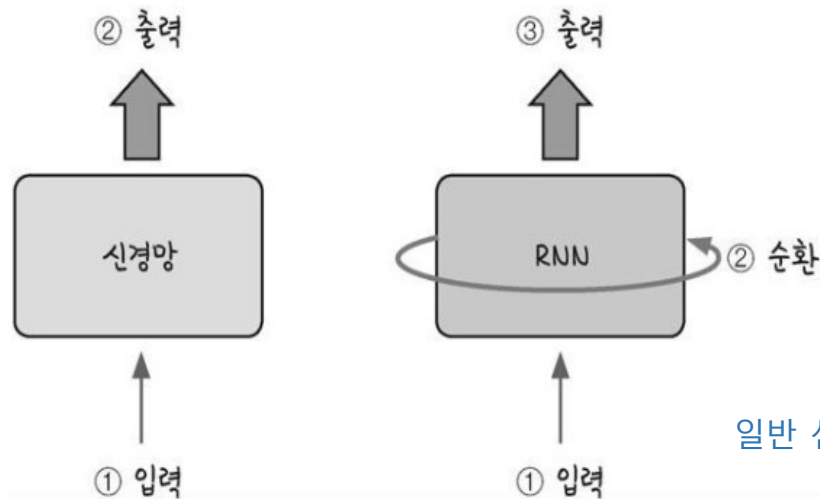


순환 신경망(Recurrent Neural Network, RNN)

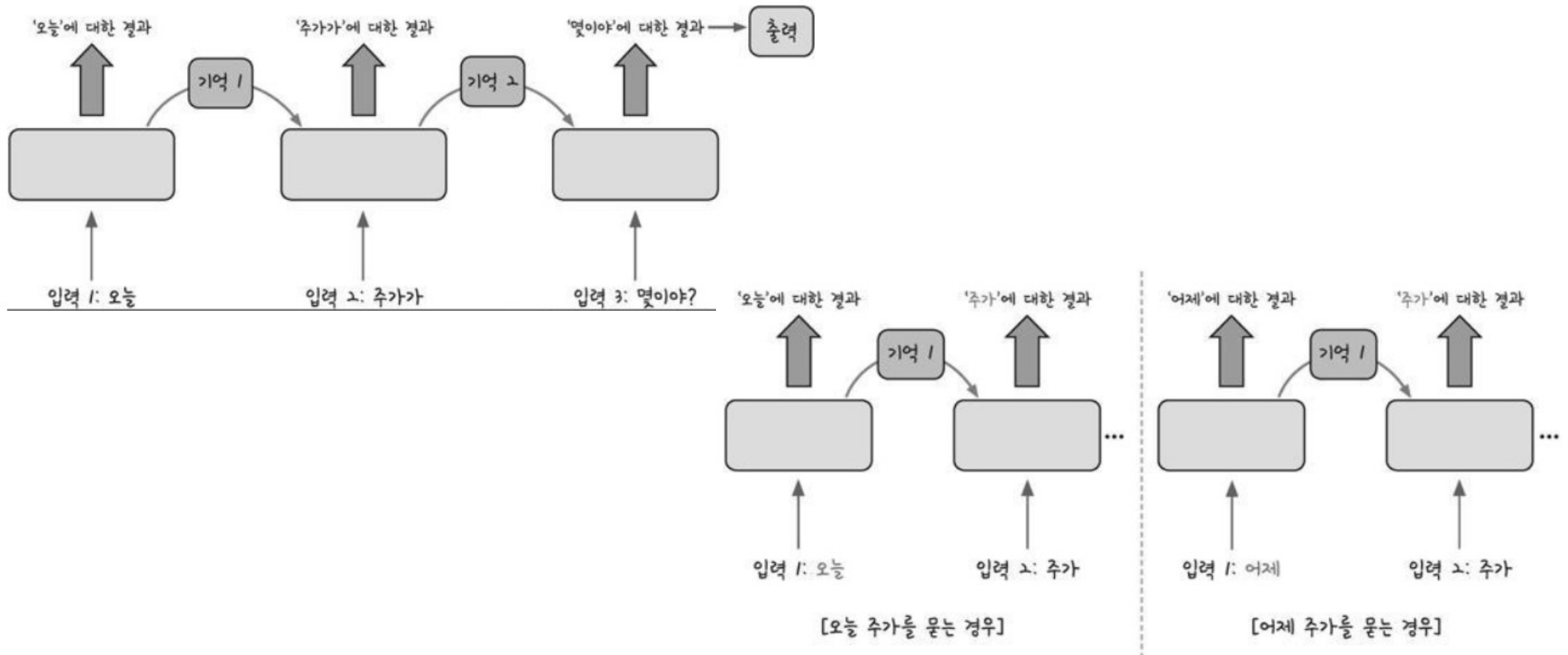
- 인공지능이 문장을 듣고 이해한다는 것은 과거에 입력된 데이터와 나중에 입력된 데이터 사이의 관계를 고려해서 학습하는 것입니다.
- 입력된 문장의 의미를 파악하는 것은 곧 모든 단어를 종합하여 하나의 카테고리로 분류하는 작업입니다.
- 대화형 인공지능 - 문장을 듣고 무엇을 의미하는지를 알아야 서비스를 제공
- 문장 학습 - 여러 개의 단어로 이루어진 문장의 각 단어가 정해진 순서대로 입력되어야 의미를 전달 할 수 있으므로 입력된 데이터 사이의 관계를 고려해야 합니다.
- **순환 신경망(Recurrent Neural Network, RNN)**은 여러 개의 데이터가 순서대로 입력되었을 때 앞서 입력 받은 데이터를 잠시 기억해 놓는 방법입니다.
- 모든 입력 값에 기억된 데이터가 얼마나 중요한지를 판단하여 별도의 가중치를 줘서 다음 층으로 데이터로 넘어갑니다.



일반 신경망과 순환 신경망의 차이

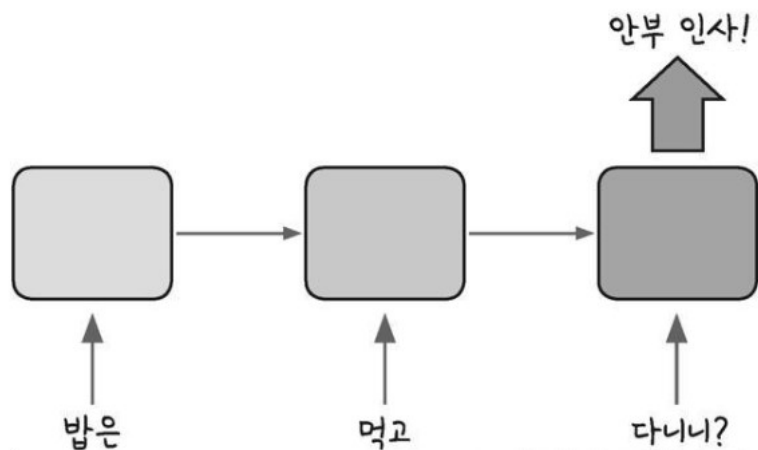
순환 신경망(Recurrent Neural Network, RNN)

- RNN 처리 방식 : 순환 부분에서 단어를 하나 처리할 때마다 단어마다 기억하여 다음 입력 값의 출력을 결정합니다.
- 순환이 되는 가운데 앞서 나온 입력에 대한 결과가 뒤에 나오는 입력 값에 영향을 주는 것을 알 수 있습니다. 이렇게 해야지만, 비슷한 두 문장이 입력되었을 때 그 차이를 구별하여 출력 값에 반영할 수가 있습니다.



