5. RNN과 LSTM

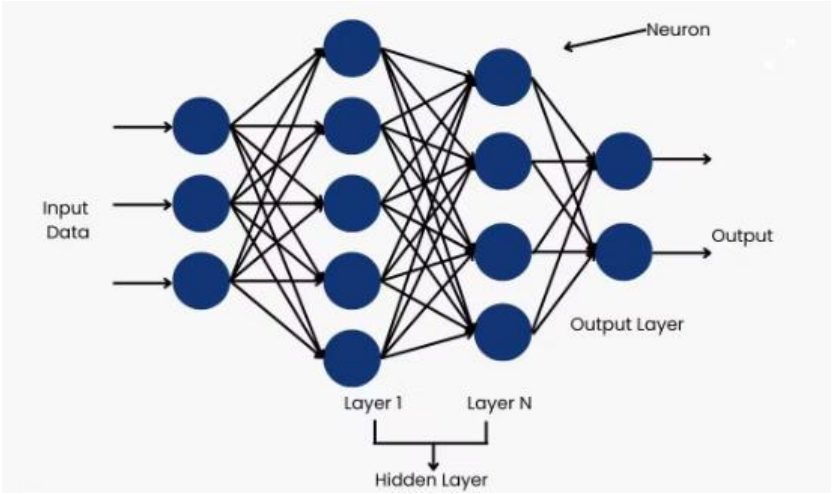
INDEX

- RNN
- LSTM

RNN

다중 퍼셉트론(Multi-Layer Perceptron, MLP) (1/2)

- 인공 신경망(Neural Network) 의 한 종류로, 퍼셉트론을 여러 층으로 쌓은 모델
- 입력층과 출력층 사이에 1개 이상의 은닉층을 가짐
- MLP의 학습 과정
  1. 순전파(Forward Propagation)
  2. 손실 함수 계산(Loss Function)
  3. 역전파(Back Propagation) 및 가중치 업데이트 (Optimization)
  4. 1~3번 반복 후 종료 (Training 완료)



https://www.turing.com/kb/explanation-of-deep-neural-network-multilayer-perceptron-deep-q-network

다중 퍼셉트론(Multi-Layer Perceptron, MLP) (2/2)

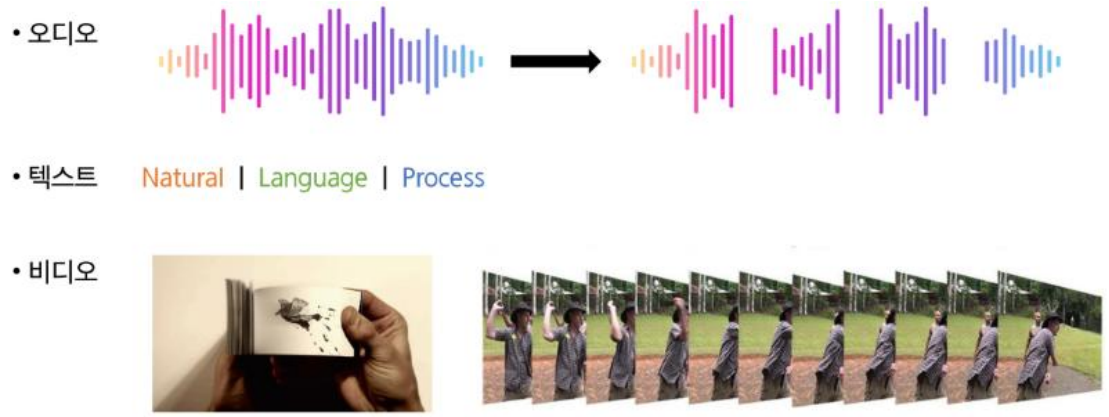
- 은닉층과 비선형 활성화 함수를 사용하여 복잡한 데이터 패턴을 학습할 수 있음
- 완전 연결 구조로써 각 층의 뉴런은 이전 층의 모든 뉴런과 연결되고, 이로 인해 특징 조합을 학습할 수 있음
- 분류와 회귀 등 다양한 문제에 적용할 수 있는 매우 유연한 모델

- 입력 데이터의 순서를 고려하지 못함
  - 예1) I go to scholl 과 School go I to 는 동일한 정보로 처리하여, 단어의 순서에 담긴 의미를 학습할 수 없음
  - 예2) 어제 / 오늘 / 내일의 날씨를 예측할 때, 각 날짜를 개별적인 사건으로 볼 뿐, 시간의 흐름에 따른 연속적인 관계를 학습하지 못함
- 고정된 입출력 크기
  - 모델을 설계할 때 입력과 출력의 크기(차원)이 고정되어야 함
  - 문자열의 길이나 **순차적 데이터**의 길이가 달라지면 처리하기 힘듦

➢ 위의 한계를 극복하기 위해 등장한 것이 "RNN"

[참고] 순차적 데이터(Sequential Data)

- 순서에 의미가 있는 모든 종류의 데이터
- 데이터의 순서가 바뀌면 그 의미나 정보가 완전히 달라지는 데이터
- 특징
  - 순서가 중요
  - 장기 의존성(Logn-term dependency)
    - 과거의 정보가 현재 / 미래에 영향을 준다.
  - 가변 길이
    - 데이터의 길이가 고정되어 있지 않은 경우가 많음
    - ex) 문장의 길이, 음악의 길이

