# Why LSTM outperforms RNN?

Xintao Huan

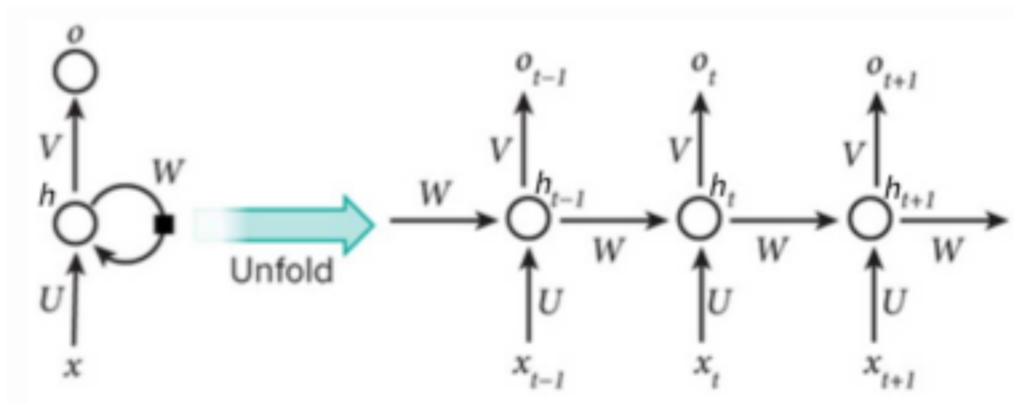
UNIVERSITY OF LIVERPOOL

# Outline

- Quick Review of RNN
- Investigate RNN
- Key Issue
- Solution - LSTM

UNIVERSITY OF LIVERPOOL

# Quick Review of RNN




Recurrent Neural Network (RNN) and the unfolding in time of the computation involved in its forward computation.

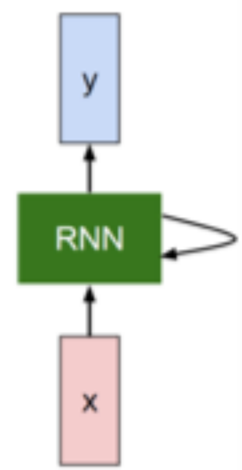
$$\boxed{h_t} = \boxed{f_W}(\boxed{h_{t-1}}, \boxed{x_t})$$

