Recurrent Neural Networks

Seminar Principles of Data Mining and Learning Algorithms

Gular Shukurova
Sheikh Mastura Farzana

Outline

Unfolding Computational Graphs

- Expressing a recurrent computation into a computational graph

$$\boldsymbol{s}^{(t)} = f(\boldsymbol{s}^{(t-1)}; \boldsymbol{\theta})$$

For t=3

$$\begin{aligned} \boldsymbol{s}^{(3)} &= f(\boldsymbol{s}^{(2)}; \boldsymbol{\theta}) \\ &= f(f(\boldsymbol{s}^{(1)}; \boldsymbol{\theta}); \boldsymbol{\theta}). \end{aligned}$$

Example

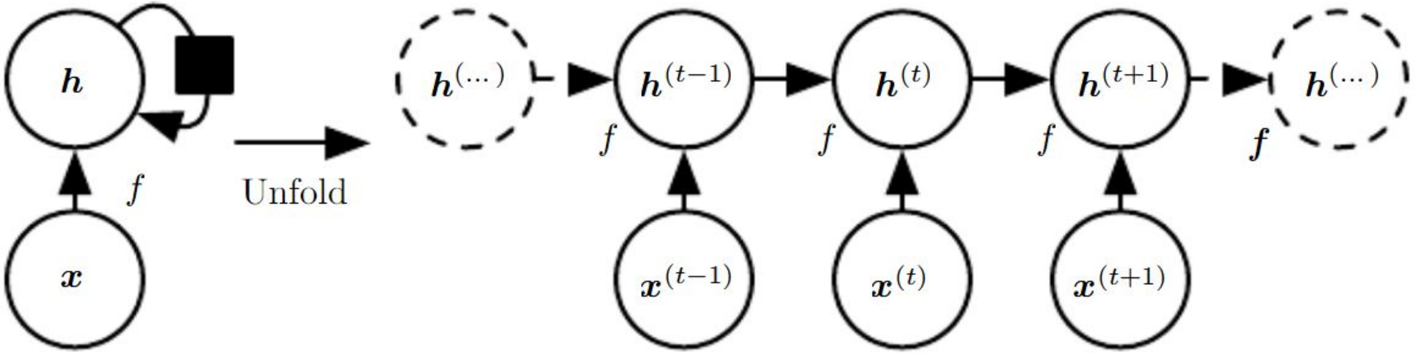


Figure: (left) Circuit Diagram, (right) unfolded computational graph, each node associated to a single timestep.

Reference: Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep Learning. http://www.deeplearningbook.org. MIT Press, 2016.

Computation

Rewriting the equation from previous slide with h(t):

$$\boldsymbol{h}^{(t)} = f(\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}; \boldsymbol{\theta}),$$

The unfolded recurrence after t steps represented with a function g(t):

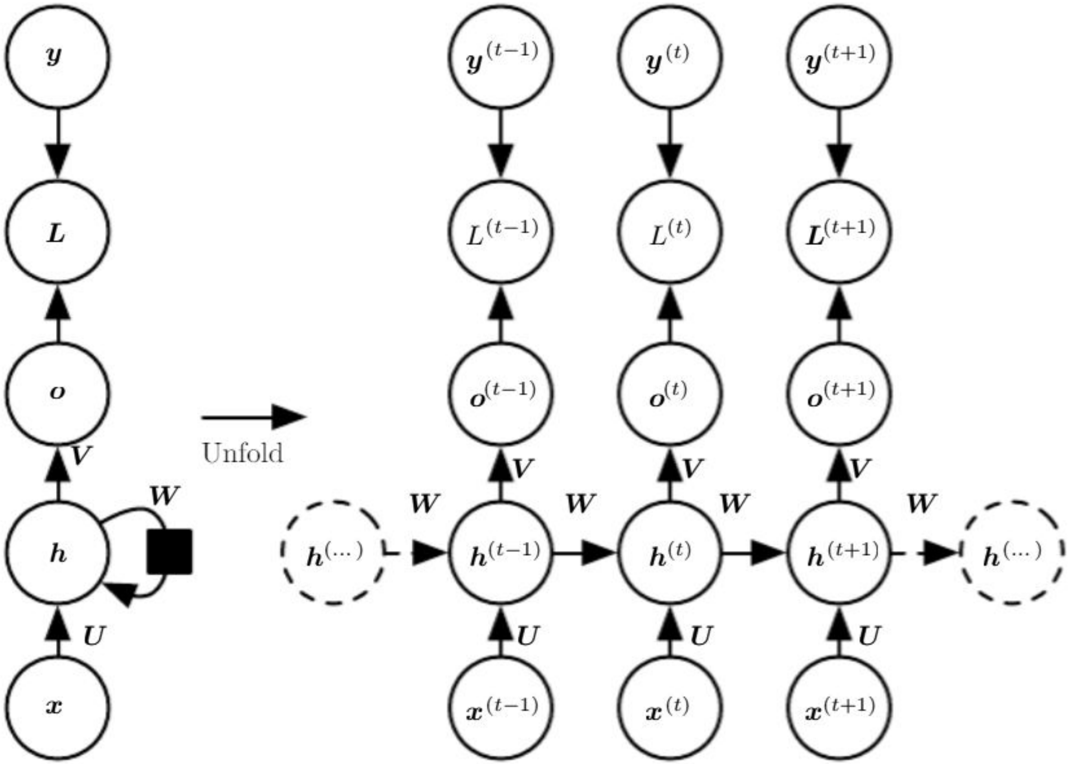
$$\begin{aligned}\boldsymbol{h}^{(t)} =& g^{(t)}(\boldsymbol{x}^{(t)}, \boldsymbol{x}^{(t-1)}, \boldsymbol{x}^{(t-2)}, \ldots, \boldsymbol{x}^{(2)}, \boldsymbol{x}^{(1)}) \\ =& f(\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}; \boldsymbol{\theta}).\end{aligned}$$

