Course number: 80240743

# Deep Learning

Xiaolin Hu (胡晓林) & Jun Zhu (朱军)
Dept. of Computer Science and Technology
Tsinghua University

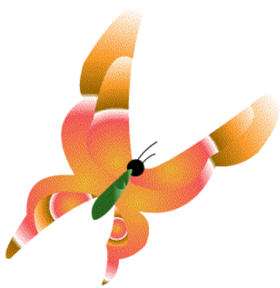
# Topic 6: Recurrent Neural Networks

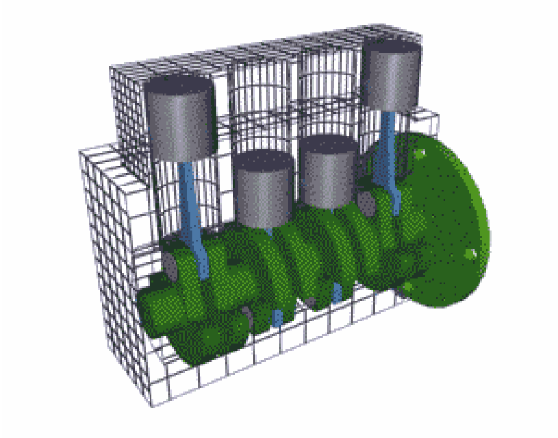
Xiaolin Hu
Dept. of Computer Science and Technology
Tsinghua University

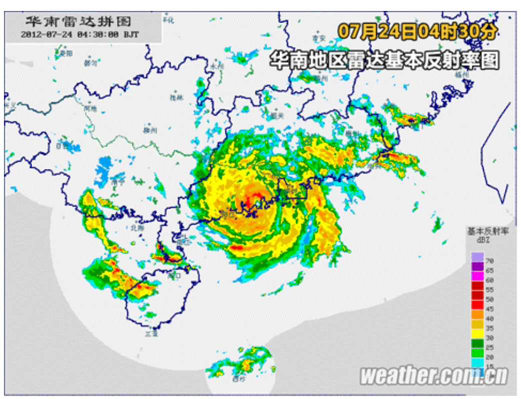
# Outline

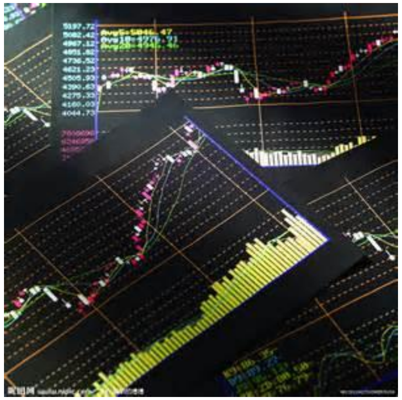
- Dynamic systems
- Simple RNNs
- Recurrent CNN
- Gated RNNs

# Dynamic systems



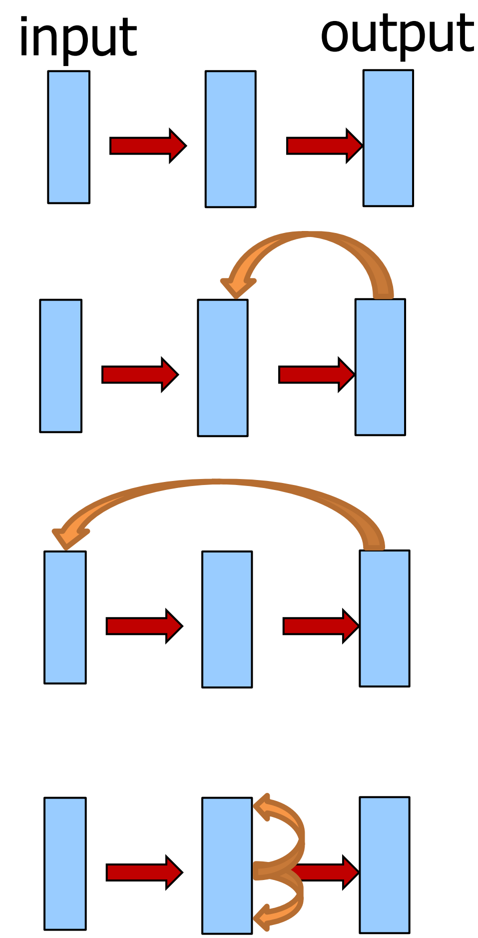




# Feedback connections

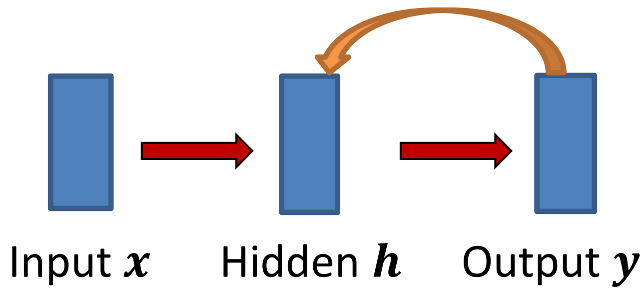
- Feedforward networks
  - No feedback
- Recurrent networks
  - Between-layer feedback: from output layer to input layer, or from hidden layer to input layer
  - Within-layer feedback




With feedback connections, the state (and therefore output) of neurons will change over time because their current input contains output of some neurons at the last time step which will change their current output

# RNNs are dynamic systems

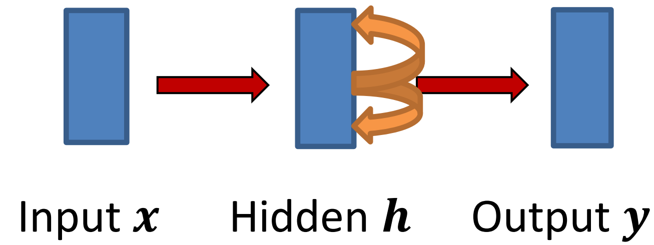
**Jordan network**




$$\boldsymbol{h}(t) = \sigma_h(\boldsymbol{W}_h\boldsymbol{x}(t) + \boldsymbol{U}_h\boldsymbol{y}(t-1) + \boldsymbol{b}_h)$$
$$\boldsymbol{y}(t) = \sigma_y(\boldsymbol{W}_y\boldsymbol{h}(t) + \boldsymbol{b}_y)$$

where $\boldsymbol{x}$ is input, $\boldsymbol{h}$ is hidden state and $\boldsymbol{y}$ is output, $\sigma_h$ and $\sigma_y$ are activation functions, $\boldsymbol{W}, \boldsymbol{U}, \boldsymbol{b}$ are parameters

**Elman network**




$$\boldsymbol{h}(t) = \sigma_h(\boldsymbol{W}_h\boldsymbol{x}(t) + \boldsymbol{U}_h\boldsymbol{h}(t-1) + \boldsymbol{b}_h)$$
$$\boldsymbol{y}(t) = \sigma_y(\boldsymbol{W}_y\boldsymbol{h}(t) + \boldsymbol{b}_y)$$

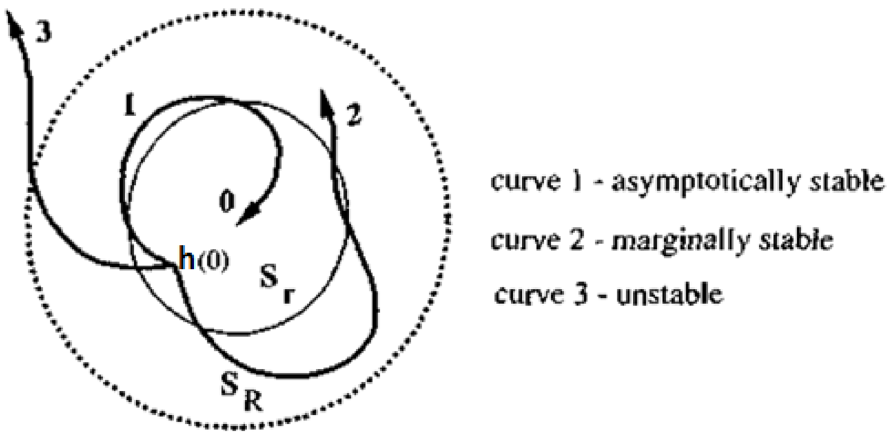
where $\boldsymbol{x}$ is input, $\boldsymbol{h}$ is hidden state and $\boldsymbol{y}$ is output, $\sigma_h$ and $\sigma_y$ are activation functions, $\boldsymbol{W}, \boldsymbol{U}, \boldsymbol{b}$ are parameters

# RNNs in general

- The states of the neurons in RNN can be written as
$$\boldsymbol{h}(t+1) = f(\boldsymbol{h}(t), \boldsymbol{x}(t))$$
where $\boldsymbol{h}$ denotes the states of *all neurons*, $\boldsymbol{x}$ denotes input to the network, and $f$ is a nonlinear function (not activation function)
- Often, the output neurons $\boldsymbol{y}$ are separated from the above equation
$$\boldsymbol{y}(t) = g(\boldsymbol{h}(t), \boldsymbol{x}(t))$$
where $g$ denotes the output function
- Such systems are termed (discrete) dynamic systems
- Linear VS nonlinear: $f$ is linear or nonlinear
- Autonomous VS non-autonomous: $f$ doesn't depend on $t$ explicitly or does

# Properties of dynamic systems

- For autonomous system
$$\boldsymbol{h}(t+1) = f(\boldsymbol{h}(t))$$
- A point $\boldsymbol{h}^*$ satisfying $\boldsymbol{h}^* = f(\boldsymbol{h}^*)$ is called a fixed point or equilibrium point
- Stability




- If $\boldsymbol{h}^*(t+T) = \boldsymbol{h}^*(t)$ for some $T$, $\boldsymbol{h}^*(t)$, $\boldsymbol{h}^*(t+1)$, … , $\boldsymbol{h}^*(t+T-1)$ is called periodic points

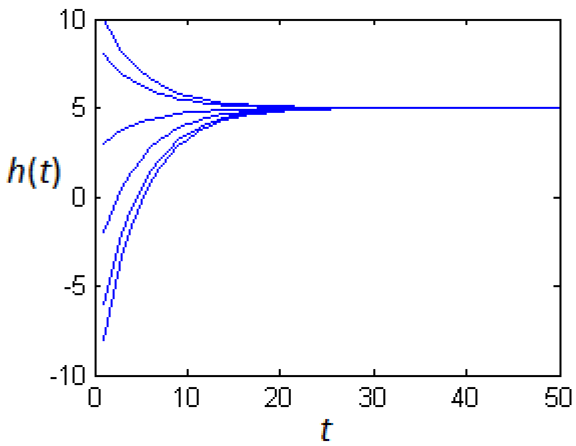
# Examples about stability

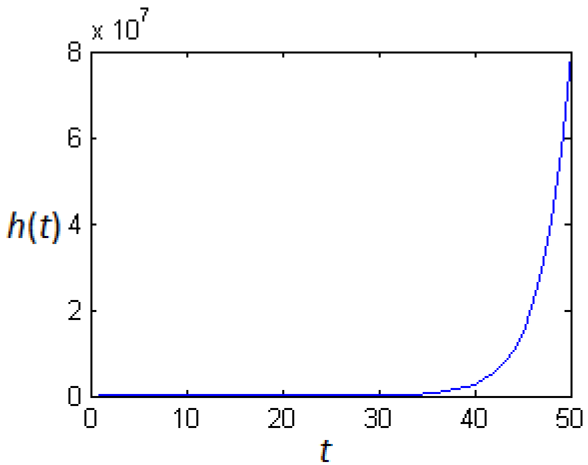
- For 1D problem, by approximating $f$ with a linear function, we get that a fixed point $h^*$ is stable whenever $|f'(h^*)| < 1$
- Consider the linear system

