Lecture 7:
Recurrent Neural Networks

Course Logistics

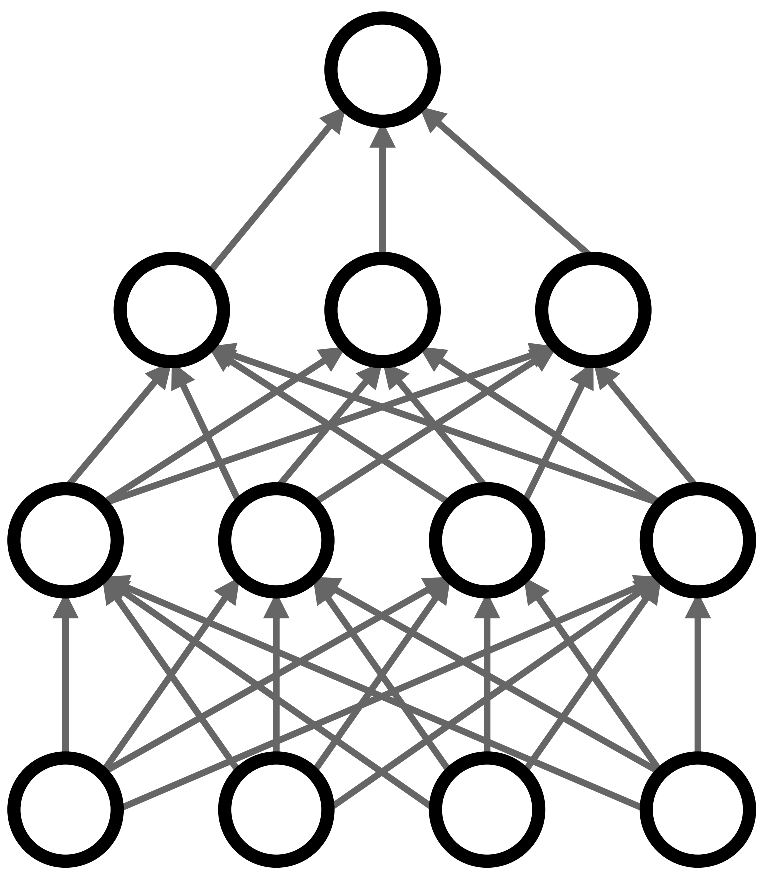
- Assignment 1 is due **tomorrow** (4/23) at 11:59PM!
- Project proposal deadline is due this **Friday** (4/25)

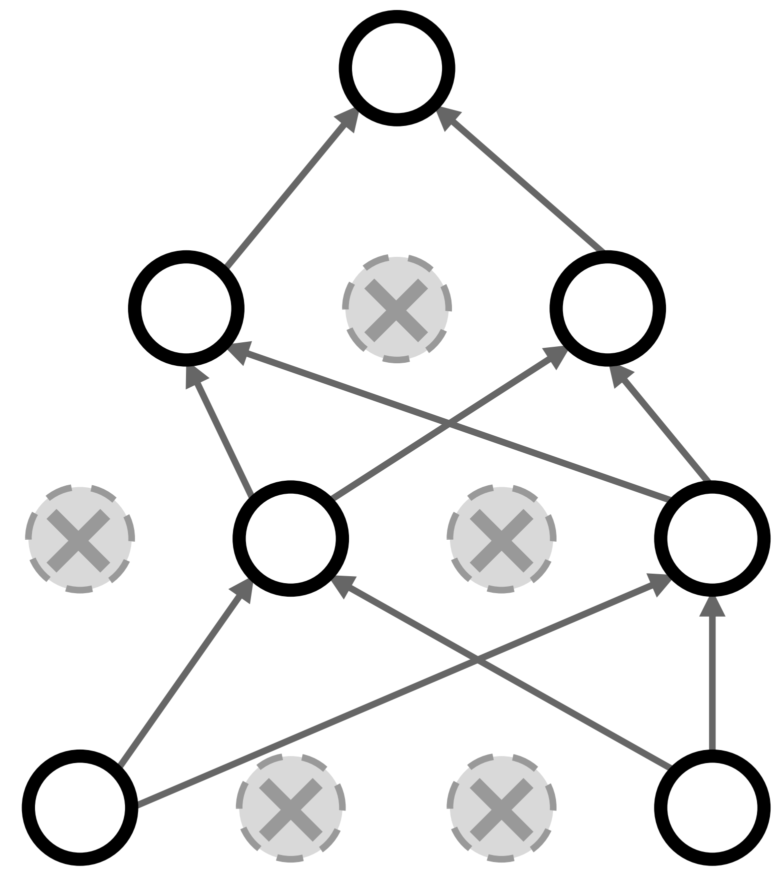
Clarifications from Last Time

- Dropout, how to scale probabilities at test-time
- Question in class about normalization vs weight initialization

Regularization: Dropout

In each forward pass, randomly set some neurons to zero
Probability of dropping is a hyperparameter; 0.5 is common





Srivastava et al, “Dropout: A simple way to prevent neural networks from overfitting”, JMLR 2014

Dropout: Test time

```
def predict(X):
  # ensembled forward pass
  H1 = np.maximum(0, np.dot(W1, X) + b1) * p # NOTE: scale the activations
  H2 = np.maximum(0, np.dot(W2, H1) + b2) * p # NOTE: scale the activations
  out = np.dot(W3, H2) + b3
```

At test time all neurons are active always
=> We must scale the activations so that for each neuron:
<u>output at test time</u> = <u>expected output at training time</u>

Dropout Summary

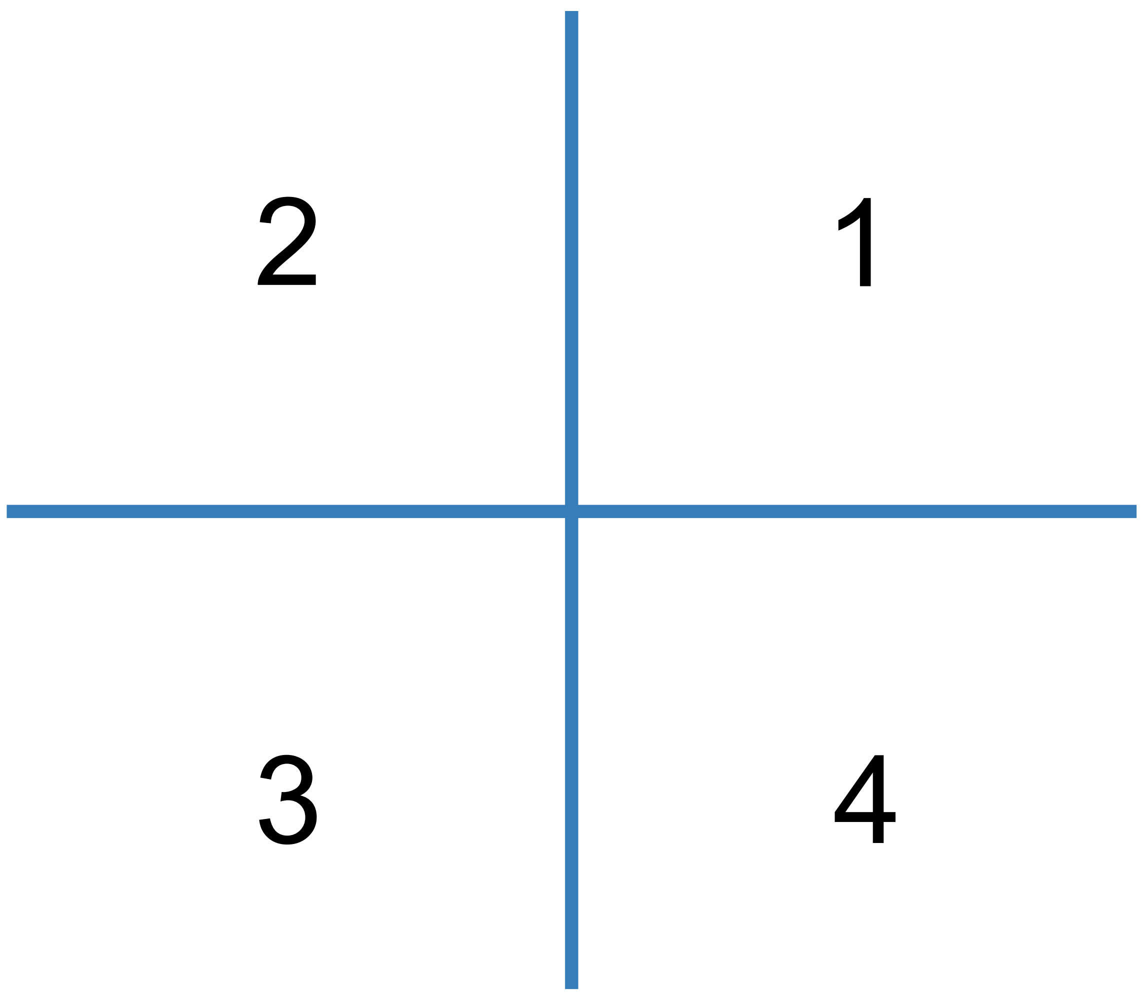
```python
""" Vanilla Dropout: Not recommended implementation (see notes below) """

p = 0.5 # probability of keeping a unit active. higher = less dropout

def train_step(X):
  """ X contains the data """

  # forward pass for example 3-layer neural network
  H1 = np.maximum(0, np.dot(W1, X) + b1)
  U1 = np.random.rand(*H1.shape) < p # first dropout mask
  H1 *= U1 # drop!
  H2 = np.maximum(0, np.dot(W2, H1) + b2)
  U2 = np.random.rand(*H2.shape) < p # second dropout mask
  H2 *= U2 # drop!
  out = np.dot(W3, H2) + b3

  # backward pass: compute gradients... (not shown)
  # perform parameter update... (not shown)

def predict(X):
  # ensembled forward pass
  H1 = np.maximum(0, np.dot(W1, X) + b1) * p # NOTE: scale the activations
  H2 = np.maximum(0, np.dot(W2, H1) + b2) * p # NOTE: scale the activations
  out = np.dot(W3, H2) + b3
```

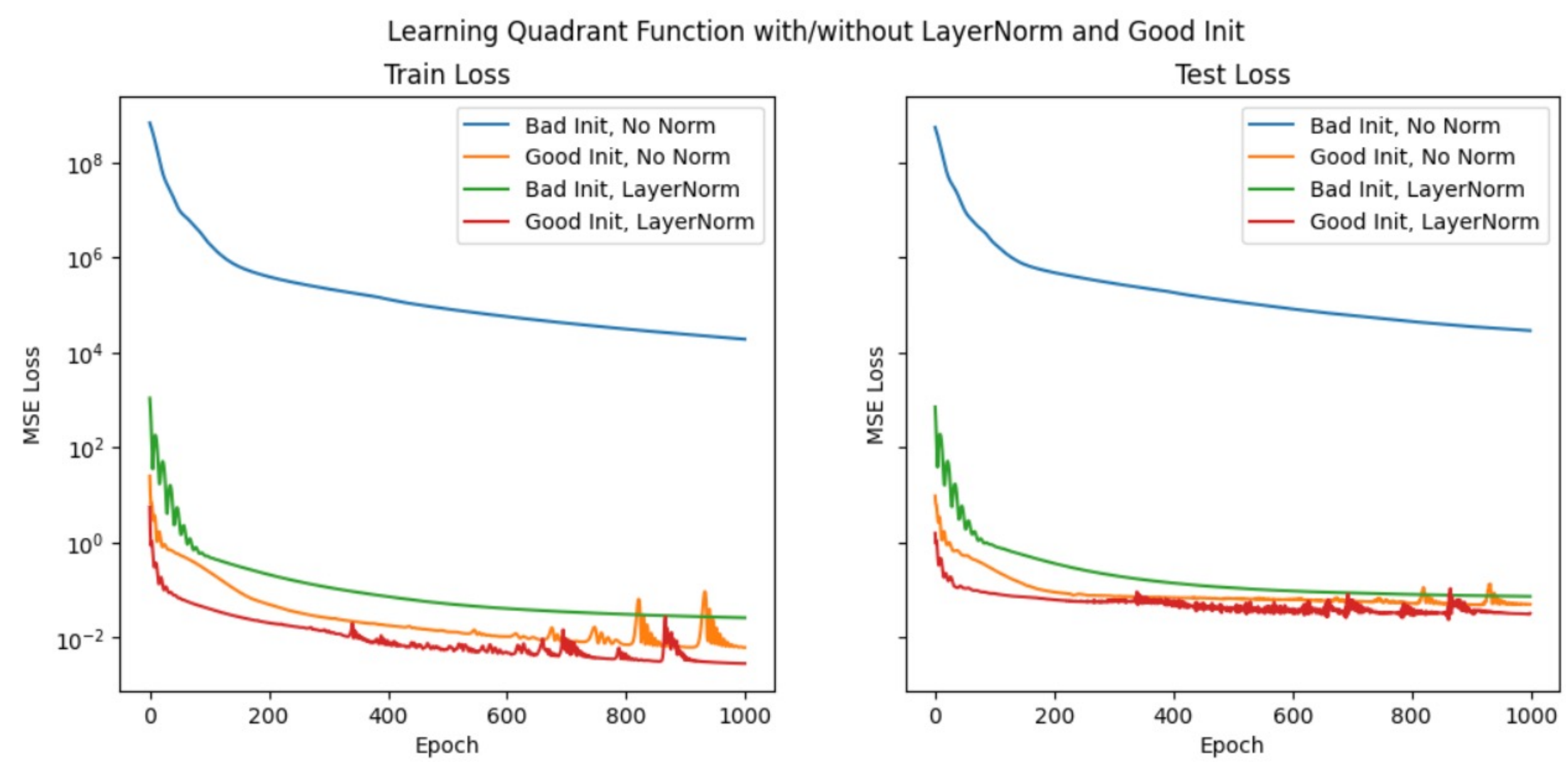
drop in train time

scale at test time

Question in class: can normalization resolve the issues that arise with having weights initialized incorrectly?

Toy setting, 2d input and 2-layer NN w/ ReLU

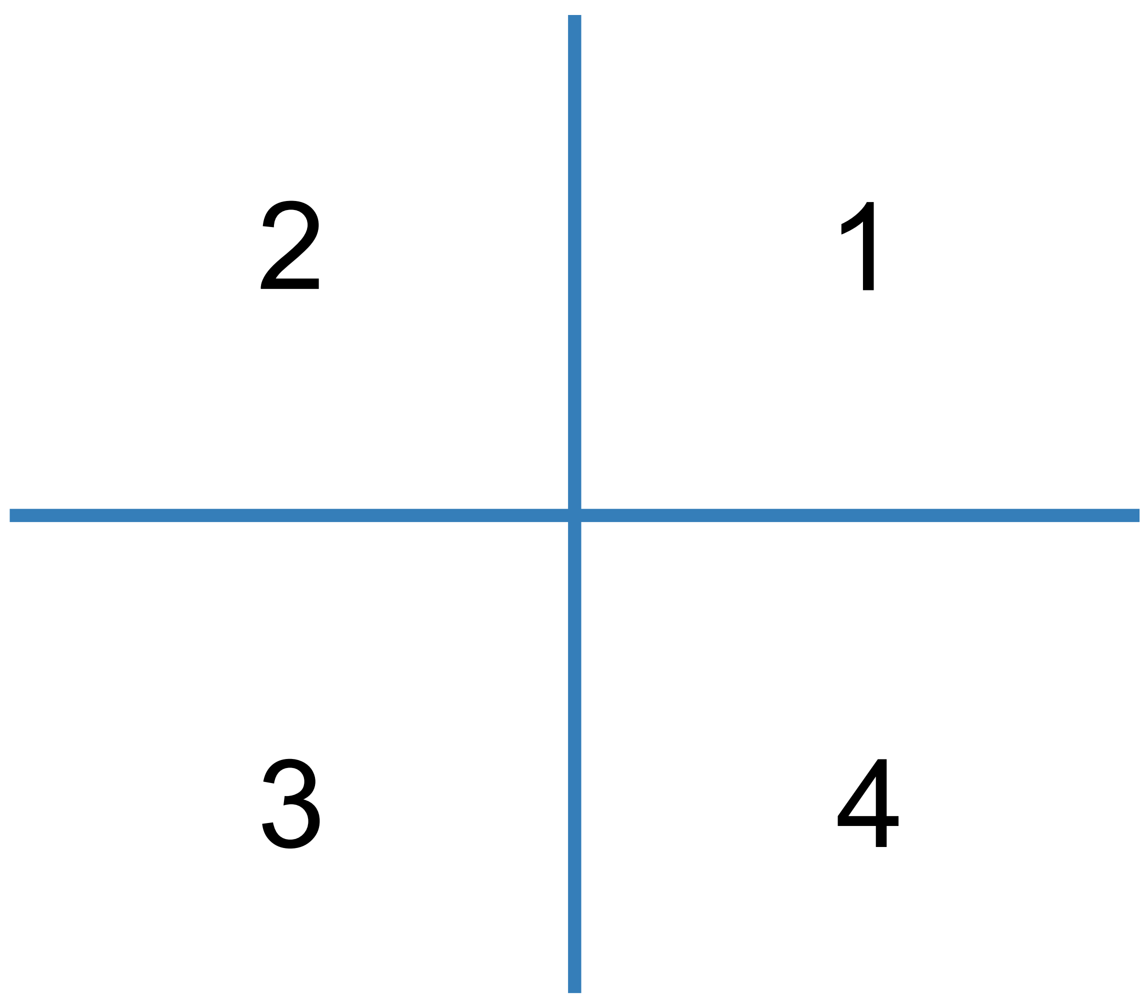




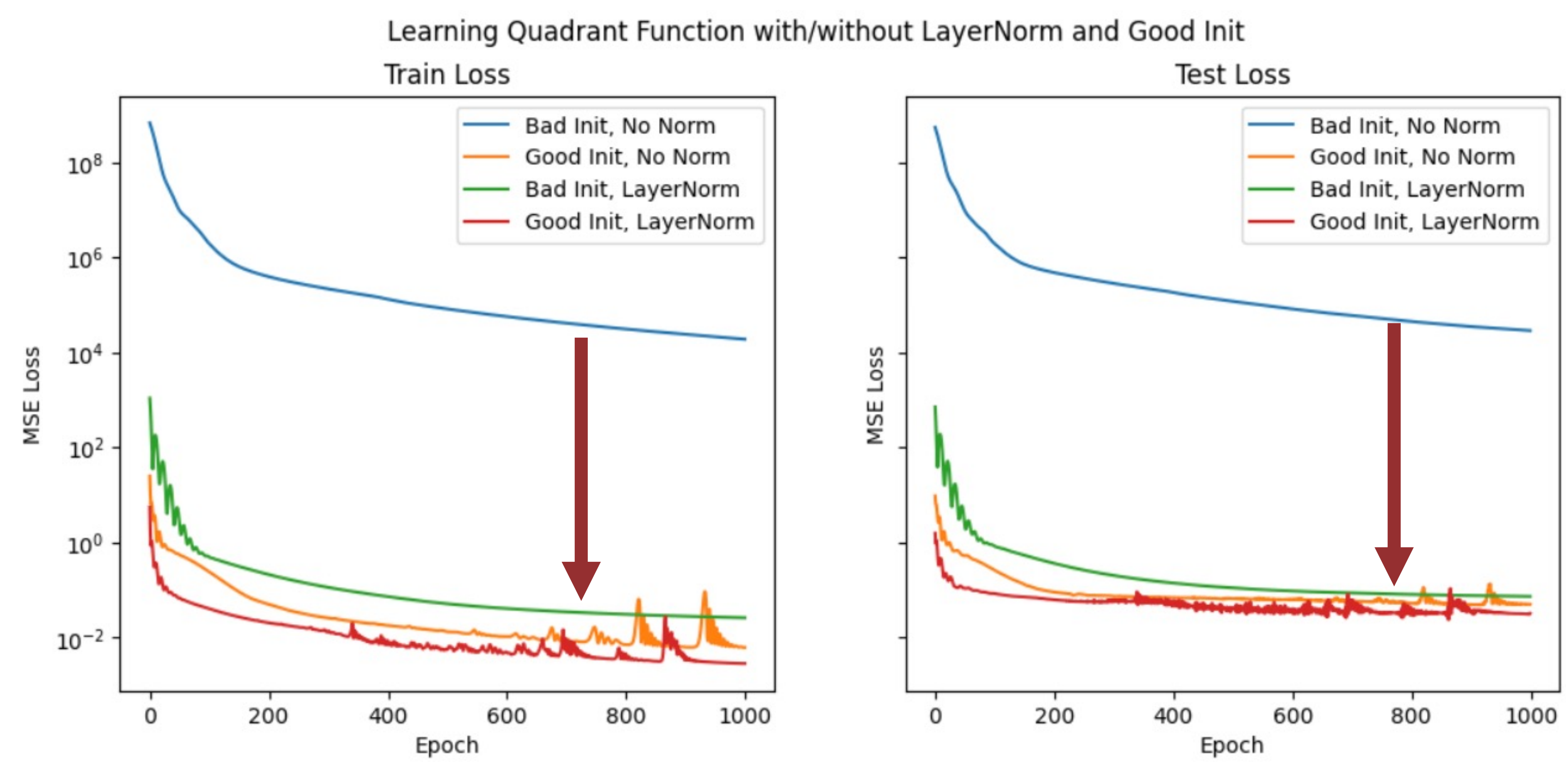
[Link to code (colab)](#) [Link to Ed Post](#)

Question in class: can normalization resolve the issues that arise with having weights initialized incorrectly?

Toy setting, 2d input and 2-layer NN w/ ReLU



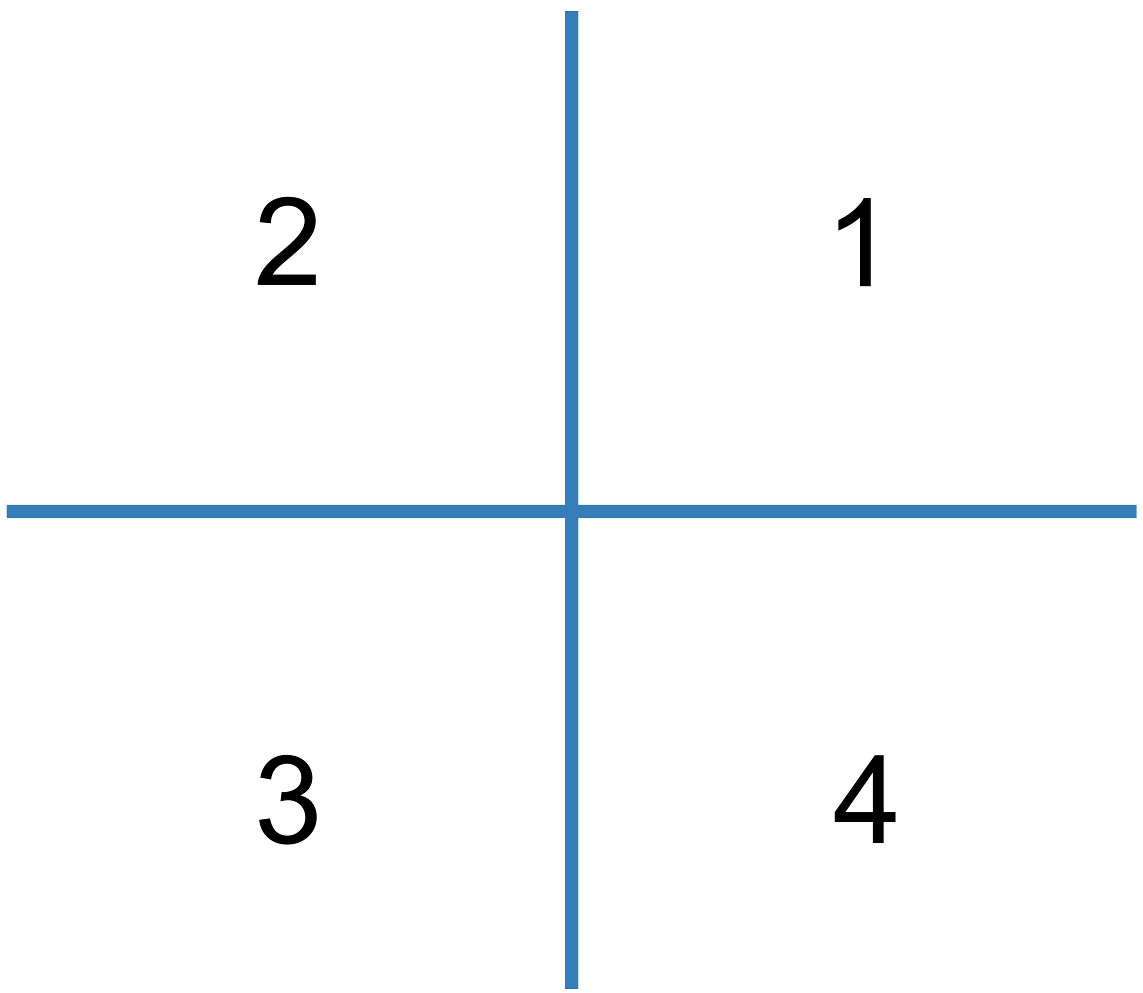
Significant reduction in error!



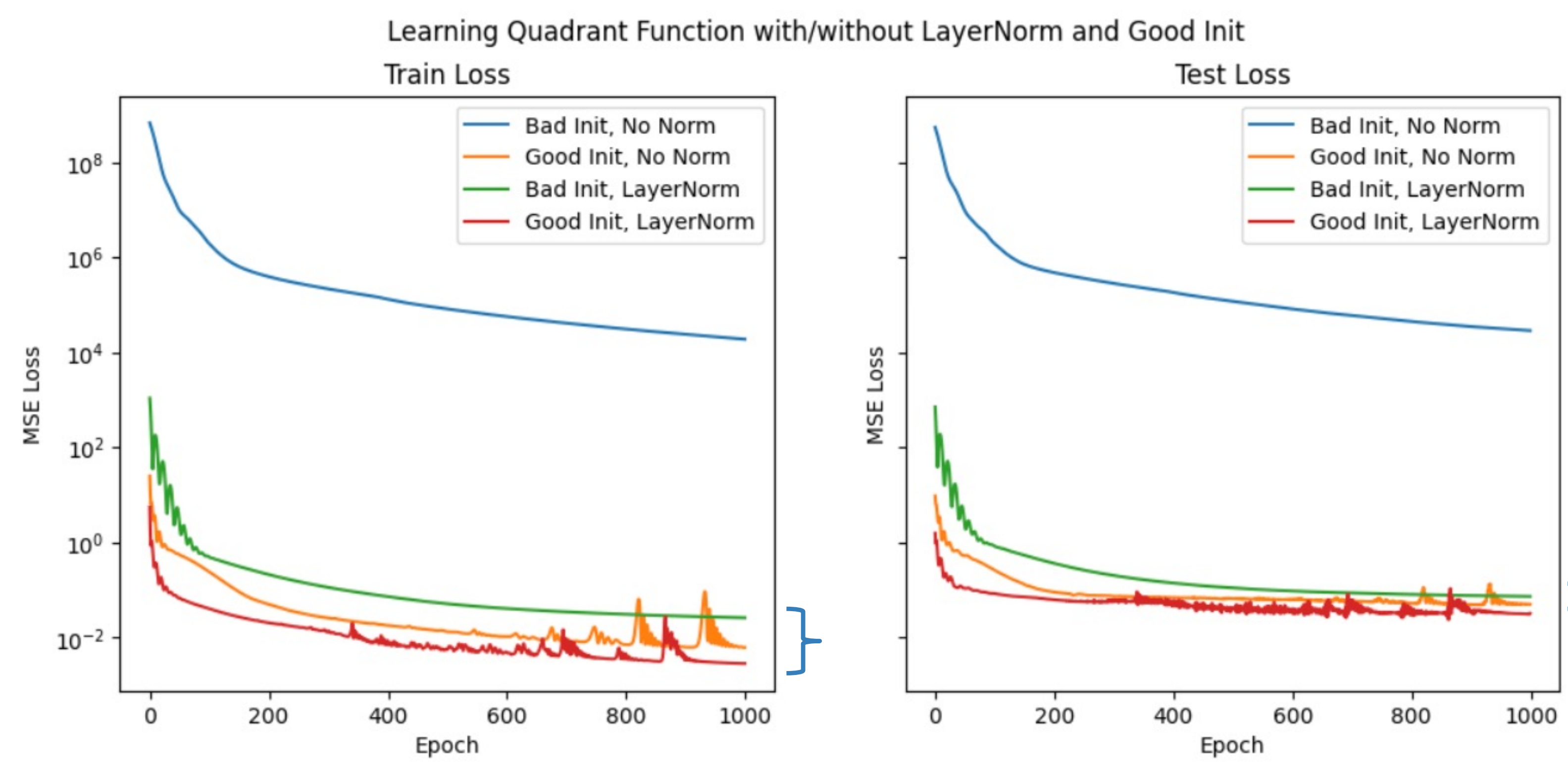
[Link to code (colab)] [Link to Ed Post]

Question in class: can normalization resolve the issues that arise with having weights initialized incorrectly?

Toy setting, 2d input and 2-layer NN w/ ReLU



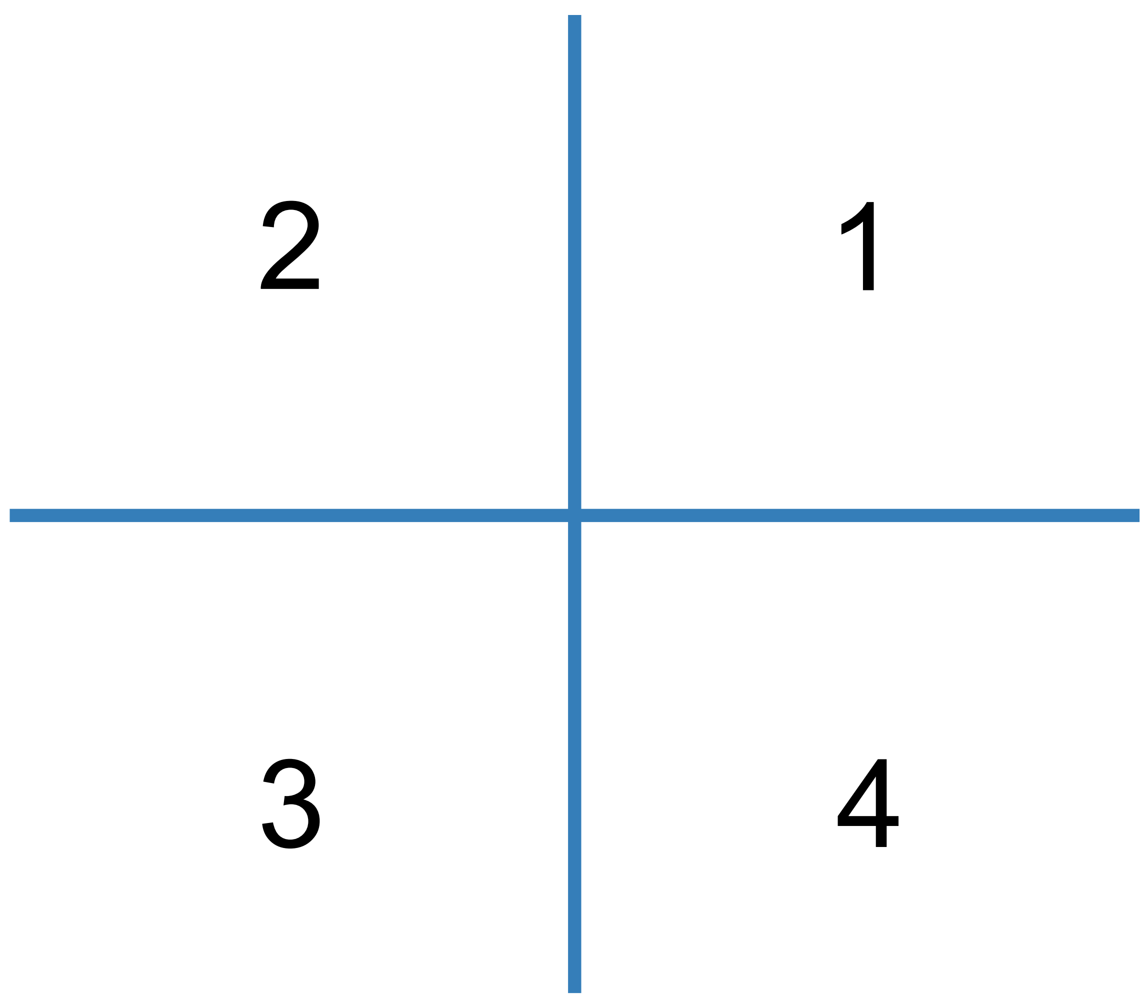
Performance gap still exists, does not resolve issues entirely, still optimization issues



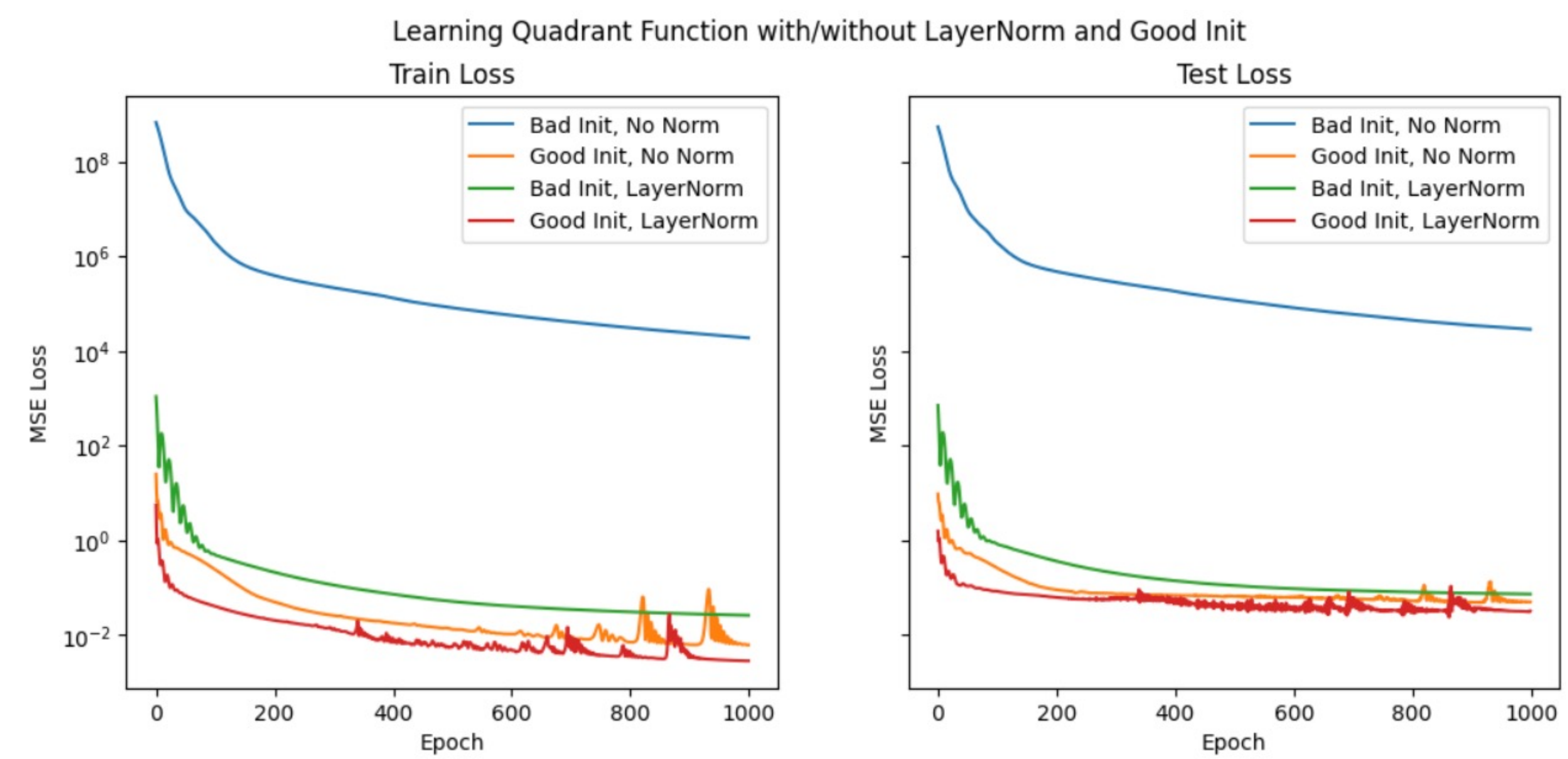
[Link to code (colab)] [Link to Ed Post]

Question in class: can normalization resolve the issues that arise with having weights initialized incorrectly?

