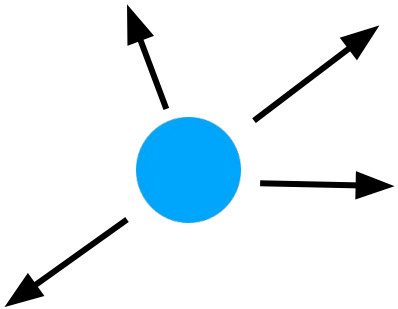
Recurrent Neural Network (RNN)

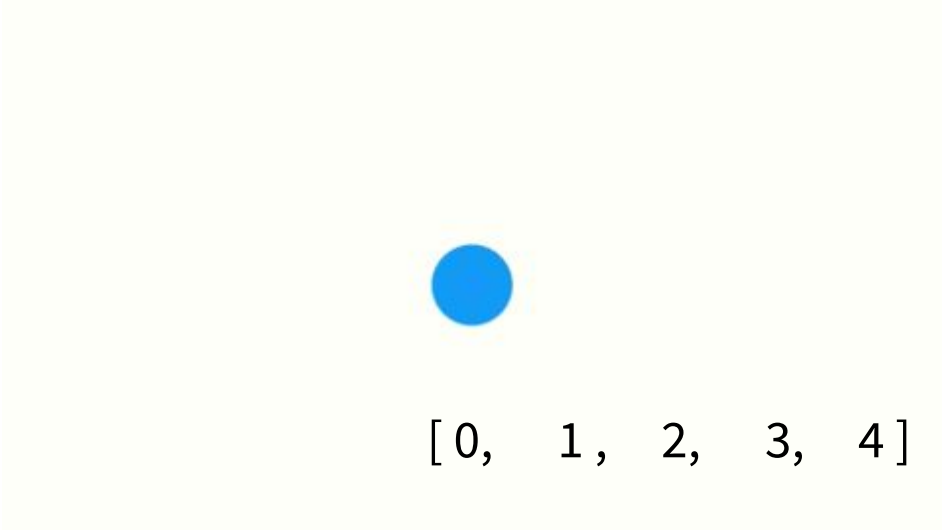
순환 신경망

공의 이미지가 주어지면,
다음에 어디로 갈지 예측할 수 있습니까?



[0] [?]

공의 이미지가 주어지면,
다음에 어디로 갈지 예측할 수 있습니까?



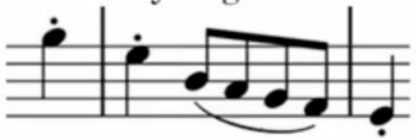
이전 위치 정보가 있으면 가능 ⇒ **순차적인(sequential) 데이타**를 처리할 필요 있음

예시

Examples of sequence data

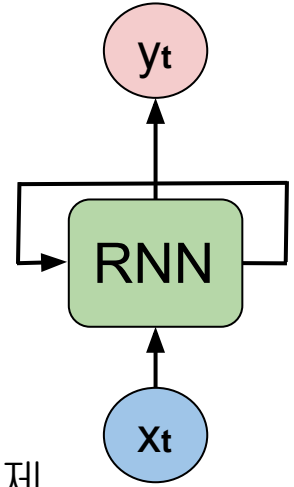
x

y

Speech recognition		"The quick brown fox jumped over the lazy dog."
Music generation	∅	
Sentiment classification	"There is nothing to like in this movie."	★☆☆☆☆
DNA sequence analysis	AGCCCCTGTGAGGAACTAG	AGCCCCTGTGAGGAACTAG
Machine translation	Voulez-vous chanter avec moi?	Do you want to sing with me?
Video activity recognition		Running
Name entity recognition	Yesterday, Harry Potter met Hermione Granger.	Yesterday, Harry Potter met Hermione Granger.

Andrew Ng

Sequence Modeling : 설계 기준



t=1, 2, 3, ...
시간대 별로
입출력 있음

시퀀스를 모델링하려면 다음을 수행해야 합니다.

1. **가변 길이(variable-length)** 시퀀스 처리
   (ex) 문장의 길이가 각각 다름
2. **장기적인 의존성(long-term dependency)** 추적
   (ex) “I grew up in **France** … I speak fluent ***French***.” 에서 ___ 예측 문제
3. **순서(order) 정보** 유지
   (ex) I google at work. / I work at google.
4. 시퀀스에서 **매개 변수(가중치) 공유** ⇐ 현재와 과거의 정보를 모두 담기위해
   (ex) "On Monday it was snowing" / "It was snowing on Monday"

순차적 데이터 처리를 위한 신경망이 필요 ⇒ **RNN (Recurrent Neural Networks)**

RNN (Recurrent Neural Network)

I love recurrent neural _____ .

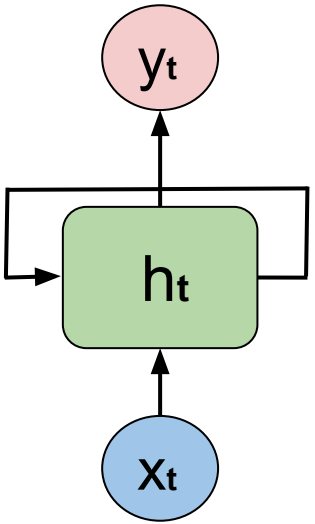
윗 문장에서 빈칸에 들어갈 단어 예측을 위한 순차적 자료 특성이 반영된 Pseudo-code

my_rnn = RNN()
hidden_state = [0, 0, 0, 0]

h: 히든 레이어의 노드

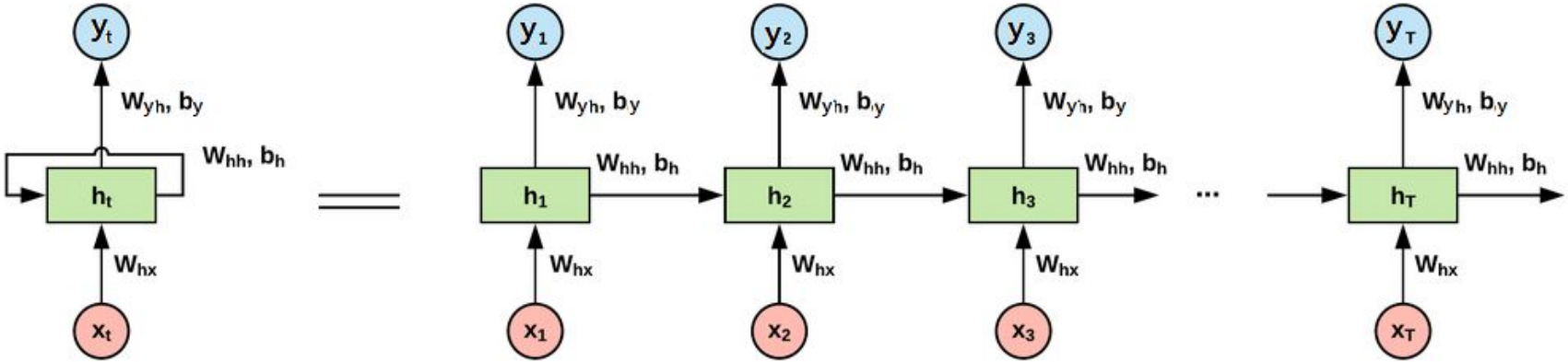
for word in [“I”, “love”, “recurrent”, “neural”]:
    prediction, hidden_state = **my_rnn**(word, hidden_state)

next_word_prediction = prediction



RNN (unrolled)

