Deep Learning

Ms Computer Science,
Data Science, Cybersecurity
2nd semester - 6 CFU

Part 3 by Alessandro Sperduti

References:
Deep Learning Book (main book)

Sequence Modeling: Recurrent and Recursive Nets (chapter 10)

…and Transformers

Learning in Sequential Domains

Large amounts of data in sequential form more and more common:

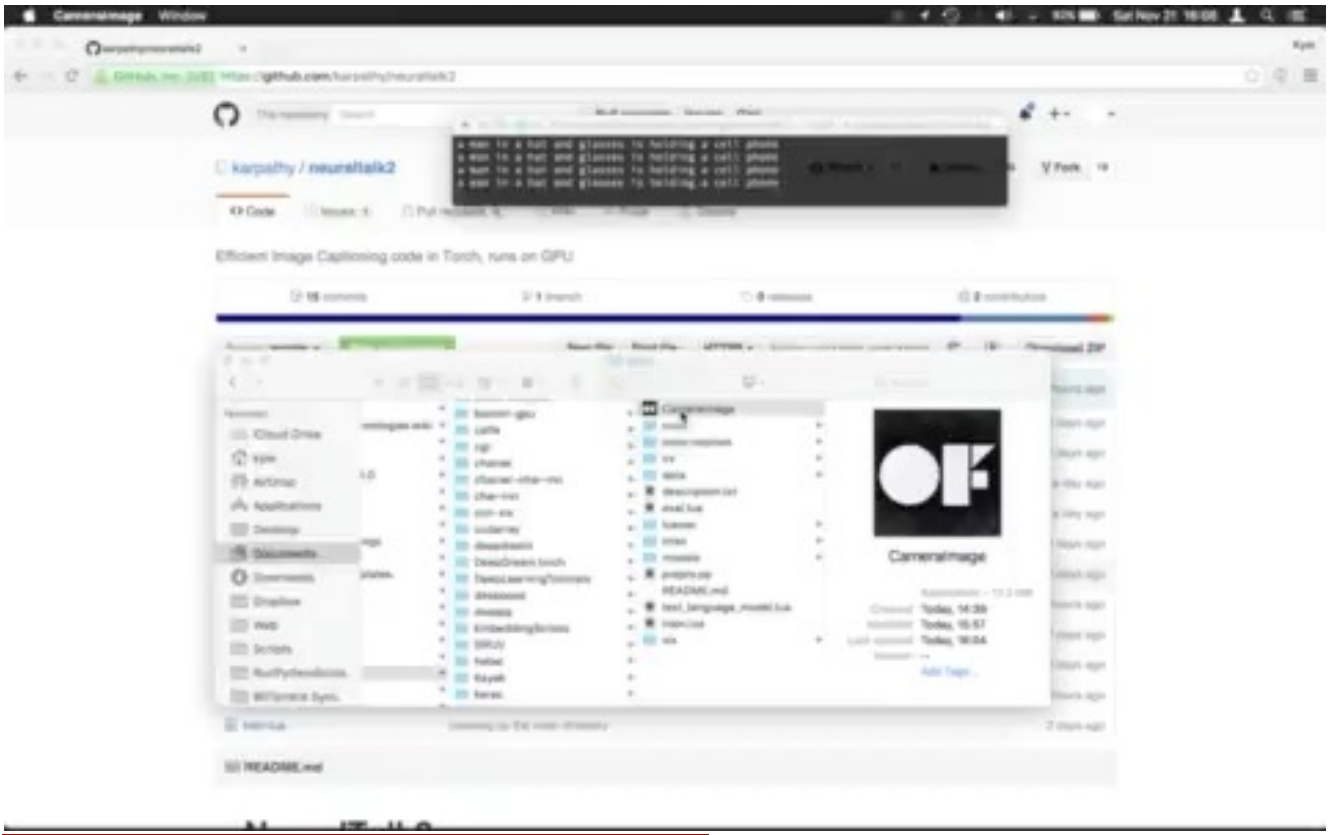
- sensor-equipped devices (e.g., drones);
- sensor networks (such as *Internet of Things*);
- development of inexpensive human-machine interaction interfaces;
- ...

Many are the applicative tasks of interest:

- monitoring and classification of human behaviour (sequence $\rightarrow$ category)
- content video captioning (image $\rightarrow$ sequence of text)
- sentiment classification (sequence of text $\rightarrow$ sentiment)
- language machine translation (sequence of text $\rightarrow$ sequence of text)
- ...

Not only classification or transduction, also generation of sequences from static data (e.g., image), or from text (e.g, ChatGPT)

Video Captioning



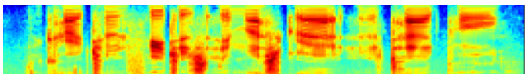
Learning in Sequential Domains

Why learning in sequential domains is different wrt static domains ?

