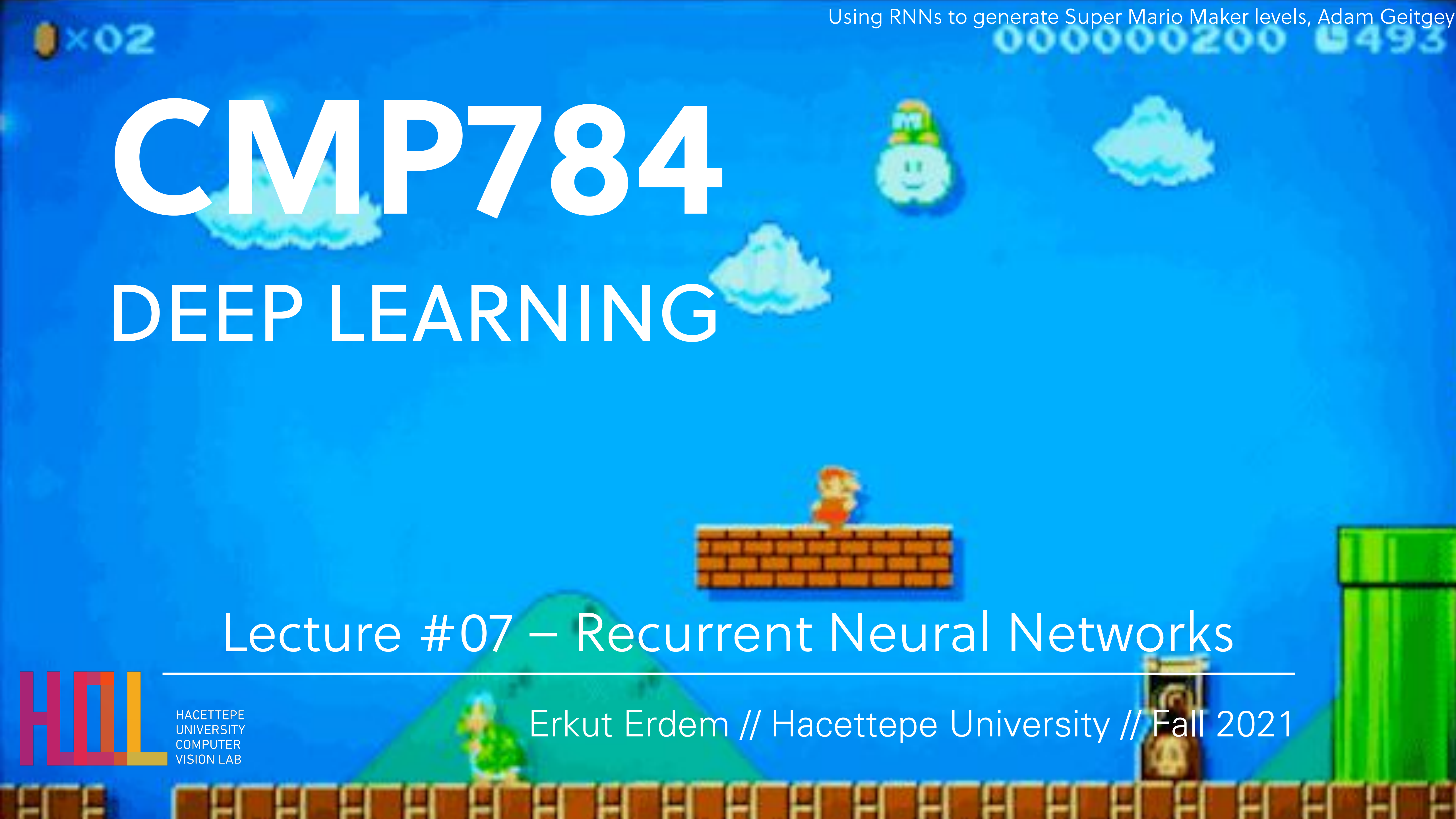


Using RNNs to generate Super Mario Maker levels, Adam Geitgey

CMP784

DEEP LEARNING

Lecture #07 – Recurrent Neural Networks

HACETTEPE UNIVERSITY COMPUTER VISION LAB

Erkut Erdem // Hacettepe University // Fall 2021

Previously on CMP784

- more on transfer learning
- interpretability
- visualizing neuron activations
- visualizing class activations
- pre-images
- adversarial examples
- adversarial training


Image from ucreative.com

Lecture overview

- Sequence modeling
- Recurrent Neural Networks (RNNs)
- The Vanilla RNN unit
- How to train RNNs
- The Long Short-Term Memory (LSTM) unit and its variants
- Gated Recurrent Unit (GRU)

Disclaimer: Much of the material and slides for this lecture were borrowed from

—Harini Suresh's MIT 6.S191 slides

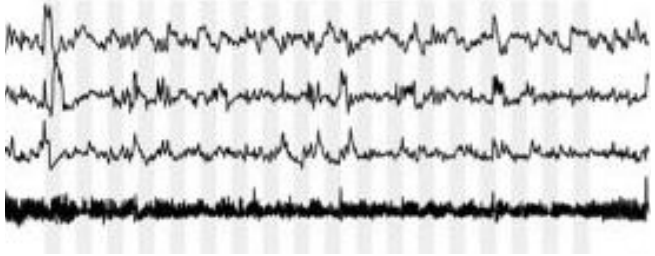
—Arun Mallya's tutorial on Recurrent Neural Networks

—Phil Blunsom's Oxford Deep NLP class

—Fei-Fei Li, Andrej Karpathy and Justin Johnson's CS231n class

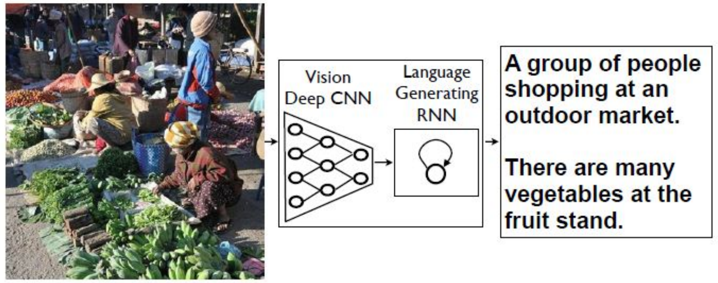
Sequence modeling

Sequential data

- "I took the dog for a walk this morning." sentence
-  medical signals
-  speech waveform
-  video frames

Modeling sequential data

- Sample data sequences from a certain distribution $P(x_1, \ldots, x_N)$
- Generate natural sentences to describe an image $P(y_1, \ldots, y_M | I)$



- Activity recognition from a video sequence $P(y | x_1, \ldots, x_N)$

- [ ] Running
- [x] Jumping
- [ ] Dancing
- [ ] Fighting
- [ ] Eating

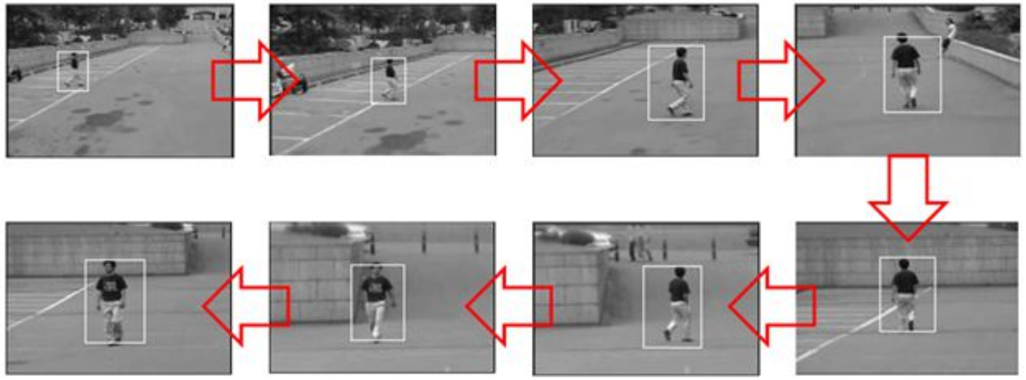
Modeling sequential data

- Speech recognition

$$P(y_1, \ldots, y_N | x_1, \ldots, x_N)$$

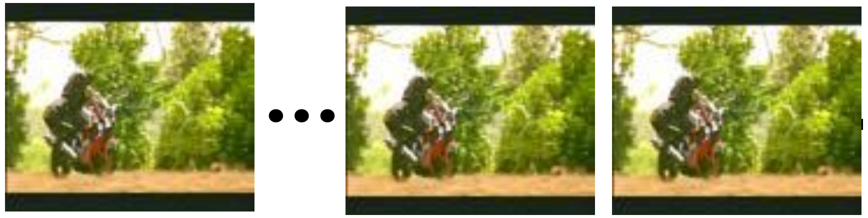
- Object tracking

$$P(y_1, \ldots, y_N | x_1, \ldots, x_N)$$



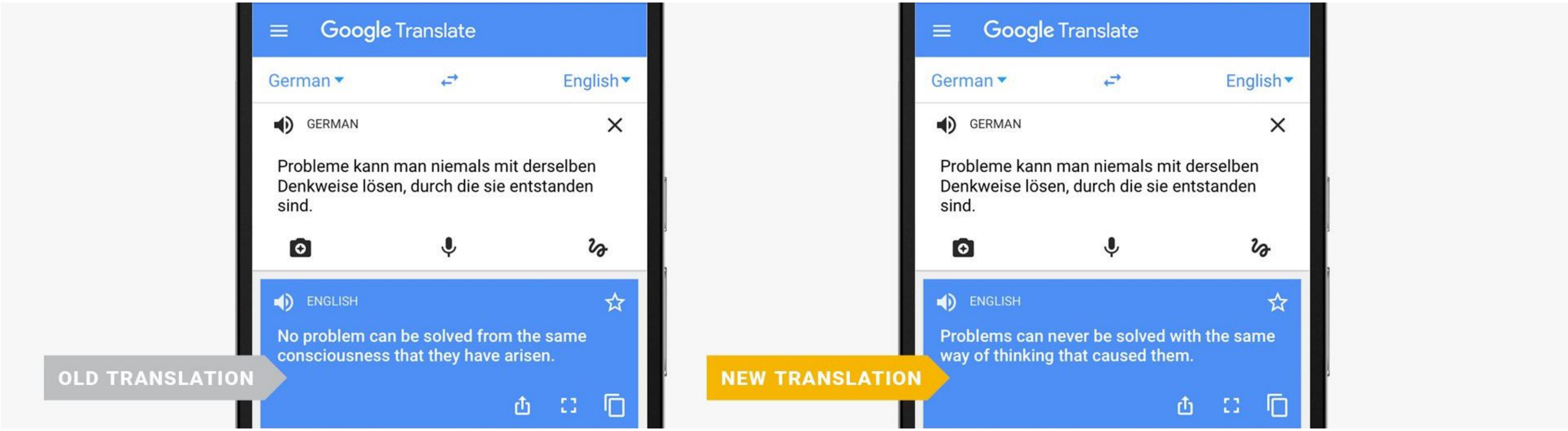
Modeling sequential data

- Generate natural sentences to describe a video $P(y_1, \ldots, y_M | x_1, \ldots, x_N)$



$\longrightarrow$ **A man is riding a bike**

- Machine translation $P(y_1, \ldots, y_M | x_1, \ldots, x_N)$



Represent a sequence as a *bag of words*

"I dislike rain."

[0 1 0 1 0 0 0 1]

prediction

- **Problem:** Bag of words does not preserve order

Bag of words does not preserve order!

"The food was good, not bad at all."

vs

"The food was bad, not good at all."

Maintain an ordering within feature vector

[0 0 0 1 0 0 0 1 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 1]

On Monday it was snowing

One hot feature vector indicates what each word is

prediction

- **Problem:** Hard to deal with different word orders!

Hard to deal with different word orders!

"On Monday, it was snowing."

vs

"It was snowing on Monday."

Hard to deal with different word orders!

[0 0 0 1 0 0 0 1 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 1]

On Monday it was snowing

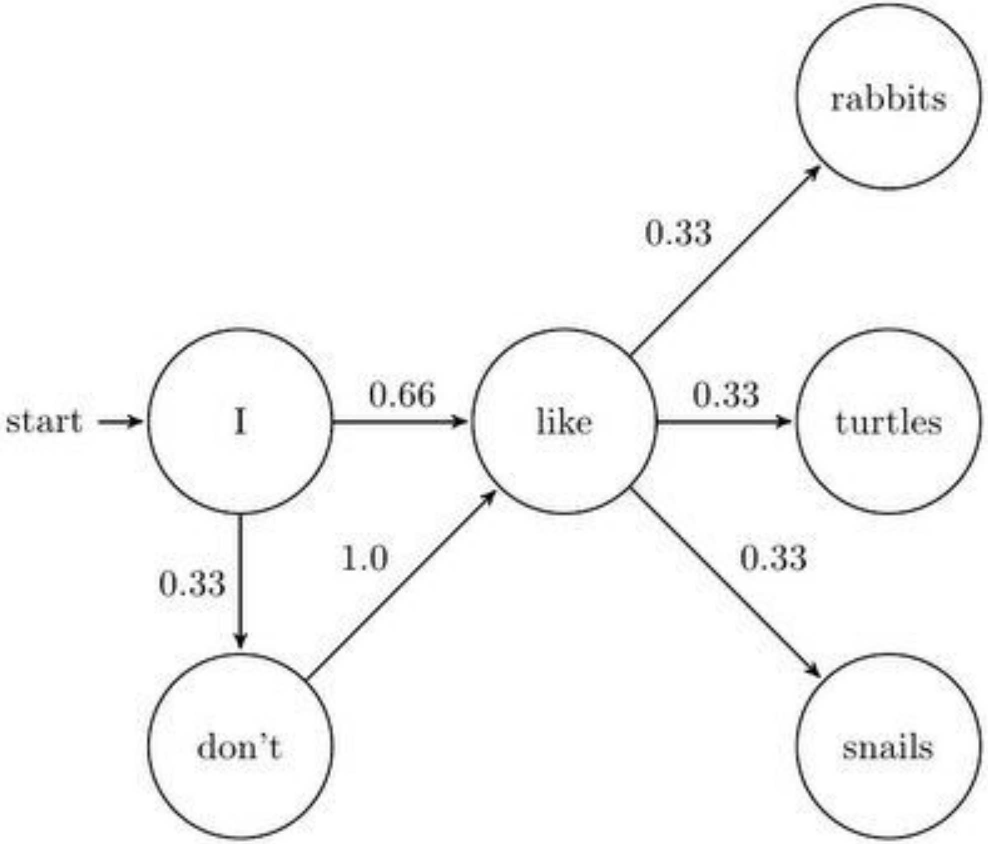
vs

[1 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0]

It was snowing on Monday

- we would have to relearn the rules of language at each point in the sentence

Markov Models



- **Problem:** we can't model long-term dependencies

Markov Models

- **Markov assumption:** Each state depends only on the last state.

"In France, I had a great time and I learnt some of the _____ language."

- We need information from the far past and future to accurately guess the correct word.

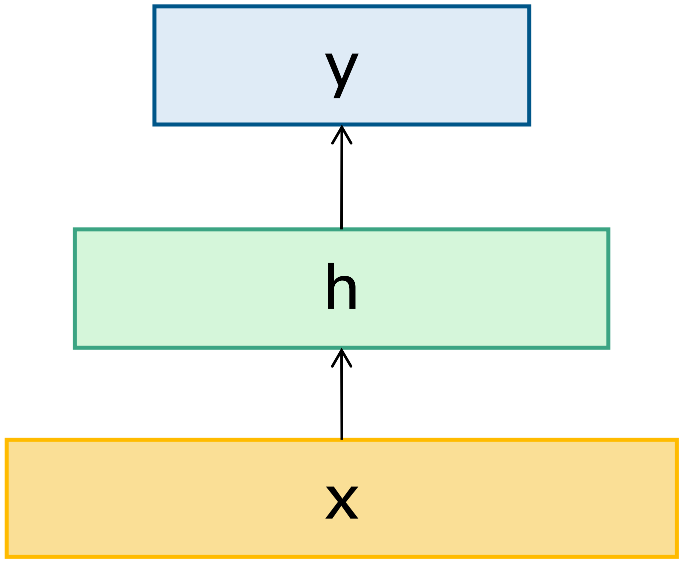
To model sequences, we need

1. to deal with variable length sequences
2. to maintain sequence order
3. to keep track of long-term dependencies
4. to share parameters across the sequence

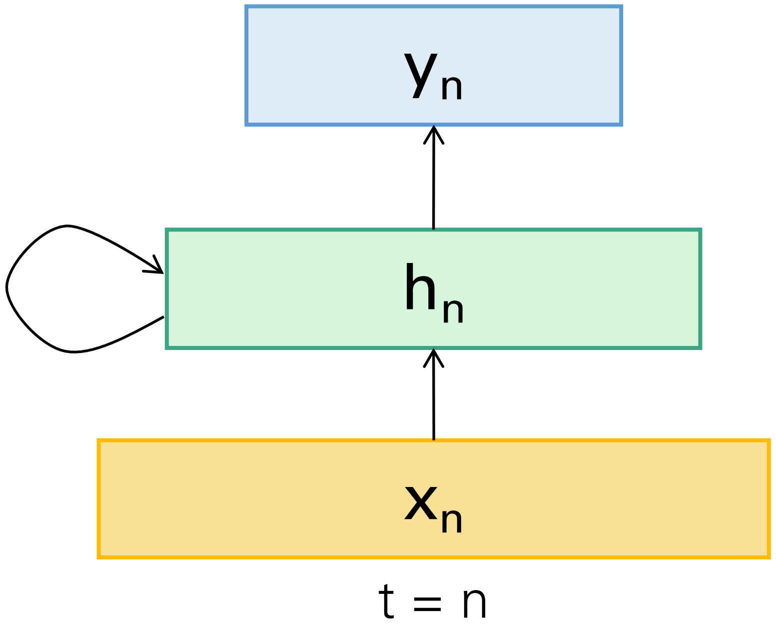
Recurrent Neural Networks

Recurrent Neural Networks

Feed Forward Network

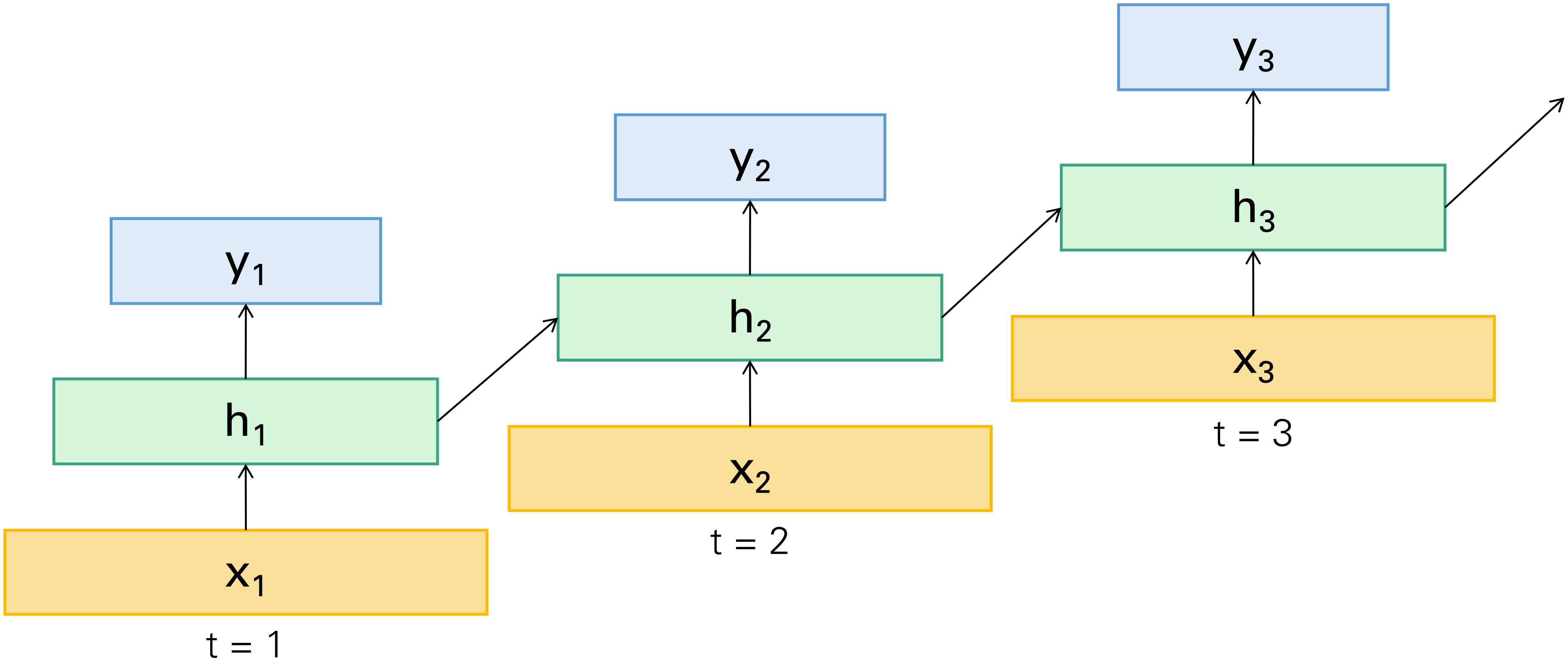


Recurrent Network

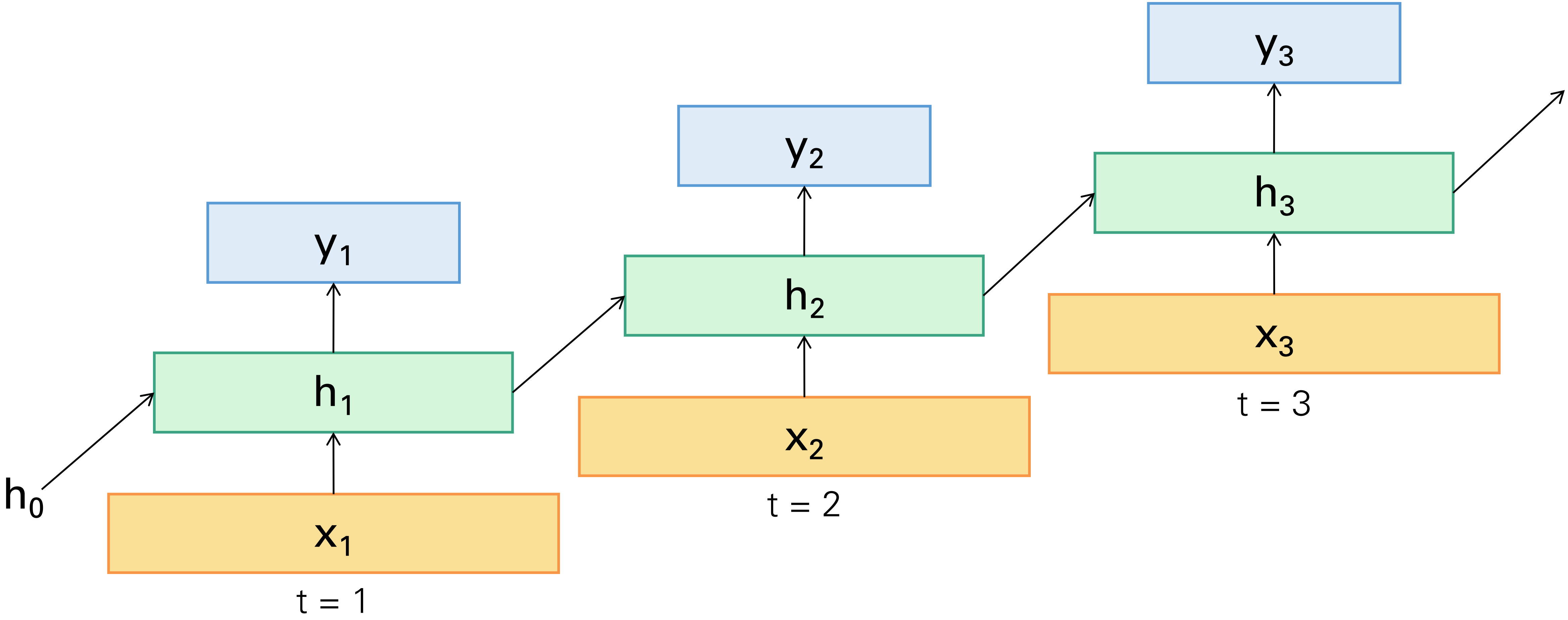


Notice: the same function and the same set of parameters are used at every time step.

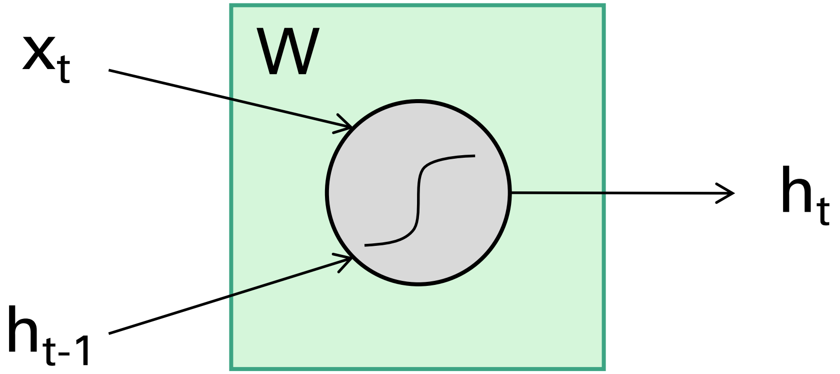
Unrolled RNN



Sample RNN



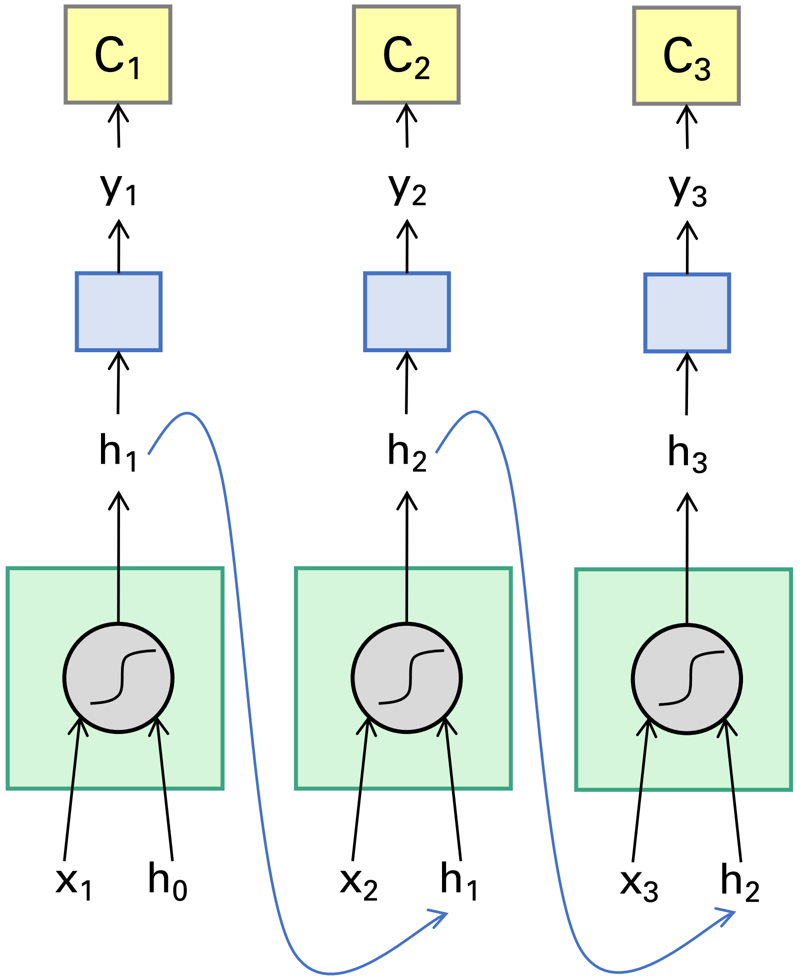
The Vanilla RNN Cell



$$h_t = \tanh W \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix}$$

cell state

The Vanilla RNN Forward

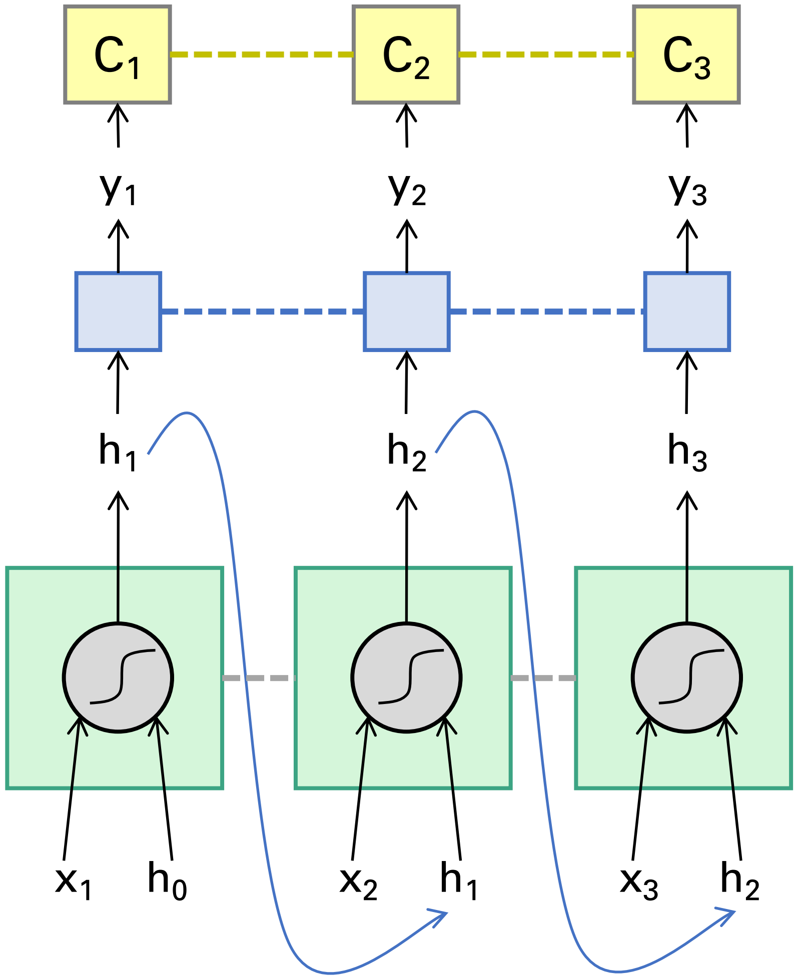


$$h_t = \tanh W \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix}$$

$$y_t = \mathrm{F}(h_t)$$

$$C_t = \mathrm{Loss}(y_t, \mathrm{GT}_t)$$

The Vanilla RNN Forward



$$h_t = \tanh W \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix}$$

$$y_t = \mathrm{F}(h_t)$$

$$C_t = \mathrm{Loss}(y_t, \mathrm{GT}_t)$$

------ indicates shared weights

- Note that the weights are shared over time
- Essentially, copies of the RNN cell are made over time (unrolling/unfolding), with different inputs at different time steps

Sentiment Classification

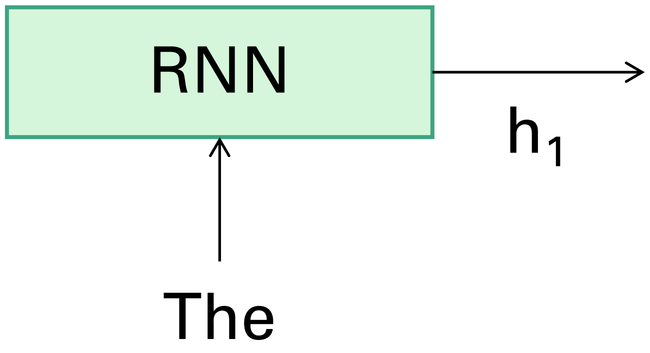
Sentiment Classification

- Classify a
  restaurant review from Yelp! OR
  movie review from IMDB OR
  …
  as positive or negative

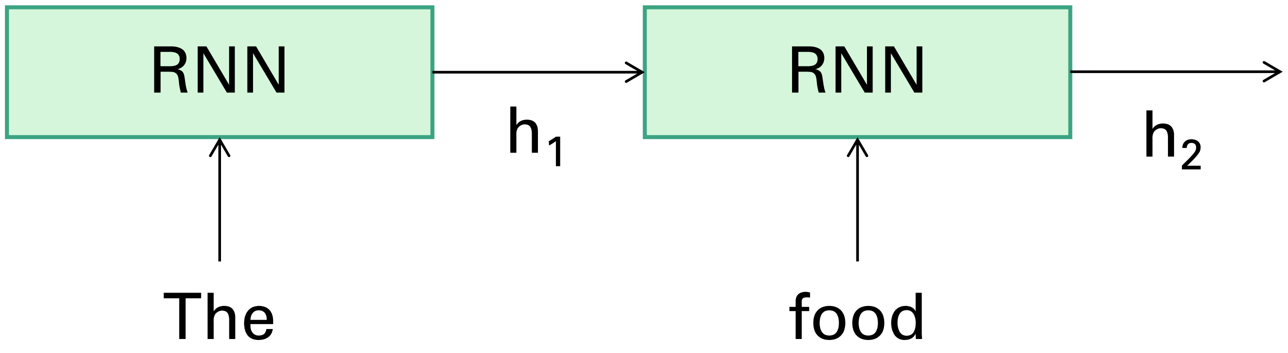
- **Inputs:** Multiple words, one or more sentences
- **Outputs:** Positive / Negative classification

- "The food was really good"
- "The chicken crossed the road because it was uncooked"

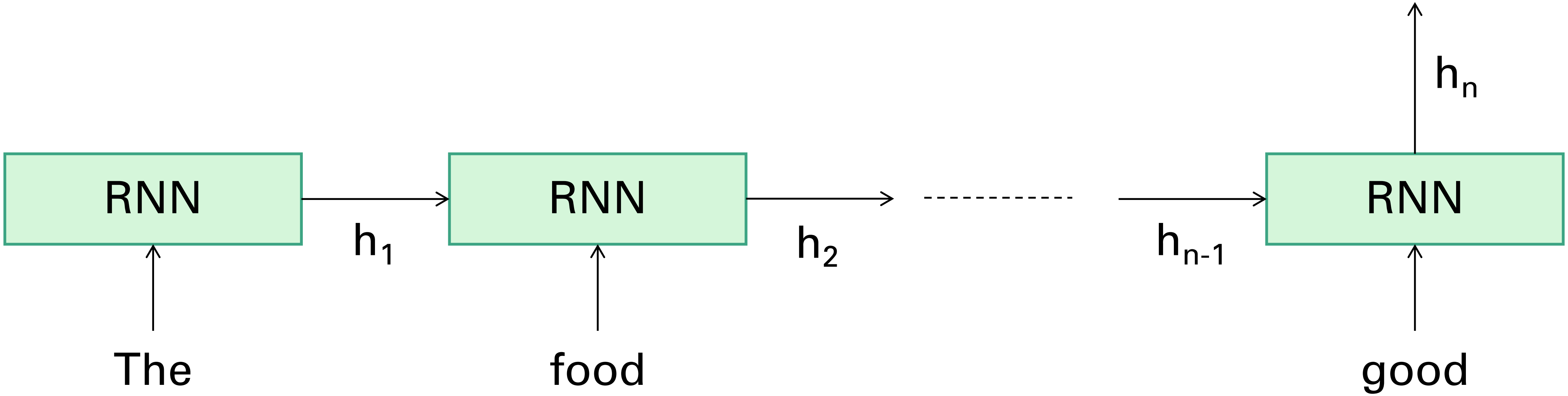
Sentiment Classification



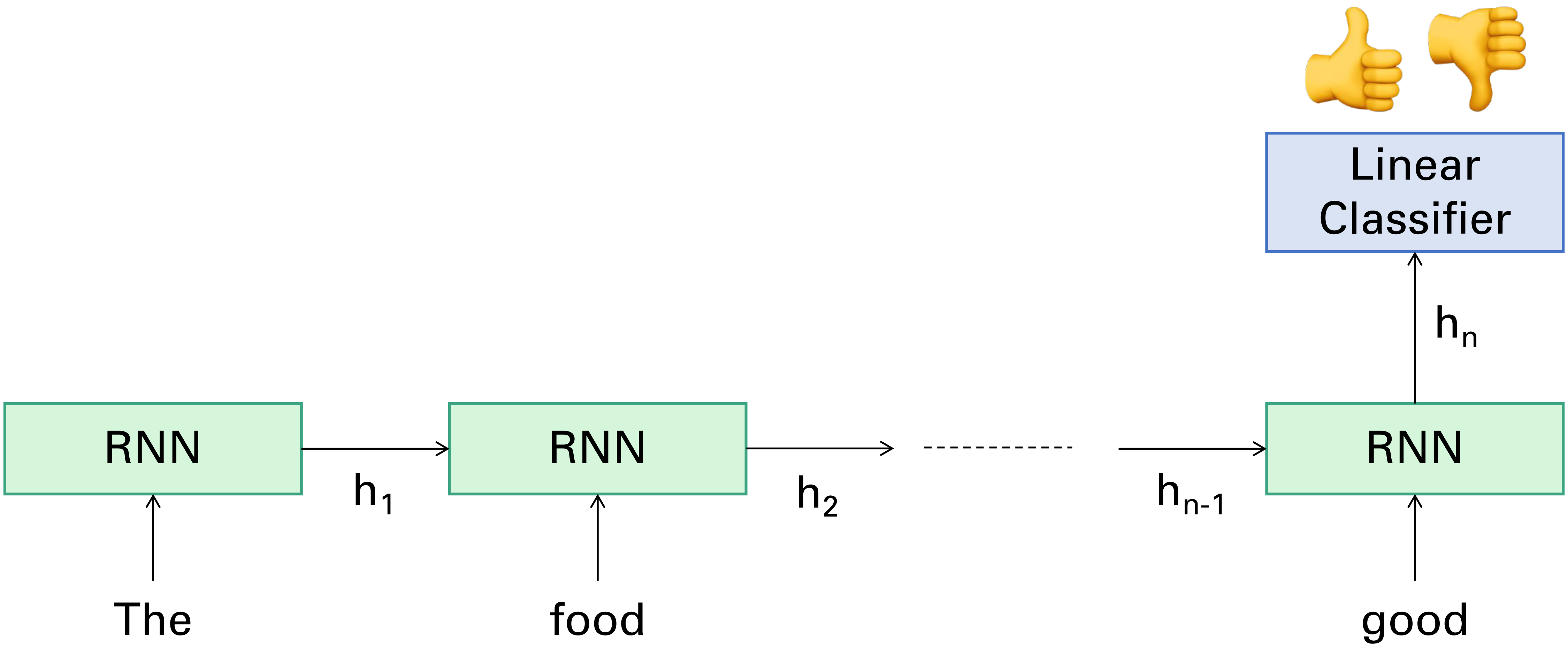
Sentiment Classification



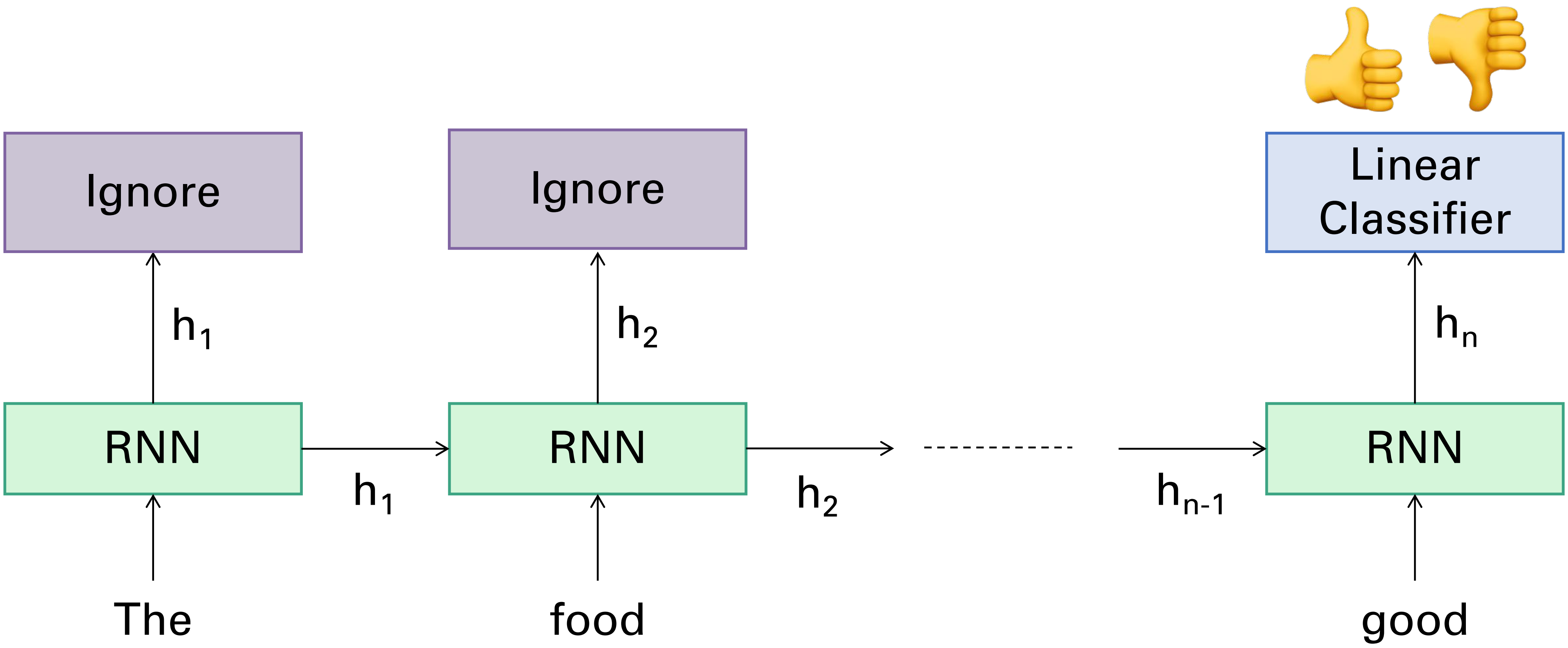
Sentiment Classification



Sentiment Classification



Sentiment Classification



Language Modeling

Language Modeling

- Language models aim to represent the history of observed text $(w_1,\ldots,w_{t-1})$ succinctly in order to predict the next word (w_t):

all the works of shakespeare → *language model* →

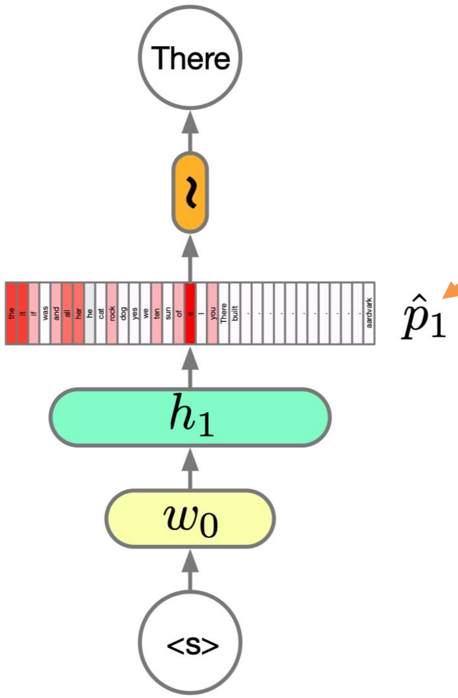
KING LEAR:
O, if you were a feeble sight,
the courtesy of your law,
Your sight and several breath,
will wear the gods
With his heads, and my hands
are wonder'd at the deeds,
So drop upon your lordship's
head, and your opinion
Shall be against your honour.

