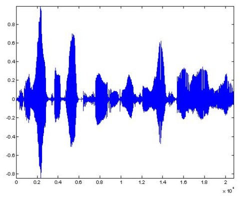
Recurrent Neural Networks

Girish Varma
IIIT Hyderabad
http://bit.ly/2ubR4iC

Sequence to Sequence Problems

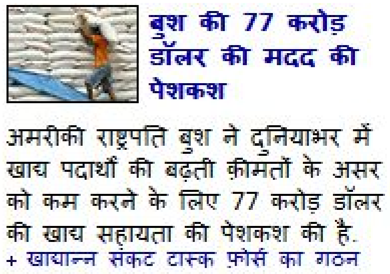
Convert $x_1,x_2,...,x_s$ to $y_1,y_2,...,y_t$

Speech to Text: speech signal x(i) to text y(i)



$\longrightarrow$ Hello

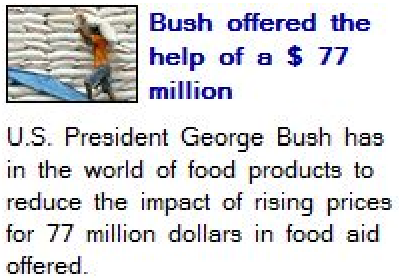
Language Translation: text in language 1 x(i) to text in language 2 y(i)



बुश की 77 करोड़ डॉलर की मदद की पेशकश

अमरीकी राष्ट्रपति बुश ने दुनियाभर में खाद्य पदार्थों की बढ़ती क़ीमतों के असर को कम करने के लिए 77 करोड़ डॉलर की खाद्य सहायता की पेशकश की है.

+ खाद्यान्न संकट टास्क फ़ोर्स का गठन



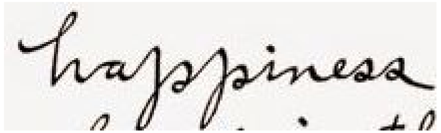
Bush offered the help of a $ 77 million

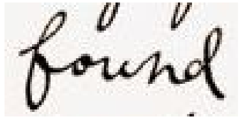
U.S. President George Bush has in the world of food products to reduce the impact of rising prices for 77 million dollars in food aid offered.

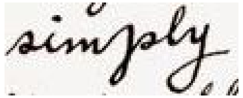
Sequence to Sequence Problems

Convert $x_1,x_2,\ldots,x_s$ to $y_1,y_2,\ldots,y_t$

OCR : Convert these images to unicode strings

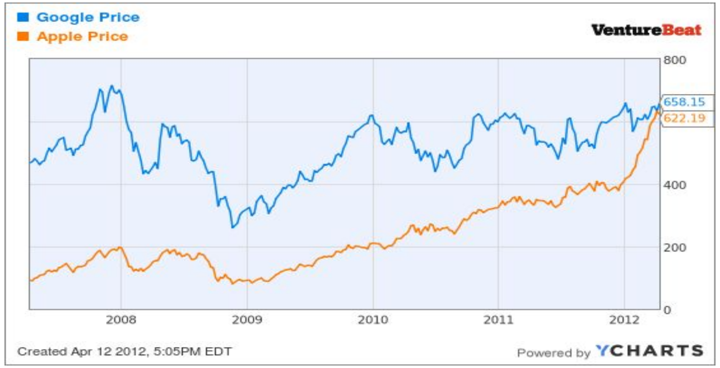






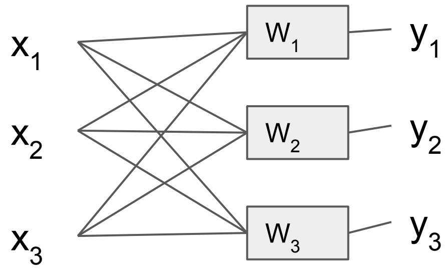
Time series data : Given a sequence predict the next element

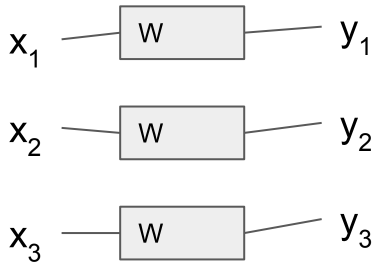
- Predict stock prices
- Weather prediction



First Attempt at a Model

Convert x_1,x_2,x_3 to y_1,y_2,y_3





1. Variable length sequences.
2. Each network will have too many inputs.
3. Using a different Network each time, results in explosion of parameters.

Need information about previous inputs/outputs for predicting the next.

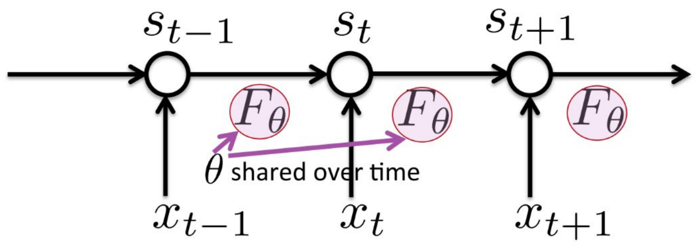
Recurrent Neural Networks

Convert $x_1, x_2, \dots, x_s$ to $o_1, o_2, \dots, o_t$

Summarise input sequence in a fixed length state vector with recursive updates.

$s_t = F_{\boldsymbol{\theta}}(s_{t-1}, x_t)$

$o_t = G_{\boldsymbol{\theta}'}(s_t)$



Generalizes to lengths not seen during training.

