

Lecture 10:

Recurrent Neural Networks

Administrative

- Project proposal grades should be out by now
 - If concerned, come to my office hours.

Administrative

- A2 is due Friday April 30th, 11:59pm

Administrative: Midterm

Covers material through **Lecture 10** (Thu April 29).

- Tues, **May 4** and is worth **15%** of your grade.
- available for **24 hours** on Gradescope from May 4, **12PM PDT** to May 5, 11:59 AM PDT.
- **3-hour** consecutive timeframe
- Exam will be designed for **1.5 hours**.
- Open book and open internet but no collaboration
- Only make private posts during those 24 hours

Administrative

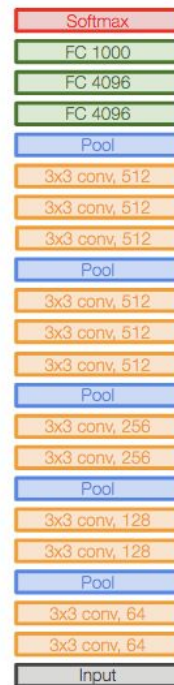
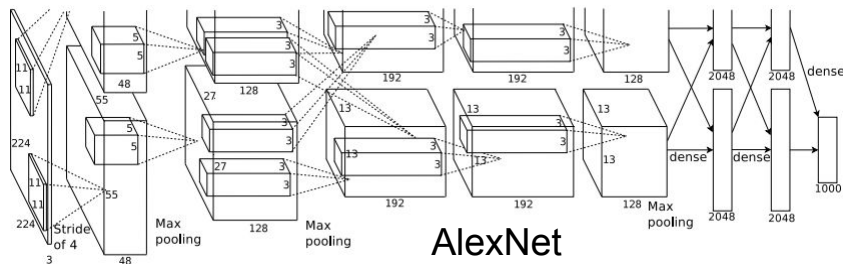
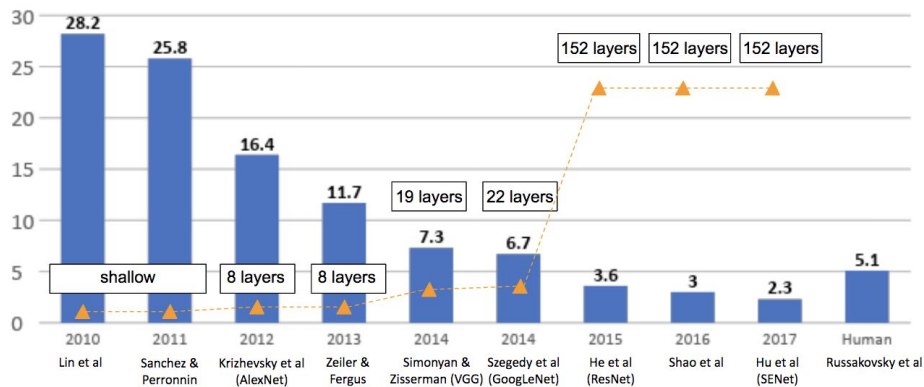
Midterm review session: Fri April 30th discussion section

Sample midterm has been released on Piazza.

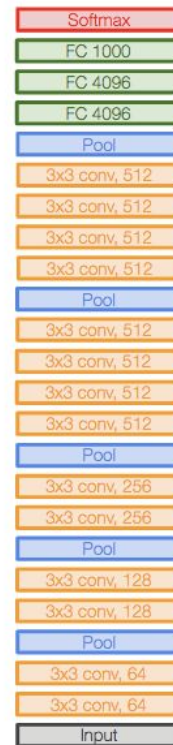
OAE accommodations: If you have not received an email from us, please reach out to the staff mailing list ASAP.

Last Time: CNN Architectures

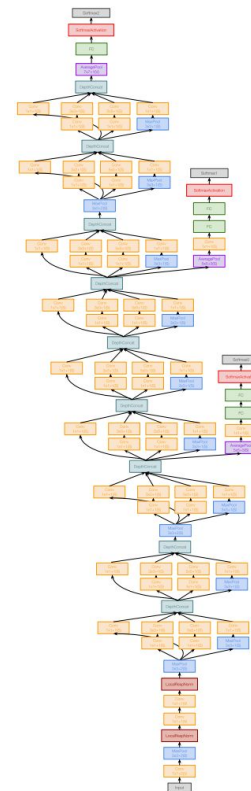
ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



VGG16



VGG19

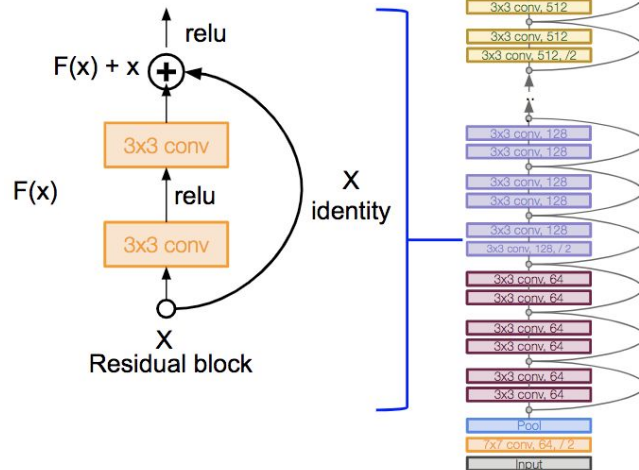
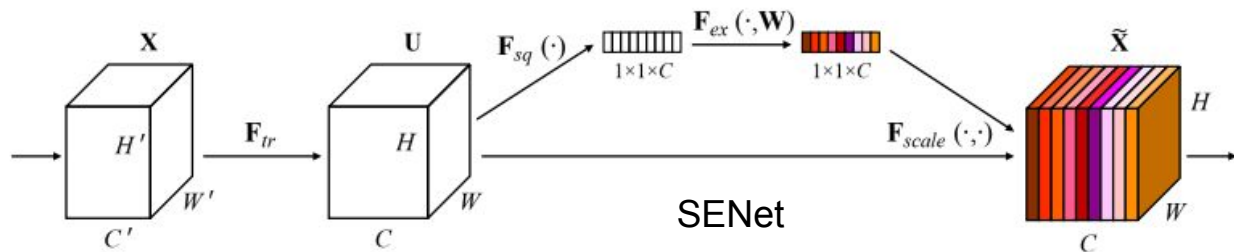
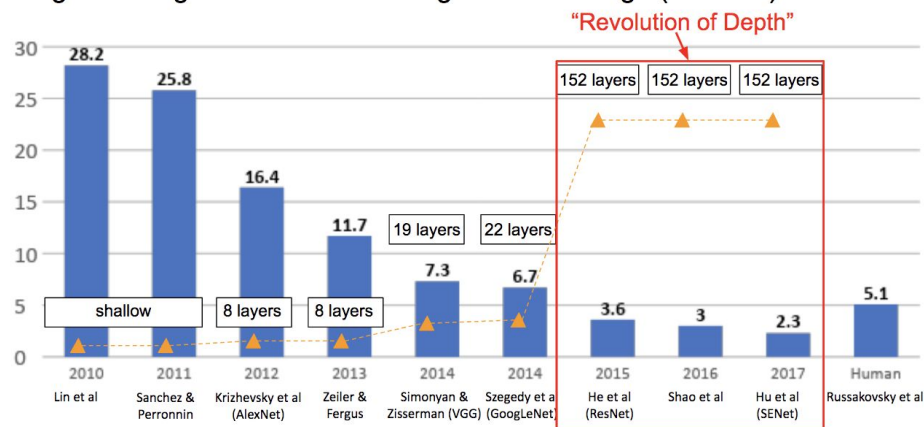


GoogLeNet

Let's jump to lecture 9 - slide 106

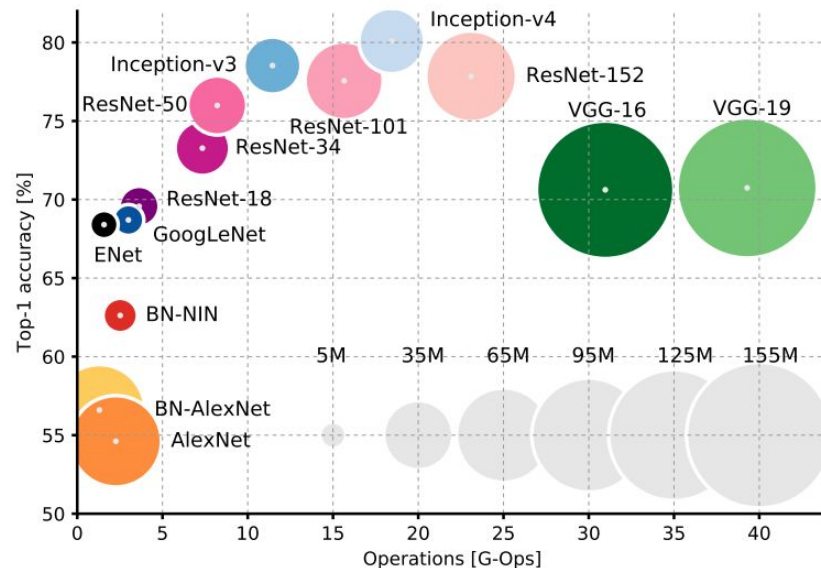
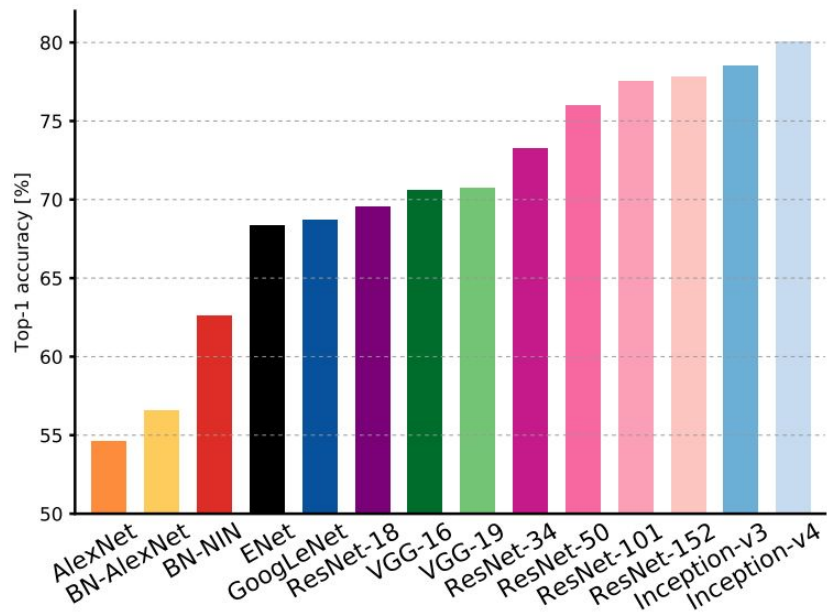
Last Time: CNN Architectures

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



ResNet

Comparing complexity...



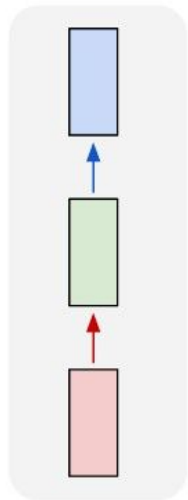
An Analysis of Deep Neural Network Models for Practical Applications, 2017.

Figures copyright Alfredo Canziani, Adam Paszke, Eugenio Culurciello, 2017. Reproduced with permission.

Today: Recurrent Neural Networks

“Vanilla” Neural Network

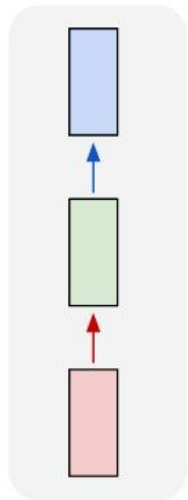
one to one



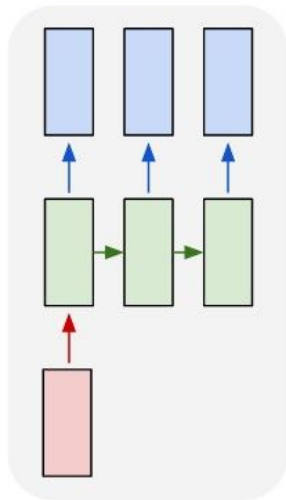
Vanilla Neural Networks

Recurrent Neural Networks: Process Sequences

one to one



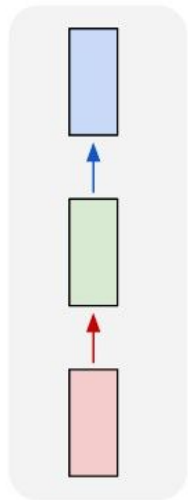
one to many



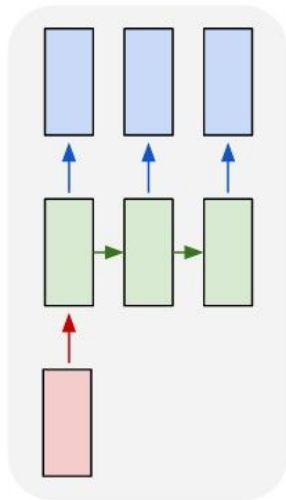
↖ e.g. **Image Captioning**
image -> sequence of words

Recurrent Neural Networks: Process Sequences

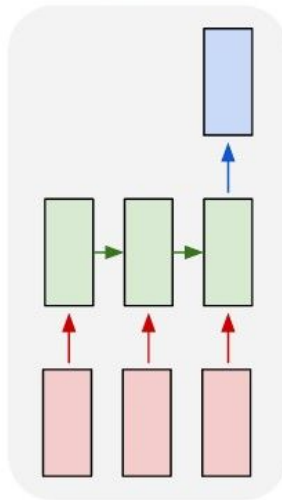
one to one



one to many



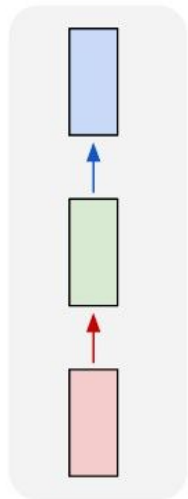
many to one



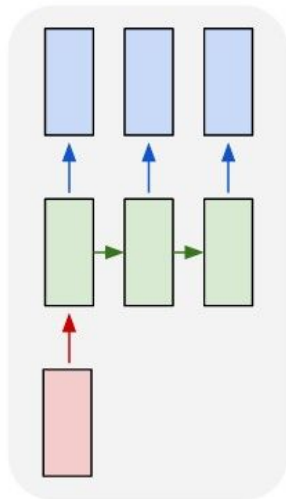
↖ e.g. **action prediction**
sequence of video frames -> action class

Recurrent Neural Networks: Process Sequences

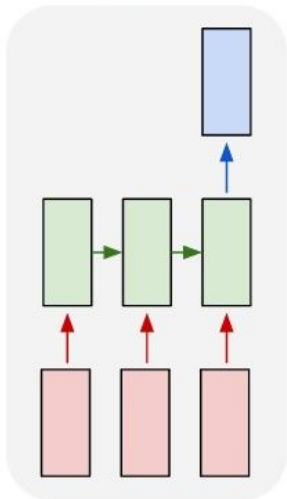
one to one



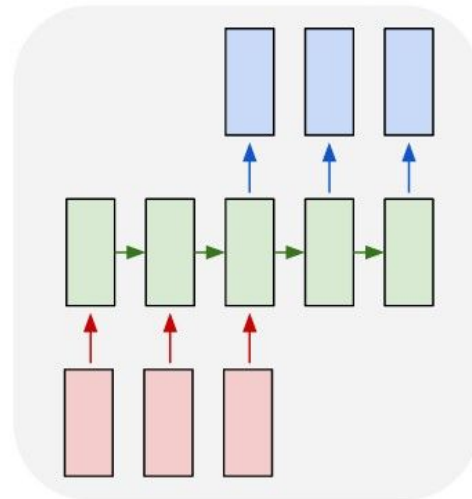
one to many



many to one



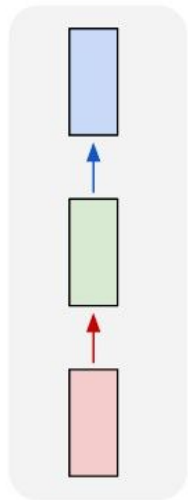
many to many



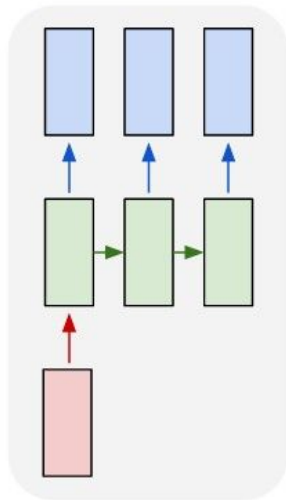
↖ E.g. **Video Captioning**
Sequence of video frames ->
caption

Recurrent Neural Networks: Process Sequences

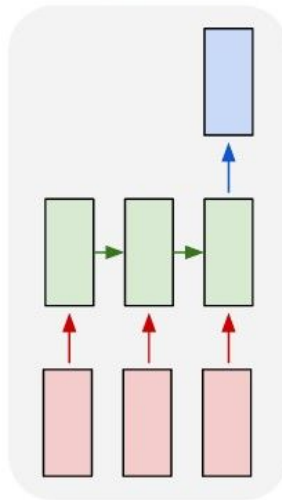
one to one



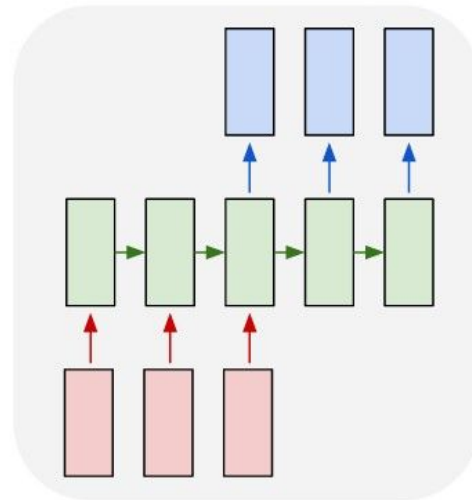
one to many



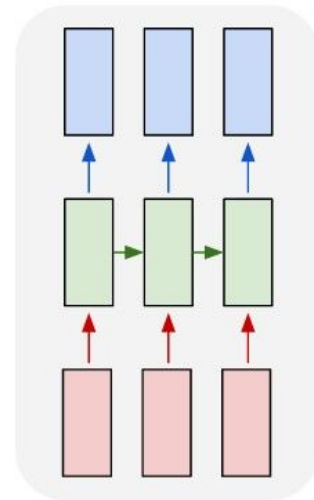
many to one



many to many



many to many



e.g. Video classification on frame level

Sequential Processing of Non-Sequence Data

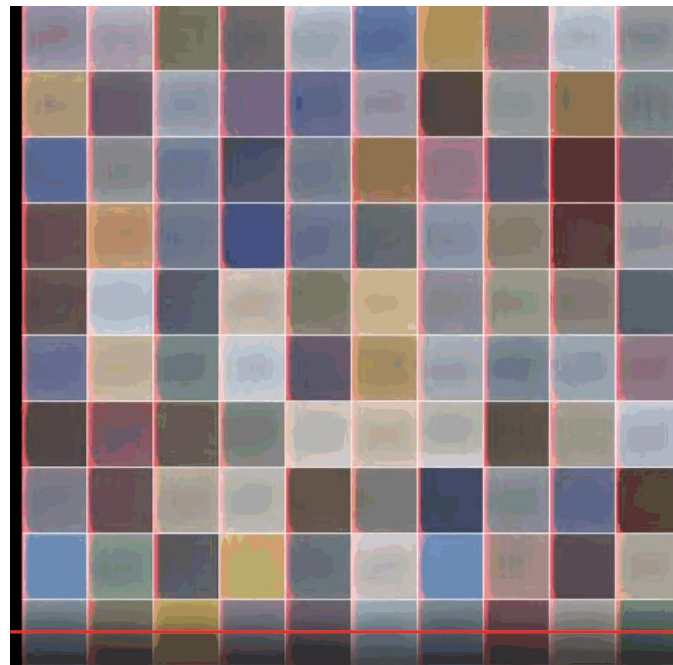
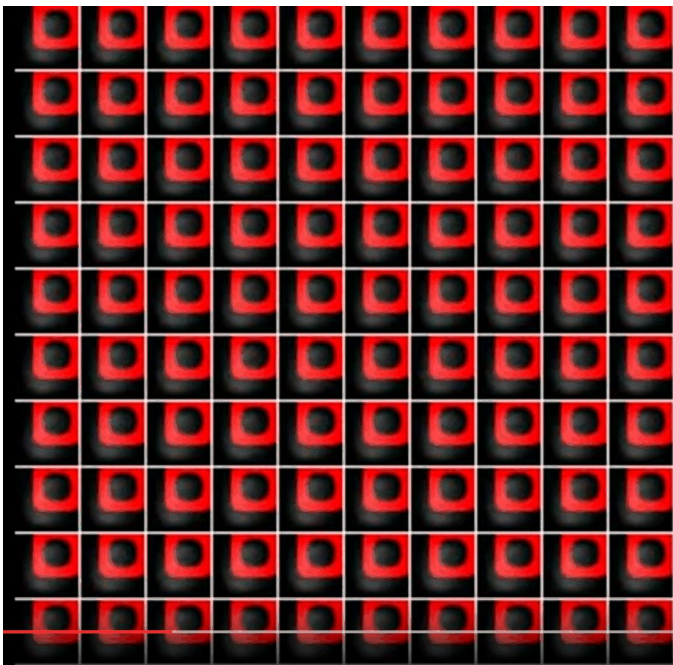
Classify images by taking a series of “glimpses”



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
Figure copyright Karol Gregor, Ivo Danihelka, Alex Graves, Danilo Jimenez Rezende, and Daan Wierstra, 2015. Reproduced with permission.

Sequential Processing of Non-Sequence Data

Generate images one piece at a time!



Gregor et al, "DRAW: A Recurrent neural network For Image Generation", ICML 2015
Figure copyright Karol Gregor, Ivo Danihelka, Alex Graves, Danilo Jimenez Rezende, and Daan Wierstra, 2015. Reproduced with permission.

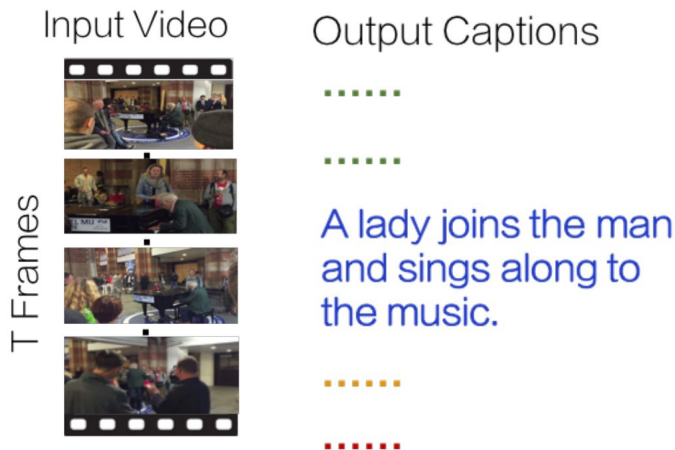
Why existing convnets are insufficient?

Variable sequence length inputs and outputs!

