

# Natural Language Processing using RNNs and LSTMs

Dr. Amit Sethi  
IIT Bombay

# Objectives

- Explain why recurrence is necessary
- Explain BPTT and its problems
- Understand what gates do in LSTM
- Draw the internal diagram of an LSTM unit
- Write the forward and BPTT equations
- Design LSTM structures to solve problems

# Contents

- **Need for memory to process sequential data**
- Recurrent neural networks
- LSTM basics
- Some applications of LSTM in NLP
- Some advanced LSTM structures

# What is sequential data?

- One-dimensional discrete index
  - Example: time instances, character position
- Each data point can be a scalar, vector, or a symbol from an alphabet



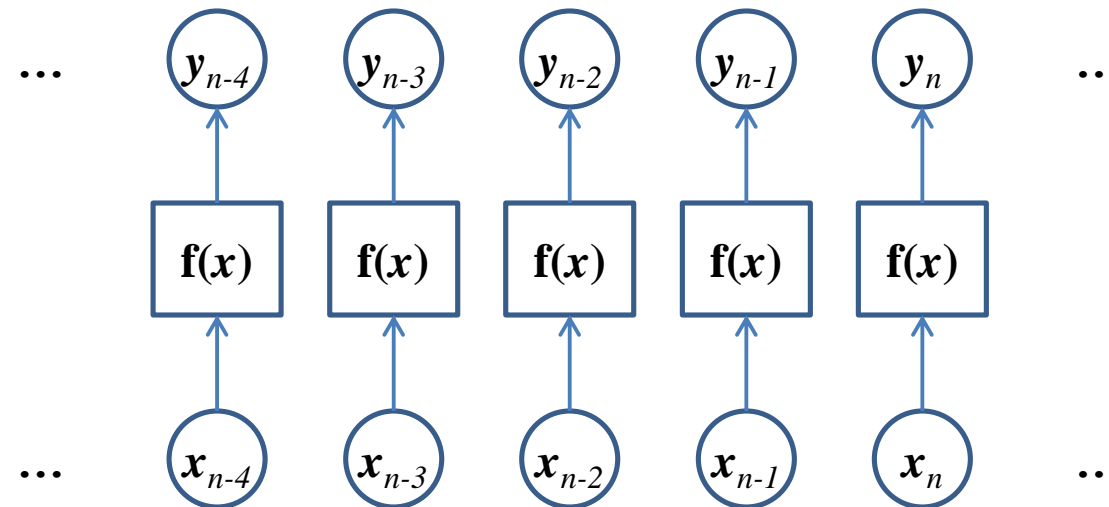
- Number of data points in a series can be ***variable***

# Examples of sequential data

- Speech
- Text (NLP)
- Music
- Protein and DNA sequences
- Stock prices and other time series

# Traditional ML is one-to-one

- POS tagging in NLP (input: words, output: POS tag)
- Stock trade: {Buy, NoAction, Sell}



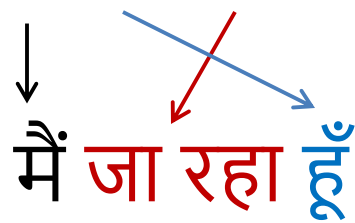
- What about taking past data into account?

# Need for past data or context

- Different POS
  - It is a quick **read**
  - I like to **read**

- Translation

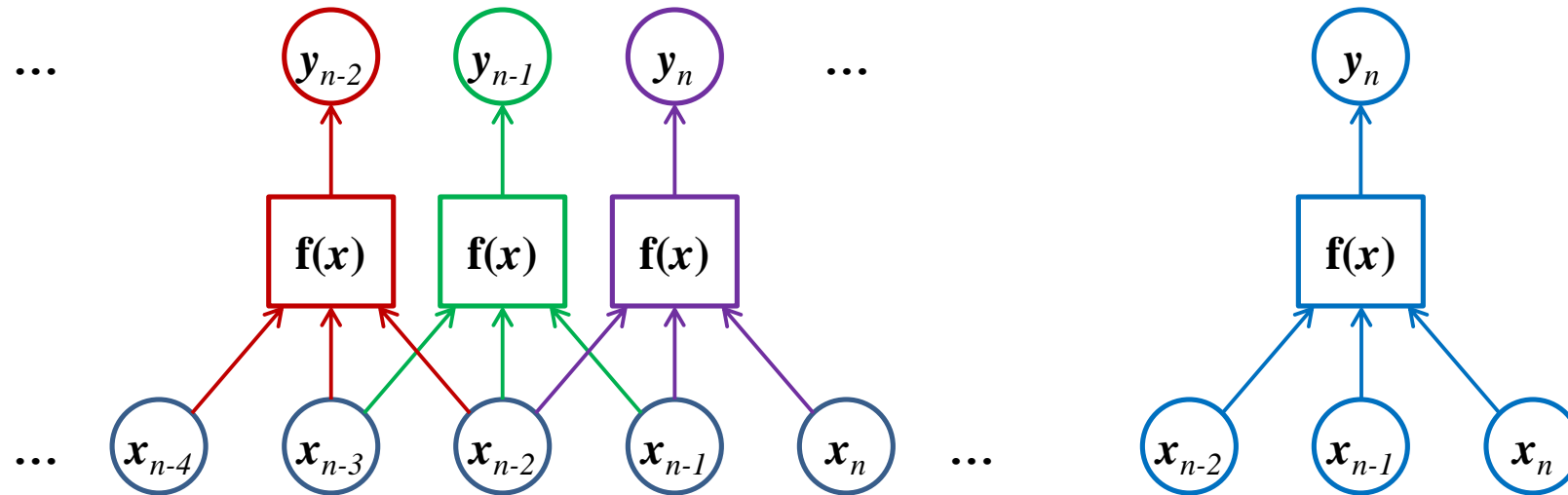
I am going


 मैं जा रहा हूँ (Re-ordered, ideal)

मैं हूँ जा रहा (Word by word, less than ideal)

# Using traditional ML for sequences

- Work with a fixed window

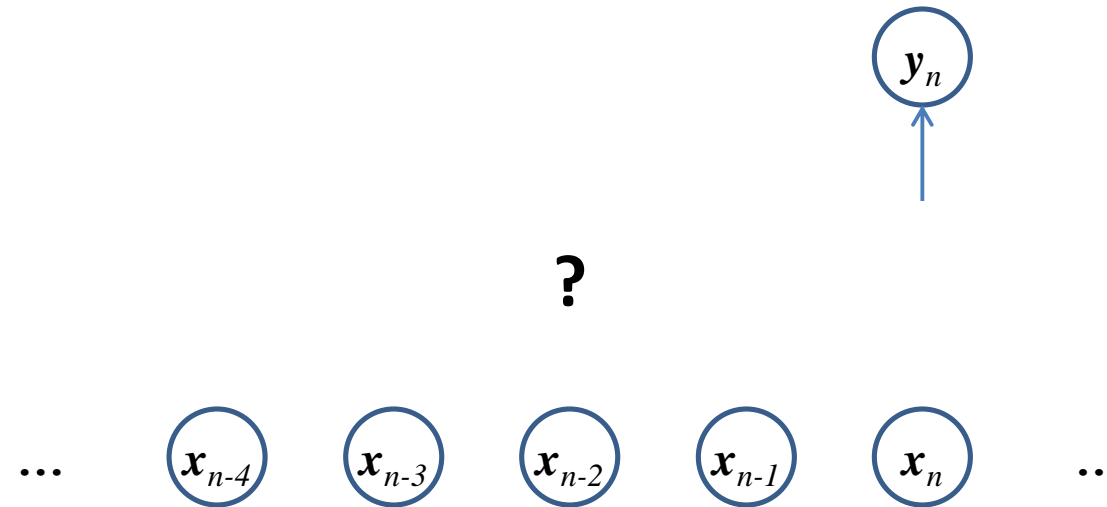


- What about influence of distant past?



# What if we want to do many to one

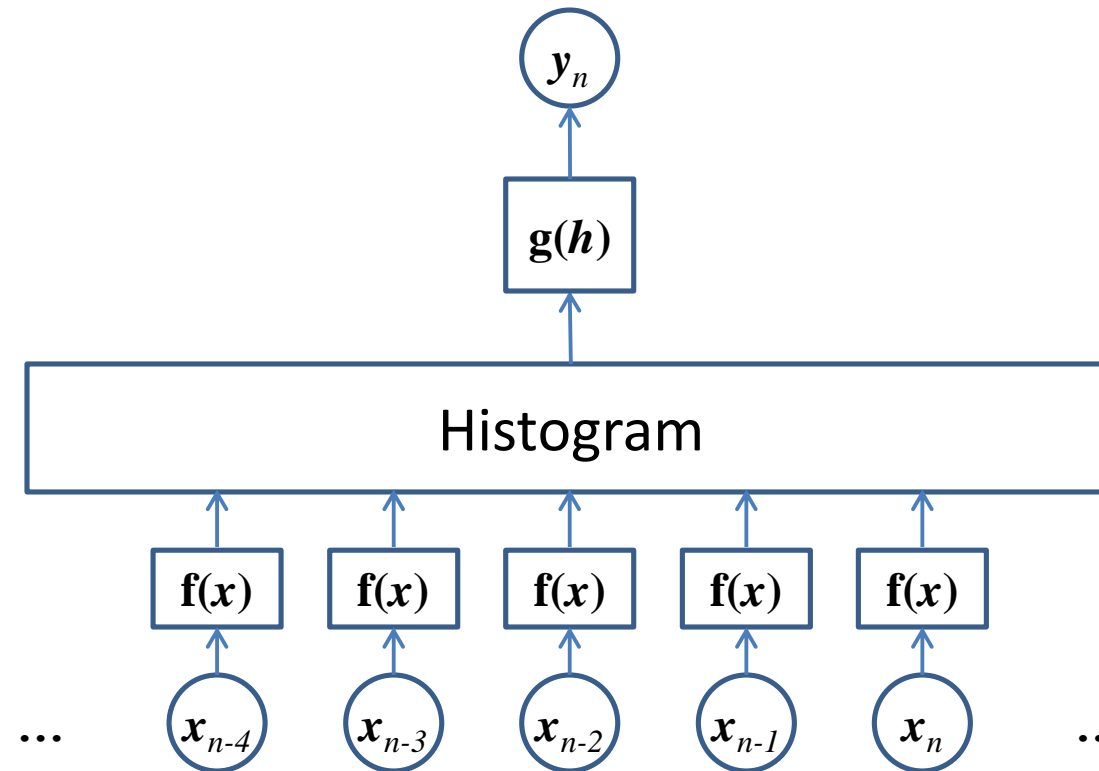
- Sentiment analysis in NLP



- What about taking past data into account?

# Using traditional ML for sequences

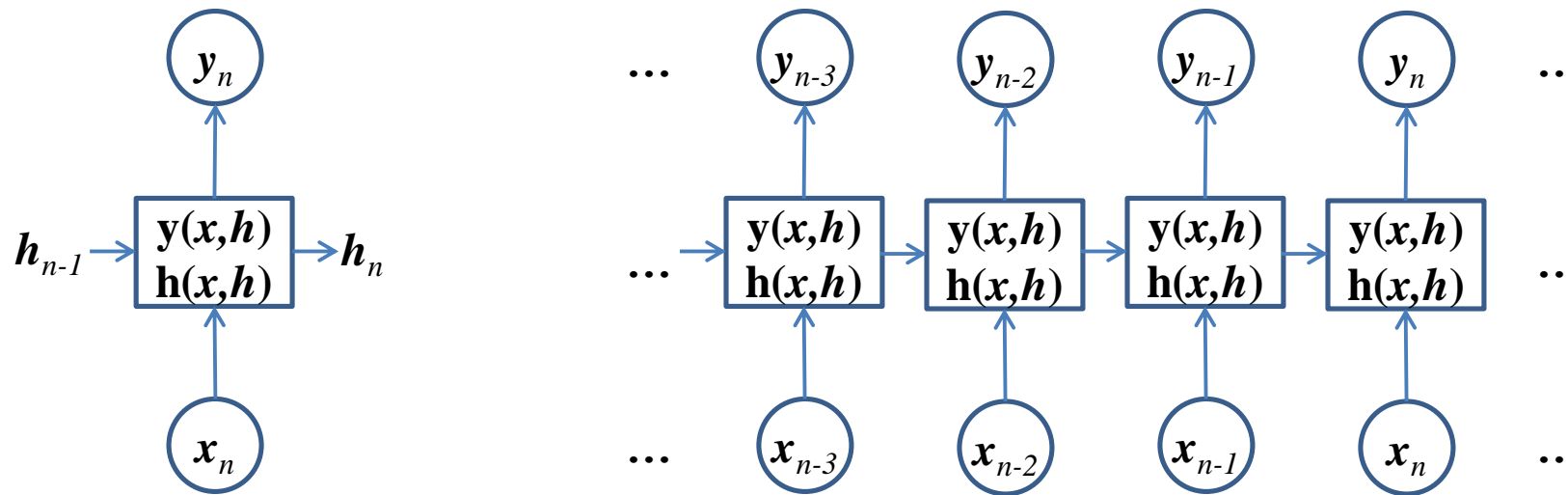
- Convert sequence into a feature vector



- What about using the order of the data?

