용
- 출력은 h0, h1, h2, ...ht 등 은닉상태(hidden state)의 출력과 다음 cell로의 출력 두 가지 경로(path)가 존재
- RNN의 내부는 이전 Cell의 출력과 현재 Cell 입력을 tanh 활성화 함수로 처리한 후 현재 Cell의 출력과 다음 Cell의 입력으로 출력

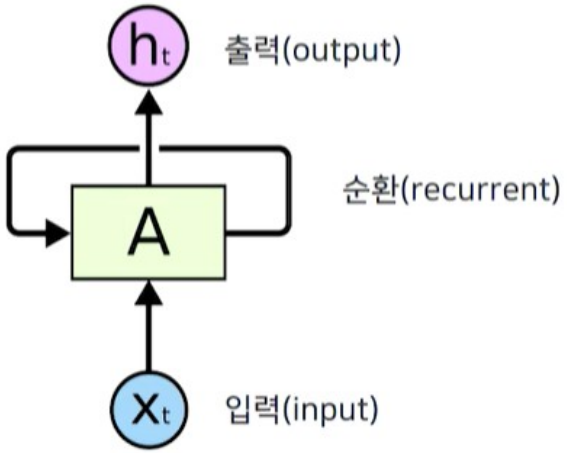


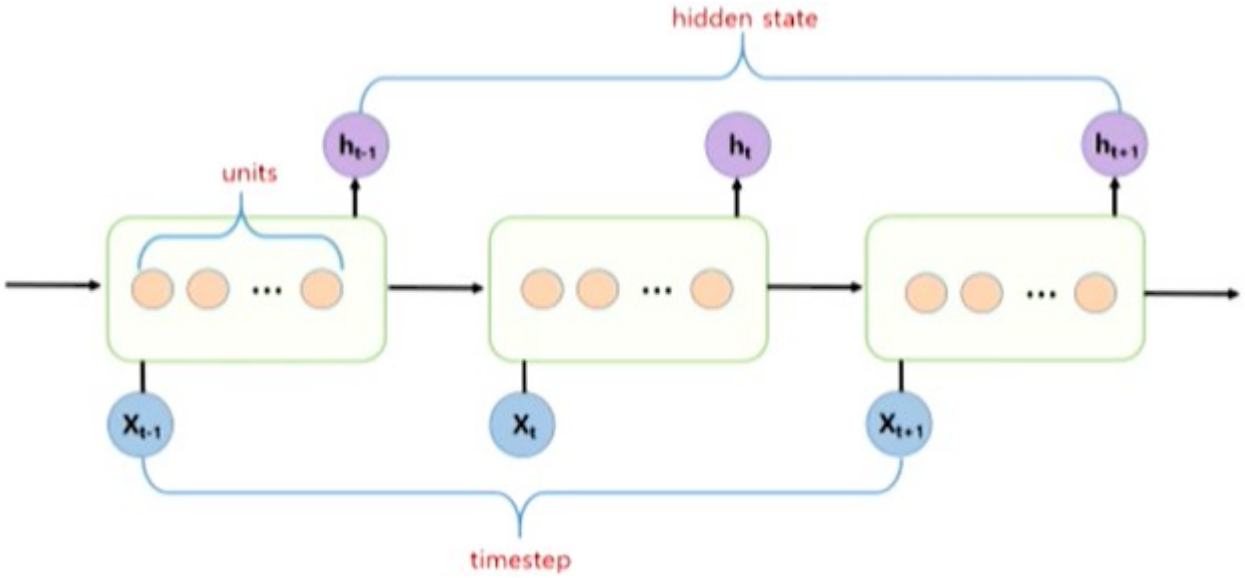

뉴런 단위로 RNN이 시점 2일때의 모습을 시각화 :
입력층의 뉴런 수는 4, 은닉층의 뉴런 수는 2, 출력층의 뉴런 수는 2

# 순환 신경망(Recurrent Neural Network, RNN)

## ➢ 순환 신경망(Recurrent Neural Network, RNN)

- Timestep : RNN 입력데이터로 전달 되는 시퀀스(Sequence)
- (hidden) units : RNN Cell 내의 hidden 노드 수
- Hidden state : 각 timestep의 입력에 대한 RNN Cell 출력

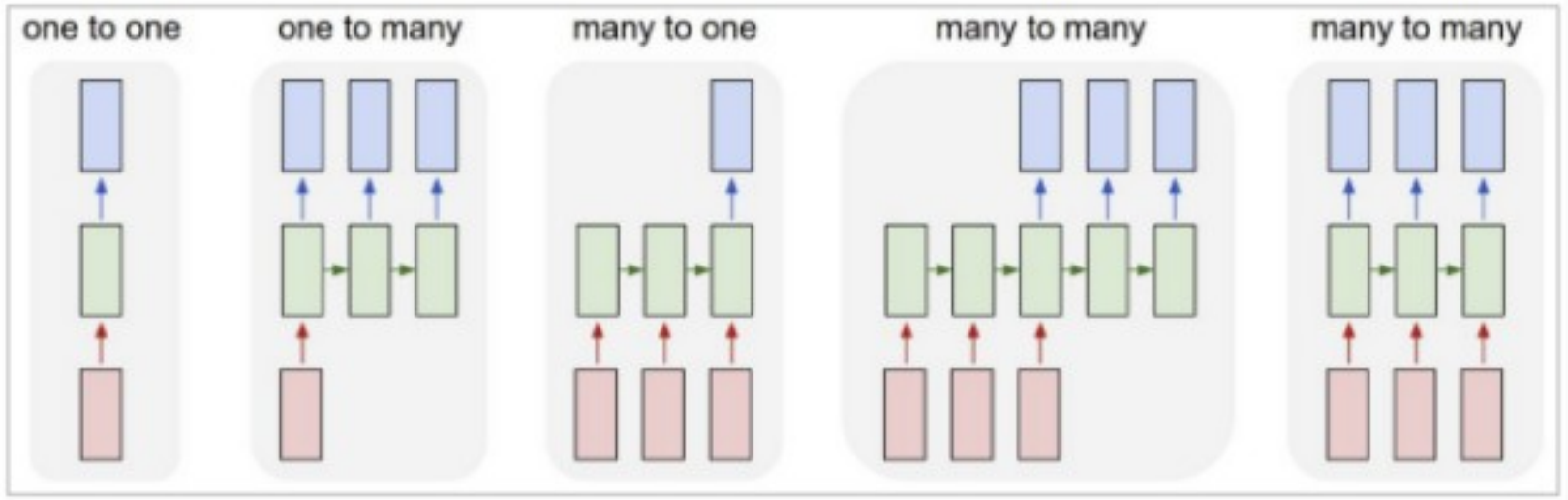


# 순환 신경망(Recurrent Neural Network, RNN)

## ➢ RNN의 형태

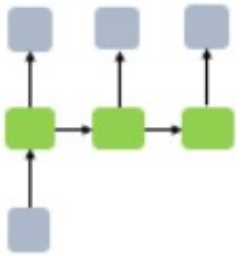
- RNN은 입력과 출력의 길이를 다르게 설계 할 수 있으므로 다양한 용도로 사용할 수 있습니다.




1. 고정크기 입력 & 고정크기 출력. 순환적인 부분이 없기 때문에 RNN이 아닙니다.
2. 고정크기 입력 & 시퀀스 출력. 예) 이미지를 입력해서 이미지에 대한 설명을 문장으로 출력하는 이미지 캡션 생성
3. 시퀀스 입력 & 고정크기 출력. 예) 문장을 입력해서 긍부정 정도를 출력하는 감성 분석기
4. 시퀀스 입력 & 시퀀스 출력. 예) 영어를 한국으로 번역하는 자동 번역기
5. 동기화된 시퀀스 입력 & 시퀀스 출력. 예) 문장에서 다음에 나올 단어를 예측하는 언어 모델

# 순환 신경망(Recurrent Neural Network, RNN)

- **RNN의 형태**
  - **일 대 다(one-to-many) 구조의 모델** : 하나의 이미지 입력에 대해서 사진의 제목을 출력하는 이미지 캡셔닝(Image Captioning) 작업에 사용할 수 있습니다. 사진의 제목은 단어들의 나열이므로 시퀀스 출력입니다.



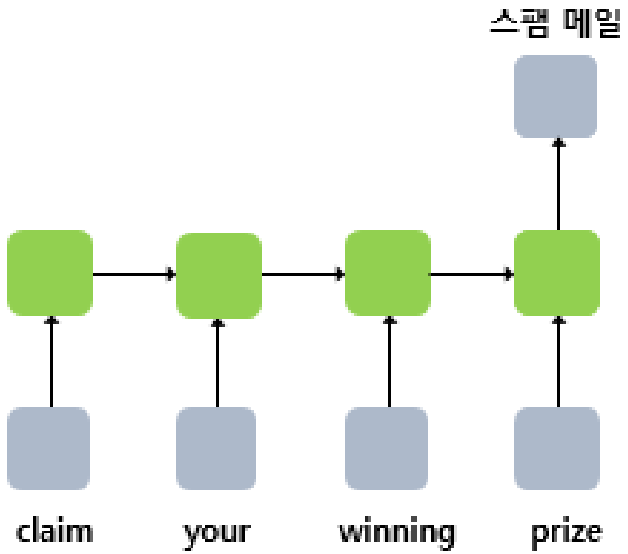
일 대 다(one-to-many)



이미지를 처리하는 CNN(Convolutional Neural Network)과 RNN을 결합하여 이미지를 텍스트로 설명해주는 모델을 만드는 것이 가능합니다.
고정크기 입력 & 시퀀스 출력에 해당하는 구조
이미지라는 고정된 크기의 입력을 받아서 몇 단어로 표현될지 모를 가변적인 길이의 문장을 만들어냄

