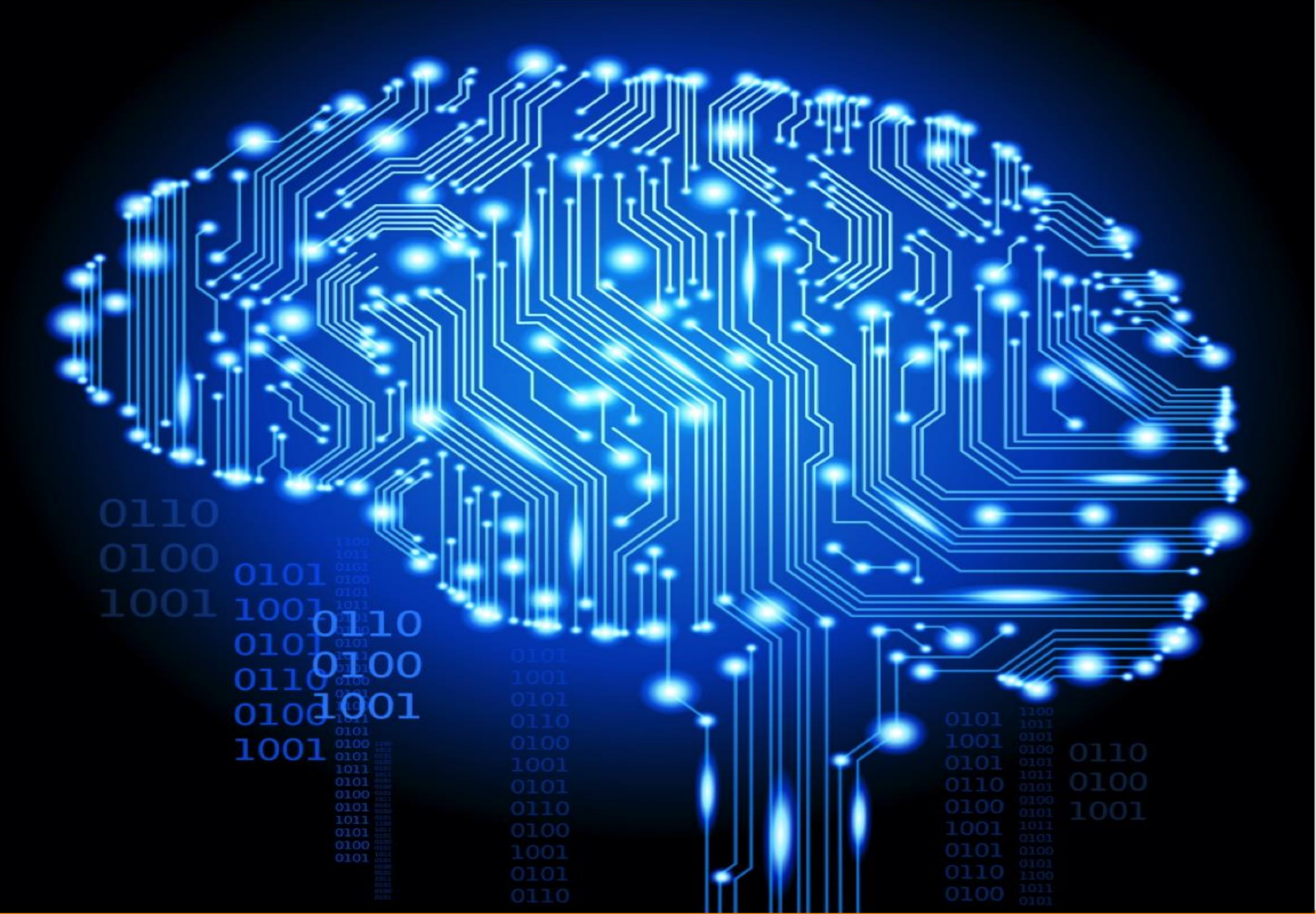


Lecture 8: Recurrent Neural Networks

Deep Learning @ UvA

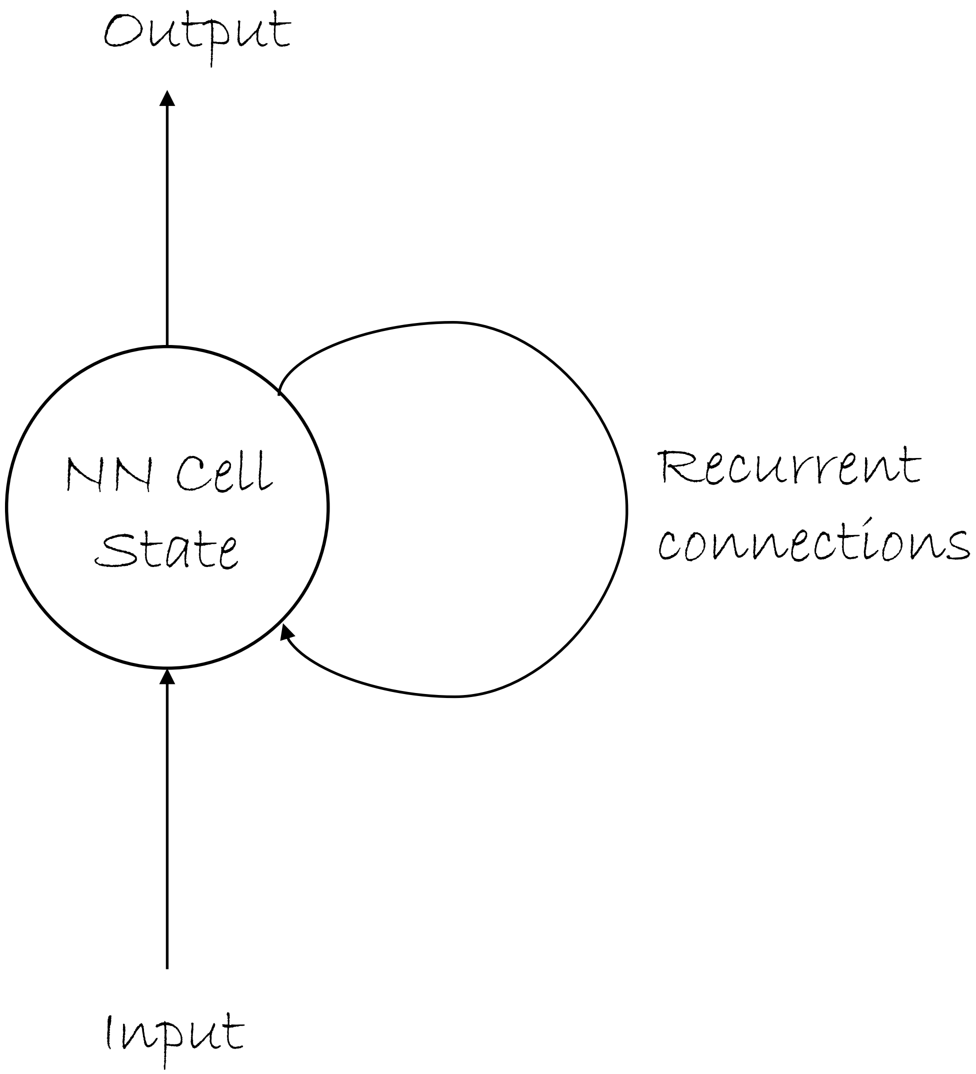
Previous Lecture

- Word and Language Representations
- From n-grams to Neural Networks
- Word2vec
- Skip-gram

Lecture Overview

- Recurrent Neural Networks (RNN) for sequences
- Backpropagation Through Time
- RNNs using Long Short-Term Memory (LSTM)
- Applications of Recurrent Neural Networks

Recurrent Neural Networks



Sequences

- Next data depend on previous data
- Roughly equivalent to predicting what comes next

$$\Pr(x) = \prod_i \Pr(x_i | x_1, \dots, x_{i-1})$$

What

Sequences

- Next data depend on previous data
- Roughly equivalent to predicting what comes next

$$\Pr(x) = \prod_i \Pr(x_i \mid x_1, \dots, x_{i-1})$$

What about

Sequences

- Next data depend on previous data
- Roughly equivalent to predicting what comes next

$$\Pr(x) = \prod_i \Pr(x_i | x_1, \dots, x_{i-1})$$

What about inputs

Sequences

- Next data depend on previous data
- Roughly equivalent to predicting what comes next

$$\Pr(x) = \prod_i \Pr(x_i \mid x_1, \ldots, x_{i-1})$$

What about inputs that appear in sequences, such as text? Could a neural network handle such modalities?

Why sequences?

Why sequences?

- Considering small chunks x_i → fewer parameters, easier modelling
- Generalizes well to arbitrary lengths

$$RecurrentModel(\text{I think, therefore, I am!})$$

$$\equiv$$

$$RecurrentModel(\text{Everything is repeated, in a circle. History is a master because it teaches us that it doesn't exist. It's the permutations that matter.})$$

- However, often we pick a "frame" T instead of an arbitrary length

$$\Pr(x) = \prod_i \Pr(x_i | x_{i-T}, \dots, x_{i-1})$$

What a sequence *really* is?

- Data inside a sequence are non i.i.d.
  - Identically, independently distributed
- The next “word” depends on the previous “words”
  - Ideally on all of them
- We need **context,** and we need **memory!**
- How to model context and memory ?

I am Bond , James

McGuire

Bond

tired

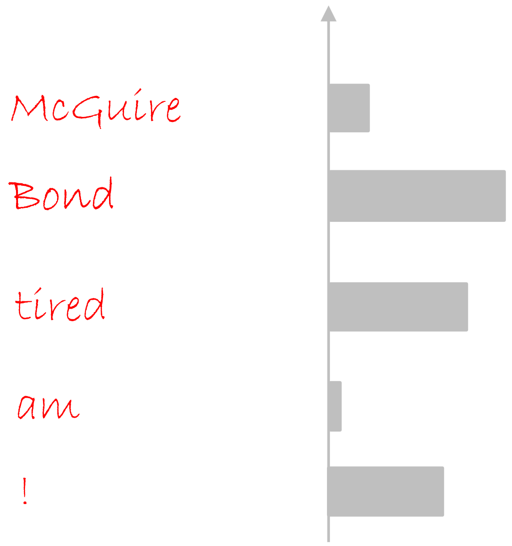
am

!

What a sequence *really* is?

- Data inside a sequence are non i.i.d.
  - Identically, independently distributed
- The next "word" depends on the previous "words"
  - Ideally on all of them
- We need **context,** and we need **memory!**
- How to model context and memory ?

I am Bond , James Bond



$x_i \equiv$ One-hot vectors

- A vector with all zeros except for the active dimension
- 12 words in a sequence → 12 One-hot vectors
- After the one-hot vectors apply an embedding (Word2Vec, GloVE)

Vocabulary	One-hot vectors							
I	I	1	I	0	I	0	I	0
am	am	0	am	1	am	0	am	0
Bond	Bond	0	Bond	0	Bond	1	Bond	0
James	James	0	James	0	James	0	James	1
tired	tired	0	tired	0	tired	0	tired	0
,	,	0	,	0	,	0	,	0
McGuire	McGuire	0	McGuire	0	McGuire	0	McGuire	0
!	!	0	!	0	!	0	!	0

Indices instead of one-hot vectors?

- Can't we simply use indices as features?
- No, great solution, because introduces artificial bias between inputs

$$x_{t=1,2,3,4} = \begin{array}{cccc} \text{I} & \text{am} & \text{James} & \text{McGuire} \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{array}$$

$$q_{t=1,2,3,4} = \begin{array}{cccc} \text{I} & \text{am} & \text{James} & \text{McGuire} \\ 1 & 2 & 4 & 7 \end{array}$$

$$L_2(q_1, q_4) = 3 < L_2(q_1, q_7) = 6$$

Is "I" closer to James than to McGuire?

Memory

- A representation of the past
- Project information at timestep t on a latent space c_t using parameters θ
- Re-using the projected information from t at $t+1$

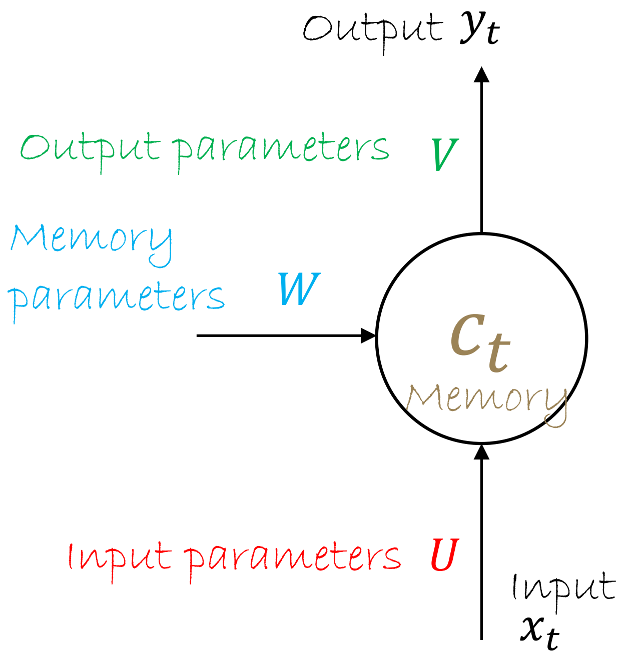
$$c_{t+1} = h(x_{t+1}, c_t; \theta)$$

- Recurrent parameters θ are the shared for all timesteps $t = 0, \dots$

$$c_{t+1} = h(x_{t+1}, h(x_t, h(x_{t-1}, \dots h(x_1, c_0; \theta); \theta); \theta); \theta)$$

