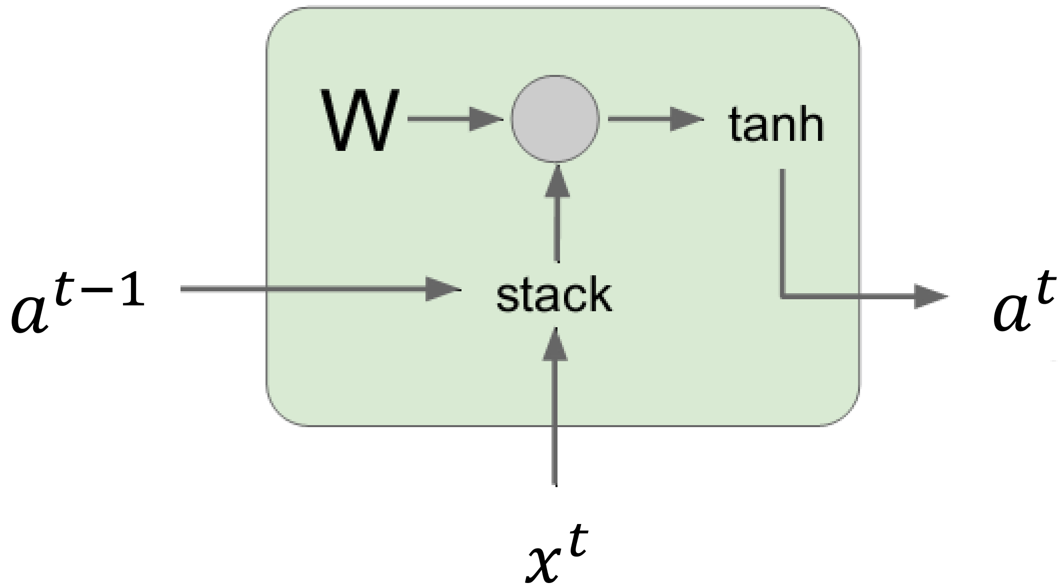
LSTM and GRU

Handling Vanishing Gradients

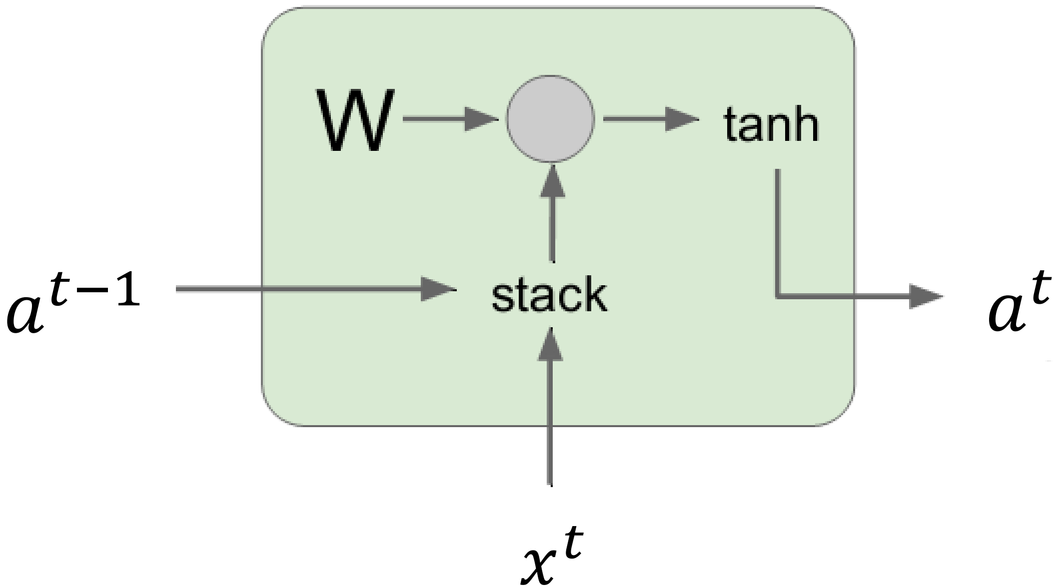
Based on lecture slides of the course CS231n: Stanford Univ.

