인공지능 기초

인공지능_ Day10

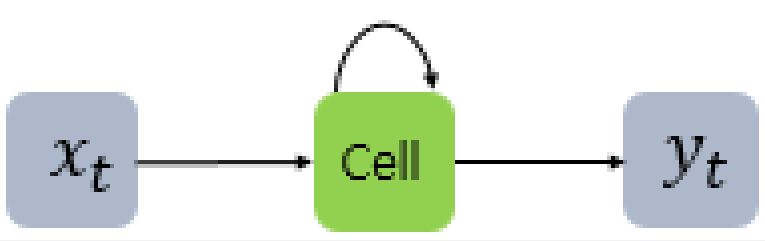
2023. 07. 07

CONTENTS

RNN(Recurrent Neural Network)
순환 신경망

RNN

- 순환 신경망(Recurrent Neural Network)은 은닉 계층 안에 하나 이상의 순환 계층을 갖는 신경망을 의미함
- 순환 신경망은 다른 네트워크들과 입력을 받는 방식에서 다른데 순서가 있는 데이터에 주로 사용됨
- 해당 데이터를 입력으로, 하나의 네트워크를 통해서 순서대로 출력을 얻음
- 순서가 있는 데이터는 소리, 언어, 날씨, 주가 등의 데이터처럼 시간의 변화에 함께 변화하면서 그 영향을 받는 데이터를 의미함
- input->RNN->output->input->RNN->output….이런 구조로 계속 반복되는 모델



시계열(Time Series)

- 시계열(time series) : 일정 시간 간격으로 배치된 데이터들의 수열
- 시계열 예측(time series prediction) : 주어진 시계열을 보고 수학적인 모델을 만들어서 미래에 일어날 것들을 예측하는 것
- 일반적으로 공학이나 과학계산, 혹은 금융시장에서의 주가 예측 등에서 많이 사용됨
- 시계열 데이터는 통상 y가 없다
- RNN의 문제점

  1) 시간차가 많이 나는 정보는 기울기 소실

  2) 장기의존성(long-term dependencies)

RNN

사용 Library

```
import numpy as np
from keras.models import Sequential
From keras.layers import Dense, SimpleRNN
```

1. 데이터

```
datasets = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
x = np.array([[1, 2, 3], [2, 3, 4], [3, 4, 5], [4, 5, 6], [5, 6, 7], [6, 7, 8], [7, 8, 9]])
y = np.array([4, 5, 6, 7, 8, 9, 10])

# x의 shape = (행, 열, timesteps!!!)
x = x.reshape(7, 3, 1)
print(x.shape)
```

RNN

2. 모델 구성

```
model = Sequential()
model.add(SimpleRNN(128, input_shape=(3, 1)))    # [batch, feature, timesteps]
model.add(Dense(128, activation='relu'))         #  RNN은 3차원의 데이터를 input하여
model.add(Dense(128, activation='relu'))         #  2차원의 output이 나오기 때문에 Dense로 받아줌
model.add(Dense(32, activation='relu'))
model.add(Dense(1))
```

