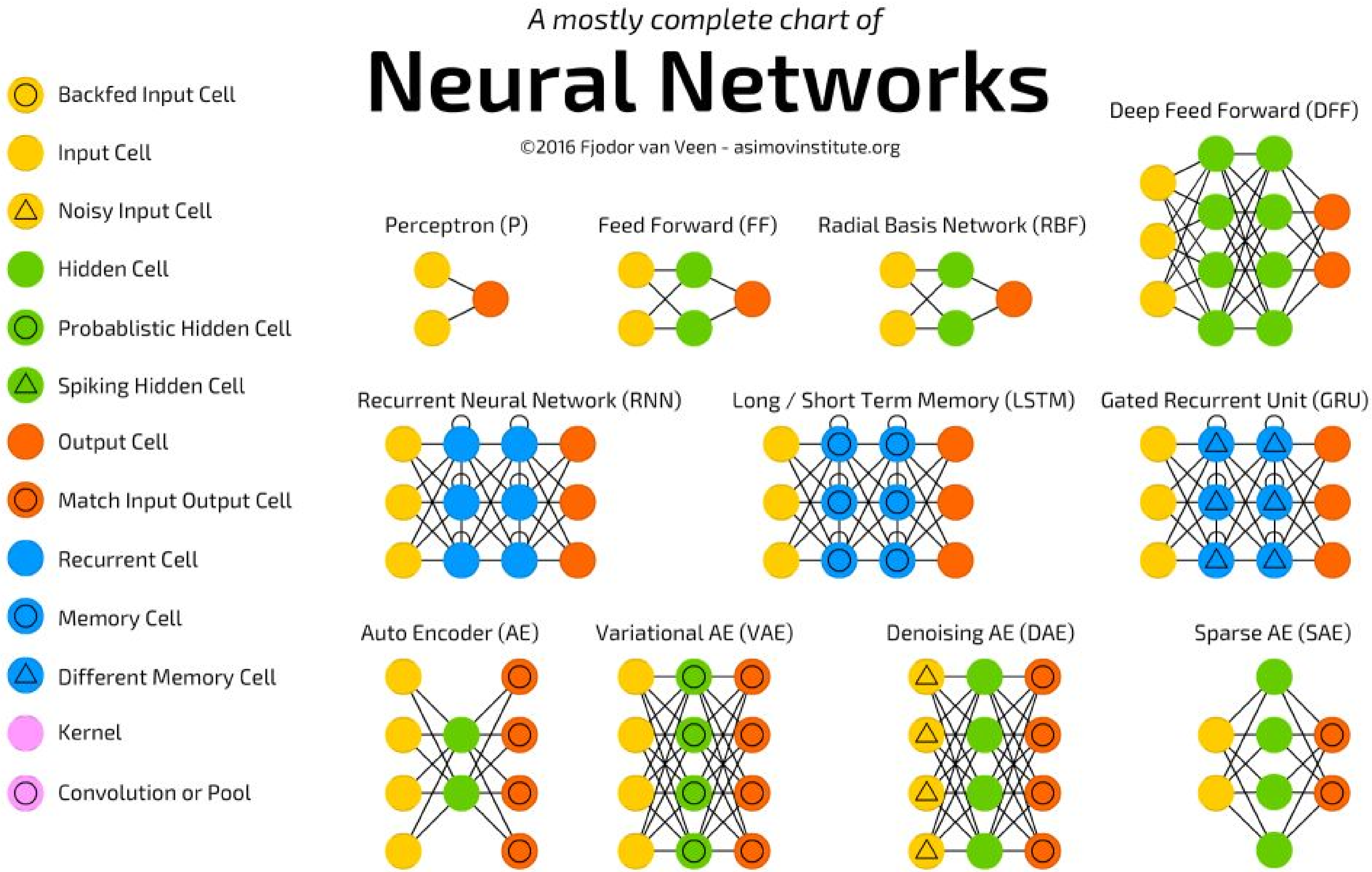
“Intuition behind LSTM”

