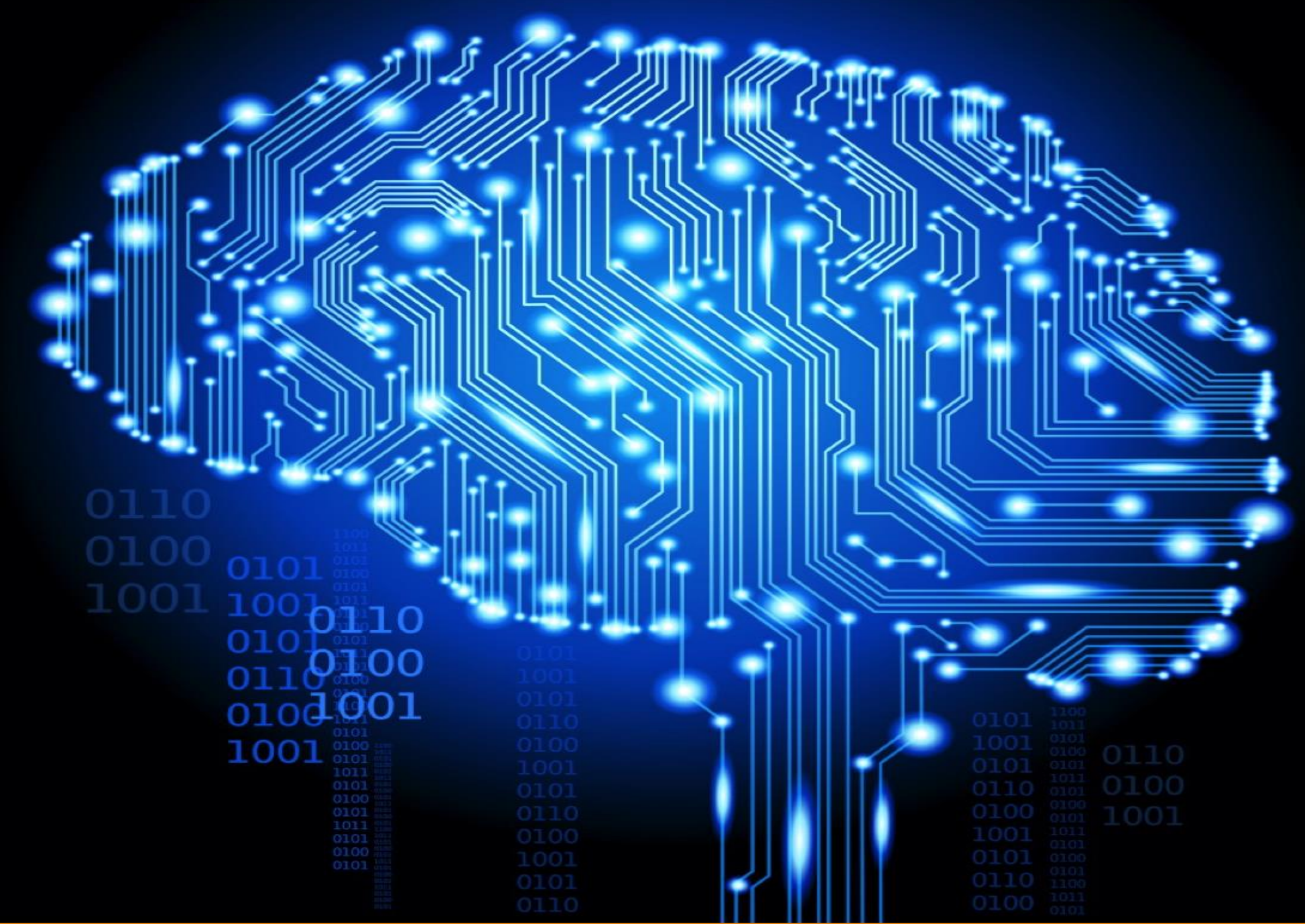


Lecture 6: Recurrent & Graph Neural Networks

Efstratios Gavves

Lecture overview

- Sequential data
- Recurrent Neural Networks
- Backpropagation through time
- Exploding and vanishing gradients
- LSTMs and variants
- Encoder-Decoder Architectures
- Graph Neural Networks

Sequence data

Sequence applications

Example of sequential data

- Videos
- Other?

Example of sequential data

- Videos
- Other?
- Time series data
  - Stock exchange
  - Biological measurements
  - Climate measurements
  - Market analysis
- Speech/Music
- User behavior in websites
- .....

Applications

- Machine translation
- Image captioning
- Question answering
- Video generation
- Speech synthesis
- Speech recognition

A sequence of probabilities

- Sequence → Chain rule of probabilities

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

- For instance, let's model that "`This is the best course!`"

$$p(\texttt{This is the best course!}) =$$
$$= p(\texttt{This}) \cdot$$
$$p(\texttt{is}|\texttt{This}) \cdot$$
$$p(\texttt{the}|\texttt{This is}) \cdot \dots \cdot$$
$$p(\texttt{!}|\texttt{This is the best course})$$

What is the problem with sequences?

- ???

What is the problem with sequences?

- Sequences might be of arbitrary or even infinite lengths
- Infinite parameters?

What is the problem with sequences?

- Sequences might be of arbitrary or even infinite lengths
- Infinite parameters?
- No, better share and reuse parameters
- RecurrentModel(`I think, therefore, I am.` | θ)

  can be reused also for

  RecurrentModel(`Everything is repeated in circles. History is a Master because it teaches that it doesn't exist. It is the permutations that matter`| θ)
- For a ConvNet that is not straightforward
- Why?

What is the problem with sequences?

- Sequences might be of arbitrary or even infinite lengths
- Infinite parameters?
- No, better share and reuse parameters
- RecurrentModel(`I think, therefore, I am.` | θ)

  can be reused also for

  RecurrentModel(`Everything is repeated in circles. History is a Master because it teaches that it doesn’t exist. It is the permutations that matter`| θ)
- For a ConvNet that is not straightforward
- Why? Fixed dimensionalities

Some properties of sequences?

Some properties of sequences

- Data inside a sequence are non identically, independently distributed (IID)
  - The next “word” depends on the previous “words”
  - Ideally on all of them
- We need context, and we need memory!

- **Big question:** How to model context and memory ?

I am Bond , James Bond

McGuire
Bond
tired
am
!

Properties of sequences

- Data inside a sequence are non identically, independently distributed (IID)
  - The next “word” depends on the previous “words”
  - Ideally on all of them
- We need context, and we need memory!

- **Big question:** How to model context and memory ?

McGuire

Bond

I am Bond , James Bond

tired

am

!

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

Why not indices instead of one-hot vectors?

One-hot representation

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

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

OR?

Index representation

I am James McGuire

$$x_{"I"} = 1$$

$$x_{"am"} = 2$$

$$x_{"James"} = 4$$

$$x_{"McGuire"} = 7$$

Why not indices instead of one-hot vectors?

One-hot representation | OR? | Index representation

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

$$
\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}
$$

I am James McGuire

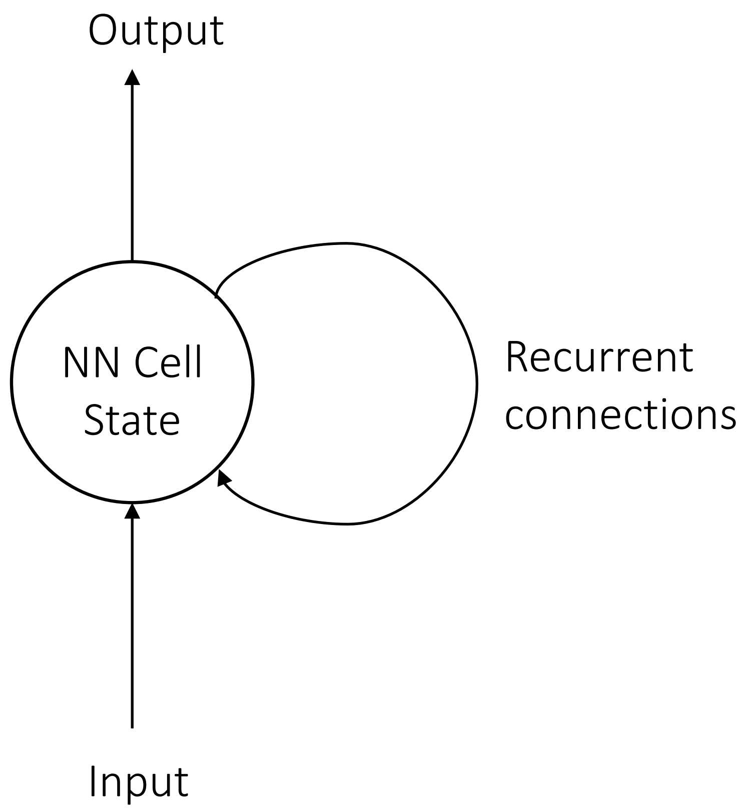
$$x_{"I"} = 1$$
$$x_{"am"} = 2$$
$$x_{"James"} = 4$$
$$x_{"McGuire"} = 7$$

$$\ell_2(x_{am}, x_{McQuire}) = \sqrt{2}$$
$$=$$
$$\ell_2(x_I, x_{am}) = \sqrt{2}$$

$$\ell_2(x_{am}, x_{McQuire}) = (7-2)^2 = 5$$
$$\neq$$
$$\ell_2(x_I, x_{am}) = (2-1)^2 = 1$$

Recurrent Neural Networks

Backprop through time



Memory

- Memory is a mechanism that learns a representation of the past
- At timestep t project all previous information $1, \dots, t$ onto a latent space c_t
  - Memory controlled by a neural network h_θ with shared parameters θ
- Then, at timestep $t+1$ re-use the parameters θ and the previous c_t

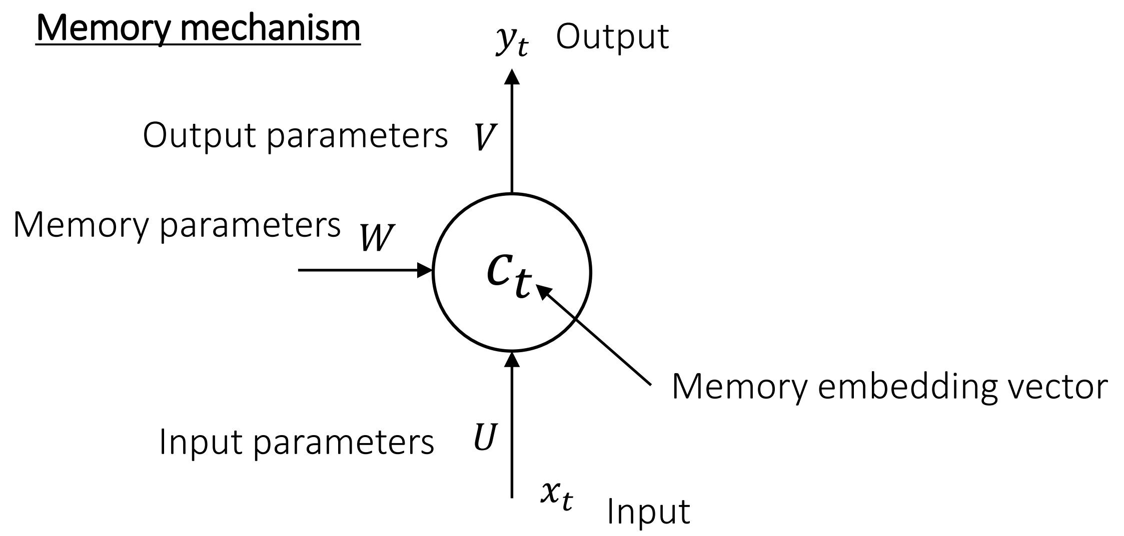
$$c_{t+1} = h_\theta(x_{t+1}, c_t)$$

$$\dots$$

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

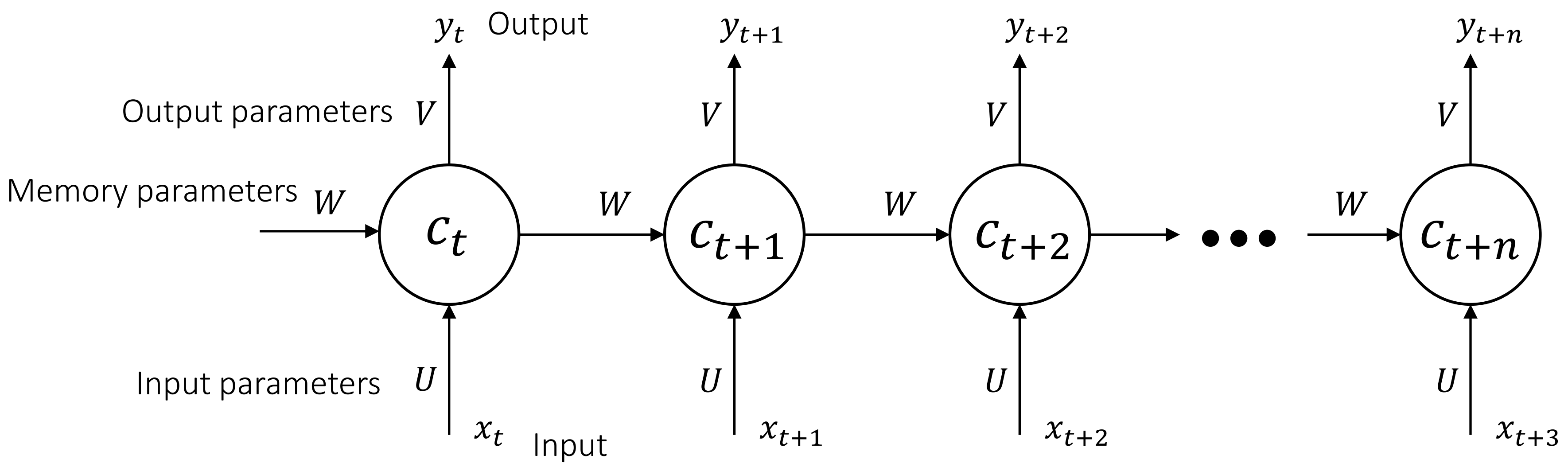
A graphical representation of memory

- In the simplest case, what are the Inputs/Outputs of our system
- Sequence inputs → we model them with parameters U
- Sequence outputs → we model them with parameters V
- Memory I/O → we model it with parameters W

