2019
怪兽
学堂

RNN

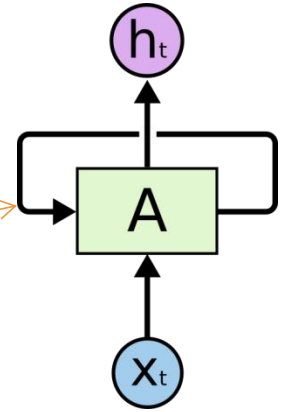
虾米　　2019-5

Introduction

- RNN (Recurrent neural network) is a form of neural networks that feed outputs back to the inputs during operation
- LSTM (Long short-term memory) is a form of RNN. It fixes the vanishing gradient problem of the original RNN.
  - Application: Sequence to sequence model based using LSTM for machine translation
- Materials are mainly based on links found in https://www.tensorflow.org/tutorials

What is RNN (Recurrent neural network) ?

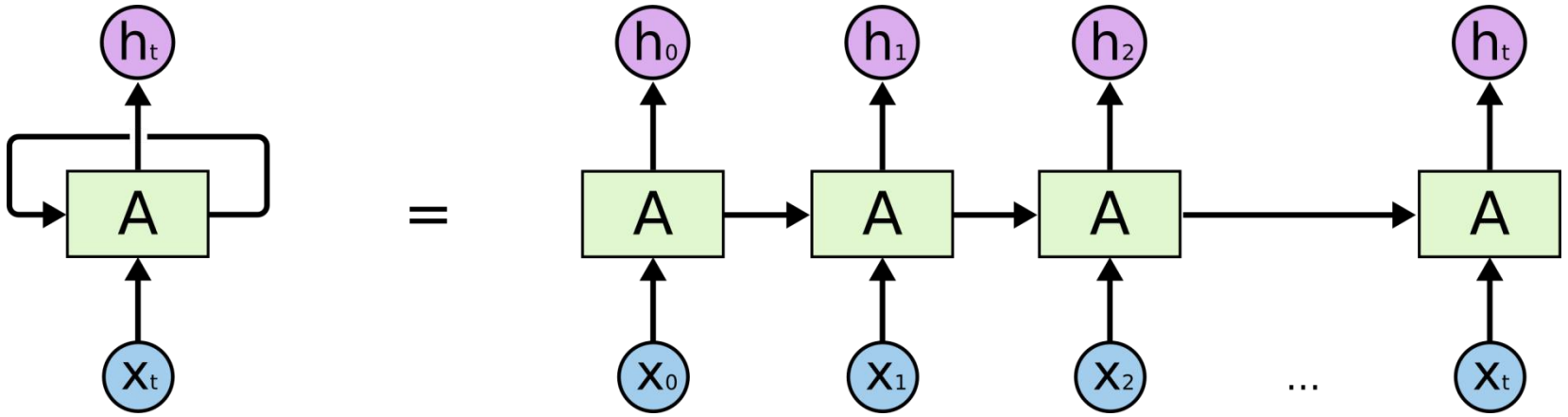
- X_t= input at time t
- h_t= output at time t
- A=neural network
- The loop allows information to pass from *t* to *t+1*
- reference: http://colah.github.io/posts/2015-08-Understanding-LSTMs/



RNN unrolled

But RNN suffers from the vanishing gradient problem

- Unroll and treat each time sample as an unit.



An unrolled RNN

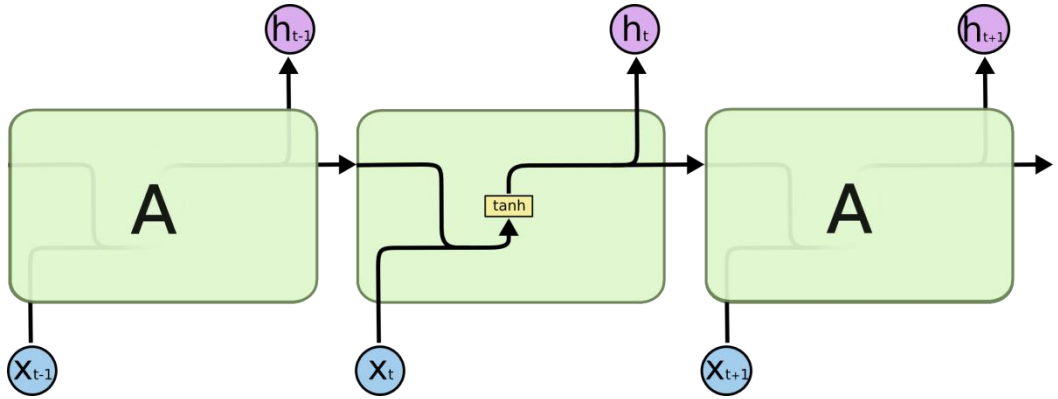
Problem:

Learning long-term dependencies with gradient descent is difficult , Bengio, et al. (1994)

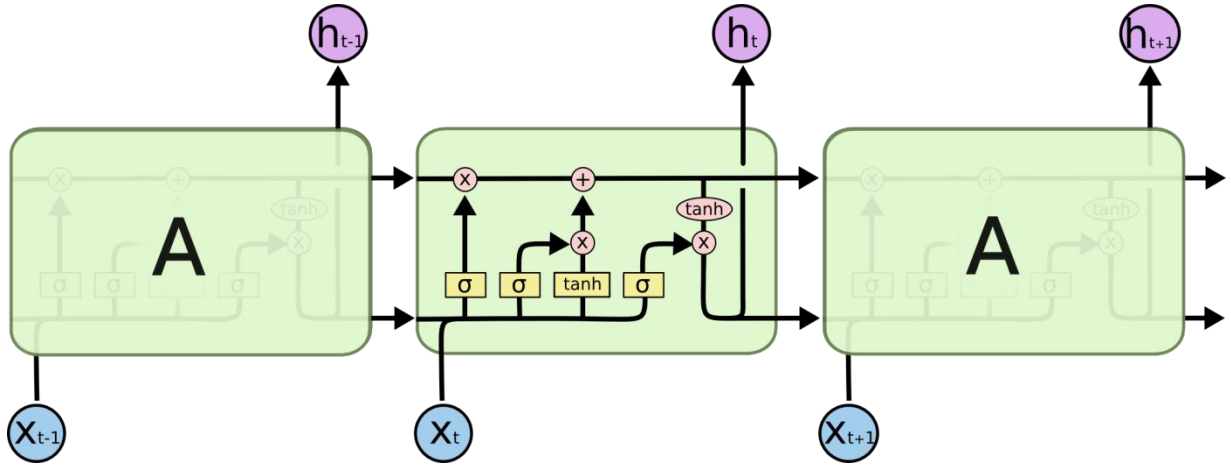
LSTM can fix the vanishing gradient problem

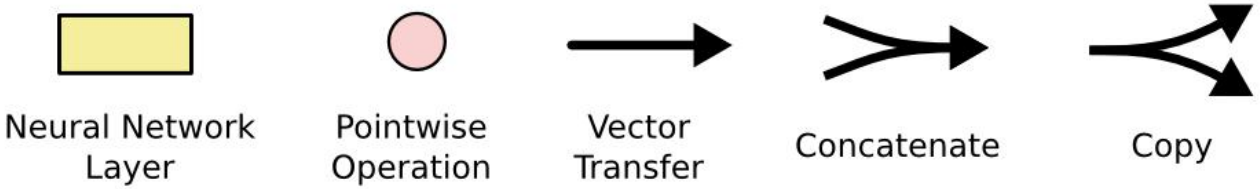
LSTM (Long short-term memory)

- Standard RNN
  - Input concatenate with output then feed to input again



- LSTM
  - The repeating structure is more complicated





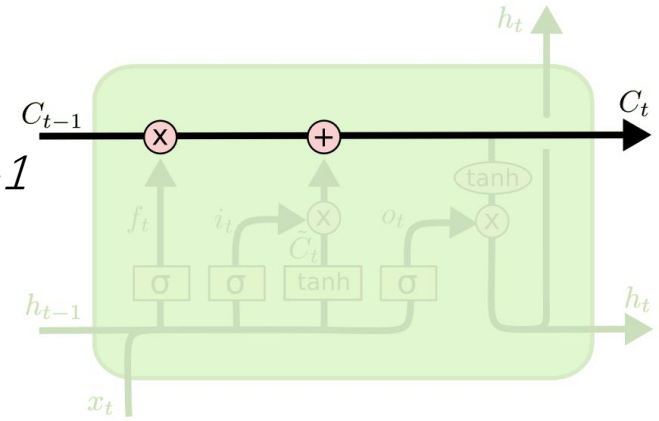
Why add C (cell state) ?