Toy setting, 2d input and 2-layer NN w/ ReLU



Normalization may not always make sense! In this case, easy to see why it's helpful (LayerNorm does not change quadrant of inputs)



Link to code (colab)   Link to Ed Post

Training Non-Recurrent Neural Networks

1. **One time setup:** activation functions, preprocessing, weight initialization, normalization, transfer learning

2. **Training dynamics:** babysitting the learning process, parameter updates, hyperparameter optimization

3. **Evaluation:** validation performance, test-time augmentation

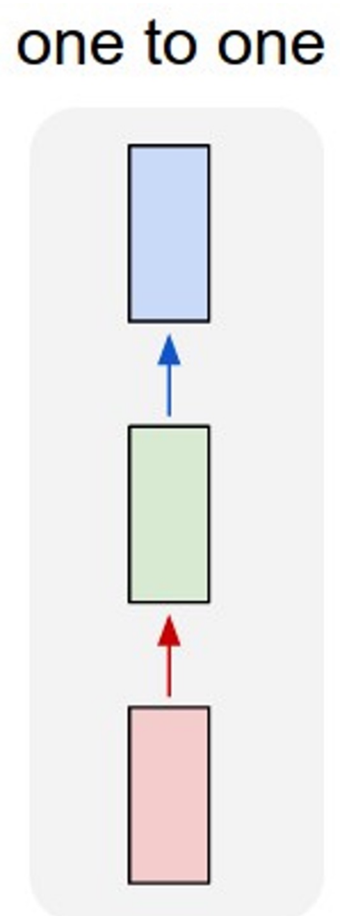
Evaluate models and tune hyperparameters

https://docs.wandb.ai/guides/track/app

Rest of Today’s Lecture

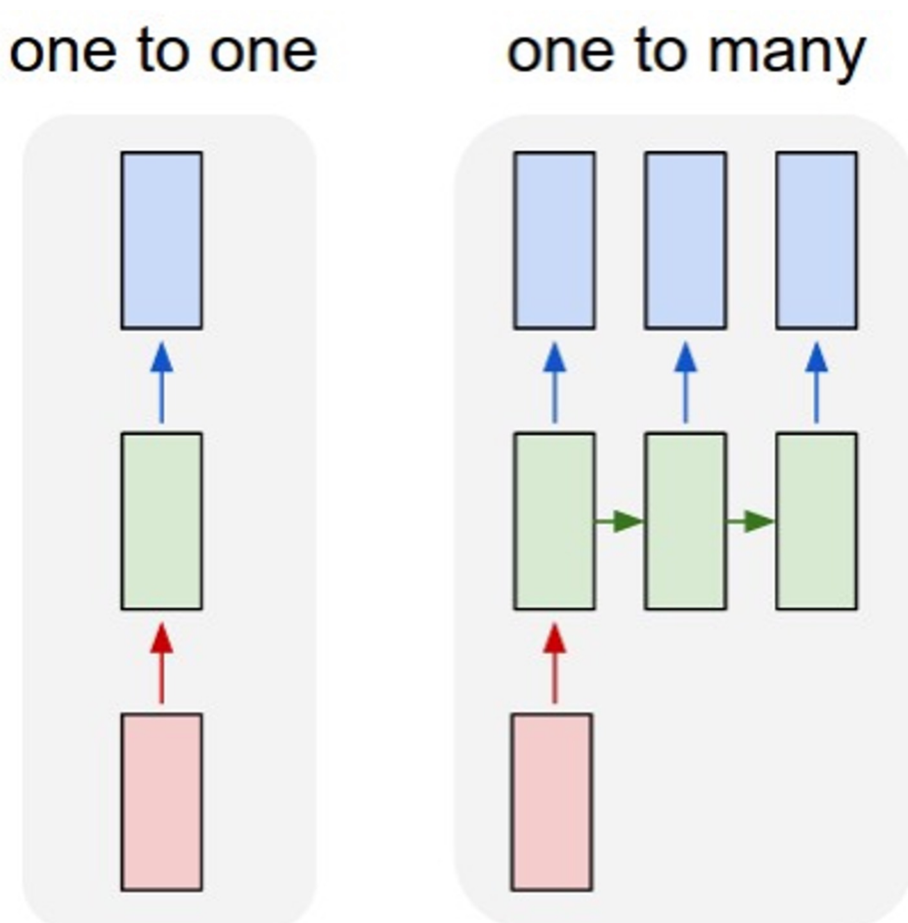
- Discuss sequence modeling (assumed fixed-length inputs so far)
- Simple models commonly used before the era of transformers
    - RNNs and some variants
- Relation to modern state-space models (e.g. Mamba)

“Vanilla” Neural Network



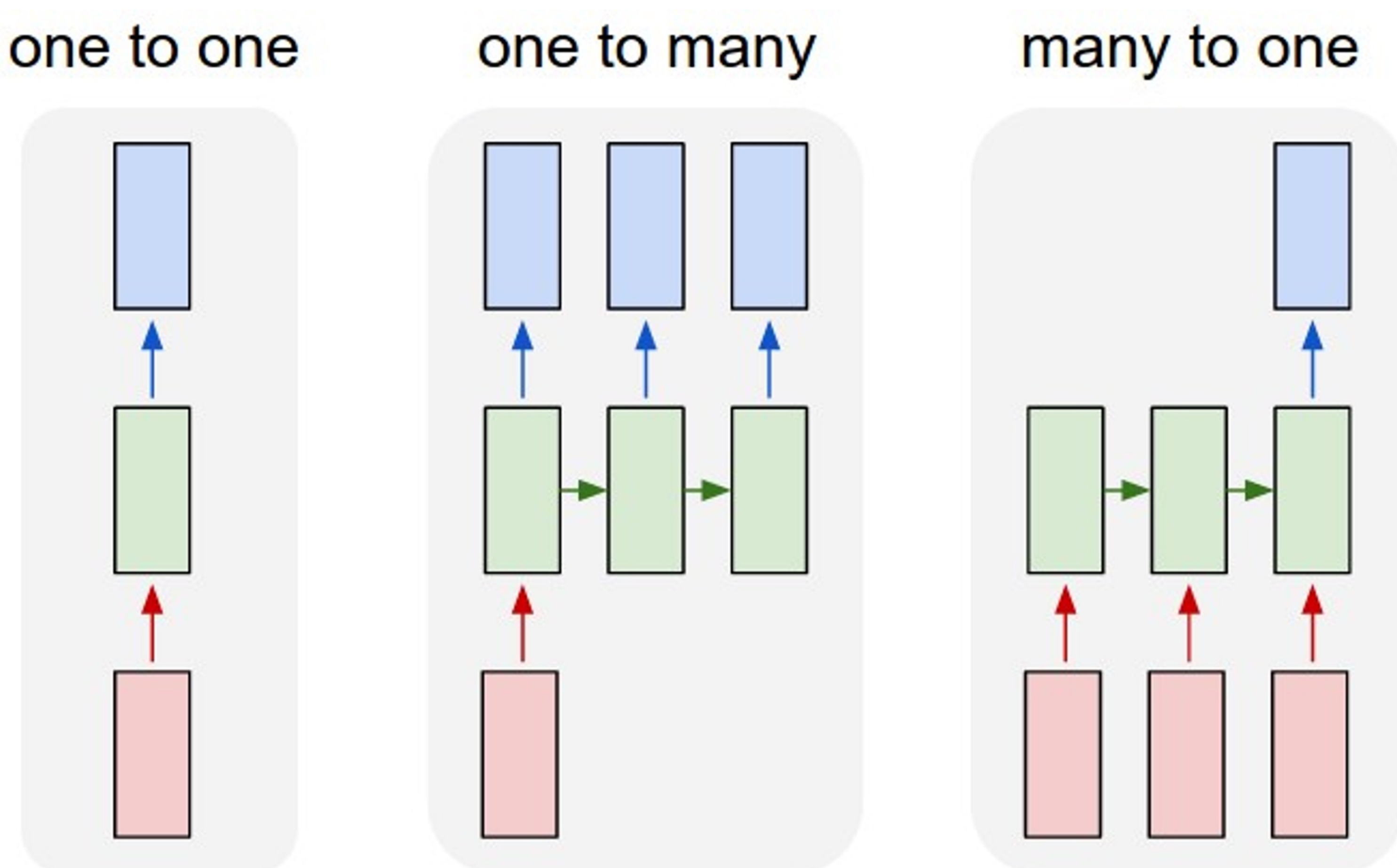
Vanilla Neural Networks

Recurrent Neural Networks: Process Sequences



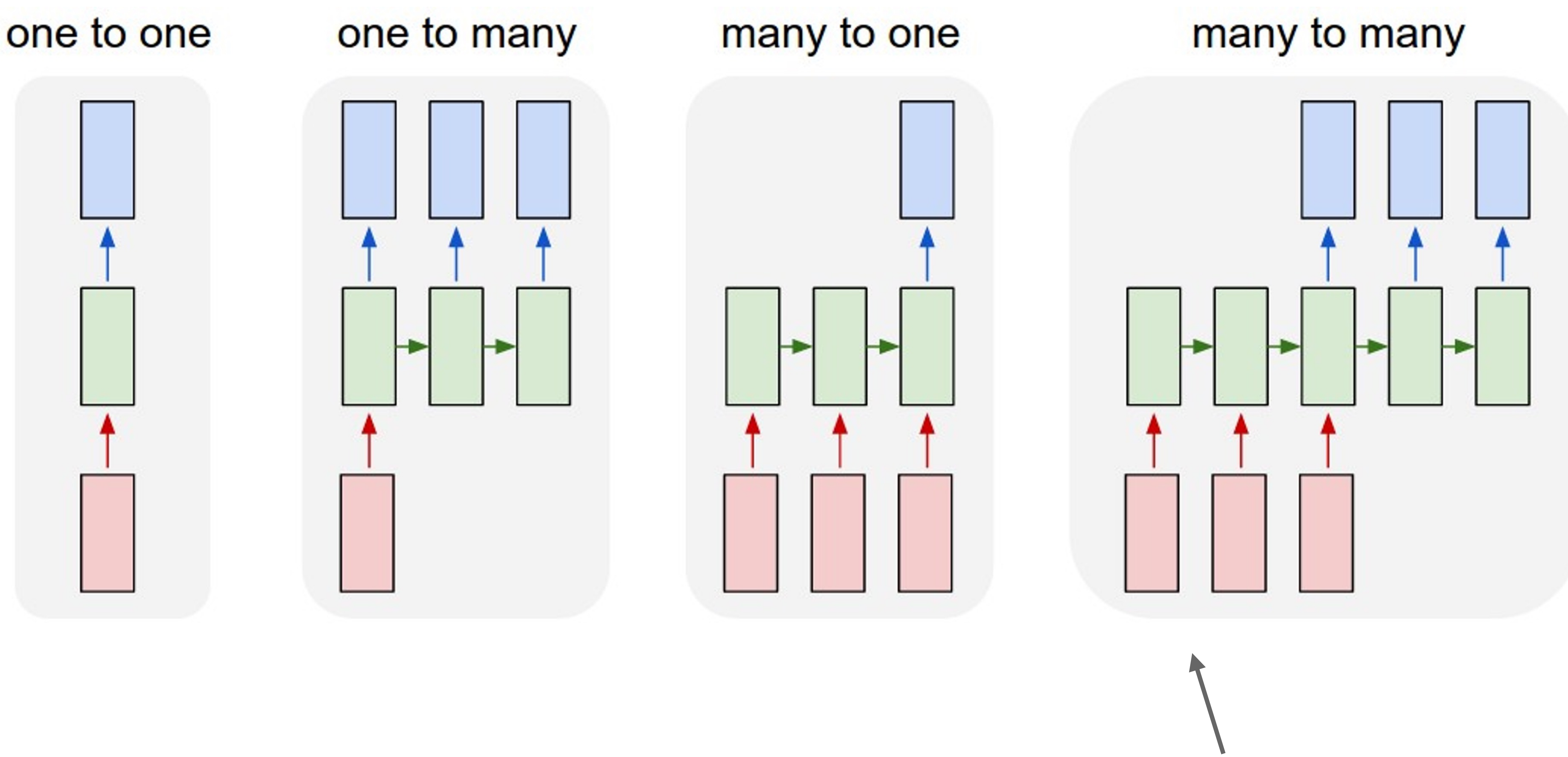
e.g. Image Captioning
image -> sequence of words

Recurrent Neural Networks: Process Sequences



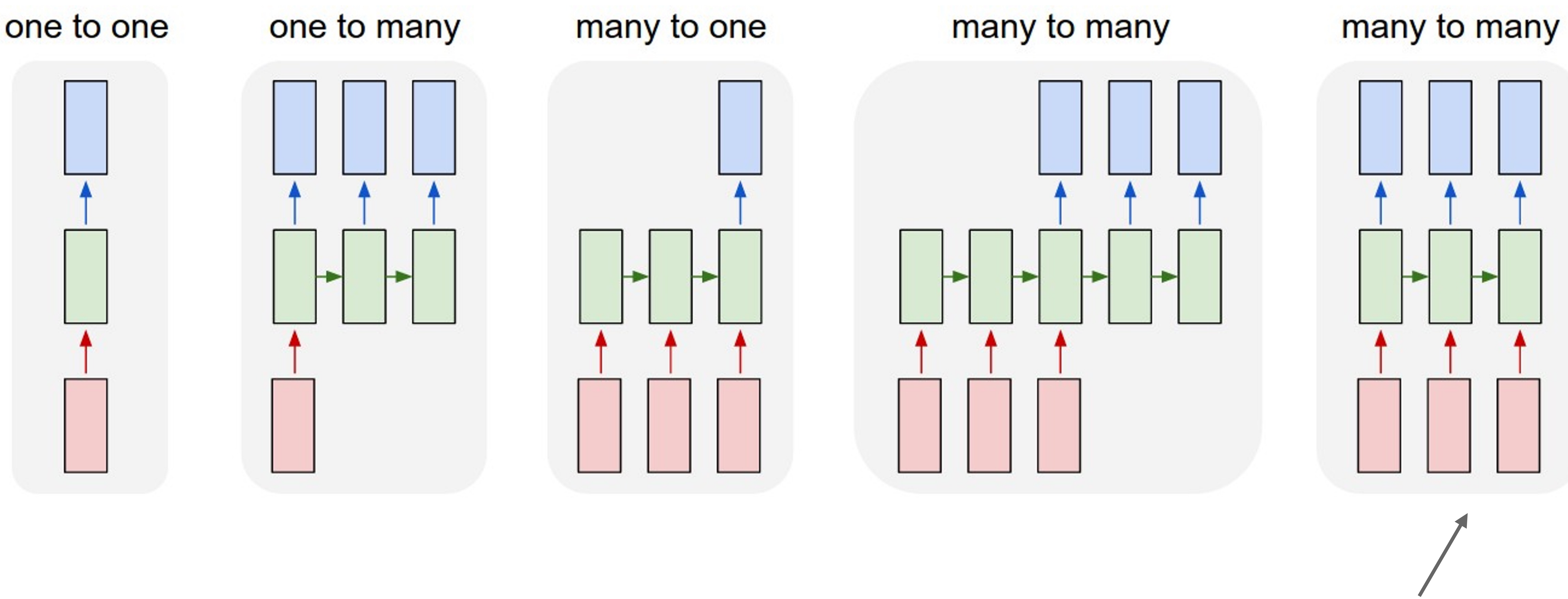
e.g. action prediction
sequence of video frames -> action class

Recurrent Neural Networks: Process Sequences



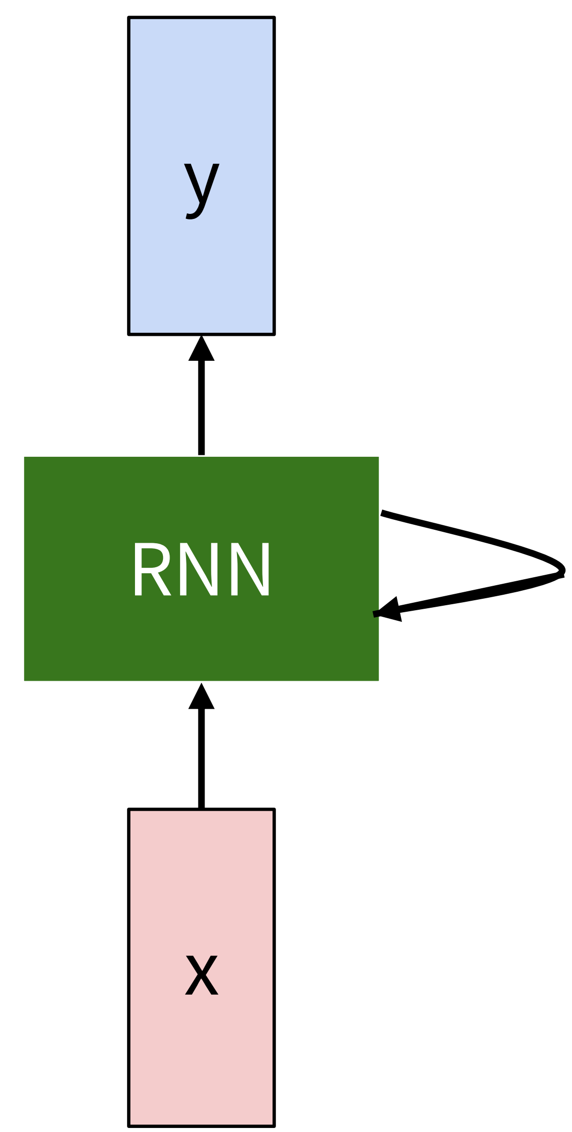
E.g. Video Captioning
Sequence of video frames -> caption

Recurrent Neural Networks: Process Sequences

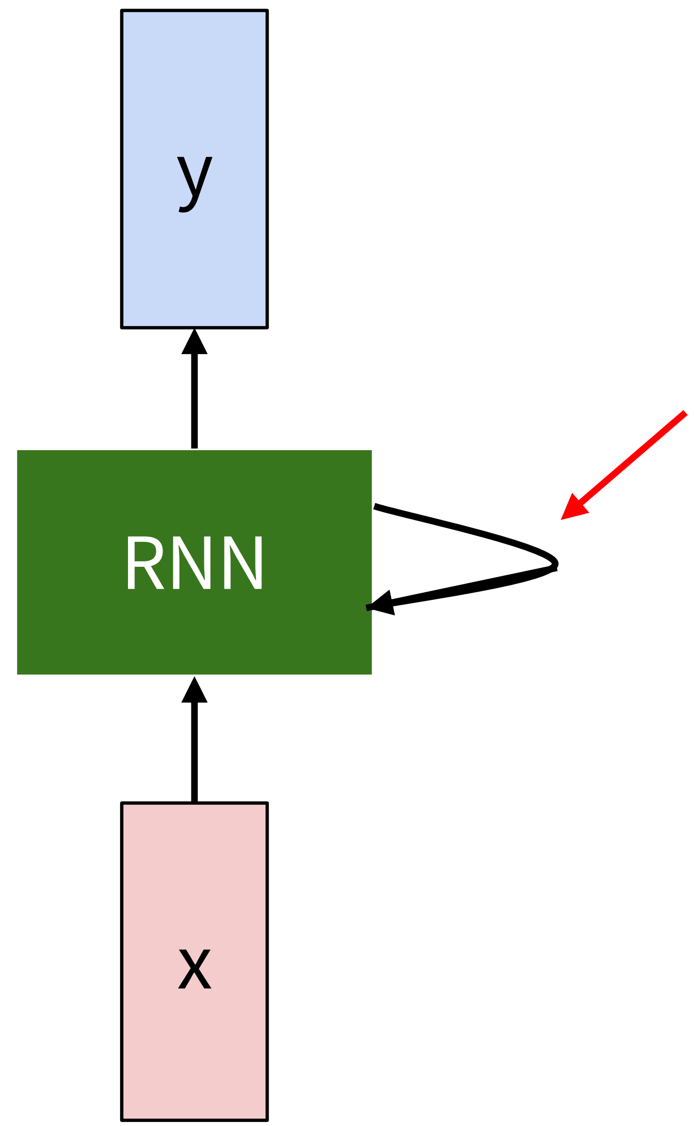


e.g. Video classification on frame level

Recurrent Neural Network

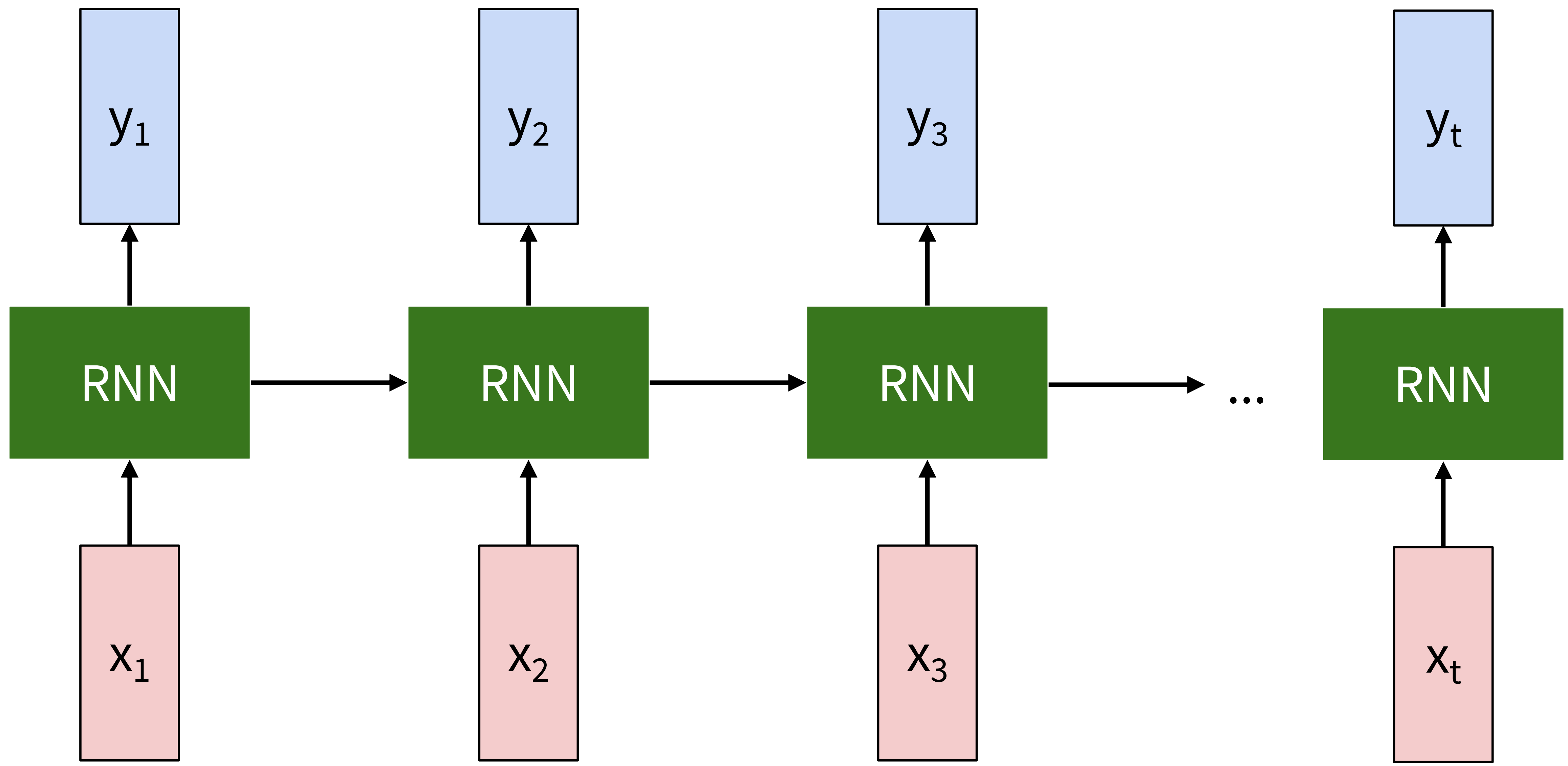


Recurrent Neural Network



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

Unrolled RNN



RNN hidden state update

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

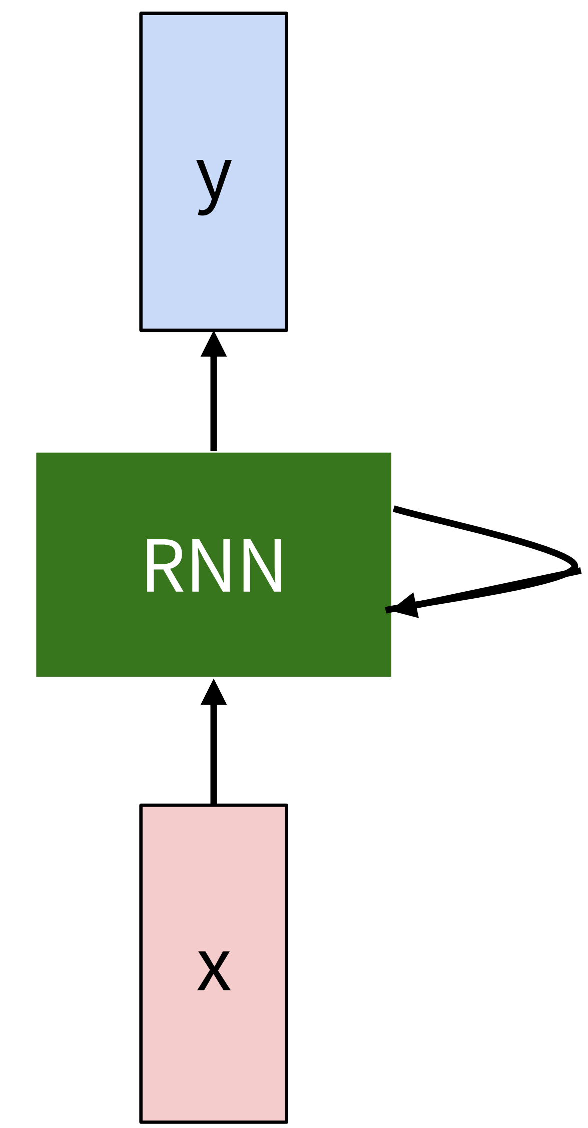
$$h_t = f_W(h_{t-1}, x_t)$$

new state

old state

input vector at some time step

some function with parameters W



RNN output generation

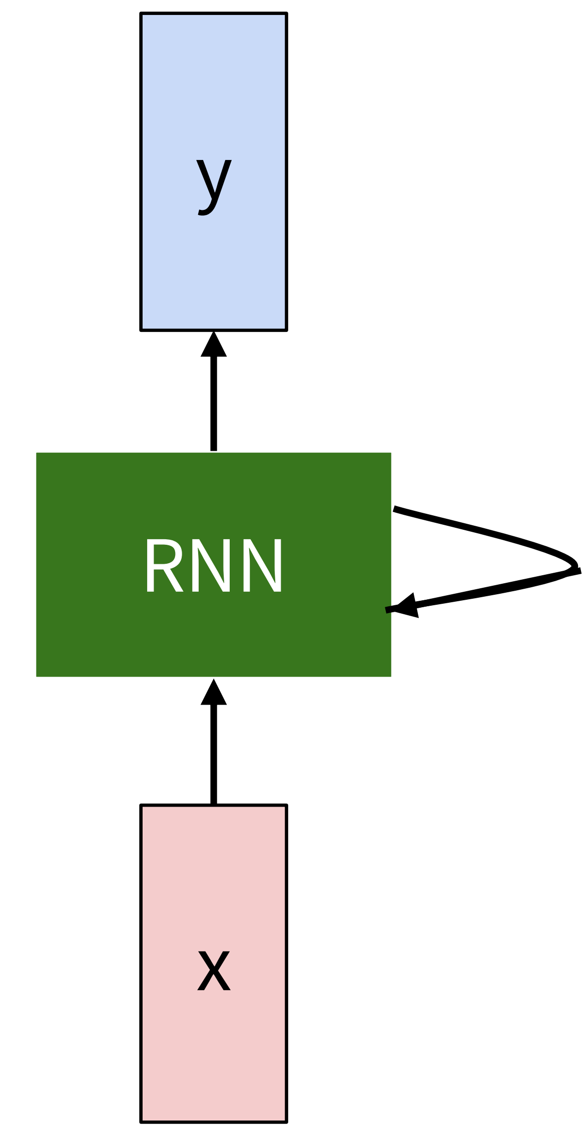
We can process a sequence of vectors x by applying a recurrence formula at every time step:

$$y_t = f_{W_{hy}}(h_t)$$

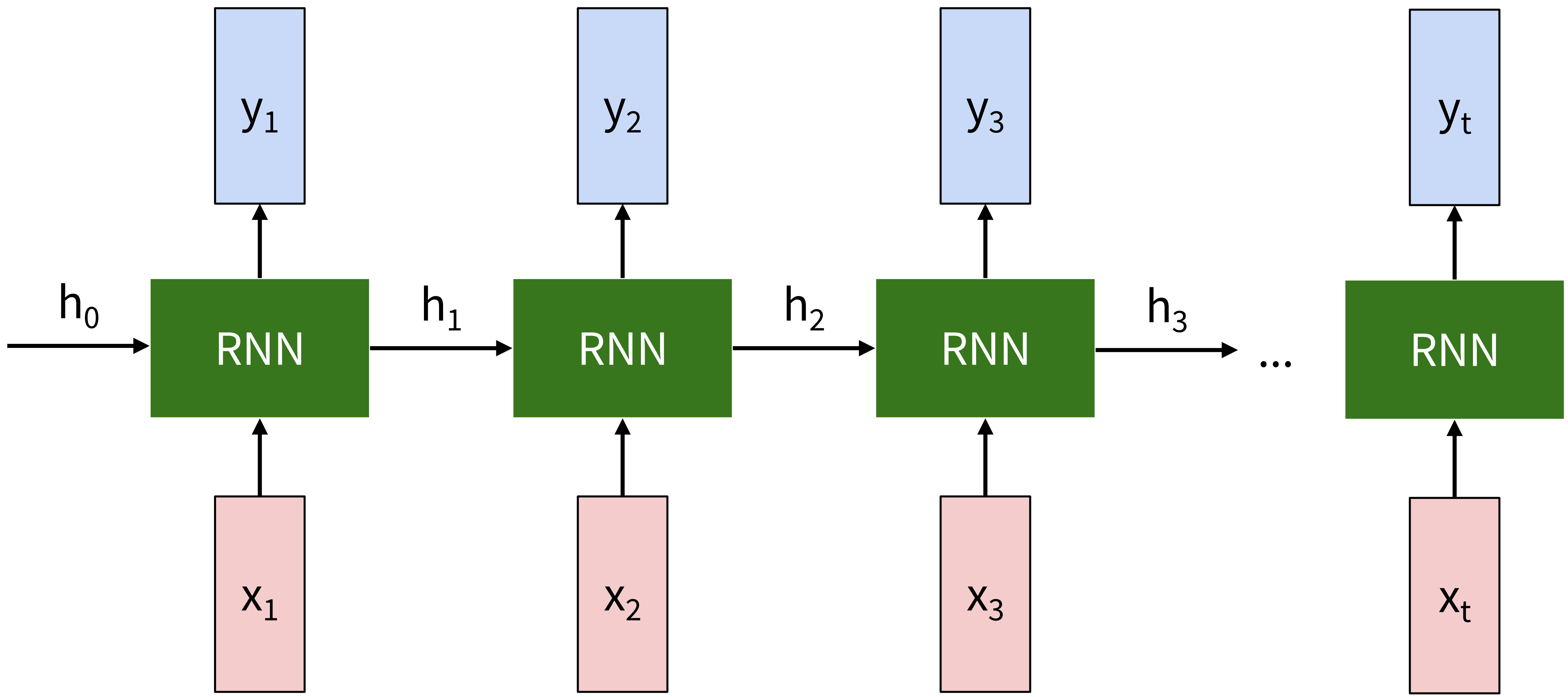
output

new state

another function with parameters W_{hy}



Recurrent Neural Network

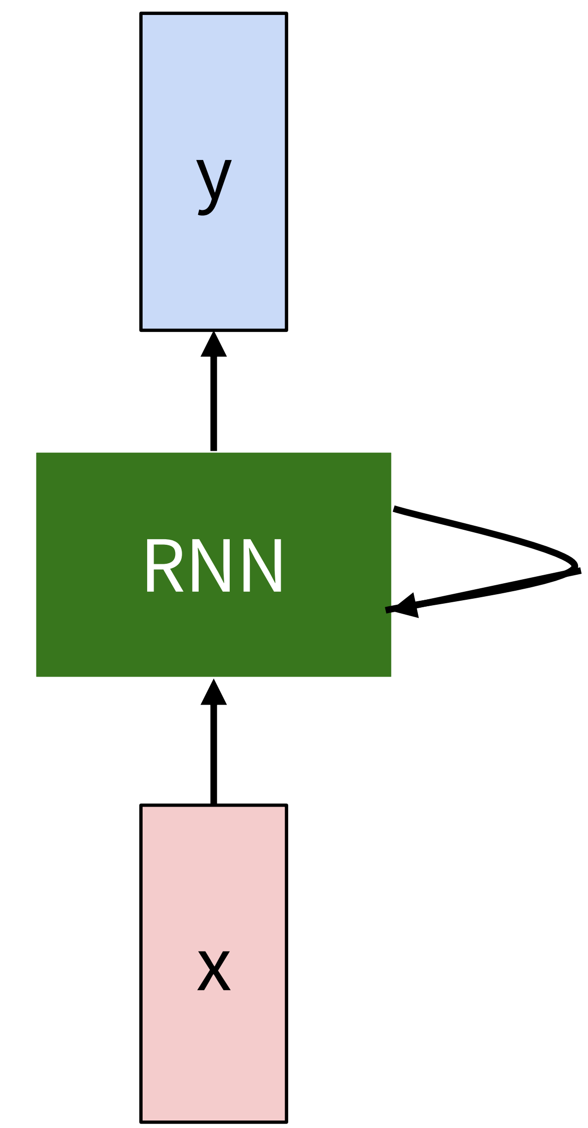


Recurrent Neural Network

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

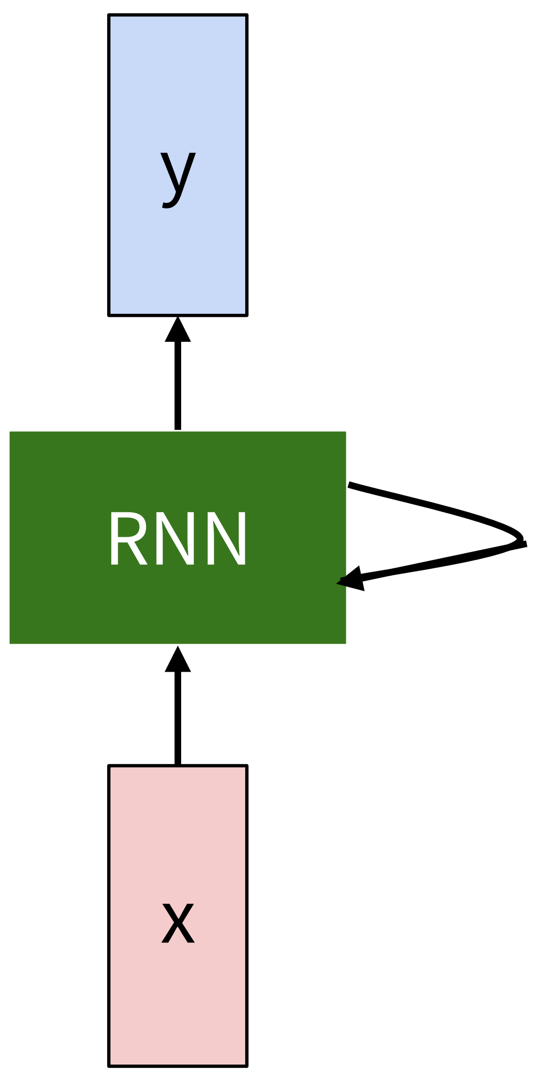
$$h_t = f_W(h_{t-1}, x_t)$$

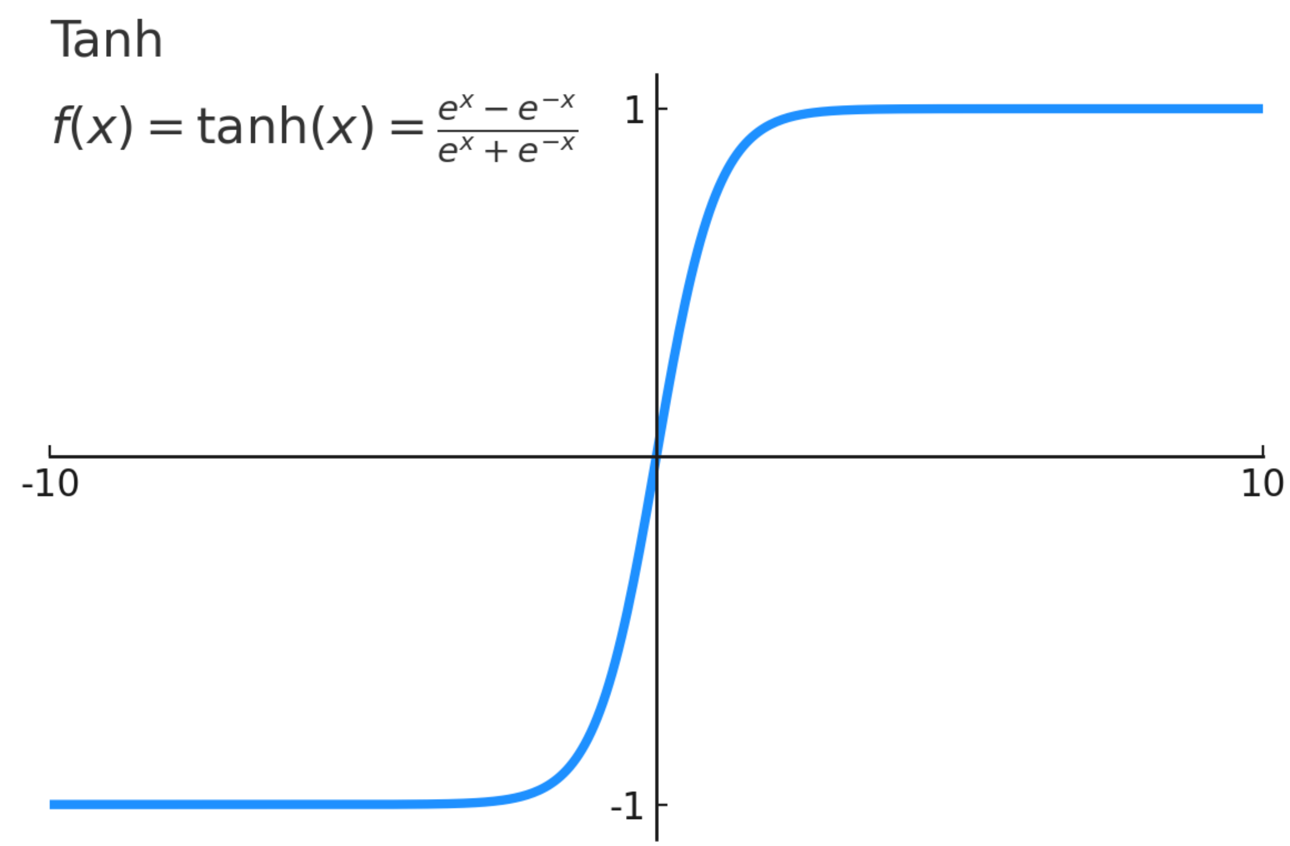
Notice: the same function and the same set of parameters are used at every time step.



(Vanilla) Recurrent Neural Network

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





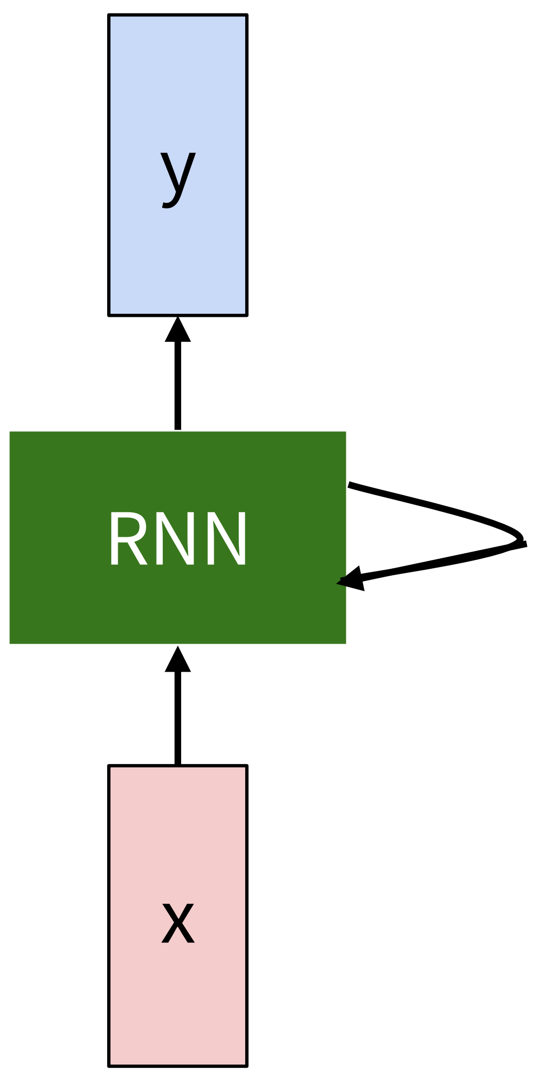
$$h_t = \tanh(W_{hh} h_{t-1} + W_{xh} x_t) \quad y_t = W_{hy} h_t$$

Sometimes called a "Vanilla RNN" or an "Elman RNN" after Prof. Jeffrey Elman

Often, we also have an output function: f_y

Vanilla(-ish) RNN: Concrete Example

Manually creating a recurrent network for detecting repeated 1s



X	RNN	Y
0	→	0
1		0
0		0
1		0
1		**1**
1		**1**
0		0
1		0
1		**1**

"Many to many" sequence modeling task

$$h_t = f_W(h_{t-1}, x_t)$$

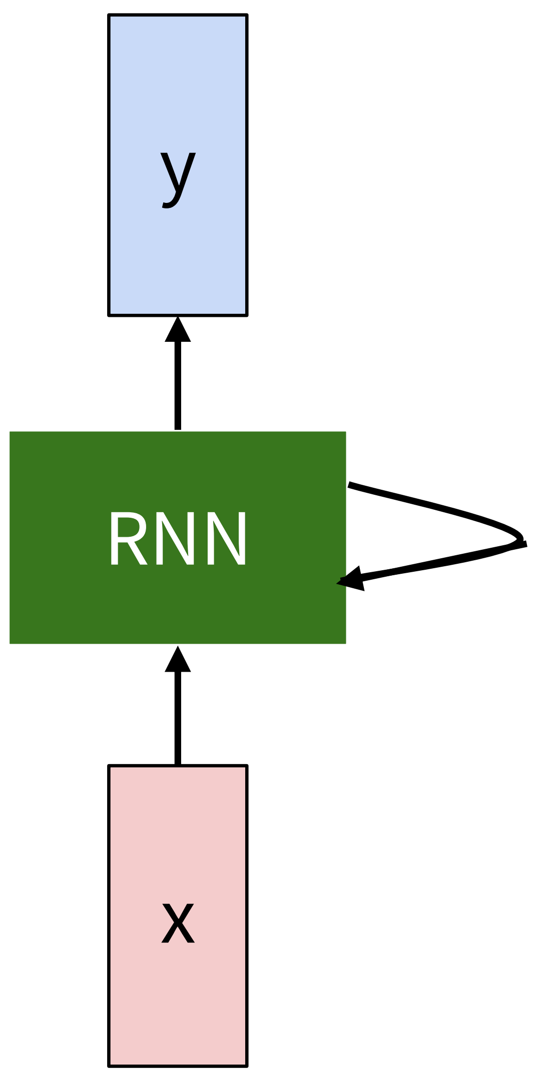
$$y_t = f_y(W_{hy} h_t)$$

How could we create an RNN to do this?

Q: What information should be captured in the "hidden" state?

Vanilla(-ish) RNN: Concrete Example

Manually creating a recurrent network for detecting repeated 1s



RNN

X	Y
0	0
1	0
0	0
1	0
1	**1**
1	**1**
0	0
1	0
1	**1**

"Many to many" sequence modeling task

$$h_t = f_W(h_{t-1}, x_t)$$

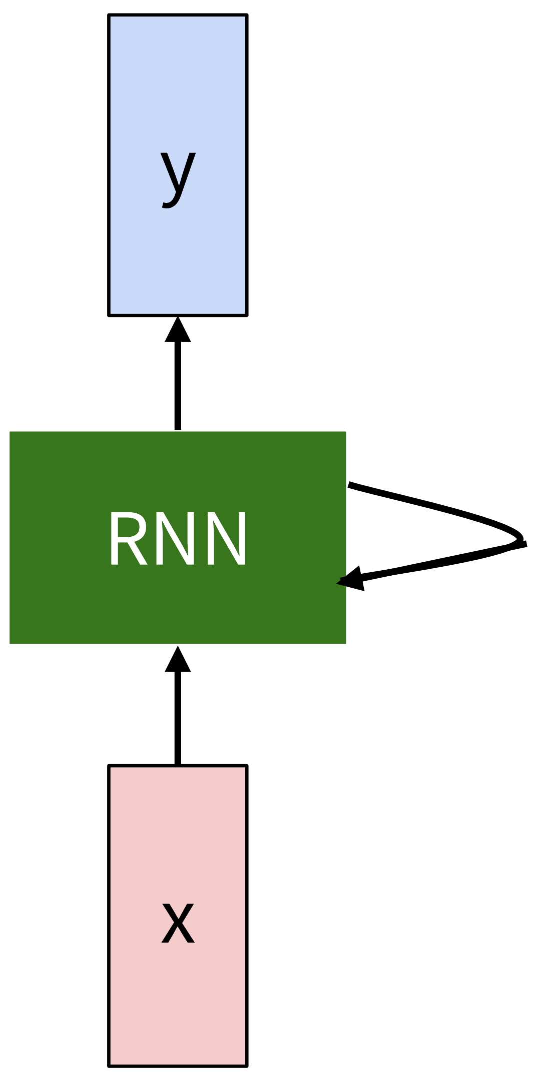
$$y_t = f_y(W_{hy} h_t)$$

How could we create an RNN to do this?