Recurrent Neural Networks

- **Recurrent Neural Networks (RNNs)** are a class of neural networks for processing sequential data.
- RNNs use **feedback loops** to process a sequence of data that allow information to persist.
- Reducing the complexity of parameters by **parameter sharing**
- A powerful tool in applications like text processing, speech recognition, language translation and DNA sequences, where the output depends on the previous computations.

RNNs by examples

- Example #1



from [1]

RNNs by examples (cont)

- Forward propagation, *t∈[1, τ]*:

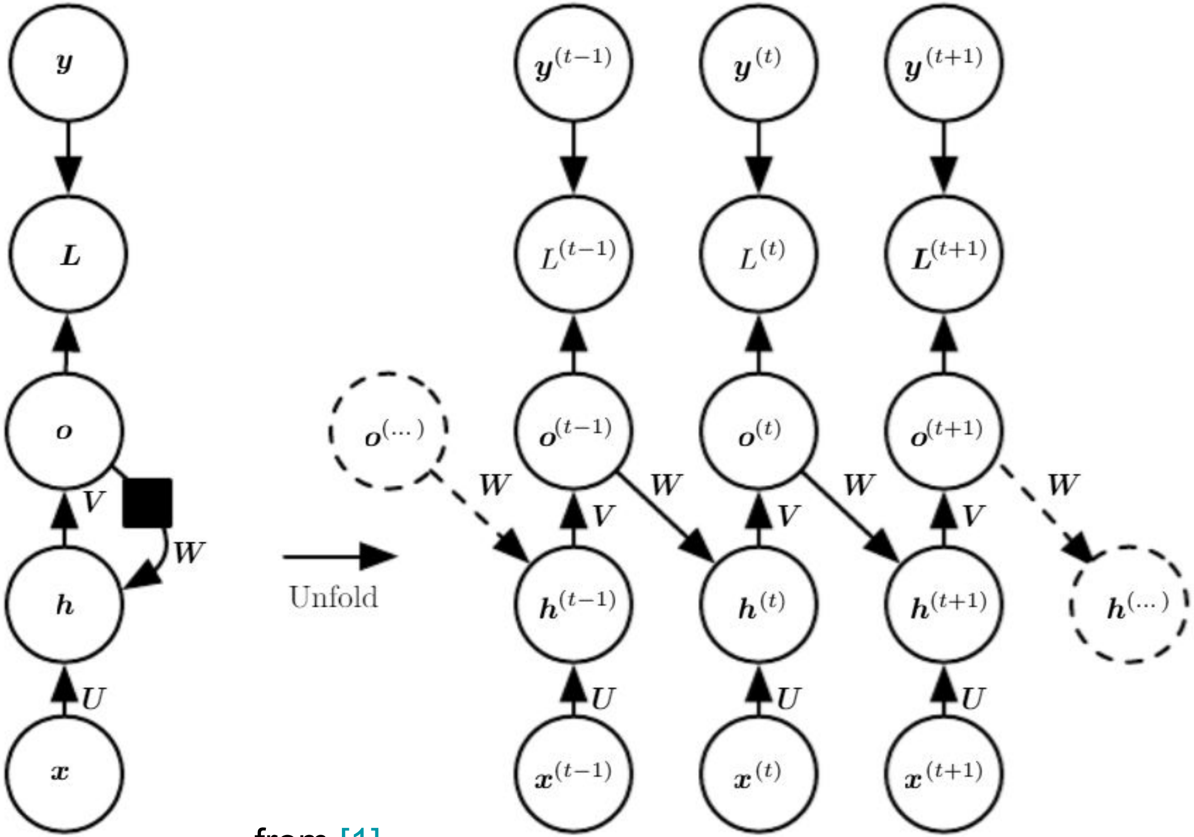
$$
\begin{aligned}
\boldsymbol{a}^{(t)} &= \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)}, \\
\boldsymbol{h}^{(t)} &= \tanh(\boldsymbol{a}^{(t)}), \\
\boldsymbol{o}^{(t)} &= \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(t)}, \\
\hat{\boldsymbol{y}}^{(t)} &= \operatorname{softmax}(\boldsymbol{o}^{(t)}),
\end{aligned}
$$

- Total loss:

$$
\begin{aligned}
& L\left(\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(\tau)}\}, \{\boldsymbol{y}^{(1)}, \ldots, \boldsymbol{y}^{(\tau)}\}\right) \\
&= \sum_t L^{(t)} \\
&= -\sum_t \log p_{\text{model}}\left(y^{(t)} \mid \{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(t)}\}\right)
\end{aligned}
$$