From Bengio's slides :
https://drive.google.com/open?id=0ByUKRdiCDK7-LXZkM3hVSzFGTkE

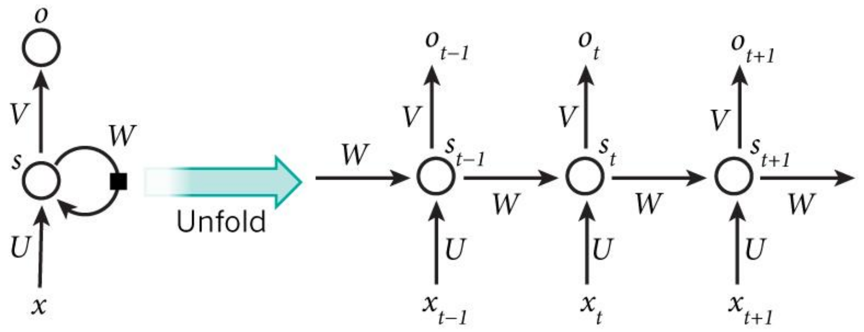
RNNs

$s_t = F_{U,W,b}(s_{t-1}, x_t)$

$\quad = \tanh(Ux_t + Ws_{t-1} + b)$

$o_t = G_V(s_t)$

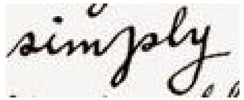
$\quad = Vs_t$

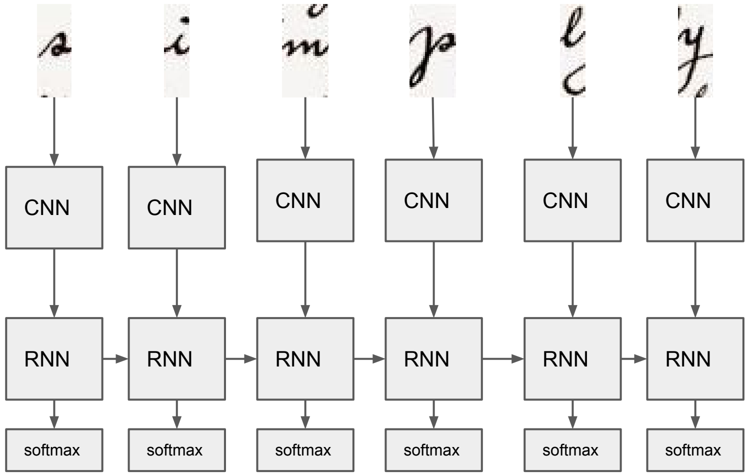


Source: Nature

Case Study : Scene Text with char segmentation

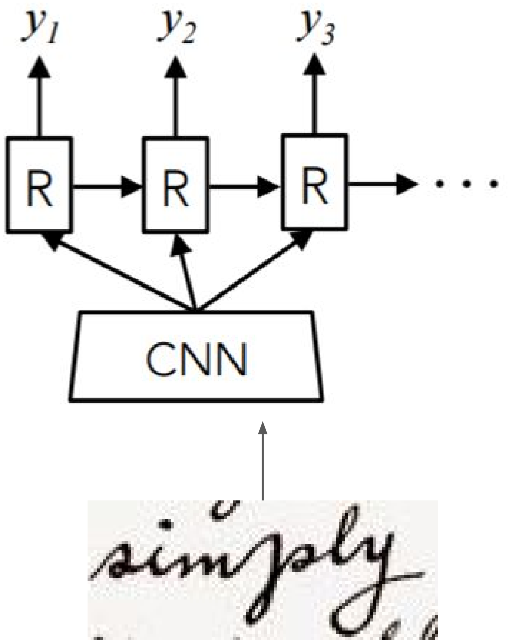
1. Segment the image into characters using some algorithm.
2. Extract CNN features
3. Pass CNN feature sequence through RNN
4. Pass RNN output though softmax to get alphabet probabilities
5. Loss function: $\sum_i \mathrm{Error}(y_i, y_i^{correct})$





Case Study : Scenetxt without char segmentation

1. CNN for image feature extraction.
2. Duplicate CNN features into a sequence of Maxlen.
3. RNN followed by softmax gives probabilities of alphabets of length Maxlen.



Backpropagation in RNNs

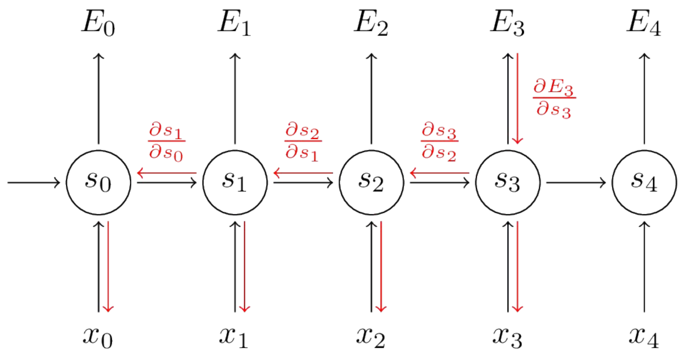
$h_t = Wf(h_{t-1}) + Vx_t$

- Total error is the sum of each error at time steps t

$$\frac{\partial E}{\partial W} = \sum_{t=1}^{T} \frac{\partial E_t}{\partial W}$$

- Hardcore chain rule application:

$$\frac{\partial E_t}{\partial W} = \sum_{k=1}^{t} \frac{\partial E_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial W}$$



E_0
E_1
E_2
E_3
E_4
$\frac{\partial E_3}{\partial s_3}$
$\frac{\partial s_1}{\partial s_0}$
$\frac{\partial s_2}{\partial s_1}$
$\frac{\partial s_3}{\partial s_2}$
s_0
s_1
s_2
s_3
s_4
x_0
x_1
x_2
x_3
x_4

Slides from
http://web.stanford.edu/class/cs224n/syllabus.html

The vanishing gradient problem - Details

- Similar to backprop but less efficient formulation
- Useful for analysis we'll look at:

$$\frac{\partial E_t}{\partial W} = \sum_{k=1}^{t} \frac{\partial E_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \boxed{\frac{\partial h_t}{\partial h_k}} \frac{\partial h_k}{\partial W}$$

