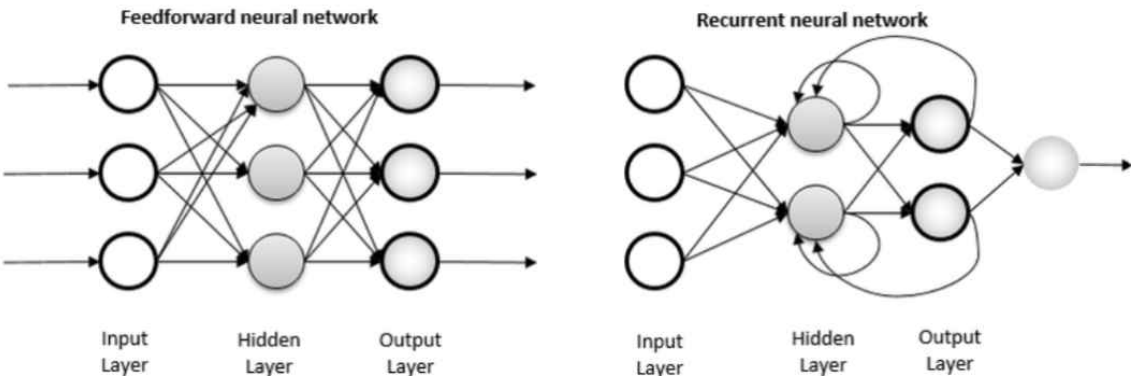
4주차 CNN과 RNN의 이해

8차시: RNN(순환 신경망)과 LSTM의 이해

- RNN의 구조와 순차 데이터 처리 방법
- LSTM(Long Short-Term Memory)의 개념과 개선된 RNN 모델
- GRU(Gated Recurrent Unit)
- RNN & LSTM 적용사례

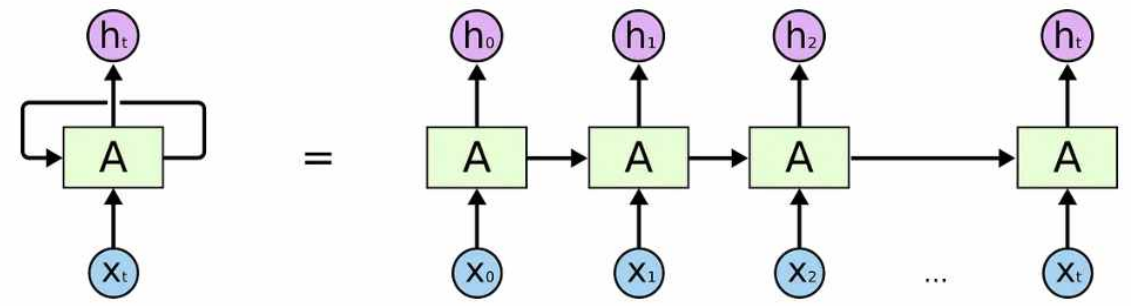
○ RNN의 구조와 순차 데이터 처리 방법

- RNN (Recurrent Neural Network)은 순차 데이터(Sequential data)[1]나, 처리시간 의존 데이터를 처리[2]하는 데 특화된 신경망 구조입니다.
- 순차적 관계를 가지는 데이터를 다루기 위해, RNN은 과거의 정보를 기억하고 이를 바탕으로 현재의 출력을 생성하는 순환 구조를 가지고 있습니다.



결국 RNN은 전 시점(t-1)의 어떤 정보를 다음 시점(t)으로 넘겨준다고 볼 수 있습니다.
다음 시점(t)의 정보는 전 시점(t-1)의 정보만이 아니라 이전까지의 정보(t-2 + t-3 + … + t-n)들을 모두 가지고 있을 것입니다.
그리고 이처럼 정보를 가지고 있는 것을 cell이라고 하며 현재 cell이 가지고 있는 정보, 즉 다음 시점으로 넘겨줄 정보를 hidden state[3] 라고 합니다.

(hidden layer= hidden state 와 입력값)



Xt 는 입력값, ht의 결과값

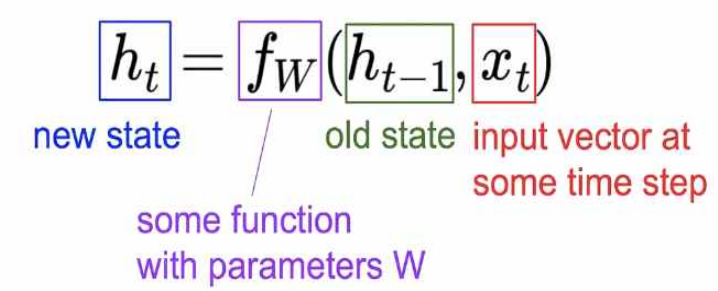
---

1) 주석1
2) 주석2
3) 주석3

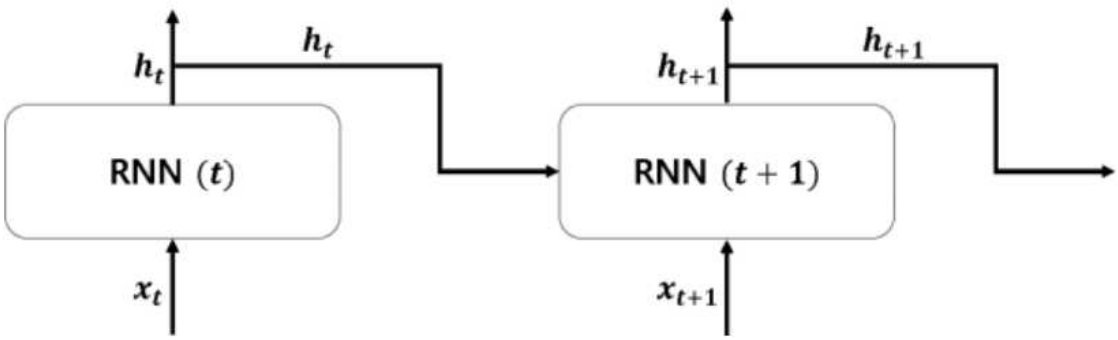
순환 단계(Recurrence step)
시점(time step)마다 입력 데이터와 이전 시점의 **은닉 상태(hidden state)**를 바탕으로 새로운 은닉 상태를 계산하는 과정입니다.
이 순환 단계는 순차적 데이터에서 시간적 의존성을 학습하는 RNN의 핵심 메커니즘입니다.

1. RNN(Recurrent Neural Networks, 순환신경망) 구조

- RNN은 일반적인 신경망과 달리, 순환 구조를 가지고 있습니다.
- 즉, RNN의 뉴런은 이전 단계의 출력을 현재 입력과 함께 사용하는 구조로, 순환 연결을 통해 과거의 정보를 저장합니다.



- RNN은 오직 ***은닉층에서 활성화 함수를 지난 값이 출력층의 방향으로만 이동하는 피드 포워드 신경망과는 다르게*** *은닉층에서 활성화 함수를 지난 값이 출력층의 방향으로도 이동하지만 더불어 다시 은닉층의 노드의 다음 계산의 입력으로 보내는 특징이 있습니다.*



입력()을 출력()로 변환하기 위한 가중치와, RNN 출력을 다음 시각(t)의 출력으로 변환하기 위한 가중치, 편향으로 이루어져 있습니다.
먼저 행렬 곱을 계산한 후, 그 합을 tanh 함수(tanh; Hyperbolic tangent, 쌍곡 탄젠트 함수)를 이용해 변환하여 시간이 출력됩니다.
이는 다른 계층을 향해 위쪽으로 출력되는 동시에, 다음 시각 의 RNN 계층으로도 출력됩니다.

2. RNN의 주요 구성 요소:

- 입력층(Input Layer): 순차 데이터(예: 단어, 시간 단계별 데이터 등)를 입력받는 층
- **은닉층(Hidden Layer):**

▷ RNN에서 은닉층은 **입력과 은닉 상태가 결합**되는 뉴런들의 층입니다.
▷ 입력층과 출력층 사이에 위치하며, RNN의 계산을 수행하는 신경망 계층입니다.
▷ RNN의 핵심인 순환 구조가 있는 층→ 이전 단계의 **은닉 상태(hidden state)**와 현재 입력을 결합하여 새로운 은닉 상태를 계산합니다.
▷ 은닉층은 RNN의 입력 데이터를 처리하고, 은닉 상태를 업데이트하는 역할을 합니다.
▷ 일반적으로, 입력층에서 들어오는 데이터와 이전 시간의 은닉 상태가 이 은닉층에서 계산되고 다음 시간의 은닉 상태로 전달됩니다.

- 출력층(Output Layer): 각 시점에서의 출력 값을 생성하는 층 →입력 시퀀스 전체에 대한 출력(예: 문장의 다음 단어 예측, 시계열의 미래 값 예측 등)을 만듭니다.

3. RNN의 작동 원리

1) 기본 순환 구조 : **입력 - 상태 - 출력**

RNN의 은닉층은 순환 구조를 가지며, 매 시점에서 다음과 같은 방식으로 작동합니다:

- 입력: RNN은 시퀀스 데이터의 각 시점에 대해 입력 값을 받습니다. → 이를 x_t로 나타냅니다. 여기서 t는 시간 단계(time step)를 의미합니다.
- 은닉 상태 업데이트: 이전 시점의 은닉 상태 h{t-1}과 현재 시점의 입력 값 x_t를 사용하여 새로운 은닉 상태 h_t를 계산합니다.

$$h_t = f(W_h h_{t-1} + W_x x_t + b_h)$$

- h_t : 현재 시점 t에서의 **은닉 상태(hidden state)**
- h_{t-1} : 이전 시점 t-1에서의 은닉 상태
- x_t : 현재 시점 t에서의 입력
- W_h : 은닉 상태에서 은닉 상태로 연결되는 가중치 행렬
- W_x : 입력에서 은닉 상태로 연결되는 가중치 행렬
- b_h : 은닉 상태에 대한 편향(bias)
- f : 비선형 활성화 함수(예: tanh 시그모이드, ReLU 등)

이 수식은 RNN이 매 시점마다 현재 입력과 이전 은닉 상태를 조합하여 새로운 은닉 상태를 계산하는 방식입니다.

▶ RNN과 **First Order System 개념** 비교

- First Order System:

물리학, 제어 시스템 등에서의 일차 시스템(First Order System)은 현재 상태가 이전 상태에 의존하고, 그 상태가 시간에 따라 변화하는 시스템을 의미합니다.
가장 일반적인 형태의 1차 시스템은 **시간에 따라 선형적으로 변화하는 방정식으로 표현**됩니다.
예를 들어, 전기 회로에서 RC 회로나 기계적인 감쇠 시스템 등이 1차 시스템의 예입니다. 이

는 시스템이 이전 입력값의 영향을 받아 천천히 수렴하거나 변동하는 특성을 가집니다.

