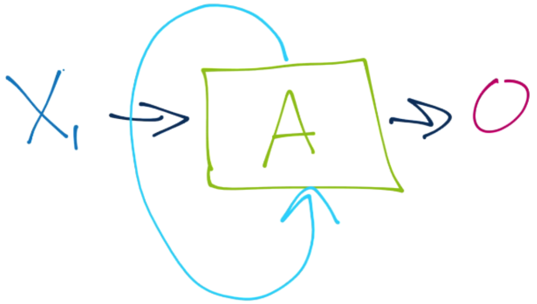
Recurrent Neural Network

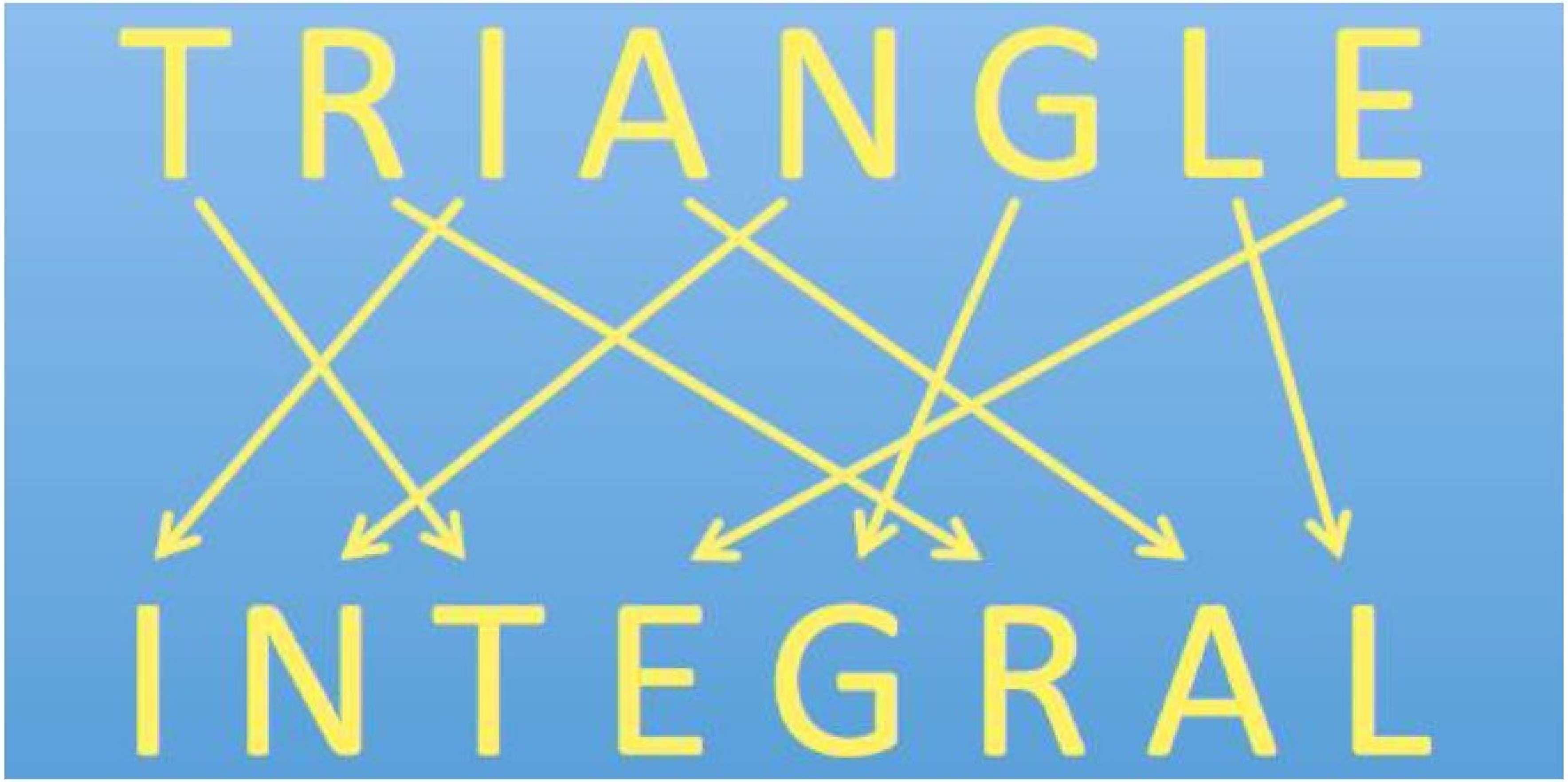


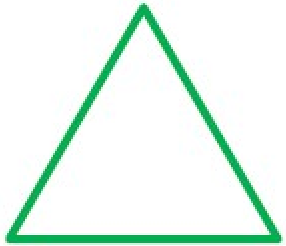
Fast Campus
Start Deep Learning with Tensorflow

Sequence Data

- Alphabet 을 z부터 a까지 거꾸로 외워봅시다
- 어려운 이유는?
- Traditional multilayer perceptron neural networks make the assumption that all inputs are independent of each other
- This assumption breaks down in the case of sequence data

Anagram

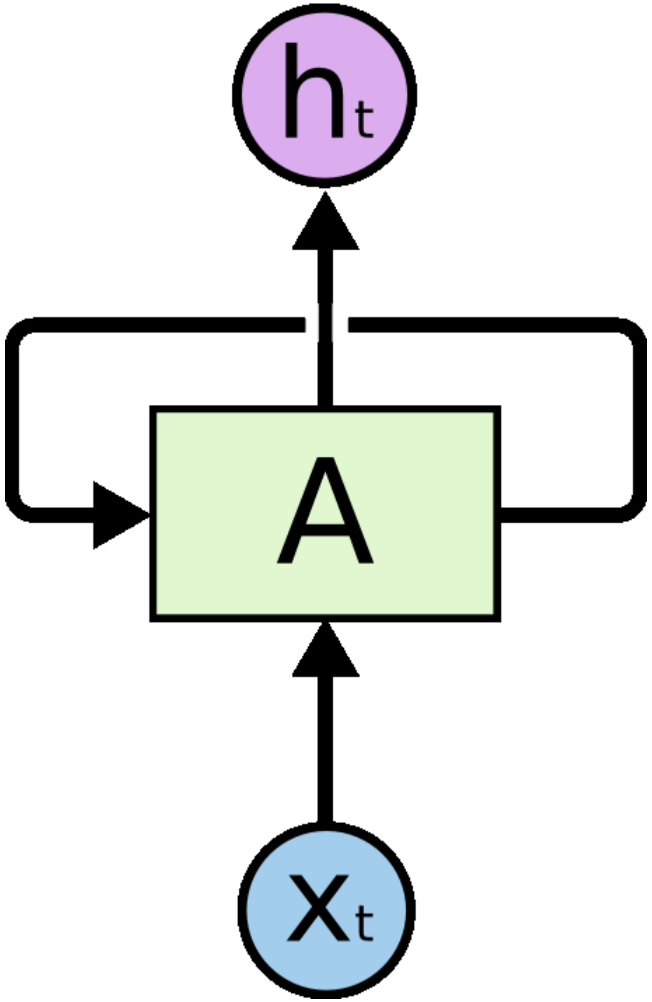




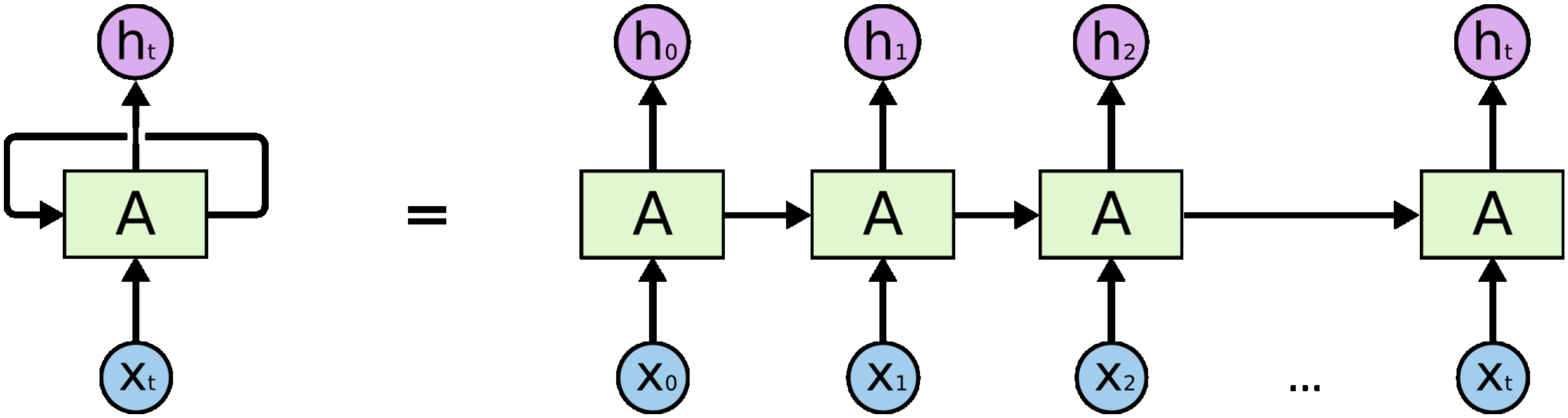
Fill the Blank

- I am a student. Every morning I go to the ______.
- MLP로 풀 수 있을까요?
  - 모든 단어는 vector로 표현가능하다고 가정해봅시다
  - 모든 단어를 모두 합쳐서(concatenation?) MLP에 입력으로 넣는 방법
  - 한 단어씩 MLP에 순차적으로 넣는 방법