3. 컴파일, 훈련

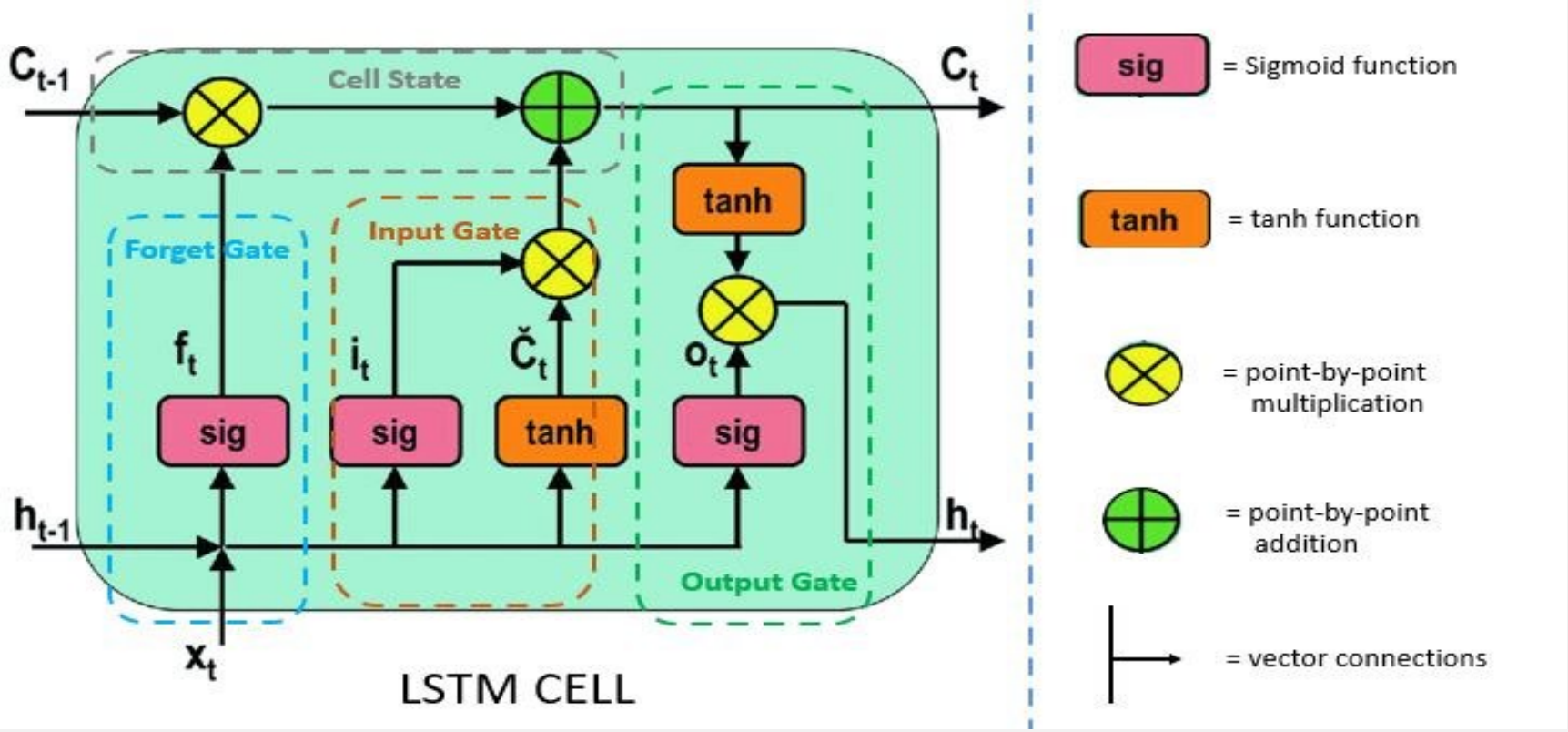
```
model.compile(loss='mse', optimizer='adam')
model.fit(x, y, epochs=500, batch_size=1)
```

4. 평가, 예측

```
loss = model.evaluate(x, y)
y_pred = np.array([8, 9, 10]).reshape(1, 3, 1)
result = model.predict(y_pred)
print('loss : ', loss)
print('[8, 9, 10]의 결과 : ', result)
```

LSTM

- LSTM는 기존의 RNN이 출력과 먼 위치에 있는 정보를 기억할 수 없다는 단점을 보완하여 장/단기 기억을 가능하게 설계한 모델이며 주로 시계열 처리나 자연어 처리에 사용됨