RNN에서 First Order System과의 유사성:

- 과거 정보 반영: RNN은 순차적인 데이터에서 과거 정보를 반영하여 미래의 상태를 예측하거나 처리하는 방식으로 동작합니다. 마치 1차 시스템이 현재 상태가 과거 입력에 의해 결정되는 것처럼, RNN도 과거의 숨겨진 상태와 현재 입력을 통해 새로운 상태를 계산합니다.
- 시간에 따른 변화: 물리 시스템에서 1차 시스템은 시간이 지남에 따라 입력이 시스템에 영향을 미치며, RNN도 시계열 데이터에서 이전 상태와 현재 입력을 결합하여 시간이 지남에 따라 변화하는 상태를 모델링합니다.
- 메모리 및 상태 유지: 1차 시스템은 일차 방정식에서 과거 상태가 일정 비율로 반영되며, RNN 역시 현재 상태를 계산할 때 이전 상태를 반영하여 일종의 메모리를 유지합니다. 이는 RNN이 시계열 데이터나 자연어 처리와 같은 연속적인 데이터를 처리하는 데 매우 효과적인 이유 중 하나입니다.

▶ **State Space Model**(상태 공간 모델)

- State Space Model(상태 공간 모델)은 시스템의 현재 상태가 이전 상태와 현재 입력에 따라 결정된다는 원리에서 출발합니다.
- 상태 공간 모델(State Space Model)은 제어 시스템이나 신호 처리에서 자주 사용되는 모델링 기법으로, 시간에 따른 시스템의 동작을 설명하는 데 쓰입니다.
- RNN도 마찬가지로, 이전 상태와 현재 입력을 기반으로 미래 상태를 예측하므로, 상태 공간 모델과 유사한 개념을 가지고 있습니다.

State Space Model(상태 공간 모델)의 기본 개념
상태 공간 모델에서는 시스템의 동작을 상태(state)와 입력(input)**을 통해 수학적으로 나타냅니다.
시스템의 동작을 묘사하는 기본 방정식은 다음과 같습니다.

상태 공간 방정식

$$\mathbf{h}_t = A\mathbf{h}_{t-1} + B\mathbf{x}_t + \mathbf{b}$$

h t : 현재 상태 벡터 (현재 시간 t 에서의 상태)
h $t-1$: 이전 상태 벡터 (이전 시간 $t-1$에서의 상태)
$x\ t$: 현재 입력 벡터
A: 상태 전이 행렬 (현재 상태를 계산하기 위해 이전 상태에 곱하는 행렬)
B: 입력 행렬 (입력 값이 상태에 미치는 영향을 조정하는 행렬)
b: 편향 벡터

출력 계산

- 출력 계산: 은닉 상태 h_t에서 출력을 계산합니다.

$$y_t = W_{hy}h_t + b_y$$

W_{hy} : 은닉 상태에서 출력으로의 가중치

RNN의 구조도 상태 공간 모델과 매우 유사합니다. RNN은 이전 상태와 현재 입력을 결합하여 새로운 상태를 계산하고, 이를 기반으로 출력도 예측합니다.

RNN을 상태 공간 모델로 시각화

- 상태: h t 는 RNN에서의 숨겨진 상태로, 상태 공간 모델에서 시스템의 현재 상태에 해당합니다.
- 입력: x t 는 현재 입력으로, 상태 공간 모델에서의 외부 입력에 해당합니다.
- 출력: y t 는 RNN의 출력으로, 상태 공간 모델에서의 시스템 출력과 유사합니다.

2) RNN의 순환 과정 시각화

RNN에서의 순환 관계는 시간축을 따라 반복적으로 계산되는 과정을 시각적으로 표현

- **시점 t-1**에서: $h_{t-1} = f(W_h h_{t-2} + W_x x_{t-1} + b_h)$
- **시점 t**에서: $h_t = f(W_h h_{t-1} + W_x x_t + b_h)$
- **시점 t+1**에서: $h_{t+1} = f(W_h h_t + W_x x_{t+1} + b_h)$

3) . RNN 순환 과정의 장점과 한계

장점:

- 순차적 데이터를 처리하는 데 탁월하며, 시간적 의존성을 학습할 수 있습니다.
- 이전 시점의 정보를 기억하여, 다음 시점에서 이를 활용할 수 있습니다.

한계:

- RNN은 장기 의존성을 학습하는 데 어려움을 겪을 수 있습니다.
- 기울기 소실(Vanishing Gradient) 문제가 발생하여, 멀리 떨어진 과거 정보가 현재 시점에 제대로 전달되지 않을 수 있습니다.
- 이 문제를 해결하기 위해 LSTM이나 GRU와 같은 변형 모델이 도입되었습니다. 이들은 게이트 구조를 통해 장기 의존성을 효과적으로 처리할 수 있습니다.

3) 순환 연결

- RNN의 주요 특징은 은닉 상태 h_t가 이전 시점의 은닉 상태 h_{t-1}을 입력으로 받아들여, 시간적인 의존성을 고려할 수 있다는 점입니다.
- 이를 통해 RNN은 시퀀스에서 이전 정보를 학습하고 이를 바탕으로 현재 상태를 결정할 수 있습니다.

4) **역전파 (Backpropagation Through Time, BPTT)**

RNN은 시간을 따라 역전파하는 방식을 사용하여 학습됩니다.

→ BPTT(Backpropagation Through Time)이라고 하며, 기본적인 역전파 알고리즘을 시퀀스의 각 시점으로 확장한 개념입니다.

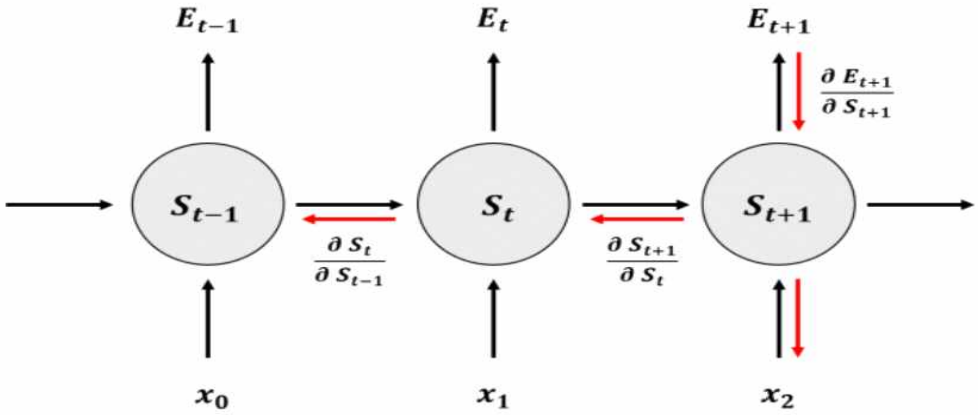
* BPTT(BackPropagation Through Time)

RNN의 학습도 보통 신경망과 같은 순서로 진행되는데 이때 가중치 매개변수의 기울기를 효율적으로 계산할 수 있는 오차 역전파 법(Backpropagation)을 사용합니다.

역전파란 인공 신경망을 학습시키기 위한 일반적인 알고리즘 중 하나로, 역방향으로 해당 함수의 국소적 미분을 곱해 나가는 방법입니다. 내가 출력하고자 하는 값과 실제 모델이 계산한 값이 얼마나 차이가 나는지 계산한 후 그 오차 값을 다시 전달하며 각 노드가 가지고 있는 값(Weight, Bias)들을 업데이트하기 위한 알고리즘입니다.

빨간색으로 표기된 수식들이 역전파 계산을 의미하며, 역방향으로 해당 함수의 미분을 곱해 나가는 모습을 보이고 있습니다. 위의 역전파를 기본으로 하여 RNN은 시간 방향으로 펼친 신경망의 역전파를 수행하며 이를 BPTT(BackPropagation Through Time)라고 합니다.

위의 그림과 같이 진행하게 될 경우 시계열 데이터의 시간 크기가 커지는 것에 비례하여



BPTT(BackPropagation Through Time)

BPTT가 소비하는 컴퓨팅 자원이 증가하며, 시간 크기가 커지면 역전파 시의 기울기가 불안정해지는 문제가 생깁니다. 이러한 문제를 해결하기 위해 큰 시계열 데이터를 취급할 때는 흔히 신경망 연결을 적당한 길이로 끊습니다. 그리고 이 잘라낸 신경망에서 역전파를 수행하게 되는데 이를 Truncated-BPTT(Truncated-BackPropagation Through Time)라고 합니다.

3. 순차 데이터 처리 방법

1) 시계열 데이터 처리

RNN은 각 시간 단계에서 이전 시간 단계의 정보를 고려하여 현재 값을 예측할 수 있습니다.

예시) 과거 몇 분간의 주식 가격을 사용해 다음 순간의 주식 가격을 예측할 수 있습니다.

2) 자연어 처리

RNN은 단어 시퀀스를 처리하여 문장을 생성하거나 번역하는 데 자주 사용됩니다.

예시) 입력으로 "The cat is"라는 문장을 받으면, RNN은 앞의 단어들을 바탕으로 다음 단어를 예측할 수 있습니다.

각 단어가 이전 단어와 문맥적으로 연결되므로, RNN은 문맥을 유지하면서 텍스트 데이터를 처리할 수 있습니다.

3) 시퀀스-투-시퀀스(Sequence-to-Sequence) 모델

RNN은 입력 시퀀스를 다른 시퀀스로 변환하는 Seq2Seq 모델에서 널리 사용됩니다.

예시) 기계 번역에서 영어 문장을 입력으로 받고, 대응되는 프랑스어 문장을 출력으로 생성하는 방식입니다.

4. RNN의 한계와 문제점

1) 기울기 소실 문제(Vanishing Gradient Problem)

- RNN은 시간이 길어질수록 기울기 소실 문제가 발생하여, 오래된 정보를 학습하는 데 어려움을 겪습니다.
- 시퀀스가 길어질수록 역전파 과정에서 이전 시점의 기울기가 매우 작아져, ***먼 과거의 정보가 제대로 전달되지 않는 문제가*** 발생합니다.

2) 장기 의존성(Long-Term Dependencies)의 처리 어려움

RNN은 가까운 시점의 정보는 잘 처리할 수 있지만, 오랜 시간 간격을 두고 관련된 정보를 처리하는 데 어려움이 있습니다. 즉, 장기 의존성을 학습하는 데 한계가 있습니다.