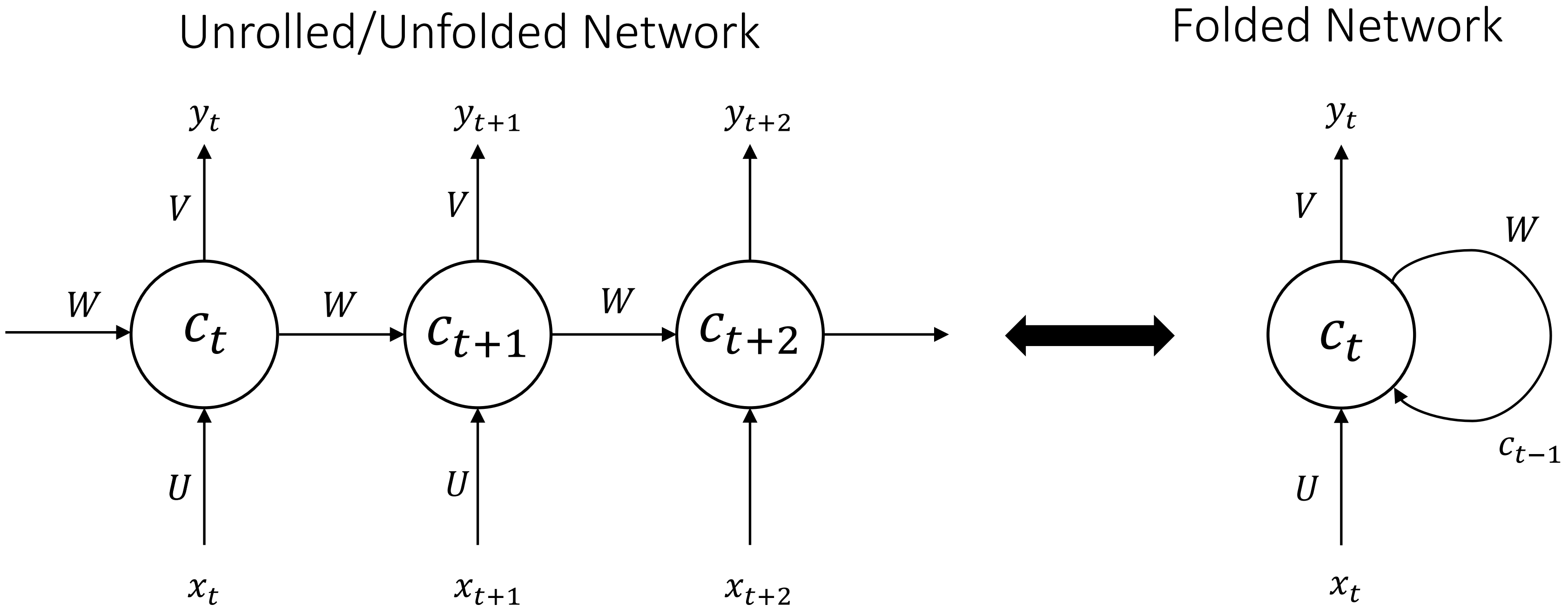


A graphical representation of memory

- In the simplest case, what are the Inputs/Outputs of our system
- Sequence inputs → we model them with parameters U
- Sequence outputs → we model them with parameters V
- Memory I/O → we model it with parameters W



Folding the memory



Recurrent Neural Networks - RNNs

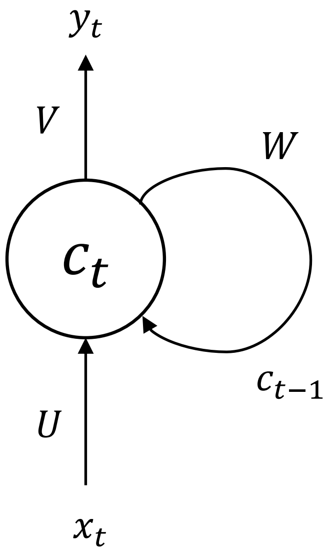
- Basically, two equations

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

- And a loss function

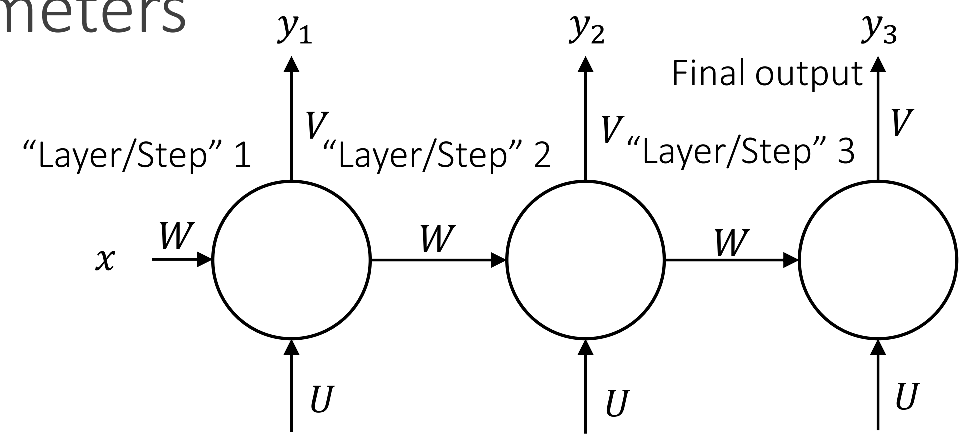
$$\mathcal{L} = \sum_t \mathcal{L}_t(y_t, y_t^*)$$
$$= \sum_t y_t^* \log y_t$$

assuming the cross-entropy loss function

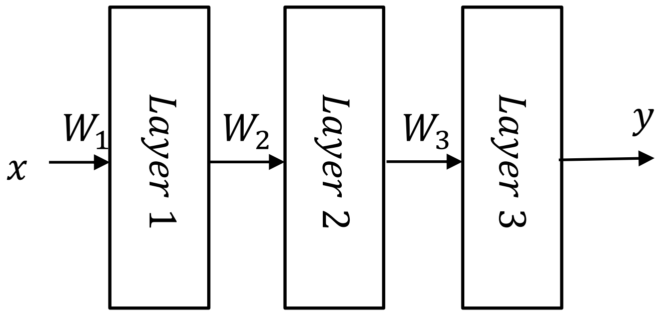


RNNs vs MLPs

- Is there a big difference?
- Instead of layers → Steps
- Outputs at every step → MLP outputs in every layer possible
- Main difference: Instead of layer-specific parameters → Layer-shared parameters



3-gram Unrolled Recurrent Network



3-layer Neural Network

Hmm, layers share parameters ?!?

- How is the training done? Does Backprop remain the same?

Hmm, layers share parameters ?!?

- How is the training done? Does Backprop remain the same?
- Basically, chain rule
  - So, again the same concept
- Yet, a bit more tricky this time, as the gradients survive over time

Backpropagation through time

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

- Let's say we focus on the third timestep loss

$$\frac{\partial \mathcal{L}}{\partial V} = \cdots$$
$$\frac{\partial \mathcal{L}}{\partial W} = \cdots$$
$$\frac{\partial \mathcal{L}}{\partial U} = \cdots$$

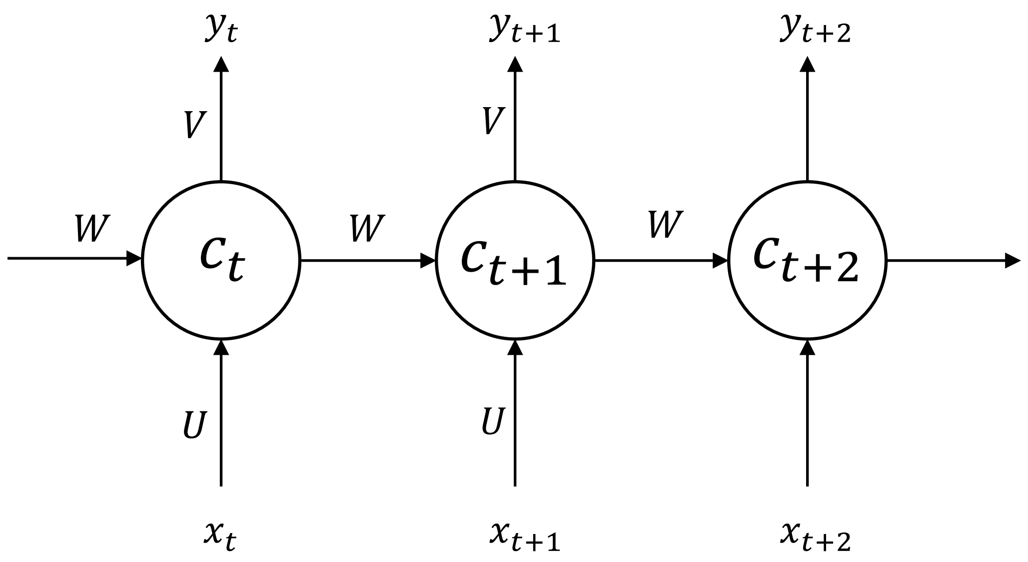
Backpropagation through time: $\partial \mathcal{L}_t / \partial V$

- Expanding the chain rule

$$\frac{\partial \mathcal{L}_t}{\partial V} = \frac{\partial \mathcal{L}_t}{\partial y_{t_k}} \frac{\partial y_{t_k}}{\partial c_{t_l}} \frac{\partial c_{t_l}}{\partial V_{ij}} = \cdots$$
$$= \cdots = (y_t - y_t^*) \otimes c_t$$

- All terms depend only on the current timestep t
- Then, we should sum up all the gradients for all time steps

$$\frac{\partial \mathcal{L}}{\partial V} = \sum_t \frac{\partial \mathcal{L}_t}{\partial V}$$



Backpropagation through time: $\partial\mathcal{L}_t/\partial W$

- Expanding with the chain rule

$$\frac{\partial\mathcal{L}_t}{\partial W} = \frac{\partial\mathcal{L}_t}{\partial y_t}\frac{\partial y_t}{\partial c_t}\frac{\partial c_t}{\partial W}$$

- However, c_t itself depends on $c_{t-1} \rightarrow \frac{\partial c_t}{\partial W}$ depends also on $c_{t-1} \rightarrow$ The current dependency of c_t to W is recurrent
  - And continuing till we reach $c_{-1} = [0]$
- So, in the end we have

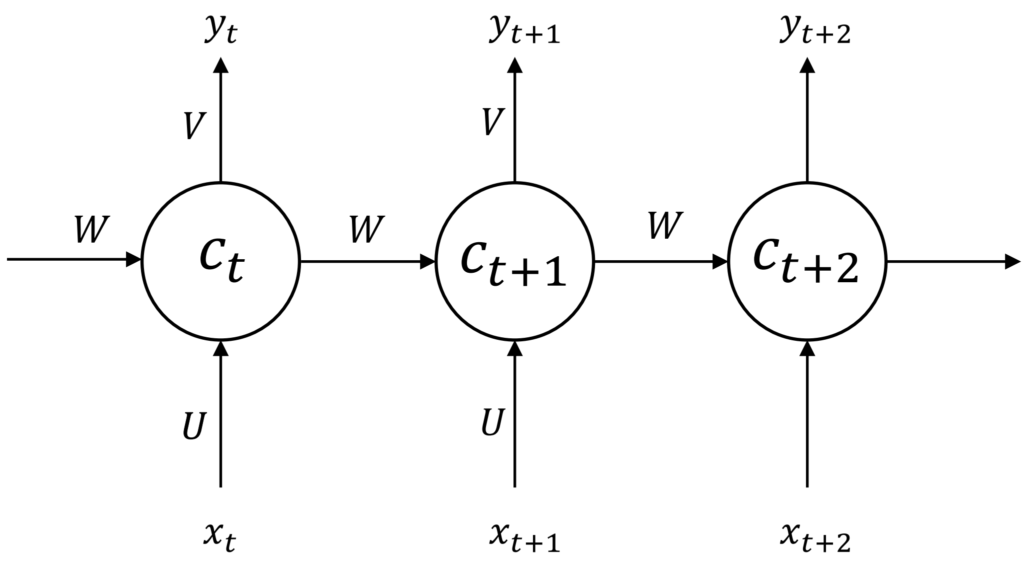
$$\frac{\partial\mathcal{L}_t}{\partial W} = \sum_{k=0}^{t} \frac{\partial\mathcal{L}_t}{\partial y_t}\frac{\partial y_t}{\partial c_t}\frac{\partial c_t}{\partial c_k}\frac{\partial c_k}{\partial W}$$

- The gradient $\frac{\partial c_t}{\partial c_k}$ itself is subject to the chain rule

$$\frac{\partial c_t}{\partial c_k} = \frac{\partial c_t}{\partial c_{t-1}}\frac{\partial c_{t-1}}{\partial c_{t-2}} \dots \frac{\partial c_{k+1}}{\partial c_k} = \prod_{j=k+1}^{t} \frac{\partial c_j}{\partial c_{j-1}}$$