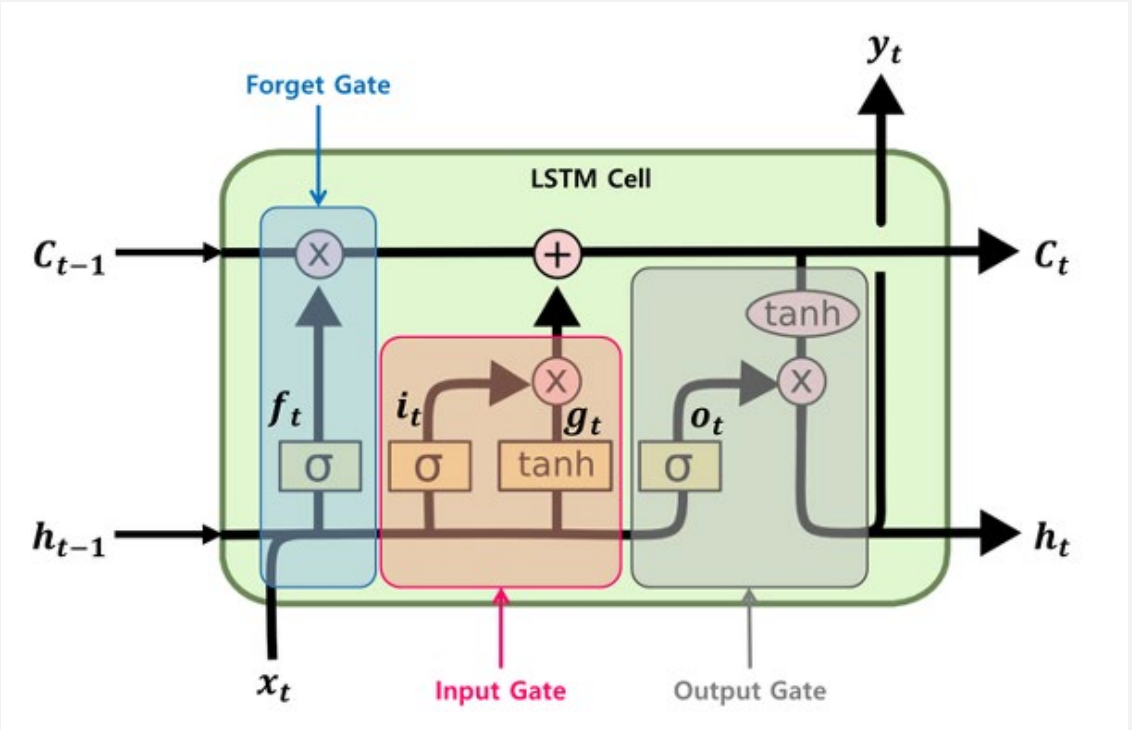
LSTM

1) Forget gate : cell state에서 sigmoid layer를 거쳐 어떤 정보를 버릴 것인지 결정

2) Input gate : 앞으로 들어오는 새로운 정보 중 어떤 것을 저장할 것인지 결정. Sigmoid layer를 거쳐 어떤 값을 업데이트 할 것인지를 정한 후 tanh layer에서 새로운 후보 Vector를 만듦

3) Cell state : 버릴 정보들과 업데이트할 정보를 받아 업데이트를 진행함

4) Output gate : 어떤 정보를 내보낼 지 결정



LSTM

사용 Library

```
import numpy as np
from keras.models import Sequential
From keras.layers import Dense, LSTM
```

1. 데이터

```
datasets = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
x = np.array([[1, 2, 3], [2, 3, 4], [3, 4, 5], [4, 5, 6], [5, 6, 7], [6, 7, 8], [7, 8, 9]])
y = np.array([4, 5, 6, 7, 8, 9, 10])

# x의 shape = (행, 열, timesteps!!!)
x = x.reshape(7, 3, 1)
print(x.shape)
```

LSTM

2. 모델 구성

```
model = Sequential()
model.add(LSTM(128, input_shape=(3, 1)))    # [batch, feature, timesteps]
model.add(Dense(128, activation='relu'))          #  LSTM은 3차원의 데이터를 input하여
model.add(Dense(128, activation='relu'))          #  2차원의 output이 나오기 때문에 Dense로 받아줌
model.add(Dense(32, activation='relu'))
model.add(Dense(1))
```

3. 컴파일, 훈련

```
model.compile(loss='mse', optimizer='adam')
model.fit(x, y, epochs=300, batch_size=1)
```

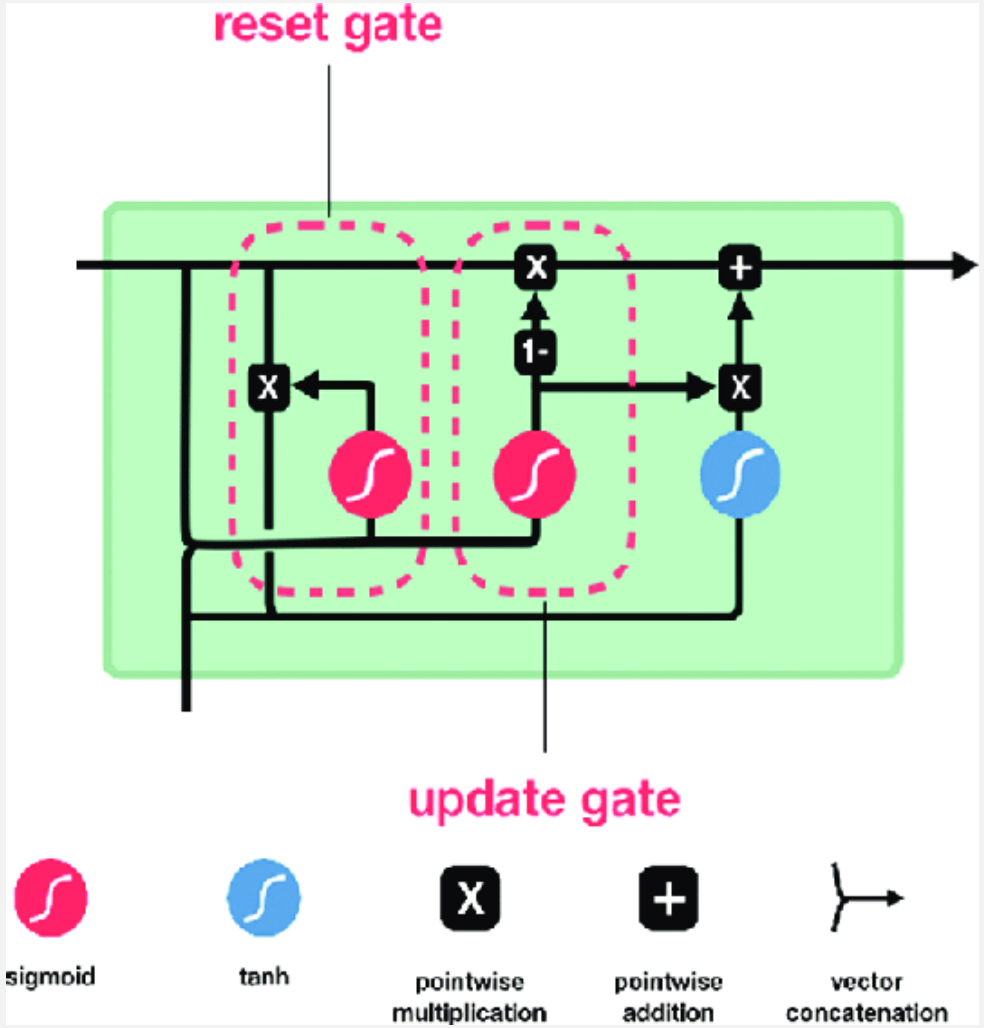
4. 평가, 예측

```
loss = model.evaluate(x, y)
y_pred = np.array([8, 9, 10]).reshape(1, 3, 1)
result = model.predict(y_pred)
print('loss : ', loss)
print('[8, 9, 10]의 결과 : ', result)
```