Vikram Voleti
IIIT Hyderabad

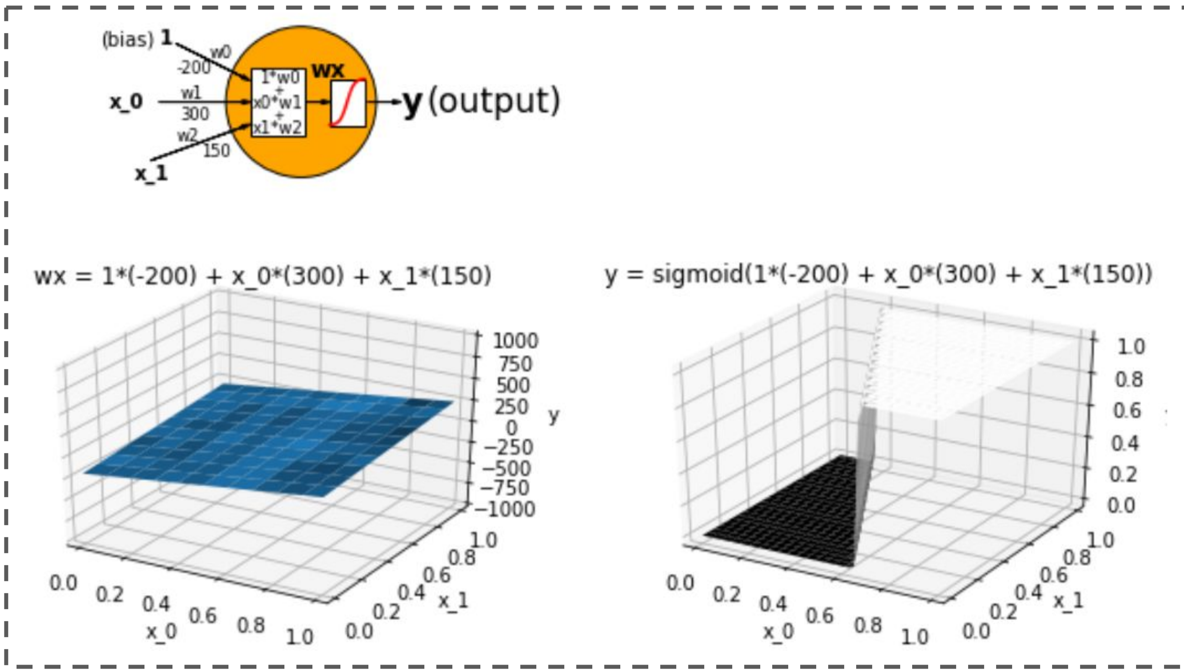
Neural Networks



Neural Networks

Perceptron: linear combination **->** non-linearity

Non-linearity “squashifies” the output - *sigmoid*: 0 to 1, *tanh*: -1 to 1, *relu*: 0 to inf



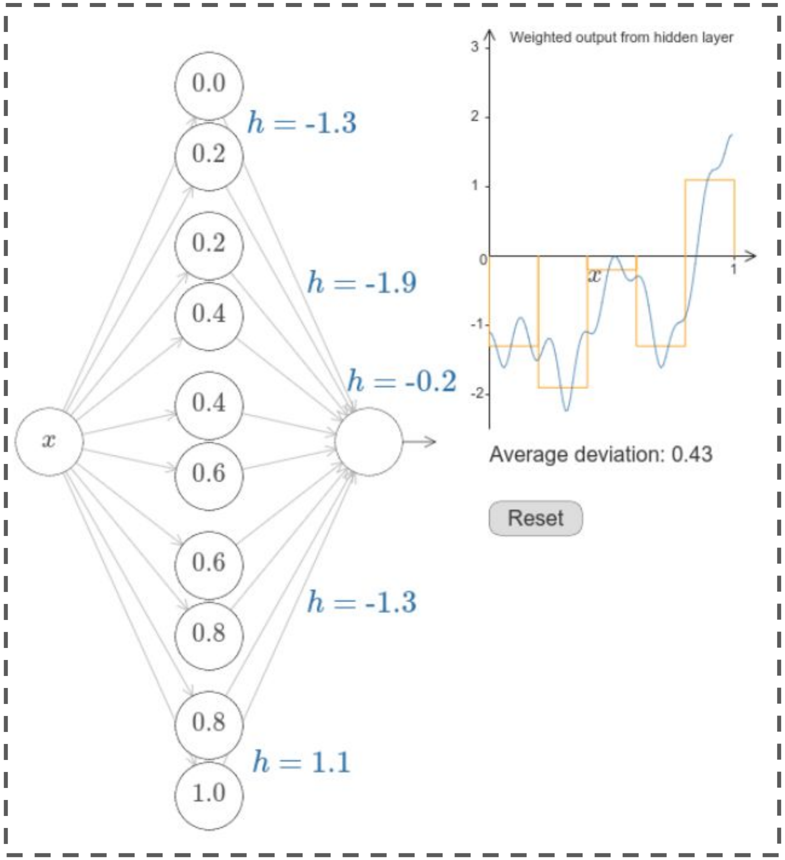
Sigmoid was chosen because: 1) “squashifies” between 0 & 1 for convenient binary classification
2) Can act as probability, to put a thresholded on
3) derivative is easy to compute

Source: https://github.com/voletiv/myPythonNeuralNetwork

Neural Networks

Why non-linearity?