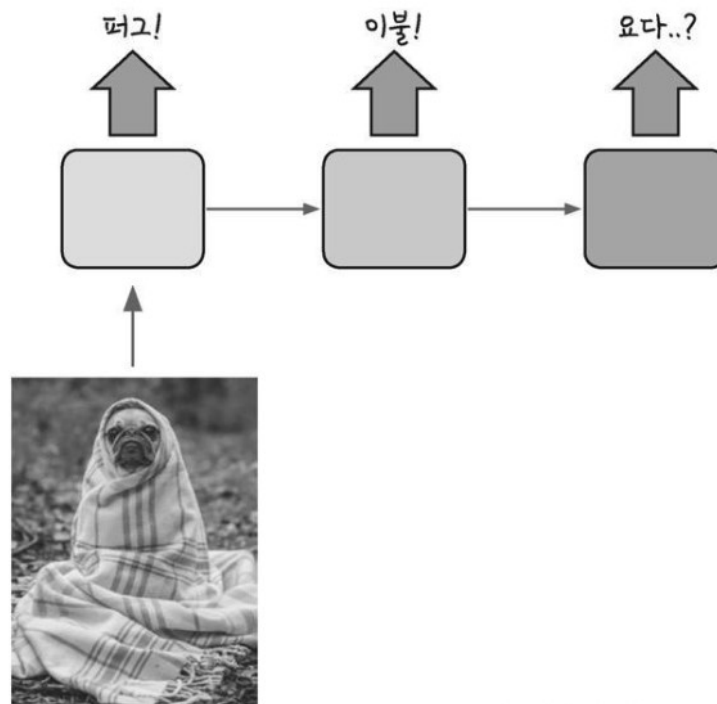
순환 신경망(Recurrent Neural Network, RNN)

- RNN 방식의 장점은 입력 값과 출력 값을 어떻게 설정하느냐에 따라 여러 가지 상황에서 이를 적용할 수 있다는 것입니다.
- RNN의 입력과 출력은 우리가 네트워크에게 시키고 싶은 것이 무엇이냐에 따라 얼마든지 달라질 수 있습니다.



다수 입력 단일 출력(문장을 읽고 뜻을 파악할 때 활용)

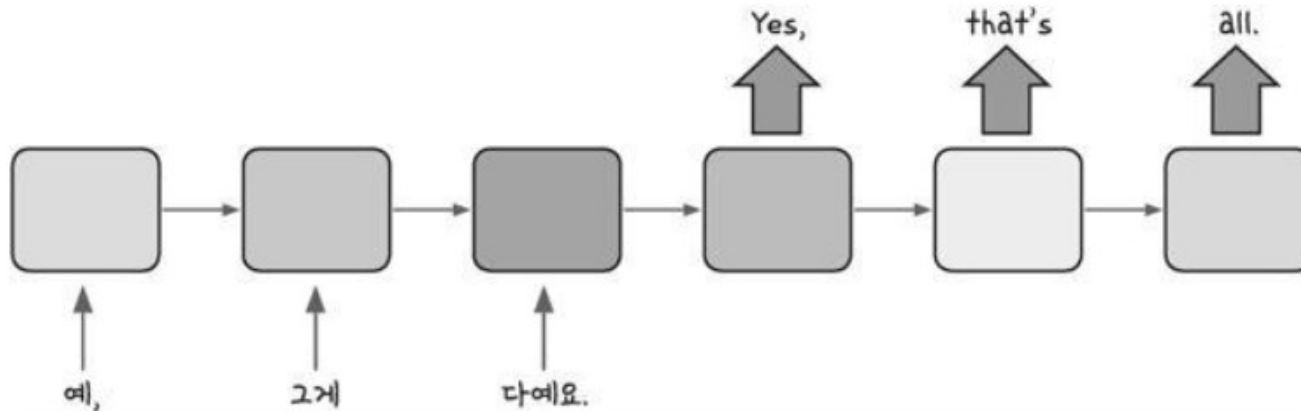
시퀀스 입력 & 고정크기 출력. 예) 문장을 입력해서 긍정 부정 정도를 출력하는 감성 분석기



단일 입력 다수 출력(사진의 캡션을 만들 때 활용)

고정크기 입력 & 시퀀스 출력. 예) 이미지를 입력해서 이
이미지에 대한 설명을 문장으로 출력하는 이미지 캡션 생성

순환 신경망(Recurrent Neural Network, RNN)



다수 입력 다수 출력(문장을 번역할 때 활용)

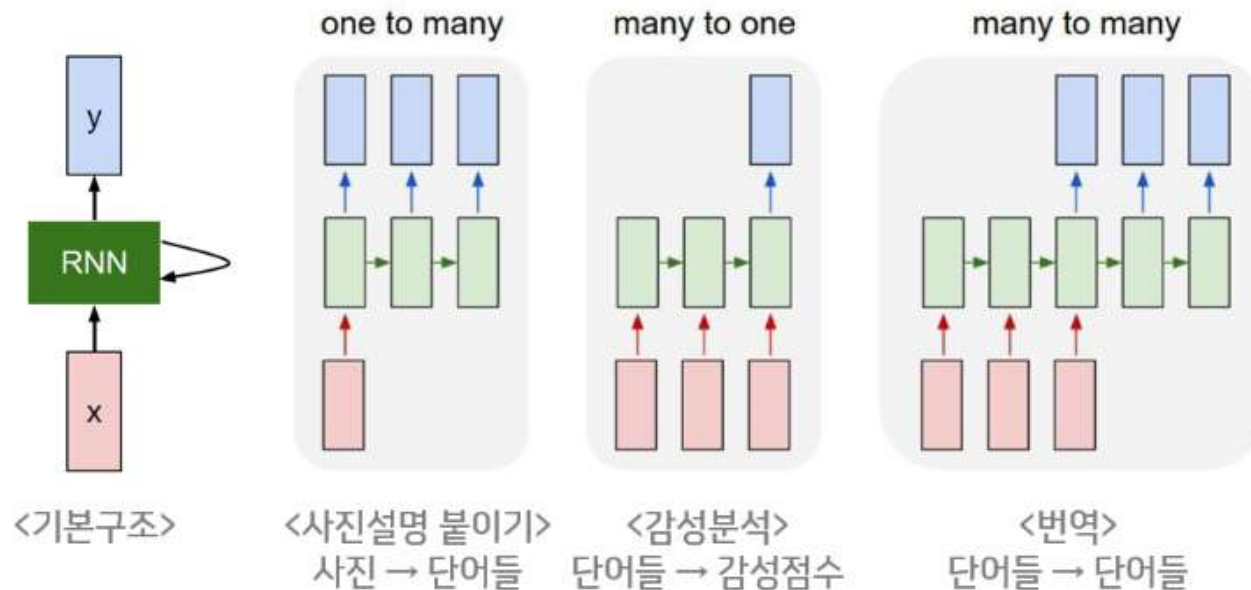
시퀀스 입력 & 시퀀스 출력. 예) 영어를 한국어로 번역하는 자동 번역기