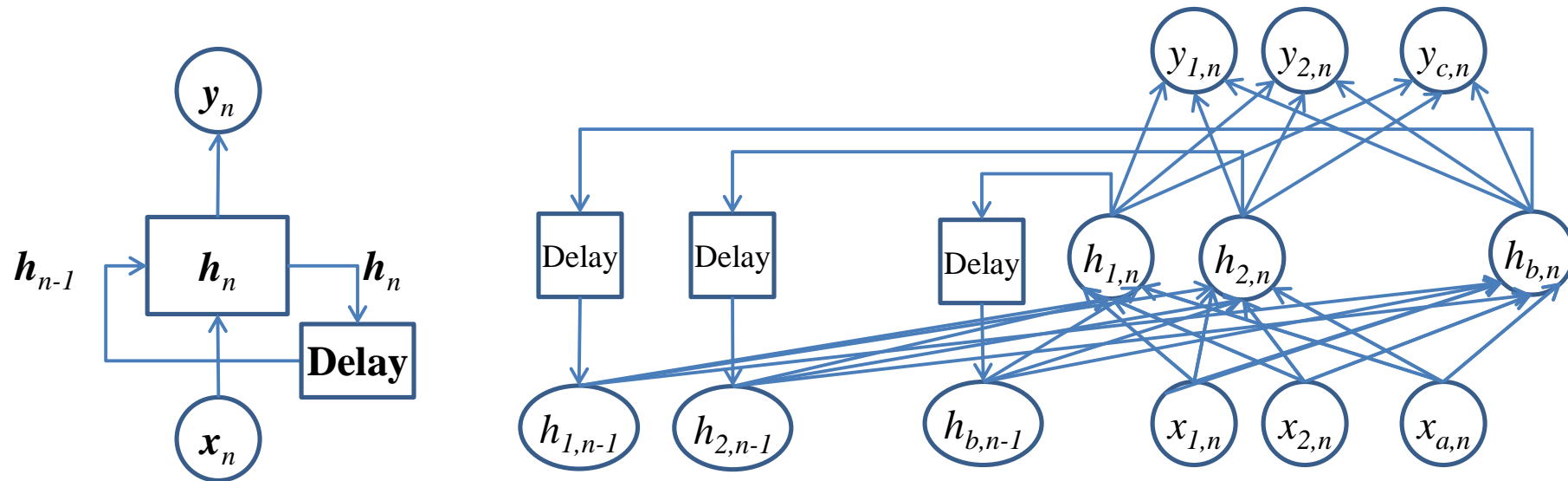
# Introducing memory (recurrence or state) in neural networks

- A memory state is computed in addition to an output, which is sent to the next time instance



# Another view of recurrence

- In the most basic form, memory state are simply the hidden neurons

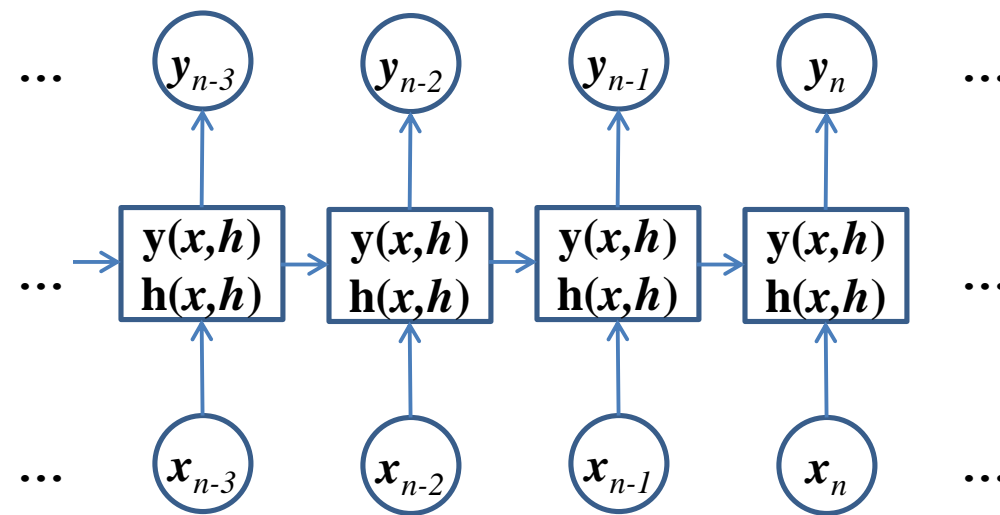


# Types of analysis possible on sequential data using “recurrence”

- One to one
- One to many
- Many to one
- Many to many

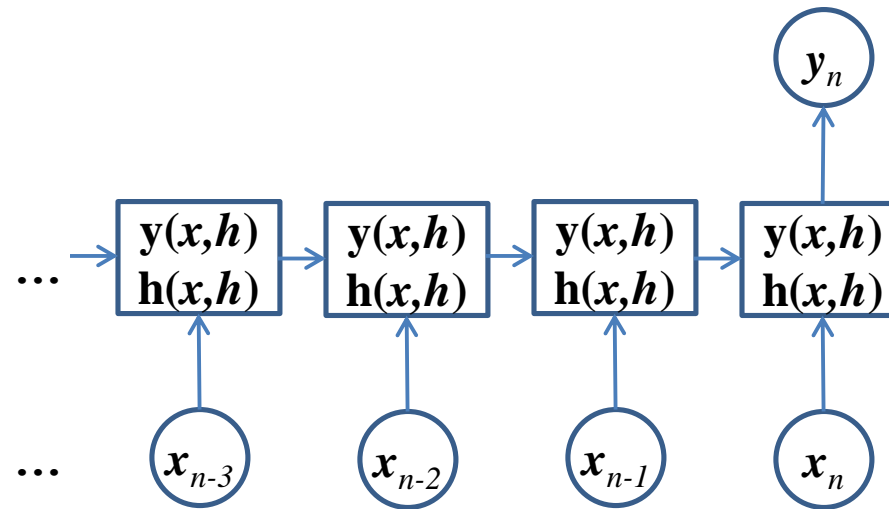
# Examples: One to one

- POS tagging in NLP
- Stock trade: {Buy, NoAction, Sell}



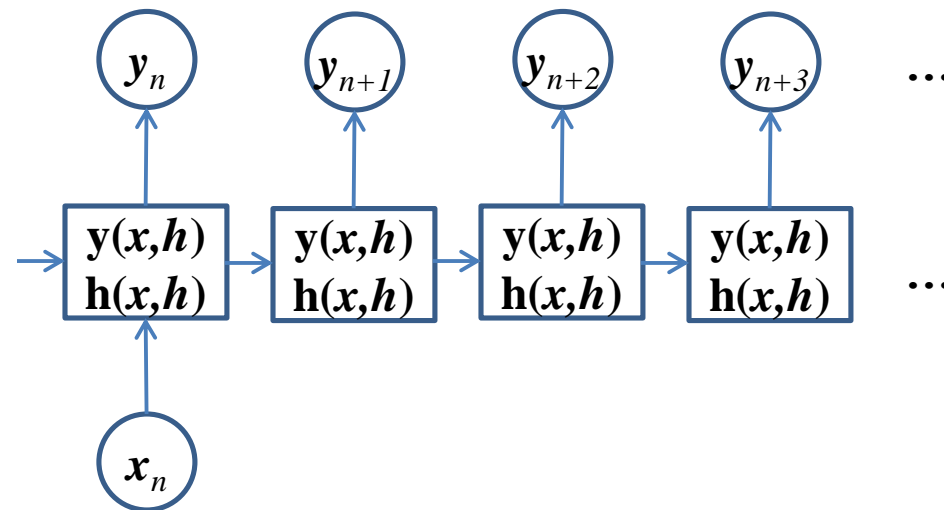
# Examples: Many to one

- Sentiment analysis in NLP



# Examples: One to many

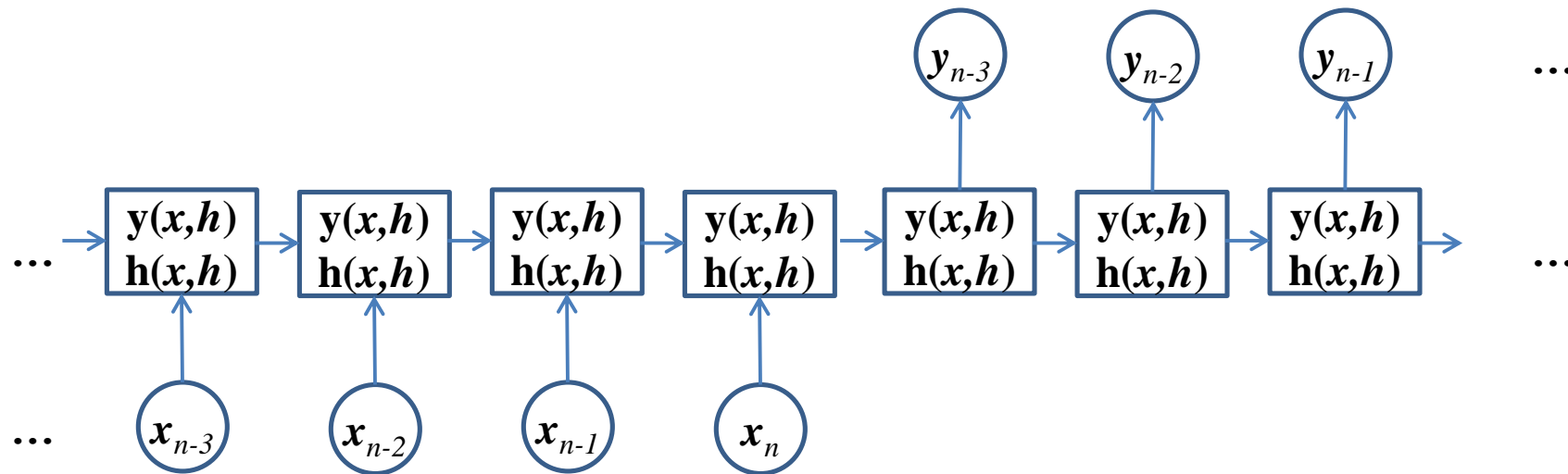
- Generate caption based on an image
- Generate text given topic





# Examples: Many to many

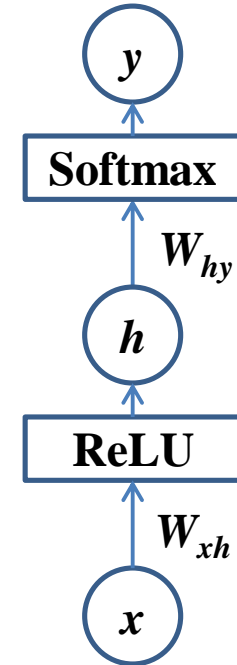
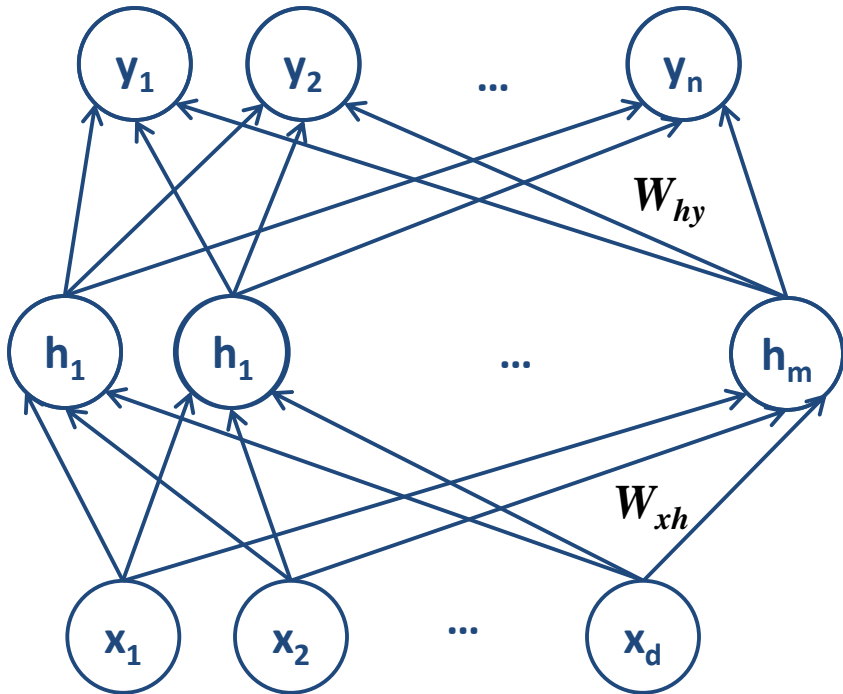
- Language translation



# Contents

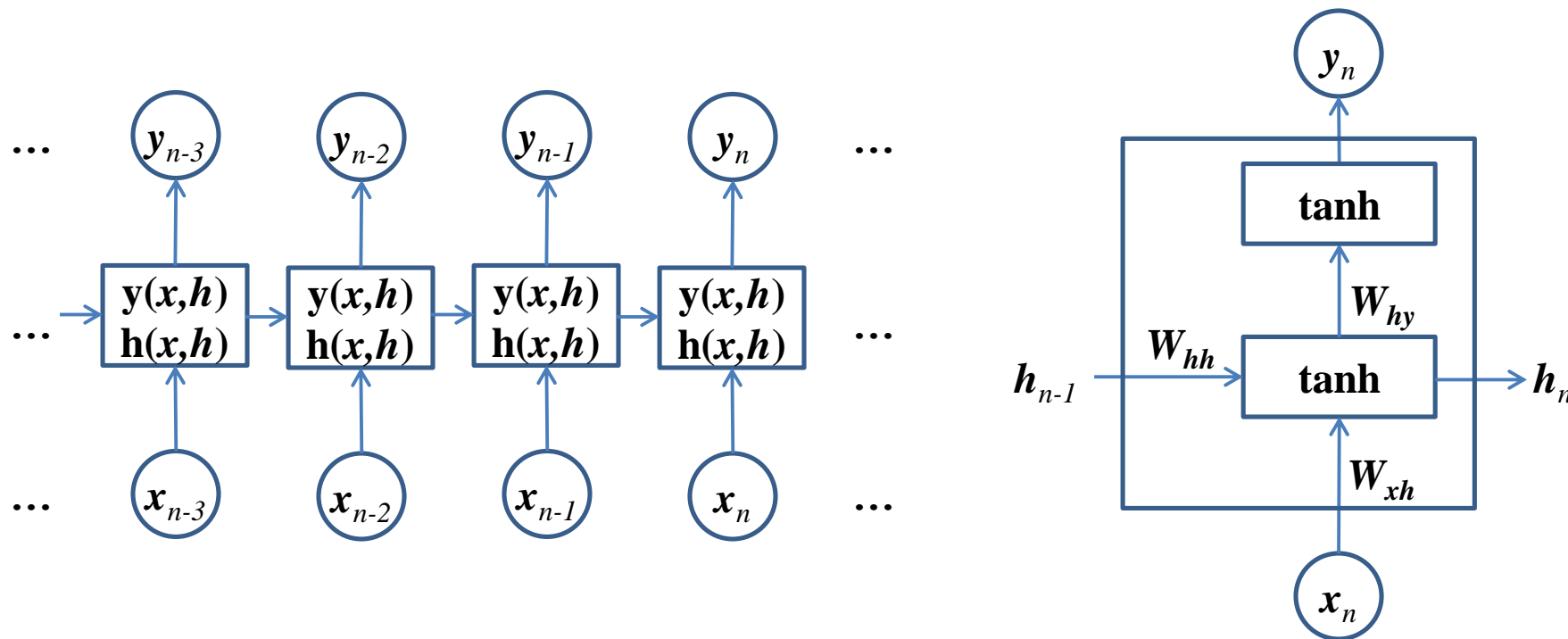
- Need for memory to process sequential data
- **Recurrent neural networks**
- LSTM basics
- Some applications of LSTM in NLP
- Some advanced LSTM structures