- Remember: $h_t = Wf(h_{t-1}) + Vx_t$
- More chain rule, remember:

$$\frac{\partial h_t}{\partial h_k} = \prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}}$$

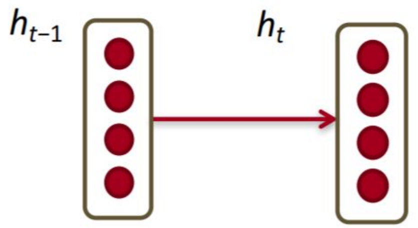
- Each partial is a Jacobian:

$$\frac{d\mathbf{f}}{d\mathbf{x}} = \left[\frac{\partial \mathbf{f}}{\partial x_1} \cdots \frac{\partial \mathbf{f}}{\partial x_n} \right] = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$$

Slides from
http://web.stanford.edu/class/cs224n/syllabus.html

The vanishing gradient problem - Details

- From previous slide: $\frac{\partial h_t}{\partial h_k} = \prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}}$



- Remember: $h_t = Wf(h_{t-1}) + Vx_t$

- To compute Jacobian, derive each element of matrix: $\frac{\partial h_{j,m}}{\partial h_{j-1,n}}$

$$\frac{\partial h_t}{\partial h_k} = \prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}} = \prod_{j=k+1}^{t} W^T \text{diag}[f'(h_{j-1})]$$

- Where: $\text{diag}(z) = \begin{pmatrix} z_1 & & & & \\ & z_2 & & 0 & \\ & & \ddots & & \\ & 0 & & z_{n-1} & \\ & & & & z_n \end{pmatrix}$

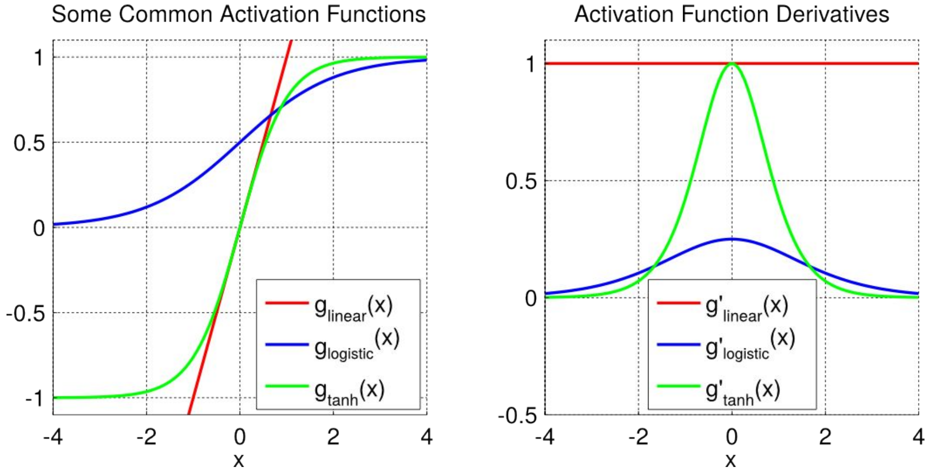
Check at home that you understand the diag matrix formulation

Slides from http://web.stanford.edu/class/cs224n/syllabus.html

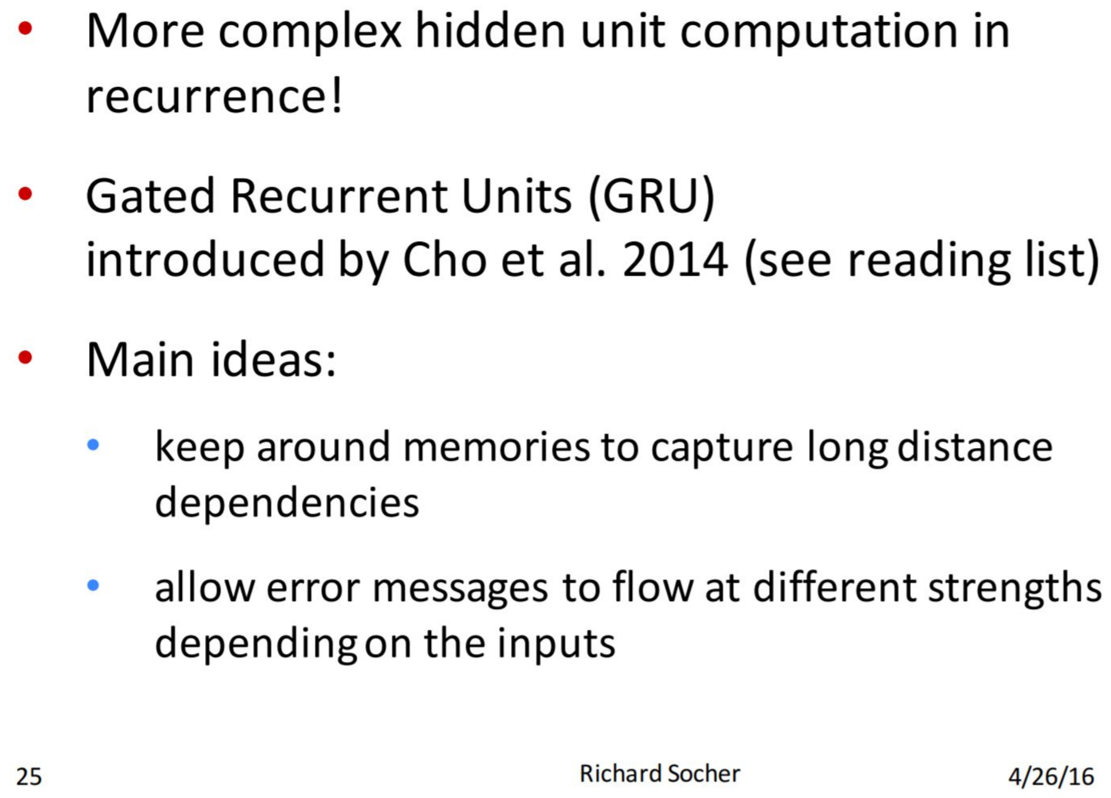
Vanishing gradients

- If t - k = d, gradient will have a term of W^d.
- If $||W||$ is small, all long range dependencies will be lost.
- Tanh is used as activation function to prevent this problem from worsening