new state

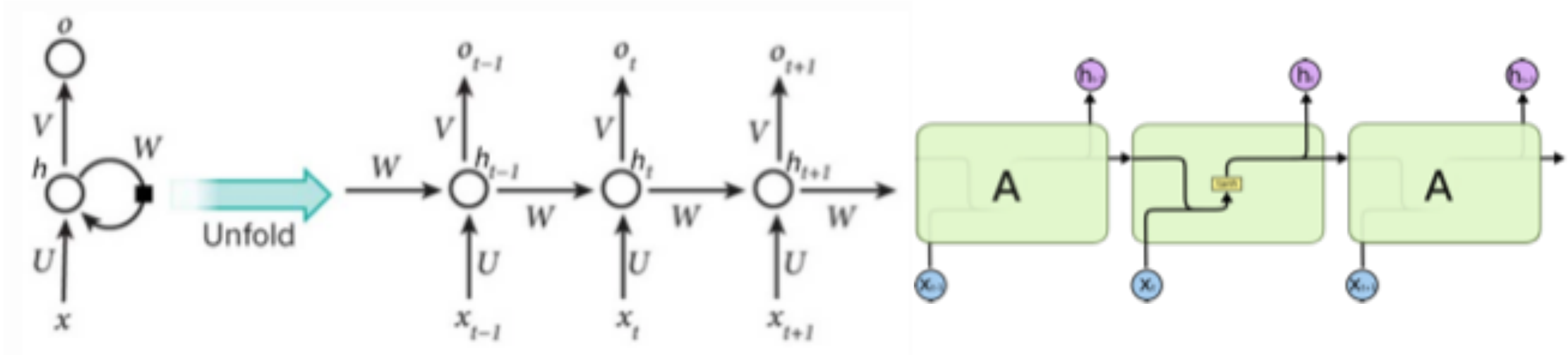
function with parameters W

old state

input vector at time step t



# Investigate RNN



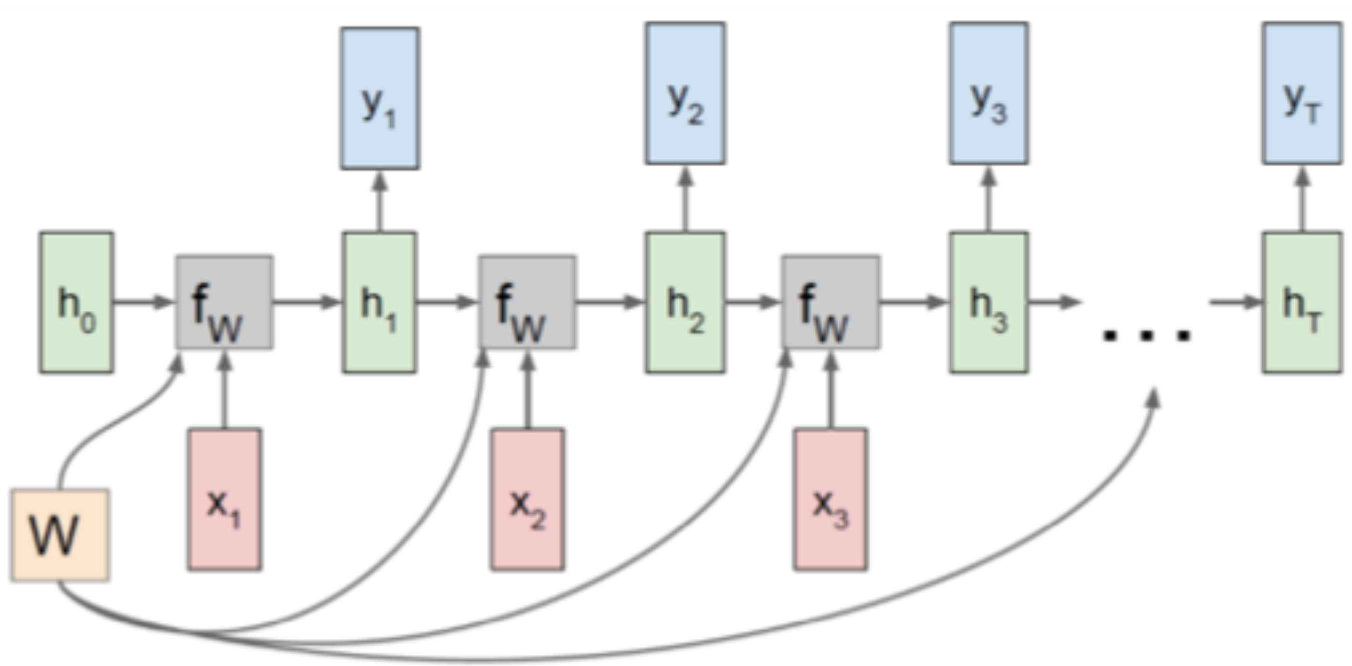

$$h_t = f_W(h_{t-1}, x_t)$$

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

$$y_t = W_{hy}h_t$$

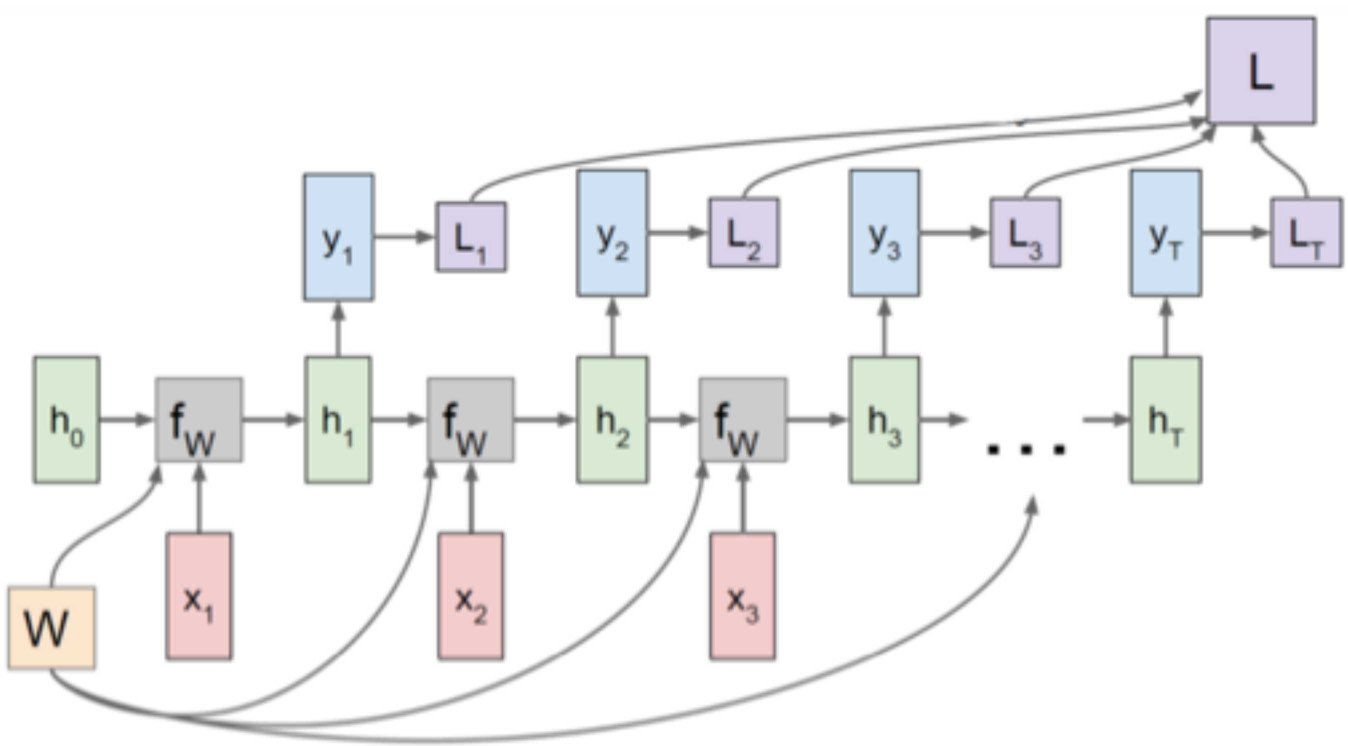
W*hh*: weight between hidden layers
W*xh*: weight between input layer and hidden layer
W*hy*: weight between hidden layer and output layer

# Investigate RNN




In the computation of the hidden layer, Weight Matrix **W** is **shared** in **all time-steps**!

# Investigate RNN




In TRAINING,

we compare the output of the time-step $\mathbf{y_t}$ with the reference result, then the loss $\mathbf{L_t}$ is obtained, and sequence loss $\mathbf{L}$ will be back propagated from the end time-step to the first time-step.

