CS60010: Deep Learning

Recurrent Neural Network

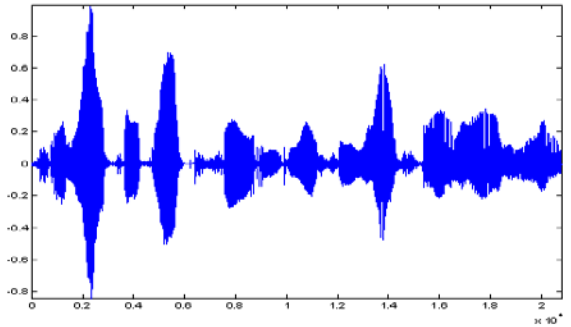
Sudeshna Sarkar

Spring 2018

5 Feb 2018

Sequence

- Sequence data: sentences, speech, stock market, signal data
  - Sequence of words in an English sentence
  - Acoustic features at successive time frames in speech recognition
  - Successive frames in video classification
  - Rainfall measurements on successive days in Hong Kong
  - Daily values of current exchange rate



Modeling Sequential Data

- Sample data sequences from a certain distribution

$$P(x_1, x_2, \dots, x_T)$$

- Generate natural sentences to describe an image

$$P(y_1, y_2, \dots, y_T | I)$$

- Activity recognition from a video sequence

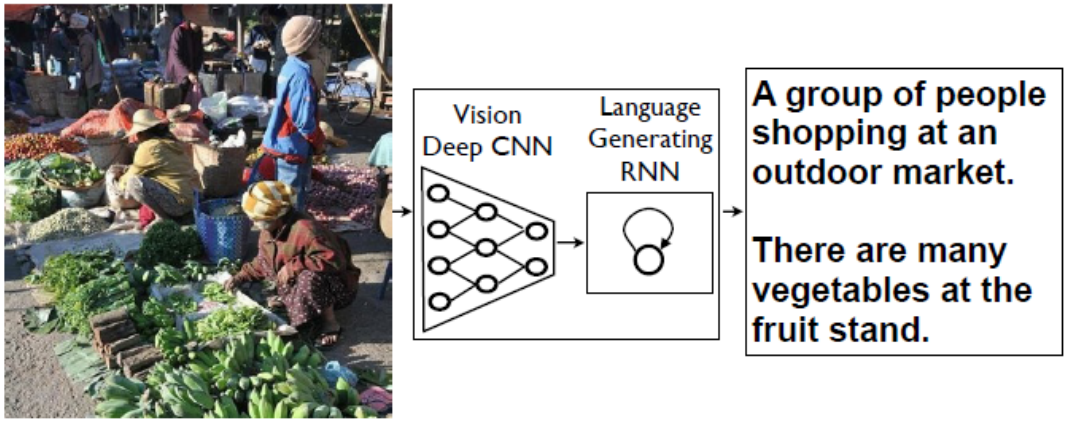
$$P(y | x_1, x_2, \dots, x_T)$$

- Speech Recognition

$$P(y_1, y_2, \dots, y_T | x_1, x_2, \dots, x_T)$$

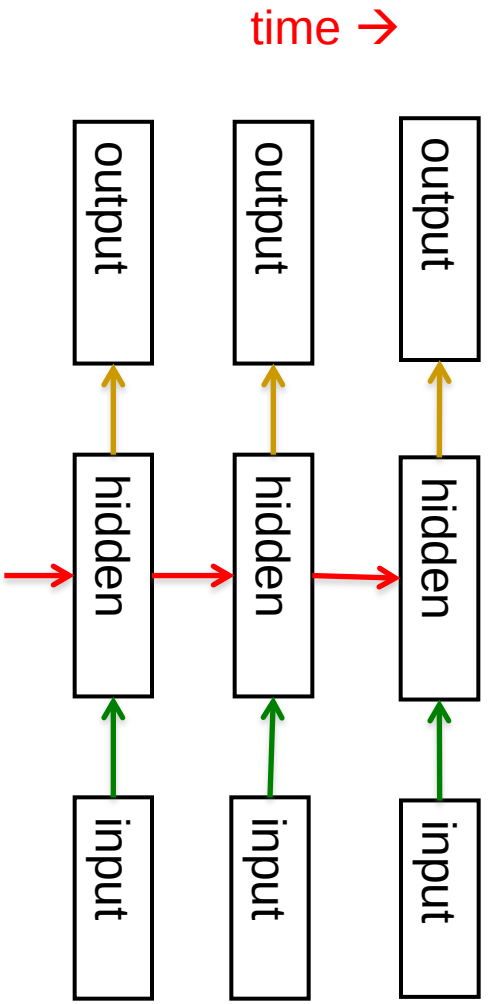
- Machine Translation

$$P(y_1, y_2, \dots, y_T | x_1, x_2, \dots, x_S)$$