http://karpathy.github.io/2015/05/21/rnn-effectiveness/

RNN Language Models

$$h_n = g(V\,[x_n; h_{n-1}] + c)$$

$$\hat{y}_n = W h_n + b$$

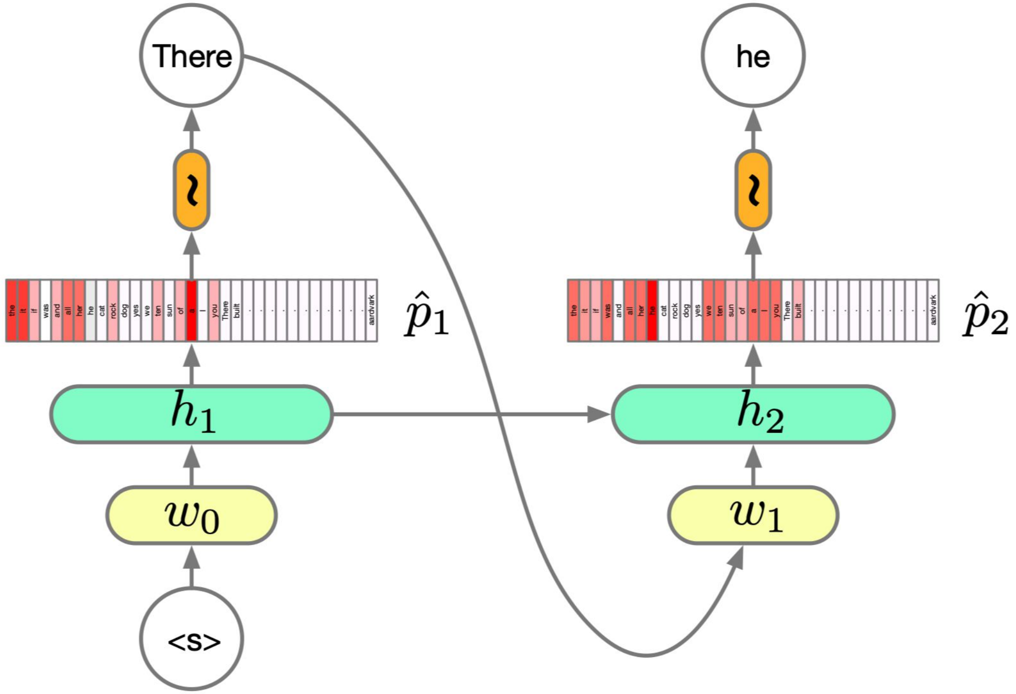


a probability distribution over possible next words, aka a softmax

RNN Language Models

$$h_n = g(V\,[x_n; h_{n-1}] + c)$$

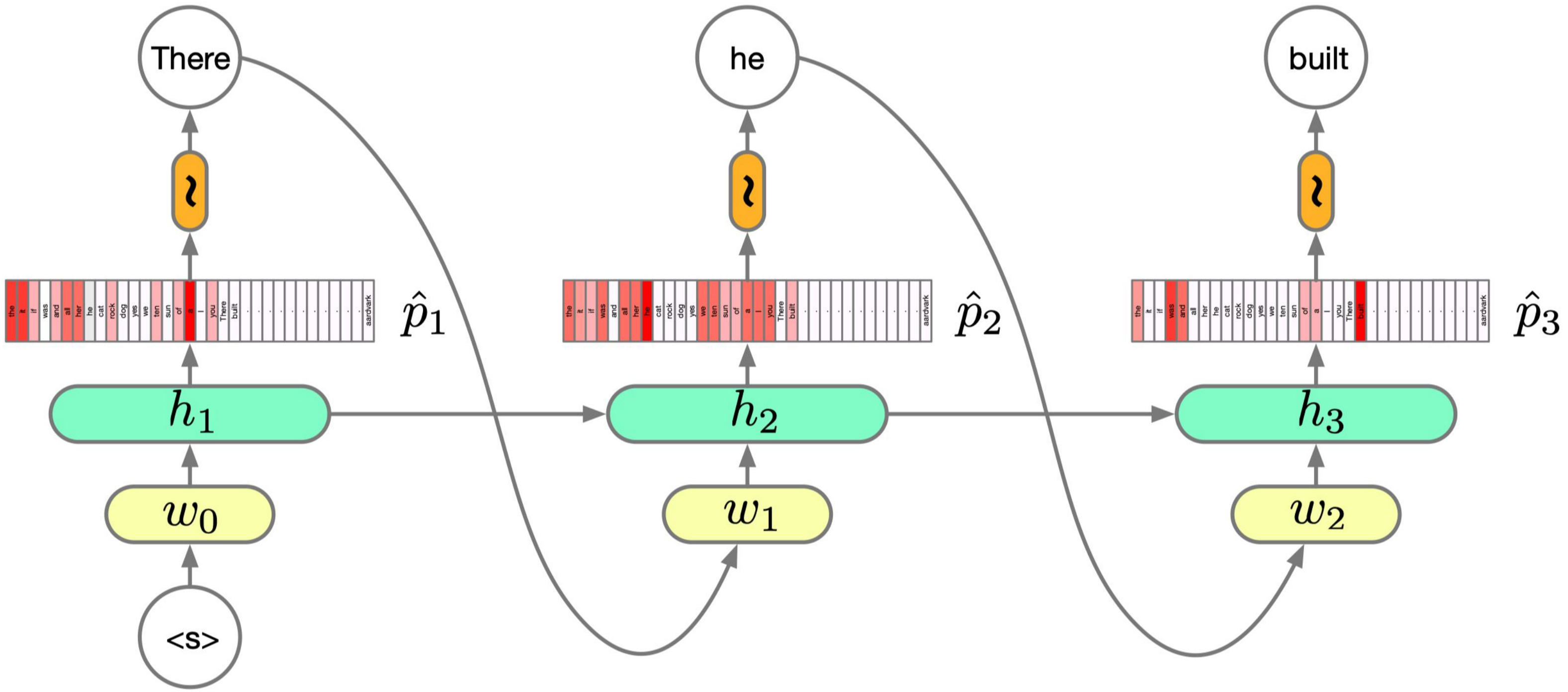
$$\hat{y}_n = W h_n + b$$



RNN Language Models

$$h_n = g(V\,[x_n; h_{n-1}] + c)$$

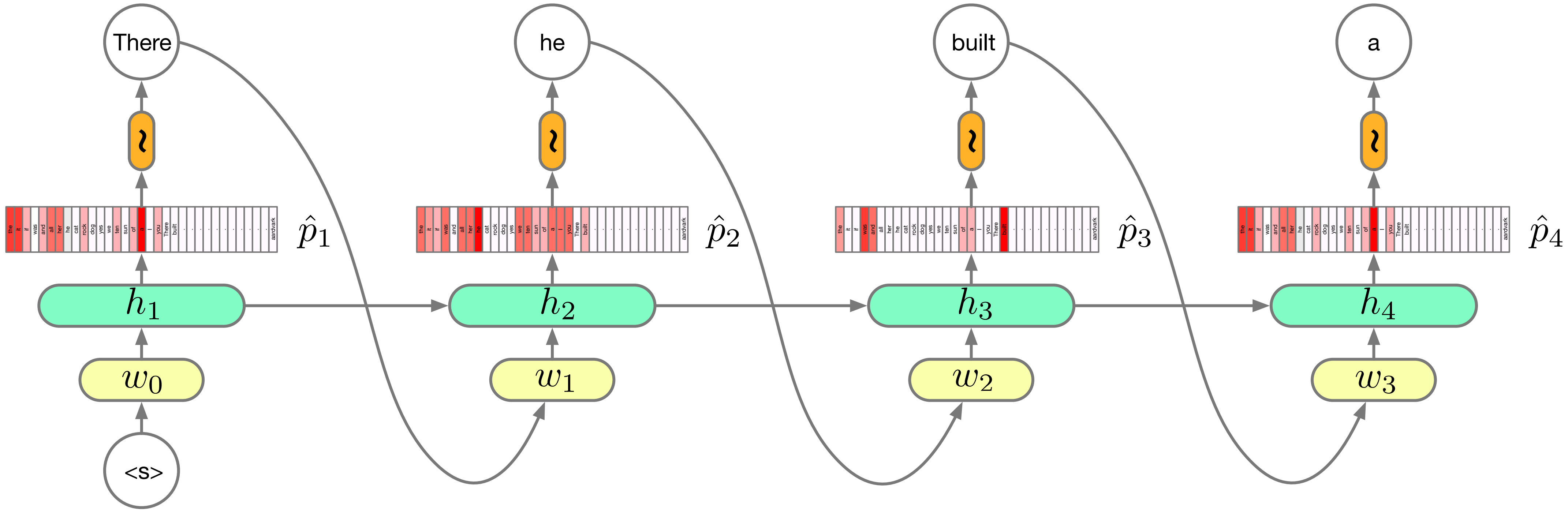
$$\hat{y}_n = W h_n + b$$



RNN Language Models

$$h_n = g(V\,[x_n; h_{n-1}] + c)$$

$$\hat{y}_n = W h_n + b$$



at first:

```
tyntd-iafhatawiaoihrdemot  lytdws  e ,tfti, astai f ogoh eoase rrranbyne 'nhthnee e
plia tklrgd t o idoe ns,smtt   h ne etie h,hregtrs nigtike,aoaenns lng
```

train more

```
"Tmont thithey" fomesscerliund
Keushey. Thom here
sheulke, anmerenith ol sivh I lalterthend Bleipile shuwy fil on aseterlome
coaniogennc Phe lism thond hon at. MeiDimorotion in ther thize."
```

train more

```
Aftair fall unsuch that the hall for Prince Velzonski's that me of
her hearly, and behs to so arwage fiving were to it beloge, pavu say falling misfort
how, and Gogition is so overelical and ofter.
```

train more

```
"Why do what that day," replied Natasha, and wishing to himself the fact the
princess, Princess Mary was easier, fed in had oftened him.
Pierre aking his soul came to the packs and drove up his father-in-law women.
```

http://karpathy.github.io/2015/05/21/rnn-effectiveness/

```
PANDARUS:
Alas, I think he shall be come approached and the day
When little srain would be attain'd into being never fed,
And who is but a chain and subjects of his death,
I should not sleep.

Second Senator:
They are away this miseries, produced upon my soul,
Breaking and strongly should be buried, when I perish
The earth and thoughts of many states.

DUKE VINCENTIO:
Well, your wit is in the care of side and that.

Second Lord:
They would be ruled after this chamber, and
my fair nues begun out of the fact, to be conveyed,
Whose noble souls I'll have the heart of the wars.

Clown:
Come, sir, I will make did behold your worship.

VIOLA:
I'll drink it.
```

```
VIOLA:
Why, Salisbury must find his flesh and thought
That which I am not aps, not a man and in fire,
To show the reining of the raven and the wars
To grace my hand reproach within, and not a fair are hand,
That Caesar and my goodly father's world;
When I was heaven of presence and our fleets,
We spare with hours, but cut thy council I am great,
Murdered and by thy master's ready there
My power to give thee but so much as hell:
Some service in the noble bondman here,
Would show him to her wine.

KING LEAR:
O, if you were a feeble sight, the courtesy of your law,
Your sight and several breath, will wear the gods
With his heads, and my hands are wonder'd at the deeds,
So drop upon your lordship's head, and your opinion
Shall be against your honour.
```

http://karpathy.github.io/2015/05/21/rnn-effectiveness/

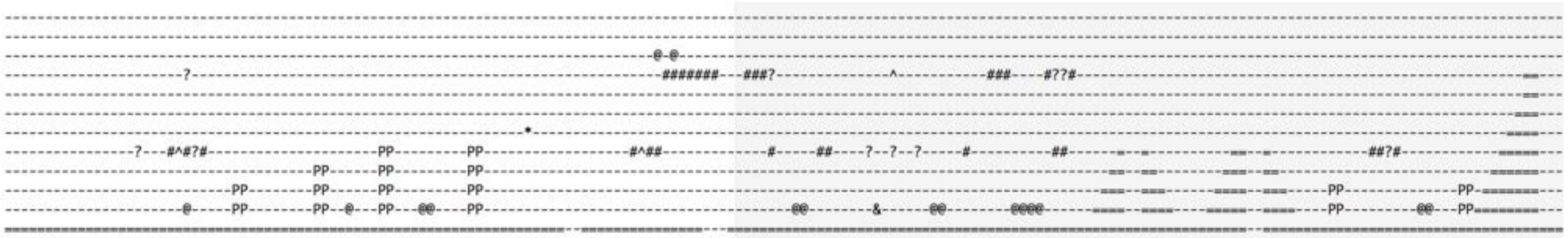
More Language Modeling Fun – DeepDrumpf

https://twitter.com/deepdrumpf

More Language Modeling Fun – Generating Super Mario Levels

Original Level:

Textual Representation:

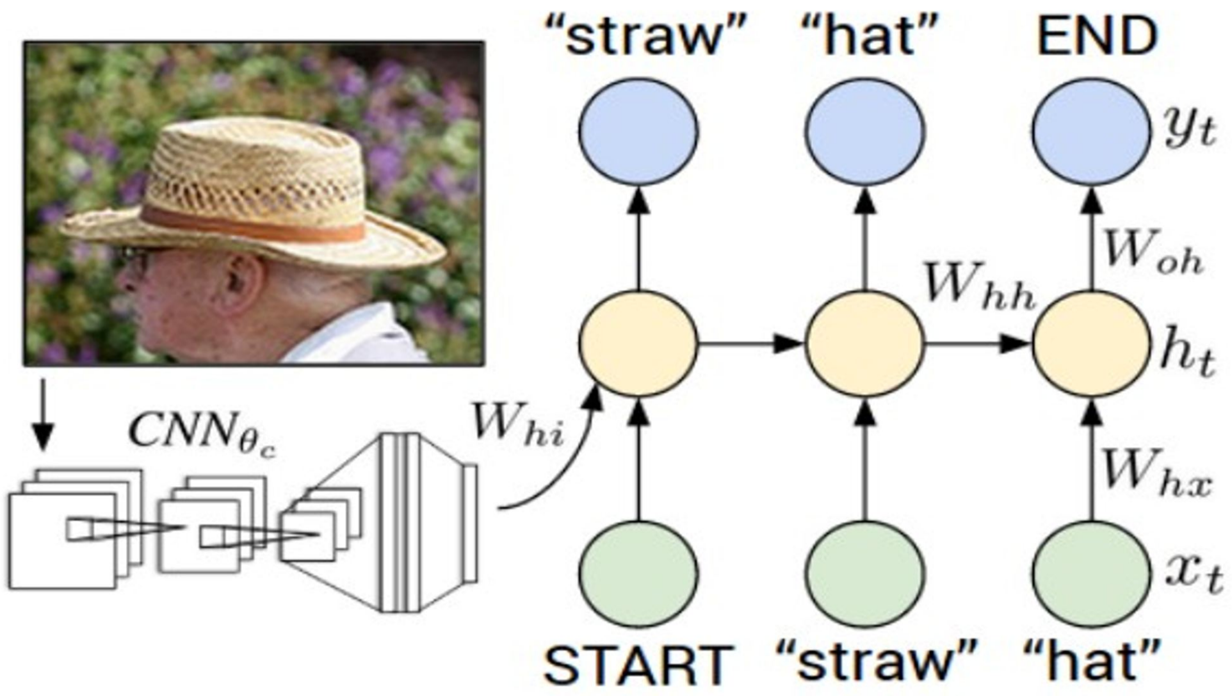


A level generated by a RNN:

https://medium.com/@ageitgey/machine-learning-is-fun-part-2-a26a10b68df3

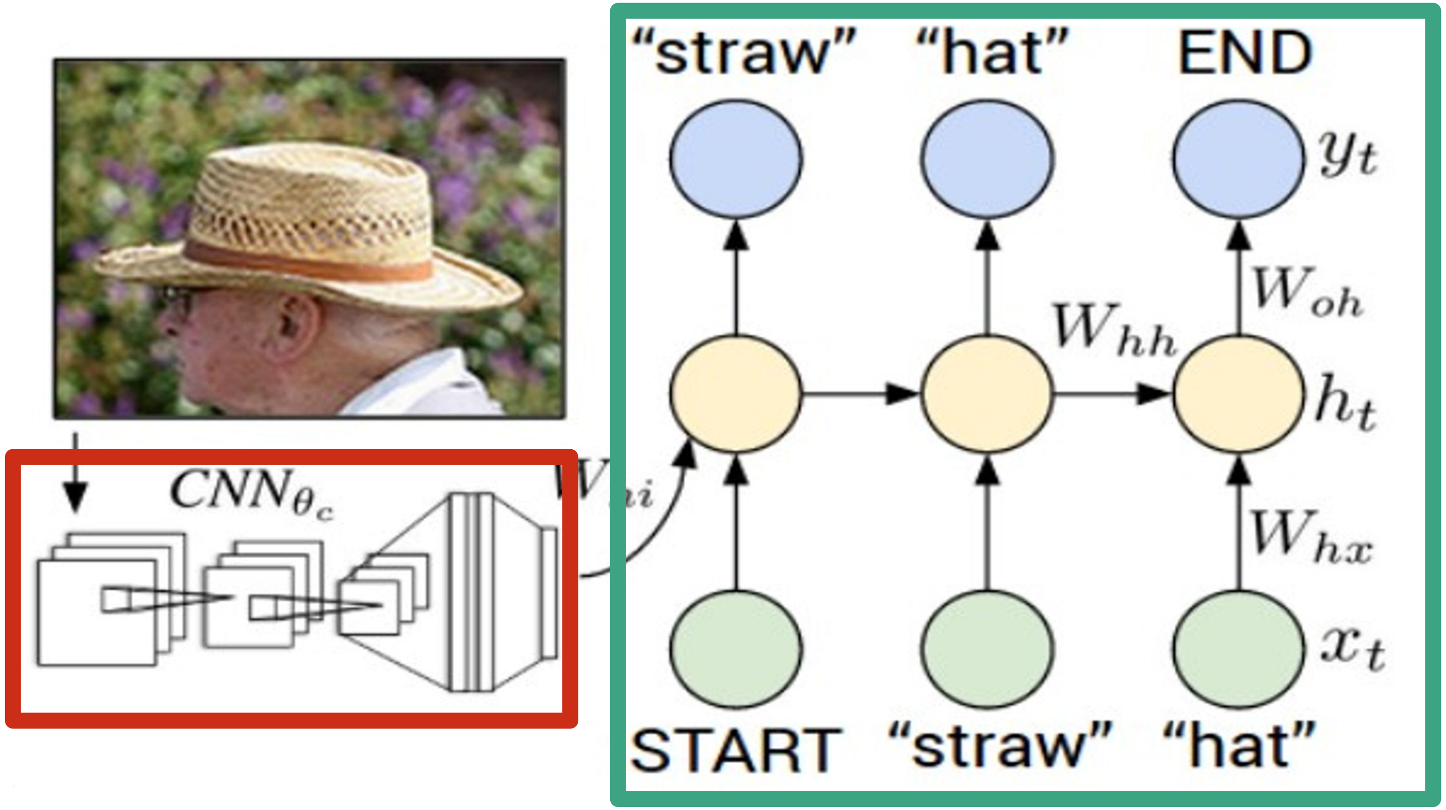
Image Captioning

Image Captioning



Explain Images with Multimodal Recurrent Neural Networks [Mao et al.]
Deep Visual-Semantic Alignments for Generating Image Descriptions [Karpathy and Fei-Fei]
Show and Tell: A Neural Image Caption Generator [Vinyals et al.]
Long-term Recurrent Convolutional Networks for Visual Recognition and Description [Donahue et al.]
Learning a Recurrent Visual Representation for Image Caption Generation [Chen and Zitnick]

Recurrent Neural Network



Convolutional Neural Network

test image

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

test image

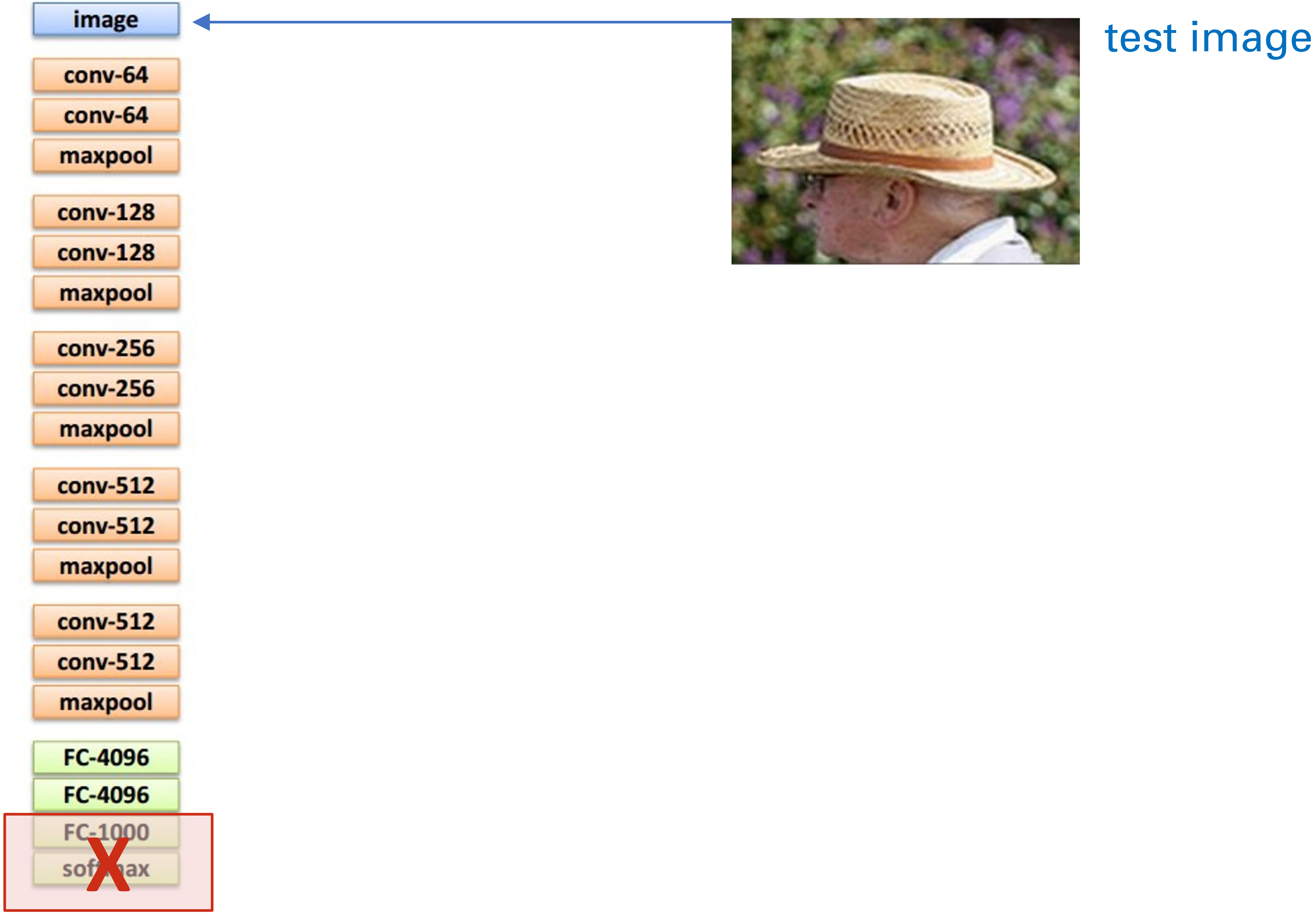


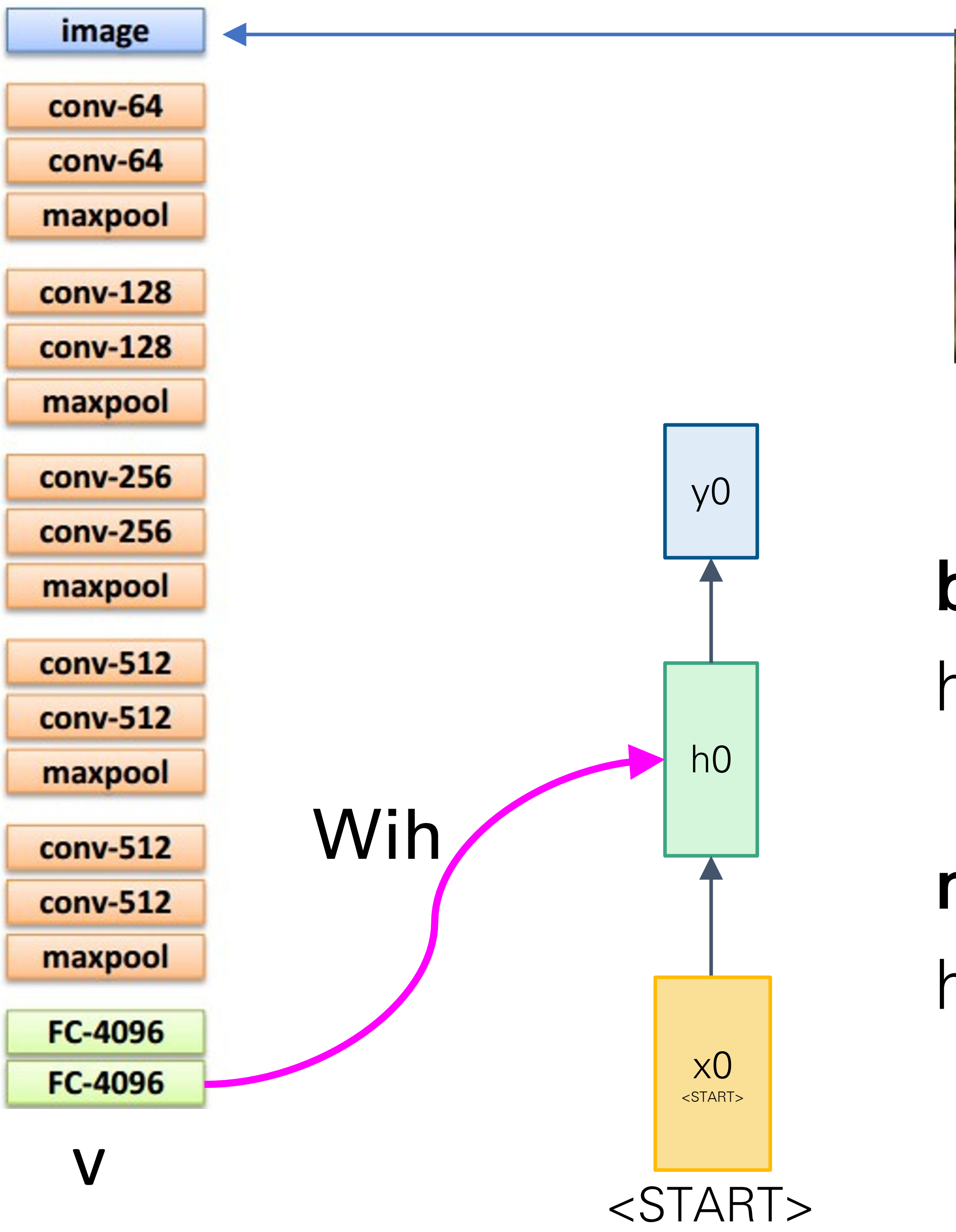
image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
conv-256
conv-256
maxpool
conv-512
conv-512
maxpool
conv-512
conv-512
maxpool
FC-4096
FC-4096
FC-1000
X
test image

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

x0
<START>

<START>



test image

before:

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

now:

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

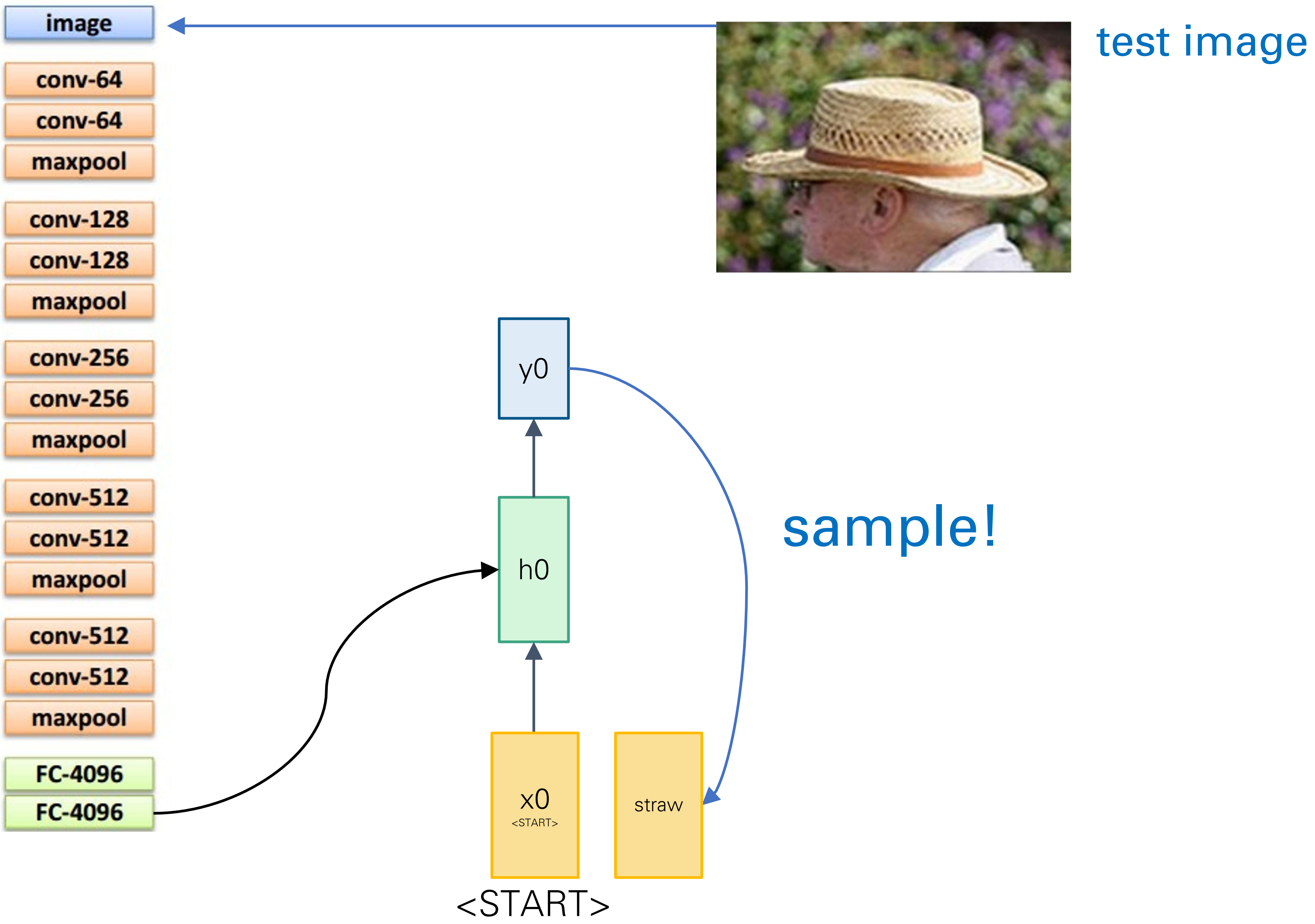


image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
conv-256
conv-256
maxpool
conv-512
conv-512
maxpool
conv-512
conv-512
maxpool
FC-4096
FC-4096
test image
y0
h0
sample!
x0
<START>
straw
<START>

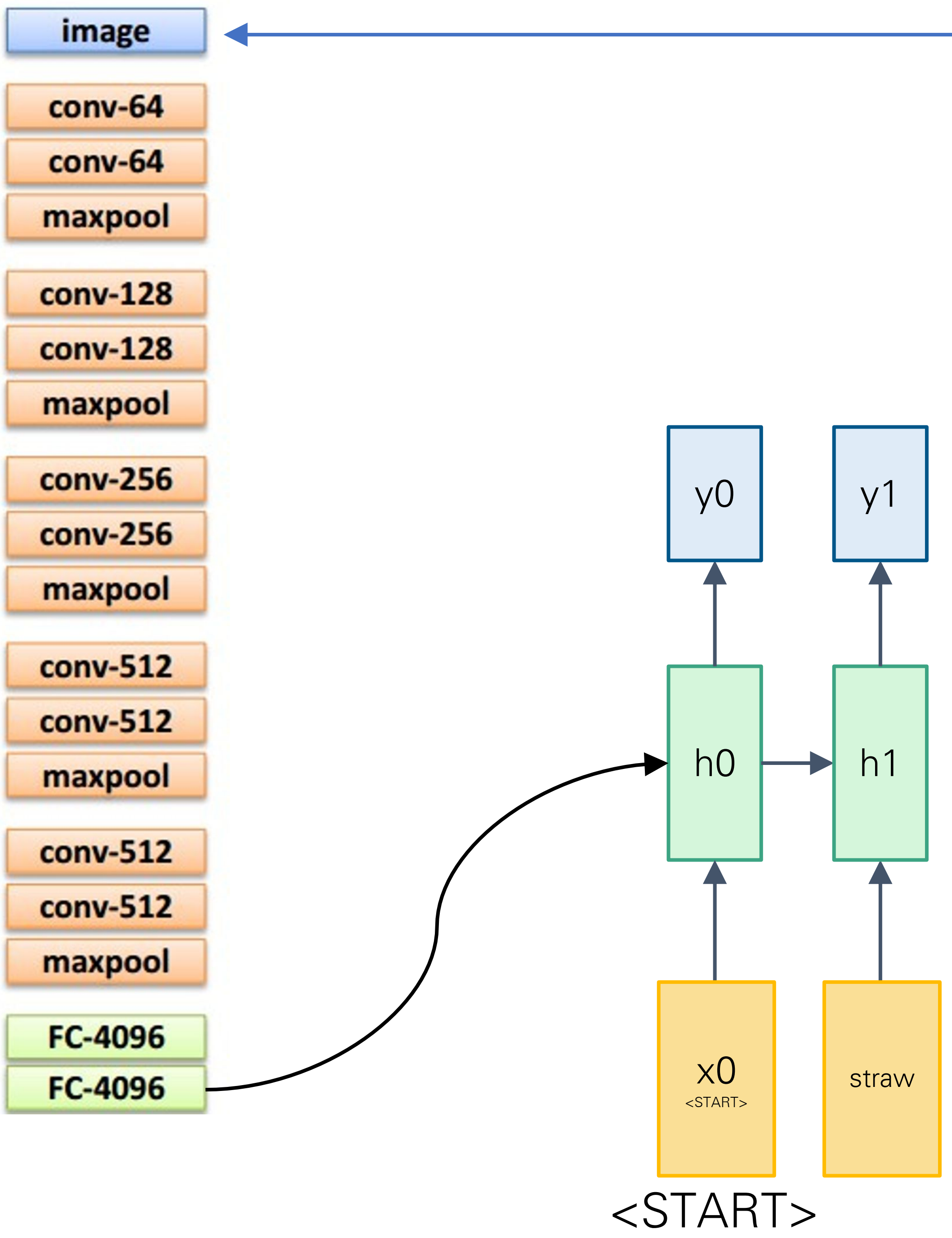


image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
conv-256
conv-256
maxpool
conv-512
conv-512
maxpool
conv-512
conv-512
maxpool
FC-4096
FC-4096
y0
y1
h0
h1
x0
<START>
straw
<START>

test image

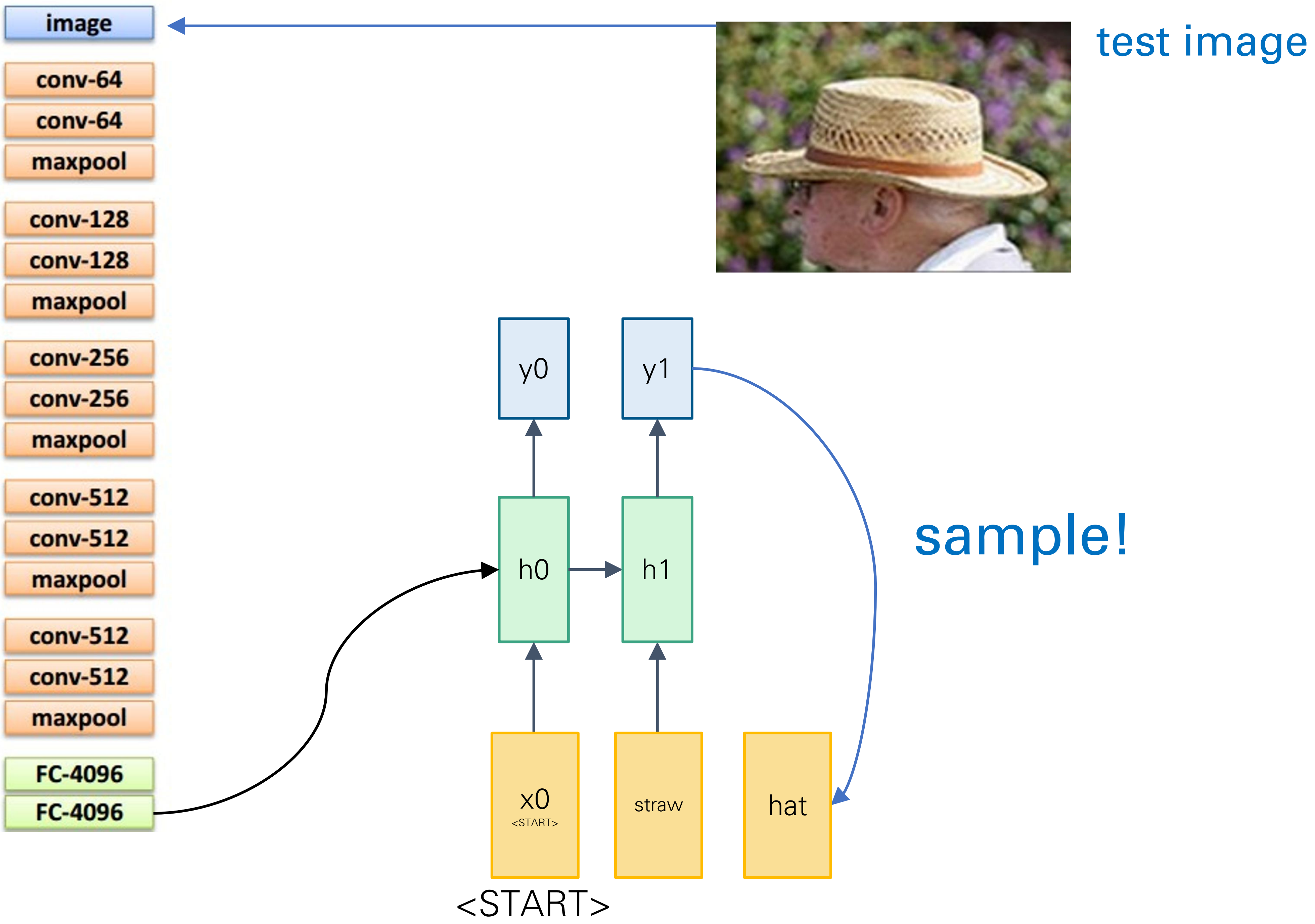


image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
conv-256
conv-256
maxpool
conv-512
conv-512
maxpool
conv-512
conv-512
maxpool
FC-4096
FC-4096
test image
y0
y1
h0
h1
sample!
x0
<START>
straw
hat
<START>

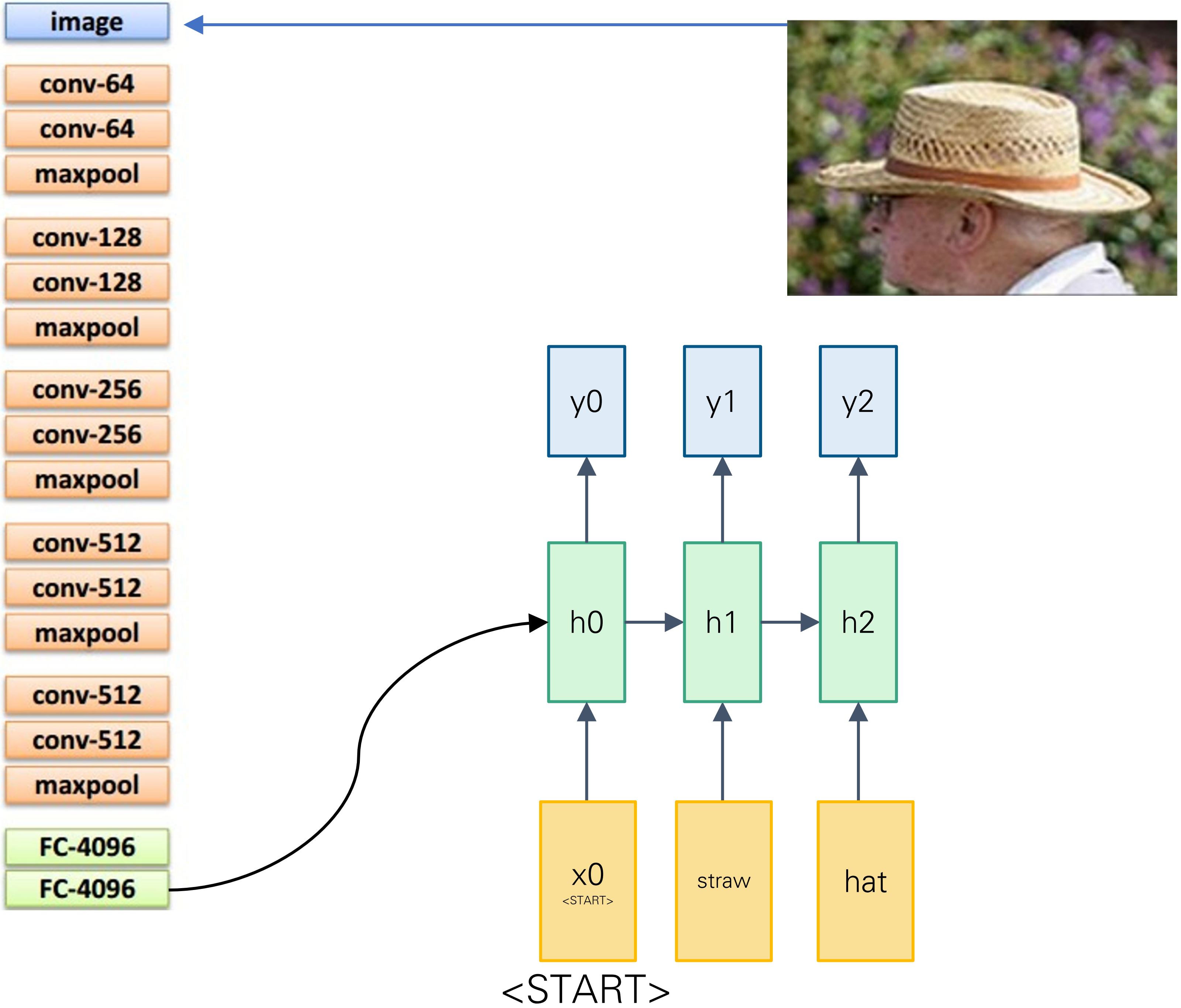
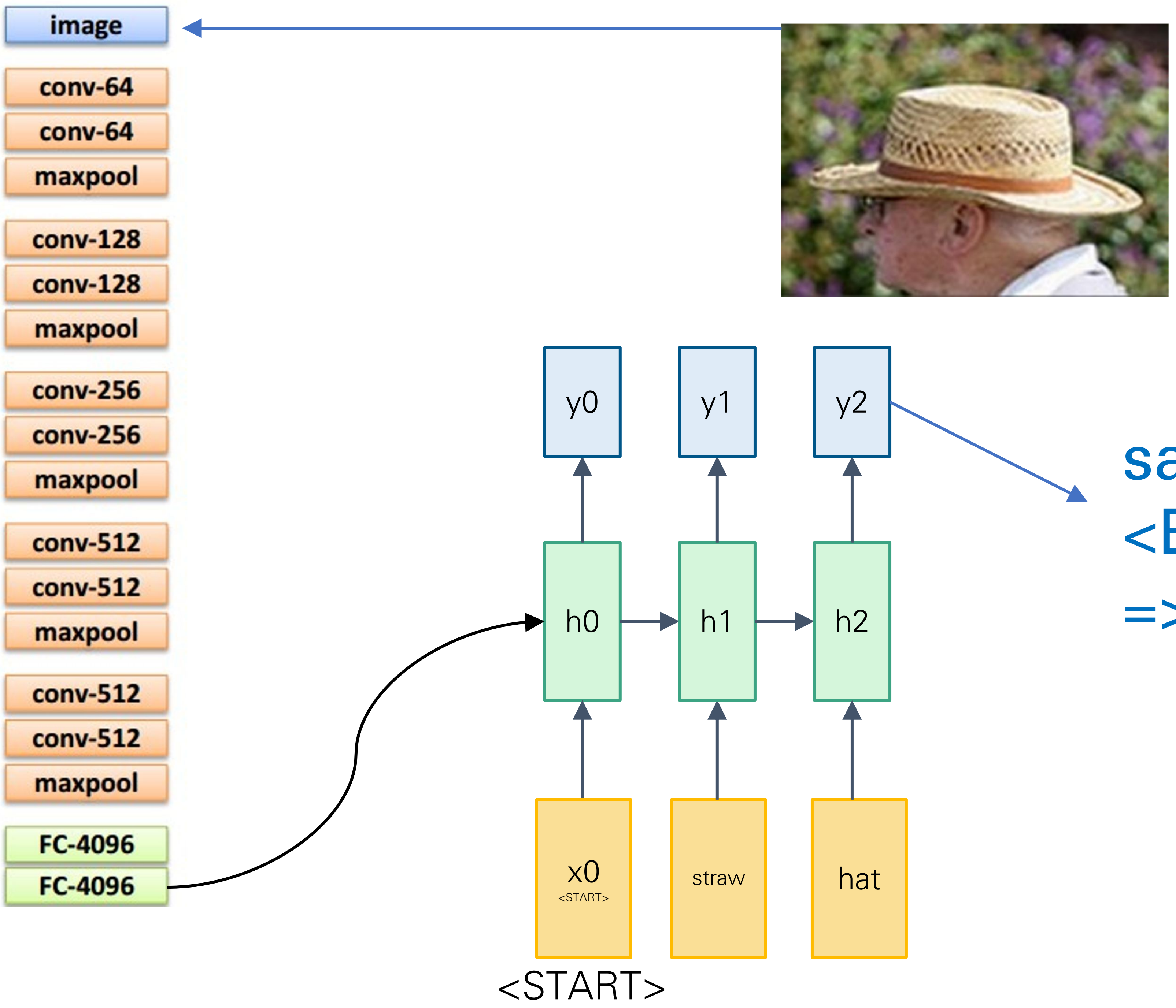


image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
conv-256
conv-256
maxpool
conv-512
conv-512
maxpool
conv-512
conv-512
maxpool
FC-4096
FC-4096
y0
y1
y2
h0
h1
h2
x0
<START>
straw
hat
<START>

test image



test image

sample <END> token => finish.