- Then, we should sum up all the gradients for all time steps

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

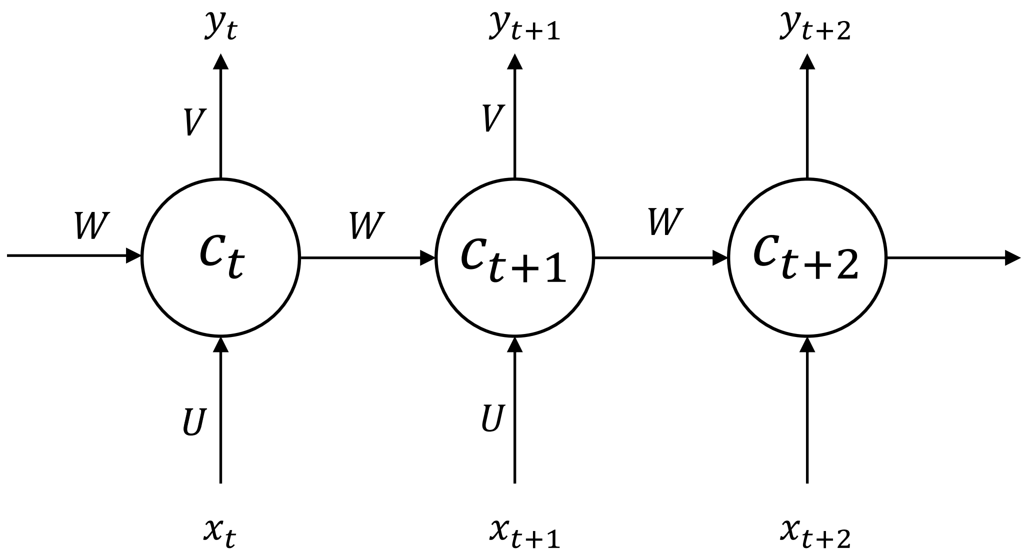


Backpropagation through time: $\partial \mathcal{L}_t / \partial U$

- For parameter matrix U a similar process

$$\frac{\partial \mathcal{L}_t}{\partial U} = \sum_{k=0}^{t} \frac{\partial \mathcal{L}_t}{\partial y_t} \frac{\partial y_t}{\partial c_t} \frac{\partial c_t}{\partial c_k} \frac{\partial c_k}{\partial U}$$

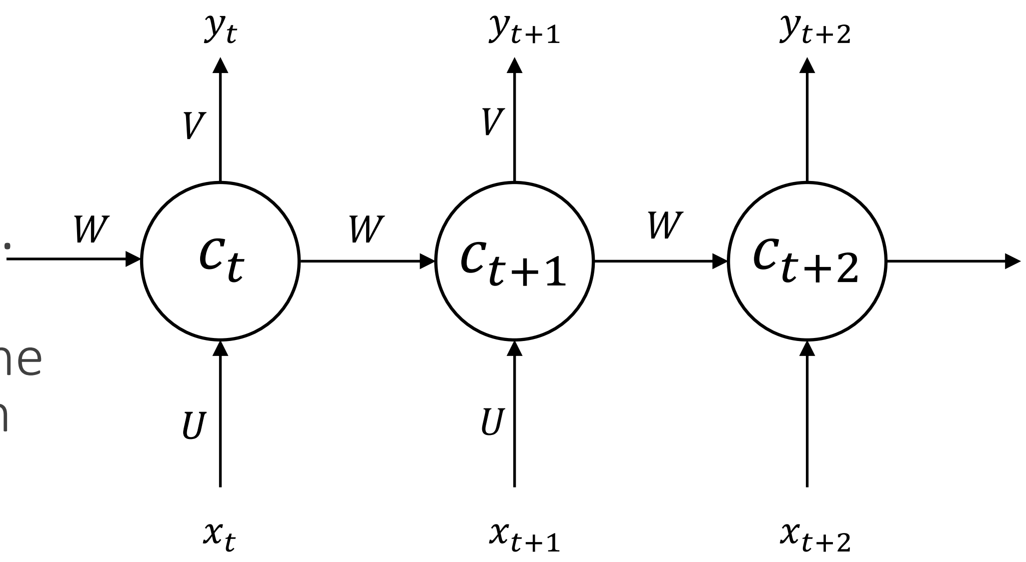
$$c_t = \tanh(U\, x_t + W c_{t-1})$$
$$y_t = \mathrm{softmax}(V\, c_t)$$



Trading off Weight Update Frequency & Gradient Accuracy

- At time t we use current weights w_t to compute states c_t and outputs y_t
- Then, we use the states and outputs to backprop and get w_{t+1}
- Then, at $t+1$ we use w_{t+1} and the current state c_t to y_{t+1} and c_{t+1}
- Then we update the weights again with y_{t+1}.
  - The problem is y_{t+1} was computed with c_t in mind, which in turns depends on the old weights w_t, not the current ones w_{t+1}. So, the new gradients are only an estimate
  - Getting worse and worse, the more we backprop through time

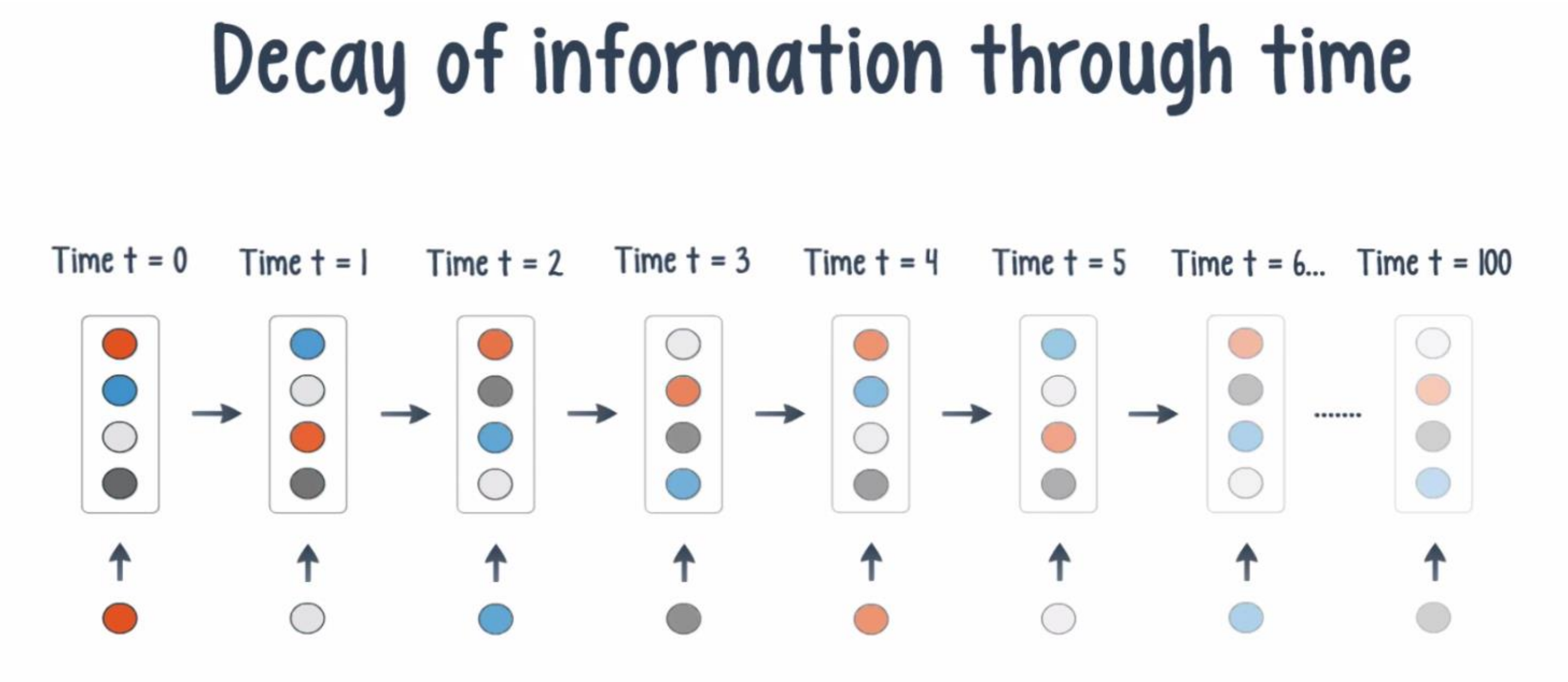
$$c_t = \tanh(U\, x_t + W c_{t-1})$$
$$y_t = \mathrm{softmax}(V\, c_t)$$



Potential solutions

- Do fewer updates
  - That might slow down training
- We can also make sure we do not backprop through more steps than our frequency of updates
  - But then we do not compute the full gradients
  - Bias again → not really gaining much

Vanishing gradients
Exploding gradients
Truncated backprop



An alternative formulation of an RNN

- Easier for mathematical analysis, and doesn't change the mechanics of the recurrent neural network

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

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

$$\theta = \{W, U, b\}$$

What is the problem

- As we just saw, the gradient $\frac{\partial c_t}{\partial c_k}$ itself is subject to the chain rule

$$\frac{\partial c_t}{\partial c_k} = \frac{\partial c_t}{\partial c_{t-1}} \frac{\partial c_{t-1}}{\partial c_{t-2}} \dots \frac{\partial c_{k+1}}{\partial c_k} = \prod_{j=k+1}^{t} \frac{\partial c_j}{\partial c_{j-1}}$$

- Product of ever expanding Jacobians
  - Ever expanding because we multiply more and more for longer dependencies

Let's look again the gradients

- Minimize the total loss over all time steps

$$\arg\min_{\theta} \sum_{t} \mathcal{L}_t(c_{t,\theta})$$

$$\frac{\partial \mathcal{L}_t}{\partial W} = \cdots$$

Let's look again the gradients

- Minimize the total loss over all time steps

$$\arg\min_{\theta} \sum_{t} \mathcal{L}_t(c_{t,\theta})$$

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

$$\frac{\partial \mathcal{L}}{\partial c_t} \frac{\partial c_t}{\partial c_\tau} = \underbrace{\frac{\partial \mathcal{L}}{\partial \mathrm{c_t}} \cdot \frac{\partial c_t}{\partial \mathrm{c_{t-1}}} \cdot \frac{\partial c_{t-1}}{\partial \mathrm{c_{t-2}}}}_{t \ll \tau \,\rightarrow\, \textit{short-term factors}} \cdot \underbrace{\ldots \cdot \frac{\partial c_{\tau+1}}{\partial \mathrm{c_\tau}}}_{t \gg \tau \,\rightarrow\, \textit{long-term factors}}$$

Let's look again the gradients

- Minimize the total loss over all time steps

$$\arg\min_{\theta} \sum_{t} \mathcal{L}_t(c_{t,\theta})$$

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

$$\frac{\partial \mathcal{L}}{\partial c_t} \frac{\partial c_t}{\partial c_\tau} = \frac{\partial \mathcal{L}}{\partial \mathrm{c_t}} \cdot \frac{\partial c_t}{\partial \mathrm{c_{t-1}}} \cdot \frac{\partial c_{t-1}}{\partial \mathrm{c_{t-2}}} \cdot \ldots \cdot \frac{\partial c_{\tau+1}}{\partial \mathrm{c_\tau}}$$

$$\left\|\frac{\partial c_{t+1}}{\partial \mathrm{c_t}}\right\| \leq \|W^T\| \cdot \|diag(\sigma'(c_t))\|$$

Let's look again the gradients

$$\left\|\frac{\partial c_{t+1}}{\partial \mathrm{c_t}}\right\| \leq \|W^T\| \cdot \|diag(\sigma'(c_t))\|$$

- If we assume that the norm of the weight W is bounded
  - Spectral radius (max eigenvalue) is smaller than an arbitrary small number $\lambda_1 < \frac{1}{\gamma}$

- And if we assume that the non linearity is bounded

$$\|diag(\sigma'(c_t))\| < \gamma$$

$$\Rightarrow$$

$$\left\|\frac{\partial c_{t+1}}{\partial \mathrm{c_t}}\right\| < \frac{1}{\gamma}\gamma < 1$$

Let's look again the gradients

- Minimize the total loss over all time steps

$$\arg\min_{\theta} \sum_{t} \mathcal{L}_t(c_{t,\theta})$$

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

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

- RNN gradients expanding product of $\frac{\partial c_t}{\partial \mathrm{c_{t-1}}}$
- With $\eta < 1$ long-term factors $\rightarrow 0$ exponentially fast

[Pascanu, Mikolov, Bengio, On the difficulty of training recurrent neural networks, JMLR 2013]

Some cases

- Let's assume we have 10 time steps and $\frac{\partial c_t}{\partial c_{t-1}} > 1$, e.g. $\frac{\partial c_t}{\partial c_{t-1}} = 1.5$
- What would happen to the total $\frac{\partial \mathcal{L}_t}{\partial W}$?

Some cases

- Let's assume we have 100 time steps and $\frac{\partial c_t}{\partial \mathrm{c_{t-1}}} > 1$, e.g. $\frac{\partial c_t}{\partial \mathrm{c_{t-1}}} = 1.5$
- What would happen to the total $\frac{\partial \mathcal{L}_t}{\partial W}$?

$$\frac{\partial \mathcal{L}}{\partial c_t}\frac{\partial c_t}{\partial c_\tau} \propto 1.5^{10} = 4.06 \cdot 10^{17}$$

Some cases

- Let's assume now that $\frac{\partial c_t}{\partial c_{t-1}} < 1$, e.g. $\frac{\partial c_t}{\partial c_{t-1}} = 0.5$
- What would happen to the total $\frac{\partial \mathcal{L}_t}{\partial W}$?

Some cases

- Let's assume now that $\frac{\partial c_t}{\partial c_{t-1}} < 1$, e.g. $\frac{\partial c_t}{\partial c_{t-1}} = 0.5$
- What would happen to the total $\frac{\partial \mathcal{L}_t}{\partial W}$?

$$\frac{\partial \mathcal{L}}{\partial c_t} \frac{\partial c_t}{\partial c_\tau} \propto 0.5^{10} = 9.7 \cdot 10^{-5}$$

- Do you think our optimizers like these kind of gradients?

Some cases

- Let's assume now that $\frac{\partial c_t}{\partial \mathrm{c_{t-1}}} < 1$, e.g. $\frac{\partial c_t}{\partial \mathrm{c_{t-1}}} = 0.5$
- What would happen to the total $\frac{\partial \mathcal{L}_t}{\partial W}$?

$$\frac{\partial \mathcal{L}}{\partial c_t} \frac{\partial c_t}{\partial c_\tau} \propto 0.5^{10} = 9.7 \cdot 10^{-5}$$

- Do you think our optimizers like these kind of gradients?
- Too large → unstable training, oscillations, divergence
- Too small → very slow training, has it converged?

