

## ARTIFICIAL INTELLIGENCE

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Maecenas at dolor nunc. consequat a gravida non, lacinia vel mi. Fusce semper ex vitae bibendum lacinia.

[read more](#)

DELPHI

# 인공지능

딥러닝 중급\_RNN\_LSTM\_GRU

## BRAINSTORM

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Maecenas at dolor nunc. consequat a gravida non, lacinia vel mi. Fusce semper ex vitae bibendum lacinia.

[read more](#)

CSS

CMS

강성관 (silicon1@hanmail.net)

C++

JAVA

# RNN



### RNN (Recurrent Neural Network)

- 말이나 문장 등은 **이어지는 데이터(Sequential data)**로 구성되어 있기 때문에 그 의미를 전달하려면 각 단어가 정해진 순서대로 입력되어야 하므로 과거에 입력된 데이터와 나중에 입력된 데이터 사이의 관계를 고려해야 함 → RNN 고안



하나의 단어로 문장을 이해하기 어렵다



이전 단어를 고려해야 함



### RNN (Recurrent Neural Network)

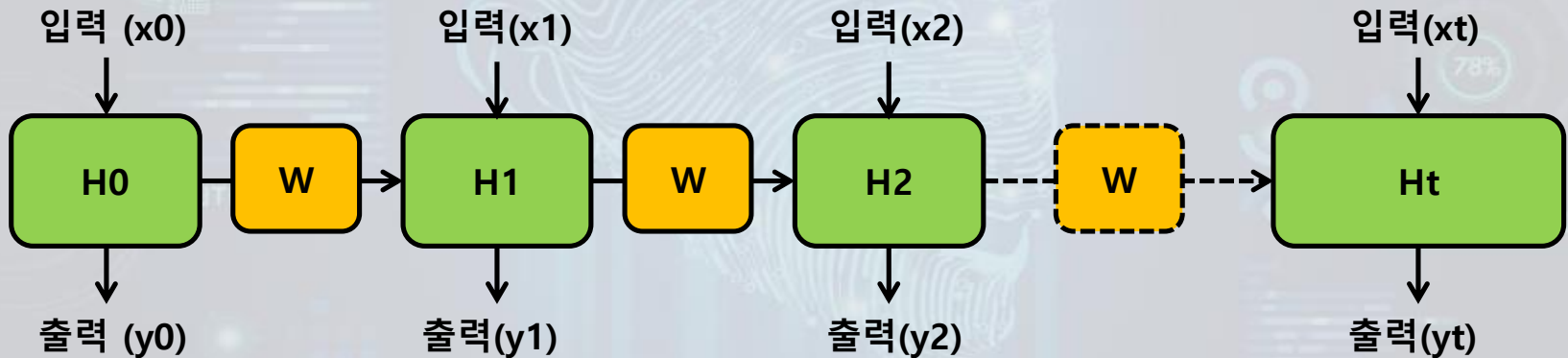
- 여러 개의 데이터가 순서대로 입력되었을 때 이전 입력 데이터를 저장했다가 저장 데이터의 중요도를 판단하여 가중치를 부여하고 가중치를 통해 현재 학습에 반영
- 반복적이고 순차적인 데이터(Sequential data) 학습에 특화된 인공지능망의 한 종류





### RNN (Recurrent Neural Network)

- 특징 : 시간에 따른 순서를 기억한다는 것으로 신경망의 출력이 다른 신경망과 연결되어 여러 개의 신경망으로 이어져 있음
- 단점 : 처음 시작한 Weight의 값이 점차 학습이 될 수록 1보다 작은 값이 곱해져서 상쇄 된다는 것  
→ Vanishing Gradient Problem



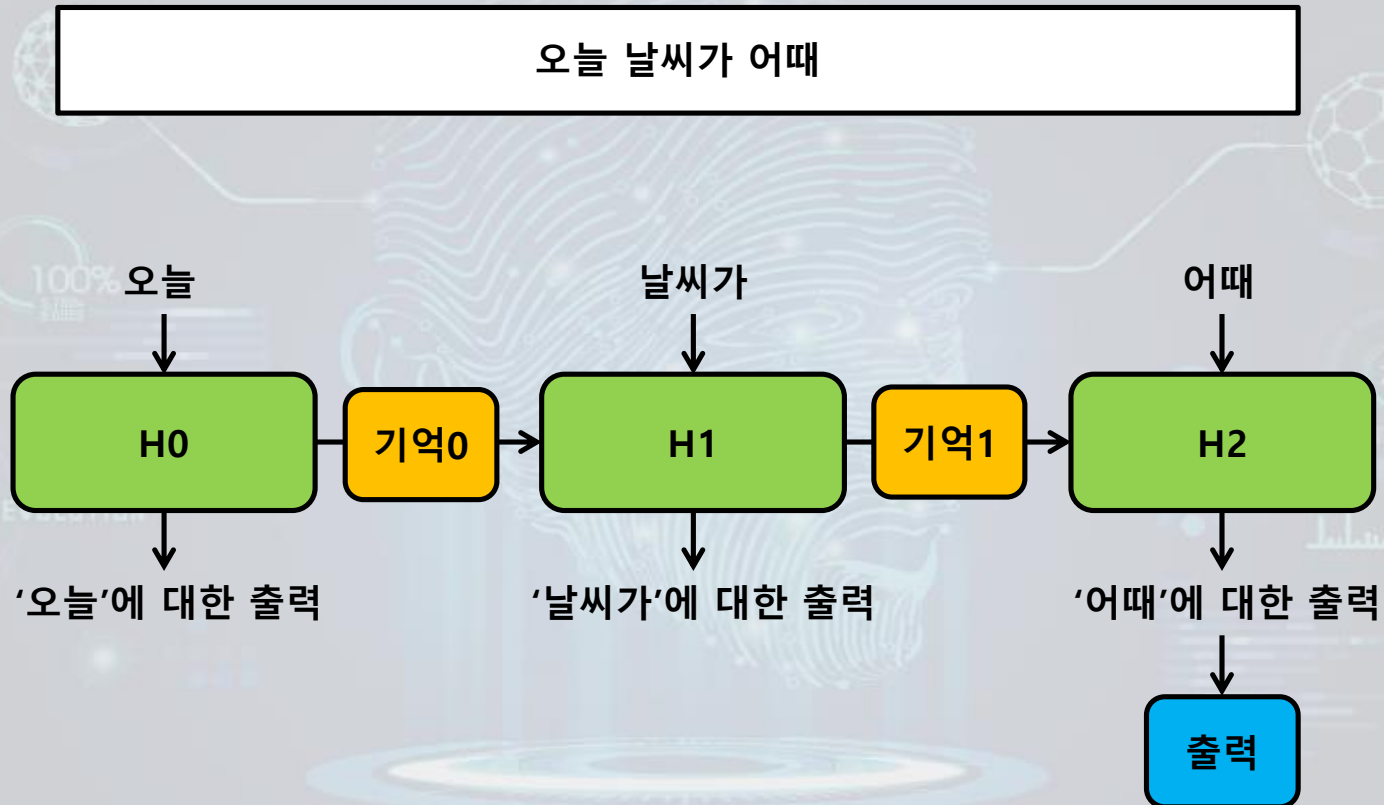
$$H_t = \tanh(W_{hh}H_{t-1} + W_{xh}x_t + b_h)$$

$$y_t = W_{hy}H_t + b_y$$

- $W_{hh}$  : 이전 히든레이어에서 다음 히든레이어로 보내는 가중치
- $W_{xh}$  : 입력  $x$ 을 히든레이어  $H$ 로 보내는 가중치
- $W_{hy}$  : 히든레이어  $h$ 에서 출력  $y$ 로 보내는 가중치

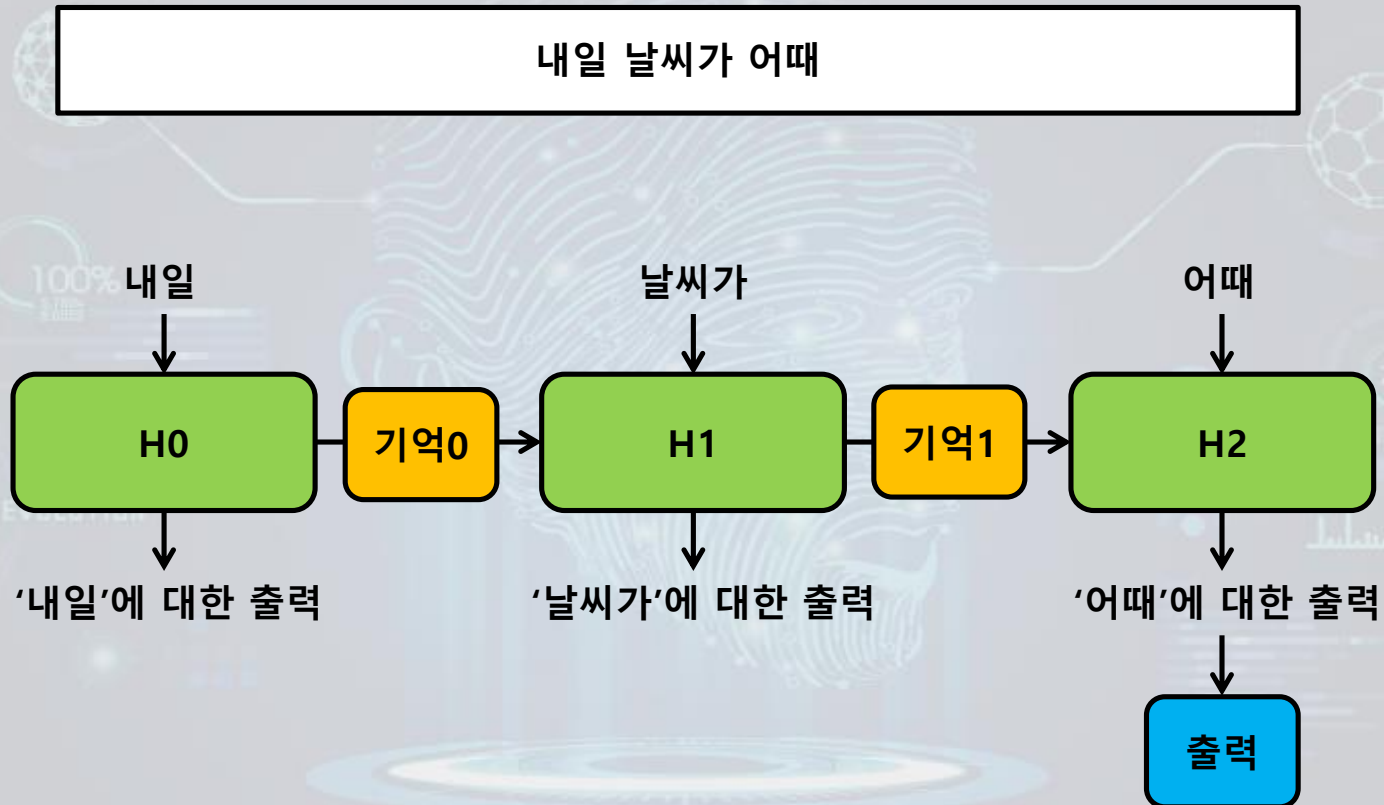


### RNN (Recurrent Neural Network)





### RNN (Recurrent Neural Network)

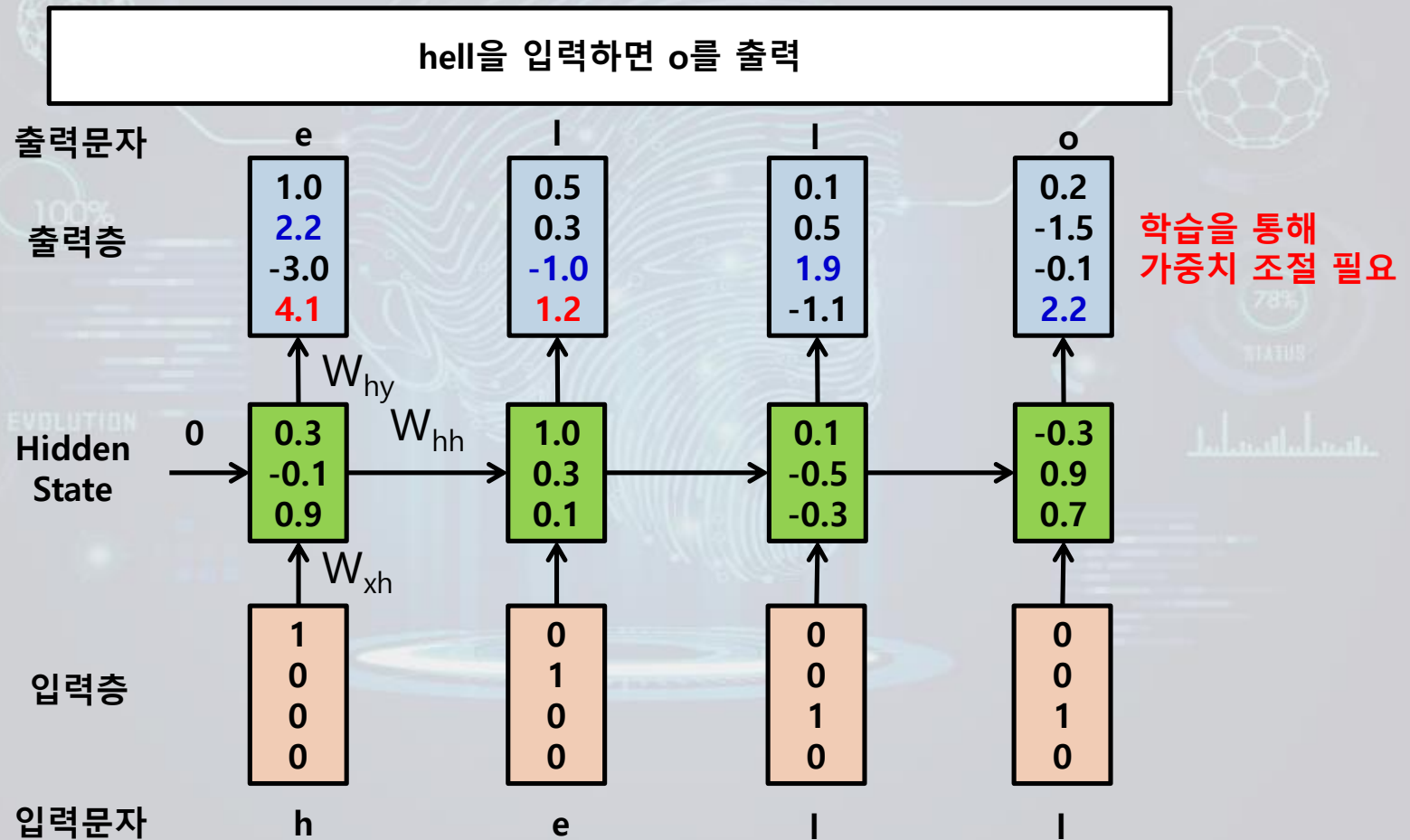






### RNN (Recurrent Neural Network)

- 어떤 글자가 주어졌을 때 다음 글자를 예측하는 character-level-model을 만든다면

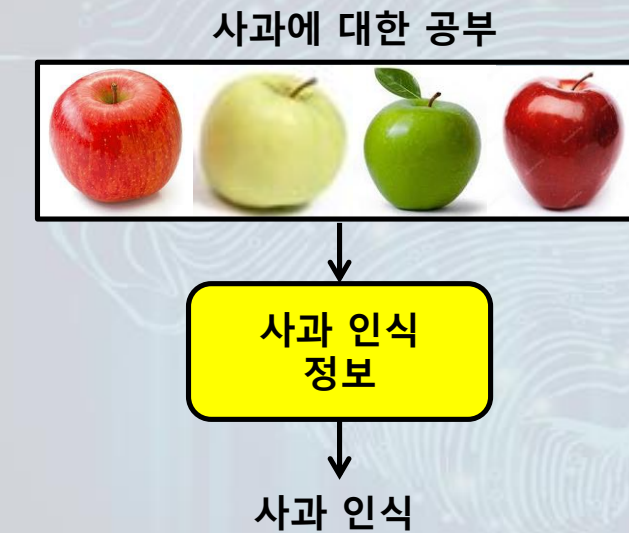






### RNN (Recurrent Neural Network)

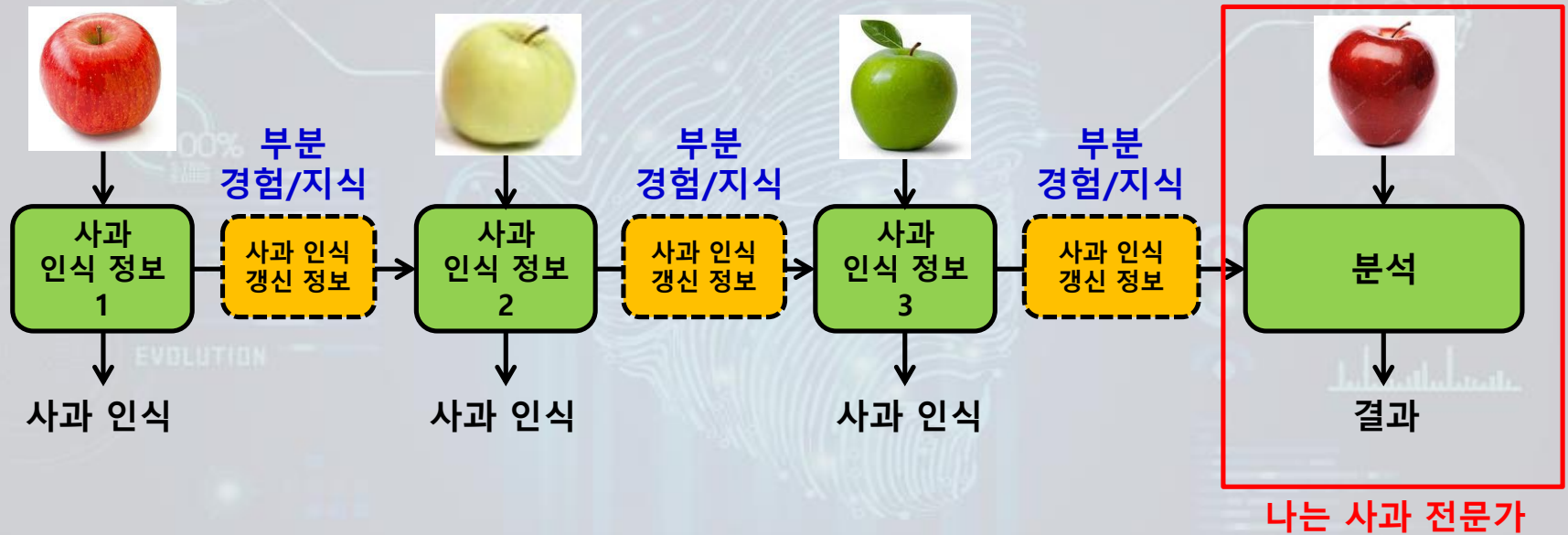
- 적용 예 - 일반 신경망





### RNN (Recurrent Neural Network)

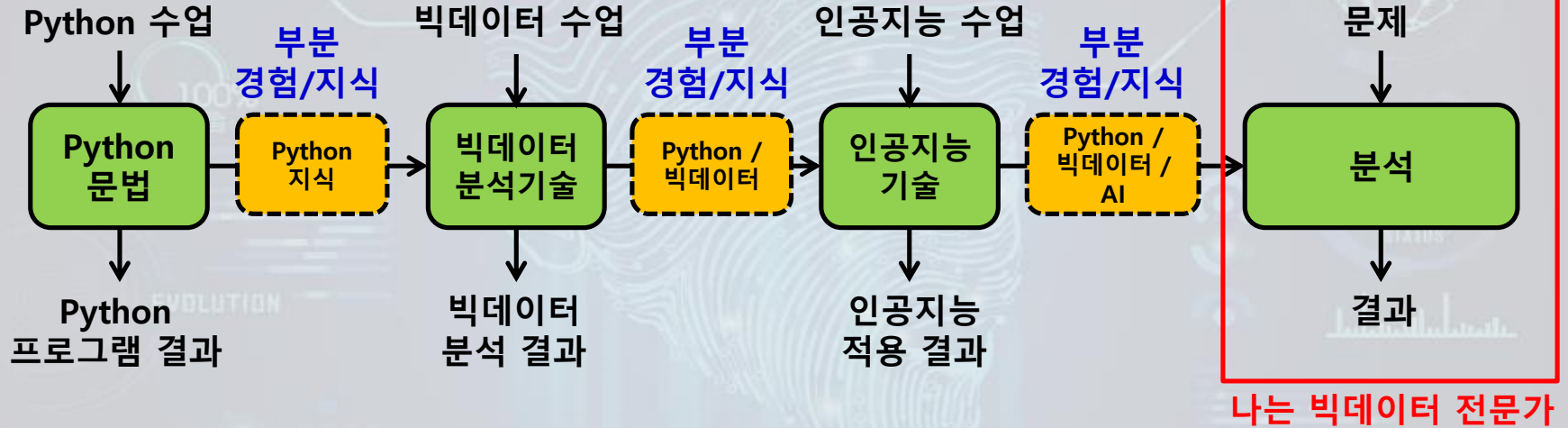
● 적용 예 - RNN





### RNN (Recurrent Neural Network)

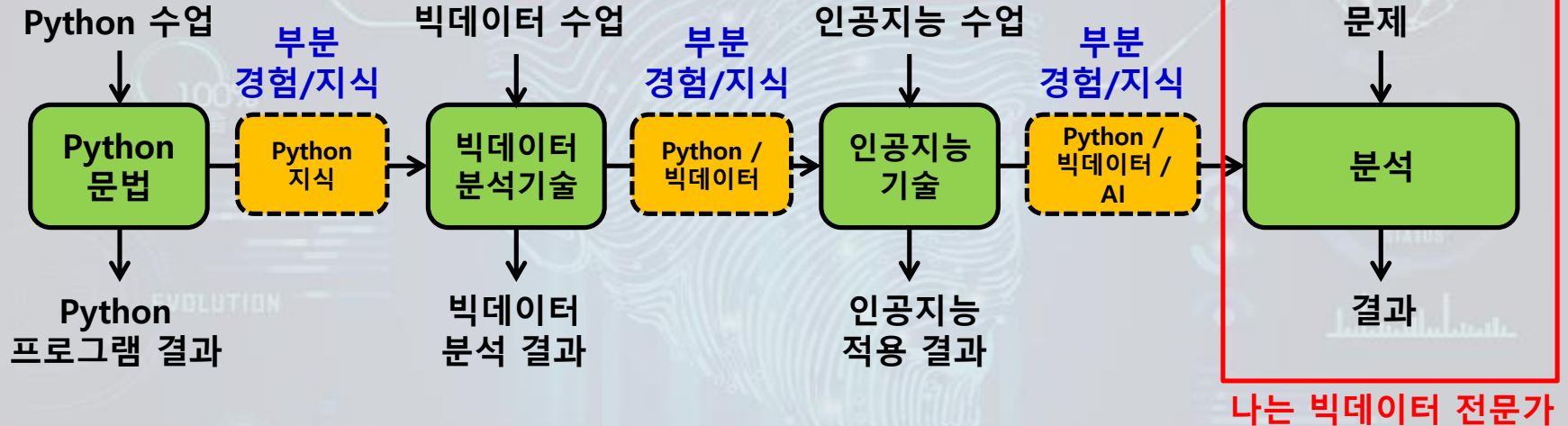
● 적용 예





### RNN (Recurrent Neural Network)

● 적용 예

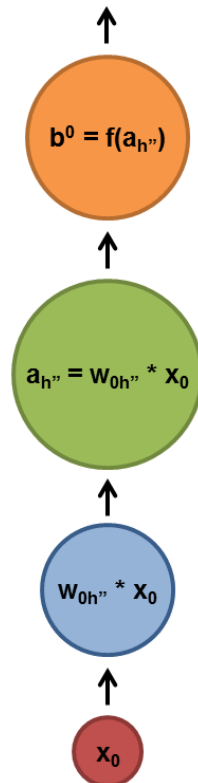




### RNN (Recurrent Neural Network)

- $x$ 는 입력,  $w$ 는 입력 데이터에 곱해지는 가중치,  $a$ 는 은닉층의 활성화값(=입력과 은닉층 값을 고려해서 구해지는 값),  $b$ 는 은닉층이 sigmoid 함수를 통과한 출력