- 동기화된 시퀀스 입력 & 시퀀스 출력. 예) 문장에서 다음에 나올 단어를 예측하는 언어 모델
- 이미지를 처리하는 CNN(Convolutional Neural Network)과 RNN을 결합하여 이미지를 텍스트로 설명해주는 모델을 만드는 것이 가능합니다 (고정크기 입력 & 시퀀스 출력에 해당)
- 구글의 번역기와 네이버의 파파고는 RNN을 응용한 모델로 만들어졌습니다. RNN 기반 모델은 기존 통계 기반 모델의 비해 우수한 성능을 보입니다. (시퀀스 입력 & 시퀀스 출력 구조)

머신러닝


➤ Recurrent Neural Networks(RNN)

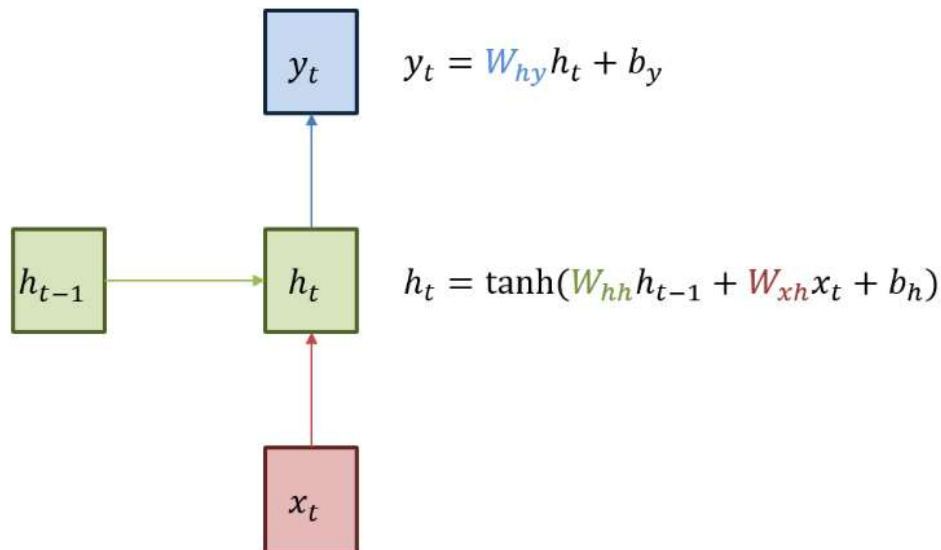
- RNN은 **히든(기억) 노드가 방향을 가진 엣지로 연결돼 순환구조를 이루는(directed cycle)** 인공신경망의 한 종류
- 음성, 문자 등 순차적으로 등장하는 데이터 처리에 적합한 모델
- 시퀀스 길이에 관계없이 입력과 출력을 받아들일 수 있는 네트워크 구조이기 때문에 필요에 따라 다양하고 유연하게 구조를 만들 수 있다
- 네트워크의 기억은 지금까지의 입력 데이터를 요약한 정보라고 볼 수 있습니다. 새로운 입력이 들어 올 때마다 네트워크는 자신의 기억을 조금씩 수정합니다. 결국 입력을 모두 처리하고 난 후 네트워크에게 남겨진 기억은 시퀀스 전체를 요약하는 정보가 됩니다.



머신러닝

➤ Recurrent Neural Networks(RNN)

- 히든 state(RNN 셀), input x , output y
- 현재 상태의 히든 state h_t 는 직전 시점의 히든 state h_{t-1} 를 받아 갱신됩니다.
- 현재 상태의 아웃풋 y_t 는 h_t 를 전달받아 갱신되는 구조
- 히든 state의 활성화함수(activation function)은 비선형 함수인 하이퍼볼릭탄젠트(tanh)
- RNN은 관련 정보와 그 정보를 사용하는 지점 사이 거리가 멀 경우 역전파시 그래디언트가 점차 줄어 학습능력이 크게 저하되는 단점이 있습니다. (vanishing gradient problem) 



RNN(Recurrent Neural Network) :

입력과 출력을 시퀀스 (Sequence) 단위로 처리하는 모델입니다.

피드 포워드 신경망(Feed Forward Neural Network) :

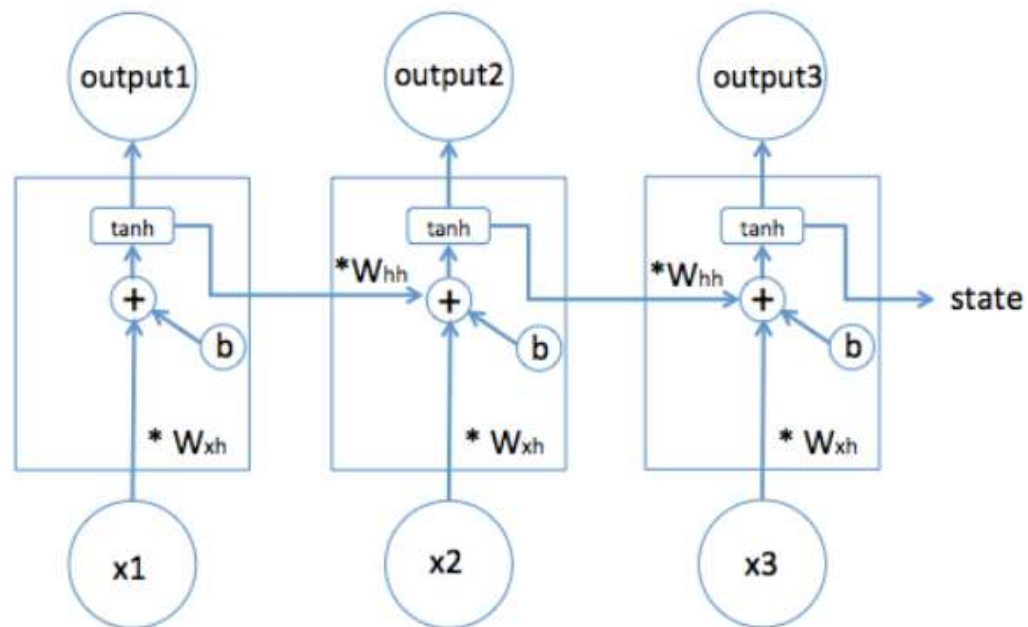
은닉층에서 활성화 함수를 지난 값은 오직 출력층 방향으로만 함함

RNN은 은닉층의 노드에서 활성화 함수를 통해 나온 결과값을 출력층 방향으로도 보내면서, 다시 은닉층 노드의 다음 계산의 입력으로 보내는 특징을 갖고 있습니다.