A visual example

- Recurrent networks as iterated functions

[Credit: R. Grosse]

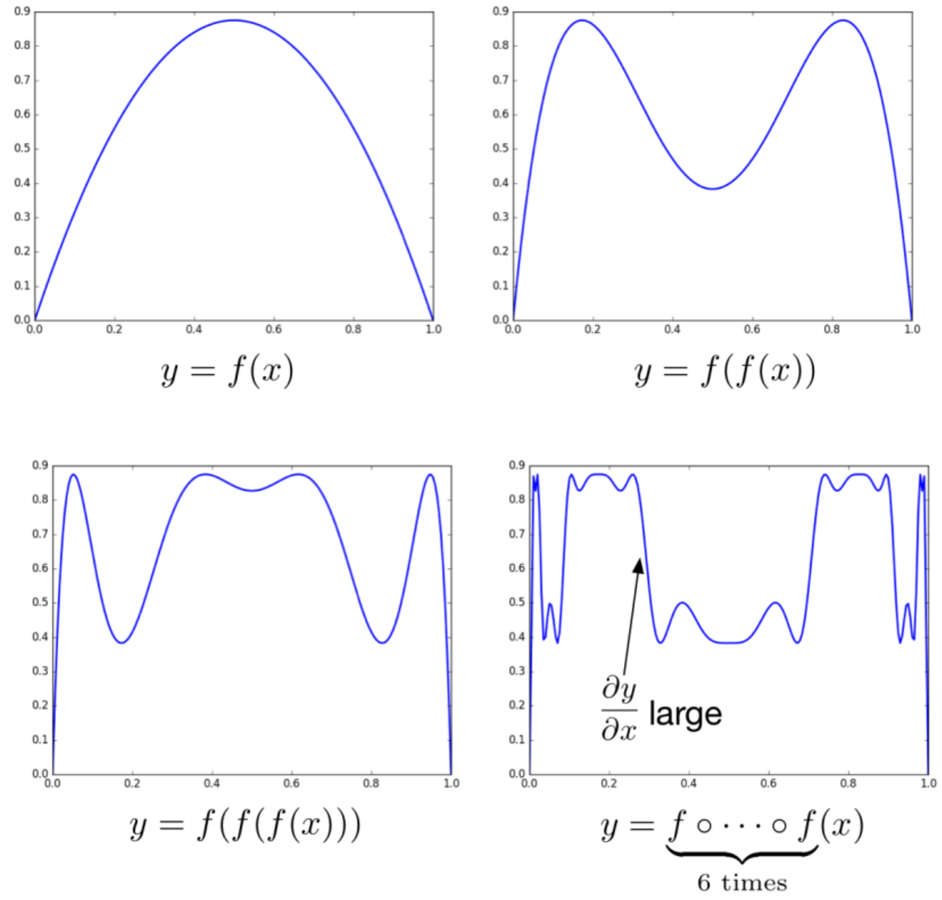


Figure 2: Iterations of the function $f(x) = 3.5\,x\,(1-x)$.

Vanishing & Exploding Gradients

- In recurrent networks, and in very deep networks in general (an RNN is not very different from an MLP), gradients are much affected by depth

$$\frac{\partial \mathcal{L}}{\partial c_t} = \frac{\partial \mathcal{L}}{\partial \mathrm{c_T}} \cdot \frac{\partial c_T}{\partial \mathrm{c_{T-1}}} \cdot \frac{\partial c_{T-1}}{\partial \mathrm{c_{T-2}}} \cdot \ldots \cdot \frac{\partial c_{t+1}}{\partial \mathrm{c}_{c_t}} \text{ and } \frac{\partial c_{t+1}}{\partial \mathrm{c}_t} < 1 \Rightarrow \frac{\partial \mathcal{L}}{\partial W} \ll 1 \Rightarrow \text{Vanishing gradient}$$

$$\frac{\partial \mathcal{L}}{\partial c_t} = \frac{\partial \mathcal{L}}{\partial \mathrm{c_T}} \cdot \frac{\partial c_T}{\partial \mathrm{c_{T-1}}} \cdot \frac{\partial c_{T-1}}{\partial \mathrm{c_{T-2}}} \cdot \ldots \cdot \frac{\partial c_{t+1}}{\partial \mathrm{c}_{c_t}} \text{ and } \frac{\partial c_{t+1}}{\partial \mathrm{c}_t} > 1 \Rightarrow \frac{\partial \mathcal{L}}{\partial W} \gg 1 \Rightarrow \text{Exploding gradient}$$

Vanishing gradients & long memory

- Vanishing gradients are particularly a problem for long sequences
- Why?

Vanishing gradients & long memory

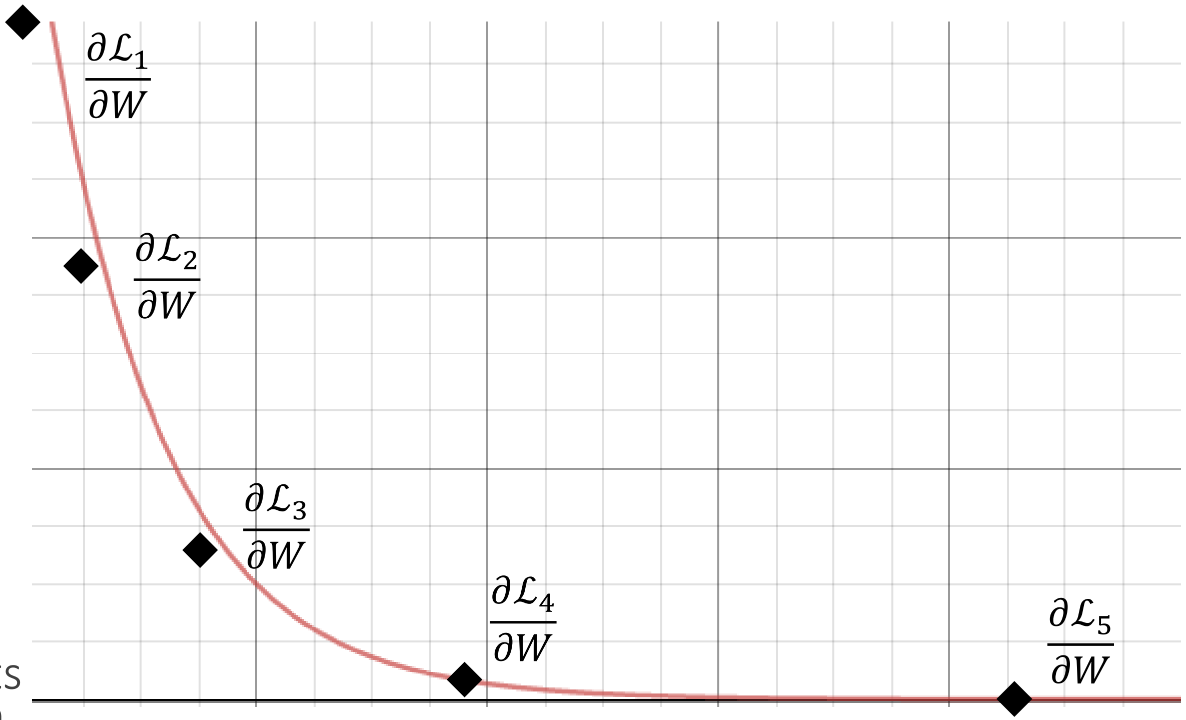
- Vanishing gradients are particularly a problem for long sequences
- Why?
- Exponential decay

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

- The further back we look (long-term dependencies), the smaller the weights automatically become
  - exponentially smaller weights

Why are vanishing gradients bad?

- Weight updates focus on early time steps
- Updates for longer time steps become exponentially smaller
- Bad learning, even if we train the model exponentially longer. Why?
- Weights quickly learn (prefer) to “model” short-term transitions
  - And ignore long-term transitions
- At best, even after longer training, they will try “fine-tune” the whatever bad “modelling” of long-term transitions
  - After the short-term transitions are learned, the weights are set for them and are likely suboptimal for long-term
- Eventually, as the short-term transitions are inherently more prevalent, they will dominate the learning and gradients



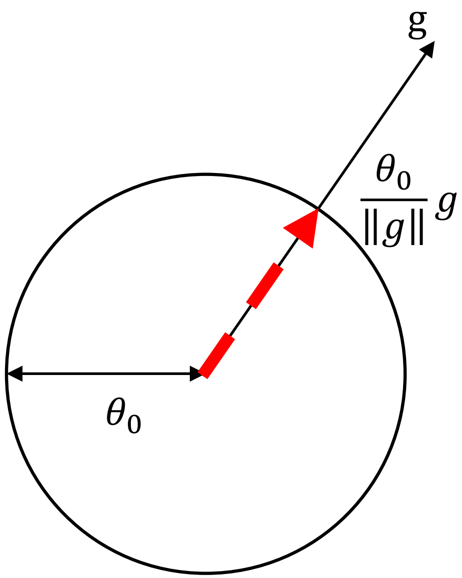
$$\frac{\partial \mathcal{L}}{\partial W} = \frac{\partial \mathcal{L}_1}{\partial W} + \frac{\partial \mathcal{L}_2}{\partial W} + \frac{\partial \mathcal{L}_3}{\partial W} + \frac{\partial \mathcal{L}_4}{\partial W} + \frac{\partial \mathcal{L}_5}{\partial W}$$

Quick fix for exploding gradients: Rescaling!

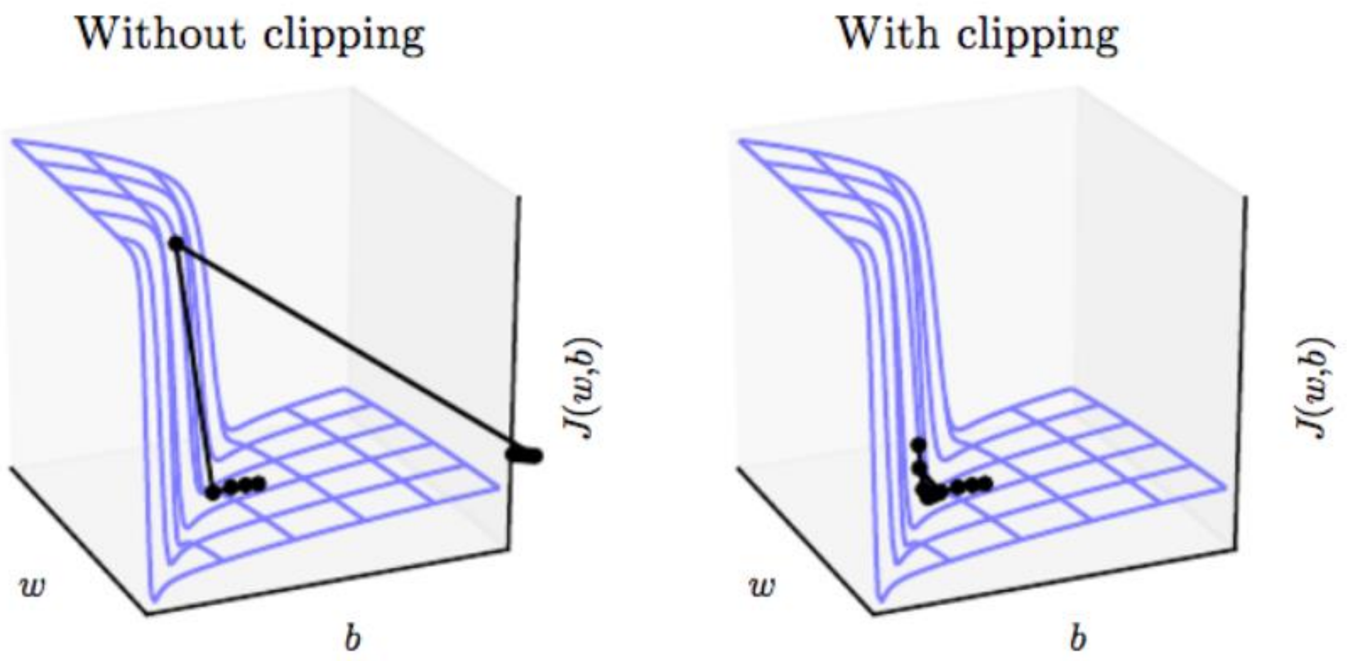
- First, get the gradient $\mathrm{g} \leftarrow \frac{\partial \mathcal{L}}{\partial W}$
- Check if the norm is larger than a threshold θ_0
- If it is, rescale it to have same direction and threshold norm

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

- Simple, but works!



An illustration



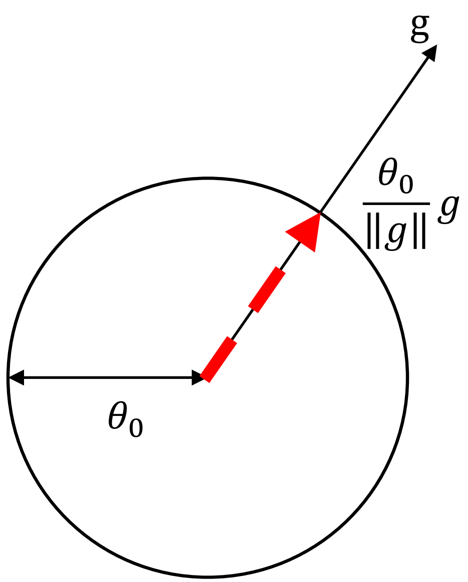
— Goodfellow et al., *Deep Learning*

Can we rescale gradients also for vanishing gradients? No!