Example task: video captioning

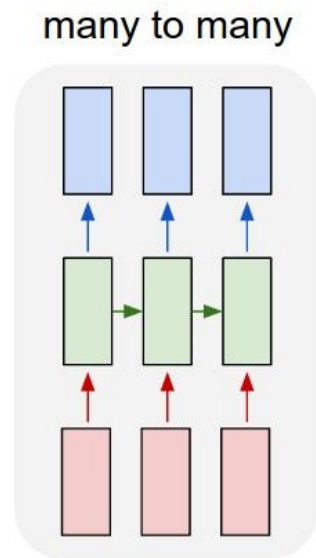
Input video can have variable number of frames

Output captions can be variable length.

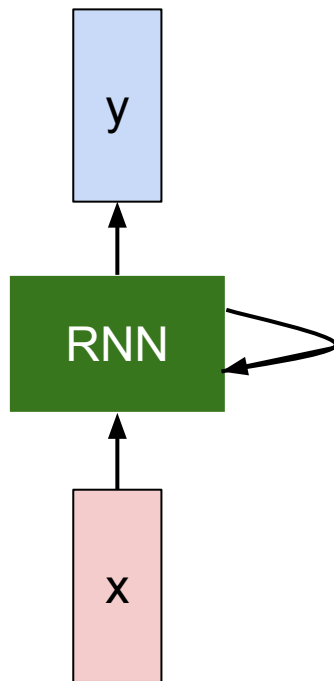


Krishna, Hata, Ren, Fei-Fei, Niebles. Dense captioning Events in Videos. ICCV 2019

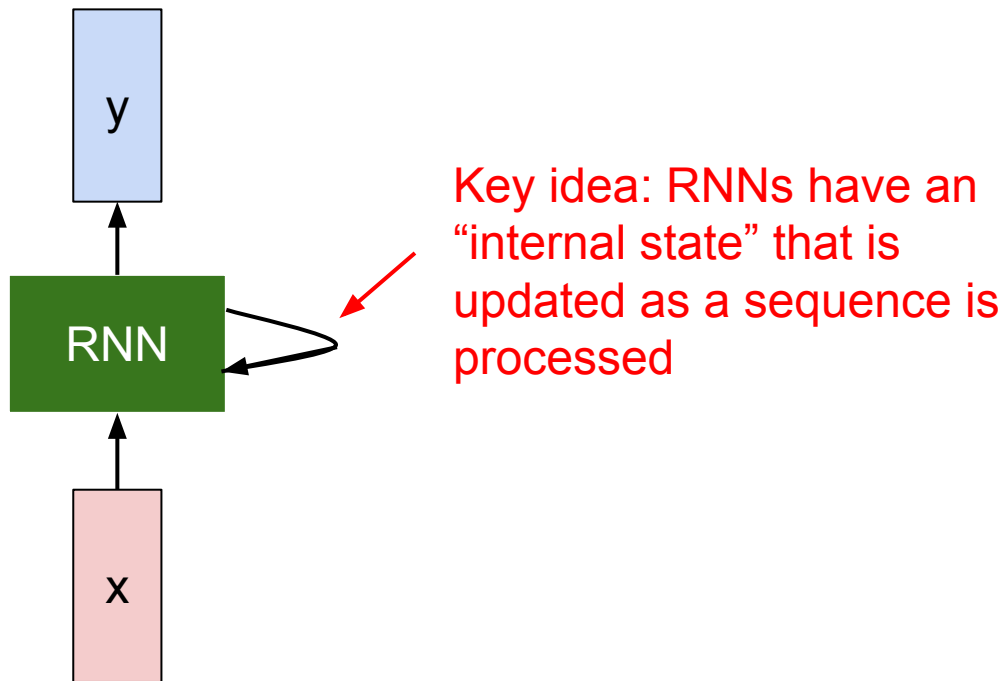
Let's start with a task that takes a variable input and produces an output at every step



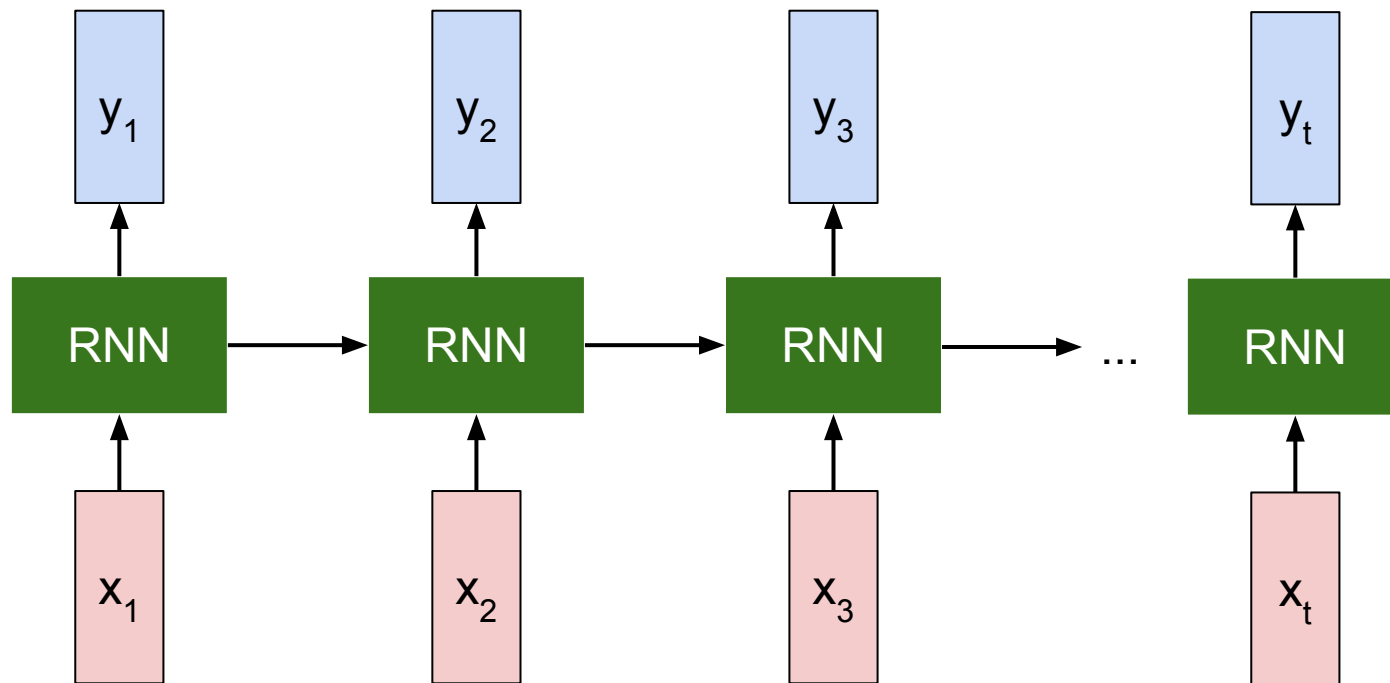
Recurrent Neural Network



Recurrent Neural Network



Unrolled RNN

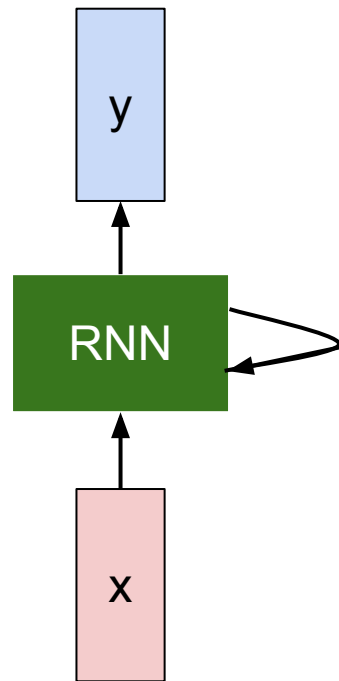


RNN hidden state update

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

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

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



RNN output generation

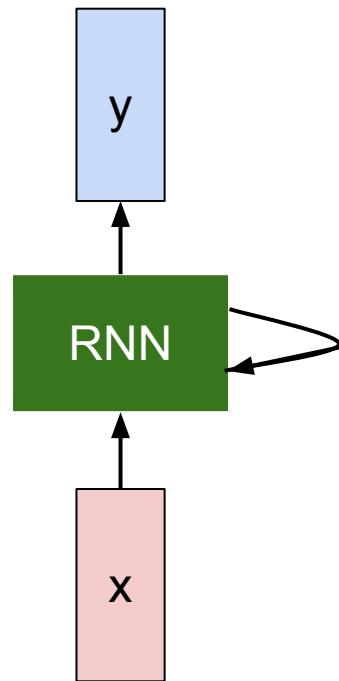
We can process a sequence of vectors $\mathbf{x}$ by applying a **recurrence formula** at every time step:

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

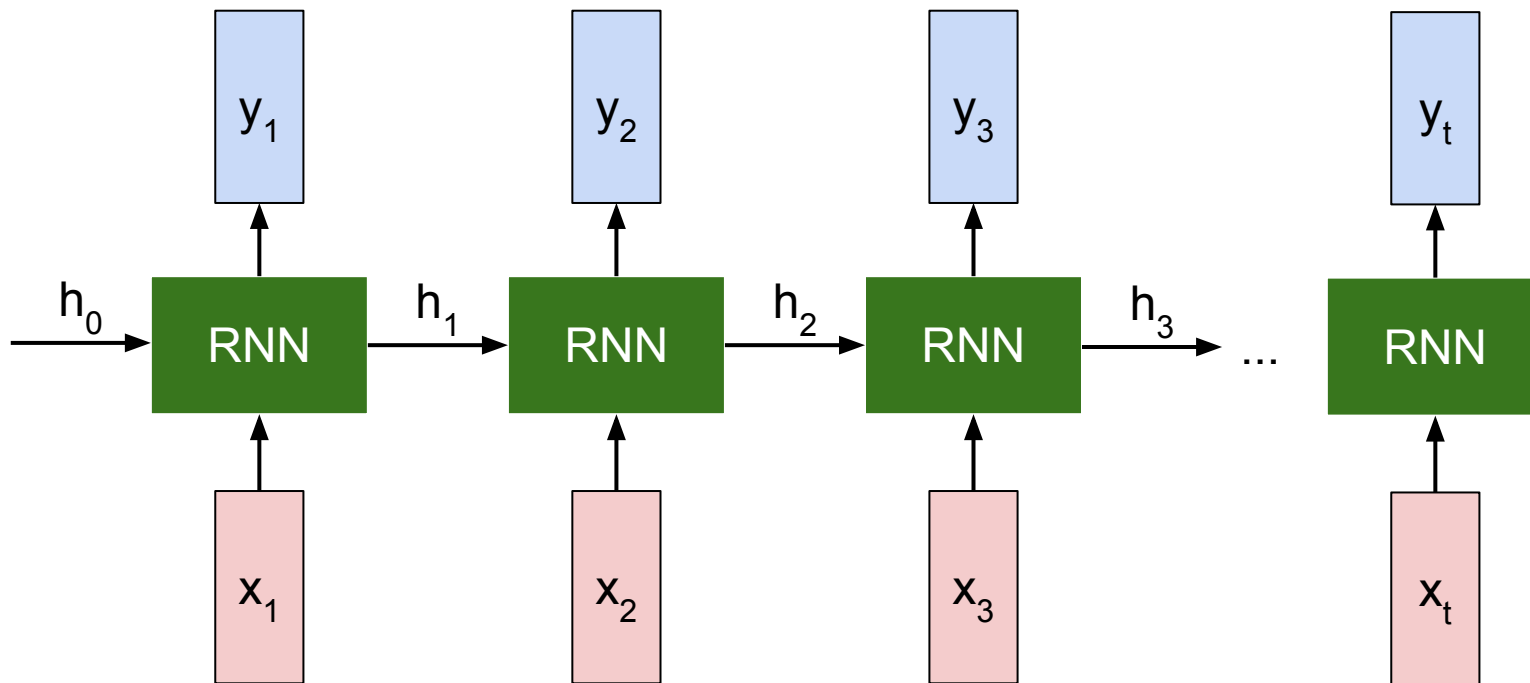
output

another function with parameters W_o

new state



Recurrent Neural Network

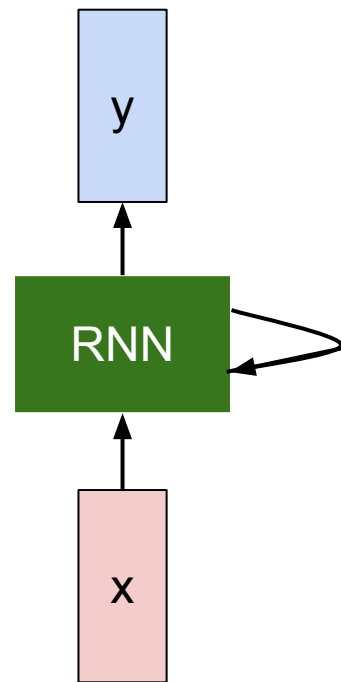


Recurrent Neural Network

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

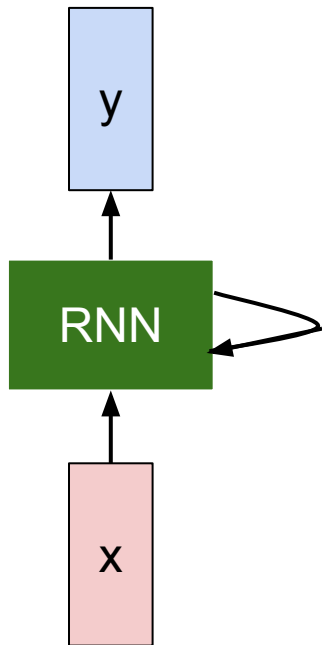
$$h_t = f_W(h_{t-1}, x_t)$$

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



(Simple) Recurrent Neural Network

The state consists of a single “hidden” vector h :



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

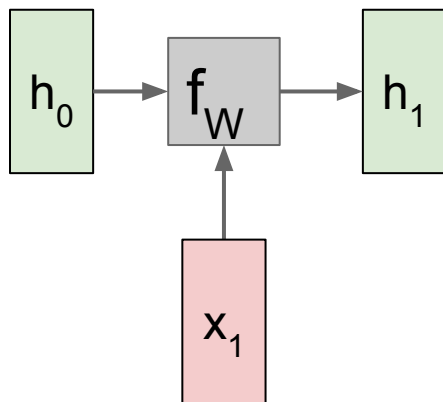


$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

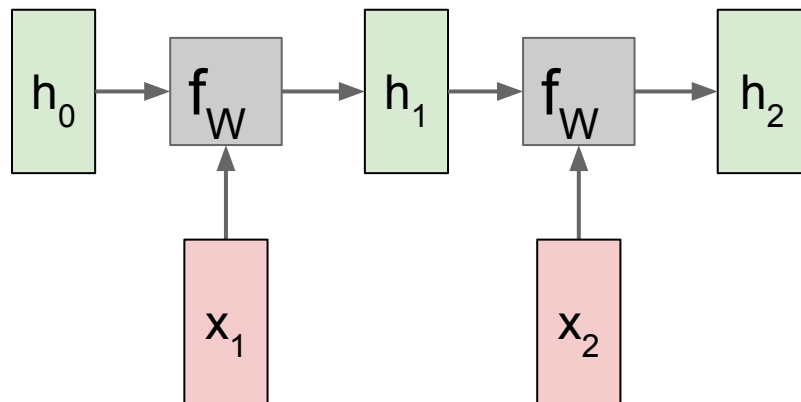
$$y_t = W_{hy}h_t$$

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

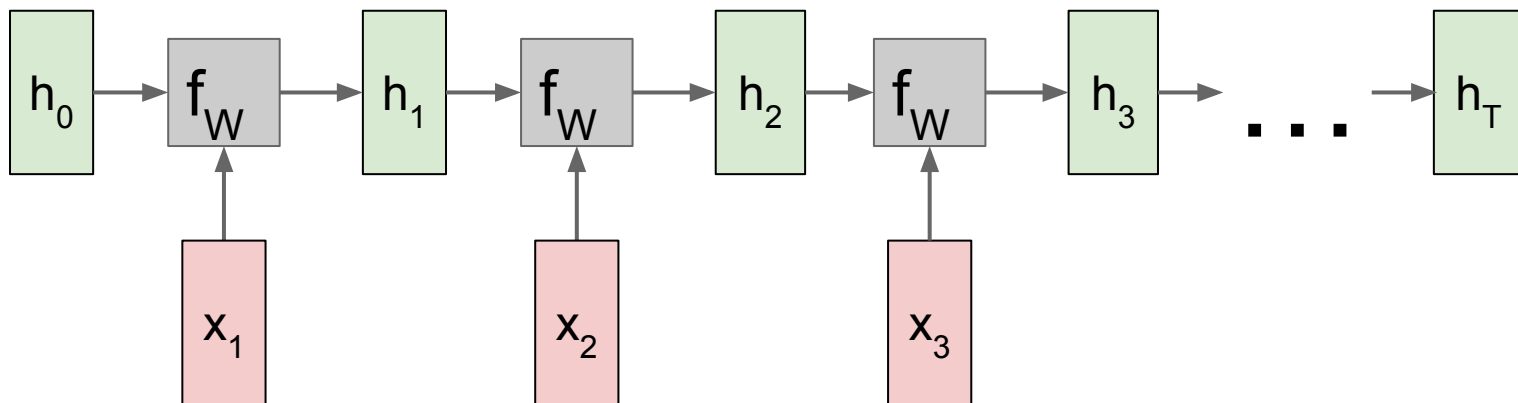
RNN: Computational Graph



RNN: Computational Graph

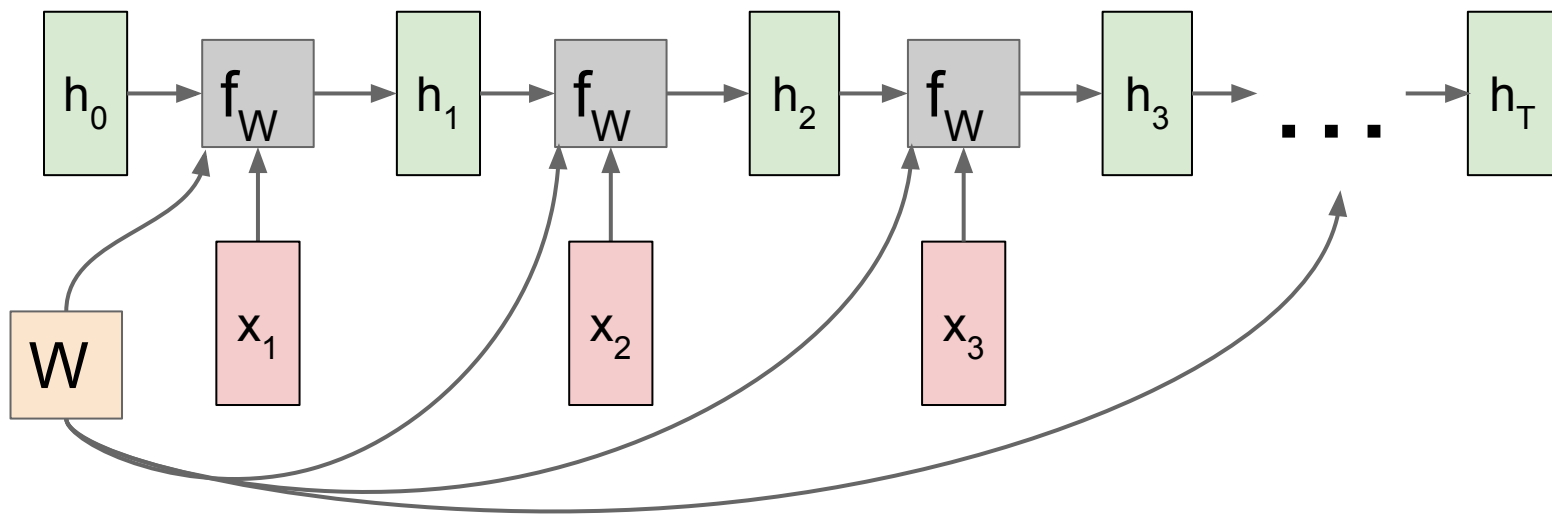


RNN: Computational Graph

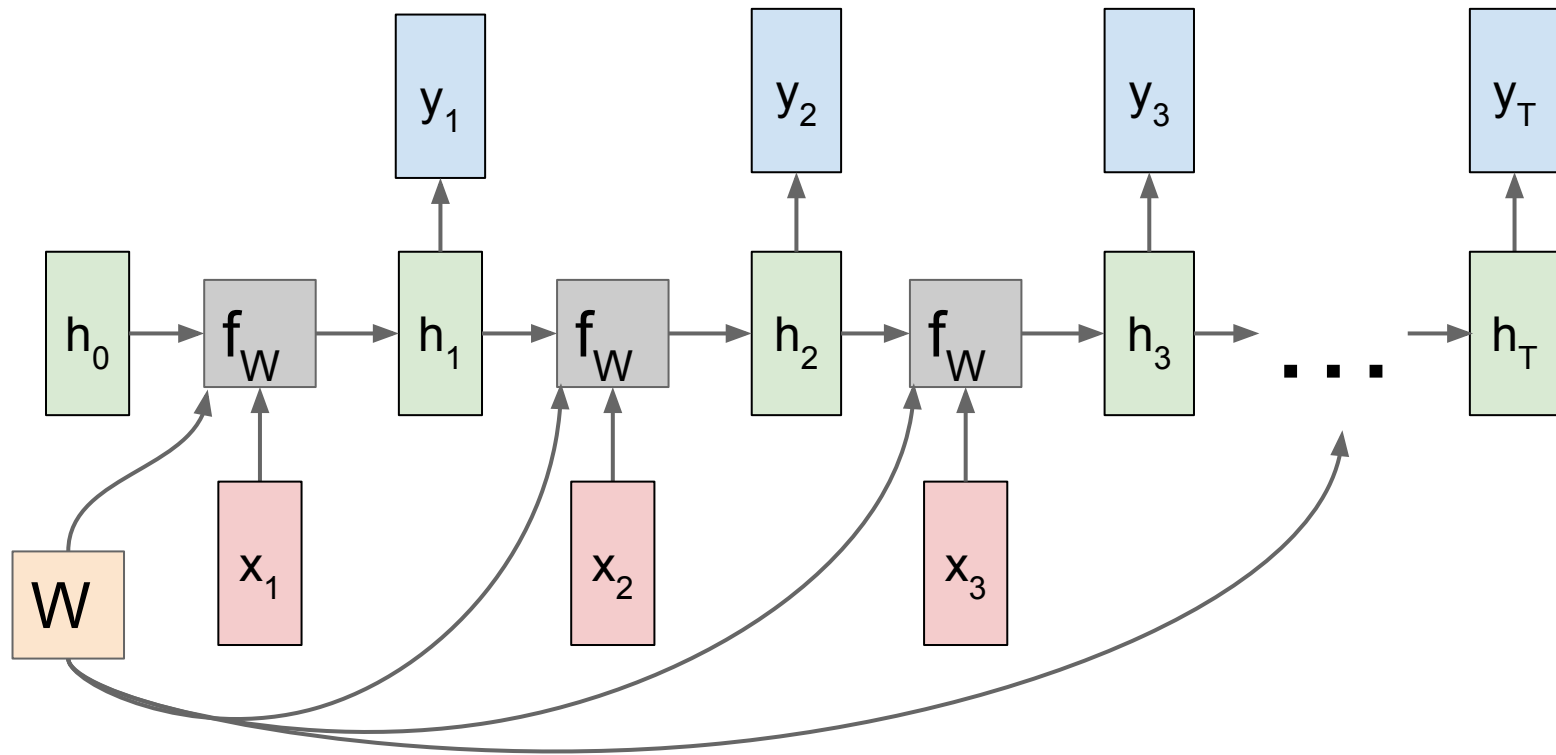


RNN: Computational Graph

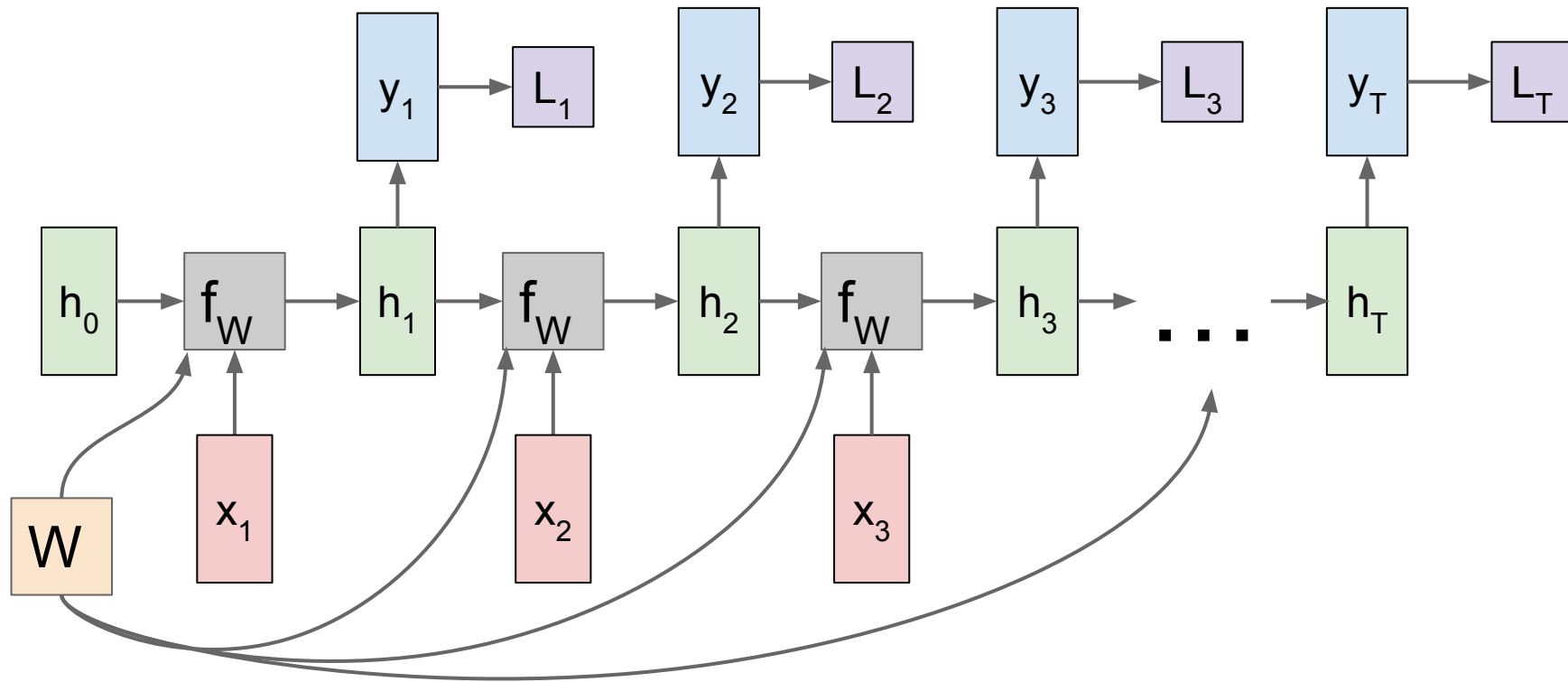
Re-use the same weight matrix at every time-step