(어려운 이유)

- 기울기 소실과 폭주 문제: RNN은 시간 축을 따라 오차를 역전파하며 가중치를 업데이트하는데, 이 과정에서 긴 시퀀스를 다루다 보면 기울기 값이 급격히 작아지거나 (기울기 소실) 커질 수 (기울기 폭주) 있습니다. 기울기가 소실되면 이전 입력으로부터 멀리 떨어진 정보를 학습하는 것이 어려워지므로 장기 의존성 학습에 어려움이 생깁니다.
- 짧은 기억력: 전통적인 RNN은 짧은 시퀀스 내에서 패턴을 학습하는 데 더 적합합니다. 긴 시퀀스를 처리할 때 RNN은 이전 정보를 "잊어버리는" 경향이 있어, 긴 문맥에 걸친 관계나 의존성을 이해하기가 어렵습니다.
- 순차적 처리: RNN은 입력을 한 단계씩 순차적으로 처리하는데, 이는 긴 시퀀스를 처리할수록 누적된 오차가 커질 수 있습니다. 결과적으로 이전 단계의 정보를 후속 단계에 전달하는 데 한계가 생깁니다.
- 게이트 메커니즘 부재: RNN에는 정보를 선택적으로 유지하거나 잊는 기능이 부족합니다. LSTM(Long Short-Term Memory)이나 GRU(Gated Recurrent Unit) 모델은 이러한 문제를 해결하기 위해 게이트 메커니즘을 도입하여 장기 정보를 효율적으로 조절하고 기억할 수 있습니다.
- 계산 효율성: 순차적인 특성 때문에 긴 시퀀스를 다룰 때 RNN은 많은 계산을 요구하며, 이로 인해 긴 문맥을 관리하는 데 효율성이 떨어집니다.

RNN의 구조와 학습 과정 :기울기 소실과 폭주 문제 상세 들여다 보기

RNN은 순차적인 데이터를 처리하기 위해 이전 단계의 출력을 다음 단계의 입력으로 사용하면서 시간적 의존성을 유지합니다. RNN의 구조는 다음과 같습니다:

시점(t)마다 입력 x_t 가 들어오면, RNN은 그 입력과 이전 시점의 은닉 상태(hidden state) h_{t-1} 를 결합하여 새로운 은닉 상태 h_t 를 계산합니다.

은닉 상태 h_t 는 입력 데이터의 과거 정보를 기억하여, 현재 시점의 출력을 결정하는 데 사용됩니다.

RNN의 학습은 역전파를 통해 이루어지며, 이때 각 시점의 가중치는 과거 시점에서 발생한 오류(손실 함수의 기울기)를 고려하여 조정됩니다.
이 과정에서 손실 함수의 기울기(gradient)가 연속된 시점에서 곱셈으로 전파되기 때문에 기울기 소실 또는 기울기 폭발 문제가 발생할 수 있습니다.

기울기 소실(Vanishing Gradient) 문제
장기 의존성(Long-Term Dependencies)이란, 현재의 출력이 과거의 특정 정보(예: 긴 텍스트에서 멀리 떨어진 단어들)에 크게 의존하는 경우를 의미합니다.
하지만 RNN은 이런 장기 의존성을 학습하는 데 어려움을 겪습니다.
그 이유는 주로 기울기 소실로 인해 과거의 정보를 효과적으로 전달하지 못하기 때문입니다.

기울기 소실은 역전파 과정에서 손실 함수의 기울기가 계속해서 작아지는 현상을 말합니다.
RNN은 각 시점에서의 손실을 역전파하는데, 이 과정에서 기울기 값이 0에 가까워지면 가중치가 거의 업데이트되지 않게 됩니다.
특히, 시퀀스가 길어질수록 과거 시점에서 발생한 오류 정보가 현재 시점으로 전달되기 어렵습니다. 이는 과거 정보가 현재 시점에서 중요한 역할을 할 때 문제가 됩니다.
기울기 소실이 발생하는 이유:
RNN의 은닉 상태는 다음과 같은 재귀적 구조로 계산됩니다:

$$h_t = f(W_h h_{t-1} + W_x x_t)$$

Wh 와 W_x : 가중치 행렬
f: 비선형 활성화 함수(예: tanh 또는 ReLU)
역전파 시, 손실 함수에 대한 각 가중치의 기울기는 연속된 곱으로 표현되기 때문에, 기울기가 점점 작아져서 0에 수렴하는 경우가 발생합니다.

활성화 함수로 많이 사용되는 시그모이드(sigmoid)나 하이퍼볼릭 탄젠트(tanh)는 출력값이 −1에서 1 사이로 제한되므로, 연속된 미분 값이 1보다 작은 값을 갖습니다.
이 작은 값들이 계속 곱해지면 기울기가 기하급수적으로 감소하게 되어, 결국 아주 작은 값이 되어버립니다.
따라서 긴 시퀀스에서 멀리 떨어진 과거의 정보는 현재 시점에서 거의 반영되지 않으며, 모델은 장기 의존성을 학습하는 데 어려움을 겪게 됩니다.

기울기 폭발(Exploding Gradient)
기울기 폭발은 기울기 소실과 반대로, 역전파 과정에서 기울기 값이 지수적으로 커지는 현상입니다. 기울기가 매우 커지면 가중치 값이 급격하게 변하면서 학습이 불안정해지고, 최적화가 제대로 이루어지지 않을 수 있습니다. 그러나 RNN의 주요 문제는 기울기 폭발보다는 기울기 소실에 가깝습니다.

LSTM과 GRU의 도입: 장기 의존성 문제 해결

RNN이 장기 의존성 문제를 해결하는 데 한계를 보이기 때문에, 이를 개선하기 위해 **LSTM(Long Short-Term Memory)과 GRU(Gated Recurrent Unit) 같은 변형 모델,**그리고 트랜스포머(Transformer) 아키텍처가 RNN을 대체하여 더 많이 사용되고 있습니다.

5. RNN의 변형 모델

RNN의 한계를 극복하기 위해 몇 가지 변형 모델이 개발되었습니다.

그 중 대표적인 두 가지는 LSTM(Long Short-Term Memory)과 GRU(Gated Recurrent Unit)입니다.

1) LSTM(Long Short-Term Memory)

LSTM은 RNN의 기울기 소실 문제를 해결하기 위해 게이트 메커니즘을 도입한 구조입니다.

LSTM은 **입력 게이트(Input Gate)**, **망각 게이트(Forget Gate)**, **출력 게이트(Output Gate)**를 사용하여 필요한 정보만 저장하고, 불필요한 정보는 제거함으로써 장기 의존성을 처리할 수 있습니다.

2) GRU(Gated Recurrent Unit)

GRU는 LSTM보다 구조가 단순하지만, 유사한 성능을 발휘하는 순환 신경망의 변형 모델입니다.

GRU는 LSTM과 마찬가지로 게이트 메커니즘을 사용하여 장기 의존성을 학습할 수 있으며, 계산 비용이 적고 학습 속도가 빠릅니다.

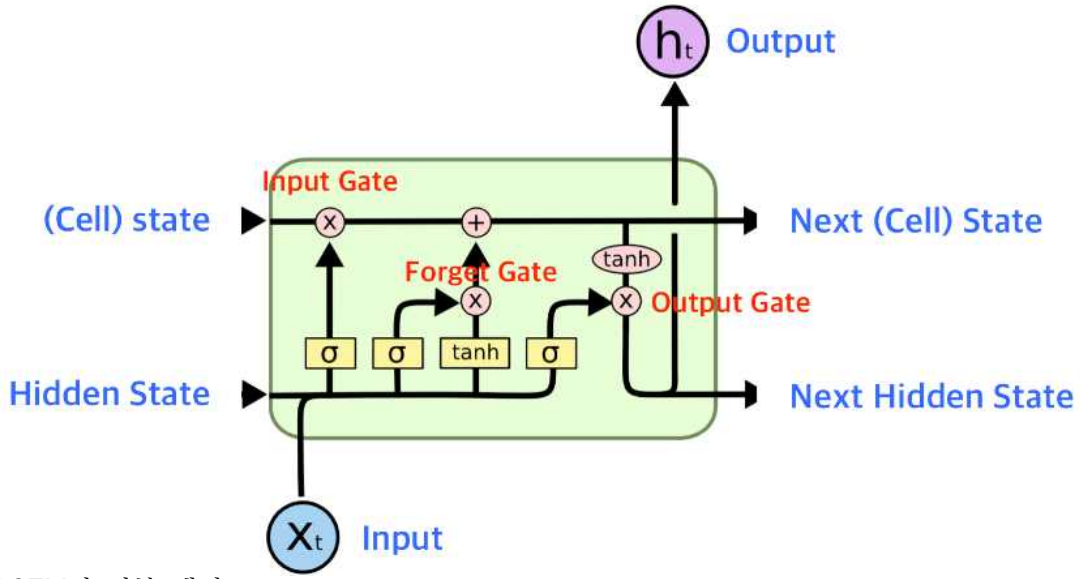
6. RNN의 실습 예시 (간단한 시계열 예측)

아래는 TensorFlow를 사용한 간단한 RNN 모델을 구축하는 예시입니다. 이 모델은 시계열 데이터를 학습하여 다음 값을 예측합니다.

python 실습 사례 : RNN OOO 주가 예측

○ LSTM(Long Short-Term Memory)의 개념과 개선된 RNN 모델

- LSTM(Long Short-Term Memory)는 순환 신경망(RNN)의 한 종류로 변형 모델입니다.
- 기울기 소실 문제(Vanishing Gradient Problem)를 해결하여 장기 의존성(Long-term dependencies)을 효과적으로 처리하기 위해 개발된 모델
- 게이트 구조를 도입하여 중요한 정보를 기억하고 불필요한 정보를 버리면서, 오랜 시퀀스에 걸친 의존성을 학습할 수 있습니다.



1. LSTM의 기본 개념

기본 RNN은 시간이 지나면서 과거 정보를 잊어버리기 쉽고, 특히 긴 시퀀스 데이터를 처리할 때는 기울기 소실 문제로 인해 오래된 정보를 활용하기 어렵습니다. LSTM은 이러한 한계를 극복하기 위해 메모리 셀과 게이트 구조를 사용하여 정보를 선택적으로 기억하거나 잊을 수 있도록 설계되었습니다.

LSTM의 핵심 특징

- 장기 및 단기 메모리 유지: LSTM은 오랜 시간 동안 중요한 정보를 기억하고, 필요 없는 정보는 잊어버릴 수 있는 메커니즘을 갖추고 있습니다.
- 게이트 구조: LSTM은 입력 게이트, 출력 게이트, 망각 게이트의 세 가지 게이트를 통해 정보 흐름을 제어합니다.
- 기울기 소실 문제 해결: LSTM은 각 시점에서의 기울기를 잘 유지할 수 있어, 긴 시퀀스에서도 정보가 손실되지 않고 학습할 수 있습니다.

2. LSTM의 구조

- LSTM의 각 셀(cell)은 세 가지 게이트를 가지고 있으며, 각 게이트는 시퀀스의 정보를 어떻게 처리할지 결정합니다.
- LSTM의 게이트(Gate)는 정보의 흐름을 제어하는 역할을 합니다.
- 각 게이트는 시그모이드(sigmoid) 활성화 함수를 사용하여 0에서 1 사이의 값을 출력합니다. 이 값은 정보를 얼마나 유지할지, 얼마나 잊을지를 결정합니다.