머신러닝

➤ Recurrent Neural Networks(RNN)

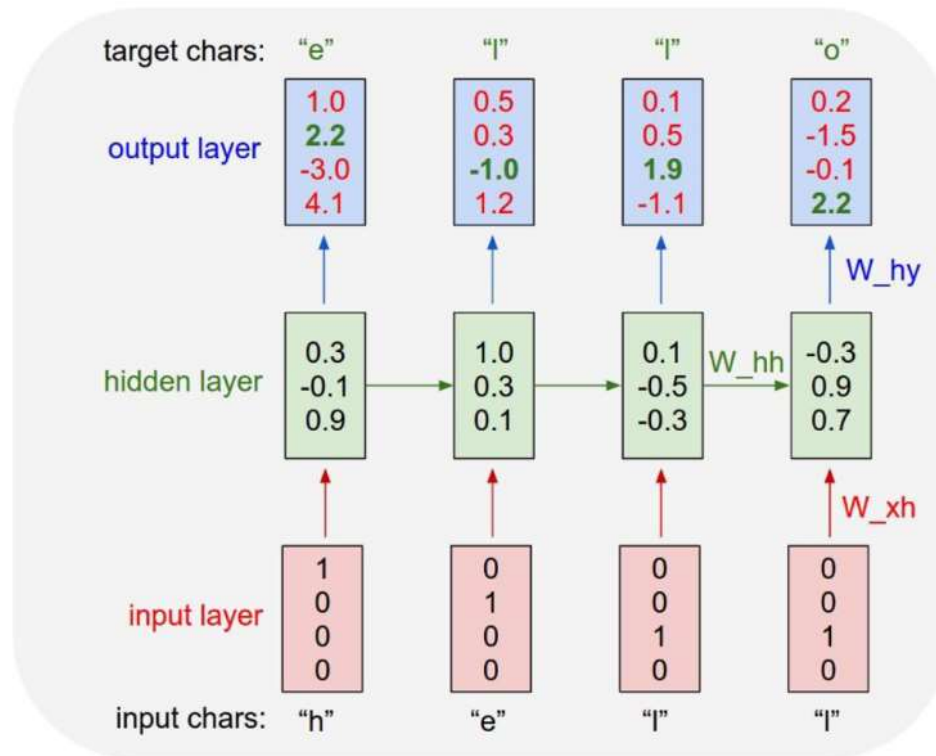
- RNN은 입력값(x), 출력값(output), 상태값(state), 가중치(W), 편향값(b), 그리고 활성화함수(tanh)로 구성



머신러닝

➤ Recurrent Neural Networks(RNN)

- 어떤 글자가 주어졌을 때 바로 다음 글자를 예측하는 character-level-mode
- RNN 모델에 'hell'을 넣으면 'o'를 반환하게 해 결과적으로는 'hello'를 출력
- 학습데이터의 글자는 'h', 'e', 'l', 'o' -> one-hot-vector로 변환 $[1,0,0,0], [0,1,0,0], [0,0,1,0], [0,0,0,1]$