$$s_t = \tanh(Ux_t + Ws_{t-1} + b)$$



Gated Recurrent Units : GRUs



- More complex hidden unit computation in recurrence!
- Gated Recurrent Units (GRU) introduced by Cho et al. 2014 (see reading list)
- Main ideas:
  - keep around memories to capture long distance dependencies
  - allow error messages to flow at different strengths depending on the inputs

25 Richard Socher 4/26/16

GRUs

- Standard RNN computes hidden layer at next time step directly: $h_t = f\left(W^{(hh)}h_{t-1} + W^{(hx)}x_t\right)$
- GRU first computes an update **gate** (another layer) based on current input word vector and hidden state

$$z_t = \sigma\left(W^{(z)}x_t + U^{(z)}h_{t-1}\right)$$

- Compute reset gate similarly but with different weights

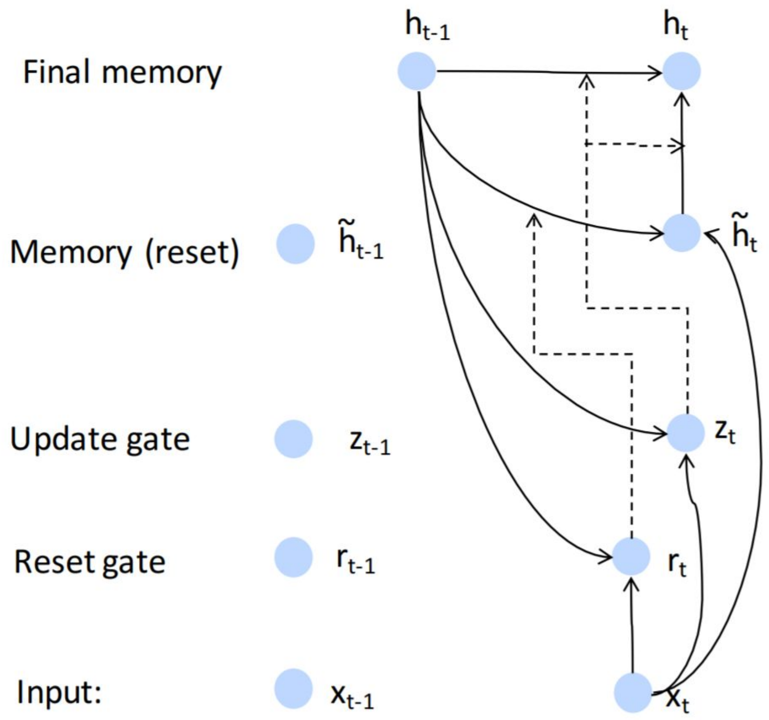
$$r_t = \sigma\left(W^{(r)}x_t + U^{(r)}h_{t-1}\right)$$

GRUs

- Update gate $z_t = \sigma\left(W^{(z)}x_t + U^{(z)}h_{t-1}\right)$
- Reset gate $r_t = \sigma\left(W^{(r)}x_t + U^{(r)}h_{t-1}\right)$
- New memory content: $\tilde{h}_t = \tanh\left(Wx_t + r_t \circ Uh_{t-1}\right)$
  If reset gate unit is ~0, then this ignores previous memory and only stores the new word information
- Final memory at time step combines current and previous time steps: $h_t = z_t \circ h_{t-1} + (1 - z_t) \circ \tilde{h}_t$

Slides from
http://web.stanford.edu/class/cs224n/syllabus.html

Attempt at a clean illustration



$$z_t = \sigma\left(W^{(z)}x_t + U^{(z)}h_{t-1}\right)$$
$$r_t = \sigma\left(W^{(r)}x_t + U^{(r)}h_{t-1}\right)$$
$$\tilde{h}_t = \tanh\left(Wx_t + r_t \circ Uh_{t-1}\right)$$
$$h_t = z_t \circ h_{t-1} + (1 - z_t) \circ \tilde{h}_t$$

Slides from
http://web.stanford.edu/class/cs224n/syllabus.html

GRU intuition

- If reset is close to 0, ignore previous hidden state → Allows model to drop information that is irrelevant in the future

$$z_t = \sigma\left(W^{(z)}x_t + U^{(z)}h_{t-1}\right)$$
$$r_t = \sigma\left(W^{(r)}x_t + U^{(r)}h_{t-1}\right)$$
$$\tilde{h}_t = \tanh\left(Wx_t + r_t \circ Uh_{t-1}\right)$$
$$h_t = z_t \circ h_{t-1} + (1 - z_t) \circ \tilde{h}_t$$

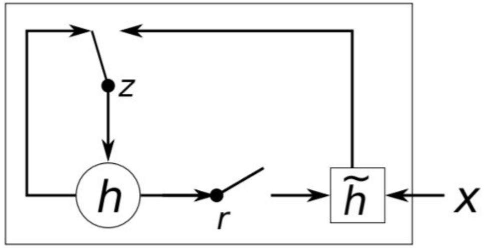
- Update gate z controls how much of past state should matter now.
  - If z close to 1, then we can copy information in that unit through many time steps! **Less vanishing gradient**!
- Units with short-term dependencies often have reset gates very active