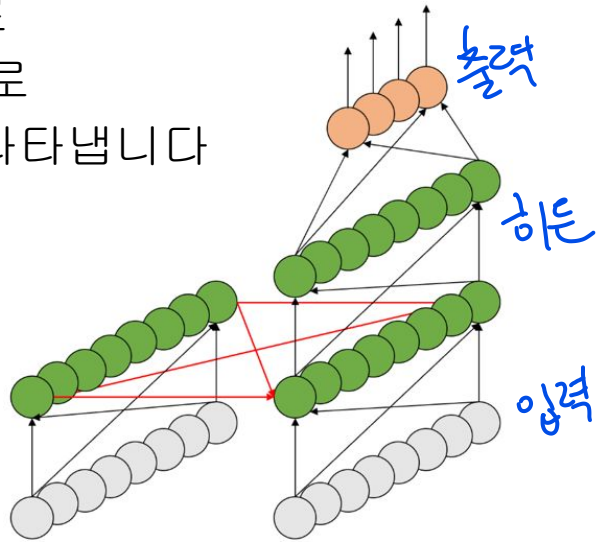
다음의 다이어그램에서 신경망 W_{hx} 는 일부 입력 x_t 를 보고 값 y_t 를 출력합니다.
루프는 정보를 네트워크의 한 단계에서 다음 단계로 전달할 수 있게 합니다.



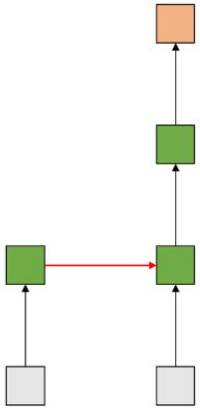
먼저 순차적 입력에서 x_1 을 가져와 이전 단계에서 입력될 h_0 과 함께 h_1 을 출력합니다.
따라서 h_1 와 x_2 는 다음 단계의 입력입니다. 마찬가지로 다음 단계의 h_2 는 그 다음 단계에 x_3 과 함께 입력이 된다. 이런 식으로 훈련하는 동안 문맥을 계속 기억합니다.

RNN - network structure

- 매번 입력은 벡터
- 각 층에는 많은 뉴런이 있음
- 출력 레이어에도 많은 뉴런이 있을 수 있음.
- 그러나 모두를 간단한 상자로
  나타냅니다. 각 상자는 실제로
  많은 요소로 전체 레이어를 나타냅니다



Simplified

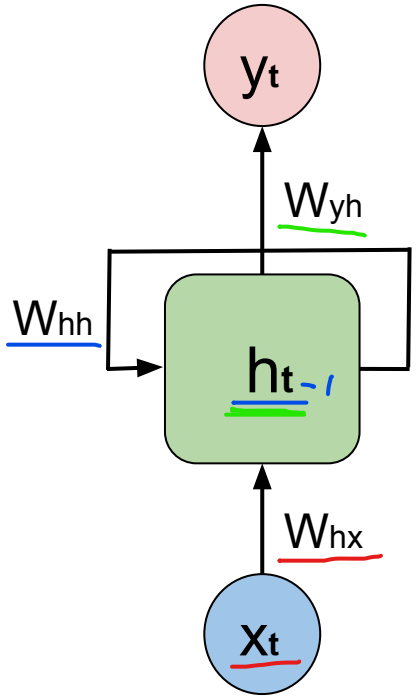


RNN - Feedforward Propagation

- 현재 상태 (**ht**) 계산: **ht = fw(ht-1, xt)**
  h 는 히든 벡터, **x** 는 입력

- 활성화 및 가중치와 함께: **ht = tanh(Whh ht-1 + Whx xt)**
  Whh 는 이전 히든 상태의 가중치,
  Whx 는 현재 입력 상태의 가중치,
  tanh 는 활성화 기능을 구현하는 비선형성을 구현합니다.
  (-1. ~ 1.)

- 출력 계산: **yt = softmax(Wyh ht)**
  여기에서 **Wyh** 출력 상태에서의 가중치이다.



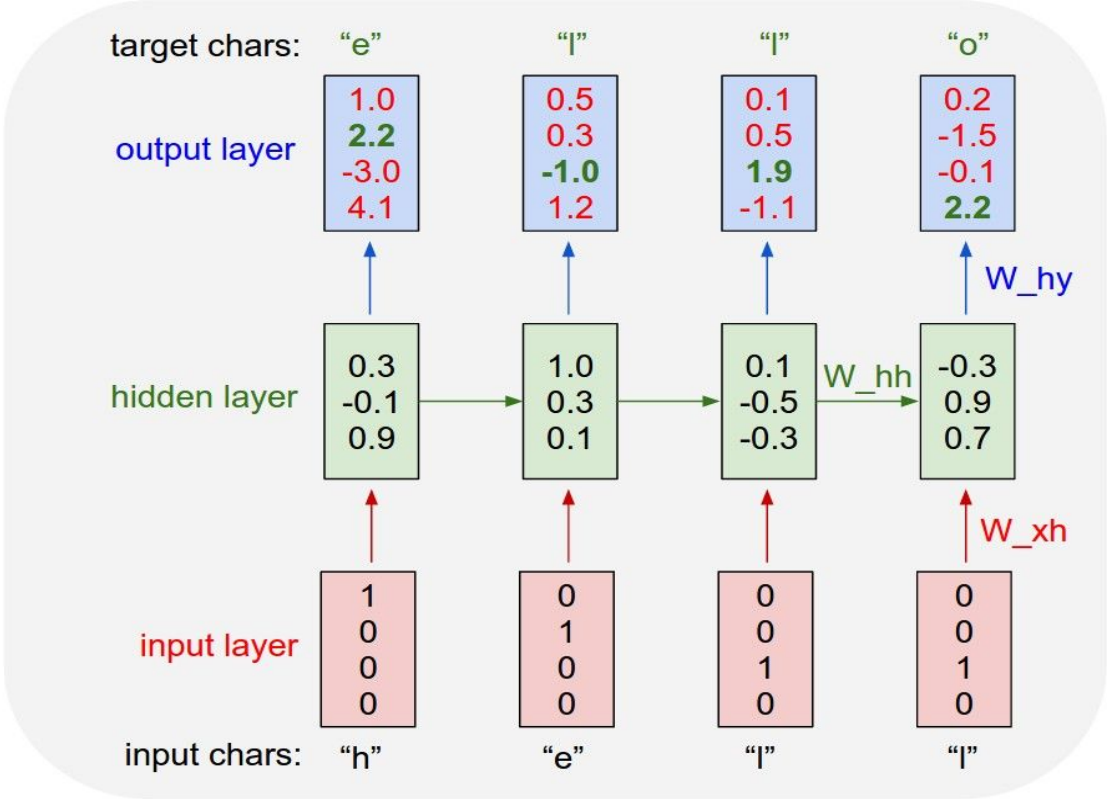
Character-Level Language Model

h e l l _ 에서 **_** 에 들어갈 문자 예측하기 ⇒ 이 때 각 문자를 표시하는데 (사용 가능한 문자가 총 4개 뿐이라고 가정하고)

one-hot encoding 을 사용한다면 ...