# 순환 신경망(Recurrent Neural Network, RNN)

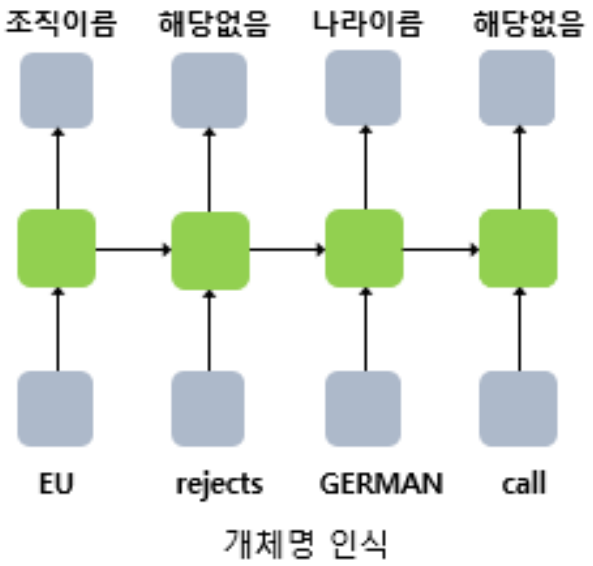
- **RNN의 형태**
  - **다 대 일(many-to-one) 구조의 모델** : 단어 시퀀스에 대해서 하나의 출력
    입력 문서가 긍정적인지 부정적인지를 판별하는 감성 분류(sentiment classification)
    메일이 정상 메일인지 스팸 메일인지 판별하는 스팸 메일 분류(spam detection) 등에 사용

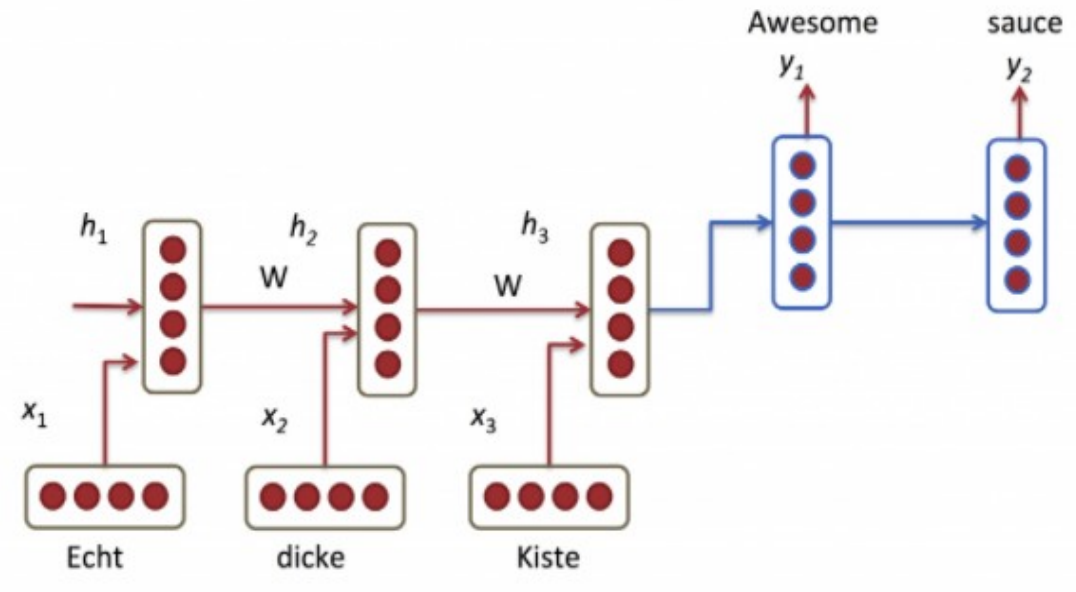


스팸 메일 분류

# 순환 신경망(Recurrent Neural Network, RNN)

- **RNN의 형태**
  - **다 대 다(many-to-many)구조의 모델** : 사용자가 문장을 입력하면 대답 문장을 출력하는 챗봇
    입력 문장으로부터 번역된 문장을 출력하는 번역기
    개체명 인식이나 품사 태깅과 같은 작업

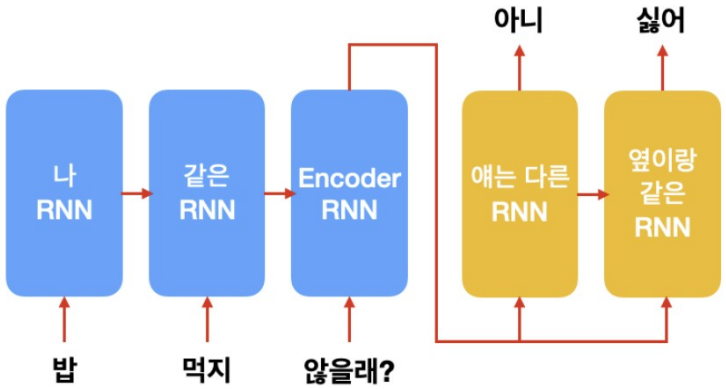



구글의 번역기와 네이버의 파파고는 RNN을 응용한 모델
RNN 기반 모델은 기존 통계 기반 모델의 비해 우수한 성능을 낸다
시퀀스 입력 & 시퀀스 출력 구조 (encoder-decoder 모델)

# 순환 신경망(Recurrent Neural Network, RNN)

- **RNN의 형태**
  - **다 대 다(many-to-many)구조의 모델** : 사용자가 문장을 입력하면 대답 문장을 출력하는 챗봇
    입력 문장으로부터 번역된 문장을 출력하는 번역기
    개체명 인식이나 품사 태깅과 같은 작업




**이름:** 이루다

**나이:** 20

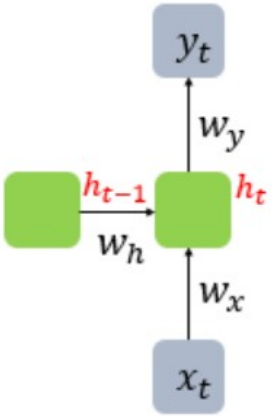
**특징:** 인공지능✨

**좋아하는 가수:** 블랙핑크

**취미:** 일상의 작은 부분을 사진과 글로 기록하기

# 순환 신경망(Recurrent Neural Network, RNN)

## ➢ RNN (Recurrent neural network) 학습 과정




현재 시점 t에서의 은닉 상태값 ht 을 라고 정의
은닉층의 메모리 셀은 ht를 계산하기 위해서 총 두 개의 가중치를 가집니다.
입력층을 위한 가중치 Wx
이전 시점 ht-1의 은닉 상태값인 을 위한 가중치 Wh

은닉층 : $h_t = tanh(W_x x_t + W_h h_{t-1} + b)$

출력층 : $y_t = f(W_y h_t + b)$

$$\tanh\left(\begin{bmatrix} & W_h & \\ D_h \times D_h \end{bmatrix} \times \begin{bmatrix} h_{t-1} \\ D_h \times 1 \end{bmatrix} + \begin{bmatrix} & W_x & \\ D_h \times d \end{bmatrix} \times \begin{bmatrix} x_t \\ d \times 1 \end{bmatrix} + \begin{bmatrix} b \\ D_h \times 1 \end{bmatrix}\right) = \begin{bmatrix} h_t \\ D_h \times 1 \end{bmatrix}$$

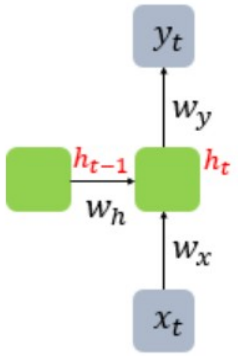
ht를 계산하기 위한 활성화 함수로는 주로 하이퍼볼릭탄젠트 함수(tanh)가 사용됩니다.
은닉층이 2개 이상일 경우에는 각 은닉층에서의 가중치는 서로 다릅니다.

# 순환 신경망(Recurrent Neural Network, RNN)

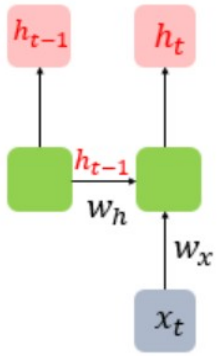
## ➢ 케라스(Keras)로 RNN 구현

- RNN 층은 (**batch_size, timesteps, input_dim**) 크기의 **3D 텐서를 입력**으로 받습니다.
- hidden_units = 은닉 상태의 크기를 정의. 메모리 셀이 다음 시점의 메모리 셀과 출력층으로 보내는 값의 크기(output_dim)와도 동일. RNN의 용량(capacity)을 늘린다고 보면 되며, 중소형 모델의 경우 보통 128, 256, 512, 1024 등의 값을 가진다.
- timesteps = 입력 시퀀스의 길이(input_length)라고 표현하기도 함. 시점의 수.
- input_dim = 입력의 크기.

