

Lecture 16: Recurrent Neural Networks

Admin: A4

A4 is finally released!

Will be due Tuesday 3/29, 11:59pm ET

Admin: Midterm Grades

Grading is nearly complete, should be released by tonight

Regrade requests: Submit a private piazza post by
Wednesday 3/23 (1 week from today)

Admin: Project

Will write up more guidelines this week, but rough sketch: Pick one of the following:

- Collect your own classification dataset, apply transfer learning
- Single-Image Super-Resolution
- Neural Radiance Fields (NeRF) for novel view synthesis
- Self-Supervised Learning (*maybe, not sure)
- Suggest your own

You get ~1 page of instructions for each with pointers to key papers, and instructions for what key results we want to see. No starter code. You implement and turn in a Colab / Jupyter notebook (with supporting code) that implements the model and walks through the key deliverables, similar to the homework notebooks.

For suggest your own project, you need to provide us with a similar one-page plan for what you will implement and we need to approve the project.

Last Time: Localization Tasks

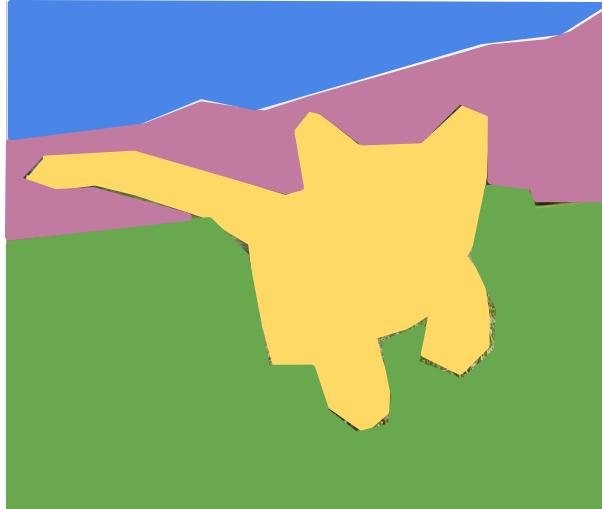
Classification



CAT

No spatial extent

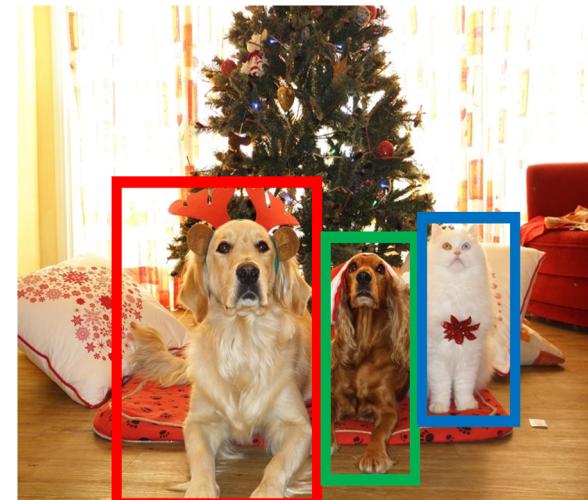
Semantic Segmentation



GRASS, CAT, TREE,
SKY

No objects, just pixels

Object Detection



DOG, DOG, CAT

Multiple Objects

Instance Segmentation

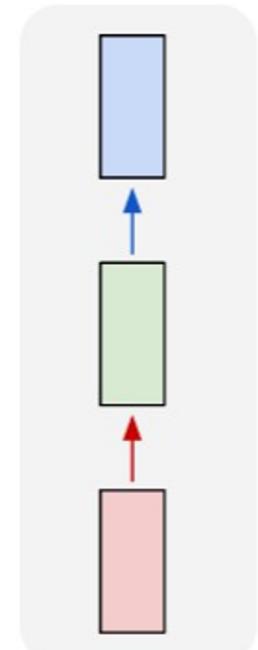


DOG, DOG, CAT

[This image is CC0 public domain](#)

So far: “Feedforward” Neural Networks

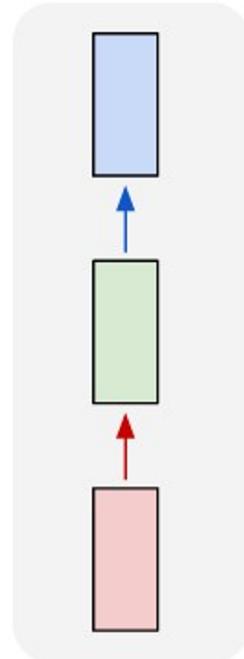
one to one



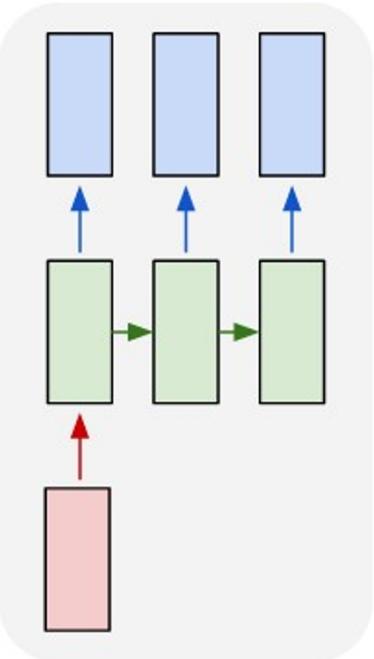
e.g. **Image classification**
Image -> Label

Recurrent Neural Networks: Process Sequences

one to one



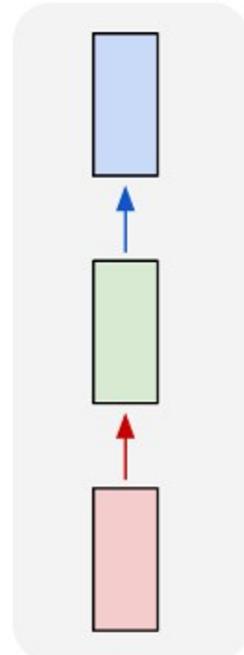
one to many



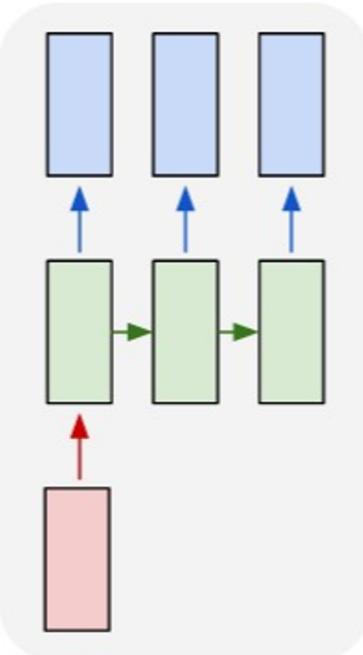
e.g. **Image Captioning:**
Image -> sequence of words

Recurrent Neural Networks: Process Sequences

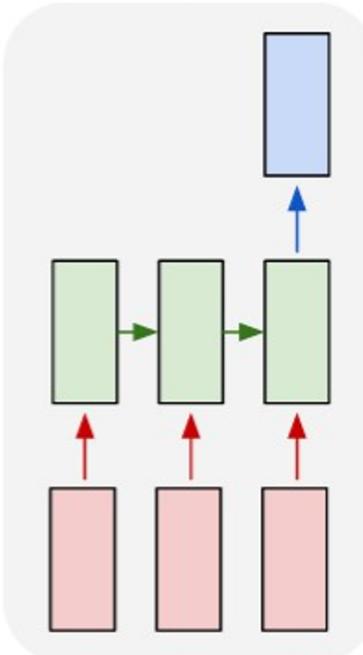
one to one



one to many



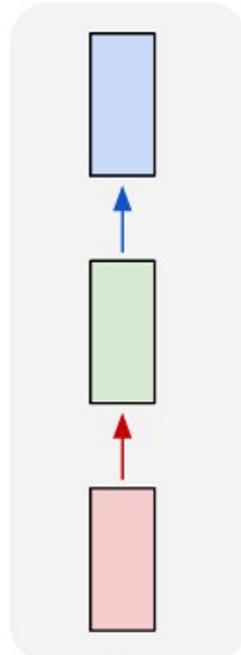
many to one



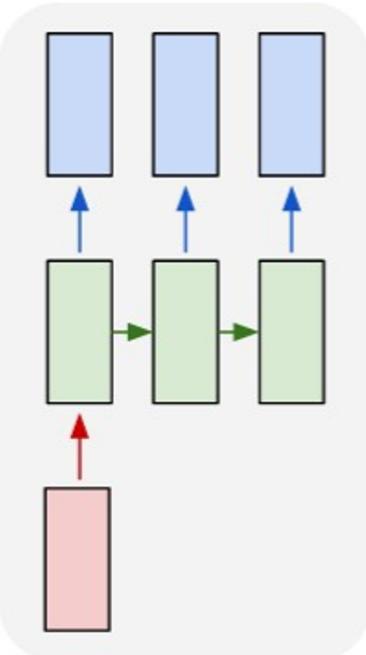
e.g. **Video classification:**
Sequence of images -> label

Recurrent Neural Networks: Process Sequences

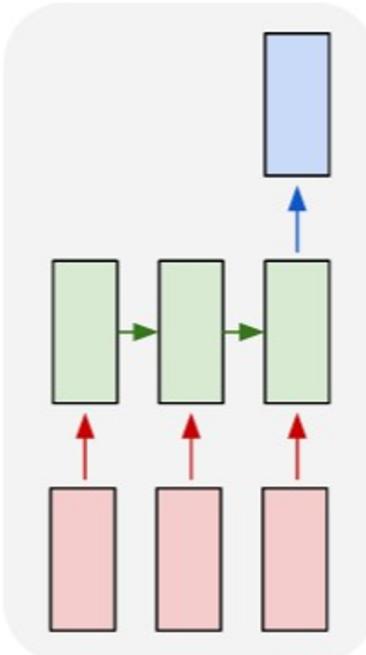
one to one



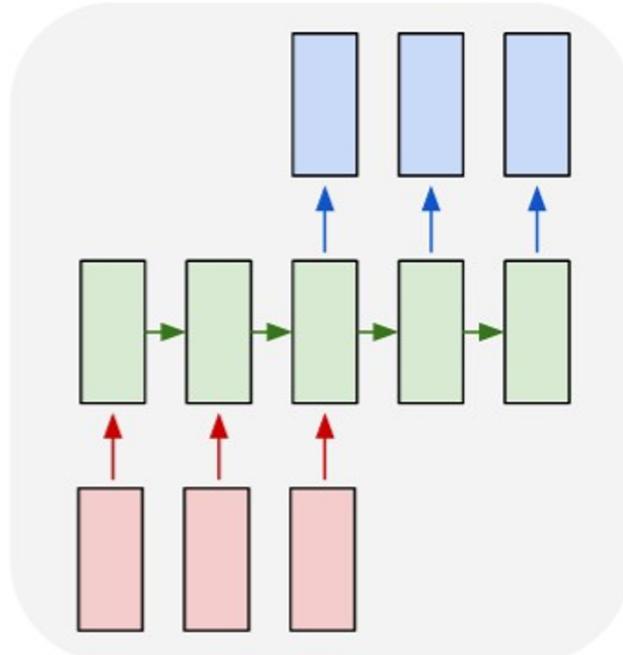
one to many



many to one



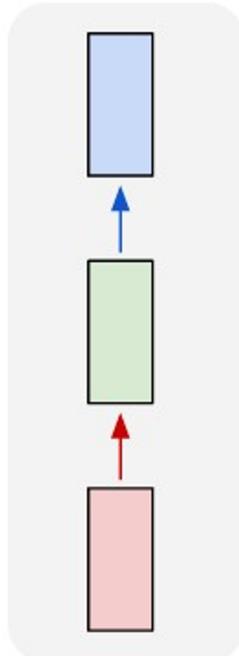
many to many



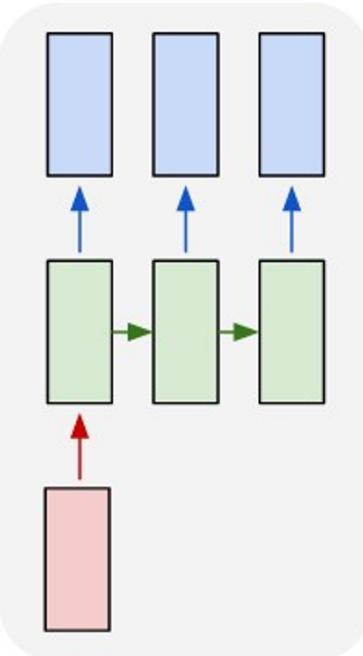
e.g. **Machine Translation:**
Sequence of words -> Sequence of words

Recurrent Neural Networks: Process Sequences

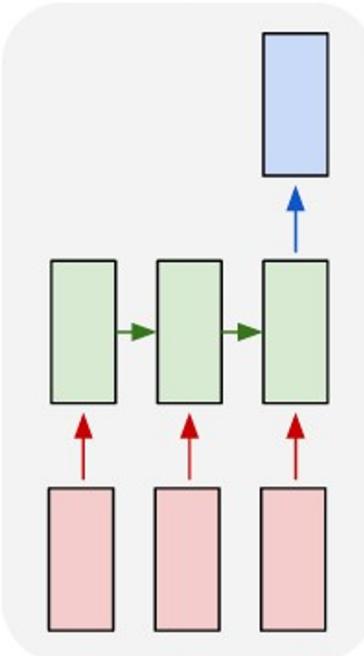
one to one



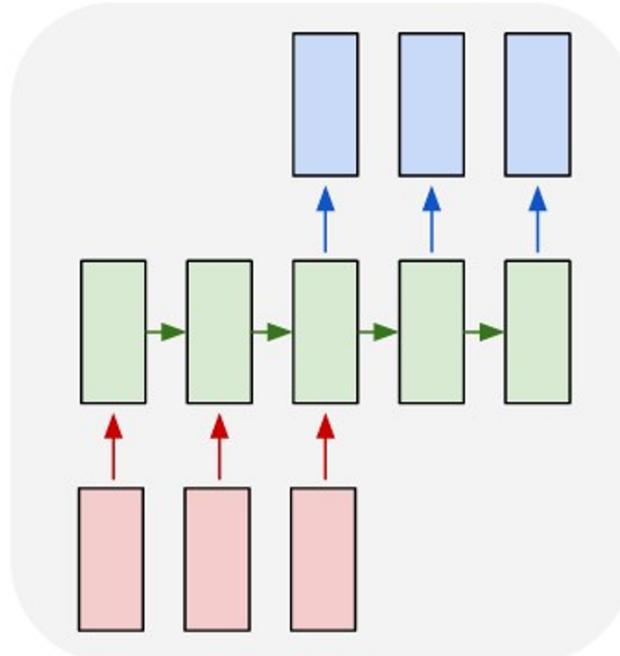
one to many



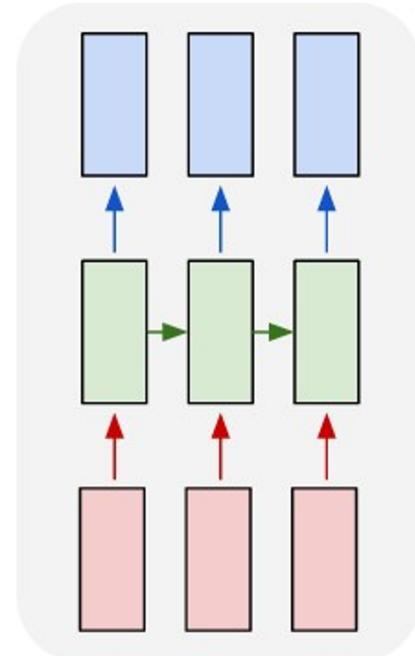
many to one



many to many



many to many



e.g. **Per-frame video classification:**
Sequence of images -> Sequence of labels

Sequential Processing of Non-Sequential Data

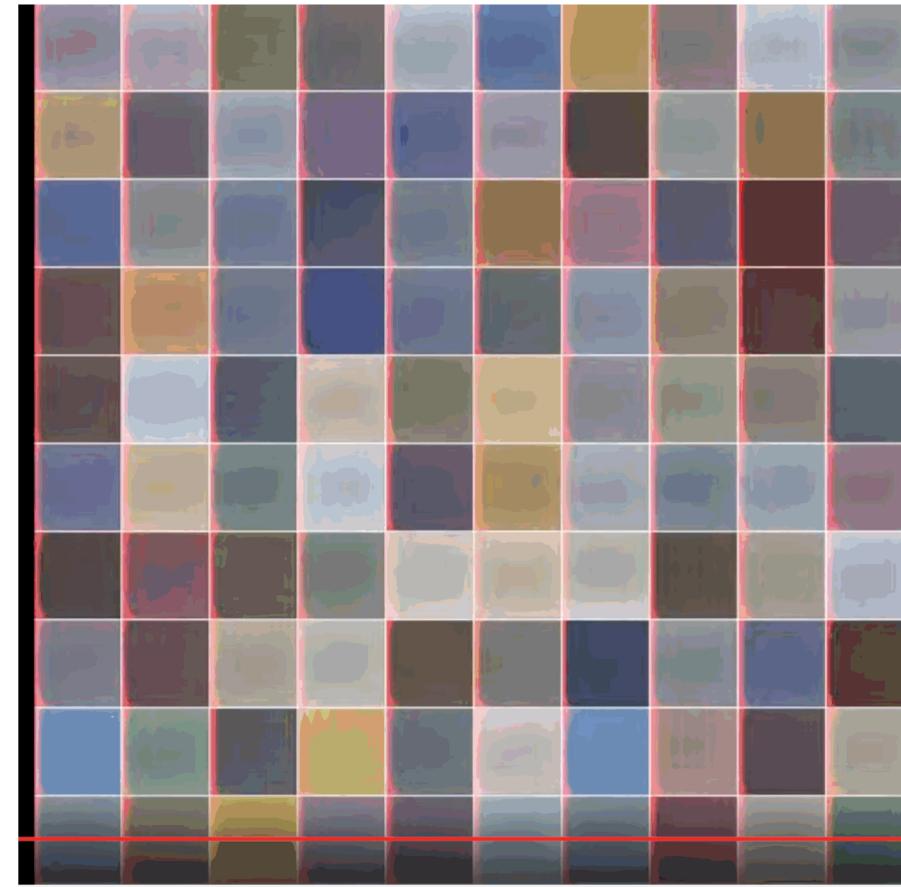
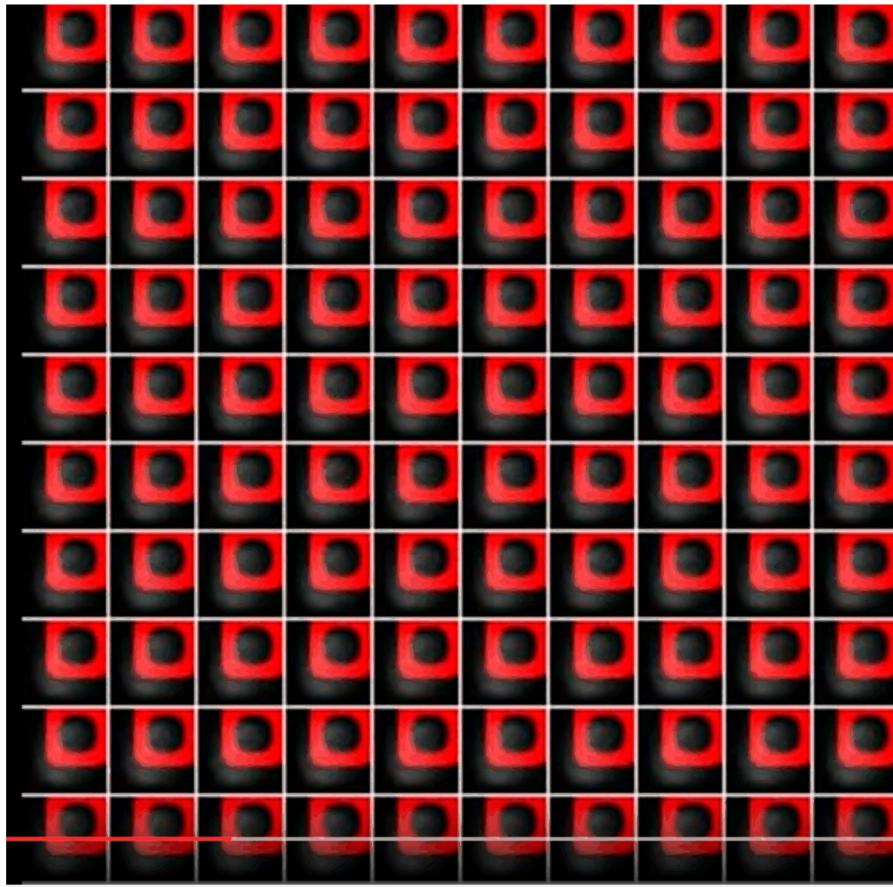
Classify images by taking
a series of “glimpses”



Ba, Mnih, and Kavukcuoglu, “Multiple Object Recognition with Visual Attention”, ICLR 2015.
Gregor et al, “DRAW: A Recurrent Neural Network For Image Generation”, ICML 2015

Sequential Processing of Non-Sequential Data

Generate images one piece at a time!



Sequential Processing of Non-Sequential Data

Integrate with oil
paint simulator – at
each timestep output
a new stroke

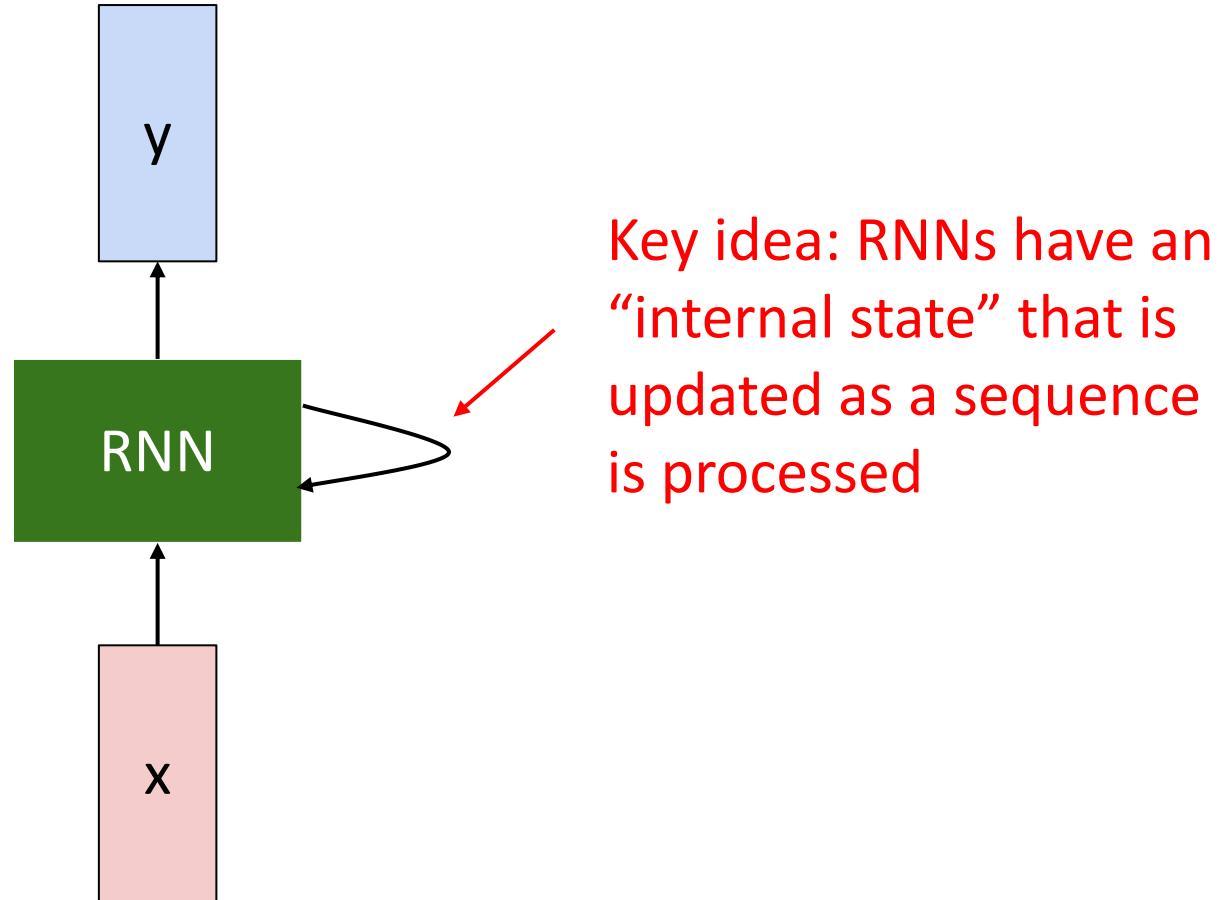


Ganin et al, "Synthesizing Programs for Images using Reinforced Adversarial Learning", ICML 2018

https://twitter.com/yaroslav_ganin/status/1180120687131926528

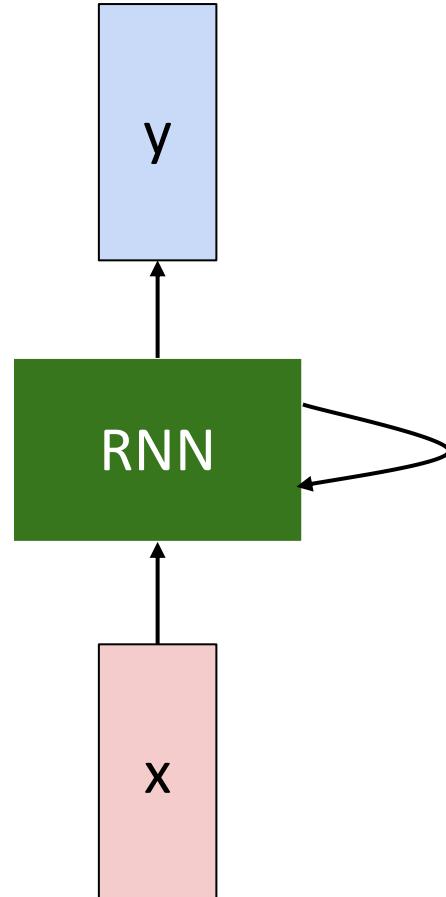
Reproduced with permission

Recurrent Neural Networks



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

Recurrent Neural Networks



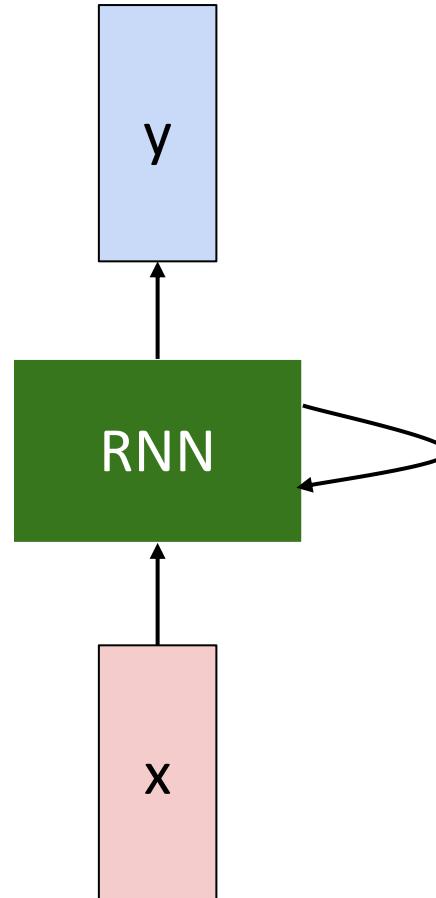
We can process a sequence of vectors \mathbf{x} by applying a **recurrence formula** at every time step:

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

new state old state input vector at
some function with parameters W some time step

