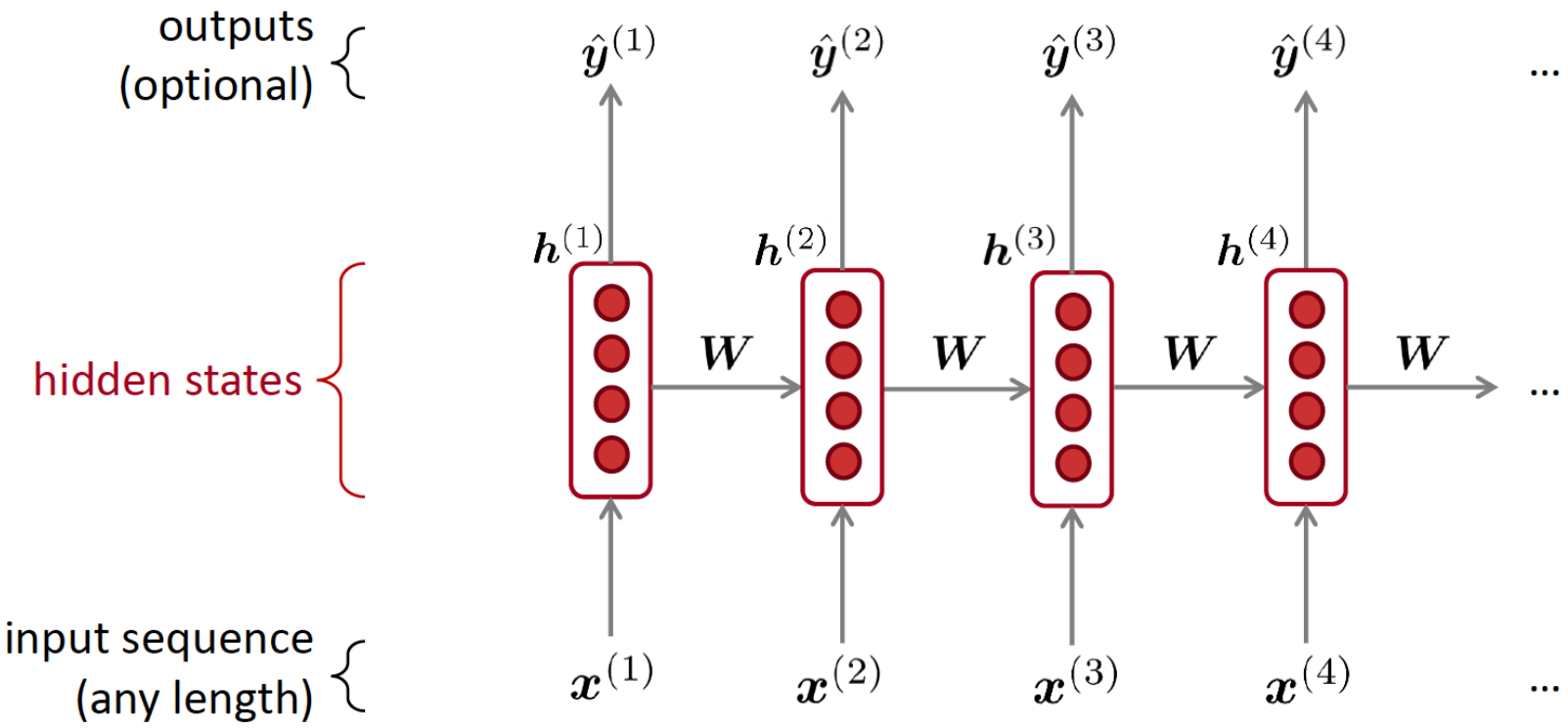
Recurrent Neural Network

Sequence Data

- I am a boy
- 위 문장을 컴퓨터 언어로 임베딩
- 각 단어: 앞 뒤 단어, 그리고 문장 전체와의 관계 고려해야!
- 문장(NLP), 음성, 동영상 등 time dependency가 있는 시계열 데이터
- 일반적인 Neural Network나 CNN은 처리할 수 없음!

Recurrent Neural Network

- 핵심 아이디어: 매 번 같은 weight matrix W를 적용한다!