GRU intuition

- Units with long term dependencies have active update gates z

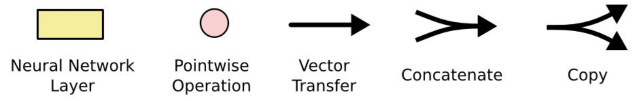
$$z_t = \sigma\left(W^{(z)}x_t + U^{(z)}h_{t-1}\right)$$
$$r_t = \sigma\left(W^{(r)}x_t + U^{(r)}h_{t-1}\right)$$
$$\tilde{h}_t = \tanh\left(Wx_t + r_t \circ Uh_{t-1}\right)$$
$$h_t = z_t \circ h_{t-1} + (1 - z_t) \circ \tilde{h}_t$$

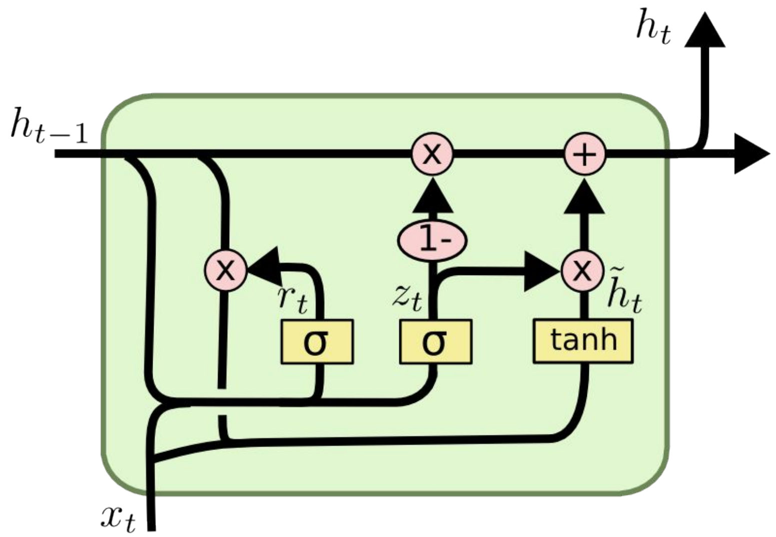
- Illustration:



- Derivative of $\frac{\partial}{\partial x_1} x_1 x_2$? → rest is same chain rule, but implement with **modularization** or automatic differentiation

GRU : Gated Recurrent Units





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

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

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

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

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

LSTMs : Long Short Term Memory

Hochreiter & Schmidhuber (1997) solved the problem of getting an RNN to remember things for a long time (like hundreds of time steps).

Based on a standard RNN whose neuron activates with *tanh*...

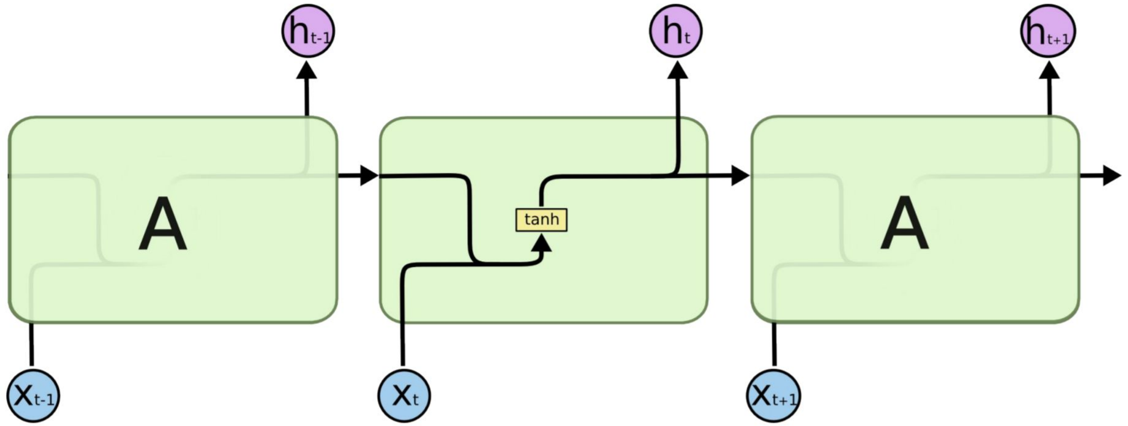


Image from
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTM

C_t is the cell state, which flows through the entire chain...

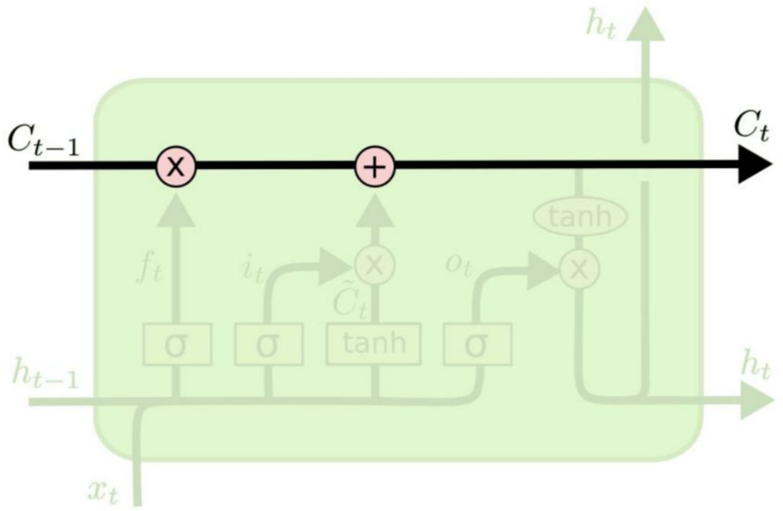
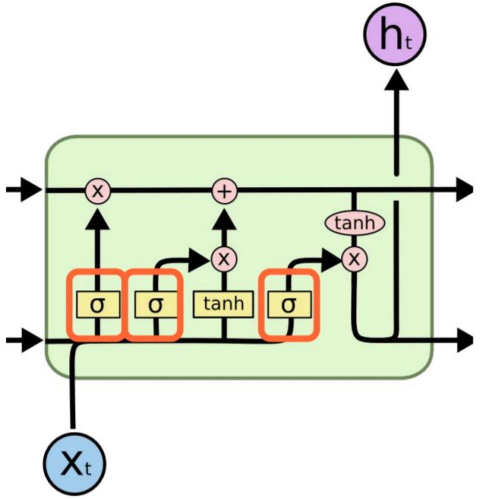
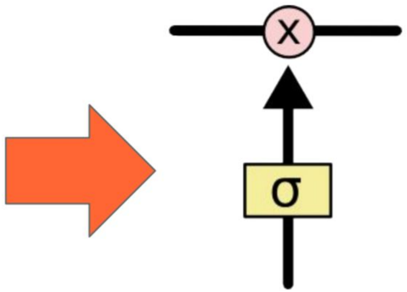