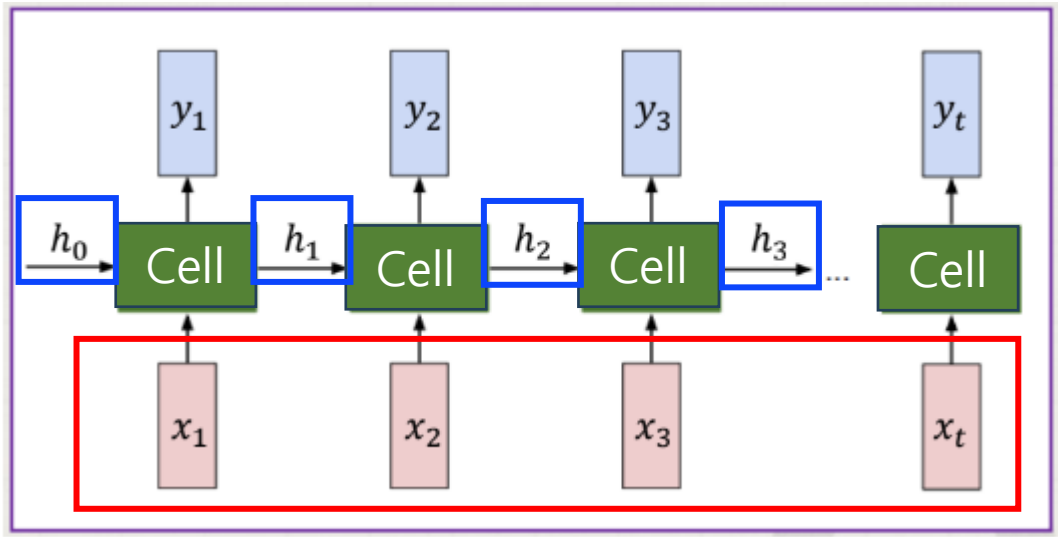


- 대표적인 예시
  - 자연서(텍스트), 시계열 데이터(날씨 데이터, 주식 가격), 오디오 / 비디오 데이터

RNN(Recurrent Neural Network)

- 순환 신경망
- **순서가 있는 데이터**(Sequential Data)를 처리하기 위해 고안된 인공 신경망(Neural Network)
- 과거의 정보를 기억하고, 기억한 정보를 활용해 현재의 데이터를 해석
- "Cell" 이 입력 데이터와 과거의 데이터(은닉 상태) 를 섞어서 "새로운 데이터"를 만들어 냄

과거의 데이터(은닉 상태)
- 이전 시점의 데이터
- 초기에는 [0, 0, 0, 0]
(현재 벡터는 예시)

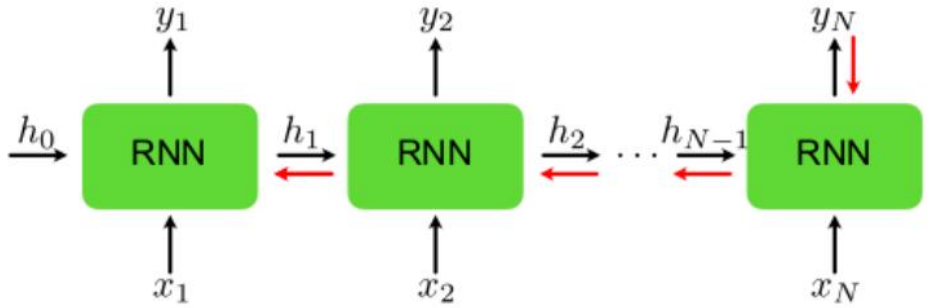


입력 데이터(순차 데이터)
- 인코딩/임베딩이 진행된 벡터가 들어감
- ex) I => [1, 0, 0, 0]

https://velog.io/@nayoong/%EB%94%A5%EB%9F%AC%EB%8B%9D-%EC%88%9C%ED%99%98-%EC%8B%A0%EA%B2%BD%EB%A7%9D

RNN - Cell (1/4)

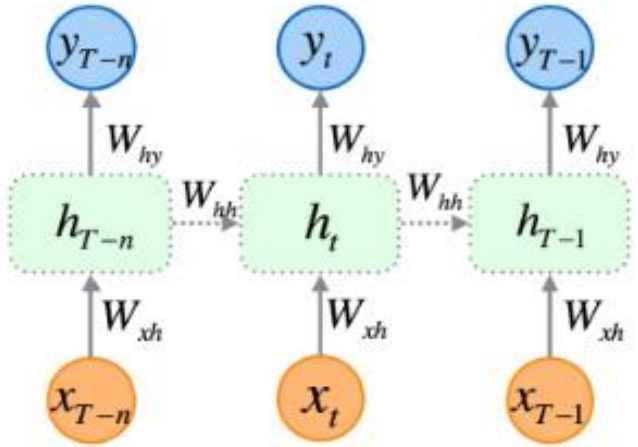
- 과거의 정보와 현재의 입력 데이터를 섞어서 새로운 데이터를 만들어내는 곳 (like 퍼셉트론)
- 과거의 정보를 담아두는 곳이 바로 "은닉 상태(Hidden State)"
- 동작 방식
  1. 셀은 두 가지 데이터를 입력 받음
     - 현재 데이터(x_t)
     - 이전 셀의 은닉 상태(h_{t-1})
  2. 셀 내부에서 두 가지 데이터를 조합하여 새로운 은닉 상태를 만듦(h_t)
  3. 새로운 은닉 상태는 두 곳으로 전달 됨
     - 출력 값(y_t): 현재 시험의 예측 결과 (필요한 경우에 사용)
     - 다음 셀: 다음 순서의 데이터를 처리할 때 참고할 '기억'으로 전달