RNN의 구조

We can process a sequence of vectors **x** by applying a recurrence formula at every time step:

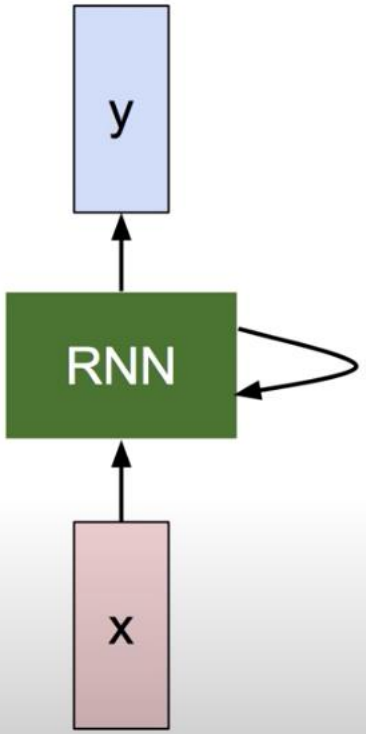
$$h_t = f_W(h_{t-1}, x_t)$$

new state

old state

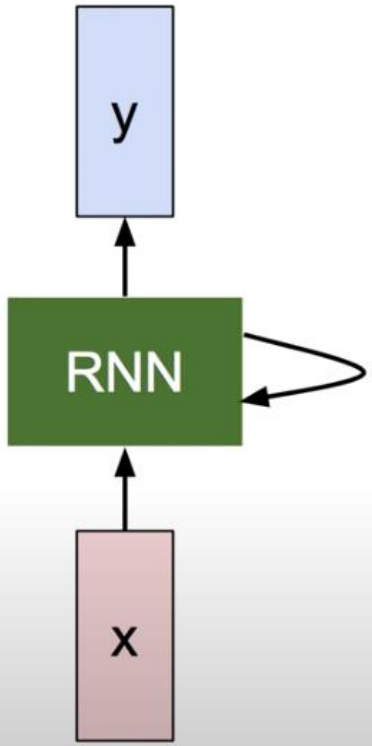
input vector at some time step

some function with parameters W



- Input 뿐만 아니라 hidden state를 이용해서 연산 진행