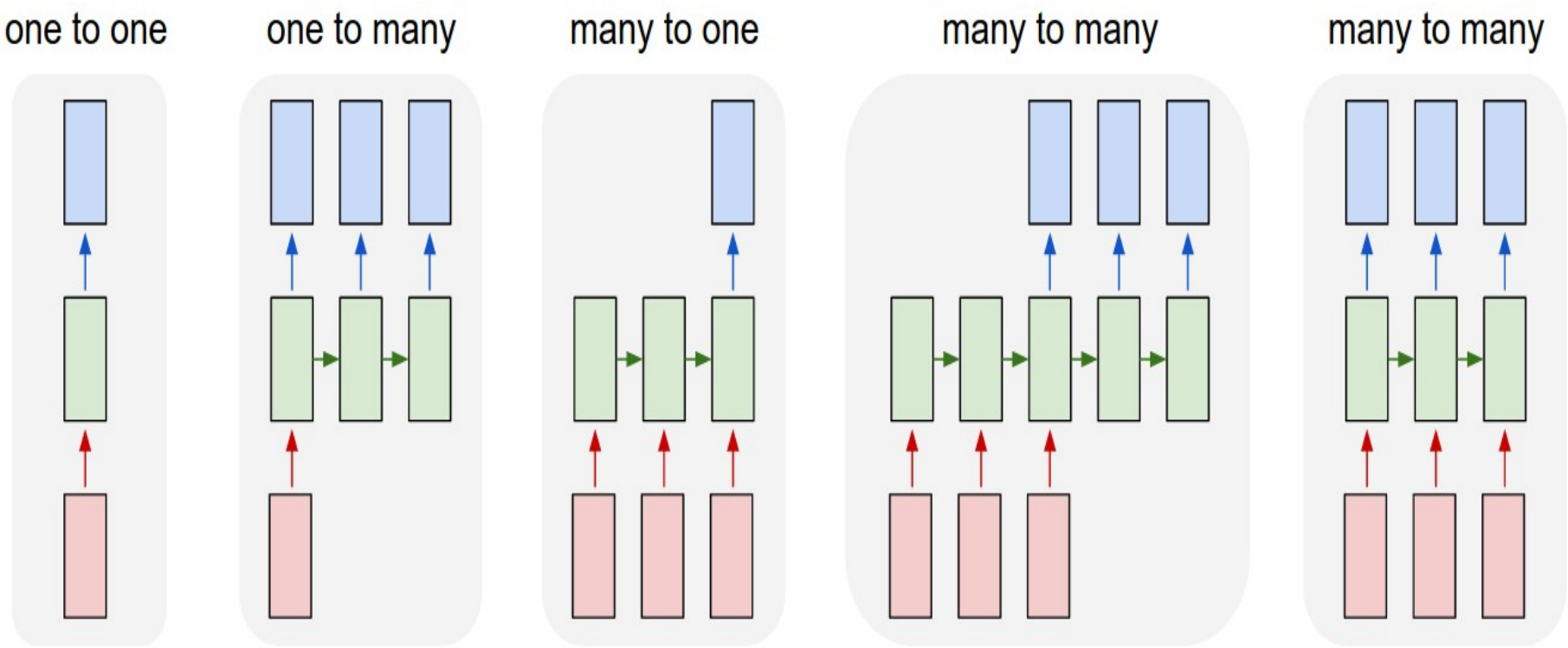


Recurrent neural networks

- RNNs are very powerful, because they combine two properties:
  - Distributed hidden state that allows them to store a lot of information about the past efficiently.
  - Non-linear dynamics that allows them to update their hidden state in complicated ways.
- With enough neurons and time, RNNs can compute anything that can be computed by your computer.



Recurrent Networks offer a lot of flexibility:



Vanilla Neural Networks

Why RNNs?

- Can model sequences having variable length
- Efficient: Weights shared across time-steps

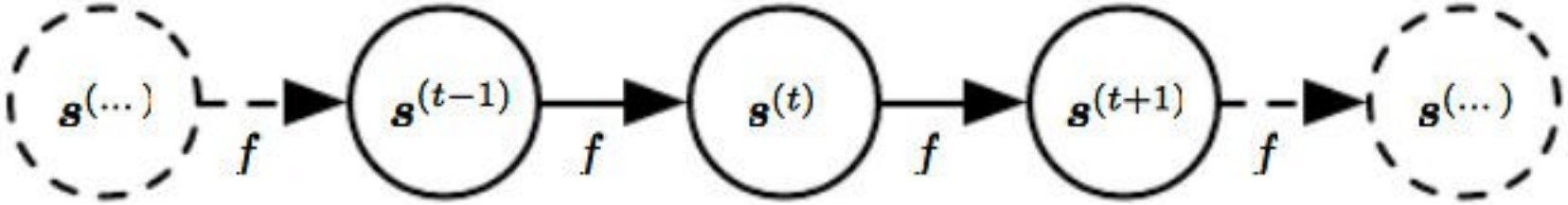
Dynamic system; Unfolded; Computation graph

- Dynamical system: classical form

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

$$s^{(3)} = f\left(f\left(s^{(1)}; \theta\right)\right)$$

- For a finite no. of time steps τ, the graph can be unfolded by applying the definition τ-1 times.