RNNs by examples (cont)

- Example #2



from [1]

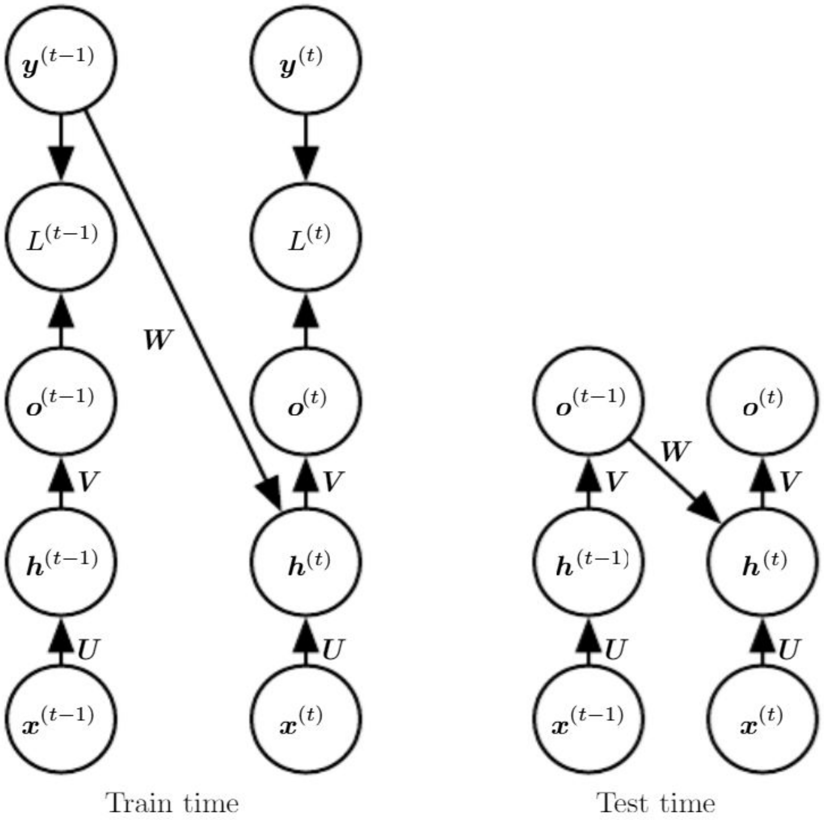
Teacher Forcing

- Conditional maximum likelihood criterion:

$$\log p\left(\boldsymbol{y}^{(1)}, \boldsymbol{y}^{(2)} \mid \boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}\right)$$
$$= \log p\left(\boldsymbol{y}^{(2)} \mid \boldsymbol{y}^{(1)}, \boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}\right) + \log p\left(\boldsymbol{y}^{(1)} \mid \boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}\right)$$

- Advantage: to avoid **BPTT** in models that lack hidden-to-hidden connections
- Disadvantage: works poorly in **open-loop** mode
  - In this case the kind of inputs that it will see during training time could be quite different from that it will see at test time

Teacher Forcing (cont)



from [1]

Computing the gradient

- Based on equations on slide 5

$$(\nabla_{\boldsymbol{o}^{(t)}} L)_i = \frac{\partial L}{\partial o_i^{(t)}} = \frac{\partial L}{\partial L^{(t)}} \frac{\partial L^{(t)}}{\partial o_i^{(t)}} = \hat{y}_i^{(t)} - \mathbf{1}_{i=y^{(t)}}$$

- $t = \boldsymbol{\tau}$:

$$\nabla_{\boldsymbol{h}^{(\tau)}} L = \boldsymbol{V}^\top \nabla_{\boldsymbol{o}^{(\tau)}} L$$

- $t \in [\boldsymbol{\tau}\text{-}1, 1]$

$$\nabla_{\boldsymbol{h}^{(t)}} L = \left(\frac{\partial \boldsymbol{h}^{(t+1)}}{\partial \boldsymbol{h}^{(t)}}\right)^\top (\nabla_{\boldsymbol{h}^{(t+1)}} L) + \left(\frac{\partial \boldsymbol{o}^{(t)}}{\partial \boldsymbol{h}^{(t)}}\right)^\top (\nabla_{\boldsymbol{o}^{(t)}} L)$$