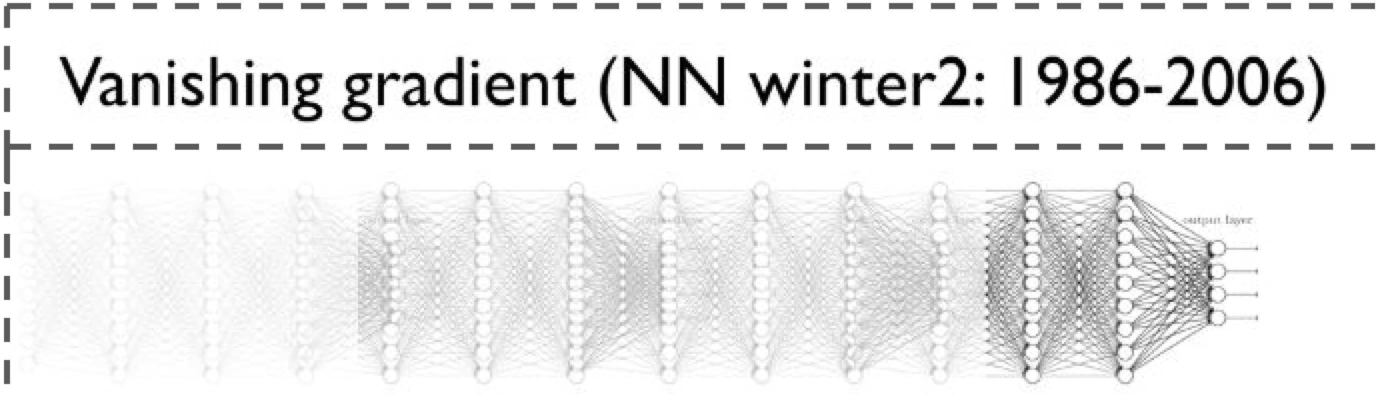
Linear combination will only give linear boundaries between classes. Non-linearities make neural networks **universal approximators**.



Source: http://neuralnetworksanddeeplearning.com/chap4.htm

Vanishing Gradient Problem

Problem 1: Training neural networks via gradient descent using backpropagation incurs vanishing/exploding gradient problem.

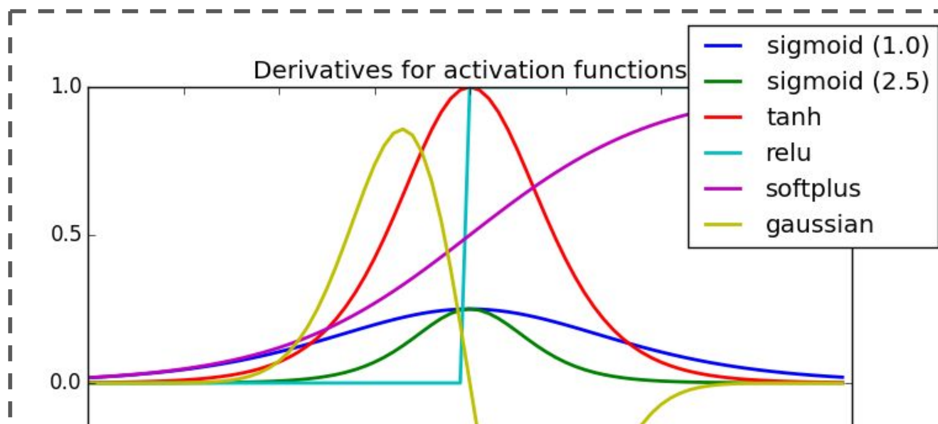


(Gradient gets worse with number of layers)

Key reason:

Fractional derivatives of non-linearities

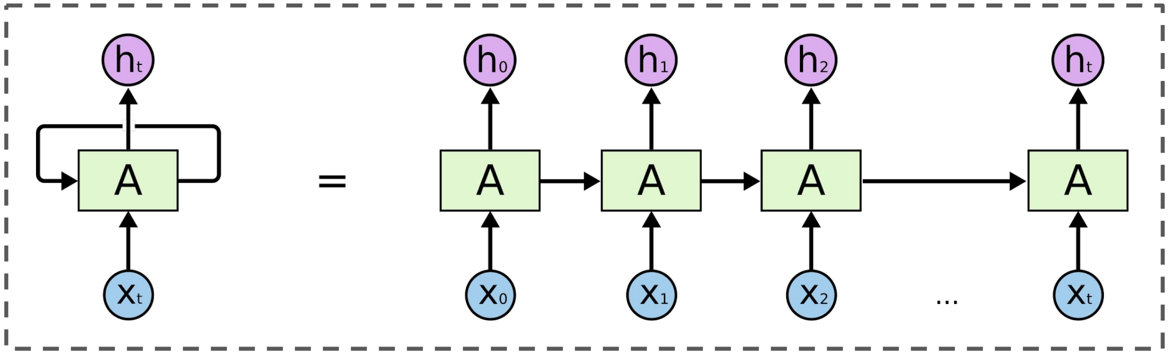
(That's why ReLU is preferred.)