A: Previous input and current value for x

Vanilla(-ish) RNN: Concrete Example

Manually creating a recurrent network for detecting repeated 1s



X	RNN →	Y
0		0
1		0
0		0
1		0
1		**1**
1		**1**
0		0
1		0
1		**1**

"Many to many" sequence modeling task

$$h_t = f_W(h_{t-1}, x_t)$$

$$y_t = f_y(W_{hy} h_t)$$

Set f_W and f_y to both be ReLU for simplicity

$$h_t = \begin{pmatrix} \text{Current} \\ \text{Previous} \\ 1 \end{pmatrix}$$

Initialize h_0 to (0, 0, 1)

Vanilla(-ish) RNN: Concrete Example

Manually creating a recurrent network for detecting repeated 1s

```
w_xh = np.array([[1], [0], [0]])

w_hh = np.array([[0, 0, 0],
                 [1, 0, 0],
                 [0, 0, 1]])

w_yh = np.array([1, 1, -1])

x_seq = [0, 1, 0, 1, 1, 1, 0, 1, 1]

h_t_prev = np.array([[0], [0], [1]])

for t, x in enumerate(x_seq):
    h_t = relu(w_hh @ h_t_prev + (w_xh @ x))
    y_t = relu(w_yh @ h_t)
    h_t_prev = h_t
```

* Code is missing parts, for full code see here

$$h_t = ReLU(W_{hh}h_{t-1} + W_{xh}x_t)$$

$$y_t = ReLU(W_{hy}h_t)$$

$$h_t = \begin{pmatrix} \text{Current} \\ \text{Previous} \\ 1 \end{pmatrix}$$

X	RNN →	Y
0		0
1		0
0		0
1		0
1		**1**
1		**1**
0		0
1		0
1		**1**

Vanilla(-ish) RNN: Concrete Example

Manually creating a recurrent network for detecting repeated 1s

```
w_xh = np.array([[1], [0], [0]])

w_hh = np.array([[0, 0, 0],
                 [1, 0, 0],
                 [0, 0, 1]])

w_yh = np.array([1, 1, -1])

x_seq = [0, 1, 0, 1, 1, 1, 0, 1, 1]

h_t_prev = np.array([[0], [0], [1]])

for t, x in enumerate(x_seq):
    h_t = relu(w_hh @ h_t_prev + (w_xh @ x))
    y_t = relu(w_yh @ h_t)
    h_t_prev = h_t
```

$$h_t = ReLU(W_{hh}h_{t-1} + W_{xh}x_t)$$

$$y_t = ReLU(W_{hy}h_t)$$

$$h_t = \begin{pmatrix} \text{Current} \\ \text{Previous} \\ 1 \end{pmatrix}$$

Right hand term
x=0 → [0, 0, 0]
X=1 → [1, 0, 0]

X	RNN →	Y
0		0
1		0
0		0
1		0
1		**1**
1		**1**
0		0
1		0
1		**1**

* Code is missing parts, for full code see here

Vanilla(-ish) RNN: Concrete Example

Manually creating a recurrent network for detecting repeated 1s

```
w_xh = np.array([[1], [0], [0]])

w_hh = np.array([[0, 0, 0],
                 [1, 0, 0],
                 [0, 0, 1]])

w_yh = np.array([1, 1, -1])

x_seq = [0, 1, 0, 1, 1, 1, 0, 1, 1]

h_t_prev = np.array([[0], [0], [1]])

for t, x in enumerate(x_seq):
    h_t = relu(w_hh @ h_t_prev + (w_xh @ x))
    y_t = relu(w_yh @ h_t)
    h_t_prev = h_t
```

* Code is missing parts, for full code see here

$$h_t = ReLU(W_{hh}h_{t-1} + W_{xh}x_t)$$

$$y_t = ReLU(W_{hy}h_t)$$

$$h_t = \begin{pmatrix} \text{Current} \\ \text{Previous} \\ 1 \end{pmatrix}$$

0 for top row of left term (only use value from right term)

X	RNN →	Y
0		0
1		0
0		0
1		0
1		**1**
1		**1**
0		0
1		0
1		**1**

Vanilla(-ish) RNN: Concrete Example

Manually creating a recurrent network for detecting repeated 1s

```
w_xh = np.array([[1], [0], [0]])

w_hh = np.array([[0, 0, 0],
                 [1, 0, 0],
                 [0, 0, 1]])

w_yh = np.array([1, 1, -1])

x_seq = [0, 1, 0, 1, 1, 1, 0, 1, 1]

h_t_prev = np.array([[0], [0], [1]])

for t, x in enumerate(x_seq):
    h_t = relu(w_hh @ h_t_prev + (w_xh @ x))
    y_t = relu(w_yh @ h_t)
    h_t_prev = h_t
```

* Code is missing parts, for full code see here

$$h_t = ReLU(W_{hh}h_{t-1} + W_{xh}x_t)$$

$$y_t = ReLU(W_{hy}h_t)$$

$$h_t = \begin{pmatrix} \text{Current} \\ \text{Previous} \\ 1 \end{pmatrix}$$

Copy over "current" value from previous hidden state to be "previous"

X	RNN →	Y
0		0
1		0
0		0
1		0
1		**1**
1		**1**
0		0
1		0
1		**1**

Vanilla(-ish) RNN: Concrete Example

Manually creating a recurrent network for detecting repeated 1s

```
w_xh = np.array([[1], [0], [0]])

w_hh = np.array([[0, 0, 0],
                 [1, 0, 0],
                 [0, 0, 1]])

w_yh = np.array([1, 1, -1])

x_seq = [0, 1, 0, 1, 1, 1, 0, 1, 1]

h_t_prev = np.array([[0], [0], [1]])

for t, x in enumerate(x_seq):
    h_t = relu(w_hh @ h_t_prev + (w_xh @ x))
    y_t = relu(w_yh @ h_t)
    h_t_prev = h_t
```

$$h_t = ReLU(W_{hh}h_{t-1} + W_{xh}x_t)$$

$$y_t = ReLU(W_{hy}h_t)$$

$$h_t = \begin{pmatrix} \text{Current} \\ \text{Previous} \\ 1 \end{pmatrix}$$

Keep 1 on the bottom (helpful for output)

X	RNN →	Y
0		0
1		0
0		0
1		0
1		**1**
1		**1**
0		0
1		0
1		**1**

* Code is missing parts, for full code see here

Vanilla(-ish) RNN: Concrete Example

Manually creating a recurrent network for detecting repeated 1s

```
w_xh = np.array([[1], [0], [0]])

w_hh = np.array([[0, 0, 0],
                 [1, 0, 0],
                 [0, 0, 1]])

w_yh = np.array([1, 1, -1])

x_seq = [0, 1, 0, 1, 1, 1, 0, 1, 1]

h_t_prev = np.array([[0], [0], [1]])

for t, x in enumerate(x_seq):
    h_t = relu(w_hh @ h_t_prev + (w_xh @ x))
    y_t = relu(w_yh @ h_t)
    h_t_prev = h_t
```

Max(Current + Previous – 1, 0)

$$h_t = ReLU(W_{hh}h_{t-1} + W_{xh}x_t)$$

$$y_t = ReLU(W_{hy}h_t)$$

$$h_t = \begin{pmatrix} \text{Current} \\ \text{Previous} \\ 1 \end{pmatrix}$$

X → RNN → Y

X	Y
0	0
1	0
0	0
1	0
1	**1**
1	**1**
0	0
1	0
1	**1**

* Code is missing parts, for full code see here

Vanilla(-ish) RNN: Concrete Example

Manually creating a recurrent network for detecting repeated 1s

```
w_xh = np.array([[1], [0], [0]])

w_hh = np.array([[0, 0, 0],
                 [1, 0, 0],
                 [0, 0, 1]])

w_yh = np.array([1, 1, -1])

x_seq = [0, 1, 0, 1, 1, 1, 0, 1, 1]

h_t_prev = np.array([[0], [0], [1]])

for t, x in enumerate(x_seq):
    h_t = relu(w_hh @ h_t_prev + (w_xh @ x))
    y_t = relu(w_yh @ h_t)
    h_t_prev = h_t
```

* Code is missing parts, for full code see here

$$h_t = ReLU(W_{hh}h_{t-1} + W_{xh}x_t)$$

$$y_t = ReLU(W_{hy}h_t)$$

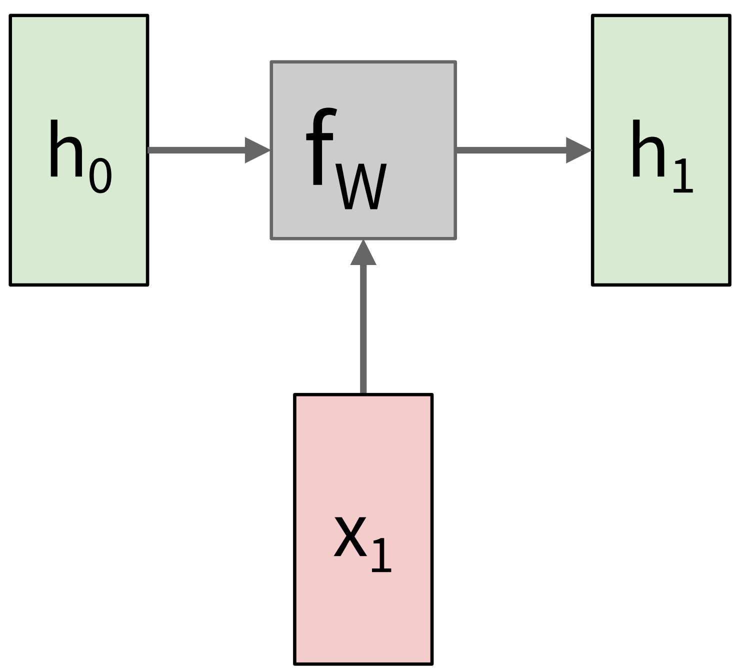
$$h_t = \begin{pmatrix} \text{Current} \\ \text{Previous} \\ 1 \end{pmatrix}$$

And it just works! But how do we find the Ws in practice?

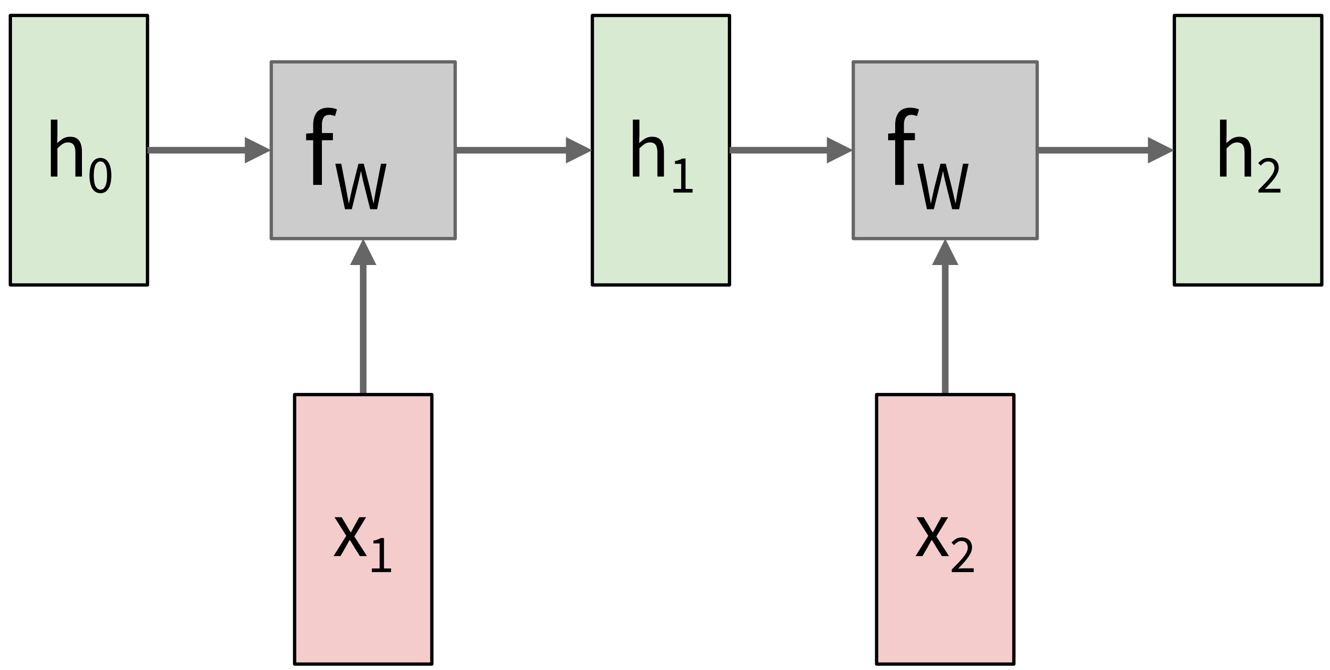
RNN →

X	Y
0	0
1	0
0	0
1	0
1	**1**
1	**1**
0	0
1	0
1	**1**

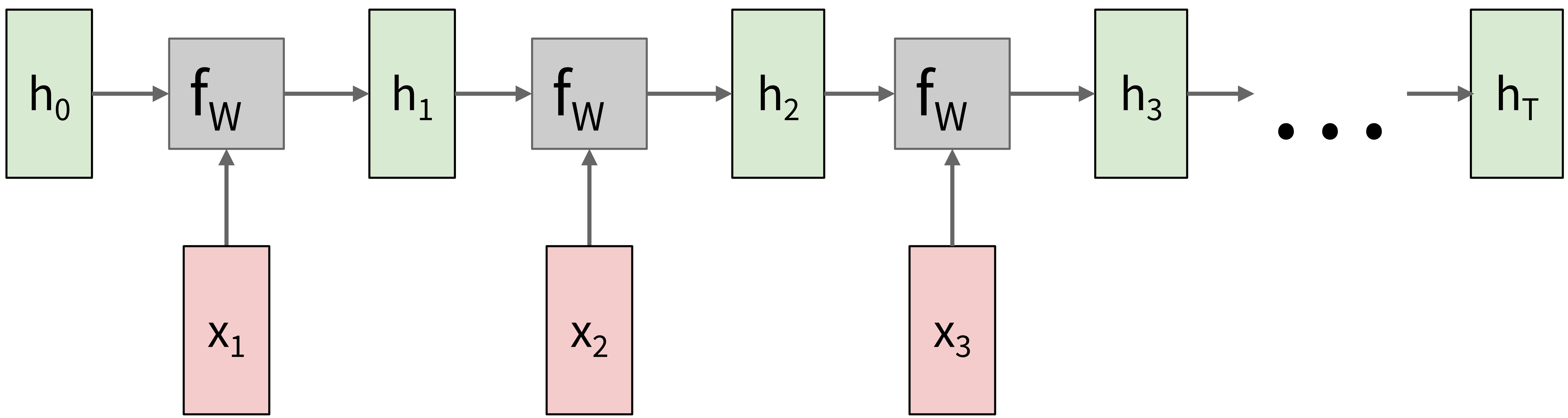
RNN: Computational Graph



RNN: Computational Graph

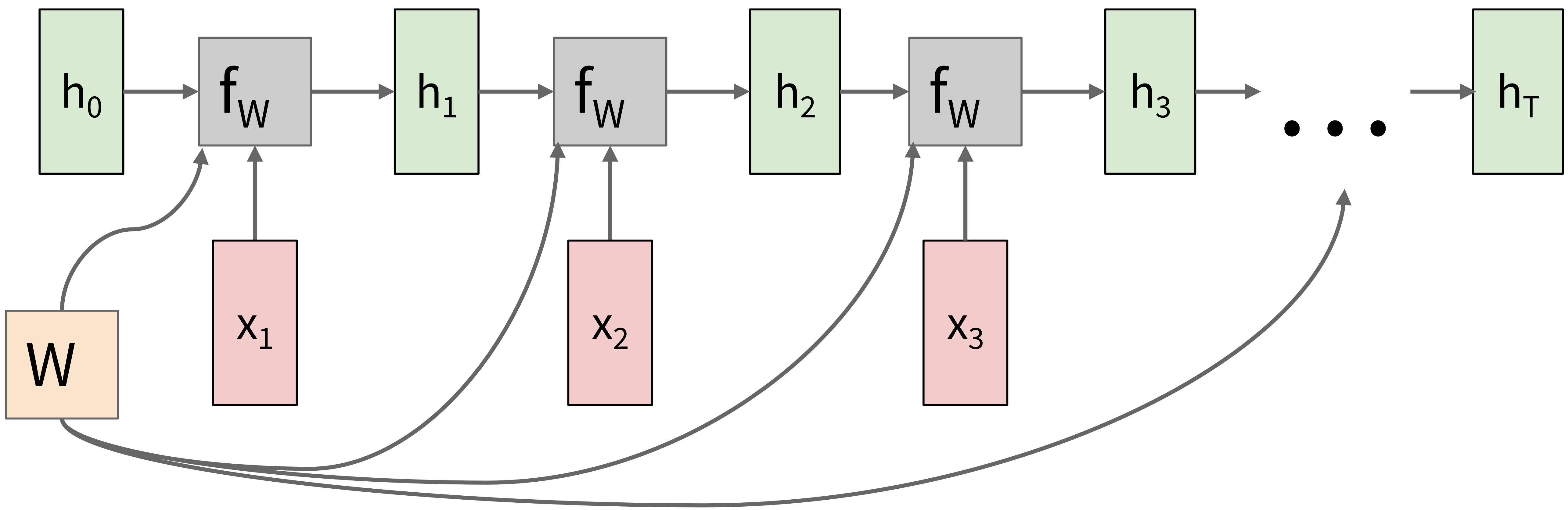


RNN: Computational Graph

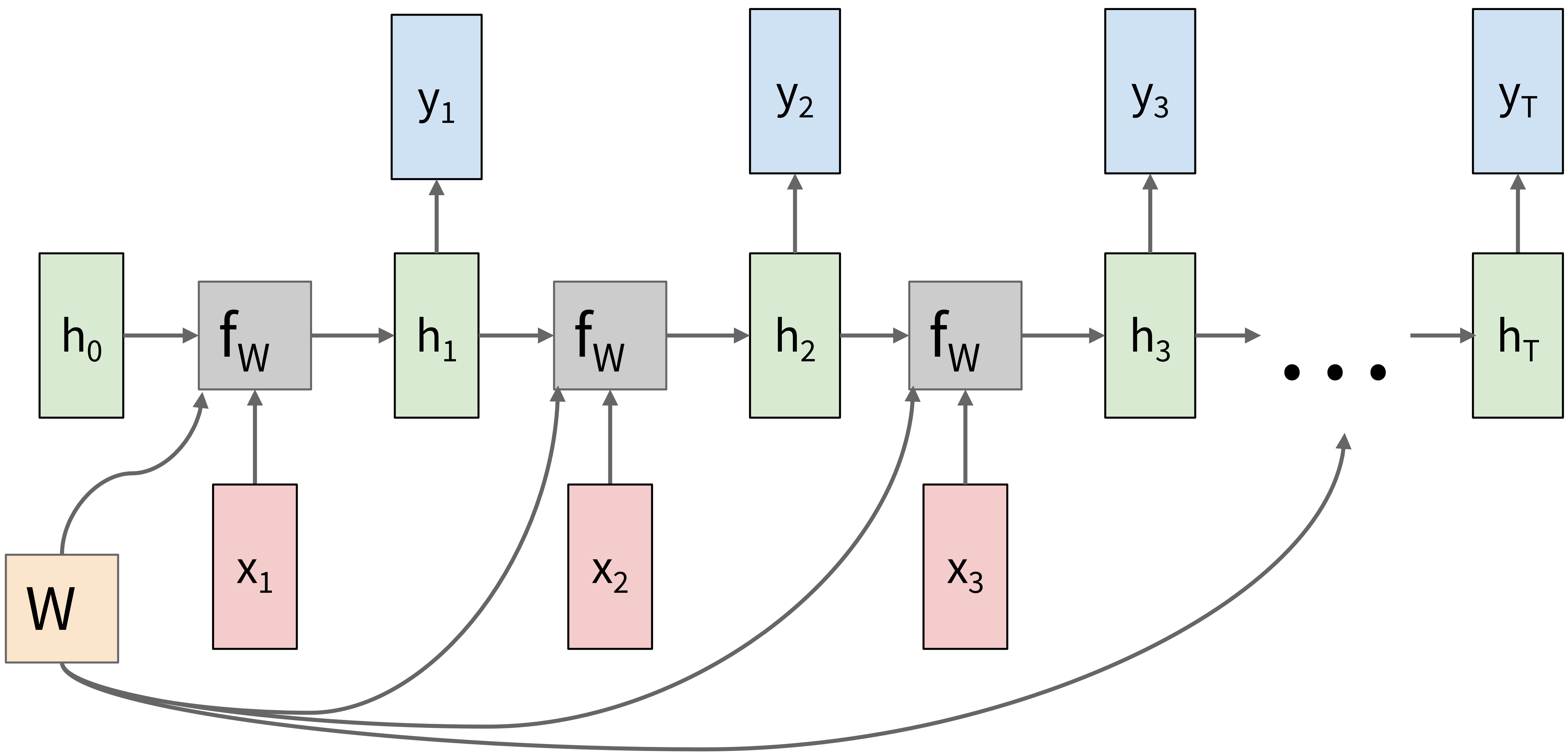


RNN: Computational Graph

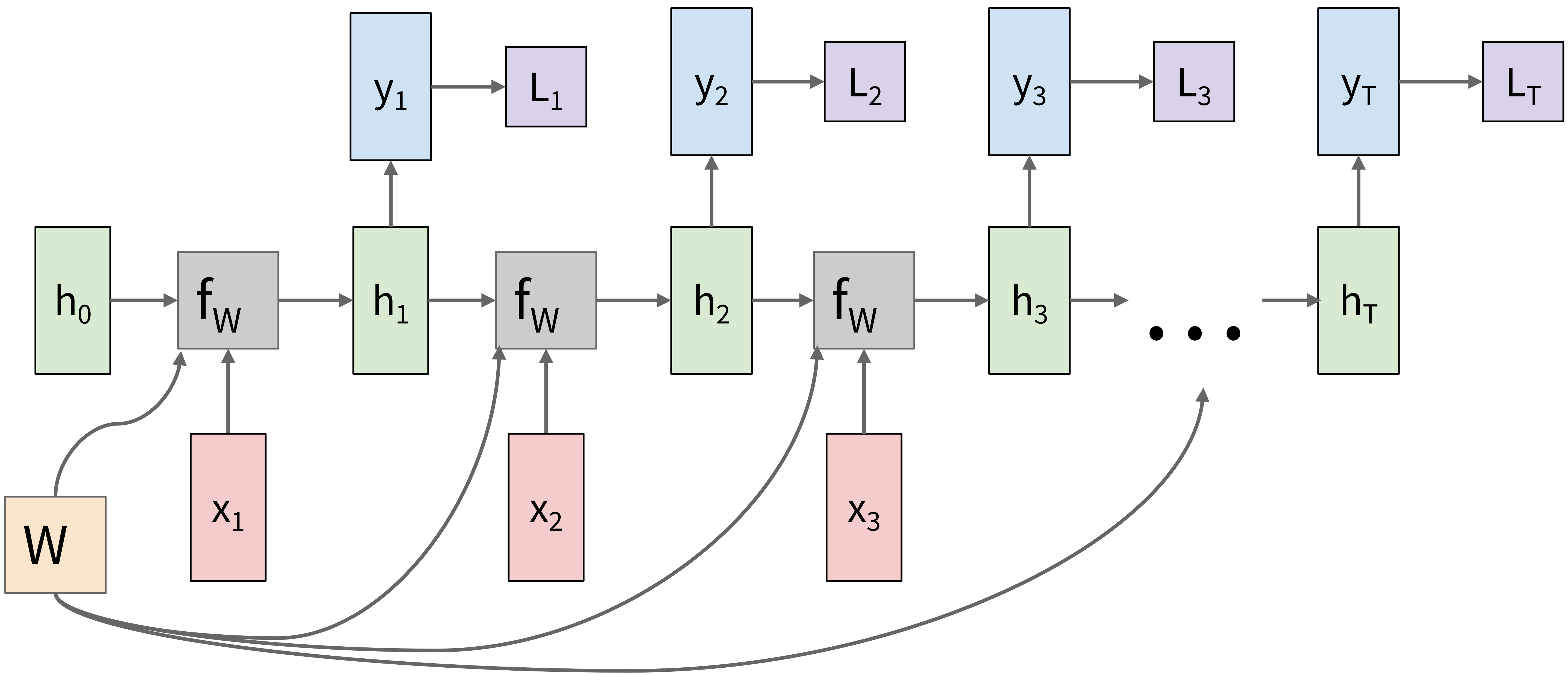
Re-use the same weight matrix at every time-step