- 매 번 같은 함수 f를 적용하기 때문에 위와 같은 그림으로 표현

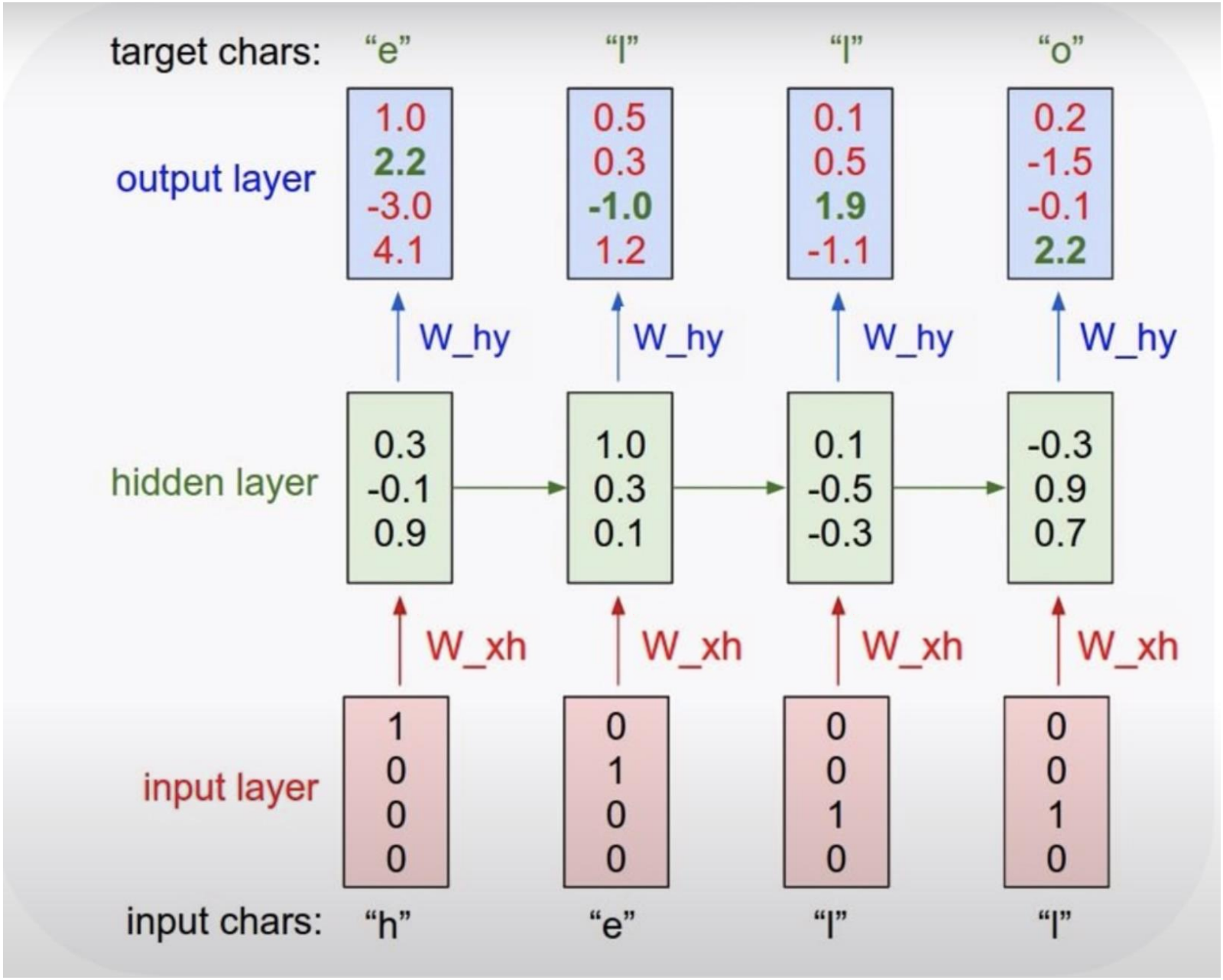
RNN의 구조



$$h_t = f_W(h_{t-1}, x_t)$$

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

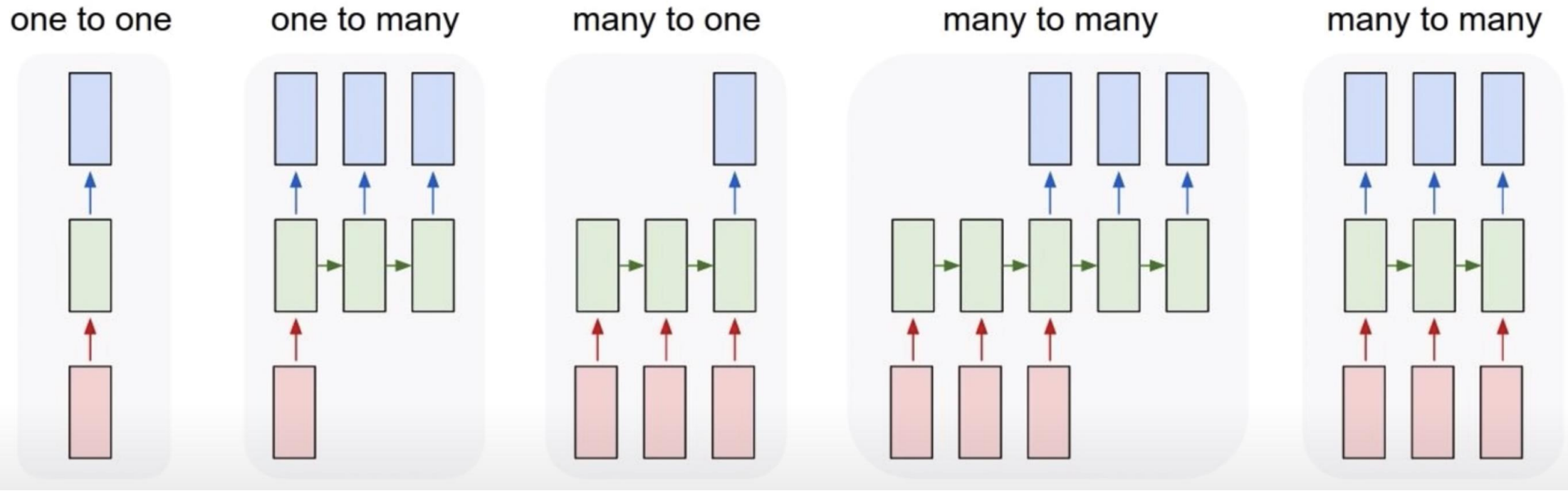
$$y_t = W_{hy}h_t$$



"hello"라는 input을 처리하는 과정

1. 각 알파벳을 벡터로 임베딩
2. 앞에서부터 차례대로 주어진 input과 hidden state 연산
3. 이 결과를 바탕으로 다음에 올 문자를 예측

