

RNN & LSTM



발표 4조

엄성원 이예령
임형수 차용진

RNN 개념



RNN
Recurrent Neural Network



[순환되는, 반복되는]

-> 이전 시점의 정보를 담고 있는 Weight 값의 반복

시퀀스 데이터



시퀀스 데이터
시간이나 순서에 따라 변화하는 데이터

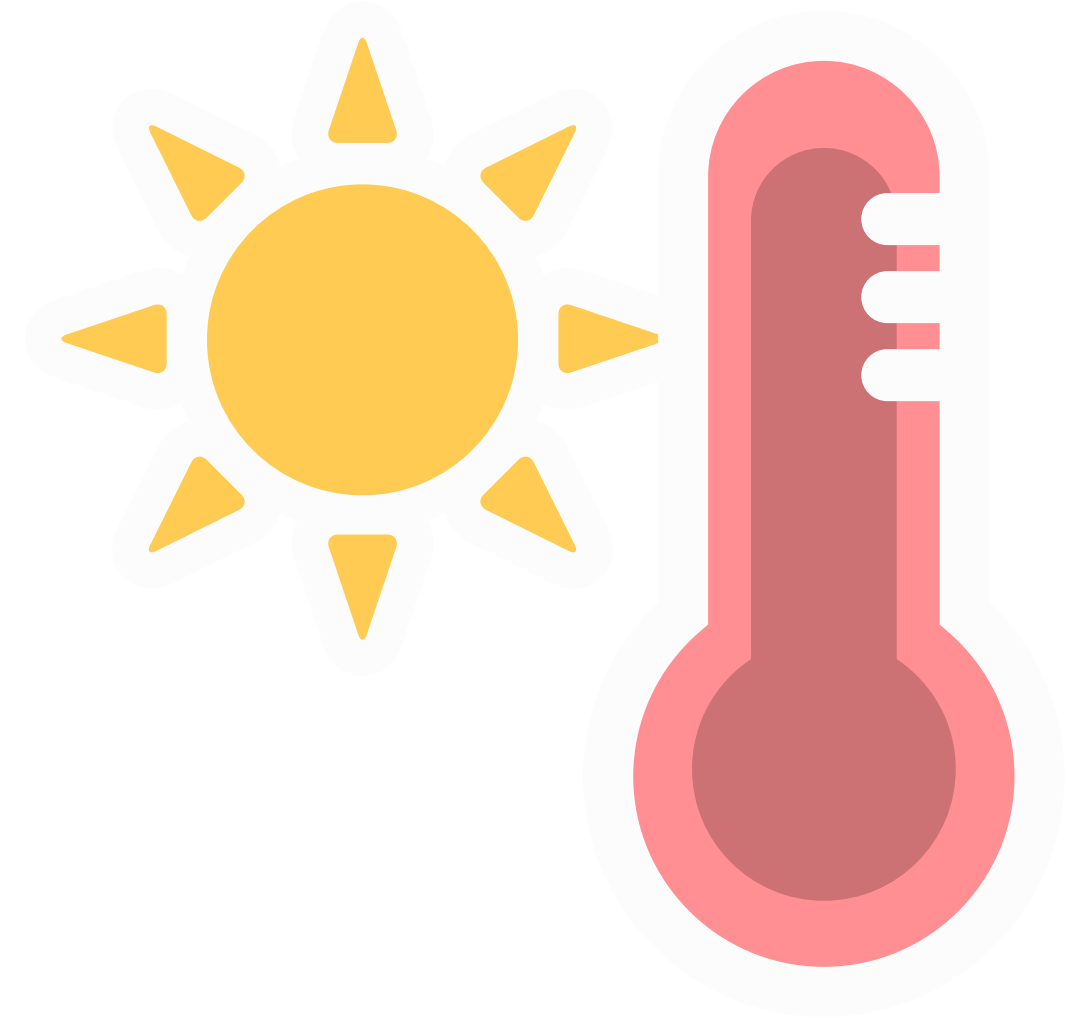
사용 예시: 텍스트, 음성, 주가
사용 분야: 자연어 처리, 음성 인식, 시계열 데이터 분석

이전 입력값을 메모리에 저장,
현재의 예측에 반영

순환 구조

순환

- 네트워크가 이전 상태를 기억하여 현재 예측을 수행할 수 있게 함
- 순서가 중요한 데이터에서 앞의 정보가 뒤의 정보에 미치는 영향을 이해하고 활용할 수 있음.