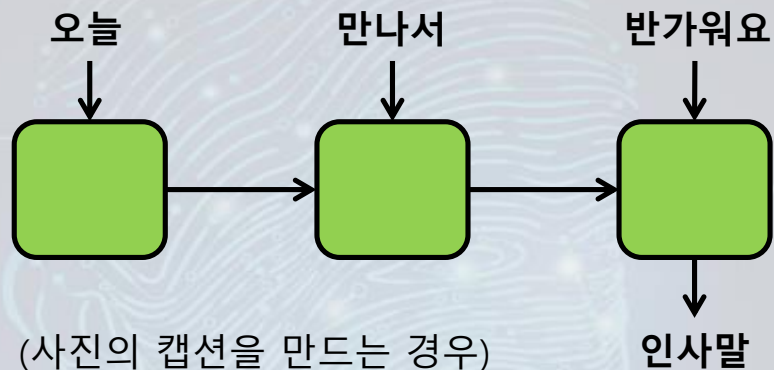
$b^0$  is fed to next layer



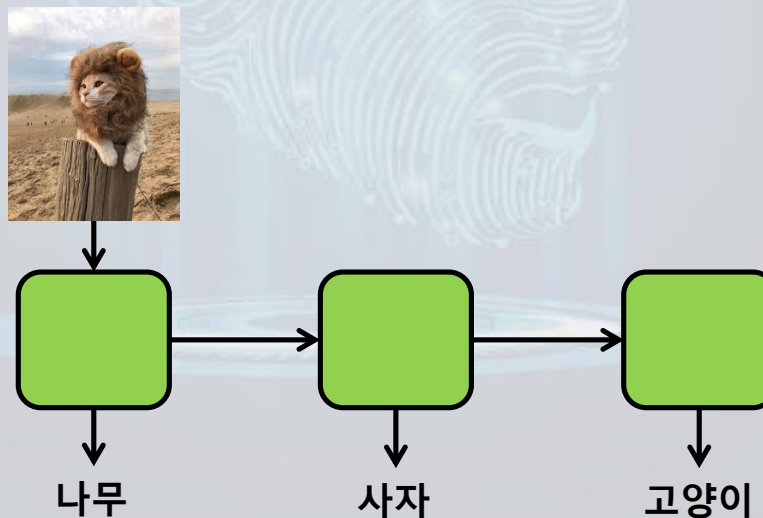


## RNN (Recurrent Neural Network)

- 적용 A : 다수 입력 단일 출력 (문장을 읽고 뜻을 파악하는 경우)



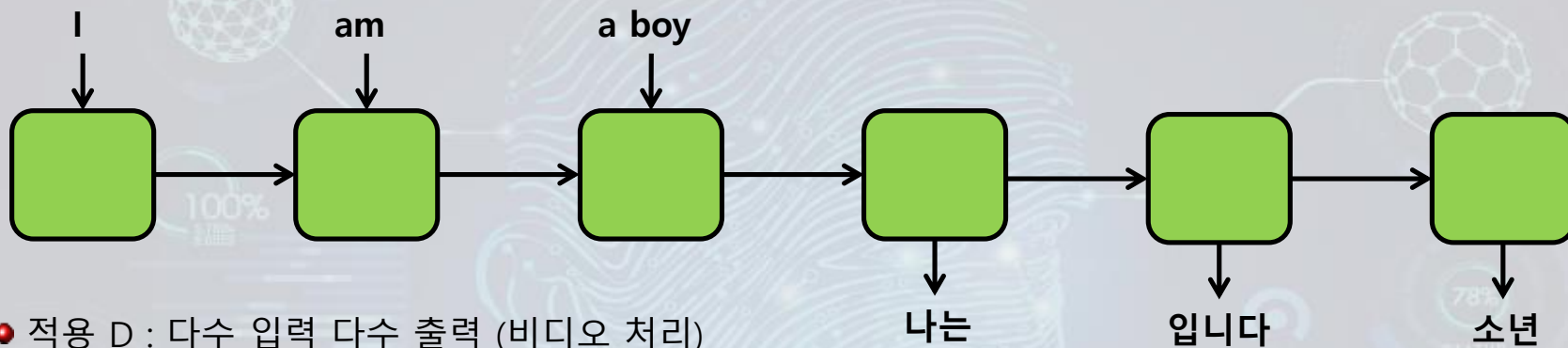
- 적용 B : 단일 입력 다수 출력 (사진의 캡션을 만드는 경우)



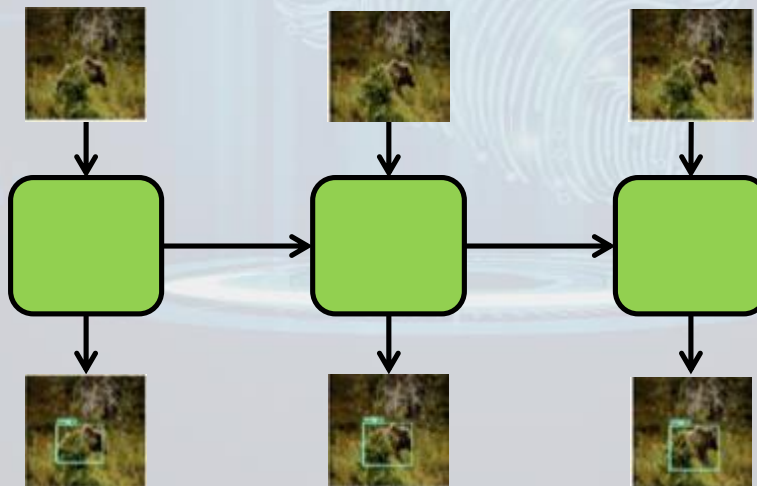


### RNN (Recurrent Neural Network)

- 적용 C : 다수 입력 다수 출력 (문장 번역)



- 적용 D : 다수 입력 다수 출력 (비디오 처리)

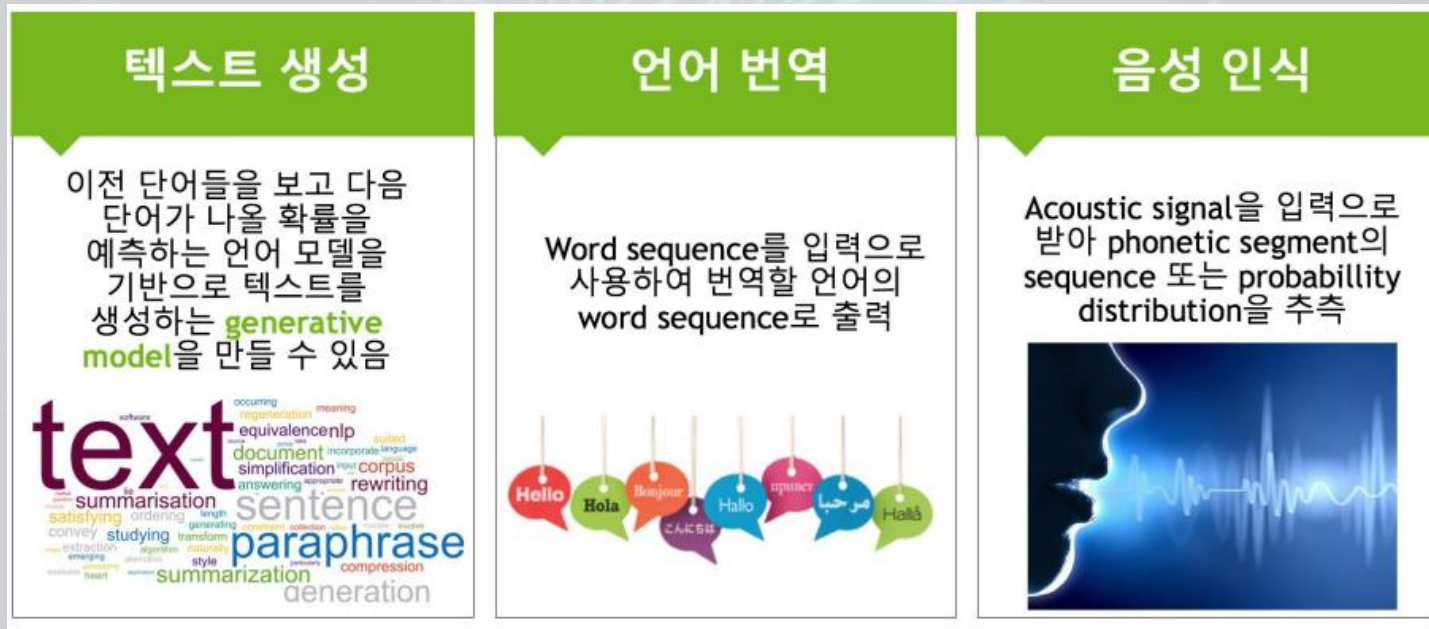






### RNN (Recurrent Neural Network)

#### ● 용도





### RNN (Recurrent Neural Network)

- Keras의 **SimpleRNN** Layer
- 가장 단순한 구조의 RNN로 입력데이터는 (batch\_size, timesteps, input\_dim)으로 3차원 텐서 형태여야 함
  - batch\_size : 자료의 수
  - timesteps : 순서열의 길이
  - input\_dim : 벡터의 길이
- SimpleRNN은 2가지 모드로 실행
  - (1) 각 타임스텝의 출력을 모은 전체 시퀀스를 반환 (batch\_size, timesteps, output\_features)
  - (2) 입력 시퀀스에 대한 마지막 출력만 반환 (batch\_size, output\_features)
- SimpleRNN 함수

**SimpleRNN(units, activation, return\_sequences, ....)**

- units : 출력 공간의 차원
- return\_sequences : 출력 시퀀스의 전체를 반환할 지 여부



### RNN (Recurrent Neural Network)

- Keras의 SimpleRNN Layer 사용 – 마지막 타임스텝의 출력만 얻음

```
from keras.models import Sequential
from keras.layers import Embedding, SimpleRNN
```

```
model = Sequential()
model.add(Embedding(10000, 32))
model.add(SimpleRNN(32))
model.summary()
```

Layer (type)	Output Shape	Param #
embedding_1 (Embedding)	(None, None, 32)	320000
simple_rnn_1 (SimpleRNN)	(None, 32)	2080

Total params: 322,080  
Trainable params: 322,080  
Non-trainable params: 0



## RNN (Recurrent Neural Network)

- Keras의 SimpleRNN Layer 사용 – 전체 상태 시퀀스를 반환

```
from keras.models import Sequential
from keras.layers import Embedding, SimpleRNN

model = Sequential()
model.add(Embedding(10000, 32))
model.add(SimpleRNN(32, return_sequences=True))
model.summary()
```

Layer (type)	Output Shape	Param #
embedding_1 (Embedding)	(None, None, 32)	320000
simple_rnn_1 (SimpleRNN)	(None, None, 32)	2080
Total params: 322,080		
Trainable params: 322,080		
Non-trainable params: 0		



## IMDB 데이터셋 적용하기

- 라이브러리, 변수 설정, 데이터 로딩

```
1 from keras.datasets import imdb
2 from keras.preprocessing import sequence
3 from keras.models import Sequential
4 from keras.layers import Embedding, SimpleRNN, Dense
5 import matplotlib.pyplot as plt
6
7 max_features = 10000
8 maxlen = 500 #500개의 훈련데이터만 사용
9 batch_size = 32
10
11 (x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_features)
12 x_train = sequence.pad_sequences(x_train, maxlen=maxlen)
13 x_test = sequence.pad_sequences(x_test, maxlen=maxlen)
```



## IMDB 데이터셋 적용하기

- 신경망 설계, 컴파일, 학습하기, 모델 저장

```
15 model = Sequential()
16 model.add(Embedding(max_features, 32))
17 model.add(SimpleRNN(32))
18 model.add(Dense(1, activation='sigmoid'))
19 model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])
20
21 history = model.fit(x_train, y_train, epochs=10, batch_size=128, validation_split=0.2)
22 model.save('simpleRNN_model.h5')
```



## IMDB 데이터셋 적용하기

### ● 결과 출력하기

```
24 acc = history.history['acc']
25 val_acc = history.history['val_acc']
26 loss = history.history['loss']
27 val_loss = history.history['val_loss']
28
29 epochs = range(1, len(acc) + 1)
30
31 plt.plot(epochs, acc, 'bo', label='Training acc')
32 plt.plot(epochs, val_acc, 'b', label='Validation acc')
33 plt.title('Training and Validation accuracy')
34 plt.legend()
35 plt.figure()
36 plt.plot(epochs, loss, 'bo', label='Training loss')
37 plt.plot(epochs, val_loss, 'b', label='Validation loss')
38 plt.title('Training and Validation loss')
39 plt.legend()
40 plt.show()
```

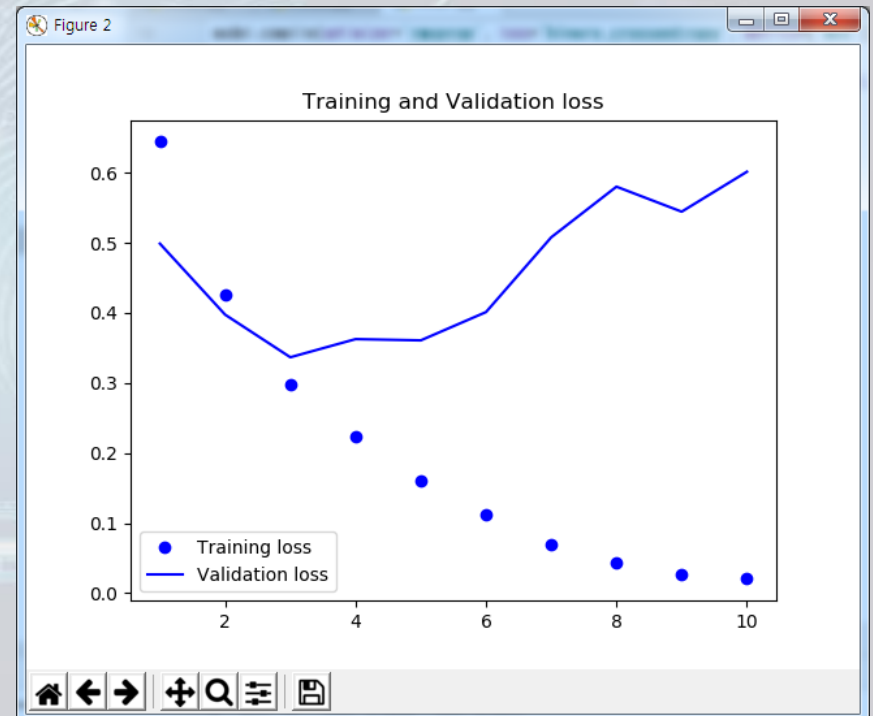
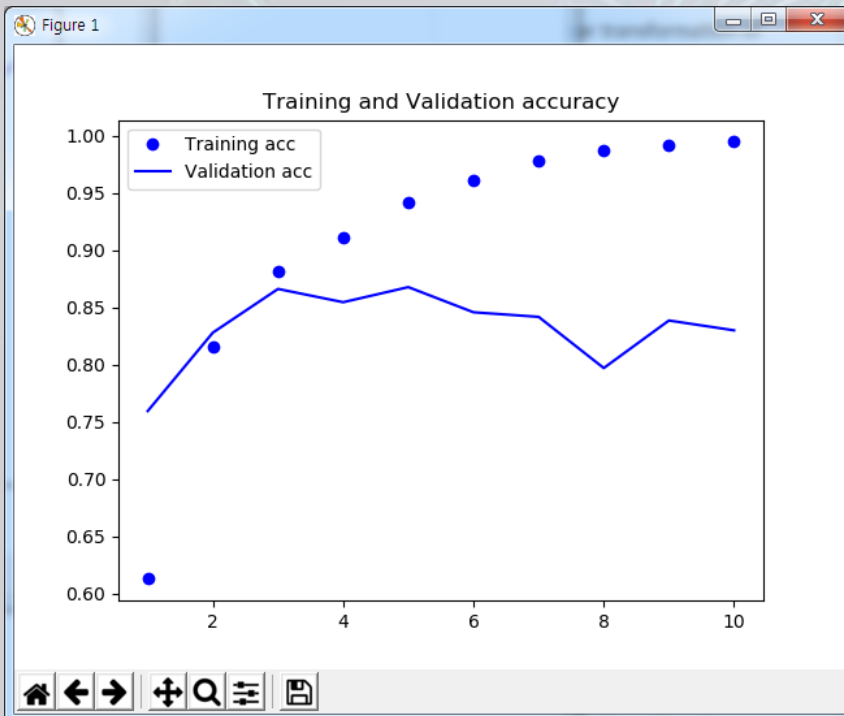




### IMDB 데이터셋 적용하기

- SimpleRNN은 텍스트처럼 긴 시퀀스 (은닉층의 증가)를 처리하는 데는 적합하지 않음

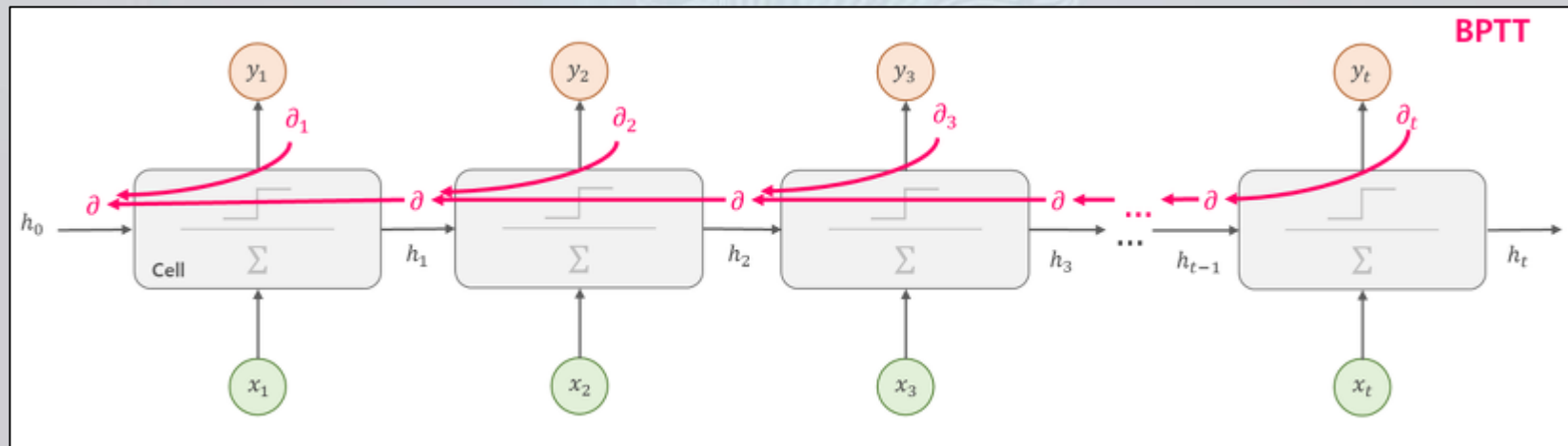
loss: 0.0204 - acc: 0.9949 - val\_loss: 0.6016 - **val\_acc: 0.8300**





### RNN (Recurrent Neural Network)

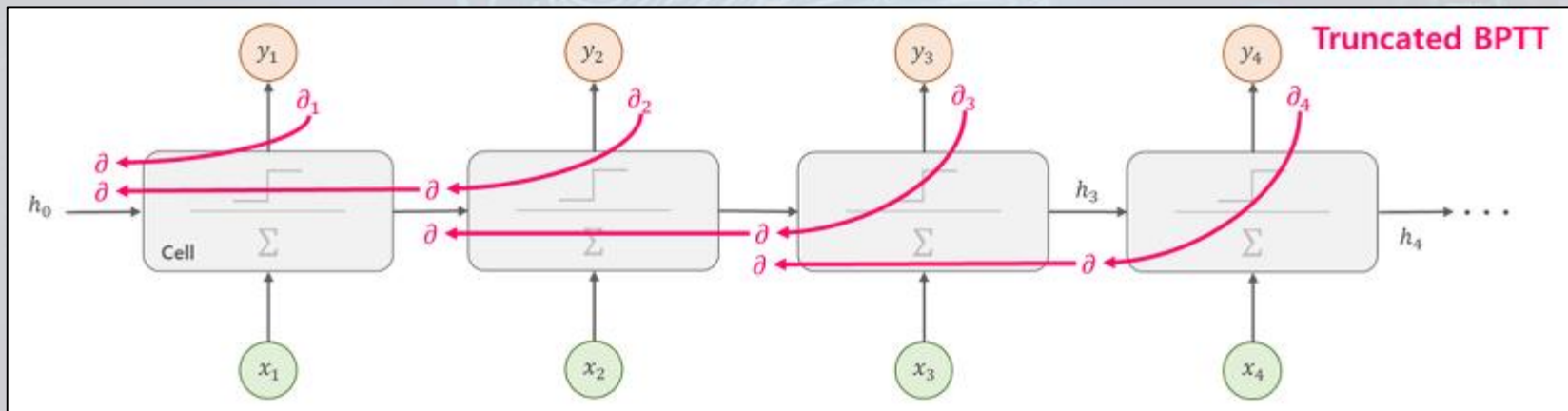
- RNNs의 목적은 **배열 (시계열) 데이터를 분류**하는 것으로 학습은 **오차역전파와 경사하강법**을 사용
- 일반적인 인공 신경망의 오차역전파는 출력단에서 구한 오차를 신경망을 거슬러 가면서 각 계수를 업데이트 하는 것
  - 각 계수가 출력단의 오차에 얼마나 기여하는지를 편미분  $\partial E / \partial w$ 으로 계산하고 이 값을 사용해 각 계수를 얼마나 증가 혹은 감소시킬지 결정
- RNNs는 역전파의 확장판인 **BPTT (Backpropagation Through Time)**을 사용해 계수를 학습
  - RNNs의 구조가 시간에 따라 연결되어 있기 때문에 역전파는 끝까지 시간을 거슬러 올라가며 적용