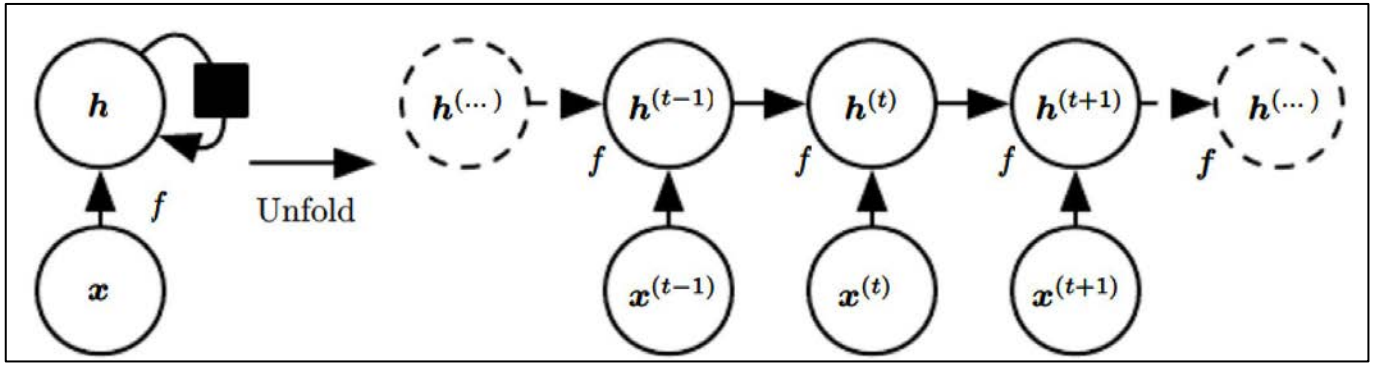


- The same parameters are used at all time steps.

Dynamical system driven by external signal

- Consider a dynamical system driven by external (input) signal $x^{(t)}$: $s^{(t)} = f\left(s^{(t-1)}, x^{(t)}; \theta\right)$
  - The state now contains information about the whole past input sequence. To indicate that the state is hidden rewrite using variable h for state:

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



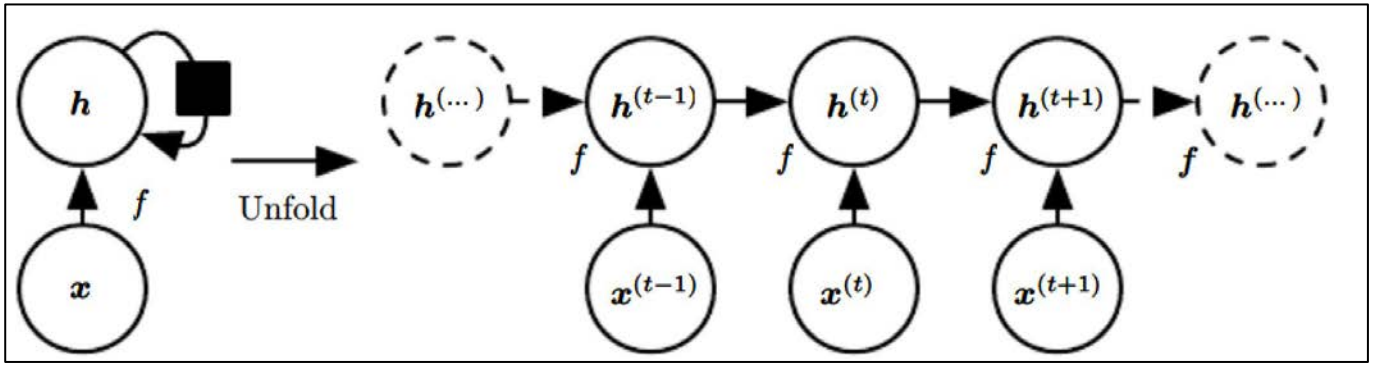
Output prediction by RNN

- Task : To predict the future from the past
- The network typically learns to use $\boldsymbol{h}^{(t)}$ as a 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 $(\boldsymbol{x}^{(t)}, \boldsymbol{x}^{(t-1)},..,\boldsymbol{x}^{(2)},\boldsymbol{x}^{(1)})$ to a fixed length vector $\boldsymbol{h}^{(t)}$
- Depending on the training criterion, summary keeps some aspects of past sequence more precisely than other aspects

Unfolding: from circuit diagram to computational graph

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

can be written in two different ways: circuit diagram or an unfolded computational graph



The unfolded graph has a size dependent on the sequence length.

Unfolding

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

$$h^{(t)} = g^{(t)}\left(x^{(t)}, x^{(t-1)} \ldots, x^{(1)}\right)$$

- The function $g^{(t)}$ takes in whole past sequence $\left(x^{(t)}, x^{(t-1)} \ldots, x^{(1)}\right)$ 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.

