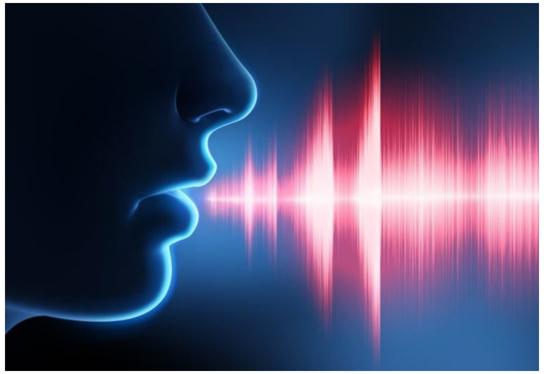
Recurrent Neural Networks

RNN basics

Yingzhen Li (yingzhen.li@imperial.ac.uk)

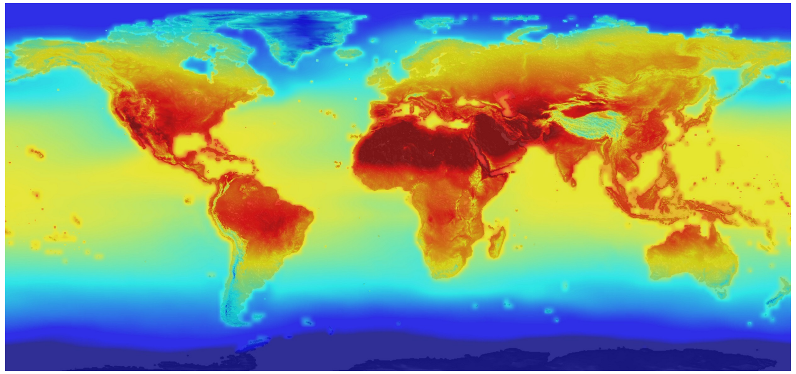
Sequential data is everywhere

Video data



Speech data

Financial time series data



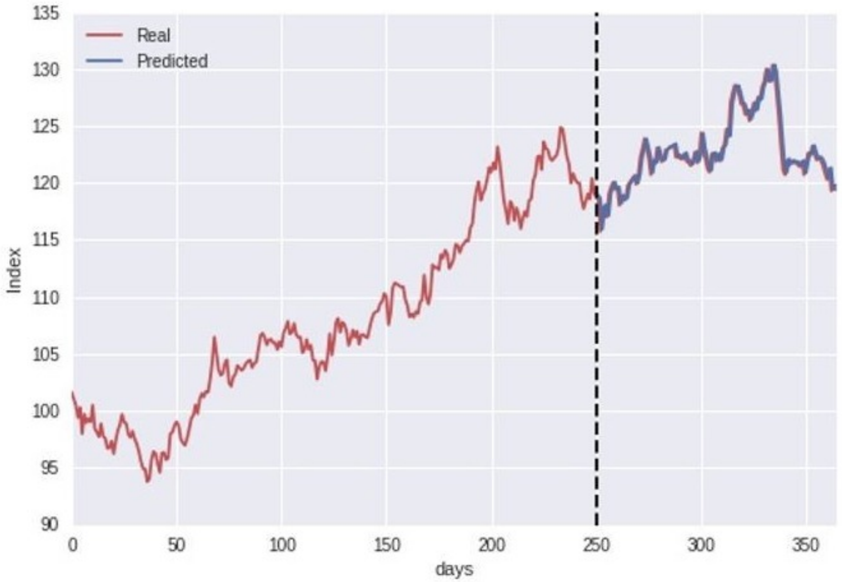
Climate science data (spatial-temporal)

Sequential data is everywhere

Time series prediction:

- Data sequence: $(x_1, \dots, x_T)$
    - x_t: data frame at time t
- Goal: predict future values

$$(x_1, \dots, x_T) \rightarrow x_{T+1}, x_{T+2}, \dots$$

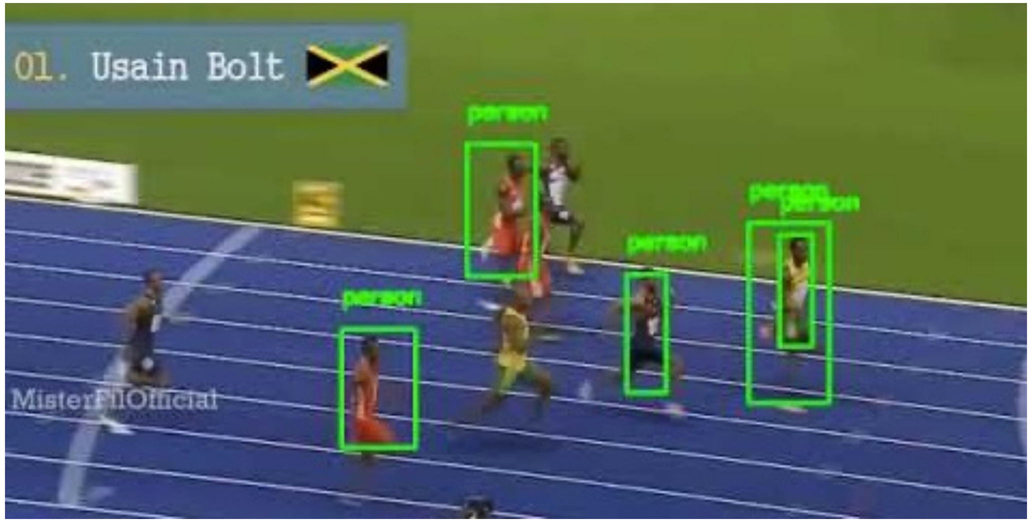


Sequential data is everywhere

Object tracking in videos:

- Data sequence: $(x_1, \ldots, x_T)$
  - x_t: video frame at time t
- Label sequence: $(y_1, \ldots, y_T)$
  - y_t: object identifier, bounding box coordinates, … at time t
- Goal: learn a mapping

$$(x_1, \ldots, x_T) \rightarrow (y_1, \ldots, y_T)$$

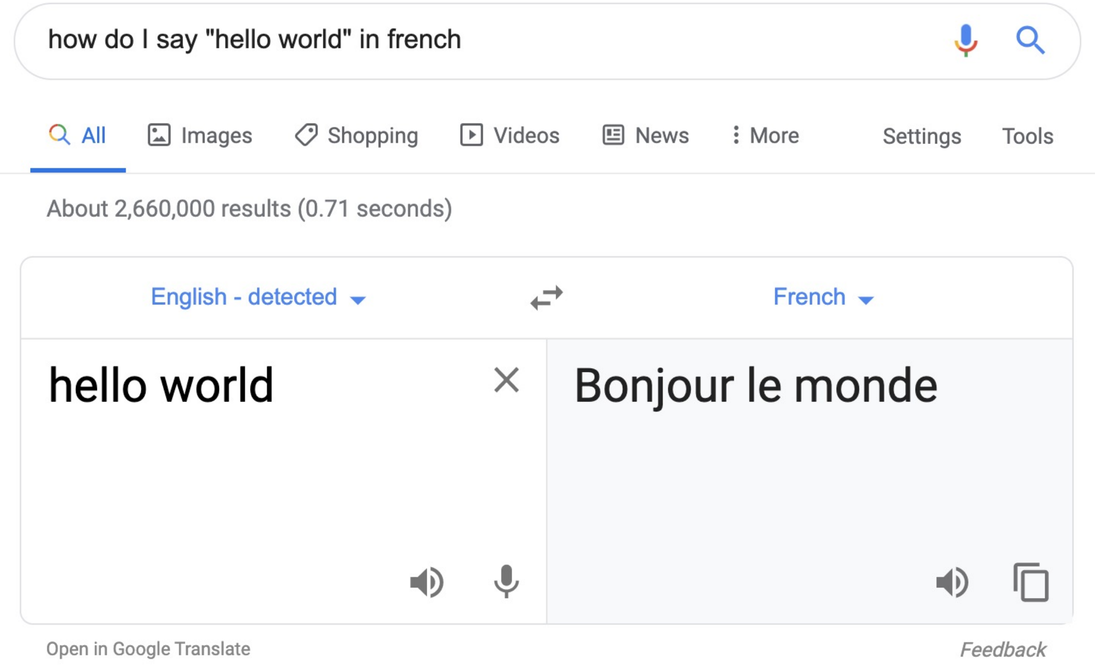


Sequential data is everywhere

Machine translation (e.g. EN to FR):