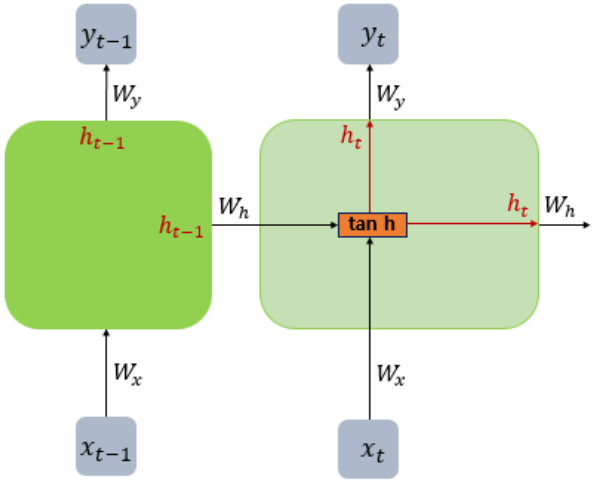
```
from tensorflow.keras.layers import SimpleRNN

model.add(SimpleRNN(hidden_units))
# 추가 인자를 사용할 때
model.add(SimpleRNN(hidden_units, input_shape=(timesteps, input_dim)))
# 다른 표기
model.add(SimpleRNN(hidden_units, input_length=M, input_dim=N))
```




출력층을 포함한 그림



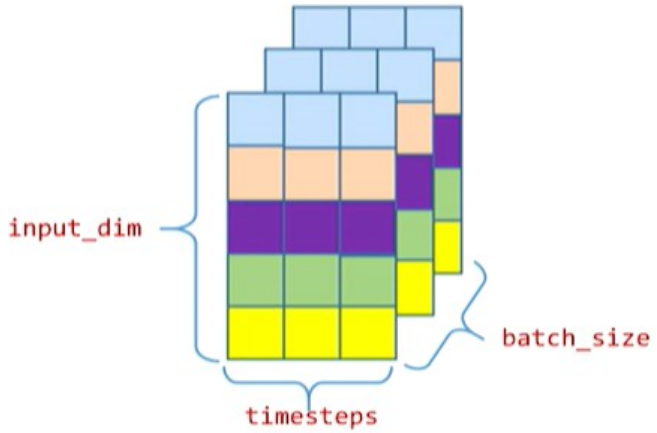

입력층과 은닉층만을 표현한 그림



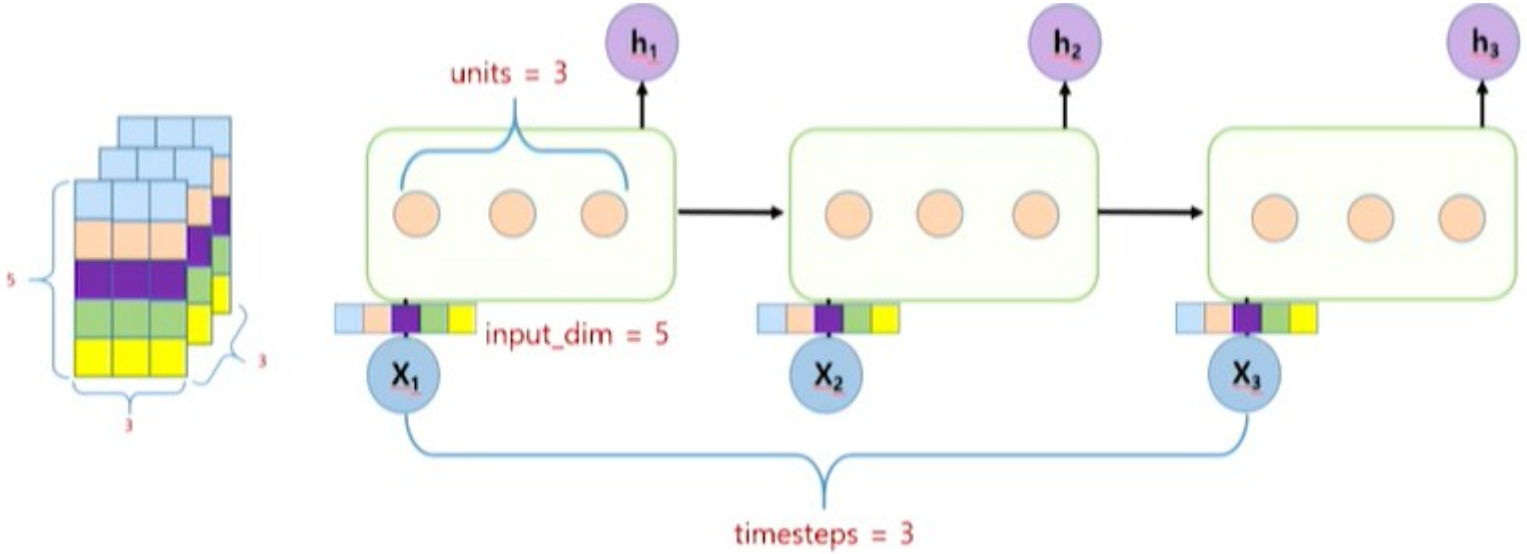

$$h_t = tanh(W_x x_t + W_h h_{t-1} + b)$$

# 순환 신경망(Recurrent Neural Network, RNN)

- **케라스(Keras)로 RNN 구현**
  - RNN 입력 데이터는 batch size, time steps, input dimension 등으로 구성



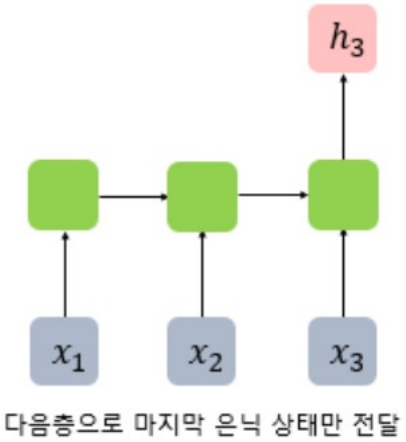

```
model.add(SimpleRNN(units=3, input_shape=(3, 5)))
```



# 순환 신경망(Recurrent Neural Network, RNN)

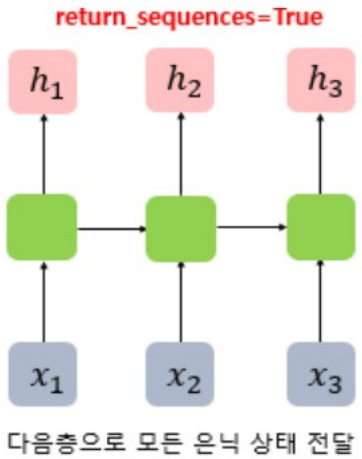
## ➢ 케라스(Keras)로 RNN 구현

- RNN 은닉 상태를 출력 :
  메모리 셀의 최종 시점의 은닉 상태만을 리턴하고자 한다면 (batch_size, output_dim) 크기의 2D 텐서를 리턴합니다.
  메모리 셀의 각 시점(time step)의 은닉 상태값들을 모아서 전체 시퀀스를 리턴하고자 한다면 (batch_size, timesteps, output_dim) 크기의 3D 텐서를 리턴합니다.
- 첫번째 은닉층은 다음 은닉층이 존재하므로 return_sequences = True를 설정하여 모든 시점에 대해서 은닉 상태 값을 다음 은닉층으로 보낼수 있습니다



다음층으로 마지막 은닉 상태만 전달

time step=3일 때
return_sequences = False
다 대 일(many-to-one) 문제
마지막 시점(time step)의 은닉 상태값만 출력



다음층으로 모든 은닉 상태 전달

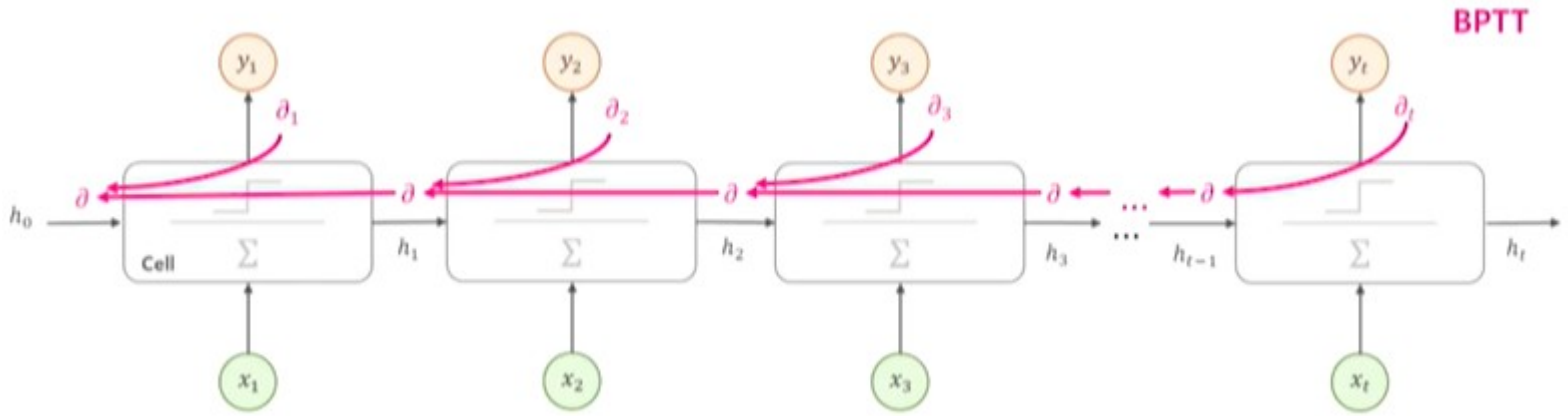
time step=3일 때
return_sequences = True
다 대 다(many-to-many) 문제 해결
RNN셀의 모든 시점(time step)에 대해 은닉 상태값을 출력

```
model = Sequential()
model.add(SimpleRNN(3, batch_input_shape=(8,2,10), return_sequences=True))
model.summary()
```

# 순환 신경망(Recurrent Neural Network, RNN)

## ➢ 기본(vanilla) RNN 의 문제점

- RNN에서의 역전파 BPTT(Back Propagation Through Time)는 timestep의 마지막단계에서 시작단계까지 역전파가 진행되는 과정에서 Gradient Vanishing 발생 가능