Because successive points in sequential data are strongly correlated

Machine learning models and algorithms for sequence learning have to

- consider that data points are not independent
- deal with sequential distortions and/or variations (e.g. In speech, variations in speaking rate)



- make use of contextual information

Static Data

Learn $P(\boldsymbol{o}|\boldsymbol{x})$

- typeof$(\boldsymbol{x})$:
  - a *fixed-size tuple* of predictive attributes

    When using neural networks categorical attributes are encoded using numerical values $\Rightarrow$ a fixed-size array of real numbers

- typeof$(\boldsymbol{o})$:
  - Classification: a categorical variable
  - Regression: a numerical (multivariate) variable

Sequential Data

Learn $P(\boldsymbol{o}|\boldsymbol{x})$

- typeof($\boldsymbol{x}$): a **sequence** $\boldsymbol{x}^{(1)}, ..., \boldsymbol{x}^{(t)}, ..$ where each $\boldsymbol{x}^{(t)}$ has a static type

  Examples:

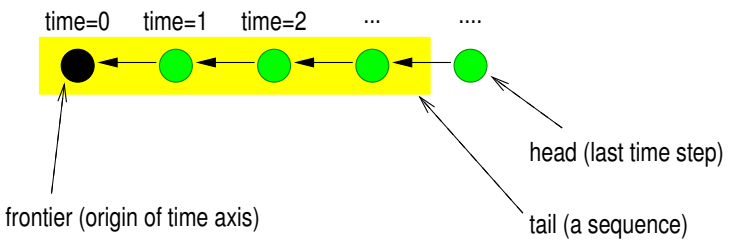
  - Speech recognition: $\boldsymbol{x}^{(t)}$ is an array of k real values representing acoustic features measured in a 10 ms time interval
  - Molecular biology: $\boldsymbol{x}^{(t)}$ is a symbol in a finite alphabet of 20 amino-acids
  - Image captioning: only one item per sequence, i.e. $\boldsymbol{x}^{(1)}$ is an image

- typeof($\boldsymbol{o}$) may be either static (e.g., sequence classification) or a sequence

Sequences

Using mathematical induction, a sequence is either

- an external vertex, or
- an ordered pair (t, h) where the *head* h is a vertex and the *tail* t is a sequence



Sequencial Transductions

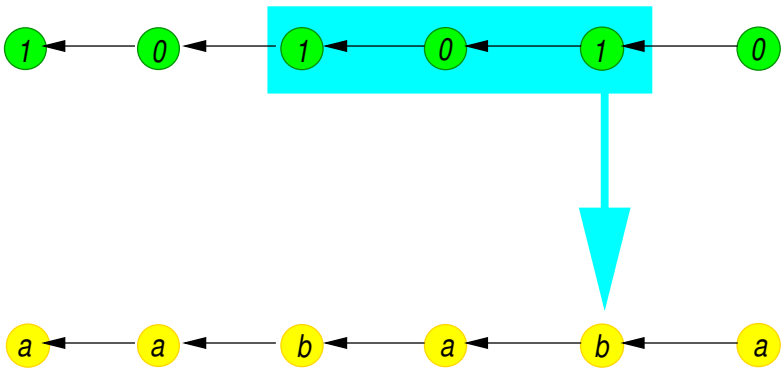
- Let $\mathcal{X}$ and $\mathcal{O}$ be the **input** and **output** label spaces
- We denote by $\mathcal{X}^*$ the set of all sequences with labels in $\mathcal{X}$
- Cf. the classical definition in language theory: $\mathcal{X} = \Sigma$ (a set of terminal symbols) and Σ^* is the free monoid over Σ
- A general transduction $\mathcal{T}$ is a subset of $\mathcal{X}^* \times \mathcal{O}^*$
- We shall limit our discussion to functions:

$$\mathcal{T} : \mathcal{X}^* \rightarrow \mathcal{O}^*$$

Algebraic and Finite-Memory Transductions

$\mathcal{T}(\cdot)$ has **finite memory** $k \in \mathbb{N}$ if

- $\forall \boldsymbol{x} \in \mathcal{X}^*$ and $\forall t$, $\mathcal{T}(\boldsymbol{x}^{(t)})$ only depends on $\{\boldsymbol{x}^{(t)}, \boldsymbol{x}^{(t-1)}, \ldots, \boldsymbol{x}^{(t-k)}\}$



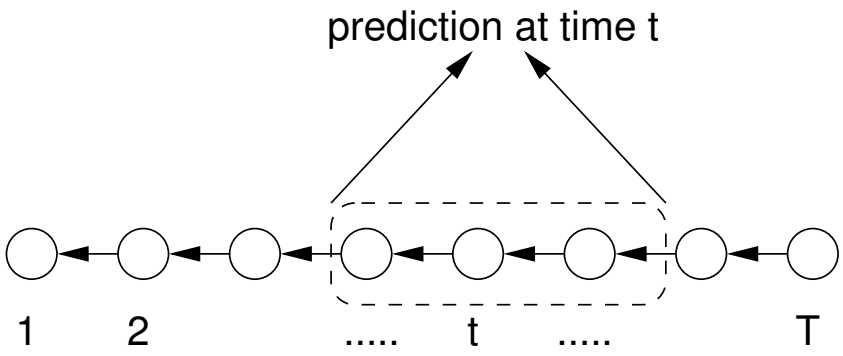
$\mathcal{T}(\cdot)$ is **algebraic** if it has 0 finite memory (i.e., no memory at all)