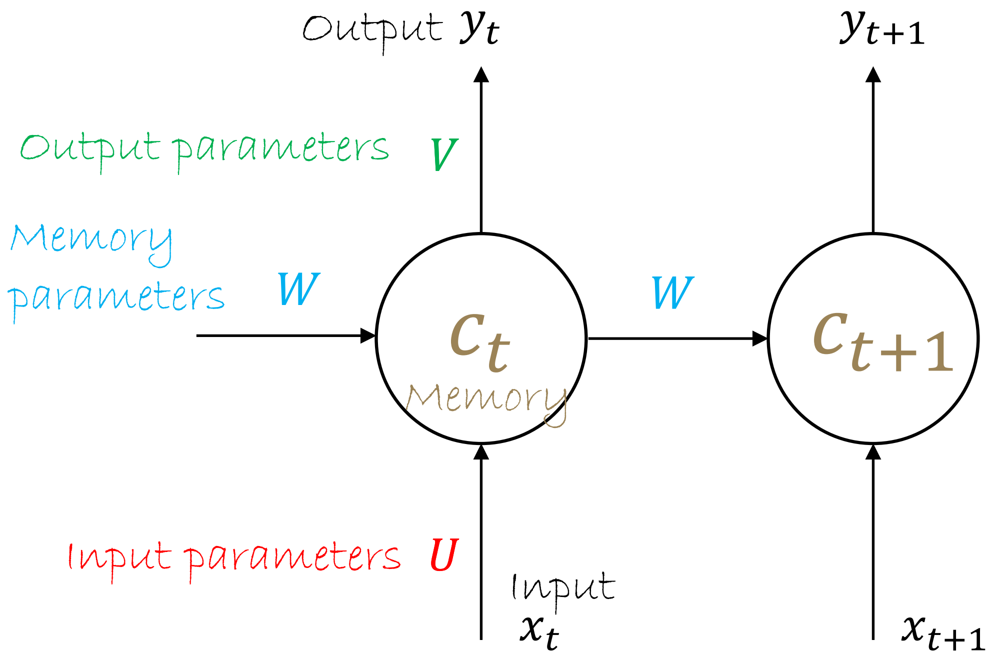
Memory as a Graph

- Simplest model
  - Input with parameters U
  - Memory embedding with parameters W
  - Output with parameters V



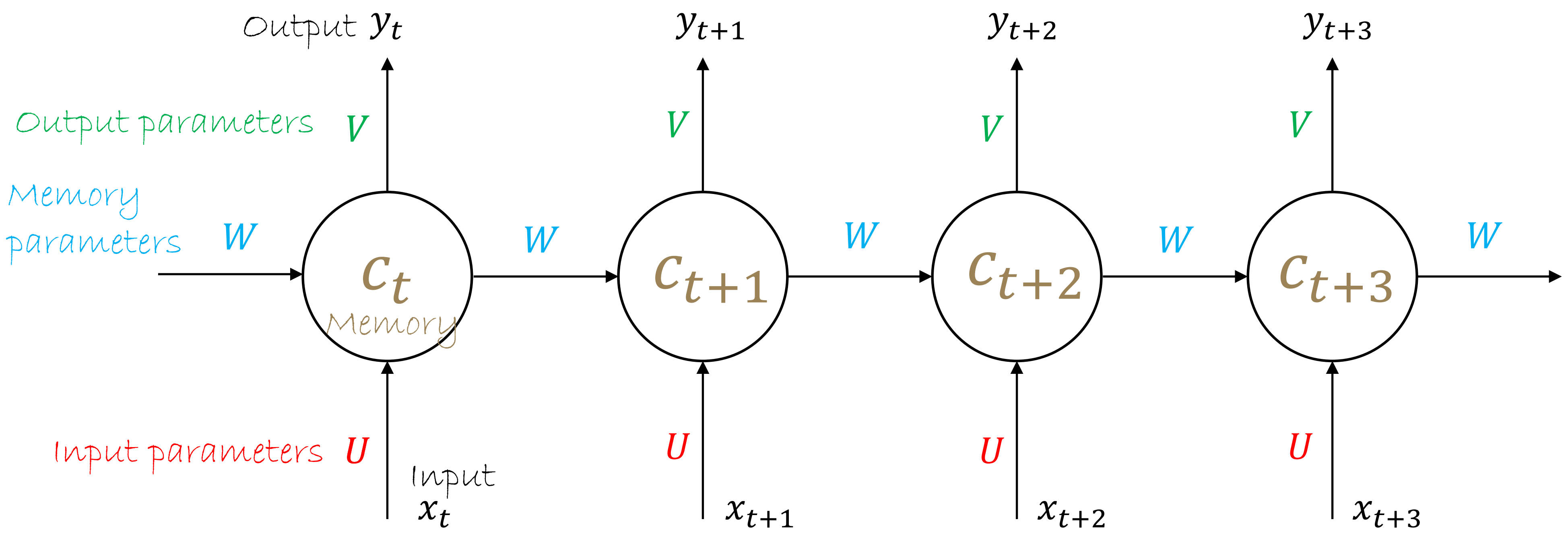
Memory as a Graph

- Simplest model
  - Input with parameters U
  - Memory embedding with parameters W
  - Output with parameters V

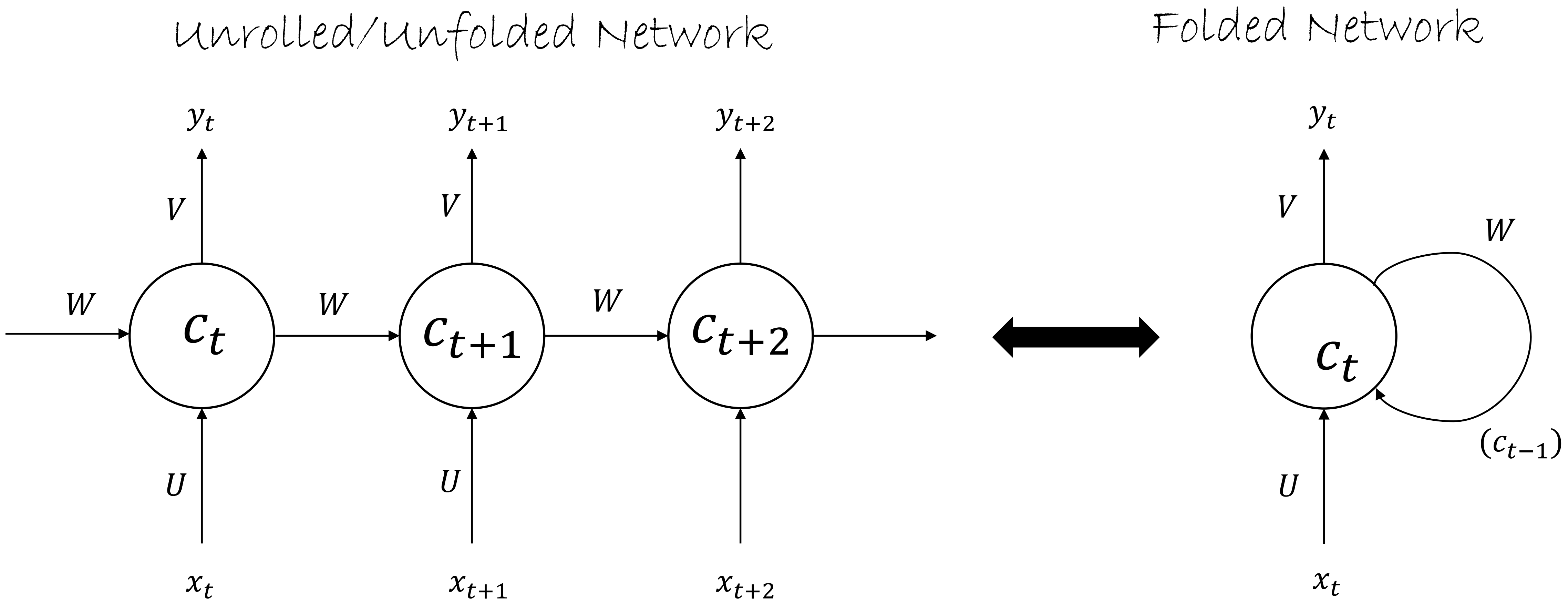


Memory as a Graph

- Simplest model
  - Input with parameters U
  - Memory embedding with parameters W
  - Output with parameters V



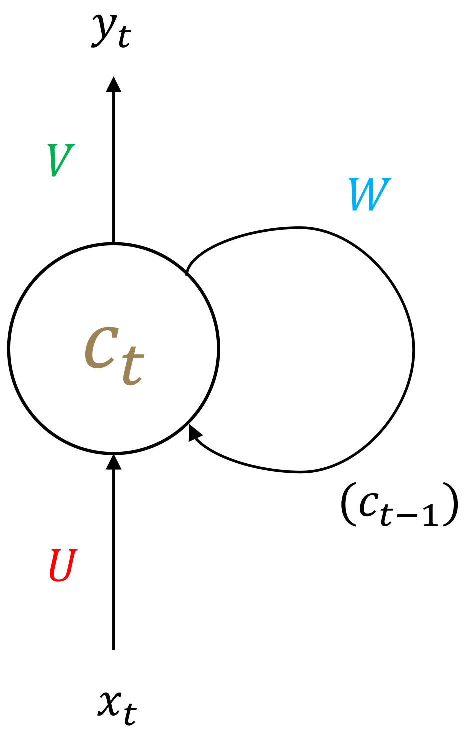
Folding the memory



Recurrent Neural Network (RNN)

- Only <u>**two**</u> equations

$$c_t = \tanh(U\, x_t + W\, c_{t-1})$$
$$y_t = \text{softmax}(V\, c_t)$$



RNN Example

- Vocabulary of 5 words
- A memory of 3 units [Hyperparameter that we choose like layer size]
  - $c_t: [3 \times 1], \mathrm{W}: [3 \times 3]$
- An input projection of 3 dimensions
  - $\mathrm{U}: [3 \times 5]$
- An output projections of 10 dimensions
  - $\mathrm{V}: [10 \times 3]$

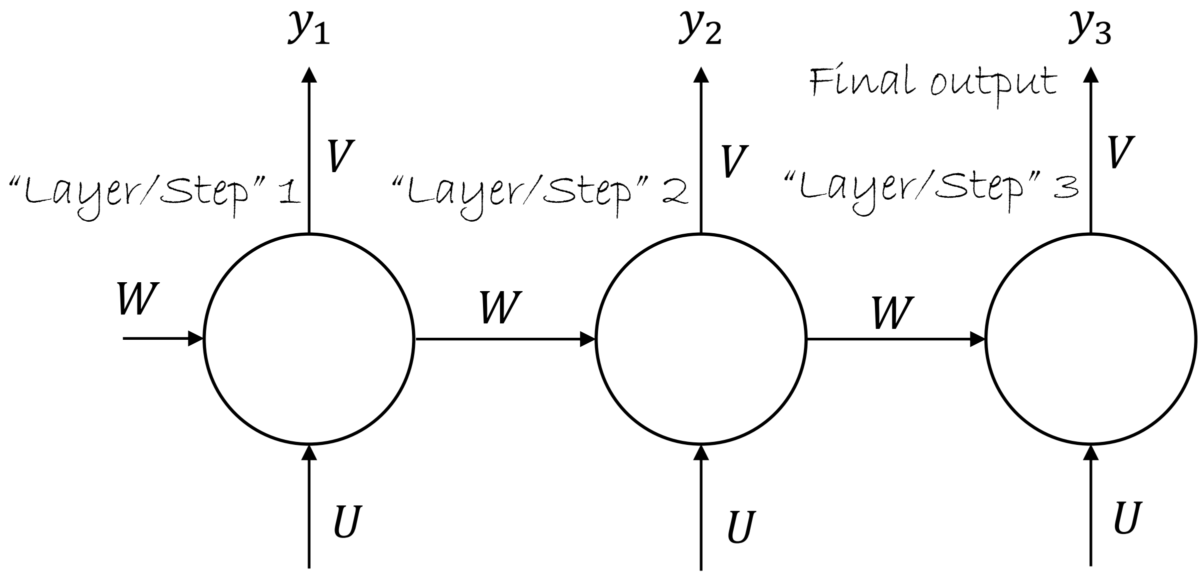
$$c_t = \tanh(U\, x_t + W\, c_{t-1})$$

$$y_t = \mathrm{softmax}(V\, c_t)$$

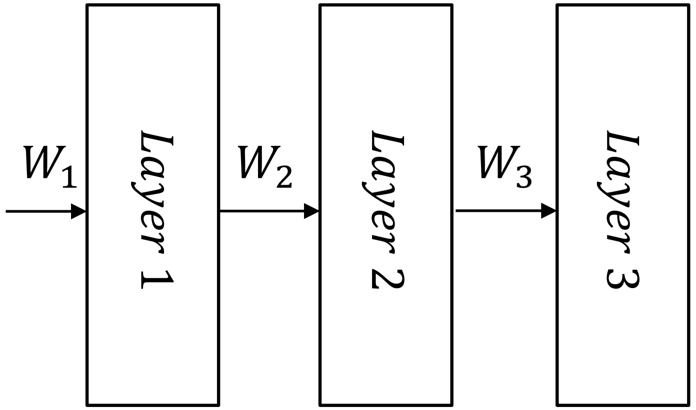
$$U \cdot x_{t=4} = \begin{bmatrix} 0.1 & -0.3 & 1.2 & 0.6 & -0.8 \\ -0.2 & 0.4 & 0.5 & 0.9 & -0.1 \\ -0.1 & 0.2 & -0.7 & -0.8 & 0.3 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0.6 \\ 0.9 \\ -0.8 \end{bmatrix} = U^{(4)}$$

Rolled Network vs. Multi-layer Network?

