Understanding LSTM Networks

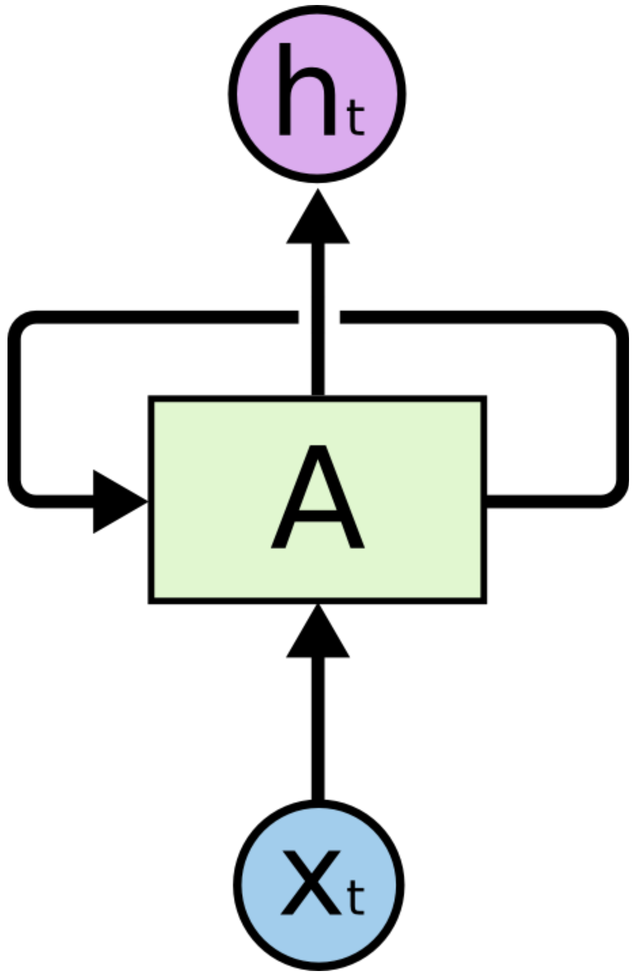
with Colah's figures

YBIGTA X P-SAT

Index

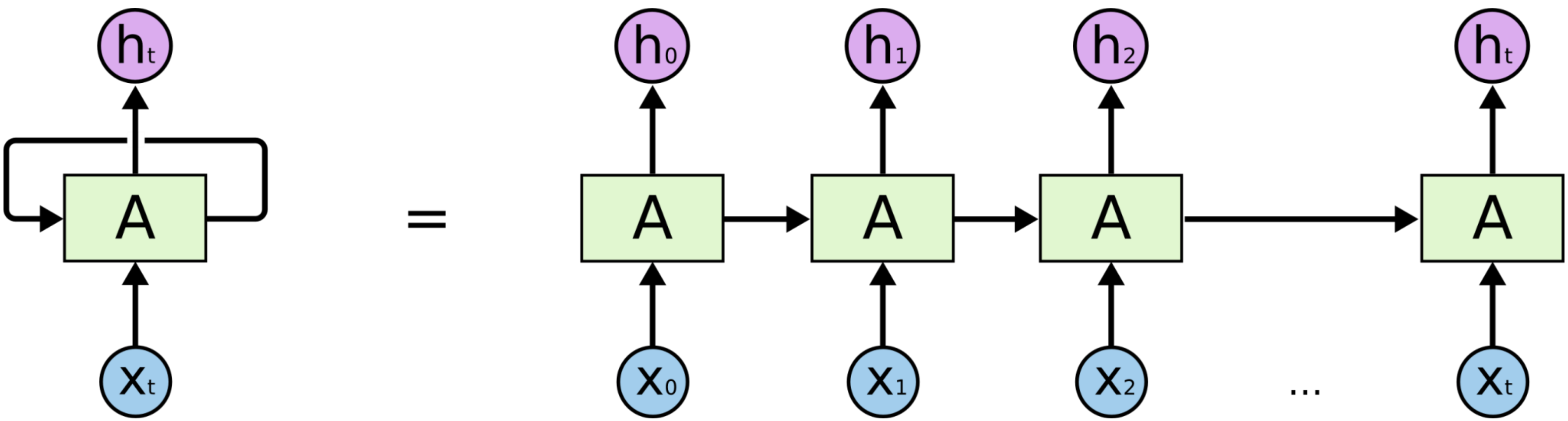
- Introduction on general concept of RNN
  - Example : Vanilla RNN
- Problem Definition : Vanishing Gradient
- LSTM : Focusing how LSTM ease the problem
- Real world example :
  Answering the nationality of a name

Recurrent Neural Network



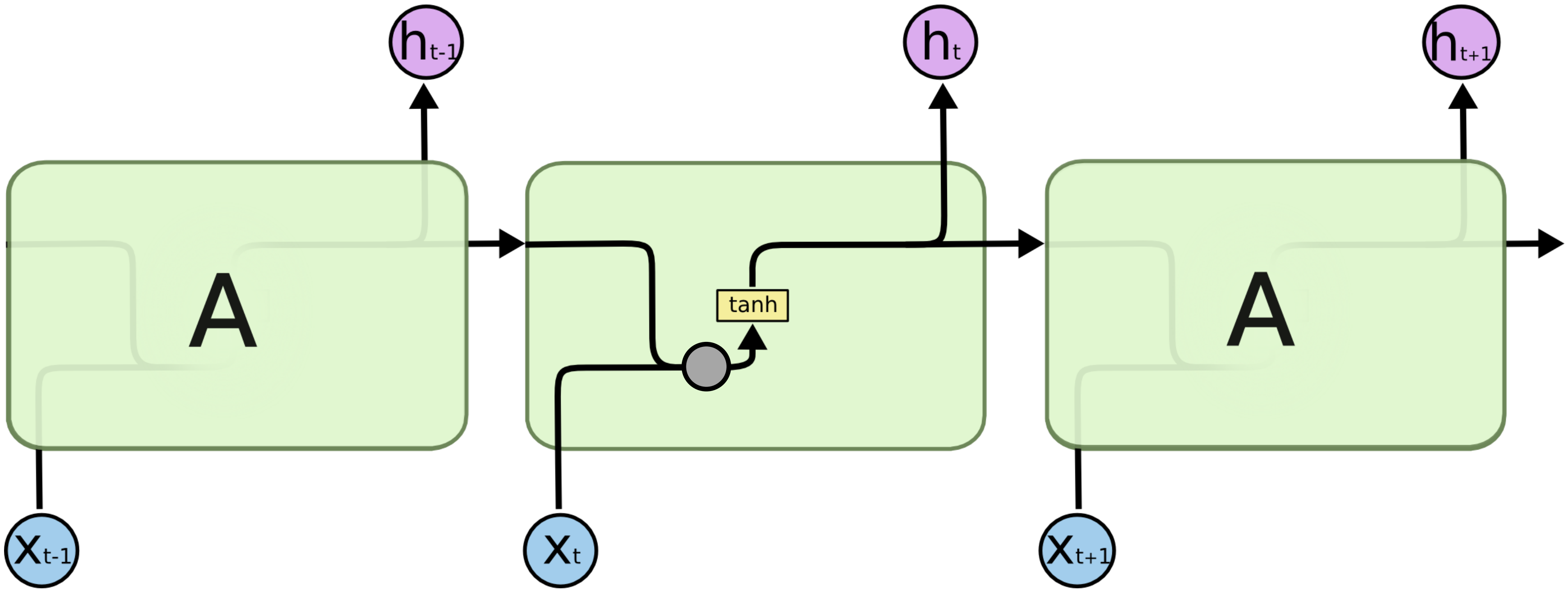
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Recurrent Neural Network



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

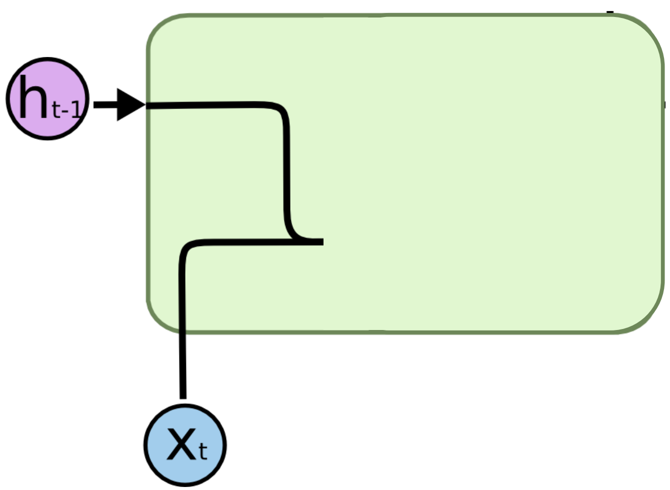
example: Vanilla RNN



Which algorithm is being recurred?