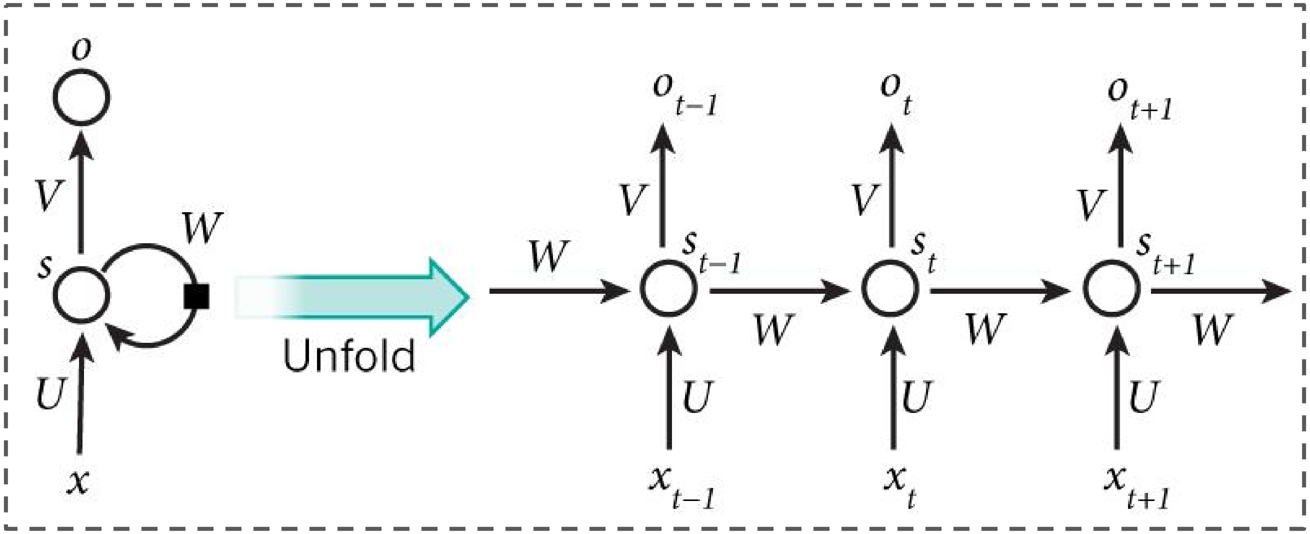


Vanilla RNN

Problem 2: Fixed input size. (Sequence Learning?)

Solution: Recurrent Neural Networks

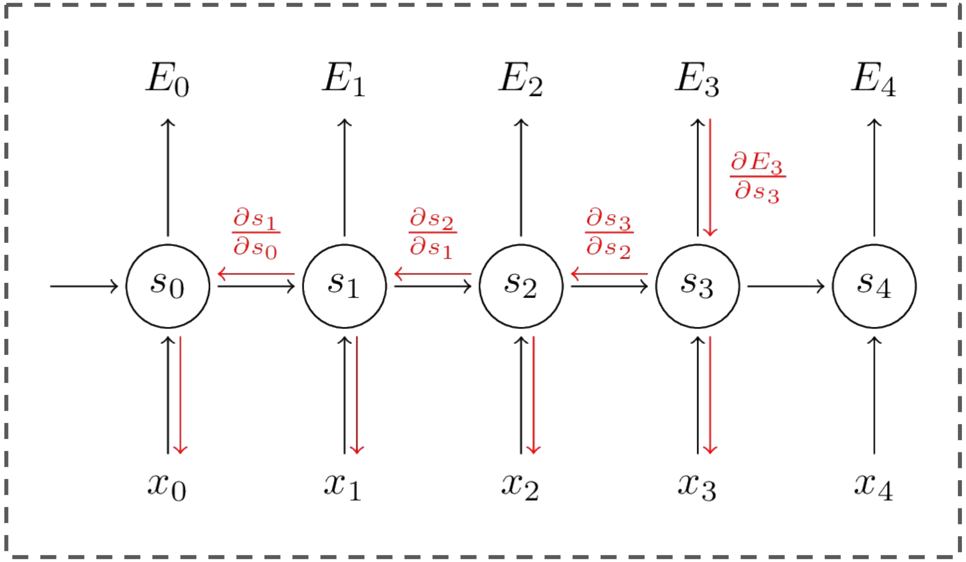




Source: colah.github.io, Nature

Vanilla RNN - Vanishing Gradient Problem

Problem 3: Training recurrent neural networks incurs vanishing/exploding gradient problem.



Back-Propagation Through Time (BPTT)

(Gradient gets worse with time)