Learning Sequences

Sequences have variable length but typical *machine learning models* have a fixed number of inputs

Solutions:

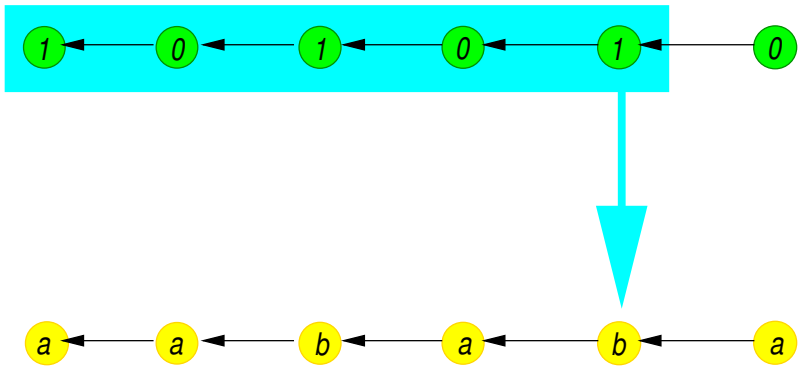
- Limit context to a fixed-size window



- Use *recurrent* models (causality)
- Use *Transformers* for non-causal sequences (e.g., text)

Causality

A transduction $\mathcal{T}(\cdot)$ is **causal** if the output at time t does not depend on **future** inputs (at time $t+1, t+2, \ldots$)



Recursive State Representation

Intuition:

outputs depend on **hidden state variables (label space $\mathcal{H}$)**

For each time t:

$$\begin{aligned} \boldsymbol{h}^{(t)} &= f(\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}, t) \\ \boldsymbol{o}^{(t)} &= g(\boldsymbol{h}^{(t)}, \boldsymbol{x}^{(t)}, t) \end{aligned}$$

where

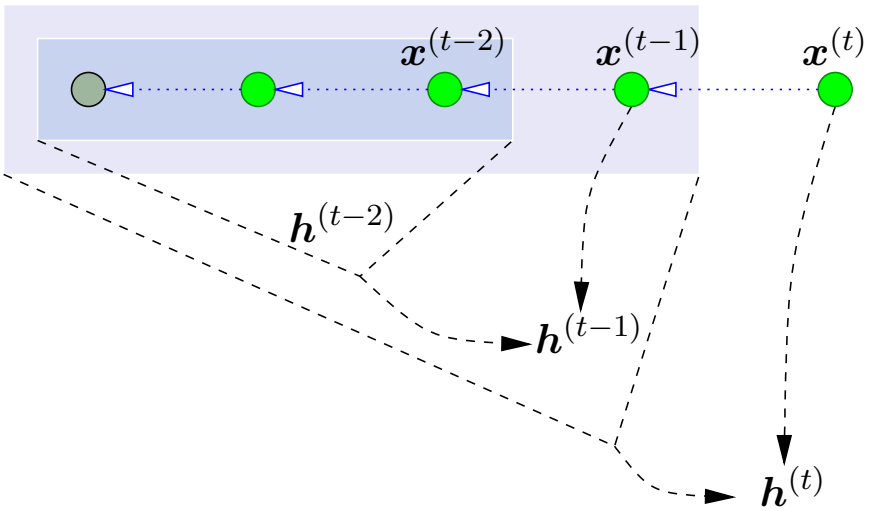
$f : \mathcal{H} \times \mathcal{X} \to \mathcal{H}$ *state transition function*

$g : \mathcal{H} \times \mathcal{X} \to \mathcal{O}$ *output function*

A recursive state representation exists only if $\mathcal{T}(\cdot)$ is *causal*

$\mathcal{T}(\cdot)$ is **stationary** if $f(\cdot)$ and $g(\cdot)$ do not depend on t

Recursive State Update



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

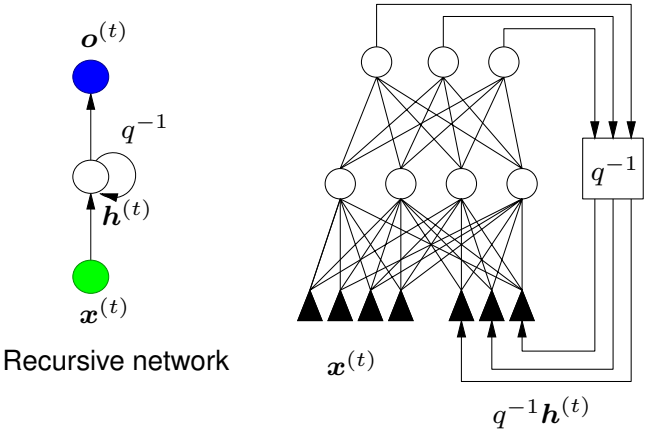
for $t = 0$, $\boldsymbol{h} = \boldsymbol{h}^{(0)}$ is the **initial state**

The initial state is associated with the external vertex (frontier)

Graphical Description

Graphical representation of the state transition function using the time shift operator q^{-1}:

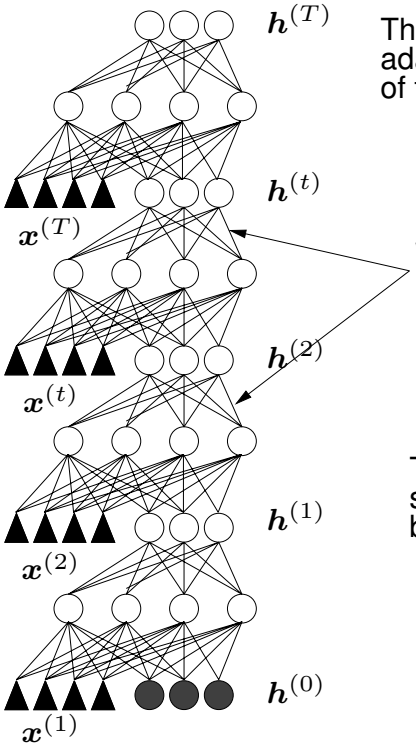
$$\boldsymbol{h}^{(t)} = f(\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)})$$



Recursive network

$q^{-1}\boldsymbol{h}^{(t)} = \boldsymbol{h}^{(t-1)}$ (unitary time delay)

Time Unfolding



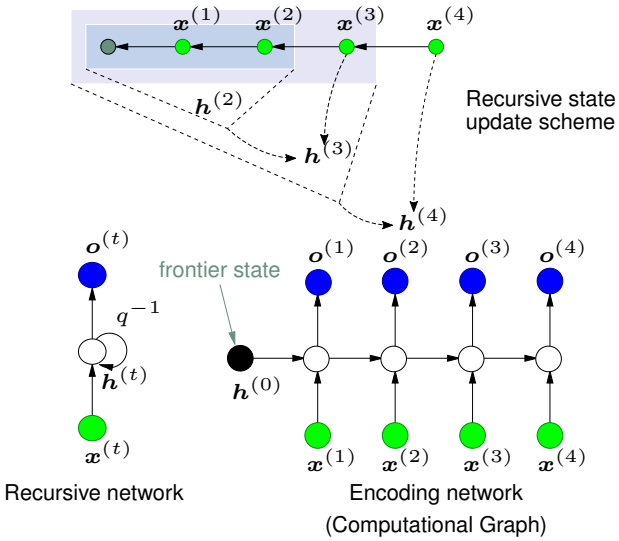
The last state is an adaptive encoding of the whole sequence

Weights are shared (replicated)

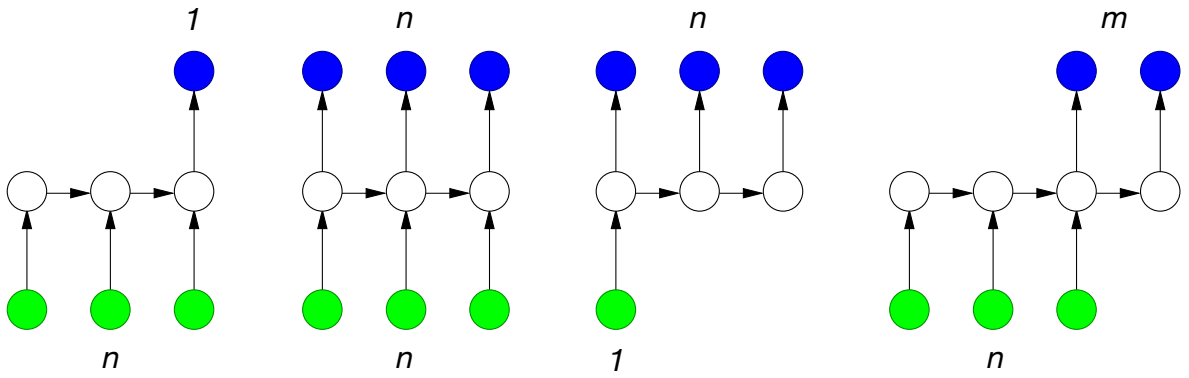
The unfolded network has a feedforward structure: e.g., gradient can be computed by Backpropagation (through time)

Encoding Networks
(Computational Graph)