- Data sequence: $x = (x_1, \dots, x_T)$
  - x_t: the t^{th} word in the English sentence
- Output sequence: $y = (y_1, \dots, y_L)$
  - y_l: the l^{th} word in the French sentence
- Goal: learn a mapping

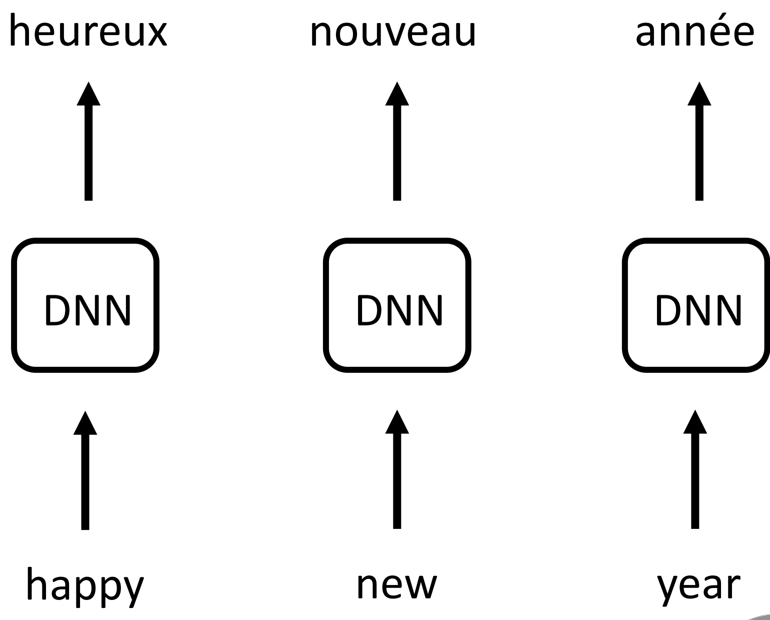
$$x \to y$$



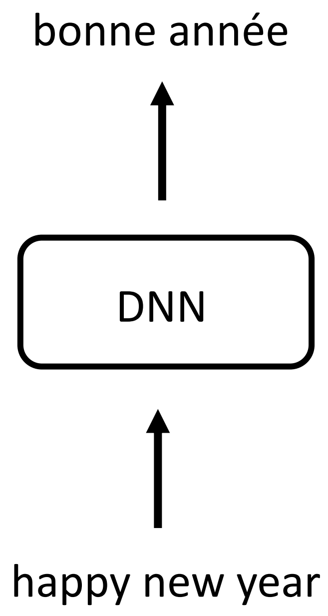
Deep learning's solution: recurrent neural networks

Why Recurrent Neural Networks

Machine translation as a motivating example:



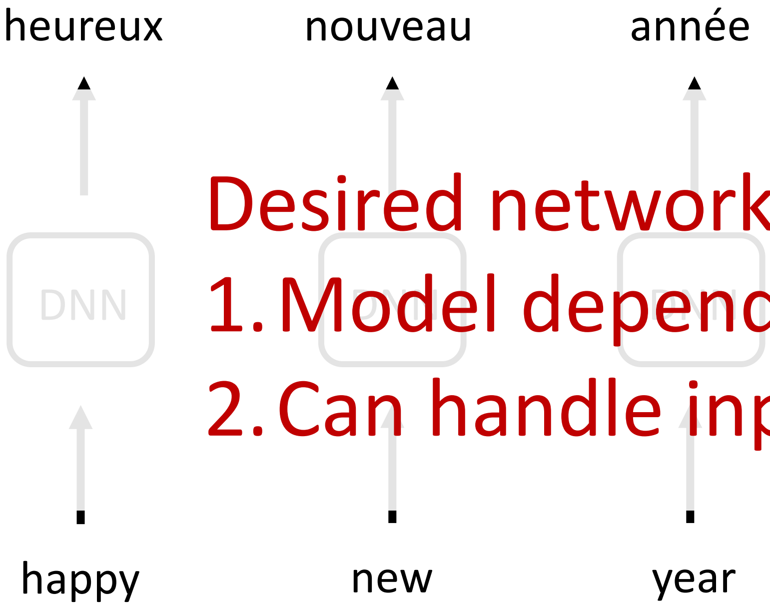
ignoring dependencies

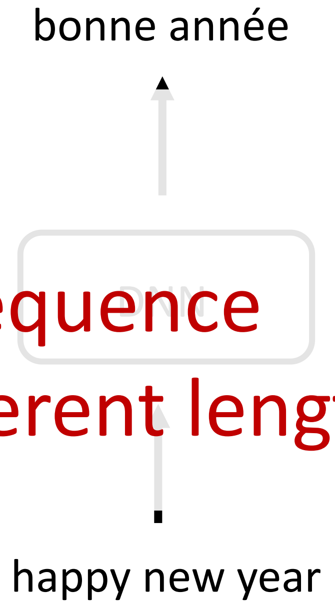


cannot handle inputs of varied lengths

Why Recurrent Neural Networks

Machine translation as a motivating example:





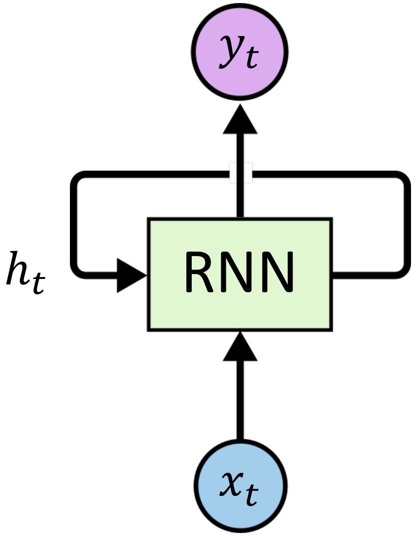
Desired network architecture:

1. Model dependences within the sequence
2. Can handle inputs/outputs of different lengths

ignoring dependencies

cannot handle inputs of varied lengths

Simple RNNs



$$h_t = \phi_h(W_h h_{t-1} + W_x x_t + b_h)$$
$$y_t = \phi_y(W_y h_t + b_y)$$

ϕ_h: activation function for recurrent state
ϕ_y: activation function for output

Figure adapted from https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Simple RNNs

Unrolling the RNN architecture through time:

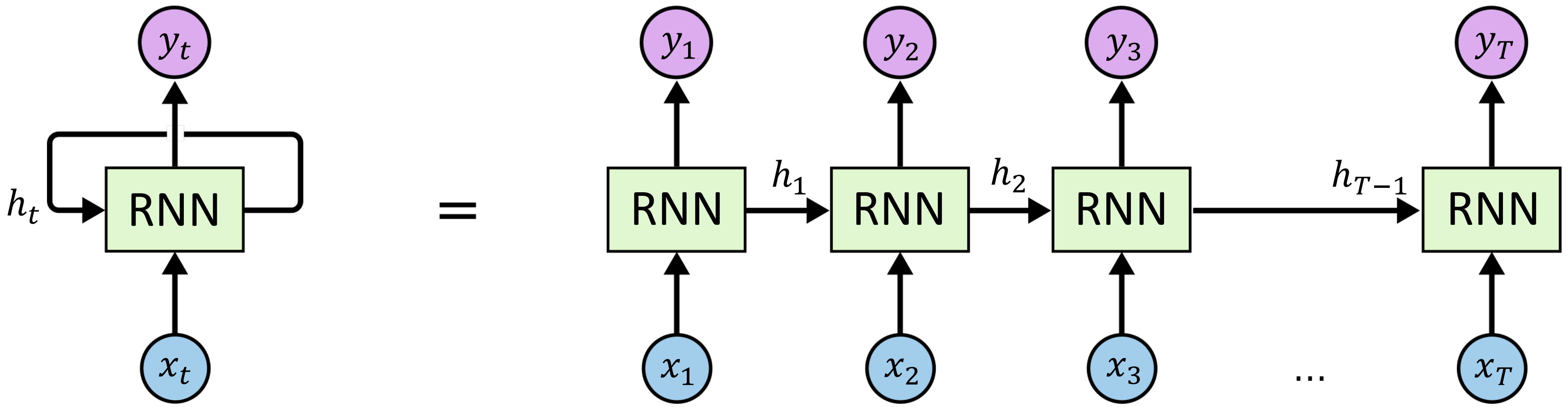
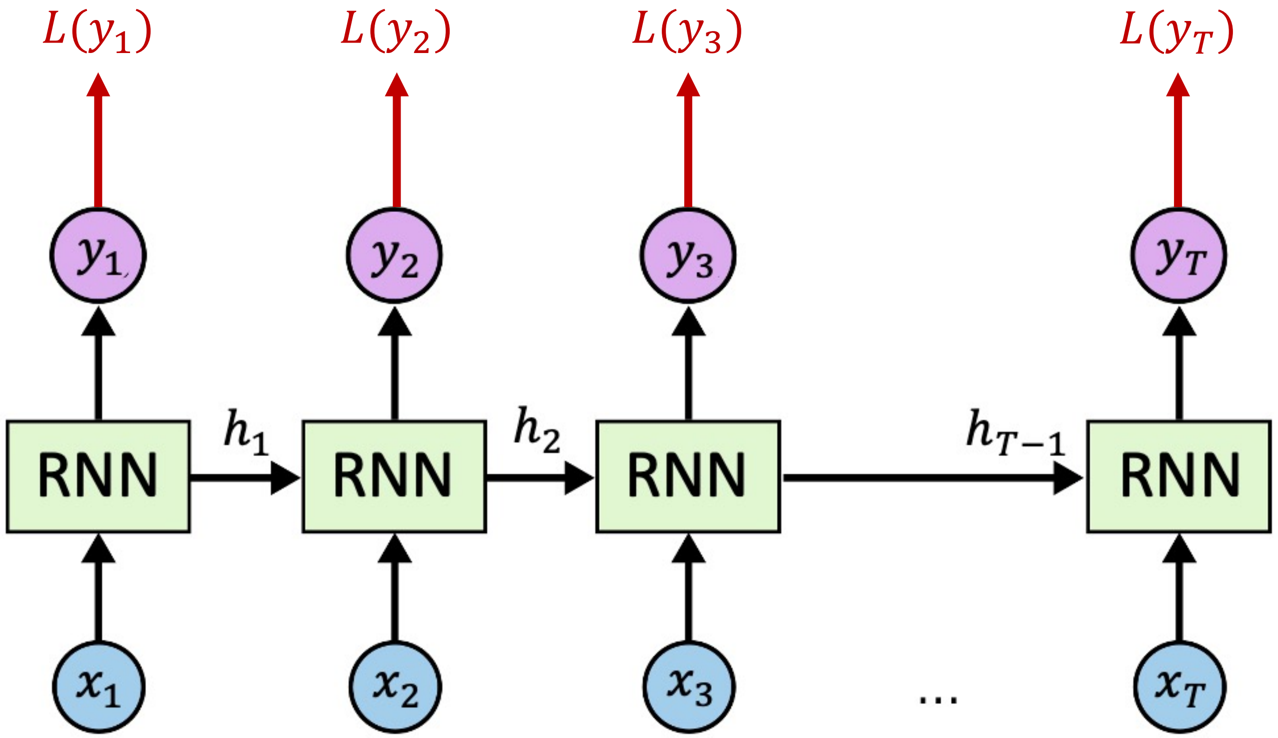


Figure adapted from https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Training RNNs

Forward pass: $L_{total}(\theta) = \sum_{t=1}^{T} L(y_t),\ \theta = \{W_h, W_x, W_y, b_h, b_y\}$

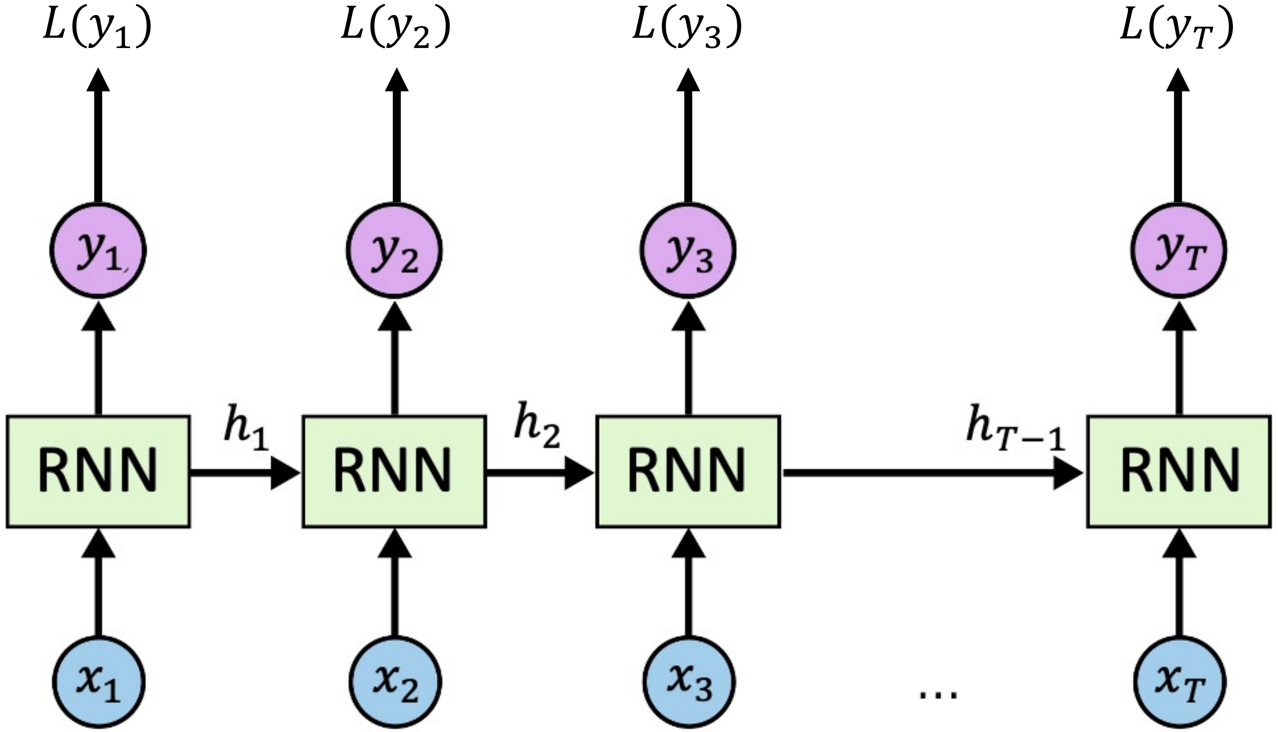


$$h_t = \phi_h(W_h h_{t-1} + W_x x_t + b_h)$$
$$y_t = \phi_y(W_y h_t + b_y)$$

Figure adapted from https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Training RNNs

Backward pass: $\frac{d}{d\theta} L_{total}(\theta) = \sum_{t=1}^{T} \frac{d}{d\theta} L(y_t), \theta = \{W_h, W_x, W_y, b_h, b_y\}$



