Comprendre les données visuelles à grande échelle

ENSIMAG
2018-2019

Karteek Alahari & Diane Larlus

https://project.inria.fr/bigvisdata/

Au programme

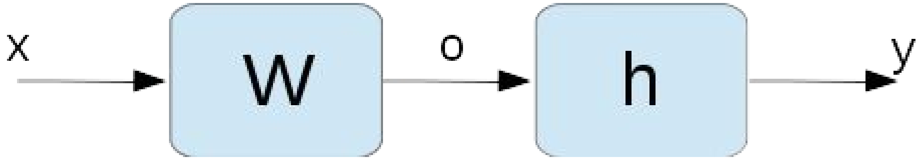
- Organisation du cours
- Introduction
  - Contexte et applications
  - Aperçus des taches
  - Evaluation
- Représentation des données visuelles
  - Descripteurs locaux et globaux, réseaux de neurones
  - Application à la fouille de donnée
- Problème de la reconnaissance
  - Classification d’images et de vidéo
  - Séparateurs à Vaste marge (SVM)
  - Pour aller plus loin : **RNN**

Crédits pour les transparents: T. Lucas, M. Pedersoli

Outline

- Motivation: why and when are RNN useful
- Formal definition of a RNN
- Backpropagation Through Time
- Vanishing Gradients
- Improved RNN
- Regularization for RNN
- Teacher Forcing Training

Introduction

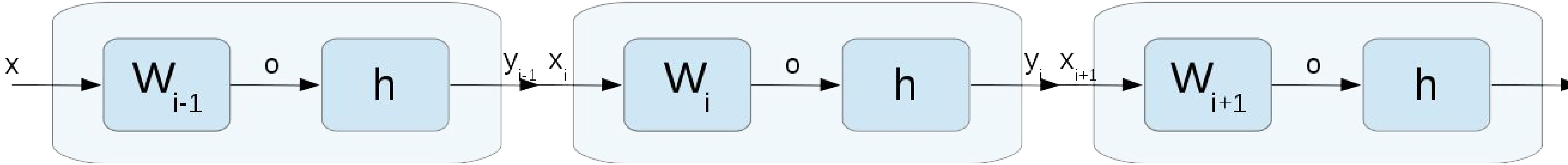


Building block: y = *h*(Wx)

W = linear (also convolutional, low rank, etc...)

h = element-wise non-linear operation (tanh, reLU, etc..)

Multilayer Network



Same input and output, but hidden representations: more powerful!

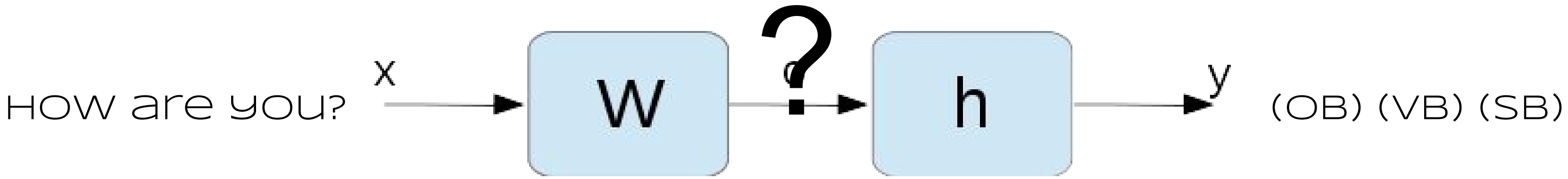
Composition of building blocks

Can still learn parameters with backpropagation

Sequential Input/Output

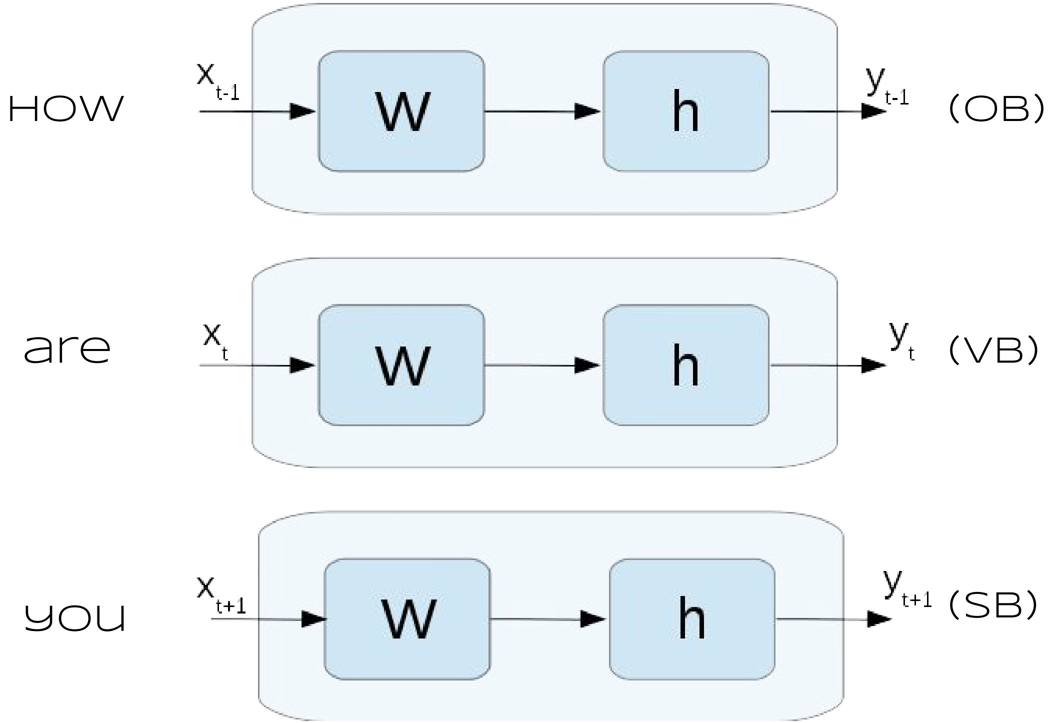
So far x,y fixed size vectors --> limiting factor!

We want input output with a complex structure eg. variable length sequence!



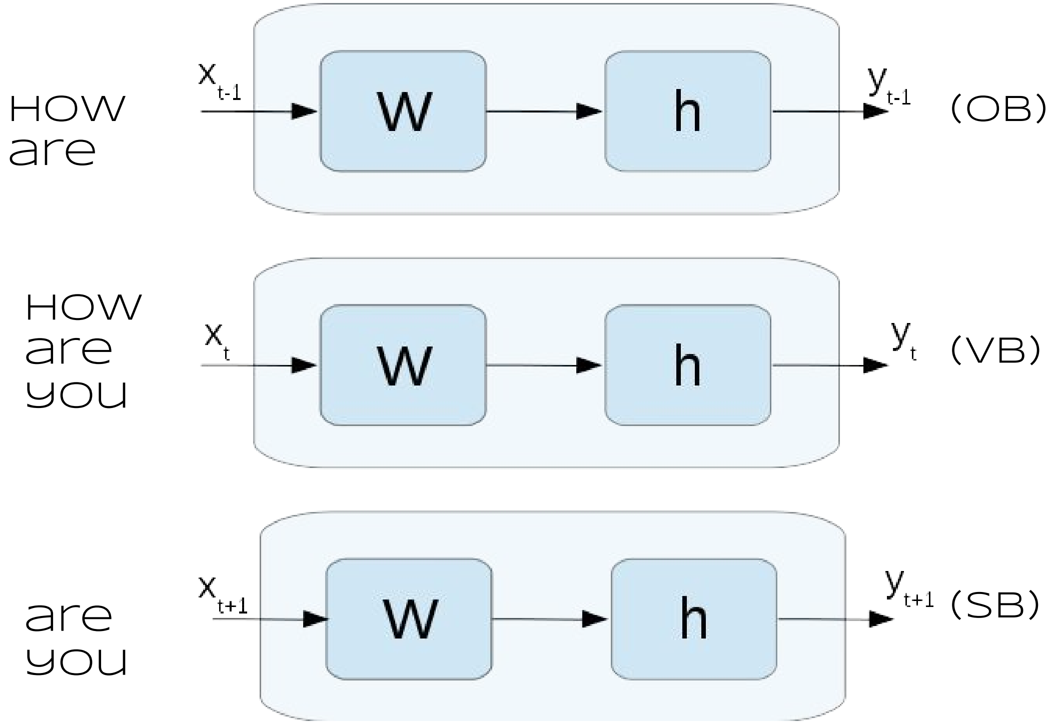
Can we reuse the same building block?

