AOS2– Deep learning

Lecture 04: Introduction to Recurrent Neural Networks

Sylvain Rousseau

Introduction

Sequential data

- Conventional neural networks (MLP, CNN) are good for:
  - Tabular data
  - Image data
- What if data if sequential ? (NLP, speech processing, time series,...)
  - Collection of examples: $\left\{\mathbf{x}_1 = \left(\mathbf{x}_t^{(1)}\right)_{t \in I_1}, \ldots, \mathbf{x}_N = \left(\mathbf{x}_t^{(N)}\right)_{t \in I_N}\right\}$
  - Order matters
  - Different length
  - Different indexing
- Cast as tabular data ?
  - Each $\mathbf{x}_i$ as an example in a tabular data? ...but then different number of features!

Recurrent neural networks are specially designed to handle sequential data

Sequential data models

Two different approaches

- Markov chains
  - Model $p(x_{t+1} \mid x_t, \ldots x_1)$
    - Number of inputs varies
  - Autoregressive models: $p(x_{t+1} \mid x_t, \ldots x_{t-k+1})$
    - x_{t+1} is independent from x_{t-i+1} with $i > k$: no long-term dependency
- Latent variable
  - Model x_{t+1} from a summary of past observations h_t

$$p(x_{t+1} \mid h_t) \quad \text{with} \quad \begin{cases} h_t = f(h_{t-1}, x_t; \boldsymbol{\theta}) \\ h_0 = 0 \end{cases}$$

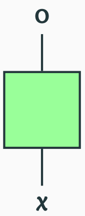
  - h_t is a latent variable summarizing past observations $x_1, \ldots, x_t$
  - f is recurrent! How to learn $\boldsymbol{\theta}$ to fit the data?

Recurrent neural networks

Feed forward *vs* recurrent neural network

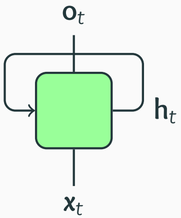
Feed forward neural networks:

- one input $\mathbf{x}$
- one output $\mathbf{o}$



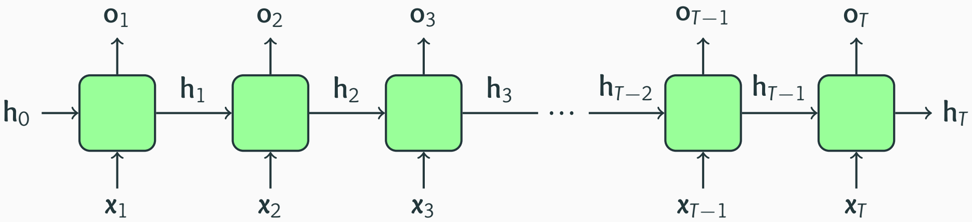
Recurrent neural networks:

- one input $\mathbf{x}_t$ at time t
- one output $\mathbf{o}_t$ at time t
- a hidden state $\mathbf{h}_t$ passed to the next iteration



Unrolled RNN

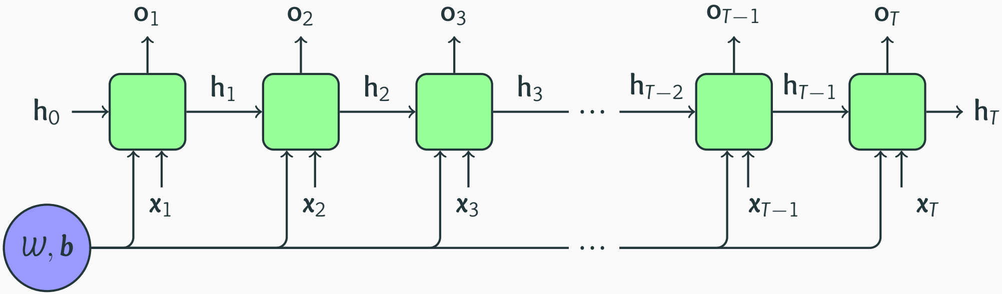
- For a sequence of length T: $\mathbf{x}_1, \ldots, \mathbf{x}_T$



- Equivalent to a feed forward neural network once unrolled except that
  - Parameters are shared across layers
  - Structure depends on (length of) input

Computational graph

- Complete computation graph with parameter dependencies



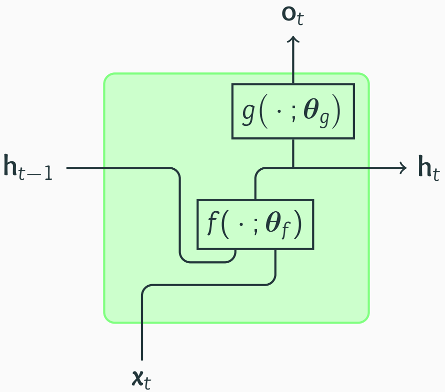
- Parameters are shared across unrolled units
- Gradients receive update from all recurrent layers

General recurrent cell

Forward propagation equations:

$$
\begin{cases}
\mathbf{h}_t = f(\mathbf{h}_{t-1}, \mathbf{x}_t; \boldsymbol{\theta}_f) & (1) \\
\mathbf{o}_t = g(\mathbf{h}_t; \boldsymbol{\theta}_g) & (2)
\end{cases}
$$

- f: compute current hidden state
- g: compute current output hidden state
- $\mathbf{x}_t$: input at time t
- $\mathbf{h}_{t-1}$: hidden state before time t
- $\mathbf{o}_t$: output at time t
- $\mathbf{h}_t$: hidden state after time t
- Parameters: $\boldsymbol{\theta}_f$, $\boldsymbol{\theta}_g$



Loss function and gradient

- The loss over the whole sequence can be written

$$\mathcal{L} = \frac{1}{T}\sum_{t=1}^{T} \ell(\mathbf{o}_t, \boldsymbol{y}_t) \tag{3}$$