Vanilla RNN Gradient Flow



$$a^t = \tanh(W_a\,[a^{t-1}, x^t] + b_a)$$

Vanilla RNN Gradient Flow

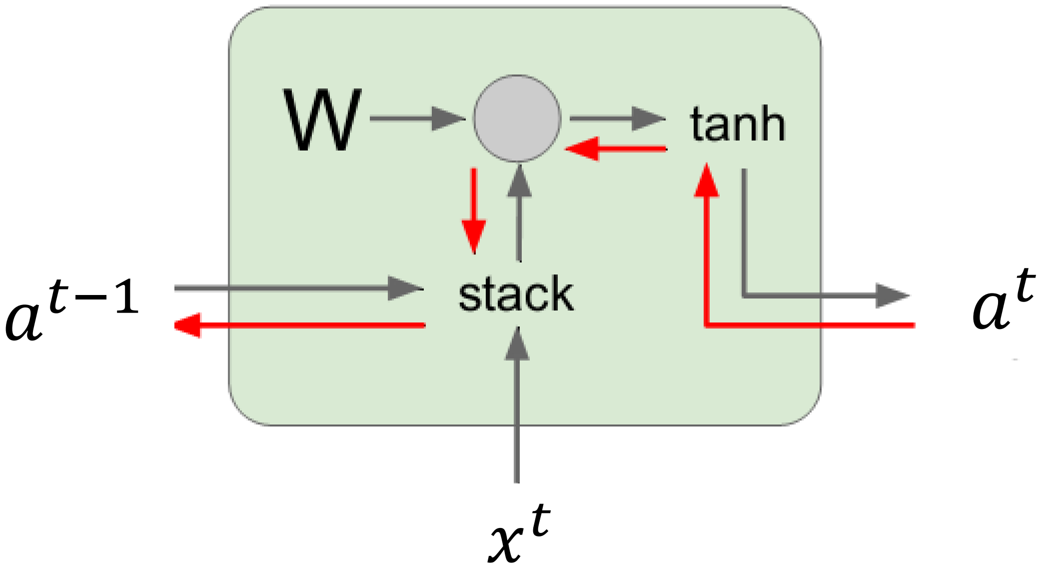


$$a^t = \tanh(W_a\,[a^{t-1}, x^t])$$

For the sake of convenience we take b_a = 0

Vanilla RNN Gradient Flow

Backpropagation from $a^{<t>}$ to $a^{<t-1>}$ multiplies by ${W_{aa}}^T$

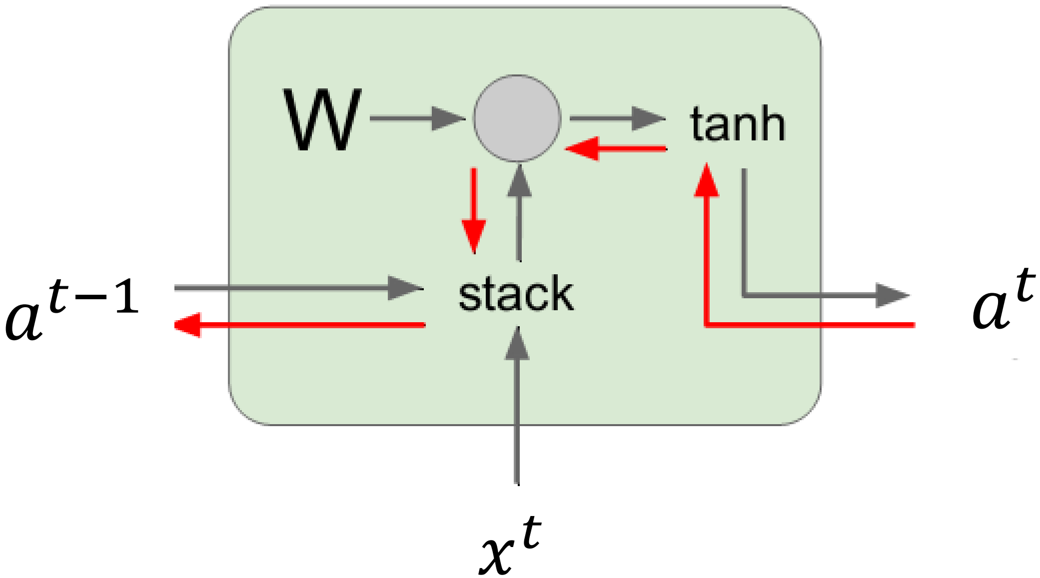


$$a^t = \tanh(W_a\,[a^{t-1}, x^t])$$

$$\frac{\partial L}{\partial a^{t-1}} = \frac{\partial L}{\partial a^t}\frac{da^t}{da^{t-1}}$$

Vanilla RNN Gradient Flow

Backpropagation from $a^{<t>}$ to $a^{<t-1>}$ multiplies by $W_{aa}{}^{T}$

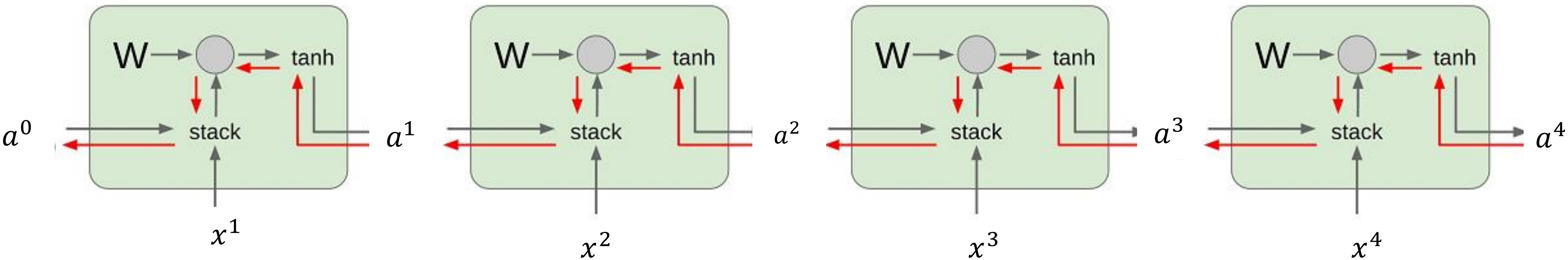


$$a^t = \tanh(W_a\,[a^{t-1}, x^t])$$

$$\frac{\partial L}{\partial a^{t-1}} = \frac{\partial L}{\partial a^t}\frac{da^t}{da^{t-1}}$$

