DeepLearningLisbonMeetup

Recursive Neural Networks

Lisboa, 23 August 2019

Eduardo Fierro Farah

Applied AI, Unbabel

eduardo.farah@unbabel.com

01

Recursive NeurWHAT?

What are RNNs?

- You've probably seen different types of data by now.
- Recursive Neural Networks, or RNNs, are basically the first answer to the question: How can we process and build a network for ***variable length input***?
- For now, let's make it simple and assume we have a **fix size output.**
- **How do we map each sequence with different lengths to a token?**

$$x^1 = (x_1^1, x_2^1, x_3^1)$$

$$\downarrow$$

$$y^1$$

$$x^2 = (x_1^2, x_2^2, x_3^2, x_4^2, x_5^2)$$

$$\downarrow$$

$$y^2$$


TELL ME HOW!

Vanilla RNNs

- We want to get to a point similar to the multilayer perceptron, right?

$$y = \sigma\left(Wh_t\right)$$

Activation function

Some input

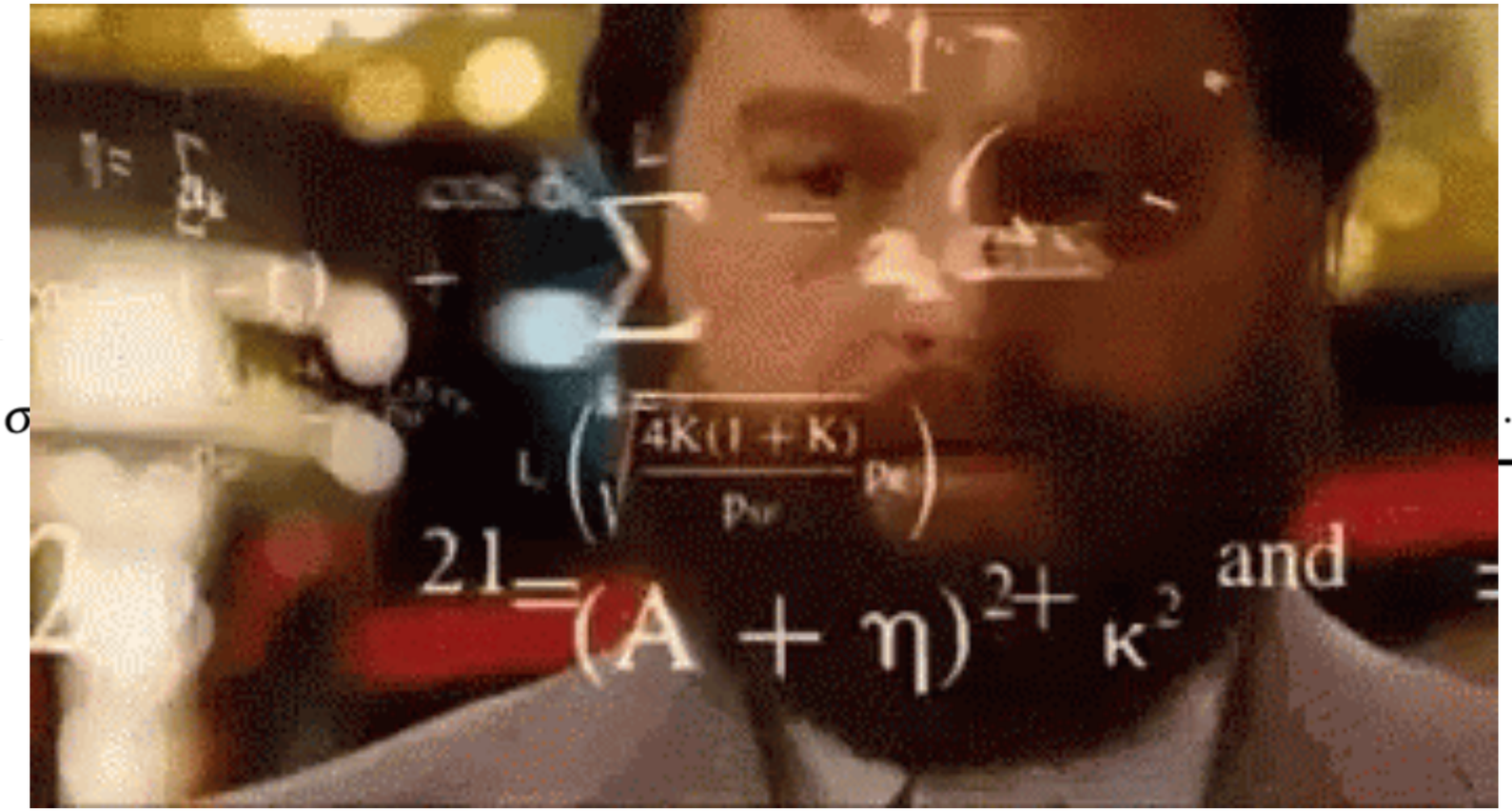
Some parameters
we want to learn

- IDEA: Let's do this, over and over again!

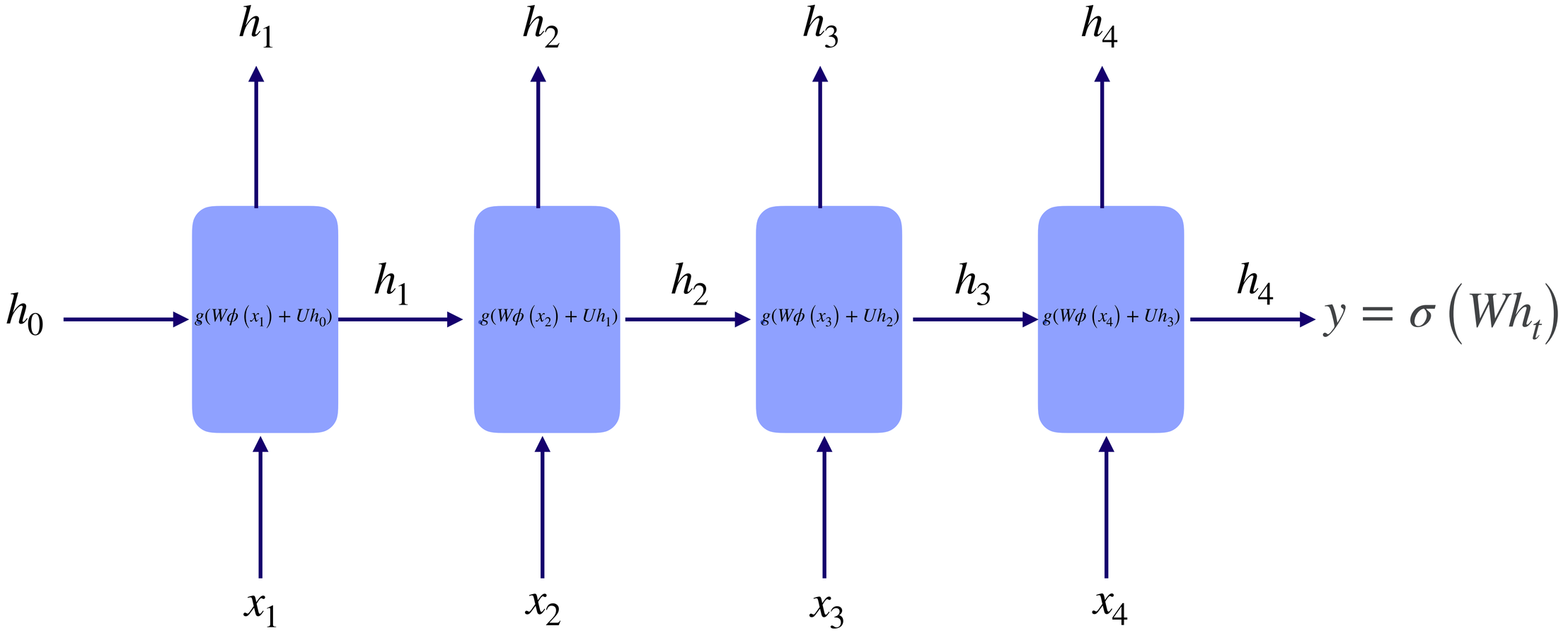
$$\mu = \sigma(\mathbf{v}^\top \underbrace{g(\mathbf{W}\boldsymbol{\phi}(x_l) + \mathbf{U}g(\mathbf{W}\boldsymbol{\phi}(x_{l-1}) + \mathbf{U}g(\mathbf{W}\boldsymbol{\phi}(x_{l-2}) + \cdots g(\mathbf{W}\boldsymbol{\phi}(x_1) + \mathbf{U}\mathbf{h}_0)\cdots)))}_{(a)\ \text{recurrence}}),$$

