😀

#28 - Advanced Architectures of RNN (LSTM, GRU)

주제 : Advanced Architectures of RNN (LSTM, GRU)

링크 :

[#28.Lec] Advanced Architectures of RNN (LSTM, GRU) - 딥러닝 홀로서기
강의 자료 링크 : https://github.com/heartcored98/Standalone-DeepLearning/blob/master/Lec8/Lec8-A.pdf자료 저장소 링크 : https://github.com/heartcored98/Standalone-DeepLear...
https://youtu.be/cs3tSnAsyRs

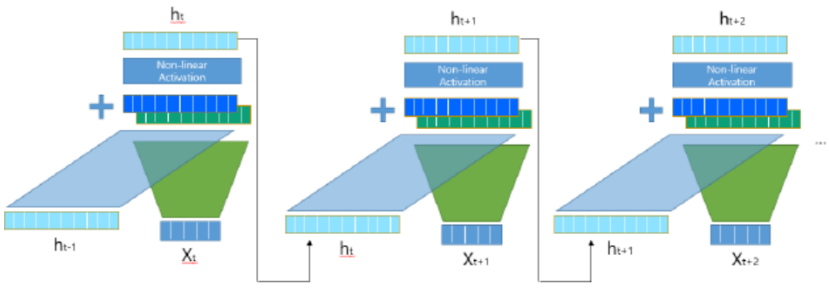
1. Review From Last Lecture

- Sequence Data를 분석할 때 4가지 형태가 있음
  - One to Many
  - Many to One
  - Many to Many (해석하는 문제, Encoder - Decoder Framework)
  - Many to Many (품사 맞추는 문제, Image Captioning)
- RNN : 각 Step의 새로운 Input을 이전 Step의 Output과 함께 고려하자

- 방식
  - Output과 Input을 차원을 맞춰서 Element 끼리 더해준 다음, Non-Linear Activation Function을 거치면 새로운 Output이 발생됨
  - 이를 반복하여 최종 Output에 선택적으로 모델을 거치면 최종 결과물이 나옴
  - $h_t = f(Ux_t + Wh_{t-1}) = W[h_{t-1}, x_t]$
    - RNN에서 사용하는 $h_t = f(Ux_t + Wh_{t-1})$방식은 Input x와 이전 Output h를 각각 연산하여 더해주는 방식인데, 이는 곧 Input X와 Output h를 하나의 Vector로 Concatenate하여 W (여기서는 W는 W와 U의 연산을 합친 W)연산 해주는 것과 동일한 결과를 낳는다

- Vanishing Gradient

- RNN은 Vanishing Gradient문제가 발생할 수 있음

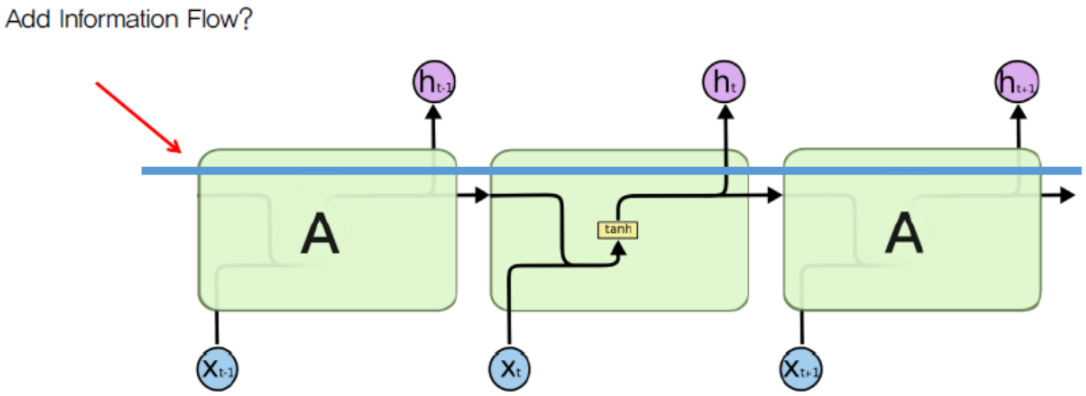


$$h_{t-2} = tanh(W[h_{t-3}, x_{t-2}])$$
$$h_{t-1} = tanh(W[h_{t-2}, x_{t-1}])$$
$$h_t = tanh(W[h_{t-1}, x_t])$$
$$h_t = tanh(W[tanh(..tanh(..h_{t-3})), x_t])$$