시간 순서에 따라 이전 데이터를 기억하고,
미래 데이터를 예측할 때 사용

RNN 제한점

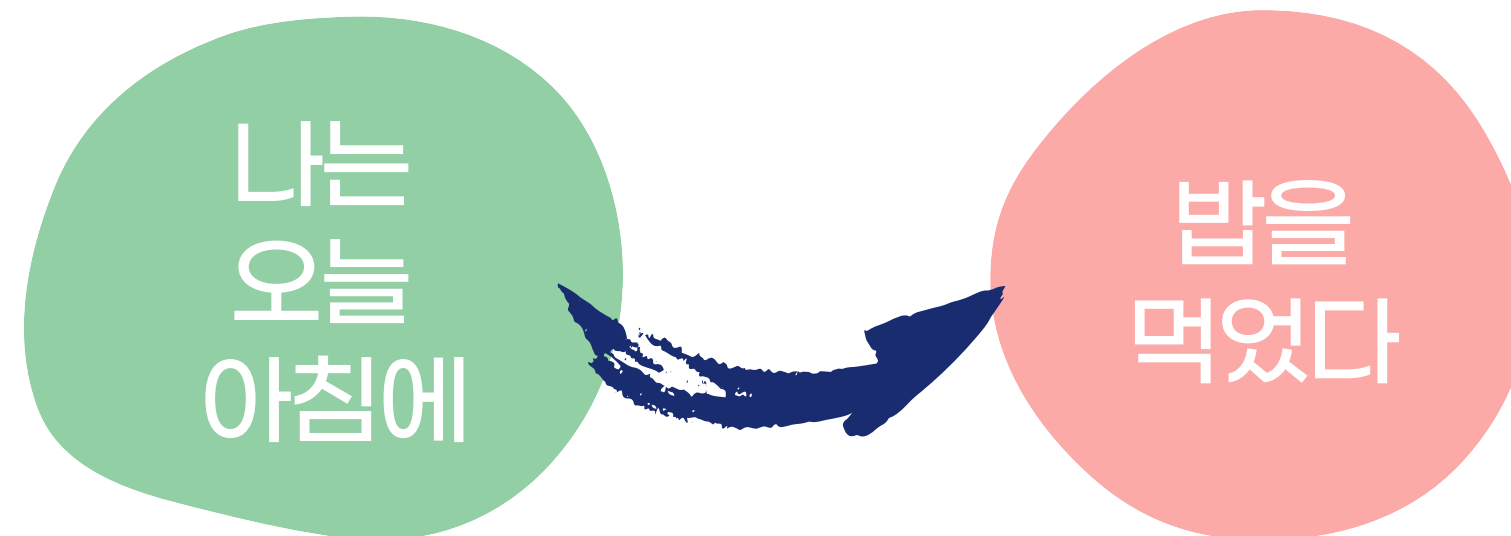


Quiz 1

초기 단계에서 학습된 정보가
후반부로 갈수록 제대로 전달되지 않는 한계점

언어 모델

주어진 문맥에서 다음 단어를 예측하거나,
문장을 생성하기 위해서 사용



이전 단어들을 기억하고, 다음에 올 단어를 예측 가능
-> 순서가 중요한 데이터 처리에 사용

RNN 사용 분야



텍스트 분류

감정 분석

언어 번역

RNN vs Transformer

RNN

- 시퀀스 데이터를 순차적으로 처리, Vanishing Gradient의 문제 발생
- 시퀀스가 너무 길어지면 앞의 중요한 정보가 뒤까지 충분히 전달되지 않는 상황 초래

Transformer

- 데이터를 순차적으로 처리하지 않고, 병렬적으로 처리
- 모든 단어가 서로 어떤 관계를 가지는지 파악하는 Quiz 2 매커니즘

RNN 구현 예제



```
import torch
import torch.nn as nn
import torch.optim as optim
import numpy as np

# RNN 모델 정의
class RNNModel(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(RNNModel, self).__init__()
        self.rnn = nn.RNN(input_size=input_size, hidden_size=hidden_size, batch_first=True)
        self.fc = nn.Linear(hidden_size, output_size)

    def forward(self, x):
        out, _ = self.rnn(x)
        out = self.fc(out[:, -1, :]) # 마지막 시점의 출력을 사용
        return out
```

RNN 구현 예제



```
# 모델, 손실 함수, 최적화 기법 설정
model = RNNModel(input_size, hidden_size, output_size)
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=learning_rate)

# 데이터 준비 (예제 데이터, 실제로는 데이터셋을 사용)
X_train = np.random.random((100, 10, 1)).astype(np.float32)
y_train = np.random.random((100, 1)).astype(np.float32)

# 데이터를 PyTorch 텐서로 변환
X_train_tensor = torch.tensor(X_train)
y_train_tensor = torch.tensor(y_train)
```

RNN 구현 예제



```
# 모델 학습
for epoch in range(num_epochs):
    model.train()

    # Forward pass
    outputs = model(X_train_tensor)
    loss = criterion(outputs, y_train_tensor)

    # Backward and optimize
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    if (epoch+1) % 1 == 0:
        print(f'Epoch [{epoch+1}/{num_epochs}], Loss: {loss.item()}')
```