- From “Natural Language Understanding with Distributed Representation” Cho 2014. https://arxiv.org/abs/1511.07916

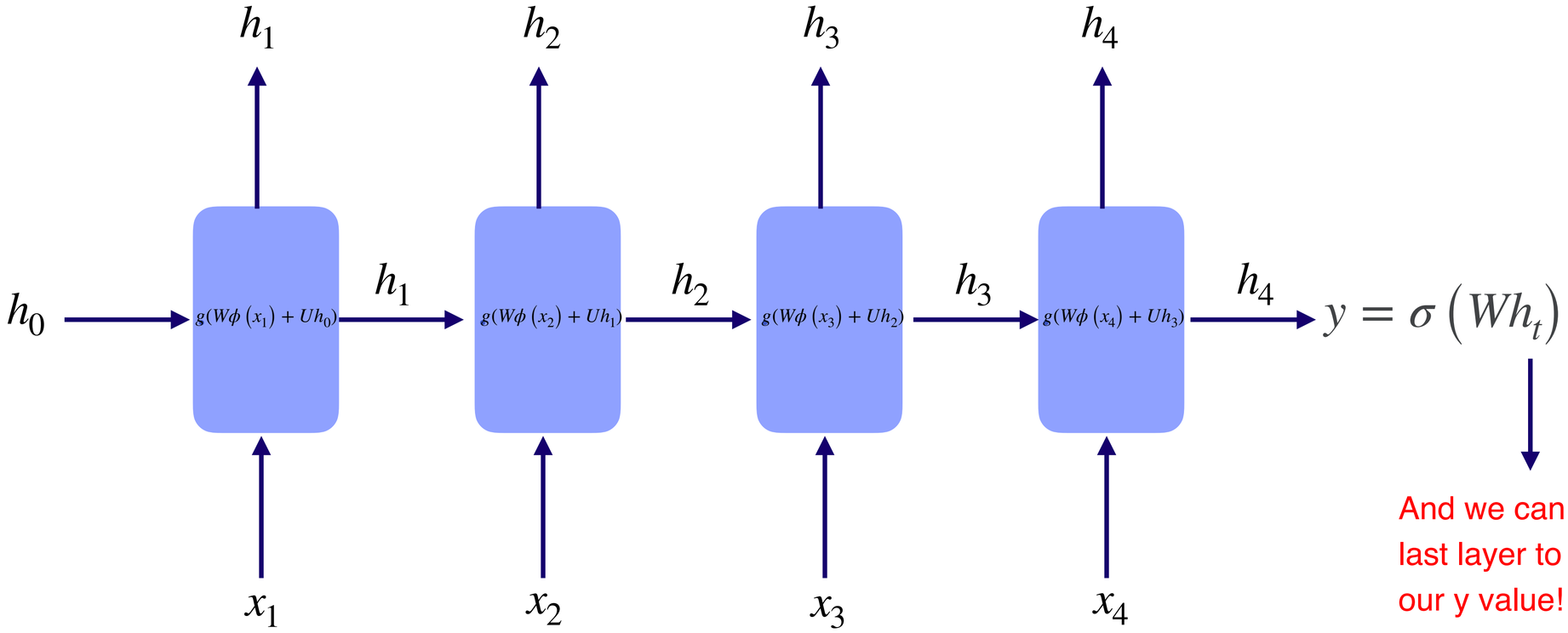
$\mu = \sigma$



$\cdots))))$,



$$\mu = \sigma(\mathbf{v}^\top \underbrace{g(\mathbf{W}\boldsymbol{\phi}(x_l) + \mathbf{U}g(\mathbf{W}\boldsymbol{\phi}(x_{l-1}) + \mathbf{U}g(\mathbf{W}\boldsymbol{\phi}(x_{l-2}) + \cdots g(\mathbf{W}\boldsymbol{\phi}(x_1) + \mathbf{U}\mathbf{h}_0)\cdots)))}_{(a)\text{ recurrence}}),$$



And we can use the last layer to predict our y value!

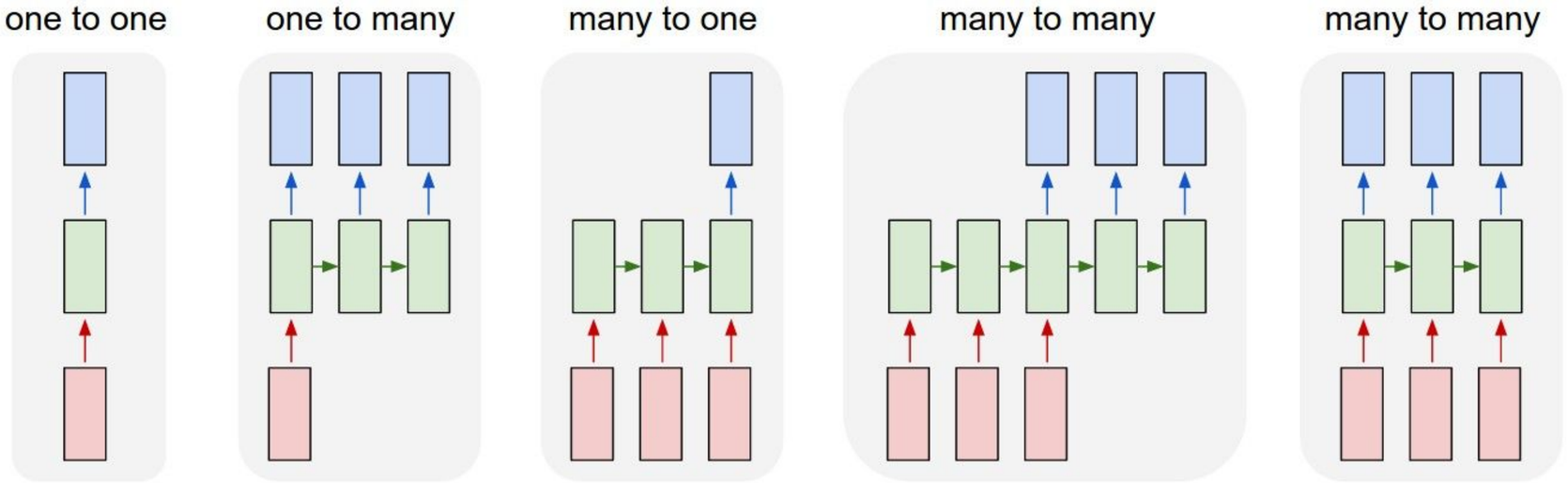
Activation functions

Inputs

$$g(W\phi(x_t) + Uh_{t-1})$$

Shared weights / parameters

And there's so much we can do with them!



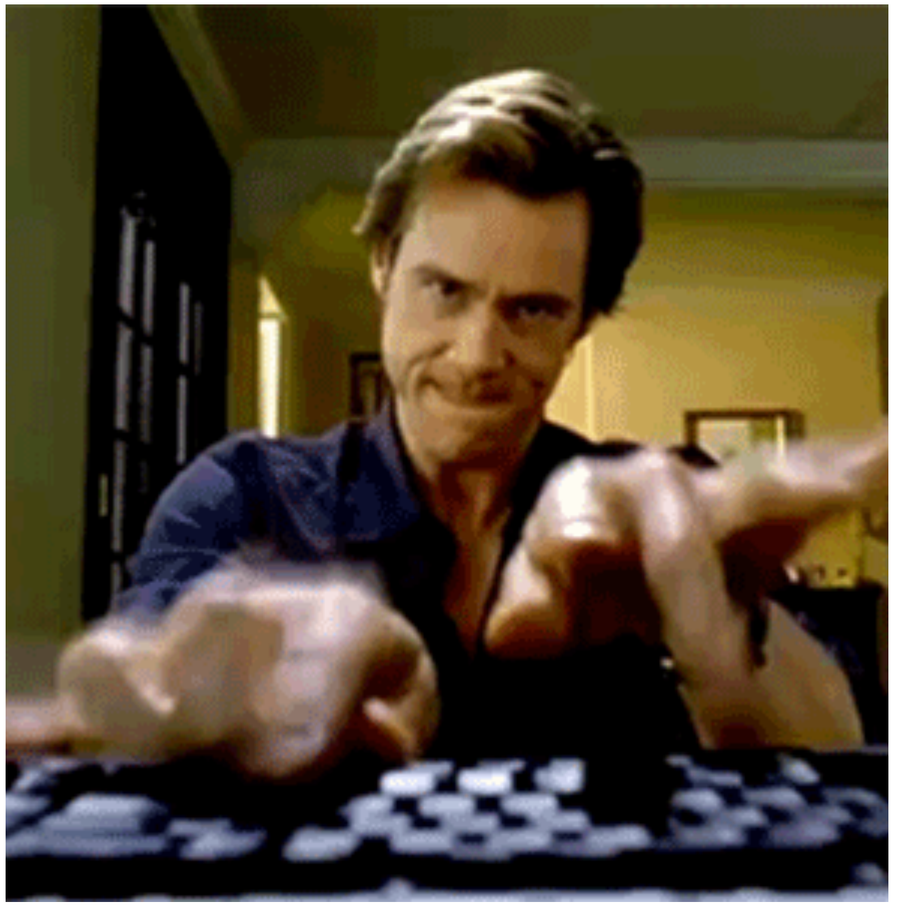
- Different structures to different types of data