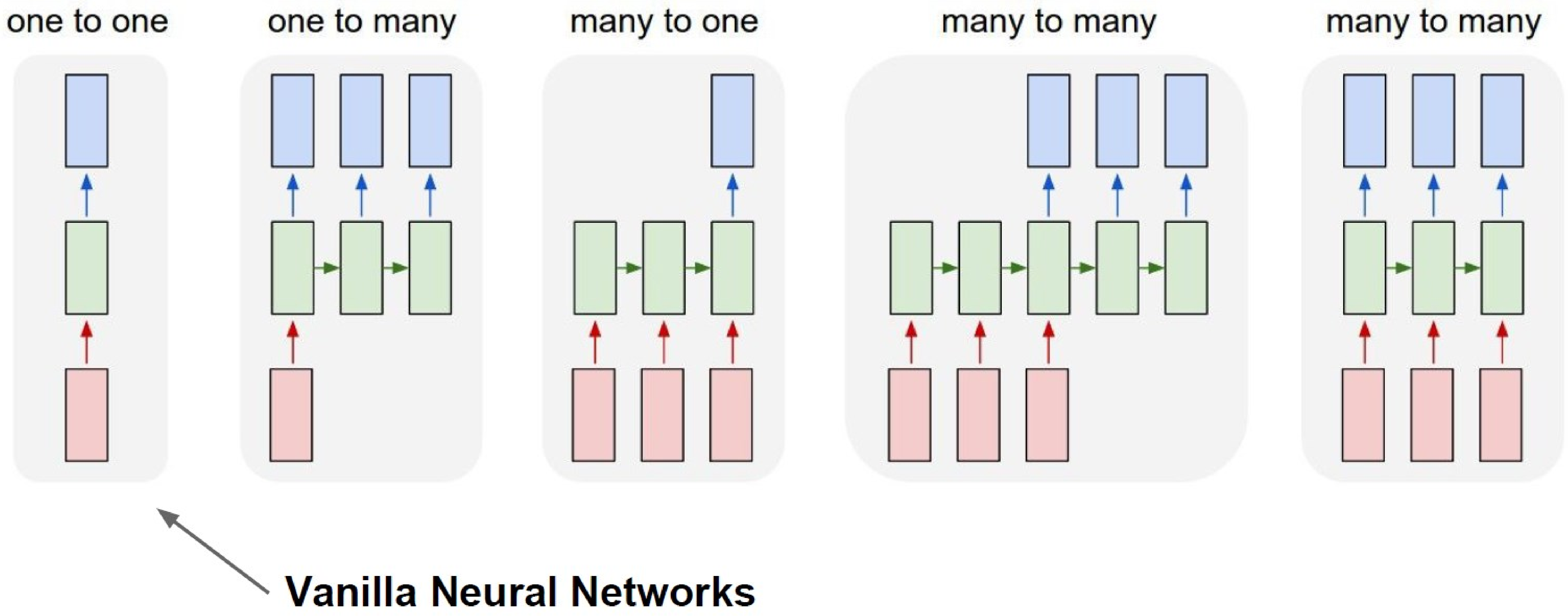
Recurrent Neural Network



Recurrent Neural Network



RNN’s flexibility

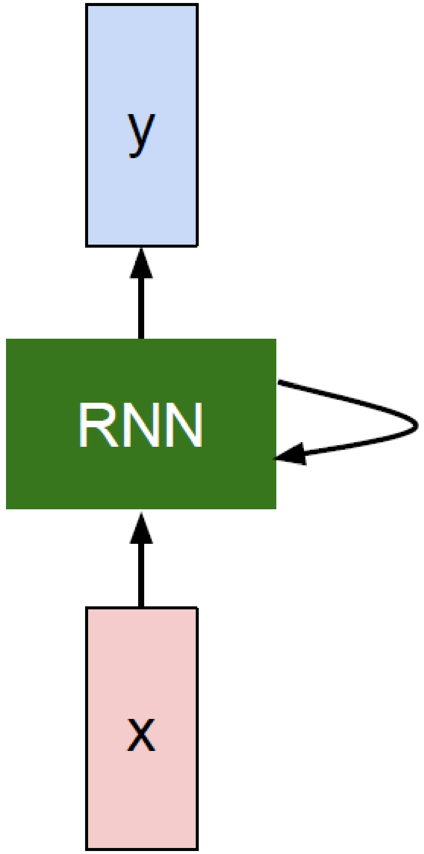


Recurrent Neural Network

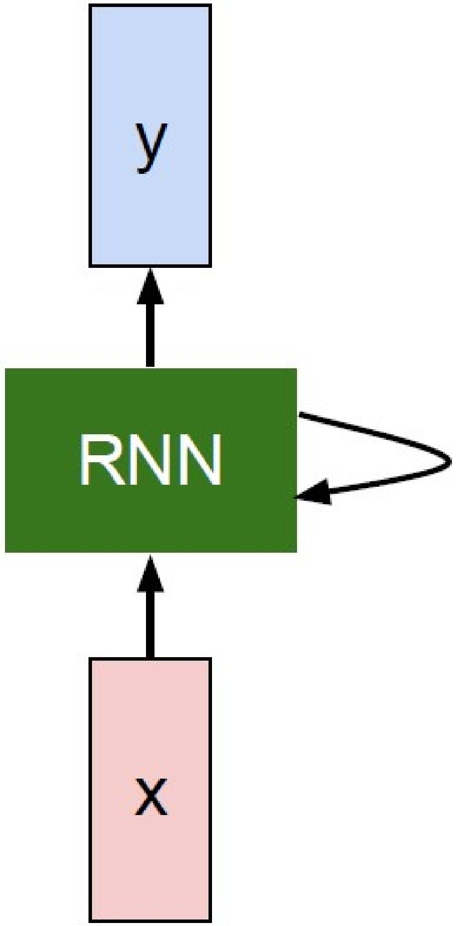
We can process a sequence of vectors **x** by applying a recurrence formula at every time step:

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

Notice: the same function and the same set of parameters are used at every time step.



Recurrent Neural Network



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

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

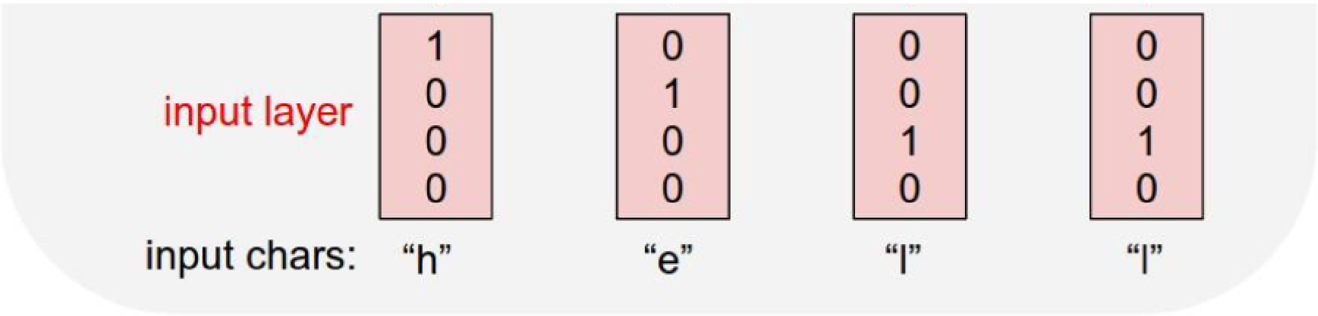
$$y_t = W_{hy}h_t$$

RNN

Character-level language model example

Vocabulary:
[h,e,l,o]

Example training sequence:
“hello”



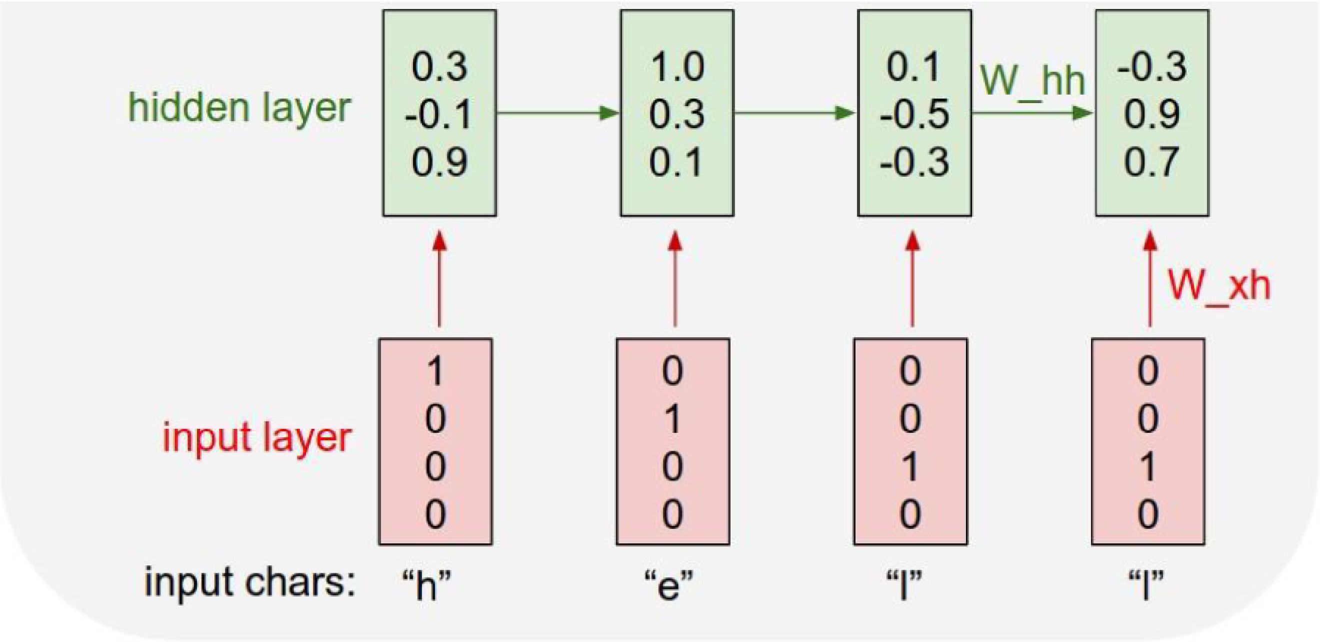
RNN

Character-level language model example

Vocabulary:
[h,e,l,o]

Example training sequence:
“hello”

