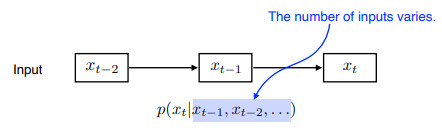
\* Sequential Model

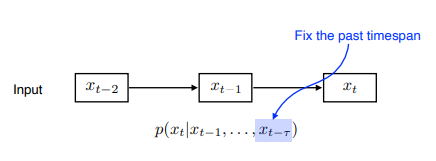
Naive sequence model



Sequential 데이터는 말, 영상, 손의 동작 등을 말한다. 그럼 Sequential 데이터를 처리하는데 가장 큰 어려움이 무엇일까? 우리가 얻고 싶은 것은 하나의 label인 경우가 많다. 혹은 '어느 영역에 무엇이 있다'와 같은 어떤 정보이다. 하지만 Sequential 데이터는 정의상 길이가 언제 끝날지 모른다. 그래서 내가 받아들여야 하는 입력의 차원을 알 수가 없다. 그래서 fully connected layer, convolution neural network를 사용할 수가 없다. 왜냐하면 몇 개의 단어, 몇 개의 음절, 몇 개의 이미지가 주어질지 모르기 때문이다. 바꿔 말하면 몇 개의 입력이 들어오든 상관없이 이 모델은 동작할 수 있어야 한다.

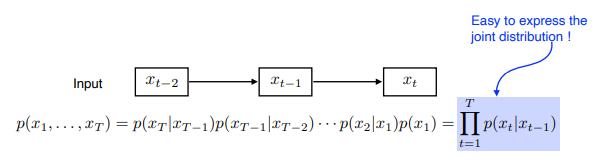
가장 기본적인 Sequential 모델은 입력이 여러 개 들어왔을 때 다음 번 입력에 대한 예측을 하는 것이다. 가장 간단한 예로는 language 모델이 있다. 내가 이전에 어떤 말이 나올 때 다음에 어떤 단어가 나오는지 예측하는 것이다. 이때 시간이 지남에 따라 과거에 고려해야하는 정보량이 점점 늘어나는 문제를 가진다.

\* Autoregressive model



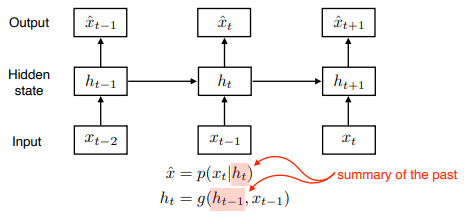
위 문제가 발생했을 때 가장 쉽게 할 수 있는 것은 Fix the past timespan이다. 단순히 과거의 몇 개의 데이터만 참고하는 것이다. 항상 몇 개의 데이터만 참고하면 되므로 계산이 훨씬 쉬워진다. 이런 전반적인 것들을 Autoregressive model이라고 한다.

\* Markov model(first-order autoregressive model)



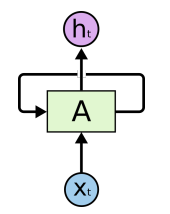
위 방법 중 가장 쉬운 것 중 하나가 Markov model이다. Markov model의 가장 큰 특징은 내가 가정하기에 나의 현재는 과거에만 dependent하다. 이때 과거는 바로 직전의 과거를 말한다. 그래서 Markov model은 너무 많은 정보를 버리게 된다. 가장 큰 장점은 위에 나온 것처럼 joint distribution을 표현하는 것이 굉장히 쉬워진다.

\* Latent autoregressive model

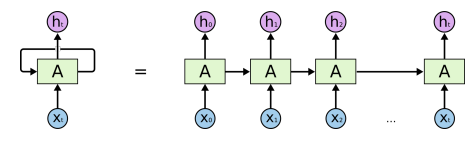


우리가 앞에서 본 Markov model의 단점은 내가 사실은 과거의 많은 정보를 고려해야 하는데 이 고려를 할 수가 없다는 것이다. 그래서 Latent autoregressive model은 중간에 hidden state가 존재한다. 그리고 이 hidden state가 과거의 정보를 summarize하고 있다. 그래서 다음 time step은 이 hidden state 하나에만 dependent한 것이다.

**1. Recurrent Neural Network(RNN)**



우리가 앞에서 말했던 concept들을 가장 쉽게 설명할 수 있고 가장 잘 구현한 방법이 RNN이다. RNN은 앞에서 봤던 MLP와 다 똑같은데 하나 다른 점이 있다. 자기 자신으로 돌아오는 구조를 가진다는 점이다. 그래서 h\_t는 X\_t에만 dependent한 것이 아니라 이전에 얻어진 어떤 데이터들에게도 dependent하다.



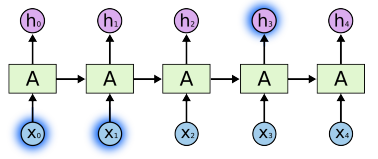
위와 같은 그림을 'RNN을 시간상으로 푼다'고 말한다.

(1) x0에서 A라는 모델로 들어간다.