- RNN only produces output h at every time update, so the gradient vanishing problem may exist
- In LSTM, *C* and *h* are produced during each time update
  - *C*=cell state (ranges from -1 to 1)
  - *h*= output (ranges from -1 to 1)
  - The system learns C and h together

Core idea of LSTM

- C= State

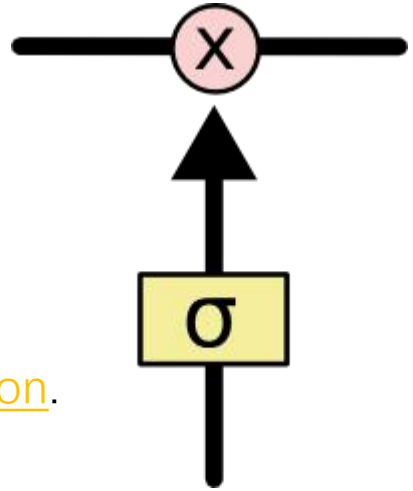
C_{t-1} =State of time t-1



C_t =State of time t

- Using gates it can add or remove information to avoid the long term dependencies problem Bengio, et al. (1994)

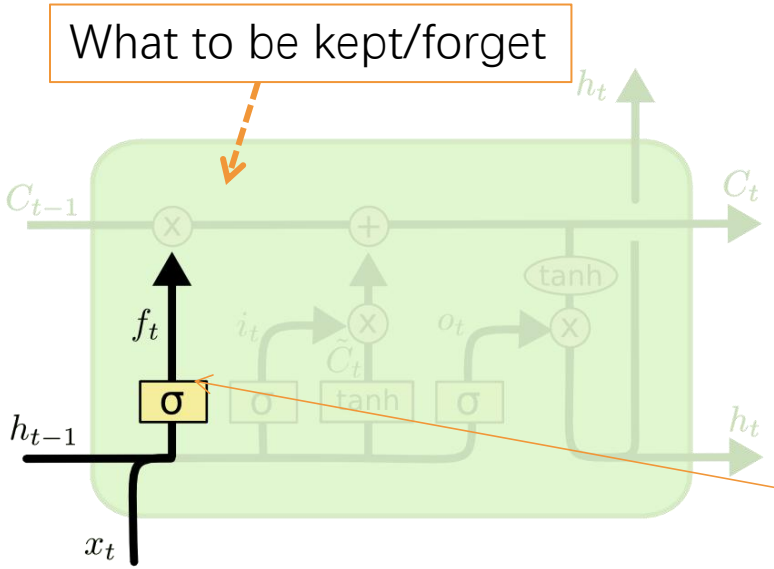
σ=a sigmoid function.



A gate controlled by σ :
The sigmoid layer outputs numbers between zero and one, describing how much of each component should be let through. A value of zero means "let nothing through," while a value of one means "let everything through!"
*An LSTM has **three** of these gates, to protect and control the cell state*
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

First step: forget gate layer

- Decide what to throw away from the cell state
- Depends on current time x (x_t) and previous h (h_{t-1}), if they match keep C ($C_{t-1} \rightarrow C_t$), otherwise, throw away C_{t-1}



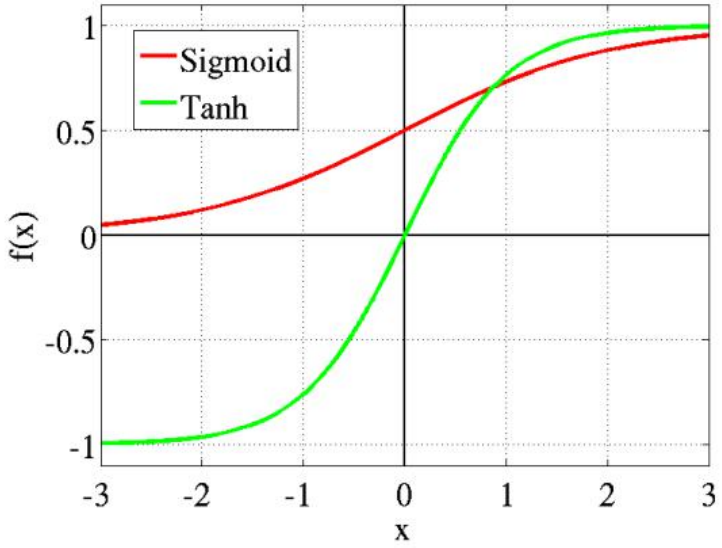
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

"It looks at h_{t-1} and x_t, and outputs a number between 0 and 1 for each number in the cell state C_{t-1}. A 1 represents "completely keep this" while a 0 represents "completely get rid of this." "

