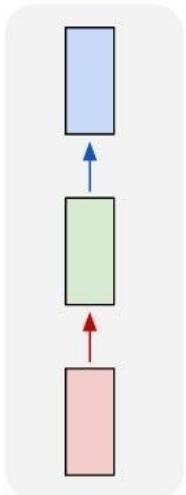


# Recurrent Neural Networks

Slides adapted from Stanford CS231n  
Mingchen Gao

# “Vanilla” Neural Network

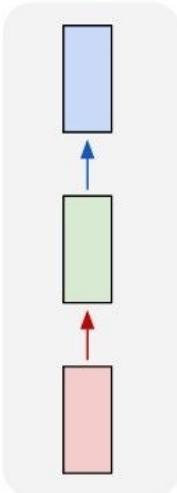
one to one



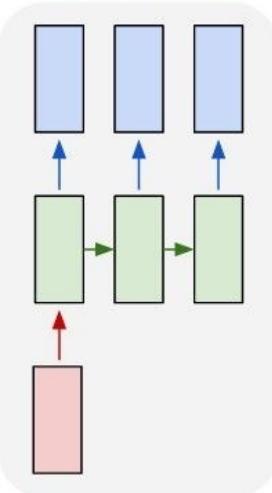
**Vanilla Neural Networks**

# Recurrent Neural Networks: Process Sequences

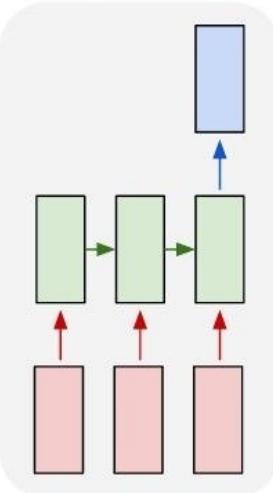
one to one



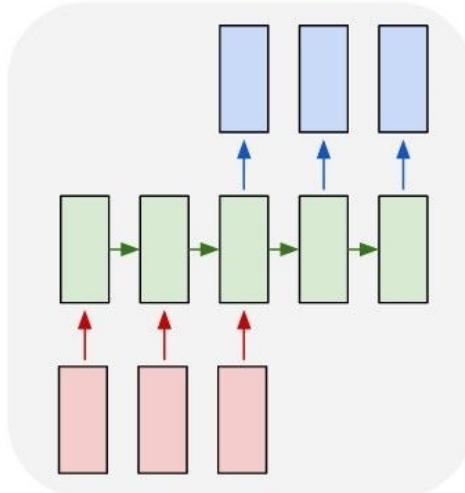
one to many



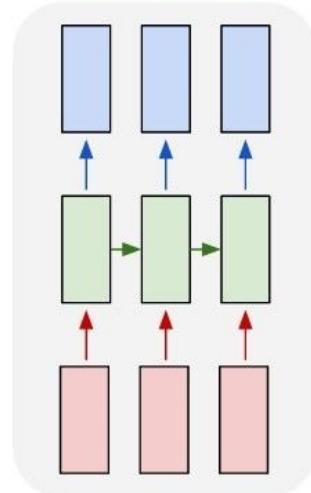
many to one



many to many



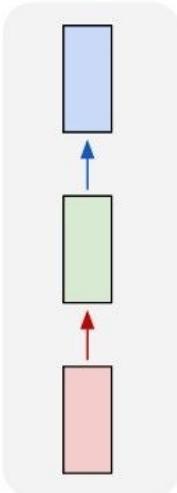
many to many



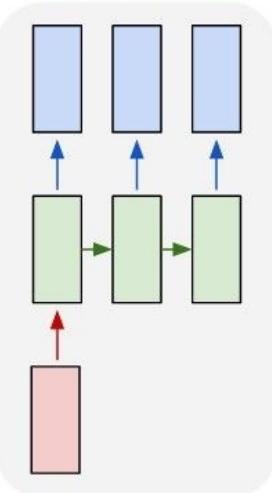
→  
e.g. **Image Captioning**  
image -> sequence of words

# Recurrent Neural Networks: Process Sequences

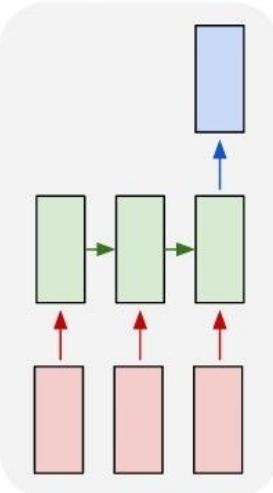
one to one



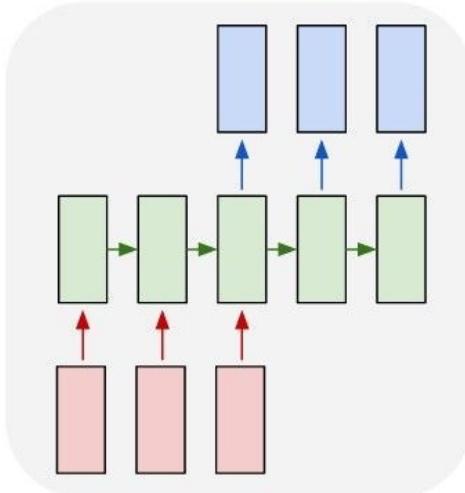
one to many



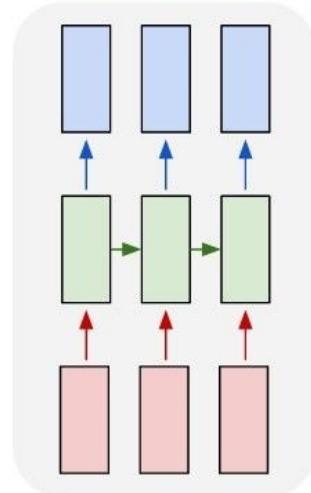
many to one



many to many



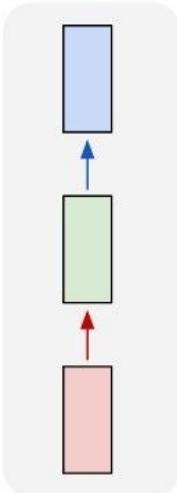
many to many



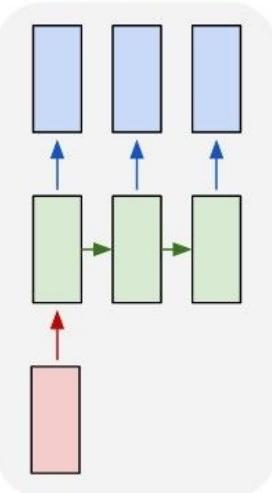
e.g. **action prediction**  
sequence of video frames -> action class

# Recurrent Neural Networks: Process Sequences

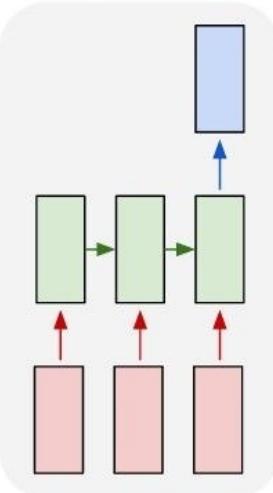
one to one



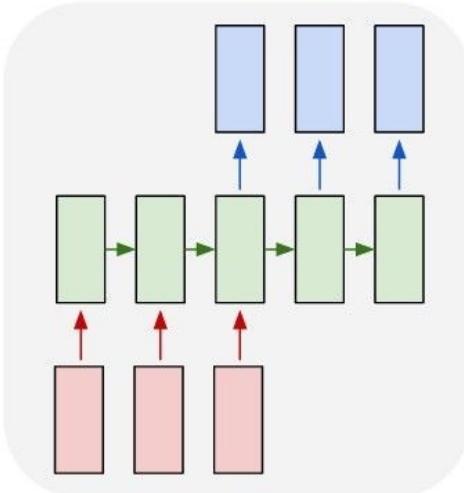
one to many



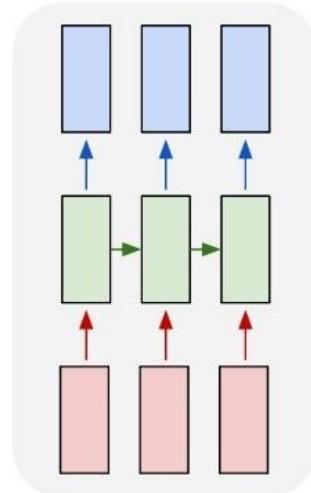
many to one



many to many



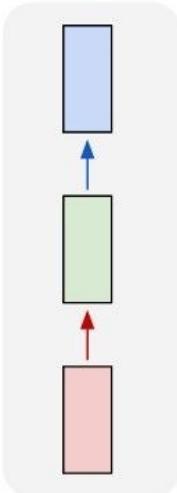
many to many



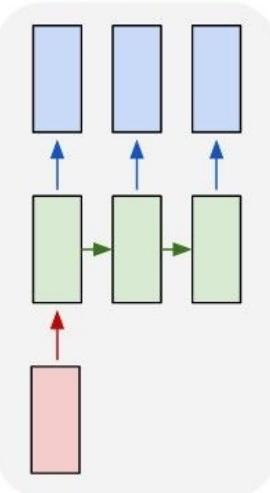
↑  
E.g. **Video Captioning**  
Sequence of video frames ->  
caption

# Recurrent Neural Networks: Process Sequences

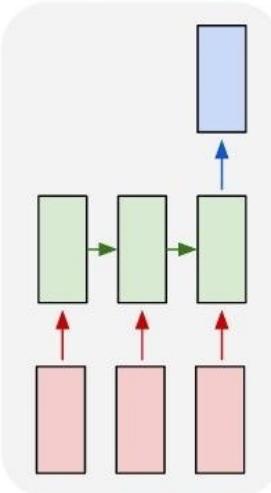
one to one



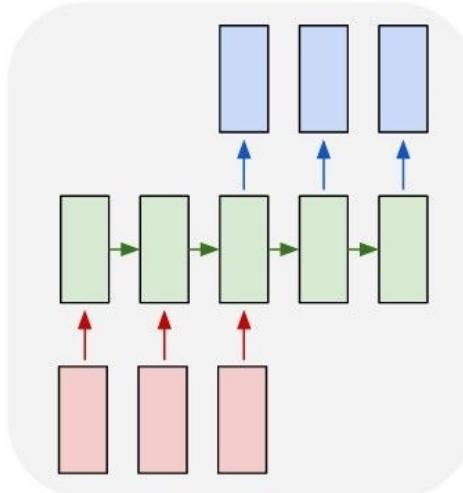
one to many



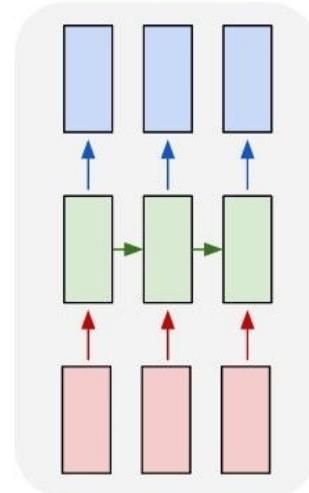
many to one



many to many



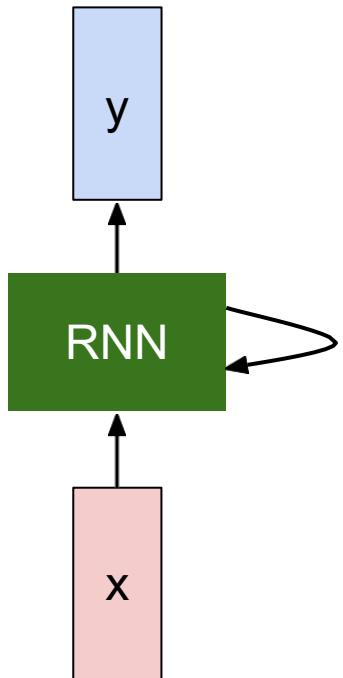
many to many



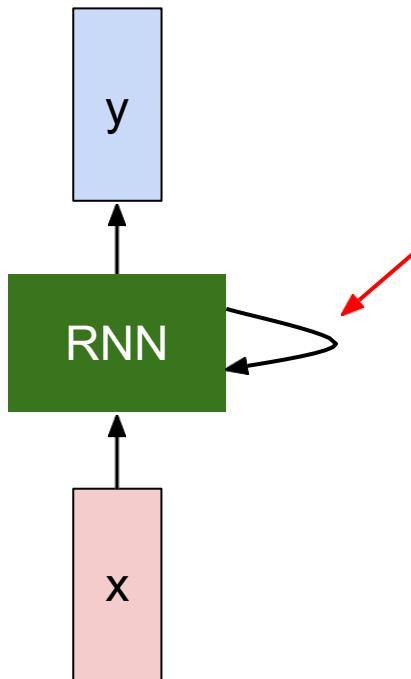
e.g. **Video classification on frame level**



# Recurrent Neural Network

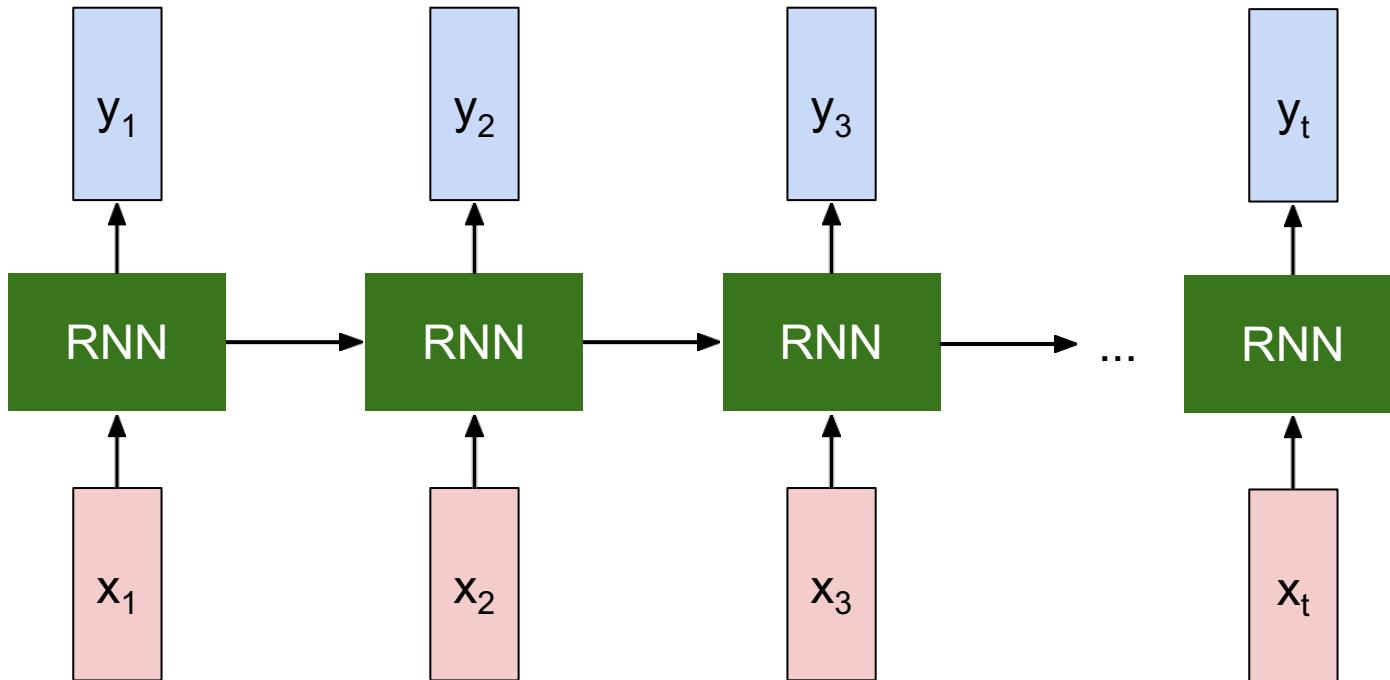


# Recurrent Neural Network



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

# Recurrent Neural Network

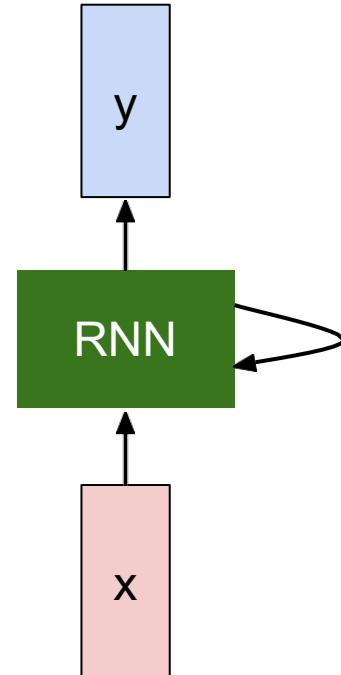


# Recurrent Neural Network

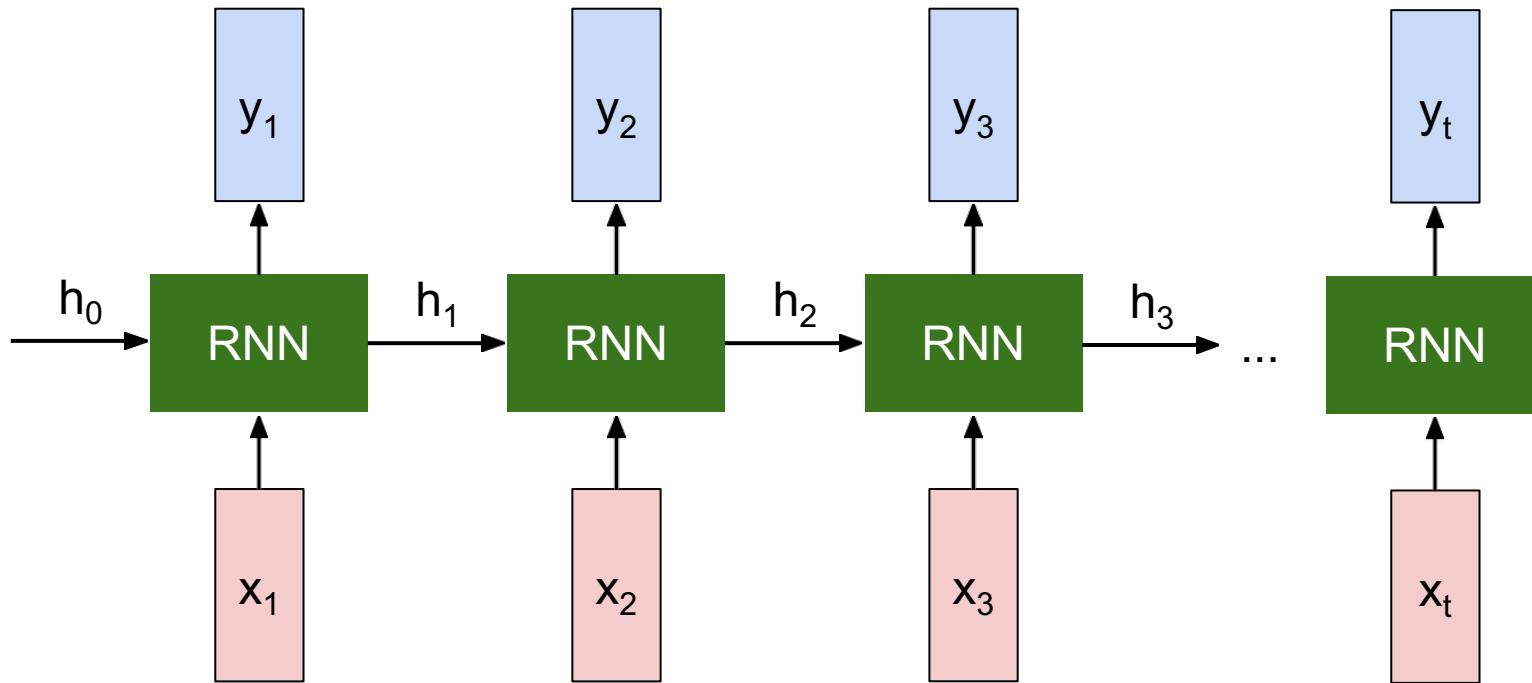
We can process a sequence of vectors  $\mathbf{x}$  by applying a **recurrence formula** at every time step:

