

CS60050

Recurrent Neural Networks

Somak Aditya
Assistant Professor
Department of CSE, IIT Kharagpur



Recurrent Neural Network: A Bit of History

It is quite fundamental and was mentioned in rudimentary form in many papers:

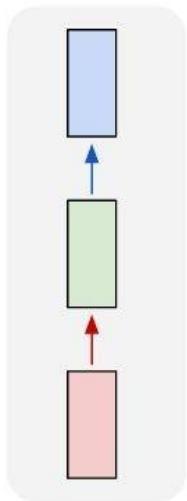
1. McCulloch and Pitts discusses Nets with and without circles. *McCulloch, W.S., Pitts, W. A logical calculus of the ideas immanent in nervous activity. Bulletin of Mathematical Biophysics 5, 115–133 (1943).*
2. Rumelhart et al. 1985 chapter and Jordan et al. 1986 explicitly discusses Recurrent Nets.
 1. *Rumelhart, David E; Hinton, Geoffrey E, and Williams, Ronald J (Sept. 1985). Learning internal representations by error propagation. Tech. rep. ICS 8504. San Diego, California: Institute for Cognitive Science, University of California.*
 2. *Jordan, Michael I. (May 1986). Serial order: a parallel distributed processing approach. Tech. rep. ICS 8604. San Diego, California: Institute for Cognitive Science, University of California*

Many interesting theoretical results:

- Any seq-to-seq function computable by a Turing machine can also be computed by an RNN ([Siegelmann and Sontag, 1992](#)) [On the computational power of neural nets](#).

“Vanilla” Neural Network

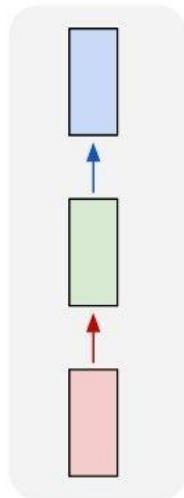
one to one



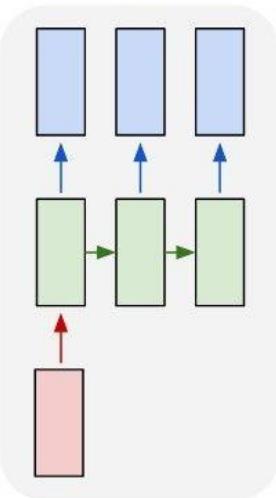
Vanilla Neural Networks

Recurrent Neural Networks: Process Sequences

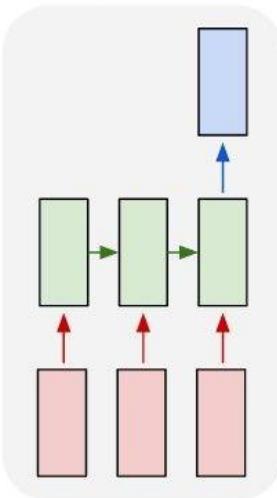
one to one



one to many



many to one

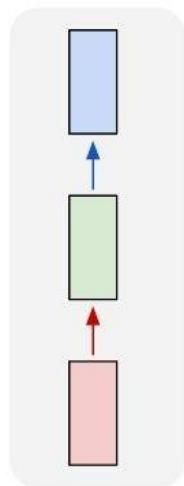


e.g. Image Captioning
image -> sequence of words

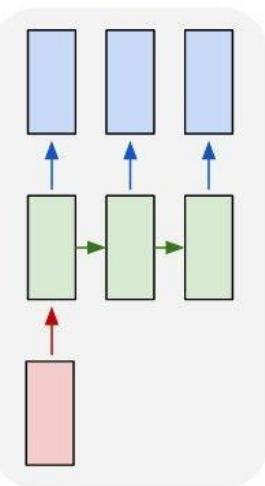
e.g. Sentence classification sent sentiment
action prediction
sequence of video frames -> action class