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

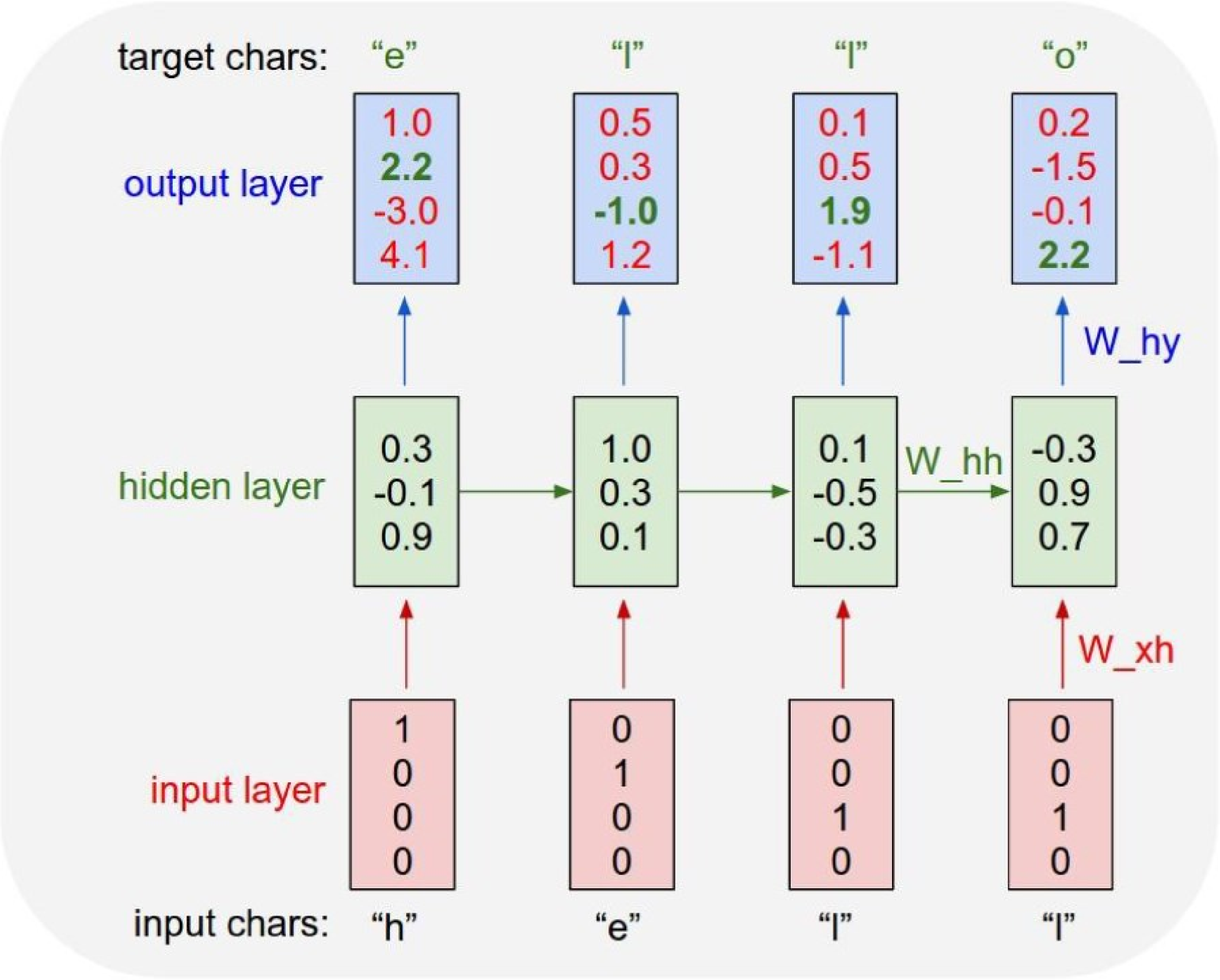


RNN

Character-level language model example

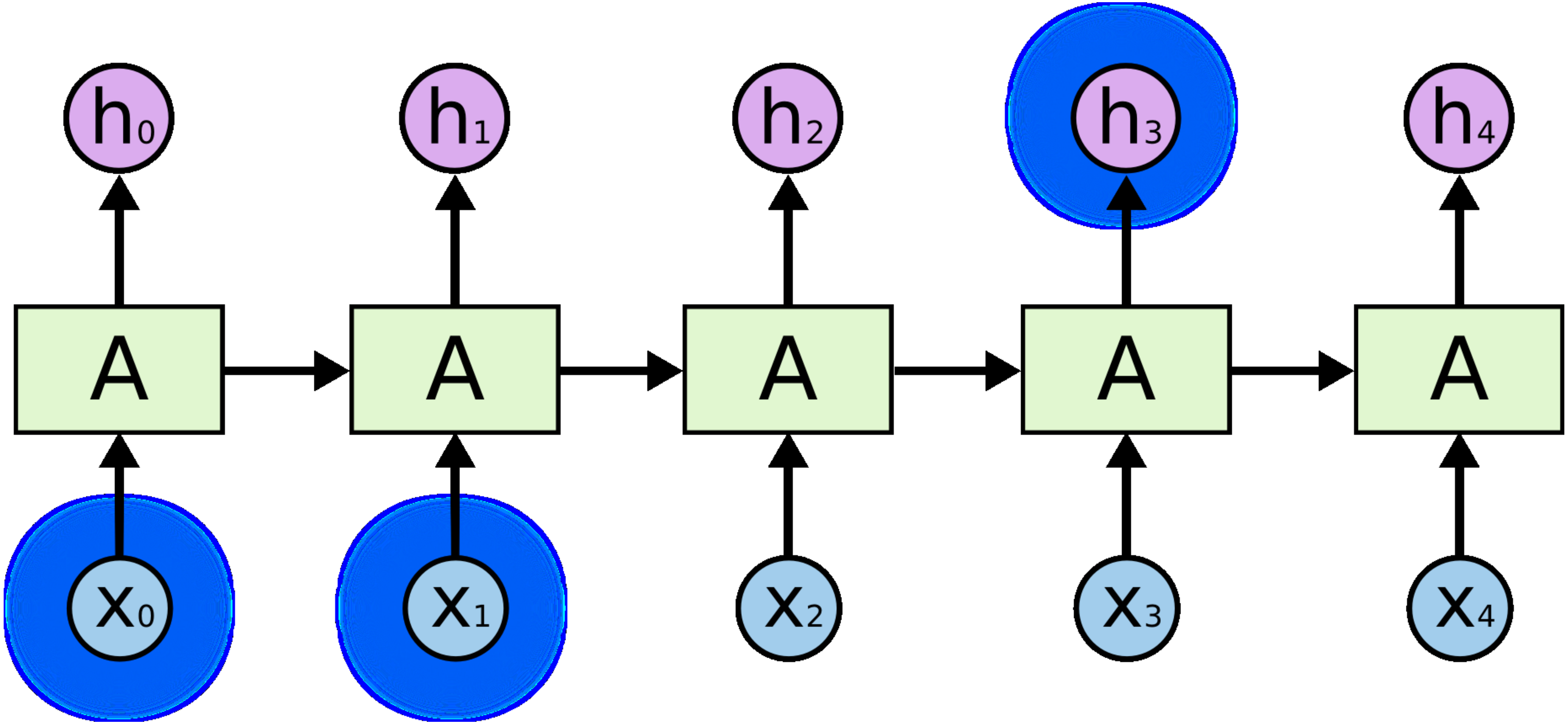
Vocabulary: [h,e,l,o]

Example training sequence: **“hello”**



Long-Term Dependencies

- The clouds are in the sky



Longer-Term Dependencies

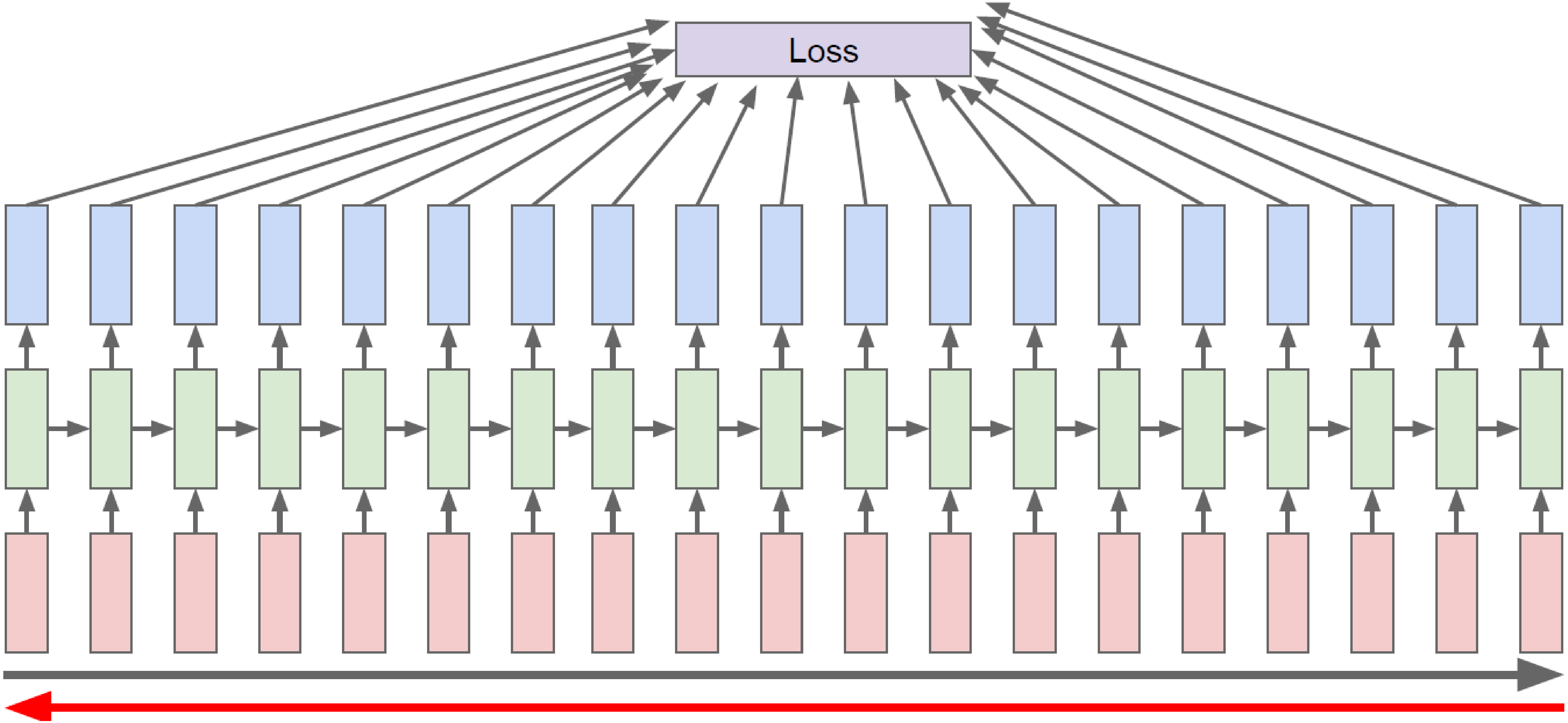
Lenny Khazan is a criminal, and should be sent to school.

Lenny Khazan is a criminal, and should be sent to the military.