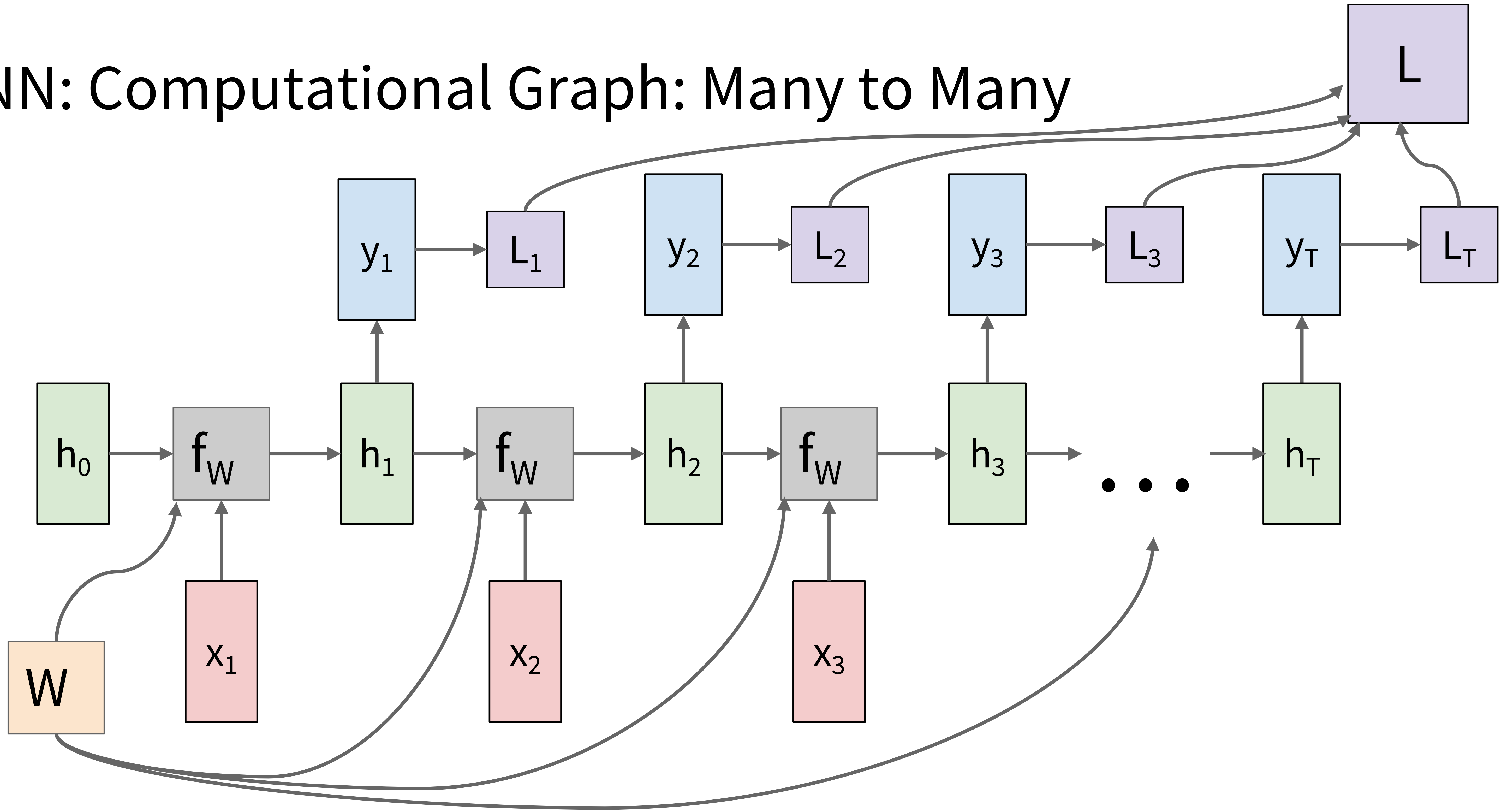
RNN: Computational Graph: Many to Many



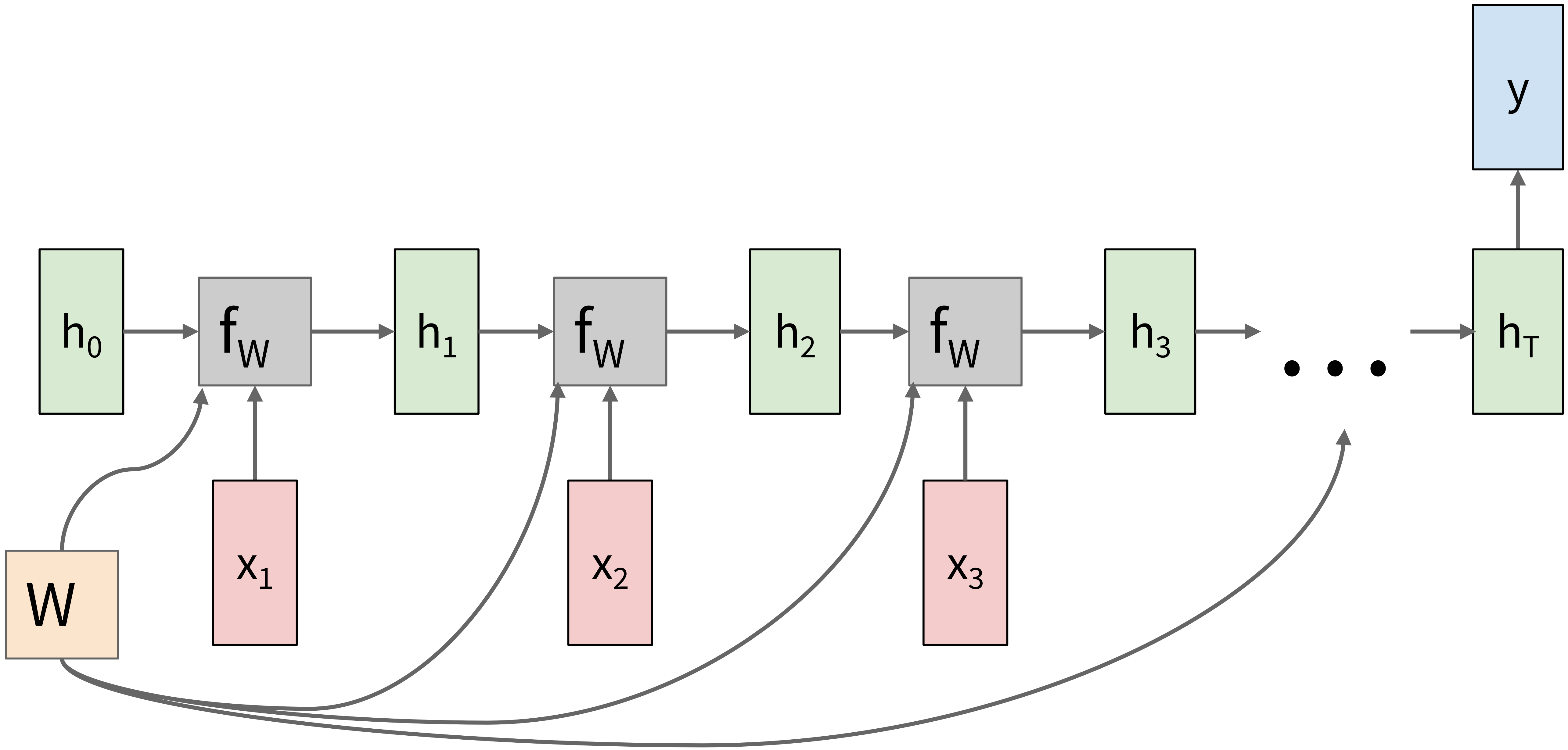
RNN: Computational Graph: Many to Many



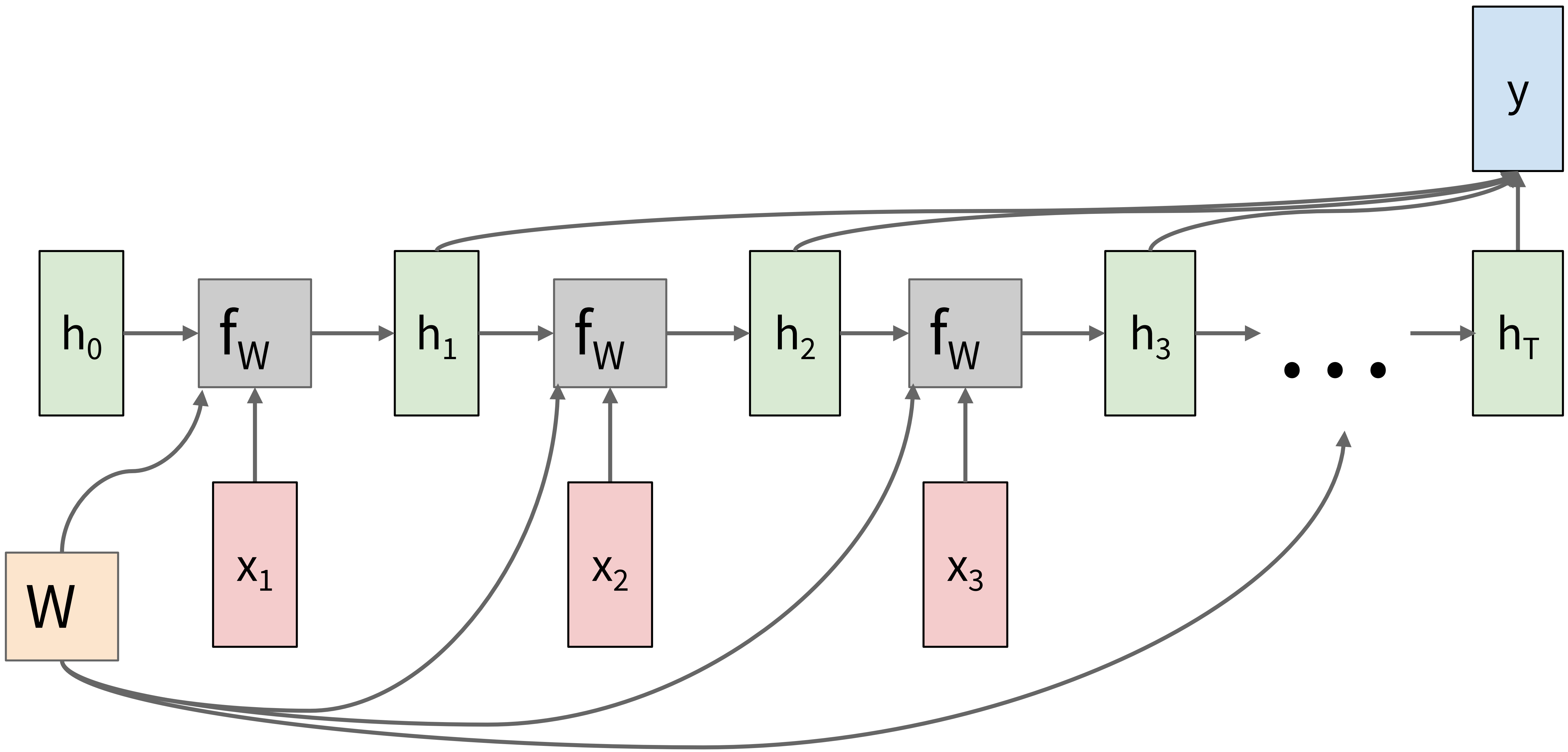
RNN: Computational Graph: Many to Many



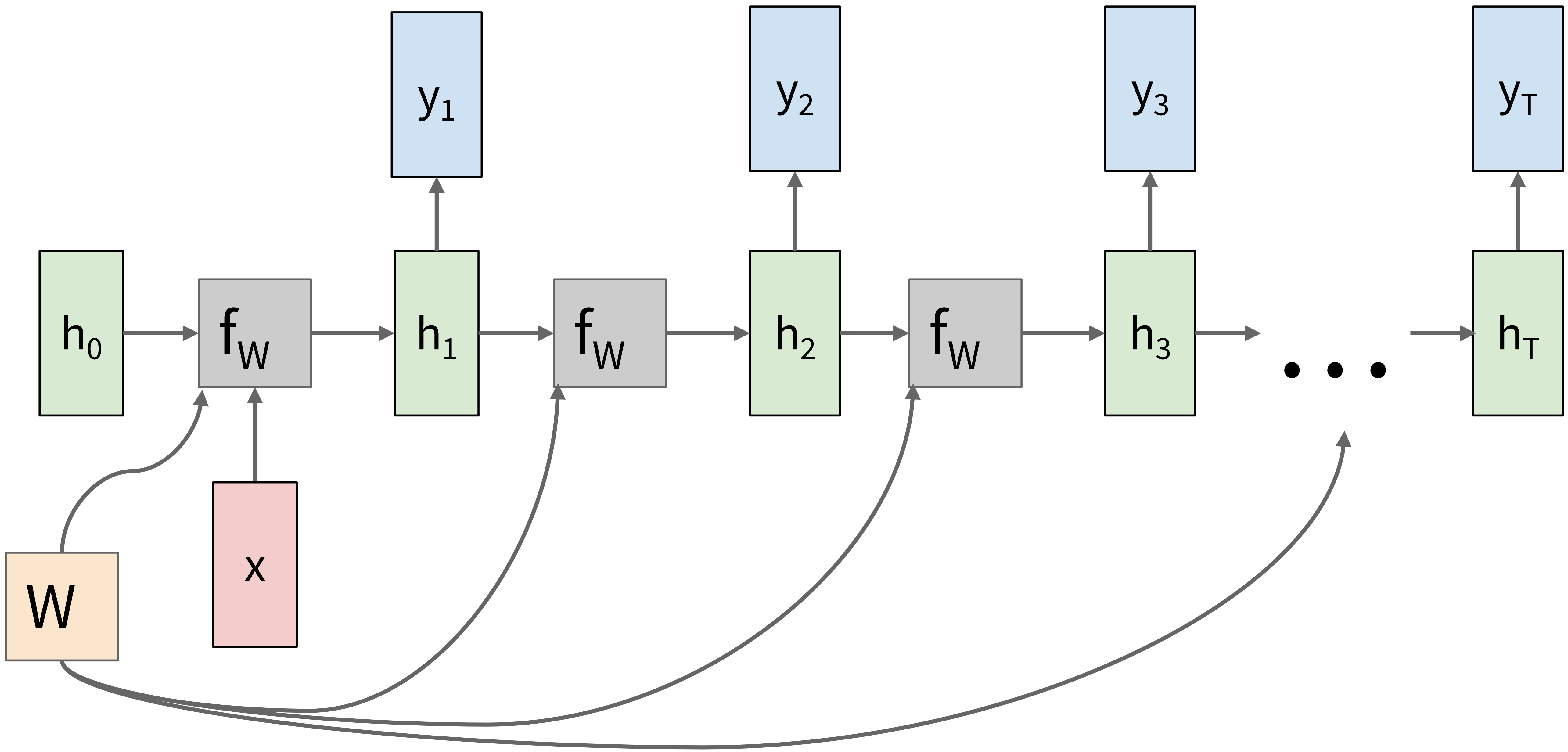
RNN: Computational Graph: Many to One



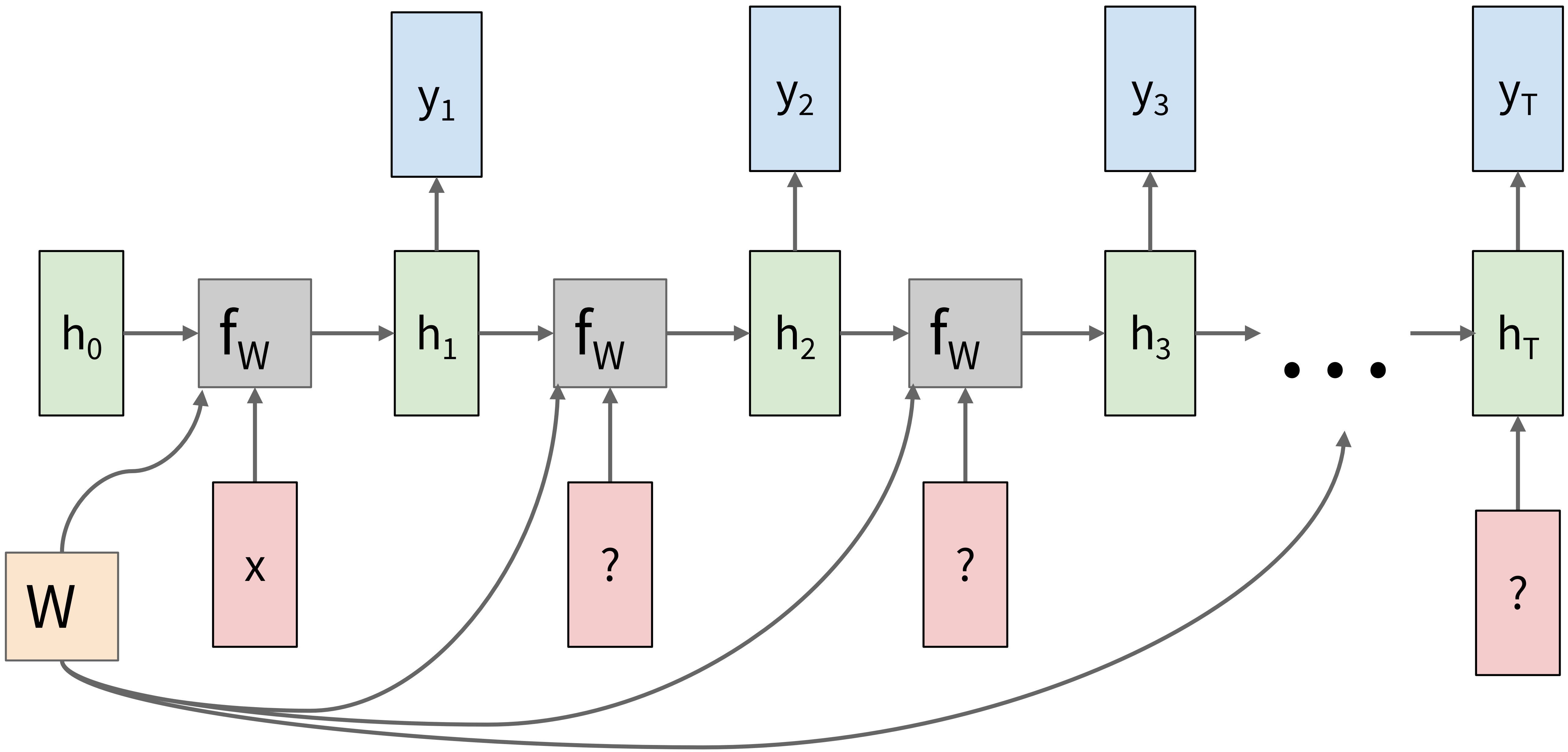
RNN: Computational Graph: Many to One



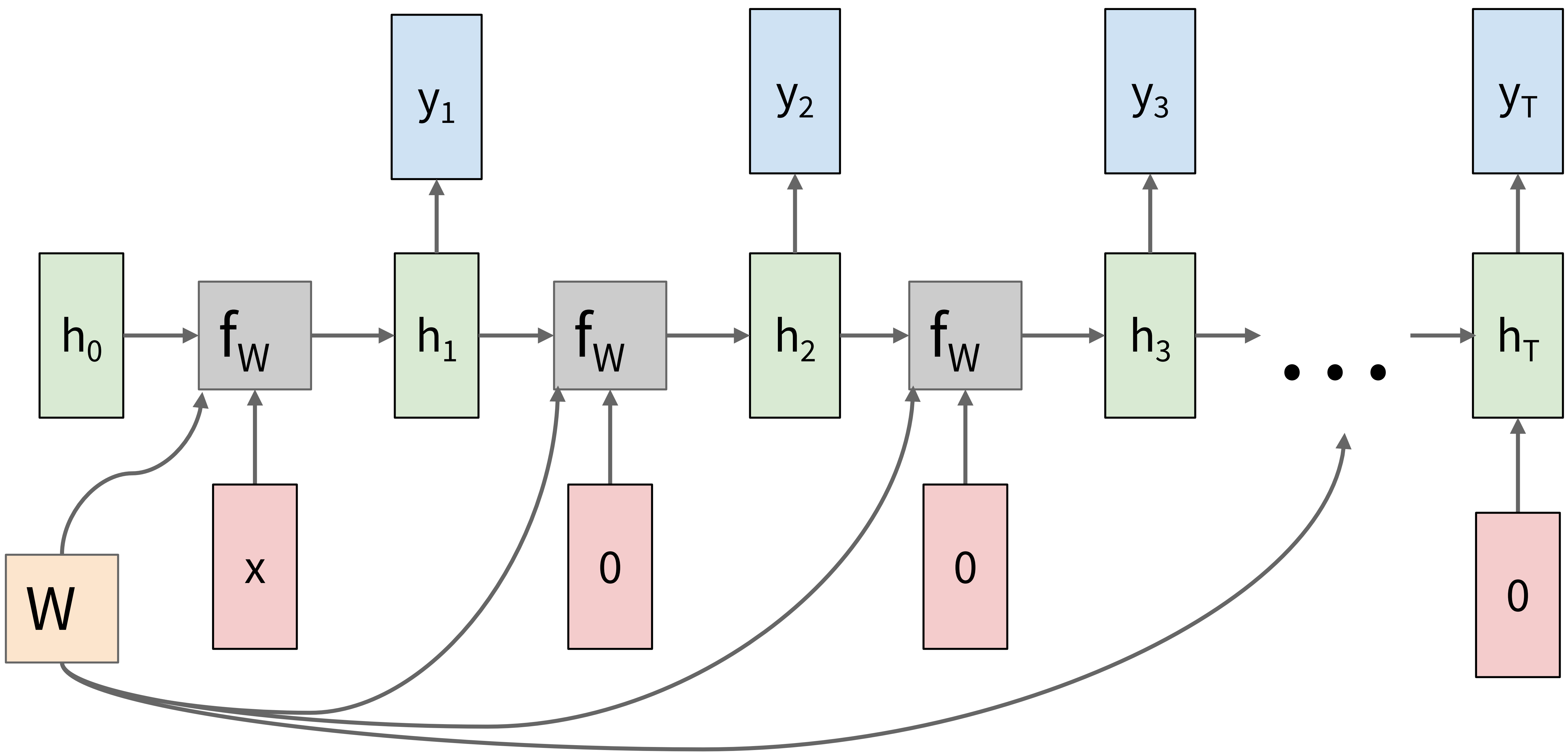
RNN: Computational Graph: One to Many



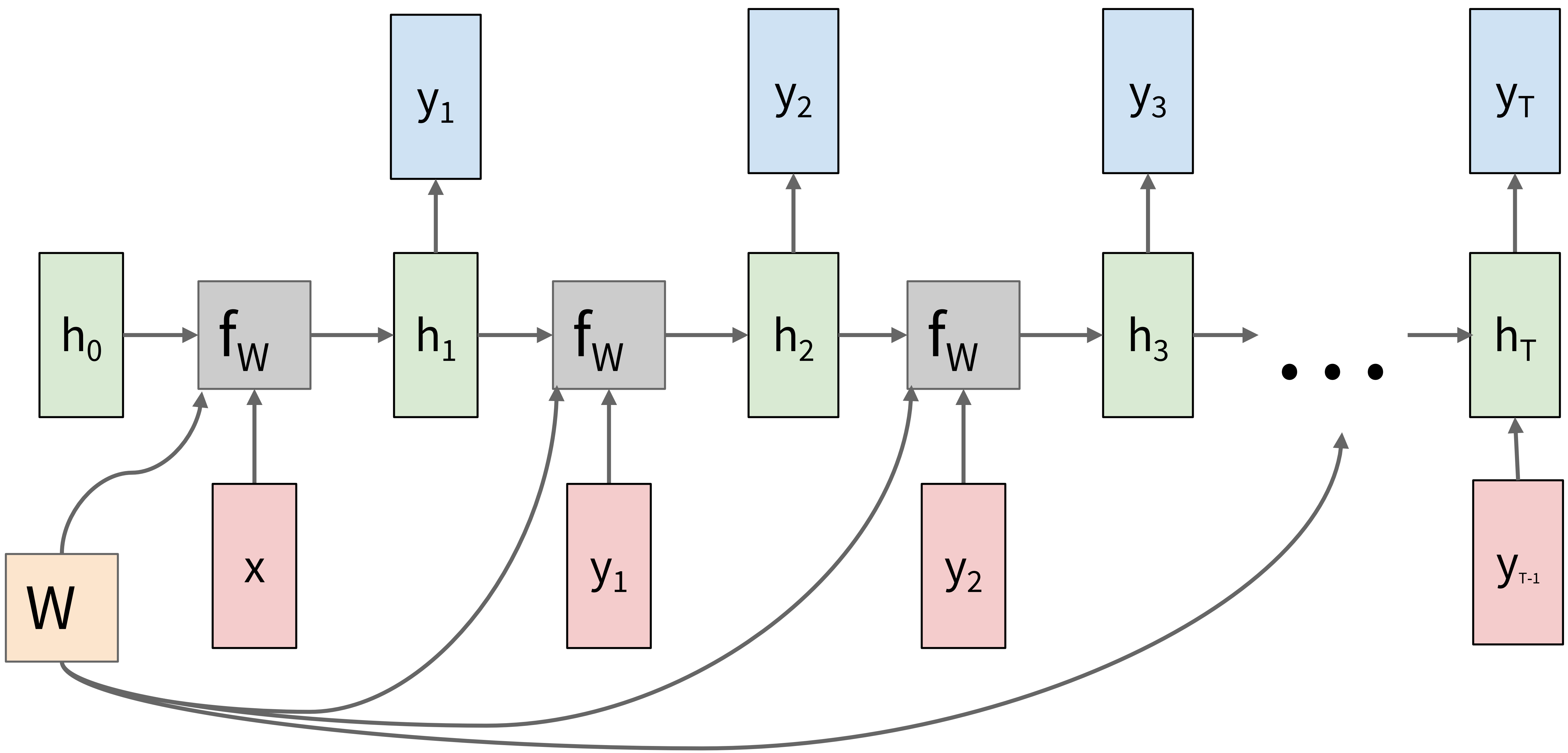
RNN: Computational Graph: One to Many



RNN: Computational Graph: One to Many

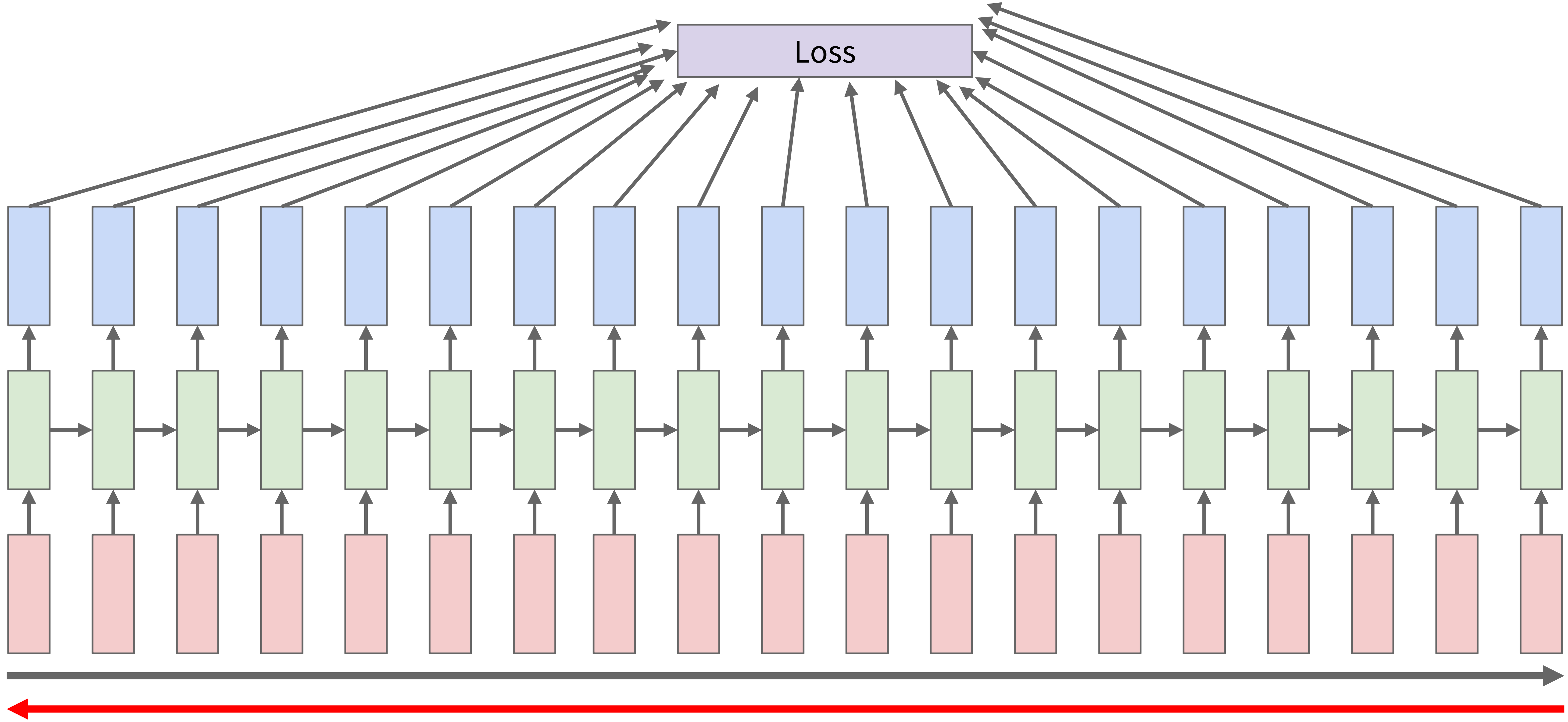


RNN: Computational Graph: One to Many

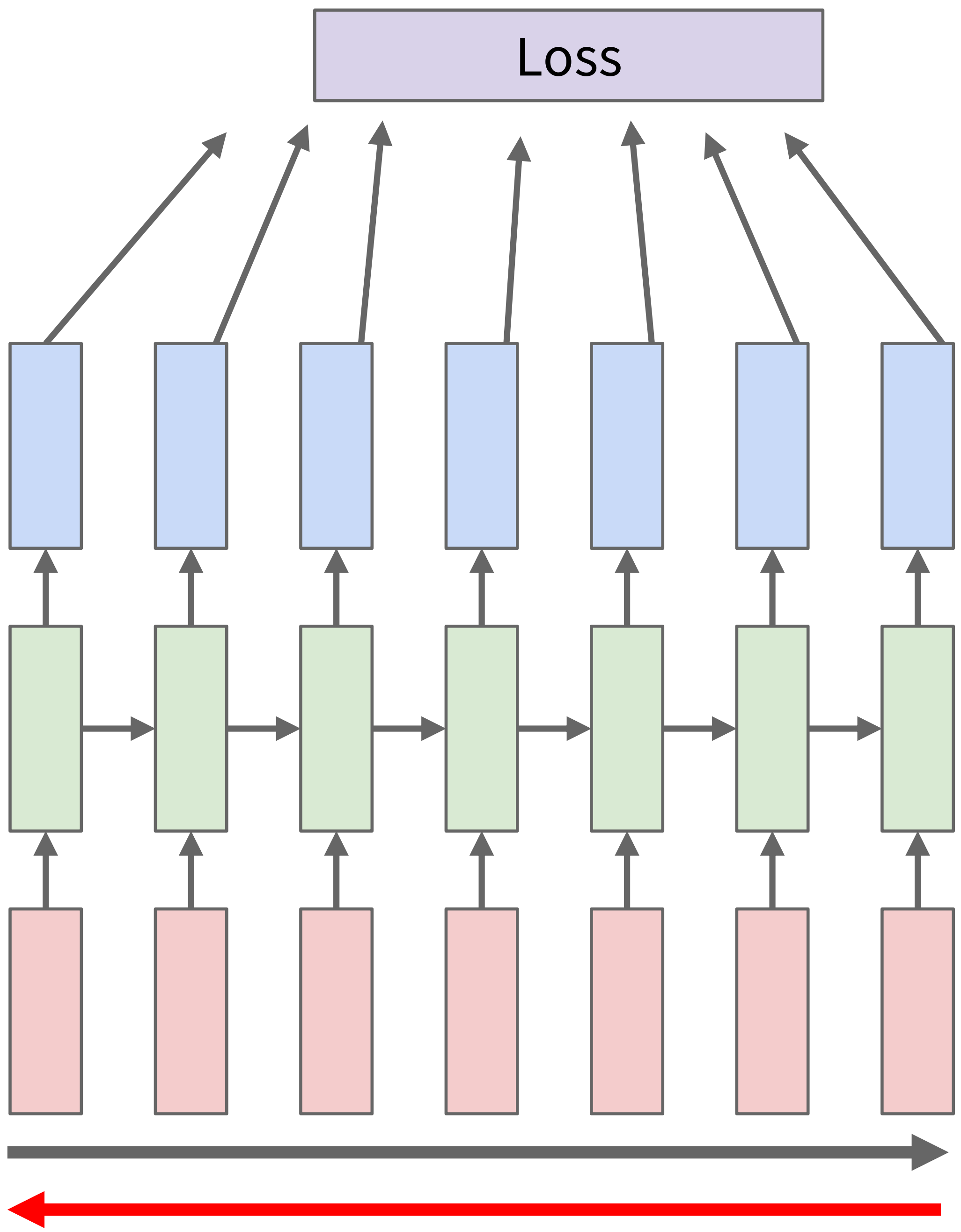


Backpropagation through time

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

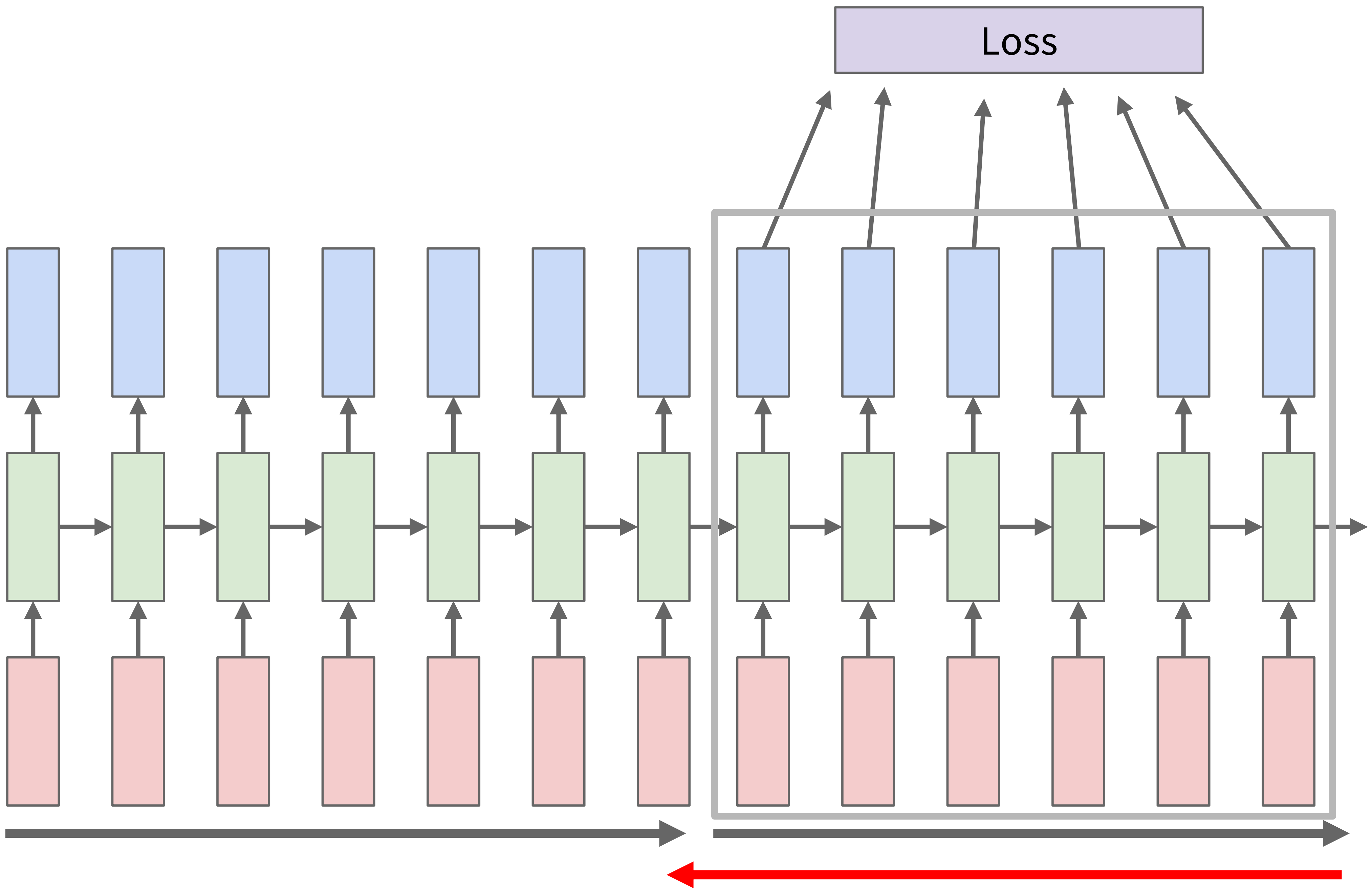


Truncated Backpropagation through time



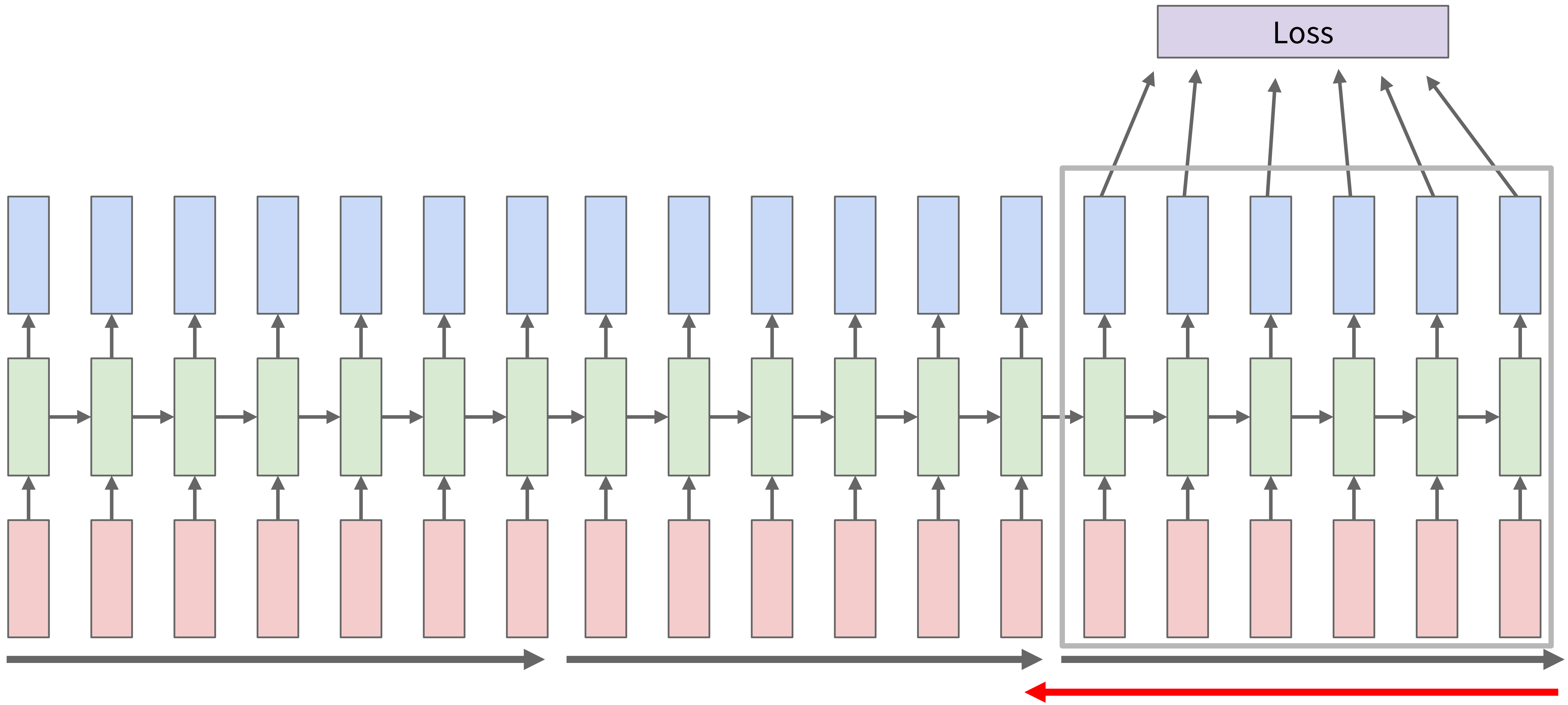
Run forward and backward through chunks of the sequence instead of whole sequence

Truncated Backpropagation through time



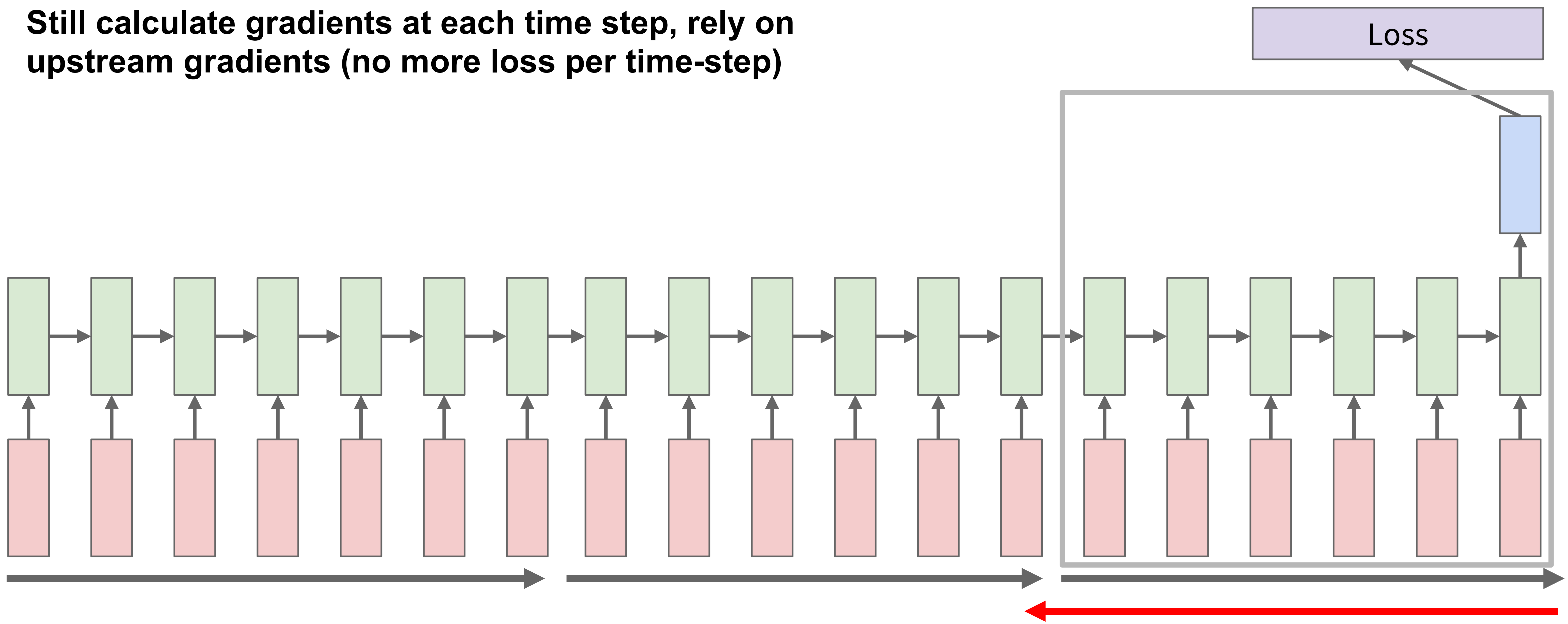
Carry hidden states forward in time forever, but only backpropagate for some smaller number of steps

Truncated Backpropagation through time



Truncated BPTT: Single Output

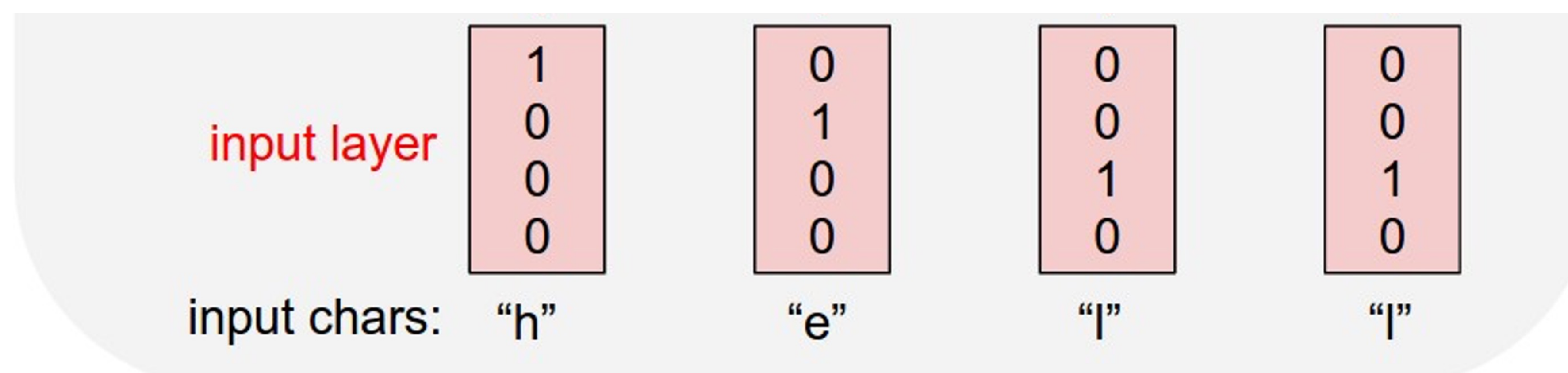
Still calculate gradients at each time step, rely on upstream gradients (no more loss per time-step)



A more practical example: Character-level **Language Model**

Vocabulary: [h,e,l,o]

Example training sequence: “hello”

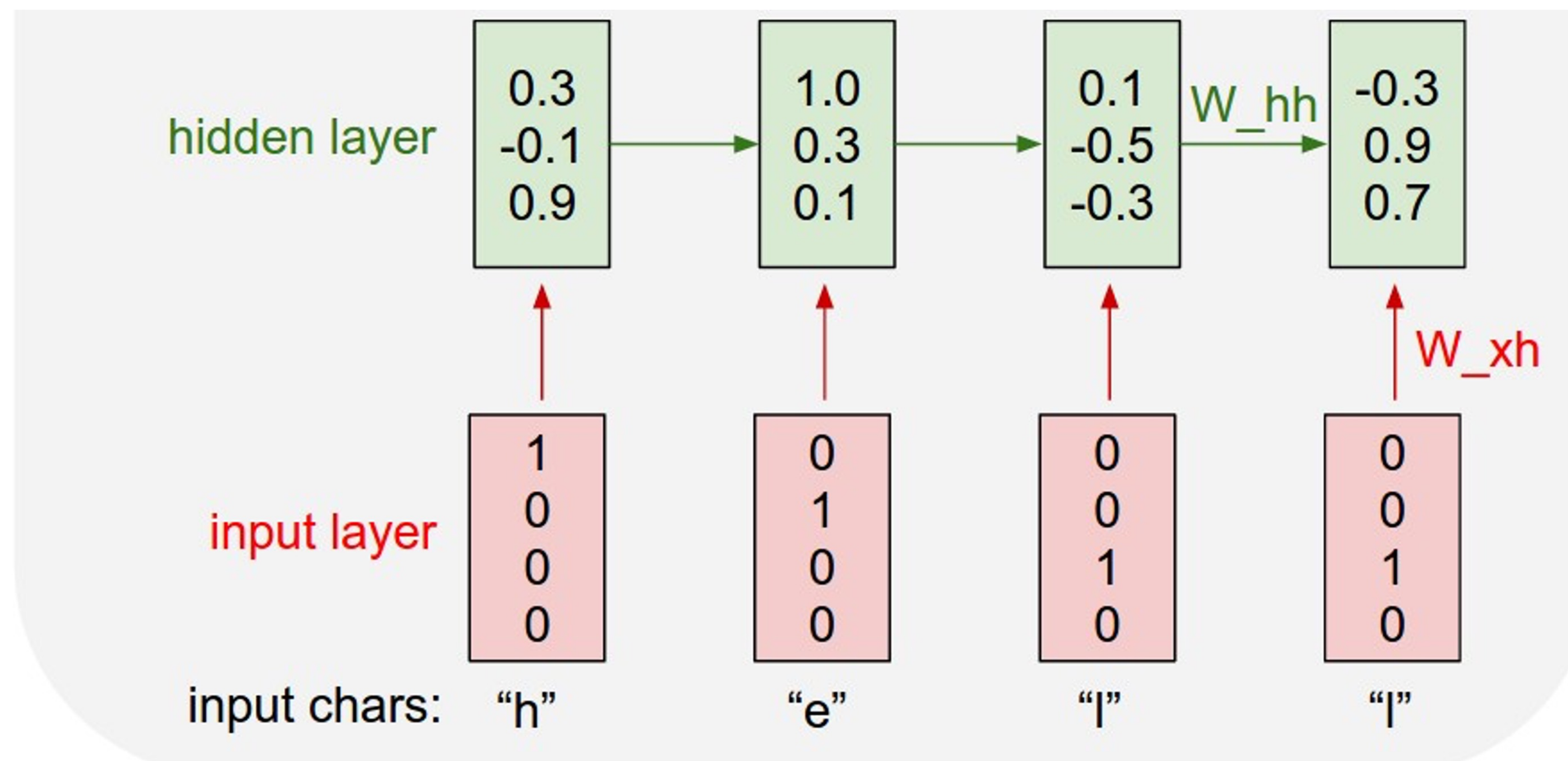


A more practical example: Character-level **Language Model**

Vocabulary: [h,e,l,o]

Example training sequence: “hello”

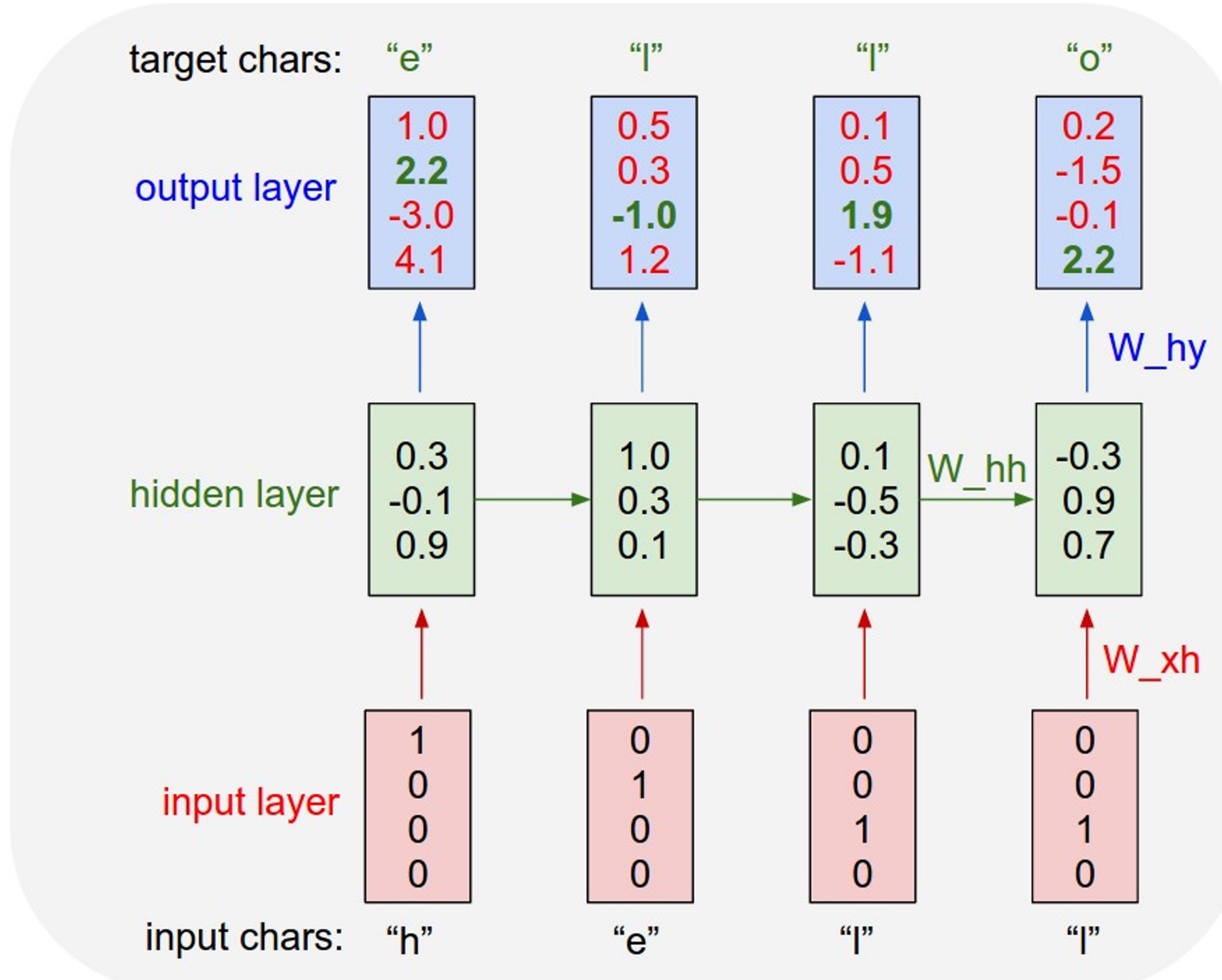
$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$



A more practical example: Character-level **Language Model**

Vocabulary: [h,e,l,o]

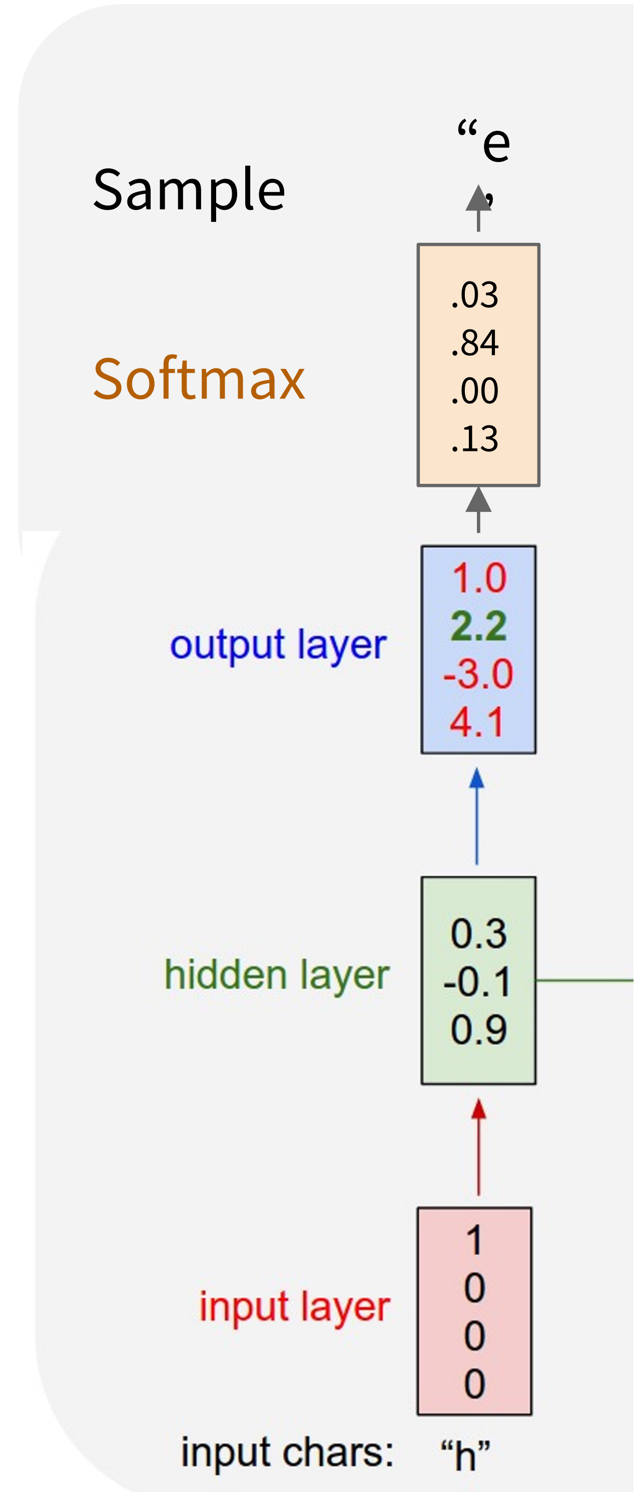
Example training sequence: “hello”



Example: Character-level Language Model
Sampling

Vocabulary:
[h,e,l,o]

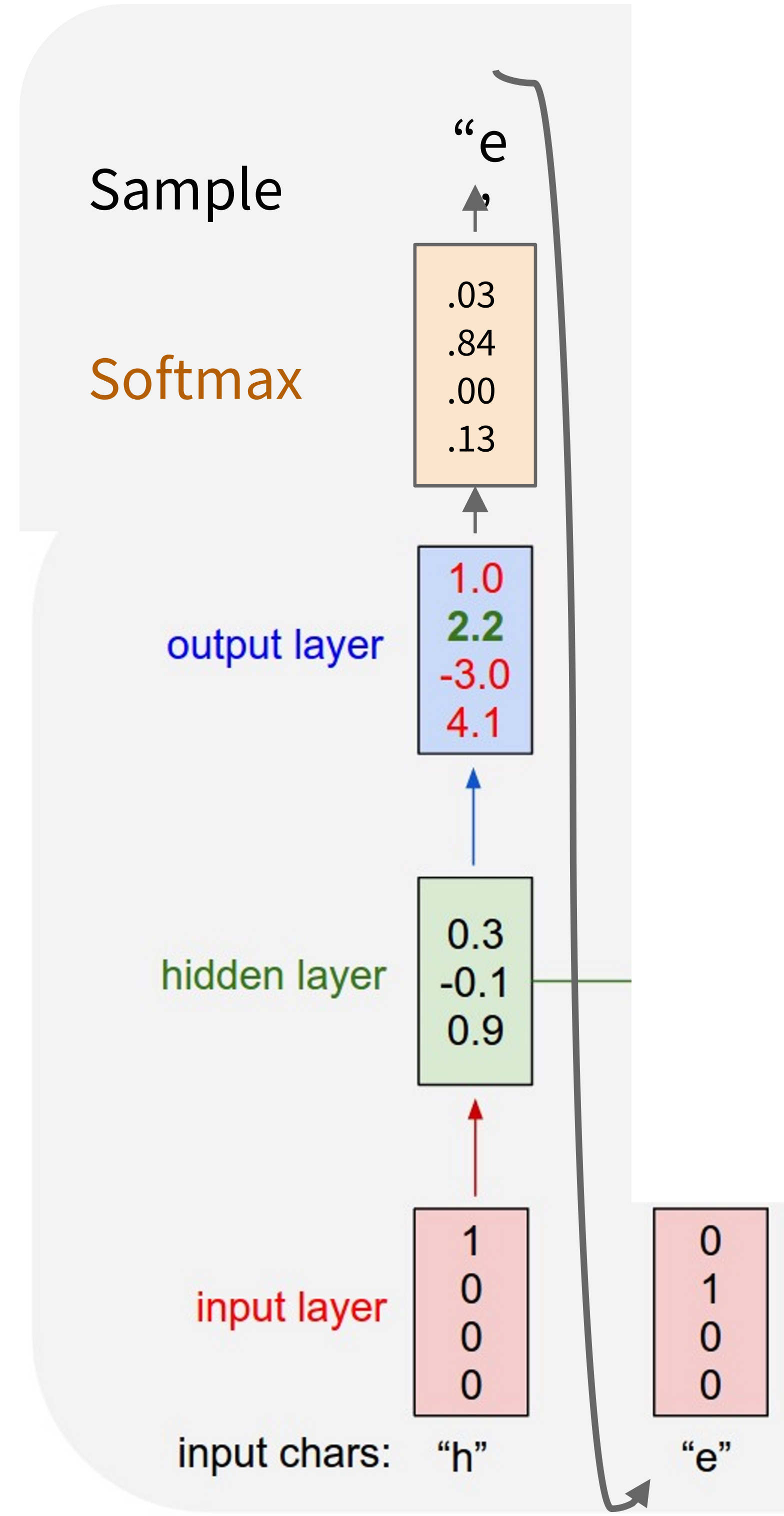
At test-time sample characters one at a time, feed back to model