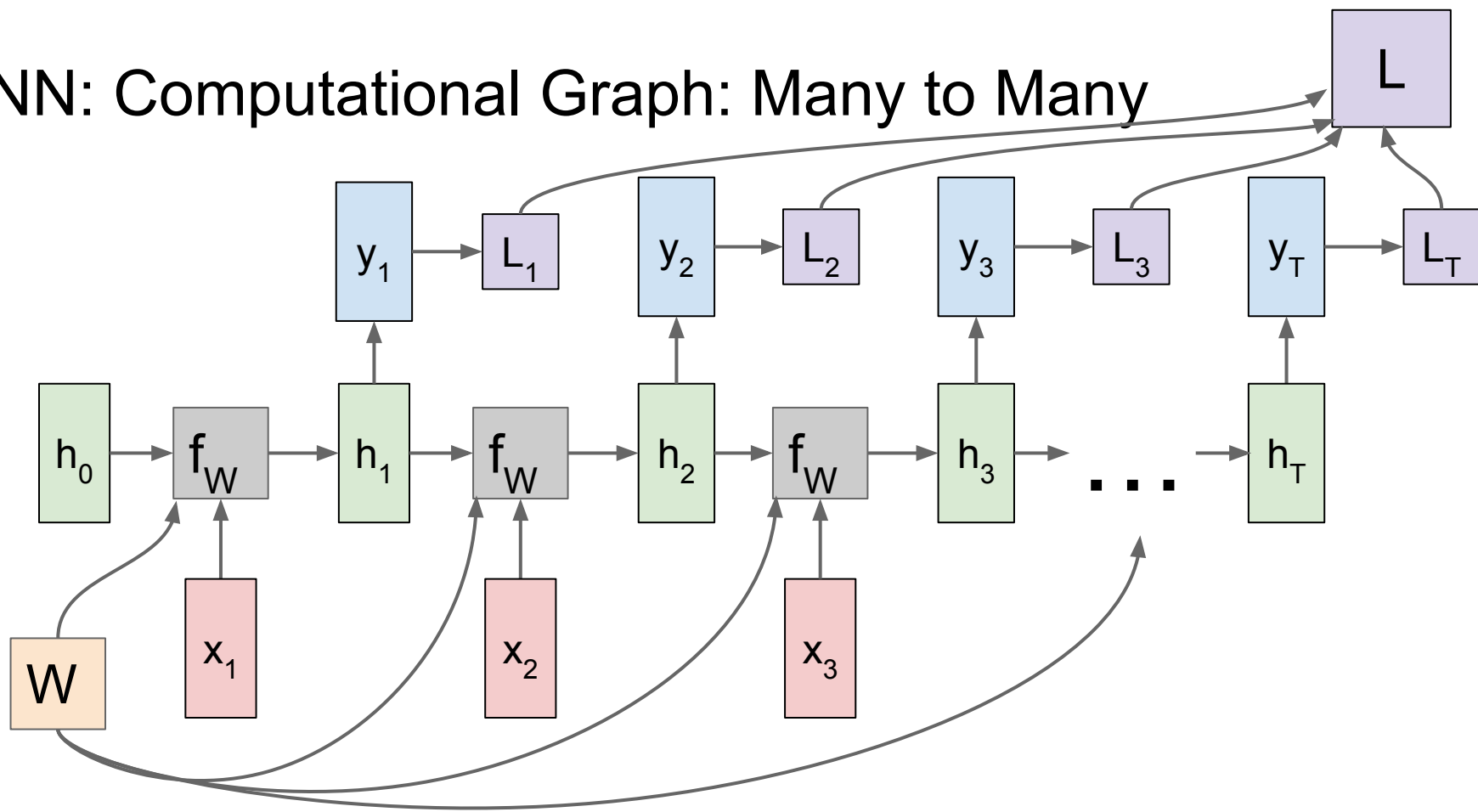
RNN: Computational Graph: Many to Many



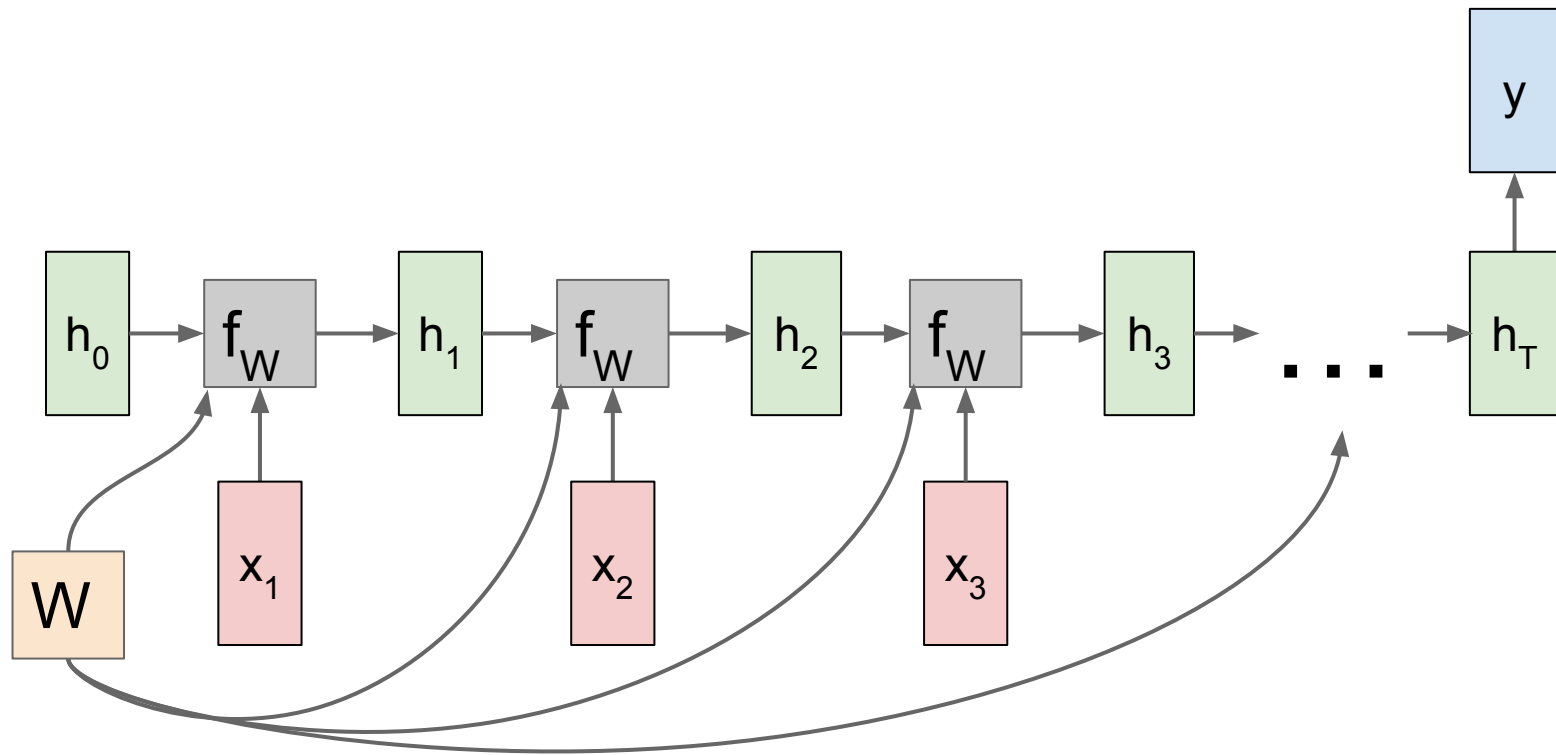
RNN: Computational Graph: Many to Many



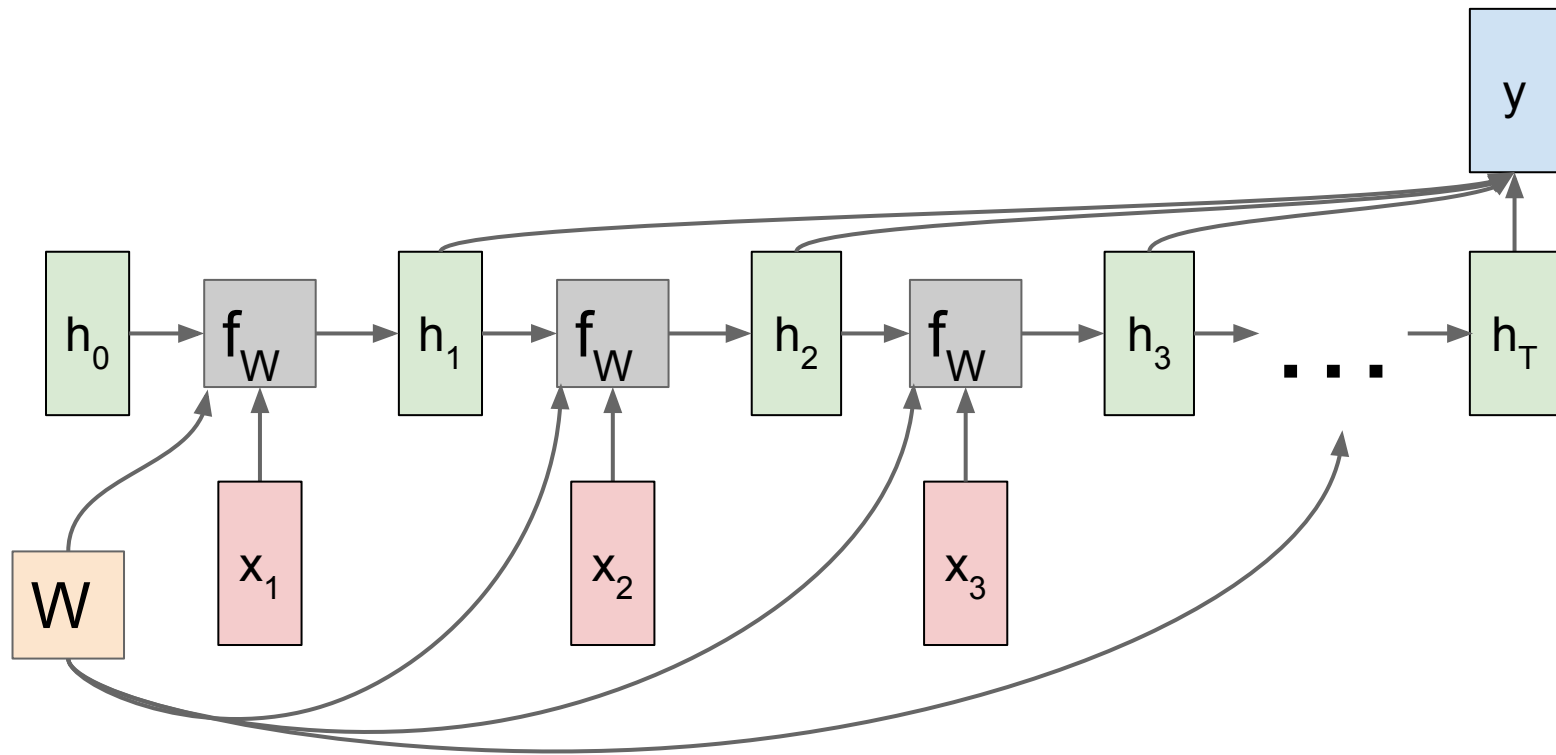
RNN: Computational Graph: Many to Many



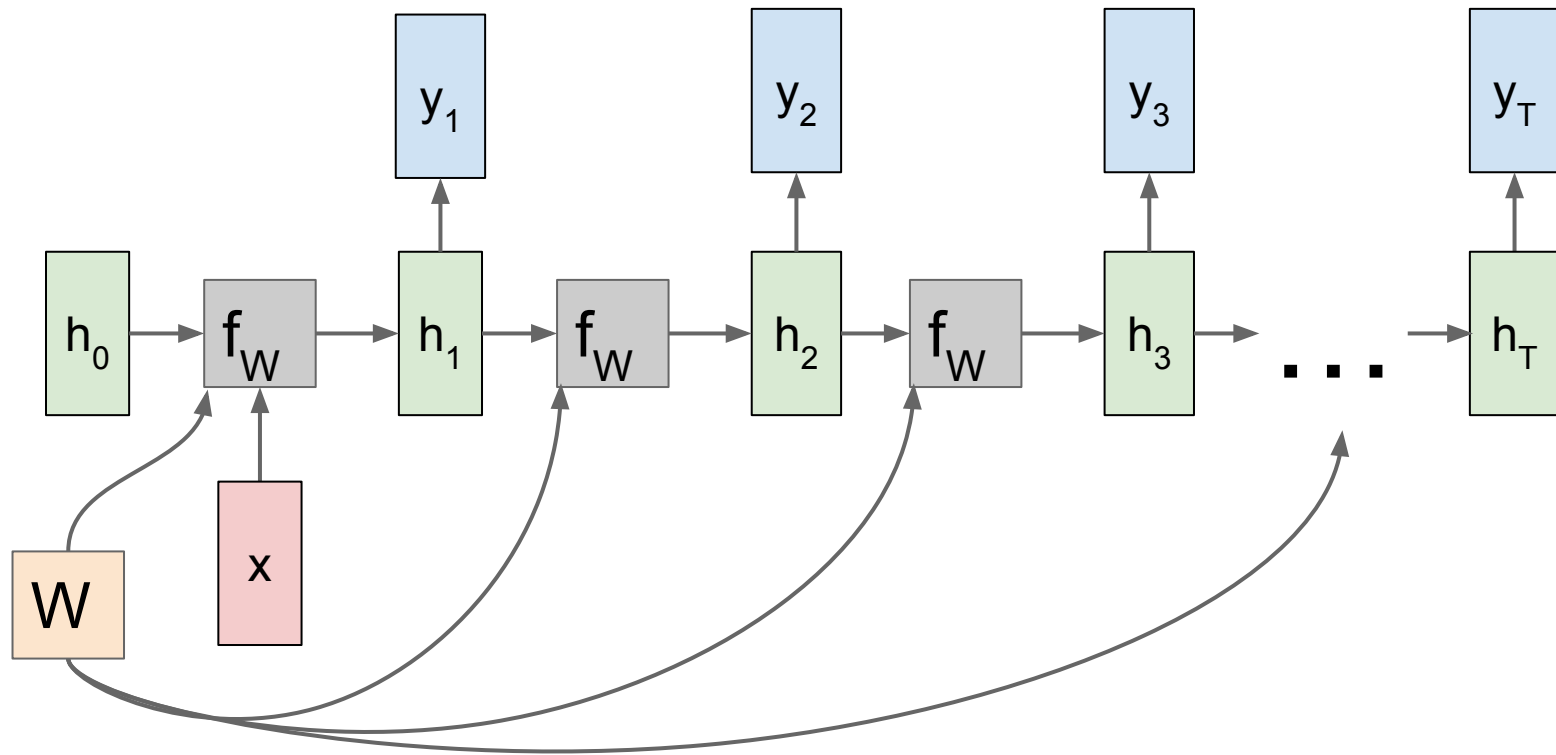
RNN: Computational Graph: Many to One



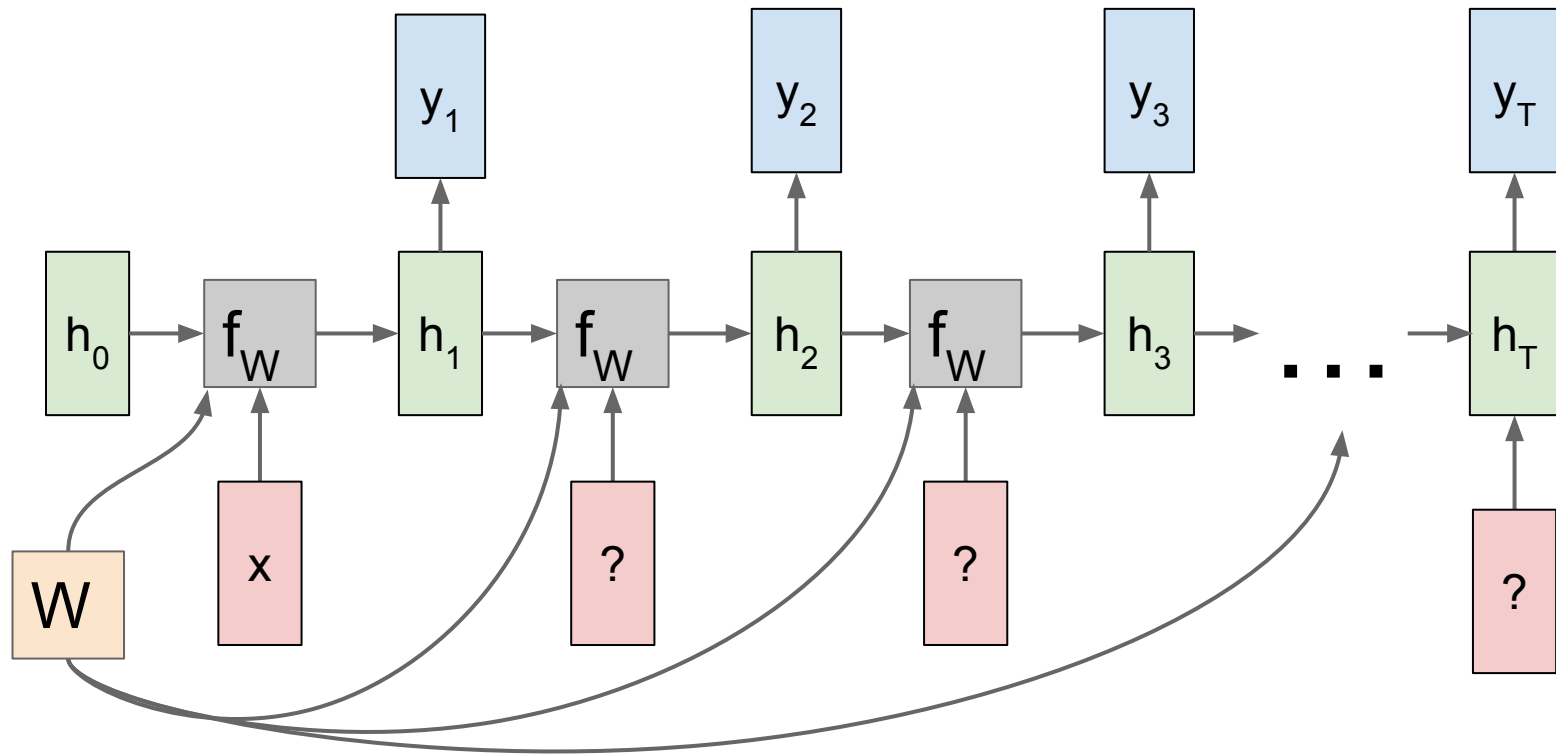
RNN: Computational Graph: Many to One



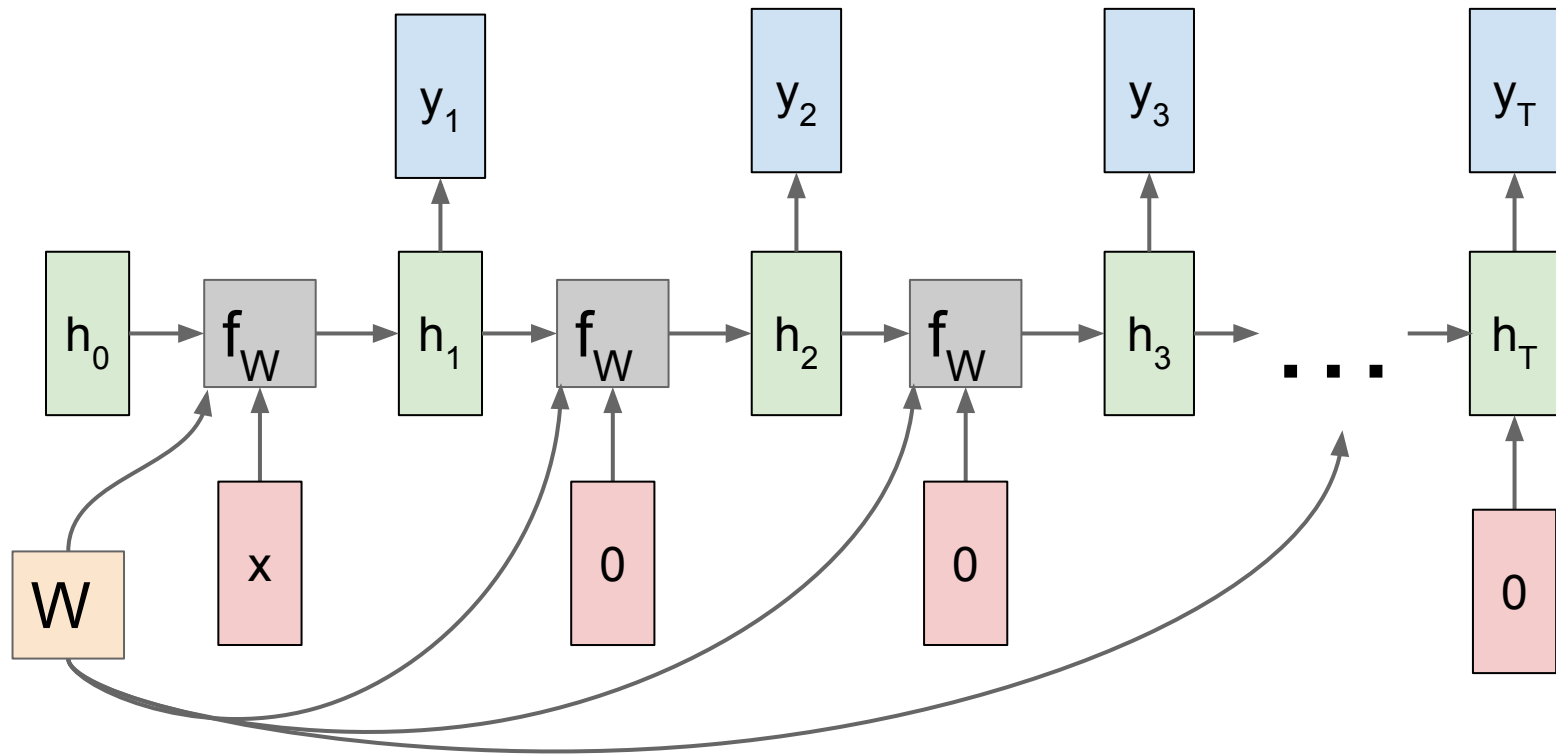
RNN: Computational Graph: One to Many



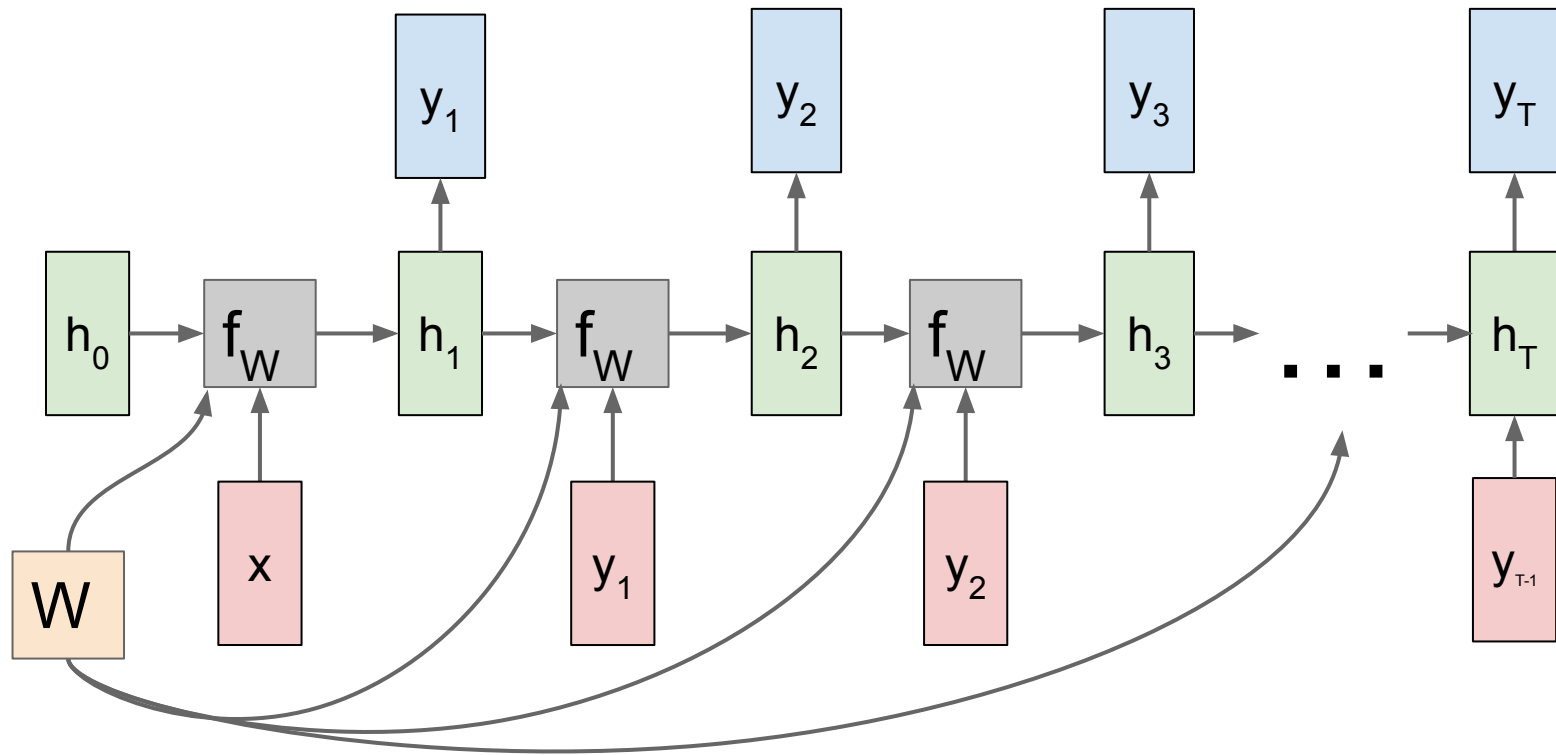
RNN: Computational Graph: One to Many



RNN: Computational Graph: One to Many

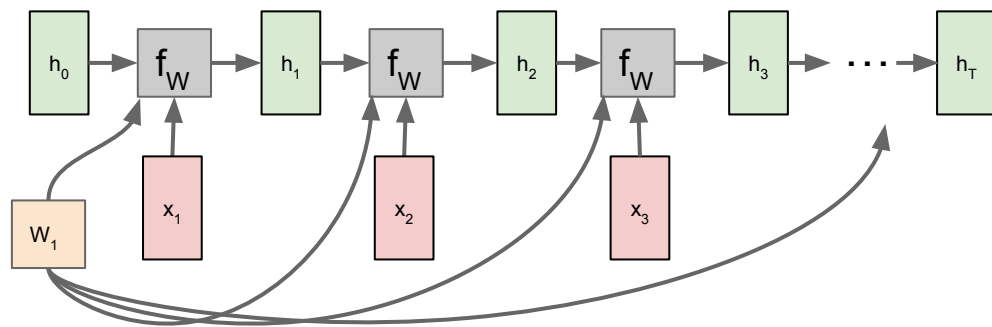


RNN: Computational Graph: One to Many



Sequence to Sequence: Many-to-one + one-to-many

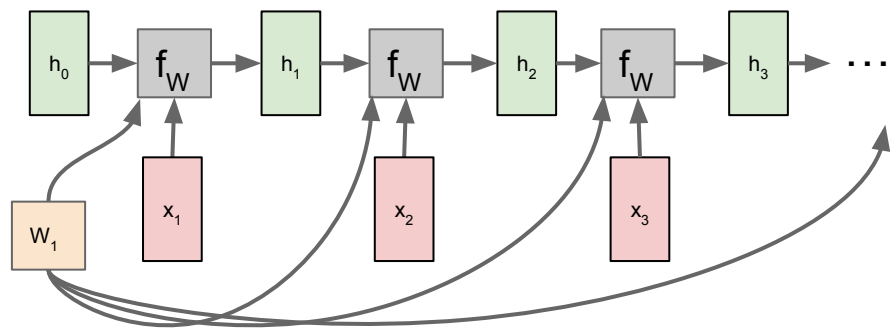
Many to one: Encode input sequence in a single vector



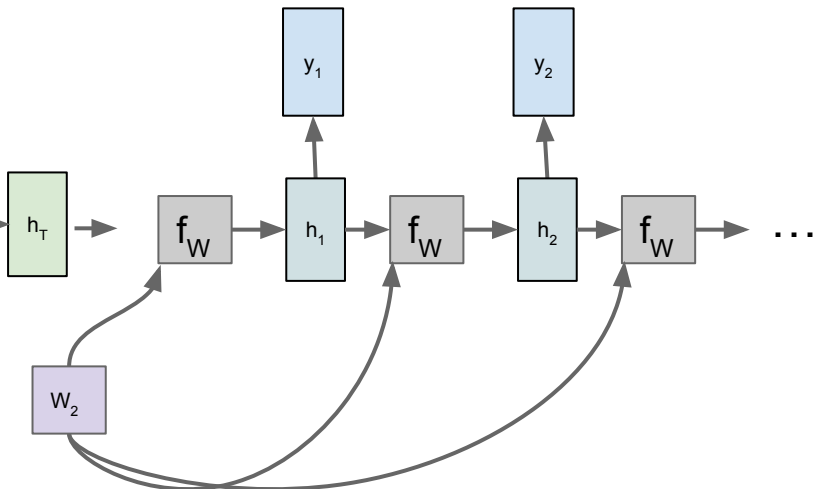
Sutskever et al, "Sequence to Sequence Learning with Neural Networks", NIPS 2014

Sequence to Sequence: Many-to-one + one-to-many

Many to one: Encode input sequence in a single vector



One to many: Produce output sequence from single input vector