- What is really different?
  - Steps instead of layers
  - Step parameters shared whereas in a Multi-Layer Network they are different



3-gram Unrolled Recurrent Network

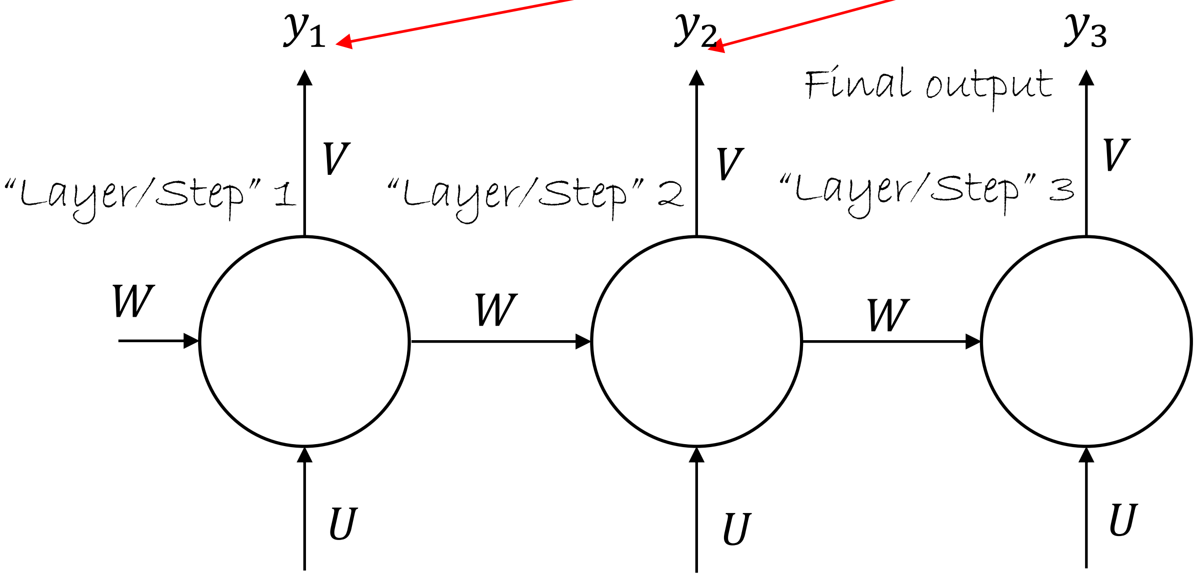


3-layer Neural Network

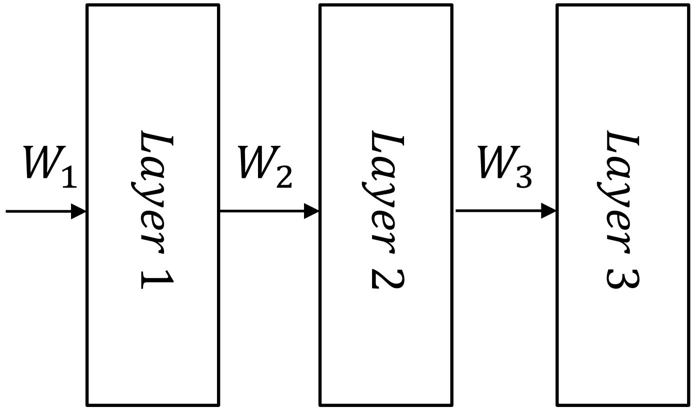
Rolled – Unrolled networks

- Sometimes intermediate outputs are not even needed
- Removing them, we almost end up to a standard Neural Network

- What is really different?
  - Steps instead of layers
  - Step parameters shared whereas in a Multi-Layer Network they are different



3-gram Unrolled Recurrent Network



3-layer Neural Network

Training Recurrent Networks

- Cross-entropy loss

$$P = \prod_{t,k} y_{tk}^{l_{tk}} \quad \Rightarrow \quad \mathcal{L} = -\log P = \sum_t \mathcal{L}_t = -\frac{1}{T}\sum_t l_t \log y_t$$

- Backpropagation Through Time (BPTT)
  - Again, chain rule
  - Only difference: Gradients survive over time steps

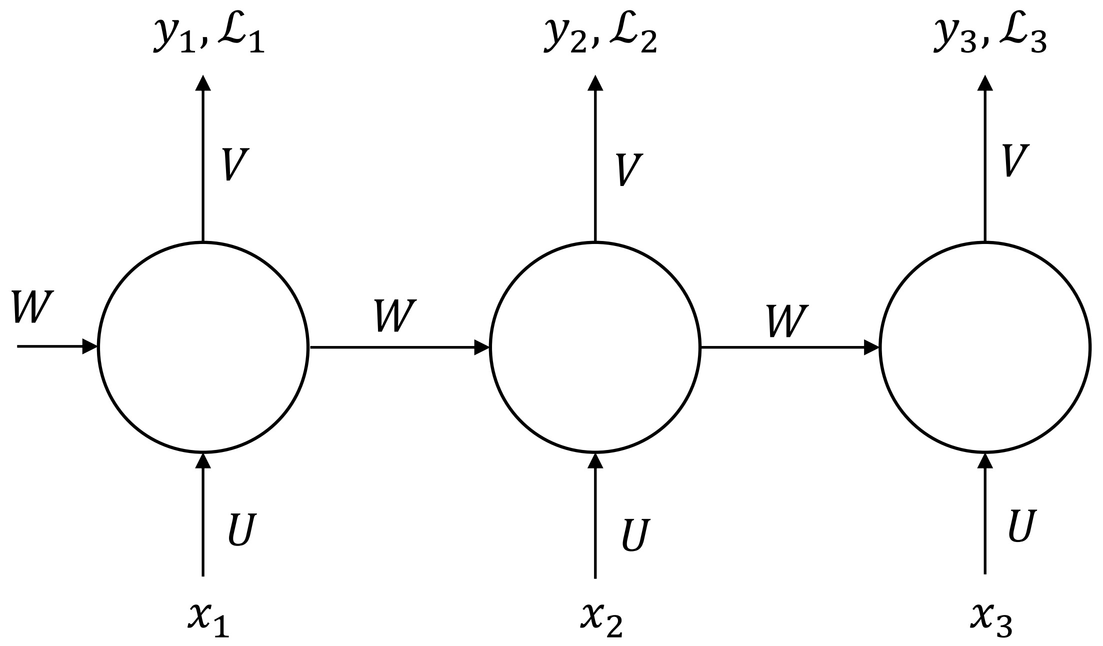
Backpropagation Through Time: An Example

- $\frac{\partial \mathcal{L}}{\partial V}, \frac{\partial \mathcal{L}}{\partial W}, \frac{\partial \mathcal{L}}{\partial U}$
- To make it simpler let's focus on step 3

$$\frac{\partial \mathcal{L}_3}{\partial V}, \frac{\partial \mathcal{L}_3}{\partial W}, \frac{\partial \mathcal{L}_3}{\partial U}$$

Step by step explanation at:
http://www.wildml.com/2015/10/recurrent-neural-networks-tutorial-part-3-backpropagation-through-time-and-vanishing-gradients/

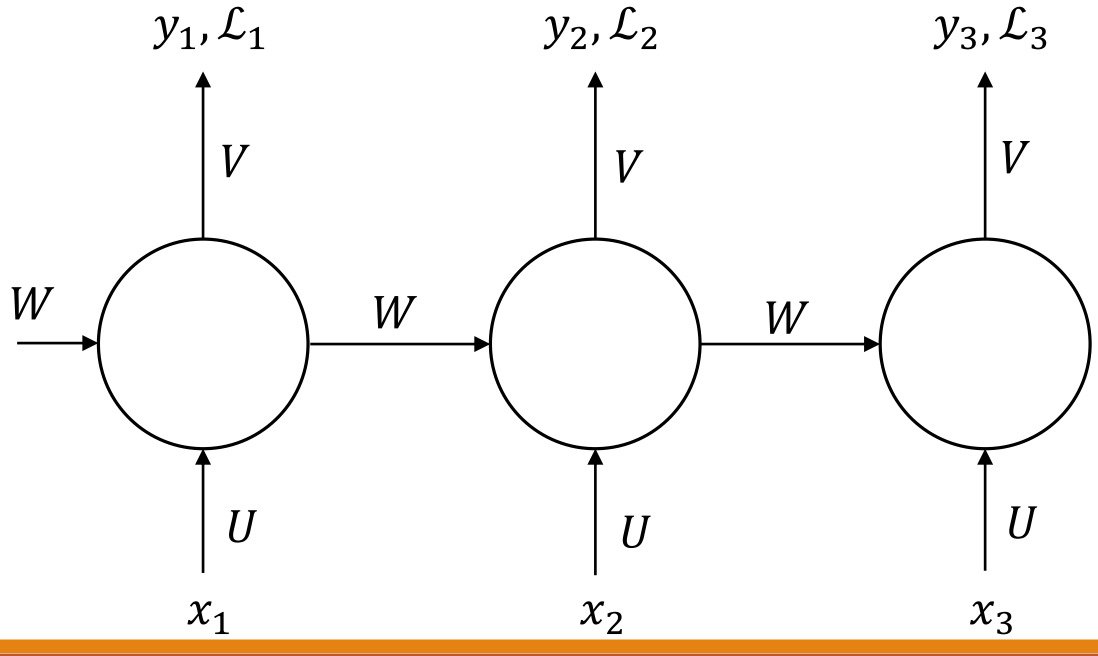
$$\boxed{\begin{aligned} c_t &= \tanh(U\,x_t + W c_{t-1}) \\ y_t &= \mathrm{softmax}(V c_t) \\ \mathcal{L} &= -\sum_t l_t \log y_t = \sum_t \mathcal{L}_t \end{aligned}}$$



Backpropagation Through Time

$$\frac{\partial \mathcal{L}_3}{\partial V} = \frac{\partial \mathcal{L}_3}{\partial y_3}\frac{\partial y_3}{\partial V} = (y_3 - l_3) \cdot c_3$$

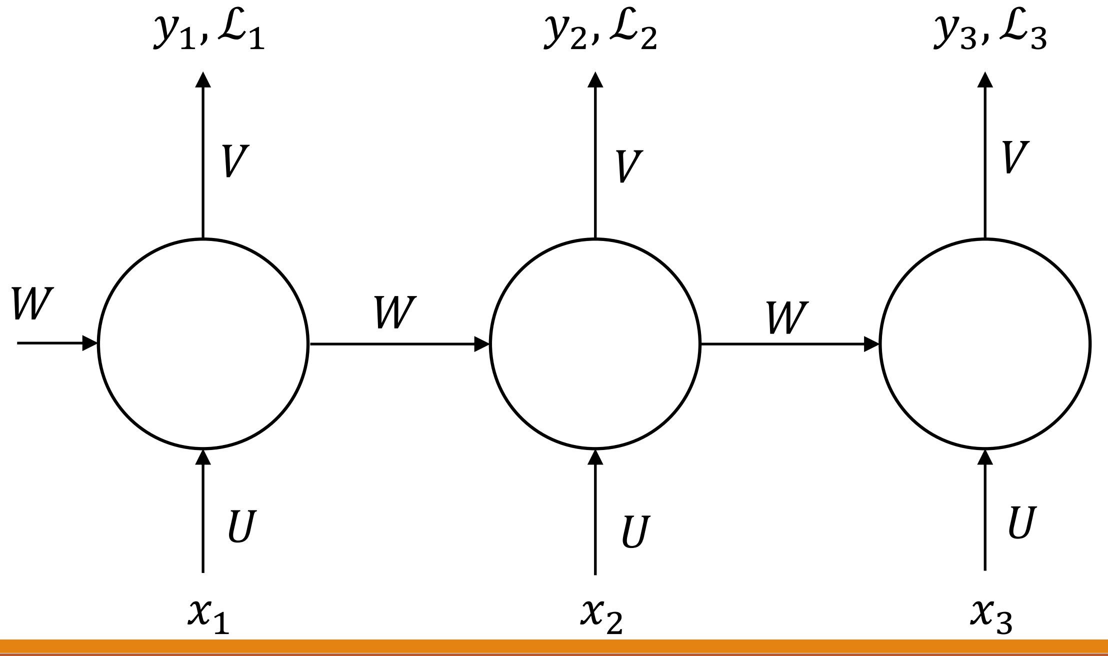
$$\boxed{\begin{aligned} c_t &= \tanh(U\, x_t + W c_{t-1}) \\ y_t &= \mathrm{softmax}(V c_t) \\ \mathcal{L} &= -\sum_t l_t \log y_t = \sum_t \mathcal{L}_t \end{aligned}}$$



Backpropagation Through Time

- $\frac{\partial \mathcal{L}_3}{\partial W} = \frac{\partial \mathcal{L}_3}{\partial y_3} \frac{\partial y_3}{\partial c_3} \frac{\partial c_3}{\partial W}$
- What is the relation between c_3 and W?
  - Two-fold: $c_t = \tanh(U\, x_t + W c_{t-1})$
- $\frac{\partial\, f(\varphi(x), \psi(x))}{\partial x} = \frac{\partial f}{\partial \varphi} \frac{\partial \varphi}{\partial x} + \frac{\partial f}{\partial \psi} \frac{\partial \psi}{\partial x}$
- $\frac{\partial c_3}{\partial W} \propto c_2 + \frac{\partial c_2}{\partial W} \quad (\frac{\partial W}{\partial W} = 1)$

$$c_t = \tanh(U\, x_t + W c_{t-1})$$
$$y_t = \mathrm{softmax}(V c_t)$$
$$\mathcal{L} = -\sum_t l_t \log y_t = \sum_t \mathcal{L}_t$$

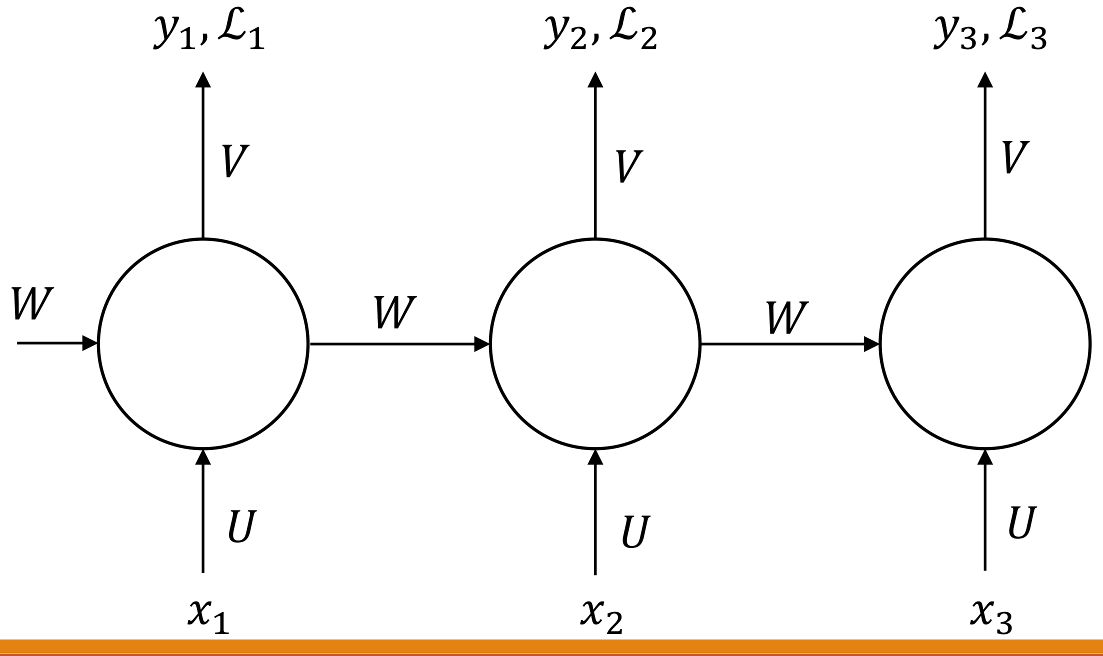


Recursively

- $\frac{\partial c_3}{\partial W} = c_2 + \frac{\partial c_2}{\partial W}$
- $\frac{\partial c_2}{\partial W} = c_1 + \frac{\partial c_1}{\partial W}$
- $\frac{\partial c_1}{\partial W} = c_0 + \frac{\partial c_0}{\partial W}$