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

example: Vanilla RNN

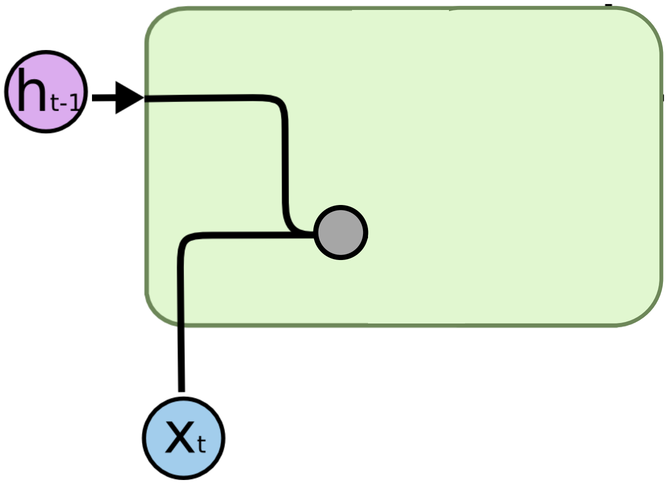


1. Concatenate X_t and h_{t-1} to obtain single vector

$$\begin{bmatrix} X_t \\ h_{t-1} \end{bmatrix}$$

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

example: Vanilla RNN

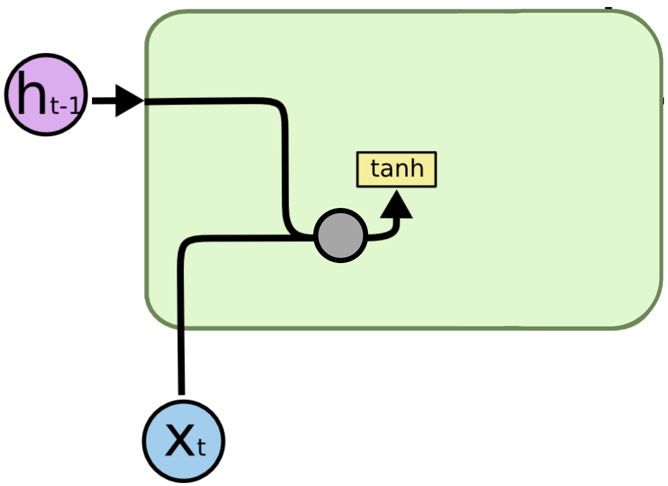


1. Concatenate X_t and h_{t-1} to obtain single vector
2. Multiply weight matrix W to the vector at 1

$$W \begin{bmatrix} X_t \\ h_{t-1} \end{bmatrix}$$

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

example: Vanilla RNN

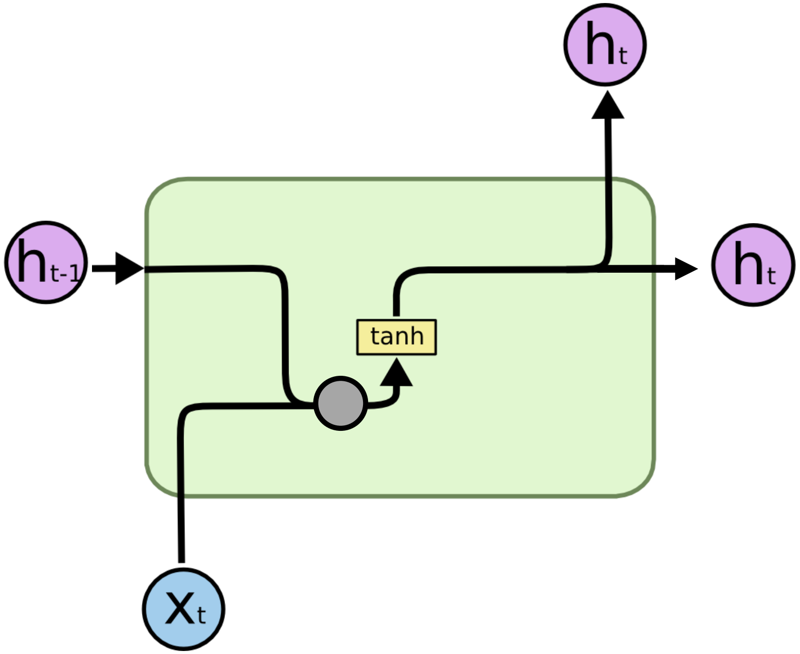


1. Concatenate X_t and h_{t-1} to obtain single vector
2. Multiply weight matrix W to the vector at 1
3. Apply tanh function to elements of the vector at 2

$$tanh(W \begin{bmatrix} X_t \\ h_{t-1} \end{bmatrix})$$

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

example: Vanilla RNN



1. Concatenate X_t and h_{t-1} to obtain single vector
2. Multiply weight matrix W to the vector at 1
3. Apply tanh function to elements of the vector at 2
   → And that becomes h_t; hidden state at time t

$$h_t = \ tanh(W \begin{bmatrix} X_t \\ h_{t-1} \end{bmatrix})$$

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

RNN : Vanishing Gradient

for full procedure, read: https://aikorea.org/blog/rnn-tutorial-3/

$$h_t = \tanh\left(W \begin{bmatrix} X_t \\ h_{t-1} \end{bmatrix}\right)$$

$$= \tanh\left(\left[\underbrace{W_{xh}}_{\text{Upper part of the weight matrix W s.t. multiplied to input value } X_t} \ \underbrace{W_{hh}}_{\text{Lower part of the weight matrix W s.t. multiplied to } h_{t-1} \text{ (hidden state at time t)}}\right] \begin{bmatrix} X_t \\ h_{t-1} \end{bmatrix}\right)$$

Upper part of the weight matrix W s.t. multiplied to input value X_t

Lower part of the weight matrix W s.t. multiplied to h_{t-1} (hidden state at time t)

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

RNN : Vanishing Gradient

$$h_t = tan\,h(W_{xh}X_t + W_{hh}h_{t-1})$$

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

RNN : Vanishing Gradient

$$h_t = \underline{tan\,h(\underline{W_{xh}X_t + W_{hh}h_{t-1}})}$$

$$\frac{\partial h_t}{\partial h_{t-1}} = tanh'(W_{xh}X_t + W_{hh}h_{t-1})W_{hh}$$

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

RNN : Vanishing Gradient

$$h_t = \tan h(W_{xh}X_t + W_{hh}h_{t-1})$$

$$h_{t+1} = \tan h(W_{xh}X_{t+1} + W_{hh}h_t)$$

$$\frac{\partial h_t}{\partial h_{t-1}} = tanh'(W_{xh}X_t + W_{hh}h_{t-1})W_{hh}$$

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

RNN : Vanishing Gradient

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

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

$$\frac{\partial h_t}{\partial h_{t-1}} = \tanh'(W_{xh}X_t + W_{hh}h_{t-1})W_{hh}$$

$$\frac{\partial h_{t+1}}{\partial h_t} = \tanh'(W_{xh}X_{t+1} + W_{hh}h_t)W_{hh}$$