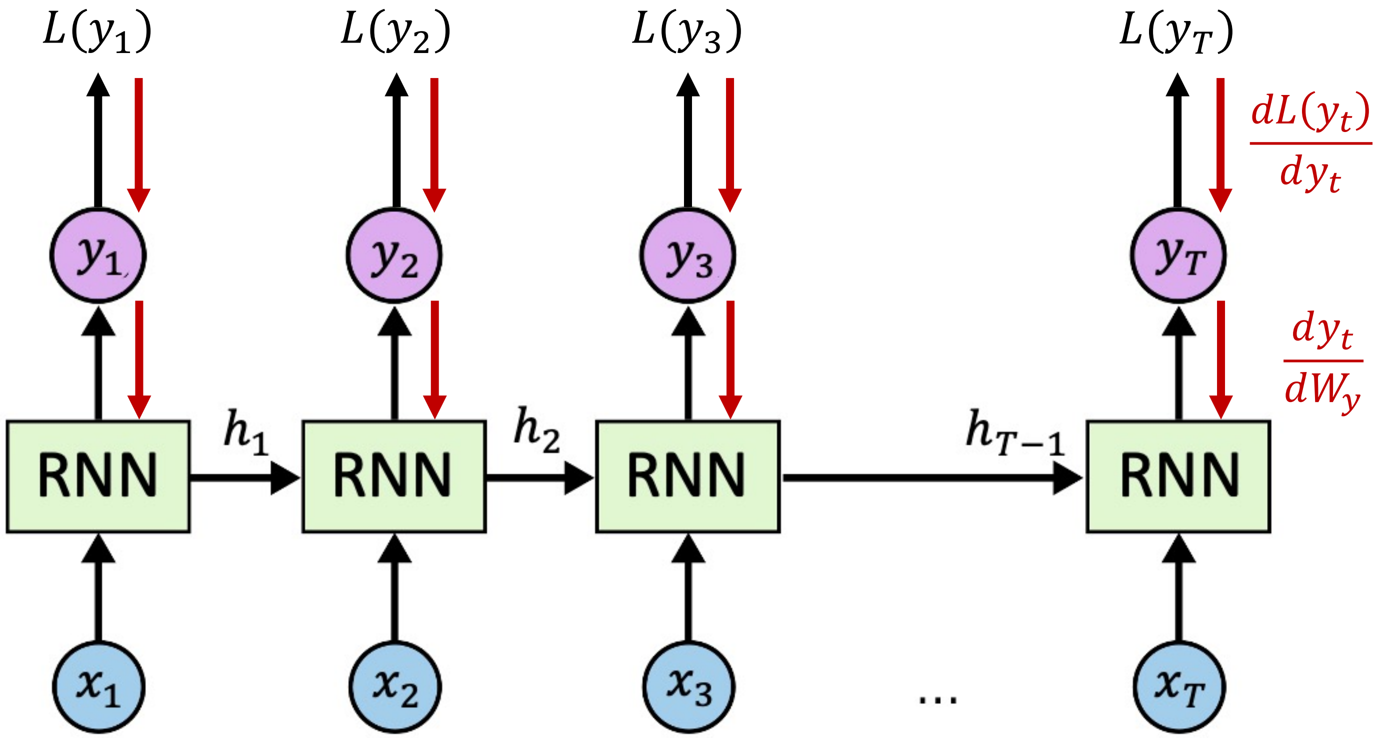
$$h_t = \phi_h(W_h h_{t-1} + W_x x_t + b_h)$$
$$y_t = \phi_y(W_y h_t + b_y)$$

Figure adapted from https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Training RNNs

Backward pass: $\frac{d}{dW_y} L_{total} = \sum_{t=1}^{T} \frac{d}{dW_y} L(y_t)$

$\frac{d}{dW_y} L(y_t) = \frac{dL(y_t)}{dy_t} \frac{dy_t}{dW_y}$



$$h_t = \phi_h(W_h h_{t-1} + W_x x_t + b_h)$$
$$y_t = \phi_y(W_y h_t + b_y)$$

Figure adapted from https://colah.github.io/posts/2015-08-Understanding-LSTMs/

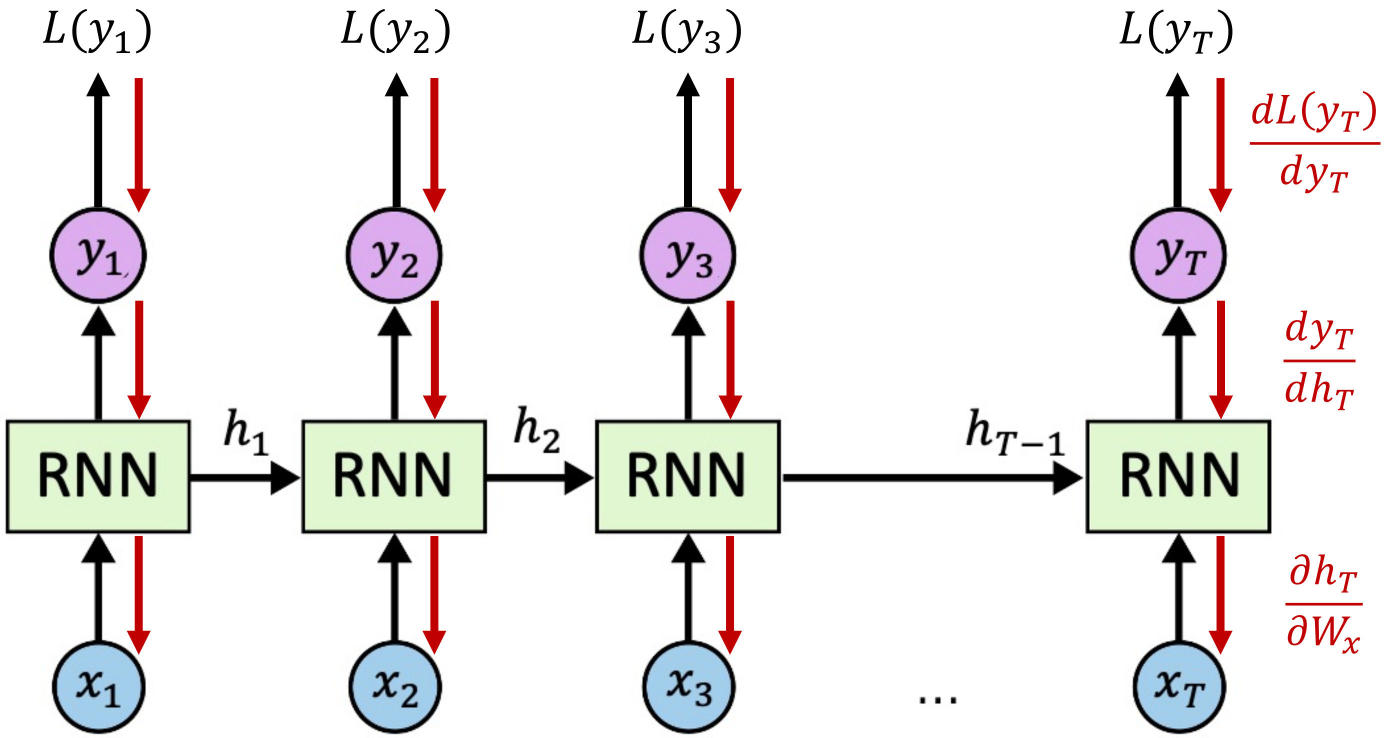
Training RNNs

$$\frac{dL(y_t)}{dW_x} = \frac{dL(y_t)}{dy_t}\frac{dy_t}{dh_t}\frac{dh_t}{dW_x}$$

$$\frac{dh_t}{dW_x} = \frac{\partial h_t}{\partial W_x} + \frac{\partial h_t}{\partial h_{t-1}}\frac{dh_{t-1}}{dW_x}$$

