Vanishing Gradient and Fancy RNNs

2018.06.07

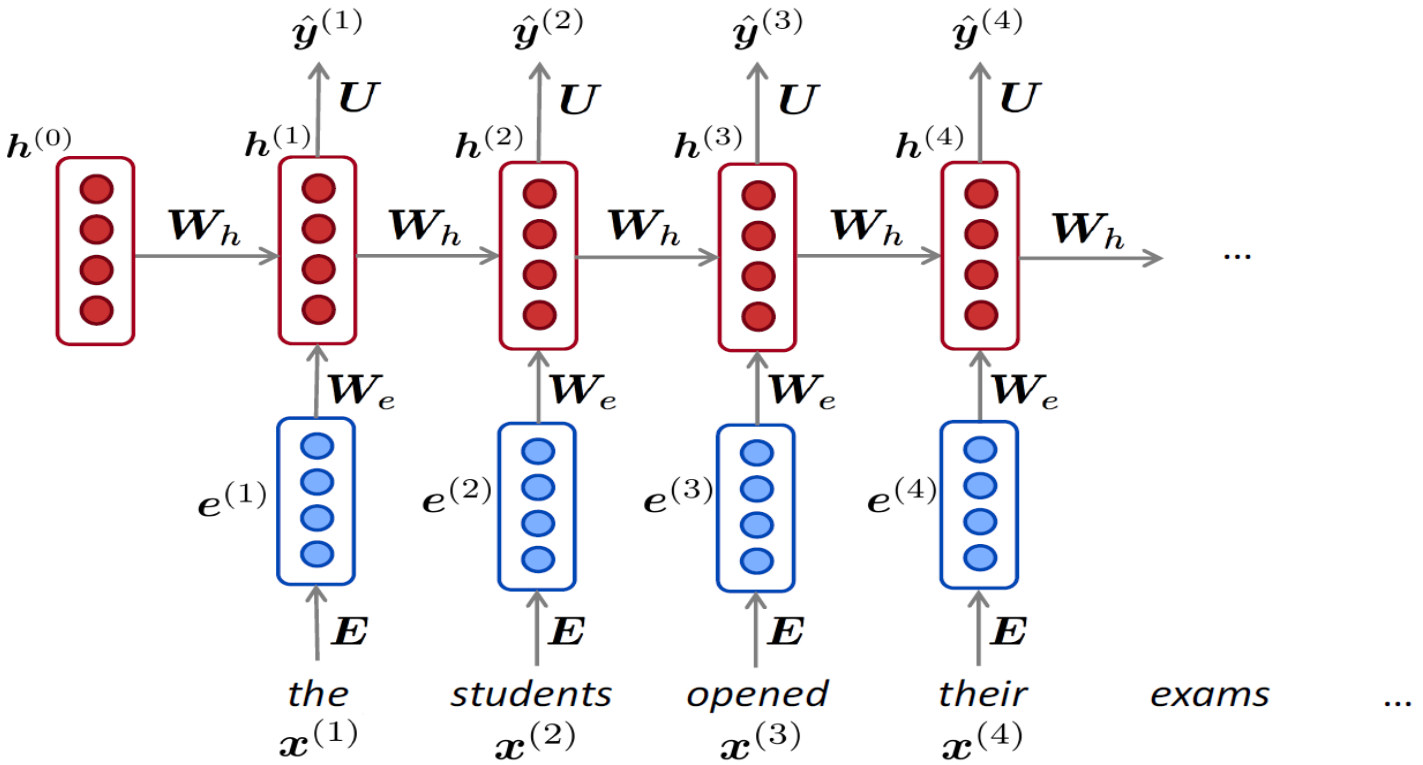
최경아

Contents

1. **Vanishing Gradient problem**
2. **Fancy RNNs:**
    1) LSTM
    2) GRU
    3) Bidirectional RNNs