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

Sigmoid and Tanh

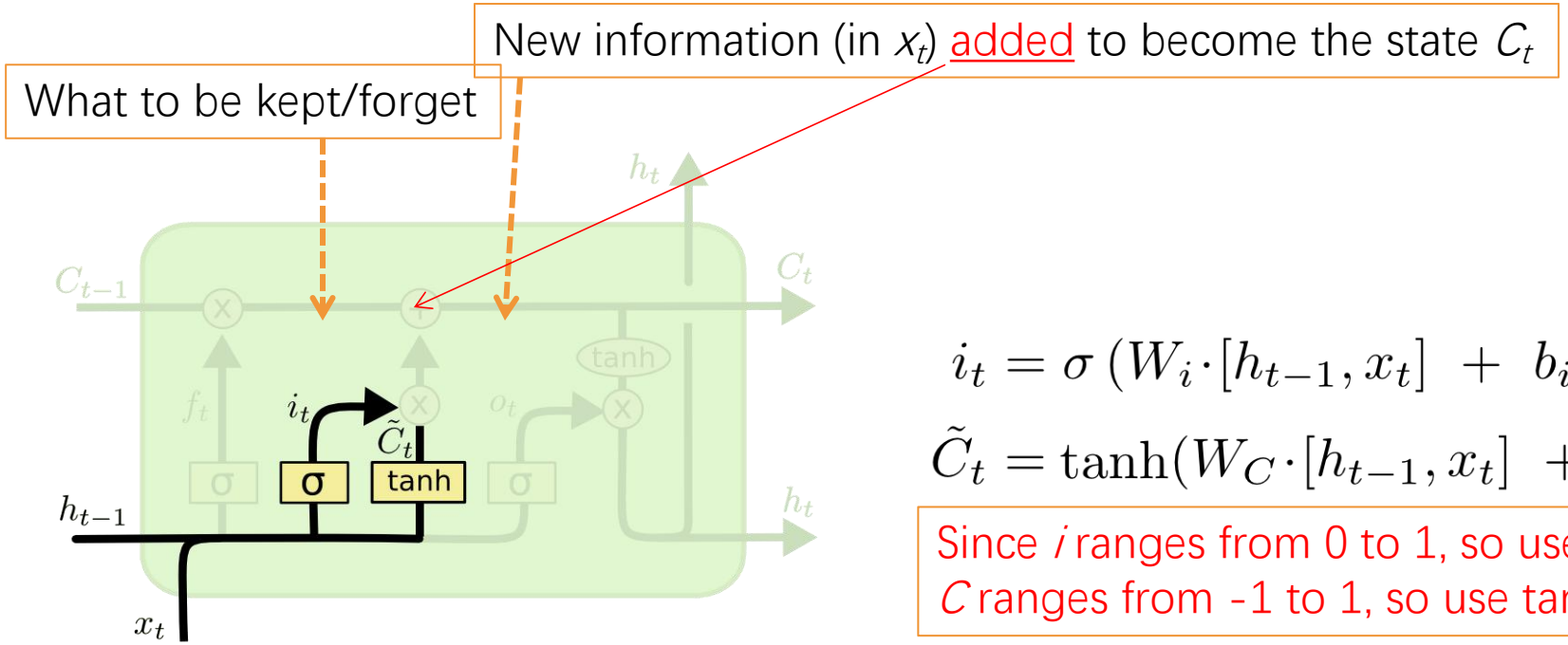
- The ranges of input/output are different
  - Sigmoid output ranges from 0 to 1
  - Tanh output ranges from -1 to 1



https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

Second step (a): input gate layer

- Decide what information to store in the cell state
- if x (x_t) and previous h (h_{t-1}) match, x_t and h_{t-1} work out some output to be stored in C_t .



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

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

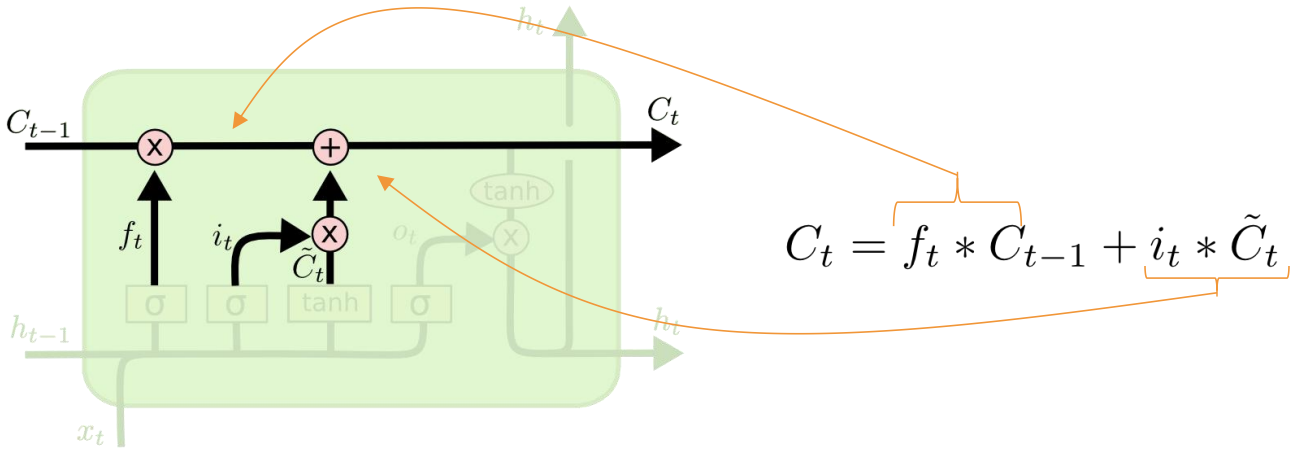
Since i ranges from 0 to 1, so use sigmoid;
C ranges from -1 to 1, so use tanh