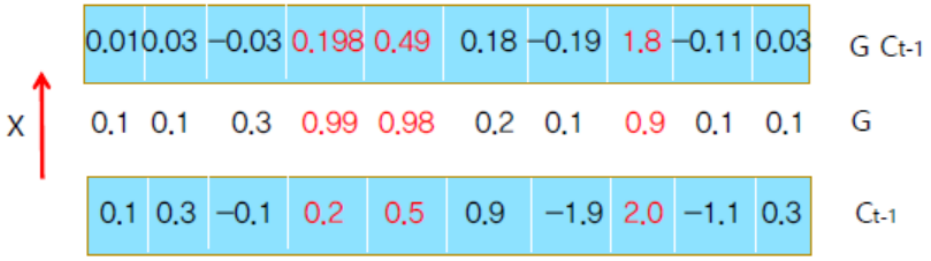
  - Long Sequence의 경우, 여러번 Activation Function을 취하면서,(=tahn (h)를 거치면) 점점 Gradient가 0으로 수렴해 버림
  - Back-propagation은 적절하게 수행할 수 없다
  - Vanilla RNN은 Long Sequence를 학습하는데 약하다

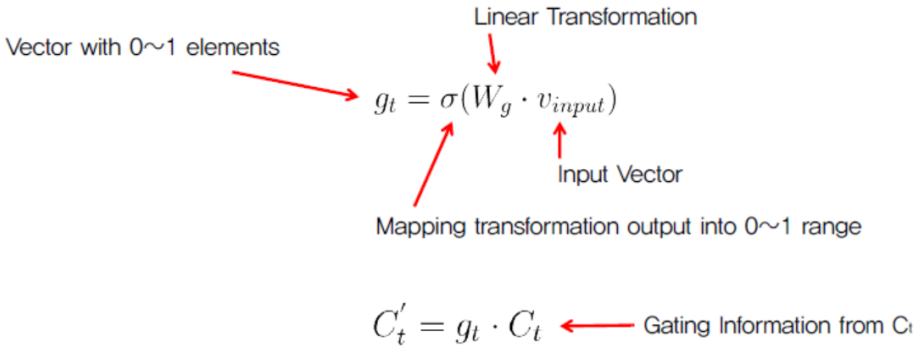
2. Long Short Term Memory Network (LSTM)

- 개념
  - RNN에서 발생한 Vanishing Gradient 문제를 해결하기 위해 Information Flow를 추가해주면 어떨까?



- 기존에는 h라인을 통해서 이전의 Output과 새로운 Input 뿐만 아니라, 그 다음 Step으로 넘어가는 결과치까지도 전달했어야 하기 때문에 병목 현상이 발생할 수 있다
- 한 채널에 두가지 역할이 부여되니, 시간에 따라 정보를 공급할 수 있는 Flow를(Cell State) 만들어보자 → RNN과의 차이점
- 남길 건 남기고, 잊어버릴 건 잊어버리고, 새로 추가할 건 추가해서 Cell State에 중요한 정보만 계속 흘러가도록!
- Hidden State는 Cell State를 적당히 가공해서 내보내자!
  - Hidden State는 각 Step에서의 Output을 나타낸다
  - Cell State는 현재 State의 정보와 내보낼 정보가 담겨져 있으므로, 현재 State의 정보 중 중요한 정보만 선별적으로 내보내자는 의미
- 어떻게 정보를 선별할 것인가? 이를 위해 Gate를 활용한다
  - Gate : 엘리먼트 와이즈하게 계수를(Coefficient) 곱해주는 것
  - 예시)
    - Gate : Cell State C_{t-1}에서 각각의 정보가 얼마나 중요한지 0~1까지로 표현
    - Cell State C_{t-1}와 Gate를 곱하여 나온 결과치(C_t')로 어떤 정보가 중요한 정보인지 식별





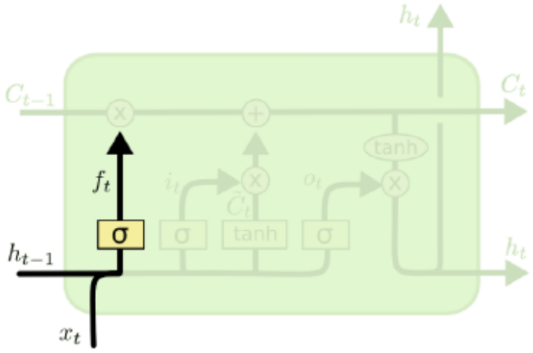
- 그럼 어떻게 Gate Coefficient 값을 얻을 수 있을까?
  - Small Non-Linear Layer를 활용하여 계산한다는 개념

- h값과 x값을 담은 Input Vector에 Cell State의 Dimension으로 변형하는 W_g를 곱해준 후(Linear Transformation), Sigmoid를 취해준다(Mapping transformation output into 0~1 range)

- LSTM 순서

  ① C_{t-1}에서 불필요한 정보를 지운다.

