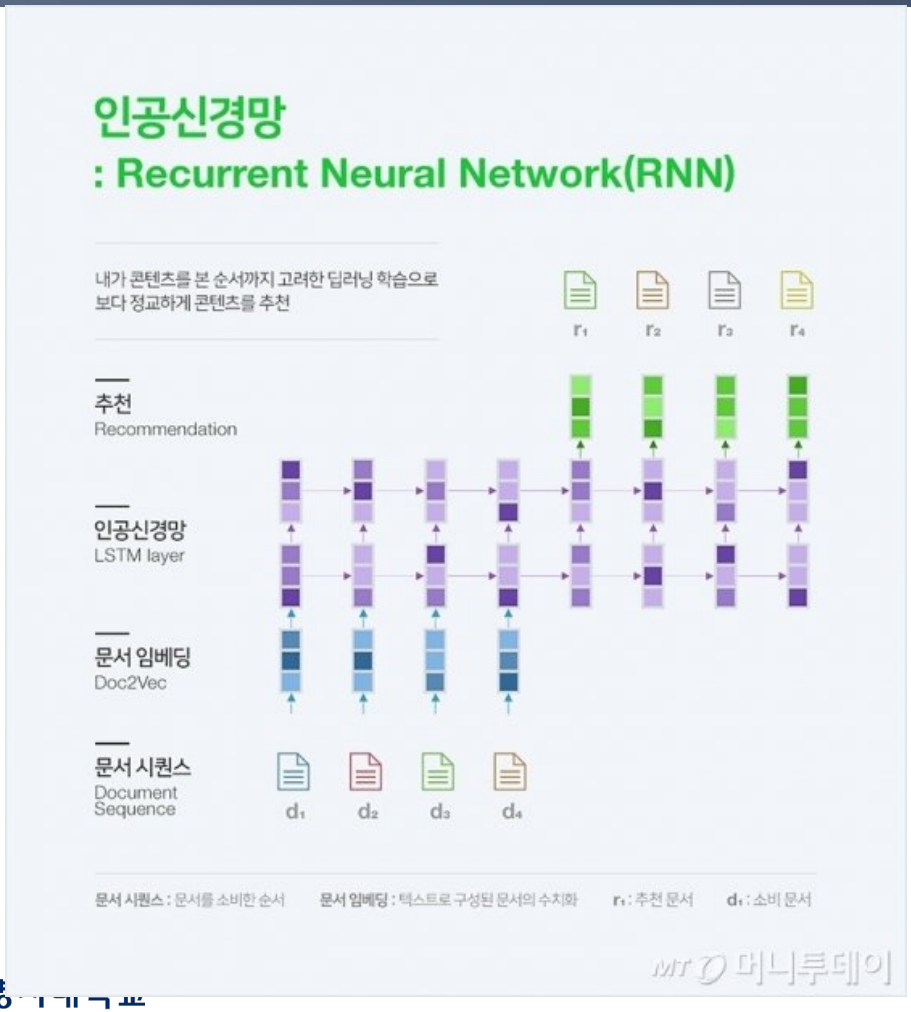
# Recurrent Neural Network (RNN)

순환 신경망

# 네이버 모바일 뉴스, 딥러닝 기술 적용..."사용자별 맞춤 추천"




네이버가 모바일 뉴스에 새로운 딥러닝 기술을 적용, 콘텐츠 추천 기술력을 한 단계 끌어 올린다.

네이버는 28일 인공지능 콘텐츠 추천 시스템 '에어스'(AiRS)에 딥러닝 기반의 인공신경망 기술 'RNN'(Recurrent Neural Network)을 추가, 모바일 뉴스 판에 시범 적용한다고 밝혔다.

기존의 에어스는 CF(협력필터) 기술을 중심으로 비슷한 관심을 가진 사용자 그룹을 시시각각 생성해 이들이 많이 읽은 뉴스를 랭킹화해 추천했다. 새롭게 추가된 RNN 기술은 사용자 개인의 뉴스 소비 '패턴'을 학습, 예측해 맥락에 따라 뉴스를 추천해준다. 이를 통해 이용자는 본인이 읽었던 뉴스와 관련 있고 한 단계 더 깊이 있는 뉴스들을 접할 수 있을 것으로 보인다.

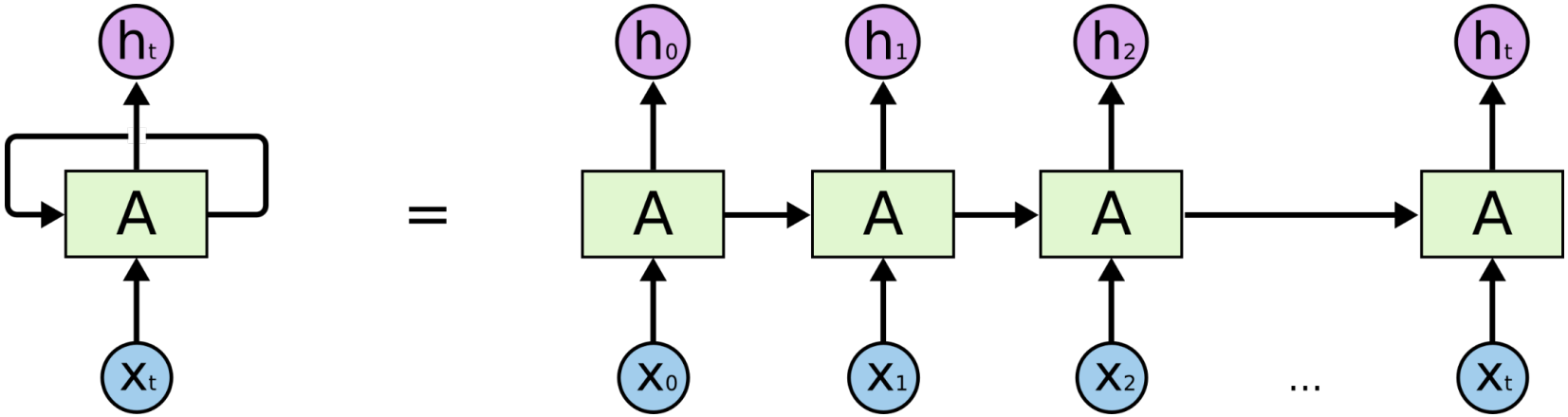
네이버는 CF기술과 RNN 기술 각각의 장점을 융합한 AiRS를 모바일 뉴스판의 '에어스 추천 뉴스(베타)' 영역에 적용, 다양성 뿐만 아니라 심도 있는 뉴스도 추천한다는 방침이다.