"Next, a tanh layer creates a vector of new candidate values, ~C_t, that could be added to the state. In the next step, we'll combine these two to create an update to the state."

Second step (b): update the old cell state

- $C_{t-1} \rightarrow C_t$

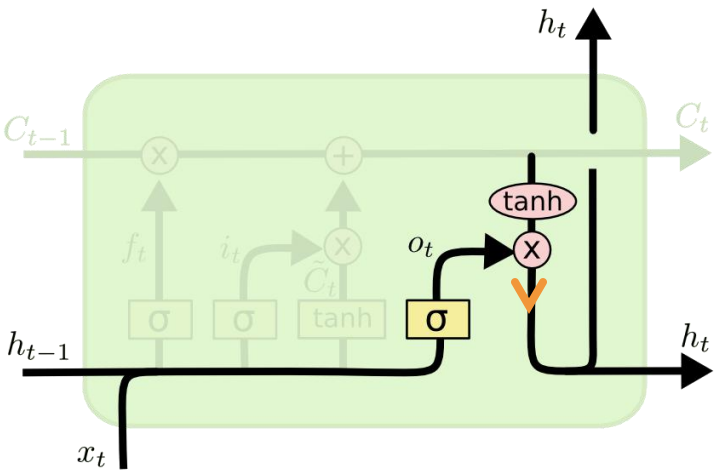
*"We multiply the old state by f_t, forgetting the things we decided to forget earlier. Then we add i_t * ~C_t. This is the new candidate values, scaled by how much we decided to update each state value."*
http://colah.github.io/posts/2015-08-Understanding-LSTMs/



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

Third step: output layer

- Decide what to output (h_t).



"Finally, we need to decide what we're going to output. This output will be based on our cell state, but will be a filtered version. First, we run a sigmoid layer which decides what parts of the cell state we're going to output. Then, we put the cell state through tanh (to push the values to be between −1 and 1) and multiply it by the output of the sigmoid gate, so that we only output the parts we decided to."

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

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

h ranges from -1 to 1, so use tanh

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

$$f_{t(m\times1)} = \sigma(W_{xf(m\times n)}x_{t(n\times1)} + W_{hf(m\times m)}h_{t-1(m\times1)} + b_{f(m\times1)})$$

$$c_{t(m\times1)} = i_{t(m\times1)} \circ .* u_{t(m\times1)} + f_{t(m\times1)} .* c_{t-1(m\times1)}$$

.* = pointwise multiplication

$$o_{t(m\times1)} = \sigma(W_{xo(m\times n)}x_{t(n\times1)} + W_{ho(m\times m)}h_{t-1(m\times1)} + b_{o(m\times1)})$$