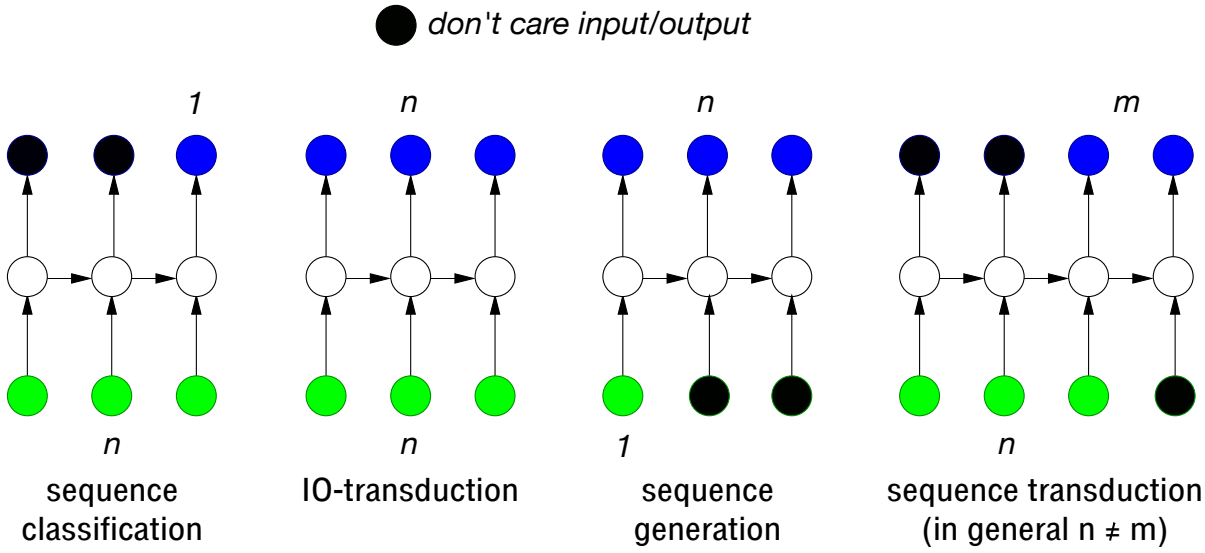
Given a sequence $s \in \mathcal{X}^*$ and a recursive transduction $\mathcal{T}$, the *encoding network* associated to s and $\mathcal{T}$ is formed by unrolling (*time unfolding*) the recursive network of $\mathcal{T}$ through the input sequence s



Examples of Sequential Transductions



A Unifying Perspective as IO-transductions



Recursive State Representation

Intuition:

outputs depend on **hidden state variables (label space $\mathcal{H}$)**

For each time t:

$$\begin{aligned} \boldsymbol{h}^{(t)} &= f(\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}, t) \\ \boldsymbol{o}^{(t)} &= g(\boldsymbol{h}^{(t)}, \boldsymbol{x}^{(t)}, t) \end{aligned}$$

where

$$\begin{aligned} f &: \mathcal{H} \times \mathcal{X} \rightarrow \mathcal{H} \quad \textit{state transition function} \\ g &: \mathcal{H} \times \mathcal{X} \rightarrow \mathcal{O} \quad \textit{output function} \end{aligned}$$

A recursive state representation exists only if $\mathcal{T}(\cdot)$ is *causal*

$\mathcal{T}(\cdot)$ is **stationary** if $f(\cdot)$ and $g(\cdot)$ do not depend on t

Model Types

How to implement $f(\cdot)$ and $g(\cdot)$?

Two general families of models:

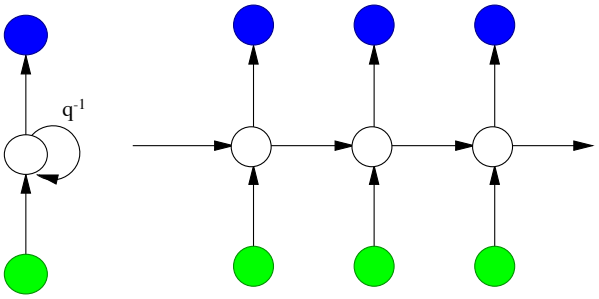
- Linear
  - Kalman Filter
  - Hidden Markov Models
  - Linear Dynamical Systems
    - Linear Autoencoders for Sequences
  - . . .
- Nonlinear
  - Recurrent Neural Networks
  - . . .

Shallow Recurrent Neural Networks

Non-linear Dynamical System

$$\boldsymbol{h}^{(t)} = f(\boldsymbol{U}\boldsymbol{x}^{(t)} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{b}),$$
$$\boldsymbol{o}^{(t)} = g(\boldsymbol{V}\boldsymbol{h}^{(t)} + \boldsymbol{c}),$$

where $f()$ and $g()$ are non-linear functions (e.g. $tanh()$), and $\boldsymbol{h}^{(0)} = \mathbf{0}$ (or can be learned jointly with the other parameters)