2.1 게이트 구성 (3개 Gate)

1) 입력 게이트(Input Gate)

입력 게이트는 새로운 정보가 셀 상태(Cell State)에 얼마나 반영될지를 결정하는 역할을 합니다. 현재 시점에서 들어오는 새로운 정보 중 중요한 정보만 선택적으로 셀 상태에 반영합니다.

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

- i t : 시점 t에서의 입력 게이트 값,
- h_{t-1} : 이전 시점의 은닉 상태
- x_t : 현재 시점의 입력 ,
- W_i : 입력 게이트의 가중치 행렬
- σ : 시그모이드 함수, 0에서 1 사이의 값으로 조정됩니다.
- b_i : 입력 게이트의 편향

역할:
입력 게이트는 새로운 정보 C~ t 가 셀 상태 C_t 에 얼마나 반영될지를 결정합니다.
시그모이드 함수의 출력이 0에 가까우면 그 정보는 거의 반영되지 않고, 1에 가까우면 정보가 셀 상태에 많이 반영됩니다.
입력 게이트와 새로운 정보는 다음 수식으로 결합됩니다.

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

최종적으로, 새로운 정보는 셀 상태에 반영됩니다.

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

2) 망각 게이트(Forget Gate)
: 망각 게이트는 셀 상태에서 이전 정보를 얼마나 잊을지(삭제할지)를 결정합니다.
즉, 셀 상태에 저장된 이전 정보 중에서 유지할 정보와 삭제할 정보를 선택하는 역할을 합니다.
수식:

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

- f t : 시점 t에서의 망각 게이트 값
- h_{t-1} : 이전 시점의 은닉 상태
- x_t :는 현재 시점의 입력
- W_f : 망각 게이트의 가중치 행렬
- σ : 시그모이드 함수
- b_f : 망각 게이트의 편향

역할:

망각 게이트는 셀 상태
C_{t-1} 의 어느 부분을 잊을지를 결정합니다.
f_t 의 값이 0에 가까우면 그 정보는 완전히 잊혀지고, 1에 가까우면 그대로 유지됩니다.
이 값은 다음과 같이 셀 상태와 곱해져 기존 셀 상태에서 일부 정보가 제거됩니다.

$$C_t = f_t \cdot C_{t-1}$$

3) 출력 게이트(Output Gate)

- 출력 게이트는 은닉 상태(hidden state)를 통해 출력될 정보를 결정하는 역할을 합니다.
- 셀 상태에 저장된 정보를 은닉 상태로 얼마나 반영할지를 조절하여, 최종적으로 출력되는 값을 결정합니다.

수식:

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

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

- o t : 시점 t에서의 출력 게이트 값
- h $t-1$:는 이전 시점의 은닉 상태
- x_t : 현재 시점의 입력
- W_o : 출력 게이트의 가중치 행렬
- σ : 시그모이드 함수
- b_o : 출력 게이트의 편향

역할:
출력 게이트는 셀 상태 C t 의 정보를 어느 정도 은닉 상태로 내보낼지를 결정합니다.
셀 상태는 전체 정보가 저장된 곳이며, 이 정보 중 일부를 현재 시점에서 출력으로 내보냅니다.
출력 값은 은닉 상태 h t 를 통해 표현되며, 다음 수식으로 계산됩니다.

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

o_t 는 출력 게이트의 값으로, 셀 상태를 얼마나 반영할지를 조정합니다.

2-2 셀 상태(Cell State)

- LSTM의 셀 상태(C_t)는 시퀀스의 중요한 정보를 저장하는 핵심 메모리입니다.
- 셀 상태는 LSTM의 장기 메모리 역할을 하며, 필요할 때만 업데이트되므로 정보를 오래

유지할 수 있습니다.

- 셀 상태는 망각 게이트와 입력 게이트의 조절을 받으며, 정보가 점차 업데이트되거나 삭제됩니다.

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

기존 셀 상태 C_{t-1}은 망각 게이트에 의해 결정된 비율만큼 유지됩니다.

3. LSTM의 작동 원리

LSTM은 시간 단계 t에서 다음과 같은 순서로 작동합니다:

- 망각 게이트(Forget Gate): 셀 상태에서 어떤 정보를 잊을지 결정합니다.
- 입력 게이트(Input Gate): 새로운 정보를 선택하고, 셀 상태에 얼마나 반영할지를 결정합니다.
- 셀 상태 업데이트: 이전 셀 상태와 새로운 정보를 결합하여 현재 셀 상태를 업데이트합니다.
- 출력 게이트(Output Gate): 셀 상태를 기반으로 은닉 상태를 계산하고, 이를 출력으로 전달합니다.

4. LSTM의 주요 특징과 장점

1) 장기 의존성 문제 해결

LSTM은 기울기 소실 문제를 해결하고, 장기 의존성을 처리할 수 있습니다.

일반적인 RNN은 먼 과거의 정보를 잘 기억하지 못하는 반면, LSTM은 필요한 정보를 오랜 시간 유지하면서도 불필요한 정보를 버릴 수 있습니다.

2) 게이트를 통한 정보 제어

LSTM의 게이트 구조는 정보를 선택적으로 기억하거나 잊을 수 있도록 설계되었습니다.

이를 통해 모델은 중요한 정보만 유지하고, 학습 과정에서 중요한 부분만 반영할 수 있습니다.

5. LSTM 실습 예시

아래는 TensorFlow를 사용하여 LSTM을 활용한 간단한 시계열 데이터 예측 모델의 예시

python 실습 사례 : LSTM

6. LSTM의 한계와 개선 모델

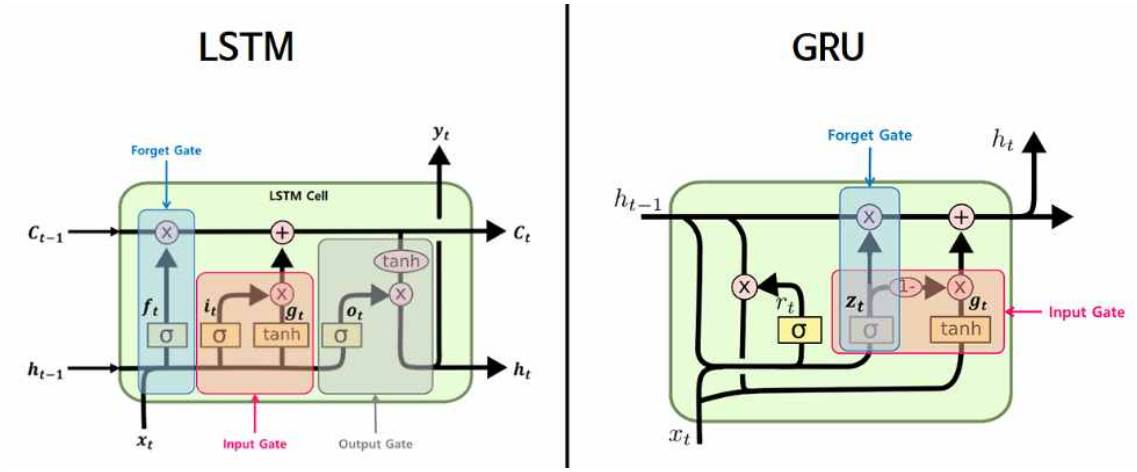
1) 계산 복잡성

LSTM은 RNN보다 복잡한 구조를 가지고 있어, 계산 비용이 큽니다. 특히 대규모 데이터나 긴 시퀀스를 처리할 때는 학습 시간이 오래 걸립니다.

2) GRU(Gated Recurrent Unit) 변형

○ GRU(Gated Recurrent Unit)

- GRU(Gated Recurrent Unit)는 순환 신경망(RNN)의 변형 모델
- LSTM(Long Short-Term Memory)과 유사한 구조를 가지고 있지만, 좀 더 간단하게 설계된 모델
- GRU는 LSTM처럼 장기 의존성(Long-term dependencies) 문제를 해결하기 위해 고안
  LSTM에 비해 계산량이 적고 학습 속도가 빠릅니다.



1. GRU의 기본 개념

- GRU는 게이트 메커니즘을 사용하여 RNN의 기울기 소실 문제(Vanishing Gradient Problem)를 해결합니다.
- GRU는 LSTM과 비교하여 게이트 수를 줄이고 메모리 셀 없이 은닉 상태를 직접 관리하는 방식으로, 더 간단한 구조로 설계되었습니다.
- GRU는 입력 게이트와 망각 게이트의 역할을 업데이트 게이트와 리셋 게이트로 통합하여 정보 흐름을 제어합니다.

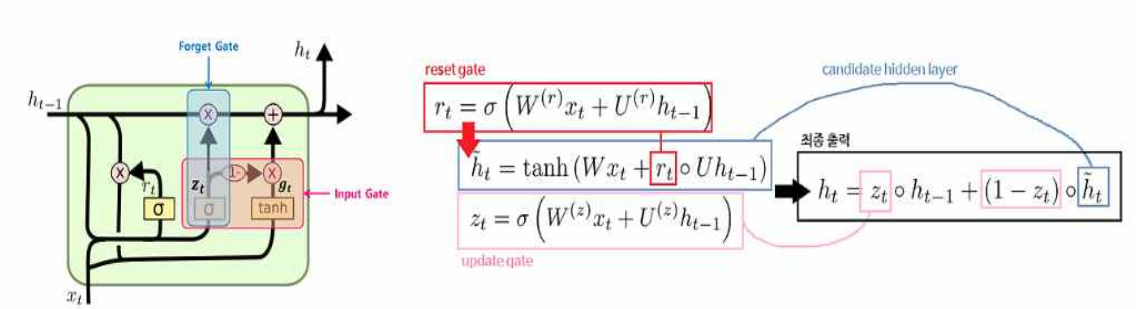
GRU의 주요 특징:

- 게이트 구조 간소화: LSTM의 세 개의 게이트(입력, 망각, 출력 게이트) 대신, GRU는 업데이트 게이트와 리셋 게이트라는 두 개의 게이트로 정보를 제어합니다.
- 단일 은닉 상태: LSTM의 셀 상태와 은닉 상태를 통합하여 하나의 은닉 상태로 관리합니다.
- 더 빠른 학습: GRU는 계산량이 적고, 학습 속도가 LSTM보다 빠르며, 성능 또한 비슷한 경우가 많습니다.

2. GRU의 구조

GRU는 두 개의 주요 게이트인 업데이트 게이트(Update Gate)와 리셋 게이트(Reset Gate)를 사용합니다.

1) 리셋 게이트 (**Reset Gate**)



리셋 게이트는 과거 정보를 얼마나 무시할지 결정하는 게이트로, 새로운 입력과 관련된 정보를 더 많이 반영할 수 있도록 합니다. 이를 통해, 이전 시점의 정보가 불필요하다면 이를 잊어버리도록 돕습니다.