total gradient partial gradient

Backward pass: $\frac{d}{dW_x}L_{total} = \sum_{t=1}^{T}\frac{d}{dW_x}L(y_t)$



$$h_t = \phi_h(W_h h_{t-1} + W_x x_t + b_h)$$
$$y_t = \phi_y(W_y h_t + b_y)$$

Figure adapted from https://colah.github.io/posts/2015-08-Understanding-LSTMs/

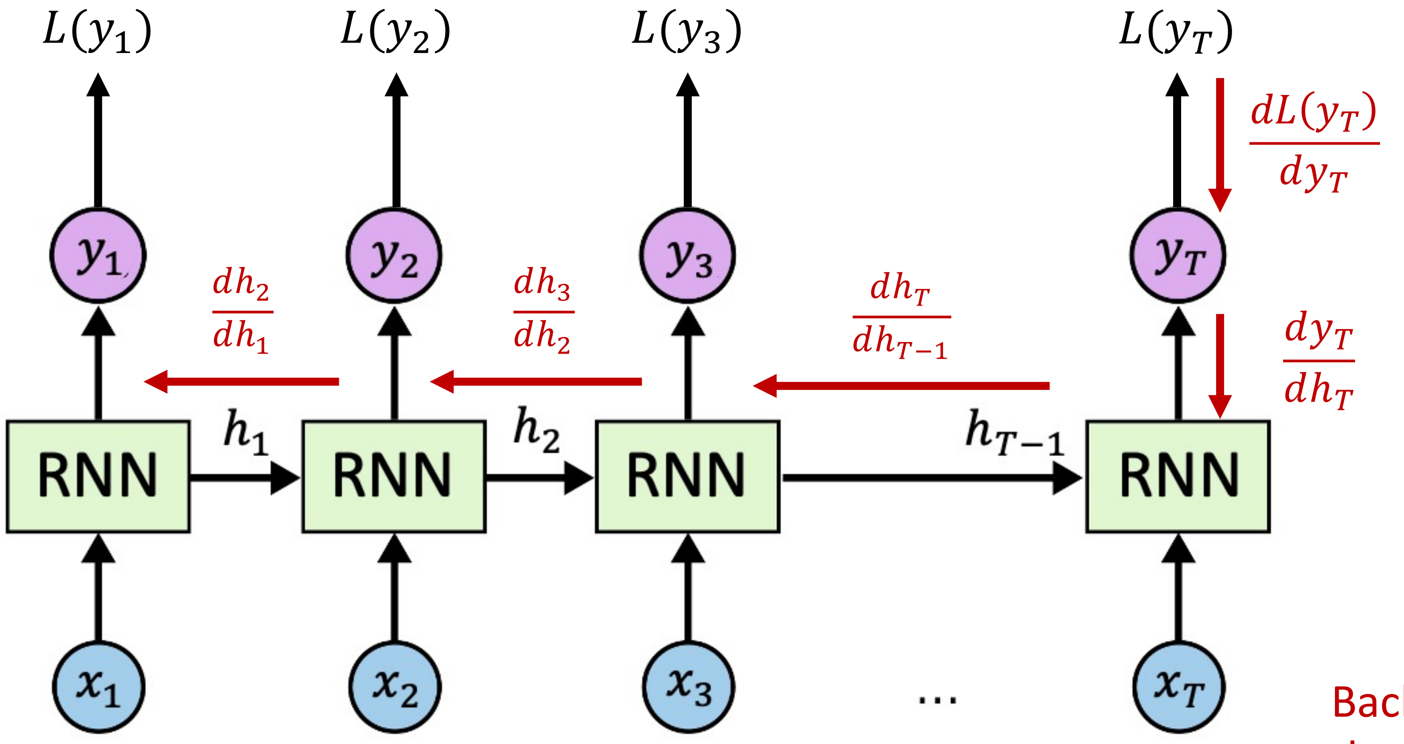
Training RNNs

$$\frac{dL(y_t)}{dW_h} = \frac{dL(y_t)}{dy_t}\frac{dy_t}{dh_t}\frac{dh_t}{dW_h}$$

$$\frac{dh_t}{dW_h} = \frac{\partial h_t}{\partial W_h} + \frac{\partial h_t}{\partial h_{t-1}}\frac{dh_{t-1}}{dW_h}$$

total gradient partial gradient

Backward pass: $\frac{d}{dW_h} L_{total} = \sum_{t=1}^{T} \frac{d}{dW_h} L(y_t)$



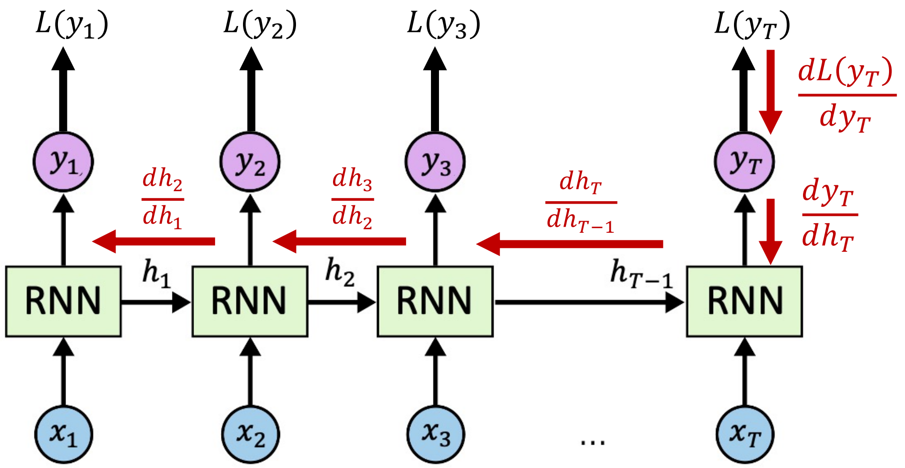
$$h_t = \phi_h(W_h h_{t-1} + W_x x_t + b_h)$$
$$y_t = \phi_y(W_y h_t + b_y)$$

Back-propagation through time (BPTT)

Figure adapted from https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Issues of simple RNNs

Consider gradient of loss w.r.t. W_h:



$$\frac{dL(y_t)}{dW_h} = \frac{dL(y_t)}{dy_t}\frac{dy_t}{dh_t}\frac{dh_t}{dW_h}$$

$$\frac{dh_t}{dW_h} = \frac{\partial h_t}{\partial W_h} + \frac{dh_t}{dh_{t-1}}\frac{dh_{t-1}}{dW_h}$$

$$= \frac{\partial h_t}{\partial W_h} + \frac{dh_t}{dh_{t-1}}\left(\frac{\partial h_{t-1}}{\partial W_h} + \frac{dh_{t-1}}{dh_{t-2}}\frac{dh_{t-2}}{dW_h}\right) = \ \ldots$$

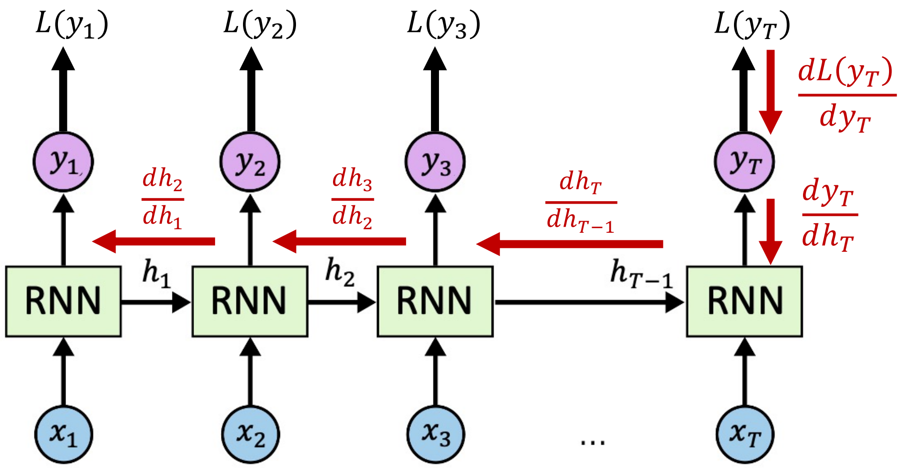
$$\Rightarrow \frac{dh_t}{dW_h} = \sum_{\tau=1}^{t}\left(\prod_{l=\tau}^{t-1}\frac{dh_{l+1}}{dh_l}\right)\frac{\partial h_\tau}{\partial W_h}, \qquad \frac{dh_{l+1}}{dh_l}^{\top} = \phi_h'(W_h h_l + W_x x_{l+1} + b_h) \odot W_h$$

contain products of W_h and ϕ_h' for $t - \tau$ times

Depending on W_h and the non-linearity ϕ_h, when $t \to \infty$, $\prod_{l=1}^{t-1}\frac{dh_{l+1}}{dh_l}$ can vanish or explode!