Recurrent Neural Networks: Process Sequences

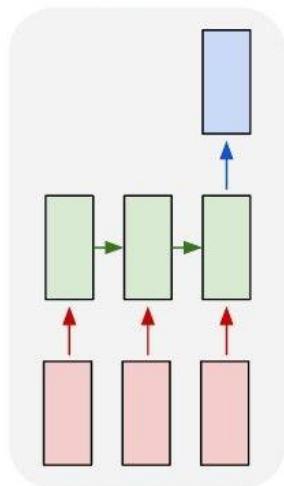
one to one



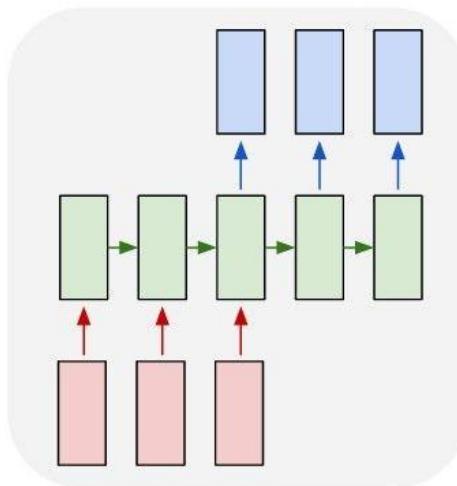
one to many



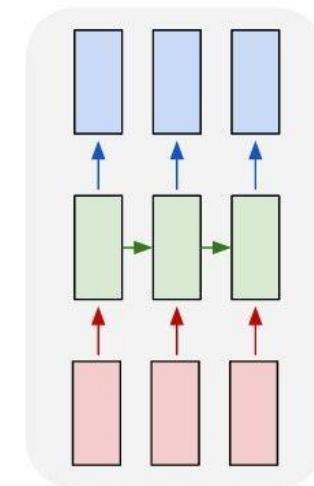
many to one



many to many



many to many



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

e.g. Sentence
Parts-of-speech tagging

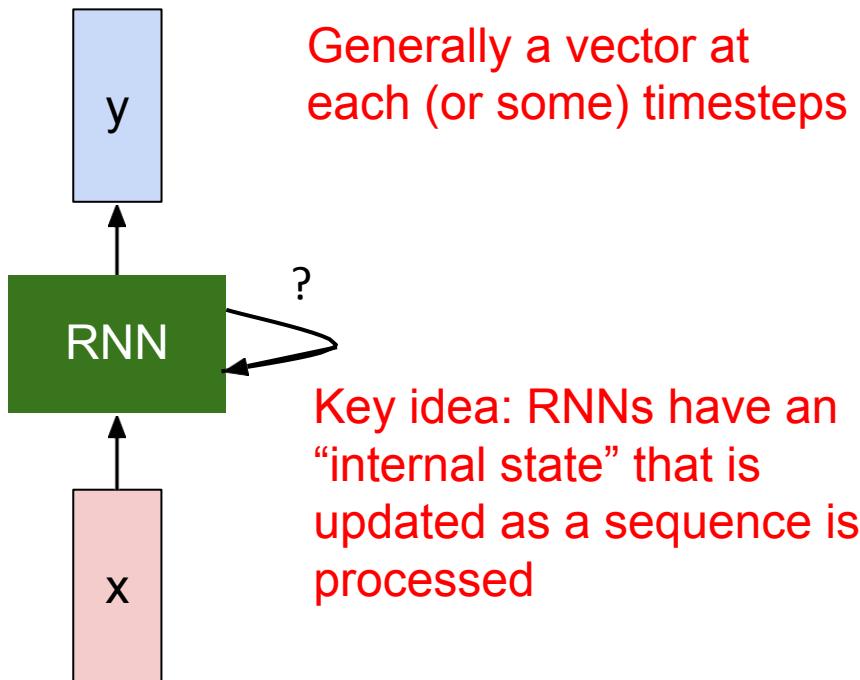
Sequential Processing of Non-Sequence Data

Classify images by taking a series of “glimpses”