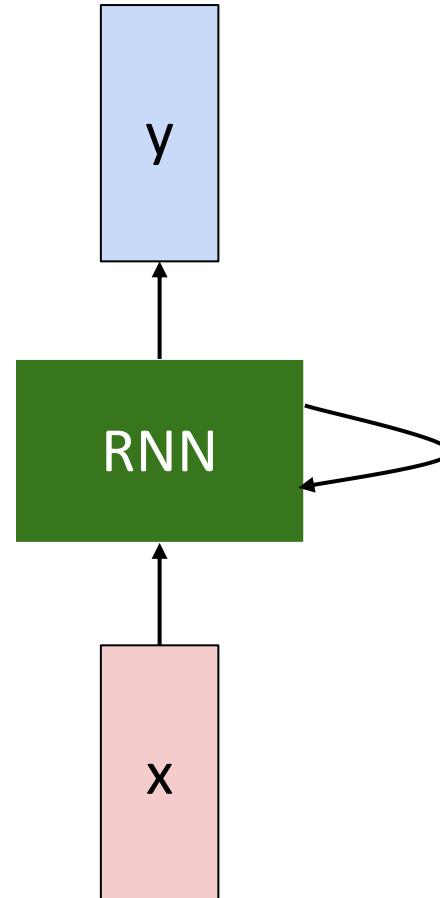
Recurrent Neural Networks

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



We can process a sequence of vectors \mathbf{x} by applying a **recurrence formula** at every time step:

(Vanilla) Recurrent Neural Networks



The state consists of a single “*hidden*” vector \mathbf{h} :

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



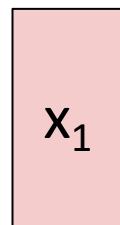
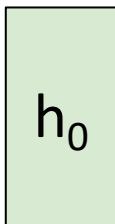
$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t + b_h)$$

$$y_t = W_{hy}h_t + b_y$$

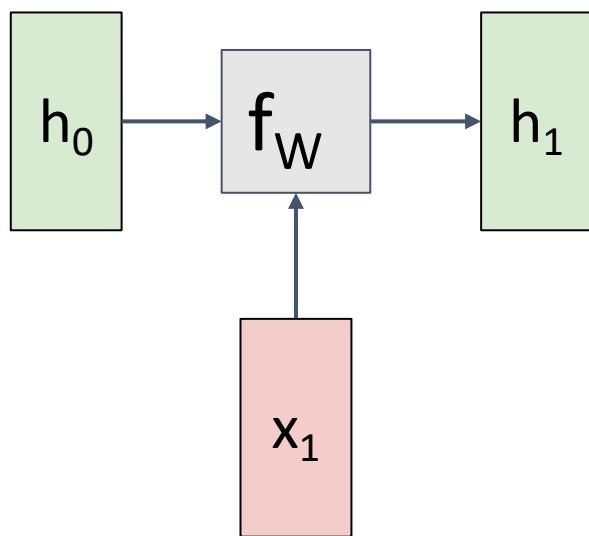
Sometimes called a “Vanilla RNN” or an
“Elman RNN” after Prof. Jeffrey Elman

RNN Computational Graph

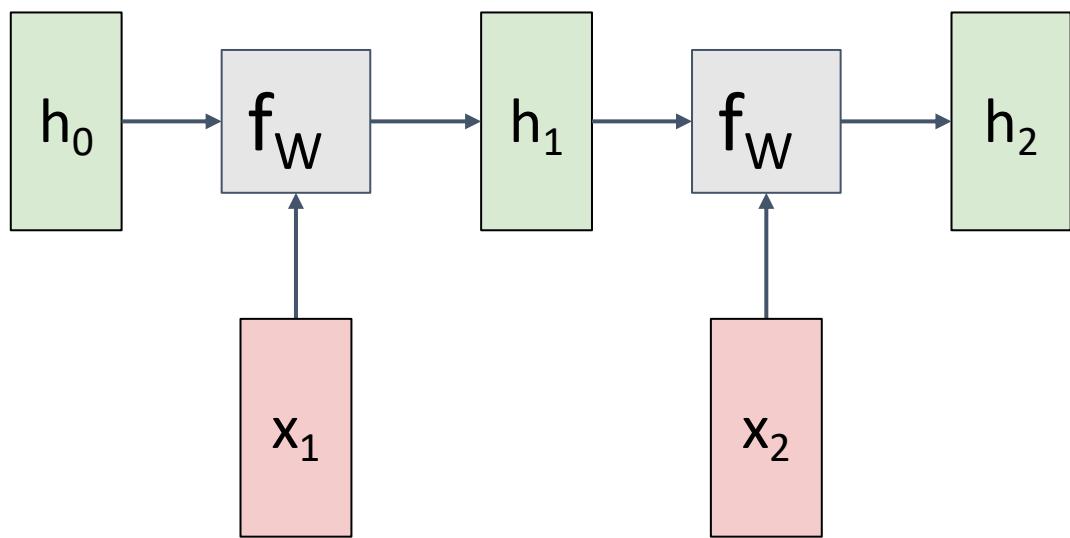
Initial hidden state
Either set to all 0,
Or learn it



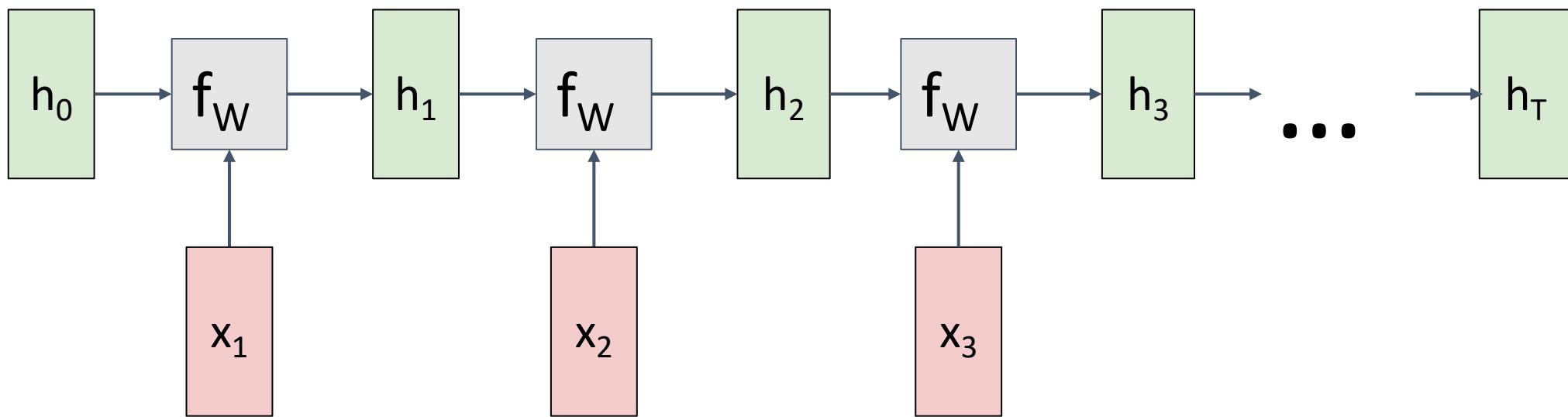
RNN Computational Graph



RNN Computational Graph

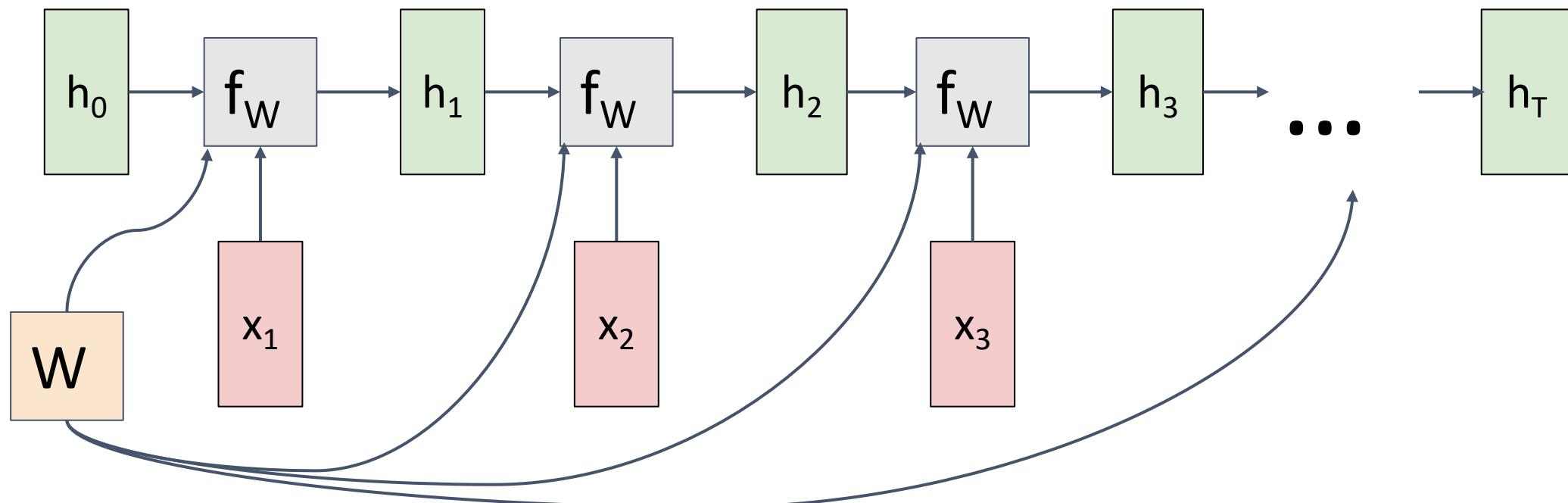


RNN Computational Graph

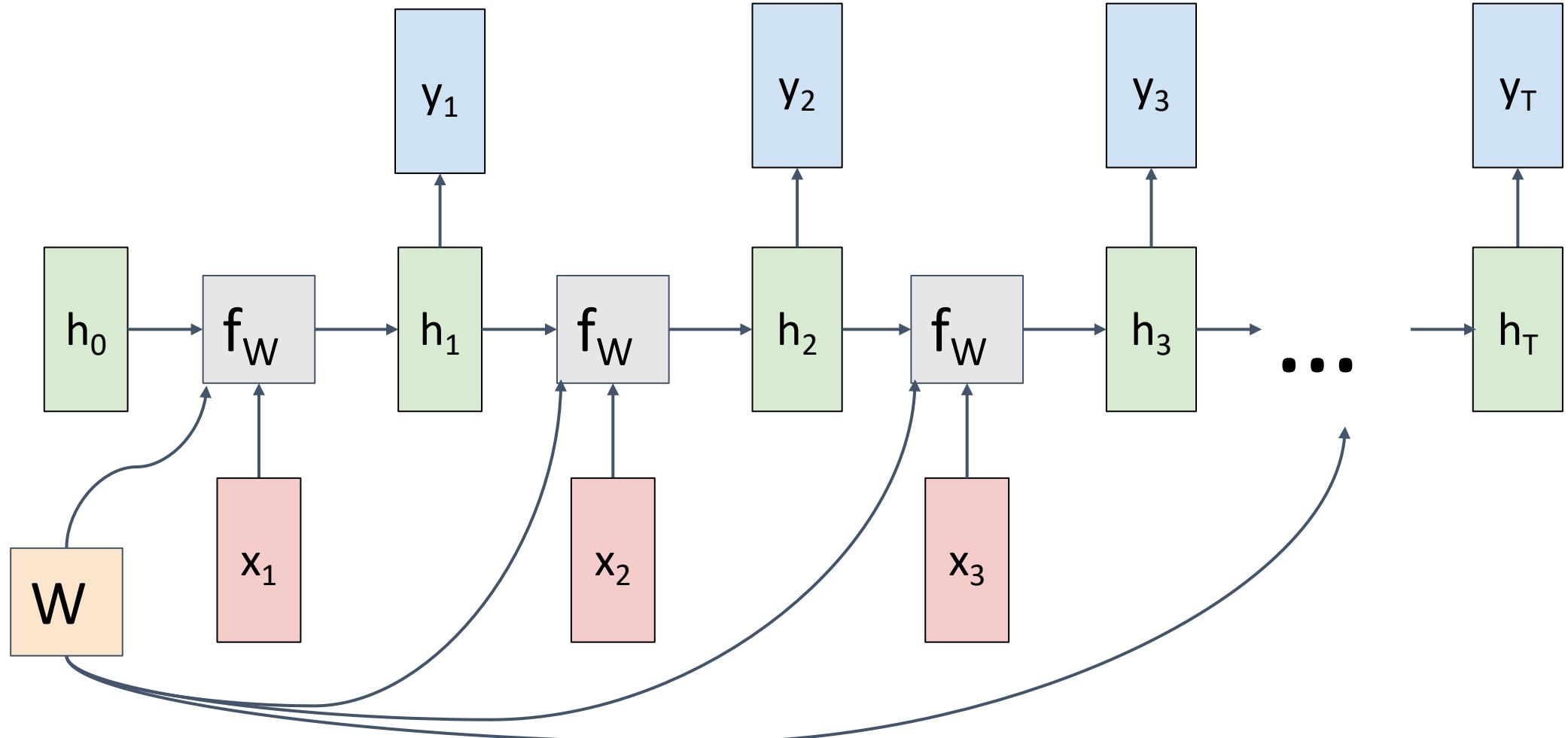


RNN Computational Graph

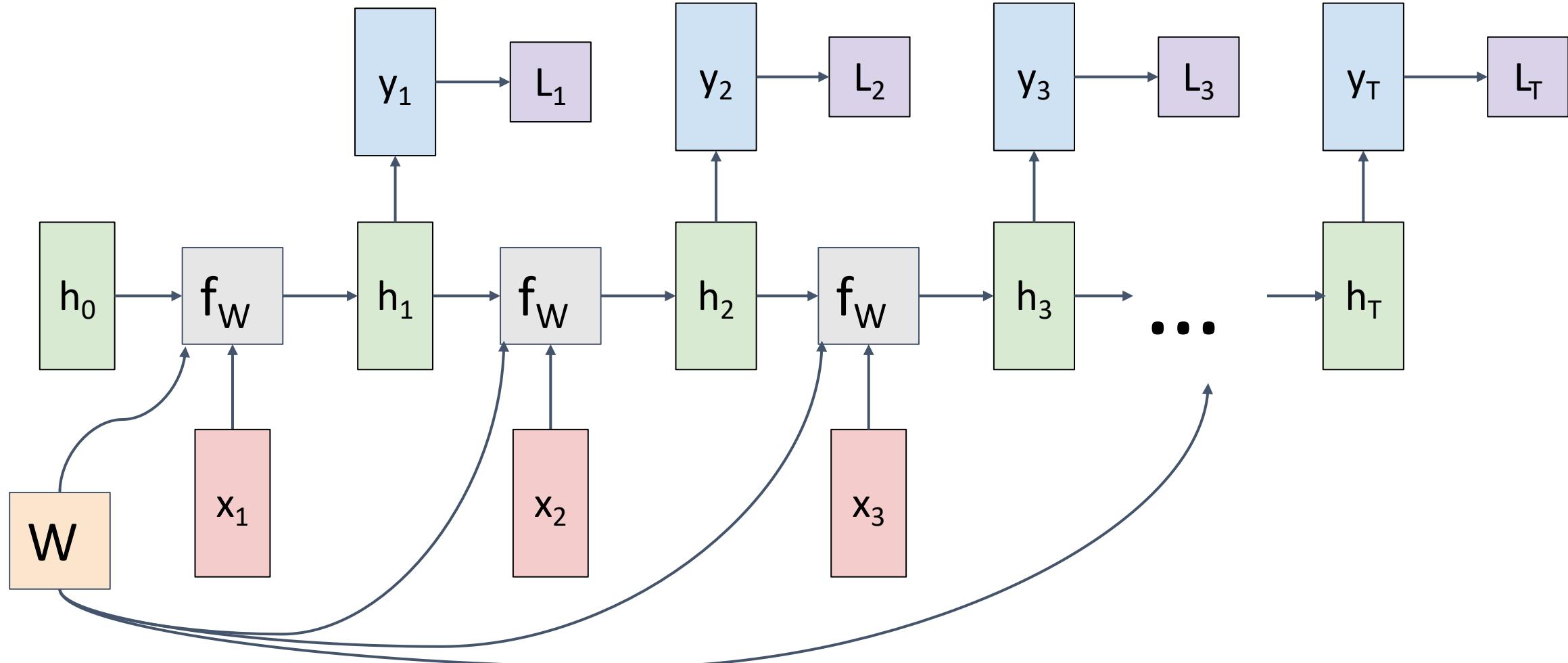
Re-use the same weight matrix at every time-step



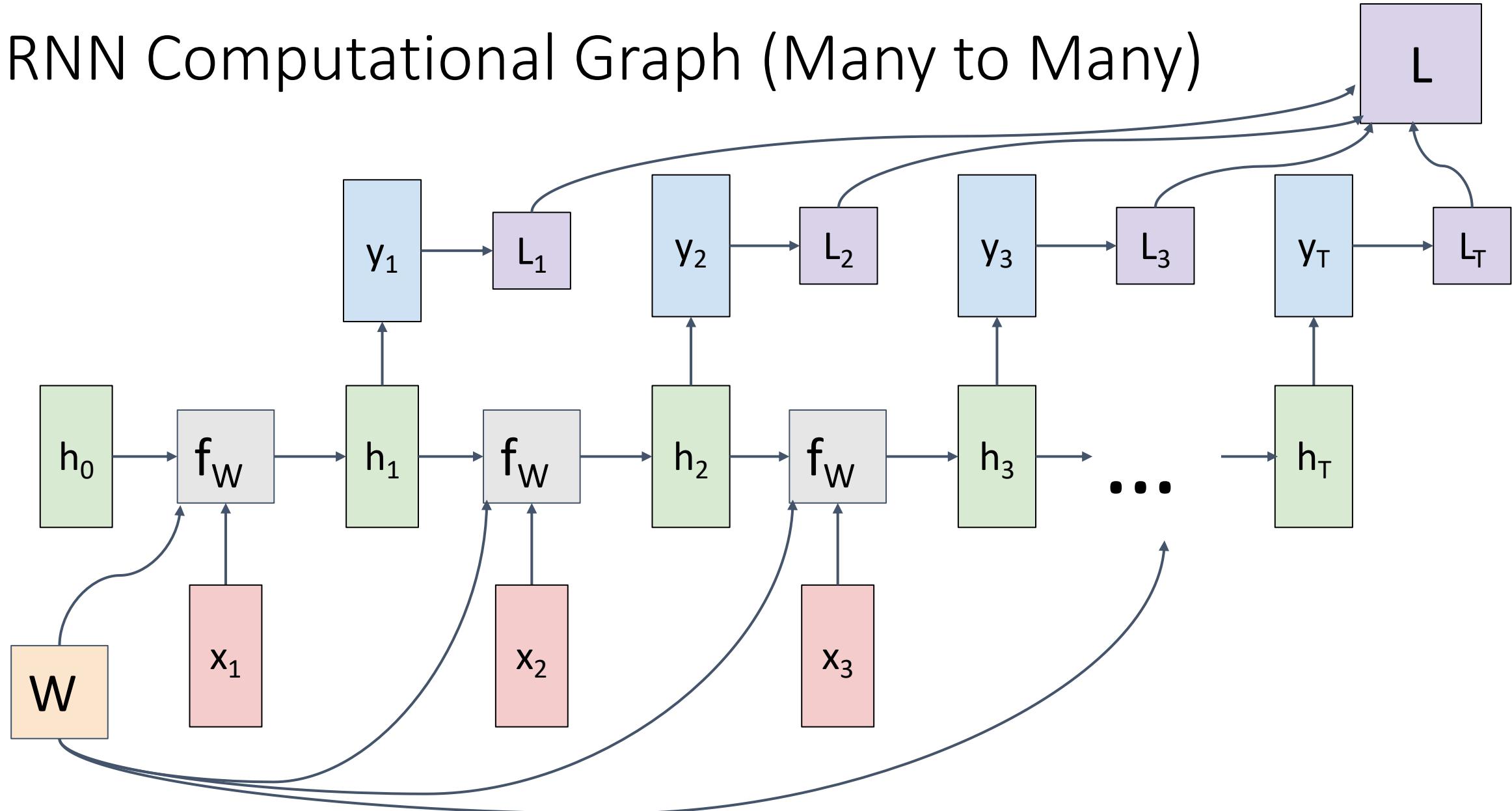
RNN Computational Graph (Many to Many)



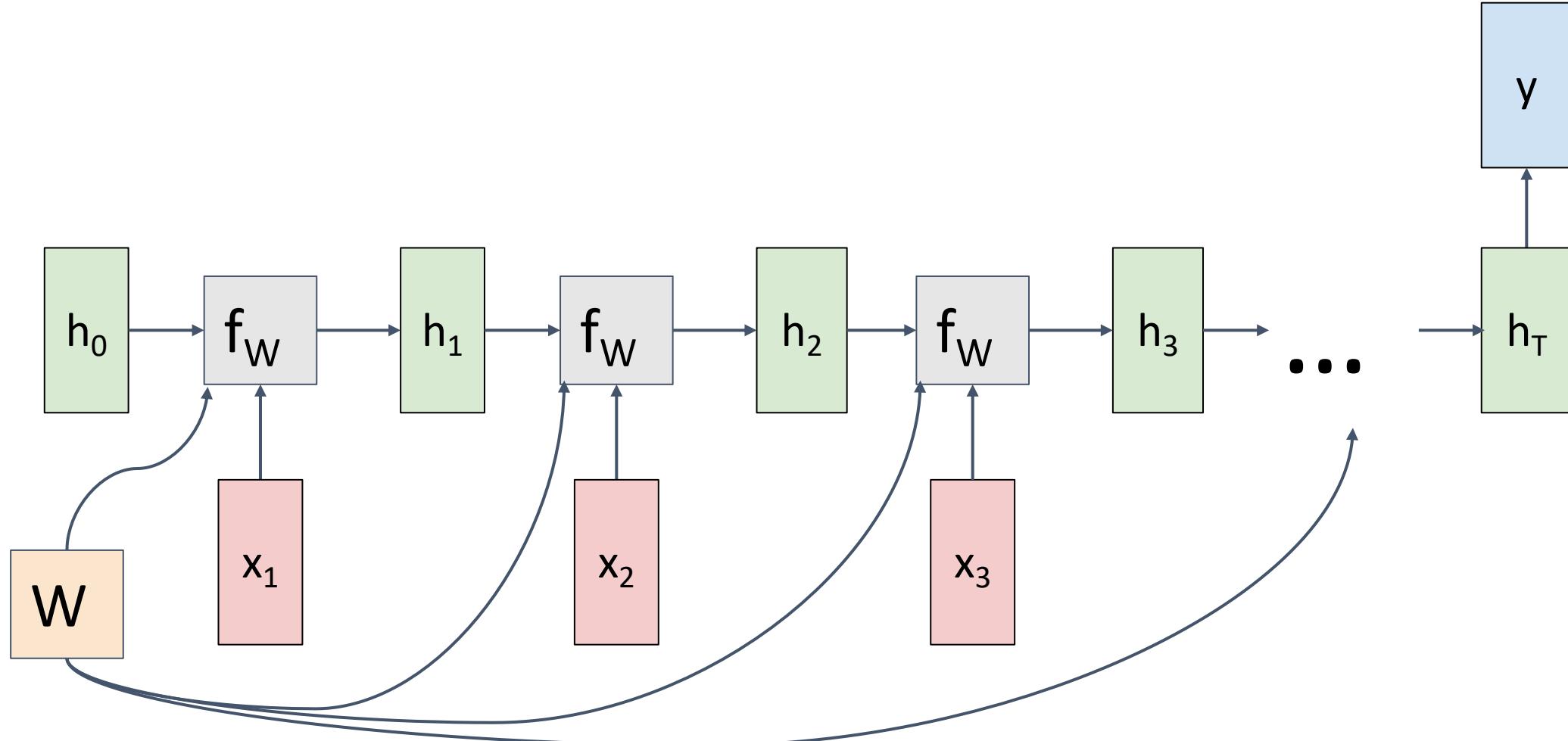
RNN Computational Graph (Many to Many)



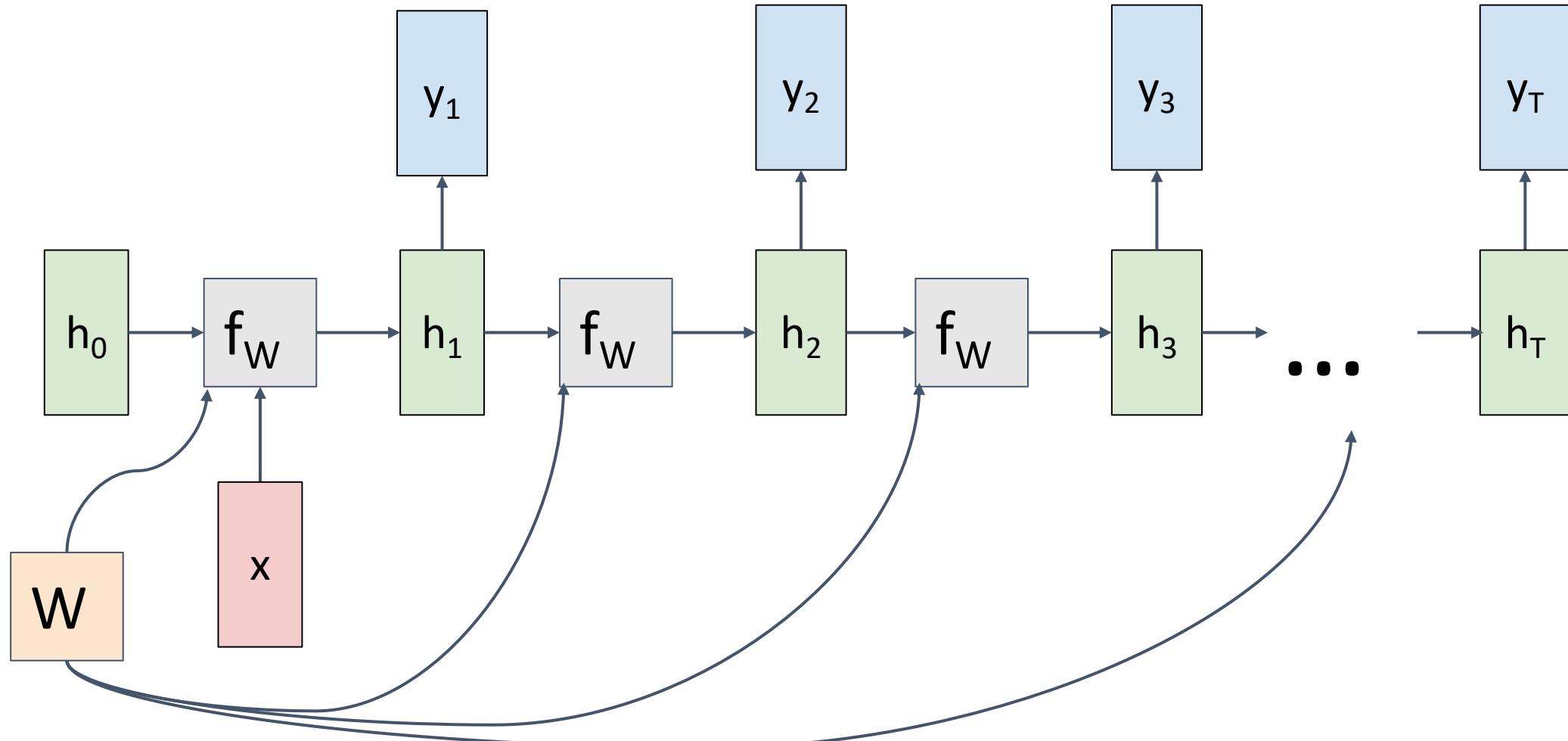
RNN Computational Graph (Many to Many)



RNN Computational Graph (Many to One)