1. Vanishing Gradient problem

- 연속열 데이터의 문맥을 포착하여 추정하는 것이 가능한 모델
- Forward prop 동안에 매 time step에서 동일한 W metric이 곱해짐
- Back prop 과정에서도 매 time step에서 같은 metrix를 곱해감
- 현재 시각으로부터 얼마나 먼 과거의 입력까지 반영할 수 있는지가 중요
- 실제 RNN의 출력에 반영시킬 수 있는 것은 기껏해야 과거 10시각 정도로 알려짐(딥러닝 제대로 시작하기)



1. Vanishing Gradient problem

- RNN은 관련 정보와 그 정보를 사용하는 지점 사이 거리가 멀 경우 Back prop과정에서 기울기가 점차 줄거나 폭발적으로 커져 (가중치 업데이트가 잘 안되거나 학습 불가)학습능력의 저하를 가져옴 : **vanishing gradient problem**

- Similar but simpler RNN formulation:

$$h_t = W f(h_{t-1}) + W^{(hx)} x_{[t]}$$
$$\hat{y}_t = W^{(S)} f(h_t)$$

- Total error is the sum of each error (aka cost function, aka J in previous lectures when it was cross entropy error, could be other cost functions too), but at time steps t

$$\frac{\partial E}{\partial W} = \sum_{t=1}^{T} \frac{\partial E_t}{\partial W}$$

- Hardcore chain rule application:

$$\frac{\partial E_t}{\partial W} = \sum_{k=1}^{t} \frac{\partial E_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial W}$$

- Similar to backprop but less efficient formulation
- Useful for analysis we'll look at:

$$\frac{\partial E_t}{\partial W} = \sum_{k=1}^{t} \frac{\partial E_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \boxed{\frac{\partial h_t}{\partial h_k}} \frac{\partial h_k}{\partial W}$$

We'll show this can quickly become very small or very large

1. Vanishing Gradient problem

- Remember: $h_t = W f(h_{t-1}) + W^{(hx)} x_{[t]}$
- More chain rule, remember:

$$\frac{\partial h_t}{\partial h_k} = \prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}}$$

- Each partial is a Jacobian:

$$\frac{d\mathbf{f}}{d\mathbf{x}} = \begin{bmatrix} \frac{\partial \mathbf{f}}{\partial x_1} & \cdots & \frac{\partial \mathbf{f}}{\partial x_n} \end{bmatrix} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$$

- Analyzing the norms of the Jacobians yields:

$$\left\| \frac{\partial h_j}{\partial h_{j-1}} \right\| \leq \|W^T\| \|\mathrm{diag}[f'(h_{j-1})]\| \leq \beta_W \beta_h$$

- Where we defined β's as upper bounds of the norms
- The gradient is a product of Jacobian matrices, each associated with a step in the forward computation.

$$\left\| \frac{\partial h_t}{\partial h_k} \right\| = \left\| \prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}} \right\| \leq (\beta_W \beta_h)^{t-k}$$

- This can become very small or very large quickly [Bengio et al 1994], and the locality assumption of gradient descent breaks down. → **Vanishing or exploding gradient**

1. Vanishing Gradient problem

- Language model 에서 RNN의 vanishing gradient 문제는 긴 과거 time step 정보를 고려할 수 없는 문제가 발생함

The vanishing gradient problem for language models

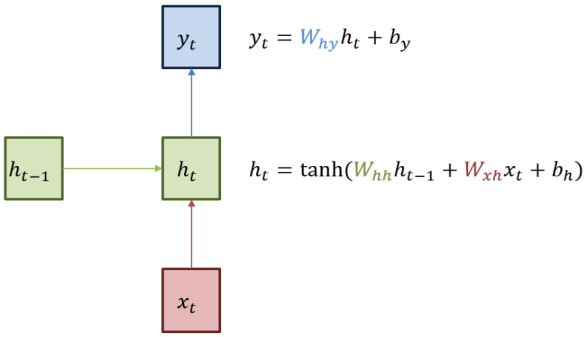
- The vanishing gradient problem can cause problems for RNN Language Models:
- When predicting the next word, information from many time steps in the past is not taken into consideration.