Issues of simple RNNs

Consider gradient of loss w.r.t. W_h:



$$\Rightarrow \frac{dh_t}{dW_h} = \sum_{\tau=1}^{t} \left(\prod_{l=\tau}^{t-1} \frac{dh_{l+1}}{dh_l} \right) \frac{\partial h_\tau}{\partial W_h},$$

contain products of W_h and ϕ_h' for $t - \tau$ times

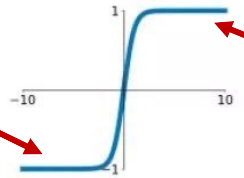
Identity mapping: $\phi_h(t) = t$

$$\prod_{l=1}^{t-1} \frac{dh_{l+1}}{dh_l} = (W_h^{t-1})^\top$$

(explode or vanish depending on the largest singular value)

Tanh mapping: $\phi_h(t) = \tanh(t)$

$\phi_h'(t) \approx 0$



$\phi_h'(t) \approx 0$

$$\frac{dh_{l+1}}{dh_l}^\top = \phi_h'(W_h h_l + W_x x_{l+1} + b_h) \odot W_h$$

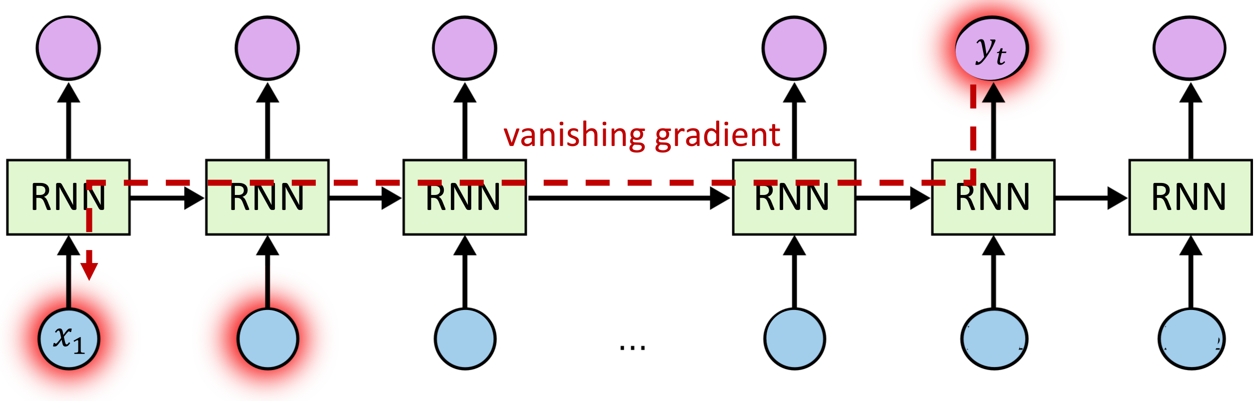
Depending on the non-linearity ϕ_h, when $t \to \infty$, $\prod_{l=1}^{t-1} \frac{dh_{l+1}}{dh_l}$ can vanish or explode!

Issues of simple RNNs

- Consider gradient of loss w.r.t. W_h:

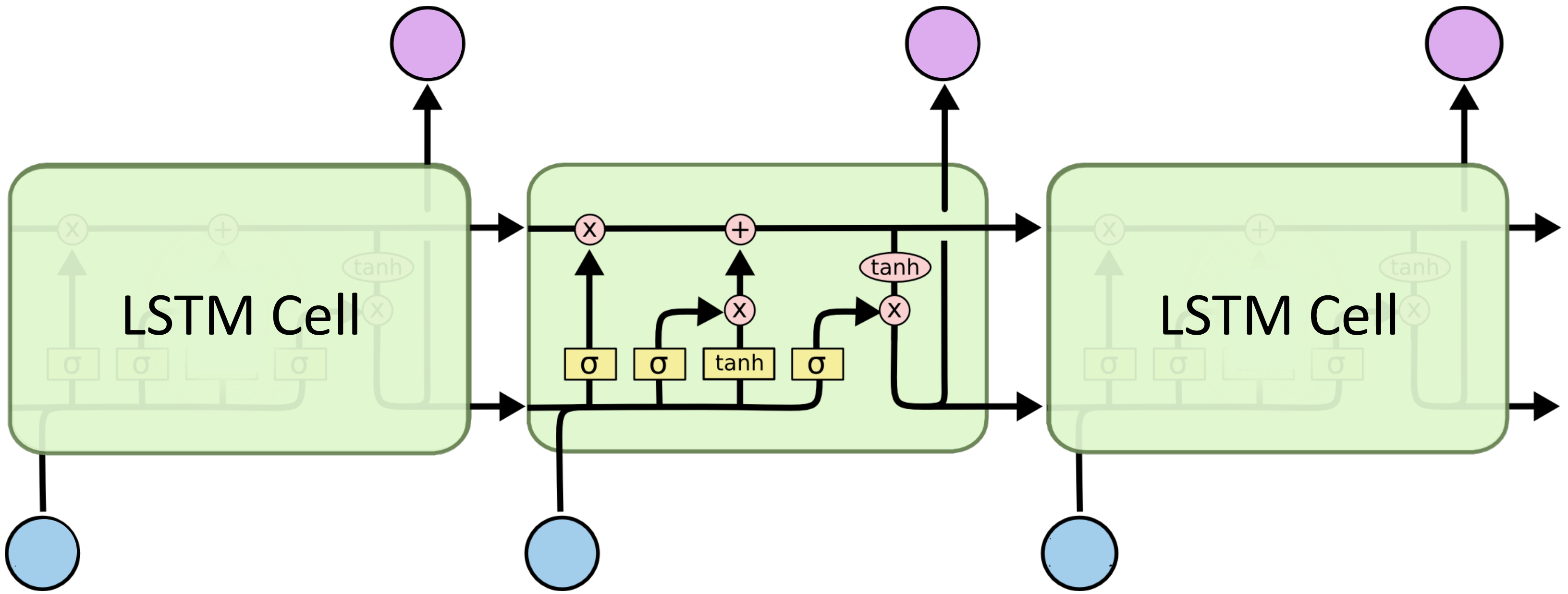
$$\Rightarrow \frac{dh_t}{dW_h} = \sum_{\tau=1}^{t} \left(\prod_{l=\tau}^{t-1} \frac{dh_{l+1}}{dh_l} \right) \frac{\partial h_\tau}{\partial W_h}, \qquad \frac{dh_{l+1}}{dh_l}^{\top} = \phi_h'(W_h h_l + W_x x_{l+1} + b_h) \odot W_h$$

can vanish or explode (especially when $t \gg \tau$)



Dependency of y_t on x_1 gets harder to learn as t increases

Long Short-Term Memory (LSTM)