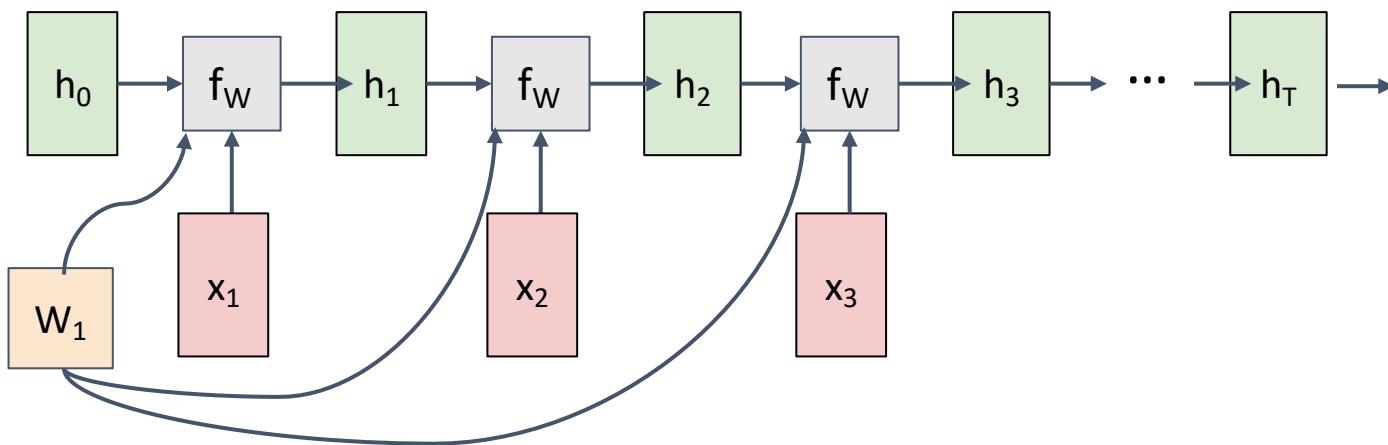


RNN Computational Graph (One to Many)



Sequence to Sequence (seq2seq) (Many to one) + (One to many)

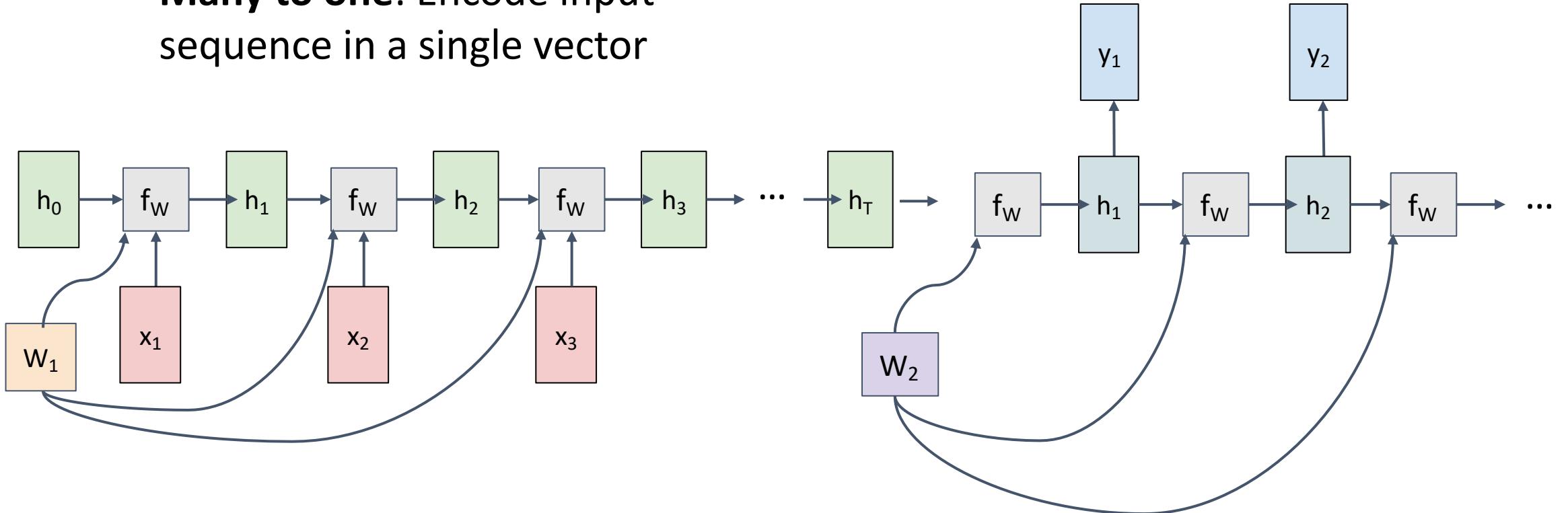
Many to one: Encode input sequence in a single vector



Sequence to Sequence (seq2seq) (Many to one) + (One to many)

One to many: Produce output sequence from single input vector

Many to one: Encode input sequence in a single vector

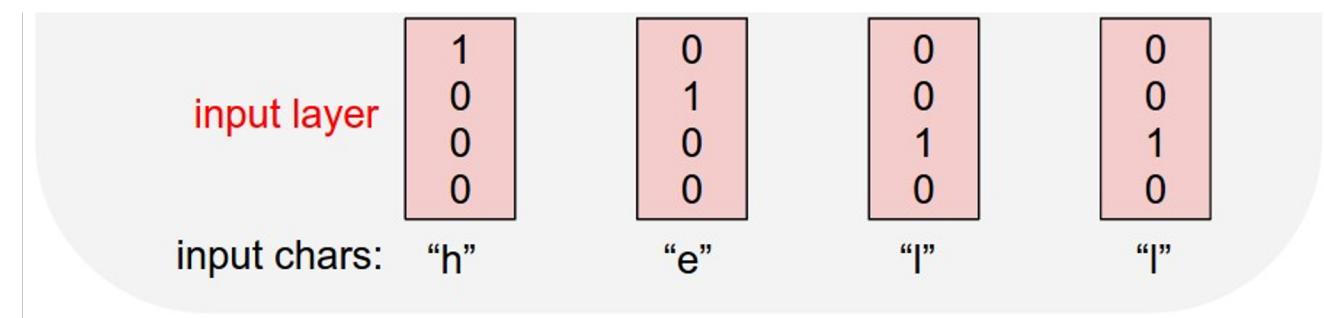


Example: Language Modeling

Given characters 1, 2, ..., t-1,
model predicts character t

Training sequence: "hello"

Vocabulary: [h, e, l, o]



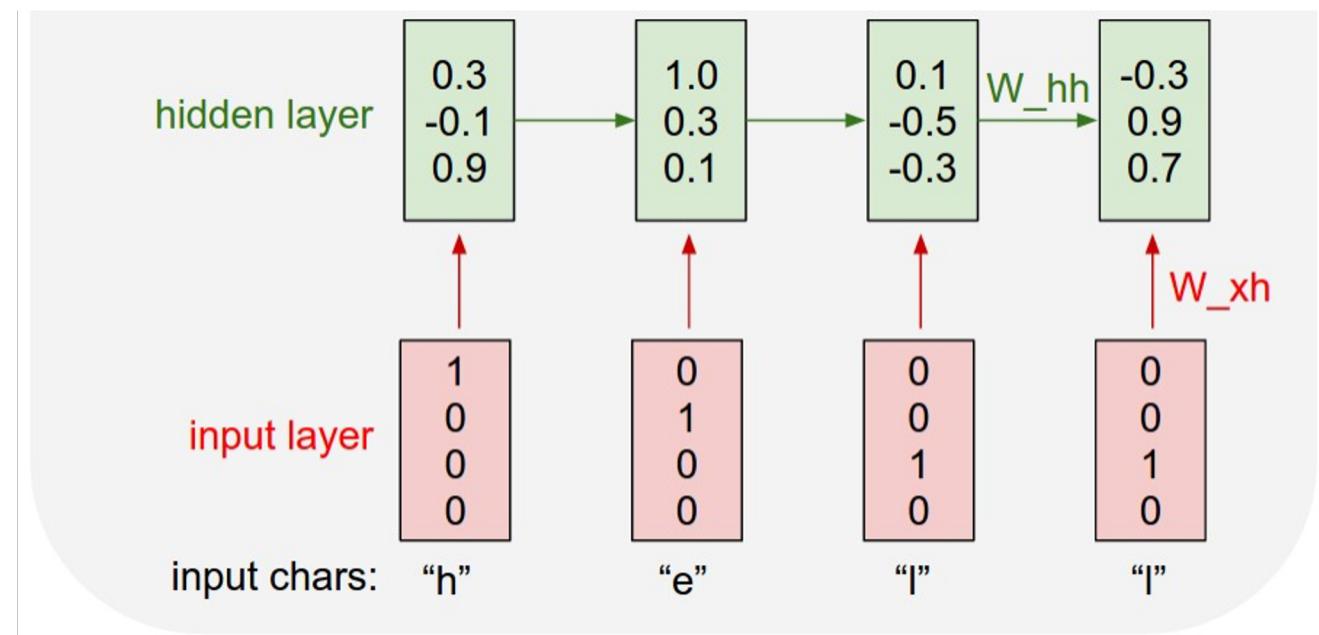
Example: Language Modeling

Given characters 1, 2, ..., t-1,
model predicts character t

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

Training sequence: "hello"

Vocabulary: [h, e, l, o]



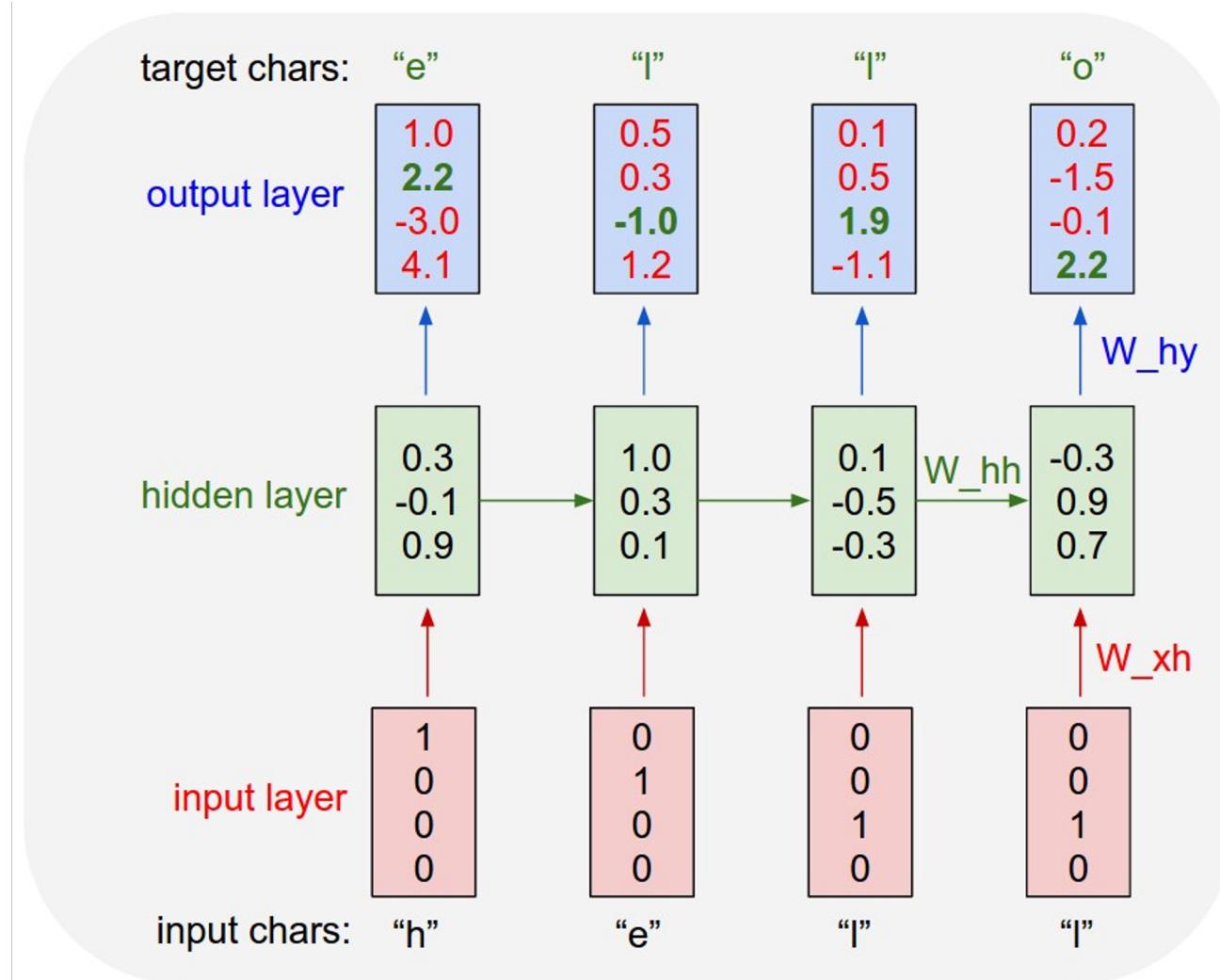
Example: Language Modeling

Given characters 1, 2, ..., t-1,
model predicts character t

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

Training sequence: "hello"

Vocabulary: [h, e, l, o]



Example: Language Modeling

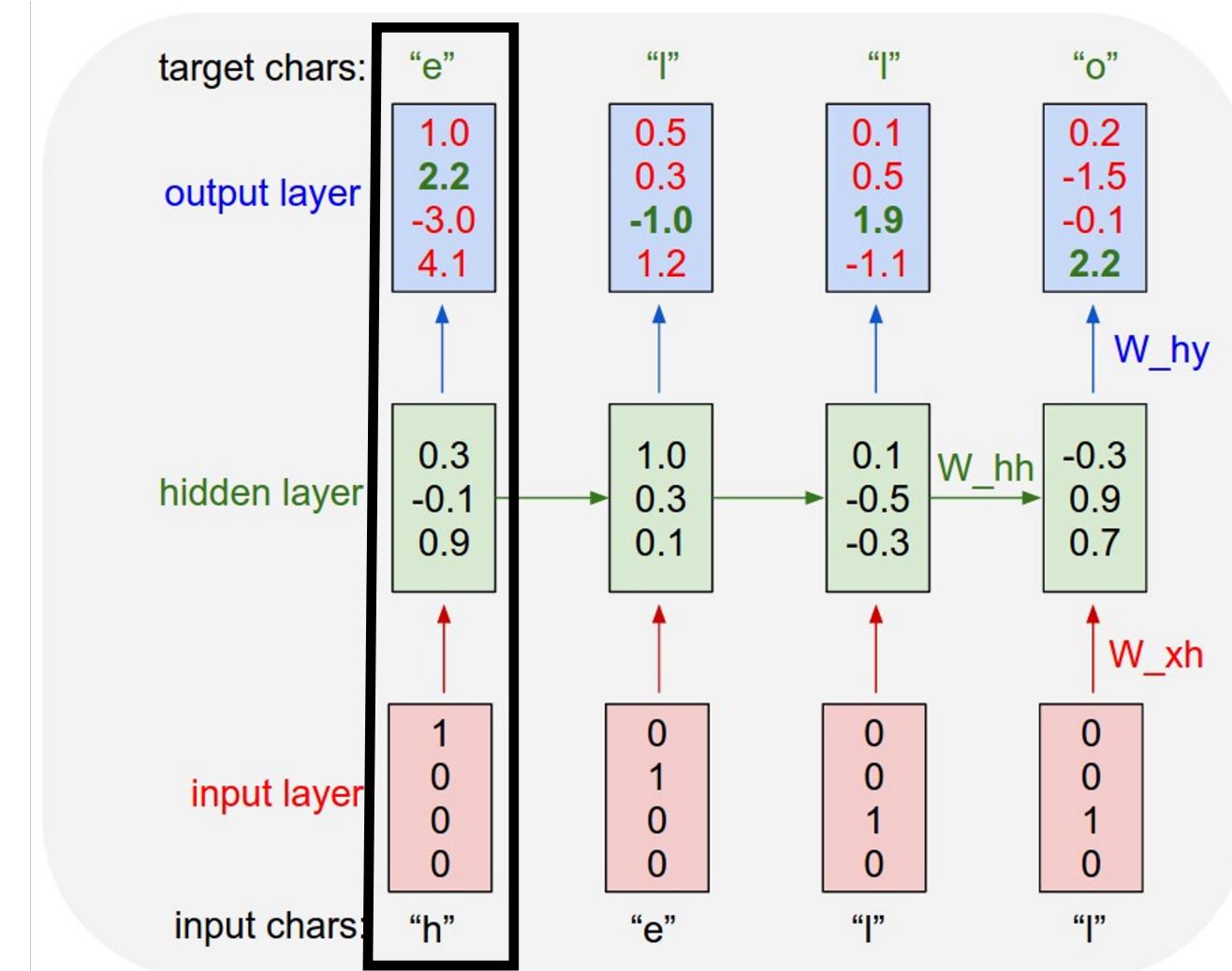
Given characters 1, 2, ..., t-1,
model predicts character t

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

Training sequence: "hello"

Vocabulary: [h, e, l, o]

Given "h", predict "e"



Example: Language Modeling

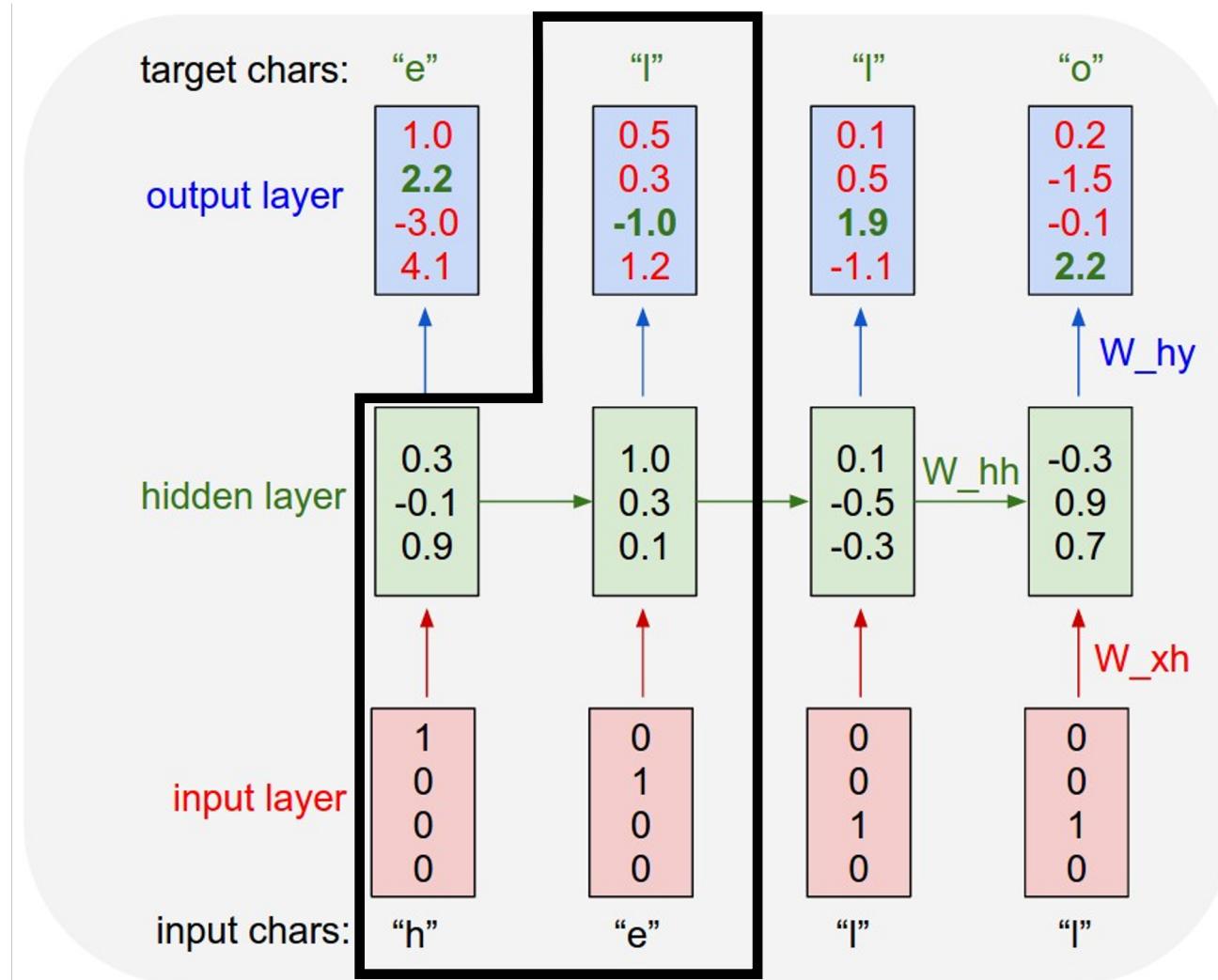
Given “he”, predict “l”

Given characters 1, 2, ..., t-1,
model predicts character t

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

Training sequence: “hello”

Vocabulary: [h, e, l, o]



Example: Language Modeling

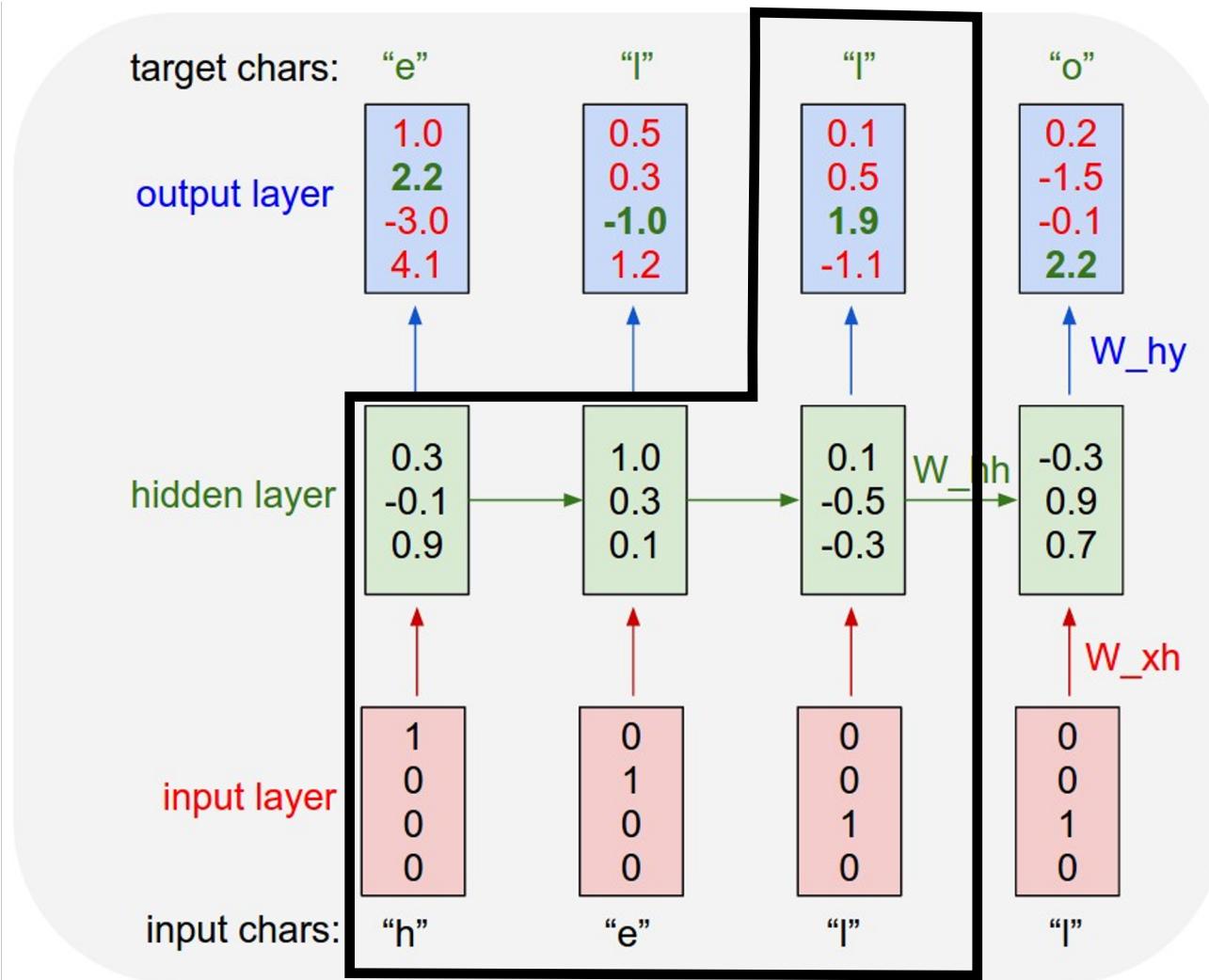
Given “hel”, predict “l”

Given characters 1, 2, ..., t-1,
model predicts character t

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

Training sequence: “hello”

Vocabulary: [h, e, l, o]



Example: Language Modeling

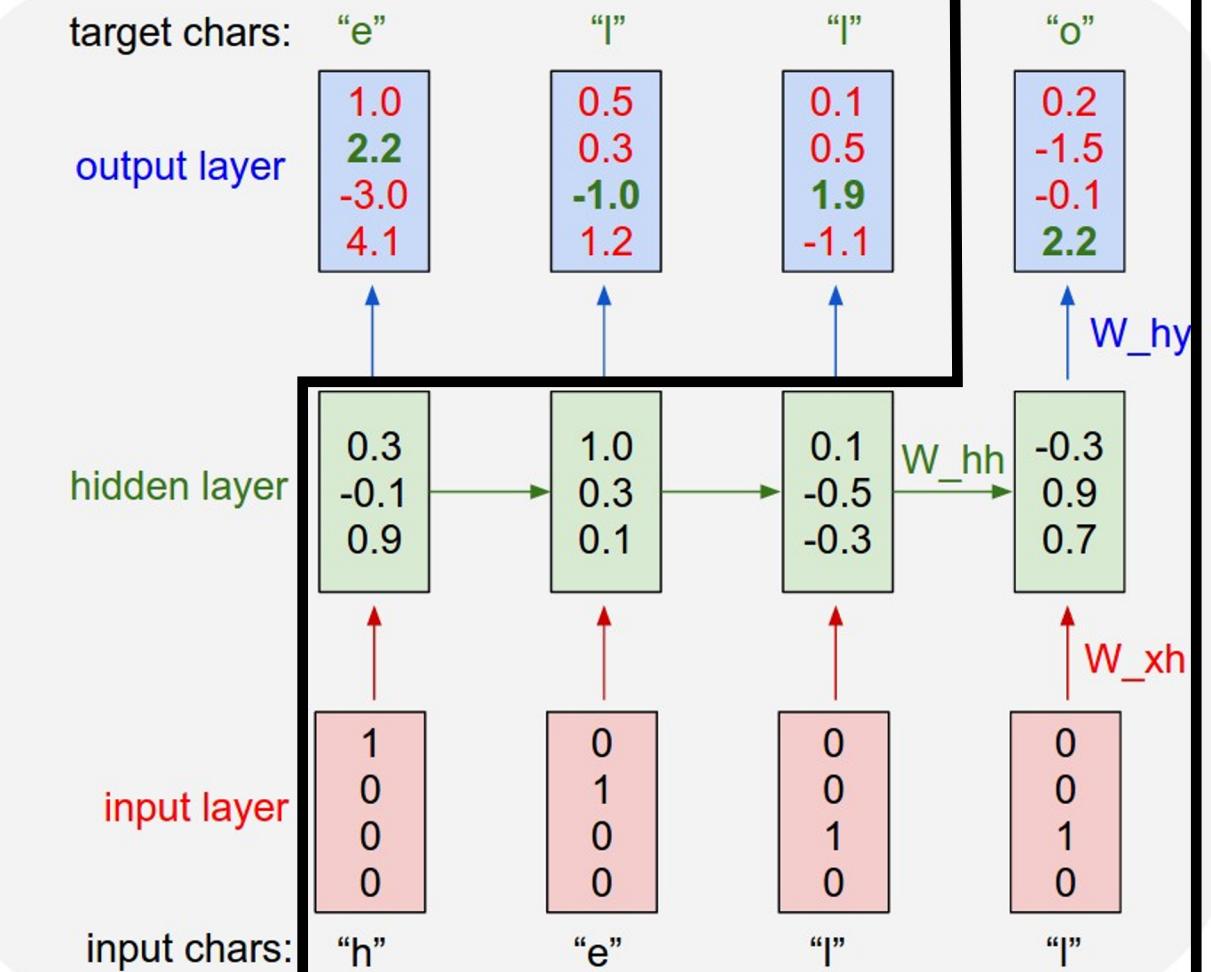
Given “hell”, predict “o”

Given characters 1, 2, ..., t-1,
model predicts character t

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

Training sequence: “hello”

Vocabulary: [h, e, l, o]