Beam Search (K = 3)

a

the

red

For t =1...T :

- For all k and for all possible output words w:

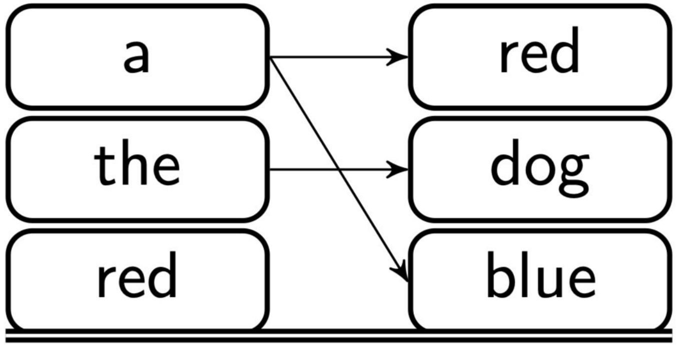
$$s(w, \hat{y}_{1:t-1}^{(k)}) \leftarrow \log p(\hat{y}_{1:t-1}^{(k)}|x) + \log p(w|\hat{y}_{1:t-1}^{(k)}, x)$$

- Update beam:

$$\hat{y}_{1:t}^{(1:K)} \leftarrow \text{K-}\arg\max\, s(w, \hat{y}_{1:t-1}^{(k)})$$

Slide credit: Alexander Rush

Beam Search (K = 3)



For t =1...T :

- For all k and for all possible output words w:

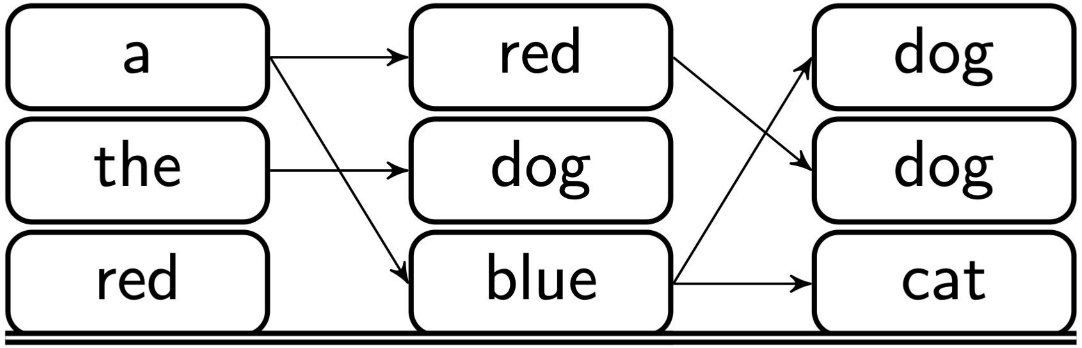
$$s(w, \hat{y}_{1:t-1}^{(k)}) \leftarrow \log p(\hat{y}_{1:t-1}^{(k)}|x) + \log p(w|\hat{y}_{1:t-1}^{(k)}, x)$$

- Update beam:

$$\hat{y}_{1:t}^{(1:K)} \leftarrow \text{K-arg max } s(w, \hat{y}_{1:t-1}^{(k)})$$

Slide credit: Alexander Rush

Beam Search (K = 3)



For t =1…T :

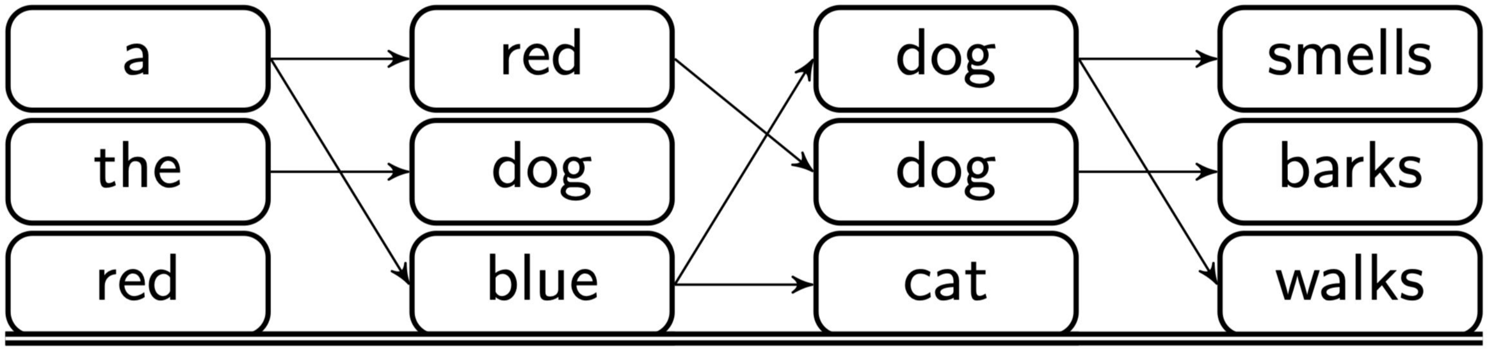
- For all k and for all possible output words w:

$$s(w, \hat{y}^{(k)}_{1:t-1}) \leftarrow \log p(\hat{y}^{(k)}_{1:t-1}|x) + \log p(w|\hat{y}^{(k)}_{1:t-1}, x)$$

- Update beam:

$$\hat{y}^{(1:K)}_{1:t} \leftarrow \text{K-arg max } s(w, \hat{y}^{(k)}_{1:t-1})$$

Beam Search (K = 3)



For t =1...T :

- For all k and for all possible output words w:

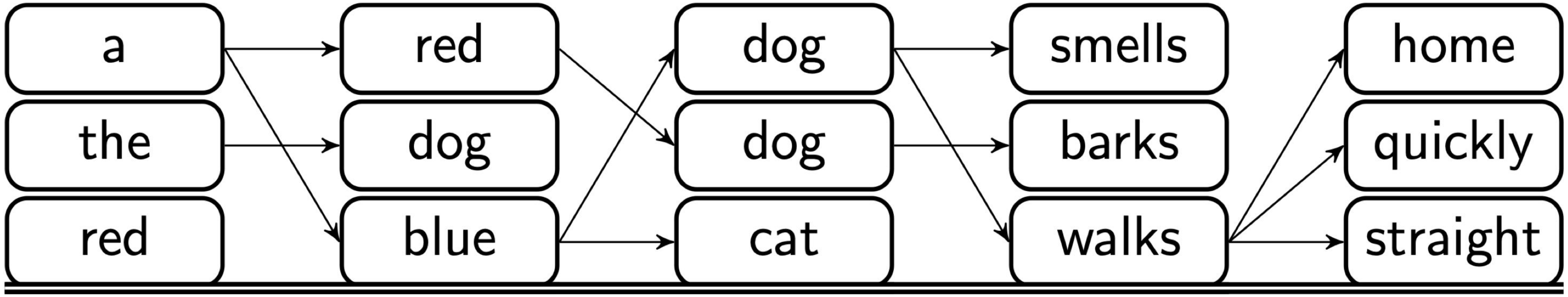
$$s(w, \hat{y}^{(k)}_{1:t-1}) \leftarrow \log p(\hat{y}^{(k)}_{1:t-1}|x) + \log p(w|\hat{y}^{(k)}_{1:t-1}, x)$$

- Update beam:

$$\hat{y}^{(1:K)}_{1:t} \leftarrow \text{K-arg max } s(w, \hat{y}^{(k)}_{1:t-1})$$

Slide credit: Alexander Rush

Beam Search (K = 3)



For t =1...T :

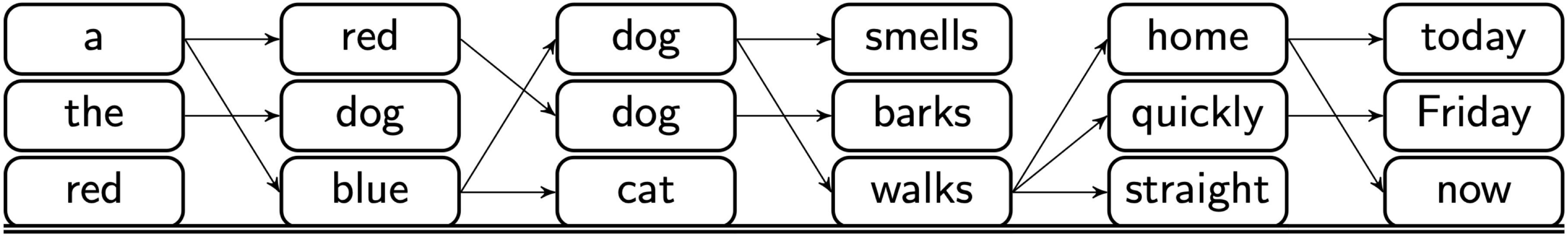
- For all k and for all possible output words w:

$$s(w, \hat{y}^{(k)}_{1:t-1}) \leftarrow \log p(\hat{y}^{(k)}_{1:t-1}|x) + \log p(w|\hat{y}^{(k)}_{1:t-1}, x)$$

- Update beam:

$$\hat{y}^{(1:K)}_{1:t} \leftarrow \text{K-arg max } s(w, \hat{y}^{(k)}_{1:t-1})$$

Beam Search (K = 3)



For t =1...T :

- For all k and for all possible output words w:

$$s(w, \hat{y}^{(k)}_{1:t-1}) \leftarrow \log p(\hat{y}^{(k)}_{1:t-1}|x) + \log p(w|\hat{y}^{(k)}_{1:t-1}, x)$$

- Update beam:

$$\hat{y}^{(1:K)}_{1:t} \leftarrow \text{K-arg}\max\, s(w, \hat{y}^{(k)}_{1:t-1})$$

Slide credit: Alexander Rush

Image Description Datasets

a man riding a bike on a dirt path through a forest.
bicyclist raises his fist as he rides on desert dirt trail.
this dirt bike rider is smiling and raising his fist in triumph.
a man riding a bicycle while pumping his fist in the air.
a mountain biker pumps his fist in celebration.

Microsoft COCO
[Tsung-Yi Lin et al. 2014]
mscoco.org

currently:
~120K images
~5 sentences each

"man in black shirt is playing guitar."

"construction worker in orange safety vest is working on road."

"two young girls are playing with lego toy."

"boy is doing backflip on wakeboard."

"man in black shirt is playing guitar."

"construction worker in orange safety vest is working on road."

"two young girls are playing with lego toy."

"boy is doing backflip on wakeboard."

"a young boy is holding a baseball bat."

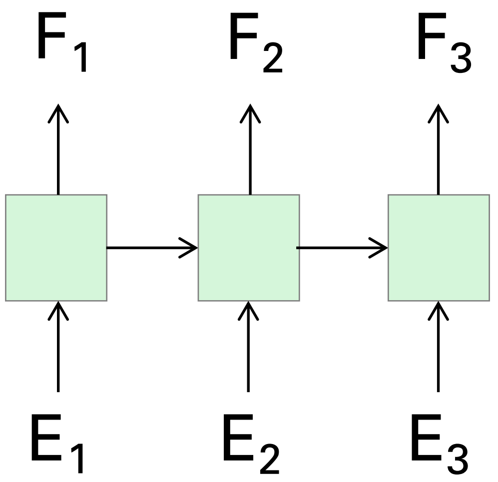
"a cat is sitting on a couch with a remote control."

"a woman holding a teddy bear in front of a mirror."

"a horse is standing in the middle of a road."

Class Exercise

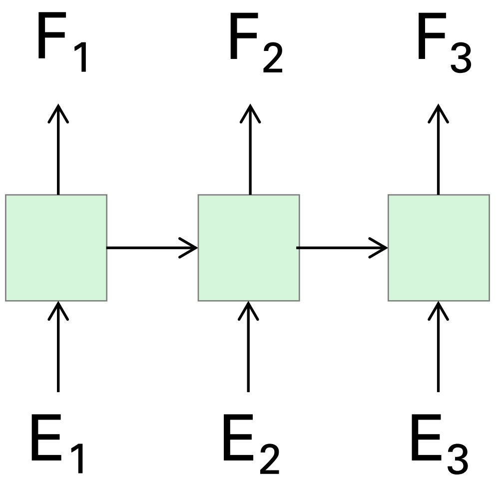
- Consider the problem of translation of English to French
- E.g. **What is your name ⟶ Comment tu t'appelle**
- Is the below architecture suitable for this problem?



Adapted from http://www.cs.toronto.edu/~rgrosse/csc321/lec10.pdf

Class Exercise

- Consider the problem of translation of English to French
- E.g. What is your name $\longrightarrow$ Comment tu t'appelle
- Is the below architecture suitable for this problem?

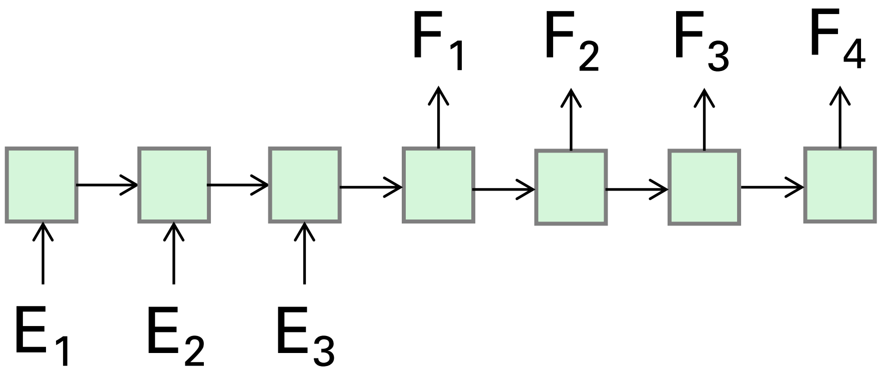


- No, sentences might be of different length and words might not align. Need to see entire sentence before translating

Adapted from http://www.cs.toronto.edu/~rgrosse/csc321/lec10.pdf

Encoder-Decoder Seq2Seq Model

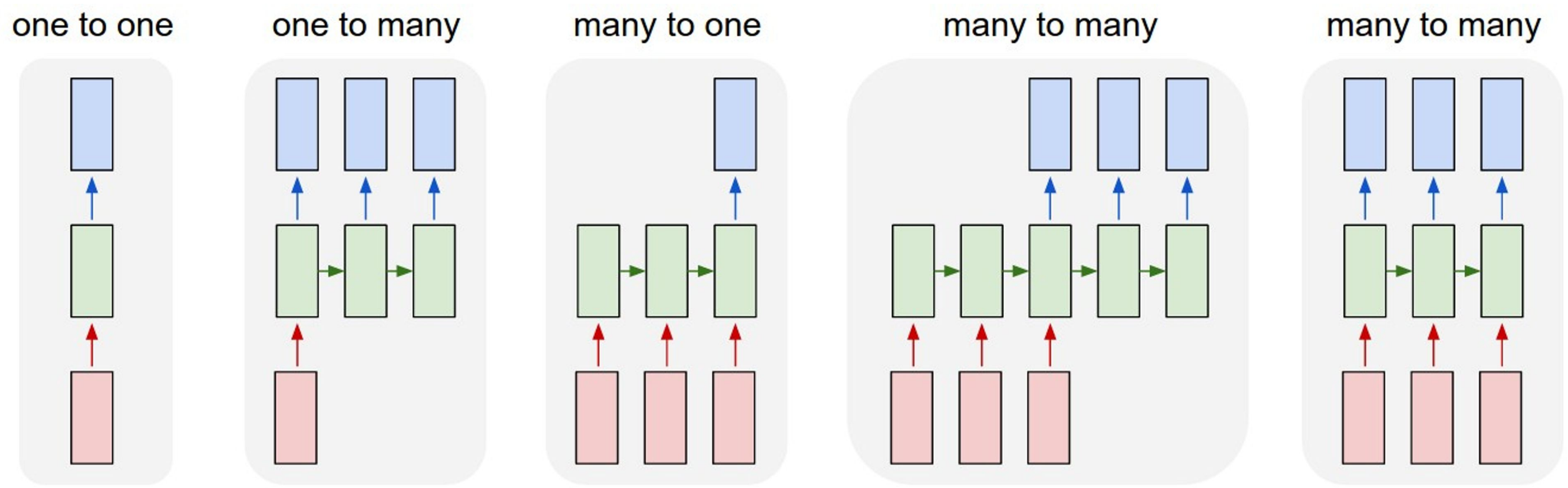
- Consider the problem of translation of English to French
- E.g. What is your name $\longrightarrow$ Comment tu t'appelle
- Sentences might be of different length and words might not align. Need to see entire sentence before translating



- Input-Output nature depends on the structure of the problem at hand

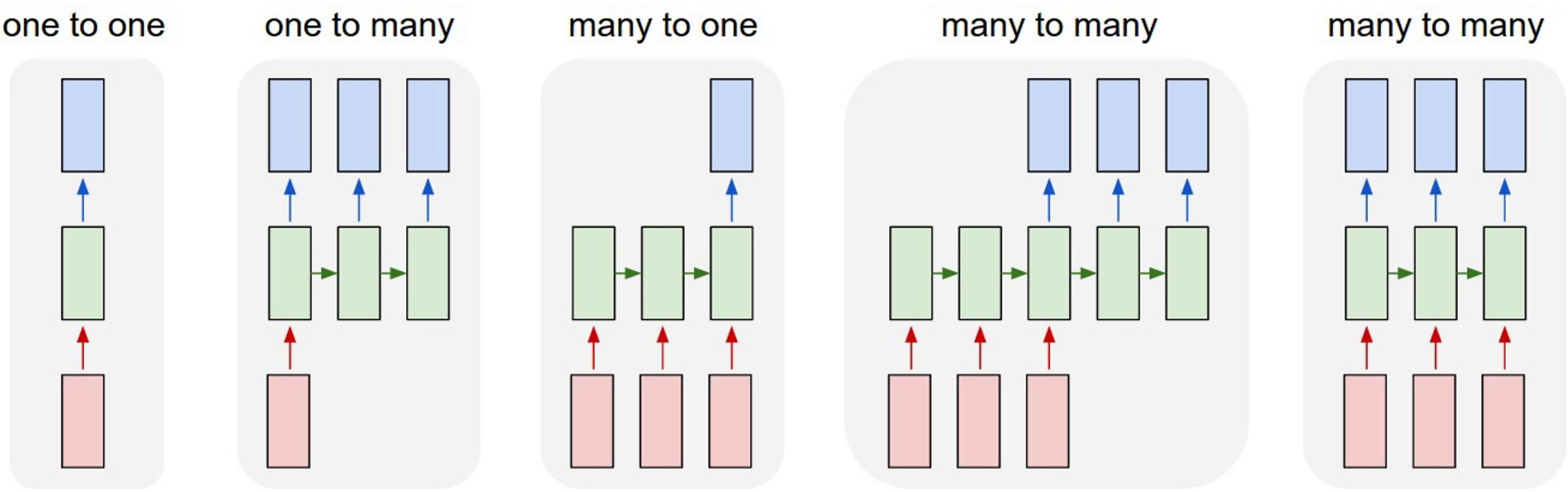
[Seq2Seq Learning with Neural Networks. Sutskever et al., NIPS 2014]

Recurrent Networks offer a lot of flexibility:



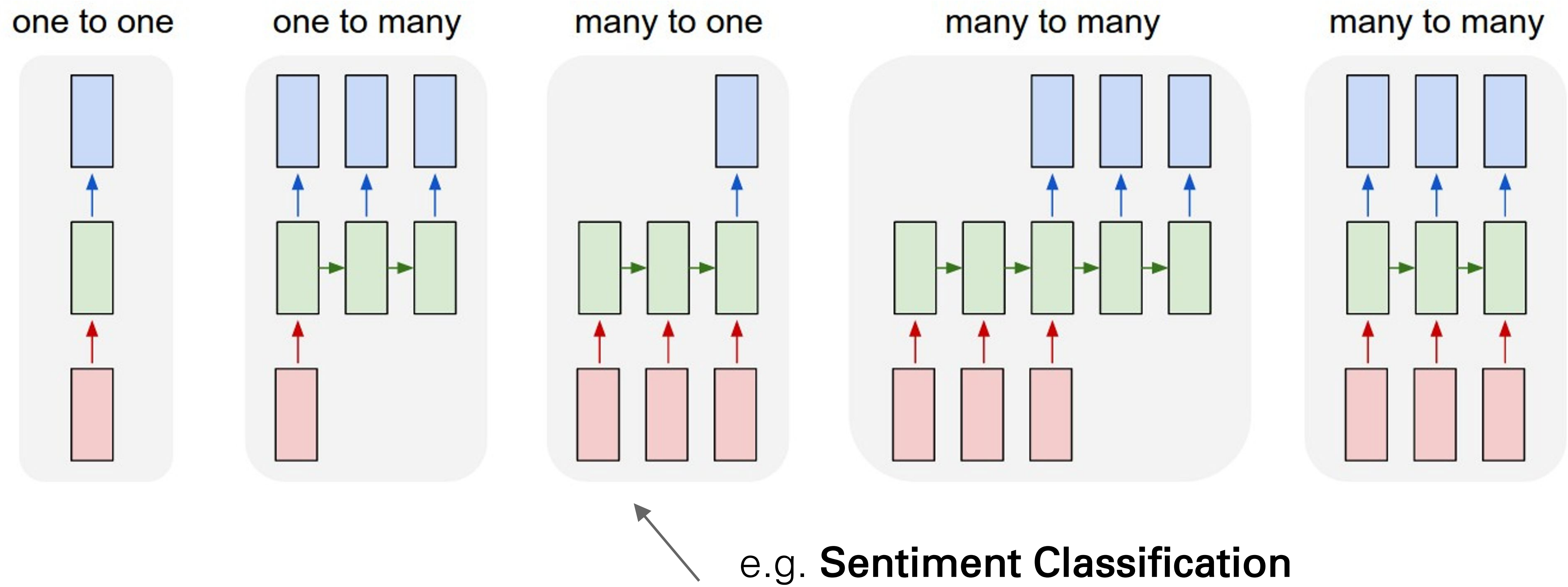
Vanilla Neural Networks

Recurrent Networks offer a lot of flexibility:



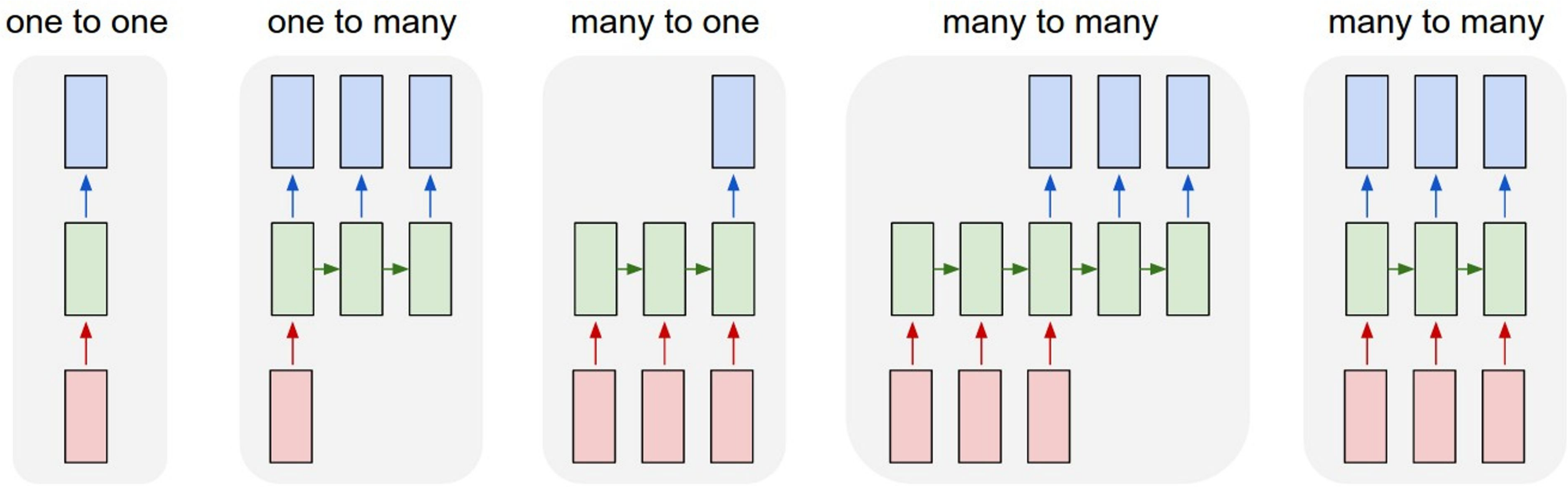
e.g. **Image Captioning**
image -> sequence of words

Recurrent Networks offer a lot of flexibility:



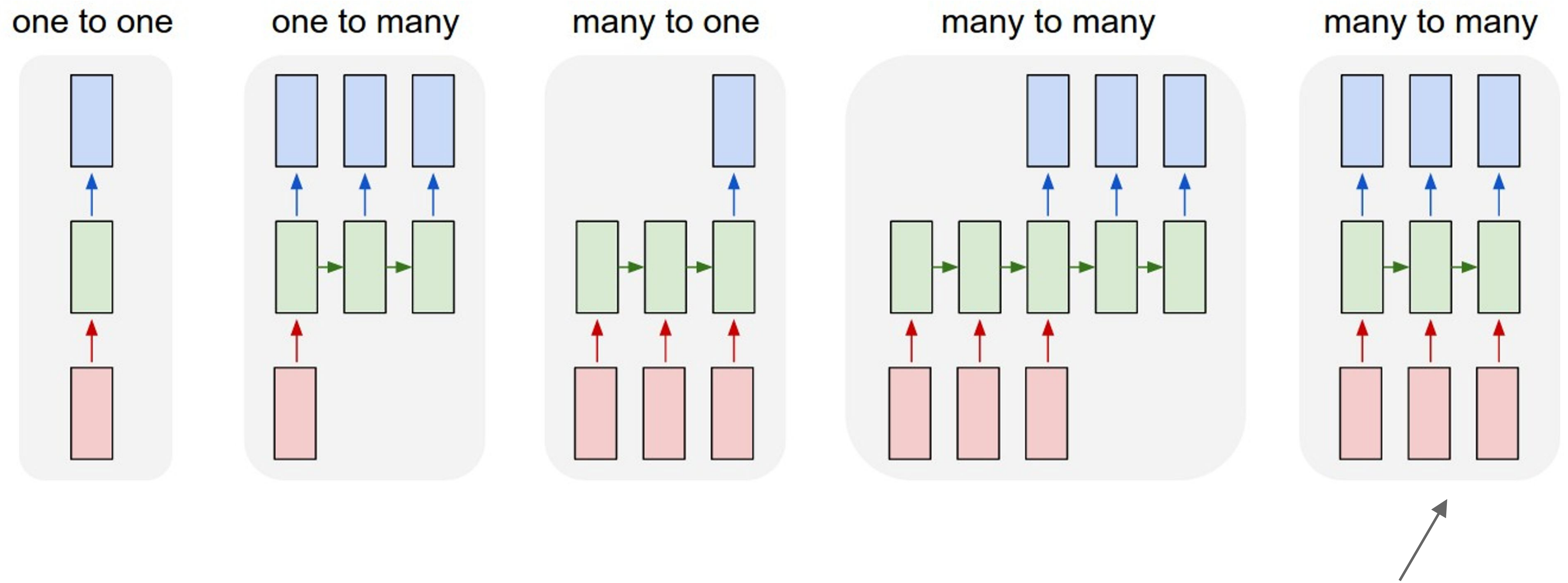
e.g. **Sentiment Classification**
sequence of words -> sentiment

Recurrent Networks offer a lot of flexibility:



e.g. **Machine Translation**
seq of words -> seq of words

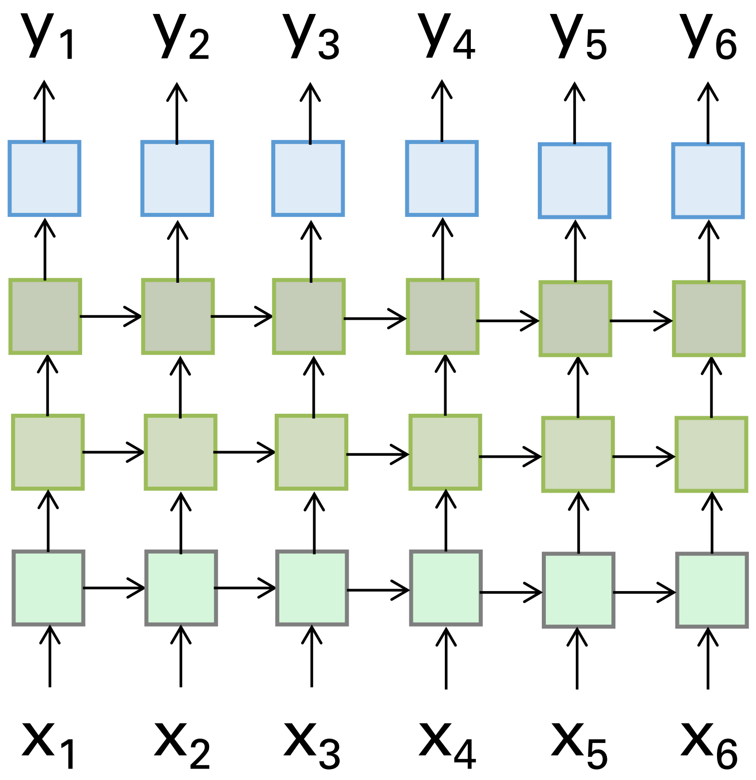
Recurrent Networks offer a lot of flexibility:



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

Multi-layer RNNs

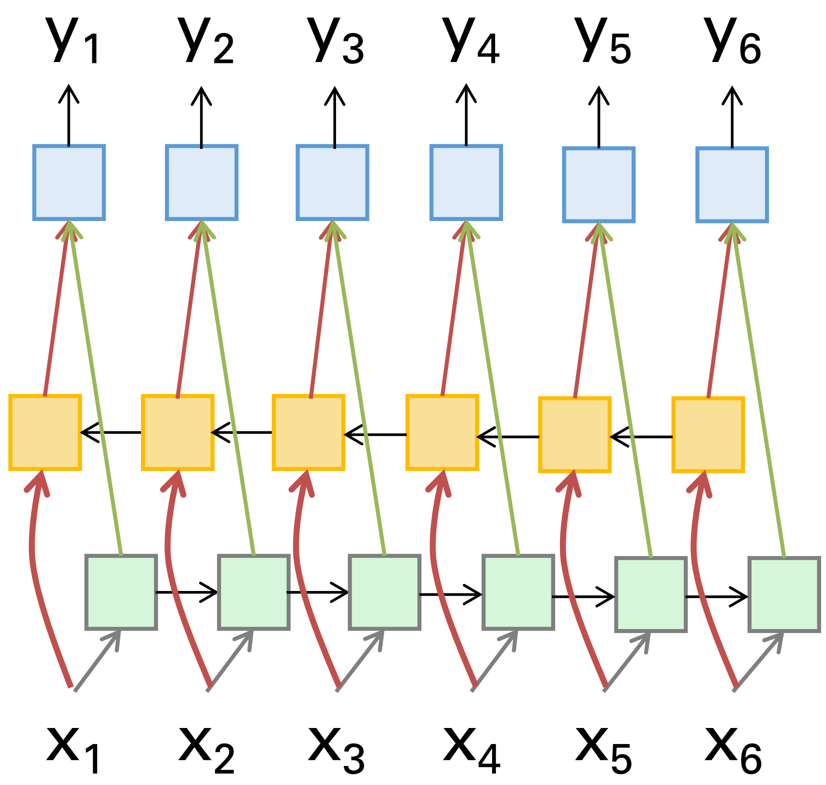
- We can of course design RNNs with multiple hidden layers



- Think exotic: Skip connections across layers, across time, …

Bi-directional RNNs

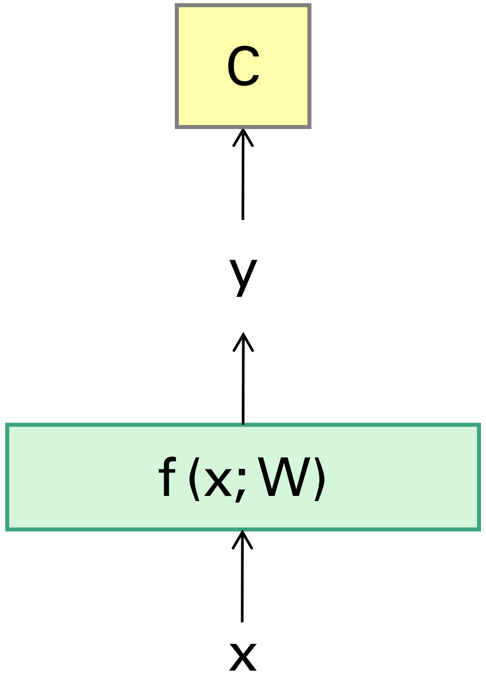
- RNNs can process the input sequence in forward and in the reverse direction



- Popular in speech recognition and machine translation

How to Train Recurrent Neural Networks

BackPropagation Refresher



$$y = f(x;W)$$

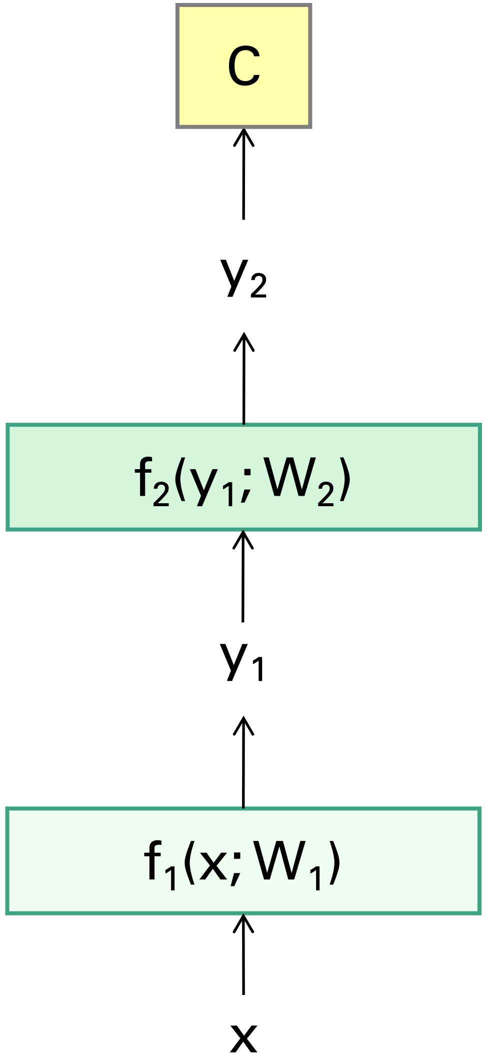
$$C = \text{Loss}(y, y_{GT})$$

SGD Update

$$W \leftarrow W - \eta \frac{\partial C}{\partial W}$$

$$\frac{\partial C}{\partial W} = \left(\frac{\partial C}{\partial y}\right)\left(\frac{\partial y}{\partial W}\right)$$

Multiple Layers



$$y_1 = f_1(x; W_1)$$

$$y_2 = f_2(y_1; W_2)$$

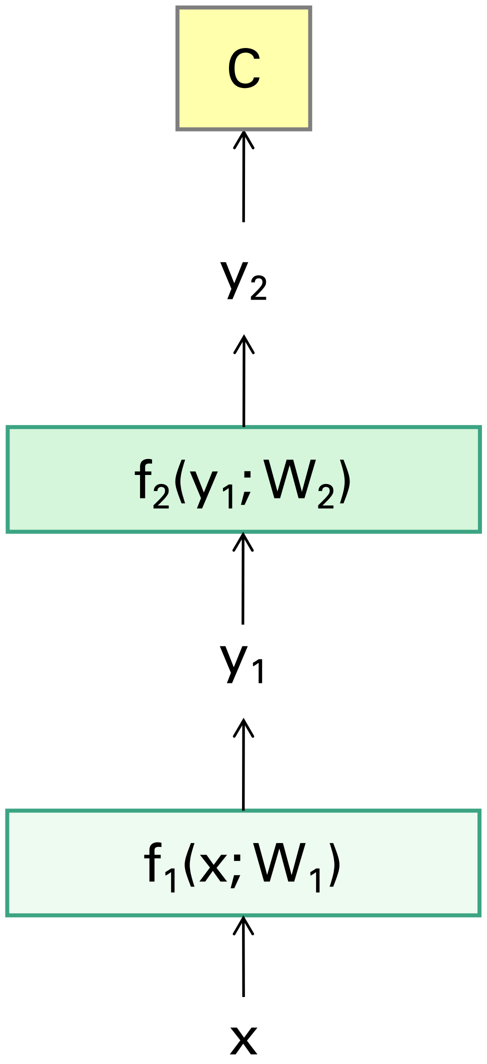
$$C = \text{Loss}(y_2, y_{GT})$$

SGD Update

$$W_2 \leftarrow W_2 - \eta \frac{\partial C}{\partial W_2}$$

$$W_1 \leftarrow W_1 - \eta \frac{\partial C}{\partial W_1}$$

Chain Rule for Gradient Computation



$$y_1 = f_1(x; W_1)$$

$$y_2 = f_2(y_1; W_2)$$

$$C = \mathrm{Loss}(y_2, y_{GT})$$

Find $\frac{\partial C}{\partial W_1}, \frac{\partial C}{\partial W_2}$

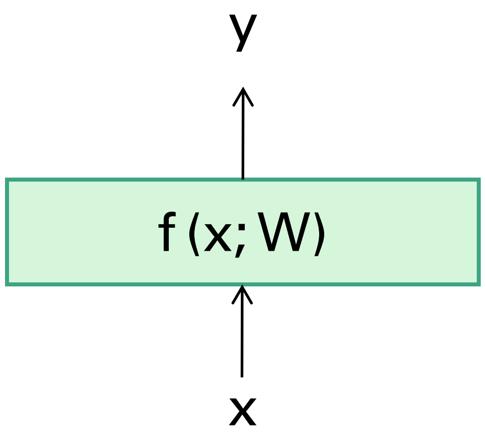
$$\frac{\partial C}{\partial W_2} = \left(\frac{\partial C}{\partial y_2}\right)\left(\frac{\partial y_2}{\partial W_2}\right)$$

$$\frac{\partial C}{\partial W_1} = \left(\frac{\partial C}{\partial y_1}\right)\left(\frac{\partial y_1}{\partial W_1}\right)$$

$$= \left(\frac{\partial C}{\partial y_2}\right)\left(\frac{\partial y_2}{\partial y_1}\right)\left(\frac{\partial y_1}{\partial W_1}\right)$$

Application of the Chain Rule

Chain Rule for Gradient Computation



Given: $\left(\frac{\partial C}{\partial y}\right)$

We are interested in computing: $\left(\frac{\partial C}{\partial W}\right), \left(\frac{\partial C}{\partial x}\right)$

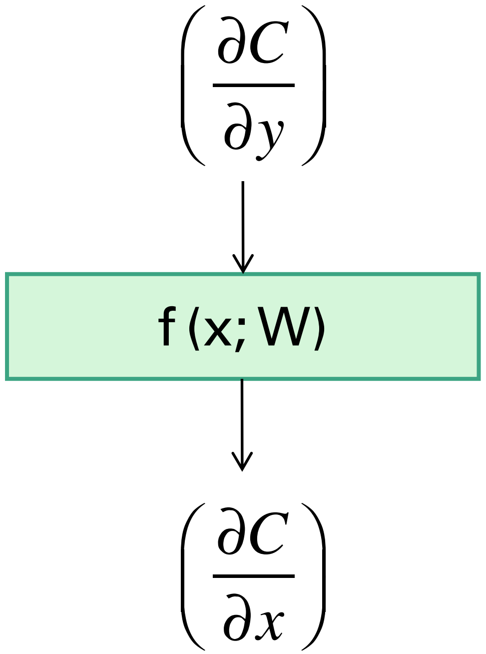
Intrinsic to the layer are:

$\left(\frac{\partial y}{\partial W}\right)$ – How does output change due to params

$\left(\frac{\partial y}{\partial x}\right)$ – How does output change due to inputs

$$\left(\frac{\partial C}{\partial W}\right) = \left(\frac{\partial C}{\partial y}\right)\left(\frac{\partial y}{\partial W}\right) \qquad \left(\frac{\partial C}{\partial x}\right) = \left(\frac{\partial C}{\partial y}\right)\left(\frac{\partial y}{\partial x}\right)$$

Chain Rule for Gradient Computation



Given: $\left(\frac{\partial C}{\partial y}\right)$

We are interested in computing: $\left(\frac{\partial C}{\partial W}\right), \left(\frac{\partial C}{\partial x}\right)$

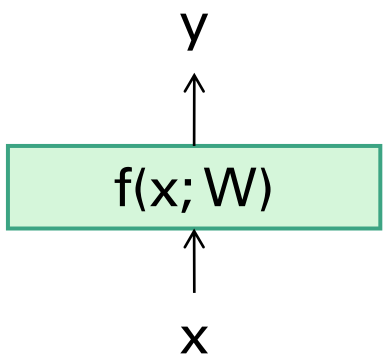
Intrinsic to the layer are:

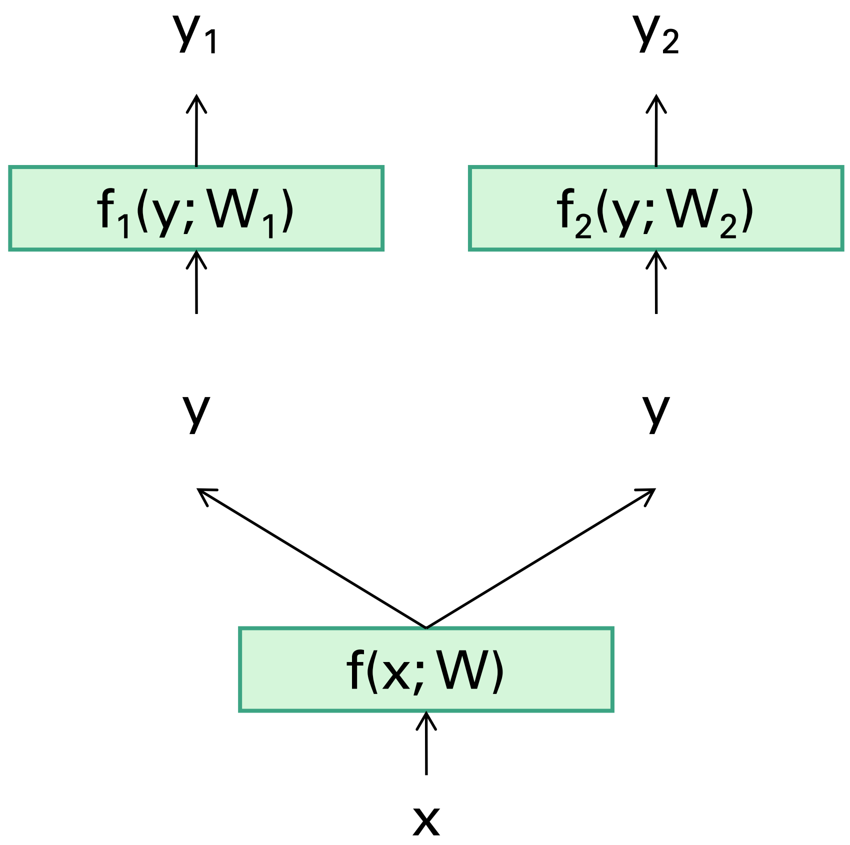
$\left(\frac{\partial y}{\partial W}\right)$ – How does output change due to params

$\left(\frac{\partial y}{\partial x}\right)$ – How does output change due to inputs

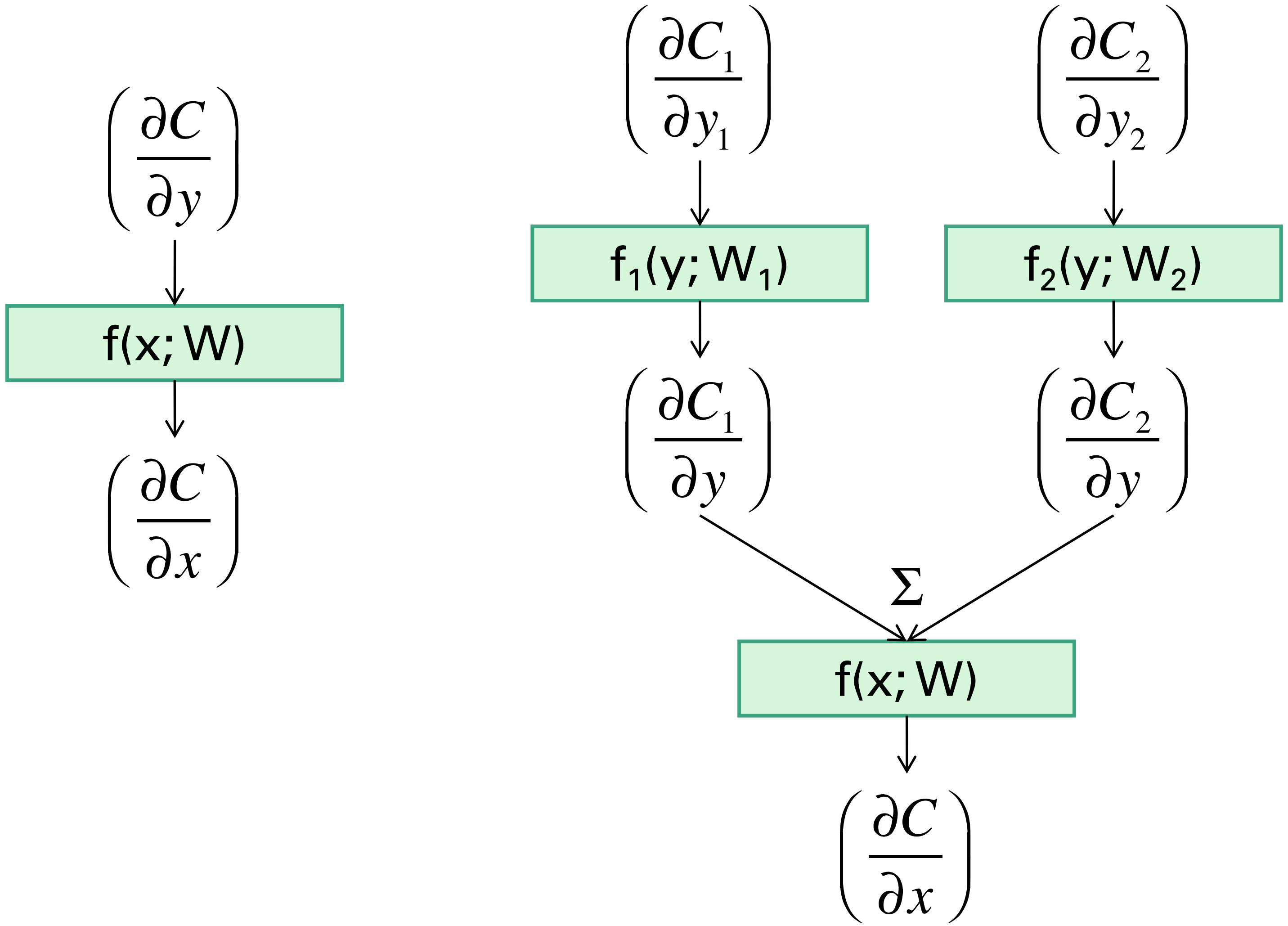
$$\left(\frac{\partial C}{\partial W}\right) = \left(\frac{\partial C}{\partial y}\right)\left(\frac{\partial y}{\partial W}\right) \qquad \left(\frac{\partial C}{\partial x}\right) = \left(\frac{\partial C}{\partial y}\right)\left(\frac{\partial y}{\partial x}\right)$$

Extension to Computational Graphs

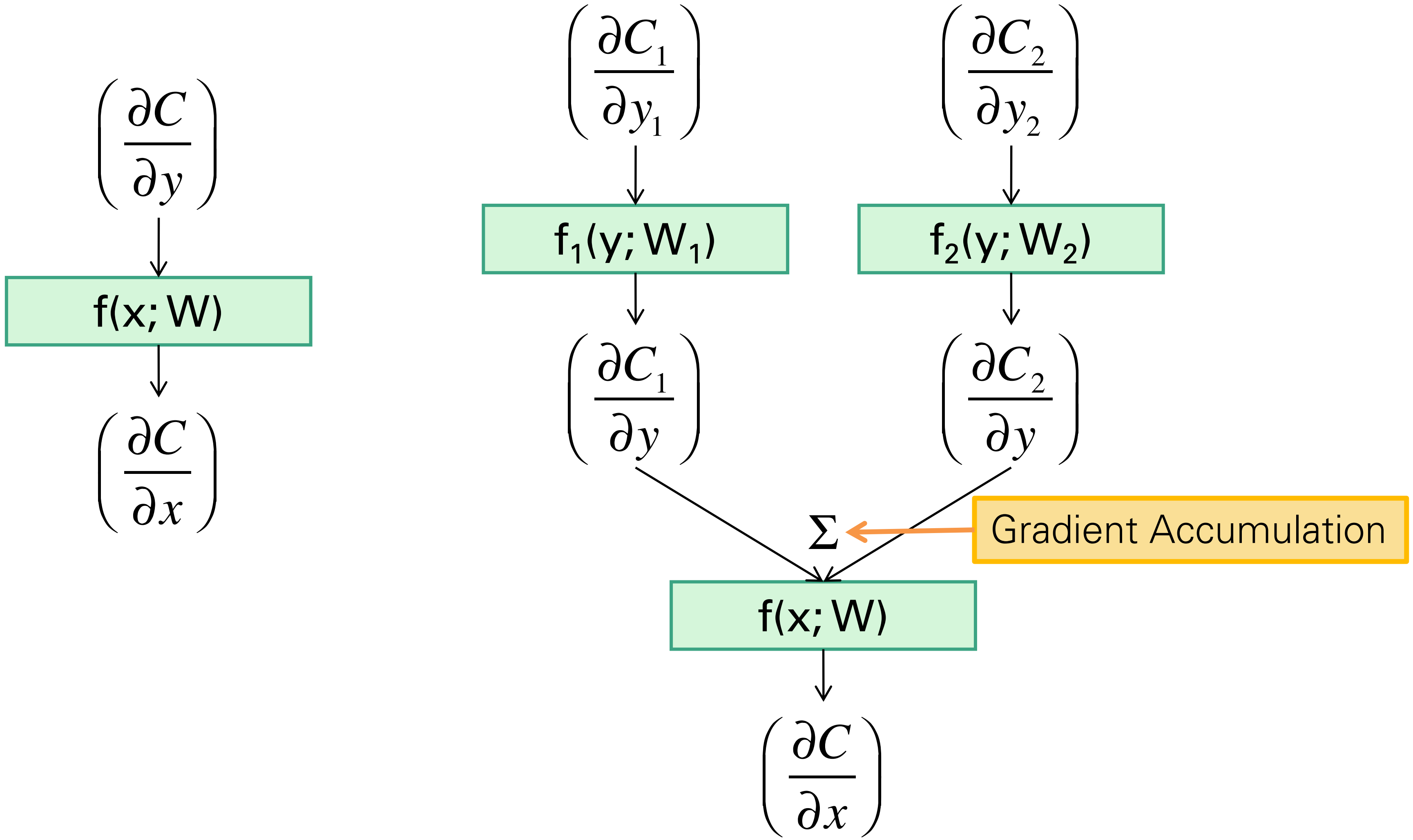




Extension to Computational Graphs

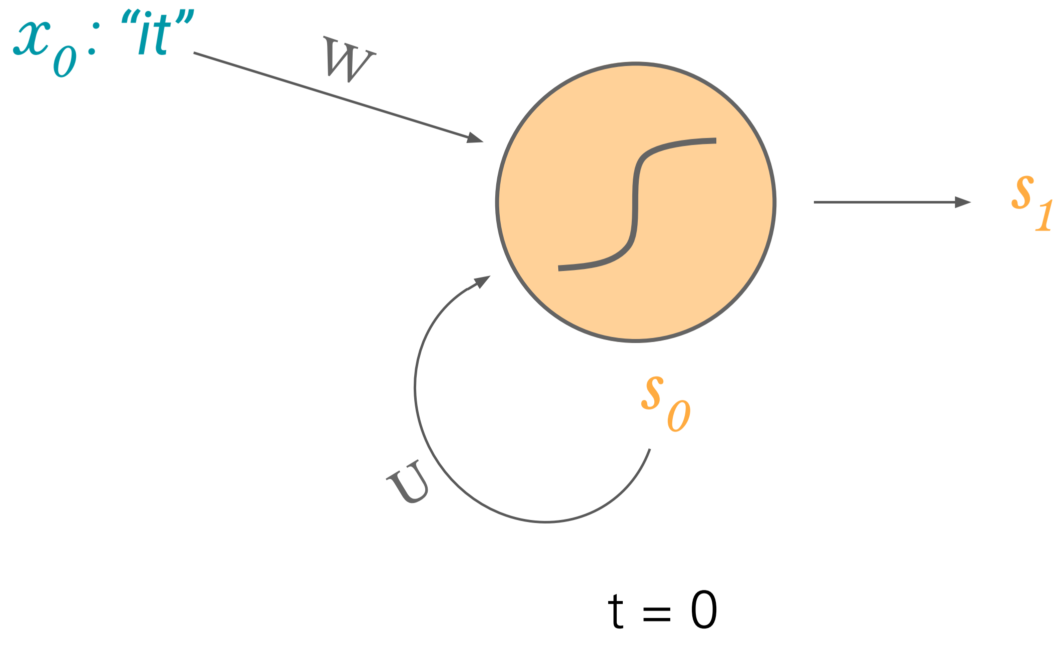


Extension to Computational Graphs



Sample RNN

- RNNs remember their previous state:



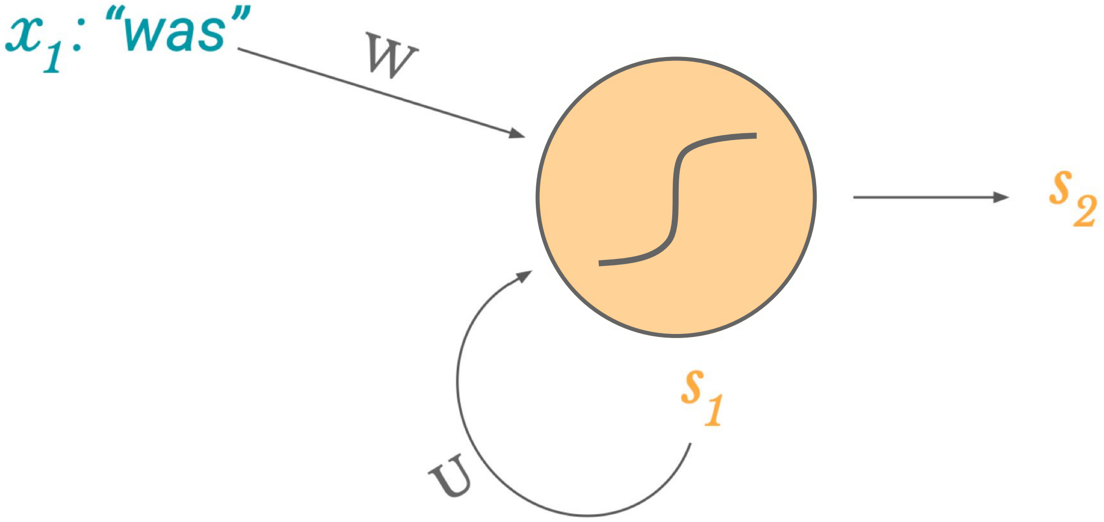
x_0 : vector representing first word
s_0 : cell state at $t = 0$ (some initialization)
s_1 : cell state at $t = 1$

$$s_1 = tanh(Wx_0 + Us_0)$$

$W,\ U$: weight matrices

Sample RNN

- RNNs remember their previous state:



t = 1

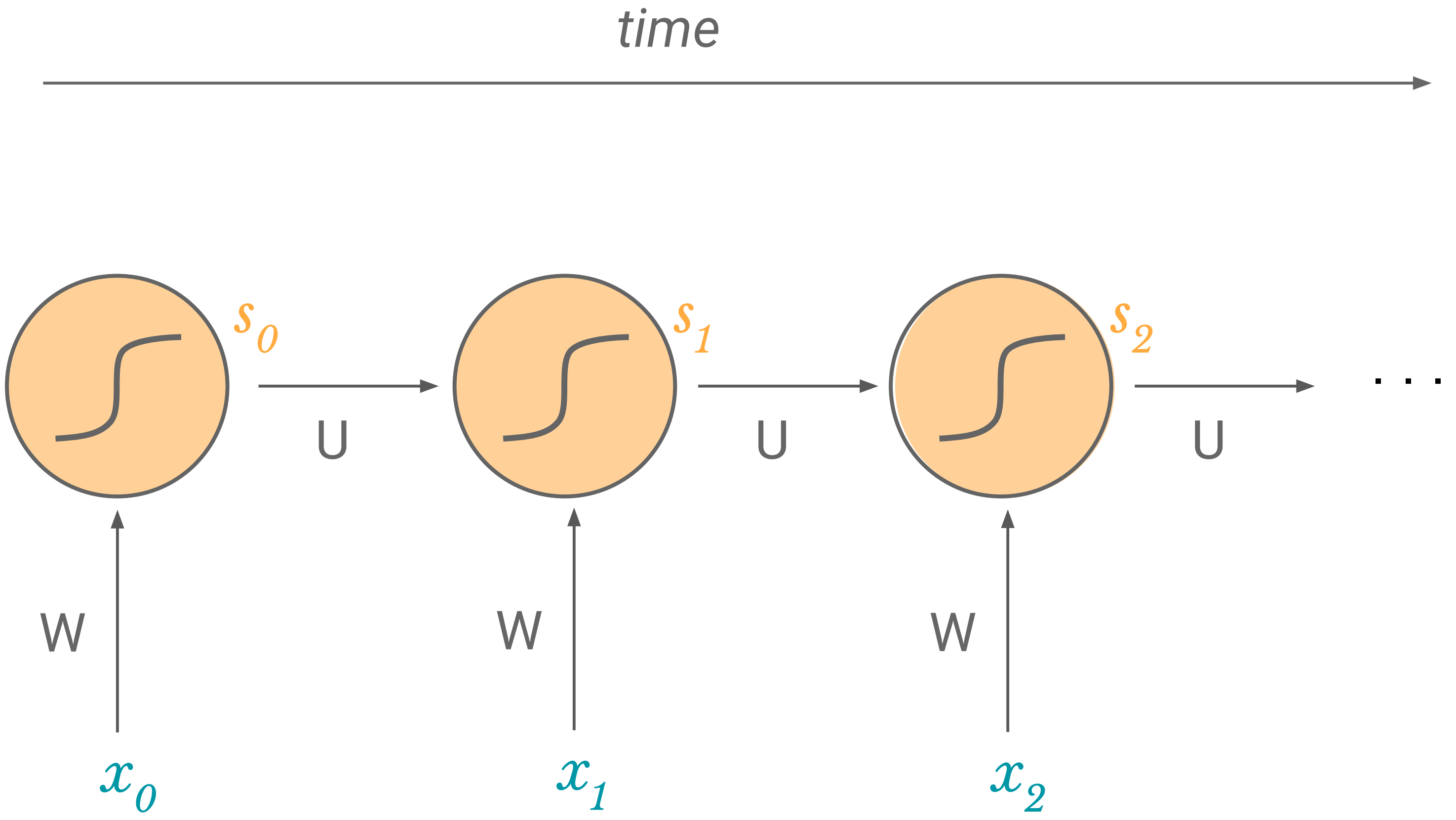
x_1 : vector representing second word
s_1 : cell state at $t = 1$
s_2 : cell state at $t = 2$

$$s_2 = tanh(Wx_1 + Us_1)$$

$W,\ U$: weight matrices

Sample RNN

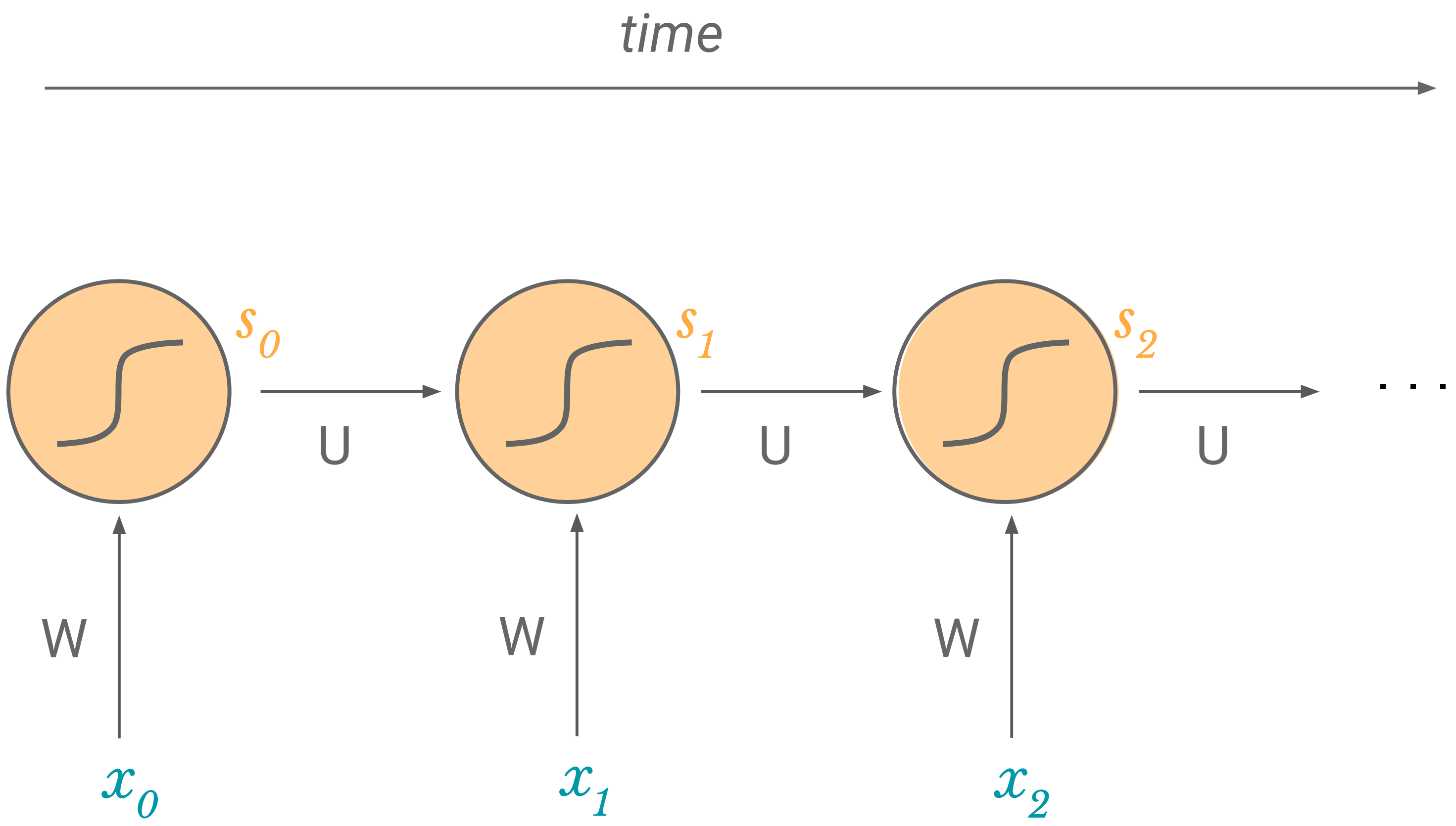
- "unfolding" the RNN across time:



notice that W and U stay the same!

Sample RNN

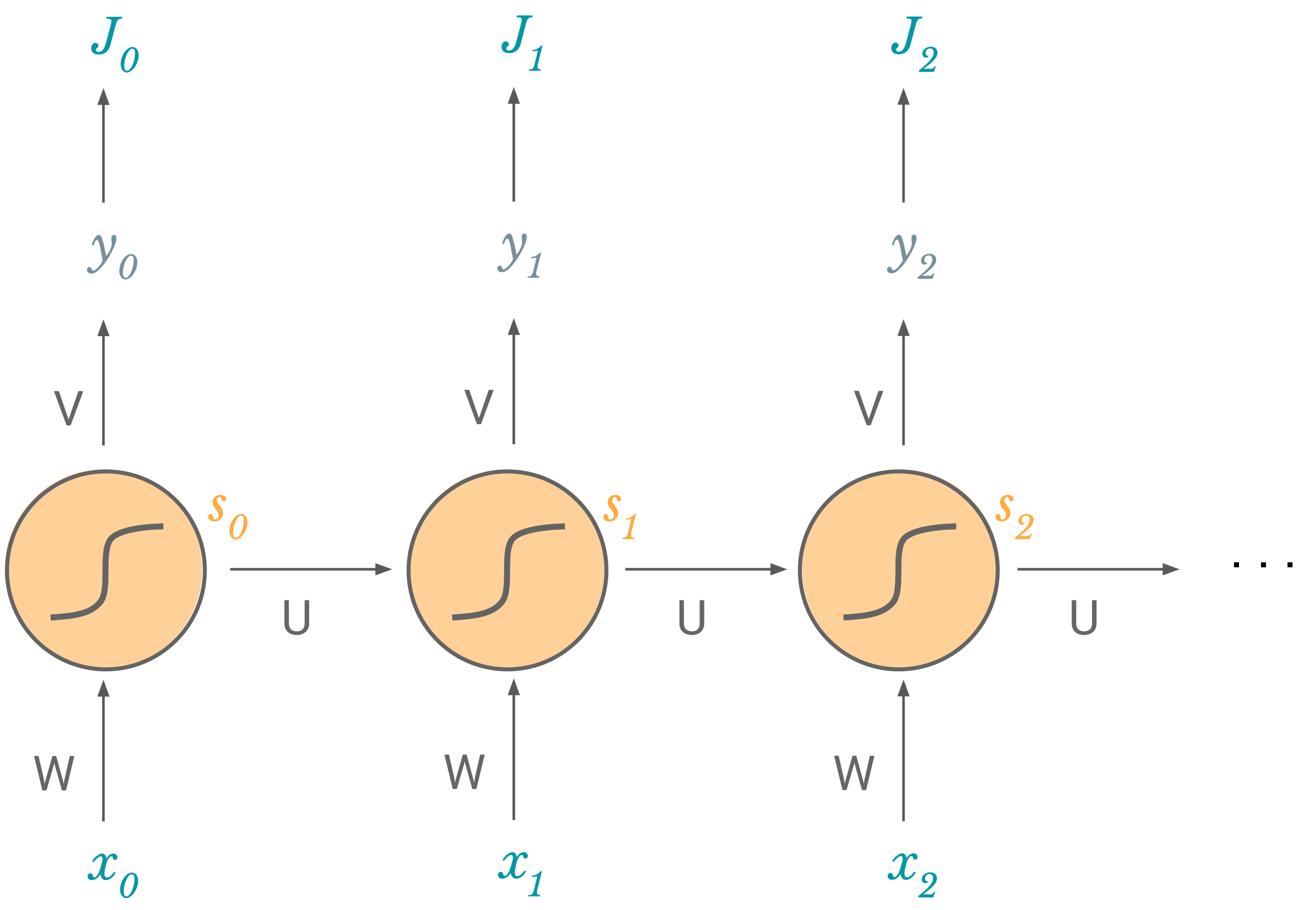
- "unfolding" the RNN across time:



s_n can contain information from all past timesteps

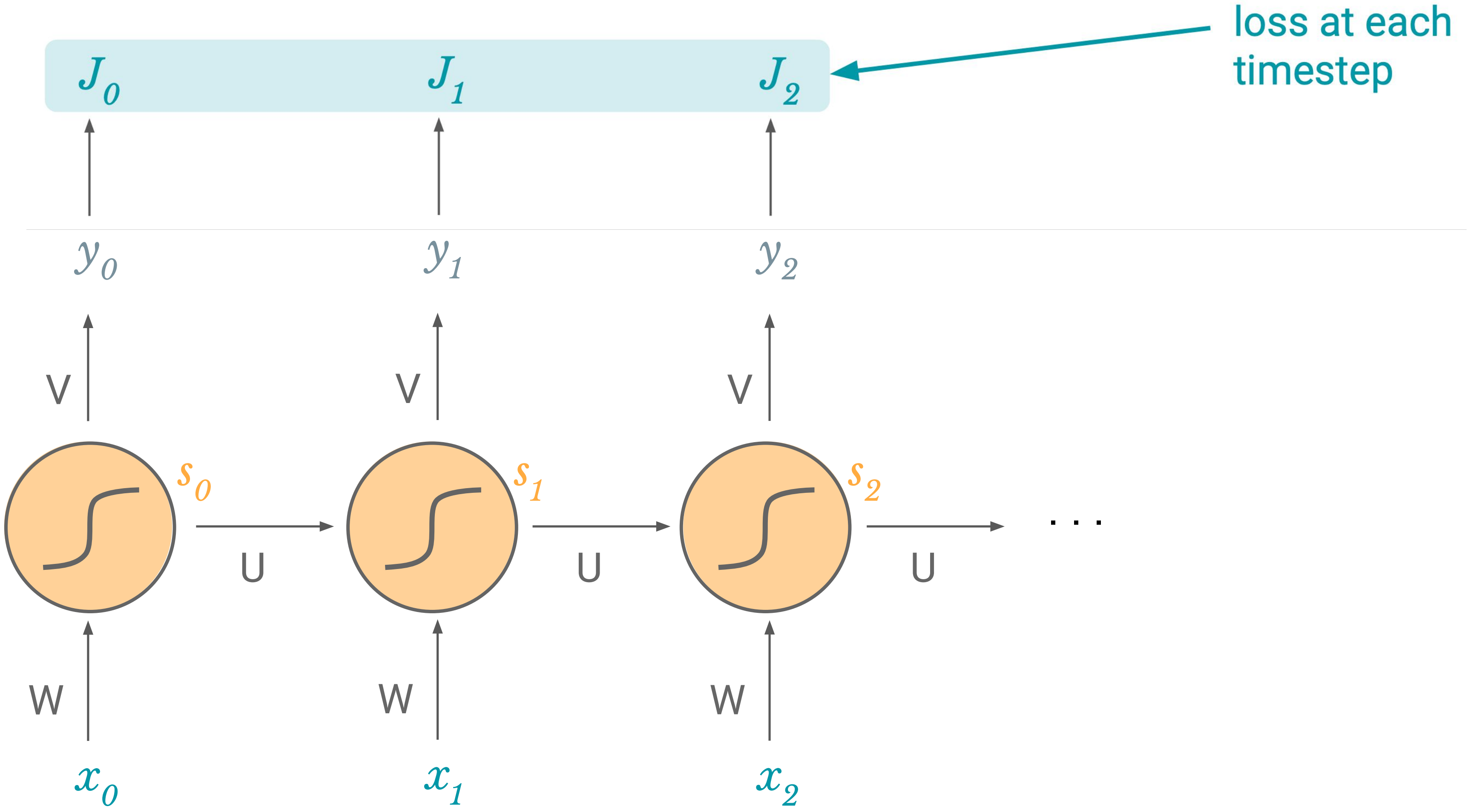
We have a **loss at each timestep**:

(since we're making a prediction at each timestep)

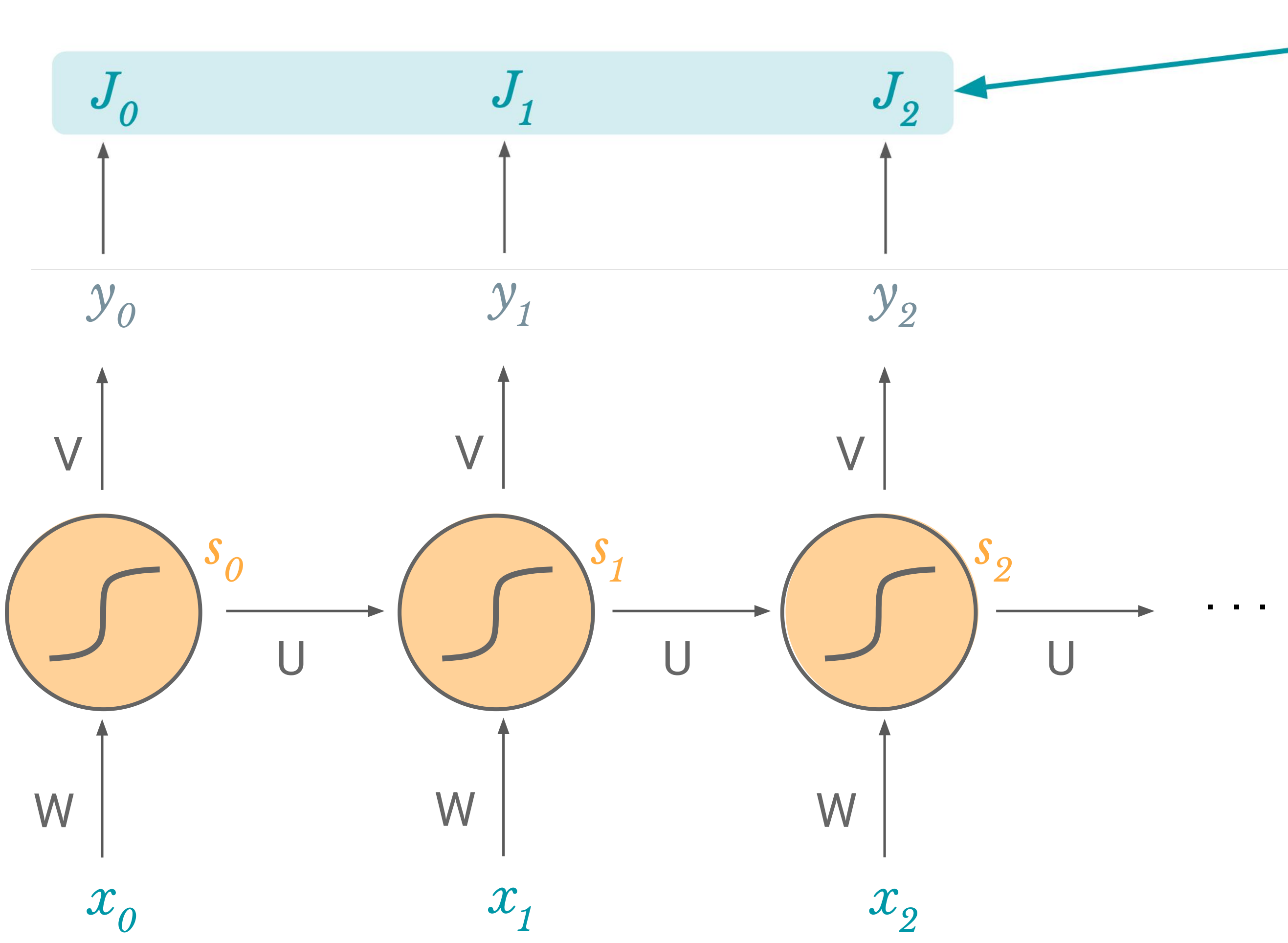


We have a **loss at each timestep**:

(since we're making a prediction at each timestep)



We **sum the losses** across time:

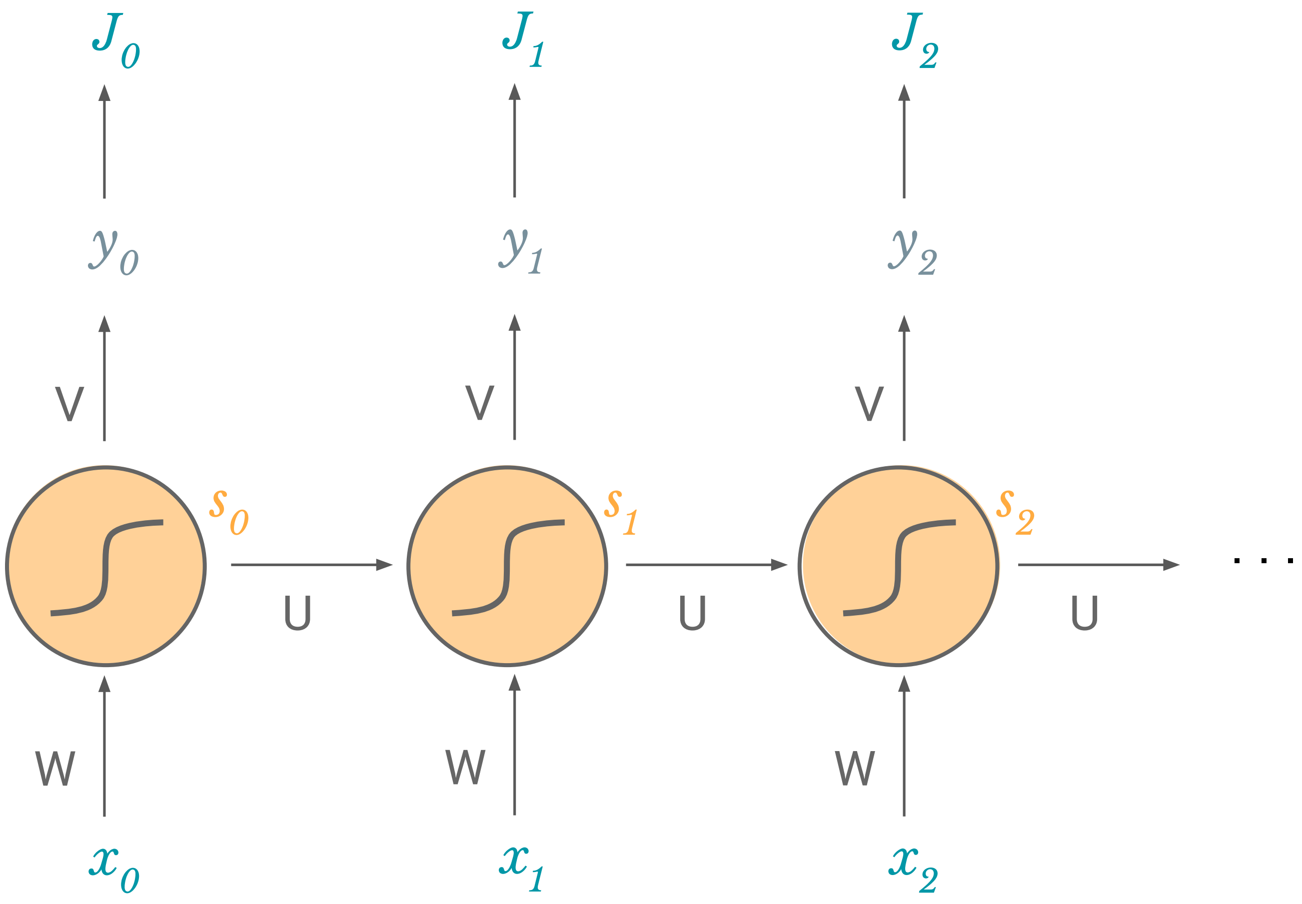


Loss at time t: $J_t(\Theta)$

Total loss:

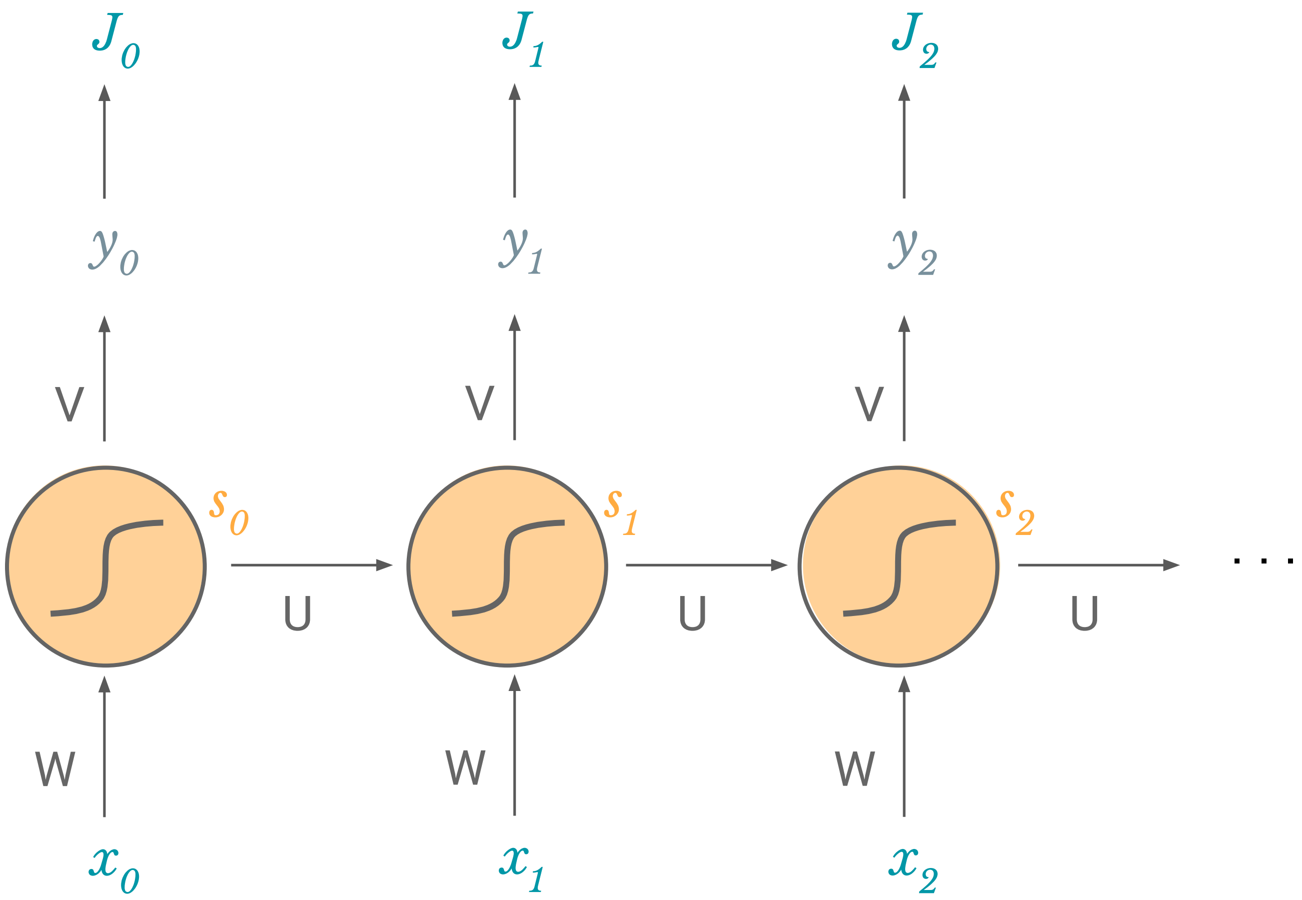
$$J(\Theta) = \sum_t J_t(\Theta)$$

Let's try it our for W with the **chain rule:**



$$\frac{\partial J}{\partial W} = \sum_t \frac{\partial J_t}{\partial W}$$

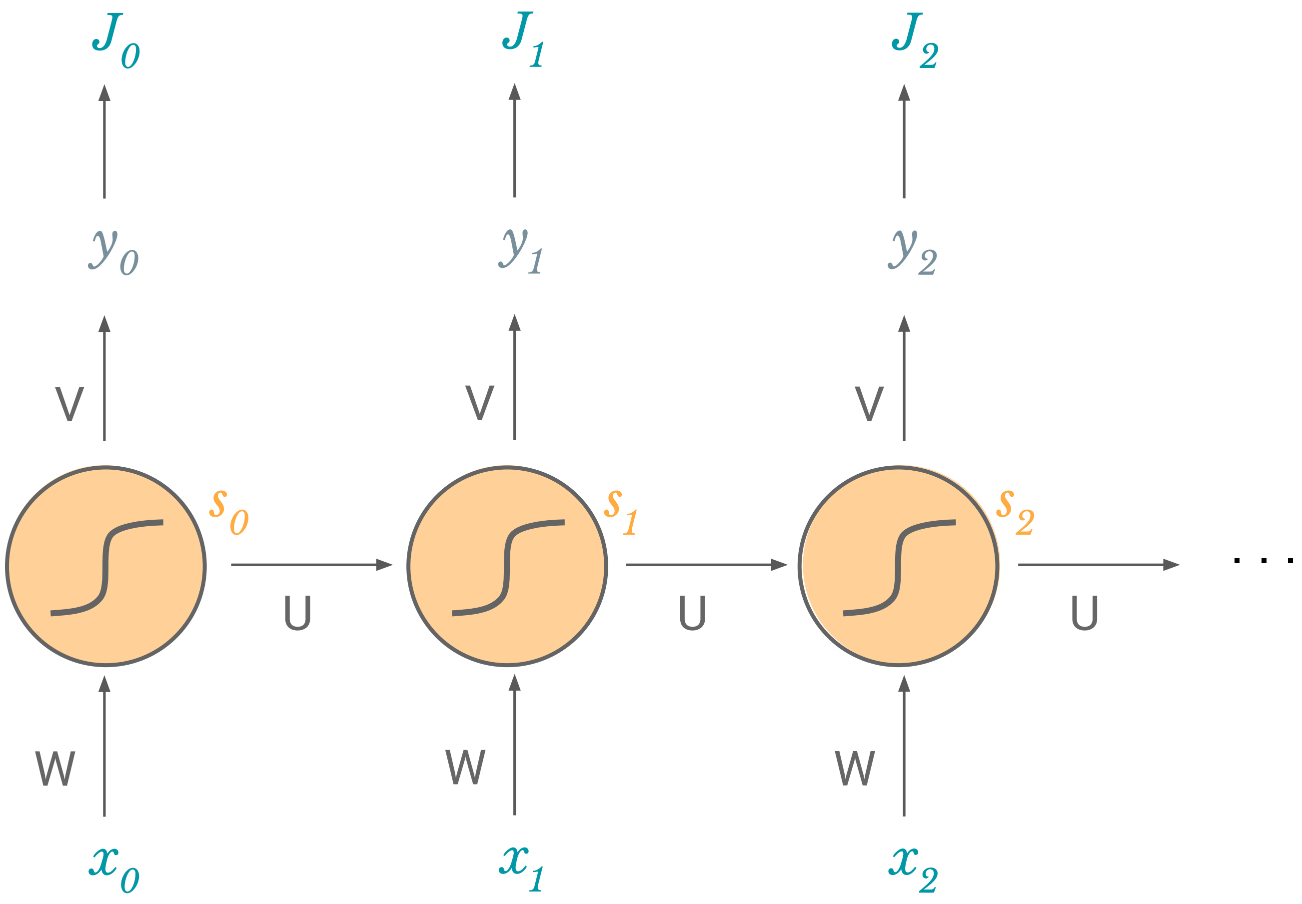
Let's try it our for W with the **chain rule:**



$$\frac{\partial J}{\partial W} = \sum_t \frac{\partial J_t}{\partial W}$$

so let's take a single timestep t:

Let's try it our for W with the **chain rule:**

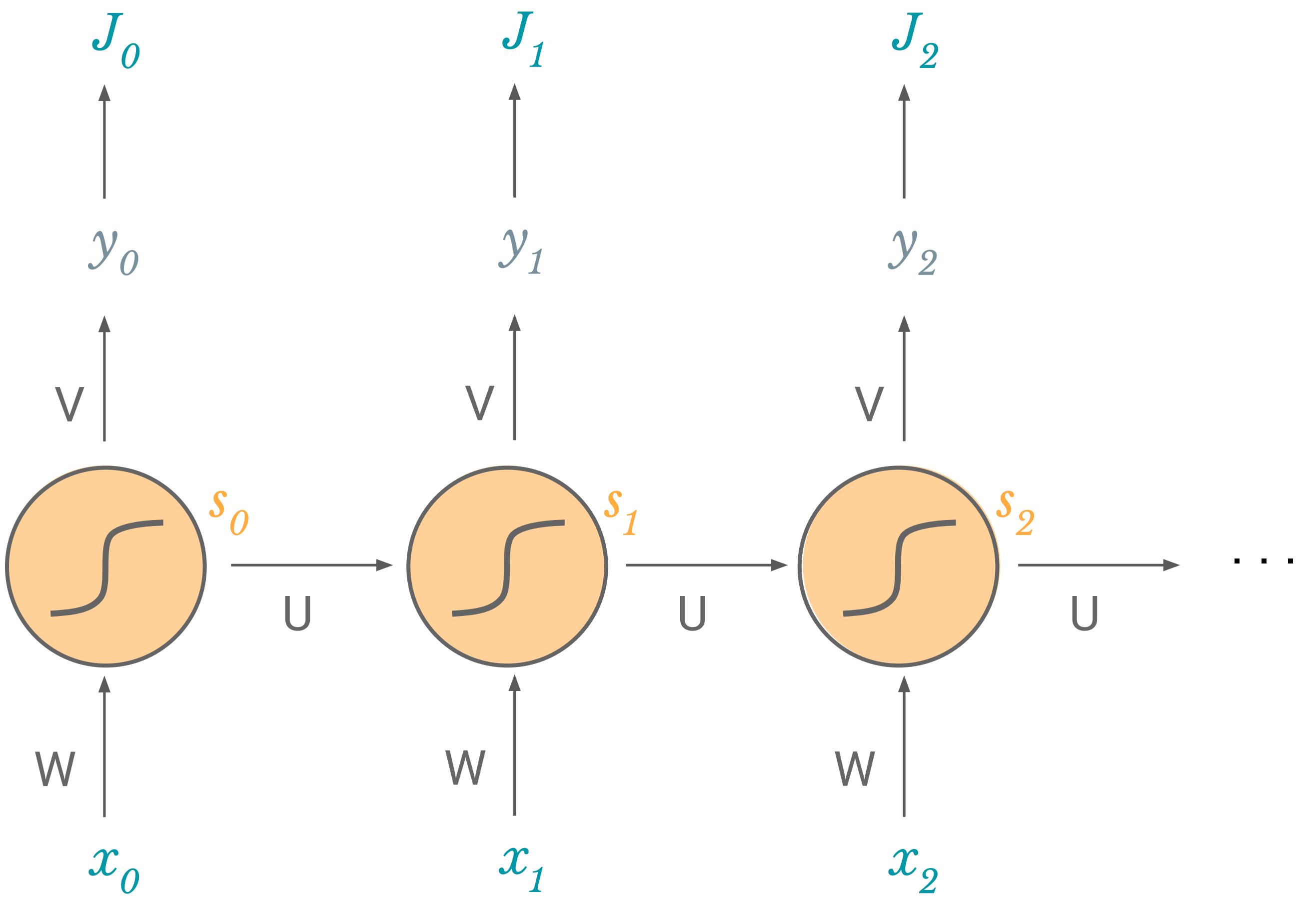


$$\frac{\partial J}{\partial W} = \sum_t \frac{\partial J_t}{\partial W}$$

so let's take a single timestep t:

$$\frac{\partial J_2}{\partial W}$$

Let's try it our for W with the **chain rule:**

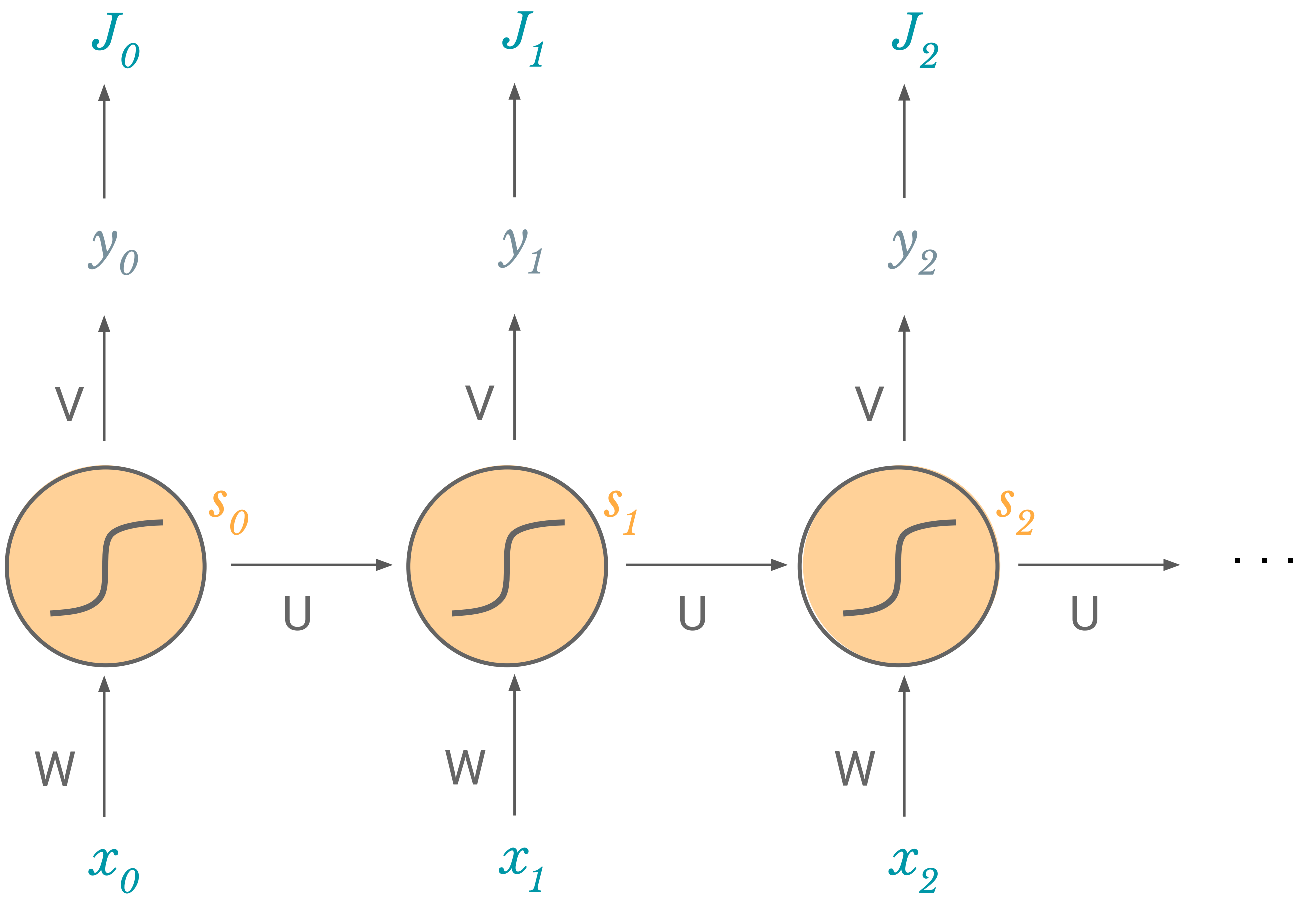


$$\frac{\partial J}{\partial W} = \sum_t \frac{\partial J_t}{\partial W}$$

so let's take a single timestep t:

$$\frac{\partial J_2}{\partial W} = \frac{\partial J_2}{\partial y2}$$

Let's try it our for W with the **chain rule:**

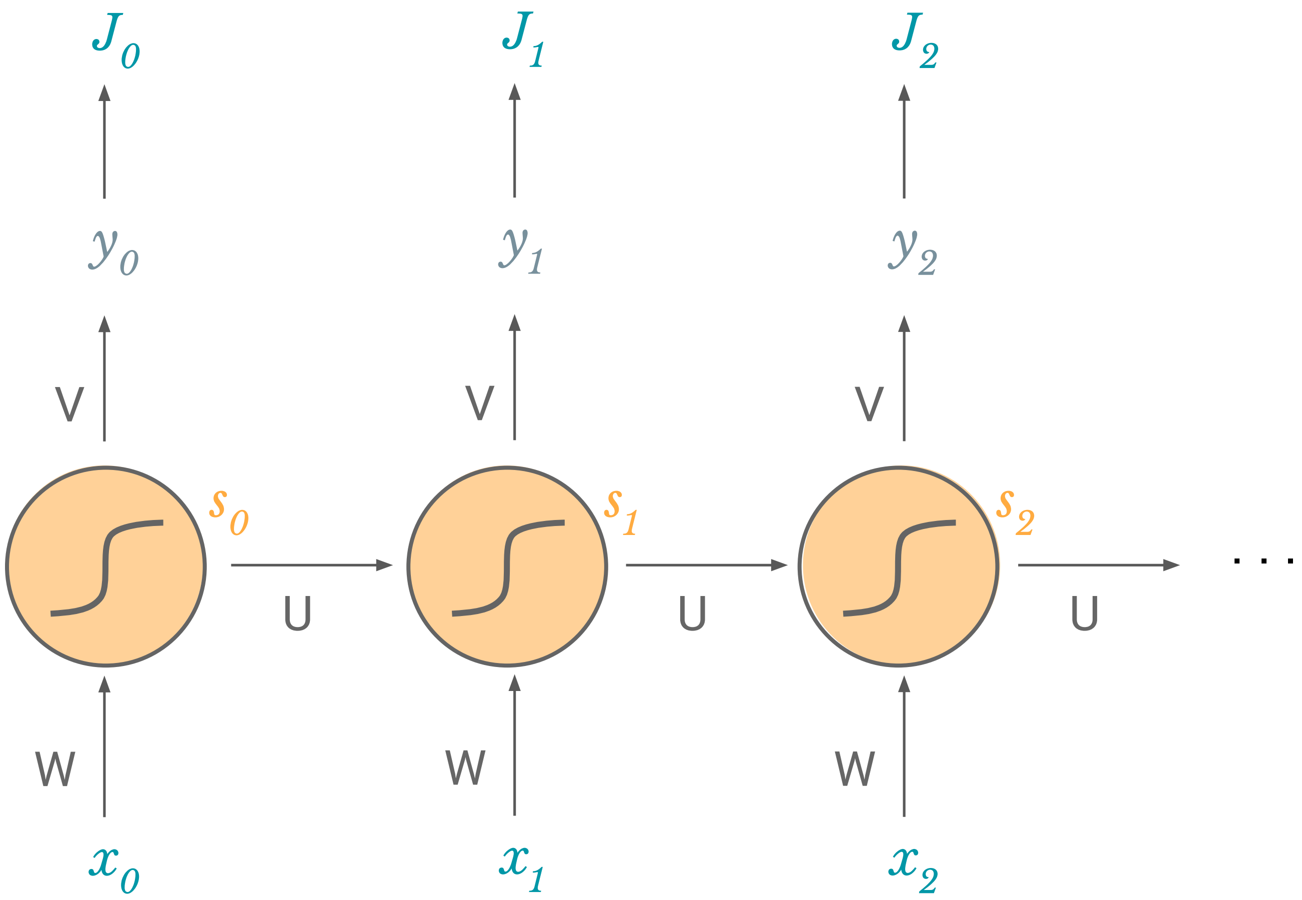


$$\frac{\partial J}{\partial W} = \sum_t \frac{\partial J_t}{\partial W}$$

so let's take a single timestep t:

$$\frac{\partial J_2}{\partial W} = \frac{\partial J_2}{\partial y2}\frac{\partial y_2}{\partial s_2}$$

Let's try it our for W with the **chain rule:**

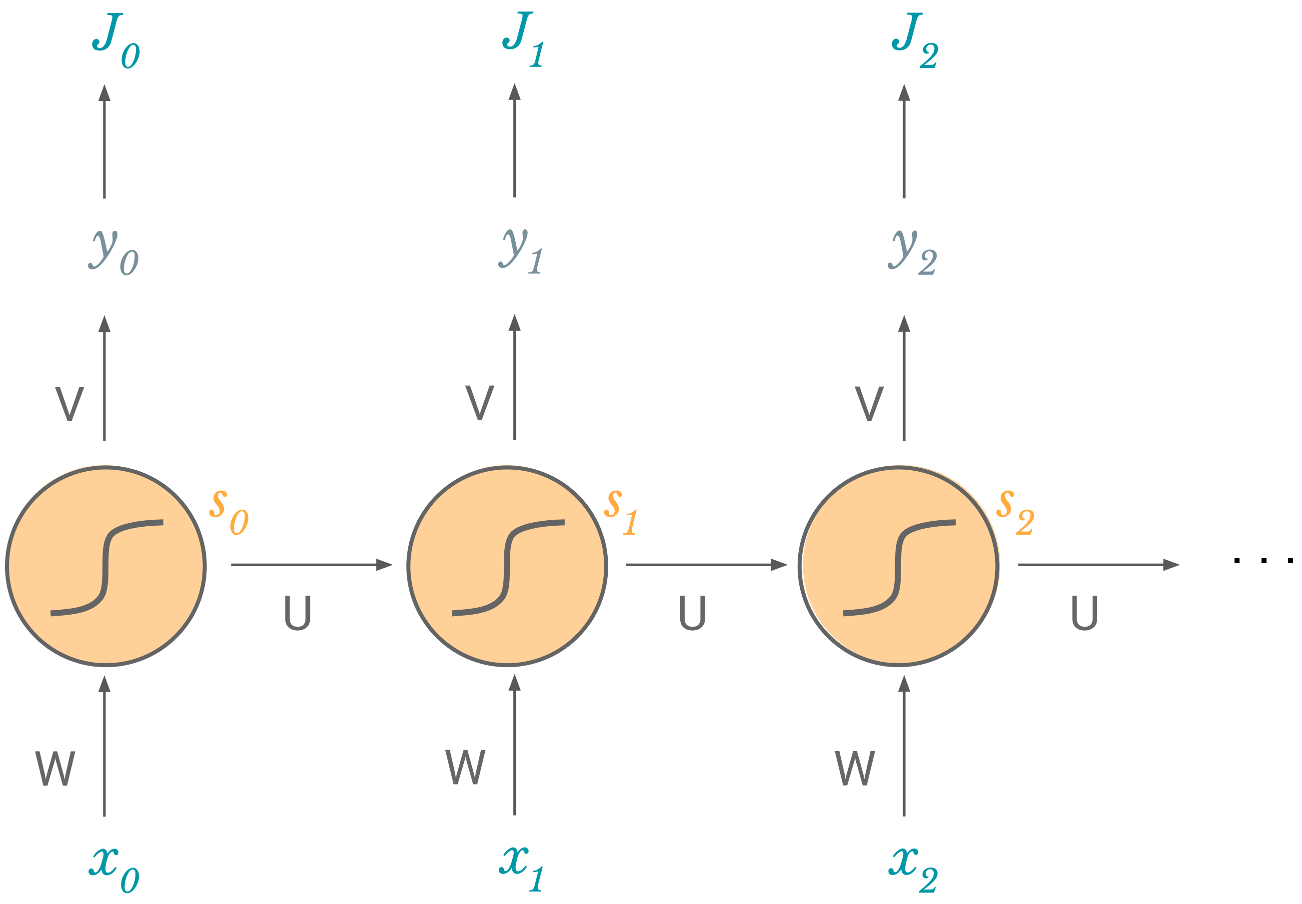


$$\frac{\partial J}{\partial W} = \sum_t \frac{\partial J_t}{\partial W}$$

so let's take a single timestep t:

$$\frac{\partial J_2}{\partial W} = \frac{\partial J_2}{\partial y2} \frac{\partial y_2}{\partial s_2} \frac{\partial s_2}{\partial W}$$

Let's try it our for W with the **chain rule:**



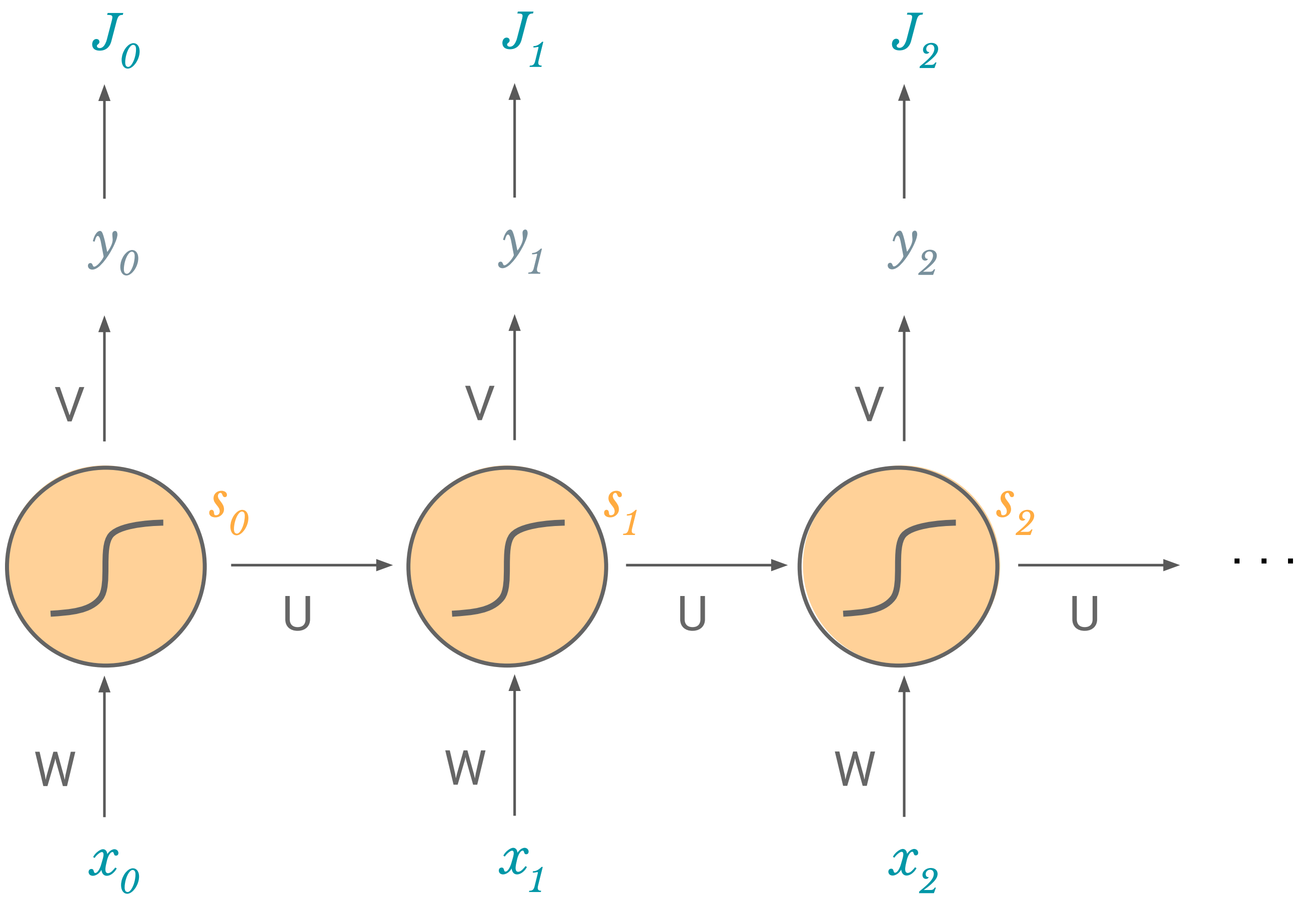
$$\frac{\partial J}{\partial W} = \sum_t \frac{\partial J_t}{\partial W}$$

so let's take a single timestep t:

$$\frac{\partial J_2}{\partial W} = \frac{\partial J_2}{\partial y2}\frac{\partial y_2}{\partial s_2}\boxed{\frac{\partial s_2}{\partial W}}$$

but wait…

Let's try it our for W with the **chain rule:**



$$\frac{\partial J}{\partial W} = \sum_t \frac{\partial J_t}{\partial W}$$

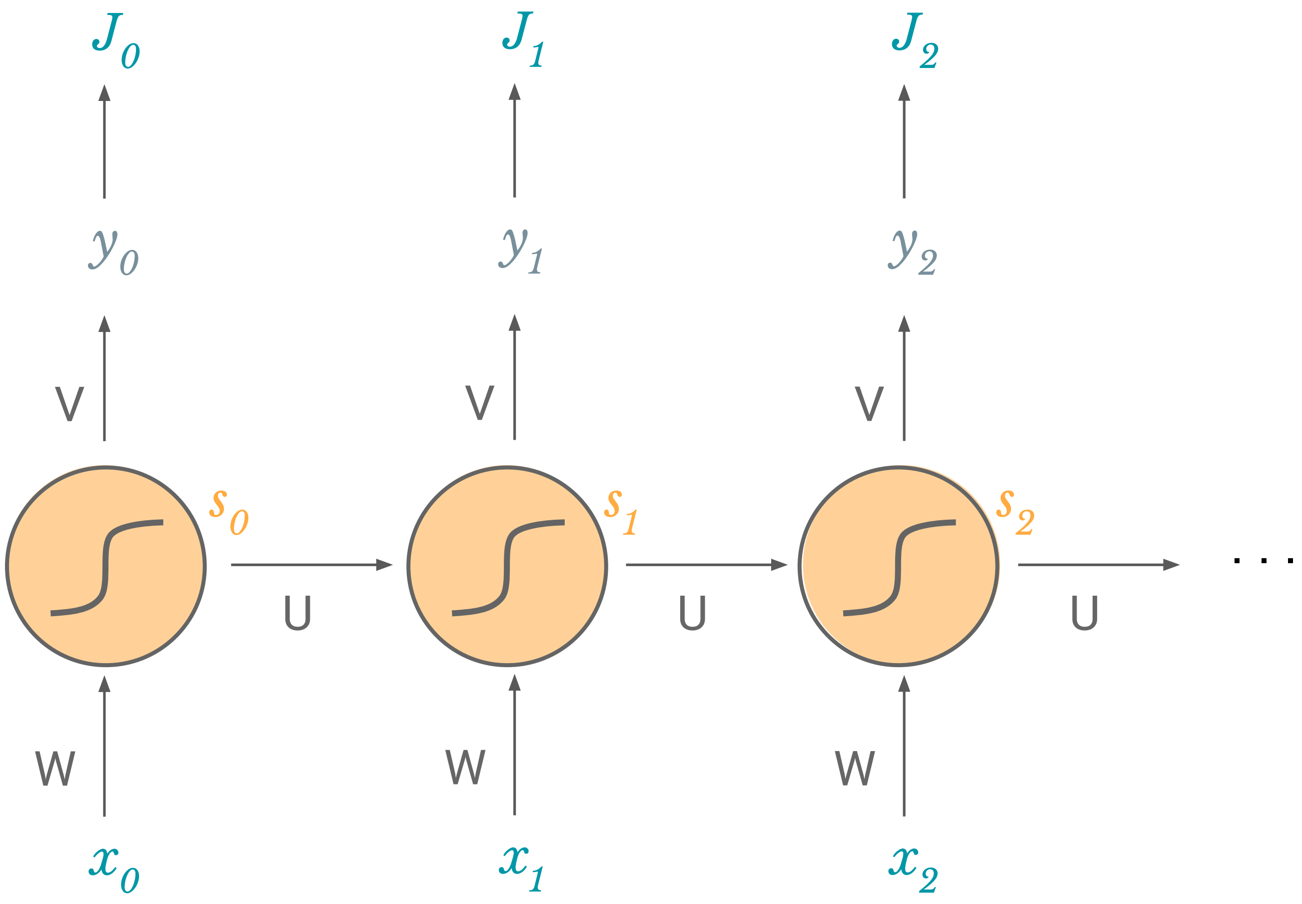
so let's take a single timestep t:

$$\frac{\partial J_2}{\partial W} = \frac{\partial J_2}{\partial y2} \frac{\partial y_2}{\partial s_2} \boxed{\frac{\partial s_2}{\partial W}}$$

but wait…

$$s_2 = tanh(Us_1 + Wx_2)$$

Let's try it our for W with the **chain rule:**



$$\frac{\partial J}{\partial W} = \sum_t \frac{\partial J_t}{\partial W}$$

so let's take a single timestep t:

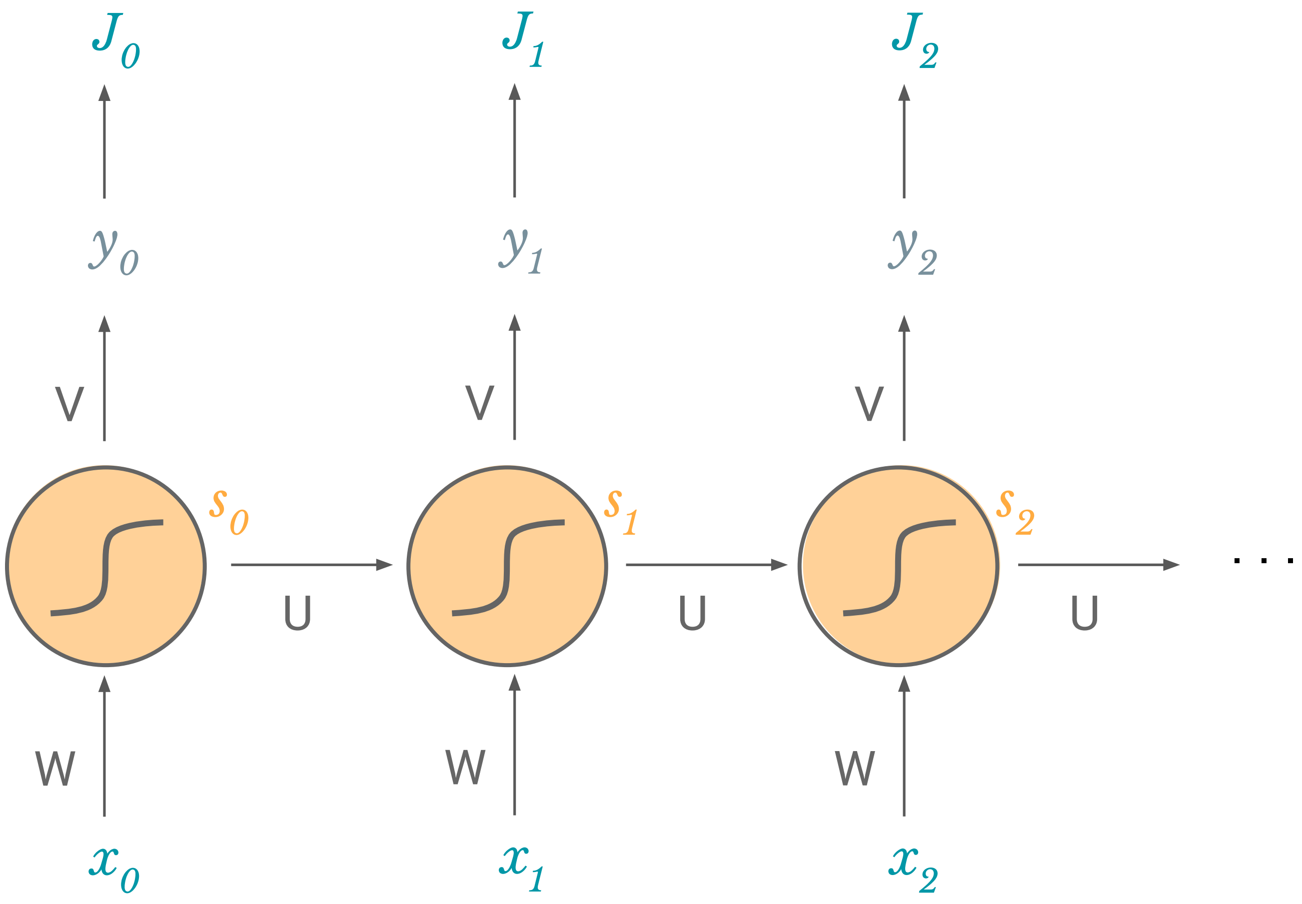
$$\frac{\partial J_2}{\partial W} = \frac{\partial J_2}{\partial y2}\frac{\partial y_2}{\partial s_2}\boxed{\frac{\partial s_2}{\partial W}}$$

but wait…

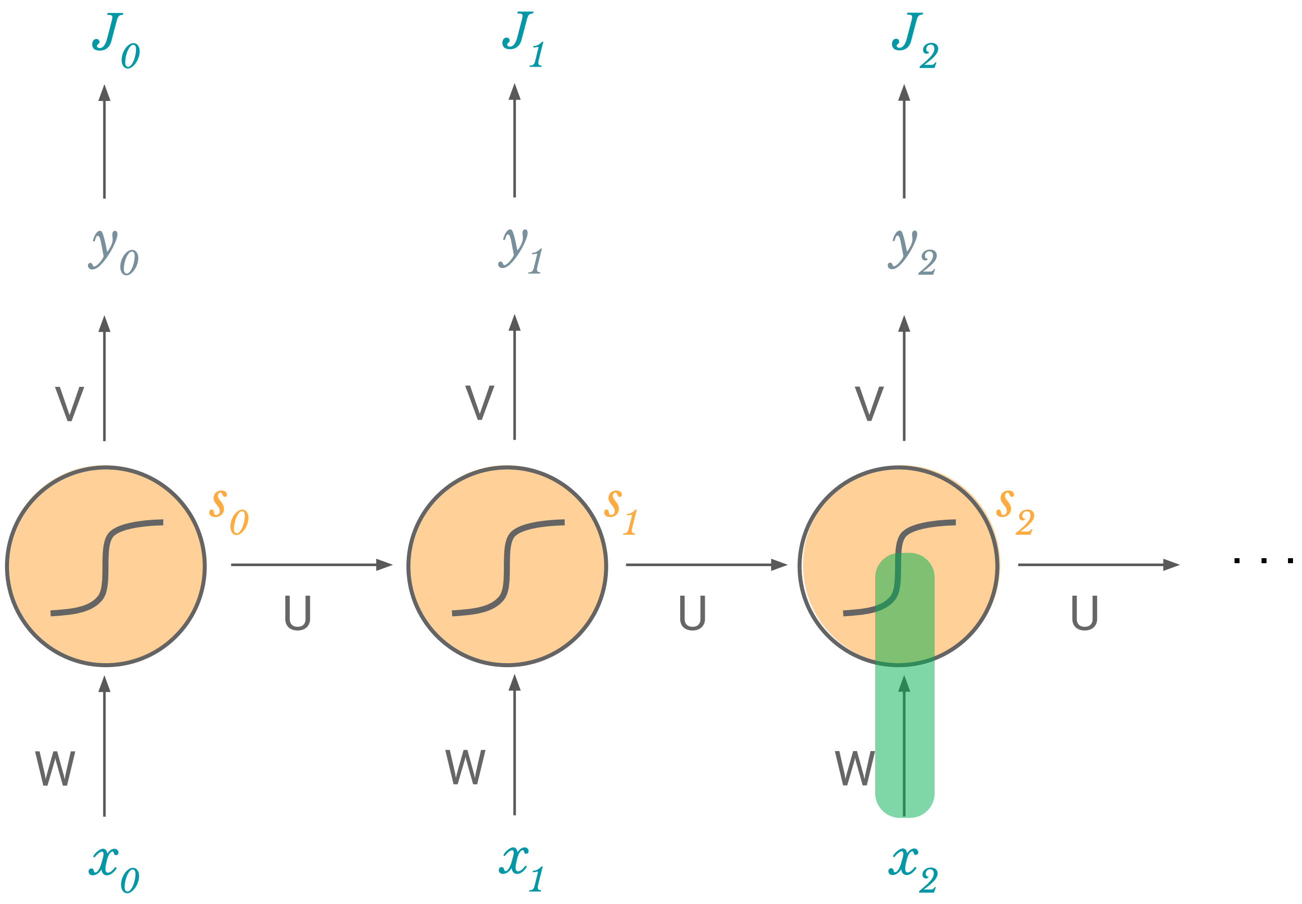
$$s_2 = tanh(U\boxed{s_1} + Wx_2)$$

s_1 also depends on W so we can't just treat $\frac{\partial s_2}{\partial W}$ as a constant!

Let's try it our for W with the **chain rule:**

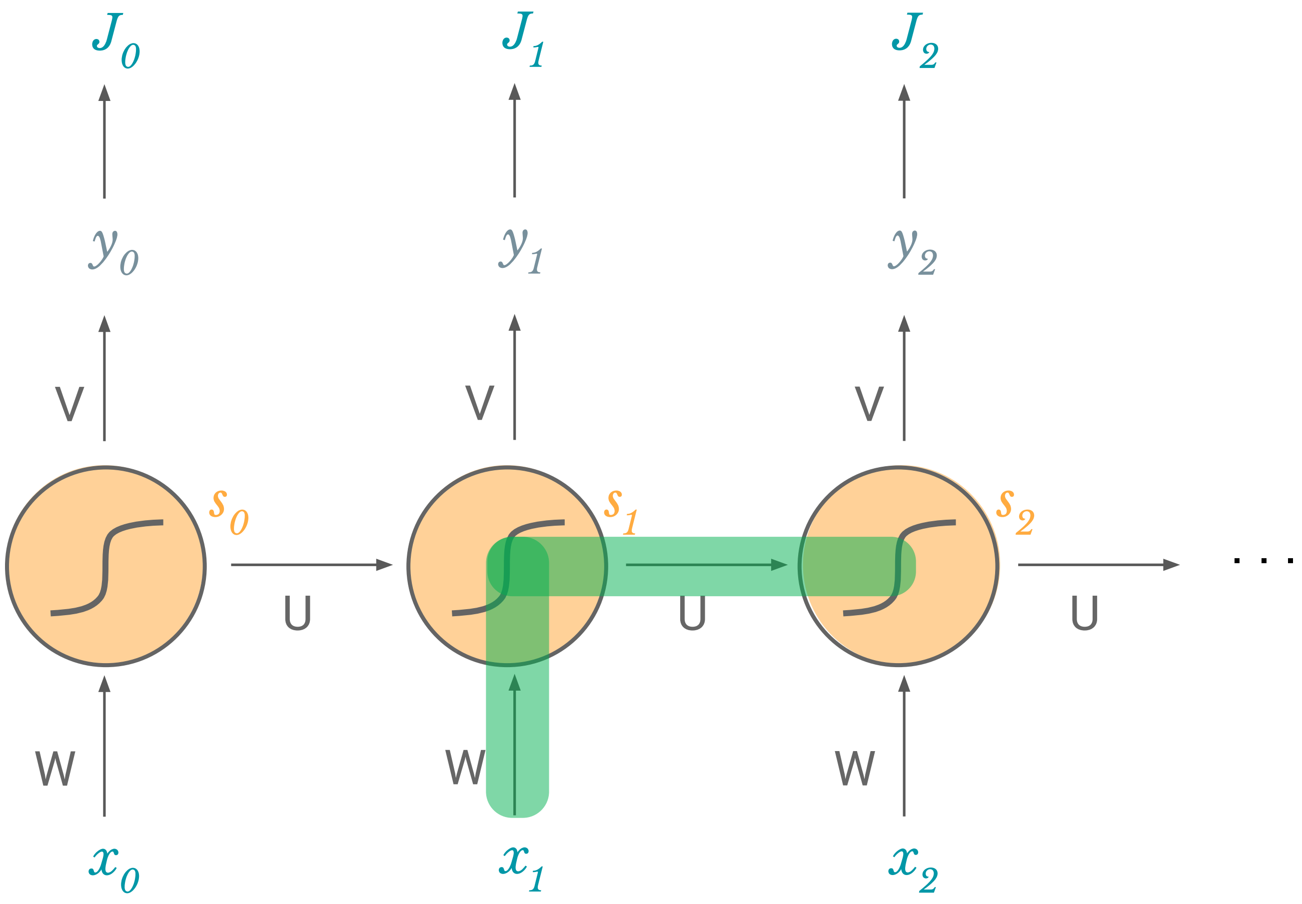


Let's try it our for W with the **chain rule:**



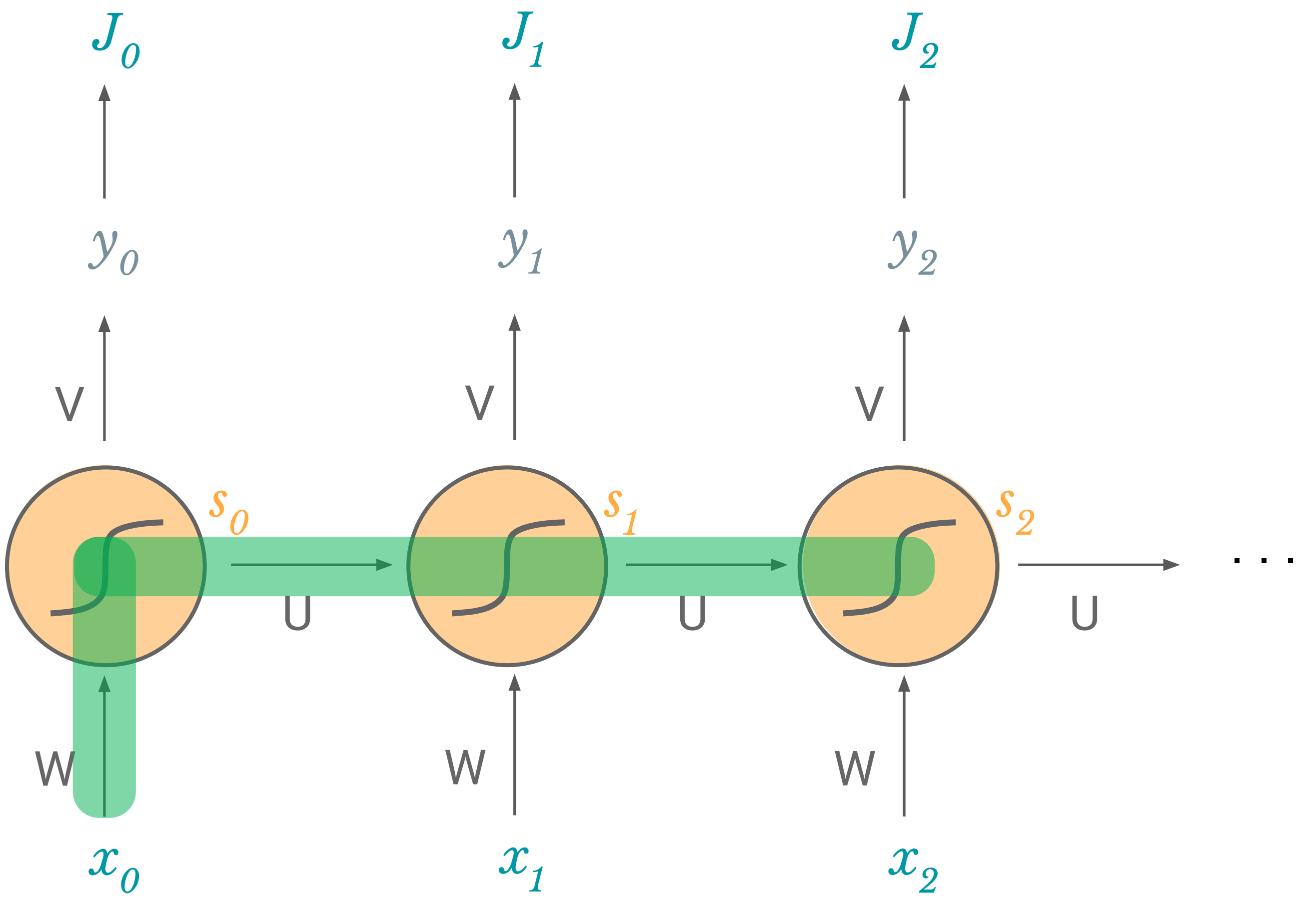
$$\frac{\partial s_2}{\partial W}$$

Let's try it our for W with the **chain rule:**



$$\frac{\partial s_2}{\partial W} + \frac{\partial s_2}{\partial s_1}\frac{\partial s_1}{\partial W}$$

Let's try it our for W with the **chain rule:**



$$\frac{\partial s_2}{\partial W} + \frac{\partial s_2}{\partial s_1}\frac{\partial s_1}{\partial W} + \frac{\partial s_2}{\partial s_0}\frac{\partial s_0}{\partial W}$$

Backpropagation through time:

$$\frac{\partial J_2}{\partial W} = \sum_{k=0}^{2} \frac{\partial J_2}{\partial y_2} \frac{\partial y_2}{\partial s_2} \underbrace{\frac{\partial s_2}{\partial s_k} \frac{\partial s_k}{\partial W}}$$

Contributions of W in previous timesteps to the error at timestep t

Backpropagation through time:

$$\frac{\partial J_t}{\partial W} = \sum_{k=0}^{t} \frac{\partial J_t}{\partial y_t} \frac{\partial y_t}{\partial s_t} \underbrace{\frac{\partial s_t}{\partial s_k} \frac{\partial s_k}{\partial W}}$$

Contributions of W in previous timesteps to the error at timestep t

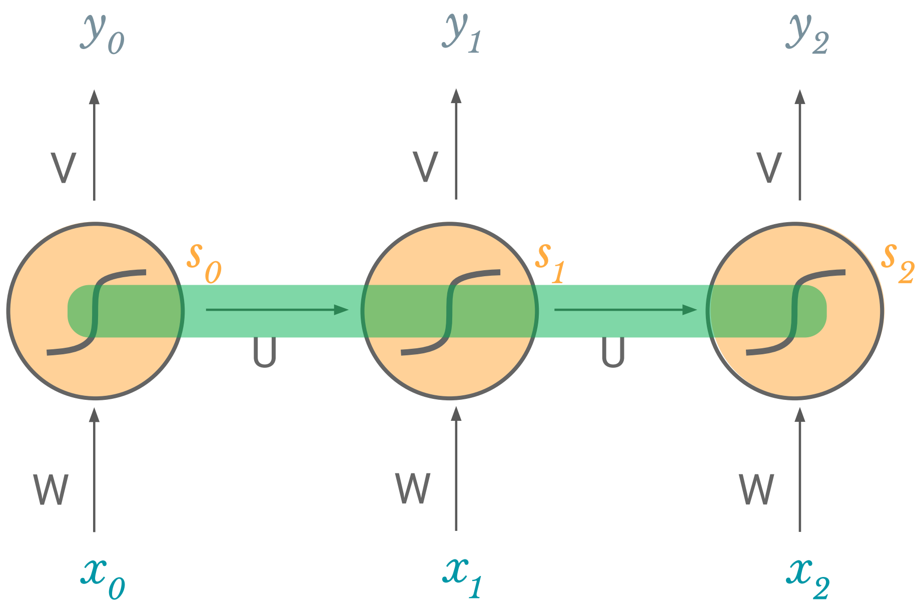
Why are RNNs hard to train?