https://ml-lectures.org/docs/supervised_learning_w_NNs/ml_rnn.html

RNN - Cell (2/4)

- 입력 데이터와 은닉 상태를 조합하는 방법에 대해서 살펴보자.
- 각 입력데이터와 은닉 상태의 영향력을 나타내는 가중치가 존재
  - 입력 가중치 (W_{xh}) : 입력 데이터의 영향력
  - 순환 가중치(W_{hh}) : 이전 기억의 영향력
  - 초기에 해당 값은 랜덤으로 설정 됨

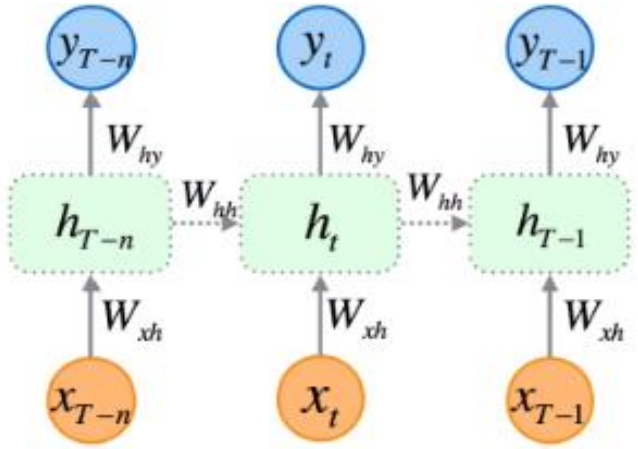


- 이후에 다시 언급하겠지만
  - 모든 입력 가중치는 같은 값
  - 모든 순환 가중치는 같은 값
  - 입력 가중치와 순환 가중치가 같다는 말은 아님!!

https://needjarvis.tistory.com/684

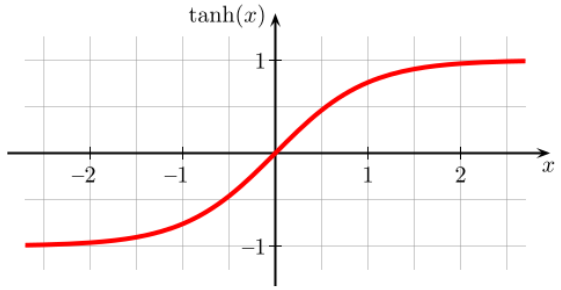
RNN - Cell (3/4)

- 입력 데이터와 은닉 상태 조합의 동작 방식
  1. 셀은 두 가지 데이터를 그대로 받지 않고, 각자의 가중치를 곱해 중요도를 반영
     - 가중치가 반영된 현재 정보 = 현재 데이터(x_t) * 입력 가중치(W_{xh})
     - 가중치가 반영된 과거 기억 = 이전 기억(h_{t-1}) * 순환 가중치(W_{hh})
  2. 두 정보를 합침
     - 조합된 정보 = 현재 정보 + 과거 기억
     - 실제로는 편향(default 값)을 추가로 더하지만, 단순화하기 위해 생략
     - 각 데이터는 벡터(특징)로 표현되며, 더하는 것만으로도 새로운 특징을 만들어 냄
  3. 합쳐진 정보를 활성화 함수(tanh)에 넣어 압축
  4. 새로운 은닉 상태(h_t)를 다음 셀로 전달!