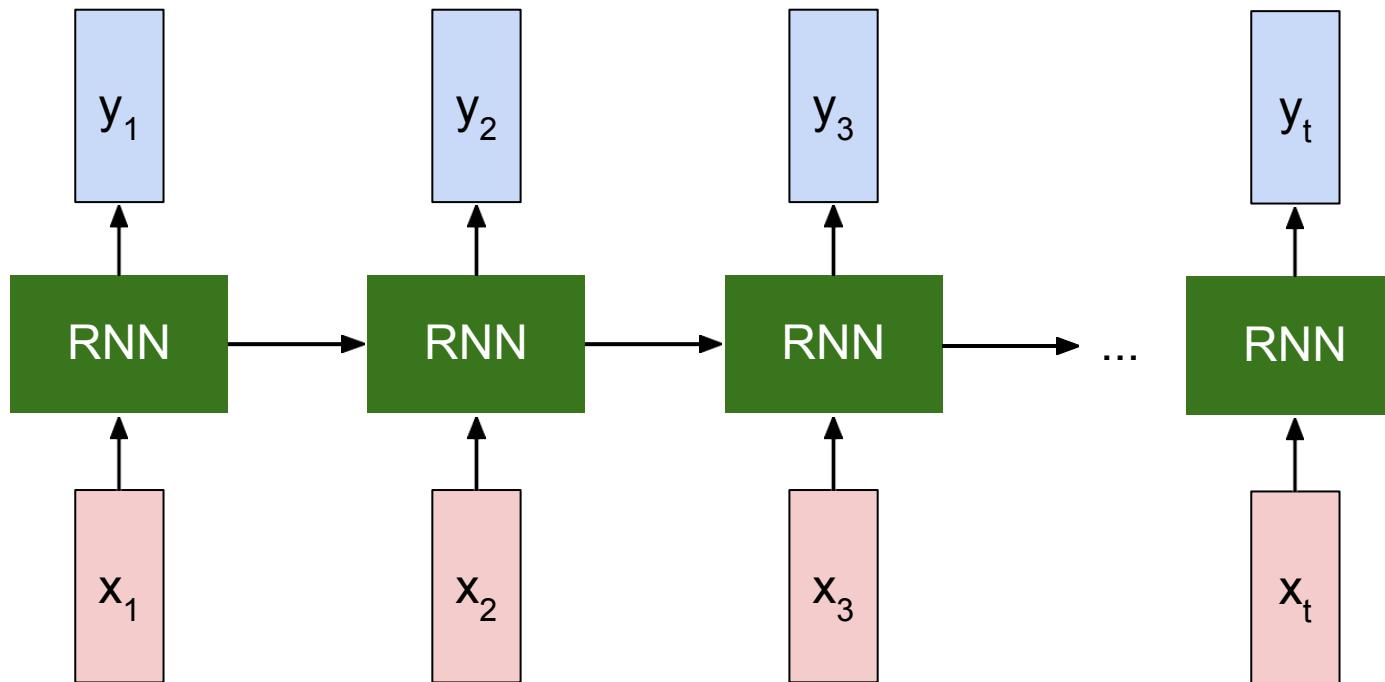


Ba, Mnih, and Kavukcuoglu, “Multiple Object Recognition with Visual Attention”, ICLR 2015.
Gregor et al, “DRAW: A Recurrent Neural Network For Image Generation”, ICML 2015
Figure copyright Karol Gregor, Ivo Danihelka, Alex Graves, Danilo Jimenez Rezende, and Daan Wierstra, 2015. Reproduced with permission.

RNN Output Generation

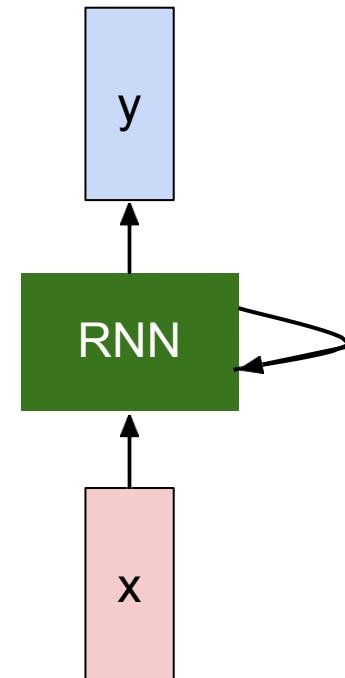


Unrolled RNN



RNN hidden state update

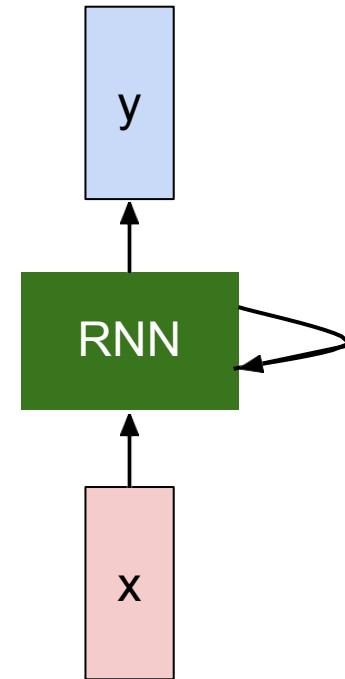
We can process a sequence of vectors \mathbf{x} by applying a **recurrence formula** at every time step:



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.