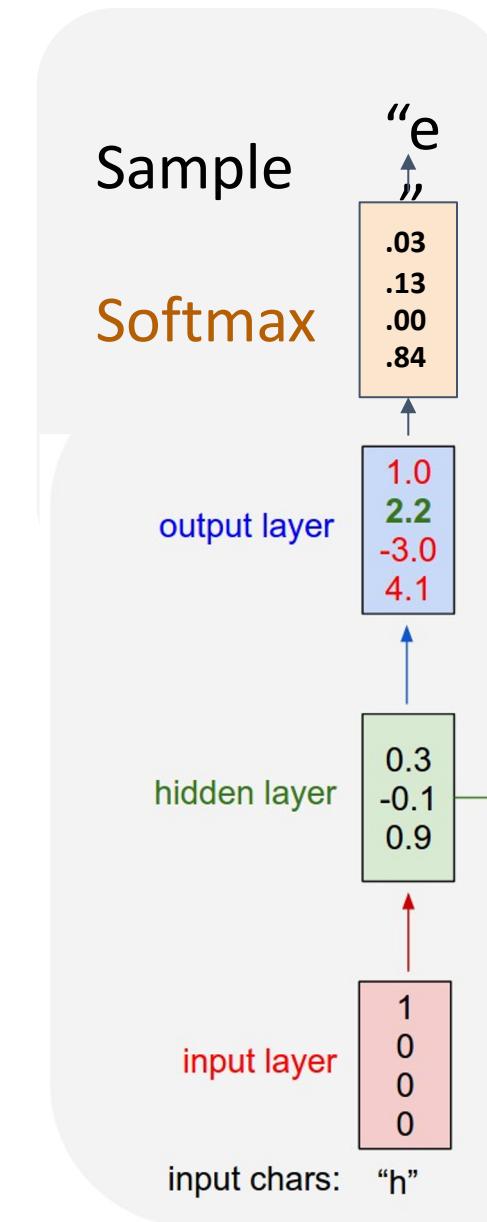


Example: Language Modeling

At test-time, **generate** new text: sample characters one at a time, feed back to model

Training sequence: "hello"

Vocabulary: [h, e, l, o]

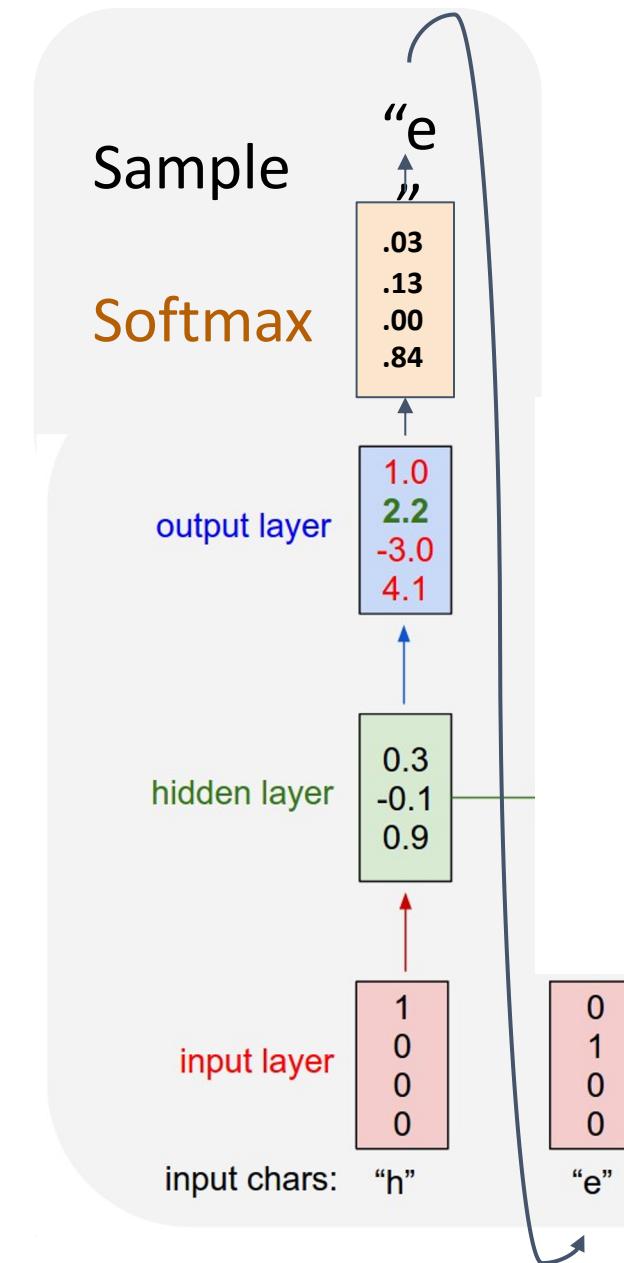


Example: Language Modeling

At test-time, **generate** new text: sample characters one at a time, feed back to model

Training sequence: "hello"

Vocabulary: [h, e, l, o]

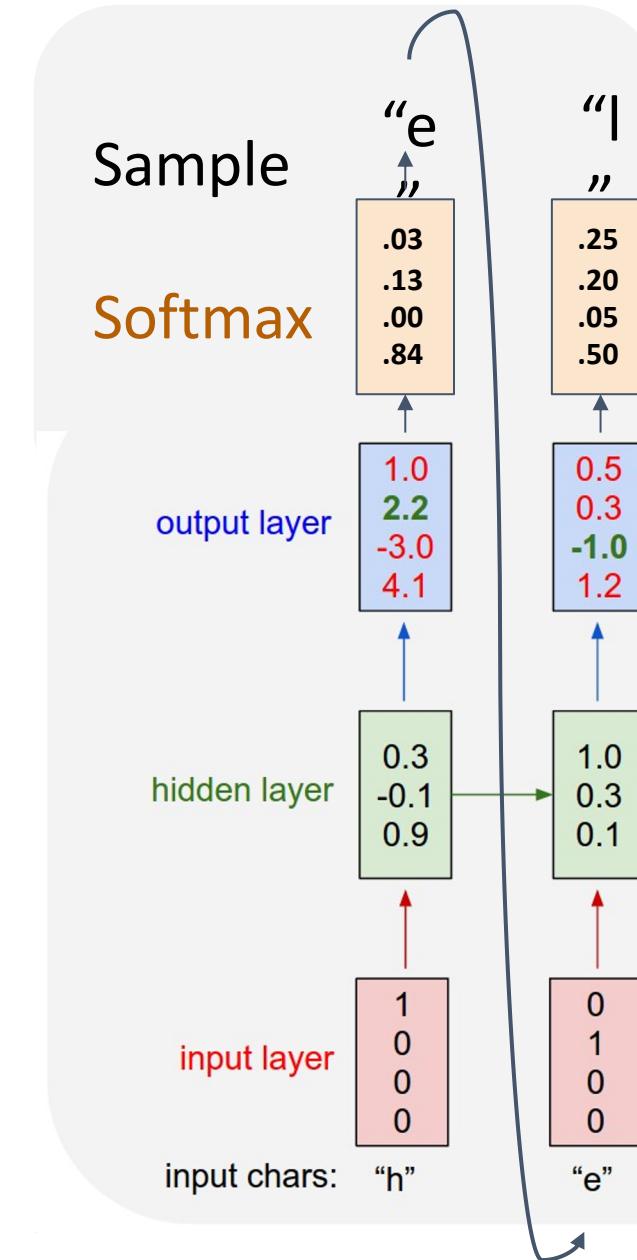


Example: Language Modeling

At test-time, **generate** new text: sample characters one at a time, feed back to model

Training sequence: "hello"

Vocabulary: [h, e, l, o]

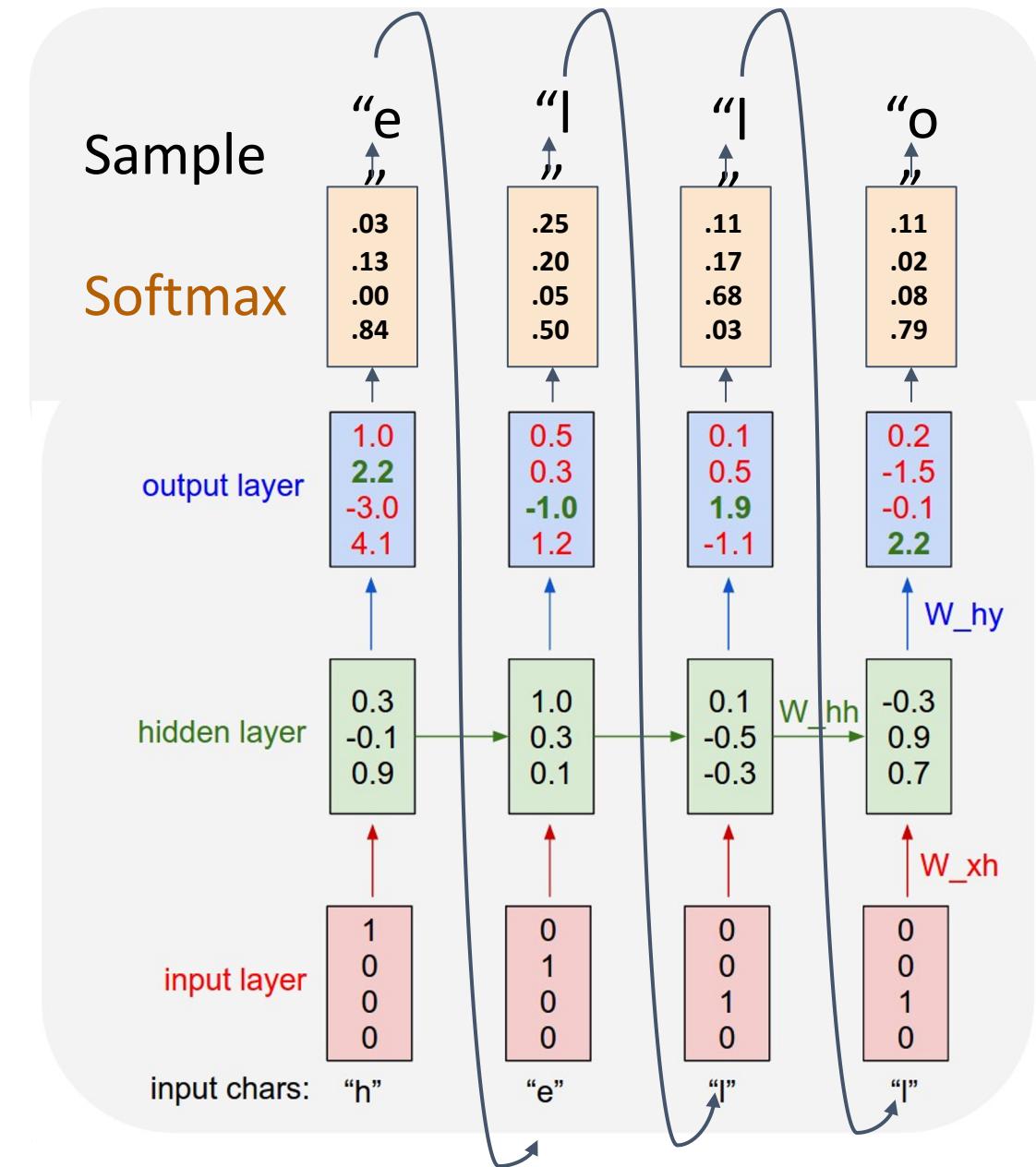


Example: Language Modeling

At test-time, **generate** new text: sample characters one at a time, feed back to model

Training sequence: "hello"

Vocabulary: [h, e, l, o]

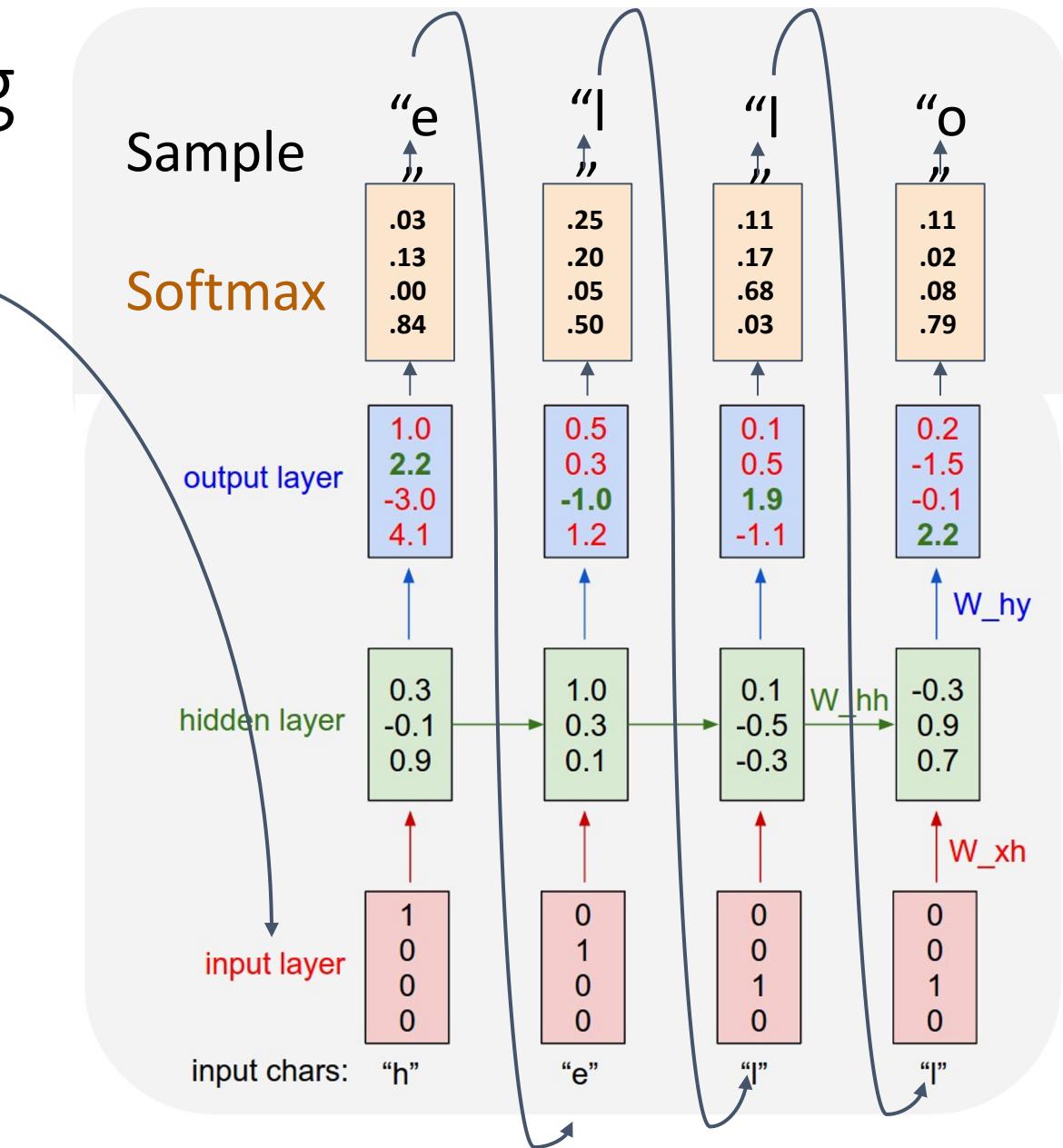


Example: Language Modeling

So far: encode inputs as **one-hot-vector**

$$\begin{aligned} [w_{11} w_{12} w_{13} w_{14}] [1] &= [w_{11}] \\ [w_{21} w_{22} w_{23} w_{14}] [0] &= [w_{21}] \\ [w_{31} w_{32} w_{33} w_{14}] [0] &= [w_{31}] \\ &\quad [0] \end{aligned}$$

Matrix multiply with a one-hot vector just extracts a column from the weight matrix. Often extract this into a separate **embedding layer**

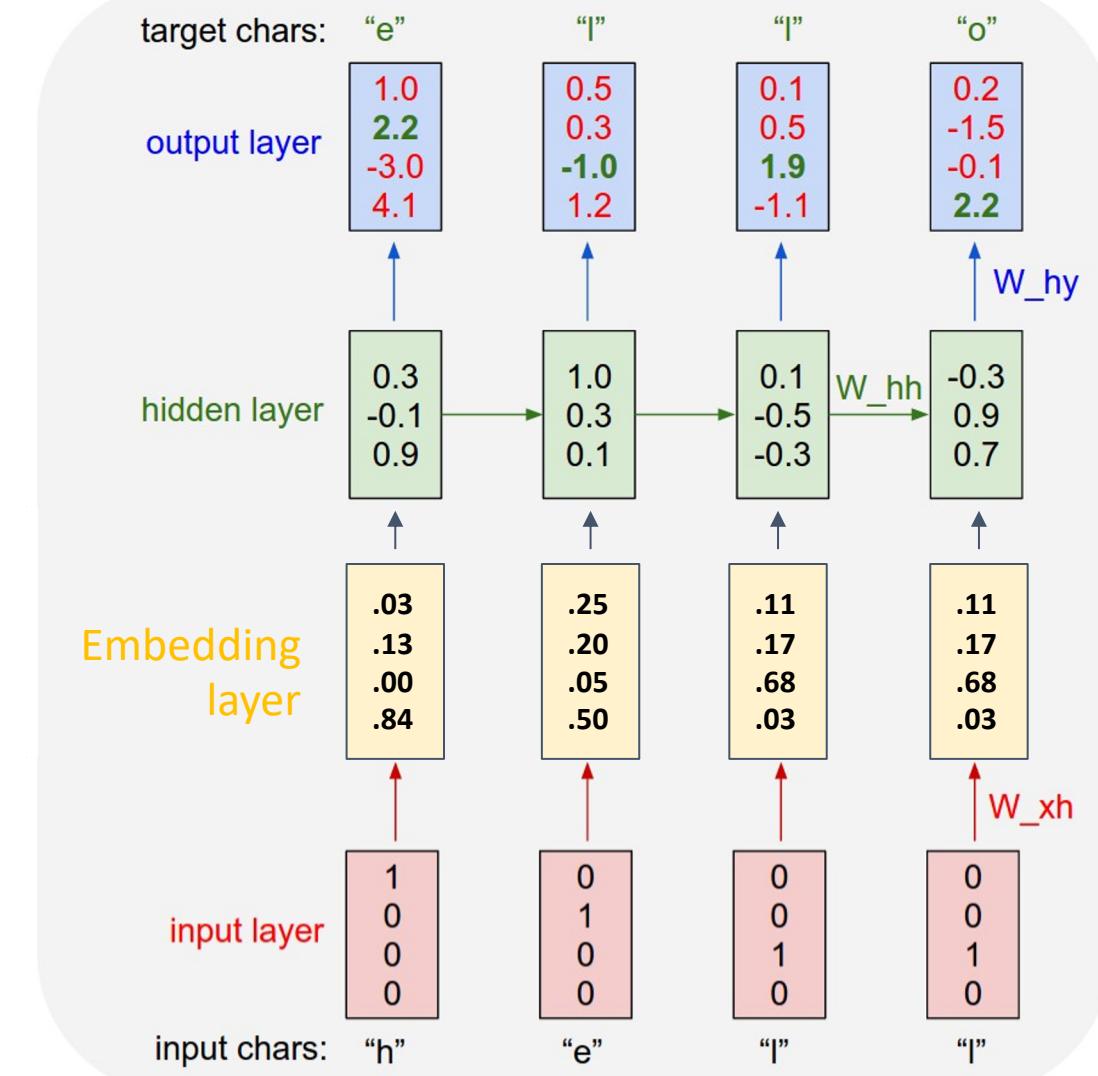


Example: Language Modeling

So far: encode inputs as **one-hot-vector**

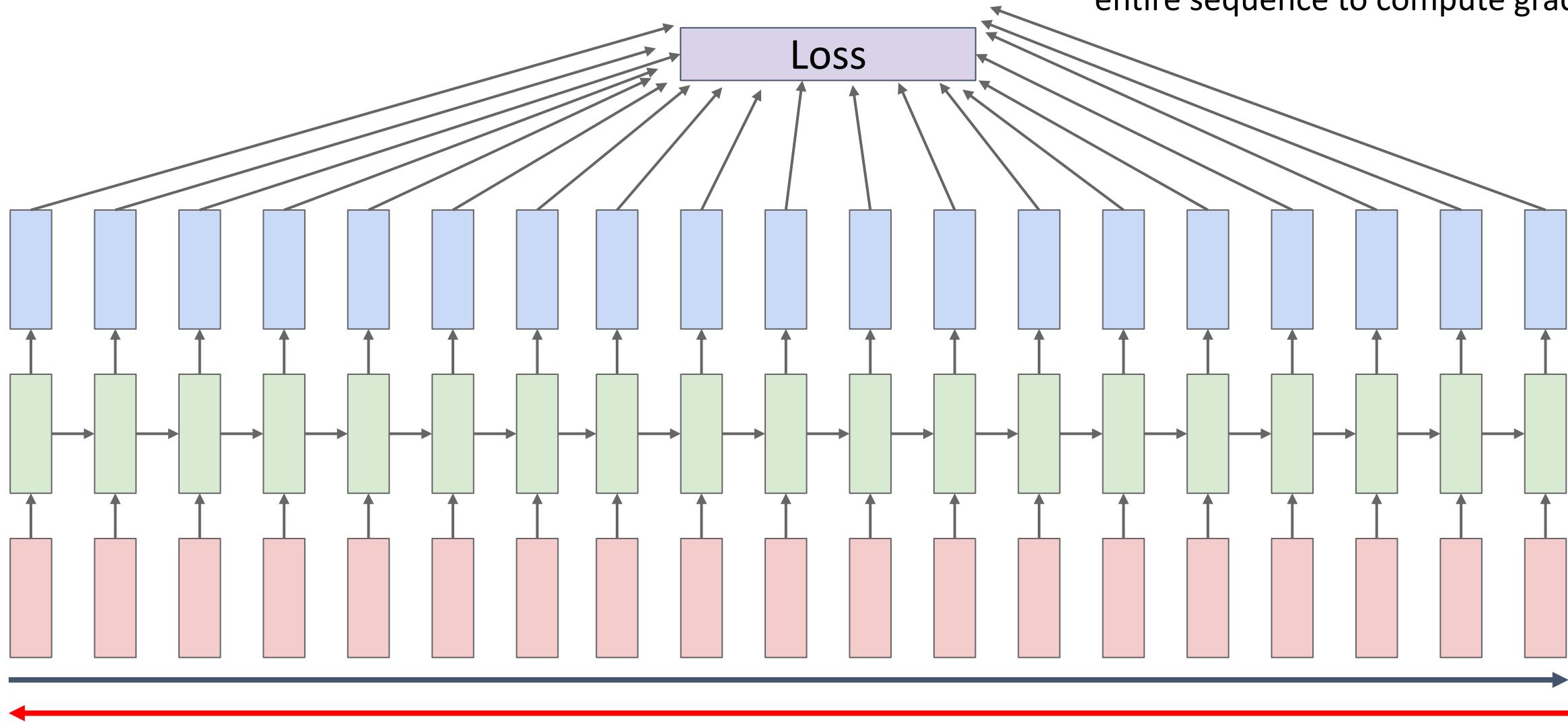
$$\begin{aligned} [w_{11} \ w_{12} \ w_{13} \ w_{14}] [1] &= [w_{11}] \\ [w_{21} \ w_{22} \ w_{23} \ w_{14}] [0] &= [w_{21}] \\ [w_{31} \ w_{32} \ w_{33} \ w_{14}] [0] &= [w_{31}] \\ &\quad [0] \end{aligned}$$

Matrix multiply with a one-hot vector just extracts a column from the weight matrix. Often extract this into a separate **embedding layer**



Backpropagation Through Time

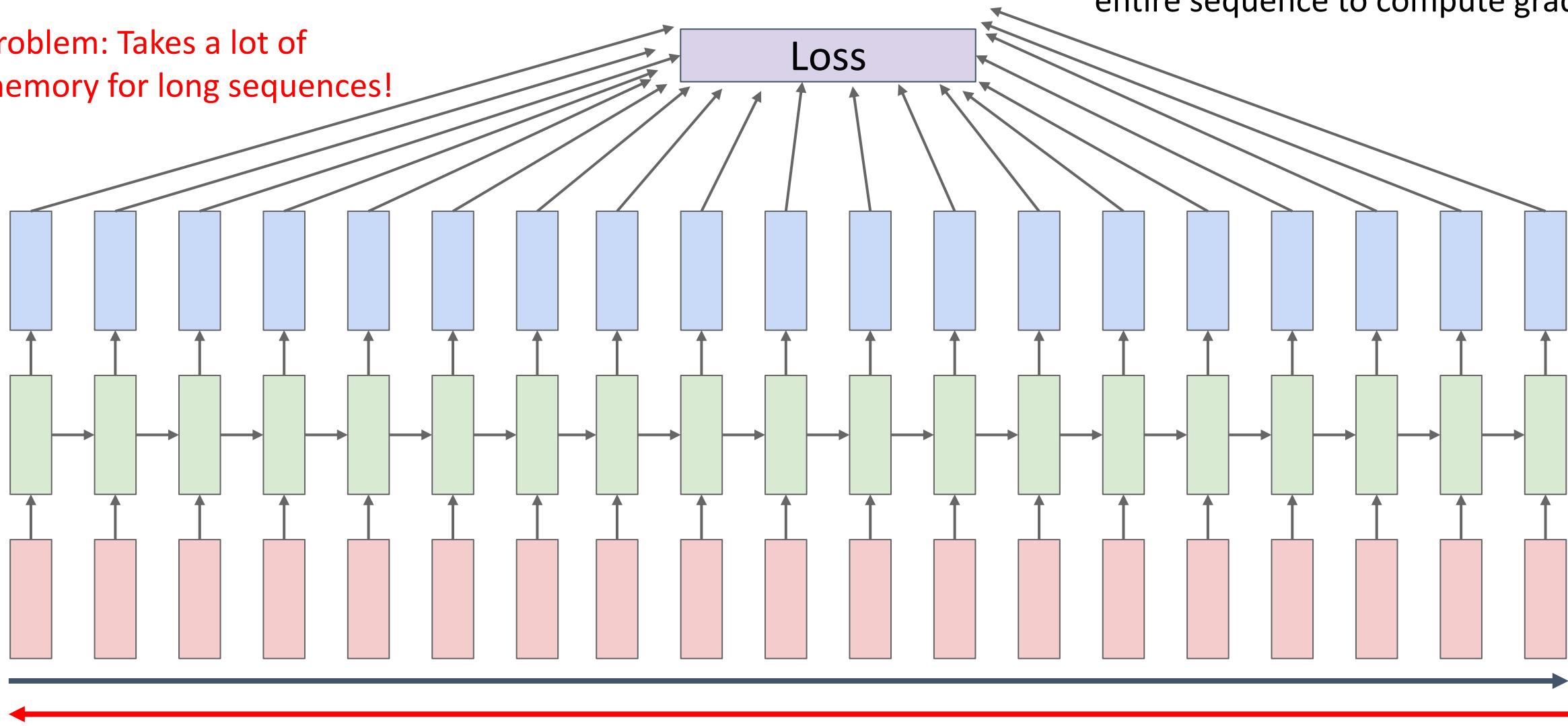
Forward through entire sequence to compute loss, then backward through entire sequence to compute gradient



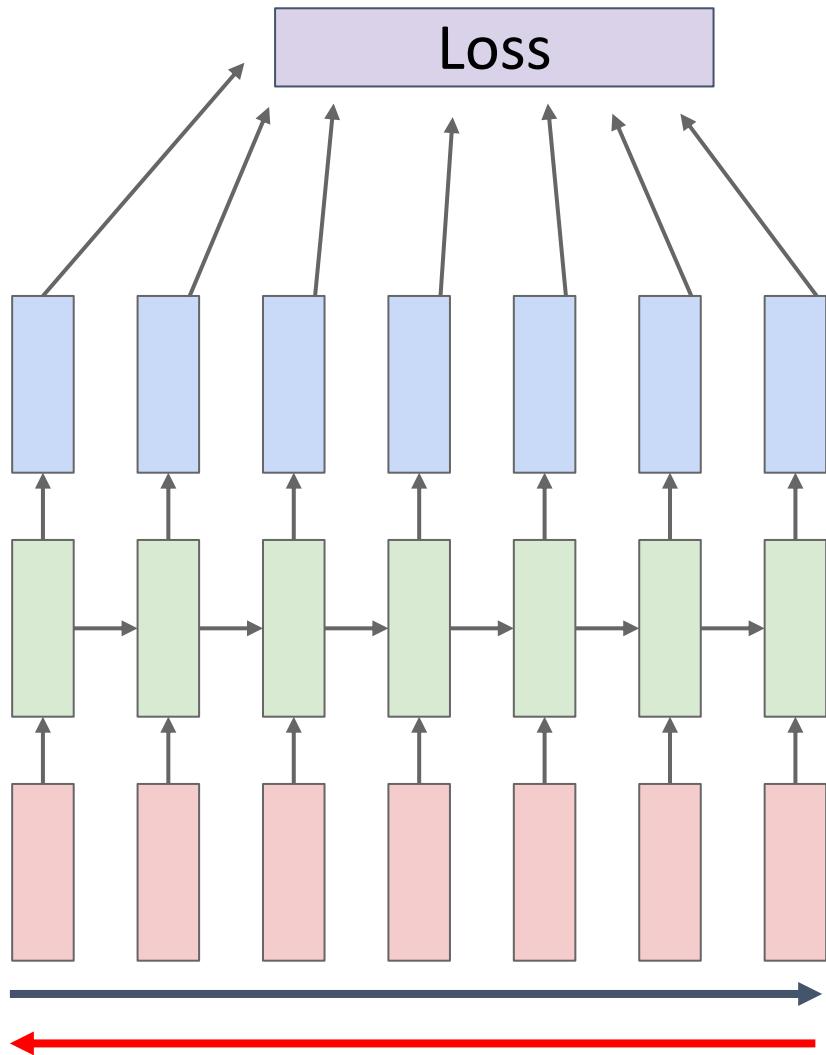
Backpropagation Through Time

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

Problem: Takes a lot of memory for long sequences!

