Recurrent Neural Networks

Deep Learning – Lecture 5

Efstratios Gavves

UNIVERSITY OF AMSTERDAM

Sequential Data

So far, all tasks assumed ***stationary*** data

Neither all data, nor all tasks are stationary though

Sequential Data: Text

What

Sequential Data: Text

What about

Sequential Data: Text

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

Memory needed

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

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

Sequential data: Video

Quiz: Other sequential data?

Quiz: Other sequential data?

Time series data

- ❑ Stock exchange
- ❑ Biological measurements
- ❑ Climate measurements
- ❑ Market analysis

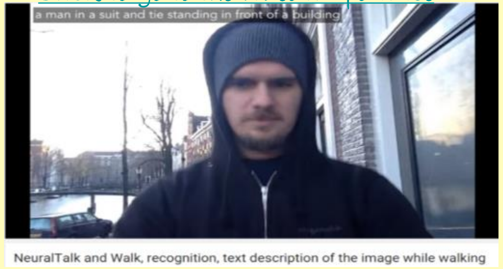
Speech/Music

User behavior in websites

...

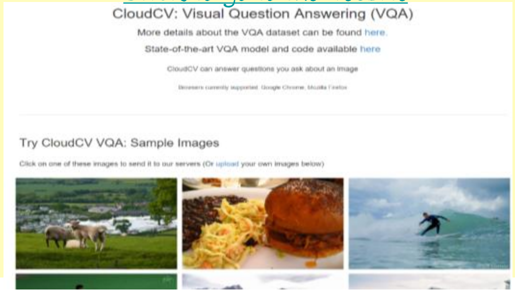
Applications

Click to go to the video in Youtube



NeuralTalk and Walk, recognition, text description of the image while walking

Click to go to the website


CloudCV: Visual Question Answering (VQA)

More details about the VQA dataset can be found here.

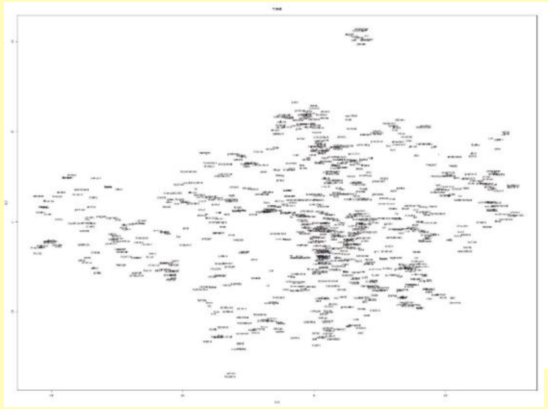
State-of-the-art VQA model and code available here

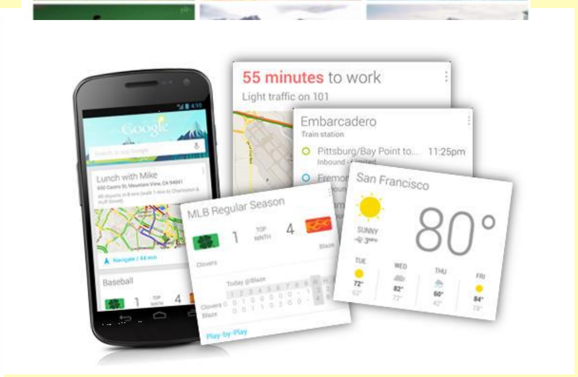
CloudCV can answer questions you ask about an image

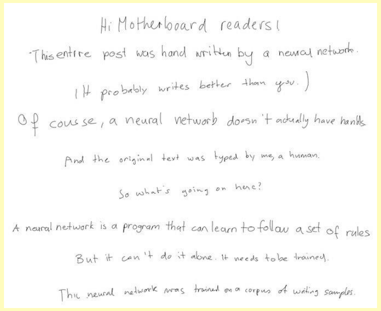
Browsers currently supported: Google Chrome, Mozilla Firefox

Try CloudCV VQA: Sample Images

Click on one of these images to send it to our servers (Or upload your own images below)






Hi Motherboard readers!

This entire post was hand written by a neural network.

(It probably writes better than you.)

Of course, a neural network doesn't actually have hands

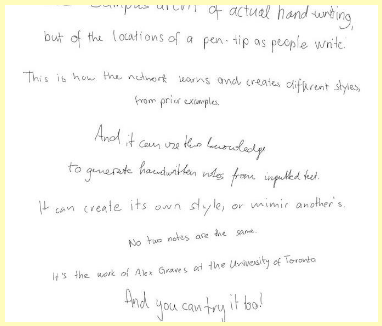
And the original text was typed by me, a human.

So what's going on here?

A neural network is a program that can learn to follow a set of rules

But it can't do it alone. It needs to be trained.

This neural network was trained on a corpus of writing samples.


of actual hand-writing,

but of the locations of a pen-tip as people write.

This is how the network learns and creates different styles,

from prior examples.

And it can use this knowledge

to generate handwritten notes from inputted text.

It can create its own style, or mimic another's.

No two notes are the same.

It's the work of Alex Graves at the University of Toronto

And you can try it too!

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

- "Погода сегодня хорошая"

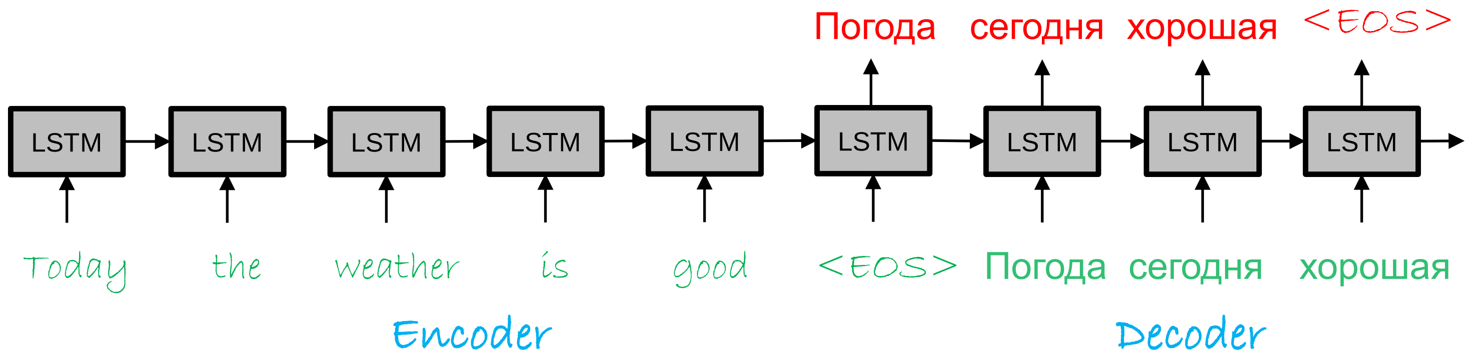


Image captioning

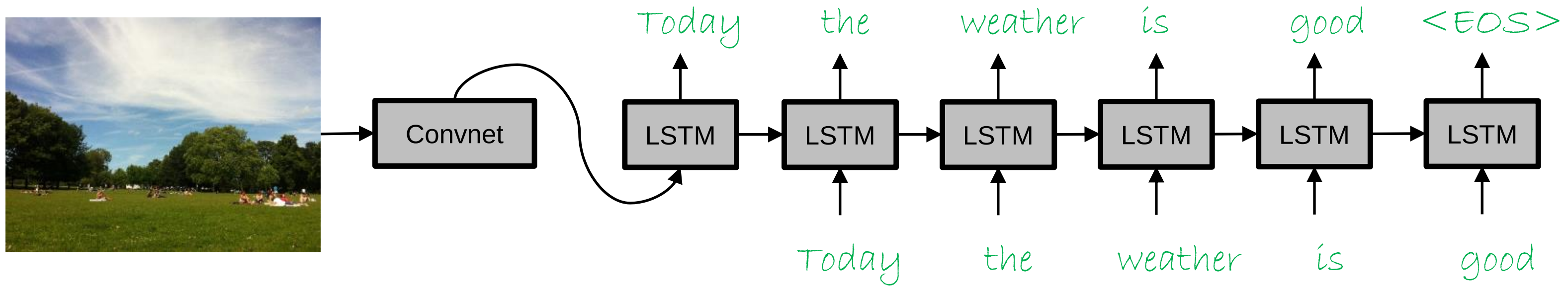
An image is a thousand words, literally!