$$= \frac{\partial L}{\partial a^t}\{1 - tanh^2\,(W_a[a^{t-1}, x^t])\}W_a$$

Vanilla RNN Gradient Flow

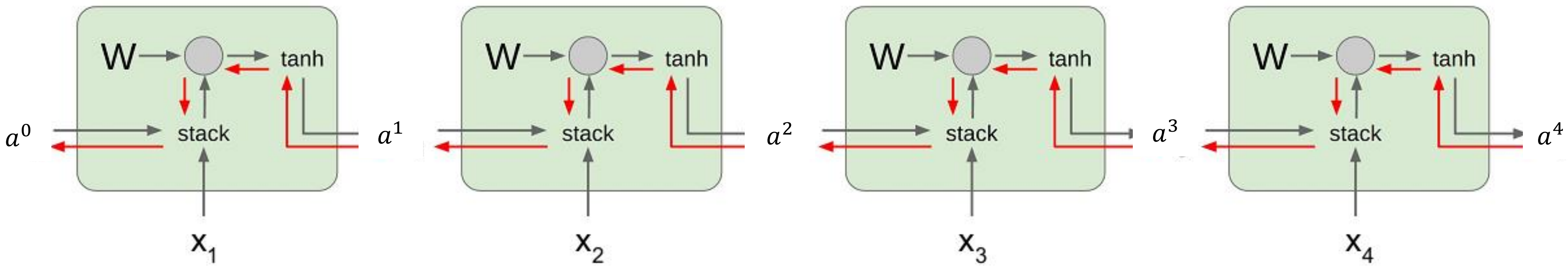


- $\frac{\partial L}{\partial a^0} = \frac{\partial L}{\partial a^4} \frac{da^4}{da^3} \frac{da^3}{da^2} \frac{da^2}{da^1} \frac{da^1}{da^0}$

$$= \frac{\partial L}{\partial a^4}\{1 - tanh^2\,(W_a[a^3, x^4])\} \cdot \{1 - tanh^2\,(W_a[a^2, x^3])\}$$

$$\cdot \{1 - tanh^2\,(W_a[a^1, x^2])\} \cdot \{1 - tanh^2\,(W_a[a^0, x^1])\} (W_a)^4$$

Vanilla RNN Gradient Flow



- Computing gradient involves many factors of W and contribution of tanh.
- May result in exploding gradients or vanishing gradients.
- Gradient values are clipped if they are larger than a threshold.
- What if the gradient values are very small ?
    - Use LSTM or GRU.

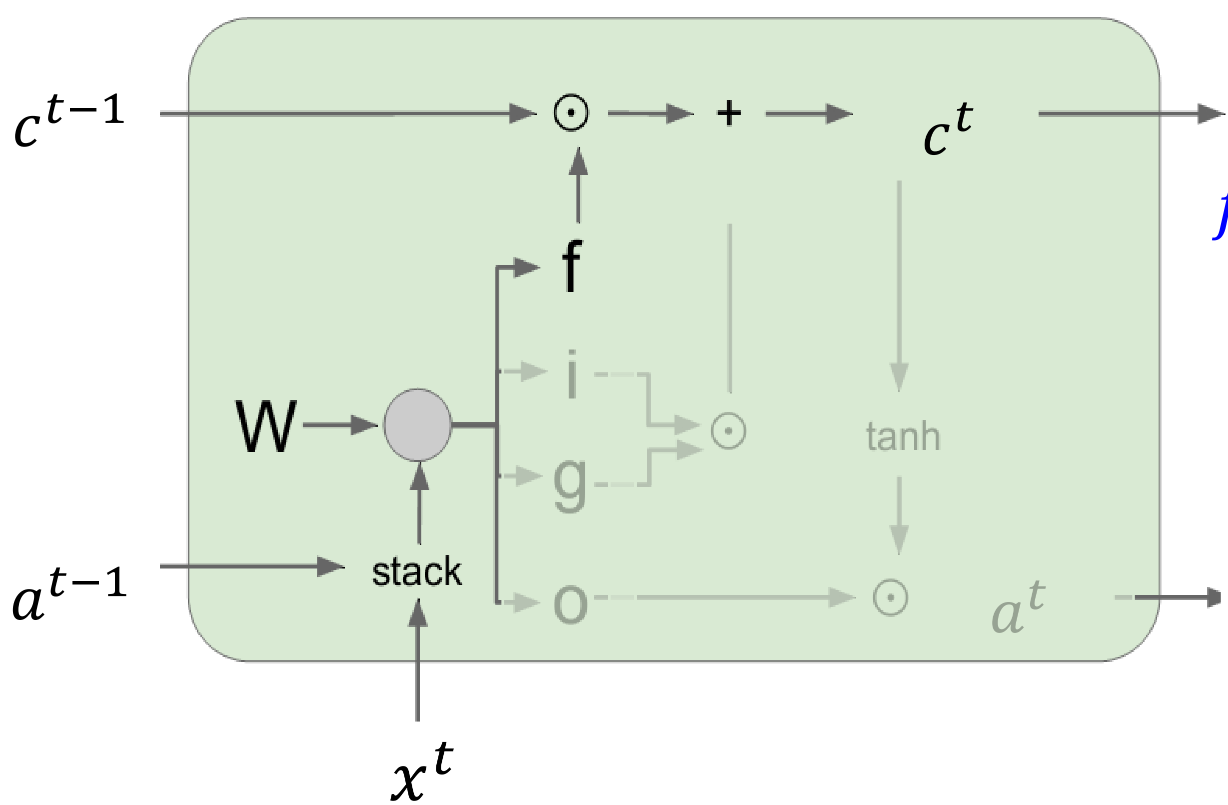
Long Short Term Memory (LSTM)

- A special kind of RNN, capable of learning long-term dependencies.
- Introduced by Hochreiter & Schmidhuber (1997)
- LSTMs are designed to avoid the long-term dependencies.
- Keeping relevant information for long period of time is their default behavior.
- Have been refined and popularized by many researchers.
- Successfully applied in many problems that have sequential behaviour.