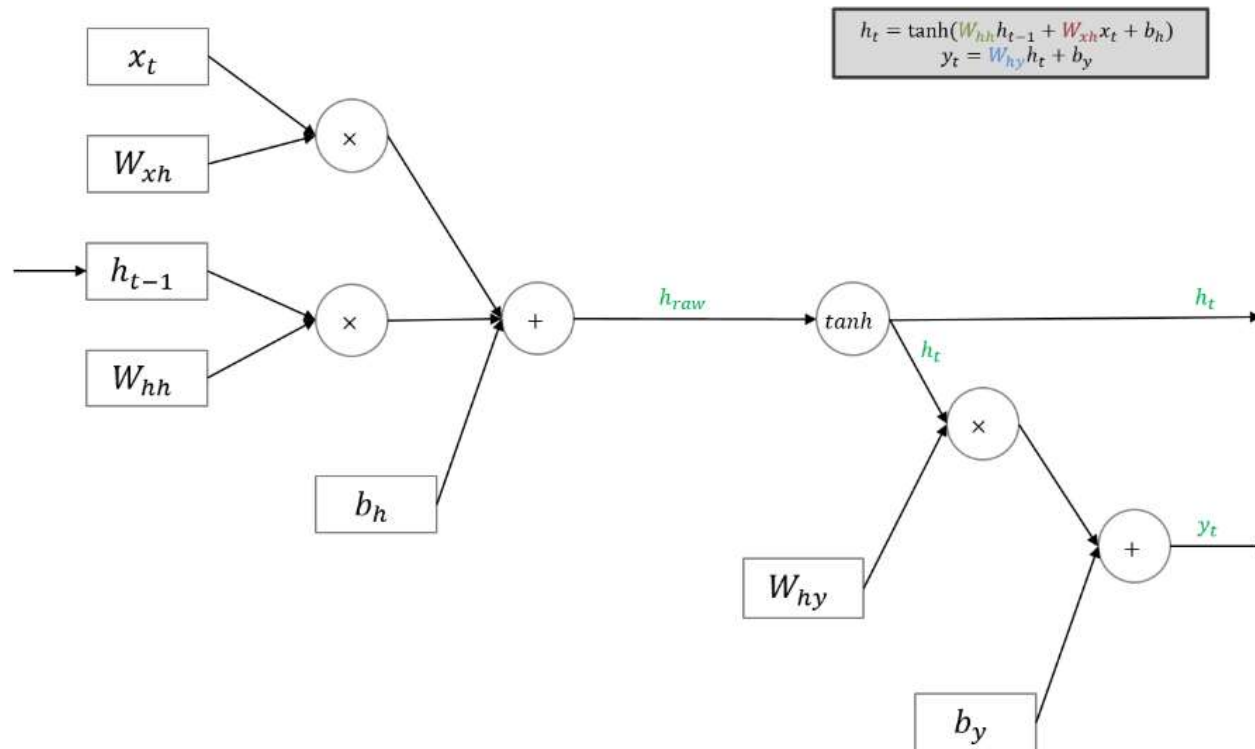


이전 hidden state가 없고
forward propagation 으로 수행

머신러닝

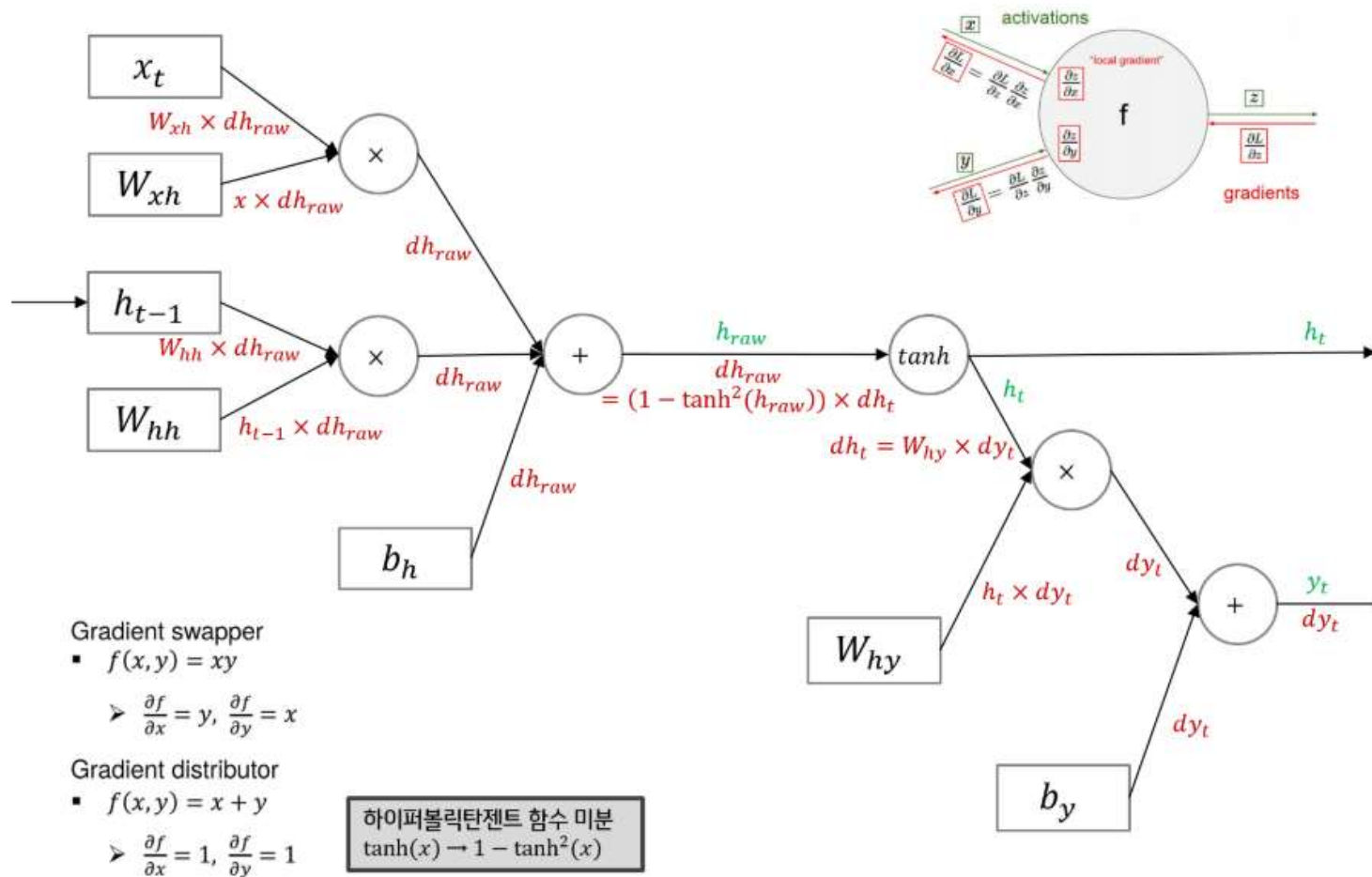
➤ Recurrent Neural Networks(RNN) 순전파

- 인풋 x 를 히든레이어 h 로 보내는 W_{xh} , 이전 히든레이어 h 에서 다음 히든레이어 h 로 보내는 W_{hh} , 히든레이어 h 에서 아웃풋 y 로 보내는 W_{hy} 가 parameter
- 모든 시점의 state에서 이 parameter는 동일하게 적용됩니다(shared weights).



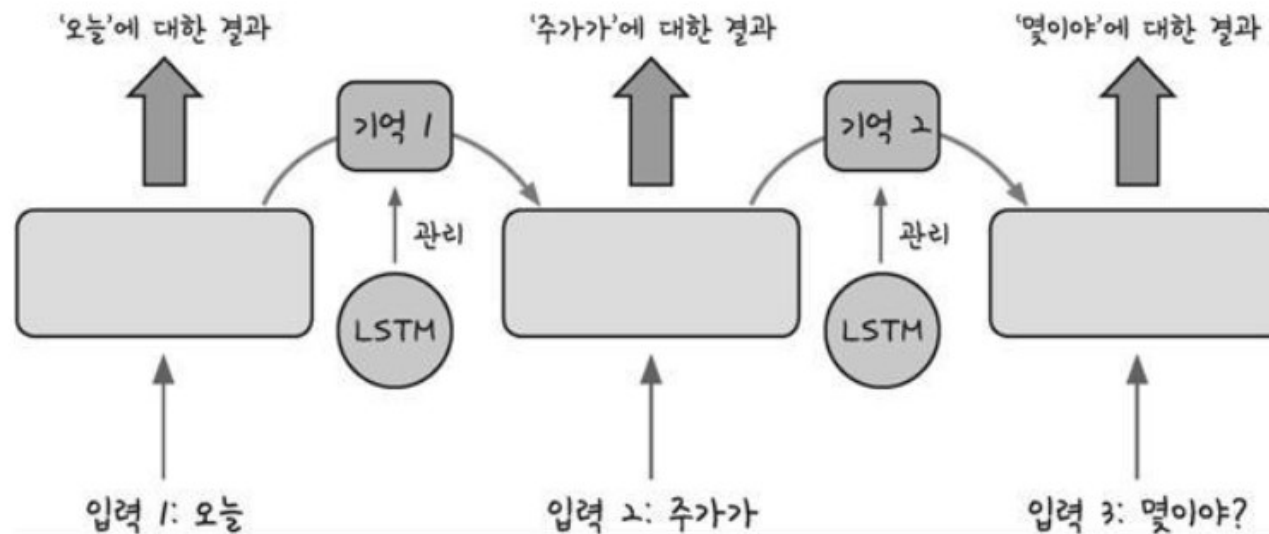
머신러닝

➤ Recurrent Neural Networks(RNN) 역전파



순환 신경망(Recurrent Neural Network, RNN)

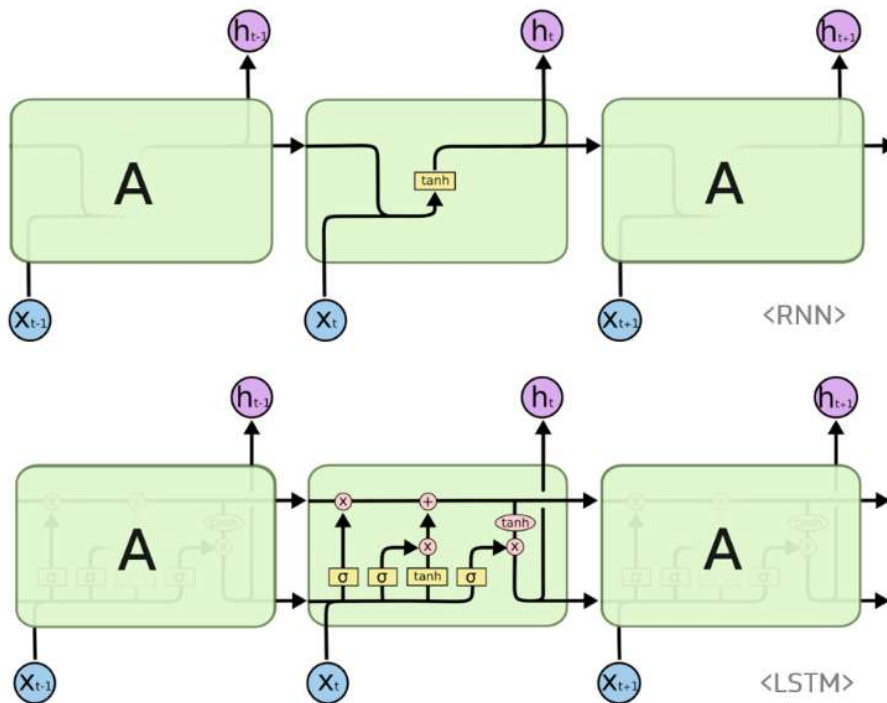
- **LSTM(Long Short Term Memory)** : 한 층 안에서 반복을 많이 해야 하는 RNN의 특성상 일반 신경망보다 기울기 소실 문제가 더 많이 발생하고 이를 해결하기 어렵다는 단점을 보완한 방법입니다.
- LSTM(Long Short Term Memory)은 반복되기 직전에 다음 층으로 기억된 값을 넘길지 안 넘길지를 관리하는 단계를 하나 더 추가하는 것입니다



머신러닝

➤ Long Short-Term Memory models(LSTM)

- LSTM은 RNN의 히든 state에 cell-state를 추가한 구조입니다
- cell state는 컨베이어 벨트 역할을 수행 (state가 꽤 오래 경과하더라도 그래디언트가 비교적 전파가 잘 되게 함)
- \odot 는 요소별 곱셈을 뜻하는 Hadamard product 연산자입니다.