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

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



# Recurrent Neural Network

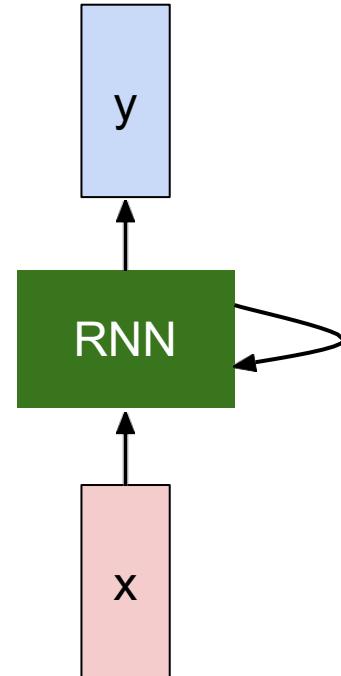


# Recurrent Neural Network

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

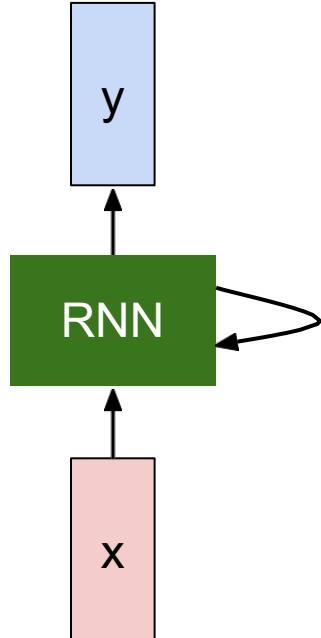
$$h_t = f_W(h_{t-1}, x_t)$$

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  $\mathbf{h}$ :



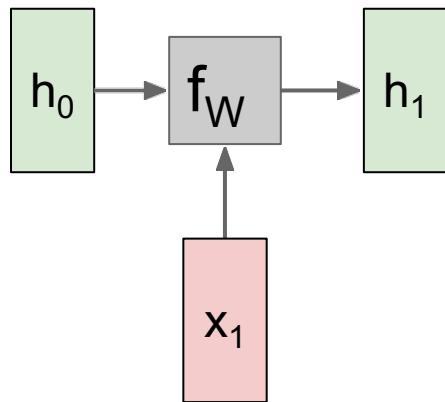
$$\mathbf{h}_t = f_W(\mathbf{h}_{t-1}, \mathbf{x}_t)$$

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

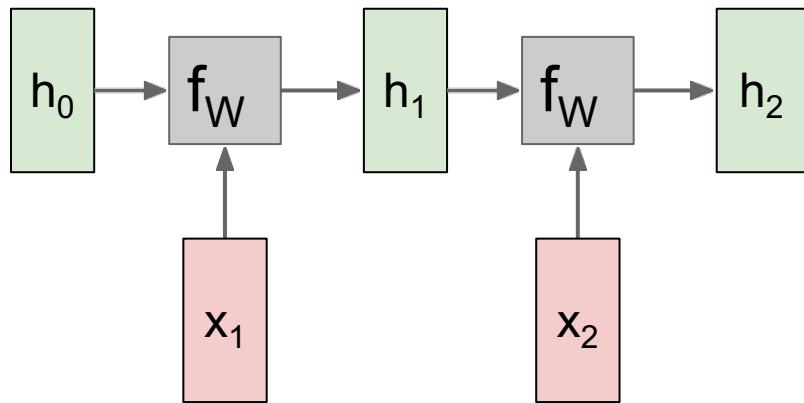
$$y_t = W_{hy}\mathbf{h}_t$$

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

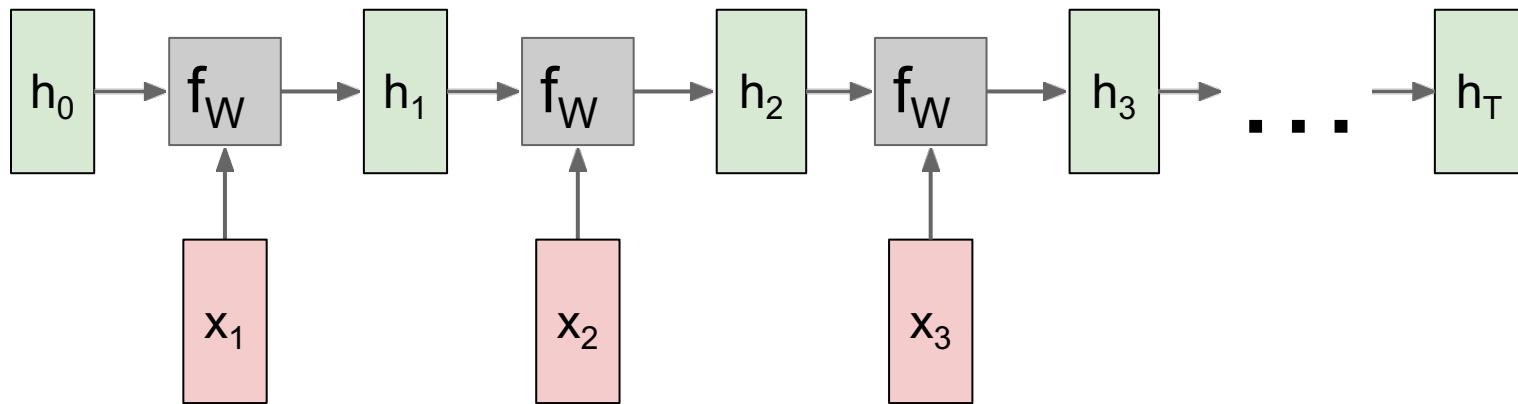
# RNN: Computational Graph



# RNN: Computational Graph

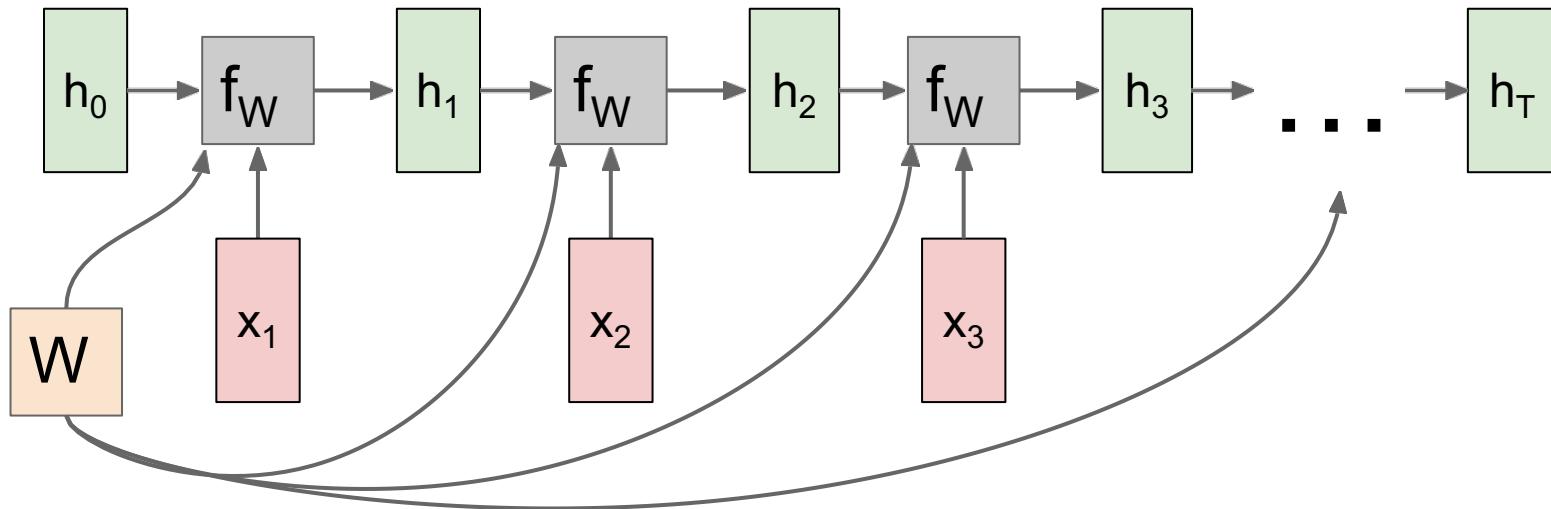


# RNN: Computational Graph

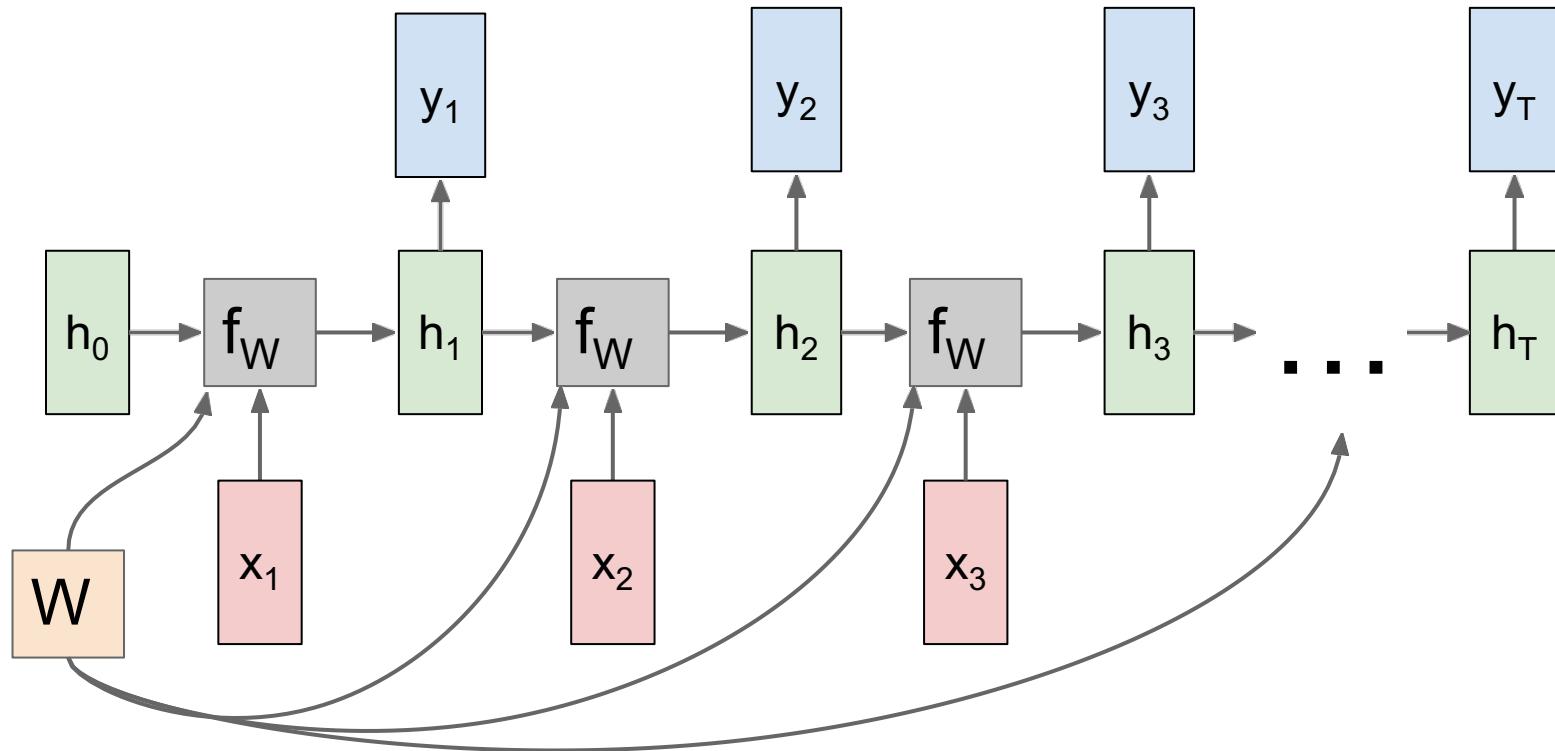


# RNN: Computational Graph

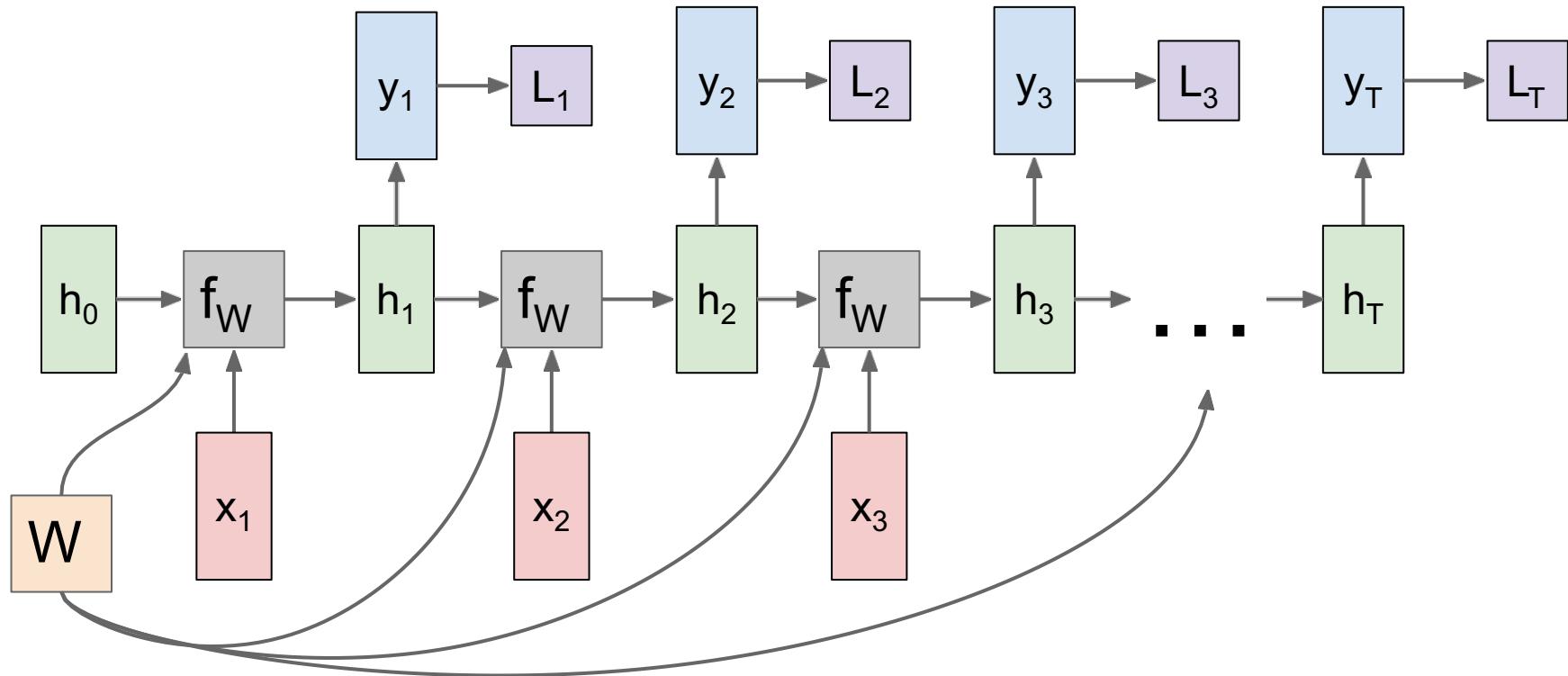
Re-use the same weight matrix at every time-step



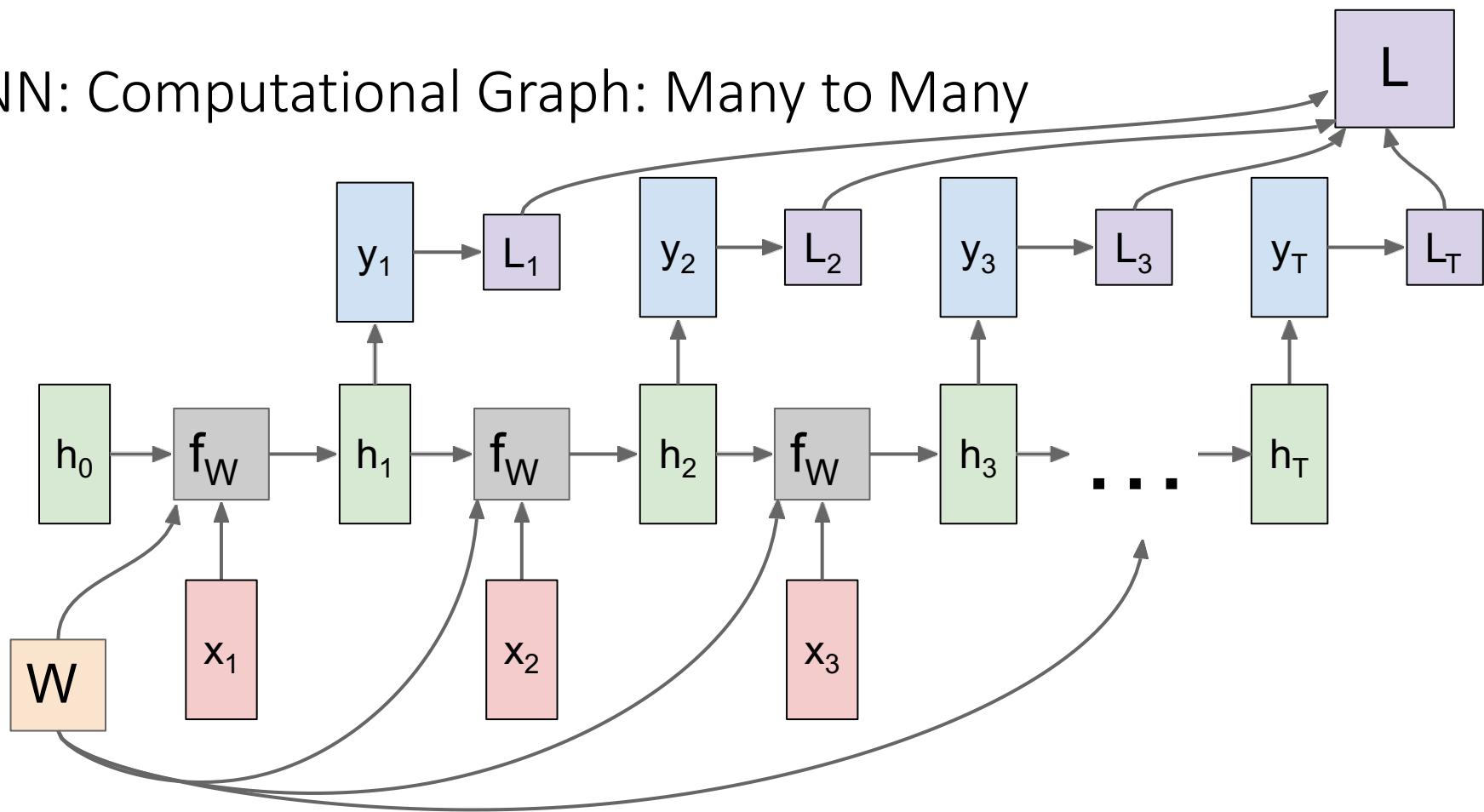
# RNN: Computational Graph: Many to Many