==> Reset gate는 short-term dependency를 의미

- r_t : 리셋 게이트의 출력 값

값이 0에 가까우면 이전 은닉 상태를 무시하고, 값이 1에 가까우면 이전 상태를 완전히 유지합니다.

- W_r : 리셋 게이트의 가중치 행렬.
- h_{t-1} : 이전 시점의 은닉 상태
- x_t ; 현재 시점의 입력
- b_r : 리셋 게이트의 편향
- σ : 시그모이드 함수로, 0과 1 사이의 값을 출력합니다.

리셋 게이트는 현재 입력 x_t 와 결합할 이전 은닉 상태의 비율을 조절하며, 특정 정보가 현재 시점에서 필요하지 않다면 이를 초기화(무시)합니다.

2) 업데이트 게이트 (**Update Gate**)

- 이는 현재 시점의 정보와 이전 시점의 은닉 상태를 얼마나 업데이트할지 결정하는 게이트입니다.
- 업데이트 게이트는 기억 유지와 새로운 정보 업데이트 간의 균형을 조절하는 역할을 합니다.

수식:

업데이트 게이트는 LSTM의 입력 게이트와 망각 게이트를 결합한 역할을 합니다.

$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t] + b_z)$$

z_t : 업데이트 게이트에서 출력된 값으로, 0에서 1 사이의 값입니다.

이 값은 이전 은닉 상태 h_{t-1}와 현재 입력 x_t를 얼마나 유지할지를 결정합니다.

값이 1에 가까울수록 이전 정보를 더 유지하고, 0에 가까울수록 이전 정보를 덜 유지합니다.

==> Update gate는 long-term dependency를 의미

- W_z : 업데이트 게이트의 가중치 행렬
- h_{t-1} ; 이전 시점의 은닉 상태
- x_t : 현재 시점의 입력
- b_z : 업데이트 게이트의 편향

업데이트 게이트 z_t 는 새로운 정보 $\tilde{h}_t$ 와 이전 은닉 상태 h_{t-1} 사이에서 가중 평균을 취합니다.

z_t 가 1에 가까울수록 새로운 정보를 많이 반영하고, 0에 가까울수록 이전 정보를 유지합니다.

3) 새로운 은닉 상태 (Candidate Hidden State)

- 리셋 게이트의 결과를 바탕으로 새로운 은닉 상태를 계산합니다.
- 현재 시점의 입력과 이전 은닉 상태를 사용하여 새로운 후보 은닉 상태를 만듭니다.

리셋 게이트 r_t 를 사용해 이전 은닉 상태 h_{t-1} 의 일부만을 고려한 새로운 상태 $\tilde{h}_t$ 를 계산합니다.

$$\tilde{h}_t = \tanh(W_h \cdot [r_t * h_{t-1}, x_t] + b_h)$$

최종 은닉 상태 : 업데이트 게이트로 은닉 상태 업데이트

최종적으로, 업데이트 게이트 z_t를 사용하여 이전 은닉 상태 h_{t-1}와 새로운 후보 은닉 상태 \tilde{h}_t를 혼합하여 최종 은닉 상태를 계산합니다.

$$h_t = z_t * h_{t-1} + (1 - z_t) * \tilde{h}_t$$

여기서 h_t는 현재 시점의 최종 은닉 상태입니다.

업데이트 게이트 z_t 를 사용해 이전 은닉 상태와 새로운 상태를 가중 합산하여 은닉 상태를 결정합니다.

3. GRU의 작동 원리

- 리셋 게이트는 이전 은닉 상태의 정보를 얼마나 유지할지를 결정합니다.
- 이를 통해 이전 정보가 새로운 입력과 얼마나 연관이 있는지 조정할 수 있습니다.
- 업데이트 게이트는 현재 시점에서의 새로운 정보를 얼마나 반영할지를 결정합니다. 이는 이전 정보와 새로운 정보 사이의 균형을 맞추는 역할을 합니다.
- 새로운 은닉 상태는 리셋 게이트의 조정 후 계산되며, 최종 은닉 상태는 업데이트 게이트를 사용하여 이전 은닉 상태와 결합됩니다.

4. GRU와 LSTM의 차이점

특징	LSTM	GRU
게이트 수	3개: 입력 게이트, 망각 게이트, 출력 게이트	2개: 리셋 게이트, 업데이트 게이트
셀 상태	셀 상태와 은닉 상태 두 개의 상태를 유지	은닉 상태 하나만 관리
구조적 복잡성	복잡함 (3개의 게이트와 셀 상태)	단순함 (2개의 게이트, 셀 상태 없음)
학습 속도	상대적으로 느릴 수 있음	구조가 단순하여 상대적으로 빠름
장기 의존성 처리	게이트가 세분화되어 복잡한 장기 의존성 문제를 처리할 수 있음	LSTM과 비슷한 성능을 보이지만, LSTM보다는 단순한 의존성 처리에 더 적합
메모리 및 계산 비용	셀 상태를 포함하고 있어 더 많은 메모리와 계산 비용이 필요함	더 적은 메모리와 계산 비용을 요구함
실제 성능 차이	복잡한 시퀀스 데이터에서 더 나은 성능을 보이는 경우가 많음	간단한 데이터에서 성능이 비슷하거나 더 나은 경우도 있음

5. GRU의 장점/단점

단순한 구조로 인해 LSTM보다 학습 속도가 빠르며 계산량이 적습니다.
특정 문제에서 LSTM과 유사한 성능을 발휘하면서 더 적은 자원을 사용합니다.

LSTM에 비해 단점:
더 복잡한 시퀀스나 장기 의존성을 다루는 경우, LSTM이 더 나은 성능을 보일 수 있습니다. LSTM은 더 세밀한 정보 제어가 가능하기 때문에, 특정 문제에서는 LSTM이 더 적합할 수 있습니다.

6. GRU 실습 예시

다음은 TensorFlow를 사용하여 간단한 GRU 모델을 구축하는 예시입니다. 이 모델은 시계열 데이터를 학습하여 다음 값을 예측하는 작업을 수행합니다.

6. 결론

GRU(Gated Recurrent Unit)는 LSTM과 비슷하게 순차 데이터에서 장기 의존성을 처리할 수 있는 능력을 가지고 있으며, 구조가 더 단순해 계산 비용이 적고 학습 속도가 빠르다는 장점이 있습니다.

○ RNN & LSTM 적용사례

▶ RNN(Recurrent Neural Network) 사례

- RNN(Recurrent Neural Network) 모델은 순차 데이터나 시간에 따른 변화를 다루는 문제에서 매우 강력한 성능을 발휘합니다.
- RNN은 입력 데이터의 순서를 고려하여 처리할 수 있어, 텍스트 데이터, 음성 신호, 시간 시계열 데이터 등 다양한 분야에서 중요한 역할을 합니다.

1. 자연어 처리 (Natural Language Processing, NLP)

RNN은 텍스트 데이터를 처리하는 데 매우 적합합니다. 특히, 텍스트의 순차적인 특성을 학습하고, 문맥을 이해하는 데 사용됩니다.

RNN은 단어 간의 관계를 이해하고, 다음 단어를 예측하거나, 문장의 의미를 분석하는 데 탁월한 성능을 발휘합니다.

사례:

- 언어 모델링(Language Modeling): RNN은 다음 단어 예측에 자주 사용됩니다. 예를 들어, 입력 문장의 앞부분을 기반으로 다음에 올 단어를 예측하는 모델입니다. 구글의 스마트 키보드 제안, 자동 완성 기능 등이 RNN을 기반으로 작동합니다.
- 기계 번역(Machine Translation): Google Translate와 같은 기계 번역 시스템은 RNN을 사용하여 하나의 언어에서 다른 언어로 변환합니다. 예를 들어, 영어 문장을 한국어 문장으로 변환할 때, RNN이 문맥을 이해하여 각 단어를 번역하는 데 사용됩니다.
- 텍스트 생성(Text Generation): RNN을 사용하면 시나리오나 소설과 같은 긴 텍스트를 자동으로 생성할 수 있습니다. RNN은 주어진 주제나 스타일을 기반으로 새로운 텍스트를 생성하는 데 유용합니다.

2. 음성 인식 (Speech Recognition)

RNN은 음성과 같은 연속적인 신호 데이터를 처리하는 데도 널리 사용됩니다.

RNN은 시간에 따른 음성 데이터를 학습하여, 말하는 내용을 텍스트로 변환하거나, 특정 패턴을 인식할 수 있습니다.

사례:

- Apple Siri 및 Google Assistant: 음성 인식 시스템에서 RNN 기반 모델을 사용하여 사용자의 음성을 텍스트로 변환합니다. RNN이 시간에 따른 음성 신호를 분석하여, 그 내용을 파악하고 명령을 수행할 수 있습니다.
- 자동 자막 생성: YouTube와 같은 플랫폼에서는 음성 데이터를 실시간으로 분석하여 자동 자막을 생성하는 데 RNN을 사용합니다. 음성 신호를 분석하고 이를 텍스트로 변환하는 작업을 실시간으로 처리합니다.

3. 감정 분석 (Sentiment Analysis)

RNN은 텍스트 감정 분석에서 사용됩니다.

사용자의 리뷰나 소셜 미디어 포스트의 긍정적, 부정적 감정을 파악하거나, 더 복잡한 감정 상태를 분석하는 데 적합합니다.

RNN은 텍스트의 순서를 고려하여, 문장의 전체적인 맥락을 파악할 수 있습니다.

사례:

- 영화 리뷰 분석: 영화 리뷰 데이터를 RNN을 통해 분석하여, 사용자가 긍정적 또는 부정적인 감정을 표현하는지 파악할 수 있습니다. 예를 들어, 영화 리뷰의 텍스트를 입력받아 영

화에 대한 전반적인 평점을 예측할 수 있습니다.

- 소셜 미디어 감정 분석: 트위터와 같은 소셜 미디어 플랫폼에서 RNN을 사용하여 트윗의 감정을 분석할 수 있습니다. 예를 들어, 특정 이슈에 대한 트윗을 수집해 해당 이슈에 대한 대중의 감정(긍정/부정)을 분석하는 데 사용됩니다.

4. 시계열 데이터 예측 (Time Series Prediction)

RNN은 시계열 데이터를 분석하고, 이를 바탕으로 미래 값을 예측하는 데 매우 유용합니다. 주식 시장, 날씨 예측, 경제 데이터 등의 데이터를 처리할 때, RNN은 시간 순서에 따른 패턴을 학습하여 다음 값들을 예측할 수 있습니다.

사례:

- 주식 가격 예측: RNN은 과거 주식 데이터를 분석하여, 다음 날 주식 가격을 예측하는 데 사용됩니다. 이 모델은 시계열 데이터를 바탕으로 시장의 트렌드를 학습하고, 미래 주식 가격을 예측합니다.
- 날씨 예측: RNN은 기상 데이터를 바탕으로 미래의 날씨 패턴을 예측하는 데 사용됩니다. 온도, 습도, 기압 등 시간에 따른 변화 데이터를 입력으로 받아 다음날 날씨를 예측할 수 있습니다.