Key reason: Haphazard updation of cell state

Hint: Related to eigenvalues of weight matrices

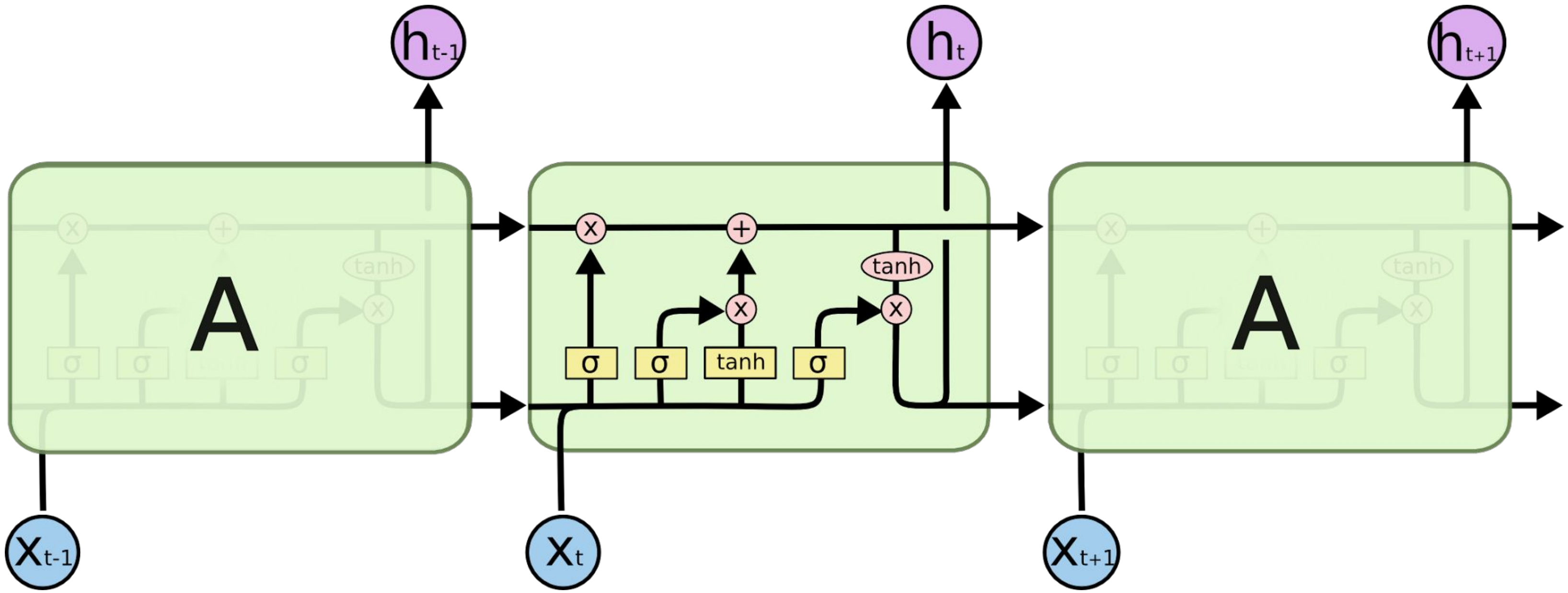
Source: http://www.wildml.com/2015/10/recurrent-neural-networks-tutorial-part-3-backpropagation-through-time-and-vanishing-gradients/

LSTM

Problem: Vanishing/Exploding gradients in RNNs

Solution: Long Short-Term Memory (Hochreiter and Schmidhuber, 1997)

[link]



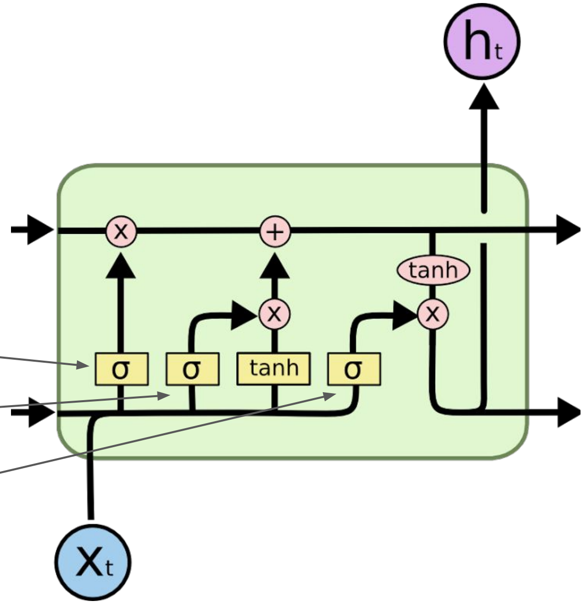
Introducing: Long-term memory (cell state), short-term memory (working memory/cell output)

Source: colah.github.io

LSTM

3 Gates: (sigmoid units in the diagram)

1. **Forget gate**
2. **Input gate**
3. **Output gate**

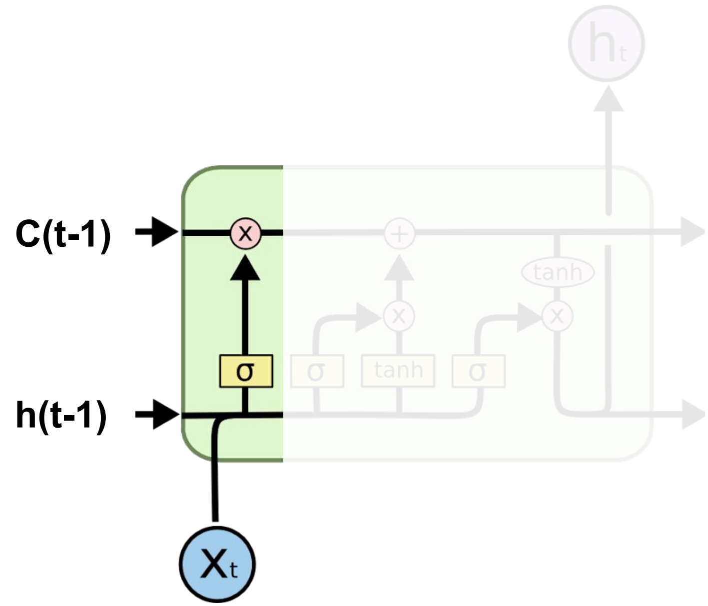


Source: colah.github.io, http://blog.echen.me/2017/05/30/exploring-lstms

LSTM - forget gate

1. Forget Gate:

- Remember only some parts of the long-term memory and forget the rest.
- Decide what to remember based on current input, and previous working memory.