- Example:

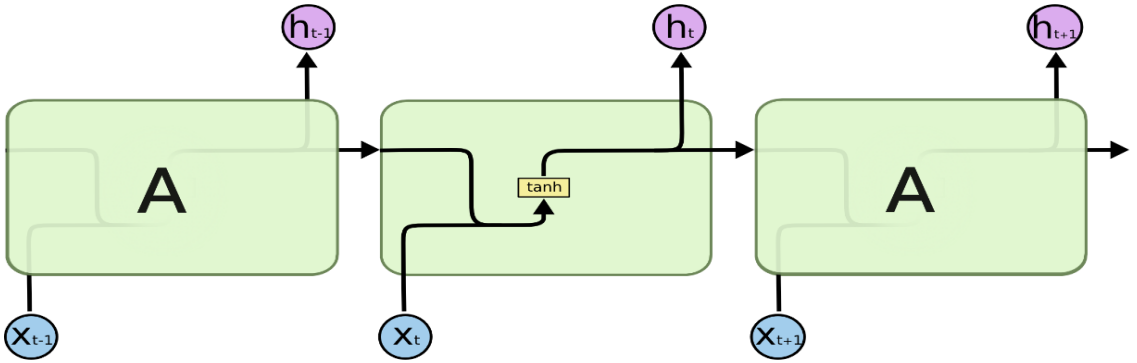
  Jane walked into the room. John walked in too. It was late in the day. Jane said hi to ____

2. Fancy RNNs
2.1 LSTM

- LSTM은 long term distance 문제를 해결할 수 있도록 설계된 모델
- 하나의 simple layer 구조의 RNN에서 4개의 layer 들이 상호작용하는 구조

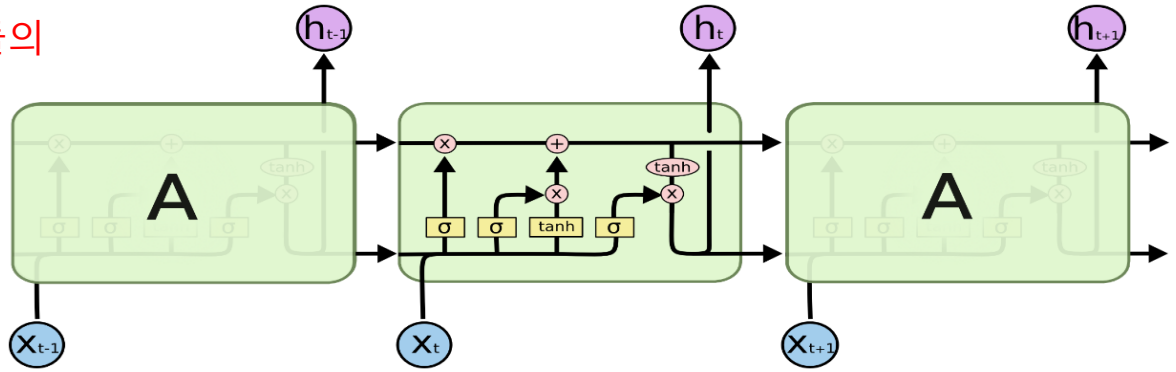


하나의 tanh simple layer

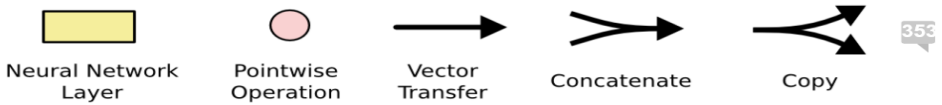


The repeating module in a standard RNN contains a single layer.

4개의 layer들의 상호작용



The repeating module in an LSTM contains four interacting layers.