# Revising feedforward neural networks



# Recurrent neural networks

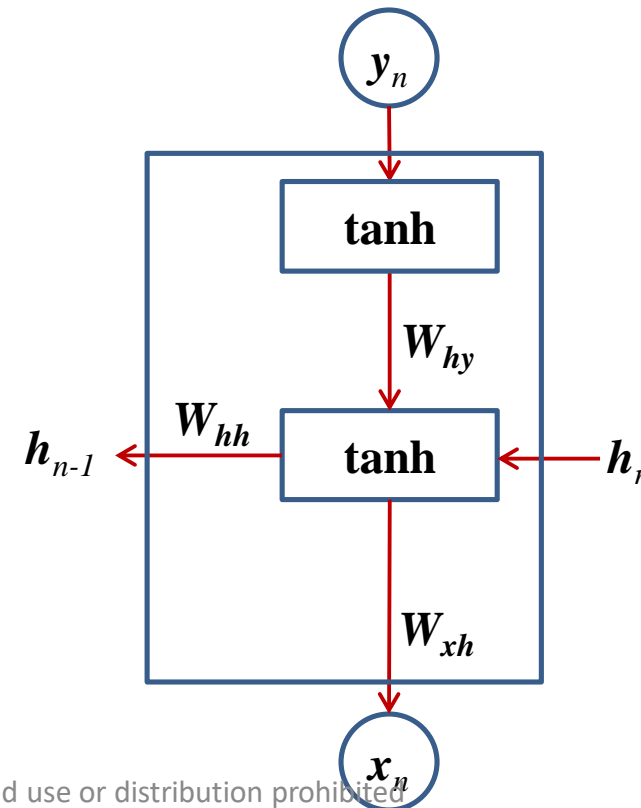
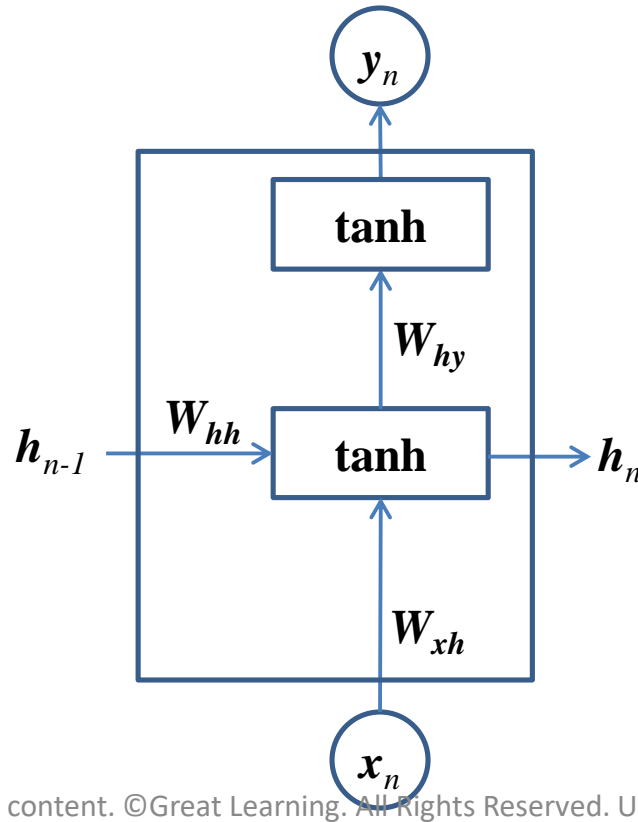
- Vanilla RNNs used the hidden layer activation as a state



# Backpropagation through time (BPTT)

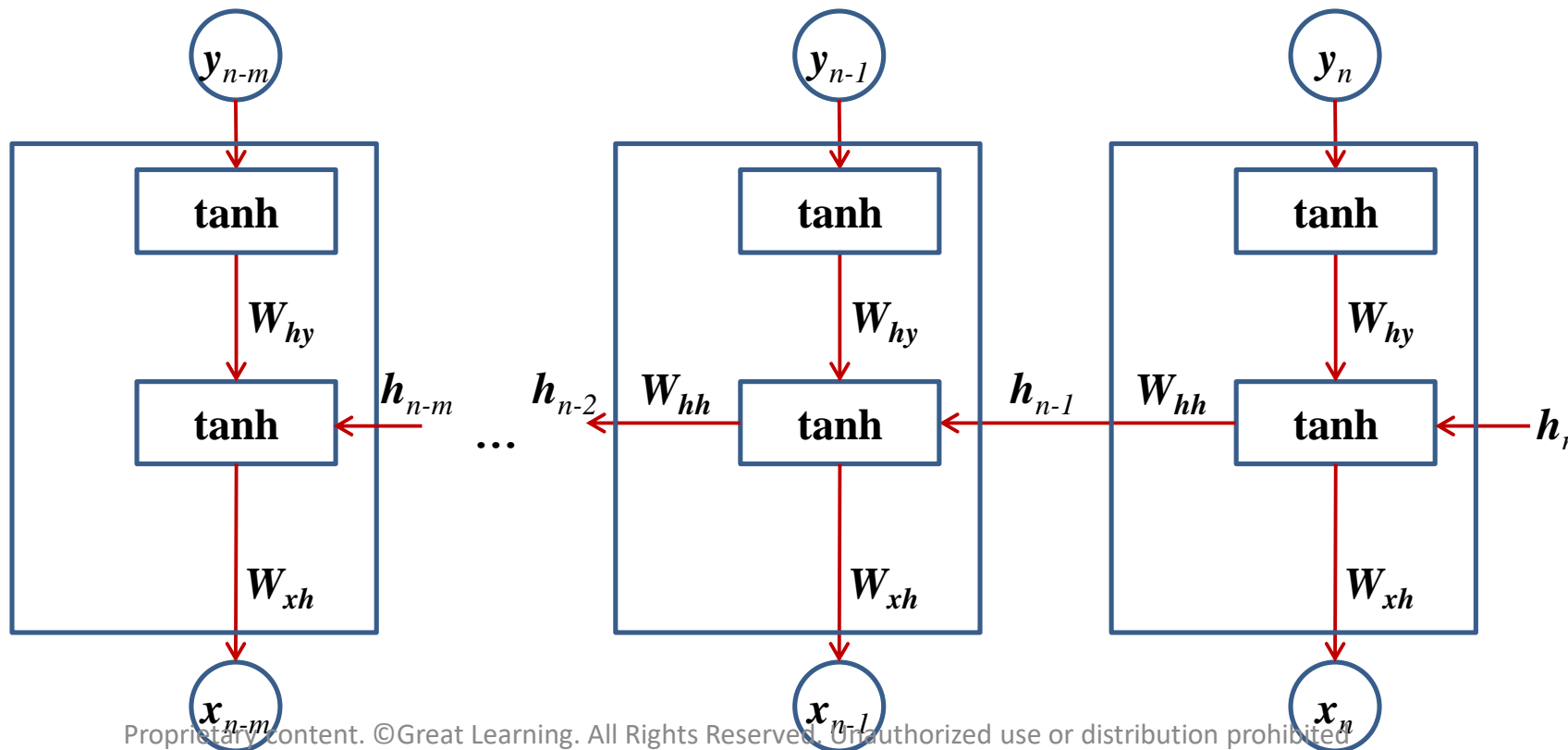
- Just like how forward propagation uses previous state
- Backpropagation uses derivative from future output

...



# Use of a window length

- We need to put a limit on how long will the gradient travel back in time



# Mathematical expression for BPTT

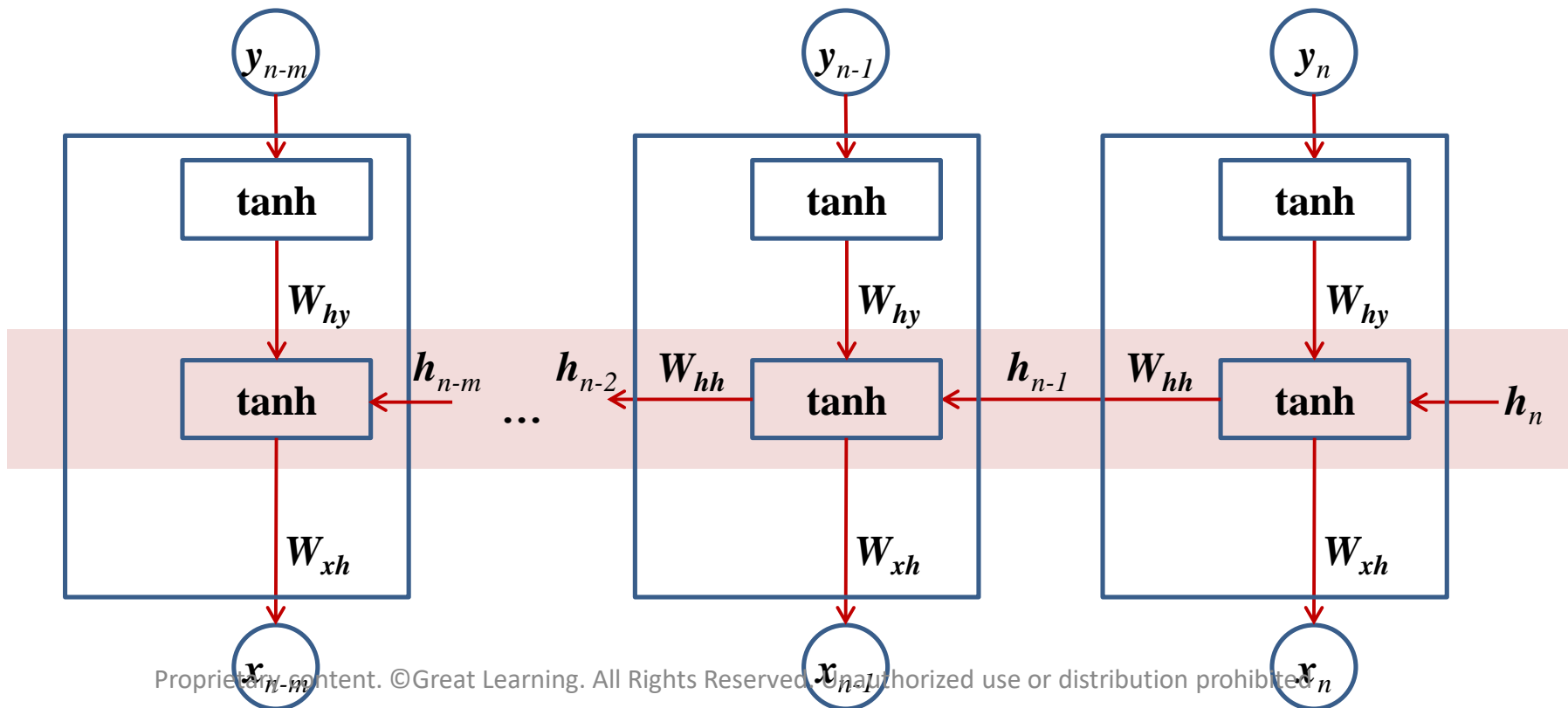
- Forward:  $\mathbf{y}_n = \mathbf{g}(\mathbf{W}_{hy}\mathbf{h}_n)$   
 $= \mathbf{g}(\mathbf{W}_{hy}\mathbf{f}(\mathbf{W}_{hh}\mathbf{h}_{n-1} + \mathbf{W}_{xh}\mathbf{x}_n))$

- Backward example:

$$\frac{\partial \mathbf{y}}{\partial \mathbf{W}_{hh}} = \mathbf{g}'\mathbf{W}_{hy}\mathbf{h}_n' = \mathbf{g}'\mathbf{W}_{hy}\mathbf{f}'(\mathbf{h}_{n-1} + \mathbf{W}_{hh}\mathbf{h}_{n-1}')$$

# Vanishing and exploding gradient

- Gradient gets repeatedly multiplied by  $\mathbf{W}_{hh}$
- This can lead to vanishing or exploding gradient depending on the norm of  $\mathbf{W}_{hh}$