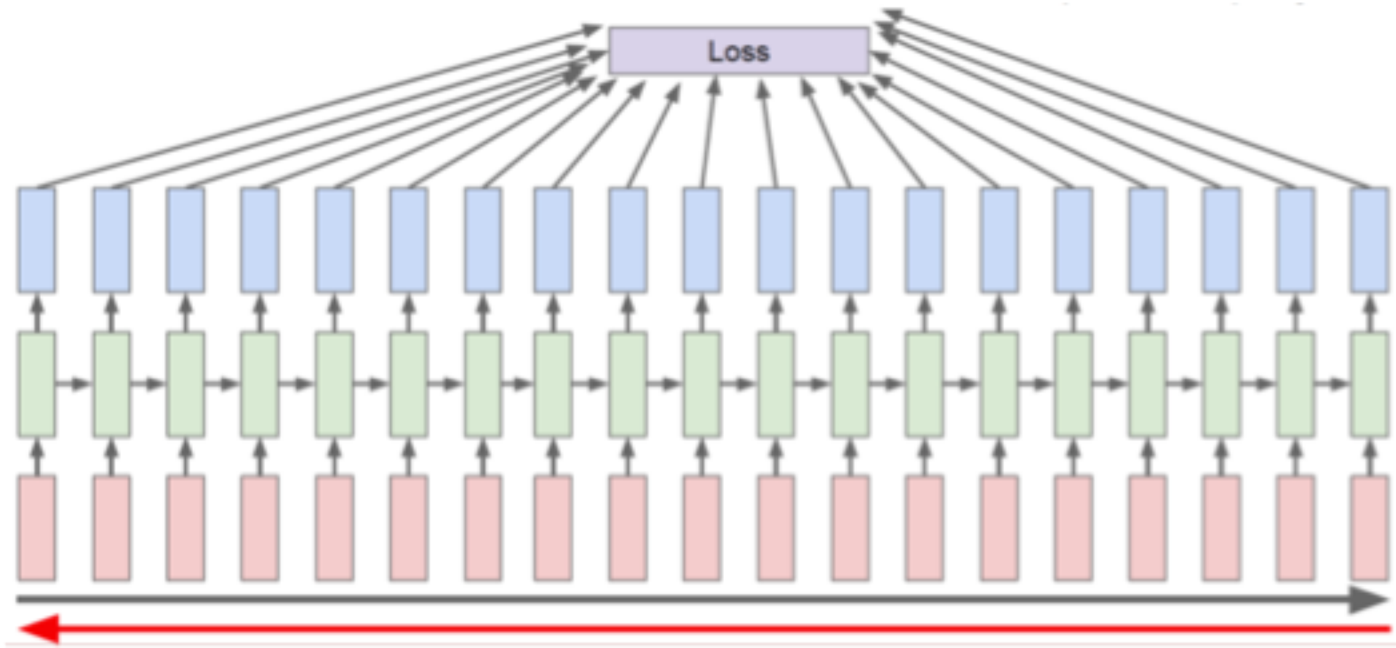
Next,

we employ **Stochastic Gradient Descent (SGD)** to minimize the loss and update the parameters in **W**.

UNIVERSITY OF LIVERPOOL

# Investigate RNN

Back Propagation Through Time (BPTT)

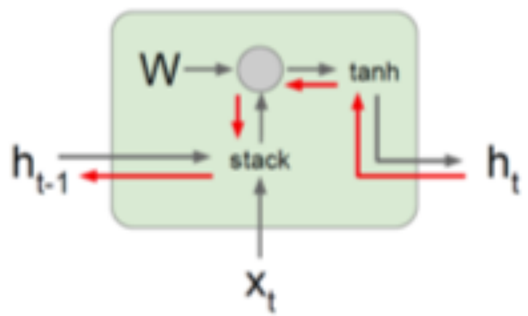



Forward through entire sequence to compute the **Loss**.

Backward through entire sequence to **minimize the loss** and **update the parameters** in **W**.

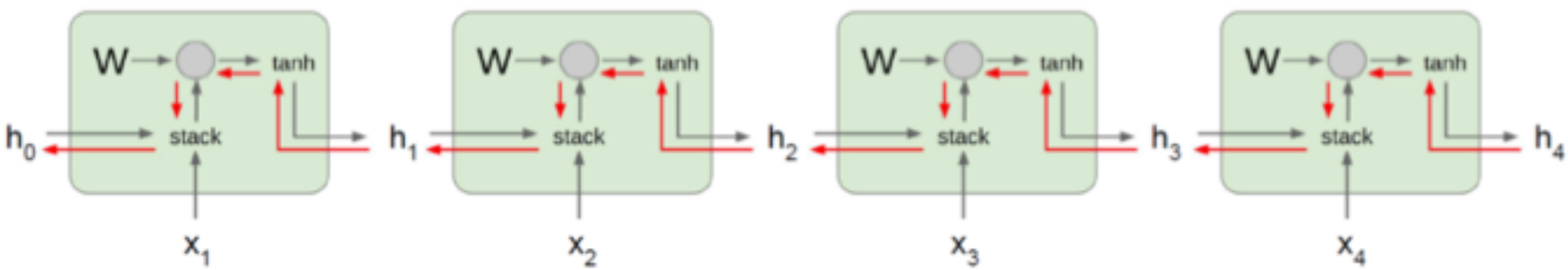
# Key Issue

In every backpropagation from $h_t$ to $h_{t-1}$:




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

**W is multiplied!**




e.g. computing gradient of $h_0$ involves many factors of W !