https://news.mt.co.kr/mtview.php?no=2017092816573536073

# 로이터 뉴스 분류하기

- 총 11,258개의 뉴스 기사
- 46개의 뉴스 카테고리로 분류되는 뉴스 기사 데이터

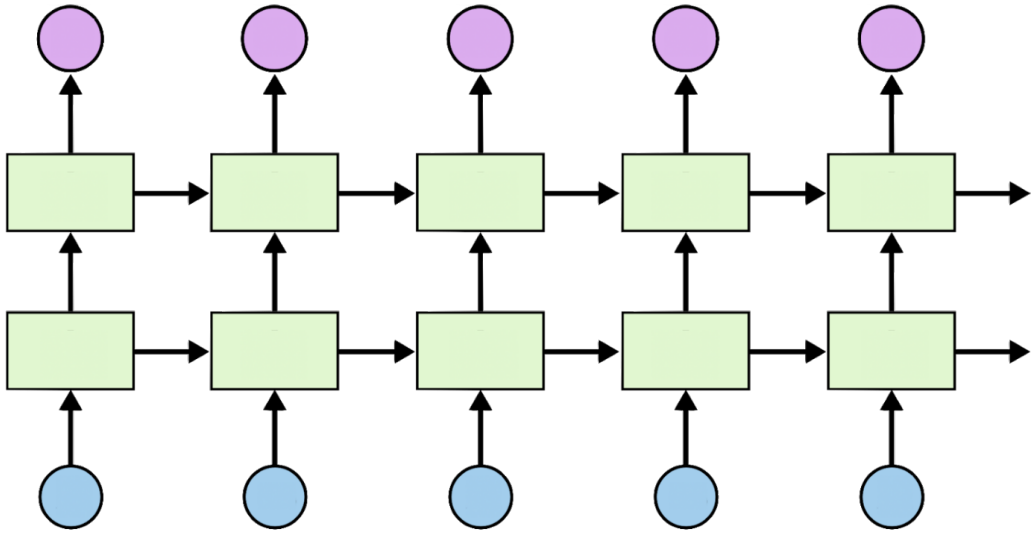
# Vanilla RNN




```
model = Sequential()
model.add(SimpleRNN(50, input_shape=(49, 1), return_sequences=False))
model.add(Dense(46))
model.add(Activation('softmax'))
```