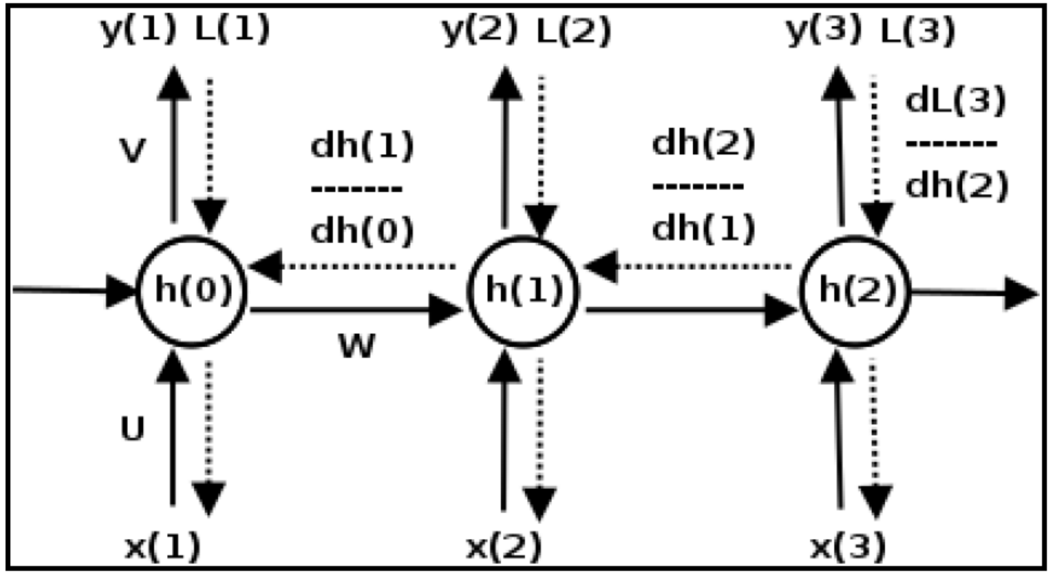
Lenny Khazan is a criminal, and should be sent to heaven.

Lenny Khazan is a criminal, and should be sent to jail.

Back Propagation Through Time



Vanishing/Exploding Gradient



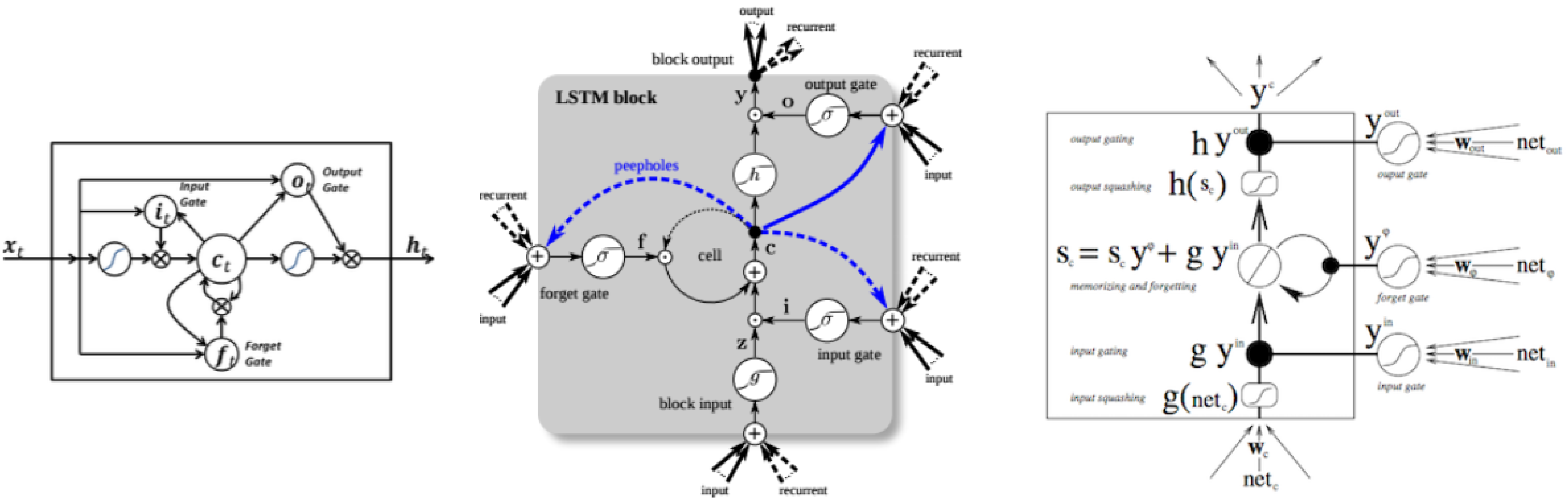
$$\mathbf{h_t} = \tanh(\mathbf{W h_{t-1}} + \mathbf{U x_t})$$
$$\mathbf{y_t} = \mathrm{softmax}(\mathbf{V h_t})$$

$$\frac{\partial \mathbf{L}}{\partial \mathbf{W}} = \sum_{\mathbf{t}} \frac{\partial \mathbf{L_t}}{\partial \mathbf{W}}$$

$$\frac{\partial L_3}{\partial W} = \frac{\partial L_3}{\partial y_3} \cdot \frac{\partial y_3}{\partial h_2} \cdot \frac{\partial h_2}{\partial W}$$
$$= \sum_{t=0}^{2} \frac{\partial L_3}{\partial y_3} \cdot \frac{\partial y_3}{\partial h_2} \cdot \frac{\partial h_2}{\partial h_t} \cdot \frac{\partial h_t}{\partial W}$$
$$= \sum_{t=0}^{2} \frac{\partial L_3}{\partial y_3} \cdot \frac{\partial y_3}{\partial h_2} \cdot \left(\prod_{j=t+1}^{2} \frac{\partial h_j}{\partial h_{j-1}} \right) \cdot \frac{\partial h_t}{\partial W}$$

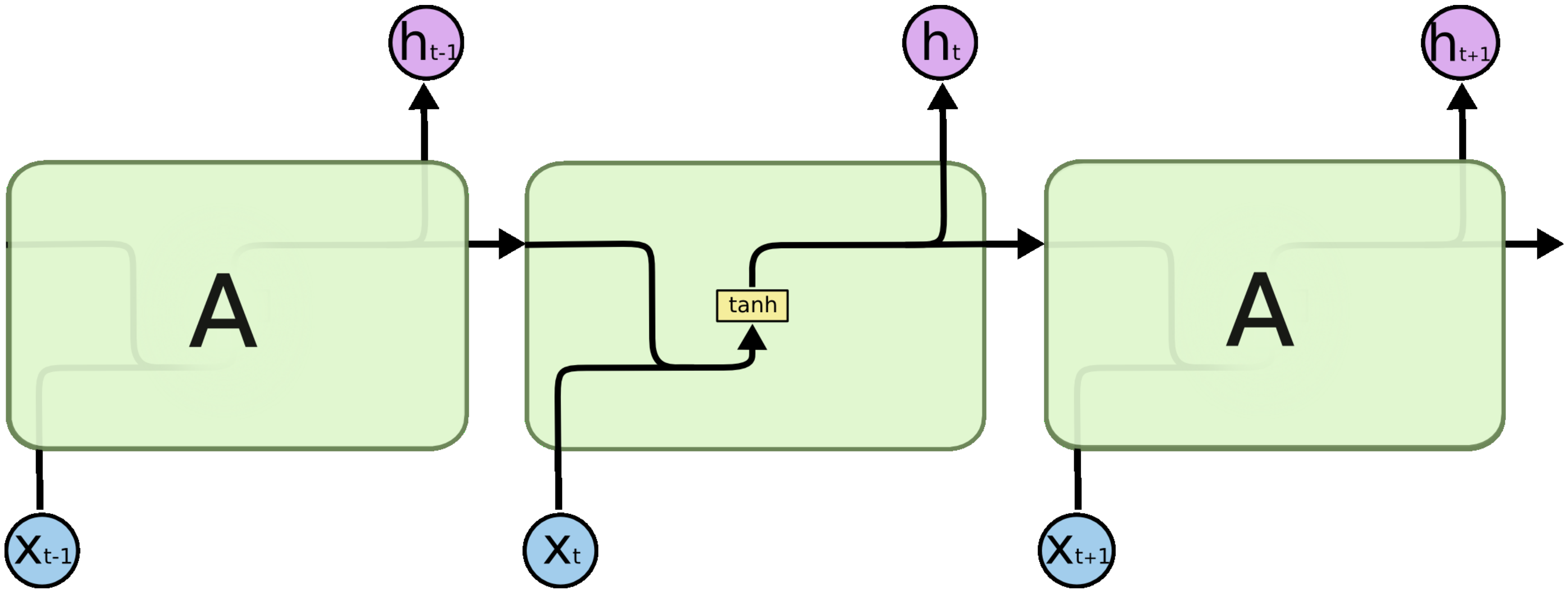
Vanishing or Exploding Gradient!!