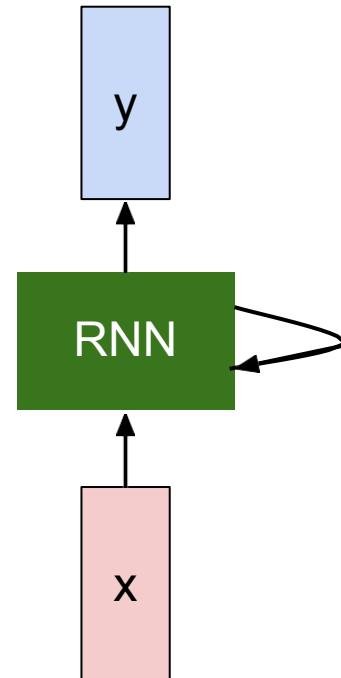
RNN output generation

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

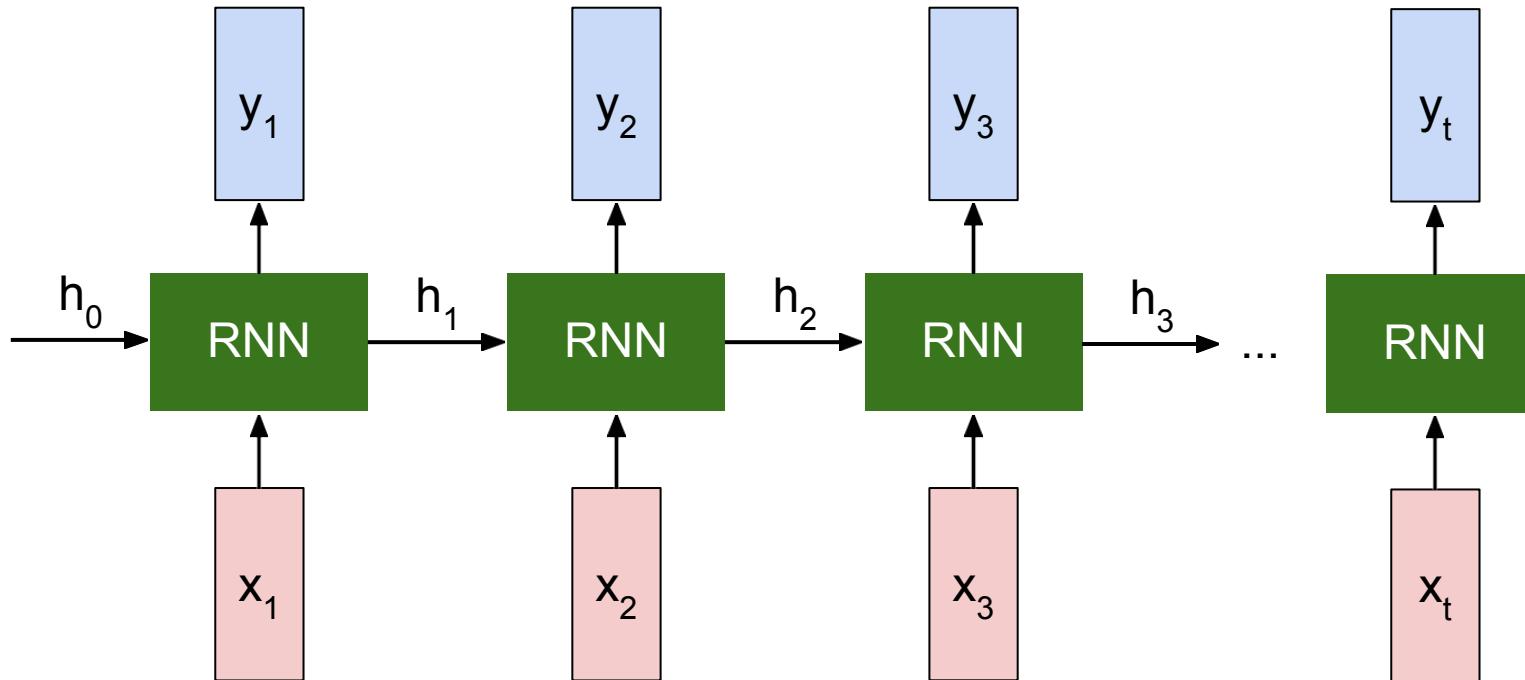
$$y_t = f_{W_{hy}}(h_t)$$

output new state

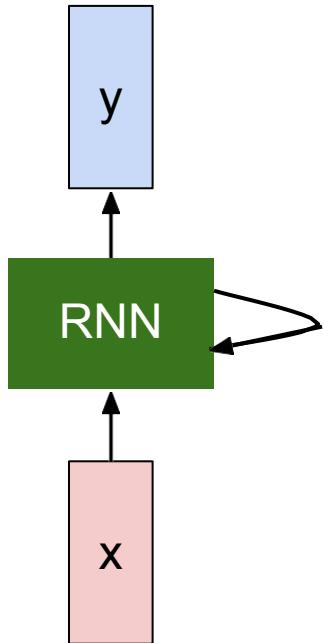
another function
with parameters W_o



Recurrent Neural Network