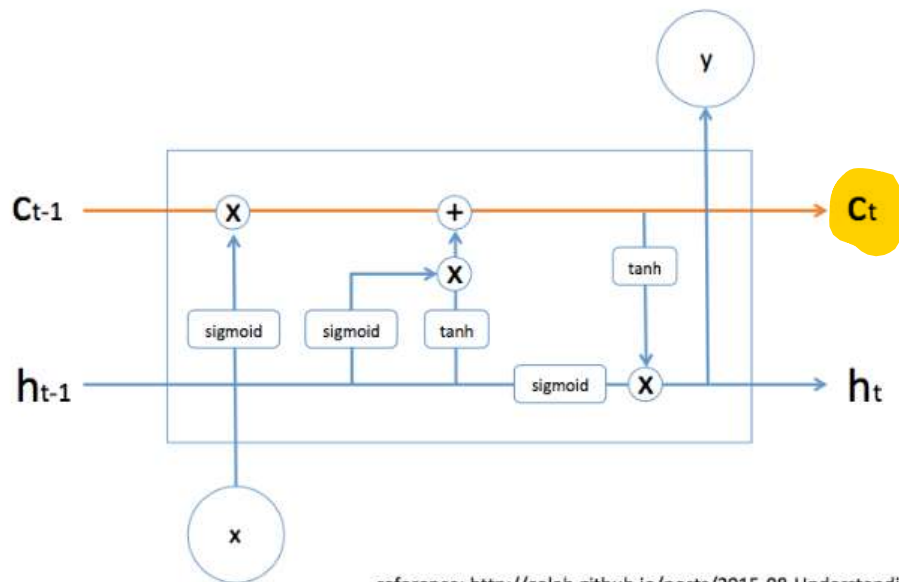


$$\begin{aligned}f_t &= \sigma(W_{xh_f}x_t + W_{hh_f}h_{t-1} + b_{h_f}) \\i_t &= \sigma(W_{xh_i}x_t + W_{hh_i}h_{t-1} + b_{h_i}) \\o_t &= \sigma(W_{xh_o}x_t + W_{hh_o}h_{t-1} + b_{h_o}) \\g_t &= \tanh(W_{xh_g}x_t + W_{hh_g}h_{t-1} + b_{h_g}) \\c_t &= f_t \odot c_{t-1} + i_t \odot g_t \\h_t &= o_t \odot \tanh(c_t)\end{aligned}$$

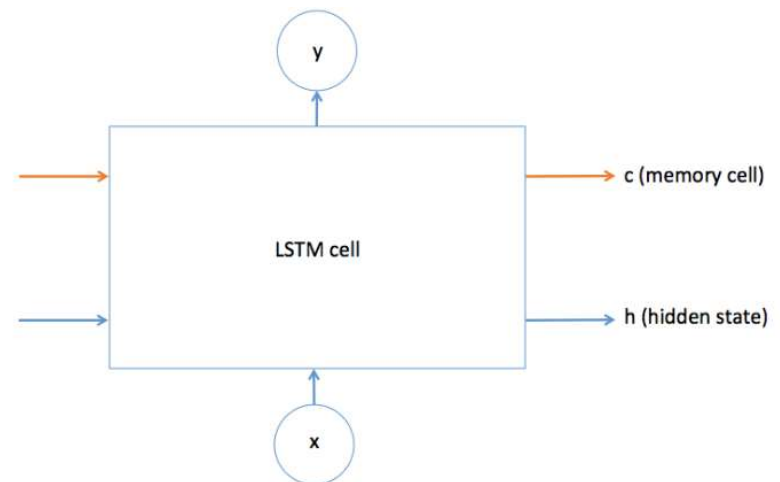
머신러닝

➤ Long Short-Term Memory models(LSTM)

- LSTM은 gradient vanishing 또는 gradient exploding과 같은 기존 RNN의 단점을 극복하고자 만들어진 조금 더 진화된 RNN 셀입니다.
- LSTM 셀 내부는 **이전 정보를 지우거나 기억하기 위한 로직**과 **현재 정보를 기억하기 위한 로직**이 구현되어 있습니다.
- 주황색 선은 메모리 셀이라고 부릅니다. 주황색 선상의 곱하기 기호에서, 0부터 1까지의 값인 시그모이드 출력값이 곱해지게 되어, 메모리 셀의 기존 정보를 어느정도까지 기억할 지 결정하게 됩니다.
- 주황색 선상의 더하기 기호는 새로운 정보를 메모리셀의 기존 정보에 더하는 로직입니다.
- h_t 선상의 곱하기 기호에서 메모리셀의 정보와 현재 정보가 함께 계산되어 상태값을 출력