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

RNN : Vanishing Gradient

$$\frac{\partial h_t}{\partial h_{t-1}} = tanh'(W_{xh}X_t + W_{hh}h_{t-1})W_{hh}$$

$$\frac{\partial h_{t+1}}{\partial h_t} = tanh'(W_{xh}X_{t+1} + W_{hh}h_t)W_{hh}$$

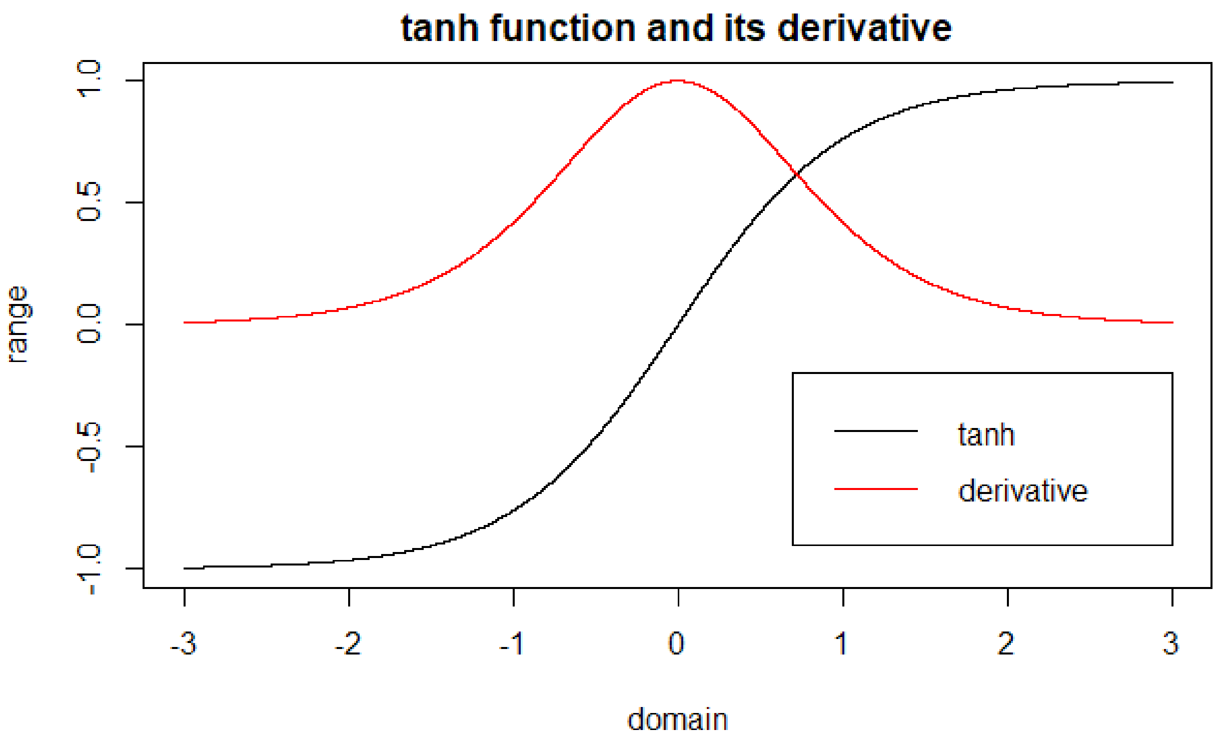
$$\rightarrow \frac{\partial h_{t+1}}{\partial h_{t-1}} = \frac{\partial h_{t+1}}{\partial h_t}\frac{\partial h_t}{\partial h_{t-1}}$$

$$= W_{hh}^2 \prod_{i=t}^{t+1} tanh'(W_{xh}X_i + W_{hh}h_{i-1})$$

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

RNN : Vanishing Gradient

$$\therefore \ \frac{\partial h_T}{\partial h_t} = W_{hh}^{T-t} \prod_{i=t+1}^{T} tanh'(W_{xh}X_i + W_{hh}h_{i-1}) \ (T > t)$$



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

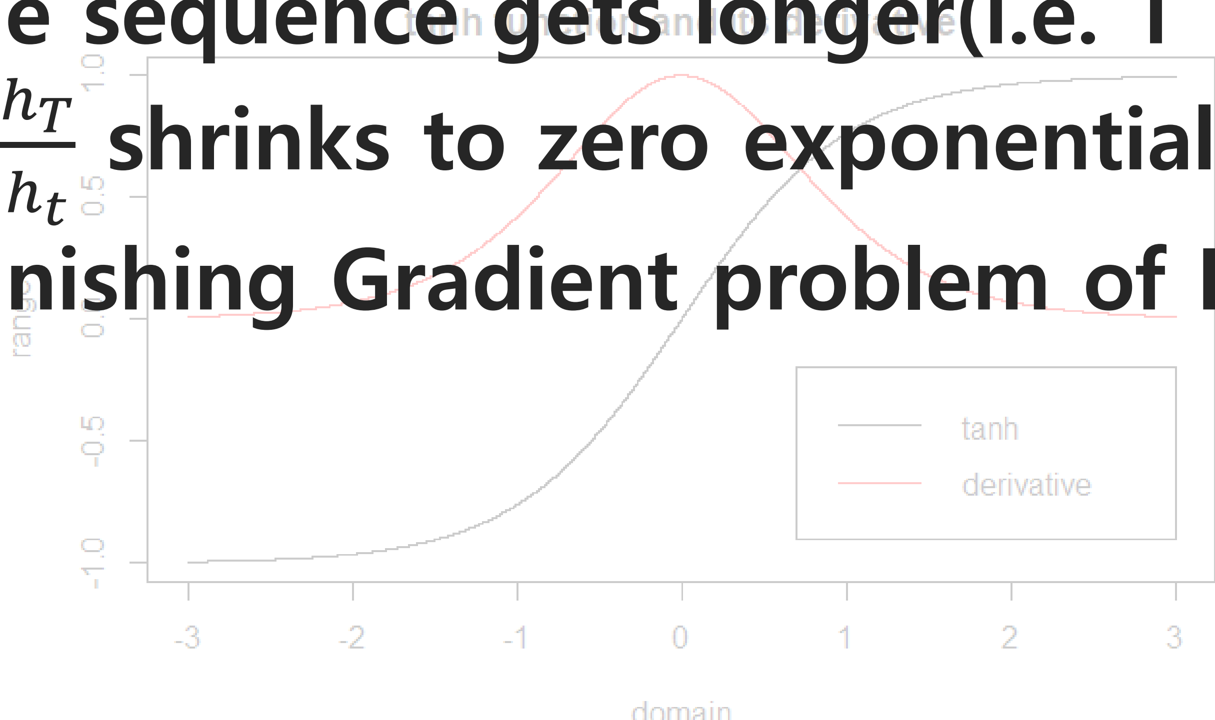
RNN : Vanishing Gradient

$$\therefore \ \frac{\partial h_T}{\partial h_t} = W_{hh}^{T-t} \prod_{i=t+1}^{T} tanh'(W_{xh}X_i + W_{hh}h_{i-1}) \ (T > t)$$