- Change the structure of the how the hidden layer is computed

$$g\left(W\phi\begin{pmatrix}x_t\\h_t\end{pmatrix}\right)$$

- Image taken from: https://blog.floydhub.com/a-beginners-guide-on-recurrent-neural-networks-with-pytorch/

Let’s go to the code!!!

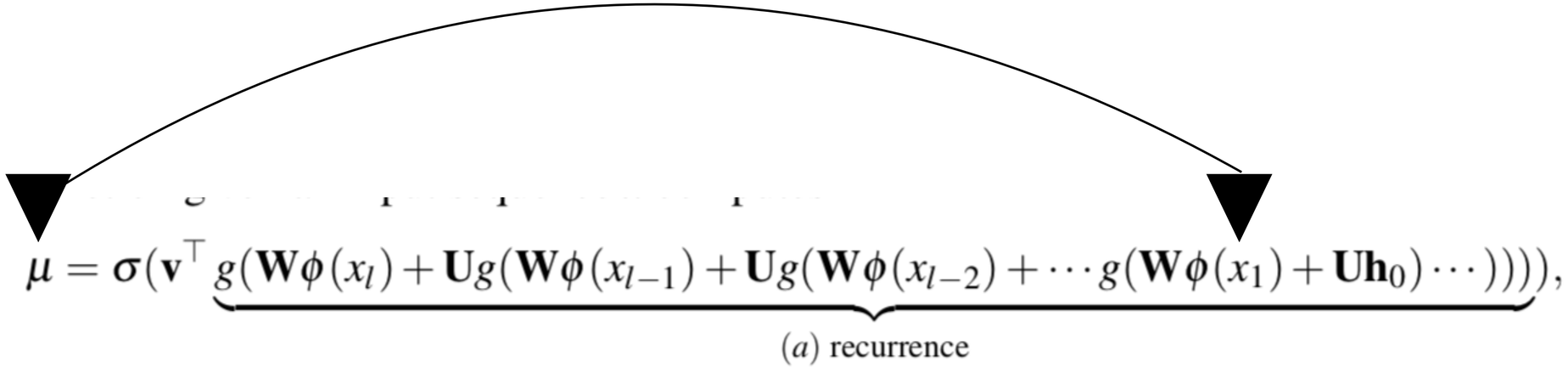


02

Gated Units

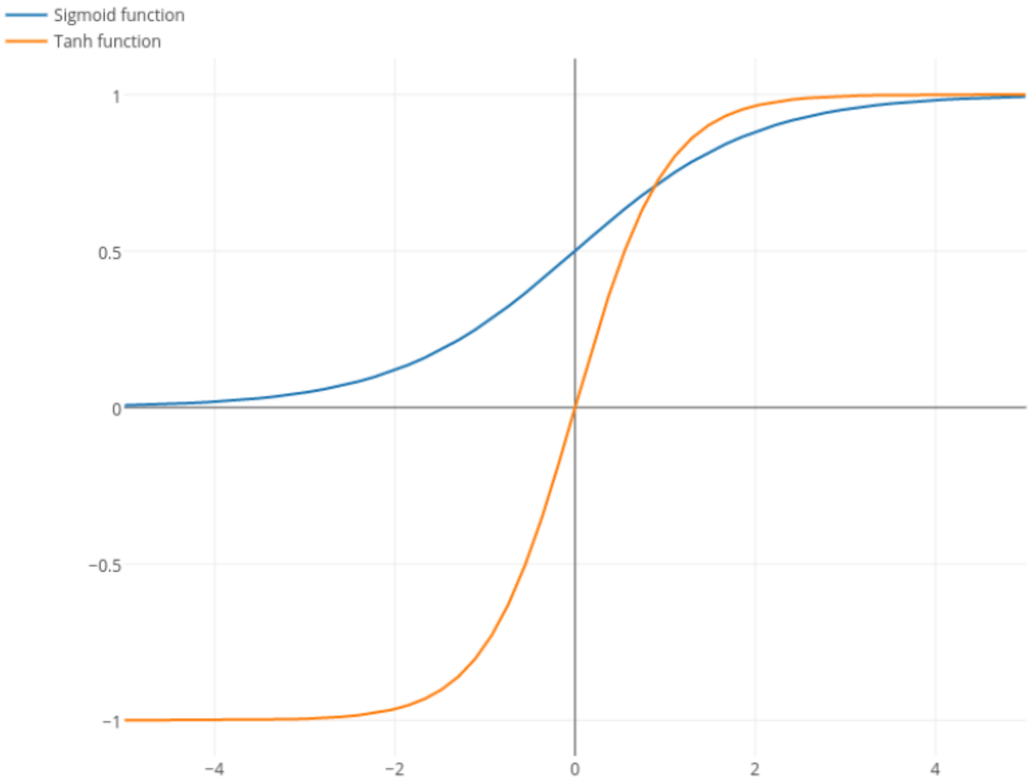
What can go wrong with this?

The information between the first input might not reach the output!



$$\mu = \sigma(\mathbf{v}^\top \underbrace{g(\mathbf{W}\boldsymbol{\phi}(x_l) + \mathbf{U}g(\mathbf{W}\boldsymbol{\phi}(x_{l-1}) + \mathbf{U}g(\mathbf{W}\boldsymbol{\phi}(x_{l-2}) + \cdots g(\mathbf{W}\boldsymbol{\phi}(x_1) + \mathbf{U}\mathbf{h}_0)\cdots)))}_{(a)\text{ recurrence}}),$$

Vanishing and exploding gradient!



Vanishing Gradient

- The activation functions (like a sigmoid, a tanh) will significantly reduce the value over time.
- This mean, when backpropagated, n very small derivates will end up multiplying each other.