https://needjarvis.tistory.com/684

[참고] 활성화 함수 - tanh

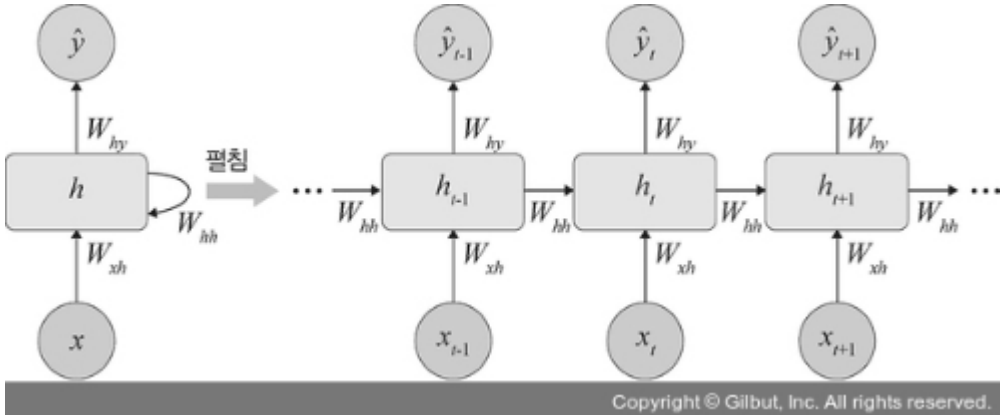


- 입력받은 값을 -1과 1 사이의 값으로 압축시켜주는 역할
- 장점
  1. 출력값을 1 과 -1 사이로 제한하여, 기울기 폭발 방지
     - 이전 단계의 결과에 계속해서 같은 가중치를 곱하는 구조이기 때문에 기울기 폭발 가능성이 있음
  2. 0을 중심으로 대칭이여서, 더 빠르고 안정적인 학습이 가능
     - 학습 과정에서 가중치를 업데이트할 때, 모든 가중치가 같은 방향(모두 증가/모두 감소)으로 움직이는 편향이 줄어듬
- 여전히 기울기 소실이라는 한계가 있음
- RNN에서 다른 활성화 함수를 쓰지 않는 이유
  - ReLU: 양수를 그대로 출력하기 때문에 기울기 폭발이 발생
  - 시그모이드(Sigmoid): 0과 1 사이로 바꿔주면서, 0을 중심으로 하지 않아서 학습이 느리고 장기 기억을 못함

https://wikidocs.net/152159

RNN - Cell (4/4)

- 사실 "셀(cell)은 하나다!"
- 이전에 말한 각각의 입력 데이터, 은닉 상태 가중치가 같은 이유
- 하나의 셀을 재사용하는 이유(가중치 공유)
  - **압도적인 효율성**
    - 입력 데이터가 아무리 길어도, 학습시킬 가중치 세트가 하나임
    - 모델이 매우 가볍고 효율적
  - **좋은 일반화 성능**
    - 각 셀이 정해진 순서에 따른 데이터를 학습하는 것이 아님!! (ex: 첫 번째 단어를 처리하는 방법, 세 번째 단어를 처리하는 방법, …)
    - 어떤 위치의 단어든 이전 기억과 조합하는 일반적인 방법을 학습
    - 훈련 때 학습하지 않은 길이의 문장에도 유연하게 대처할 수 있음

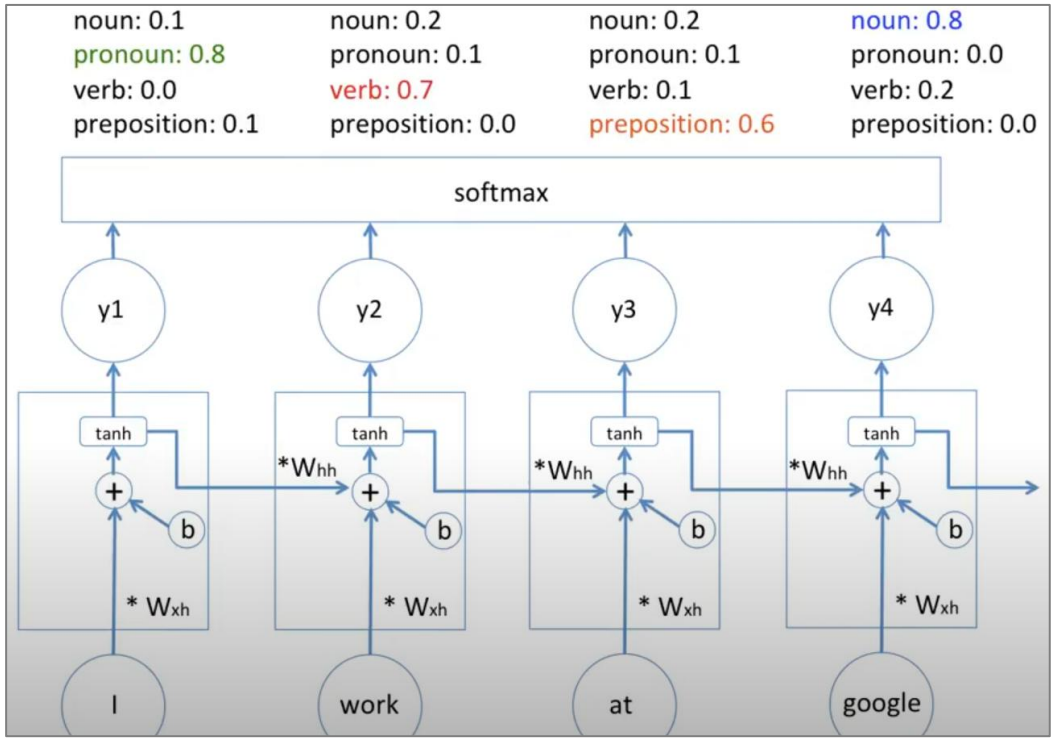


$$h_t = tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

RNN 의 학습 방식 (1/2)

- RNN이 학습하는 과정(오차를 최소로 하는 가중치를 찾아가는 과정)