If sequence is long enough and
W > 1, ***exploding gradients!***
W < 1, ***vanishing gradients!***

UNIVERSITY OF LIVERPOOL

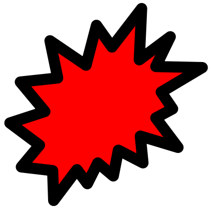
# Key Issue

*Exploding gradients:*

Employ *Gradient Clipping* to scale the gradient (e.g. cut the value).

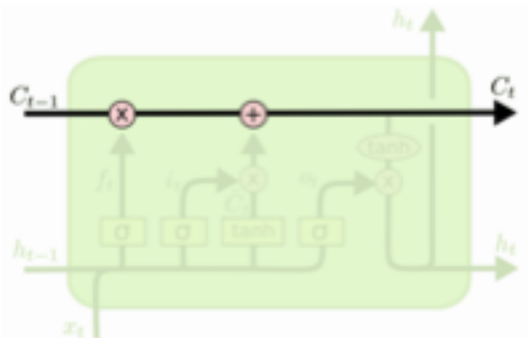
```
grad_norm = np.sum(grad * grad)
if grad_norm > threshold:
    grad *= (threshold / grad_norm)
```

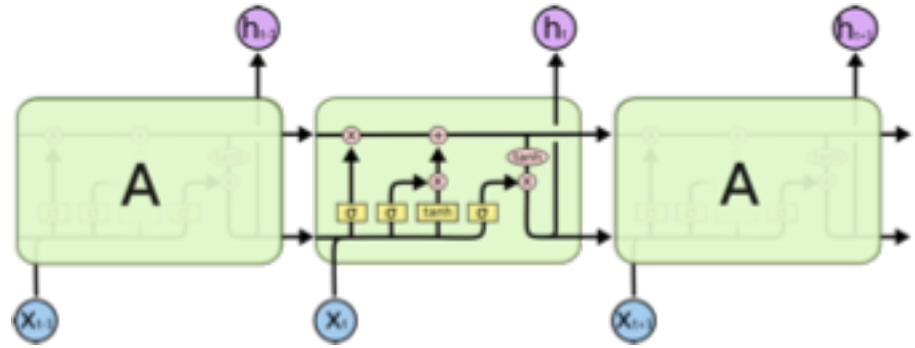
*Vanishing gradients:*



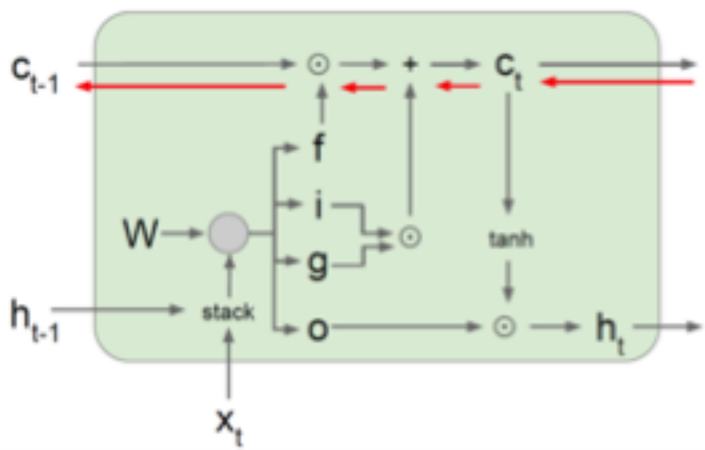
**Change RNN architecture !**

# Solution - LSTM




With the cell state, it runs straight down the entire chain, with only some **minor linear interactions (NOT matrix multiplication like in RNN)**
It's very easy for information to just flow along it unchanged.