Pretty much the same as machine transation

Replace the encoder part with the output of a Convnet

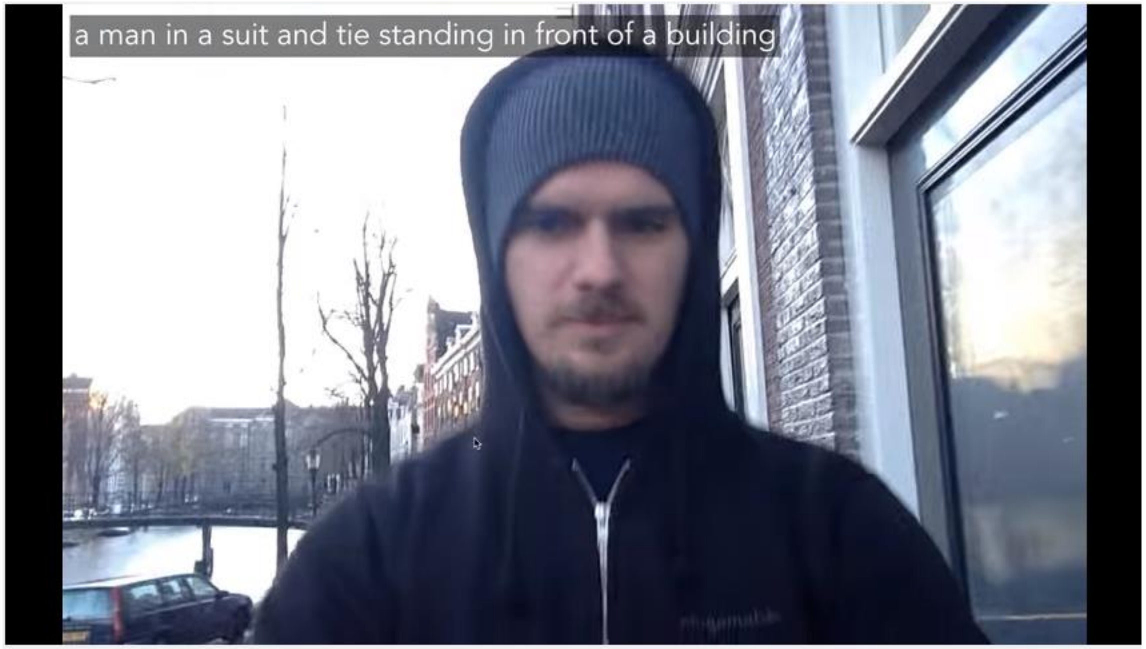
- E.g. use Alexnet or a VGG16 network

Keep the decoder part to operate as a translator



Demo

Click to go to the video in Youtube



NeuralTalk and Walk, recognition, text description of the image while walking

Question answering

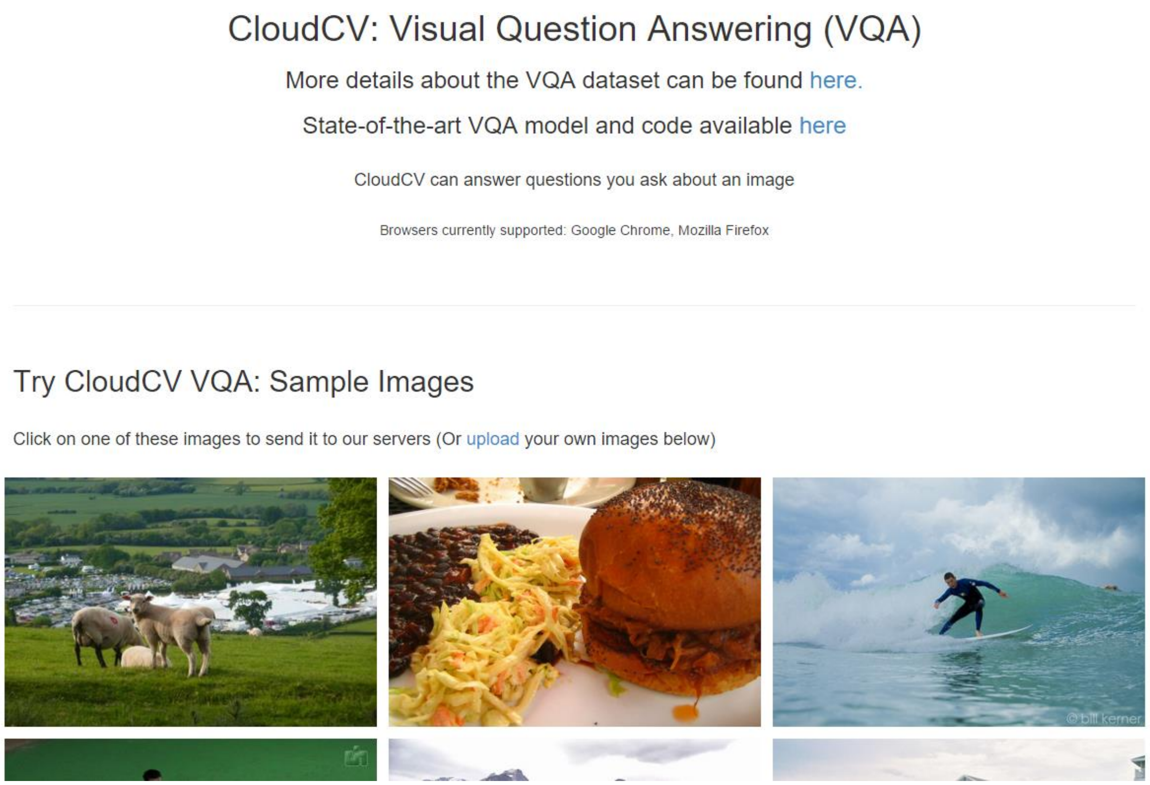
- Bleeding-edge research, no real consensus
  - Very interesting open, research problems
- Again, pretty much like machine translation
- Again, Encoder-Decoder paradigm
  - Insert the question to the encoder part
  - Model the answer at the decoder part
- Question answering with images also
  - Again, 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

Demo

Click to go to the website



CloudCV: Visual Question Answering (VQA)

More details about the VQA dataset can be found here.

State-of-the-art VQA model and code available here

CloudCV can answer questions you ask about an image

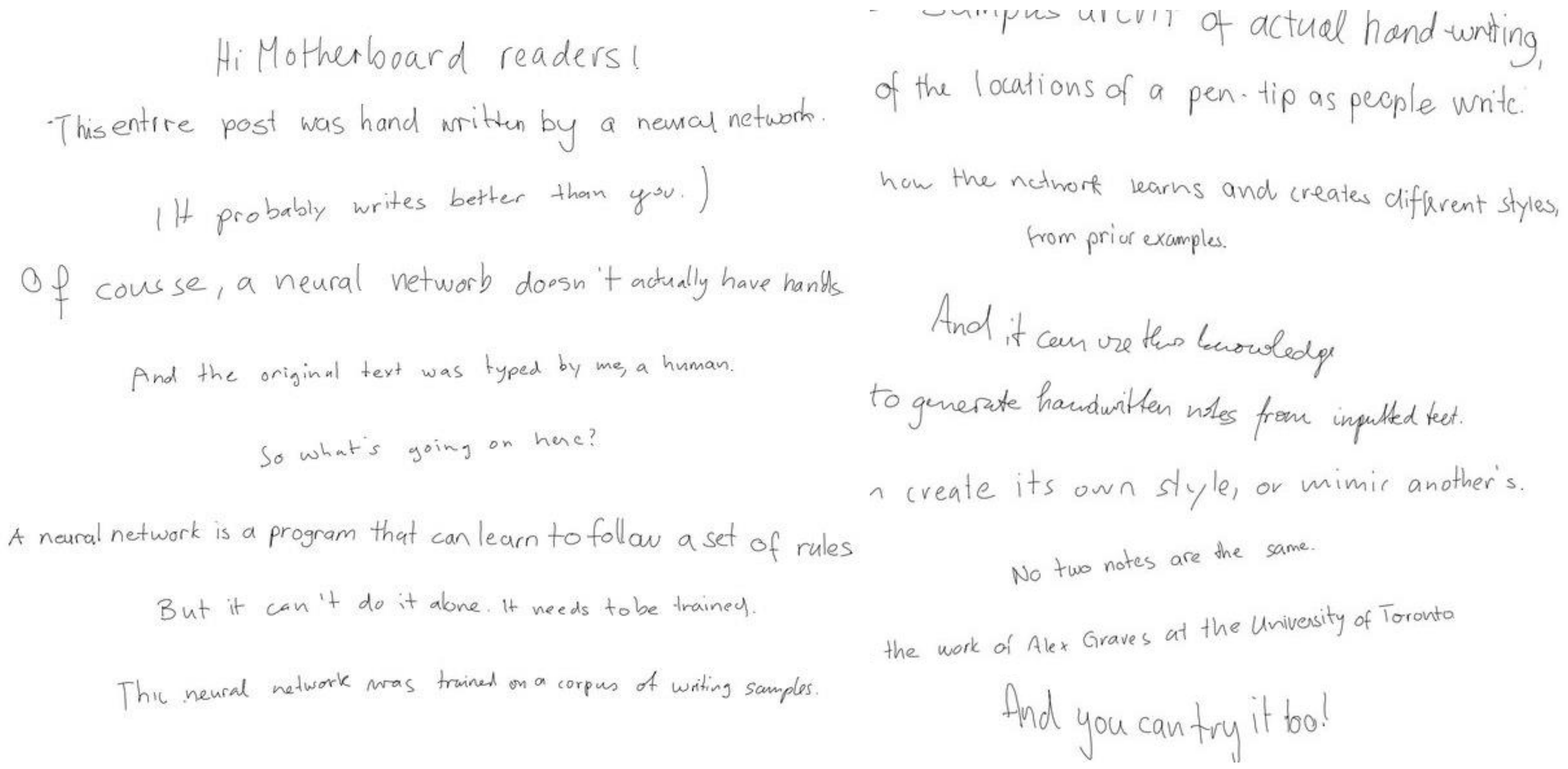
Browsers currently supported: Google Chrome, Mozilla Firefox

Try CloudCV VQA: Sample Images

Click on one of these images to send it to our servers (Or upload your own images below)

Handwriting

Click to go to the website



Hi Motherboard readers!

This entire post was hand written by a neural network.

(It probably writes better than you.)

Of course, a neural network doesn't actually have hands.

And the original text was typed by me, a human.

So what's going on here?

A neural network is a program that can learn to follow a set of rules

But it can't do it alone. It needs to be trained.

Thic neural network was trained on a corpus of writing samples.