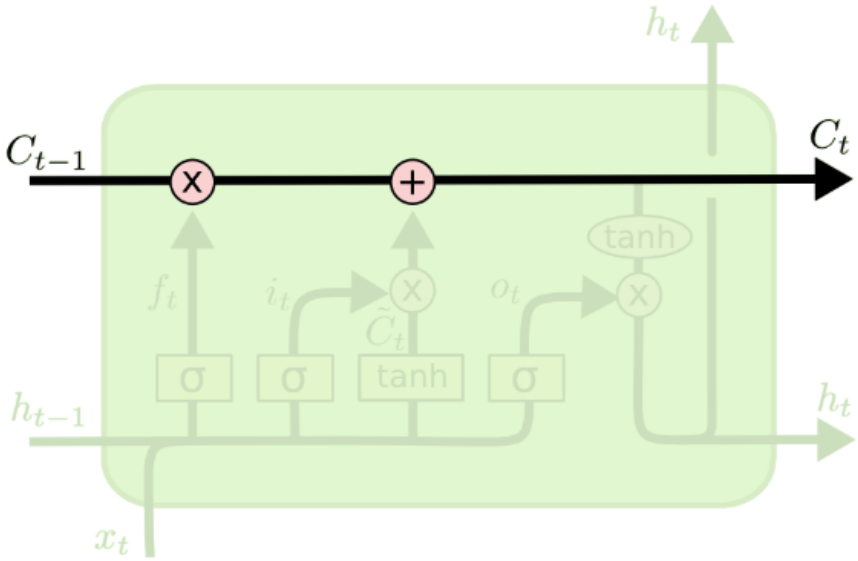


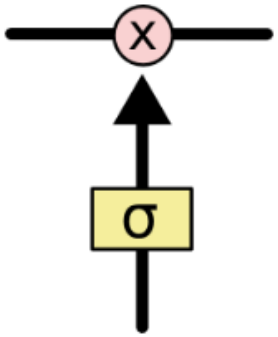
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

2. Fancy RNNs
2.1 LSTM

[LSTM의 Core idea]

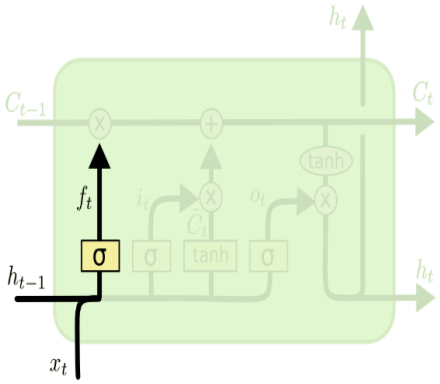
- LSTM의 key로 cell state로 컨베이어 벨트처럼 전체를 가로지르며 흐른다. (정보 전달하기 좋은 구조)
- LSTM은 정보를 더하거나 제거하는 능력을 가지고 있음
- 시그모이드와 곱으로 이루어진 GATE에서 정보를 얼마나 흘려보낼지 선택 (시그모이드 0~1, 0이면 정보를 보내지 않음, 1이면 정보 모두 흘려보냄)





2. Fancy RNNs
2.1 LSTM

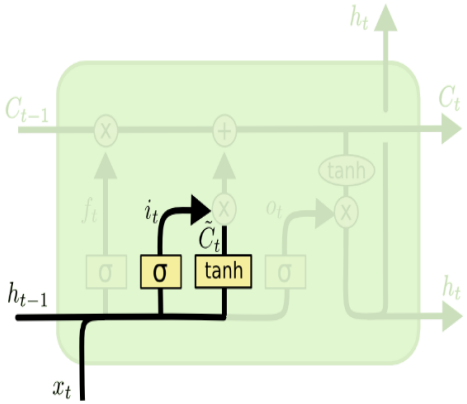
[LSTM 구조]



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

1단계 : 이전 cell state로부터 얼마나 정보를 흘려보낼지 결정

❖ **Forget gate**
- 과거의 정보를 잊기위한 gate
- ht-1과 xt를 받아 시그모이드를 취해준 값을 내보냄
- 시그모이드 0~1사이값, 0이면 이전 상태정보를 잊고, 1이면 이전 상태 전부 기억



