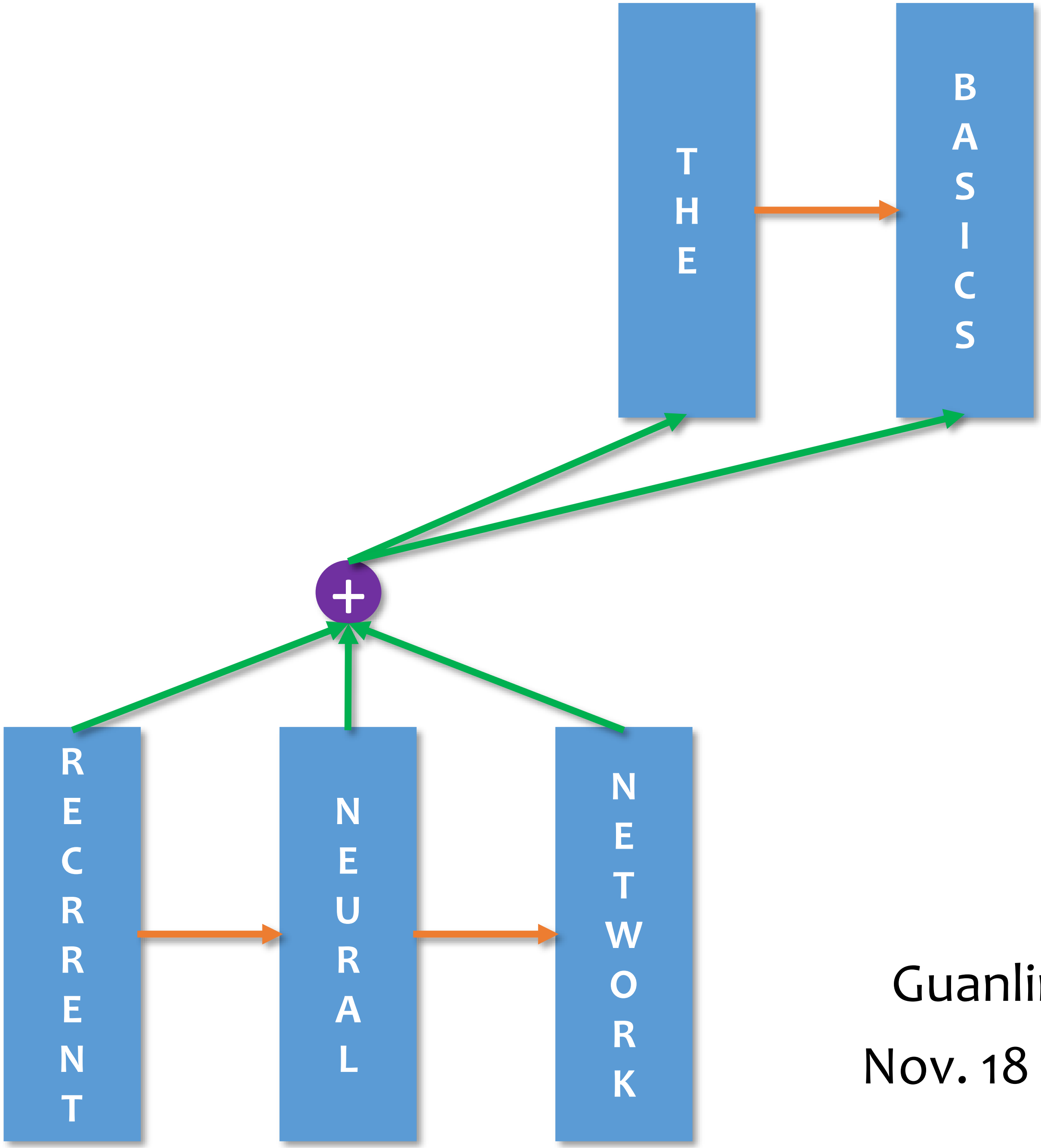


Guanlin Li

Nov. 18 2016

Outline

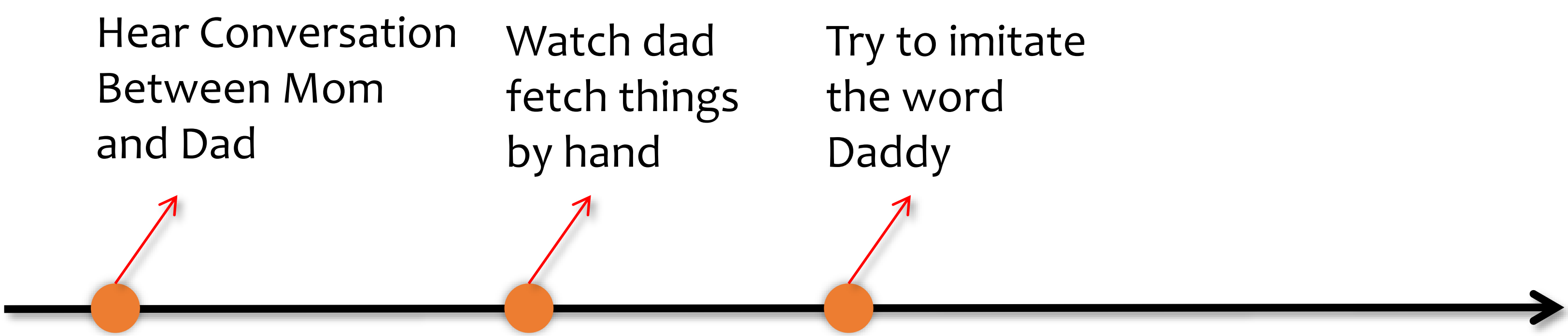
- Sequence with Order
  - Unfolding Computational Graph
- Recurrent Neural Network
- Recursive Neural Network
- Challenge of Long-Term Dependencies
  - Long Short-term Memory Unit
  - Gated Recurrent Unit
- Explicit Memory
  - Memory Network (Weston et al)
  - Neural Turing Machine (Graves et al)

Outline

- Sequence with Order
  - Unfolding Computational Graph
- Recurrent Neural Network
- Recursive Neural Network
- Challenge of Long-Term Dependencies
  - Long Short-term Memory Unit
  - Gated Recurrent Unit
- Explicit Memory
  - Memory Network (Weston et al)
  - Neural Turing Machine (Graves et al)

Sequence with Order

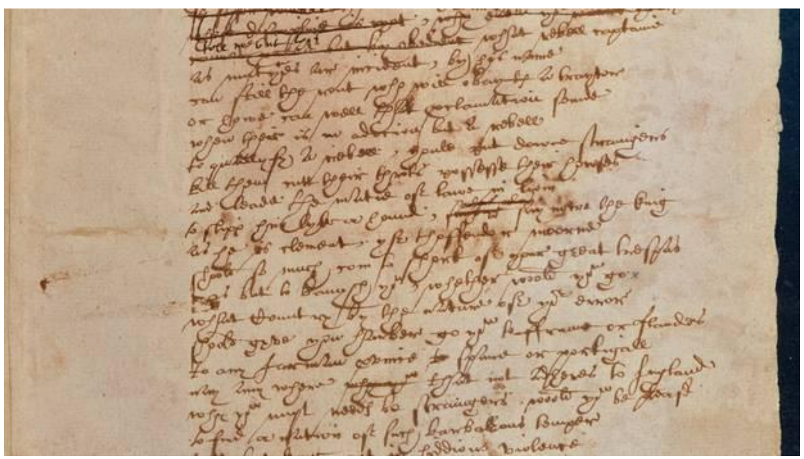
- Natural Order
  - All kinds of datum that is **generated/produced** through **time**.

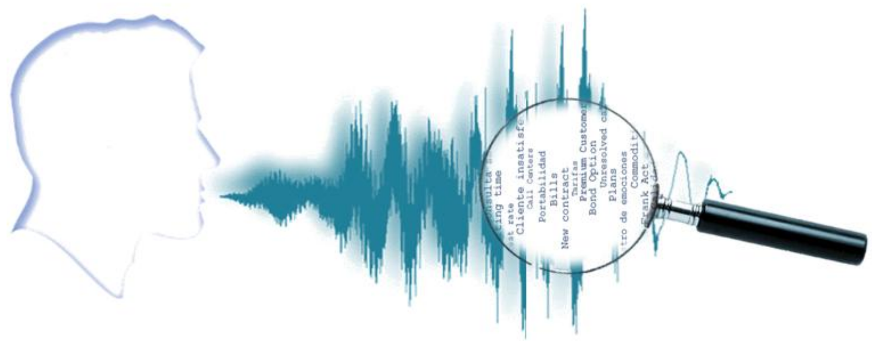


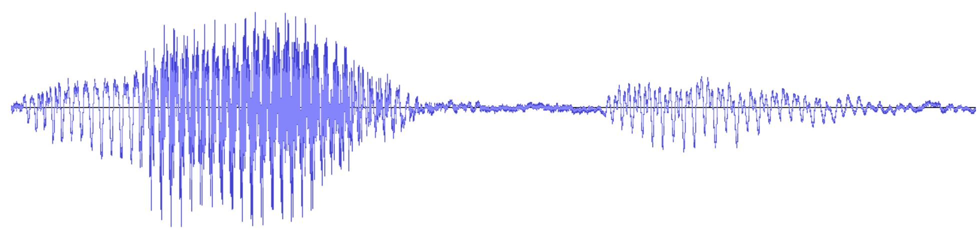
Timeline of a Baby

Sequence with Order

- Examples of Natural Order
  - Language data (Text, speech)





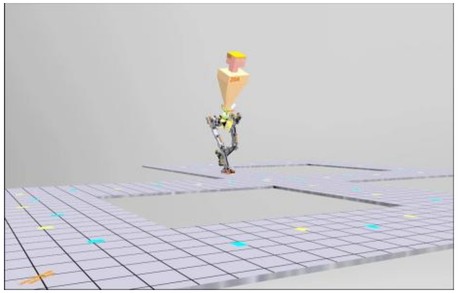
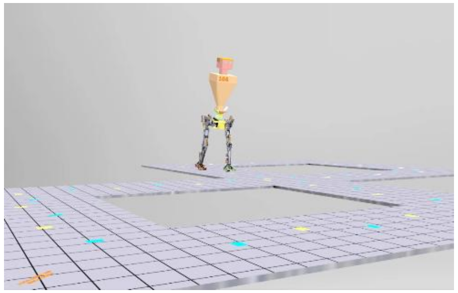
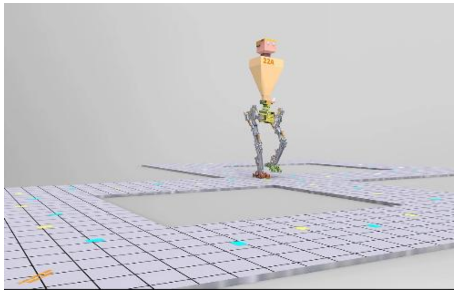
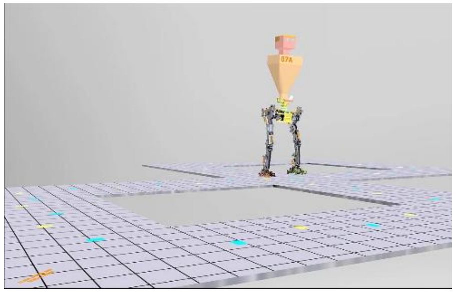
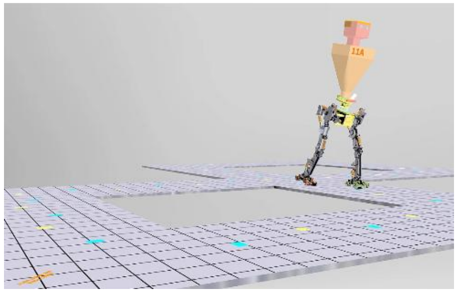
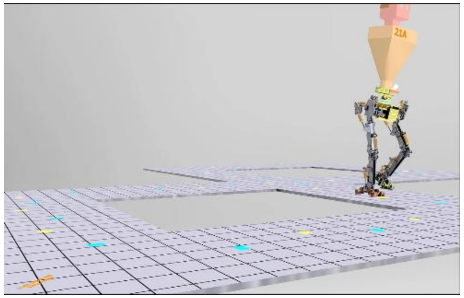


Sequence with Order

- Examples of Natural Order
  - Language data (Text, speech)
  - Financial Market (Stock Prize)

Sequence with Order

- Examples of Natural Order
  - Language data (Text, speech)
  - Financial Market (Stock Prize)
  - Behavior/action sequence of robots

Property of Sequence

- Infinity length
  - We must choose design the capacity to deal with it
  - *E.g. till the end of time/universe*
- Nondeterministic length
  - Actually, we sample sequences with different length to deal with, e.g.
    - This year's stock movement
    - This commercial prize before Spring festival
    - Actions taken by the robot of achieving this NL instruction

Sequence as random process

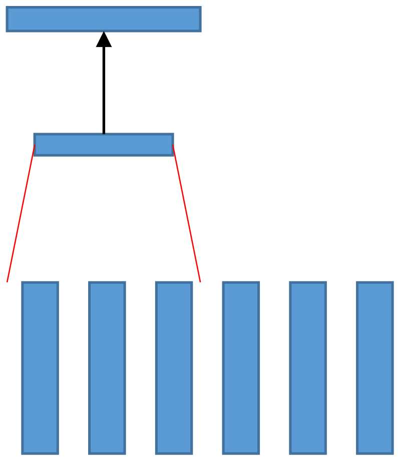
- Formally, sequence data can be described as a sequence of random variables
  - $X_1, X_2, X_3, X_4, \dots, X_t, X_{t+1}, X_{t+2}, \dots$
- X_t can be a random scalar
  - e.g. Stock prize
- X_t can be a random vector
  - e.g. location of the robot in 3D space
- X_t can be a random matrix
  - e.g. video stream

Random Process Definition

A *random process or stochastic process* is a collection of random variables $\boldsymbol{X} = \{X_t : t \in T\}$, defined on an underlying probability space $(\Omega, \mathcal{F}, P)$, where X_t takes value in a state space S for each t in an index set T.

Who can deal with sequence data?

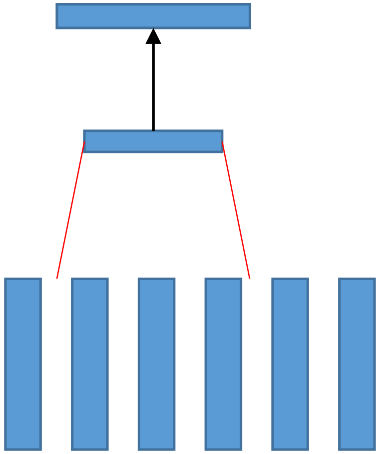
- Take language modelling as example
  - Remember in Bengio 03, they use a FFNN to sliding on a sequence of words with One-hot representation



$P(x_t|x_{t-3}, x_{t-2}, x_{t-1})$

Who can deal with sequence data?

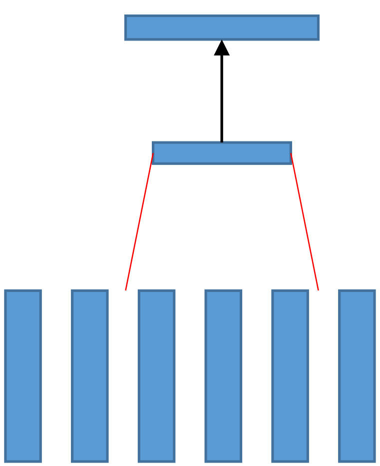
- Take language modelling as example
  - Remember in Bengio 03, they use a FFNN to sliding on a sequence of words with One-hot representation



$P(x_t|x_{t-3}, x_{t-2}, x_{t-1})$

Who can deal with sequence data?

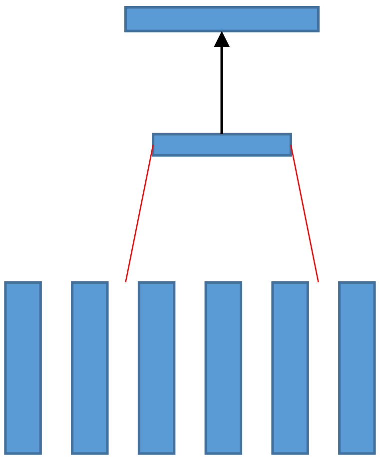
- Take language modelling as example
  - Remember in Bengio 03, they use a FFNN to sliding on a sequence of words with One-hot representation



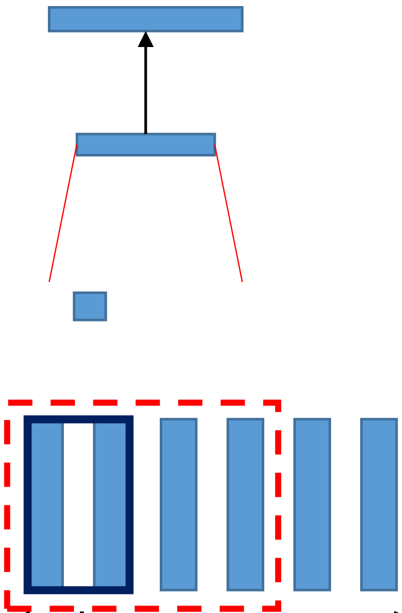
$P(x_t|x_{t-3}, x_{t-2}, x_{t-1})$

Who can deal with sequence data?

- Take language modelling as example
  - Remember in Bengio 03, they use a FFNN to sliding on a sequence of words with One-hot representation
  - CNN can also be used, with multiple conv operation at one time step



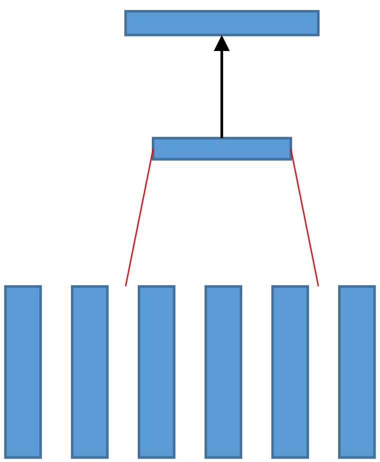
$P(x_t|x_{t-3}, x_{t-2}, x_{t-1})$



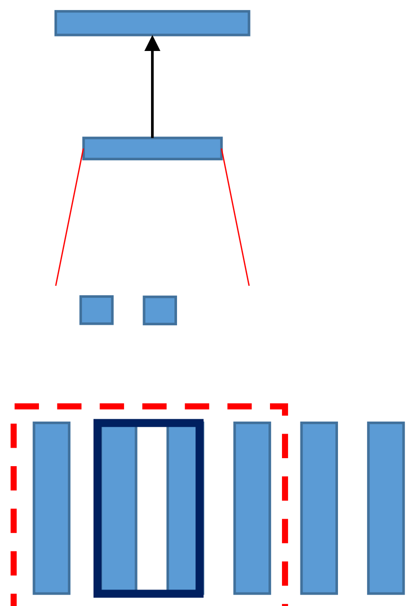
$P(x_t|x_{t-3}, x_{t-2}, x_{t-1})$

Who can deal with sequence data?

- Take language modelling as example
  - Remember in Bengio 03, they use a FFNN to sliding on a sequence of words with One-hot representation
  - CNN can also be used , with multiple conv operation at one time step



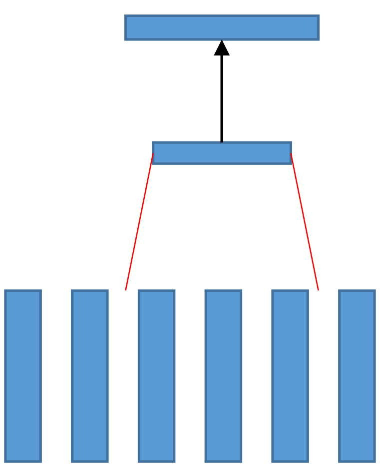
$P(x_t | x_{t-3}, x_{t-2}, x_{t-1})$



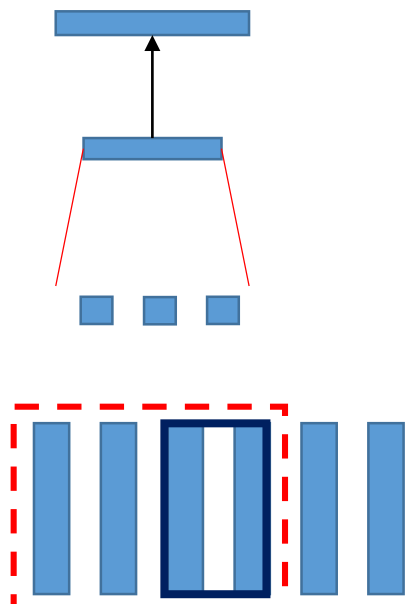
$P(x_t | x_{t-3}, x_{t-2}, x_{t-1})$

Who can deal with sequence data?

- Take language modelling as example
  - Remember in Bengio 03, they use a FFNN to sliding on a sequence of words with One-hot representation
  - CNN can also be used , with multiple conv operation at one time step



$P(x_t | x_{t-3}, x_{t-2}, x_{t-1})$



$P(x_t | x_{t-3}, x_{t-2}, x_{t-1})$

Deficiency

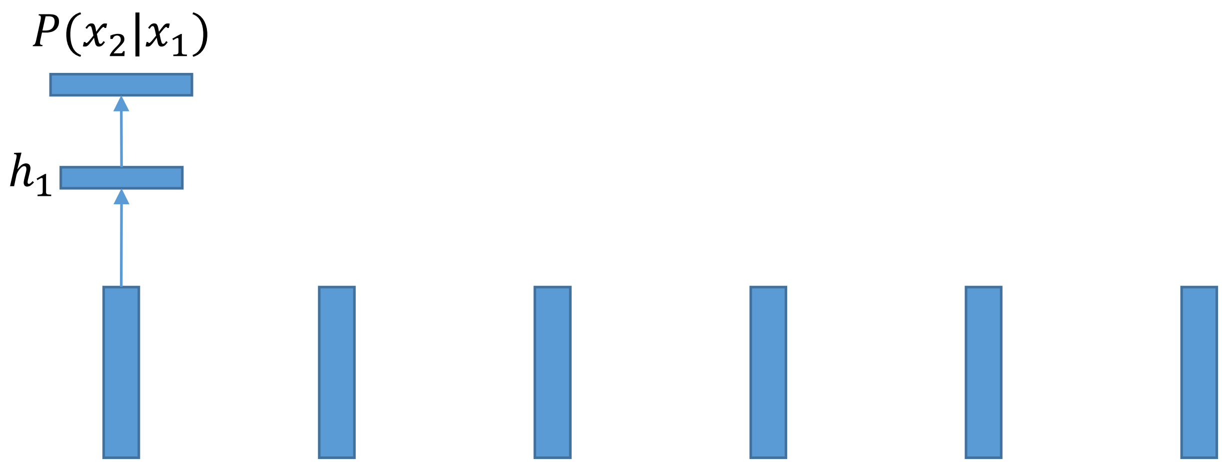
- Still limited context size

- But in practice, we do not know what is a proper context size

- Extend context size will increase the number of parameters & computation

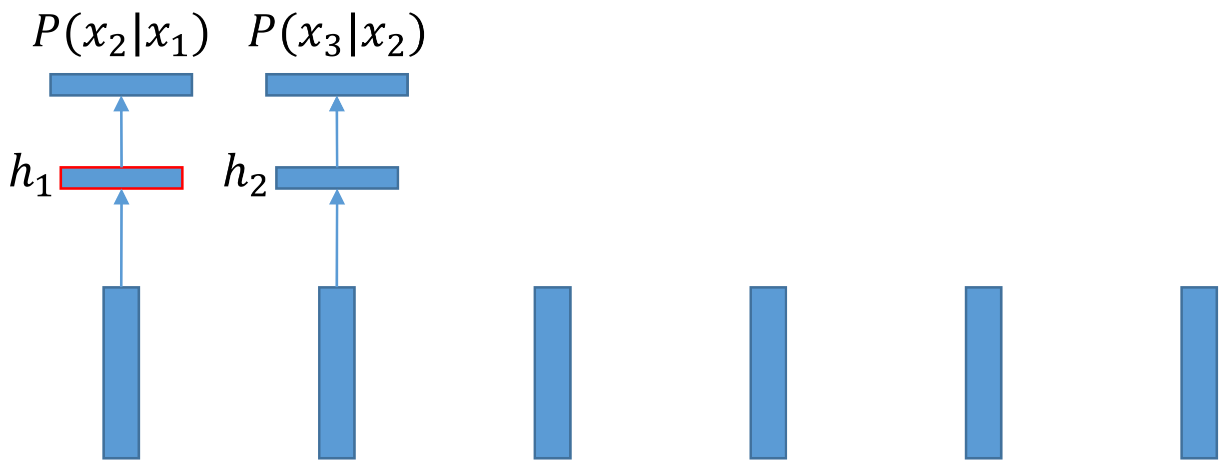
Feedback Connection

- Computation graph of 1-gram FFNN language model



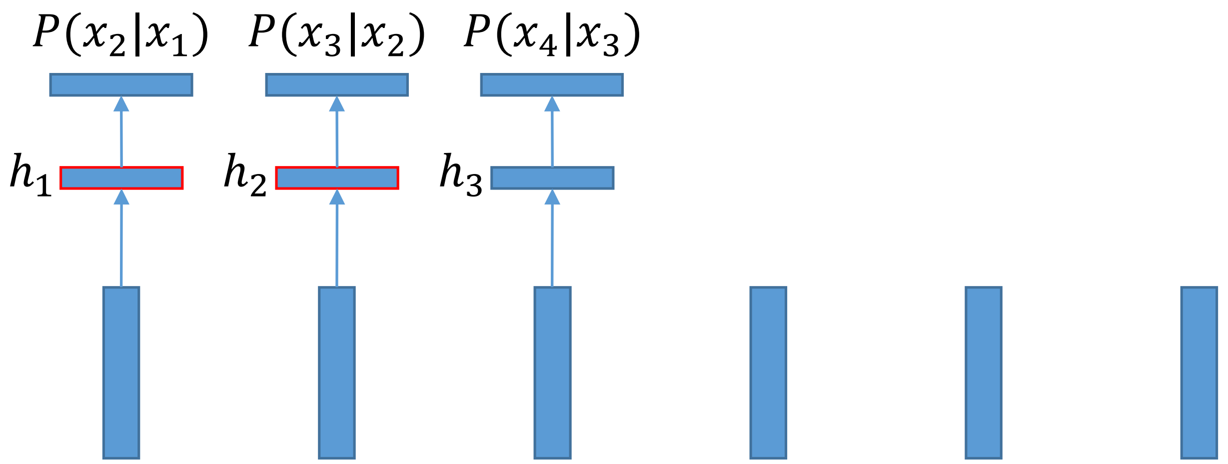
Feedback Connection

- Computation graph of 1-gram FFNN language model



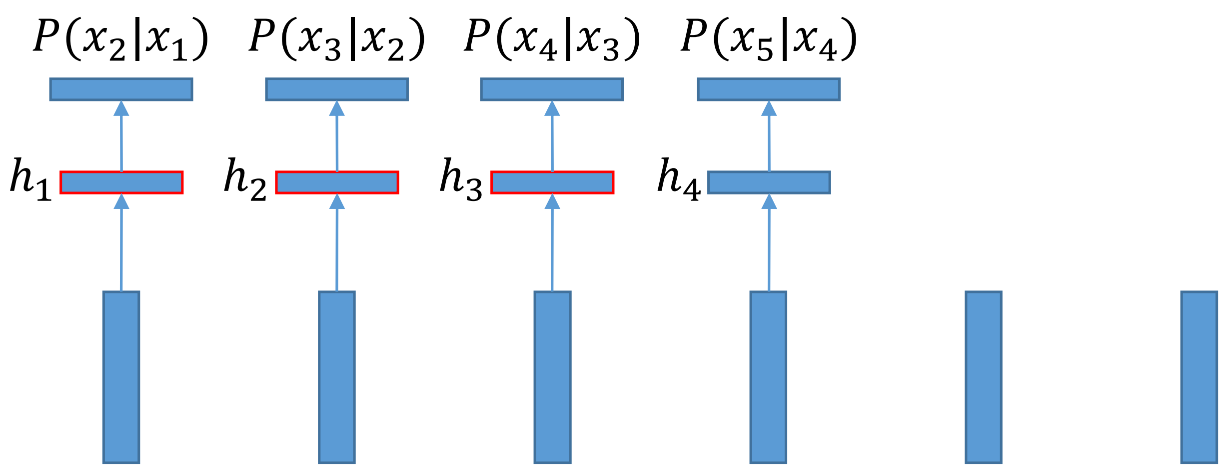
Feedback Connection

- Computation graph of 1-gram FFNN language model



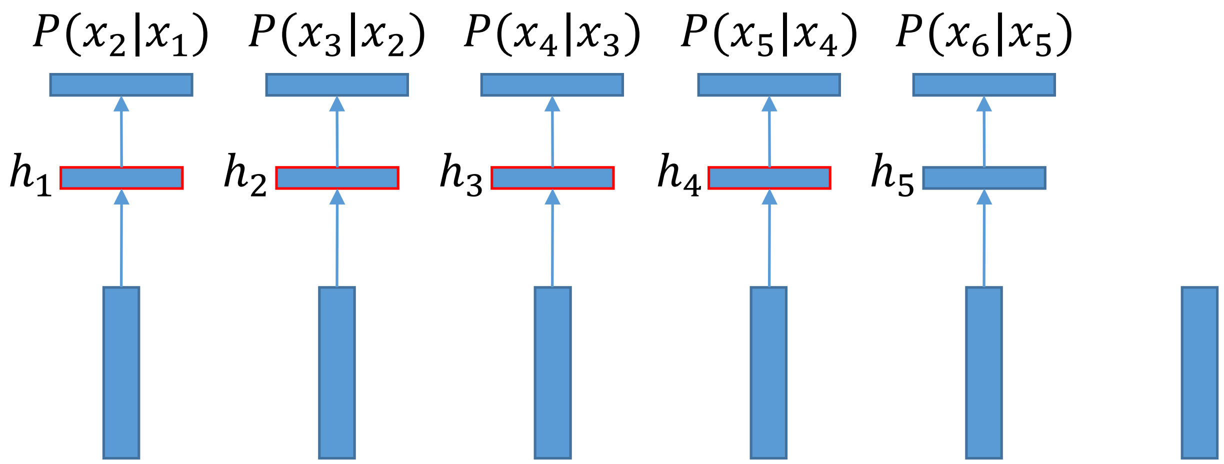
Feedback Connection

- Computation graph of 1-gram FFNN language model



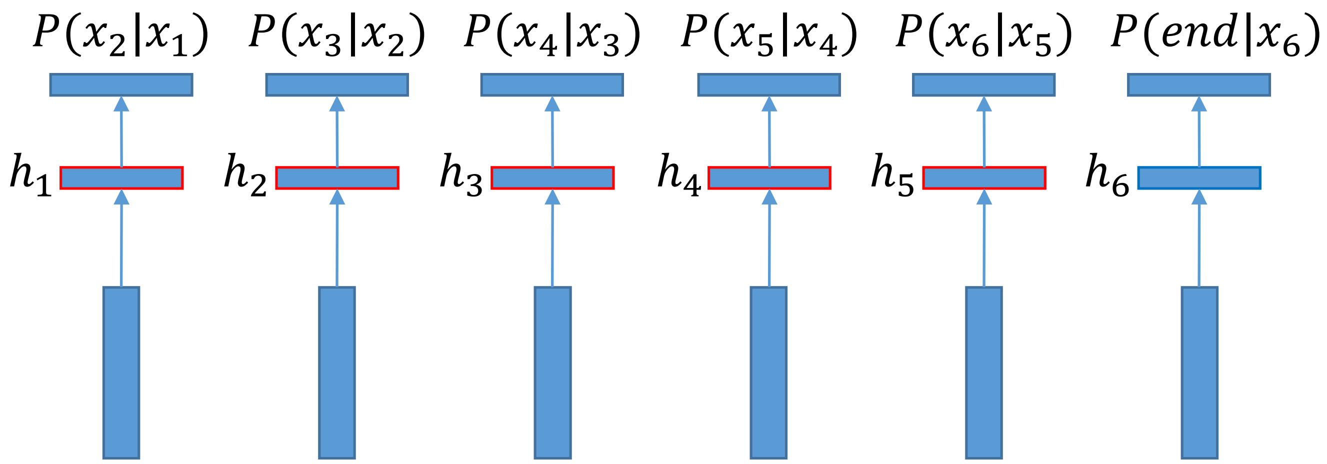
Feedback Connection

- Computation graph of 1-gram FFNN language model



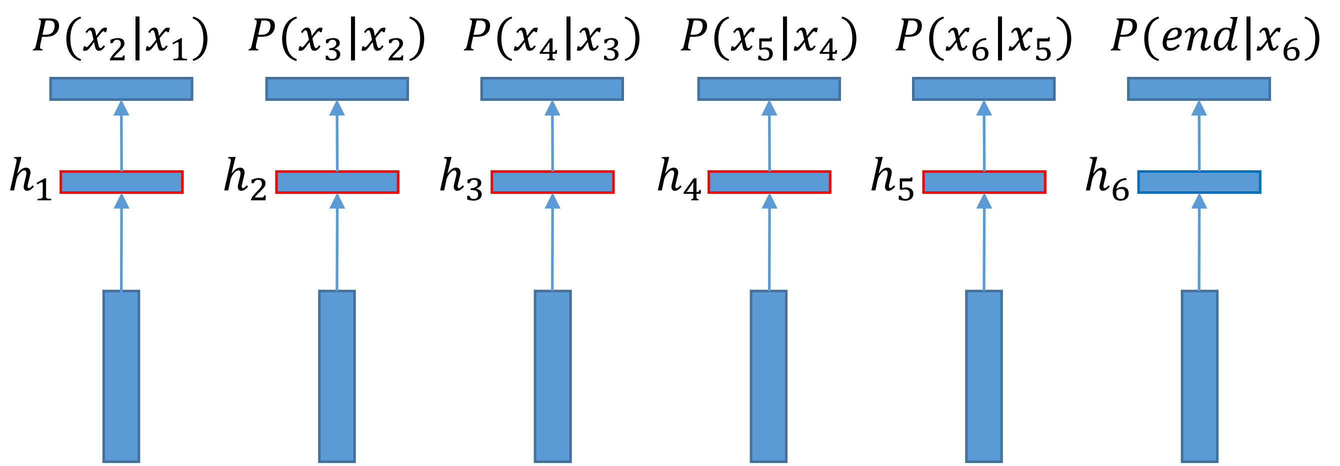
Feedback Connection

- Computation graph of 1-gram FFNN language model
  - We find, at each time step, the hidden h_t is totally recomputed from new input



Feedback Connection

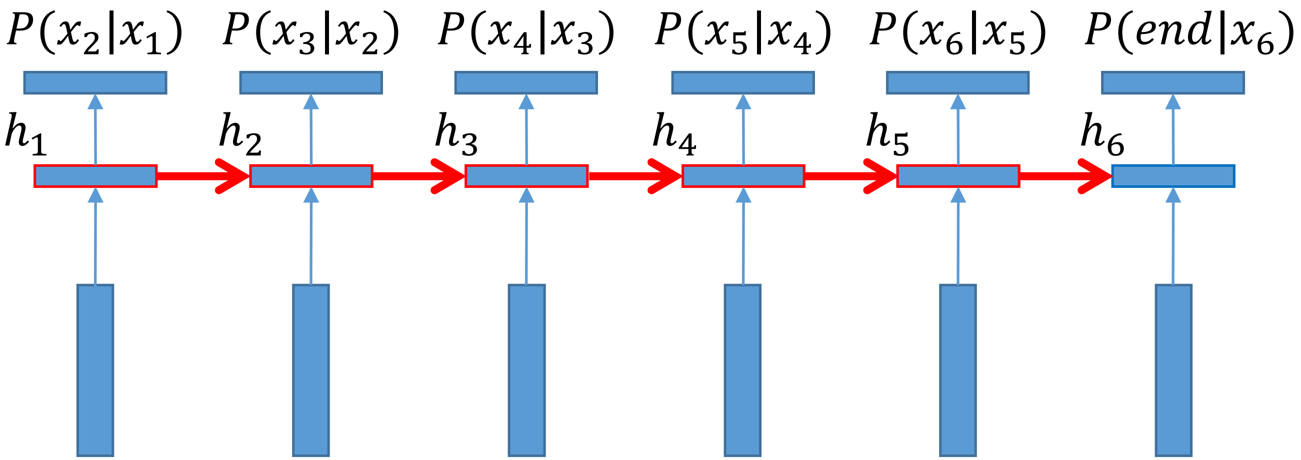
- Computation graph of 1-gram FFNN language model
  - We find, at each time step, the hidden h_t is recomputed from new input
  - And each hidden h_t contain the information about the corresponding time step



What if we can use the hidden information from previous time step?

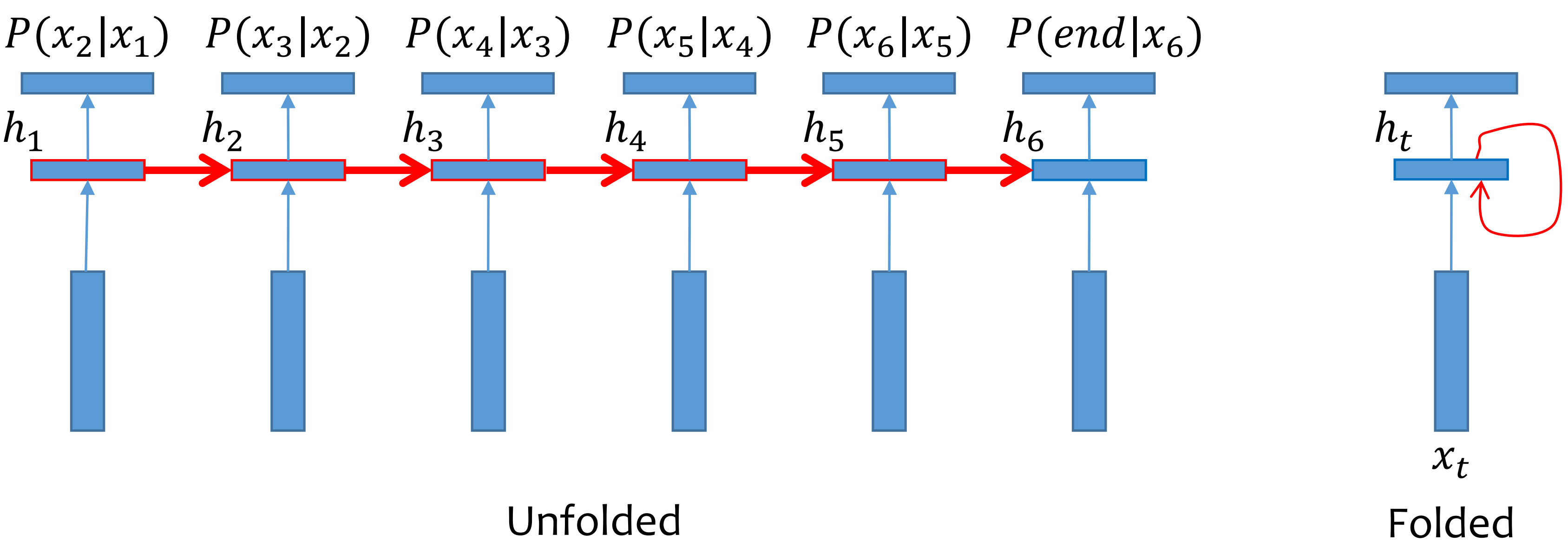
Feedback Connection

- Feedback connection
  - It is like Forward connection in the unfolded computational graph
  - h_t is computed from both *current input* and *previous hidden* h_{t-1}



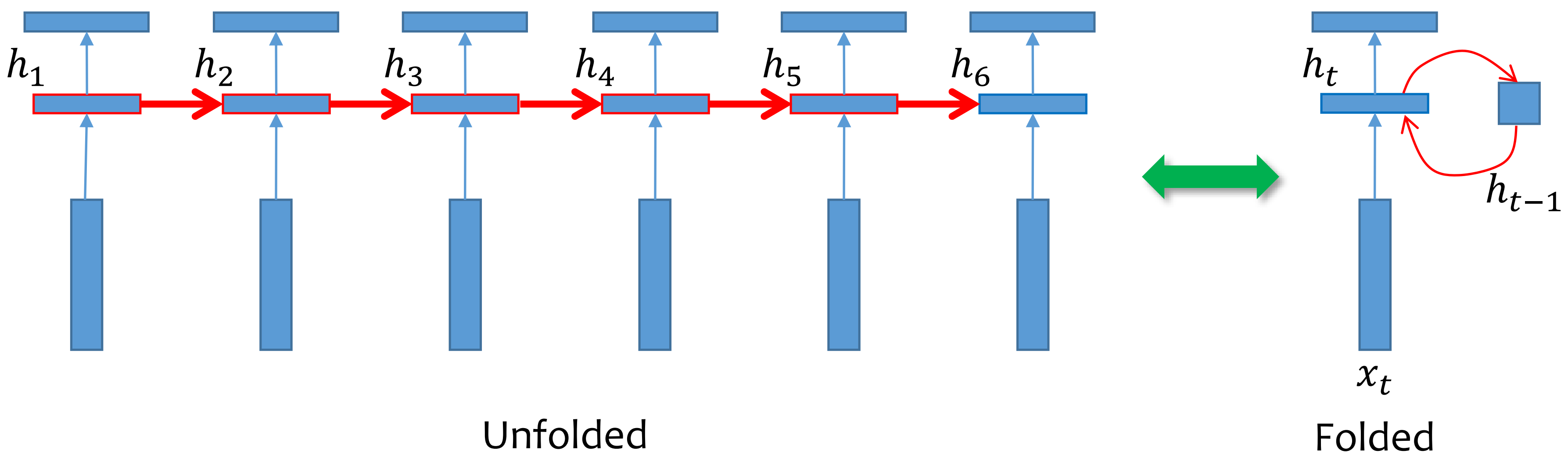
Feedback Connection

- Feedback connection
  - It is like Forward connection in the unfolded computational graph
  - h_t is computed from both *current input* and *previous hidden* h_{t-1}



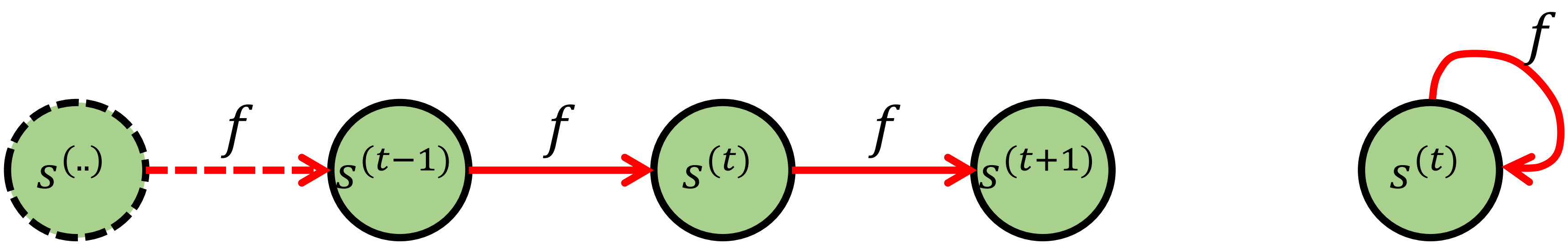
Feedback Connection

- Feedback connection
  - By connecting the previous hidden, the current hidden actually has access to arbitrarily long history
  - Since the connection opens a pathway along all time steps



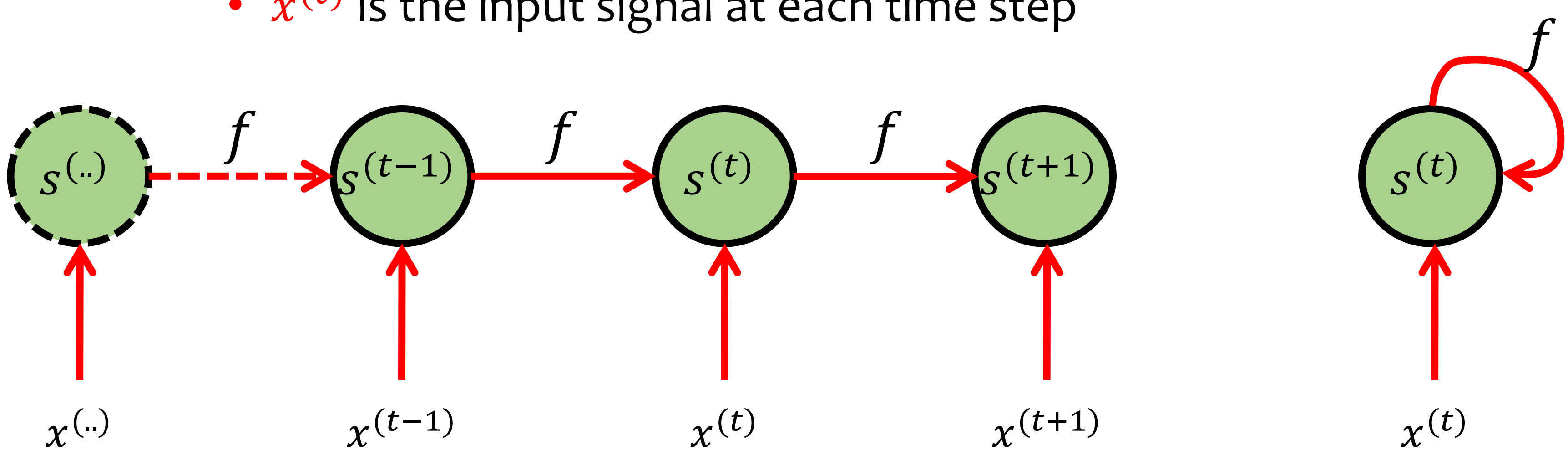
A Dynamic System View

- Classic dynamic system
  - $S^{(t)} = f(s^{(t-1)}; \theta)$
    - $s^{(t)}$ is the state of the system
    - θ is the parameter that control the behavior of f
    - f is the state transition function
- Internal state will change at each time step
  - Unfolded state transition graph



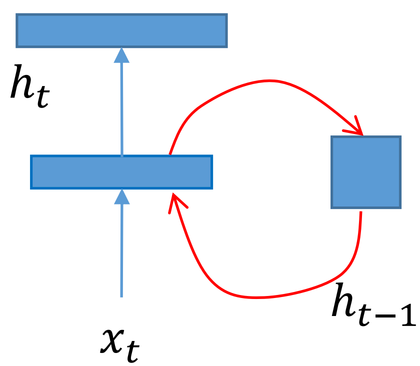
A Dynamic System View

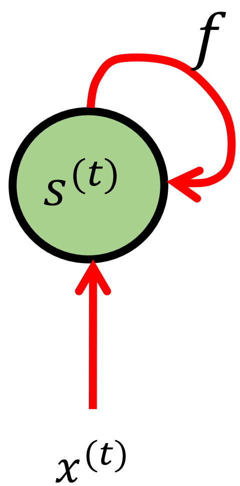
- Dynamic system driven by external signal
  - $S^{(t)} = f(s^{(t-1)}, x^{(t)}; \theta)$
    - $s^{(t)}$ is the state of the system
    - θ is the parameter that control the behavior of f
    - f is the state transition function
    - $x^{(t)}$ is the input signal at each time step



Parameter Sharing

- At each time step, it is the same network that does the computation over input and previous hidden, which has the same parameters
- This is called parameter sharing through time





The View of Information Flow

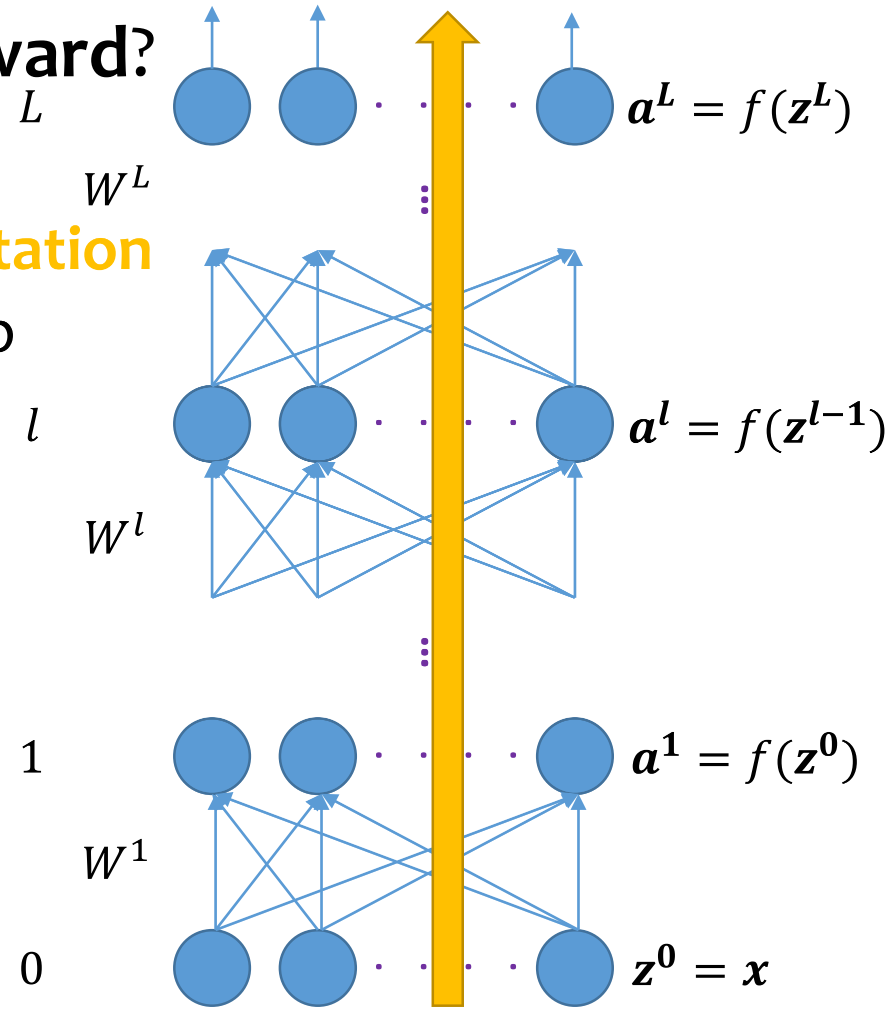
- A sequence is an information carrier
  - $x_1, x_2, x_3, \dots, x_t$
  - Assume: each time step, the variable represent the smallest granularity of processing

- Information Flow is the flow of computation (can be more general) explicitly or implicitly between information carriers goes all the way to the decision or prediction end

Remember in our first lecture

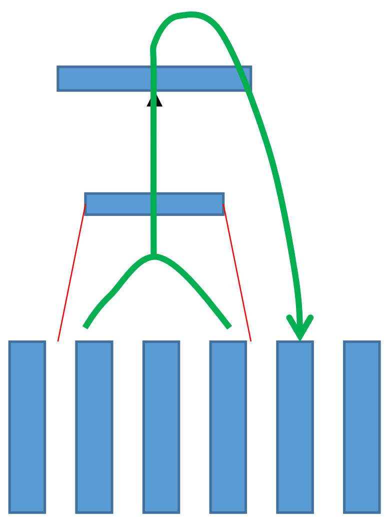
Architecture

- Why call it **Feed Forward**?
  - **Information/Computation**
  - flow from bottom up

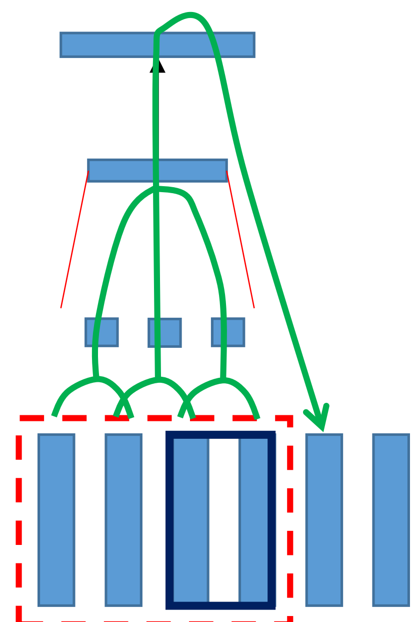


Information Flow in Language Models

- In FFNN and CNN language models



$P(x_t | x_{t-3}, x_{t-2}, x_{t-1})$



$P(x_t | x_{t-3}, x_{t-2}, x_{t-1})$

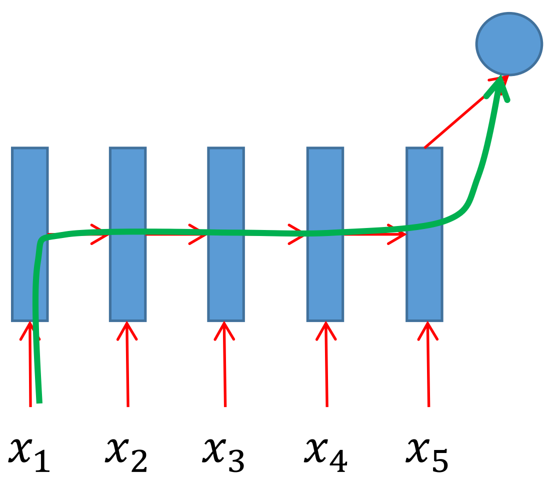
Information Flow

- More generally, as a human, you always use the *current* information in your *induced Working Memory* to make decisions or take actions, e.g.
  - Reasoning
  - Classifying
  - Calculating
  - Creative thinking etc.
- So all your *past experience* stored in episode memory/long-term memory will partially form the information flow during your *decision making*.

Information Carrier Formally

- Number system is the mostly used measure system
  - Scalar information carrier
  - Vector information carrier
  - Matrix/tensor information carrier

No doubt, info is increasing



Bonus: Mo Yu's PhD Work

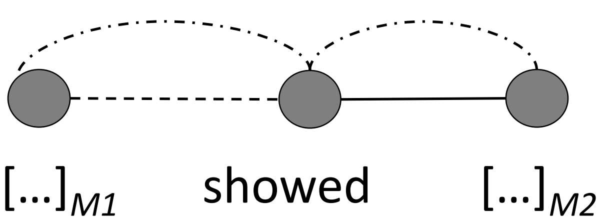
On Path = True	1
Is Head of M1 = False	0
In between = True	**1**
Is Head of M2 = False	0
Before M1= False	0
Before M2 = True	1
…	

&

-.5	.3	.8	.7

word embedding of "showed"

Lexical feature



$f_{wi} \otimes e_{wi}$

f_{wi}	f_2=(w_i is in between entities?)			
1	-.5	.3	.8	.7
0	0	0	0	0
1	**-.5**	**.3**	**.8**	**.7**
0	0	0	0	0
0	0	0	0	0
1	-.5	.3	.8	.7

-.5	.3	.8	.7

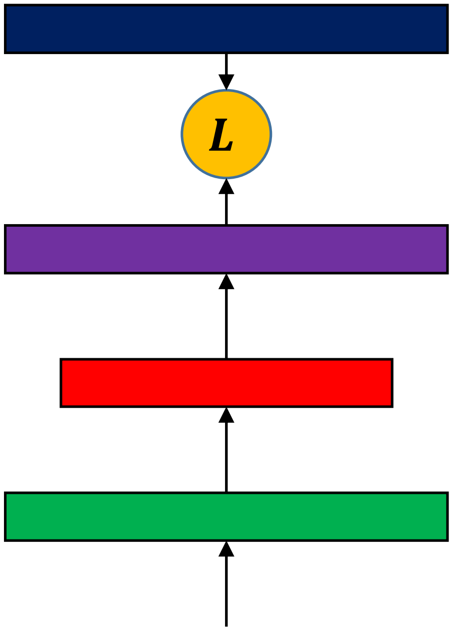
e_{wi} (w_i= "*showed*")

Outline

- Sequence with Order
  - Unfolding Computational Graph
- Recurrent Neural Network
- Recursive Neural Network
- Challenge of Long-Term Dependencies
  - Long Short-term Memory Unit
  - Gated Recurrent Unit
- Explicit Memory
  - Memory Network (Weston et al)
  - Neural Turing Machine (Graves et al)

Recurrent Neural Network

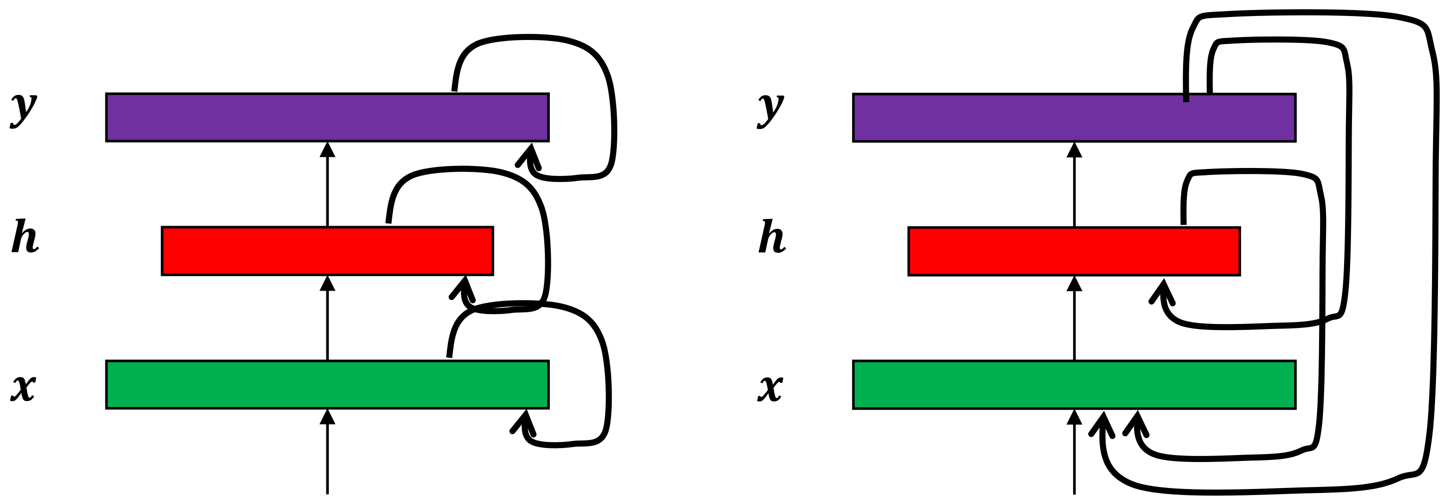
- Firstly, let us review general considerations of NN architecture
  - **Input**
  - **Hidden** (One-layer)
  - **Output**
  - **Loss**



- So does RNN!

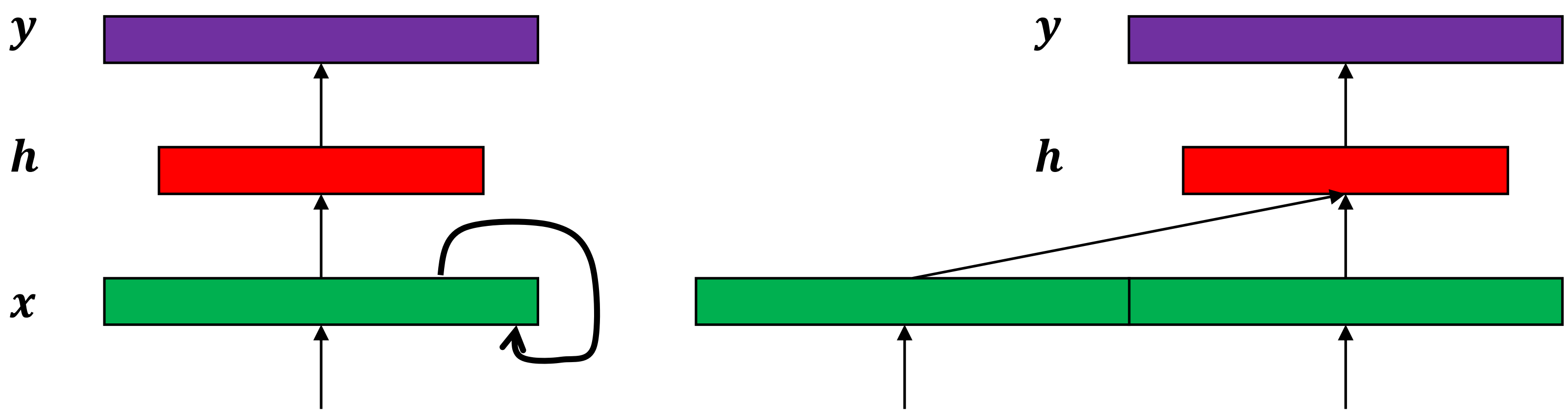
Recurrent Neural Network

- Naïve RNN
  - Where should the recurrence be?



Recurrent Neural Network

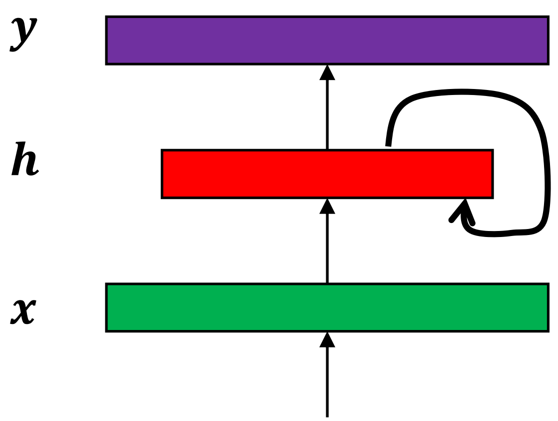
- Naïve RNN
  - Where should the recurrence be?



- This one is like a FFNN which has a window size of 2 over the input sequence.
- Since input layer involves no computation, just concatenation.

Recurrent Neural Network

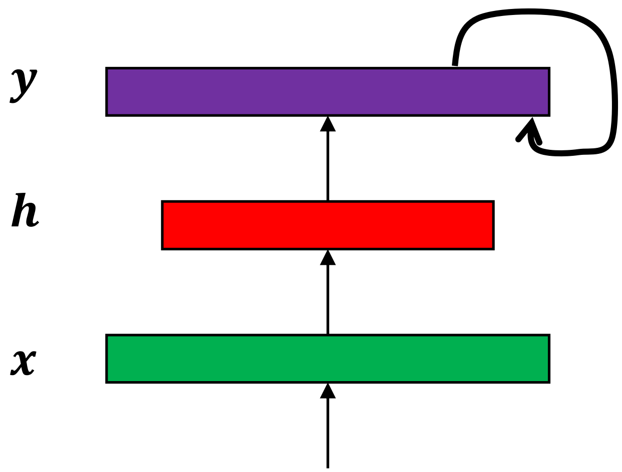
- Naïve RNN
  - Where should the recurrence be?



- This one is the standard RNN we will focus our discussion on.
- It is called *hidden recurrence*.

Recurrent Neural Network

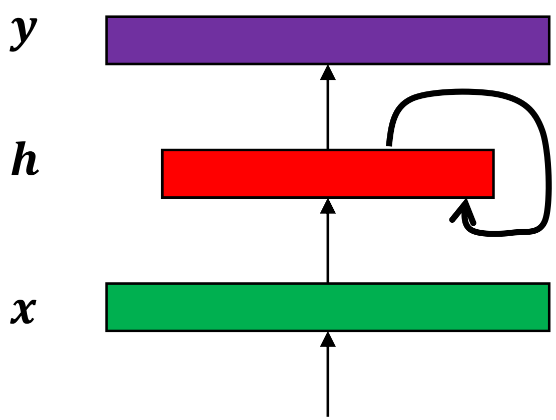
- Naïve RNN
  - Where should the recurrence be?

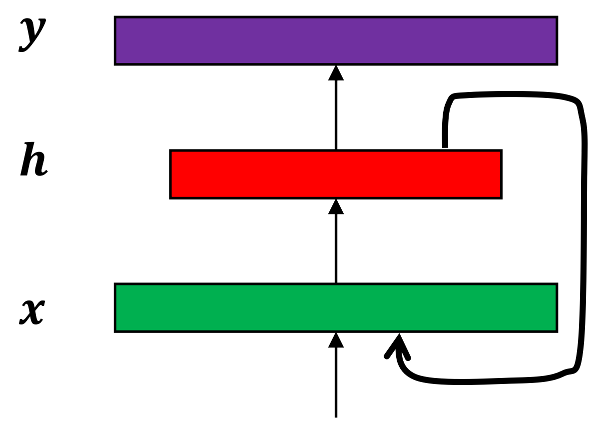


This one might be applied to those type of sequences where the output directly influence the future output.

Recurrent Neural Network

- Naïve RNN
  - Where should the recurrence be?

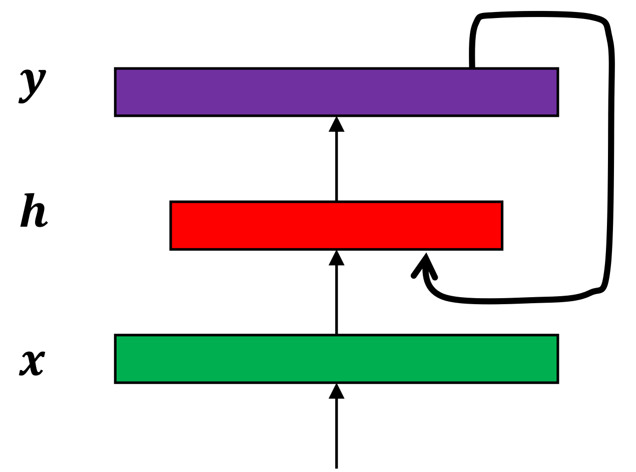




This one is equal to this one, why?

Recurrent Neural Network

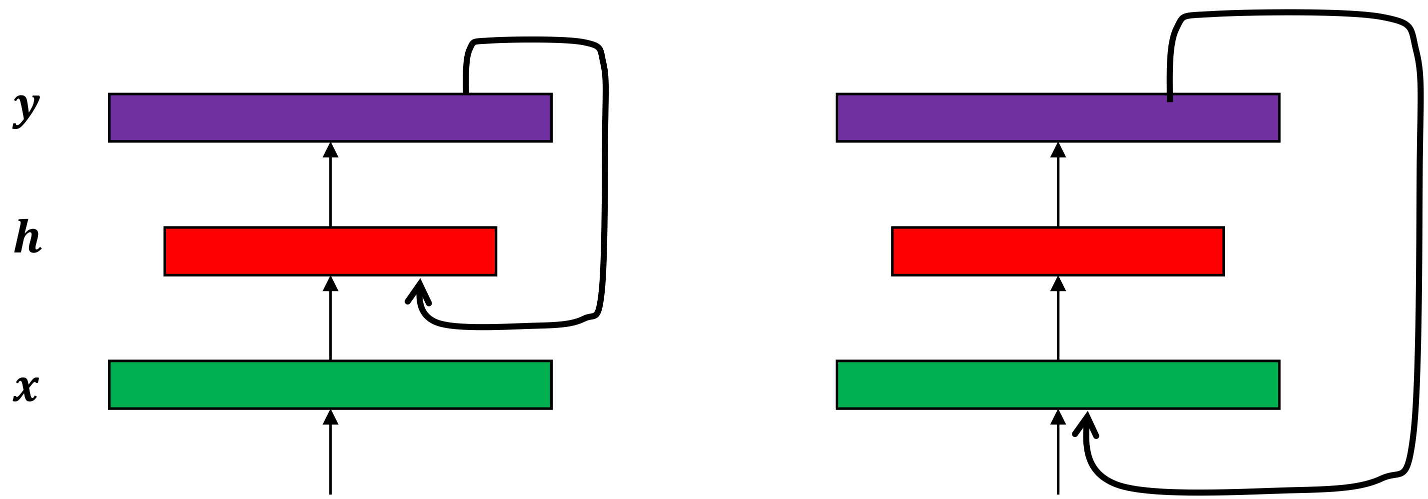
- Naïve RNN
  - Where should the recurrence be?



Output back to hidden, this is called output recurrence.

Recurrent Neural Network

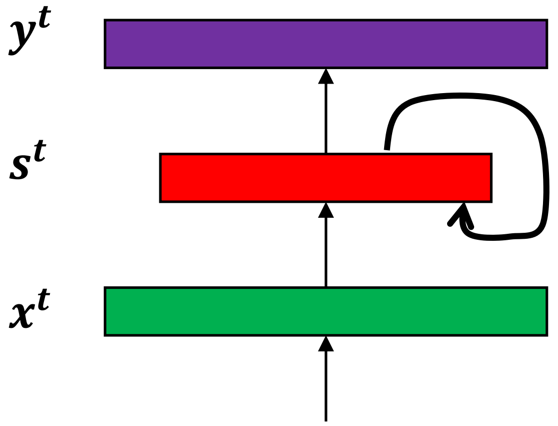
- Naïve RNN
  - Where should the recurrence be?



This one is equal to the previous one, why?

Hidden recurrence

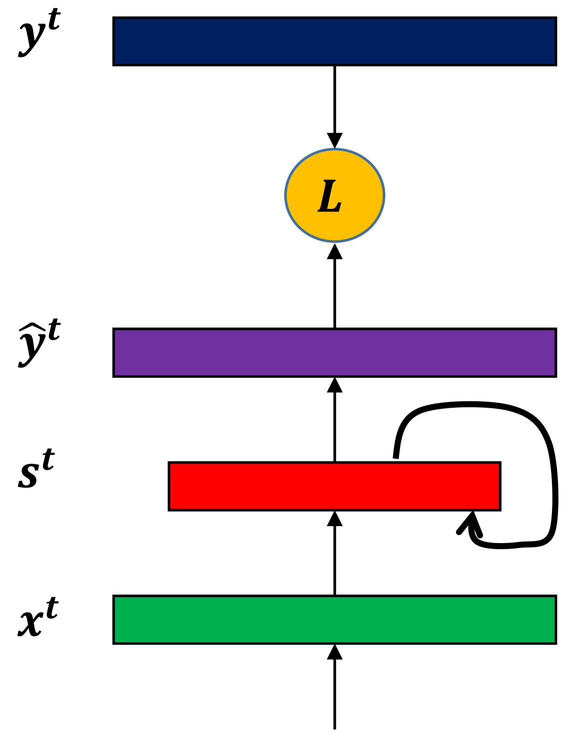
- In hidden to hidden mode, we give one kind of the forward computation formula
  - $a^{(t)} = b + Ws^{(t-1)} + Ux^{(t)}$
  - $s^{(t)} = \tanh(a^{(t)})$
  - $o^{(t)} = c + Vs^{(t)}$
  - $\hat{y} = softmax(o^{(t)})$



- Here, we assume:
  - hidden activation is $tanh$
  - Output is *discrete* with finite domain, so we use $softmax$
  - Each connection is specified with a *transformation matrix*, there are W, U, V, and bias b, c

Hidden recurrence

- Forward computation formula
  - $a^{(t)} = b + Ws^{(t-1)} + Ux^{(t)}$
  - $s^{(t)} = \tanh(a^{(t)})$
  - $o^{(t)} = c + Vs^{(t)}$
  - $\hat{y}^{(t)} = softmax(o^{(t)})$

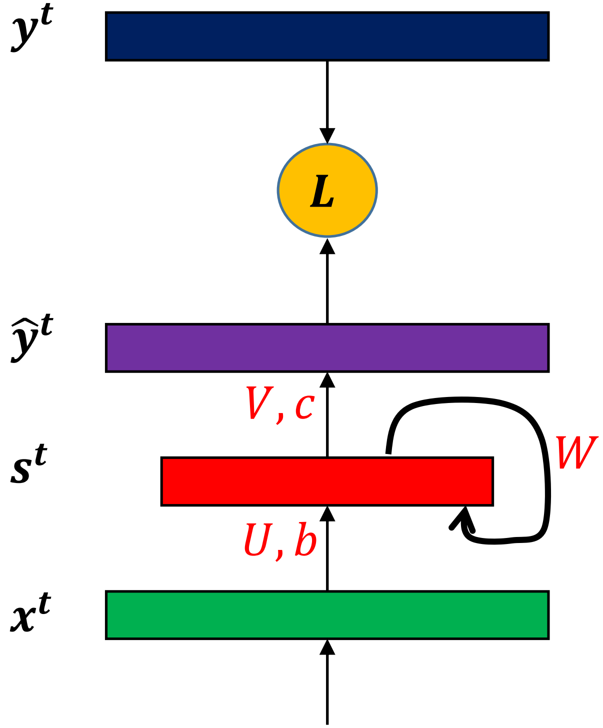


- Since in supervised FFNN, *every output* will be associated with a *supervision signal*
- In RNN, every time step, the output $\hat{y}^{(t)}$ will be compared to a supervision signal $y^{(t)}$ to produce a loss

Gradient Computation

- Now, let us consider training the RNN, or fitting the parameters to the given training data.

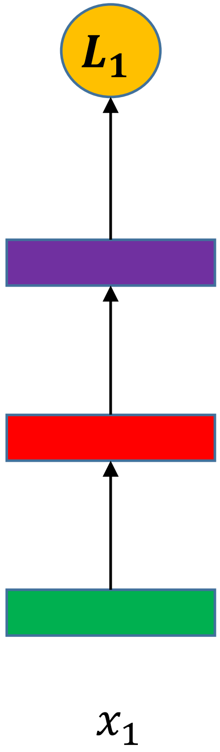
- Firstly, what are our parameters?
  - $a^{(t)} = b + Ws^{(t-1)} + Ux^{(t)}$
  - $s^{(t)} = \tanh(a^{(t)})$
  - $o^{(t)} = c + Vs^{(t)}$
  - $\hat{y}^{(t)} = softmax(o^{(t)})$



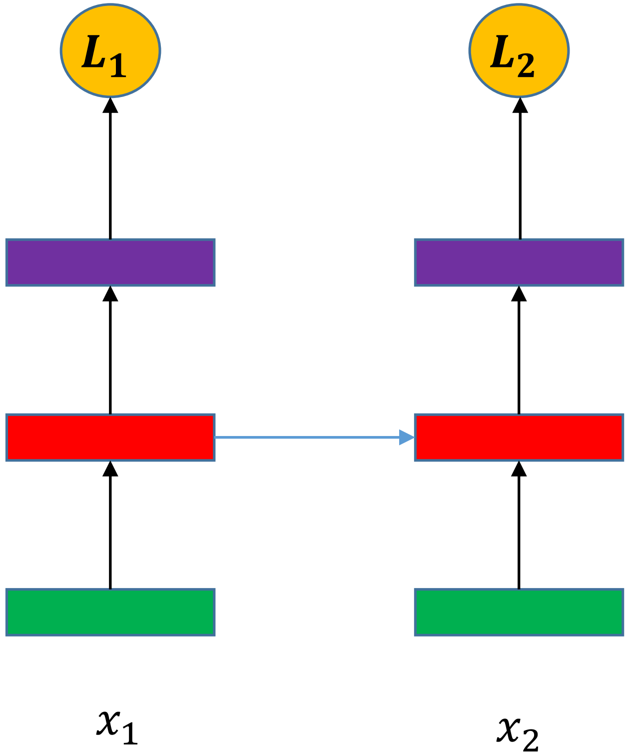
Backpropagation through Time

- Let us assume that
  - We are going to back propagate error after T steps of recurrence
  - Each time step, there is a loss over $\hat{y}^{(t)}$, we take the negative likelihood $L^{(t)} = -log\hat{y}_{y^{(t)}}^{(t)}$
  - So the total loss is $L = -\Sigma_t log\hat{y}_{y^{(t)}}^{(t)}$

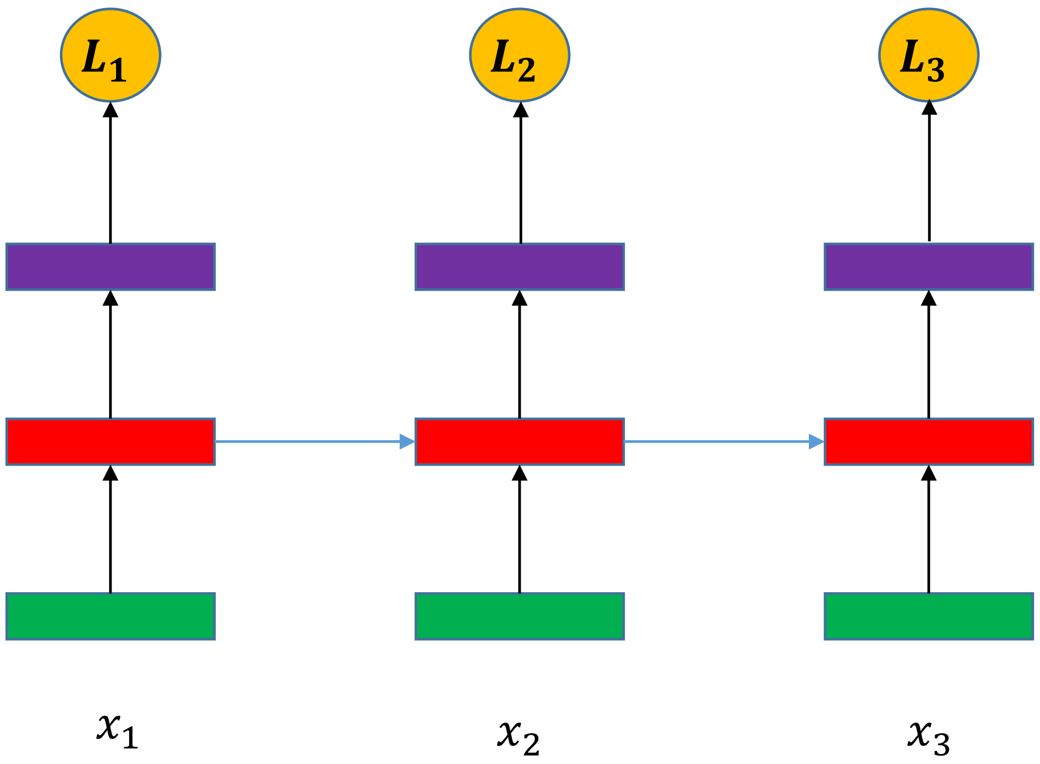
Backpropagation through Time



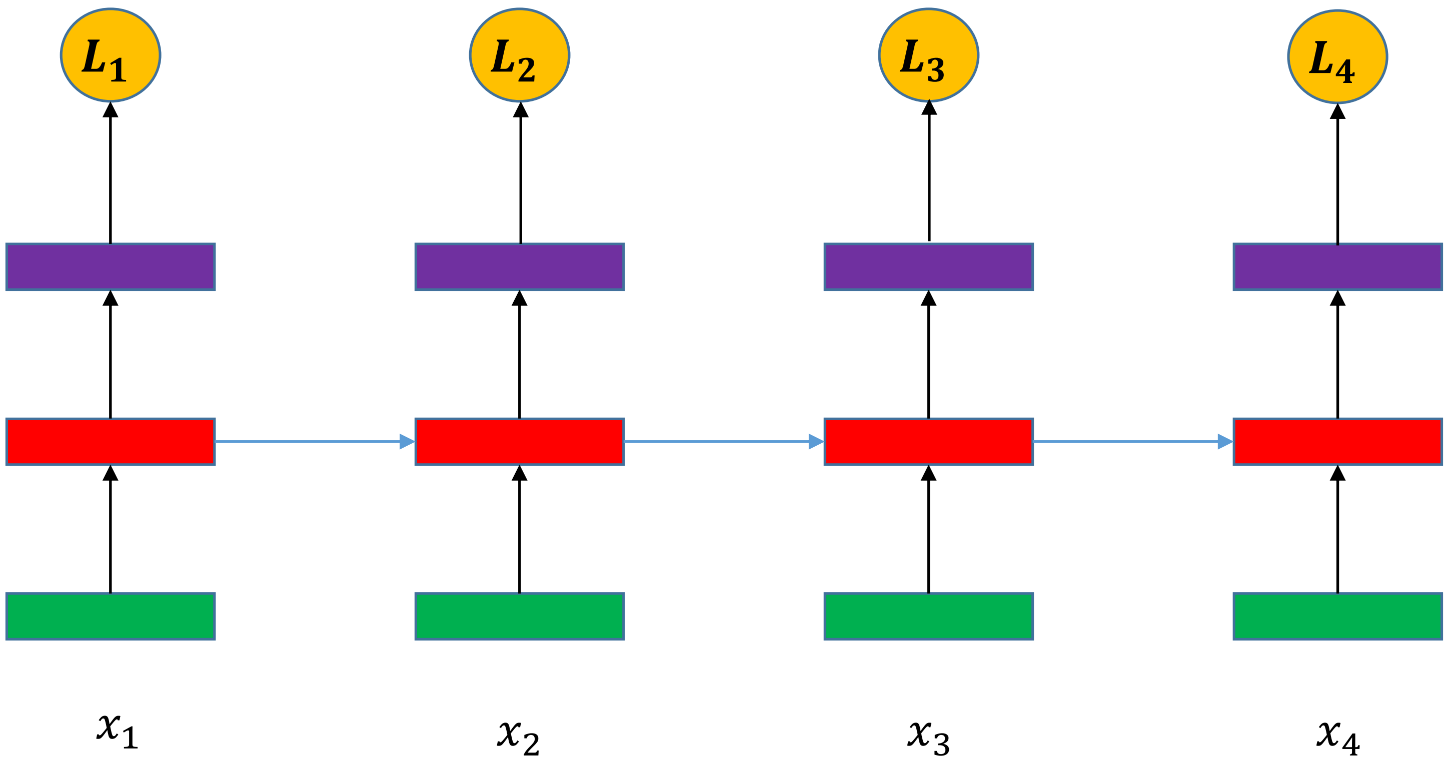
Backpropagation through Time



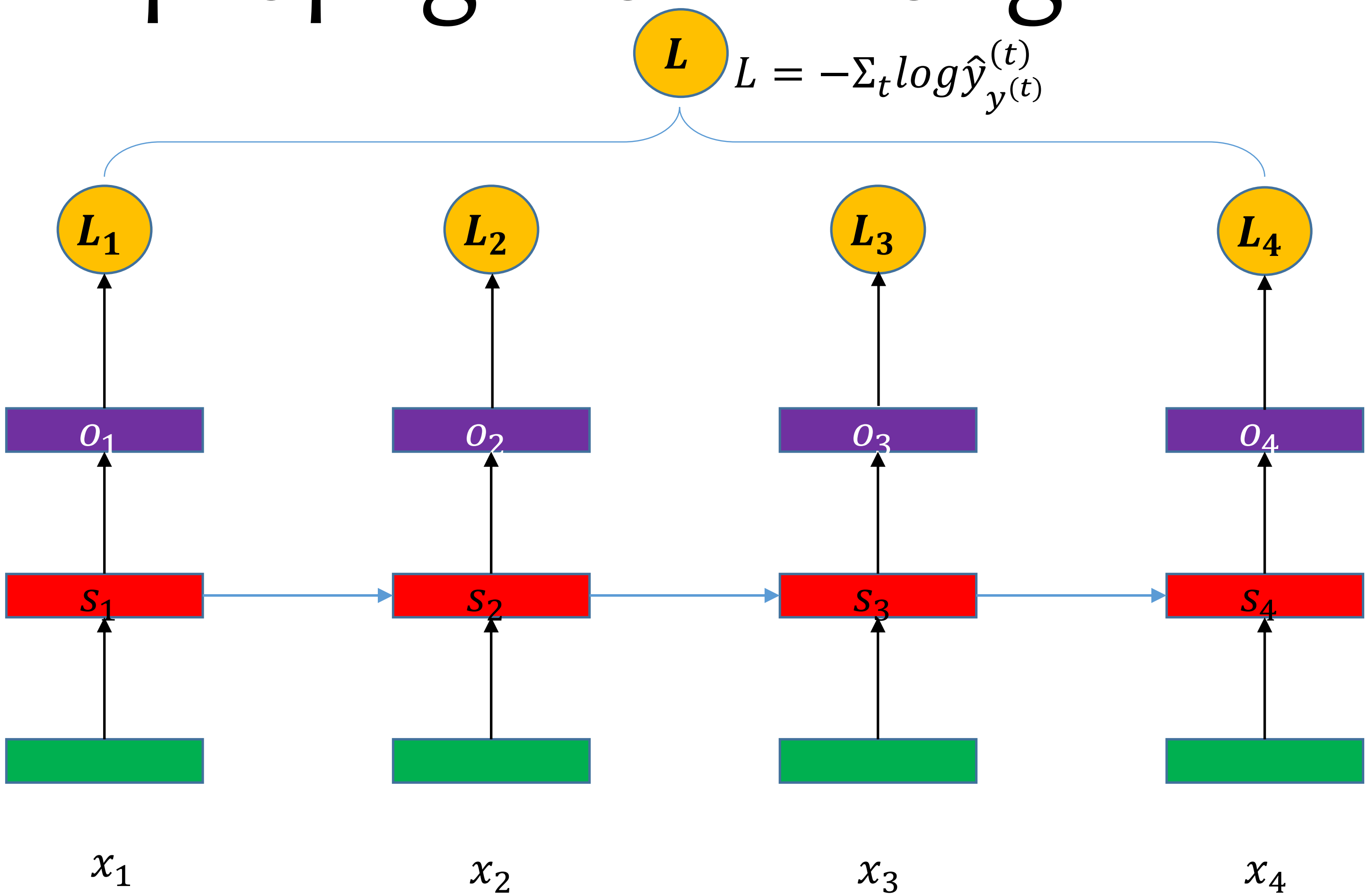
Backpropagation through Time



Backpropagation through Time



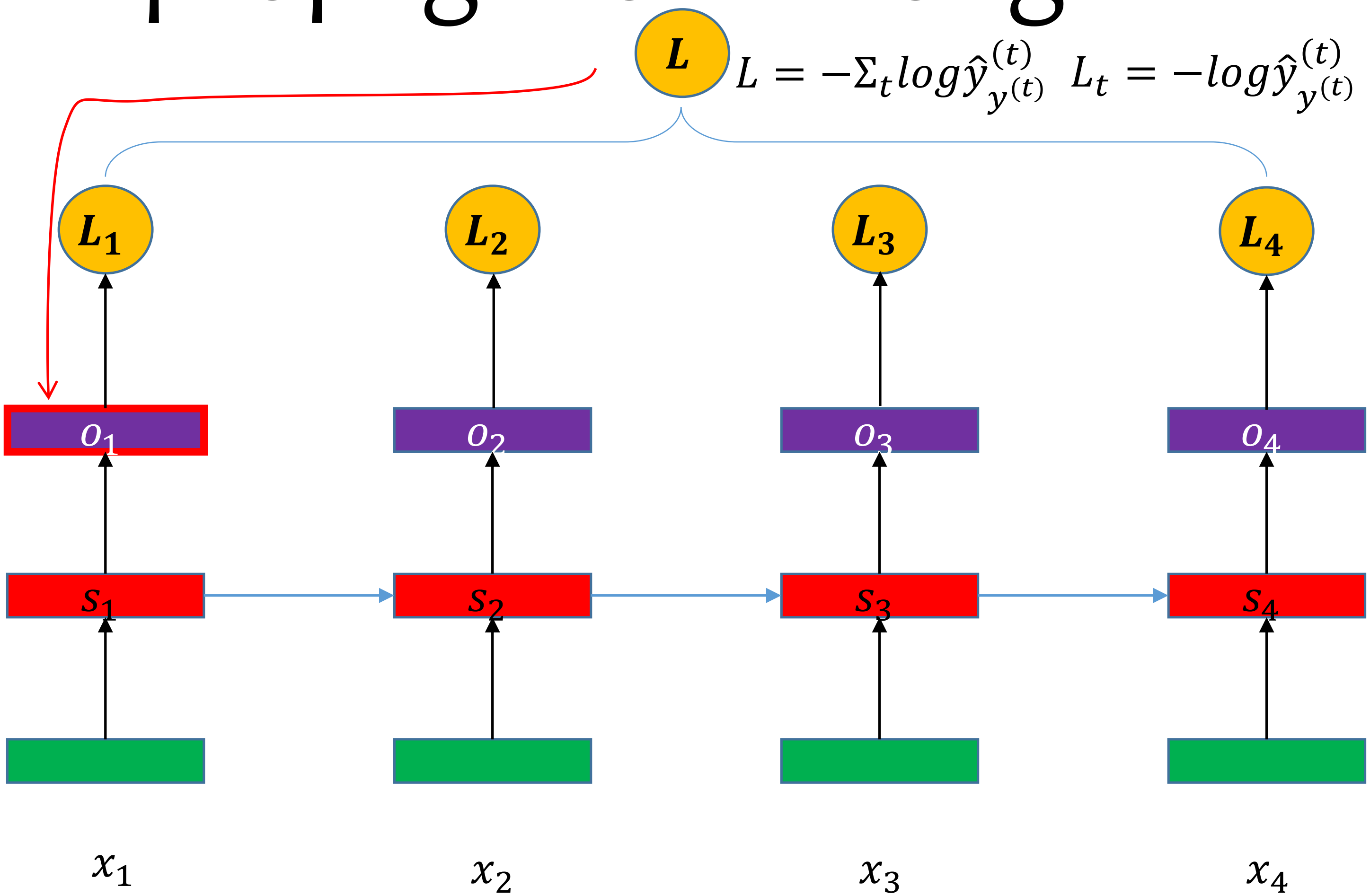
Backpropagation through Time



- $a_t = Ws_{t-1} + Ux_t + b$
- $s_t = tanh(s_{t-1})$
- $o_t = Vs_t + c$
- $\hat{y}_t = softmax(o_t)$

Backpropagation through Time

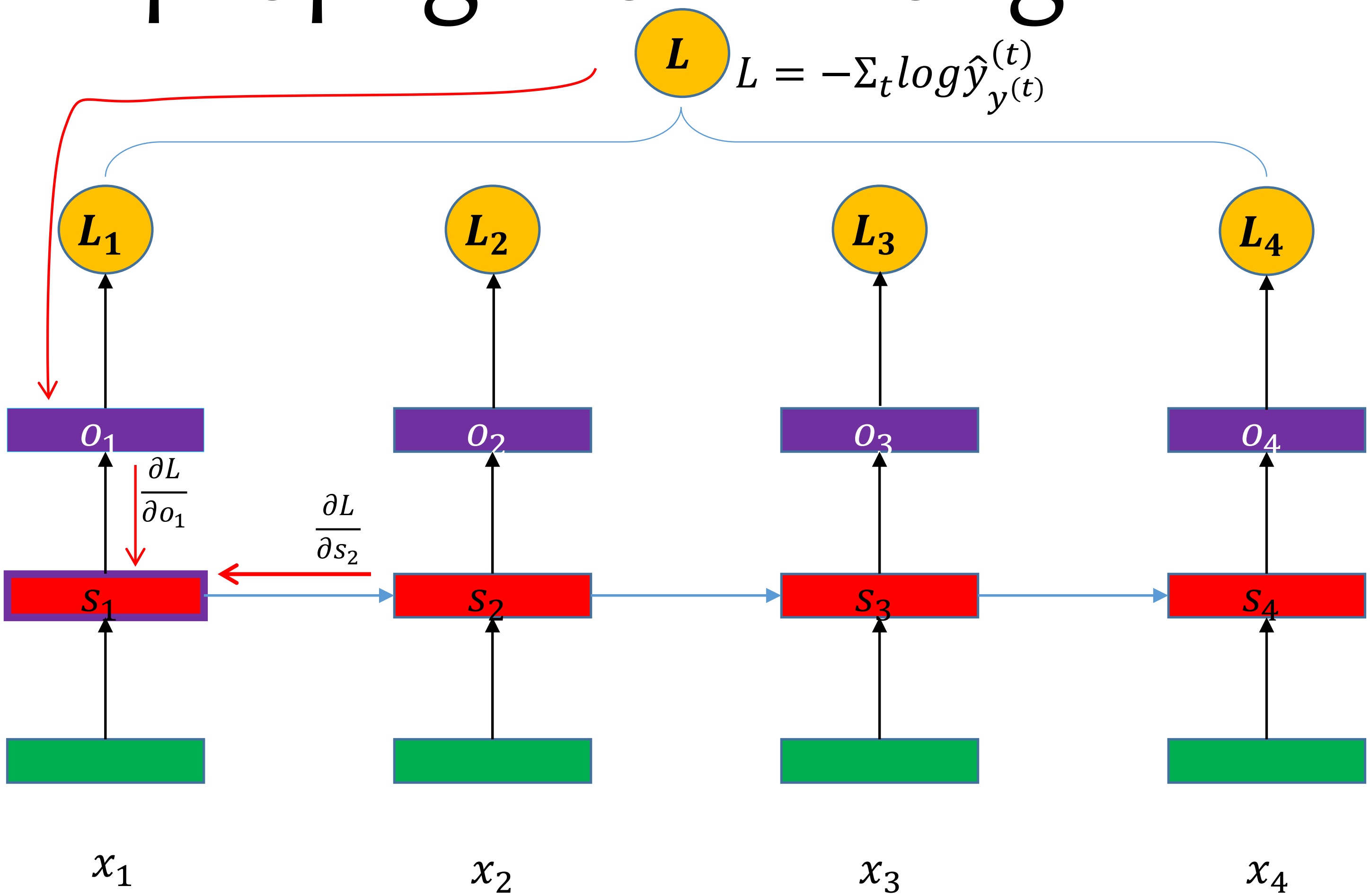
$L = -\Sigma_t log \hat{y}_{y^{(t)}}^{(t)}$ $L_t = -log \hat{y}_{y^{(t)}}^{(t)}$



- $a_t = Ws_{t-1} + Ux_t + b$
- $s_t = tanh(s_{t-1})$
- $o_t = Vs_t + c$
- $\hat{y}_t = softmax(o_t)$

- $\frac{\partial L}{\partial o_t} = \frac{\partial L}{\partial L_t} \frac{\partial L_t}{\partial o_t} = 1 \frac{\partial L_t}{\partial o_t} = \frac{\partial \log(softmax(o_t)_{y_t})}{\partial o_t}$

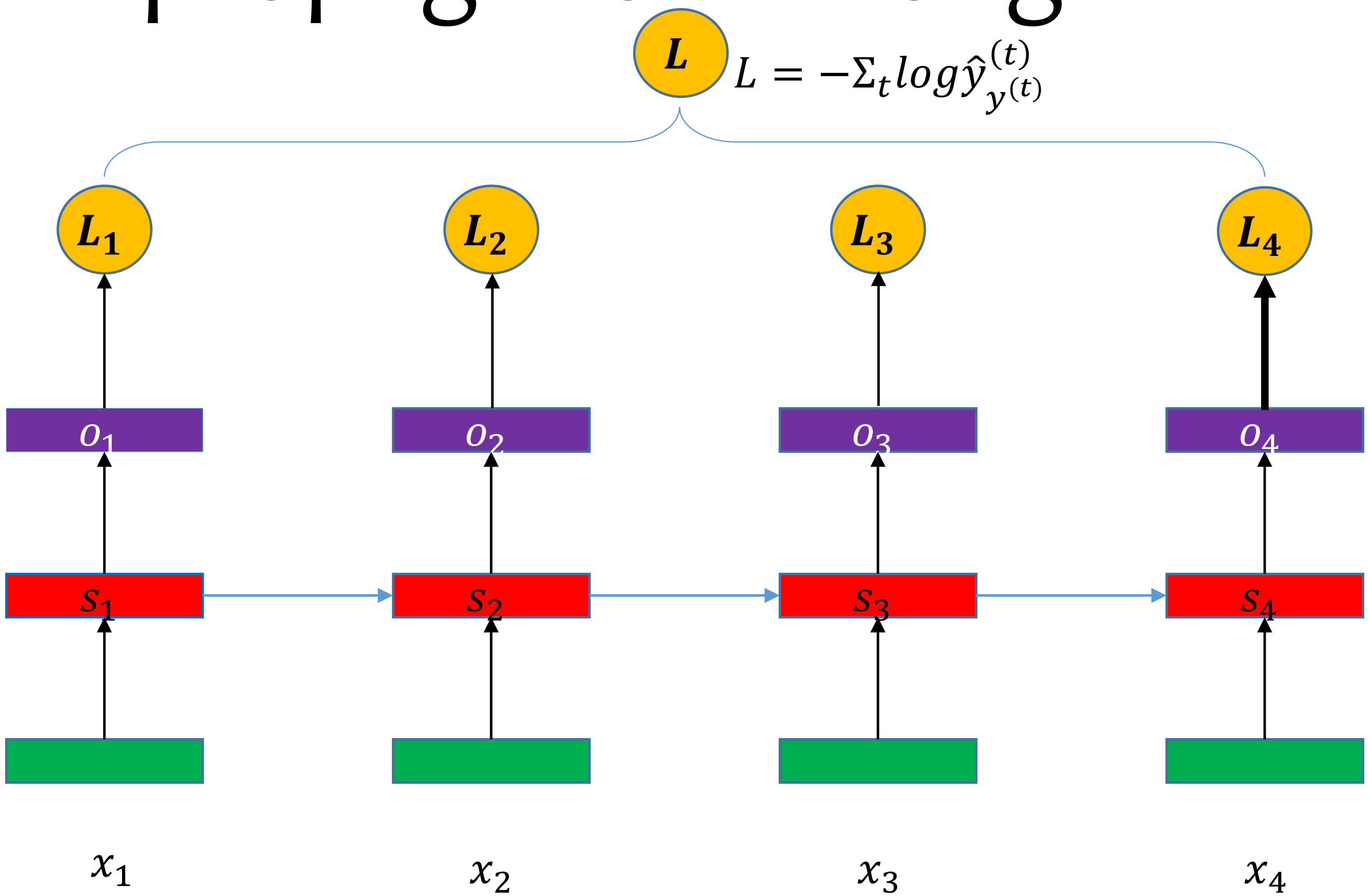
Backpropagation through Time



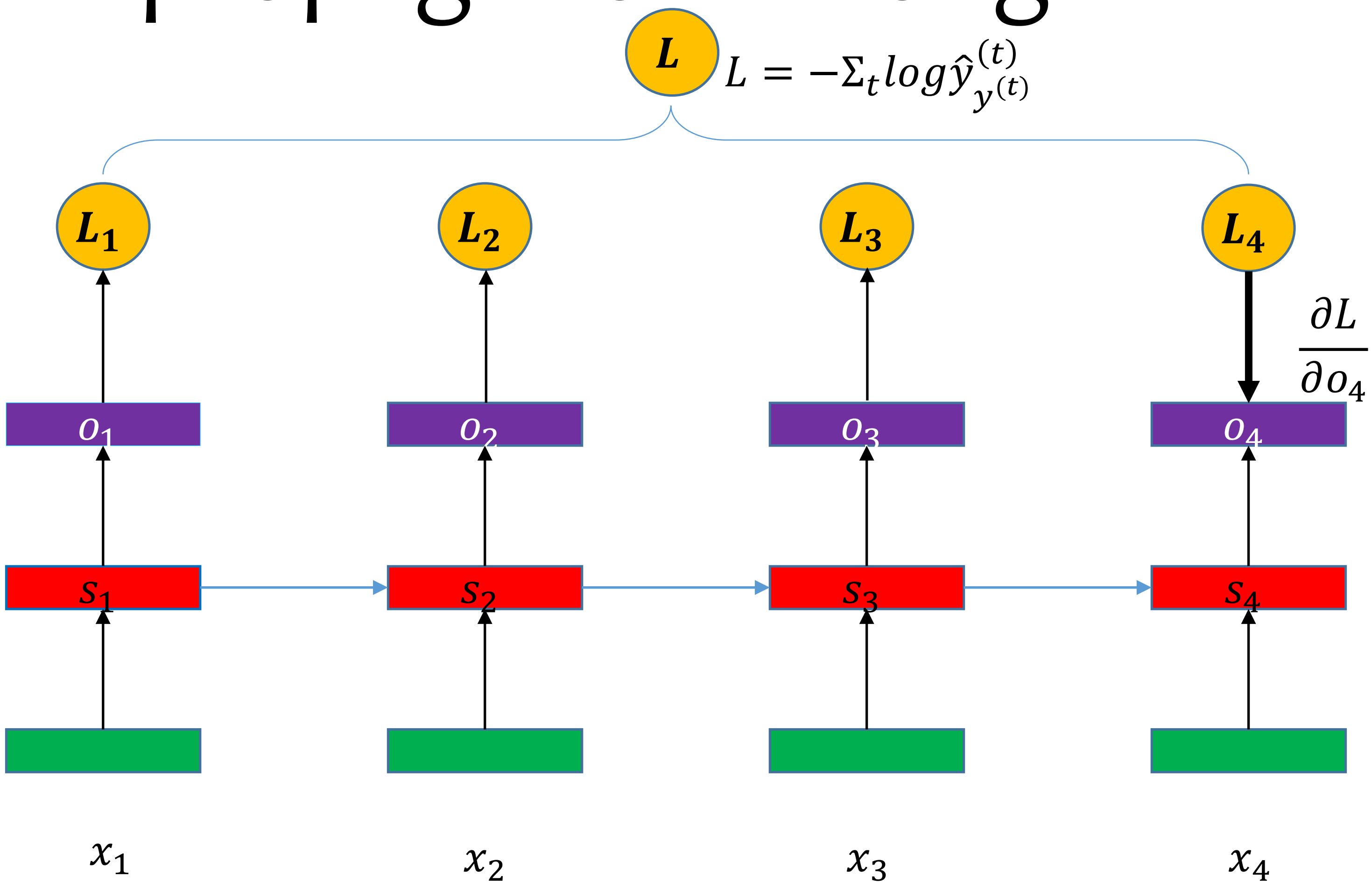
- $a_t = Ws_{t-1} + Ux_t + b$
- $s_t = tanh(s_{t-1})$
- $o_t = Vs_t + c$
- $\hat{y}_t = softmax(o_t)$

- $\frac{\partial L}{\partial o_t} = \frac{\partial L}{\partial L_t}\frac{\partial L_t}{\partial o_t} = 1\,\frac{\partial L_t}{\partial o_t} = \frac{\partial \log(softmax(o_t)_{y_t})}{\partial o_t}$
- $\frac{\partial L}{\partial s_t} = \frac{\partial L}{\partial o_t}\frac{\partial o_t}{\partial s_t} + \frac{\partial L}{\partial s_{t+1}}\frac{\partial s_{t+1}}{\partial s_t}$
- $\frac{\partial L}{\partial s_T} = \frac{\partial L}{\partial o_T}\frac{\partial o_T}{\partial s_T}$

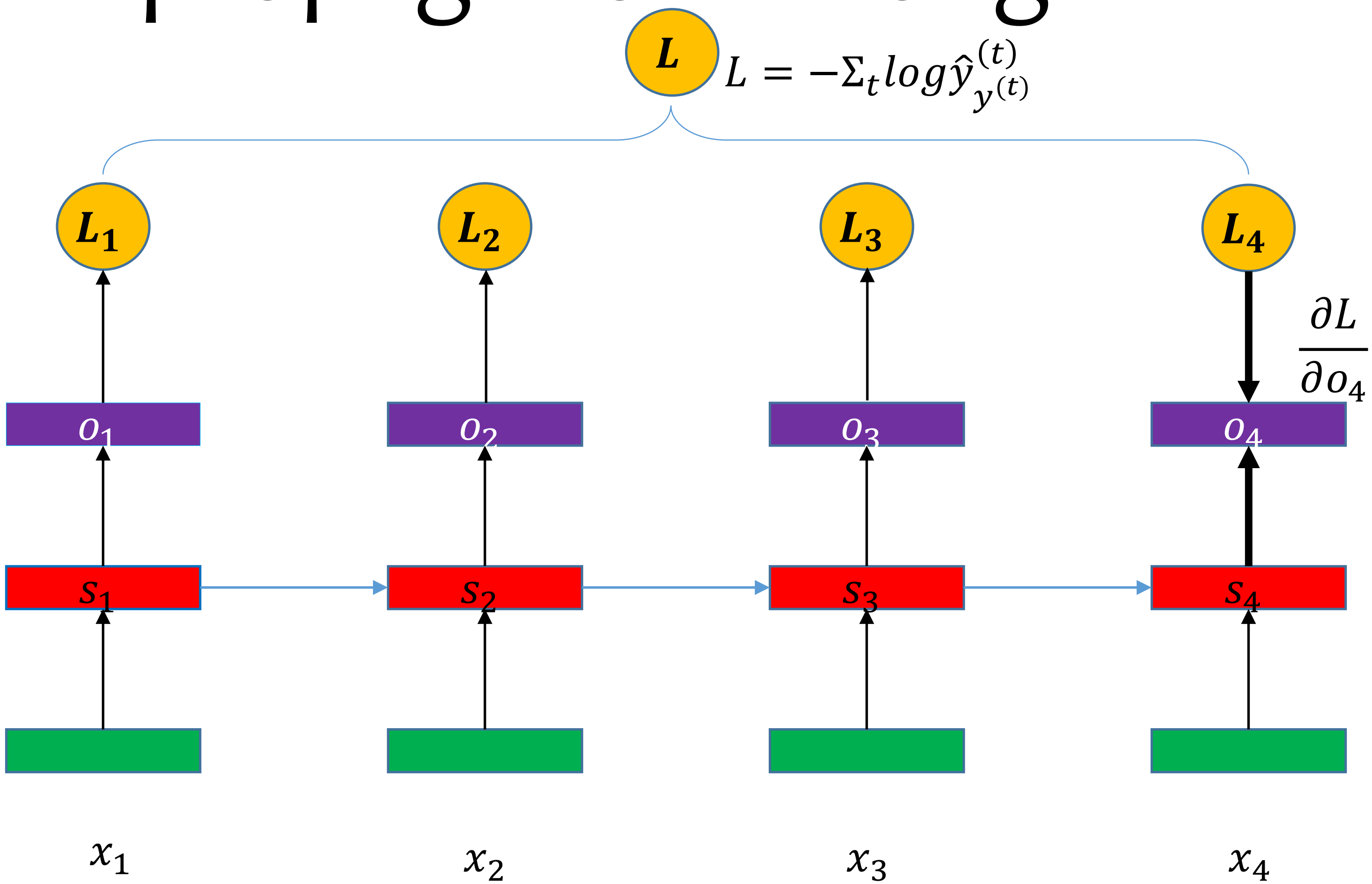
Backpropagation through Time



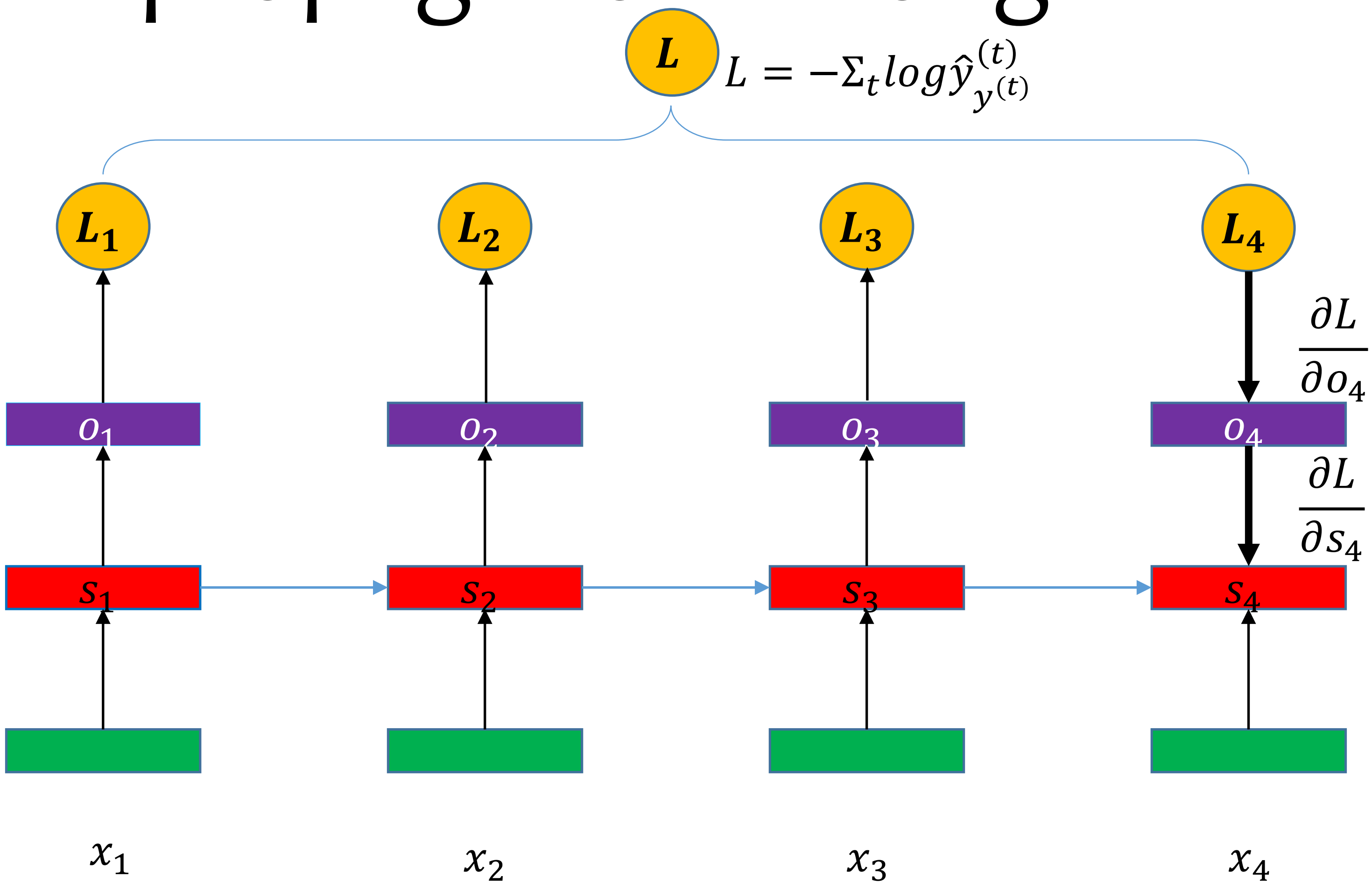
Backpropagation through Time



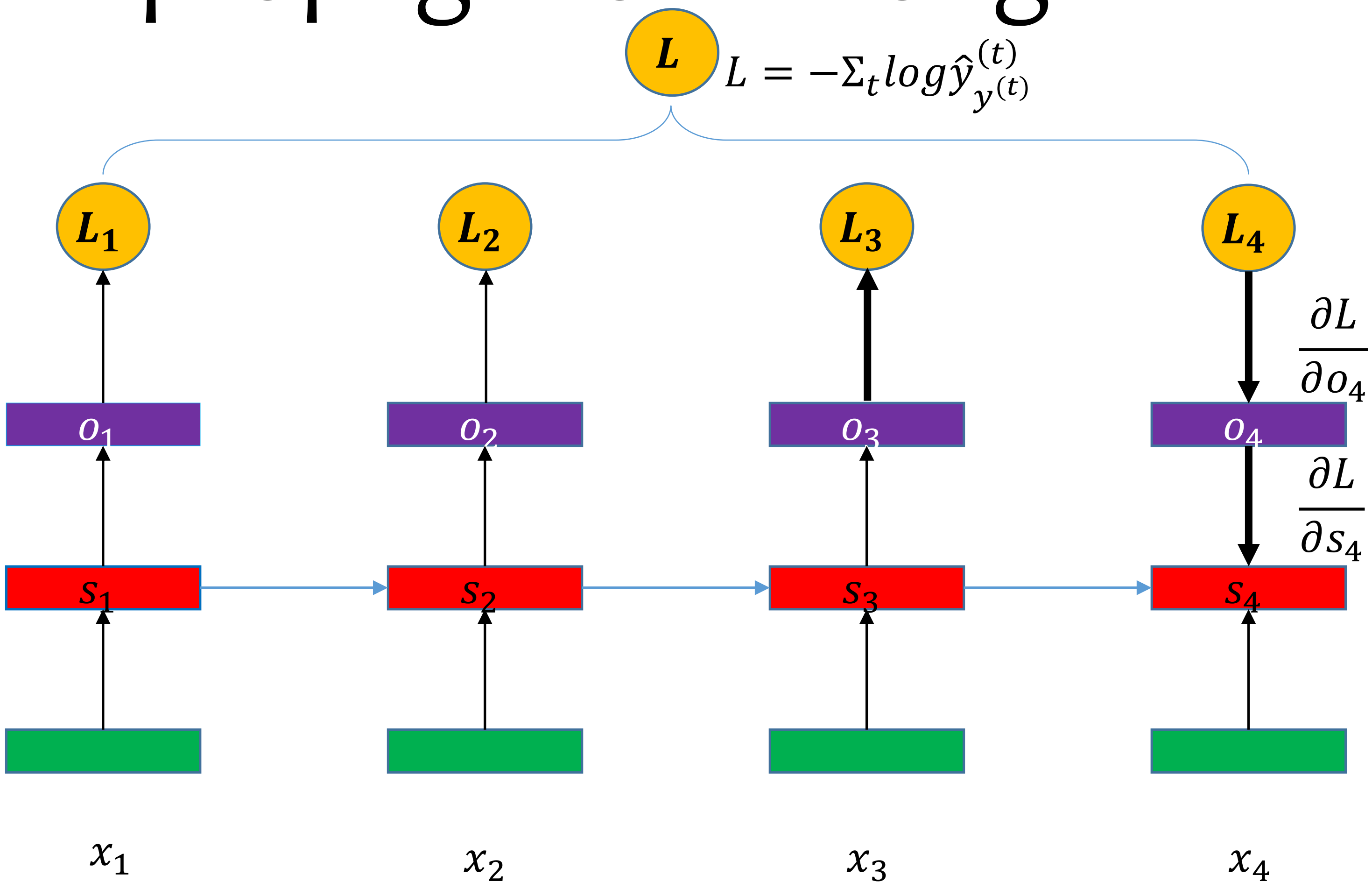
Backpropagation through Time



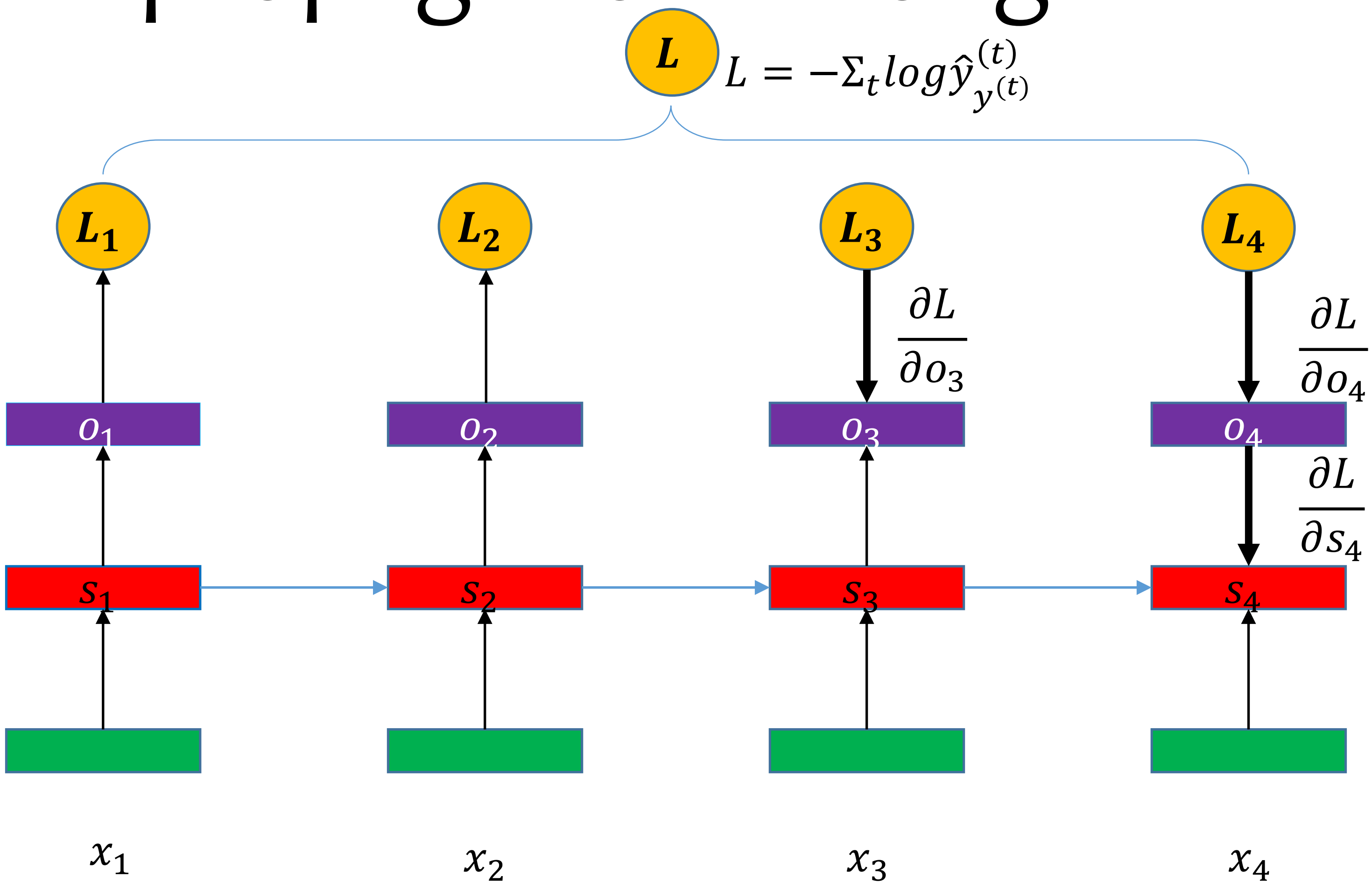
Backpropagation through Time



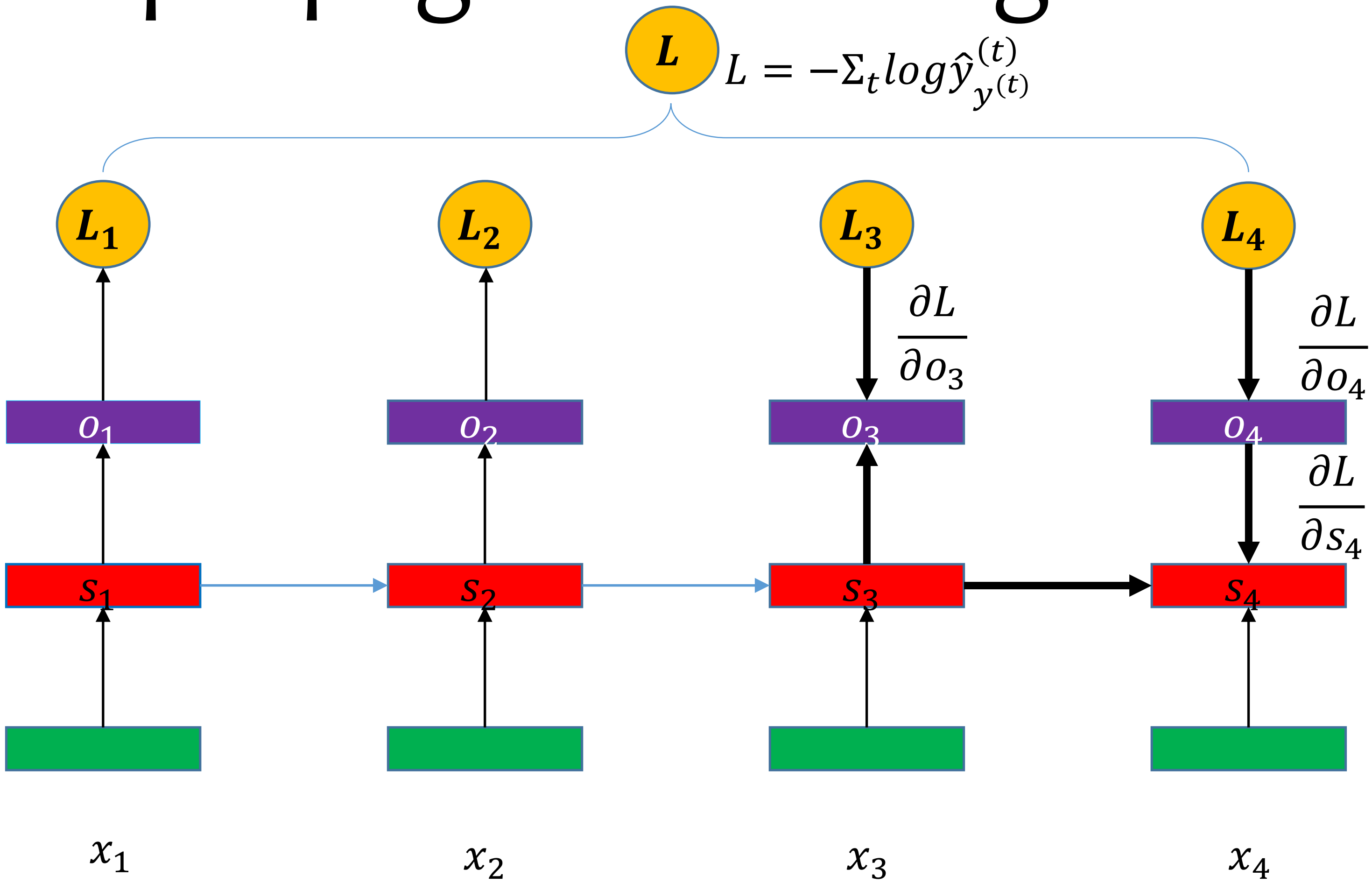
Backpropagation through Time



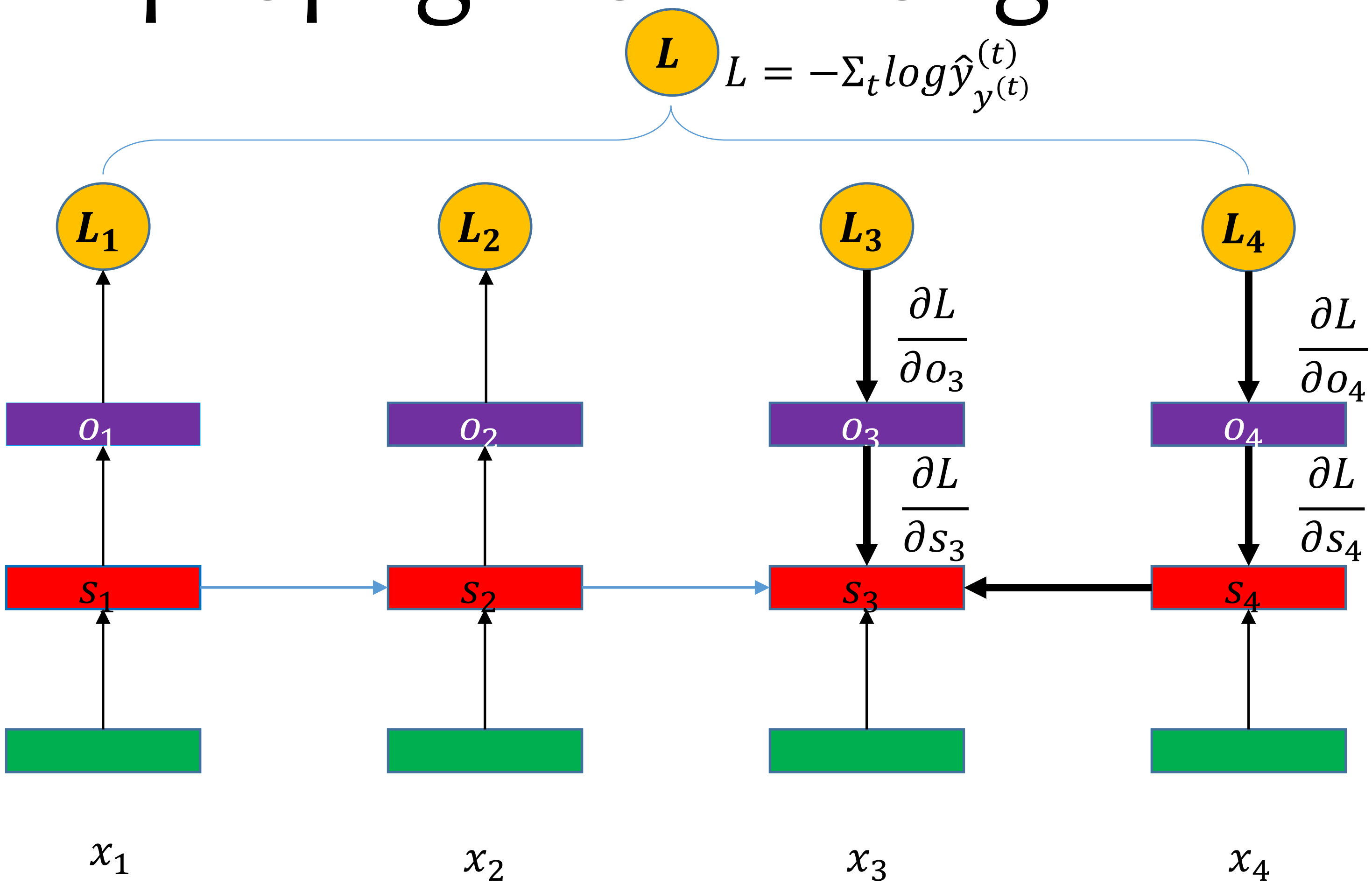
Backpropagation through Time



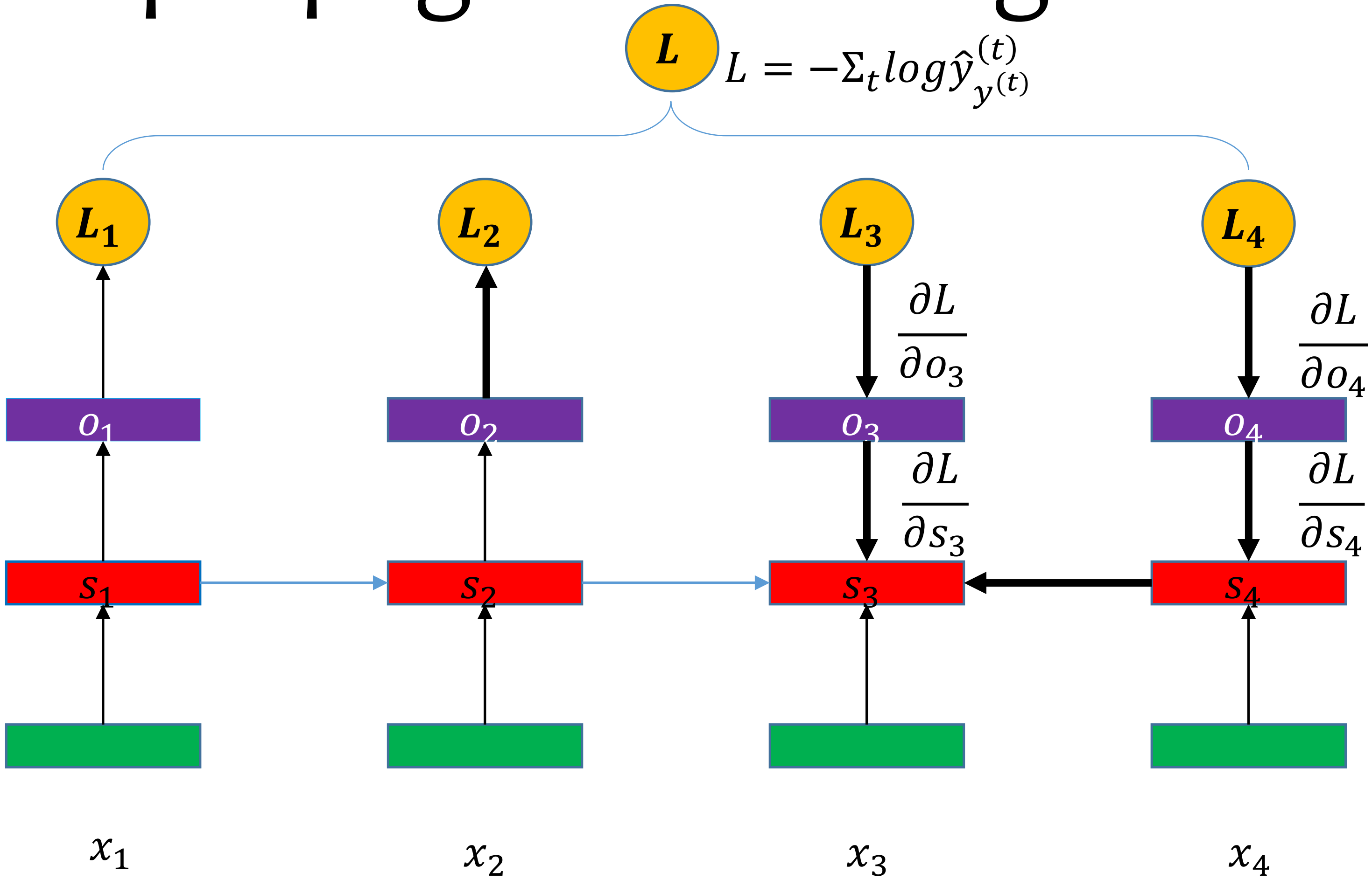
Backpropagation through Time



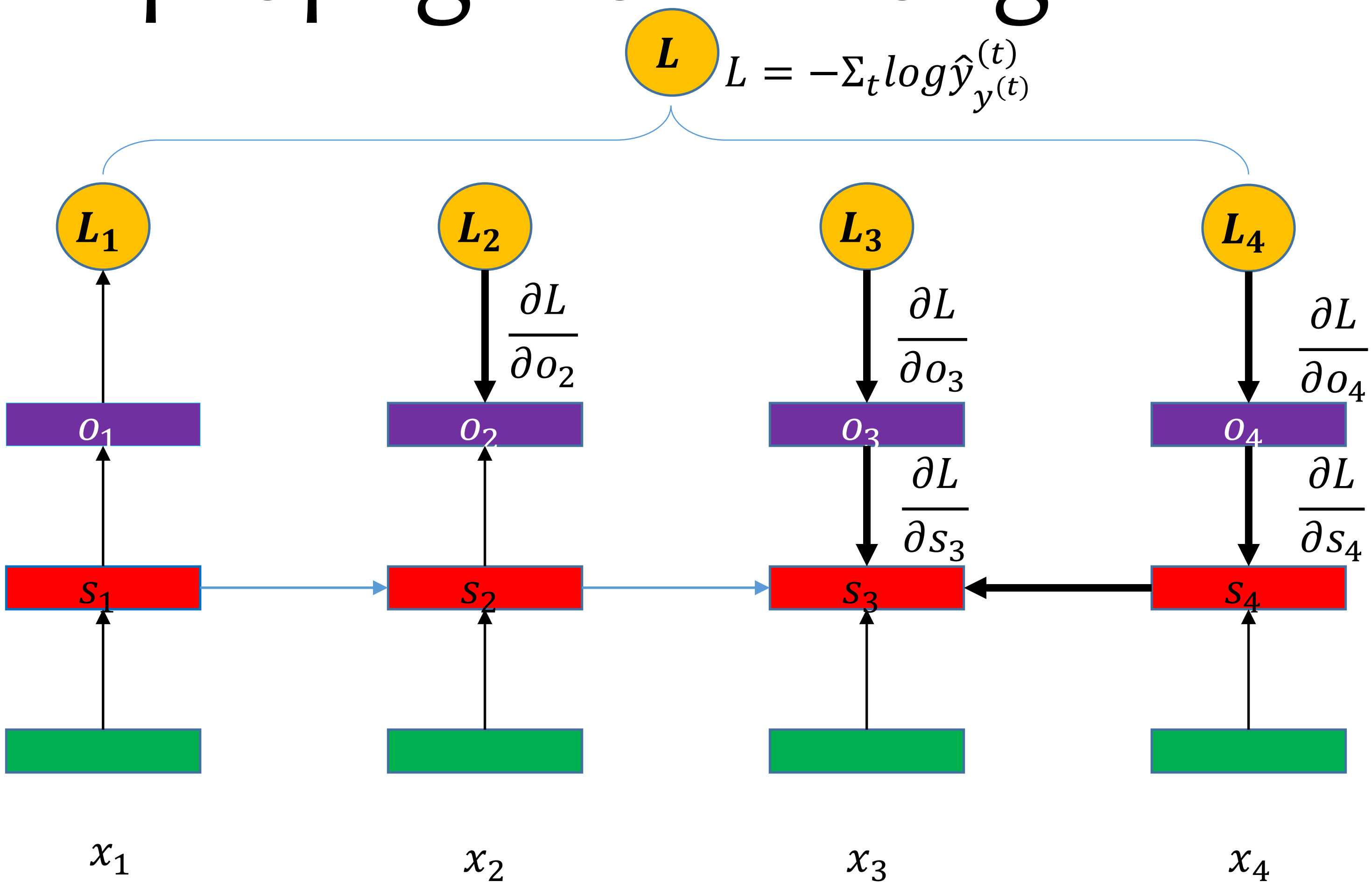
Backpropagation through Time



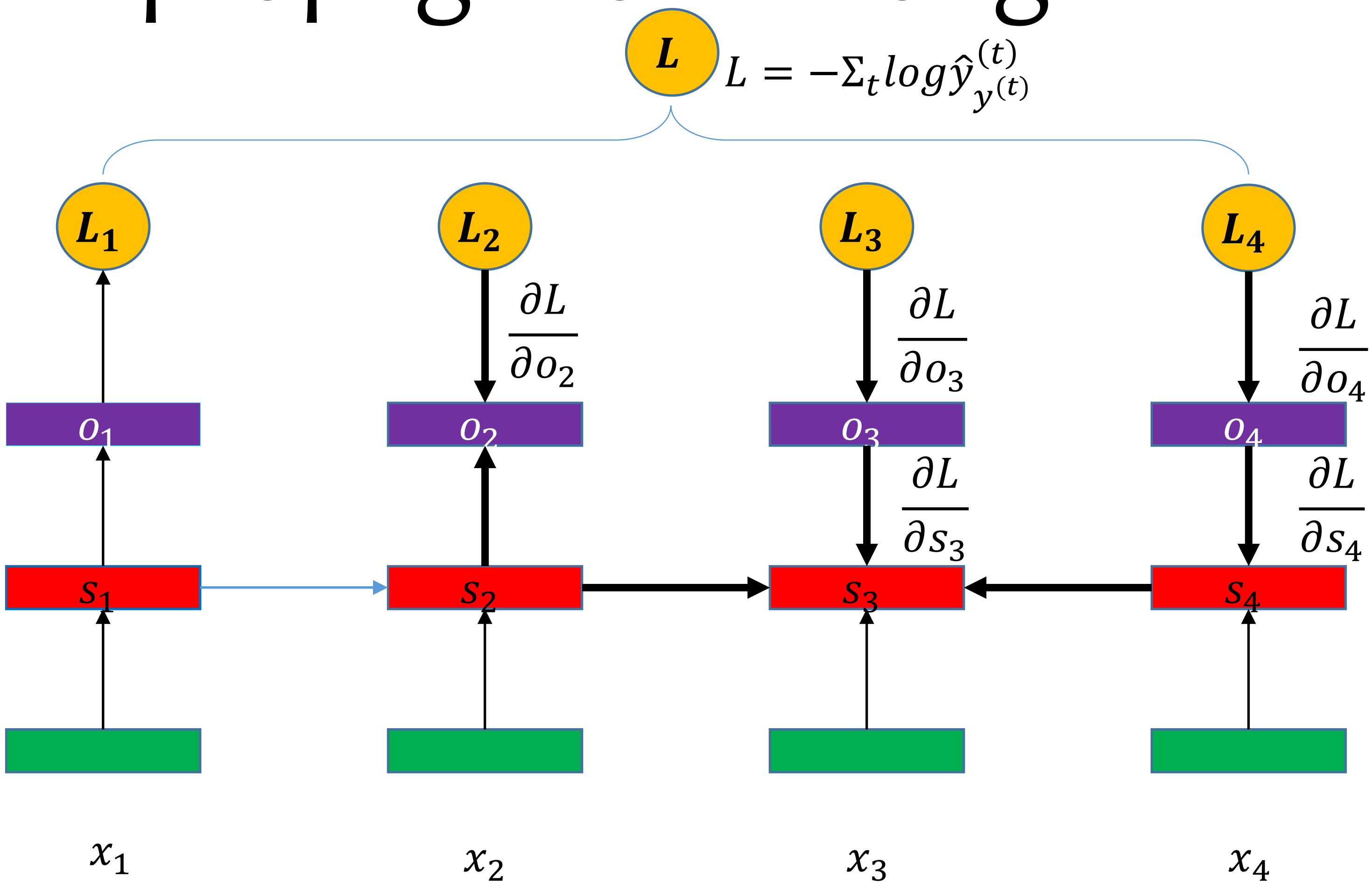
Backpropagation through Time



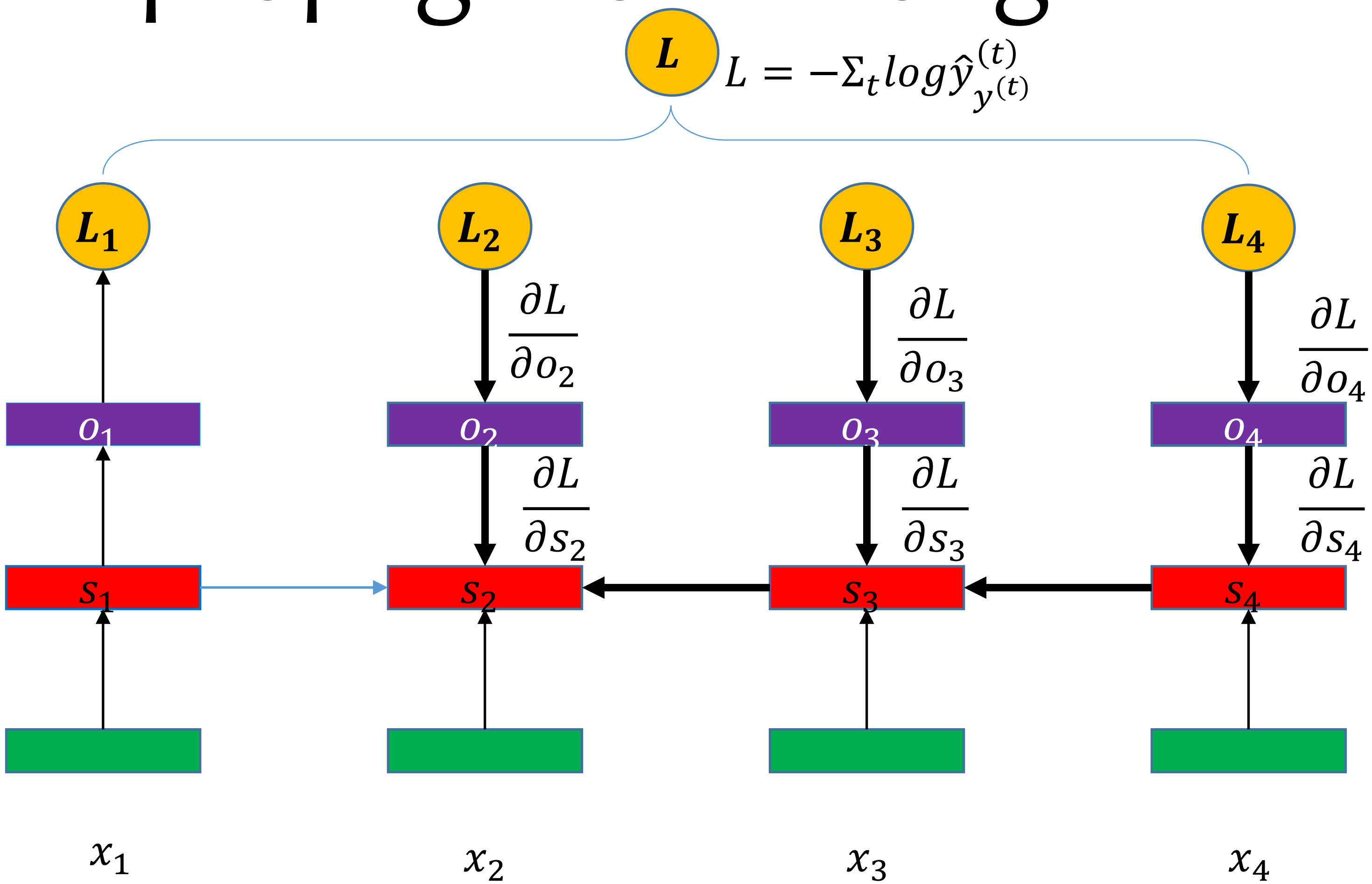
Backpropagation through Time



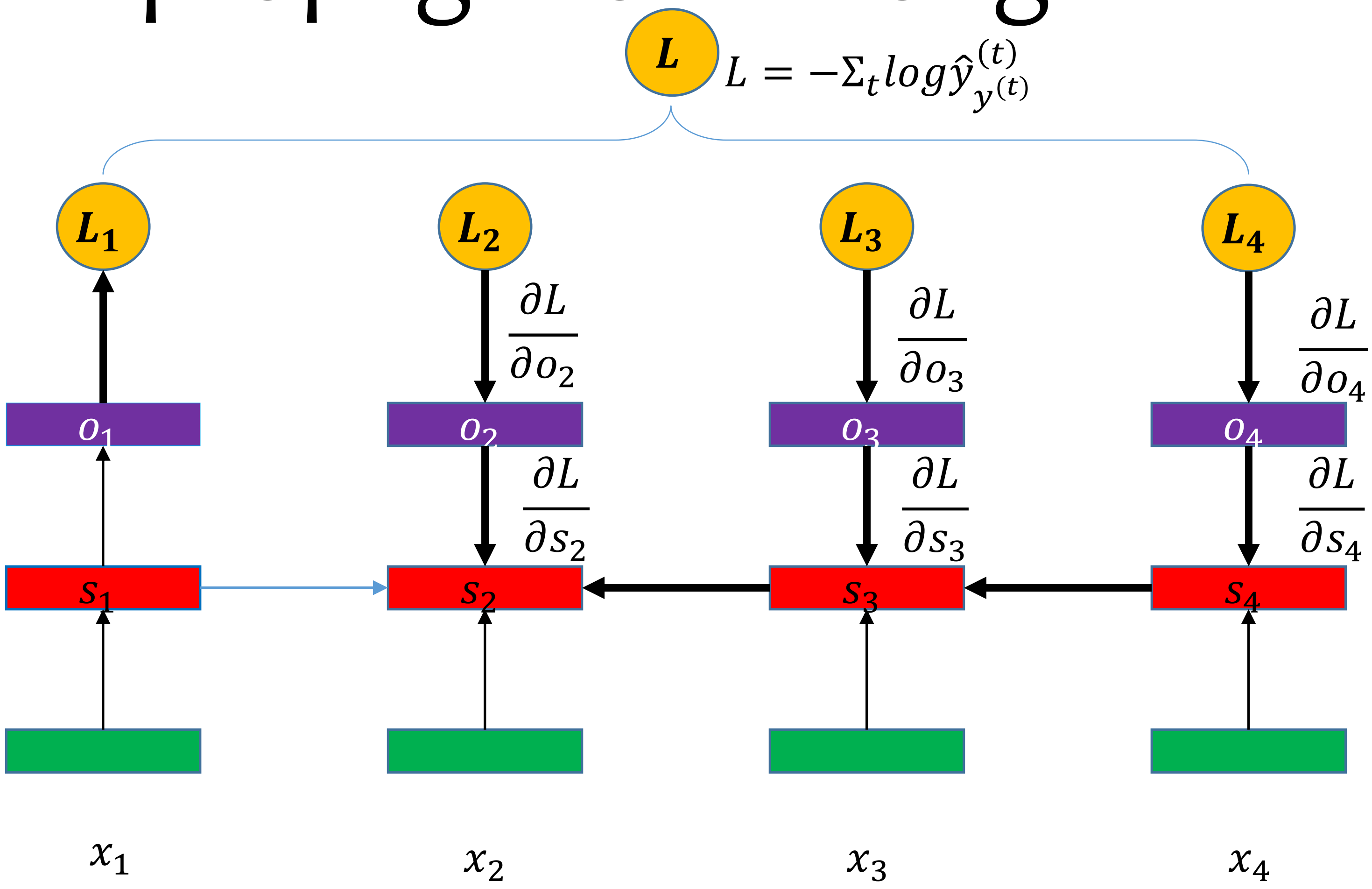
Backpropagation through Time



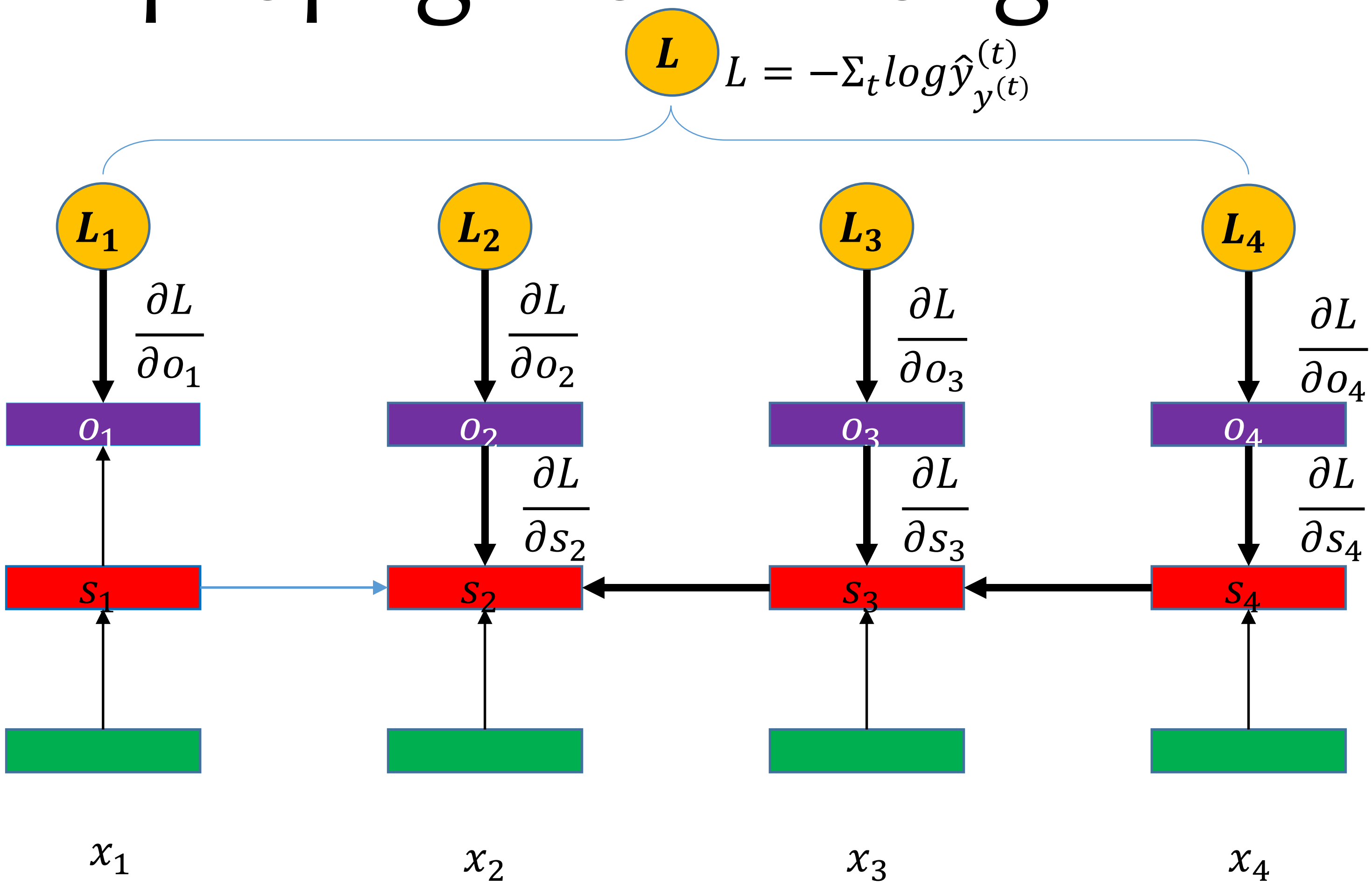
Backpropagation through Time



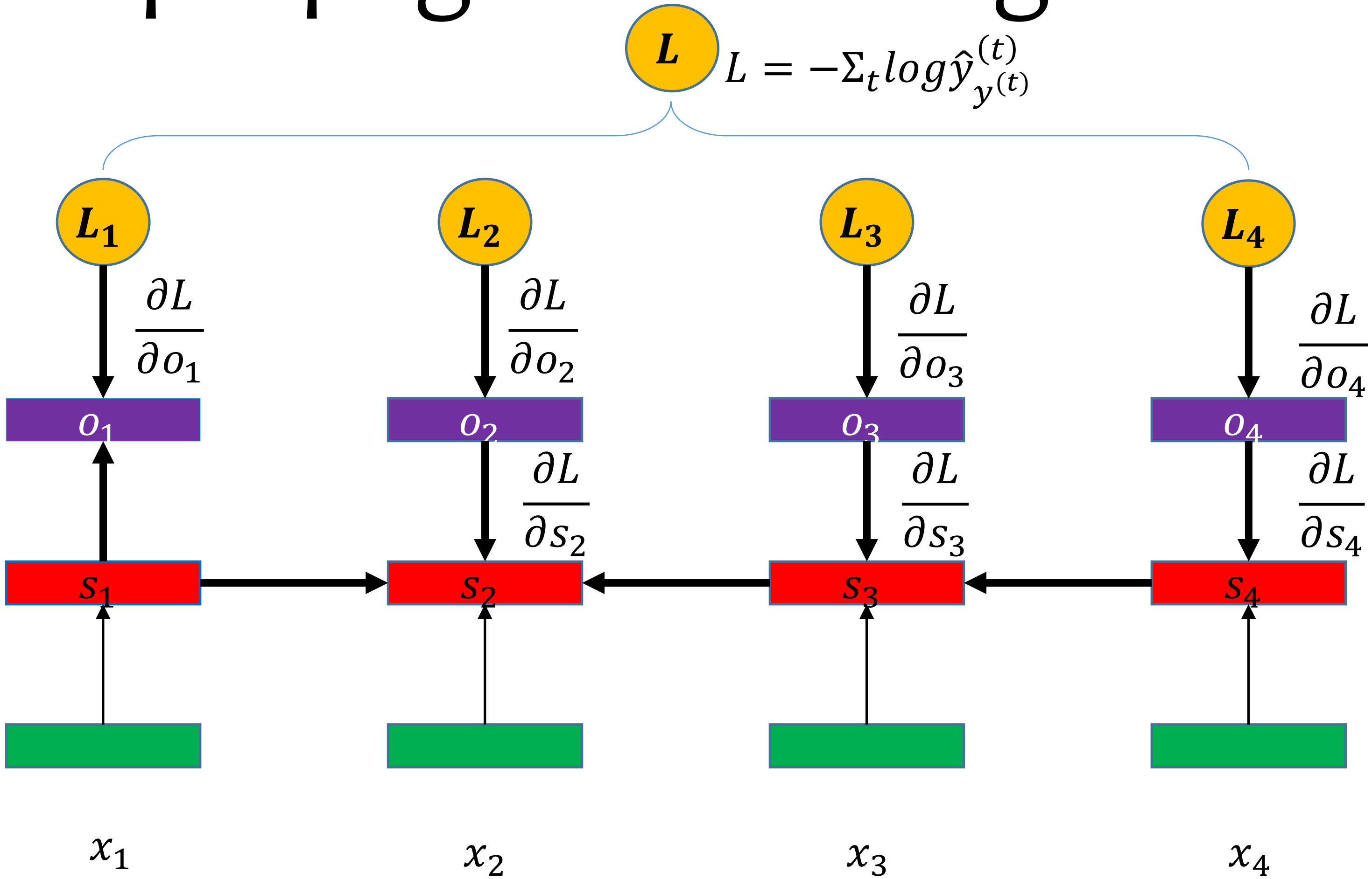
Backpropagation through Time



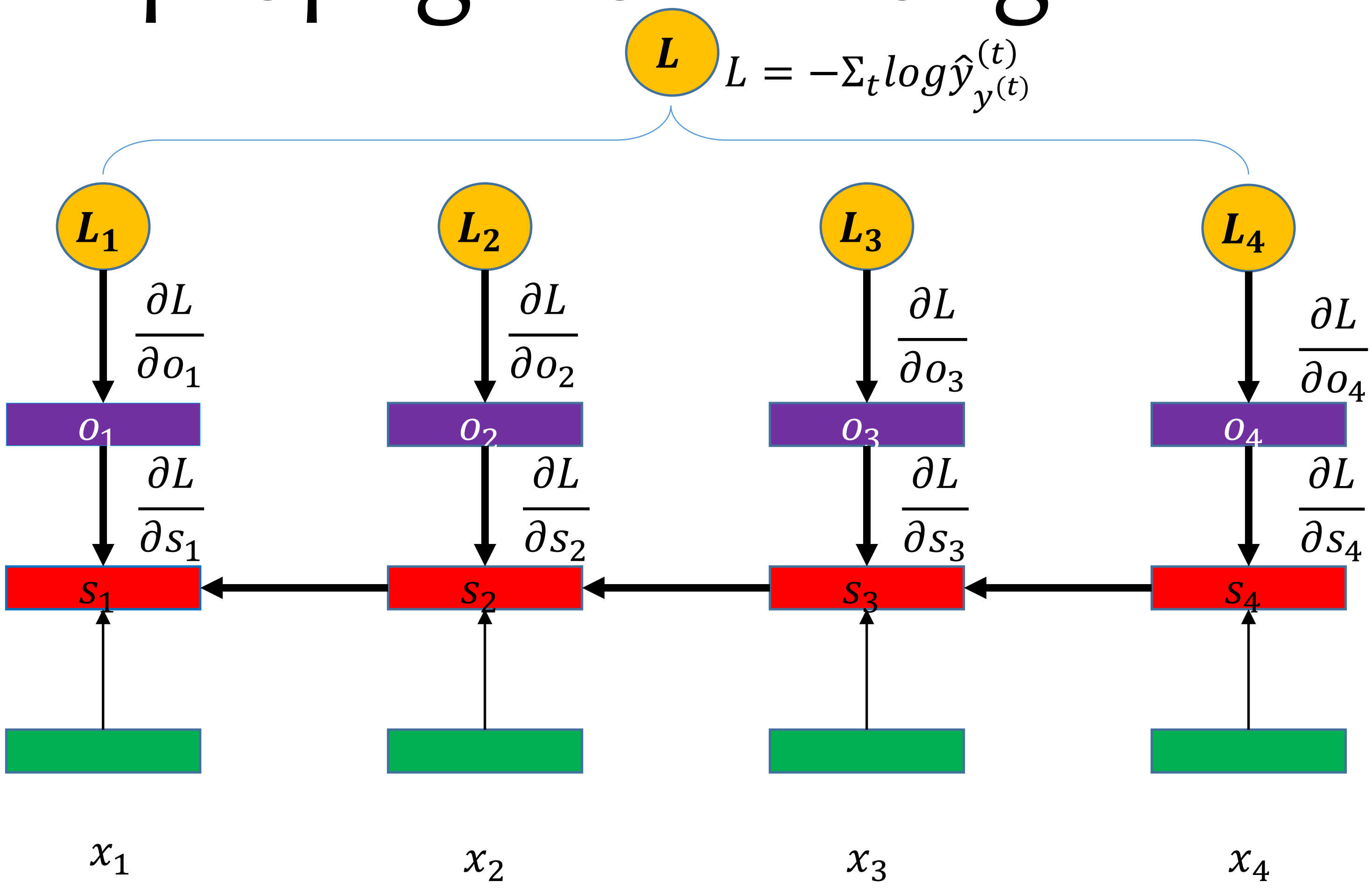
Backpropagation through Time



Backpropagation through Time



Backpropagation through Time



What RNN Really Model?

- Concept of Parameterization
  - In language modeling, we are trying to compute the probability
  - $P(s) = P(x_1, x_2, \ldots, x_{|s|})$

What RNN Really Model?

- Concept of Parameterization
  - In language modeling, we are trying to compute the probability
  - $P(s) = P(x_1, x_2, \ldots, x_{|s|})$

Conceptually, we want to model the probability of a sentence.

What RNN Really Model?

- Concept of Parameterization
  - In language modeling, we are trying to compute the probability
  - $P(s) = P(x_1, x_2, \ldots, x_{|s|})$

Since a sentence is composed of several words, we can instead model words.

What RNN Really Model?

- Concept of Parameterization
  - In language modeling, we are trying to compute the probability
  - $P(s) = P\left(x_1, x_2, \ldots, x_{|s|}\right) = \Sigma_{\mathrm{t}} P(x_t | x_{1:t-1})$

Chain rule can help us derive this formula further more.

What RNN Really Model?

- Concept of Parameterization
  - In language modeling, we are trying to compute the probability
  - $P(s) = P\left(x_1, x_2, \dots, x_{|s|}\right) = \Sigma_{\mathrm{t}} P(x_t | x_{t-2:t-1})$

Under 2nd-order Markov Assumption.

What RNN Really Model?

- Concept of Parameterization
  - In language modeling, we are trying to compute the probability
  - $P(s) = P\left(x_1, x_2, \dots, x_{|s|}\right) = \Sigma_{\mathrm{t}} P(x_t | x_{t-2:t-1})$

Counting-based estimation.

What RNN Really Model?

- Concept of Parameterization
  - In language modeling, we are trying to compute the probability
  - $P(s) = P\left(x_1, x_2, \dots, x_{|s|}\right) = \Sigma_{\mathrm{t}} P(x_t | x_{t-2:t-1})$

Feedforward Neural Network parameterization.

What RNN Really Model?

- Concept of Parameterization
  - After *mathematical derivation* (that is logically rigorous)
  - Try to use some *data model(s)* to further transform the modeling problem into a *parameter estimation* problem.
  - Since the data model always has *learnable parameters*, so we call the whole process as **PARAMETERIZATION**.

What RNN Really Model?

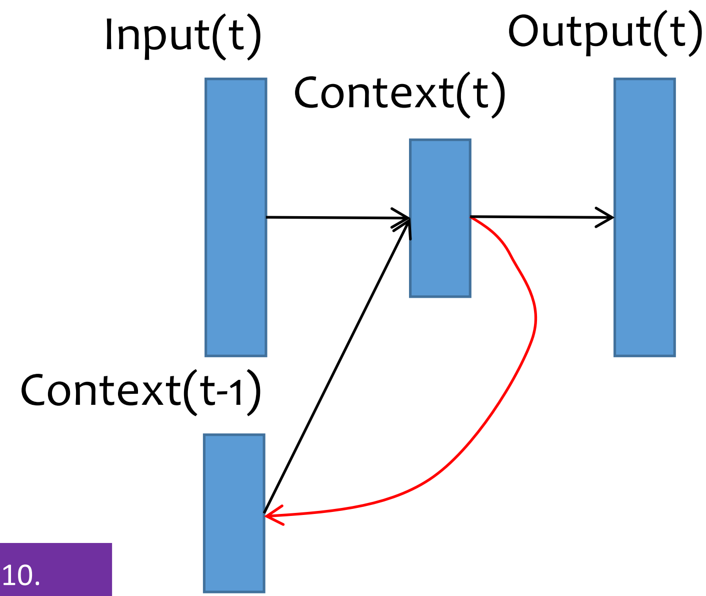
- Concept of Parameterization
  - In language modeling, we are trying to compute the probability
  - $P(s) = P\left(x_1, x_2, \dots, x_{|s|}\right) = \Sigma_{\mathrm{t}} P(x_t | x_{1:t-1})$

Back to the previous language modeling problem.

What RNN Really Model?

- Concept of Parameterization
  - In language modeling, we are trying to compute the probability
  - $P(s) = P\left(x_1, x_2, \dots, x_{|s|}\right) = \Sigma_{\mathrm{t}} P(x_t | x_{1:t-1})$

An RNN can be used to parameterize this term as well.



Recurrent neural network based language model, Mikolov et al. Interspeech 2010.

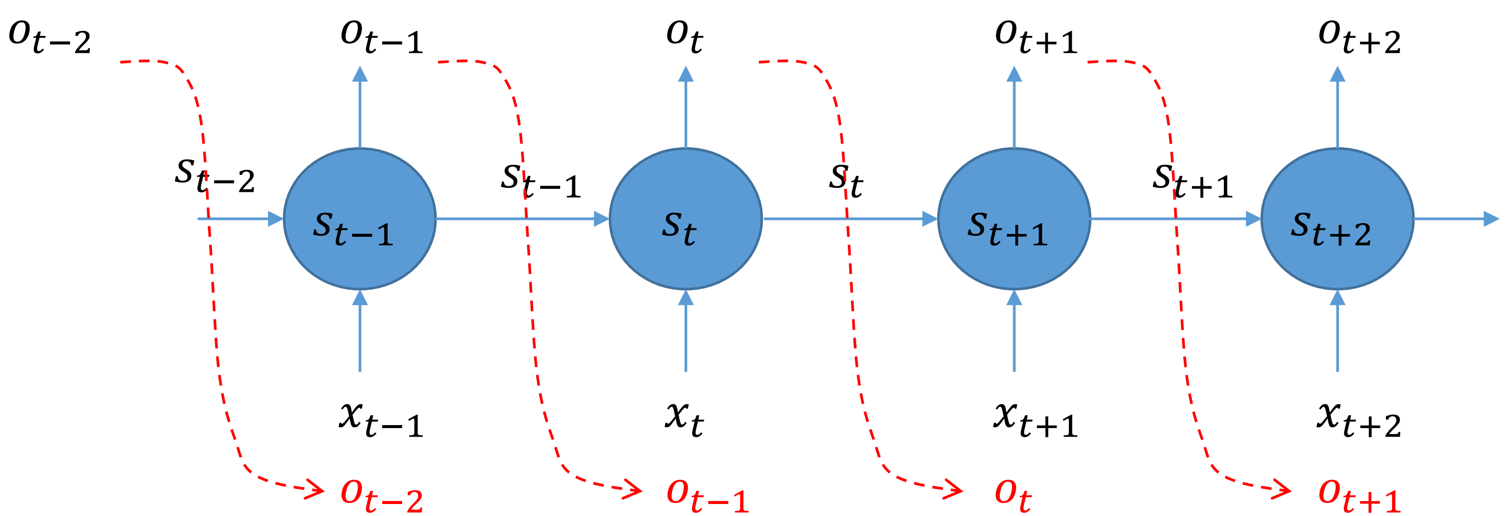
What RNN Really Model?

- Basically, modeling conditional probability
  - $P(y_t|x_{1:t-1})$

- We are always involved in some prediction issue, that is, be given some observations and use it to predict the future.
  - Language modeling/Stock prize prediction
  - Speech Recognition
  - Machine Translation
  - Etc.

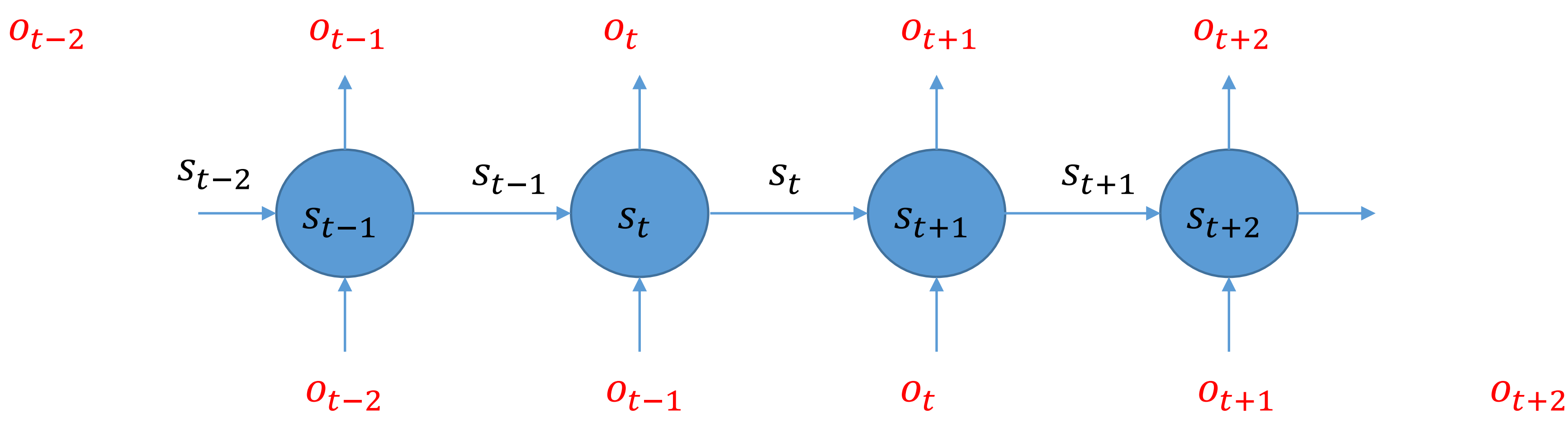
Case 1: Language or Stock Prize Modeling

- Output feeds to next-time-step input.
  - $x_t = o_{t-1}$
  - $s_t = f(s_{t-1}, x_t)$
  - $o_t = g(s_t)$



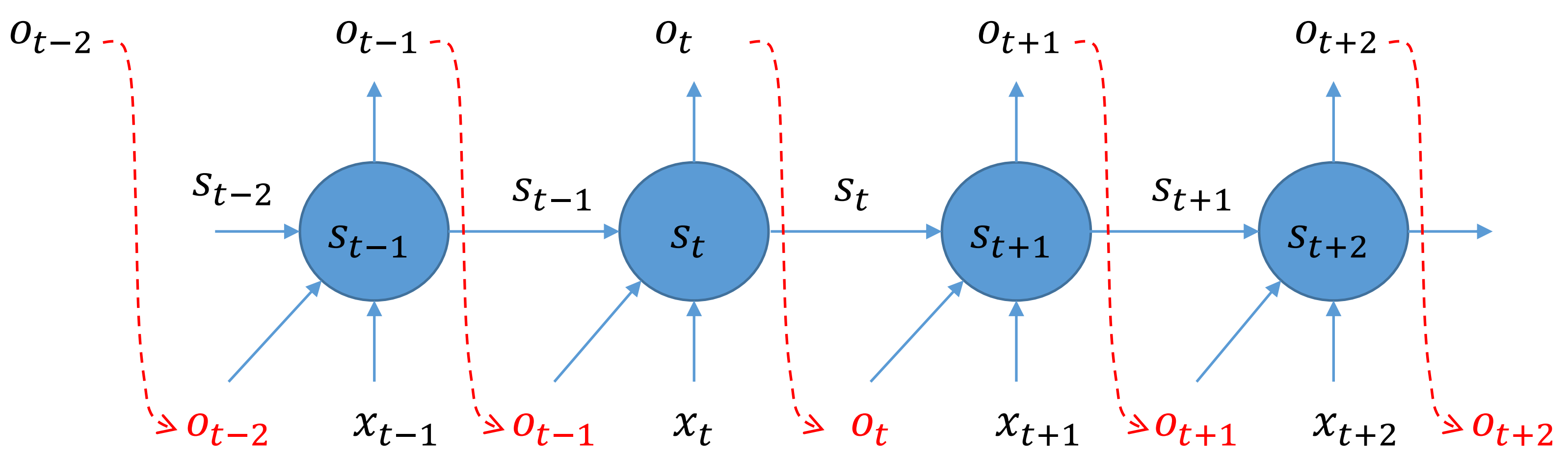
Case 1: Language or Stock Prize Modeling

- Input output sequences are the same
    - Same length
    - Identical symbols
    - $P(x_t|x_{1:t-1})$



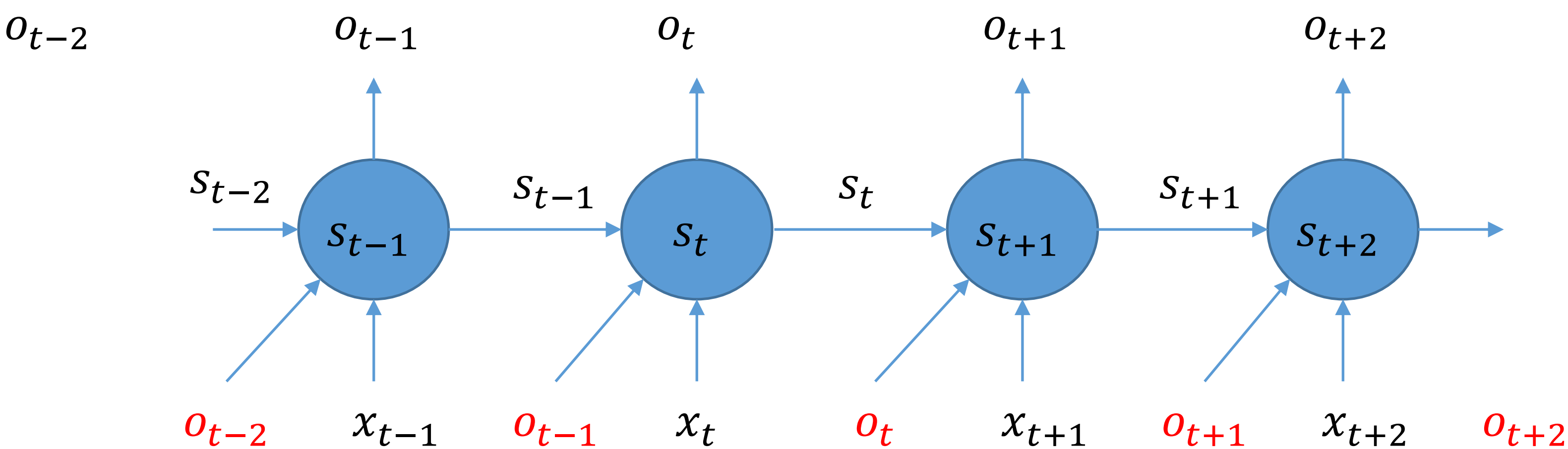
Case 2: Speech Recognition

- Output feeds to next-time-step input with new input at that time step
  - $s_t = f(o_{t-1}, s_{t-1}, x_t)$
  - $o_t = g(s_t)$
  - Why we need feedback from previous output?



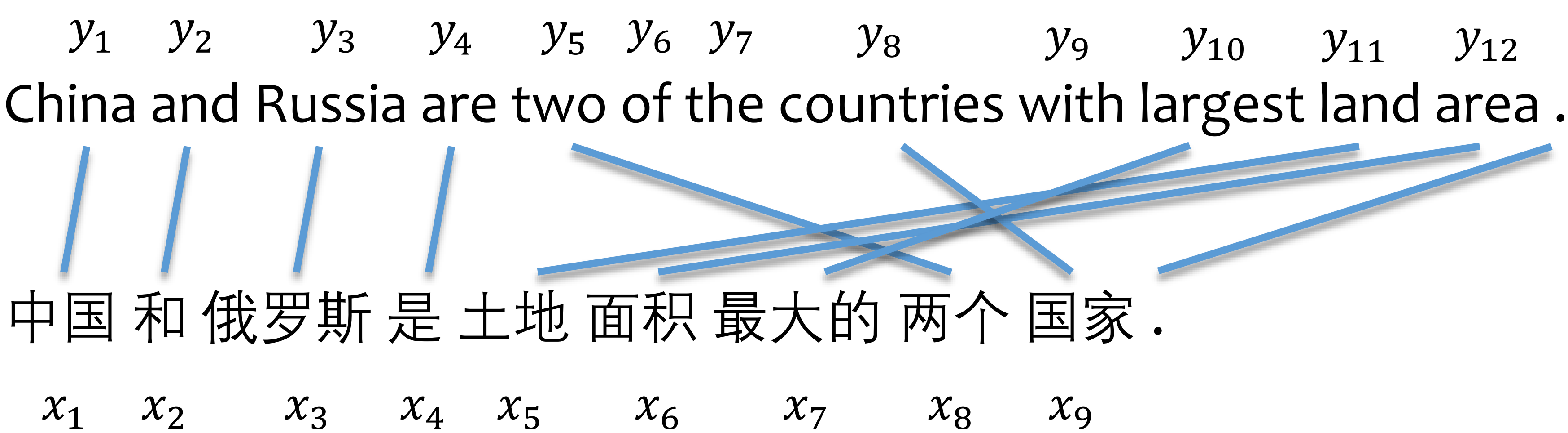
Case 2: Speech Recognition

- Input output sequences are not the same
  - But same length
  - Different modality, i.e. speech signals, word symbols



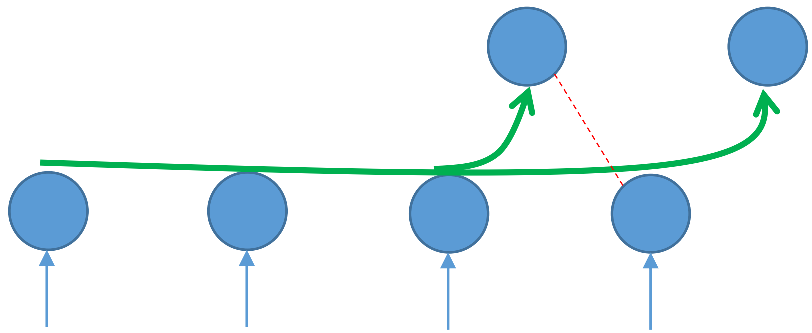
Case 3: Machine Translation

- Machine Translation is more flexible.
  - Suppose MT system is trying to translate the following Chinese to English.
  - x_i does not necessarily correspond to y_i, because of permutated ordering, and different length.

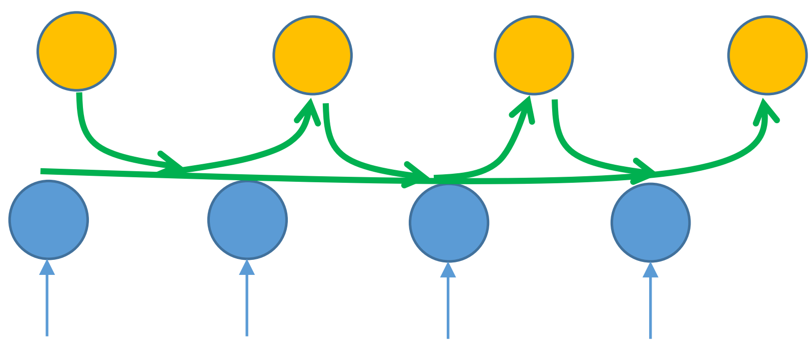


Taken an Information Flow View, Again!

Language modeling



Speech recognition

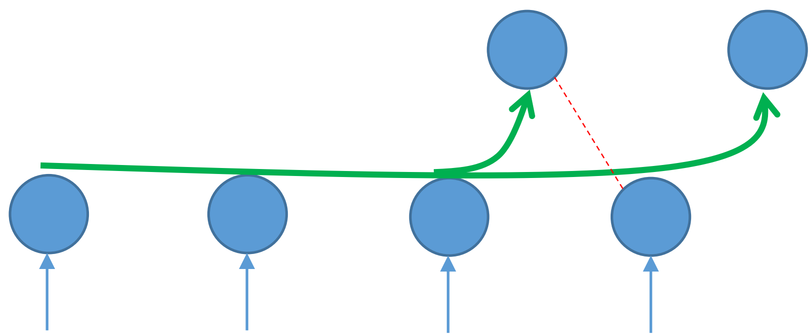


Machine Translation

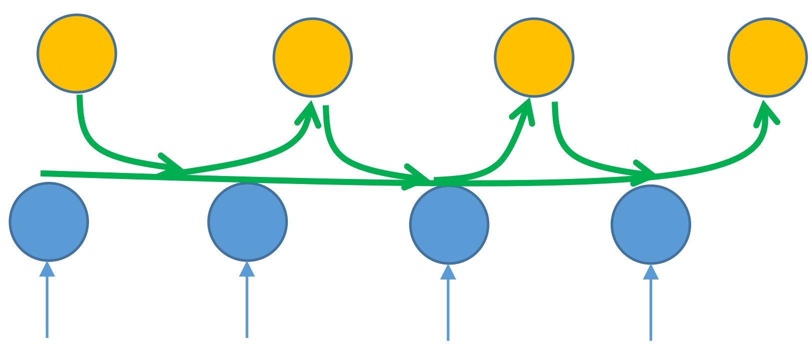
?

Taken an Information Flow View, Again!

Language modeling



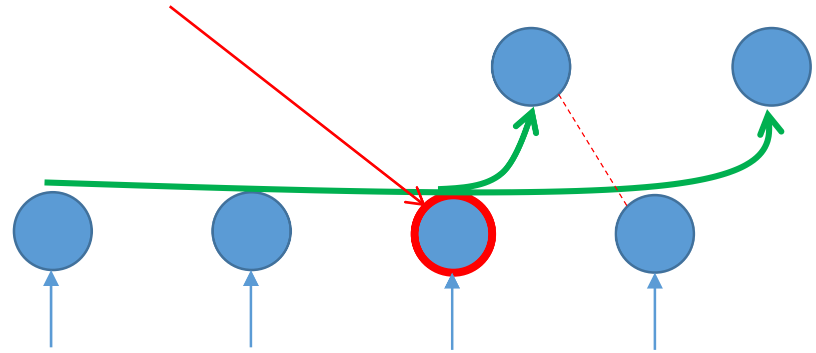
Speech recognition



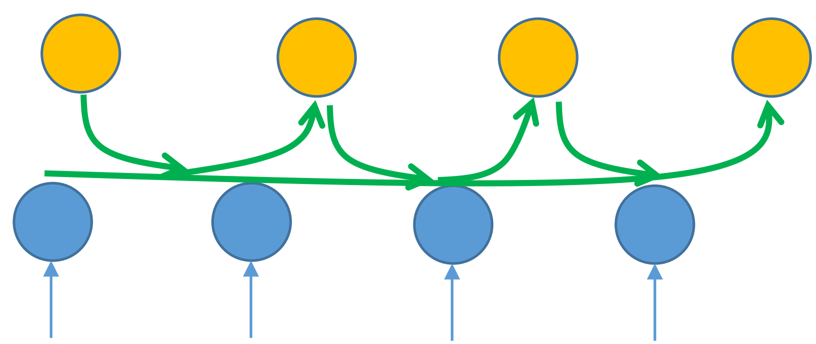
Taken an Information Flow View, Again!

What information does this node have?

Language modeling



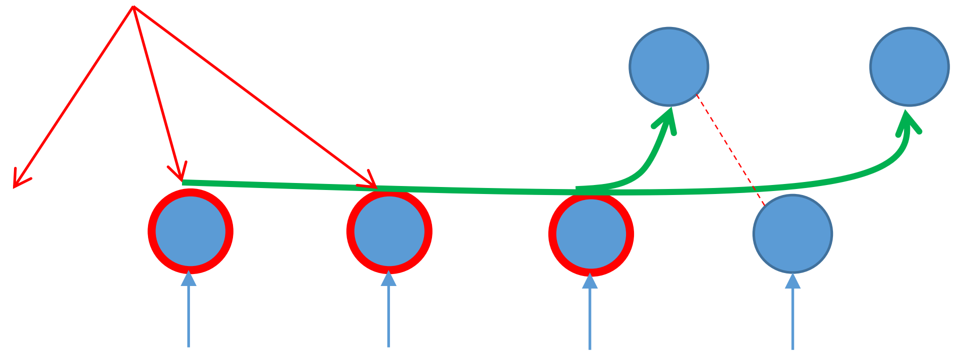
Speech recognition



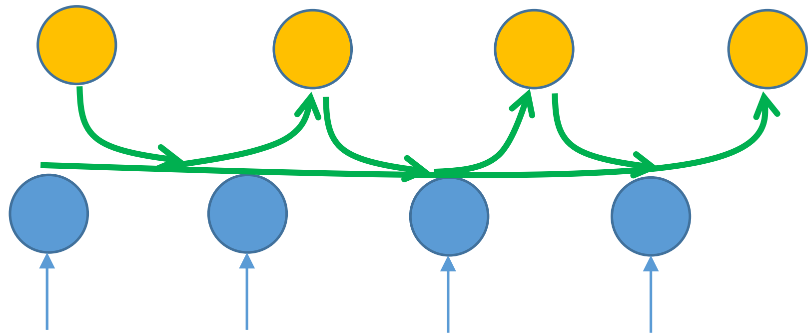
Taken an Information Flow View, Again!

All past hidden nodes.

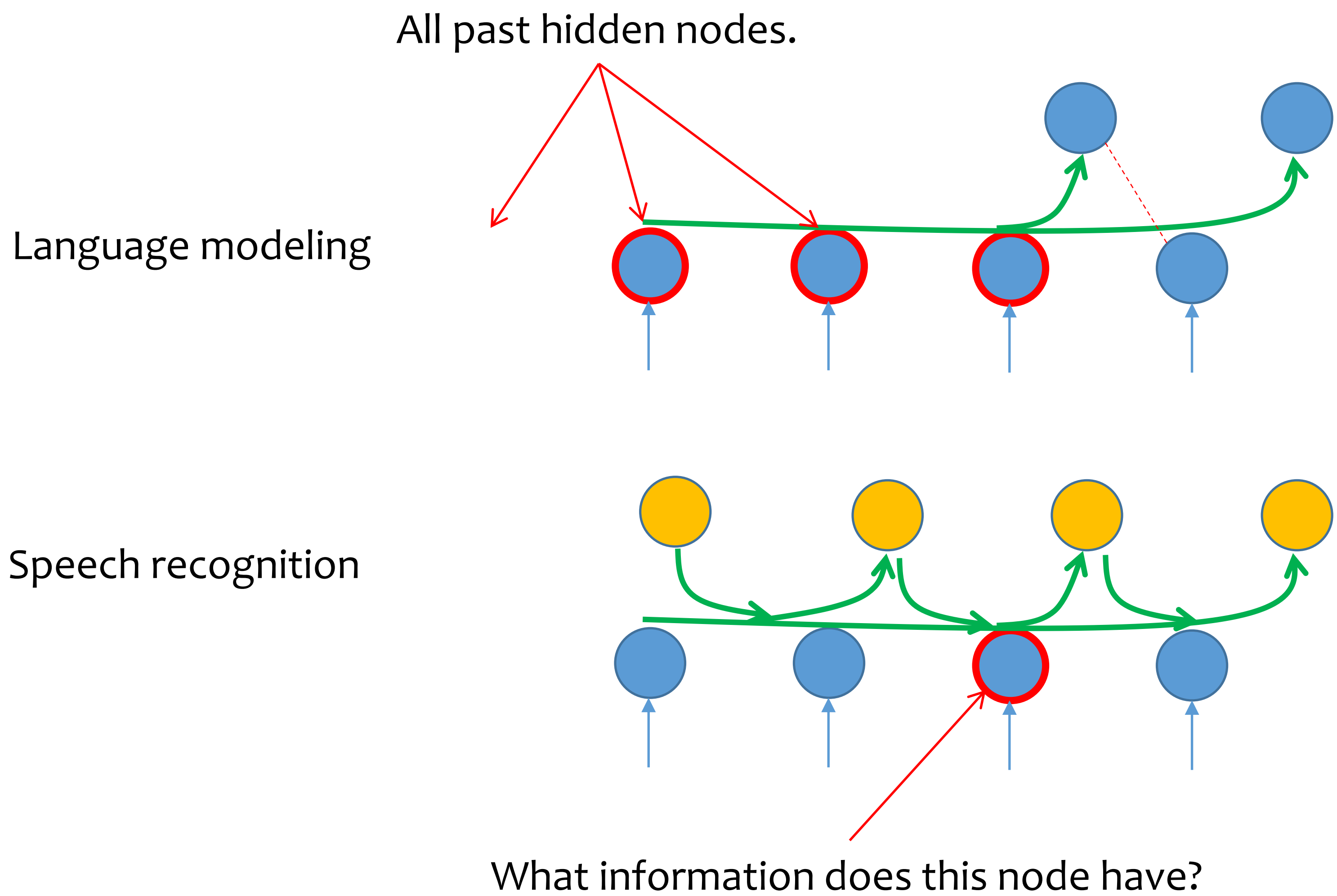
Language modeling



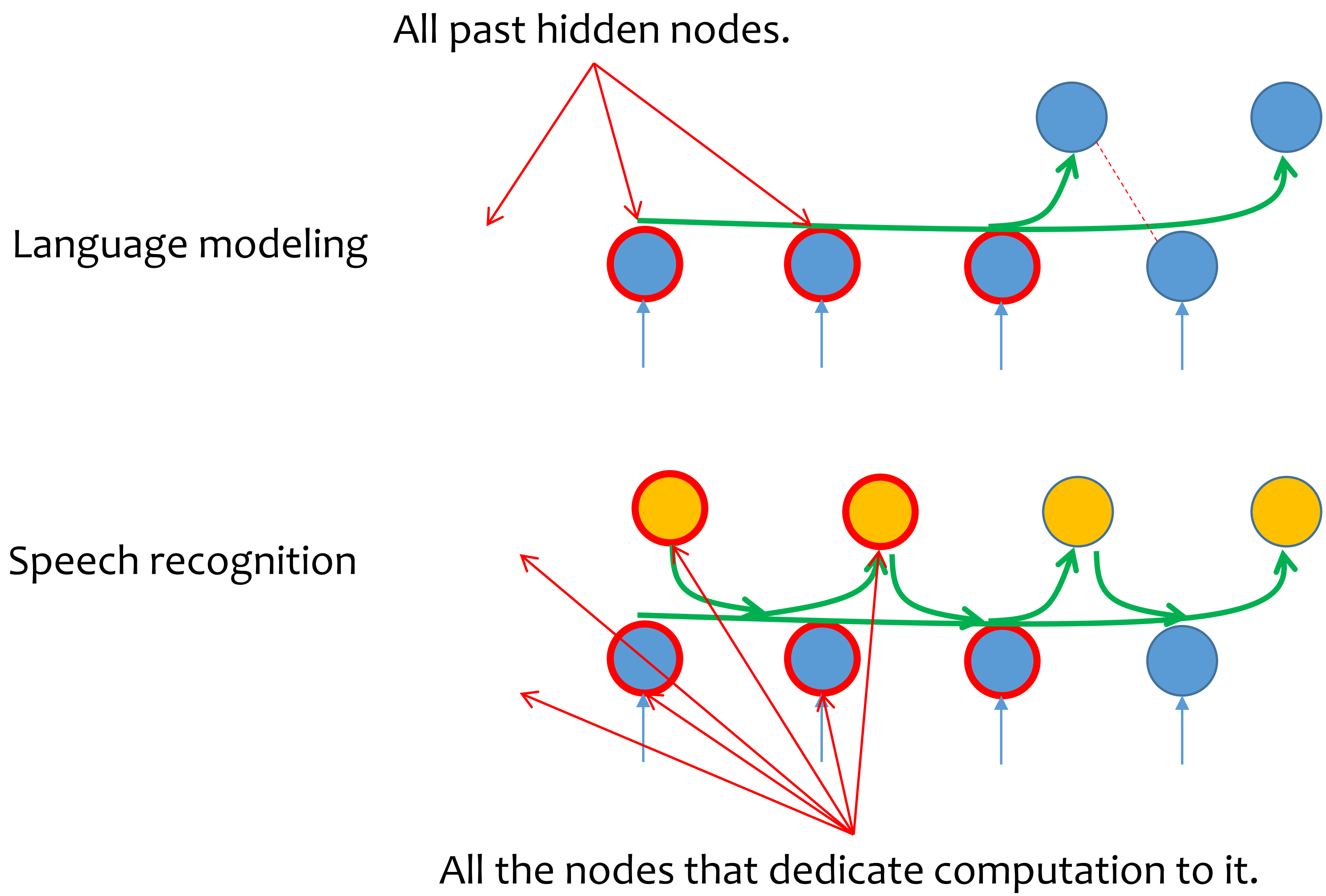
Speech recognition



Taken an Information Flow View, Again!



Taken an Information Flow View, Again!

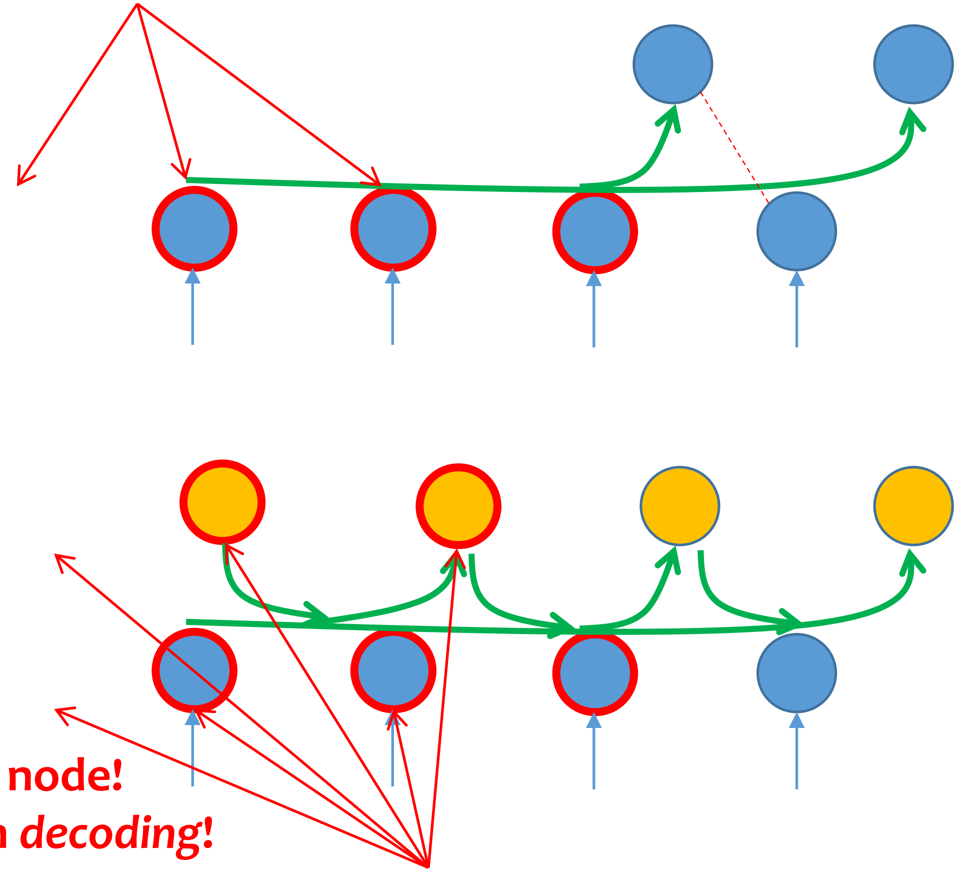


Taken an Information Flow View, Again!

All past hidden nodes.

Language modeling

Speech recognition



- **We call the info *encoded* in that node!**
- **We call the step-wise prediction *decoding*!**

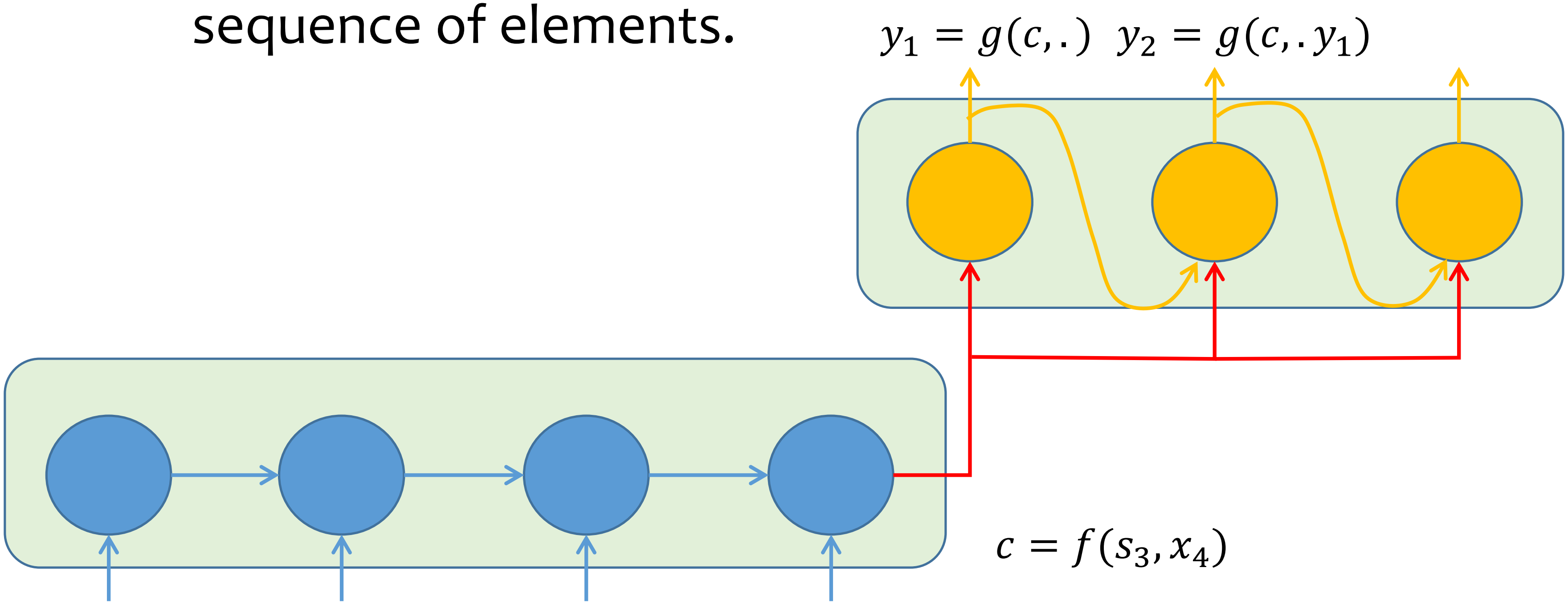
All the nodes that dedicate computation to it.

Rethink about MT

- As with previous discussion, a direct way of doing MT with RNN is:
  - to first *encode* the whole information of the source sentence into an information carrier.
  - Then use the information carrier to *decode* each word of the target sentence.

Encoder-Decoder Architecture

- Basic Encoder-Decoder Architecture is typically
  - composed of two RNNs,
  - with the first one to encode a sequence of elements,
  - the second use the encoded info to decode another sequence of elements.



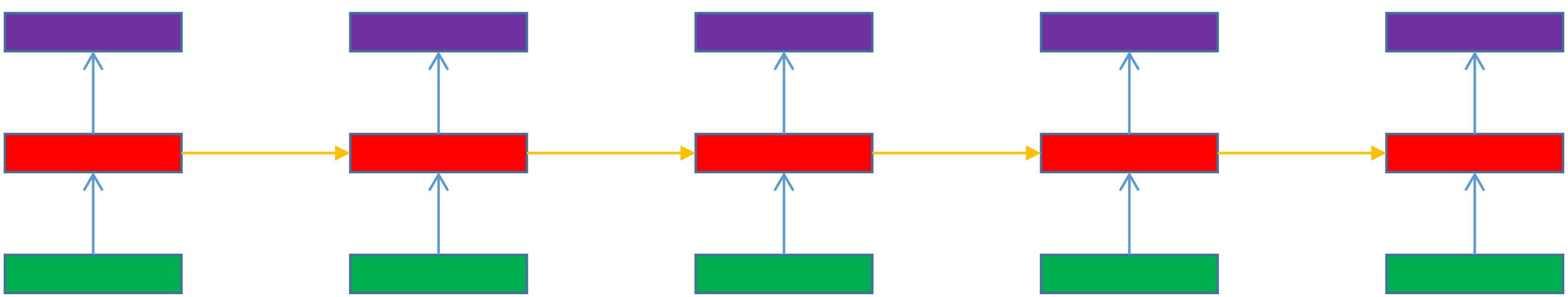
Encoder-Decoder Architecture

- What we can do with Encoder-Decoder
  - Machine Translation
    - Encoder, Decoder are both RNNs.
  - Image Captioning
  - Summarization/Simplification
  - Parsing
  - Etc.

Bonus I: Intermediate Info Carrier

- While we use RNN to process a sequence of length L, there will be L hidden state vectors as byproduct.

- These are *information carriers* that are rich representation about their around neighborhood.



Bonus II: Attention Mechanism

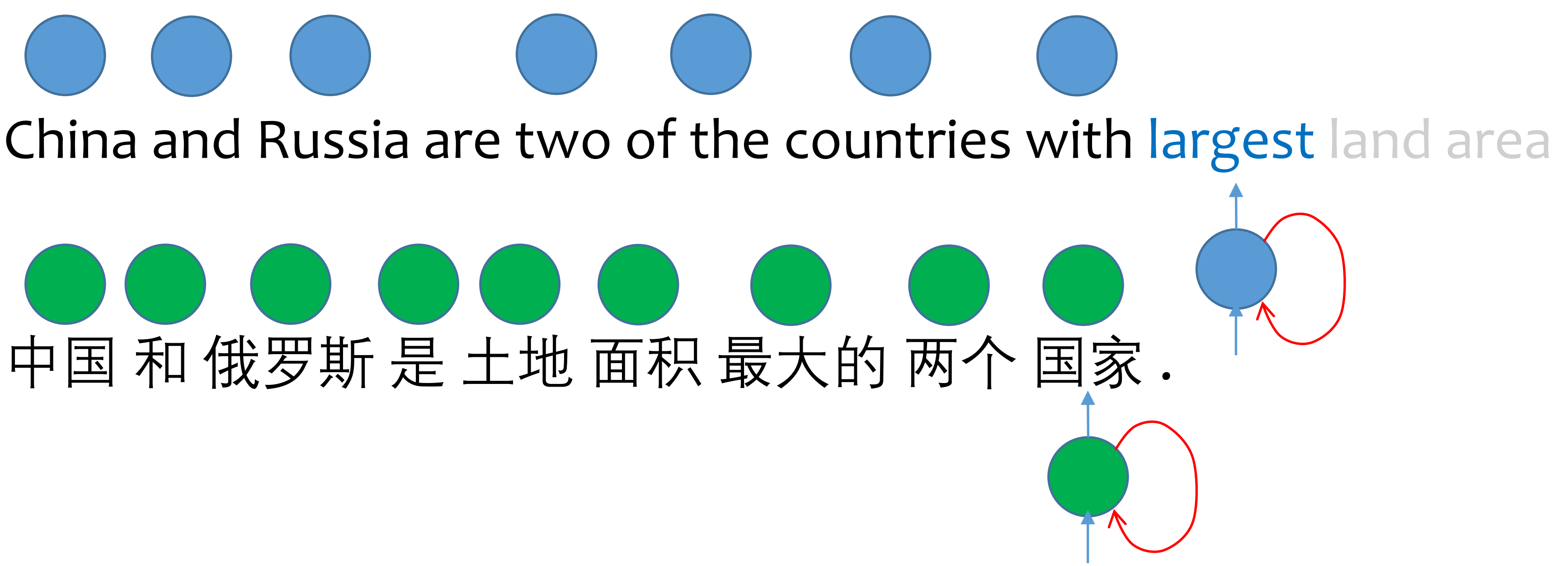
- The term *attention* is from Cognitive Science.
  - From Wikipedia
    - *"Attention is the behavioral and cognitive process of selectively concentrating on a discrete aspect of information, whether deemed subjective or objective, while ignoring other perceivable information."*
- Take machine translation as an example.

China and Russia are two of the countries with largest land area .

中国 和 俄罗斯 是 土地 面积 最大的 两个 国家 .

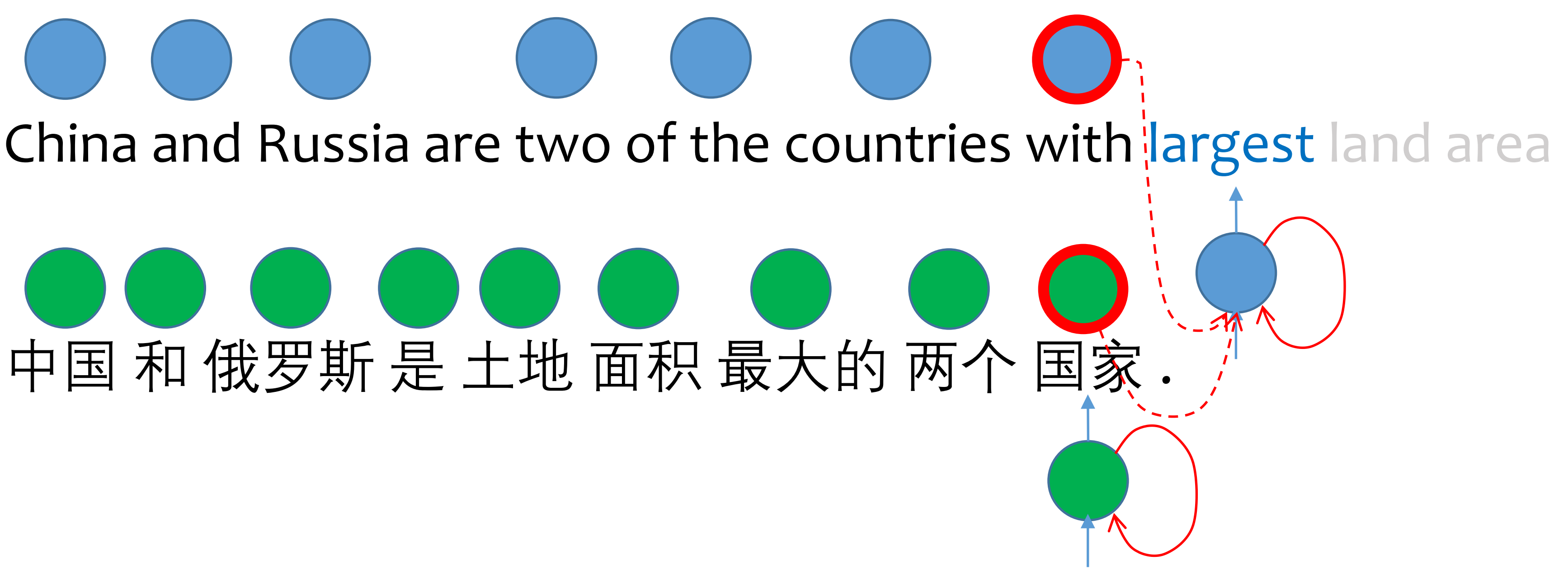
Bonus II: Attention Mechanism

- We use a traditional Encoder-Decoder framework, and see if there is any improvement.
  - Green and blue balls are *intermediate info* (hidden state).
  - Now, RNN is going to predict '*largest*' as next candidate.



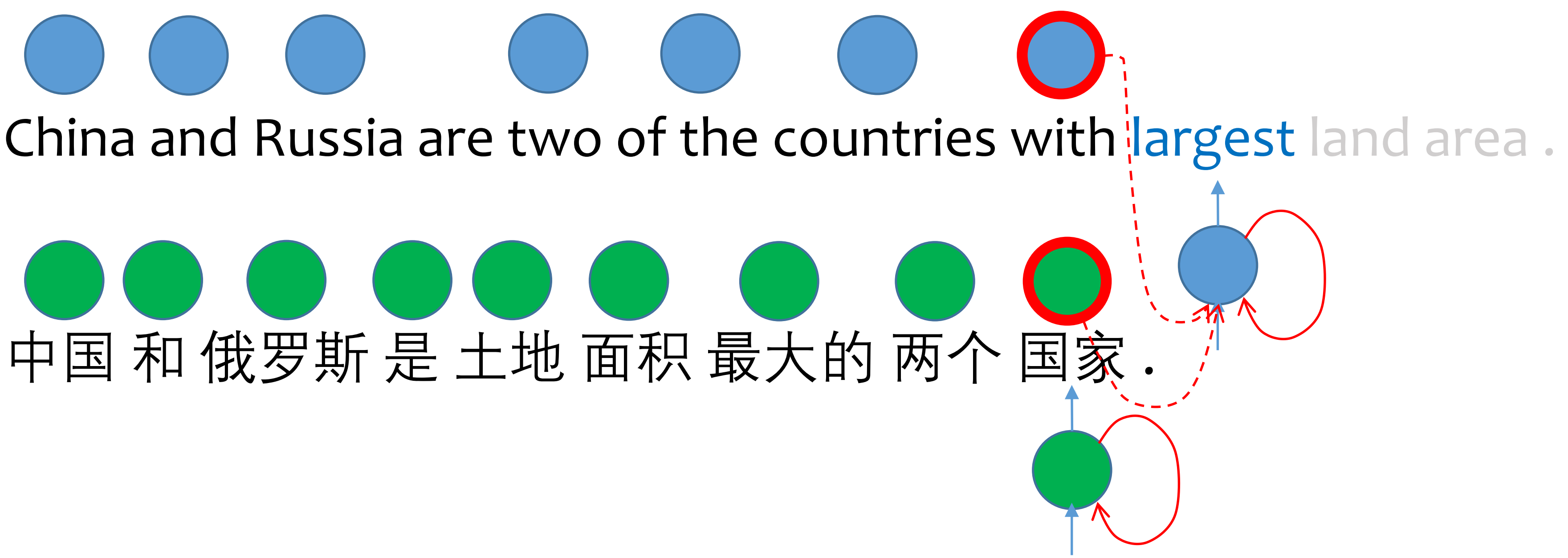
Bonus II: Attention Mechanism

- It is very easy! Just take the *previous* hidden and the *global* encoded info of the source sentence



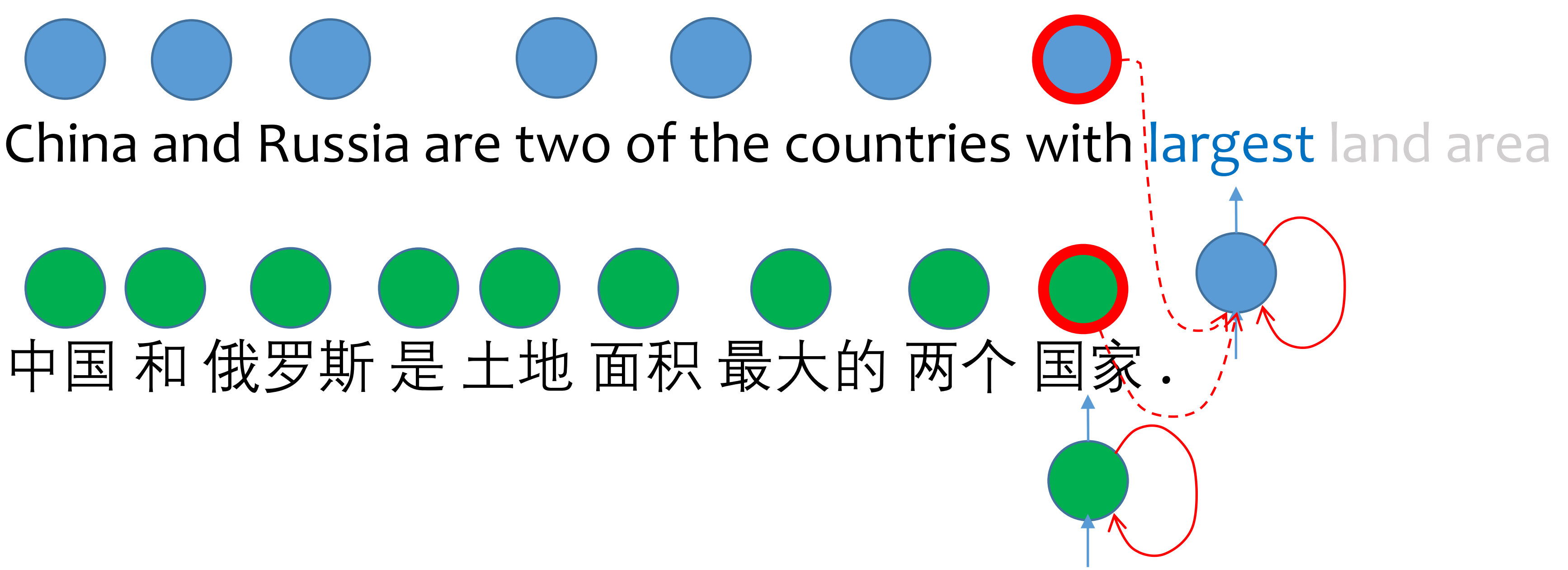
Bonus II: Attention Mechanism

- What is the disadvantage of using all the info?



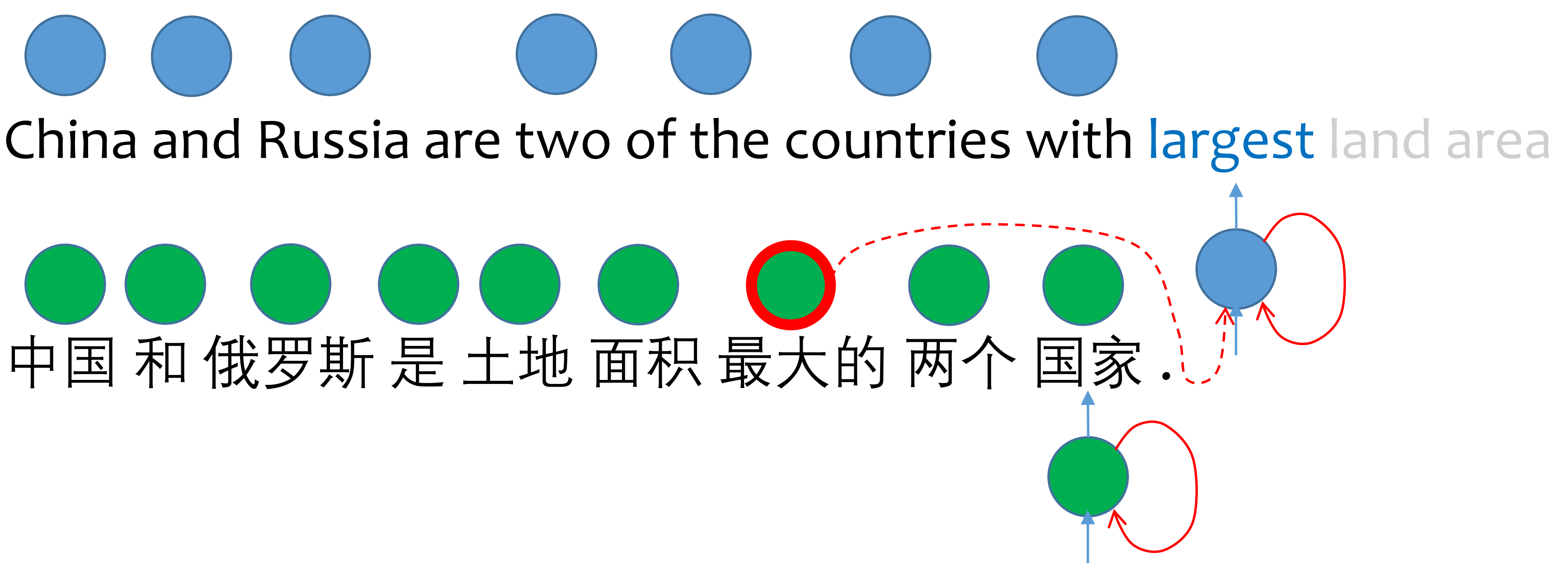
Bonus II: Attention Mechanism

- What is the disadvantage of using all the info?
  - The problem of **CAPACITY!**



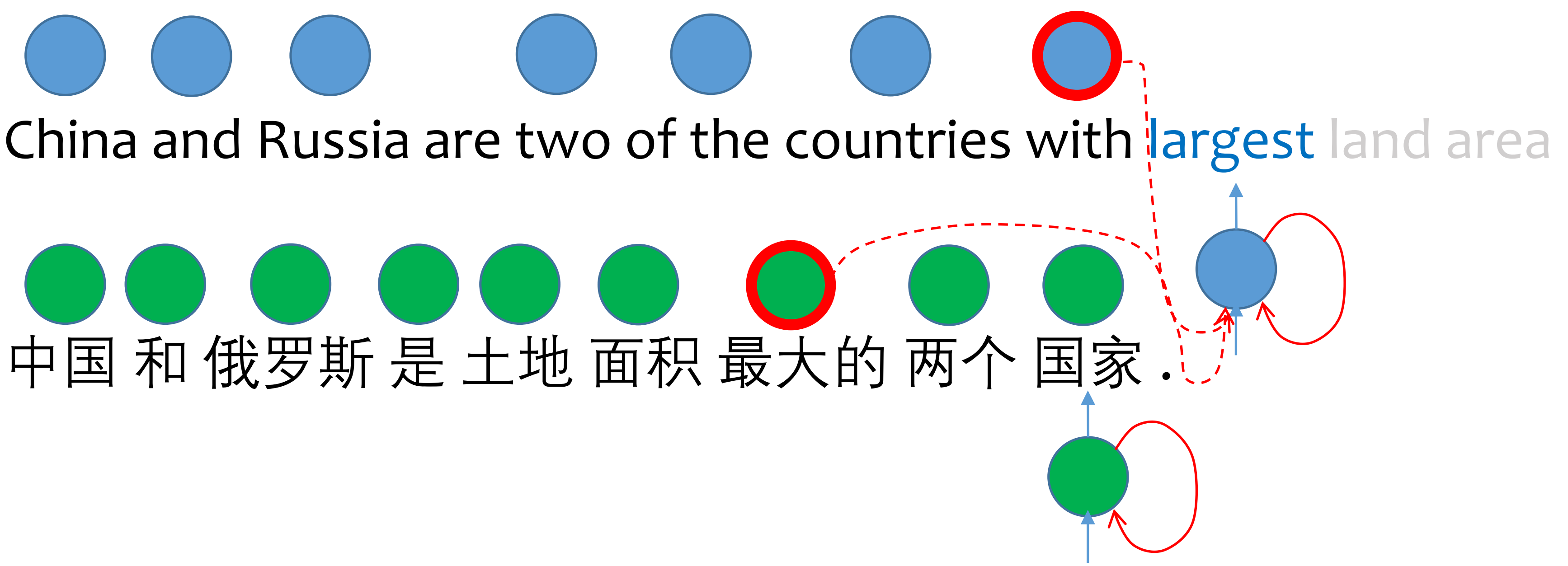
Bonus II: Attention Mechanism

- If possible, attended on the most relevant source word!



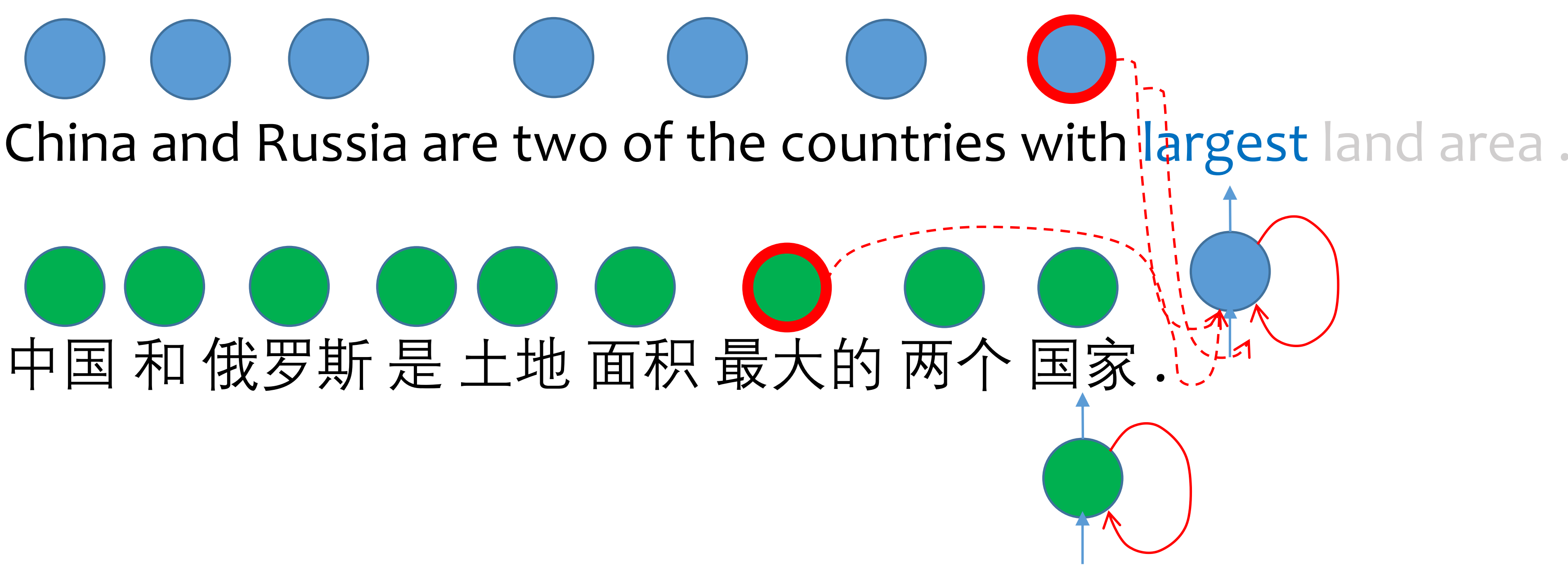
Bonus II: Attention Mechanism

- If possible, attended on the most relevant source word!
- And the previous word to guarantee coherence.



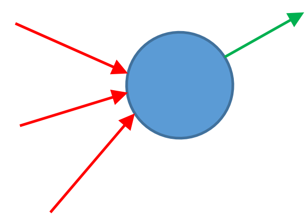
Bonus II: Attention Mechanism

- How to do this?

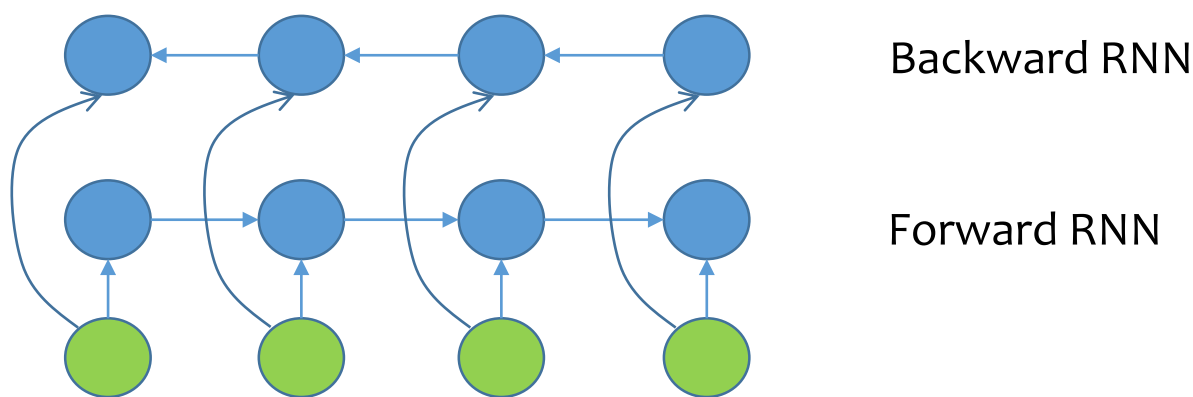


Bi-directional RNNs

- Recap Composition, now you can think of it as
  - Information fusion when computation happens
  - 3 in-come info flow
  - 1 out-go info flow



- Information flow order may matter
  - Because composition encode order information.

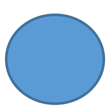


Outline

- Sequence with Order
  - Unfolding Computational Graph
- Recurrent Neural Network
- Recursive Neural Network
- Challenge of Long-Term Dependencies
  - Long Short-term Memory Unit
  - Gated Recurrent Unit
- Explicit Memory
  - Memory Network (Weston et al)
  - Neural Turing Machine (Graves et al)

Recursive Neural Network

- Recursive Neural Network is a *golden way* to do **composition** over hypothetical *directed* structure.



Outline

- Sequence with Order
  - Unfolding Computational Graph
- Recurrent Neural Network
- Recursive Neural Network
- Challenge of Long-Term Dependencies
  - Long Short-term Memory Unit
  - Gated Recurrent Unit
- Explicit Memory
  - Memory Network (Weston et al)
  - Neural Turing Machine (Graves et al)

Long-Term Dependency

- Let’s do a toy sequence binary classification question.
  - The experiment is described in Bengio 94.

- The experimented RNN is asked to correctly *classify* different sequences, many!
- The class is *only* determined by the first 3 letters.
  - aba class 1
  - abc class 2
- The rest of the letters are like *noise*.

Learning long-term dependencies with gradient descent is difficult, Bengio 1994.

Long-Term Dependency

- The experiment is *simple*, just use an RNN to *encode* the whole sequence to a vector value.

- Then, *classify* this vector information carrier.

- However, RNN **struggles** to have a high accuracy especially when the sequence of *noise* become *longer*.
  - Take a long time to convergent.

Long-Term Dependency

Definition (Long-Term Dependency)

- A task displays long-term dependencies if computation of the desired output at time t depends on input presented at an earlier time $\tau \ll t$.

- If RNN is going to learn t depends on τ, then the weight gradient *at τ time step* should be more *sensitive* to the loss *at t time step*.

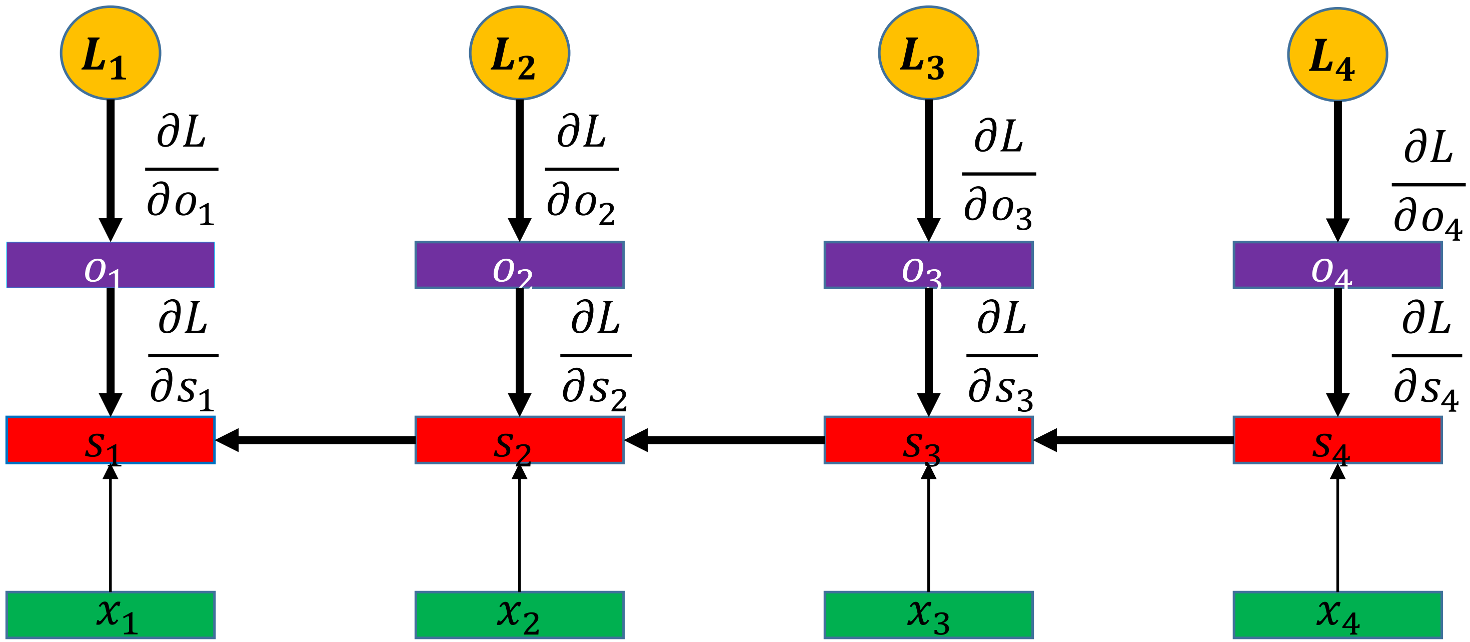
- That is how we wish our life is ☺

Here needs a ‘HOWEVER’ word

- Vanishing or exploding of gradient through long-term weights.

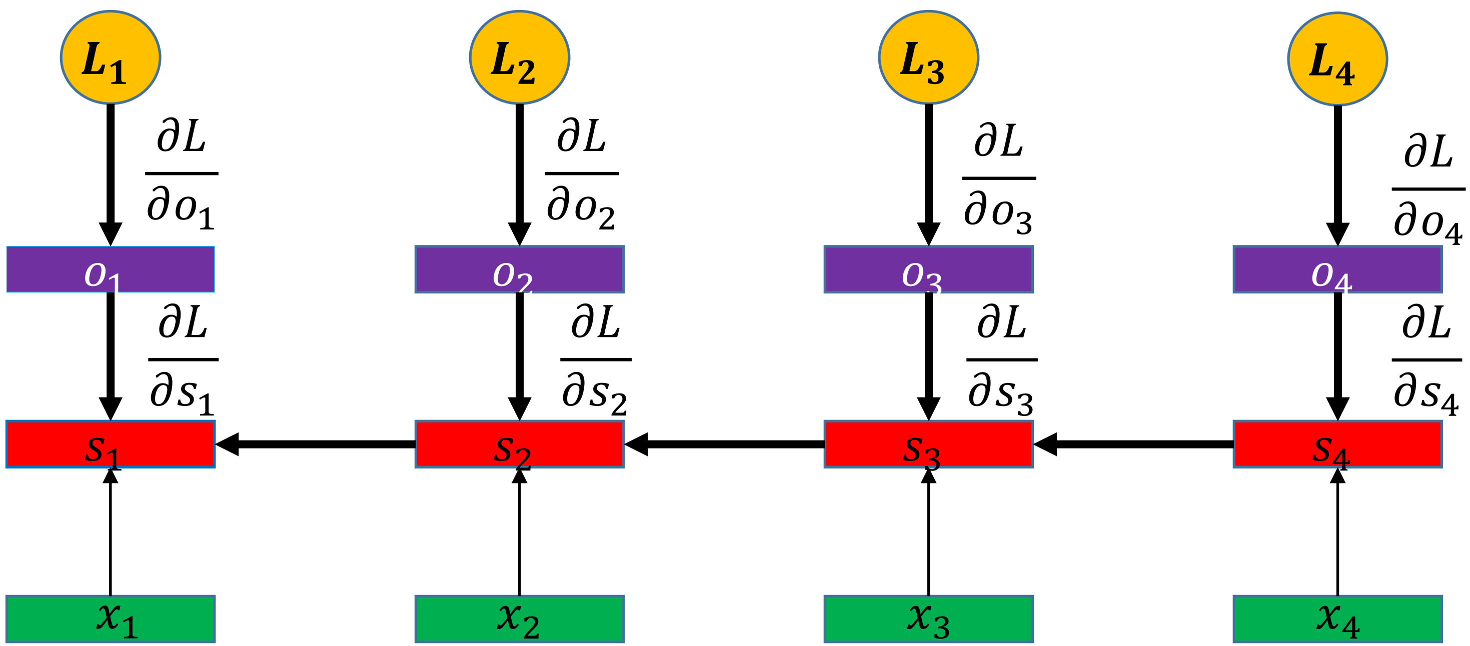
- Now, let us take a linear example for toy proof!
  - We assume all the transformation be linear
  - That is:
    - $s_t = W s_{t-1} + U x_t + b$
    - No non-linearity

Backpropagation through Time



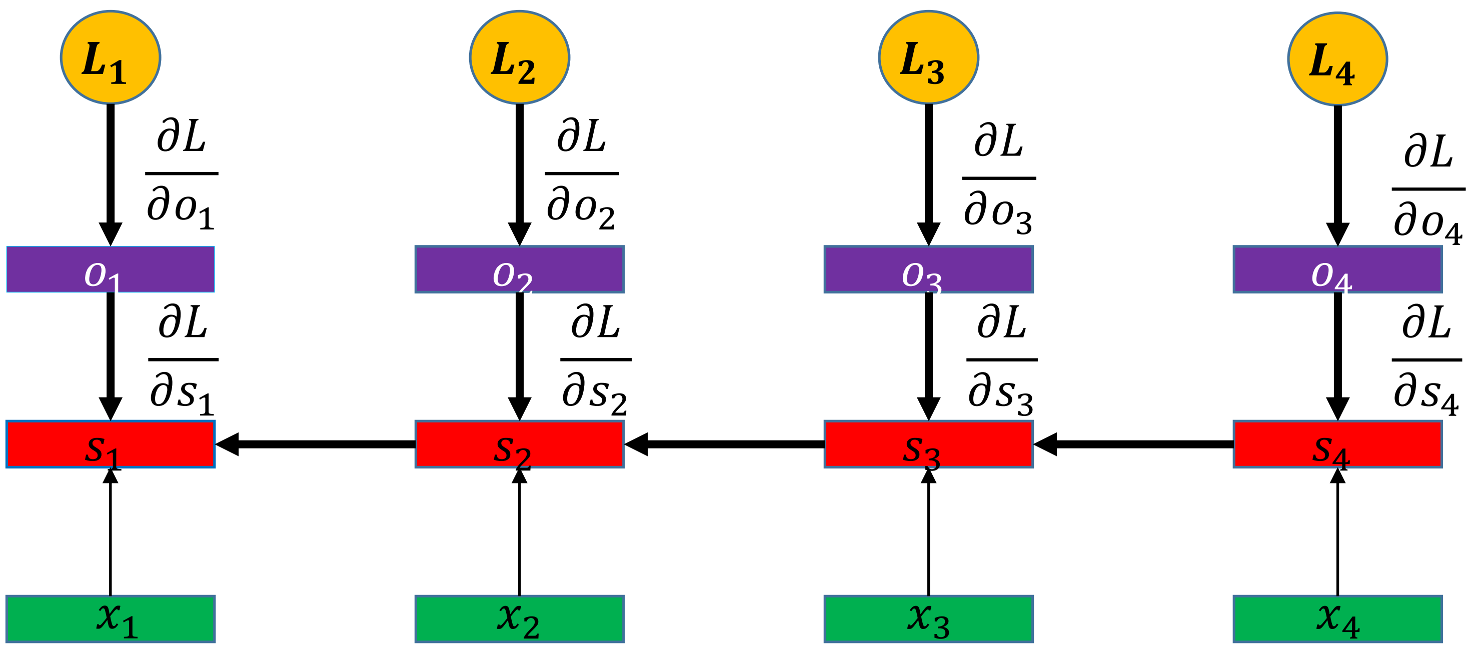
- Now, let us assume $t = 4$ depends on $\tau = 1$, with linear transform.
- $\frac{\partial L}{\partial s_1} = \frac{\partial L}{\partial s4} \frac{\partial s_4}{\partial s_3} \frac{\partial s_3}{\partial s_2} \frac{\partial s_2}{\partial s_1}$, every term is a Jacobian matrix, and square.
- Since $s_t = W s_{t-1} + U x_t + b$, $\frac{\partial s_i}{\partial s_{i-1}} = W$

Backpropagation through Time



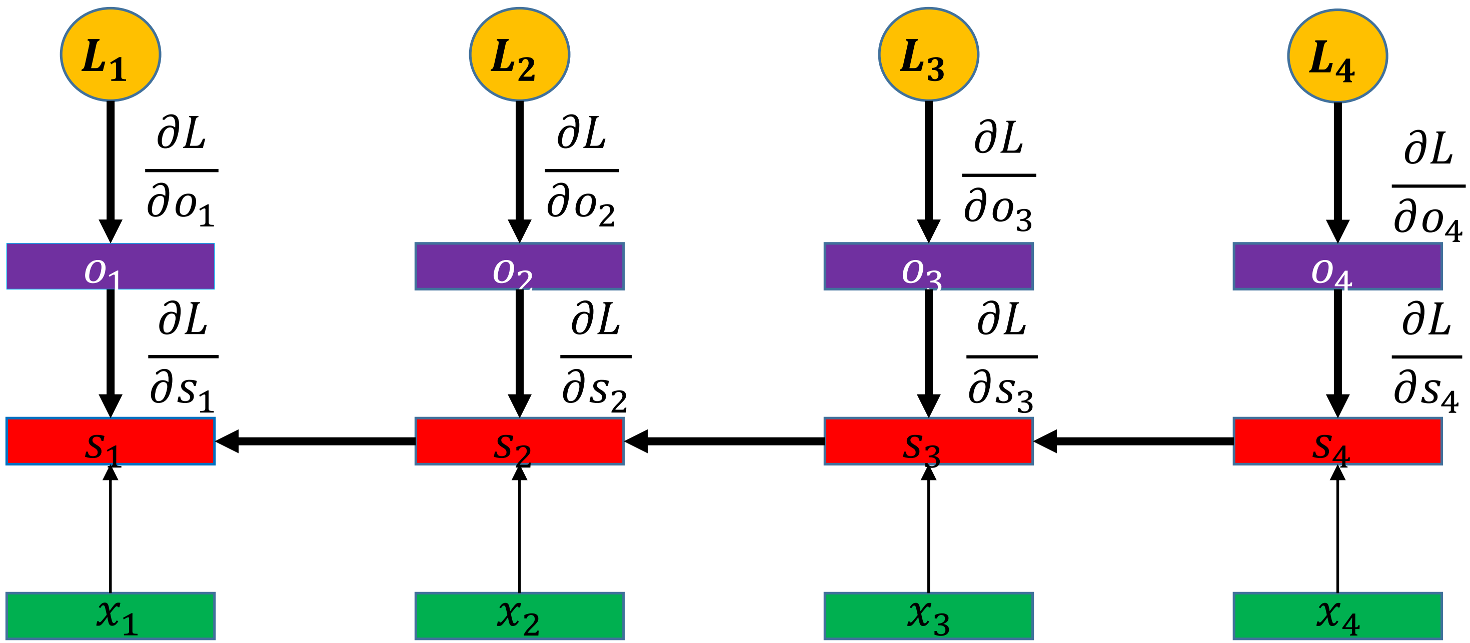
- Now, let us assume $t = 4$ depends on $\tau = 1$, with linear transform.
- $\frac{\partial L}{\partial s_1} = \frac{\partial L}{\partial s4} \frac{\partial s_4}{\partial s_3} \frac{\partial s_3}{\partial s_2} \frac{\partial s_2}{\partial s_1}$, every term is a Jacobian matrix, and square.
- Since $s_t = W s_{t-1} + U x_t + b, \frac{\partial s_i}{\partial s_{i-1}} = W \xRightarrow{Eigendecomp} U \Lambda U^T, UU^T = I$

Backpropagation through Time



- Now, let us assume $t = 4$ depends on $\tau = 1$, with linear transform.
- $\frac{\partial L}{\partial s_1} = \frac{\partial L}{\partial s4} \frac{\partial s_4}{\partial s_3} \frac{\partial s_3}{\partial s_2} \frac{\partial s_2}{\partial s_1} = \frac{\partial L}{\partial s_4} W^3 = \frac{\partial L}{\partial s_4} U \Lambda^3 U^T$.
- Since $s_t = W s_{t-1} + U x_t + b$, $\frac{\partial s_i}{\partial s_{i-1}} = W \xRightarrow{Eigendecomp} U \Lambda U^T, UU^T = I$

Vanishing & Exploding Gradient



- $\frac{\partial L}{\partial s_1} = \frac{\partial L}{\partial s_4} W^3 = \frac{\partial L}{\partial s_4} U \Lambda^3 U^T$
- Since Λ is *diagonal*, if the *largest* element is smaller than 1, every element of the Jacobian vanishes to *zero*, leads to Slow Learning.
- Vice, exploding to infinitely *large*, leads to Unstable Learning.

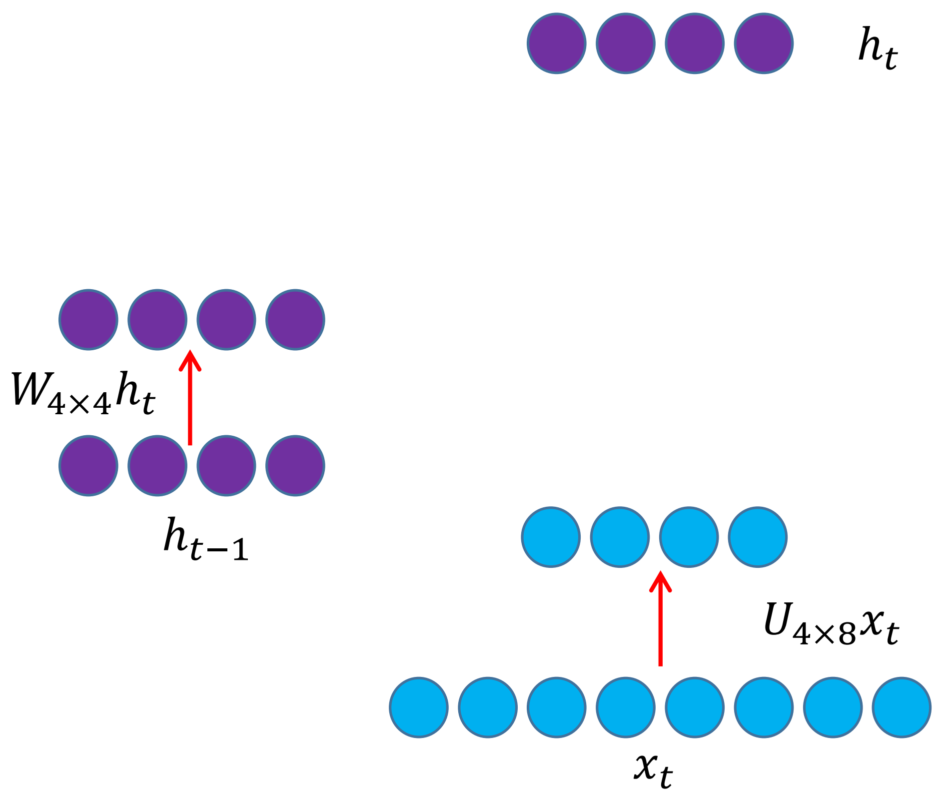
Long Short-Term Memory Unit

- How to overcome vanishing & exploding by ourselves?
  - Find the root of the problem!
  - $\frac{\partial L_t}{\partial s_\tau} = \frac{\partial L_t}{\partial s_t} U \Lambda^{t-\tau} U^T$, here Λ_{ii} less than or larger than 1
  - Easy! Fix it to one! How?
  - Change $s_t = \tanh(W s_{t-1} + U x_t + b)$ to
  - $s_t = \boldsymbol{(1-\alpha)} s_{t-1} + \boldsymbol{\alpha} \tanh(.)$

  - The red term is extremely clever design, since if $\alpha = 1$, s_t remain the same, which is an *identity transform* with gradient equal to 1.
  - LSTM is just more considerable, ‘he’ parameterize α to be adaptive to past history and current input ☺

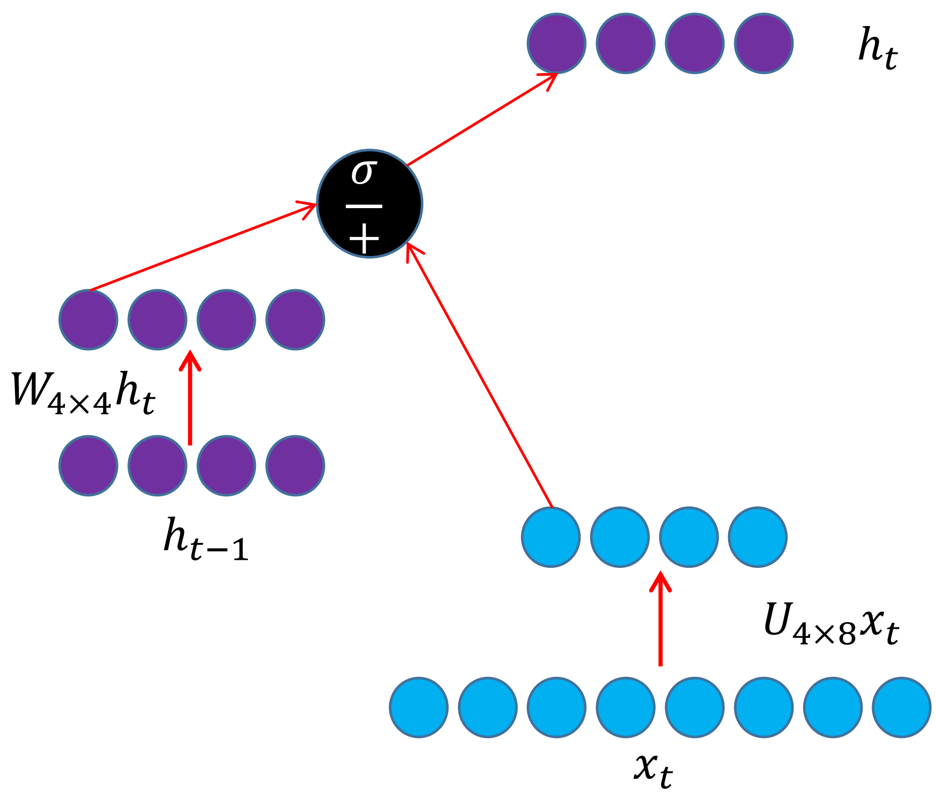
Long Short-Term Memory Unit

- Naïve RNN revisit with fine granularity view.



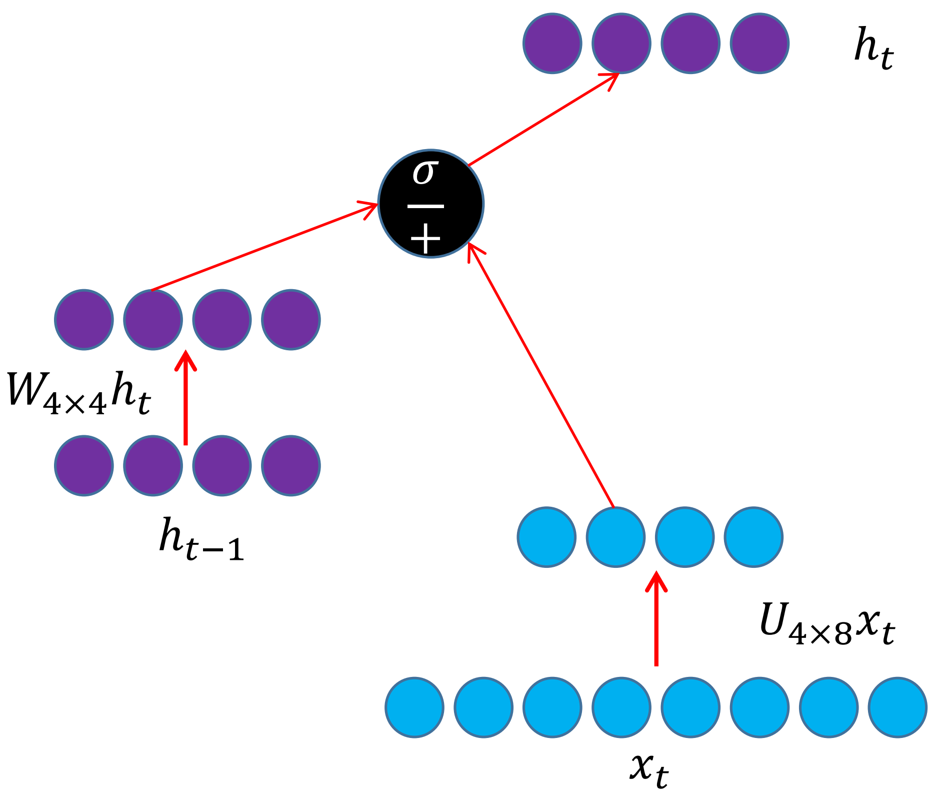
Long Short-Term Memory Unit

- Naïve RNN revisit with fine granularity view.



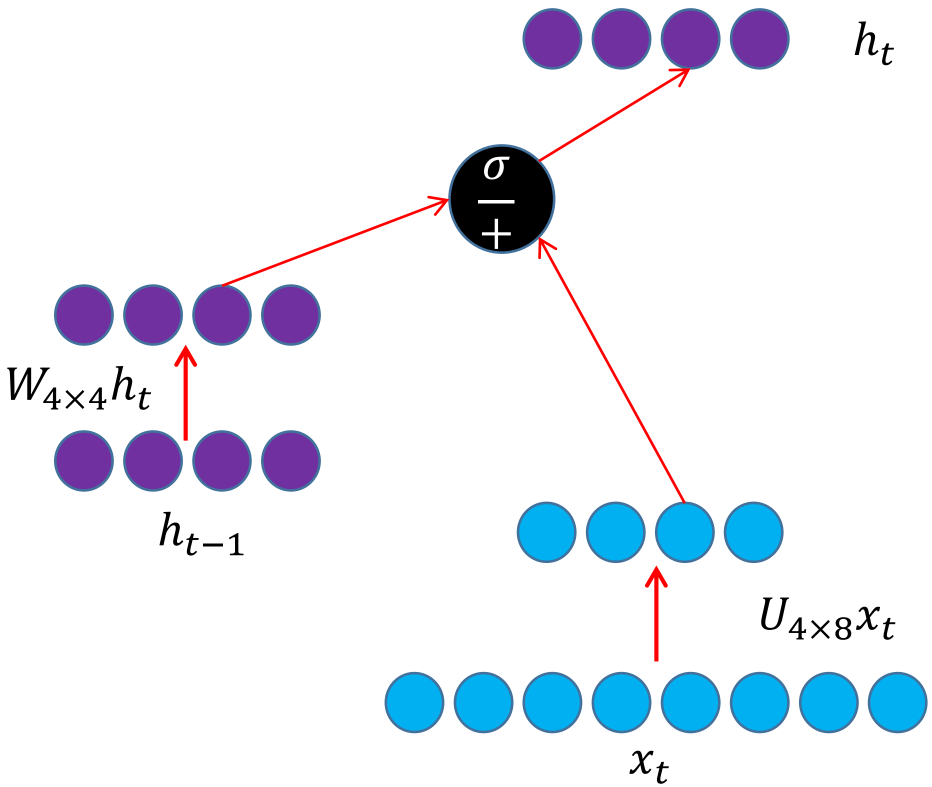
Long Short-Term Memory Unit

- Naïve RNN revisit with fine granularity view.



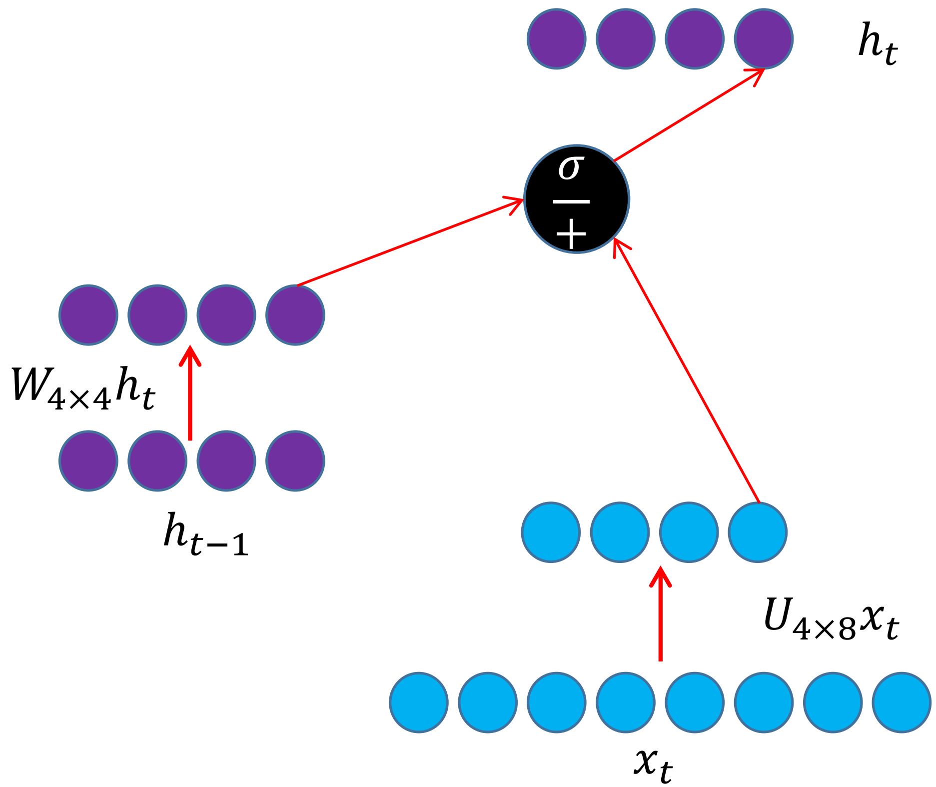
Long Short-Term Memory Unit

- Naïve RNN revisit with fine granularity view.



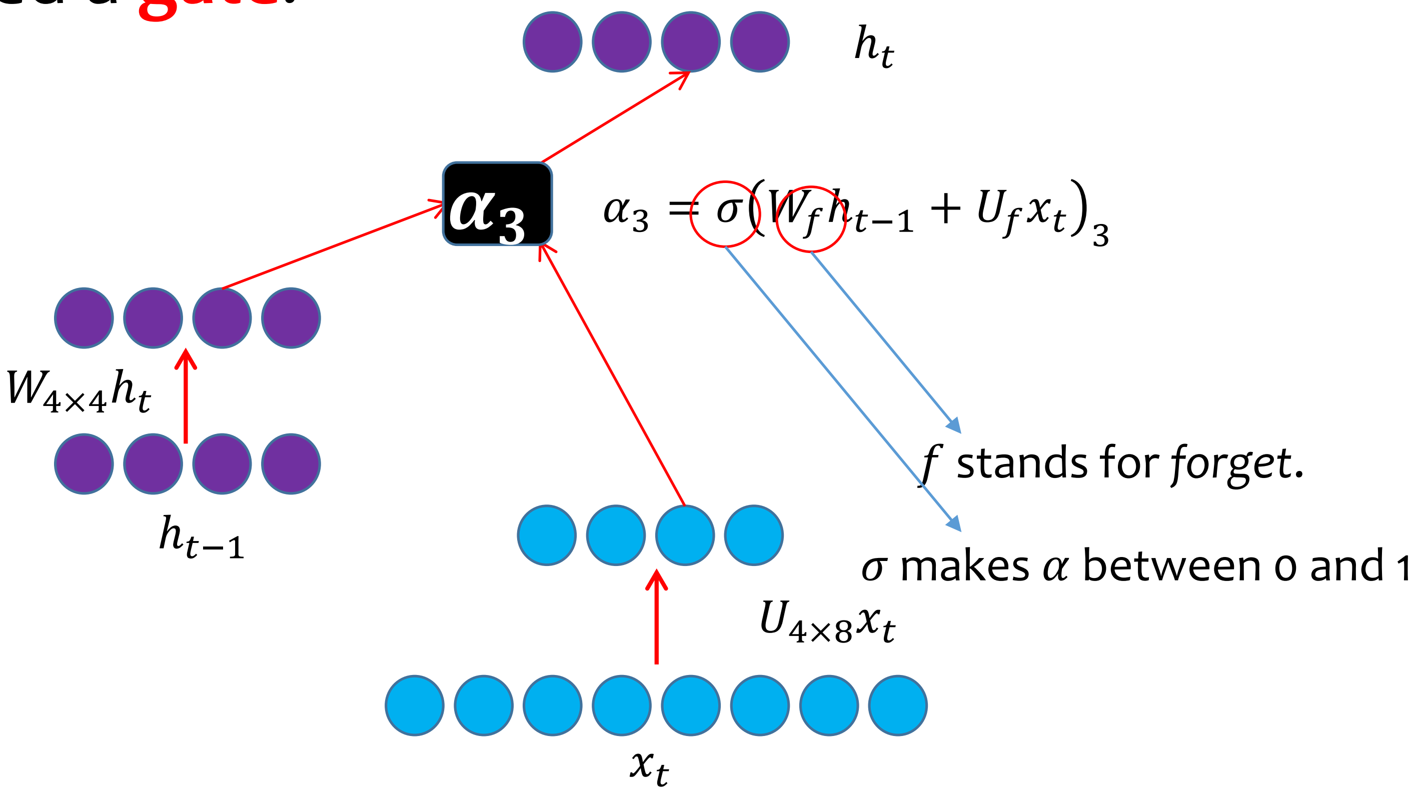
Long Short-Term Memory Unit

- Naïve RNN revisit with fine granularity view.



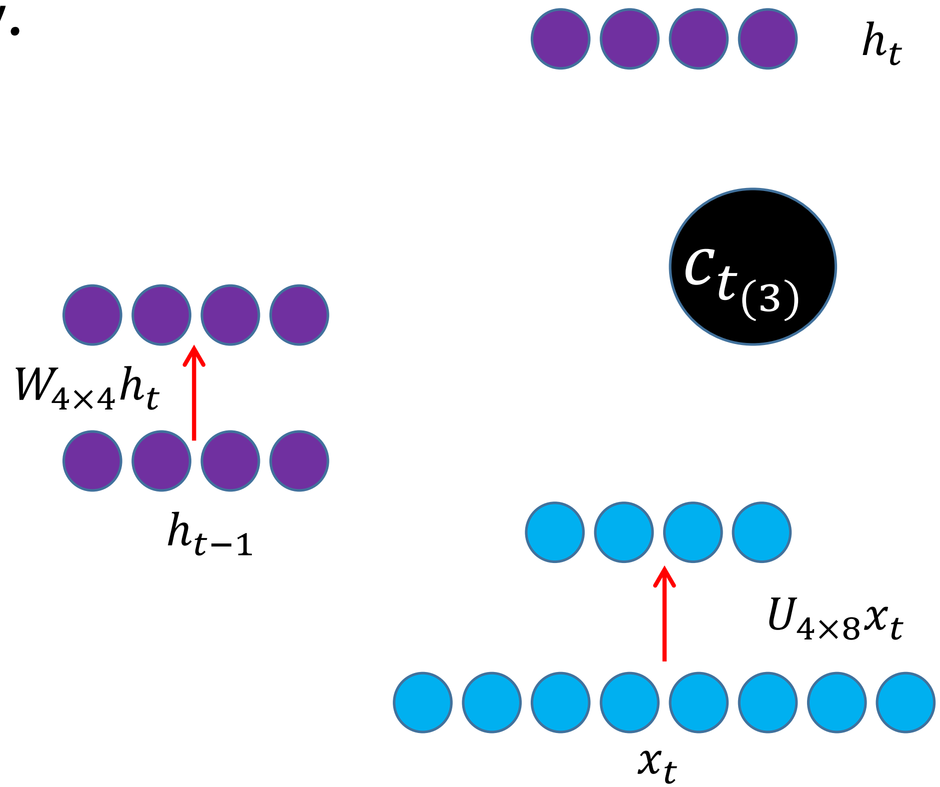
Long Short-Term Memory Unit

- Our solution: $h_t = \boldsymbol{\alpha} \odot h_{t-1} + (\mathbf{1} - \boldsymbol{\alpha}) \odot \tanh(.)$.
  - Since *element-wise*, it is like the following.
- α is called a **gate**.



Long Short-Term Memory Unit

- Instead, LSTM Unit has an explicit memory cell c_t besides h_t, which evolves over time.
- Moreover, LSTM Unit has **3 gates** to adaptively control info flow.



Long Short-Term Memory Unit

- c_t is a storage of info, which can be used in far away future.

- Every time step t, we pick some info stored in c_{t-1}, together with current input x_t, and previous hidden h_{t-1} which is used for predicting to make decisions:

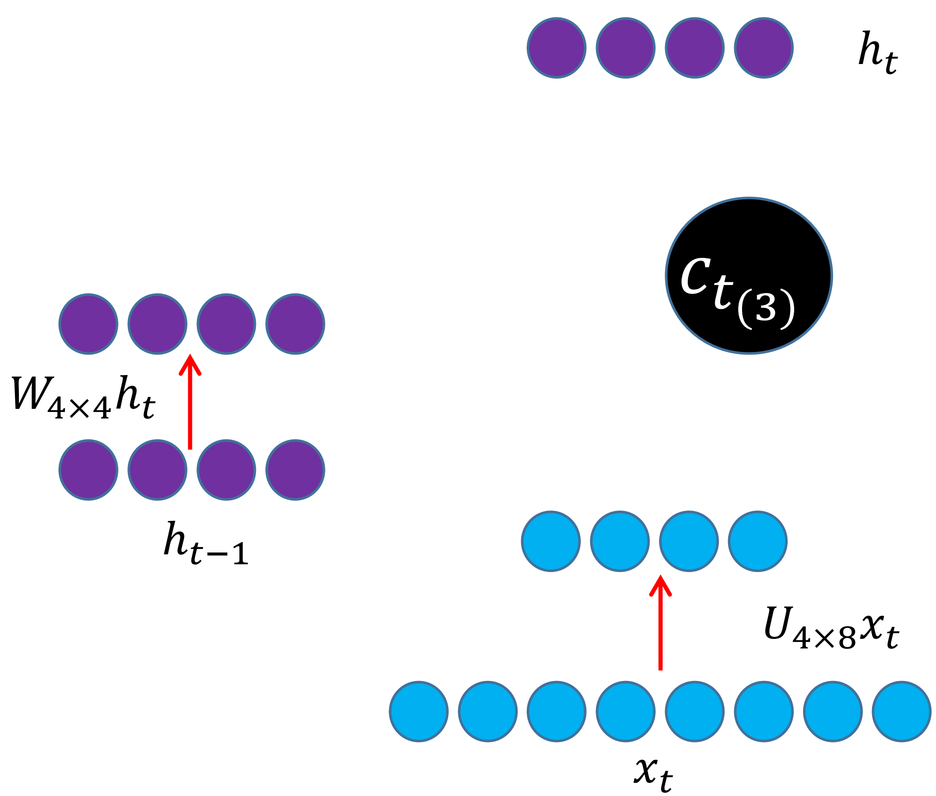
  - How much new info to store into the new c_t?

  - How much old info to discard from c_t?

  - How much info to use for prediction?

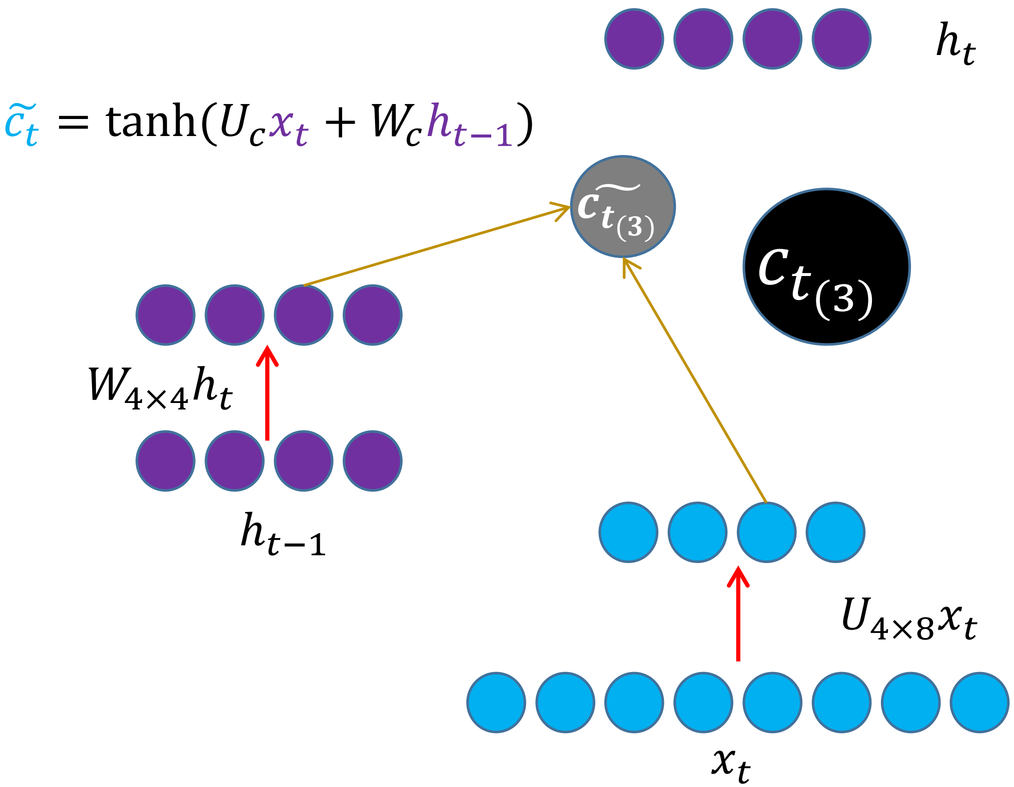
Long Short-Term Memory Unit

- How much new info to store into c_t?
  - What is the new info?



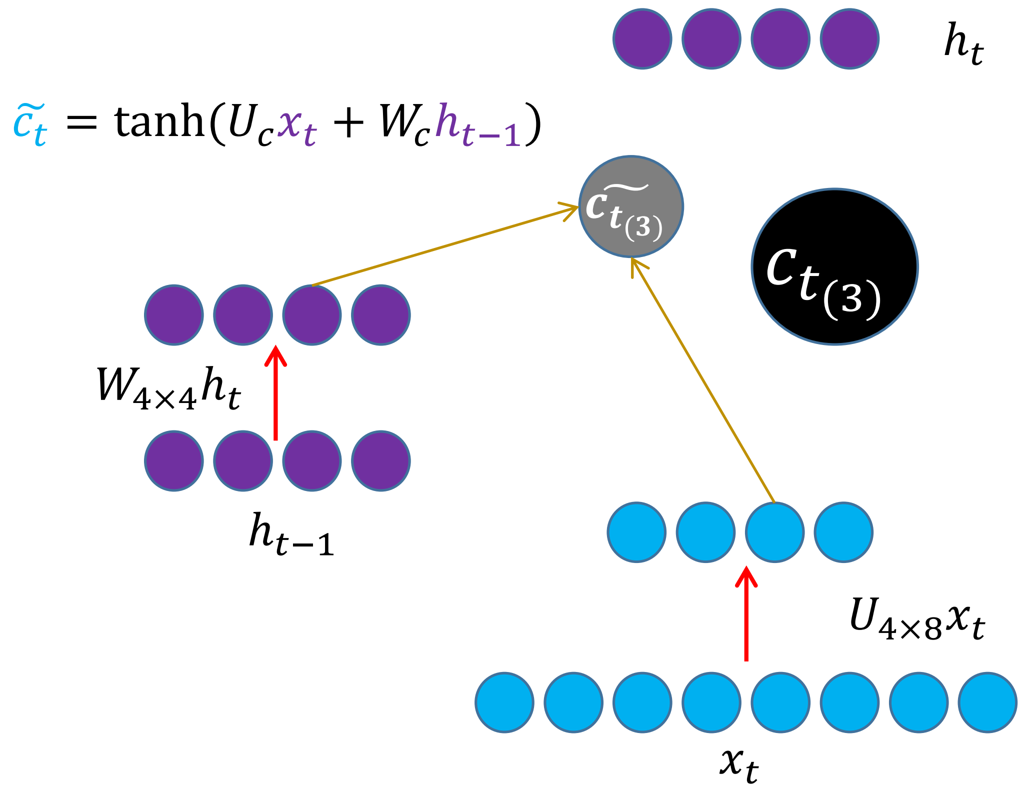
Long Short-Term Memory Unit

- How much new info to store into c_t?
  - What is the new info?



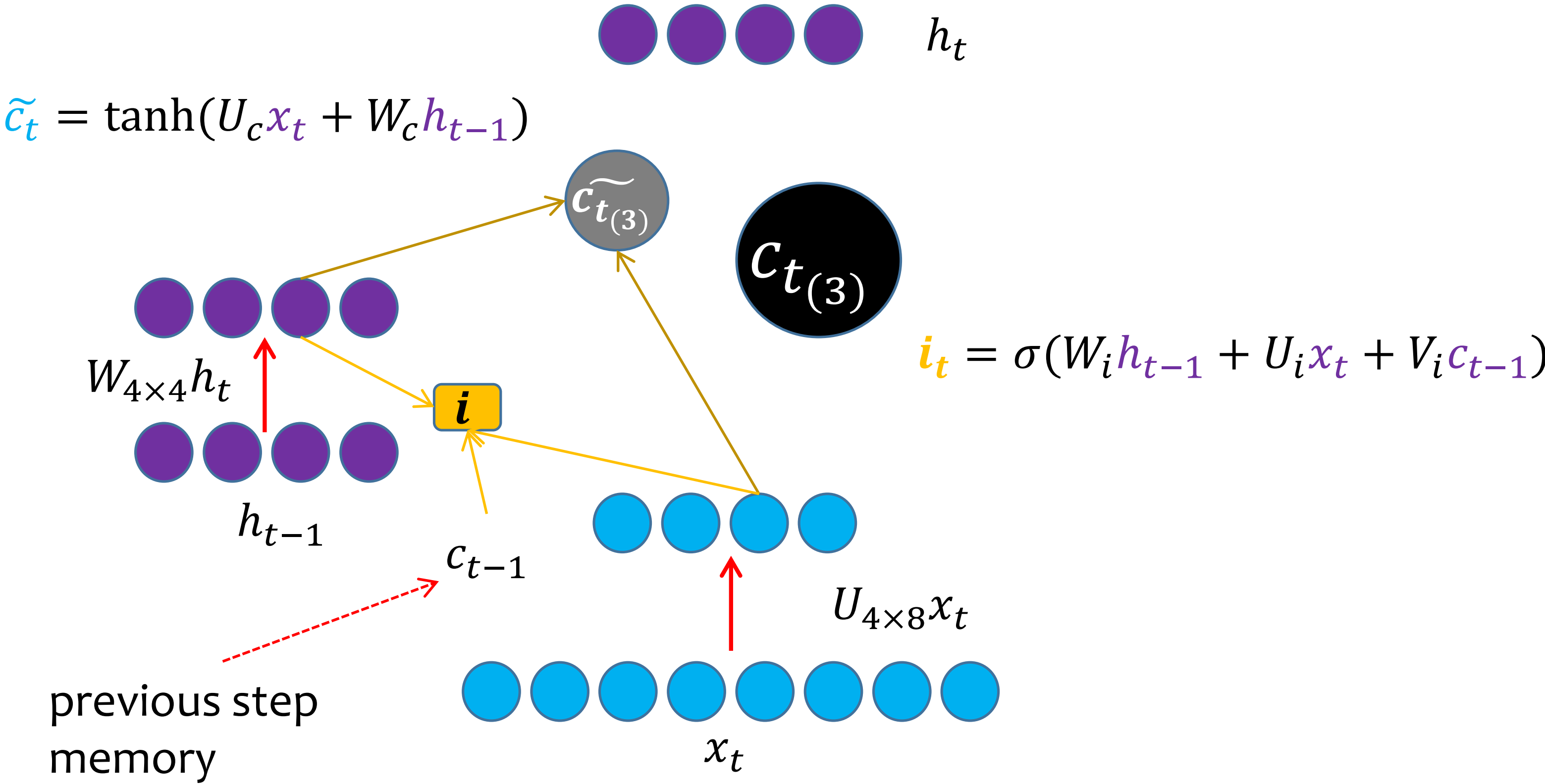
Long Short-Term Memory Unit

- How much new info to store into c_t?
  - What is the new info?
  - How much to store?



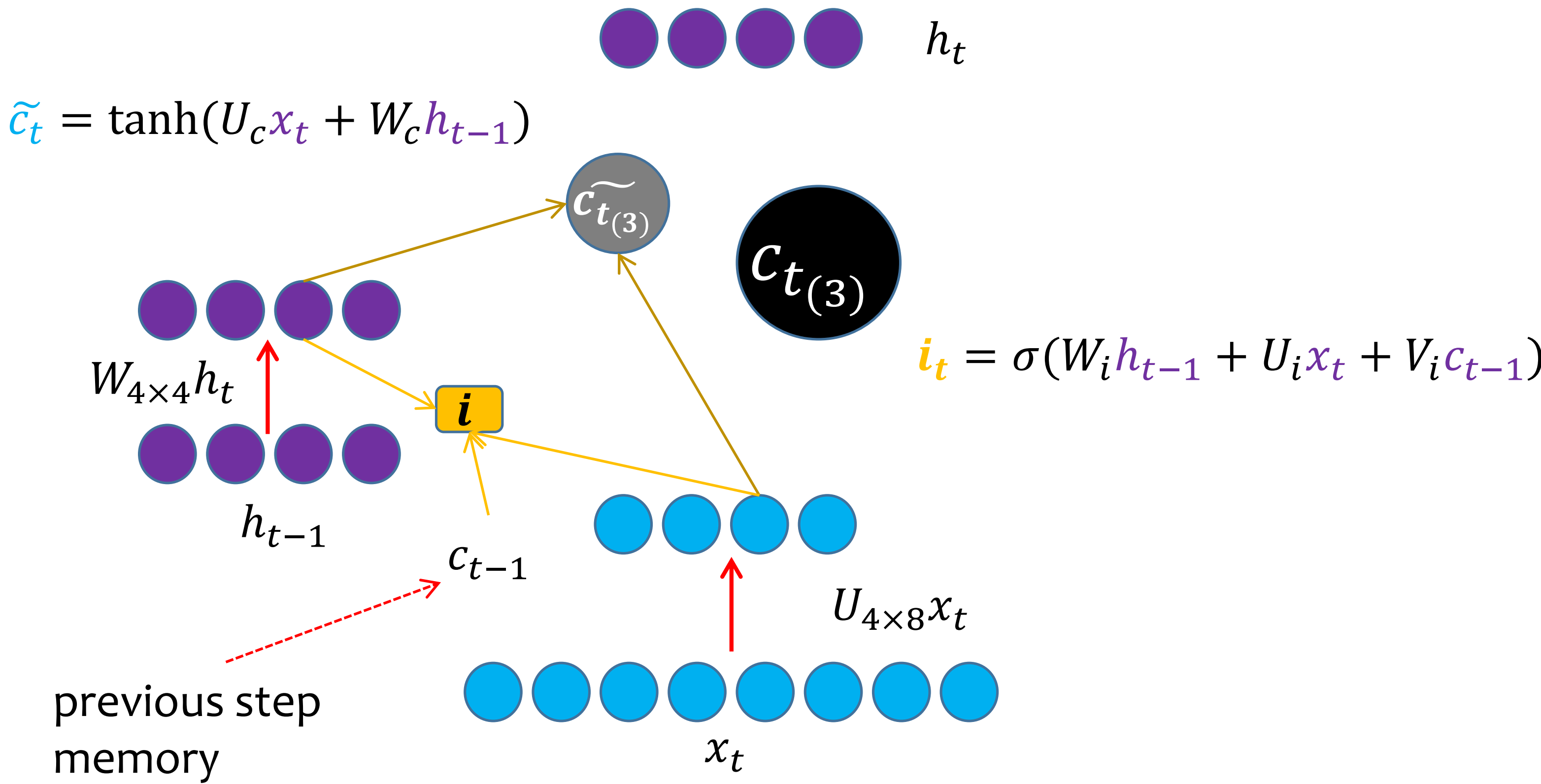
Long Short-Term Memory Unit

- How much new info to store into c_t?
  - What is the new info?
  - How much to store? (Input gate)



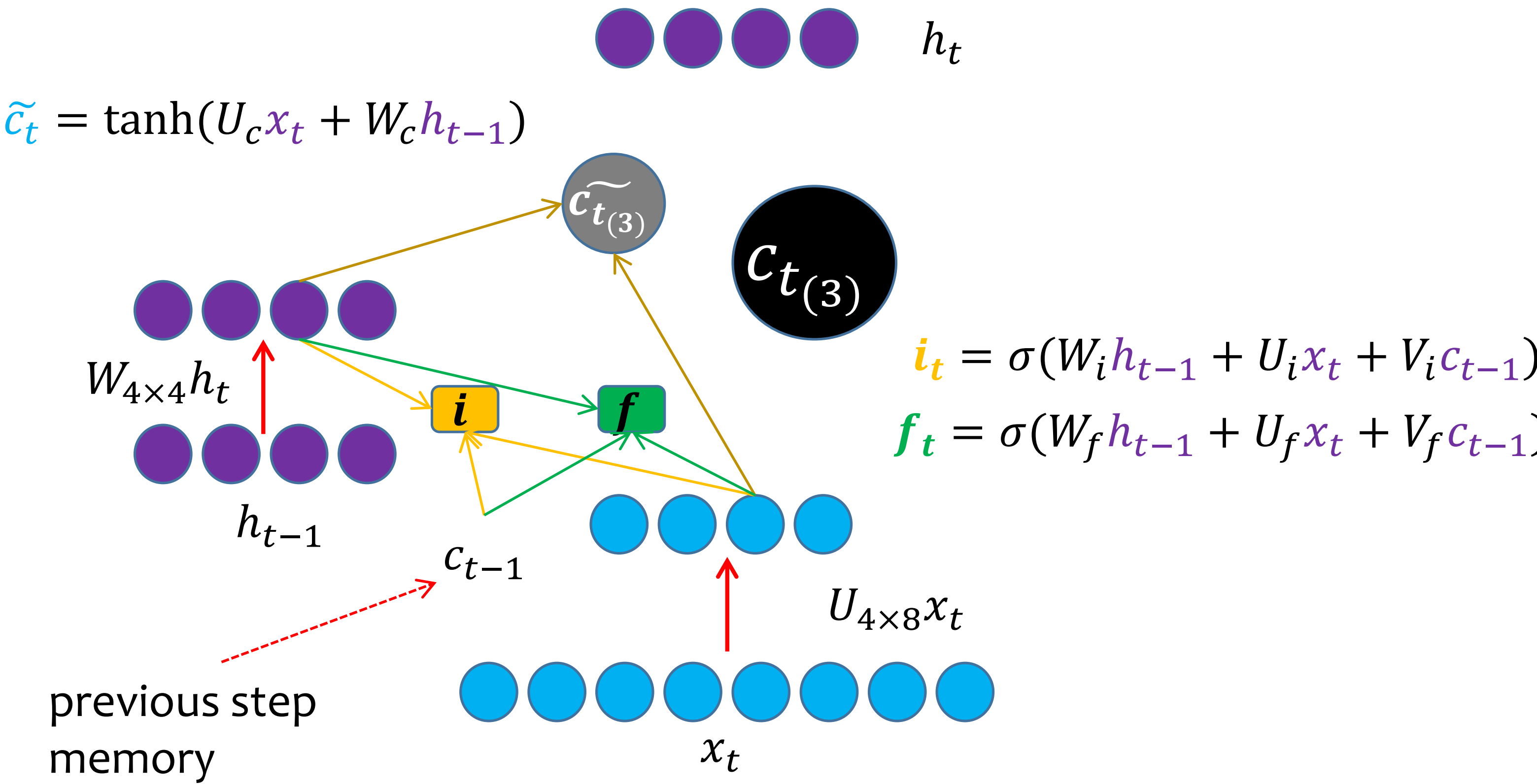
Long Short-Term Memory Unit

- How much old info to discard from c_t?



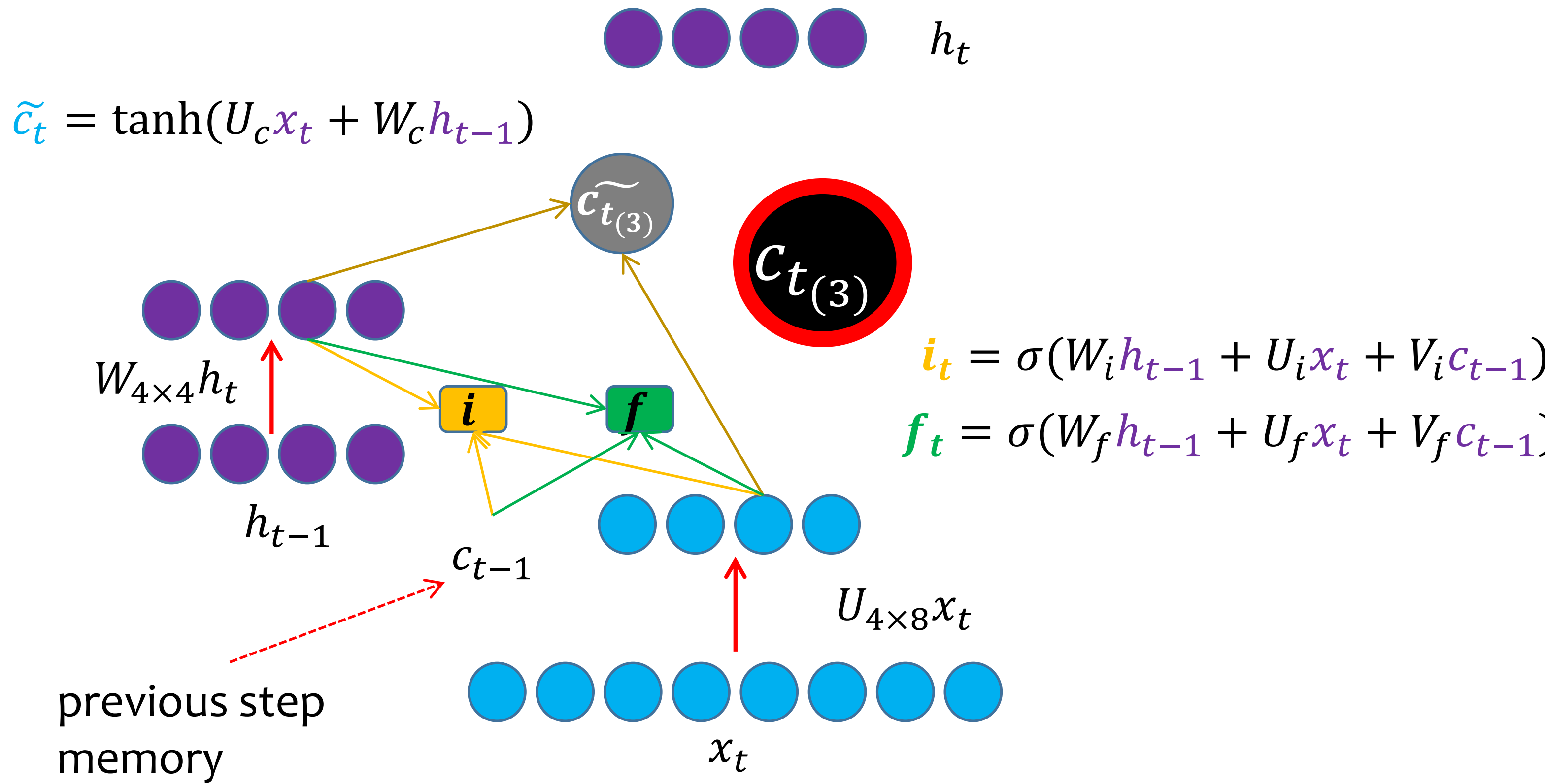
Long Short-Term Memory Unit

- How much old info to discard from c_t? (Forget gate)



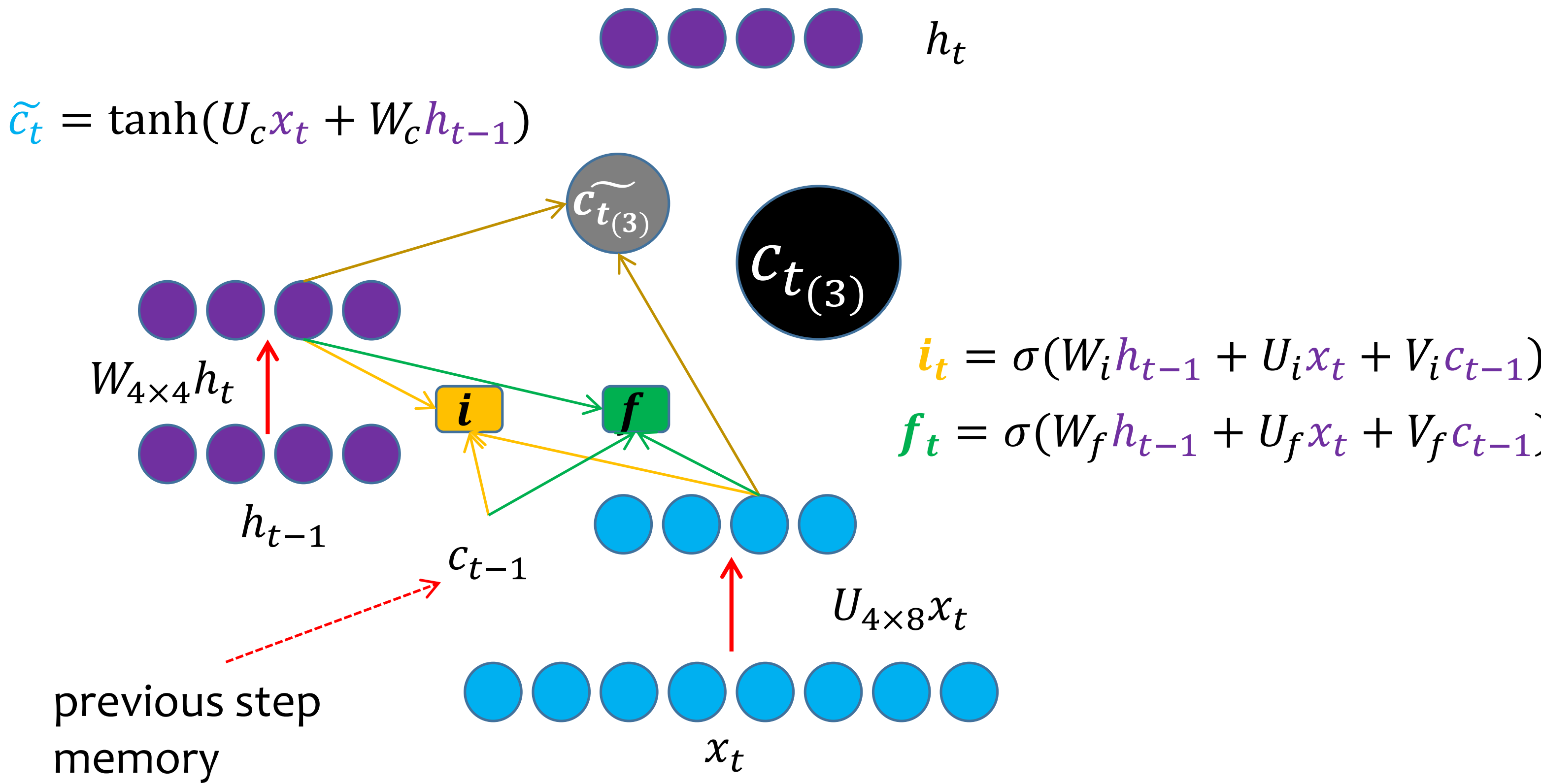
Long Short-Term Memory Unit

- Before answering the third question, we can compute new c_t
  - $c_t = i \odot \widetilde{c_t} + f \odot c_{t-1}$



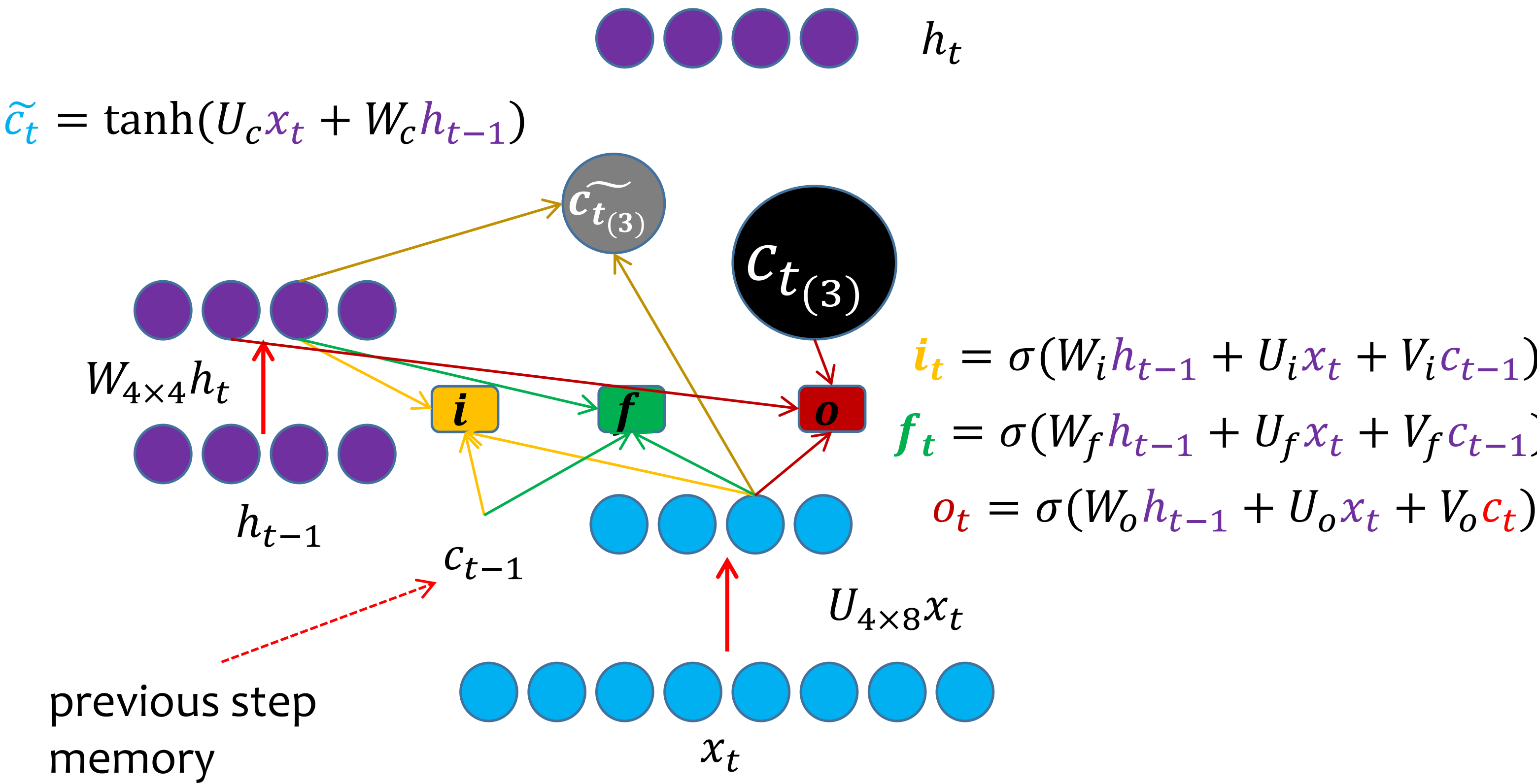
Long Short-Term Memory Unit

- Now, we should decide how much info to use in new c_t to make prediction.
  - Compute h_t?



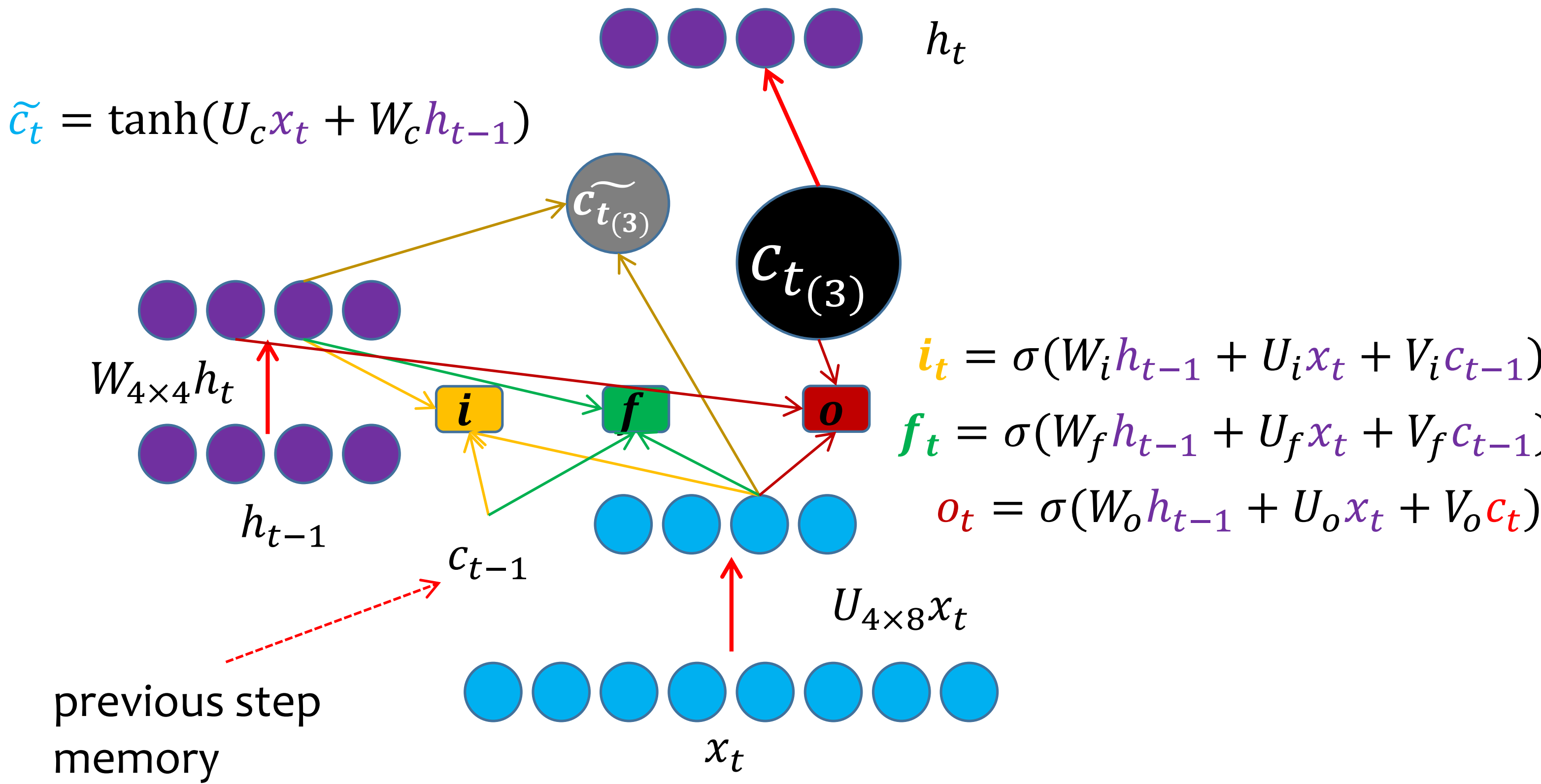
Long Short-Term Memory Unit

- Now, we should decide how much info to use in new c_t to make prediction.
  - Compute h_t? (Output gate)



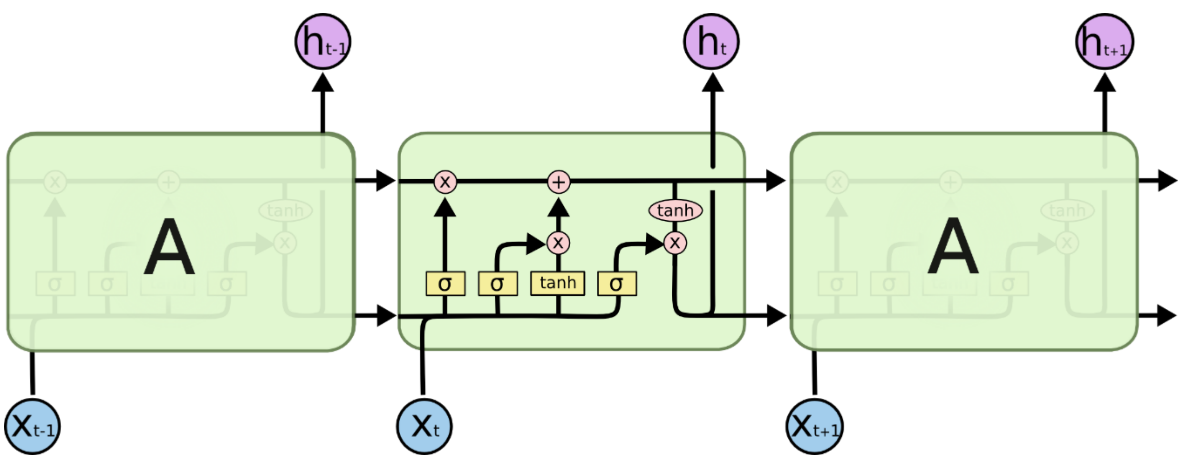
Long Short-Term Memory Unit

- Then we can compute h_t by:
    - $h_t = o_t \odot \tanh(c_t)$



Long Short-Term Memory Unit

- At each time step, we have previous h_{t-1}, c_{t-1} and current step x_t:
  - $i_t = \sigma(W_i h_{t-1} + U_i x_t + V_i c_{t-1})$ input ratio
  - $f_t = \sigma(W_f h_{t-1} + U_f x_t + V_f c_{t-1})$ forget ratio
  - $\widetilde{c}_t = \tanh(U_c x_t + W_c h_{t-1})$ input info to the memory cell
  - $c_t = f_t \odot c_{t-1} + i_t \odot \widetilde{c}_t$
  - $o_t = \sigma(W_o h_{t-1} + U_o x_t + V_o c_t)$ output ratio
  - $h_t = o_t \odot \tanh(c_t)$



Bonus: RNN v.s. HMM

- Difference

- Similarity

Bonus: RNN Visualization

Outline

- Sequence with Order
  - Unfolding Computational Graph
- Recurrent Neural Network
- Recursive Neural Network
- Challenge of Long-Term Dependencies
  - Long Short-term Memory Unit
  - Gated Recurrent Unit
- Explicit Memory
  - Memory Network (Weston et al)
  - Neural Turing Machine (Graves et al)

Rethink: Information Flow View