- For parameters $\boldsymbol{\theta}_g$ in function g

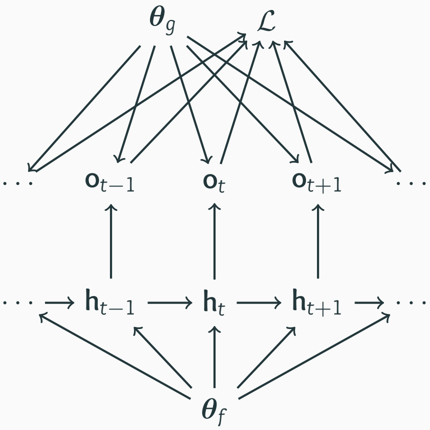
$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}_g} = \sum_{t=1}^{T} \frac{\partial \mathcal{L}}{\partial \mathbf{o}_t} \frac{\partial \mathbf{o}_t}{\partial \boldsymbol{\theta}_g}$$

  only one dependency!

- For parameters $\boldsymbol{\theta}_f$ in function f

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}_f} = \sum_{t=1}^{T} \frac{\partial \mathcal{L}}{\partial \mathbf{h}_t} \frac{\partial \mathbf{h}_t}{\partial \boldsymbol{\theta}_f}$$

  multiple dependencies...

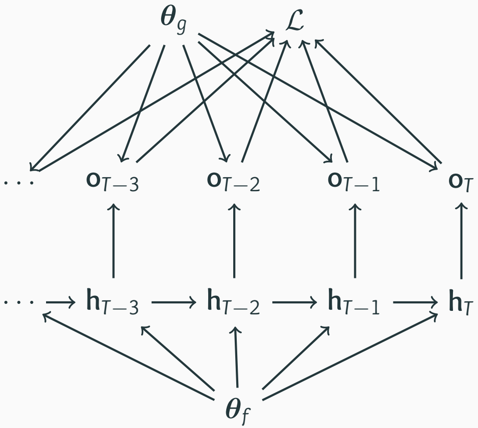


Back propagation for last token

- For last iteration, no extra dependencies

$$\frac{\partial \mathcal{L}}{\partial \mathbf{h}_T} = \frac{\partial \mathcal{L}}{\partial \mathbf{o}_T} \frac{\partial \mathbf{o}_T}{\partial \mathbf{h}_T}$$

- Easy from equations (2) and (3)

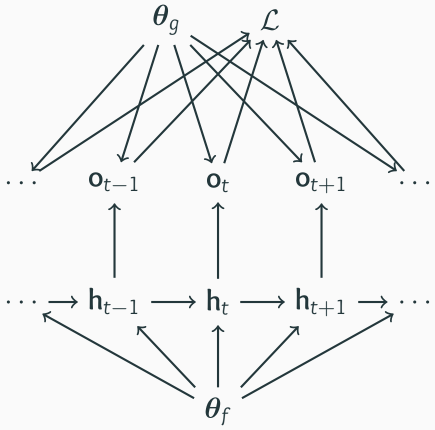


Back propagation

- If $t < T$, we have

$$\frac{\partial \mathcal{L}}{\partial \mathbf{h}_t} = \frac{\partial \mathcal{L}}{\partial \mathbf{h}_{t+1}} \frac{\partial \mathbf{h}_{t+1}}{\partial \mathbf{h}_t} + \frac{\partial \mathcal{L}}{\partial \mathbf{o}_t} \frac{\partial \mathbf{o}_t}{\partial \mathbf{h}_t} \qquad (4)$$

- Recurrent relation giving $\frac{\partial \mathcal{L}}{\partial \mathbf{h}_t}$ from $\frac{\partial \mathcal{L}}{\partial \mathbf{h}_{t+1}}$



Recurrent neural network

- The output $\mathbf{o}_t$ is the hidden state, we choose

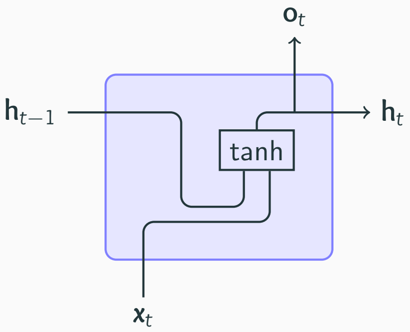
$$g(x) = x$$

- f is implemented as:
  - a linear transform of $\mathbf{x}_t$ and $\mathbf{h}_{t-1}$
  - a bias $\boldsymbol{b}$
  - an entrywise non-linearity $\phi = \tanh$

  We have

$$\mathbf{h}_t = \tanh\left(W_i \mathbf{x}_t + W_h \mathbf{h}_{t-1} + \boldsymbol{b}\right)$$

- f is parametric with parameters W_i, W_h and $\boldsymbol{b}$.



Unfolding $\frac{\partial \mathcal{L}}{\partial \mathbf{h}_t}$

Starting from the recurrent relation (4)