Shallow Recurrent Neural Networks

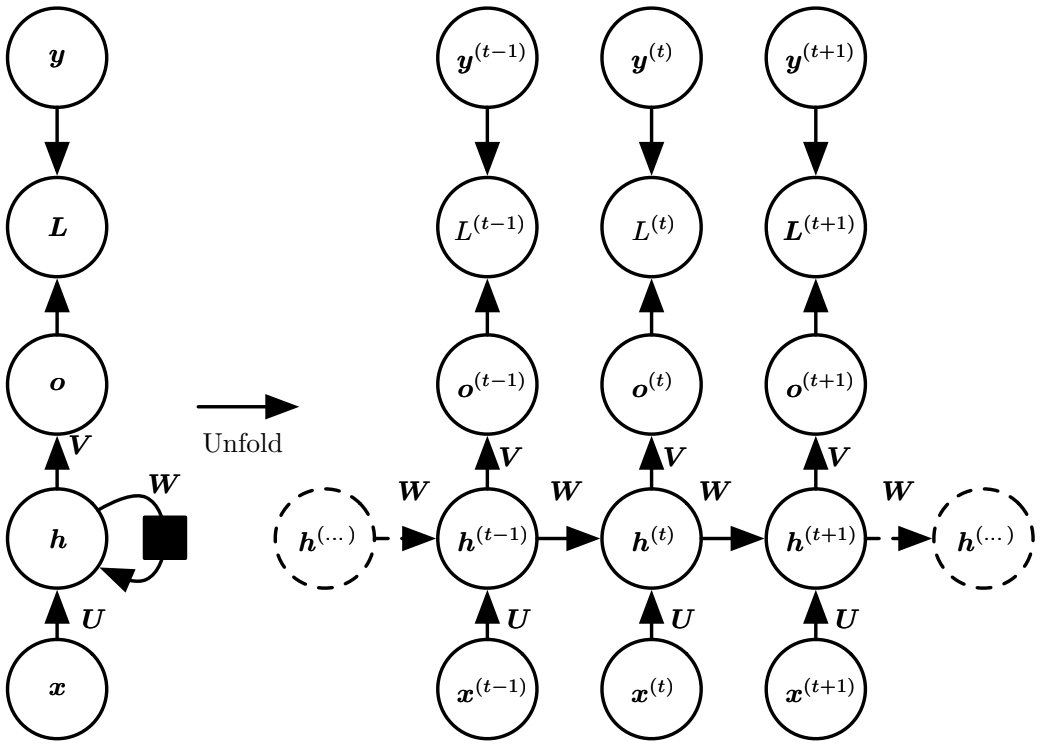
Example: IO-transduction RNN with discrete outputs

$$
\begin{aligned}
\boldsymbol{h}^{(t)} &= tanh(\boldsymbol{U}\boldsymbol{x}^{(t)} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{b}), \\
\boldsymbol{o}^{(t)} &= \boldsymbol{V}\boldsymbol{h}^{(t)} + \boldsymbol{c}, \\
\hat{\boldsymbol{y}}^{(t)} &= softmax(\boldsymbol{o}^{(t)}), \\
\text{(Loss) } L = \sum_t L^{(t)} &= -\sum_t \log p_{\text{model}} \left(y^{(t)} | \{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(t)}\} \right)
\end{aligned}
$$

where

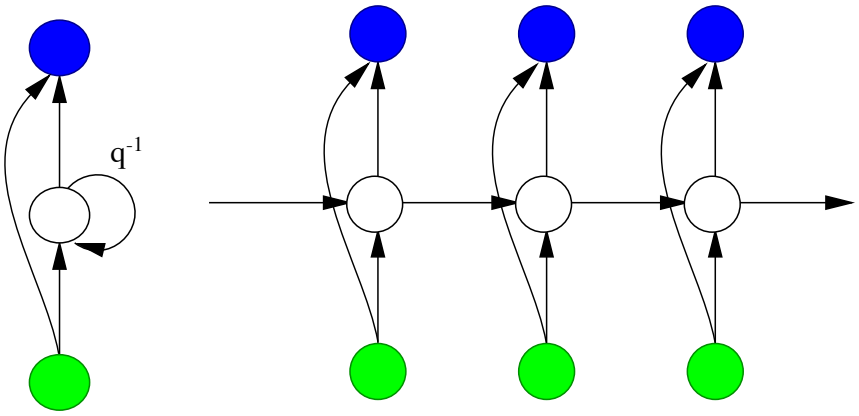
- $\boldsymbol{o}^{(t)}$ is the unnormalized log probabilities a time t
- $\boldsymbol{y}^{(t)}$ is the target vector a time t
- $p_{\text{model}} \left(y^{(t)} | \{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(t)}\} \right)$ is given by reading the entry for $y^{(t)}$ from the model's output vector $\hat{\boldsymbol{y}}^{(t)}$, i.e. $\hat{\boldsymbol{y}}^{(t)}$ is computed internally to L

Recurrent and Computational Graph



Some Additional Architectural Features for RNN

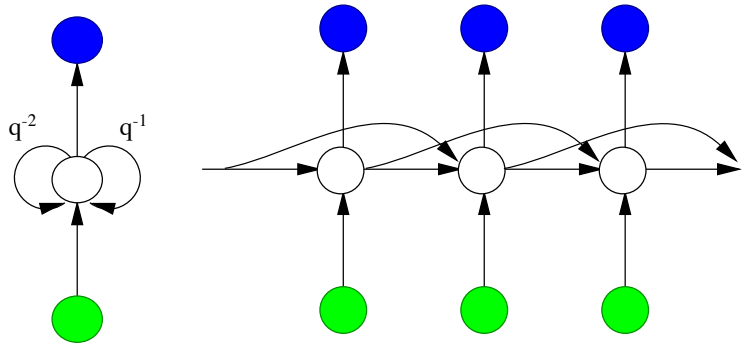
short-cut connections



$$\boldsymbol{o}^{(t)} = g(\boldsymbol{V}^h \boldsymbol{h}^{(t)} + \boldsymbol{V}^x \boldsymbol{x}^{(t)} + \boldsymbol{b})$$