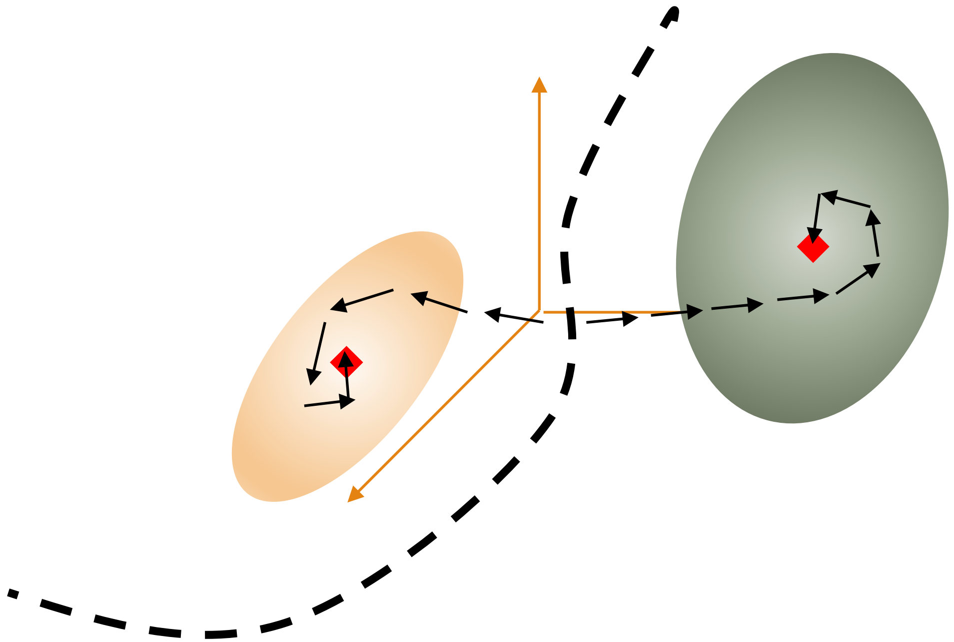
$$\frac{\partial c_3}{\partial W} = \sum_{t=1}^{3} \frac{\partial c_3}{\partial c_t} \frac{\partial c_t}{\partial W} \Rightarrow \boxed{\frac{\partial \mathcal{L}_3}{\partial W} = \sum_{t=1}^{3} \frac{\partial \mathcal{L}_3}{\partial y_3} \frac{\partial y_3}{\partial c_3} \frac{\partial c_3}{\partial c_t} \frac{\partial c_t}{\partial W}}$$

$$c_t = \tanh(U\, x_t + W c_{t-1})$$
$$y_t = \mathrm{softmax}(V c_t)$$
$$\mathcal{L} = -\sum_t l_t \log y_t = \sum_t \mathcal{L}_t$$



What makes RNNs tick?

- The latent memory space is composed of multiple dimensions
- A subspace of the memory state space can store information if multiple basins ◆ of attraction in some dimensions exist
- Gradients must be strong near the basin boundaries



Training RNNs is hard

- Vanishing gradients
  - After a few time steps the gradients become almost 0
- Exploding gradients
  - After a few time steps the gradients become huge
- Can’t capture long-term dependencies

Alternative formulation for RNNs

- An alternative formulation to derive conclusions and intuitions

$$c_t = W \cdot \tanh(c_{t-1}) + U \cdot x_t + b$$

$$\mathcal{L} = \sum_t \mathcal{L}_t(c_t)$$

Another look at the gradients

- $\mathcal{L} = L(c_T(c_{T-1}(\dots(c_1(x_1, c_0; W); W); W); W)$
- $\frac{\partial \mathcal{L}_t}{\partial W} = \sum_{\tau=1}^{t} \frac{\partial \mathcal{L}_t}{\partial c_t} \frac{\partial c_t}{\partial c_\tau} \frac{\partial c_\tau}{\partial W}$
- $\frac{\partial \mathcal{L}}{\partial c_t} \frac{\partial c_t}{\partial c_\tau} = \underbrace{\frac{\partial \mathcal{L}}{\partial c_t} \cdot \frac{\partial c_t}{\partial c_{t-1}} \cdot \frac{\partial c_{t-1}}{\partial c_{t-2}}}_{Rest \;\to\; \text{short-term factors}} \cdot \dots \cdot \underbrace{\frac{\partial c_{\tau+1}}{\partial c_\tau}}_{t \gg \tau \;\to\; \text{long-term factors}} \leq \eta^{t-\tau} \frac{\partial \mathcal{L}_t}{\partial c_t}$

η determines the norm of the gradients

- The RNN gradient is a recursive product of $\frac{\partial c_t}{\partial c_{t-1}}$

RNN gradients in 1D

- $$\frac{\partial \mathcal{L}}{\partial c_t} = \frac{\partial \mathcal{L}}{\partial c_T} \cdot \underbrace{\frac{\partial c_T}{\partial c_{T-1}}}_{<1} \cdot \underbrace{\frac{\partial c_{T-1}}{\partial c_{T-2}}}_{<1} \cdot \ldots \cdot \underbrace{\frac{\partial c_{t+1}}{\partial c_{c_t}}}_{<1} \Bigg\} \quad \frac{\partial \mathcal{L}}{\partial W} \ll 1 \Longrightarrow \text{ Vanishing gradient}$$
- $$\frac{\partial \mathcal{L}}{\partial c_t} = \frac{\partial \mathcal{L}}{\partial c_T} \cdot \underbrace{\frac{\partial c_T}{\partial c_{T-1}}}_{>1} \cdot \underbrace{\frac{\partial c_{T-1}}{\partial c_{T-2}}}_{>1} \cdot \ldots \cdot \underbrace{\frac{\partial c_1}{\partial c_{c_t}}}_{>1} \Bigg\} \quad \frac{\partial \mathcal{L}}{\partial W} \gg 1 \Longrightarrow \text{ Exploding gradient}$$

RNN gradients in N-D

- When $c_T \in \mathbb{R}^N$ then $\frac{\partial c_t}{\partial \mathrm{c}_{\mathrm{t}-1}}$ is a Jacobian

- $$\frac{\partial \mathcal{L}}{\partial c_t} = \frac{\partial \mathcal{L}}{\partial \mathrm{c}_{\mathrm{T}}} \cdot \underbrace{\frac{\partial c_T}{\partial \mathrm{c}_{\mathrm{T}-1}}}_{<1} \cdot \underbrace{\frac{\partial c_{T-1}}{\partial \mathrm{c}_{\mathrm{T}-2}}}_{<1} \cdot \ldots \cdot \underbrace{\frac{\partial c_{t+1}}{\partial \mathrm{c}_{\mathrm{t}}}}_{<1} \quad \frac{\partial \mathcal{L}}{\partial \theta} \ll 1 \Longrightarrow \text{ Vanishing gradient}$$

- $$\frac{\partial \mathcal{L}}{\partial c_t} = \frac{\partial \mathcal{L}}{\partial \mathrm{c}_{\mathrm{T}}} \cdot \underbrace{\frac{\partial c_T}{\partial \mathrm{c}_{\mathrm{T}-1}}}_{>1} \cdot \underbrace{\frac{\partial c_{T-1}}{\partial \mathrm{c}_{\mathrm{T}-2}}}_{>1} \cdot \ldots \cdot \underbrace{\frac{\partial c_{t+1}}{\partial \mathrm{c}_{\mathrm{t}}}}_{>1} \quad \frac{\partial \mathcal{L}}{\partial \theta} \gg 1 \Longrightarrow \text{ Exploding gradient}$$

The Jacobian

$$y \in \mathbb{R}^2, x \in \mathbb{R}^3: \ \frac{d\boldsymbol{y}}{d\boldsymbol{x}} = \begin{bmatrix} \frac{\partial y^{(1)}}{\partial x^{(1)}} & \frac{\partial y^{(1)}}{\partial x^{(2)}} & \frac{\partial y^{(1)}}{\partial x^{(3)}} \\ \frac{\partial y^{(2)}}{\partial x^{(1)}} & \frac{\partial y^{(2)}}{\partial x^{(2)}} & \frac{\partial y^{(2)}}{\partial x^{(3)}} \end{bmatrix}$$

RNN gradients in N-D

- When $c_T \in \mathbb{R}^N$ then $\frac{\partial c_t}{\partial \mathrm{c}_{t-1}}$ is a Jacobian
- Spectral radius ρ (~largest eigenvalue) of Jacobian is important

- $$\frac{\partial \mathcal{L}}{\partial c_t} = \frac{\partial \mathcal{L}}{\partial \mathrm{c}_T} \cdot \left.\underset{\rho<1}{\frac{\partial c_T}{\partial \mathrm{c}_{T-1}}} \cdot \underset{\rho<1}{\frac{\partial c_{T-1}}{\partial \mathrm{c}_{T-2}}} \cdot \ldots \cdot \underset{\rho<1}{\frac{\partial c_{t+1}}{\partial \mathrm{c}_{c_t}}}\right\} \quad \frac{\partial \mathcal{L}}{\partial c_t} \ll 1 \Longrightarrow \text{ Vanishing gradient}$$

- $$\frac{\partial \mathcal{L}}{\partial c_t} = \frac{\partial \mathcal{L}}{\partial \mathrm{c}_T} \cdot \left.\underset{\rho>1}{\frac{\partial c_T}{\partial \mathrm{c}_{T-1}}} \cdot \underset{\rho>1}{\frac{\partial c_{T-1}}{\partial \mathrm{c}_{T-2}}} \cdot \ldots \cdot \underset{\rho>1}{\frac{\partial c_{t+1}}{\partial c_t}}\right\} \quad \frac{\partial \mathcal{L}}{\partial c_t} \gg 1 \Longrightarrow \text{ Exploding gradient}$$

Gradient clipping for exploding gradients

- Scale the gradients to a threshold

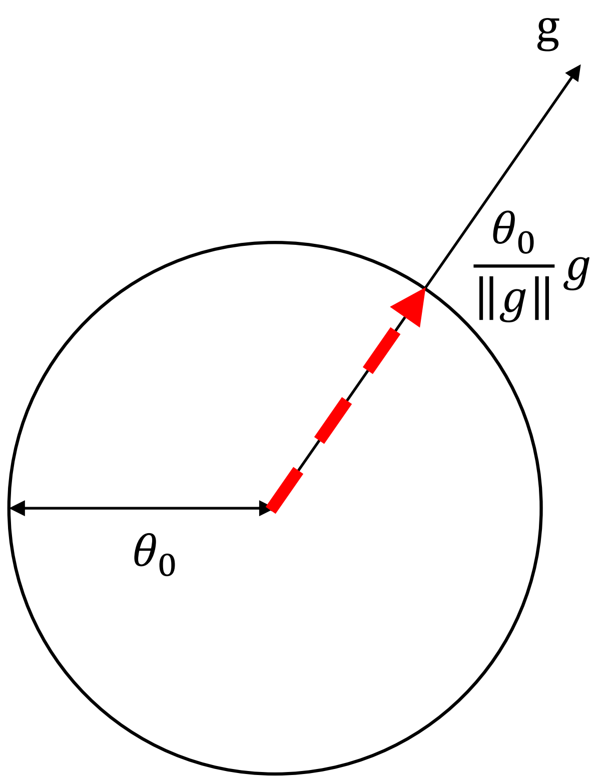
Pseudocode

1. $g \leftarrow \frac{\partial \mathcal{L}}{\partial W}$
2. if $\|g\| > \theta_0$:

   $g \leftarrow \frac{\theta_0}{\|g\|} g$

   else:

   print('Do nothing')



- Simple, but works!

Vanishing gradients

- The gradient of the error w.r.t. to intermediate cell

$$\frac{\partial \mathcal{L}_t}{\partial W} = \sum_{\tau=1}^{t} \frac{\partial \mathcal{L}_r}{\partial y_t} \frac{\partial y_t}{\partial c_t} \frac{\partial c_t}{\partial c_\tau} \frac{\partial c_\tau}{\partial W}$$

$$\frac{\partial c_t}{\partial c_\tau} = \prod_{t \geq k \geq \tau} \frac{\partial c_k}{\partial c_{k-1}} = \prod_{t \geq k \geq \tau} W \cdot \partial \tanh(c_{k-1})$$

Vanishing gradients

- For $\mathrm{t}=1, \mathrm{r}=2 \implies \frac{\partial \mathcal{L}_2}{\partial W} \propto \frac{\partial c_2}{\partial c_1}$
- For $\mathrm{t}=1, \mathrm{r}=3 \implies \frac{\partial \mathcal{L}_3}{\partial W} \propto \frac{\partial c_3}{\partial c_1} = \frac{\partial c_3}{\partial c_2} \cdot \frac{\partial c_2}{\partial c_1}$
- For $\mathrm{t}=1, \mathrm{r}=4 \implies \frac{\partial \mathcal{L}_4}{\partial W} \propto \frac{\partial c_4}{\partial c_1} = \frac{\partial c_4}{\partial c_3} \cdot \frac{\partial c_3}{\partial c_2} \cdot \frac{\partial c_2}{\partial c_1}$

Vanishing gradients

- The gradient of the error w.r.t. to intermediate cell

$$\frac{\partial \mathcal{L}_t}{\partial W} = \sum_{\tau=1}^{\mathrm{t}} \frac{\partial \mathcal{L}_r}{\partial y_t} \frac{\partial y_t}{\partial c_t} \frac{\partial c_t}{\partial c_\tau} \frac{\partial c_\tau}{\partial W}$$

$$\frac{\partial c_t}{\partial c_\tau} = \prod_{t \geq k \geq \tau} \frac{\partial c_k}{\partial c_{k-1}} = \prod_{t \geq k \geq \tau} W \cdot \partial \tanh(c_{k-1})$$

- Long-term dependencies get exponentially smaller weights

Rescaling vanishing gradients?