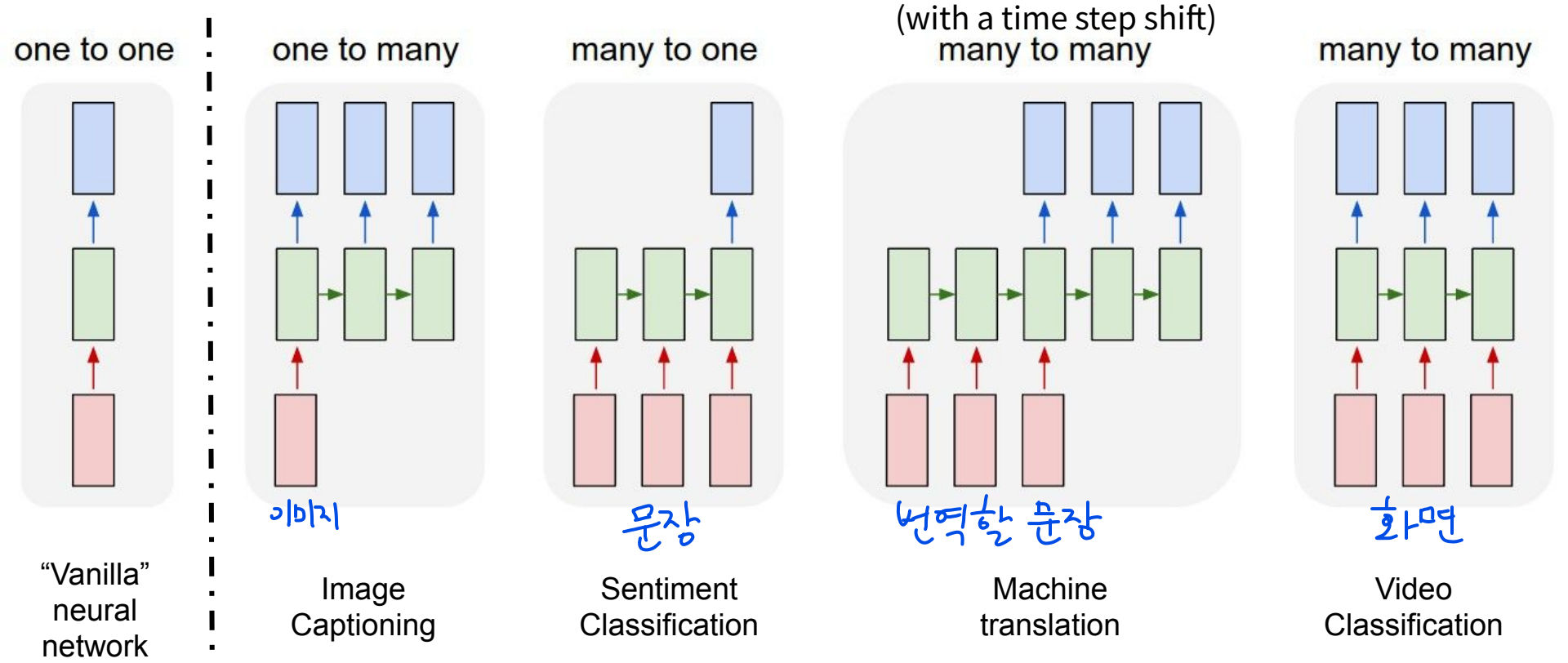
h = [1,0,0,0]
e = [0,1,0,0]
l = [0,0,1,0]
o = [0,0,0,1]

벡터

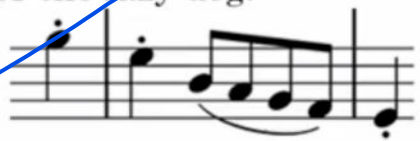


다양한 RNN 구조의 다이어그램 ☆

Various RNN architectures



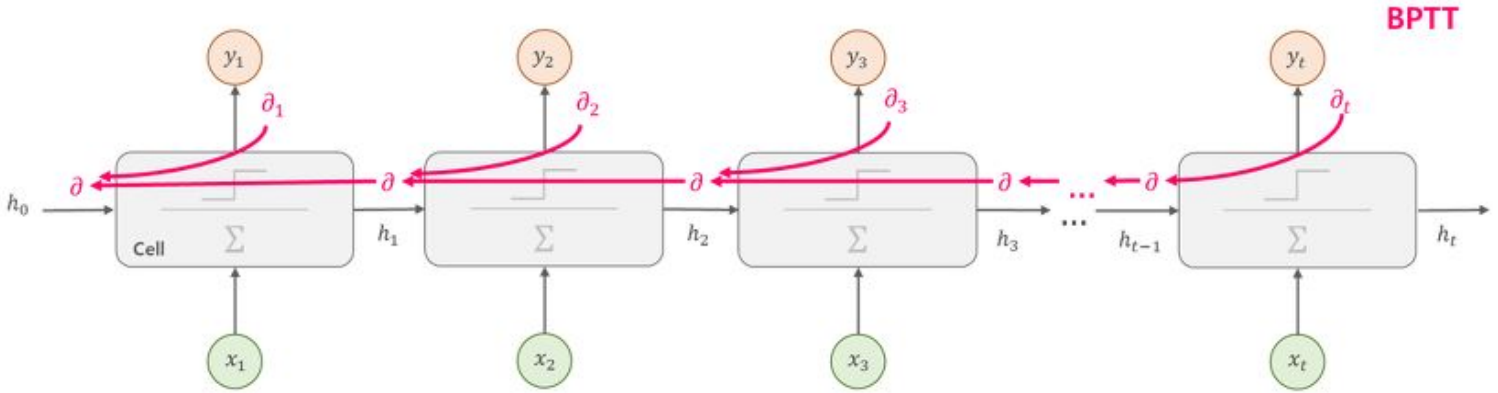
Examples of sequence data

Speech recognition		"The quick brown fox jumped over the lazy dog."
Music generation	∅	
Sentiment classification	"There is nothing to like in this movie."	★☆☆☆☆
DNA sequence analysis	AGCCCCTGTGAGGAACTAG	AGCCCCTGTGAGGAACTAG
Machine translation	Voulez-vous chanter avec moi?	Do you want to sing with me?
Video activity recognition		Running
Name entity recognition	Yesterday, Harry Potter met Hermione Granger.	Yesterday, Harry Potter met Hermione Granger.

Andrew Ng

RNN - BackPropagation Through Time

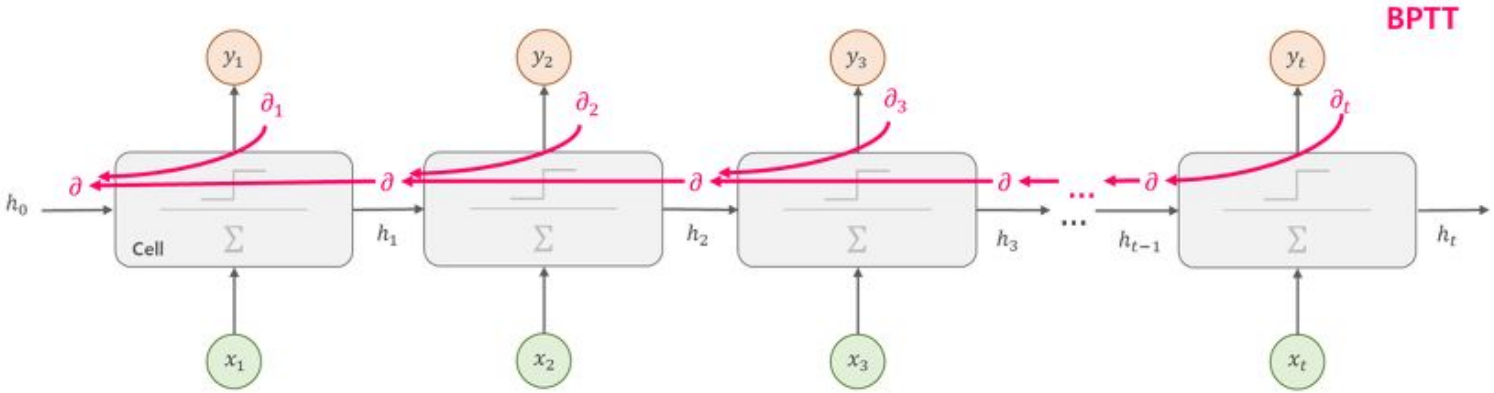
RNN은 타임 스텝별로 네트워크를 펼친 다음, **역전파 알고리즘**을 사용하는데 이를 **BPTT (BackPropagation Through Time)**라고 한다.