RNN 구현 예제



```
# 평가 데이터 준비
X_test = np.random.random((10, 10, 1)).astype(np.float32)
y_test = np.random.random((10, 1)).astype(np.float32)

# 데이터를 PyTorch 텐서로 변환
X_test_tensor = torch.tensor(X_test)
y_test_tensor = torch.tensor(y_test)

# 모델 평가
model.eval()
with torch.no_grad():
    test_outputs = model(X_test_tensor)
    test_loss = criterion(test_outputs, y_test_tensor)
    print(f'Test Loss: {test_loss.item()}')
```

LSTM 개념

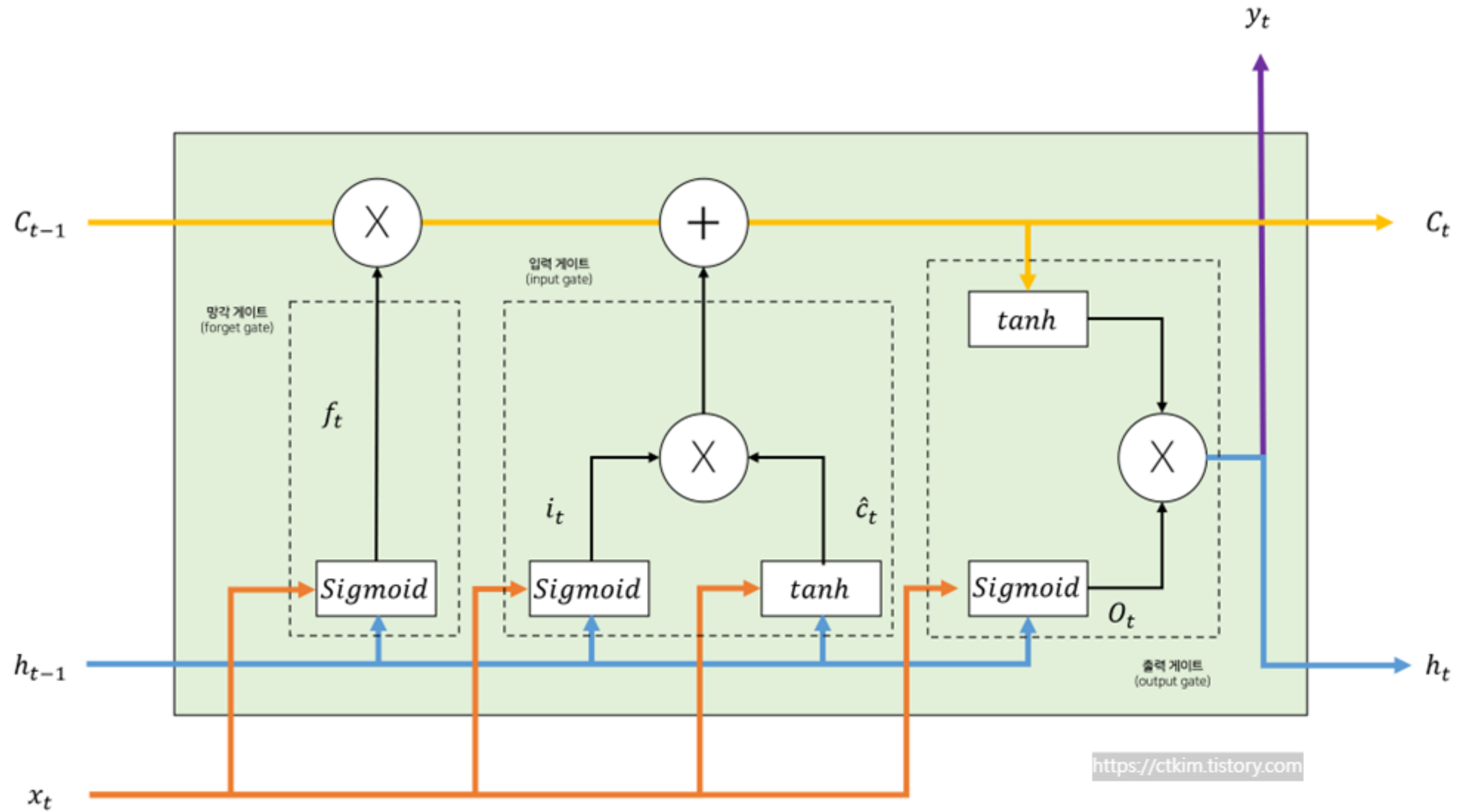


입력 데이터의 길이가 길어지면
Vanishing Gradient 문제 발생

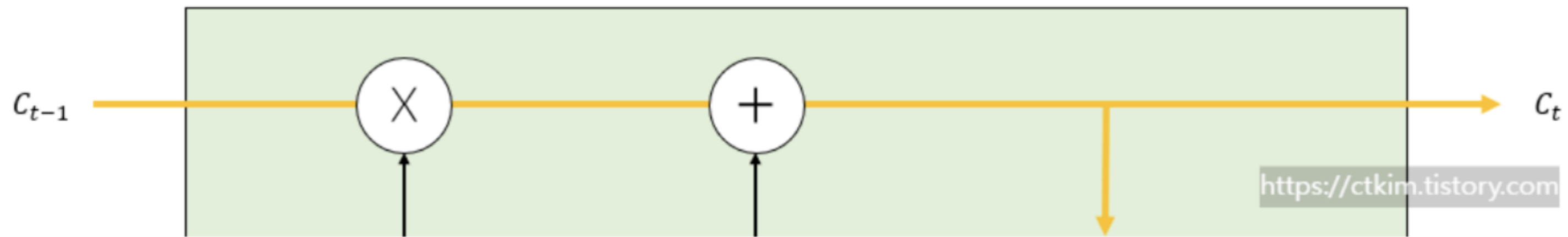


이전 정보를 오랫동안 기억할 수 있는
메모리셀을 통해 긴 시퀀스 데이터 처리

LSTM 구조



LSTM 구조



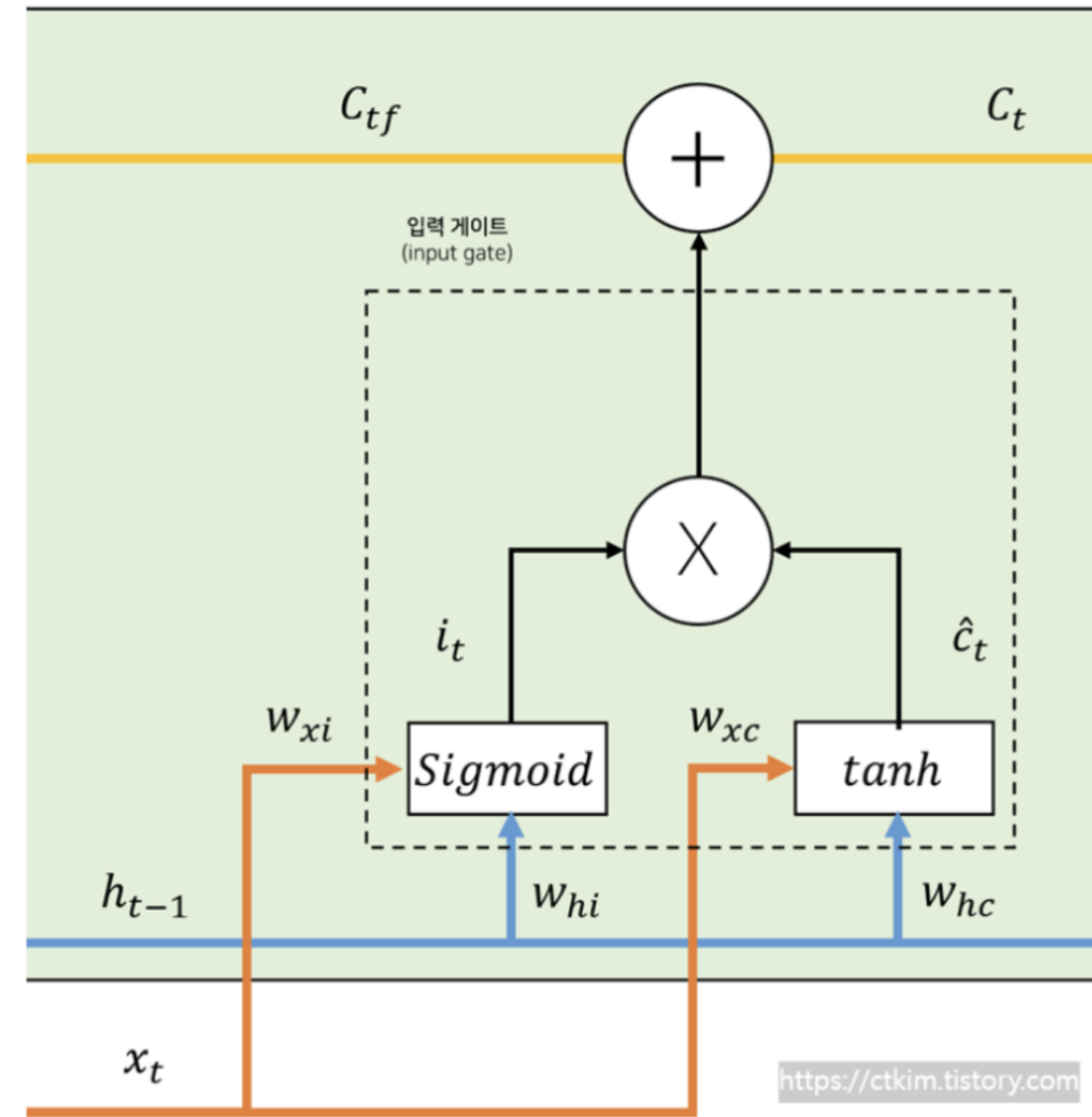
- 이전 상태에서 현재 상태까지 유지되는 정보의 흐름을 나타냄
- 오래된 정보를 기억하고 새로운 정보를 적절하게 갱신