$$\frac{\partial \mathcal{L}}{\partial \mathbf{h}_t} = \frac{\partial \mathcal{L}}{\partial \mathbf{h}_{t+1}} \frac{\partial \mathbf{h}_{t+1}}{\partial \mathbf{h}_t} + \frac{\partial \mathcal{L}}{\partial \mathbf{o}_t} \frac{\partial \mathbf{o}_t}{\partial \mathbf{h}_t}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{h}_t} = \left(\frac{\partial \mathcal{L}}{\partial \mathbf{h}_{t+2}} \frac{\partial \mathbf{h}_{t+2}}{\partial \mathbf{h}_{t+1}} + \frac{\partial \mathcal{L}}{\partial \mathbf{o}_{t+1}} \frac{\partial \mathbf{o}_{t+1}}{\partial \mathbf{h}_{t+1}} \right) \frac{\partial \mathbf{h}_{t+1}}{\partial \mathbf{h}_t} + \frac{\partial \mathcal{L}}{\partial \mathbf{o}_t} \frac{\partial \mathbf{o}_t}{\partial \mathbf{h}_t}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{h}_t} = \left(\left(\frac{\partial \mathcal{L}}{\partial \mathbf{h}_{t+3}} \frac{\partial \mathbf{h}_{t+3}}{\partial \mathbf{h}_{t+2}} + \frac{\partial \mathcal{L}}{\partial \mathbf{o}_{t+2}} \frac{\partial \mathbf{o}_{t+2}}{\partial \mathbf{h}_{t+2}} \right) \frac{\partial \mathbf{h}_{t+2}}{\partial \mathbf{h}_{t+1}} + \frac{\partial \mathcal{L}}{\partial \mathbf{o}_{t+1}} \frac{\partial \mathbf{o}_{t+1}}{\partial \mathbf{h}_{t+1}} \right) \frac{\partial \mathbf{h}_{t+1}}{\partial \mathbf{h}_t} + \frac{\partial \mathcal{L}}{\partial \mathbf{o}_t} \frac{\partial \mathbf{o}_t}{\partial \mathbf{h}_t}$$

$$\vdots$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{h}_t} = \sum_{k=t}^{T} \frac{\partial \mathcal{L}}{\partial \mathbf{o}_k} \frac{\partial \mathbf{o}_k}{\partial \mathbf{h}_k} \prod_{i=t}^{k-1} \frac{\partial \mathbf{h}_{i+1}}{\partial \mathbf{h}_i}$$

Vanishing gradient/ gradient explosion

- Product of $\mathcal{O}(T)$ factors, parameters in red product only

$$\frac{\partial \mathcal{L}}{\partial \mathbf{h}_t} = \sum_{k=t}^{T} \frac{\partial \mathcal{L}}{\partial \mathbf{o}_k} \frac{\partial \mathbf{o}_k}{\partial \mathbf{h}_k} \prod_{i=t}^{k-1} \frac{\partial \mathbf{h}_{i+1}}{\partial \mathbf{h}_i}$$

- Let $\mathbf{z}_t = W_i \mathbf{x}_t + W_h \mathbf{h}_{t-1} + \boldsymbol{b}$, from $\mathbf{h}_t = \tanh\left(W_i \mathbf{x}_t + W_h \mathbf{h}_{t-1} + \boldsymbol{b}\right)$ we have

$$\frac{\partial \mathbf{h}_{i+1}}{\partial \mathbf{h}_i} = \operatorname{diag}\left(\tanh'\left(\mathbf{z}_i\right)\right) W_h$$

- So that we have

$$\begin{aligned} \left\| \frac{\partial \mathbf{h}_{i+1}}{\partial \mathbf{h}_i} \right\|_2 &\leqslant \left\| \operatorname{diag}\left(\tanh'\left(\mathbf{z}_i\right)\right) \right\|_2 \lambda_{\max}(W_h) \\ &\leqslant \lambda_{\max}(W_h) && \text{(greatest eigenvalue in magnitude)} \end{aligned}$$

Vanishing/exploding gradient

- **Vanishing gradient** when $\left\|\prod_{i=t}^{k-1} \frac{\partial \mathbf{h}_{i+1}}{\partial \mathbf{h}_i}\right\| \longrightarrow 0$

  If $\lambda_{\max}(W_h) < 1$

$$\left\|\prod_{i=t}^{k-1} \frac{\partial \mathbf{h}_{i+1}}{\partial \mathbf{h}_i}\right\| \leqslant \lambda_{\max}(W_h)^{k-t} \longrightarrow 0 \quad \text{as} \quad k-t \longrightarrow +\infty$$

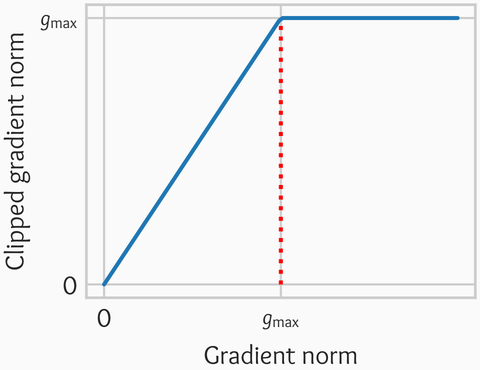
  - $\frac{\partial \mathcal{L}}{\partial \mathbf{h}_t}$ is practically independent from $\mathbf{x}_k$ with $k \gg t$
  - Slow learning or no learning at all
  - Fails to learn long-term dependencies
- **Exploding gradient** when $\left\|\prod_{i=t}^{k-1} \frac{\partial \mathbf{h}_{i+1}}{\partial \mathbf{h}_i}\right\| \longrightarrow +\infty$

  Overflow error: Nans everywhere...

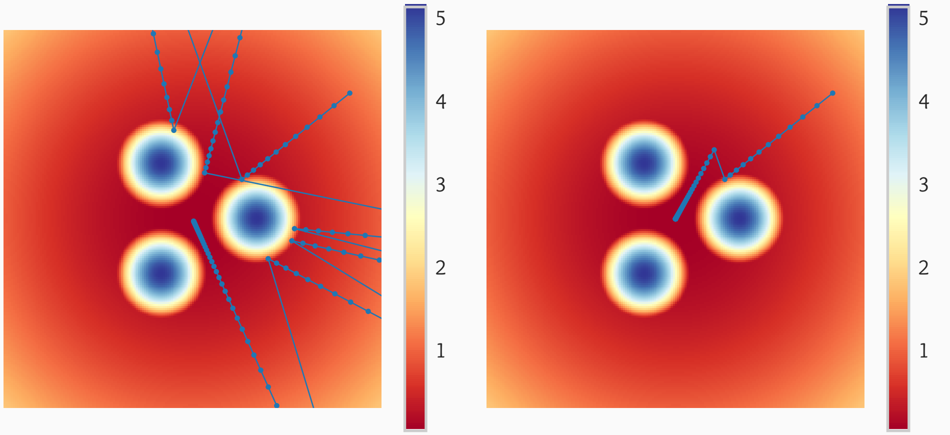
Gradient clipping

$$\boldsymbol{g}_{\text{clipped}} = \min\left(g_{\max}, \|\boldsymbol{g}\|\right) \cdot \frac{\boldsymbol{g}}{\|\boldsymbol{g}\|}$$