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

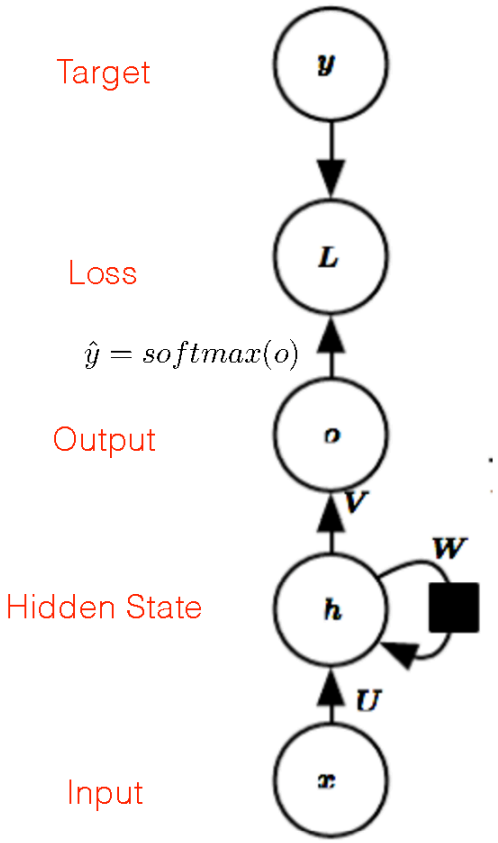
Advantages of unfolding model

1. Regardless of sequence length, the learned model has the 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. It is 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.
- Learning a single shared model: can apply the network to input sequences of different lengths and predict sequences of different lengths and to work with fewer training examples.

Recurrent Hidden Units

Basic Form



For Every Time Step!!

Recurrent Hidden Units

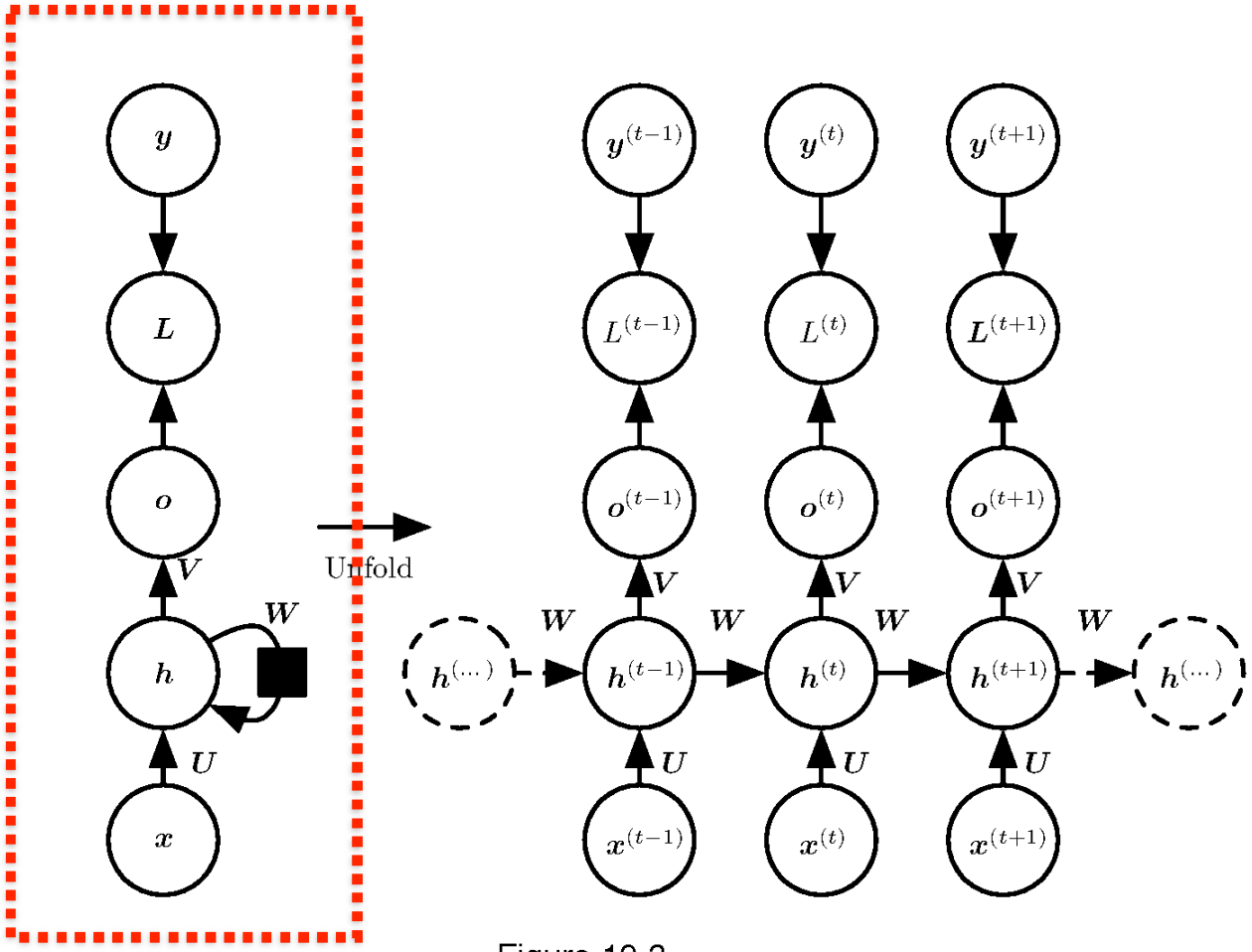
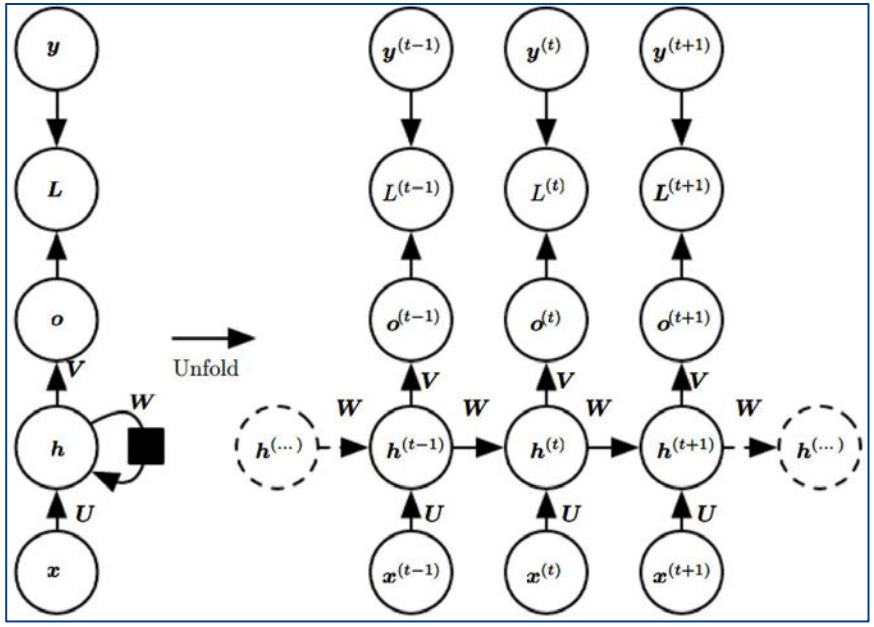