RNN의 활용



Vanilla NN

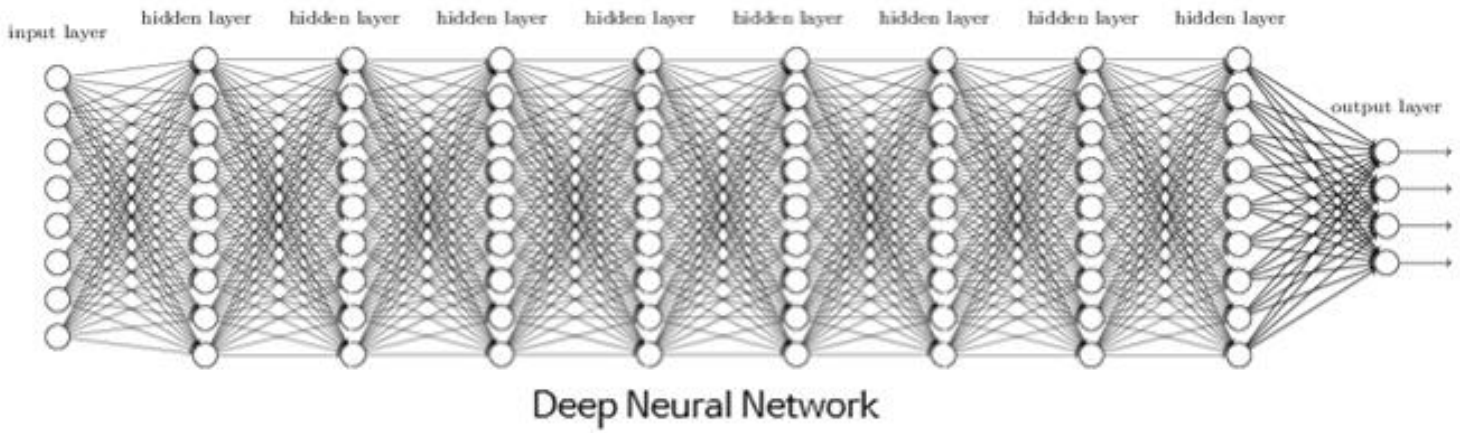
Image captioning

Sentiment classification

Machine translation

Video classification

Vanishing Gradient Problem

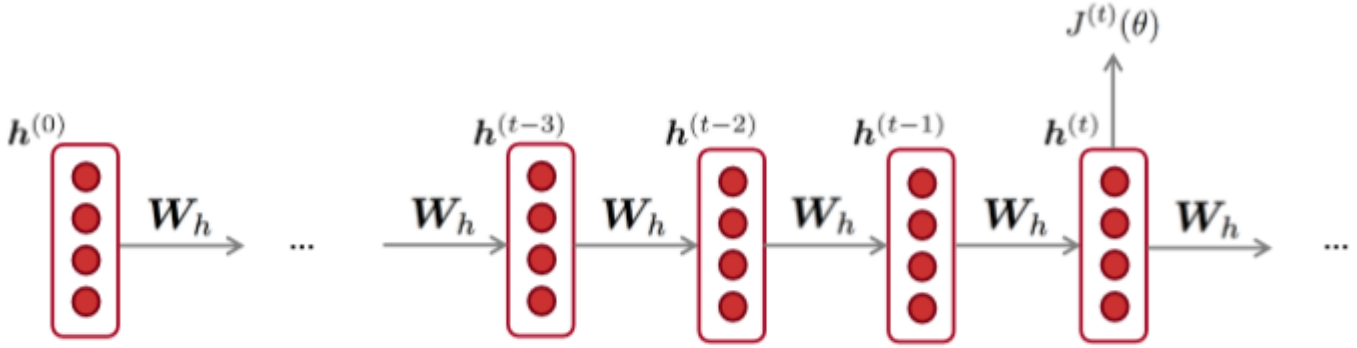


Vanishing/Exploding Gradient Problem

- RNN의 가장 큰 특징: 똑같은 weight matrix W가 반복적으로 사용됨

-> cost function을 계산할 때 매 번 W가 곱해짐

Backpropagation for RNNs



Question: What's the derivative of $J^{(t)}(\theta)$ w.r.t. the repeated weight matrix $\boldsymbol{W_h}$?

Answer: $$\frac{\partial J^{(t)}}{\partial \boldsymbol{W_h}} = \sum_{i=1}^{t} \frac{\partial J^{(t)}}{\partial \boldsymbol{W_h}}\bigg|_{(i)}$$

"The gradient w.r.t. a repeated weight is the sum of the gradient w.r.t. each time it appears"

Why?