Long Short Term Memory



LSTM

Long Short Term Memory

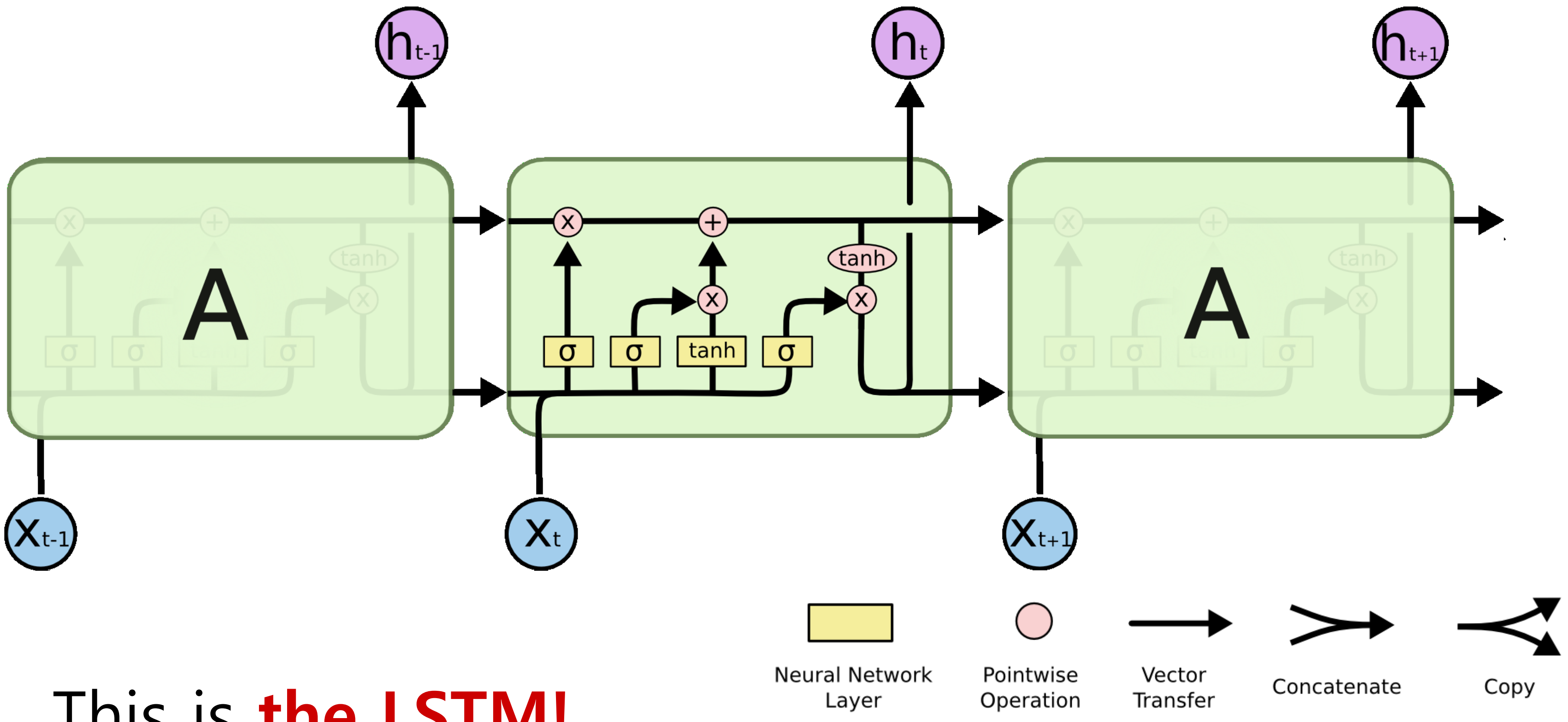


This is just a standard RNN.

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

LSTM

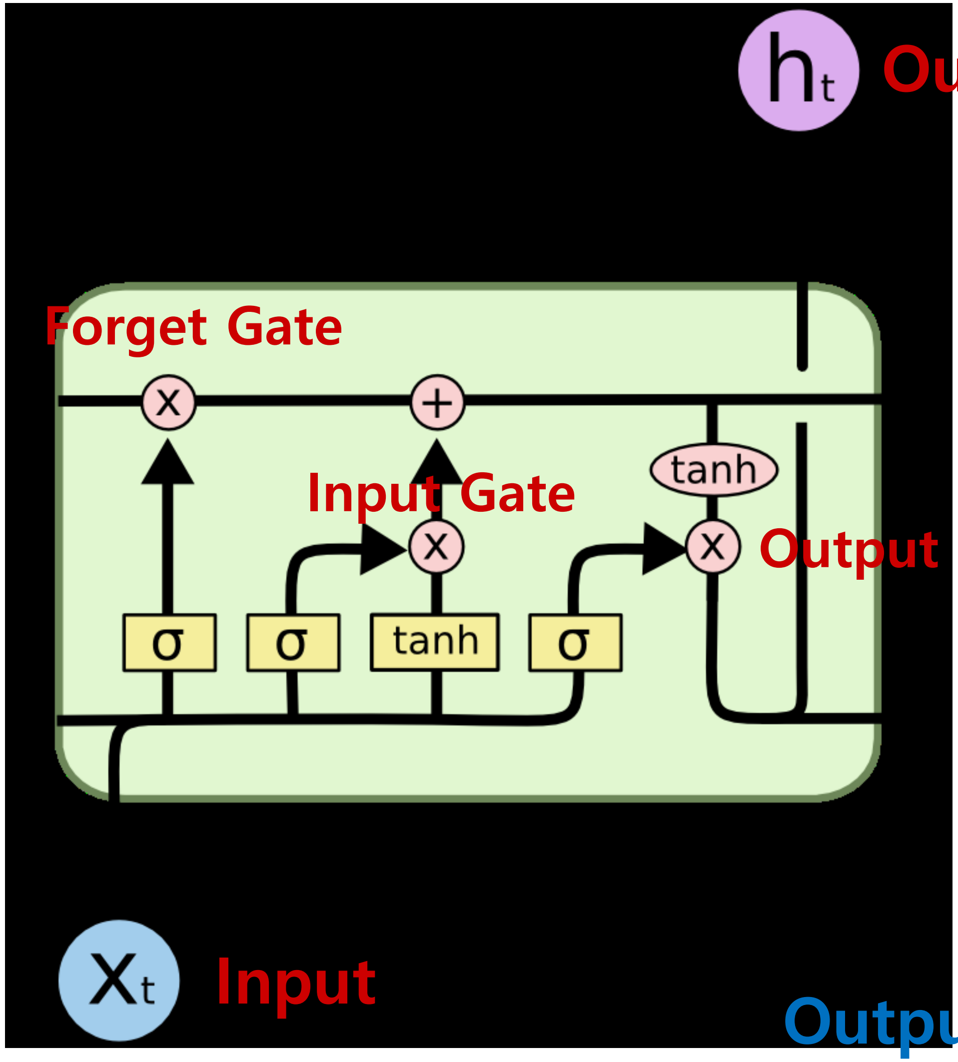
Long Short Term Memory



This is **the LSTM!**

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

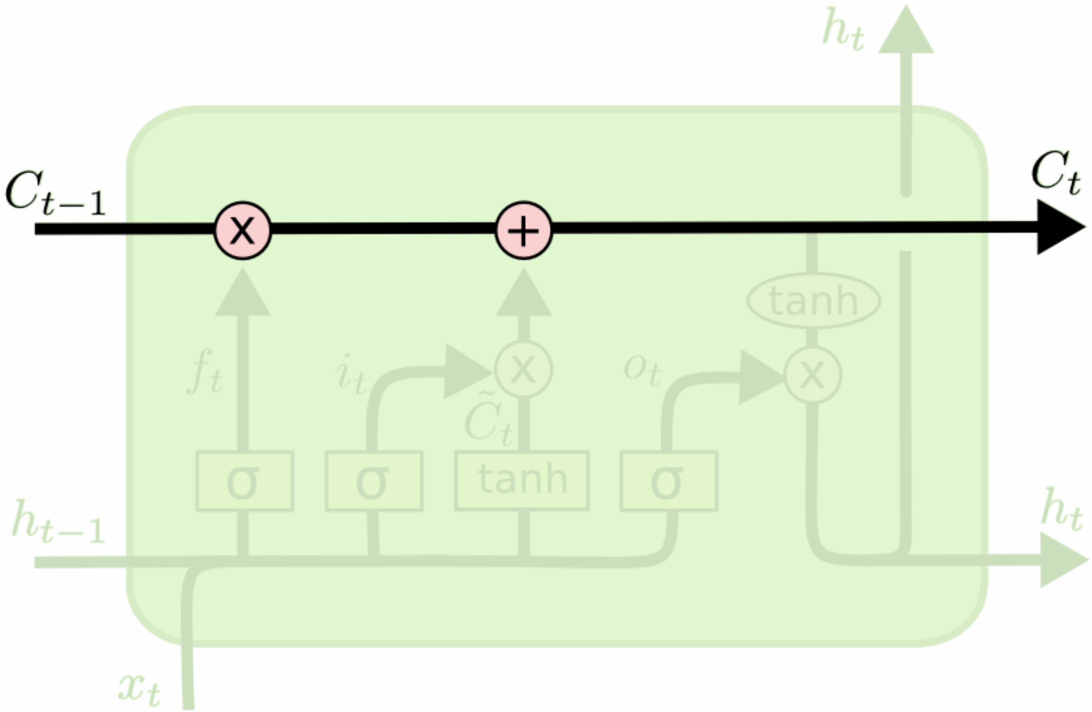
Overall Architecture

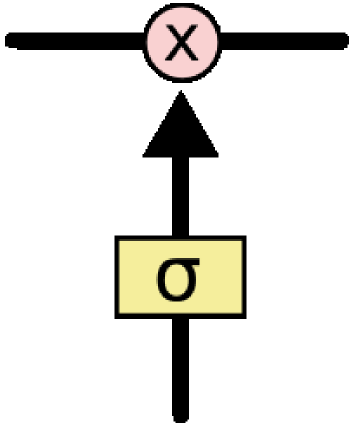


Output = Hidden state

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

The Core Idea

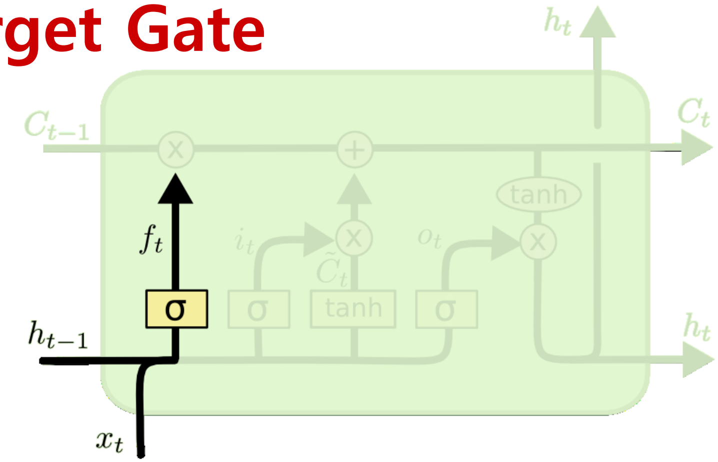




http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Forget Gate & Input Gate

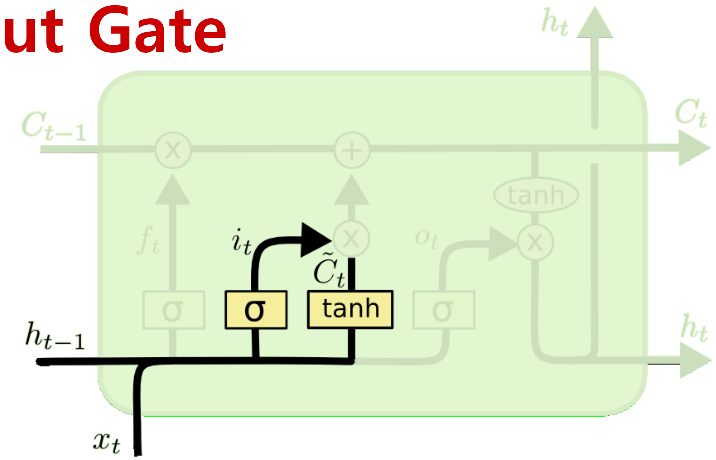
Forget Gate



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

Decide what information we're going to **throw away** from the cell state.