Example: Character-level Language Model Sampling

Vocabulary:
[h,e,l,o]

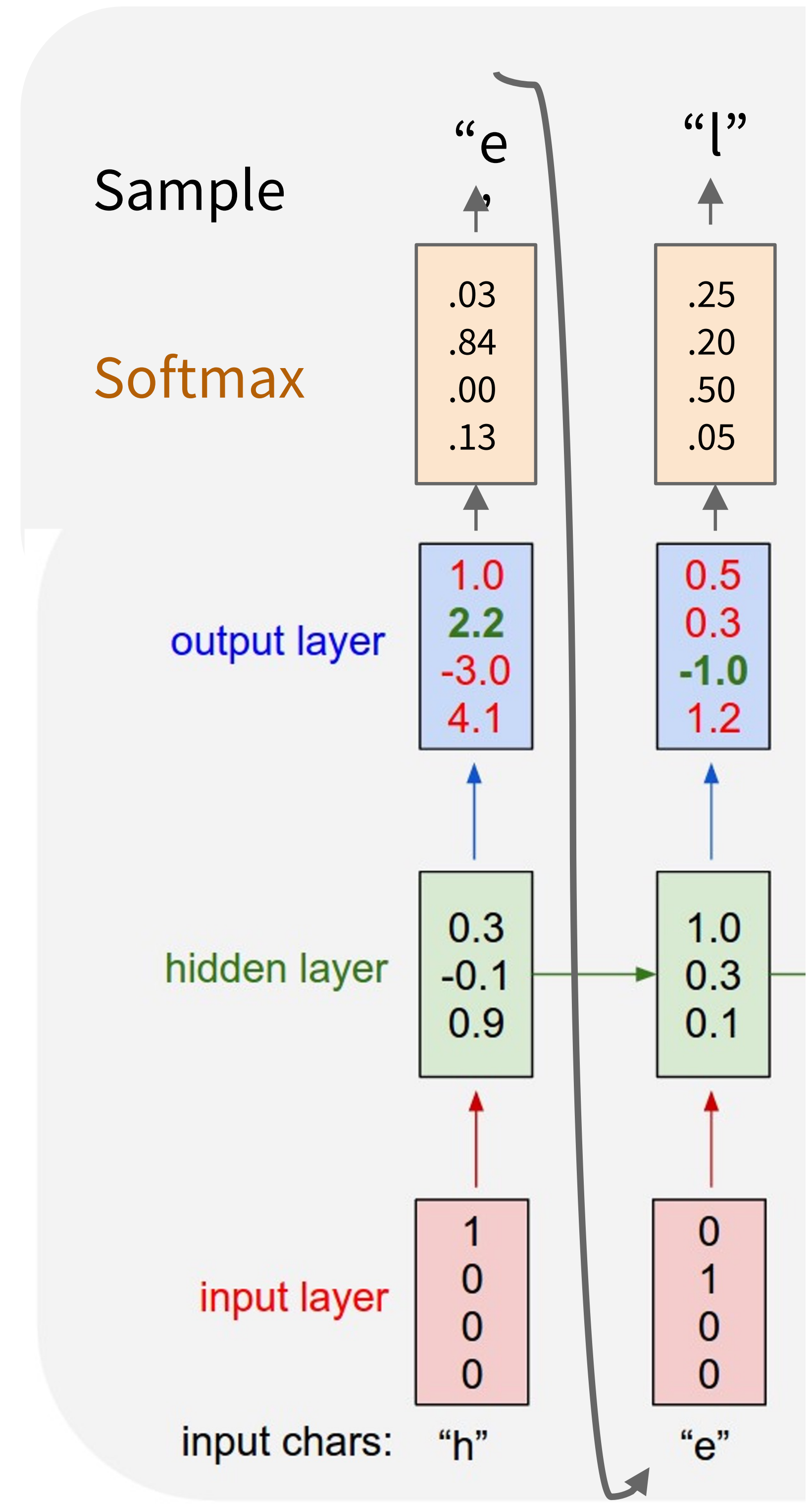
At test-time sample characters one at a time, feed back to model



Example: Character-level Language Model Sampling

Vocabulary:
[h,e,l,o]

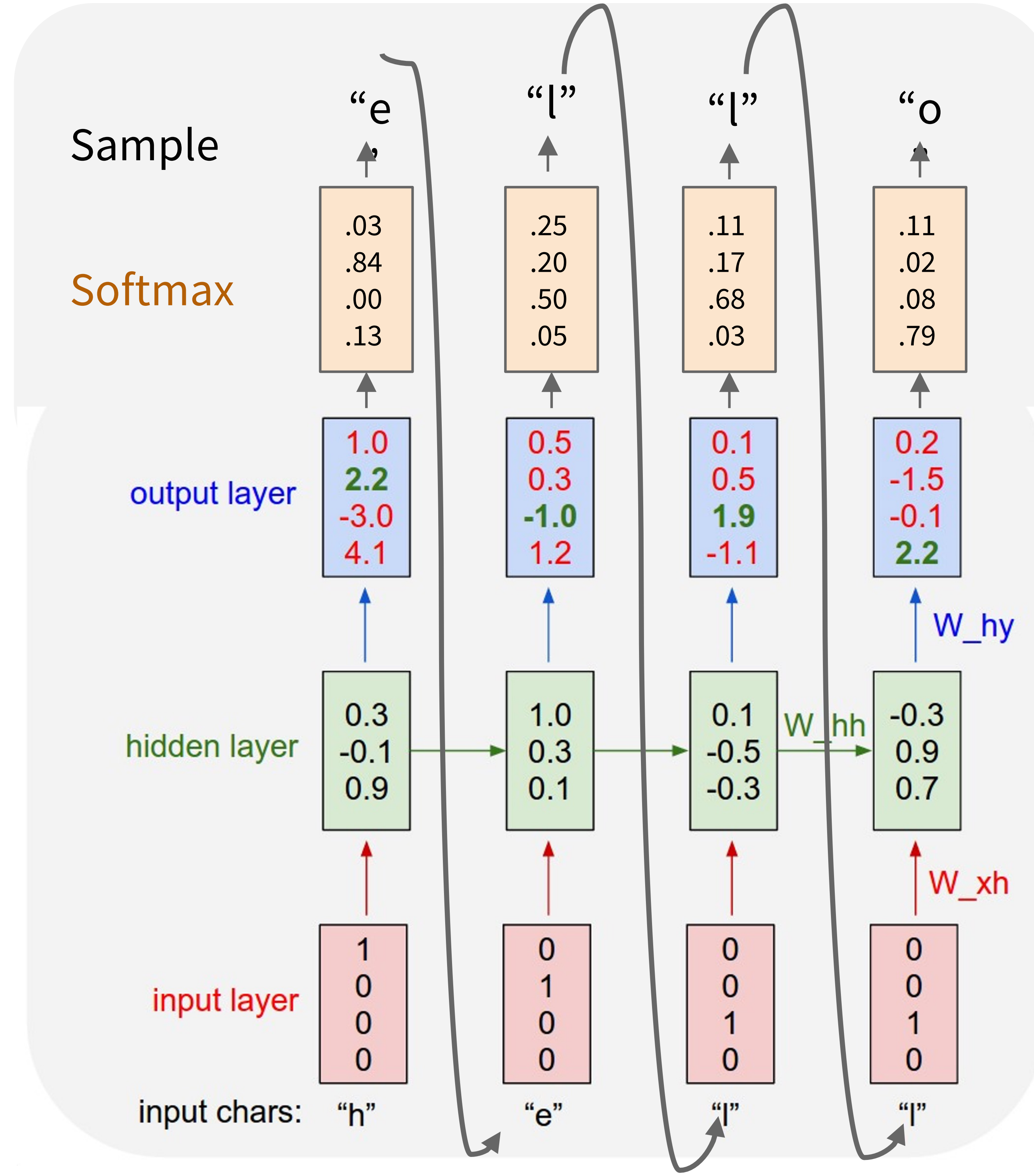
At test-time sample characters one at a time, feed back to model



Example: Character-level Language Model Sampling

Vocabulary:
[h,e,l,o]

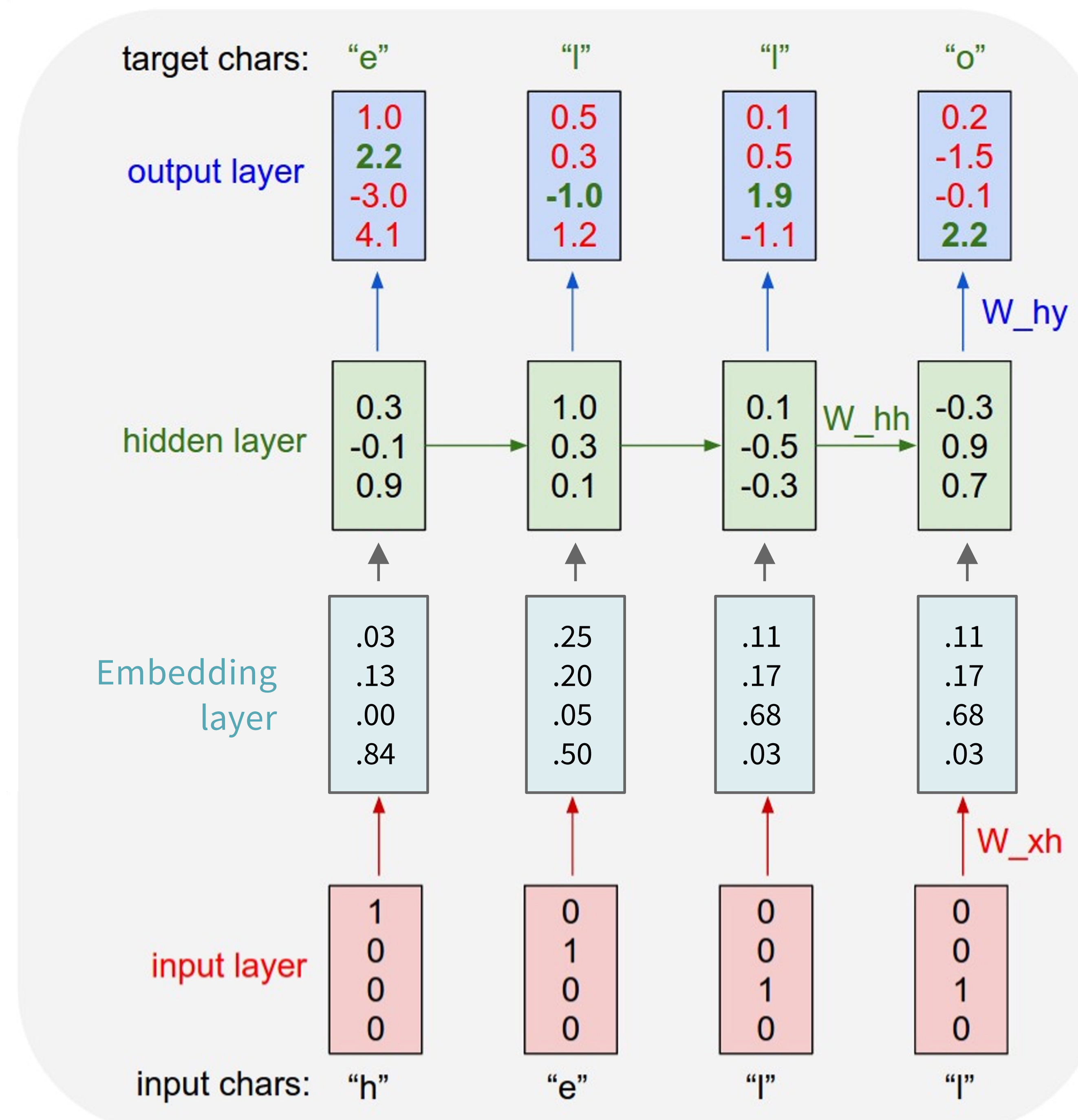
At test-time sample characters one at a time, feed back to model



Example: Character-level Language Model Sampling

$$\begin{bmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ w_{21} & w_{22} & w_{23} & w_{14} \\ w_{31} & w_{32} & w_{33} & w_{14} \\ w_{41} & w_{42} & w_{43} & w_{44} \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} w_{11} \\ w_{21} \\ w_{31} \\ w_{41} \end{bmatrix}$$

Matrix multiplication with a one-hot vector just extracts a column from the weight matrix. We often put a separate embedding layer between the input and hidden layers.



min-char-rnn.py gist: 112 lines of Python

```python
"""
Minimal character-level Vanilla RNN model. Written by Andrej Karpathy (@karpathy)
BSD License
"""
import numpy as np

# data I/O
data = open('input.txt', 'r').read() # should be simple plain text file
chars = list(set(data))
data_size, vocab_size = len(data), len(chars)
print 'data has %d characters, %d unique.' % (data_size, vocab_size)
char_to_ix = { ch:i for i,ch in enumerate(chars) }
ix_to_char = { i:ch for i,ch in enumerate(chars) }

# hyperparameters
hidden_size = 100 # size of hidden layer of neurons
seq_length = 25 # number of steps to unroll the RNN for
learning_rate = 1e-1

# model parameters
Wxh = np.random.randn(hidden_size, vocab_size)*0.01 # input to hidden
Whh = np.random.randn(hidden_size, hidden_size)*0.01 # hidden to hidden
Why = np.random.randn(vocab_size, hidden_size)*0.01 # hidden to output
bh = np.zeros((hidden_size, 1)) # hidden bias
by = np.zeros((vocab_size, 1)) # output bias

def lossFun(inputs, targets, hprev):
  """
  inputs,targets are both list of integers.
  hprev is Hx1 array of initial hidden state
  returns the loss, gradients on model parameters, and last hidden state
  """
  xs, hs, ys, ps = {}, {}, {}, {}
  hs[-1] = np.copy(hprev)
  loss = 0
  # forward pass
  for t in xrange(len(inputs)):
    xs[t] = np.zeros((vocab_size,1)) # encode in 1-of-k representation
    xs[t][inputs[t]] = 1
    hs[t] = np.tanh(np.dot(Wxh, xs[t]) + np.dot(Whh, hs[t-1]) + bh) # hidden state
    ys[t] = np.dot(Why, hs[t]) + by # unnormalized log probabilities for next chars
    ps[t] = np.exp(ys[t]) / np.sum(np.exp(ys[t])) # probabilities for next chars
    loss += -np.log(ps[t][targets[t],0]) # softmax (cross-entropy loss)
  # backward pass: compute gradients going backwards
  dWxh, dWhh, dWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
  dbh, dby = np.zeros_like(bh), np.zeros_like(by)
  dhnext = np.zeros_like(hs[0])
  for t in reversed(xrange(len(inputs))):
    dy = np.copy(ps[t])
    dy[targets[t]] -= 1 # backprop into y
    dWhy += np.dot(dy, hs[t].T)
    dby += dy
    dh = np.dot(Why.T, dy) + dhnext # backprop into h
    dhraw = (1 - hs[t] * hs[t]) * dh # backprop through tanh nonlinearity
    dbh += dhraw
    dWxh += np.dot(dhraw, xs[t].T)
    dWhh += np.dot(dhraw, hs[t-1].T)
    dhnext = np.dot(Whh.T, dhraw)
  for dparam in [dWxh, dWhh, dWhy, dbh, dby]:
    np.clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients
  return loss, dWxh, dWhh, dWhy, dbh, dby, hs[len(inputs)-1]
```

```python
def sample(h, seed_ix, n):
  """
  sample a sequence of integers from the model
  h is memory state, seed_ix is seed letter for first time step
  """
  x = np.zeros((vocab_size, 1))
  x[seed_ix] = 1
  ixes = []
  for t in xrange(n):
    h = np.tanh(np.dot(Wxh, x) + np.dot(Whh, h) + bh)
    y = np.dot(Why, h) + by
    p = np.exp(y) / np.sum(np.exp(y))
    ix = np.random.choice(range(vocab_size), p=p.ravel())
    x = np.zeros((vocab_size, 1))
    x[ix] = 1
    ixes.append(ix)
  return ixes

n, p = 0, 0
mWxh, mWhh, mWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
mbh, mby = np.zeros_like(bh), np.zeros_like(by) # memory variables for Adagrad
smooth_loss = -np.log(1.0/vocab_size)*seq_length # loss at iteration 0
while True:
  # prepare inputs (we're sweeping from left to right in steps seq_length long)
  if p+seq_length+1 >= len(data) or n == 0:
    hprev = np.zeros((hidden_size,1)) # reset RNN memory
    p = 0 # go from start of data
  inputs = [char_to_ix[ch] for ch in data[p:p+seq_length]]
  targets = [char_to_ix[ch] for ch in data[p+1:p+seq_length+1]]

  # sample from the model now and then
  if n % 100 == 0:
    sample_ix = sample(hprev, inputs[0], 200)
    txt = ''.join(ix_to_char[ix] for ix in sample_ix)
    print '----\n %s \n----' % (txt, )

  # forward seq_length characters through the net and fetch gradient
  loss, dWxh, dWhh, dWhy, dbh, dby, hprev = lossFun(inputs, targets, hprev)
  smooth_loss = smooth_loss * 0.999 + loss * 0.001
  if n % 100 == 0: print 'iter %d, loss: %f' % (n, smooth_loss) # print progress

  # perform parameter update with Adagrad
  for param, dparam, mem in zip([Wxh, Whh, Why, bh, by],
                                [dWxh, dWhh, dWhy, dbh, dby],
                                [mWxh, mWhh, mWhy, mbh, mby]):
    mem += dparam * dparam
    param += -learning_rate * dparam / np.sqrt(mem + 1e-8) # adagrad update

  p += seq_length # move data pointer
  n += 1 # iteration counter
```

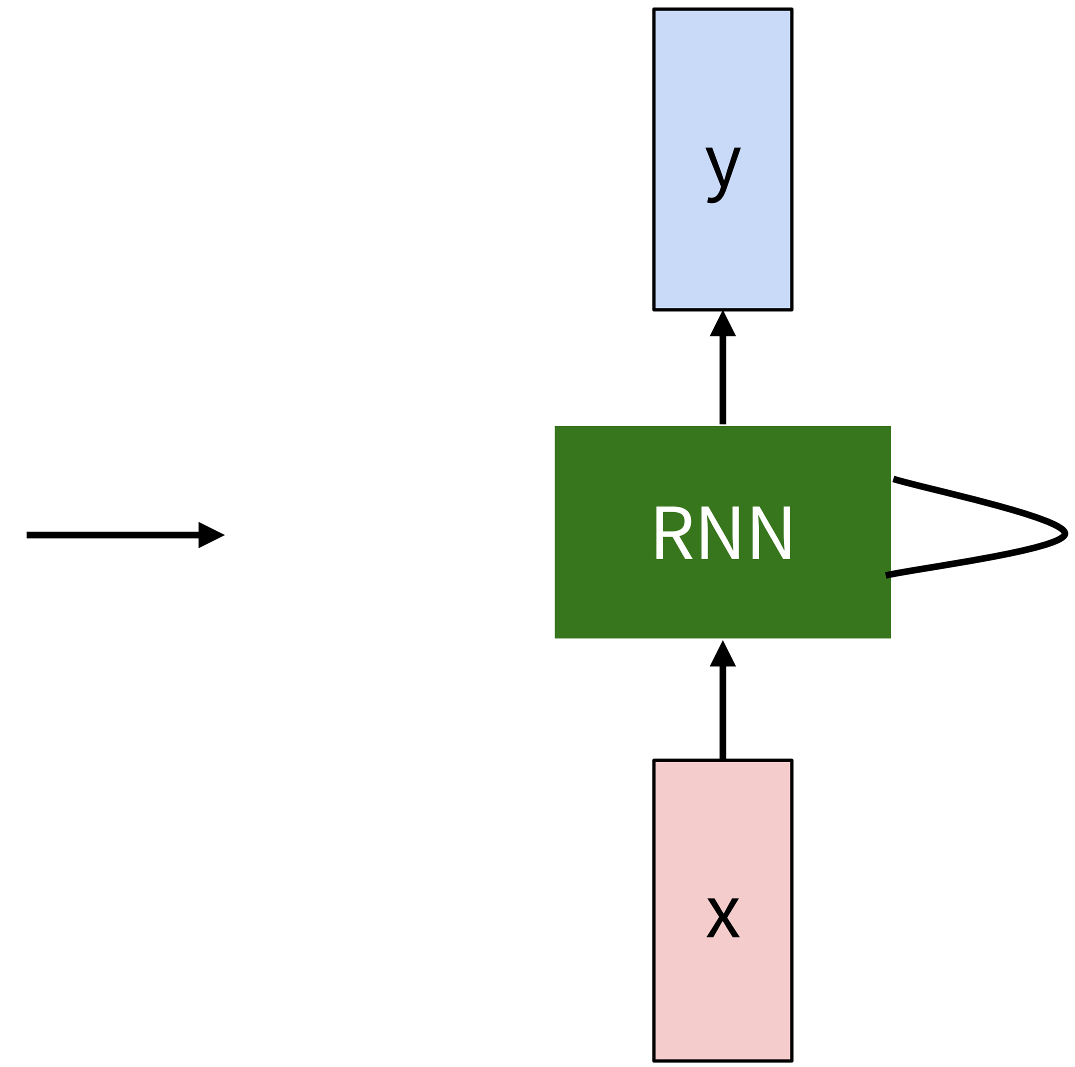
(https://gist.github.com/karpathy/d4dee566867f8291f086)

THE SONNETS

by William Shakespeare

From fairest creatures we desire increase,
That thereby beauty's rose might never die,
But as the riper should by time decease,
His tender heir might bear his memory:
But thou, contracted to thine own bright eyes,
Feed'st thy light's flame with self-substantial fuel,
Making a famine where abundance lies,
Thyself thy foe, to thy sweet self too cruel:
Thou that art now the world's fresh ornament,
And only herald to the gaudy spring,
Within thine own bud buriest thy content,
And tender churl mak'st waste in niggarding:
Pity the world, or else this glutton be,
To eat the world's due, by the grave and thee.

When forty winters shall besiege thy brow,
And dig deep trenches in thy beauty's field,
Thy youth's proud livery so gazed on now,
Will be a tatter'd weed of small worth held:
Then being asked, where all thy beauty lies,
Where all the treasure of thy lusty days;
To say, within thine own deep sunken eyes,
Were an all-eating shame, and thriftless praise.
How much more praise deserv'd thy beauty's use,
If thou couldst answer 'This fair child of mine
Shall sum my count, and make my old excuse,'
Proving his beauty by succession thine!
This were to be new made when thou art old,
And see thy blood warm when thou feel'st it cold.



Blogpost from Andrej Karpathy back in 2015!

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

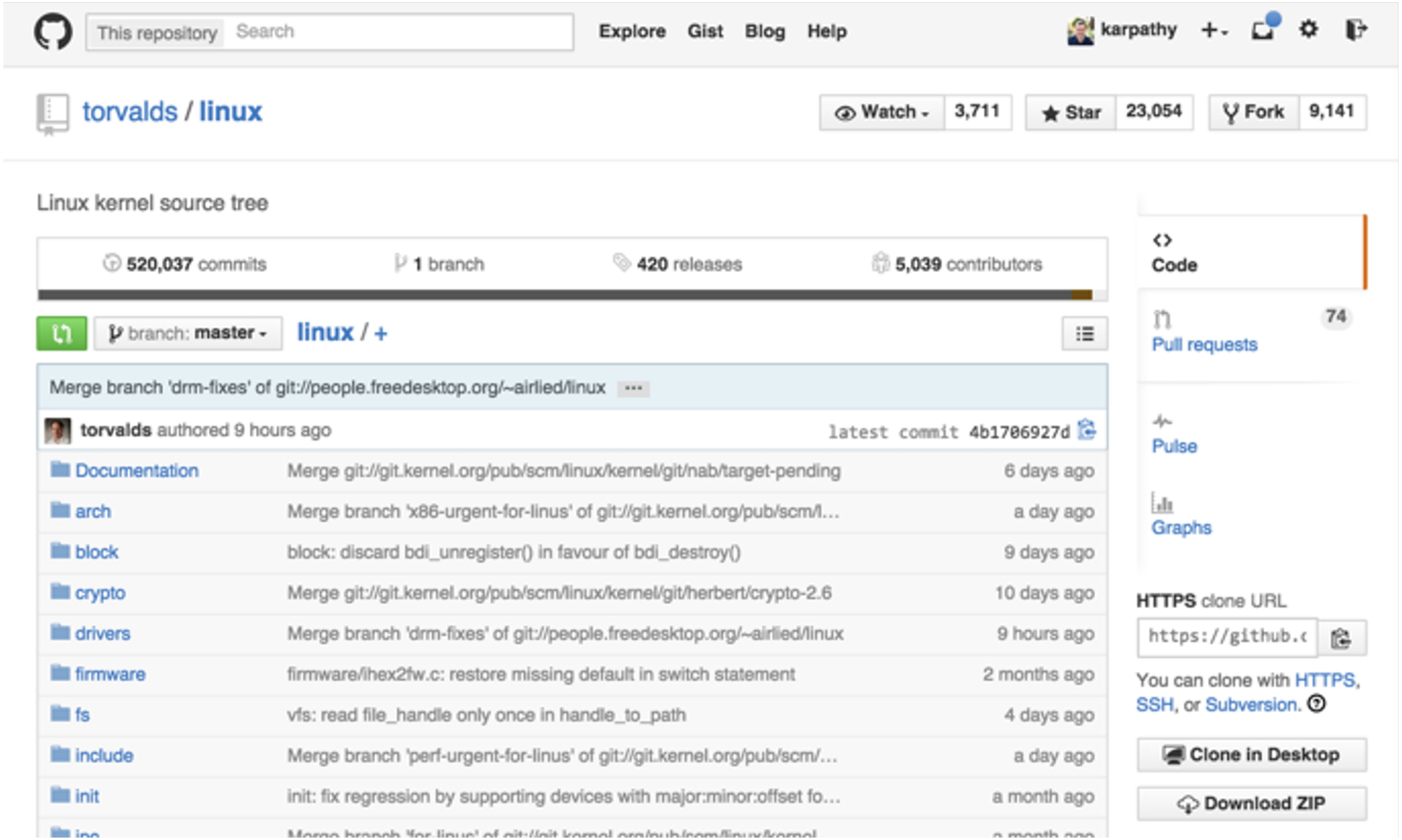
```
Aftair fall unsuch that the hall for Prince Velzonski's that me of
her hearly, and behs to so arwage fiving were to it beloge, pavu say falling misfort
how, and Gogition is so overelical and ofter.
```

↓ 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.
```

```
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.
```

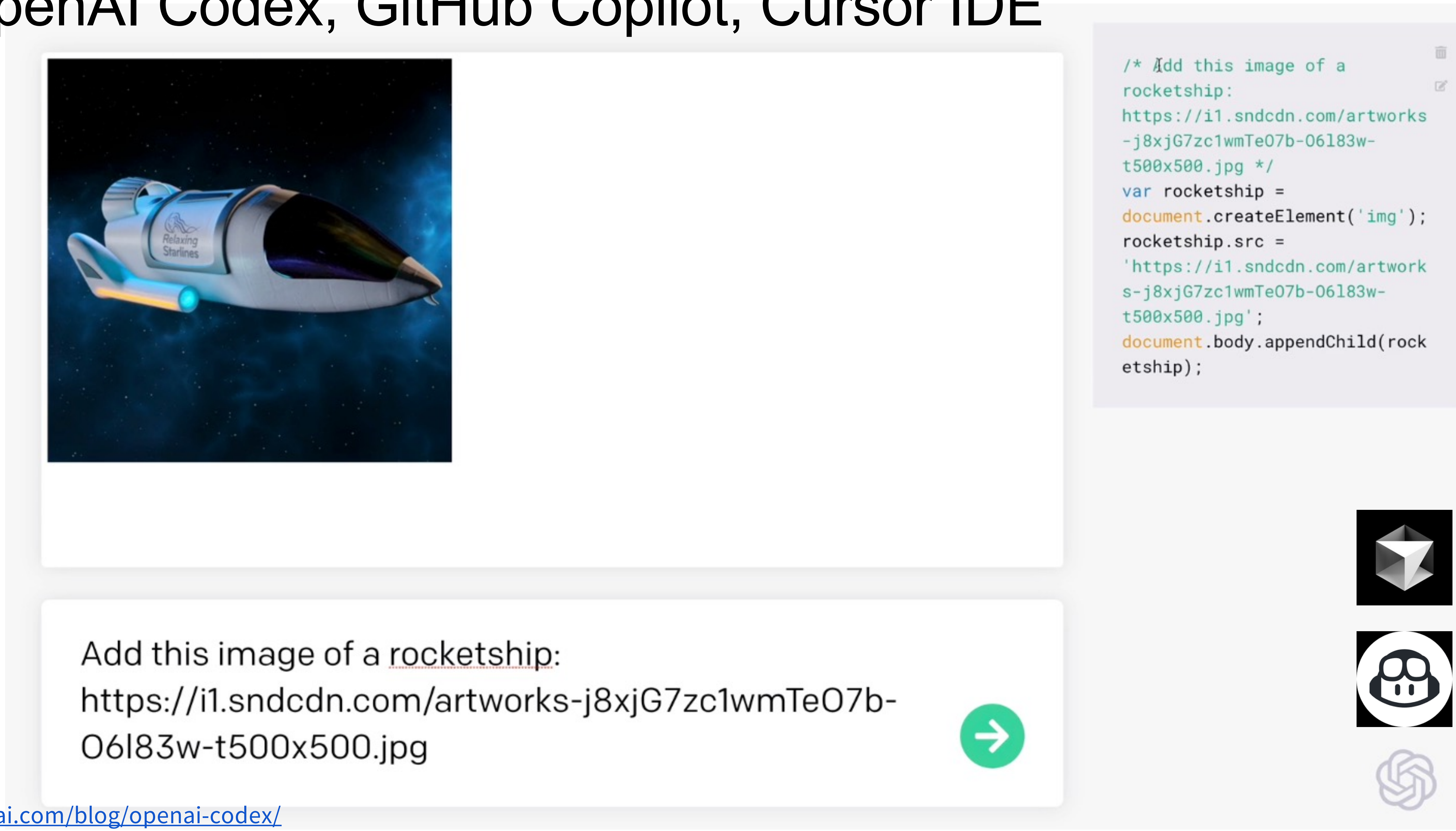
```
VIOLA:
Why, Salisbury must find his flesh and thought
That which I am not aps, not a man and in fire,
To show the reining of the raven and the wars
To grace my hand reproach within, and not a fair are hand,
That Caesar and my goodly father's world;
When I was heaven of presence and our fleets,
We 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.
```


This repository
Search
Explore
Gist
Blog
Help
karpathy
torvalds / linux
Watch
3,711
Star
23,054
Fork
9,141
Linux kernel source tree
520,037 commits
1 branch
420 releases
5,039 contributors
Code
Pull requests
74
Pulse
Graphs
branch: master
linux / +
Merge branch 'drm-fixes' of git://people.freedesktop.org/~airlied/linux
torvalds authored 9 hours ago
latest commit 4b1706927d
Documentation
Merge git://git.kernel.org/pub/scm/linux/kernel/git/nab/target-pending
6 days ago
arch
Merge branch 'x86-urgent-for-linus' of git://git.kernel.org/pub/scm/l…
a day ago
block
block: discard bdi_unregister() in favour of bdi_destroy()
9 days ago
crypto
Merge git://git.kernel.org/pub/scm/linux/kernel/git/herbert/crypto-2.6
10 days ago
drivers
Merge branch 'drm-fixes' of git://people.freedesktop.org/~airlied/linux
9 hours ago
firmware
firmware/ihex2fw.c: restore missing default in switch statement
2 months ago
fs
vfs: read file_handle only once in handle_to_path
4 days ago
include
Merge branch 'perf-urgent-for-linus' of git://git.kernel.org/pub/scm/…
a day ago
init
init: fix regression by supporting devices with major:minor:offset fo…
a month ago
HTTPS clone URL
https://github.c
You can clone with HTTPS, SSH, or Subversion.
Clone in Desktop
Download ZIP