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

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

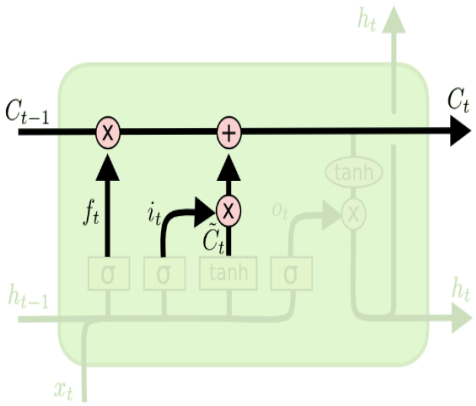
2단계 : cell state에 새로운 정보를 얼마나 저장할지 결정

❖ **Input gate**
- 현재의 정보를 저장하기 위한 gate
- 0~1사이 값
- 업데이트 할 값을 결정

❖ **New memory cell**
- -1~1사이값
- Candidate value(RNN계산과 동일)
- Input gate와 연산하여 일부만 출력

2. Fancy RNNs
2.1 LSTM

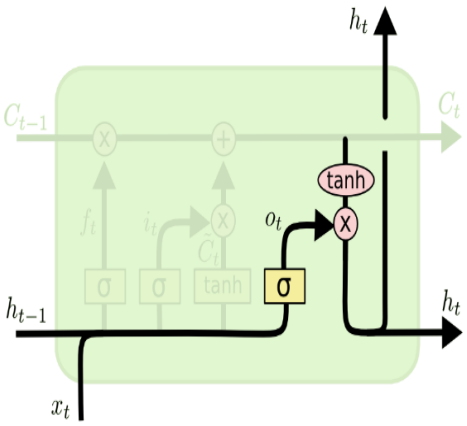
[LSTM 구조]



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

3단계 : Cell State업데이트

- ❖ **Final memory cell**
  - Forget gate를 통해 이전 cell의 정보를 조절하고, input gate를 통해 현재 입력(candidate value)정보를 전달



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

$$h_t = o_t * \tanh\left(C_t\right)$$

4단계 :어느 정도로 현재 state의 정보를 출력이나 상위 layer(stacked layer의 경우)에 전달할지 결정

- ❖ Output gate
  - Final memory cell 값을 얼마나 출력할지 수준 정의
- ❖ Final hidden state
  - Final memory cell를 tanh 변환 후 output gate 곱

2. Fancy RNNs
2.1 LSTM

[왜 LSTM은 vanishing gradient 문제에 강한가?]

LSTM의 Cell State C_t에 대한 계산식은 다음과 같다.

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

이때, C_T를 C_t에 대해 미분한다고 하면(T>t), 이는 chain rule에 의해 다음과 같이 표현될 수 있다.

$$\frac{\partial C_T}{\partial C_t} = \frac{\partial C_T}{\partial C_{T-1}} * \frac{\partial C_{T-1}}{\partial C_{T-2}} * \ldots * \frac{\partial C_{t+1}}{\partial C_t}$$

여기서, $\frac{\partial C_T}{\partial C_{T-1}} = f_T$, $\frac{\partial C_{T-1}}{\partial C_{T-2}} = f_{T-1}, \ldots, \frac{\partial C_{t+1}}{\partial C_t} = f_{t+1}$ 이므로