Some Additional Architectural Features for RNN

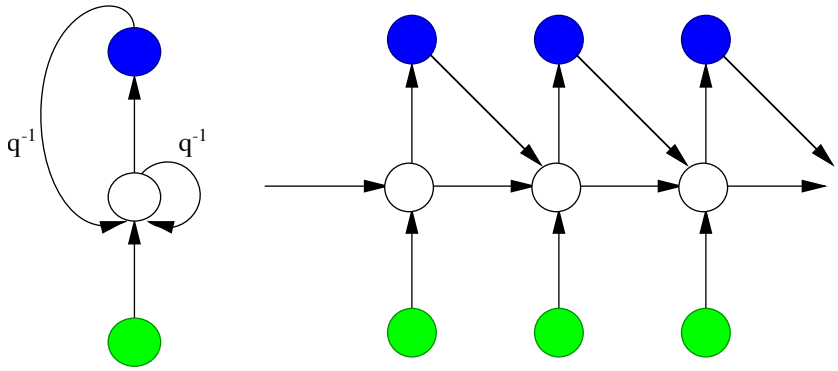
higher-order states (e.g., 2*nd* order)



$$\boldsymbol{h}^{(t)} = f(\boldsymbol{U}\boldsymbol{x}^{(t)} + \boldsymbol{W}^{(1)}\boldsymbol{h}^{(t-1)} + \boldsymbol{W}^{(2)}\boldsymbol{h}^{(t-2)} + \boldsymbol{c})$$

Some Additional Architectural Features for RNN

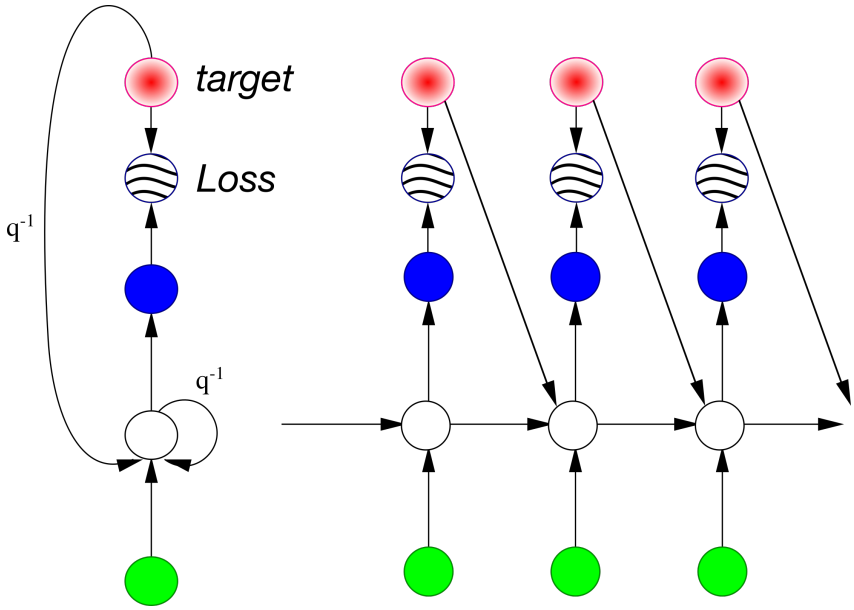
feedback from output (context)



$$\boldsymbol{h}^{(t)} = f(\boldsymbol{U}\boldsymbol{x}^{(t)} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{Z}\boldsymbol{o}^{(t-1)})$$

Some Additional Architectural Features for RNN

feedback connections allow to *force* the target signal: teacher forcing



Some Additional Architectural Features for RNN

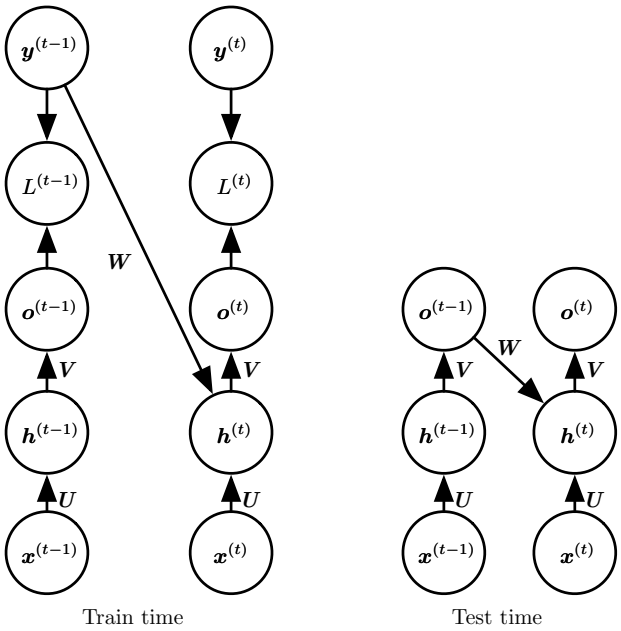
Example of Teacher Forcing

$$
\begin{aligned}
\boldsymbol{h}^{(t)} &= tanh(\boldsymbol{U}\boldsymbol{x}^{(t)} + \boldsymbol{W}\boldsymbol{o}^{(t-1)} + \boldsymbol{b}), \\
\boldsymbol{o}^{(t)} &= \boldsymbol{V}\boldsymbol{h}^{(t)} + \boldsymbol{c}, \\
\hat{\boldsymbol{y}}^{(t)} &= softmax(\boldsymbol{o}^{(t)}), \\
\text{(Loss) } L &= -\sum_t \log p_{\text{model}} \left(\boldsymbol{y}^{(t)} | \boldsymbol{y}^{(1)}, \ldots, \boldsymbol{y}^{(t-1)}, \boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(t)} \right)
\end{aligned}
$$

e.g., for two steps the conditional maximum likelihood criterion is:

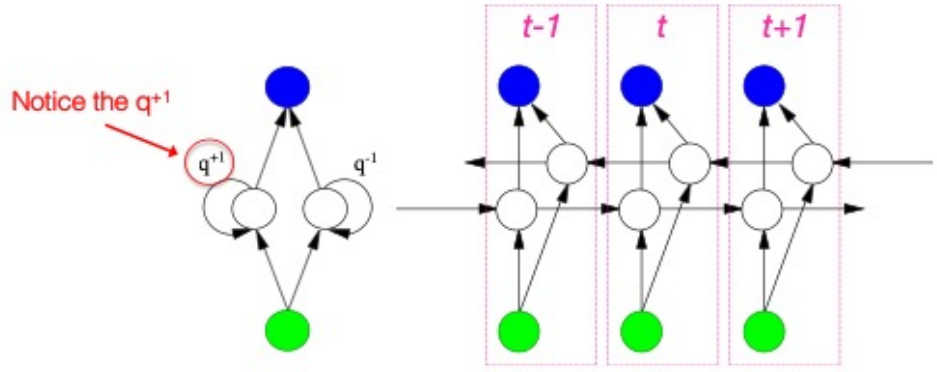
$$
\begin{aligned}
&\log p_{\text{model}} \left(\boldsymbol{y}^{(1)}, \boldsymbol{y}^{(2)} | \boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)} \right) \\
&= \log p_{\text{model}} \left(\boldsymbol{y}^{(2)} | \boldsymbol{y}^{(1)}, \boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)} \right) + \log p_{\text{model}} \left(\boldsymbol{y}^{(1)} | \boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)} \right)
\end{aligned}
$$

Example of Teacher Forcing



Some Additional Architectural Features for RNN

Bidirectional RNN: when the sequence is not temporal (or off-line processing)



$$\boldsymbol{o}^{(t)} = g(\boldsymbol{V}^{(p)}\mathbf{h}_p^{(t)} + \boldsymbol{V}^{(f)}\mathbf{h}_f^{(t)})$$

$$\underbrace{\mathbf{h}_p^{(t)} = f(\boldsymbol{U}^{(p)}\boldsymbol{x}^{(t)} + \boldsymbol{W}^{(p)}\boldsymbol{h}_p^{(t-1)})}_{past\ history} \qquad \underbrace{\mathbf{h}_f^{(t)} = f(\boldsymbol{U}^{(f)}\boldsymbol{x}^{(t)} + \boldsymbol{W}^{(f)}\boldsymbol{h}_f^{(t+1)})}_{future\ history}$$

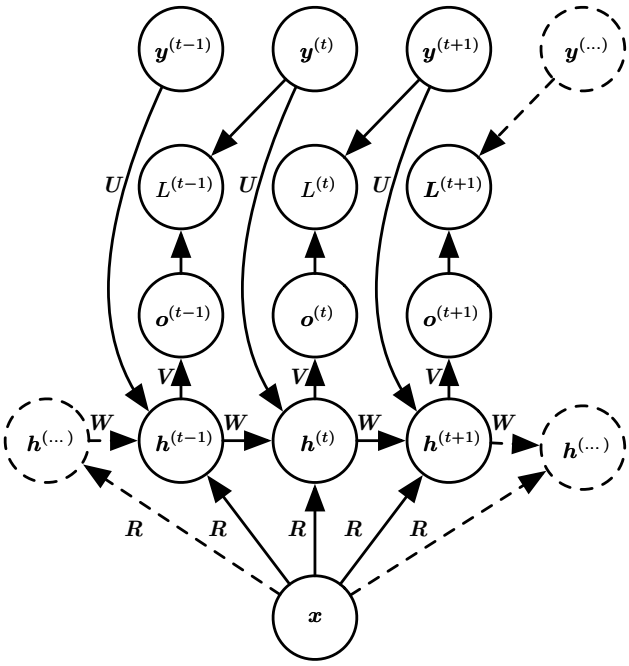
e.g. DNA sequence in bioinformatics

Some Additional Architectural Features for RNN

- All these architectural features (and others...) are *orthogonal*, i.e. they can be combined together
- We will see examples of such combinations when looking at some recent architectures for machine translation and image captioning (can you figure out which kind of *transductions* they call for ?)
- For these applications there are typical architectural *building blocks* that are worth discussing right now:
  - generation of a sequence from a vector (1 to n transduction)
  - sequence to sequence transduction via Encoder-Decoder (n to m transduction)

Some Building Blocks for RNN

Generation of a sequence from a vector



Some Building Blocks for RNN

Encoder-Decoder