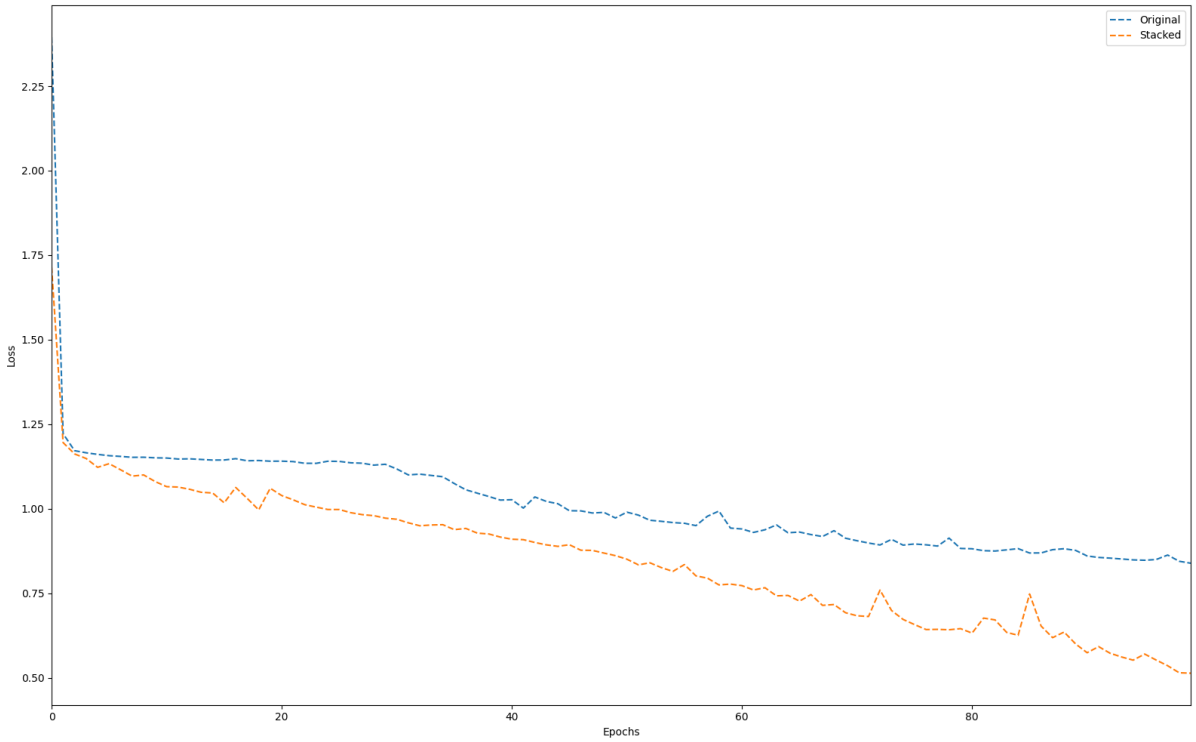
# Stacked Vanilla RNN

## 딥러닝의 핵심 아이디어틀 wide하고 deep 하게 쌓기

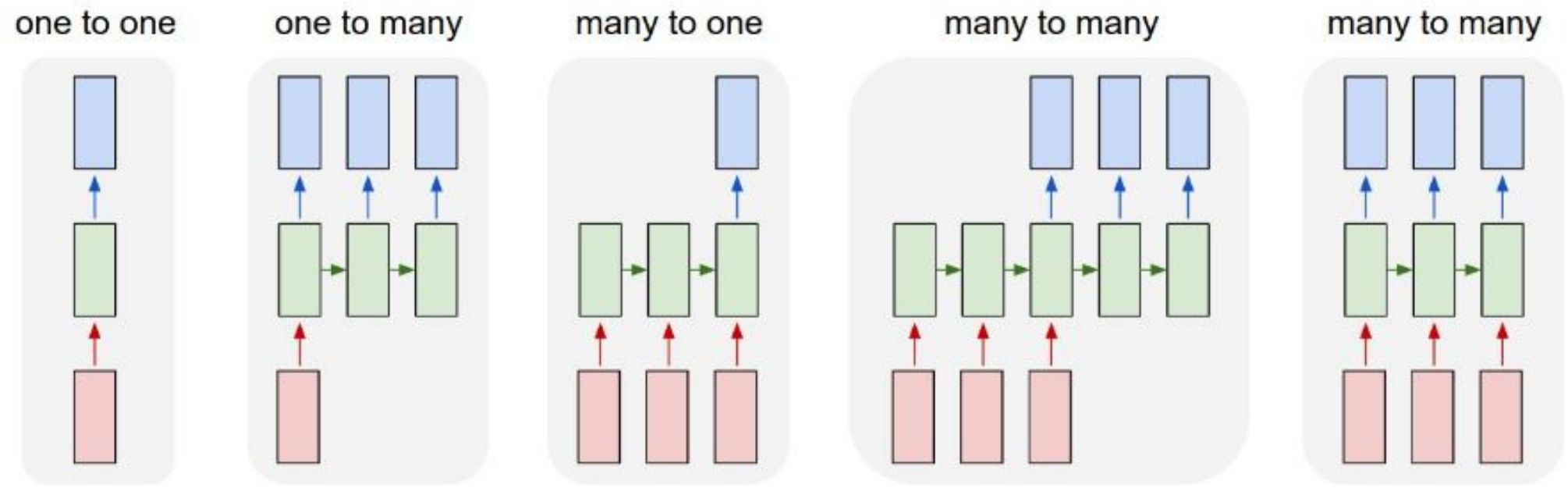


```
model = Sequential()
model.add(SimpleRNN(50, input_shape=(49, 1), return_sequences=True))
model.add(SimpleRNN(50, return_sequences= True))
model.add(SimpleRNN(50, return_sequences=False))
model.add(Dense(46))
model.add(Activation('softmax'))
```

# Stacked Vanilla RNN

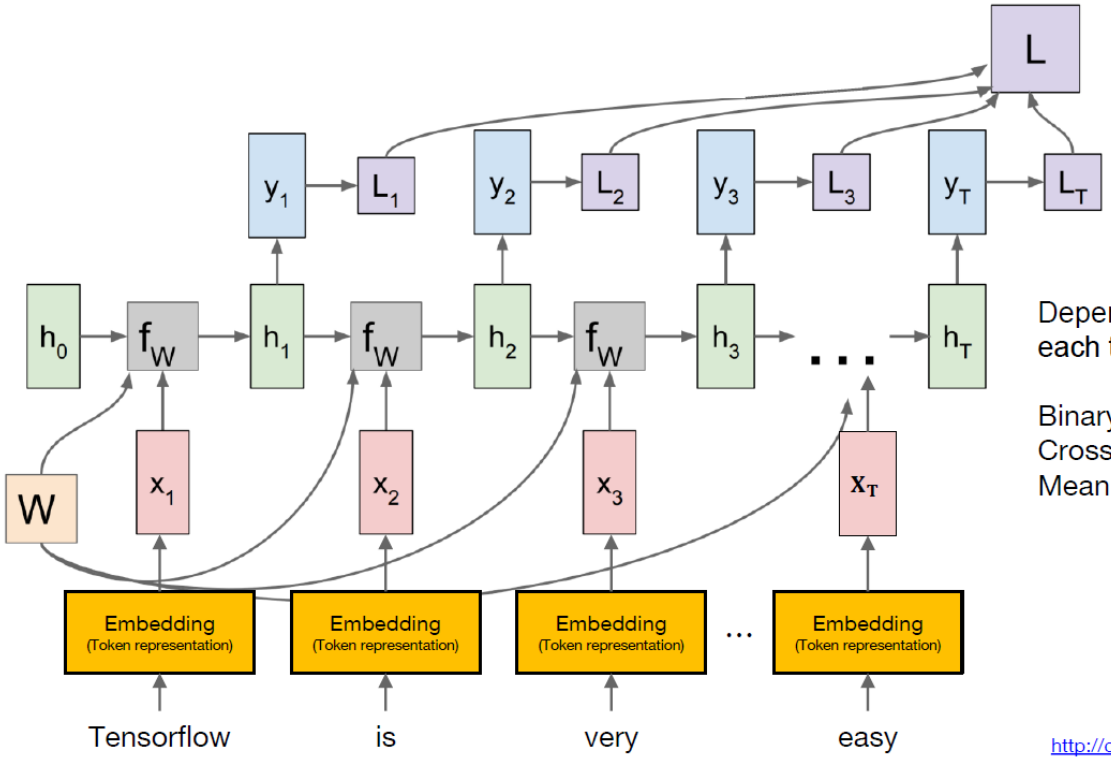


# 다양한 RNN 형태



# Many to many

## What is “many to many”?




Depending on the task with regard to each time step…

Binary entropy loss (binary classifier)
Cross entropy loss (softmax classifier)
Mean squared loss (regression)

aka Sequence loss!

http://cs231n.stanford.edu/slides/2018/cs231n_2018_lecture10.pdf

# 패딩 시퀀스 데이터

## 패딩

- 앞서 말한 대로 스텝 수를 맞추는 과정
- 이미지에 있는 zero-padding처럼, 모자란 스텝에 해당하는 만큼을 무의미한 값으로 채우는 것

텍스트가 단어로 토큰 화

```
[
  ["Hello", "world", "!"],
  ["How", "are", "you", "doing", "today"],
  ["The", "weather", "will", "be", "nice", "tomorrow"],
]
```

```
[
  [71, 1331, 4231]
  [73, 8, 3215, 55, 927],
  [83, 91, 1, 645, 1253, 927],
]
```

데이터는 정수로 벡터화

# 패딩 시퀀스 데이터

```
raw_inputs = [
    [711, 632, 71],
    [73, 8, 3215, 55, 927],
    [83, 91, 1, 645, 1253, 927],
]
```

```
padded_inputs = tf.keras.preprocessing.sequence.pad_sequences(  raw_inputs, padding="post")
```

```
[[ 711 632 71 0 0 0]
 [ 73 8 3215 55 927 0],
 [ 83 91 1 645 1253 927]]
```

데이터는 정수로 벡터화

# 시퀀스 마스킹

## 시퀀스 마스킹

- 이제 모든 샘플의 길이가 균일하므로 데이터의 일부가 실제로 채워지고 무시되어야 한다는 사실을 모델에 알려야합니다. 그 메커니즘은 마스킹 입니다.
- Keras 모델에 입력 마스크를 도입하는 세 가지 방법이 있습니다.