Eg.: Remember that a character had died, forget the colour of their shirt.
Remember the currently called function, forget a returned value.

$$\mathit{forget\_gate}(t) = \mathit{sigmoid}(\, \boldsymbol{W_f}\, (\, \boldsymbol{x}(t),\, \boldsymbol{h}(t\text{-}1)\,)\,)$$

$$\mathit{remembered\_cell\_state}(t) = \mathit{forget\_gate}(t)\ .\!*\ \boldsymbol{C}(t\text{-}1)$$

The ***forget_gate*** has a ***sigmoid*** activation so as to act as a fraction on the previous long-term memory/cell state - hence deciding what fraction to remember and what fraction to forget.

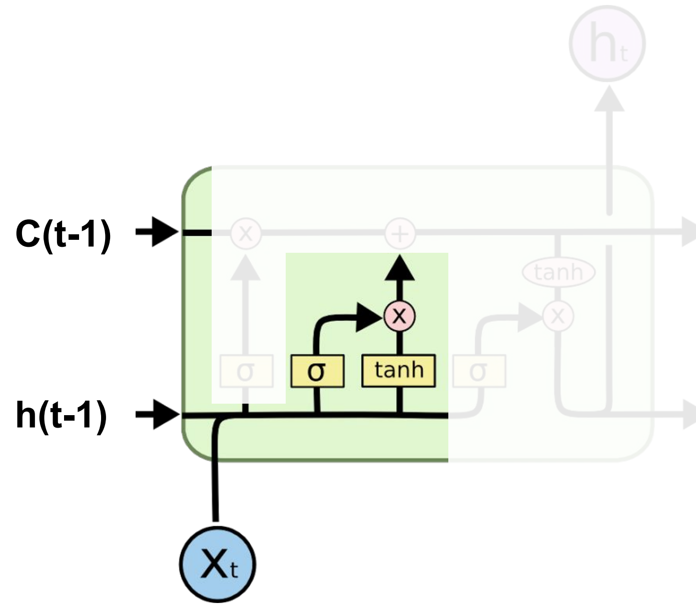
($\boldsymbol{W_f}$ includes bias)

Source: colah.github.io, http://blog.echen.me/2017/05/30/exploring-lstms

LSTM - input gate

2. Input Gate:

- Remember only some parts of the current input & previous working memory.
- Decide what to remember based on current input & previous working memory.



Eg.: The latest murder news, not an irrelevant character.
A new variable, not a comment.

input_gate(t) = ***sigmoid***($\boldsymbol{W_i}$ (**x**(t), **h**(t-1)))

input_information(t) = ***tanh***($\boldsymbol{W_a}$ (**x**(t), **h**(t-1)))

relevant_input_information(t) = ***input_gate***(t) .* ***input_information***(t)

The ***input_gate*** has a ***sigmoid*** activation so as to act as a fraction on the input information - hence deciding what fraction to consider and what fraction to let go.

The ***input_information*** has a ***tanh*** activation so as to squashify the information between -1 and 1.

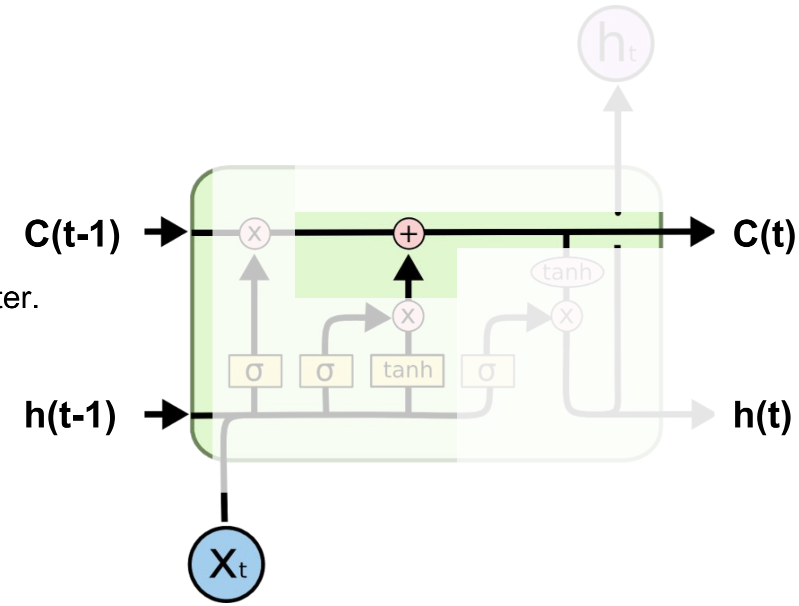
Source: colah.github.io, http://blog.echen.me/2017/05/30/exploring-lstms

LSTM - update long-term memory

Update long-term memory:

- Add the relevant input information to the long-term memory.