$$\boldsymbol{g}_{\text{clipped}} = \begin{cases} \boldsymbol{g} & \text{if } \|\boldsymbol{g}\| \leqslant g_{\max} \\ g_{\max} \cdot \dfrac{\boldsymbol{g}}{\|\boldsymbol{g}\|} & \text{otherwise} \end{cases}$$



Gradient clipping: illustrations



(a) Without gradient clipping

(b) With gradient clipping

Modern recurrent neural networks

- Vanilla RNN shortcomings:
  - RNN suffers from numerical instability
  - Unable to learn long-term dependencies
- Ideas
  - Change the structure of recurrent cell
  - Introduce regulating gates
- Alternatives
  - Long Short-Term Memory (LSTM), Hochreiter and Schmidhuber 1997
  - Gated Recurrent Unit (GRU), Cho et al. 2014

Key idea

- Given that $\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}}$ is a problem
- Why not adding a regulation mechanism such that $\mathbf{h}_t = \mathbf{h}_{t-1}$?
- Add a gate $\mathbf{u}_t \in (0, 1)$:

$$\mathbf{h}_t = \mathbf{h}_{t-1} + \mathbf{u}_t \cdot \tilde{\mathbf{h}}_t$$

- How to decide when $\mathbf{u}_t$ should be zero?
- Learn it as well!

$$\mathbf{u}_t = \sigma(U\mathbf{x}_t + W\mathbf{h}_{t-1} + \boldsymbol{b})$$

Gated Recurrent Unit

Gated Recurrent Unit GRU

- Reset gate

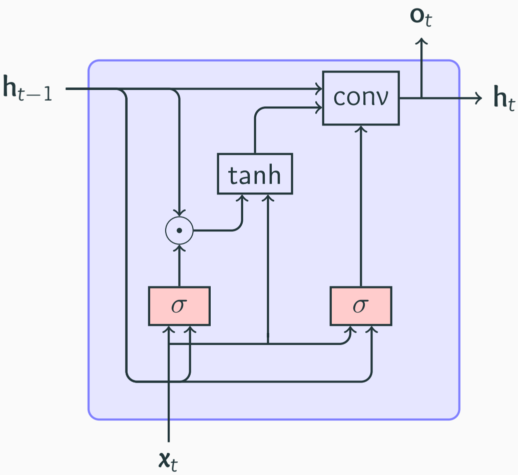
$$\mathbf{r}_t = \sigma(U_r\mathbf{x}_t + W_r\mathbf{h}_{t-1} + \boldsymbol{b}_r)$$

- Candidate hidden

$$\tilde{\mathbf{h}}_t = \tanh\left(U_c\mathbf{x}_t + W_c(\mathbf{r}_t \odot \mathbf{h}_{t-1}) + \boldsymbol{b}_c\right)$$

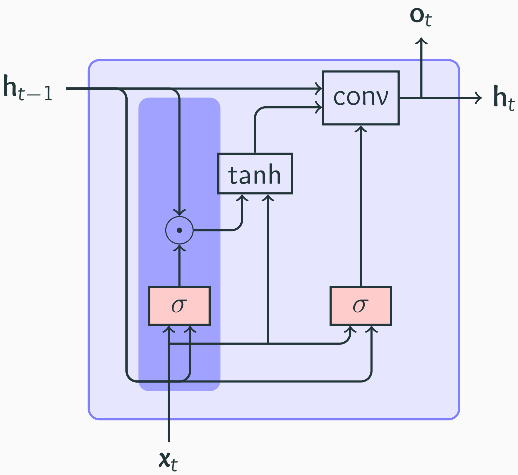
- Update gate

$$\mathbf{u}_t = \sigma(U_u\mathbf{x}_t + W_u\mathbf{h}_{t-1} + \boldsymbol{b}_u)$$
$$\mathbf{h}_t = (1 - \mathbf{u}_t) \odot \mathbf{h}_{t-1} + \mathbf{u}_t \odot \tilde{\mathbf{h}}_t$$



Reset gate

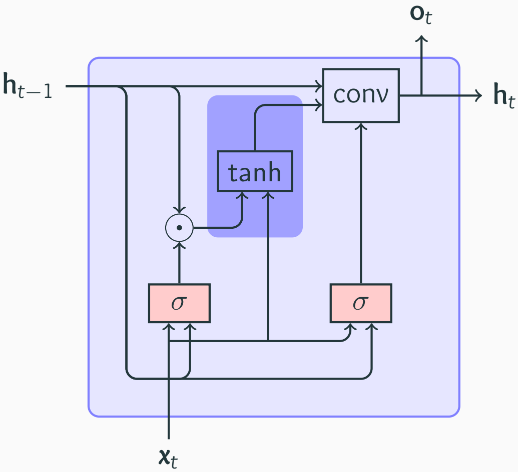
- Linear transformation of $\mathbf{h}_{t-1}$ and $\mathbf{x}_t$ followed by a sigmoid
$$\mathbf{r}_t = \sigma(U_r \mathbf{x}_t + W_r \mathbf{h}_{t-1} + \boldsymbol{b}_r)$$
- All entries of $\mathbf{r}_t$ are in $[0, 1]$, can be used as a ratio to reset $\mathbf{h}_{h-1}$
- $\mathbf{r}_t$ is used to reset $\mathbf{h}_{t-1}$: $\mathbf{r}_t \odot \mathbf{h}_{t-1}$
- $\mathbf{r}_t \odot \mathbf{h}_{t-1}$ is used instead of $\mathbf{h}_{t-1}$
    - If $(\mathbf{r}_t)_i = 1$, no change:
    $$(\mathbf{r}_t \odot \mathbf{h}_{t-1})_i = (\mathbf{h}_{t-1})_i$$
    - If $(\mathbf{r}_t)_i = 0$,
    $$(\mathbf{r}_t \odot \mathbf{h}_{t-1})_i = 0$$