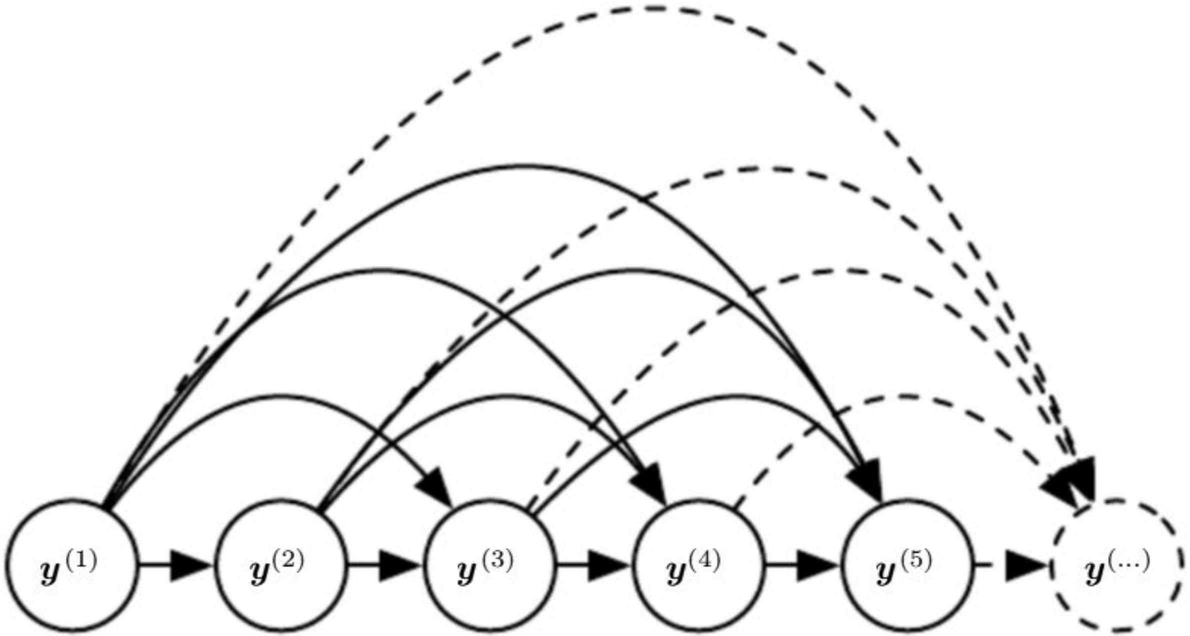
$$= \boldsymbol{W}^\top \mathrm{diag}\left(1 - \left(\boldsymbol{h}^{(t+1)}\right)^2\right) (\nabla_{\boldsymbol{h}^{(t+1)}} L) + \boldsymbol{V}^\top (\nabla_{\boldsymbol{o}^{(t)}} L)$$

Computing the gradient (cont)

$$
\begin{aligned}
\nabla_{\boldsymbol{c}} L &= \sum_t \left(\frac{\partial \boldsymbol{o}^{(t)}}{\partial \boldsymbol{c}}\right)^\top \nabla_{\boldsymbol{o}^{(t)}} L = \sum_t \nabla_{\boldsymbol{o}^{(t)}} L, \\
\nabla_{\boldsymbol{b}} L &= \sum_t \left(\frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{b}^{(t)}}\right)^\top \nabla_{\boldsymbol{h}^{(t)}} L = \sum_t \operatorname{diag}\left(1 - \left(\boldsymbol{h}^{(t)}\right)^2\right) \nabla_{\boldsymbol{h}^{(t)}} L, \\
\nabla_{\boldsymbol{V}} L &= \sum_t \sum_i \left(\frac{\partial L}{\partial o_i^{(t)}}\right) \nabla_{\boldsymbol{V}^{(t)}} o_i^{(t)} = \sum_t \left(\nabla_{\boldsymbol{o}^{(t)}} L\right) \boldsymbol{h}^{(t)^\top}, \\
\nabla_{\boldsymbol{W}} L &= \sum_t \sum_i \left(\frac{\partial L}{\partial h_i^{(t)}}\right) \nabla_{\boldsymbol{W}^{(t)}} h_i^{(t)} \\
&= \sum_t \operatorname{diag}\left(1 - \left(\boldsymbol{h}^{(t)}\right)^2\right) \left(\nabla_{\boldsymbol{h}^{(t)}} L\right) \boldsymbol{h}^{(t-1)^\top}, \\
\nabla_{\boldsymbol{U}} L &= \sum_t \sum_i \left(\frac{\partial L}{\partial h_i^{(t)}}\right) \nabla_{\boldsymbol{U}^{(t)}} h_i^{(t)} \\
&= \sum_t \operatorname{diag}\left(1 - \left(\boldsymbol{h}^{(t)}\right)^2\right) \left(\nabla_{\boldsymbol{h}^{(t)}} L\right) \boldsymbol{x}^{(t)^\top},
\end{aligned}
$$

RNN as Directed Graphical Models

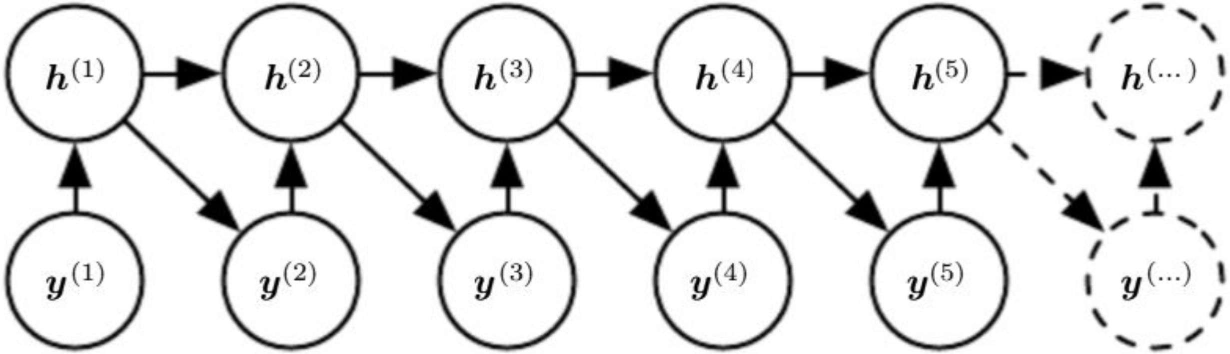
- Ignoring the hidden units
- inefficient



from [1]

RNN as Directed Graphical Models (cont)

- very efficient parametrization



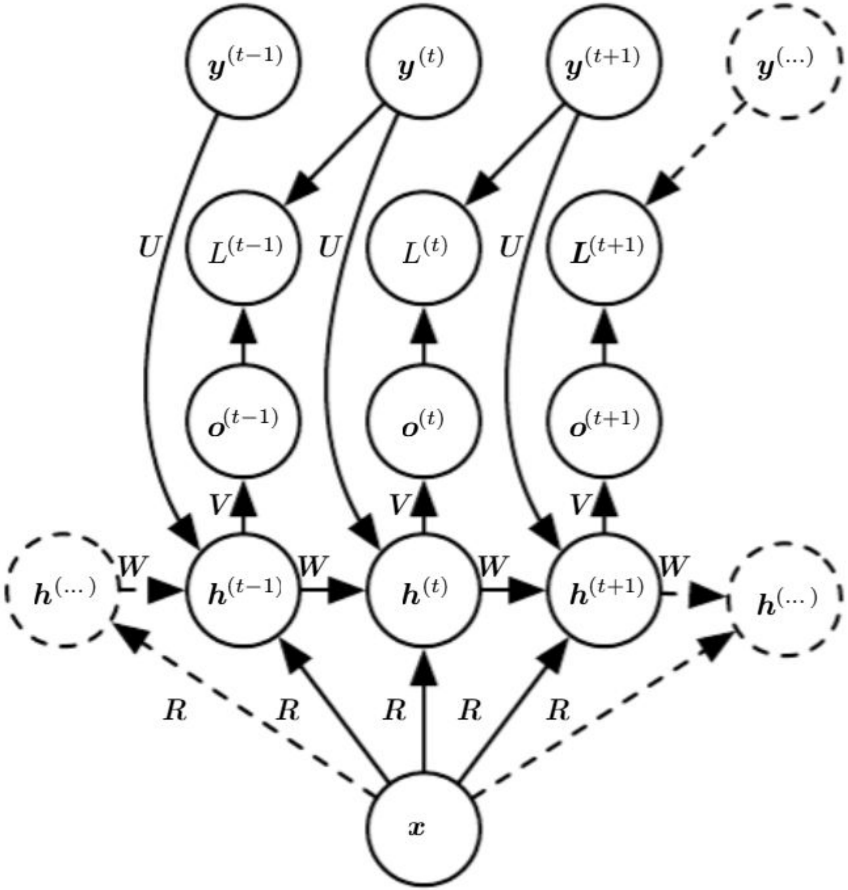
from [1]

RNN as Directed Graphical Models (cont)

- Determining the length of the sequence
  - Special symbol at the end of the sequence
  - extra Bernoulli output
  - Predicting sequence length $\boldsymbol{\tau}$

Modeling Sequences Conditioned on Context

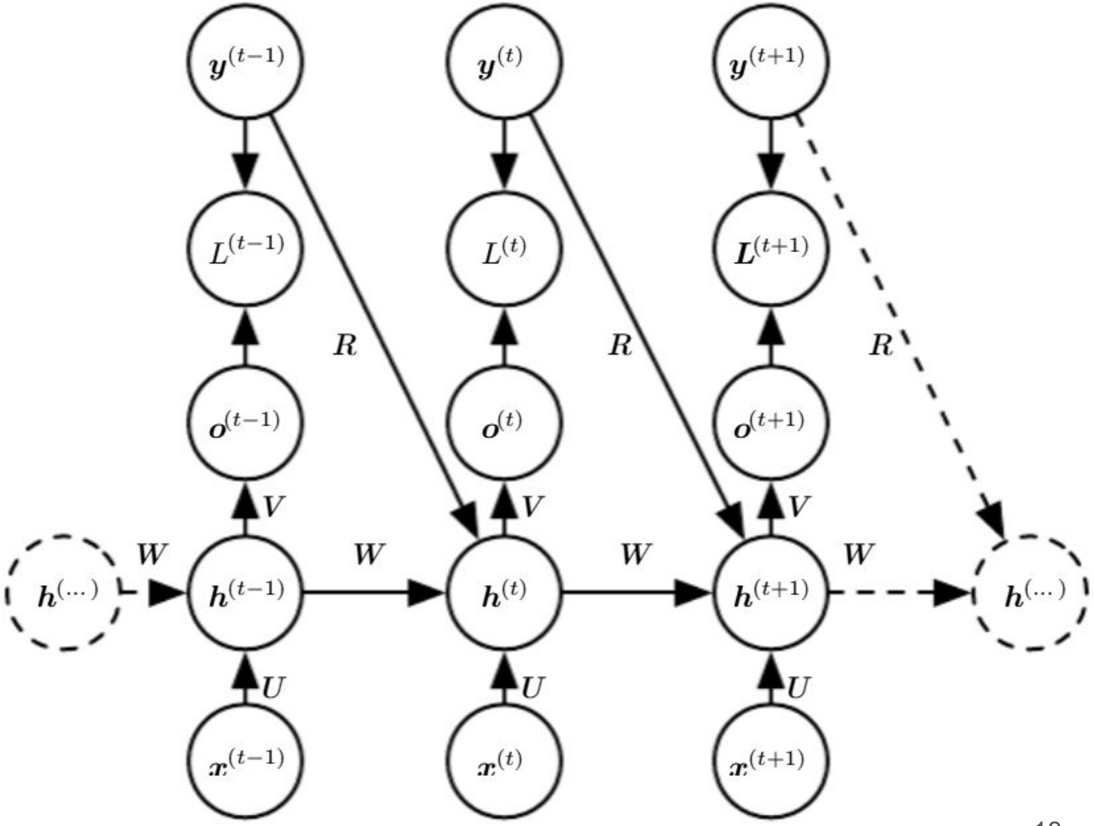
- A single vector as input



from [1]

Modeling Sequences Conditioned on Context (cont)

- A sequence of vectors as input



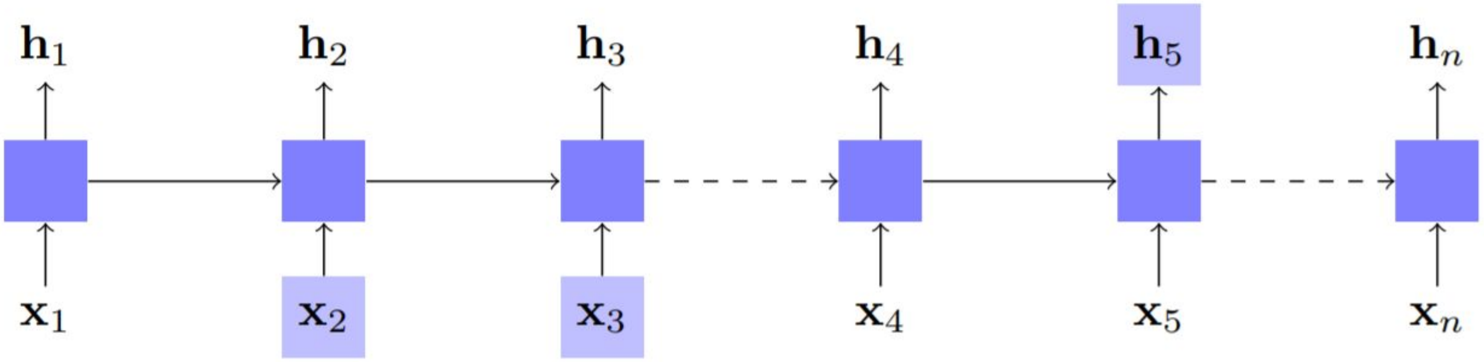
from [1]

Long Term Dependencies

- Vanishing and exploding gradients in long-term propagation.
- Exponentially smaller magnitude of gradient for long term dependencies.
- Gradient based optimization is difficult.

Why do we want longterm dependencies?

I grew up in France ... I speak fluent French ...



Reference: RNNII, DLVR, Lecture by Dr. Michael Weinmann, Informatik, University of Bonn

Long Short Term Memory (LSTM)

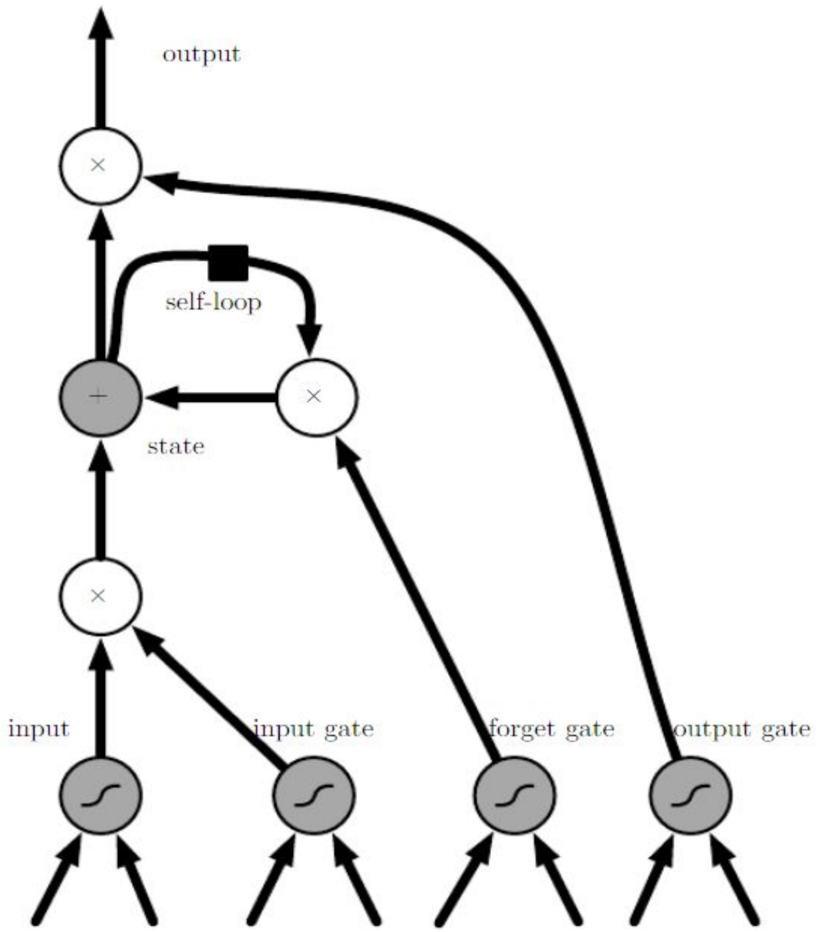


Figure: LSTM recurrent network cell block diagram.

Reference: RNNII, DLVR, Lecture by Dr. Michael Weinmann, Informatik, University of Bonn

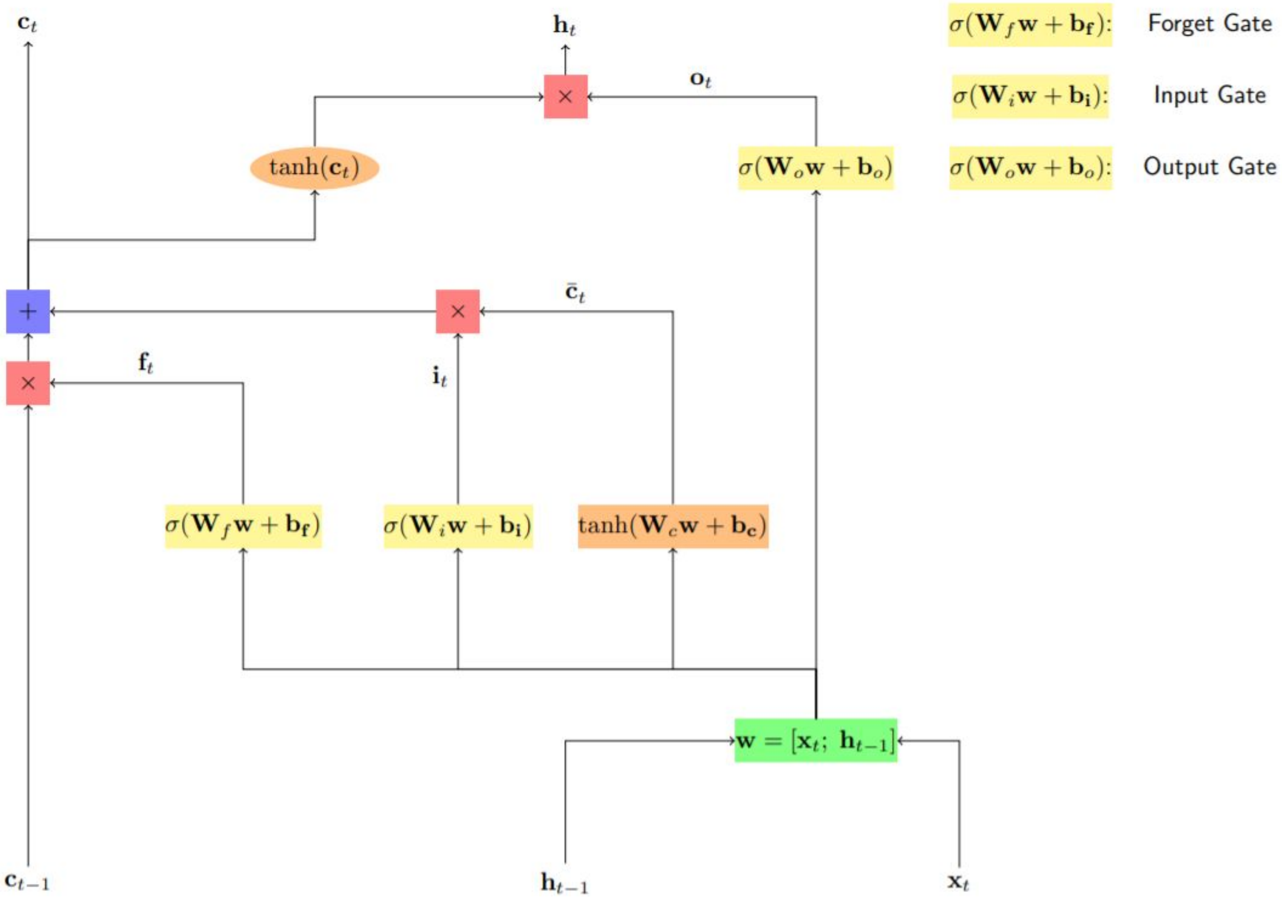


Figure: LSTM with gate equations.