Figure 10.3

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

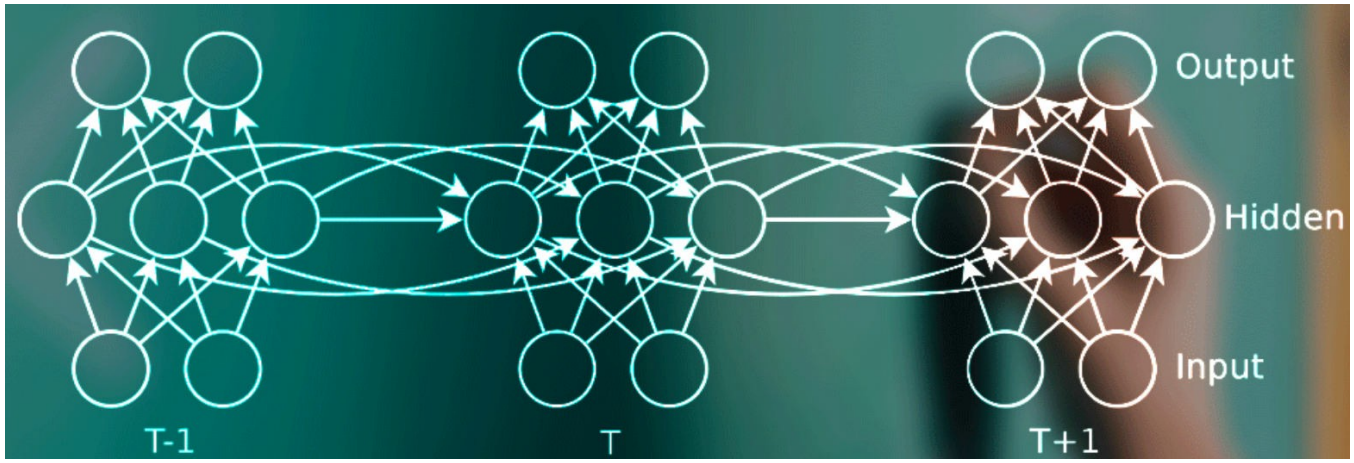


$$\boldsymbol{o}^{(t)} = \boldsymbol{c} + V\boldsymbol{h}^{(t)}$$

$$\boldsymbol{h}^{(t)} = \tanh(\boldsymbol{a}^{(t)})$$

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

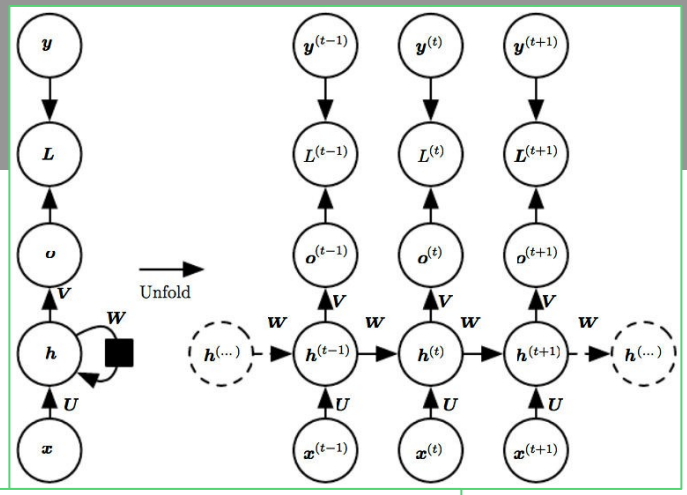
A more complex unfolded computational graph