[illegible] of actual hand writing,

of the locations of a pen-tip as people write.

how the network learns and creates different styles,

from prior examples.

And it can use this knowledge

to generate handwritten notes from inputted text.

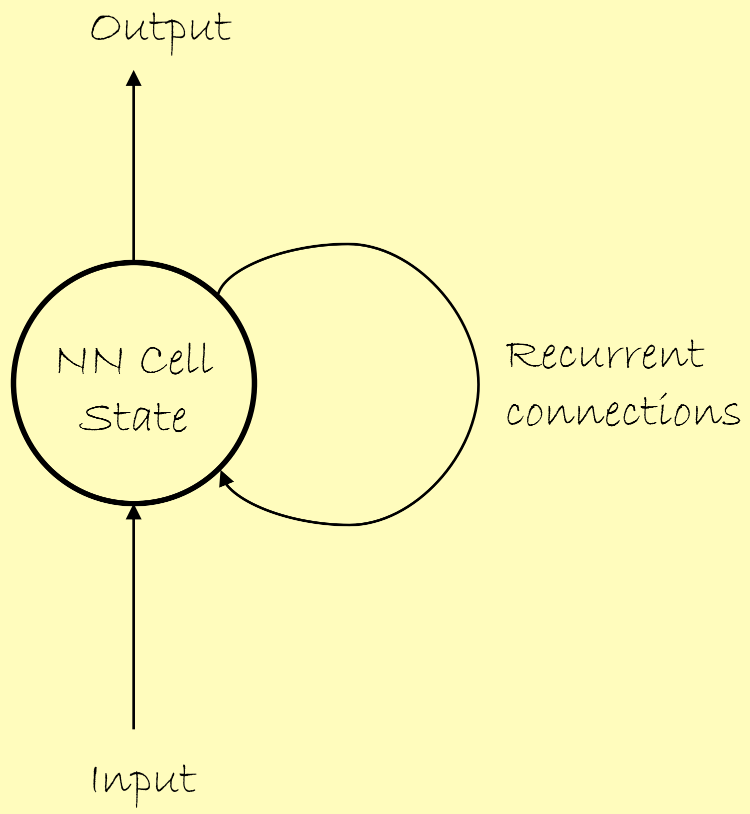
n create its own style, or mimic another's.

No two notes are the same.

the work of Alex Graves at the University of Toronto

And you can try it too!

Recurrent Networks



Sequences

Next data depend on previous data

Roughly equivalent to predicting what comes next

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

Whatabout inputsthat appear in sequences,suc as text?Coulda neural networkhandle suchmodalities?

Why sequences?

Parameters are reused → Fewer parameters → Easier modelling

Generalizes well to arbitrary lengths → Not constrained to specific length

$RecurrentModel($ I think, therefore, I am! $)$

$\equiv$

$RecurrentModel($ 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

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

Quiz: What is really a sequence?

Data inside a sequence are … ?

I am Bond , James ☐

McGuire

Bond

tired

am

!

Quiz: What is really a sequence?

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 ?

McGuire

Bond

I am Bond , James $\boxed{\text{Bond}}$ tired

am

!

$x_i \equiv$ 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

Quiz: Why not just indices?

One-hot representation | OR? | Index representation

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

I am James McGuire

$$x_{"I"} = 1$$
$$x_{"am"} = 2$$
$$x_{"James"} = 4$$
$$x_{"McGuire"} = 7$$

Quiz: Why not just indices?

One-hot representation OR? Index representation

$$\begin{array}{cccc} \text{I} & \text{am} & \text{James} & \text{McGuire} \\ \left[\begin{array}{c} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array}\right. & \begin{array}{c} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array} & \begin{array}{c} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{array} & \left.\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{array}\right] \\ x_{"I"} & x_{"am"} & x_{"James"} & x_{"McGuire"} \end{array}$$

I am James McGuire

$$q_{"I"} = 1$$
$$q_{"am"} = 2$$
$$q_{"James"} = 4$$
$$q_{"McGuire"} = 7$$

$$distance(x_{"am"}, x_{"McGuire"}) = 1$$
$$distance(x_{"I"}, x_{"am"}) = 1$$

$$distance(q_{"am"}, q_{"McGuire"}) = 5$$
$$distance(q_{"I"}, q_{"am"}) = 1$$

No, because then some words get closer together for no good reason (artificial bias between words)

Memory

A representation of the past

A memory must project information at timestep t on a latent space c_t using parameters θ

Then, re-use the projected information from t at $t+1$

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

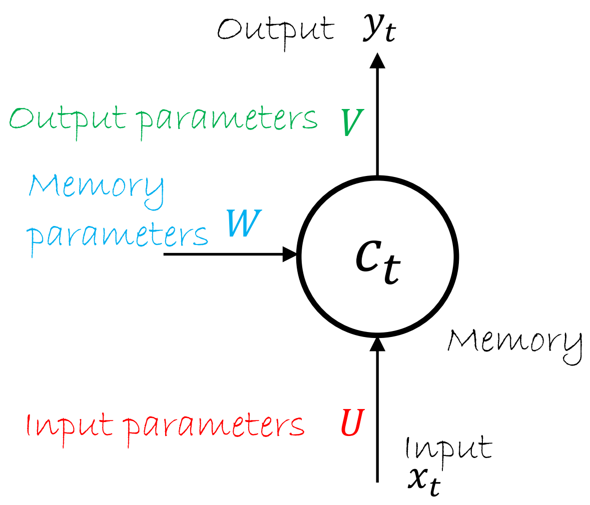
Memory parameters θ are shared for all timesteps $t = 0, \ldots$

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

Memory as a Graph

Simplest model

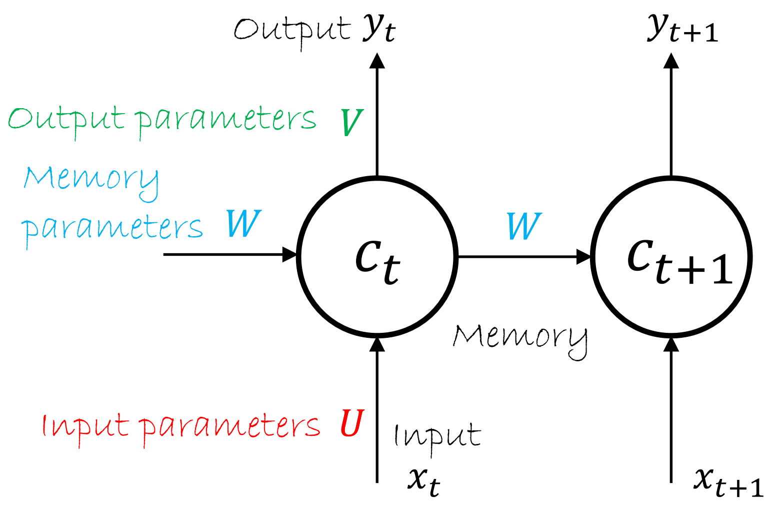
- Input with parameters U
- Memory embedding with parameters W
- Output with parameters V



Memory as a Graph

Simplest model

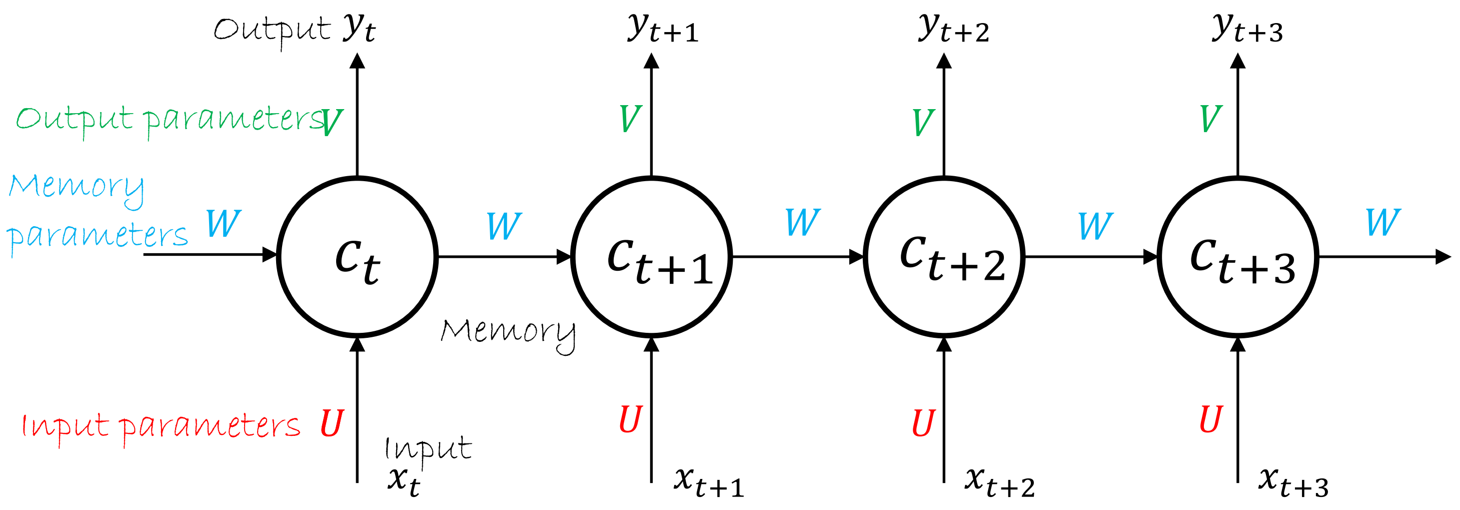
- Input with parameters U
- Memory embedding with parameters W
- Output with parameters V