1. keras.layers.Masking 레이어를 추가합니다.
2. mask_zero=True 로 keras.layers.Embedding 레이어를 구성합니다.
3. 이 인수를 지원하는 레이어 (예 : RNN 레이어)를 호출 할 때 mask 인수를 수동으로 전달합니다.

```
embedding = layers.Embedding(input_dim=5000, output_dim=16, mask_zero=True)
masked_output = embedding(padded_inputs)

print(masked_output._keras_mask)
```

# 시퀀스 마스킹

## 시퀀스 마스킹

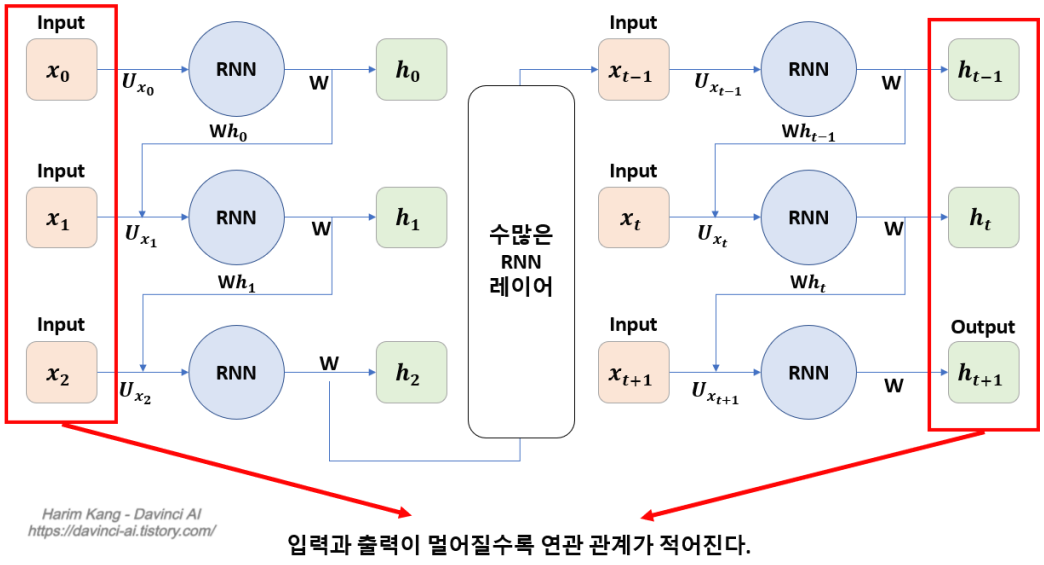
- 이제 모든 샘플의 길이가 균일하므로 데이터의 일부가 실제로 채워지고 무시되어야 한다는 사실을 모델에 알려야합니다. 그 메커니즘은 마스킹 입니다.
- Keras 모델에 입력 마스크를 도입하는 세 가지 방법이 있습니다.

1. keras.layers.Masking 레이어를 추가합니다.
2. mask_zero=True 로 keras.layers.Embedding 레이어를 구성합니다.
3. 이 인수를 지원하는 레이어 (예 : RNN 레이어)를 호출 할 때 mask 인수를 수동으로 전달합니다.

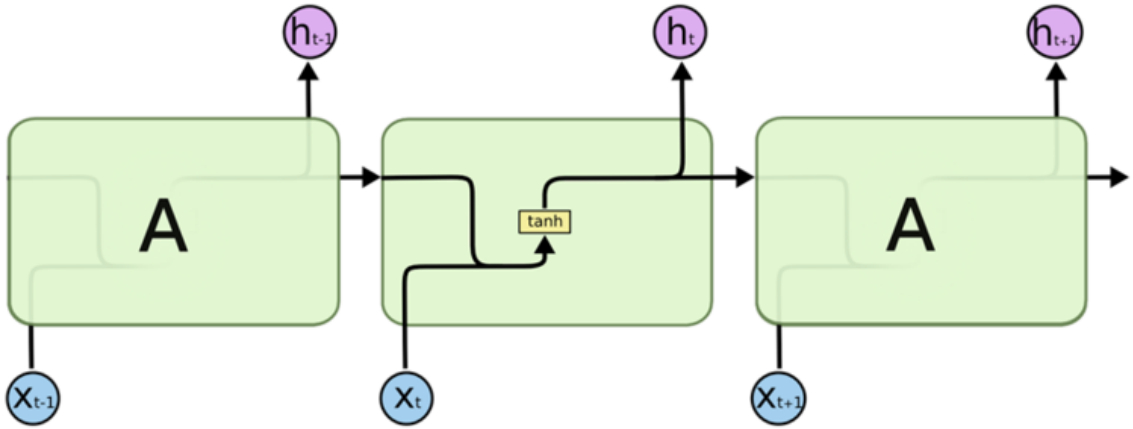
```
masking_layer = layers.Masking()
# Simulate the embedding lookup by expanding the 2D input to 3D,
# with embedding dimension of 10.
unmasked_embedding = tf.cast(
    tf.tile(tf.expand_dims(padded_inputs, axis=-1), [1, 1, 10]), tf.float32
)

masked_embedding = masking_layer(unmasked_embedding)
print(masked_embedding._keras_mask)
```

# RNN 단점

- 입력 데이터가 커지면, 학습 능력이 저하
- 데이터의 뒤쪽으로 갈수록, 앞쪽의 입력 데이터 정보 소실
- 입력 데이터와 출력 데이터 사이의 길이가 멀어질수록 연관 관계가 줄어듦
  - 장기 의존성(Long-Term Dependency)문제