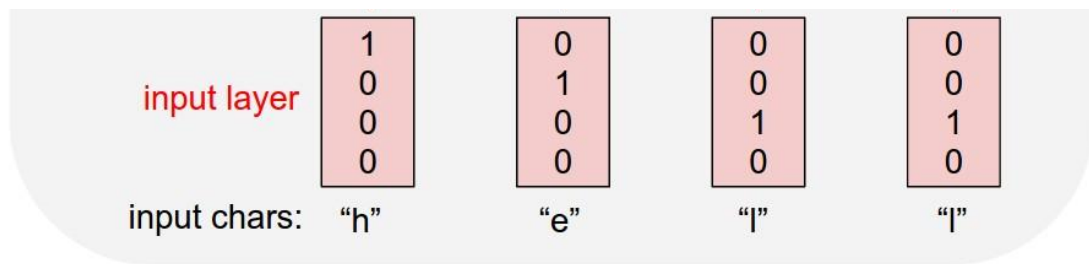


Sutskever et al, "Sequence to Sequence Learning with Neural Networks", NIPS 2014

Example: Character-level Language Model

Vocabulary:
[h,e,l,o]

Example training
sequence:
“hello”

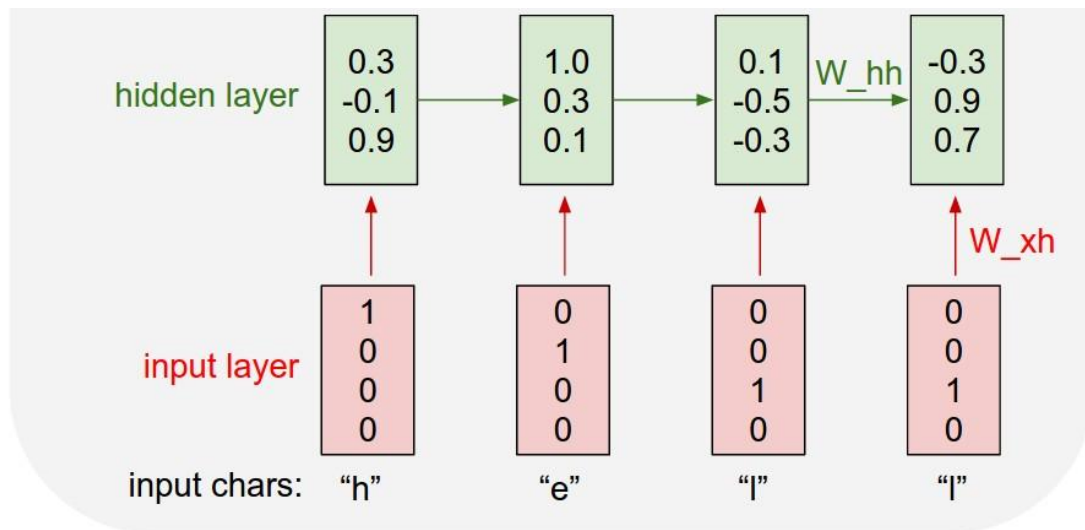


Example: Character-level Language Model

Vocabulary:
[h,e,l,o]

Example training
sequence:
“hello”

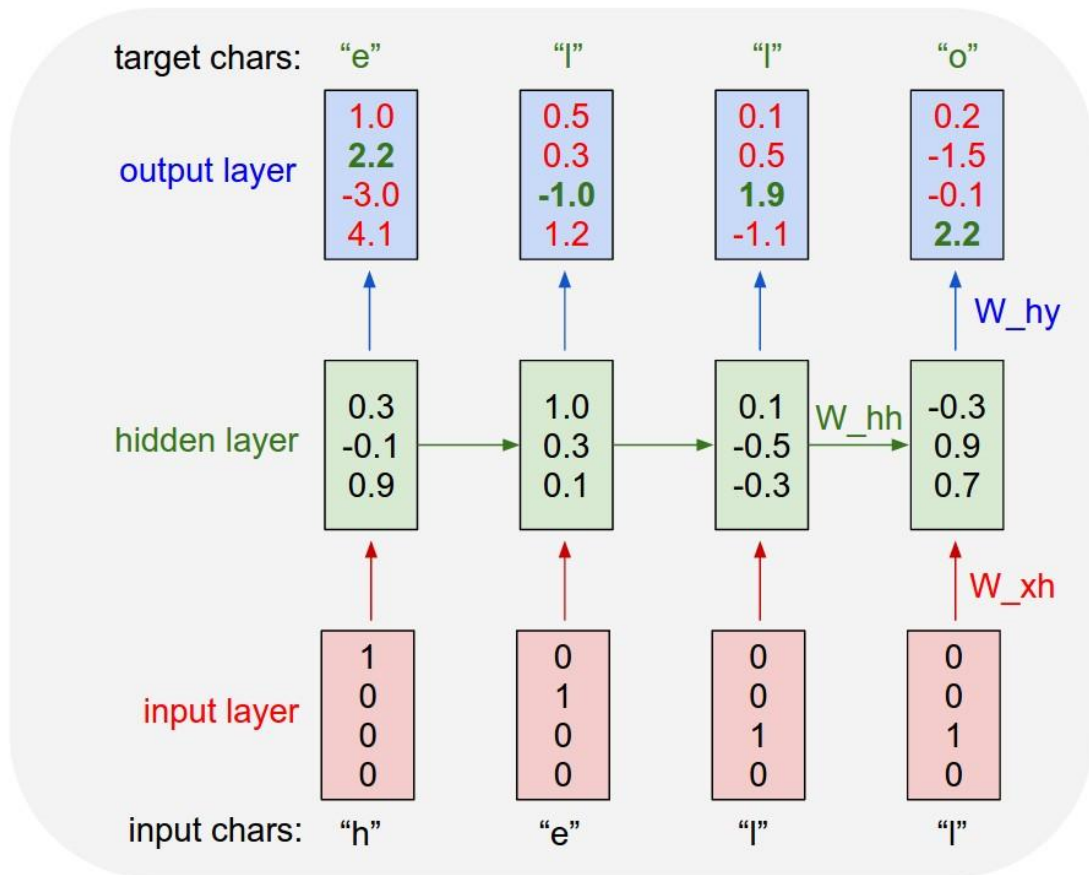
$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$



Example: Character-level Language Model

Vocabulary:
[h,e,l,o]

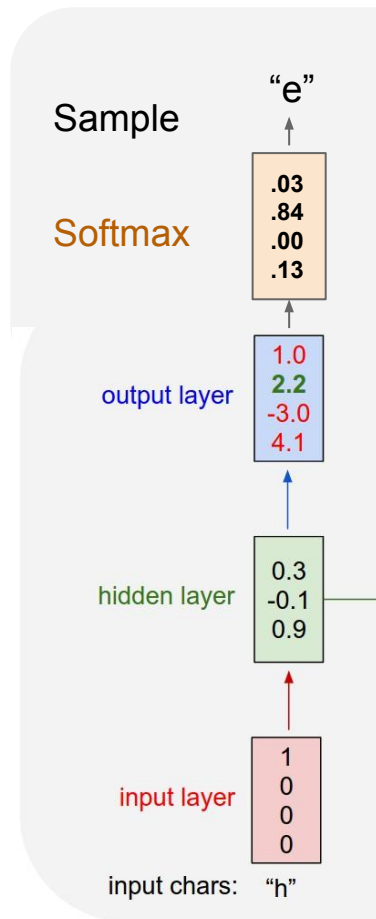
Example training
sequence:
“hello”



Example: Character-level Language Model Sampling

Vocabulary:
[h,e,l,o]

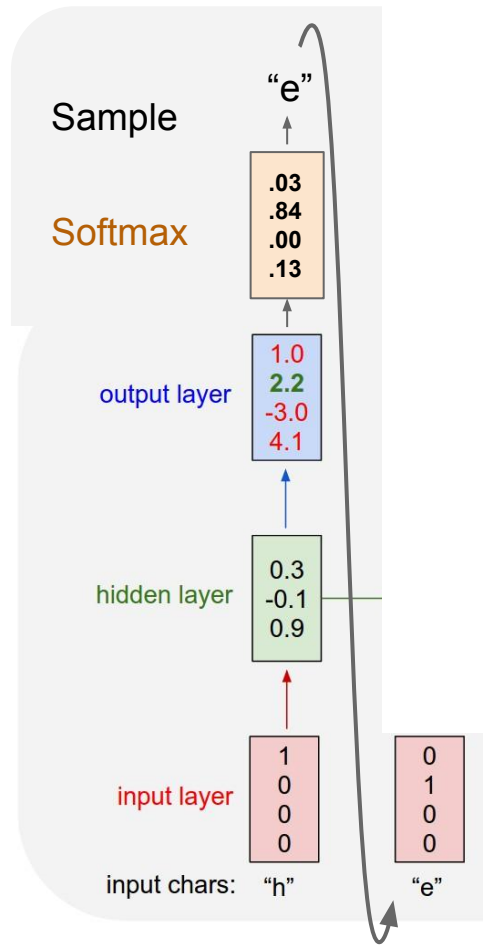
At test-time sample
characters one at a time,
feed back to model



Example: Character-level Language Model Sampling

Vocabulary:
[h,e,l,o]

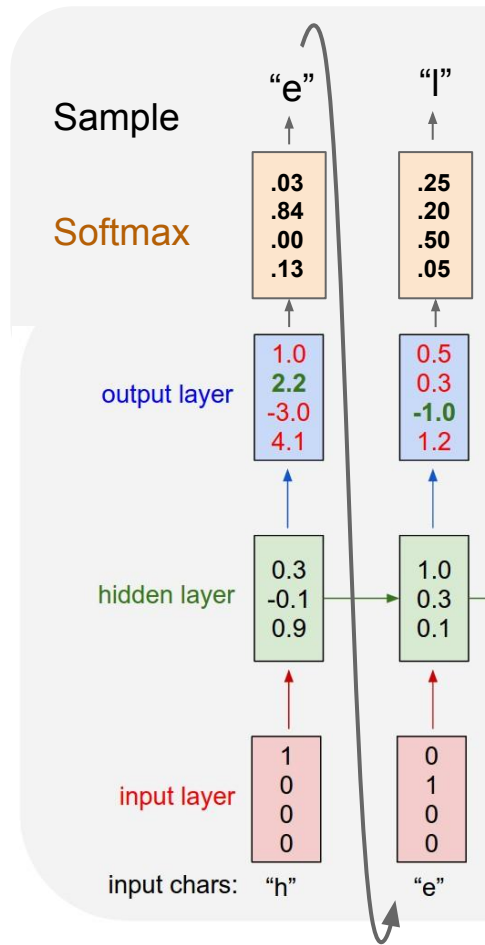
At test-time sample
characters one at a time,
feed back to model



Example: Character-level Language Model Sampling

Vocabulary:
[h,e,l,o]

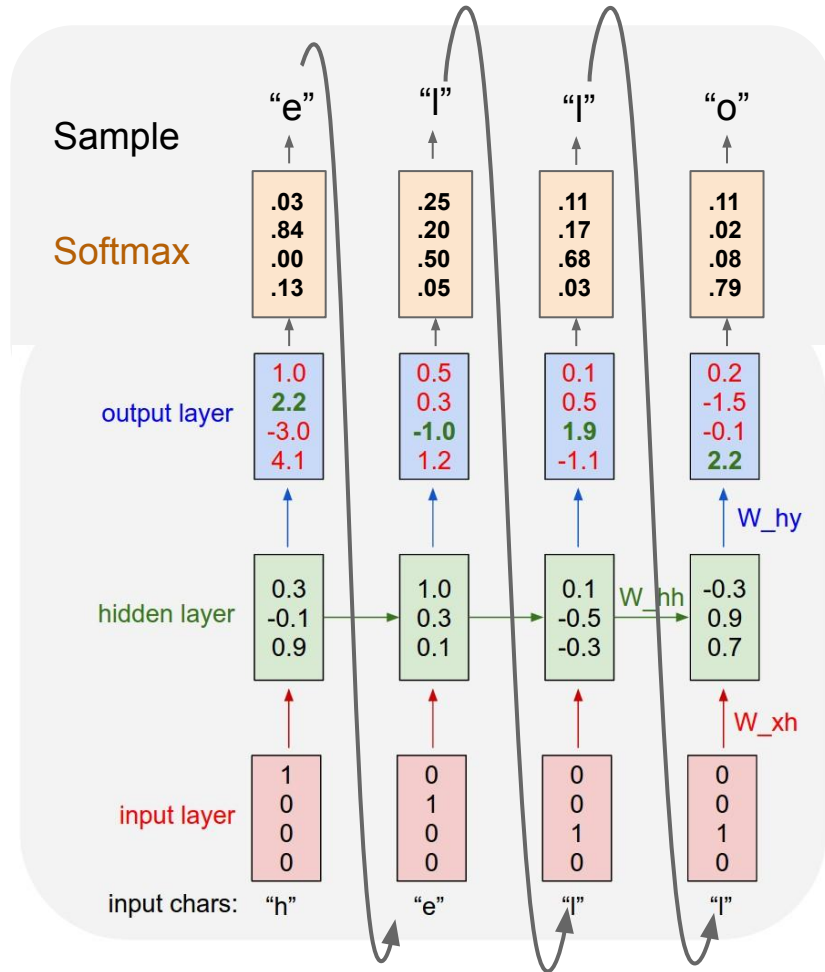
At test-time sample
characters one at a time,
feed back to model