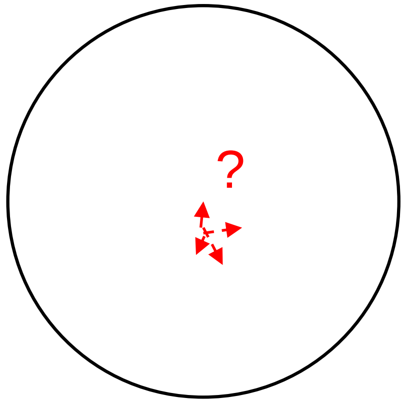
- The nature of the problem is different
- Exploding gradients → you might have bouncing and unstable optimization
- Vanishing gradients → you simply do not have a gradient to begin with
  - Rescaling of what exactly?
- Unclear how would you rescale in a principled way, without affecting the rest of the time-steps
- In any case, even with re-scaling we would still focus on the short-term gradients
  - Long-term dependencies would still be ignored

Building intuition

- With exploding gradients, the gradient is sort of good just too large
  - That is, the direction of the gradient is good, but the magnitude is too much
  - Problem with optimization → bouncing, oscillation, etc.



- With vanishing gradients, the gradient is not good in the first place
  - Neither the direction because of numerical instabilities, nor the magnitude are good
  - Even if we rescale, are we sure we are going to change weights in the right direction? We cannot be sure.



Biased gradients?

- Backpropagating all the way till infinity is unrealistic
  - We would backprop forever (or simply it would be computationally very expensive)
  - And in case, the gradients would be inaccurate because of intermediate updates
- What about truncating backprop to the last K steps

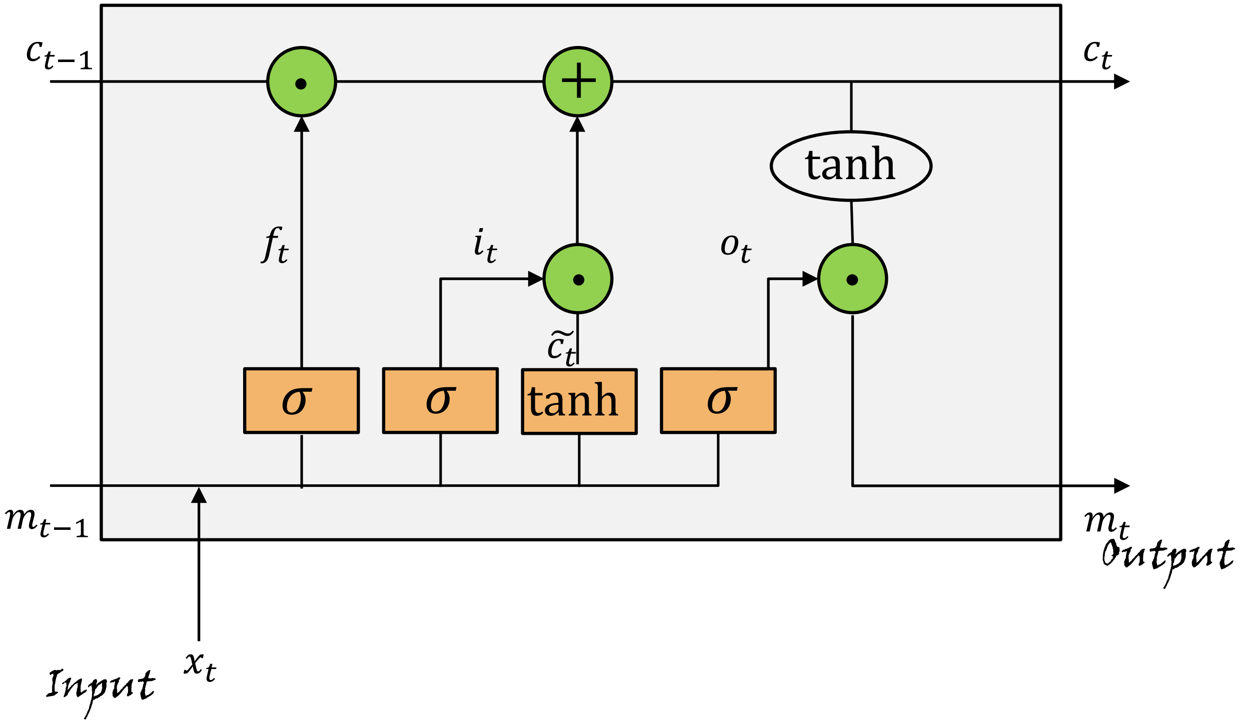
$$\tilde{g}_{t+1} \propto \frac{\partial \mathcal{L}}{\partial w}\Big|_{t=0}^{t=k}$$

- Unfortunately, this leads to biased gradients

$$g_{t+1} = \frac{\partial \mathcal{L}}{\partial w}\Big|_{t=0}^{t=\infty} \neq \tilde{g}_{t+1}$$

- Other algorithms exist but they are not as successful
  - Maybe we will visit them later

LSTM and variants



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 make the product roughly the same no matter the length?

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 make the product roughly the same no matter the length?
- Use the identity function with gradient of 1

Main idea of LSTMs

- Over time the state change is $c_{t+1} = c_t + \Delta \mathrm{c}_{\mathrm{t+1}}$
- This constant over-writing over long time steps leads to chaotic behavior

- Input weight conflict
  - Are all inputs important enough to write them down?

- Output conflict
  - Are all outputs important enough to be read?

- Forget conflict
  - Is all information important enough to be remembered over time?

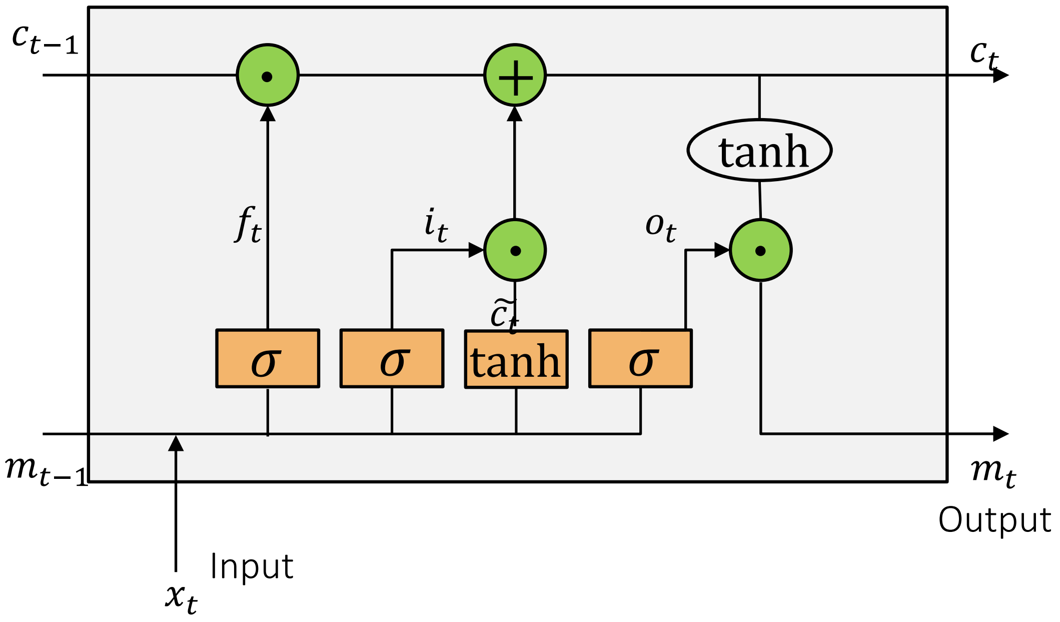
LSTMs

- RNNs

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

- LSTMs

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



LSTMs: A marking difference

- RNNs

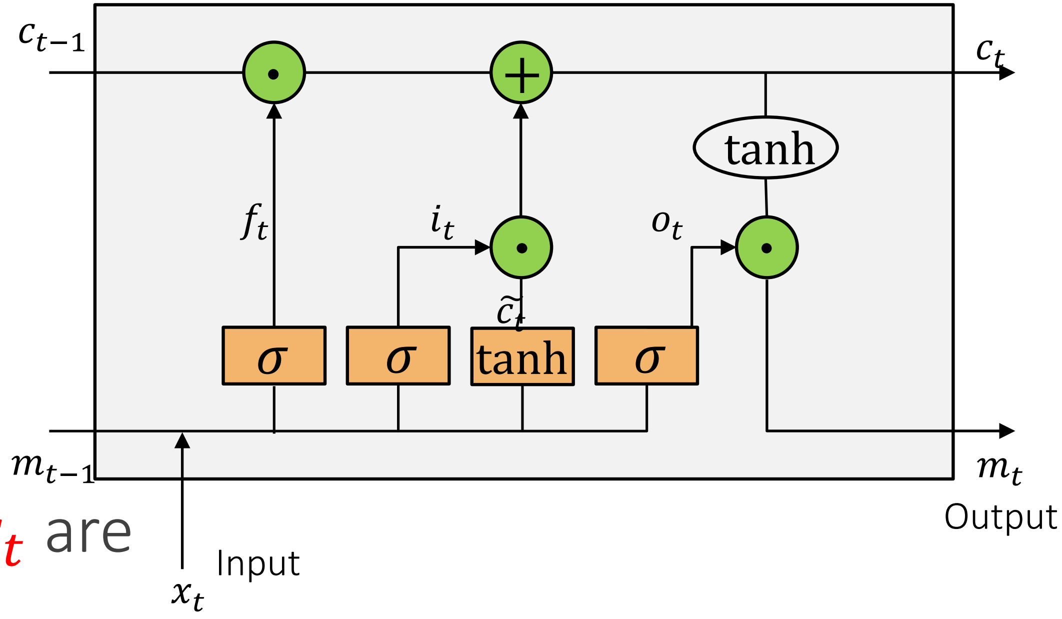
$$c_t = W \cdot \tanh(c_{t-1}) + U \cdot x_t + b$$

- LSTMs

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

- The previous state c_{t-1} and the next state c_t are also connected by addition
  - It is also connected by the $\tanh$, but at least there is the addition to make sure of good gradients

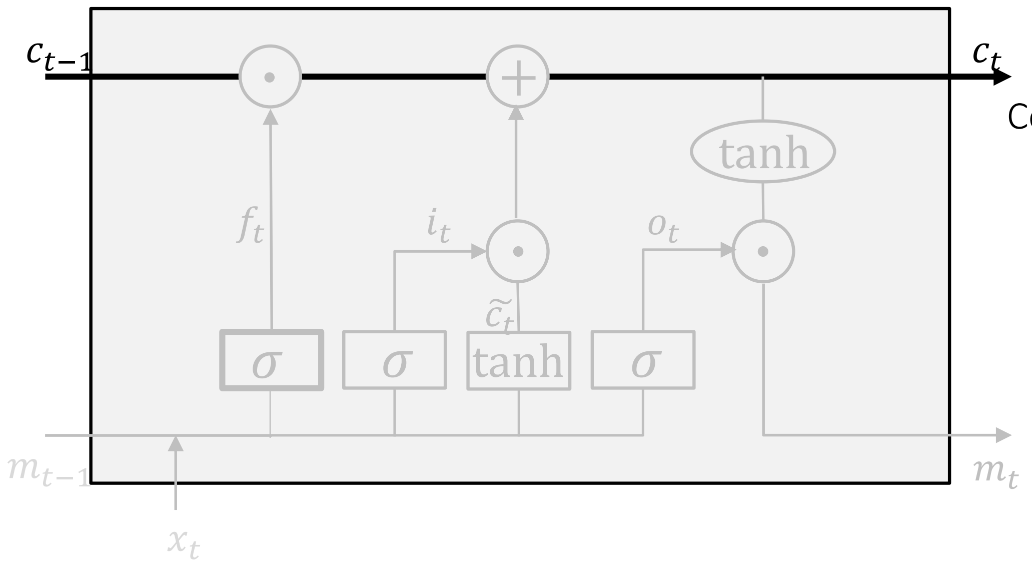
Additivity leads to strong gradients
Bounded by sigmoidal f