How Does LSTM Work ?

- LSTM is designed using several gates as described here.
- Forget Gate: To decide what to forget and what information to carry forward.
- For example in language modeling, given all previous inputs in a piece of text, one wants to keep track of a given subject being plural /singular.
- When the state of subject is fixed at the point of time, that it is singular, one would like to have a way of getting rid of those singular \plural memory states. Forget Gate is a mechanism to handle this issue.

Forget Gate



$$f = \sigma(W_f\,[a^{t-1}, x^t] + b_f)$$

Forget Gate

$$a^t = \begin{bmatrix} 2.9 \\ 0.5 \\ 0.7 \end{bmatrix} \;;\; W_f = \begin{bmatrix} 1.0 & 0.4 & 0.7 \\ -0.6 & -0.5 & -0.2 \end{bmatrix} ; \; b_f = \begin{bmatrix} 0.0 \\ 0.0 \end{bmatrix}$$

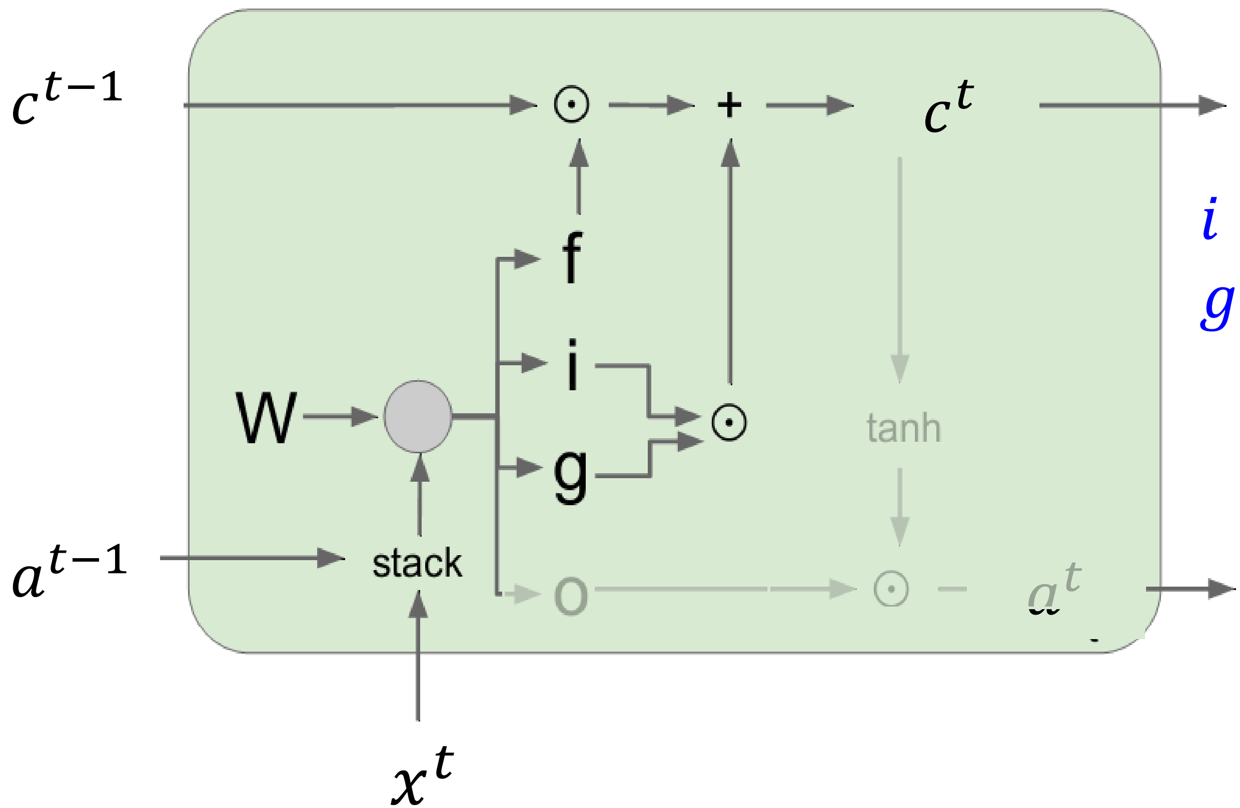
$$f = \sigma(W_f\,[a^{t-1}] + b_f) = \sigma\begin{pmatrix} 3.59 \\ -2.13 \end{pmatrix} = \begin{pmatrix} 0.7941 \\ 0.1062 \end{pmatrix}$$

Forget

This is called a *gate* because the sigmoid function squashes the values of these vectors between 0 and 1.