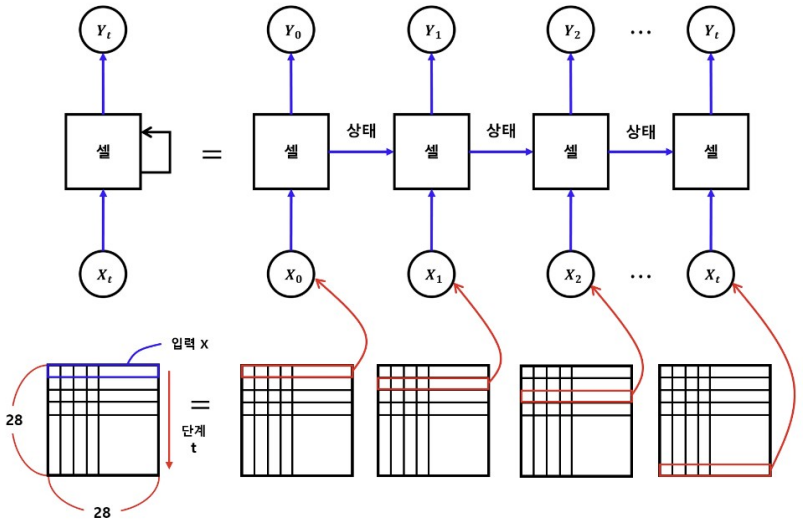



- 장기 의존성 무제 (the problem of Long-Term Dependencies) – 관련 정보와 그 정보가 필요한 곳의 거리가 먼 , 장기 memory에 대한 학습 능력 저하

# 순환 신경망(Recurrent Neural Network, RNN)

## ➢ MNIST를 RNN으로 학습

- RNN은 셀(신경망)을 여러 개 중첩하여 심층 신경망을 만듬
- 학습 데이터를 단계별로 구분하여 입력해야 함
- MNIST의 입력값도 단계별로 입력할 수 있는 형태로 변경해줘야 합니다
- MNIST 데이터가 가로·세로 28x28 크기
  → 가로 한 줄의 28픽셀을 한 단계의 입력으로 처리하여
- 세로줄이 총 28개이므로 28단계를 거쳐 데이터를 입력받는 개념




```
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation, SimpleRNN
from tensorflow.keras.utils import to_categorical, plot_model
from tensorflow.keras.datasets import mnist

(x_train, y_train), (x_test, y_test) = mnist.load_data()
num_labels = len(np.unique(y_train))

y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
```

# 순환 신경망(Recurrent Neural Network, RNN)

- **MNIST를 RNN으로 학습**

```
image_size = x_train.shape[1]
x_train = np.reshape(x_train,[-1, image_size, image_size])     ## resize and normalize
x_test = np.reshape(x_test,[-1, image_size, image_size])
x_train = x_train.astype('float32') / 255
x_test = x_test.astype('float32') / 255

input_shape = (image_size, image_size)    # network parameters
batch_size = 128
units = 256
dropout = 0.2

#모델은 256개의 유닛이 있는 RNN이고, 입력은 28-dim 벡터 28 입니다
model = Sequential()
model.add(SimpleRNN(units=units,
                    dropout=dropout,
                    input_shape=input_shape))
model.add(Dense(num_labels))
model.add(Activation('softmax'))
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
simple_rnn (SimpleRNN)       (None, 256)               72960
_________________________________________________________________
dense (Dense)                (None, 10)                2570
_________________________________________________________________
activation (Activation)      (None, 10)                0
=================================================================
Total params: 75,530
Trainable params: 75,530
Non-trainable params: 0
```

# 순환 신경망(Recurrent Neural Network, RNN)

- **MNIST를 RNN으로 학습**

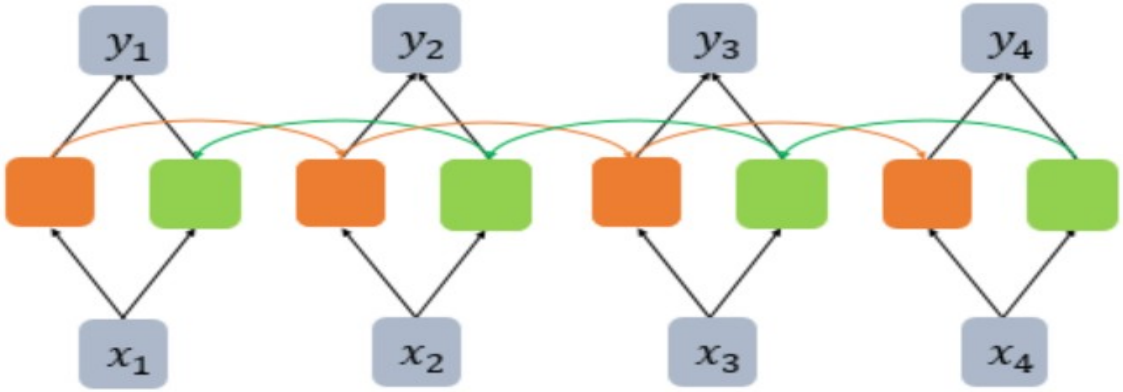
```
model.compile(loss='categorical_crossentropy',
              optimizer='sgd',
              metrics=['accuracy'])
model.fit(x_train, y_train, epochs=20, batch_size=batch_size)

_, acc = model.evaluate(x_test,
                        y_test,
                        batch_size=batch_size,
                        verbose=0)
print("₩nTest accuracy: %.1f%%" % (100.0 * acc))
```

```
Epoch 1/20
469/469 [==============================] - 25s 50ms/step - loss: 0.7451 - accuracy: 0.7867
Epoch 2/20
469/469 [==============================] - 23s 49ms/step - loss: 0.3257 - accuracy: 0.9057
Epoch 3/20
469/469 [==============================] - 23s 49ms/step - loss: 0.2407 - accuracy: 0.9281

Epoch 19/20
469/469 [==============================] - 23s 50ms/step - loss: 0.0750 - accuracy: 0.9770
Epoch 20/20
469/469 [==============================] - 23s 49ms/step - loss: 0.0708 - accuracy: 0.9780

Test accuracy: 97.5%
```

# 순환 신경망(Recurrent Neural Network, RNN)

## ➢ 양방향 순환 신경망(Bidirectional Recurrent Neural Network)

- RNN이 풀고자 하는 문제 중에서는 과거 시점의 입력 뿐만 아니라 미래 시점의 입력에 힌트가 있는 경우도 많습니다.
- 시점 t에서의 출력값을 예측할 때 이전 시점의 입력뿐만 아니라, 이후 시점의 입력 또한  예측을 더욱 정확하게 할 수 있도록 고안된 것이 양방향 RNN입니다.
- 양방향 RNN은 하나의 출력값을 예측하기 위해 기본적으로 두 개의 메모리 셀을 사용합니다.
- 첫번째 메모리 셀은 앞에서 배운 것처럼 앞 시점의 은닉 상태(Forward States) 를 전달받아 현재의 은닉 상태를 계산합니다.
- 두번째 메모리 셀은 뒤 시점의 은닉 상태(Backward States) 를 전달 받아 현재의 은닉 상태를 계산합니다. 입력 시퀀스를 반대 방향으로 읽는 것입니다.



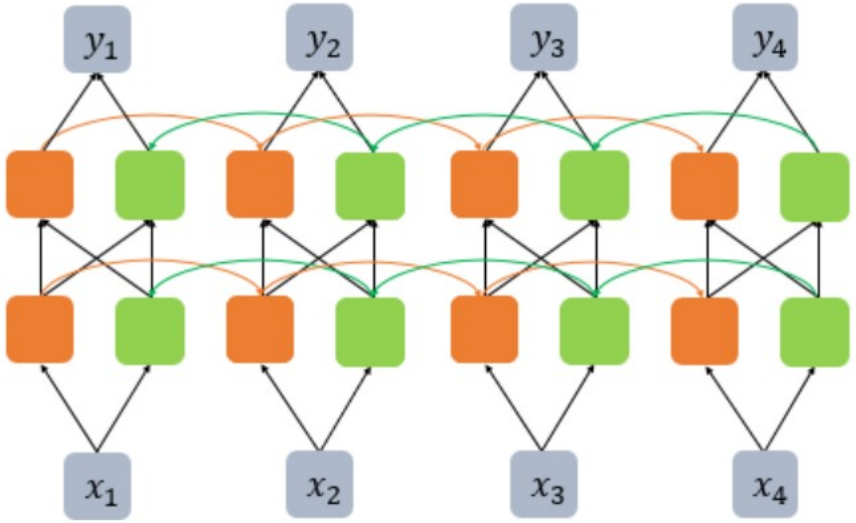

```
from tensorflow.keras.layers import Bidirectional
timesteps = 10
input_dim = 5
model = Sequential()
model.add(Bidirectional(SimpleRNN(hidden_units, return_sequences=True), input_shape=(timesteps, input_dim))
```

# 순환 신경망(Recurrent Neural Network, RNN)

## ➢ 양방향 순환 신경망(Bidirectional Recurrent Neural Network)

- 양방향 RNN도 다수의 은닉층을 가질 수 있습니다.

```
model = Sequential()
model.add(Bidirectional(SimpleRNN(hidden_units, return_sequences=True), input_shape=(timesteps, input_dim)))
model.add(Bidirectional(SimpleRNN(hidden_units, return_sequences=True)))
model.add(Bidirectional(SimpleRNN(hidden_units, return_sequences=True)))
model.add(Bidirectional(SimpleRNN(hidden_units, return_sequences=True)))
```