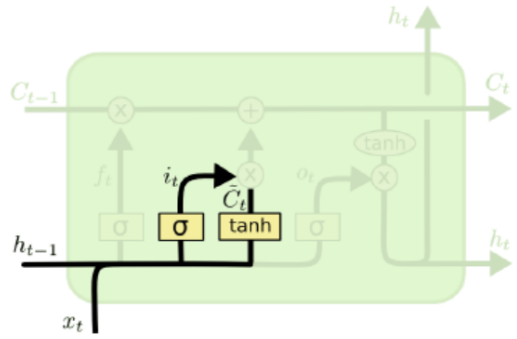
  - f는(=forget gate) 정보를 지울 지 말지를 결정하는 gate



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

  ② C_{t-1}에 새로운 인풋 x_t와h_{t-1}를 보고 중요한 정보를 넣는다

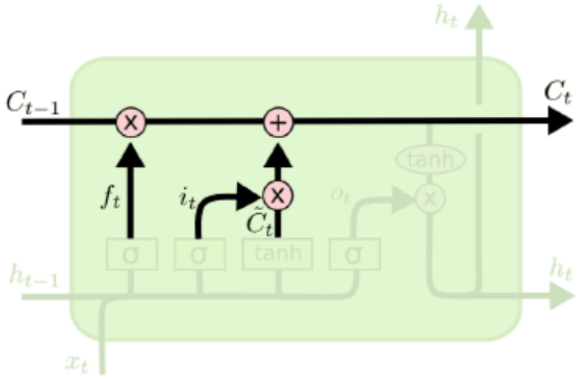
  - Input gate : h_{t-1}과x_t를 Concatenate한 후, W_i으로 Projection시키고(h dimension과 차원은 동일함), bias b_i를 더해준 후, Sigmoid를 활용하여 Input t를 만든다
  - $\tilde{C}_t$또한 비슷한 방식으로 만들 되, 마지막에 tanh함수를 이용하여 만들어준다
    - RNN에서 Non-Linear Activation을 취해준 것과 동일한 개념



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

  ③ 위 과정을 통해 C_t 를 만든다 .

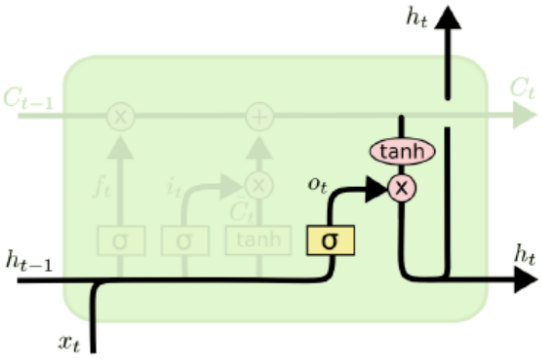
  - 이전 Step에서 온 Cell State 가공 과정 → C_{t-1}에 forget gate를 곱해서 정보를 선별
  - 현재 Step에서 온 Cell State 가공 과정 → 임시 $\tilde{C}_t$와 input gate를 곱하여 정보를 선별
  - 위 두 결과물을 서로 더하여 최종 C_t를 도출



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

④ C_t 를 적절히 가공해 해당 t에서의 h_t를 만든다.