### RNN (Recurrent Neural Network)

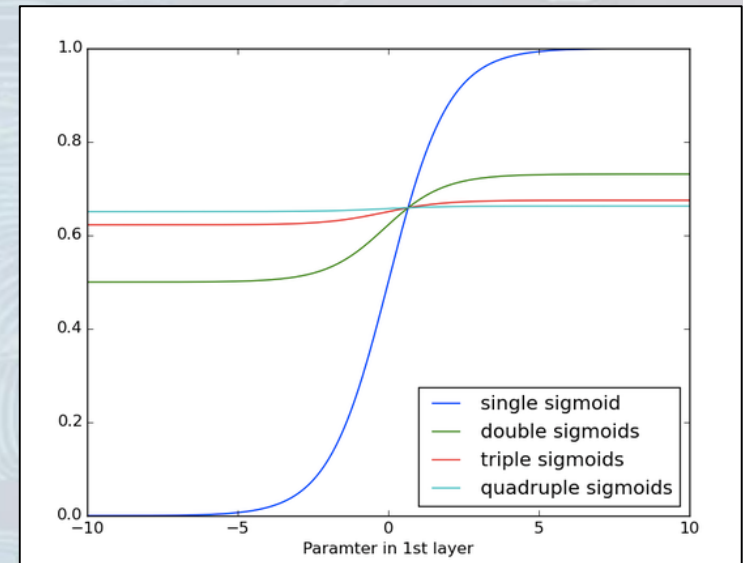
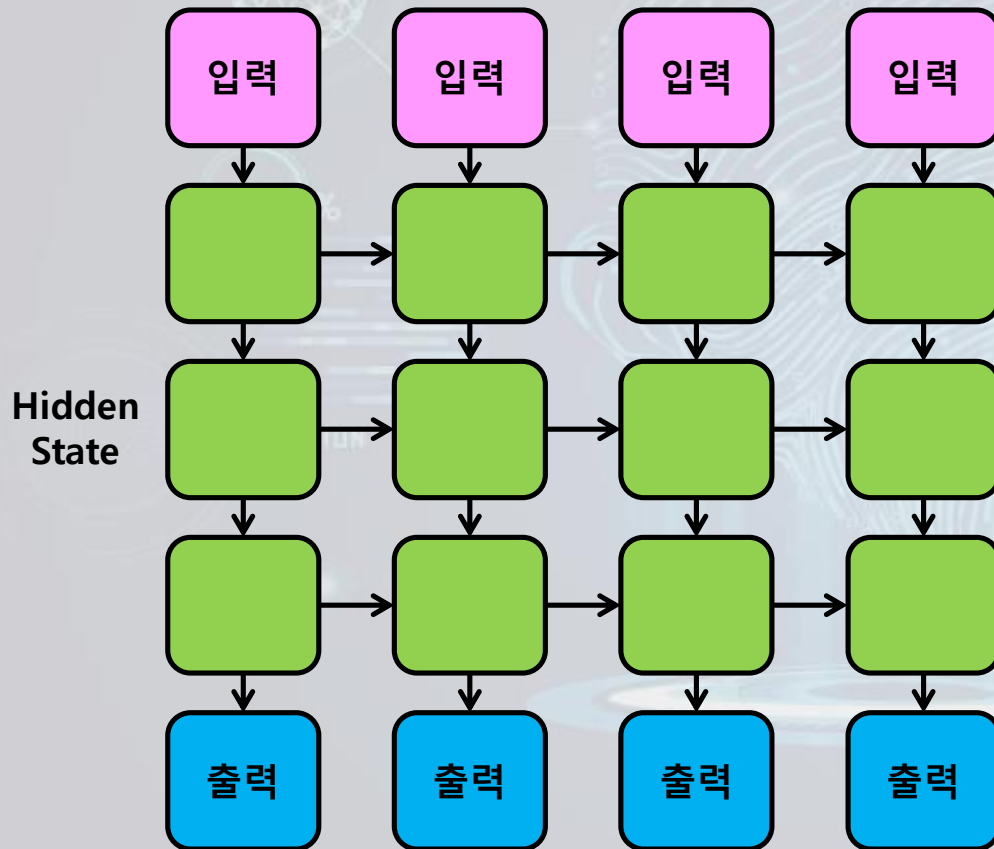
- **Truncated BPTT (단기 BPTT)** : 시간 전체를 거슬러 올라가는 BPTT를 간략화 한 것
  - 시계열 데이터가 길어지면 은닉층에 저장해야 하는 양이 계속 늘어나기 때문에 모든 시간에 대한 은닉층의 값을 저장하는 것은 현실적으로 불가능하므로 적당한 선에서 타협
  - 단기 BPTT를 사용하면 기준 길이(보통 5 steps)보다 오래된 값은 반영하지 않으므로 장기간에 걸친 패턴이라면 RNNs의 기억력이 짧아지는 문제가 있음





### RNN (Recurrent Neural Network)

● 은닉층 증가 → Vanishing Gradient 문제 발생 → LSTM, GRU를 사용하여 해결



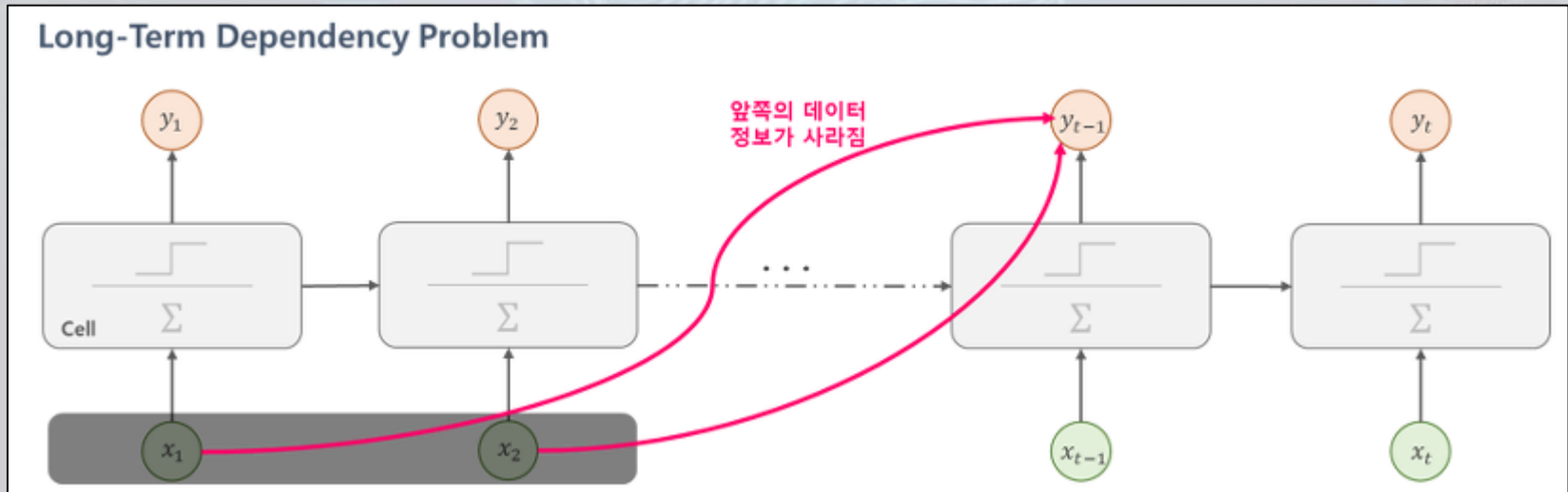
시그모이드 함수를 계속해서 곱하면 0에 가까워져 그레디언트가 제대로 전파되지 않음

# LSTM



### LSTM (Long Short-Term Memory)

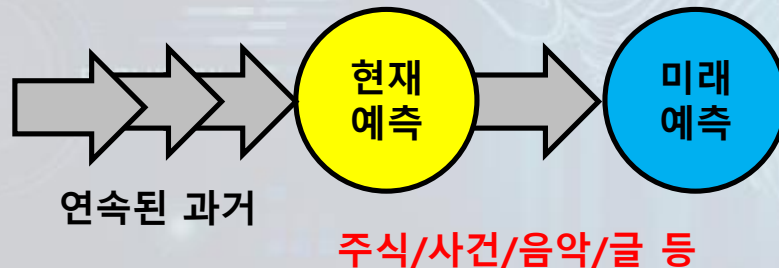
- RNN은 타임 스텝  $t$ 에서 이전 스텝  $(t-1)$ 의 상태 ( $h_{t-1}$ )를 입력으로 받는 구조이므로 이전의 정보가 현재 타임 스텝  $t$ 에 영향을 줌
- **메모리 셀 (Memory Cell)** : RNN의 최종 출력은 이전 타임 스텝의 모든 입력에 대한 함수를 의미
- **장기의존성 (Long-Term Dependency)** : RNN 타임스텝이 길어질수록 영향을 주지 못하는 문제 발생  
→ 입력데이터가 RNN Cell을 지나면서 특정 연산 (Sigmoid)에 의해 사라짐 (Vanishing Gradient)  
→ 장기간 메모리를 가질 수 있는 셀이 필요 → LSTM, GRU





### LSTM (Long Short-Term Memory)

- 기존 RNN의 경우, 정보와 정보 사이의 거리가 멀면, 초기의 Weight값이 유지되지 않아 학습능력이 저하 → LSTM은 과거의 data를 계속해서 update 하므로 RNN보다 지속적
- 장점 : 각각의 메모리 컨트롤이 가능하다는 점과 결과값이 컨트롤이 가능하다는 점
- 단점 : 메모리가 뒤편어씩워질 가능성이 있고, 연산속도가 느리다는 단점



시계열 데이터에는 여러가지 사건이 다른 시간 주기로 동시에 일어나며 LSTM은 이런 여러가지 요인 분석을 가능하게 함



전화한 이유 ?

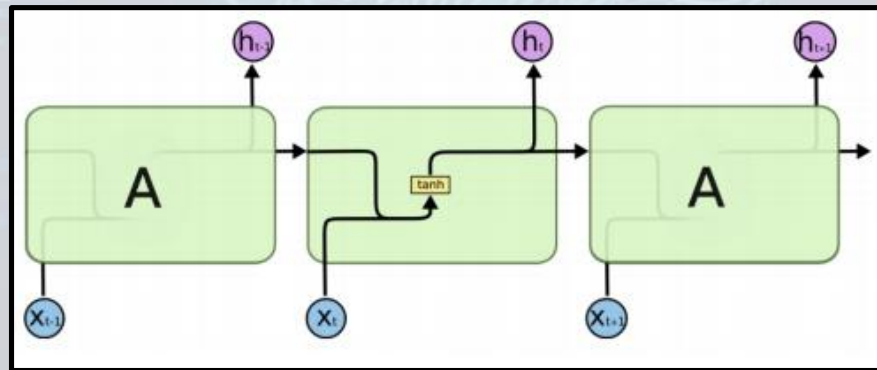
매 주말마다 전화한다면 항상 열린 게이트라고 할 수 있음





### LSTM (Long Short-Term Memory)

- 1997년에 등장하였으며 오차의 그래디언트가 1000단계가 넘는 시간을 거슬러 올라갈 수 있도록 함.
- 정보를 추가하거나 삭제하는 기능을 담당하는 **Cell State**라고 불리는 특징층을 하나 더 넣어 Weight를 계속 기억할 것인지 결정 → **Vanishing Gradient의 문제를 해결**
- LSTM은 3개의 gates(input, forget, output)로 현재 노드의 상태 정보를 제어
- Forget gate**는 이전 상태 정보를 저장할지를 결정하고, **Input gate**는 입력되는 새로운 정보를 저장할지 결정하고 **Output gate**는 갱신된 cell의 출력값을 제어 → 0~1 사이 값을 가짐 (Sigmoid)

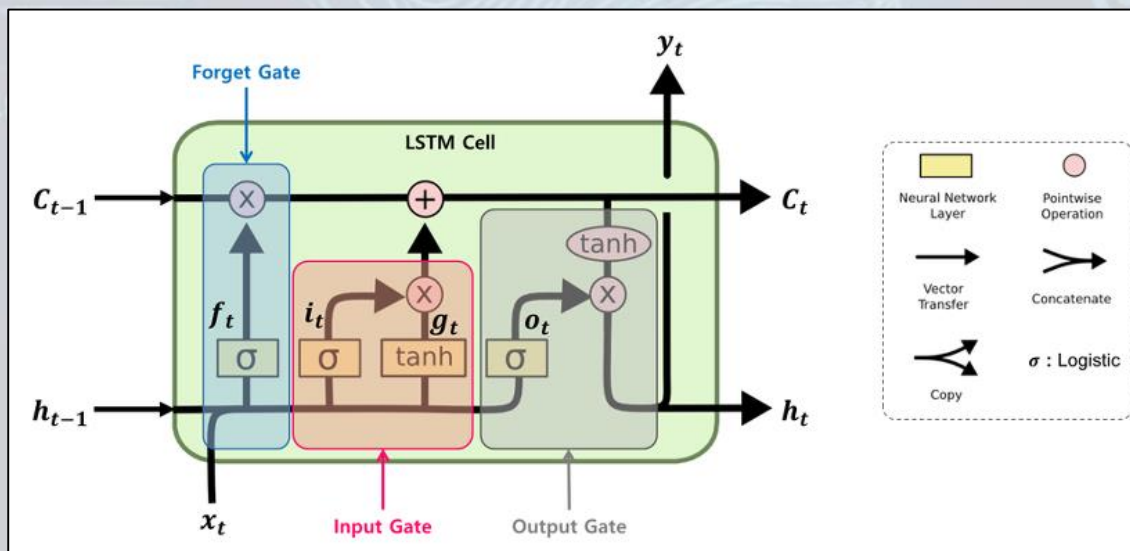


RNN



### LSTM (Long Short-Term Memory)

- $h$ 는 단기 상태 (Short-Term state)를  $c$ 는 장기 상태(Long-Term state)라고 볼 수 있음
- 장기 기억  $C_{t-1}$ 은 왼쪽에서 오른쪽으로 통과하면서 Forget gate를 지나면서 일부 정보를 잃고 덧셈 연산으로 Input gate로부터 덧셈 (+) 연산을 통해 새로운 정보를 추가
- 연산을 통해 만들어진  $C_t$ 는 바로 출력되고  $C_t$ 는 타입 스텝마다 일부 기억을 삭제하고 추가하는 과정을 거침
- 덧셈 연산 후에  $C_t$ 는 Output gate의 tanh 함수로 전달되어 단기 상태  $h_t$ 와 셀 출력  $y_t$ 를 만듦

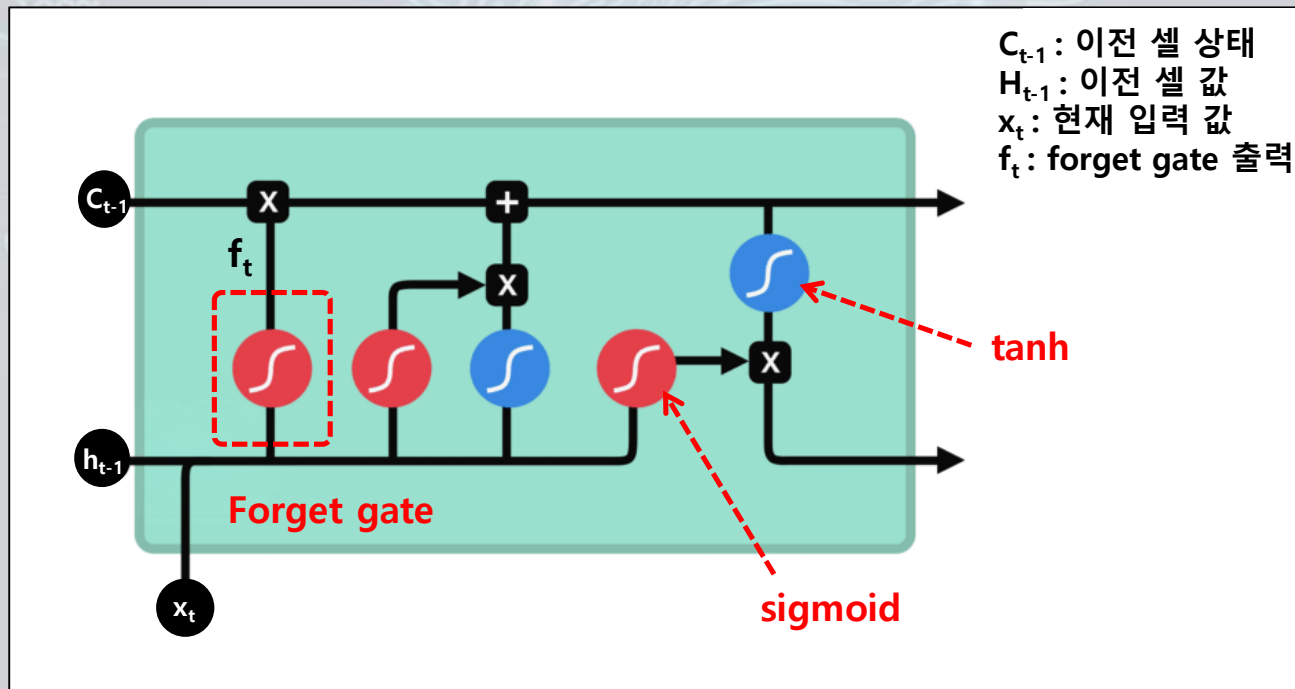




### LSTM (Long Short-Term Memory)

- **Forget gate** : 이전 정보를 저장할 것인지 결정 - 시그모이드 함수 사용, 1에 가까울 수록 저장
- $f_t$ 에 의해 제어되며 장기 상태  $C_t$ 의 어느 부분을 삭제할지 제어

$$f_t = \sigma(X_t W_{xh\_f} + H_{t-1} W_{hh\_f} + b_f)$$

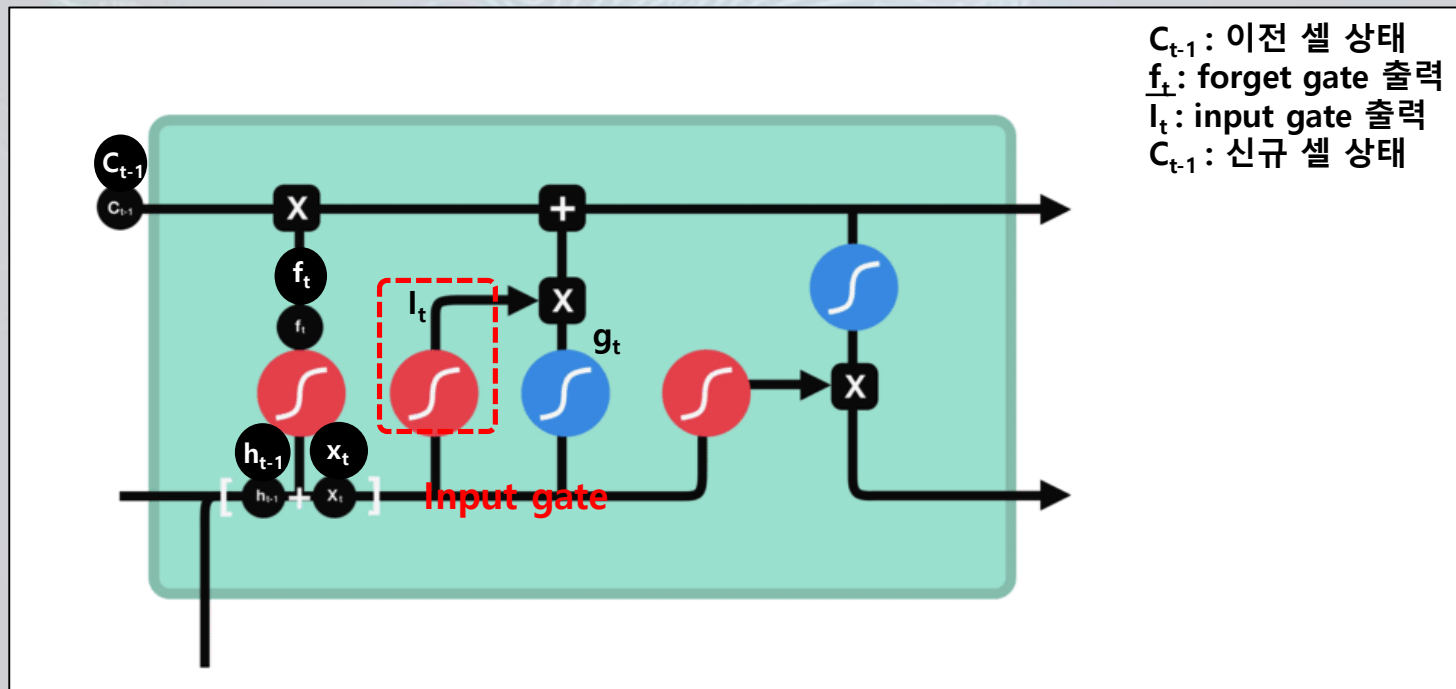




### LSTM (Long Short-Term Memory)

- **Input gate** : 입력되는 정보를 저장할 것인지 결정 - 시그모이드 함수 사용, 1에 가까울 수록 저장
- $I_t$ 에 의해 제어되며  $g_t$ 의 어느 부분이 장기 상태  $C_t$ 에 더해져야 되는지 제어