Nice tutorial: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

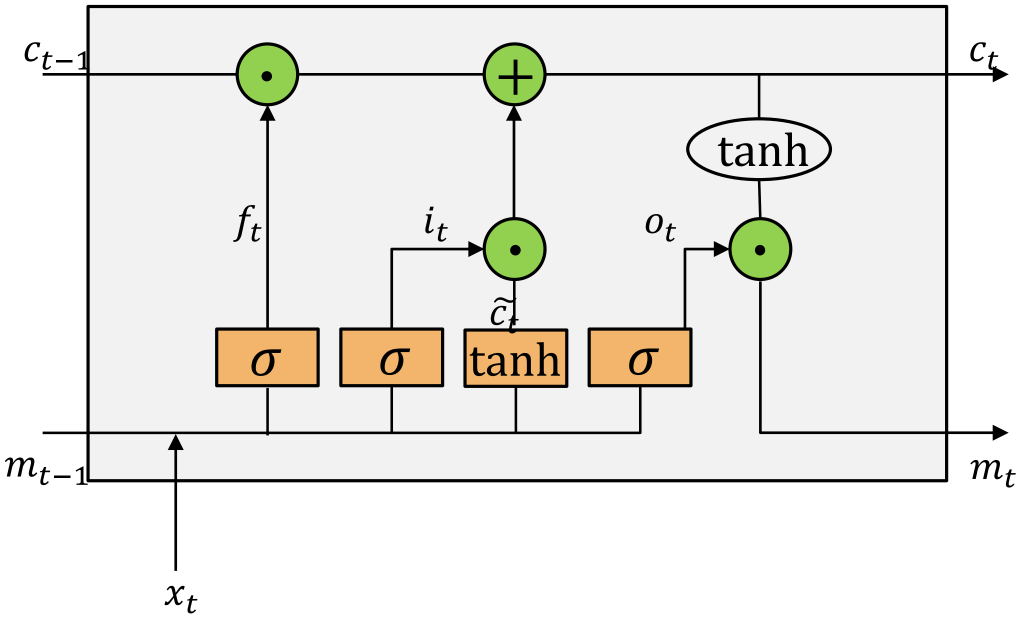
Cell state

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



LSTM nonlinearities

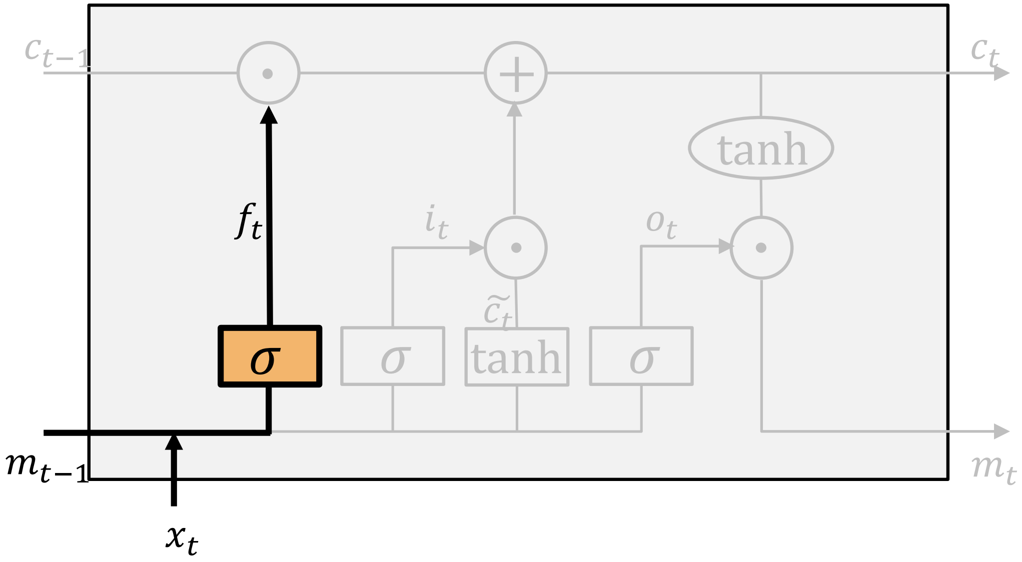
$$i = \sigma\left(x_t U^{(i)} + m_{t-1} W^{(i)}\right)$$
$$f = \sigma\left(x_t U^{(f)} + m_{t-1} W^{(f)}\right)$$
$$o = \sigma\left(x_t U^{(o)} + m_{t-1} W^{(o)}\right)$$
$$\widetilde{c}_t = \tanh(x_t U^{(g)} + m_{t-1} W^{(g)})$$
$$c_t = c_{t-1} \odot f + \widetilde{c}_t \odot i$$
$$m_t = \tanh(c_t)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



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

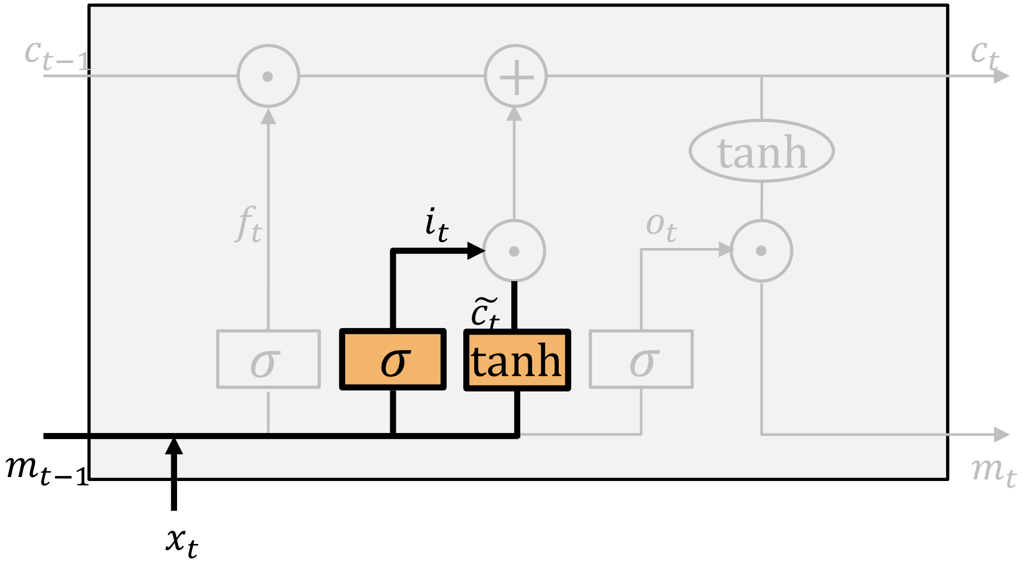
LSTM Step by Step #1

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



LSTM Step by Step #2

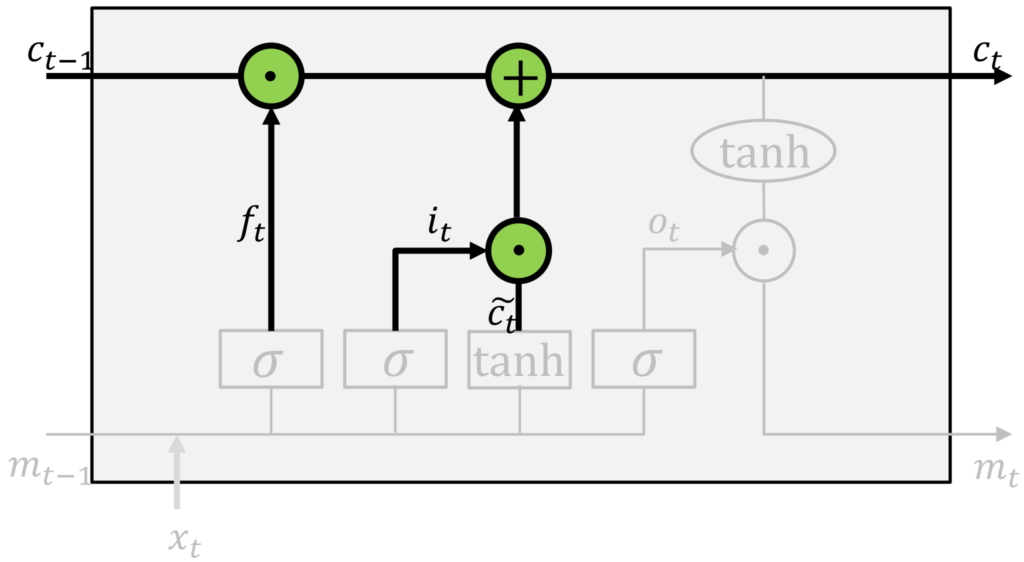
$$i = \sigma\left(x_t U^{(i)} + m_{t-1} W^{(i)}\right)$$
$$f = \sigma\left(x_t U^{(f)} + m_{t-1} W^{(f)}\right)$$
$$o = \sigma\left(x_t U^{(o)} + m_{t-1} W^{(o)}\right)$$
$$\widetilde{c}_t = \tanh(x_t U^{(g)} + m_{t-1} W^{(g)})$$
$$c_t = c_{t-1} \odot f + \widetilde{c}_t \odot i$$
$$m_t = \tanh(c_t)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



- Decide what new information is relevant from the new input and should be added to the new memory
  - Modulate the input i_t
  - Generate candidate memories $\widetilde{c}_t$

LSTM Step by Step #3

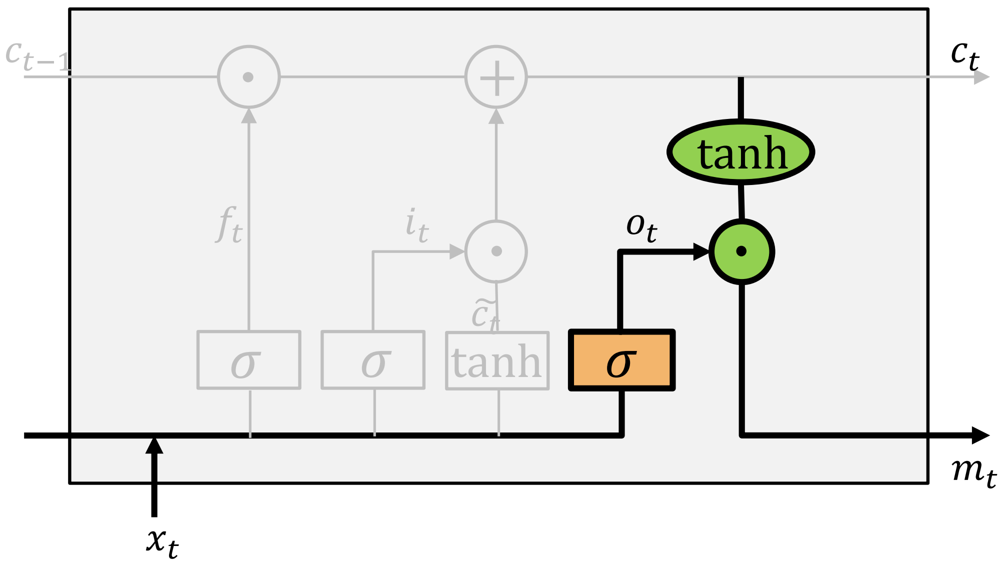
$$i = \sigma\left(x_t U^{(i)} + m_{t-1} W^{(i)}\right)$$
$$f = \sigma\left(x_t U^{(f)} + m_{t-1} W^{(f)}\right)$$
$$o = \sigma\left(x_t U^{(o)} + m_{t-1} W^{(o)}\right)$$
$$\widetilde{c}_t = \tanh(x_t U^{(g)} + m_{t-1} W^{(g)})$$
$$c_t = c_{t-1} \odot f + \widetilde{c}_t \odot i$$
$$m_t = \tanh(c_t)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



- 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

LSTM Step by Step #4

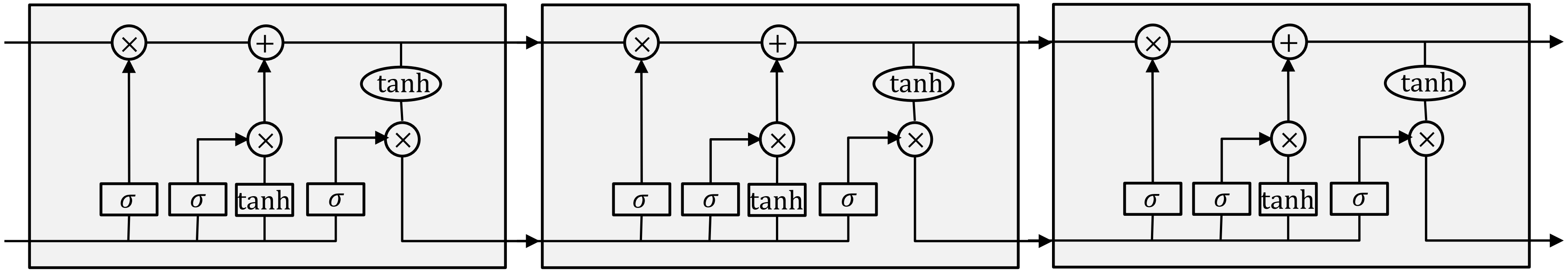
$$i = \sigma\left(x_t U^{(i)} + m_{t-1} W^{(i)}\right)$$
$$f = \sigma\left(x_t U^{(f)} + m_{t-1} W^{(f)}\right)$$
$$o = \sigma\left(x_t U^{(o)} + m_{t-1} W^{(o)}\right)$$
$$\widetilde{c}_t = \tanh(x_t U^{(g)} + m_{t-1} W^{(g)})$$
$$c_t = c_{t-1} \odot f + \widetilde{c}_t \odot i$$
$$m_t = \tanh(c_t)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



- Modulate the output
  - Does the new cell state relevant? → Sigmoid 1
  - If not → Sigmoid 0
- Generate the new memory

Unrolling the LSTMs

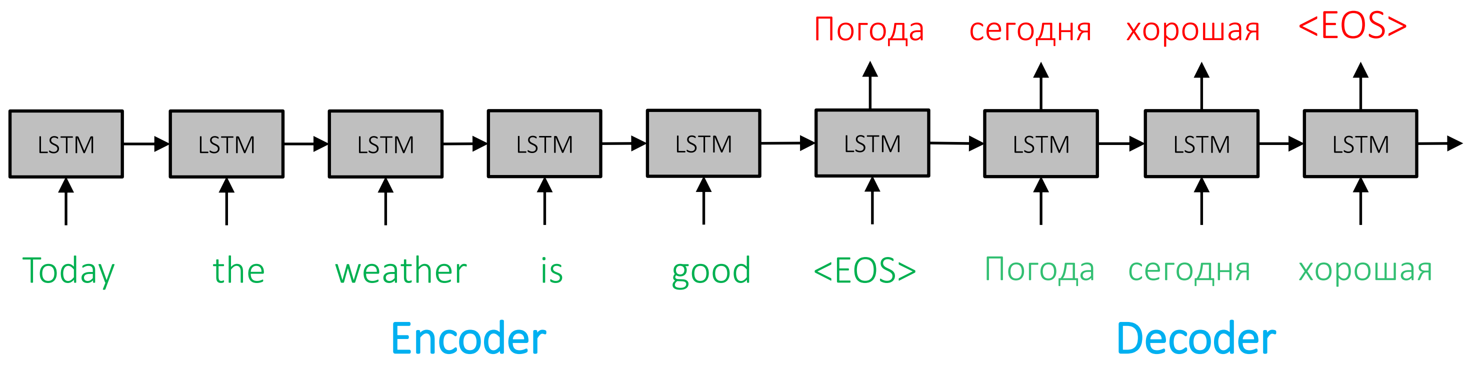
- Just the same like for RNNs
- The engine is a bit different (more complicated)
  - Because of their gates LSTMs capture long and short term dependencies



LSTM variants

- LSTM with peephole connections
- Gates have access also to the previous cell states c_{t-1} (not only memories)
- Bi-directional recurrent networks
- Gated Recurrent Units (GRU)
- Phased LSTMs
- Skip LSTMs
- And many more …

Encoder-Decoder Architectures



Machine translation

- The phrase in the source language is one sequence
  - "Today the weather is good"
- It is captured by an Encoder LSTM
- The phrase in the target language is also a sequence
  - "Погода сегодня хорошая"
- It is captured by a Decoder LSTM