(Vanilla) Recurrent Neural Network



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

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

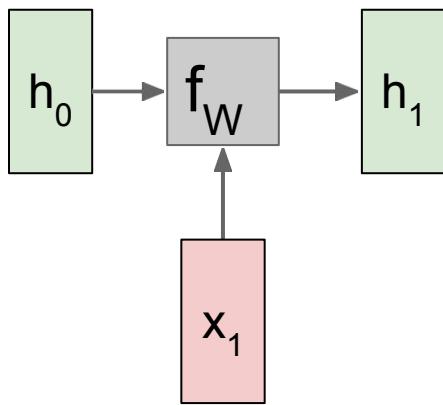


$$\mathbf{h}_t = \tanh(W_{hh}\mathbf{h}_{t-1} + W_{xh}\mathbf{x}_t)$$

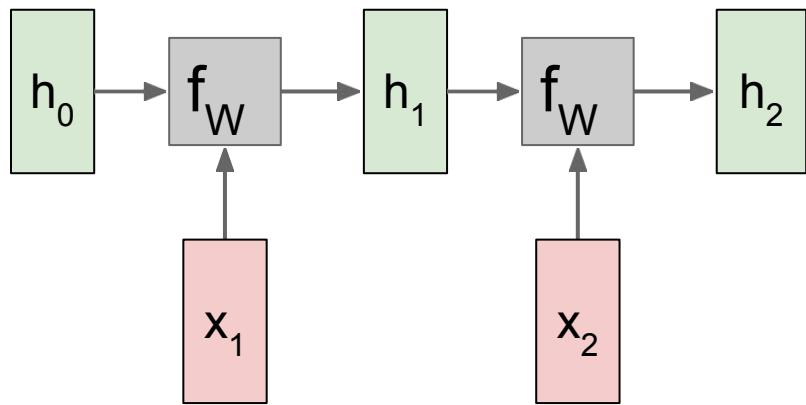