BPTT 또한 일반적인 역전파와 같이 먼저 **순전파(forward prop)**로 **각 타임 스텝별 시퀀스를 출력**한다. 그런 다음 이 출력 시퀀스와 손실(비용)함수를 사용하여 **각 타임 스텝별 Loss**를 구하고, **손실 함수의 그래디언트**는 위의 그림과 같이 펼쳐진 네트워크를 따라 **역방향으로 전파**된다. BPTT는 그래디언트가 마지막 타임 스텝인 출력뿐만 아니라 손실함수를 사용한 **모든 출력에서 역방향으로 전파**된다.

RNN - BackPropagation Through Time

RNN은 **각 타임 스텝마다 같은 매개변수가 사용**되기 때문에 역전파가 진행되면서 **모든 타임 스텝에 걸쳐 매개변수 값이 합산**된다. 이렇게 업데이트된 가중치는 순전파 동안에는 모든 타임 스텝에 동일한 가중치가 적용된다.



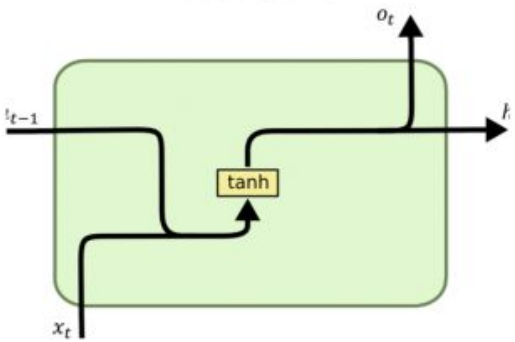
RNN / LSTM / GRU

요약 : RNN은 학습할때 Back Propagation을 쓰는데, 시간이 아주 오래 걸려서 심층 신경망이 된다,

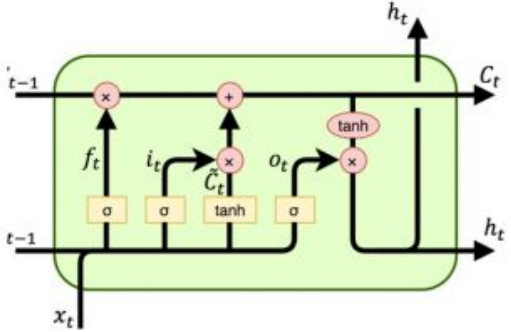
기본적 RNN 의 문제 - **시간 방향으로 심층 신경망**에 따른 **그라디언트 소실**(Vanishing Gradient) 문제가 발생