reference: <http://colah.github.io/posts/2015-08-Understanding-LSTMs/>



머신러닝

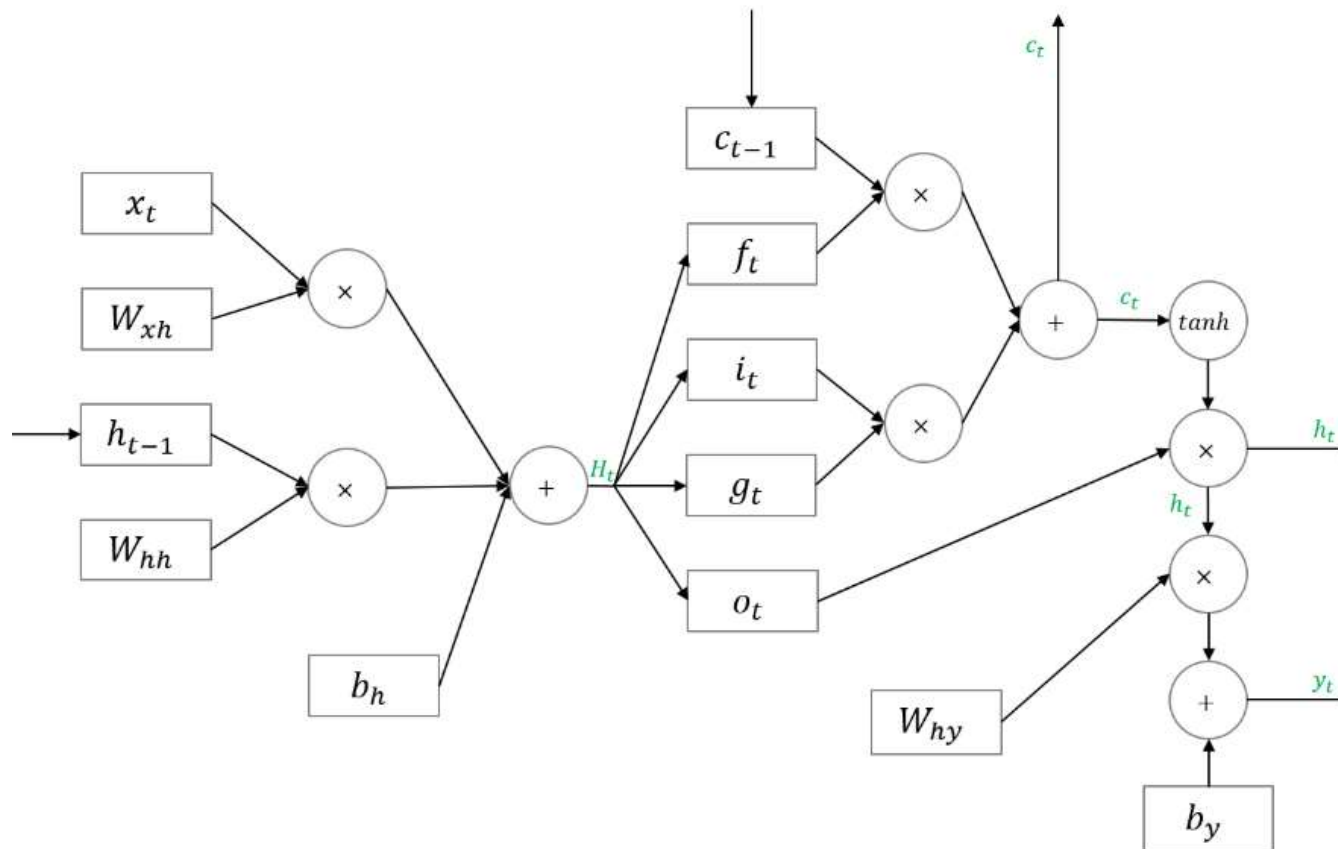
➤ Long Short-Term Memory models(LSTM)

- forget gate f_t 는 '과거 정보를 잊기'를 위한 게이트입니다. h_{t-1} 과 x_t 를 받아 시그모이드를 취해준 값이 바로 forget gate가 내보내는 값이 됩니다. 시그모이드 함수의 출력 범위는 0에서 1 사이이기 때문에 그 값이 0이라면 이전 상태의 정보는 잊고, 1이라면 이전 상태의 정보를 온전히 기억하게 됩니다.
- input gate $i_t \odot g_t$ 는 '현재 정보를 기억하기' 위한 게이트입니다. h_{t-1} 과 x_t 를 받아 시그모이드를 취하고, 또 같은 입력으로 하이퍼볼릭탄젠트를 취해준 다음 Hadamard product 연산을 한 값이 바로 input gate가 내보내는 값이 됩니다.
- i_t 의 범위는 0~1, g_t 의 범위는 -1~1이기 때문에 각각 강도와 방향을 나타냅니다

머신러닝

➤ Long Short-Term Memory models(LSTM)의 순전파

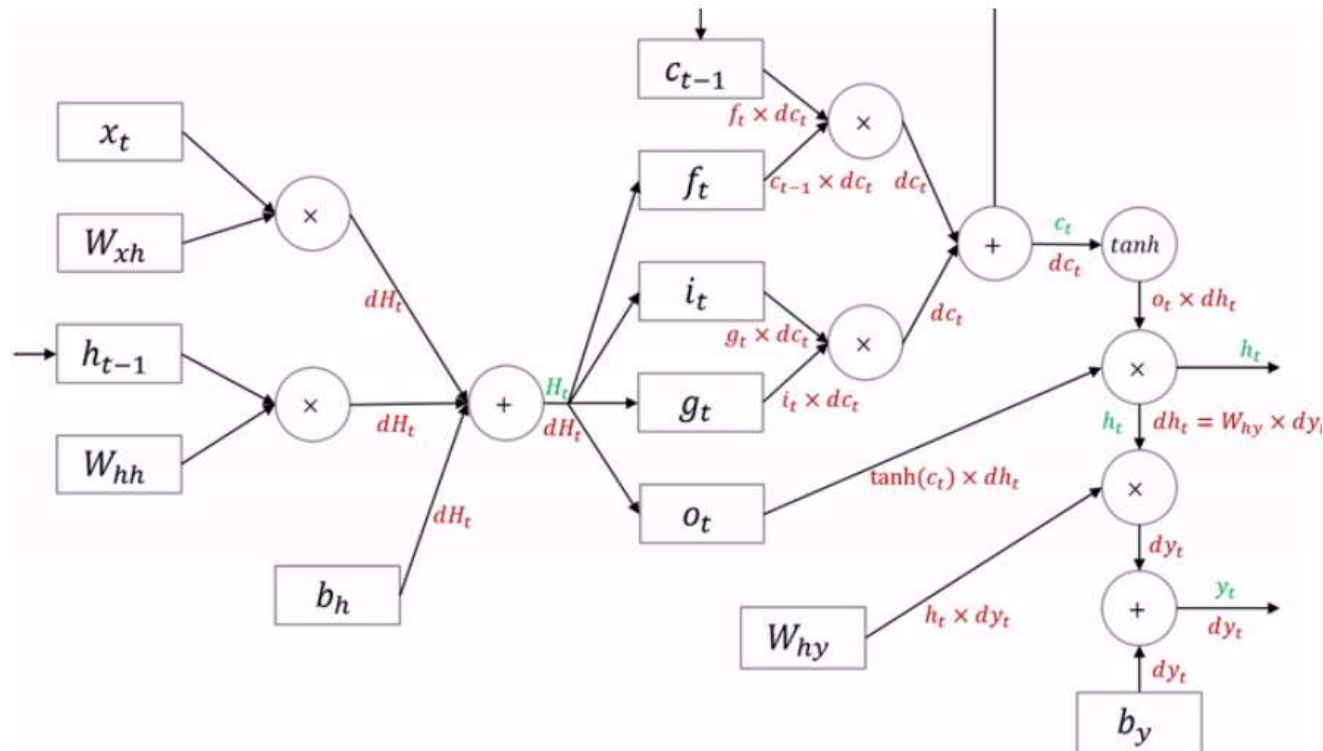
- 행 기준으로 4등분해 i, f, o, g 각각에 해당하는 활성화함수를 적용하는 방식으로 i, f, o, g 를 계산합니다.



머신러닝

➤ Long Short-Term Memory models(LSTM)의 역전파

- dH_t 를 구하는 과정
- H_t 는 i_t, f_t, o_t, g_t 로 구성된 행렬입니다.
- 각각에 해당하는 그래디언트를 이를 합치면(merge) dH_t 를 만들 수 있다
- i, f, o 의 활성화함수는 시그모이드이고, g 만 하이퍼볼릭탄젠트입니다.
- 각각의 활성화함수에 대한 로컬 그래디언트를 구해 흘러들어온 그래디언트를 곱해주면 됩니다..



순환 신경망(Recurrent Neural Network, RNN)

➤ LSTM을 이용한 로이터 뉴스 카테고리 분류

- 입력된 문장의 의미를 파악하는 것은 곧 모든 단어를 종합하여 하나의 카테고리로 분류하는 작업이라고 할 수 있습니다.