Elementwise multiplication with another vector defines which components of that other vector we want to “carry forward” and which to forget. Similarly for other gates.

Update (Input) Gate

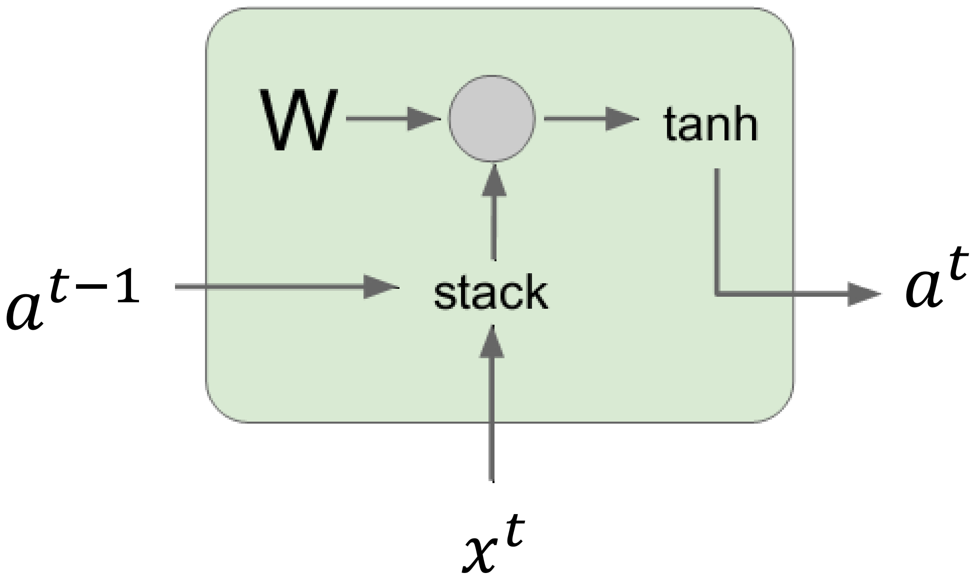


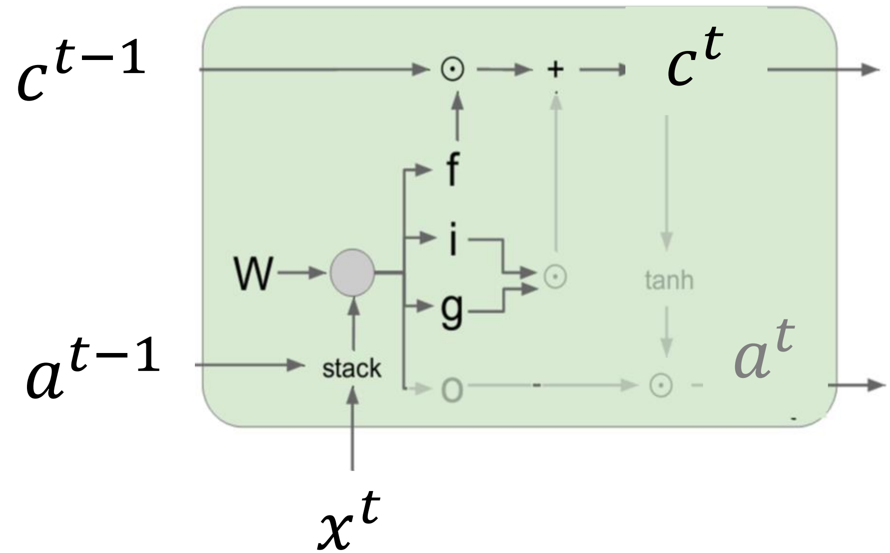
$$i = \sigma(W_i\,[a^{t-1}, x^t] + b_i)$$
$$g = \tanh(W_g\,[a^{t-1}, x^t] + b_g)$$

Update (Input) Gate

- Once it is finalized things to forget like ‘singular\plural state of the subject’ in language modeling, there has to be a way to decide whether to update (singular \plural) and how much to update.
- The input gate lets one decide how much of the computed state for the current input one wants to carry forward.
- Here g is a “candidate” hidden state.
- The candidate hidden state is based on the current input and the previous hidden state.
- Recall that g is exactly the same as we had in vanilla RNN,