Exploding Gradient

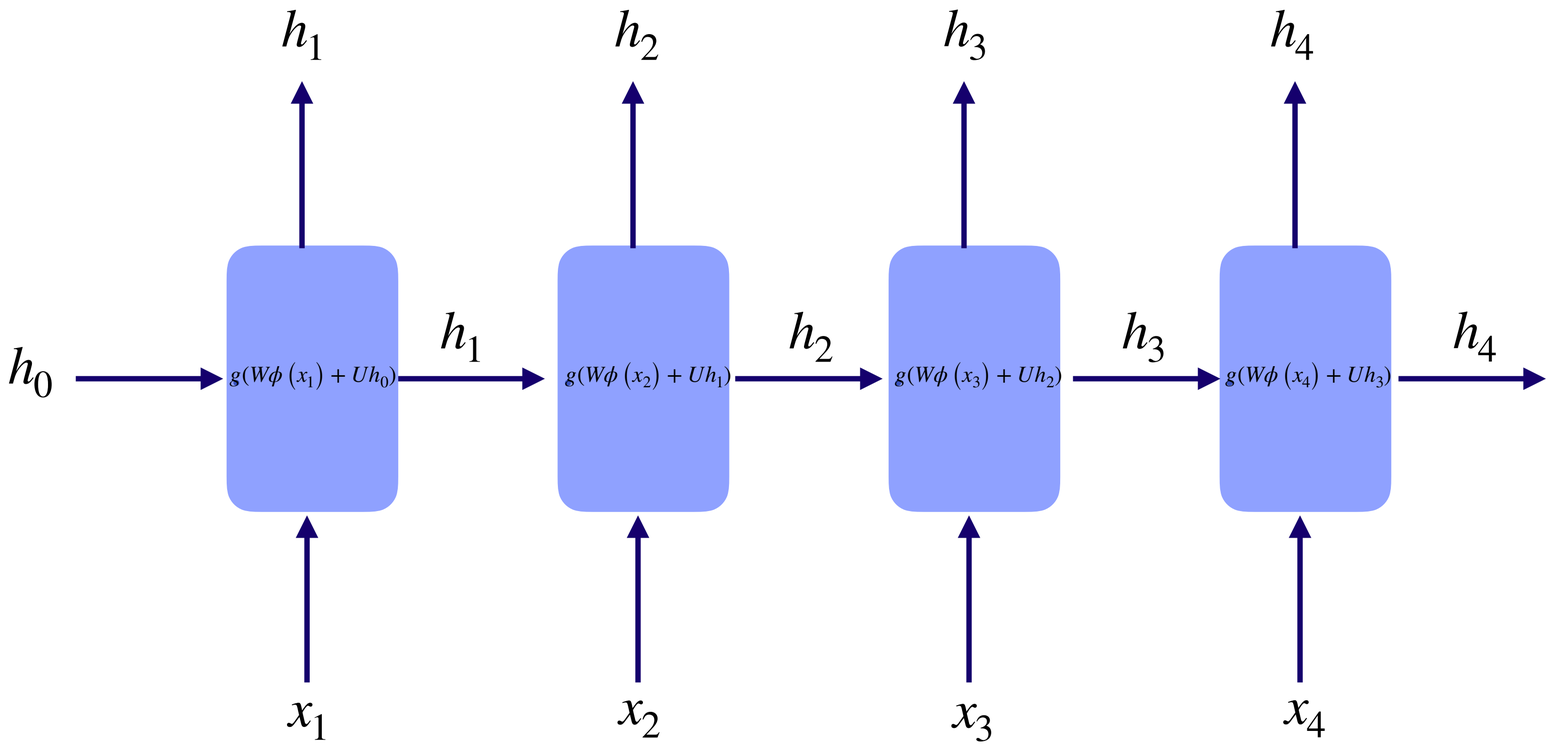
- Less of a problem, we can clip the gradient for example

The solution?

Gated UNITS!

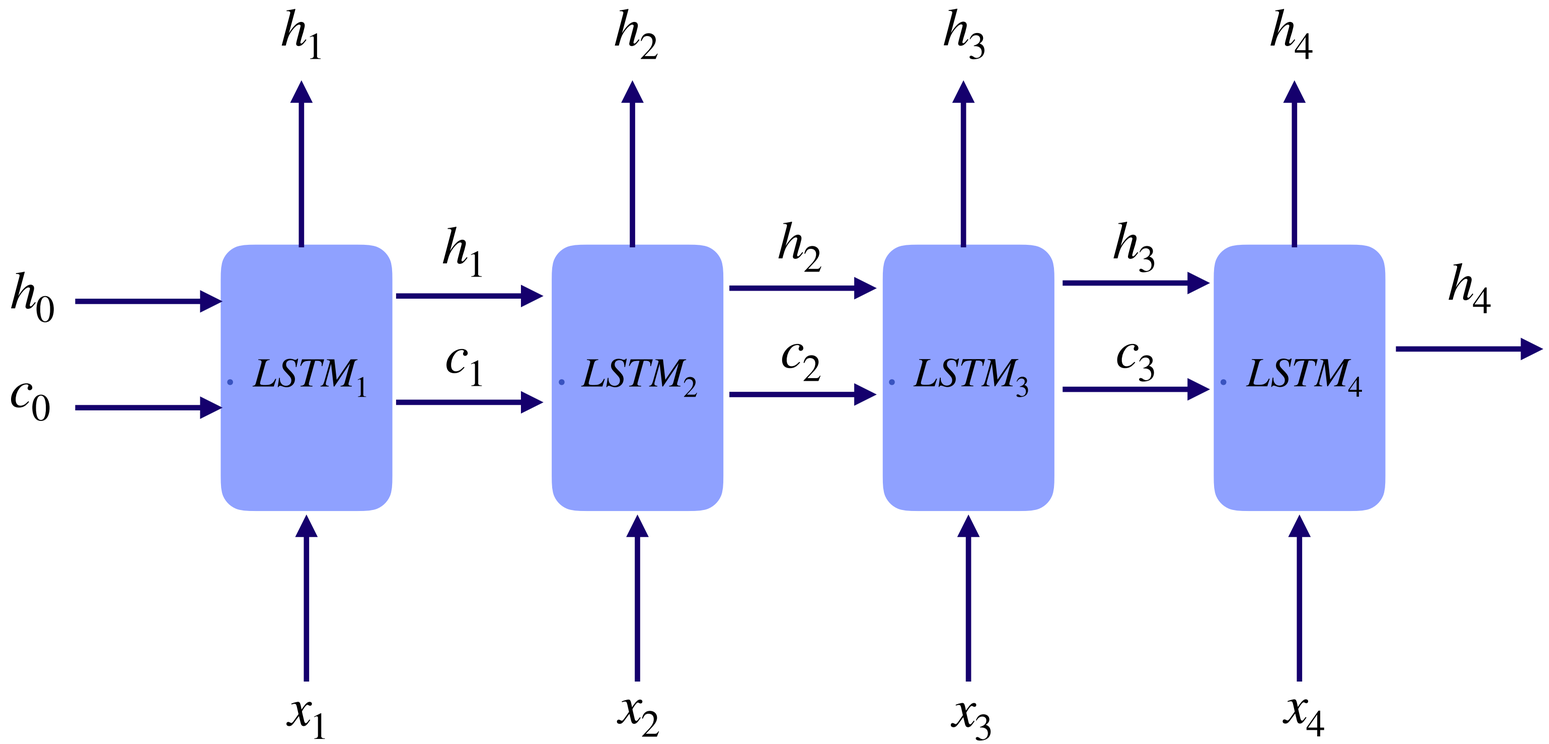
Long Short Term Memory (LSTMs)

Instead of this…



Long Short Term Memory (LSTMs)

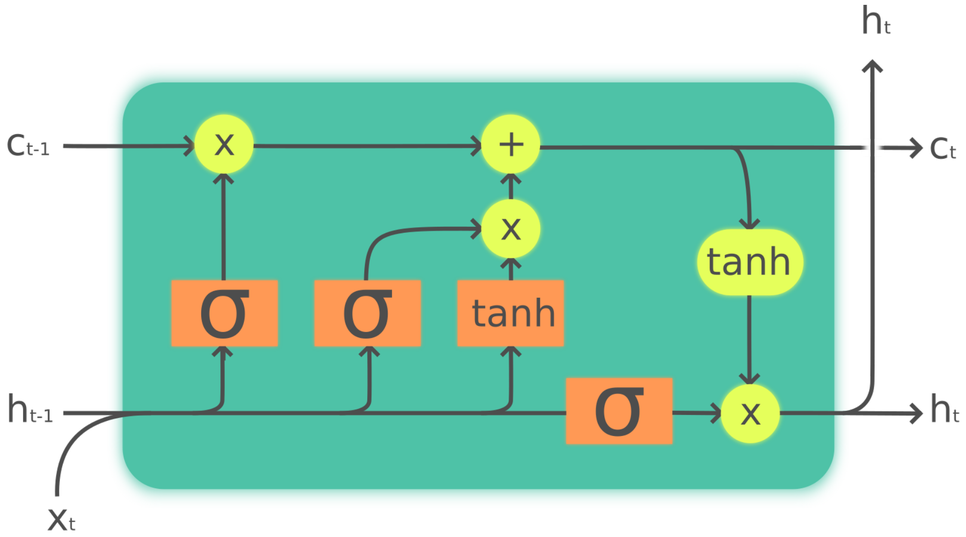
We'll have something that looks like this…



But… what's inside of them???