Image from
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTM

Three **gates** are governed by *sigmoid* units (btw [0,1]) define the control of in & out information..





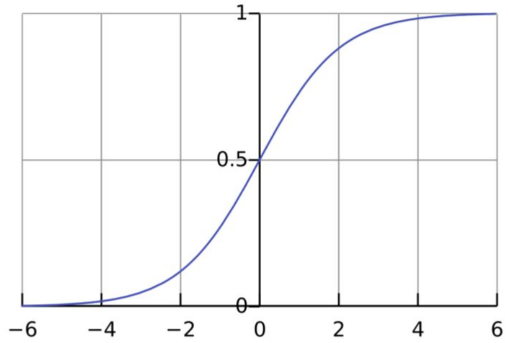
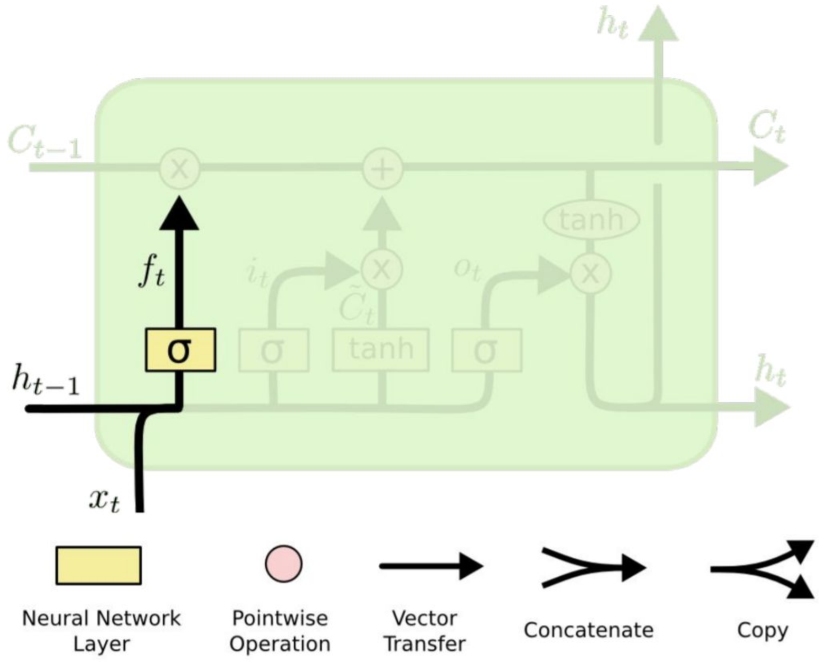


Image from
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTM

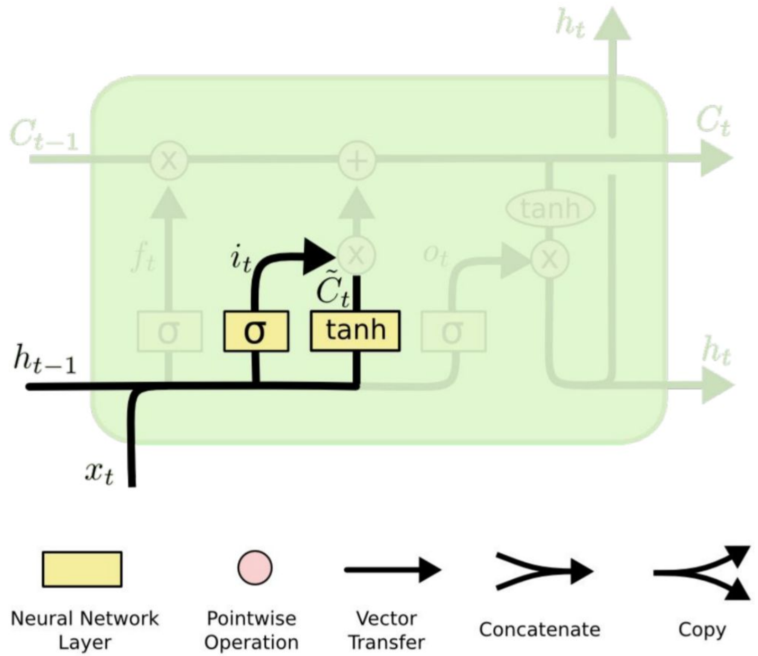


Forget Gate:

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

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

LSTM



Input Gate Layer

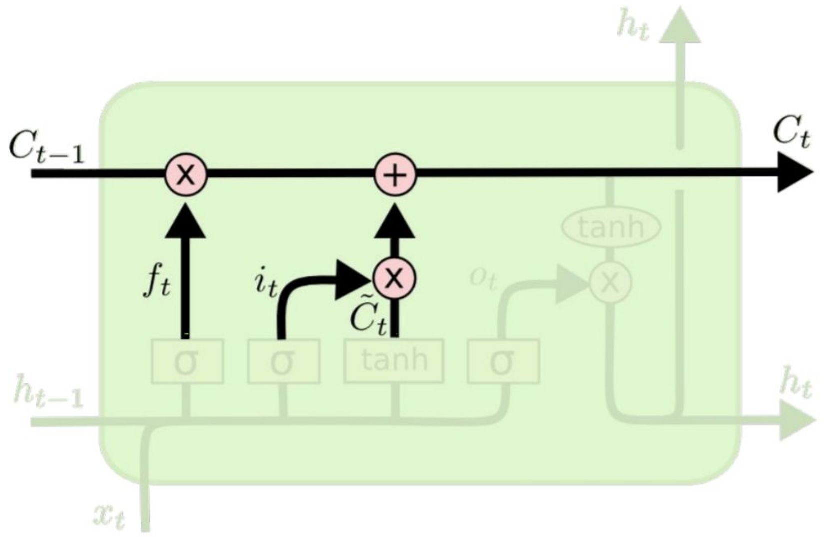
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$

New contribution to cell state

$$\tilde{C}_t = \underbrace{\tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)}_{\text{Classic neuron}}$$

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

LSTM

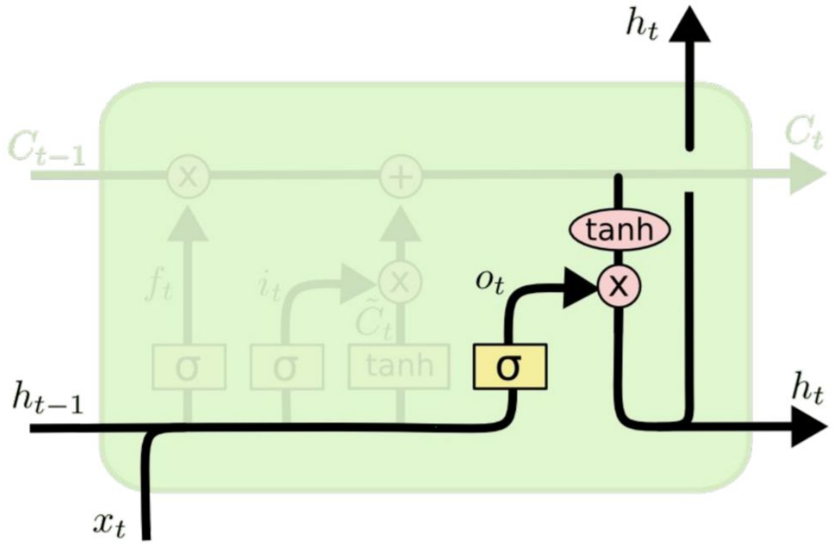


Update Cell State (memory):

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

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

LSTM



Output Gate Layer

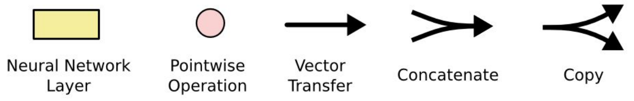
$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$

