Unfolding Computational Graphs

- A Computational Graph is a way to formalize the structure of a set of computations
    - Such as mapping inputs and parameters to outputs and loss
- We can unfold a recursive or recurrent computation into a computational graph that has a repetitive structure
    - Corresponding to a chain of events
- Unfolding this graph results in sharing of parameters across a deep network structure

Example of unfolding a recurrent equation

- Classical form of a dynamical system is

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

  - where $\boldsymbol{s}^{(t)}$ is called the state of the system
  - Equation is recurrent because the definition of $\boldsymbol{s}$ at time t refers back to the same definition at time t-1
- For a finite no. of time steps τ, the graph can be unfolded by applying the definition τ-1 times
  - E,g, for τ = 3 time steps we get

$$\boldsymbol{s}^{(3)}= f\,(\boldsymbol{s}^{(2)};\ \boldsymbol{\theta})$$
$$= f\,(f\,(\boldsymbol{s}^{(1)};\ \boldsymbol{\theta});\ \boldsymbol{\theta})$$

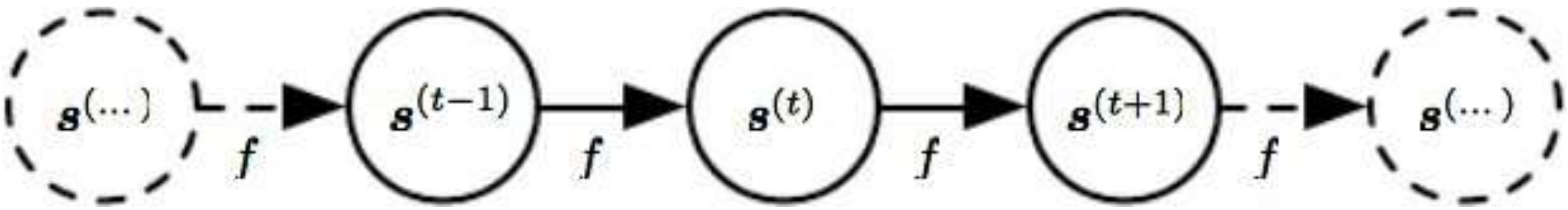
- Unfolding equation by repeatedly applying the definition in this way has yielded expression without recurrence
  - $\boldsymbol{s}^{(1)}$ is ground state and $\boldsymbol{s}^{(2)}$ computed by applying f
- Such an expression can be represented by a traditional acyclic computational graph (as shown next)

Unfolded dynamical system

- The classical dynamical system described by

$$\boldsymbol{s}^{(t)}=f\,(\boldsymbol{s}^{(t\text{-}1)};\ \boldsymbol{\theta}\ )\ \text{and}\ \boldsymbol{s}^{(3)}=f\,(f\,(\boldsymbol{s}^{(1)};\ \boldsymbol{\theta});\ \boldsymbol{\theta})$$

  is illustrated as an unfolded computational graph



- Each node represents state at some time t
- Function f maps state at time t to the state at t+1
- The same parameters (the same value of $\boldsymbol{\theta}$ used to parameterize f) are used for all time steps

Dynamical system driven by external signal

- As another example, consider a dynamical system driven by external (input) signal $\boldsymbol{x}^{(t)}$

$$\boldsymbol{s}^{(t)}=f\left(\boldsymbol{s}^{(t\text{-}1)}, \boldsymbol{x}^{(t)}; \boldsymbol{\theta}\right)$$

  - State now contains information about the whole past input sequence
  - Note that the previous dynamic system was simply $\boldsymbol{s}^{(t)}=f\left(\boldsymbol{s}^{(t\text{-}1)}; \boldsymbol{\theta}\right)$