$$h(t+1) = 0.8h(t) + 1$$

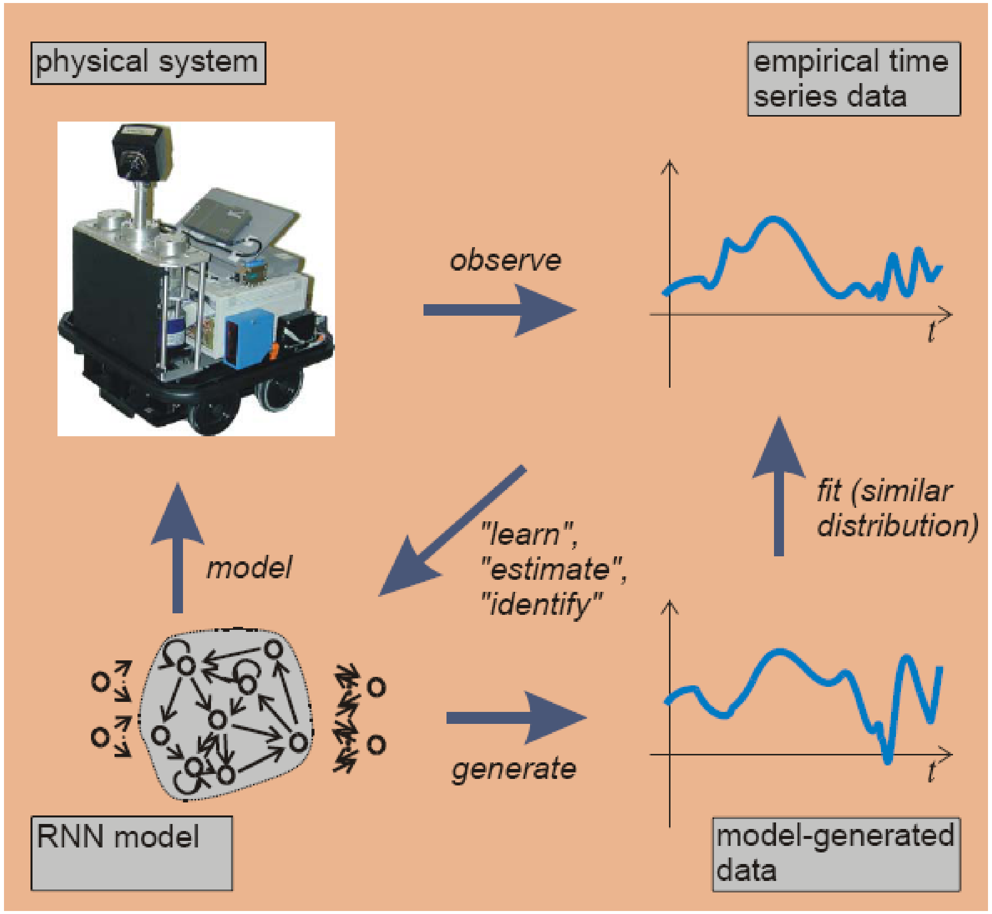
It is stable (globally converges to $h^* = 5$)

- But $h(t+1) = 1.4h(t) + 1$ is unstable



# Modeling dynamic systems with RNN



# Why do we need RNN?

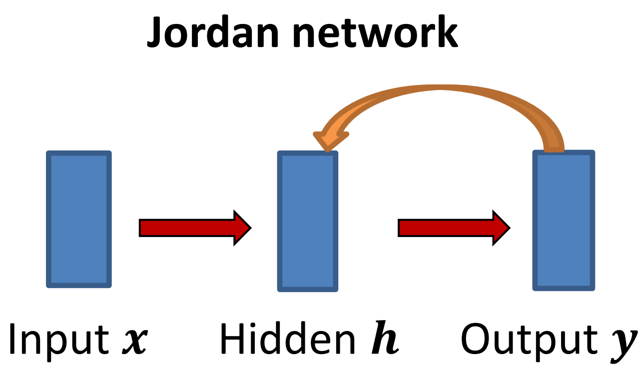
- If at every time step $t$ you always have an input $\boldsymbol{x}(t)$ and the desired output $\boldsymbol{r}(t)$
  - Then you have a set of labeled data $\{\boldsymbol{x}(t), \boldsymbol{r}(t)\}$ mapping from the input space to the output space
  - You can construct a feedforward model (e.g., MLP) to learn this mapping
  - But you cannot expect that the approach works well as it ignores the dynamic nature of the system under study
    - Different orders of inputs may induce totally different outputs (not only differing in the order)!
- Sometimes, you only have input at the beginning, but the desired output at different time is different

We use another dynamic system (RNN) to approximate the real dynamic system!

# Outline

- Dynamic systems
- Simple RNNs
- Recurrent CNN
- Gated RNNs

# Jordan network



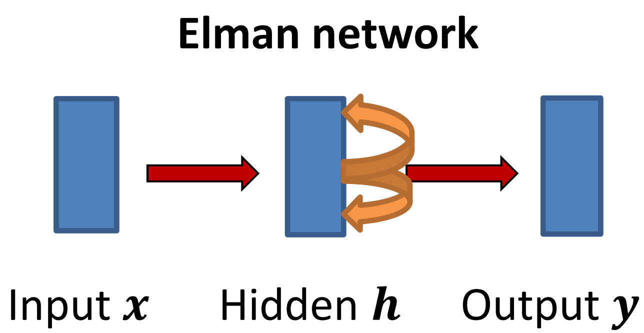

- Dynamic system:

$$\boldsymbol{h}(t) = \sigma_h(\boldsymbol{W}_h\boldsymbol{x}(t) + \boldsymbol{U}_h\boldsymbol{y}(t-1) + \boldsymbol{b}_h)$$
$$\boldsymbol{y}(t) = \sigma_y(\boldsymbol{W}_y\boldsymbol{h}(t) + \boldsymbol{b}_y)$$

where $\boldsymbol{x}$ is input, $\boldsymbol{h}$ is hidden state and $\boldsymbol{y}$ is output, $\sigma_h$ and $\sigma_y$ are activation functions, $\boldsymbol{W}, \boldsymbol{U}, \boldsymbol{b}$ are parameters

- Michael I Jordan
  - BS in Psychology in 1978 from the Louisiana State University,
  - MS in Mathematics in 1980 from Arizona State University
  - PhD in Cognitive Science in 1985 from the UCSD
- At UCSD, Jordan was a student of David Rumelhart
- Now at UC Berkeley

# Elman network




January 22, 1948 – June 28, 2018

- Dynamic system:

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

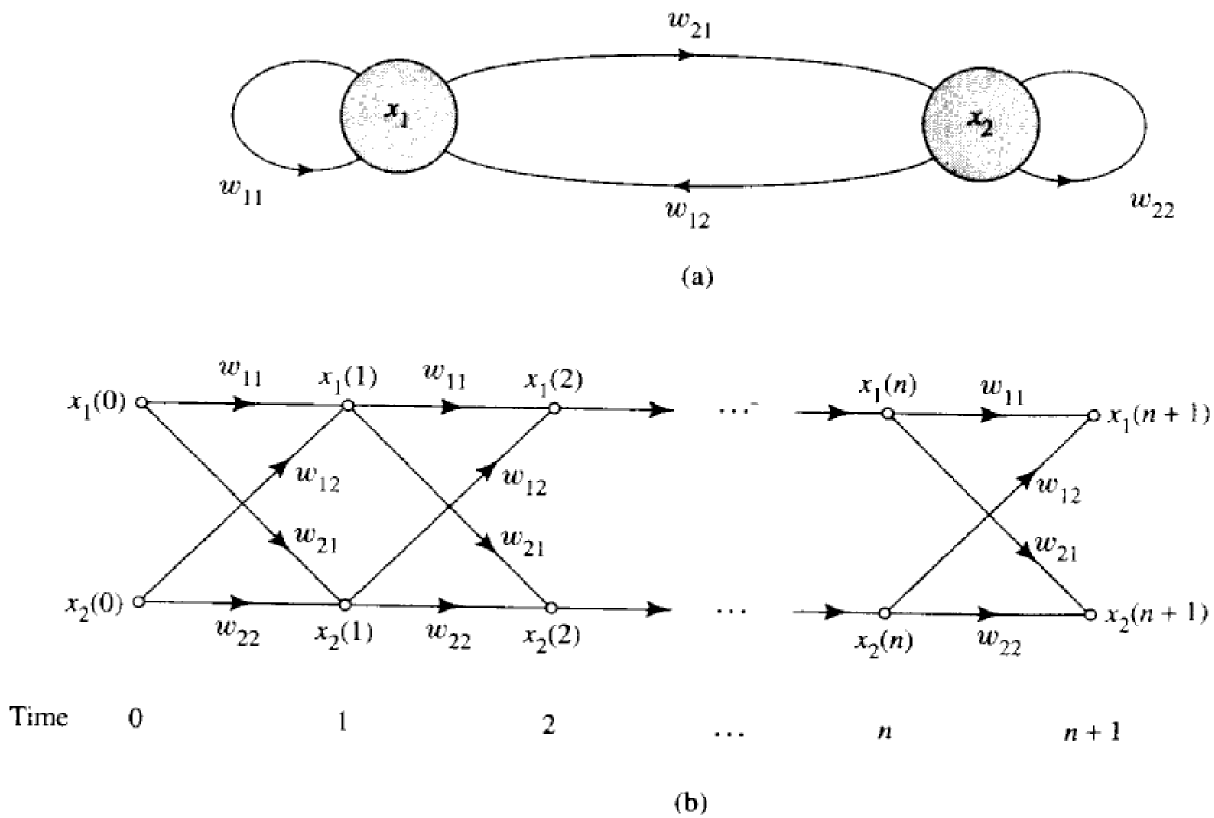
$$\boldsymbol{y}(t) = \sigma_y(\boldsymbol{W}_y\boldsymbol{h}(t) + \boldsymbol{b}_y)$$

where $\boldsymbol{x}$ is input, $\boldsymbol{h}$ is hidden state and $\boldsymbol{y}$ is output, $\sigma_h$ and $\sigma_y$ are activation functions, $\boldsymbol{W}, \boldsymbol{U}, \boldsymbol{b}$ are parameters

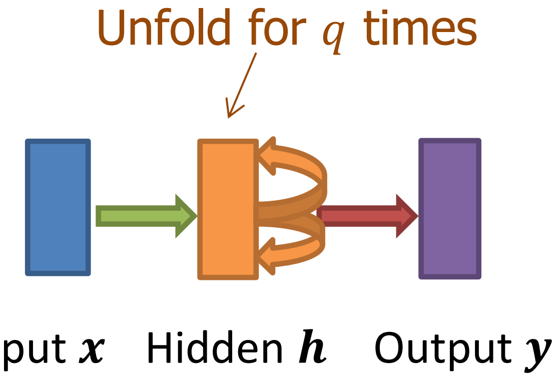
- Jeffrey Elman
  - BS in 1969 from Harvard University
  - Ph.D. in 1977 from the University of Texas at Austin
- Professor of cognitive science at the UCSD

# Back-propagation through time (BPTT)

Unfold the temporal operation of the network into a layered feedforward network, the topology of which grows by one layer at every time step.



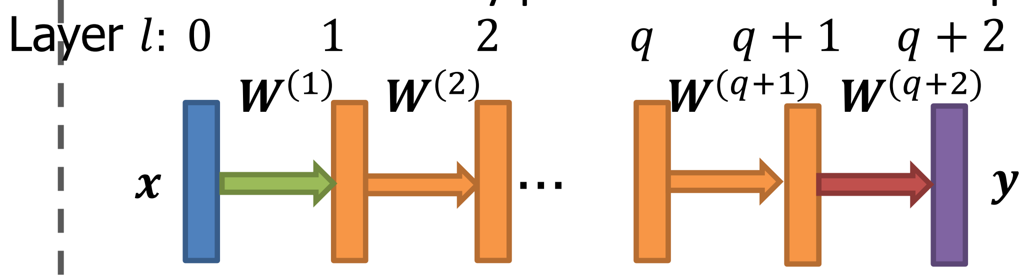
# Unfold the Elman network




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

$$\boldsymbol{y}(t) = \sigma_y(\boldsymbol{W}_y \boldsymbol{h}(t) + \boldsymbol{b}_y)$$

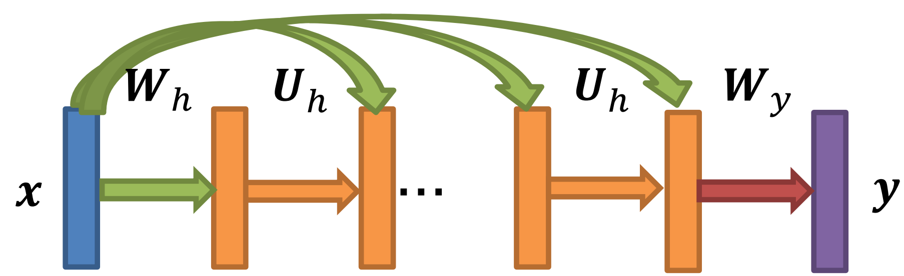
- Case 1:
  - $\boldsymbol{x}$ is only present at the first step
  - Label $\boldsymbol{r}$ is only present at the last step




$$\boldsymbol{W}^{(1)} = \boldsymbol{W}_h, \boldsymbol{W}^{(2)} = \cdots = \boldsymbol{W}^{(q+1)} = \boldsymbol{U}_h$$

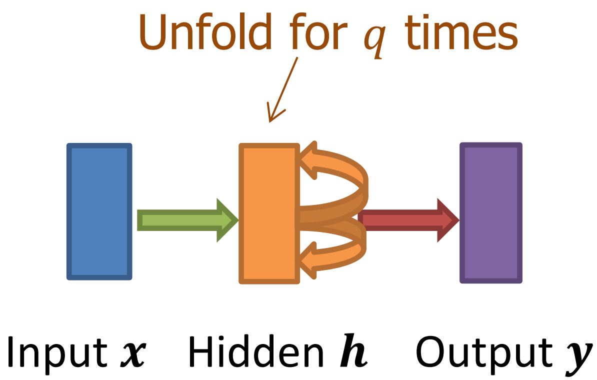
$$\boldsymbol{W}^{(q+2)} = \boldsymbol{W}_y$$

- Case 2:
  - $\boldsymbol{x}$ is fixed but present at all steps
  - Label $\boldsymbol{r}$ is only present at the last step
  - E.g., image classification (Liang, Hu, CVPR 2015)




(Arrows in the same color share weights)

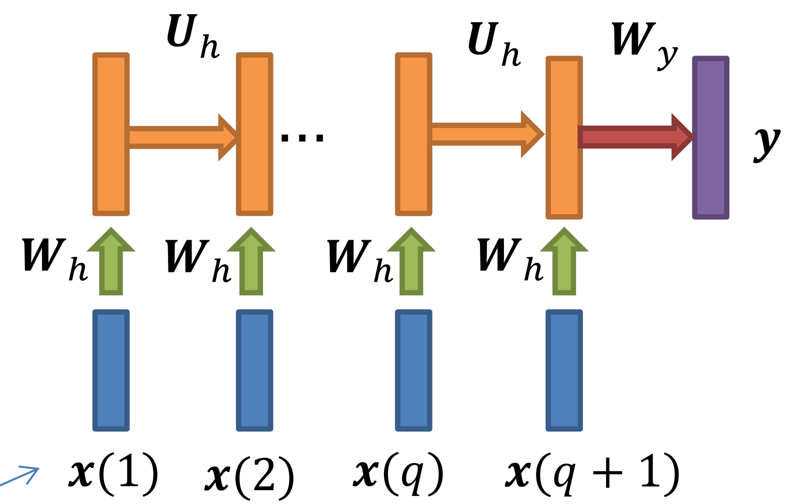
# Unfold the Elman network




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

$$\boldsymbol{y}(t) = \sigma_y(\boldsymbol{W}_y\boldsymbol{h}(t) + \boldsymbol{b}_y)$$

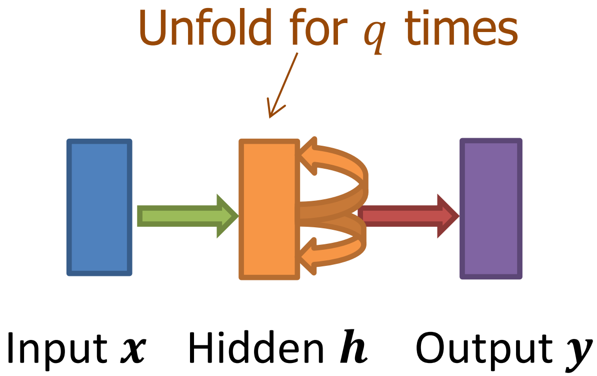
- Case 3:
  - $\boldsymbol{x}$ is time-varying
  - Label $\boldsymbol{r}$ is only present at the last step
  - E.g., sentence classification




This can be viewed as layer 0 attached to the orange backbone

(Arrows in the same color share weights)

# Unfold the Elman network

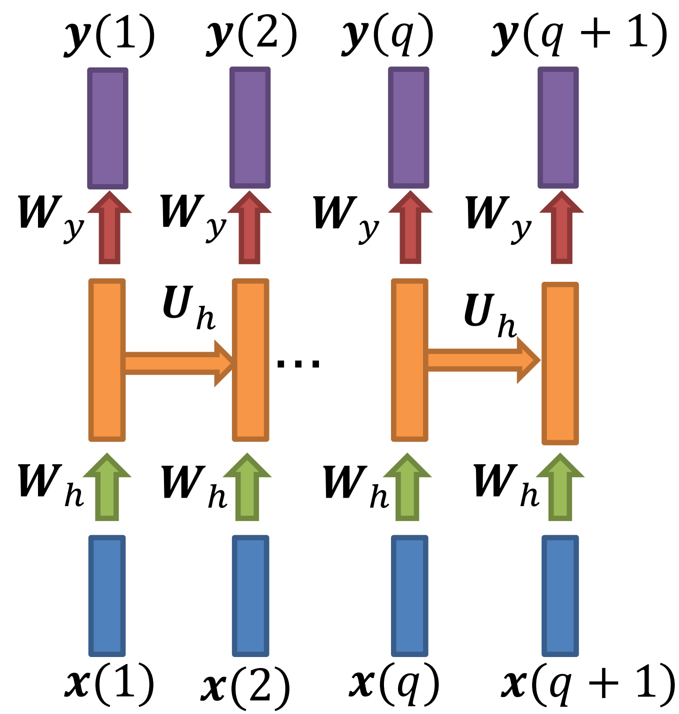



$$\boldsymbol{h}(t) = \sigma_h(\boldsymbol{W}_h\boldsymbol{x}(t) + \boldsymbol{U}_h\boldsymbol{h}(t-1) + \boldsymbol{b}_h)$$

$$\boldsymbol{y}(t) = \sigma_y(\boldsymbol{W}_y\boldsymbol{h}(t) + \boldsymbol{b}_y)$$

Do you know other cases?

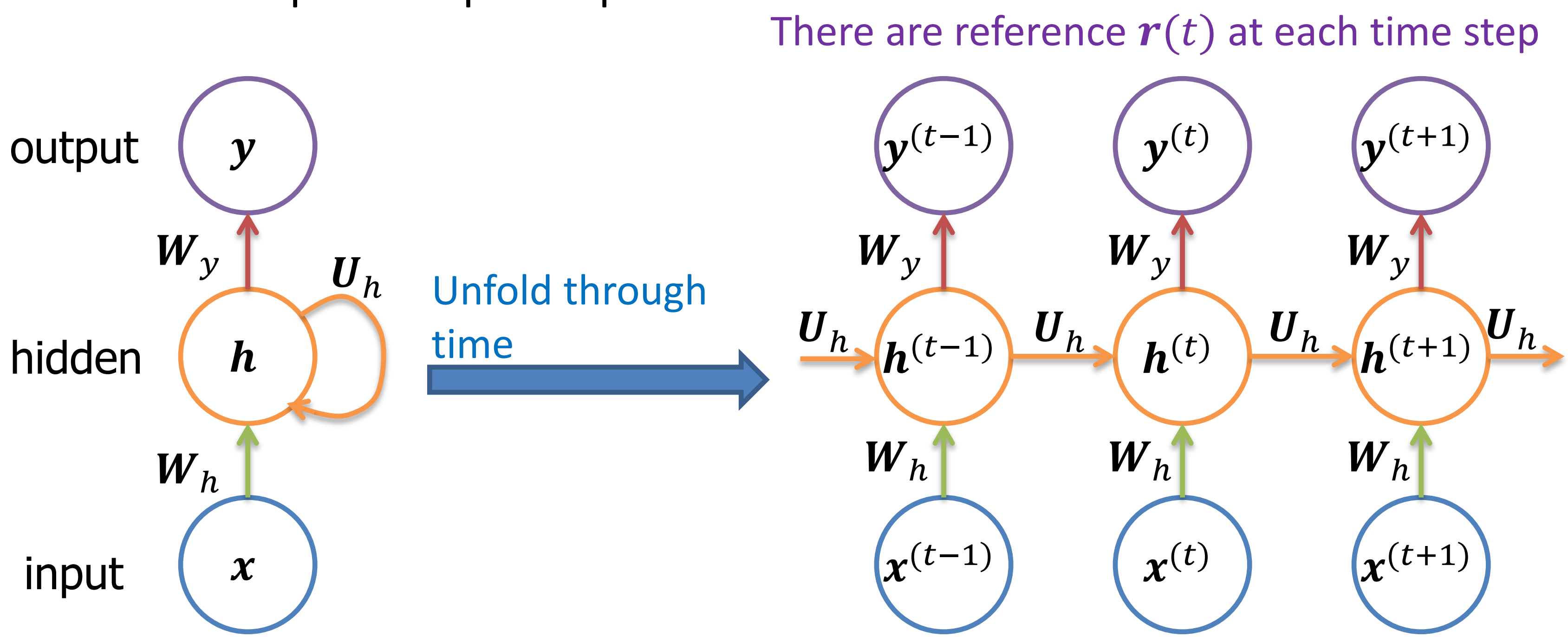
- Case 4:
  - $\boldsymbol{x}$ is time-varying
  - Label $\boldsymbol{r}$ is present at all steps
  - E.g., speech recognition




(Arrows in the same color share weights)

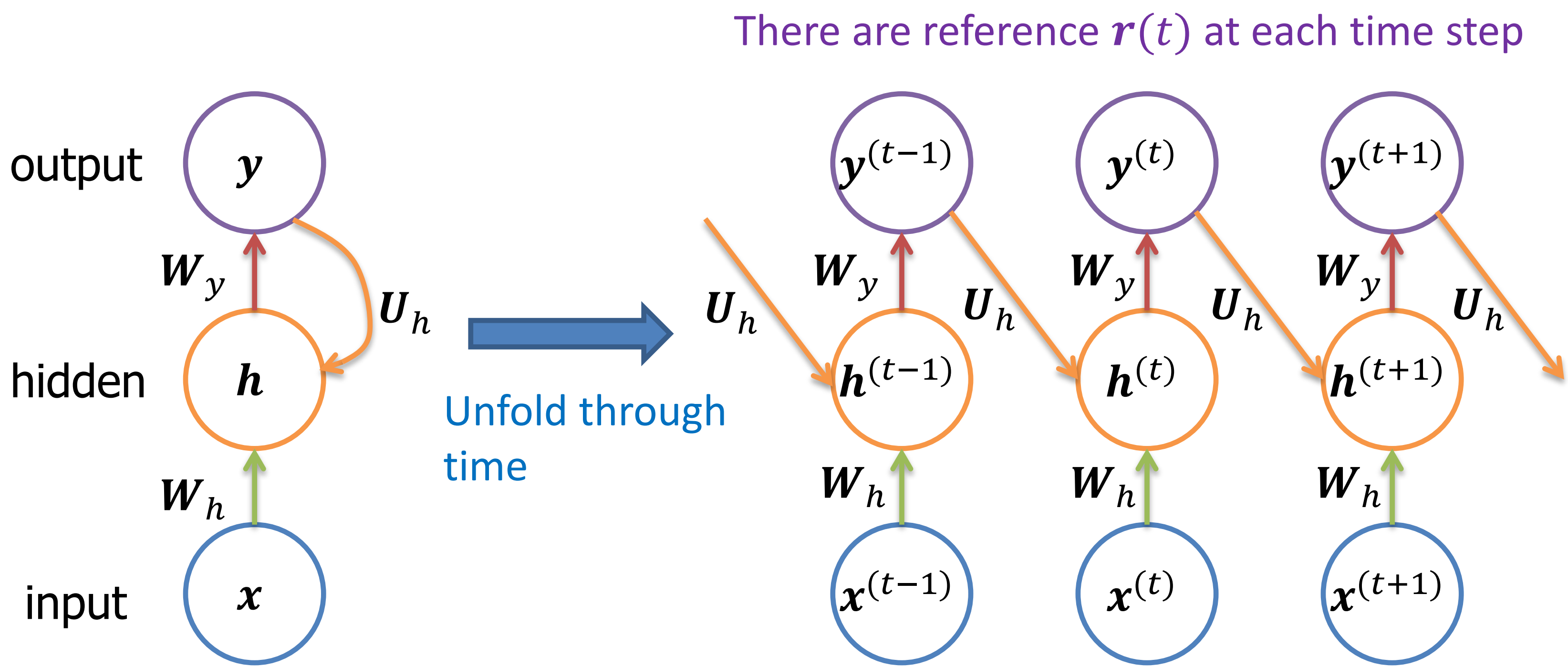
# Simplified illustration (Elman network)

- Use circles to represent vectors (one circle one layer)
- Put time step into superscript




- The forward propagation runtime is $O(q)$ and cannot be reduced
- Equivalent to a Turing machine (due to the hidden-to-hidden) connections

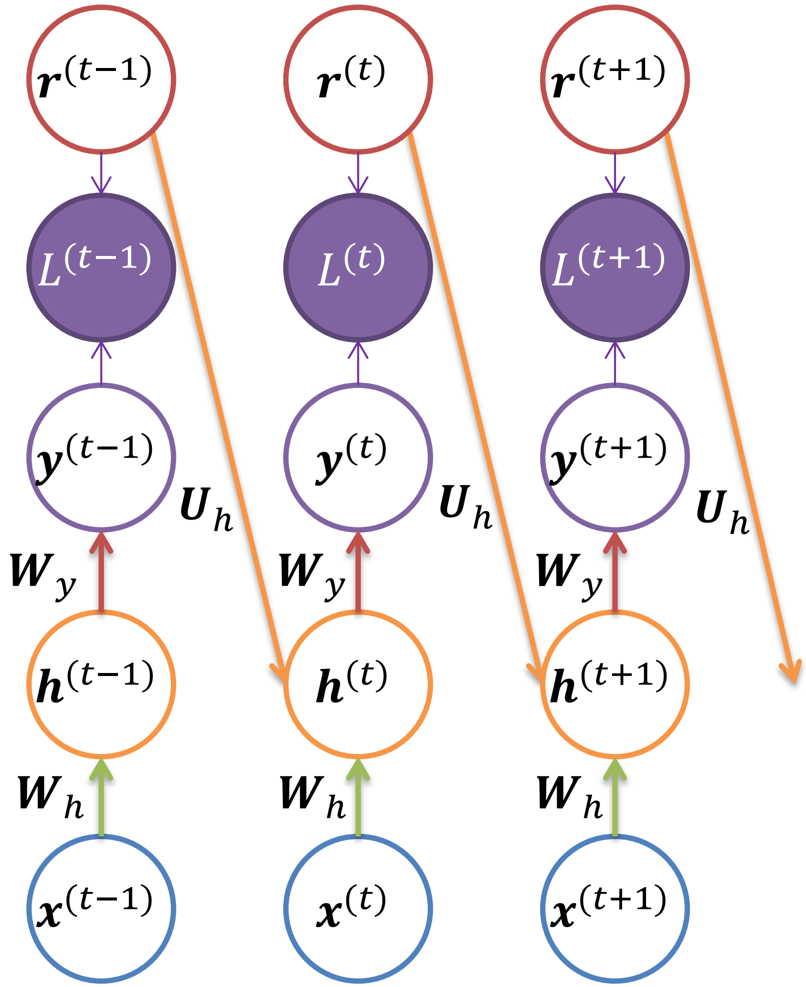
# Unfold the Jordan network




- It is strictly less powerful and it cannot simulate a universal Turing machine
- If the loss is based on comparing $\boldsymbol{y}(t)$ and $\boldsymbol{r}(t)$, all time steps are decoupled and training can be parallelized

# Teacher forcing

- Some networks such as Jordan network, have connections from the output at one time step to values computed in the next time step
- Then, what should be input to the next time step to represent the output?
- Teacher forcing: in training, we use the reference signal



# Teacher forcing

- However, in testing, there is no reference signal and we have to use the network's output at time $t$
- The kind of inputs that the network sees during training could be quite different from the kind of inputs that it will see at test time
- To mitigate this problem:
  - Alternate use of teacher-forced inputs and free-running inputs for a number of time steps
  - Randomly choose between the teacher-forced input and free-running input at every time step

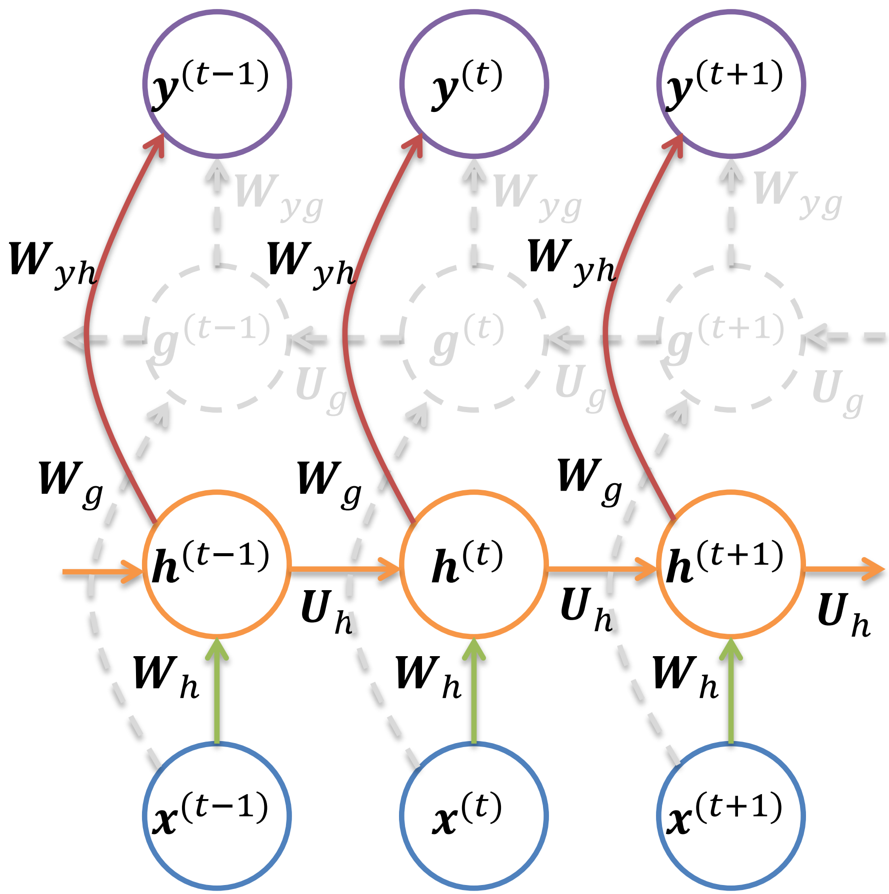
# Bidirectional RNN

- In many applications, the prediction at the current time step may depend on the whole input sequence (both the past and the future)

  *Bank is the side of a river.*
  *Bank provides various financial services.*

- Bidirection RNNs combine an RNN that moves forward through time with another RNN that moves backward through time
- The output of the entire network at every time step then receives two inputs

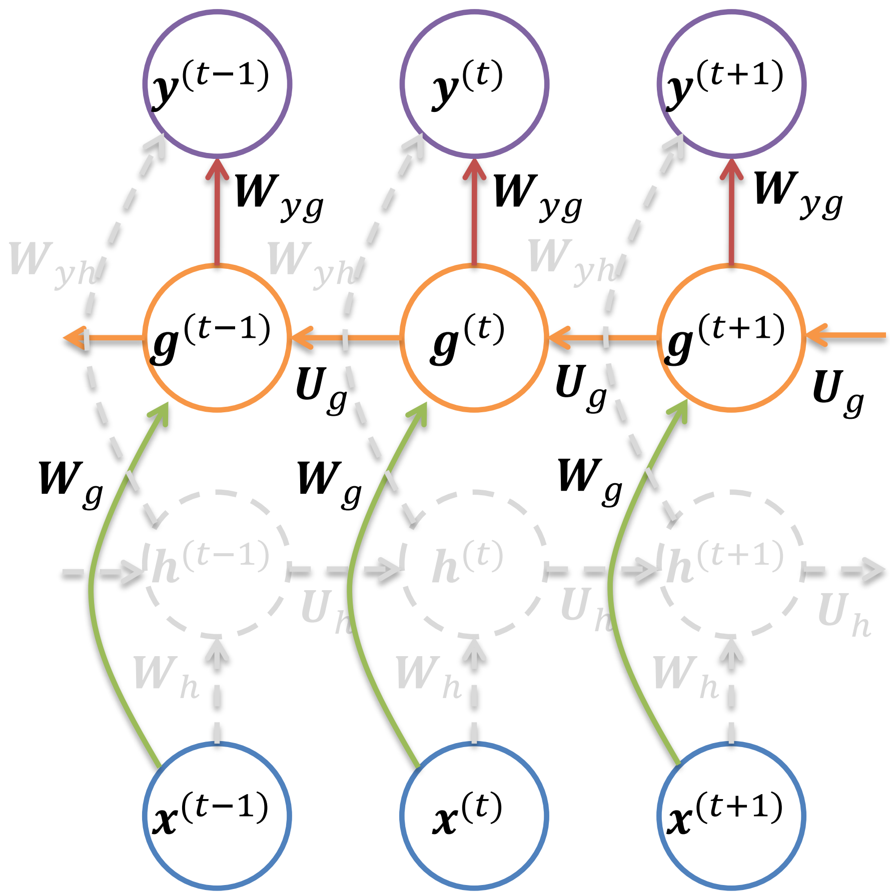
# Bidirectional RNN




$\boldsymbol{W}_{yh}$: connection weights from the sub-RNN $\boldsymbol{g}$ to output

One sub-RNN moves forward, same as before

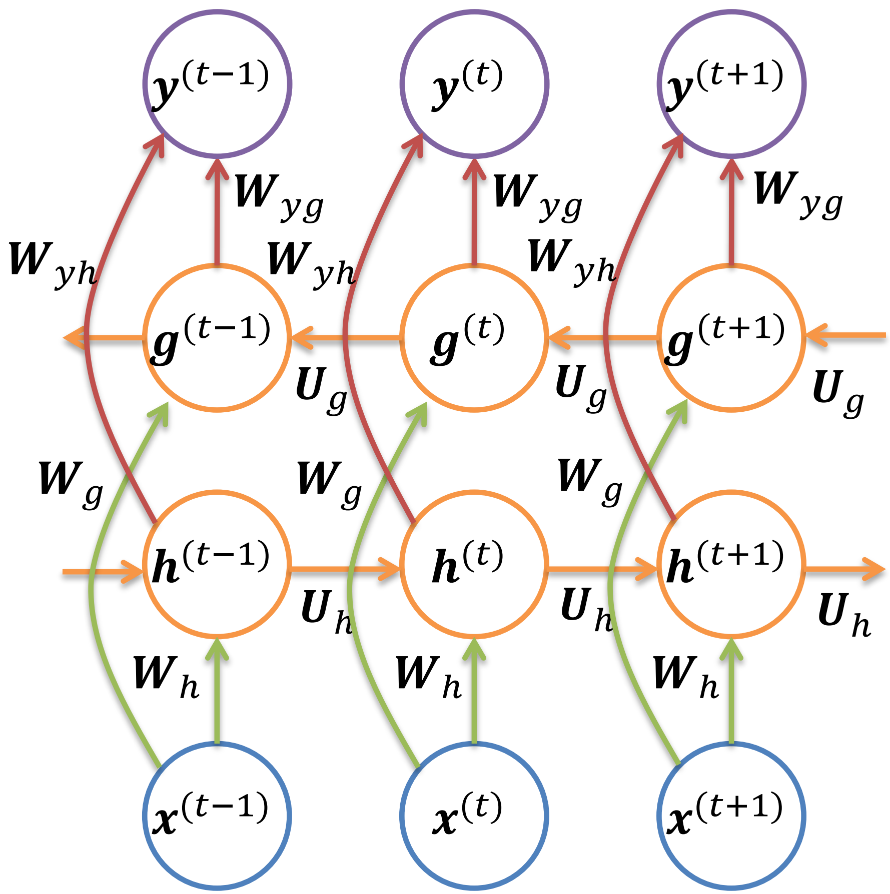
# Bidirectional RNN




$W_{yg}$: connection weights from the sub-RNN $g$ to output

One sub-RNN moves backward

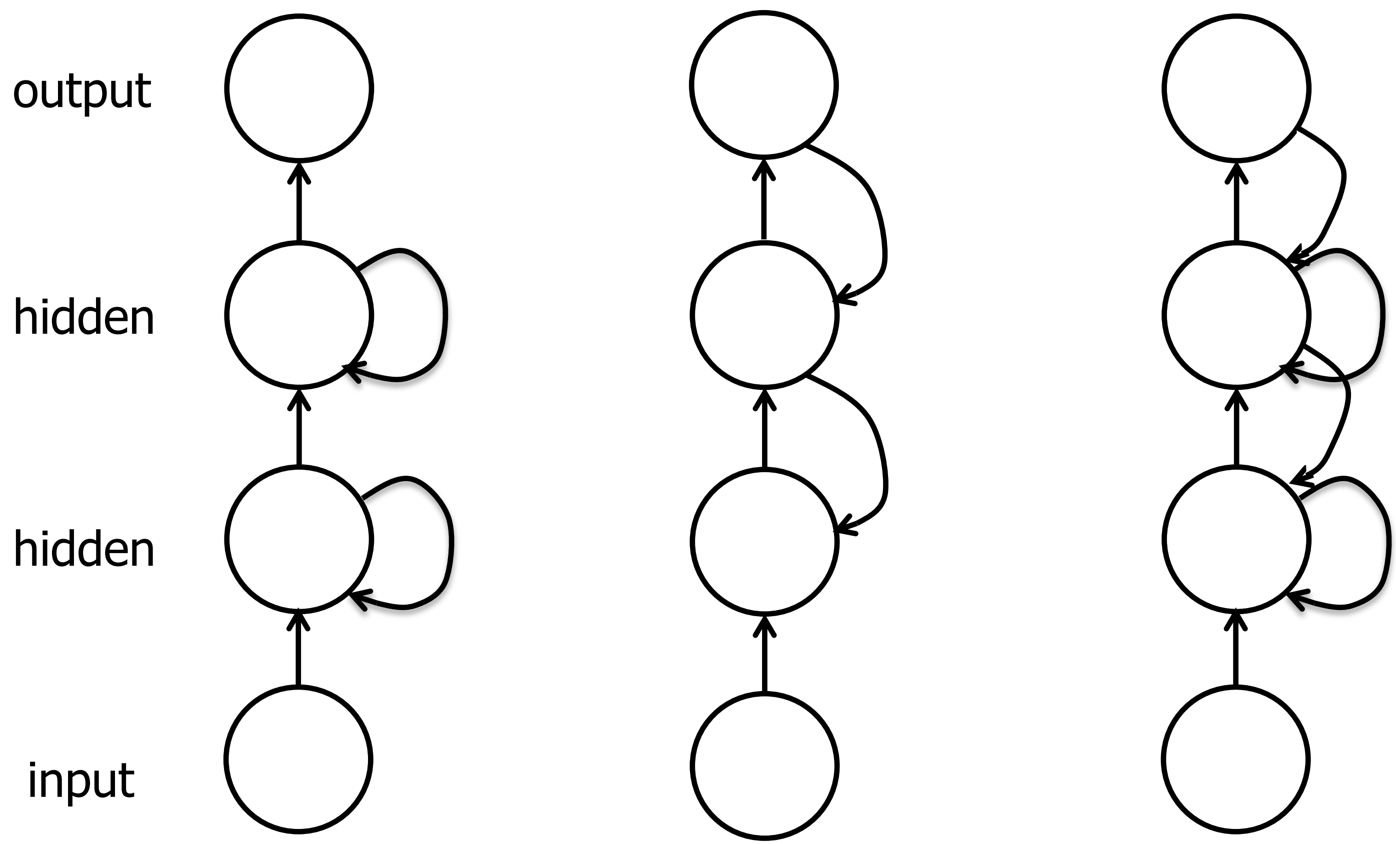
# Bidirectional RNN




The output $\boldsymbol{y}^{(t)}$ recives input from both sub-RNNs via $\boldsymbol{W}_{yh}$ and $\boldsymbol{W}_{yg}$

# Deep RNNs

- Many ways to construct deep RNNs



# Challenges

- Simple RNNs tend to converge to fixed points or explode
- Recall the Elman network:

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

  - Suppose $\sigma_h$ is an identity mapping, $\boldsymbol{b} = \boldsymbol{0}$ and the input is only present at the beginning. After $t$ steps from zero

$$\boldsymbol{h}(t) = \boldsymbol{U}^t \boldsymbol{h}(0)$$

  - Suppose $\boldsymbol{U}^t$ has an eigen-decomposition $\boldsymbol{U} = \boldsymbol{V}\mathrm{diag}(\boldsymbol{\lambda})\boldsymbol{V}^{-1}$, then

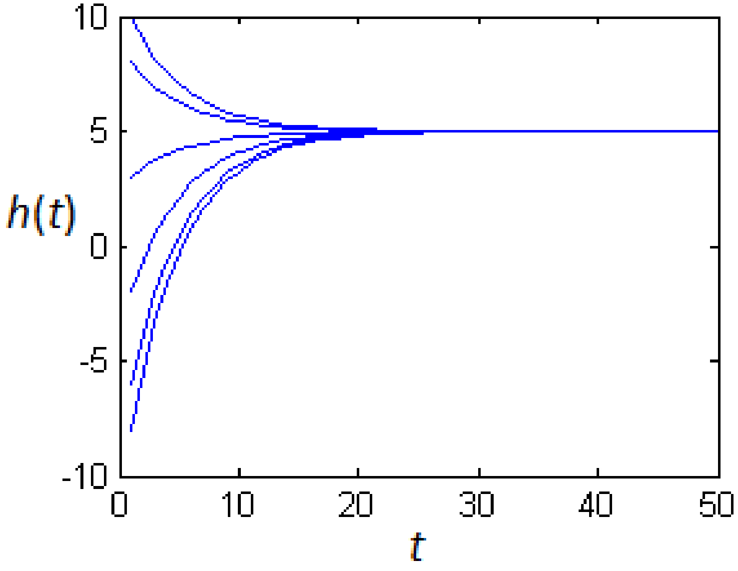
$$\boldsymbol{U}^t = (\boldsymbol{V}\mathrm{diag}(\boldsymbol{\lambda})\boldsymbol{V}^{-1})^t = \boldsymbol{V}\mathrm{diag}(\boldsymbol{\lambda})^t\boldsymbol{V}^{-1}$$

  - Any eigenvalue $\lambda_i$ that is not near an absolute value of 1 will either explode or vanish
  - Then the components of $\boldsymbol{h}(t)$ would
    - either converge to *zero*
    - or explode

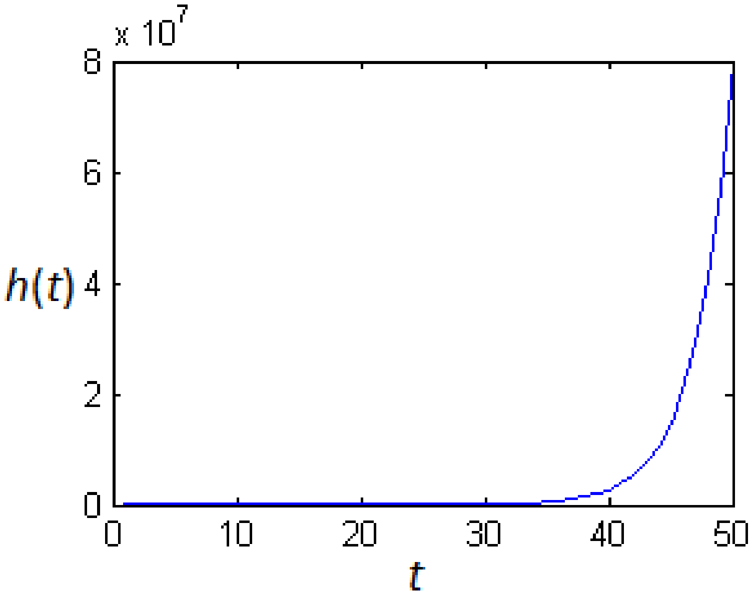
If $\boldsymbol{b}$ is not zero, then converge to nonzero values

# Recall the 1D case

$$h(t+1) = 0.8h(t) + 1$$




$$h(t+1) = 0.8h(t) + 1$$



# Challenges (backward pass)

- Recall the BP algorithm for MLP

$$\boldsymbol{\delta}^{(l)} = (\boldsymbol{W}^{(l+1)})^\top \boldsymbol{\delta}^{(l+1)} \odot \boldsymbol{f}'(\boldsymbol{u}^{(l)})$$

- For the Elman network, $\boldsymbol{W}$ changes to $\boldsymbol{U}$
- With the same assumption as in the previous slides, we have

$$\boldsymbol{\delta}^{(T-t)} = (\boldsymbol{U}^\top)^t \boldsymbol{\delta}^{(T)}$$

  where $T$ denotes the last time step

  - Suppose $\boldsymbol{U}^\top$ has an eigen-decomposition $\boldsymbol{U}^\top = \boldsymbol{V}\mathrm{diag}(\boldsymbol{\lambda})\boldsymbol{V}^{-1}$, then

    $$(\boldsymbol{U}^\top)^t = (\boldsymbol{V}\mathrm{diag}(\boldsymbol{\lambda})V^{-1})^t = \boldsymbol{V}\mathrm{diag}(\boldsymbol{\lambda})^t V^{-1}$$

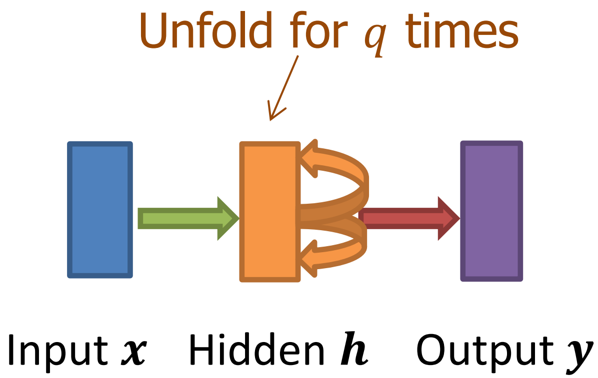
  - Any eigenvalue $\lambda_i$ that is not near an absolute value of 1 will either explode or vanish
  - Then the gradient either vanishes or explodes

Even if the input is always present, the challenges exist

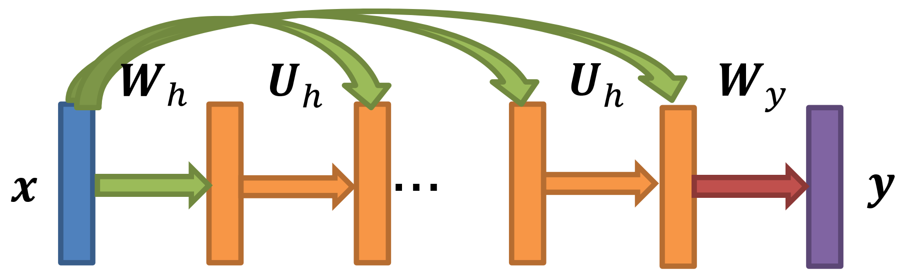
# Outline

- Dynamic systems
- Simple RNNs
- Recurrent CNN
- Gated RNNs

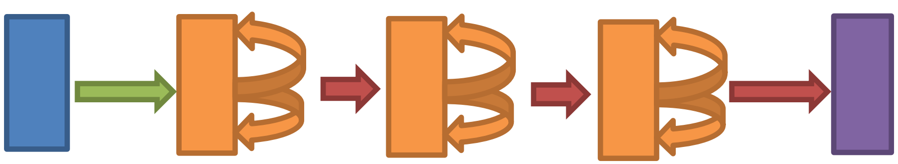
# Recall one case of unfolding the Elman network




- Case 2:
  - $\boldsymbol{x}$ is fixed but present at all steps
  - Label $\boldsymbol{r}$ is only present at the last step



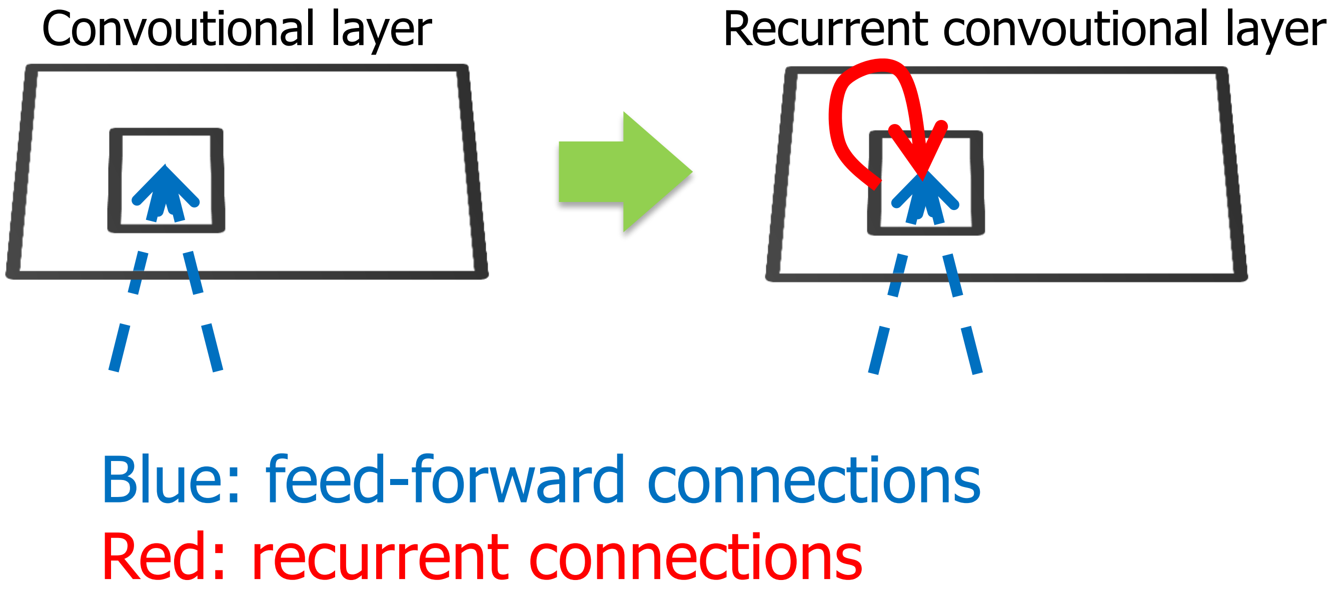

- If you have multiple such layers, it is called a recurrent MLP



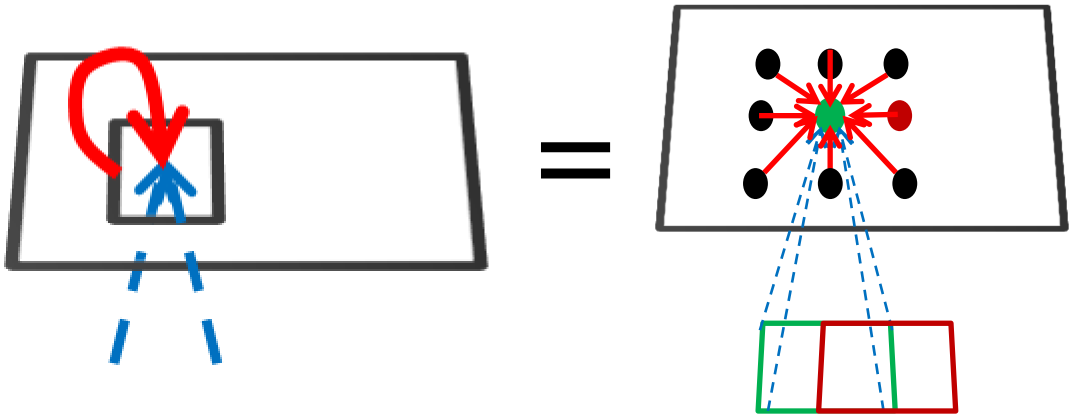
- If this idea is applied to CNN, then you obtain a recurrent CNN (Liang, Hu, CVPR 2015; Liang, Hu, Zhang, NIPS 2015; Zhao et al., ICASSP 2017)

# Recurrent convolutional layer (RCL)

- Incorporate intra-layer recurrent connections into each convolutional layer



# RCL as 2D RNN



At each iter, the state of a unit is influenced by a *larger* area in the layer below

↓

*Context modulation!*

It becomes a dynamic system since the states change over time (here discrete iterations)

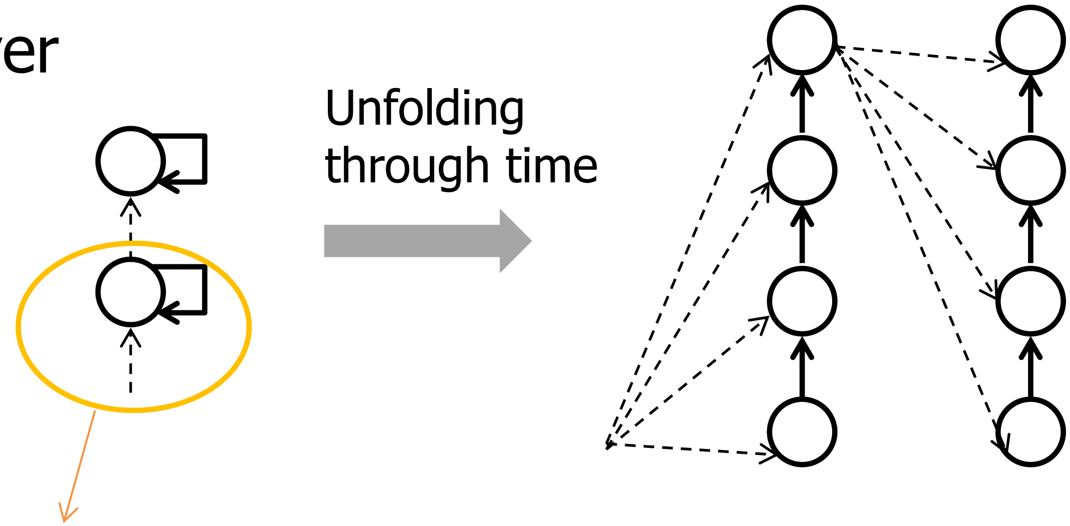
State (time-varying) ↓ Input (static) ↓

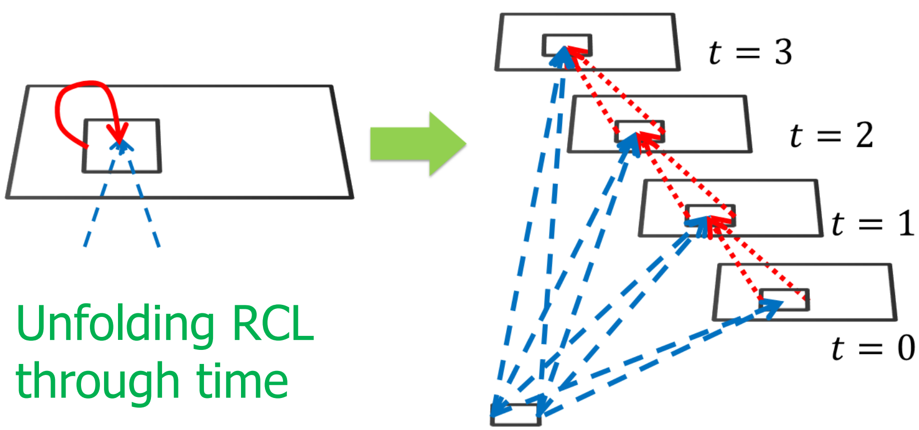
$$x_{ijk}(t) = \boldsymbol{\sigma}\left(\left(\mathbf{w}_k^{in}\right)^{\mathrm{T}}\mathbf{u}^{(i,j)} + \left(\mathbf{w}_k^{rec}\right)^{\mathrm{T}}\mathbf{x}^{(i,j)}(t-1) + b_k\right)$$

In RCL, both feed-forward and recurrent computation take the form of convolution (local connections)

# Stacking RCL's to a deep structure

A two-layer RCNN



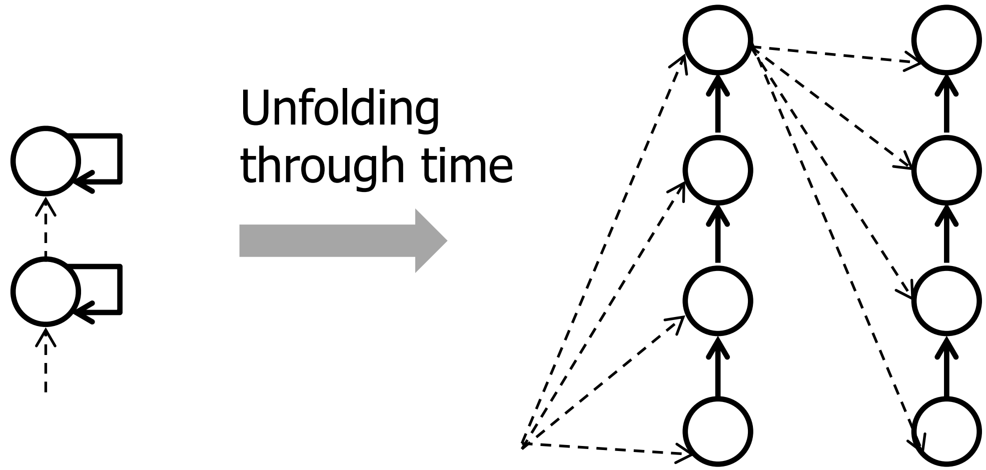




Unfolding RCL through time

$$x_{ijk}(t) = \boldsymbol{\sigma}\left(\left(\mathbf{w}_k^{in}\right)^{\mathrm{T}}\mathbf{u}^{(i,j)} + \left(\mathbf{w}_k^{rec}\right)^{\mathrm{T}}\mathbf{x}^{(i,j)}(t-1) + b_k\right)$$

# RCNN properties

A two-layer RCNN

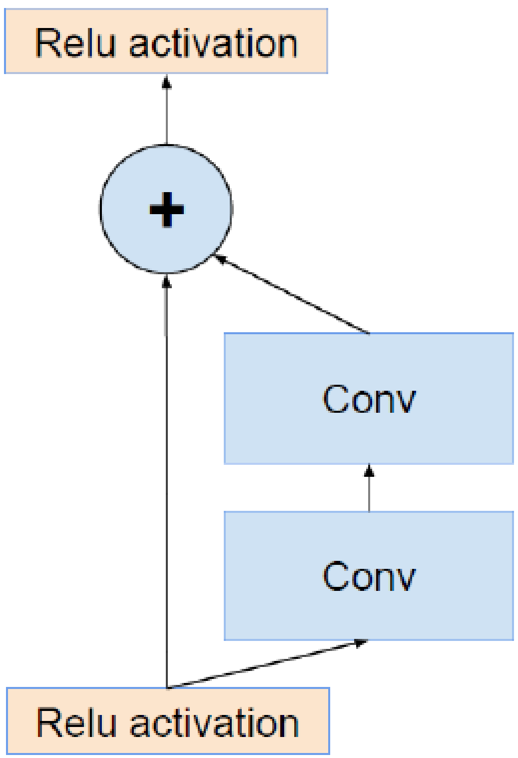



- A deep model with limited number of parameters
- Neurons in each layer can capture global context
- Multi-path structure

# Multi-path in Residual Net

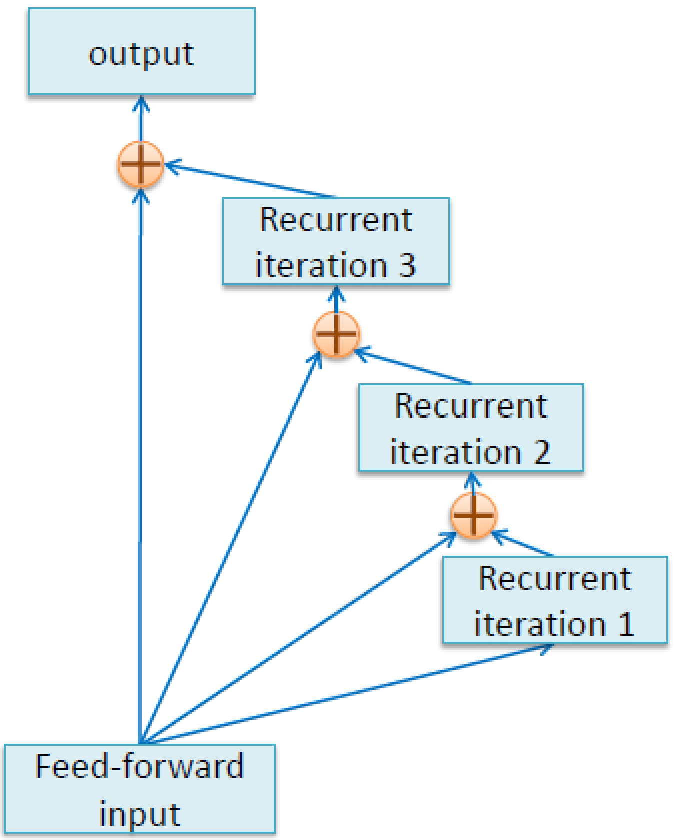
- 152 layers for ImageNet; 100 or 1000 layers for CIFAR-10 (He et al., 2016)

Residual Net




Adapted from Szegedy et al. 2016

RCL



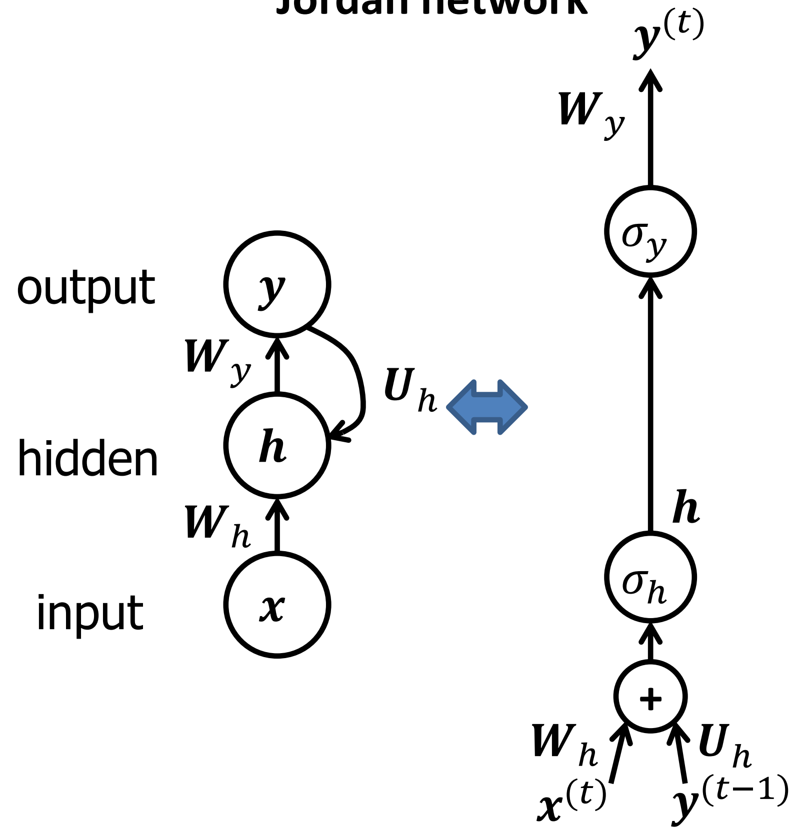
# Outline

- Dynamic systems
- Simple RNNs
- Recurrent CNN
- Gated RNNs

# Long short-term memory (LSTM) cell

- It can be viewed as a combination of the Jordan network and the Elman network
  - The output is connect to the input
  - A self-loop is used to capture the information about the past
- Redraw the Jordan network
  - Use circles to denote operations
  - Variables are indicated on arrows

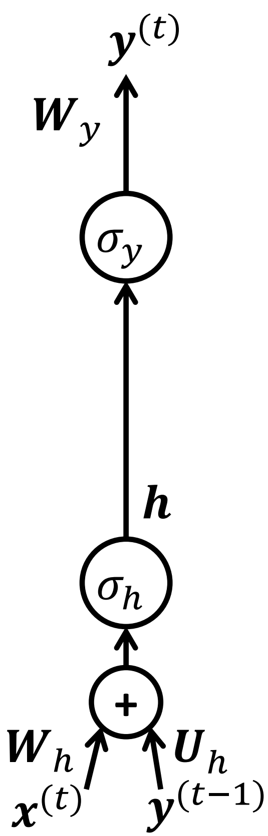
**Jordan network**




$$\boldsymbol{h}^{(t)} = \sigma_h\left(\boldsymbol{W}_h\boldsymbol{x}^{(t)} + \boldsymbol{U}_h\boldsymbol{y}^{(t-1)} + \boldsymbol{b}_h\right)$$
$$\boldsymbol{y}^{(t)} = \sigma_y(\boldsymbol{W}_y\boldsymbol{h}^{(t)} + \boldsymbol{b}_y)$$

# Step 1: add a self-loop

**Jordan network**

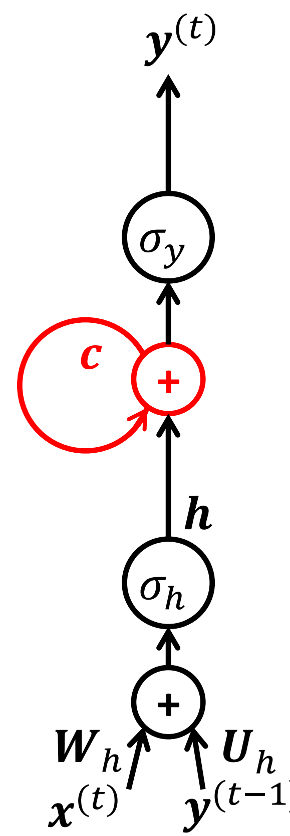



$$\boldsymbol{h}^{(t)} = \sigma_h\left(\boldsymbol{W}_h \boldsymbol{x}^{(t)} + \boldsymbol{U}_h \boldsymbol{y}^{(t-1)} + \boldsymbol{b}_h\right)$$
$$\boldsymbol{y}^{(t)} = \sigma_y(\boldsymbol{W}_y \boldsymbol{h}^{(t)} + \boldsymbol{b}_y)$$

$$\boldsymbol{h}^{(t)} = \sigma_h\left(\boldsymbol{W}_h \boldsymbol{x}^{(t)} + \boldsymbol{U}_h \boldsymbol{y}^{(t-1)} + \boldsymbol{b}_h\right)$$
$$\boldsymbol{c}^{(t)} = \boldsymbol{c}^{(t-1)} + \boldsymbol{h}^{(t)}$$
$$\boldsymbol{y}^{(t)} = \sigma_y(\boldsymbol{c}^{(t)})$$

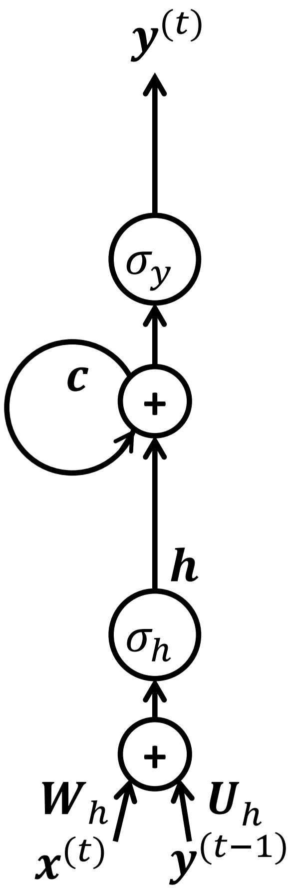
(We eliminate the linear transformation in the output)

- $\sigma_h$ is either the logistic sigmoid function or tanh function
- $\sigma_y$ is often tanh function



# Step 2: add three gates

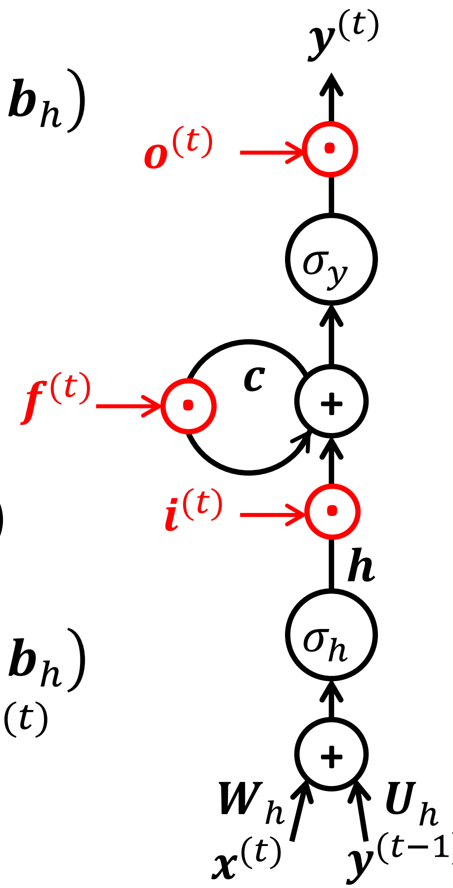
Gates are introduced to adaptively control the flow of information



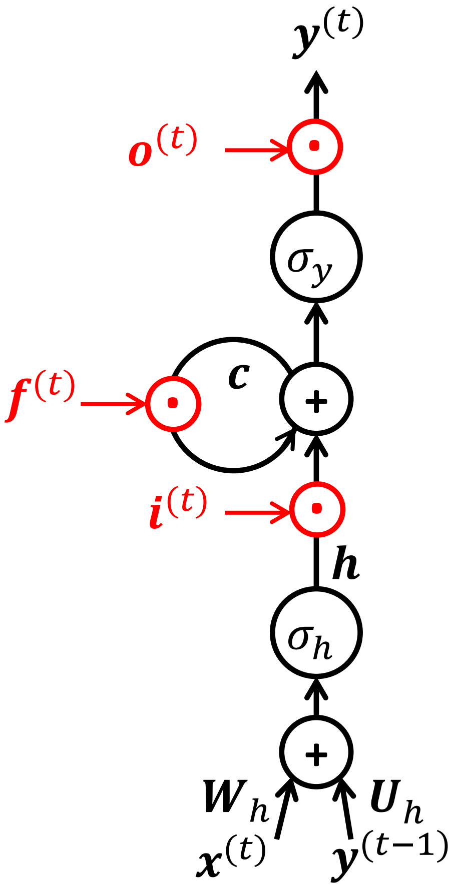

$$\boldsymbol{h}^{(t)} = \sigma_h\left(\boldsymbol{W}_h \boldsymbol{x}^{(t)} + \boldsymbol{U}_h \boldsymbol{y}^{(t-1)} + \boldsymbol{b}_h\right)$$
$$\boldsymbol{c}^{(t)} = \boldsymbol{c}^{(t-1)} + \boldsymbol{h}^{(t)}$$
$$\boldsymbol{y}^{(t)} = \sigma_y(\boldsymbol{c}^{(t)})$$

Forget gate $f^{(t)}$, input gate $i^{(t)}$,
output gate $o^{(t)}$: between (0,1)

$$\boldsymbol{h}^{(t)} = \sigma_h\left(\boldsymbol{W}_h \boldsymbol{x}^{(t)} + \boldsymbol{U}_h \boldsymbol{y}^{(t-1)} + \boldsymbol{b}_h\right)$$
$$\boldsymbol{c}^{(t)} = \boldsymbol{f}^{(t)} \odot \boldsymbol{c}^{(t-1)} + \boldsymbol{i}^{(t)} \odot \boldsymbol{h}^{(t)}$$
$$\boldsymbol{y}^{(t)} = \boldsymbol{o}^{(t)} \odot \sigma_y(\boldsymbol{c}^{(t)})$$



# What determine these gates?




$$\boldsymbol{h}^{(t)} = \sigma_h\left(\boldsymbol{W}_h \boldsymbol{x}^{(t)} + \boldsymbol{U}_h \boldsymbol{y}^{(t-1)} + \boldsymbol{b}_h\right)$$
$$\boldsymbol{c}^{(t)} = \boldsymbol{f}^{(t)} \odot \boldsymbol{c}^{(t-1)} + \boldsymbol{i}^{(t)} \odot \boldsymbol{h}^{(t)}$$
$$\boldsymbol{y}^{(t)} = \boldsymbol{o}^{(t)} \odot \sigma_y(\boldsymbol{c}^{(t)})$$

- All of the gates are determined by the input $\boldsymbol{x}^{(t)}$ and $\boldsymbol{y}^{(t-1)}$

$$\boldsymbol{f}^{(t)} = \sigma\left(\boldsymbol{W}_f \boldsymbol{x}^{(t)} + \boldsymbol{U}_f \boldsymbol{y}^{(t-1)} + \boldsymbol{b}_f\right)$$
$$\boldsymbol{i}^{(t)} = \sigma\left(\boldsymbol{W}_i \boldsymbol{x}^{(t)} + \boldsymbol{U}_i \boldsymbol{y}^{(t-1)} + \boldsymbol{b}_i\right)$$
$$\boldsymbol{o}^{(t)} = \sigma\left(\boldsymbol{W}_o \boldsymbol{x}^{(t)} + \boldsymbol{U}_o \boldsymbol{y}^{(t-1)} + \boldsymbol{b}_o\right)$$

  where $\sigma$ is the logistic sigmoid function
- Sometimes, they are also determined by $\boldsymbol{c}^{(t)}$ and $\boldsymbol{c}^{(t-1)}$: *peepholes*

Note: sometimes, the output $\boldsymbol{y}$ is also called *hidden state of LSTM*, especially when LSTM is integrated into a larger system.

# Gated recurrent unit (GRU)

- In the Elman network, the hidden units $\boldsymbol{h}$ are used to capture the history information

$$\boldsymbol{h}^{(t)} = \sigma_h\left(\boldsymbol{W}_h \boldsymbol{x}^{(t)} + \boldsymbol{U}_h \boldsymbol{h}^{(t-1)} + \boldsymbol{b}_h\right)$$

- In an LSTM cell without gates, a new vector $\boldsymbol{c}$ is introduced for this purpose

$$\boldsymbol{h}^{(t)} = \sigma_h\left(\boldsymbol{W}_h \boldsymbol{x}^{(t)} + \boldsymbol{U}_h \boldsymbol{y}^{(t-1)} + \boldsymbol{b}_h\right)$$

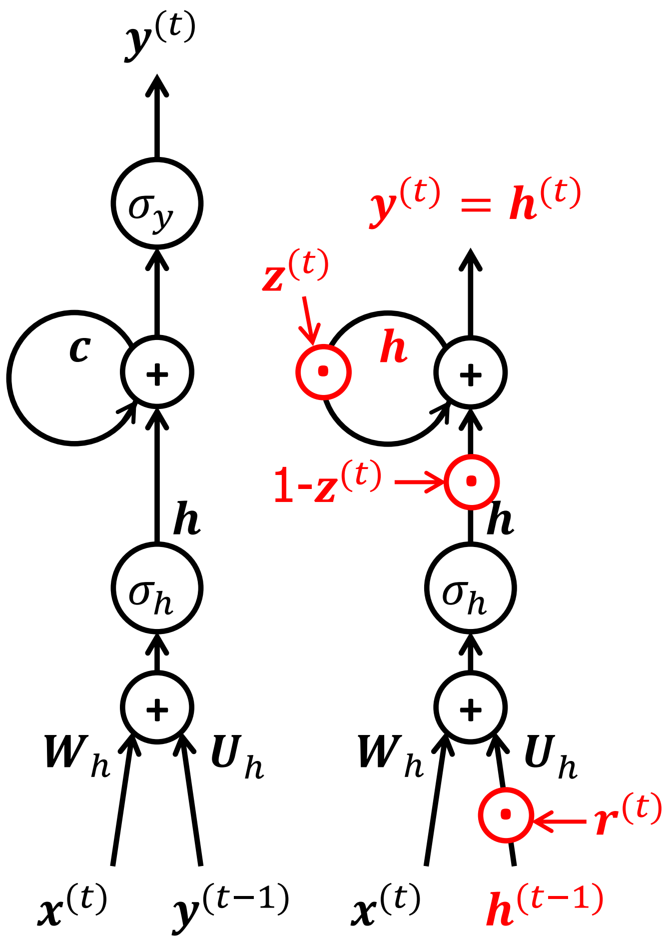
$$\boldsymbol{c}^{(t)} = \boldsymbol{c}^{(t-1)} + \boldsymbol{h}^{(t)}$$

- Why not use $\boldsymbol{h}$ directly?
- This is the 1st idea of GRU

$$\boldsymbol{h}^{(t)} = \boldsymbol{z}^{(t)} \odot \boldsymbol{h}^{(t-1)} + \left(1 - \boldsymbol{z}^{(t)}\right) \odot \widetilde{\boldsymbol{h}}^{(t)}$$

where $\boldsymbol{z}^{(t)} \in (0,1)$ and

$$\widetilde{\boldsymbol{h}}^{(t)} = \sigma_h\left(\boldsymbol{W}_h \boldsymbol{x}^{(t)} + \boldsymbol{U}_h \boldsymbol{h}^{(t-1)} + \boldsymbol{b}_h\right)$$



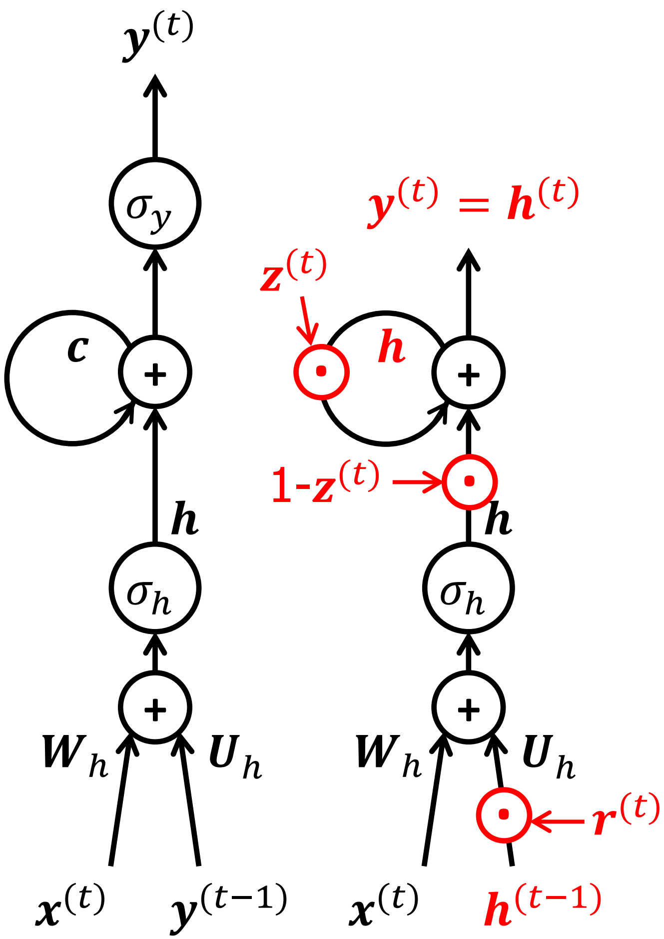

LSTM cell w/o gates GRU

# Gated recurrent unit (GRU)

- The 2nd idea of GRU
  - Let output be equal to hidden states: $\boldsymbol{y}^{(t)} = \boldsymbol{h}^{(t)}$
- The 3rd idea of GRU
  - Use a gate to modulate the recurrent input

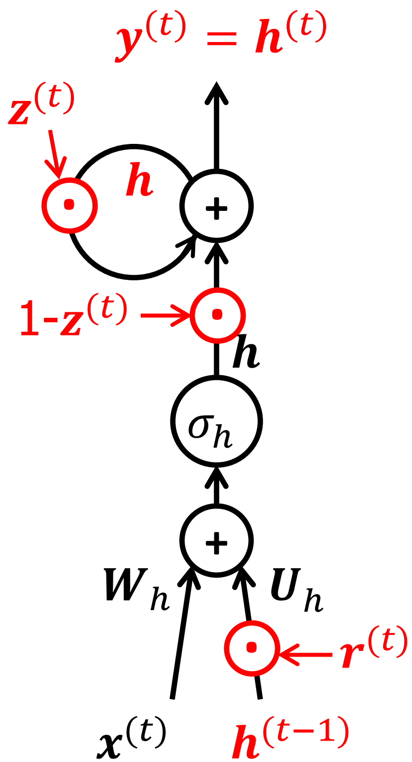
$$\tilde{\boldsymbol{h}}^{(t)} = \sigma_h\left(\boldsymbol{W}_h \boldsymbol{x}^{(t)} + \boldsymbol{U}_h\left(\boldsymbol{r}^{(t)} \odot \boldsymbol{h}^{(t-1)}\right) + \boldsymbol{b}_h\right)$$

- The gates depend on input and hidden states
  - Update gate $\boldsymbol{z}^{(t)} = \sigma\left(\boldsymbol{W}_z \boldsymbol{x}^{(t)} + \boldsymbol{U}_z \boldsymbol{h}^{(t-1)} + \boldsymbol{b}_z\right)$
  - Reset gate $\boldsymbol{r}^{(t)} = \sigma\left(\boldsymbol{W}_r \boldsymbol{x}^{(t)} + \boldsymbol{U}_r \boldsymbol{h}^{(t-1)} + \boldsymbol{b}_r\right)$



LSTM cell w/o gates GRU

# GRU in summary




- Dynamic equations

$$\boldsymbol{h}^{(t)} = \boldsymbol{z}^{(t)} \odot \boldsymbol{h}^{(t-1)} + \left(1 - \boldsymbol{z}^{(t)}\right) \odot \widetilde{\boldsymbol{h}}^{(t)}$$
$$\widetilde{\boldsymbol{h}}^{(t)} = \sigma_h\left(\boldsymbol{W}_h \boldsymbol{x}^{(t)} + \boldsymbol{U}_h\left(\boldsymbol{r}^{(t)} \odot \boldsymbol{h}^{(t-1)}\right) + \boldsymbol{b}_h\right)$$

  where $\sigma_h$ is either the logistic sigmoid function or tanh function

- The gates

$$\boldsymbol{z}^{(t)} = \sigma\left(\boldsymbol{W}_z \boldsymbol{x}^{(t)} + \boldsymbol{U}_z \boldsymbol{h}^{(t-1)} + \boldsymbol{b}_z\right)$$
$$\boldsymbol{r}^{(t)} = \sigma\left(\boldsymbol{W}_r \boldsymbol{x}^{(t)} + \boldsymbol{U}_r \boldsymbol{h}^{(t-1)} + \boldsymbol{b}_r\right)$$

  $\sigma$ is the logistic sigmoid function

There are many other variants of LSTM and GRU, but none of them would clearly beat both of these across a wide range of tasks (Greff et al., 2015)

# Summary

- Dynamic systems
  - Autonomous and non-autonomous
- Simple RNNs
  - Jordan network and Elman network
  - BPTT and teacher forcing
  - Bidirectional RNN
  - Deep RNN
- Recurrent CNN
  - CNN + recurrent connections
- Gated RNNs
  - LSTM and GRU

# Further reading

- Goodfellow, Bengio and Courville, 2016
  Deep Learning, MIT Press, Chapters 10
- Understanding LSTM networks
  http://colah.github.io/posts/2015-08-Understanding-LSTMs/

# Prepare for the next lecture

- Form groups of 2 and every group prepares a 5-minute presentation with slides for the following paper
  - Santurkar, Tsipras, Ilyas, Madry (2018) How does batch normalization help optimization? NeurIPS

Nobody prepared for this paper today. So I'll randomly select a group to present the paper in the next week (April 25)!

April 18