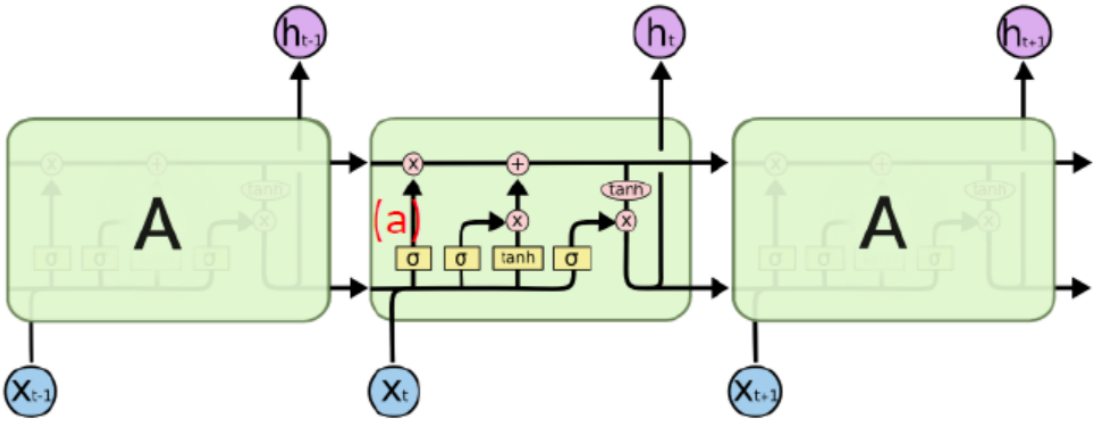
- Output Gate(o_t) : Input Vector (h_{t-1}, x_t)를 활용하여 Gate Coefficient 생성
- C_t에 Activation Function을 취한 후, 이를 Gate Coefficient에 곱하여 최종 h_t를 도출
- 즉, C_t의 값들을 모두 사용할 지 말지를 Output gate를 활용하여 선별



$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

⑤ C_t 와 h_t를 다음 스텝 t+1 로 전달한다

- C_t와h_t를 함께 전달하는 이유
  - C_t에 담겨져 있는 현재 정보들과 C_t로 가공한h_t를 함께 전달함으로써 다음 Step에서 새로운 Input으로 활용하기 위함



- 그럼 어떻게 LSTM이 Vanishing Gradient 문제를 해결할 수 있는가?
  - t 시점에서 Cell state에 기록된 정보가 만약 지워지지 않고 t+n에서 활용 되었을 때 중간에 non-linear activation function을 거치지 않고 t+n시점까지 흘러오기 때문에 Vanishing Gradient Problem을 상당 부분 해결 할 수 있음
  - Standard RNN은 중요하든 중요하지 않든 모두 Activation Function을 거치면서 0으로 수렴해 버렸는데, LSTM은 중요한 정보의 경우, Cell State를 통하여 Activation Function을 거치지 않고 전달될 수 있다는 차이점이 있다

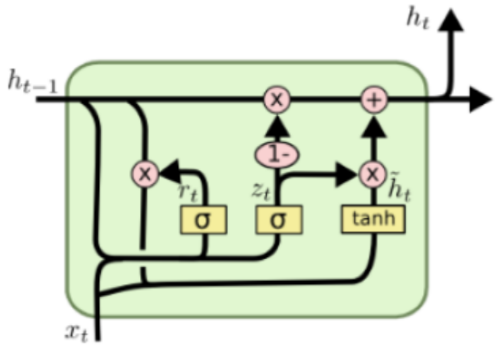
Q. 결국 RNN과 다르게, LSTM은 각 Step마다 h_t를 계산 시, Activation Function을 두번씩 취해주는 C_t를 다음 Step으로 전달해주게 되는데, 그럼 문제가 없을까?

→ 다음 Step에서 h_t는x_{t+1}과 함께 gate를 만드는데 사용됨

→ 어차피 C_t를 생성시, Activation Function을 취하지 않은C_{t-1}을 더한 것으로 생성하였으므로 C_t는 Activation Function을 취하지 않은 결과값과 같음

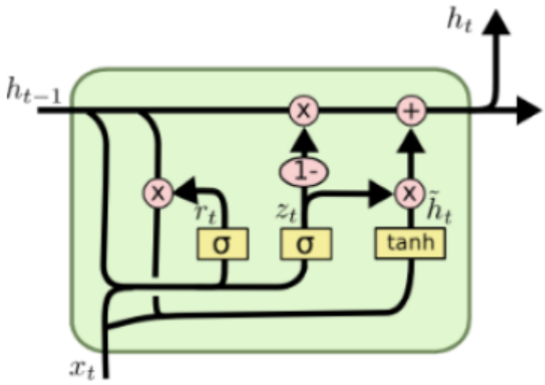
3. Gate Recurrent Unit (GRU)

- 개념
  - LSTM은 Gate가 너무 많은 느낌이라, 이런 Gate를 줄일 수 없을까라는 의문에서 시작됨
- 순서
  - Reset gate를 계산해서 임시 $\tilde{h}_t$를 만든다
    - h_{t-1}과x_t를 W Projection하여 Sigmoid한 숫자를 Reset Gate로 생성
    - Reset Gate를 h_{t-1}에 곱한 수로 다시금 x_t와 W로 Projection하여 Activation function (tahnh)을 취하여 임시 $\tilde{h}_t$를 생성