$$\frac{\partial C_T}{\partial C_t} = \prod_{i=t+1}^{T} f_i$$

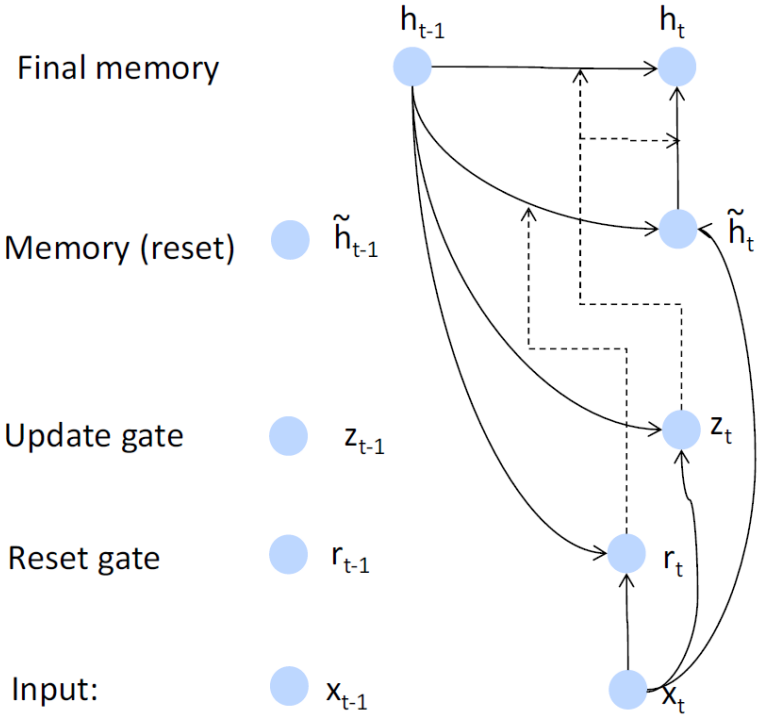
위 식의 f는 sigmoid함수의 output이기 때문에 (0,1)의 값을 갖게 되는데, 이 값이 1에 가까운 값을 갖게되면 미분값(gradient)이 소멸(vanished)되는 것을 최소한으로 줄일 수 있게된다. f값이 1에 가깝다는 것은, Cell State 공식에 의하면 오래된 기억(long term memory)에 대해 큰 비중을 둔다는 것과 같은데, 이로인해 gradient 또한 오래 유지된다는 것은 꽤나 흥미로운 현상이다.

+더불어 f는 1보다 큰 값을 가질 수 없으므로 미분식이 깊어진다고 해서(T-t값이 커진다고 해서) 이로인해 그 값이 넘치게(exploded) 되지는 않는다.

2. Fancy RNNs
2.2 GRU

- GRU는 LSTM의 장점을 유지하면서 계산의 복잡성을 낮춘 셀 구조
- Update gate와 reset gate로 구성
- New memory 를 보면 현시점정보와 과거 정보를 반영하며, 과거 정보를 얼마나 반영할지는 reset gate에 의존(reset gate와 별개로 현시점 정보는 모두 반영)

- Update gate $z_t = \sigma\left(W^{(z)}x_t + U^{(z)}h_{t-1}\right)$
- Reset gate $r_t = \sigma\left(W^{(r)}x_t + U^{(r)}h_{t-1}\right)$
- New memory content: $\tilde{h}_t = \tanh\left(Wx_t + r_t \circ Uh_{t-1}\right)$
  If reset gate unit is ~0, then this ignores previous memory and only stores the new word information
- Final memory at time step combines current and previous time steps: $h_t = z_t \circ h_{t-1} + (1 - z_t) \circ \tilde{h}_t$



2. Fancy RNNs
2.2 GRU

- Update gate는 과거의 정보를 얼마나 중요하게 받아들일지 조정
- Z=1에 가까우면, 매우 긴 time step 안의 정보도 보존
- short-term 의존도를 갖는 unit은 reset gate가 활성화 되어 있고, long-term 의존도를 갖는 unit은 update gate가 활성화

- If reset is close to 0, ignore previous hidden state → Allows model to drop information that is irrelevant in the future

$$z_t = \sigma\left(W^{(z)}x_t + U^{(z)}h_{t-1}\right)$$
$$r_t = \sigma\left(W^{(r)}x_t + U^{(r)}h_{t-1}\right)$$
$$\tilde{h}_t = \tanh\left(Wx_t + r_t \circ Uh_{t-1}\right)$$
$$h_t = z_t \circ h_{t-1} + (1 - z_t) \circ \tilde{h}_t$$

- Update gate z controls how much of past state should matter now.
  - If z close to 1, then we can copy information in that unit through many time steps! **Less vanishing gradient**!