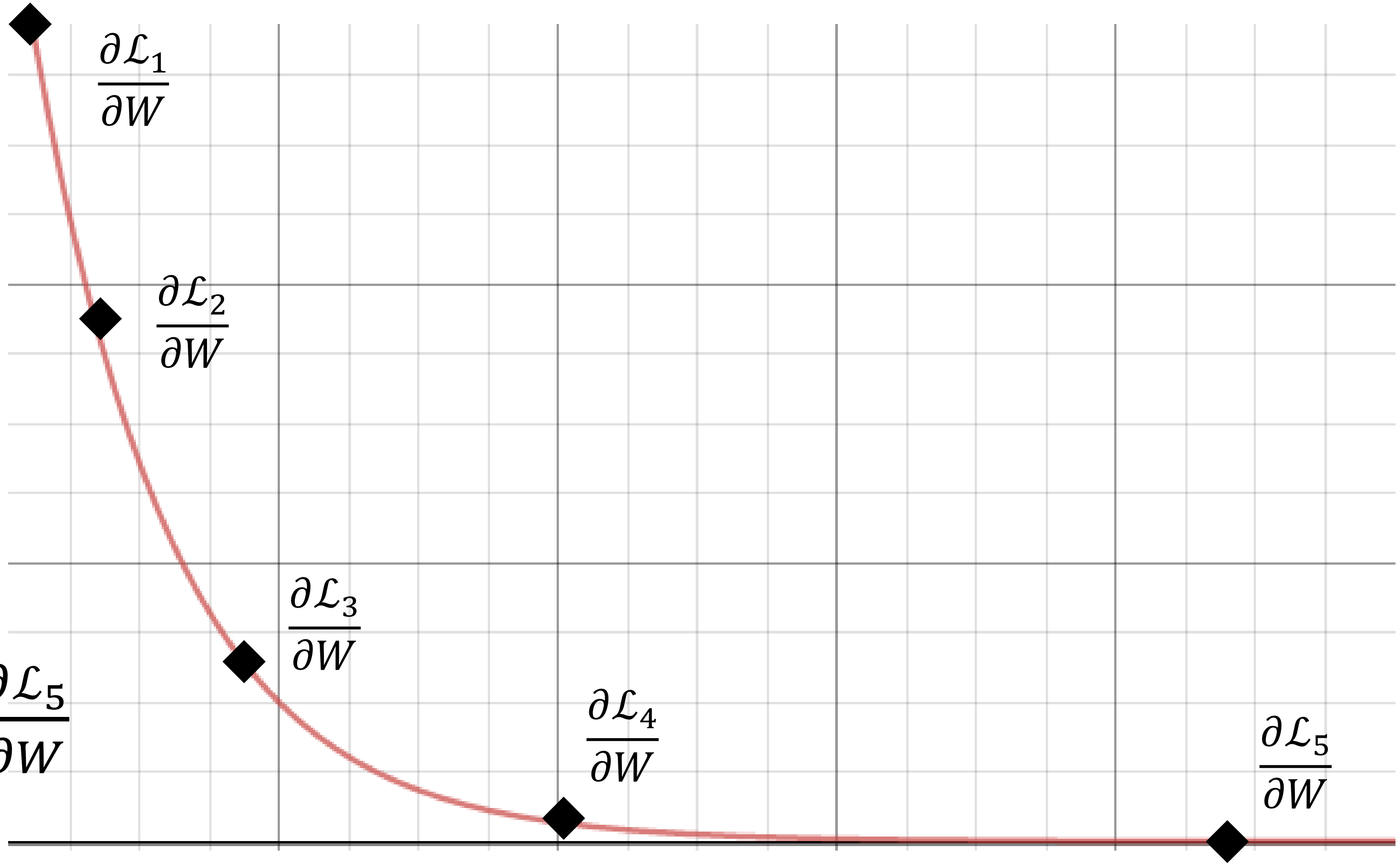
- Not good solution
- Weights are shared between timesteps → Loss summed over timesteps

$$\mathcal{L} = \sum_t \mathcal{L}_t \quad \Longrightarrow \quad \frac{\partial \mathcal{L}}{\partial W} = \sum_t \frac{\partial \mathcal{L}_t}{\partial W}$$

$$\frac{\partial \mathcal{L}_t}{\partial W} = \sum_{\tau=1}^{t} \frac{\partial \mathcal{L}_t}{\partial c_\tau} \frac{\partial c_\tau}{\partial W} = \sum_{\tau=1}^{t} \frac{\partial \mathcal{L}_t}{\partial c_t} \frac{\partial c_t}{\partial c_\tau} \frac{\partial c_\tau}{\partial W}$$

- Rescaling for one timestep ($\frac{\partial \mathcal{L}_t}{\partial W}$) affects all timesteps
  - The rescaling factor for one timestep does not work for another

More intuitively

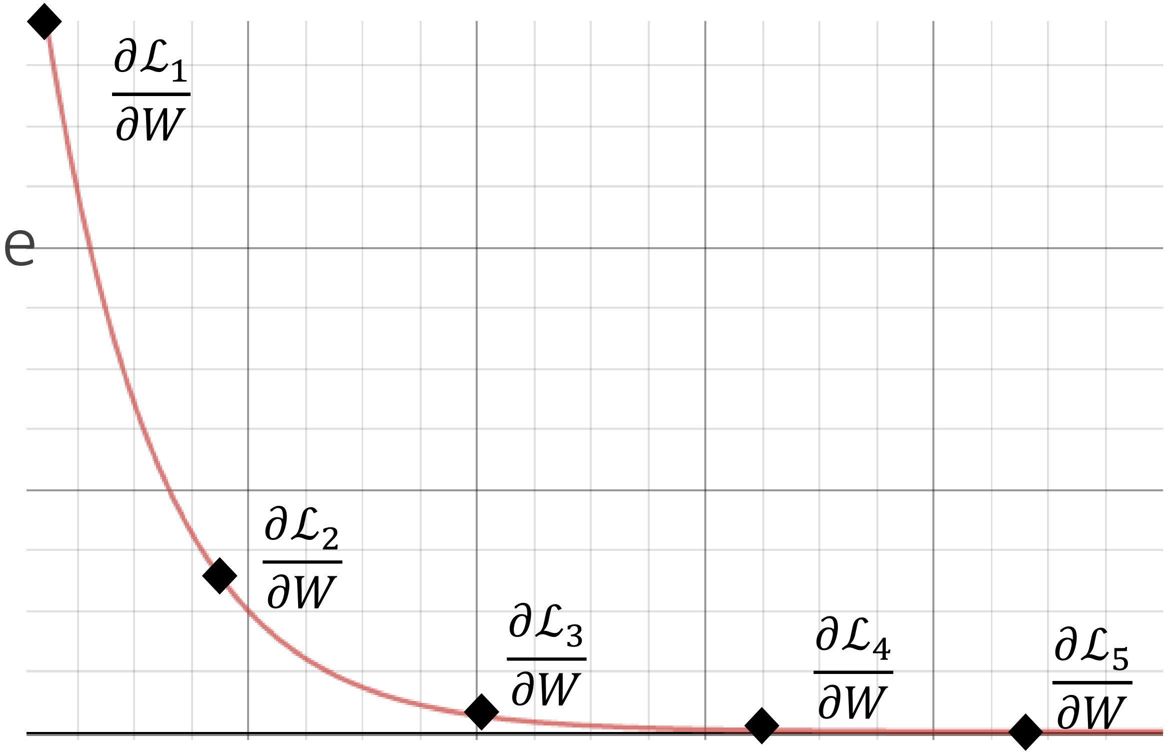


$$\frac{\partial \mathcal{L}}{\partial W} = \frac{\partial \mathcal{L}_1}{\partial W} + \frac{\partial \mathcal{L}_2}{\partial W} + \frac{\partial \mathcal{L}_3}{\partial W} + \frac{\partial \mathcal{L}_4}{\partial W} + \frac{\partial \mathcal{L}_5}{\partial W}$$

More intuitively

- Let's say $\frac{\partial \mathcal{L}_1}{\partial W} \propto 1, \frac{\partial \mathcal{L}_2}{\partial W} \propto 1/10, \frac{\partial \mathcal{L}_3}{\partial W} \propto 1/100, \frac{\partial \mathcal{L}_4}{\partial W} \propto 1/1000, \frac{\partial \mathcal{L}_5}{\partial W} \propto 1/10000$
- $\frac{\partial \mathcal{L}}{\partial W} = \sum_r \frac{\partial \mathcal{L}_r}{\partial W} = 1.1111$
- If $\frac{\partial \mathcal{L}}{\partial W}$ rescaled to 1 → $\frac{\partial \mathcal{L}_5}{\partial W} \propto 10^{-5}$
- Longer-term dependencies negligible
  - Weak recurrent modelling
  - Learning focuses on the short-term only

$$\frac{\partial \mathcal{L}}{\partial W} = \frac{\partial \mathcal{L}_1}{\partial W} + \frac{\partial \mathcal{L}_2}{\partial W} + \frac{\partial \mathcal{L}_3}{\partial W} + \frac{\partial \mathcal{L}_4}{\partial W} + \frac{\partial \mathcal{L}_5}{\partial W}$$



Recurrent networks ∝ Dynamical systems

- In the figures $\mathrm{x_t} \propto c_t$ and $x_t \propto \mathrm{F}(Wx_{t-1} + Uu_t + b)$

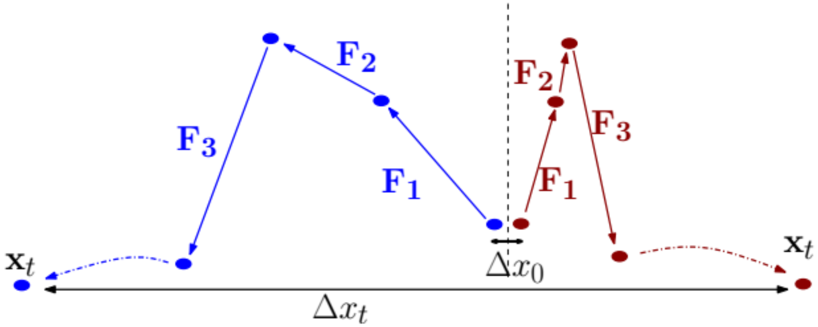


Figure 4. This diagram illustrates how the change in $\mathbf{x}_t$, $\Delta\mathbf{x}_t$, can be large for a small $\Delta\mathbf{x}_0$. The blue vs red (left vs right) trajectories are generated by the same maps $F_1, F_2, \ldots$ for two different initial states.

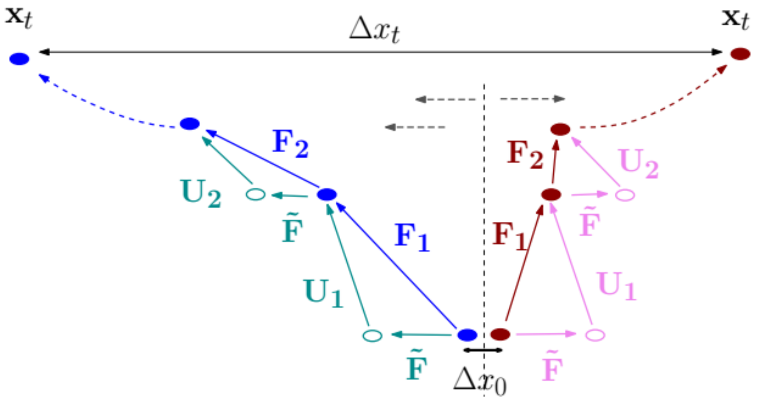


Figure 5. Illustrates how one can break apart the maps $F_1, ..F_t$ into a constant map $\tilde{F}$ and the maps $U_1, .., U_t$. The dotted vertical line represents the boundary between basins of attraction, and the straight dashed arrow the direction of the map $\tilde{F}$ on each side of the boundary. This diagram is an extension of Fig. 4.

Fixing vanishing gradients

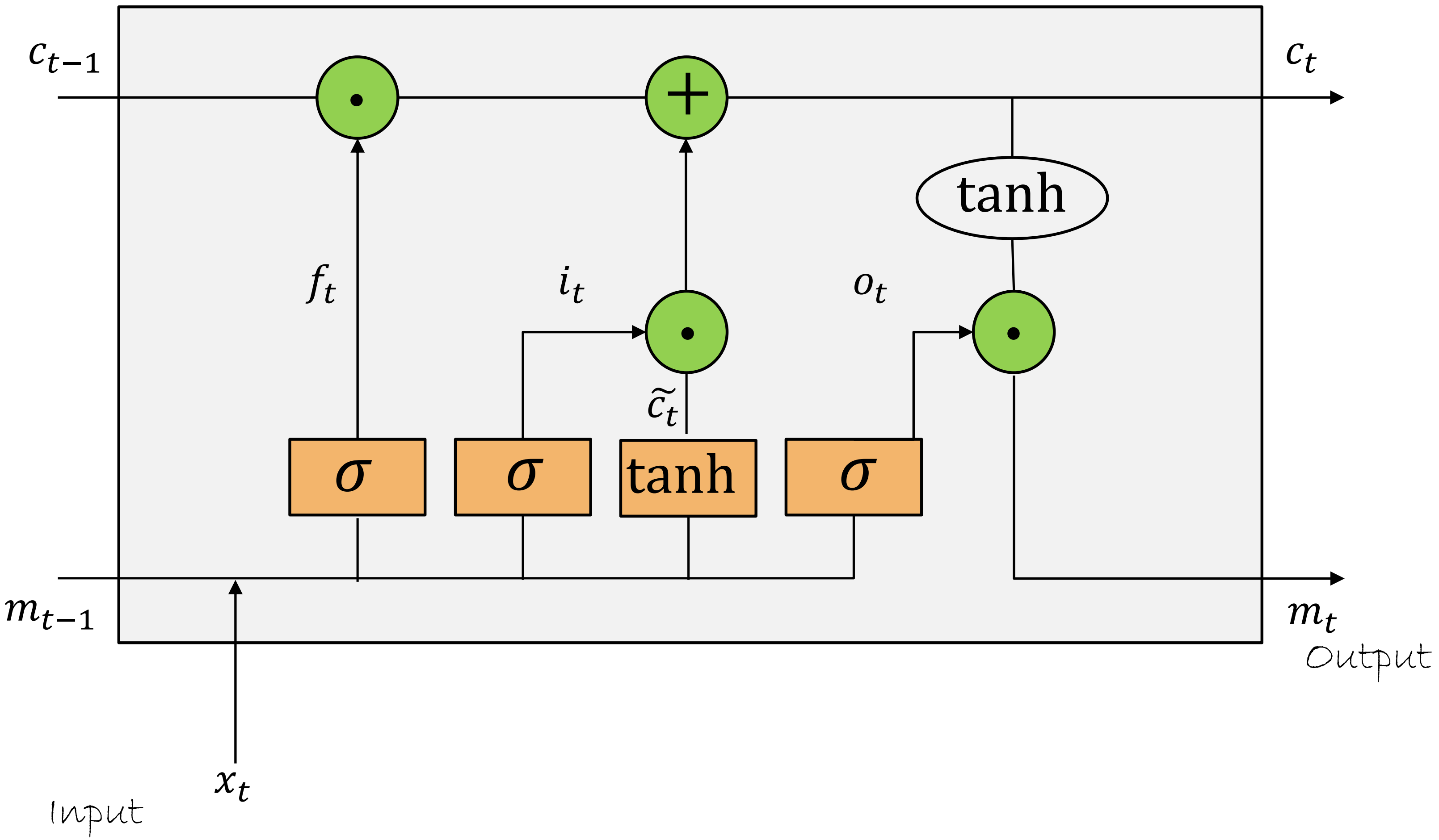
- Regularization on the recurrent weights
  - Force the error signal not to vanish

$$\Omega = \sum_t \Omega_t = \sum_t \left(\frac{\left| \frac{\partial \mathcal{L}}{\partial c_{t+1}} \frac{\partial c_{t+1}}{\partial c_t} \right|}{\left| \frac{\partial \mathcal{L}}{\partial c_{t+1}} \right|} - 1 \right)^2$$

- Advanced recurrent modules
- Long-Short Term Memory module
- Gated Recurrent Unit module

Pascanu et al., On the diculty of training Recurrent Neural Networks, 2013

Advanced RNNs



How to fix the vanishing gradients?