As the sequence gets longer(i.e. T → ∞),

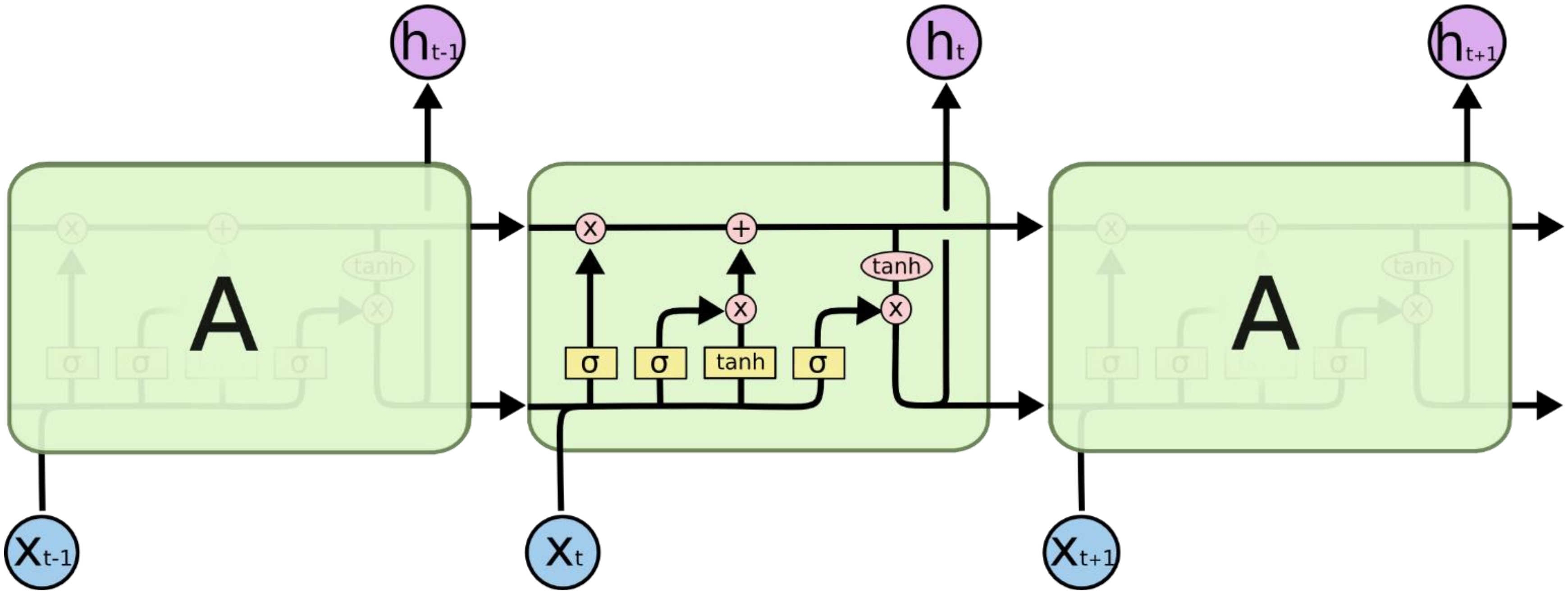
$\frac{\partial h_T}{\partial h_t}$ **shrinks to zero exponentially**

: Vanishing Gradient problem of RNN



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

LSTM : Remedy

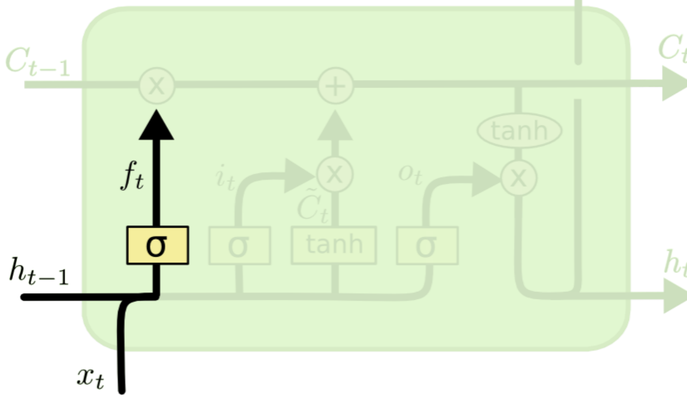


How LSTM eases the vanishing gradient problem?

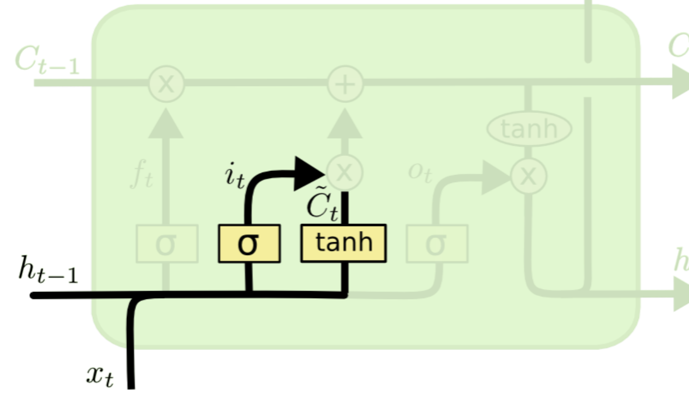
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTM : Step-by-Step