Output to next layer

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

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

LSTM



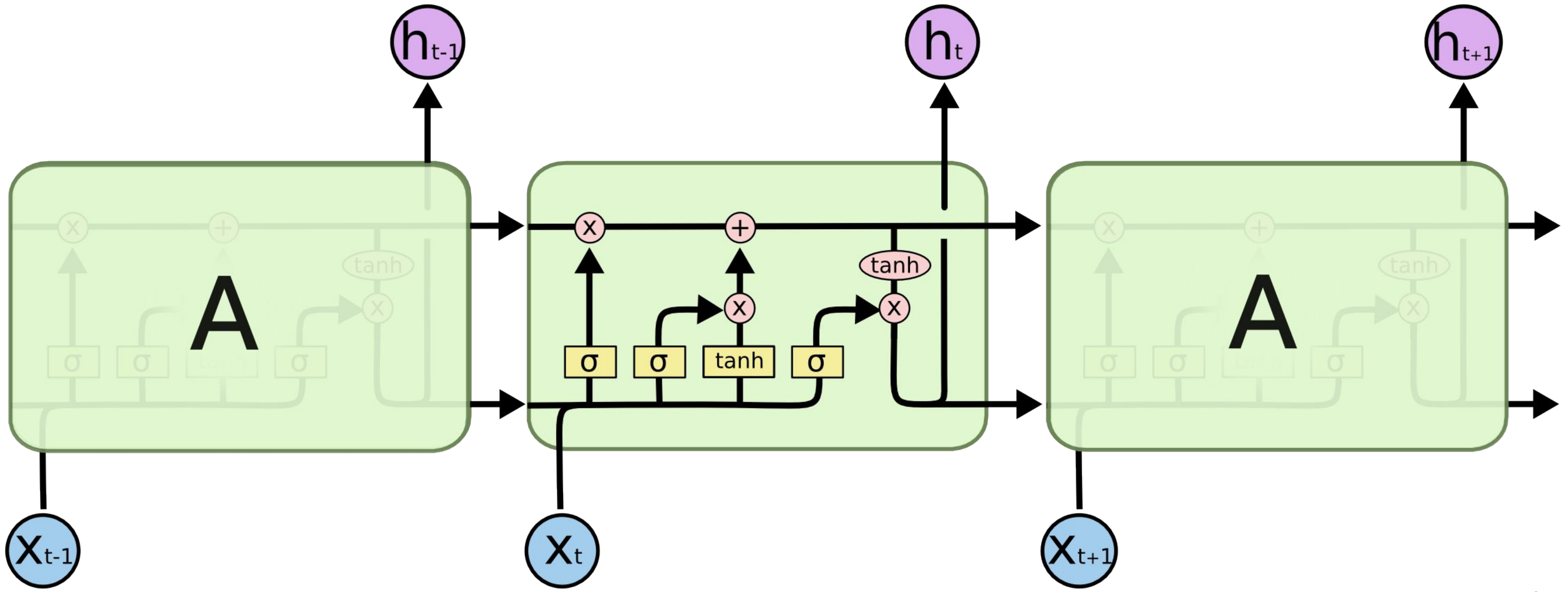
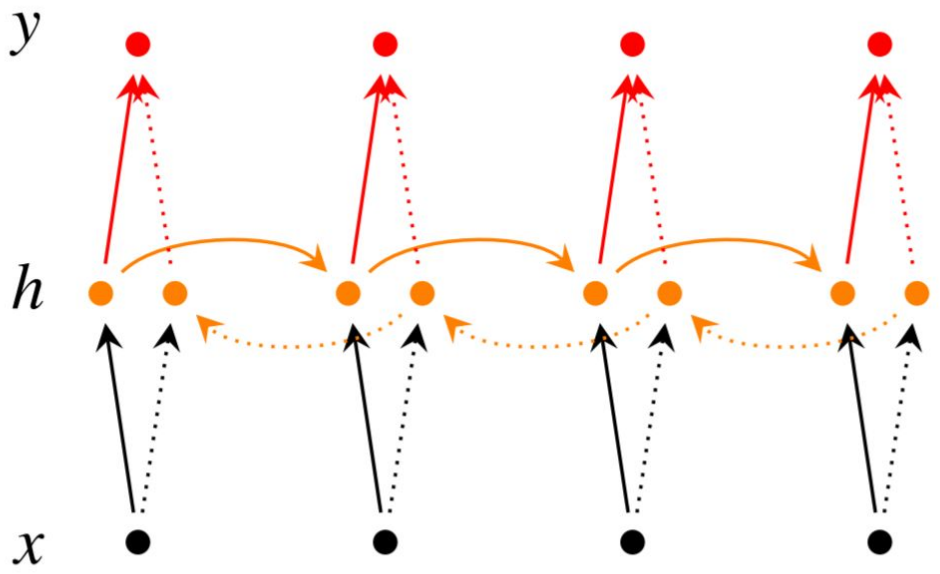


Image from
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Bidirectional RNNs



$$\overrightarrow{h}_t = f(\overrightarrow{W} x_t + \overrightarrow{V} \overrightarrow{h}_{t-1} + \overrightarrow{b})$$

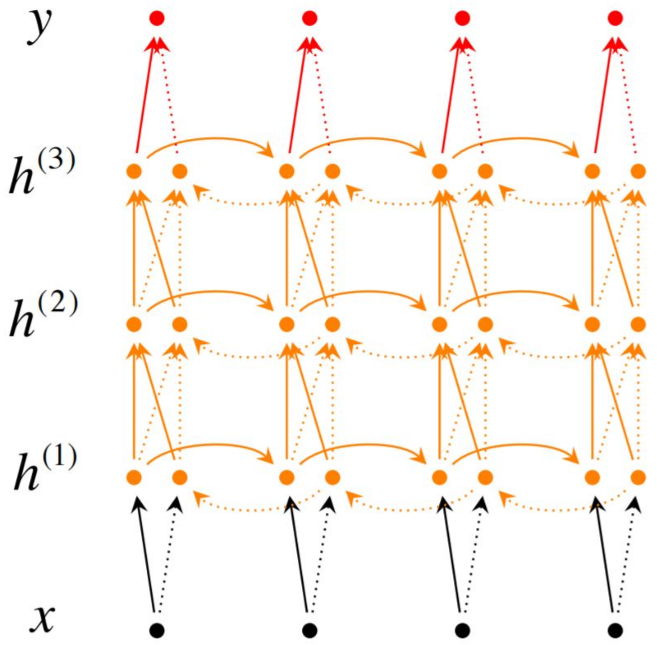
$$\overleftarrow{h}_t = f(\overleftarrow{W} x_t + \overleftarrow{V} \overleftarrow{h}_{t+1} + \overleftarrow{b})$$

$$y_t = g(U[\overrightarrow{h}_t; \overleftarrow{h}_t] + c)$$

$h = [\overrightarrow{h}; \overleftarrow{h}]$ now represents (summarizes) the past and future around a single token.

credit:??

Stacking RNNs



$$\overrightarrow{h}_t^{(i)} = f(\overrightarrow{W}^{(i)} h_t^{(i-1)} + \overrightarrow{V}^{(i)} \overrightarrow{h}_{t-1}^{(i)} + \overrightarrow{b}^{(i)})$$

$$\overleftarrow{h}_t^{(i)} = f(\overleftarrow{W}^{(i)} h_t^{(i-1)} + \overleftarrow{V}^{(i)} \overleftarrow{h}_{t+1}^{(i)} + \overleftarrow{b}^{(i)})$$

$$y_t = g(U[\overrightarrow{h}_t^{(L)}; \overleftarrow{h}_t^{(L)}] + c)$$

Each memory layer passes an intermediate sequential representation to the next.

credit:??

References

1. Stanford NLP course : http://web.stanford.edu/class/cs224n/syllabus.html
2. Wild ML blog post : http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/
3. Colahs blog post : http://colah.github.io/posts/2015-08-Understanding-LSTMs/
4. Bengio’s slides : https://drive.google.com/open?id=0ByUKRdiCDK7-LXZkM3hVSzFGTkE
5.