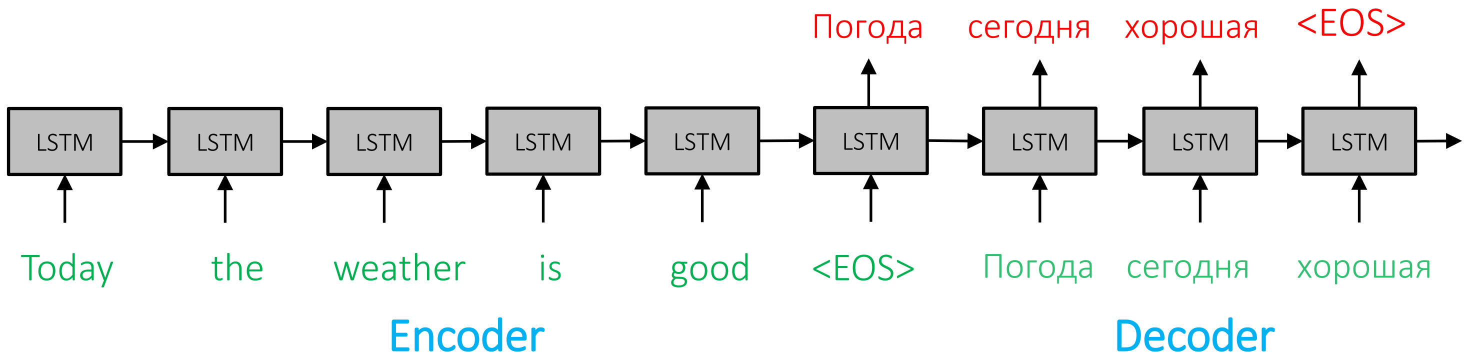


Image captioning

- Similar to image translation
- The only difference is that the Encoder LSTM is an image ConvNet
  - VGG, ResNet, …
- Keep decoder the same

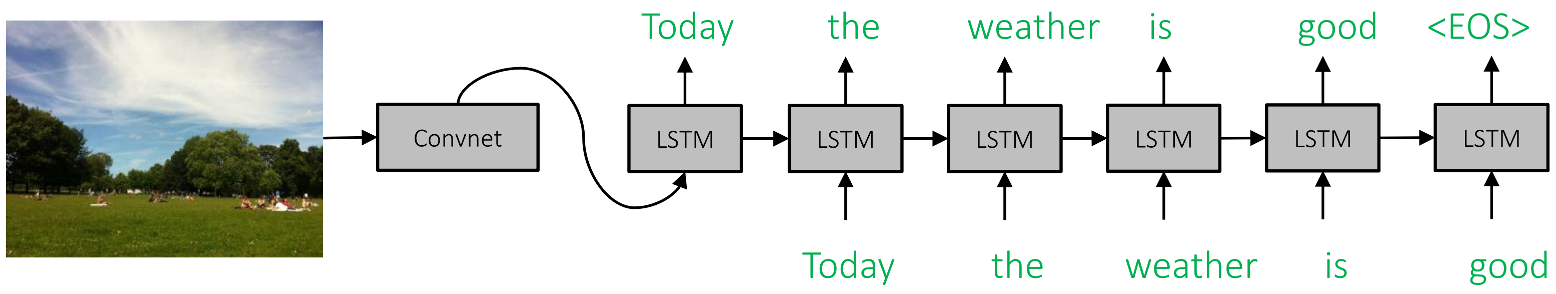
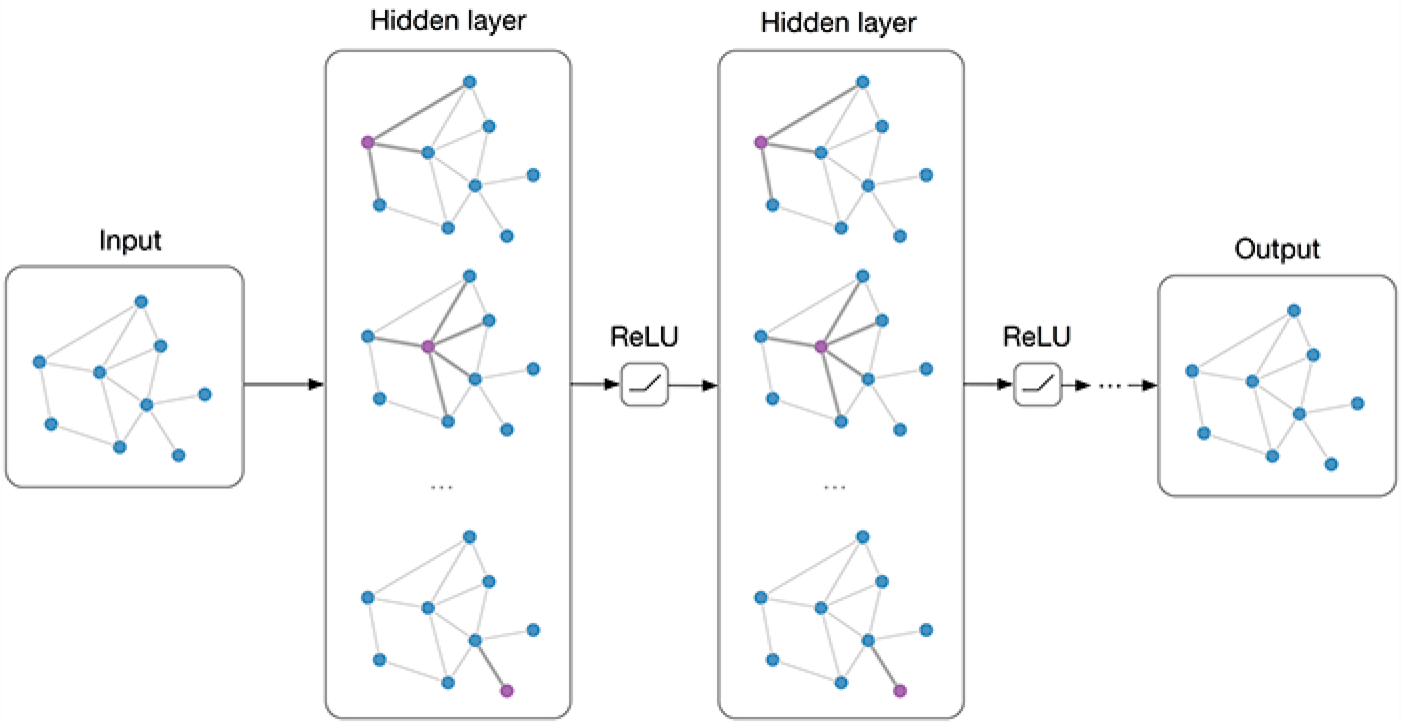


Image captioning demo

[Click to go to the video in Youtube]

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

Graph Neural Networks



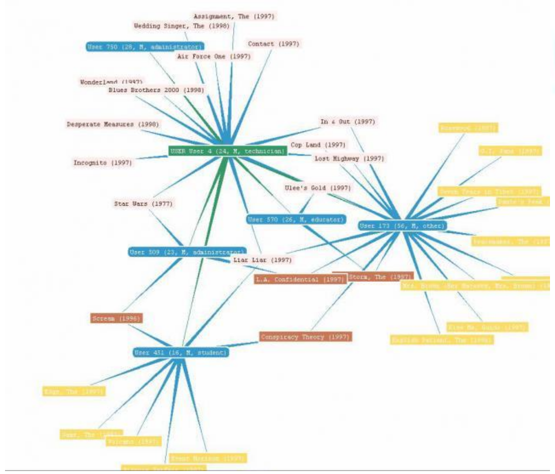
Why Graphs?

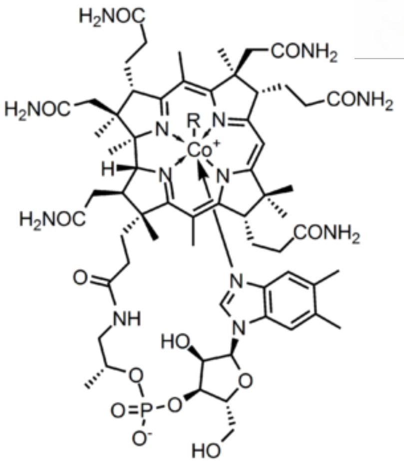
- Many domains & data have graph structure
- Examples?

Why Graphs?

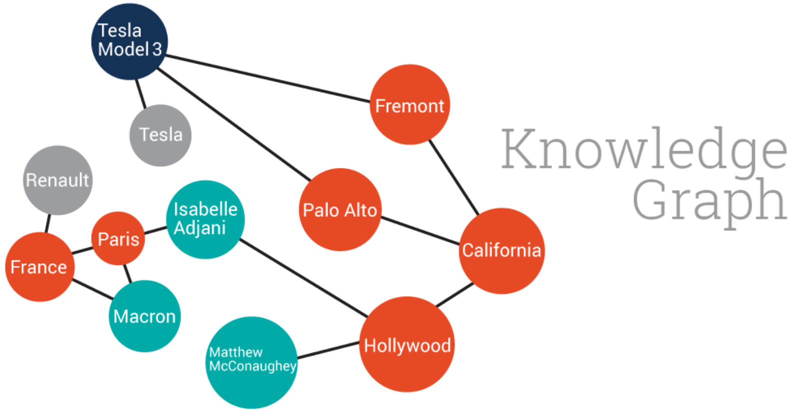
- Many domains & data have graph structure

- Social networks
- Knowledge graphs
- Recommender systems
- Chemical compounds
- And more





R = 5'-deoxyadenosyl, Me, OH, CN



Predictions tasks on graphs?

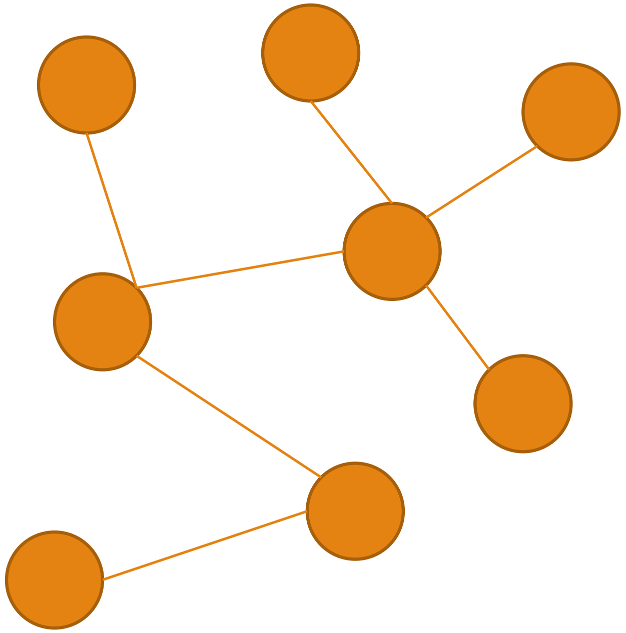
Predictions tasks on graphs?

- Node classification
- Filling out missing edges
- Filling out missing nodes
- Novel graph generation

DeepWalk

Algorithm

1. Perform random walks on the graph to generate node sequences

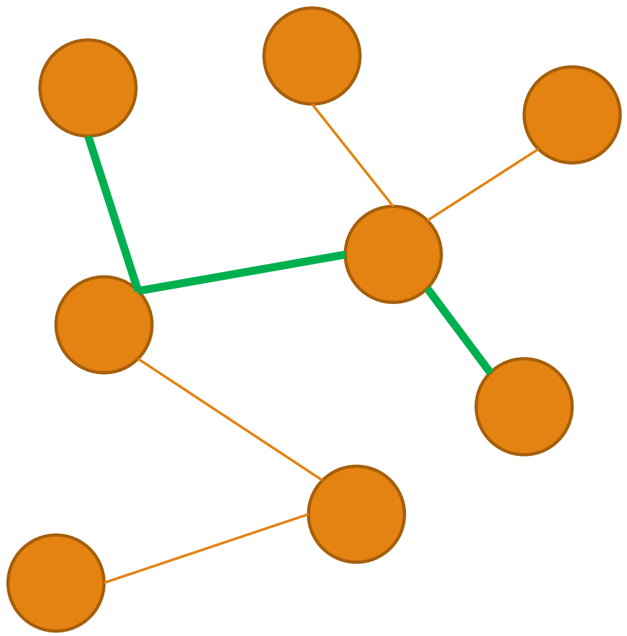


[DeepWalk: Online Learning of Social Representations, Perozzi et al, 2014]()

DeepWalk

Algorithm

1. Perform random walks on the graph to generate node sequences
2. Run skip-gram to learn the node embedding

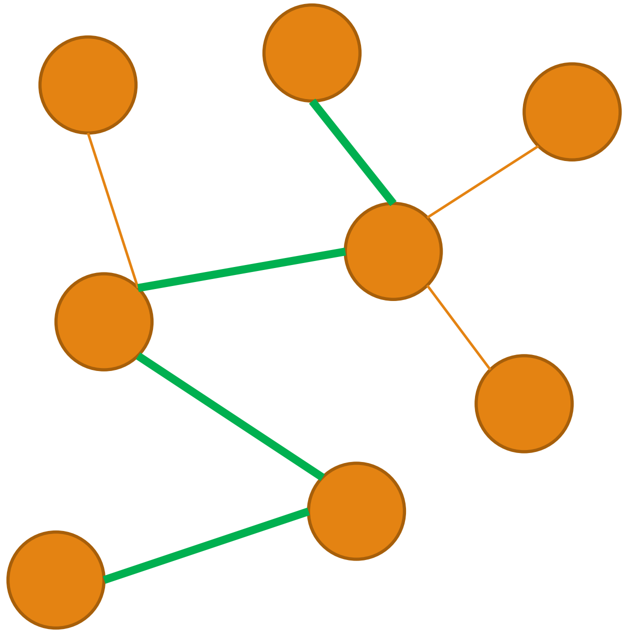


[DeepWalk: Online Learning of Social Representations, Perozzi et al, 2014]