Long Short Term Memory (LSTMs)

$$i_t = \sigma\left(W_i\begin{pmatrix}x_t\\h_{t-1}\end{pmatrix} + b_i\right) \quad f_t = \sigma\left(W_f\begin{pmatrix}x_t\\h_{t-1}\end{pmatrix} + b_f\right) \quad o_t = \sigma\left(W_o\begin{pmatrix}x_t\\h_{t-1}\end{pmatrix} + b_o\right)$$



Three identical calculations, each with it's own weights and biases: Input, Forget and Output

Legend:

Layer

Pointwize op

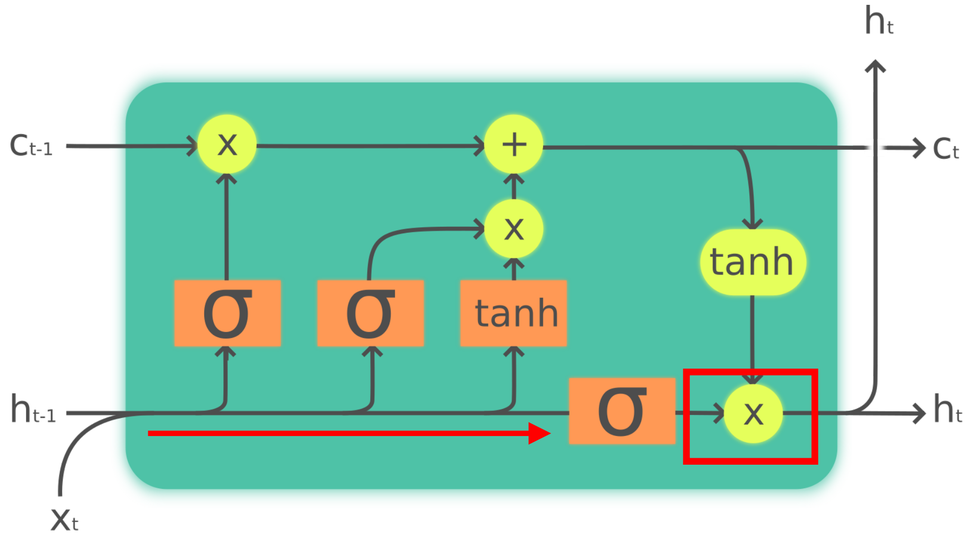
Copy

- Image taken from wikipedia. I'm OBVIOUSLY not that good at design :-P:https://en.wikipedia.org/wiki/Long_short-term_memory

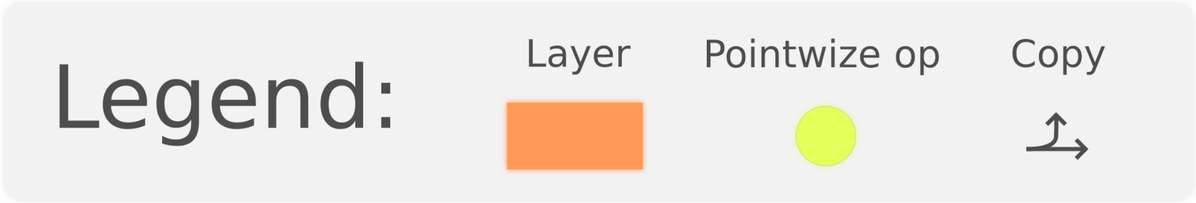
Long Short Term Memory (LSTMs)

$$i_t = \sigma\left(W_i\begin{pmatrix}x_t\\h_{t-1}\end{pmatrix} + b_i\right) \quad f_t = \sigma\left(W_f\begin{pmatrix}x_t\\h_{t-1}\end{pmatrix} + b_f\right) \quad o_t = \sigma\left(W_o\begin{pmatrix}x_t\\h_{t-1}\end{pmatrix} + b_o\right)$$

Output gate



Sigmoid [0,1] will control through a point-wise multiplication what goes out and what doesn't

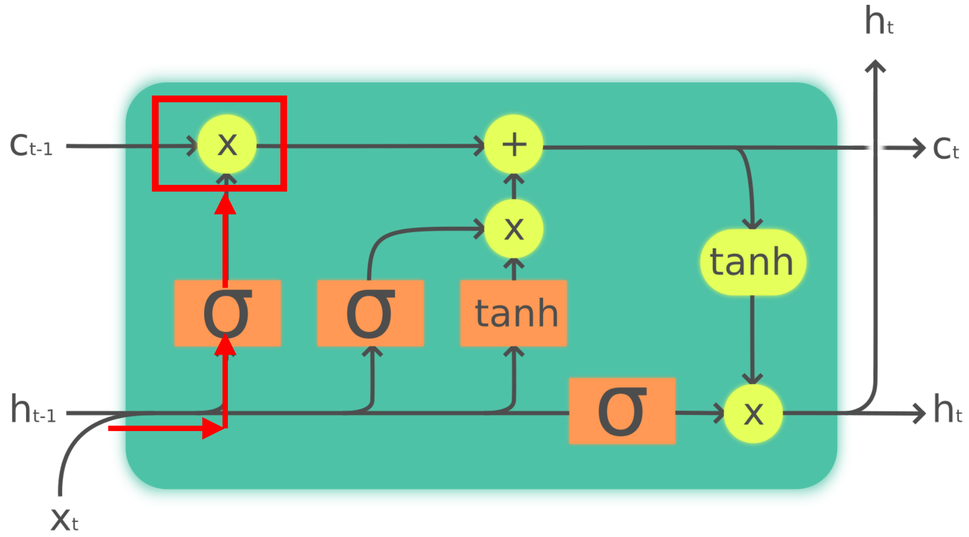


- Image taken from wikipedia. I'm OBVIOUSLY not that good at design :-P:https://en.wikipedia.org/wiki/Long_short-term_memory

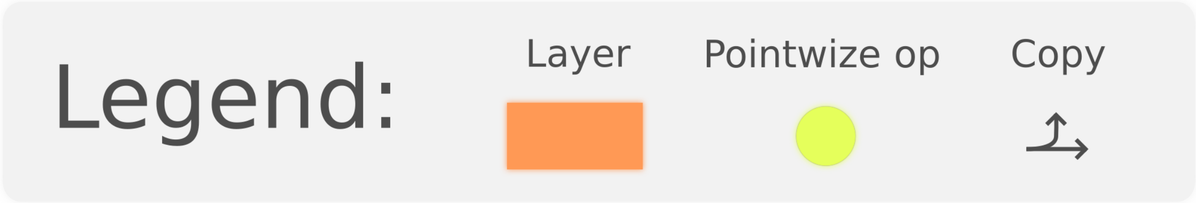
Long Short Term Memory (LSTMs)

$$i_t = \sigma\left(W_i\begin{pmatrix}x_t\\h_{t-1}\end{pmatrix} + b_i\right) \quad f_t = \sigma\left(W_f\begin{pmatrix}x_t\\h_{t-1}\end{pmatrix} + b_f\right) \quad o_t = \sigma\left(W_o\begin{pmatrix}x_t\\h_{t-1}\end{pmatrix} + b_o\right)$$

Forget gate



Sigmoid [0,1] will control through a point-wise multiplication what goes in from the old cell state

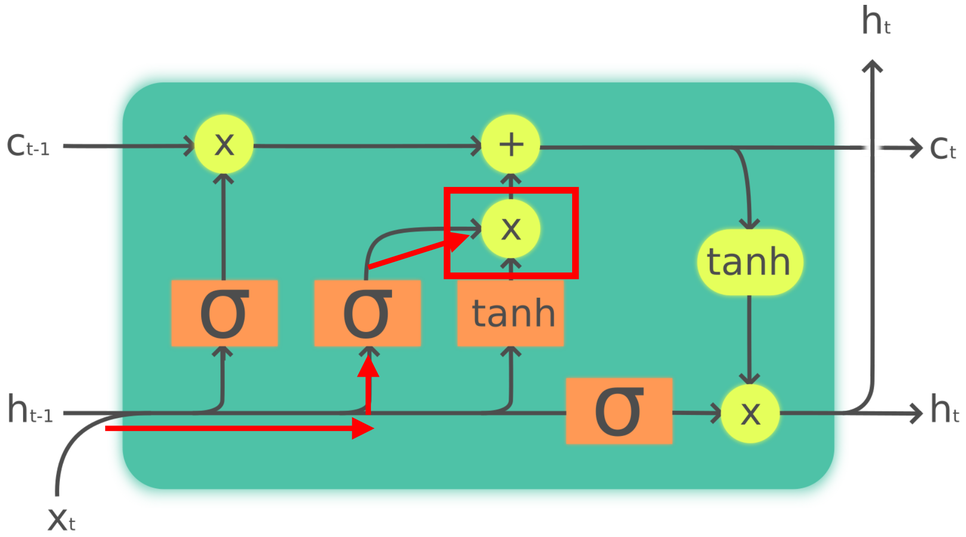


- Image taken from wikipedia. I'm OBVIOUSLY not that good at design :-P:https://en.wikipedia.org/wiki/Long_short-term_memory