$\left|\frac{\partial h_j}{\partial h_{j-1}}\right| \approx 1$

- Units with short-term dependencies often have reset gates very active
- Units with long term dependencies have active update gates z

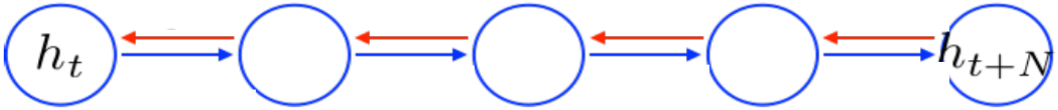
2. Fancy RNNs
2.2 GRU

[GRU는 어떻게 vanishing gradient 문제를 해결할 수 있는가?]

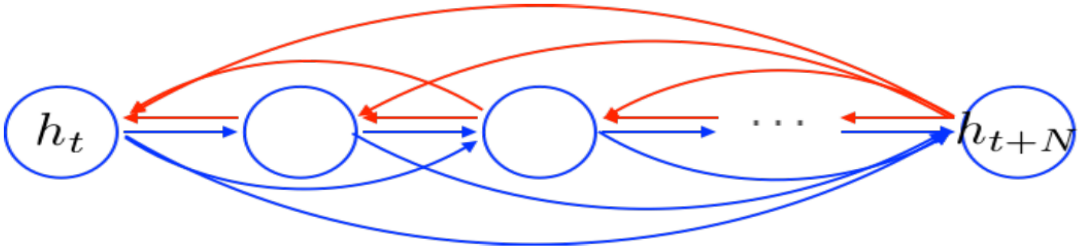
- Is the problem with standard RNNs the naïve transition function?

$$h_t = f\left(W^{(hh)}h_{t-1} + W^{(hx)}x_t\right)$$

- It implies that the error must backpropagate through all the intermediate nodes:



- Perhaps we can create *adaptive* shortcut connections.
- Let the net prune unnecessary connections *adaptively*.