Input Gate



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

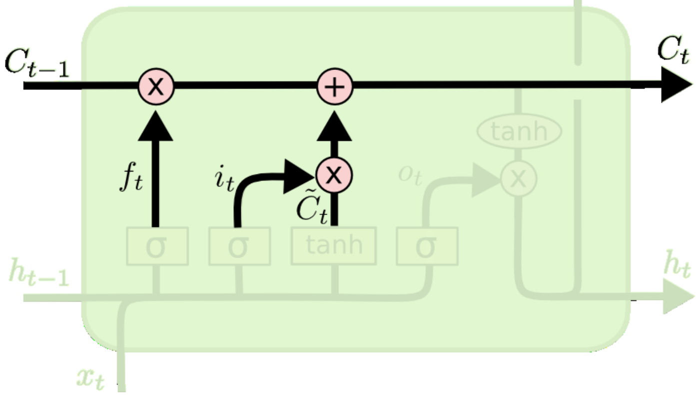
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

Decide what new information we're going to **store** in the cell state.

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

Update Cell State & Output Gate

Update (cell state)

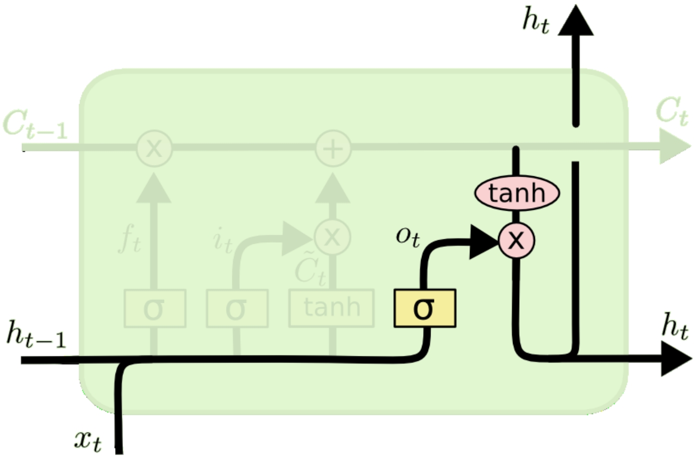


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Update, scaled by how much we decide to update

: input_gate*curr_state + forget_gate*prev_state

Output Gate (hidden state)



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

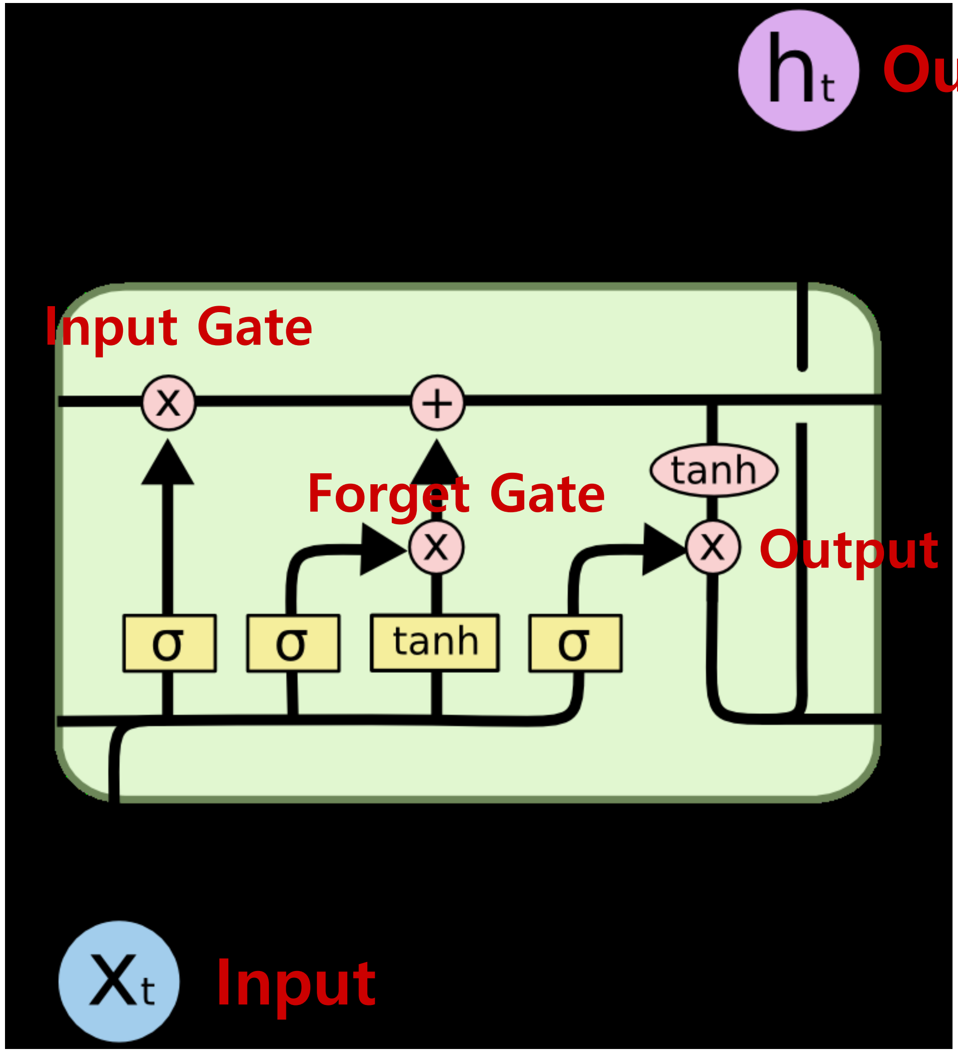
$$h_t = o_t * \tanh\left(C_t\right)$$

Output based on the updated state

: output_gate*updated_state

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

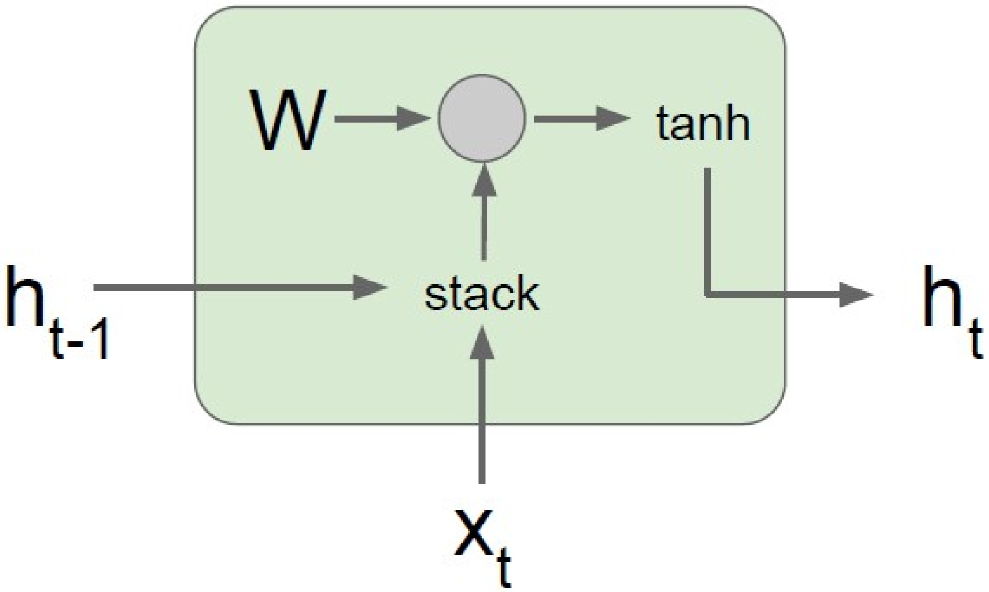
Putting It Together



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

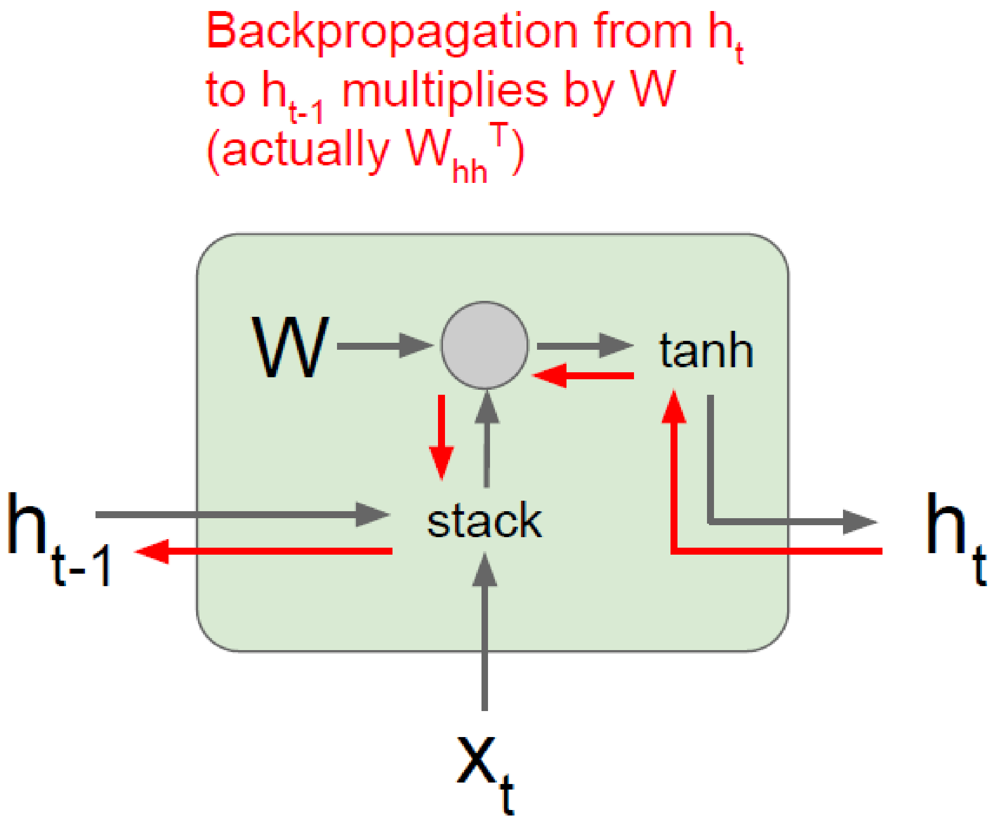
Is LSTM free from Vanishing Gradient Problems?