- Error signal over time must have not too large, not too small norm
- Solution: have an activation function with derivative equal to 1
  - Identify function
- By doing so, gradients do not become too small not too large

Long Short-Term Memory (LSTM: Beefed up RNN)

$$i = \sigma\left(x_t U^{(i)} + m_{t-1} W^{(i)}\right)$$

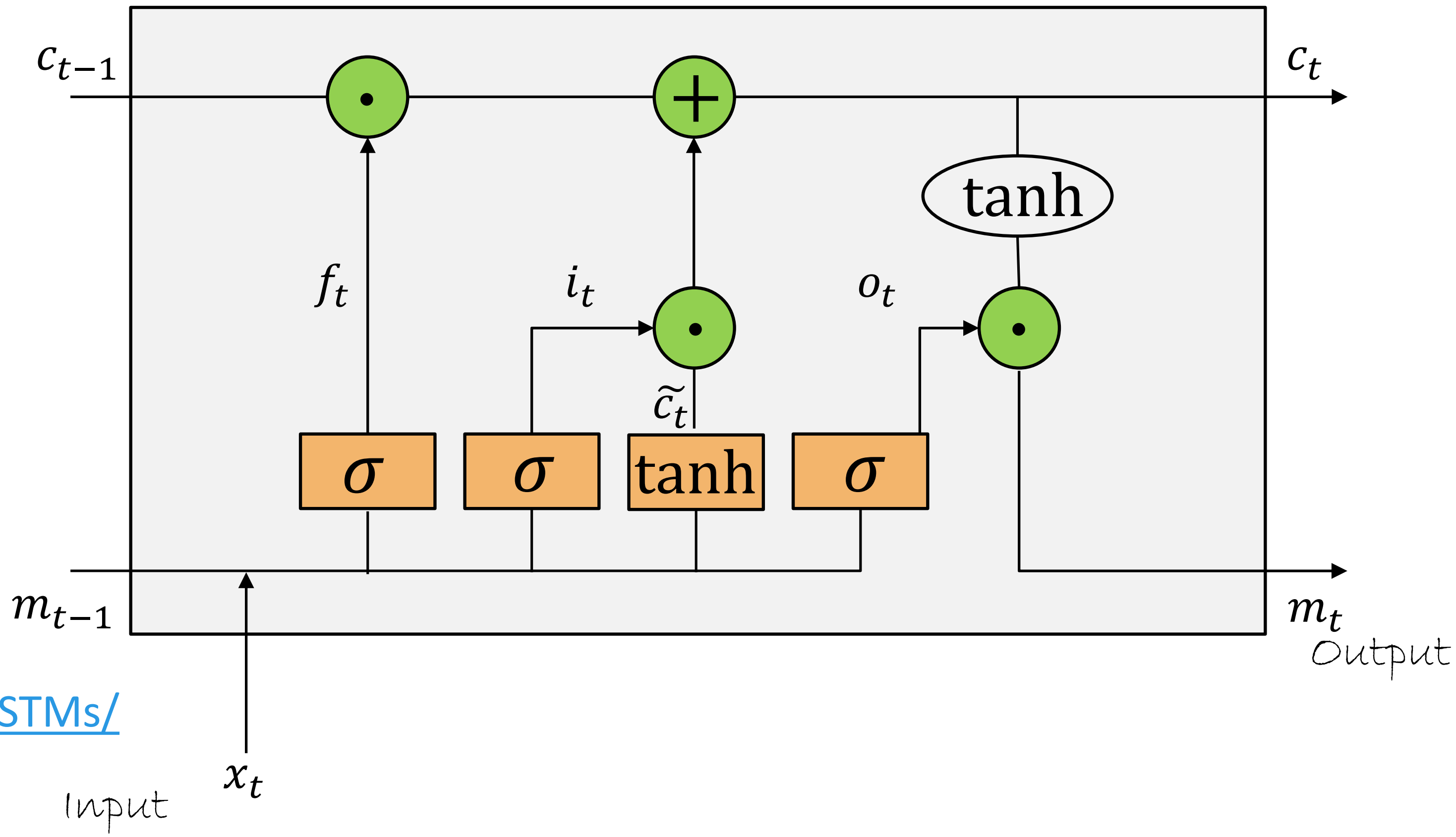
$$f = \sigma\left(x_t U^{(f)} + m_{t-1} W^{(f)}\right)$$

$$o = \sigma\left(x_t U^{(o)} + m_{t-1} W^{(o)}\right)$$

$$\widetilde{c}_t = \tanh(x_t U^{(g)} + m_{t-1} W^{(g)})$$

$$c_t = c_{t-1} \odot f + \widetilde{c}_t \odot i$$

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



More info at:
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Cell state

- The cell state carries the essential information over time

Cell state line

$$i = \sigma\left(x_t U^{(i)} + m_{t-1} W^{(i)}\right)$$

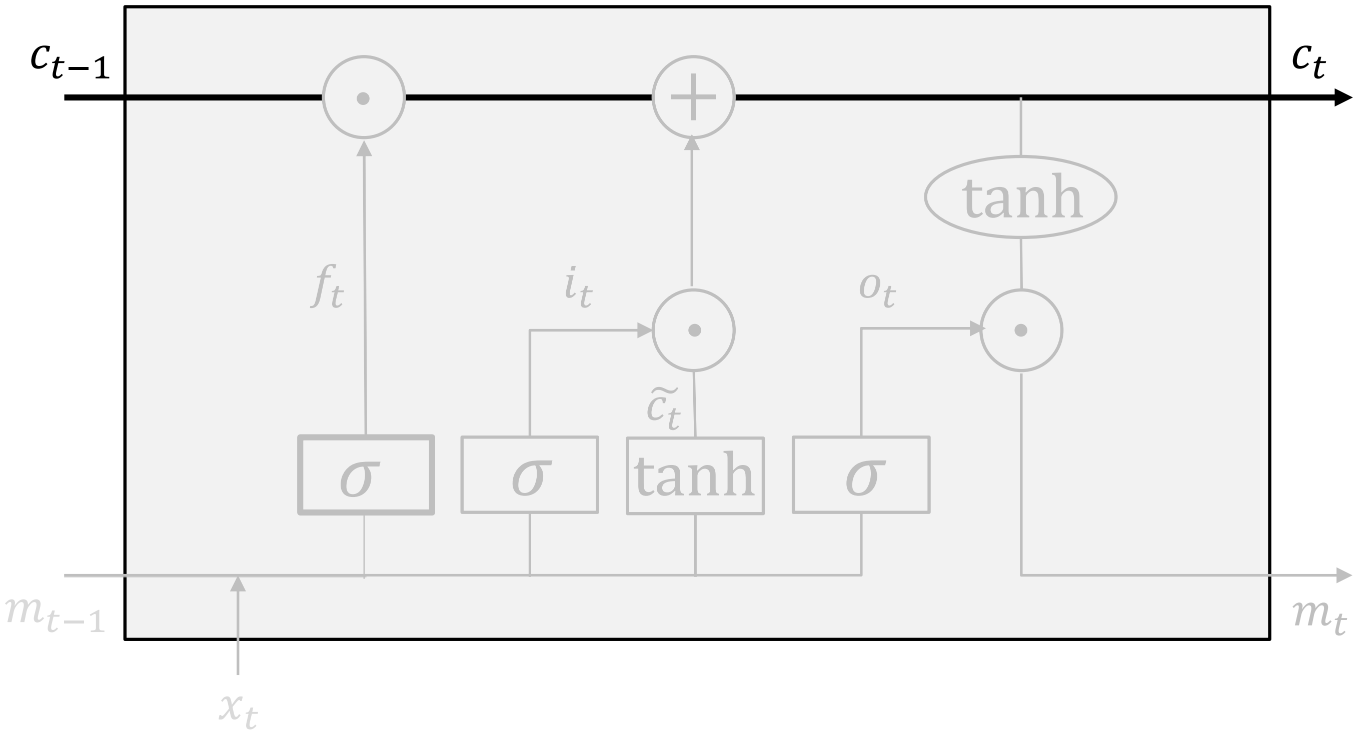
$$f = \sigma\left(x_t U^{(f)} + m_{t-1} W^{(f)}\right)$$

$$o = \sigma\left(x_t U^{(o)} + m_{t-1} W^{(o)}\right)$$

$$\tilde{c}_t = \tanh(x_t U^{(g)} + m_{t-1} W^{(g)})$$

$$c_t = c_{t-1} \odot f + \tilde{c}_t \odot i$$

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



LSTM nonlinearities

- $\sigma \in (0,1)$: control gate – something like a switch
- $\tanh \in (-1,1)$: recurrent nonlinearity

$$i = \sigma\left(x_t U^{(i)} + m_{t-1} W^{(i)}\right)$$

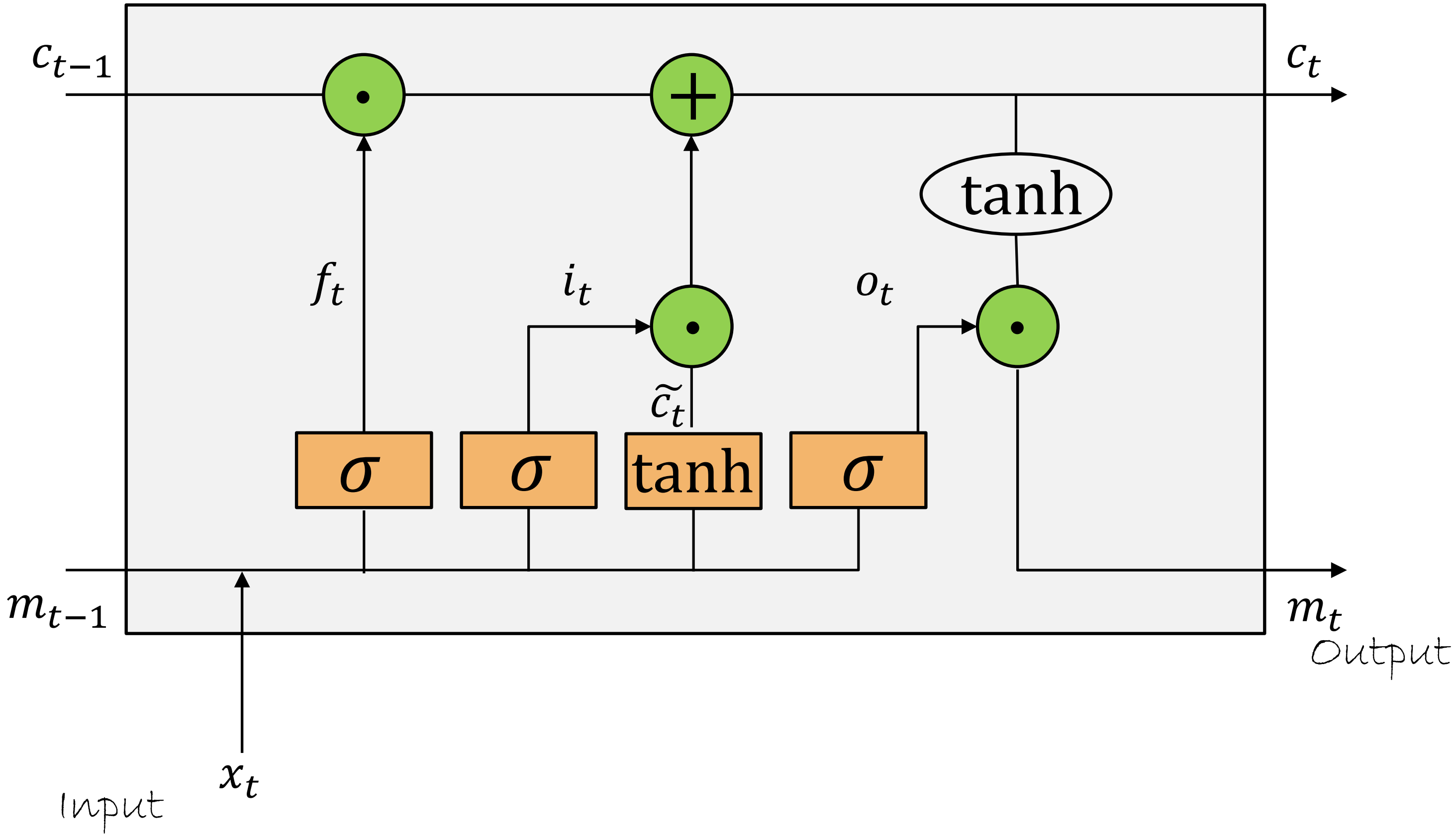
$$f = \sigma\left(x_t U^{(f)} + m_{t-1} W^{(f)}\right)$$

$$o = \sigma\left(x_t U^{(o)} + m_{t-1} W^{(o)}\right)$$

$$\tilde{c}_t = \tanh(x_t U^{(g)} + m_{t-1} W^{(g)})$$

$$c_t = c_{t-1} \odot f + \tilde{c}_t \odot i$$

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



LSTM Step-by-Step: Step (1)