수요 예측: 유통업체에서는 RNN을 사용하여 재고 수요를 예측할 수 있습니다. 과거의 판매 데이터를 학습하여 특정 제품의 미래 수요를 예측함으로써, 재고 관리에 도움이 됩니다.

5. 비디오 처리 (Video Processing)

RNN은 비디오 분석에서 각 프레임 간의 연관성을 고려하여, 동영상의 시간적 패턴을 분석할 수 있습니다.

이는 비디오의 중요한 정보나 객체를 감지하고, 동작을 인식하는 데 효과적입니다.

사례:

- 비디오 자막 생성: RNN을 사용하여 비디오에서 자동으로 자막을 생성할 수 있습니다. 비디오의 각 프레임에서 추출된 정보와 시간적 관계를 분석하여 자막을 생성하는 데 사용됩니다.
- 동작 인식: 비디오 속 인물의 동작(걷기, 뛰기, 물체 집기 등)을 RNN이 분석하여, 동작을 인식하거나 동영상 분류에 사용할 수 있습니다. 예를 들어, 운동 영상을 분석해 특정 운동 동작을 인식할 수 있습니다.

6. 기계 번역 (Machine Translation)

RNN은 기계 번역에서 크게 사용됩니다.

문장을 하나의 언어에서 다른 언어로 번역할 때, RNN이 각 단어 간의 문맥을 파악하여 자연스러운 번역을 제공합니다.

특히, 시퀀스-투-시퀀스(Seq2Seq) 모델에서 RNN이 두 언어 간의 문장을 변환하는 데 핵심 역할을 합니다.

사례:

- Google Translate: Google Translate는 Seq2Seq RNN 모델을 사용하여 텍스트를 여러 언어로 번역합니다. 원래 문장의 순차적인 단어를 분석하고, 이를 기반으로 목표 언어로 번역된 문장을 생성합니다.

- 실시간 번역: 회의나 대화 중 실시간으로 번역하는 시스템에서도 RNN이 사용됩니다. 마이크에서 입력된 음성을 분석하고, 해당 언어로 실시간 번역이 가능합니다.

7. 음악 및 텍스트 생성 (Music and Text Generation)
RNN은 음악 생성이나 텍스트 생성 작업에서도 매우 유용합니다.
RNN은 시간 순서에 따라 입력된 데이터를 학습하여, 다음에 올 데이터를 예측하고, 이를 바탕으로 새로운 음악이나 텍스트를 생성할 수 있습니다.
사례:
- 음악 생성: RNN을 사용해 새로운 음악을 생성할 수 있습니다. 예를 들어, 과거의 음악 데이터를 학습한 RNN이 다음에 나올 음표를 예측하여 새로운 곡을 생성할 수 있습니다.
- 시 또는 이야기 생성: RNN을 통해 자동으로 시나 이야기를 생성할 수 있습니다. 주어진 텍스트를 학습하여 비슷한 스타일의 텍스트를 생성합니다.

8. 챗봇 (Chatbot)
RNN은 대화형 챗봇에서 문맥을 이해하고, 대화를 자연스럽게 이어나가는 데 사용됩니다.
사용자의 입력을 분석하고, 그에 맞는 응답을 생성하는 역할을 합니다.
사례:
- 고객 지원 챗봇: RNN 기반 챗봇은 고객의 질문을 이해하고, 적절한 답변을 제공합니다. 고객이 입력한 문장의 의미를 분석하고, 그에 맞는 적절한 응답을 생성합니다.
- 대화형 에이전트: Siri나 Google Assistant와 같은 대화형 에이전트도 RNN을 사용하여 사용자와의 대화를 이어나가고, 사용자의 명령을 이해하고 처리할 수 있습니다.

▶ LSTM(Long Short-Term Memory) 모델

1. 자연어 처리 (Natural Language Processing, NLP)
LSTM은 자연어 처리에서 매우 중요한 역할을 합니다.
문장의 순차적 관계를 학습하고, 긴 문맥을 유지하면서도 단기 의존성을 해결하는 데 매우 유용합니다.
사례:
- 언어 모델링(Language Modeling): LSTM은 문장 내에서 단어 간의 관계를 학습하여, 주어진 단어 시퀀스에서 다음 단어를 예측할 수 있습니다. 예를 들어, 스마트폰의 자동 완성 기능은 LSTM을 사용해 다음에 올 단어를 예측합니다.
- 텍스트 생성(Text Generation): LSTM은 문맥을 학습하여 자연어 텍스트 생성에 사용됩니다. 예를 들어, 주어진 텍스트의 스타일과 문맥을 바탕으로 새로운 문장이나 소설을 생성할 수 있습니다.
- 기계 번역(Machine Translation): LSTM은 하나의 언어에서 다른 언어로 변환하는 기계 번역에서 널리 사용됩니다. LSTM을 통해 긴 문장의 문맥을 유지하면서 자연스러운 번역이 가능해졌습니다. 예를 들어, Google Translate는 LSTM을 사용해 문맥에 맞는 번역을 생성합니다.

2. 음성 인식 (Speech Recognition)

음성 데이터는 시간에 따른 변화가 중요한 순차적 데이터로, LSTM은 음성 신호에서 시간적 패턴을 잘 학습합니다.
LSTM은 장기 의존성을 처리하는 데 강점이 있어, 긴 음성 신호에서 유용하게 사용됩니다.
사례:

- Siri, Google Assistant: 음성 인식 시스템에서 LSTM은 사용자의 음성 데이터를 분석하여, 말한 내용을 텍스트로 변환하는 데 사용됩니다. 이는 실시간 음성 인식, 명령어 처리 등의 작업에서 매우 중요한 역할을 합니다.
- 자동 자막 생성: LSTM은 음성을 분석하여 비디오나 음성 파일에서 실시간 자막을 자동으로 생성하는 데 사용됩니다. YouTube의 자동 자막 생성 기능은 음성 데이터를 분석하여, 이를 텍스트로 변환하는 과정에서 LSTM을 사용합니다.

3. 시계열 데이터 예측 (Time Series Prediction)
LSTM은 시계열 데이터를 처리하는 데 특히 뛰어납니다.
주식 가격, 날씨 예측, 경제 데이터와 같은 시간 순서가 중요한 데이터에서 LSTM은 과거 데이터를 바탕으로 미래 값을 예측할 수 있습니다.
사례:

- 주식 가격 예측: LSTM은 주식 시장 데이터를 분석하여 다음 날의 주식 가격이나 미래의 주식 트렌드를 예측하는 데 사용됩니다. LSTM은 과거 데이터를 장기적으로 학습하여 패턴을 파악하는 데 탁월합니다.
- 날씨 예측: LSTM은 기상 데이터를 바탕으로 미래 날씨를 예측하는 데 사용됩니다. 온도, 습도, 기압 등의 데이터를 분석하여 향후 날씨 변화를 예측하는 데 사용됩니다.
- 수요 예측: LSTM은 과거 판매 데이터를 바탕으로 미래의 제품 수요를 예측하는 데 사용됩니다. 이를 통해 재고 관리를 효율적으로 할 수 있습니다.

4. 비디오 처리 (Video Processing)
LSTM은 비디오 처리와 같은 시퀀스 데이터에서 매우 유용합니다.
비디오의 각 프레임 간의 시간적 연속성을 학습하여 동작 인식이나 행동 예측을 할 수 있습니다.

- 사례:
- 동작 인식: LSTM은 비디오에서 사람의 동작을 인식하는 데 사용됩니다. 예를 들어, 특정 사람이 걷고 있는지, 뛰고 있는지, 앉고 있는지 등을 LSTM이 분석하여 예측할 수 있습니다. 이러한 동작 인식은 스포츠 분석이나 감시 시스템에서 유용하게 사용됩니다.
- 비디오 요약: LSTM은 비디오 데이터를 분석하여 중요한 장면을 요약하는 데 사용할 수 있습니다. 이는 긴 동영상을 짧게 요약하거나, 영화의 핵심 장면만 추출하는 데 유용합니다.

5. 감정 분석 (Sentiment Analysis)
LSTM은 문장 내에서 단어의 순서를 고려하여, 텍스트 데이터에서 감정(긍정, 부정)을 분석하는 데 매우 효과적입니다.
이는 영화 리뷰, 소셜 미디어 글 등에서 사용자의 감정을 파악하는 데 사용됩니다.
사례:

- 리뷰 감정 분석: LSTM을 사용하여 영화, 제품, 음식 리뷰 등의 텍스트에서 긍정적인지, 부정적인지를 분석할 수 있습니다. 예를 들어, 특정 제품 리뷰에서 LSTM이 사용자의 감정

을 파악하여 해당 리뷰가 긍정적 또는 부정적임을 예측할 수 있습니다.

- 트위터 감정 분석: 소셜 미디어 글을 LSTM을 통해 분석하여, 특정 주제에 대해 사람들이 어떻게 느끼는지 파악할 수 있습니다. 예를 들어, 정치적인 이슈에 대한 트윗을 분석해 사람들의 의견을 감정적으로 분류할 수 있습니다.

6. 챗봇 (Chatbot)

LSTM은 대화형 AI 시스템에서 사용되며, 사용자의 입력에 따라 자연스럽게 대화를 이어나가는 데 중요한 역할을 합니다.

LSTM은 대화의 흐름을 유지하면서, 장기적인 문맥을 기억할 수 있습니다.

사례:

- 고객 지원 챗봇: LSTM을 사용한 챗봇은 고객의 질문을 이해하고, 적절한 답변을 제공합니다. LSTM은 고객의 이전 질문을 기억하고, 이를 바탕으로 대화를 자연스럽게 이어갈 수 있습니다.
- 대화형 AI: Google Assistant, Amazon Alexa와 같은 대화형 AI는 사용자의 질문을 이해하고 일관된 답변을 제공하기 위해 LSTM을 사용합니다. 예를 들어, 사용자가 질문을 이어서 했을 때, LSTM은 앞선 질문의 내용을 기억하고 이에 맞는 답변을 제시할 수 있습니다.

7. 음악 생성 (Music Generation)

LSTM은 음악 데이터와 같은 순차적인 데이터를 학습하여, 새로운 음악을 생성하는 데 사용할 수 있습니다.

LSTM은 기존 음악 데이터를 학습하여, 다음에 나올 음표나 멜로디를 예측하고 생성합니다.

사례:

- AI 음악 작곡: LSTM을 사용하여 주어진 멜로디에 맞는 새로운 음악을 자동으로 작곡할 수 있습니다. 예를 들어, 클래식 음악의 패턴을 학습한 LSTM 모델이 새로운 클래식 곡을 생성하는 방식입니다.
- 음악 스타일 학습: 특정 작곡가의 음악 스타일을 학습한 LSTM이 그 스타일에 맞는 새로운 음악을 생성할 수 있습니다. 이는 영화 음악이나 게임 음악을 자동으로 생성하는 데 유용합니다.