Vanishing Gradient Problem

$$\frac{\partial J_2}{\partial W} = \sum_{k=0}^{2} \frac{\partial J_2}{\partial y_2} \frac{\partial y_2}{\partial s_2} \boxed{\frac{\partial s_2}{\partial s_k}} \frac{\partial s_k}{\partial W}$$

Vanishing Gradient Problem

$$\frac{\partial J_2}{\partial W} = \sum_{k=0}^{2} \frac{\partial J_2}{\partial y_2} \frac{\partial y_2}{\partial s_2} \boxed{\frac{\partial s_2}{\partial s_k}} \frac{\partial s_k}{\partial W}$$



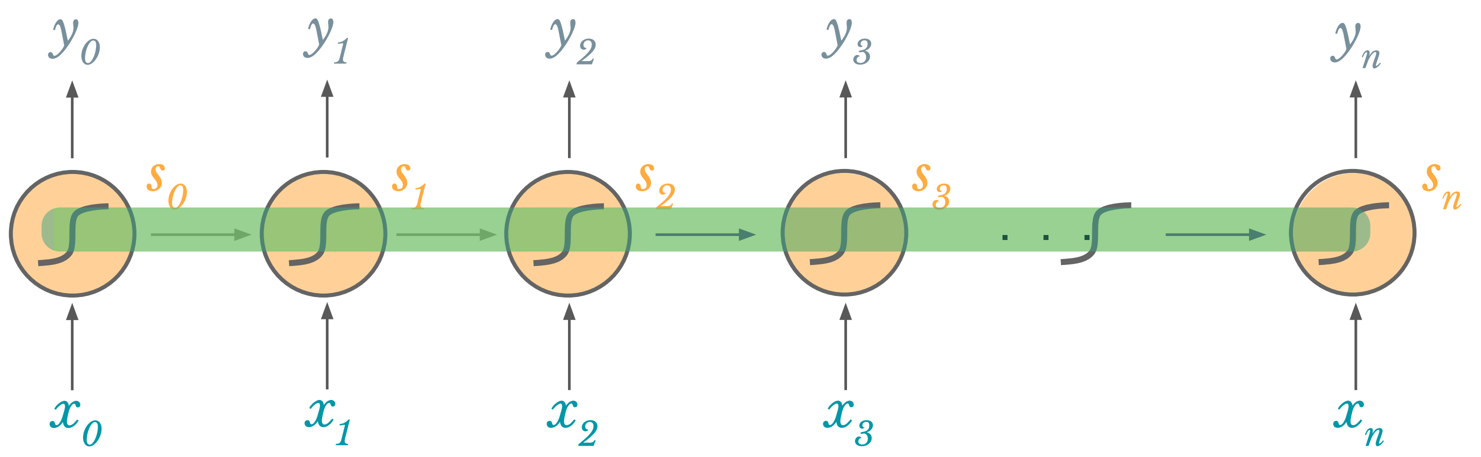
at k=0: $\boxed{\frac{\partial s_2}{\partial s_0} = \frac{\partial s_2}{\partial s_1} \frac{\partial s_1}{\partial s_0}}$

Vanishing Gradient Problem

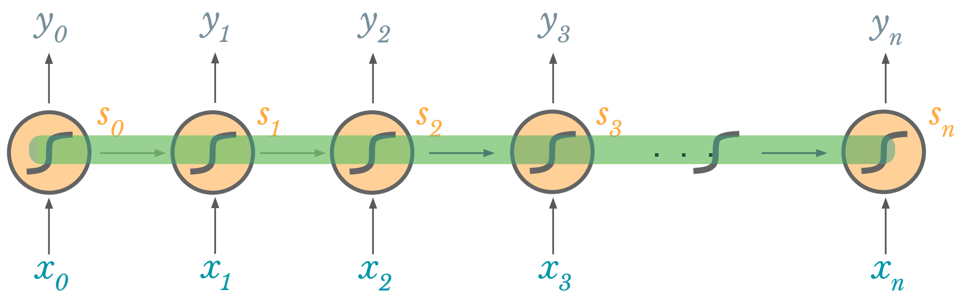
$$\frac{\partial J_n}{\partial W} = \sum_{k=0}^{n} \frac{\partial J_n}{\partial y_n} \frac{\partial y_n}{\partial s_n} \boxed{\frac{\partial s_n}{\partial s_k}} \frac{\partial s_k}{\partial W}$$

$$\boxed{\frac{\partial s_n}{\partial s_{n-1}} \frac{\partial s_{n-1}}{\partial s_{n-2}} \cdots \frac{\partial s_3}{\partial s_2} \frac{\partial s_2}{\partial s_1} \frac{\partial s_1}{\partial s_0}}$$

as the gap between timesteps gets bigger, this product gets longer and longer!



Vanishing Gradient Problem



what are each of these terms?

$$\frac{\partial s_n}{\partial s_{n-1}} \frac{\partial s_{n-1}}{\partial s_{n-2}} \cdots \frac{\partial s_3}{\partial s_2} \frac{\partial s_2}{\partial s_1} \frac{\partial s_1}{\partial s_0}$$

$$\frac{\partial s_n}{\partial s_{n-1}} = W^T diag\left[f'(W_{s_{j-1}+Ux_j})\right]$$

W = sampled from standard normal distribution = mostly < 1

f = tanh or sigmoid so f' < 1

we're multiplying a lot of **small numbers** together.

Vanishing Gradient Problem

we’re multiplying a lot of **small numbers** together.

so what?

errors due to further back timesteps have increasingly **smaller gradients**.

so what?

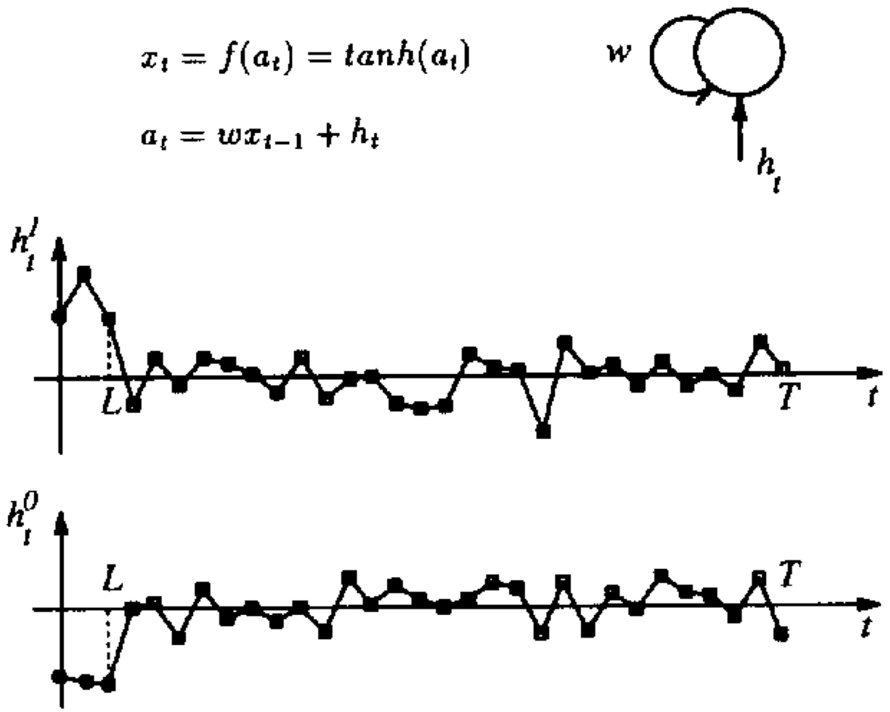
parameters become biased to **capture shorter-term** dependencies.

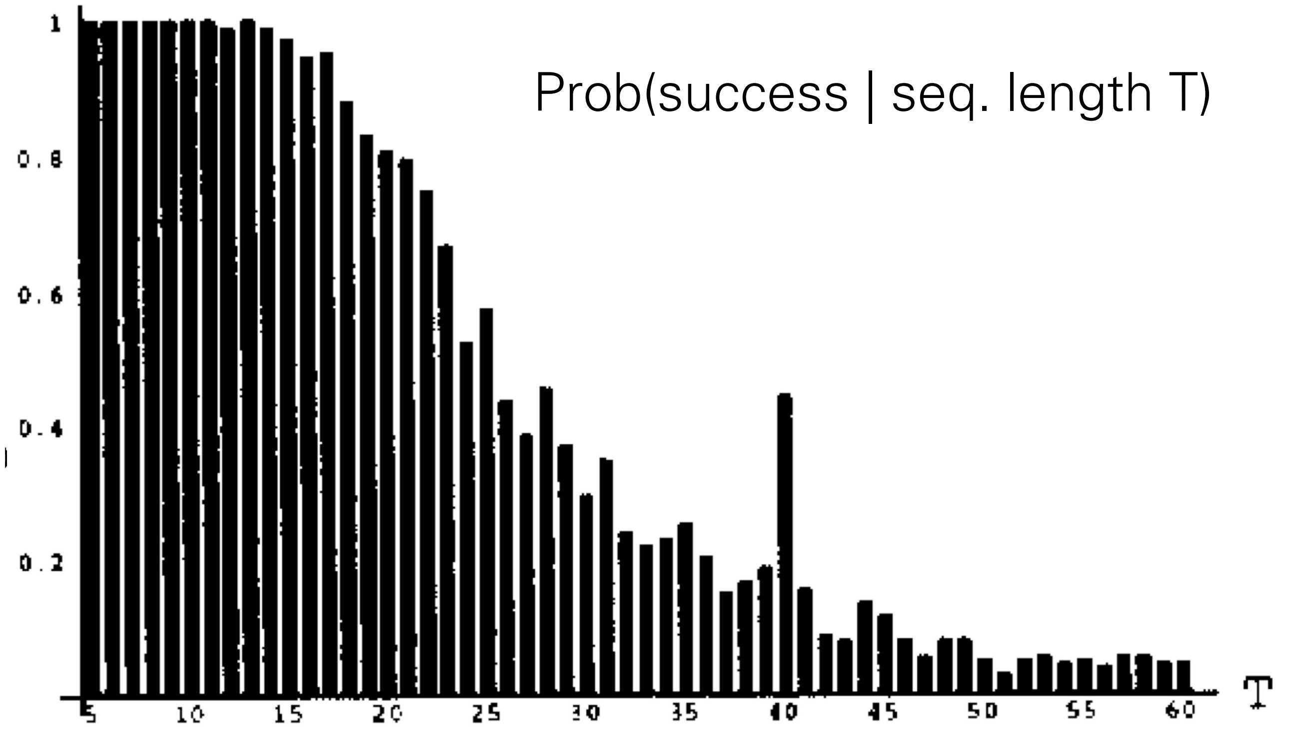
A Toy Example

- 2 categories of sequences
- Can the single tanh unit learn to store for T time steps 1 bit of information given by the sign of initial input?

$$x_t = f(a_t) = tanh(a_t)$$

$$a_t = wx_{t-1} + h_t$$





Vanishing Gradient Problem

"In France, I had a great time and I learnt some of the _____ language."

our parameters are not trained to capture long-term dependencies, so the word we predict will mostly depend on the previous few words, not much earlier ones

Long-Term Dependencies

- The RNN gradient is a product of Jacobian matrices, each associated with a step in the forward computation. To store information robustly in a finite-dimensional state, the dynamics must be contractive [Bengio et al 1994].

$$L = L(s_T(s_{T-1}(\ldots s_{t+1}(s_t, \ldots))))$$
$$\frac{\partial L}{\partial s_t} = \frac{\partial L}{\partial s_T}\frac{\partial s_T}{\partial s_{T-1}} \cdots \frac{\partial s_{t+1}}{\partial s_t}$$

- Problems:
  - sing. values of Jacobians > 1 → **gradients explode**
  - or sing. values < → **gradients shrink & vanish**
  - or random → **variance grows exponentially**

Slide credit: Yoshua Bengio

RNN Tricks

(Pascanu et al., 2013; Bengio et al., 2013; Gal and Ghahramani, 2016;Morishita et al., 2017)

- Mini-batch creation strategies (efficient computations)
- Clipping gradients (avoid exploding gradients)
- Leaky integration (propagate long-term dependencies)
- Momentum (cheap 2nd order)
- Dropout (avoid overfitting)
- Initialization (start in right ballpark avoids exploding/vanishing)
- Sparse Gradients (symmetry breaking)
- Gradient propagation regularizer (avoid vanishing gradient)
- Gated self-loops (LSTM & GRU, reduces vanishing gradient)

Slide adapted from Yoshua Bengio

Mini-batching in RNNs

- Mini-batching makes things much faster!
- But mini-batching in RNNs is harder than in feed-forward networks
  - Each word depends on the previous word
  - Sequences are of various length
- Padding:

this	is	an	example	</s>
this	is	another	</s>	**</s>**

- If we use sentences of different lengths, too much padding and sorting can **result in decreased performance**
- To remedy this: **sort sentences** so similarly-lengthed seqs are in the same batch

Slide adapted from Graham Neubig

Mini-batching in RNNs

- Many alternatives:
  1. Shuffle the corpus randomly before creating mini-batches (with no sorting).
  2. Sort based on the source sequence length.
  3. Sort based on the target sequence length.
  4. Sort using the source sequence length, break ties by sorting by target sequence length.
  5. Sort using the target sequence length, break ties by sorting by source sequence length.

Algorithm 1 Create mini-batches

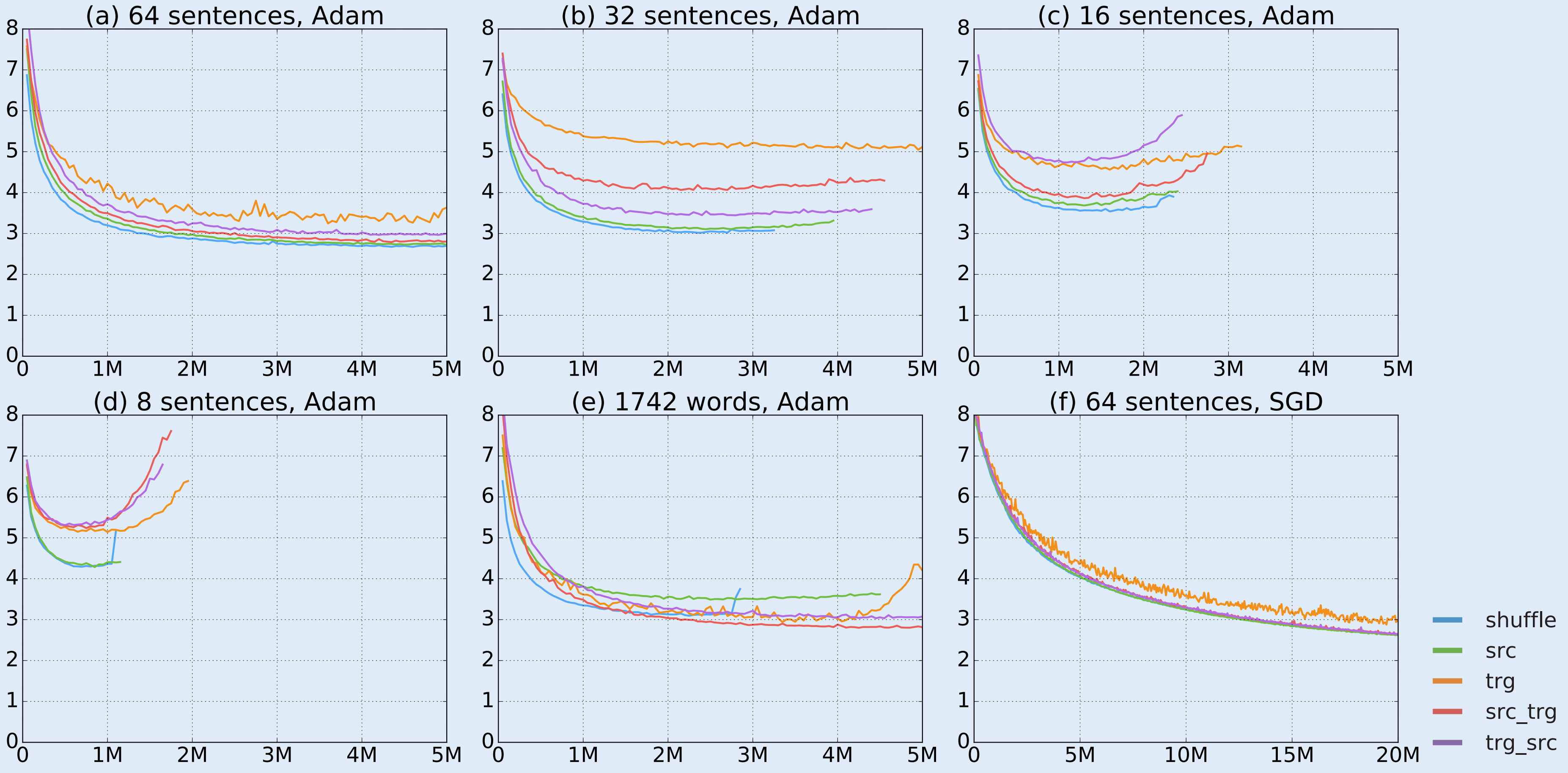
$\boldsymbol{C} \leftarrow$ Training corpus
$\boldsymbol{C} \leftarrow$ sort$(\boldsymbol{C})$ or shuffle$(\boldsymbol{C})$ $\triangleright$ sort or shuffle the whole corpus
$\boldsymbol{B} \leftarrow \{\}$ $\triangleright$ mini-batches
$i \leftarrow 0,\ j \leftarrow 0$
while $i < \boldsymbol{C}$.size() **do**
  $\boldsymbol{B}[j] \leftarrow \boldsymbol{B}[j] + \boldsymbol{C}[i]$
  if $\boldsymbol{B}[j]$.size() $\geq$ max mini-batch size **then**
    $\boldsymbol{B}[j] \leftarrow$ padding$(\boldsymbol{B}[j])$ $\triangleright$ Padding tokens to the longest sentence in the mini-batch
    $j \leftarrow j + 1$
  end if
  $i \leftarrow i + 1$
end while
$\boldsymbol{B} \leftarrow$ shuffle$(\boldsymbol{B})$ $\triangleright$ shuffle the order of the mini-batches

M. Morishita, Y. Oda, G. Neubig, K. Yoshino, K. Sudoh, and S. Nakamura. **"An Empirical Study of Mini-Batch Creation Strategies for Neural Machine Translation"**. 1st Workshop on NMT 2017

Mini-batching in RNNs

- Many

1. Sh
   mi
2. So
3. So
4. So
   tie
5. So
   tie



- **May affect performance!**

rt or shuffle

nini-batches

ch size **then**

▷

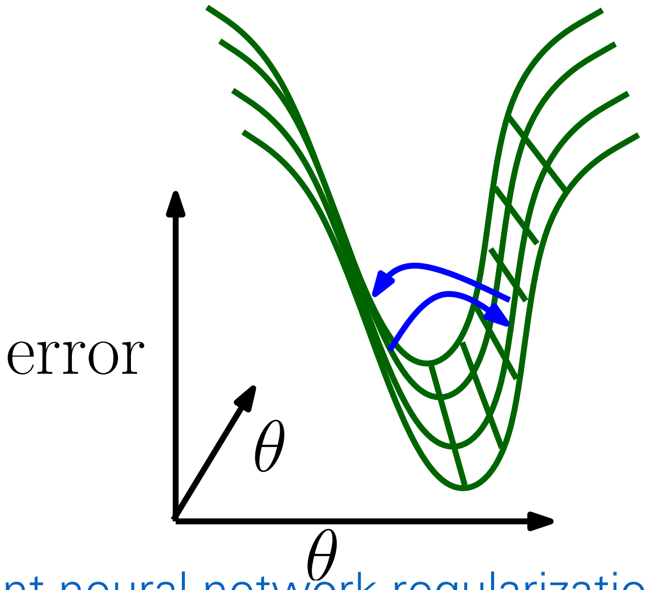
ntence in the

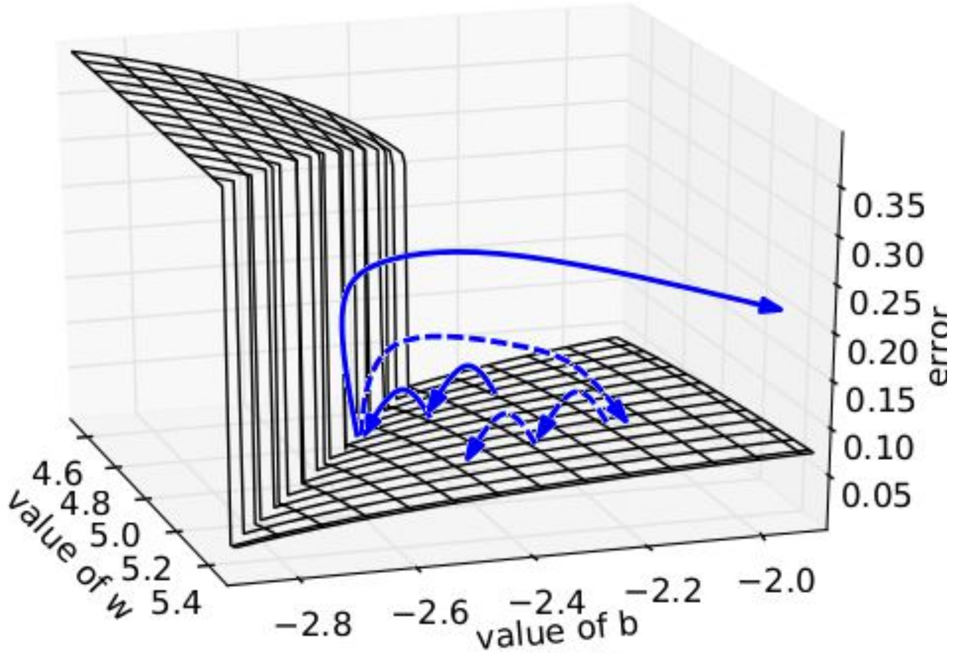
order of the

M. Morishita, Y. Oda, G. Neubig, K. Yoshino, K. Sudoh, and S. Nakamura. **"An Empirical Study of Mini-Batch Creation Strategies for Neural Machine Translation"**. 1st Workshop on NMT 2017

Gradient Norm Clipping

$$
\begin{aligned}
&\hat{\mathbf{g}} \leftarrow \frac{\partial error}{\partial \theta} \\
&\textbf{if } \ \|\hat{\mathbf{g}}\| \geq \textit{threshold} \textbf{ then} \\
&\quad \hat{\mathbf{g}} \leftarrow \frac{\textit{threshold}}{\|\hat{\mathbf{g}}\|}\hat{\mathbf{g}} \\
&\textbf{end if}
\end{aligned}
$$





Recurrent neural network regularization. Zaremba et al., arXiv 2014.

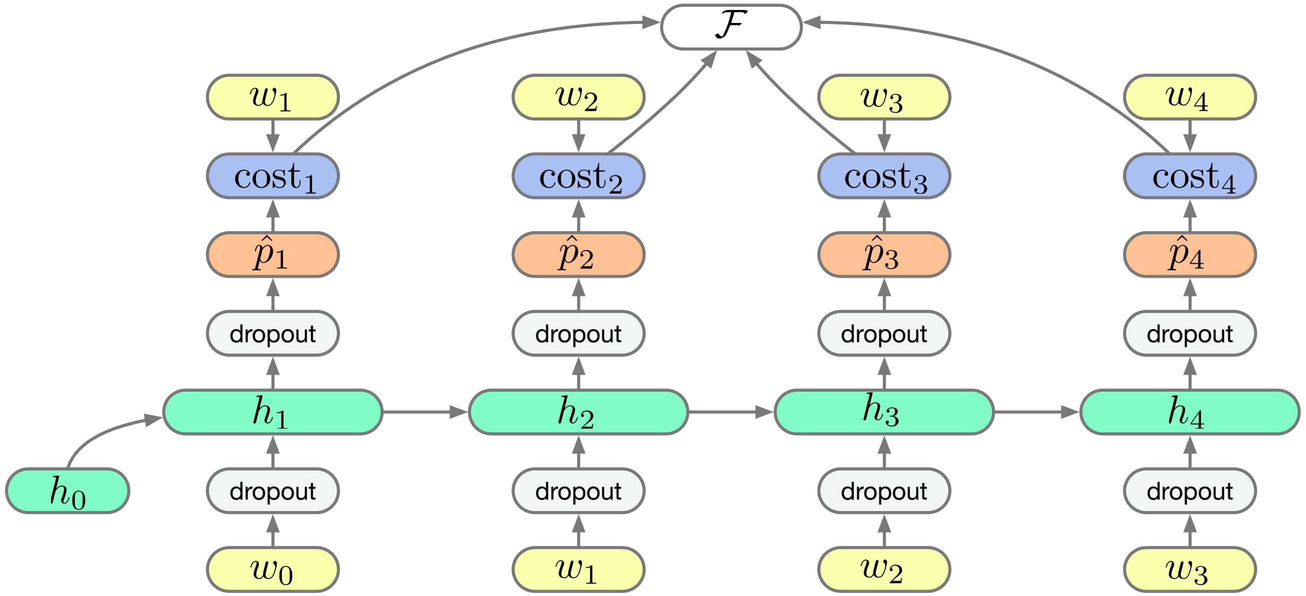
Regularization: Dropout

- Large recurrent networks often overfit their training data by memorizing the sequences observed. Such models generalize poorly to novel sequences.
- A common approach in Deep Learning is to overparametrize a model, such that it could easily memorize the training data, and then heavily regularize it to facilitate generalization.
- The regularization method of choice is often Dropout.

[Dropout: A Simple Way to Prevent Neural Networks from Overfitting. Srivastava et al. JMLR 2014.]

Regularization: Dropout

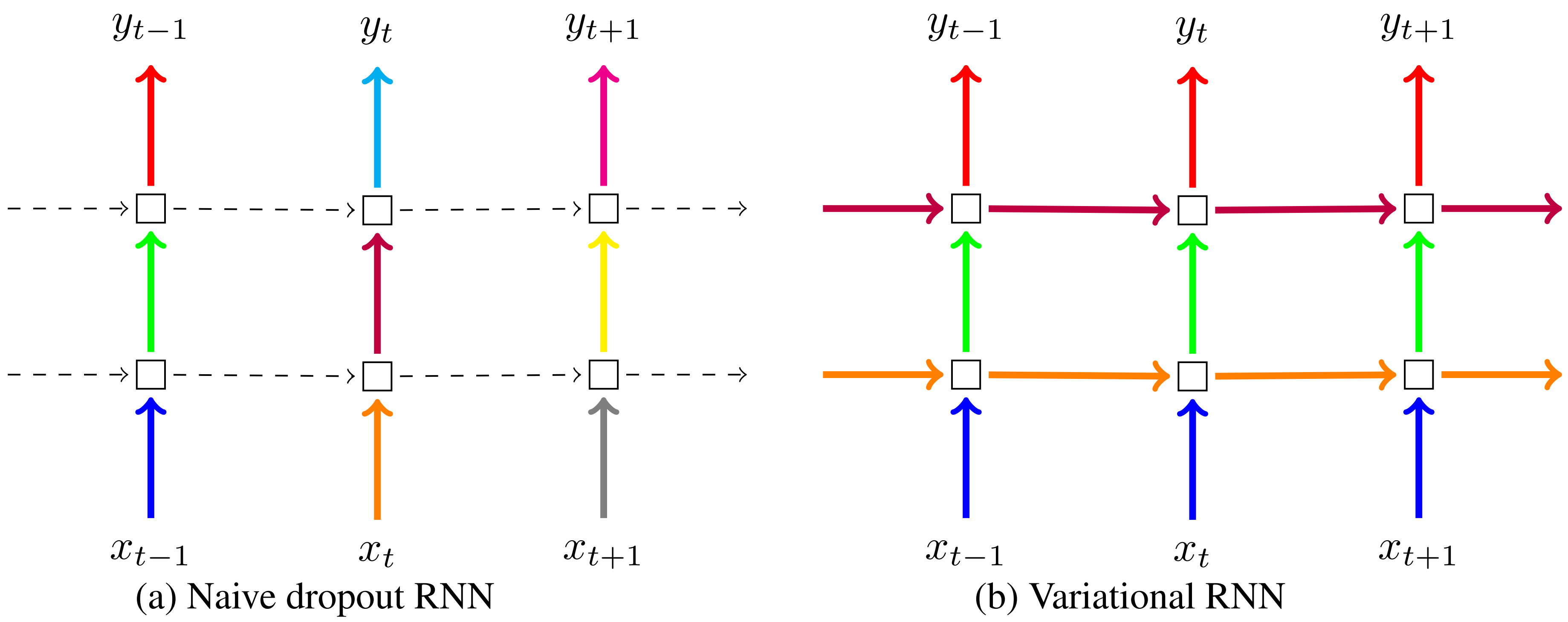
- Dropout is ineffective when applied to recurrent connections, as repeated random masks zero all hidden units in the limit.
- The most common solution is to only apply dropout to non-recurrent connections



[Recurrent neural network regularization. Zaremba et al., arXiv 2014.]

Regularization: Dropout

- **A Better Solution:** Use the same dropout mask at each time step for both inputs, outputs, and recurrent layers.



Each square represents an RNN unit, with horizontal arrows representing recurrent connections. Vertical arrows represent the input and output to each RNN unit. Coloured connections represent dropped-out inputs, with different colours corresponding to different dropout masks. Dashed lines correspond to standard connections with no dropout.

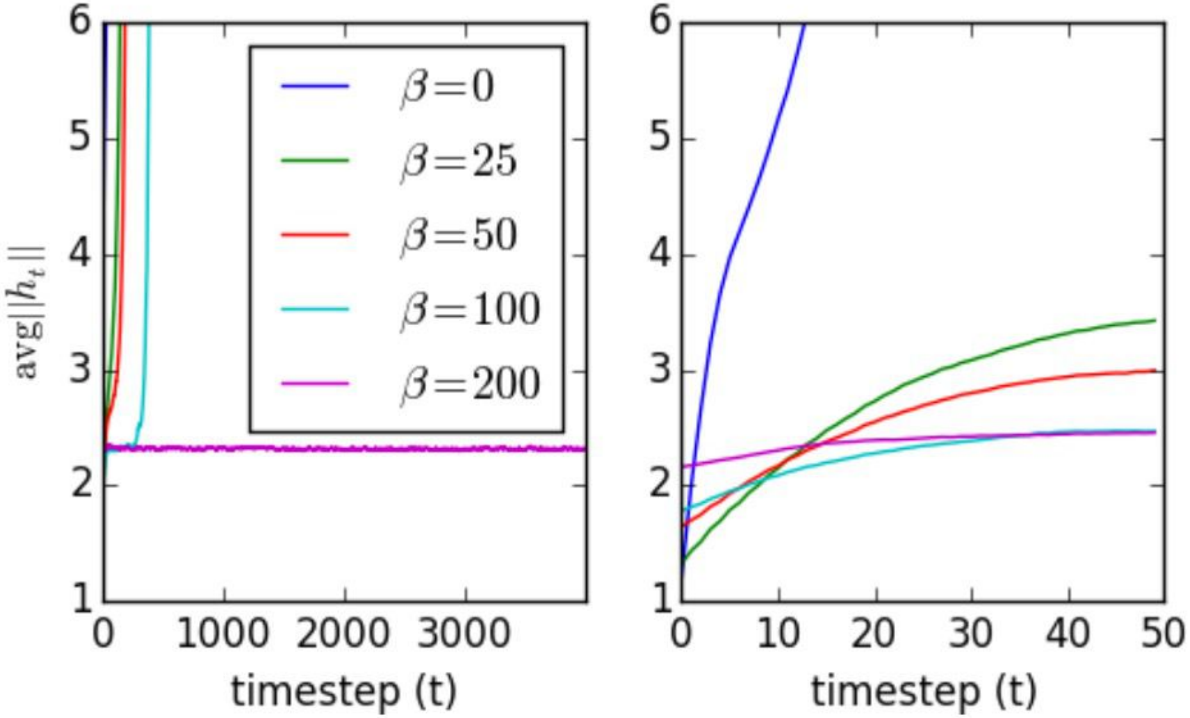
[A Theoretically Grounded Application of Dropout in Recurrent Neural Networks. Gal and Ghahramani. NIPS 2016]

Regularization: Norm-stabilizer

- Stabilize the activations of RNNs by penalizing the squared distance between successive hidden states' norms

$$\beta \frac{1}{T} \sum_{t=1}^{T} (\|h_t\|_2 - \|h_{t-1}\|_2)^2$$

- Enforce the norms of the hidden layer activations approximately constant across time



Regularizing RNNs by Stabilizing Activations. Krueger and Memisevic, ICLR 2016

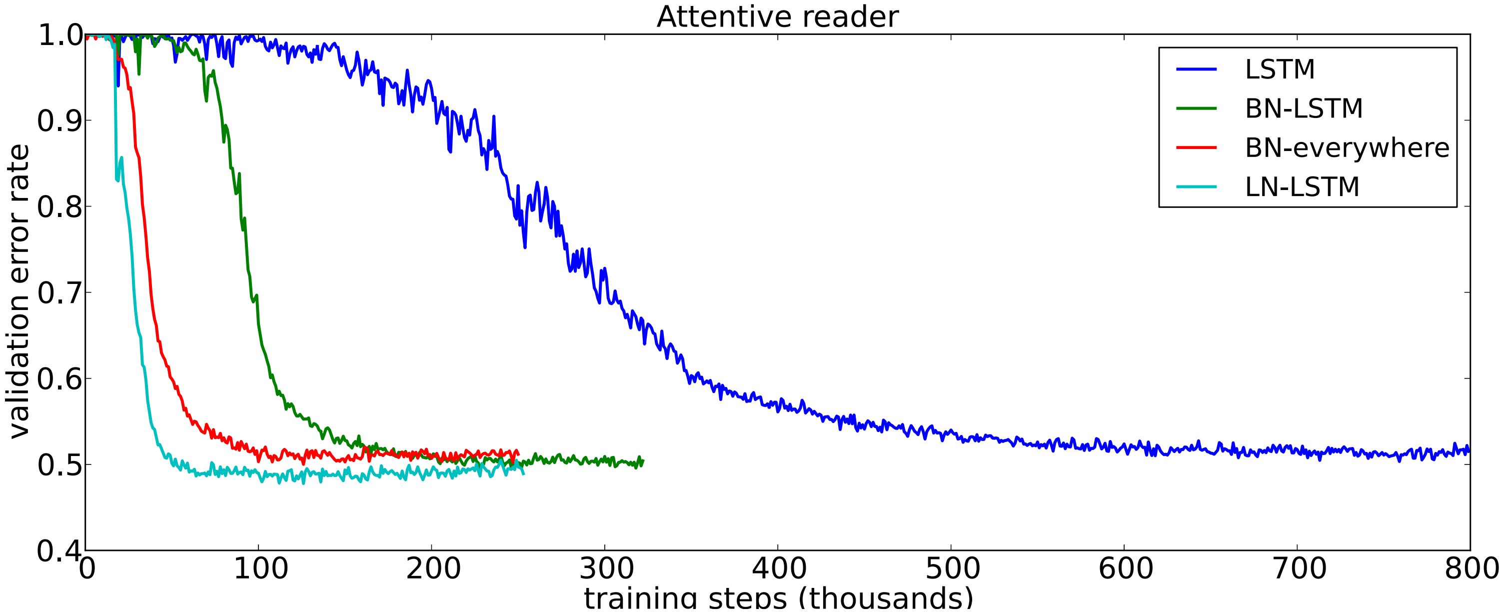
Regularization: Layer Normalization

- Similar to batch normalization
- Computes the normalization statistics separately at each time step
- Effective for stabilizing the hidden state dynamics in RNNs
- Reduces training time

$$\mathbf{h}^t = f\left[\frac{\mathbf{g}}{\sigma^t} \odot \left(\mathbf{a}^t - \mu^t\right) + \mathbf{b}\right]$$

$$\mu^t = \frac{1}{H}\sum_{i=1}^{H} a_i^t$$

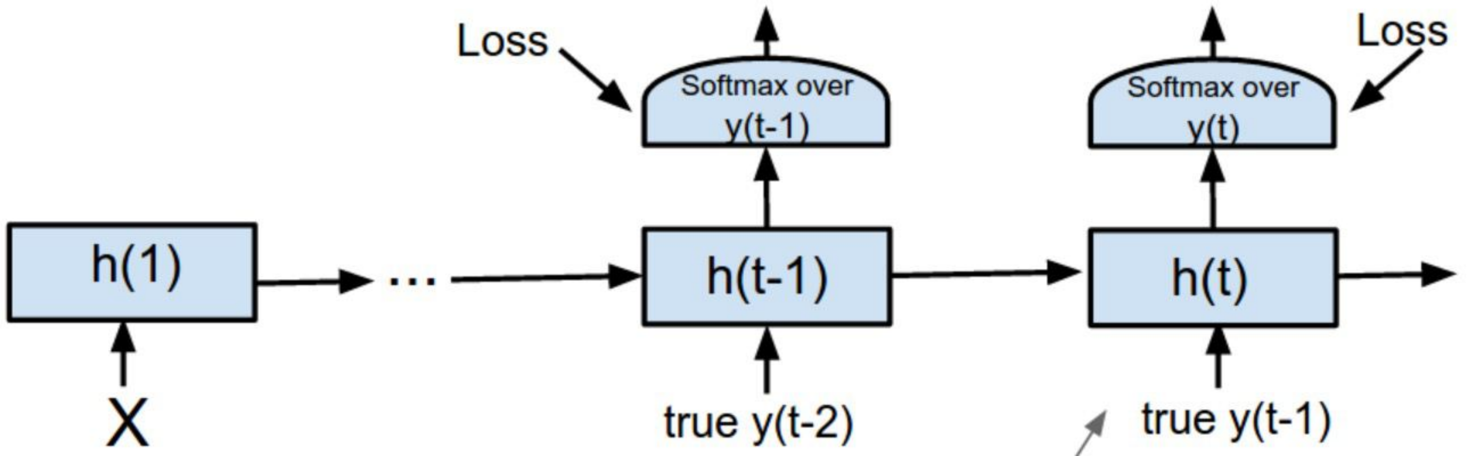
$$\sigma^t = \sqrt{\frac{1}{H}\sum_{i=1}^{H}\left(a_i^t - \mu^t\right)^2}$$



Layer Normalization [Ba, Kiros & Hinton, 2016]

Scheduled Sampling

- *"change the training process from a fully guided scheme using the true previous token, towards a less guided scheme which mostly uses the generated token instead."*