New memory

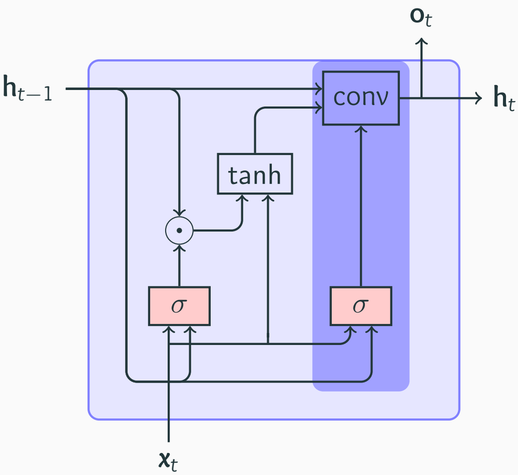
- Basic RNN unit except that
  - $\mathbf{r}_t \odot \mathbf{h}_{t-1}$ is used instead of $\mathbf{h}_{t-1}$

$$\tilde{\mathbf{h}}_t = \tanh\left(U_c \mathbf{x}_t + W_c(\mathbf{r}_t \odot \mathbf{h}_{t-1}) + \mathbf{b}_c\right)$$



Update gate

- Update gate
$$\mathbf{u}_t = \sigma(U_u \mathbf{x}_t + W_u \mathbf{h}_{t-1} + \boldsymbol{b}_u)$$
- Same as reset gate with own parameters
- Convex combination of $\mathbf{h}_{t-1}$ and $\tilde{\mathbf{h}}_{t-1}$ controlled be $\mathbf{u}_t$
$$\mathbf{h}_t = (1 - \mathbf{u}_t) \odot \mathbf{h}_{t-1} + \mathbf{u}_t \odot \tilde{\mathbf{h}}_t$$
  - If $(\mathbf{u}_t)_i = 1$, $(\mathbf{h}_t)_i = \left(\tilde{\mathbf{h}}_t\right)_i$
  - If $(\mathbf{u}_t)_i = 0$, $(\mathbf{h}_t)_i = (\mathbf{h}_{t-1})_i$



Summary

- Equations

$$\mathbf{r}_t = \sigma(U_r \mathbf{x}_t + W_r \mathbf{h}_{t-1} + \boldsymbol{b}_r)$$

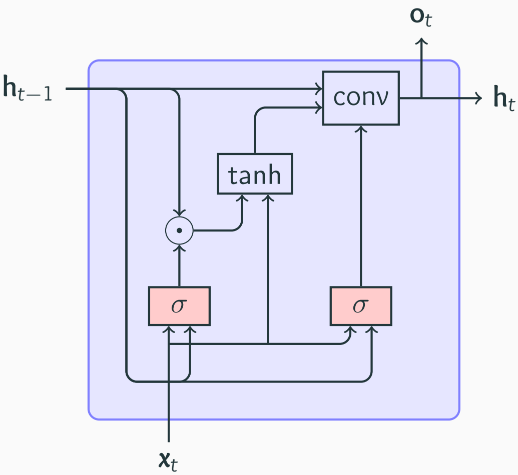
$$\tilde{\mathbf{h}}_t = \tanh\left(U_c \mathbf{x}_t + W_c(\mathbf{r}_t \odot \mathbf{h}_{t-1}) + \boldsymbol{b}_c\right)$$

$$\mathbf{u}_t = \sigma(U_u \mathbf{x}_t + W_u \mathbf{h}_{t-1} + \boldsymbol{b}_u)$$

$$\mathbf{h}_t = (1 - \mathbf{u}_t) \odot \mathbf{h}_{t-1} + \mathbf{u}_t \odot \tilde{\mathbf{h}}_t$$

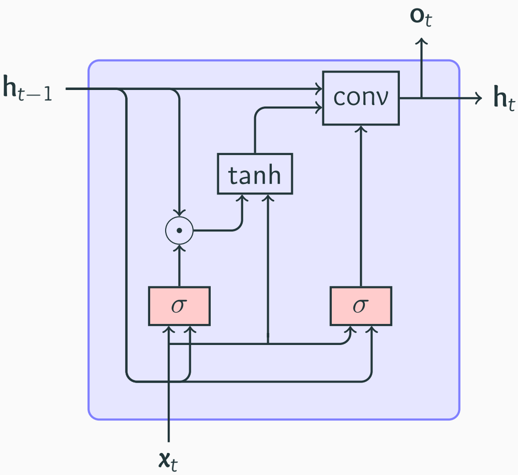
- Behavior

$(\mathbf{r}_t)_i$	$(\mathbf{u}_t)_i$	Result
1	1	Regular RNN update
0	1	Reset hidden
–	0	Keep hidden