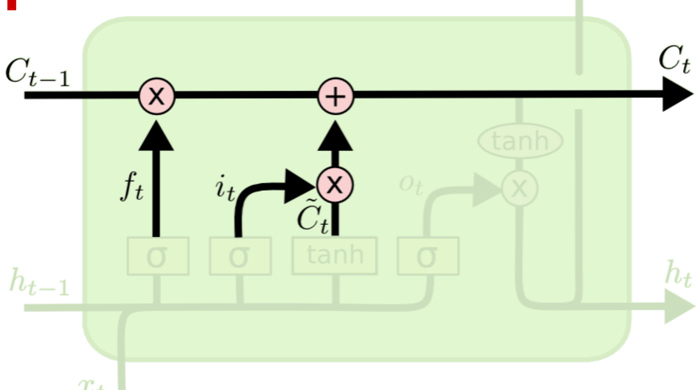
1. Pass Forget gate



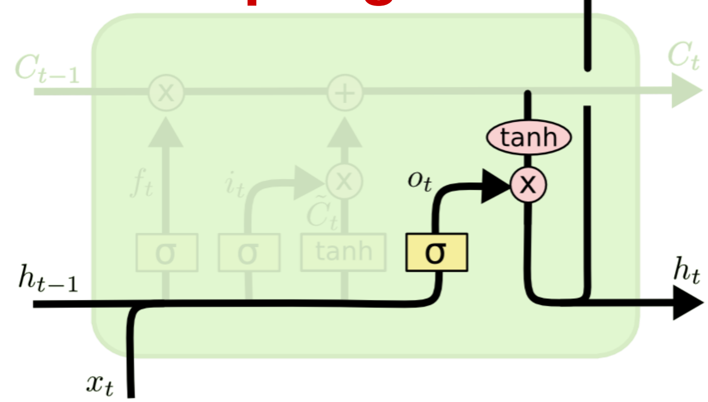
2. Pass Input gate



3. Update cell state



4. Pass Output gate

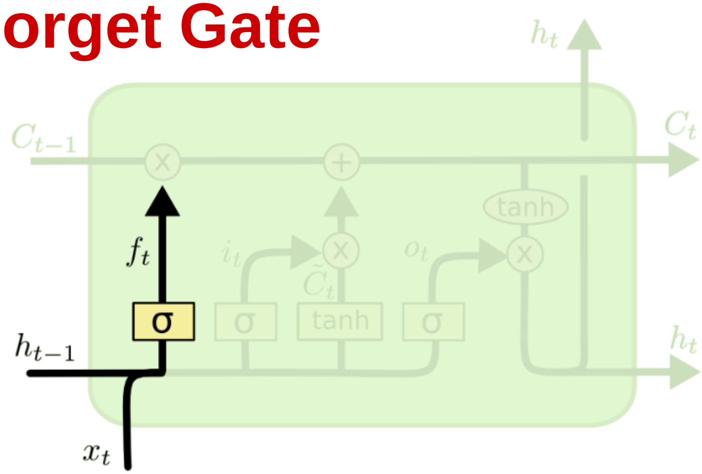


LSTM feeding is made by 4 steps

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

LSTM : Step-by-Step

1. Forget Gate

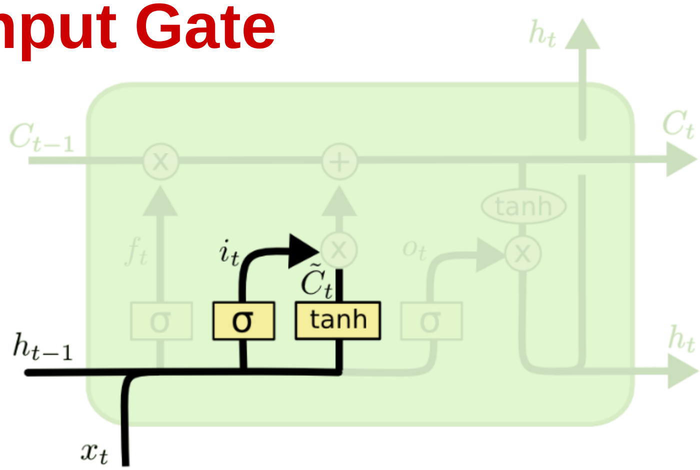


Notation for sigmoid function which ranges from 0 to 1

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

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

2. Input Gate



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

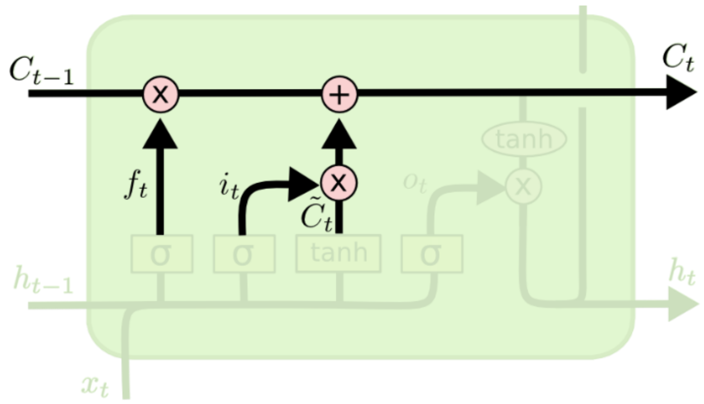
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

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

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

LSTM : Step-by-Step

3. Update (Obtain cell state)

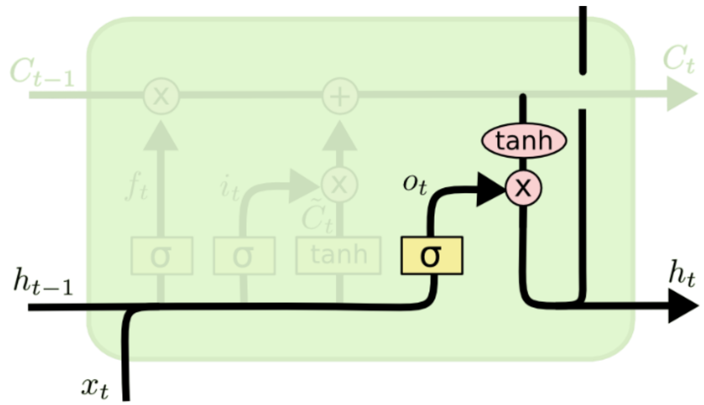


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Update cell state scaled by how much we decide to update

: input_gate*curr_state + forget_gate*prev_state

4. Output Gate (Get 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/

Flow of gradient in LSTM

for full procedure, read: https://medium.com/@aidangomez/let-s-do-this-f9b699de31d9

Observe cell state, which influences update process of hidden state:

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

$$C_{t+1} = f_{t+1} C_t + i_{t+1} \tilde{C}_{t+1}$$

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

Flow of gradient in LSTM

Observe cell state, which influences update process of hidden state:

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

Then the gradient of cell state with respect to previous cell state becomes respectively:

$$\frac{\partial C_t}{\partial C_{t-1}} = f_t \,, \qquad \frac{\partial C_{t+1}}{\partial C_t} = f_{t+1}$$

$$\therefore \; \frac{\partial C_{t+1}}{\partial C_{t-1}} = \frac{\partial C_{t+1}}{\partial C_t} \frac{\partial C_t}{\partial C_{t-1}} = f_{t+1} * f_t$$

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

Flow of gradient in LSTM

Observe cell state, which influences update process of hidden state:

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

Then the gradient of cell state with respect to previous cell state becomes respectively:

$$\frac{\partial C_{t+1}}{\partial C_{t-1}} = \frac{\partial C_{t+1}}{\partial C_t} \frac{\partial C_t}{\partial C_{t-1}} = f_{t+1} * f_t$$

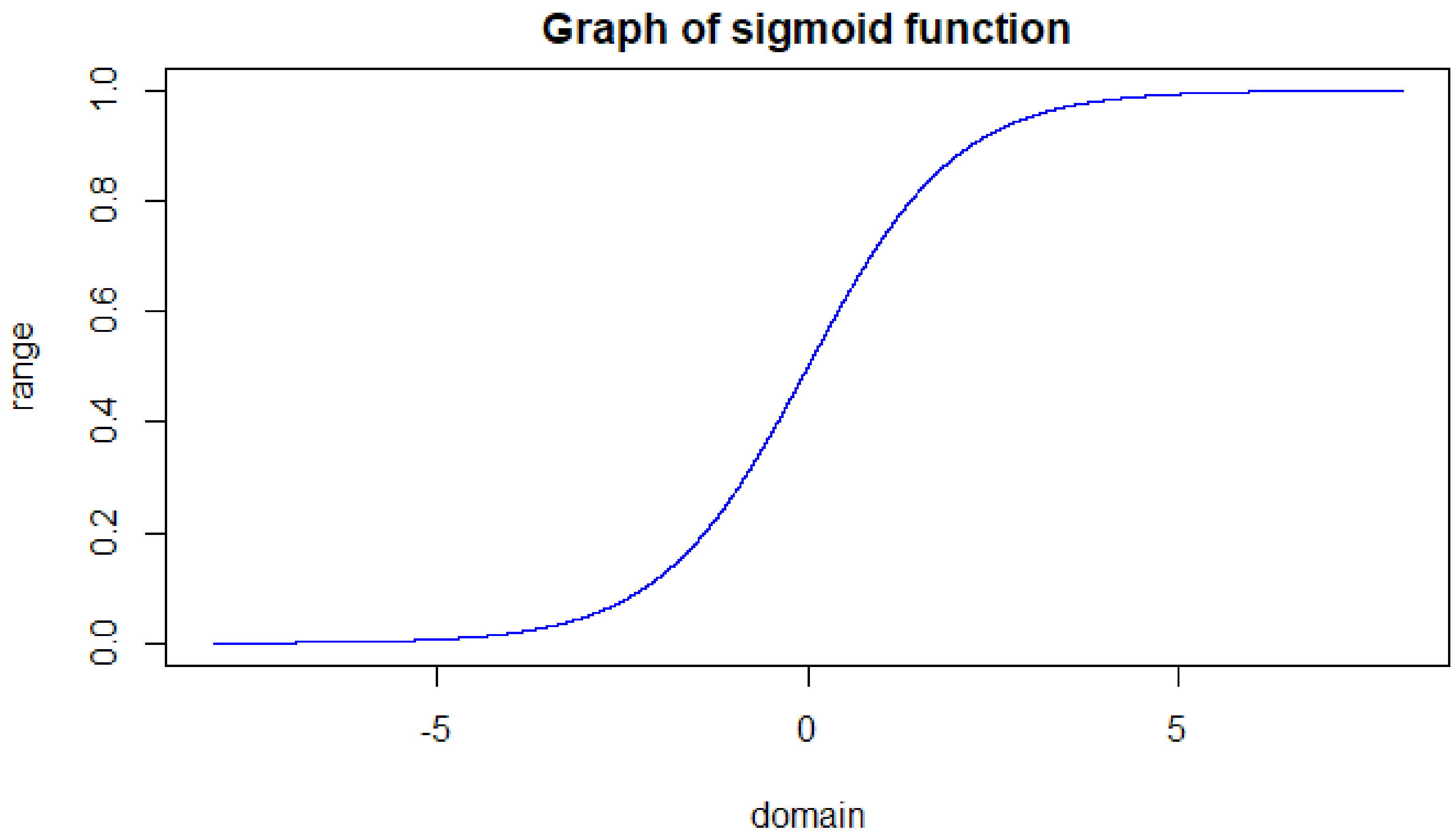
To generalize, gradient formula for cell state becomes:

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

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

Flow of gradient in LSTM

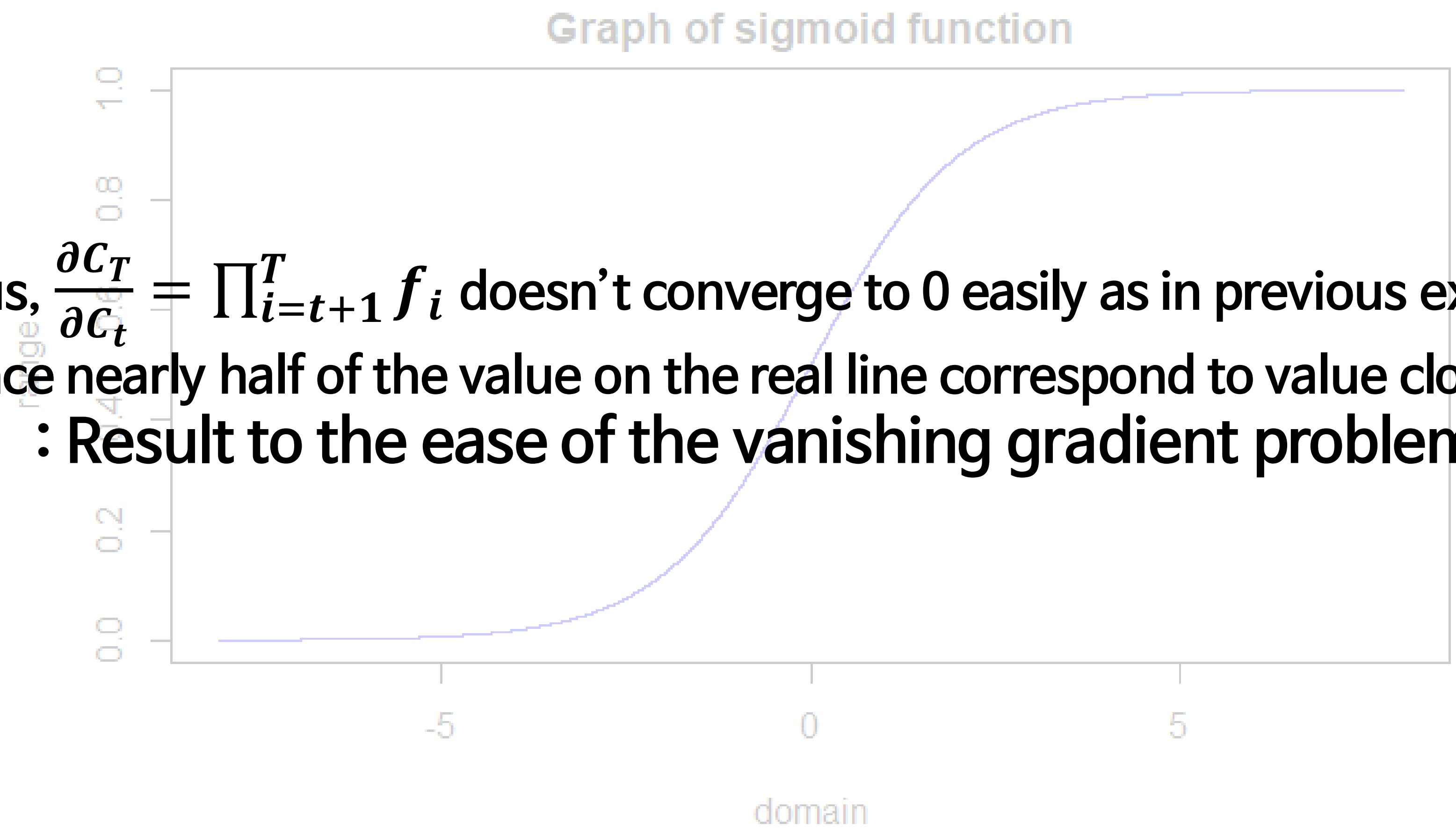
Note that f_i ranges from 0 to 1 since it is value from sigmoid function.



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

Flow of gradient in LSTM

Note that f_i ranges from 0 to 1 since it is value from sigmoid function.

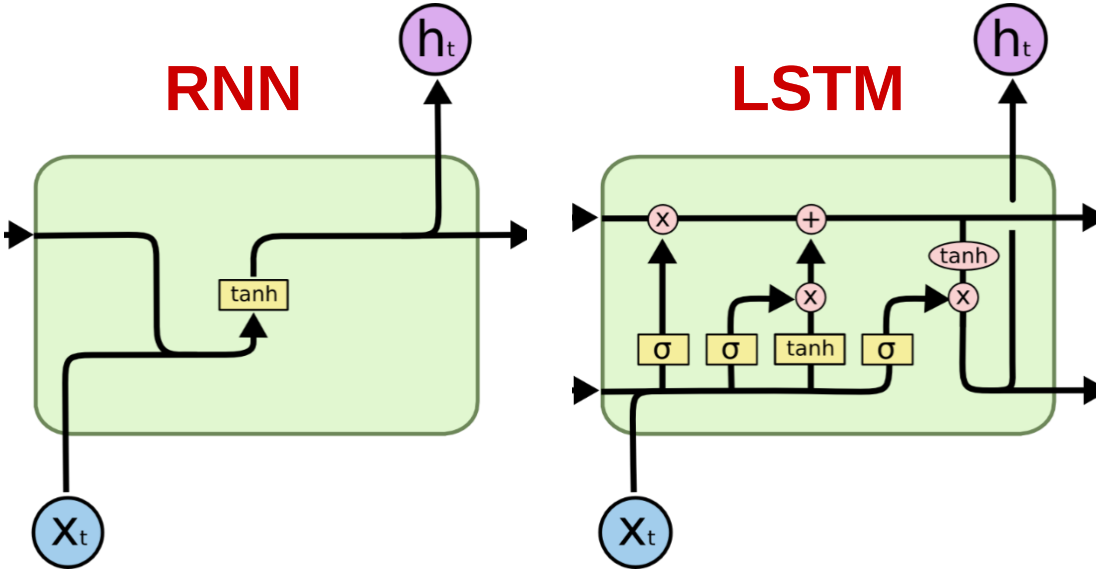


Thus, $\frac{\partial C_T}{\partial C_t} = \prod_{i=t+1}^{T} f_i$ doesn't converge to 0 easily as in previous example, since nearly half of the value on the real line correspond to value close to 1

: Result to the ease of the vanishing gradient problem!

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

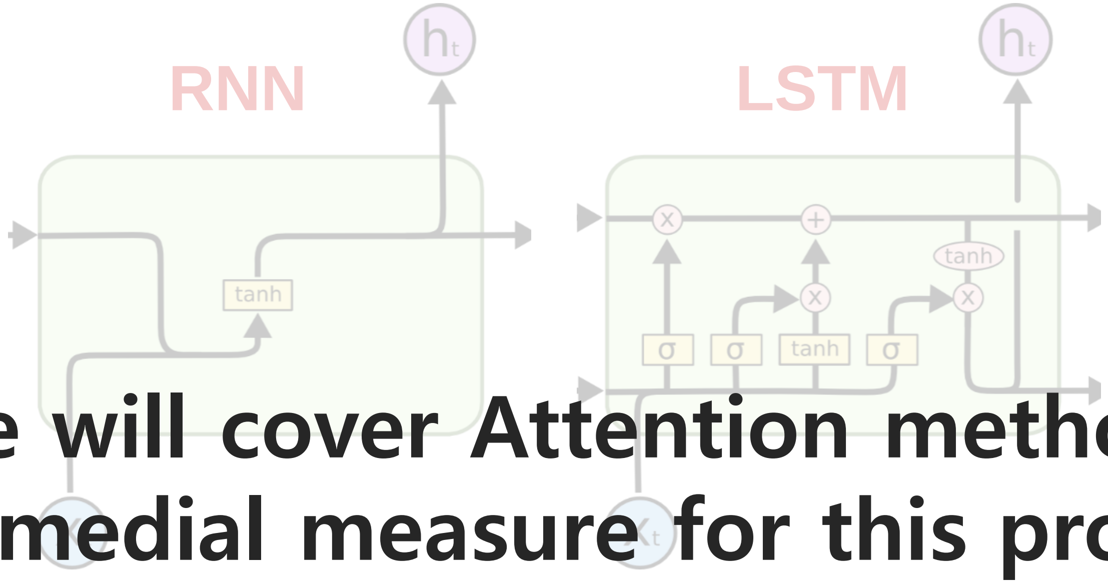
Limitation



Nevertheless, LSTM also struggles to learn sequence whose length is more than 1000s or 10,000s or more

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

Limitation



We will cover Attention method as a remedial measure for this problem!