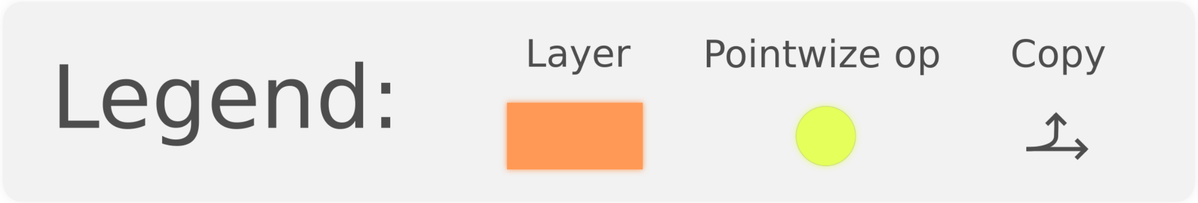
Long Short Term Memory (LSTMs)

$$i_t = \sigma\left(W_i\begin{pmatrix} x_t \\ h_{t-1}\end{pmatrix} + b_i\right) \quad f_t = \sigma\left(W_f\begin{pmatrix} x_t \\ h_{t-1}\end{pmatrix} + b_f\right) \quad o_t = \sigma\left(W_o\begin{pmatrix} x_t \\ h_{t-1}\end{pmatrix} + b_o\right)$$

Input gate



Sigmoid [0,1] will control through a point-wise multiplication what goes in from our proposed new cell state



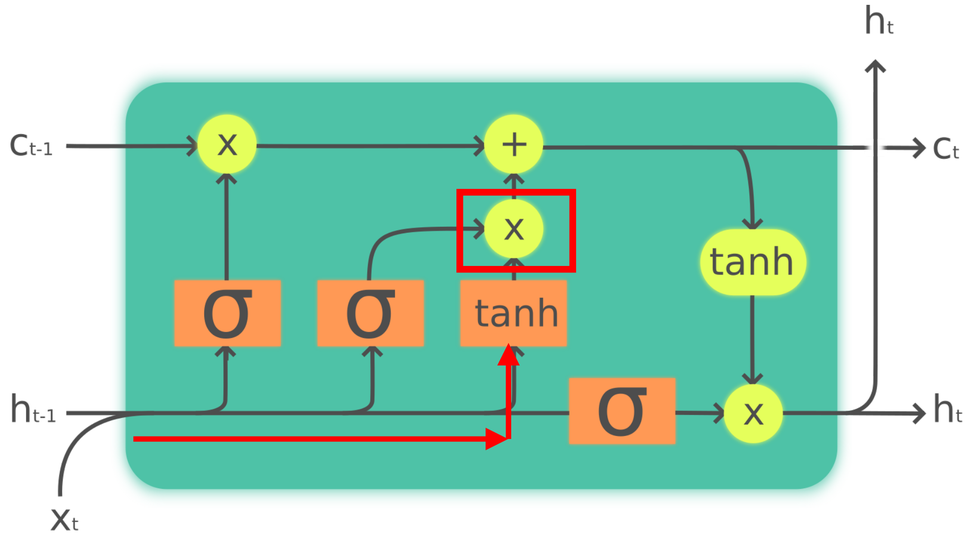
- Image taken from wikipedia. I'm OBVIOUSLY not that good at design :-P:https://en.wikipedia.org/wiki/Long_short-term_memory

Long Short Term Memory (LSTMs)

$$i_t = \sigma\left(W_i\begin{pmatrix} x_t \\ h_{t-1}\end{pmatrix} + b_i\right) \quad f_t = \sigma\left(W_f\begin{pmatrix} x_t \\ h_{t-1}\end{pmatrix} + b_f\right) \quad o_t = \sigma\left(W_o\begin{pmatrix} x_t \\ h_{t-1}\end{pmatrix} + b_o\right)$$

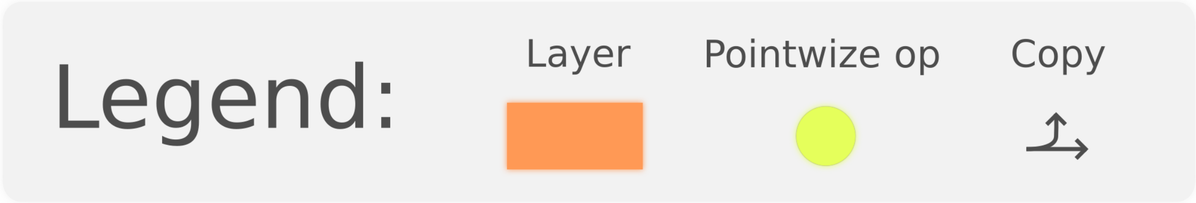
$$\widetilde{c}_t = \tanh\left(W_c\begin{pmatrix} x_t \\ h_{t-1}\end{pmatrix} + b_c\right)$$

→ Proposed cell state



The input gate will control the flow of information

we're proposing to pass to the next layer



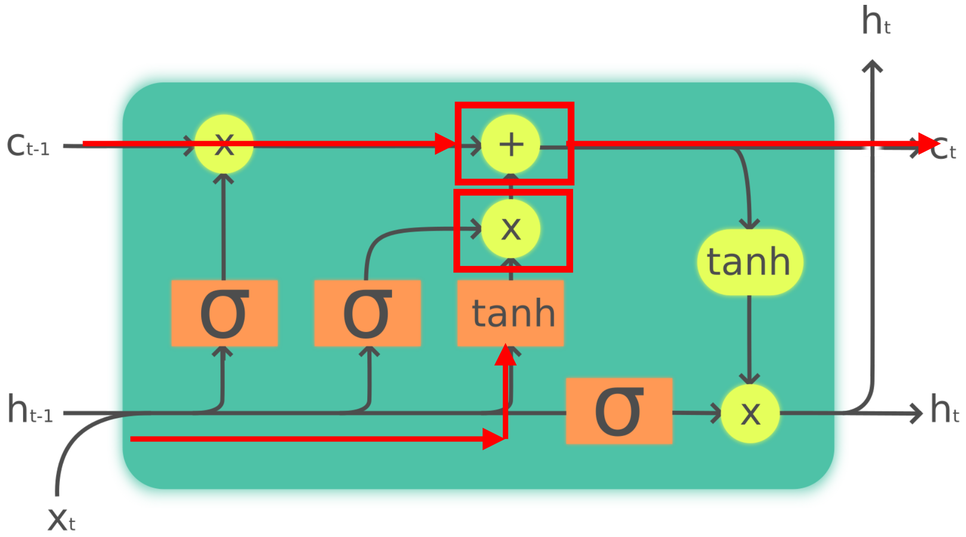
- Image taken from wikipedia. I'm OBVIOUSLY not that good at design :-P:https://en.wikipedia.org/wiki/Long_short-term_memory

Long Short Term Memory (LSTMs)

$$i_t = \sigma\left(W_i\begin{pmatrix} x_t \\ h_{t-1}\end{pmatrix} + b_i\right) \quad f_t = \sigma\left(W_f\begin{pmatrix} x_t \\ h_{t-1}\end{pmatrix} + b_f\right) \quad o_t = \sigma\left(W_o\begin{pmatrix} x_t \\ h_{t-1}\end{pmatrix} + b_o\right)$$

$$\widetilde{c}_t = \tanh\left(W_c\begin{pmatrix} x_t \\ h_{t-1}\end{pmatrix} + b_c\right)$$

$$c_t = i_t \odot \widetilde{c}_t + f_t \odot c_{t-1}$$ → New Cell State



So our NEW cell state, will be what we have decided to keep from the past, and what we have decided to keep from the present

Legend: Layer Pointwize op Copy

- Image taken from wikipedia. I'm OBVIOUSLY not that good at design :-P:https://en.wikipedia.org/wiki/Long_short-term_memory

Long Short Term Memory (LSTMs)