GRU

- LSTM는 기존의 RNN의 문제를 해결하면서 긴 시퀀스를 가진 데이터에서도 좋은 성능을 내는 모델이지만, 복잡한 구조때문에 RNN에 비해 파라미터가 많이 필요하고 데이터가 충분하지 않은 경우 오버피팅이 발생

1) Reset gate : 이전의 정보를 얼마나 잊어야하는지 결정
2) Update gate : 이전의 정보를 얼마나 통과시킬지 결정
   LSTM의 forget gate + input gate
3) hidden state : LSTM의 cell state의 역할을 함
   이전 셀에서 넘어온 정보를 output으로 내보냄



GRU

사용 Library

```
import numpy as np
from keras.models import Sequential
From keras.layers import Dense, GRU
```

1. 데이터

```
datasets = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
x = np.array([[1, 2, 3], [2, 3, 4], [3, 4, 5], [4, 5, 6], [5, 6, 7], [6, 7, 8], [7, 8, 9]])
y = np.array([4, 5, 6, 7, 8, 9, 10])

# x의 shape = (행, 열, timesteps!!!)
x = x.reshape(7, 3, 1)
print(x.shape)
```

GRU

2. 모델 구성

```
model = Sequential()
model.add(GRU(128, input_shape=(3, 1)))          # [batch, feature, timesteps]
model.add(Dense(128, activation='relu'))         #  GRU는 3차원의 데이터를 input하여
model.add(Dense(128, activation='relu'))         #  2차원의 output이 나오기 때문에 Dense로 받아줌
model.add(Dense(32, activation='relu'))
model.add(Dense(1))
```

3. 컴파일, 훈련

```
model.compile(loss='mse', optimizer='adam')
model.fit(x, y, epochs=300, batch_size=1)
```

4. 평가, 예측

```
loss = model.evaluate(x, y)
y_pred = np.array([8, 9, 10]).reshape(1, 3, 1)
result = model.predict(y_pred)
print('loss : ', loss)
print('[8, 9, 10]의 결과 : ', result)
```

실 습

1. RNN
2. LSTM
3. GRU
4. 팀프로젝트

Day07. 인공지능 Study

1. 인공지능 개념 정리 – 머신러닝, 딥러닝
2. 퍼셉트론 (Perceptron)
3. 다층 퍼셉트론 (Multi-Layer Perceptron: MLP)
4. 옵티마이저 (Optimizer)
5. 학습률 (learning rate)
6. 경사하강법 (Gradient Descent)
7. 손실함수 (Loss Function)
8. 활성화 함수 (Activation Function) – Sigmoid, ReLU, Softmax

9. 회귀분석

10. 결정계수 R2 score

11. 분류분석

12. 원 핫 인코딩 (One Hot Encoding)

13. 난수값 (random_state)

14. 정확도 accuracy score

15. 과적합 (overfitting)

16. 합성곱신경망(CNN)

17. 이미지증강(ImageDataGenerator)

18. 자연어처리(Word Embedding)

19. SVM model

20. Decision Tree model

21. K-Fold

22. Boosting model

23. 그리드서치

24. 배깅

25. 보팅

26. 아웃라이어

27. RNN (순환 신경망)

28. LSTM

수고하셨습니다.