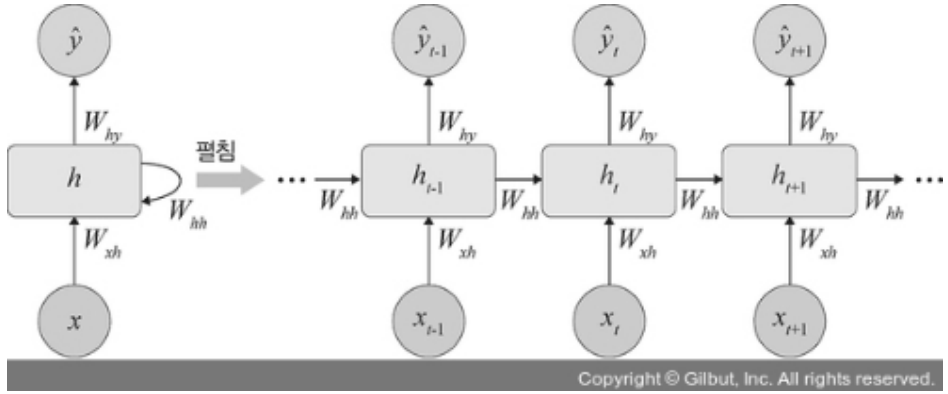
1. 순전파(Forward Propagation)
   - 우리가 배웠던 과정을 그대로 진행
   - 랜덤하게 초기화 된 가중치(W_{xh}, W_{hh}) 를 사용
   - 단어를 순서대로 셀을 통과시키면서 은닉 상태를 업데이트
   - 각 시점에서, 혹은 마지막 시점에서 최종 예측값을 출력
2. 오차 계산
   - 각 문제에 맞는 채점 기준에 맞춰 오차를 계산
   - 예시) 품사 분류기 (지도학습)
   - softmax: 클래스 총합이 1이 되도록, 각 확률을 구함



- 학습 과정에서는 softmax를 통과 후 나온 결과 중 가장 높은 값과 정답을 비교해서 오차를 구함

https://www.youtube.com/watch?v=PahF2hZM6cs

RNN 의 학습 방식 (2/2)



- RNN이 학습하는 과정(오차를 최소로 하는 가중치를 찾아가는 과정)
  1. 역전파(Backpropagation)
     - 정확히는 BPTT(Backpropagation Through Time)

1. 최종 오차의 영향력을 역순으로 추적, 가중치가 오차에 미친 영향력을 구함
2. 셀에는 은닉 상태와 입력 데이터의 가중치가 존재, 해당 시점의 "입력데이터의 가중치"가 오차에 미친 영향력을 구함
   - 우리가 원하는건 가중치를 업데이트하는 것이다.
   - 기존 역전파와 마찬가지로 "입력 데이터"에 따라 영향력(기울기)를 구함
3. 해당 시점에서 최종 오차가 "은닉 상태 가중치"와 "입력 데이터 가중치"로 적절하게 나뉘면, "은닉 상태 가중치의 오차"를 이전 셀로 넘긴다.
4. 2번 ~3번 과정을 첫 번째 시점에 도달할 때까지 반복한다.
5. RNN은 모든 시점에서 동일한 가중치를 사용하기 때문에 모든 시점의 가중치들을 합산하고, 이를 학습률과 결합해 오차가 줄어드는 방향으로 가중치를 업데이트

https://thebook.io/080289/0468/

RNN 의 한계

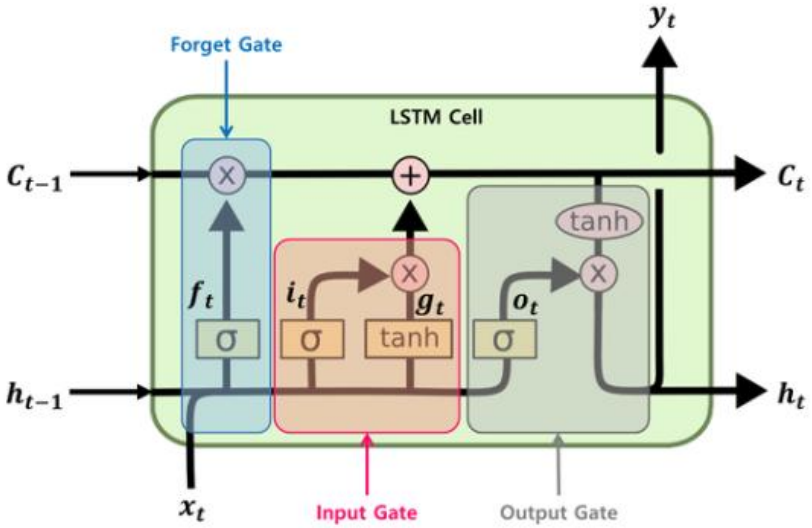
- 기울기 소실 문제(Vanishing Gradient Problem)
  - BPTT(역전파) 과정에서 "이전 정보의 책임" + "현재 시점의 책임"으로 오차가 나눠지면서, 뒤로 갈수록 오차 정보가 거의 전달되지 않음
  - 결국 오래된 정보에 대해서 학습이 제대로 이루어지지 않음
  - ➢ 장기 의존성 문제(Logn-Term Dependency Problem)