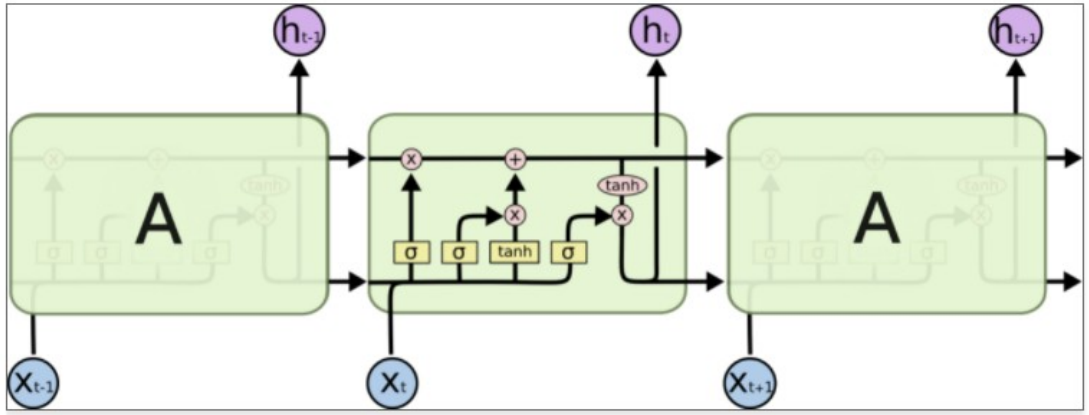


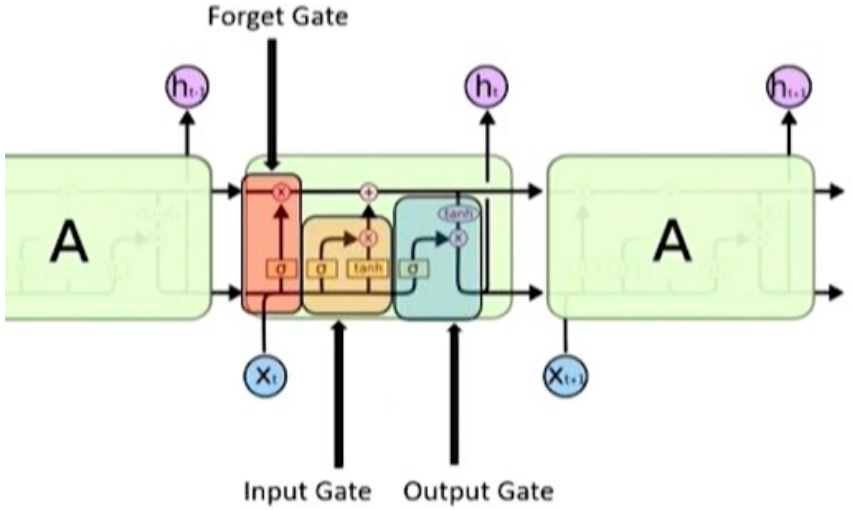

은닉층이 2개인 깊은(deep) 양방향 순환 신경망

# 순환 신경망(Recurrent Neural Network, RNN)

## ➢ LSTM (Long Short Term Memory)

- RNN이 출력과 먼 위치에 있는 정보를 기억할 수 없다는 장기 의존성 문제(the problem of Long-Term Dependencies) 단점을 보완하여 장/단기 기억을 가능하게 설계한 신경망의 구조
- RNN과 같은 체인 구조로 되어 있지만, 반복 모듈은 단순한 한 개의 tanh layer가 아닌 4개의 layer가 서로 정보를 주고받는 구조
- LSTM은 은닉층의 메모리 셀에 입력 게이트, 망각 게이트, 출력 게이트를 추가하여 불필요한 기억을 지우고, 기억해야할 것들을 정합니다.
- LSTM 셀에서는 상태(state)가 크게 두 개의 벡터로 나누어집니다. ( ht를 단기 상태(short-term state), ct를 장기 상태 (long-term state))
- 기억의 유지와 망각, 기억의 선택과 집중

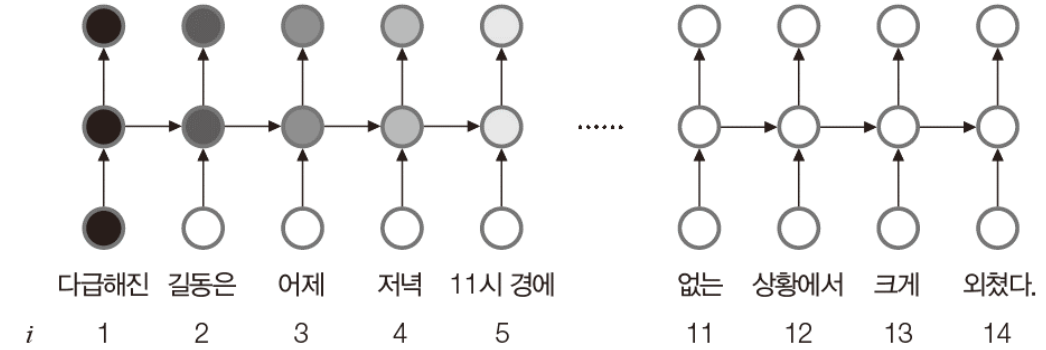


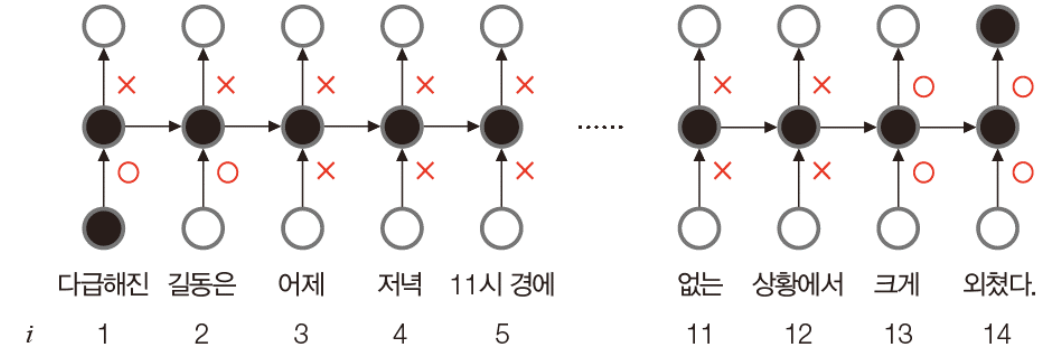
# 순환 신경망(Recurrent Neural Network, RNN)

## ➢ 선별 기억력을 갖춘 LSTM

- LSTM은 게이트라는 개념으로 선별 기억 확보
  - o는 열림, x는 닫힘
  - 실제로는 게이트는 0~1 사이의 실수값으로 열린 정도를 조절
  - 게이트의 여닫는 정도는 가중치로 표현되며 가중치는 학습으로 알아냄
- LSTM의 가중치
  - 순환 신경망의 {**U**, **V**, **W**}에 4개를 추가하여 {**U**, $\mathbf{U}^i$, $\mathbf{U}^o$, **W**, $\mathbf{W}^i$, $\mathbf{W}^o$, **V**}
  - i는 입력 게이트, o는 출력 게이트




(a) 선별 기억 능력이 없는 순환 신경망



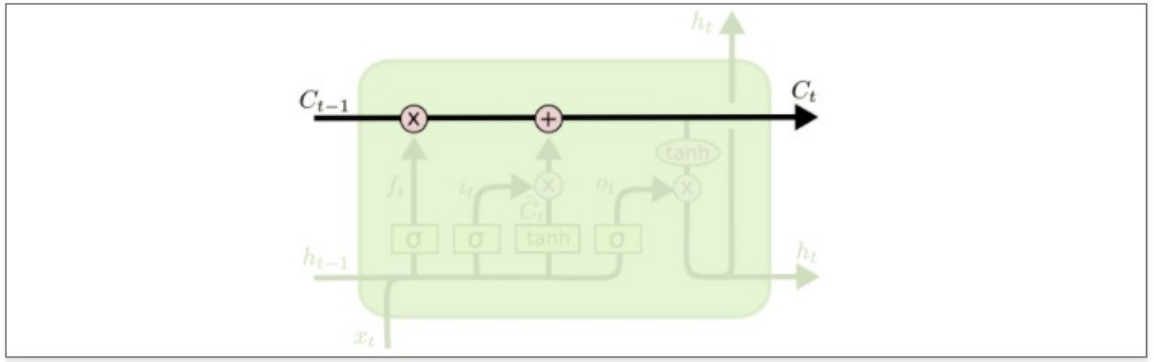

(b) 선별 기억 능력을 지닌 LSTM

순환 신경망과 LSTM

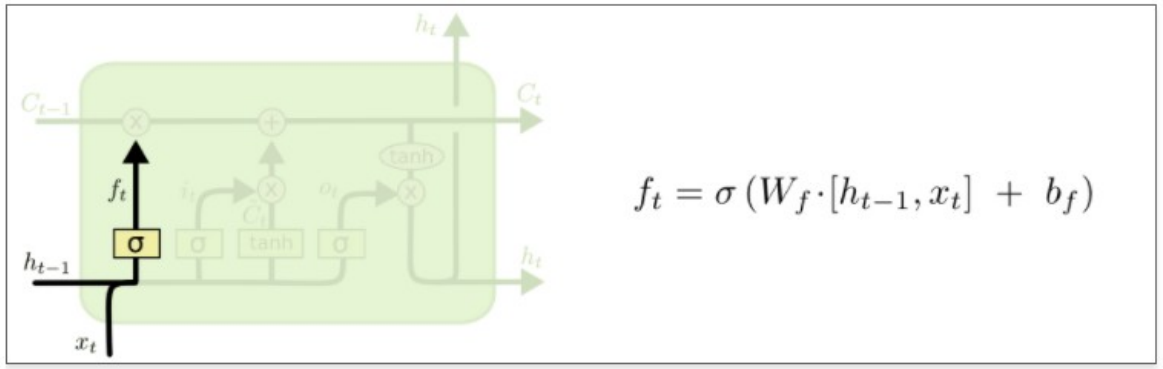
# 순환 신경망(Recurrent Neural Network, RNN)

## ➢ LSTM (Long Short Term Memory)

- **Cell state** : 정보가 바뀌지 않고 그대로 흐르도록 하는 역할
  이전 시점의 셀 상태가 다음 시점의 셀 상태를 구하기 위한 입력으로서 사용됩니다.