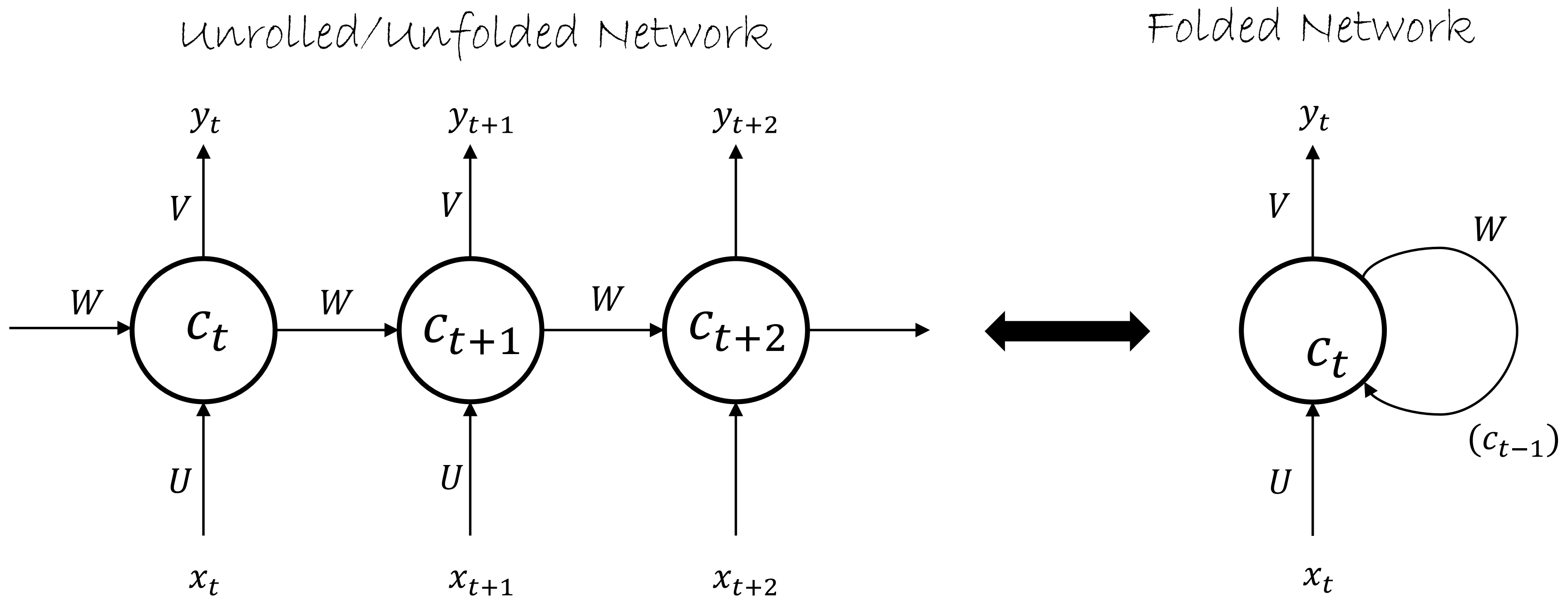
Memory as a Graph

Simplest model

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



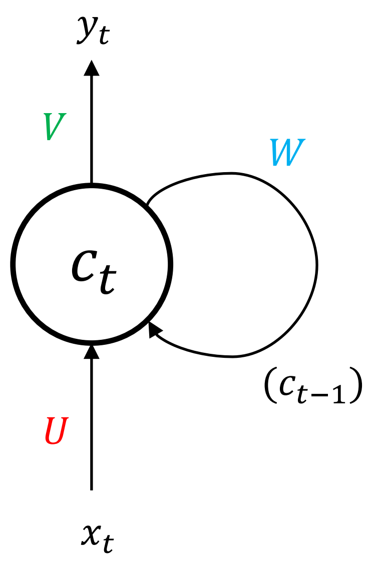
Folding the memory



Recurrent Neural Network (RNN)

Only **two** 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: $[3 \times 1]$, W: $[3 \times 3]$
- An input projection of 3 dimensions
  - U: $[3 \times 5]$
- An output projections of 10 dimensions
  - V: $[10 \times 3]$

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

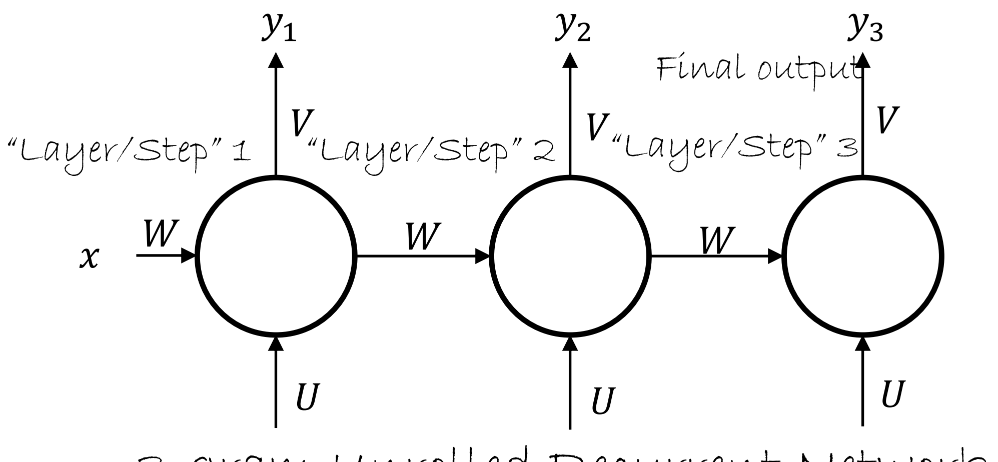
$$c_t = \tanh(U\, x_t + W c_{t-1})$$

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

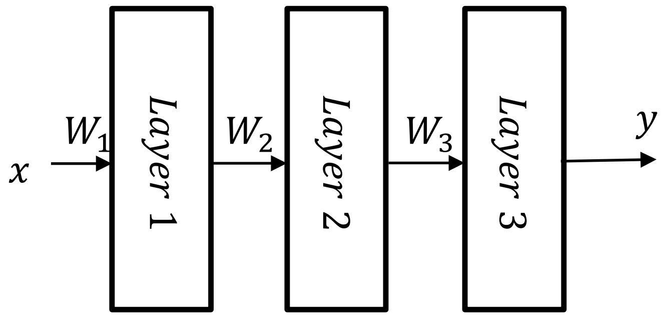
Rolled Network vs. MLP?

What is really different?

- Steps instead of layers
- Step parameters shared in Recurrent Network
- In a Multi-Layer Network parameters are different



3-gram Unrolled Recurrent Network

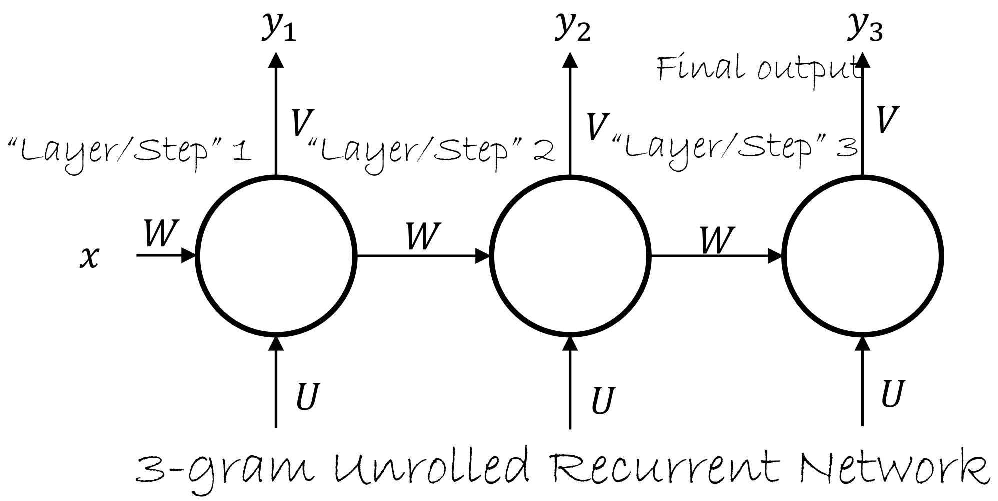


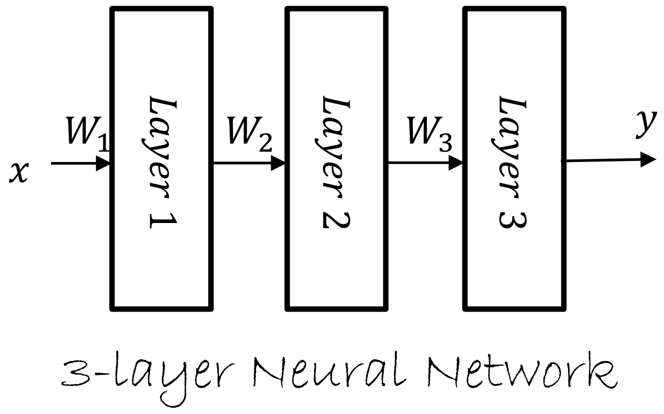
3-layer Neural Network

Rolled Network vs. MLP?

What is really different?