- E.g. Model the sentence "Yesterday she slapped me. Today she loves me."
- Decide what to forget and what to remember for the new memory
  - Sigmoid 1 → Remember everything
  - Sigmoid 0 → Forget everything

$$i_t = \sigma\left(x_t U^{(i)} + m_{t-1} W^{(i)}\right)$$

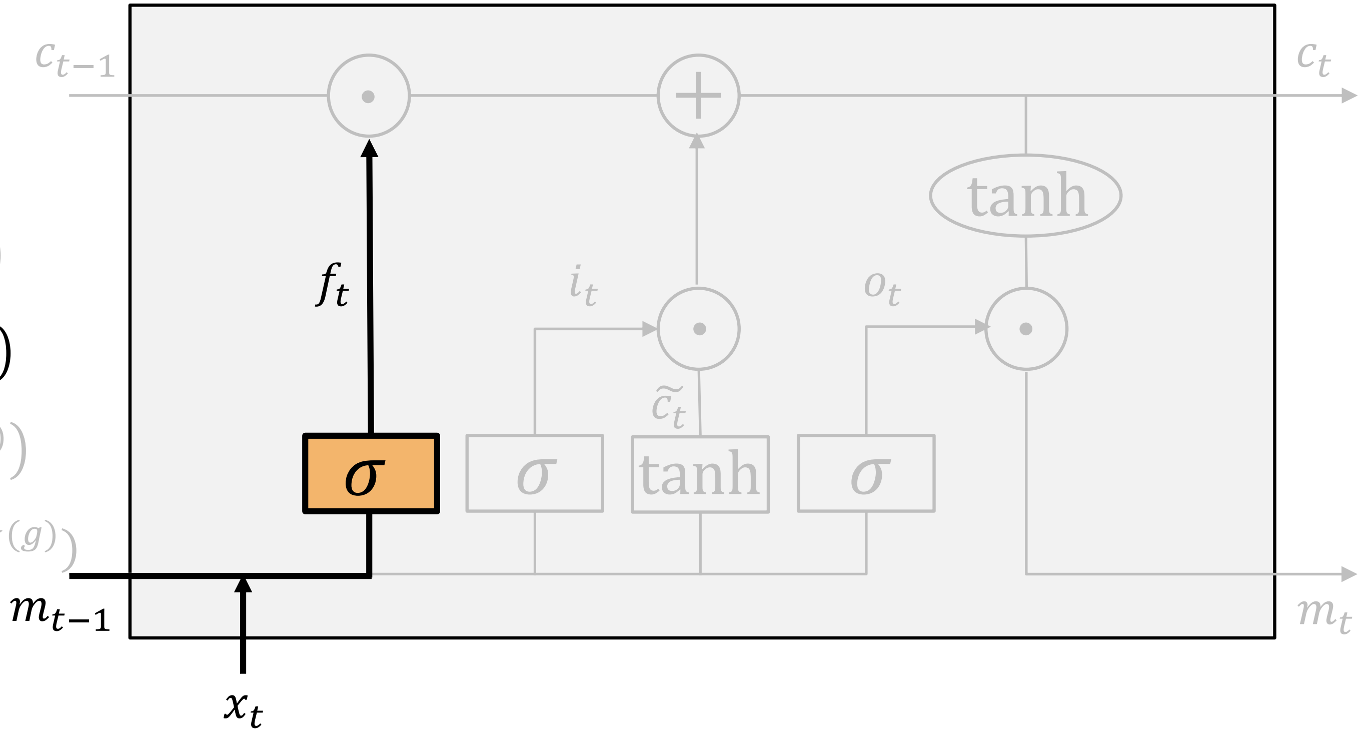
$$f_t = \sigma\left(x_t U^{(f)} + m_{t-1} W^{(f)}\right)$$

$$o_t = \sigma\left(x_t U^{(o)} + m_{t-1} W^{(o)}\right)$$

$$\tilde{c}_t = \tanh(x_t U^{(g)} + m_{t-1} W^{(g)})$$

$$c_t = c_{t-1} \odot f + \tilde{c}_t \odot i$$

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



LSTM Step-by-Step: Step (2)

- Decide what new information should you add to the new memory
  - Modulate the input i_t
  - Generate candidate memories $\widetilde{c_t}$

$$i_t = \sigma\left(x_t U^{(i)} + m_{t-1} W^{(i)}\right)$$

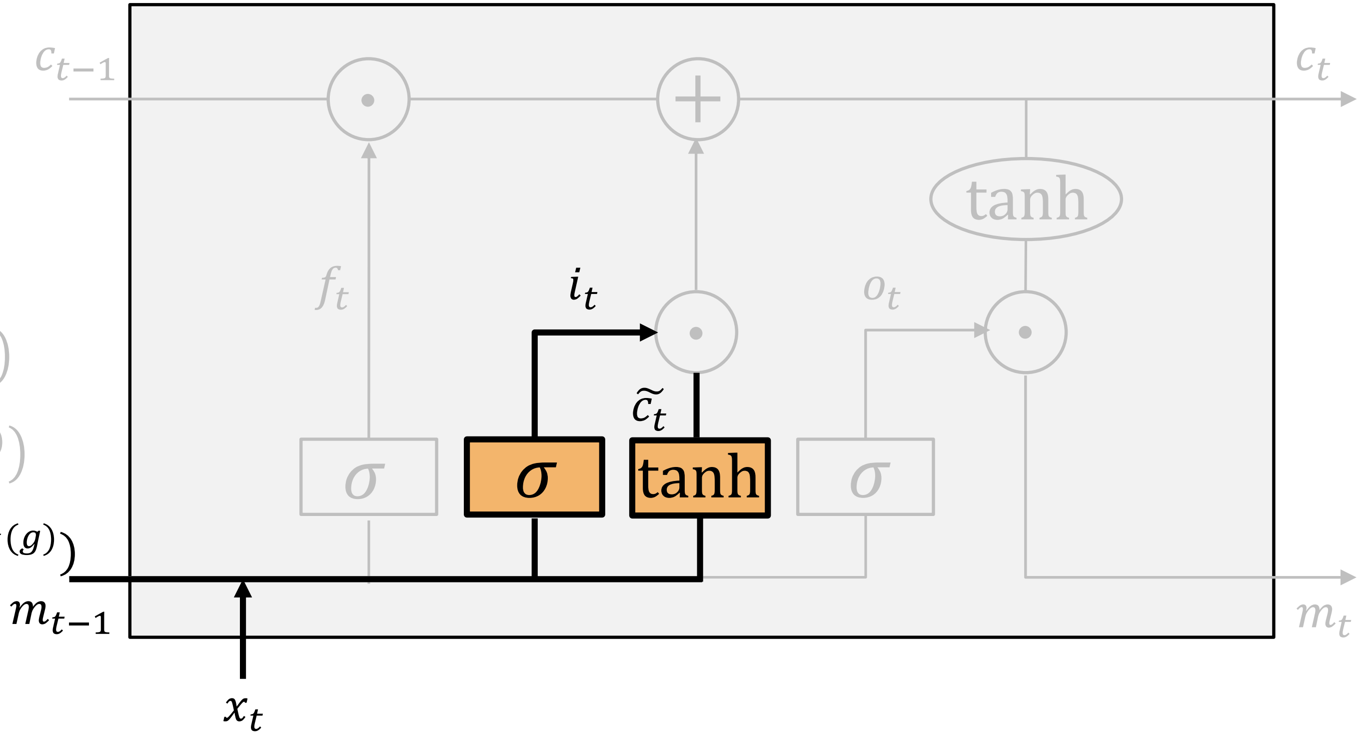
$$f_t = \sigma\left(x_t U^{(f)} + m_{t-1} W^{(f)}\right)$$

$$o_t = \sigma\left(x_t U^{(o)} + m_{t-1} W^{(o)}\right)$$

$$\widetilde{c_t} = \tanh(x_t U^{(g)} + m_{t-1} W^{(g)})$$

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

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



LSTM Step-by-Step: Step (3)

- Compute and update the current cell state c_t
  - Depends on the previous cell state
  - What we decide to forget
  - What inputs we allow
  - The candidate memories

$$i_t = \sigma\left(x_t U^{(i)} + m_{t-1} W^{(i)}\right)$$

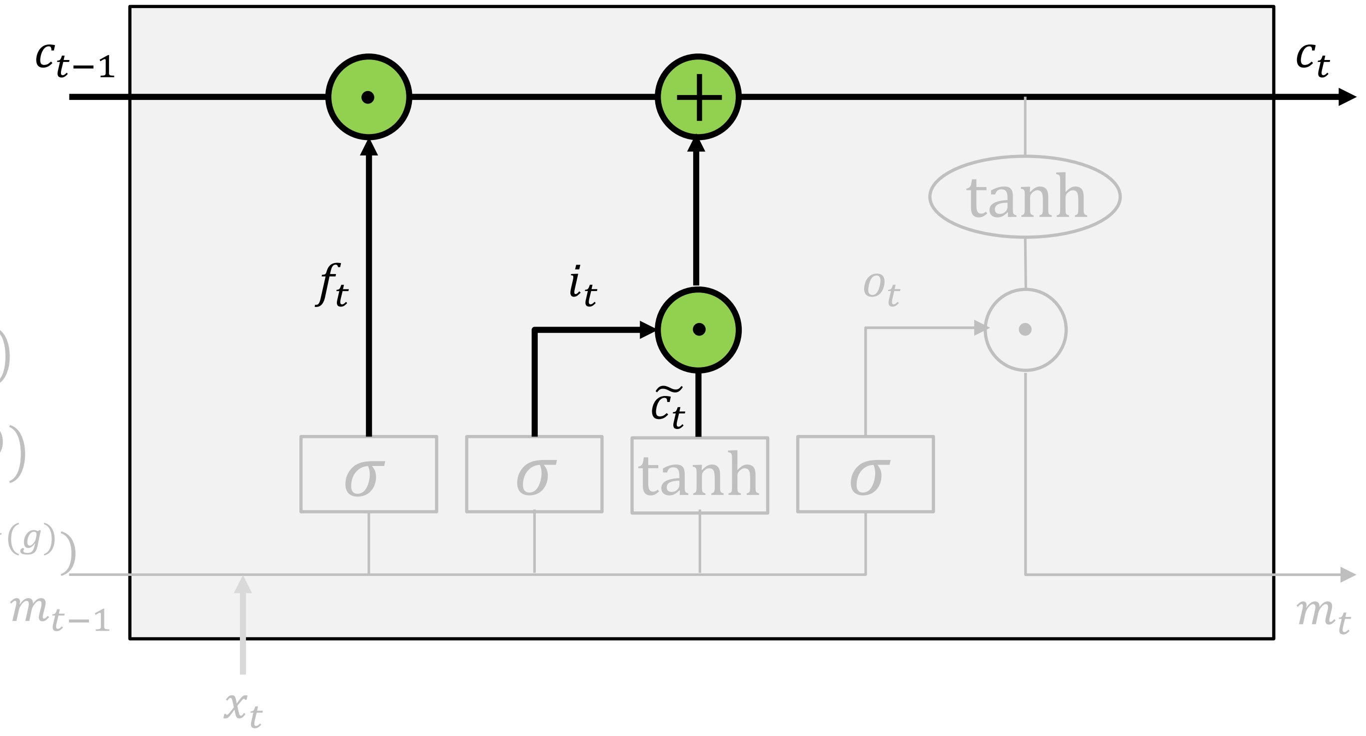
$$f_t = \sigma\left(x_t U^{(f)} + m_{t-1} W^{(f)}\right)$$

$$o_t = \sigma\left(x_t U^{(o)} + m_{t-1} W^{(o)}\right)$$

$$\widetilde{c_t} = \tanh(x_t U^{(g)} + m_{t-1} W^{(g)})$$

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

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



LSTM Step-by-Step: Step (4)

- Modulate the output
  - Does the cell state contain something relevant? → Sigmoid 1
- Generate the new memory

$$i_t = \sigma\left(x_t U^{(i)} + m_{t-1} W^{(i)}\right)$$

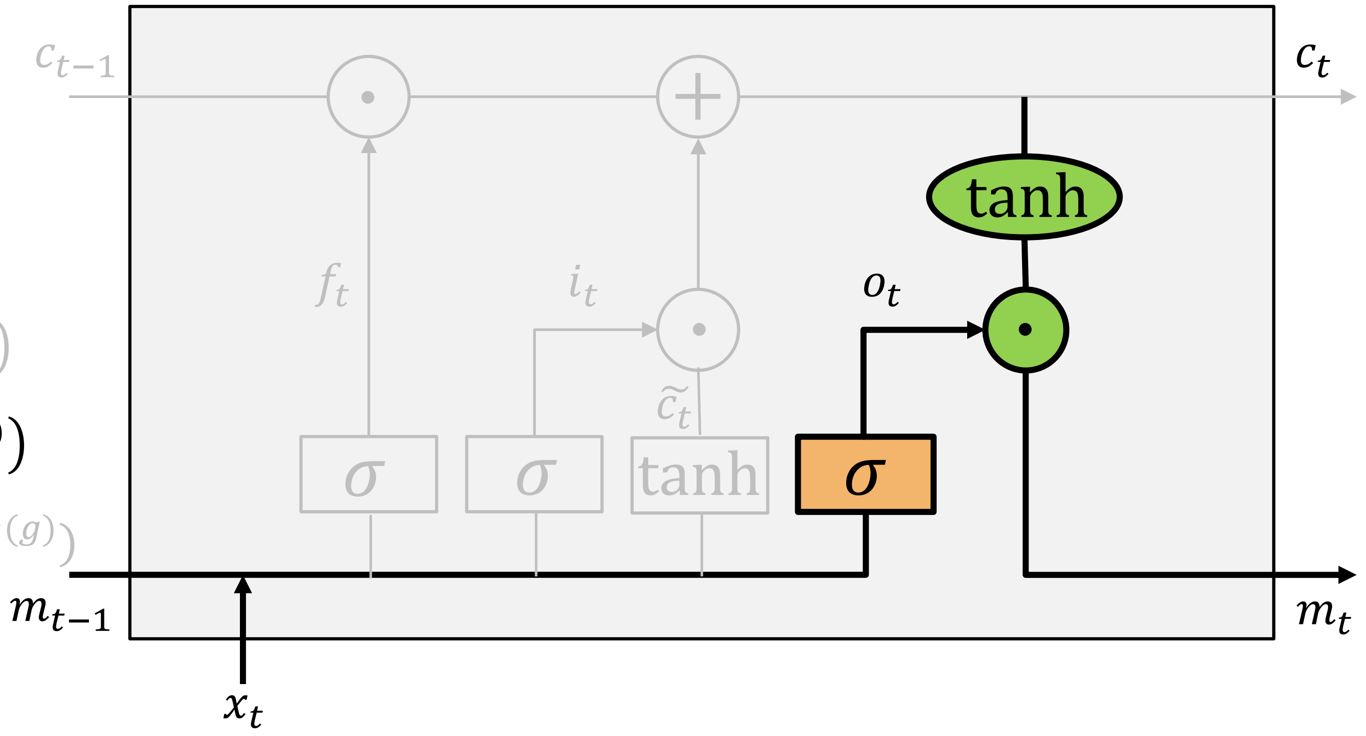
$$f_t = \sigma\left(x_t U^{(f)} + m_{t-1} W^{(f)}\right)$$