Nevertheless, LSTM also struggles to learn sequence whose length is more than 1000s or 10,000s or more

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

Real world example
: Answering the nationality of a name
from https://pytorch.org/tutorials/intermediate/char_rnn_classification_tutorial.html

Input data

```
18 countries

Arabic ['Khoury', 'Nahas', 'Daher', 'Gerges', 'Nazari'] --> 2000 names
Chinese ['Ang', 'AuYong', 'Bai', 'Ban', 'Bao'] --> 268 names
Czech ['Abl', 'Adsit', 'Ajdrna', 'Alt', 'Antonowitsch'] --> 519 names
Dutch ['Aalsburg', 'Aalst', 'Aarle', 'Achteren', 'Achthoven'] --> 297 names
English ['Abbas', 'Abbey', 'Abbott', 'Abdi', 'Abel'] --> 3668 names
French ['Abel', 'Abraham', 'Adam', 'Albert', 'Allard'] --> 277 names
German ['Abbing', 'Abel', 'Abeln', 'Abt', 'Achilles'] --> 724 names
Greek ['Adamidis', 'Adamou', 'Agelakos', 'Akrivopoulos', 'Alexandropoulos'] --> 203 names
Irish ['Adam', 'Ahearn', 'Aodh', 'Aodha', 'Aonghuis'] --> 232 names
Italian ['Abandonato', 'Abatangelo', 'Abatantuono', 'Abate', 'Abategiovanni'] --> 709 names
Japanese ['Abe', 'Abukara', 'Adachi', 'Aida', 'Aihara'] --> 991 names
Korean ['Ahn', 'Baik', 'Bang', 'Byon', 'Cha'] --> 94 names
Polish ['Adamczak', 'Adamczyk', 'Andrysiak', 'Auttenberg', 'Bartosz'] --> 139 names
Portuguese ['Abreu', 'Albuquerque', 'Almeida', 'Alves', 'Araujo'] --> 74 names
Russian ['Ababko', 'Abaev', 'Abagyan', 'Abaidulin', 'Abaidullin'] --> 9408 names
Scottish ['Smith', 'Brown', 'Wilson', 'Campbell', 'Stewart'] --> 100 names
Spanish ['Abana', 'Abano', 'Abarca', 'Abaroa', 'Abascal'] --> 298 names
Vietnamese ['Nguyen', 'Tron', 'Le', 'Pham', 'Huynh'] --> 73 names
```

: 20074 names of 18 countries

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

Real world example

: Answering the nationality of a name

```
print(letterToTensor('J'))
print(lineToTensor('Jones'))
print(lineToTensor('Jones').size())
```

```
tensor([[0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
         0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1.,
         0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
         0., 0., 0.]])
tensor([[[0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
          0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
          0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
          0., 0., 0., 0., 0., 0.]],

        [[0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0.,
          0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
          0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
          0., 0., 0., 0., 0., 0.]],

        [[0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0.,
          0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
          0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
          0., 0., 0., 0., 0., 0.]],

        [[0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
          0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
          0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
          0., 0., 0., 0., 0., 0.]],

        [[0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
          0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
          0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
          0., 0., 0., 0., 0., 0.]]])
torch.Size([5, 1, 57])
```

J
o
n
e
s

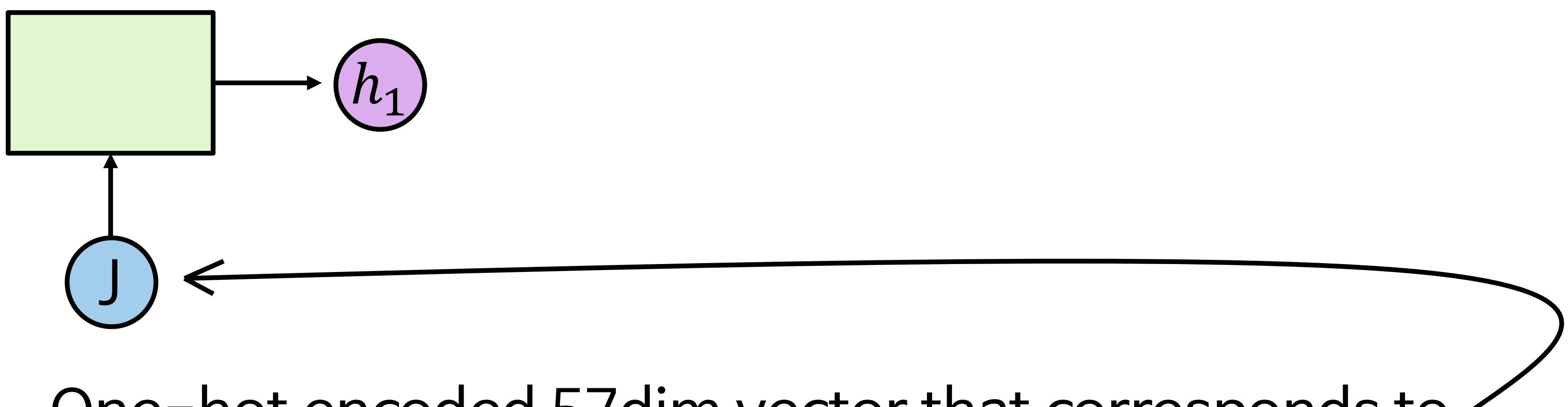
Character embedding
: One-Hot Encoding

(i.e. Each character is represented as a 57 dimensional array)

Agg. number of
lower case letters (26),
upper case letters (26)
and additional letters to
prevent Unicode encoding
error (5)

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

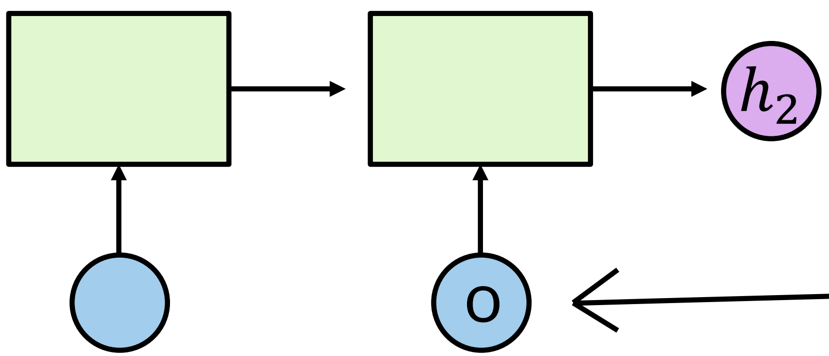
Real world example
: Answering the nationality of a name