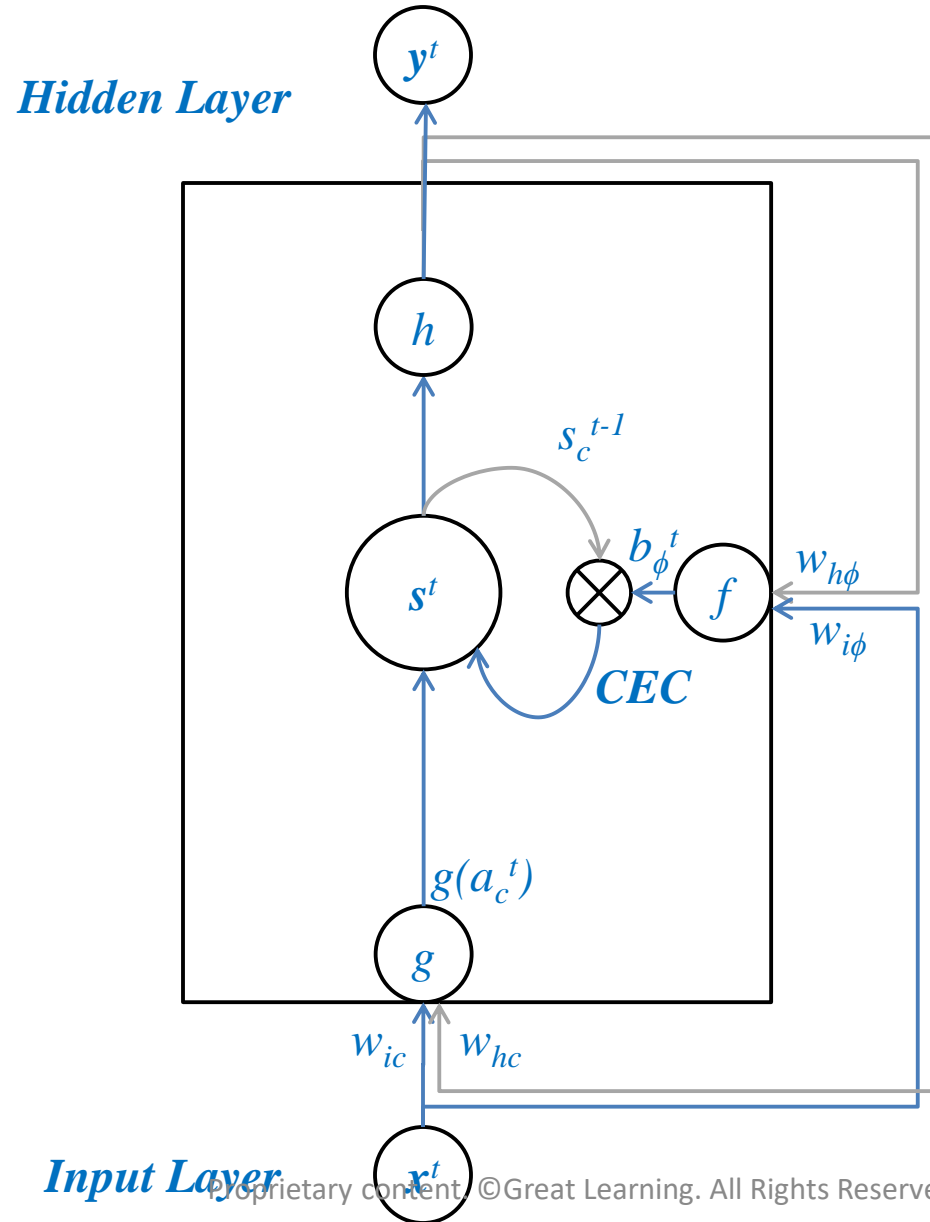




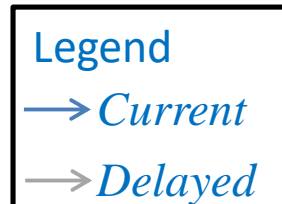
# Contents

- Need for memory to process sequential data
- Recurrent neural networks
- **LSTM basics**
- Some applications of LSTM in NLP
- Some advanced LSTM structures

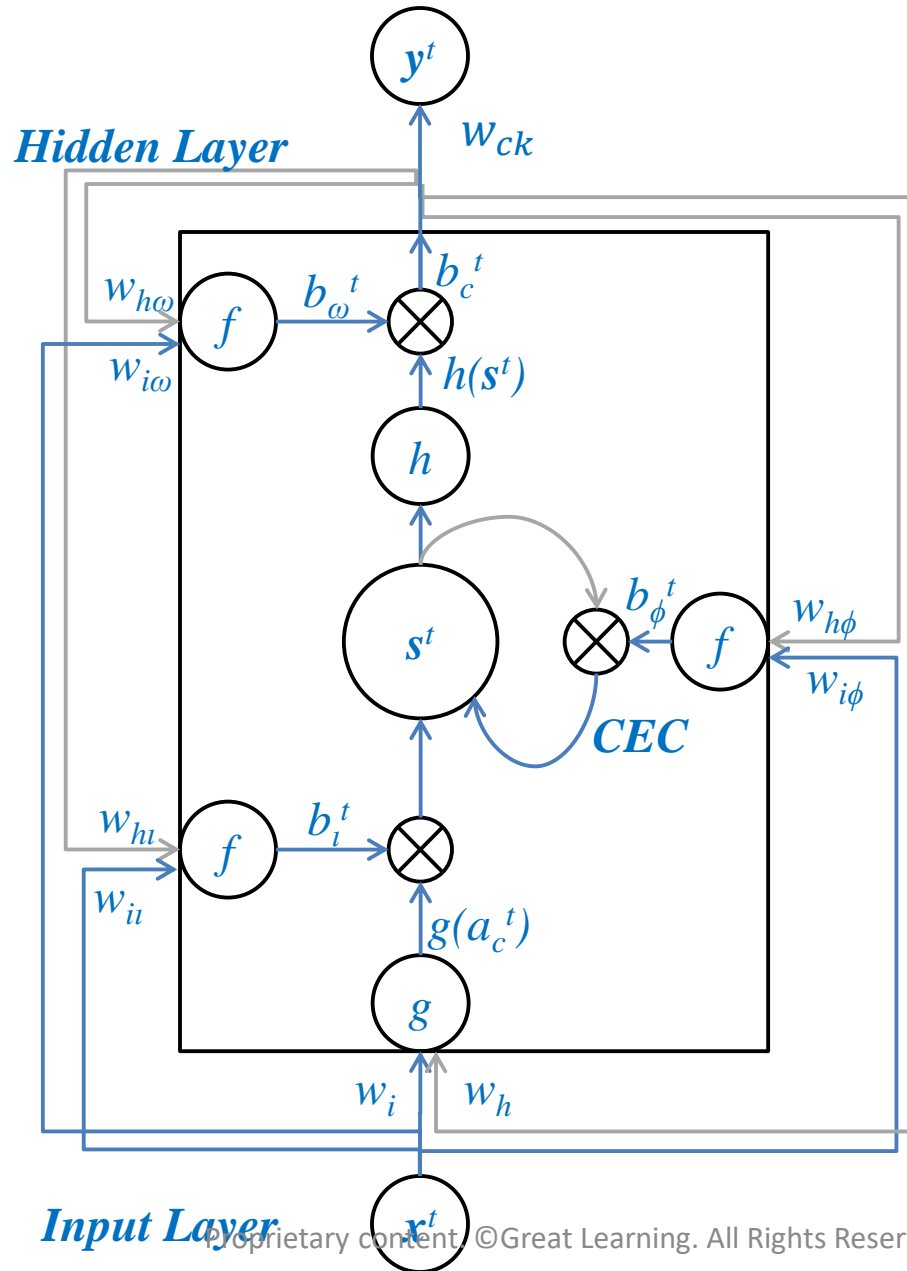
# Introducing a forget gate to control the gradient



- The state doesn't multiply with a constant weight
- A gate function  $f$  (usually a sigmoid) represents *on* or *off*
- State is *forgotten* and replaced by the input  $g$  if  $f = 0$
- But, what if  $f = 1$ ? How do we control the influence of input?



# Adding input and output gates



- On similar lines as the forget gate, an input gate decides whether the input will override the state or not
- Similarly, an output gate will decide whether the output will be passed out or not
- The input to all these gates are NN inputs, and NN hidden-layer output