$$I_t = \sigma(X_t W_{xh\_i} + H_{t-1} W_{hh\_i} + b_i)$$

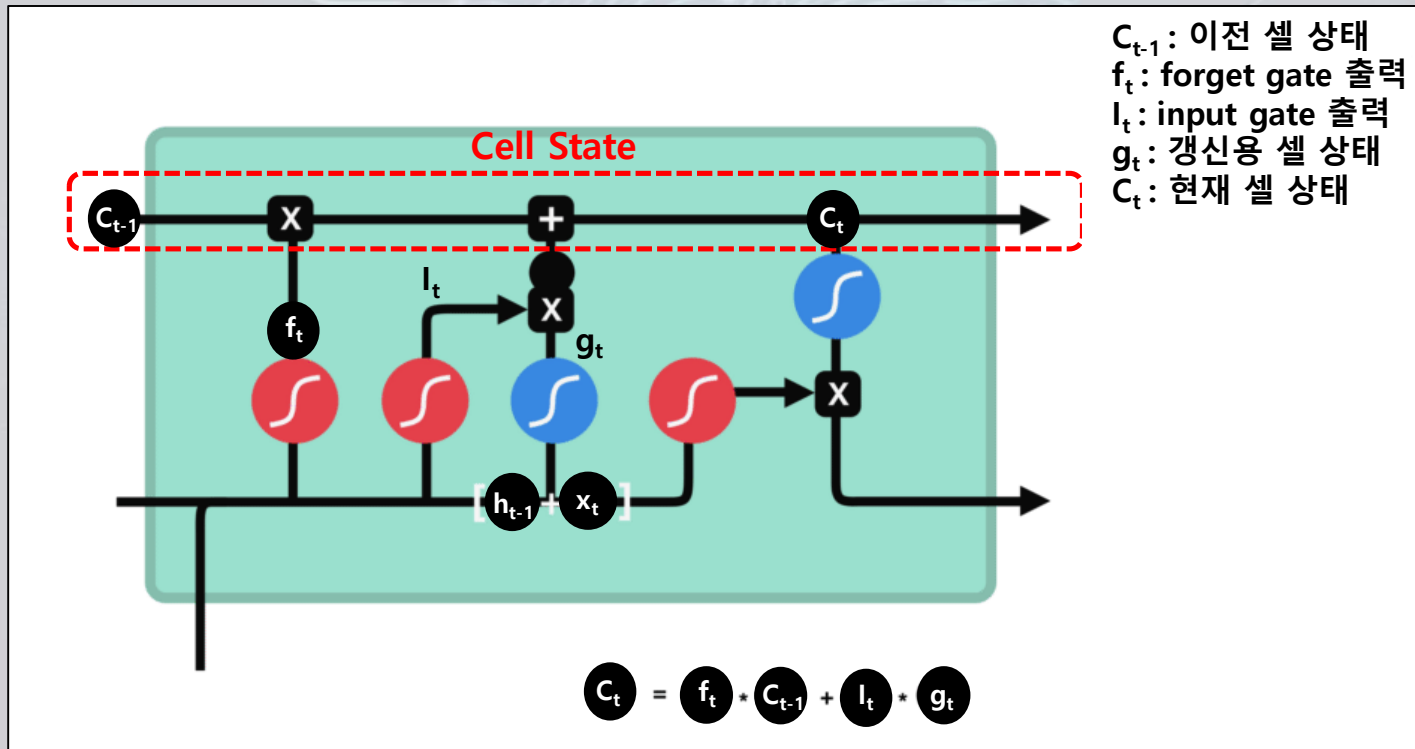




### LSTM (Long Short-Term Memory)

- Cell state : 정보를 추가하거나 삭제하는 기능을 수행

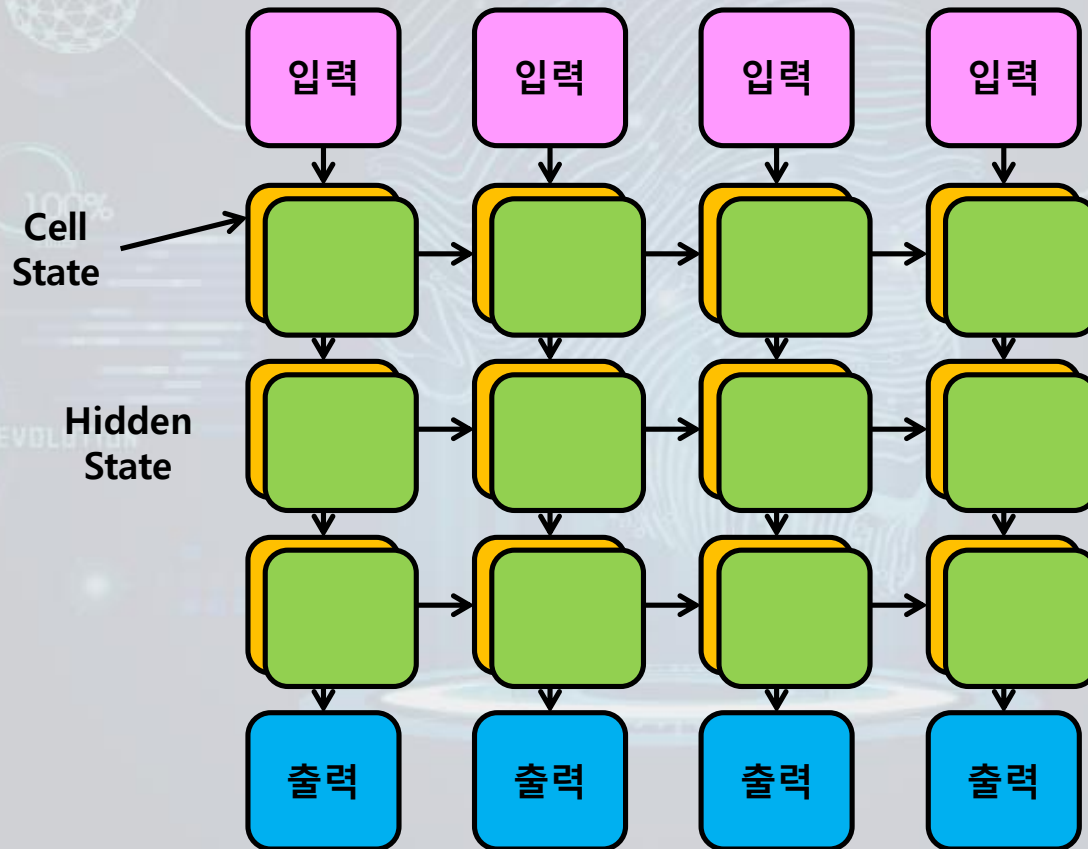
$$g_t = \tanh(X_t W_{xh\_g} + H_{t-1} W_{hh\_g} + b_c)$$





## LSTM (Long Short-Term Memory)

• Cell State





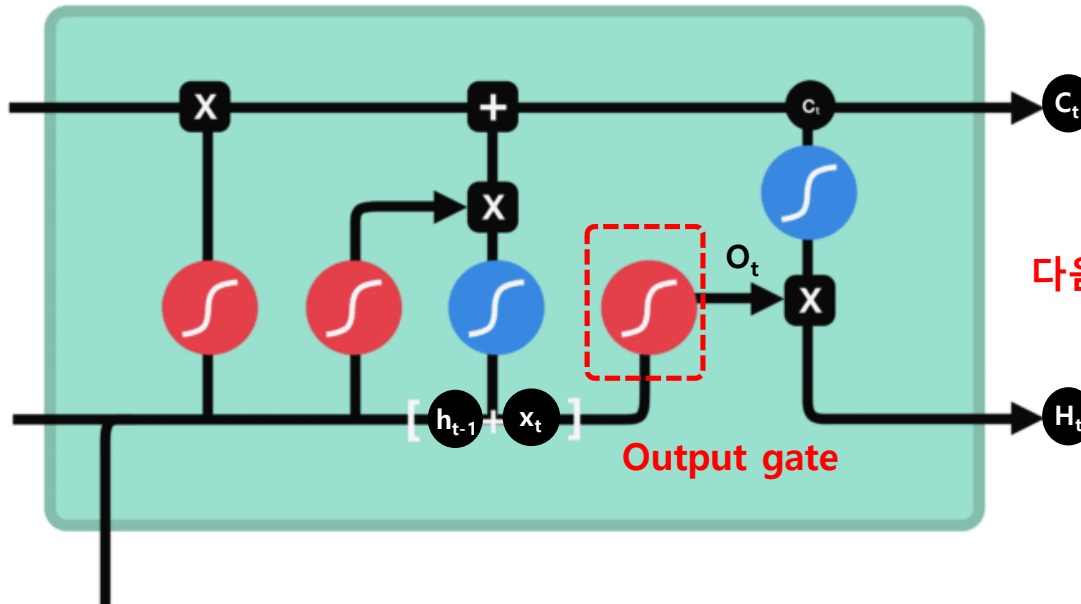
### LSTM (Long Short-Term Memory)

- **Output gate** : 갱신된 정보를 전달할 것인지 결정
- $O_t$ 는 장기 상태  $C_t$ 의 어느 부분을 읽어서  $h_t$ 와  $y_t$ 를 출력해야 하는지 제어

$$O_t = \sigma(X_t W_{xh_o} + H_{t-1} W_{hh_o} + b_o)$$

$$C_t = f_t C_{t-1} + I_t g_t$$

$$H_t = O_t \tanh(C_t)$$



$C_{t-1}$ : 이전 셀 상태  
 $f_t$ : forget gate 출력  
 $I_t$ : input gate 출력  
 $C_t$ : 갱신용 셀 상태  
 $g_t$ : 현재 셀 상태  
 $O_t$ : output gate 출력  
 $H_t$ : 은닉 상태

다음 노드로 전달





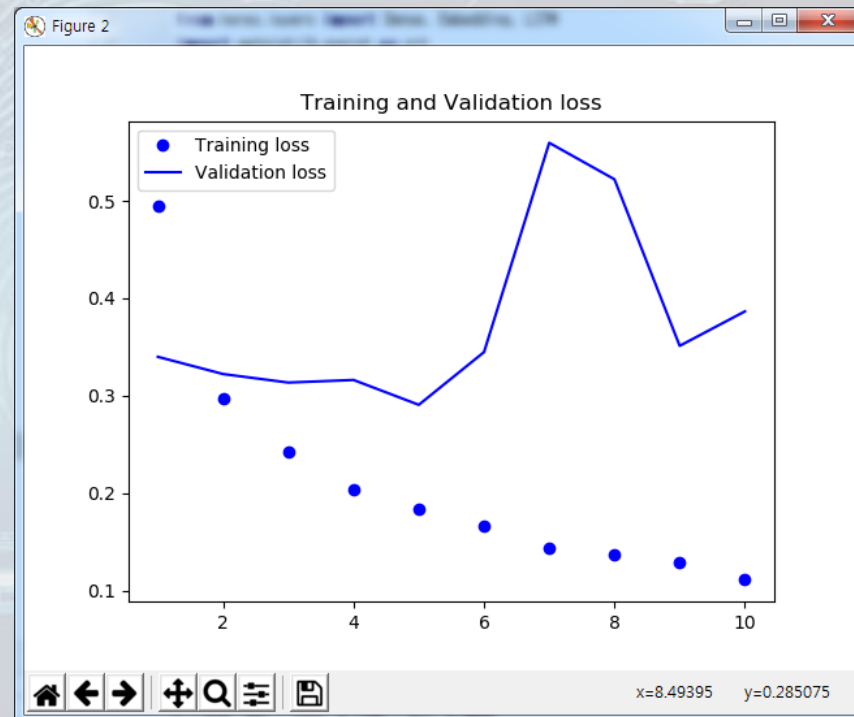
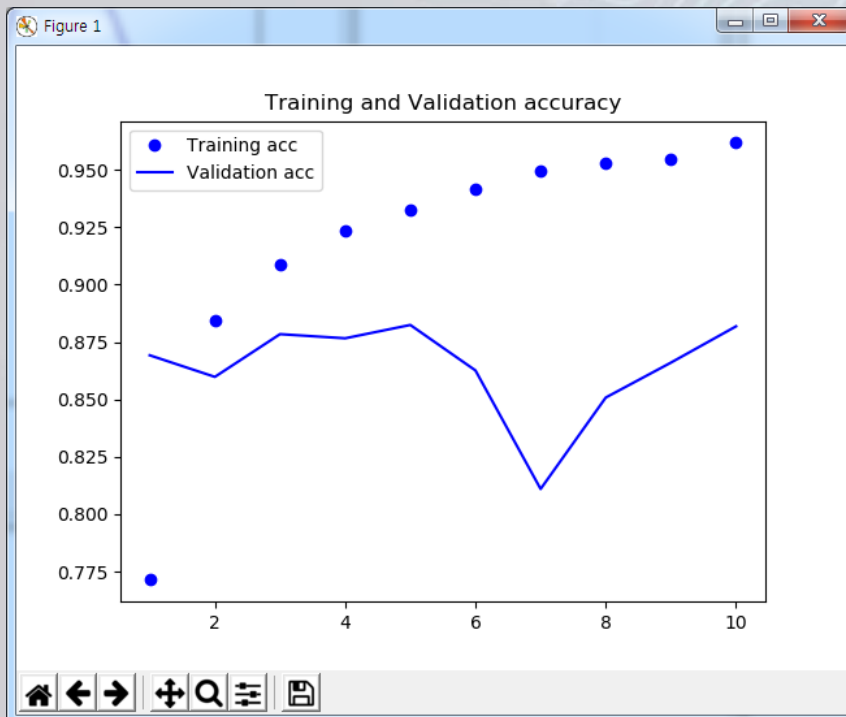
## IMDB 데이터셋 적용하기

```
...  
4  from keras.layers import Dense, Embedding, LSTM  
...  
15 model = Sequential()  
16 model.add(Embedding(max_features, 32))  
17 model.add(LSTM(32))  
18 model.add(Dense(1, activation='sigmoid'))  
19 model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])  
20  
21 history = model.fit(x_train, y_train, epochs=10, batch_size=128, validation_split=0.2)  
22 model.save('LSTM_model.h5')
```



## IMDB 데이터셋 적용하기

loss: 0.1109 - acc: 0.9617 - val\_loss: 0.3866 - val\_acc: 0.8818





### LSTM (Long Short-Term Memory)

- 임베딩 차원이나 LSTM 출력 차원 같은 하이퍼파라미터를 튜닝하거나 규제를 사용하여 성능 향상 가능
- 리뷰를 전체적으로 길게 분석하는 것은 감성 분류 문제에 적합하지 않으므로 정확도가 떨어질 수 있음
- LSTM은 질문-응답과 기계 번역 분야에서 뛰어난 성능을 발휘함



## LSTM (Long Short-Term Memory)

- 학습된 모델로 예측하기 : 10개의 데이터에 대해 예측

```
1 from keras.datasets import imdb
2 from keras.models import load_model
3 from keras.preprocessing import sequence
4 import numpy as np
5
6 max_features = 10000
7 maxlen = 500
8
9 (x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_features)
10 x_train = sequence.pad_sequences(x_train, maxlen=maxlen)
11 x_test = sequence.pad_sequences(x_test, maxlen=maxlen)
12
13 model = load_model('LSTM_model.h5')
14
15 Y_prediction = model.predict(x_test[:10])
16 prediction = np.where(Y_prediction > .5, "긍정", "부정")
17
18 for i in range(10):
19     print("%s --> %s" % (y_test[i:i+1], prediction[i]))
```



### LSTM (Long Short-Term Memory)

• 학습된 모델로 예측하기 : 10개의 데이터에 대해 예측

```
[0] --> ['부정']  
[1] --> ['긍정']  
[1] --> ['부정']  
[0] --> ['부정']  
[1] --> ['긍정']  
[1] --> ['긍정']  
[1] --> ['긍정']  
[0] --> ['부정']  
[0] --> ['긍정']  
[1] --> ['긍정']
```



### LSTM과 CNN의 조합을 이용한 영화 리뷰 분류하기 (IMDB)

#### • 라이브러리 업로드

```
1 from keras.datasets import imdb
2 from keras.preprocessing import sequence
3 from keras.models import Sequential
4 from keras.layers import Dense, Embedding, LSTM, Dropout, Conv1D, MaxPooling1D
5 import numpy as np
6 import tensorflow as tf
7 import matplotlib.pyplot as plt
```



### LSTM과 CNN의 조합을 이용한 영화 리뷰 분류하기 (IMDB)

#### • 데이터 전처리

```
9 seed = 0
10 np.random.seed(seed)
11 tf.set_random_seed(seed)
12
13 (x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=5000)
14 x_train = sequence.pad_sequences(x_train, maxlen=100)
15 x_test = sequence.pad_sequences(x_test, maxlen=100)
```



### LSTM과 CNN의 조합을 이용한 영화 리뷰 분류하기 (IMDB)

#### ● 모델 설정

```
17 model = Sequential()  
18 model.add(Embedding(5000, 100))  
19 model.add(Dropout(0.5))  
20 model.add(Conv1D(64, 5, padding='valid', activation='relu', strides=1))  
21 model.add(MaxPooling1D(pool_size=4))  
22 model.add(LSTM(55))  
23 model.add(Dense(1, activation='sigmoid'))
```





### LSTM과 CNN의 조합을 이용한 영화 리뷰 분류하기 (IMDB)

#### ● 모델 설정

Layer (type)	Output Shape	Param #
embedding_1 (Embedding)	(None, None, 100)	500000
dropout_1 (Dropout)	(None, None, 100)	0
conv1d_1 (Conv1D)	(None, None, 64)	32064
max_pooling1d_1 (MaxPooling1D)	(None, None, 64)	0
lstm_1 (LSTM)	(None, 55)	26400
dense_1 (Dense)	(None, 1)	56

Total params: 558,520

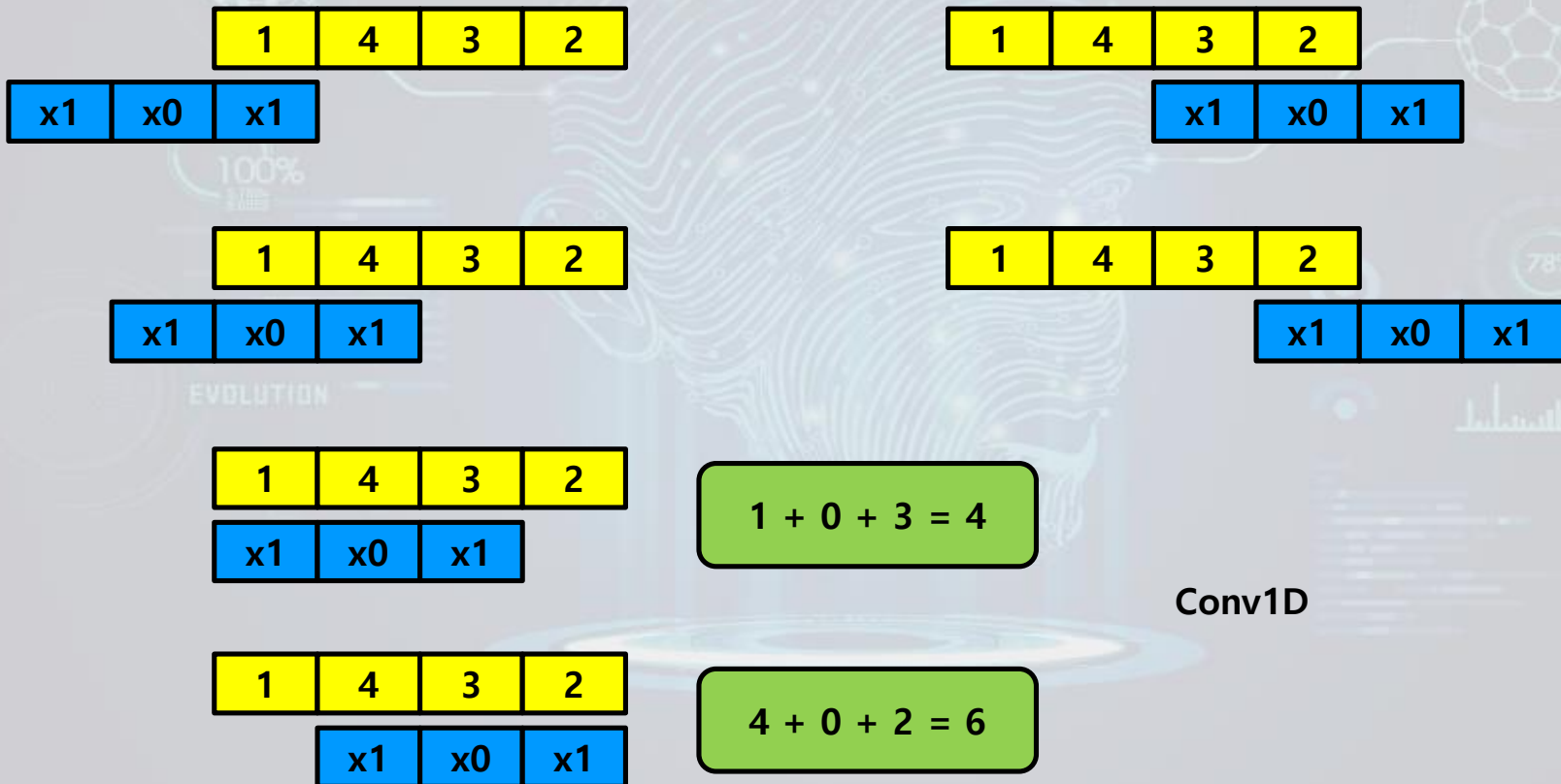
Trainable params: 558,520

Non-trainable params: 0



### LSTM과 CNN의 조합을 이용한 영화 리뷰 분류하기 (IMDB)

- 데이터가 1차원 데이터이므로 Conv1D와 MaxPooling1D를 사용





### LSTM과 CNN의 조합을 이용한 영화 리뷰 분류하기 (IMDB)

#### • MaxPooling1D





### LSTM과 CNN의 조합을 이용한 영화 리뷰 분류하기 (IMDB)

#### ● 모델 평가

```
25 model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['acc'])
26
27 history = model.fit(x_train, y_train, epochs=5, batch_size=100,
28 validation_data=(x_test, y_test))
29 model.save('LSTM_CNN_model.h5')
30
31 print("\n Test Accuracy %.4f" % (model.evaluate(x_test, y_test)[1]))
```

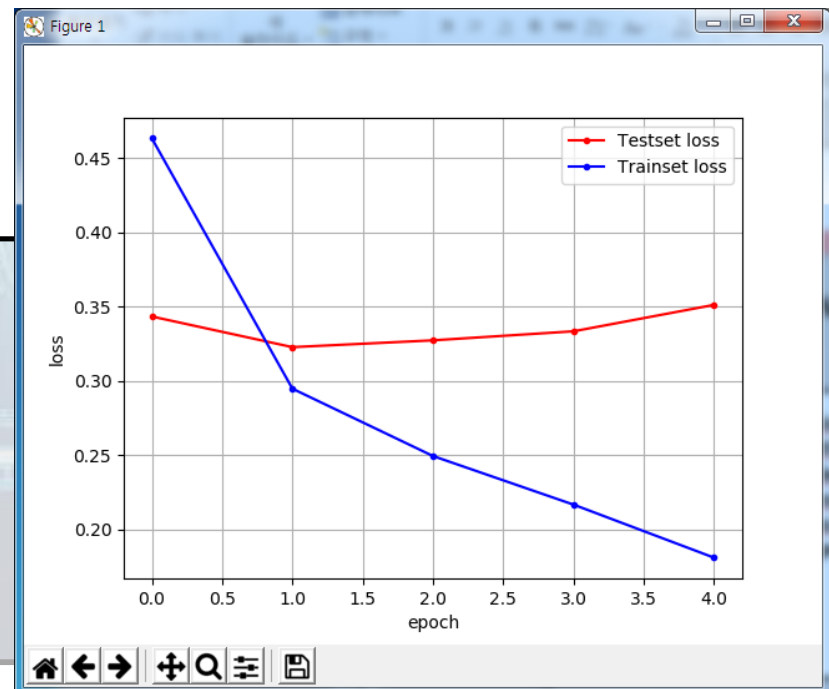
Test Accuracy 0.8532



### LSTM과 CNN의 조합을 이용한 영화 리뷰 분류하기 (IMDB)

#### ● 그래프 출력

```
33 y_vloss = history.history['val_loss']
34 y_loss = history.history['loss']
35 x_len = np.arange(len(y_loss))
36 plt.plot(x_len, y_vloss, marker='.', c='red', label='Testset loss')
37 plt.plot(x_len, y_loss, marker='.', c='blue', label='Trainset loss')
38 plt.legend(loc='upper right')
39 plt.grid()
40 plt.xlabel('epoch')
41 plt.ylabel('loss')
42 plt.show()
```





## 로이터 뉴스 카테고리 분류하기

### 로이터 뉴스 데이터

- 11,258개의 뉴스 기사를 46개의 카테고리로 구분된 텍스트 데이터

• 입력된 문장의 의미를 파악하는 것 → 모든 단어를 종합하여 하나의 카테고리로 분류하는 것

중부 지방은 대체로 맑겠으나, 남부 지방은 구름이 많겠습니다

날씨

올 초부터 유동성의 힘으로 주가가 일정하게 상승했습니다

주식

이번 선거는 누가 이길 것 같아 ?