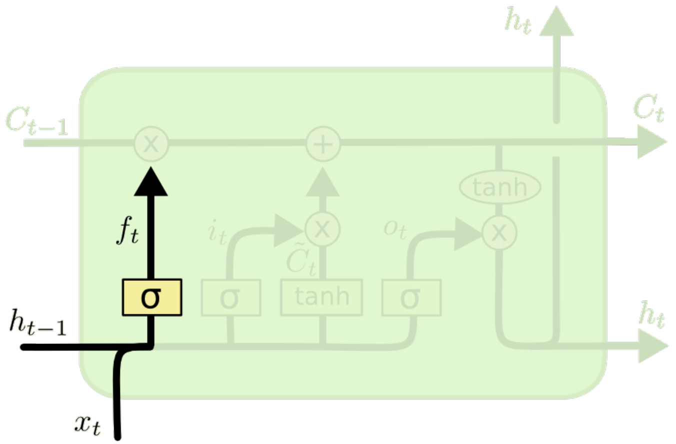


Key ideas of LSTM:

- Introduce cell state C_t
- Gating mechanisms to control cell state updates and output values

Hochreiter and Schmidhuber (1997). Long Short-Term Memory. Neuro Computation.
Figure adapted from https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Long Short-Term Memory (LSTM)

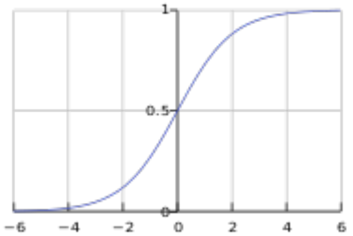


Forget gate f_t:

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

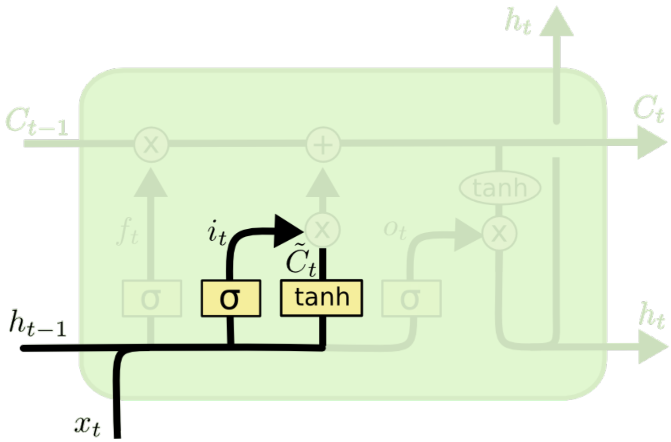
sigmoid activation function

$$\sigma(t) = \frac{1}{1 + \exp(-t)}$$



Hochreiter and Schmidhuber (1997). Long Short-Term Memory. Neuro Computation.
Figure adapted from https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Long Short-Term Memory (LSTM)

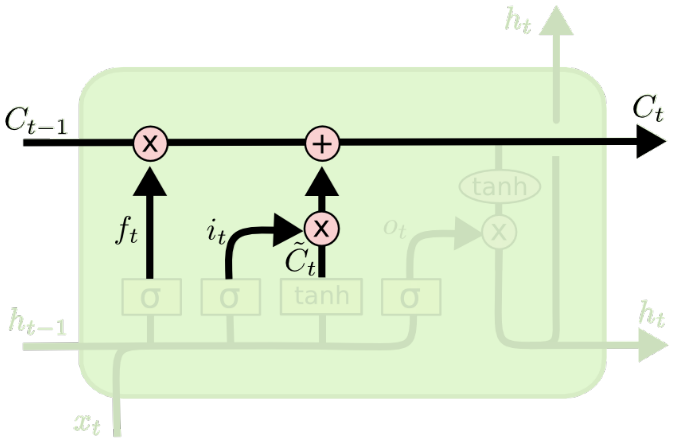


Input gate: i_t
Candidate cell state update: $\tilde{c}_t$

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$
$$\tilde{c}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c)$$

Hochreiter and Schmidhuber (1997). Long Short-Term Memory. Neuro Computation.
Figure adapted from https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Long Short-Term Memory (LSTM)



Cell state update:

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

$f_t[d] \in (0,1)$

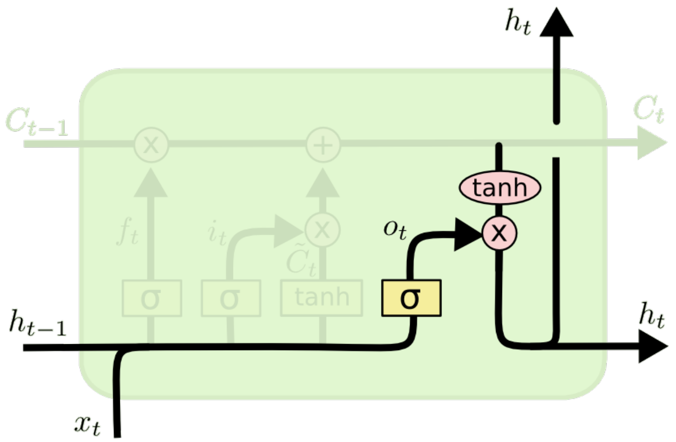
$f_t[d] \to 0$: forget previous state
$f_t[d] \to 1$: maintain previous state

$i_t[d] \in (0,1)$

$i_t[d] \to 0$: discard candidate cell state update
$i_t[d] \to 1$: enable cell state update

Hochreiter and Schmidhuber (1997). Long Short-Term Memory. Neuro Computation.
Figure adapted from https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Long Short-Term Memory (LSTM)



Update the hidden state h_t
with output gating o_t

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$
$$h_t = o_t \odot \tanh(c_t)$$

$o_t[d] \in (0,1)$

$o_t[d] \to 0$: zero output
$o_t[d] \to 1$: output cell state (squashed in $(-1, 1)$)

Prediction of y_t can proceed in a similar way as in simple RNNs:

$$y_t = \phi_y(W_y h_t + b_y)$$

Hochreiter and Schmidhuber (1997). Long Short-Term Memory. Neuro Computation.
Figure adapted from https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Long Short-Term Memory (LSTM)

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$
$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$
$$\tilde{c}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c)$$
$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$$
$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$
$$h_t = o_t \odot \tanh(c_t)$$
$$y_t = \phi_y(W_y h_t + b_y)$$

BPTT in LSTMs:

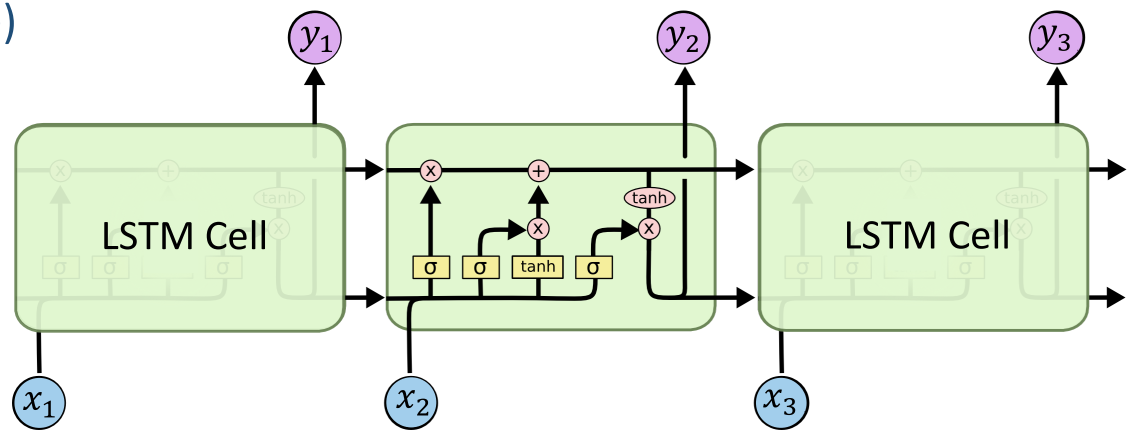
Requires computing $\prod_{l=1}^{t-1} \frac{dc_{l+1}}{dc_l} = \prod_{l=1}^{t-1}(f_{l+1} + o_l \odot \frac{dtanh(c_l)}{dc_l} \odot \frac{dc_{l+1}}{dh_l})$

How to derive $\frac{dc_t}{dc_{t-1}}$: Notice c_t depends on c_{t-1} in 4 paths:

direct dependence

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

Depends on $h_{t-1} = o_{t-1} \odot c_{t-1}$ (indirect dependence)