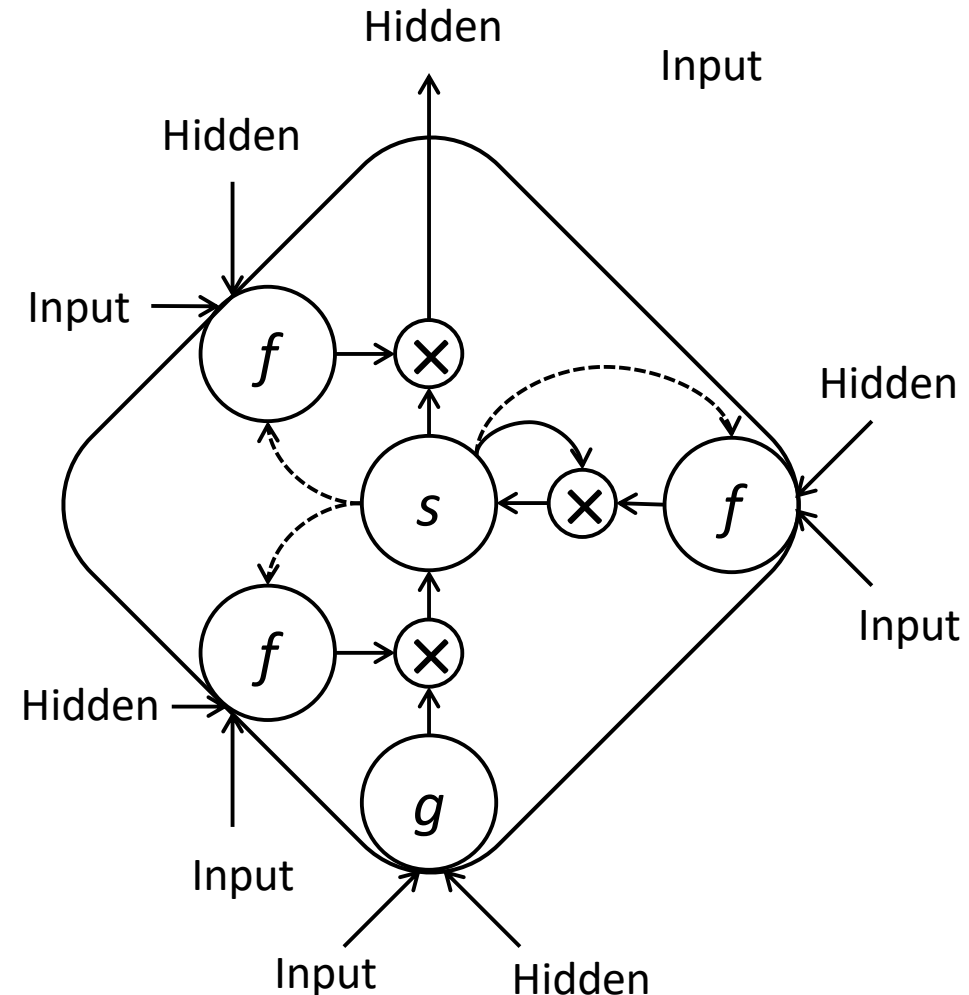
Legend

→ Current

→ Delayed

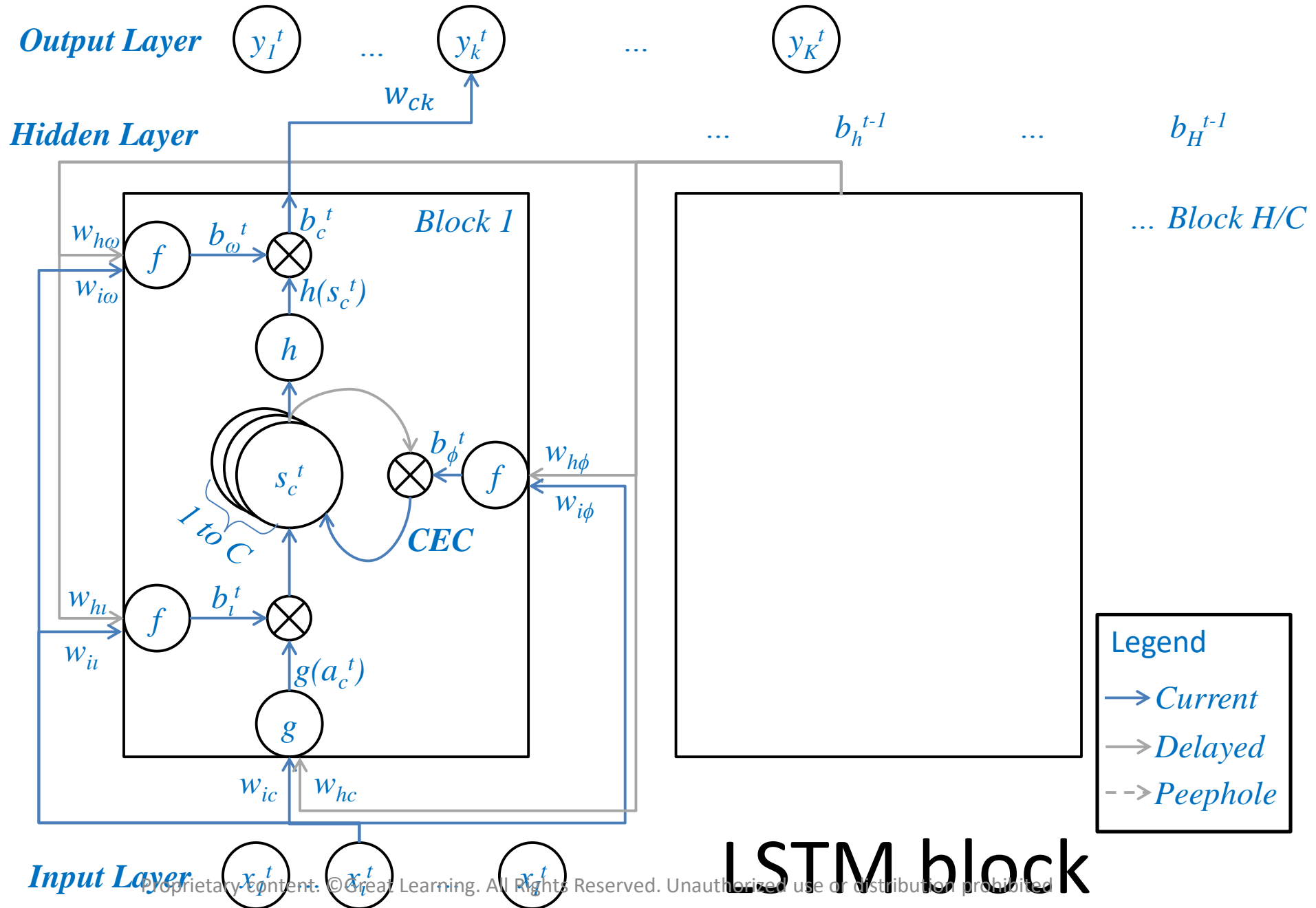
# Another view

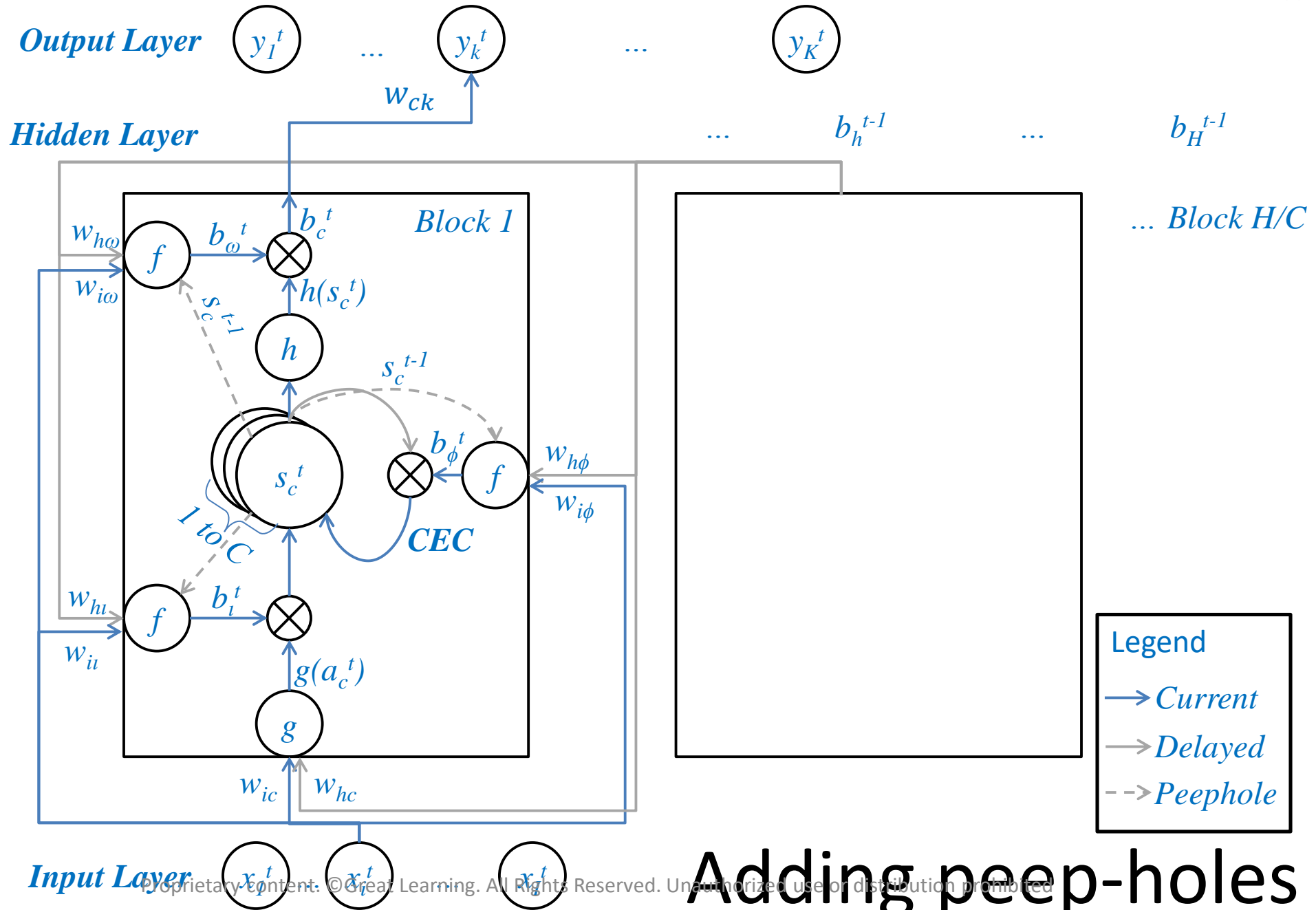
- CEC is constant error carrousel
  - No vanishing gradients
  - But, it is not always on
- Introducing gates:
  - Allow or disallow input
  - Allow or disallow output
  - Remember or forget state



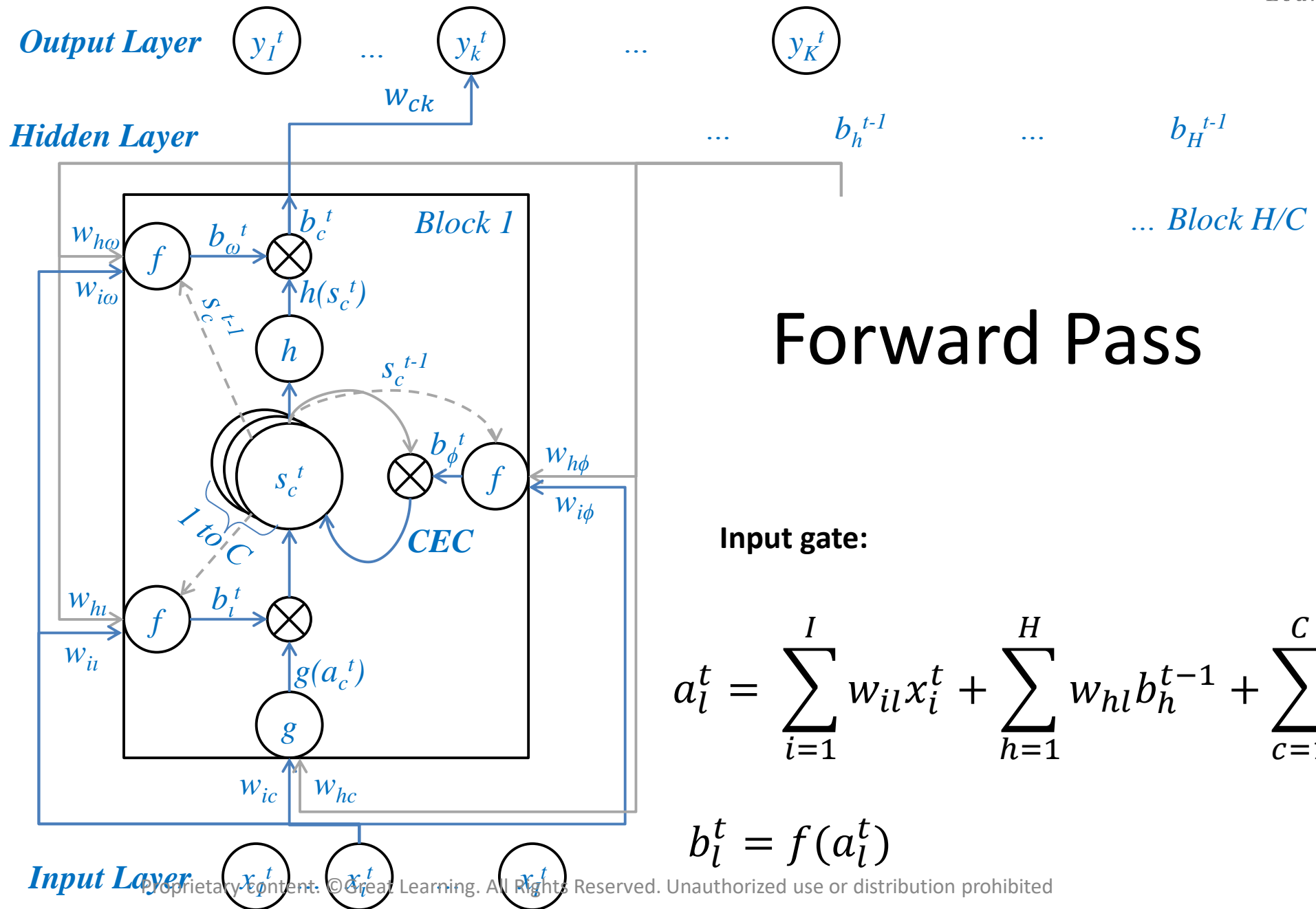
# A few words about the LSTM

- **CEC:** With the forget gate, influence of the state forward can be modulated such that it can be remembered for a long time, until the state or the input changes to make LSTM forget it. This ability or the path to pass the past-state unaltered to the future-state (and the gradient backward) is called constant error carousel (CEC). It gives LSTM the ability to remember long term (hence, long short term memory)
- **Blocks:** Since there are just too many weights to be learnt for a single state bit, several state bits can be combined into a single block such that the state bits in a block share gates
- **Peepholes:** The state itself can be an input for the gate using *peephole* connections
- **GRU:** In a variant of LSTM called gated recurrent unit (GRU), input gate can simply be one-minus-forget-gate. That is, if the state is being *forgotten*, then replace it by input, and if it is being *remembered*, then block the input