Summary

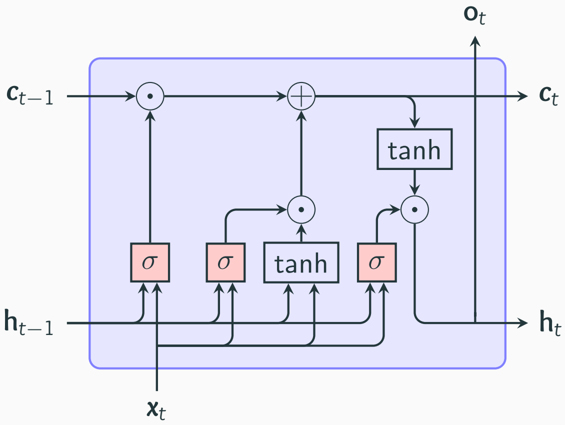
- Input of size d: $x_t \in \mathbb{R}^d$
- Hidden state of size h: $h_t \in \mathbb{R}^h$
- $U_r, U_c, U_u \in \mathbb{R}^{h \times d}$
- $W_r, W_c, W_u \in \mathbb{R}^{h \times h}$
- $\boldsymbol{b}_r, \boldsymbol{b}_c, \boldsymbol{b}_u \in \mathbb{R}^h$
- Number of parameters: $3h(d + h + 1)$



Long Short-Term Memory (LSTM) networks

LSTM

- LSTM has 3 gates
- The hidden state is split
    - a cell state $\mathbf{c}_t$
    - a real hidden state $\mathbf{h}_t$



Forget gate

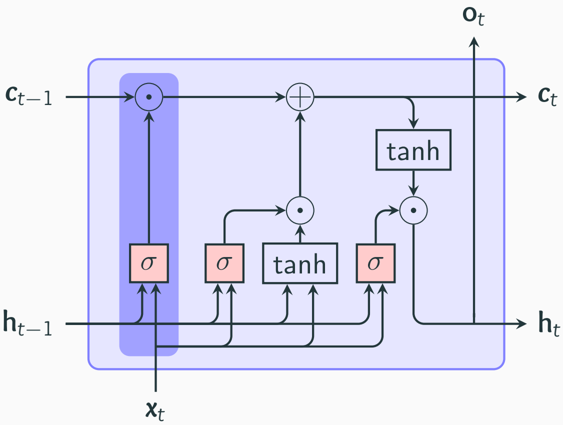
- Decides what to forget in $\mathbf{c}_{t-1}$

$$\mathbf{f}_t = \sigma\left(U^f \mathbf{x}_t + W^f \mathbf{h}_{t-1} + \mathbf{b}^f\right)$$

- Applied to cell state

$$\mathbf{f}_t \odot \mathbf{c}_{t-1}$$

- Parameters: U^f, W^f and $\mathbf{b}^f$

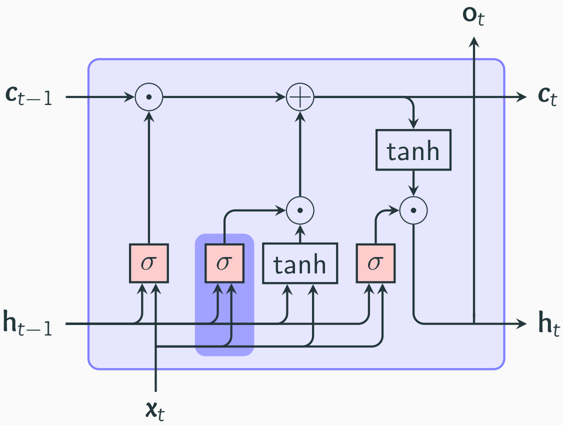


Input gate

- Used to compute the new cell state

$$\mathbf{i}_t = \sigma\left(U^i \mathbf{x}_t + W^i \mathbf{h}_{t-1} + \boldsymbol{b}^i\right)$$

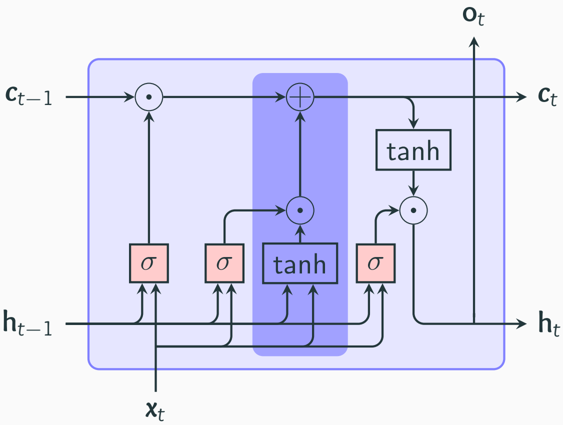
- Parameters: U^i, W^i and $\boldsymbol{b}^i$



New cell state

- Basic RNN unit
  - $\tilde{\mathbf{c}}_t = \tanh\left(U^c \mathbf{x}_t + W^c \mathbf{h}_{t-1} + \boldsymbol{b}^c\right)$
  - Parameters U^c, W^c and $\boldsymbol{b}^c$
- New cell state using forget and input gates

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_t$$

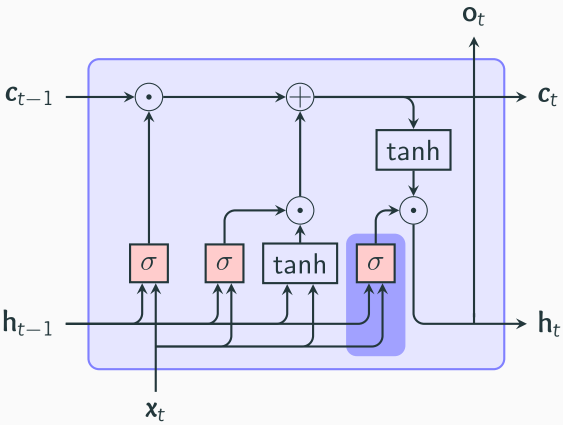


Output gate

- Used to compute new hidden statement

$$\boldsymbol{g}_t = \sigma\left(U^o \mathbf{x}_t + W^o \mathbf{h}_{t-1} + \boldsymbol{b}^o\right)$$