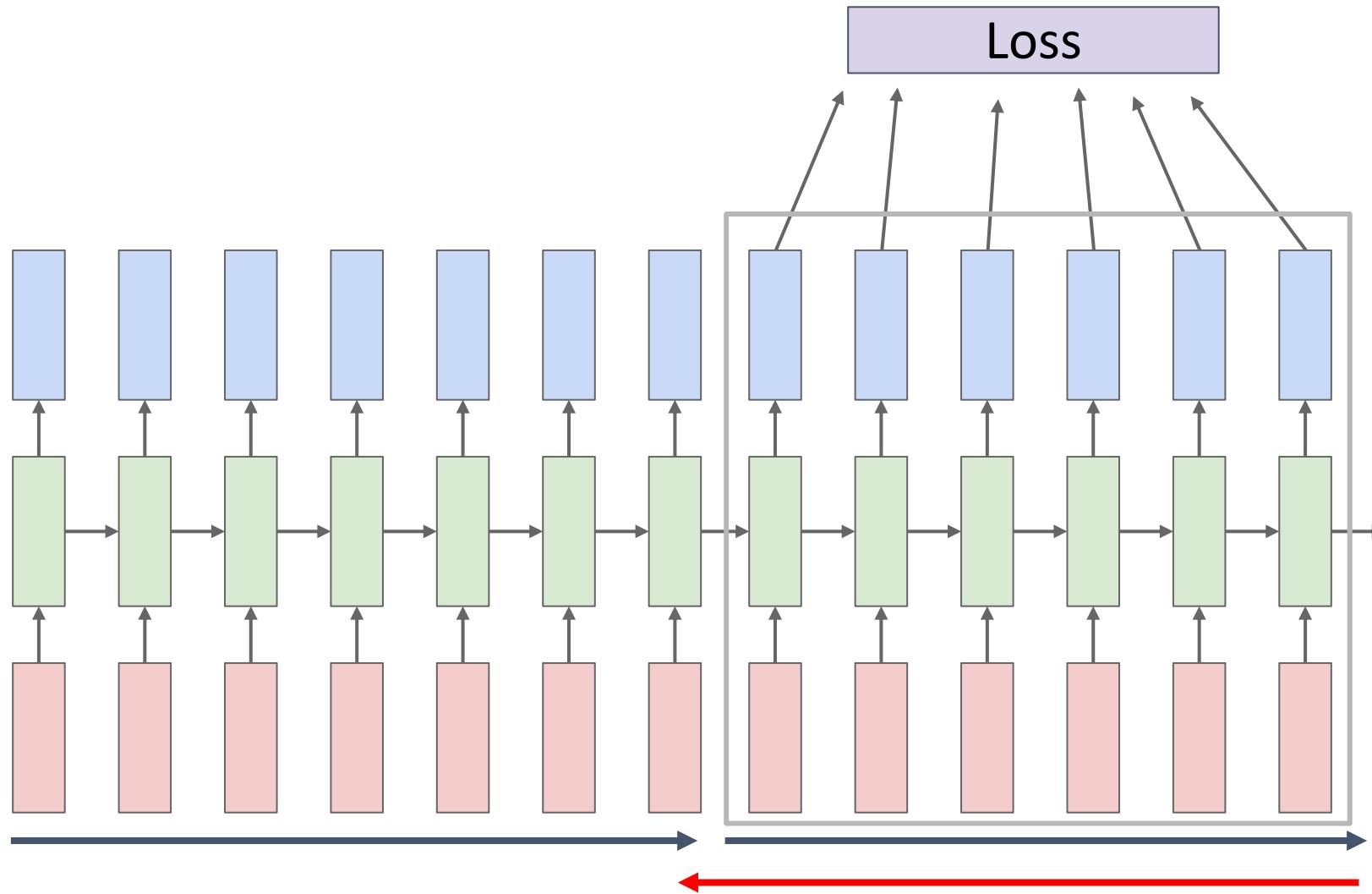


Truncated Backpropagation Through Time



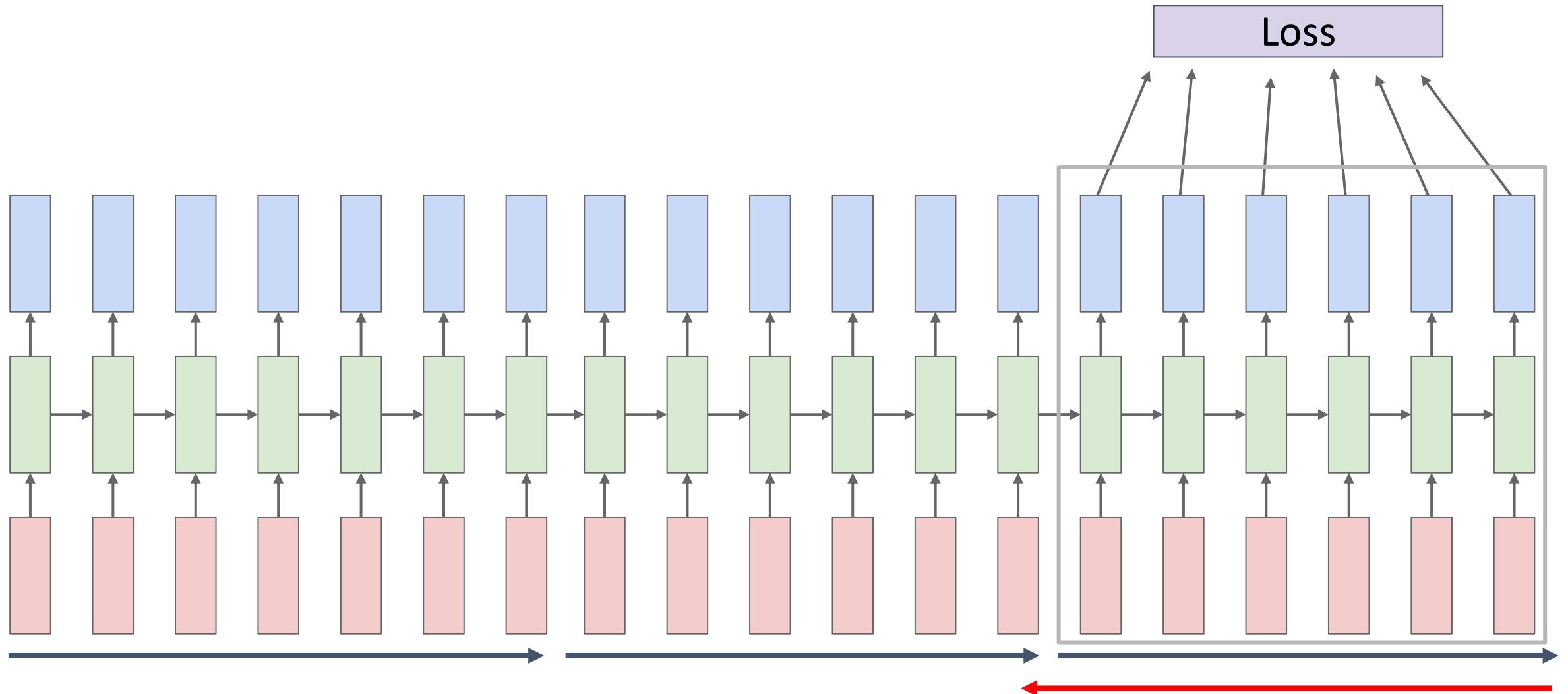
Run forward and backward
through chunks of the sequence
instead of whole sequence

Truncated Backpropagation Through Time



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

Truncated Backpropagation Through Time



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

```
1 """
2 Minimal character-level Vanilla RNN model. Written by Andrej Karpathy (@karpathy)
3 BSD License
4 """
5 import numpy as np
6
7 # data I/O
8 data = open('input.txt', 'r').read() # should be simple plain text file
9 chars = list(set(data))
10 data_size, vocab_size = len(data), len(chars)
11 print 'data has %d characters, %d unique.' % (data_size, vocab_size)
12 char_to_ix = { ch:i for i,ch in enumerate(chars) }
13 ix_to_char = { i:ch for i,ch in enumerate(chars) }
14
15 # hyperparameters
16 hidden_size = 100 # size of hidden layer of neurons
17 seq_length = 25 # number of steps to unroll the RNN for
18 learning_rate = 1e-1
19
20 # model parameters
21 wkh = np.random.randn(hidden_size, vocab_size)*0.01 # input to hidden
22 whh = np.random.randn(hidden_size, hidden_size)*0.01 # hidden to hidden
23 why = np.random.randn(vocab_size, hidden_size)*0.01 # hidden to output
24 bh = np.zeros((hidden_size, 1)) # hidden bias
25 by = np.zeros((vocab_size, 1)) # output bias
26
27 def lossFun(inputs, targets, hprev):
28     """
29     inputs,targets are both list of integers.
30     hprev is Hx1 array of initial hidden state
31     returns the loss, gradients on model parameters, and last hidden state
32     """
33     xs, hs, ys, ps = {}, {}, {}, {}
34     hs[-1] = np.copy(hprev)
35     loss = 0
36     # forward pass
37     for t in xrange(len(inputs)):
38         xs[t] = np.zeros((vocab_size,1)) # encode in 1-of-k representation
39         xs[t][inputs[t]] = 1
40         hs[t] = np.tanh(np.dot(wkh, xs[t]) + np.dot(whh, hs[t-1]) + bh) # hidden state
41         ys[t] = np.dot(why, hs[t]) + by # unnormalized log probabilities for next chars
42         ps[t] = np.exp(ys[t]) / np.sum(np.exp(ys[t])) # probabilities for next chars
43         loss += -np.log(ps[t][targets[t],0]) # softmax (cross-entropy loss)
44     # backward pass: compute gradients going backwards
45     dwxh, dwhh, dwhy = np.zeros_like(wkh), np.zeros_like(whh), np.zeros_like(why)
46     dbh, dy = np.zeros_like(bh), np.zeros_like(by)
47     dhnext = np.zeros_like(hs[0])
48     for t in reversed(xrange(len(inputs))):
49         dy = np.copy(ps[t])
50         dy[targets[t]] -= 1 # backprop into y
51         dwhy += np.dot(dy, hs[t].T)
52         dy += dy
53         dh = np.dot(why.T, dy) + dhnext # backprop into h
54         ddraw = (1 - hs[t] * hs[t]) * dh # backprop through tanh nonlinearity
55         dbh += ddraw
56         dwkh += np.dot(draw, xs[t].T)
57         dwhh += np.dot(draw, hs[t-1].T)
58         dhnext = np.dot(why.T, draw)
59         for dparam in [dwkh, dwhh, dwhy, dbh, dy]:
60             np.clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients
61     return loss, dwkh, dwhh, dwhy, dbh, dy, hs[len(inputs)-1]
62
63 def sample(h, seed_ix, n):
64     """
65     sample a sequence of integers from the model
66     h is memory state, seed_ix is seed letter for first time step
67     """
68     x = np.zeros((vocab_size, 1))
69     x[seed_ix] = 1
70     ixes = []
71     for t in xrange(n):
72         h = np.tanh(np.dot(wkh, x) + np.dot(whh, h) + bh)
73         y = np.dot(why, h) + by
74         p = np.exp(y) / np.sum(np.exp(y))
75         ix = np.random.choice(range(vocab_size), p=p.ravel())
76         x[ix] = 1
77         ixes.append(ix)
78     return ixes
79
80 n, p = 0, 0
81 mwxh, mwkh, mwwhy = np.zeros_like(wkh), np.zeros_like(whh), np.zeros_like(why)
82 mbh, mbp = np.zeros_like(bh), np.zeros_like(by) # memory variables for Adagrad
83 smooth_loss = -np.log(1.0/vocab_size)*seq_length # loss at iteration 0
84 while True:
85     # prepare inputs (we're sweeping from left to right in steps seq_length long)
86     if p+seq_length+1 >= len(data) or n == 0:
87         hprev = np.zeros((hidden_size,1)) # reset RNN memory
88         p = 0 # go from start of data
89         inputs = [char_to_ix[ch] for ch in data[p:p+seq_length]]
90         targets = [char_to_ix[ch] for ch in data[p+1:p+seq_length+1]]
91
92     # sample from the model now and then
93     if n % 100 == 0:
94         sample_ix = sample(hprev, inputs[0], 200)
95         txt = ''.join(ix_to_char[ix] for ix in sample_ix)
96         print '----\n%s----' % (txt, )
97
98     # forward seq_length characters through the net and fetch gradient
99     loss, dwkh, dwhh, dwhy, dbh, dy, hprev = lossFun(inputs, targets, hprev)
100    smooth_loss = smooth_loss * 0.999 + loss * 0.001
101    if n % 100 == 0: print 'iter %d, loss: %f' % (n, smooth_loss) # print progress
102
103    # perform parameter update with Adagrad
104    for param, dparam, mem in zip([wkh, whh, why, bh, by],
105                                 [dwkh, dwhh, dwhy, dbh, dy],
106                                 [mwxh, mwkh, mwwhy, mbh, mbp]):
107        mem += dparam * dparam
108        param += -learning_rate * dparam / np.sqrt(mem + 1e-8) # adagrad update
109
110    p += seq_length # move data pointer
111    n += 1 # iteration counter
```

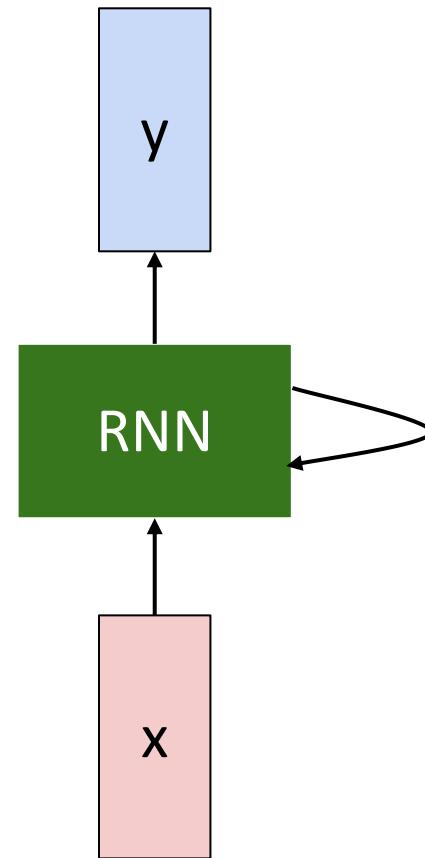
(<https://gist.github.com/karpathy/d4dee566867f8291f086>)

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 deserved 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.



Source: Karpathy, "The Unreasonable Effectiveness of Recurrent Neural Networks", 2015. <http://karpathy.github.io/2015/05/21/rnn-effectiveness/>

at first:

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

Source: Karpathy, "The Unreasonable Effectiveness of Recurrent Neural Networks", 2015. <http://karpathy.github.io/2015/05/21/rnn-effectiveness/>

at first:

tyntd-iafhatawiaoihrdemot lytdws e ,tfti, astai f ogoh eoase rrranbyne 'nhthnee e
plia tkldrgd 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."

Source: Karpathy, "The Unreasonable Effectiveness of Recurrent Neural Networks", 2015. <http://karpathy.github.io/2015/05/21/rnn-effectiveness/>

at first:

tyntd-iafhatawiaoihrdemot lytdws e ,tfti, astai f ogoh eoase rrranbyne 'nhthnee e
plia tkldrgd 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.

Source: Karpathy, "The Unreasonable Effectiveness of Recurrent Neural Networks", 2015. <http://karpathy.github.io/2015/05/21/rnn-effectiveness/>

at first:

tyntd-iafhatawiaoihrdemot lytdws e ,tfti, astai f ogoh eoase rrranbyne 'nhthnee e
plia tkldrgd 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.

Source: Karpathy, "The Unreasonable Effectiveness of Recurrent Neural Networks", 2015. <http://karpathy.github.io/2015/05/21/rnn-effectiveness/>

PANDARUS:

Alas, I think he shall be come approached and the day
When little strain 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.

Source: Karpathy, "The Unreasonable Effectiveness of Recurrent Neural Networks", 2015. <http://karpathy.github.io/2015/05/21/rnn-effectiveness/>

The Stacks Project: Open-Source Algebraic Geometry Textbook

The screenshot shows the homepage of the Stacks Project. The top navigation bar includes links for home, about, tags explained, tag lookup, browse, search, bibliography, recent comments, blog, and add slogans. Below this is a "Browse chapters" section with a table. The table has two columns: "Part" and "Chapter". The "Part" column lists "Preliminaries", "Schemes", "Topics in Scheme Theory", "Algebraic Spaces", "Topics in Geometry", "Deformation Theory", "Algebraic Stacks", and "Miscellany". The "Chapter" column lists numbered chapters from 1 to 10, each with links for online, TeX source, and PDF versions. To the right of the table is a "Parts" section containing a numbered list of the same eight parts. Below this is a "Statistics" section with a summary of the project's size.

Part	Chapter	online	TeX source	view pdf
Preliminaries	1. Introduction	online	tex	pdf
	2. Conventions	online	tex	pdf
	3. Set Theory	online	tex	pdf
	4. Categories	online	tex	pdf
	5. Topology	online	tex	pdf
	6. Sheaves on Spaces	online	tex	pdf
	7. Sites and Sheaves	online	tex	pdf
	8. Stacks	online	tex	pdf
	9. Fields	online	tex	pdf
	10. Commutative Algebra	online	tex	pdf

Parts

- [Preliminaries](#)
- [Schemes](#)
- [Topics in Scheme Theory](#)
- [Algebraic Spaces](#)
- [Topics in Geometry](#)
- [Deformation Theory](#)
- [Algebraic Stacks](#)
- [Miscellany](#)

Statistics

The Stacks project now consists of

- 455910 lines of code
- 14221 tags (56 inactive tags)
- 2366 sections

Latex source

<http://stacks.math.columbia.edu/>
The stacks project is licensed under the [GNU Free Documentation License](#)

For $\bigoplus_{n=1,\dots,m} \mathcal{L}_{m,n} = 0$, hence we can find a closed subset \mathcal{H} in \mathcal{H} and any sets \mathcal{F} on X , U is a closed immersion of S , then $U \rightarrow T$ is a separated algebraic space.

Proof. Proof of (1). It also start we get

$$S = \text{Spec}(R) = U \times_X U \times_X U$$

and the comparicoly in the fibre product covering we have to prove the lemma generated by $\coprod Z \times_U U \rightarrow V$. Consider the maps M along the set of points Sch_{fppf} and $U \rightarrow U$ is the fibre category of S in U in Section, ?? and the fact that any U affine, see Morphisms, Lemma ???. Hence we obtain a scheme S and any open subset $W \subset U$ in $\text{Sh}(G)$ such that $\text{Spec}(R') \rightarrow S$ is smooth or an

$$U = \bigcup U_i \times_{S_i} U_i$$

which has a nonzero morphism we may assume that f_i is of finite presentation over S . We claim that $\mathcal{O}_{X,x}$ is a scheme where $x, x' \in S'$ such that $\mathcal{O}_{X,x'} \rightarrow \mathcal{O}'_{X',x'}$ is separated. By Algebra, Lemma ?? we can define a map of complexes $\text{GL}_{S'}(x'/S'')$ and we win. \square

To prove study we see that $\mathcal{F}|_U$ is a covering of \mathcal{X}' , and \mathcal{T}_i is an object of $\mathcal{F}_{X/S}$ for $i > 0$ and \mathcal{F}_p exists and let \mathcal{F}_i be a presheaf of \mathcal{O}_X -modules on \mathcal{C} as a \mathcal{F} -module. In particular $\mathcal{F} = U/\mathcal{F}$ we have to show that

$$\widetilde{M}^\bullet = \mathcal{I}^\bullet \otimes_{\text{Spec}(k)} \mathcal{O}_{S,s} - i_X^{-1} \mathcal{F}$$

is a unique morphism of algebraic stacks. Note that

$$\text{Arrows} = (\text{Sch}/S)^{opp}_{fppf}, (\text{Sch}/S)_{fppf}$$

and

$$V = \Gamma(S, \mathcal{O}) \hookrightarrow (U, \text{Spec}(A))$$

is an open subset of X . Thus U is affine. This is a continuous map of X is the inverse, the groupoid scheme S .

Proof. See discussion of sheaves of sets. \square

The result for prove any open covering follows from the less of Example ???. It may replace S by $X_{\text{spaces},\text{étale}}$ which gives an open subspace of X and T equal to S_{Zar} , see Descent, Lemma ???. Namely, by Lemma ?? we see that R is geometrically regular over S .

Lemma 0.1. Assume (3) and (3) by the construction in the description.

Suppose $X = \lim |X|$ (by the formal open covering X and a single map $\underline{\text{Proj}}_X(\mathcal{A}) = \text{Spec}(B)$ over U compatible with the complex

$$\text{Set}(\mathcal{A}) = \Gamma(X, \mathcal{O}_{X,\mathcal{O}_X}).$$

When in this case of to show that $\mathcal{Q} \rightarrow \mathcal{C}_{Z/X}$ is stable under the following result in the second conditions of (1), and (3). This finishes the proof. By Definition ?? (without element is when the closed subschemes are catenary. If T is surjective we may assume that T is connected with residue fields of S . Moreover there exists a closed subspace $Z \subset X$ of X where U in X' is proper (some defining as a closed subset of the uniqueness it suffices to check the fact that the following theorem

(1) f is locally of finite type. Since $S = \text{Spec}(R)$ and $Y = \text{Spec}(R)$.

Proof. This is form all sheaves of sheaves on X . But given a scheme U and a surjective étale morphism $U \rightarrow X$. Let $U \cap U = \coprod_{i=1,\dots,n} U_i$ be the scheme X over S at the schemes $X_i \rightarrow X$ and $U = \lim_i X_i$. \square

The following lemma surjective restrocomposes of this implies that $\mathcal{F}_{x_0} = \mathcal{F}_{x_0} = \mathcal{F}_{\mathcal{X},\dots,0}$.

Lemma 0.2. Let X be a locally Noetherian scheme over S , $E = \mathcal{F}_{X/S}$. Set $\mathcal{I} = \mathcal{J}_1 \subset \mathcal{I}'_n$. Since $\mathcal{I}^n \subset \mathcal{I}^n$ are nonzero over $i_0 \leq p$ is a subset of $\mathcal{J}_{n,0} \circ \overline{A}_2$ works.

Lemma 0.3. In Situation ???. Hence we may assume $q' = 0$.

Proof. We will use the property we see that p is the next functor (??). On the other hand, by Lemma ?? we see that

$$D(\mathcal{O}_{X'}) = \mathcal{O}_X(D)$$

where K is an F -algebra where δ_{n+1} is a scheme over S . \square

Proof. Omitted. □

Lemma 0.1. Let \mathcal{C} be a set of the construction.

Let \mathcal{C} be a gerber covering. Let \mathcal{F} be a quasi-coherent sheaves of \mathcal{O} -modules. We have to show that

$$\mathcal{O}_{\mathcal{O}_X} = \mathcal{O}_X(\mathcal{L})$$

Proof. This is an algebraic space with the composition of sheaves \mathcal{F} on $X_{\text{étale}}$ we have

$$\mathcal{O}_X(\mathcal{F}) = \{\text{morph}_1 \times_{\mathcal{O}_X} (\mathcal{G}, \mathcal{F})\}$$

where \mathcal{G} defines an isomorphism $\mathcal{F} \rightarrow \mathcal{F}$ of \mathcal{O} -modules. □

Lemma 0.2. This is an integer \mathcal{Z} is injective.

Proof. See Spaces, Lemma ??.

Lemma 0.3. Let S be a scheme. Let X be a scheme and X is an affine open covering. Let $\mathcal{U} \subset \mathcal{X}$ be a canonical and locally of finite type. Let X be a scheme. Let X be a scheme which is equal to the formal complex.

The following to the construction of the lemma follows.

Let X be a scheme. Let X be a scheme covering. Let

$$b : X \rightarrow Y' \rightarrow Y \rightarrow Y \rightarrow Y' \times_X Y \rightarrow X.$$

be a morphism of algebraic spaces over S and Y .

Proof. Let X be a nonzero scheme of X . Let X be an algebraic space. Let \mathcal{F} be a quasi-coherent sheaf of \mathcal{O}_X -modules. The following are equivalent

- (1) \mathcal{F} is an algebraic space over S .
- (2) If X is an affine open covering.

Consider a common structure on X and X the functor $\mathcal{O}_X(U)$ which is locally of finite type. □

This since $\mathcal{F} \in \mathcal{F}$ and $x \in \mathcal{G}$ the diagram

$$\begin{array}{ccccc}
 S & \longrightarrow & & & \\
 \downarrow & & & & \\
 \xi & \longrightarrow & \mathcal{O}_{X'} & \nearrow & \\
 \text{gor}_s & & \uparrow & & \\
 & & =\alpha' & \longrightarrow & \\
 & & \downarrow & & \\
 & & =\alpha' & \longrightarrow & \alpha \\
 & & & & \\
 \text{Spec}(K_\psi) & & \text{Mor}_{\text{Sets}} & & d(\mathcal{O}_{X_{/\mathbb{A}}}, \mathcal{G}) \\
 & & & & \\
 & & & & X \\
 & & & & \downarrow \\
 & & & &
 \end{array}$$

is a limit. Then \mathcal{G} is a finite type and assume S is a flat and \mathcal{F} and \mathcal{G} is a finite type f_* . This is of finite type diagrams, and

- the composition of \mathcal{G} is a regular sequence,
- $\mathcal{O}_{X'}$ is a sheaf of rings.

□

Proof. We have see that $X = \text{Spec}(R)$ and \mathcal{F} is a finite type representable by algebraic space. The property \mathcal{F} is a finite morphism of algebraic stacks. Then the cohomology of X is an open neighbourhood of U . □

Proof. This is clear that \mathcal{G} is a finite presentation, see Lemmas ??.

A reduced above we conclude that U is an open covering of C . The functor \mathcal{F} is a “field”

$$\mathcal{O}_{X,x} \rightarrow \mathcal{F}_{\bar{x}} \dashrightarrow (\mathcal{O}_{X_{\text{étale}}}) \rightarrow \mathcal{O}_{X_{\ell}}^{-1} \mathcal{O}_{X_{\lambda}}(\mathcal{O}_{X_{\eta}}^{\bar{v}})$$

is an isomorphism of covering of \mathcal{O}_{X_i} . If \mathcal{F} is the unique element of \mathcal{F} such that X is an isomorphism.

The property \mathcal{F} is a disjoint union of Proposition ?? and we can filtered set of presentations of a scheme \mathcal{O}_X -algebra with \mathcal{F} are opens of finite type over S . If \mathcal{F} is a scheme theoretic image points. □

If \mathcal{F} is a finite direct sum $\mathcal{O}_{X_{\lambda}}$ is a closed immersion, see Lemma ???. This is a sequence of \mathcal{F} is a similar morphism.