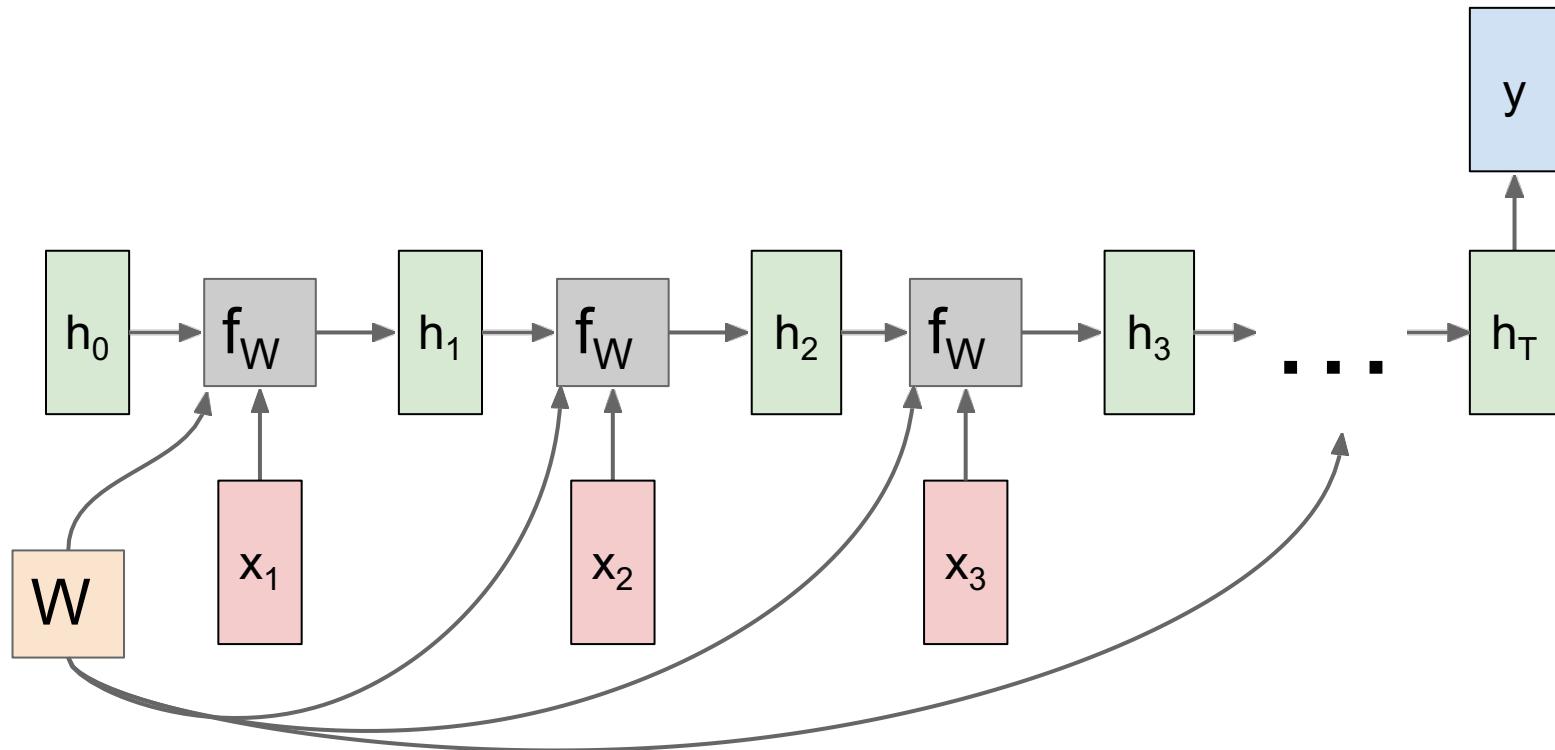
# RNN: Computational Graph: Many to Many



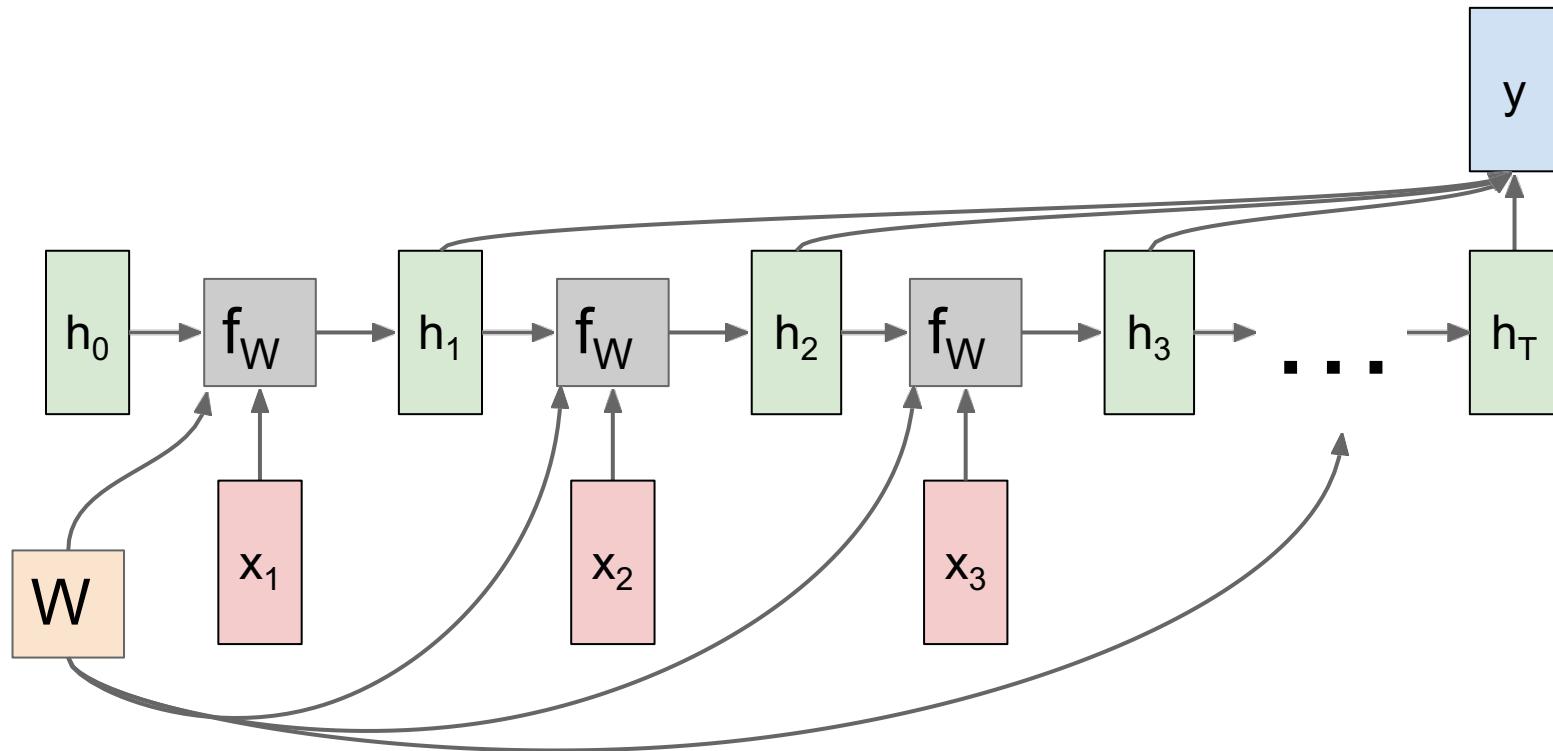
# RNN: Computational Graph: Many to Many



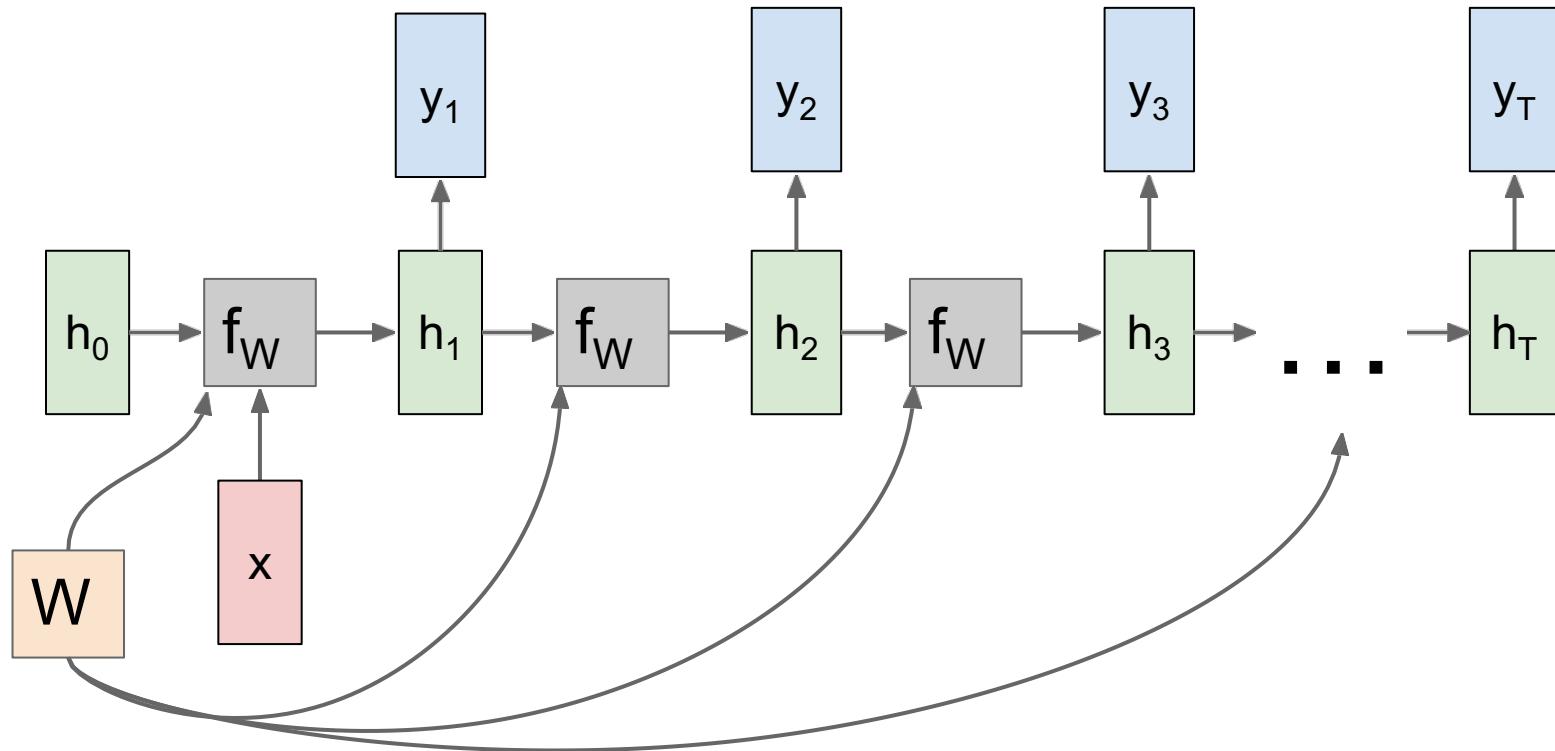
# RNN: Computational Graph: Many to One



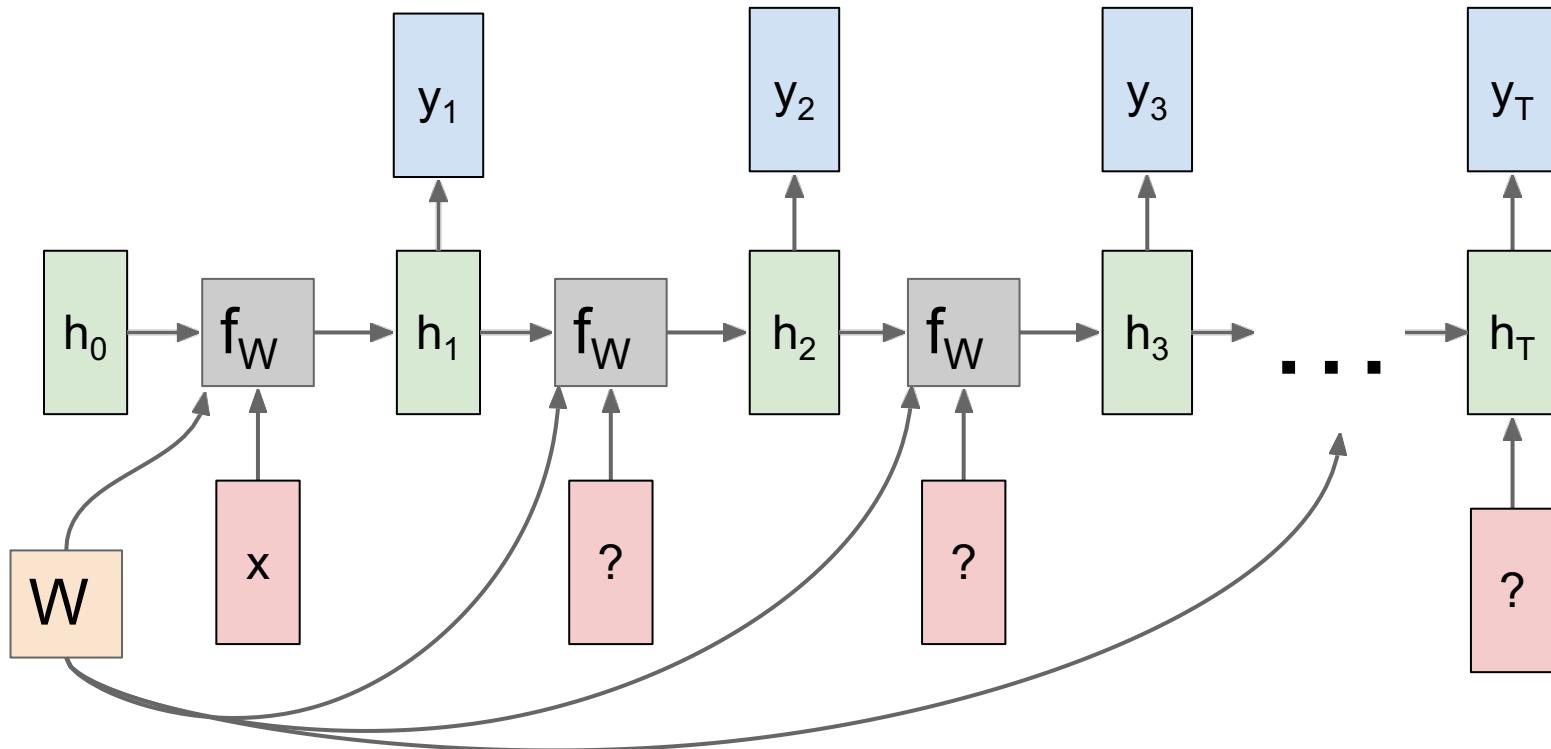
# RNN: Computational Graph: Many to One



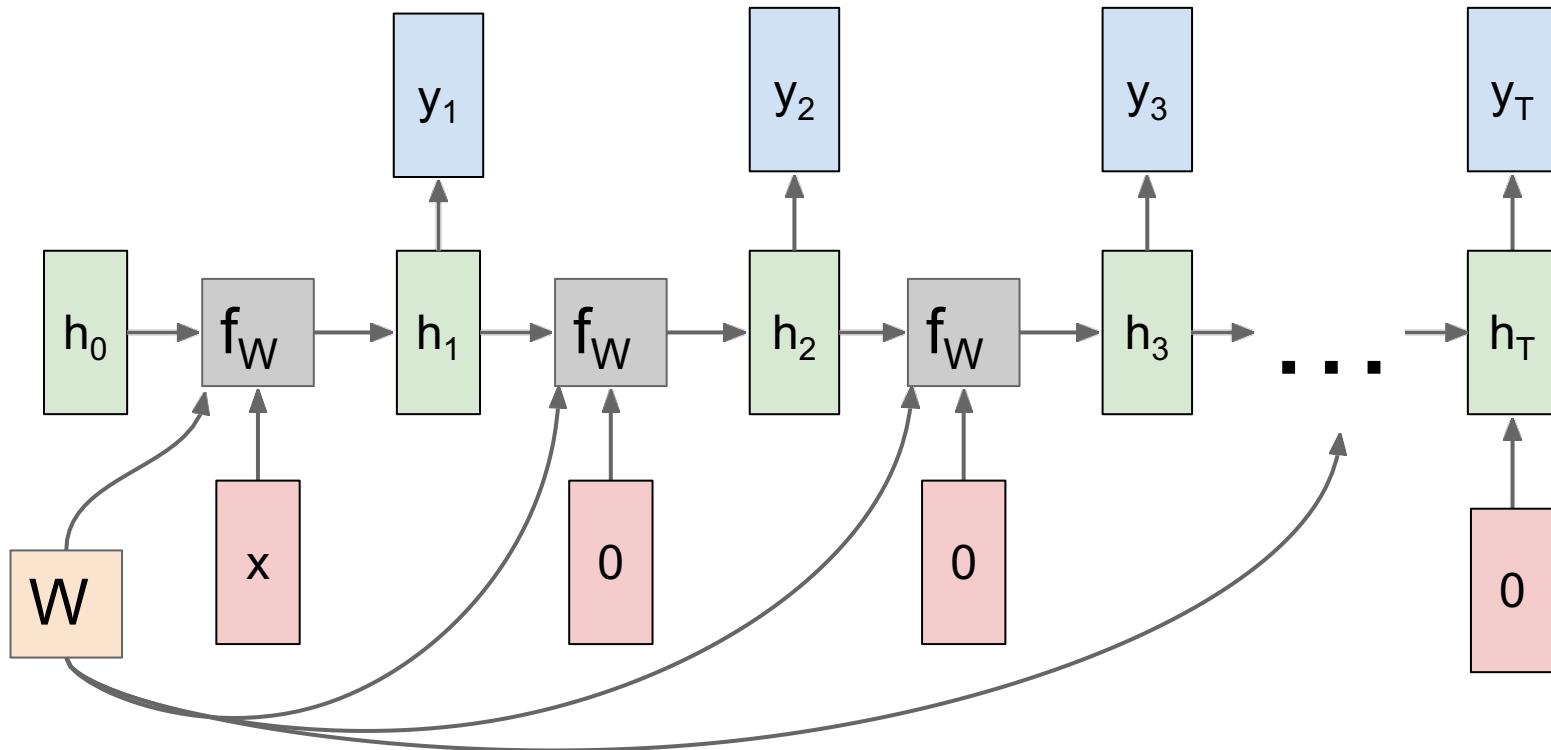
# RNN: Computational Graph: One to Many



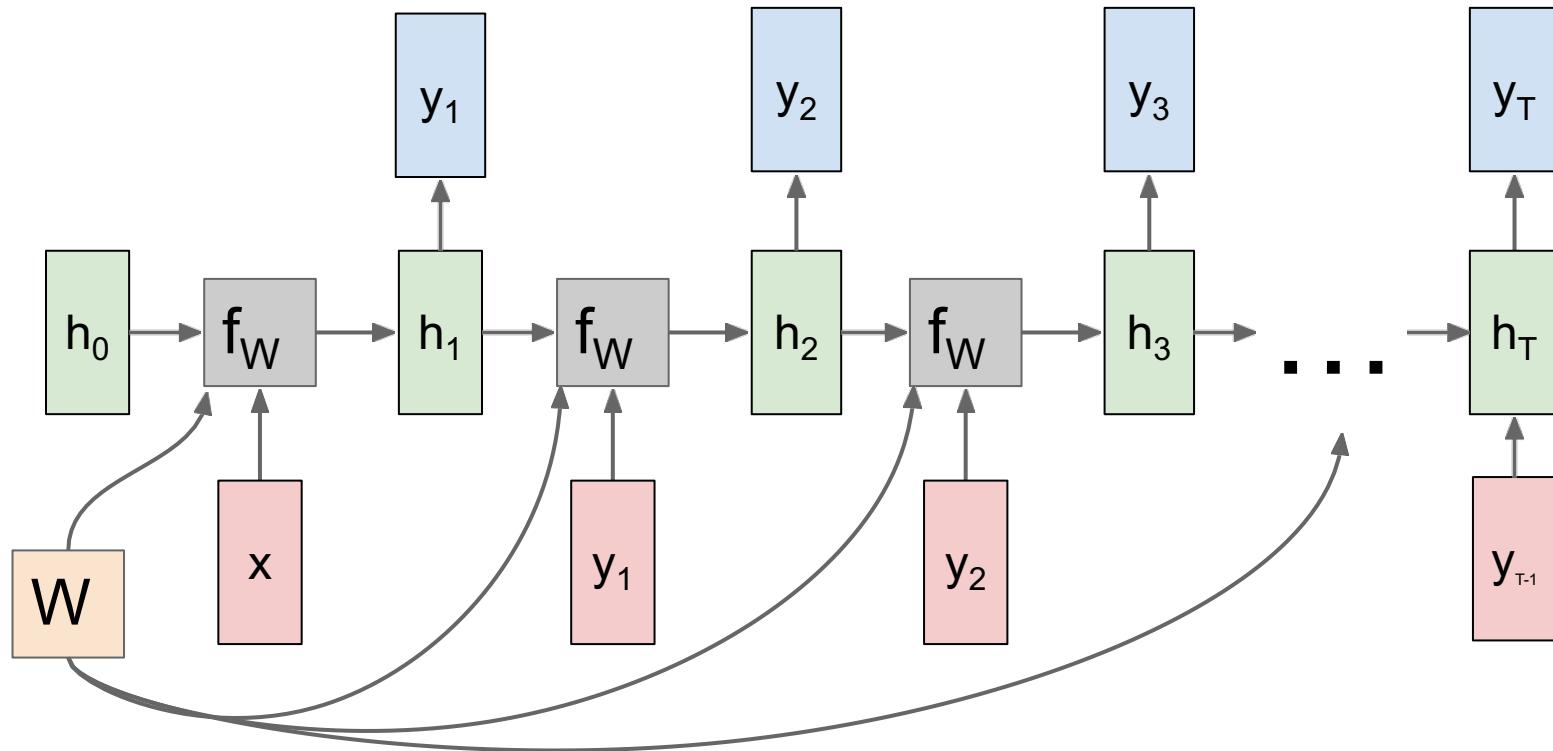
# RNN: Computational Graph: One to Many



# RNN: Computational Graph: One to Many

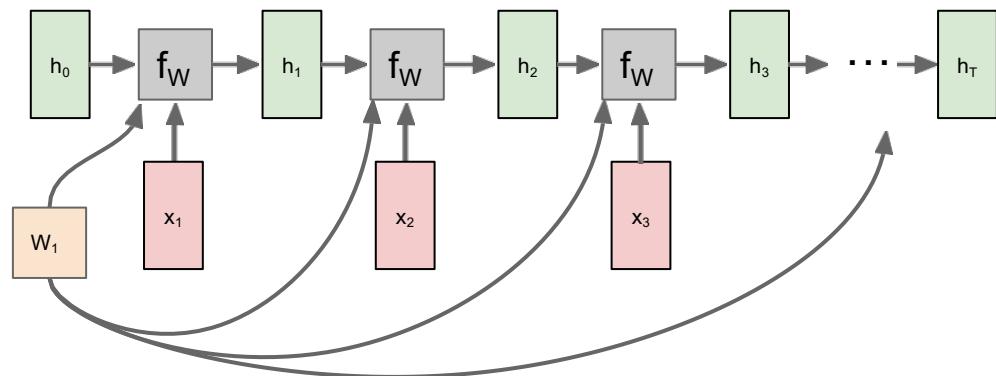


# RNN: Computational Graph: One to Many



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

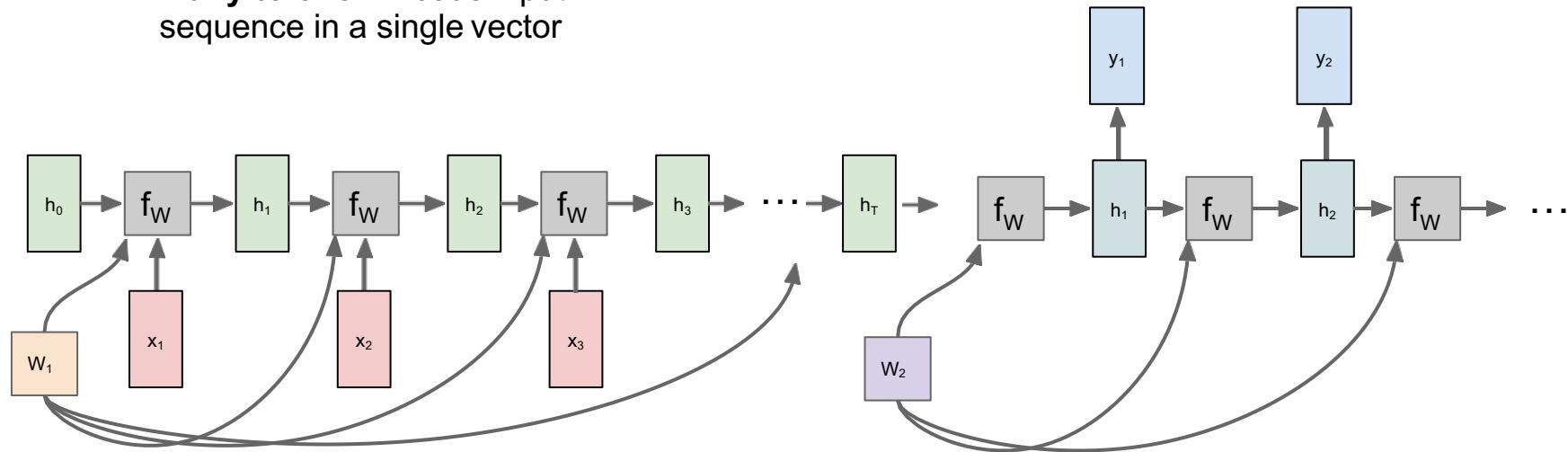
**Many to one:** Encode input sequence in a single vector



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

**Many to one:** Encode input sequence in a single vector

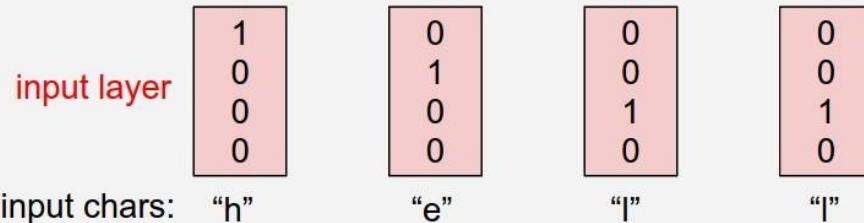
**One to many:** Produce output sequence from single input vector



# Example: Character-level Language Model

Vocabulary:  
[h,e,l,o]

Example training  
sequence:  
“hello”

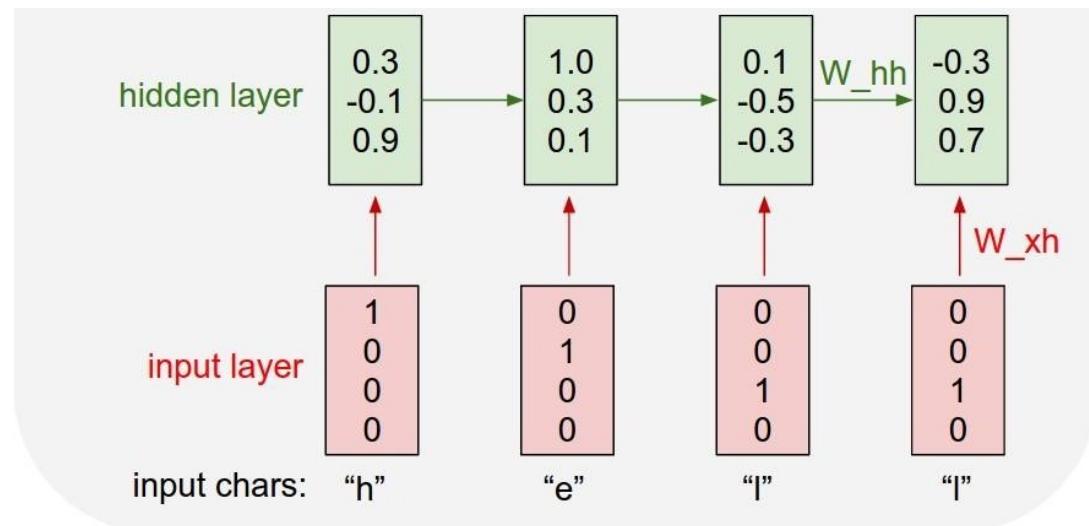


# Example: Character-level Language Model

Vocabulary:  
[h,e,l,o]

Example training  
sequence:  
“hello”

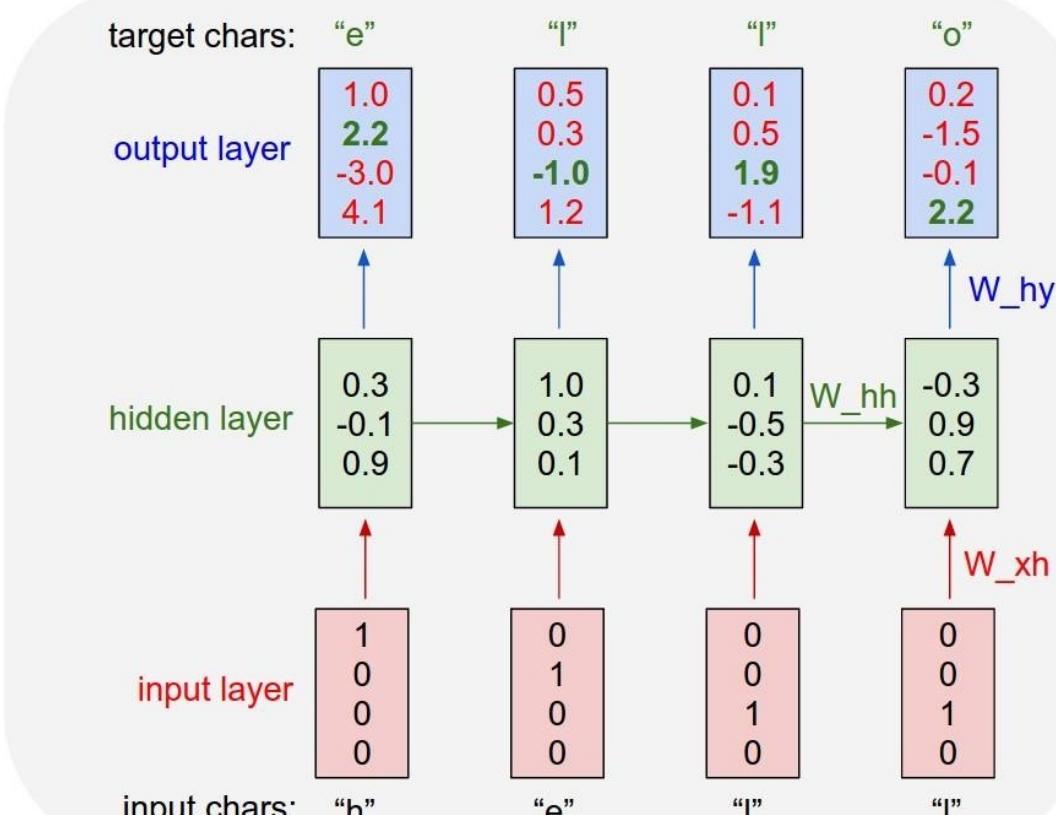
$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$



# Example: Character-level Language Model

Vocabulary:  
[h,e,l,o]

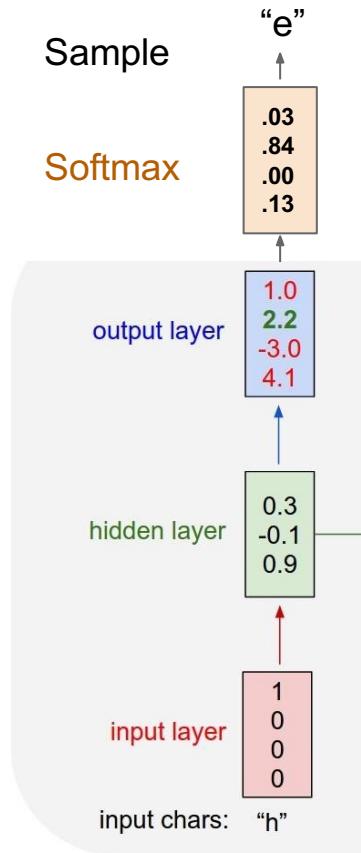
Example training  
sequence:  
“hello”



# Example: Character-level Language Model Sampling

Vocabulary:  
[h,e,l,o]

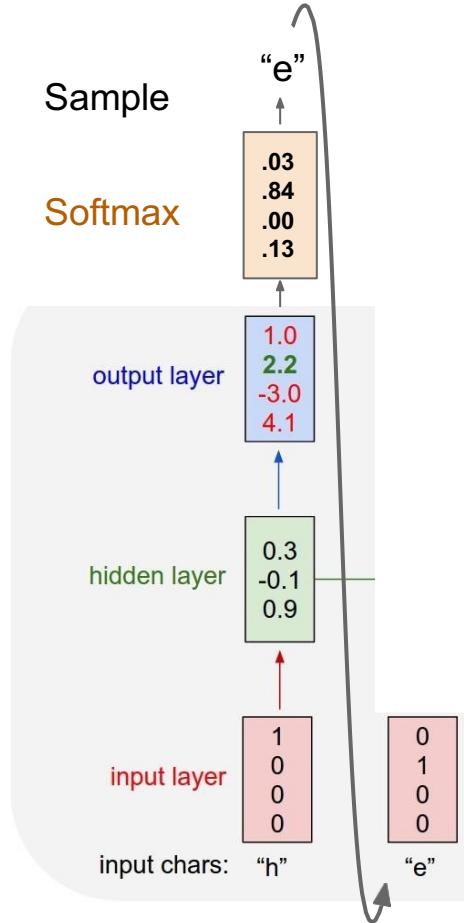
At test-time sample  
characters one at a time,  
feed back to model



# Example: Character-level Language Model Sampling

Vocabulary:  
[h,e,l,o]

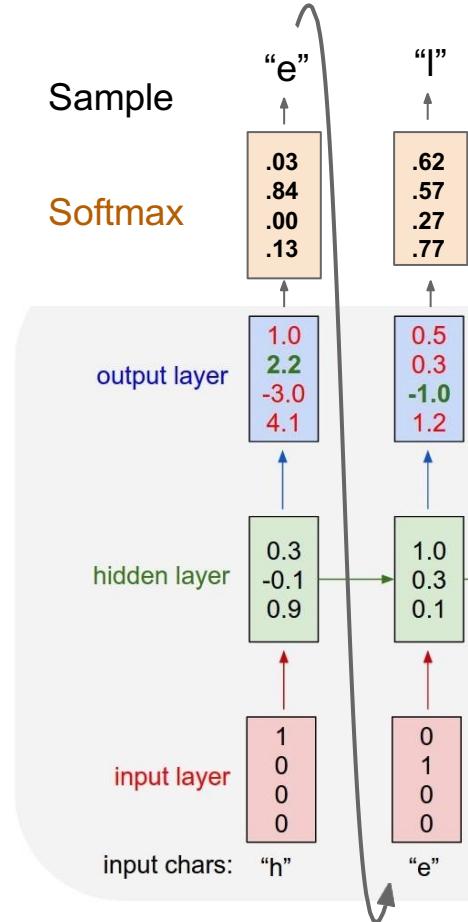
At test-time sample  
characters one at a time,  
feed back to model



# Example: Character-level Language Model Sampling

Vocabulary:  
[h,e,l,o]

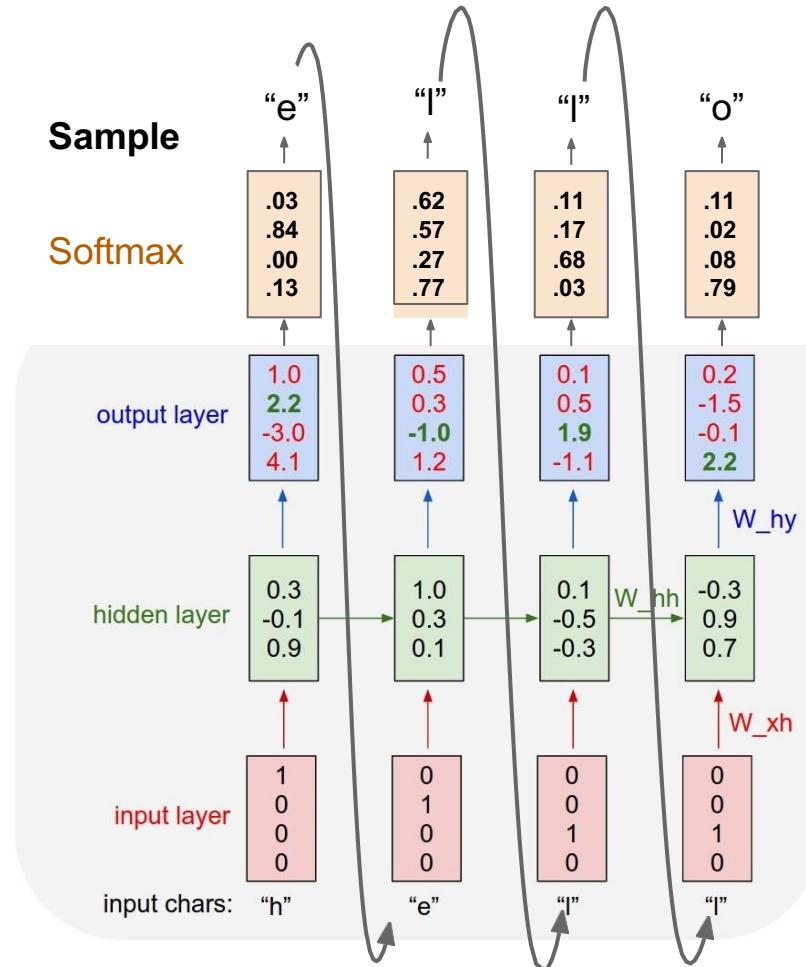
At test-time sample  
characters one at a time,  
feed back to model



# Example: Character-level Language Model Sampling

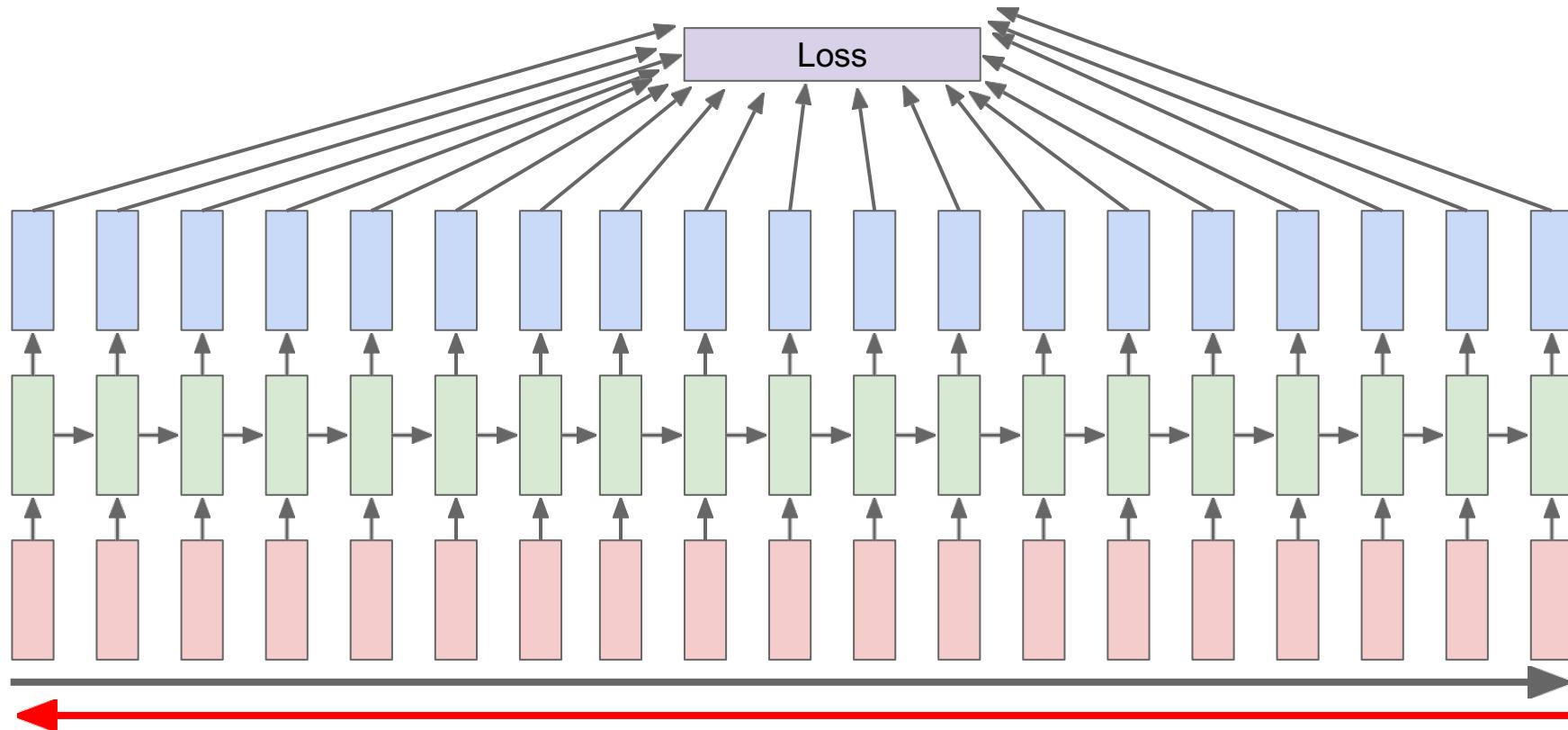
Vocabulary:  
[h,e,l,o]

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

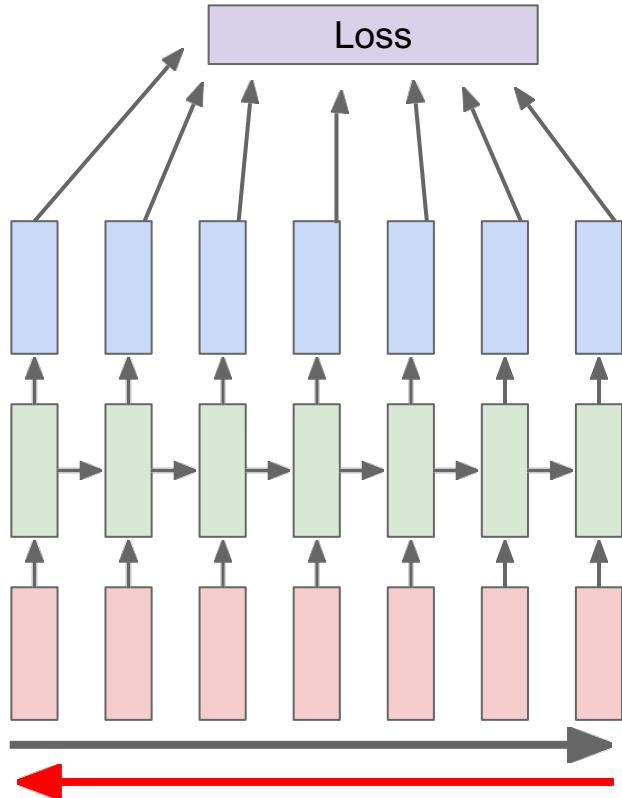


# Backpropagation through time

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

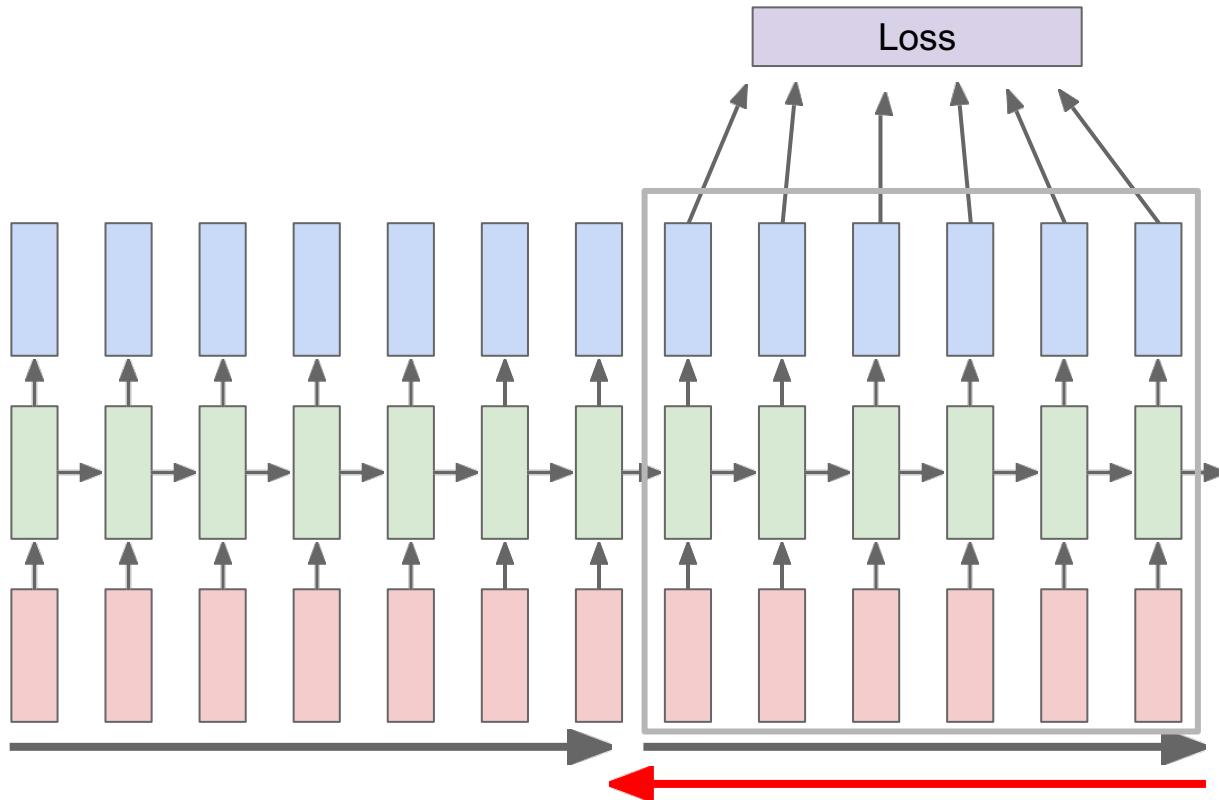


# Truncated Backpropagation through time



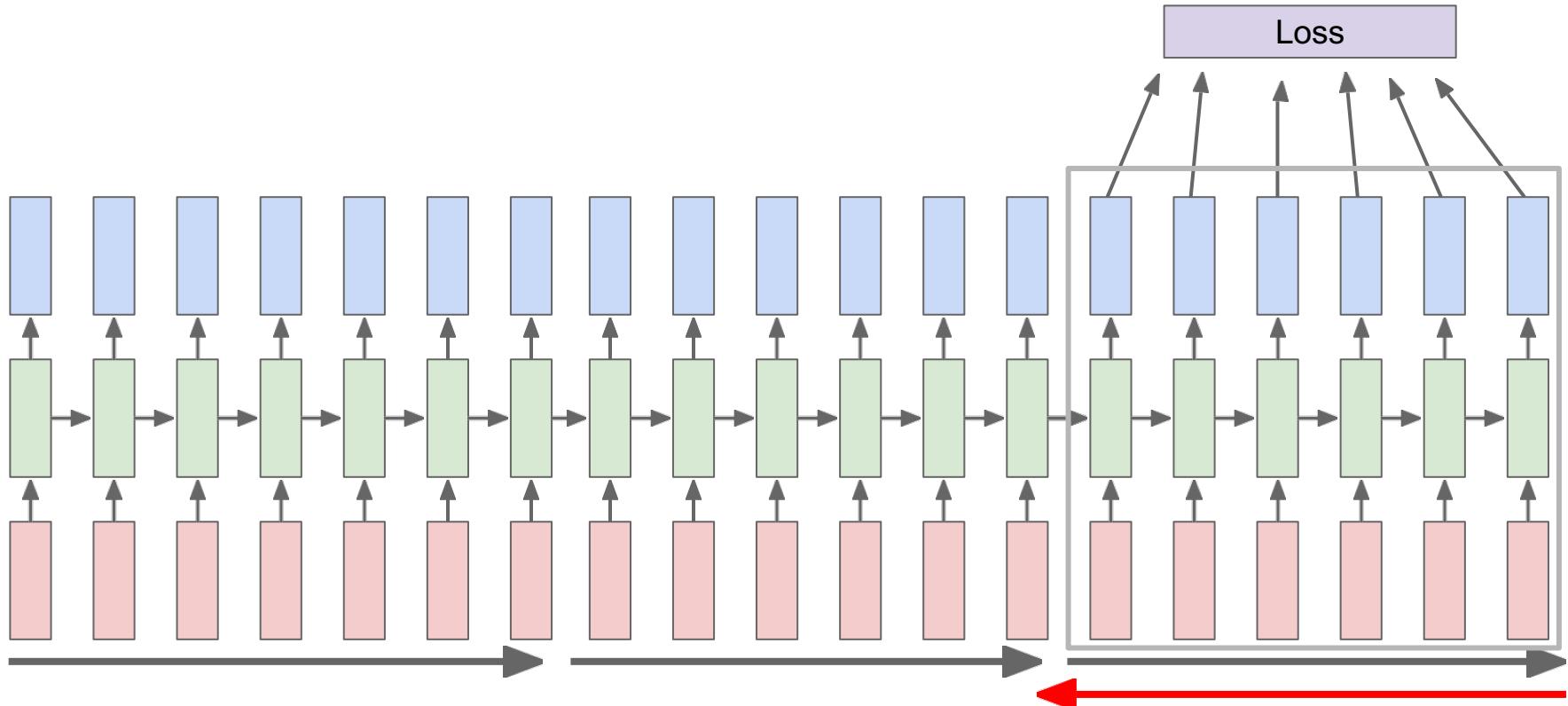
Run forward and backward  
through chunks of the  
sequence instead of whole  
sequence

# 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 wkh = np.random.rand(hidden_size, vocab_size)*0.01 # input to hidden
22 whh = np.random.rand(hidden_size, hidden_size)*0.01 # hidden to hidden
23 why = np.random.rand(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(dwhh, 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
45         # backward pass: compute gradients going backwards
46         dwhh, dwhy, dbh = np.zeros_like(wkh), np.zeros_like(why), np.zeros_like(by)
47         dwhh, dwhy, dbh = np.zeros_like(bh), np.zeros_like(by)
48         dhnext = np.zeros_like(hs[0])
49         for t in reversed(xrange(len(inputs))):
50             dy = np.copy(ps[t])
51             dy[targets[t]] -= 1 # backprop into y
52             dyh = np.dot(dy, hs[t].T)
53             dh = np.dot(why.T, dy) + dhnext # backprop into h
54             ddraw = (i - hs[t]) * hs[t] * dh # backprop through tanh nonlinearity
55             dwhh += np.dot(ddraw, xs[t].T)
56             dwhy += np.dot(ddraw, hs[t-1].T)
57             dhnext = np.dot(why.T, ddraw)
58             for dparam in [dwhh, dwhy, dbh, dby]:
59                 np.clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients
60
61     return loss, dwhh, dwhy, dbh, dby, hs[len(inputs)-1]
```

```
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 = np.zeros((vocab_size, 1))
77         x[ix] = 1
78         ixes.append(ix)
79
80     return ixes
81
82 n, p = 0, 0
83 mxwh, mwhh, mwhy = np.zeros_like(wkh), np.zeros_like(whh), np.zeros_like(why)
84 mbh, mbw = np.zeros_like(bh), np.zeros_like(by) # memory variables for Adagrad
85 smooth_loss = -np.log(1.0/vocab_size)*seq_length # loss at iteration 0
86 while True:
87     # prepare inputs (we're sweeping from left to right in steps seq_length long)
88     if p+seq_length+1 >= len(data) or n == 0:
89         hprev = np.zeros((hidden_size,1)) # reset RNN memory
90         p = 0 # go from start of data
91     inputs = [char_to_ix[ch] for ch in data[p:p+seq_length]]
92     targets = [char_to_ix[ch] for ch in data[p+1:p+seq_length+1]]
93
94     # sample from the model now and then
95     if n % 100 == 0:
96         sample_ix = sample(hprev, inputs[0], 200)
97         txt = ''.join(ix_to_char[ix] for ix in sample_ix)
98         print '----\n' + ' '.join(['%s' % (txt, )]
99
100     # forward seq_length characters through the net and fetch gradient
101     loss, dwhh, dwhy, dbh, dby, hprev = lossFun(inputs, targets, hprev)
102     smooth_loss = smooth_loss * .999 + loss * .001
103     if n % 100 == 0: print 'iter %d, loss: %f' % (n, smooth_loss) # print progress
104
105     # perform parameter update with Adagrad
106     for param, dparam, mem in zip([wkh, whh, why, bh, by],
107                                   [dwhh, dwhy, dbh, dby, dbw]):
108         mem += dparam * dparam
109         param += -learning_rate * param / np.sqrt(mem + 1e-8) # adagrad update
110
111     p += seq_length # move data pointer
112     n += 1 # iteration counter
```

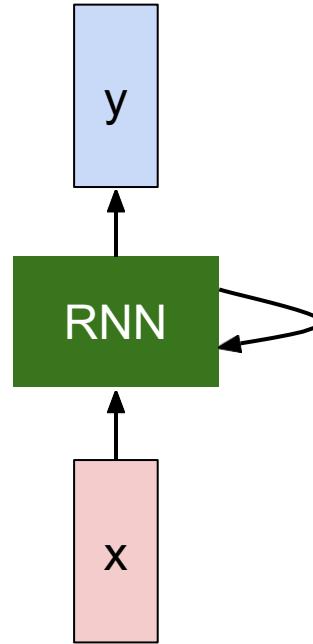
(<https://gist.github.com/karpathy/d4dee566867f8291f086>)

# THE SONNETS

by William Shakespeare

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

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



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 shuwyl 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.

# 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	<a href="#">online</a>	<a href="#">tex</a>	<a href="#">pdf</a>
	2. Conventions	<a href="#">online</a>	<a href="#">tex</a>	<a href="#">pdf</a>
	3. Set Theory	<a href="#">online</a>	<a href="#">tex</a>	<a href="#">pdf</a>
	4. Categories	<a href="#">online</a>	<a href="#">tex</a>	<a href="#">pdf</a>
	5. Topology	<a href="#">online</a>	<a href="#">tex</a>	<a href="#">pdf</a>
	6. Sheaves on Spaces	<a href="#">online</a>	<a href="#">tex</a>	<a href="#">pdf</a>
	7. Sites and Sheaves	<a href="#">online</a>	<a href="#">tex</a>	<a href="#">pdf</a>
	8. Stacks	<a href="#">online</a>	<a href="#">tex</a>	<a href="#">pdf</a>
	9. Fields	<a href="#">online</a>	<a href="#">tex</a>	<a href="#">pdf</a>
	10. Commutative Algebra	<a href="#">online</a>	<a href="#">tex</a>	<a href="#">pdf</a>

**Parts**

1. [Preliminaries](#)
2. [Schemes](#)
3. [Topics in Scheme Theory](#)
4. [Algebraic Spaces](#)
5. [Topics in Geometry](#)
6. [Deformation Theory](#)
7. [Algebraic Stacks](#)
8. [Miscellany](#)

**Statistics**

The Stacks project now consists of

- o 455910 lines of code
- o 14221 tags (56 inactive tags)
- o 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  $\text{Sch}_{fppf}$  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

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

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  $S_{\text{Zar}}$ , 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  $\mathcal{F}_{x_0} = \mathcal{F}_{x_0} = \mathcal{F}_{x,\dots,x_0}$ .

**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  $\mathcal{J}_{n,0} \circ \mathcal{A}_2$  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  $\delta_{n+1}$  is a scheme over  $S$ .  $\square$

*Proof.* Omitted.  $\square$

**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.  $\square$

**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.  $\square$

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

$$\begin{array}{ccccc}
 S & \xrightarrow{\quad} & & & \\
 \downarrow & & & & \\
 \xi & \xrightarrow{\quad} & \mathcal{O}_{X'} & \xleftarrow{\quad} & \\
 & & \uparrow \text{gor}_s & & \\
 & & = \alpha' & \longrightarrow & \\
 & & \downarrow & & \\
 & & = \alpha' & \longrightarrow & \alpha \\
 & & \uparrow & & \\
 \text{Spec}(K_\psi) & & \text{Mor}_{\text{Sets}} & & d(\mathcal{O}_{X_{/\mathbb{A}^1}}, \mathcal{G}) \\
 & & \downarrow X & & \\
 & & & & \text{d}(\mathcal{O}_{X_{/\mathbb{A}^1}}, \mathcal{G})
 \end{array}$$

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  $U$ .  $\square$

*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”

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

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.  $\square$

If  $\mathcal{F}$  is a finite direct sum  $\mathcal{O}_X$  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](#) [+.](#)   [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](#)[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](#)

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

latest commit [4b1706927d](#) 

torvalds authored 9 hours 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/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](#) libio: introduce New\_Req and multi-blk\_t to avoid memory allocation in ... 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) & 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

# 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

# Machine Translation is Everywhere



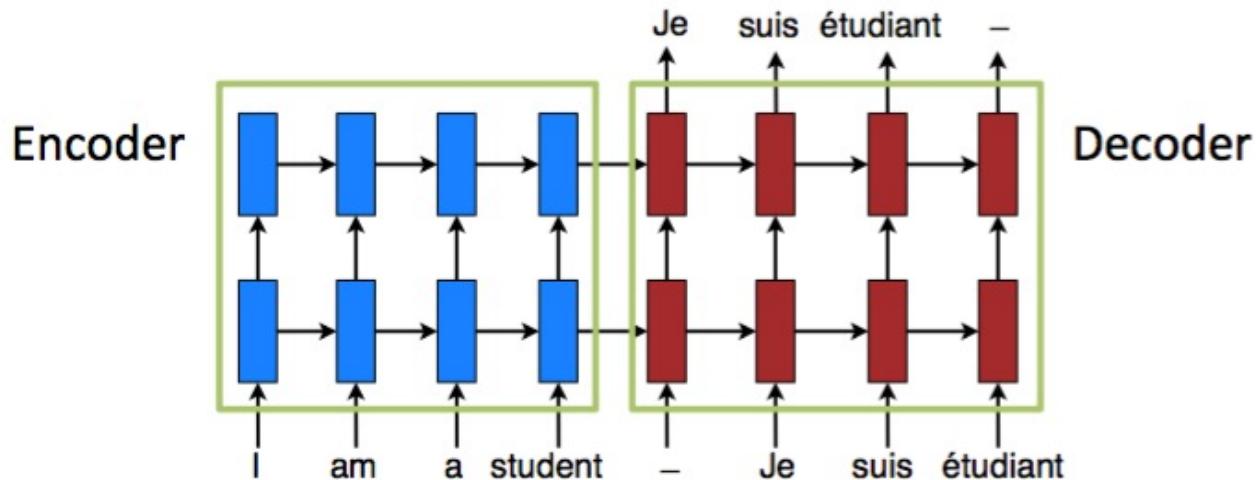
Sisi roasted husband



Meat Muscle Stupid  
Bean Sprouts

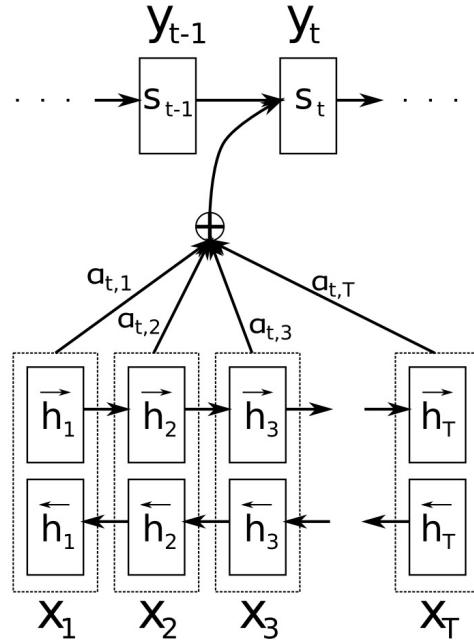
# Neural Machine Translation

- Sequence2Sequence/Encoder-Decoder model