Example: Character-level Language Model Sampling

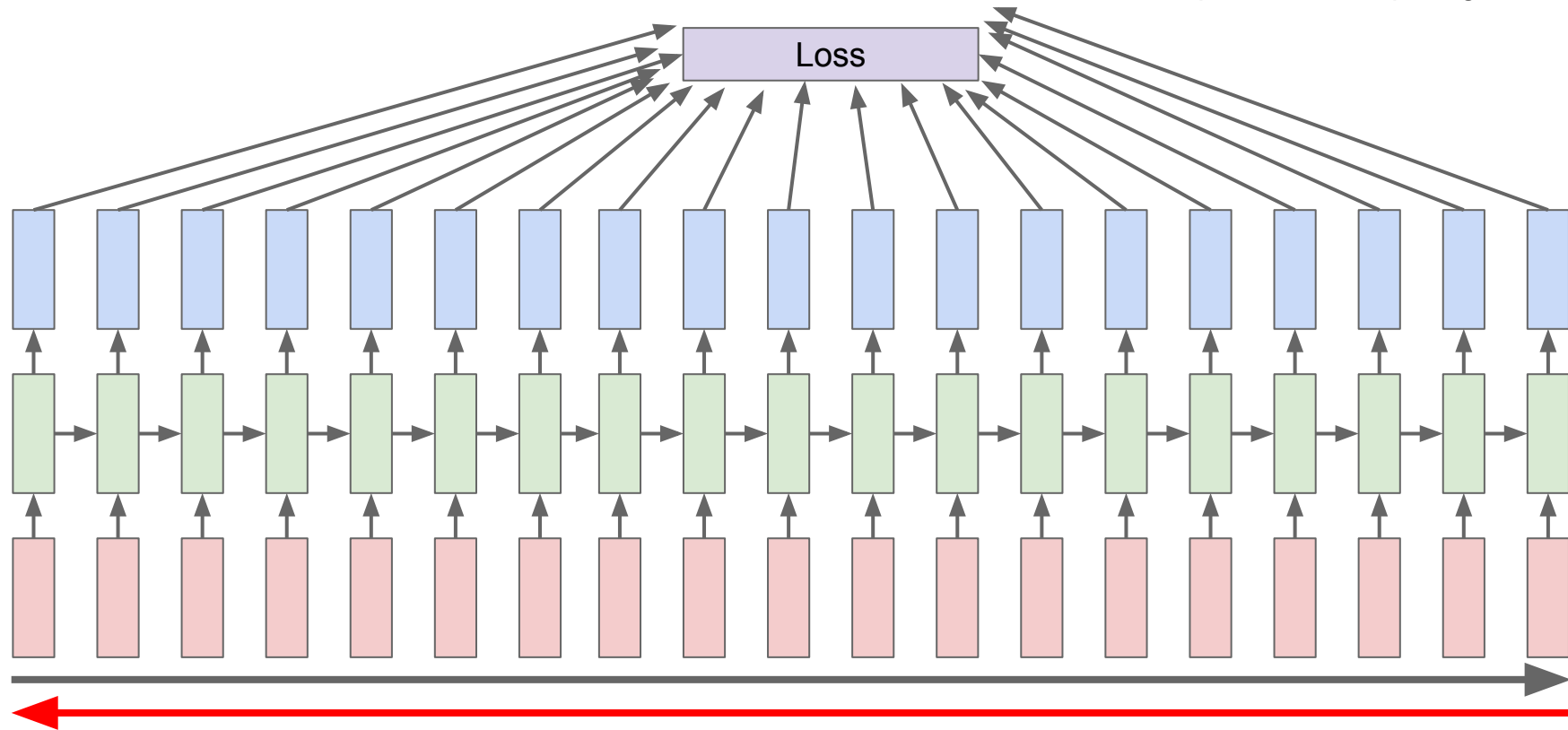
Vocabulary:
[h,e,l,o]

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

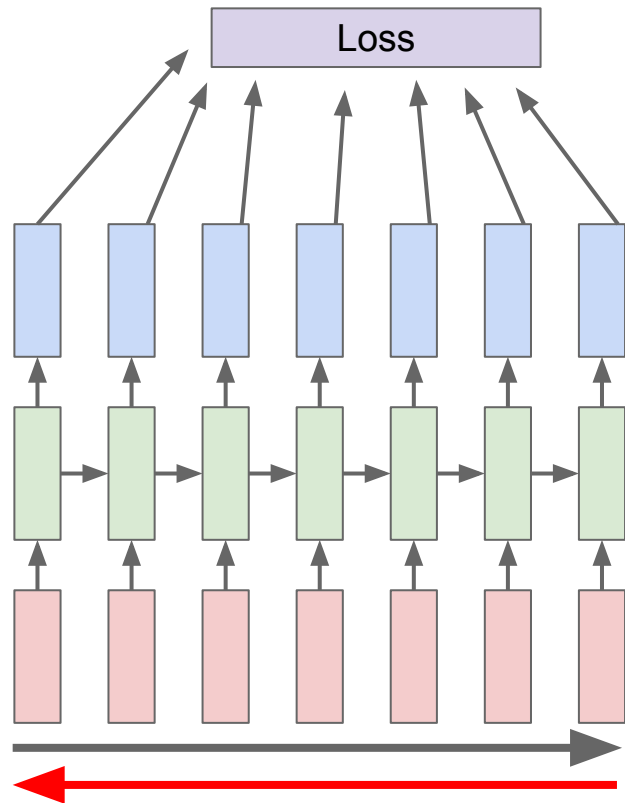


Backpropagation through time

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

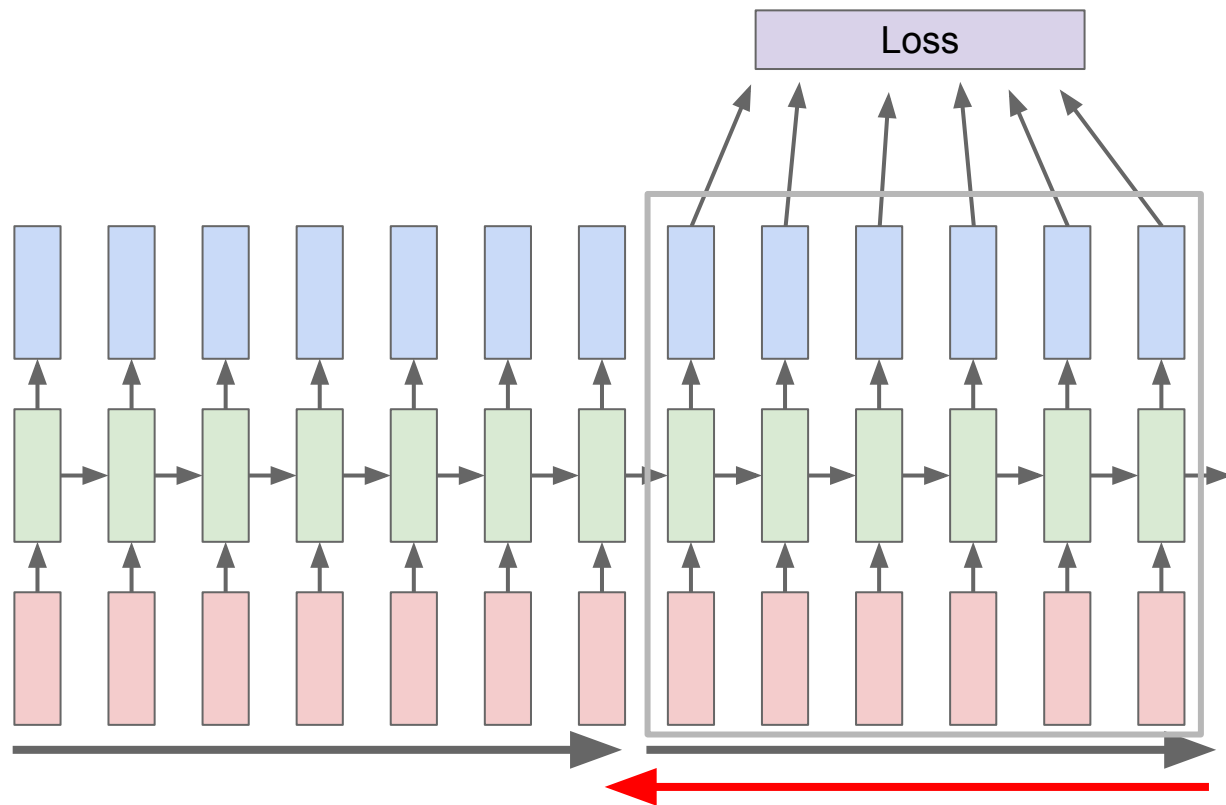


Truncated Backpropagation through time



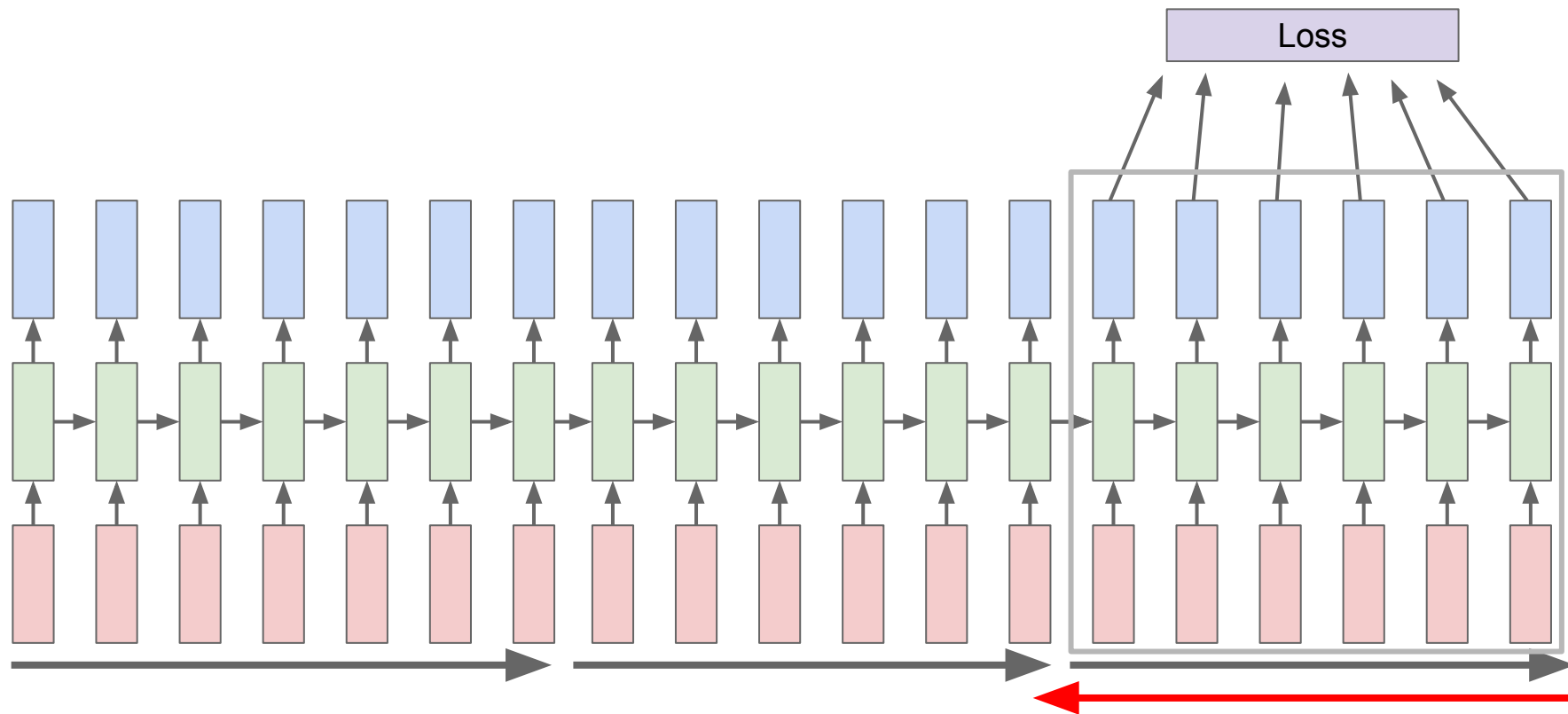
Run forward and backward through chunks of the sequence instead of whole sequence

Truncated Backpropagation through time



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

Truncated Backpropagation through time



min-char-rnn.py gist: 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 wxh = 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 Nx1 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(wxh, 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(wxh), np.zeros_like(whh), np.zeros_like(why)
46     dbh, dby = 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         dby += dy
53         dh = np.dot(why.T, dy) + dhnext # backprop into h
54         dhrdw = (1 - hs[t]**2) * dh # backprop through tanh nonlinearity
55         dbh += dhrdw
56         dwxh += np.dot(dhrdw, xs[t].T)
57         dwhh += np.dot(dhrdw, hs[t-1].T)
58         dhnext = np.dot(whh.T, dhrdw)
59     for dparam in [dwxh, dwhh, dwhy, dbh, dby]:
60         np.clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients
61     return loss, dwxh, dwhh, dwhy, dbh, dby, 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(wxh, 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 = np.zeros((vocab_size, 1))
77         x[ix] = 1
78         ixes.append(ix)
79     return ixes
80
81 n, p = 0, 0
82 mxhx, mwhh, mwhy = np.zeros_like(wxh), np.zeros_like(whh), np.zeros_like(why)
83 mbh, mby = np.zeros_like(bh), np.zeros_like(by) # memory variables for Adagrad
84 smooth_loss = -np.log(1.0/vocab_size)*seq_length # loss at iteration 0
85 while True:
86     # prepare inputs (we're sweeping from left to right in steps seq_length long)
87     if p+seq_length+1 >= len(data) or n == 0:
88         hprev = np.zeros((hidden_size,1)) # reset RNN memory
89         p = 0 # go from start of data
90     inputs = [char_to_ix[ch] for ch in data[p:p+seq_length]]
91     targets = [char_to_ix[ch] for ch in data[p+1:p+seq_length+1]]
92
93     # sample from the model now and then
94     if n % 100 == 0:
95         sample_ix = sample(hprev, inputs[0], 200)
96         txt = ''.join(ix_to_char[ix] for ix in sample_ix)
97         print '---->%s\n----' % (txt, )
98
99     # forward seq_length characters through the net and fetch gradient
100     loss, dwxh, dwhh, dwhy, dbh, dby, hprev = lossFun(inputs, targets, hprev)
101     smooth_loss = smooth_loss * 0.999 + loss * 0.001
102     if n % 100 == 0: print 'iter %d, loss: %f' % (n, smooth_loss) # print progress
103
104     # perform parameter update with Adagrad
105     for param, dparam, mem in zip([dwxh, dwhh, dwhy, dbh, dby],
106                                   [mxhx, mwhh, mwhy, mbh, mby]):
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
```

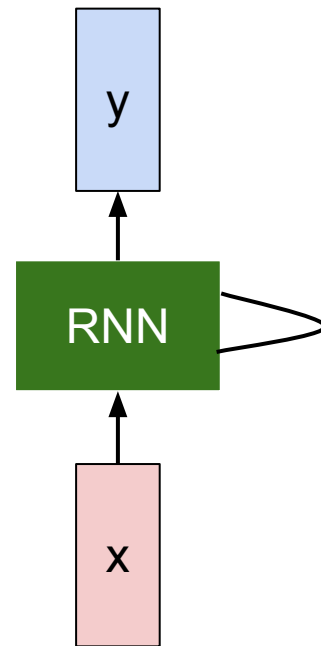
(<https://gist.github.com/karpathy/d4dee566867f8291f086>)

THE SONNETS

by William Shakespeare

From fairest creatures we desire increase,
That thereby beauty's rose might never die,
But as the ripper 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.



at first:

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

train more

"Tmont thithey" fomesscerliund
Keushey. Thom here
sheulke, anmerenith ol sivh I lalterthend Bleipile shuw y 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 oftended 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 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 nudes 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 apt, 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.

The Stacks Project: open source algebraic geometry textbook

 **The Stacks Project**

[home](#) [about](#) [tags explained](#) [tag lookup](#) [browse](#) [search](#) [bibliography](#) [recent comments](#) [blog](#) [add slogans](#)

Browse chapters

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 $\bigoplus_{n=1,\dots,m}$ where $\mathcal{L}_{m,*} = 0$, 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 $x, x', s'' \in S'$ 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

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

and

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

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 $X_{spaces, \acute{e}tale}$ 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 $\text{Proj}_X(\mathcal{A}) = \text{Spec}(B)$ 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 $\mathcal{F}_{x_0} = \mathcal{F}_{x_0} = \mathcal{F}_{X,\dots,0}$.

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 $i_0 \leq \mathfrak{p}$ is a subset of $\mathcal{J}_{n,0} \circ \overline{A}_2$ works.

Lemma 0.3. In Situation ?? . Hence we may assume $\mathfrak{q}' = 0$.

Proof. We will use the property we see that $\mathfrak{p}$ 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 $\mathbb{Z}$ 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

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

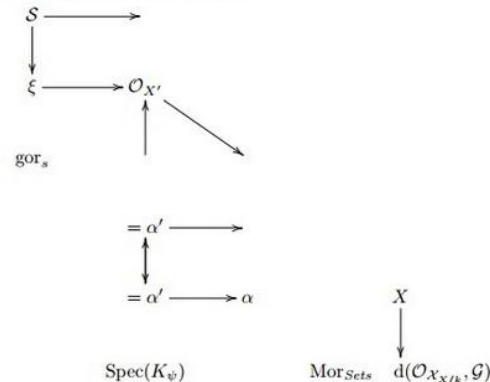
be a morphism of algebraic spaces over S and Y .

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 $\mathcal{O}_X(\mathcal{U})$ which is locally of finite type. □

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



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 $\mathcal{C}$. The functor $\mathcal{F}$ is a “field

$$\mathcal{O}_{X,x} \longrightarrow \mathcal{F}_x \xrightarrow{-1(\mathcal{O}_{X_{\text{étale}}})} \mathcal{O}_{X_t}^{-1} \mathcal{O}_{X_\lambda}(\mathcal{O}_{X_n}^\vee)$$

is an isomorphism of covering of $\mathcal{O}_{X_t}$. 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.

Linux kernel source tree

Repository statistics: 520,037 commits, 1 branch, 420 releases, 5,039 contributors.

Current branch: **master** - **linux** / +

Merge branch 'drm-fixes' of git://people.freedesktop.org/~airlied/linux

torvalds authored 9 hours ago | latest commit 4b1706927d

Directory	Commit Message	Time Ago
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/lin...	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 'perl-urgent-for-linus' of git://git.kernel.org/pub/scm/lin...	a day ago
init	init: fix regression by supporting devices with major:minor:offset fo...	a month ago
io	Merge branch 'for-linus' of git://git.kernel.org/pub/scm/linux/kernel...	a month ago

Code

Pull requests 74

Pulse

Graphs

HTTPS clone URL

https://github.com/torvalds/linux.git

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

Clone in Desktop

Download ZIP

```

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 << i))
            pipe = (in_use & UMXTHREAD_UNCCA) +
                ((count & 0x00000000ffffffff) & 0x0000000f) << 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/setew.h>
#include <asm/pgproto.h>

```

```

#include <asm/io.h>
#include <asm/prom.h>
#include <asm/e820.h>
#include <asm/system_info.h>
#include <asm/setew.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;
}

```

OpenAI GPT-2 generated text

[source](#)

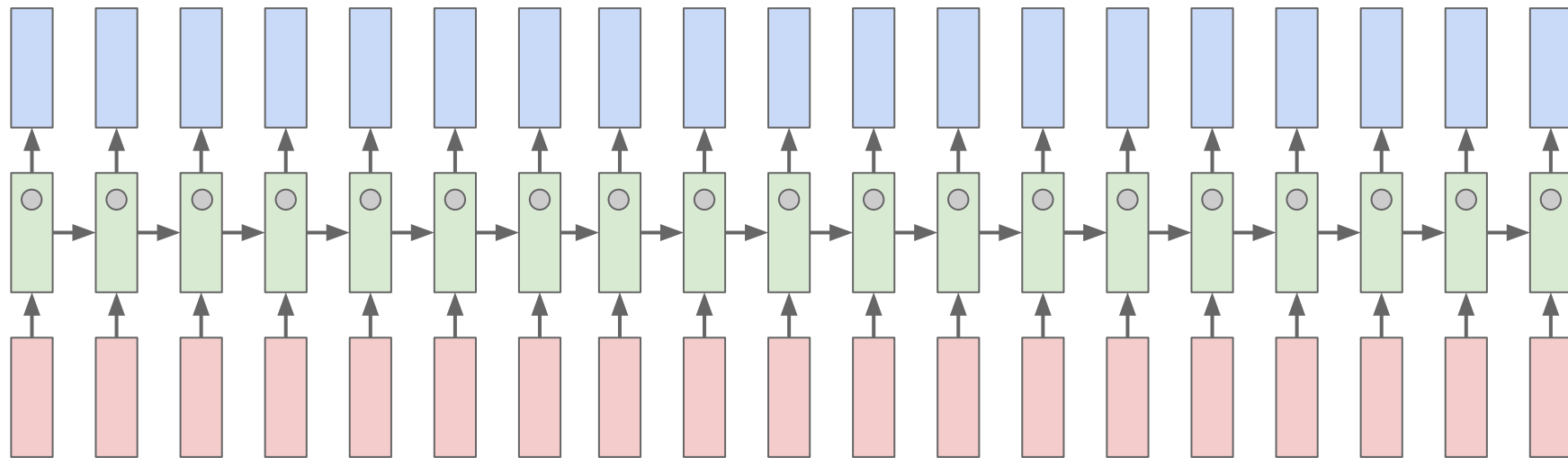
Input: In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English.

Output: The scientist named the population, after their distinctive horn, Ovid's Unicorn. These four-horned, silver-white unicorns were previously unknown to science.

Now, after almost two centuries, the mystery of what sparked this odd phenomenon is finally solved.

Dr. Jorge Pérez, an evolutionary biologist from the University of La Paz, and several companions, were exploring the Andes Mountains when they found a small valley, with no other animals or humans. Pérez noticed that the valley had what appeared to be a natural fountain, surrounded by two peaks of rock and silver snow.

Searching for interpretable cells



Searching for interpretable cells

```
/* 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.
     */
}
```

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

Figures copyright Karpathy, Johnson, and Fei-Fei, 2015; reproduced with permission

Searching for interpretable cells

"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

Karpathy, Johnson, and Fei-Fei: Visualizing and Understanding Recurrent Networks, ICLR Workshop 2016
Figures copyright Karpathy, Johnson, and Fei-Fei, 2015; reproduced with permission

Searching for interpretable cells

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, 2015; reproduced with permission

Searching for interpretable cells

```
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

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

Figures copyright Karpathy, Johnson, and Fei-Fei, 2015; reproduced with permission

Searching for interpretable cells

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

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

Figures copyright Karpathy, Johnson, and Fei-Fei, 2015; reproduced with permission