Vanilla RNN vs LSTM

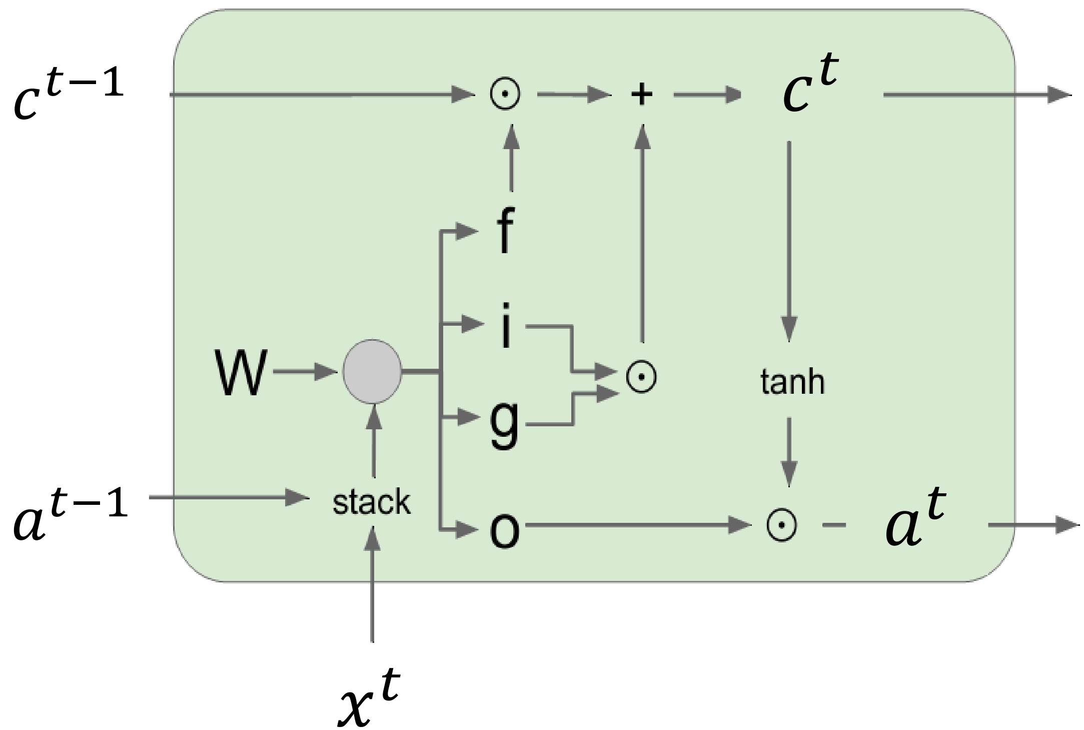




$$a^t = \tanh(W_a\,[a^{t-1}, x^t] + b_a)$$

$$i = \sigma(W_i\,[a^{t-1}, x^t] + b_i)$$
$$g = \tanh(W_g\,[a^{t-1}, x^t] + b_g)$$

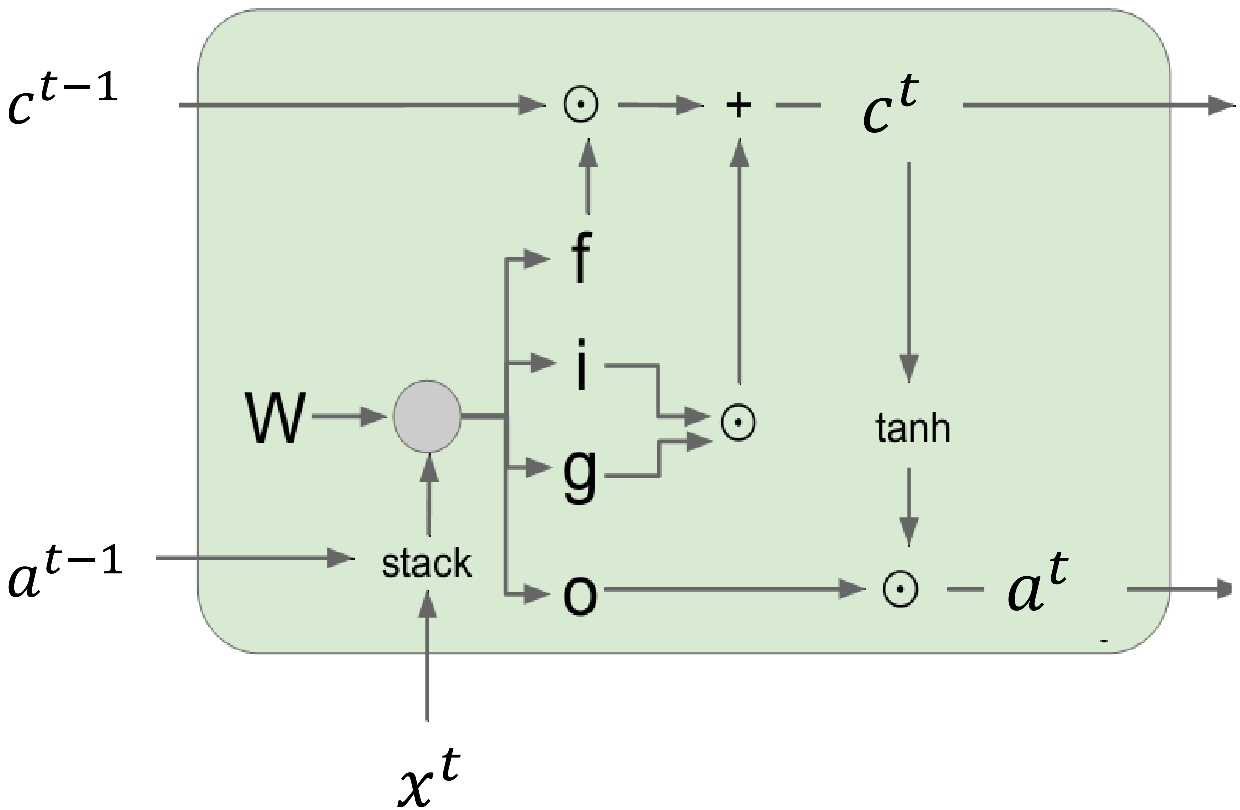
Output Gate



$$o = \sigma(W_o\,[a^{<t-1>}, x^{<t>}] + b_o)$$

Long Short Term Memory (LSTM)