Reference: RNNII, DLVR, Lecture by Dr. Michael Weinmann, Informatik, University of Bonn

The LSTM is a differentiable memory cell. We have now three gates: an input gate $\boldsymbol{i}^{(t)}$, a forgetting gate $\boldsymbol{f}^{(t)}$and an output gate $\boldsymbol{o}^{(t)}$:

$$
\begin{aligned}
\boldsymbol{i}^{(t)} &= g_i(\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}; \boldsymbol{\theta}_i) \\
\boldsymbol{f}^{(t)} &= g_f(\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}; \boldsymbol{\theta}_f) \\
\boldsymbol{o}^{(t)} &= g_o(\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}; \boldsymbol{\theta}_o)
\end{aligned}
$$

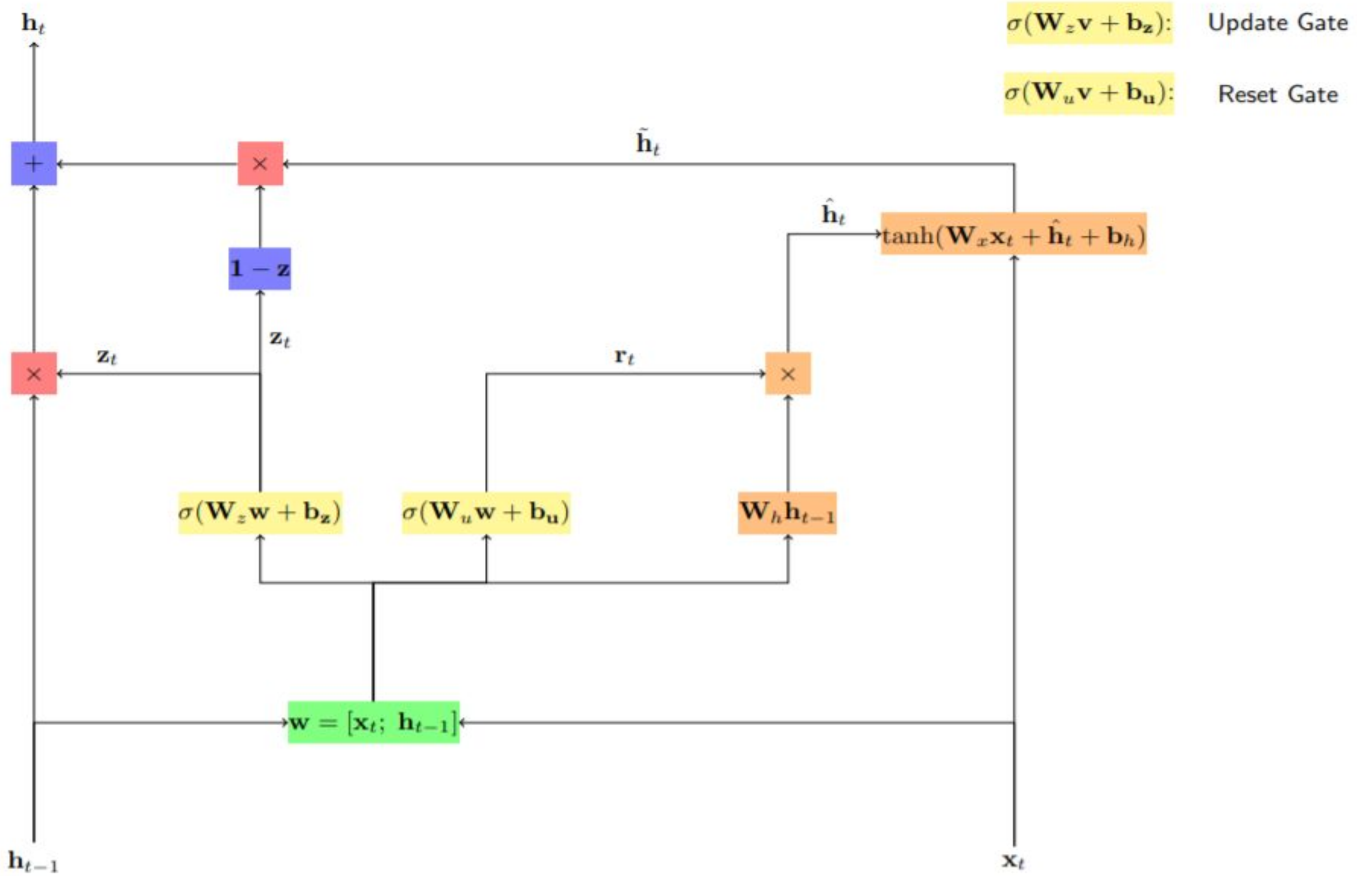
We now define a memory cell $\boldsymbol{c}^{(t)}$ by gating previous memory and new content

$$\boldsymbol{c}^{(t)} = \boldsymbol{f}^{(t)} \cdot \boldsymbol{c}^{(t-1)} + \boldsymbol{i}^{(t)} \cdot \tanh(\boldsymbol{a}^{(t)}) \quad \text{where} \quad \boldsymbol{a}^{(t)} = \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)}.$$

The final hidden state is a gated activation of the memory cell:

$$\boldsymbol{h}_{\mathsf{LSTM}}^{(t)} = \boldsymbol{o}^{(t)} \cdot \tanh \boldsymbol{c}^{(t)}.$$

Gated Recurrent Unit



Reference: RNNII, DLVR, Lecture by Dr. Michael Weinmann, Informatik, University of Bonn

References

[1] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep Learning. http://www.deeplearningbook.org. MIT Press, 2016.

[2] RNNII, DLVR, Lecture by Dr. Michael Weinmann, Informatik, University of Bonn