DeepWalk

Algorithm

1. Perform random walks on the graph to generate node sequences

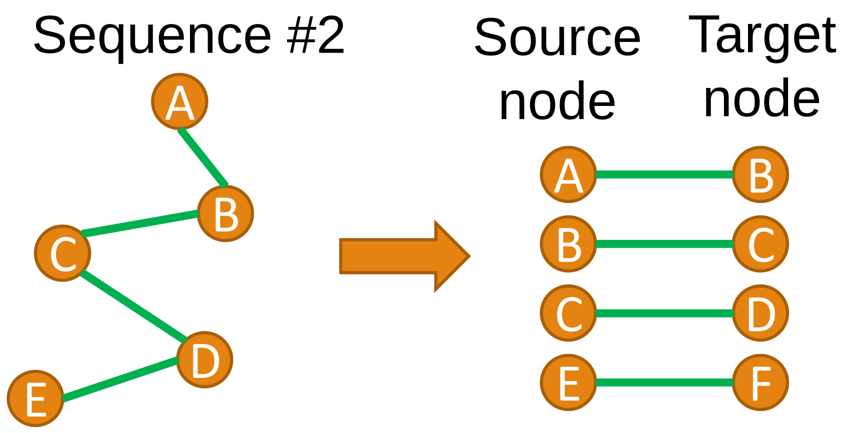


[DeepWalk: Online Learning of Social Representations, Perozzi et al, 2014]

DeepWalk

Algorithm

1. Perform random walks on the graph to generate node sequences
2. Run skip-gram to learn node embeddings

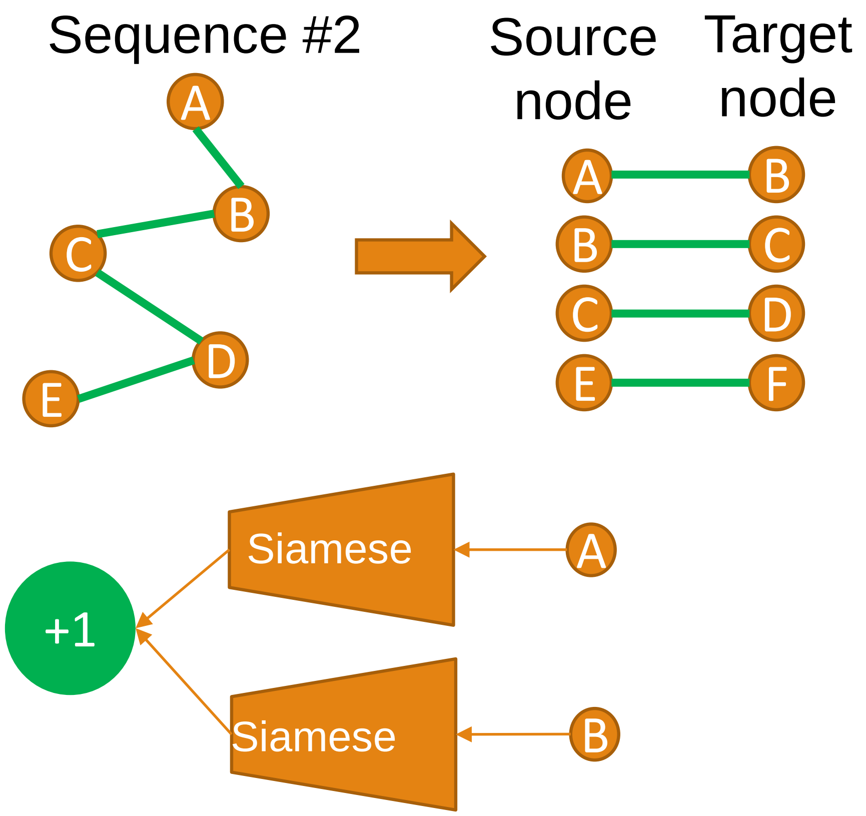


[DeepWalk: Online Learning of Social Representations, Perozzi et al, 2014]

DeepWalk

Algorithm

1. Perform random walks on the graph to generate node sequences
2. Run skip-gram to learn node embeddings

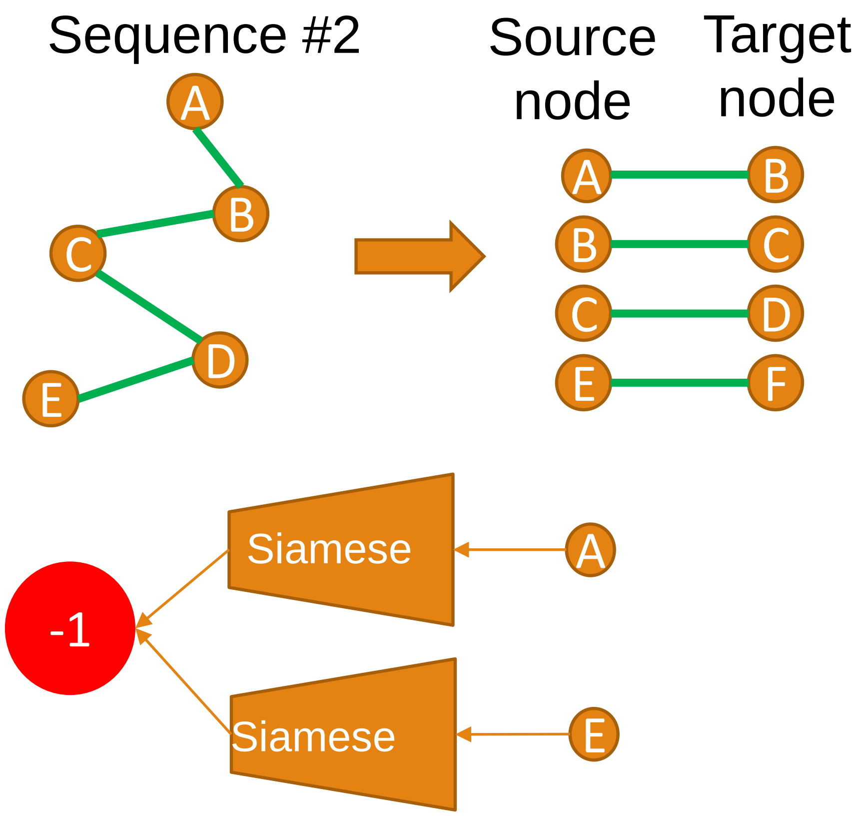


[DeepWalk: Online Learning of Social Representations, Perozzi et al, 2014]

DeepWalk

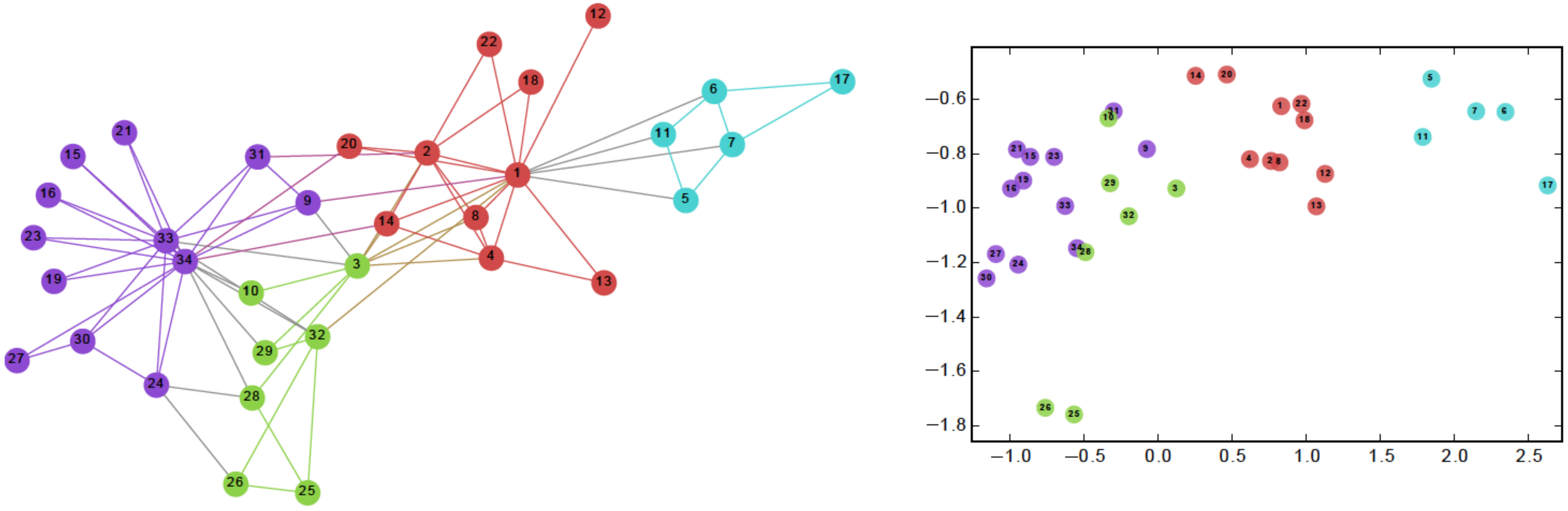
Algorithm

1. Perform random walks on the graph to generate node sequences
2. Run skip-gram to learn node embeddings



[DeepWalk: Online Learning of Social Representations, Perozzi et al, 2014]()

DeepWalk: Results



(a) Input: Karate Graph (b) Output: Representation

[DeepWalk: Online Learning of Social Representations, Perozzi et al, 2014]

DeepWalk: A problem

- The method is transductive
- Whenever a new node is added to the graph, the model must be retrained
- This is not useful for dynamic graphs

[DeepWalk: Online Learning of Social Representations, Perozzi et al, 2014]

GraphSage

Algorithm 1: GraphSAGE embedding generation (i.e., forward propagation) algorithm

Input : Graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$; input features $\{\mathbf{x}_v, \forall v \in \mathcal{V}\}$; depth K; weight matrices $\mathbf{W}^k, \forall k \in \{1, ..., K\}$; non-linearity σ; differentiable aggregator functions $\text{AGGREGATE}_k, \forall k \in \{1, ..., K\}$; neighborhood function $\mathcal{N} : v \rightarrow 2^{\mathcal{V}}$

Output : Vector representations $\mathbf{z}_v$ for all $v \in \mathcal{V}$

$\mathbf{h}_v^0 \leftarrow \mathbf{x}_v, \forall v \in \mathcal{V}$;
for $k = 1...K$ **do**
  for $v \in \mathcal{V}$ **do**
    $\mathbf{h}_{\mathcal{N}(v)}^k \leftarrow \text{AGGREGATE}_k(\{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\})$;
    $\mathbf{h}_v^k \leftarrow \sigma\left(\mathbf{W}^k \cdot \text{CONCAT}(\mathbf{h}_v^{k-1}, \mathbf{h}_{\mathcal{N}(v)}^k)\right)$
  end
  $\mathbf{h}_v^k \leftarrow \mathbf{h}_v^k / \|\mathbf{h}_v^k\|_2, \forall v \in \mathcal{V}$
end
$\mathbf{z}_v \leftarrow \mathbf{h}_v^K, \forall v \in \mathcal{V}$

[GraphSage: Inductive Representation Learning on Large Graphs, Hamilton et al., 2017]

GraphSage: How to aggregate?

- Mean aggregation $\mathbf{h}_v^k \leftarrow \sigma(\mathbf{W} \cdot \text{MEAN}(\{\mathbf{h}_v^{k-1}\} \cup \{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\})$
- LSTM aggregation
- Pooling aggregation $\text{AGGREGATE}_k^{\text{pool}} = \max(\{\sigma\left(\mathbf{W}_{\text{pool}}\mathbf{h}_{u_i}^k + \mathbf{b}\right), \forall u_i \in \mathcal{N}(v)\})$
- Loss $J_{\mathcal{G}}(\mathbf{z}_u) = -\log\left(\sigma(\mathbf{z}_u^\top \mathbf{z}_v)\right) - Q \cdot \mathbb{E}_{v_n \sim P_n(v)} \log\left(\sigma(-\mathbf{z}_u^\top \mathbf{z}_{v_n})\right)$

Graph Convolutional Networks

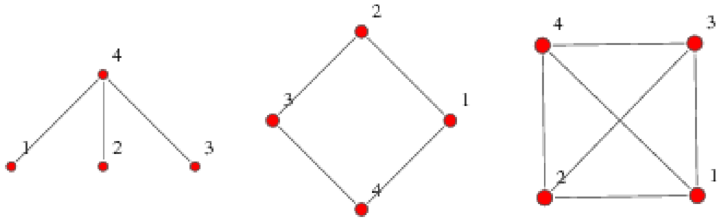
- Assuming a graph $G = (\mathcal{V}, \mathcal{E})$

- A node has a description x_i, all stored in a $N \times D$ matrix $X = [\dots, x_i, \dots]$

- The graph structure is encoded by the adjacency matrix A

- A neural network on this graph then is

$$H^{(l+1)} = h(H^{(l)}, A)$$



$$\begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

Graph Convolutional Networks, Kipf and Welling, 2016

Graph Convolutional Networks: A simple example

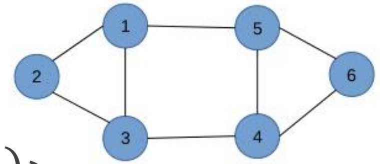
- $h\left(H^{(l)}, A\right) = \sigma(AH^{(l)}W^{(l)})$
- Two problems
  - Given a node, the adjacency matrix A considers neighboring nodes but not the node itself → Aggregation does not use the node itself
  - A node might have different numbers of neighbors and change the scale of the multiplication
- Add the identity matrix to A
- Left multiply by $D^{-1}A$: D is the degree matrix
- Combining all, we have the following module

$$h\left(H^{(l)}, A\right) = \sigma(D^{-\frac{1}{2}}\,\hat{A}\,D^{-\frac{1}{2}}\;H^{(l)}W^{(l)})$$
$$\hat{A} = A + I$$

Graph Convolutional Networks, Kipf and Welling, 2016

Degree matrix

$$D_{ij} = \begin{cases} d(i) & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$



$$D = \begin{bmatrix} 3 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 \end{bmatrix}$$

19

Summary

- Sequential data
- Recurrent Neural Networks
- Backpropagation through time
- Exploding and vanishing gradients
- LSTMs and variants
- Encoder-Decoder Architectures
- Graph Neural Networks