LSTM 구조

$$i_t = \sigma(W_{hi}h_{t-1} + W_{xi}x_t)$$

$$\hat{c}_t = \tanh(W_{hc}h_{t-1} + W_{xc}x_t)$$

$$C_t = C_{tf} \oplus i_t \otimes \hat{c}_t$$

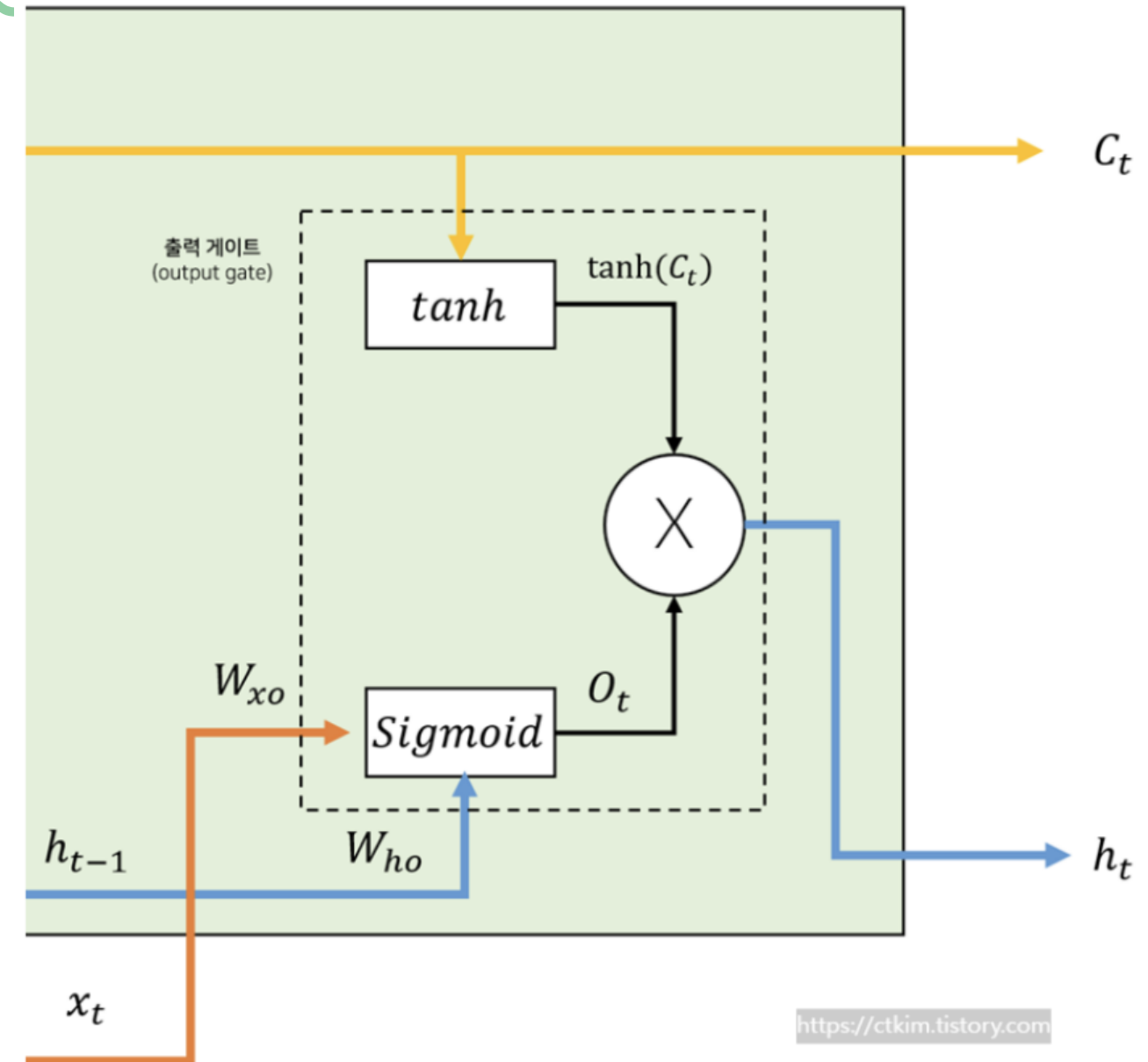


- 새로운 정보를 어떻게 반영할 것인가에 대한 결정

LSTM 구조

$$O_t = \sigma(W_{ho}h_{t-1} + W_{xo}x_t)$$

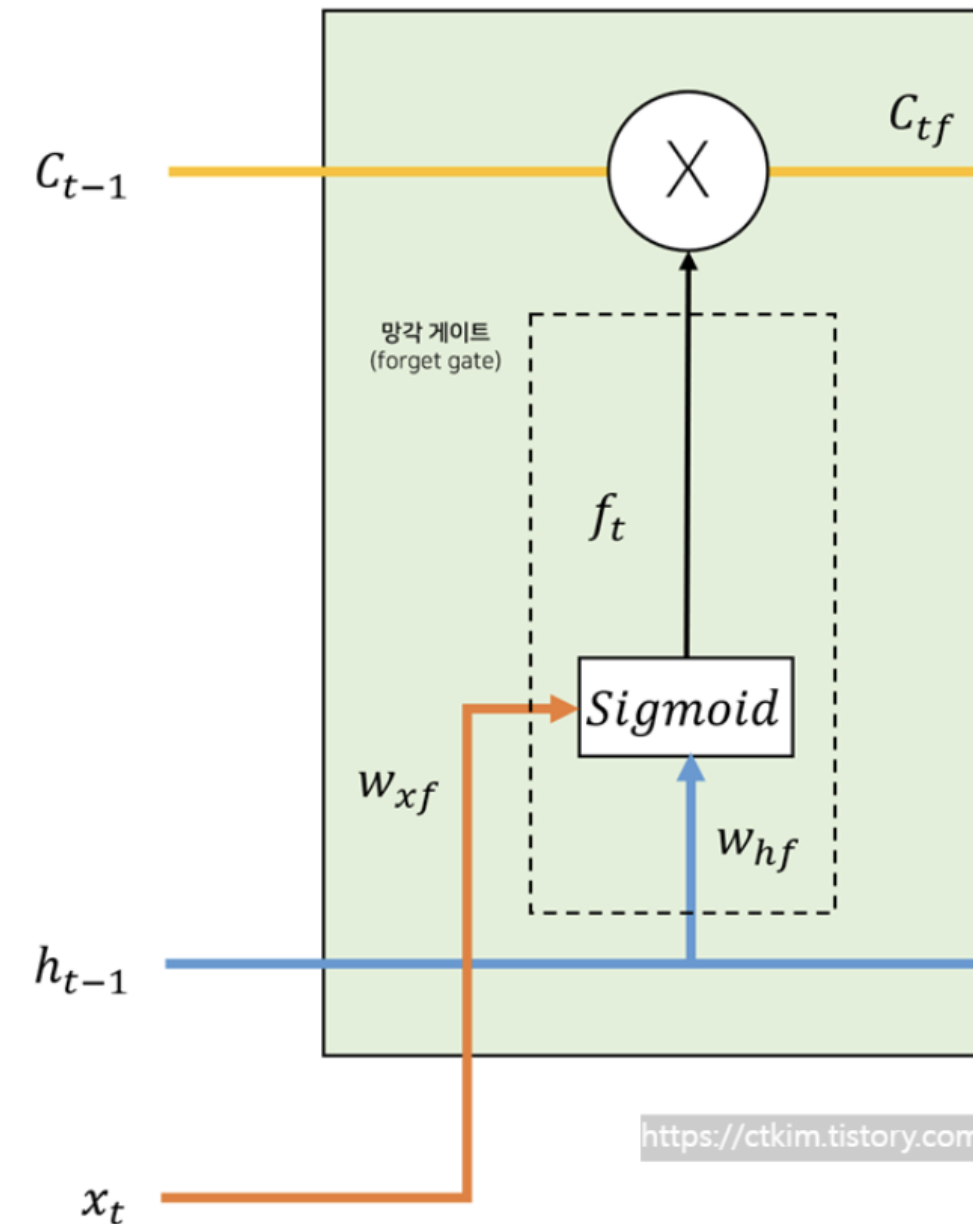
$$h_t = O_t \otimes \tanh(c_t)$$



- 셀 상태 값에 탄젠트 함수를 적용한 값을 출력으로 사용

LSTM 구조

$$f_t = \sigma(W_{xf}h_{t-1} + W_{xf}x_t)$$



- 이전 셀 상태의 정보를 지울 것인지 말 것인지를 결정

LSTM 언어모델



문장 내에서 장기적인 문맥 정보 유지 &
다음에 나올 단어 예측에 적합

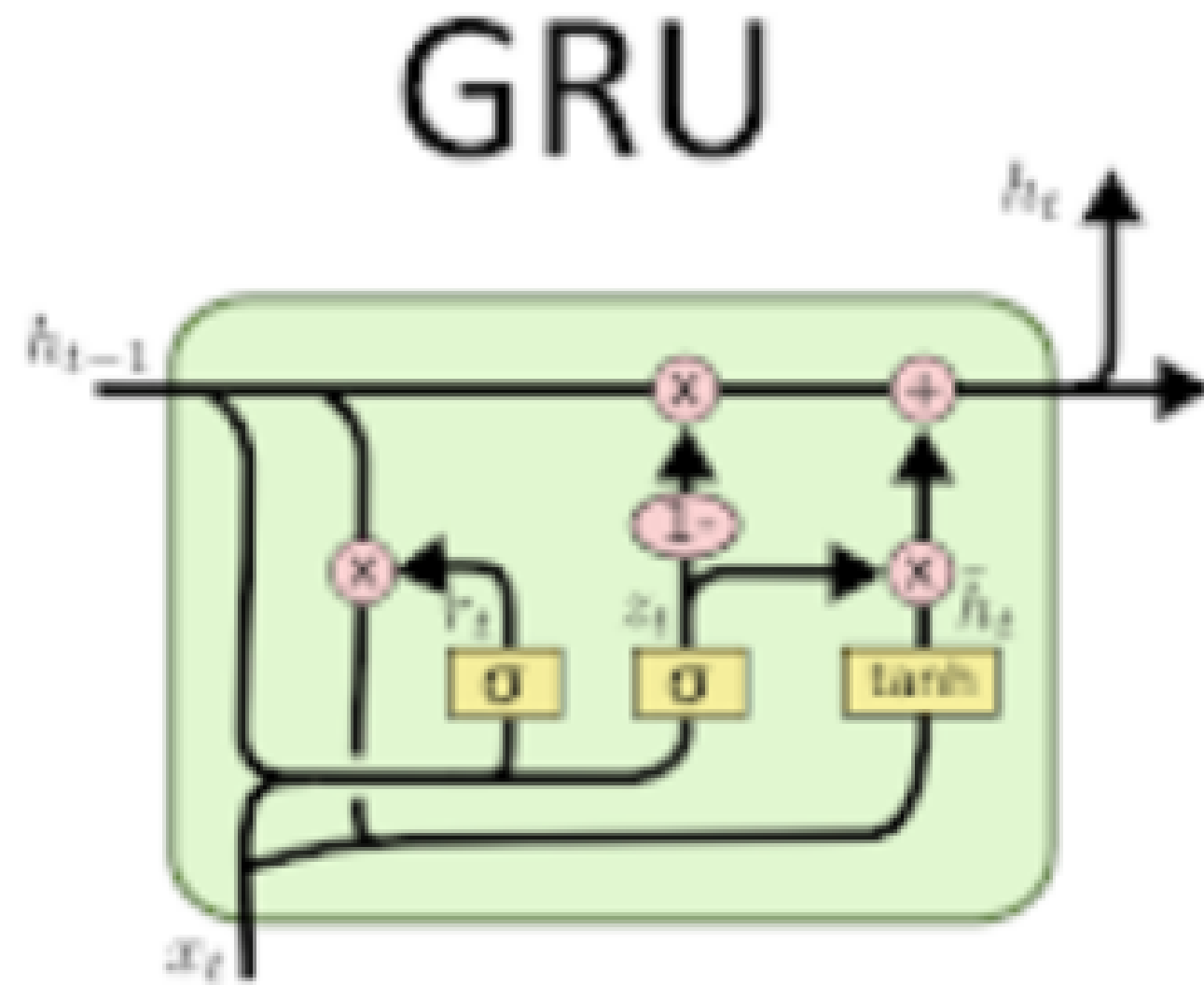
I went to the