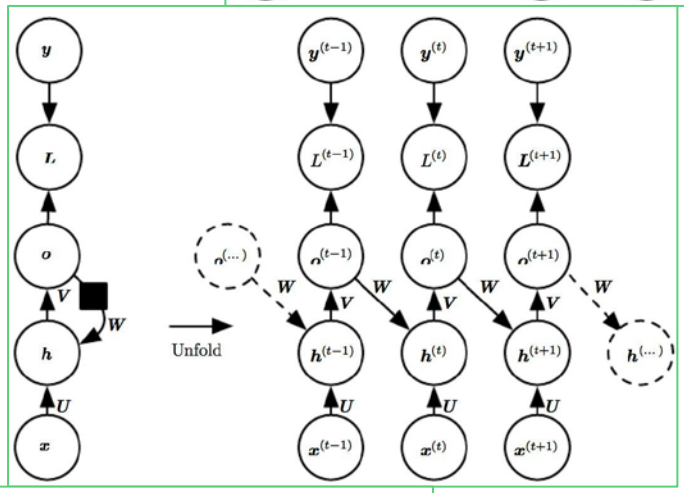


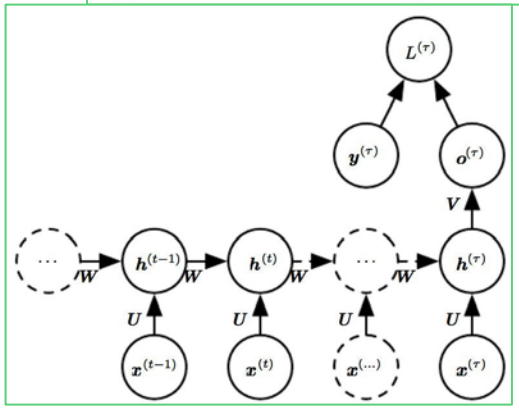
Three design patterns of RNNs

1. Output at each time step; recurrent connections between hidden units
2. Output at each time step; recurrent connections only from output at one time step to hidden units at next time step
3. Recurrent connections between hidden units to read entire input sequence and produce a single output

   Can summarize sequence to produce a fixed size representation for further processing







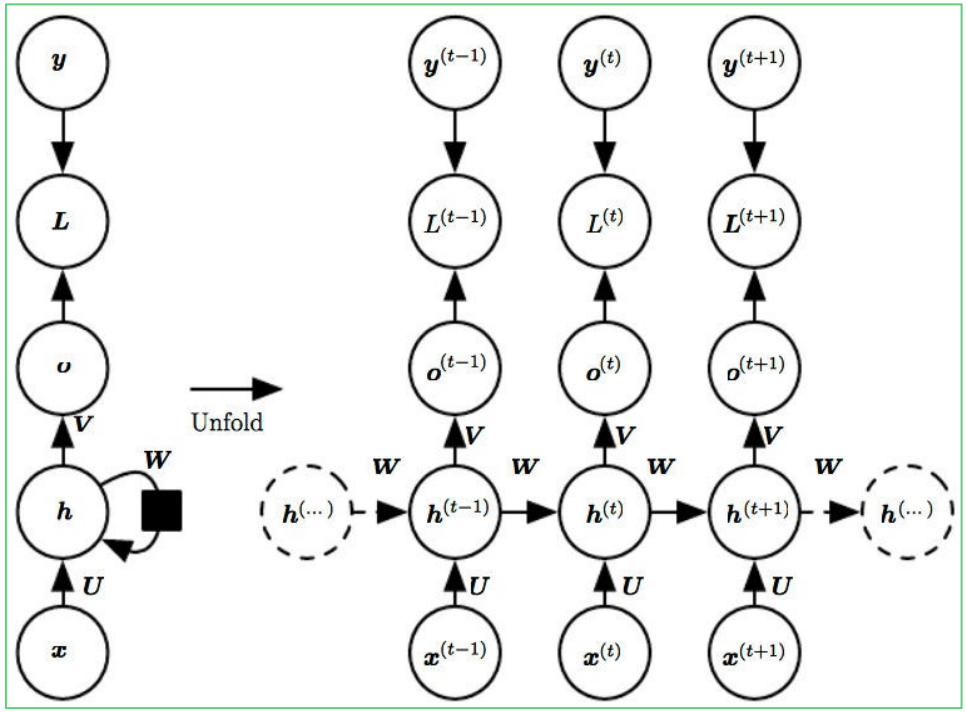
RNN1: with recurrence between hidden units

Maps input sequence ***x*** to output ***o***

With softmax outputs Loss L internally computes $\hat{y}$= softmax(o) and compares to target y

- Update equation applied for each time step from $t = 1$ to $t = \tau$

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



Parameters:

- bias vectors ***b*** and ***c***
- weight matrices U (input-to-hidden), V (hidden-to-output) and W (hidden- to-hidden) connections

Loss function for a given sequence

- The total loss for a given sequence of $\boldsymbol{x}$ values with a sequence of $\boldsymbol{y}$ values is the sum of the losses over the time steps
- If $L^{(t)}$ is the negative log-likelihood of $\boldsymbol{y}^{(t)}$ given $\boldsymbol{x}^{(1)},..\boldsymbol{x}^{(t)}$ then

$$L\left(\{x^{(1)}, x^{(2)}, \ldots, x^{(t)}\}, \{y^{(1)}, y^{(2)}, \ldots, y^{(t)}\}\right) = \sum_t L^{(t)}$$

$$= -\sum_t \log p_{model}(y^{(t)} | \{x^{(1)}, x^{(2)}, \ldots, x^{(t)}\})$$

Backpropagation through time

- We can think of the recurrent net as a layered, feed-forward net with shared weights and then train the feed-forward net with weight constraints.
- We can also think of this training algorithm in the time domain:
  - The forward pass builds up a stack of the activities of all the units at each time step.
  - The backward pass peels activities off the stack to compute the error derivatives at each time step.
  - After the backward pass we add together the derivatives at all the different times for each weight.