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



branch: master

linux / +



Code



74
Pull requests



Pulse



Graphs

HTTPS clone URL

<https://github.com/torvalds/linux>



You can clone with [HTTPS](#),
[SSH](#), or [Subversion](#).

Clone in Desktop

Download ZIP

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
io	Memory barriers: New version of writeback_lru_index() introduced	a month ago

```
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

```
/*
 * Copyright (c) 2006-2010, Intel Mobile Communications. All rights reserved.
 *
 * This program is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 as published by
 * the Free Software Foundation.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 *
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software Foundation,
 * Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
 */

#include <linux/kexec.h>
#include <linux/errno.h>
#include <linux/io.h>
#include <linux/platform_device.h>
#include <linux/multi.h>
#include <linux/ckevent.h>

#include <asm/io.h>
#include <asm/prom.h>
#include <asm/e820.h>
#include <asm/system_info.h>
#include <asm/seteew.h>
#include <asm/pgproto.h>
```

```
#include <asm/io.h>
#include <asm/prom.h>
#include <asm/e820.h>
#include <asm/system_info.h>
#include <asm/seteew.h>
#include <asm/pgproto.h>

#define REG_PG      vesa_slot_addr_pack
#define PFM_NOCOMP  AFSR(0, load)
#define STACK_DDR(type)      (func)

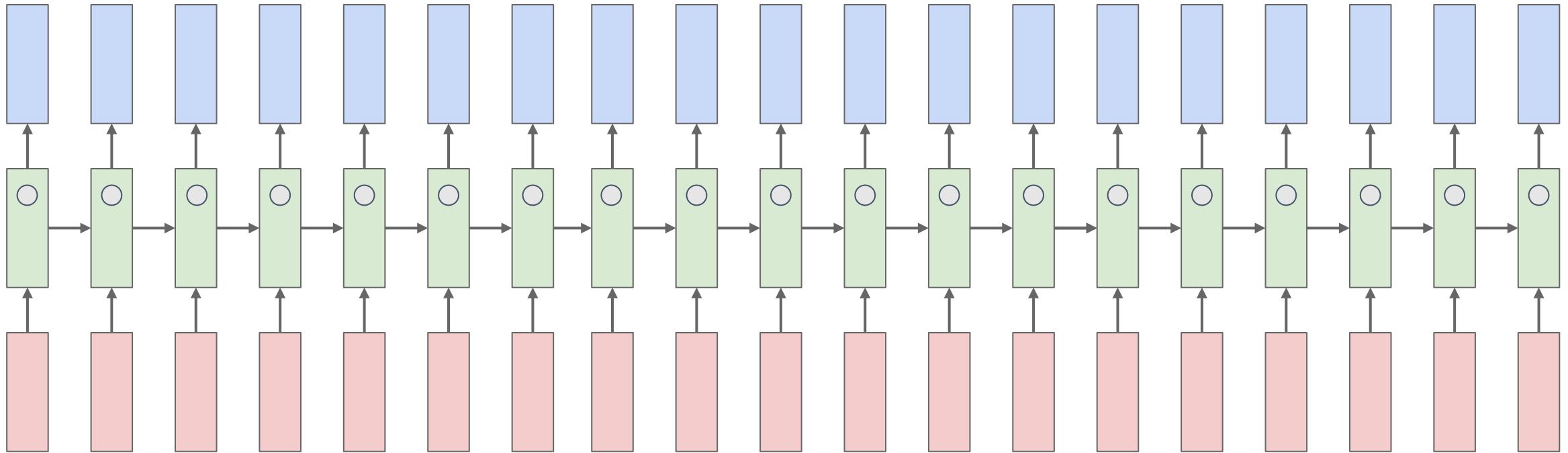
#define SWAP_ALLOCATE(nr)      (e)
#define emulate_sigs()  arch_get_unaligned_child()
#define access_rw(TST)  asm volatile("movd %%esp, %0, %3" : : "r" (0)); \
    if (__type & DO_READ)

static void stat_PC_SEC __read_mostly offsetof(struct seq_argsqueue, \
    pC>[1]);

static void
os_prefix(unsigned long sys)
{
#ifdef CONFIG_PREEMPT
    PUT_PARAM_RAID(2, sel) = get_state_state();
    set_pid_sum((unsigned long)state, current_state_str(),
                (unsigned long)-1->lr_full; low;
}

```

Searching for Interpretable Hidden Units



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

Searching for Interpretable Hidden Units

```
/* Unpack a filter field's string representation from user-space
 * buffer. */
char *audit_unpack_string(void **bufp, size_t *remain, size_t len)
{
    char *str;
    if (!*bufp || (len == 0) || (len > *remain))
        return ERR_PTR(-EINVAL);
    /* Of the currently implemented string fields, PATH_MAX
     * defines the longest valid length.
*/
```

Searching for Interpretable Hidden Units

"You mean to imply that I have nothing to eat out of.... On the contrary, I can supply you with everything even if you want to give dinner parties," warmly replied Chichagov, who tried by every word he spoke to prove his own rectitude and therefore imagined Kutuzov to be animated by the same desire.

Kutuzov, shrugging his shoulders, replied with his subtle penetrating smile: "I meant merely to say what I said."

quote detection cell

Searching for Interpretable Hidden Units

Cell sensitive to position in line:

The sole importance of the crossing of the Berezina lies in the fact that it plainly and indubitably proved the fallacy of all the plans for cutting off the enemy's retreat and the soundness of the only possible line of action--the one Kutuzov and the general mass of the army demanded--namely, simply to follow the enemy up. The French crowd fled at a continually increasing speed and all its energy was directed to reaching its goal. It fled like a wounded animal and it was impossible to block its path. This was shown not so much by the arrangements it made for crossing as by what took place at the bridges. When the bridges broke down, unarmed soldiers, people from Moscow and women with children who were with the French transport, all--carried on by vis inertiae--pressed forward into boats and into the ice-covered water and did not, surrender.

line length tracking cell

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

Figures copyright Karpathy, Johnson, and Fei-Fei; reproduced with permission

Searching for Interpretable Hidden Units

```
static int __dequeue_signal(struct sigpending *pending, sigset_t *mask,
    siginfo_t *info)
{
    int sig = next_signal(pending, mask);
    if (sig) {
        if (current->notifier) {
            if (sigismember(current->notifier_mask, sig)) {
                if (!(current->notifier)(current->notifier_data)) {
                    clear_thread_flag(TIF_SIGPENDING);
                    return 0;
                }
            }
        }
        collect_signal(sig, pending, info);
    }
    return sig;
}
```

if statement cell

Searching for Interpretable Hidden Units

Cell that turns on inside comments and quotes:

```
/* Duplicate LSM field information. The lsm_rule is opaque, so
 * re-initialized. */
static inline int audit_dupe_lsm_field(struct audit_field *df,
    struct audit_field *sf)
{
    int ret = 0;
    char *lsm_str;
    /* our own copy of lsm_str */
    lsm_str = kstrdup(sf->lsm_str, GFP_KERNEL);
    if (unlikely(!lsm_str))
        return -ENOMEM;
    df->lsm_str = lsm_str;
    /* our own (refreshed) copy of lsm_rule */
    ret = security_audit_rule_init(df->type, df->op, df->lsm_str,
        (void **) &df->lsm_rule);
    /* Keep currently invalid fields around in case they
     * become valid after a policy reload. */
    if (ret == -EINVAL) {
        pr_warn("audit rule for LSM \\'%s\\' is invalid\n",
            df->lsm_str);
        ret = 0;
    }
    return ret;
}
```

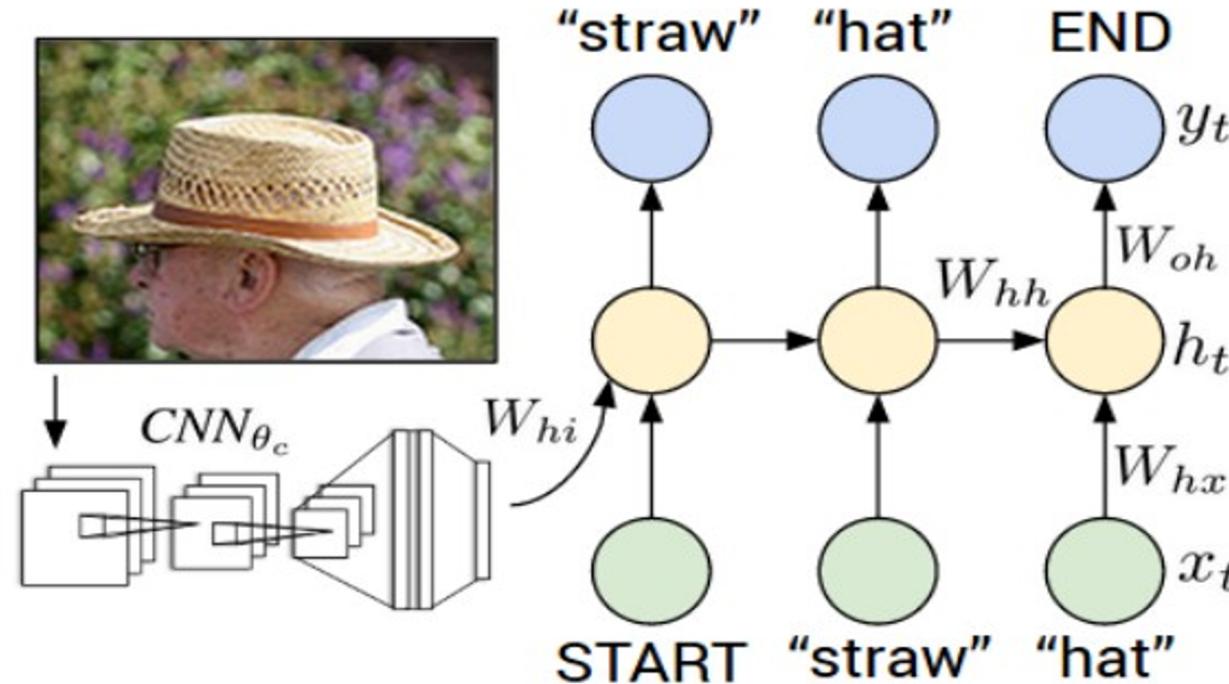
quote/comment cell

Searching for Interpretable Hidden Units