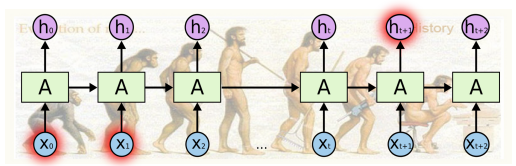
(2) 위에서 나온 정보가 x1과 합쳐져서 h1이 나온다.

(3) 위에서 나온 정보가 x2와 합쳐져서 h2가 나온다.

\* Short-term dependencies

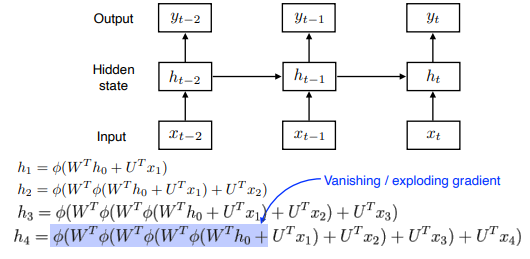


\* Long-term dependencies



RNN의 가장 큰 단점은 과거에 얻어진 정보들이 모두 취합되어서 미래에 이것을 고려해야 하는데 RNN은 하나의 fixed rule로 이 정보들을 계속 취합하기 때문에 먼 과거에 있던 데이터가 살아남기 힘들다. 그래서 몇 step전에 있는 정보는 현재 고려가 잘 되는데 한참 멀리 있는 정보는 고려하기 힘들다. 즉, Short-term dependencies는 잘 고려할 수 있지만 Long-term dependencies는 고려하기 어렵다.

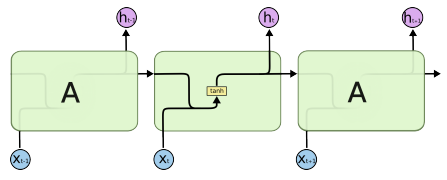
이러한 단점을 해결하고자 나온 것이 Long Short Term Memory이다.



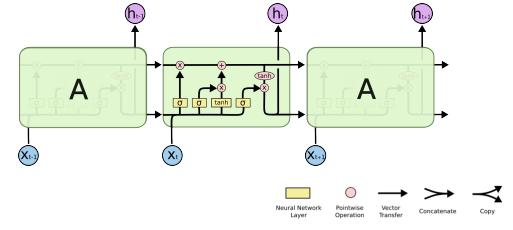
위 수식들이 RNN의 동작 원리다. 계속 중첩되는 구조임을 알 수 있는데 h0가 h4까지 가기 위해서는 똑같은 weight와 activation function을 계속 곱하는 과정을 거친다.

activation function을 sigmoid라고 생각해보자. 그러면 h0에서 왔던 정보는 계속 줄여지므로 나중에는 의미가 없어지게 된다. RELU의 경우에는 h0의 정보가 가면 갈 수록 엄청 커지게 된다. 이를 각각 Vanish gradient, Exploding gradient라고 부른다.

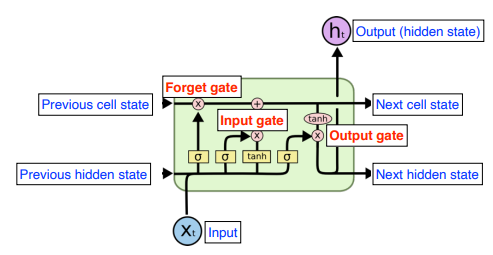
**2. Long Short Term Memory(LSTM)**



우리가 앞에서 봤던 RNN구조 를 생각해보면 위와 같이 생겼다.



위 구조가 바로 LSTM이다.

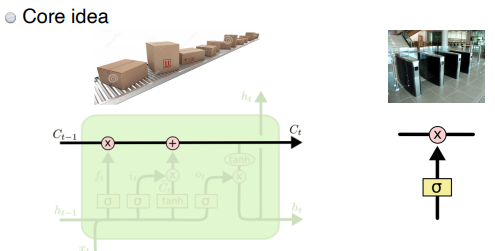


앞의 LSTM 구조에서 하나를 따온 것이다.

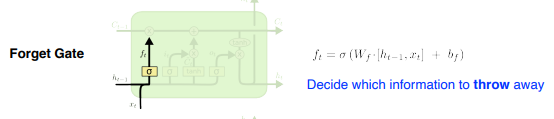
Xt는 Input이다. language model이라면 단어가 될 것이다. ht는 Ouput으로 hidden state라고 부르기로 하자.

위쪽 라인에 있는 것이 Previous cell state이다. Previous cell state를 잘보면 LSTM network 밖으로 나가지 않는다. 즉, cell state는 내부에서만 흘러가고 결과론 적으로 말하면 지금까지 들어왔던 t+1개의 정보를 모두 취합하고 있다. 아래 라인 좌측에 있는 것이 Previous hidden state이다. 그래서 LSTM의 입력으로 들어가는 것은 Previous hidden state와 Previous cell state 그리고 Xt이다.