input gate
forget gate
output gate
update gate

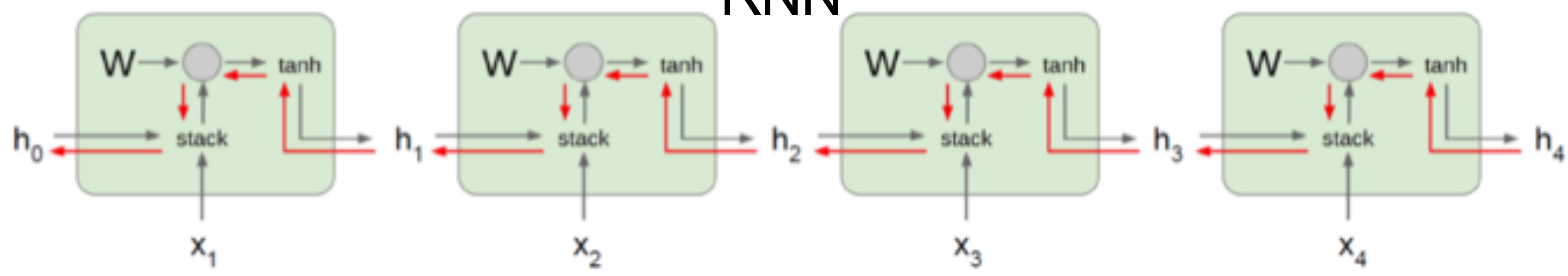
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

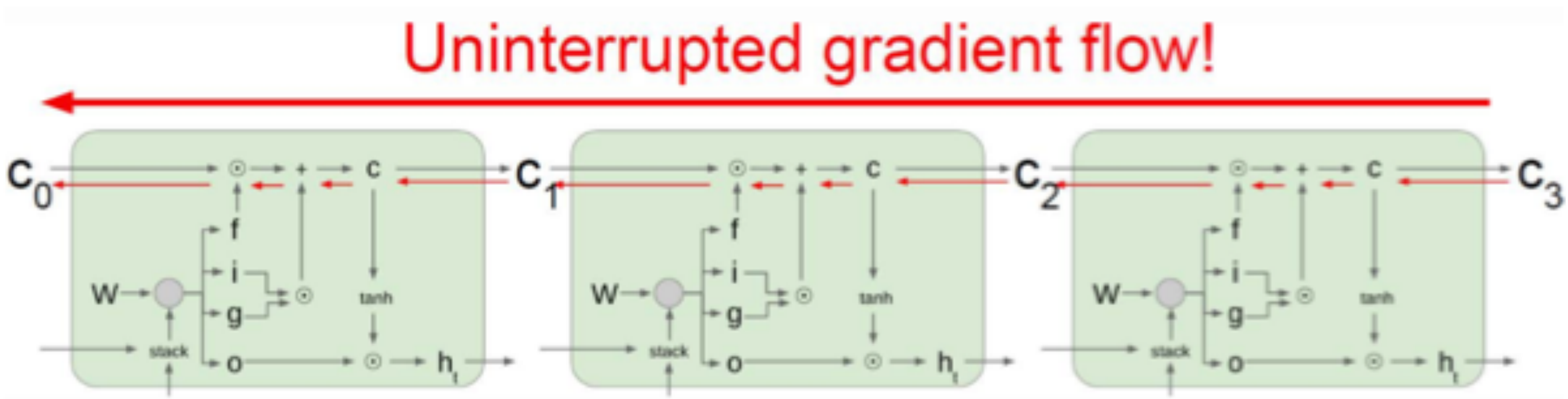
$$c_t = f \odot c_{t-1} + i \odot g$$

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

# Solution - LSTM

RNN




LSTM

***Vanishing gradients SOLVED!***

# References

1. Understanding LSTM http://colah.github.io/posts/2015-08-Understanding-LSTMs/
2. The Unreasonable Effectiveness of Recurrent Neural Networks http://karpathy.github.io/2015/05/21/rnn-effectiveness/
3. An Empirical Exploration of Recurrent Network Architectures http://proceedings.mlr.press/v37/jozefowicz15.pdf
4. A Critical Review of Recurrent Neural Networks for Sequence Learning https://arxiv.org/pdf/1506.00019.pdf
5. https://www.zhihu.com/question/44895610
6. https://my.oschina.net/u/2719468/blog/662099
7. https://yq.aliyun.com/articles/574218
8. https://juejin.im/post/59ae29d36fb9a024966cac99

Thanks for your attention!