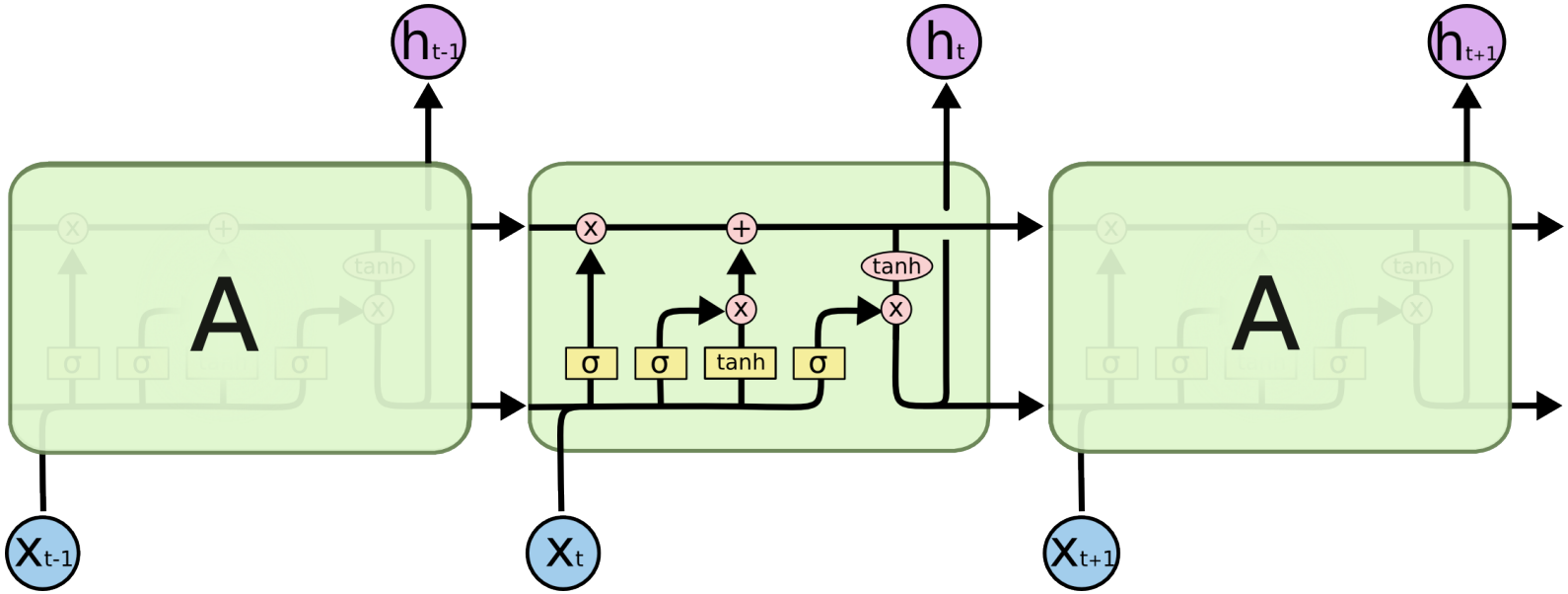


# Vanila RNN



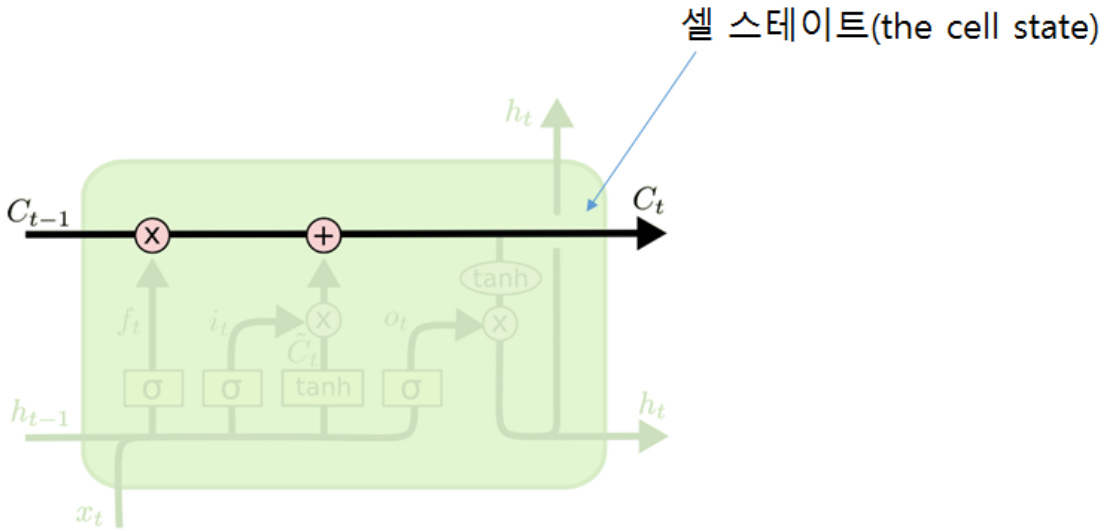
# LSTM

- LSTM은 RNN의 히든 state에 cell-state를 추가한 구조



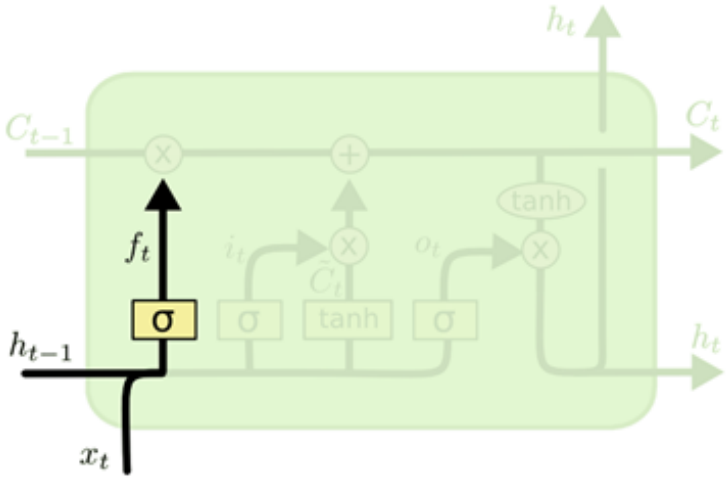
# LSTM

- cell state는 일종의 컨베이어 벨트 역할
- state가 꽤 오래 경과하더라도 그래디언트가 비교적 전파
- ⊙는 요소별 곱셈을 뜻하는 Hadamard product 연산자