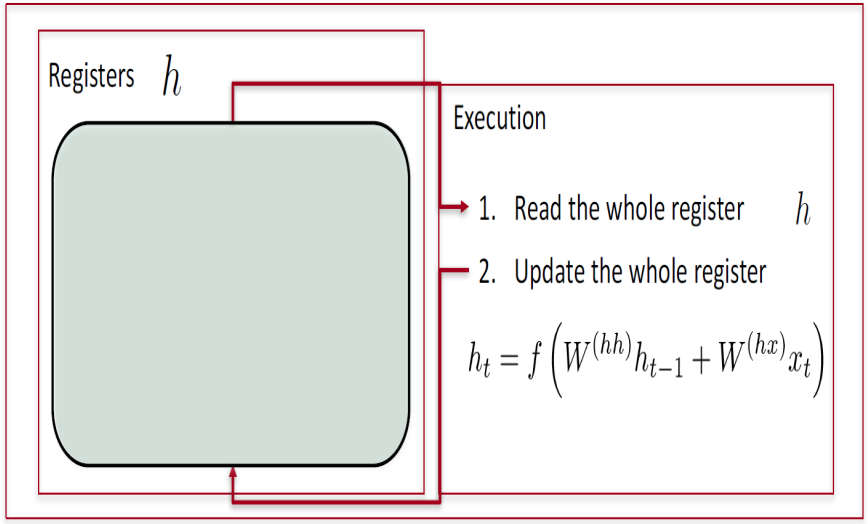


2. Fancy RNNs
2.2 GRU

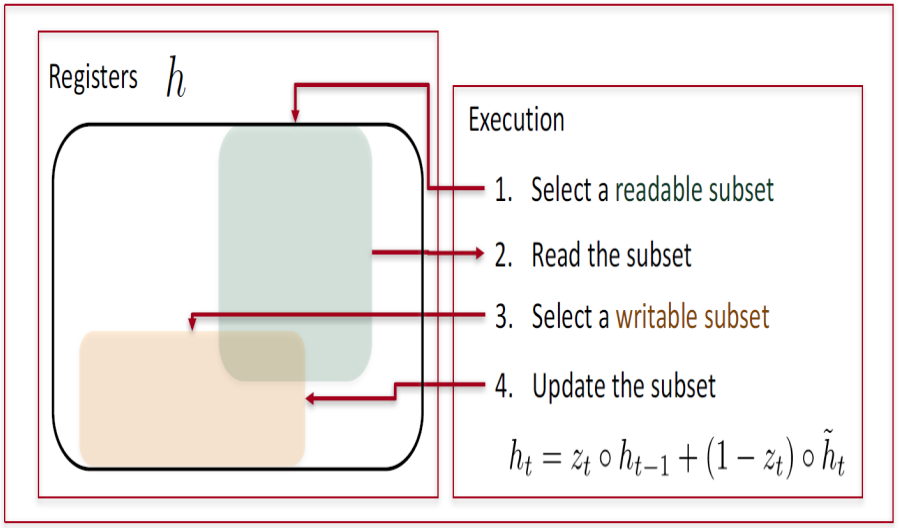
[RNN과 GRU 차이]

- RNN은 모든 정보를 다 읽고, Update
- GRU는 필요한 정보만 일부 읽고, Update 함

Vanilla RNN ...



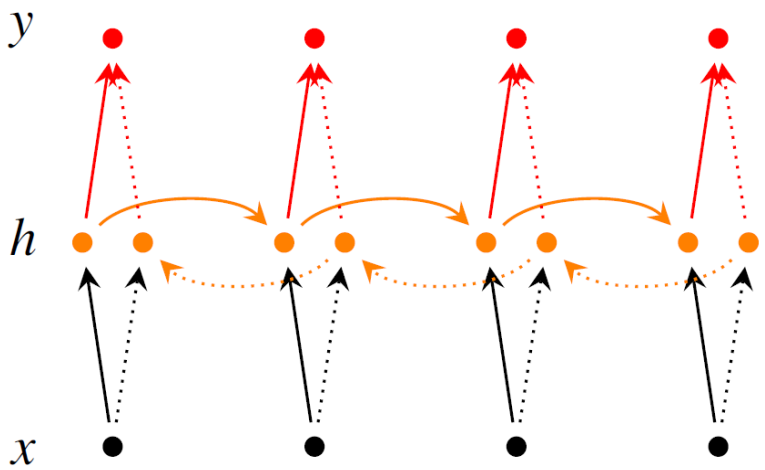
GRU ...



2. Fancy RNNs
2.3. Bidirectional RNNs

- 이전 정보 뿐 아니라 이후의 정보까지 반영하는 모델
  예를 들어 "푸른 하늘에 oo이 떠있다."라는 문장에서 oo에 들어갈 단어를 예측하고자 한다면, 우리는 앞의 정보인 "푸른", "하늘"이라는 정보를 가지고 oo에 들어갈 단어가 "구름"이라고 예측할 수도 있지만, 뒤의 정보인 "떠있다."라는 단어까지 함께 고려하면 더 높은 확률로 정답이 "구름"이라는 것을 예측할 수 있을 것이다.

Problem: For classification you want to incorporate information from words both preceding and following



$$\vec{h}_t = f(\vec{W}x_t + \vec{V}\vec{h}_{t-1} + \vec{b})$$
$$\overleftarrow{h}_t = f(\overleftarrow{W}x_t + \overleftarrow{V}\overleftarrow{h}_{t+1} + \overleftarrow{b})$$
$$y_t = g(U[\vec{h}_t; \overleftarrow{h}_t] + c)$$

$h = [\vec{h}; \overleftarrow{h}]$ now represents (summarizes) the past and future around a single token.