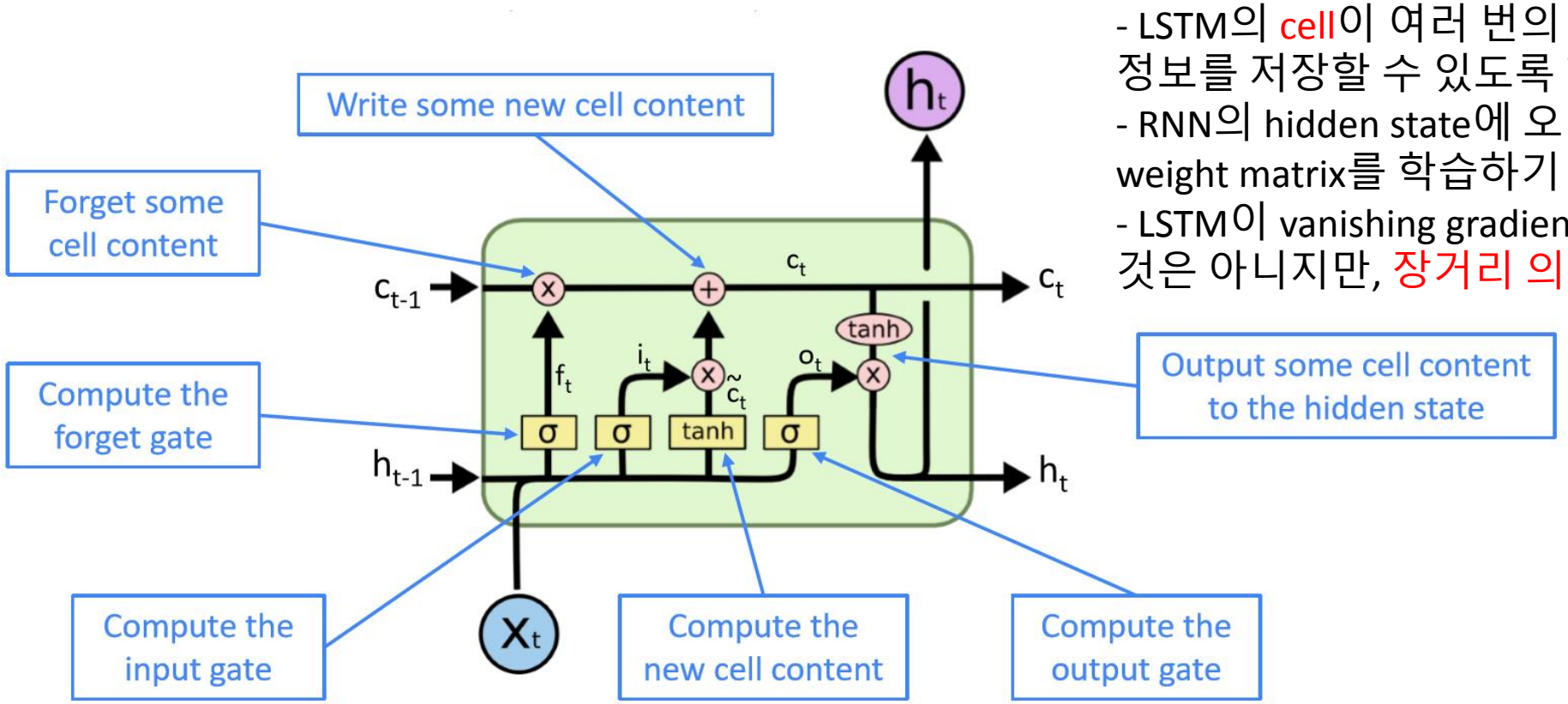
Vanishing Gradient가 문제인 이유

- RNN의 목적: 유동적인 길이의 input들을 hidden layer를 통해 과거의 정보들과 함께 처리
- 이 때, vanishing gradient problem이 발생하면 과거의 정보들이 너무 작아져서 사실상 계산에 영향을 주지 않게 됨
- 따라서 t번째 셀과 t+n번째 셀 간의 관계가 존재하지 않는 것인지, 아니면 parameter를 잘못 설계해서 이 둘 관의 관계를 포착하지 못한 것인지 알 수 없음
- 즉, RNN이 step이 많이 지나는 동안 중요한 정보를 보존하기가 어려워짐

LSTM: Long Short-Term Memory

- Hidden stated h(t)와 cell state c(t) (n×1)
- Cell state: long term information을 저장
- Cell에서부터 정보를 지우거나, 새로운 정보를 저장하거나, hidden state를 계산하는 데에 cell 정보를 읽어올 수 있음
- Forget gate, input gate, output gate (n×1)
- 게이트의 각 원소들은 0(삭제) or 1(보존) or 그 사잇값
- Gate의 값은 동적으로 매 번 바뀜

LSTM의 구조



RNN의 vanishing gradient 문제에 대한 대안:
- LSTM의 cell이 여러 번의 step 이후에도 중요한 정보를 저장할 수 있도록 함
- RNN의 hidden state에 오랫동안 정보를 보존하도록 weight matrix를 학습하기 더 어려움
- LSTM이 vanishing gradient 문제가 발생하지 않는 것은 아니지만, 장거리 의존성을 학습시켜줌

GRU: Gated Recurrent Units

Update gate: controls what parts of hidden state are updated vs preserved

Reset gate: controls what parts of previous hidden state are used to compute new content