Searching for interpretable cells

```
#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

Karpathy, Johnson, and Fei-Fei: Visualizing and Understanding Recurrent Networks, ICLR Workshop 2016
Figures copyright Karpathy, Johnson, and Fei-Fei, 2015; reproduced with permission

RNN tradeoffs

RNN Advantages:

- Can process any length input
- Computation for step t can (in theory) use information from many steps back
- Model size doesn't increase for longer input
- Same weights 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

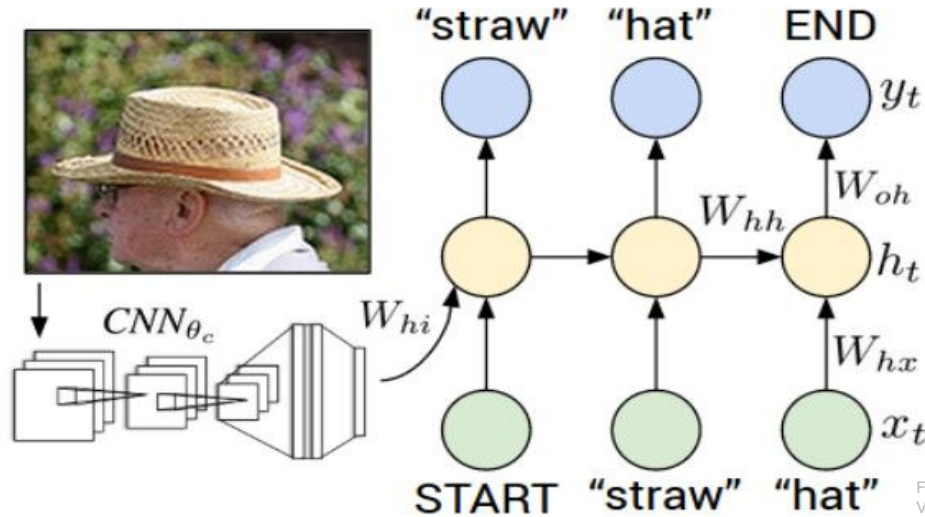


Figure from Karpathy et al, "Deep Visual-Semantic Alignments for Generating Image Descriptions", CVPR 2015; figure copyright IEEE, 2015. Reproduced for educational purposes.

Explain Images with Multimodal Recurrent Neural Networks, Mao et al.

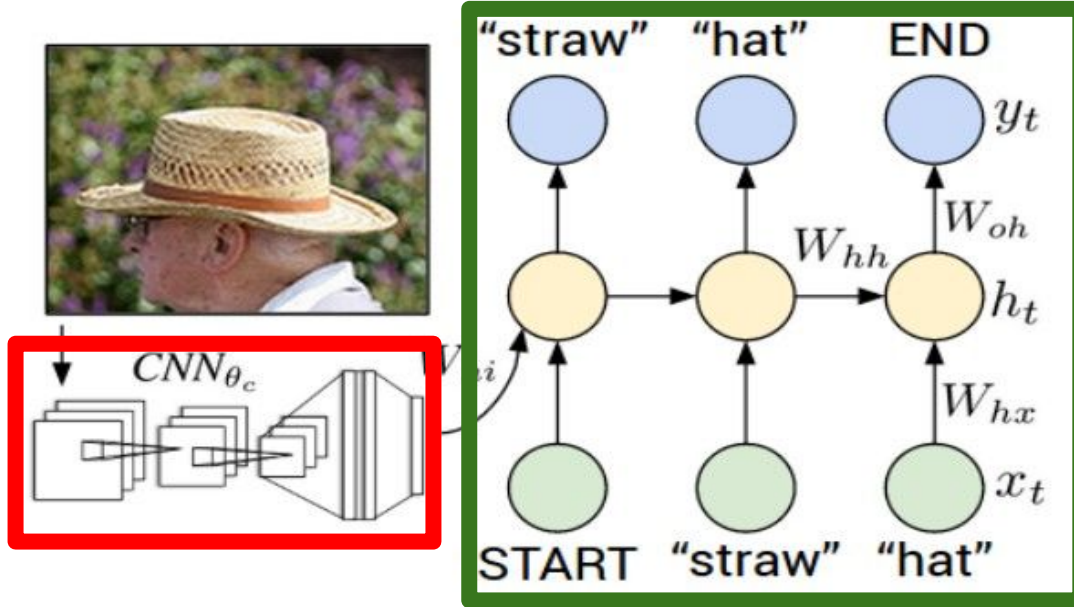
Deep Visual-Semantic Alignments for Generating Image Descriptions, Karpathy and Fei-Fei

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



test image

[This image](#) is [CC0 public domain](#)

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

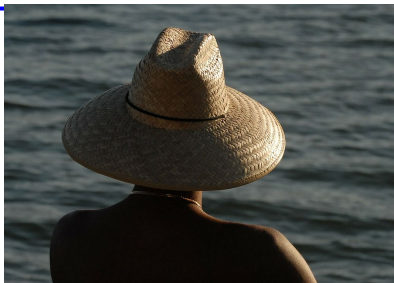
maxpool

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

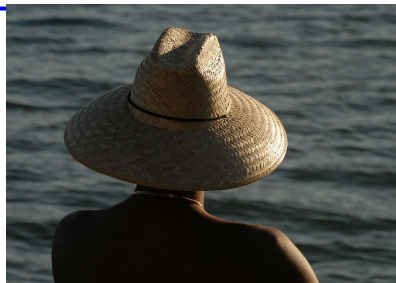
maxpool

FC-4096

FC-4096

FC-1000

softmax



test image



image

conv-64

conv-64

maxpool

conv-128

conv-128

maxpool

conv-256

conv-256

maxpool

conv-512

conv-512

maxpool

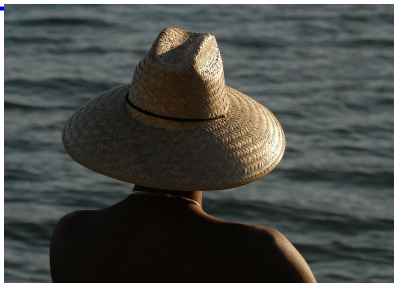
conv-512

conv-512

maxpool

FC-4096

FC-4096

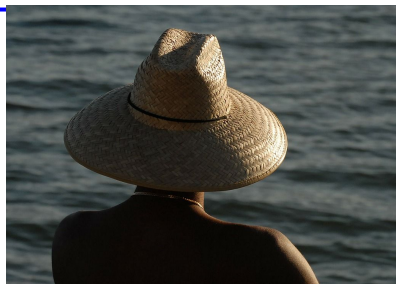


test image

x0
<START>

image

test 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

v

y_0

h_0

x_0
<START>

W_{ih}

before:

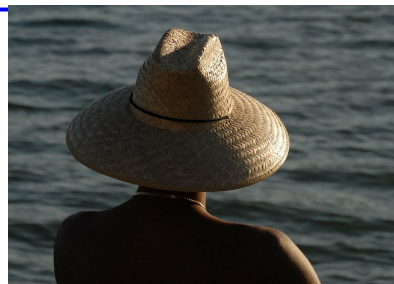
$$h = \tanh(W_{xh} * x + W_{hh} * h)$$

now:

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

image

test image



conv-64
conv-64
maxpool

conv-128
conv-128
maxpool

conv-256
conv-256
maxpool

conv-512
conv-512
maxpool

conv-512
conv-512
maxpool

FC-4096
FC-4096

y0

h0

x0
<START>

straw

sample!

image

conv-64

conv-64

maxpool

conv-128

conv-128

maxpool

conv-256

conv-256

maxpool

conv-512

conv-512

maxpool

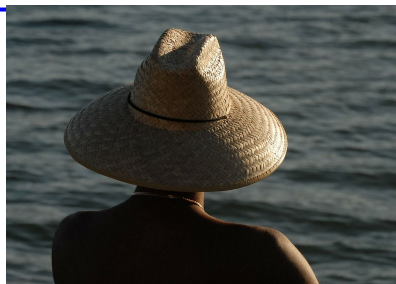
conv-512

conv-512

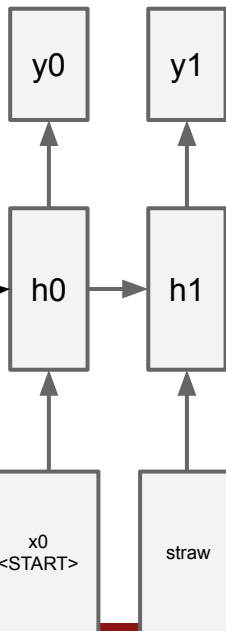
maxpool

FC-4096

FC-4096



test image



image

conv-64

conv-64

maxpool

conv-128

conv-128

maxpool

conv-256

conv-256

maxpool

conv-512

conv-512

maxpool

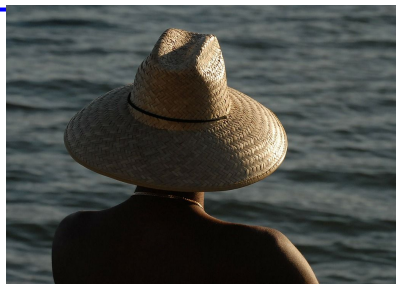
conv-512

conv-512

maxpool

FC-4096

FC-4096



test image

y0

y1

h0

h1

x0
<START>

straw

hat

sample!

image

conv-64

conv-64

maxpool

conv-128

conv-128

maxpool

conv-256

conv-256

maxpool

conv-512

conv-512

maxpool

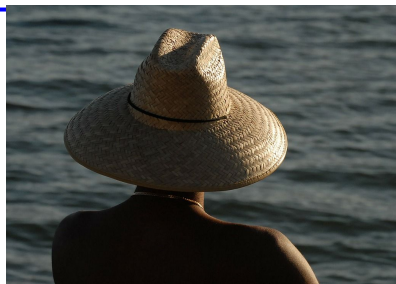
conv-512

conv-512

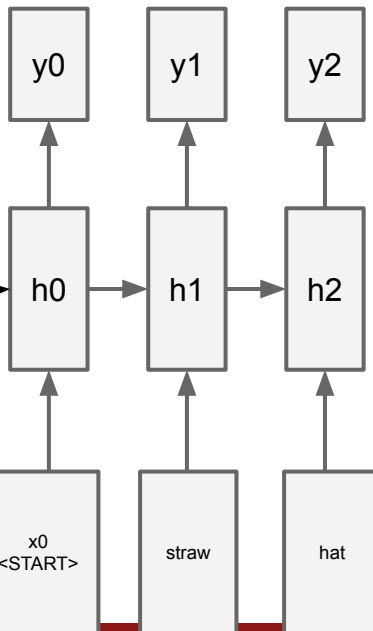
maxpool

FC-4096

FC-4096



test image



image

conv-64

conv-64

maxpool

conv-128

conv-128

maxpool

conv-256

conv-256

maxpool

conv-512

conv-512

maxpool

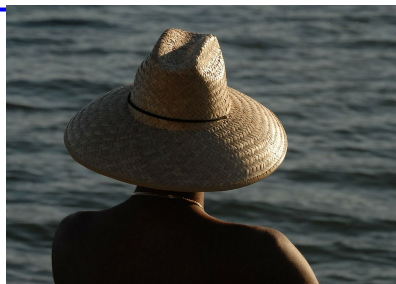
conv-512

conv-512

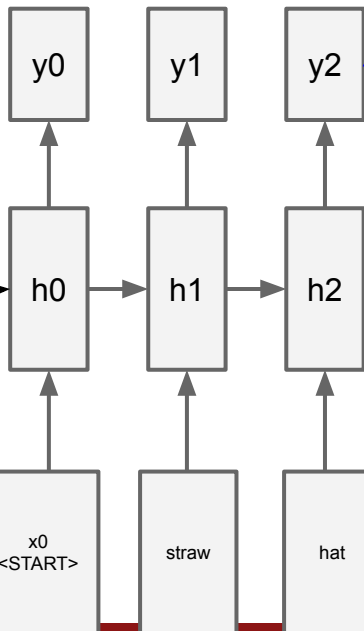
maxpool

FC-4096

FC-4096



test image



sample
<END> token
=> finish.

Image Captioning: Example Results

Captions generated using [neuraltalk2](#)

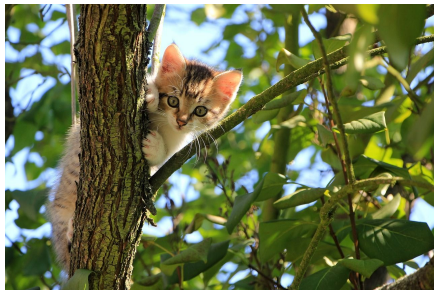
All images are [CC0 Public domain](#)

[cat suitcase](#) [cat tree](#) [dog bear](#)

[surfers](#) [tennis](#) [giraffe](#) [motorcycle](#)



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



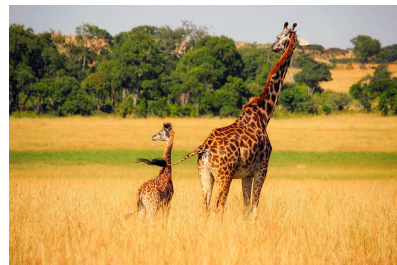
A white teddy bear sitting in the grass



Two people walking on the beach with surfboards



A tennis player in action on the court



Two giraffes standing in a grassy field



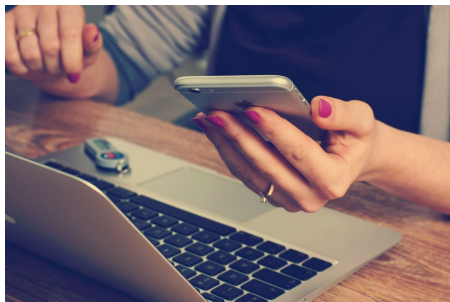
A man riding a dirt bike on a dirt track

Image Captioning: Failure Cases

Captions generated using [neuraltalk2](#)
All images are [CC0 Public domain](#): [fur](#)
[coat](#), [handstand](#), [spider web](#), [baseball](#)



*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*



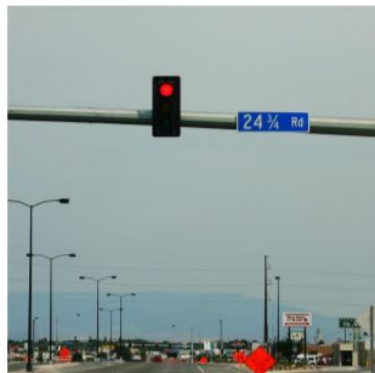
*A man in a
baseball uniform
throwing a ball*

Visual Question Answering (VQA)



Q: What endangered animal is featured on the truck?

- A: A bald eagle.
- A: A sparrow.
- A: A humming bird.
- A: A raven.



Q: Where will the driver go if turning right?

- A: Onto 24 $\frac{3}{4}$ Rd.
- A: Onto 25 $\frac{3}{4}$ Rd.
- A: Onto 23 $\frac{3}{4}$ Rd.
- A: Onto Main Street.



Q: When was the picture taken?

- A: During a wedding.
- A: During a bar mitzvah.
- A: During a funeral.
- A: During a Sunday church service

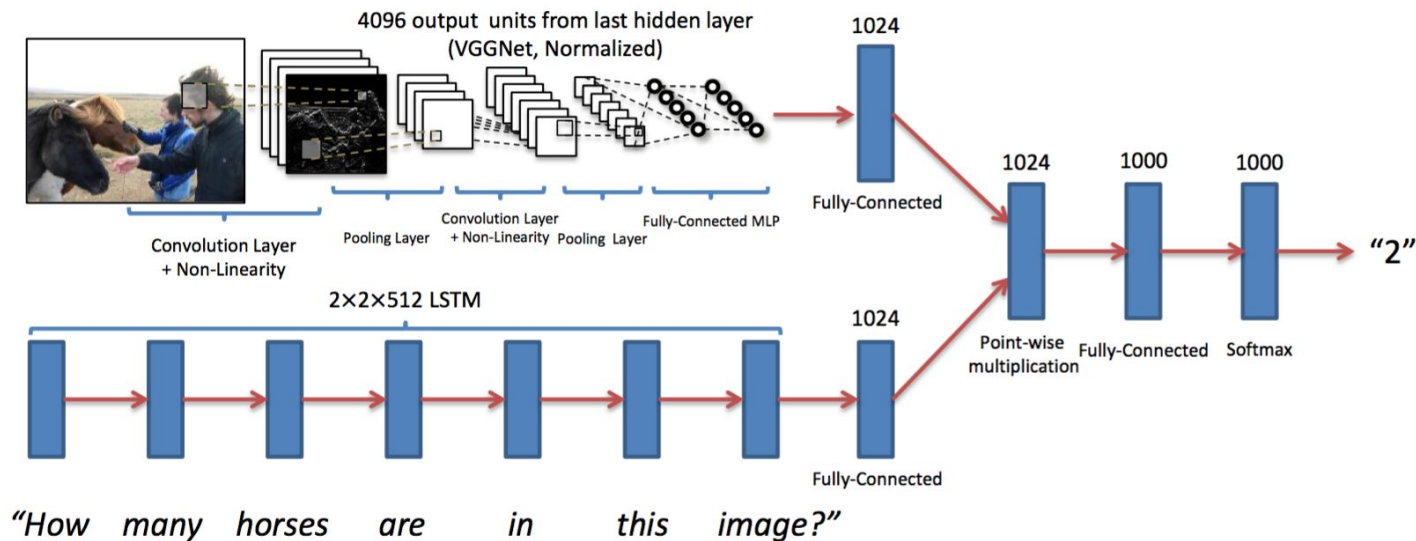


Q: Who is under the umbrella?

- A: Two women.
- A: A child.
- A: An old man.
- 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
Figure from Zhu et al, copyright IEEE 2016. Reproduced for educational purposes.

Visual Question Answering: RNNs with Attention



Agrawal et al, "Visual 7W: Grounded Question Answering in Images", CVPR 2015
Figures from Agrawal et al, copyright IEEE 2015. Reproduced for educational purposes.