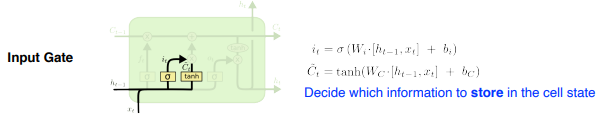
우리가 LSTM을 이해할 때 gate 위주로 이해하는 것이 좋다. LSTM은 Forget gate, Input gate, Output gate 총 3개로 이루어진다.



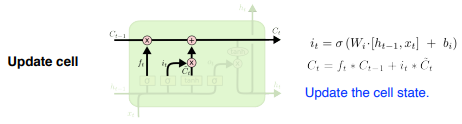
LSTM의 가장 큰 아이디어는 중간에 흘러가는 cell state이다. 그래서 이것을 어떤 container belt라고 볼 수 있다. 매번 time step t 마다 container belt에 어떤 물건이 올라온다. 그러면 container belt를 조작하는 부속품들이 이 정보를 어떤 정보가 유용하고 어떤 정보가 유용하지 않은지를 가지고 잘 조작해서 다음번에 넘겨주는 것이다. container belt에서 어떤 것을 올리고 빼고 어떻게 조작할 지에 대한 정보가 바로 이 gate에 해당한다.



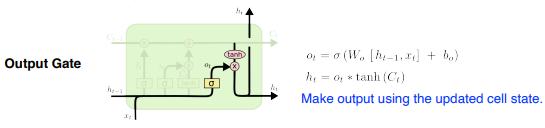
Forget Gate는 어떤 정보를 버릴지 결정한다. Forget Gate에 입력으로 들어가는 것은 현재 입력 X\_t와 이전 Output(h\_t-1)이 들어가서 f\_t라는 숫자를 얻어낸다. 위 그림에서는 sigmoid를 통과하기 때문에 항상 0~1사이의 값을 가진다. 이후 f\_t는 이전에 cell state에서 나오는 정보 중에 어떤 것을 버리고 어떤 것을 살릴지를 정해준다.



Input Gate는 현재 입력을 cell state에 무작정 올리는 것이 아니라 이 정보 중에 어떤 정보를 올릴지 말지를 결정한다. Previous hidden state와 Input을 가지고 i\_t라는 정보를 만든다. 그래서 i\_t는 어떤 정보를 추가할지 말지에 대한 정보를 가진다. 이때 추가적으로 우리가 올릴 정보를 알아야 한다. 그래서 이전의 cell state와 현재의 입력이 들어와서 다른 따로 학습되는 neural network를 통해 tanh를 통과했을 때 나오는 C\_t를 만든다. 궁극적으로 C\_t는 현재 정보와 이전 출력값을 가지고 만들어지는 Cell State Candidate인 것이다. 그래서 이전에 summarize 했던 cell state와 현재 정보와 이전 출력값에서 얻어지는 cell state candidate를 잘 섞어서 새로운 cell state로 update 해야 한다.

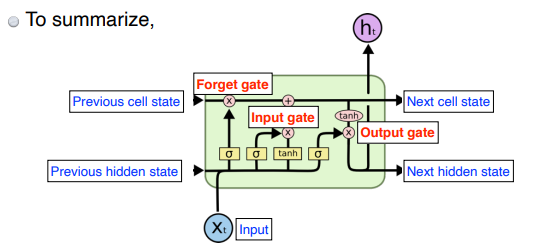


그래서 Update cell이 존재한다. Update cell은 f\_t 만큼 이전 C\_t-1을 곱해서 버릴건 버리고, C\_t를 이용해 i\_t만큼 곱해서 이 두 값을 conbine한 것을 새로운 cell state로 update한다.



Update cell까지 거치면 사실 cell state가 update되었으므로 궁극적으로 어떤 출력값을 뽑아야 한다. 그대로 cell state를 출력으로 뽑아낼 수도 있다. 하지만 LSTM에서는 한 번 더 조작을 한다. 그래서 내가 어떤 값을 밖으로 내보낼지에 해당하는 Output Gate를 만들고 Output Gate만큼 곱해서 H\_t로 Output이 나오고 Next Hidden State로 흘러가게 된다.

결과론적으로 보면 현재 이전까지 들어왔던 정보를 현재 입력을 바탕으로 지울지, 현재 입력을 바탕으로 어떤 값을 새롭게 쓸지 이 두 개의 정보를 취합하는 것이 Update Cell이고 이 취합된 Cell State를 한 번 더 조작해서 어떤 값을 빼낼지를 정하는 것이 Output Gate이다.



요약을 해보자. 먼저 Previous cell state, Previous hidden state, Input이 network 안으로 들어온다. 그다음 이전 cell state를 얼마나 지울지 정하고 Previous hidden state와 Input을 가지고 어떤 값을 올릴지 cell state Candidate를 정한다. 그리고 update된 cell state와 현재 올린 cell state candidate를 다시 조합해서 새로운 cell state를 만든다. 다음 얼만큼 값을 빼낼지를 정해서 최종적인 출력값이 나오게 된다.

**3. Gated Recurrent Unit(GRU)**

- Simpler architecture with two gates (reset gate and update gate)

- No cell state, just hidden state