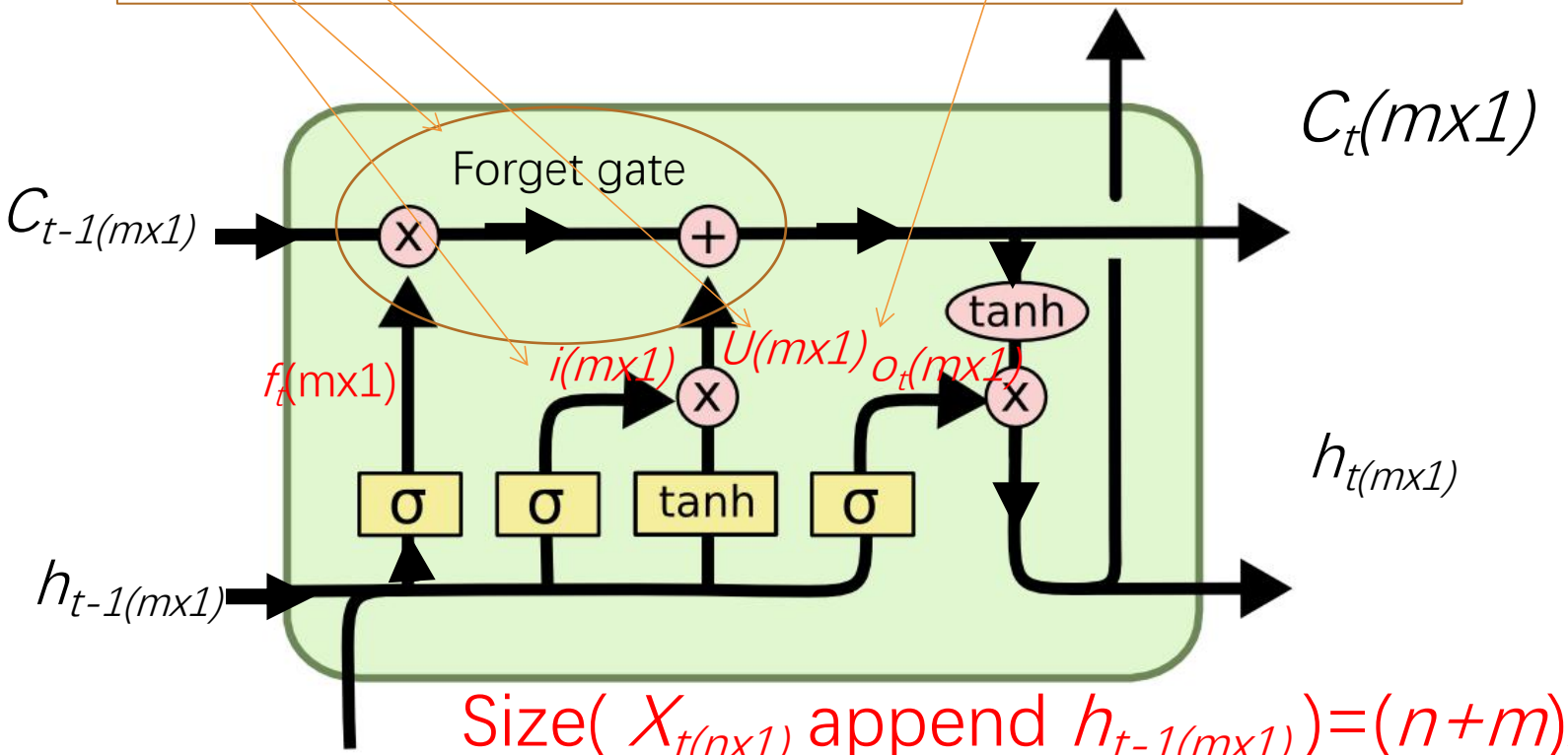
- $$u_{t(m\times1)} = \tanh(W_{xu(m\times n)}x_{t(n\times1)} + W_{hu(m\times m)}h_{t-1(m\times1)} + b_{u(m\times1)})$$

$$i_{t(m\times1)} = \sigma(W_{xi(m\times n)}x_{t(n\times1)} + W_{hi(m\times m)}h_{t-1(m\times1)} + b_{i(m\times1)})$$

X is of size nx1

h is of size mx1

http://kvitajakub.github.io/2016/04/14/rnn-diagrams/



Size($X_{t(nx1)}$ append $h_{t-1(mx1)}$)=(n+m)x1

X is of size nx1

Summary of the 7 LSTM equations

- $\sigma()$=sigmoid & *tanh()*=hyperbolic tangent are activation functions

$$i_{t(m\times 1)} = \sigma(W_{xi(m\times n)} x_{t(n\times 1)} + W_{hi(m\times m)} h_{t-1(m\times 1)} + b_{i(m\times 1)}) ----(1)$$

$$f_{t(m\times 1)} = \sigma(W_{xf(m\times n)} x_{t(n\times 1)} + W_{hf(m\times m)} h_{t-1(m\times 1)} + b_{f(m\times 1)}) ---(2)$$

$$o_{t(m\times 1)} = \sigma(W_{xo(m\times n)} x_{t(n\times 1)} + W_{ho(m\times m)} h_{t-1(m\times 1)} + b_{o(m\times 1)}) ---(3)$$

$$u_{t(m\times 1)} = \tanh(W_{xu(m\times n)} x_{t(n\times 1)} + W_{hu(m\times m)} h_{t-1(m\times 1)} + b_{u(m\times 1)}) --(4)$$