Repeated Network



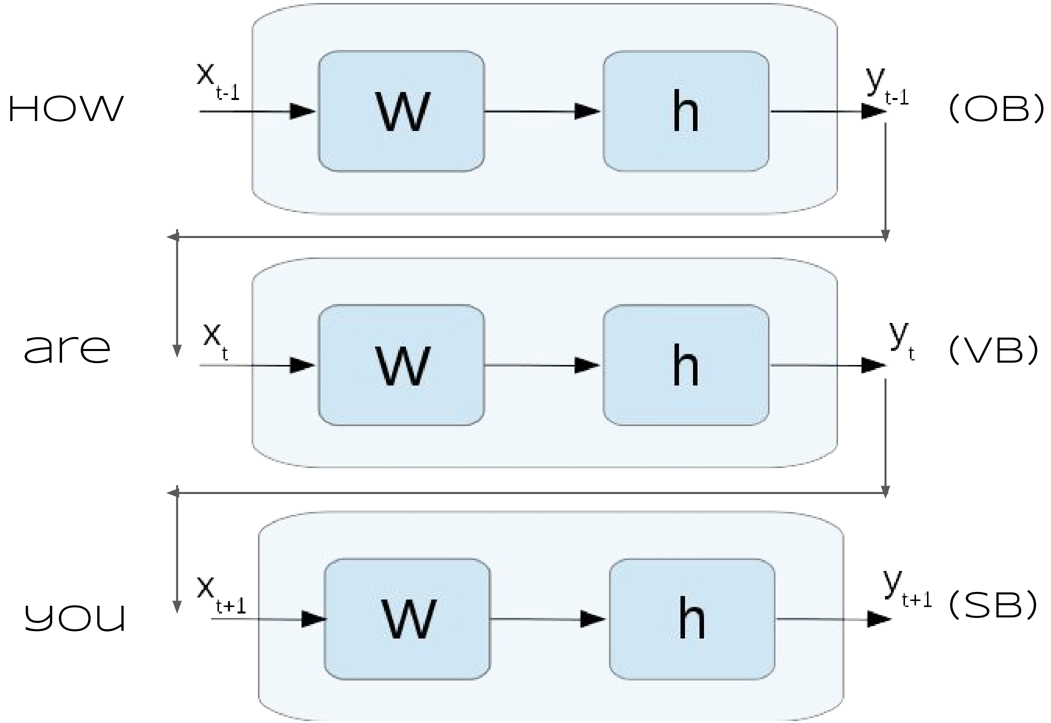
+ Works!
+ Scale well with data because reuses same parameters!
- No context, every decision is taken independently!

Convolutional Network



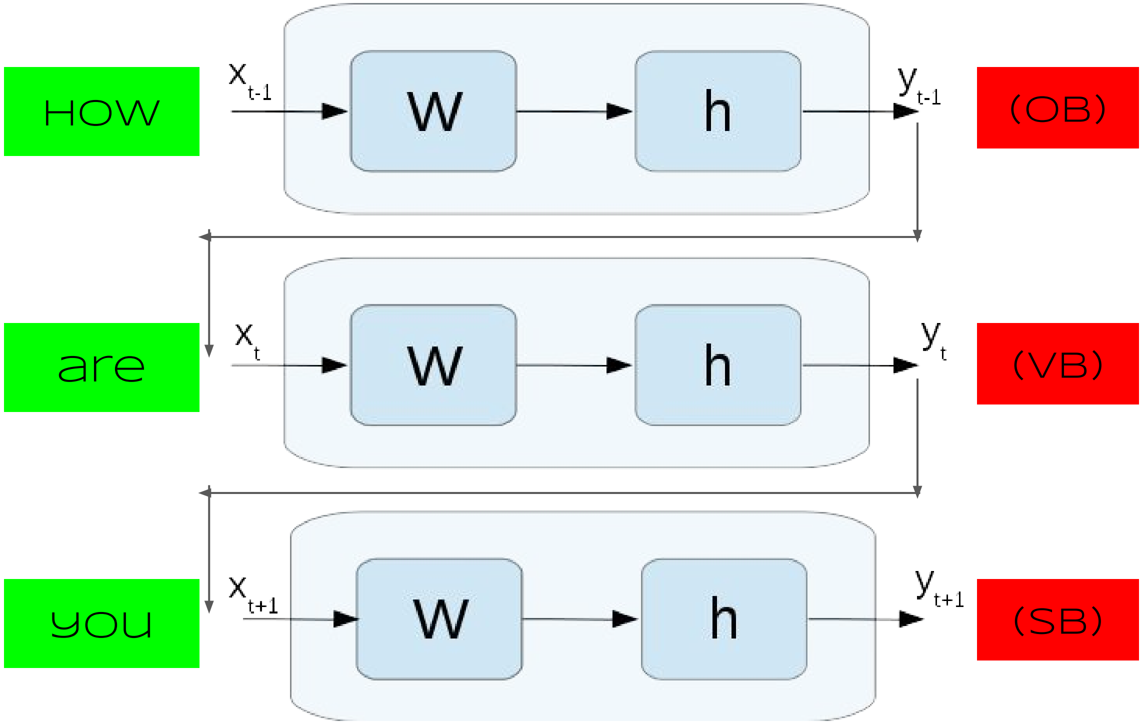
- \+ Works!
- \+ Scale well with data because same parameters reused!
- \+ Context, every decision depends also on the neighbours!
- \- Context has fixed, predefined structure!
- \- Does not scale if we want long range, sparse correlations!

Recurrent Network



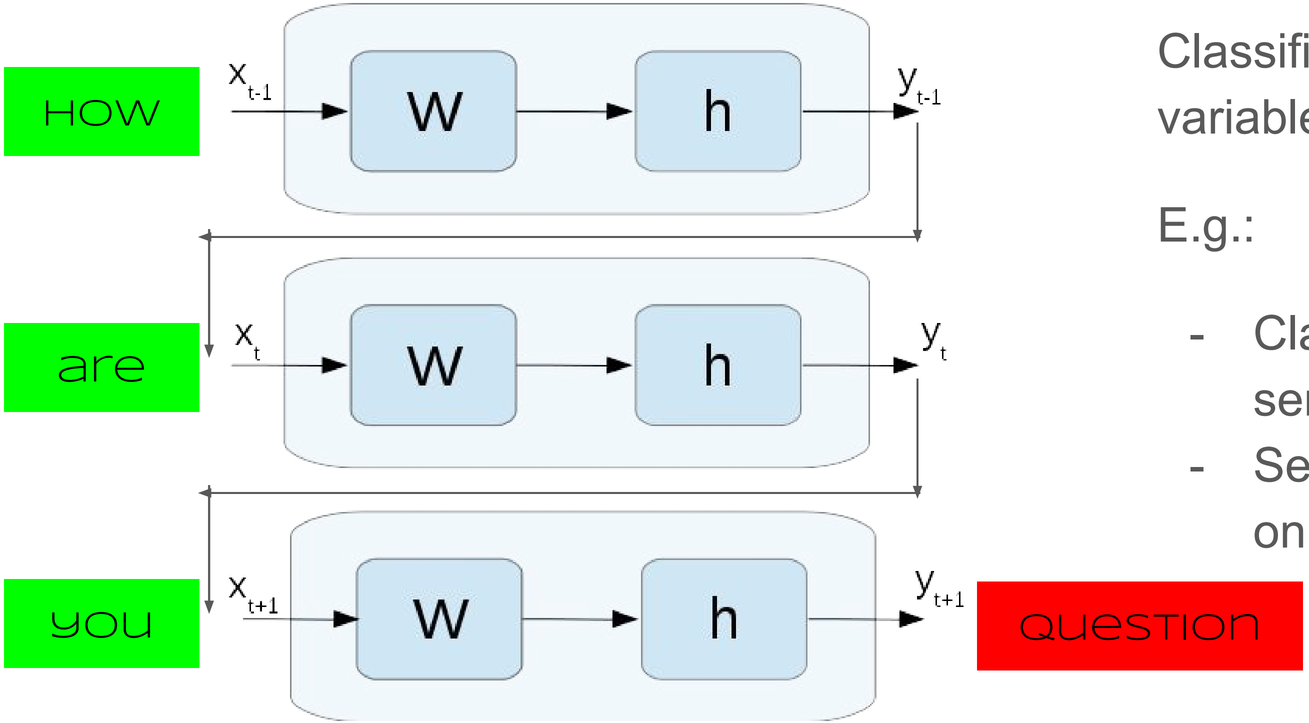
- \+ Works!
- \+ Scale well with data because same parameters reused!
- \+ State vector can summarize all the past, thus global context with long term dependencies!
- \+ Same optimization as before with Backpropagation

RNN topology: Many to Many (Coupled)



- Works!
- Scale well with data because same parameters reused!
- State vector can summarize all the past, thus global context with long term dependencies!
- Same optimization as before with Backpropagation