Eg.: Remember the latest news, don't remember an irrelevant character.
Remember a new variable, don't remember a comment.



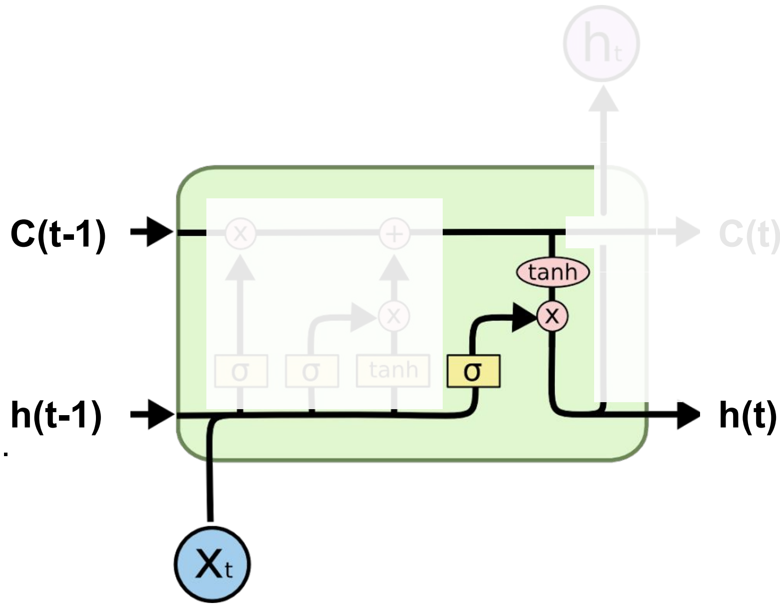
C(*t*) = ***remembered_cell_state***(*t*) + ***relevant_input_information***(*t*)

Source: colah.github.io, http://blog.echen.me/2017/05/30/exploring-lstms

LSTM - output gate

3. Output Gate:

- Having saved relevant information into long-term memory, retrieve some working memory.
- Decide what to retrieve based on current input & previous working memory.



Eg.: Retrieve the name of murderer, don't retrieve the parents of victim.
Retrieve the updated variable, don't retrieve the nesting structure.

output_gate(*t*) = ***sigmoid***(W_o (***x***(*t*), ***h***(*t*-1)))

retrieved_memory(*t*) = ***tanh***(***C***(*t*))

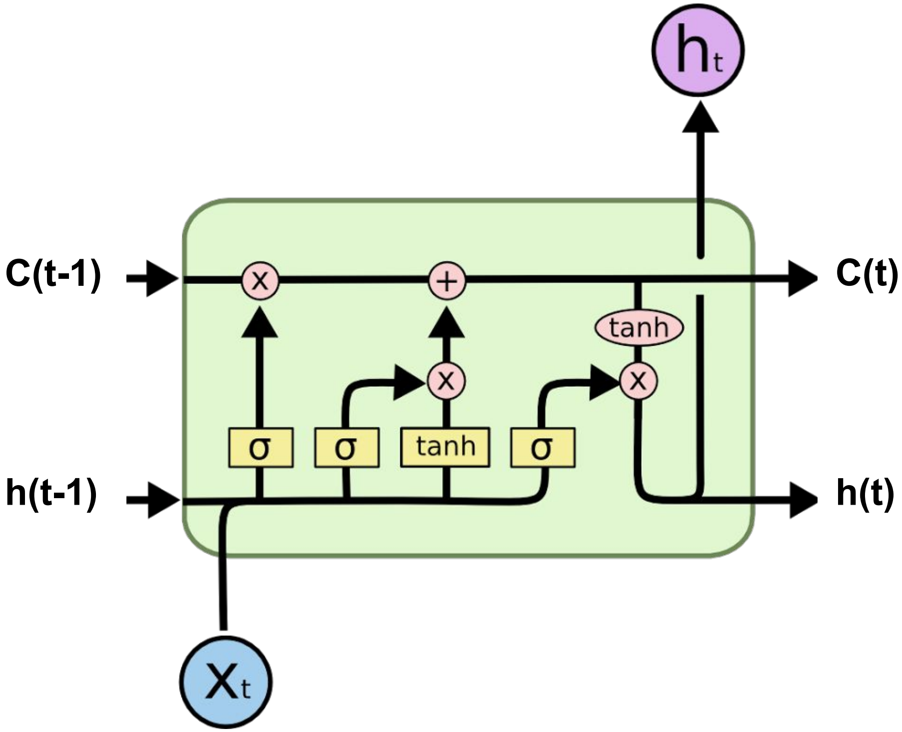
h(*t*) = ***output_gate***(*t*) .* ***retrieved_memory***(*t*)

The ***output_gate*** has a ***sigmoid*** activation so as to act as a fraction on the retrieved information - hence deciding what fraction to keep and what fraction to ignore.

The ***retrieved_memory*** has a ***tanh*** activation so as to squashify the retrieved information between -1 and 1.