중부 지방은 대체로 맑겠으나, 남부 지방은 구름이 많겠습니다. → 날씨
올 초부터 유동성의 힘으로 주가가 일정하게 상승했습니다. → 주식
이번 선거에서는 누가 이길 것 같아? → 정치
퍼셉트론의 한계를 극복한 신경망이 다시 뜨고 있대. → 딥러닝

- 로이터 뉴스 데이터 : 총 11,258개의 뉴스 기사가 46개의 카테고리로 나누어진 대용량 텍스트 데이터
- 딥러닝은 단어를 그대로 사용하지 않고 숫자로 변환한 다음 학습할 수 있습니다.
- 데이터 안에서 단어별 빈도에 따라 순서 번호를 붙였습니다.
- Embedding('불러온 단어의 총 개수', '기사당 단어 수') 층은 데이터 전처리 과정을 통해 입력된 값을 받아 다음 층이 알아들을 수 있는 형태로 변환하는 역할을 합니다.
- Embedding은 모델 설정 부분의 맨 처음에 있어야 합니다.
- LSTM(기사당 단어 수, 기타 옵션)은 RNN에서 기억 값에 대한 가중치를 제어합니다.
- LSTM의 활성화 함수로는 Tanh를 사용합니다.

순환 신경망(Recurrent Neural Network, RNN)

➤ LSTM을 이용한 로이터 뉴스 카테고리 분류

- Embedding 층은 데이터 전처리 과정을 통해 입력된 값을 받아 다음 층이 알아들을 수 있는 형태로 변환하는 역할을 합니다.
- Embedding('불러온 단어의 총 개수', '기사당 단어 수') 형식으로 사용하며, 모델 설정 부분의 맨 처음에 있어야 합니다. LSTM은 RNN에서 기억 값에 대한 가중치를 제어합니다.
- LSTM(기사당 단어 수, 기타 옵션)의 형태로 적용됩니다.
- LSTM의 활성화 함수로는 Tanh를 사용합니다.

```
from keras.datasets import reuters
from keras.models import Sequential
from keras.layers import Dense, LSTM, Embedding
from keras.preprocessing import sequence
from keras.utils import np_utils
import numpy
import tensorflow as tf
import matplotlib.pyplot as plt

seed = 0    # seed 값 설정
numpy.random.seed(seed)
tf.set_random_seed(seed)

# 불러온 데이터를 학습셋과 테스트셋으로 나누기
(X_train, Y_train), (X_test, Y_test) = reuters.load_data(num_words=1000, test_split=0.2)
```

순환 신경망(Recurrent Neural Network, RNN)

- LSTM을 이용한 로이터 뉴스 카테고리 분류

```
# 데이터 확인하기
category = numpy.max(Y_train) + 1
print(category, '카테고리')
print(len(X_train), '학습용 뉴스 기사')
print(len(X_test), '테스트용 뉴스 기사')
print(X_train[0])
# 데이터 전처리
x_train = sequence.pad_sequences(X_train, maxlen=100)
x_test = sequence.pad_sequences(X_test, maxlen=100)
y_train = np_utils.to_categorical(Y_train)
y_test = np_utils.to_categorical(Y_test)

model = Sequential() # 모델의 설정
model.add(Embedding(1000, 100))
model.add(LSTM(100, activation='tanh'))
model.add(Dense(46, activation='softmax'))
# 모델의 컴파일
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

순환 신경망(Recurrent Neural Network, RNN)

➤ LSTM을 이용한 로이터 뉴스 카테고리 분류

```
# 모델의 실행
history = model.fit(x_train, y_train, batch_size=100, epochs=20, validation_data=(x_test, y_test))
# 테스트 정확도 출력
print("\n Test Accuracy: %.4f" % (model.evaluate(x_test, y_test)[1]))
# 테스트셋의 오차
y_vloss = history.history['val_loss']

# 학습셋의 오차
y_loss = history.history['loss']

# 그래프로 표현
x_len = numpy.arange(len(y_loss))
plt.plot(x_len, y_vloss, marker='.', c="red", label='Testset_loss')
plt.plot(x_len, y_loss, marker='.', c="blue", label='Trainset_loss')

# 그래프에 그리드를 주고 레이블을 표시
plt.legend(loc='upper right')
plt.grid()
plt.xlabel('epoch')
plt.ylabel('loss')
plt.show()
```

순환 신경망(Recurrent Neural Network, RNN)

➤ LSTM과 CNN의 조합을 이용한 영화 리뷰 분류

- 인터넷 영화 데이터베이스(Internet Movie Database, IMDB)는 영화와 관련된 정보와 출연진 정보, 개봉 정보, 영화 후기, 평점에 이르기까지 매우 폭넓은 데이터가 저장된 자료입니다.
- 영화에 관해 남긴 2만 5000여 개의 영화 리뷰가 담겨 있으며, 해당 영화를 긍정적으로 평가했는지 혹은 부정적으로 평가했는지도 담겨 있습니다.
- 데이터셋에서 나타나는 빈도에 따라 번호가 정해지므로 빈도가 높은 데이터를 불러와 학습
- Conv2D와 MaxPooling2

```
from keras.preprocessing import sequence
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation
from keras.layers import Embedding
from keras.layers import LSTM
from keras.layers import Conv1D, MaxPooling1D
from keras.datasets import imdb
import numpy
import tensorflow as tf
import matplotlib.pyplot as plt

seed = 0
numpy.random.seed(seed) # seed 값 설정
tf.set_random_seed(seed)
```

순환 신경망(Recurrent Neural Network, RNN)

- LSTM과 CNN의 조합을 이용한 영화 리뷰 분류

```
# 학습셋과 테스트셋 지정하기
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=5000)
x_train = sequence.pad_sequences(x_train, maxlen=100)
x_test = sequence.pad_sequences(x_test, maxlen=100)
model = Sequential()
model.add(Embedding(5000, 100))
model.add(Dropout(0.5))
model.add(Conv1D(64, 5, padding='valid', activation='relu', strides=1))
model.add(MaxPooling1D(pool_size=4))
model.add(LSTM(55))
model.add(Dense(1))
model.add(Activation('sigmoid'))
model.summary()

# 데이터 전처리
# 모델의 설정
# 모델의 컴파일
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
# 모델의 실행
history = model.fit(x_train, y_train, batch_size=100, epochs=5, validation_data=(x_test, y_test))
# 테스트 정확도 출력
print("\n Test Accuracy: %.4f" % (model.evaluate(x_test, y_test)[1]))
```

순환 신경망(Recurrent Neural Network, RNN)

- LSTM과 CNN의 조합을 이용한 영화 리뷰 분류

```
y_vloss = history.history['val_loss']      # 테스트셋의 오차
y_loss = history.history['loss']           # 학습셋의 오차
x_len = numpy.arange(len(y_loss))
plt.plot(x_len, y_vloss, marker='.', c="red", label='Testset_loss')
plt.plot(x_len, y_loss, marker='.', c="blue", label='Trainset_loss')

# 그래프에 그리드를 주고 레이블을 표시
plt.legend(loc='upper right')
plt.grid()
plt.xlabel('epoch')
plt.ylabel('loss')
plt.show()
```