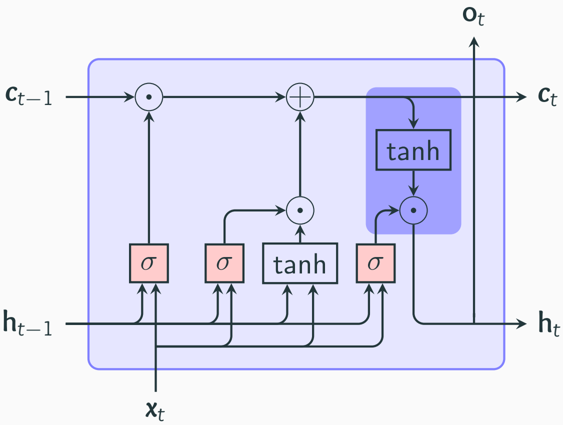
- Parameters: U^o, W^o and $\boldsymbol{b}^o$



New hidden state

Hidden state controled by the output gate

- $\mathbf{h}_t = \boldsymbol{g}_t \odot \tanh \mathbf{c}_t$



Equations

- Gates

$$\mathbf{f}_t = \sigma\left(U^f \mathbf{x}_t + W^f \mathbf{h}_{t-1} + \boldsymbol{b}^f\right)$$

$$\mathbf{i}_t = \sigma\left(U^i \mathbf{x}_t + W^i \mathbf{h}_{t-1} + \boldsymbol{b}^i\right)$$

$$\boldsymbol{g}_t = \sigma\left(U^o \mathbf{x}_t + W^o \mathbf{h}_{t-1} + \boldsymbol{b}^o\right)$$

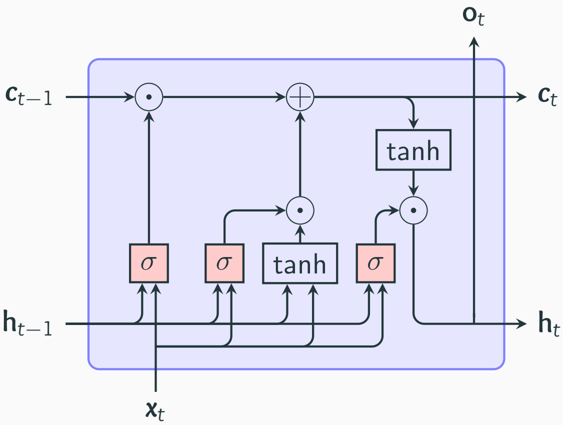
- RNN cell

$$\tilde{\mathbf{c}}_t = \tanh\left(U^c \mathbf{x}_t + W^c \mathbf{h}_{t-1} + \boldsymbol{b}^c\right)$$

- Outputs

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_t$$

$$\mathbf{h}_t = \boldsymbol{g}_t \odot \tanh \mathbf{c}_t$$



Summary

- Equations

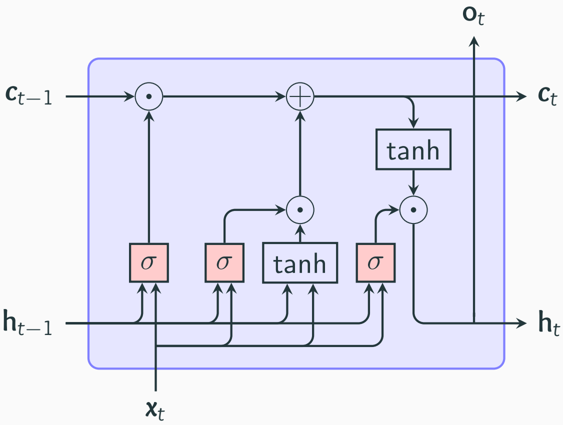
$$\tilde{\mathbf{c}}_t = \tanh\left(U^c \mathbf{x}_t + W^c \mathbf{h}_{t-1} + \boldsymbol{b}^c\right)$$

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_t$$

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh \mathbf{c}_t$$

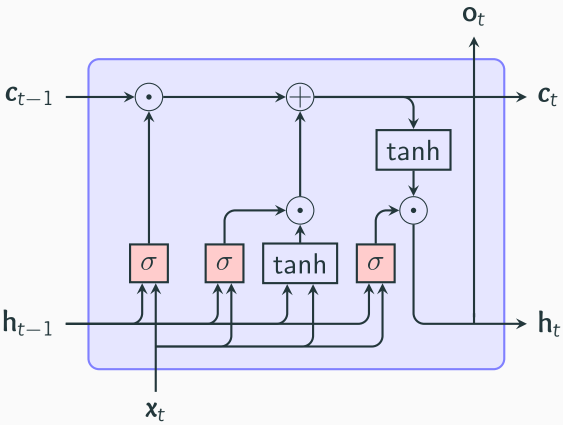
- Behavior

$\mathbf{f}_t$	$\mathbf{i}_t$	Result
0	0	Erase the state
0	1	Overwrite the state
1	0	Keep the state
1	1	Add to current state



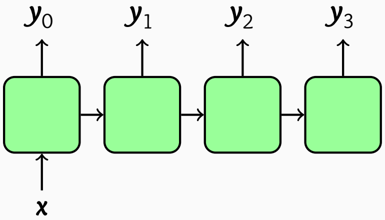
Summary

- Input of size d: $x_t \in \mathbb{R}^d$
- Hidden state of size h: $h_t \in \mathbb{R}^h$
- $U_f, U_i, U_o, U_c \in \mathbb{R}^{h \times d}$
- $W_f, W_i, W_o, W_c \in \mathbb{R}^{h \times h}$
- $\boldsymbol{b}_f, \boldsymbol{b}_i, \boldsymbol{b}_o, \boldsymbol{b}_c \in \mathbb{R}^h$
- Number of parameters: $4h(d + h + 1)$