# LSTM

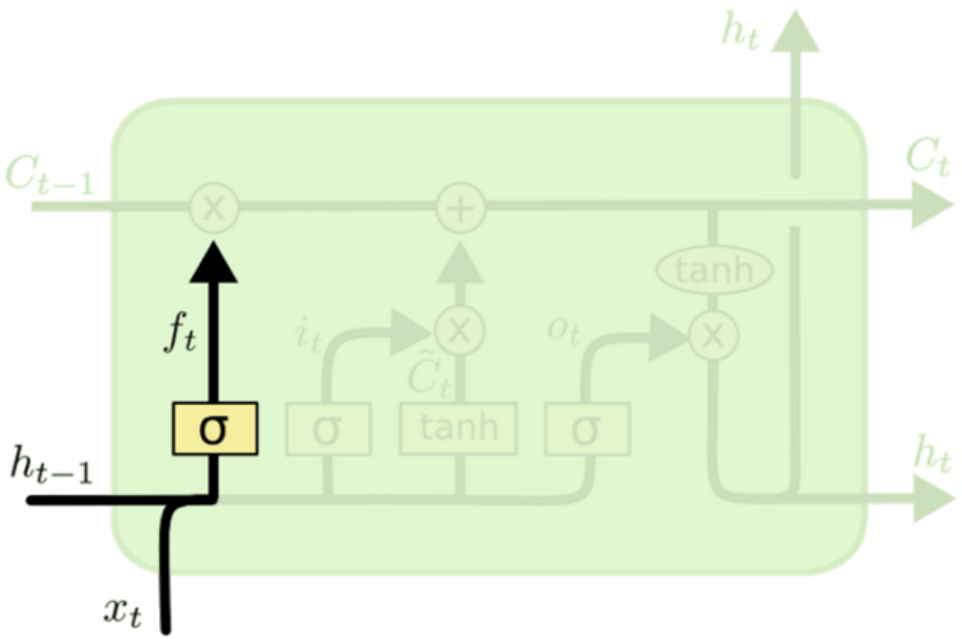
## forget gate layer(시그모이드 레이어)



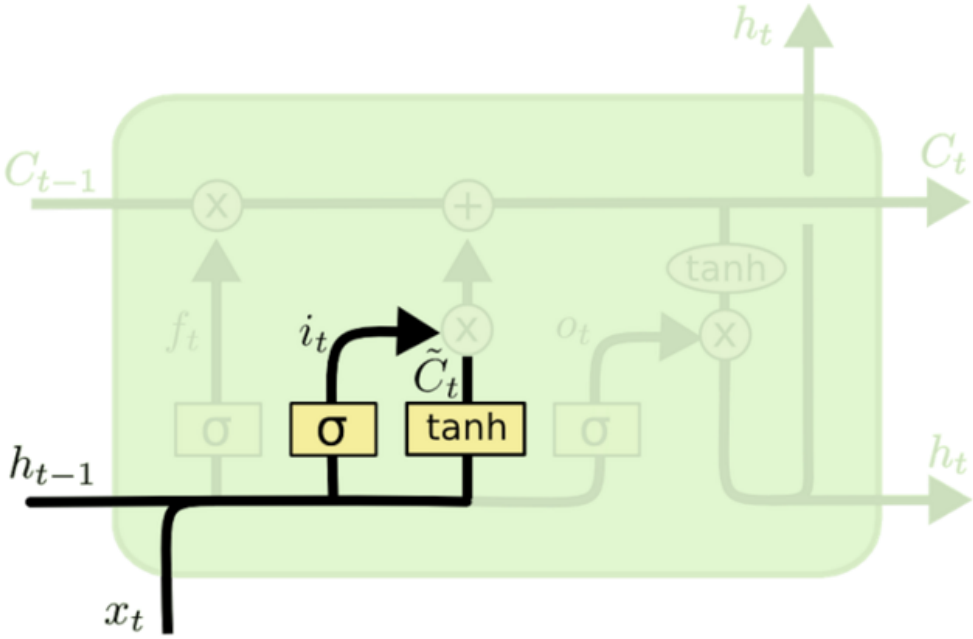

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

- $h_{t-1}$과 $x_t$를 받아 시그모이드를 취해준 값
- 시그모이드 함수의 출력 범위는 0에서 1 사이이기 때문에 그 값이 0이라면 이전 상태의 정보는 잊고, 1이라면 이전 상태의 정보를 온전히 기억

# LSTM




<forget gate>

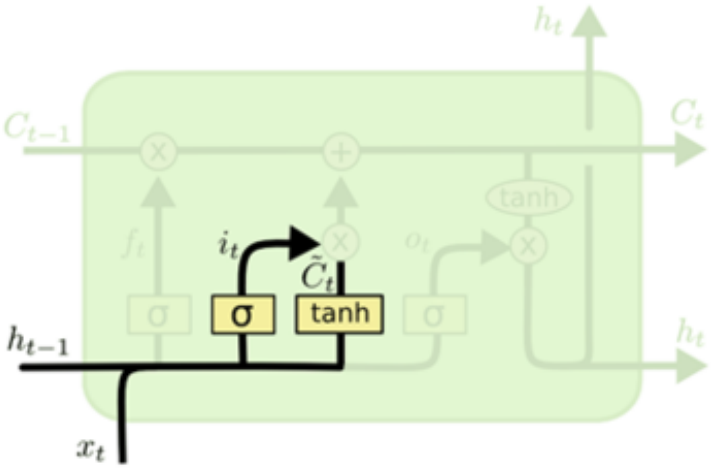



<input gate>

- 과거 정보를 잊기'를 위한 게이트
- $h_{t-1}$과 $x_t$를 받아 시그모이드를 취해준 값
- 시그모이드 함수의 출력 범위는 0에서 1 사이이기 때문에 그 값이 0이라면 이전 상태의 정보는 잊고, 1이라면 이전 상태의 정보를 온전히 기억