Visual Dialog: Conversations about images



Das et al, "Visual Dialog", CVPR 2017
Figures from Das et al, copyright IEEE 2017. Reproduced with permission.

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.

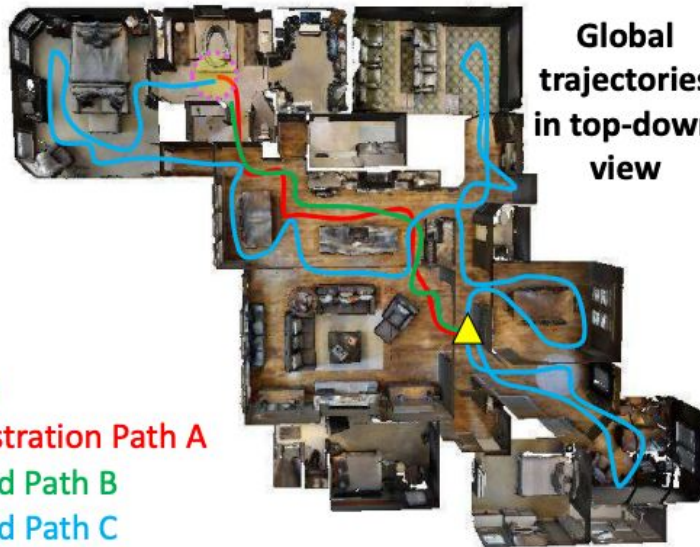
Instruction

Turn right and head towards the *kitchen*. Then turn left, pass a *table* and enter the *hallway*. Walk down the hallway and turn into the *entry* way to your right *without doors*. Stop in front of the *toilet*.

Local visual scene



Global trajectories in top-down view



Initial Position



Target Position



Demonstration Path A



Executed Path B



Executed Path C

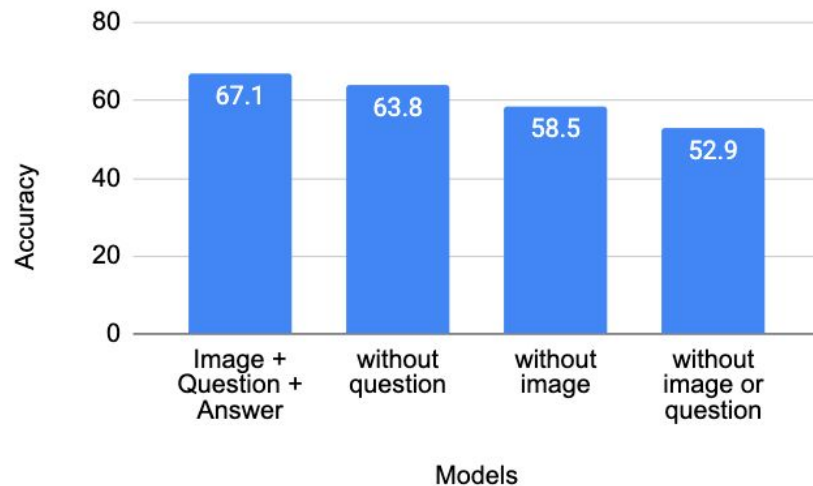
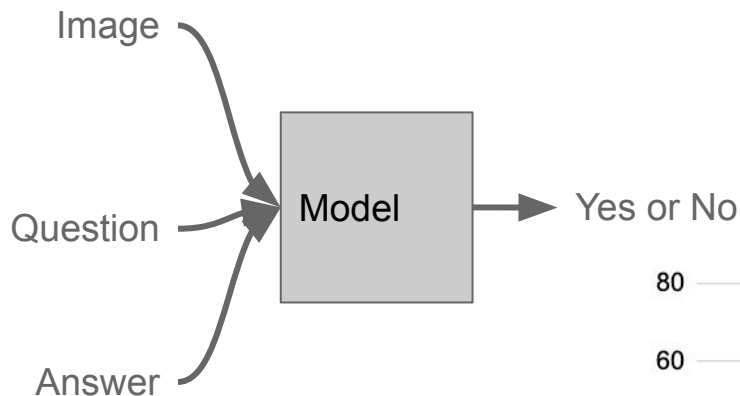
Wang et al, "Reinforced Cross-Modal Matching and Self-Supervised Imitation Learning for Vision-Language Navigation", CVPR 2018
Figures from Wang et al, copyright IEEE 2017. Reproduced with permission.

Visual Question Answering: Dataset Bias

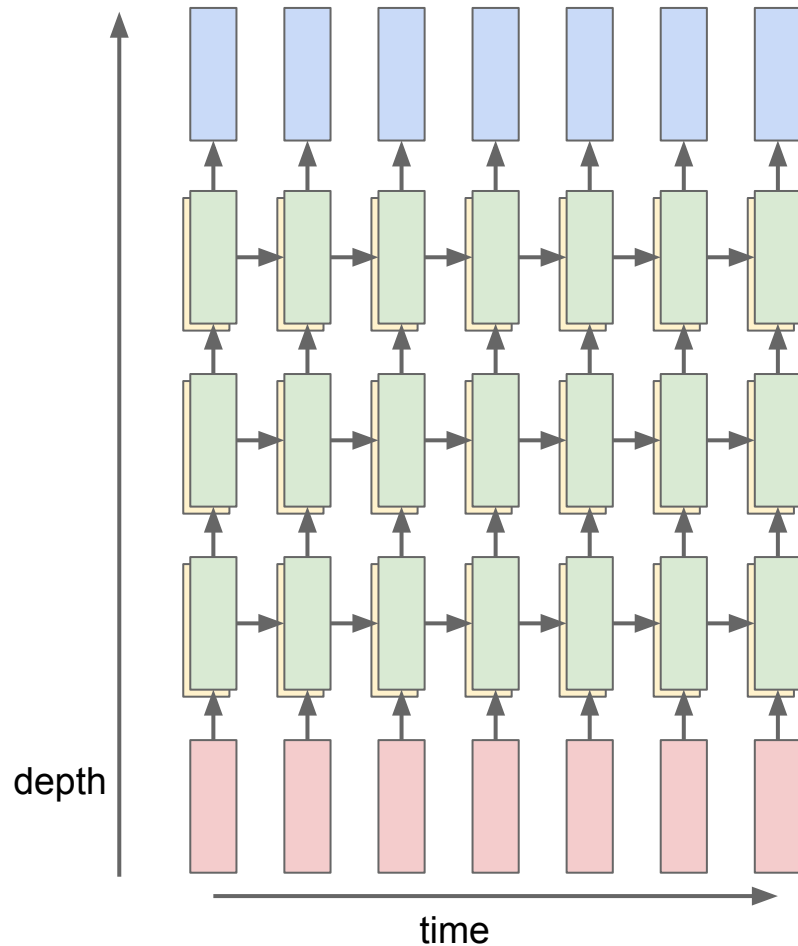


What is the dog playing with?

Frisbee



Multilayer RNNs



Long Short Term Memory (LSTM)

Vanilla RNN

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

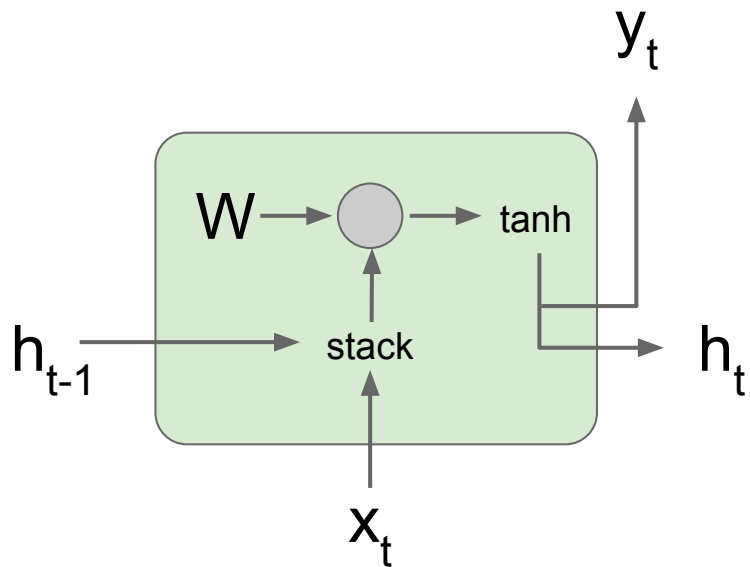
LSTM

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$
$$c_t = f \odot c_{t-1} + i \odot g$$
$$h_t = o \odot \tanh(c_t)$$

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

Vanilla RNN Gradient Flow

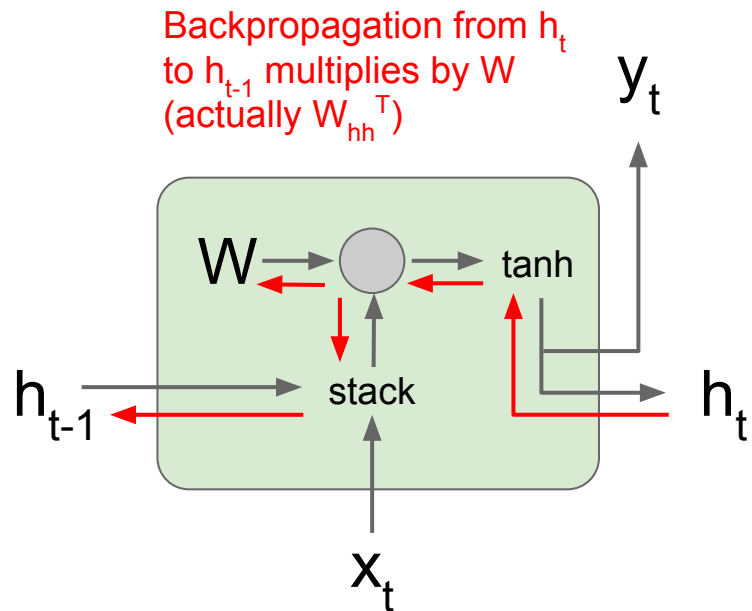
Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013



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

Vanilla RNN Gradient Flow

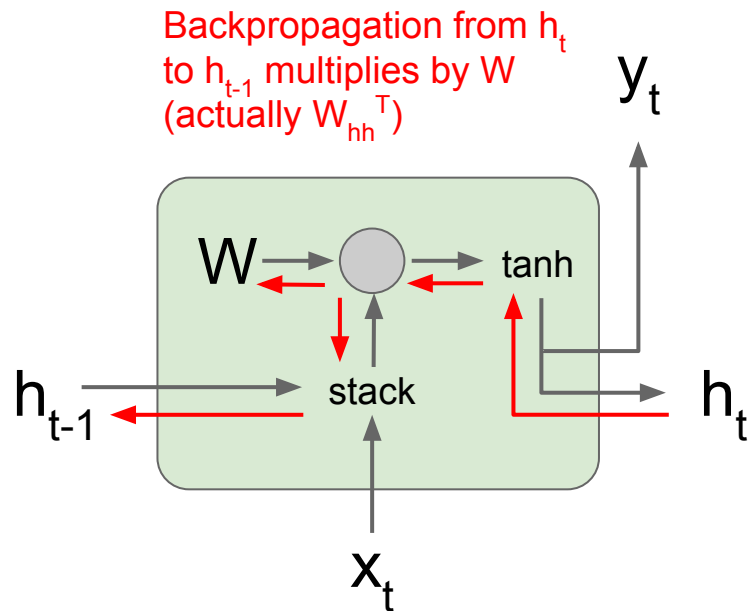
Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013



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

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

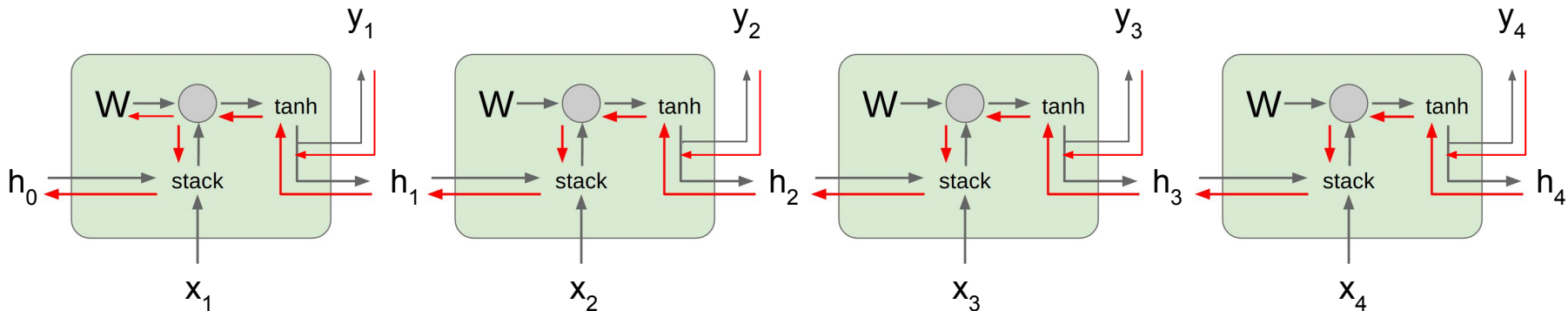


$$\begin{aligned}h_t &= \tanh(W_{hh}h_{t-1} + W_{hx}x_t) \\&= \tanh\left((W_{hh} \quad W_{hx}) \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right) \\&= \tanh\left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)\end{aligned}$$

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

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

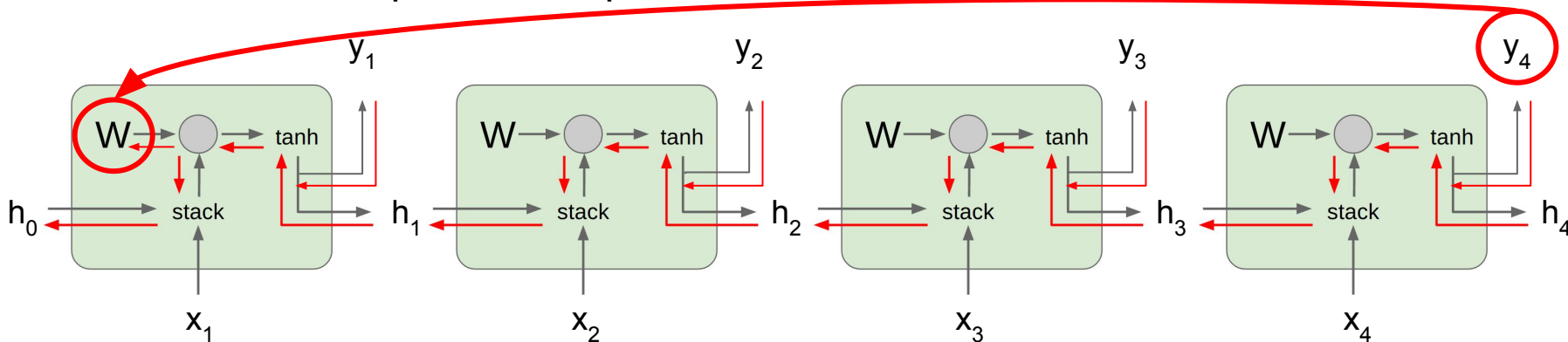


$$\frac{\partial L}{\partial W} = \sum_{t=1}^T \frac{\partial L_t}{\partial W}$$

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



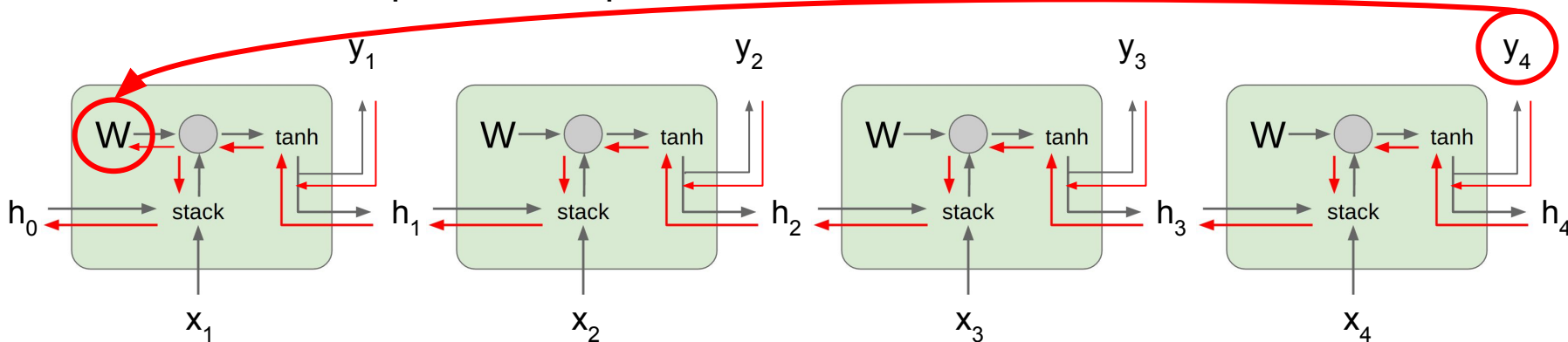
$$\frac{\partial L}{\partial W} = \sum_{t=1}^T \frac{\partial L_t}{\partial W}$$

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

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



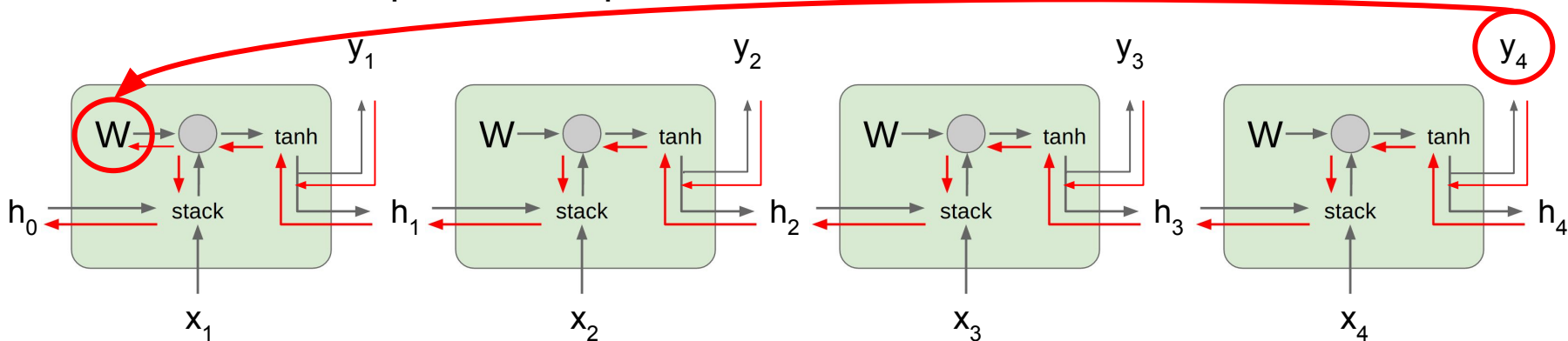
$$\frac{\partial L}{\partial W} = \sum_{t=1}^T \frac{\partial L_t}{\partial W}$$

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

Vanilla RNN Gradient Flow

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



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

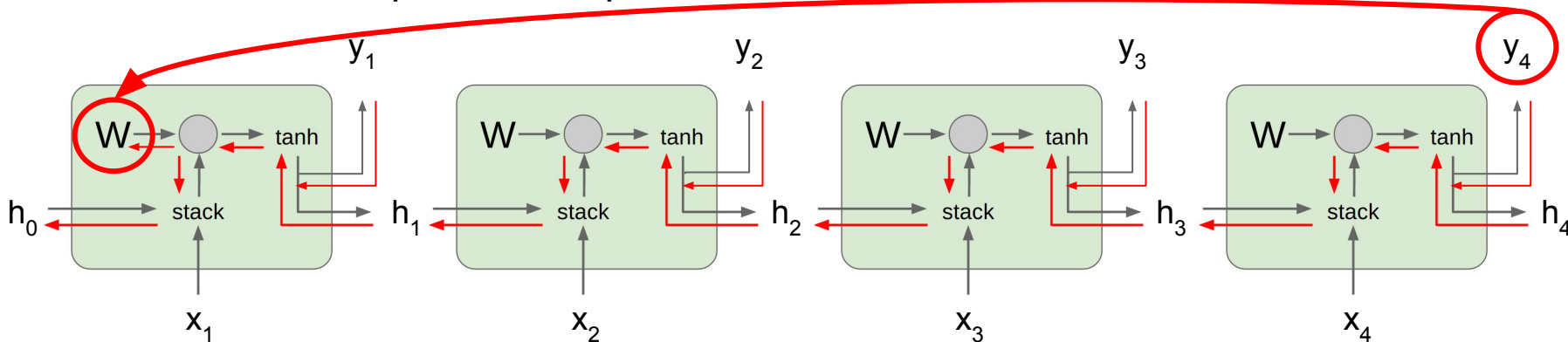
$$\frac{\partial h_t}{\partial h_{t-1}} = \tanh'(W_{hh}h_{t-1} + W_{xh}x_t)W_{hh}$$

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

Vanilla RNN Gradient Flow

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



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

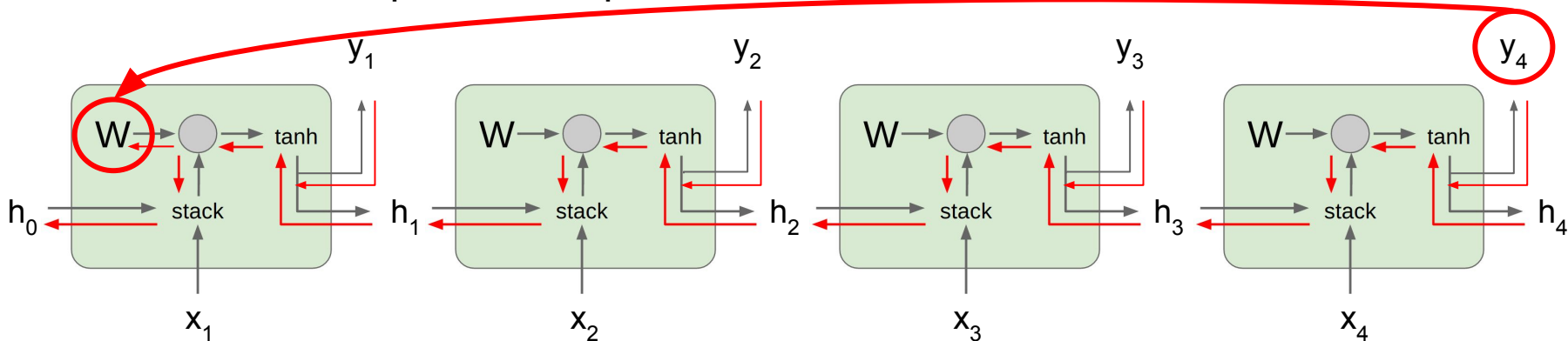
Almost always < 1
Vanishing gradients

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

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



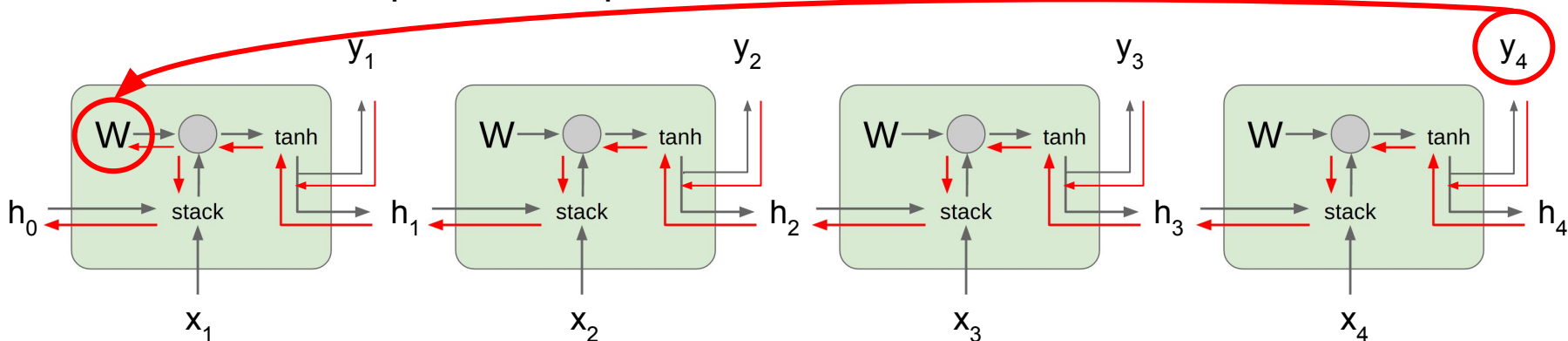
$$\frac{\partial L}{\partial W} = \sum_{t=1}^T \frac{\partial L_t}{\partial W}$$

What if we assumed no non-linearity?

Vanilla RNN Gradient Flow

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



What if we assumed no non-linearity?

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

Largest singular value > 1 :
Exploding gradients

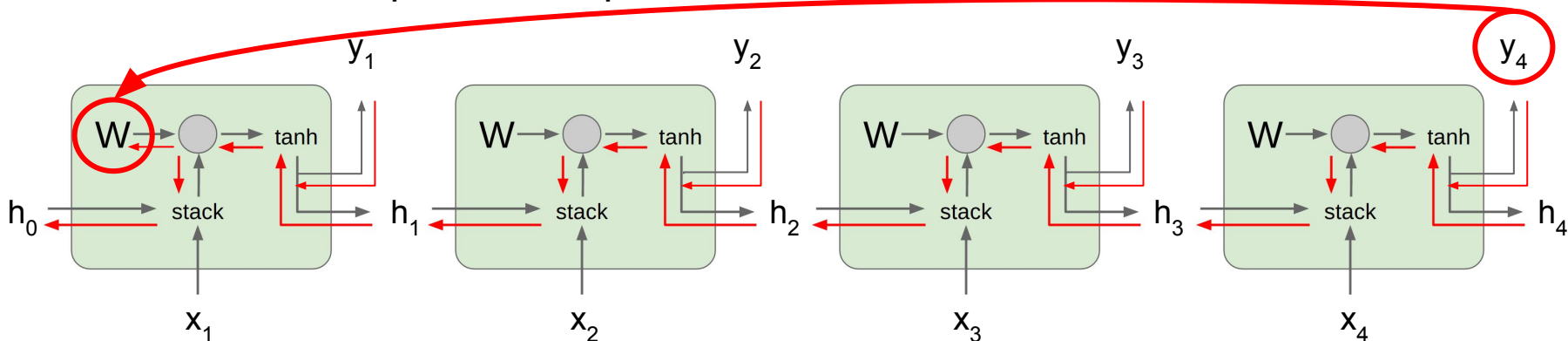
$$\frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial h_T} \boxed{W_{hh}^{T-1}} \frac{\partial h_1}{\partial W}$$

Largest singular value < 1 :
Vanishing gradients

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



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

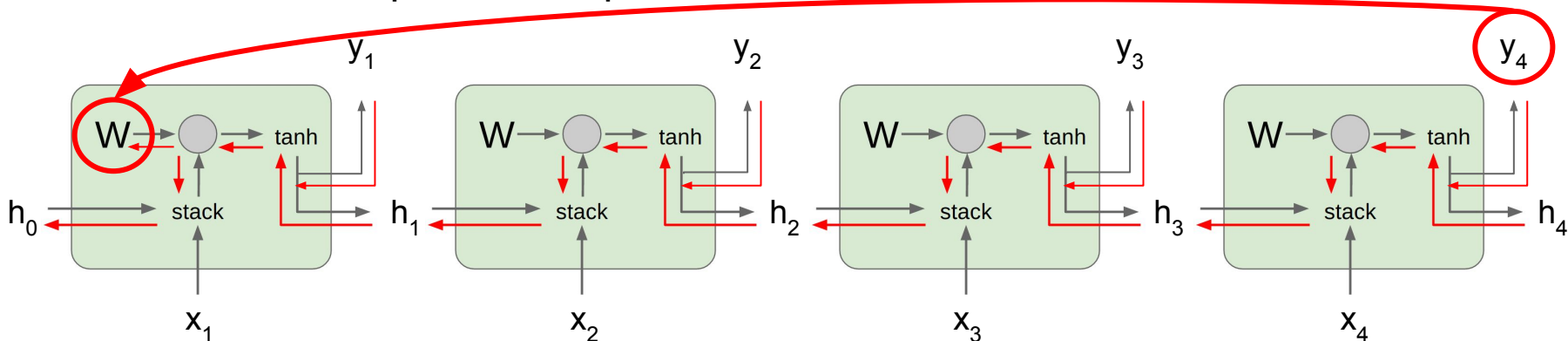
Gradient clipping:
Scale gradient if its norm is too big

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

Vanilla RNN Gradient Flow

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



What if we assumed no non-linearity?

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

Largest singular value > 1 :
Exploding gradients

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

Largest singular value < 1 :
Vanishing gradients

→ Change RNN architecture

Long Short Term Memory (LSTM)

Vanilla RNN

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

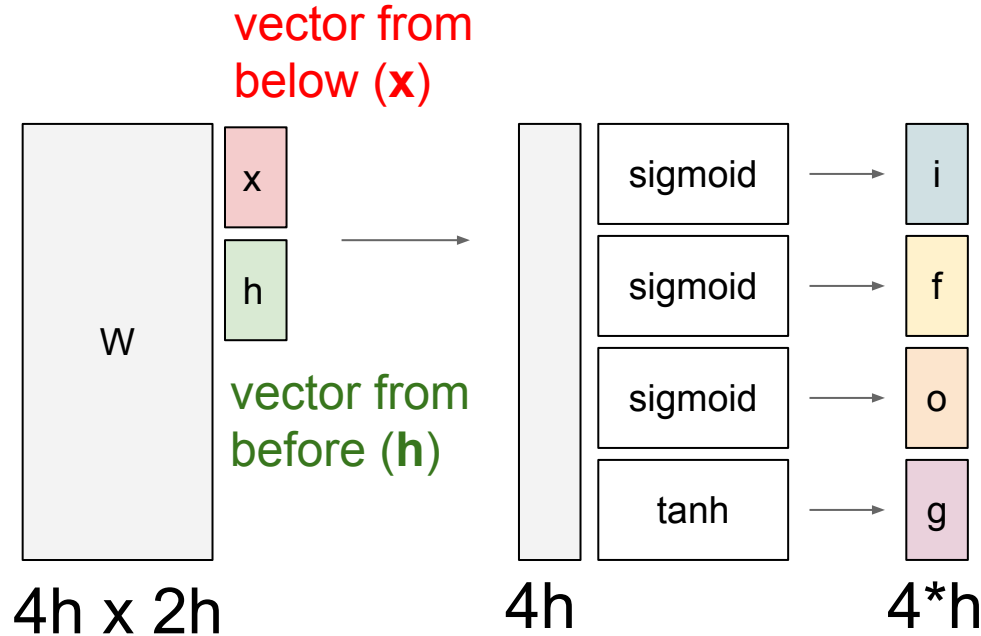
LSTM

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$
$$c_t = f \odot c_{t-1} + i \odot g$$
$$h_t = o \odot \tanh(c_t)$$

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