One-hot encoded 57dim vector that corresponds to

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

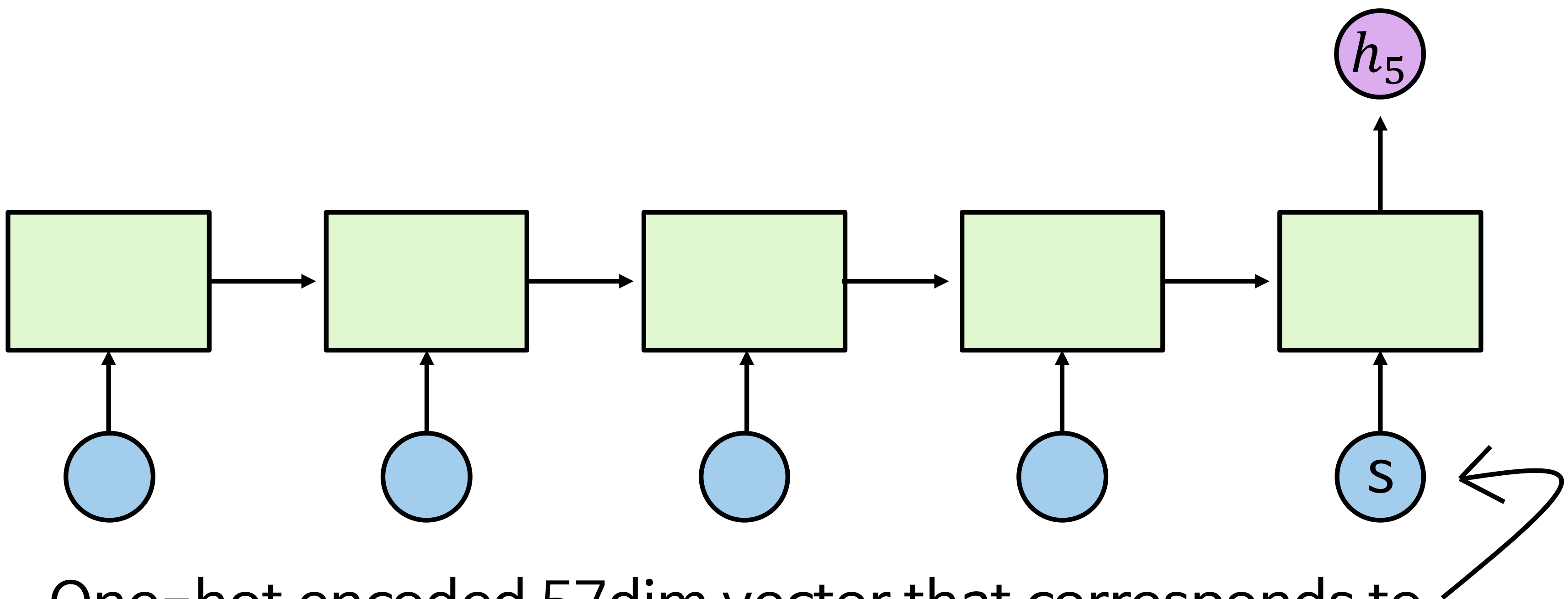
Real world example
: Answering the nationality of a name



One-hot encoded 57dim vector that corresponds to

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

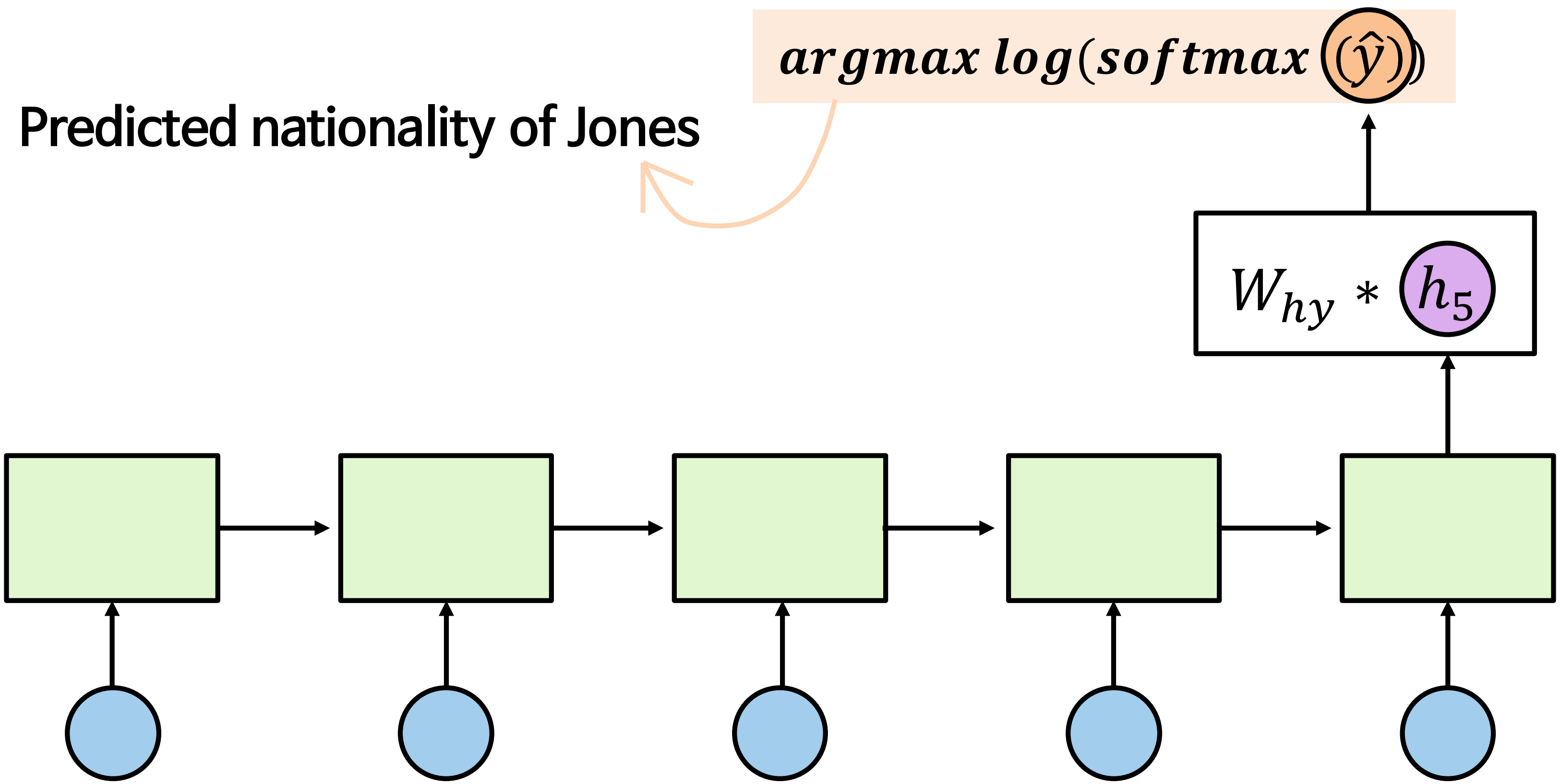
Real world example
: Answering the nationality of a name



One-hot encoded 57dim vector that corresponds to

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

Real world example
: Answering the nationality of a name



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

Real world example
: Answering the nationality of a name

Sample RNN output

```
input = lineToTensor('Albert')
hidden = torch.zeros(1, n_hidden)

output, next_hidden = rnn(input[0], hidden)
print(output)
```

```
tensor([[-2.8747, -2.8332, -2.8298, -2.8692, -2.9504, -2.8607, -2.9429, -2.8648,
         -2.9360, -3.0028, -2.8968, -2.8490, -2.9372, -2.8438, -2.8469, -2.9101,
         -2.8827, -2.9153]], grad_fn=<LogSoftmaxBackward>)
```


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

Real world example

: Answering the nationality of a name

```
predict('Dovesky')
predict('Jackson')
predict('Satoshi')
```

```
> Dovesky
(-0.69) Czech
(-0.86) Russian
(-3.68) English

> Jackson
(-0.91) English
(-1.15) Scottish
(-2.01) Russian

> Satoshi
(-1.24) Japanese
(-1.55) Italian
(-1.84) Arabic
```

Prediction with our data

: Predict with our sample names

Works Well!

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

See you next week!