# LSTM

## input gate layer(시그모이드 레이어)

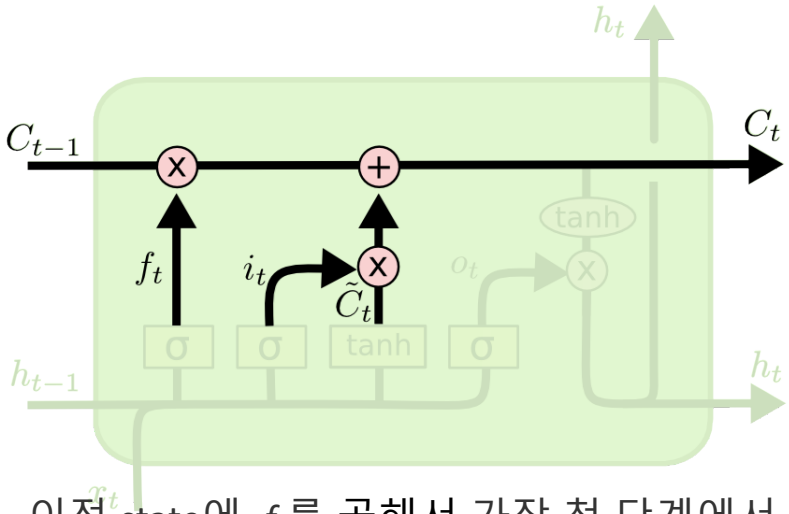



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

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

- ' 현재 정보를 기억하기' 위한 게이트
- $h_{t-1}$과 $x_t$를 받아 시그모이드를 취해준 값과 또 같은 입력으로 하이퍼볼릭탄젠트를 취해준 값 구하기
- Hadamard product 연산 (⊙는 요소별 곱셈)
- $i_t$의 범위는 0~1, $\tilde{C}_t$ 의 범위는 -1~1이기 때문에 각각 강도와 방향

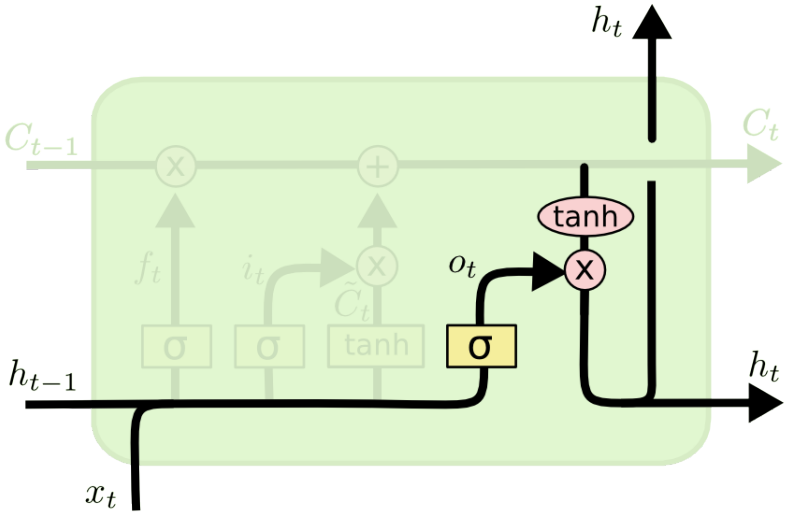
# LSTM




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

- 이전 state에 $f_t$를 곱해서 가장 첫 단계에서 잊어버리기로 정했던 것들을 진짜로 잊어 버림
- $i_t * \tilde{C}_t$ 를 더한다. (두 번째 단계에서 업데이트하기로 한 값을 얼마나 업데이트할 지 정한 만큼 scale한 값)