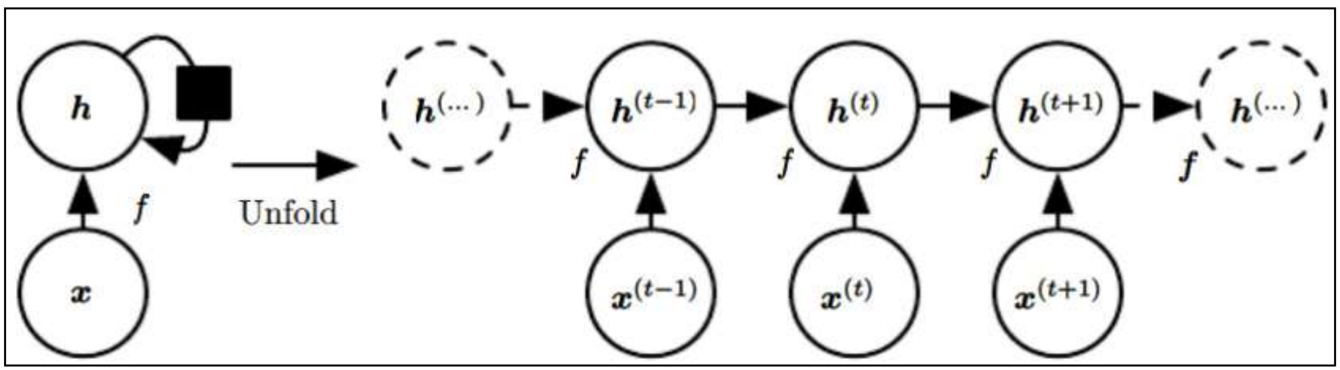
- Recurrent neural networks can be built in many ways
  - Much as almost any function is a feedforward neural network, any function involving recurrence can be considered to be a recurrent neural network

Defining values of hidden units in RNNs

- Many recurrent neural nets use same equation (as dynamical system with external input) to define values of hidden units
  - To indicate that the state is hidden rewrite using variable ***h*** for state:

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

  - Illustrated below:



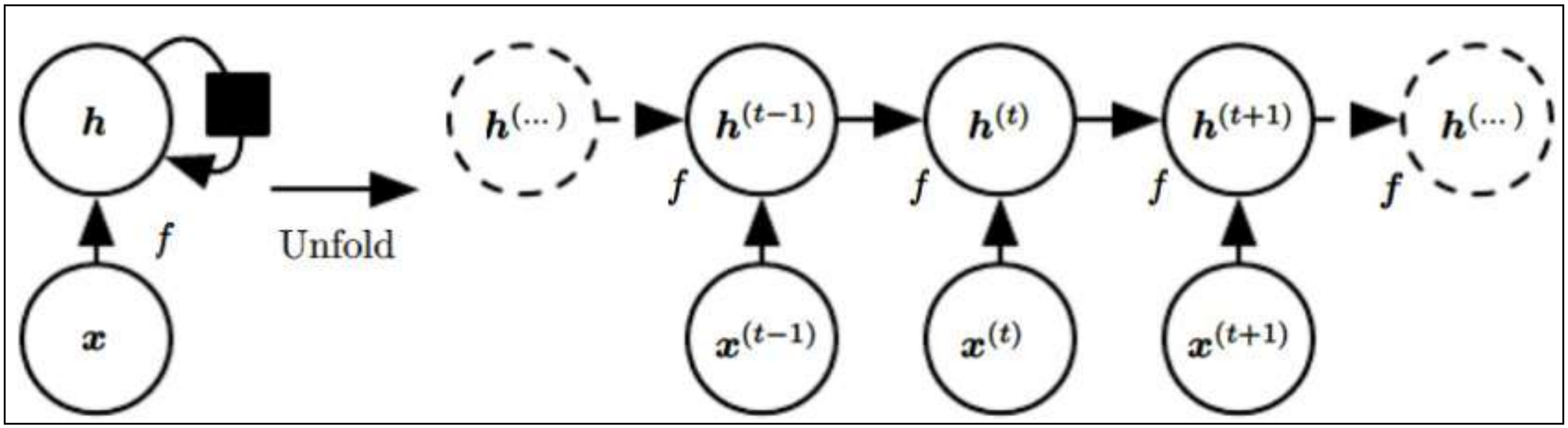
- Typical RNNs have extra architectural features such as output layers that read information out of state $\boldsymbol{h}$ to make predictions

A recurrent network with no outputs

- This network just processes information from input $\boldsymbol{x}$ by incorporating it into state $\boldsymbol{h}$ that is passed forward through time

Circuit diagram:
Black square indicates
Delay of one time step

Unfolded computational graph:
each node is now
associated with one time instance



Typical RNNs will add extra architectural features such as output layers to read information out of the state $\boldsymbol{h}$ to make predictions