$$c_{t(m\times 1)} = i_{t(m\times 1)} .* u_{t(m\times 1)} + f_{t(m\times 1)} .* c_{t-1(m\times 1)} ----------(5)$$

$$h_t = \tanh(c_t) .* o_t ----------(6)$$

$$pred\_out = \sigma(h_t * (out\_para)) ----------(7)$$

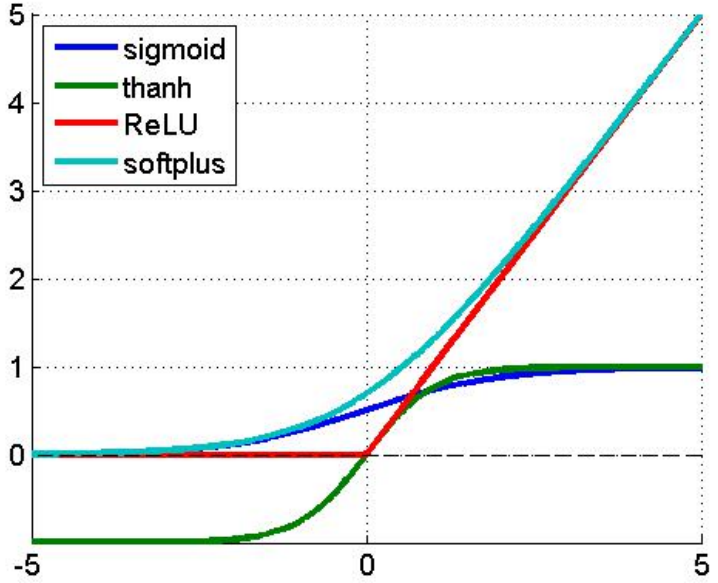
$$pred\_out = \text{predicted output},$$

$$out\_para = \text{output\_weights in softmax output}$$

$$.* = \text{pointwise multiplication}$$

Activation function choices

- **sigmoid**:
  - g(x) = 1 /(1+exp(-1)). The derivative of sigmoid function g'(x) = (1-g(x))g(x).
- **tanh** :
  - g(x) = sinh(x)/cosh(x) = (exp(x)- exp(-x)) / (exp(x) + exp(-x))
- **Rectifier:**
  - (hard ReLU) is really a max function g(x)=max(0,x)
- **Softplus:**
  - Another version is Noise ReLU max(0, x+*N*(0, σ(x)). ReLU can be approximated by a so called *softplus* function (for which the derivative is the logistic functions):
- g(x) = log(1+exp(x))



Relu is now very popular and shown to be working better other methods

https://imiloainf.wordpress.com/2013/11/06/rectifier-nonlinearities/

Run LSTM in tensorflow

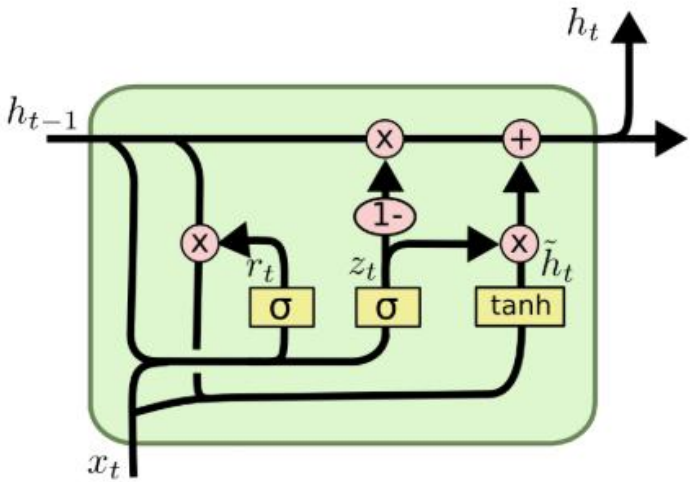
- Read https://www.tensorflow.org/tutorials/recurrent
- Download files
  - The data required for this tutorial is in the data/ directory of the PTB dataset from Tomas Mikolov's webpage. Get simple-examples.tgz, unzip into D:\tensorflow\simple-examples
  - https://github.com/tensorflow/models
  - Save in some location, e.g. D:\tensorflow\models-master\tutorials\rnn
- To run the learning program, open cdm (command window in windows)
  - cd D:\tensorflow\models-master\tutorials\rnn
  - **locate the files in these directories first
  - cd D:\tensorflow\models-master\tutorials\rnn\ptb
  - python ptb_word_lm.py --data_path=D:\tensorflow\simple-examples\data --model=small
  - Will display, ,……
    - Epoch: 1 Learning rate: 1.000
    - 0.004 perplexity: 7977.018 speed: 1398 wps
    - 0.104 perplexity: 857.681 speed: 1658 wps
    - 0.204 perplexity: 627.014 speed: 1666
- To run the Read reader test: reader_test.py