```c
static void do_command(struct seq_file *m, void *v)
{
  int column = 32 << (cmd[2] & 0x80);
  if (state)
    cmd = (int)(int_state ^ (in_8(&ch->ch_flags) & Cmd) ? 2 : 1);
  else
    seq = 1;
  for (i = 0; i < 16; i++) {
    if (k & (1 << 1))
      pipe = (in_use & UMXTHREAD_UNCCA) +
        ((count & 0x00000000fffffff8) & 0x000000f) << 8;
    if (count == 0)
      sub(pid, ppc_md.kexec_handle, 0x20000000);
    pipe_set_bytes(i, 0);
  }
  /* Free our user pages pointer to place camera if all dash */
  subsystem_info = &of_changes[PAGE_SIZE];
  rek_controls(offset, idx, &soffset);
  /* Now we want to deliberately put it to device */
  control_check_polarity(&context, val, 0);
  for (i = 0; i < COUNTER; i++)
    seq_puts(s, "policy ");
}
```

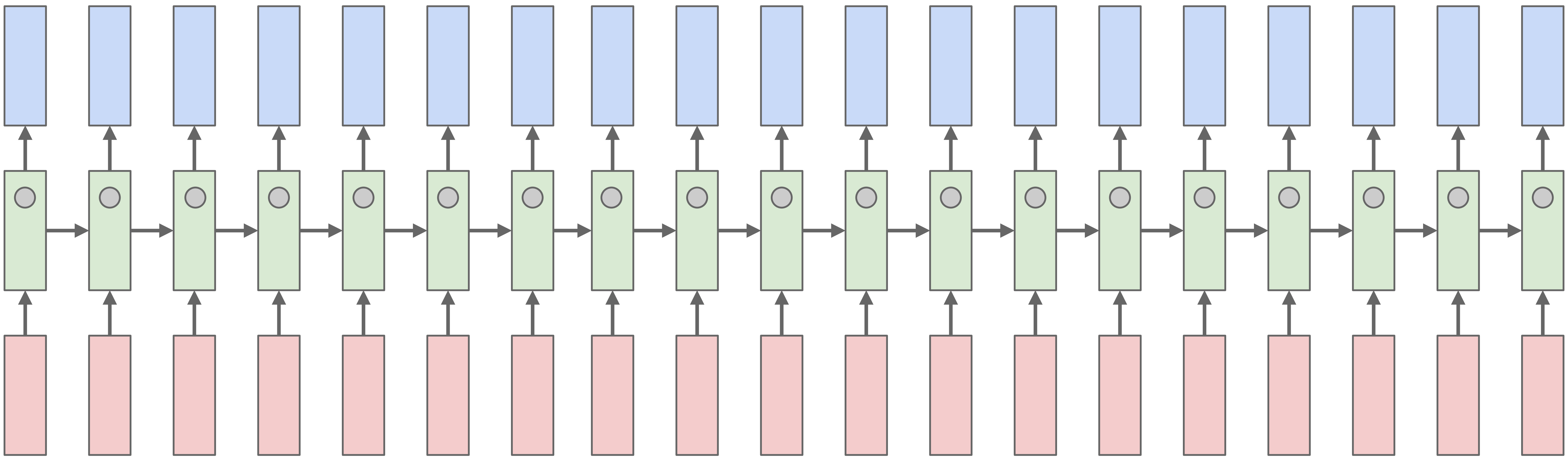
Generated C code

OpenAI Codex, GitHub Copilot, Cursor IDE

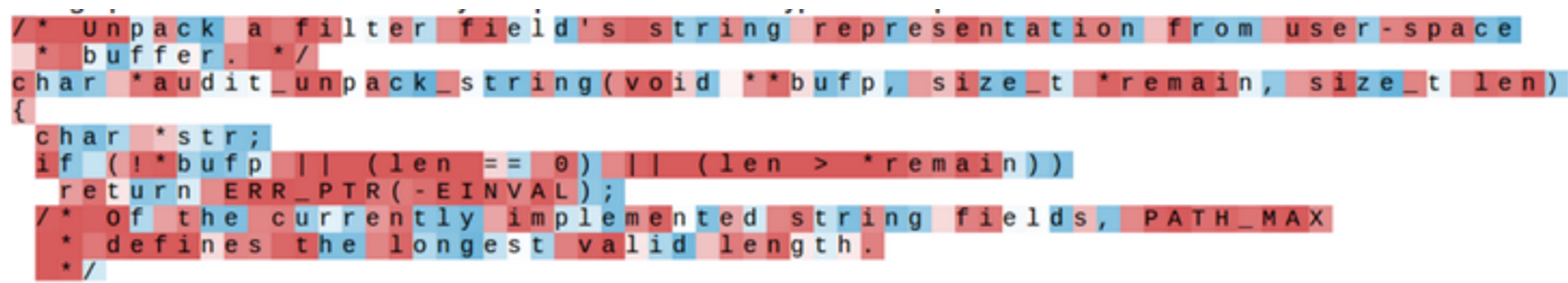


https://openai.com/blog/openai-codex/

Searching for interpretable cells

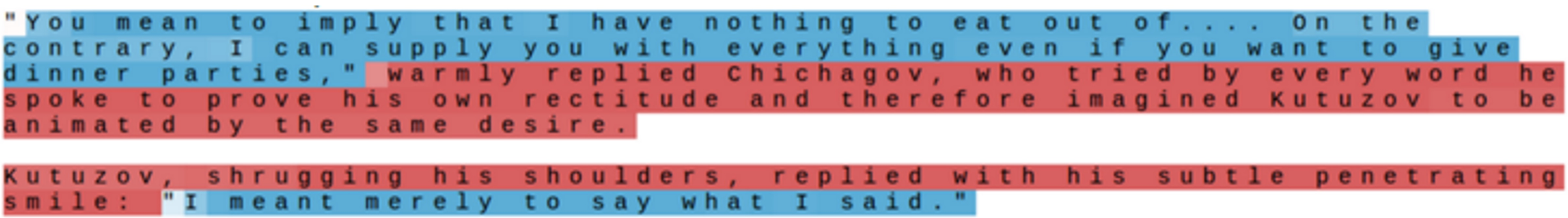


Searching for interpretable cells



Karpathy, Johnson, and Fei-Fei: Visualizing and Understanding Recurrent Networks, ICLR Workshop 2016

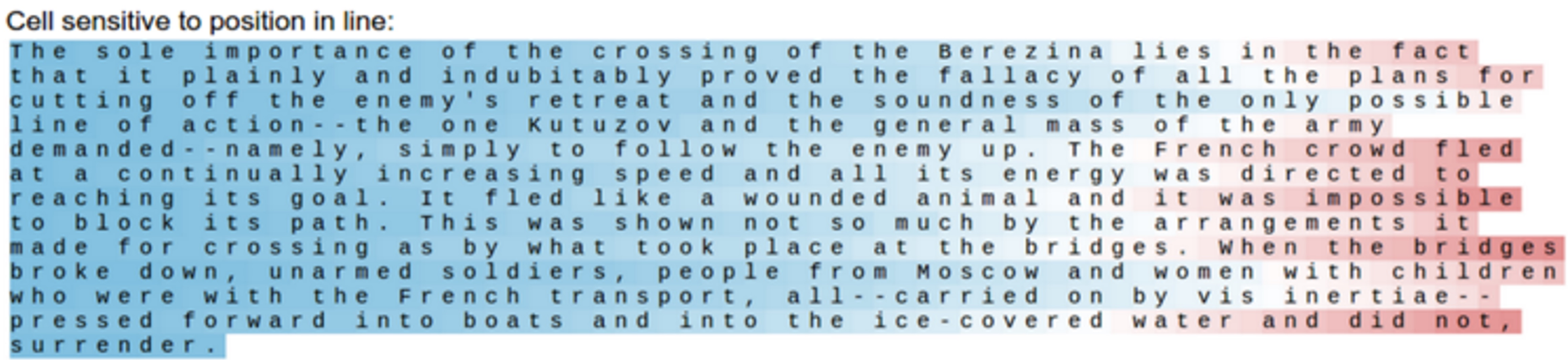
Searching for interpretable cells



quote detection cell

Karpathy, Johnson, and Fei-Fei: Visualizing and Understanding Recurrent Networks, ICLR Workshop 2016

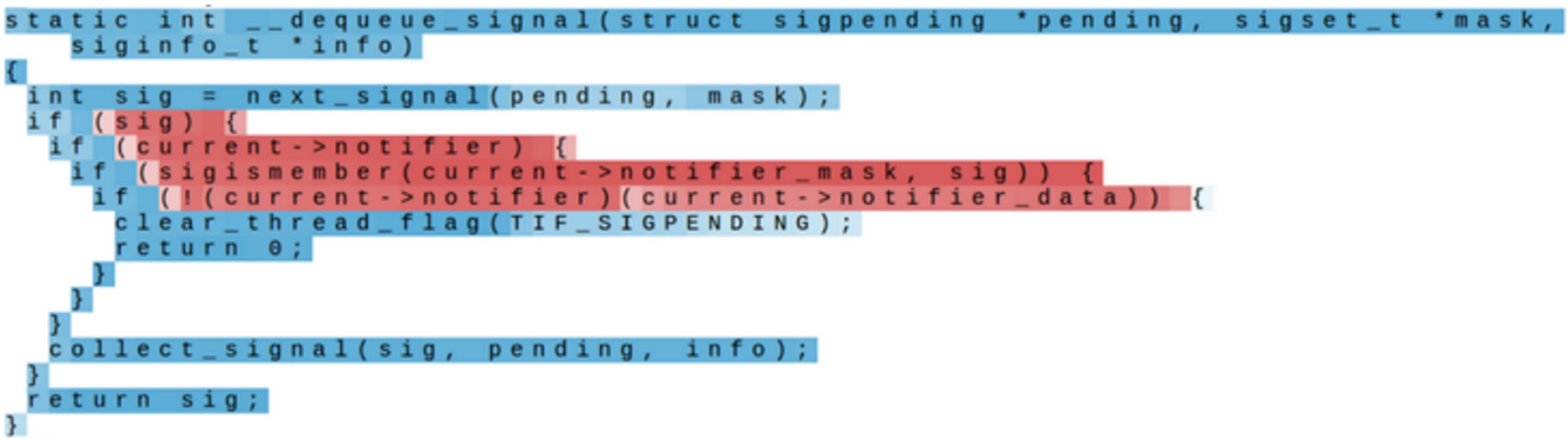
Searching for interpretable cells



line length tracking cell

Karpathy, Johnson, and Fei-Fei: Visualizing and Understanding Recurrent Networks, ICLR Workshop 2016

Searching for interpretable cells



if statement cell

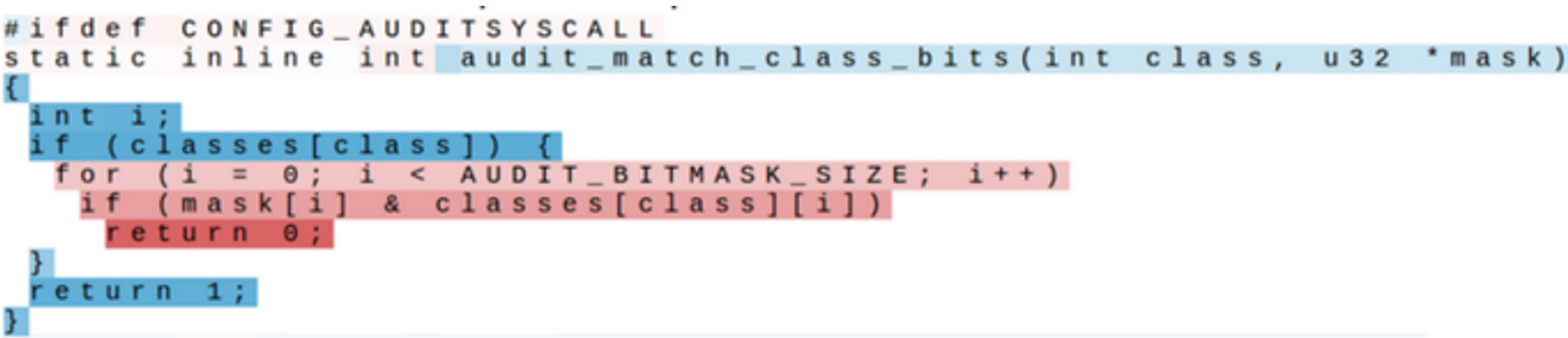
Karpathy, Johnson, and Fei-Fei: Visualizing and Understanding Recurrent Networks, ICLR Workshop 2016

Searching for interpretable cells

quote/comment cell

Karpathy, Johnson, and Fei-Fei: Visualizing and Understanding Recurrent Networks, ICLR Workshop 2016

Searching for interpretable cells



code depth cell

Karpathy, Johnson, and Fei-Fei: Visualizing and Understanding Recurrent Networks, ICLR Workshop 2016

RNN tradeoffs

RNN Advantages:

- Can process any length of the input (no context length)
- Computation for step *t* can (in theory) use information from many steps back
- Model size does not increase for longer input
- The same weights are applied on every timestep, so there is symmetry in how inputs are processed.

RNN Disadvantages:

- Recurrent computation is slow
- In practice, difficult to access information from many steps back

Image Captioning

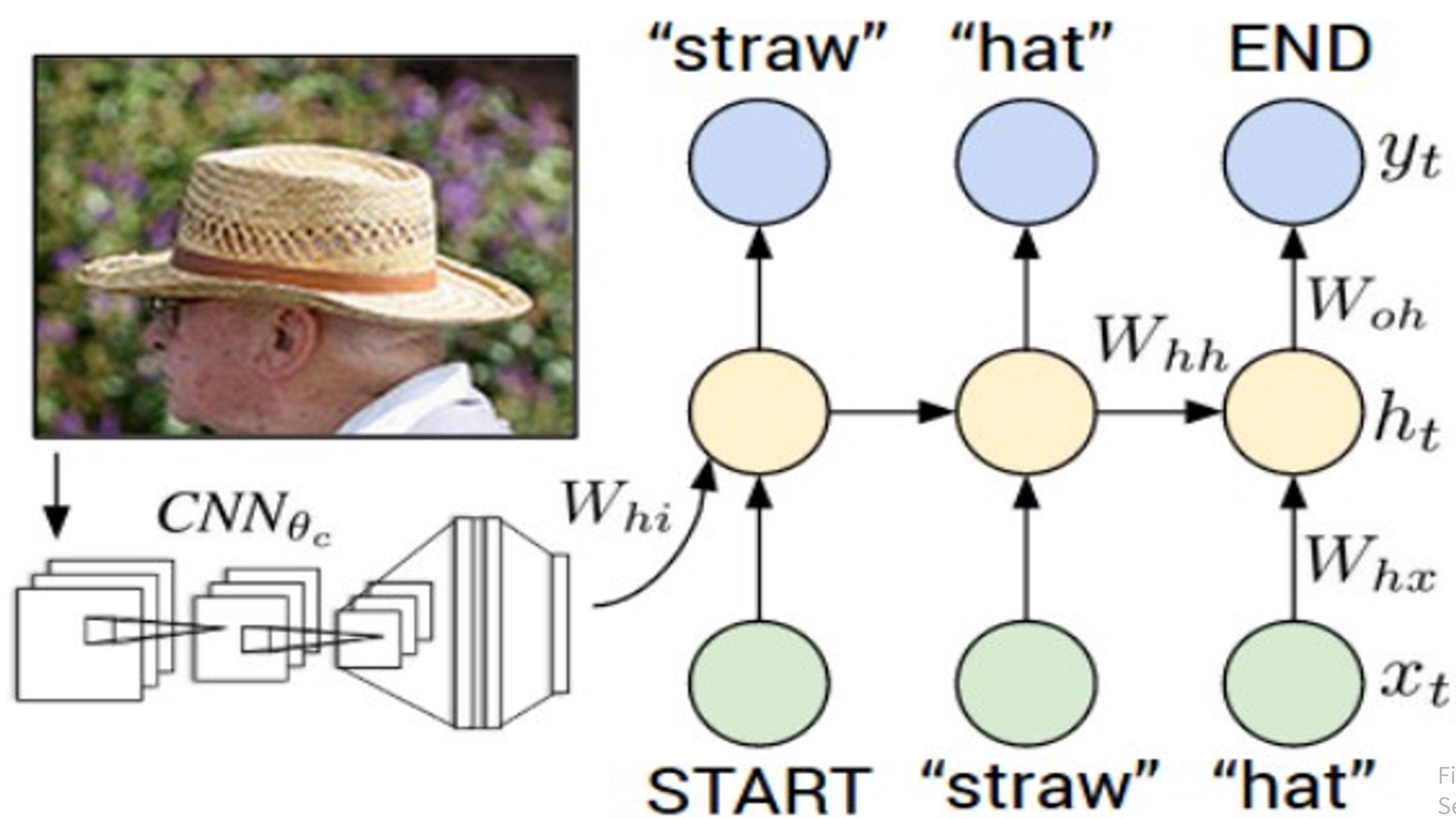
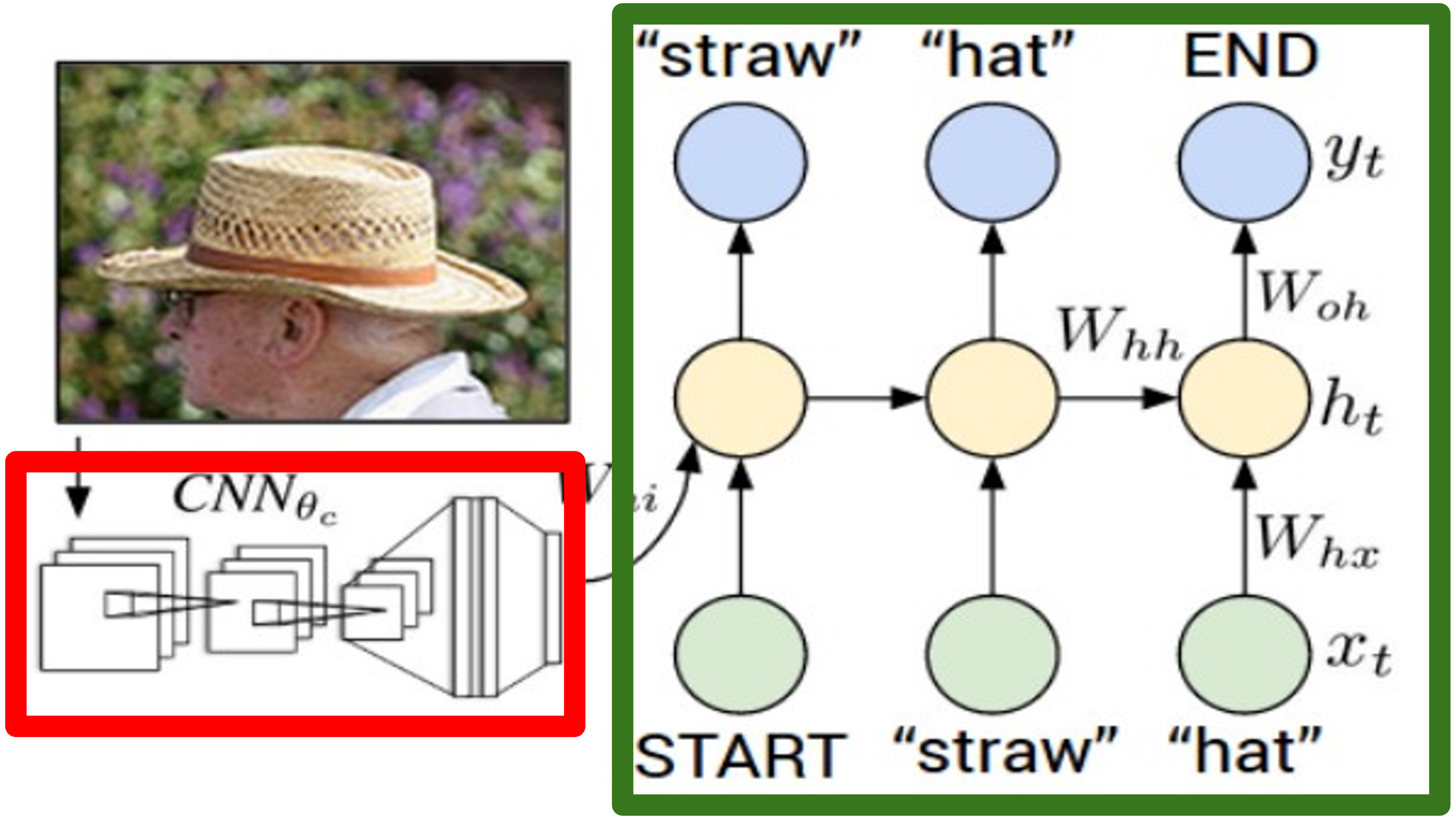


Figure from Karpathy et a, "Deep Visual-Semantic Alignments for Generating Image Descriptions", CVPR 2015; figure copyright IEEE, 2015.
Reproduced for educational purposes.

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

This image is CC0 public domain

test image

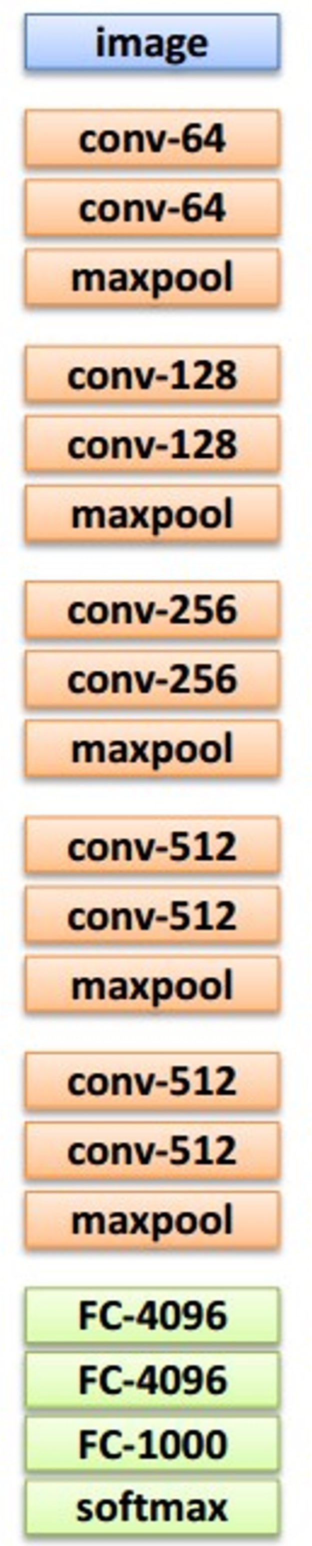


image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
conv-256
conv-256
maxpool
conv-512
conv-512
maxpool
conv-512
conv-512
maxpool
FC-4096
FC-4096
FC-1000
softmax

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
softmax
X

test image

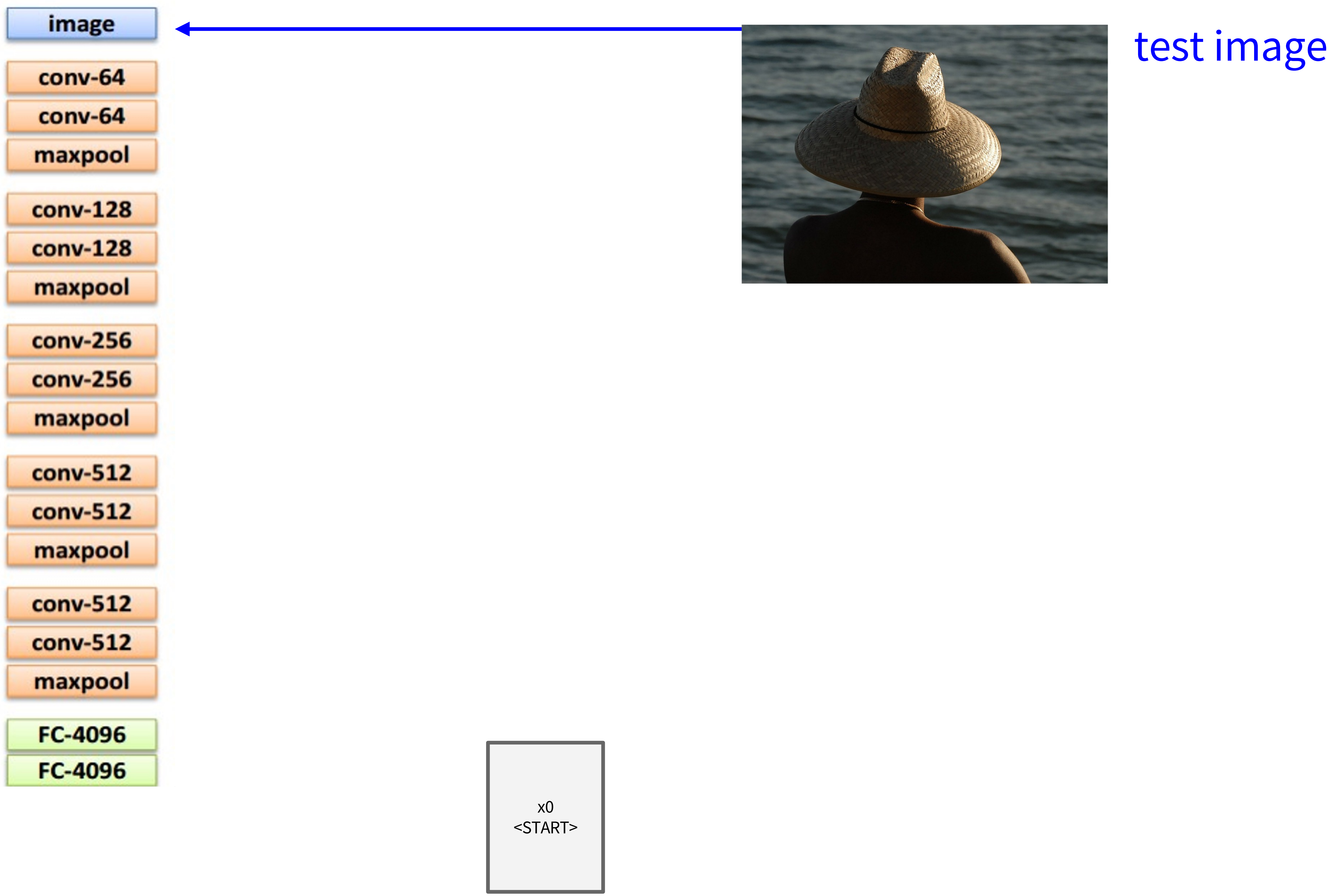
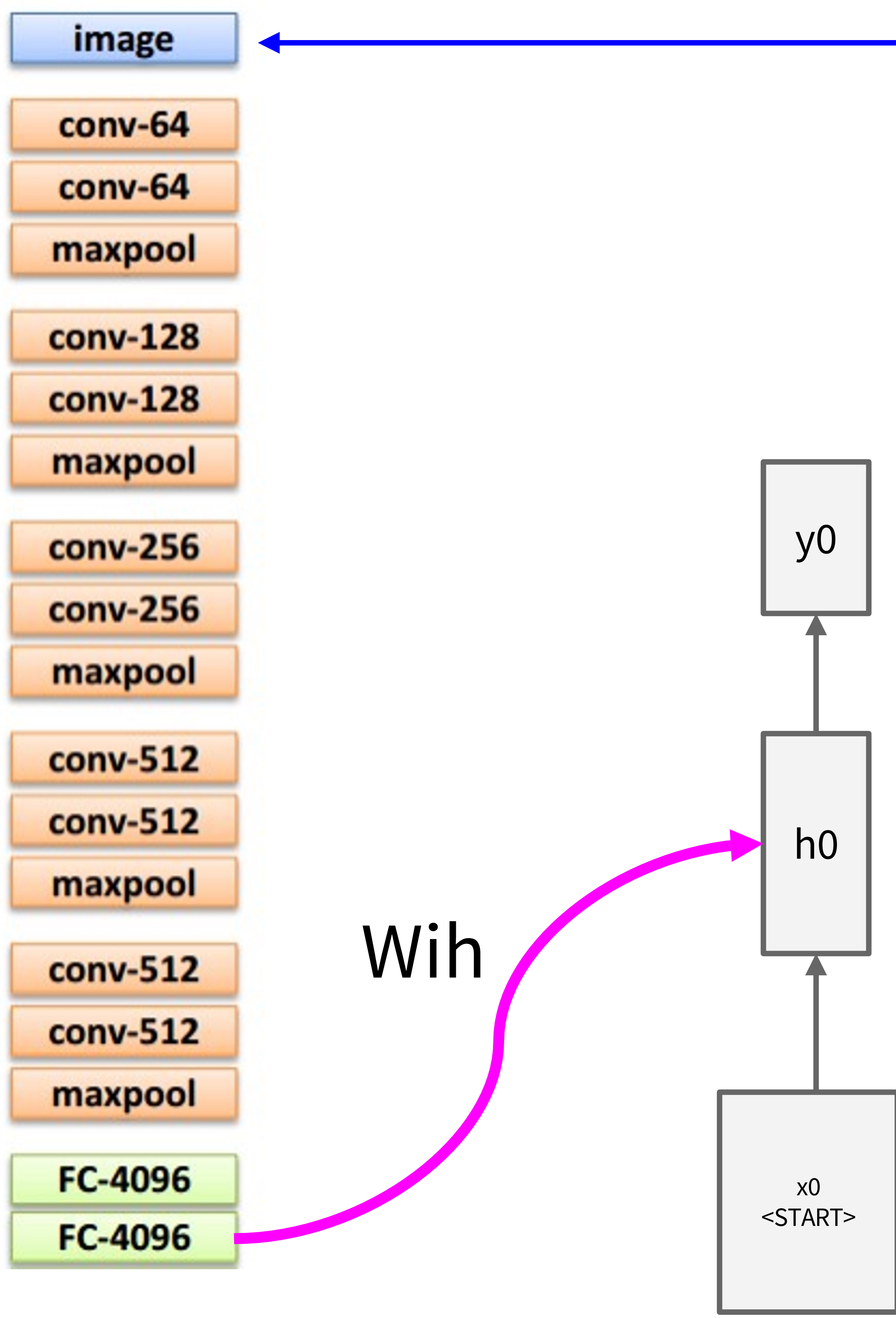


image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
conv-256
conv-256
maxpool
conv-512
conv-512
maxpool
conv-512
conv-512
maxpool
FC-4096
FC-4096
test image
x0
<START>



test image

before:

$$h = \tanh(W_{xh} * x + W_{hh} * h)$$

now:

$$h = \tanh(W_{xh} * x + W_{hh} * h + W_{ih} * v)$$

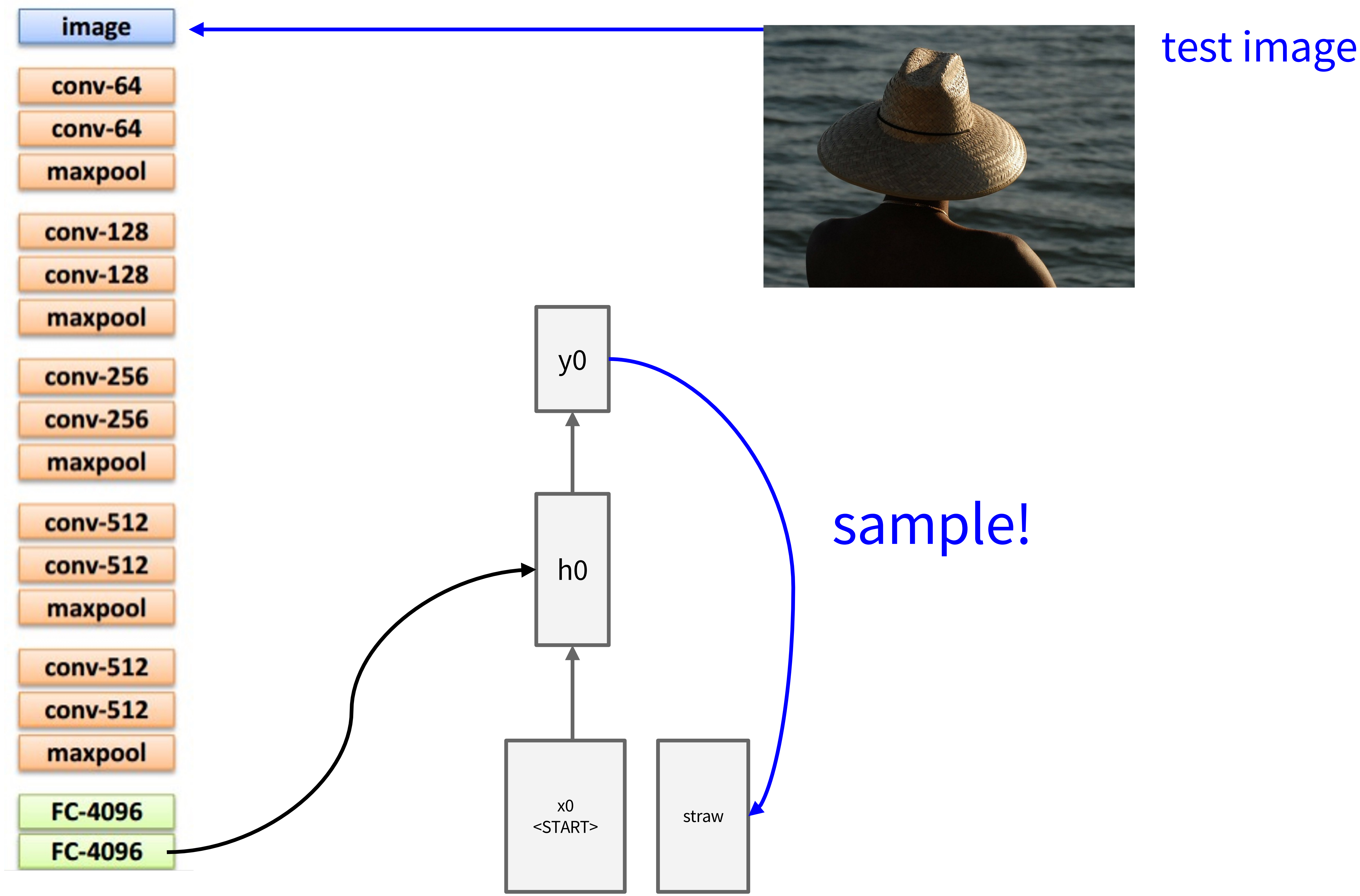


image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
conv-256
conv-256
maxpool
conv-512
conv-512
maxpool
conv-512
conv-512
maxpool
FC-4096
FC-4096
test image
y0
sample!
h0
x0
<START>
straw

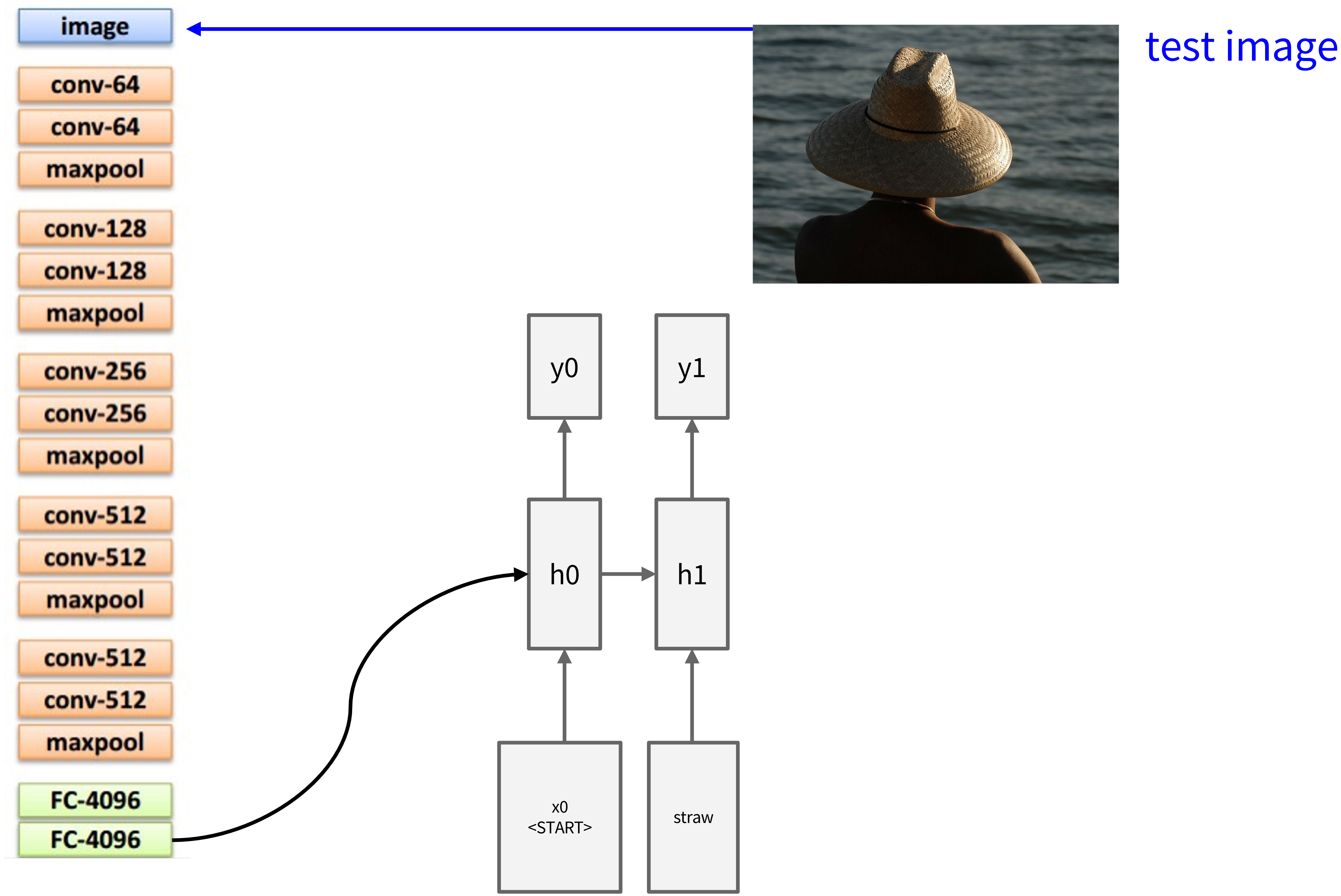


image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
conv-256
conv-256
maxpool
conv-512
conv-512
maxpool
conv-512
conv-512
maxpool
FC-4096
FC-4096
test image
y0
y1
h0
h1
x0
<START>
straw

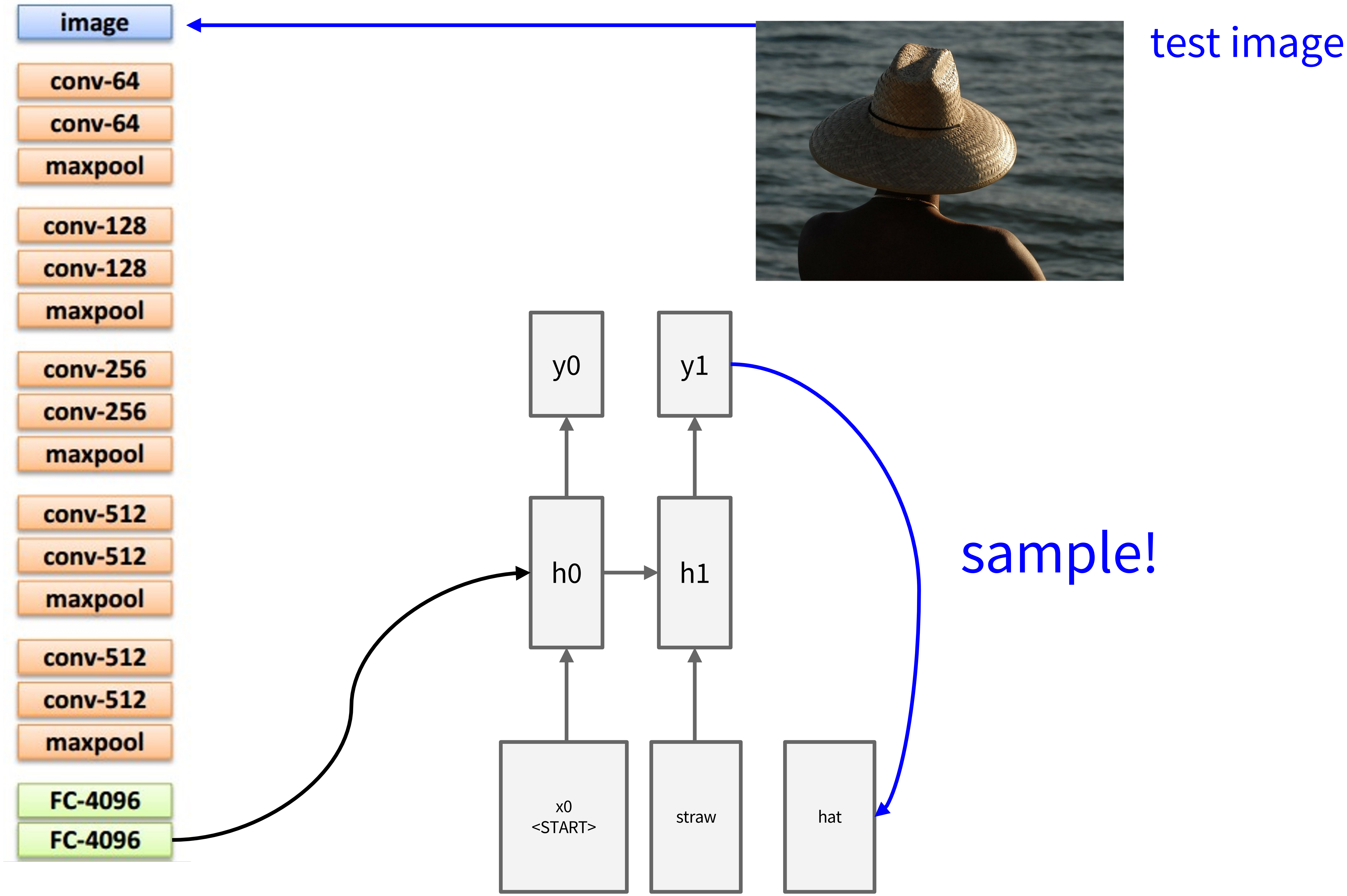


image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
conv-256
conv-256
maxpool
conv-512
conv-512
maxpool
conv-512
conv-512
maxpool
FC-4096
FC-4096
test image
y0
y1
h0
h1
sample!
x0
<START>
straw
hat

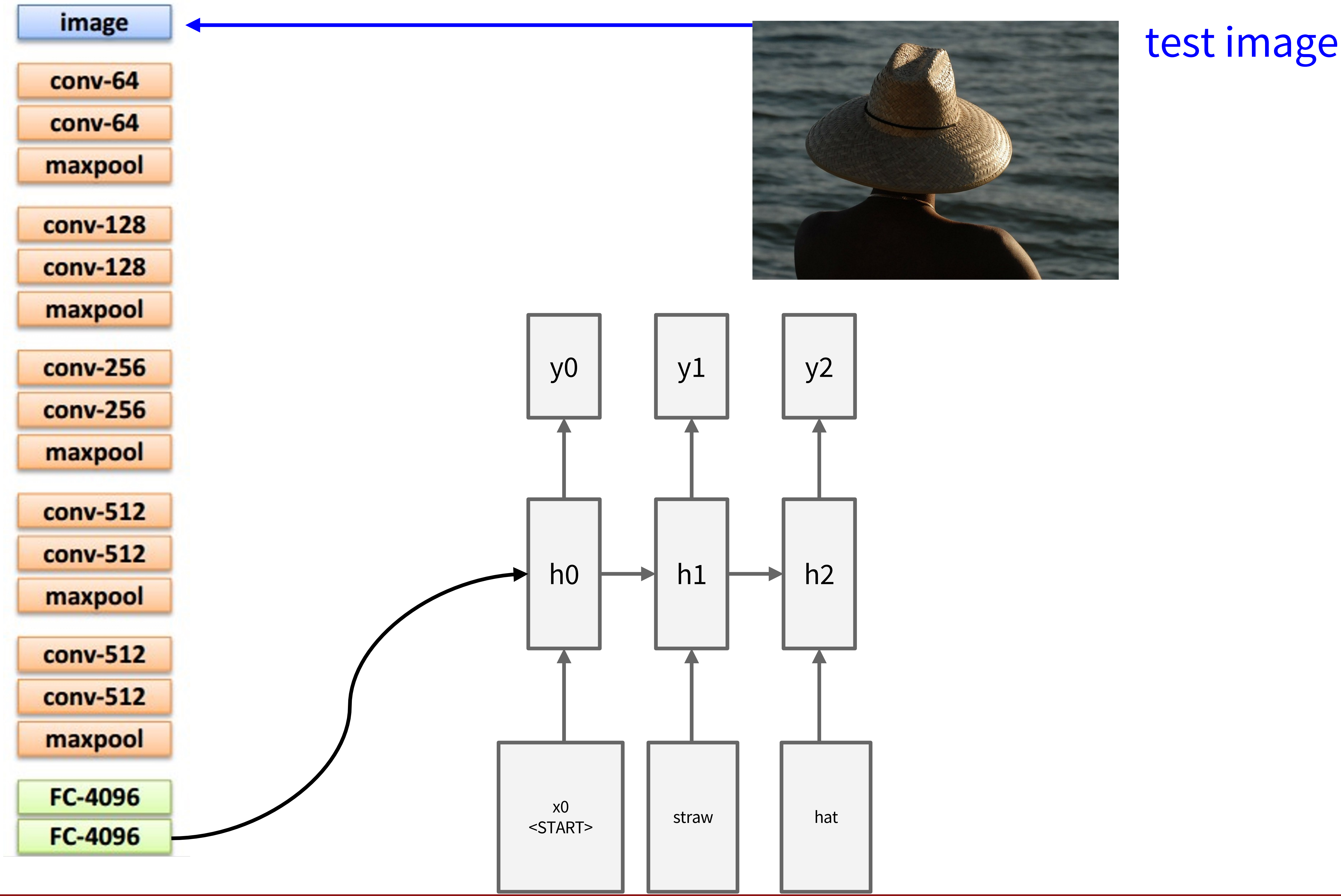


image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
conv-256
conv-256
maxpool
conv-512
conv-512
maxpool
conv-512
conv-512
maxpool
FC-4096
FC-4096
test image
y0
y1
y2
h0
h1
h2
x0
<START>
straw
hat

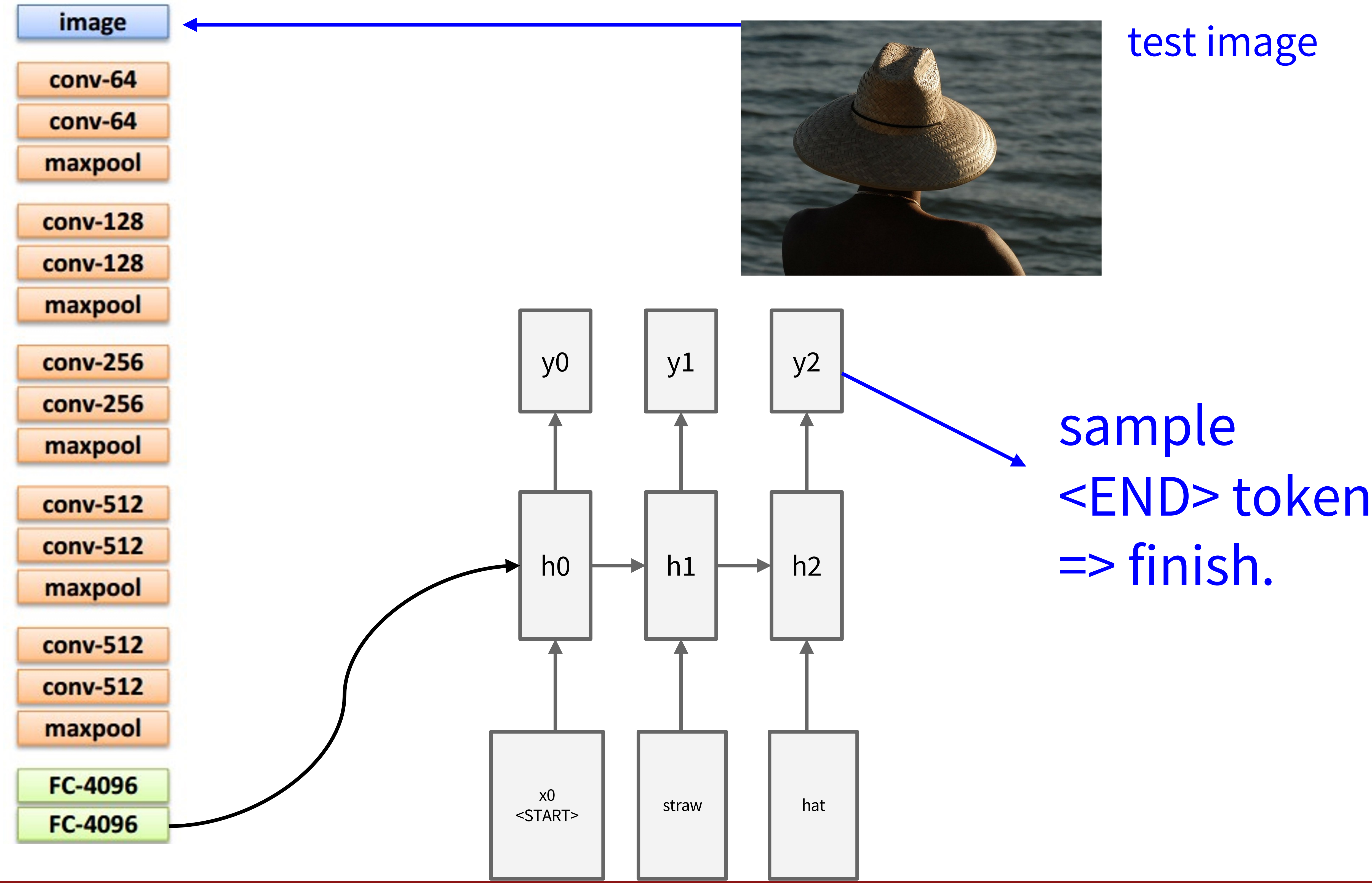


image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
conv-256
conv-256
maxpool
conv-512
conv-512
maxpool
conv-512
conv-512
maxpool
FC-4096
FC-4096
test image
y0
y1
y2
sample
<END> token
=> finish.
h0
h1
h2
x0
<START>
straw
hat

Captions generated using neuraltalk2
All images are CC0 Public domain: cat suitcase, cat tree, dog, bear, surfers, tennis, giraffe, motorcycle

Image Captioning: Example Results

A cat sitting on a suitcase on the floor

A cat is sitting on a tree branch

A dog is running in the grass with a frisbee

A white teddy bear sitting in the grass

Two people walking on the beach with surfboards

A tennis player in action on the court

Two giraffes standing in a grassy field

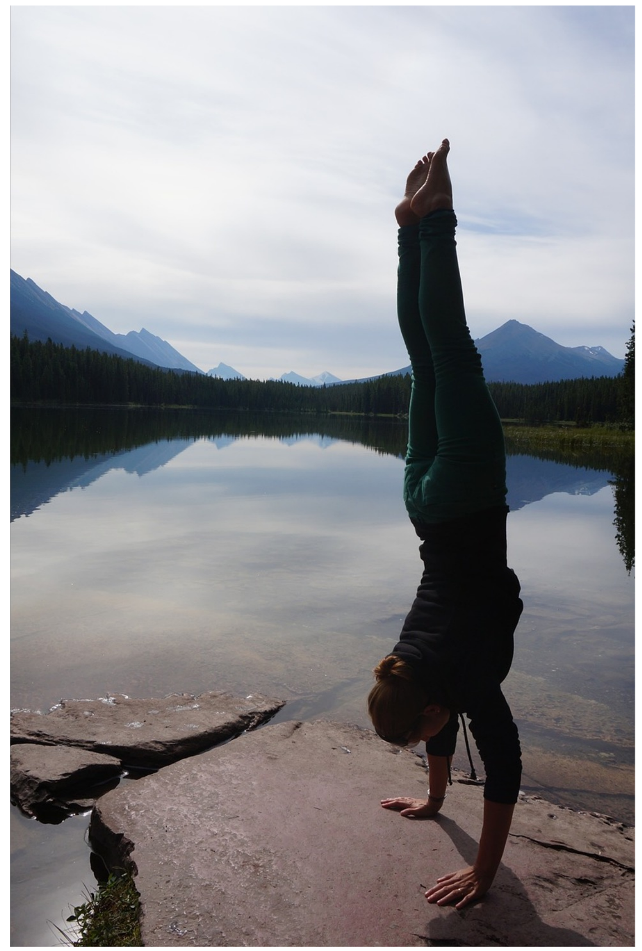
A man riding a dirt bike on a dirt track

Captions generated using neuraltalk2
All images are CC0 Public domain: fur coat, handstand, spider web, baseball

Image Captioning: Failure Cases

A woman is holding a cat in her hand

A person holding a computer mouse on a desk



A woman standing on a beach holding a surfboard

A bird is perched on a tree branch

A man in a baseball uniform throwing a ball