8. 기계 번역 (Machine Translation)

LSTM은 언어 간의 문장을 변환하는 기계 번역에서 매우 중요한 역할을 합니다.

특히 시퀀스-투-시퀀스(Seq2Seq) 모델에서 LSTM은 입력 문장의 구조를 이해하고, 이를 기반으로 다른 언어로 자연스럽게 번역할 수 있습니다.

사례:

- Google Translate: LSTM은 Google Translate와 같은 번역 시스템에서 원본 문장의 의미를 기억하고, 번역된 문장을 자연스럽게 생성하는 데 사용됩니다. 이는 문장의 길이와 복잡한 문맥을 처리할 수 있어, 번역의 정확도가 매우 높습니다.
- 실시간 번역 시스템: 회의나 이벤트에서 실시간으로 번역을 제공하는 시스템에서도 LSTM이 사용됩니다. 예를 들어, 실시간 대화를 번역할 때, LSTM은 문맥을 이해하고 자연스러운 번역을 생성합니다.

(각주 용어정리)

주1. 순차 데이터(Sequential Data)

- 순차 데이터(Sequential Data)는 데이터가 시간적 또는 순서적으로 연속성을 가지고 있는 데이터를 의미합니다.
- 이 데이터는 순서가 중요하며, 이전 또는 이후의 데이터가 현재 데이터에 영향을 미칠 수 있습니다.
- 시간, 순서, 또는 문맥에 따라 의존성을 가지므로, 순차 데이터를 처리할 때는 이전의 정보를 활용하여 현재의 정보를 분석하거나 미래의 정보를 예측하는 것이 필요합니다.

1. 순차 데이터의 특징

- 시간 의존성: 데이터가 시간에 따라 발생하며, 과거의 정보가 현재와 미래에 영향을 미칠 수 있습니다. 예를 들어, 주식 가격, 날씨 데이터, 센서 데이터 등이 이에 해당됩니다.
- 순서 중요성: 데이터의 순서가 매우 중요합니다. 데이터의 순서가 바뀌면 의미가 달라질 수 있습니다. 자연어 처리에서 단어의 순서가 문장의 의미를 바꾸는 것과 같은 예시가 있습니다.
- 변동하는 길이: 순차 데이터는 정해진 길이를 가지지 않고, 길이가 변동할 수 있습니다. 예를 들어, 문장의 길이 또는 주식 시장의 데이터 포인트 수는 일정하지 않습니다.

2. 순차 데이터의 예시

1) 시계열 데이터 (Time Series Data)

시계열 데이터는 일정한 시간 간격을 두고 수집되는 데이터입니다. 이러한 데이터는 시간에 따라 변동하며, 과거 데이터가 미래 예측에 중요한 역할을 합니다.

예시:

- 주식 시장 데이터 (주식 가격 변동)
- 날씨 데이터 (온도, 강수량, 바람 속도)
- 센서 데이터 (IoT 장치의 센서값)

2) 자연어 처리 데이터 (Natural Language Processing, NLP)

자연어 처리에서 다루는 데이터는 단어 또는 문장 단위로 순차적 관계를 가집니다. 단어의 순서는 문맥을 형성하며, 문장의 의미를 결정합니다.

예시:

- 텍스트 데이터 (단어의 순서와 문맥)
- 음성 데이터 (음성 신호에서 발음 순서)

3) 음성 데이터

음성 신호는 시간에 따라 변화하는 파형으로 나타나며, 과거의 발음이 현재와 다음 발음에 영향을 미칩니다. 이 순차적 특성을 이용해 음성 인식과 같은 문제를 해결할 수 있습니다.

예시:

- 음성 인식 (발음 순서에 따라 단어 결정)
- 음악 신호 처리 (음악의 리듬과 멜로디 분석)

4) 비디오 데이터

비디오는 연속된 이미지 프레임으로 구성되어 있으며, 각 프레임이 시간에 따라 순차적으로 변화합니다. 동작 인식이나 비디오 분석에 순차 데이터 처리가 사용됩니다.
예시:

- 비디오 스트리밍 (프레임 순서에 따른 영상 처리)
- 동작 인식 (동영상에서의 움직임 분석)

3. 순차 데이터를 처리하는 모델
순차 데이터를 처리하기 위해서는 과거 정보를 기억하거나, 시간적 의존성을 학습하는 모델이 필요합니다. 아래는 순차 데이터를 처리할 때 주로 사용되는 모델들입니다.
1) RNN(Recurrent Neural Network)
RNN은 순차 데이터를 처리하기 위해 고안된 순환 구조를 가진 신경망입니다. RNN은 이전 시점의 은닉 상태를 현재 시점의 입력과 함께 처리하여, 시간적 의존성을 학습할 수 있습니다. 다만, RNN은 기울기 소실 문제로 인해 긴 시퀀스를 처리하는 데 한계가 있습니다.
2) LSTM(Long Short-Term Memory)
LSTM은 기울기 소실 문제를 해결하기 위해 개발된 RNN의 변형 모델로, 장기 의존성을 처리할 수 있습니다. LSTM은 게이트 메커니즘을 통해 정보가 장기간 기억되도록 설계되었습니다.
3) GRU(Gated Recurrent Unit)
GRU는 LSTM과 유사하게 순차 데이터를 처리하며, 구조가 더 간단해 학습 속도가 빠르면서도 성능이 좋은 모델입니다. GRU 역시 장기 의존성을 처리할 수 있습니다.
4) 1D CNN(1-Dimensional Convolutional Neural Networks)
순차 데이터를 처리할 때 1차원 컨볼루션을 사용할 수도 있습니다. 특히 시계열 데이터나 자연어 처리에서 유용하며, 과거 정보를 효율적으로 추출할 수 있습니다.

4. 순차 데이터의 처리 방법
1) RNN을 활용한 순차 데이터 처리
RNN을 사용하면 각 시간 단계의 입력을 처리한 후, 은닉 상태를 통해 이전 시점의 정보를 현재 시점으로 전달하여 순차적 의존성을 학습합니다.
python 코드 복사

```
import tensorflow as tf
from tensorflow.keras import layers, models
# 시계열 데이터 예시 (RNN 적용)
model = models.Sequential()
model.add(layers.SimpleRNN(50, input_shape=(10, 1)))  # 10개의 타임스텝을 가진 데이터
model.add(layers.Dense(1))
model.compile(optimizer='adam', loss='mse')
```

2) LSTM을 활용한 순차 데이터 처리
LSTM은 RNN의 한계인 기울기 소실 문제를 해결하기 위해 개발된 모델로, 장기 시퀀스 데이터를 처리하는 데 매우 효과적입니다.
python 코드 복사

```
model = models.Sequential()
model.add(layers.LSTM(50, input_shape=(10, 1)))  # LSTM을 사용하여 10개의 타임스텝 처리
model.add(layers.Dense(1))
model.compile(optimizer='adam', loss='mse')
```

3) GRU를 활용한 순차 데이터 처리
GRU는 LSTM보다 간단한 구조로 빠른 학습 속도를 제공하며, 성능도 유사하게 좋은 모델입니다.
python 코드 복사

```
model = models.Sequential()
model.add(layers.GRU(50, input_shape=(10, 1)))  # GRU를 사용하여 10개의 타임스텝 처리
model.add(layers.Dense(1))
model.compile(optimizer='adam', loss='mse')
```

5. 순차 데이터 처리의 응용 분야
1) 자연어 처리(NLP)
텍스트 생성, 기계 번역, 감정 분석, 요약 생성 등에서 RNN, LSTM, GRU 모델을 사용하여 문맥 정보를 학습할 수 있습니다.
2) 음성 인식
음성 신호는 시간적 순서를 가지므로, RNN 기반의 모델을 사용하여 음성을 텍스트로 변환하는 작업에 사용됩니다.
3) 시계열 예측
주식 시장 예측, 날씨 예측, 센서 데이터 분석 등에서 과거 데이터를 기반으로 미래 값을 예측하는 데 순차 데이터 처리가 필수적입니다.
4) 비디오 처리
비디오 프레임 사이의 연속적인 변화와 동작을 분석하는 작업에 순차 데이터를 처리하는 모델이 사용됩니다.

주석2 처리 시간에 의존하는 데이터(Time-Dependent Data)

- 처리 시간에 의존하는 데이터(Time-Dependent Data)는 시간의 흐름에 따라 변하는 시계열 데이터나 순차 데이터를 의미합니다.
- 이러한 데이터는 과거의 상태가 현재 및 미래의 상태에 영향을 미치며, 시간적 순서를 유지한 채로 처리해야 유의미한 결과를 도출할 수 있습니다.
- 처리 시간에 의존하는 데이터는 다양한 분야에서 발견되며, 적절한 분석 기법을 사용해 시간적 상관관계를 파악하는 것이 중요합니다.

특징:

- 시간의 흐름에 따라 데이터가 달라지고, 시간 자체가 데이터의 중요한 변수로 작용합니다.
- 시간적 상관관계가 존재하여, 과거의 상태가 미래에 영향을 줍니다.
- 데이터의 시간 간격이 일정하거나 불규칙할 수 있으며, 시간 축을 기반으로 데이터 분석이 이루어집니다.

주요 예시

1. 시계열 데이터 (Time Series Data)

시계열 데이터는 일정한 시간 간격으로 수집된 데이터로, 시간 순서가 중요한 특징입니다. 주로 미래를 예측하거나, 시간에 따른 패턴을 분석할 때 사용됩니다.

예시:

- 주식 가격 데이터: 주식 시장에서 과거 주식 가격을 기반으로 미래 주가를 예측하는 데이터. 주식 가격은 시계열 데이터로, 과거 가격 패턴이 미래 가격에 영향을 줄 수 있습니다.
- 기상 데이터: 과거의 온도, 습도, 기압 등 기상 데이터를 바탕으로 미래 날씨를 예측하는 데 사용됩니다.
- 경제 지표: GDP, 실업률, 물가상승률 등 경제 데이터는 시계열 데이터로, 이를 통해 경제의 추세와 예측이 가능해집니다.

2. 음성 데이터 (Speech Data)

음성 데이터는 시간에 따라 연속적으로 변화하는 음파 신호로, 단어의 발음 순서나 음성의 흐름이 중요합니다. 음성 인식, 음성 합성과 같은 작업에서는 이러한 시간 의존성이 큰 역할을 합니다.

예시:

- 음성 인식(Speech Recognition): 음성 데이터를 입력받아 이를 텍스트로 변환하는 시스템에서, 음성의 시간적 패턴을 학습하여 단어와 문장을 인식합니다.
- 음성 합성(Text-to-Speech): 텍스트를 입력받아 음성으로 변환하는 과정에서 시간에 따른 발음의 순서와 톤이 중요하게 작용합니다.