RNN topology: Many to one

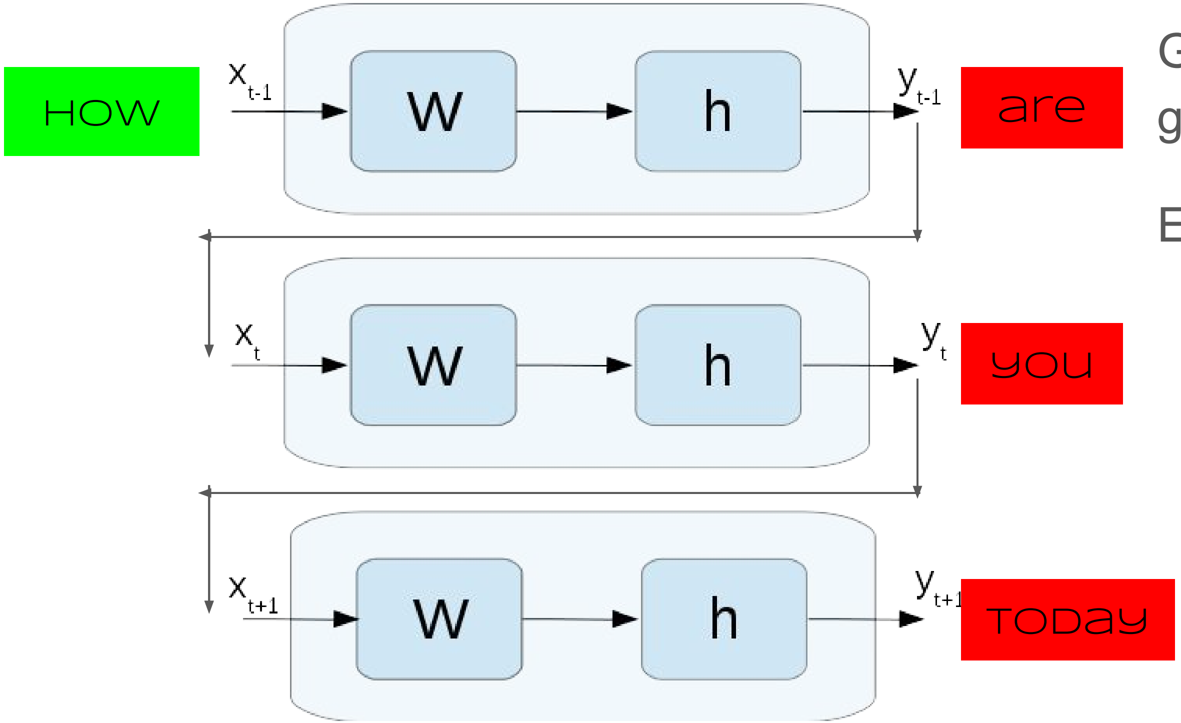


Classification problems with variable input size.

E.g.:

- Classify the category of a sentence
- Sentiment Classification on a audio track

RNN topology: One to many

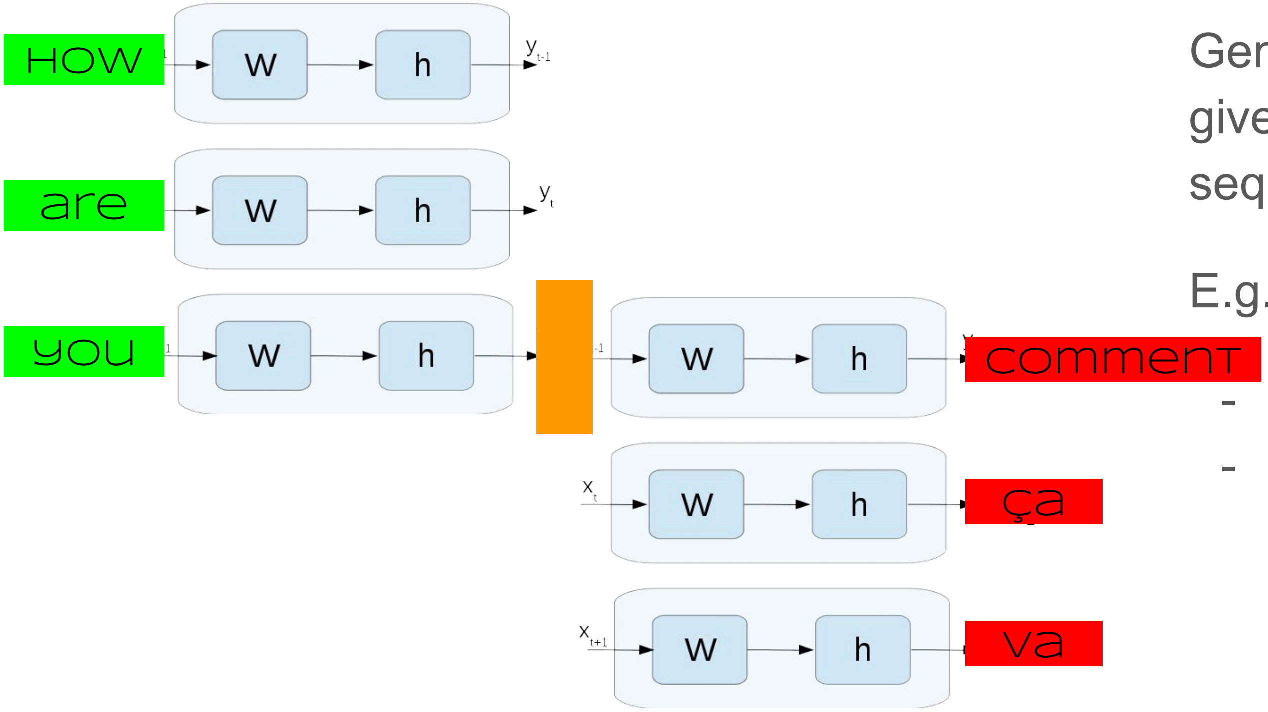


Generation of a sequence given an initial state.

E.g.:

- Generate a sentence describing an image
- Generate a sentence given the first word

RNN topology: Many to Many (Encoder/Decoder)

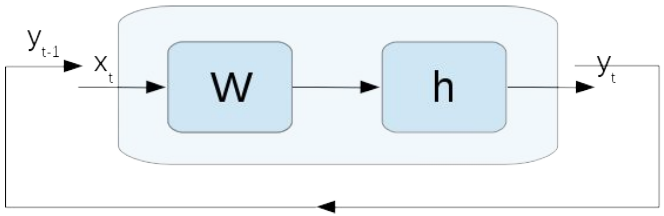


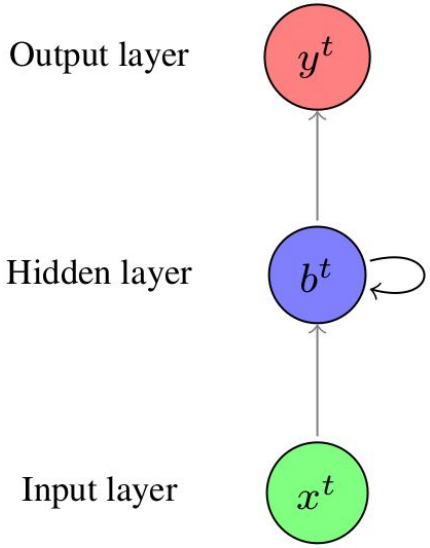
Generation of a sequence given a different length sequence.

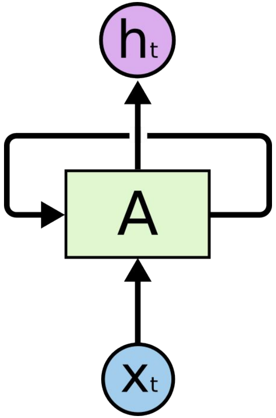
E.g.:

- Translate a sentence
- Describe a movie

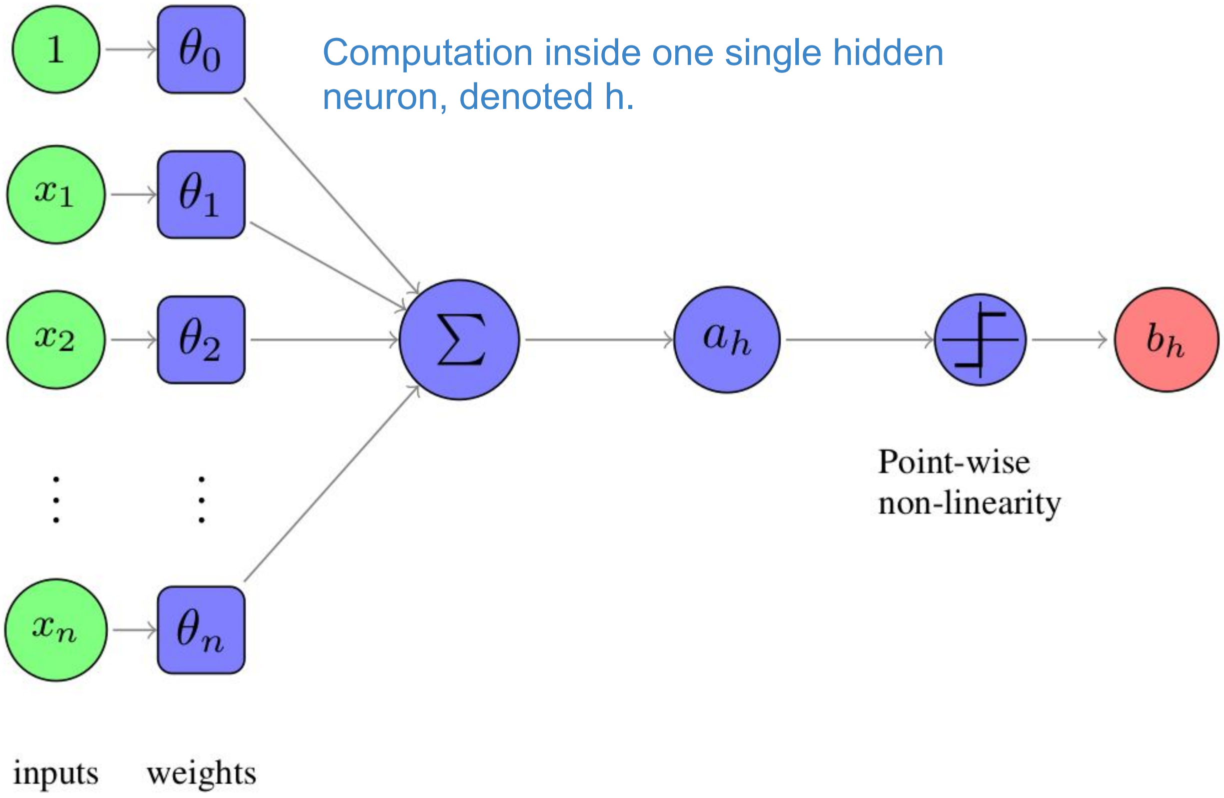
RNN graphical representation







Forward propagation in a single neuron

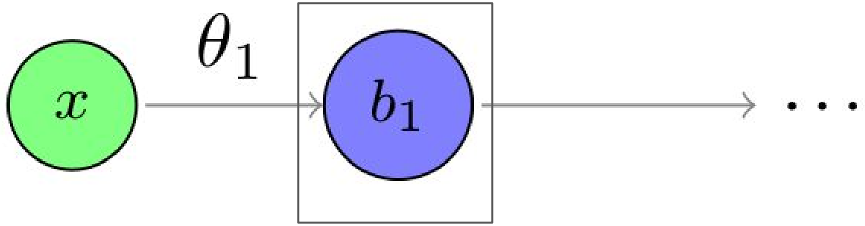


Computation inside one single hidden neuron, denoted h.