- Steps instead of layers
- Step parameters shared in Recurrent Network
- In a Multi-Layer Network parameters are different



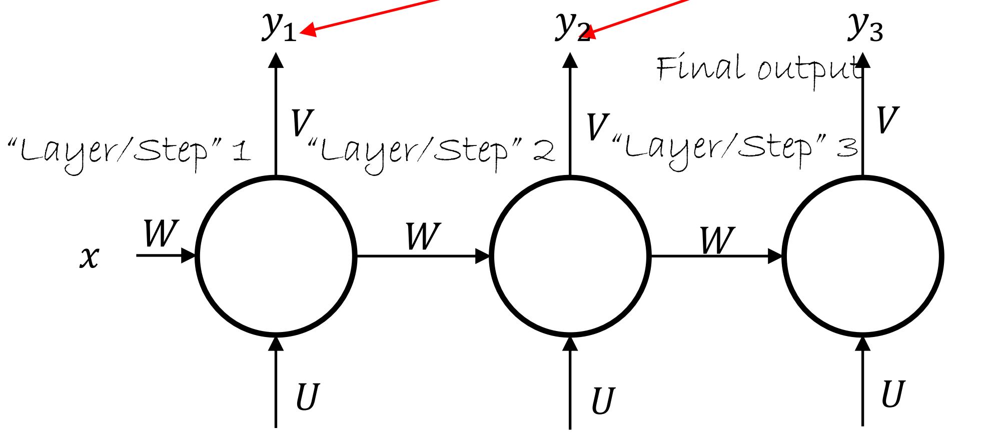


Rolled Network vs. MLP?

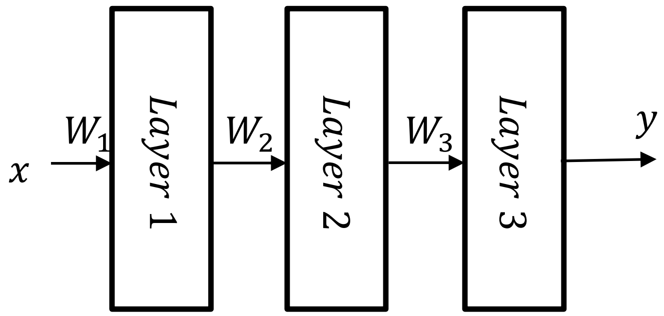
What is really different?

- Steps instead of layers
- Step parameters shared in Recurrent Network
- In a Multi-Layer Network parameters are different

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



3-gram Unrolled Recurrent Network

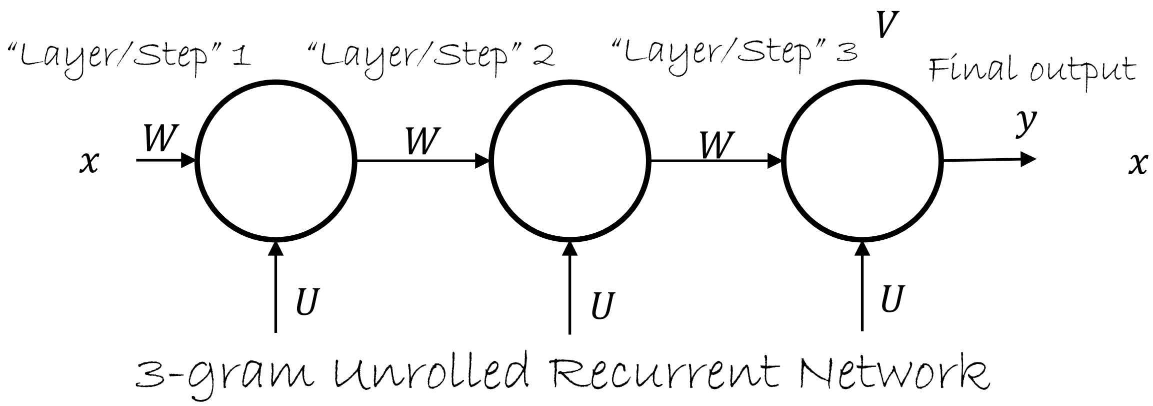


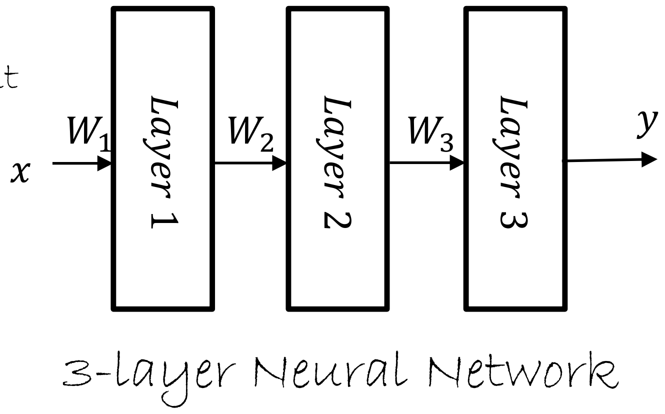
3-layer Neural Network

Rolled Network vs. MLP?

What is really different?

- Steps instead of layers
- Step parameters shared in Recurrent Network
- In a Multi-Layer Network parameters are different





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

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

$$\mathcal{L} = L(c_T(c_{T-1}(\ldots(c_1(x_1, c_0; W); W); W); W)$$

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

$$\frac{\partial \mathcal{L}}{\partial c_t} \frac{\partial c_t}{\partial c_\tau} = \frac{\partial \mathcal{L}}{\partial c_t} \cdot \underbrace{\frac{\partial c_t}{\partial c_{t-1}} \cdot \frac{\partial c_{t-1}}{\partial c_{t-2}}}_{Rest \;\rightarrow\; \text{short-term factors}} \cdot \ldots \cdot \underbrace{\frac{\partial c_{\tau+1}}{\partial c_\tau}}_{t \gg \tau \;\rightarrow\; \text{long-term factors}} \leq \eta^{t-\tau} \frac{\partial \mathcal{L}_t}{\partial c_t}$$

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

Exploding/Vanishing gradients

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

$$\left.\frac{\partial \mathcal{L}}{\partial c_t} = \frac{\partial \mathcal{L}}{\partial \mathrm{c_T}} \cdot \underbrace{\frac{\partial c_T}{\partial \mathrm{c_{T-1}}}}_{>1} \cdot \underbrace{\frac{\partial c_{T-1}}{\partial \mathrm{c_{T-2}}}}_{>1} \cdot \ldots \cdot \underbrace{\frac{\partial c_1}{\partial \mathrm{c}_{c_t}}}_{>1}\right\} \quad \frac{\partial \mathcal{L}}{\partial W} \gg 1 \Longrightarrow$$ Exploding gradient

Vanishing gradients

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

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

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

Vanishing gradients

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

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

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

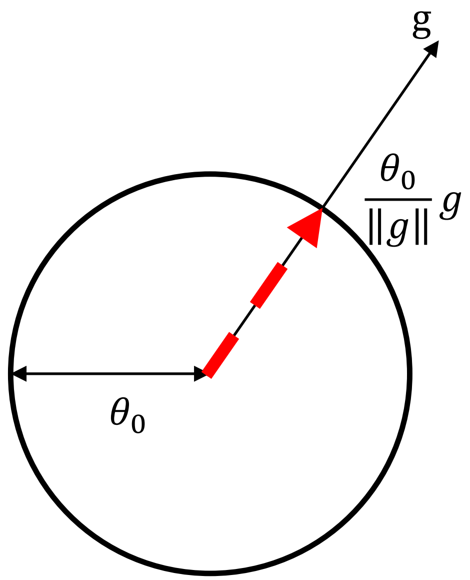
Long-term dependencies get exponentially smaller weights

Gradient clipping for exploding gradients

Scale the gradients to a threshold

Pseudocode

```
1.  g ← $\frac{\partial \mathcal{L}}{\partial W}$
2.  if $\|g\| > \theta_0$:
        g ← $\frac{\theta_0}{\|g\|} g$
    else:
        print('Do nothing')
```



Simple, but works!

Rescaling vanishing gradients?

Not good solution

Weights are shared between timesteps → Loss summed over timesteps

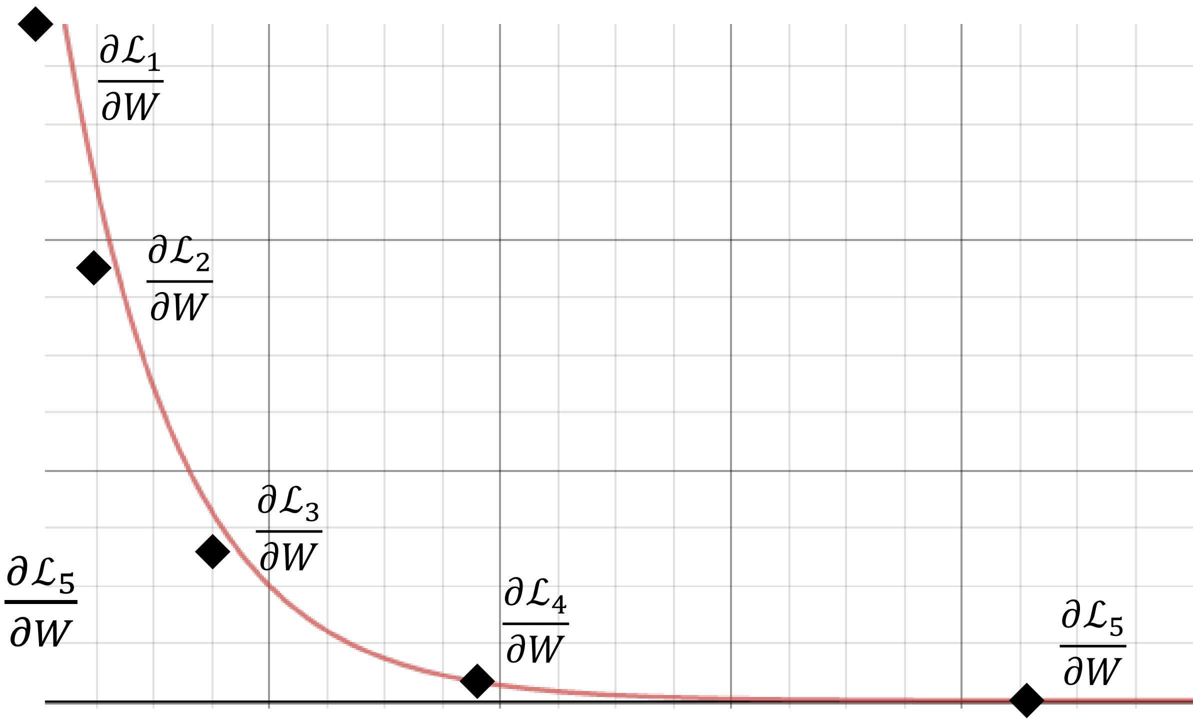
$$\mathcal{L} = \sum_t \mathcal{L}_t \quad \Longrightarrow \quad \frac{\partial \mathcal{L}}{\partial W} = \sum_t \frac{\partial \mathcal{L}_t}{\partial W}$$