Gradients on V, **c**, **W** and **U**

$$\boldsymbol{a}^{(t)} = \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)}$$
$$\boldsymbol{h}^{(t)} = \tanh(\boldsymbol{a}^{(t)})$$
$$\boldsymbol{o}^{(t)} = \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(t)}$$
$$\hat{\boldsymbol{y}}^{(t)} = \mathrm{softmax}(\boldsymbol{o}^{(t)})$$
$$L\left(\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(\tau)}\}, \{\boldsymbol{y}^{(1)}, \ldots, \boldsymbol{y}^{(\tau)}\}\right) = \sum_t L^{(t)} = -\sum_t \log p_{\text{model}}\left(y^{(t)} \mid \{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(t)}\}\right)$$

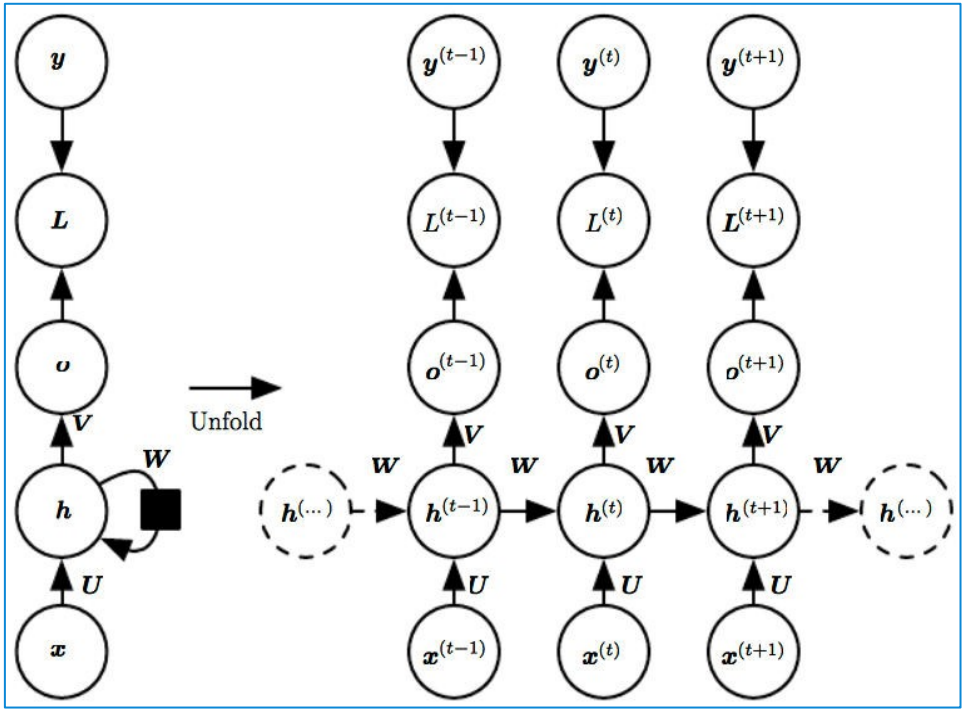
$$\frac{\partial L}{\partial L_t} = 1, \frac{\partial L}{\partial o_t} = \frac{\partial L}{\partial L_t}\frac{\partial L_t}{\partial o_t} = \frac{\partial L_t}{\partial o_t}$$

$$\frac{\partial L}{\partial V} = \sum_t \frac{\partial L_t}{\partial o_t}\frac{\partial o_t}{\partial V}$$

$$\frac{\partial L}{\partial c} = \sum_t \frac{\partial L_t}{\partial o_t}\frac{\partial o_t}{\partial c}$$

$$\frac{\partial L}{\partial W} = \sum_t \frac{\partial L_t}{\partial h_t}\frac{\partial h_t}{\partial W}$$

$$\frac{\partial L}{\partial h_t} = \frac{\partial L}{\partial h_{t+1}}\frac{\partial h_{t+1}}{\partial h_t} + \frac{\partial L}{\partial o_t}\frac{\partial o_t}{\partial h_t}$$