$$a_h^t = \sum_{i=1}^{I} \theta_{ih} x_i^t$$

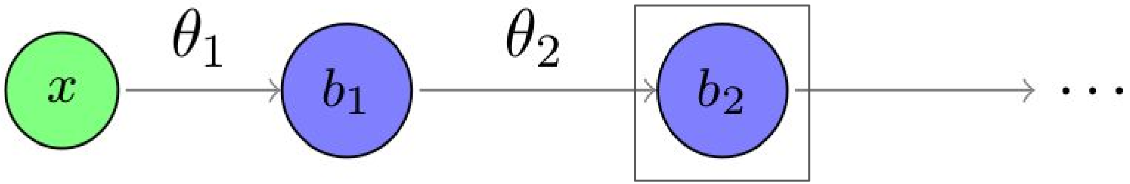
$$b_h^t = \sigma_h(a_h^t)$$

A network with one neuron per layer



$$\boxed{b_1 = \sigma(\theta_1 x)}$$

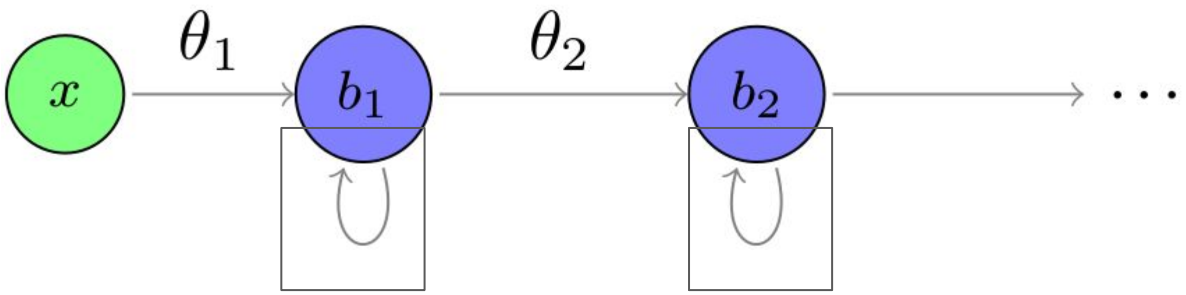
A network with one neuron per layer



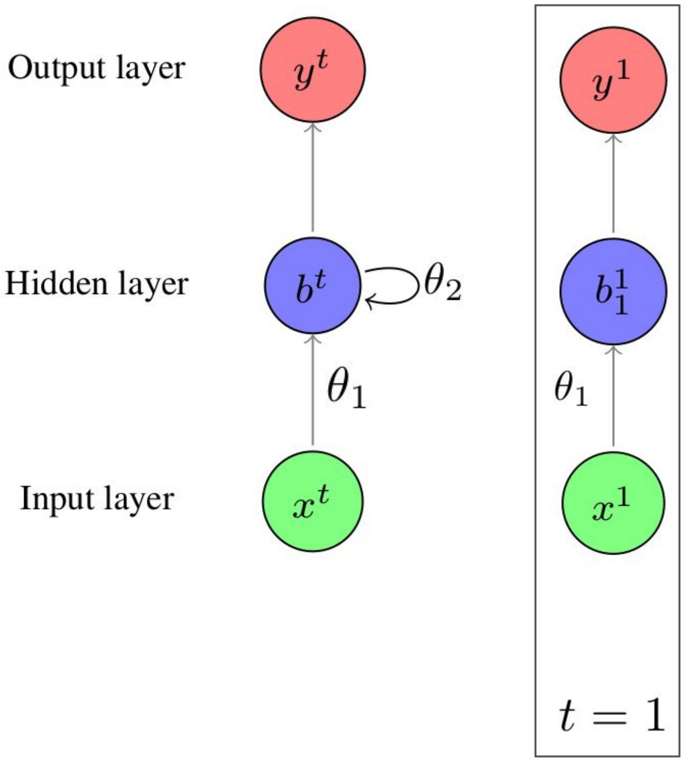
$$b_1 = \sigma(\theta_1 x)$$

$$\boxed{\begin{aligned} b_2 &= \sigma(\theta_2 b_1) \\ &= \sigma(\theta_2 \sigma(\theta_1 x)) \end{aligned}}$$