$$o_t = \sigma\left(x_t U^{(o)} + m_{t-1} W^{(o)}\right)$$

$$\widetilde{c_t} = \tanh(x_t U^{(g)} + m_{t-1} W^{(g)})$$

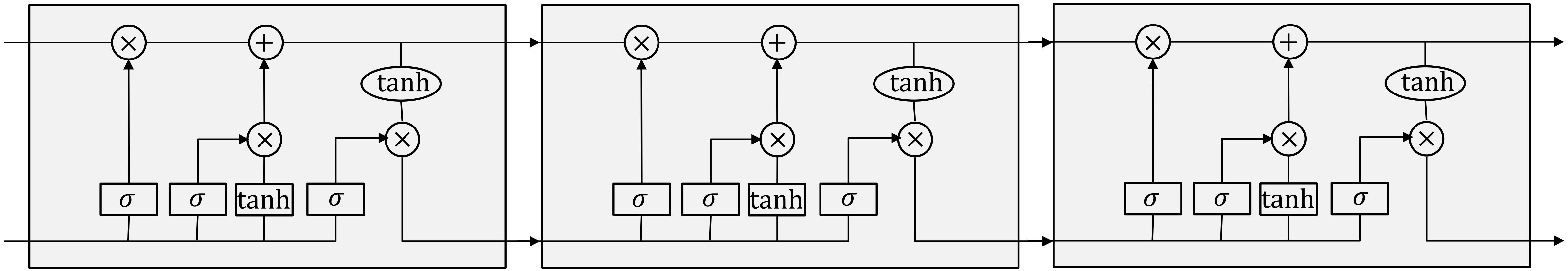
$$c_t = c_{t-1} \odot f + \widetilde{c_t} \odot i$$

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



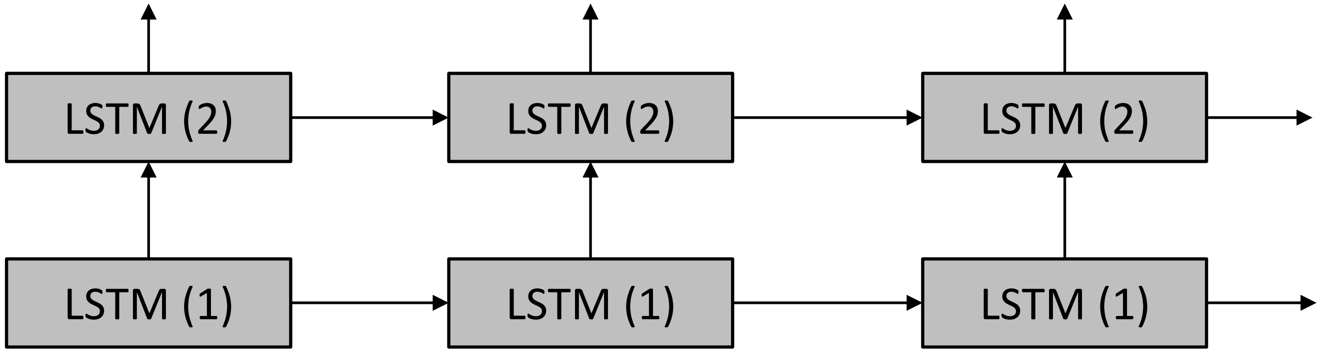
LSTM Unrolled Network

- Macroscopically very similar to standard RNNs
- The engine is a bit different (more complicated)
  - Because of their gates LSTMs capture long and short term dependencies



Beyond RNN & LSTM

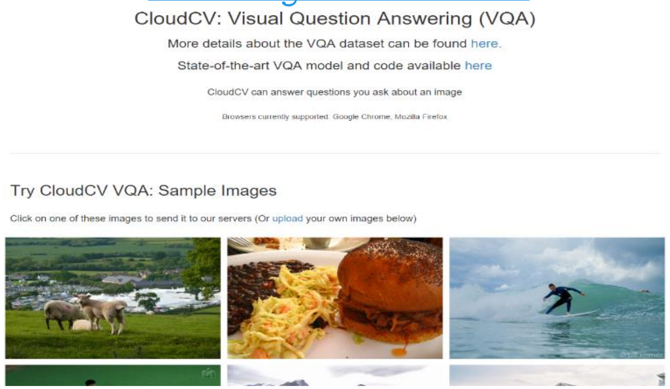
- LSTM with peephole connections
  - Gates have access also to the previous cell states c_{t-1} (not only memories)
  - Coupled forget and input gates, $c_t = f_t \odot c_{t-1} + (1 - f_t) \odot \widetilde{c_t}$
  - Bi-directional recurrent networks
- Gated Recurrent Units (GRU)
- Deep recurrent architectures
- Recursive neural networks
  - Tree structured
- Multiplicative interactions
- Generative recurrent architectures

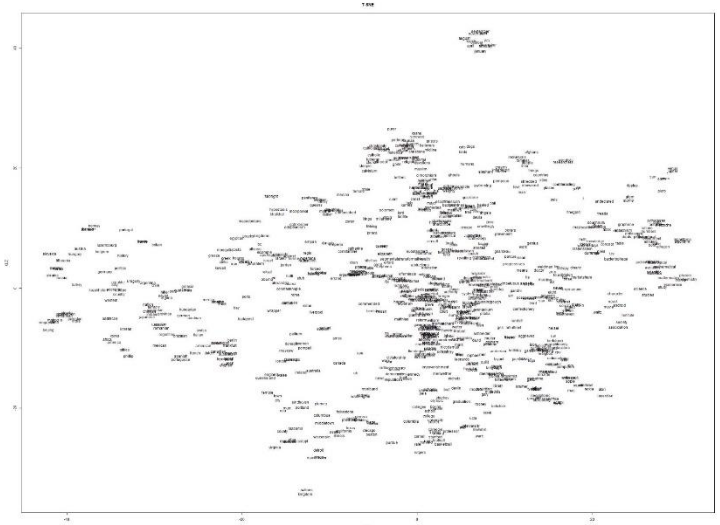


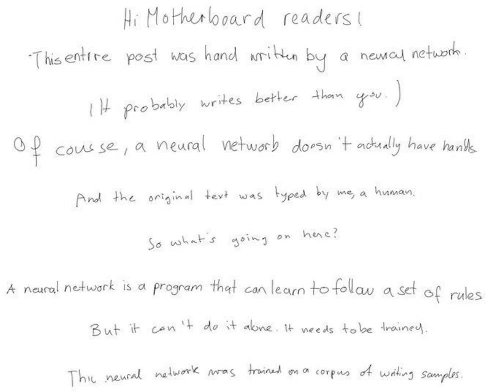
Applications of Recurrent Networks

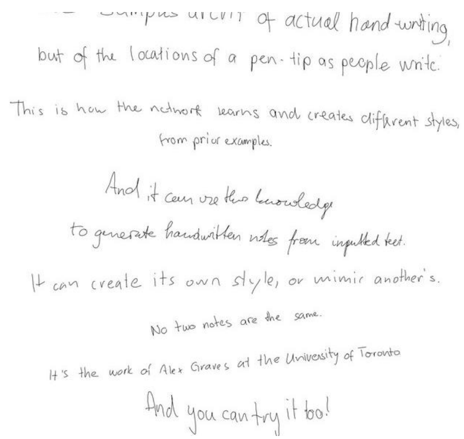
[Click to go to the video in Youtube]

[Click to go to the website]



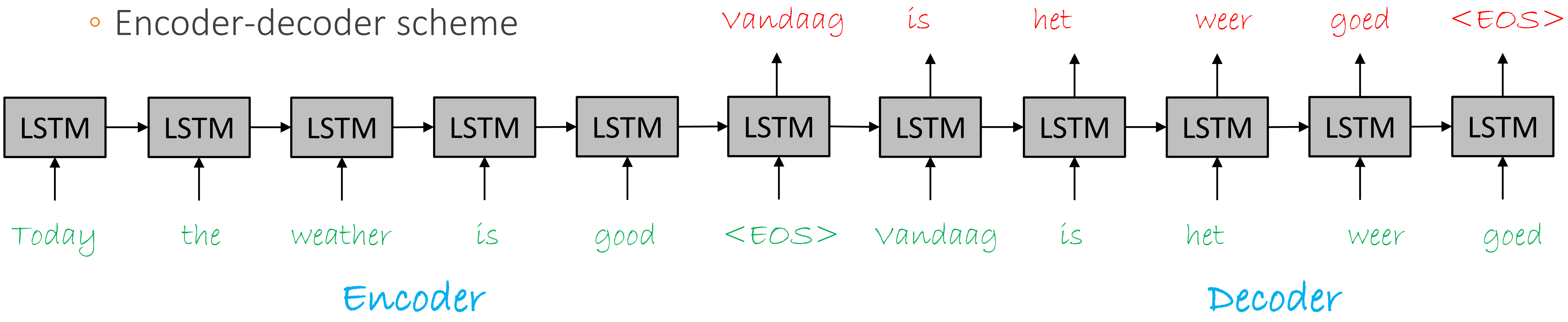






Machine Translation

- The phrase in the source language is one sequence
  - “Today the weather is good”
- The phrase in the target language is also a sequence
  - “Vandaag is het weer goed”
- Problems
  - no perfect word alignment, sentence length might differ
- Solution
  - Encoder-decoder scheme



Better Machine Translation

- It might even pay off reversing the source sentence
  - The first target words will be closer to their respective source words
- The encoder and decoder parts can be modelled with different LSTMs
- Deep LSTM

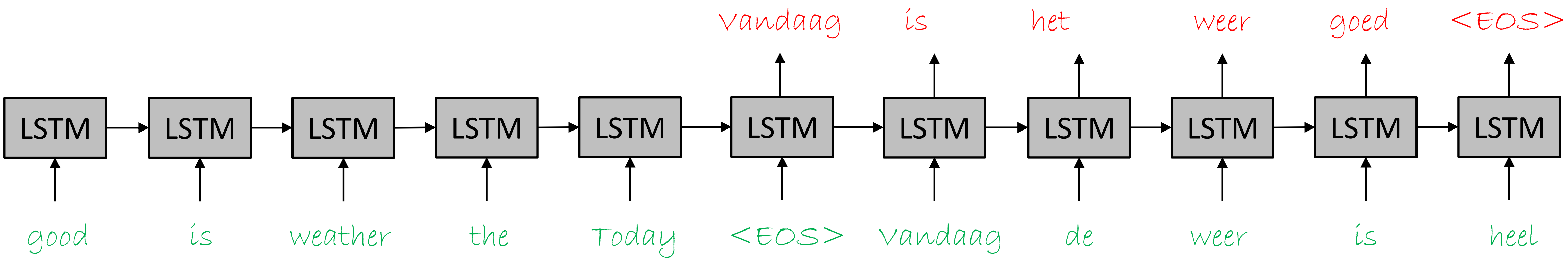
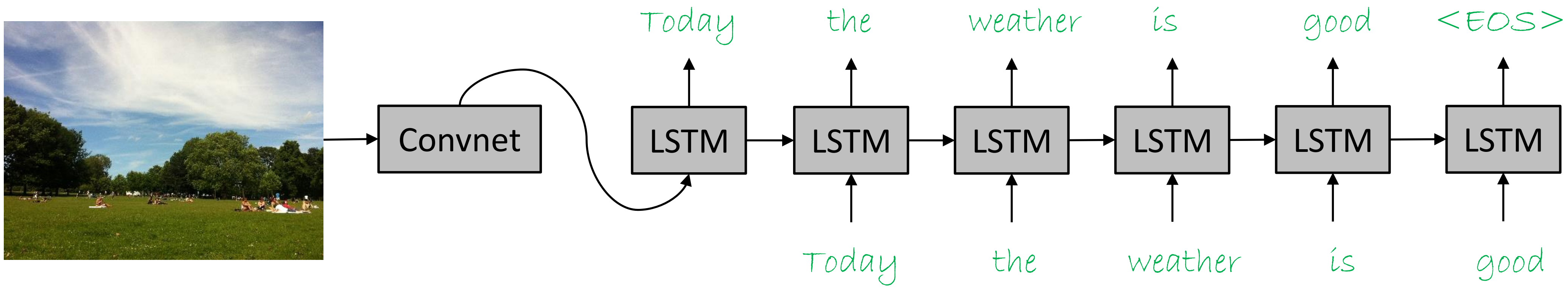


Image captioning

- An image is a thousand words, literally!
- Pretty much the same as machine transation
- Replace the encoder part with the output of a Convnet
  - E.g. use Alexnet or a VGG16 network
- Keep the decoder part to operate as a translator



Question answering

- Bleeding-edge research, no real consensus
  - Very interesting open, research problems
- Again, pretty much like machine translation
- Again, Encoder-Decoder paradigm
  - Insert the question to the encoder part
  - Model the answer at the decoder part
- Question answering with images also
  - Again, bleeding-edge research
  - How/where to add the image?
  - What has been working so far is to add the image only in the beginning

Q: John entered the living room, where he met Mary. She was drinking some wine and watching a movie. What room did John enter?

A: John entered the living room.

Q: what are the people playing?

A: They play beach football

Summary

- Recurrent Neural Networks (RNN) for sequences
- Backpropagation Through Time
- RNNs using Long Short-Term Memory (LSTM)
- Applications of Recurrent Neural Networks

Reading material & references

- http://www.deeplearningbook.org/
  - Part II: Chapter 10
- Excellent blog post on Backpropagation Through Time
  - http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/
  - http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-2-implementing-a-language-model-rnn-with-python-numpy-and-theano/
  - http://www.wildml.com/2015/10/recurrent-neural-networks-tutorial-part-3-backpropagation-through-time-and-vanishing-gradients/
- Excellent blog post explaining LSTMs
  - http://colah.github.io/posts/2015-08-Understanding-LSTMs/

[Pascanu2013] Pascanu, Mikolov, Bengio. On the difficulty of training Recurrent Neural Networks, JMLR, 2013

Next lecture

- Memory networks
- Recursive networks