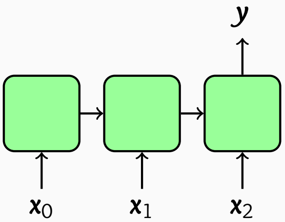
Many-to-one and One-to-many

One-to-many: a vector to a sequence



- Image captioning: An image is given a description of it

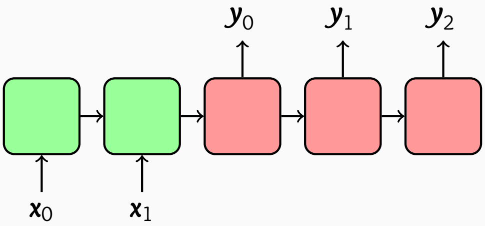
Many-to-one: a sequence to a class/score



- Sentiment analysis: A sequence is given a label or a score

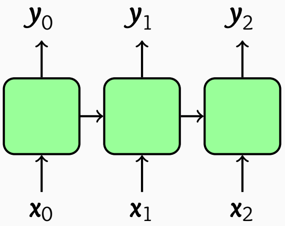
Many-to-many

Many-to-many: a sequence to another sequence



- Machine translation: a text is translated to another language

Many-to-many same length: a sequence to another sequence of same length



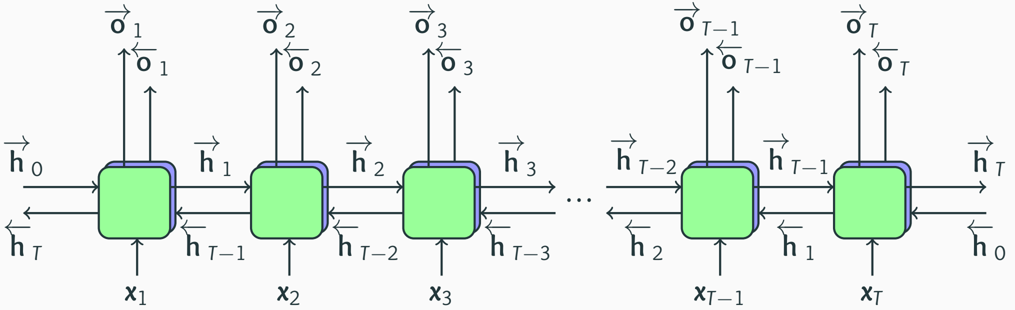
- Part-of-speech tagging: Each word is given a tag

Bidirectional RNN: motivation

- Influence of one token is exponentially decreasing as move away
- Hard to remember first token of input sequence
- Why not learn on the sequence itself and on its reverse?

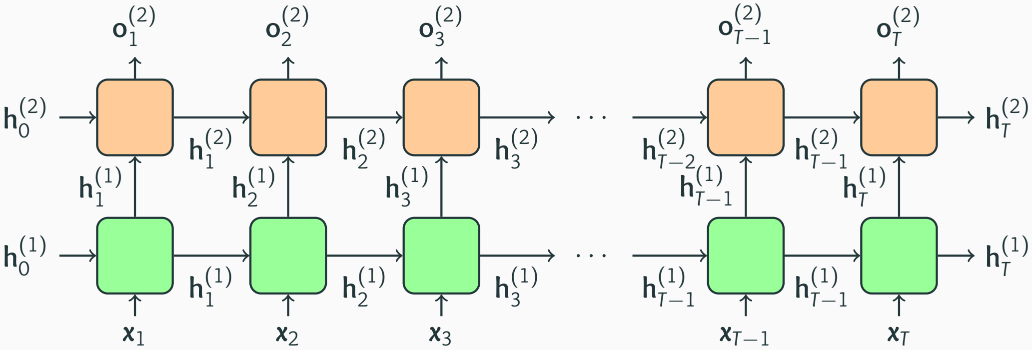
Bidirectional RNN

- Two independent RNNs:
  - A regular one (in green)
  - A reversed one (in blue), on the sequence $\mathbf{x}_T, \ldots, \mathbf{x}_1$
  - Two hidden state initialization vectors: $\overrightarrow{\mathbf{h}}_0$ and $\overleftarrow{\mathbf{h}}_0$
  - Two hidden states: $\overrightarrow{\mathbf{h}}_t$ and $\overleftarrow{\mathbf{h}}_t$ for past and future representation



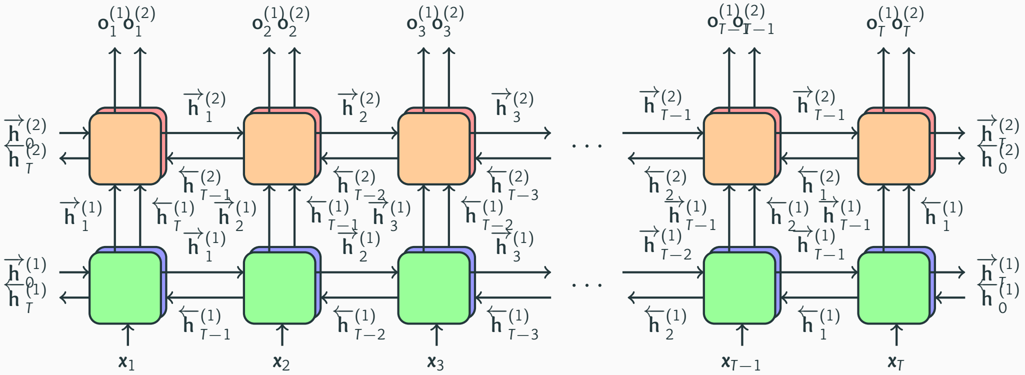
Stacked RNN

- Give hidden state of one RNN as input for another RNN



Stacked bidirectional RNN

- Give hidden state of one RNN as input to respective RNN



References i

[1] Sepp Hochreiter and Jürgen Schmidhuber. **"Long short-term memory".** In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[2] Kyunghyun Cho et al. **"On the properties of neural machine translation: Encoder-decoder approaches".** 2014. arXiv: 1409.1259.