Making the network recurrent

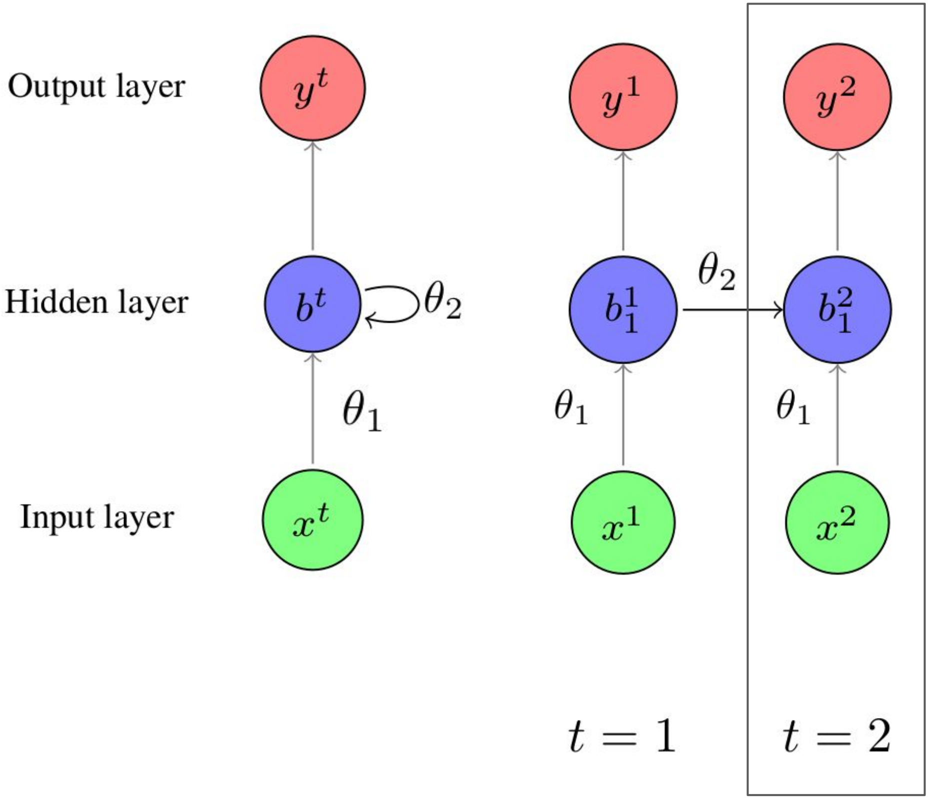


Unfolding the graph in time



$$b_1^1 = \sigma(\theta_1 x^1 + 0)$$

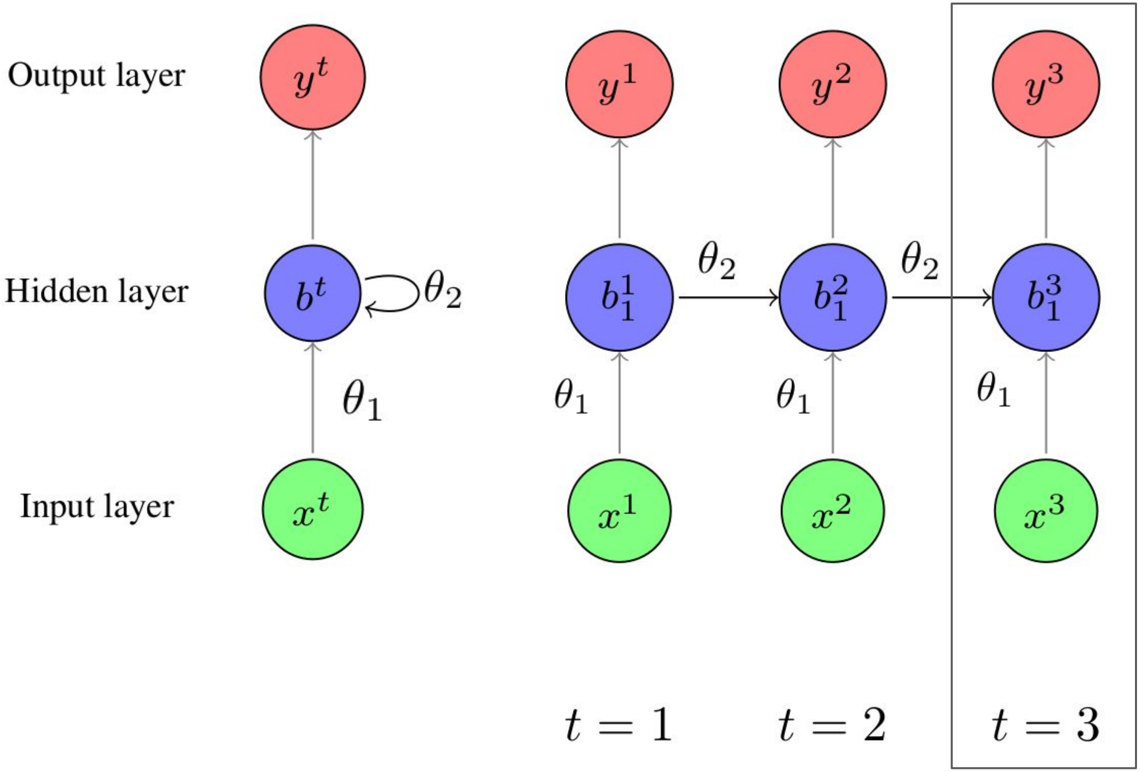
Unfolding the graph in time



$$b_1^1 = \sigma(\theta_1 x^1 + 0)$$

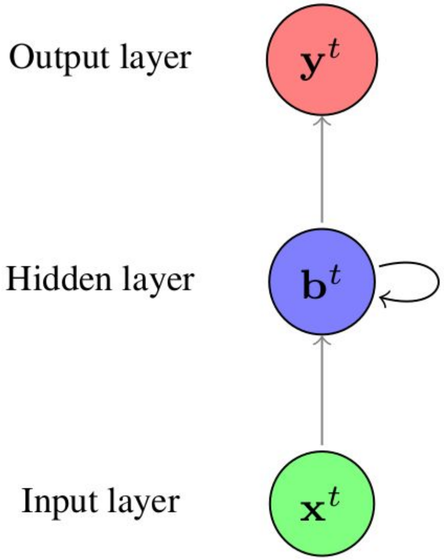
$$\boxed{\begin{aligned} a_1^2 &= \theta_1 x^2 + \theta_2 b_1^1 \\ b_1^2 &= \sigma(a_1^2) \\ &= \sigma(\theta_1 x^2 + \theta_2 b_1^1) \end{aligned}}$$

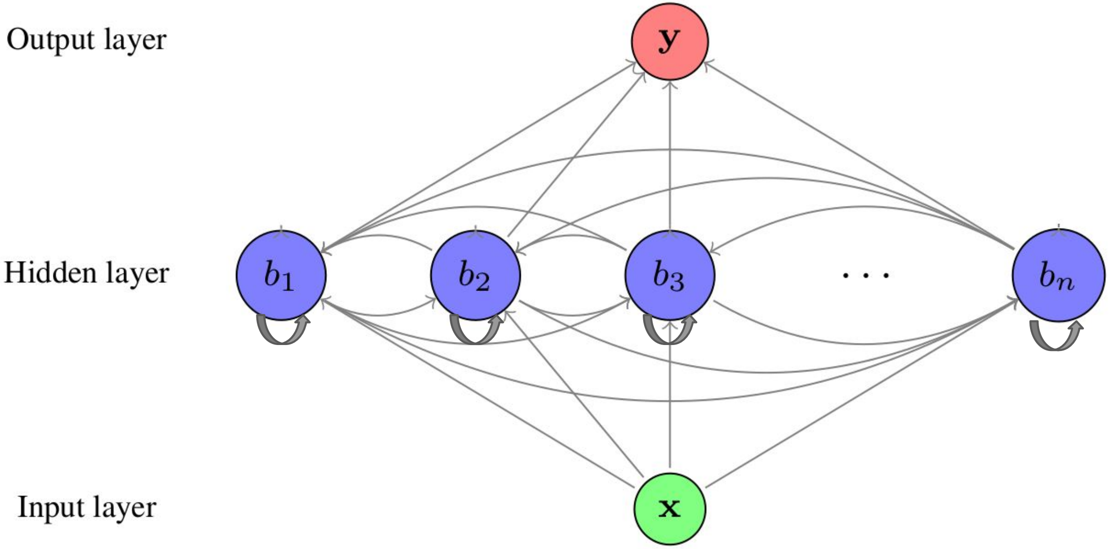
Unfolding the graph in time



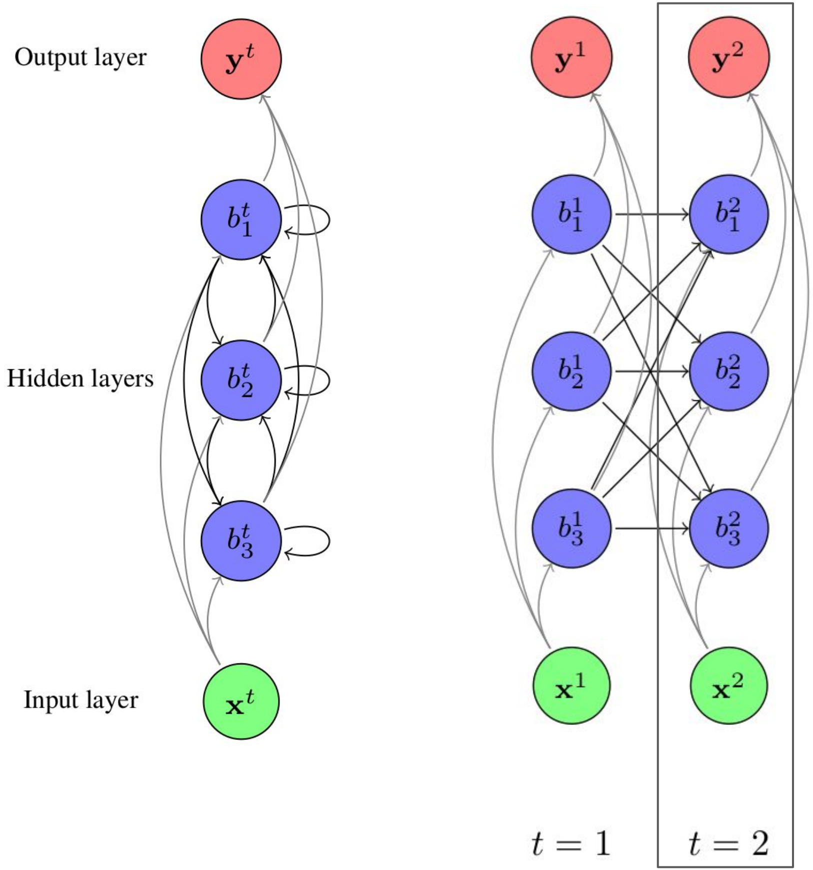
$$
\begin{aligned}
a_1^3 &= \theta_1 x^3 + \theta_2 b_1^2 \\
b_1^3 &= \sigma(a_1^3) \\
&= \sigma(\theta_1 x^3 + \theta_2 \sigma(\theta_1 x^2 + \theta_2 b_1^1))
\end{aligned}
$$