정치

퍼셉트론의 한계를 극복한 신경망이 다시 뜨고 있대

딥러닝

오늘 날씨가 참 좋네

인사



### MLP를 이용한 로이터 뉴스 카테고리 분류하기

#### • 라이브러리 로드

```
1 from keras.datasets import reuters
2 from keras.preprocessing import sequence
3 from keras.utils.np_utils import to_categorical
4 from keras.models import Sequential
5 from keras.layers import Dense, Embedding, Flatten
6 import matplotlib.pyplot as plt
7 import numpy as np
```

#### • 데이터 로드

```
9 max_features = 1000
10 text_max_words = 100
11
12 # 빈도수가 1-1000에 해당하는 단어만 선택하고 20%를 테스트 셋으로 설정
13 (X_train, Y_train), (X_test, Y_test) = reuters.load_data(num_words=max_features,
14 test_split=0.2)
15
16 # 단어 수를 100개로 맞춤
17 x_train = sequence.pad_sequences(X_train, maxlen=text_max_words)
18 x_test = sequence.pad_sequences(X_test, maxlen=text_max_words)
```



### MLP를 이용한 로이터 뉴스 카테고리 분류하기

#### 원-핫 인코딩

```
19 y_train = to_categorical(Y_train)
20 y_test = to_categorical(Y_test)
```

#### 모델 설정

- Embedding 레이어는 0에서 45의 정수값 (카테고리 종류)으로 지정된 단어를 128벡터로 인코딩
- 문장의 길이가 100이므로 Embedding 레이어는 128 속성을 가진 벡터를 100개 반환
- Flatten 레이어를 통해 1차원 벡터로 만든 뒤 전결합층으로 전달
- 46개 주제를 분류해야 하므로 출력층의 활성화 함수로 softmax를 사용

```
22 model = Sequential()
23 model.add(Embedding(max_features, 128, input_length=text_max_words))
24 model.add(Flatten())
25 model.add(Dense(256, activation='relu'))
26 model.add(Dense(46, activation='softmax'))
```





### MLP를 이용한 로이터 뉴스 카테고리 분류하기

• 모델 컴파일 / 실행 / 평가하기

```
28 model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['acc'])
29 history = model.fit(x_train, y_train, epochs=10, batch_size=64,
30                     validation_data=(x_test, y_test))
31 model.save_weights('MLP_routers_model.h5')
32 print("\n Test Accuracy %.4f" % (model.evaluate(x_test, y_test)[1]))
```

11s 1ms/step

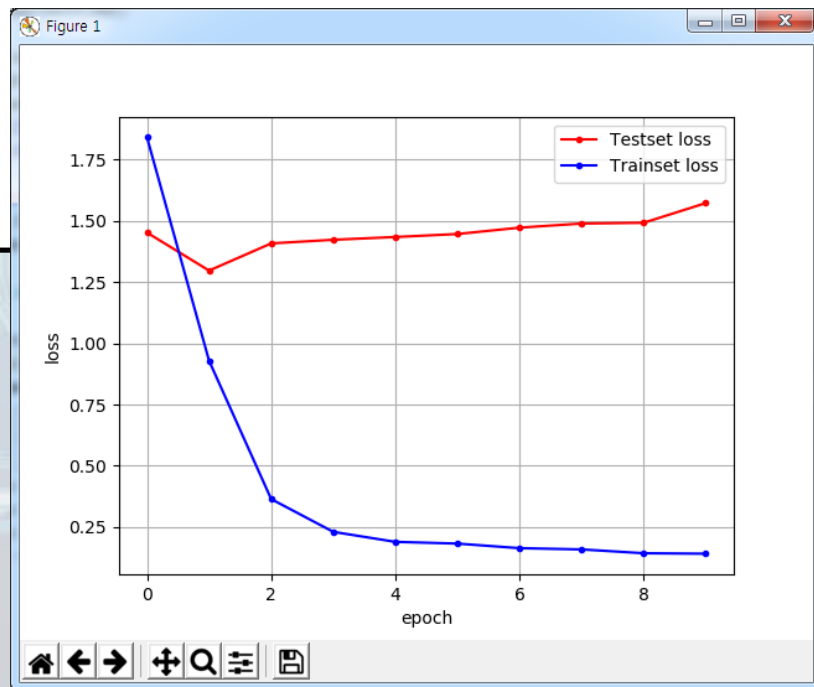
Test Accuracy 0.6759



### MLP를 이용한 로이터 뉴스 카테고리 분류하기

#### 결과 시각화

```
34 y_vloss = history.history['val_loss']
35 y_loss = history.history['loss']
36 x_len = np.arange(len(y_loss))
37 plt.plot(x_len, y_vloss, marker='.', c='red', label='Testset loss')
38 plt.plot(x_len, y_loss, marker='.', c='blue', label='Trainset loss')
39 plt.legend(loc='upper right')
40 plt.grid()
41 plt.xlabel('epoch')
42 plt.ylabel('loss')
43 plt.show()
```





### LSTM을 이용한 로이터 뉴스 카테고리 분류하기

#### • 라이브러리 로드

```
1 from keras.datasets import reuters
2 from keras.preprocessing import sequence
3 from keras.utils.np_utils import to_categorical
4 from keras.models import Sequential
5 from keras.layers import Dense, Embedding, LSTM
6 import matplotlib.pyplot as plt
7 import numpy as np
```

#### • 모델 설정

- Embedding 레이어에서 반환되는 100개 벡터를 LSTM의 타임스텝으로 입력하는 모델
- LSTM의 input\_dim은 Embedding 레이어에서 인코딩된 벡터크기는 128

```
22 model = Sequential()
23 model.add(Embedding(max_features, 128))
24 model.add(LSTM(128))
25 model.add(Dense(46, activation='softmax'))
```



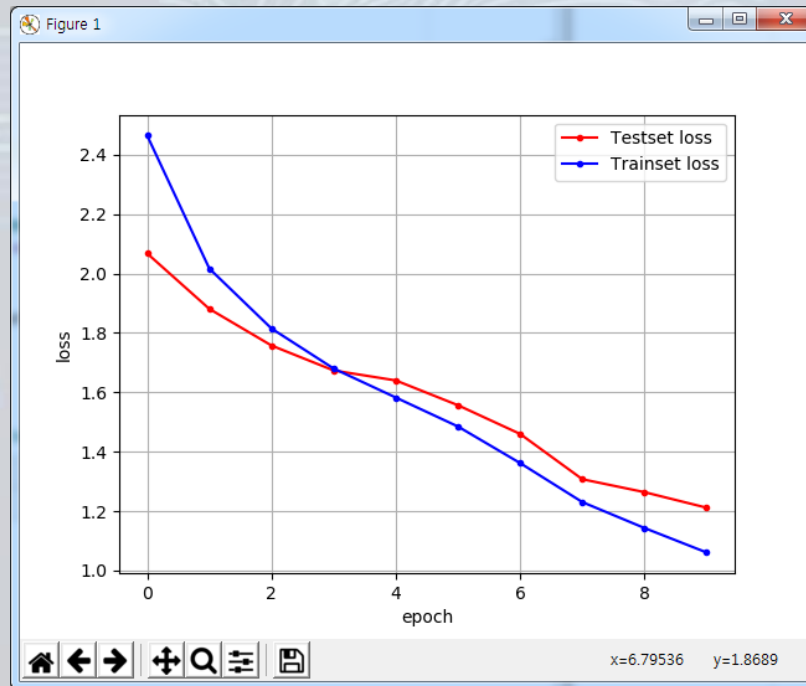
### LSTM을 이용한 로이터 뉴스 카테고리 분류하기

- 모델 컴파일 / 실행 / 평가하기

23s 3ms/step

Test Accuracy 0.6990

- 결과 시각화





### CNN을 이용한 로이터 뉴스 카테고리 분류하기

#### • 라이브러리 로드

```
1 from keras.datasets import reuters
2 from keras.preprocessing import sequence
3 from keras.utils.np_utils import to_categorical
4 from keras.models import Sequential
5 from keras.layers import Dense, Embedding, Dropout
6 from keras.layers import Conv1D, GlobalMaxPooling1D
7 import matplotlib.pyplot as plt
8 import numpy as np
```



### CNN을 이용한 로이터 뉴스 카테고리 분류하기

#### ● 모델 설정

- Embedding 레이어에서 반환되는 100개 벡터를 컨볼루션 필터를 적용한 모델
- 필터크기가 3인 Conv1D 레이어는 100개의 벡터를 입력받아 128개의 벡터를 반환
- 벡터 크기는 Conv1D 레이어를 통과하면서 128개에서 256개로 증가
- GlobalMaxPooling1D 레이어는 입력되는 128개 벡터 중 가장 큰 벡터 하나를 반환
- 벡터 하나를 전결합층을 통해서 다중클래스를 분류

```
23 model = Sequential()
24 model.add(Embedding(max_features, 128, input_length=text_max_words))
25 model.add(Dropout(0.2))
26 model.add(Conv1D(256, 3, padding='valid', activation='relu', strides=1))
27 model.add(GlobalMaxPooling1D()) #여러 개의 벡터 중 최대 벡터 선택
28 model.add(Dense(128, activation='relu'))
29 model.add(Dropout(0.2))
30 model.add(Dense(46, activation='softmax'))
```



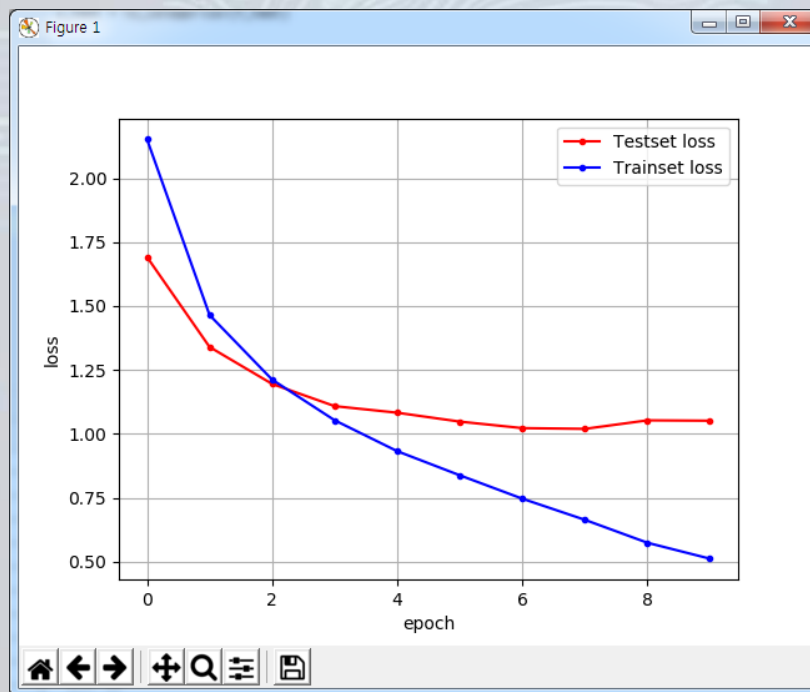
### CNN을 이용한 로이터 뉴스 카테고리 분류하기

- 모델 컴파일 / 실행 / 평가하기

17s 2ms/step

Test Accuracy 0.7493

- 결과 시각화





### CNN+LSTM을 이용한 로이터 뉴스 카테고리 분류하기

#### • 라이브러리 로드

```
1 from keras.datasets import reuters
2 from keras.preprocessing import sequence
3 from keras.utils.np_utils import to_categorical
4 from keras.models import Sequential
5 from keras.layers import Dense, Embedding, Dropout, LSTM, Flatten
6 from keras.layers import Conv1D, MaxPooling1D
7 import matplotlib.pyplot as plt
8 import numpy as np
```





### CNN+LSTM을 이용한 로이터 뉴스 카테고리 분류하기

#### ● 모델 설정

- Conv1D 레이어에서 나온 특징벡터들을 MaxPooling1D을 통해 1/4로 줄여준 다음 LSTM의 입력으로 넣어주는 모델

```
23 model = Sequential()
24 model.add(Embedding(max_features, 128, input_length=text_max_words))
25 model.add(Dropout(0.2))
26 model.add(Conv1D(256, 3, padding='valid', activation='relu', strides=1))
27 model.add(MaxPooling1D(pool_size=4))
28 model.add(LSTM(128))
29 model.add(Dense(46, activation='softmax'))
```



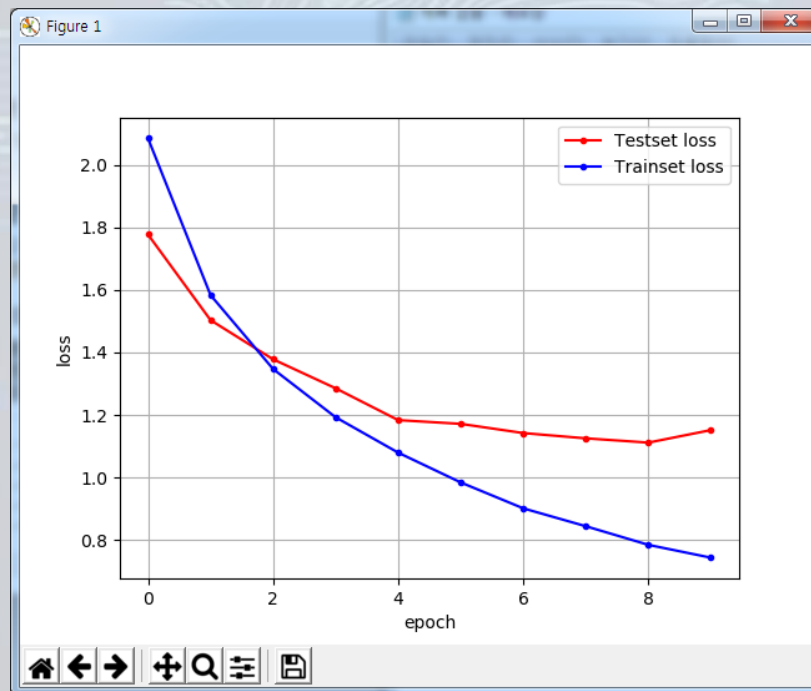
### CNN+LSTM을 이용한 로이터 뉴스 카테고리 분류하기

- 모델 컴파일 / 실행 / 평가하기

20s 2ms/step

Test Accuracy 0.7186

- 결과 시각화





### 악보 학습

- 악보는 시계열 데이터이고 음계가 문장보다 코드화가 쉬움
- 음의 코드와 음의 길이로 데이터 구성
  - c(도), d(레), e(미), f(파), g(솔), a(라), b(시)
  - 4(4분음표), 8(8분음표)

나 비 야

g8 e8 e4 f8 d8 d4 c8 d8 e8 f8 g8 g8 g4

5 g8 e8 e8 e8 f8 d8 d4 c8 e8 g8 g8 e8 e8 e4

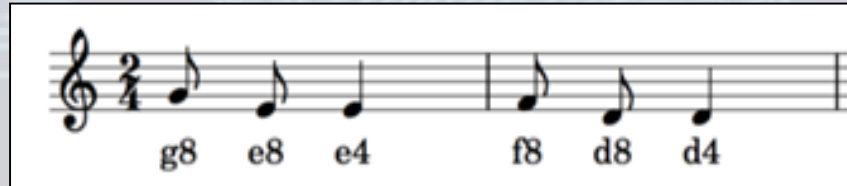
9 d8 d8 d8 d8 d8 e8 f4 e8 e8 e8 e8 e8 f8 g4

13 g8 e8 e4 f8 d8 d4 c8 e8 g8 g8 e8 e8 e4



### 악보 학습

- 첫 두 마디(g8 e8 e4 f8 d8 d4)에서 4개 음표 입력으로 다음 음표를 예측하려면
  - g8 e8 e4 f8 d8 : 1~4번째 음표, 5번째 음표
  - e8 e4 f8 d8 d4 : 2~5번째 음표, 6번째 음표
- 2개의 샘플은 4개의 입력 (feature, 속성)과 1개의 라벨 값(class)으로 구성
- 윈도우 크기 : 4



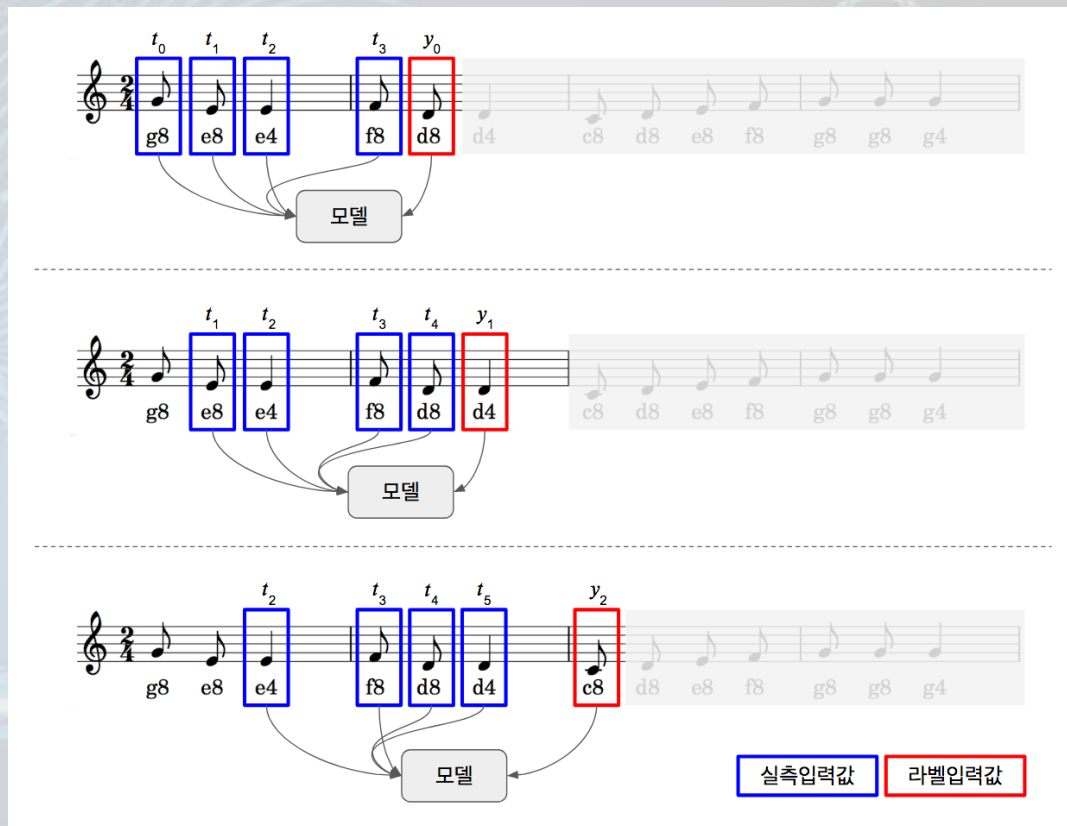


### 악보 학습 - 학습 과정

● 첫 4개 음표를 입력하면 나머지를 연주할 수 있는 모델을 만드는 것이 목표

#### ● 학습 과정

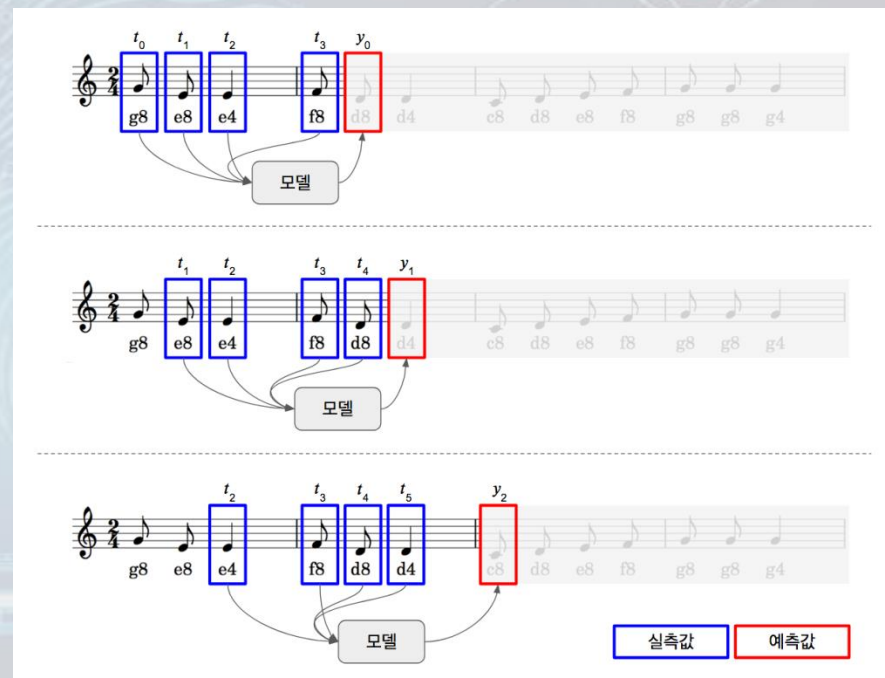
- 파란색 박스가 입력값이고, 빨간색 박스가 원하는 출력값
- 1~4번째 음표를 데이터로 5번째 음표를 라벨값으로 학습시킴
- 다음에는 2~5번째 음표를 데이터로 6번째 음표를 라벨값으로 학습시킴
- 이후 한 음표씩 넘어가면서 노래 끝까지 학습시킴