- 그리고 이와 같은 장기 의존성 문제를 해결하기 위해서 등장한 것이 "LSTM(Logn Short-Term Memory)"

LSTM

LSTM (Logn Shrot-Term Memory)

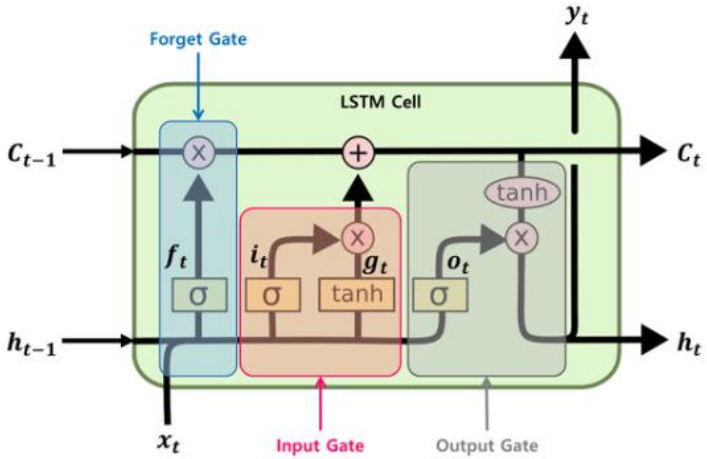
- 이름 그대로 "장단기 기억"
- RNN의 한 종류로, 기존 RNN의 장기 의존성 문제을 해결하기 위해 설계 됨
- "셀 상태(Cell State)와 게이트(Gate)" 라는 독자적인 구조를 도입하여 문제를 해결!
- 기존 RNN
  - 은닉 상태(h_t) 하나만으로 정보를 전달
- LSTM
  - 은닉 상태(h_t) + 셀 상태(c_t) 로 2가지 정보 전달 통로를 활용
  - 셀 상태(c_t) 는 장기 기억을 전달하는 통로
  - 셀 상태(c_t) 는 게이트(Gate) 를 활용하여 장기 기억을 전달할지, 삭제할지를 결정



https://chanjun-kim.github.io/data%EB%B6%84%EC%84%9D/TimeSeries2.html

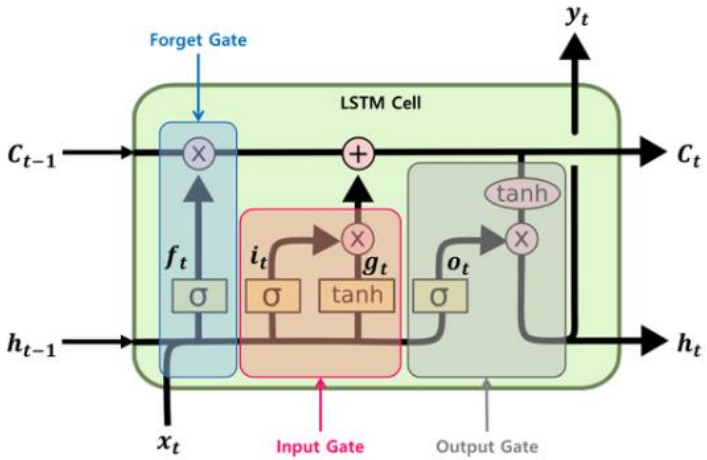
LSTM 구성요소

- 셀 상태(Cell State)
  - 장기 기억 데이터를 전송하는 통로
  - 기존 RNN의 은닉 상태와는 다르게 덮어쓰기를 하지 않고, tanh 함수를 통과하는 등의 복잡한 변환이 일어나지 않음
  - 단순하고 직접적인 경로로써 기울기가 소실되지 않고 시간을 거슬러 올라갈 수 있게 함
- 게이트(Gate)
  - 장기 기억 통로일 뿐인 "셀 상태(Cell State)"를 실질적으로 컨트롤하는 중요한 역할
  - 어떤 정보를 **기억/망각/사용**할지를 '**스스로 학습**'하여 결정
  - 각기 다른 역할을 하는 3개의 게이트가 존재
    - 망각 게이트(Forget Gate)
    - 입력 게이트(Input Gate)
    - 출력 게이트(Output Gate)
    - 모든 게이트는 각각의 가중치(W_f, W_i, W_o)를 가지고 있음



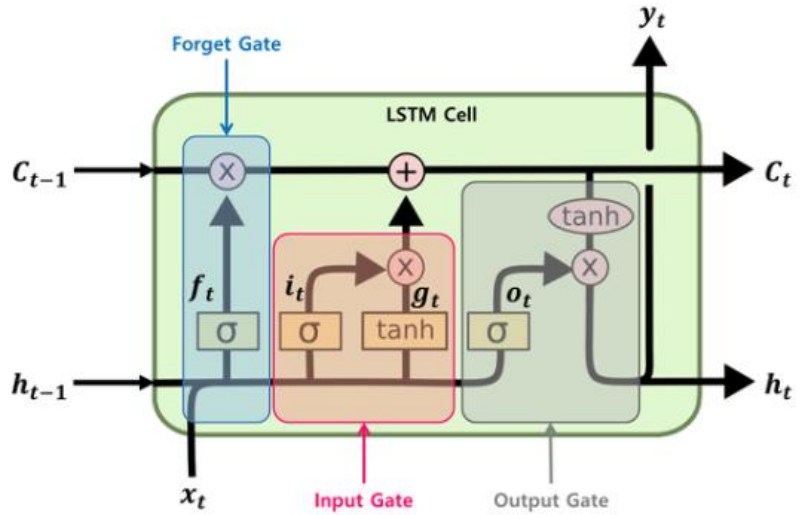
LSTM - 게이트(Gate)