A recurrent network with one hidden layer

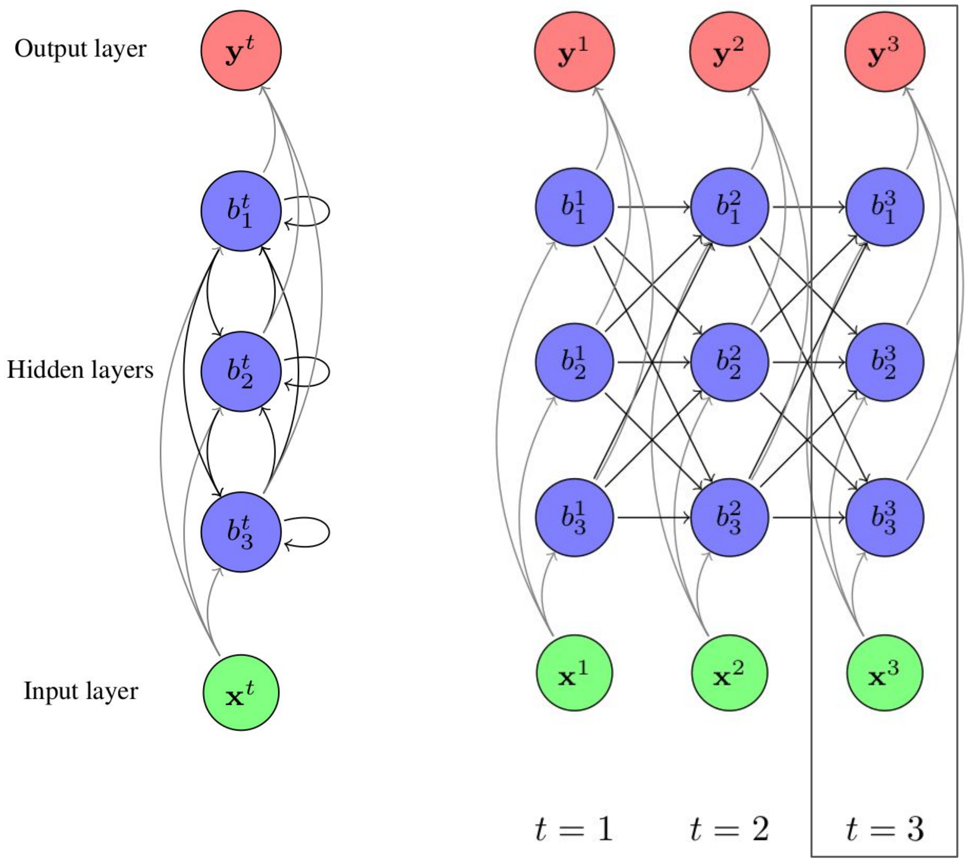




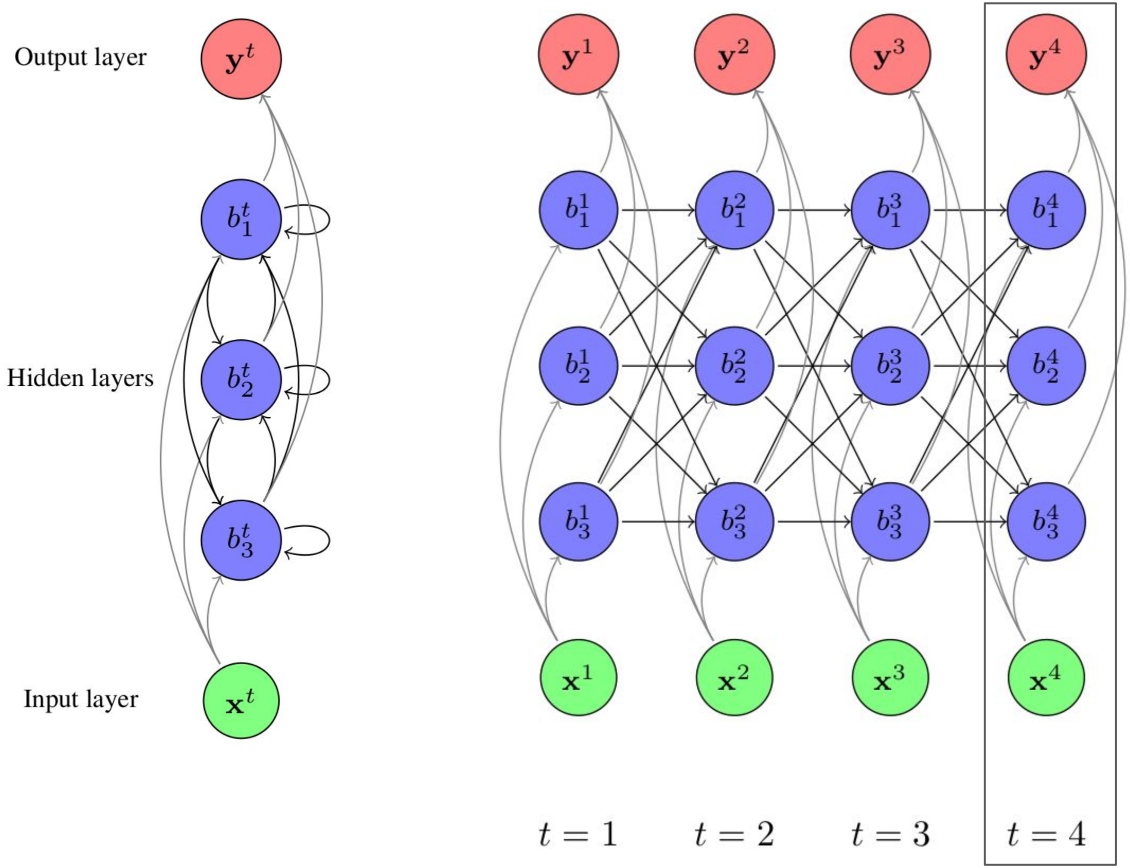
Unfolding the graph



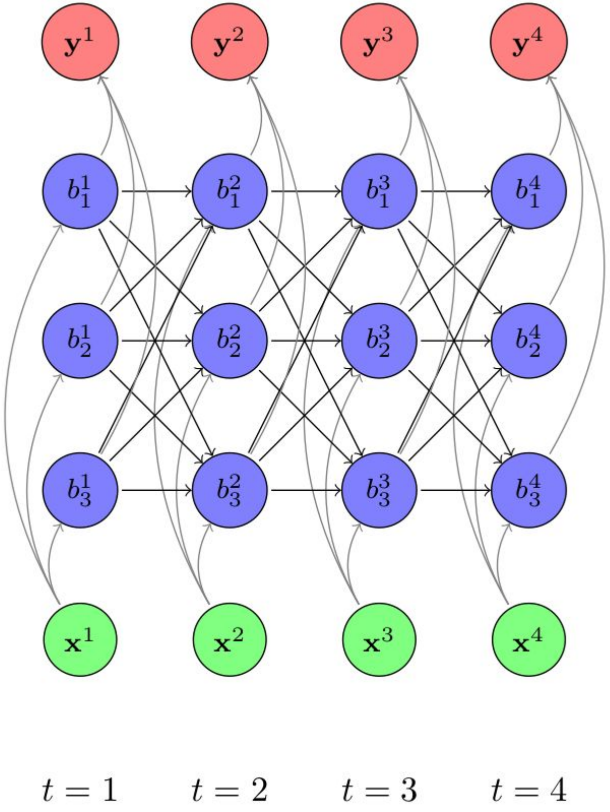
Unfolding the graph



Unfolding the graph



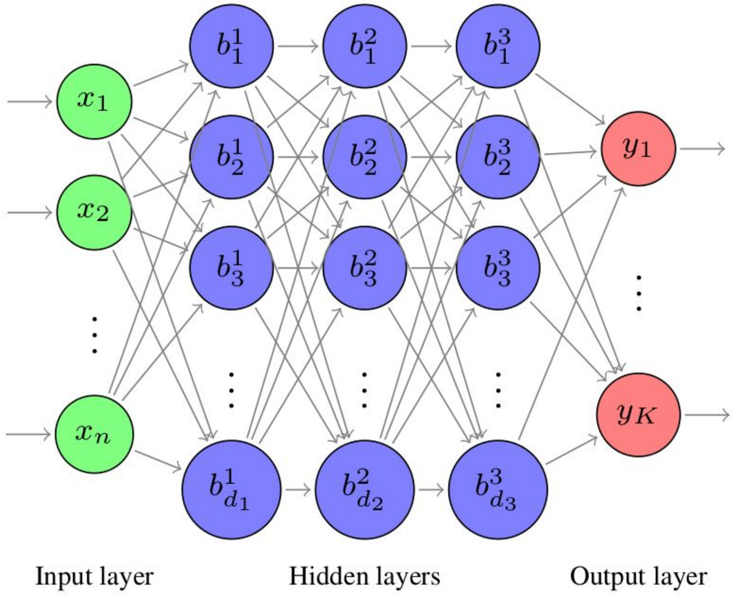
Equations of the recurrence

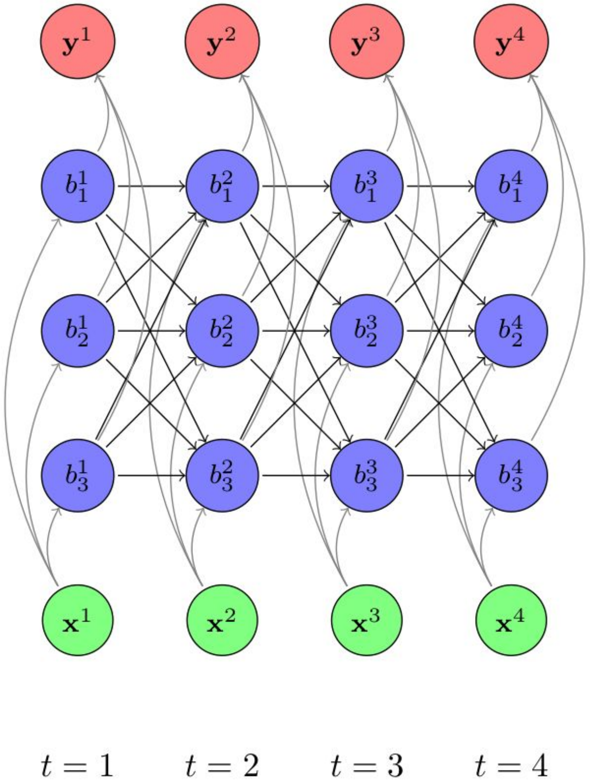


$$
\begin{aligned}
a_h^t &= \sum_{i=1}^{I} \theta_{ih} x_i^t + \sum_{h'=1}^{H} \theta_{h'h} b_{h'}^{t-1} \\
b_h^t &= \sigma_h(a_h^t)
\end{aligned}
$$

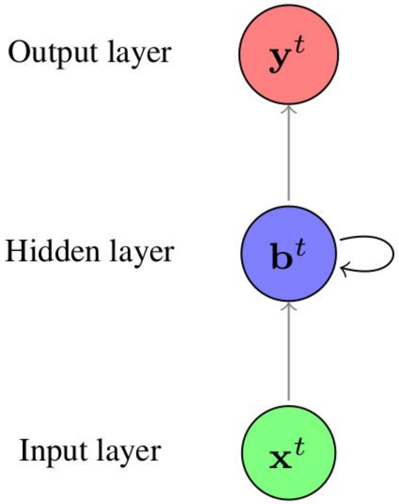
$$
\begin{aligned}
\mathbf{a}^t &= \theta_{in}^T \mathbf{x} + \theta_{rec}^T \mathbf{a}^{t-1} \\
\mathbf{b}^t &= \sigma(\mathbf{a}^t)
\end{aligned}
$$