### 악보 학습 - 예측 과정

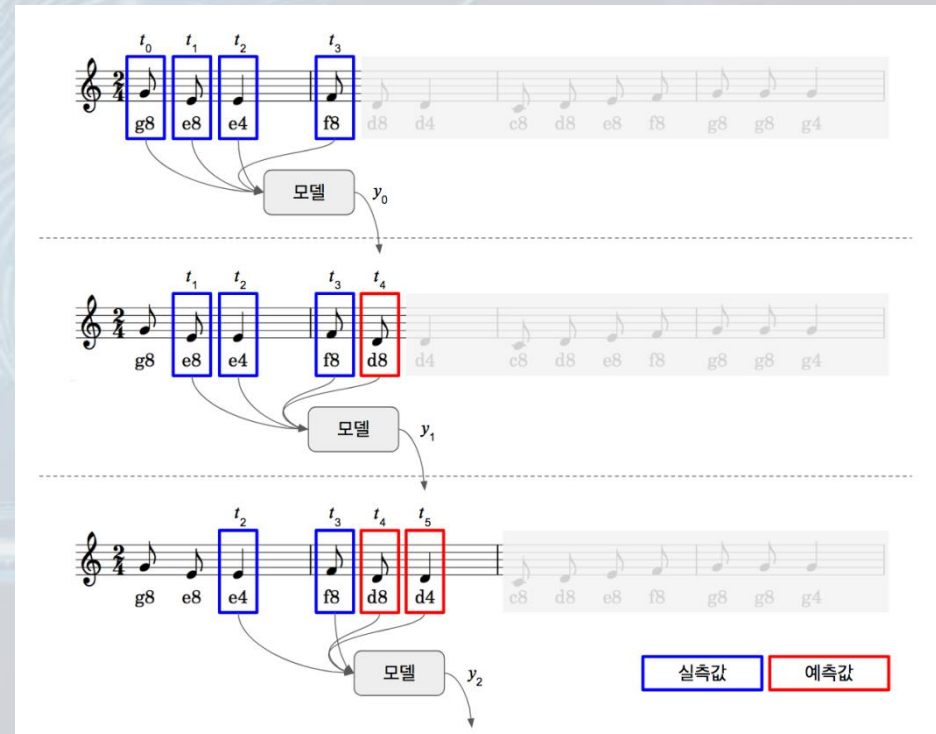
- 한 스텝 예측 : 실제 음표 4개를 입력하여 다음 음표 1개를 예측하는 것을 반복하는 것
- 모델의 입력값으로는 항상 실제 음표가 들어감.
  - 모델에  $t_0, t_1, t_2, t_3$ 를 입력하면  $y_0$  출력이 나옴
  - 모델에  $t_1, t_2, t_3, t_4$ 를 입력하면  $y_1$  출력이 나옴
  - 모델에  $t_2, t_3, t_4, t_5$ 를 입력하면  $y_2$  출력이 나옴
  - 이 과정을  $y_{49}$  출력까지 반복





### 악보 학습 - 예측 과정

- **곡 전체 예측** : 초기의 4개 음표만을 입력으로 곡 전체를 예측하는 것으로 초반부가 지나면, 예측값만으로 모델에 입력되어 다음 예측값이 나오는 방식 ("나비야~ 나"까지 알려주면 나머지 모두 연주)
- **예측 과정**
  - 모델에  $t_0, t_1, t_2, t_3$ 를 입력하면  $y_0$  출력이 나옴
  - 예측값인  $y_0$ 를  $t_4$ 라고 가정하고, 모델에  $t_1, t_2, t_3, t_4$ 을 입력하면  $y_1$  출력이 나옴
  - 예측값인  $y_1$ 을  $t_5$ 라고 가정하고, 모델에  $t_2, t_3, t_4$ (예측값),  $t_5$ (예측값)을 입력하면  $y_2$  출력이 나옴
  - 이 과정을  $y_{49}$  출력까지 반복





### 악보 학습 – 기본 LSTM

#### • 라이브러리 로딩, 시드 고정

```
1 import keras
2 import numpy as np
3 from keras.models import Sequential
4 from keras.layers import Dense, LSTM
5 from keras.utils import np_utils
6
7 # 랜덤시드 고정시키기
8 np.random.seed(5)
```





### 악보 학습 - 기본 LSTM

- 손실 이력 클래스 정의 : epoch가 끝날 때마다 losses에 loss 값을 저장

```
10 class LossHistory(keras.callbacks.Callback):
11     def init(self):
12         self.losses = []
13
14     def on_epoch_end(self, batch, logs={}):
15         self.losses.append(logs.get('loss'))
```

- 데이터 셋 생성 함수 : 시퀀스열을 받아서 window\_size 크기로 분리하고 분리된 문자에 맞는 값을 code2idx에서 찾아 dataset에 저장하여 반환

```
17 def seq2dataset(seq, window_size):
18     dataset = []
19     for i in range(len(seq)-window_size):
20         subset = seq[i:(i+window_size+1)]
21         dataset.append([code2idx[item] for item in subset])
22     return np.array(dataset)
```



### 악보 학습 - 기본 LSTM

- 코드 사전 정의 : 음표는 모델 입출력으로 사용할 수 없기 때문에 각 코드를 숫자로 변환

```

24 code2idx = {'c4':0, 'd4':1, 'e4':2, 'f4':3, 'g4':4, 'a4':5, 'b4':6,
25             'c8':7, 'd8':8, 'e8':9, 'f8':10, 'g8':11, 'a8':12, 'b8':13}
26
27 idx2code = {0:'c4', 1:'d4', 2:'e4', 3:'f4', 4:'g4', 5:'a4', 6:'b4',
28             7:'c8', 8:'d8', 9:'e8', 10:'f8', 11:'g8', 12:'a8', 13:'b8'}
```

- seq 변수에 곡 전체 음표를 저장하고 seq2dataset() 함수를 이용하여 데이터셋 생성

```

30 seq = ['g8', 'e8', 'e4', 'f8', 'd8', 'd4', 'c8', 'd8', 'e8', 'f8', 'g8', 'g8', 'g4',
31        'g8', 'e8', 'e8', 'e8', 'f8', 'd8', 'd4', 'c8', 'e8', 'g8', 'g8', 'e8', 'e8', 'e4',
32        'd8', 'd8', 'd8', 'd8', 'd8', 'e8', 'f4', 'e8', 'e8', 'e8', 'e8', 'e8', 'f8', 'g4',
33        'g8', 'e8', 'e4', 'f8', 'd8', 'd4', 'c8', 'e8', 'g8', 'g8', 'e8', 'e8', 'e4']
34
35 dataset = seq2dataset(seq, window_size = 4)
36
37 print(dataset.shape)
38 print(dataset)
```



### 악보 학습 - 기본 LSTM

- 4개의 속성 (1-4번째 음표)과 1개의 클래스 (5번째 음표)로 구성된 50개의 데이터 셋 생성

(50, 5)

```
[[11 9 2 10 8]
 [ 9 2 10 8 1]
 [ 2 10 8 1 7]
 [10 8 1 7 8]
 [ 8 1 7 8 9]
 [ 1 7 8 9 10]
 [ 7 8 9 10 11]
 [ 8 9 10 11 11]
 [ 9 10 11 11 4]
 [10 11 11 4 11]
 [11 11 4 11 9]
 [11 4 11 9 9]
 [ 4 11 9 9 9]
 [11 9 9 9 10]
 [ 9 9 9 10 8]
 [ 9 9 10 8 1]
 [ 9 10 8 1 7]
 [10 8 1 7 9]
```

```
[ 8 1 7 9 11]
[ 1 7 9 11 11]
[ 7 9 11 11 9]
[ 9 11 11 9 9]
[11 11 9 9 2]
[11 9 9 2 8]
[ 9 9 2 8 8]
[ 9 2 8 8 8]
[ 2 8 8 8 8]
[ 8 8 8 8 8]
[ 8 8 8 8 9]
[ 8 8 8 9 3]
[ 8 8 9 3 9]
[ 8 9 3 9 9]
[ 9 3 9 9 9]
[ 3 9 9 9 9]
[ 9 9 9 9 9]
[ 9 9 9 9 10]
[ 9 9 9 10 4]
```

```
[ 9 9 10 4 11]
[ 9 10 4 11 9]
[10 4 11 9 2]
[ 4 11 9 2 10]
[11 9 2 10 8]
[ 9 2 10 8 1]
[ 2 10 8 1 7]
[10 8 1 7 9]
[ 8 1 7 9 11]
[ 1 7 9 11 11]
[ 7 9 11 11 9]
[ 9 11 11 9 9]
[11 11 9 9 2]]
```



### 악보 학습 - 기본 LSTM

#### ● 원-핫 인코딩

```
40 # 입력(X)과 출력(Y) 변수로 분리하기
41 x_train = dataset[:, 0:4]
42 y_train = dataset[:, 4]
43
44 # 음표의 종류 수
45 max_idx_value = 13
46
47 # 입력값 정규화 시키기 (0-1 사이 실수로 변환)
48 x_train = x_train / float(max_idx_value)
49
50 # 샘플수(전체음표수), 타임스텝(1-4번째 음표), 특성수(5번째 음표) 차원으로 변환
51 x_train = np.reshape(x_train, (50, 4, 1))
52
53 # 라벨값에 대한 one-hot 인코딩 수행
54 y_train = np_utils.to_categorical(y_train)
55
56 # 원-핫 인코딩한 라벨값의 크기
57 one_hot_vec_size = y_train.shape[1]
58
59 print("one hot encoding vector size is ", one_hot_vec_size)
```



### 악보 학습 - 기본 LSTM

#### ● 기본 LSTM 모델

- 128개의 메모리 셀을 가진 LSTM 레이어와 Dense 레이어로 구성
- 입력은 샘플이 50개, 타임스텝이 4개, 속성이 1개로 구성
- 상태유지(stateful) 모드 비활성화

61	model = Sequential()
62	model.add(LSTM(128, input_shape = (4, 1)))
63	model.add(Dense(one_hot_vec_size, activation='softmax'))

#### ● 모델 학습 과정 설정

65	model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
----	---



### 악보 학습 - 기본 LSTM

#### ● 모델 학습

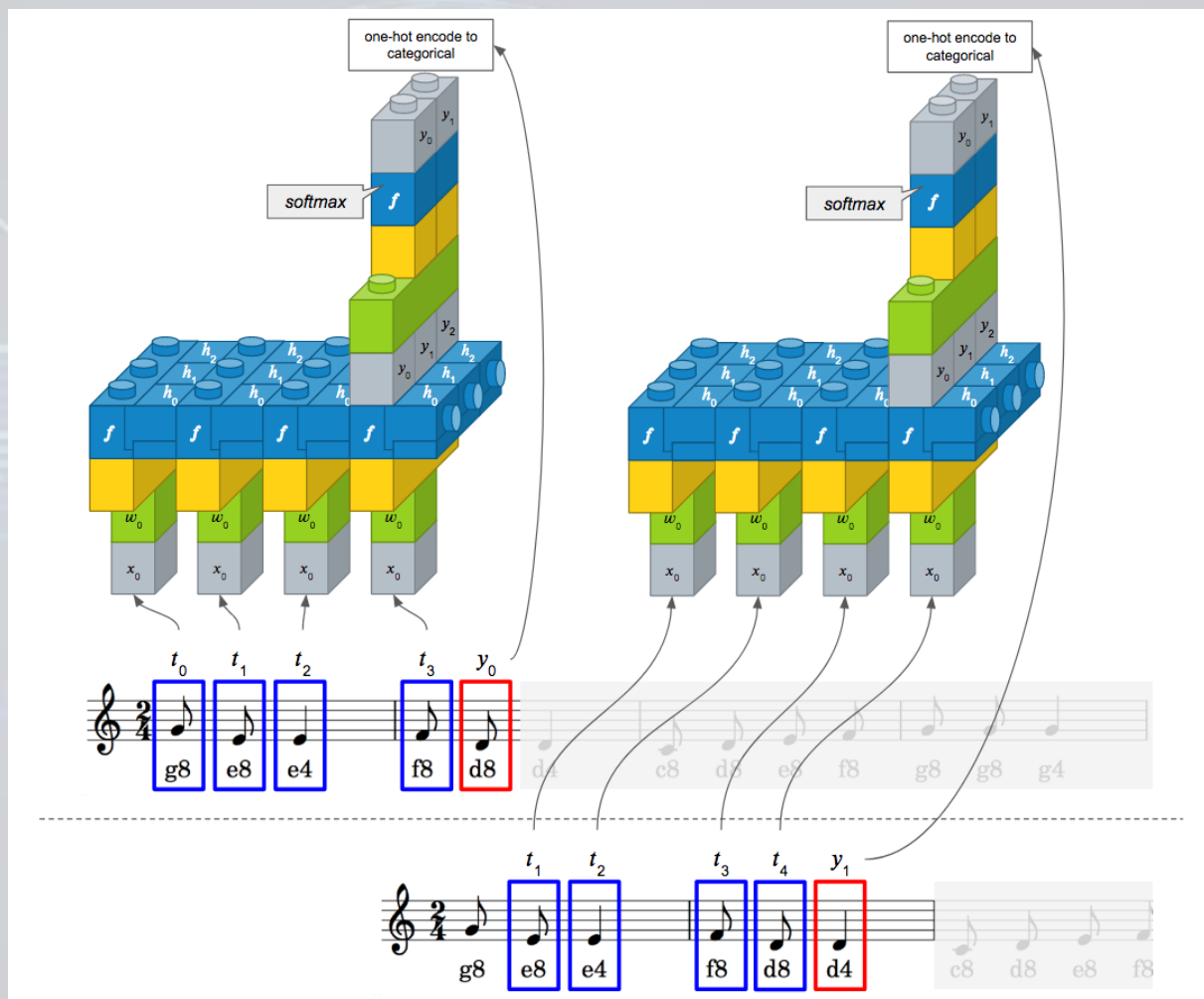
- 4개의 음표가 4개의 시퀀스 입력으로 들어가고, 그 다음 음표가 라벨값으로 지정
- 이 과정을 곡이 마칠 때까지 반복

```
67 history = LossHistory() # 손실 이력 객체 생성
68 history.init()
69
70 model.fit(x_train, y_train, epochs=2000, batch_size=14, verbose=2,
           callbacks=[history])
```



### 악보 학습 - 기본 LSTM

#### 기본 LSTM 모델





### 악보 학습 - 기본 LSTM

#### ● 학습 결과

....  
Epoch 2000/2000  
- 0s - loss: 0.2958 - **acc: 0.8800**

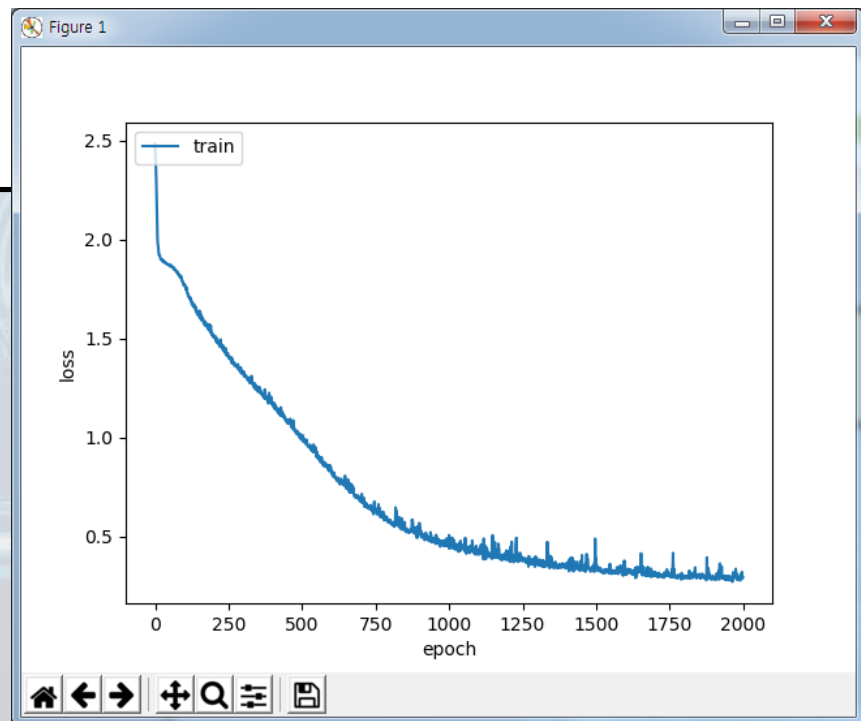




### 악보 학습 - 기본 LSTM

#### ● 학습 결과 시각화

```
72 import matplotlib.pyplot as plt
73
74 plt.plot(history.losses)
75 plt.ylabel('loss')
76 plt.xlabel('epoch')
77 plt.legend(['train'], loc='upper left')
78 plt.show()
```





### 악보 학습 - 기본 LSTM

#### • 모델 평가하기

```
80 scores = model.evaluate(x_train, y_train, batch_size=1)
81 print("%s: %.2f%%" % (model.metrics_names[1], scores[1] * 100))
```

acc: 90.00%



### 악보 학습 - 기본 LSTM

- 모델 사용하기 (한 스텝 예측)

```
83 pred_count = 50 # 최대 예측 개수 정의
84
85 # 한 스텝 예측
86 seq_out = ['g8', 'e8', 'e4', 'f8']
87 pred_out = model.predict(x_train)
88
89 for i in range(pred_count):
90     idx = np.argmax(pred_out[i]) # one-hot 인코딩을 인덱스 값으로 변환
91     # seq_out는 최종 악보이므로 인덱스 값을 코드로 변환하여 저장
92     seq_out.append(idx2code[idx])
93
94 print("one step prediction : ", seq_out)
```



## 악보 학습 – 기본 LSTM

### ● 모델 사용하기 (한 스텝 예측)

one step prediction : ['g8', 'e8', 'e4', 'f8', 'd8', 'd4', 'c8', 'e8', 'g8', 'f8', 'g8', 'g8', 'g4', 'g8', 'e8', 'e8', 'e8', 'f8', 'g4', 'd8', 'c8', 'e8', 'g8', 'g8', 'e8', 'e8', 'e4', 'd8', 'd8', 'd8', 'd8', 'd8', 'd8', 'f4', 'e8', 'e8', 'e8', 'e8', 'f8', 'f8', 'g4', 'g8', 'e8', 'e4', 'f8', 'd8', 'd4', 'c8', 'e8', 'g8', 'g8', 'e8', 'e8', 'e4']

실제 악보 seq : ['g8', 'e8', 'e4', 'f8', 'd8', 'd4', 'c8', 'd8', 'e8', 'f8', 'g8', 'g8', 'g4', 'g8', 'e8', 'e8', 'e8', 'f8', 'd8', 'd4', 'c8', 'e8', 'g8', 'g8', 'e8', 'e8', 'e4', 'd8', 'd8', 'd8', 'd8', 'd8', 'e8', 'f4', 'e8', 'e8', 'e8', 'e8', 'e8', 'f8', 'g4', 'g8', 'e8', 'e4', 'f8', 'd8', 'd4', 'c8', 'e8', 'g8', 'g8', 'e8', 'e8', 'e4']



### 악보 학습 - 기본 LSTM

#### 모델 사용하기 (곡 전체 예측)

```
96 seq_in = ['g8', 'e8', 'e4', 'f8']
97 seq_out = seq_in
98 # 코드를 인덱스값으로 변환
99 seq_in = [code2idx[it] / float(max_idx_value) for it in seq_in]
100
101 for i in range(pred_count):
102     sample_in = np.array(seq_in)
103     sample_in = np.reshape(sample_in, (1, 4, 1)) # 샘플 수, 타임스텝 수, 속성 수
104     pred_out = model.predict(sample_in)
105     idx = np.argmax(pred_out)
106     seq_out.append(idx2code[idx])
107     seq_in.append(idx / float(max_idx_value))
108     seq_in.pop(0)
109
110 print("full song prediction : ", seq_out)
```



## ● 모델 사용하기 (곡 전체 예측)

실제 악보 seq : [g8', 'e8', 'e4', 'f8', 'd8', 'd4', 'c8', 'd8', 'e8', 'f8', 'g8', 'g8', 'g4', 'g8', 'e8', 'e8', 'e8', 'f8', 'd8', 'd4', 'c8', 'e8', 'g8', 'g8', 'e8', 'e8', 'e4', 'd8', 'd8', 'd8', 'd8', 'd8', 'e8', 'f4', 'e8', 'e8', 'e8', 'e8', 'e8', 'f8', 'g4', 'g8', 'e8', 'e4', 'f8', 'd8', 'd4', 'c8', 'e8', 'g8', 'g8', 'e8', 'e8', 'e4']