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

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 $\frac{\partial \mathcal{L}_1}{\partial W} \propto 1, \frac{\partial \mathcal{L}_2}{\partial W} \propto 1/10, \frac{\partial \mathcal{L}_3}{\partial W} \propto 1/100, \frac{\partial \mathcal{L}_4}{\partial W} \propto 1/1000, \frac{\partial \mathcal{L}_5}{\partial W} \propto 1/10000$

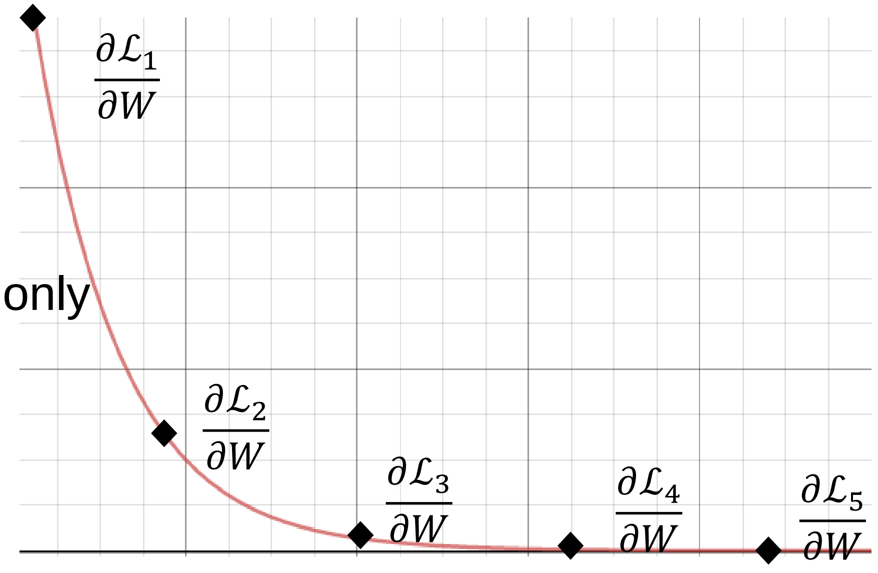
$$\frac{\partial \mathcal{L}}{\partial W} = \sum_r \frac{\partial \mathcal{L}_r}{\partial W} = 1.1111$$

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 ∝ Chaotic systems

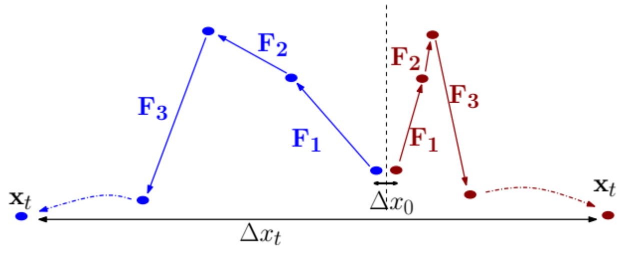


Figure 4. This diagram illustrates how the change in $\mathbf{x}_t$, $\Delta\mathbf{x}_t$, can be large for a small $\Delta\mathbf{x}_0$. The blue vs red (left vs right) trajectories are generated by the same maps $F_1, F_2, \ldots$ for two different initial states.

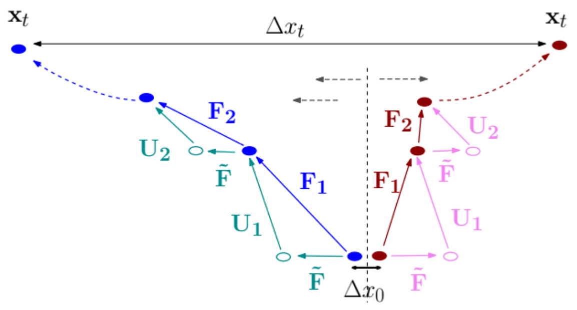


Figure 5. Illustrates how one can break apart the maps $F_1, ..F_t$ into a constant map $\tilde{F}$ and the maps $U_1, .., U_t$. The dotted vertical line represents the boundary between basins of attraction, and the straight dashed arrow the direction of the map $\tilde{F}$ on each side of the boundary.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

Fixing vanishing gradients

Regularization on the recurrent weights

- Force the error signal not to vanish

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

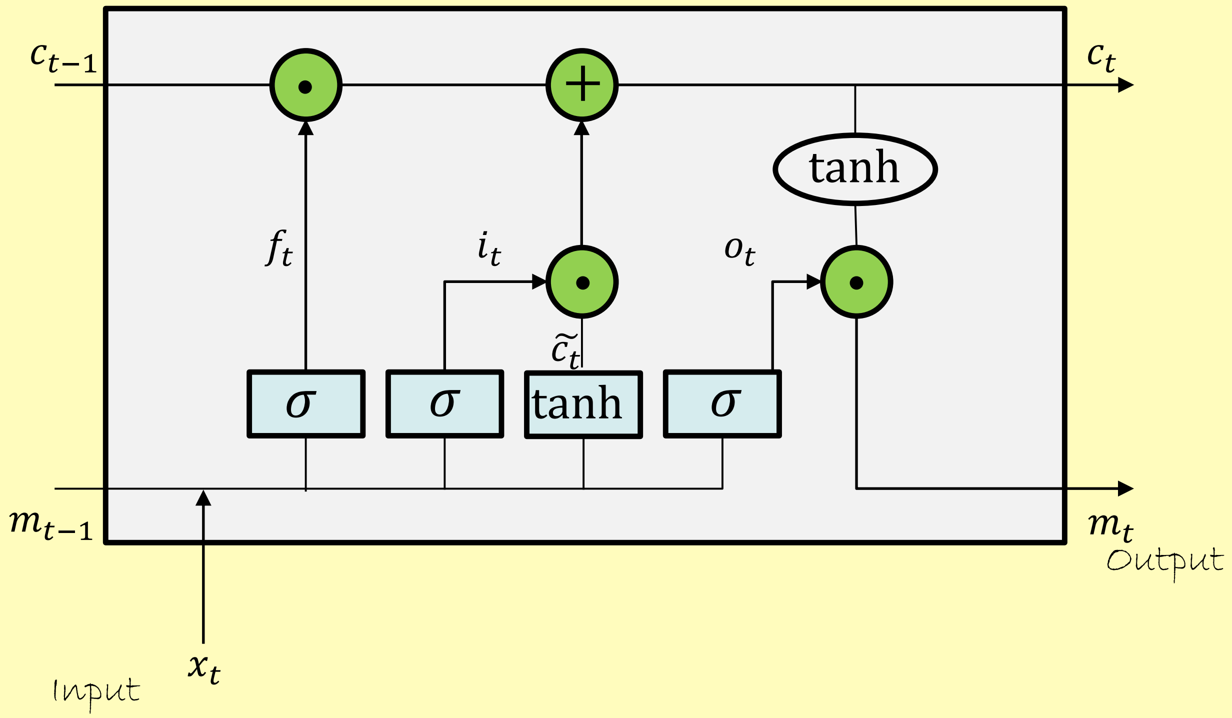
Advanced recurrent modules

Long-Short Term Memory module

Gated Recurrent Unit module

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

Advanced Recurrent Nets



How to fix the vanishing gradients?

Error signal over time must have not too large, not too small norm

Let's have a look at the loss function

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

$$\frac{\partial c_t}{\partial c_\tau} = \prod_{t \geq k \geq \tau} \frac{\partial c_k}{\partial c_{k-1}}$$

How to fix the vanishing gradients?