Predicting the Future from the Past

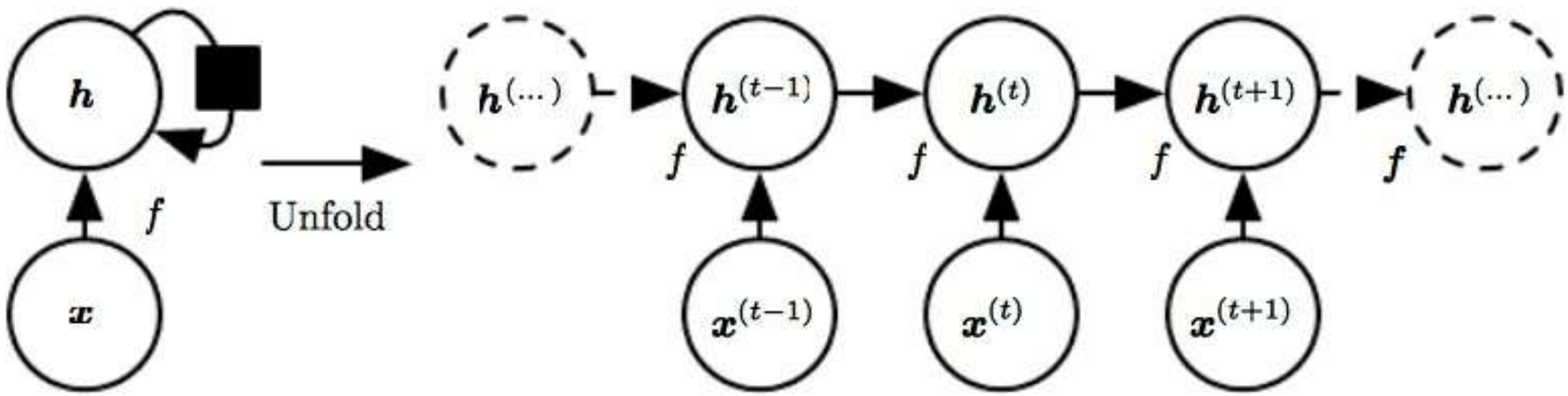
- When RNN is required to perform a task of predicting the future from the past, network typically learns to use $\boldsymbol{h}^{(t)}$ as a lossy summary of the task-relevant aspects of the past sequence of inputs upto t
- The summary is in general lossy since it maps a sequence of arbitrary length $(\boldsymbol{x}^{(t)}, \boldsymbol{x}^{(t\text{-}1)},..,\boldsymbol{x}^{(2)},\boldsymbol{x}^{(1)})$ to a fixed length vector $\boldsymbol{h}^{(t)}$

Information Contained in Summary

- Depending on criterion, summary keeps some aspects of past sequence more precisely than other aspects
- Examples:
  - RNN used in statistical language modeling, typically to predict next word from past words
    - it may not be necessary to store all information upto time t but only enough information to predict rest of sentence
  - Most demanding situation: we ask $\boldsymbol{h}^{(t)}$ to be rich enough to allow one to approximately recover the input sequence as in autoencoders

Unfolding: from circuit diagram to computational graph

- Equation $\boldsymbol{h}^{(t)}=f(\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}; \boldsymbol{\theta})$ can be written in two different ways: circuit diagram or an unfolded computational graph



- Unfolding is the operation that maps a circuit to a computational graph with repeated pieces
- The unfolded graph has a size dependent on the sequence length

Process of Unfolding

- We can represent unfolded recurrence after t steps with a function $g^{(t)}$:

$$\boldsymbol{h}^{(t)} = g^{(t)}\left(\boldsymbol{x}^{(t)}, \boldsymbol{x}^{(t-1)},..,\boldsymbol{x}^{(2)},\boldsymbol{x}^{(1)}\right)$$
$$= f\left(\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}; \boldsymbol{\theta}\right)$$

- Function $g^{(t)}$ takes in whole past sequence $\left(\boldsymbol{x}^{(t)}, \boldsymbol{x}^{(t-1)},..,\boldsymbol{x}^{(2)},\boldsymbol{x}^{(1)}\right)$ as input and produces the current state but the unfolded recurrent structure allows us to factorize $g^{(t)}$ into repeated application of a function f
- The unfolding process introduces two major advantages as discussed next.