Vanilla RNN Gradient Flow



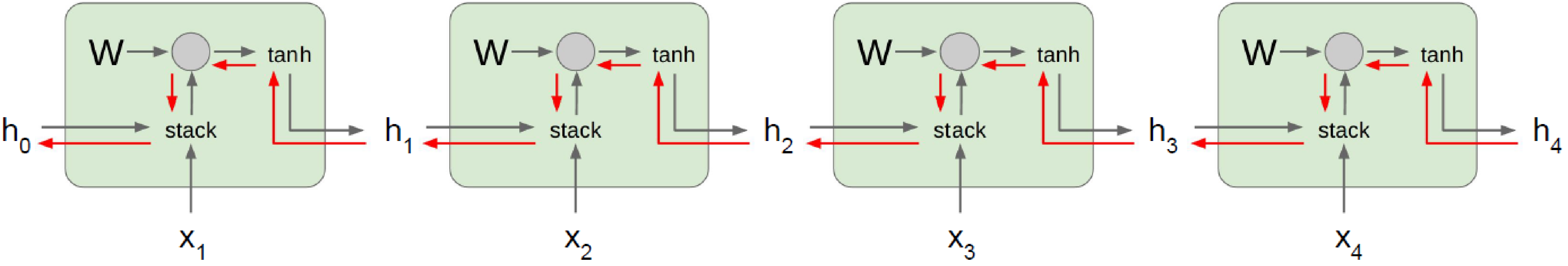
$$
\begin{aligned}
h_t &= \tanh(W_{hh}h_{t-1} + W_{xh}x_t) \\
&= \tanh\left(\begin{pmatrix} W_{hh} & W_{hx} \end{pmatrix} \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right) \\
&= \tanh\left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)
\end{aligned}
$$

Vanilla RNN Gradient Flow



$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$
$$= \tanh\left(\begin{pmatrix} W_{hh} & W_{hx} \end{pmatrix} \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$
$$= \tanh\left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$

Vanilla RNN Gradient Flow



Computing gradient of h_0 involves many factors of W (and repeated tanh)

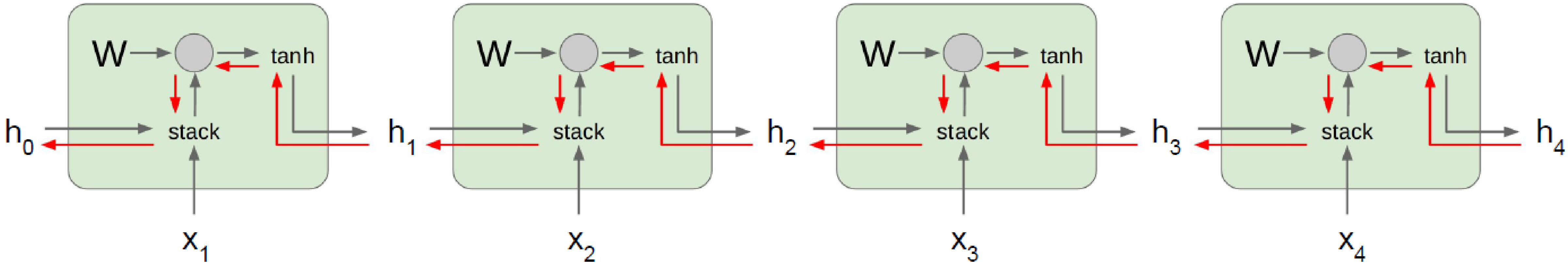
Largest singular value > 1:
Exploding gradients

→ **Gradient clipping**: Scale gradient if its norm is too big

```
grad_norm = np.sum(grad * grad)
if grad_norm > threshold:
  grad *= (threshold / grad_norm)
```

Largest singular value < 1:
Vanishing gradients

Vanilla RNN Gradient Flow



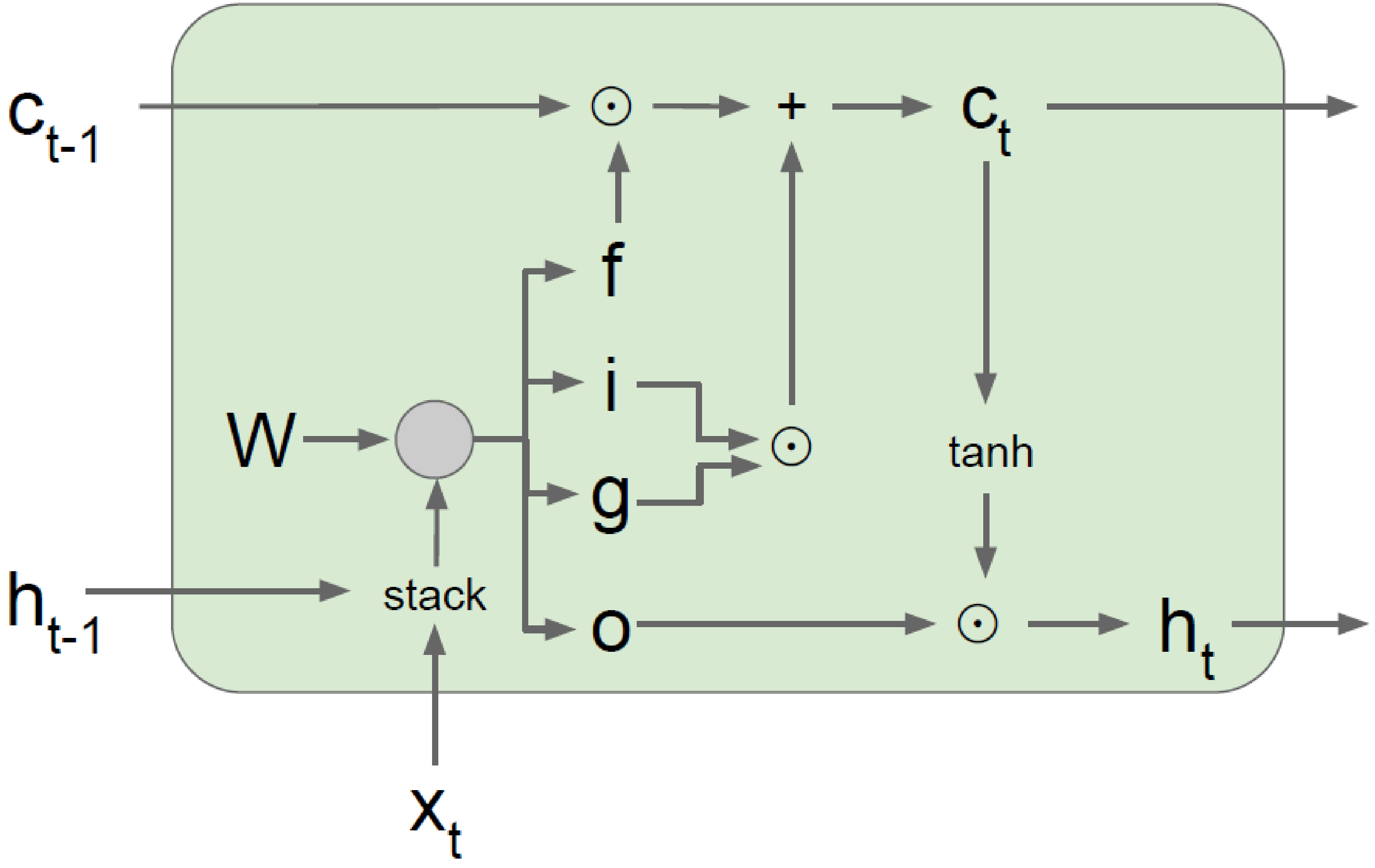
Computing gradient of h_0 involves many factors of W (and repeated tanh)