Error signal over time must have not too large, not too small norm

Let’s have a look at the loss function

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

$$\frac{\partial c_t}{\partial c_\tau} = \prod_{t \geq k \geq \tau} \frac{\partial c_k}{\partial c_{k-1}}$$

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 gradient equal to 1

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

$$\frac{\partial c_t}{\partial c_\tau} = \prod_{t \geq k \geq \tau} \frac{\partial c_k}{\partial c_{k-1}}$$

- Identify function has a gradient equal to 1

By doing so, gradients do not become too small not too large

Long Short-Term Memory

LSTM: Beefed up RNN

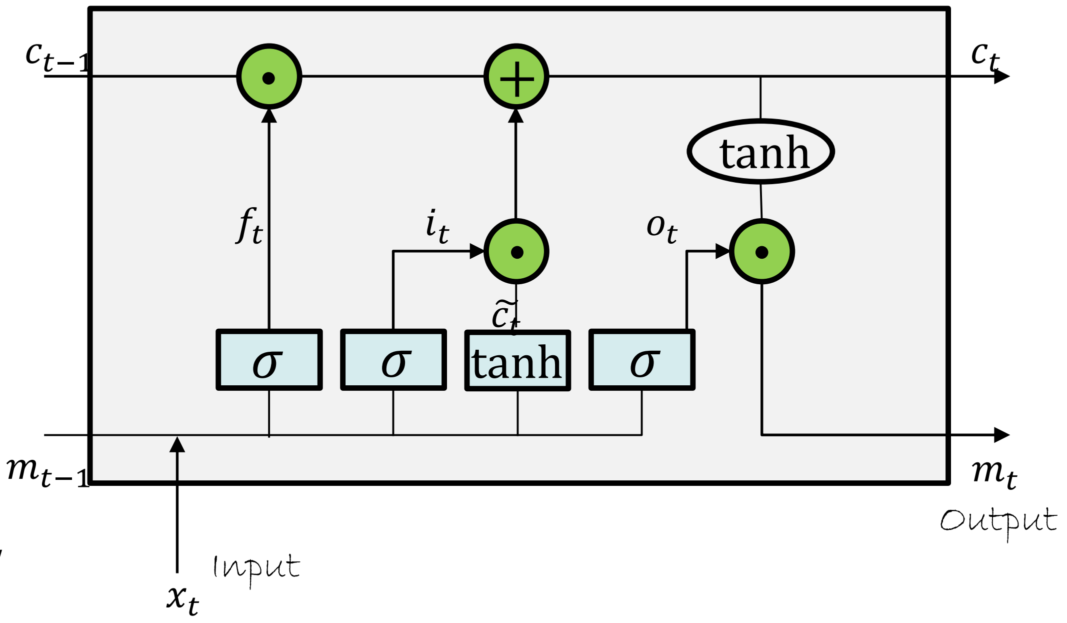
Simple RNN

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

LSTM

$$\begin{aligned}
i &= \sigma\left(x_t U^{(i)} + m_{t-1} W^{(i)}\right) \\
f &= \sigma\left(x_t U^{(f)} + m_{t-1} W^{(f)}\right) \\
o &= \sigma\left(x_t U^{(o)} + m_{t-1} W^{(o)}\right) \\
\tilde{c}_t &= \tanh(x_t U^{(g)} + m_{t-1} W^{(g)}) \\
c_t &= c_{t-1} \odot f + \tilde{c}_t \odot i \\
m_t &= \tanh(c_t) \odot o
\end{aligned}$$

- The previous state c_{t-1} is connected to new c_t with no nonlinearity (identity function).
- The only other factor is the forget gate f which rescales the previous LSTM state.



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

Cell state

The cell state carries the essential information over time

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

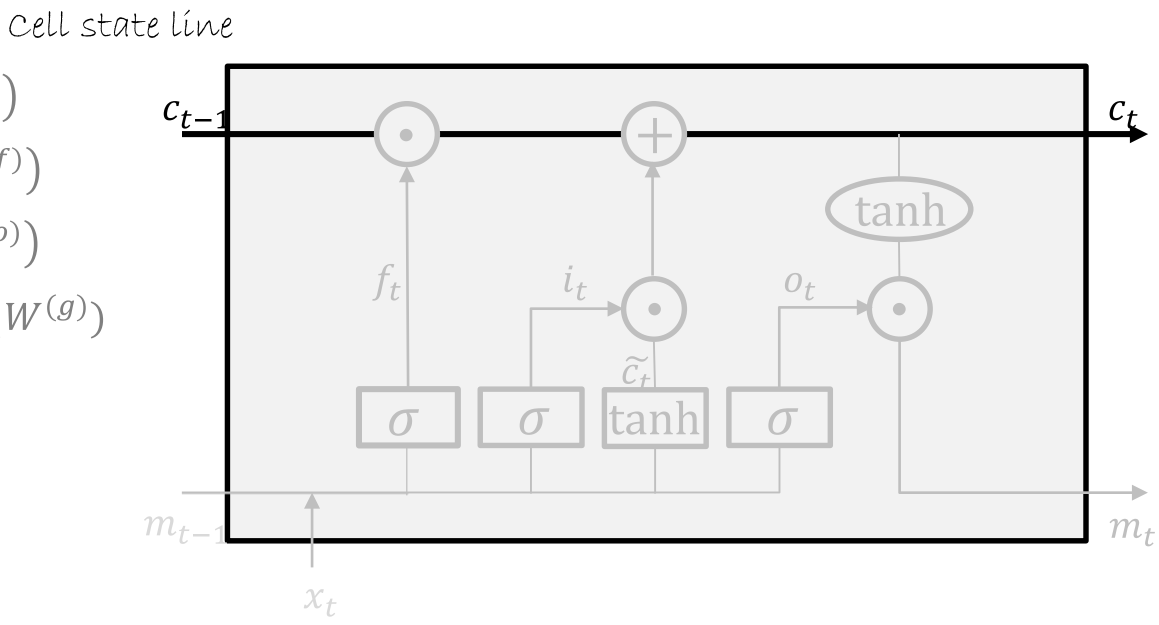
$$f = \sigma\left(x_t U^{(f)} + m_{t-1} W^{(f)}\right)$$

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

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

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

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



LSTM nonlinearities

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

$\tanh \in (-1,1)$: recurrent nonlinearity

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

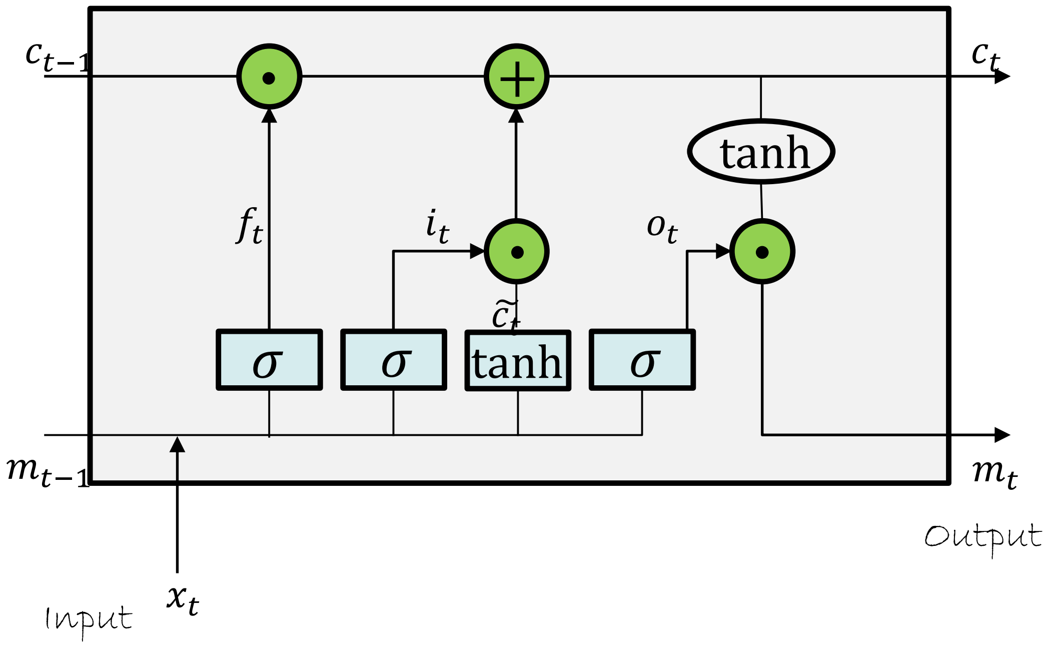
$$f = \sigma\left(x_t U^{(f)} + m_{t-1} W^{(f)}\right)$$

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

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

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

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



LSTM Step-by-Step: Step (1)

E.g. LSTM on *“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

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

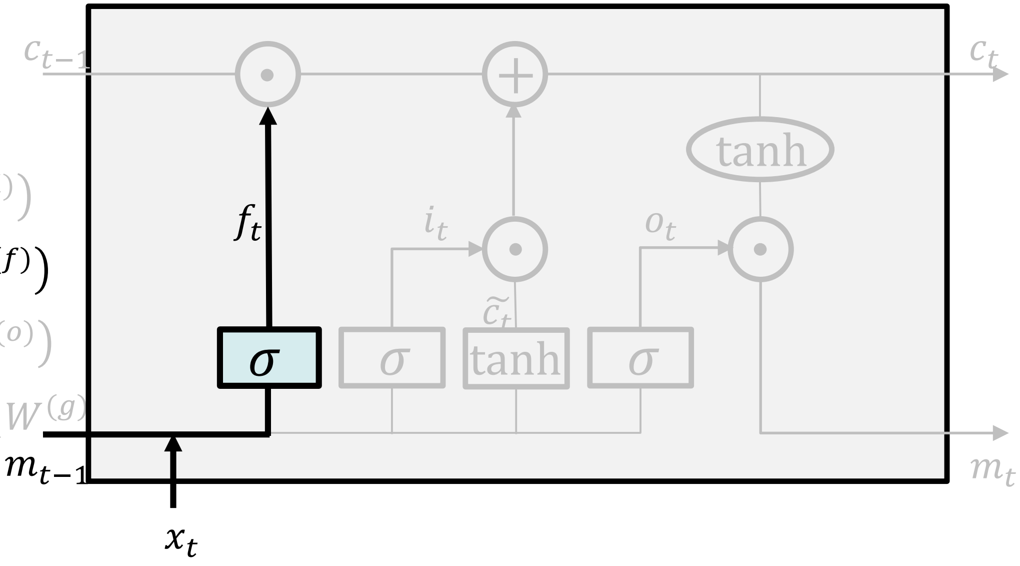
$$f_t = \sigma\left(x_t U^{(f)} + m_{t-1} W^{(f)}\right)$$