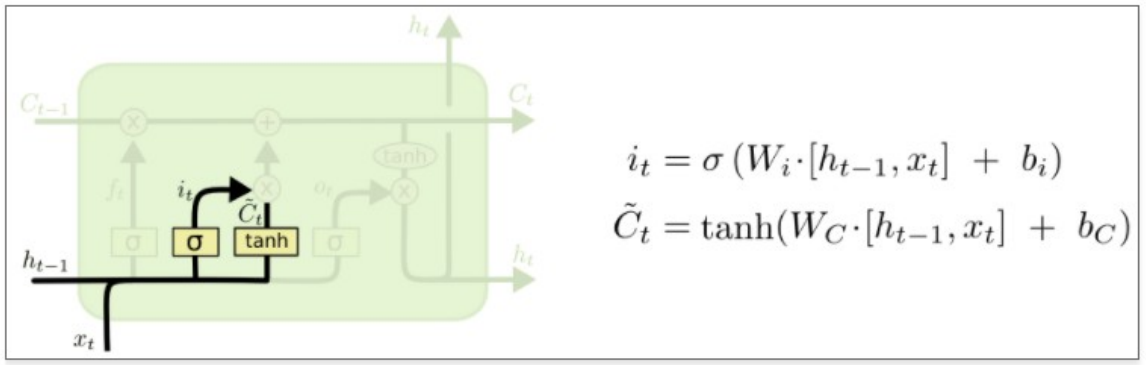

- **Forget gate** : cell state에서 sigmoid layer를 거쳐 어떤 정보를 버릴 것인지 정합니다.
  현재 시점 t의 값과 이전 시점 t-1의 은닉 상태가 시그모이드 함수를 지나게 됩니다.
  시그모이드 함수를 지나면 0과 1 사이의 값이 나오며, 값은 삭제 과정을 거친 정보의 양입니다.
  0에 가까울수록 정보가 많이 삭제된 것이고 1에 가까울수록 정보를 온전히 기억한 것입니다.



# 순환 신경망(Recurrent Neural Network, RNN)

## ➢ LSTM (Long Short Term Memory)

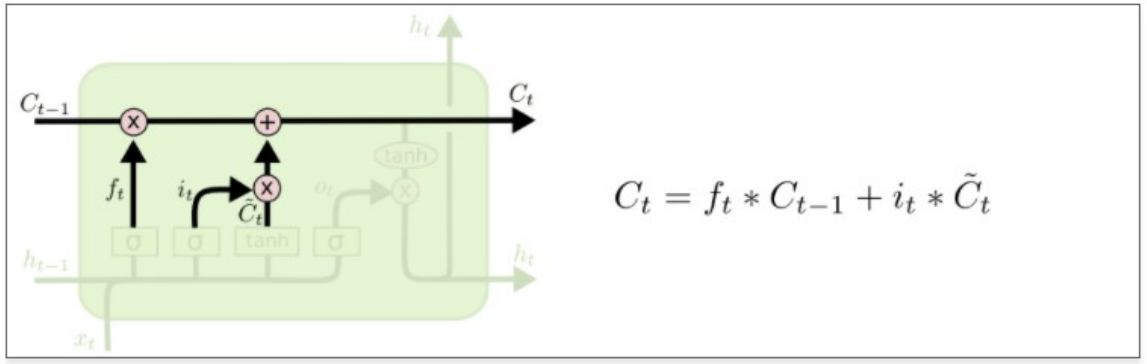
- **Input gate** : 들어오는 새로운 정보 중 어떤 것을 cell state에 저장할 것인지를 정합니다.
  sigmoid layer를 거쳐 어떤 값을 업데이트할 것인지를 정한 후 tanh layer에서 새로운 후보 Vector를 만듭니다.
  현재 시점의 입력을 얼마나 반영할지를 결정합니다.




$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

- 이전 gate에서 버릴 정보들과 업데이트할 정보들을 정했다면, Cell state update 과정에서 업데이트를 진행합니다
- 입력 게이트에서 선택된 기억을 삭제 게이트의 결과값과 더합니다

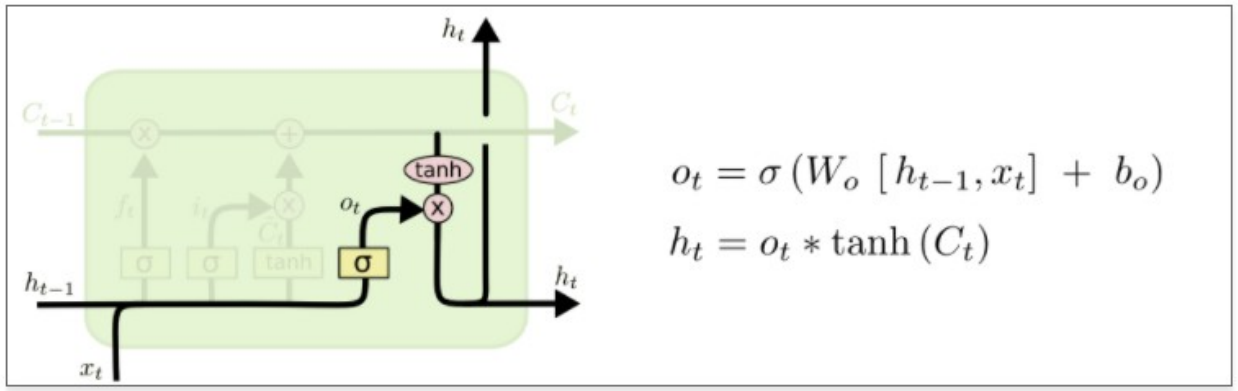


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

# 순환 신경망(Recurrent Neural Network, RNN)

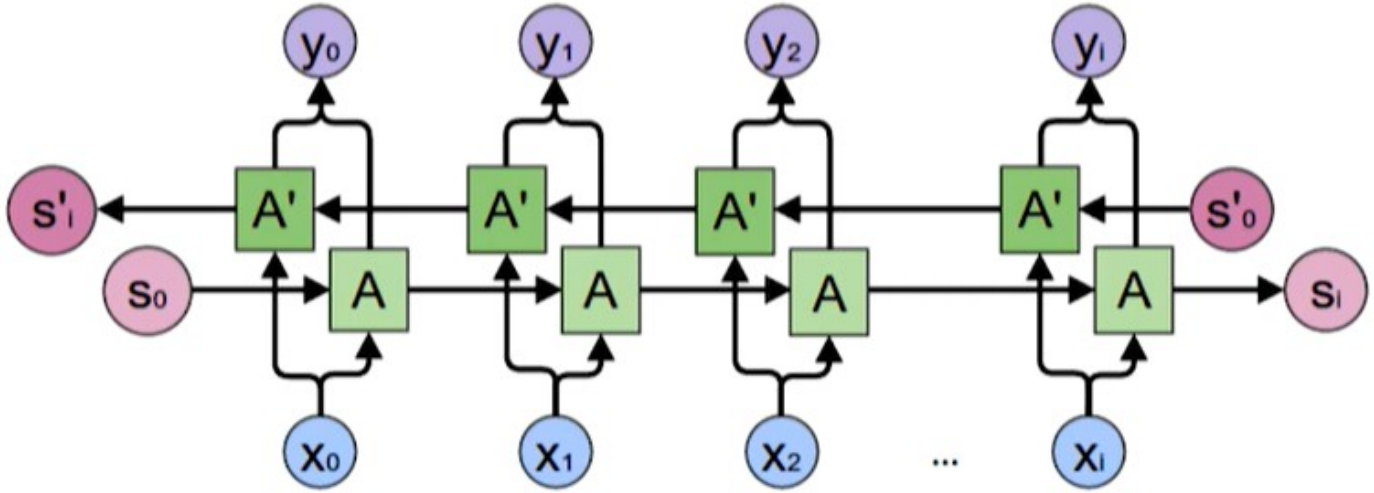
## ➢ LSTM (Long Short Term Memory)

- **Output gate** - 어떤 정보를 output으로 내보낼지 정하게 됩니다.
- sigmoid layer에 input data를 넣어 output 정보를 정한 후 Cell state를 tanh layer에 넣어 sigmoid layer의 output과 곱하여 output으로 내보냅니다.
- 출력 게이트는 현재 시점 t의 값과 이전 시점 t-1의 은닉 상태가 시그모이드 함수를 지난 값입니다.
- 출력 게이트의 값은 현재 시점 t의 은닉 상태를 결정하는 일에 쓰이게 됩니다.
- 셀 상태의 값이 하이퍼볼릭탄젠트 함수를 지나 -1과 1사이의 값이 되고, 해당 값은 출력 게이트의 값과 연산되면서, 값이 걸러지는 효과가 발생하여 은닉 상태가 됩니다.