Solutions ⇒ LSTM or GRU 해결책이 등장하여 RNN의 뒤를 잇는다

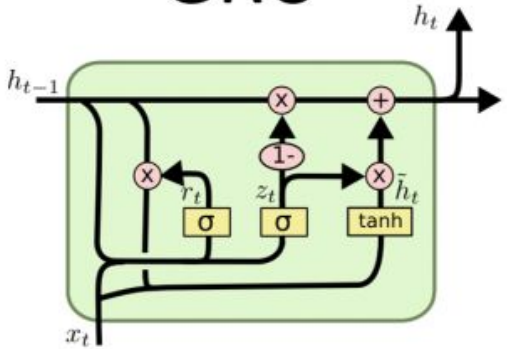
RNN



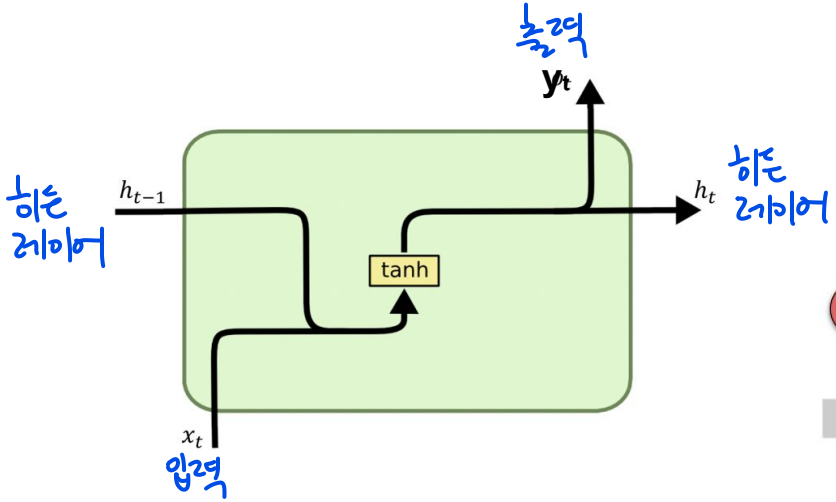
LSTM

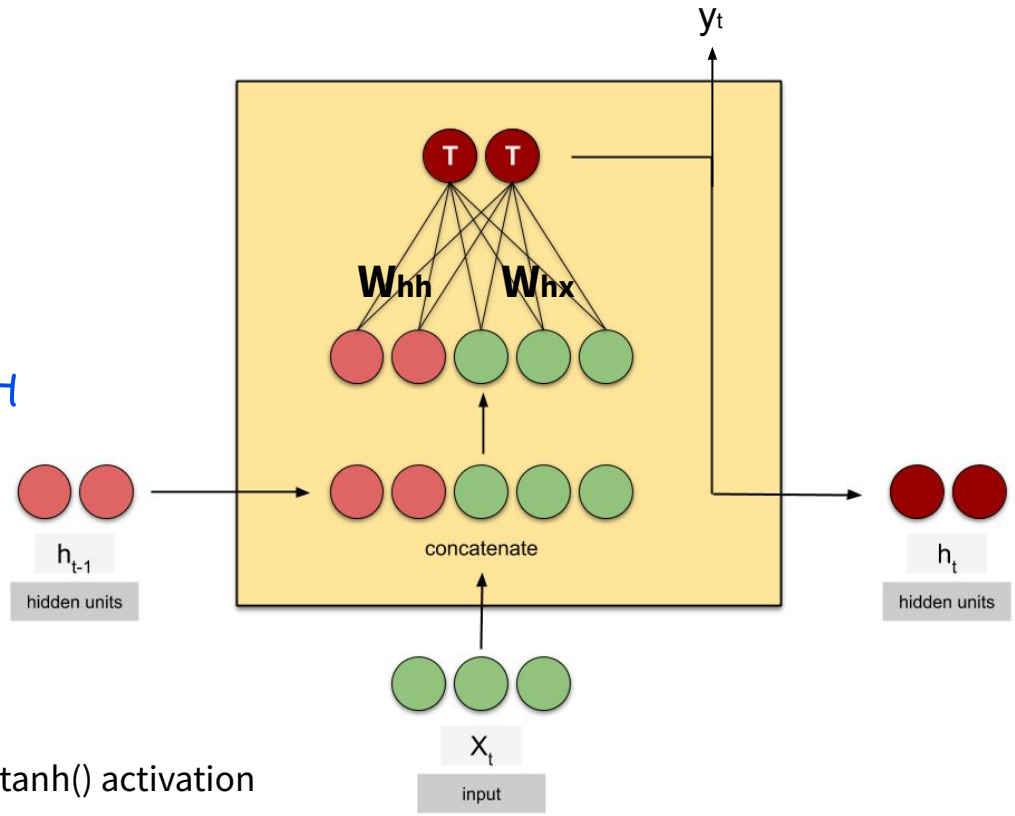


GRU



(Vanilla) RNN



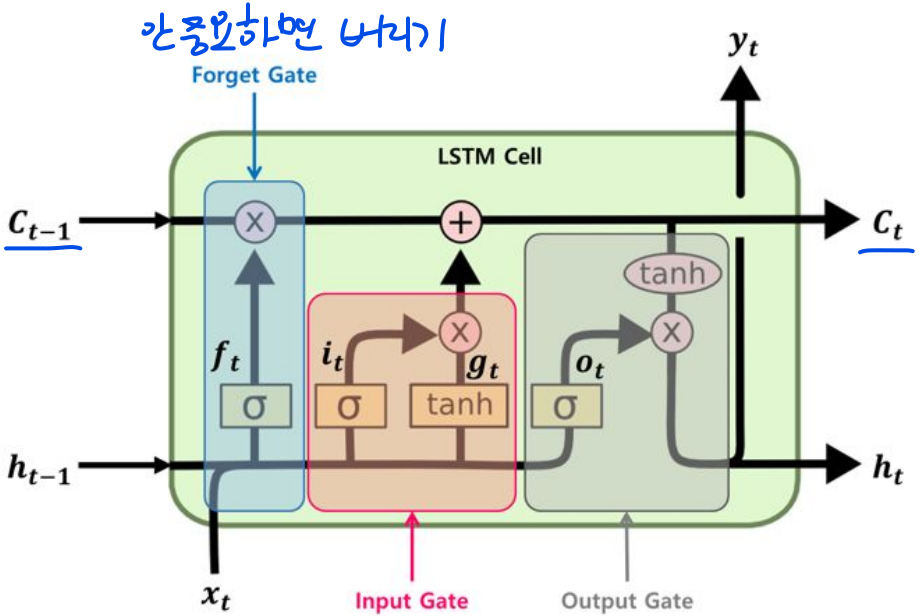


where (T) is tanh() activation

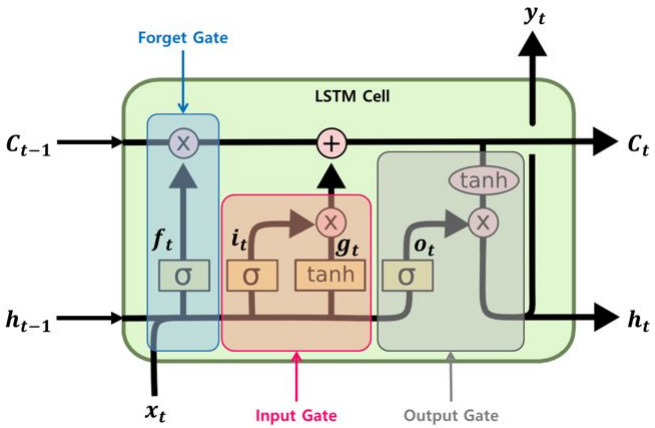
LSTM(Long Short-Term Memory)

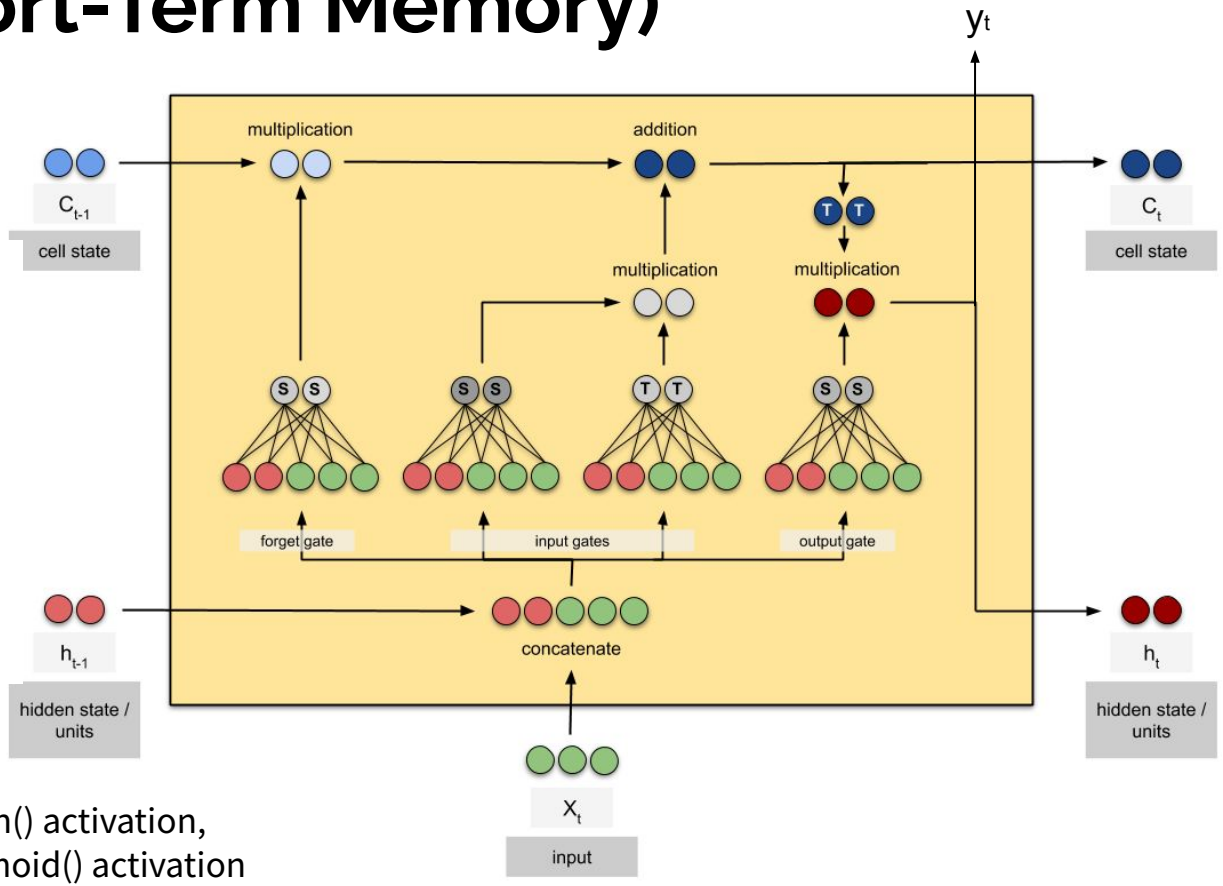
RNN의 특별한 한 종류로, **긴 의존 기간을 필요로 하는 학습을 수행할 능력**을 갖고 있다. LSTM 에서는 각 반복 모듈은 다른 구조를 갖고 있다. 단순한 신경망 한 층 대신에, 4개의 layer가 특별한 방식으로 서로 정보를 주고 받도록 되어 있다.