Hochreiter and Schmidhuber (1997). Long Short-Term Memory. Neuro Computation.
Figure adapted from https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Long Short-Term Memory (LSTM)

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$
$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$
$$\tilde{c}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c)$$
$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$$
$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$
$$h_t = o_t \odot \tanh(c_t)$$
$$y_t = \phi_y(W_y h_t + b_y)$$

BPTT in LSTMs:

Requires computing $\prod_{l=1}^{t-1} \frac{dc_{l+1}}{dc_l} = \prod_{l=1}^{t-1} (f_{l+1} + o_l \odot \frac{d\tanh(c_l)}{dc_l} \odot \frac{dc_{l+1}}{dh_l})$

Alleviating gradient explosion:

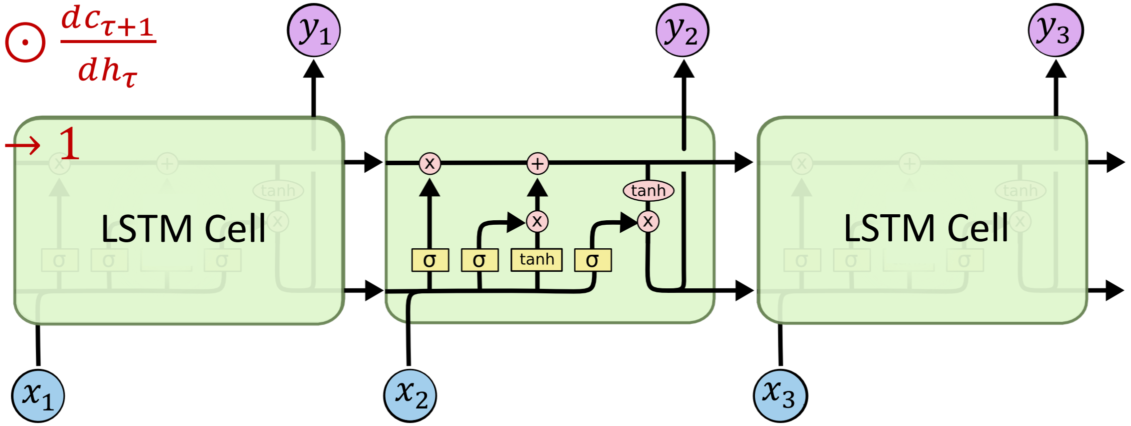
The gradient contains terms proportional to $f_{i+1} \prod_{l=1}^{i-1} o_l \odot \frac{dc_{l+1}}{dh_l}$

≈ 0 when $f_{i+1} \approx 0$

Alleviating gradient vanishing:

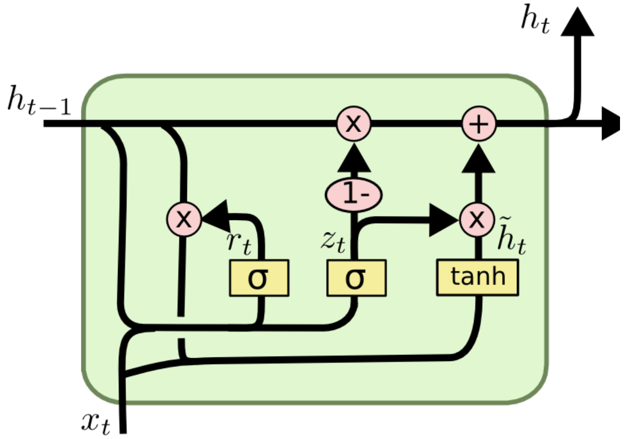
The gradient contains terms proportional to $\prod_{l=\tau+1}^{i} f_l \odot o_\tau \odot \frac{dc_{\tau+1}}{dh_\tau}$

$\approx o_\tau \odot \frac{dc_{\tau+1}}{dh_\tau}$ when $f_l \to 1$
for $l = \tau + 1, \ldots, i$



Hochreiter and Schmidhuber (1997). Long Short-Term Memory. Neuro Computation.
Figure adapted from https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Gated Recurrent Unit (GRU)



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

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

$$\tilde{h}_t = \tanh(W_h \cdot [r_t \odot h_{t-1}, x_t] + b_h)$$

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

Cho et al. Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. EMNLP 2014

Figure adapted from https://colah.github.io/posts/2015-08-Understanding-LSTMs/

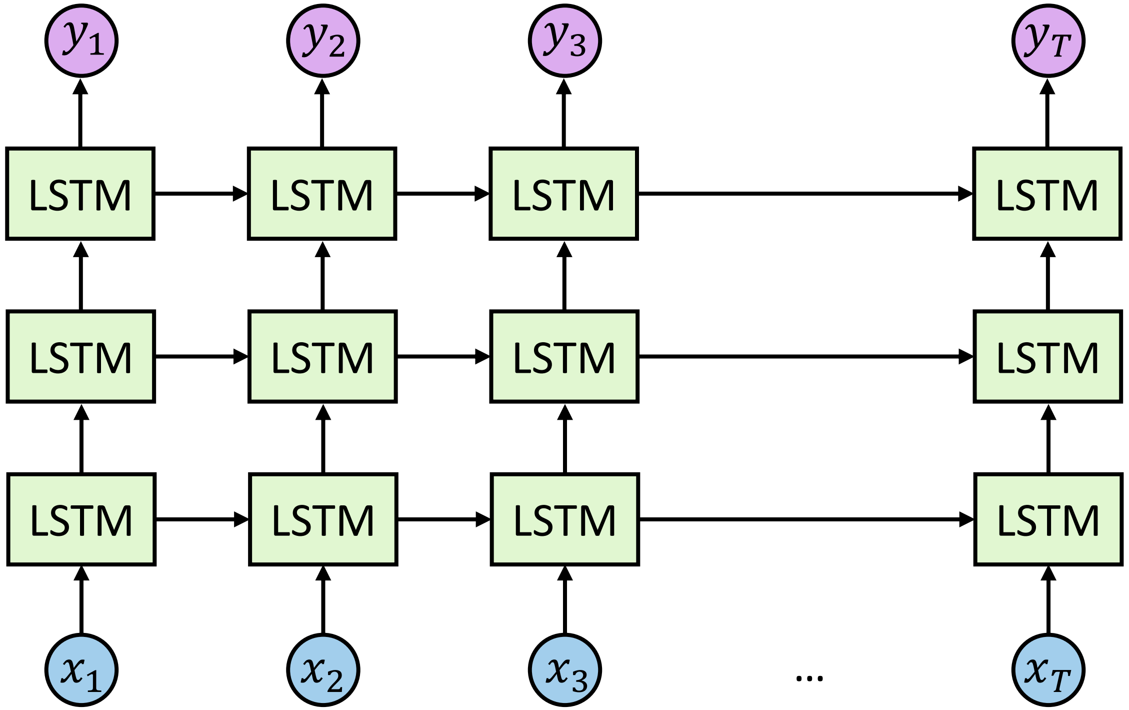
LSTM vs GRU

- Other gated RNN variants exists, but LSTM and GRU are the most widely-used
- GRU is quicker to compute and has fewer parameters
- No conclusive evidence for LSTM > GRU or vice versa
- LSTM is a good default choice (especially if your data has particularly long dependencies, or you have lots of training data)
- Switch to GRU if you want more efficient compute & less overfitting

Stacking LSTMs

Stacking multiple LSTM layers:

- Hidden states of the previous LSTM later as inputs to the next LSTM layer;
- No need to wait for previous LSTM layer to finish forward pass;



Bidirectional LSTMs

Bidirectional LSTM:

- Stacking some LSTM layers;
- For some LSTM layers, the forward pass is reversed from time $t = T$ to $t = 1$;
- If two consecutive LSTM layers are of reversed time ordering, then the top layer needs to wait for the bottom one to finish forward pass.