# 순환 신경망(Recurrent Neural Network, RNN)

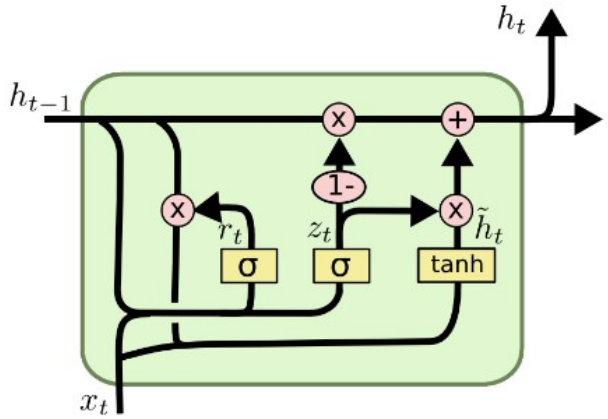
- **Bidirectional LSTM**
  - Forward 방향으로의 영향(t->t+1) 뿐만 아니라 Backward(t->t-1) 방향의 영향도 함께 고려
  - 과거와 현재, 미래의 모든 문맥 고려



# 순환 신경망(Recurrent Neural Network, RNN)

## ➢ 게이트 순환 유닛(Gated Recurrent Unit, GRU)

- LSTM의 장기 의존성 문제에 대한 해결책을 유지하면서, 은닉 상태를 업데이트하는 계산을 줄였습니다.
- GRU는 성능은 LSTM과 유사하면서 복잡했던 LSTM의 구조를 간단화 시켰습니다.
- GRU에서는 업데이트 게이트와 리셋 게이트 두 가지 게이트만이 존재합니다.
- 경험적으로 데이터 양이 적을 때는 매개 변수의 양이 적은 GRU가 조금 더 낫고, 데이터 양이 더 많으면 LSTM이 더 낫다고도 합니다.




$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

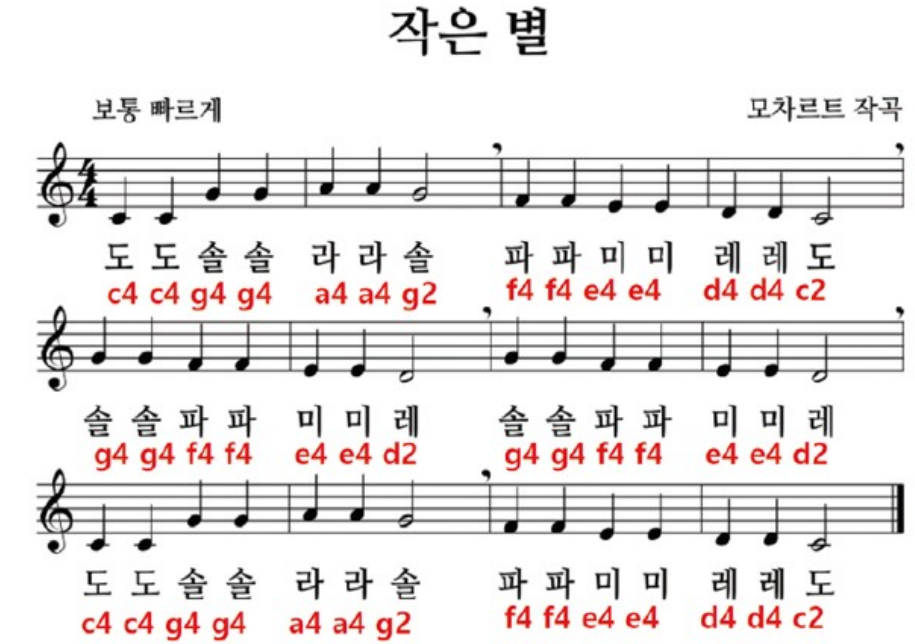
```
model.add(GRU(hidden_size, input_shape=(timesteps, input_dim)))
```

# 순환 신경망(Recurrent Neural Network, RNN)

- **인공지능의 창작 능력**
  - 딥드림으로 생성한 그림
  - 마젠타 프로젝트의 음악 창작 활동
  - 인공지능 소설 등

도 → C
레 → D　　2분 음표 → 2
미 → E
파 → F　　4분 음표 → 4
솔 → G
라 → A　　8분 음표 → 8
시 → B

(a) ABC 표기 변환 규칙




(b) '작은별' 악보를 ABC 표기로 변환한 예

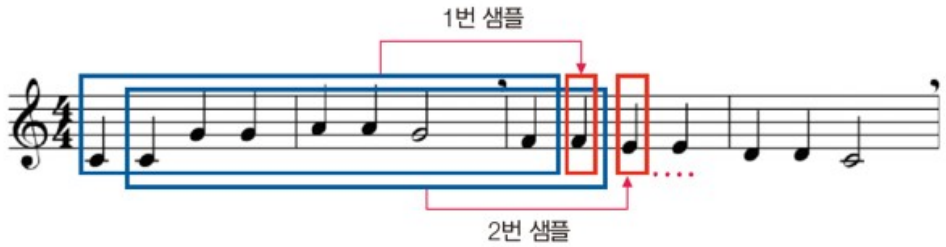
**앞 소절을 보고 다음 음표를 LSTM으로 예측하는 방식의 단순한 편곡**

# 순환 신경망(Recurrent Neural Network, RNN)

## ➢ 악보를 학습하는 LSTM 신경망

- ABC 표기 데이터는 2채널(계이름과 박자)이고, 분류 문제임

```
01  import music21
02
03  # "작은 별" 악보를 ABC 표기로 표현
04  little_star="tinynotation: 4/4 c4 c4 g4 g4 a4 a4 g2 f4 f4 e4 e4 d4 d4 c2 g4 g4
    f4 f4 e4 e4 d2 g4 g4 f4 f4 e4 e4 d2 c4 c4 g4 g4 a4 a4 g2 f4 f4 e4 e4 d4 d4 c2"
05  music21.converter.parse(little_star).show('mid')   # 스피커로 연주를 들려줌

07  import numpy as np
08
09  # 계이름과 숫자를 상호 변환하는 표(딕셔너리 자료구조를 사용함)
10  note2num={'c':1,'d':2,'e':3,'f':4,'g':5,'a':6,'b':7}
11  num2note={1:'c',2:'d',3:'e',4:'f',5:'g',6:'a',7:'b'}

13  # ABC 표기를 시계열 데이터로 변환
14  def abc2timeseries(s):
15      notes=s.split(' ')[2:]
16      seq=[]
17      for i in notes:
18          seq.append([note2num[i[0]],int(i[1])])
19      return seq
```




(a) 데이터 샘플링

특징 벡터 x: [[1 4] [1 4] [5 4] [5 4] [6 4] [6 4] [5 2] [4 4]]
레이블 y: [4 4] → 원핫 00000000001000000000

(b) 레이블의 원핫 표현(1번 샘플의 예)

# 순환 신경망(Recurrent Neural Network, RNN)

## ➢ 악보를 학습하는 LSTM 신경망

```
21  # 시계열 데이터를 ABC 표기로 변환
22  def timeseries2abc(t):
23      s='tinynotation: 4/4'
24      for i in t:
25          s=s+' '+num2note[i[0]]+str(i[1])
26      return s
27
28  # 원핫 코드로 변환하는 표
29  onehot=[[1,2],[2,2],[3,2],[4,2],[5,2],[6,2],[7,2],[1,4],[2,4],[3,4],[4,4],[5,4],[6,4],
    [7,4],[1,8],[2,8],[3,8],[4,8],[5,8],[6,8],[7,8]]
30
31  # 레이블을 원핫 코드로 변환
32  def to_onehot(l):
33      t=[]
34      for i in range(len(l)):
35          a=np.zeros(len(onehot))
36          a[onehot.index(list(l[i]))]=1.0
37          t.append(a)
38      return np.array(t)
```

```
40  # 시계열 데이터를 훈련 집합으로 자름
41  def seq2dataset(seq,window,horizon):
42      X=[]; Y=[]
43      for i in range(len(seq)-(window+horizon)+1):
44          x=seq[i:(i+window)]
45          y=(seq[i+window+horizon-1])
46          X.append(x); Y.append(y)
47      return np.array(X), np.array(Y)
```

# 순환 신경망(Recurrent Neural Network, RNN)

➢ **악보를 학습하는 LSTM 신경망**

```
49  w=8    # 윈도우 크기
50  h=1    # 수평선 계수
51
52  seq=abc2timeseries(little_star)
53  X,Y=seq2dataset(seq,w,h)
54  print(X.shape,Y.shape)
55  print(X[0],Y[0])
56
57  from tensorflow.keras.models import Sequential
58  from tensorflow.keras.layers import Dense, LSTM
59  import tensorflow as tf
60
61  # 훈련 집합 구축
62  split=int(len(X)*1.0)    # 100%를 훈련집합으로 사용
63  x_train=X[0:split]; y_train=Y[0:split]
64  y_train=to_onehot(y_train)
66  # LSTM 모델 설계와 학습
67  model=Sequential()
68  model.add(LSTM(units=128,activation='relu',input_shape=x_train[0].shape))
69  model.add(Dense(y_train.shape[1],activation='softmax'))
70  model.compile(loss='categorical_crossentropy',optimizer='Adam',metrics=
    ['accuracy'])
71  model.fit(x_train,y_train,epochs=200,batch_size=1,verbose=2)
```

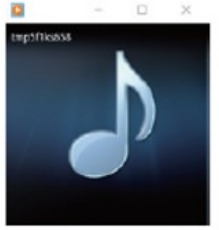
# 순환 신경망(Recurrent Neural Network, RNN)