The unfolded graph is a feed-forward graph



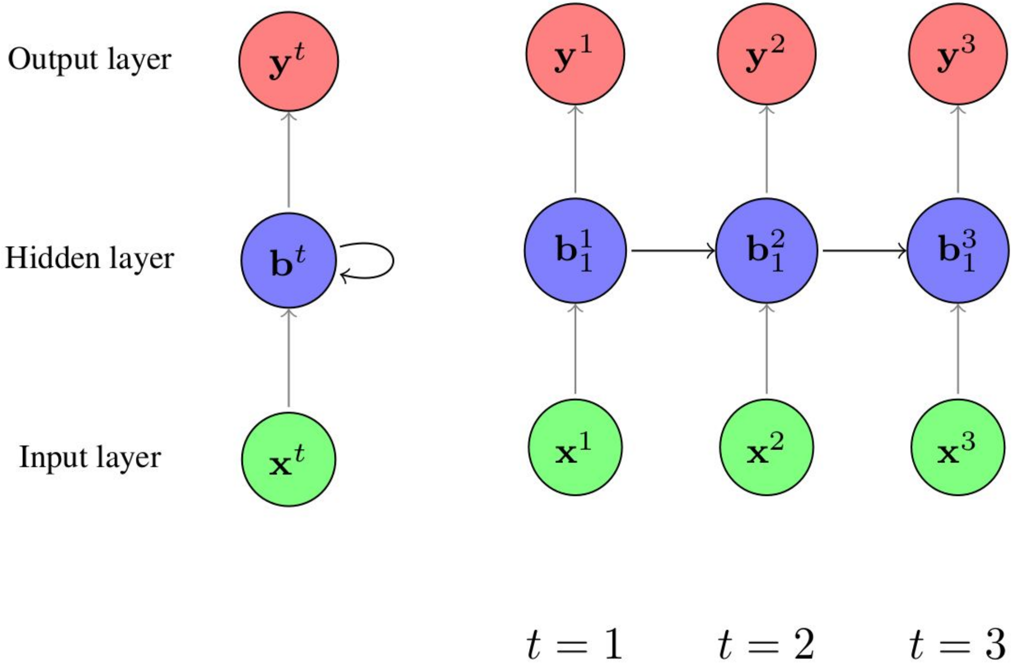


Recap



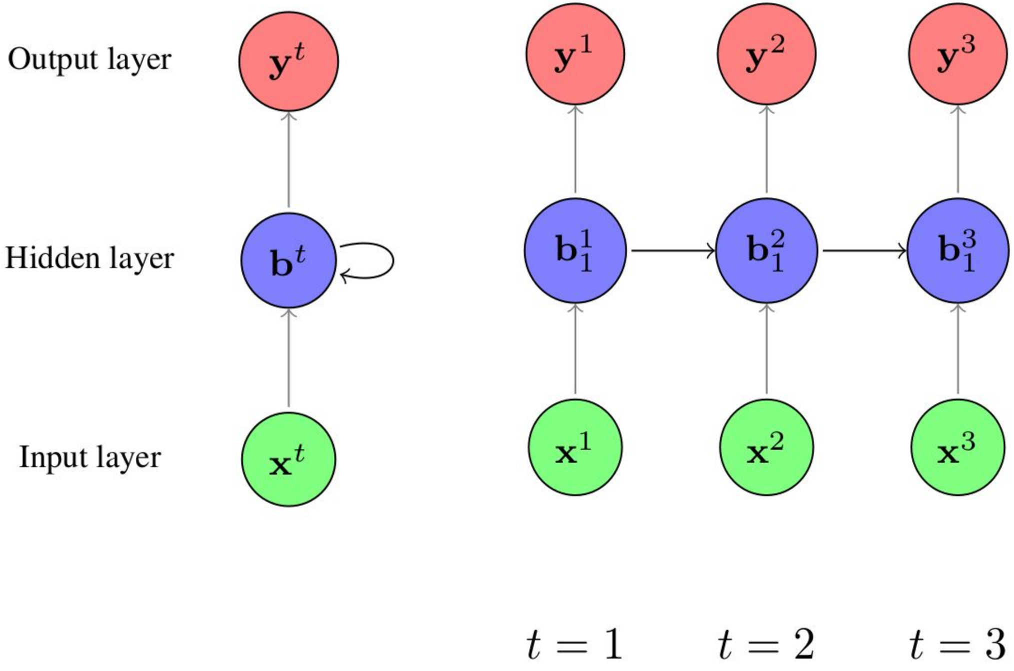
Output layer
$\mathbf{y}^t$
Hidden layer
$\mathbf{b}^t$
Input layer
$\mathbf{x}^t$

Recap



Output layer
Hidden layer
Input layer
$\mathbf{y}^t$
$\mathbf{b}^t$
$\mathbf{x}^t$
$\mathbf{y}^1$
$\mathbf{y}^2$
$\mathbf{y}^3$
$\mathbf{b}_1^1$
$\mathbf{b}_1^2$
$\mathbf{b}_1^3$
$\mathbf{x}^1$
$\mathbf{x}^2$
$\mathbf{x}^3$
$t = 1$
$t = 2$
$t = 3$

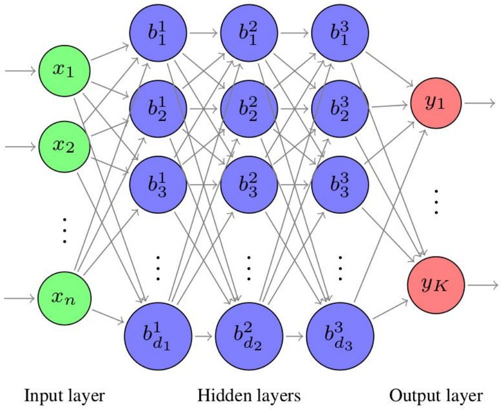
Unfolding the graph in time

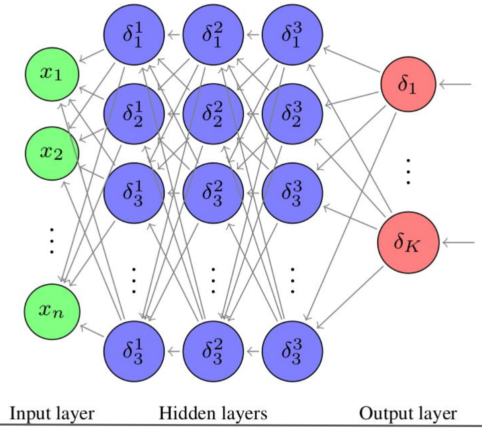


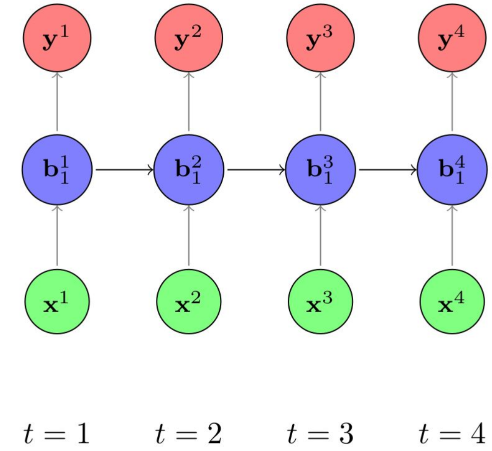
$$
\begin{aligned}
a_h^t &= \sum_{i=1}^{I} \theta_{ih} x_i^t + \sum_{h'=1}^{H} \theta_{h'h} b_{h'}^{t-1} \\
b_h^t &= \sigma_h(a_h^t)
\end{aligned}
$$

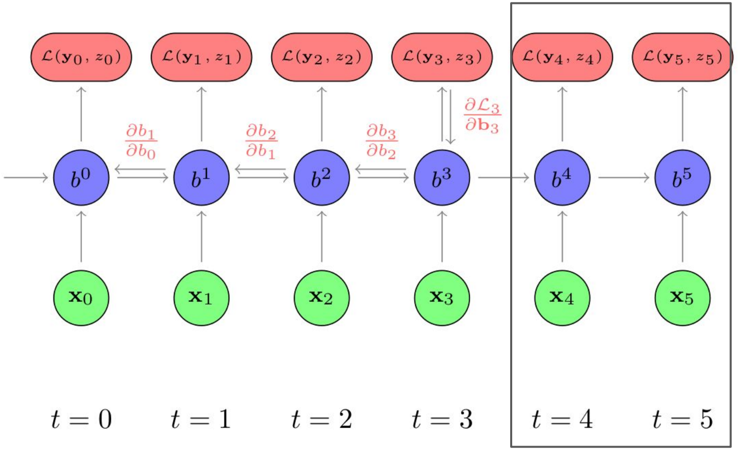
$$
\begin{aligned}
\mathbf{a}^t &= \theta_{in}^T \mathbf{x} + \theta_{rec}^T \mathbf{a}^{t-1} \\
\mathbf{b}^t &= \sigma(\mathbf{a}^t)
\end{aligned}
$$