- Decode the entire translation from one “compressed” vector



# The Attention Mechanism

- The solution: enable the network to pay attention to specific areas of the input by adding new (weighted) connections



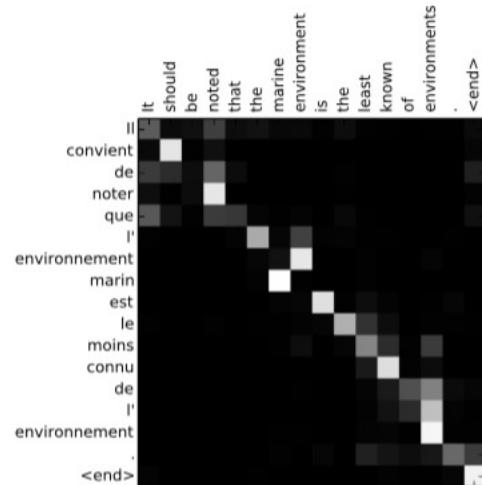
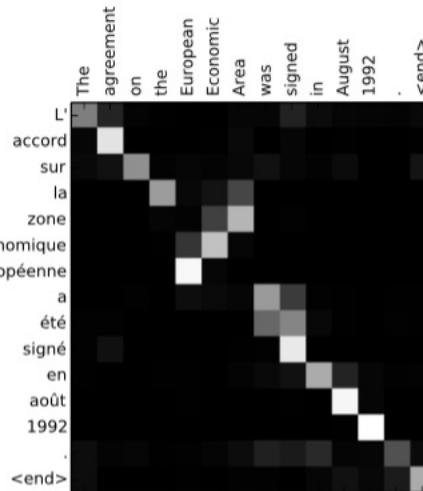
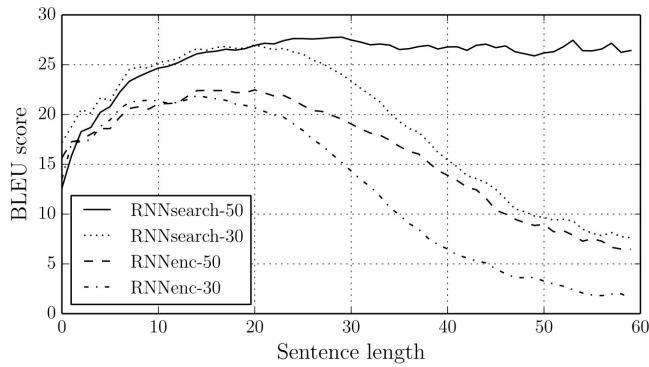
$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j$$

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})},$$

$$e_{ij} = a(s_{i-1}, h_j)$$

# Benefit of The Attention Mechanism

- The attention mechanism helps to learn longer sentences
- The model is able to learn nice alignments:



# Image Captioning

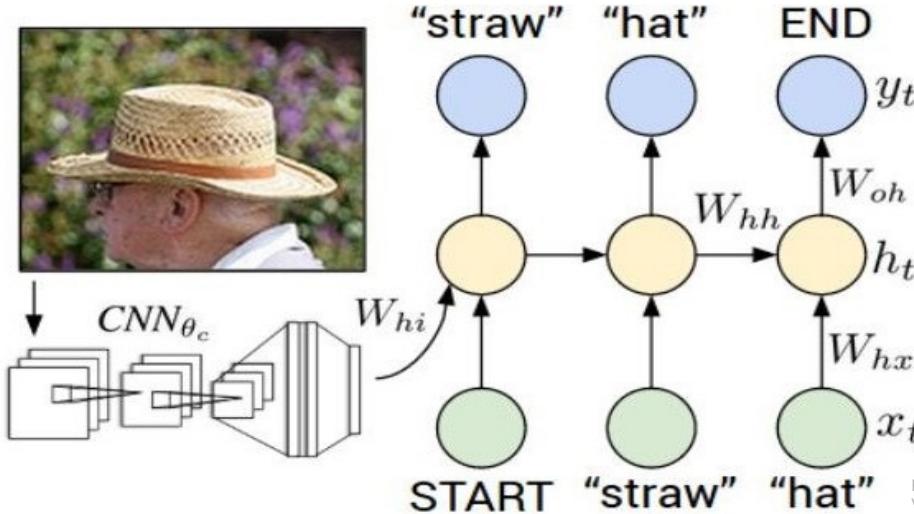


Figure from Karpathy et al, "Deep Visual-Semantic Alignments for Generating Image Descriptions", CVPR 2015; figure copyright IEEE, 2015.  
Reproduced for educational purposes.

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

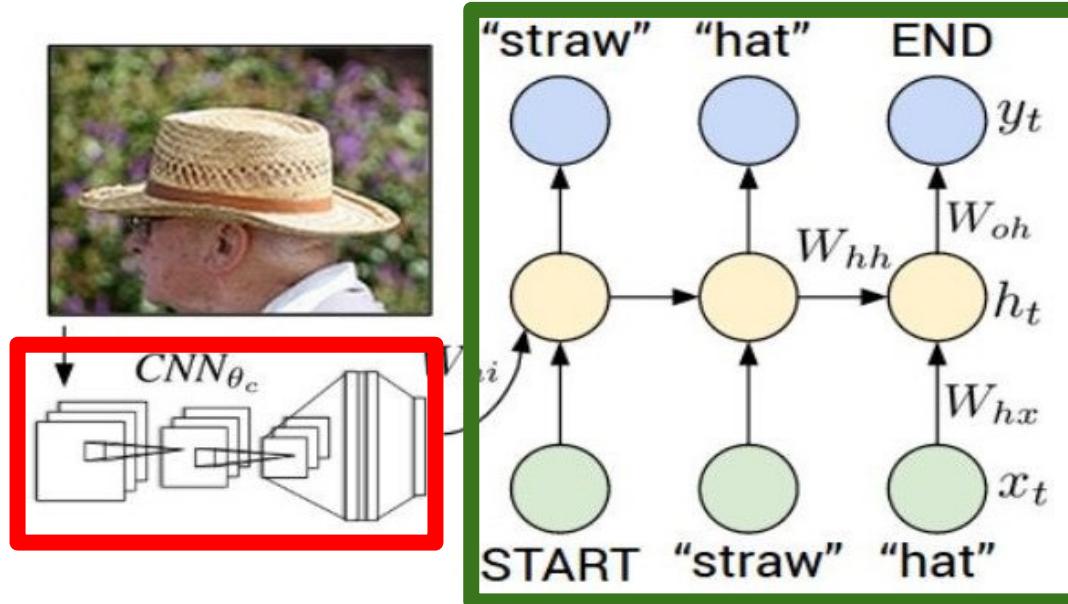
Deep Visual-Semantic Alignments for Generating Image Descriptions, Karpathy and Fei-Fei

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



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

FC-1000

softmax

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

FC-1000

softmax



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

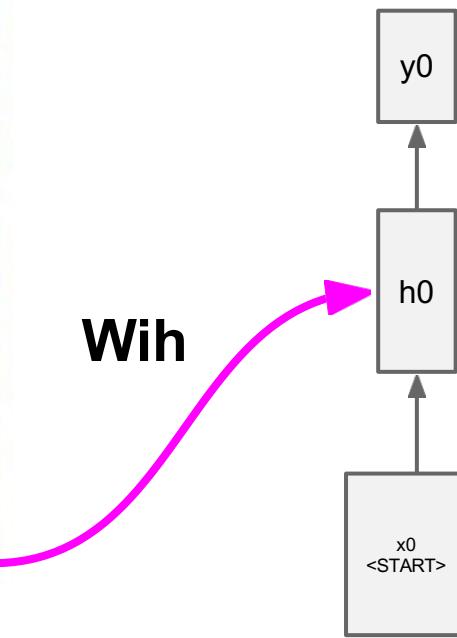
maxpool

FC-4096

FC-4096

V





**before:**

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

**now:**

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



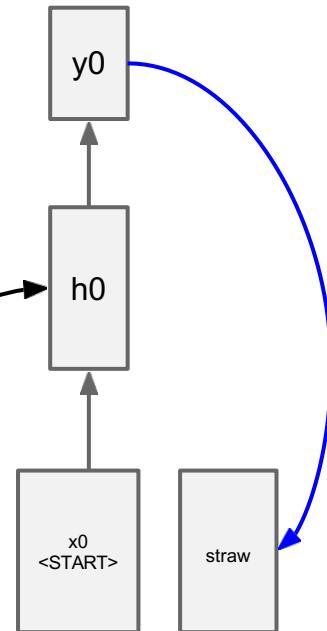
test image

v



test image

sample!



image



test image

conv-64

conv-64

maxpool

conv-128

conv-128

maxpool

conv-256

conv-256

maxpool

conv-512

conv-512

maxpool

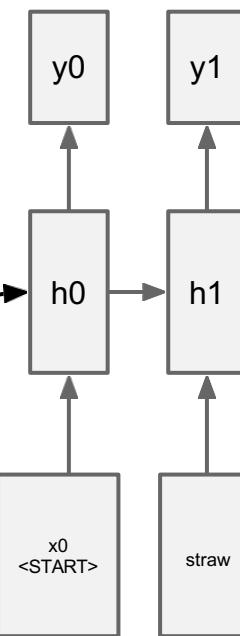
conv-512

conv-512

maxpool

FC-4096

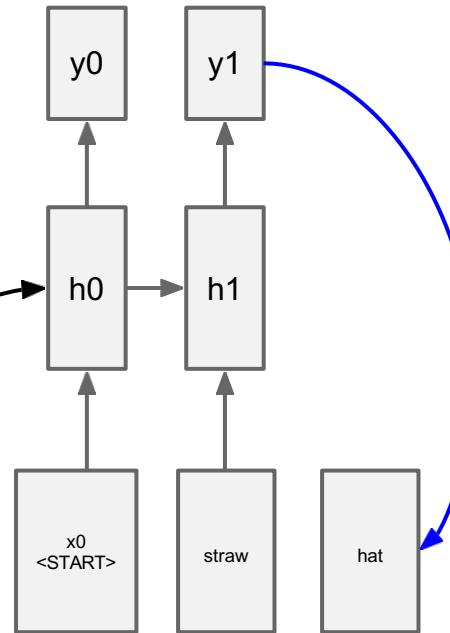
FC-4096



V



test image



V

image

conv-64

conv-64

maxpool

conv-128

conv-128

maxpool

conv-256

conv-256

maxpool

conv-512

conv-512

maxpool

conv-512

conv-512

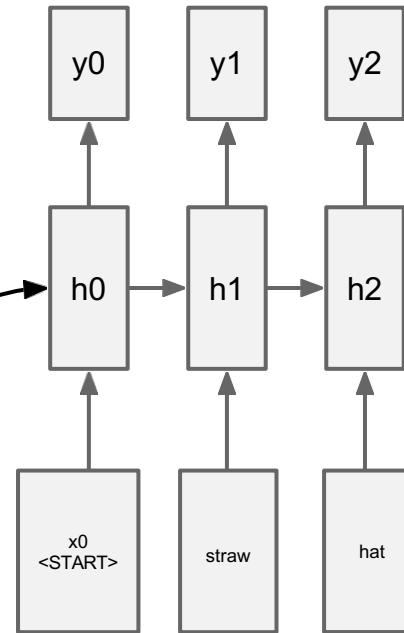
maxpool

FC-4096

FC-4096



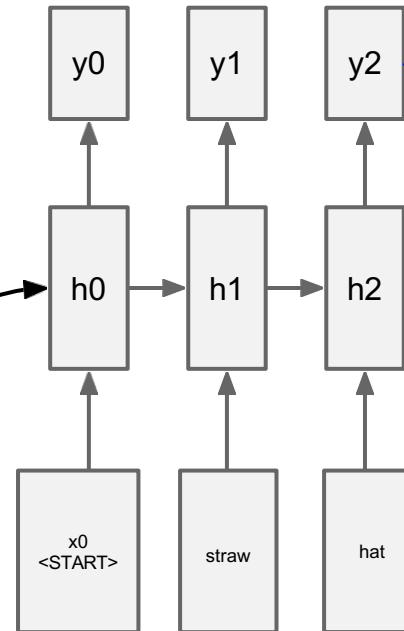
test image



V



test image



sample  
<END> token  
=> finish.

V

# 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*



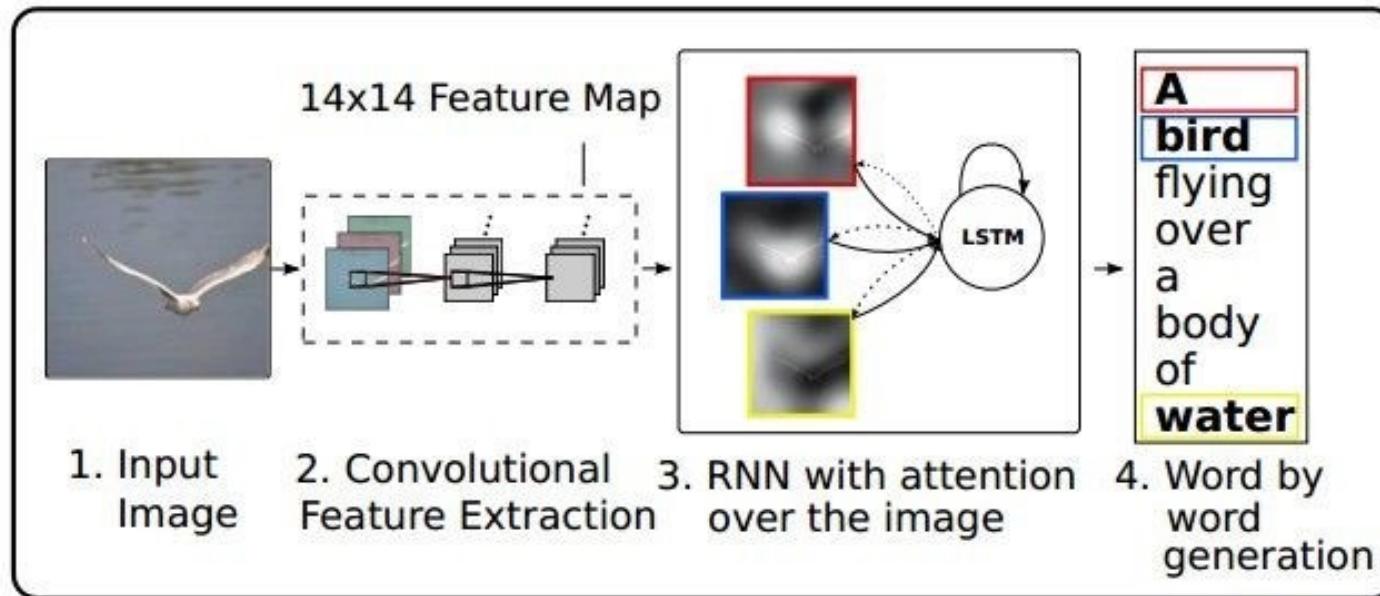
*A bird is perched on a tree branch*



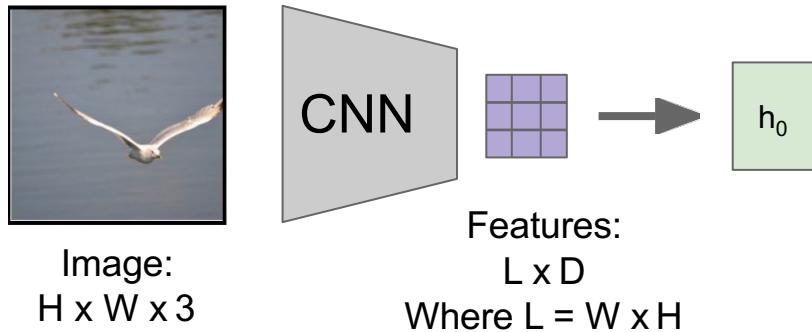
*A man in a baseball uniform throwing a ball*

# Image Captioning with Attention

RNN focuses its attention at a different spatial location when generating each word

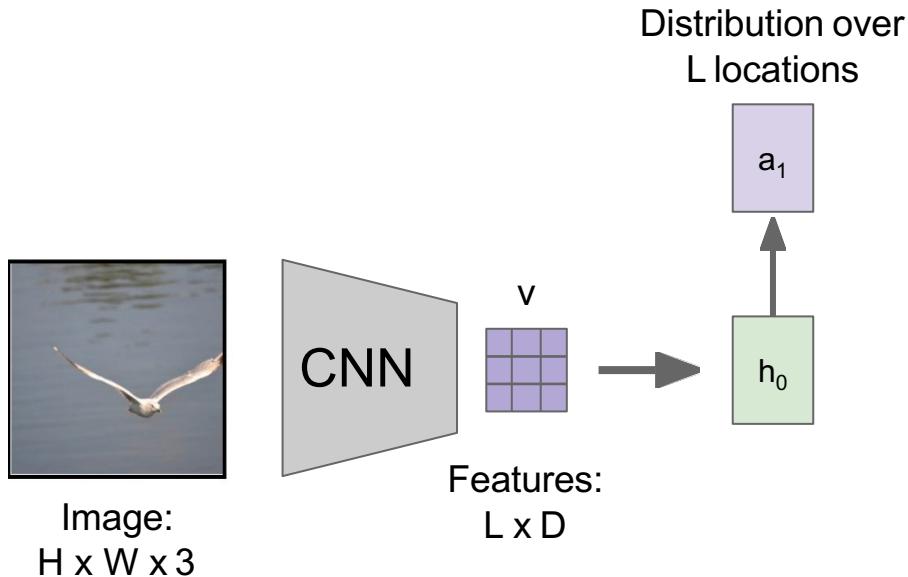


# Image Captioning with Attention



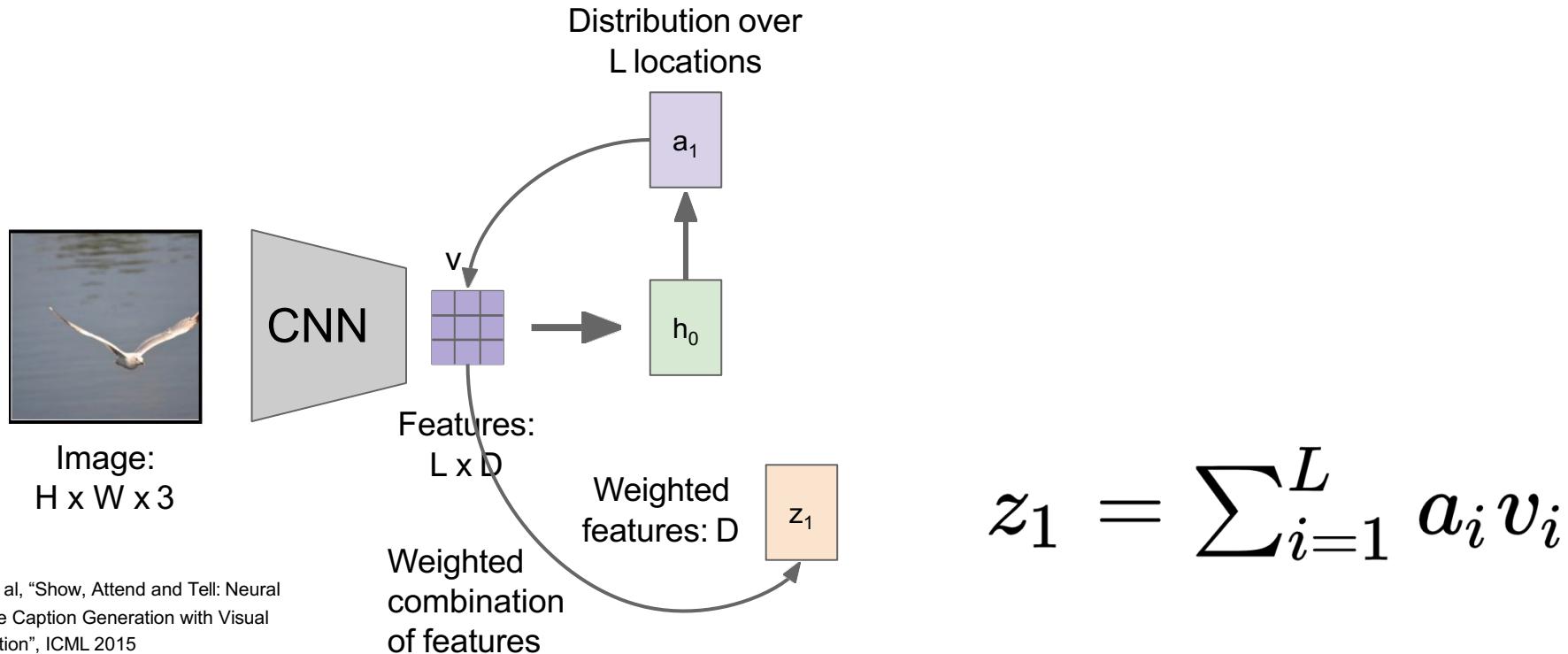
Xu et al, "Show, Attend and Tell: Neural  
Image Caption Generation with Visual  
Attention", ICML 2015

# Image Captioning with Attention



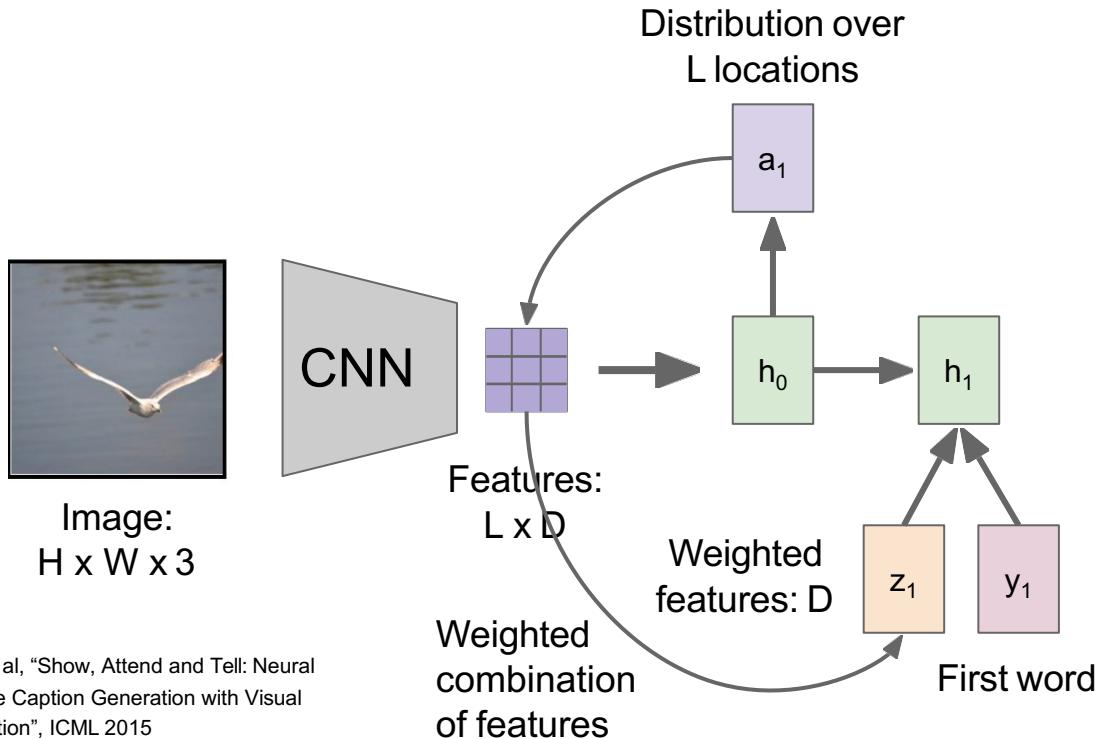
Xu et al, "Show, Attend and Tell: Neural  
Image Caption Generation with Visual  
Attention", ICML 2015

# Image Captioning with Attention



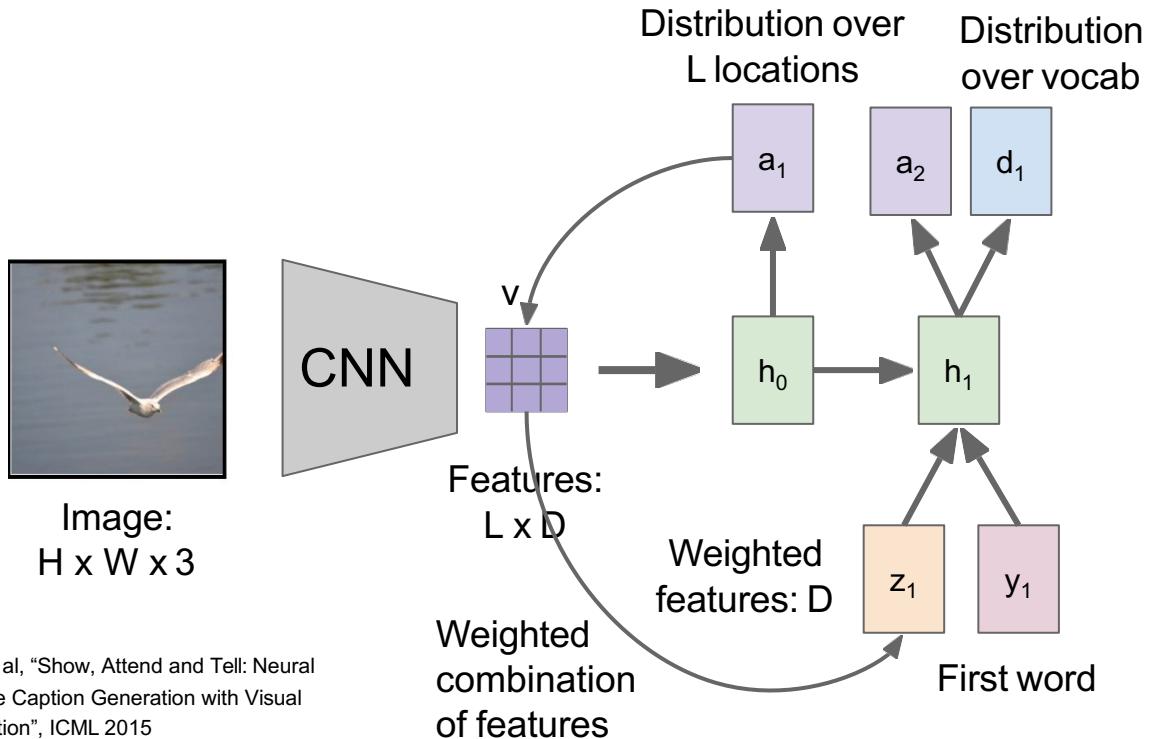
Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

# Image Captioning with Attention



Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

# Image Captioning with Attention

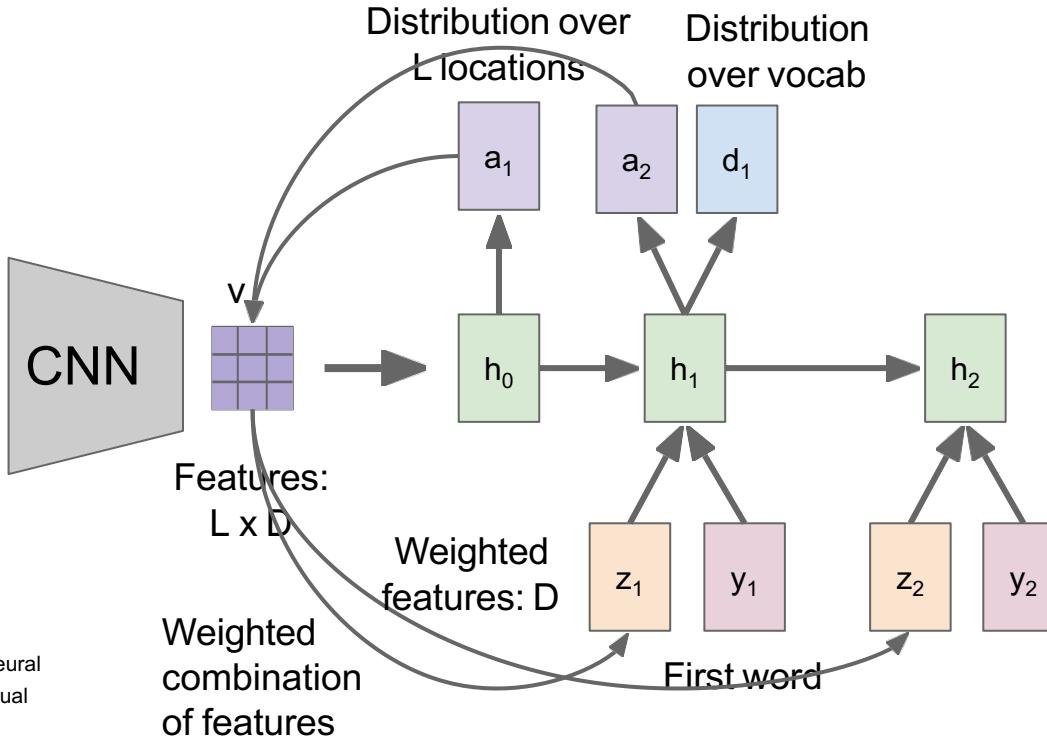


Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

# Image Captioning with Attention

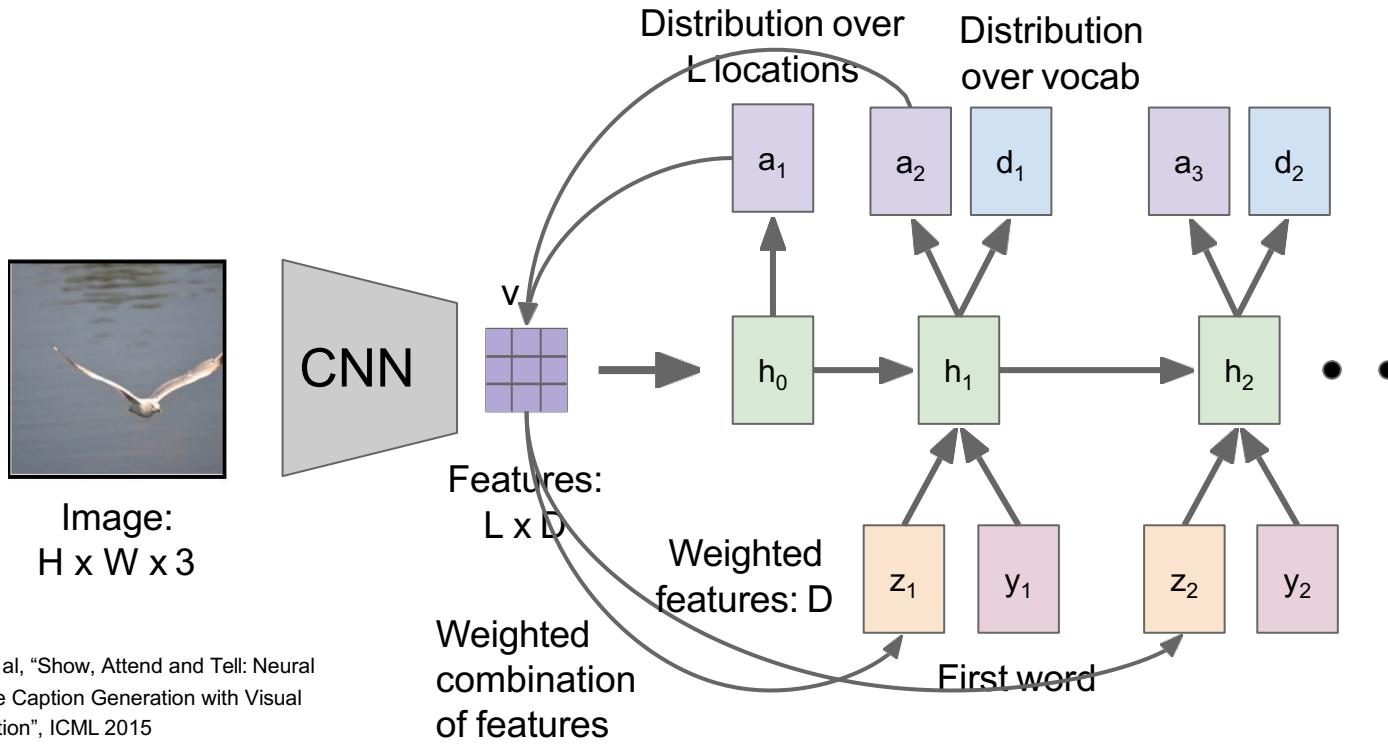


Image:  
 $H \times W \times 3$



Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

# Image Captioning with Attention



Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

# Image Captioning with Attention

Soft attention



Hard attention



A

bird

flying

over

a

body

of

water

.

# Image Captioning with Attention



A woman is throwing a frisbee in a park.

A dog is standing on a hardwood floor.

A stop sign is on a road with a mountain in the background.



A little girl sitting on a bed with a teddy bear.



A group of people sitting on a boat in the water.



A giraffe standing in a forest with trees in the background.

# 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 1/4 Rd.
- A: Onto 25 1/4 Rd.
- A: Onto 23 1/4 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?

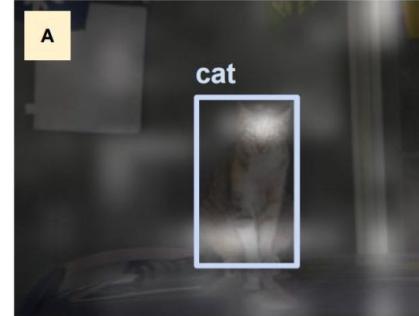
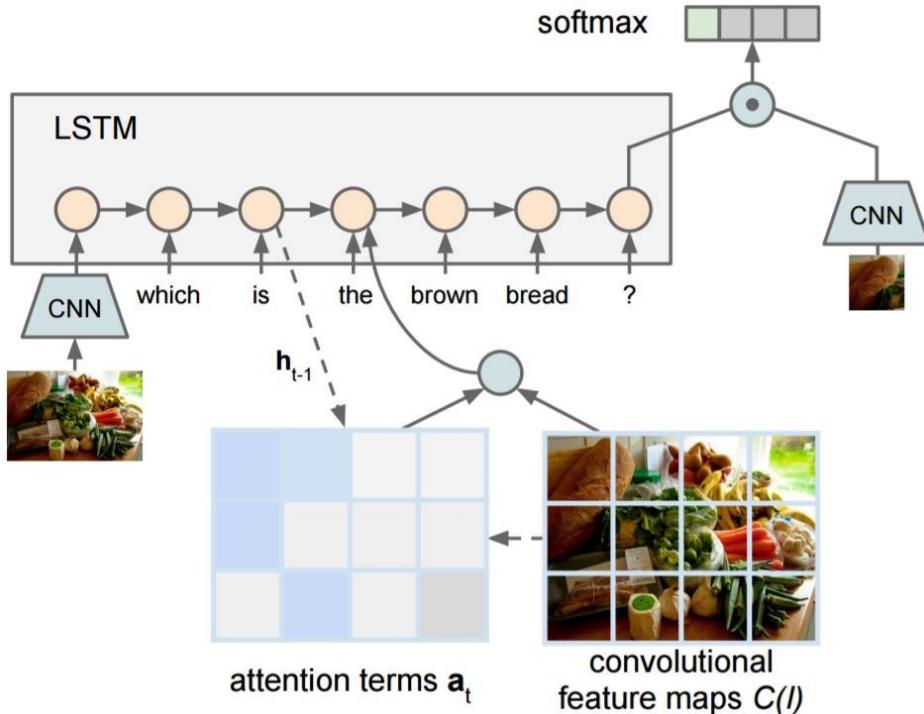
- A: Two women.
- A: A child.
- A: An old man.
- A: A husband and a wife.

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



What kind of animal is in the photo?  
A **cat**.



Why is the person holding a knife?  
To cut the **cake** with.

# Visual Dialog: Conversations about images

**Visual Dialog**

The screenshot shows a visual dialog interface. At the top, there's a blue header bar with the text "Visual Dialog". Below it is a photograph of a brown tabby cat standing on a wooden table, leaning over a white coffee mug to drink water from it. On the table next to the mug are several items: a blue and white plate, a red book or magazine, and some papers. In the background, there's a tiled wall and a black object that looks like a toaster.

A dark blue speech bubble contains the caption: "A cat drinking water out of a coffee mug."

The conversation consists of the following turns:

- User (blue bubble): "What color is the mug?"
- Robot (red robot icon): "White and red"
- User (blue bubble): "Are there any pictures on it?"
- Robot (red robot icon): "No, something is there can't tell what it is"
- User (blue bubble): "Is the mug and cat on a table?"
- Robot (red robot icon): "Yes, they are"
- User (blue bubble): "Are there other items on the table?"
- Robot (red robot icon): "Yes, magazines, books, toaster and basket, and a plate"

At the bottom of the interface, there are three icons: a camera icon, a text input field with the placeholder "Start typing question here ...", and a right-pointing arrow icon.