## ➢ 악보를 학습하는 LSTM 신경망

```
74  def arranging_music(model,first_measure,duration):
75      music=first_measure
76      for i in range(duration):
77          p=model.predict(np.float32(np.expand_dims(music[-w:],axis=0)))
78          music=np.append(music,[onehot[np.argmax(p)]],axis=0)
79      return timeseries2abc(music)
80
81  new_song=arranging_music(model,x_train[0],50)
82
83  print(new_song)
84  music21.converter.parse(new_song).show('mid')
```

```
(34, 8, 2) (34, 2)

[[1 4] [1 4] [5 4] [5 4] [6 4] [6 4] [5 2] [4 4]]  [4 4]

Train on 34 samples
Epoch 1/200
34/34 - 4s - loss: 2.6875 - accuracy: 0.1765
…
Epoch 200/200
34/34 - 0s - loss: 7.0307e-05 - accuracy: 1.0000

tinynotation: 4/4 c4 c4 g4 g4 a4 a4 g2 f4 f4 e4 e4 d4 d4 c2 g4 g4 f4 f4 e4 e4 d2 g4
g4 f4 f4 e4 e4 d2 c4 c4 g4 g4 a4 a4 g2 f4 f4 e4 e4 d4 d4 c2 g4 g4 f4 f4 e4 e4 d2 g4
g4 f4 f4 e4 e4 d2 c4 c4
```

#학습된 모델로 편곡을 하는 함수
(first_measure: 첫 소절, duration: 생성될 곡의 길이)
75행 첫 소절을 지정
76행 duration만큼 반복
77행 직전 윈도우를 보고 다음 음표를 예측
78행 가장 큰 확률을 가진 부류로 분류하고
ABC 표기로 변환하여 리스트에 추가



파란 부분은 원래 곡과 길이가 같은 곳까지이고
편곡 부분은 빨간 색으로 표시

# 순환 신경망(Recurrent Neural Network, RNN)

➢ 단어 임베딩 기술을 이용하여 IMDB의 샘플을 긍정과 부정으로 분류

```
01  from tensorflow.keras.datasets import imdb
02  from tensorflow.keras.models import Sequential
03  from tensorflow.keras.layers import Dense,Flatten,Embedding
04  from tensorflow.keras import preprocessing
05
06  dic_siz=10000        # 사전의 크기(사전에 있는 단어 개수)
07  sample_siz=512       # 샘플의 크기
08
09  # tensorflow가 제공하는 간소한 버전의 IMDB 읽기
10  (x_train,y_train),(x_test,y_test)=imdb.load_data(num_words=dic_siz)
11  print(x_train.shape,x_test.shape)
12  print(x_train[0])
13
14  # 단어를 숫자, 숫자를 단어로 변환하는데 쓰는 표(표는 딕셔너리로 구현)
15  word2id=imdb.get_word_index()
16  id2word={word:id for id,word in word2id.items()}
17
18  for i in range(1,21):    상위 빈도 20개 단어 출력
19      print(id2word[i],end='/')
```

```
(25000,) (25000,)

[1, 14, 22, 16, 43, 530, 973, 1622, 1385, 65, 458, 4468, 66, 3941, 4, 173, 36, 256, 5,
25, 100, 43, 838, 112, 50, 670, 2, 9, 35, 480, 284, 5, 150, 4, 172, 112, 167, 2, 336,
385, 39, 4, 172, 4536, 1111, 17, 546, 38, 13, 447, 4, 192, 50, 16, 6, 147, 2025, 19,
14, 22, 4, 1920, 4613, 469, 4, 22, 71, 87, 12, 16, 43, 530, 38, 76, 15, 13, 1247, 4, 22,
17, 515, 17, 12, 16, 626, 18, 2, 5, 62, 386, 12, 8, 316, 8, 106, 5, 4, 2223, 5244, 16,
480, 66, 3785, 33, 4, 130, 12, 16, 38, 619, 5, 25, 124, 51, 36, 135, 48, 25, 1415, 33,
6, 22, 12, 215, 28, 77, 52, 5, 14, 407, 16, 82, 2, 8, 4, 107, 117, 5952, 15, 256, 4, 2,
7, 3766, 5, 723, 36, 71, 43, 530, 476, 26, 400, 317, 46, 7, 4, 2, 1029, 13, 104, 88, 4,
381, 15, 297, 98, 32, 2071, 56, 26, 141, 6, 194, 7486, 18, 4, 226, 22, 21, 134, 476,
26, 480, 5, 144, 30, 5535, 18, 51, 36, 28, 224, 92, 25, 104, 4, 226, 65, 16, 38, 1334,
88, 12, 16, 283, 5, 16, 4472, 113, 103, 32, 15, 16, 5345, 19, 178, 32]

the/and/a/of/to/is/br/in/it/i/this/that/was/as/for/with/movie/but/film/on/
```

# 순환 신경망(Recurrent Neural Network, RNN)

➢ 단어 임베딩 기술을 이용하여 IMDB의 샘플을 긍정과 부정으로 분류

```
26  # 신경망 모델 설계와 학습
27  embed=Sequential()
28  embed.add(Embedding(input_dim=dic_siz,output_dim=embed_space_dim,input_
    length=sample_siz))
29  embed.add(Flatten())
30  embed.add(Dense(32,activation='relu'))
31  embed.add(Dense(1,activation='sigmoid'))
32  embed.compile(loss='binary_crossentropy',optimizer='Adam',metrics=
    ['accuracy'])
33  hist=embed.fit(x_train,y_train,epochs=20,batch_size=64,validation_data=
    (x_test,y_test),verbose=2)
34
35  embed.summary()
```

Embedding 함수는 input_dim 차원을 output_dim 차원으로 축소

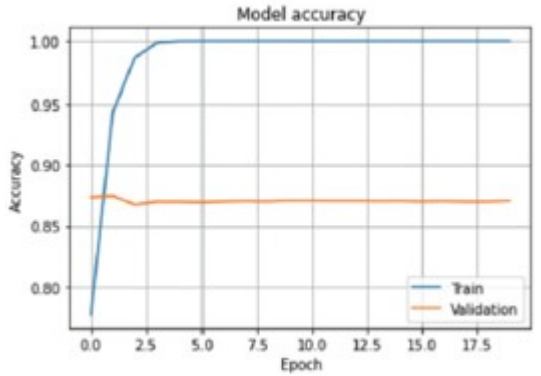
input_length*output_dim 구조의 텐서를 일렬로 펼침

0(부정) 또는 1(긍정)로 분류하므로 출력 노드는 1개
손실 함수는 binary_crossentropy

# 순환 신경망(Recurrent Neural Network, RNN)

➢ 단어 임베딩 기술을 이용하여 IMDB의 샘플을 긍정과 부정으로 분류

```
37  # 모델 평가
38  res=embed.evaluate(x_test,y_test,verbose=0)
39  print("정확률은",res[1]*100)
40
41  import matplotlib.pyplot as plt
42
43  # 학습 곡선
44  plt.plot(hist.history['accuracy'])
45  plt.plot(hist.history['val_accuracy'])
46  plt.title('Model accuracy')
47  plt.ylabel('Accuracy')
48  plt.xlabel('Epoch')
49  plt.legend(['Train','Validation'], loc='best')
50  plt.grid()
51  plt.show()
```




```
Train on 25000 samples, validate on 25000 samples
Epoch 1/20
25000/25000 - 6s - loss: 0.4595 - accuracy: 0.7503 - val_loss: 0.2825 - val_accuracy:
0.8812
...
Epoch 20/20
25000/25000 - 5s - loss: 1.6844e-05 - accuracy: 1.0000 - val_loss: 0.7613 - val_
accuracy: 0.8688

Model: "sequential_33"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_32 (Embedding)     (None, 512, 16)           160000
flatten_32 (Flatten)         (None, 8192)              0
_________________________________________________________________
dense_40 (Dense)             (None, 32)                262176
_________________________________________________________________
dense_41 (Dense)             (None, 1)                 33
=================================================================
Total params: 422,209
Trainable params: 422,209
Non-trainable params: 0
_________________________________________________________________

정확률은 86.87999844551086
```

# 순환 신경망(Recurrent Neural Network, RNN)

- **LSTM으로 네이버 영화 리뷰 감성 분류**
  - 하이퍼파라미터인 임베딩 벡터의 차원은 100, 은닉 상태의 크기는 128
  - 모델은 다 대 일 구조의 LSTM을 사용
  - 출력층에 로지스틱 회귀를 사용 - 활성화 함수로는 시그모이드 함수, 손실 함수로 크로스 엔트로피 함수, 하이퍼파라미터인 배치 크기는 64이며, 15 에포크를 수행

```
from tensorflow.keras.layers import Embedding, Dense, LSTM
from tensorflow.keras.models import Sequential
from tensorflow.keras.models import load_model
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint

embedding_dim = 100
hidden_units = 128

model = Sequential()
model.add(Embedding(vocab_size, embedding_dim))
model.add(LSTM(hidden_units))
model.add(Dense(1, activation='sigmoid'))

es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=4)
mc = ModelCheckpoint('best_model.h5', monitor='val_acc', mode='max', verbose=1, save_best_only=True)

model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])
history = model.fit(X_train, y_train, epochs=15, callbacks=[es, mc], batch_size=64, validation_split=0.2)
```

# 순환 신경망(Recurrent Neural Network, RNN)

- **GRU로 IMDB 리뷰 감성 분류**

```
import re
from tensorflow.keras.datasets import imdb
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, GRU, Embedding
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from tensorflow.keras.models import load_model

vocab_size = 10000
max_len = 500
(X_train, y_train), (X_test, y_test) = imdb.load_data(num_words=vocab_size)
X_train = pad_sequences(X_train, maxlen=max_len)
X_test = pad_sequences(X_test, maxlen=max_len)
embedding_dim = 100
hidden_units = 128
model = Sequential()
model.add(Embedding(vocab_size, embedding_dim))
model.add(GRU(hidden_units))
model.add(Dense(1, activation='sigmoid'))
```