Unfolding process allows learning a single model

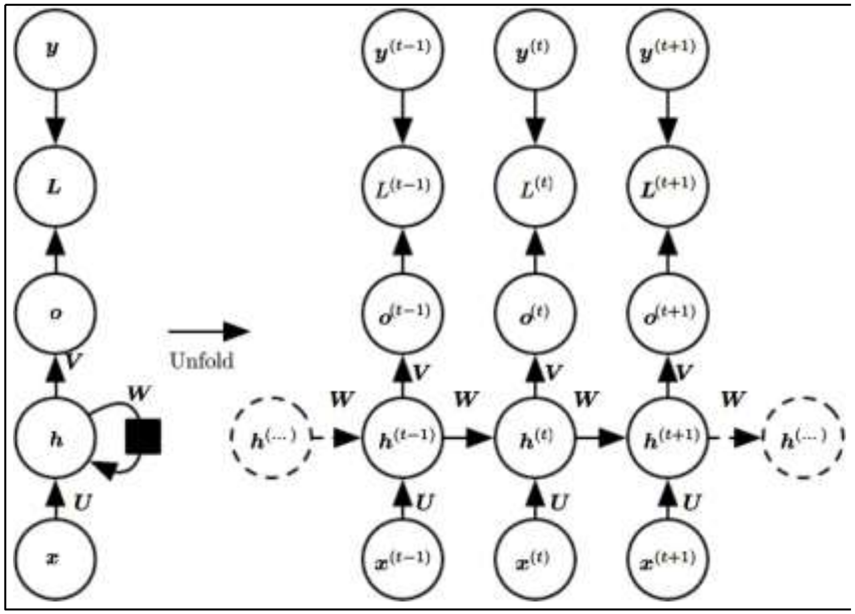
- The unfolding process introduces two major advantages:
  1. Regardless of sequence length, learned model has same input size
     - because it is specified in terms of transition from one state to another state rather than specified in terms of a variable length history of states
  2. Possible to use same function f with same parameters at every step
- These two factors make it possible to learn a single model f
  - that operates on all time steps and all sequence lengths
  - rather than needing separate model $g^{(t)}$ for all possible time steps
- Learning a single shared model allows:
  - Generalization to sequence lengths that did not appear in the training
  - Allows model to be estimated with far fewer training examples than would be required without parameter sharing

Both recurrent graph and unrolled graph are useful

- Recurrent graph is succinct
- Unrolled graph provides explicit description of what computations to perform
  - Helps illustrate the idea of information flow forward in time
    - Computing outputs and losses
  - And backwards in time
    - Computing gradients
  - By explicitly showing path along which information flows

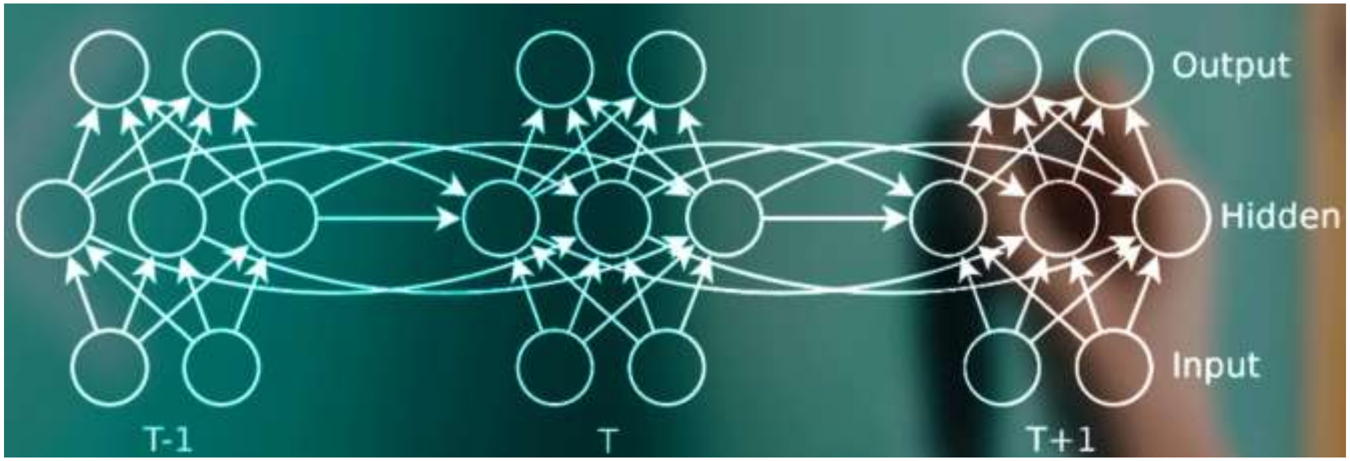
Unfolding Computational Graphs

- A Computational Graph is a way to formalize the structure of a set of computations
  - Such as mapping inputs and parameters to outputs and loss
- We can unfold a recursive or recurrent computation into a computational graph that has a repetitive structure
  - Corresponding to a chain of events
- Unfolding this graph results in sharing of parameters across a deep network structure



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

A more complex unfolded computational graph



Source: Indico corporation

Two units in input layer, instead of 1
Three units in hidden layer, instead of 1
Two units in output layer, instead of 1

Previous one: