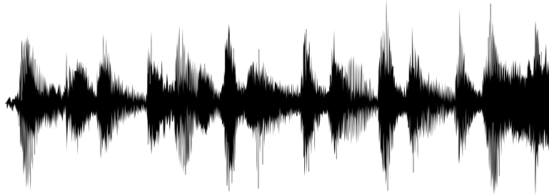
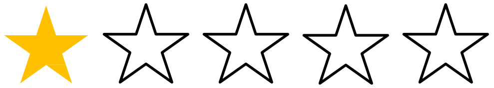
COSE474 Deep Learning

Lecture 11: Recurrent Neural Networks (RNNs)

Seungryong Kim

Computer Vision Lab. (CVLAB)
Department of Computer Science and Engineering
Korea University

Examples of Sequence Data

- Speech Recognition:  → "The quick brown fox jumped over the lazy dog."
- Music Generation: ∅ → 
- Sentiment Classification: "There is nothing to like in this movie." → 
- DNA Sequence Analysis: AGCCCCTGTGAGGAACTAG → AG**CCCCTGTGAGGAACT**AG
- Machine Translation: Voulez-vous chanter avec moi? → Do you want to sing with me?
- Video Activity Recognition:  → Running
- Name Entity Recognition: Yesterday, Harry Potter met Hermione Granger. → Yesterday, **Harry Potter** met **Hermione Granger**.

[slide courtesy: Stanford, CS230]

Examples of Sequence Data

- **Sequential Data Representation**

Input data: a set of vectors $\mathbf{x} = \left(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \cdots, \mathbf{x}^{(T)}\right)^{\mathrm{T}}$

ECG signal (3 channels)
100 samples/sec and acquire for 2 mins
-> 12,000 x 3 vector (*T*: 12,000)

$$\mathbf{x} = \left(\begin{pmatrix} 0.3 \\ 0.1 \\ 0.2 \end{pmatrix}, \begin{pmatrix} 0.5 \\ 0.6 \\ 0.4 \end{pmatrix}, \cdots \cdots \right)^{\mathrm{T}}$$

Examples of Sequence Data

- ***Text* sequential data representation**
  - Text data is represented by using a dictionary.
  - Ex) The dictionary is built with the most frequently used 30,000 words [1].

- ***Text* representation using dictionary**
  - Bag-of-words (BoW)
  - One hot code
  - Word embedding

[1] On the Properties of Neural Machine Translation: Encoder-Decoder Approaches

Examples of Sequence Data

- **Bag-of-Words (BoW)**
  - Count the frequency of each word and express it as a vector of m dimensions.
  - m: dictionary size

'April is the cruelest month' $\rightarrow$ $(0, \ldots ,0.2, \ldots, 0.2, \ldots)^{\mathrm{T}}$

  - It is good for image retrieval but is not suitable for sequential data representation.

- **One Hot Code**
  - Each word is represented by $m\times 1$ vector → very inefficient.

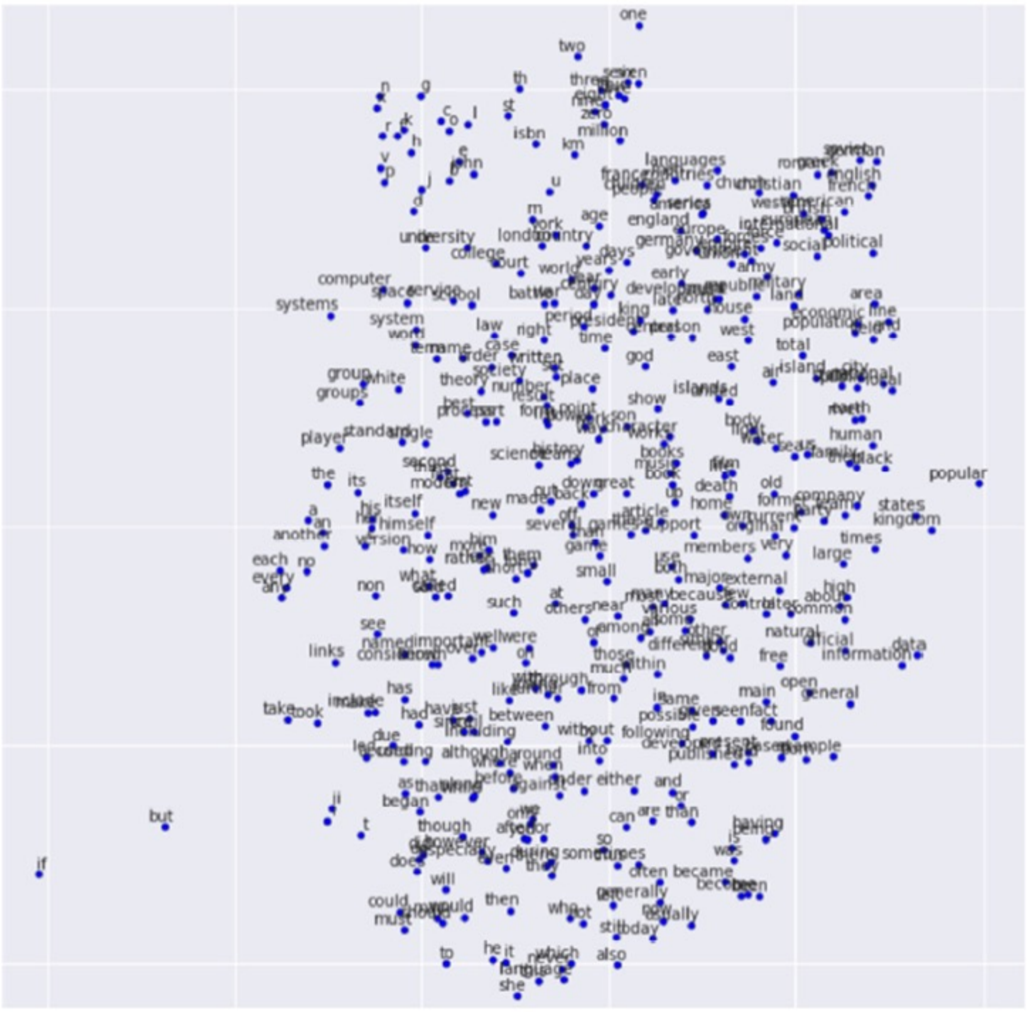
'April is the cruelest month' $\rightarrow$ $\left((0,0,1,0,0,0,\cdots)^{\mathrm{T}},(0,0,0,0,1,0,\cdots)^{\mathrm{T}},\cdots\right)^{\mathrm{T}}$

[1] On the Properties of Neural Machine Translation: Encoder-Decoder Approaches

Examples of Sequence Data

- **Word Embedding**
  - By analyzing the correlation between words in dictionary, convert the word into lower dimensional space
  - Ex) word in 30,000 dimensions → converted into 620 dimension [1]

Word embedding using word2vec



[1] On the Properties of Neural Machine Translation: Encoder-Decoder Approaches

Examples of Sequence Data

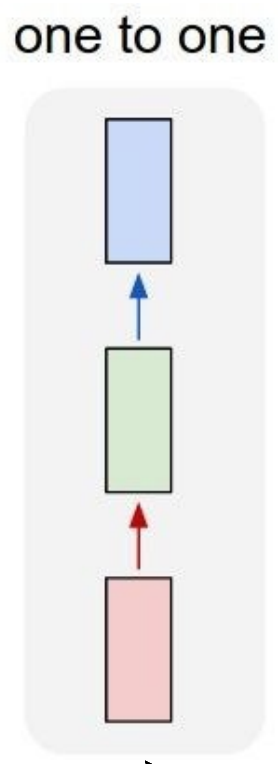
- **The *order* in which the features appear is important.**
- **Training sample may have different sizes**
- **Context dependency**
  - In non-sequential data, covariance matrix indicates a correlation over features.
  - In sequential data, context dependency is important

She got up at lunch and ate breakfast, and he came back later.

Long-term dependency
(Two distant words are correlated)

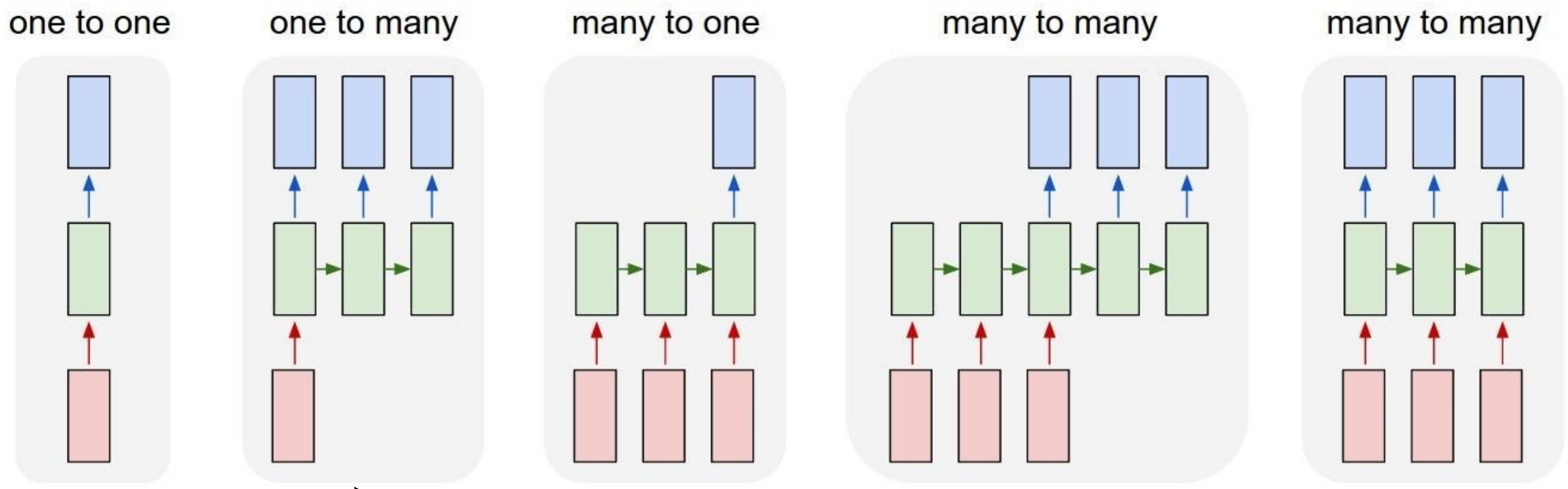
Use LSTM for handling this!

“Vanilla” Neural Networks



Vanilla Neural Networks

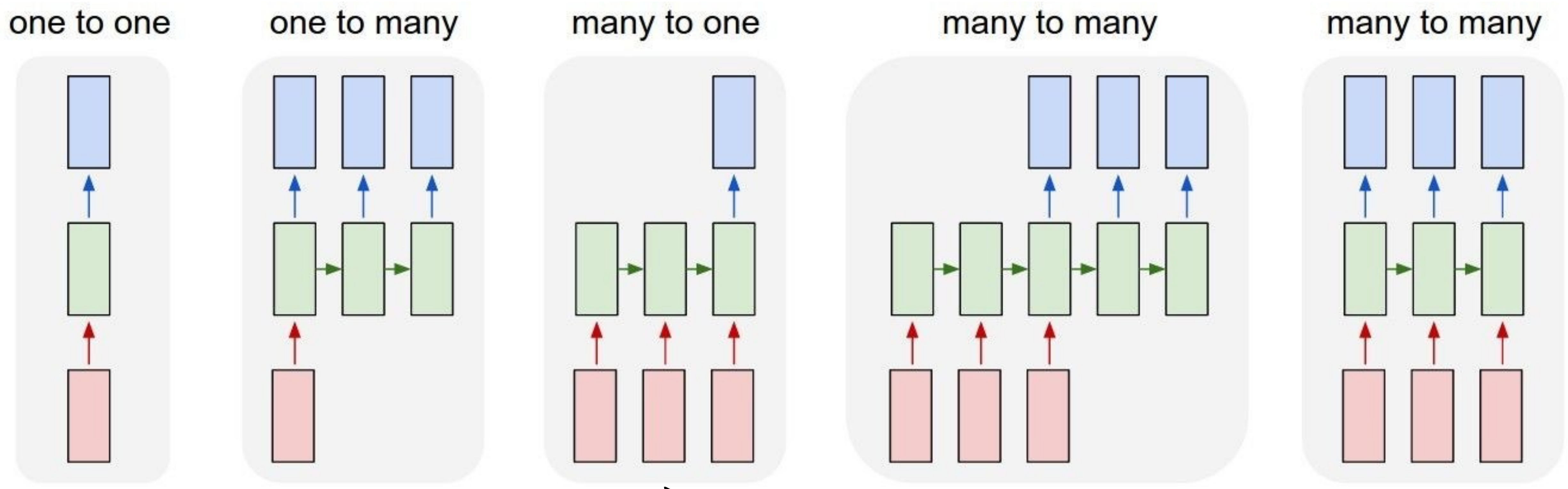
Recurrent Neural Networks (RNNs)



e.g., **Image Captioning**
Image → Sequence of words

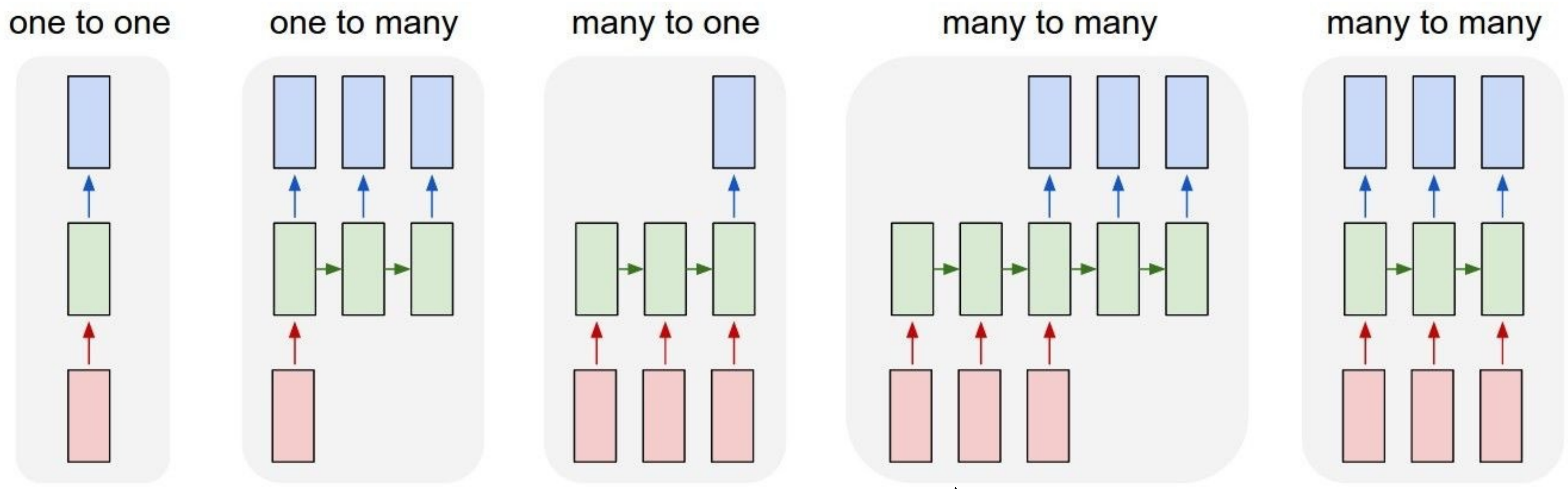
[slide courtesy: Stanford, CS231]

Recurrent Neural Networks (RNNs)



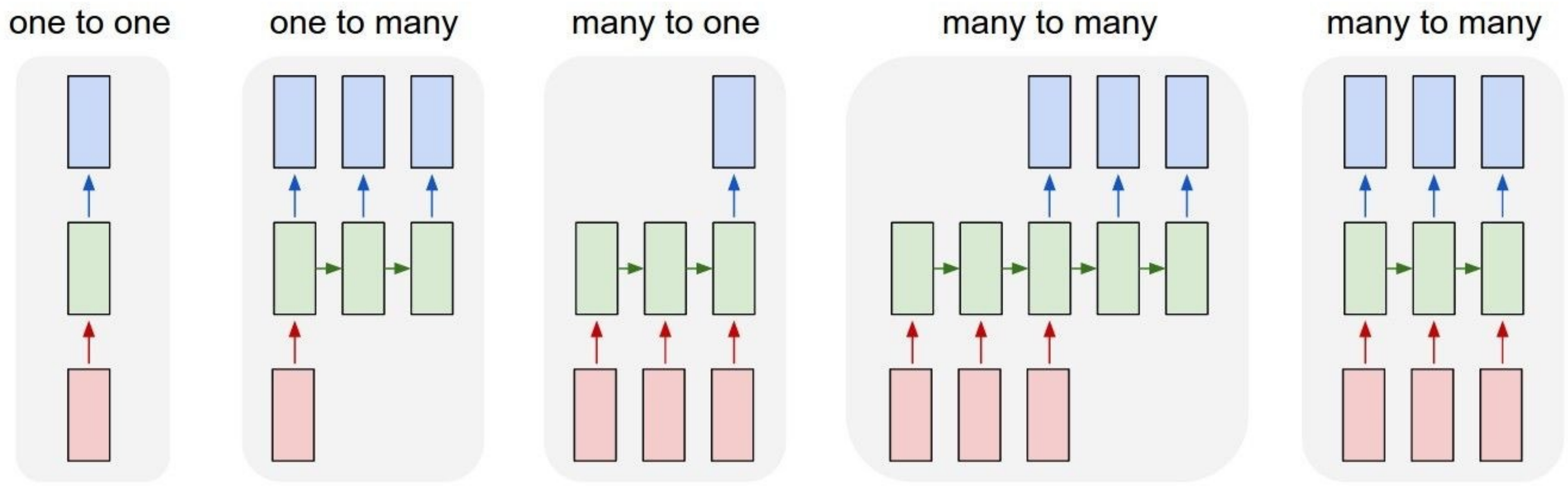
e.g., **Action Prediction**
Sequence of video frames → Action class

Recurrent Neural Networks (RNNs)



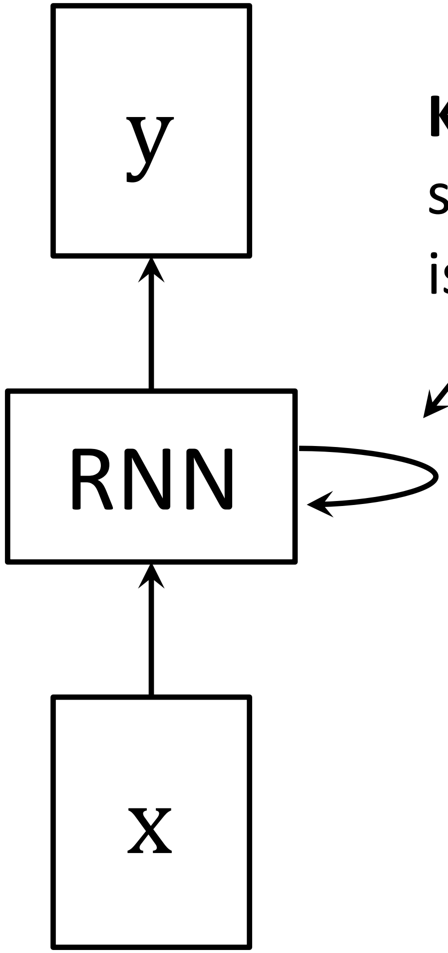
e.g., **Video Captioning**
Sequence of video frames
→ Caption

Recurrent Neural Networks (RNNs)



e.g., **Video classification on frame level**

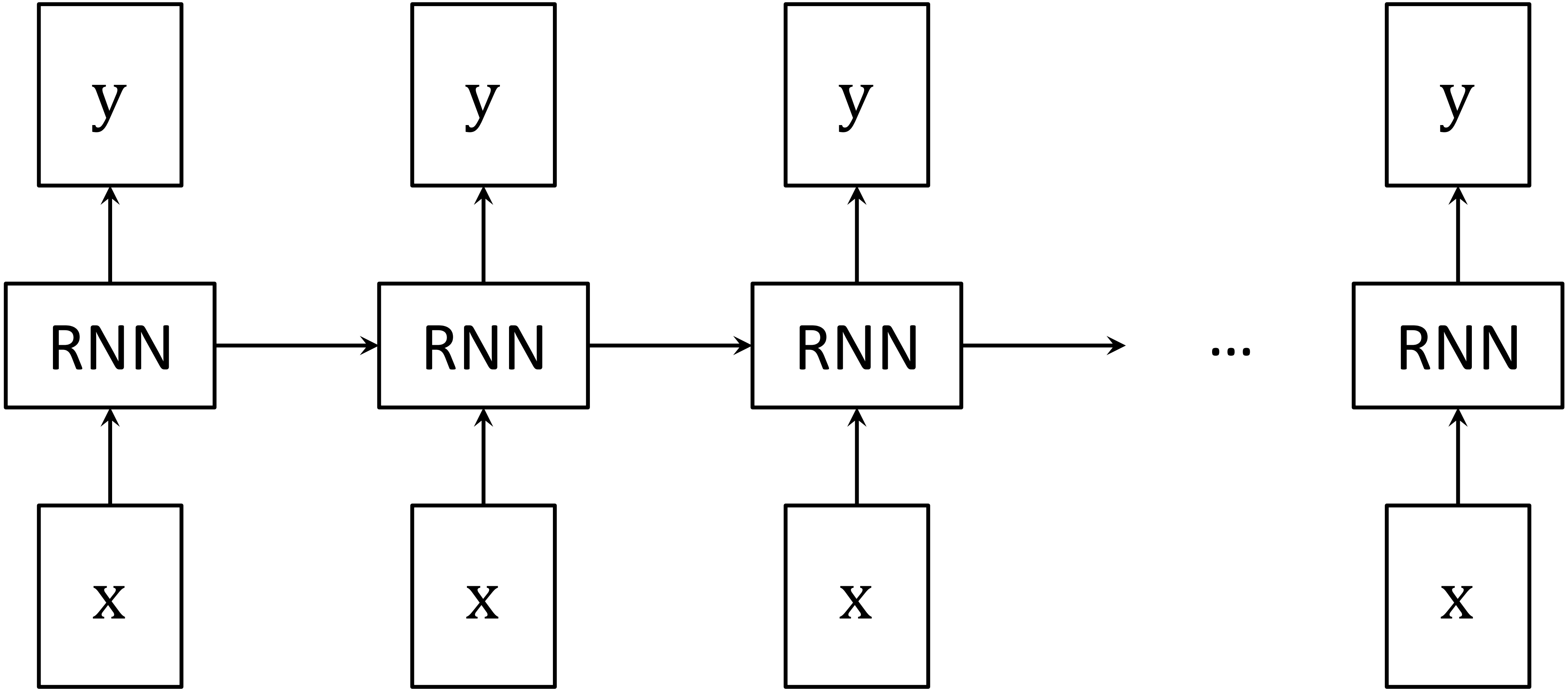
Recurrent Neural Networks (RNNs)



Key-idea: RNNs have an "internal state" that is updated as a sequence is processed

[slide courtesy: Stanford, CS231]

Recurrent Neural Networks (RNNs)



Recurrent Neural Networks (RNNs)

- We can process a sequence of vectors x by applying a **recurrence formula** at every time step:

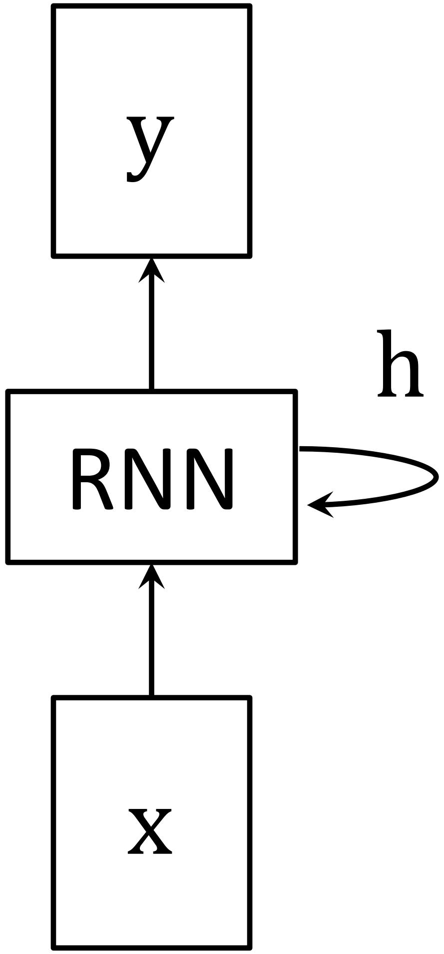
$$\mathrm{h}_t = f_W(\mathrm{h}_{t-1}, \mathrm{x}_t)$$

New state

Some function with parameters W

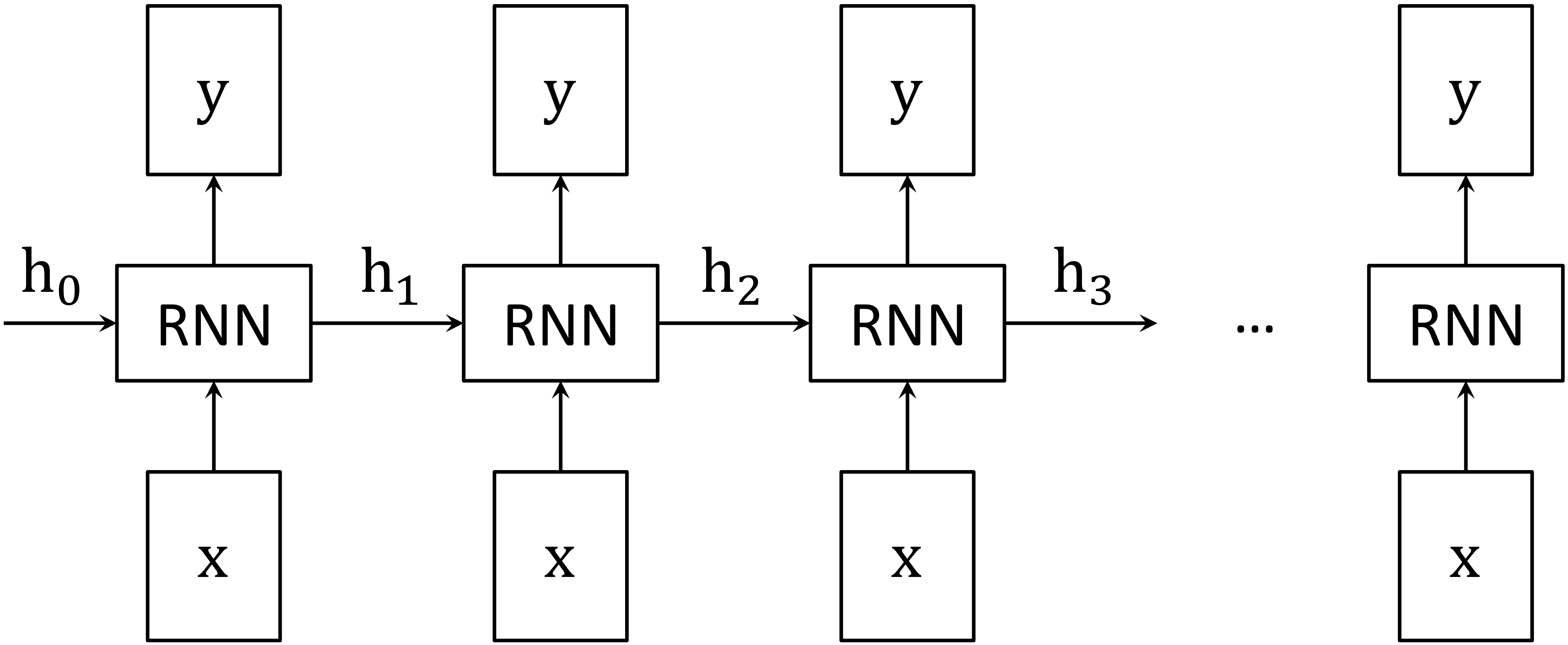
Old state

Input vector at some time step



Notice: the same function and the same set of parameters are used at every time step.

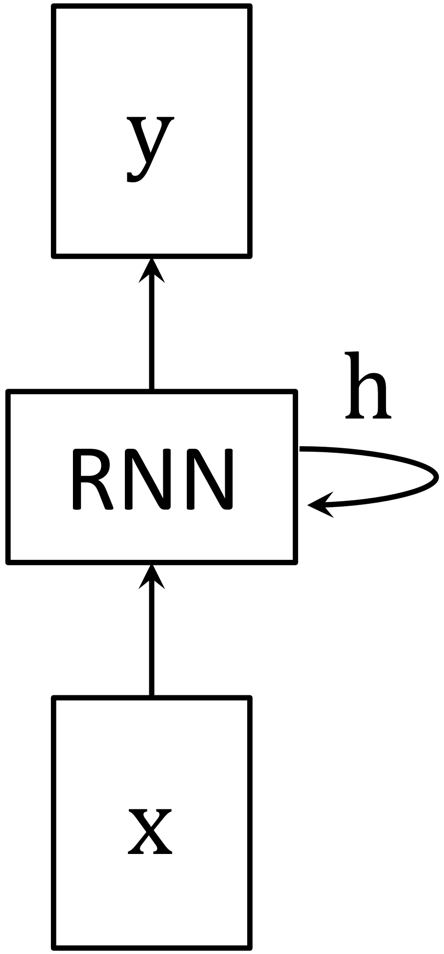
Recurrent Neural Networks (RNNs)



[slide courtesy: Stanford, CS231]

Simple Recurrent Neural Networks (RNNs)

- The state consists of a single "*hidden*" vector h:



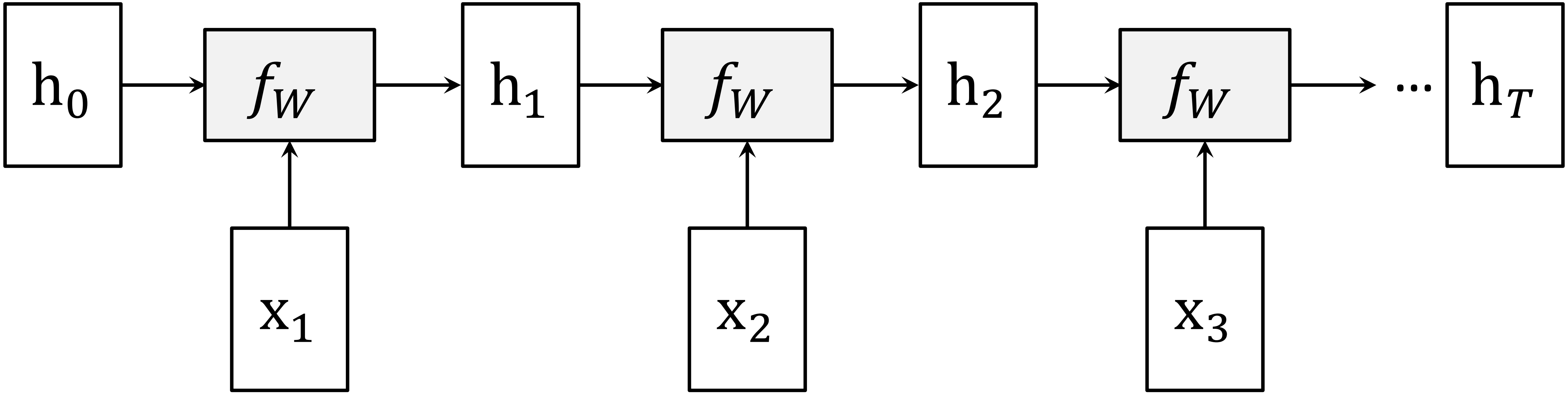
$$\mathrm{h}_t = f_W(\mathrm{h}_{t-1}, \mathrm{x}_t)$$

$$\downarrow$$

$$\mathrm{h}_t = \tanh(W_{\mathrm{hh}}\mathrm{h}_{t-1} + W_{\mathrm{xh}}\mathrm{x}_t)$$
$$\mathrm{y}_t = W_{\mathrm{hy}}\mathrm{h}_t$$

[slide courtesy: Stanford, CS231]

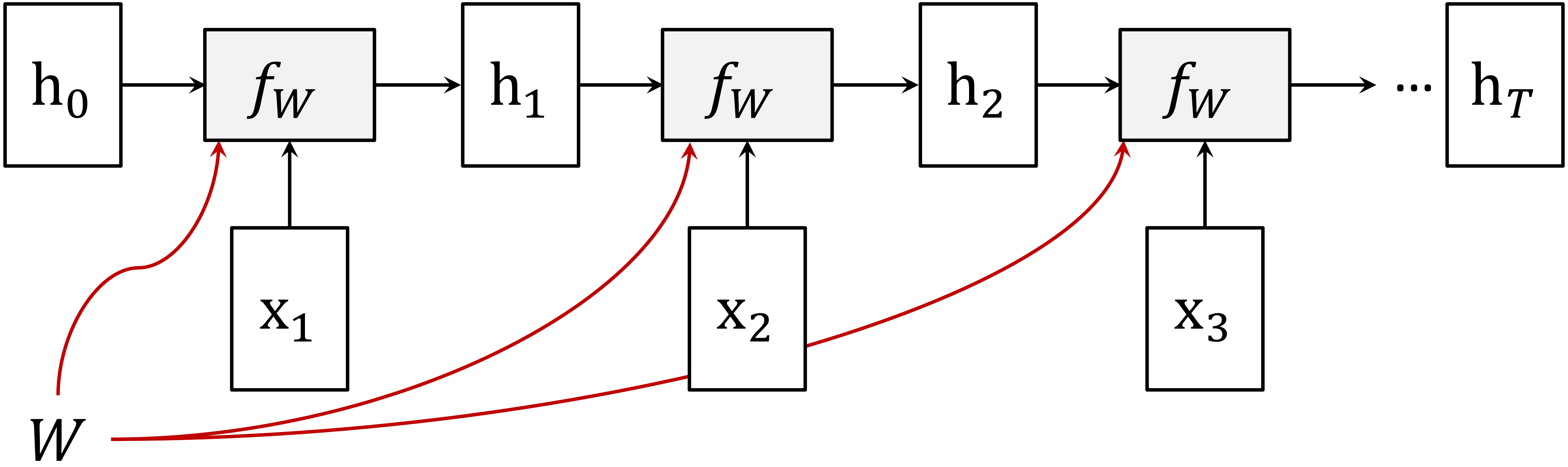
RNN: Computational Graph



[slide courtesy: Stanford, CS231]

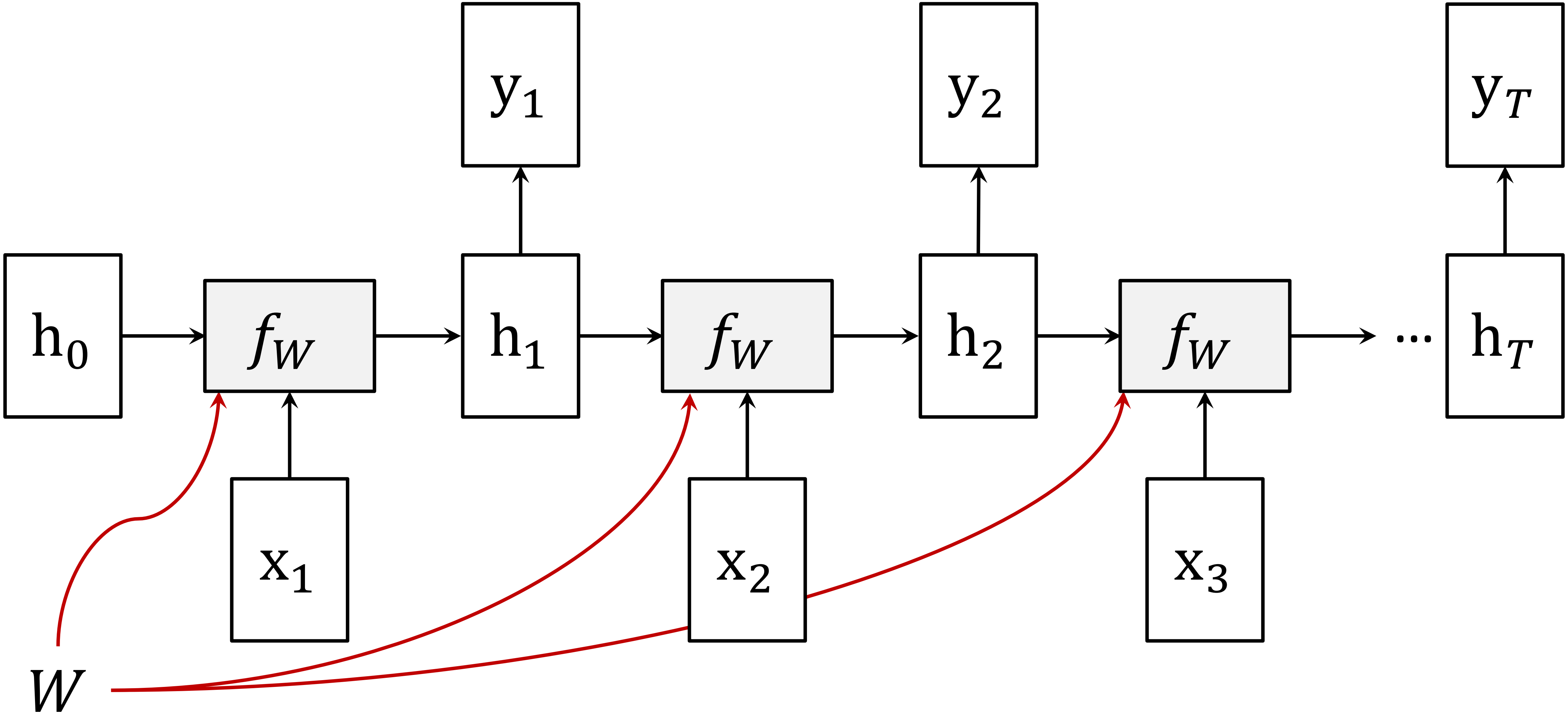
RNN: Computational Graph

- Re-use the same weight matrix at every time-step:



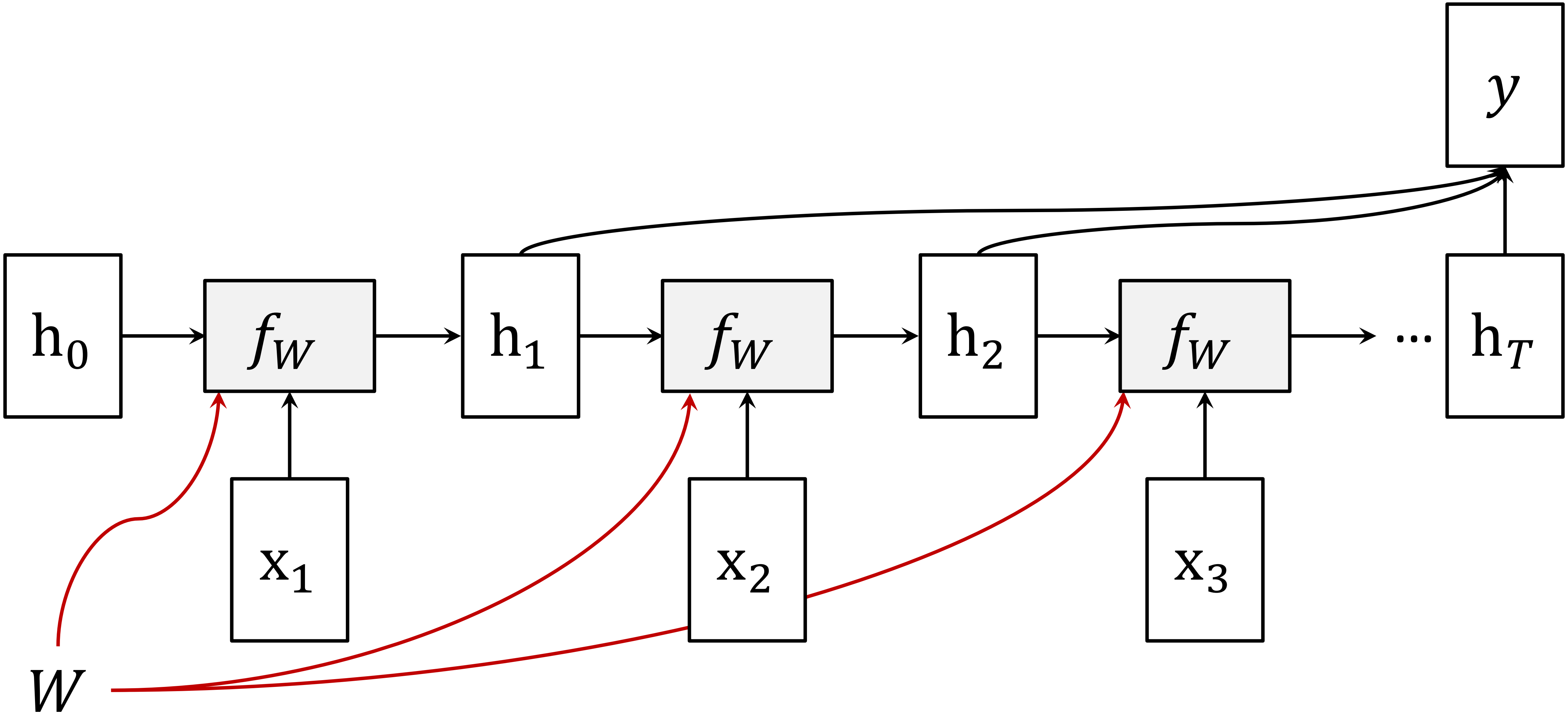
[slide courtesy: Stanford, CS231]

RNN: Computational Graph- Many to Many



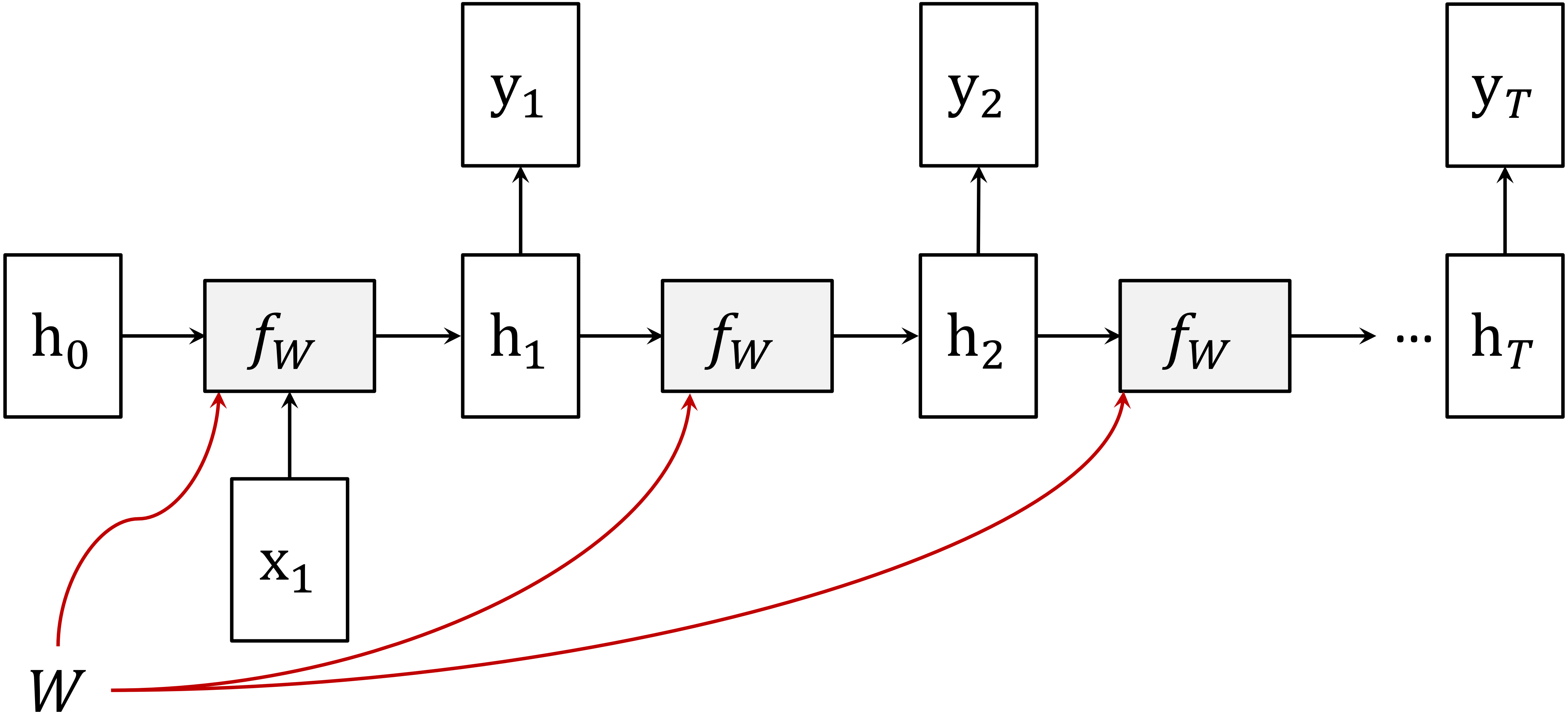
[slide courtesy: Stanford, CS231]

RNN: Computational Graph- Many to One



[slide courtesy: Stanford, CS231]

RNN: Computational Graph- One to Many

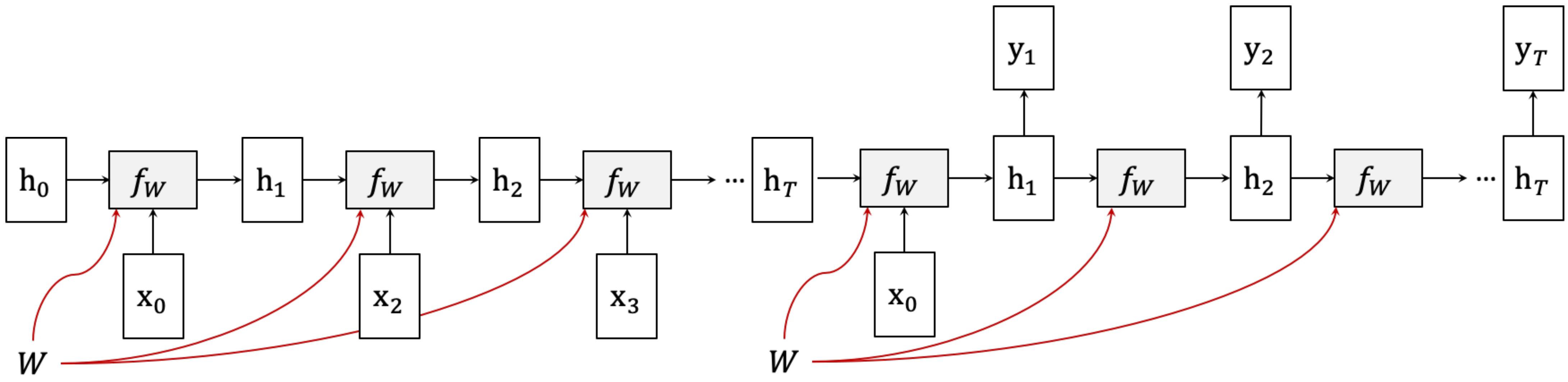


(Example) Sequence to Sequence

- “Many-to-One” + “One-to-Many”

Many-to-One: Encode input sequence in a single vector

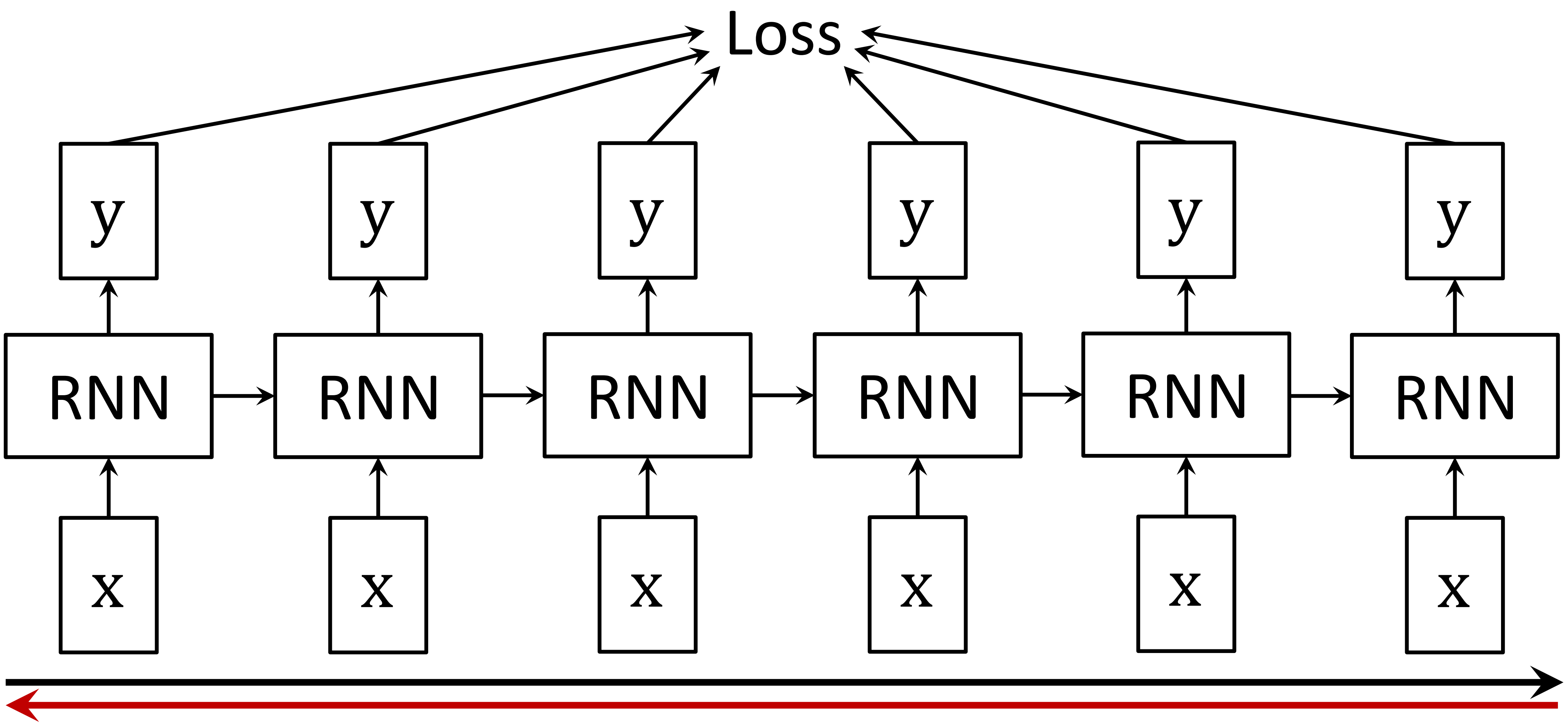
One-to-Many: Produce output sequence from single input vector



Sutskever et al., “Sequence to Sequence Learning with Neural Networks,” NIPS, 2014

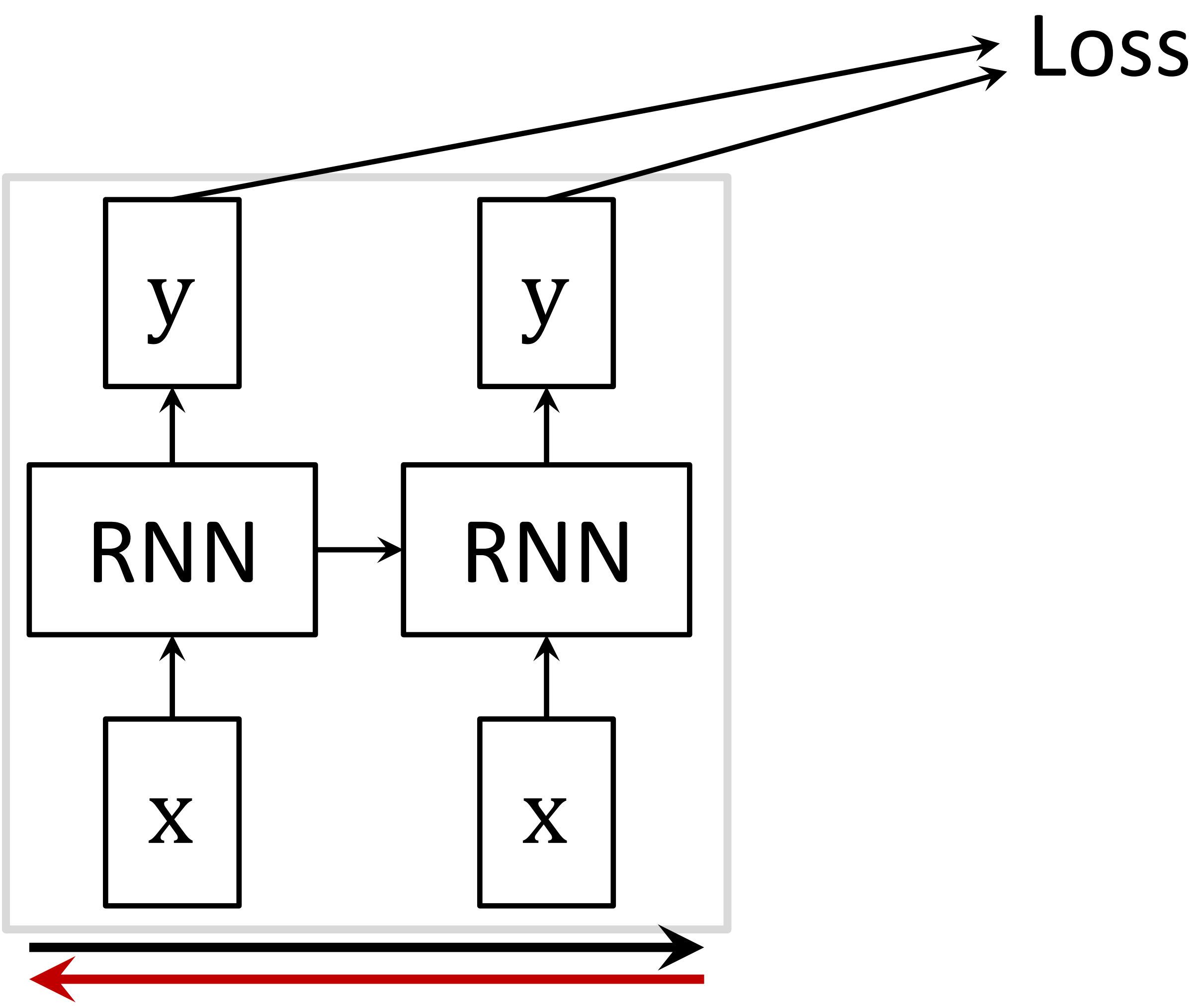
Backpropagation through Time

- Forward through entire sequence to compute loss, then backward through entire sequence to compute gradient:



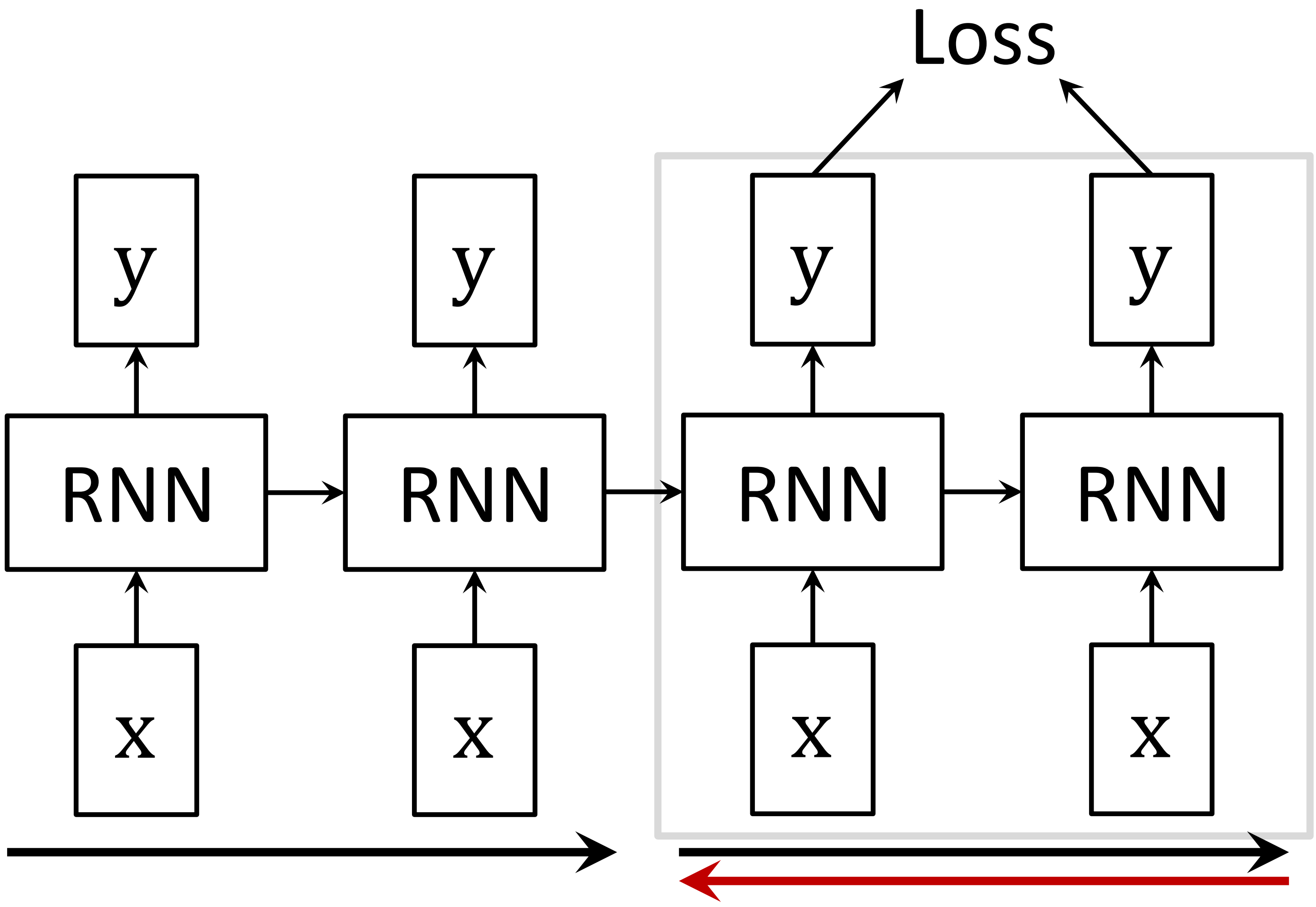
Truncated Backpropagation through Time

- Run forward and backward through chunks of the sequence instead of whole sequence:



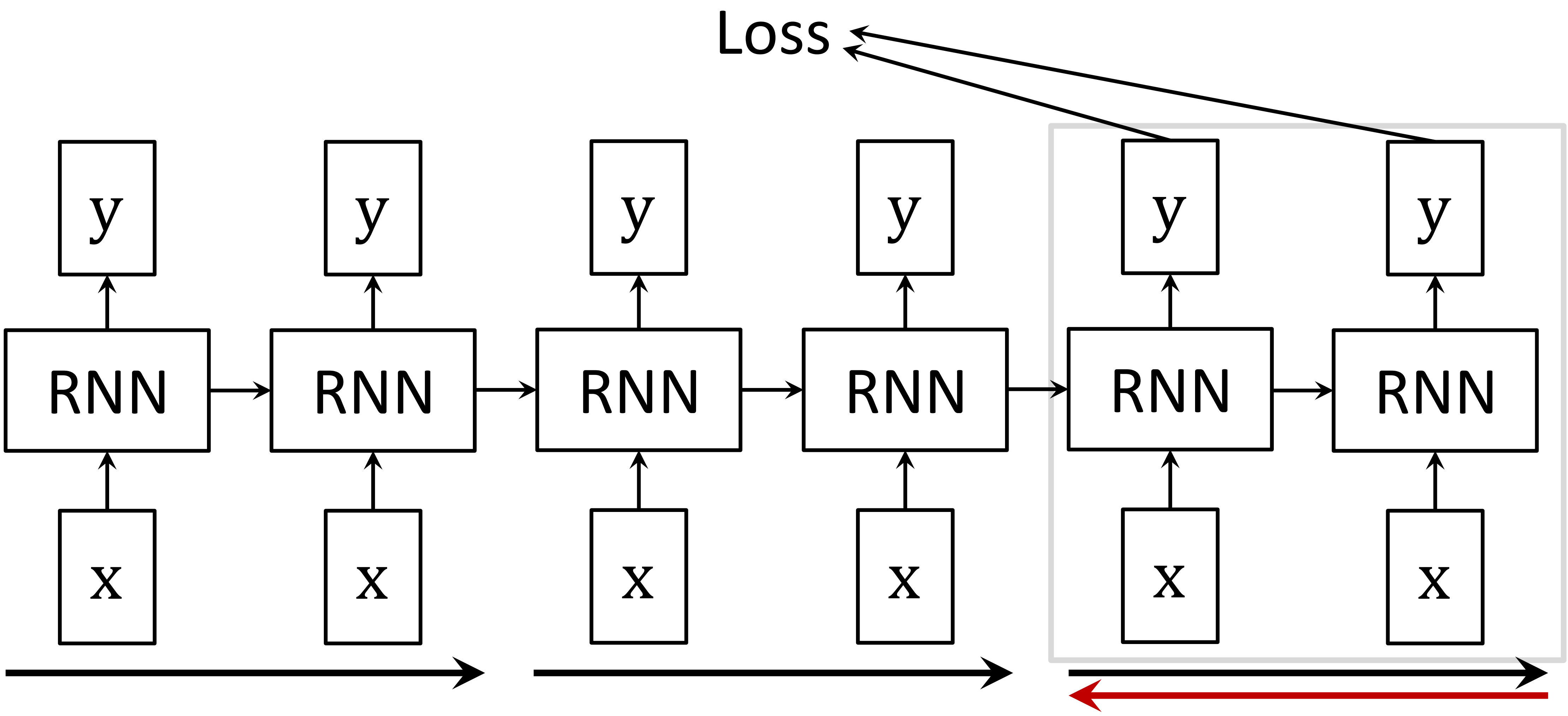
Backpropagation through Time

- Carry hidden states forward in time forever, but only backpropagate for some smaller number of steps:



Backpropagation through Time

- Carry hidden states forward in time forever, but only backpropagate for some smaller number of steps:



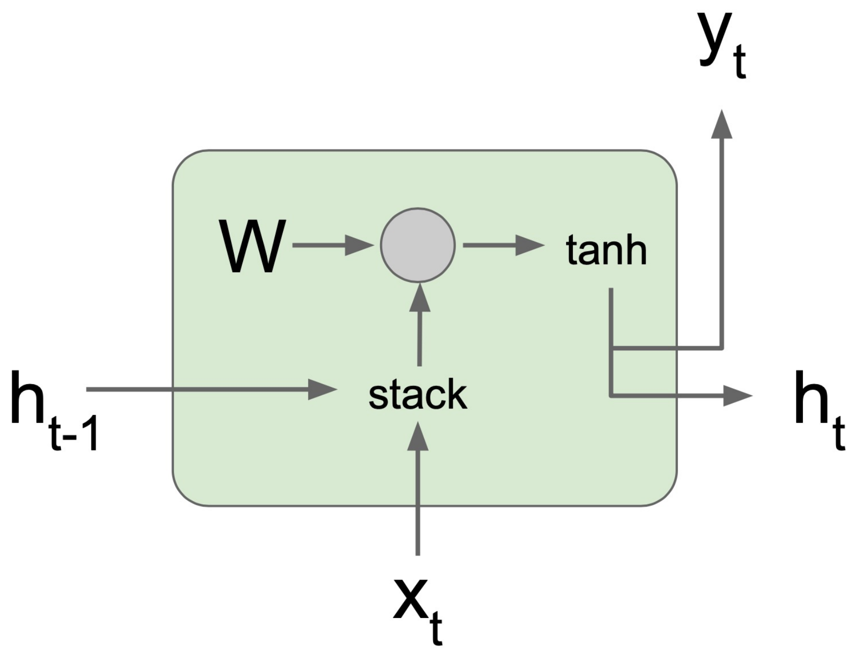
Recurrent Neural Networks Tradeoffs

- **RNN Advantage:**
  - Can process any length input.
  - Computation for step t can (in theory) use information from many steps back.
  - Model size does not increase for longer input.
  - Same weights applied on every timestep, so there is symmetry in how inputs are processed.

- **RNN Disadvantage:**
  - Recurrent computation is slow.
  - In practice, difficult to access information from many steps back.

[slide courtesy: Stanford, CS231]

Vanilla RNN Gradient Flow



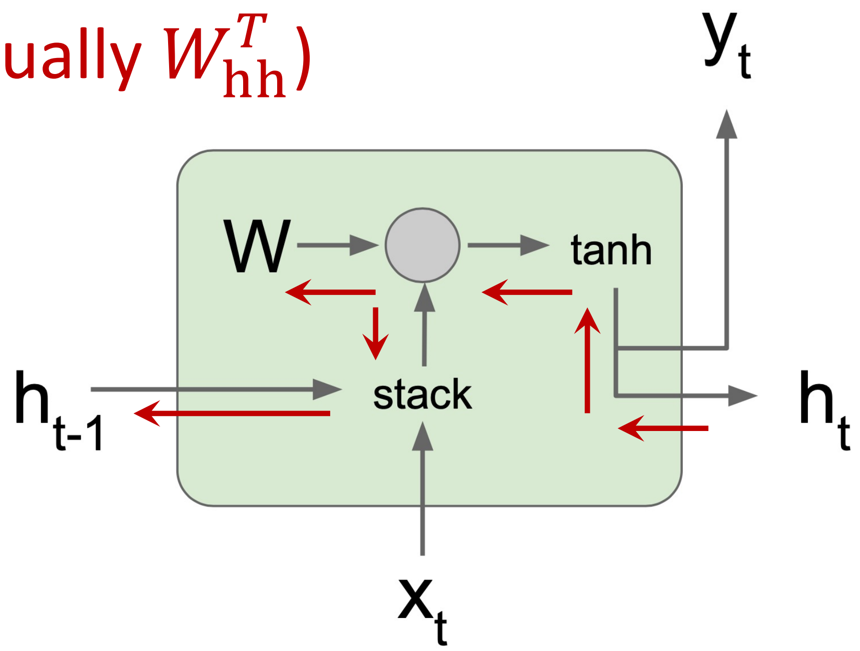
$$
\begin{aligned}
h_t &= \tanh(W_{hh}h_{t-1} + W_{xh}x_t) \\
&= \tanh\left(\begin{pmatrix} W_{hh} & W_{xh} \end{pmatrix}\begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right) \\
&= \tanh\left(W\begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)
\end{aligned}
$$

Bengio et al., “Learning long-term dependencies with gradient descent is difficult,” IEEE Trans. on NN, 1994

Pascanu et a., “On the difficulty of training recurrent neural networks,” ICML, 2013

Vanilla RNN Gradient Flow

Backpropagation from h_t to h_{t-1} multiplies by W (actually W_{hh}^T)



$$\mathrm{h}_t = \tanh(W_{\mathrm{hh}}\mathrm{h}_{t-1} + W_{\mathrm{xh}}\mathrm{x}_t)$$
$$= \tanh\left(\begin{pmatrix} W_{\mathrm{hh}} & W_{\mathrm{xh}} \end{pmatrix}\begin{pmatrix} \mathrm{h}_{t-1} \\ \mathrm{x}_t \end{pmatrix}\right)$$
$$= \tanh\left(W\begin{pmatrix} \mathrm{h}_{t-1} \\ \mathrm{x}_t \end{pmatrix}\right)$$

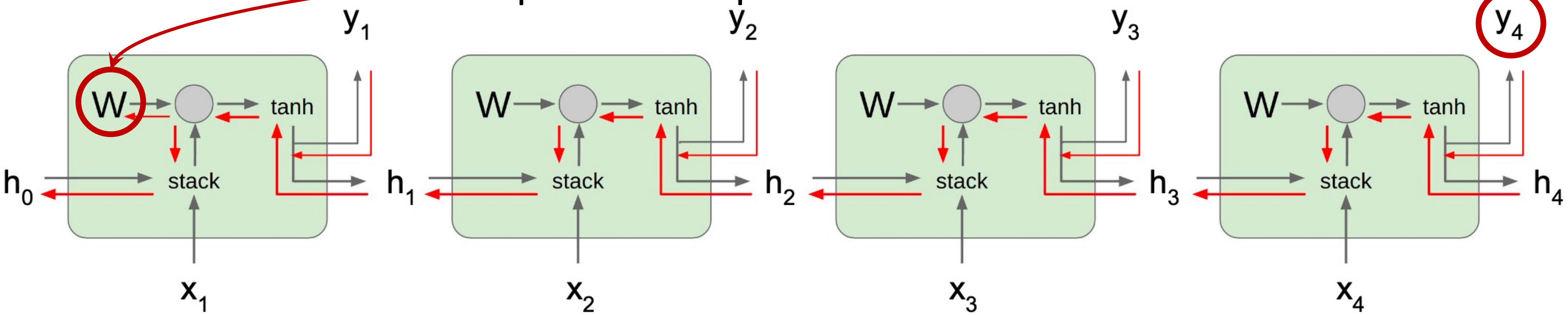
$$\frac{\partial \mathrm{h}_t}{\partial \mathrm{h}_{t-1}} = \tanh'(W_{\mathrm{hh}}\mathrm{h}_{t-1} + W_{\mathrm{xh}}\mathrm{x}_t)W_{\mathrm{hh}}$$

Bengio et al., "Learning long-term dependencies with gradient descent is difficult," IEEE Trans. on NN, 1994
Pascanu et a., "On the difficulty of training recurrent neural networks," ICML, 2013

[slide courtesy: Stanford, CS231]

Vanilla RNN Gradient Flow

- Gradients over multiple time steps:



$$\frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W}$$

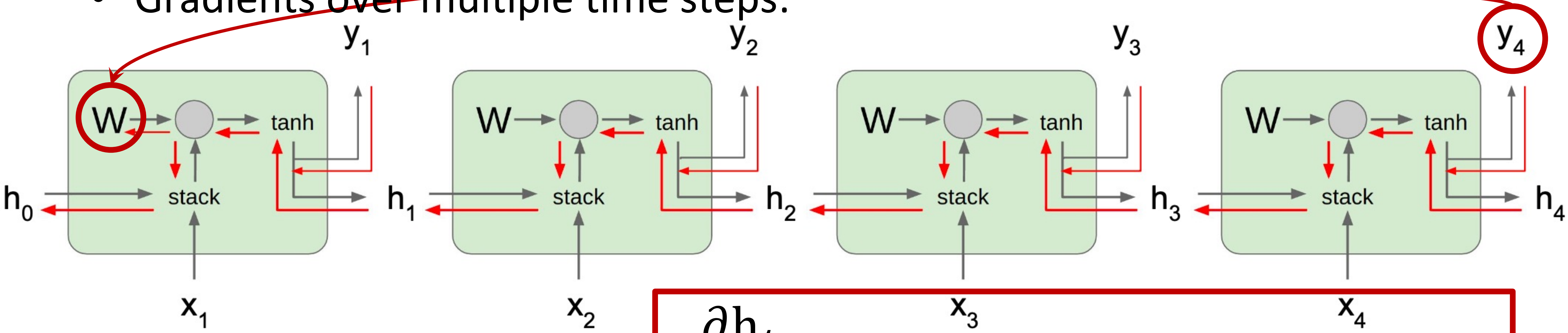
$$\frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial \mathrm{h}_t} \frac{\partial \mathrm{h}_t}{\partial \mathrm{h}_{t-1}} \cdots \frac{\partial \mathrm{h}_1}{\partial W} = \frac{\partial L_T}{\partial \mathrm{h}_t} \left(\prod_{t=2}^{T} \frac{\partial \mathrm{h}_t}{\partial \mathrm{h}_{t-1}} \right) \frac{\partial \mathrm{h}_1}{\partial W}$$

Bengio et al., “Learning long-term dependencies with gradient descent is difficult,” IEEE Trans. on NN, 1994
Pascanu et a., “On the difficulty of training recurrent neural networks,” ICML, 2013

[slide courtesy: Stanford, CS231]

Vanilla RNN Gradient Flow

- Gradients over multiple time steps:



$$\boxed{\frac{\partial \mathrm{h}_t}{\partial \mathrm{h}_{t-1}} = \tanh'(W_{\mathrm{hh}}\mathrm{h}_{t-1} + W_{\mathrm{xh}}\mathrm{x}_t)W_{\mathrm{hh}}}$$

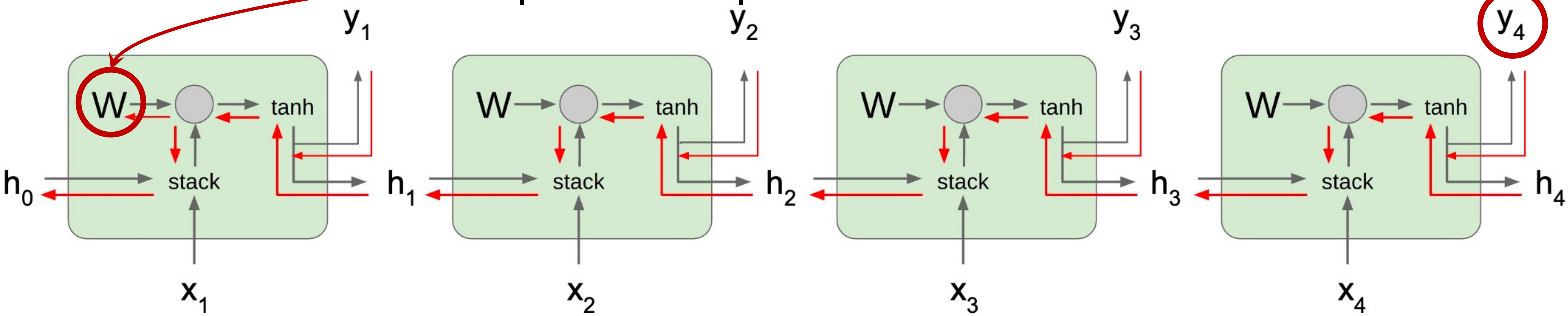
$$\frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W}$$

$$\frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial \mathrm{h}_t} \frac{\partial \mathrm{h}_t}{\partial \mathrm{h}_{t-1}} \cdots \frac{\partial \mathrm{h}_1}{\partial W} = \frac{\partial L_T}{\partial \mathrm{h}_t} \left(\prod_{t=2}^{T} \boxed{\frac{\partial \mathrm{h}_t}{\partial \mathrm{h}_{t-1}}} \right) \frac{\partial \mathrm{h}_1}{\partial W}$$

Bengio et al., “Learning long-term dependencies with gradient descent is difficult,” IEEE Trans. on NN, 1994
Pascanu et a., “On the difficulty of training recurrent neural networks,” ICML, 2013

Vanilla RNN Gradient Flow

- Gradients over multiple time steps:



$$\frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W}$$

Almost always < 1

Vanishing Gradients

$$\frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial \mathrm{h}_t}\left(\prod_{t=2}^{T} \boxed{\tanh'(W_{\mathrm{hh}}\mathrm{h}_{t-1} + W_{\mathrm{xh}}\mathrm{x}_t)} W_{\mathrm{hh}}\right)\frac{\partial \mathrm{h}_1}{\partial W}$$

Bengio et al., "Learning long-term dependencies with gradient descent is difficult," IEEE Trans. on NN, 1994
Pascanu et a., "On the difficulty of training recurrent neural networks," ICML, 2013

[slide courtesy: Stanford, CS231]

Long Short Term Memory (LSTM)

Vanilla RNN

$$\mathrm{h}_t = \tanh\left(W \begin{pmatrix} \mathrm{h}_{t-1} \\ \mathrm{x}_t \end{pmatrix} \right)$$

LSTM

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} \mathrm{h}_{t-1} \\ \mathrm{x}_t \end{pmatrix}$$

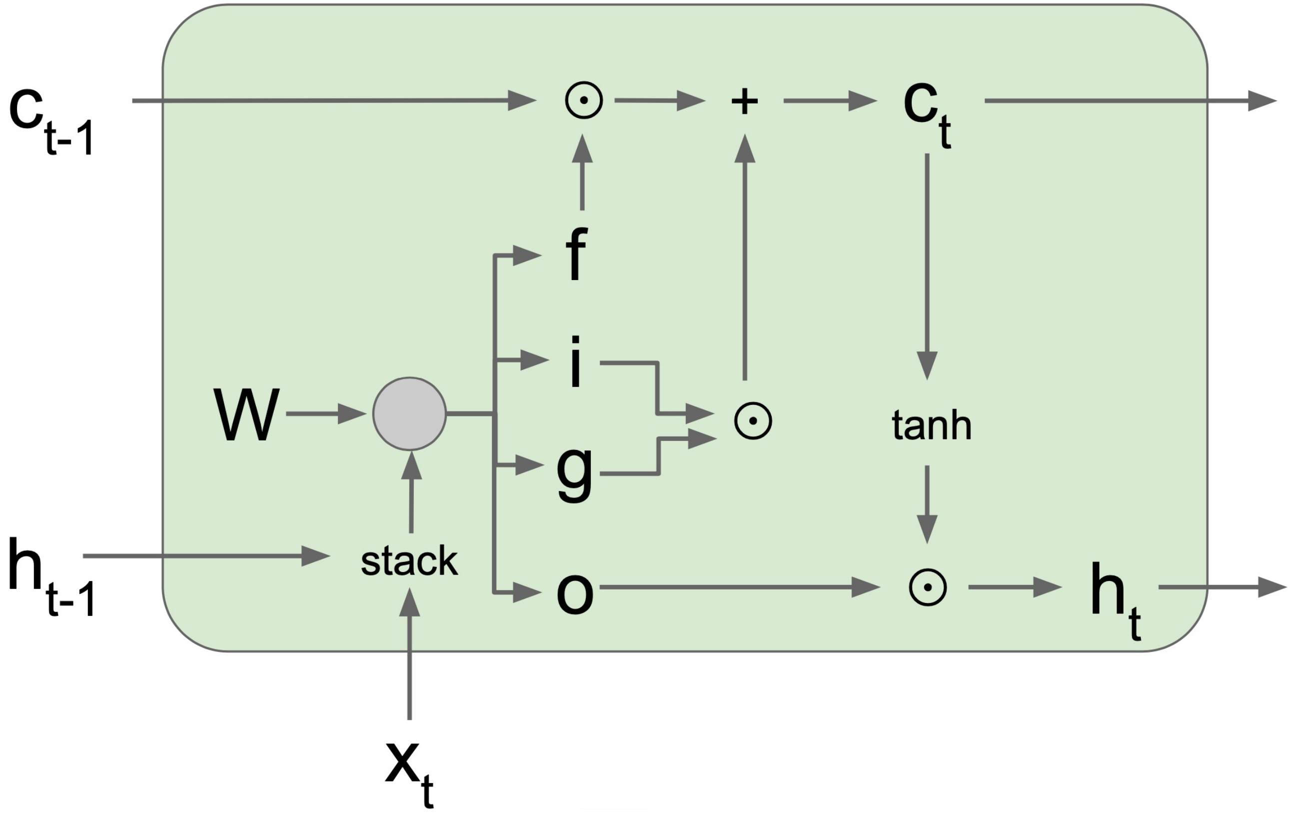
$$\mathrm{c}_t = f \odot \mathrm{c}_{t-1} + i \odot g$$

$$\mathrm{h}_t = o \odot \tanh(\mathrm{c}_t)$$

Hochreiter and Schmidhuber, “Long Short Term Memory,” Neural Computation, 1997

[slide courtesy: Stanford, CS231]

Long Short Term Memory (LSTM)



$\boldsymbol{i}$: <u>input gate</u>, whether to write to cell

$\boldsymbol{f}$: <u>forget gate</u>, whether to erase to cell

$\boldsymbol{o}$: <u>output gate</u>, how much to reveal cell

$\boldsymbol{g}$: <u>Gate gate</u>, how much to write to cell

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} \mathrm{h}_{t-1} \\ \mathrm{x}_t \end{pmatrix}$$

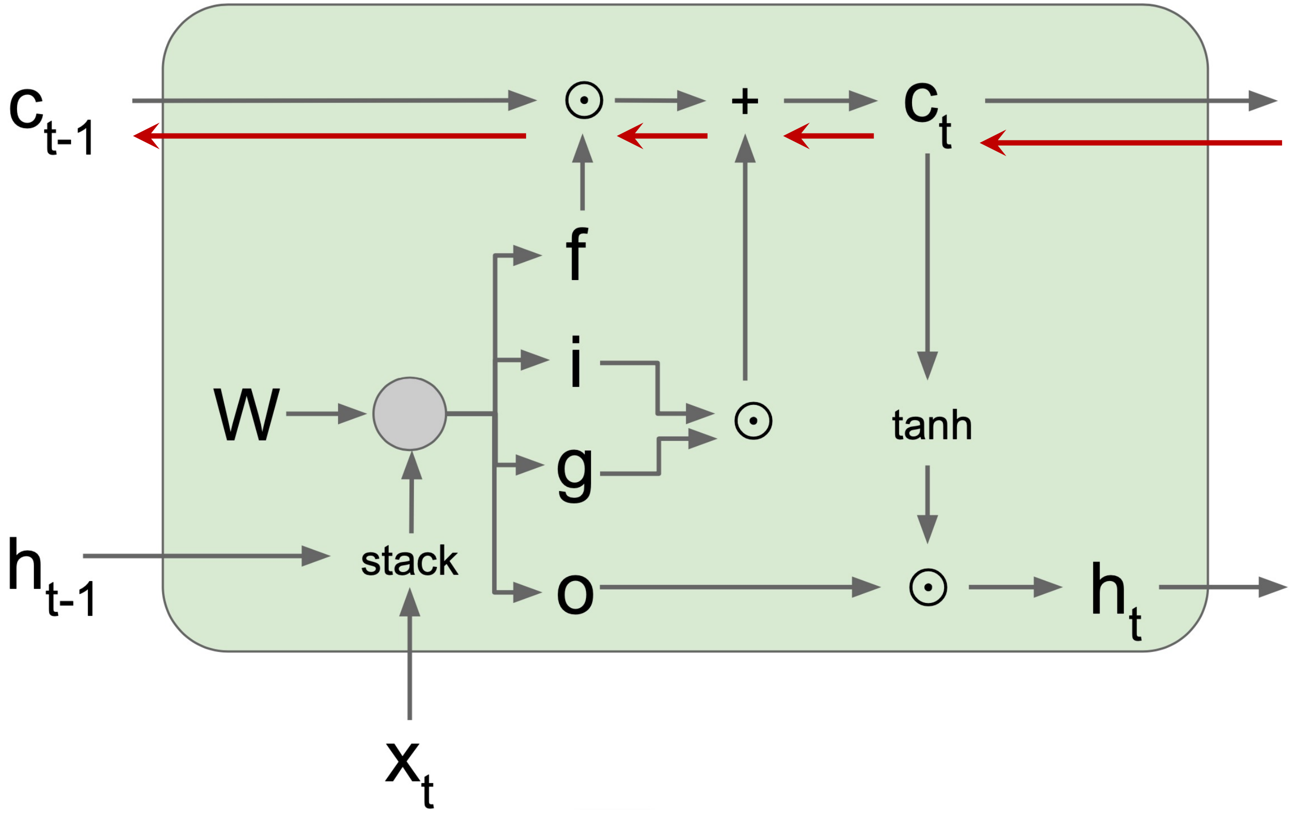
$$\mathrm{c}_t = f \odot \mathrm{c}_{t-1} + i \odot g$$

$$\mathrm{h}_t = o \odot \tanh(\mathrm{c}_t)$$

Hochreiter and Schmidhuber, "Long Short Term Memory," Neural Computation, 1997

[slide courtesy: Stanford, CS231]

LSTM Gradient Flow



Backpropagation from c_t to c_{t-1} only elementwise multiplication by f, no matrix multiply by W

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

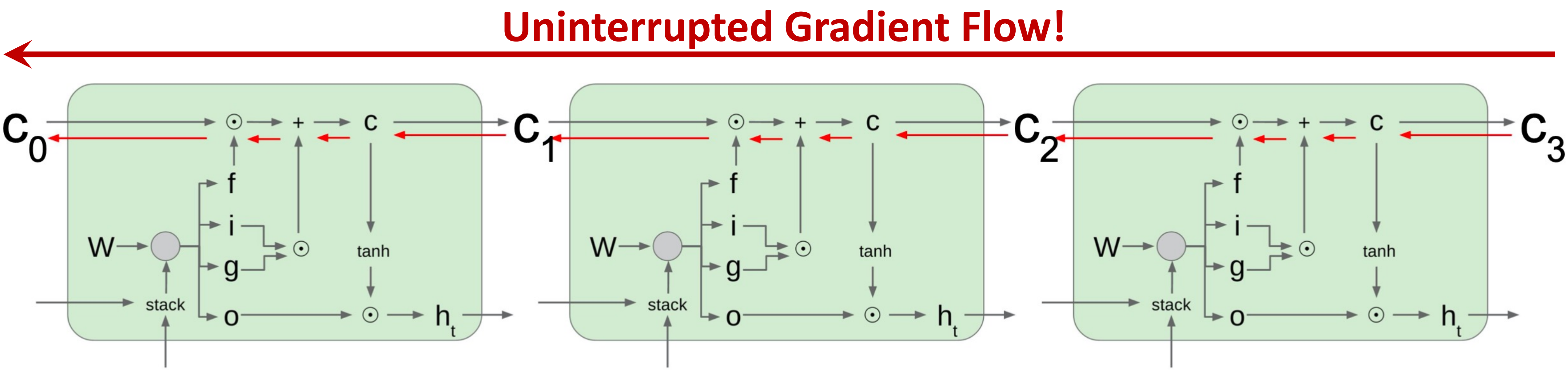
$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

Hochreiter and Schmidhuber, “Long Short Term Memory,” Neural Computation, 1997

[slide courtesy: Stanford, CS231]

LSTM Gradient Flow



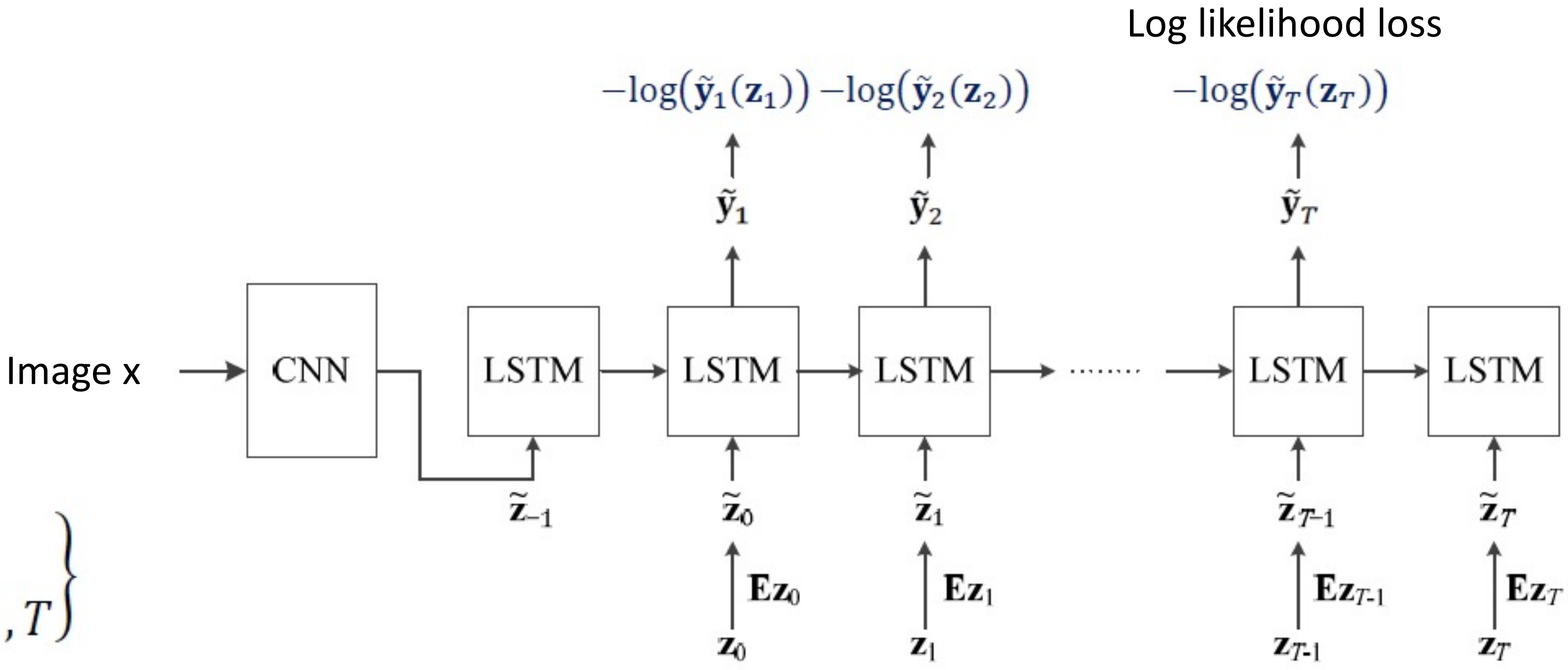
- Note that the gradient contains the $\boldsymbol{f}$ gate's vector of activations.
  - Allows better control of gradient values, using suitable parameter updates of the forget gate.
- Also notice that are added through the $\boldsymbol{f}$, $\boldsymbol{i}$, $\boldsymbol{g}$, and $\boldsymbol{o}$ gates.
  - Better balancing of gradient values.

Hochreiter and Schmidhuber, "Long Short Term Memory," Neural Computation, 1997

[slide courtesy: Stanford, CS231]

Application: Image Captioning

- **Goal:** Generate a sentence depicting what is happening in the image (e.g., recognizing objects, behaviors, interactions, and so on)



$$\left.\begin{aligned}\tilde{\mathbf{z}}_{-1} &= cnn(\mathbf{x}) \\ \tilde{\mathbf{z}}_t &= \mathbf{E}\mathbf{z}_t,\ \ t = 0,1,\cdots,T\end{aligned}\right\}$$

$(z_0, z_1, \ldots, z_T)$: ground truth sentence
E: transformation matrix for word embedding

image

conv-64
conv-64
maxpool

conv-128
conv-128
maxpool

conv-256
conv-256
maxpool

conv-512
conv-512
maxpool

conv-512
conv-512
maxpool

FC-4096
FC-4096
FC-1000
softmax

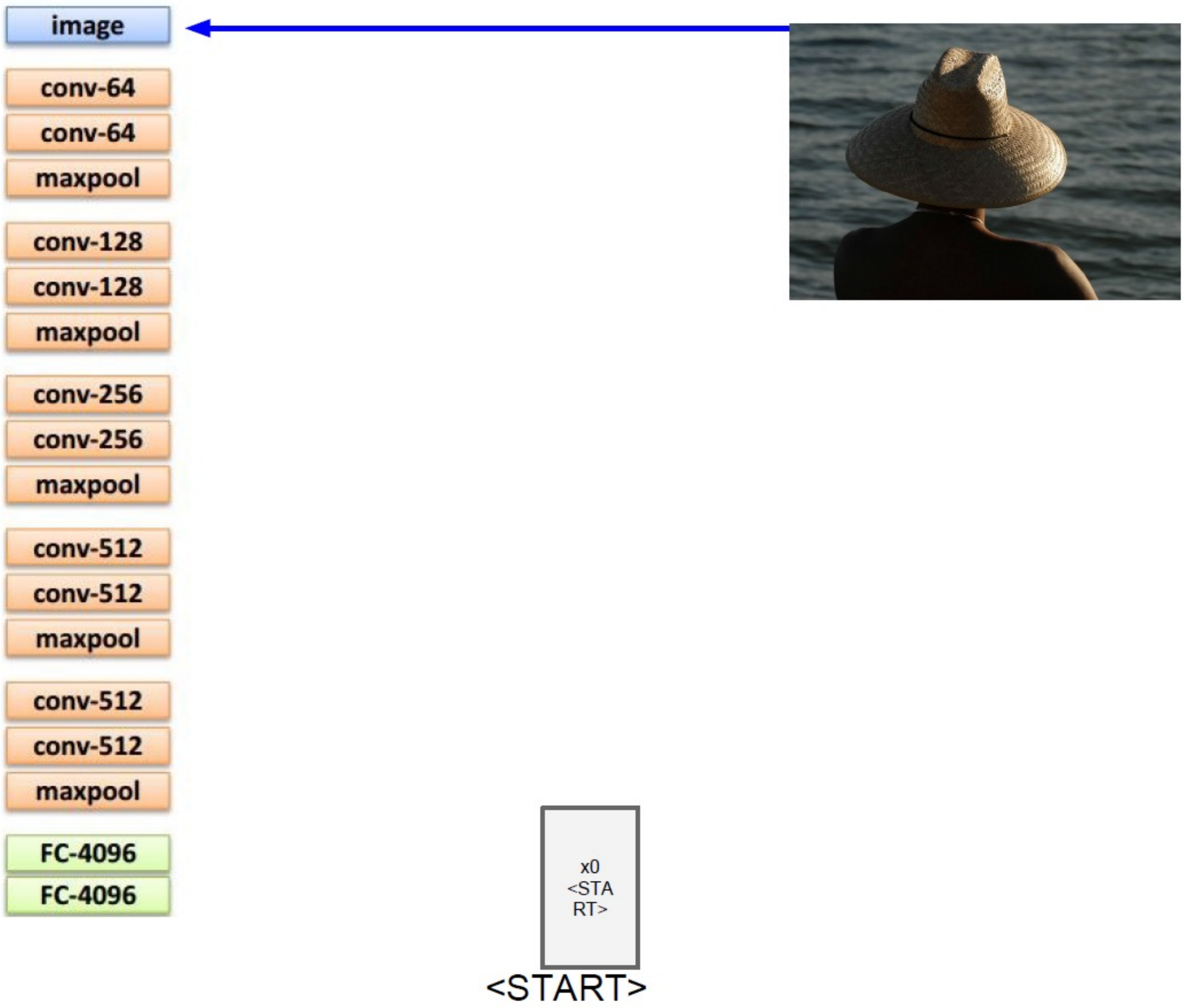
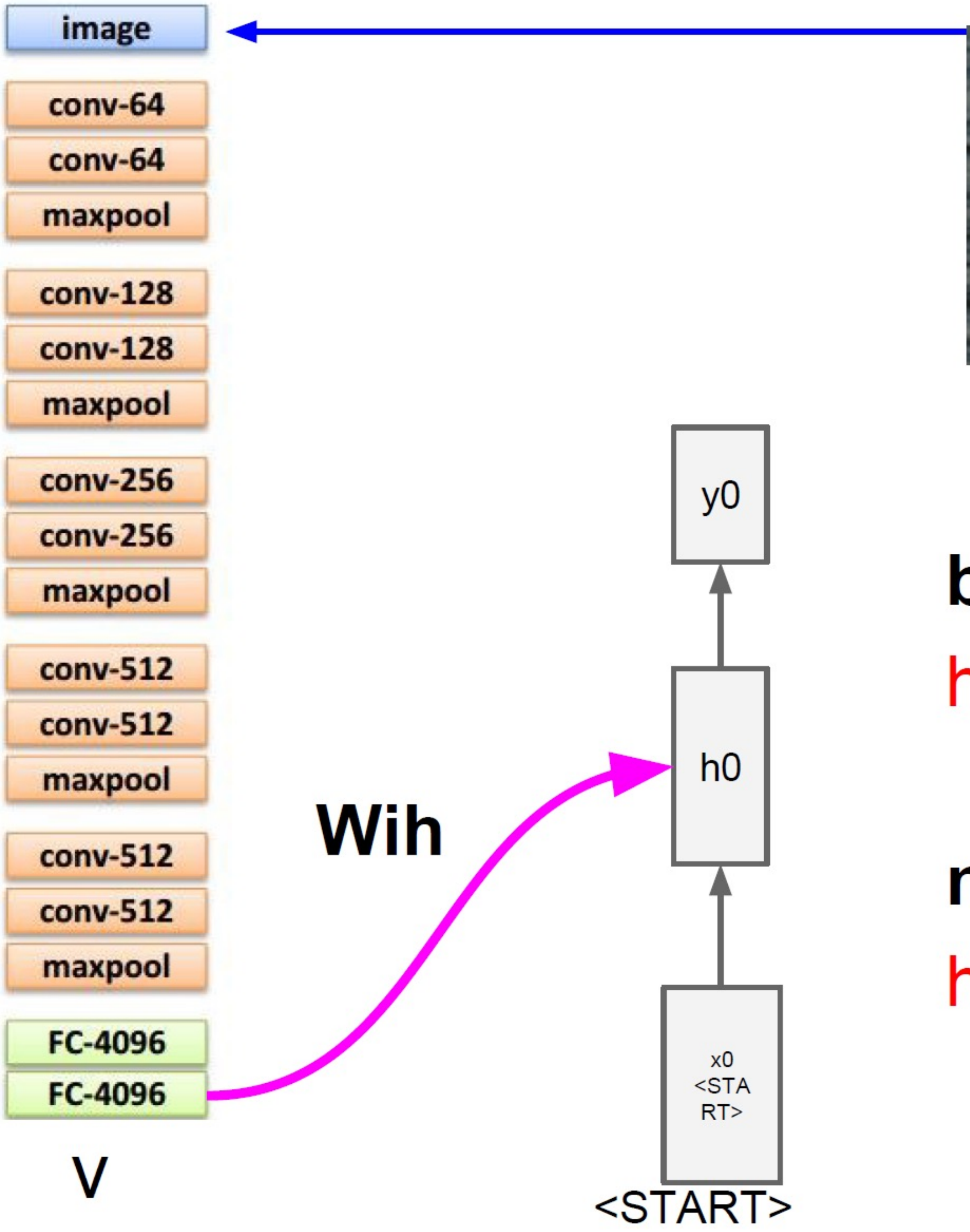


image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
conv-256
conv-256
maxpool
conv-512
conv-512
maxpool
conv-512
conv-512
maxpool
FC-4096
FC-4096
x0
<STA
RT>
<START>



test image

before:

h = tanh(Wxh * x + Whh * h)

now:

h = tanh(Wxh * x + Whh * h + **Wih * v**)

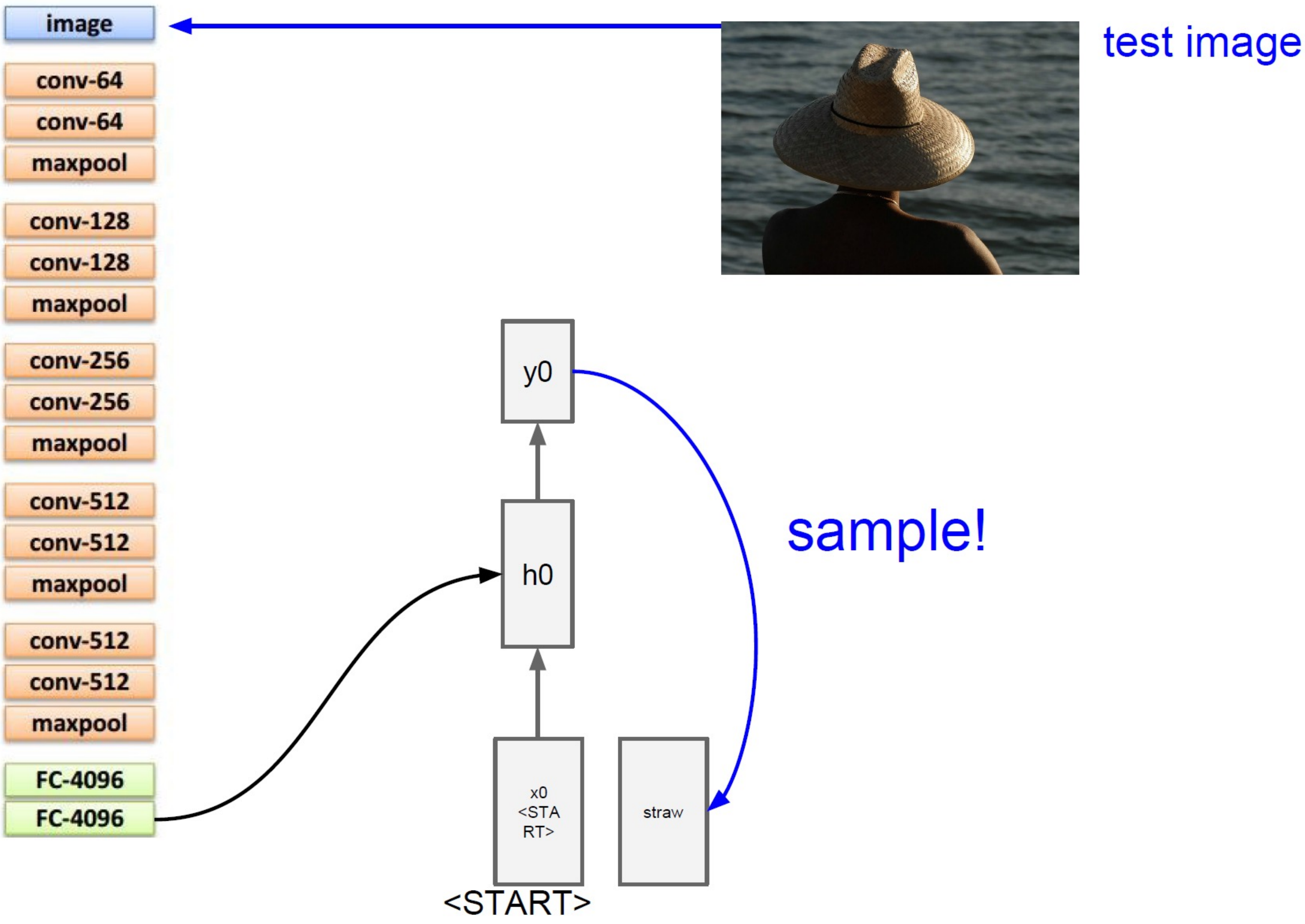


image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
conv-256
conv-256
maxpool
conv-512
conv-512
maxpool
conv-512
conv-512
maxpool
FC-4096
FC-4096
test image
y0
h0
sample!
x0
<STA
RT>
straw
<START>

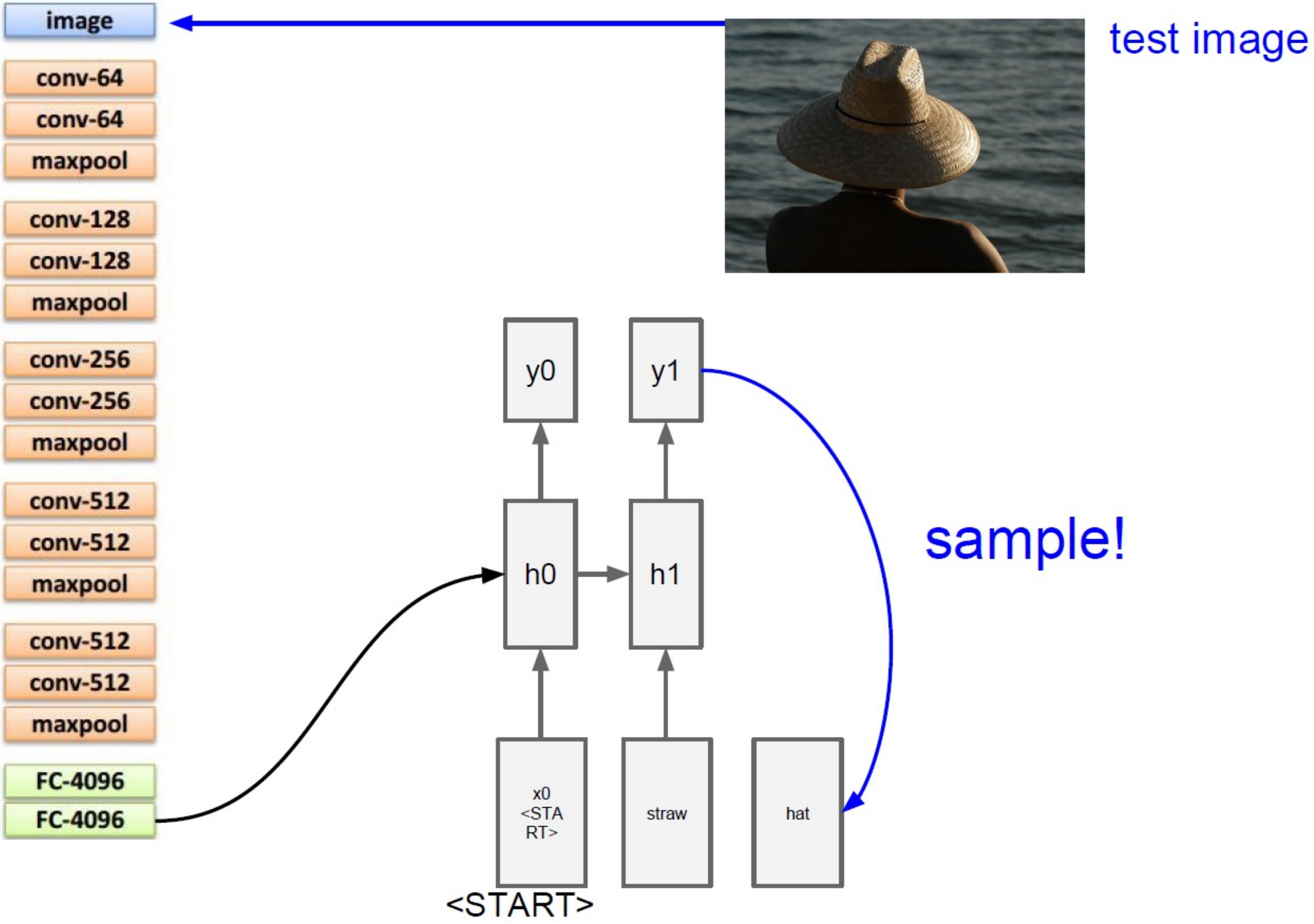
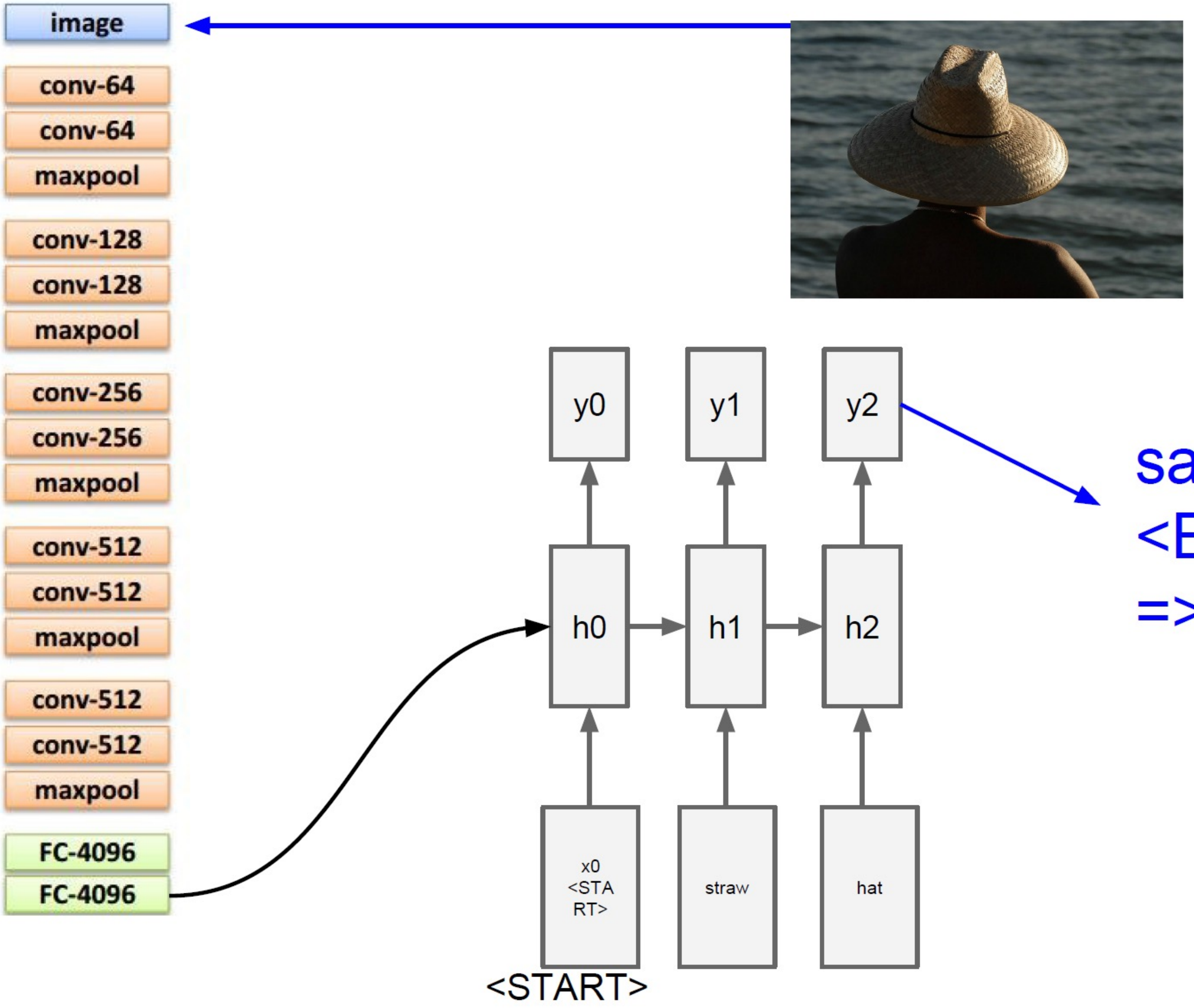


image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
conv-256
conv-256
maxpool
conv-512
conv-512
maxpool
conv-512
conv-512
maxpool
FC-4096
FC-4096
test image
y0
y1
h0
h1
sample!
x0 <START>
straw
hat
<START>



test image

sample <END> token => finish.

a man wearing a blue shirt with his arms on the grass.
a man holding a frisbee bat in front of a green field.
a man throwing a frisbee in a green field.
a boy playing ball with a disc in a field.
a young man playing in the grass with a green ball.

a red car on the side of the road in the small race.
a truck driving uphill on the side of the road.
a person driving a truck on the road.
a small car driving down a dirt and water.
a truck in a field of car is pulled up to the back.

a group of birds standing next to each other.
a group of ducks that are standing in a row.
a group of ducks that are standing on each other.
a group of sheep next to each other on sand.
a group of small birds is standing in the grass.

a kite flying over the ocean on a sunny day.
a person flying over the ocean on a sunny day.
a person flying over the ocean on a cloudy day.
a kite on the beach on the water in the sky.
a large flying over the water and rocks.

(Supp.) Other Methods for Sequences

- GRU
- Memory Networks

Cho et al., “Learning Phrase Representations Using RNN Encoder-Decoder for Statistical Machine Translation,” EMNLP, 2014
Weston et al., “Memory Networks,” ICLR, 201t5

Summary

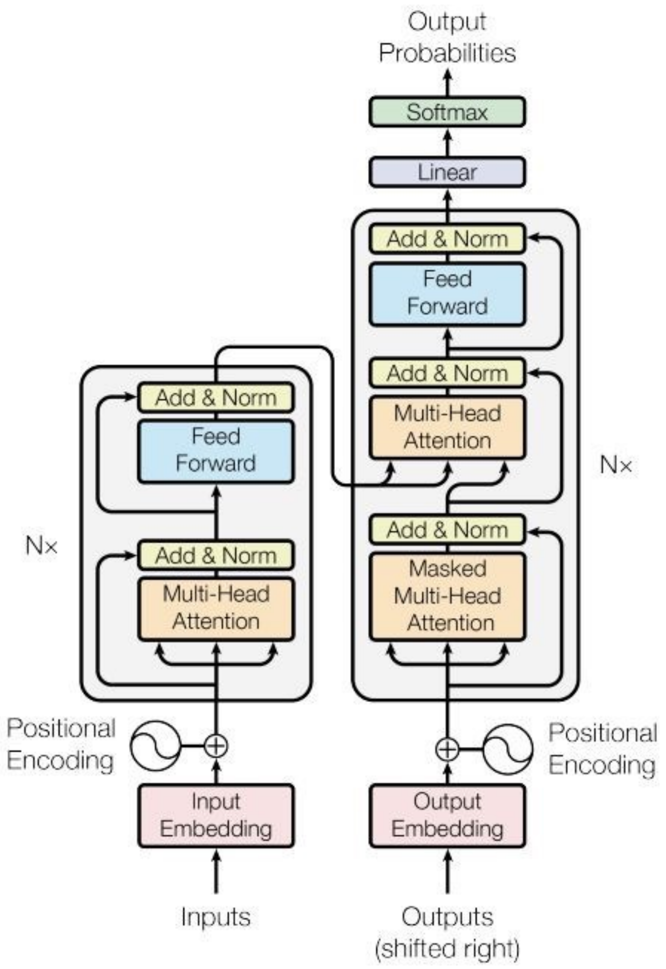
- RNNs allows a lot of flexibility in architecture design.
- Vanilla RNNs are simple but don’t work very well.
- Common to use LSTM or GRU: their additive interactions improve gradient flow.
- Backward flow of gradients in RNN can explode or vanish.
  Exploding is controlled with gradient clipping.
  Vanishing is controlled with additive interactions (LSTM).
- Better/simpler architectures are a hot topic of current research.
- Better understanding (both theoretical and empirical) is needed.

New Paradigms for Reasoning Over Sequences

- Vaswani et al., “Attention is all you need,” NeurIPS, 2017.

- New “Transformer” architecture no longer processes inputs sequentially; instead it can operate over inputs in a sequence in parallel through an **attention** mechanism.

- Has led to many state-of-the-art results and pre-training in NLP, for more interest see e.g.,
  - Devlin et al., “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding,” 2018

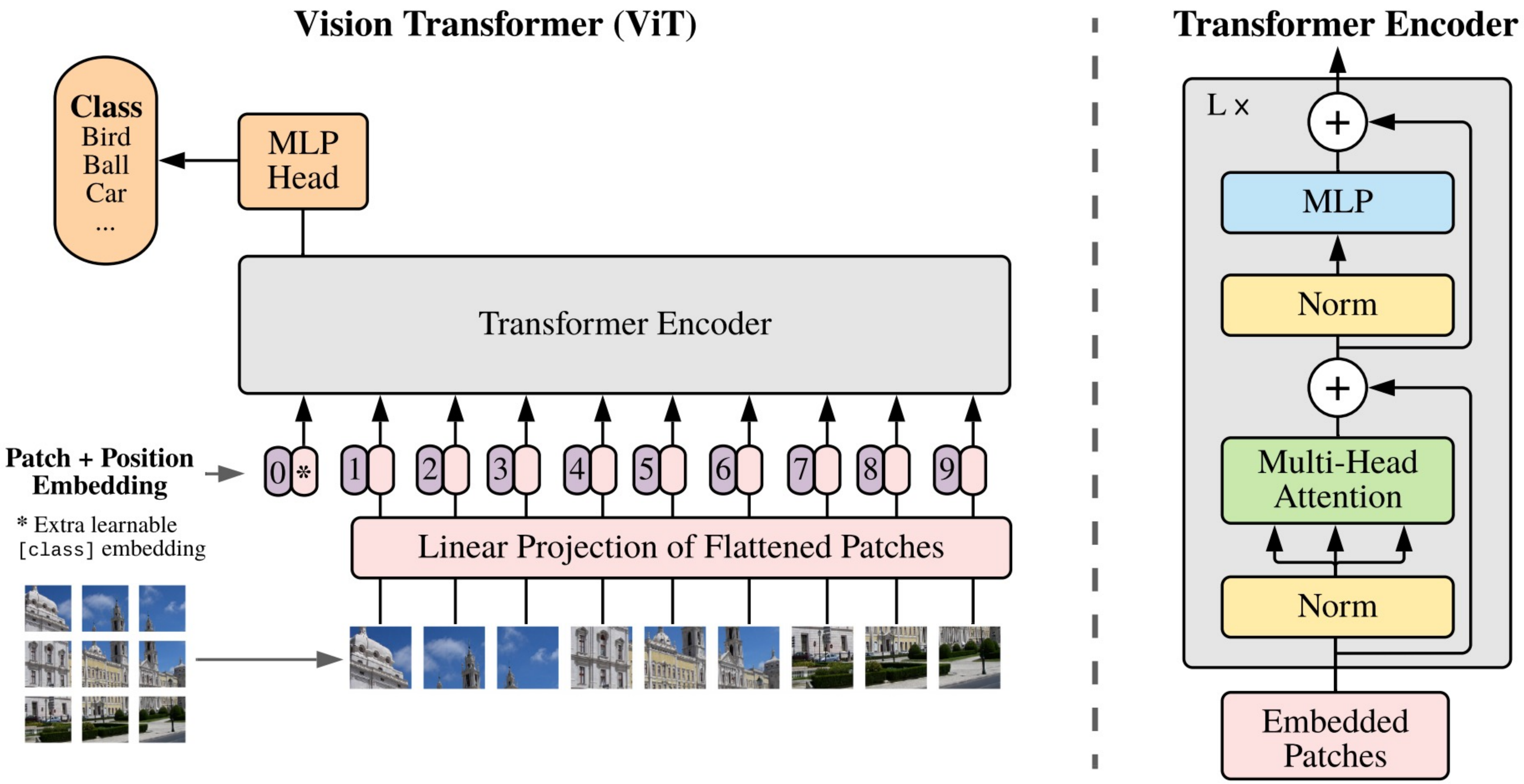


And now…

- Dosovitskiy et al., "An Image is Worth 16 x 16 words: Transformers for Image Recognition at Scale," ICLR, 2021.

- *This paper shows that the reliance on CNNs in computer vision is **not** necessary and a pure structure in place.*

And now…


Vision Transformer (ViT)
Class
Bird
Ball
Car
...
MLP Head
Transformer Encoder
Patch + Position Embedding
* Extra learnable
[class] embedding
0
*
1
2
3
4
5
6
7
8
9
Linear Projection of Flattened Patches
Transformer Encoder
L ×
+
MLP
Norm
+
Multi-Head Attention
Norm
Embedded Patches

Thank you!

Q & A