- Forget gate layer : f_t
- Input gate layer : i_t & g_t
- **Cell state update (장기 상태)** : C_t
- Output gate layer (단기 상태): h_t



LSTM(Long Short-Term Memory)

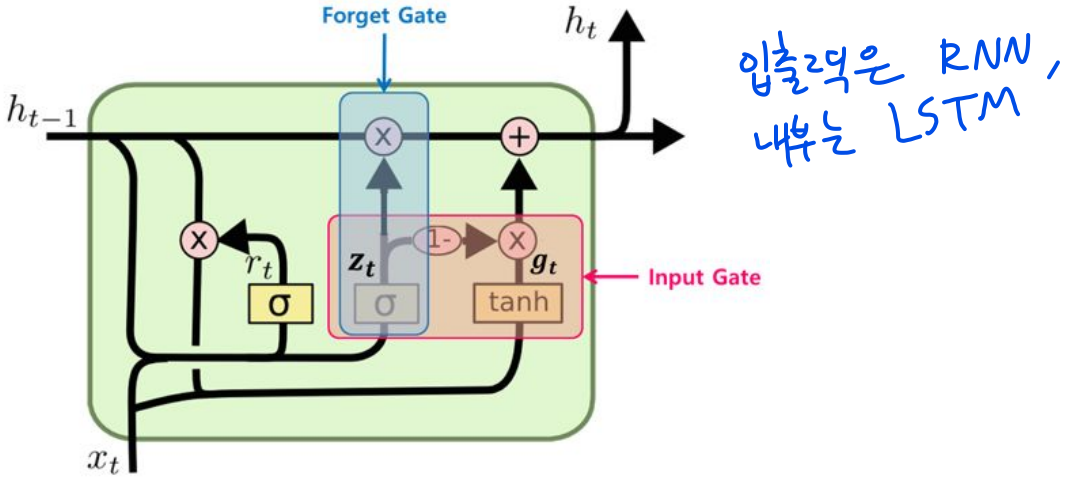




where (T) is tanh() activation,
(s) is sigmoid() activation

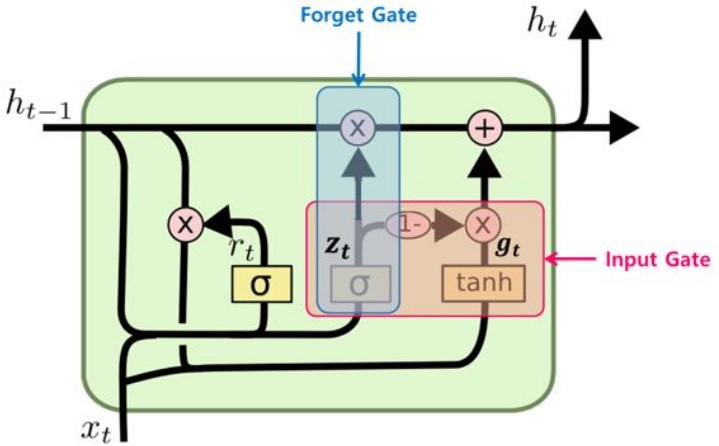
GRU (Gated Recurrent Units)

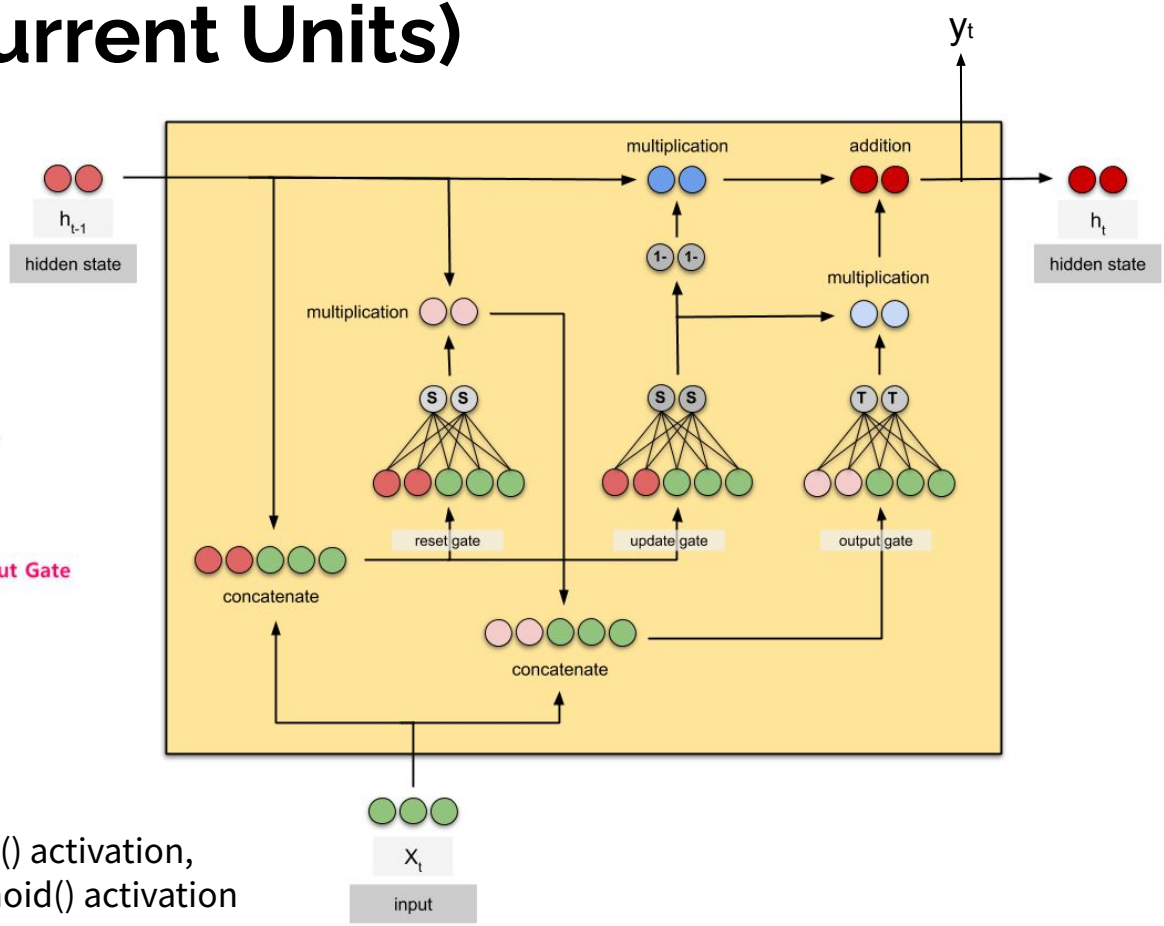
LSTM 셀의 간소화된 버전이라고 할 수 있으며, 다음의 그림과 같은 구조를 가진다. LSTM Cell에서의 두 상태 벡터 $\mathbf{h_t}$ 와 $\mathbf{c_t}$가 하나의 $\mathbf{z_t}$ 벡터 로 합쳐졌다. 하나의 gate controller인 $\mathbf{z_t}$ 가 forget과 input 게이트(gate)를 모두 제어한다.