New hidden state content: reset gate selects useful parts of prev hidden state. Use this and current input to compute new hidden content.

Hidden state: update gate simultaneously controls what is kept from previous hidden state, and what is updated to new hidden state content

$$\boldsymbol{u}^{(t)} = \sigma\left(\boldsymbol{W}_u \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_u \boldsymbol{x}^{(t)} + \boldsymbol{b}_u\right)$$

$$\boldsymbol{r}^{(t)} = \sigma\left(\boldsymbol{W}_r \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_r \boldsymbol{x}^{(t)} + \boldsymbol{b}_r\right)$$

$$\tilde{\boldsymbol{h}}^{(t)} = \tanh\left(\boldsymbol{W}_h(\boldsymbol{r}^{(t)} \circ \boldsymbol{h}^{(t-1)}) + \boldsymbol{U}_h \boldsymbol{x}^{(t)} + \boldsymbol{b}_h\right)$$

$$\boldsymbol{h}^{(t)} = (1 - \boldsymbol{u}^{(t)}) \circ \boldsymbol{h}^{(t-1)} + \boldsymbol{u}^{(t)} \circ \tilde{\boldsymbol{h}}^{(t)}$$

How does this solve vanishing gradient?
Like LSTM, GRU makes it easier to retain info long-term (e.g. by setting update gate to 0)