```
#ifdef CONFIG_AUDITSYSCALL
static inline int audit_match_class_bits(int class, u32 *mask)
{
    int i;
    if (classes[class]) {
        for (i = 0; i < AUDIT_BITMASK_SIZE; i++)
            if (mask[i] & classes[class][i])
                return 0;
    }
    return 1;
}
```

code depth cell

Example: Image Captioning



Mao et al, "Explain Images with Multimodal Recurrent Neural Networks", NeurIPS 2014 Deep Learning and Representation Workshop

Karpathy and Fei-Fei, "Deep Visual-Semantic Alignments for Generating Image Descriptions", CVPR 2015

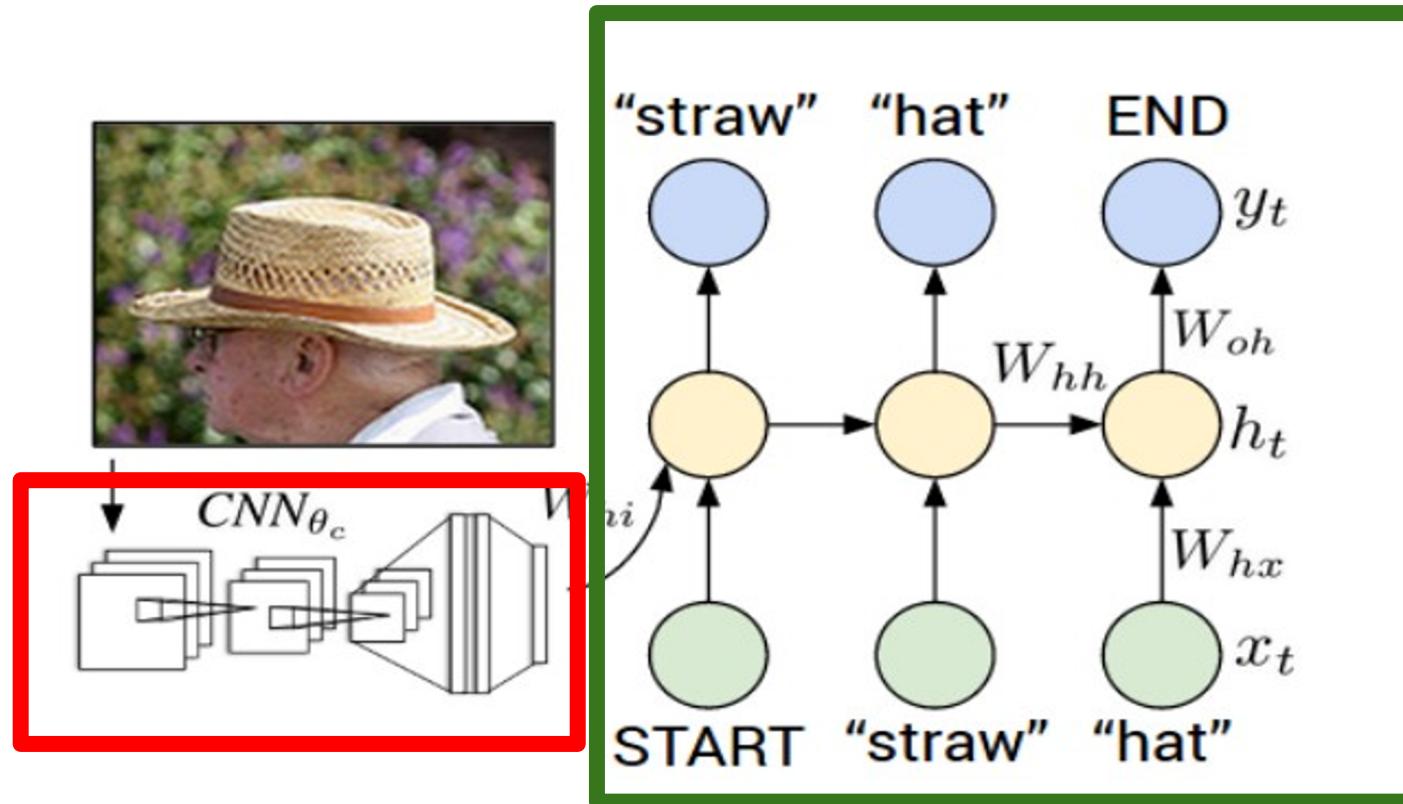
Vinyals et al, "Show and Tell: A Neural Image Caption Generator", CVPR 2015

Donahue et al, "Long-term Recurrent Convolutional Networks for Visual Recognition and Description", CVPR 2015

Chen and Zitnick, "Learning a Recurrent Visual Representation for Image Caption Generation", CVPR 2015

Figure from Karpathy et al, "Deep Visual-Semantic Alignments for Generating Image Descriptions", CVPR 2015

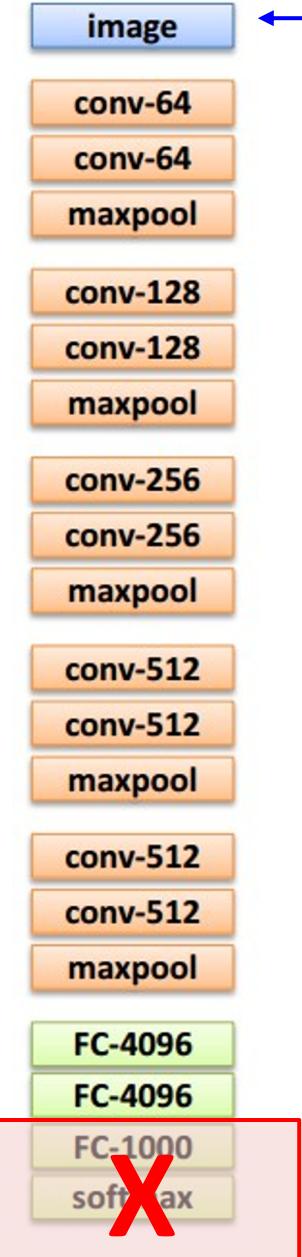
Example: Image Captioning



Recurrent
Neural
Network

Convolutional Neural Network

Figure from Karpathy et al, "Deep Visual-Semantic Alignments for Generating Image Descriptions", CVPR 2015



Transfer learning: Take
CNN trained on ImageNet,
chop off last layer



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



Before:

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

Now:

$$\tanh(W_{hh}h_{t-1} + W_{xh}x_t + W_{ih}v + b_h)$$

W_{ih}

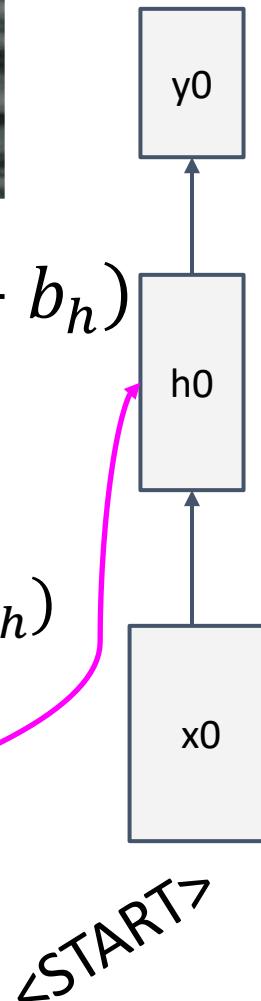


image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
conv-256
conv-256
maxpool
conv-512
conv-512
maxpool
conv-512
conv-512
maxpool
FC-4096
FC-4096

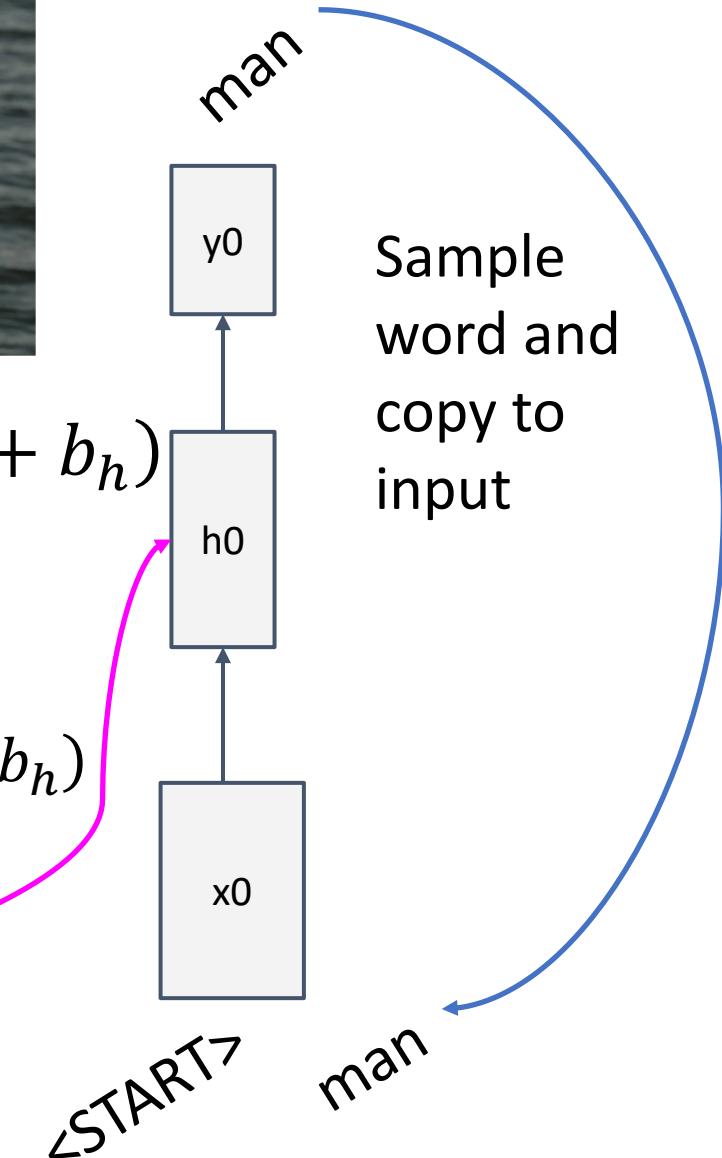


Before:

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

Now:

$$\tanh(W_{hh}h_{t-1} + W_{xh}x_t + W_{ih}v + b_h)$$



Sample
word and
copy to
input



Before:

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

Now:

$$\tanh(\mathbf{W}_{hh}h_{t-1} + \mathbf{W}_{xh}x_t + \mathbf{W}_{ih}v + b_h)$$

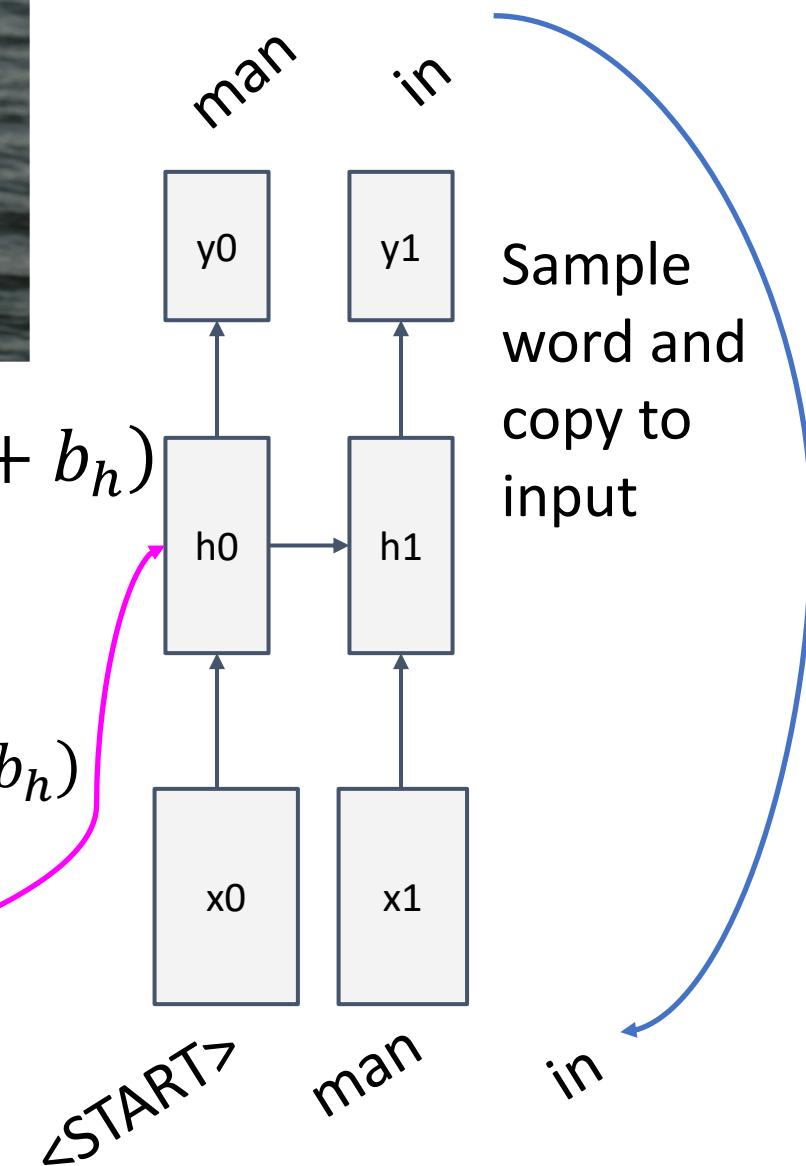


image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
conv-256
conv-256
maxpool
conv-512
conv-512
maxpool
conv-512
conv-512
maxpool
FC-4096
FC-4096



Before:

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

Now:

$$\tanh(W_{hh}h_{t-1} + W_{xh}x_t + W_{ih}v + b_h)$$

W_{ih}

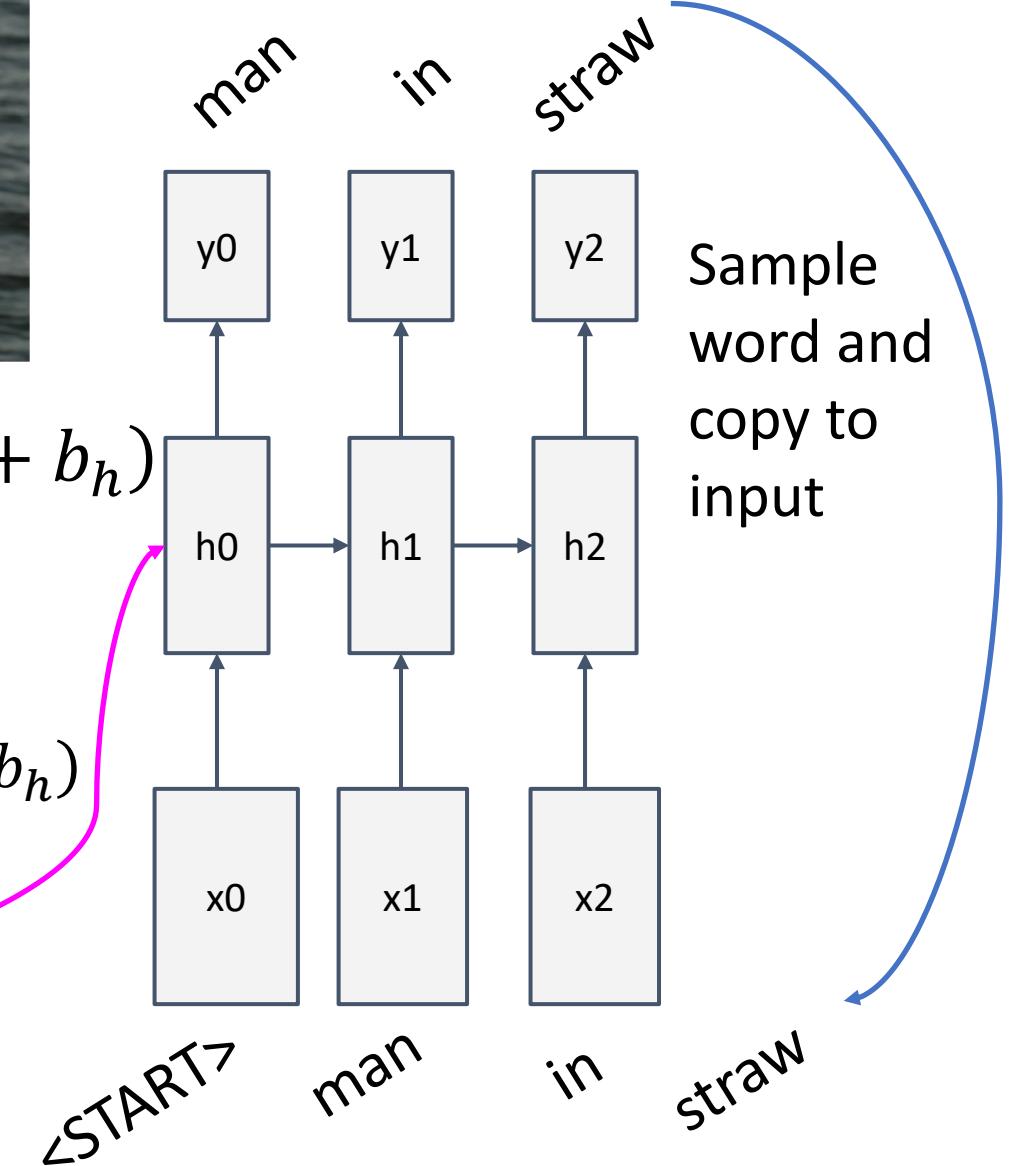


image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
conv-256
conv-256
maxpool
conv-512
conv-512
maxpool
conv-512
conv-512
maxpool
FC-4096
FC-4096



Before:

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

Now:

$$\tanh(W_{hh}h_{t-1} + W_{xh}x_t + W_{ih}v + b_h)$$

W_{ih}

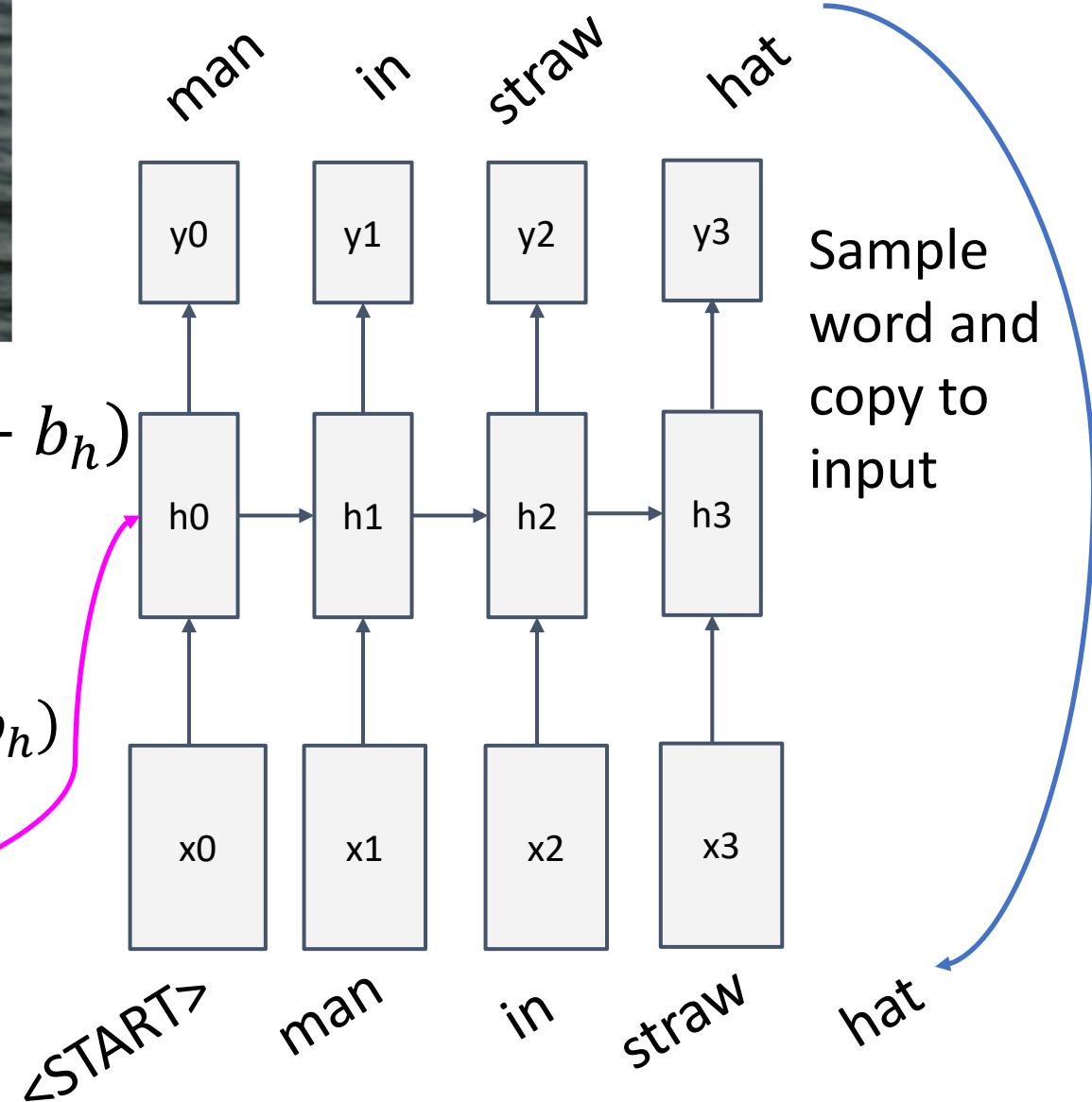


image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
conv-256
conv-256
maxpool
conv-512
conv-512
maxpool
conv-512
conv-512
maxpool
FC-4096
FC-4096



Before:

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

Now:

$$\tanh(W_{hh}h_{t-1} + W_{xh}x_t + W_{ih}v + b_h)$$

W_{ih}

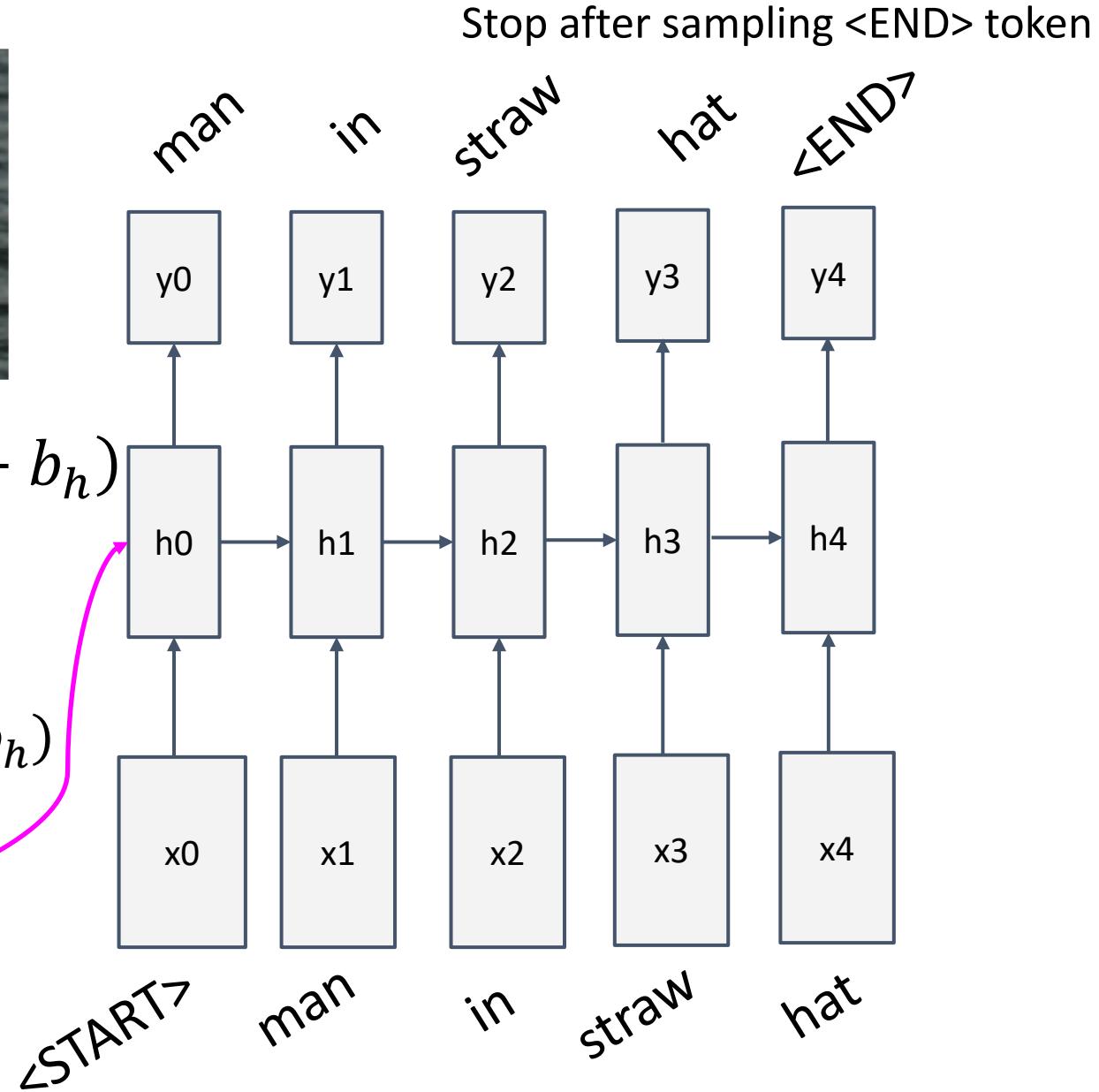


Image Captioning: Example Results



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

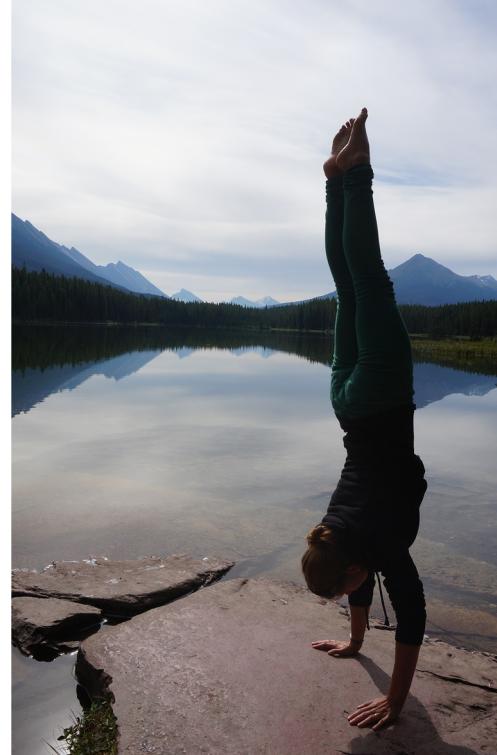
Image Captioning: Failure Cases



*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*

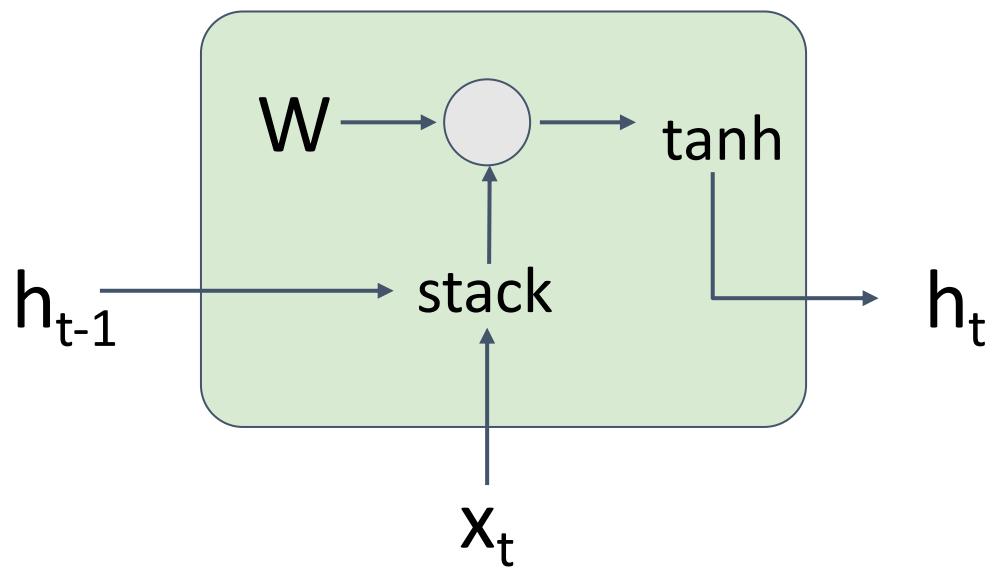


*A bird is perched on a
tree branch*



*A man in a
baseball uniform
throwing a ball*

Vanilla RNN Gradient Flow

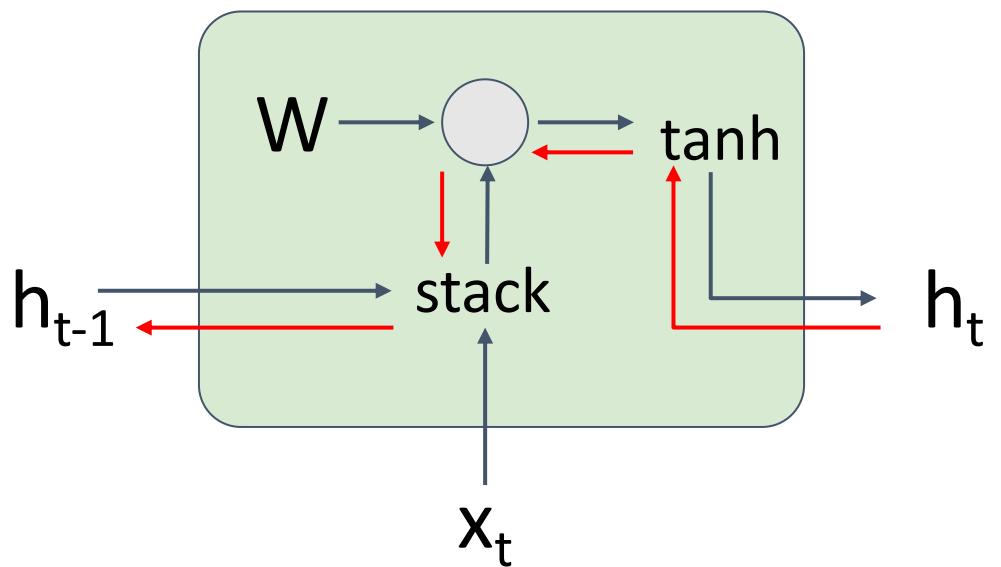


$$\begin{aligned} h_t &= \tanh(W_{hh}h_{t-1} + W_{xh}x_t + b_h) \\ &= \tanh\left(\begin{pmatrix} W_{hh} & W_{hx} \end{pmatrix} \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} + b_h\right) \\ &= \tanh\left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} + b_h\right) \end{aligned}$$

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

Vanilla RNN Gradient Flow

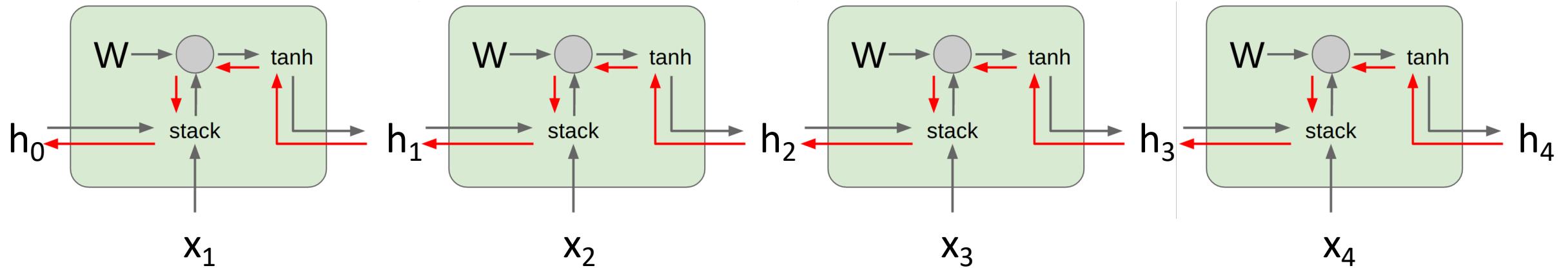
Backpropagation from
 h_t to h_{t-1} multiplies by W
(actually W_{hh}^T)



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

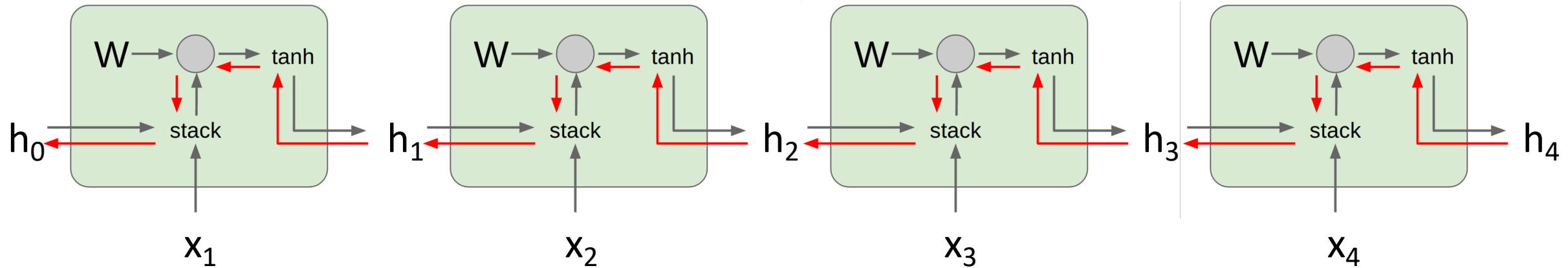
Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013

Vanilla RNN Gradient Flow



Computing gradient of
 h_0 involves many
factors of W
(and repeated tanh)

Vanilla RNN Gradient Flow

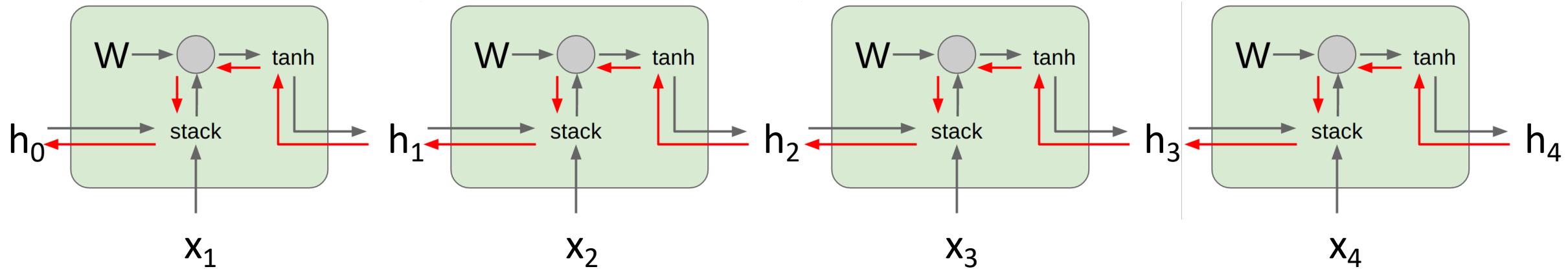


Computing gradient of h_0 involves many factors of W (and repeated tanh)

Largest singular value > 1 :
Exploding gradients

Largest singular value < 1 :
Vanishing gradients

Vanilla RNN Gradient Flow



Computing gradient of h_0 involves many factors of W (and repeated \tanh)

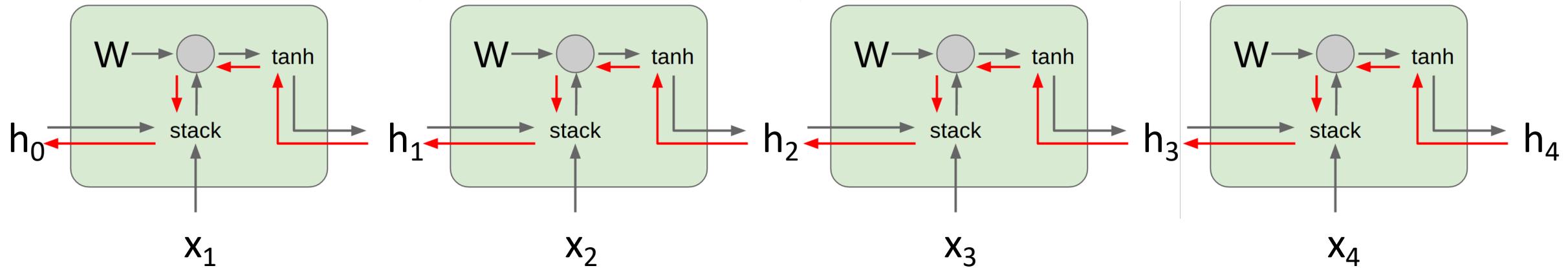
Largest singular value > 1 :
Exploding gradients

Largest singular value < 1 :
Vanishing 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)
```

Vanilla RNN Gradient Flow



Computing gradient of h_0 involves many factors of W (and repeated tanh)

Largest singular value > 1 :
Exploding gradients

Largest singular value < 1 :
Vanishing gradients

→ **Change RNN architecture!**

Long Short Term Memory (LSTM)

Vanilla RNN

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

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

Long Short Term Memory (LSTM)

Vanilla RNN

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

LSTM

$$\begin{pmatrix} i_t \\ f_t \\ o_t \\ g_t \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} \left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} + b_h \right)$$
$$c_t = f_t \odot c_{t-1} + i_t \odot g_t$$
$$h_t = o_t \odot \tanh(c_t)$$

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

Long Short Term Memory (LSTM)

Vanilla RNN

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

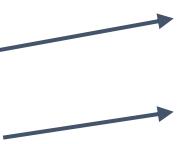
LSTM

$$\begin{pmatrix} i_t \\ f_t \\ o_t \\ g_t \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} \left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} + b_h \right)$$
$$c_t = f_t \odot c_{t-1} + i_t \odot g_t$$
$$h_t = o_t \odot \tanh(c_t)$$

Two vectors at each timestep:

Cell state: $c_t \in \mathbb{R}^H$

Hidden state: $h_t \in \mathbb{R}^H$



Long Short Term Memory (LSTM)

Vanilla RNN

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

Compute four “gates” per timestep:

$$\text{Input gate: } i_t \in \mathbb{R}^H$$

$$\text{Forget gate: } f_t \in \mathbb{R}^H$$

$$\text{Output gate: } o_t \in \mathbb{R}^H$$

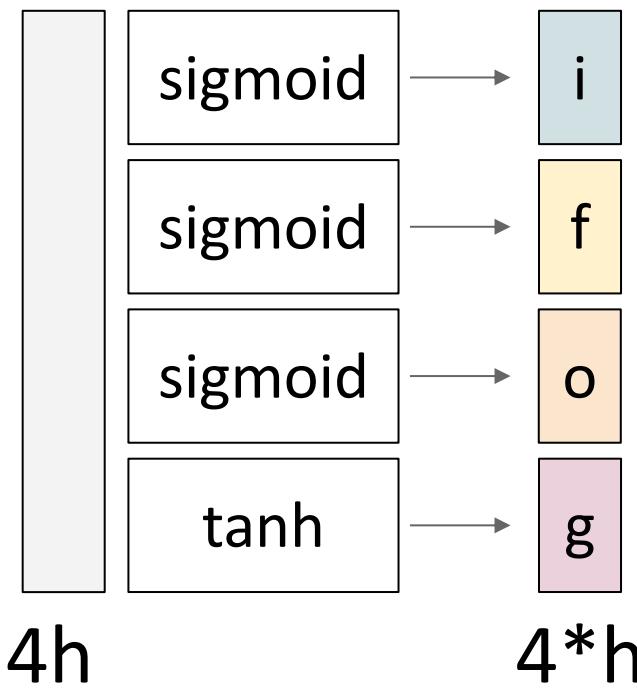
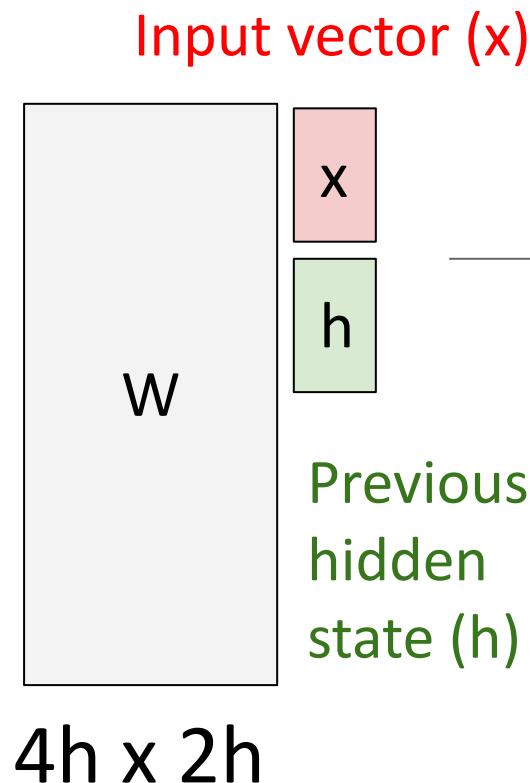
$$\text{“Gate?” gate: } g_t \in \mathbb{R}^H$$

LSTM

$$\begin{pmatrix} i_t \\ f_t \\ o_t \\ g_t \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} \left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} + b_h \right)$$
$$c_t = f_t \odot c_{t-1} + i_t \odot g_t$$
$$h_t = o_t \odot \tanh(c_t)$$

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

Long Short Term Memory (LSTM)

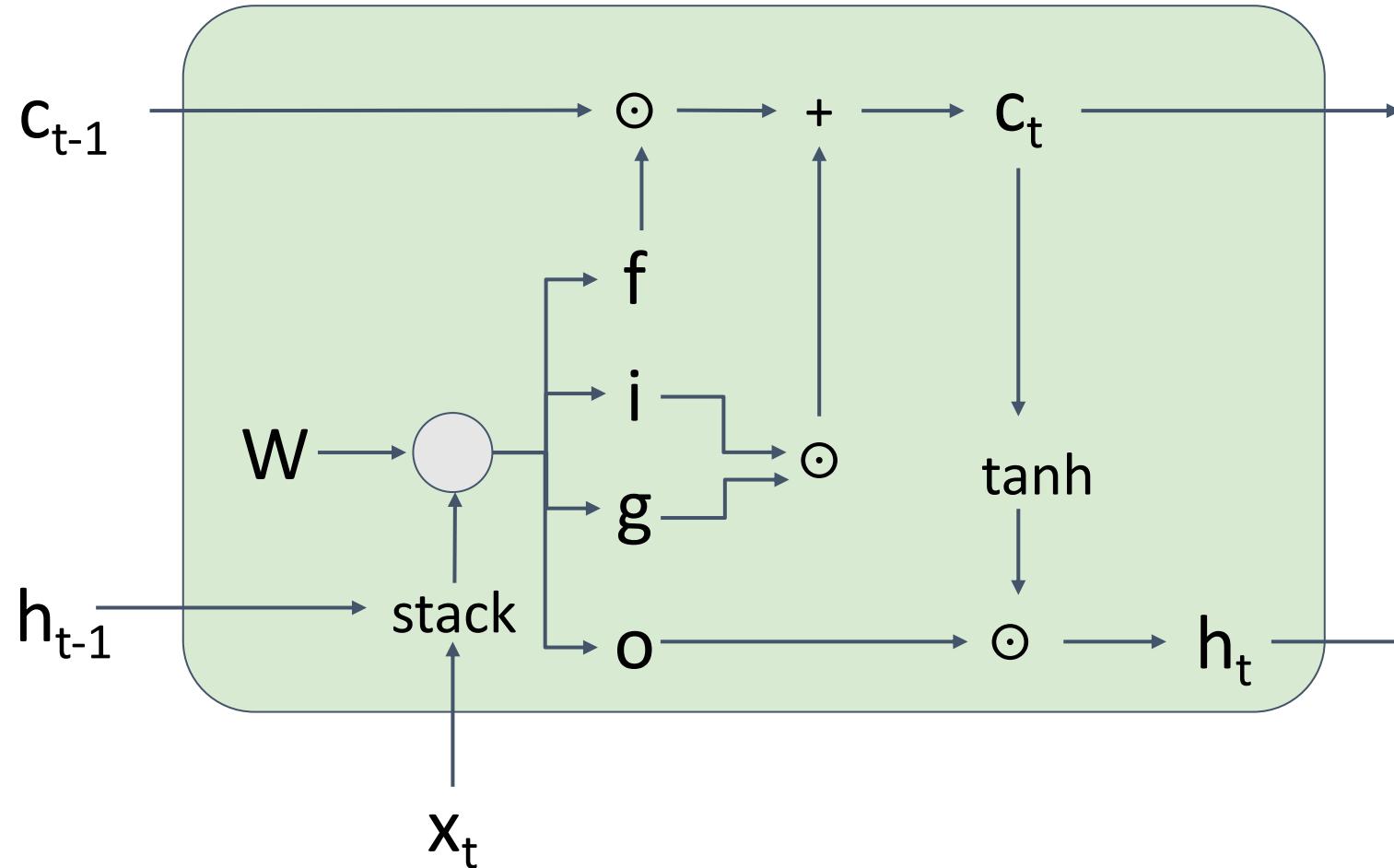


- i: Input gate, whether to write to cell
- f: Forget gate, Whether to erase cell
- o: Output gate, How much to reveal cell
- g: Gate gate (?), How much to write to cell

$$\begin{pmatrix} i_t \\ f_t \\ o_t \\ g_t \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} \left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} + b_h \right)$$