입출력은 RNN, 내부는 LSTM

GRU (Gated Recurrent Units)





where (T) is tanh() activation,
(S) is sigmoid() activation

RNN을 사용한 텍스트 분류 (demo)

이 텍스트 분류 자습서는 **감정 분석**을 위해 IMDB 대형 영화 검토 데이터 세트 (영화리뷰 훈련자료: 25,000 개, 시험자료: 25,000개) 에서 반복적인 신경망을 학습시킵니다 .

```
import tensorflow_datasets as tfds
import tensorflow as tf
```

`matplotlib` 그래프를 그리는 도우미 함수를 가져 와서 만듭니다.

```
import matplotlib.pyplot as plt

def plot_graphs(history, metric):
  plt.plot(history.history[metric])
  plt.plot(history.history['val_'+metric], '')
  plt.xlabel("Epochs")
  plt.ylabel(metric)
  plt.legend([metric, 'val_'+metric])
  plt.show()
```

입력 파이프 라인 설정

IMDB 대형 영화 리뷰 데이터 세트는 이진 분류 데이터 세트입니다. 모든 리뷰에는 긍정적 또는 부정적 감정이 있습니다.
TFDS를 사용하여 데이터 세트를 다운로드하십시오 .

```
dataset, info = tfds.load('imdb_reviews/subwords8k', with_info=True, as_supervised=True)
train_examples, test_examples = dataset['train'], dataset['test']
```

```
/home/kbuilder/tensorflow_datasets/imdb_reviews/subwords8k/1.0.0으로 데이터 세트 imdb_reviews /
subwords8k / 1.0.0 (다운로드 : 80.23MiB, 생성 : 알 수 없는 크기, 총 : 80.23MiB) 다운로드 및 준비 중 ...

HBox (children = (FloatProgress (value = 1.0, bar_style = 'info', description = 'Dl Completed ...', max
= 1.0, style = Progre…
…
…
HBox (자식 = (플로트 프로그레스 (값 = 0.0, 최대 = 50000.0), HTML (값 = '')))
     데이터 세트 imdb_reviews는 /home/kbuilder/tensorflow_datasets/imdb_reviews/subwords8k/1.0.0에
     다운로드되어 준비되었습니다. 후속 통화는 이 데이터를 재사용합니다.
```

데이터 세트 `info`에는 인코더 (`tfds.features.text.SubwordTextEncoder`)가 포함됩니다.

```
encoder = info.features['text'].encoder
print('Vocabulary size: {}'.format(encoder.vocab_size))
```

```
어휘 크기 : 8185
```

이 텍스트 인코더는 문자열을 가역적으로 인코딩하여 필요한 경우 바이트 인코딩으로 돌아갑니다.

```
sample_string = 'Hello TensorFlow.'

encoded_string = encoder.encode(sample_string)
print('Encoded string is {}'.format(encoded_string))

original_string = encoder.decode(encoded_string)
print('The original string: "{}"'.format(original_string))
```

```
Encoded string is [4025, 222, 6307, 2327, 4043, 2120, 7975]입니다.
The original string: "Hello TensorFlow"
```

```
assert original_string == sample_string
for index in encoded_string:
  print('{} ----> {}'.format(index, encoder.decode([index])))
```

```
4025 ----> Hell
222 ----> o
6307 ----> Ten
2327 ----> sor
4043 ----> Fl
2120 ----> ow
7975 ----> .
```

훈련을 위한 데이터 준비

그런 다음이 인코딩 된 문자열을 일괄적으로 생성하십시오. 이 `padded_batch` 방법을 사용하여 배치에서 가장 긴 문자열의 길이만큼 시퀀스를 0 으로 채웁니다.

```
BUFFER_SIZE = 10000
BATCH_SIZE = 64
train_dataset = (train_examples
                 .shuffle(BUFFER_SIZE)
                 .padded_batch(BATCH_SIZE, padded_shapes=([None],[])))

test_dataset = (test_examples
                .padded_batch(BATCH_SIZE,  padded_shapes=([None],[])))
```

모델 만들기

`tf.keras.Sequential` 모델을 구축하고 임베딩 레이어로 시작합니다 . 임베딩 레이어는 단어 당 하나의 벡터를 저장합니다. 호출되면 단어 인덱스 시퀀스를 벡터 시퀀스로 변환합니다. 이 벡터는 훈련 가능합니다. (충분한 데이터에 대한) 훈련 후 유사한 의미를 가진 단어는 종종 비슷한 벡터를 갖습니다. [단어 임베딩 참조]

이 인덱스 조회는 one-hot 인코딩 된 벡터를 `tf.keras.layers.Dense` 레이어에 전달하는 동등한 작업보다 훨씬 효율적입니다.

RNN (Recurrent Neural Network)은 요소를 반복하여 시퀀스 입력을 처리합니다. RNN은 출력을 한 단계에서 입력으로 전달한 다음 다음 단계로 전달합니다.

`tf.keras.layers.Bidirectional` 랩퍼는 또한 RNN 층으로 사용될 수 있다. 이는 RNN 계층을 통해 입력을 앞뒤로 전파한 다음 출력을 연결합니다. 이는 RNN이 장거리 종속성을 학습하는 데 도움이 됩니다.

```
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(encoder.vocab_size, 64),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(64)),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(1)
])
```

모델의 모든 레이어에는 단일 입력만 있고 단일 출력이 생성되므로 여기서는 Keras 순차 모델을 선택합니다. 상태 저장 RNN 계층을 사용하려는 경우 Keras functional API 또는 모델 서브 클래싱을 사용하여 모델을 빌드하여 RNN 계층 상태를 검색하고 재사용 할 수 있습니다. 자세한 내용은 Keras RNN 안내서 를 확인하십시오.

Keras 모델을 컴파일하여 교육 프로세스를 구성하십시오.

```
model.compile(loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
              optimizer=tf.keras.optimizers.Adam(1e-4),
              metrics=['accuracy'])
```

```
model.summary()
```

```
Model: "sequential 4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
embedding 4 (Embedding)      (None, None, 64)          523840    
_________________________________________________________________
bidirectional 5 (Bidirection (None, 128)               66048     
_________________________________________________________________
dense 8 (Dense)              (None, 64)                8256      
_________________________________________________________________
dense 9 (Dense)              (None, 1)                 65        
=================================================================
Total params: 598,209
Trainable params: 598,209
Non-trainable params: 0
_________________________________________________________________
```

임베딩 레이어

⇐ 8185 x 64

⇐ (64 x 128 + 64) x 8

⇐ (128 + 1) x 64

⇐ 64 + 1

모델 훈련

```
history = model.fit(train_dataset, epochs=10,
                    validation_data=test_dataset,
                    validation_steps=30)
```

```
에포크 1/10
391/391 [==============================-41s 104ms / step-손실 : 0.6525-정확도 : 0.5517-val_loss :
0.4473-val_accuracy : 0.8089
에포크 2/10
391/391 [==============================-41s 105ms / step-손실 : 0.3416-정확도 : 0.8562-val_loss :
0.3455-val_accuracy : 0.8661
.. .. ..
에포크 10/10
391/391 [==============================-41s 106ms / step-손실 : 0.1132-정확도 : 0.9629-val_loss :
0.4338-val_accuracy : 0.8604
```

```
test_loss, test_acc = model.evaluate(test_dataset)

print('Test Loss: {}'.format(test_loss))
print('Test Accuracy: {}'.format(test_acc))
```

```
391/391 [==============================-17s 44ms / step-손실 : 0.4371-정확도 : 0.8585
테스트 손실 : 0.4370759129524231
테스트 정확도 : 0.858519971370697
```