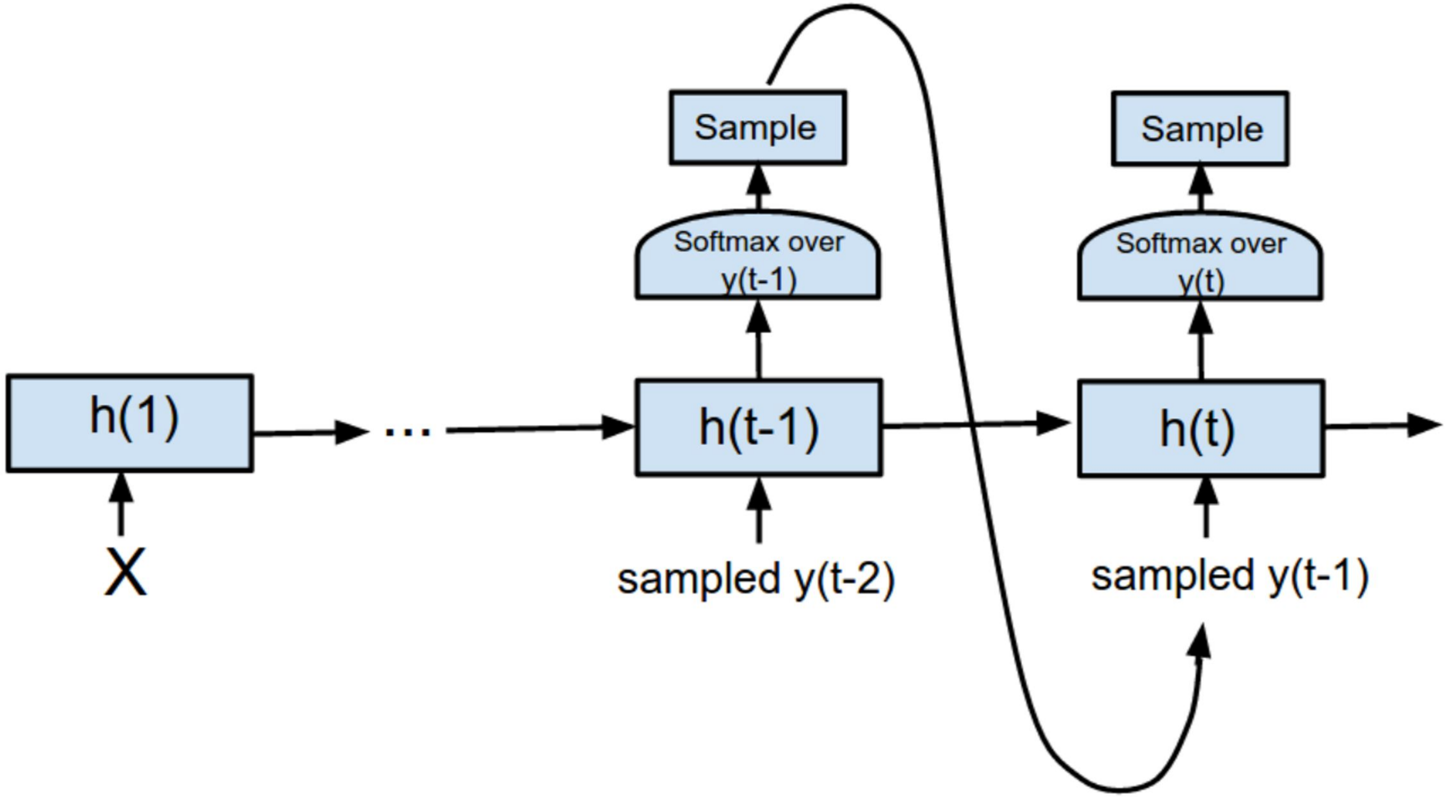


$$P(y_t|h_t) \text{ with } h_t = f(h_{t-1}, y_{t-1})$$

[Scheduled Sampling for Sequence Prediction with Recurrent Neural Networks. Bengio et al., NIPS 2015]

Scheduled Sampling

- *"change the training process from a fully guided scheme using the true previous token, towards a less guided scheme which mostly uses the generated token instead."*
- During training, randomly replace a **conditioning** ground truth token by the model's previous prediction

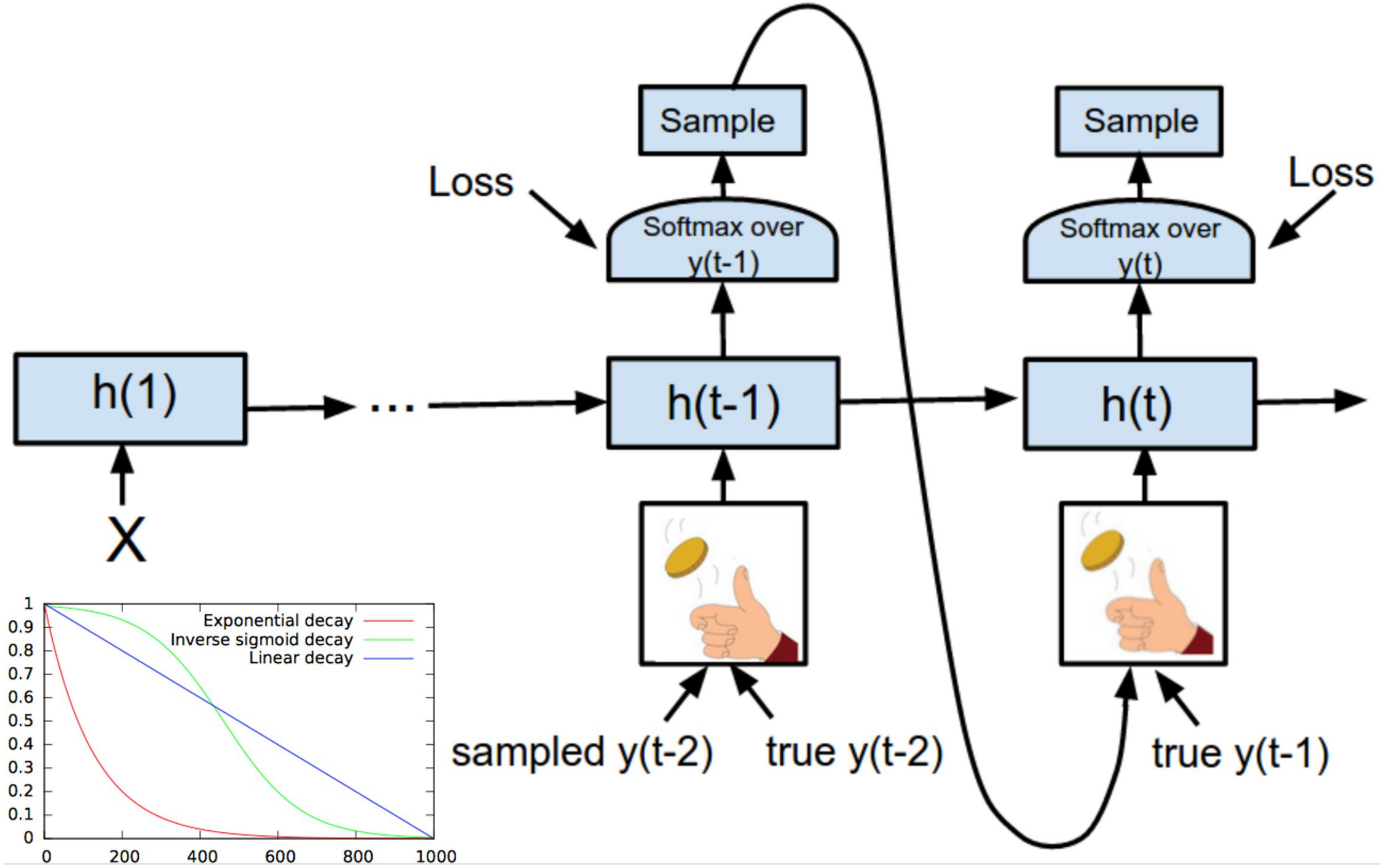


$$P(y_t|h_t) \text{ with } h_t = f(h_{t-1}, \hat{y}_{t-1})$$

[Scheduled Sampling for Sequence Prediction with Recurrent Neural Networks. Bengio et al., NIPS 2015](#)

Scheduled Sampling

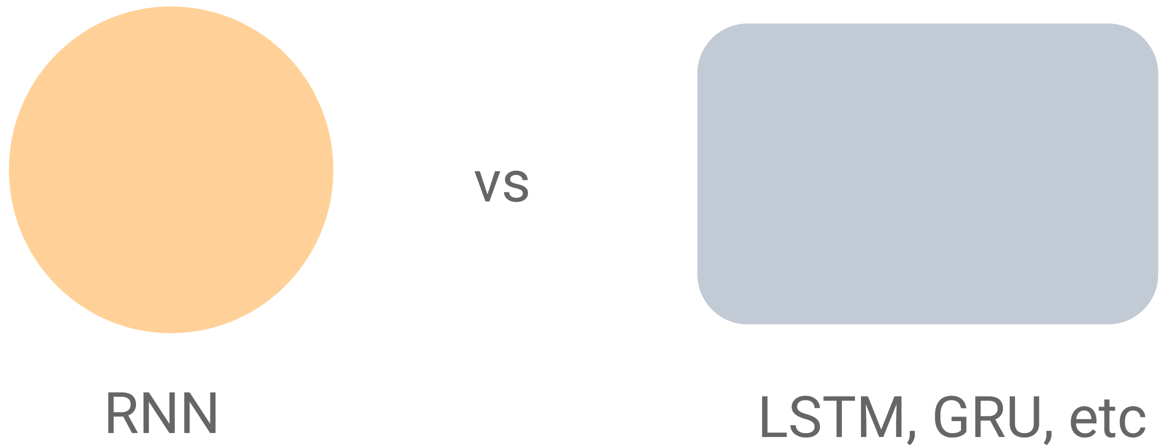
- *"change the training process from a fully guided scheme using the true previous token, towards a less guided scheme which mostly uses the generated token instead."*
- During training, randomly replace a **conditioning** ground truth token by the model's previous prediction



[Scheduled Sampling for Sequence Prediction with Recurrent Neural Networks. Bengio et al., NIPS 2015]

Gated Cells

- rather each node being just a simple RNN cell, make each node a more **complex unit with gates** controlling what information is passed through



Long short term memory cells are able to keep track of information throughout many timesteps.

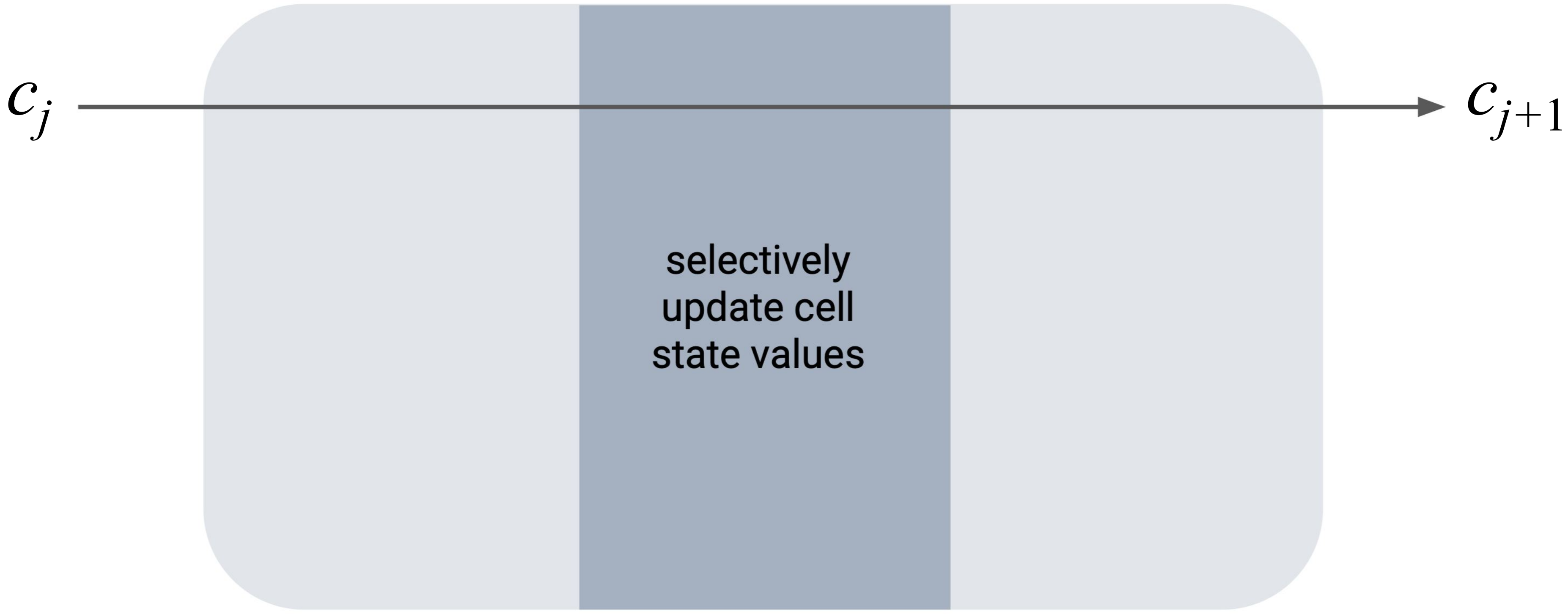
Long Short-Term Memory (LSTM)

[Long Short-Term Memory [Hochreiter et al., 1997]

Long Short-Term Memory (LSTM)

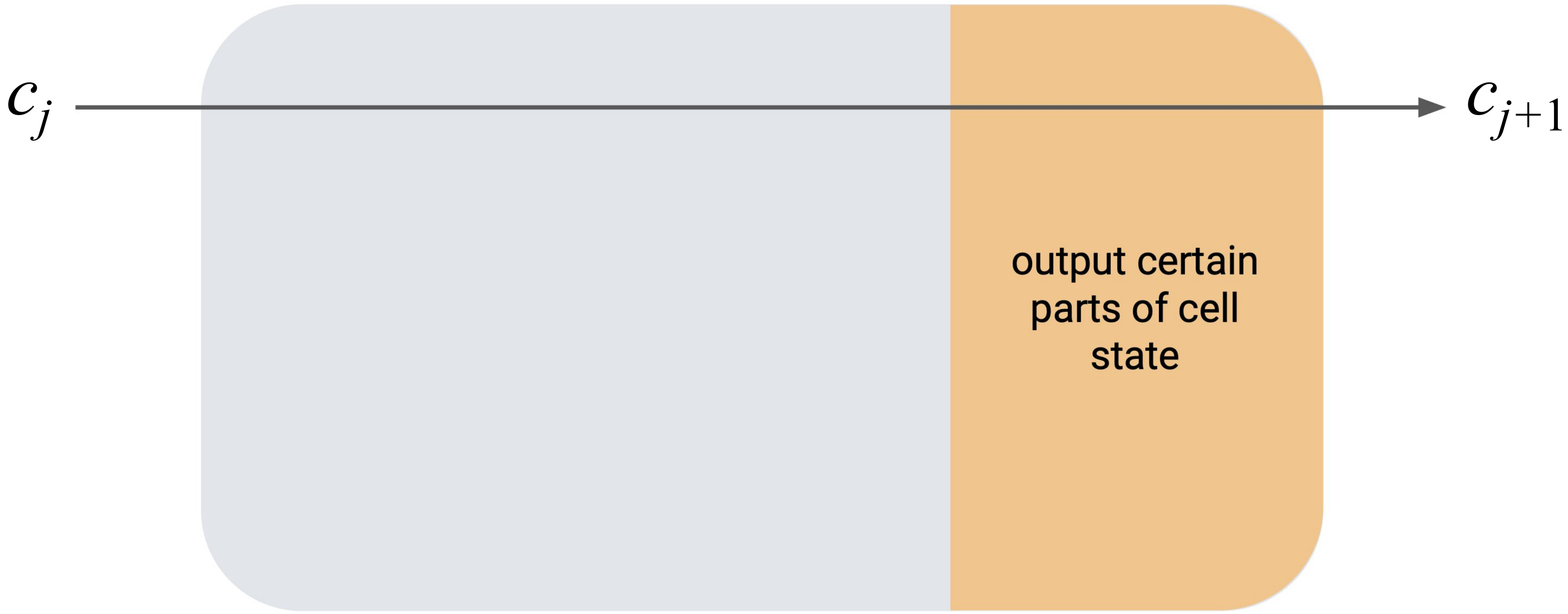
[Long Short-Term Memory [Hochreiter et al., 1997]]

Long Short-Term Memory (LSTM)



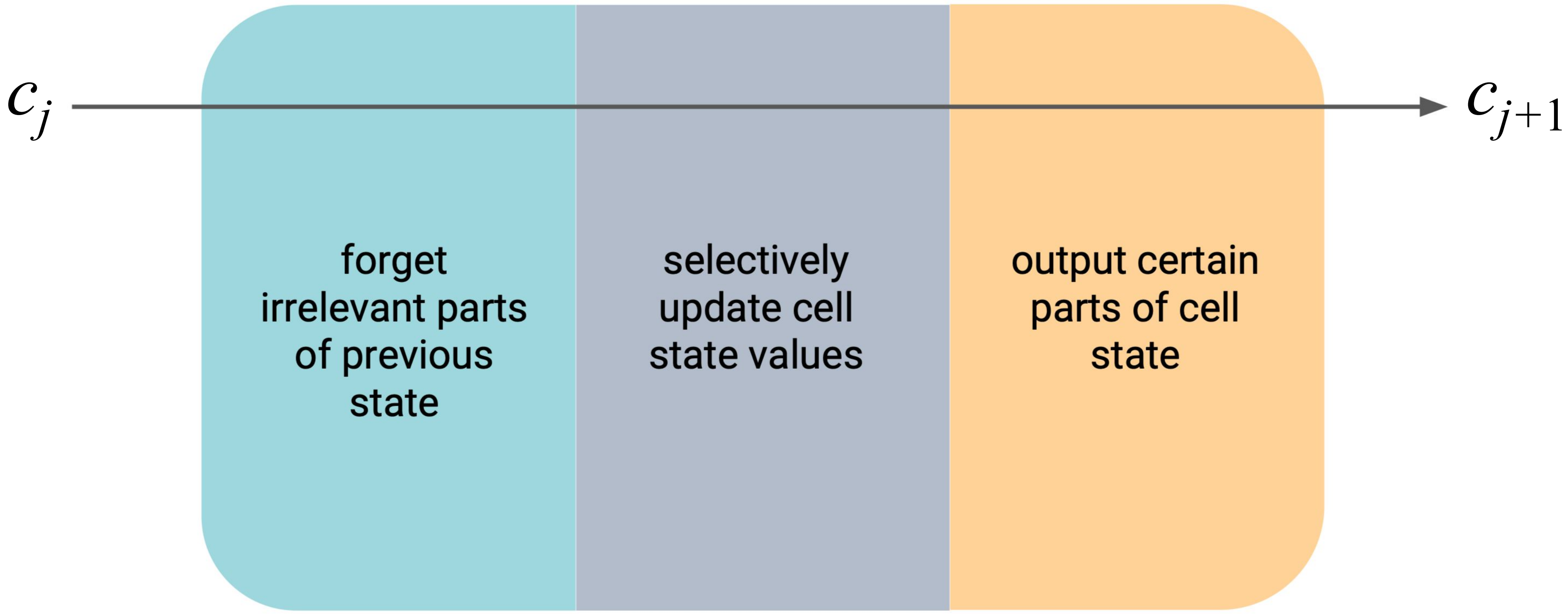
Long Short-Term Memory [Hochreiter et al., 1997]

Long Short-Term Memory (LSTM)



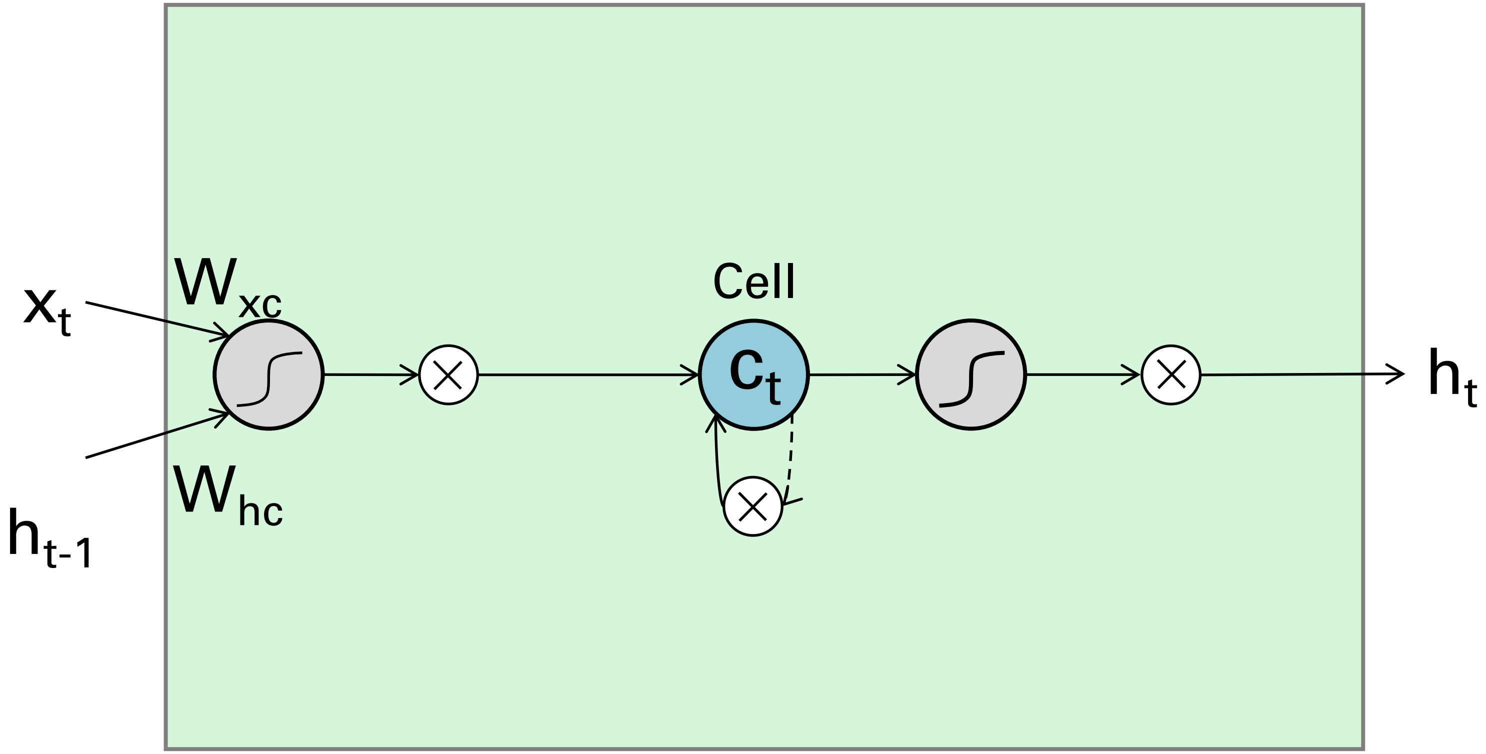
Long Short-Term Memory [Hochreiter et al., 1997]

Long Short-Term Memory (LSTM)



Long Short-Term Memory [Hochreiter et al., 1997]

The LSTM Idea

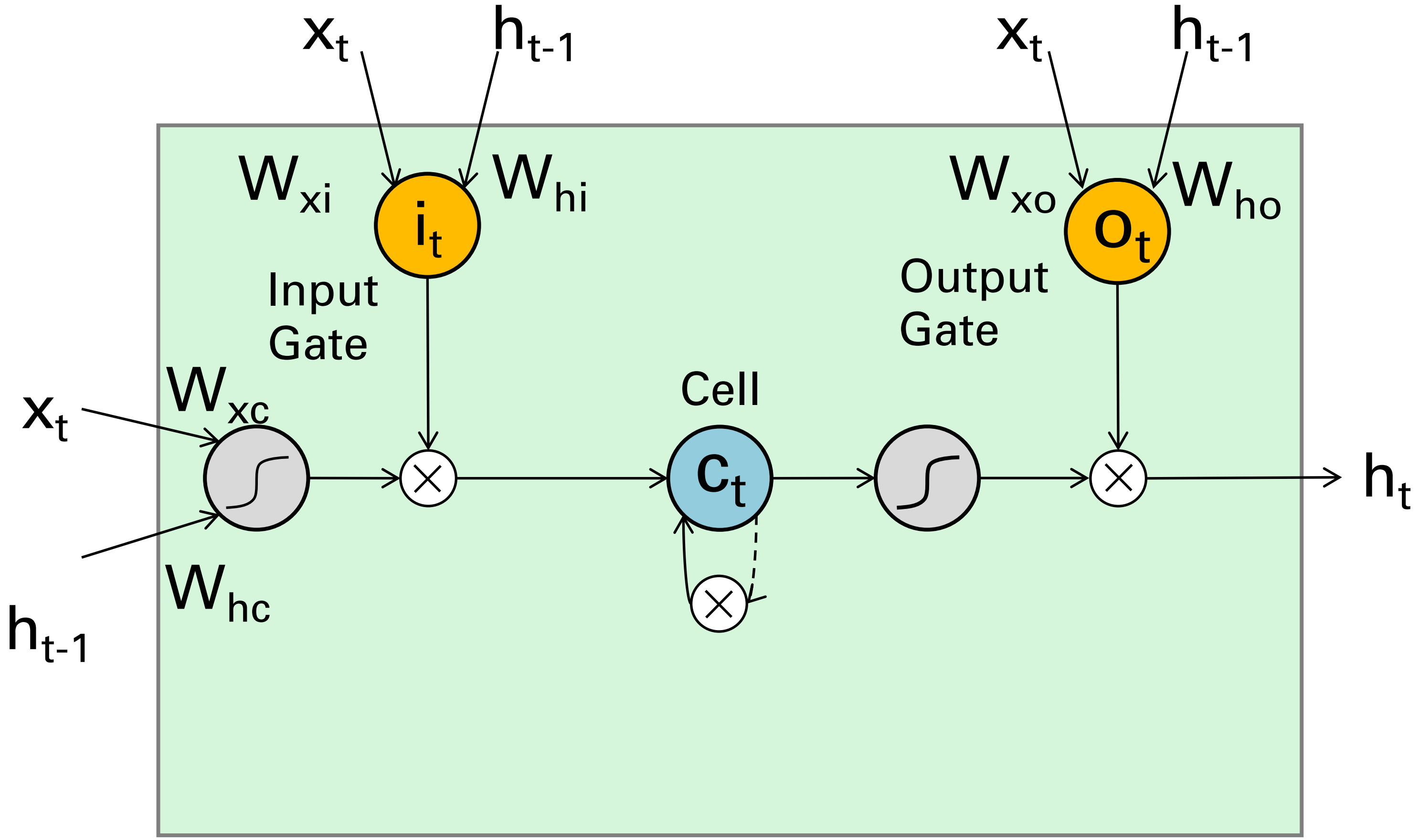


* Dashed line indicates time-lag

$$c_t = c_{t-1} + \tanh W \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix}$$

$$h_t = \tanh c_t$$

The Original LSTM Cell



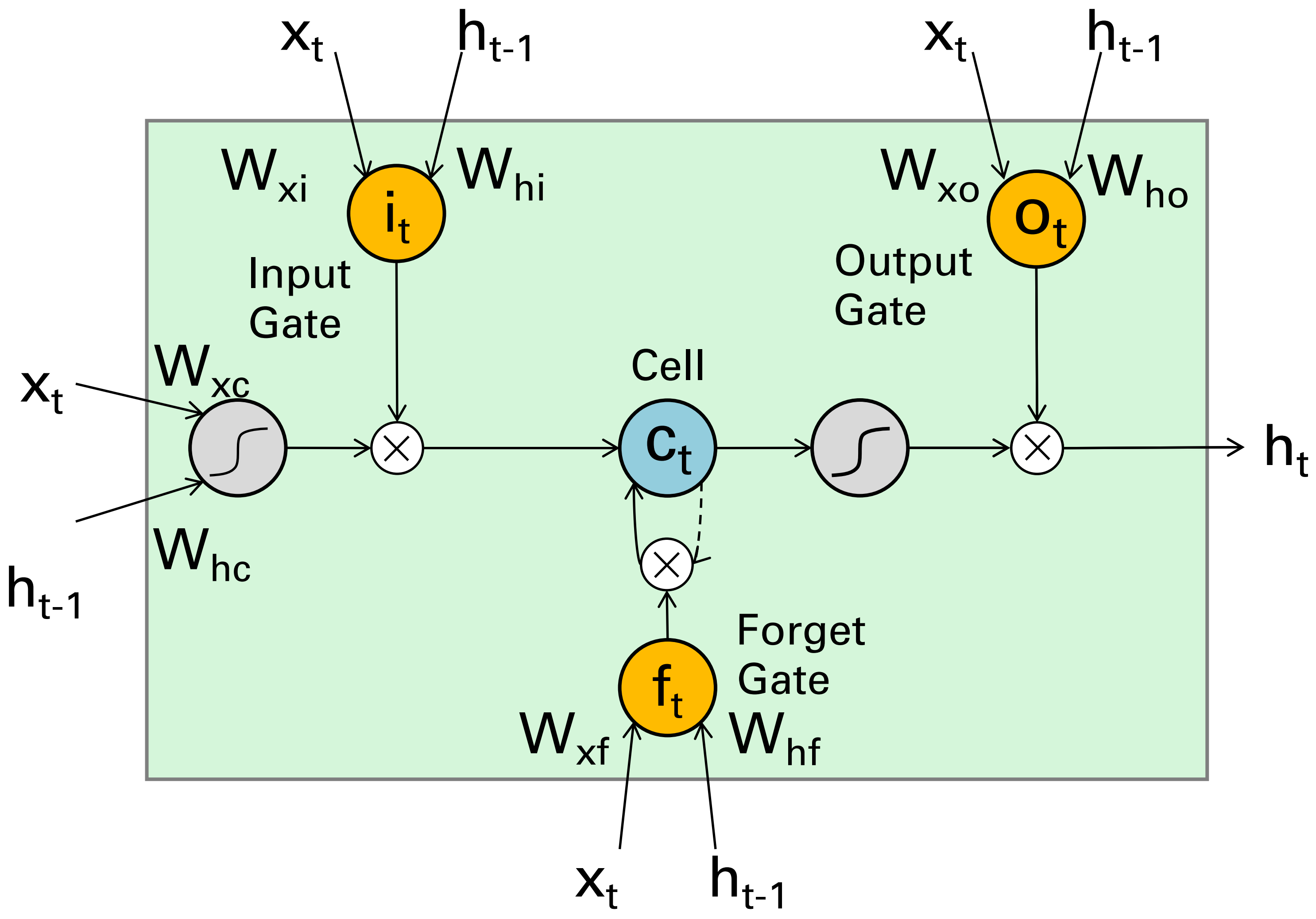
$$c_t = c_{t-1} + i_t \otimes \tanh W \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix}$$

$$h_t = o_t \otimes \tanh c_t$$

$$i_t = \sigma \left(W_i \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix} + b_i \right)$$

Similarly for o_t

The Popular LSTM Cell



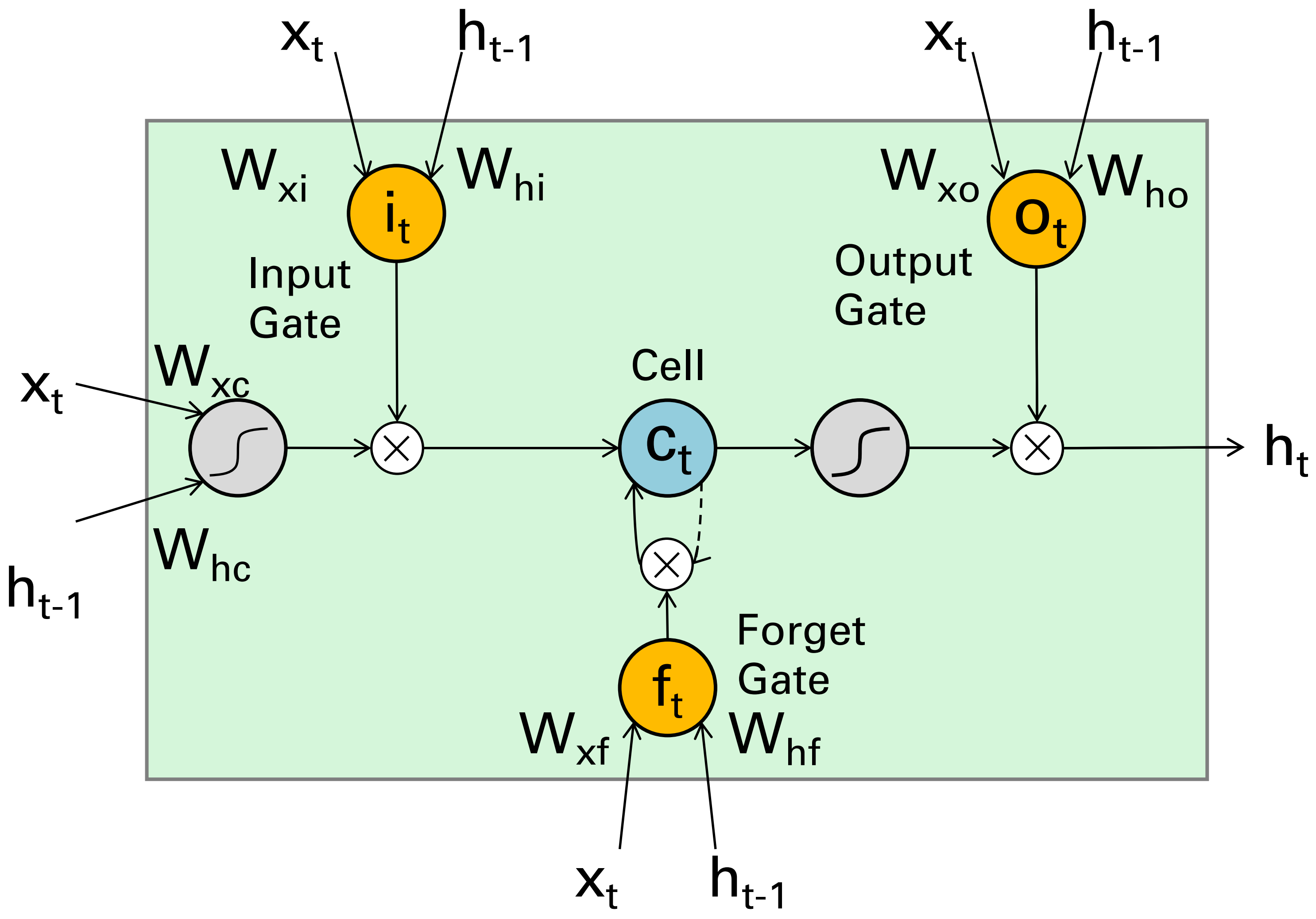
$$i_t = \sigma\left(W_i \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix} + b_i\right)$$

$$c_t = f_t \otimes c_{t-1} + i_t \otimes \tanh W \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix}$$

$$f_t = \sigma\left(W_f \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix} + b_f\right)$$

$$h_t = o_t \otimes \tanh c_t$$

The Popular LSTM Cell



$$i_t = \sigma\left(W_i\begin{pmatrix}x_t\\h_{t-1}\end{pmatrix}+b_i\right)$$

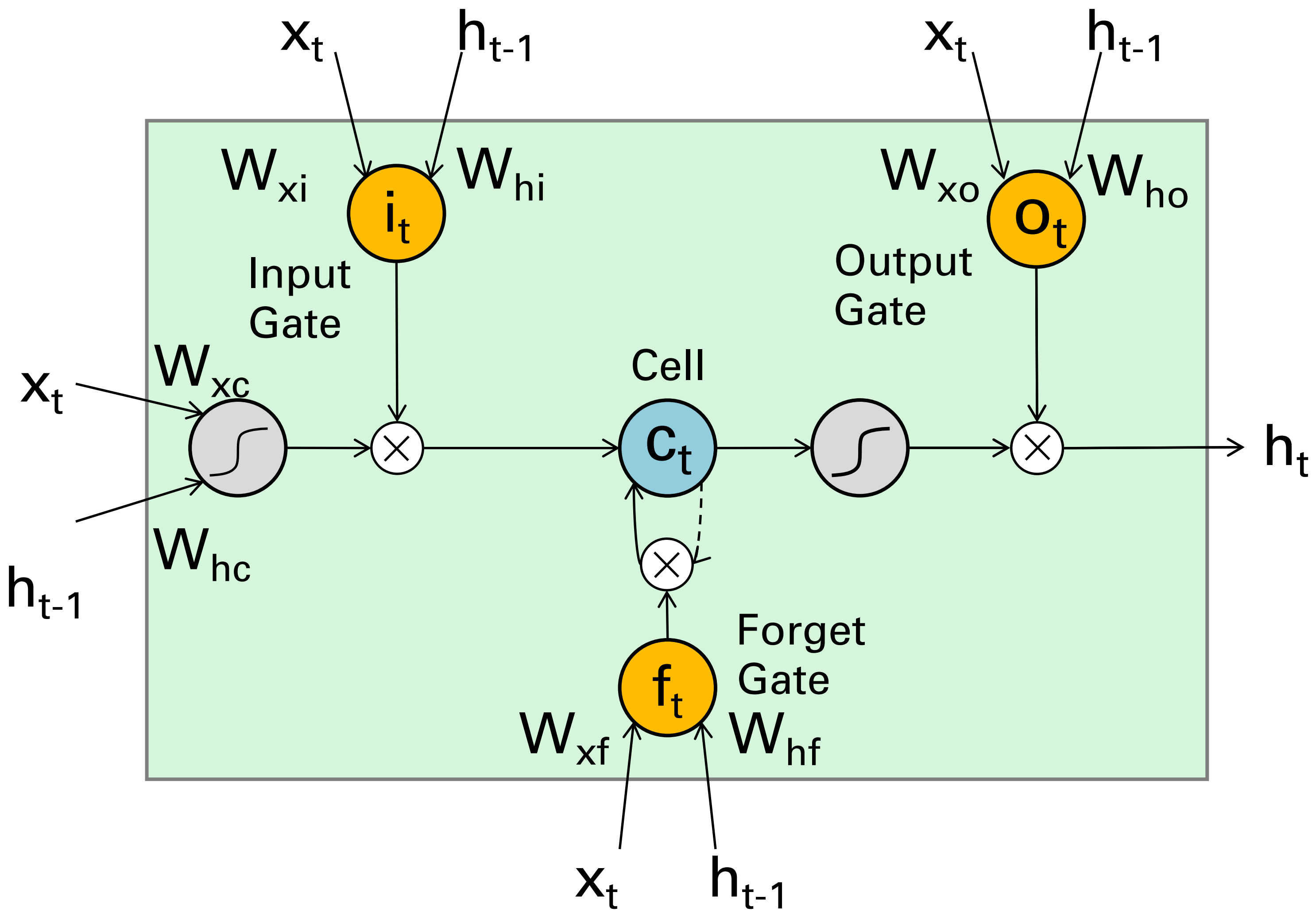
$$c_t = f_t \otimes c_{t-1} + i_t \otimes \tanh W\begin{pmatrix}x_t\\h_{t-1}\end{pmatrix}$$

$$\boxed{f_t = \sigma\left(W_f\begin{pmatrix}x_t\\h_{t-1}\end{pmatrix}+b_f\right)}$$

$$h_t = o_t \otimes \tanh c_t$$

forget gate decides what information is going to be thrown away from the cell state

The Popular LSTM Cell



$$\boxed{i_t = \sigma\left(W_i \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix} + b_i\right)}$$

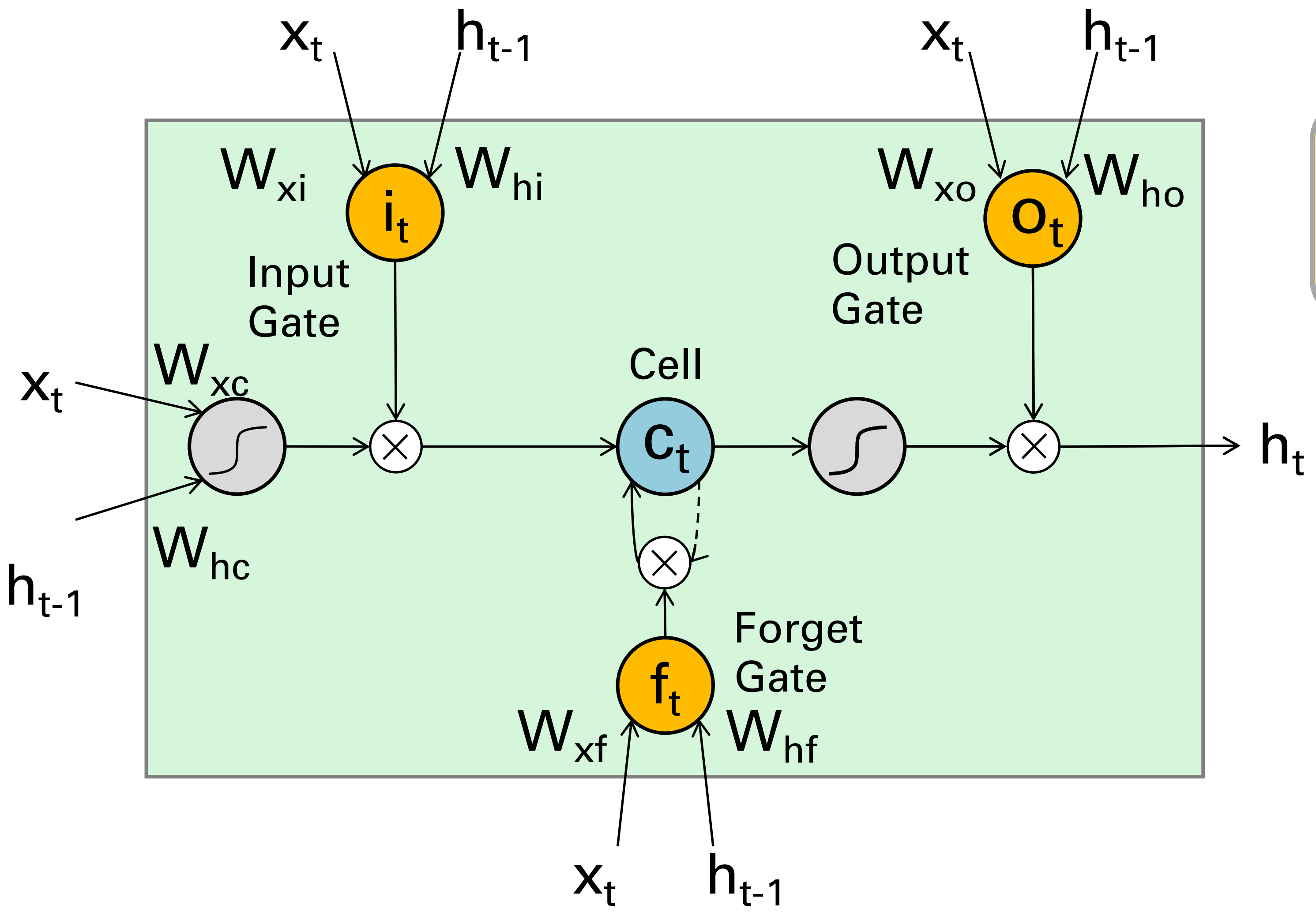
$$c_t = f_t \otimes c_{t-1} + \boxed{i_t \otimes \tanh W \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix}}$$

$$f_t = \sigma\left(W_f \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix} + b_f\right)$$

$$h_t = o_t \otimes \tanh c_t$$

input gate and **a tanh layer** decides what information is going to be stored in the cell state

The Popular LSTM Cell



$$i_t = \sigma\left(W_i \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix} + b_i\right)$$

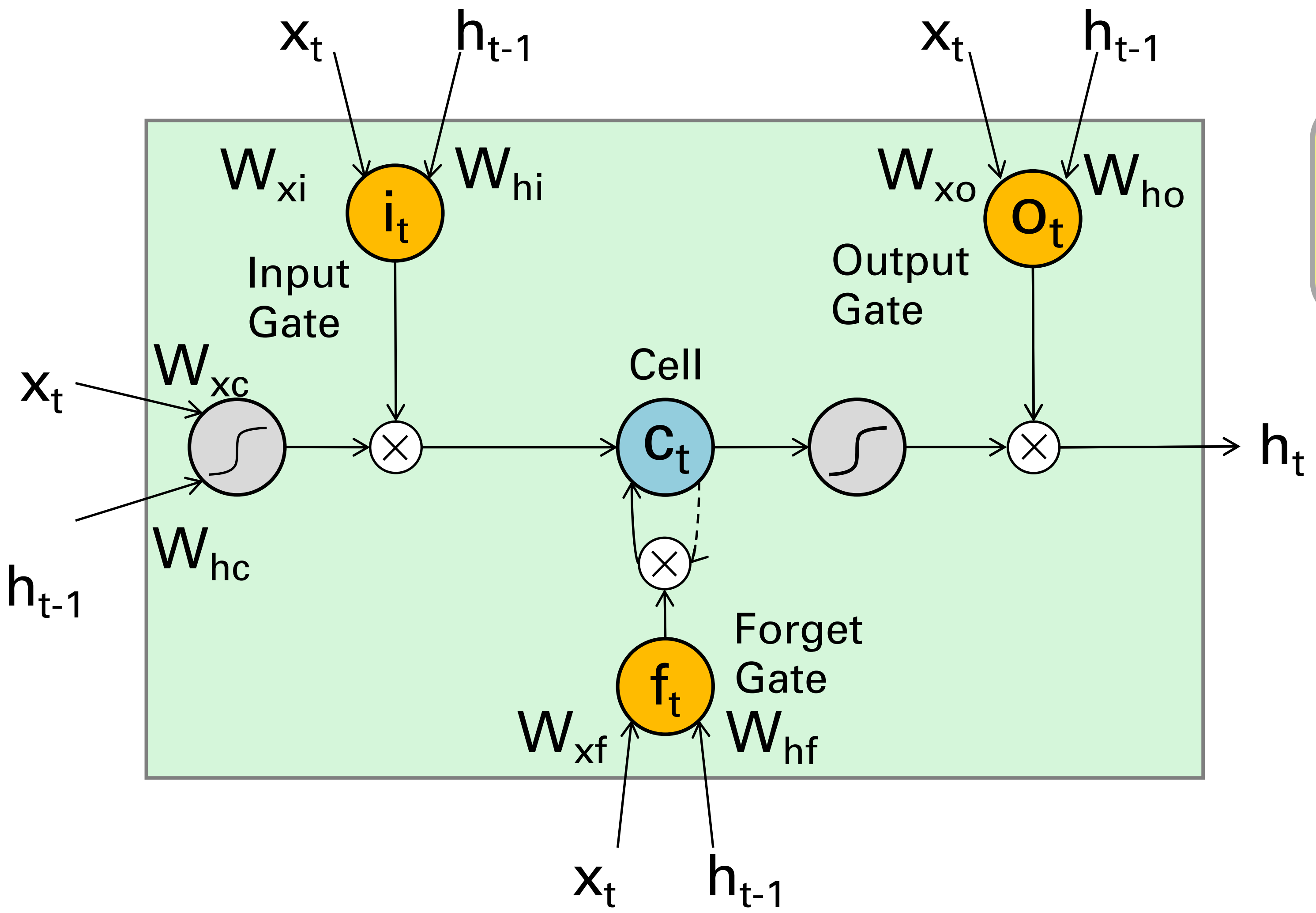
$$\boxed{c_t = f_t \otimes c_{t-1} + i_t \otimes \tanh W \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix}}$$

$$f_t = \sigma\left(W_f \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix} + b_f\right)$$

$$h_t = o_t \otimes \tanh c_t$$

Update the old cell state with the new one.