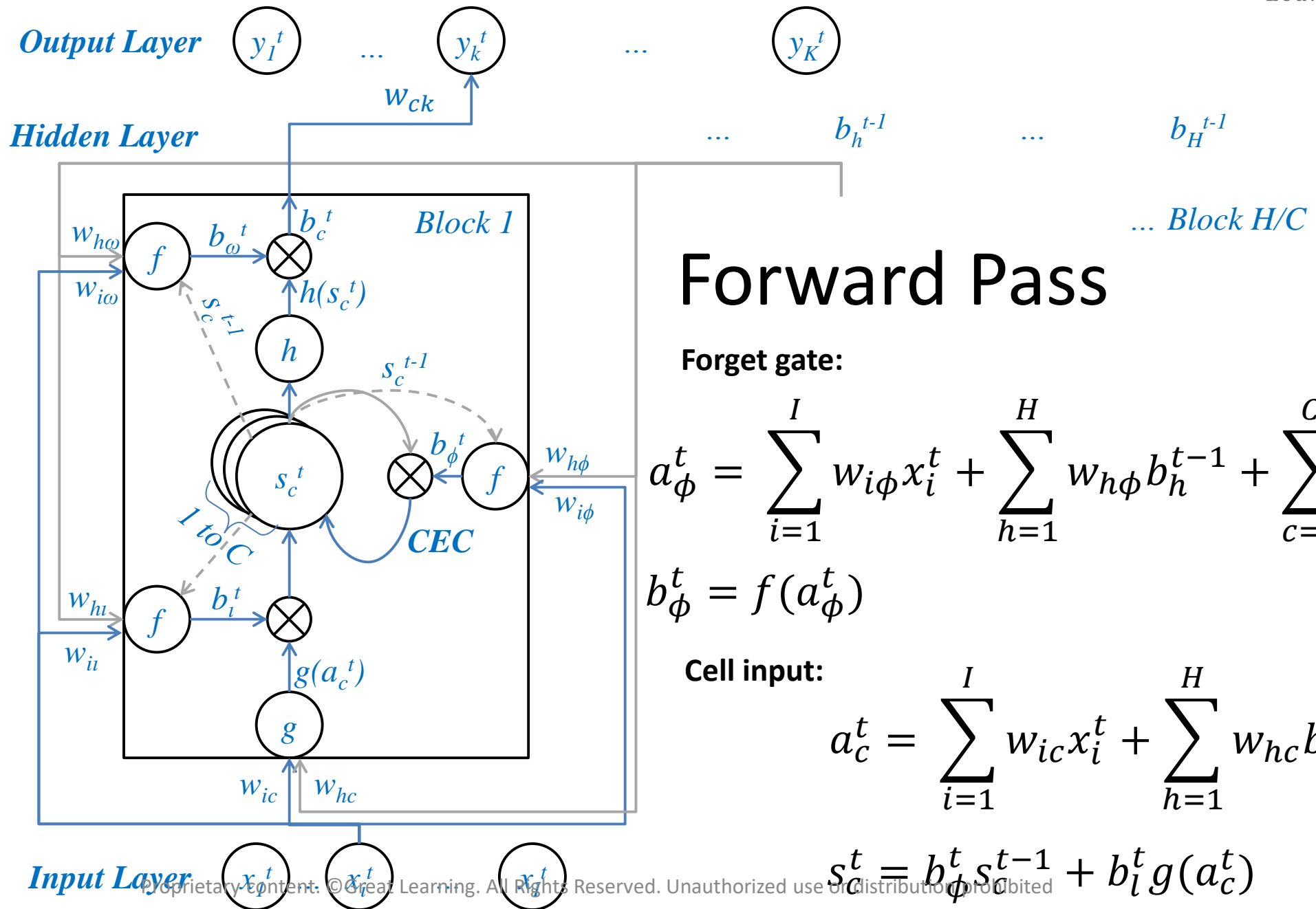


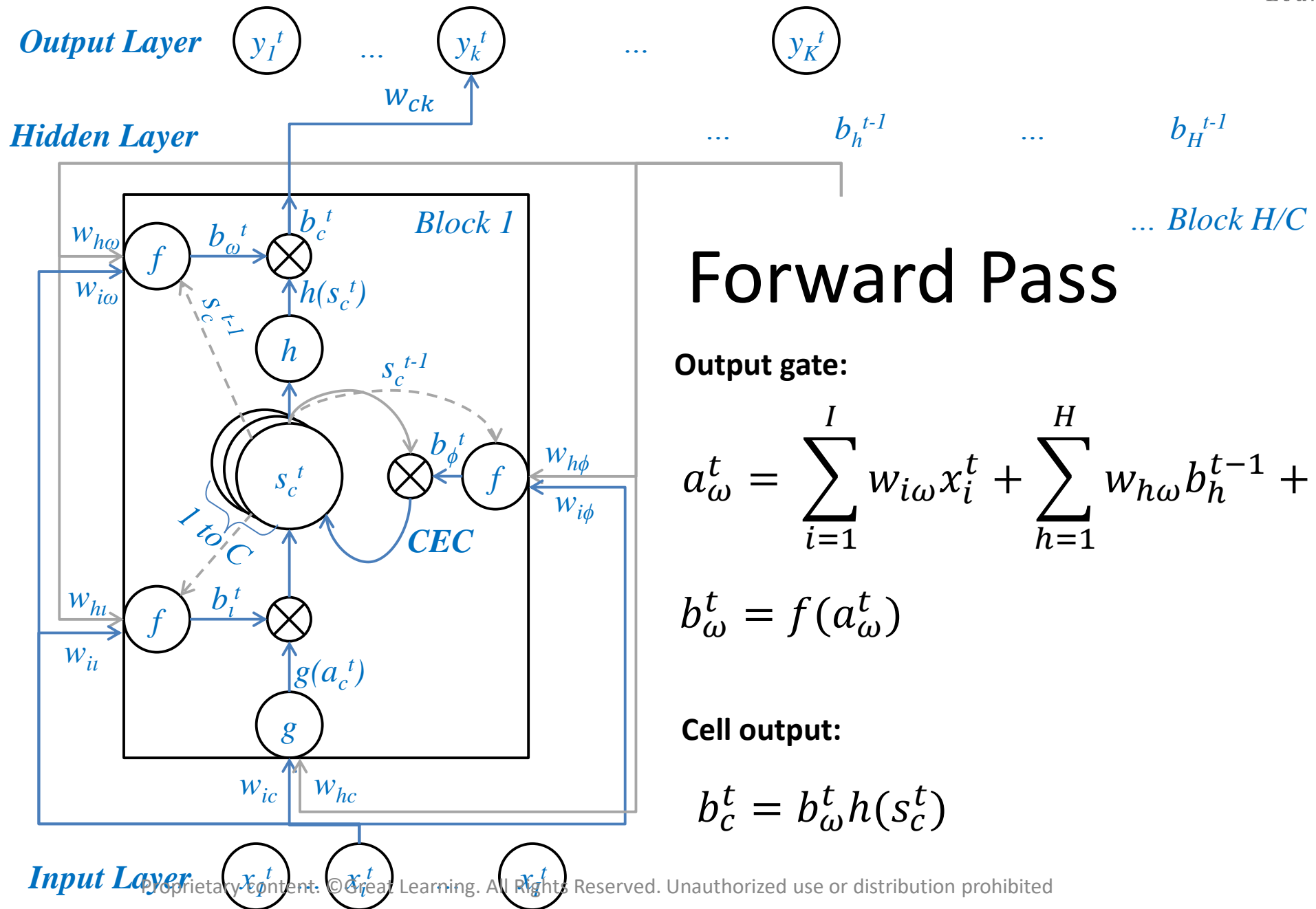


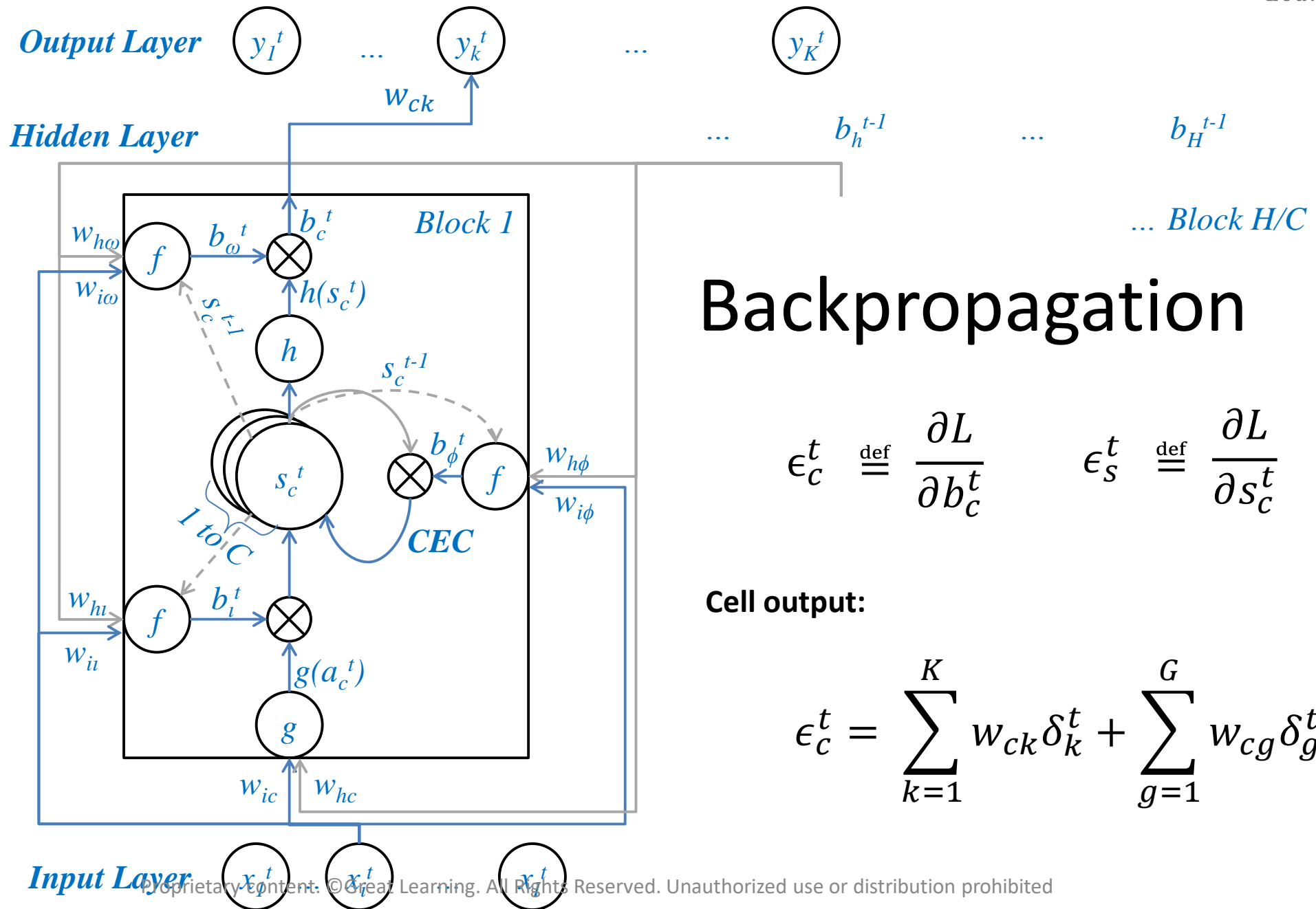
**Adding peep-holes**

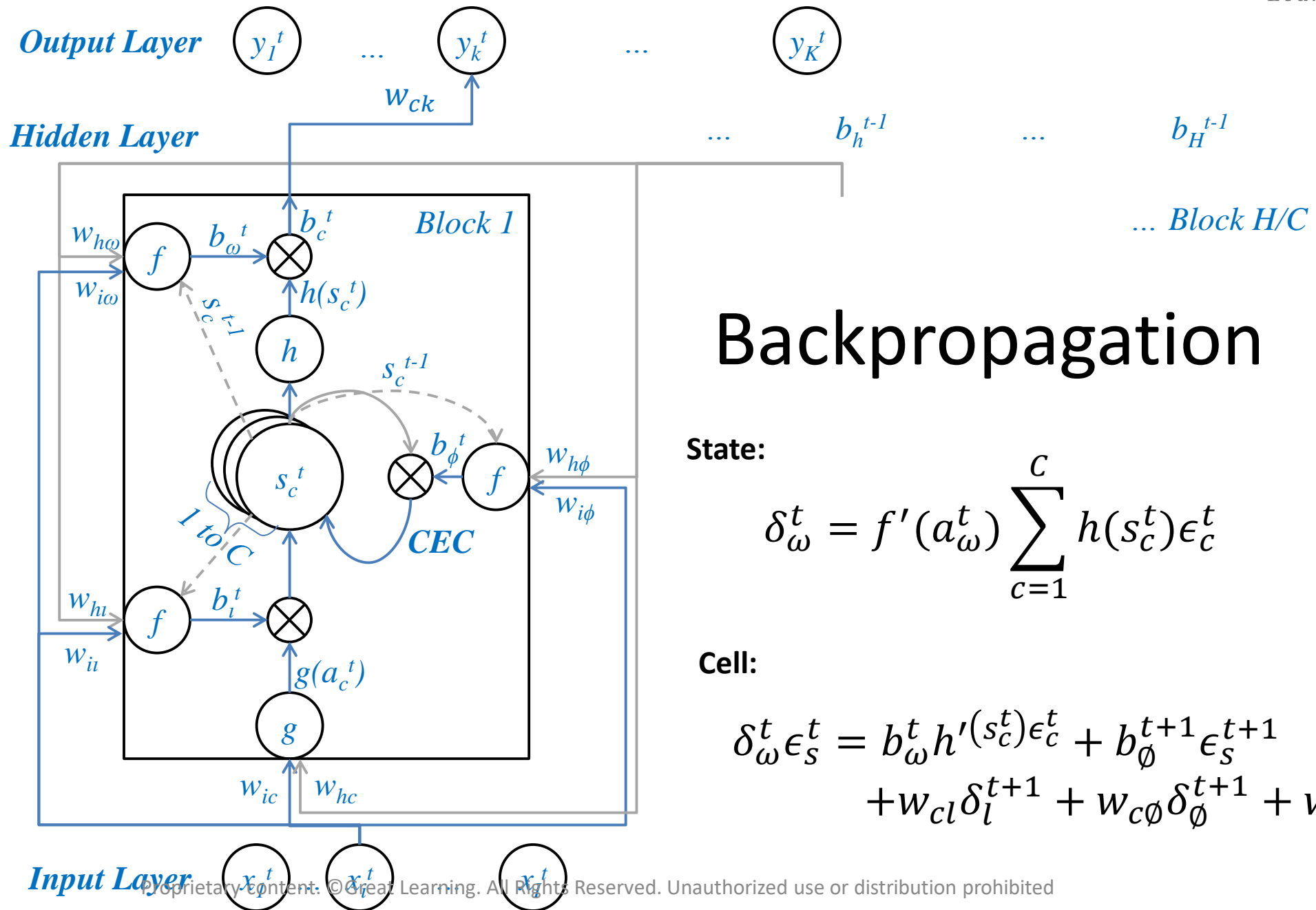


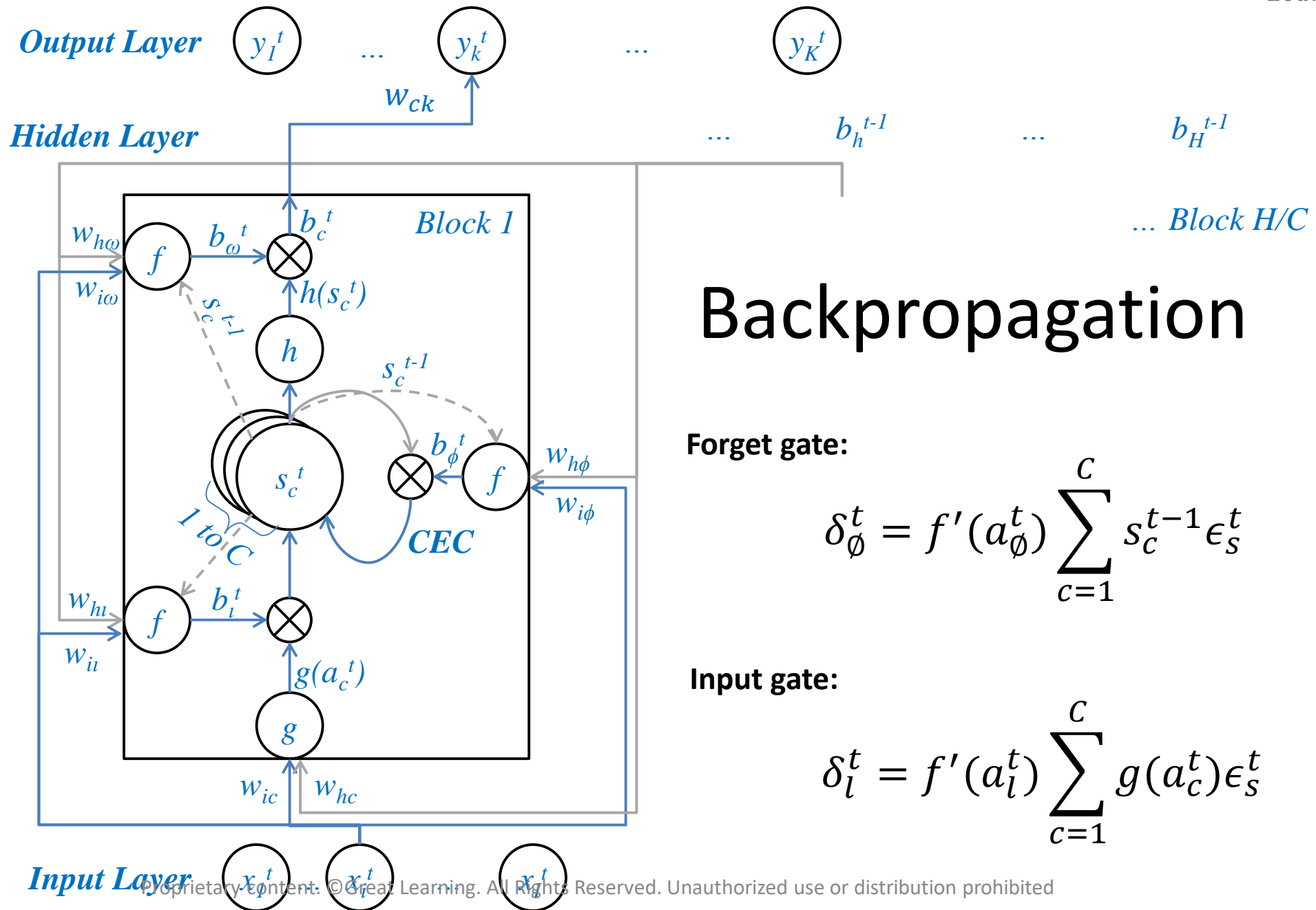












# Contents

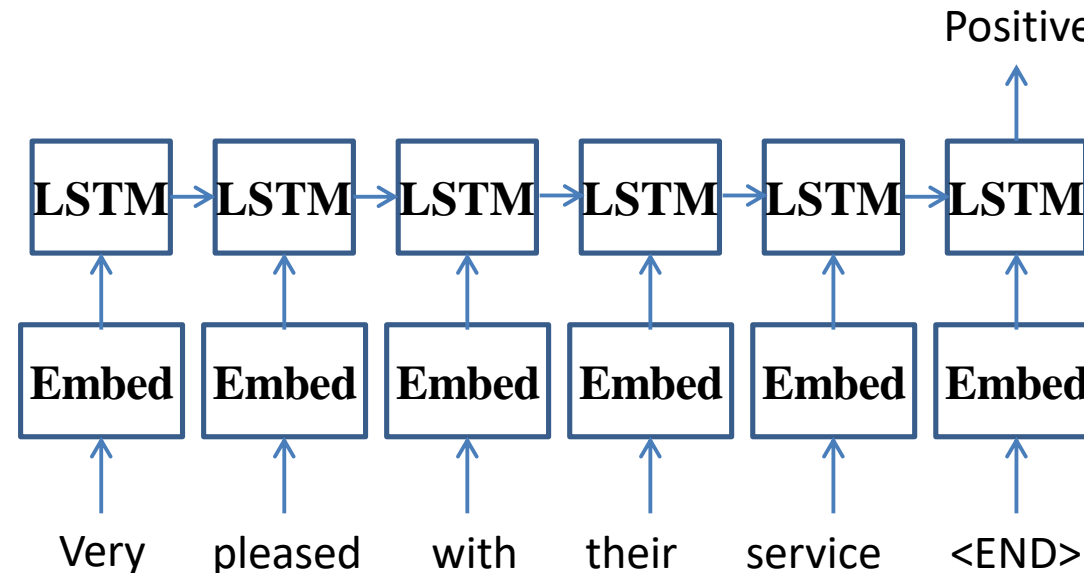
- Need for memory to process sequential data
- Recurrent neural networks
- LSTM basics
- **Some applications of LSTM in NLP**
- Some advanced LSTM structures