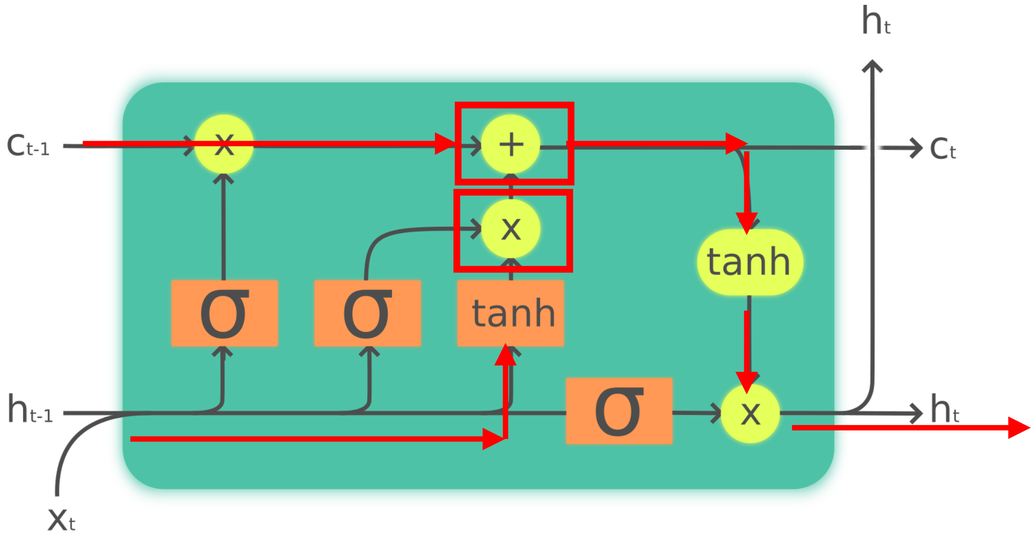
$$i_t = \sigma\left(W_i\begin{pmatrix} x_t \\ h_{t-1}\end{pmatrix} + b_i\right) \quad f_t = \sigma\left(W_f\begin{pmatrix} x_t \\ h_{t-1}\end{pmatrix} + b_f\right) \quad o_t = \sigma\left(W_o\begin{pmatrix} x_t \\ h_{t-1}\end{pmatrix} + b_o\right)$$

$$\widetilde{c}_t = \tanh\left(W_c\begin{pmatrix} x_t \\ h_{t-1}\end{pmatrix} + b_c\right)$$

$$c_t = i_t \odot \widetilde{c}_t + f_t \odot c_{t-1}$$

$$h_t = o_t \odot tanh\left(c_t\right)$$

→ New Hidden State



Legend: Layer Pointwize op Copy

And, this cell state, after a non-linearity, will pass

through the output gat that will decide it's fate

- Image taken from wikipedia. I'm OBVIOUSLY not that good at design :-P:https://en.wikipedia.org/wiki/Long_short-term_memory

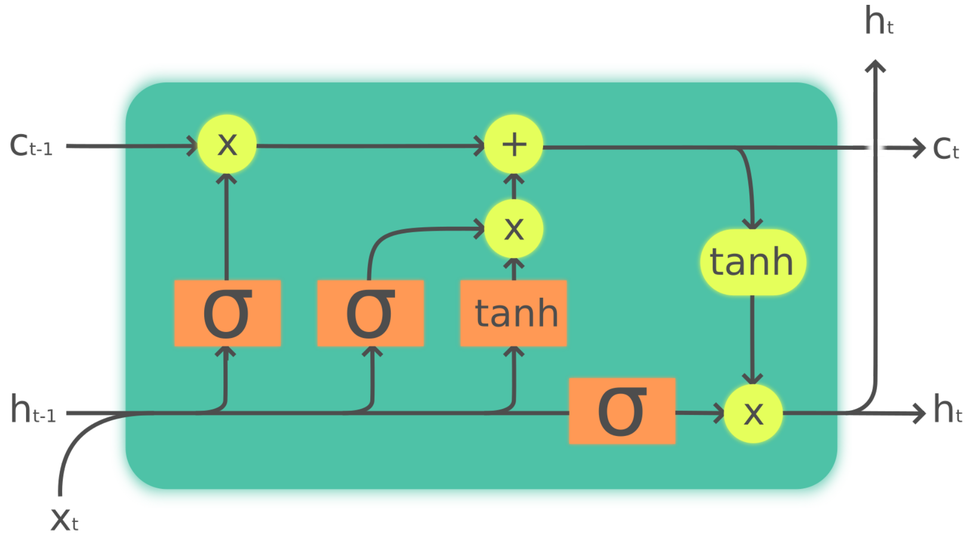
Long Short Term Memory (LSTMs)

$$i_t = \sigma\left(W_i\begin{pmatrix} x_t \\ h_{t-1}\end{pmatrix} + b_i\right) \quad f_t = \sigma\left(W_f\begin{pmatrix} x_t \\ h_{t-1}\end{pmatrix} + b_f\right) \quad o_t = \sigma\left(W_o\begin{pmatrix} x_t \\ h_{t-1}\end{pmatrix} + b_o\right)$$

$$\widetilde{c}_t = \tanh\left(W_c\begin{pmatrix} x_t \\ h_{t-1}\end{pmatrix} + b_c\right)$$

$$c_t = i_t \odot \widetilde{c}_t + f_t \odot c_{t-1}$$

$$h_t = o_t \odot tanh\left(c_t\right)$$



Some Intuition?

- The cell state carries all the flow of information, a control between the past and the present.
- The the hidden state updates this flow of information with the newest of the information that comes in.

Legend: Layer Pointwize op Copy

- Image taken from wikipedia. I'm OBVIOUSLY not that good at design :-P:https://en.wikipedia.org/wiki/Long_short-term_memory

Long Short Term Memory (LSTMs)

$$i_t = \sigma\left(W_i\begin{pmatrix}x_t\\h_{t-1}\end{pmatrix} + b_i\right) \quad f_t = \sigma\left(W_f\begin{pmatrix}x_t\\h_{t-1}\end{pmatrix} + b_f\right) \quad o_t = \sigma\left(W_o\begin{pmatrix}x_t\\h_{t-1}\end{pmatrix} + b_o\right)$$

$$\widetilde{c}_t = \tanh\left(W_c\begin{pmatrix}x_t\\h_{t-1}\end{pmatrix} + b_c\right)$$

$$c_t = i_t \odot \widetilde{c}_t + f_t \odot c_{t-1}$$

$$h_t = o_t \odot tanh\left(c_t\right)$$

In pytorch

- In pytorch, the hidden layer and the input are not concatenated, but each have it's own independent weights and bias

$$\begin{aligned}
i_t &= \sigma(W_{ii}x_t + b_{ii} + W_{hi}h_{(t-1)} + b_{hi}) \\
f_t &= \sigma(W_{if}x_t + b_{if} + W_{hf}h_{(t-1)} + b_{hf}) \\
g_t &= \tanh(W_{ig}x_t + b_{ig} + W_{hg}h_{(t-1)} + b_{hg}) \\
o_t &= \sigma(W_{io}x_t + b_{io} + W_{ho}h_{(t-1)} + b_{ho}) \\
c_t &= f_t * c_{(t-1)} + i_t * g_t \\
h_t &= o_t * \tanh(c_t)
\end{aligned}$$

- Image taken from wikipedia. I'm OBVIOUSLY not that good at design :-P:https://en.wikipedia.org/wiki/Long_short-term_memory

Gated Recurrent Units (GRUs)

The idea is very similar to LSTMs…

$$i_t = \sigma\left(W_i\begin{pmatrix}x_t\\h_{t-1}\end{pmatrix} + b_i\right) \quad f_t = \sigma\left(W_f\begin{pmatrix}x_t\\h_{t-1}\end{pmatrix} + b_f\right) \quad o_t = \sigma\left(W_o\begin{pmatrix}x_t\\h_{t-1}\end{pmatrix} + b_o\right)$$

$$\tilde{c}_t = \tanh\left(W_c\begin{pmatrix}x_t\\h_{t-1}\end{pmatrix} + b_c\right)$$

$$c_t = i_t \odot \tilde{c}_t + f_t \odot c_{t-1}$$

$$h_t = o_t \odot tanh\left(c_t\right)$$

But in GRUs…

$$z_t = \sigma\left(W_z\begin{pmatrix}x_t\\h_{t-1}\end{pmatrix} + b_z\right)$$ → Update Gate

$$r_t = \sigma\left(W_r\begin{pmatrix}x_t\\h_{t-1}\end{pmatrix} + b_r\right)$$ → Reset Gate

- Image taken from wikipedia. I'm OBVIOUSLY not that good at design :-P:https://en.wikipedia.org/wiki/Long_short-term_memory

Gated Recurrent Units (GRUs)