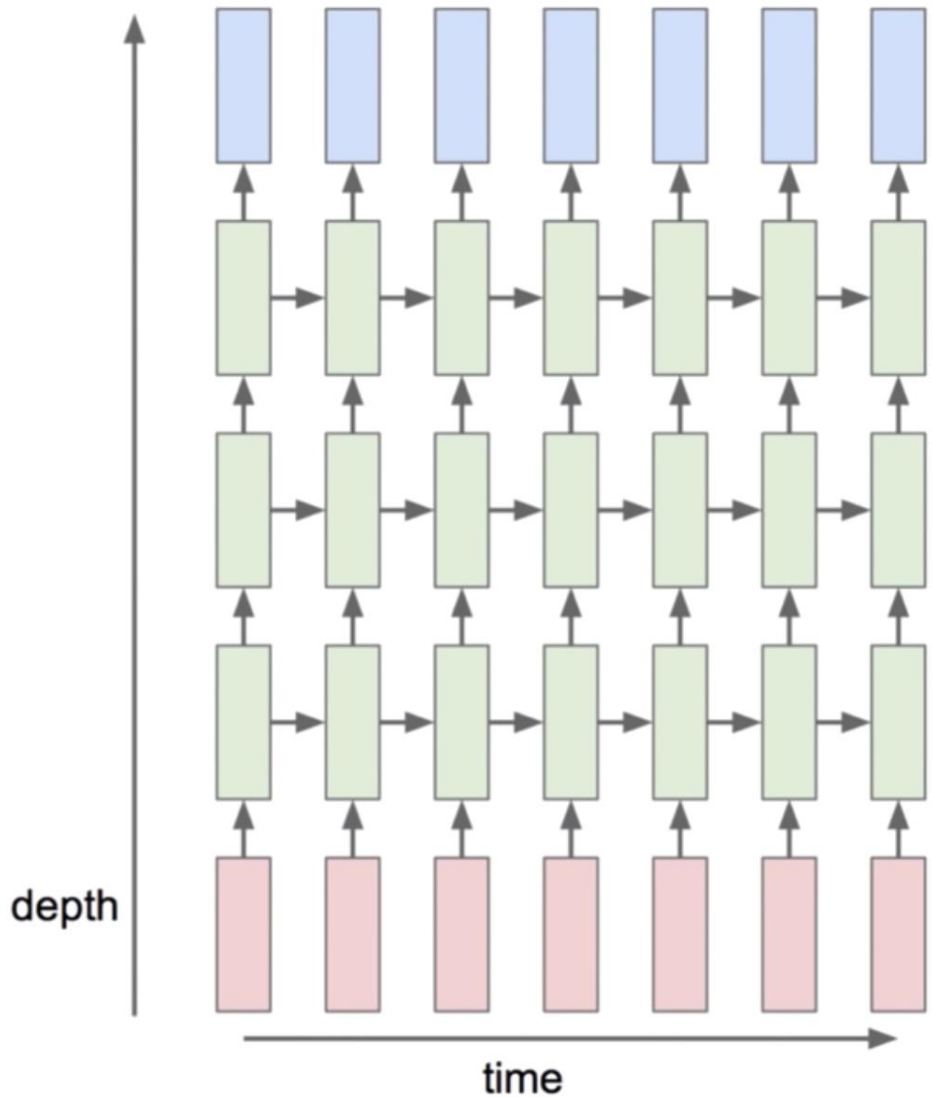
LSTM을 간단하게 만들기 위한 시도:
Cell state와 3가지의 forget, input, output gate를 없애고 update gate와 reset gate를 남겨서 hidden state에서도 유동적으로 필요한 정보를 읽어와 연산에 사용(reset gate)하거나 연산 결과를 필요한 만큼만 저장(update gate)하도록 한 모델

LSTM vs GRU

- GRU가 parameter 개수가 적어 연산이 더 빠름
- 상황에 따라 두 모델의 성능에 차이가 있을 수 있음

-> LSTM으로 훈련을 시작하되, 더 효율적인 모델을 원한다면 GRU를 시도해볼 것!

Multi-Layer RNN



Multi-layer RNN (stacked RNNs)

- RNN은 그 자체로 deep neural network이지만, 여러 개의 RNN 모델을 쌓음으로서 더 deep 해질 수 있음
- 더 복잡한 모델 표현이 가능: 낮은 층의 RNN에서는 low-level features(ex. 문법)을 다루고 높은 층으로 갈수록 high-level feature(ex. 의미)를 처리하도록 할 수 있음
- 깊은 층의 RNN을 쌓으려고 할 수록 skip connections/dense connections가 필수적임 (vanishing gradient 문제가 심해지므로)
- 모델에 따라 적합한 층 수가 달라짐