# Pre-processing for NLP

- The most basic pre-processing is to convert words into an embedding using Word2Vec or GloVe
- Otherwise, a one-hot-bit input vector can be too long and sparse, and require lots on input weights

# Sentiment analysis

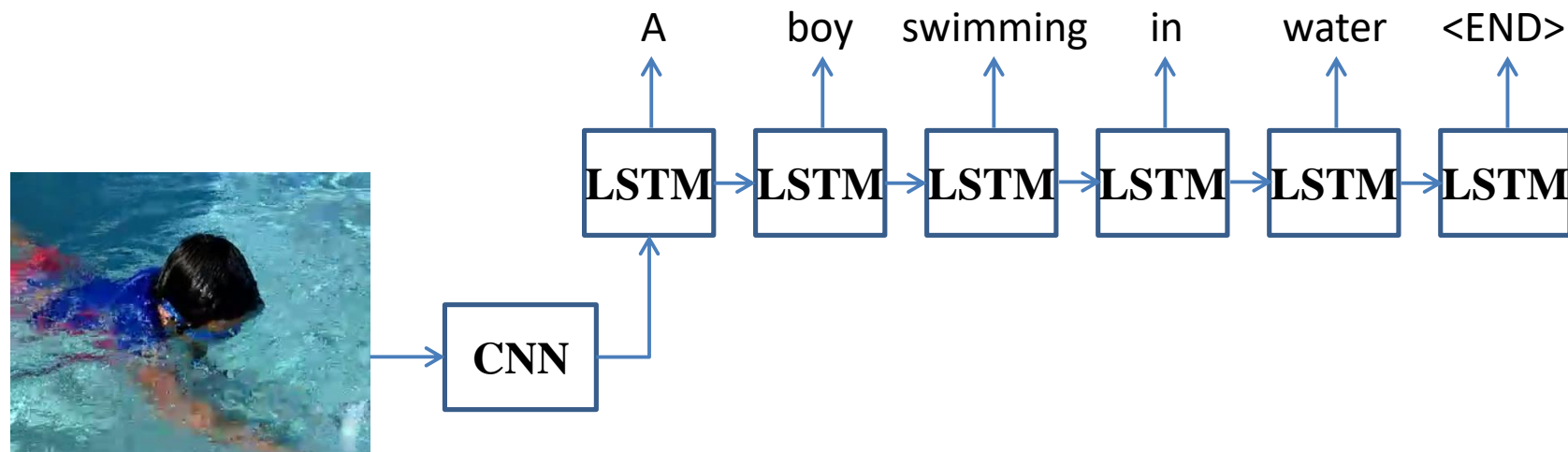
- Very common for customer review or new article analysis
- Output before the end can be discarded (not used for backpropagation)
- This is a many-to-one task





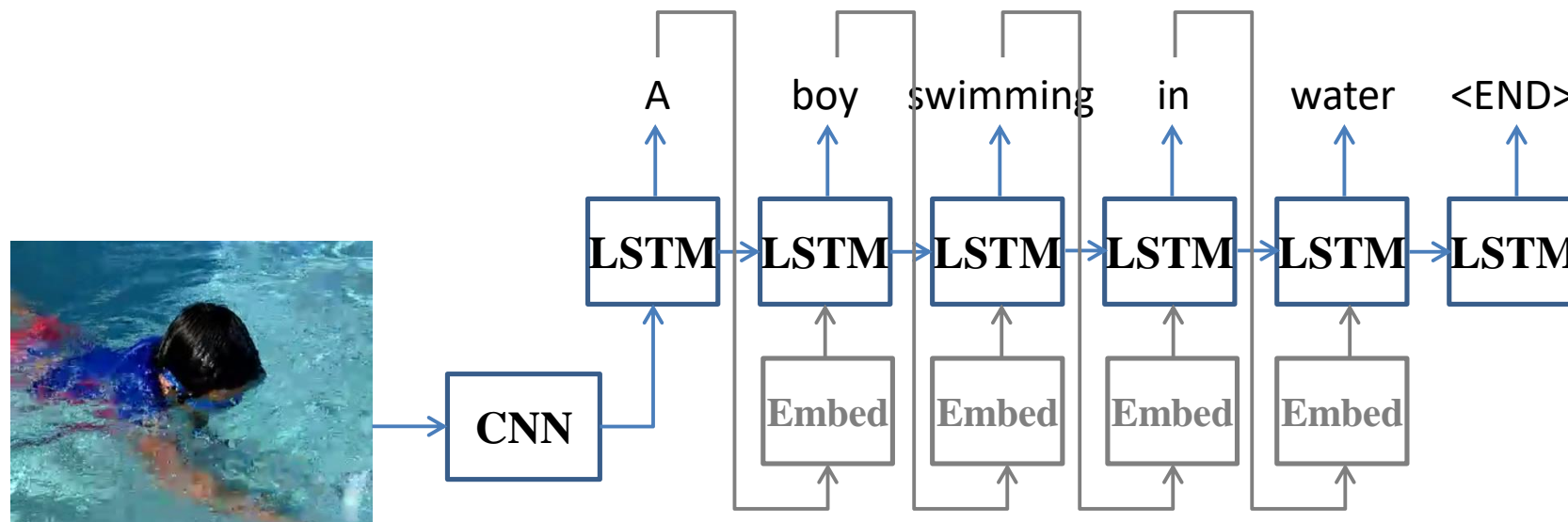
# Sentence generation

- Very common for image captioning
- Input is given only in the beginning
- This is a one-to-many task



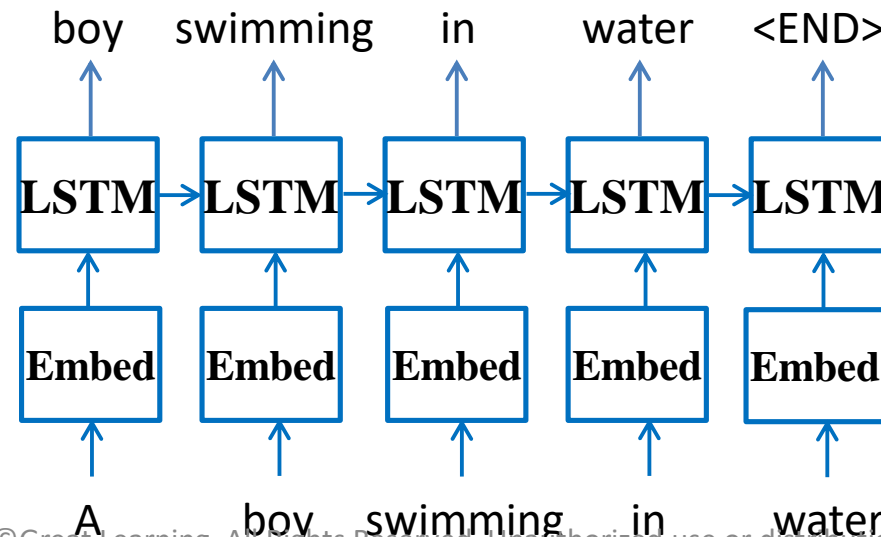
# Sentence generation

- Very common for image captioning
- Input is given only in the beginning
- This is a one-to-many task



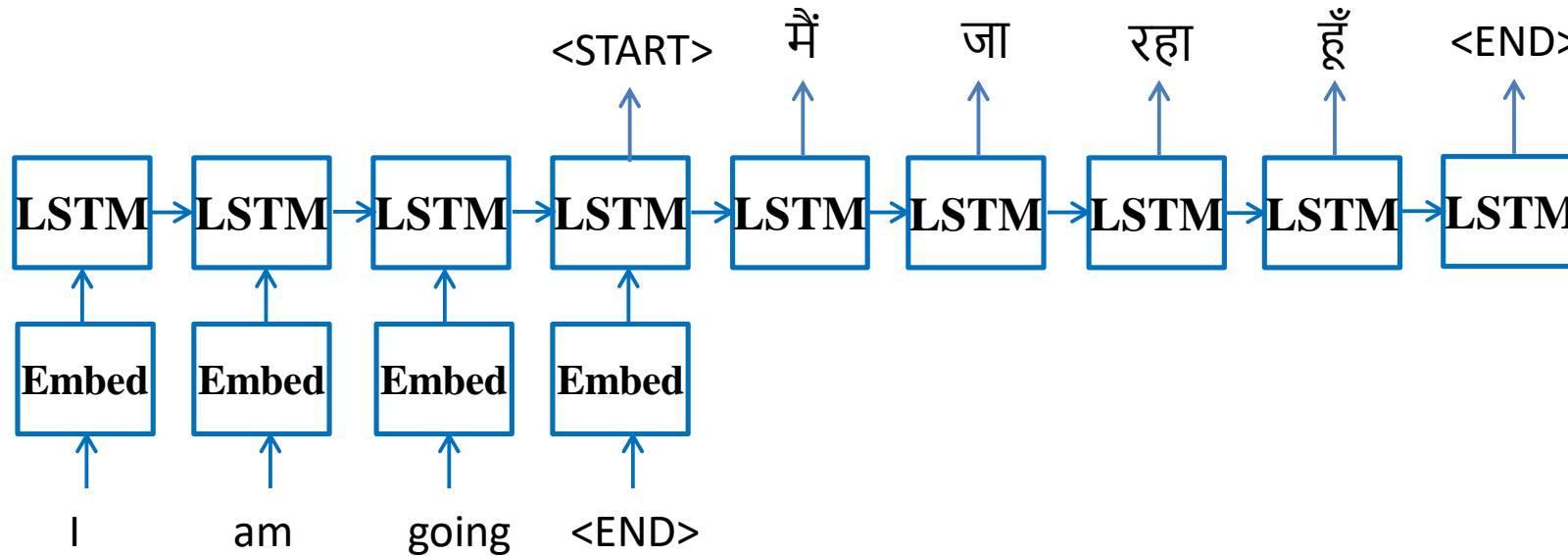
# Pre-training LSTMs

- Learning to predict the next word can imprint powerful language models in LSTMs
- This captures the grammar and syntax
- Usually, LSTMs are pre-trained on corpora