The Popular LSTM Cell

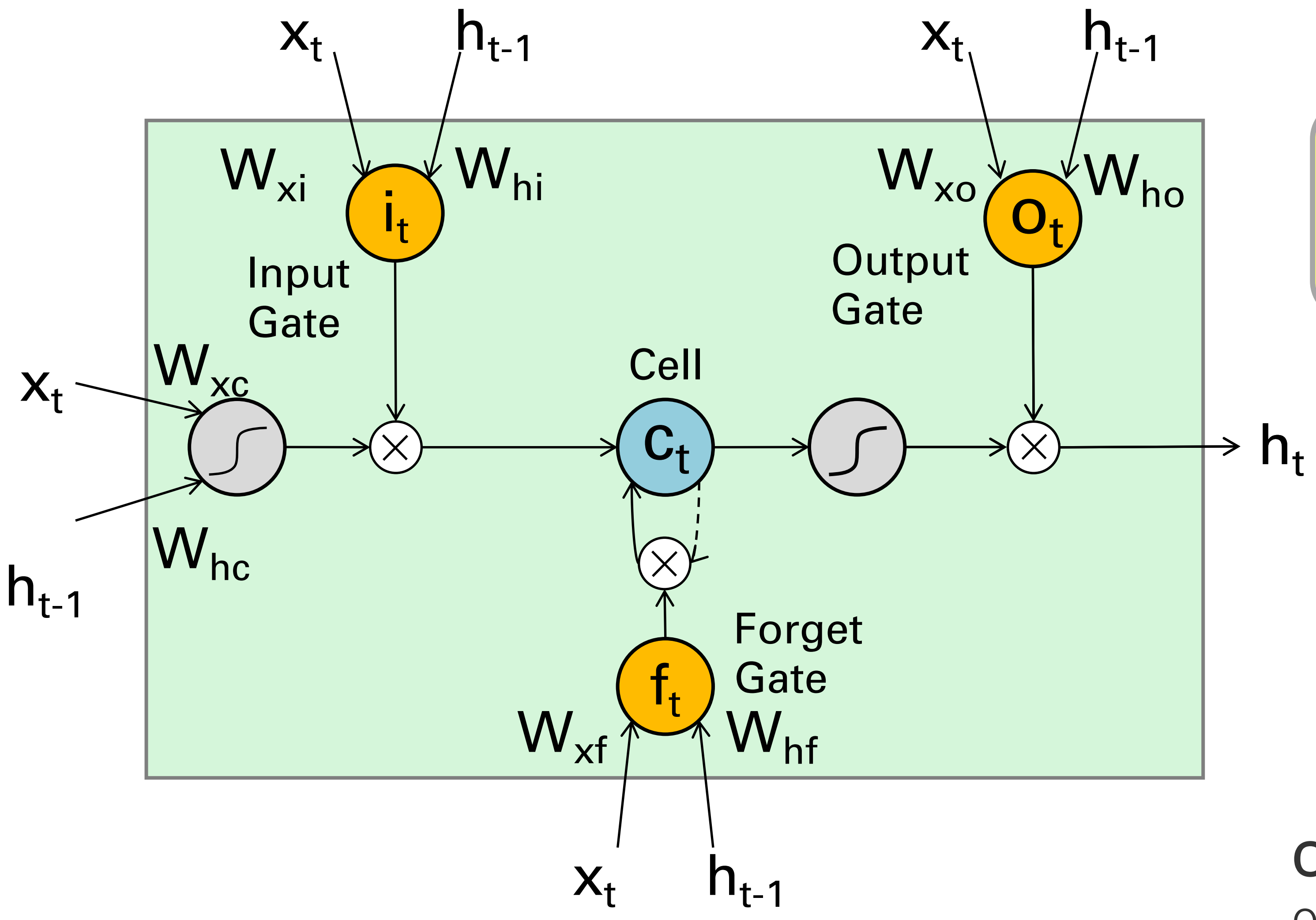


$$i_t = \sigma\left(W_i \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix} + b_i \right)$$

$$c_t = f_t \otimes c_{t-1} + i_t \otimes \tanh W \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix}$$

input gate	forget gate	behavior
0	1	remember the previous value
1	1	add to the previous value
0	0	erase the value
1	0	overwrite the value

The Popular LSTM Cell



$$i_t = \sigma\left(W_i \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix} + b_i \right)$$

$$\boxed{c_t = f_t \otimes c_{t-1} + i_t \otimes \tanh W \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix}}$$

$$f_t = \sigma\left(W_f \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix} + b_f \right)$$

$$h_t = o_t \otimes \tanh c_t$$

$$o_i = \sigma\left(W_o \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix} + b_o \right)$$

Output gate decides what is going to be outputted. The final output is based on cell state and output of sigmoid gate.

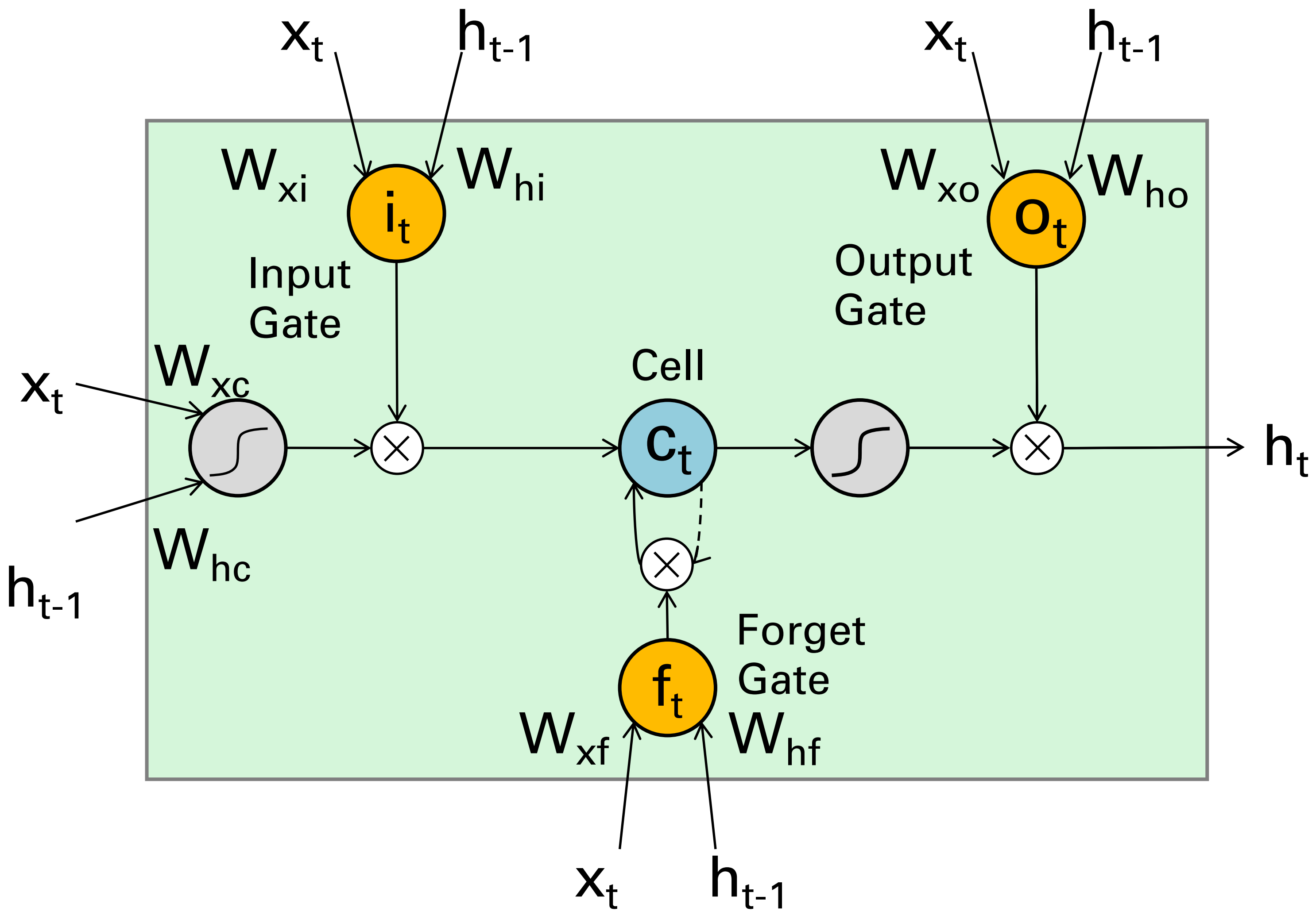
LSTM – Forward/Backward

[Illustrated LSTM Forward and Backward Pass](http://arunmallya.github.io/writeups/nn/lstm/index.html)

http://arunmallya.github.io/writeups/nn/lstm/index.html

LSTM variants

The Popular LSTM Cell



$$f_t = \sigma\left(W_f\begin{pmatrix}x_t\\ h_{t-1}\end{pmatrix} + b_f\right)$$

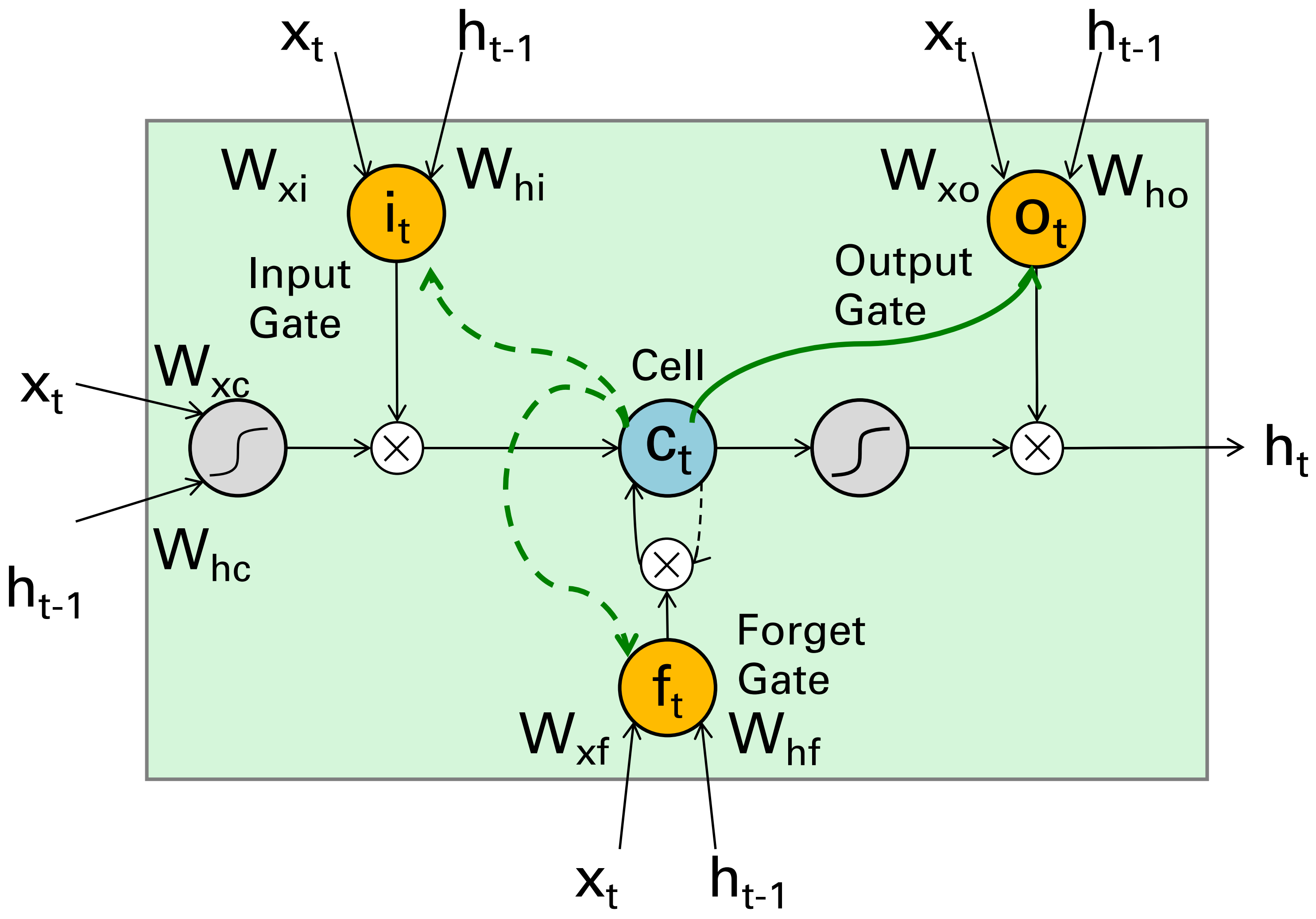
Similarly for i_t, o_t

$$c_t = f_t \otimes c_{t-1} + i_t \otimes \tanh W\begin{pmatrix}x_t\\ h_{t-1}\end{pmatrix}$$

$$h_t = o_t \otimes \tanh c_t$$

* Dashed line indicates time-lag

Extension I: Peephole LSTM



$$f_t = \sigma\left(W_f \begin{pmatrix} x_t \\ h_{t-1} \\ c_{t-1} \end{pmatrix} + b_f \right)$$

Similarly for i_t, o_t (uses c_t)

$$c_t = f_t \otimes c_{t-1} + i_t \otimes \tanh W \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix}$$

$$h_t = o_t \otimes \tanh c_t$$

- Add **peephole connections**.
- All gate layers look at the cell state!

* Dashed line indicates time-lag

Other minor variants

- Coupled Input and Forget Gate $f_t = 1 - i_t$

- Full Gate Recurrence

$$f_t = \sigma\left(W_f \begin{pmatrix} x_t \\ h_{t-1} \\ c_{t-1} \\ i_{t-1} \\ f_{t-1} \\ o_{t-1} \end{pmatrix} + b_f \right)$$

LSTM: A Search Space Odyssey

- Tested the following variants, using Peephole LSTM as standard:
  1. No Input Gate (NIG)
  2. No Forget Gate (NFG)
  3. No Output Gate (NOG)
  4. No Input Activation Function (NIAF)
  5. No Output Activation Function (NOAF)
  6. No Peepholes (NP)
  7. Coupled Input and Forget Gate (CIFG)
  8. Full Gate Recurrence (FGR)

- On the tasks of:
  – Timit Speech Recognition: Audio frame to 1 of 61 phonemes
  – IAM Online Handwriting Recognition: Sketch to characters
  – JSB Chorales: Next-step music frame prediction

LSTM: A Search Space Odyssey [Greff et al., 2015]

LSTM: A Search Space Odyssey

- The standard LSTM performed reasonably well on multiple datasets and none of the modifications significantly improved the performance
- Coupling gates and removing peephole connections simplified the LSTM without hurting performance much
- The forget gate and output activation are crucial

- Found interaction between learning rate and network size to be minimal – indicates calibration can be done using a small network first

[LSTM: A Search Space Odyssey [Greff et al., 2015]]

Gated Recurrent Unit

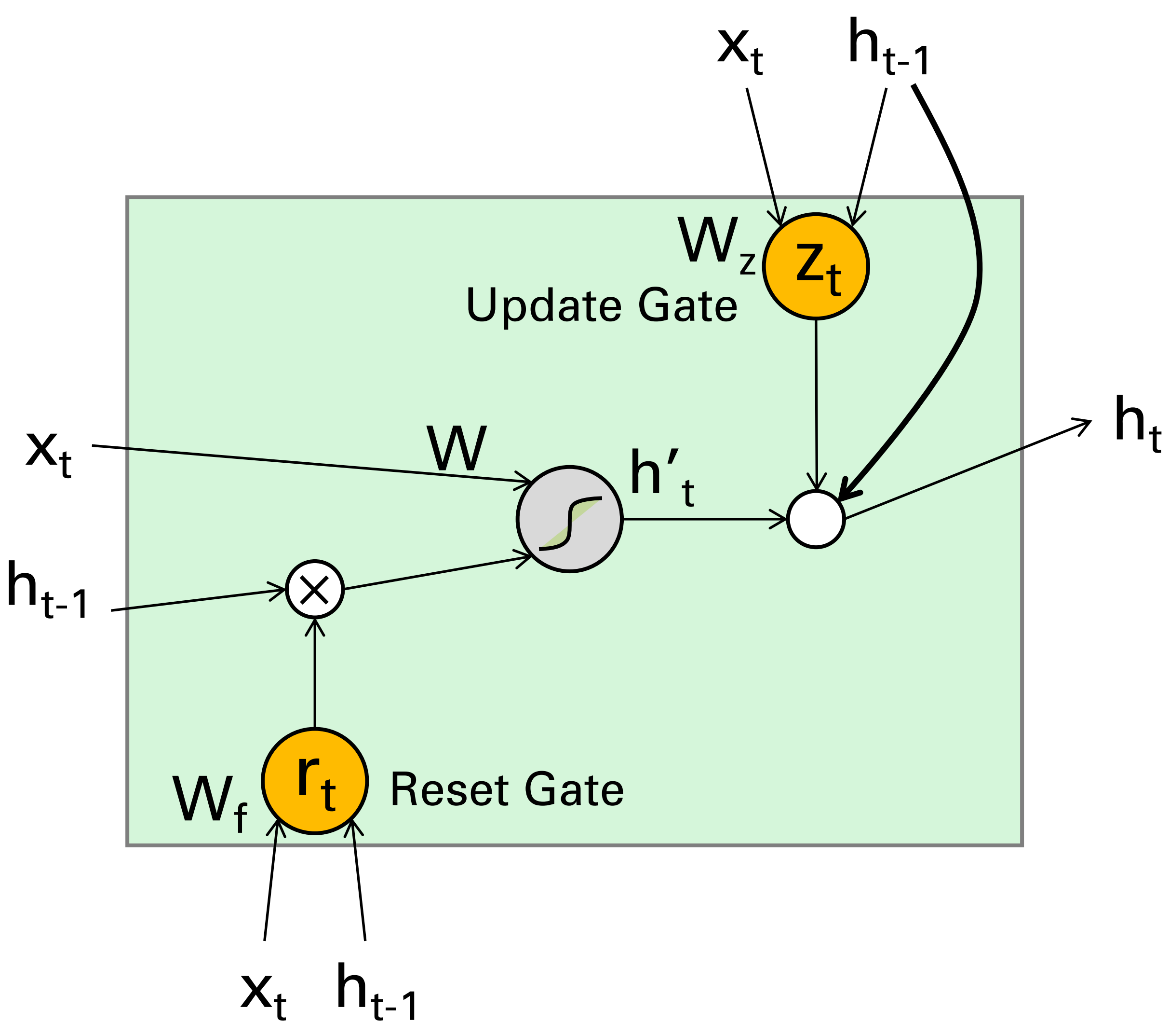
Gated Recurrent Unit (GRU)

- A very simplified version of the LSTM
  - Merges forget and input gate into a single ‘update’ gate
  - Merges cell and hidden state

- Has fewer parameters than an LSTM and has been shown to outperform LSTM on some tasks

Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation [Cho et al.,14]

GRU



$$r_t = \sigma\left(W_r \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix} + b_f \right)$$

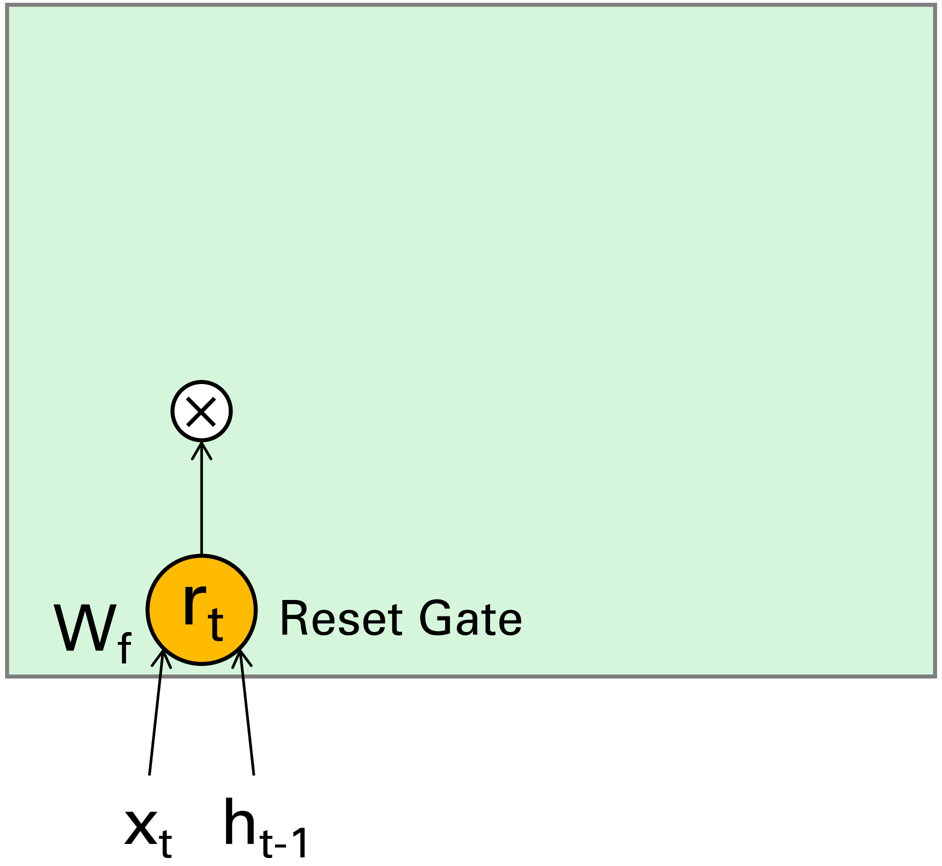
$$h'_t = \tanh W \begin{pmatrix} x_t \\ r_t \otimes h_{t-1} \end{pmatrix}$$

$$z_t = \sigma\left(W_z \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix} + b_f \right)$$

$$h_t = (1 - z_t) \otimes h_{t-1} + z_t \otimes h'_t$$

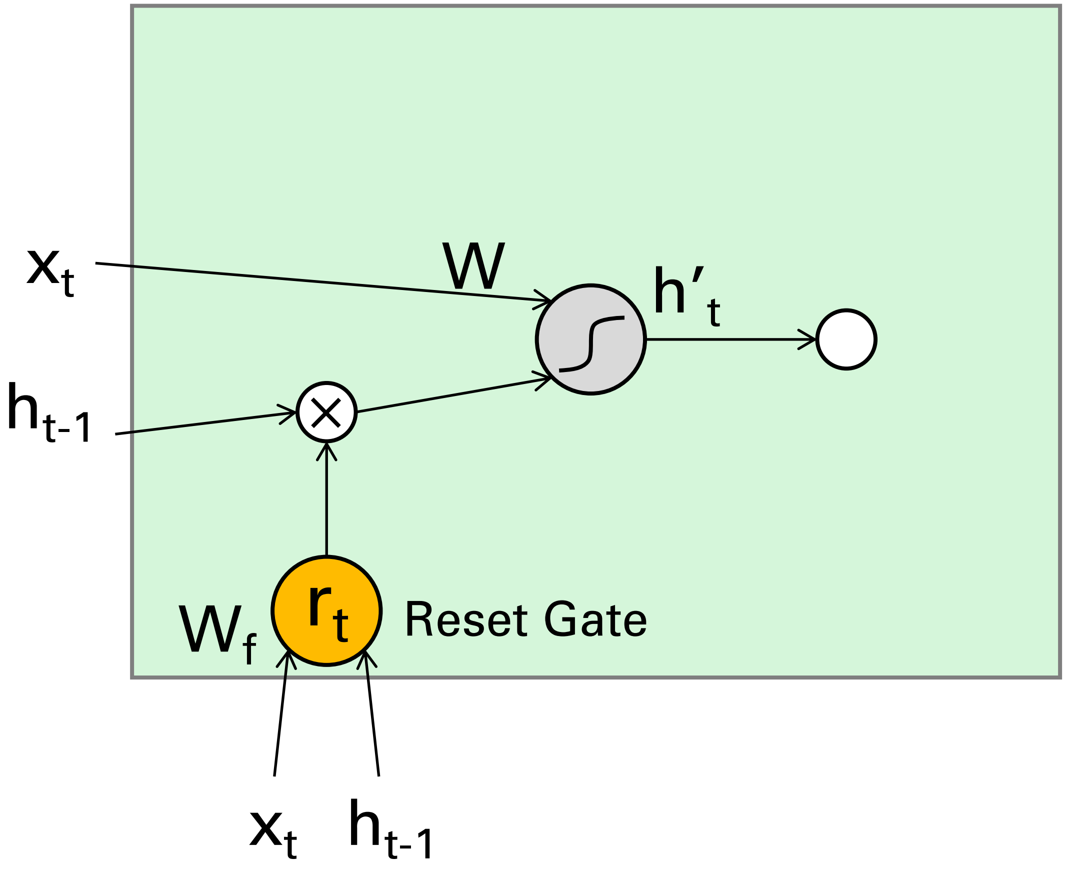
GRU

$$r_t = \sigma\left(W_r \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix} + b_f\right)$$



computes a **reset gate** based on current input and hidden state

GRU



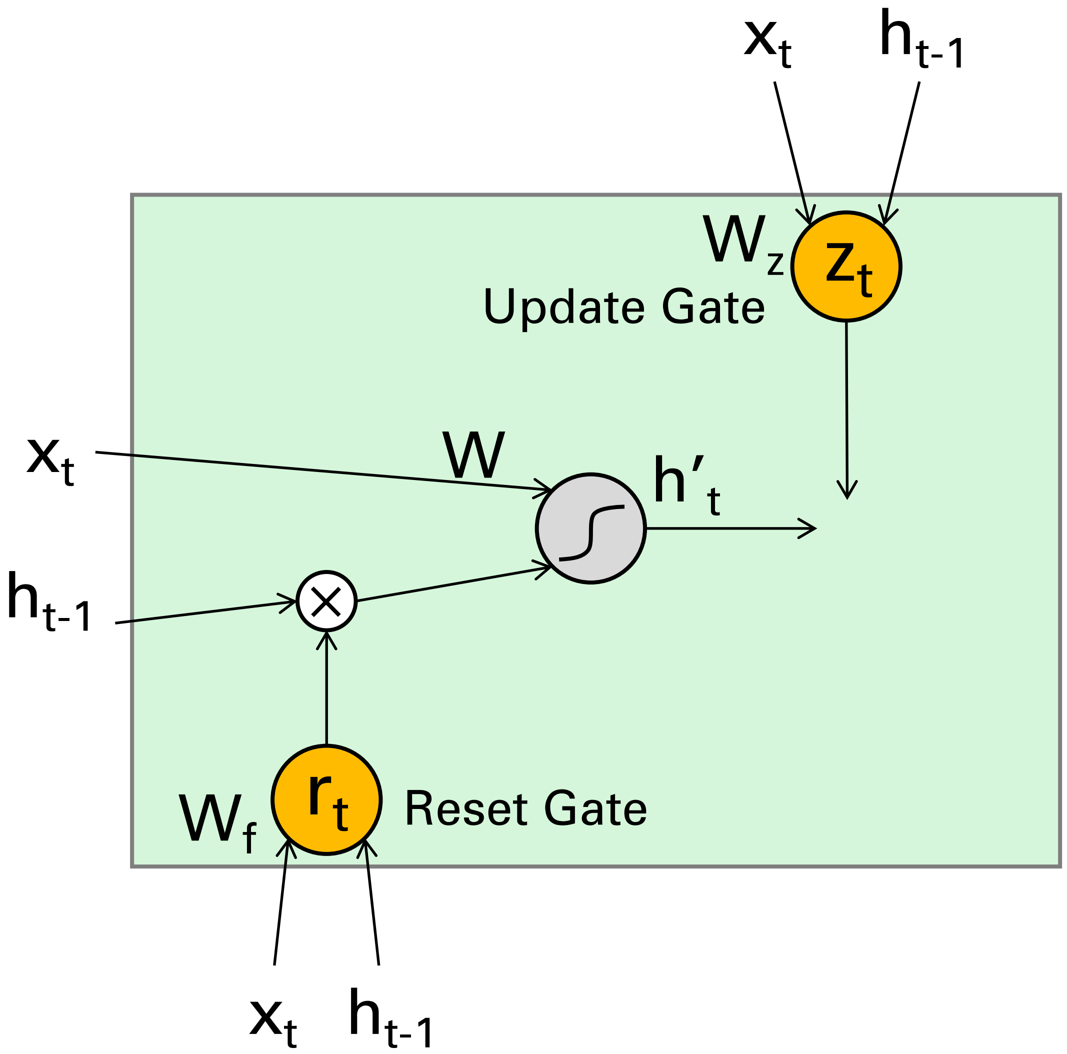
$$r_t = \sigma\left(W_r \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix} + b_f \right)$$

$$h'_t = \tanh W \begin{pmatrix} x_t \\ r_t \otimes h_{t-1} \end{pmatrix}$$

computes the **hidden state** based on current input and hidden state

if reset gate unit is ~0, then this ignores previous memory and only stores the new input information

GRU



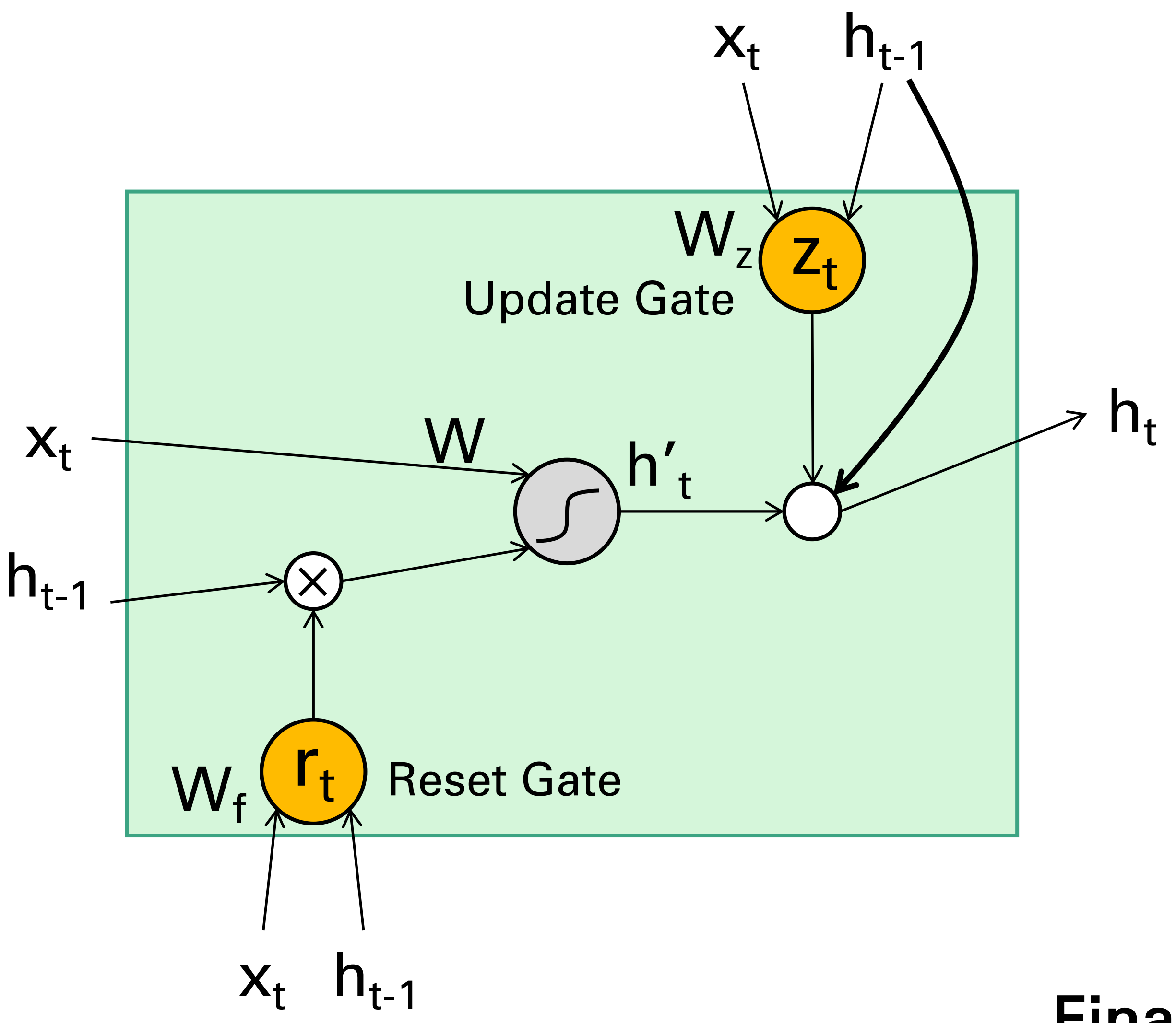
$$r_t = \sigma\left(W_r \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix} + b_f \right)$$

$$h'_t = \tanh W \begin{pmatrix} x_t \\ r_t \otimes h_{t-1} \end{pmatrix}$$

$$z_t = \sigma\left(W_z \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix} + b_f \right)$$

computes an **update gate** again based on current input and hidden state

GRU



$$r_t = \sigma\left(W_r\begin{pmatrix} x_t \\ h_{t-1}\end{pmatrix} + b_f\right)$$

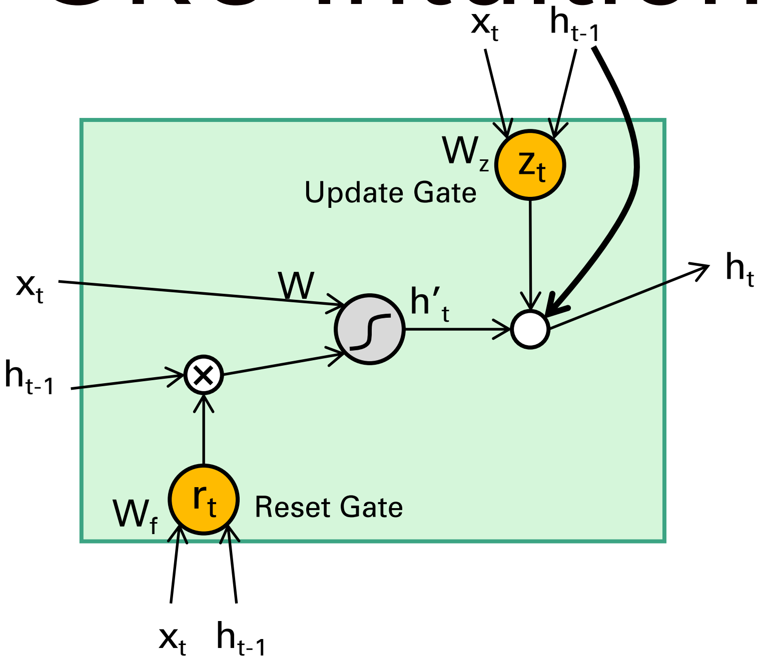
$$h'_t = \tanh W\begin{pmatrix} x_t \\ r_t \otimes h_{t-1}\end{pmatrix}$$

$$z_t = \sigma\left(W_z\begin{pmatrix} x_t \\ h_{t-1}\end{pmatrix} + b_f\right)$$

$$h_t = (1 - z_t) \otimes h_{t-1} + z_t \otimes h'_t$$

Final memory at timestep t combines both current and previous timesteps

GRU Intuition



$$r_t = \sigma\left(W_r \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix} + b_f\right)$$

$$h'_t = \tanh W \begin{pmatrix} x_t \\ r_t \otimes h_{t-1} \end{pmatrix}$$

$$z_t = \sigma\left(W_z \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix} + b_f\right)$$

$$h_t = (1 - z_t) \otimes h_{t-1} + z_t \otimes h'_t$$

- If reset is close to 0, ignore previous hidden state
  - ➢ Allows model to drop information that is irrelevant in the future
- Update gate z controls how much of past state should matter now.
  - – If z close to 1, then we can copy information in that unit through many time steps! **Less vanishing gradient!**
- Units with short-term dependencies often have reset gates very active

Slide credit: Richard Socher

An Empirical Exploration of Recurrent Network Architectures

- Given the rather ad-hoc design of the LSTM, the authors try to determine if the architecture of the LSTM is optimal

- They use an evolutionary search for better architectures

[An Empirical Exploration of Recurrent Network Architectures [Jozefowicz et al., 2015]](#)

Evolutionary Architecture Search

- A list of top-100 architectures so far is maintained, initialized with the LSTM and the GRU
- The GRU is considered as the baseline to beat
- New architectures are proposed, and retained based on performance ratio with GRU

- All architectures are evaluated on 3 problems
  - Arithmetic: Compute digits of sum or difference of two numbers provided as inputs. Inputs have distractors to increase difficulty
    3e36d9-h1h39f94eeh43keg3c = 3369 – 13994433 = -13991064
  - XML Modeling: Predict next character in valid XML modeling
  - Penn Tree-Bank Language Modeling: Predict distributions over words

An Empirical Exploration of Recurrent Network Architectures [Jozefowicz et al., 2015]

Evolutionary Architecture Search

- At each step
  - Select 1 architecture at random, evaluate on 20 randomly chosen hyperparam settings.
  - Alternatively, propose a new architecture by mutating an existing one. Choose prob. p from [0,1] uniformly and apply a transformation to each node with prob. p
    - If node is a non-linearity, replace with {tanh(x), sigmoid(x), ReLU(x), Linear(0, x), Linear(1, x), Linear(0.9, x), Linear(1.1, x)}
    - If node is an elementwise op, replace with {multiplication, addition, subtraction}
    - Insert random activation function between node and one of its parents
    - Replace node with one of its ancestors (remove node)
    - Randomly select a node (node A). Replace the current node with either the sum, product, or difference of a random ancestor of the current node and a random ancestor of A.
  - Add architecture to list based on minimum relative accuracy wrt GRU on 3 different tasks

An Empirical Exploration of Recurrent Network Architectures [Jozefowicz et al., 2015]

Evolutionary Architecture Search

- 3 novel architectures are presented in the paper
- Very similar to GRU, but slightly outperform it

- **LSTM initialized with a large positive forget gate bias outperformed both the basic LSTM and the GRU!**

An Empirical Exploration of Recurrent Network Architectures [Jozefowicz et al., 2015]

LSTM initialized with large positive forget gate bias?

- Recall

$$f_t = \sigma\left(W_f \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix} + b_f\right)$$

$$c_t = f_t \otimes c_{t-1} + i_t \otimes \tanh W \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix}$$

$$\delta c_{t-1} = \delta c_t \otimes f_t$$

- Gradients will vanish if f is close to 0. Using a large positive bias ensures that f has values close to 1, especially when training begins
- Helps learn long-range dependencies
- Originally stated in Learning to forget: Continual prediction with LSTM [Gers et al., 2000], but forgotten over time

An Empirical Exploration of Recurrent Network Architectures [Jozefowicz et al., 2015]

LSTMs and GRUs

Good

- Careful initialization and optimization of vanilla RNNs can enable them to learn long(ish) dependencies, but gated additive cells, like the LSTM and GRU, often just work.

Bad

- LSTMs and GRUs have considerably more parameters and computation per memory cell than a vanilla RNN, as such they have less memory capacity per parameter*

An LSTM with large positive forget gate bias works best!

*Capacity and Trainability in Recurrent Neural Networks. [Collins et al., arXiv 2016]

Next lecture:
Attention and Memory