- 장기 기억 통로일 뿐인 "셀 상태(Cell State)"를 실질적으로 컨트롤하는 중요한 역할
- 어떤 정보를 기억/망각/사용할지를 '스스로 학습'하여 결정
- 게이트의 작동 원리
  - 모든 게이트는 **시그모이드 함수(Sigmoid Function)**를 활용 (0 ~ 1 사이의 값으로 바꾸기)
    - 0: 게이트를 완전히 닫아 정보를 차단
    - 1: 게이트를 완전히 열어 모든 정보를 통과
    - 0.5: 게이트를 절반만 열어 정보의 50%만 통과 (0.5 외의 0 ~ 1 사이의 모든 값 가능)
    - 시그모이드를 활용하는 이유: "흐름을 조절하기에 적합한 범위를 출력"
  - (이전 시점의 단기 기억(h_{t-1}) + 현재 시점의 새로운 정보(x_t)) * **각 게이트의 가중치의 값**으로 게이트를 얼마나 열지를 결정



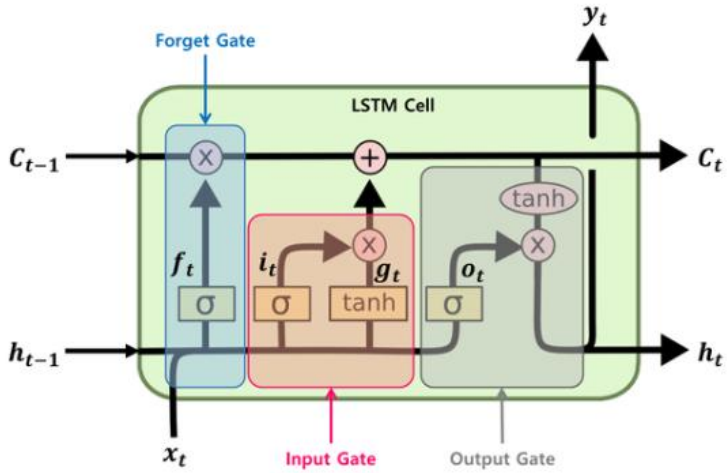
LSTM - 게이트(Gate) - 망각 게이트(Forget Gate)

- "과거의 기억(장기 기억)에서 어떤 걸 지울까?" 를 결정하는 게이트
- 불필요한 장기 기억을 제거하는 역할
- 작동 방식
  1. 이전 시점의 은닉 상태 (h_{t-1}) 과 현재 시점의 입력 (x_t) 를 하나의 벡터로 합침
  2. 망각 게이트의 가중치(Wf) 와 생성한 벡터를 곱함
     - "현재 상황을 고려할 때, 장기 기억이 얼마나 중요한가?"를 평가하는 과정
  3. 시그모이드 함수를 통과시키면서 "망각 벡터"가 생성
  4. 그리고 이 값을 셀 상태(장기 기억) * 망각 벡터 을 통해 셀 상태(장기 기억)을 선택적으로 지움



LSTM - 게이트(Gate) - **입력 게이트(Input Gate)**

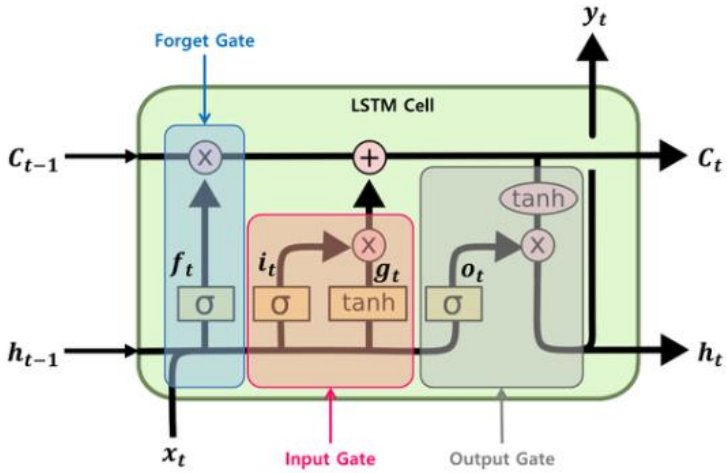
- "새로운 정보 중 어떤 것을 셀 상태(장기 기억)에 저장할까?" 를 결정하는 게이트
- 중요하다고 판단되는 정보만 선별하여 기억을 업데이트하는 역할
- 작동 방식