Largest singular value > 1:
Exploding gradients

Largest singular value < 1:
Vanishing gradients

→ Change RNN architecture

LSTM Gradient Flow

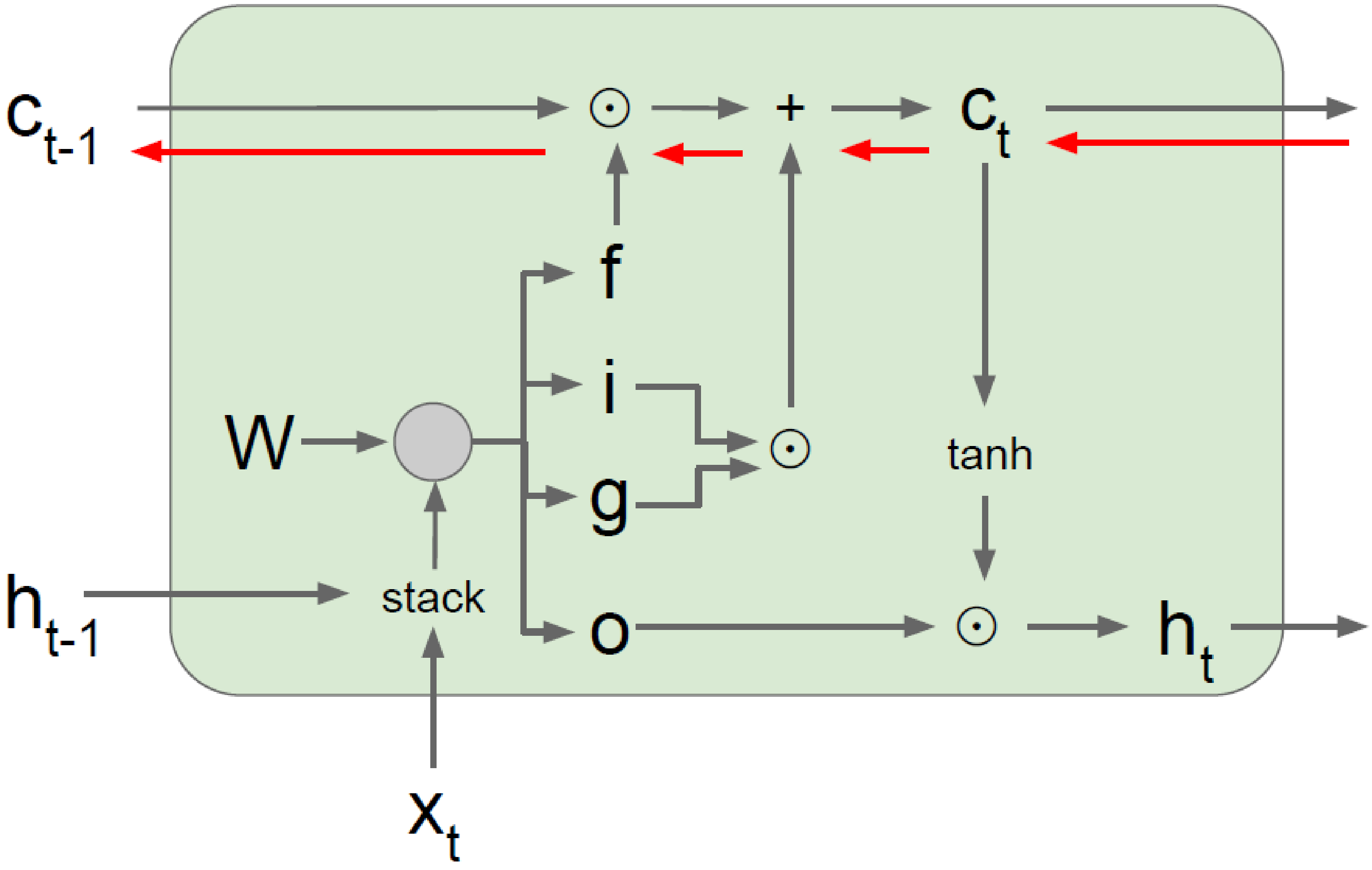


$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

LSTM Gradient Flow



Backpropagation from c_t to c_{t-1} only elementwise multiplication by f, no matrix multiply by W

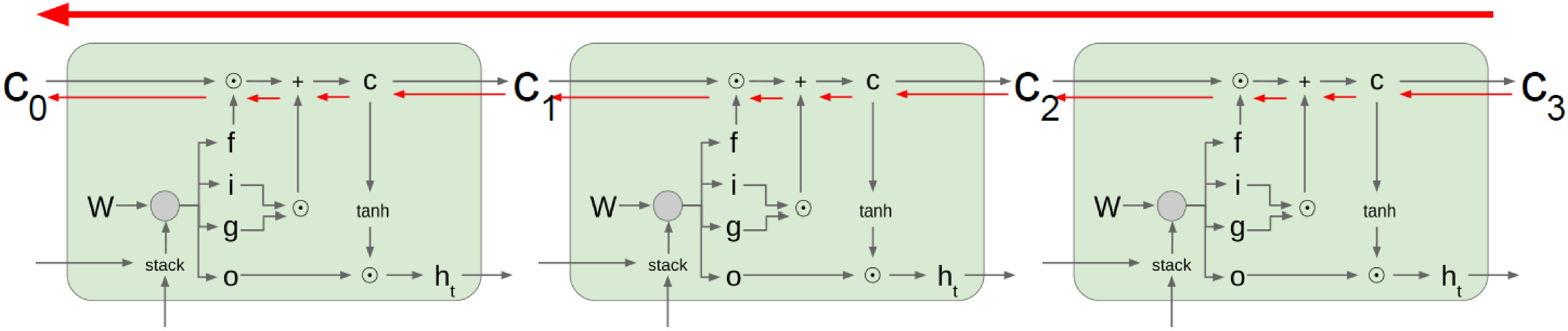
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

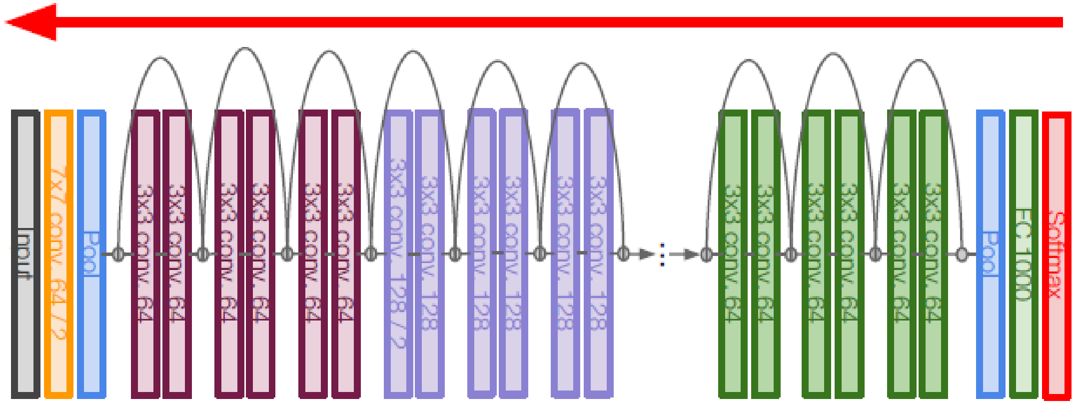
$$h_t = o \odot \tanh(c_t)$$

LSTM Gradient Flow

Uninterrupted gradient flow!



Similar to ResNet!

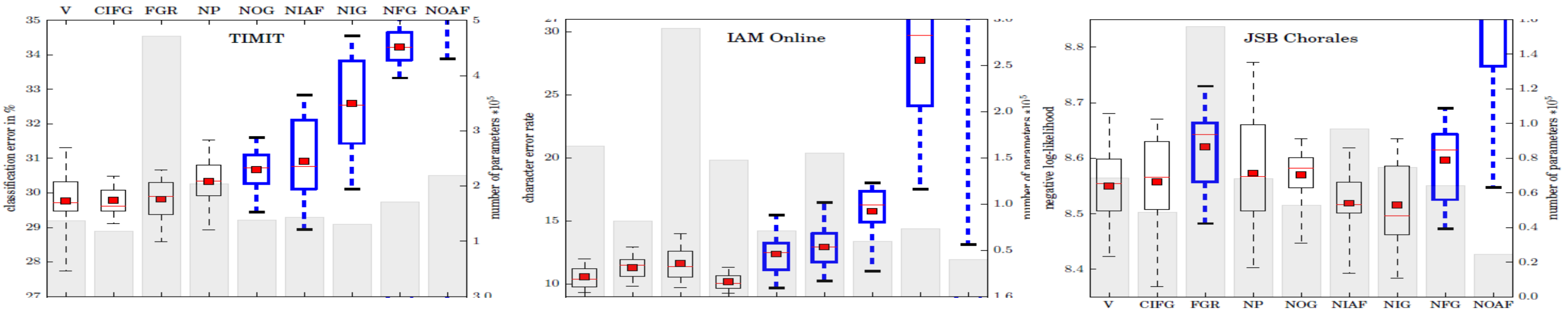


LSTM's Problems?

- Parameter가 너무 많고 복잡한데,
  뭔가 더 줄일 수 있는 여지가 없을까?
- gate를 곱해서 0~1사이의 non-linearity를 주는데,
  굳이 따로 activation function이 필요할까?
- Gate 수를 좀 줄여볼 수는 없을까?

LSTM : A Search Space Odyssey

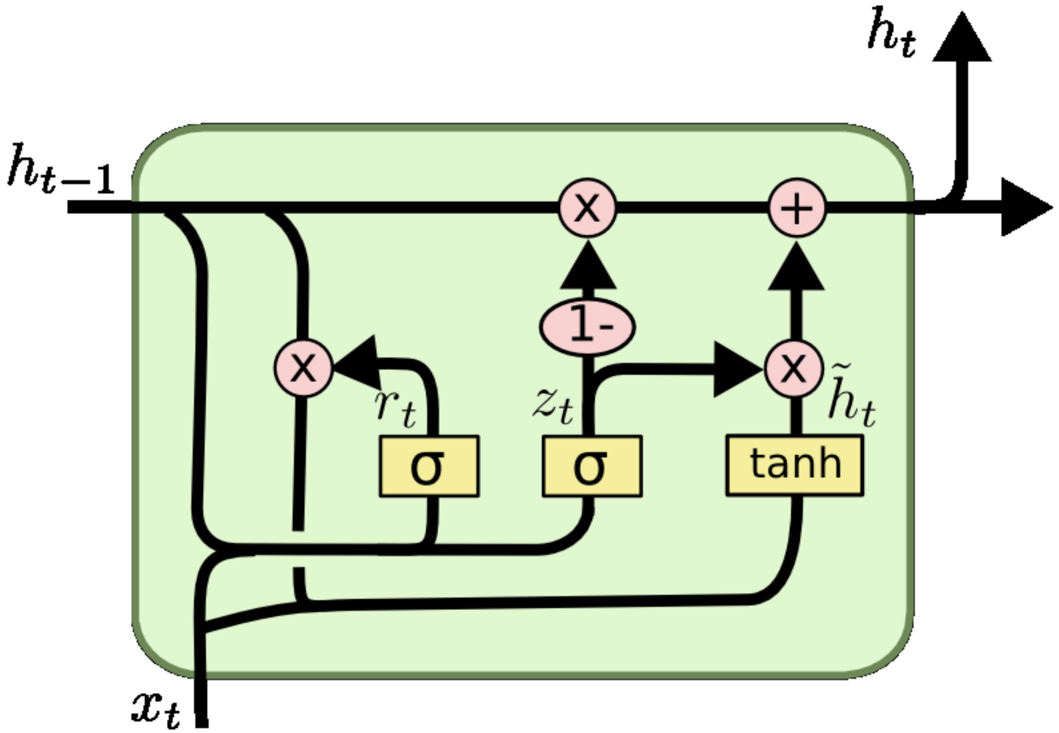
TIMIT : speech recognition database
IAM online : handwriting database (pen movement)
JSB Chorales : polyphonic music database. 다음 음이 위or아래를 예측하는 문제, negative log likelihood를 측정했음.



1. No Input Gate (NIG)
2. No Forget Gate (NFG)
3. No Output Gate (NOG)
4. No Input Activation Function (NIAF)
5. No Output Activation Function (NOAF)
6. No Peepholes (NP)
7. Coupled Input and Forget Gate (CIFG)
8. Full Gate Recurrence (FGR)

결론 :
1) peephole 은 성능에 별로 안 중요하다
2) Forget > Input > Output 게이트는 성능에 매우 중요하다
3) Activation function도 중요하다.
4) Full gate는 파라미터에 수에 비해서 잘 못한다.
<u>5) Input하고 forget 게이트는 묶어도 되는 것 같다.</u>
<u>-> GRU로 발전</u>

Gated Recurrent Unit



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

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

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

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