The idea is very similar to LSTMs…

$$i_t = \sigma\left(W_i\begin{pmatrix}x_t\\h_{t-1}\end{pmatrix}+b_i\right) \quad f_t = \sigma\left(W_f\begin{pmatrix}x_t\\h_{t-1}\end{pmatrix}+b_f\right) \quad o_t = \sigma\left(W_o\begin{pmatrix}x_t\\h_{t-1}\end{pmatrix}+b_o\right)$$

$$\tilde{c}_t = \tanh\left(W_c\begin{pmatrix}x_t\\h_{t-1}\end{pmatrix}+b_c\right)$$

$$c_t = i_t \odot \tilde{c}_t + f_t \odot c_{t-1}$$

$$h_t = o_t \odot tanh\left(c_t\right)$$

But in GRUs…

$$z_t = \sigma\left(W_z\begin{pmatrix}x_t\\h_{t-1}\end{pmatrix}+b_z\right)$$

$$r_t = \sigma\left(W_r\begin{pmatrix}x_t\\h_{t-1}\end{pmatrix}+b_r\right)$$

$$n_t = \tanh\left(W_n x_t + U_n\left(h_{t-1}\odot r_t\right)+b_n\right)$$

We propose a new input filtering out info from the past

- Image taken from wikipedia. I'm OBVIOUSLY not that good at design :-P:https://en.wikipedia.org/wiki/Long_short-term_memory

Gated Recurrent Units (GRUs)

The idea is very similar to LSTMs…

$$i_t = \sigma\left(W_i\begin{pmatrix}x_t\\h_{t-1}\end{pmatrix}+b_i\right) \quad f_t = \sigma\left(W_f\begin{pmatrix}x_t\\h_{t-1}\end{pmatrix}+b_f\right) \quad o_t = \sigma\left(W_o\begin{pmatrix}x_t\\h_{t-1}\end{pmatrix}+b_o\right)$$

$$\tilde{c}_t = \tanh\left(W_c\begin{pmatrix}x_t\\h_{t-1}\end{pmatrix}+b_c\right)$$

$$c_t = i_t \odot \tilde{c}_t + f_t \odot c_{t-1}$$

$$h_t = o_t \odot tanh\left(c_t\right)$$

But in GRUs…

$$z_t = \sigma\left(W_z\begin{pmatrix}x_t\\h_{t-1}\end{pmatrix}+b_z\right)$$

$$r_t = \sigma\left(W_r\begin{pmatrix}x_t\\h_{t-1}\end{pmatrix}+b_r\right)$$

$$n_t = \tanh\left(W_n x_t + U_n\left(h_{t-1}\odot r_t\right)+b_n\right)$$

$$h_t = \left(1-z_t\right)\odot n_t + z_t \odot h_{t-1}$$

New hidden layer is an “average” of our proposal and the last hidden, controlled by update gate

- Image taken from wikipedia. I’m OBVIOUSLY not that good at design :-P:https://en.wikipedia.org/wiki/Long_short-term_memory

Gated Recurrent Units (GRUs)

In pytorch

$$
\begin{aligned}
r_t &= \sigma(W_{ir}x_t + b_{ir} + W_{hr}h_{(t-1)} + b_{hr}) \\
z_t &= \sigma(W_{iz}x_t + b_{iz} + W_{hz}h_{(t-1)} + b_{hz}) \\
n_t &= \tanh(W_{in}x_t + b_{in} + r_t * (W_{hn}h_{(t-1)} + b_{hn})) \\
h_t &= (1 - z_t) * n_t + z_t * h_{(t-1)}
\end{aligned}
$$

- Same as the LSTM, the input and the hidden have their own set of weights and biases in the gates.

But in GRUs…

$$z_t = \sigma\left(W_z\begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix} + b_z\right)$$

$$r_t = \sigma\left(W_r\begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix} + b_r\right)$$

$$n_t = \tanh\left(W_n x_t + U_n\left(h_{t-1} \odot r_t\right) + b_n\right)$$

$$h_t = \left(1 - z_t\right) \odot n_t + z_t \odot h_{t-1}$$

- Image taken from wikipedia. I'm OBVIOUSLY not that good at design :-P:https://en.wikipedia.org/wiki/Long_short-term_memory

LSTMs VS GRUs

LSTMs

$$i_t = \sigma\left(W_i\begin{pmatrix} x_t \\ h_{t-1}\end{pmatrix} + b_i\right) \quad f_t = \sigma\left(W_f\begin{pmatrix} x_t \\ h_{t-1}\end{pmatrix} + b_f\right) \quad o_t = \sigma\left(W_o\begin{pmatrix} x_t \\ h_{t-1}\end{pmatrix} + b_o\right)$$

$$\tilde{c}_t = \tanh\left(W_c\begin{pmatrix} x_t \\ h_{t-1}\end{pmatrix} + b_c\right)$$

$$c_t = i_t \odot \tilde{c}_t + f_t \odot c_{t-1}$$

$$h_t = o_t \odot tanh\left(c_t\right)$$

- GRUs are more lightweight.
- LSTM's carry a cell state, which makes them "less forgetful"
- BUT, in practice, it tends to be a very task specific decision

GRUs

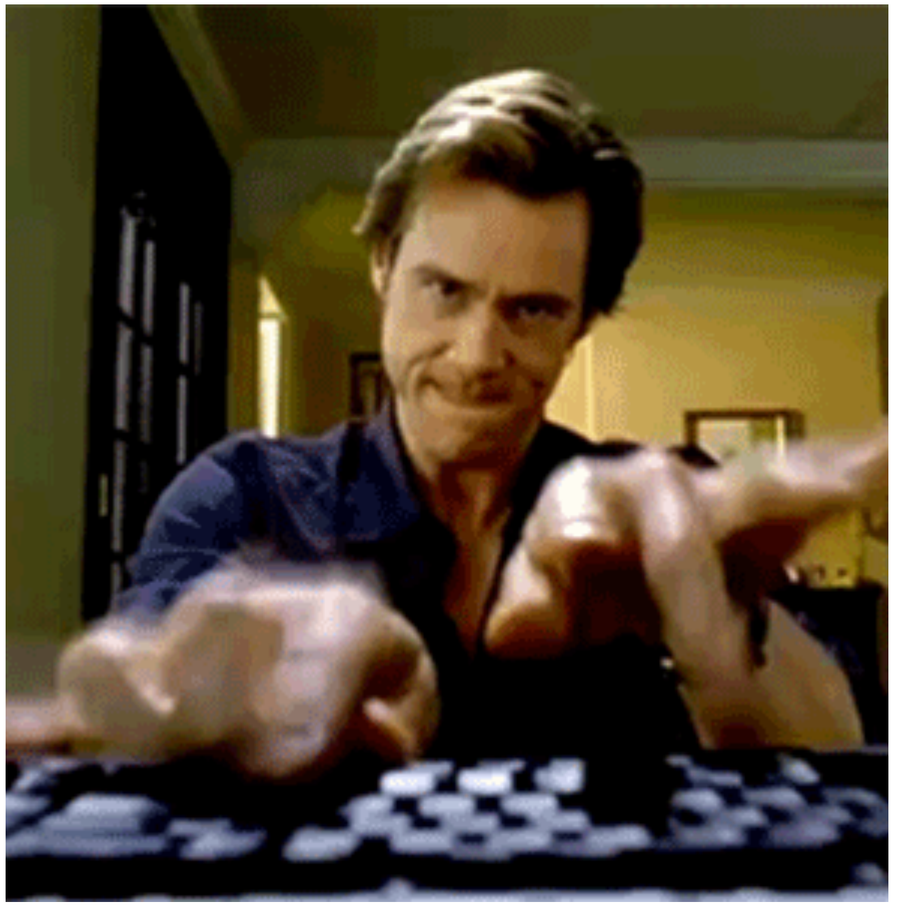
$$z_t = \sigma\left(W_z\begin{pmatrix} x_t \\ h_{t-1}\end{pmatrix} + b_z\right) \quad r_t = \sigma\left(W_r\begin{pmatrix} x_t \\ h_{t-1}\end{pmatrix} + b_r\right)$$

$$n_t = \tanh\left(W_n x_t + U_n\left(h_{t-1} \odot r_t\right) + b_n\right)$$

$$h_t = \left(1 - z_t\right) \odot n_t + z_t \odot h_{t-1}$$

- Image taken from wikipedia. I'm OBVIOUSLY not that good at design :-P:https://en.wikipedia.org/wiki/Long_short-term_memory

Back to Code!!!



03

Resources

Resources and Pointers

- IMDB data from http://ai.stanford.edu/~amaas/data/sentiment/
- Cho's notes on NLP: https://arxiv.org/abs/1511.07916
- RNN image from: http://karpathy.github.io/2015/05/21/rnn-effectiveness/
- Some code for pre-process is from https://github.com/nyu-mll/DS-GA-1011-Fall2017 (I strongly recommend you diving into these labs and code!)
- FULLY recommend to go through this blog, explaining LSTMs >> https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21