1. (동시진행)업데이트 벡터 생성 (새로운 정보를 반영할 비율을 결정)
    1. 이전 시점의 은닉 상태 (h_{t-1}) 과 현재 시점의 입력 (x_t) 를 하나의 벡터로 합침
    2. 입력 게이트의 가중치(W_i) 와 생성한 벡터를 곱함
    3. 시그모이드 함수를 통과시키면서 "업데이트 벡터"가 생성
1. (동시진행) 추가할 정보 생성
    1. 이전 시점의 은닉 상태 (h_{t-1}) 과 현재 시점의 입력 (x_t) 를 하나의 벡터로 합침
    2. 셀의 가중치(W_c) 와 생성한 벡터를 곱함
    3. tanh를 통과시키면서 "추가할 정보" (tanh: 내용과 방향을 표현하는데 적합, -1 ~ 1)
2. 그리고 이 값을 "추가할 정보"* "업데이트 벡터" 을 통해 셀 상태(장기 기억)에 업데이트

LSTM - 게이트(Gate) - **출력 게이트(Output Gate)**

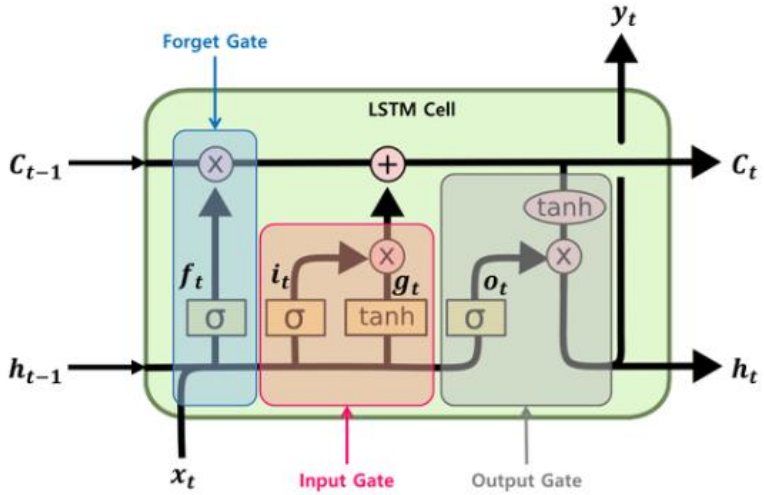
- 셀 상태(장기 기억)의 모든 정보 중에서 예측에 필요한 정보만 선별하여 외부로 내보내는 역할
- 셀의 최종 결과를 결정하는 역할
- 작동 방식
  1. 출력 벡터(마스크) 생성 (정보 선별 마스크 역할)
     1. 이전 시점의 은닉 상태 (h_{t-1}) 과 현재 시점의 입력 (x_t) 를 하나의 벡터로 합침
     2. 출력 게이트의 가중치(W_o) 와 생성한 벡터를 곱함
     3. 시그모이드 함수를 통과시키면서 "출력 벡터(마스크)"가 생성
  2. "셀 상태"* "출력 벡터(마스크)" 을 통해 중요한 데이터만 출력



- 출력 게이트는 "셀 상태"를 변경하지 않음

LSTM - 학습 방식

- LSTM 학습 과정(오차를 최소로 하는 가중치(W_f, W_i, W_o, W_c)를 찾아가는 과정)
  1. 순전파 -> 오차 계산 (RNN과 동일하게 진행)
  2. 역전파(Backpropagation Through Time, BPTT)
     - 은닉 상태(단기 기억)와 셀 상태(장기 기억) 두 경로에 오차 정보가 전달 됨
     - 은닉 상태에 영향을 준 건 "**출력 게이트(W_o)**"이므로,
       출력 게이트의 책임량을 구하면서 기존 RNN 처럼 역전파를 진행(기울기 소실 여전)
     - 셀 상태(장기 기억)에 영향을 준 건 "**입력 게이트(W_i)**" 와 "**망각 게이트(W_f)**" 이므로,
       거슬러 올라 가면서 "**입력 게이트**"의 책임량, "**망각 게이트**"의 책임량을 구하면서 역전파 진행
       - **중요)** 셀 상태는 복잡한 활성화 함수를 거치지 않았고, 단순 덧셈과 곱셈의 연산만 진행했기 때문에,
         미분을 해도 값이 "1"이 되면서, 오차 값이 손실없이 계속 전달 됨
       - 이로 인해 **기울기 소실 문제가 해결**



LSTM 의 한계

- **고정된 입출력 문제**
  - LSTM은 하나의 cell 을 재사용하고 있고, 그렇기 때문에 입력 길이는 얼마든지 달라도 상관 없음
  - 대신, Cell이 1개 이기 때문에, 입력 값에 대응하는 출력값도 1개가 나오게 됨
  - 고로 입력 값을 3개를 넣으면, 3개의 값만 출력할 수 있음
  - "나는 학생이다" 를 번역하게 된다면,
    "나는" => I , "학생이다" => am 까지만 출력하게 되고 끝이 나게 됨

- 이 문제를 해결한 방식이 바로 Seq2Seq
  - **(인코더)** 입력 받는 LSTM
  - **(디코더)** 출력하는 LSTM
  - "입력받는 역할(**인코더**)"과 "출력하는 역할(**디코더**)"를 나눠서 유연하게 대처할 수 있도록 함