### 악보 학습 - 상태 유지 LSTM 모델

- 상태유지 : 현재 학습된 상태가 다음 학습 시 초기 상태로 전달된다는 것
- 상태유지 LSTM 모델
  - batch\_input\_shape = (1, 4, 1) : 한개의 데이터가 각 배치 할 때 마다 1개의 데이터가 저장

```
61 model = Sequential()  
62 model.add(LSTM(128, batch_input_shape = (1, 4, 1), stateful=True))  
63 model.add(Dense(one_hot_vec_size, activation='softmax'))  
64  
65 model.compile(loss='categorical_crossentropy', optimizer='adam',  
metrics=['accuracy'])
```



### 악보 학습 - 상태 유지 LSTM 모델

#### 모델 학습 시키기

- 마지막 샘플 학습이 마치고, 새로운 에포크 수행 시에는 새로운 샘플 학습을 해야하므로 상태 초기화 필요
- 한 에포크 안에 여러 시퀀스 데이터 세트가 있을 경우, 새로운 시퀀스 데이터 세트를 학습 전에 상태 초기화 필요
- 현재 코드에서는 한 곡을 가지고 계속 학습을 시키고 있으므로 새로운 에포크 시작 시에만 상태 초기화를 수행

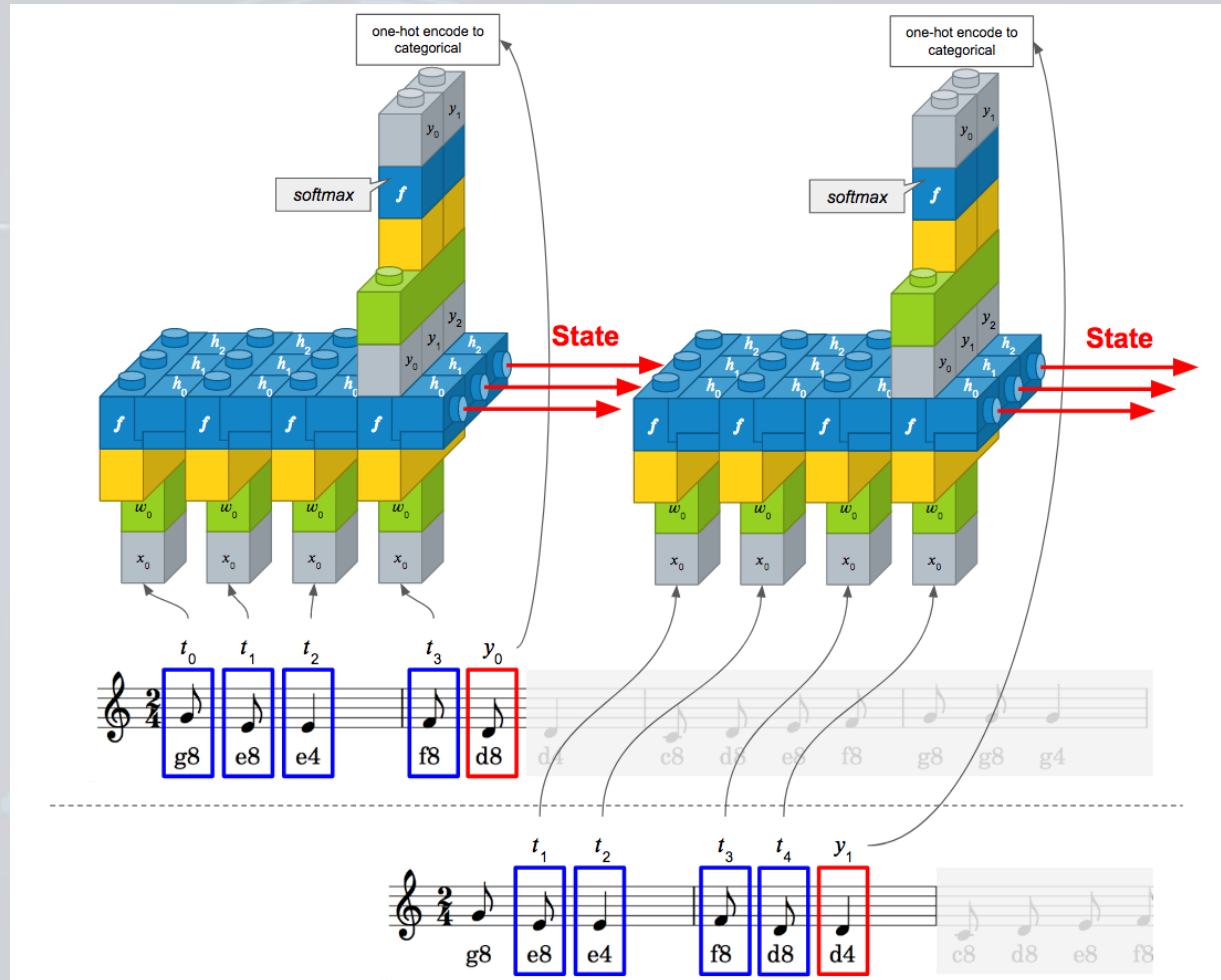
```
67 num_epochs = 2000
68
69 history = LossHistory() # 손실 이력 객체 생성
70 history.init()
71
72 for epoch_idx in range(num_epochs):
73     print ('epochs : ' + str(epoch_idx) )
74
75     model.fit(x_train, y_train, epochs=1, batch_size=1, verbose=2, shuffle=False)
76     model.reset_states()
```





### 악보 학습 - 상태 유지 LSTM 모델

#### 상태유지 LSTM 모델

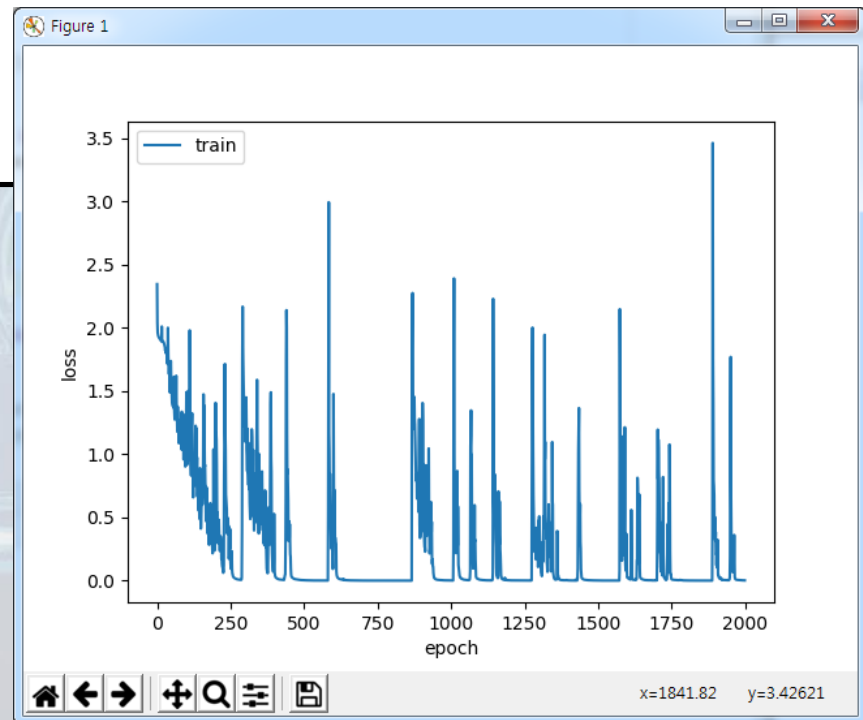




### 악보 학습 - 상태 유지 LSTM 모델

#### ● 학습 결과 시각화

```
78 import matplotlib.pyplot as plt
79
80 plt.plot(history.losses)
81 plt.ylabel('loss')
82 plt.xlabel('epoch')
83 plt.legend(['train'], loc='upper left')
84 plt.show()
```





### 악보 학습 - 상태 유지 LSTM 모델

#### 모델 평가하기

```
86 scores = model.evaluate(x_train, y_train, batch_size=1)
87 print("%s: %.2f%%" % (model.metrics_names[1], scores[1] * 100))
88 model.reset_states()
```

acc: 100.00%



### 악보 학습 - 상태 유지 LSTM 모델

#### 모델 사용하기 (한 스텝 예측)

```
90 pred_count = 50 # 최대 예측 개수 정의
91
92 # 한 스텝 예측
93 seq_out = ['g8', 'e8', 'e4', 'f8']
94 pred_out = model.predict(x_train, batch_size=1)
95
96 for i in range(pred_count):
97     idx = np.argmax(pred_out[i]) # one-hot 인코딩을 인덱스 값으로 변환
98     # seq_out는 최종 악보이므로 인덱스 값을 코드로 변환하여 저장
99     seq_out.append(idx2code[idx])
100
101 model.reset_states()
102
103 print("one step prediction : ", seq_out)
```



### 악보 학습 - 상태 유지 LSTM 모델

#### ● 모델 사용하기 (한 스텝 예측)

one step prediction : ['g8', 'e8', 'e4', 'f8', 'd8', 'd4', 'c8', 'd8', 'e8', 'f8', 'g8', 'g8', 'g4', 'g8', 'e8', 'e8', 'e8', 'f8', 'd8', 'd4', 'c8', 'e8', 'g8', 'g8', 'e8', 'e8', 'e4', 'd8', 'd8', 'd8', 'd8', 'd8', 'e8', 'f4', 'e8', 'e8', 'e8', 'e8', 'e8', 'f8', 'g4', 'g8', 'e8', 'e4', 'f8', 'd8', 'd4', 'c8', 'e8', 'g8', 'g8', 'e8', 'e8', 'e4']

실제 악보 seq : ['g8', 'e8', 'e4', 'f8', 'd8', 'd4', 'c8', 'd8', 'e8', 'f8', 'g8', 'g8', 'g4', 'g8', 'e8', 'e8', 'e8', 'f8', 'd8', 'd4', 'c8', 'e8', 'g8', 'g8', 'e8', 'e8', 'e4', 'd8', 'd8', 'd8', 'd8', 'd8', 'e8', 'f4', 'e8', 'e8', 'e8', 'e8', 'e8', 'f8', 'g4', 'g8', 'e8', 'e4', 'f8', 'd8', 'd4', 'c8', 'e8', 'g8', 'g8', 'e8', 'e8', 'e4']



### 악보 학습 - 상태 유지 LSTM 모델

#### • 모델 사용하기 (곡 전체 예측)

```
105 seq_in = ['g8', 'e8', 'e4', 'f8']
106 seq_out = seq_in
107 # 코드를 인덱스값으로 변환
108 seq_in = [code2idx[it] / float(max_idx_value) for it in seq_in]
109
110 for i in range(pred_count):
111     sample_in = np.array(seq_in)
112     sample_in = np.reshape(sample_in, (1, 4, 1)) # 샘플 수, 타임스텝 수, 속성 수
113     pred_out = model.predict(sample_in)
114     idx = np.argmax(pred_out)
115     seq_out.append(idx2code[idx])
116     seq_in.append(idx / float(max_idx_value))
117     seq_in.pop(0)
118
119 model.reset_states()
120
121 print("full song prediction : ", seq_out)
```



### 악보 학습 - 상태 유지 LSTM 모델

#### 모델 사용하기 (곡 전체 예측)

full song prediction : ['g8', 'e8', 'e4', 'f8', 'd8', 'd4', 'c8', 'd8', 'e8', 'f8', 'g8', 'g8', 'g4', 'g8', 'e8', 'e8', 'e8', 'f8', 'd8', 'd4', 'c8', 'e8', 'g8', 'g8', 'e8', 'e8', 'e4', 'd8', 'd8', 'd8', 'd8', 'd8', 'e8', 'f4', 'e8', 'e8', 'e8', 'e8', 'e8', 'f8', 'g4', 'g8', 'e8', 'e4', 'f8', 'd8', 'd4', 'c8', 'e8', 'g8', 'g8', 'e8', 'e8', 'e4']

실제 악보 seq : ['g8', 'e8', 'e4', 'f8', 'd8', 'd4', 'c8', 'd8', 'e8', 'f8', 'g8', 'g8', 'g4', 'g8', 'e8', 'e8', 'e8', 'f8', 'd8', 'd4', 'c8', 'e8', 'g8', 'g8', 'e8', 'e8', 'e4', 'd8', 'd8', 'd8', 'd8', 'd8', 'e8', 'f4', 'e8', 'e8', 'e8', 'e8', 'e8', 'f8', 'g4', 'g8', 'e8', 'e4', 'f8', 'd8', 'd4', 'c8', 'e8', 'g8', 'g8', 'e8', 'e8', 'e4']



### 악보 학습 – 입력 속성이 여러 개인 LSTM

#### • 라이브러리 로딩, 시드 고정

```
1 import keras
2 import numpy as np
3 from keras.models import Sequential
4 from keras.layers import Dense, LSTM
5 from keras.utils import np_utils
6
7 # 랜덤시드 고정시키기
8 np.random.seed(5)
9
10 # 음 종류 최대 값 (c, d, e, f, g, a, b)
11 max_scale_value = 6.0
```





### 악보 학습 – 입력 속성이 여러 개인 LSTM

- 손실 이력 클래스 정의 : epoch가 끝날 때마다 losses에 loss 값을 저장

```
12 class LossHistory(keras.callbacks.Callback):  
13     def init(self):  
14         self.losses = []  
15  
16     def on_epoch_end(self, batch, logs={}):  
17         self.losses.append(logs.get('loss'))
```



### 악보 학습 - 입력 속성이 여러 개인 LSTM

- 속성 변환 함수 : 현재 입력값이 'c4, e4, g8'등으로 되어 있는 데, 이를 음정과 음길이로 나누어서 2개의 속성으로 입력 → 'c4'는 '(c, 4)'로 나누어서 입력

```

19 def code2features(code):
20     features = []
21     features.append(code2scale[code[0]] / float(max_scale_value))
22     features.append(code2length[code[1]])
23     return features
    
```

```

[0.6666666666666666, 1] ← code가 g8인 경우 code[0]은 g, code[1]은 8
[0.3333333333333333, 1]   g에 해당하는 코드가 4이므로 4/6 = 0.666666..
[0.3333333333333333, 0]   8에 해당하는 코드는 1
[0.5, 1]
[0.3333333333333333, 1]
[0.3333333333333333, 0]
[0.5, 1]
[0.16666666666666666, 1]
[0.3333333333333333, 0]
[0.5, 1]
....
    
```



### 악보 학습 - 입력 속성이 여러 개인 LSTM

- 데이터 셋 생성 함수 : 입력 속성을 2개 (dataset\_X, dataset\_Y)로 분리
  - dataset\_X : feature값이 저장
  - dataset\_Y : class (5번째 음표의 인덱스)가 저장

```
25 def seq2dataset(seq, window_size):
26     dataset_X = []
27     dataset_Y = []
28
29     for i in range(len(seq) - window_size):
30         subset = seq[i:(i + window_size + 1)]
31         for si in range(len(subset) - 1):
32             features = code2features(subset[si])
33             dataset_X.append(features)
34             dataset_Y.append([code2idx(subset[window_size])])
35
36     return np.array(dataset_X), np.array(dataset_Y)
```

```
[[0.6666666666666666, 1], [0.3333333333333333, 1], [0.3333333333333333, 0], [0.5, 1], ...
[[8], [1], [7], [8], ...
```



### 악보 학습 - 입력 속성이 여러 개인 LSTM

- 코드 사전 정의 : 음표와 길이 데이터 정의 추가

```
38 # 음표와 길이 데이터 정의
39 code2scale = {'c': 0, 'd': 1, 'e': 2, 'f': 3, 'g': 4, 'a': 5, 'b': 6}
40 code2length = {'4': 0, '8': 1}
41
42 code2idx = {'c4': 0, 'd4': 1, 'e4': 2, 'f4': 3, 'g4': 4, 'a4': 5, 'b4': 6,
43            'c8': 7, 'd8': 8, 'e8': 9, 'f8': 10, 'g8': 11, 'a8': 12, 'b8': 13}
44
45 idx2code = {0: 'c4', 1: 'd4', 2: 'e4', 3: 'f4', 4: 'g4', 5: 'a4', 6: 'b4',
46            7: 'c8', 8: 'd8', 9: 'e8', 10: 'f8', 11: 'g8', 12: 'a8', 13: 'b8'}
47
48 # 시퀀스 데이터 정의
49 seq = ['g8', 'e8', 'e4', 'f8', 'd8', 'd4', 'c8', 'd8', 'e8', 'f8', 'g8', 'g8', 'g4',
50        'g8', 'e8', 'e8', 'e8', 'f8', 'd8', 'd4', 'c8', 'e8', 'g8', 'g8', 'e8', 'e8', 'e4',
51        'd8', 'd8', 'd8', 'd8', 'd8', 'e8', 'f4', 'e8', 'e8', 'e8', 'e8', 'e8', 'f8', 'g4',
52        'g8', 'e8', 'e4', 'f8', 'd8', 'd4', 'c8', 'e8', 'g8', 'g8', 'e8', 'e8', 'e4']
53
54 dataset = seq2dataset(seq, window_size = 4)
```



## 악보 학습 - 입력 속성이 여러 개인 LSTM

### 원-핫 인코딩

```
56 # 입력을 (샘플 수, 타임스텝, 특성 수)로 형태 변환
57 x_train = np.reshape(x_train, (50, 4, 2))
58
59 # 라벨값에 대한 one-hot 인코딩 수행
60 y_train = np_utils.to_categorical(y_train)
61
62 one_hot_vec_size = y_train.shape[1]
63
64 print("one hot encoding vector size is ", one_hot_vec_size)
```



### 악보 학습 - 입력 속성이 여러 개인 LSTM

- 모델 구성 : LSTM 모델 생성 시 batch\_input\_shape 인자의 마지막 값이 1에서 2로 수정

```
66 model = Sequential()  
67 model.add(LSTM(128, batch_input_shape = (1, 4, 2), stateful=True))  
68 model.add(Dense(one_hot_vec_size, activation='softmax'))  
69  
70 model.compile(loss='categorical_crossentropy', optimizer='adam',  
    metrics=['accuracy'])
```



## 악보 학습 - 입력 속성이 여러 개인 LSTM

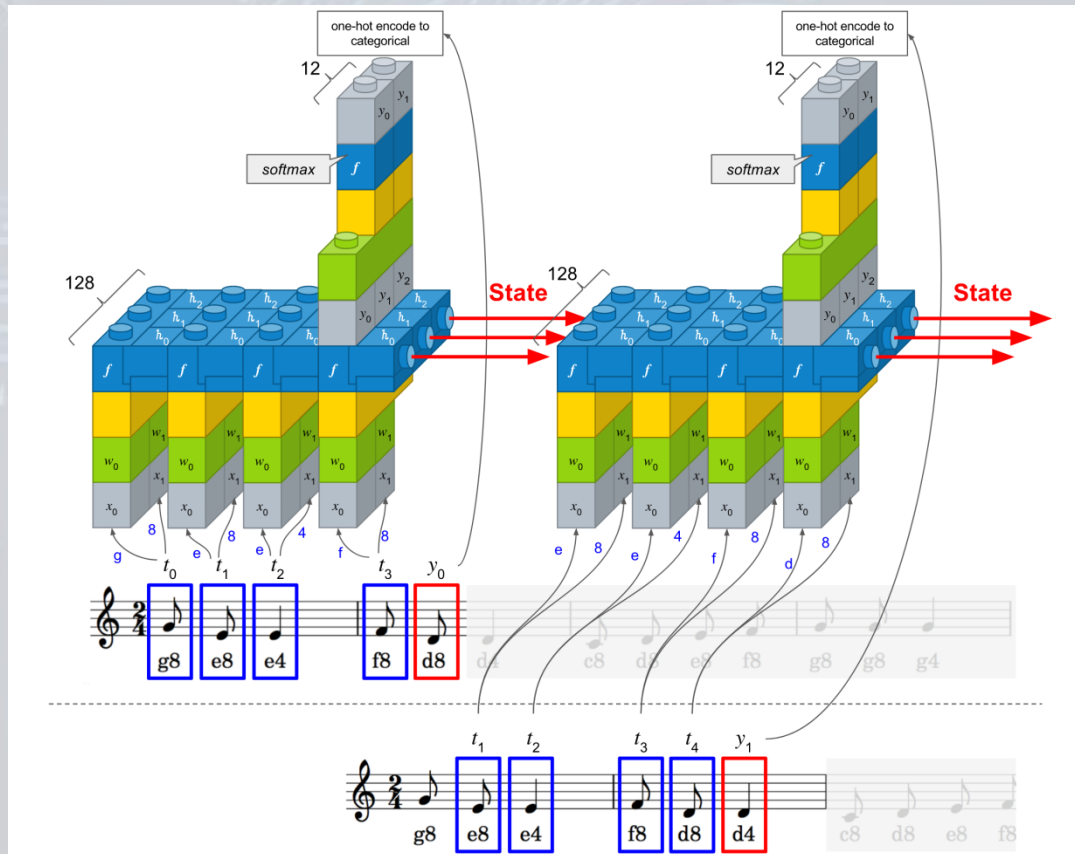
### • 모델 학습 시키기

```
72 num_epochs = 2000
73
74 history = LossHistory() # 손실 이력 객체 생성
75 history.init()
76
77 for epoch_idx in range(num_epochs):
78     print('epochs : ' + str(epoch_idx))
79     model.fit(x_train, y_train, epochs=1, batch_size=1, verbose=2, shuffle=False,
80             callbacks=[history])
81     model.reset_states()
```



### 악보 학습 - 입력 속성이 여러 개인 LSTM

- 입력 속성이 여러 개인 LSTM 모델 : 'c8'이니 'd4'처럼 코드 자체를 학습하는 것이 아니라 음정과 음길이를 나누어서 학습 → 좀 더 사람에 가까운 학습