[Hochreiter & Schmidhuber 1997.
Long short-term memory]



$$f = \sigma(W_f\,[a^{t-1}, x^t] + b_f)$$

$$i = \sigma(W_i\,[a^{t-1}, x^t] + b_i)$$

$$g = \tanh(W_g\,[a^{t-1}, x^t] + b_g)$$

$$o = \sigma(W_o\,[a^{<t-1>}, x^{<t>}] + b_o)$$

$$c^t = f * c^{t-1} + i * g$$

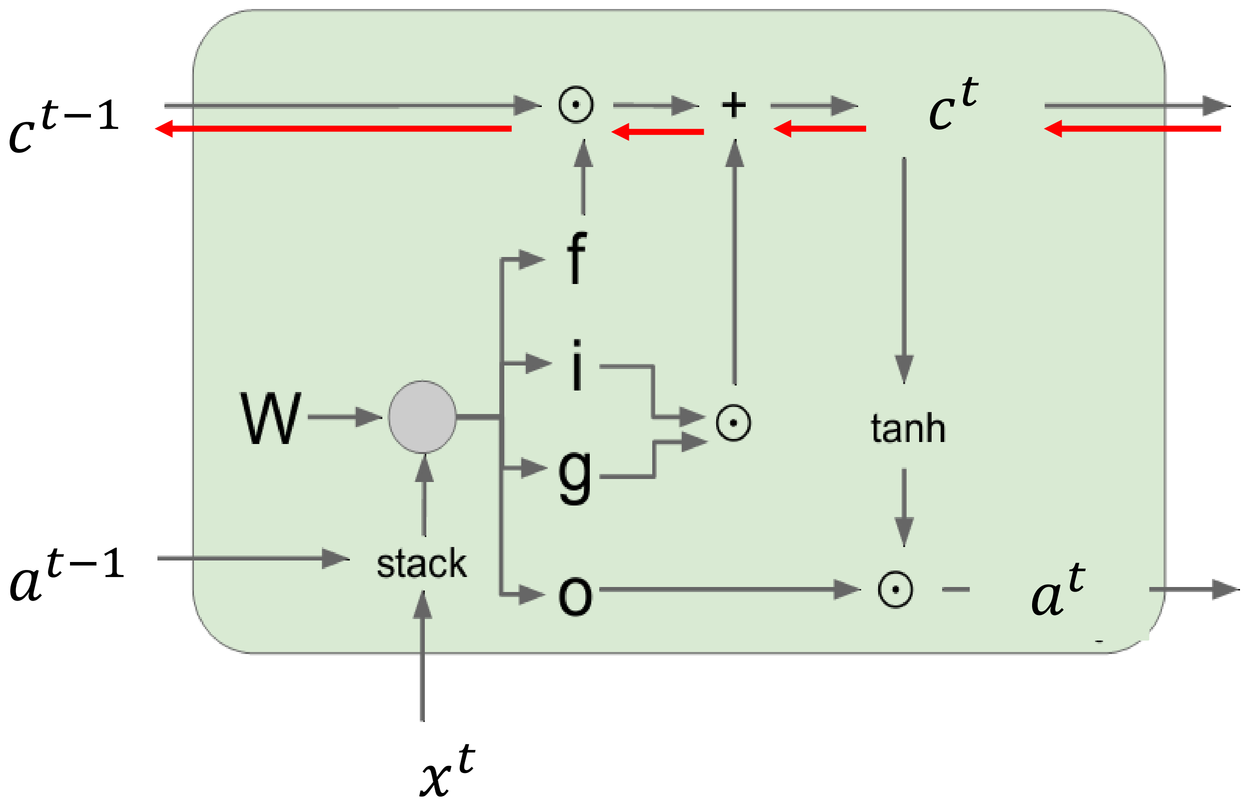
$$a^t = o * \tanh(c^t)$$

Long Short Term Memory (LSTM)

- Equations in typical Vanilla NN
  - $a^t = \tanh(W_a\,[a^{t-1}, x^t])$
- Modifications in LSTM
  - $f = \sigma(W_f\,[a^{t-1}, x^t] + b_f)$: Forget gate, whether to erase or forget cell state
  - $i = \sigma(W_i\,[a^{t-1}, x^t] + b_i)$: Input (update) gate, whether to write to cell
  - $g = \tanh(W_g\,[a^{t-1}, x^t] + b_g)$: Gate gate, How much to write to cell
  - $o = \sigma(W_o\,[a^{<t-1>}, x^{<t>}] + b_o)$: Output gate, whether to write to the cell (to send to the next hidden state)
  - $c^t = f * c^{t-1} + i * g$: Whether\what to forget and whether\what to update
  - $a^t = o * \tanh(c^t)$: Which output to use for the next layer