Back-propagation in a recurrent network

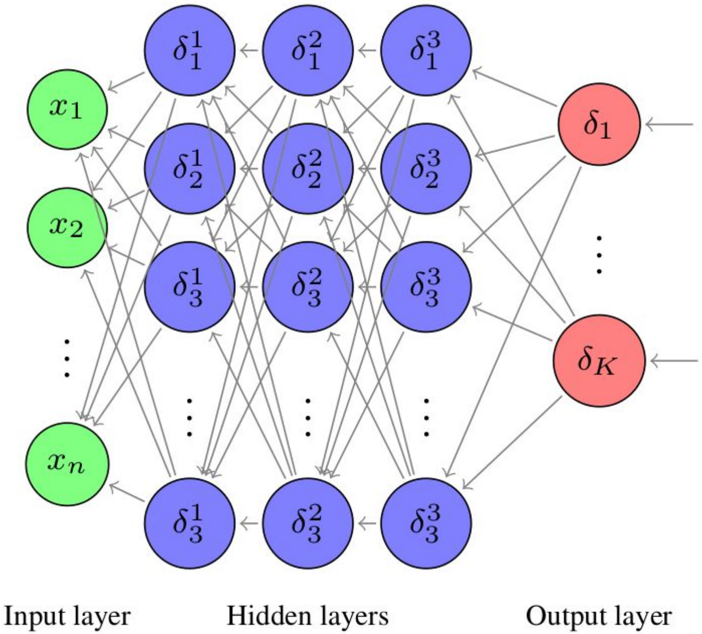


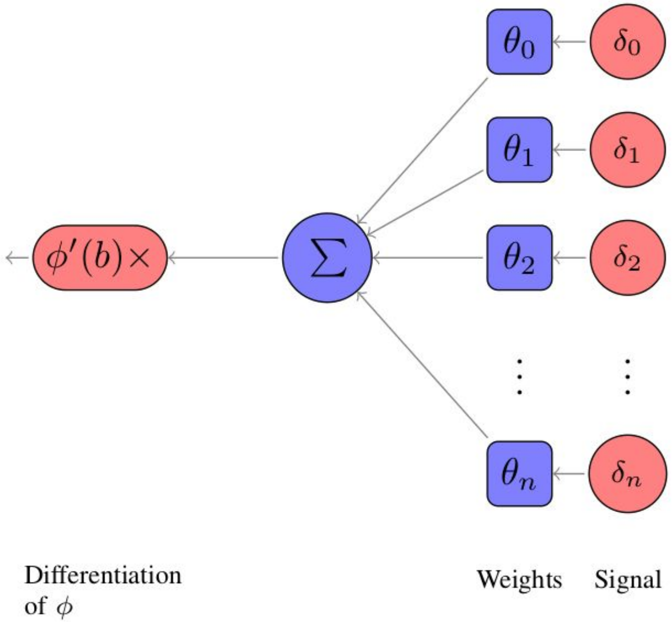






Equations of the back-propagation





$$\delta_j = \frac{\partial \mathcal{L}}{\partial a_j}$$

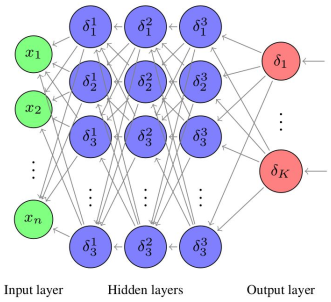
$$\delta_h = \sigma'(a_h) \sum_{k=1}^{K} \delta_k \theta_{hk}$$

Last hidden layer

$$\delta_h = \sigma'(a_h) \sum_{h' \in H^{l+1}} \delta_{h'} \theta_{hh'}$$

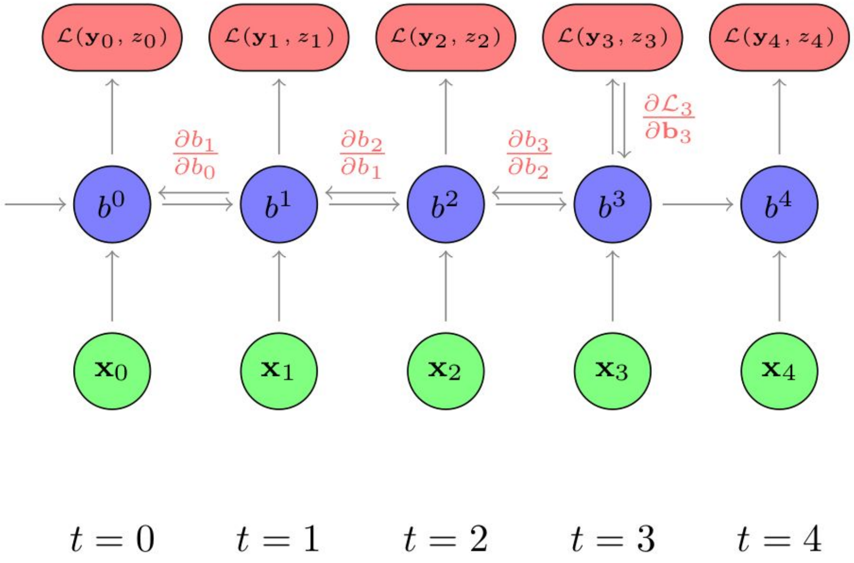
All other hidden layers

Equations of the back-propagation



$$\delta_h = \sigma'(a_h)\sum_{k=1}^{K}\delta_k\theta_{hk}$$

$$\delta_h = \sigma'(a_h)\sum_{h'\in H^{l+1}}\delta_{h'}\theta_{hh'}$$



$$\delta_j^t = \frac{\partial \mathcal{L}}{\partial a_j^t}$$

$$\delta_h^t = \sigma'(a_h^t)\left(\sum_{k=1}^{K}\delta_k^t\theta_{hk} + \sum_{h'=1}^{H}\delta_{h'}^{t+1}\theta_{hh'}\right)$$

$$\frac{\partial \mathcal{L}}{\partial \theta_{ij}} = \sum_{t=1}^{T}\frac{\partial \mathcal{L}}{\partial a_j^t}\frac{\partial a_j^t}{\partial \theta_{ij}} = \sum_{t=1}^{T}\delta_j^t b_i^t$$

The vanishing gradient problem

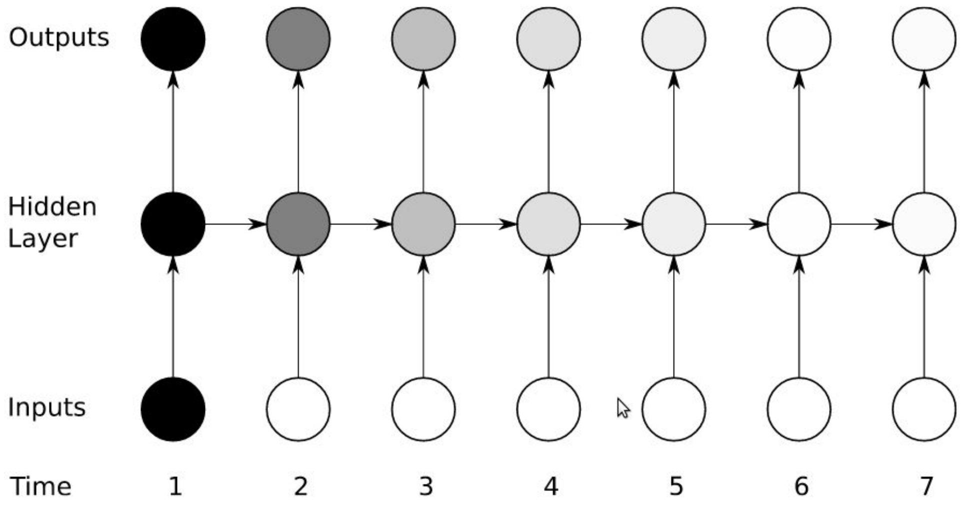
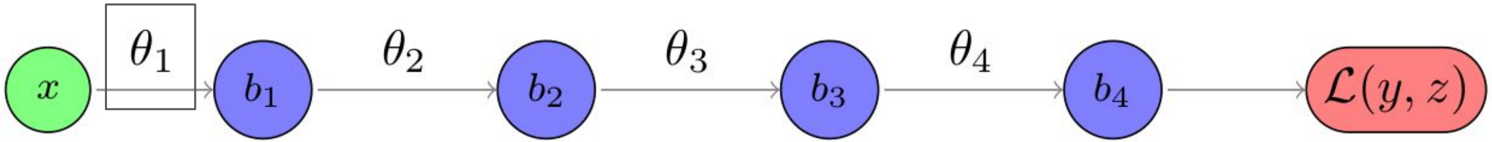


Figure credit: Alex Graves

A simple case of vanishing gradient



$$\frac{\partial \mathcal{L}}{\partial \theta_1} = \frac{\partial \mathcal{L}}{\partial h_4} \times \theta_4 \times \sigma'(h_3) \times \theta_3 \times \sigma'(h_2) \times \theta_2 \times \sigma'(h_1) \times x$$

Improved RNNs

- Gated Units
  - Long-Short Term Memory RNN
  - Gated Recurrent Unit
- Skip connections
  - Clockwork RNN
  - Hierarchical Multi-resolution RNN