3. 자연어 처리 (Natural Language Processing, NLP)

자연어 처리에서는 단어의 순서가 매우 중요합니다. 문맥을 이해하고 단어 간의 의미 관계를 파악하기 위해서는 순차적으로 처리되어야 합니다.

예시:

- 언어 모델링: 주어진 문맥에서 다음에 나올 단어를 예측하는 작업으로, 이전 단어들이 이후 단어에 영향을 미칩니다.
- 기계 번역: 문장의 단어 순서를 파악하고 원문과 번역문 간의 관계를 유지하는 작업에서 시간적 의존성이 필요합니다.
- 텍스트 생성: 주어진 문장을 바탕으로 새로운 문장을 생성하는 경우, 문장의 흐름과 단어의 순서가 중요합니다.

4. 비디오 및 이미지 시퀀스 데이터

비디오 데이터는 연속된 프레임들로 이루어져 있으며, 각 프레임 간의 시간적 변화를 고려해야 의미 있는 결과를 얻을 수 있습니다. 예를 들어, 물체의 움직임이나 사람의 행동을 인식하는 데 시간에 따른 변화가 중요한 역할을 합니다.

예시:

비디오 분석: 특정 객체가 시간에 따라 어떻게 움직이는지를 분석하거나, 사람의 행동 인식을 하는 데 사용됩니다.
비디오 생성 및 예측: 현재 프레임을 바탕으로 미래의 프레임을 예측하거나, 비디오 데이터를 통해 새로운 동영상을 생성하는 작업에서 시간 순서가 중요한 역할을 합니다.
5. 생리 신호 데이터 (Physiological Signal Data)
생체 신호 데이터는 주로 시간에 따라 변하는 심장 박동, 뇌파, 호흡 패턴 등을 포함합니다. 이러한 데이터는 의료 진단이나 건강 상태 추적에서 중요한 역할을 합니다.
예시:

- 심전도(ECG) 데이터: 환자의 심장 박동 패턴을 시간에 따라 측정하여 심장 상태를 진단하는 데 사용됩니다.
- 뇌파(EEG) 데이터: 뇌 활동을 측정하기 위해 시간에 따른 전기 신호를 분석하여 신경 활동을 추적합니다.

6. 금융 거래 데이터
금융 데이터는 시간에 따라 변화하는 거래 정보로, 주식, 채권, 외환 등 금융 자산의 가격 변동을 분석하는 데 사용됩니다. 이 데이터는 실시간으로 변화하므로, 시간 순서가 중요한 요소입니다.
예시:

- 알고리즘 거래: 과거 거래 데이터와 실시간 가격 정보를 기반으로, 자동으로 주식을 매매하는 시스템이 시간 의존적인 금융 데이터를 활용합니다.
- 리스크 관리: 금융 기관에서 미래의 시장 위험을 예측하고 관리하기 위해 시계열 데이터를 분석합니다.

7. IoT 센서 데이터
사물 인터넷(IoT) 데이터는 센서에서 수집된 데이터로, 시간에 따라 변화하는 패턴을 분석해야 의미 있는 정보를 얻을 수 있습니다.
예시:

- 스마트 홈: 스마트 디바이스에서 수집한 온도, 습도, 에너지 사용량 등의 데이터를 시간에 따라 분석하여 효율적인 에너지 관리와 자동화된 제어를 가능하게 합니다.
- 스마트 시티: 교통량, 대기질, 에너지 사용 등의 데이터를 실시간으로 모니터링하여 교통 흐름 최적화 및 도시 자원 관리를 최적화할 수 있습니다.

8. 자율 주행 및 차량 데이터
자율 주행 차량은 주행 중에 주변 환경을 실시간으로 감지하고 분석해야 합니다. 시간에 따라 변화하는 도로 상황, 차량의 위치 및 속도 정보는 안전한 주행을 위한 중요한 정보입니다.
예시:

- 자율 주행 차량의 센서 데이터: 차량의 카메라, 라이다, 레이더 등이 시간에 따라 실시간 데이터를 수집하며, 이러한 데이터를 통해 주행 경로를 예측하고 안전한 운전을 할 수 있도록 돕습니다.
- 차량 내 데이터 분석: 차량의 속도, 연료 소비량, 위치 데이터를 시간에 따라 분석하여 운전 패턴을 분석하거나 연료 효율성을 최적화할 수 있습니다.

특징	처리 시간에 의존하는 데이터	순차적 데이터
시간의 역할	시간이 중요한 변수로 작용하며, 데이터가 시간의 흐름에 따라 변화	순서는 중요하지만 시간이 직접적으로 중요한 요소는 아님
시간적 상관관계	과거 데이터가 현재와 미래 데이터에 영향을 미침	데이터 간의 순서가 의미를 결정하며, 순서가 바뀌면 의미가 달라짐
시간 간격	일정하거나 불규칙한 시간 간격으로 발생하는 데이터	시간 간격이 반드시 일정할 필요는 없으며, 순서가 중요
예시	주식 가격, 기상 데이터, 환자 생체 신호	텍스트 데이터, DNA 서열, 사용자 행동 로그
주요 분석 방법	시계열 분석, 예측 모델	순차적 처리 모델 (예: RNN, LSTM), 자연어 처리 (NLP)

주4. Hidden State (은닉 상태)

Hidden State(은닉 상태)와 Hidden Layer(은닉층)는 둘 다 인공 신경망에서 중요한 개념이지만, 서로 다른 역할과 의미를 가지고 있습니다.
이 두 용어는 특히 순환 신경망(RNN, LSTM, GRU)과 기본 신경망(MLP 등)에서 각각 다른 맥락에서 사용됩니다.
1. Hidden State(은닉 상태)
Hidden State(은닉 상태)는 순환 신경망(RNN, LSTM, GRU)에서 사용되는 개념입니다.
이는 네트워크가 시퀀스 데이터를 처리할 때, 이전 시점의 정보를 현재 시점의 정보와 결합하여 다음 시점으로 전달하는 역할을 합니다. 시간적 의존성을 유지하고, 과거의 정보를 기억하며 학습할 수 있도록 하는 것이 은닉 상태의 주요 기능입니다.
특징:

- 시간에 의존적인 정보: Hidden State는 이전 시점의 은닉 상태와 현재 시점의 입력을 결합하여, 다음 시점으로 연속적으로 정보를 전달합니다.
- 순차 데이터 처리: 순차 데이터, 시계열 데이터, 자연어 처리 등에서 과거의 정보를 기억하고, 이를 현재 상태와 결합하여 시퀀스의 흐름을 유지하는 역할을 합니다.
- 매 시점마다 갱신: 각 시간 단계에서 은닉 상태가 갱신되며, 현재 시점의 입력 및 이전 시점의 정보를 바탕으로 계산됩니다.

사용처:
RNN, LSTM, GRU: 순환 신경망에서 시간적 의존성을 처리하고, 정보를 순차적으로 전달하는 역할을 합니다.
예시:

- 자연어 처리(NLP): 문장에서 이전 단어의 의미가 현재 단어의 예측에 영향을 미칠 때, 은닉 상태는 문맥 정보를 전달하는 역할을 합니다.
- 시계열 데이터: 주식 가격 예측에서, 과거 주식 가격의 변화가 현재 및 미래 예측에 중요한 역할을 할 때 은닉 상태가 정보를 전달합니다.

2. Hidden Layer(은닉층)

- Hidden Layer(은닉층)는 기본 인공 신경망(MLP, CNN 등)에서 사용되는 개념으로, 입력층과 출력층 사이에 존재하는 층입니다.
- 은닉층은 네트워크가 입력 데이터의 복잡한 패턴을 학습하고, 그 결과를 출력층에 전달하는 역할을 합니다.
- 각 은닉층의 뉴런은 가중치와 활성화 함수를 사용하여 입력 데이터를 변환합니다.

특징:

- 비선형 변환: 은닉층은 입력 데이터를 가중치와 편향을 사용하여 변환하고, 활성화 함수를 적용하여 비선형성을 추가합니다. 이를 통해 모델은 복잡한 패턴을 학습할 수 있습니다.
- 계층적 특징 학습: 네트워크가 여러 개의 은닉층을 사용할 경우, 하위 은닉층에서는 입력 데이터의 기초적인 특징을 학습하고, 상위 은닉층에서는 더 복잡한 특징을 학습하게 됩니다.
- 고정된 상태: 입력 데이터에 대해 한 번 계산된 값을 다음 층으로 전달하며, 은닉 상태처럼 시간에 따라 변하지 않고, 입력 데이터에만 의존합니다.

사용처:

MLP(다층 퍼셉트론), CNN(합성곱 신경망) 등에서 입력 데이터를 처리하는 중간 계산 단계로 사용됩니다.

예시:

- 이미지 분류: CNN에서 이미지 데이터를 처리할 때, 여러 은닉층을 사용하여 엣지(edge), 모양(shape), 객체(object)와 같은 특징을 학습하게 됩니다.
- 다층 퍼셉트론(MLP): 입력 데이터가 은닉층을 거치며 복잡한 연산을 통해 더 높은 수준의 특징을 학습하고, 이를 기반으로 분류나 예측을 수행합니다.

3. Hidden State vs Hidden Layer의 차이점

구분	Hidden State(은닉 상태)	Hidden Layer(은닉층)
개념	순환 신경망(RNN, LSTM, GRU)에서 시간에 따른 정보를 기억하고 전달하는 상태	일반 신경망에서 입력층과 출력층 사이에서 데이터를 처리하는 계층
역할	이전 시점의 정보를 현재 시점과 결합하여 **시퀀스 데이터**를 처리	입력 데이터를 처리하여 **비선형 변환**을 통해 특징을 추출
시간 의존성	시간에 따라 변화하며, 매 시점마다 갱신됨	시간에 의존하지 않으며, 고정된 상태에서 계산된 값을 전달
사용되는 모델	순차 데이터 처리 모델 (RNN, LSTM, GRU)	MLP, CNN, 기본 인공 신경망에서 사용
정보 흐름	각 시점에서 이전 은닉 상태와 현재 입력을 결합하여 다음 시점으로 전달	입력 데이터가 은닉층을 거쳐 출력층으로 전달됨
순차 데이터 처리	시계열, 자연어 처리와 같은 순차적 의존성이 있는 데이터에 적합	순차 데이터 처리에는 사용되지 않으며, 독립적인 데이터 처리에 적합

4. Hidden State와 Hidden Layer의 연결

Hidden State는 순차 데이터의 흐름을 유지하고, 시간적 의존성을 학습하는 데 사용되는 반

면, Hidden Layer는 독립적인 입력 데이터를 처리하고, 이를 출력층에 전달하는 역할을 합니다.
RNN과 같은 순환 신경망에서는 은닉 상태와 은닉층이 함께 사용됩니다. 즉, RNN의 은닉층은 입력 데이터를 처리하는 층이고, 은닉 상태는 그 층에서 시간적으로 변화하는 정보를 유지하는 메커니즘입니다.

(파이썬 실습)
RNN
LSTN . GRU