Visual Question Answering (VQA)

Q: What endangered animal is featured on the truck?	**Q: Where will the driver go if turning right?**	**Q: When was the picture taken?**	**Q: Who is under the umbrella?**
A: **A bald eagle.**	A: **Onto 24 ¾ Rd.**	A: **During a wedding.**	A: **Two women.**
A: A sparrow.	A: Onto 25 ¾ Rd.	A: During a bar mitzvah.	A: A child.
A: A humming bird.	A: Onto 23 ¾ Rd.	A: During a funeral.	A: An old man.
A: A raven.	A: Onto Main Street.	A: During a Sunday church service	A: A husband and a wife.

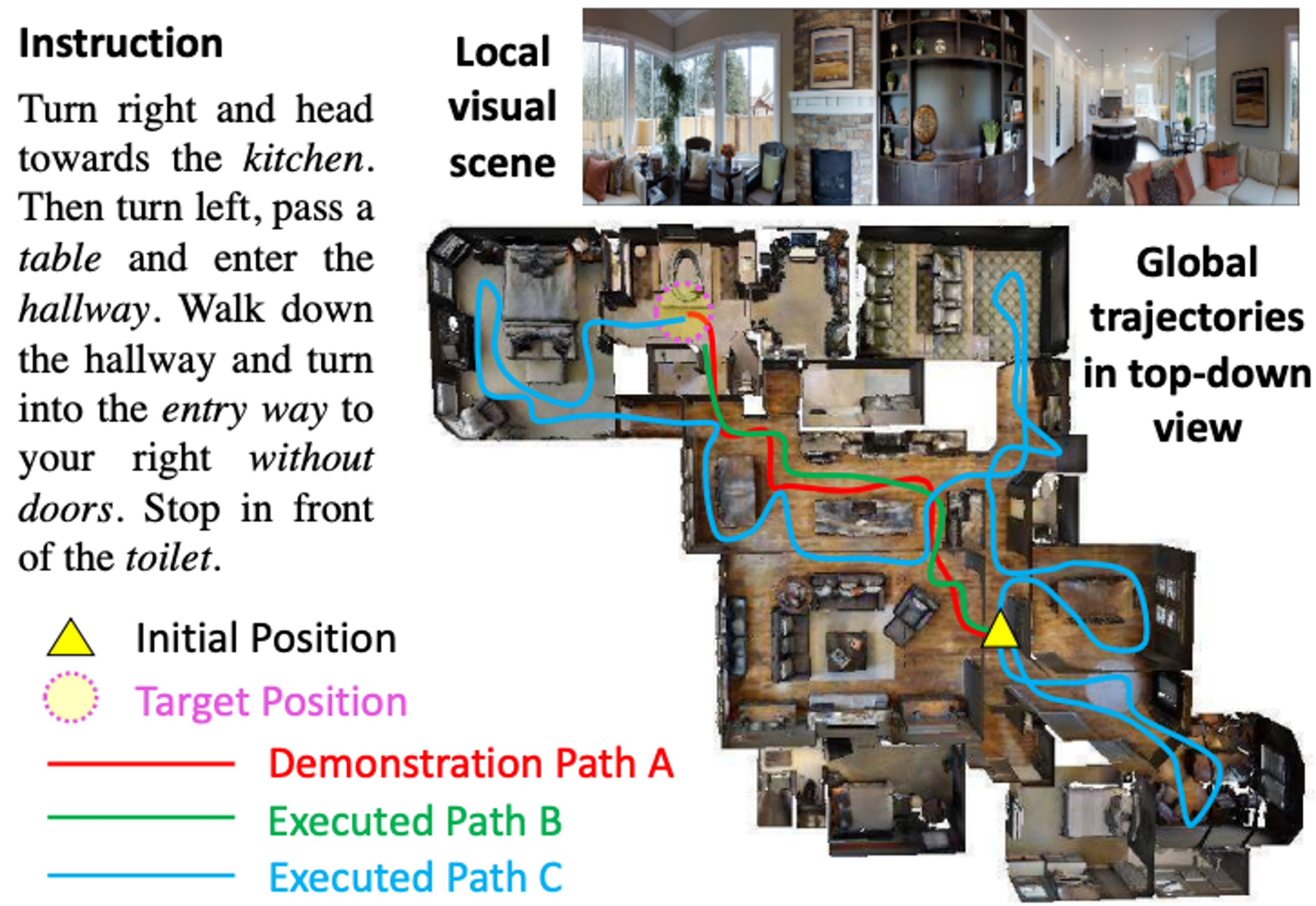
Agrawal et al, "VQA: Visual Question Answering", ICCV 2015
Zhu et al, "Visual 7W: Grounded Question Answering in Images", CVPR 2016

Visual Dialog: Conversations about images

Das et al, "Visual Dialog", CVPR 2017

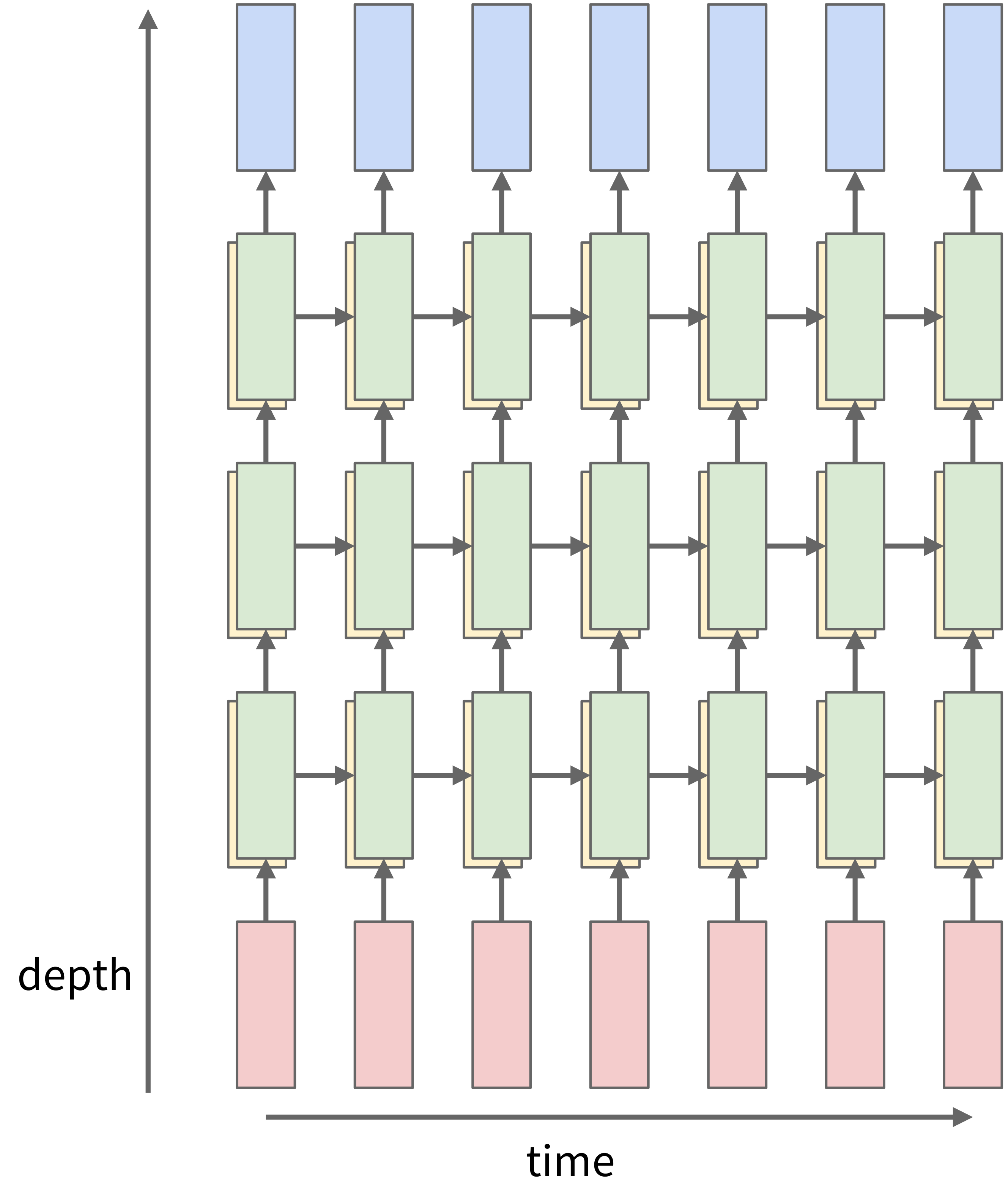
Visual Language Navigation: Go to the living room

Agent encodes instructions in language and uses an RNN to generate a series of movements as the visual input changes after each move.



Wang et al, "Reinforced Cross-Modal Matching and Self-Supervised Imitation Learning for Vision-Language Navigation", CVPR 2018

Multilayer RNNs



RNN Variants: Long Short Term Memory (LSTM)

Vanilla RNN

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

LSTM

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

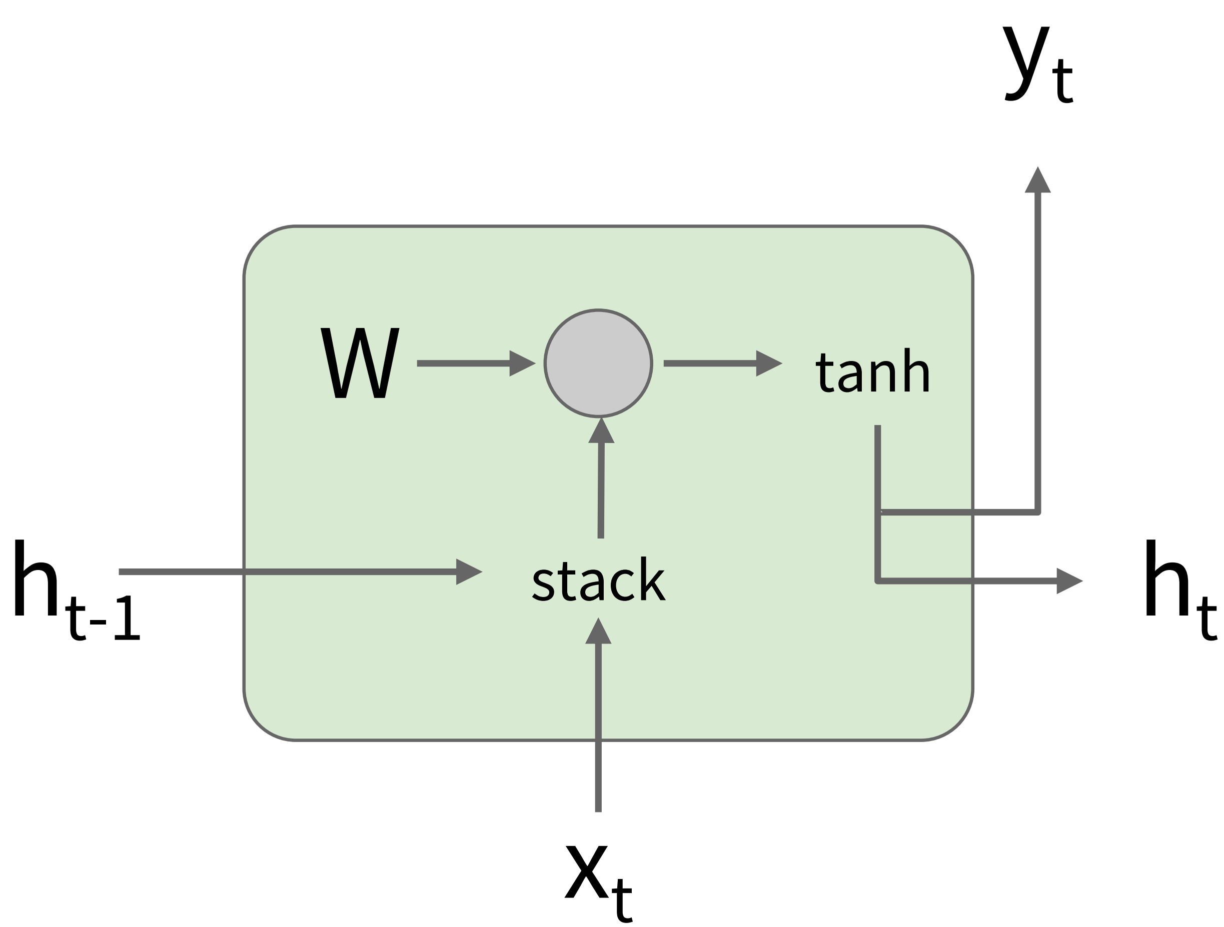
$$c_t = f \odot c_{t-1} + i \odot g$$

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

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

Vanilla RNN Gradient Flow

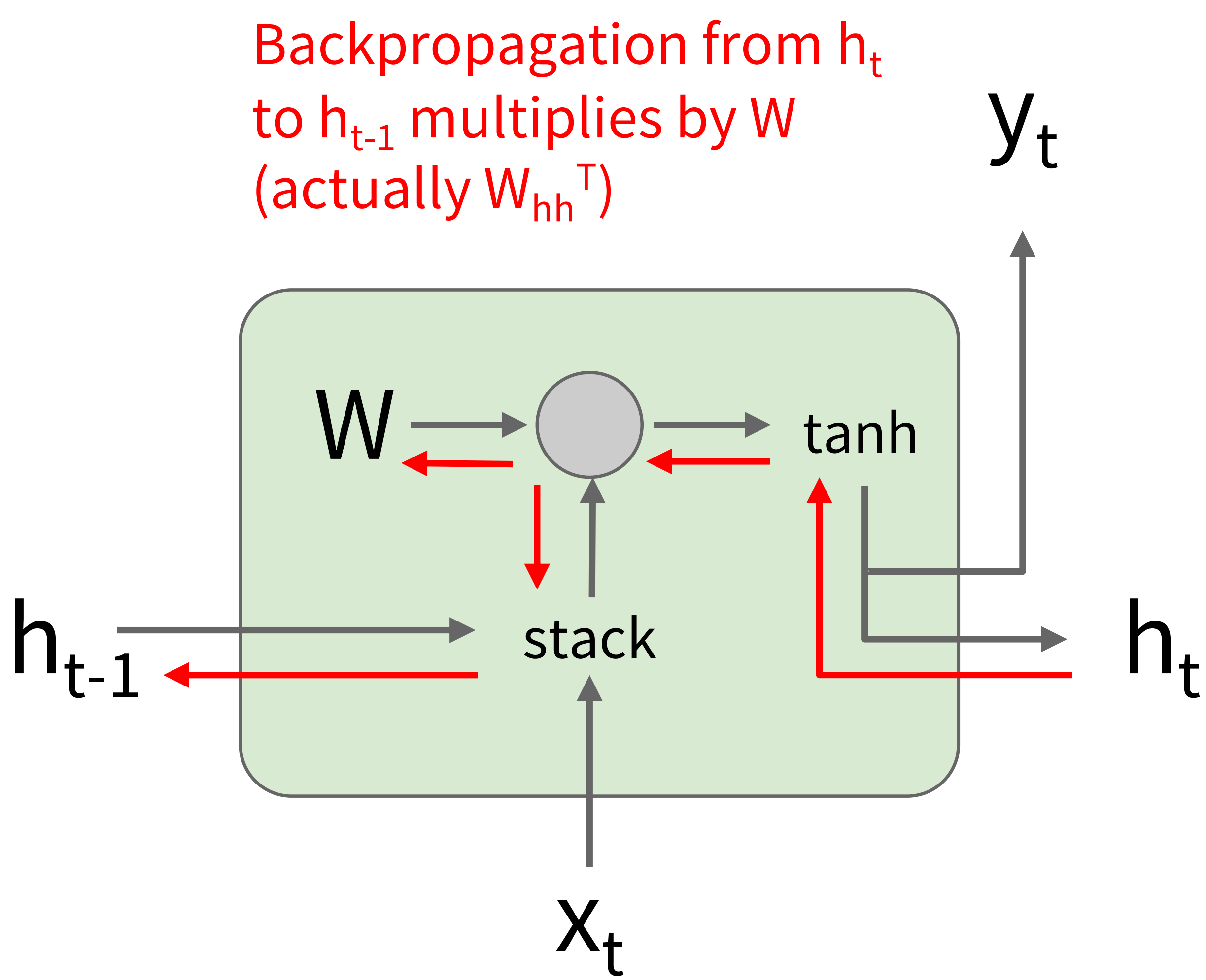
Bengio et al, “Learning long-term dependencies with gradient descent is difficult”, IEEE Transactions on Neural Networks, 1994
Pascanu et al, “On the difficulty of training recurrent neural networks”, ICML 2013



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

Vanilla RNN Gradient Flow

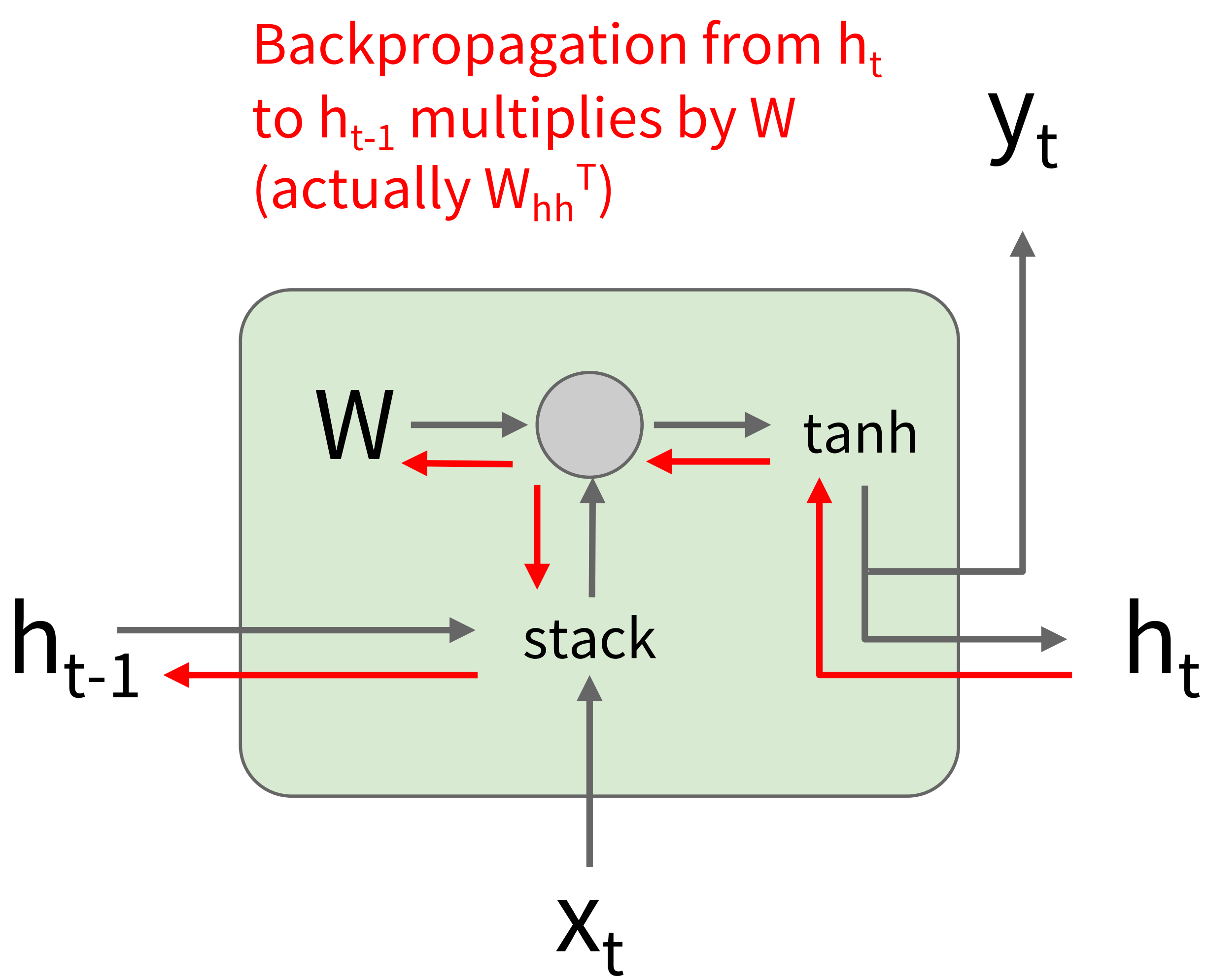
Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013



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

Vanilla RNN Gradient Flow

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

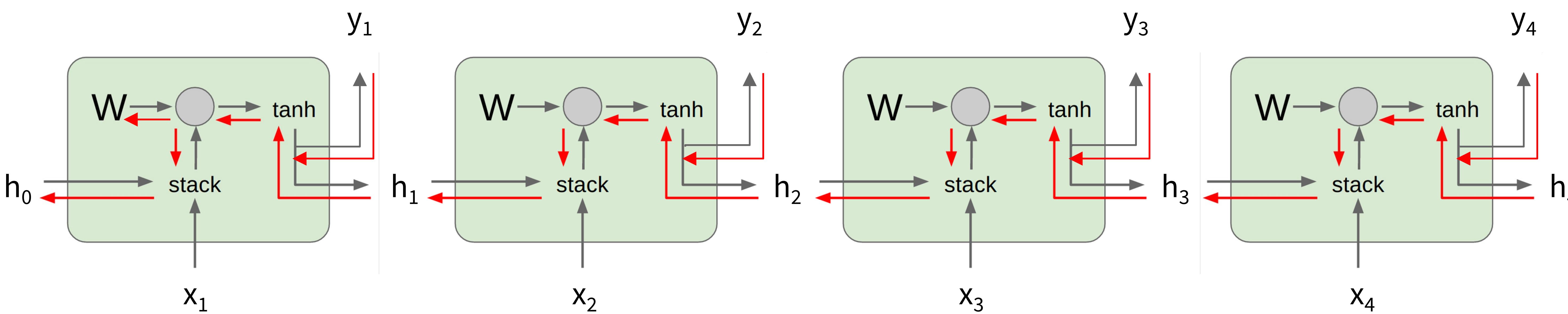


$$
\begin{aligned}
h_t &= \tanh(W_{hh} h_{t-1} + W_{xh} x_t) \\
&= \tanh\left(\begin{pmatrix} W_{hh} & W_{hx} \end{pmatrix} \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \right) \\
&= \tanh\left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \right)
\end{aligned}
$$

$$
\frac{\partial h_t}{\partial h_{t-1}} = tanh'(W_{hh} h_{t-1} + W_{xh} x_t) W_{hh}
$$

Vanilla RNN Gradient Flow

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

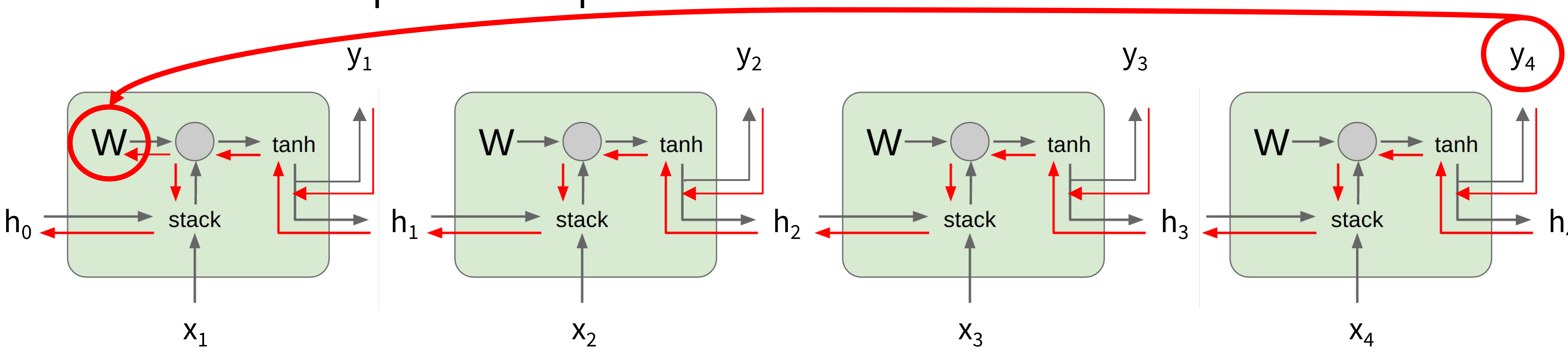


$$\frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W}$$

Vanilla RNN Gradient Flow

Gradients over multiple time steps:

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



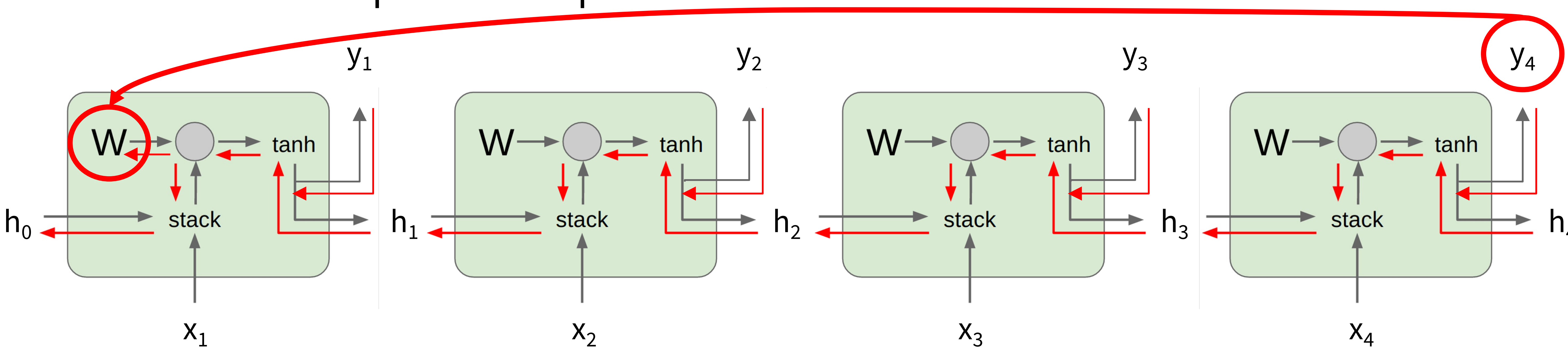
$$\frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W}$$

$$\frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial h_T} \frac{\partial h_t}{\partial h_{t-1}} \cdots \frac{\partial h_1}{\partial W}$$

Vanilla RNN Gradient Flow

Gradients over multiple time steps:

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



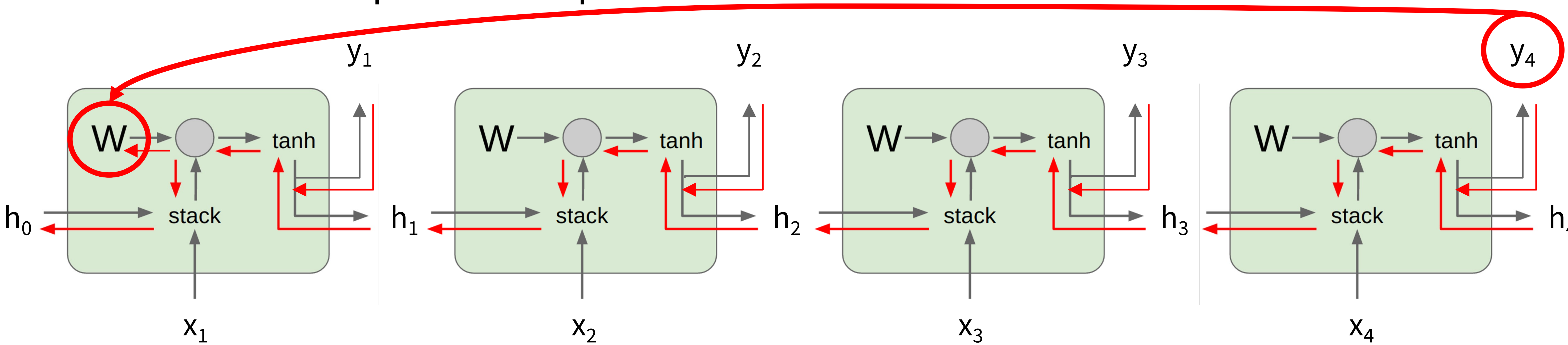
$$\frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W}$$

$$\frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial h_T} \frac{\partial h_t}{\partial h_{t-1}} \cdots \frac{\partial h_1}{\partial W} = \frac{\partial L_T}{\partial h_T} \left(\prod_{t=2}^{T} \frac{\partial h_t}{\partial h_{t-1}}\right) \frac{\partial h_1}{\partial W}$$

Vanilla RNN Gradient Flow

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

Gradients over multiple time steps:



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

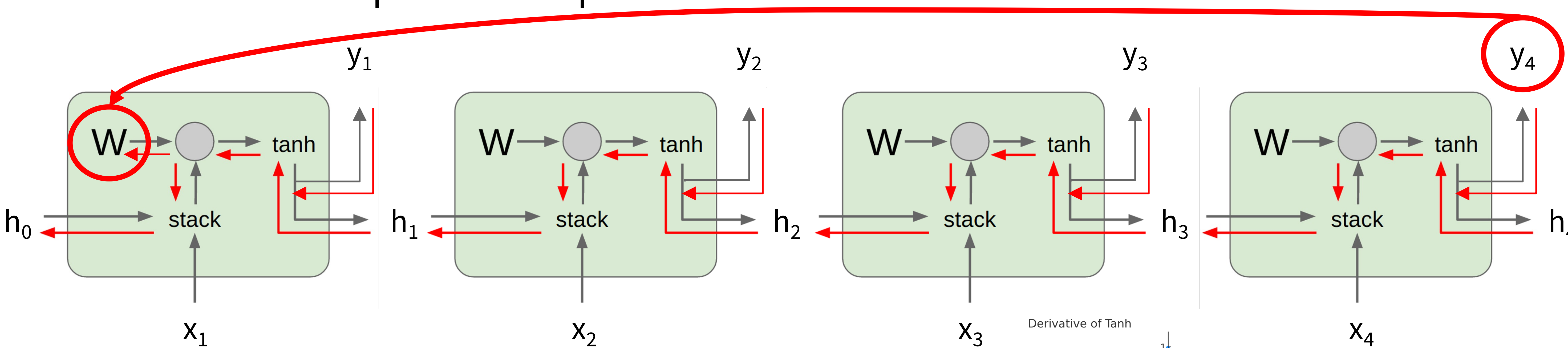
$$\frac{\partial h_t}{\partial h_{t-1}} = tanh'(W_{hh}h_{t-1} + W_{xh}x_t)W_{hh}$$

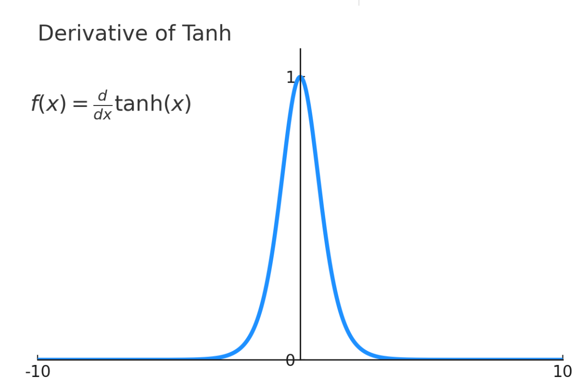
$$\frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial h_T}\frac{\partial h_t}{\partial h_{t-1}}\cdots\frac{\partial h_1}{\partial W} = \frac{\partial L_T}{\partial h_T}\left(\prod_{t=2}^{T}\frac{\partial h_t}{\partial h_{t-1}}\right)\frac{\partial h_1}{\partial W}$$

Vanilla RNN Gradient Flow

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

Gradients over multiple time steps:





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

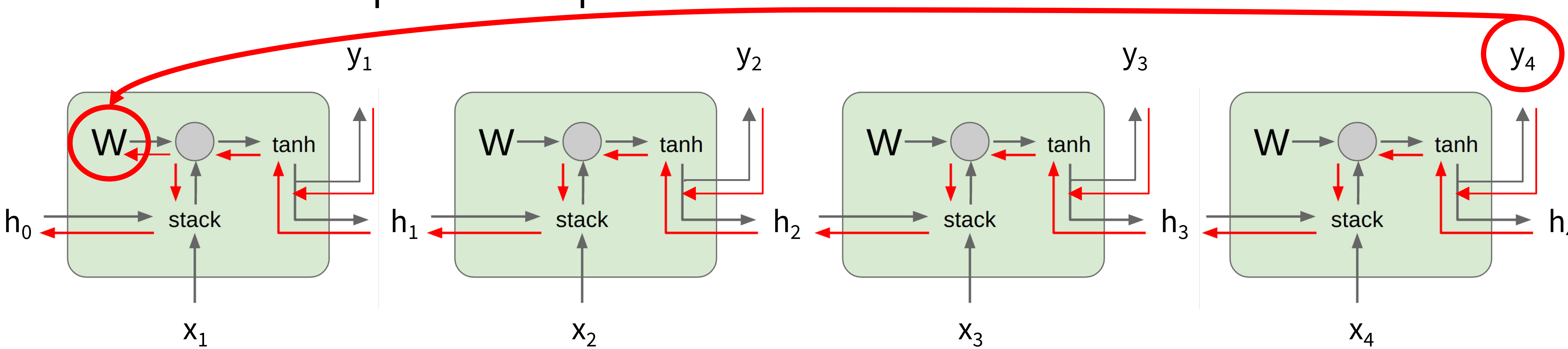
Almost always < 1
Vanishing gradients

$$\frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial h_T} \left(\prod_{t=2}^{T} \boxed{tanh'(W_{hh}h_{t-1} + W_{xh}x_t)}\right) W_{hh}^{T-1} \frac{\partial h_1}{\partial W}$$

Vanilla RNN Gradient Flow

Gradients over multiple time steps:

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



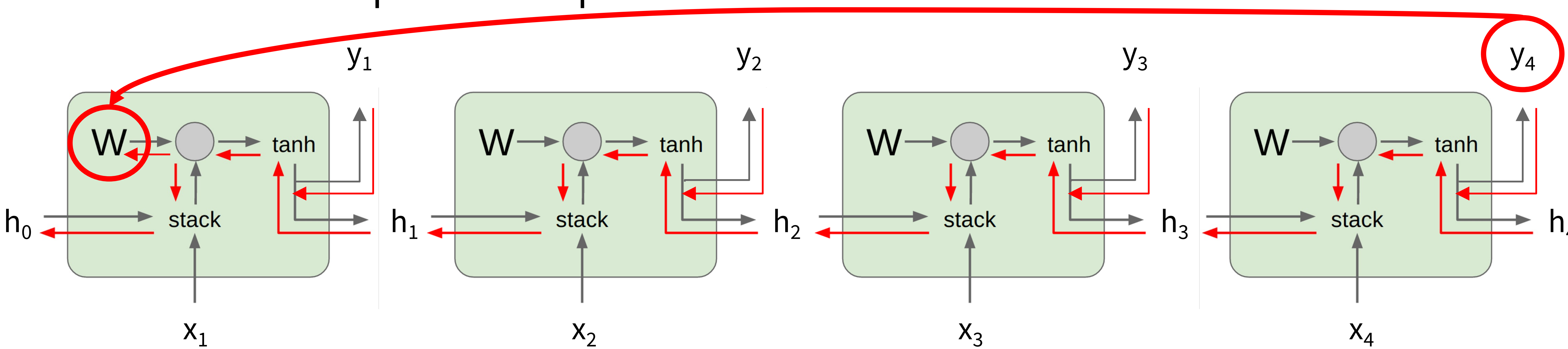
$$\frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W}$$

What if we assumed no non-linearity?

Vanilla RNN Gradient Flow

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

Gradients over multiple time steps:



What if we assumed no non-linearity?

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

$$\frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial h_T} \boxed{W_{hh}^{T-1}} \frac{\partial h_1}{\partial W}$$

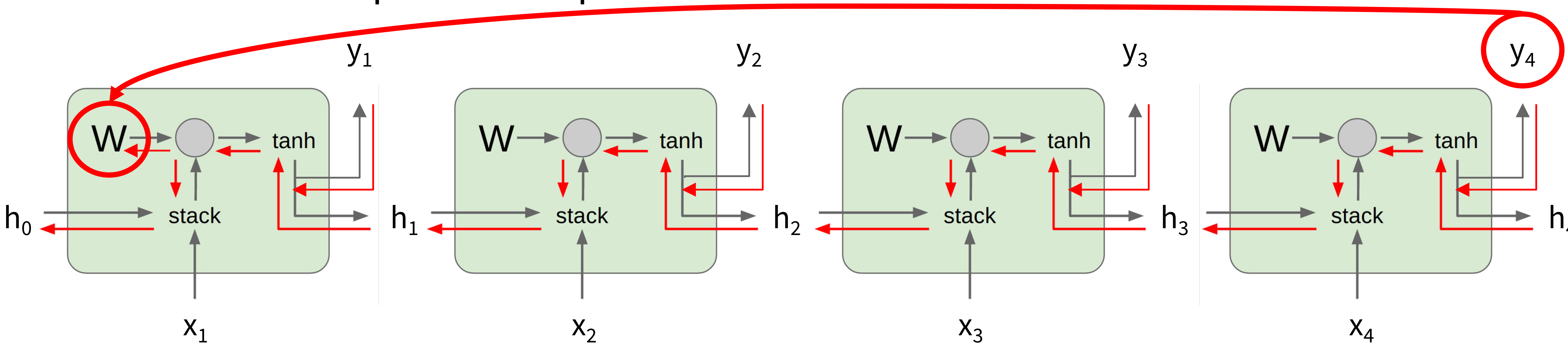
Largest singular value > 1:
Exploding gradients

Largest singular value < 1:
Vanishing gradients

Vanilla RNN Gradient Flow

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

Gradients over multiple time steps:



What if we assumed no non-linearity?

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

$$\frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial h_T} \boxed{W_{hh}^{T-1}} \frac{\partial h_1}{\partial W}$$