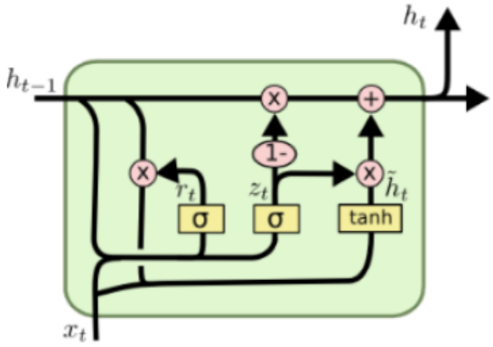
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

- Update gate를 통해 h_{t-1}과$\tilde{h}_t$를 계산한다
  - Update gate : 이전 h_{t-1}과x_t간의 밸런스를 맞춰주는 gate



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

- z_t를 이용해 최종 h_t를 계산한다
  - z_t가 1 or 0이라면, 자동적으로 h_{t-1}의 항이 0 or 1으로 변경되면서 밸런스가 맞춰짐
  - LSTM에서는 forget gate과 input gate가 하던 일을 여기서는 Update gate를 통하여 수행함



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

- 그럼 GRU가 Vanishing Gradient를 해결 할 수 있는가?

    - Non-Linear Activation을 지나지 않을 수 있기에 GRU도 Vanishing Gradient를 해결할 수 있음

- ReLU도 Activation Function으로 사용할 수 있으나, 아직까지 어느 것이 좋은지는 비교를 해보지 않았음

- Expirical Exploration of RNN Architectures (성능 차이)
    - 대부분의 RNN모델은 성능차이가 거의 없음
    - GRU가 파라메터 수가 적기에 계산할 때 GRU가 빠르다.
    - 다만, GRU가 발견되기 시점에는 GPU의 성능이 따라오지 못하여 GRU가 획기적이었지만, 현재는 GPU 성능이 좋기 때문에 Model Capacity가 크고 안정적인 LSTM이 많이 쓰고 있음

Arch.	N	N-dropout	P
Tanh	3.612	3.267	6.809
LSTM	3.492	3.403	6.866
LSTM-f	3.732	3.420	6.813
LSTM-i	3.426	**3.252**	6.856
LSTM-o	3.406	**3.253**	6.870
LSTM-b	3.419	3.345	6.820
GRU	3.410	3.427	6.876
MUT1	**3.254**	3.376	**6.792**
MUT2	3.372	3.429	6.852
MUT3	3.337	3.505	6.840

Table 2. Negative Log Likelihood on the music datasets. N stands for Nottingham, N-dropout stands for Nottingham with nonzero dropout, and P stands for Piano-Midi.

Arch.	5M-tst	10M-v	20M-v	20M-tst
Tanh	4.811	4.729	4.635	4.582 (97.7)
LSTM	4.699	4.511	4.437	4.399 (81.4)
LSTM-f	4.785	4.752	4.658	4.606 (100.8)
LSTM-i	4.755	4.558	4.480	4.444 (85.1)
LSTM-o	4.708	4.496	4.447	4.411 (82.3)
LSTM-b	4.698	4.437	4.423	**4.380 (79.83)**
GRU	4.684	4.554	4.559	4.519 (91.7)
MUT1	4.699	4.605	4.594	4.550 (94.6)
MUT2	4.707	4.539	4.538	4.503 (90.2)
MUT3	4.692	4.523	4.530	4.494 (89.47)

Table 3. Perplexities on the PTB. The prefix (e.g., 5M) denotes the number of parameters in the model. The suffix "v" denotes validation negative log likelihood, the suffix"tst" refers to the test set. The perplexity for select architectures is reported in parentheses. We used dropout only on models that have 10M or 20M parameters, since the 5M models did not benefit from dropout at all, and most dropout-free models achieved a test perplexity of 108, and never greater than 120. In particular, the perplexity of the best models without dropout is below 110, which outperforms the results of Mikolov et al. (2014).

Most RNN variants are almost the same

- Summary
    - Vanilla RNN은 Vanishing Gradient 문제를 가지고 있음
    - LSTM Architecture는 Vanishing Gradient 문제를 Cell State를 통해서 해결
    - GRU는 Cell State를 없애면서도 Vanishing Gradient 문제를 해결하는 효과적인 방안이다