Source: colah.github.io, http://blog.echen.me/2017/05/30/exploring-lstms

LSTM



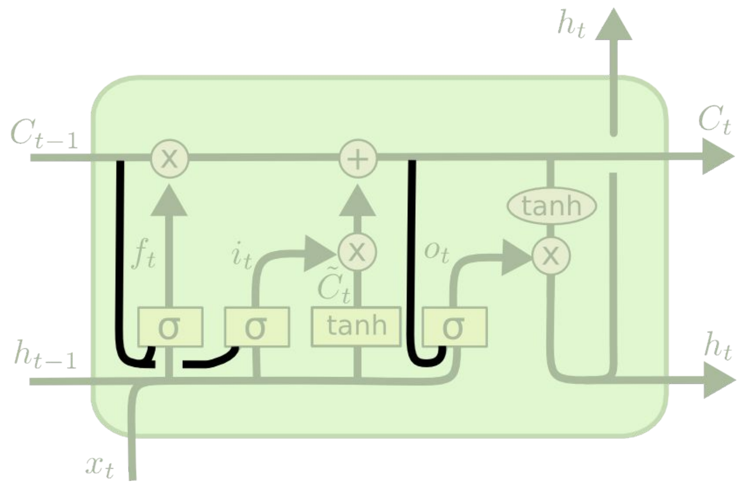
SUMMARY:

Using (***x****(t)*, ***h****(t-1)*), i.e. current input and previous working memory,

- forget unimportant long-term memory,
- compute relevant input information, and add it to the long-term memory,
- retrieve relevant working memory from long-term memory.

Source: colah.github.io, http://blog.echen.me/2017/05/30/exploring-lstms

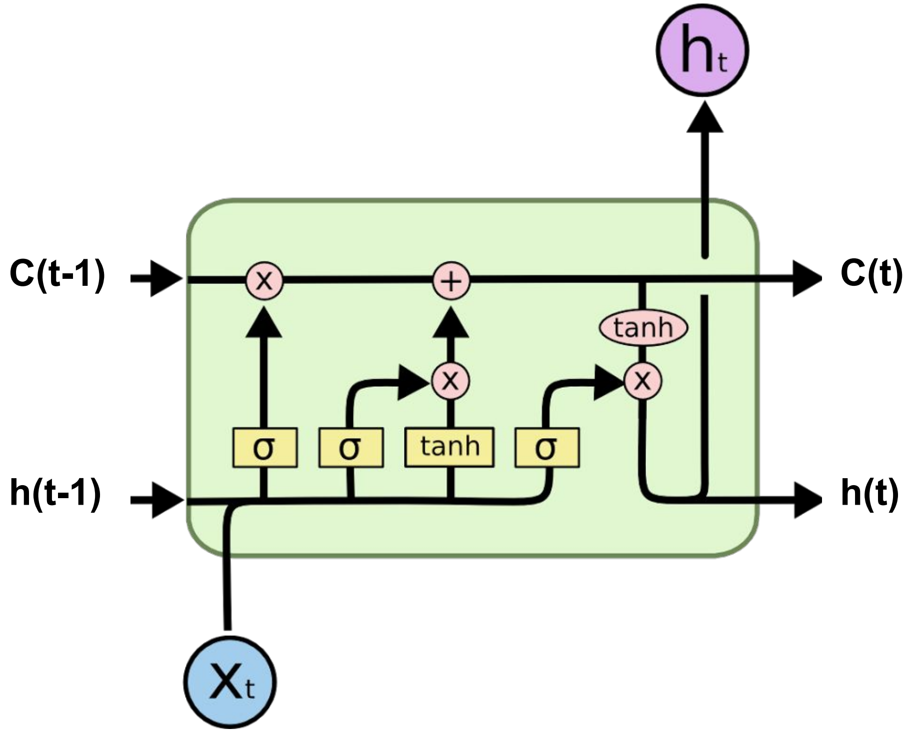
Variant - Peephole



- Same as LSTM, except use long-term memory as well for all decisions:
  - (***x***(*t*), ***h***(*t*-*1*), ***C***(*t-1*)) for forget and input gates,
  - (***x***(*t*), ***h***(*t*-*1*), ***C***(*t*)) for output gate.

Source: colah.github.io, http://blog.echen.me/2017/05/30/exploring-lstms

LSTM



NOTE:

- De-coupling short-term and long-term memory avoids vanishing/exploding gradient (haphazard updation of cell state in vanilla RNN was primary culprit)
- Methodical design of structure - no “mystery” as to why it works!

Source: colah.github.io, http://blog.echen.me/2017/05/30/exploring-lstms

References

1. Hacker's guide to NN: http://karpathy.github.io/neuralnets/
2. Interactive visualization: http://neuralnetworksanddeeplearning.com/chap4.html
3. LSTMs: colah.github.io
4. Quora answer: https://www.quora.com/What-is-an-intuitive-explanation-of-LSTMs-and-GRUs/answer/Edwin-Chen-1?share=b6d3b009&srid=Xfgu
5. http://blog.echen.me/2017/05/30/exploring-lstms/