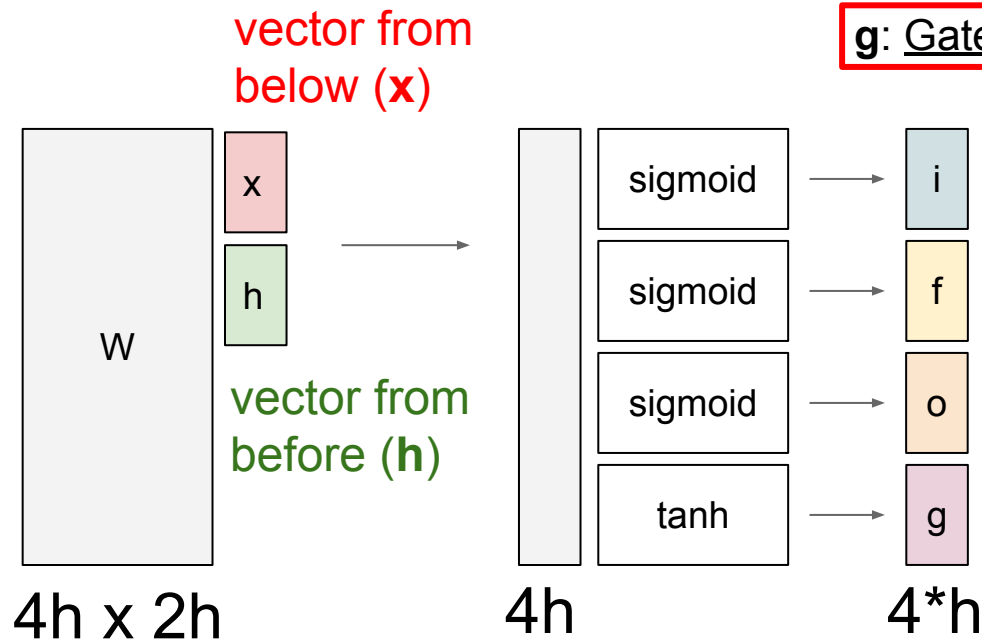
Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]



Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]



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

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

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

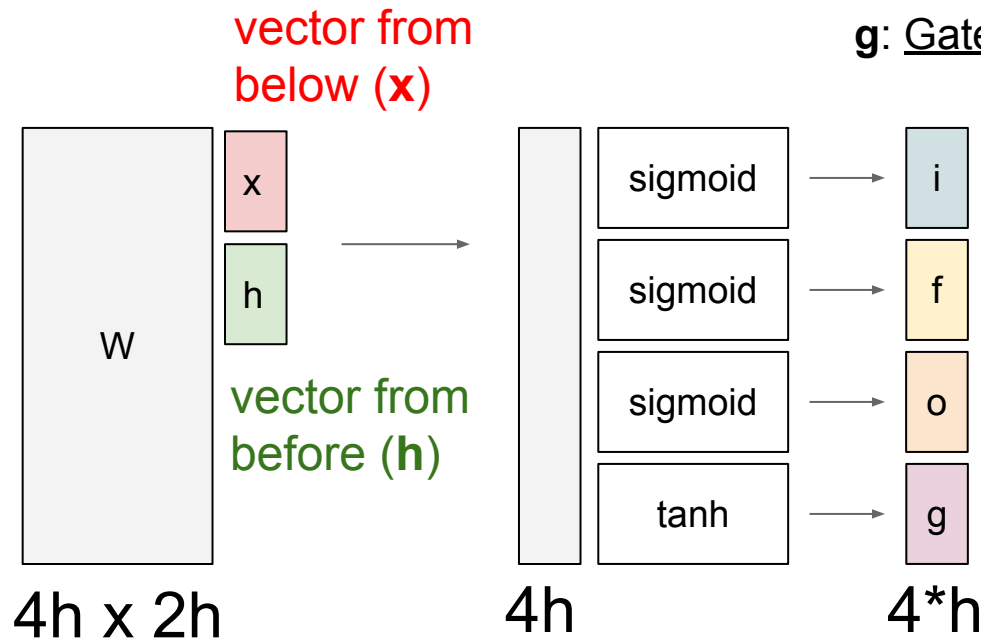
$$h_t = o \odot \tanh(c_t)$$

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



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

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

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

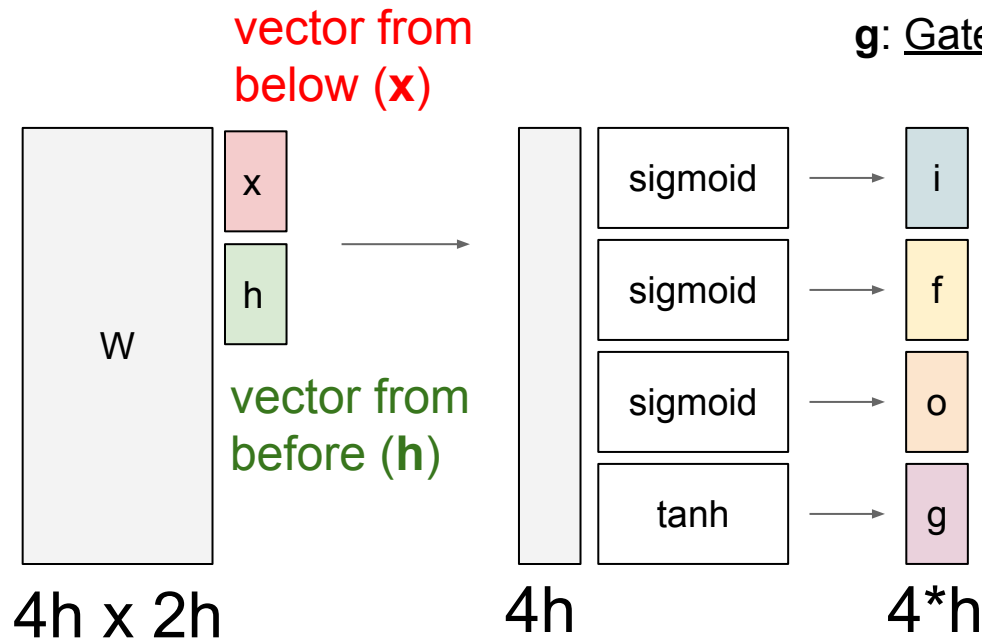
Long Short Term Memory (LSTM)

[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



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

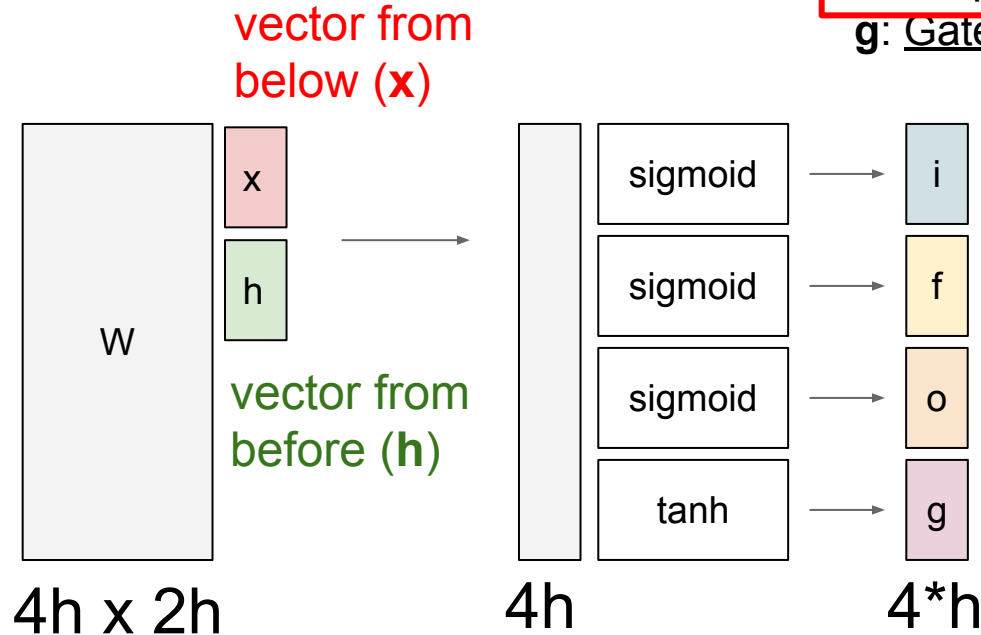
$$c_t = f \odot c_{t-1} + i \odot g$$

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

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



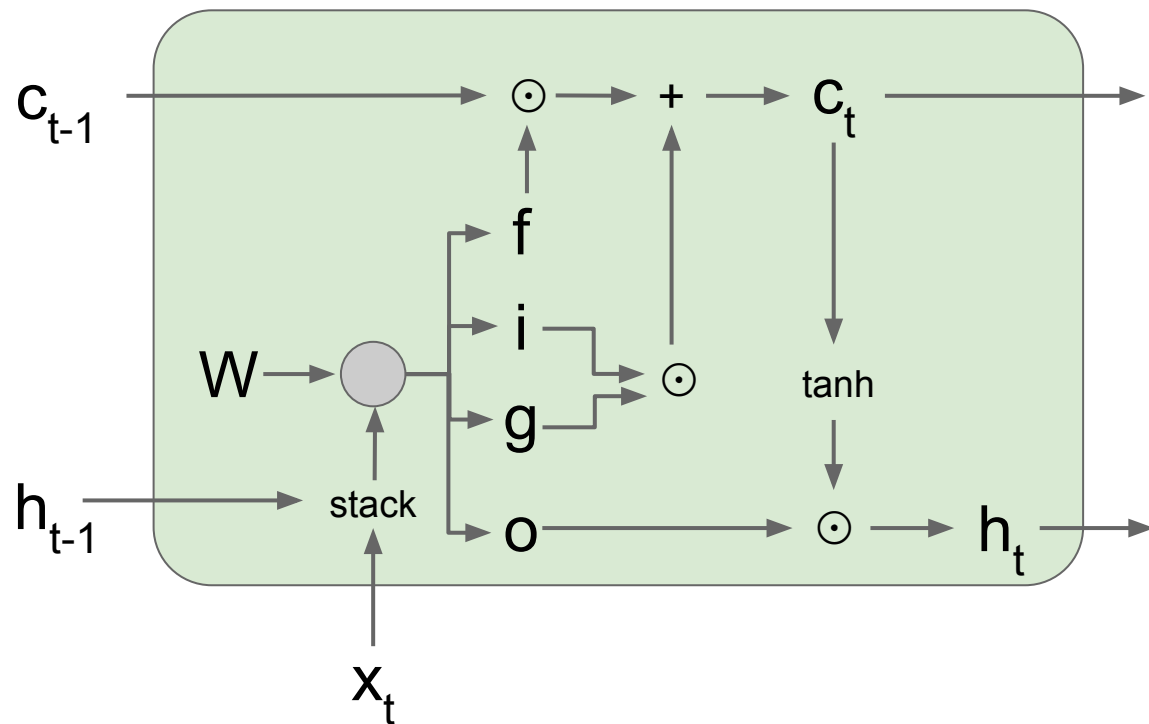
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

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

$$h_t = \boxed{o} \odot \tanh(c_t)$$

Long Short Term Memory (LSTM)

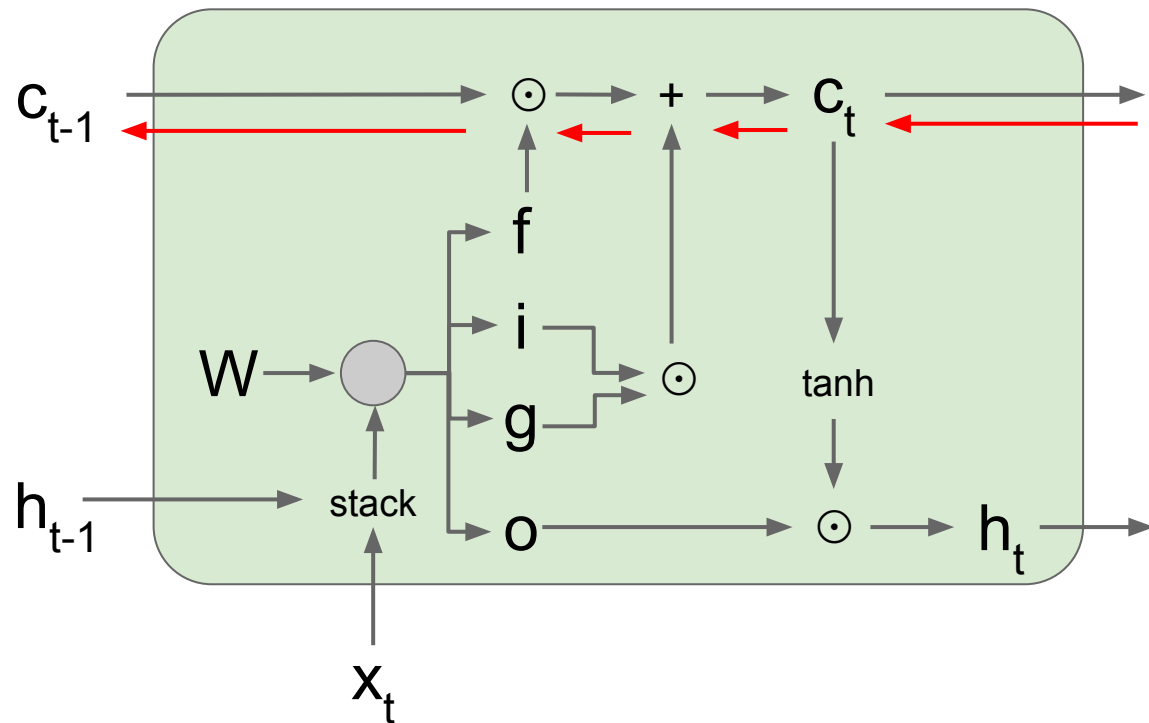
[Hochreiter et al., 1997]



$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$
$$c_t = f \odot c_{t-1} + i \odot g$$
$$h_t = o \odot \tanh(c_t)$$

Long Short Term Memory (LSTM): Gradient Flow

[Hochreiter et al., 1997]



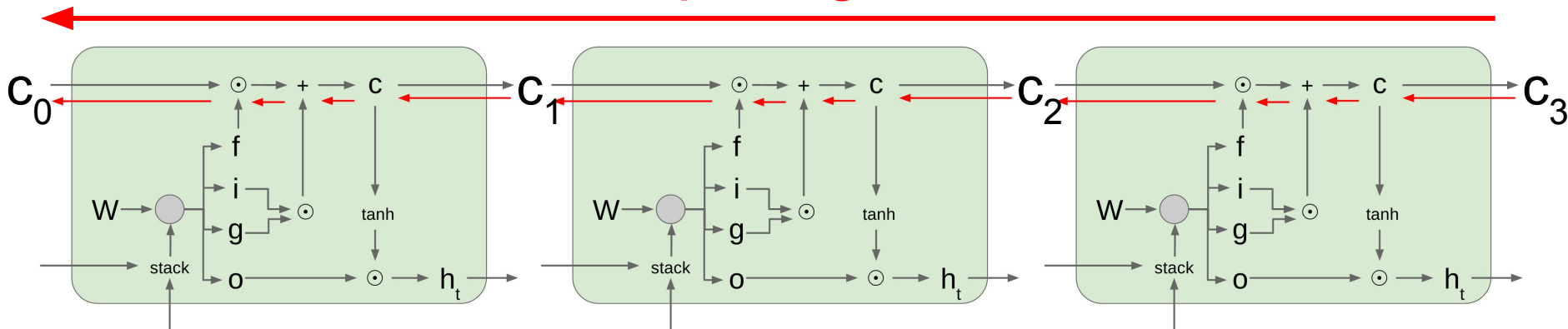
Backpropagation from c_t to c_{t-1} only elementwise multiplication by f , no matrix multiply by W

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$
$$c_t = f \odot c_{t-1} + i \odot g$$
$$h_t = o \odot \tanh(c_t)$$

Long Short Term Memory (LSTM): Gradient Flow

[Hochreiter et al., 1997]

Uninterrupted gradient flow!



Notice that the gradient contains the f gate's vector of activations

- allows better control of gradients values, using suitable parameter updates of the forget gate.

Also notice that are added through the f , i , g , and o gates

- better balancing of gradient values

Do LSTMs solve the vanishing gradient problem?

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

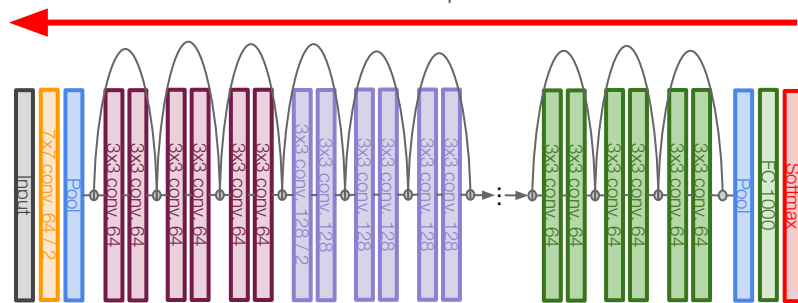
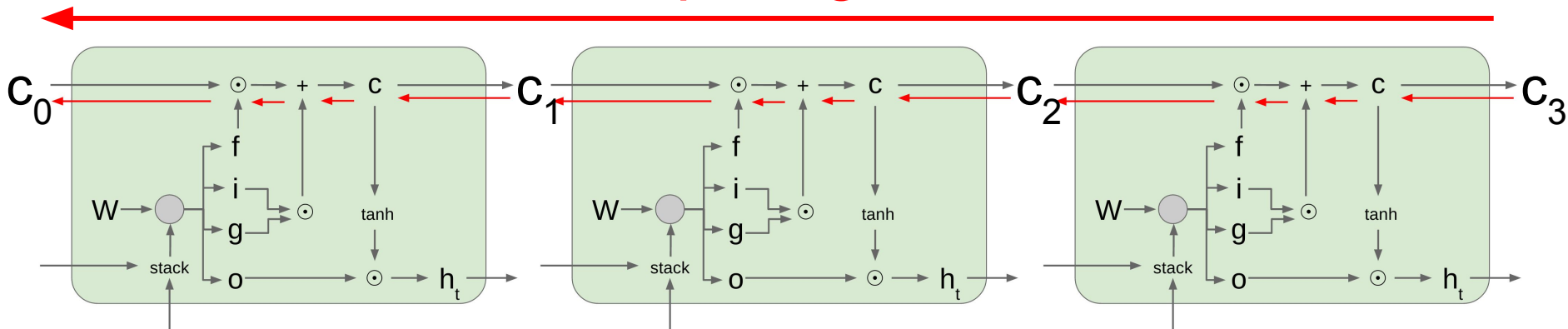
- e.g. **if the $f = 1$ and the $i = 0$** , then the information of that cell is preserved indefinitely.
- By contrast, it's harder for vanilla RNN to learn a recurrent weight matrix W_h that preserves info in hidden state

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

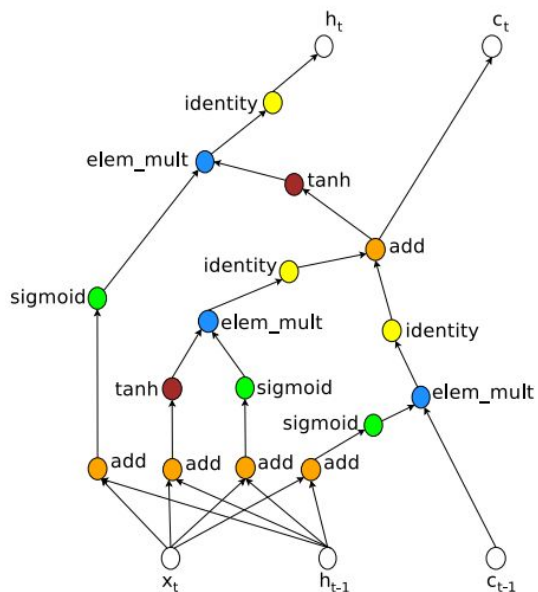
[Hochreiter et al., 1997]

Uninterrupted gradient flow!

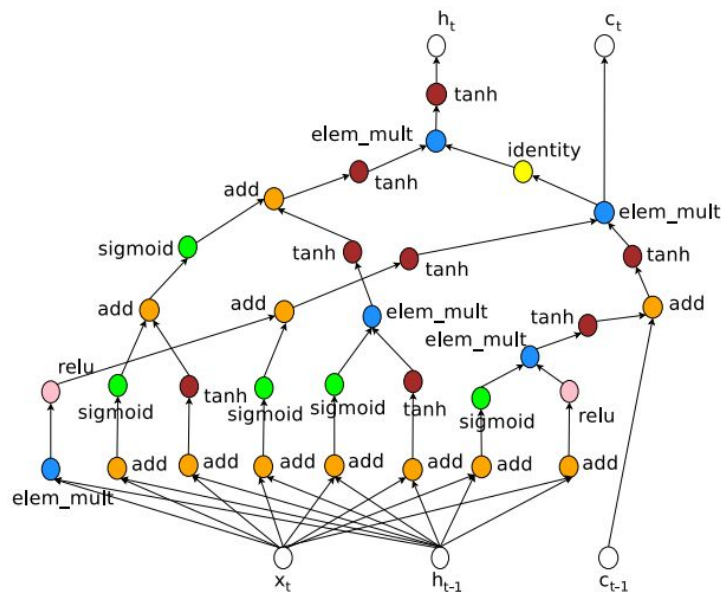


Similar to ResNet!

Neural Architecture Search for RNN architectures



LSTM cell



Cell they found

Zoph et al, "Neural Architecture Search with Reinforcement Learning", ICLR 2017
Figures copyright Zoph et al, 2017. Reproduced with permission.

Other RNN Variants

GRU [*Learning phrase representations using rnn encoder-decoder for statistical machine translation*, Cho et al. 2014]

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

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

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

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

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

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

MUT1:

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

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

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

MUT2:

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

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

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

MUT3:

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

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

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

Recurrence for Vision

- LSTM were a good default choice until this year
- Use variants like GRU if you want faster compute and less parameters
- Use transformers (next lecture) as they are dominating NLP models
 - almost everyday there is a new vision transformer model

Su et al. "Vi-bert: Pre-training of generic visual-linguistic representations." ICLR 2020

Lu et al. "Vilbert: Pretraining task-agnostic visiolinguistic representations for vision-and-language tasks." NeurIPS 2019

Li et al. "Visualbert: A simple and performant baseline for vision and language." *arXiv* 2019

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: their additive interactions improve gradient flow
- Backward flow of gradients in RNN can explode or vanish. Exploding is controlled with gradient clipping. Vanishing is controlled with additive interactions (LSTM)
- Better/simpler architectures are a hot topic of current research, as well as new paradigms for reasoning over sequences
- Better understanding (both theoretical and empirical) is needed.

Next time: Midterm!