Back Propagation in LSTM

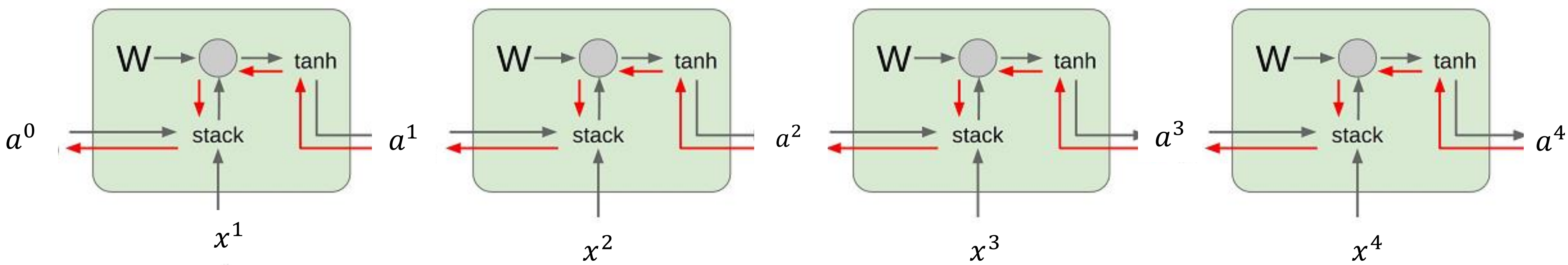
- Back propagation from c^t to c^{t-1} .
- Note that there is only elementwise multiplication by *f*, no matrix multiplication by W.



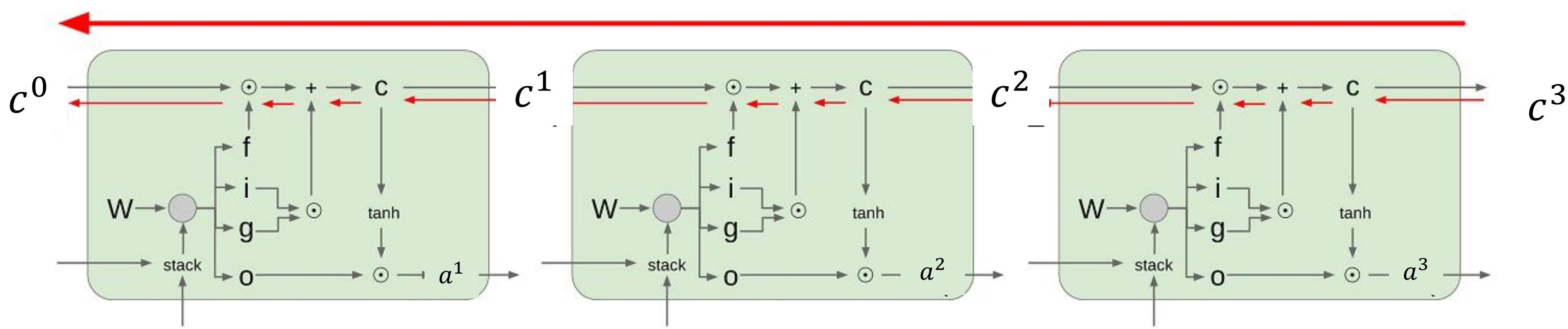
$$c^t = f * c^{t-1} + i * g$$

$$\frac{\partial L}{\partial c^{t-1}} = \frac{\partial L}{\partial c^t} \frac{d c^t}{d c^{t-1}}$$

Traditional RNN vs LSTM RNN



Gradient flow in traditional RNN



Gradient flow in LSTM

GRU: Another Variant

$$\tilde{c}^t = \tanh(W_c[r * c^{t-1}, x^t])$$

$$i = \sigma(W_i[\, c^{t-1}, x^t])$$

$$r = \sigma(W_r[\, c^{t-1}, x^t])$$

$$c^t = i * \tilde{c}^t + (1 - i) * c^{t-1}$$

GRU vs LSTM units

GRU

$$\tilde{c}^t = \tanh(W_c[r * c^{t-1}, x^t] + b_c)$$

$$i = \sigma(W_i[\, c^{t-1}, x^t] + b_i)$$

$$r = \sigma(W_r[\, c^{t-1}, x^t] + b_r)$$

$$c^t = i * \tilde{c}^t + (1 - i) * c^{t-1}$$

$$a^t = c^t$$

LSTM

$$\tilde{c}^t = \tanh(W_c[a^{<t-1>}, x^{<t>}] + b_c)$$

$$i = \sigma(W_i[\, a^{<t-1>}, x^{<t>}] + b_i)$$

$$f = \sigma(W_f[\, a^{<t-1>}, x^{<t>}] + b_f)$$

$$o = \sigma(W_o[\, a^{<t-1>}, x^{<t>}] + b_o)$$

$$c^t = \ i * \tilde{c}^t + f * c^{t-1}$$

$$a^t = o * \tanh(c^t)$$

Summary

- RNNs allow a lot of flexibility in architecture design.
- Vanilla RNNs are simple but don’t work very well in many applications.
- Back propagation of gradients can lead to exploding or vanishing gradient problems.
- Exploding is controlled with gradient clipping. Vanishing gradients are handled by changing the RNN architecture.
- LSTM or GRU solve the problem of vanishing gradients: Their additive interactions improve gradient flow.
- Better/simpler architectures are a hot topic of current research.
- Better understanding (both theoretical and empirical) is needed.

Acknowledgement

- Due acknowledgement to Coursera for their course material on the courses offered by deeplearning.ai
  - Neural Networks and Deep Learning.
  - Sequence models.

Next !

- We shall build a simple RNN step by step in Keras

- Projects will be discussed in the next lab session.

- In the project: Marks will be awarded for
  - Data preprocessing
  - Task specified (as detailed in the project idea)
  - Bonus marks for any innovative idea.