위의 모델은 시퀀스에 적용된 패딩을 가리지 않습니다. 패딩된 시퀀스를 학습하고 패딩되지 않은 시퀀스를 테스트하면 왜곡될 수 있습니다. 이상적으로는 이것을 피하기 위해 마스킹을 사용하지만 아래에서 볼 수 있듯이 출력에는 약간의 영향만 미칩니다.
예측이 >= 0.5 인 경우, 양수(긍정적)이고 그렇지 않으면 음수(부정적)입니다.

```
def pad_to_size(vec, size):
  zeros = [0] * (size - len(vec))
  vec.extend(zeros)
  return vec

def sample_predict(sample_pred_text, pad):
  encoded_sample_pred_text = encoder.encode(sample_pred_text)

  if pad:
    encoded_sample_pred_text = pad_to_size(encoded_sample_pred_text, 64)
  encoded_sample_pred_text = tf.cast(encoded_sample_pred_text, tf.float32)
  predictions = model.predict(tf.expand_dims(encoded_sample_pred_text, 0))

  return (predictions)
# predict on a sample text without padding.

sample_pred_text = ('The movie was cool. The animation and the graphics '
                    'were out of this world. I would recommend this movie.')
predictions = sample_predict(sample_pred_text, pad=False)
print(predictions)
```

```
[[0.06850696]]
```

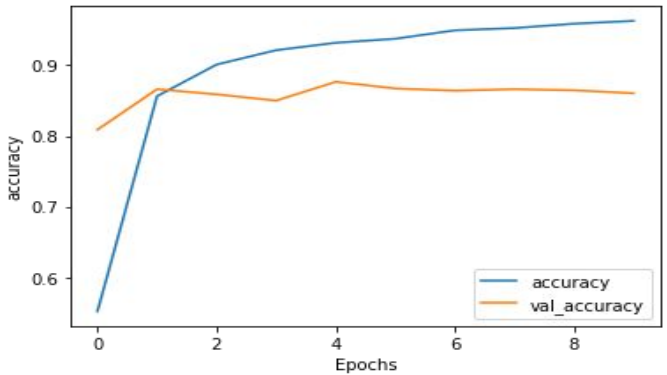
패딩X

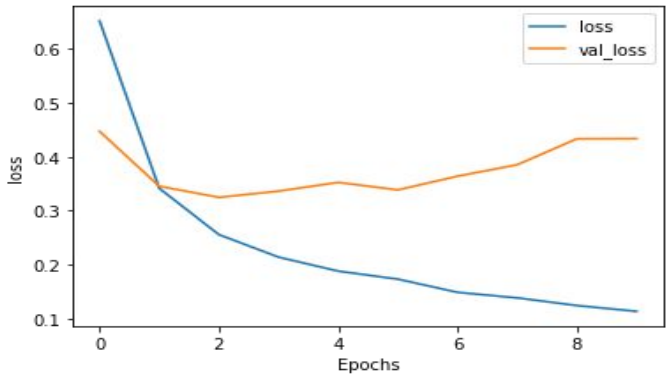
```
# predict on a sample text with padding

sample_pred_text = ('The movie was cool. The animation and the graphics '
                    'were out of this world. I would recommend this movie.')
predictions = sample_predict(sample_pred_text, pad=True)
print(predictions)
```

```
[[-0.67055]]
```

부정이야

```
plot_graphs(history, 'accuracy')
plot_graphs(history, 'loss')
```





두 개 이상의 LSTM 레이어 쌓기

Keras 반복 레이어에는 `return_sequences` 생성자 인수로 제어되는 두 가지 사용 가능한 모드가 있습니다 .

- 각 타임 스텝 (3D 모양의 텐서)에 대한 전체 연속 출력 시퀀스를 반환합니다 `(batch_size, timesteps, output_features)` .
- 각 입력 시퀀스에 대한 마지막 출력만 반환합니다 (2D 형태의 텐서 (batch_size, output_features)).

```
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(encoder.vocab_size, 64),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(64,  return_sequences=True)),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(32)),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(1)
])
model.compile(loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
              optimizer=tf.keras.optimizers.Adam(1e-4),
              metrics=['accuracy'])
history = model.fit(train_dataset, epochs=10,
                    validation_data=test_dataset,
                    validation_steps=30)
```

과적합 방지

```
에포크 1/10
391/391 [==============================-72s 185ms / step-손실 : 0.6648-정확도 : 0.5468-val_loss :
0.5265-val_accuracy : 0.7516
. . .
에포크 10/10
391/391 [==============================--75s 191ms / step-손실 : 0.1078-정확도 : 0.9720-val_loss :
0.5088-val_accuracy : 0.8495
```

```
test_loss, test_acc = model.evaluate(test_dataset)

print('Test Loss: {}'.format(test_loss))
print('Test Accuracy: {}'.format(test_acc))
```

```
391/391 [==============================-31s 80ms / step-손실 : 0.4796-정확도 : 0.8505
테스트 손실 : 0.47960710525512695
테스트 정확도 : 0.8504800200462341
```

```
# predict on a sample text without padding.
sample_pred_text = ('The movie was not good. The animation and the graphics '
                    'were terrible. I would not recommend this movie.')
predictions = sample_predict(sample_pred_text, pad=False)
print(predictions)
```

```
[[-2.6095185]]
```

패딩 X

```
# predict on a sample text without padding.
sample_pred_text = ('The movie was good. The animation and the graphics '
                    'were fantastic. I would recommend this movie.')
predictions = sample_predict(sample_pred_text, pad=False)
print(predictions)
```

```
[[0.69282794]]
```
// X

```
# predict on a sample text with padding
sample_pred_text = ('The movie was not good. The animation and the graphics '
                    'were terrible. I would not recommend this movie.')
predictions = sample_predict(sample_pred_text, pad=True)
print(predictions)
```

```
[[-4.1003203]]
```
// O

```
# predict on a sample text with padding
sample_pred_text = ('The movie was good. The animation and the graphics '
                    'were fantastic. I would recommend this movie.')
predictions = sample_predict(sample_pred_text, pad=True)
print(predictions)
```

```
[[3.4226665]]
```
// O

요약 : 패딩을 적용하면 결과가 더 명확함

(Option)Sample RNN code in Python-tensorflow

```
. . .
model = tf.keras.Sequential([
    tf.keras.layers.SimpleRNN(50, input_shape=(49,1))  ,
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense( 1)
])

model.compile(loss='categorical_crossentropy', optimizer='Adam')
lmodel.fit(X_train, y_train)
y_pred = model.predict(X_test)

. . .
```

(Option)Sample RNN code in Python-Keras

- **One-to-one**: you could use a Dense layer as you are not processing sequences:

  ```
  model.add(Dense(output_size, input_shape=input_shape))
  ```

- **One-to-many**: this option is not supported well as chaining models is not very easy in Keras, so the following version is the easiest one:

  ```
  model.add(RepeatVector(number_of_times, input_shape=input_shape))
  model.add(LSTM(output_size,  return_sequences=True))
  ```

- **Many-to-one**: actually, your code snippet is (almost) an example of this approach:

  ```
  model = Sequential()
  model.add(LSTM(1, input_shape=( timesteps, data_dim)))
  ```

- **Many-to-many**: This is the easiest snippet when the length of the input and output matches the number of recurrent steps:

  ```
  model = Sequential()
  model.add(LSTM(1, input_shape=( timesteps, data_dim), return_sequences=True))
  ```