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

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

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

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



LSTM Step-by-Step: Step (2)

Decide what new information is relevant from the new input and should be add to the new memory

- Modulate the input i_t
- Generate candidate memories $\widetilde{c}_t$

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

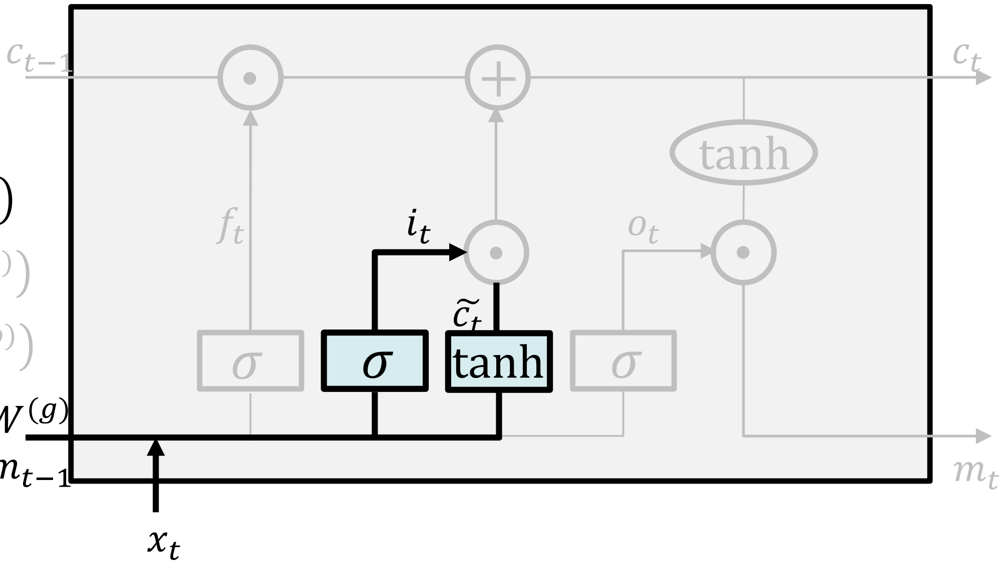
$$f_t = \sigma\left(x_t U^{(f)} + m_{t-1} W^{(f)}\right)$$

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

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

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

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



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

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

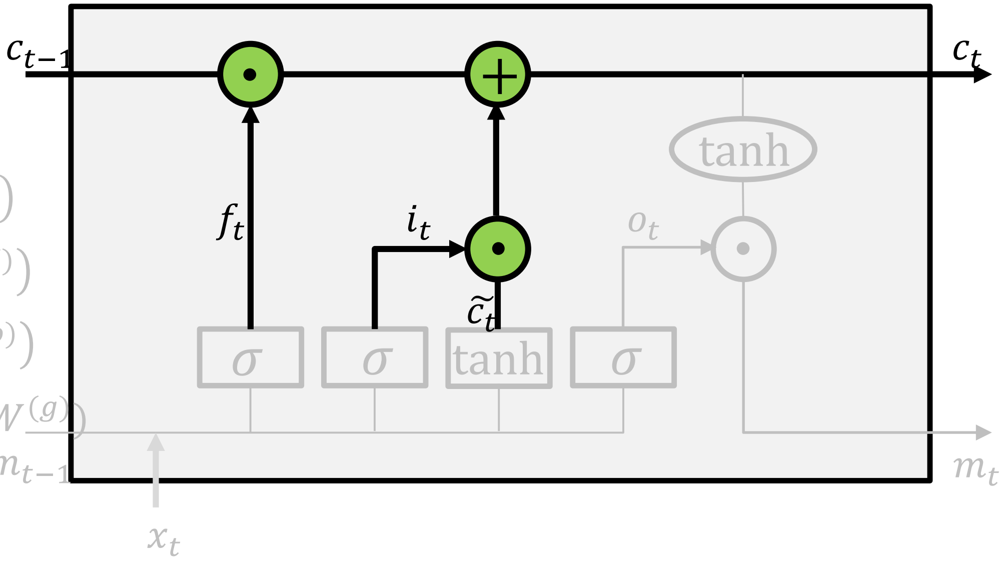
$$f_t = \sigma\left(x_t U^{(f)} + m_{t-1} W^{(f)}\right)$$

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

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

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

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



LSTM Step-by-Step: Step (4)

Modulate the output

- Does the new cell state relevant? → Sigmoid 1
- If not → Sigmoid 0

Generate the new memory

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

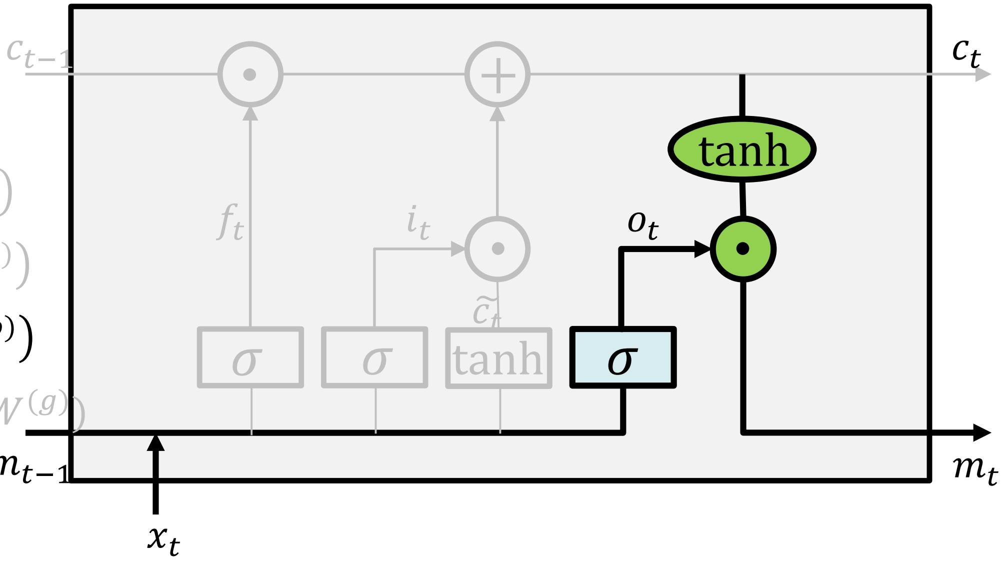
$$f_t = \sigma\left(x_t U^{(f)} + m_{t-1} W^{(f)}\right)$$

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

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

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

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

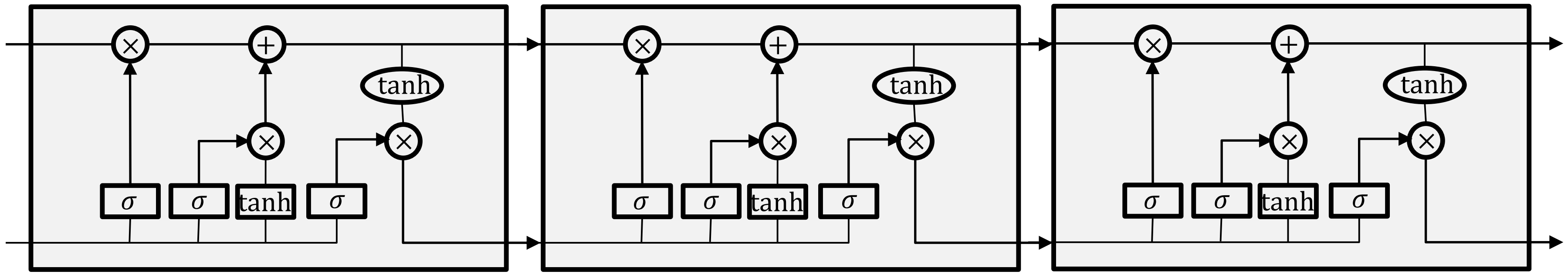


LSTM Unrolled Network

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



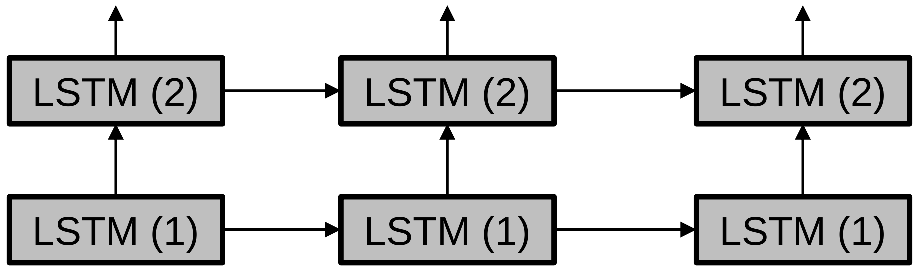
Beyond RNN & LSTM

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

Take-away message

Recurrent Neural Networks (RNN) for sequences

Backpropagation Through Time

Vanishing and Exploding Gradients and Remedies

RNNs using Long Short-Term Memory (LSTM)

Applications of Recurrent Neural Networks

Thank you!