# 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

# Multilayer RNNs

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

$$h \in \mathbb{R}^n, \quad W^l [n \times 2n]$$

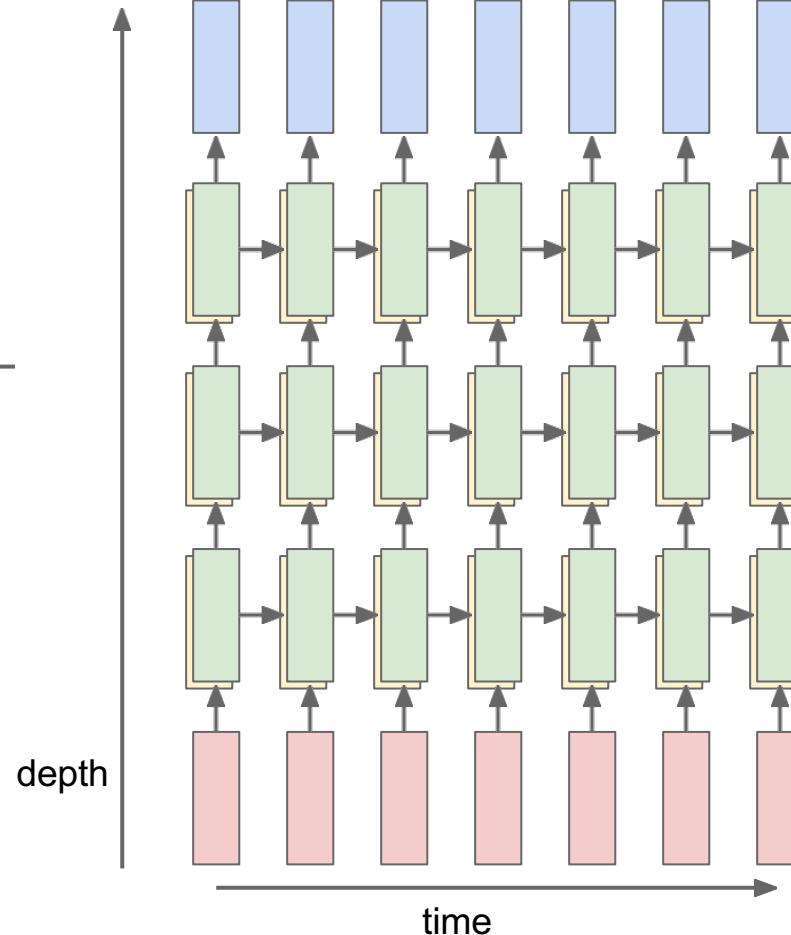
LSTM:

$$W^l [4n \times 2n]$$

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \tanh \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$

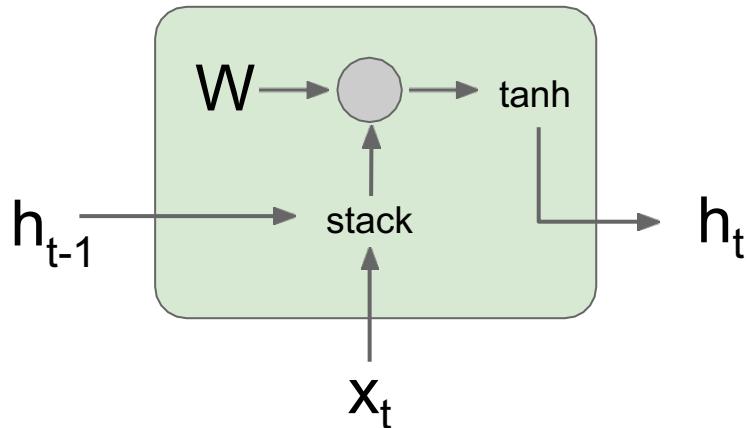
$$c_t^l = f \odot c_{t-1}^l + i \odot g$$

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



# Vanilla RNN Gradient Flow

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

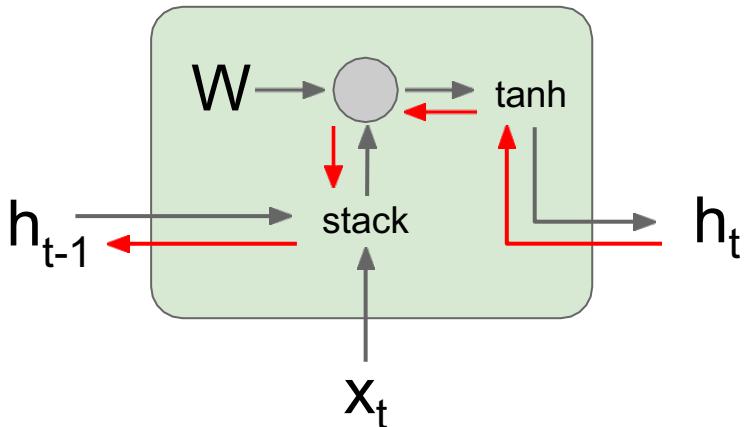


$$\begin{aligned} h_t &= \tanh(W_{hh}h_{t-1} + W_{xh}x_t) \\ &= \tanh \left( \begin{pmatrix} W_{hh} & W_{hx} \end{pmatrix} \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \right) \\ &= \tanh \left( W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \right) \end{aligned}$$

# Vanilla RNN Gradient Flow

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

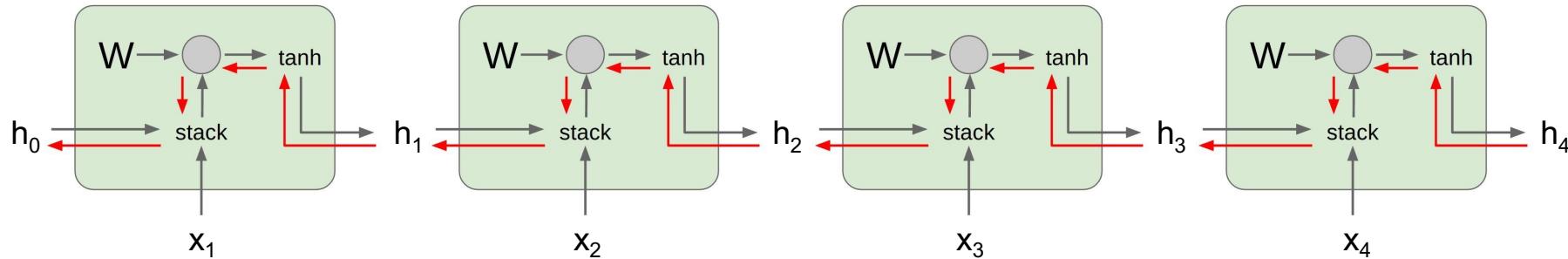
Backpropagation from  $h_t$   
to  $h_{t-1}$  multiplies by  $W$   
(actually  $W_{hh}^T$ )



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

# Vanilla RNN Gradient Flow

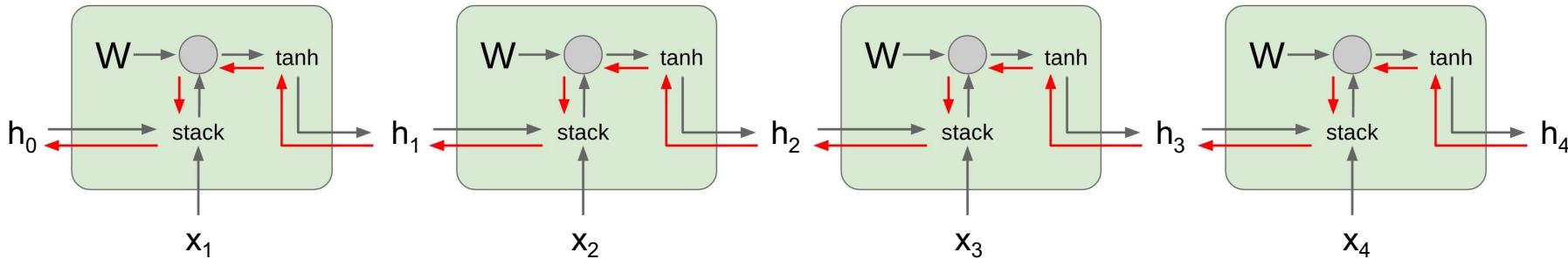
Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994  
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013



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

# 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

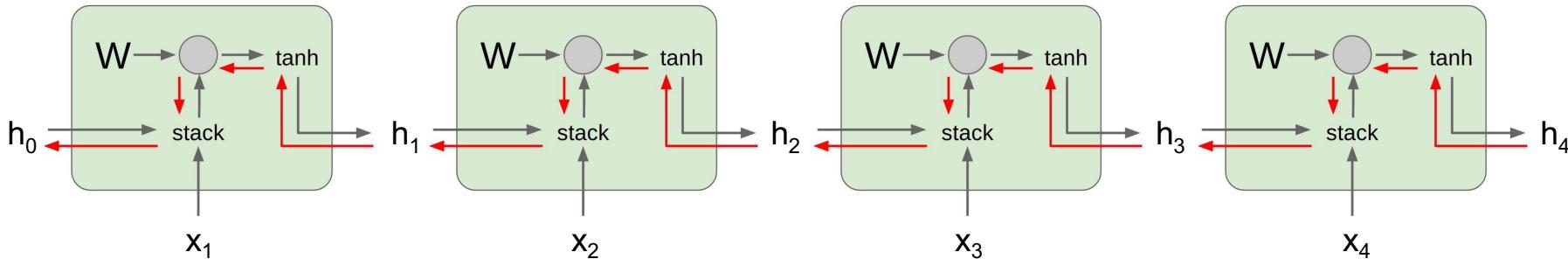


Computing gradient of  $h_0$  involves many factors of  $W$   
(and repeated tanh)

Largest singular value  $> 1$ :  
**Exploding gradients**

# Vanilla RNN Gradient Flow

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



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

Largest singular value  $> 1$ :  
**Exploding gradients**

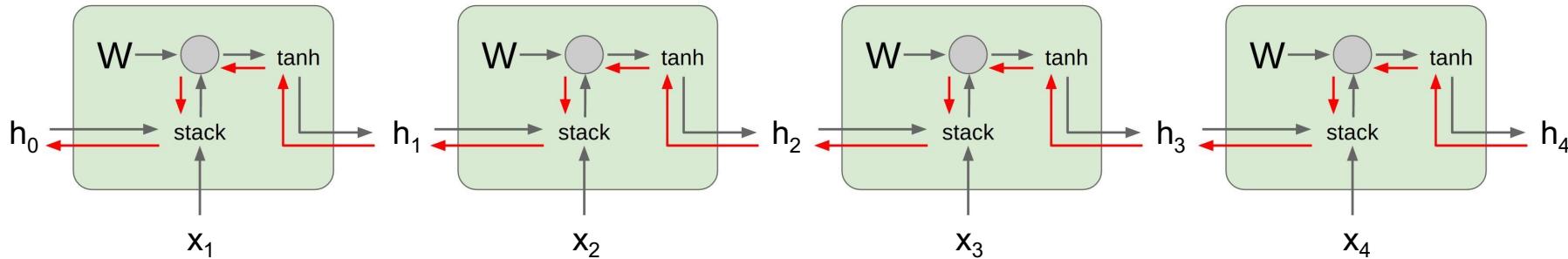
Largest singular value  $< 1$ :  
**Vanishing gradients**

**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

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



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

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

## LSTM

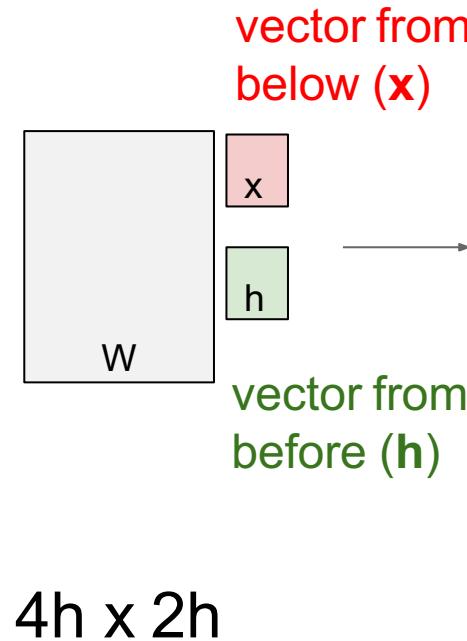
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

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

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

# Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]



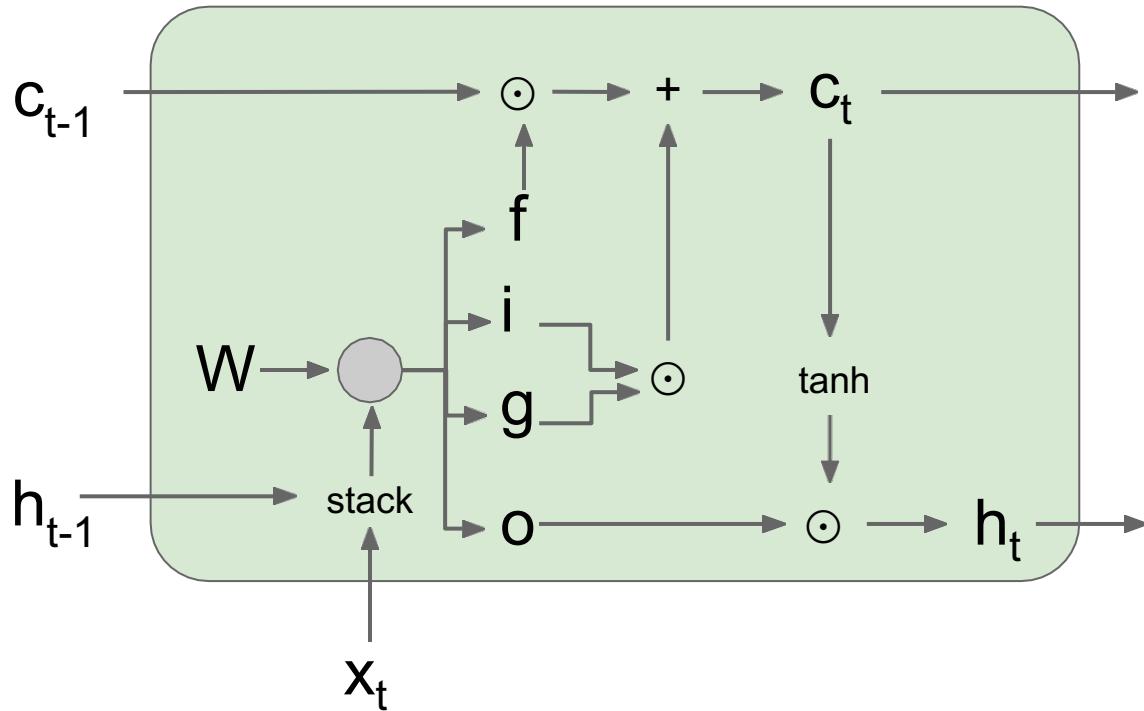
- i: Input gate, whether to write to cell
- f: Forget gate, Whether to erase cell
- o: Output gate, How much to reveal cell
- g: Gate gate (?), How much to write to cell

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

$$c_t = f \odot c_{t-1} + i \odot g$$
$$h_t = o \odot \tanh(c_t)$$

# Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]



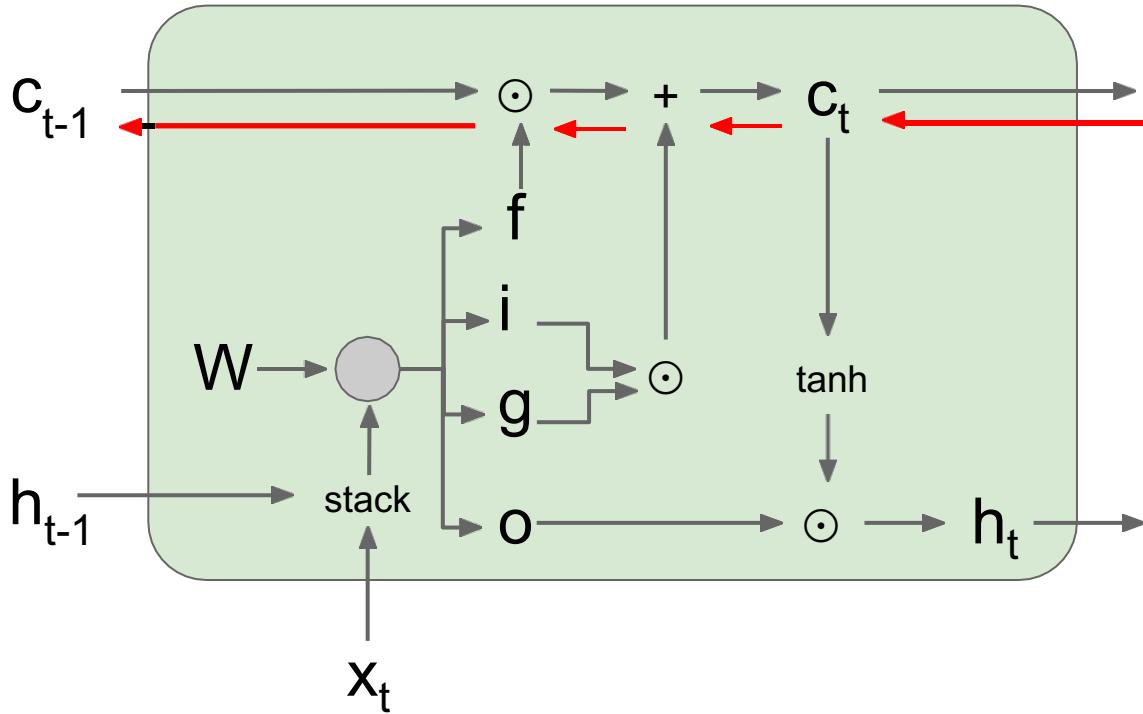
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

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

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

# Long Short Term Memory (LSTM): Gradient Flow

[Hochreiter et al., 1997]



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

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

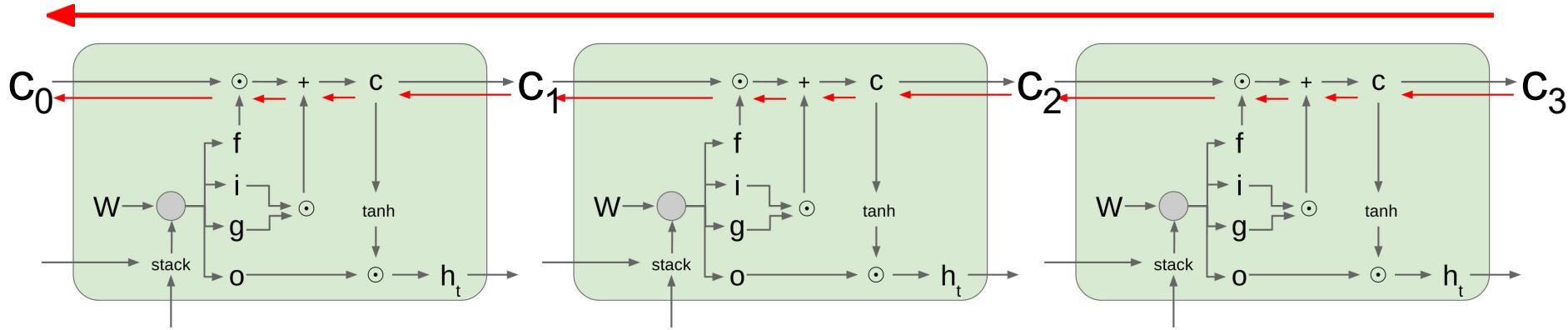
$$c_t = f \odot c_{t-1} + i \odot g$$

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

# Long Short Term Memory (LSTM): Gradient Flow

[Hochreiter et al., 1997]

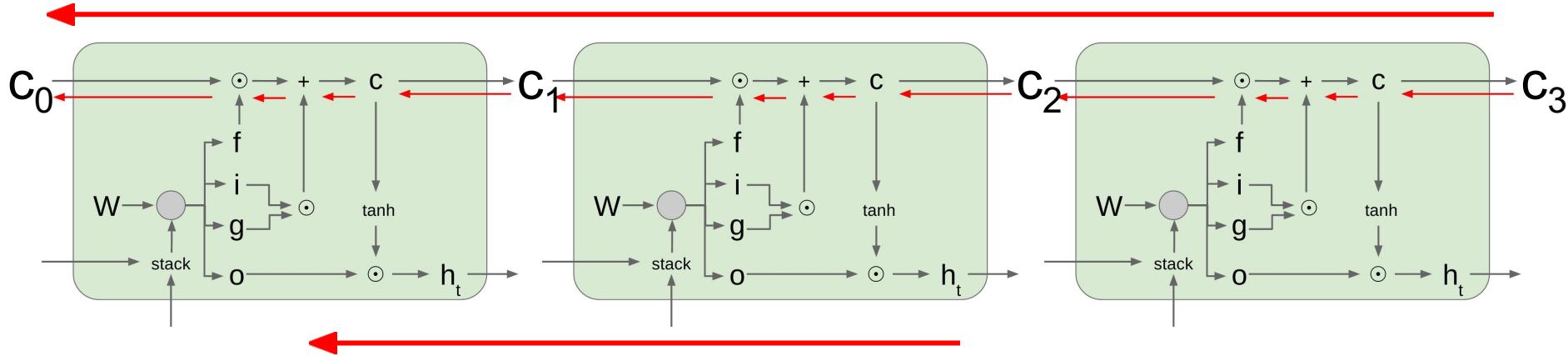
Uninterrupted gradient flow!



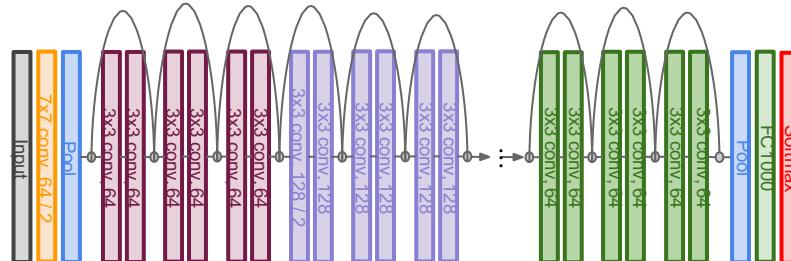
# Long Short Term Memory (LSTM): Gradient Flow

[Hochreiter et al., 1997]

Uninterrupted gradient flow!



Similar to ResNet!



In between:  
**Highway Networks**

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

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

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

# 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

# Other RNN Variants

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

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

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

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

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

[*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:

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

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

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

MUT3:

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

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

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

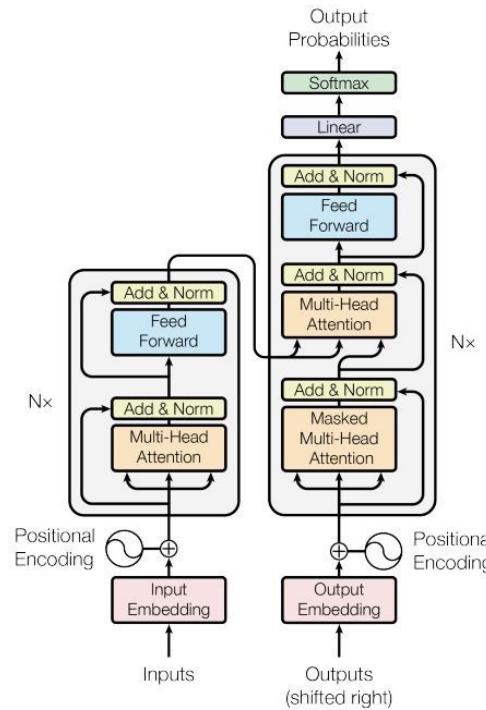
[*LSTM: A Search Space Odyssey*, Greff et al., 2015]

# Recently in Natural Language Processing...

## New paradigms for reasoning over sequences

[“Attention is all you need”, Vaswani et al., 2018]

- New “Transformer” architecture no longer processes inputs sequentially; instead it can operate over inputs in a sequence in parallel through an attention mechanism
- Has led to many state-of-the-art results and pre-training in NLP, for more interest see e.g.
  - “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”, Devlin et al., 2018
  - OpenAI GPT-2, Radford et al., 2018



# 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.

# Transformers for Vision

- LSTM is a good default choice
- Use variants like GRU if you want faster compute and less parameters
- Use transformers (not covered in this lecture) as they are dominating NLP models
  - We need more work studying vision models in tandem with transformers

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.