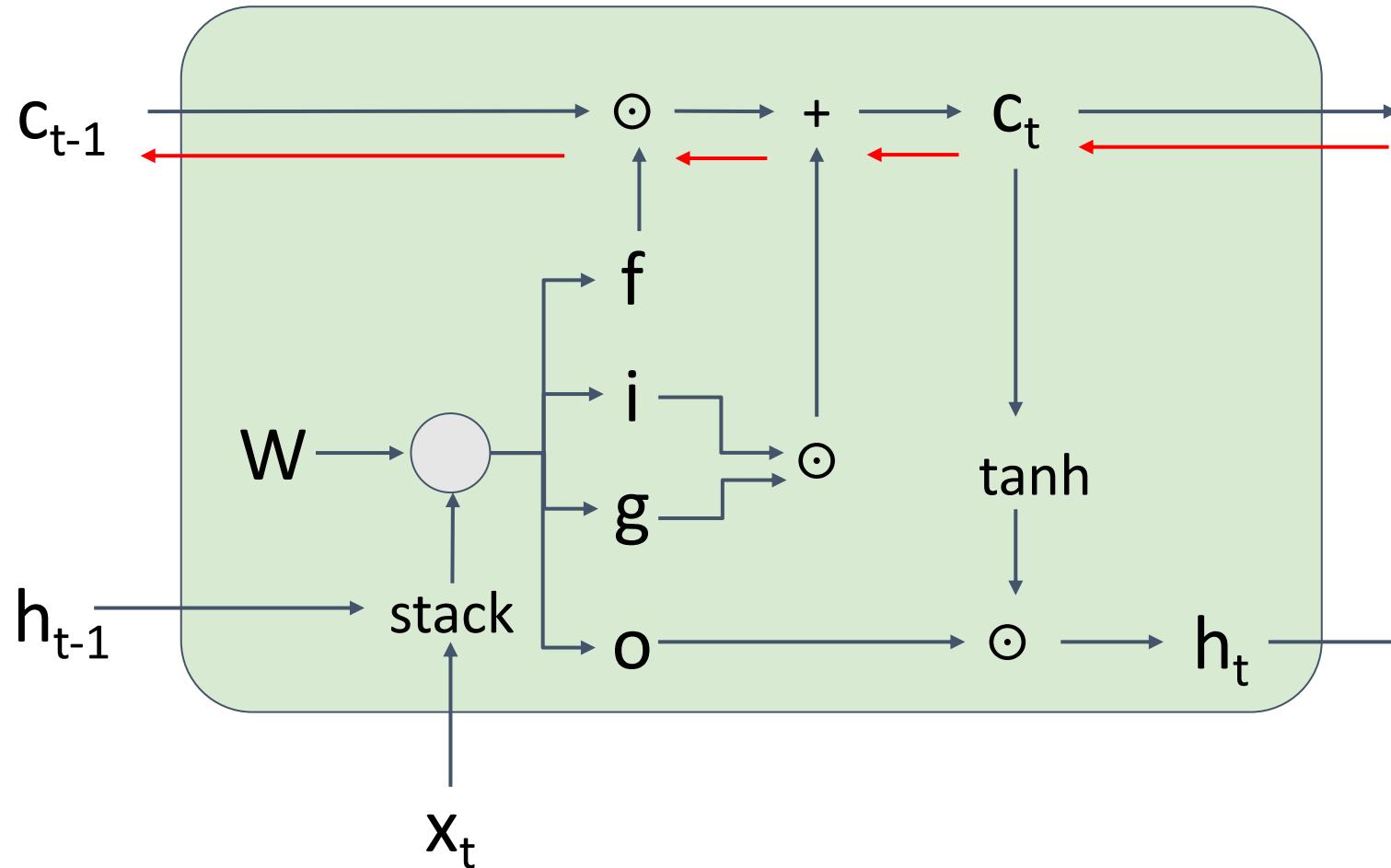
$$c_t = f_t \odot c_{t-1} + i_t \odot g_t$$
$$h_t = o_t \odot \tanh(c_t)$$

Long Short Term Memory (LSTM)



$$\begin{pmatrix} i_t \\ f_t \\ o_t \\ g_t \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} \left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} + b_h \right)$$
$$c_t = f_t \odot c_{t-1} + i_t \odot g_t$$
$$h_t = o_t \odot \tanh(c_t)$$

Long Short Term Memory (LSTM): Gradient Flow

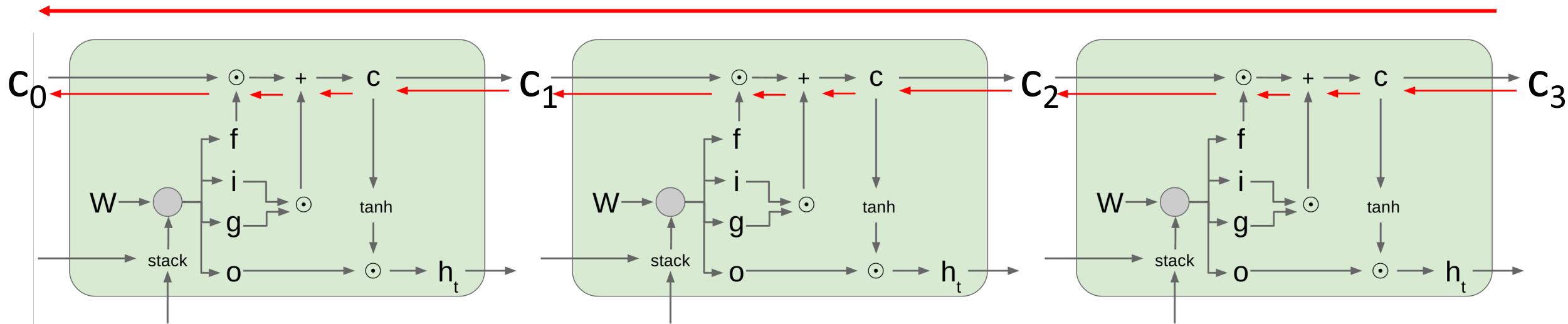


Backpropagation from c_t to c_{t-1} only elementwise multiplication by f , no matrix multiply by W

$$\begin{pmatrix} i_t \\ f_t \\ o_t \\ g_t \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} \left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} + b_h \right)$$
$$c_t = f_t \odot c_{t-1} + i_t \odot g_t$$
$$h_t = o_t \odot \tanh(c_t)$$

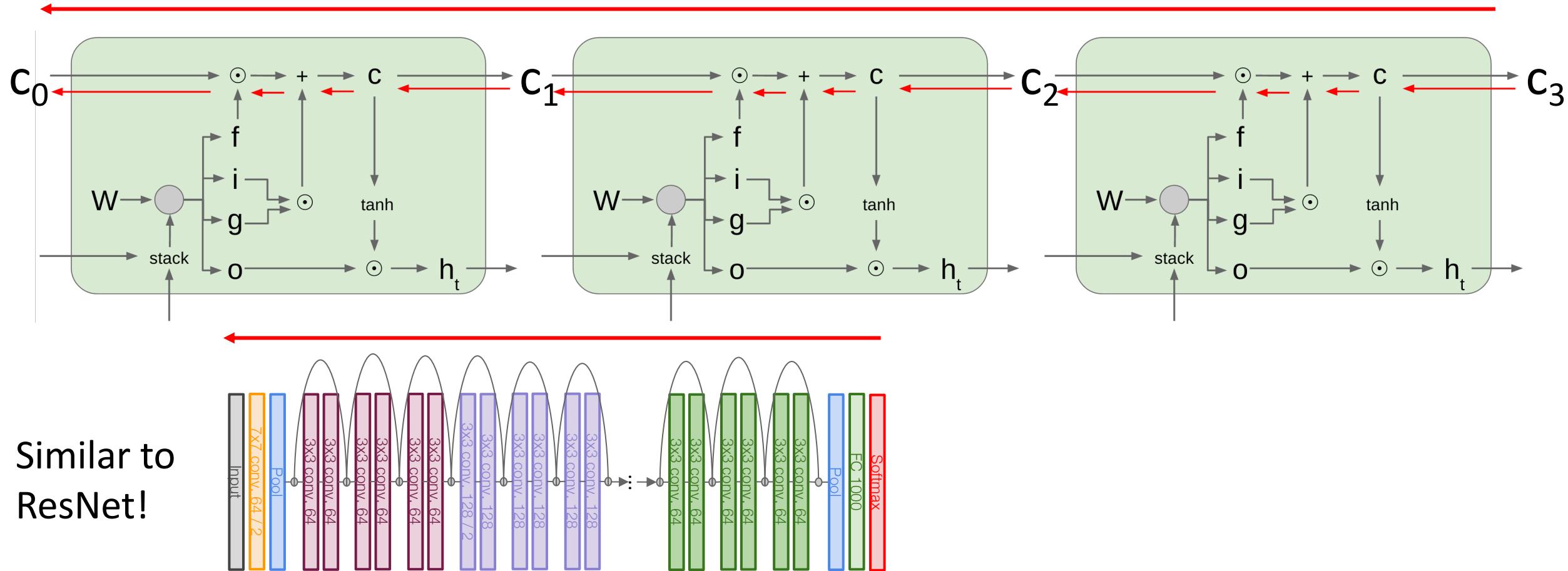
Long Short Term Memory (LSTM): Gradient Flow

Uninterrupted gradient flow!



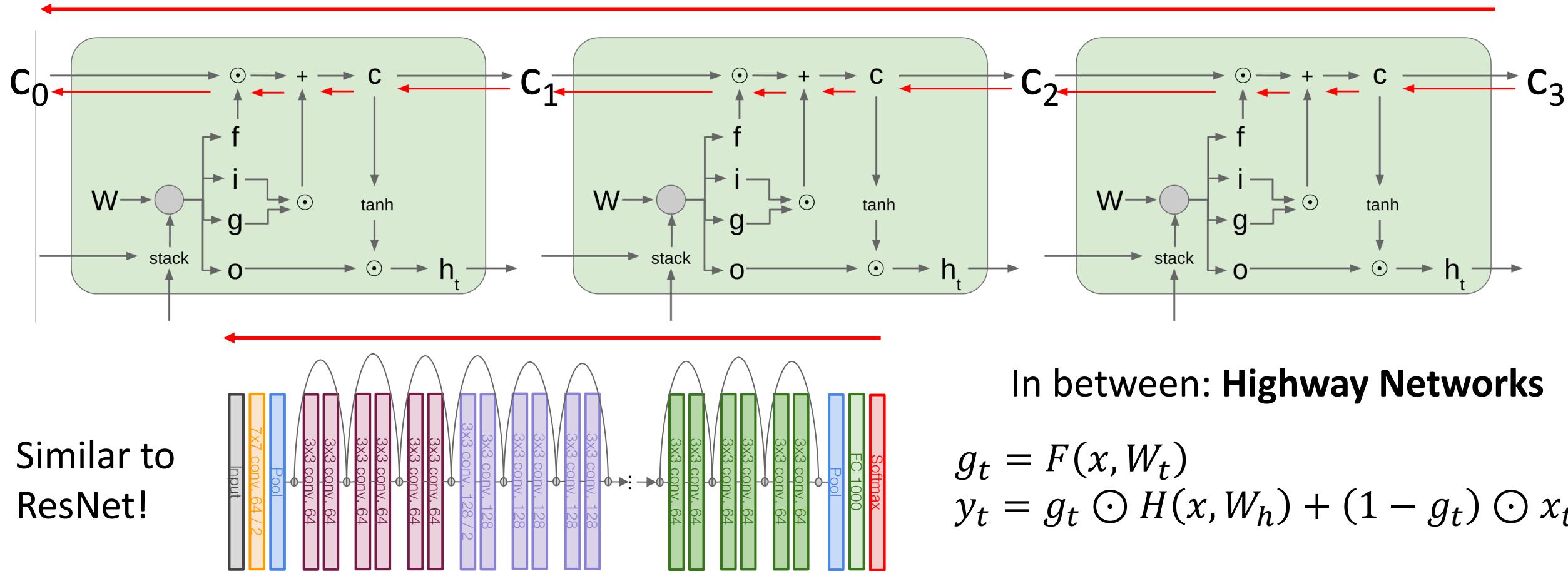
Long Short Term Memory (LSTM): Gradient Flow

Uninterrupted gradient flow!



Long Short Term Memory (LSTM): Gradient Flow

Uninterrupted gradient flow!



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

Single-Layer RNNs

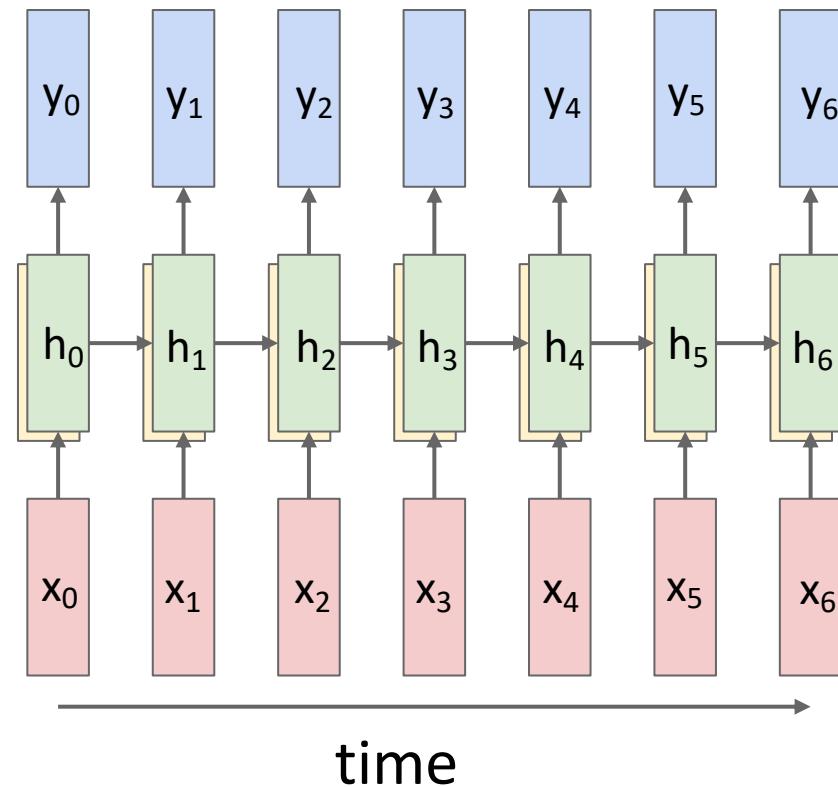
$$h_t = \tanh\left(W\begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} + b_h\right)$$

LSTM:

$$\begin{pmatrix} i_t \\ f_t \\ o_t \\ g_t \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} \left(W\begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} + b_h \right)$$

$$c_t = f_t \odot c_{t-1} + i_t \odot g_t$$

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



Mutilayer RNNs

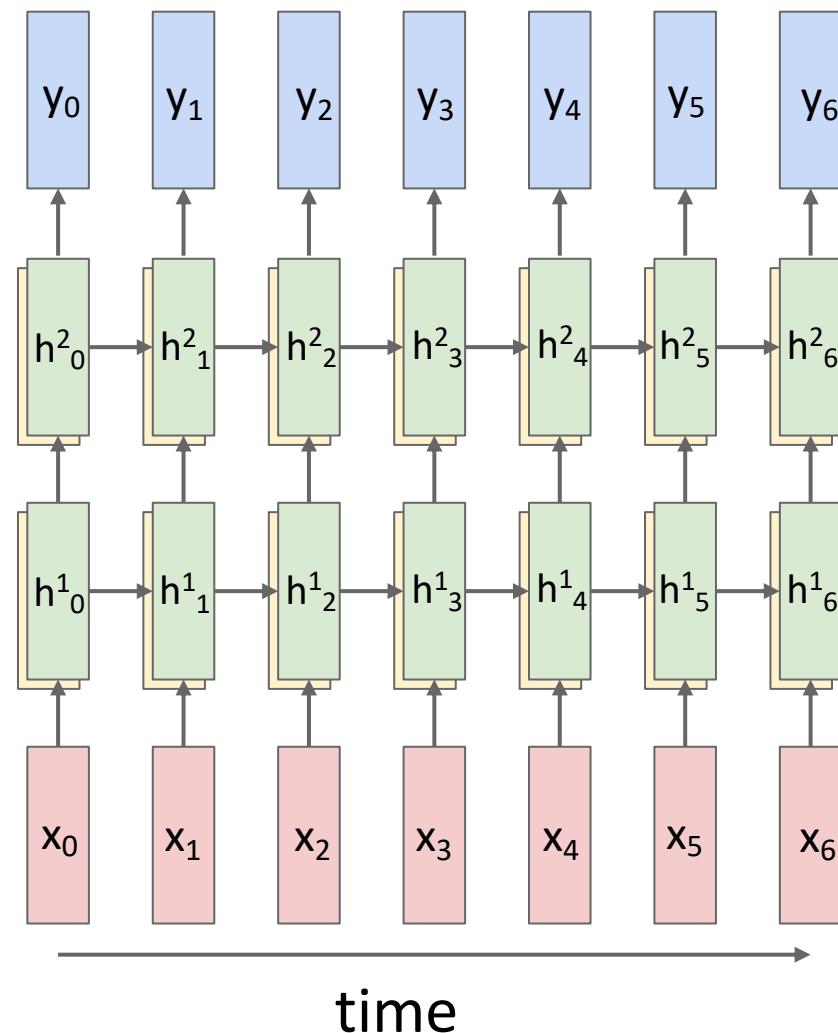
$$h_t^\ell = \tanh \left(W \begin{pmatrix} h_{t-1}^\ell \\ h_{t-1}^{\ell-1} \end{pmatrix} + b_h^\ell \right)$$

LSTM:

$$\begin{pmatrix} i_t^\ell \\ f_t^\ell \\ o_t^\ell \\ g_t^\ell \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} \left(W \begin{pmatrix} h_{t-1}^\ell \\ h_{t-1}^{\ell-1} \end{pmatrix} + b_h^\ell \right)$$
$$c_t^\ell = f_t^\ell \odot c_{t-1}^\ell + i_t^\ell \odot g_t^\ell$$
$$h_t^\ell = o_t^\ell \odot \tanh(c_t^\ell)$$

depth ↑

Two-layer RNN: Pass hidden states from one RNN as inputs to another RNN



Mutilayer RNNs

$$h_t^\ell = \tanh \left(W \begin{pmatrix} h_{t-1}^\ell \\ h_t^{\ell-1} \end{pmatrix} + b_h^\ell \right)$$

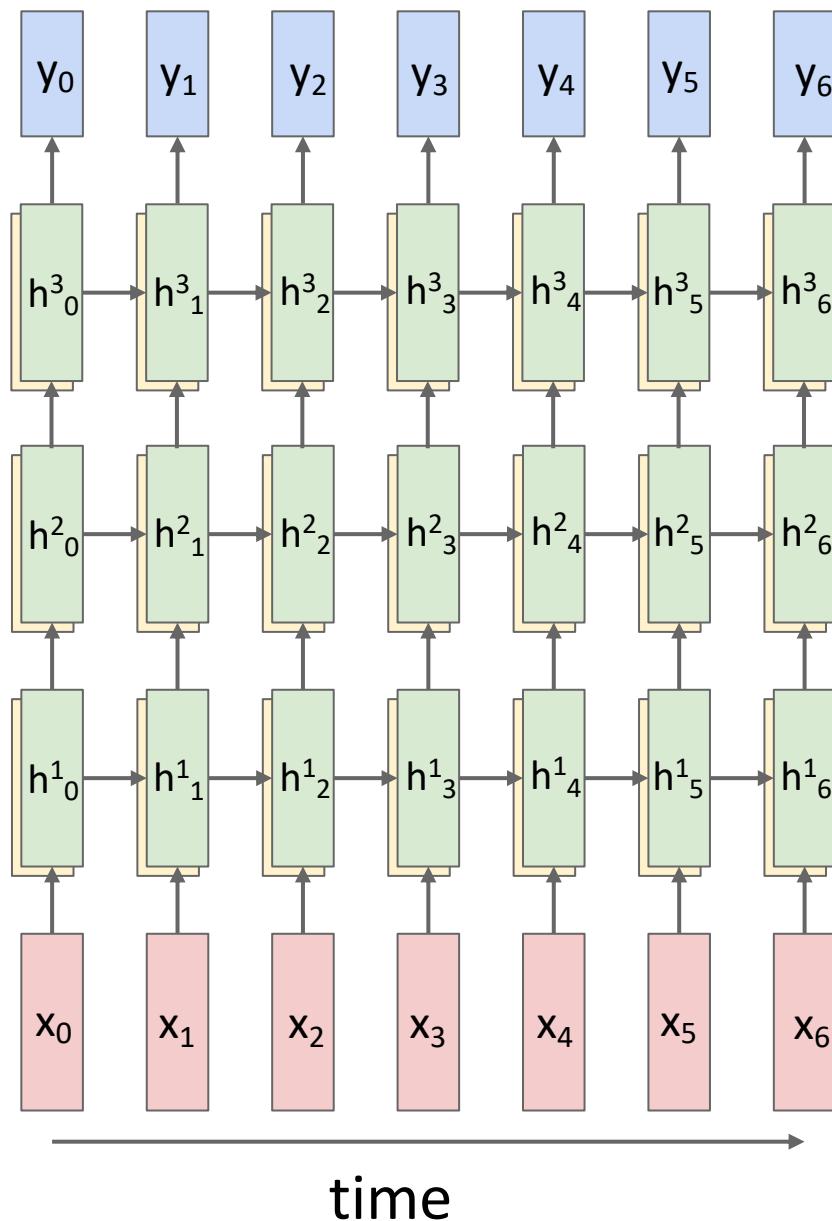
LSTM:

$$\begin{pmatrix} i_t^\ell \\ f_t^\ell \\ o_t^\ell \\ g_t^\ell \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} \left(W \begin{pmatrix} h_{t-1}^\ell \\ h_t^{\ell-1} \end{pmatrix} + b_h^\ell \right)$$

$$c_t^\ell = f_t^\ell \odot c_{t-1}^\ell + i_t^\ell \odot g_t^\ell$$

$$h_t^\ell = o_t^\ell \odot \tanh(c_t^\ell)$$

Three-layer RNN



Other RNN Variants

Gated Recurrent Unit (GRU)

Cho et al “Learning phrase representations using RNN encoder-decoder for statistical machine translation”, 2014

$$r_t = \sigma(W_{xr}x_t + W_{hr}h_{t-1} + b_r)$$

$$z_t = \sigma(W_{xz}x_t + W_{hz}h_{t-1} + b_z)$$

$$\tilde{h}_t = \tanh(W_{xh}x_t + W_{hh}(r_t \odot h_{t-1}) + b_h)$$

$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot \tilde{h}_t$$

Other RNN Variants

Gated Recurrent Unit (GRU)

Cho et al “Learning phrase representations using RNN encoder-decoder for statistical machine translation”, 2014

$$r_t = \sigma(W_{xr}x_t + W_{hr}h_{t-1} + b_r)$$

$$z_t = \sigma(W_{xz}x_t + W_{hz}h_{t-1} + b_z)$$

$$\tilde{h}_t = \tanh(W_{xh}x_t + W_{hh}(r_t \odot h_{t-1}) + b_h)$$

$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot \tilde{h}_t$$

10,000 architectures with evolutionary search:
Jozefowicz et al, “An empirical exploration of recurrent network architectures”, ICML 2015

MUT1:

$$z = \text{sigm}(W_{xz}x_t + b_z)$$

$$r = \text{sigm}(W_{xr}x_t + W_{hr}h_t + b_r)$$

$$\begin{aligned} h_{t+1} = & \tanh(W_{hh}(r \odot h_t) + \tanh(x_t) + b_h) \odot z \\ & + h_t \odot (1 - z) \end{aligned}$$

MUT2:

$$z = \text{sigm}(W_{xz}x_t + W_{hz}h_t + b_z)$$

$$r = \text{sigm}(x_t + W_{hr}h_t + b_r)$$

$$\begin{aligned} h_{t+1} = & \tanh(W_{hh}(r \odot h_t) + W_{xh}x_t + b_h) \odot z \\ & + h_t \odot (1 - z) \end{aligned}$$

MUT3:

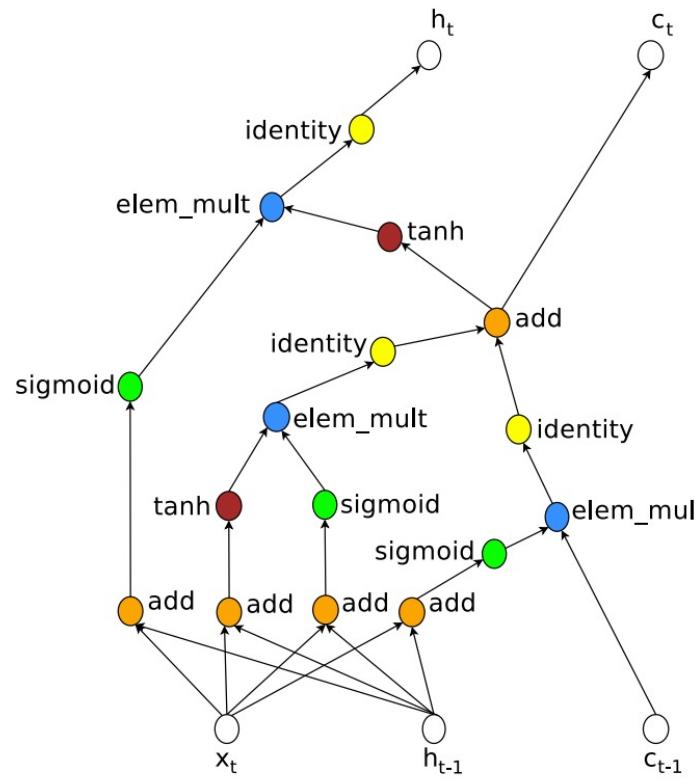
$$z = \text{sigm}(W_{xz}x_t + W_{hz}\tanh(h_t) + b_z)$$

$$r = \text{sigm}(W_{xr}x_t + W_{hr}h_t + b_r)$$

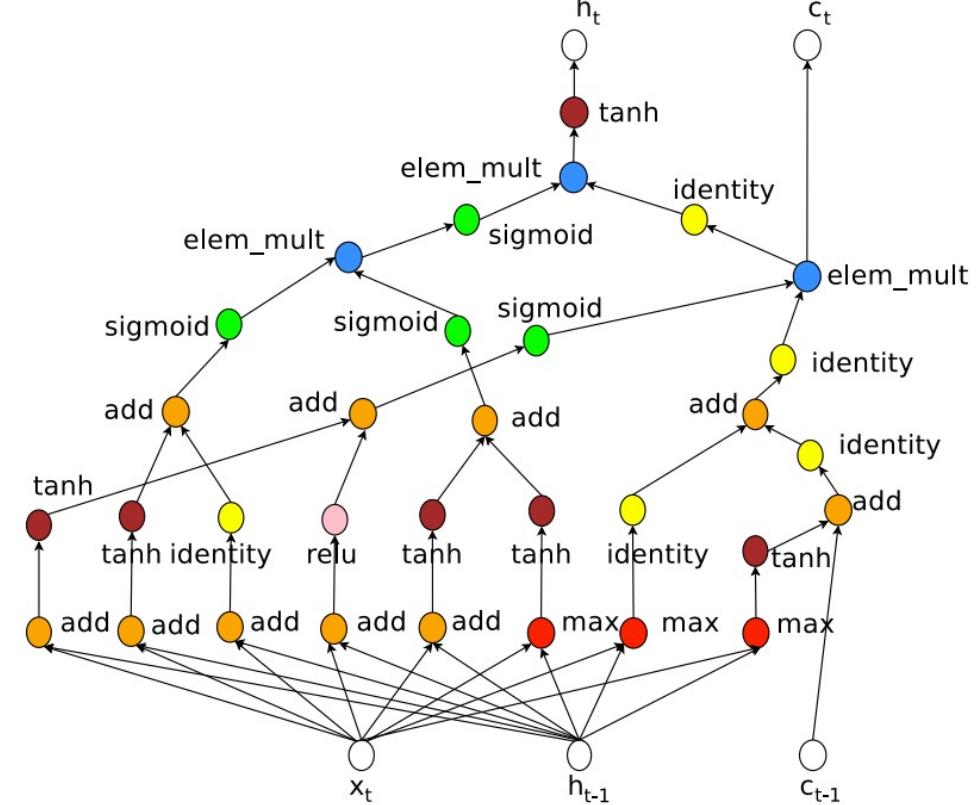
$$\begin{aligned} h_{t+1} = & \tanh(W_{hh}(r \odot h_t) + W_{xh}x_t + b_h) \odot z \\ & + h_t \odot (1 - z) \end{aligned}$$

RNN Architectures: Neural Architecture Search

LSTM



Learned Architecture



Zoph and Le, "Neural Architecture Search with Reinforcement Learning", ICLR 2017

Summary

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

Next Time: Attention