store / market 등

장기 기억 특성이 Quiz 3에 유용하게 작용

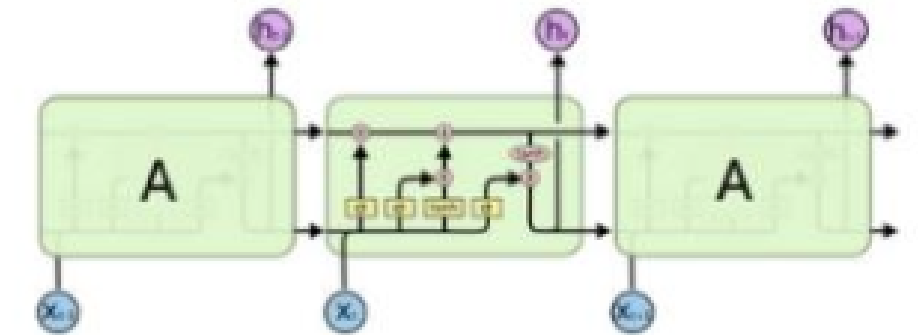
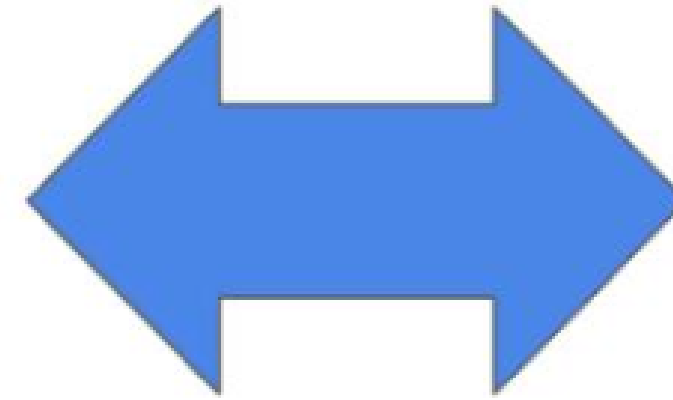
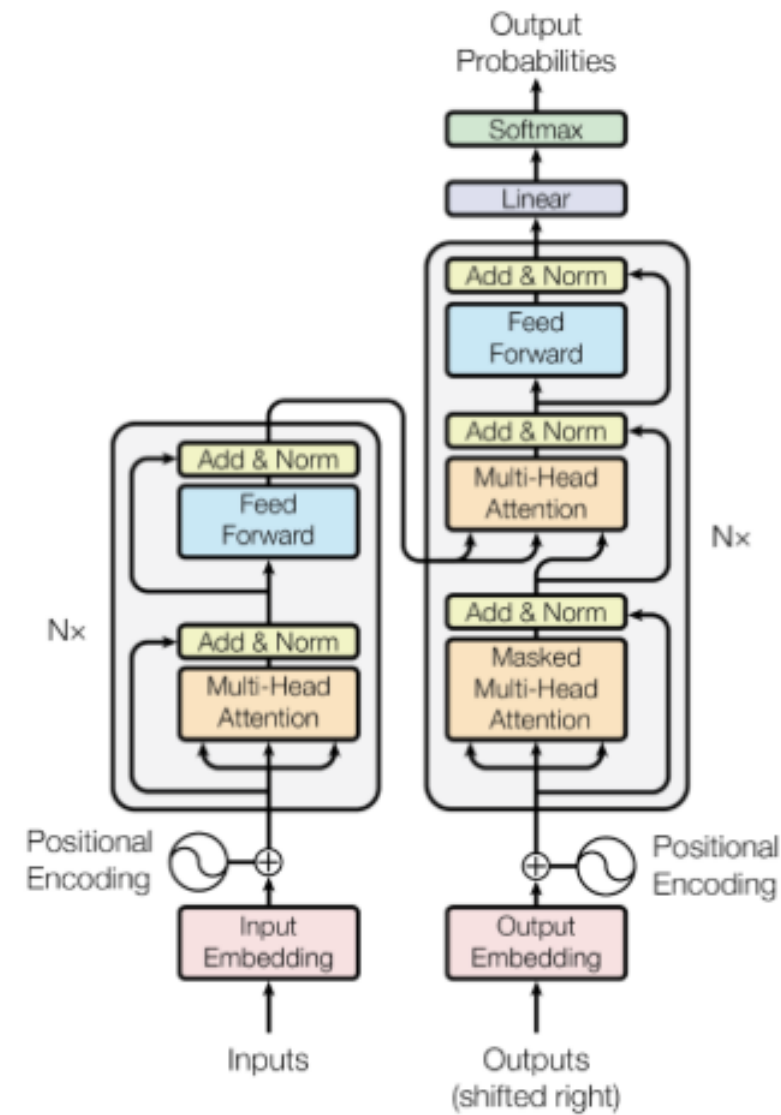
시퀀스 모델 비교



GRU

- LSTM의 변형 모델로 구조가 더욱 간단함
- LSTM의 입력 게이트와 망각 게이트를 하나의 업데이트 게이트로 통합하여 계산 효율성 높임

시퀀스 모델 비교



- Quiz 4 연산이 가능하여 더 빠르게 훈련할 수 있음
- 더 긴 문맥을 한 번에 처리할 수 있음

LSTM 수학적 기법



입력 게이트

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

망각 게이트

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

셀 상태 업데이트

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

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

출력 게이트


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

최종 출력

$$h_t = o_t \cdot \tanh(C_t)$$



LSTM 구현 예제



```
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.keras.datasets import imdb
from tensorflow.keras.preprocessing import sequence

# Load the IMDB dataset
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=10000)

# Pad the sequences to the same length
max_len = 500
x_train = sequence.pad_sequences(x_train, maxlen=max_len)
x_test = sequence.pad_sequences(x_test, maxlen=max_len)

# Build the model
model = tf.keras.Sequential()
model.add(tf.keras.layers.Embedding(10000, 128))
model.add(tf.keras.layers.LSTM(128, dropout=0.2, recurrent_dropout=0.2))
model.add(tf.keras.layers.Dense(1, activation='sigmoid'))
```

LSTM 구현 예제



```
# Compile the model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# Train the model
history = model.fit(x_train, y_train, batch_size=32, epochs=10, validation_data=(x_test, y_test))

# Plot the loss and accuracy results
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper right')
plt.show()

plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='lower right')
plt.show()
```


동영상 링크

감사합니다