# LSTM

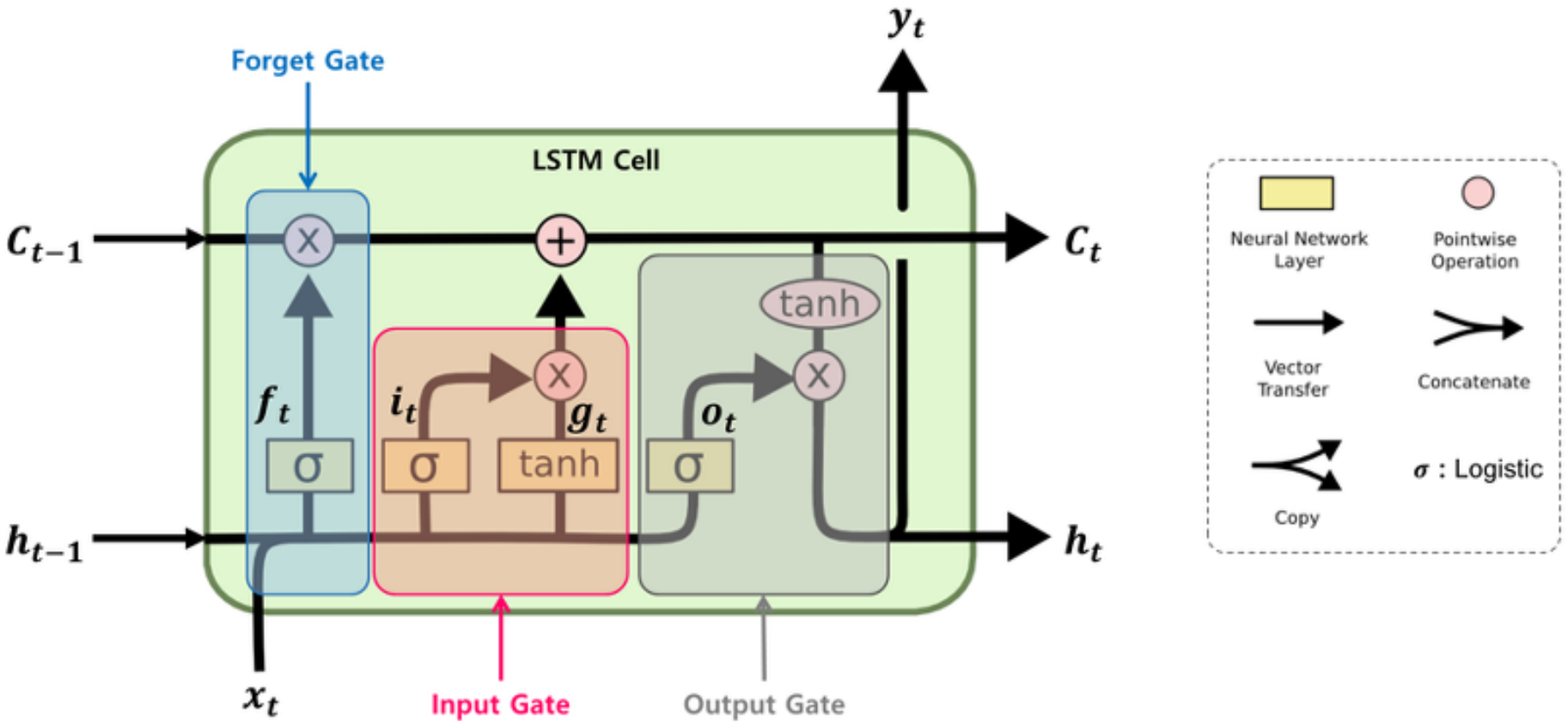



$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

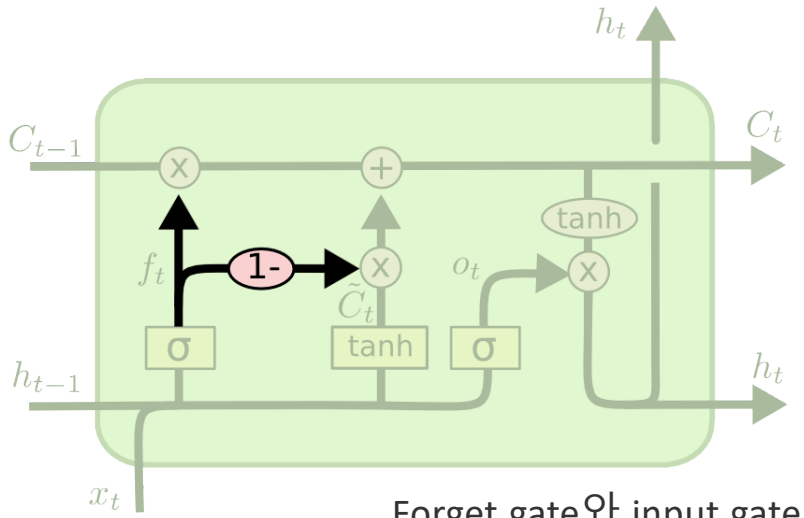
$$h_t = o_t * \tanh\left(C_t\right)$$

- sigmoid layer에 input 데이터를 태워서 cell state의 어느 부분을 output으로 내보낼 지를 정하기
- cell state를 tanh layer에 태워서 -1과 1 사이의 값을 받은 뒤에 방금 전에 계산한 sigmoid gate의 output과 곱하기

# LSTM


$y_t$
Forget Gate
LSTM Cell
$C_{t-1}$
$C_t$
$f_t$
$i_t$
$g_t$
$o_t$
σ
σ
tanh
tanh
σ
$h_{t-1}$
$h_t$
$x_t$
Input Gate
Output Gate
Neural Network Layer
Pointwise Operation
Vector Transfer
Concatenate
Copy
σ : Logistic

# LSTM의 변형 모델들 (GRU)




Forget gate와 input gate가 합쳐진 LSTM

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

- forget gate와 input gate를 하나의 "update gate" 합치기
- cell state를 tanh layer에 태워서 -1과 1 사이의 값을 받은 뒤에 방금 전에 계산한 sigmoid gate의 output과 곱하기

https://arxiv.org/pdf/1406.1078v3.pdf

# 원핫인코딩

- 간단하게 피처 값의 유형에 따라 새로운 피처를 추가해 고유 값에 해당하는 칼럼에만 1을 표시하고 나머지 칼럼에는 0을 표시하는 방법

Label Encoding

| Food Name | Categorical # | Calories |
|---|---|---|
| Apple | 1 | 95 |
| Chicken | 2 | 231 |
| Broccoli | 3 | 50 |

→

One Hot Encoding

| Apple | Chicken | Broccoli | Calories |
|---|---|---|---|
| 1 | 0 | 0 | 95 |
| 0 | 1 | 0 | 231 |
| 0 | 0 | 1 | 50 |

# 임베딩

- 범주형 자료를 연속형 벡터 형태로 변환시키는 것
- 주로 인공신경망 학습을 통해 범주형 자료를 벡터 형태로 변환
- 인공신경망 학습 과정을 통해 각 벡터에 가장 알맞는 값을 할당
- t-SNE와 같은 방법을 활용하면 이 벡터들을 시각화해서 표현할 수도 있으며, 매우 직관적으로 표현

# 임베딩

## 장점

- 차원을 축소할 수 있다.
  범주형 자료를 one-hot-encoding으로 표현할 경우 n-1개의 차원이 필요한 반면, embedding을 활용하면 2,3차원으로도 자료를 표현할 수 있다.
- 범주형 자료를 연속형으로 표현할 수 있다
- 의미를 도출하기에 용이하다

# 원핫인코딩 vs 임베딩

| Index | Word | Ont-hot vector | Embedding vector |
|---|---|---|---|
| 0 | 영웅은 | [1 0 0 0 0] | [0.1 4.2 1.5 1.1 2.8] |
| 1 | 죽지 | [0 1 0 0 0] | [1.0 3.1 2.5 0.7 1.1] |
| 2 | 않아요 | [0 0 1 0 0] | [0.3 2.1 1.5 2.1 0.1] |
| 3 | 너만 | [0 0 0 1 0] | [2.2 1.4 0.5 0.9 1.1] |
| 4 | 빼고 | [0 0 0 0 1] | [0.7 1.7 0.5 0.3 0.2] |

| | 한 벡터 | 임베딩 벡터 |
|---|---|---|
| 차원 | 고차원(단어 집합의 크기) | 저차원 |
| 다른 표현 | 희소 벡터의 일종 | 밀집 벡터의 일종 |
| 표현 방법 | 수동 | 훈련 데이터로부터 학습함 |
| 값의 타입 | 1과 0 | 실수 |