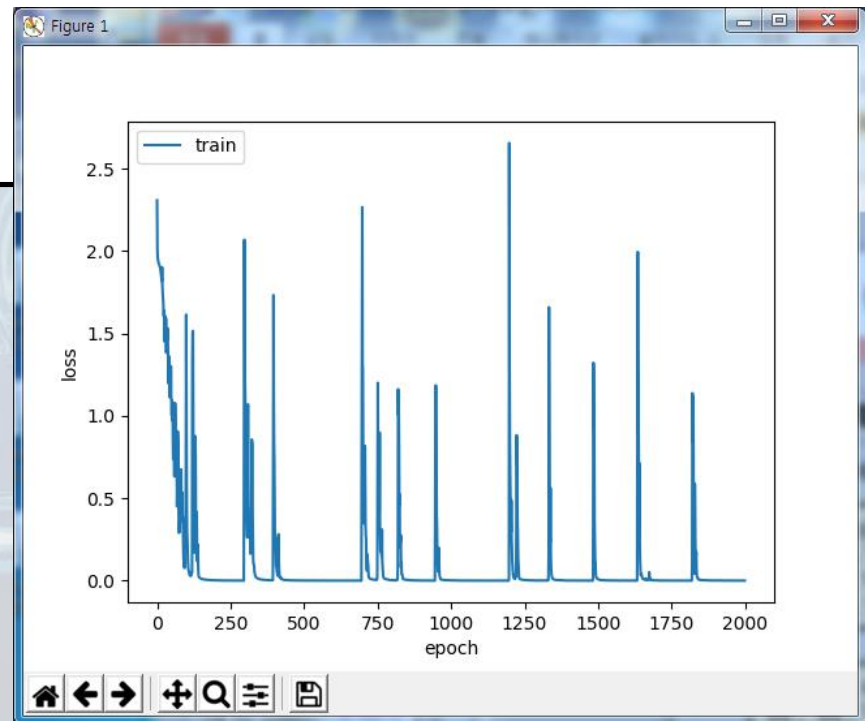




### 악보 학습 - 입력 속성이 여러 개인 LSTM

#### ● 학습 결과 시각화

```
83 import matplotlib.pyplot as plt
84
85 plt.plot(history.losses)
86 plt.ylabel('loss')
87 plt.xlabel('epoch')
88 plt.legend(['train'], loc='upper left')
89 plt.show()
```





### 악보 학습 - 입력 속성이 여러 개인 LSTM

#### 모델 평가하기

```
91 scores = model.evaluate(x_train, y_train, batch_size=1)
92 print("%s: %.2f%%" % (model.metrics_names[1], scores[1] * 100))
93 model.reset_states()
```

acc: 100.00%



### 악보 학습 - 입력 속성이 여러 개인 LSTM

#### ● 모델 사용하기 (한 스텝 예측)

```
95 pred_count = 50 # 최대 예측 개수 정의
96
97 # 한 스텝 예측
98 seq_out = ['g8', 'e8', 'e4', 'f8']
99 pred_out = model.predict(x_train, batch_size=1)
100
101 for i in range(pred_count):
102     idx = np.argmax(pred_out[i]) # one-hot 인코딩을 인덱스 값으로 변환
103     # seq_out는 최종 악보이므로 인덱스 값을 코드로 변환하여 저장
104     seq_out.append(idx2code[idx])
105
106 model.reset_states()
107
108 print("one step prediction : ", seq_out)
```



### 악보 학습 - 입력 속성이 여러 개인 LSTM

#### ● 모델 사용하기 (한 스텝 예측)

one step prediction : ['g8', 'e8', 'e4', 'f8', 'd8', 'd4', 'c8', 'd8', 'e8', 'f8', 'g8', 'g8', 'g4', 'g8', 'e8', 'e8', 'e8', 'f8', 'd8', 'd4', 'c8', 'e8', 'g8', 'g8', 'e8', 'e8', 'e4', 'd8', 'd8', 'd8', 'd8', 'd8', 'e8', 'f4', 'e8', 'e8', 'e8', 'e8', 'e8', 'f8', 'g4', 'g8', 'e8', 'e4', 'f8', 'd8', 'd4', 'c8', 'e8', 'g8', 'g8', 'e8', 'e8', 'e4']

실제 악보 seq : ['g8', 'e8', 'e4', 'f8', 'd8', 'd4', 'c8', 'd8', 'e8', 'f8', 'g8', 'g8', 'g4', 'g8', 'e8', 'e8', 'e8', 'f8', 'd8', 'd4', 'c8', 'e8', 'g8', 'g8', 'e8', 'e8', 'e4', 'd8', 'd8', 'd8', 'd8', 'd8', 'e8', 'f4', 'e8', 'e8', 'e8', 'e8', 'e8', 'f8', 'g4', 'g8', 'e8', 'e4', 'f8', 'd8', 'd4', 'c8', 'e8', 'g8', 'g8', 'e8', 'e8', 'e4']



### 악보 학습 - 입력 속성이 여러 개인 LSTM

- 모델 사용하기 (곡 전체 예측)

```
110 seq_in = ['g8', 'e8', 'e4', 'f8']
111 seq_out = seq_in
112
113 seq_in_feats = []
114
115 for si in seq_in:
116     features = code2features(si)
117     seq_in_feats.append(features)
118
119 for i in range(pred_count):
120     sample_in = np.array(seq_in_feats)
121     sample_in = np.reshape(sample_in, (1, 4, 2)) # 샘플 수, 타임스텝 수, 속성 수
122     pred_out = model.predict(sample_in)
123     idx = np.argmax(pred_out)
124     seq_out.append(idx2code[idx])
125     seq_in.append(idx / float(max_idx_value))
126     seq_in.pop(0)
127
128 model.reset_states()
129 print("full song prediction : ", seq_out)
```



### 악보 학습 – 입력 속성이 여러 개인 LSTM

#### 모델 사용하기 (곡 전체 예측)

full song prediction : ['g8', 'e8', 'e4', 'f8', 'd8', 'd4', 'c8', 'd8', 'e8', 'f8', 'g8', 'g8', 'g4', 'g8', 'e8', 'e8', 'e8', 'f8', 'd8', 'd4', 'c8', 'e8', 'g8', 'g8', 'e8', 'e8', 'e4', 'd8', 'd8', 'd8', 'd8', 'd8', 'e8', 'f4', 'e8', 'e8', 'e8', 'e8', 'e8', 'f8', 'g4', 'g8', 'e8', 'e4', 'f8', 'd8', 'd4', 'c8', 'e8', 'g8', 'g8', 'e8', 'e8', 'e4']

실제 악보 seq : ['g8', 'e8', 'e4', 'f8', 'd8', 'd4', 'c8', 'd8', 'e8', 'f8', 'g8', 'g8', 'g4', 'g8', 'e8', 'e8', 'e8', 'f8', 'd8', 'd4', 'c8', 'e8', 'g8', 'g8', 'e8', 'e8', 'e4', 'd8', 'd8', 'd8', 'd8', 'd8', 'e8', 'f4', 'e8', 'e8', 'e8', 'e8', 'e8', 'f8', 'g4', 'g8', 'e8', 'e4', 'f8', 'd8', 'd4', 'c8', 'e8', 'g8', 'g8', 'e8', 'e8', 'e4']

# GRU



### GRUs (Gated Recurrent Units)

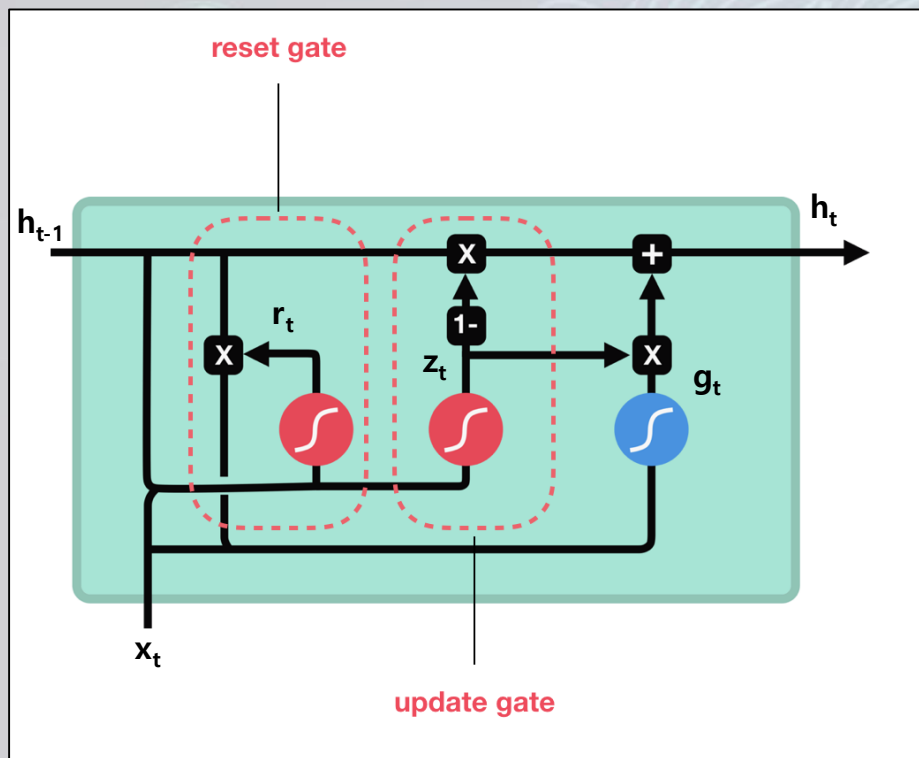
- 2014년에 K. Cho(조경현) 등이 LSTM을 간소화 한 버전으로, **Gradient Vanishing**의 문제를 해결
- LSTM은 초기의 weight가 계속 지속적으로 업데이트되었지만, GRUs는 Cell State를 없애고 **Update Gate와 Reset Gate를 추가**하여, 과거의 정보를 어떻게 반영할 것인지 결정
- **Update Gate**는 과거의 상태를 반영하는 **Gate** (Input gate와 유사한 동작)이며, **Reset Gate**는 현재 점 정보와 과거시점 정보의 반영 비율을 결정
- 장점 : 연산속도가 빠르며, 메모리가 LSTM처럼 덮여 쓰여질 가능성이 없음
- 단점 : 메모리와 결과값의 컨트롤이 불가능





### GRU (Gated Recurrent Units)

- LSTM의  $C_t$ 와  $h_t$ 가  $h_t$ 로 통합
- 하나의 게이트 제어기인  $z_t$ 가 0이면 이전( $t-1$ ) 정보가 전달되고 1이면 현재 ( $t$ ) 정보가 전달
- 전체 상태 벡터  $h_t$ 가 타임 스텝마다 출력되며  $r_t$ 는 이전 상태  $h_{t-1}$ 의 어느 부분이 출력될 지 제어



$$z_t = \sigma(X_t W_{xh\_z} + h_{t-1} W_{hh\_z} + b_z)$$

$$r_t = \sigma(X_t W_{xh\_r} + h_{t-1} W_{hh\_r} + b_r)$$

$$g_t = \tanh(X_t W_{xh\_h} + r_t h_{t-1} W_{hh\_h} + b_h)$$

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



### RNN의 고급 사용법

#### ● RNN의 성능과 일반화 능력 향상 기술

- (1) **순환 드롭아웃 (Recurrent dropout)** : RNN 층에서 과대적합을 방지하기 위해 케라스에 내장되어 있는 Dropout을 사용
- (2) **스태킹 순환 층 (Stacking recurrent layer)** : 네트워크의 표현 능력 향상 (계산 비용 상승)
- (3) **양방향 순환 층 (Bidirectional recurrent layer)** : RNN에 같은 정보를 다른 방향으로 주입하여 정확도를 높이고 기억을 좀 더 오래 유지시킴



### 기온 예측 문제 (24시간 후)

- 데이터 파싱하기 – 폴더로 부터 데이터 읽기

```
1 import os
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5 data_dir = 'D:/datasets/jena_climate'
6 fname = os.path.join(data_dir, 'jena_climate_2009_2016.csv')
7 f = open(fname)
8 data = f.read()
9 f.close()
```



### 기온 예측 문제 (24시간 후)

- 한 줄씩 읽어서 줄넘김과 콤마(,)로 분리하기

```
11 lines = data.split('\n')
12 header = lines[0].split(',')
13 lines = lines[1:]
14
15 print(header)
16 print(lines[0])
17 print(len(lines))
```

```
["Date Time", "p (mbar)", "T (degC)", "Tpot (K)", "Tdew (degC)", "rh (%)", "VPmax  
(mbar)", "VPact (mbar)", "VPdef (mbar)", "sh (g/kg)", "H2OC (mmol/mol)", "rho  
(g/m**3)", "wv (m/s)", "max. wv (m/s)", "wd (deg)"]
```

```
01.01.2009 00:10:00,996.52,-8.02,265.40,-  
8.90,93.30,3.33,3.11,0.22,1.94,3.12,1307.75,1.03,1.75,152.30
```

```
420551
```



### 기온 예측 문제 (24시간 후)

- 데이터 파싱하기 – numpy 배열로 저장

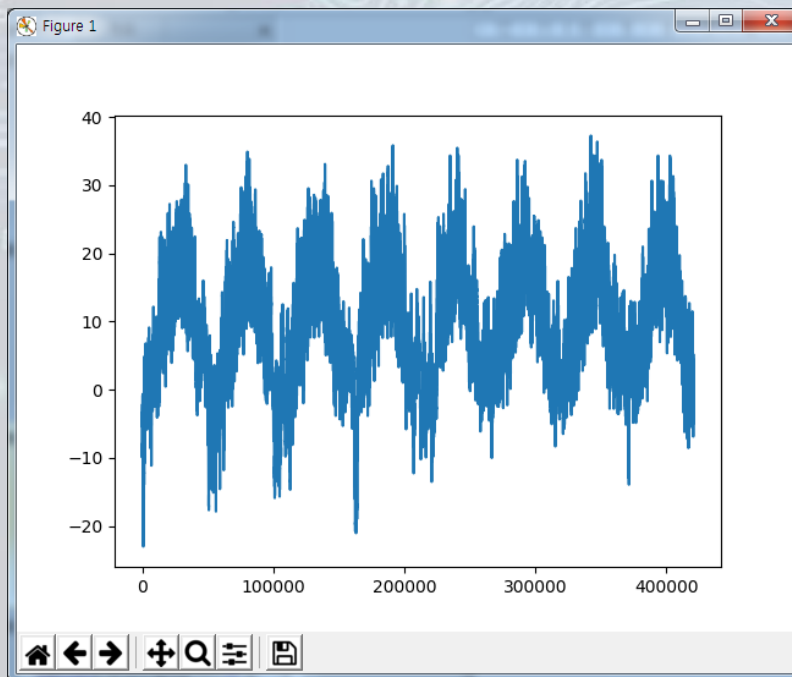
```
19 float_data = np.zeros((len(lines), len(header)-1))
20
21 for i, line in enumerate(lines):
22     values = [float(x) for x in line.split(',')[1:]]
23     float_data[i, :] = values
```



### 기온 예측 문제 (24시간 후)

#### 데이터 파싱하기

```
25 temp = float_data[:, 1]
26 plt.plot(range(len(temp)), temp)
27 plt.show()
```



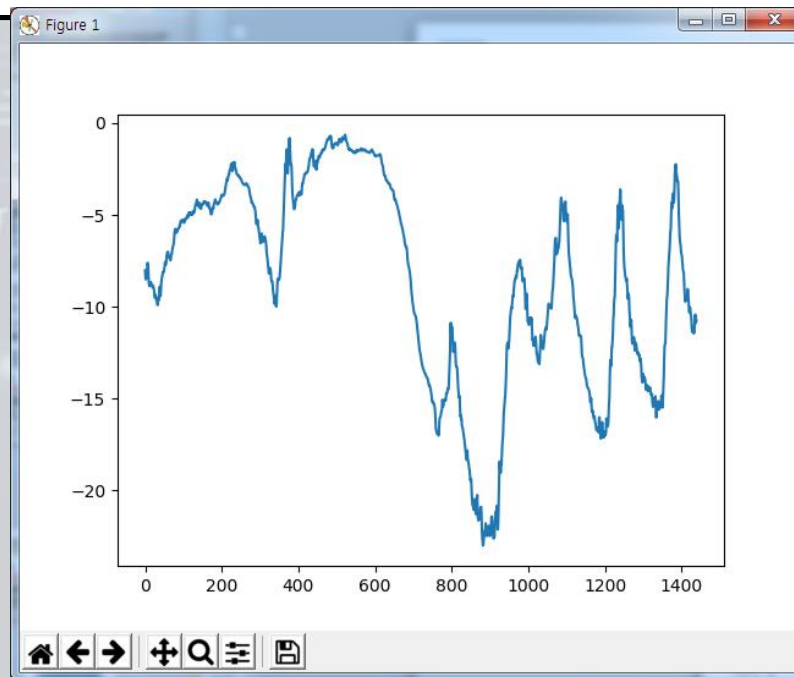
장기간을 보면 온도변화의  
규칙성이 존재  
→ 온도 예측 쉬움



### 기온 예측 문제 (24시간 후)

- 10일간의 온도 데이터 보기 : 10분 단위로 데이터가 기록되어 있으므로 하루에 총 144개의 데이터가 존재

```
25 temp = float_data[:, 1]
26 plt.plot(range(1440), temp[:1440])
27 plt.show()
```



단기간만 보면 온도변화의  
규칙성을 알 수 없음  
→ 온도 예측 가능 ?



### 기온 예측 문제 (24시간 후)

- 데이터 정규화 – 값 (온도, 기압 등)의 범위가 다르므로 비슷한 범위를 갖도록 작은 값으로 변경
  - 20만개의 데이터만 사용
  - 각 시계열 데이터에 대해 평균을 빼고 표준편차로 나눔

```
25 mean = float_data[:200000].mean(axis=0)
26 float_data -= mean
27 std = float_data[:200000].std(axis=0)
28 float_data /= std
```





### 기온 예측 문제 (24시간 후)

#### ● 시계열 데이터와 타겟을 반환하는 제네레이터 함수

- sample : 입력 데이터로 사용할 배치
- targets : 이에 대응하는 타겟 온도 배열
- 매개변수
  - data : 원본 배열
  - lookback : 참조할 이전 데이터의 타임스텝
  - delay : 타겟으로 사용할 미래의 타임스텝
  - min\_index / max\_index : 추출할 타임 스텝 범위 지정하기 위한 인덱스 배열 → 훈련과 테스트 데이터 분리
  - shurffle : 샘플 셔플 여부
  - batch\_size : 배치 샘플 수
  - step : 데이터를 샘플링할 타임스텝 간격 (1시간에 1개의 데이터 포인트 추출 → 6)



### 기온 예측 문제 (24시간 후)

- 시계열 데이터와 타겟을 반환하는 제네레이터 함수

```
30 def generator(data, lookback, delay, min_index, max_index,
31               shuffle=False, batch_size=128, step=6):
32     if max_index is None:
33         max_index = len(data) - delay - 1
34     i = min_index + lookback
35     while 1:
36         if shuffle:
37             rows = np.random.randint(
38                 min_index + lookback, max_index, size=batch_size)
39         else:
40             if i + batch_size >= max_index:
41                 i = min_index + lookback
42             rows = np.arange(i, min(i + batch_size, max_index))
43             i += len(rows)
44
45     samples = np.zeros((len(rows),
46                        lookback // step,
47                        data.shape[-1]))
48     targets = np.zeros((len(rows),))
```



### 기온 예측 문제 (24시간 후)

- 시계열 데이터와 타겟을 반환하는 제네레이터 함수

```
49
50     for j, row in enumerate(rows):
51         indices = range(rows[j] - lookback, rows[j], step)
52         samples[j] = data[indices]
53         targets[j] = data[rows[j] + delay][1]
54     yield samples, targets
```



### 기온 예측 문제 (24시간 후)

- 제네레이터를 이용하여 훈련, 검증, 테스트 데이터 준비

```
56  lookback = 1440    #10일전 데이터
57  step = 6           # 1시간마다 샘플링
58  delay = 144        # 24시간이 지난 데이터가 타겟
59  batch_size = 128
60
61  train_gen = generator(float_data,
62                        lookback=lookback,
63                        delay=delay,
64                        min_index=0,
65                        max_index=200000,    # 처음부터 20만개 데이터 사용
66                        shuffle=True,
67                        step=step,
68                        batch_size=batch_size)
69  val_gen = generator(float_data,
70                     lookback=lookback,
71                     delay=delay,
72                     min_index=200001,
73                     max_index=300000,    # 다음 10만개 데이터 사용
74                     step=step,
75                     batch_size=batch_size)
```



### 기온 예측 문제 (24시간 후)

- 제네레이터를 이용하여 훈련, 검증, 테스트 데이터 준비

```
76 test_gen = generator(float_data,  
77                     lookback=lookback,  
78                     delay=delay,  
79                     min_index=300001,      # 30만개를 제외한 나머지 데이터 사용  
80                     max_index=None,  
81                     step=step,  
82                     batch_size=batch_size)
```

- 제네레이터 회수 계산

```
84 # 전체 검증 세트를 순회하기 위해 val_gen에서 추출할 횟수  
85 val_steps = (300000 - 200001 - lookback) // batch_size  
86  
87 # 전체 테스트 세트를 순회하기 위해 test_gen에서 추출할 횟수  
88 test_steps = (len(float_data) - 300001 - lookback) // batch_size
```



### 기온 예측 문제 (24시간 후)

#### • 상식적인 기준 모델의 MAE 계산

- 온도는 연속성이 있고 일자별로 주기성이 있다고 가정 (오늘-내일의 온도는 비슷할 가능성이 높음)
- 지금부터 24시간 이후의 온도는 지금과 동일하다고 예측하는 것

```
90 def evaluate_naive_method():
91     batch_maes = []
92     for step in range(val_steps):
93         samples, targets = next(val_gen)
94         preds = samples[:, -1, 1]
95         mae = np.mean(np.abs(preds - targets))
96         batch_maes.append(mae)
97     print(np.mean(batch_maes))
98
99 evaluate_naive_method()
```

0.2897359729905486



## 기온 예측 문제 (24시간 후)

### • GRU를 사용한 모델

```
101 from keras.models import Sequential
102 from keras import layers
103 from keras.optimizers import RMSprop
104
105 model = Sequential()
106 model.add(layers.GRU(32, input_shape=(None, float_data.shape[-1])))
107 model.add(layers.Dense(1))
108
109 model.compile(optimizer=RMSprop(), loss='mae')
110 history = model.fit_generator(train_gen,
111                             steps_per_epoch=500,
112                             epochs=20,
113                             validation_data=val_gen,
114                             validation_steps=val_steps)
115
116 model.save('GRU_model.h5')
```



### 기온 예측 문제 (24시간 후)

#### • 결과 시각화

```
118 loss = history.history['loss']
119 val_loss = history.history['val_loss']
120
121 epochs = range(1, len(loss) + 1)
122
123 plt.figure()
124
125 plt.plot(epochs, loss, 'bo', label='Training loss')
126 plt.plot(epochs, val_loss, 'b', label='Validation loss')
127 plt.title('Training and validation loss')
128 plt.legend()
129
130 plt.show()
```





### 기온 예측 문제 (24시간 후)

#### 결과 시각화