Largest singular value > 1:
Exploding gradients

→ Gradient clipping: Scale gradient if its norm is too big

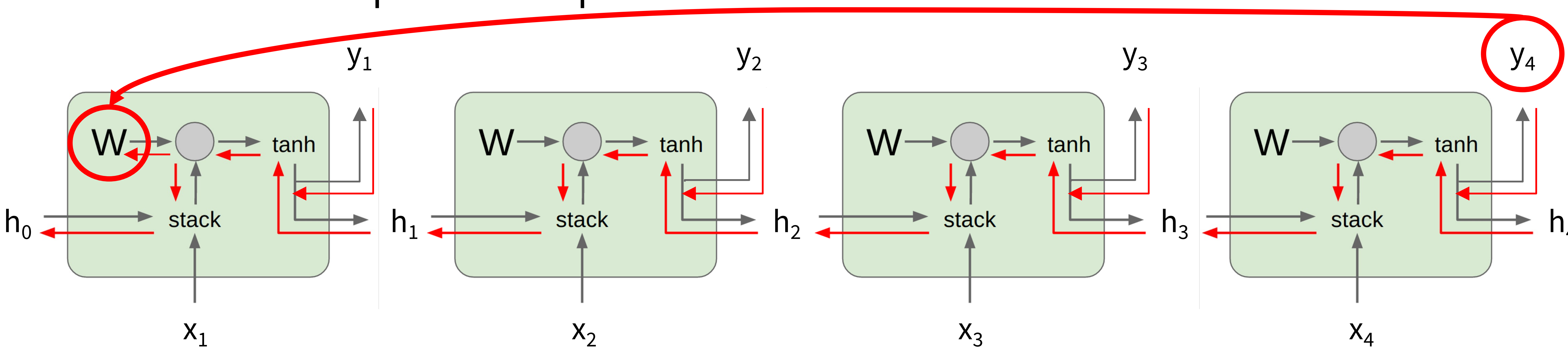
```
grad_norm = np.sum(grad * grad)
if grad_norm > threshold:
  grad *= (threshold / grad_norm)
```

Largest singular value < 1:
Vanishing gradients

Vanilla RNN Gradient Flow

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

Gradients over multiple time steps:



What if we assumed no non-linearity?

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

$$\frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial h_T} W_{hh}^{T-1} \frac{\partial h_1}{\partial W}$$

Largest singular value > 1:
Exploding gradients

Largest singular value < 1:
Vanishing gradients

→ Change RNN architecture

Long Short Term Memory (LSTM) – A Historical Note

Vanilla RNN

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

LSTM

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

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

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

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

Long Short Term Memory (LSTM)

Vanilla RNN

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

LSTM

Four gates

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

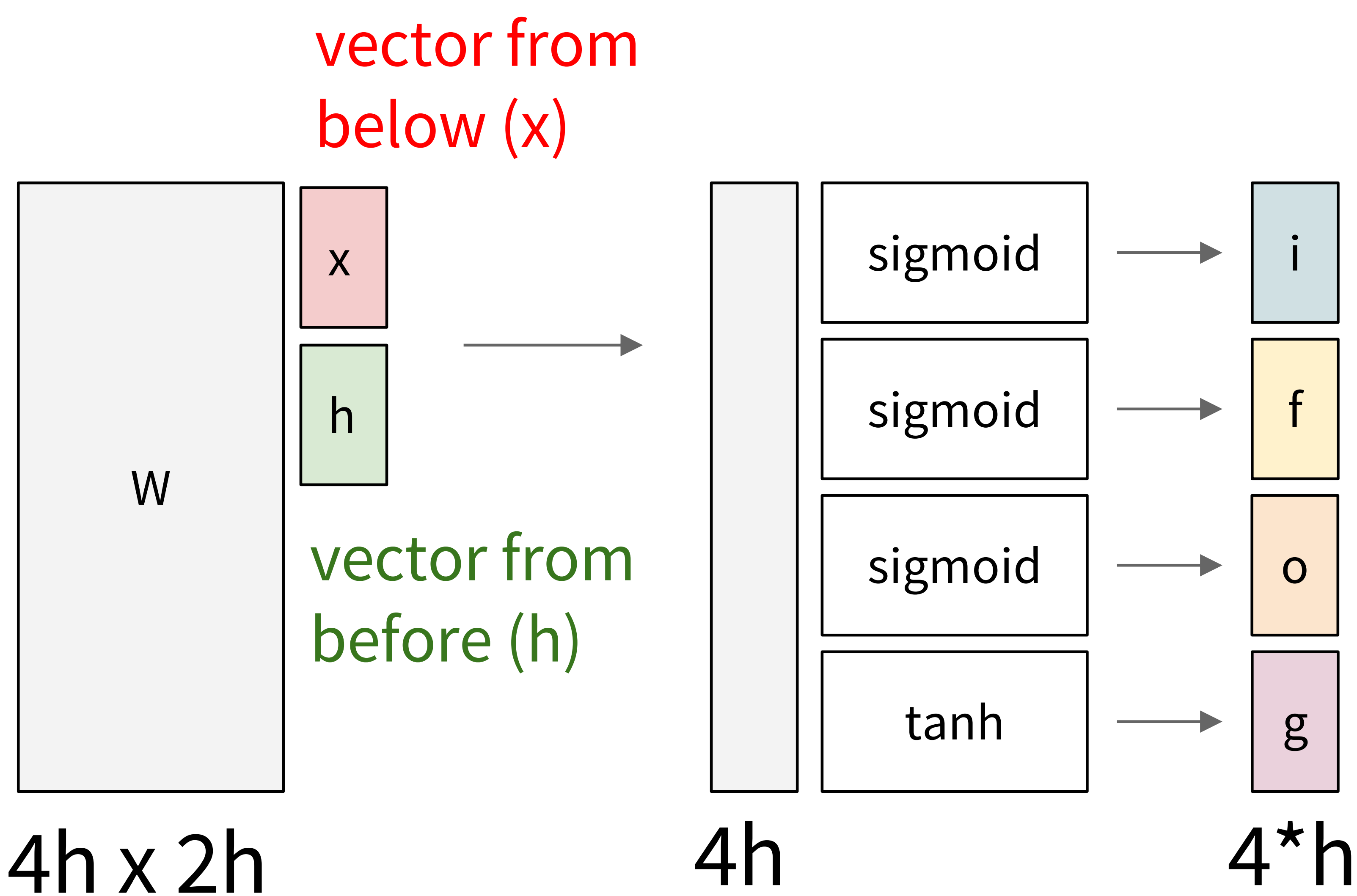
Cell state → $c_t = f \odot c_{t-1} + i \odot g$

Hidden state → $h_t = o \odot \tanh(c_t)$

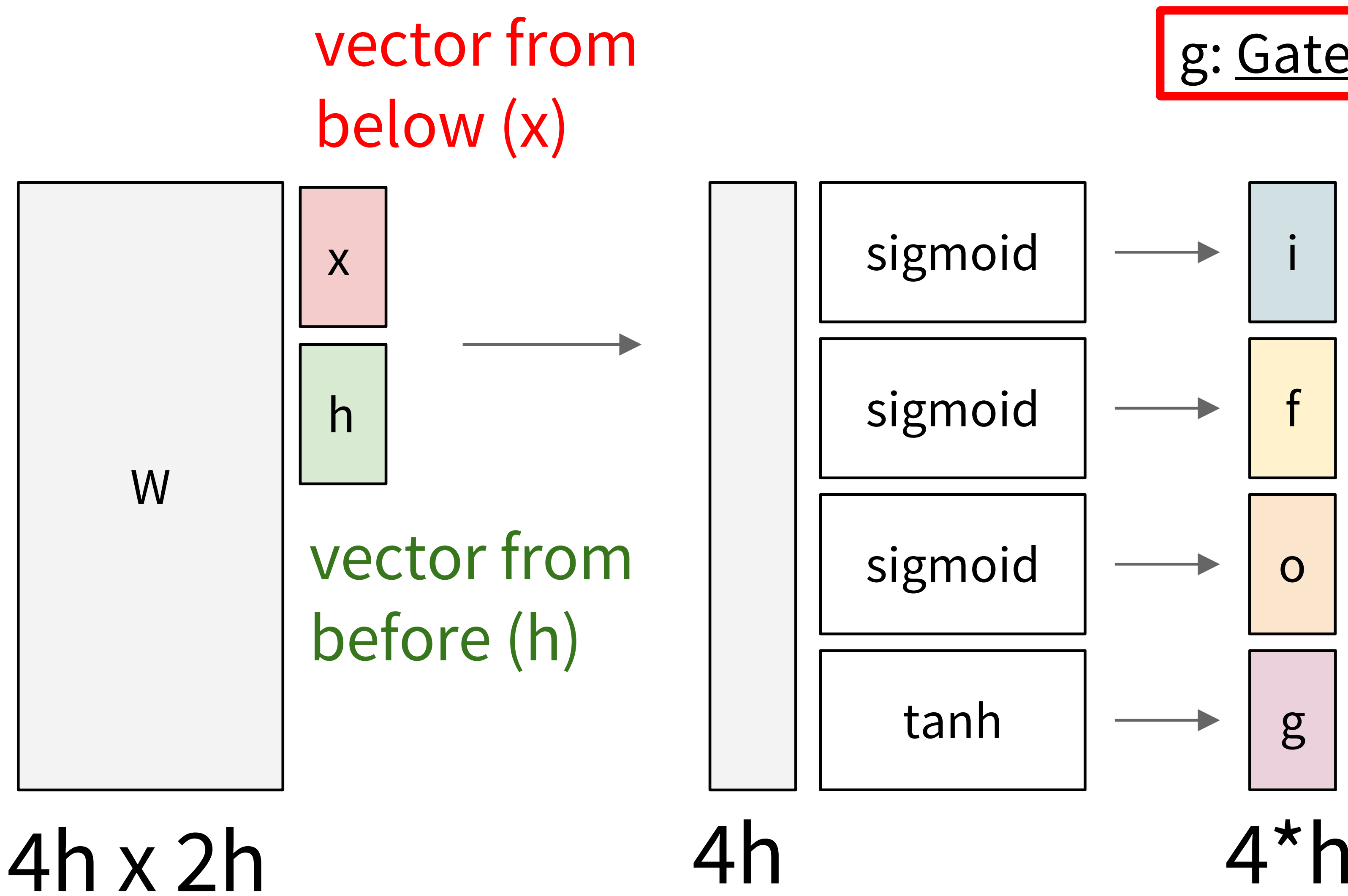
Hochreiter and Schmidhuber, "Long Short Term Memory", Neural Computation 1997

Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]



Long Short Term Memory (LSTM)
[Hochreiter et al., 1997]



g: Gate gate (?), How much to write to cell

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

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

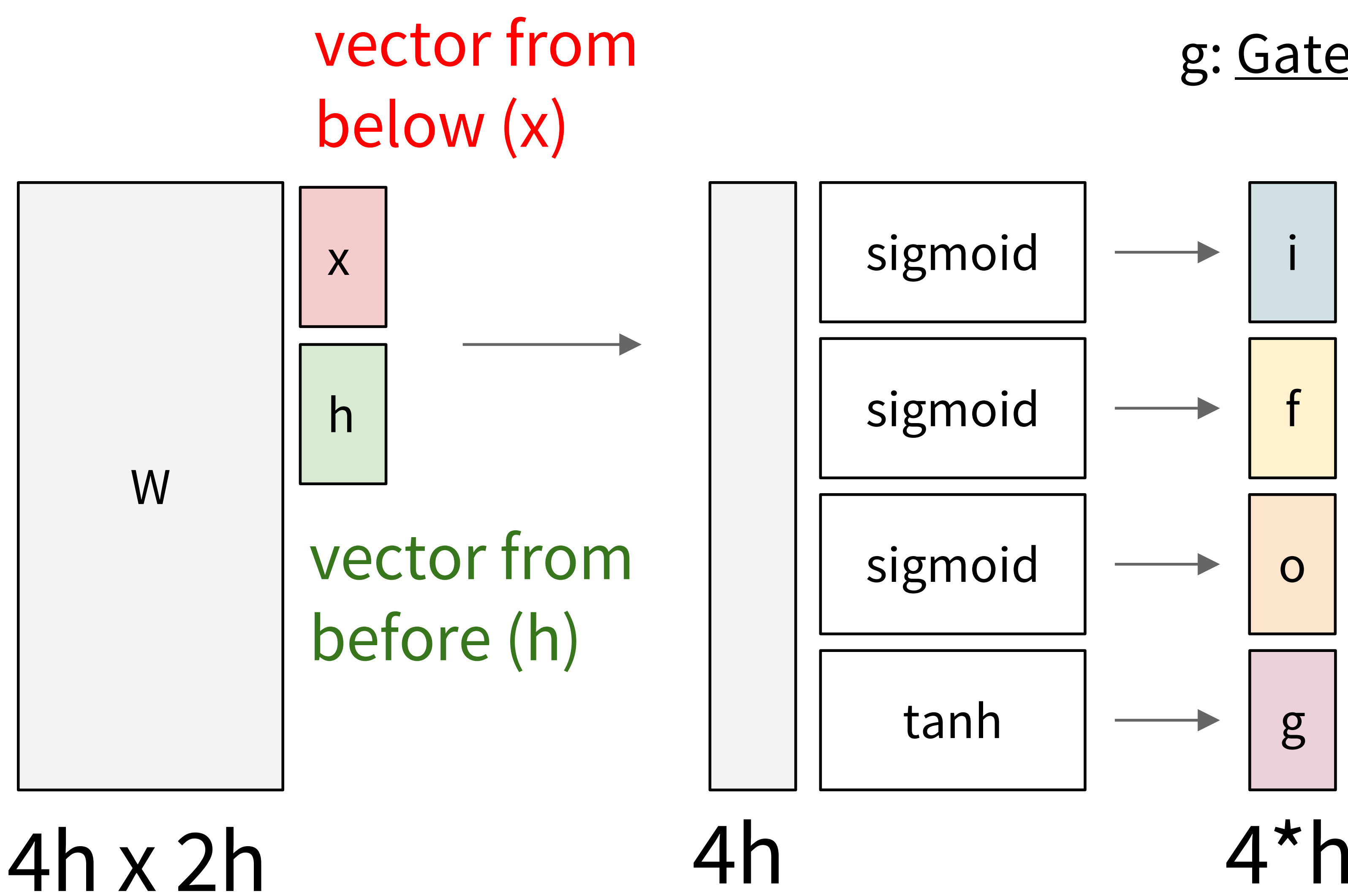
$$h_t = o \odot \tanh(c_t)$$

Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]

i: Input gate, whether to write to cell

g: Gate gate (?), How much to write to cell



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

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

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

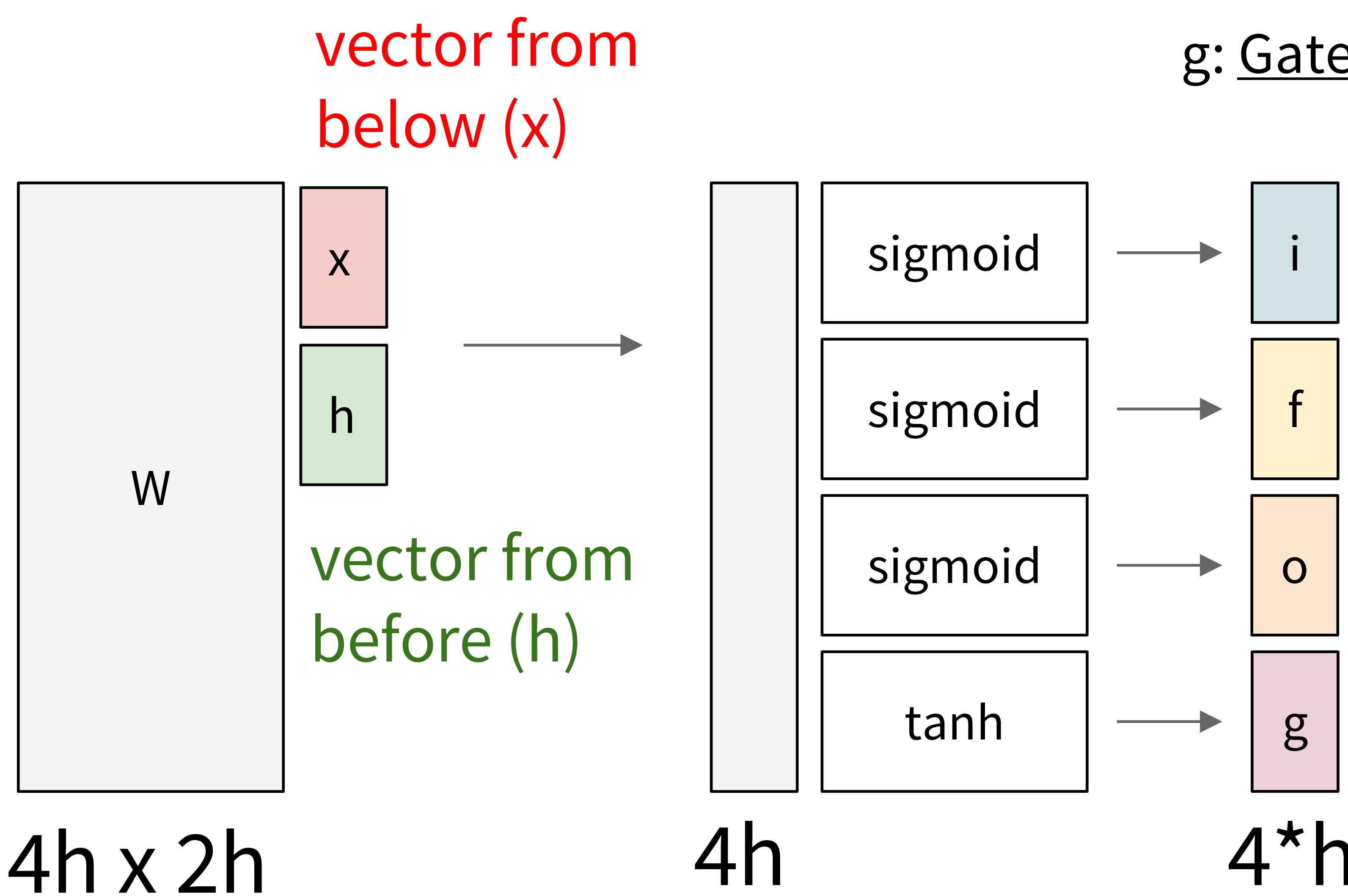
Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]

i: Input gate, whether to write to cell

f: Forget gate, Whether to erase cell

g: Gate gate (?), How much to write to cell



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

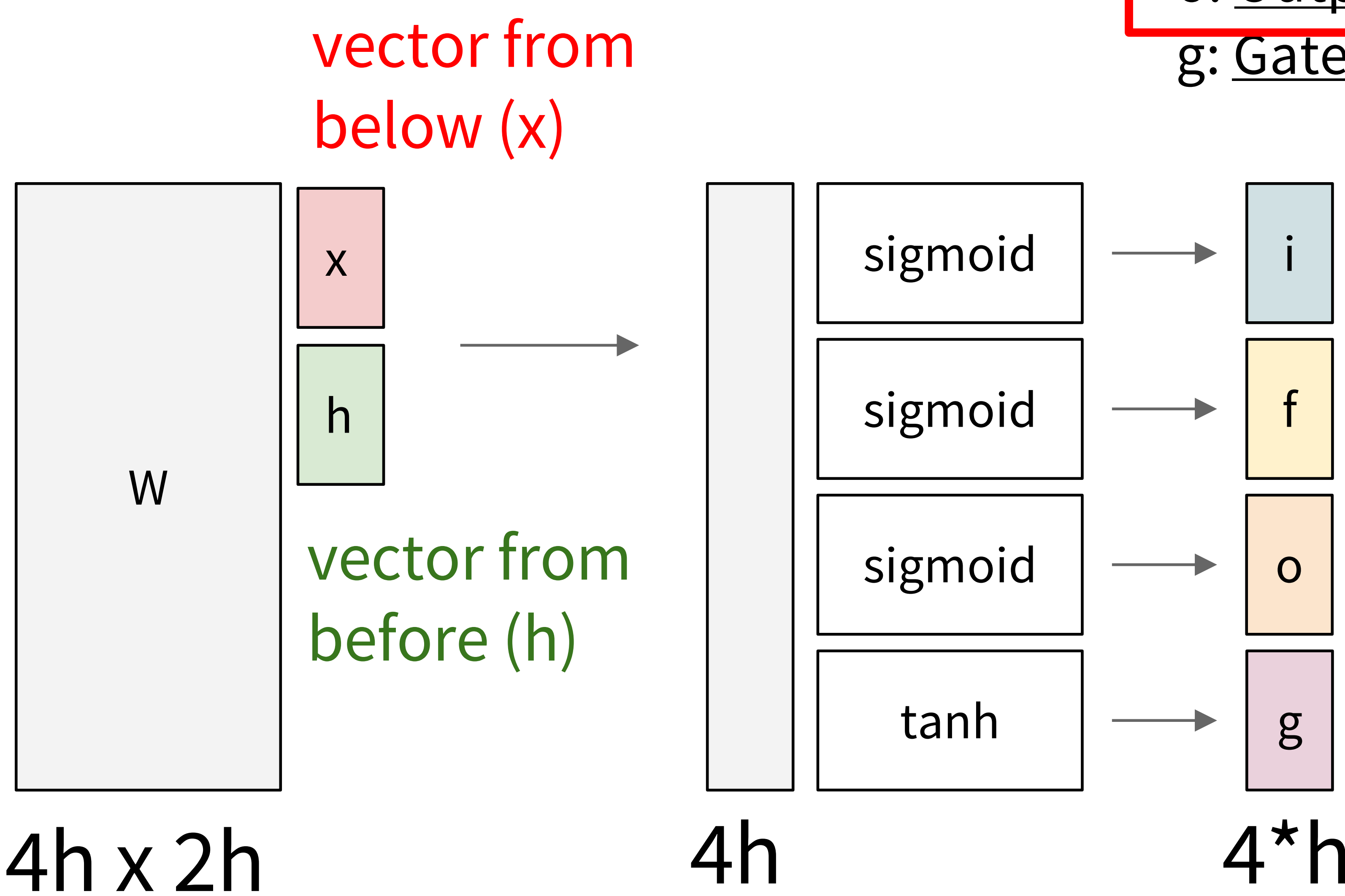
$$c_t = f \odot c_{t-1} + i \odot g$$

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

Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]

i: Input gate, whether to write to cell
f: Forget gate, Whether to erase cell
o: Output gate, How much to reveal cell
g: Gate gate (?), How much to write to cell



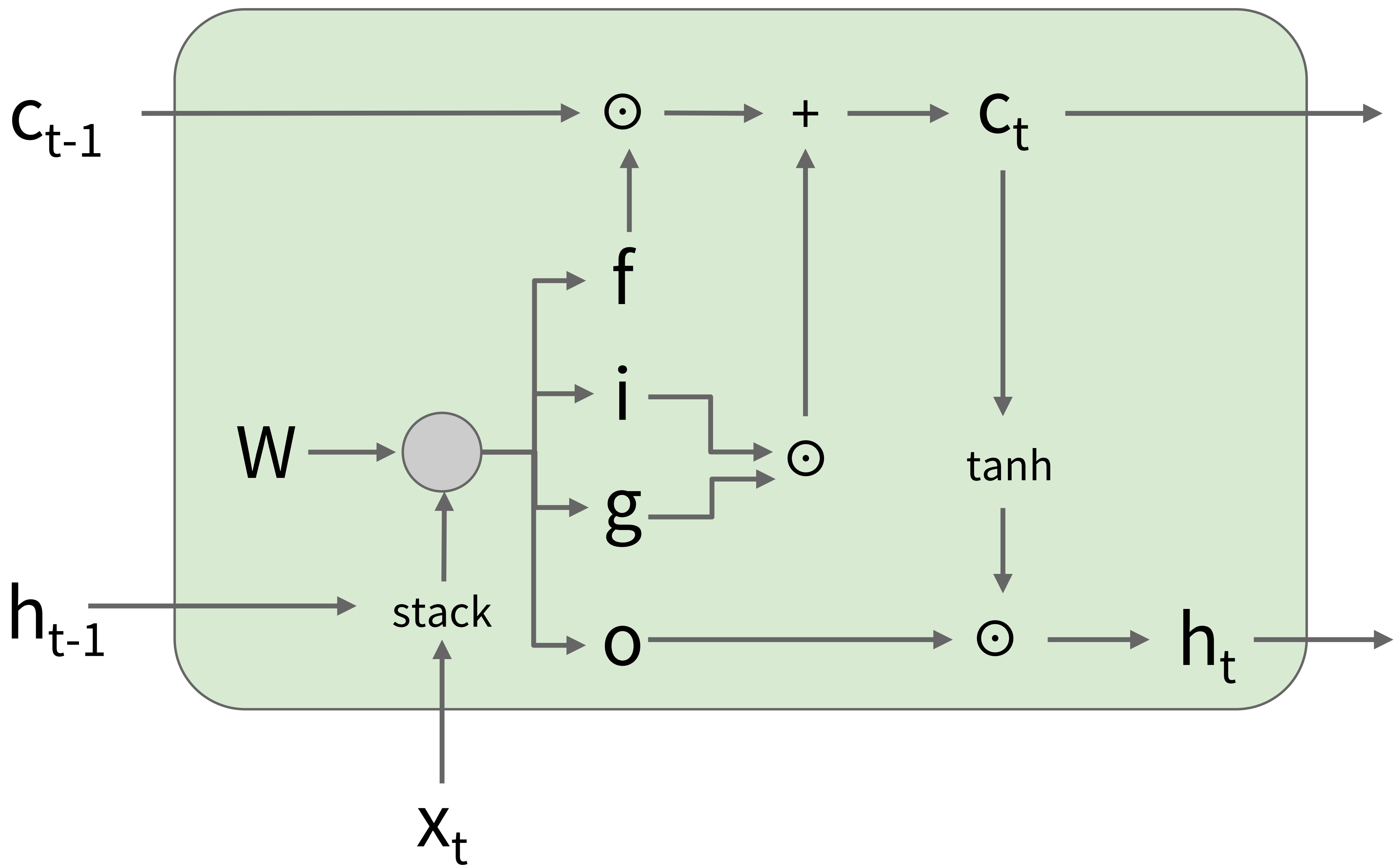
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

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

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

Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]

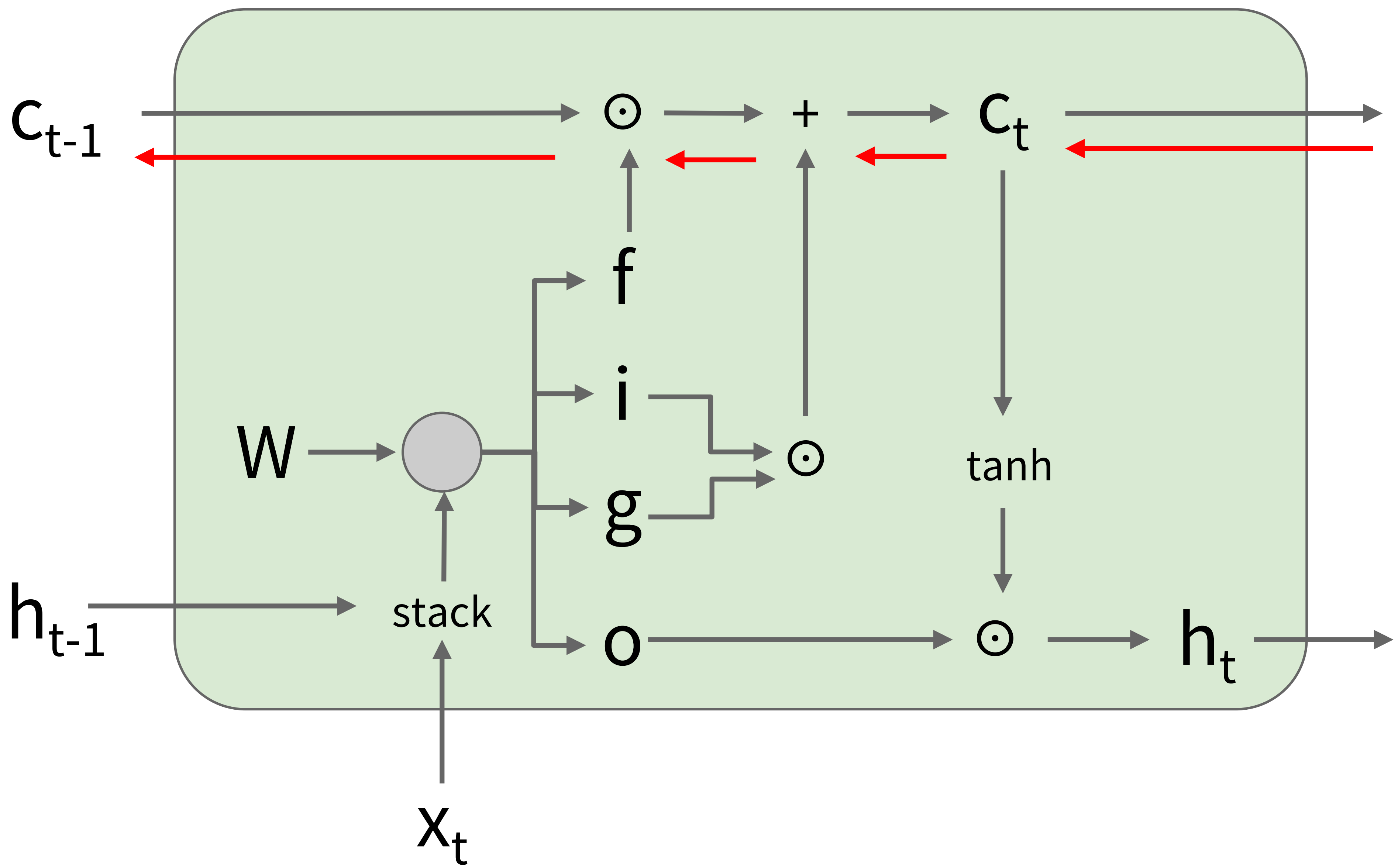


$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

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

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

Long Short Term Memory (LSTM): Gradient Flow
[Hochreiter et al., 1997]



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

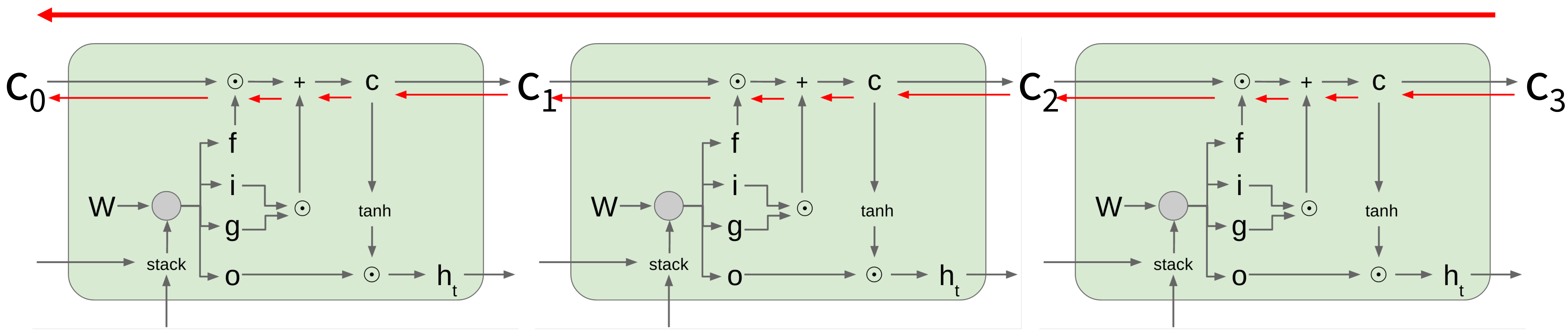
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

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

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

Long Short Term Memory (LSTM): Gradient Flow
[Hochreiter et al., 1997]

Uninterrupted gradient flow!



Do LSTMs solve the vanishing gradient problem?

The LSTM architecture makes it easier for the RNN to preserve information over many timesteps

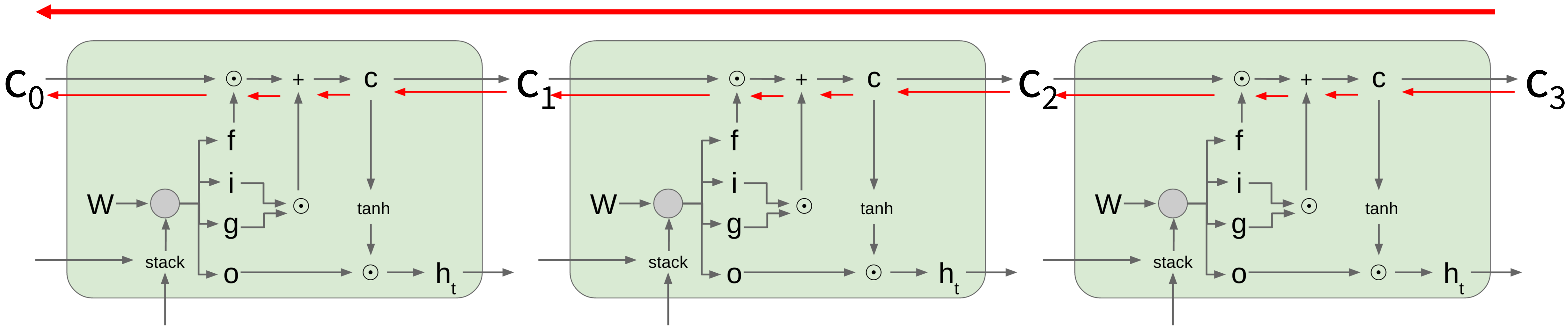
- e.g. if the f = 1 and the i = 0, then the information of that cell is preserved indefinitely.
- By contrast, it's harder for vanilla RNN to learn a recurrent weight matrix Wh that preserves info in hidden state

LSTM doesn't guarantee that there is no vanishing/exploding gradient, but it does provide an easier way for the model to learn long-distance dependencies

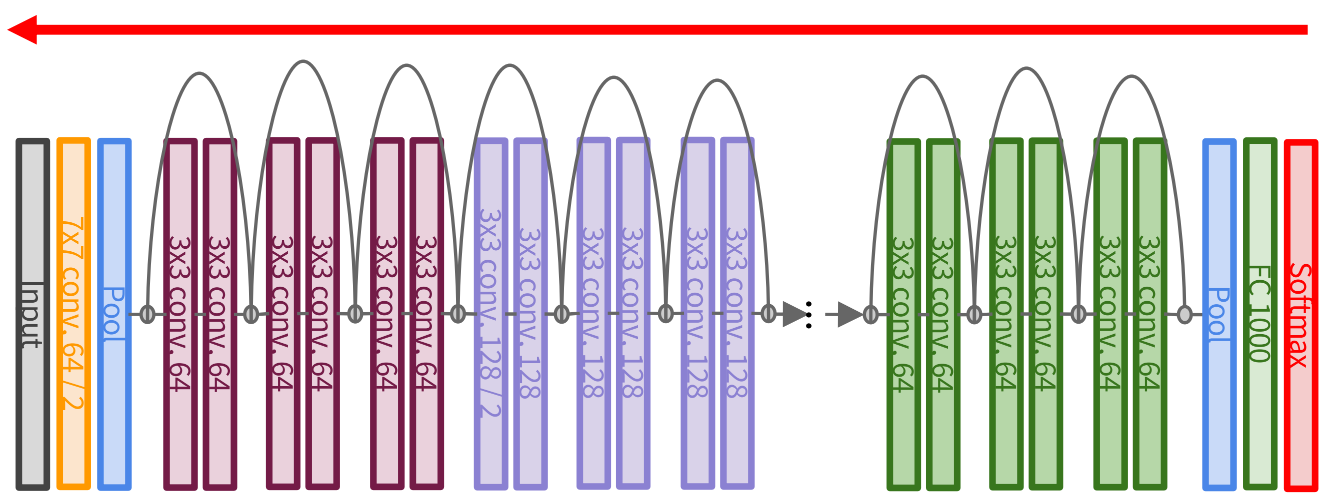
Long Short Term Memory (LSTM): Gradient Flow

[Hochreiter et al., 1997]

Uninterrupted gradient flow!



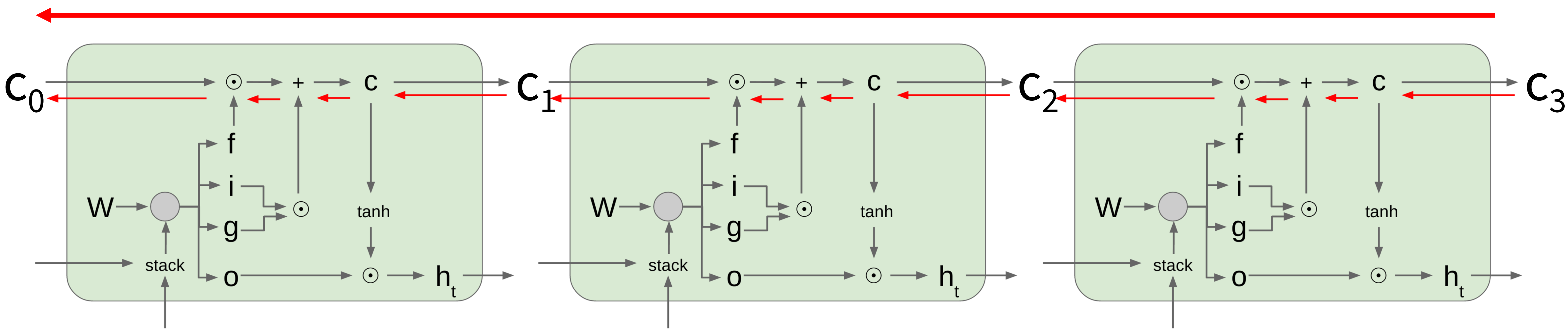
Similar to ResNet!



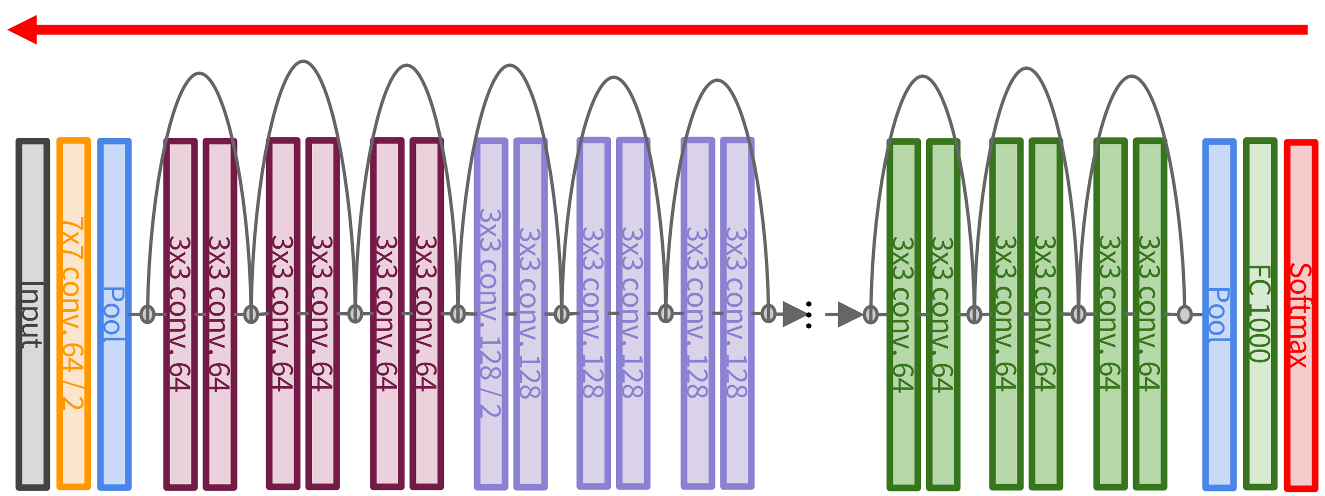
Long Short Term Memory (LSTM): Gradient Flow

[Hochreiter et al., 1997]

Uninterrupted gradient flow!



Similar to ResNet!



In between:
Highway Networks

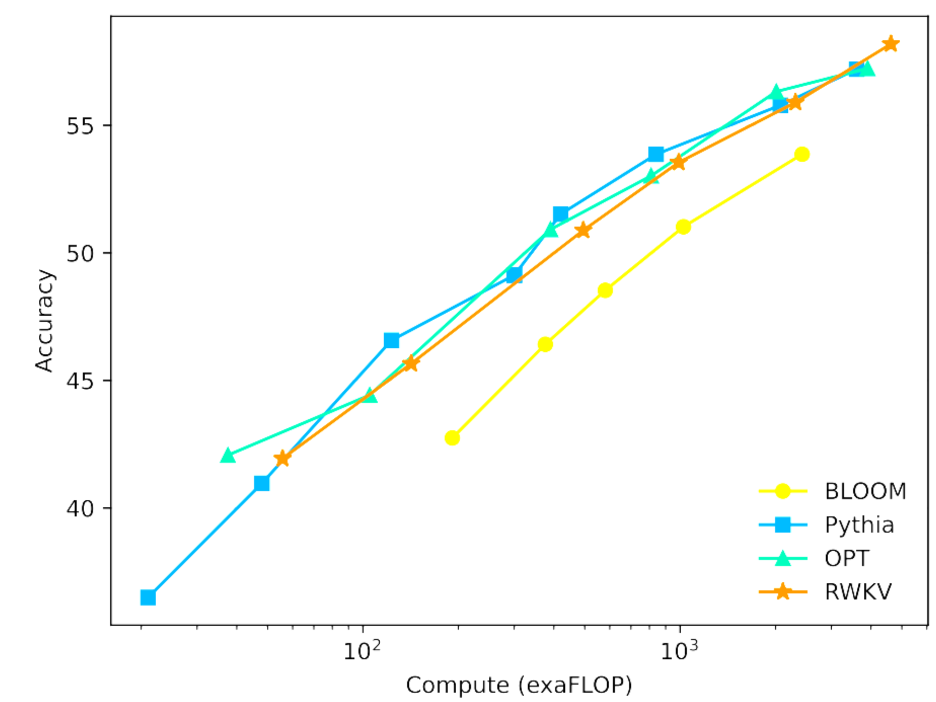
$$g = T(x, W_T)$$

$$y = g \odot H(x, W_H) + (1 - g) \odot x$$

Srivastava et al, "Highway Networks", ICML DL Workshop 2015

Modern RNNs

- Sometimes called “state space models”
  - Hidden state
- Main advantages:
  - Unlimited context length
  - Compute scales linearly with sequence length



RWKV Scaling (arXiv)

SIMPLIFIED STATE SPACE LAYERS FOR SEQUENCE MODELING

Mamba: Linear-Time Sequence Modeling with Selective State Spaces

Summary

- RNNs allow a lot of flexibility in architecture design
- Vanilla RNNs are simple but don't work very well
- More complex variants (e.g. LSTMs, Mamba) can introduce ways to selectively pass information forward
- Backward flow of gradients in RNN can explode or vanish. Exploding is controlled with gradient clipping. Backpropagation through time is often needed.
- Better/simpler architectures are a hot topic of current research, as well as new paradigms for reasoning over sequences

Next time: Attention and Transformers