Extensions of LSTM

- Gated Recurrent Unit (GRU)
- CNN (convolution neural network)+LSTM (long short-term memory)

Modification of LSTM : GRU

- A slightly more dramatic variation on the LSTM is the Gated Recurrent Unit, or GRU, introduced by Cho, et al. (2014). It combines the forget and input gates into a single "update gate." It also merges the cell state and hidden state, and makes some other changes. The resulting model is simpler than standard LSTM models, and has been growing increasingly popular.
- http://colah.github.io/posts/2015-08-Understanding-LSTMs/



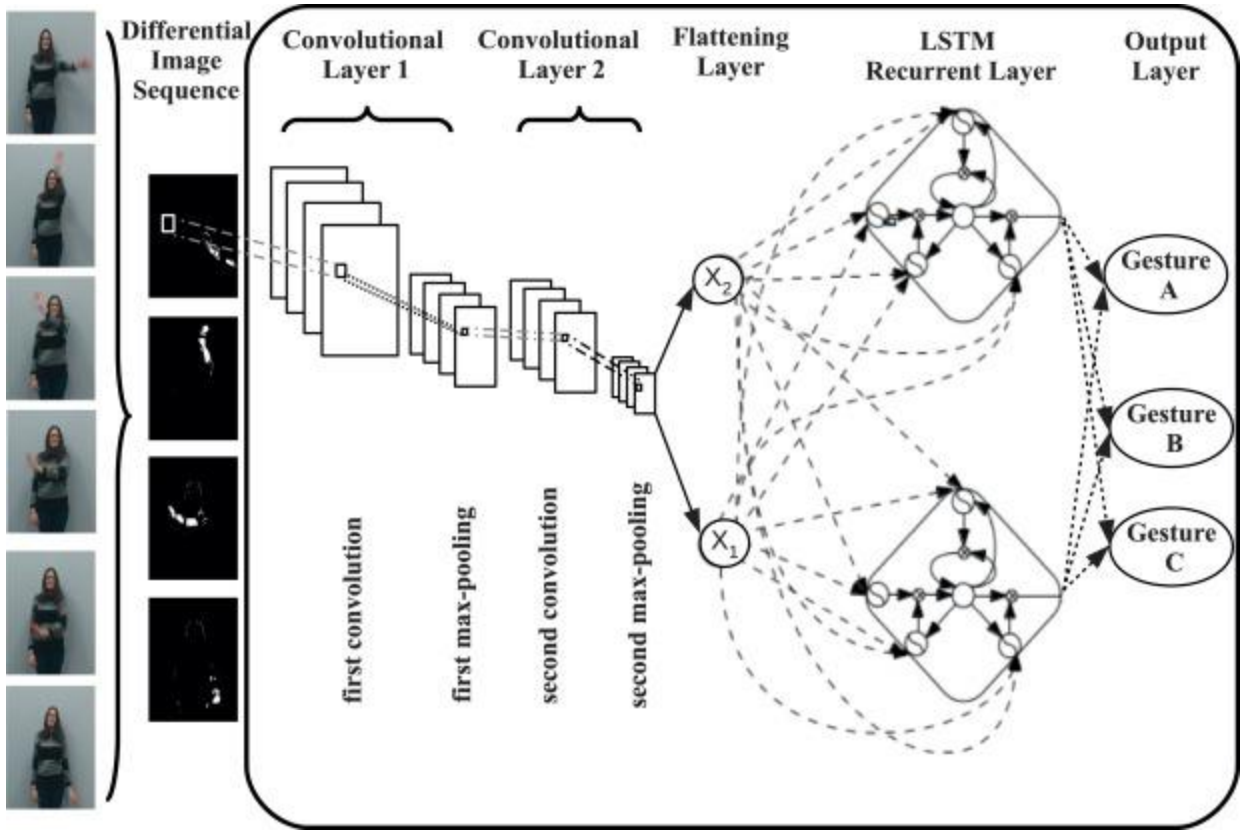
$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

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

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

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

LSTM can combine with CNN example: CNN +LSTM

- 

https://www.sciencedirect.com/science/article/pii/S0925231217307555

Summary

- Introduced the idea of Recurrent Neural Networks RNN and Long Short-Term Memory LSTM
- Gave and explained an example of implementing a digital adder of using LSTM

2019
怪兽
学堂

THANKS