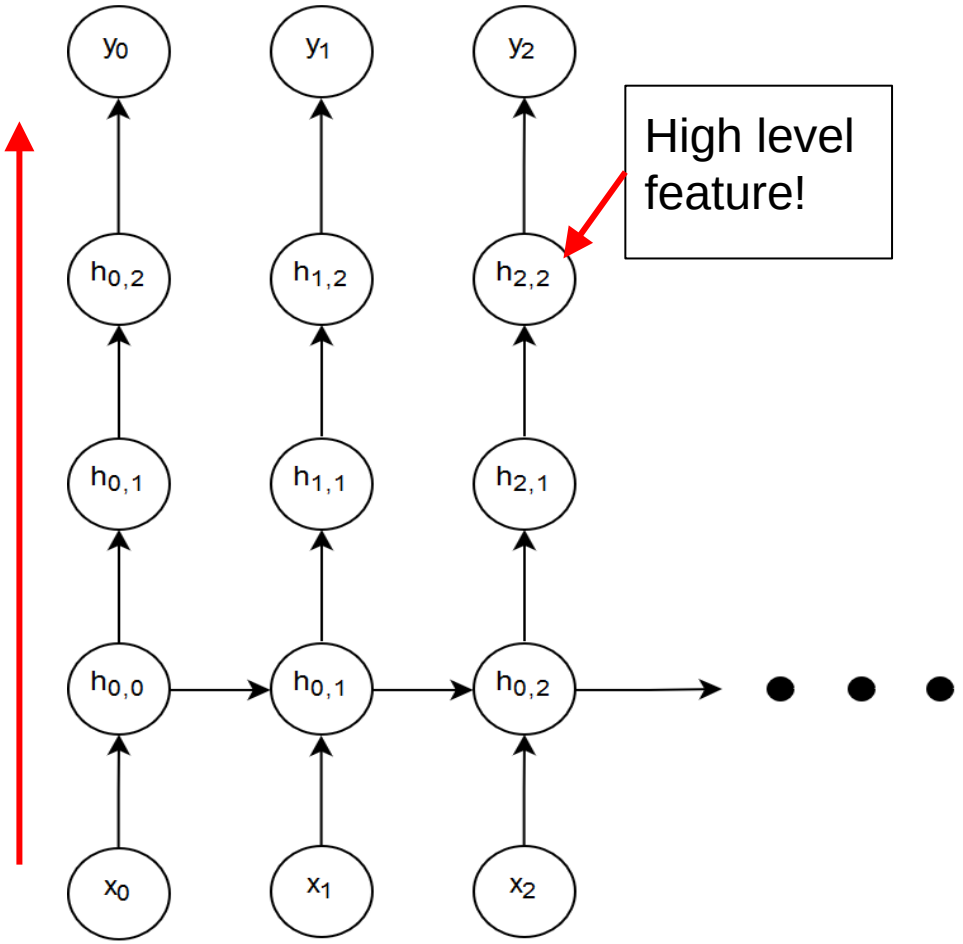


Feedforward Depth (d_f)

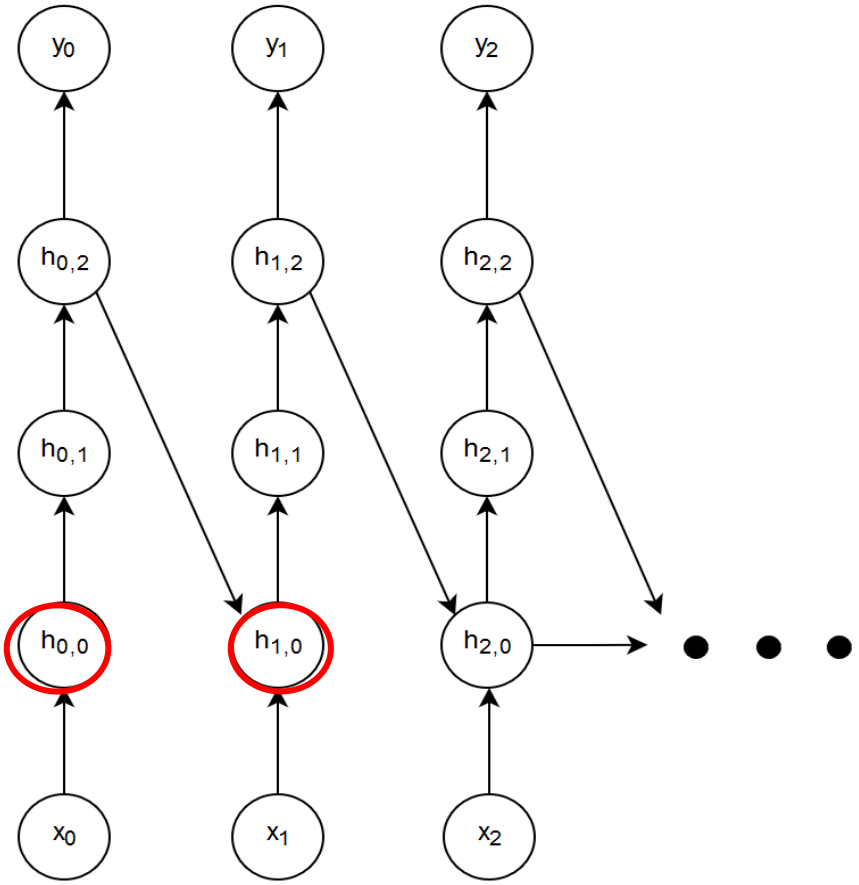
Notation: $h_{0,1}$ ⇒ time step 0, neuron #1

Feedforward depth: **longest** path between an input and output at the **same timestep**

Feedforward depth = 4



Option 2: Recurrent Depth (d_r)



Recurrent depth: Longest path between **same hidden state** in **successive timesteps**

Recurrent depth = 3