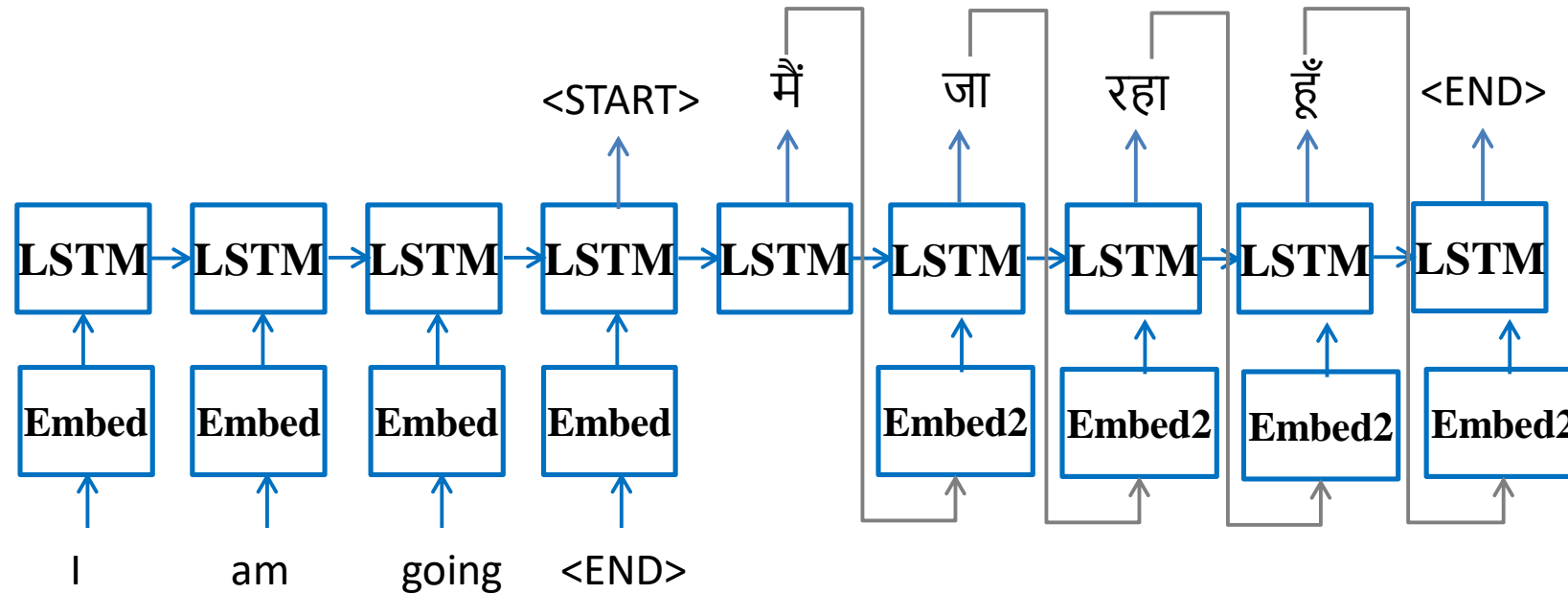
# Machine translation

- A naïve model would be to use a many-to-many network and directly train it



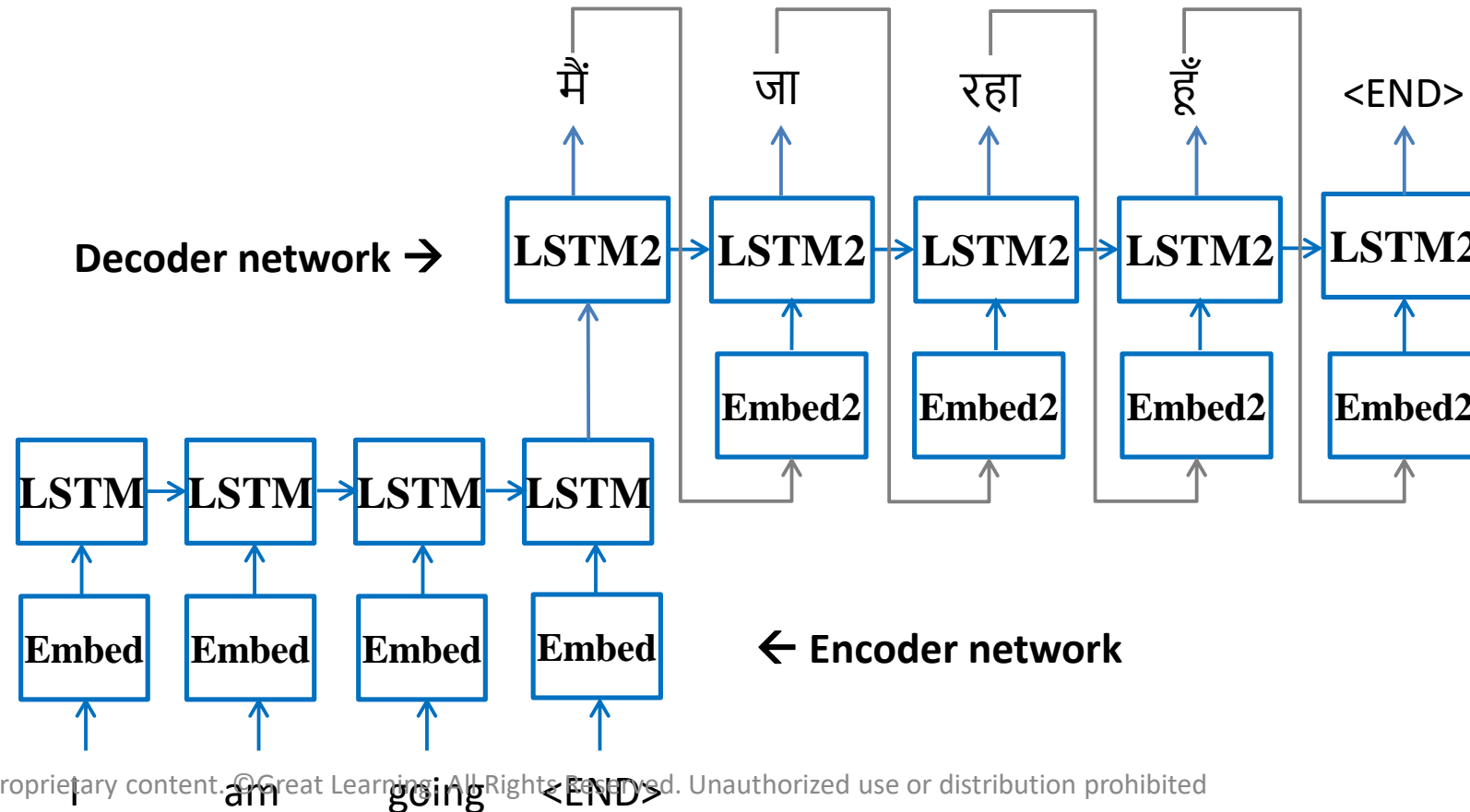
# Machine translation

- One could also feed in the output to the next instance input to predict a coherent structure



# Machine translation

- In actuality, one would use separate LSTMs pre-trained on two different languages

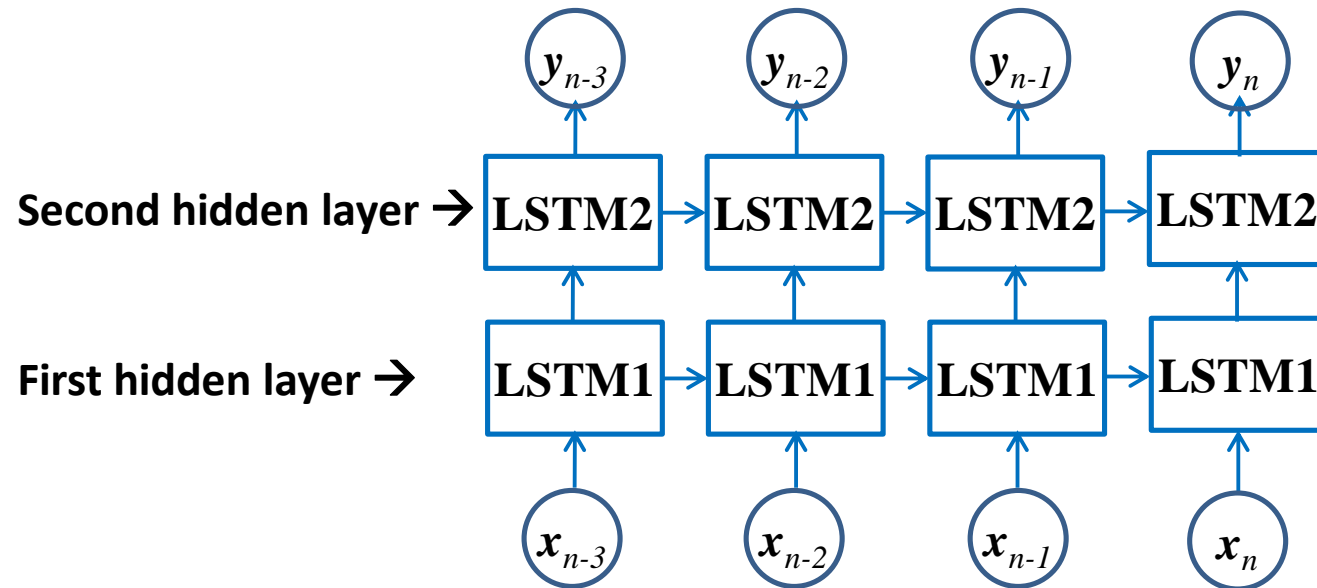


# Contents

- Need for memory to process sequential data
- Recurrent neural networks
- LSTM basics
- Some applications of LSTM in NLP
- **Some advanced LSTM structures**

# Multi-layer LSTM

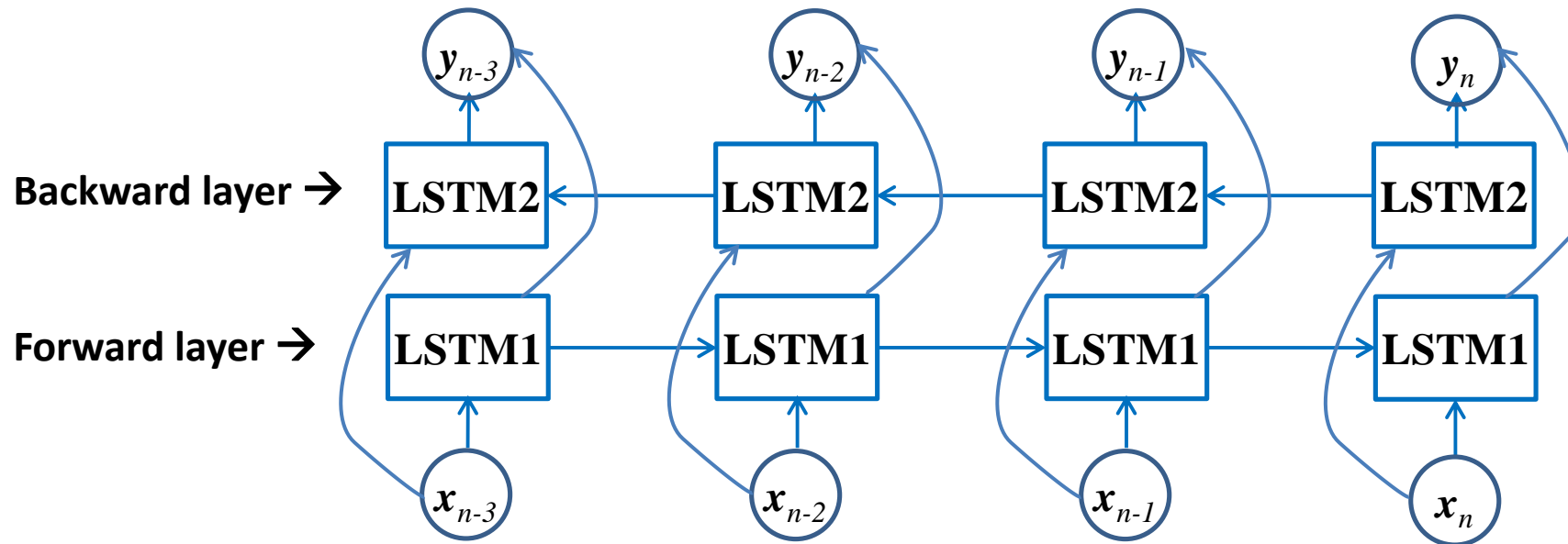
- More than one hidden layer can be used



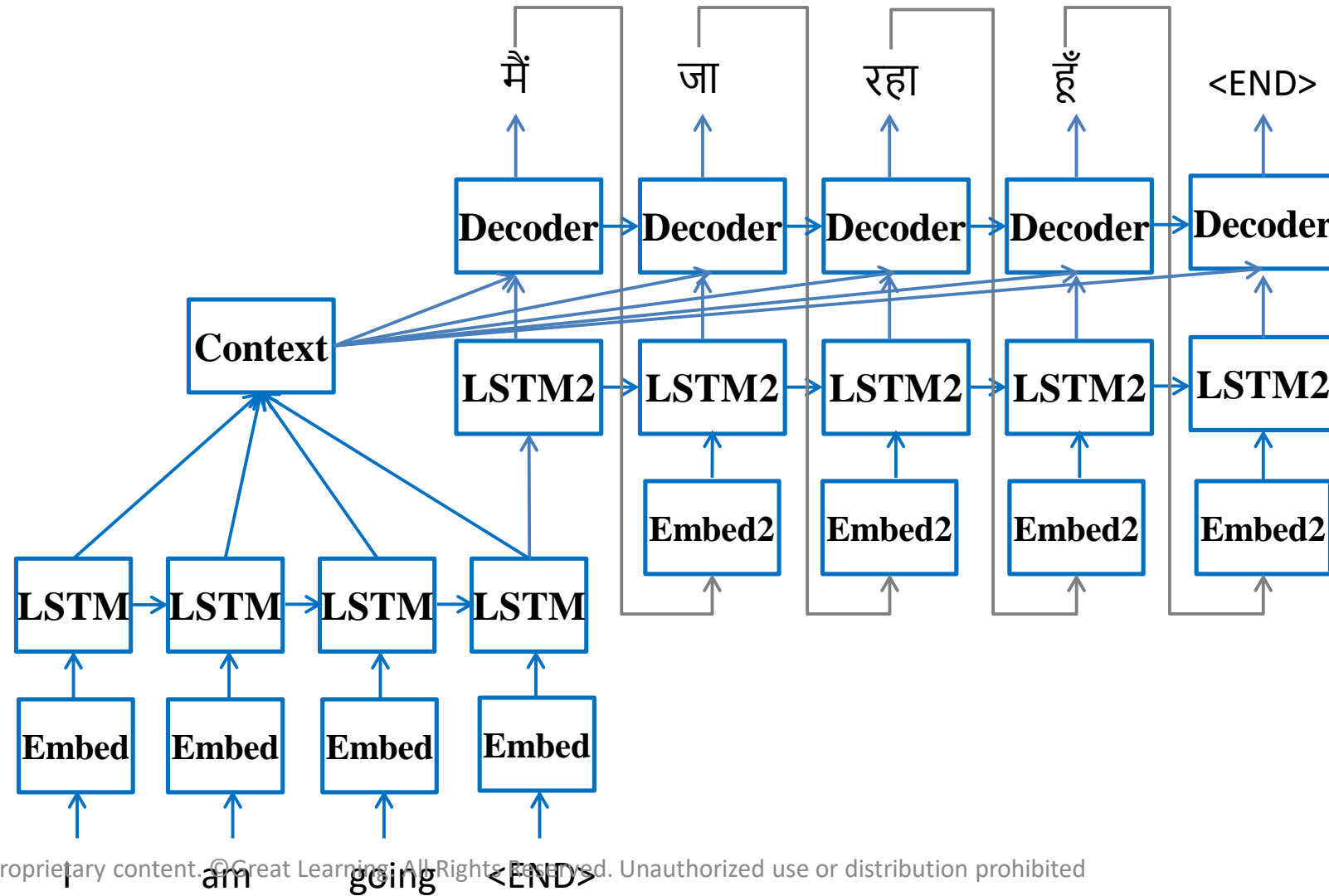


# Bi-directional LSTM

- Many problems require a reverse flow of information as well
- For example, POS tagging may require context from future words



# LSTM with Attention Mechanism



# Some problems in LSTM and its troubleshooting

- Inappropriate model
  - Identify the problem: One-to-many, many-to-one etc.
  - Loss only for outputs that matter
  - Separate LSTMs for separate languages
- High training loss
  - Model not expressive
    - Too few hidden nodes
    - Only one hidden layer
- Overfitting
  - Model has too much freedom
    - Too many hidden nodes
    - Too many blocks
    - Too many layers
    - Not bi-directional