$$y_t = W_{hy}\mathbf{h}_t$$

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

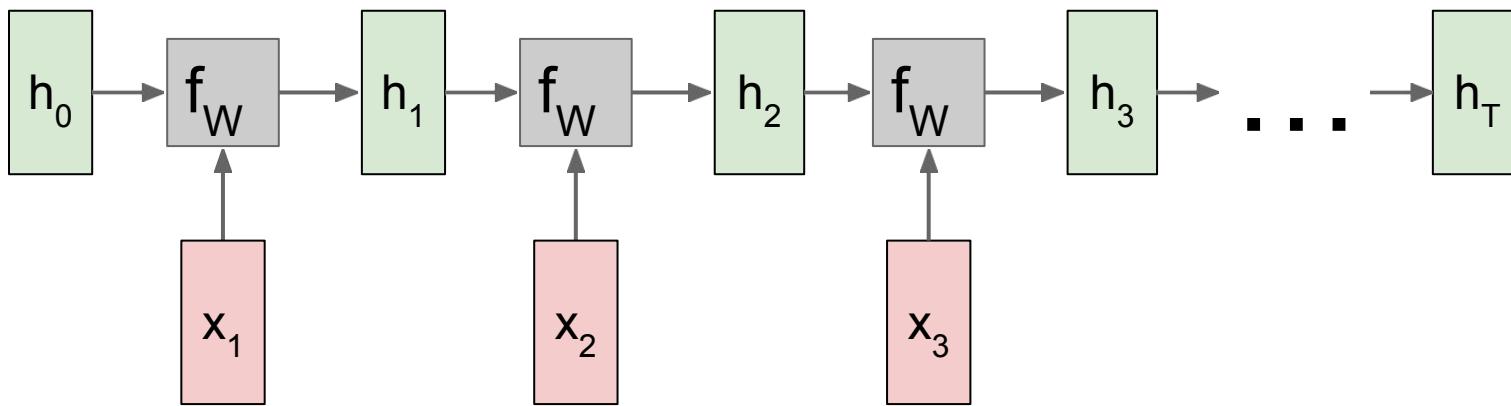
RNN: Computational Graph



RNN: Computational Graph

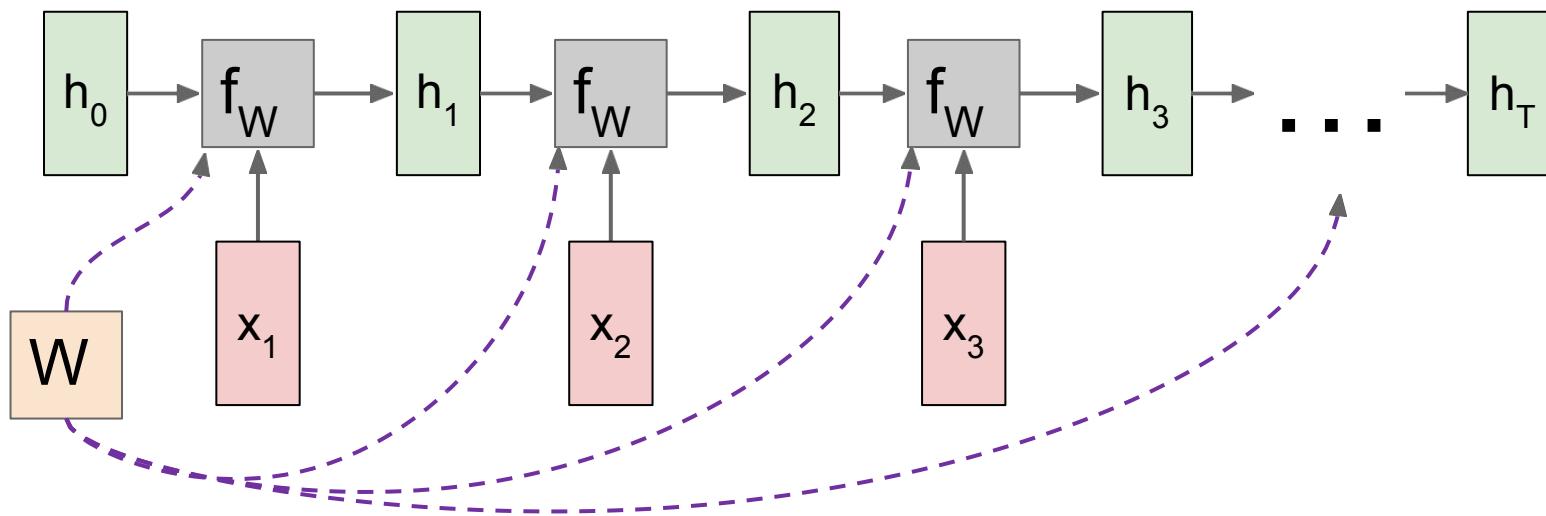


RNN: Computational Graph

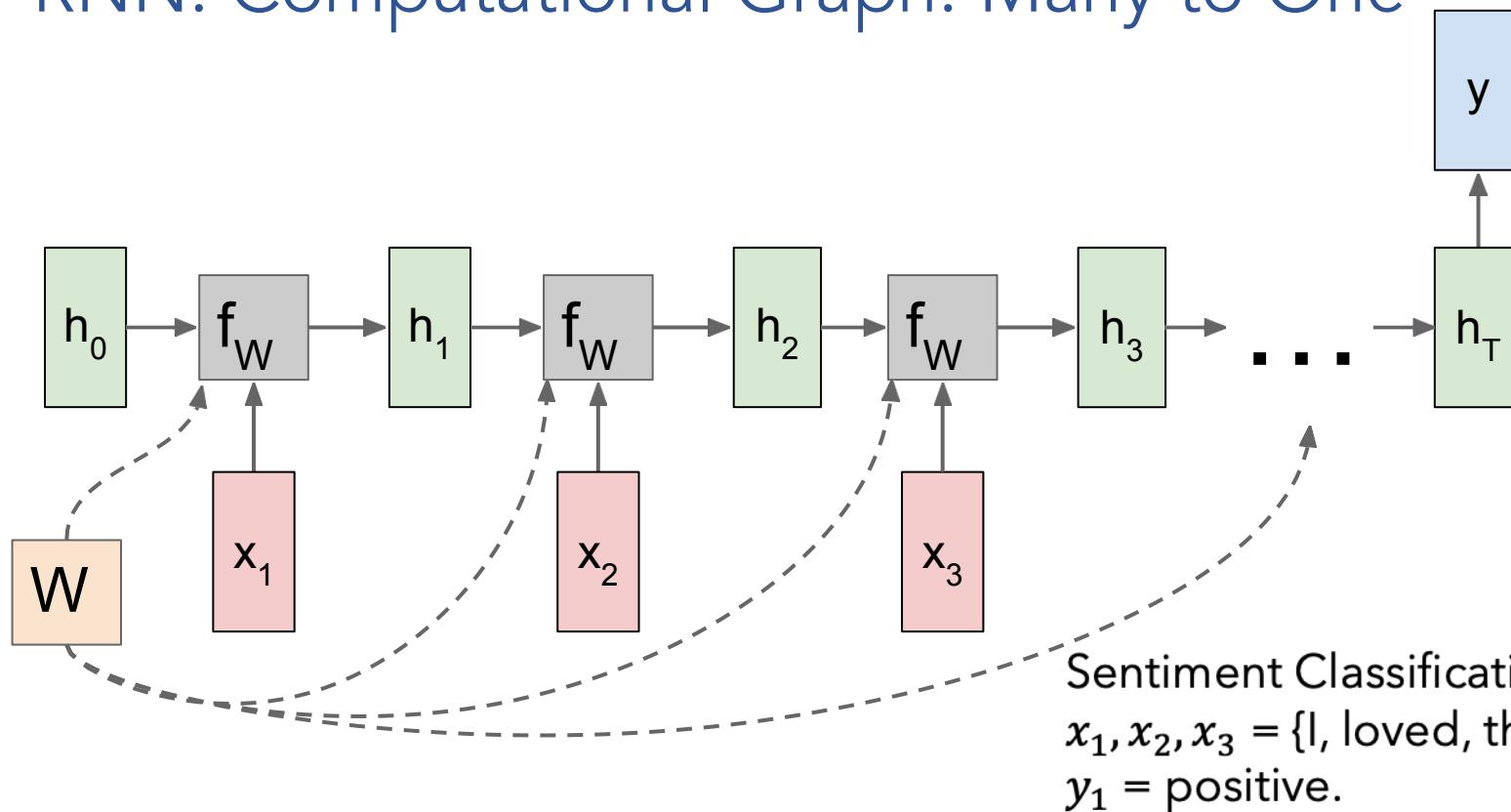


RNN: Computational Graph

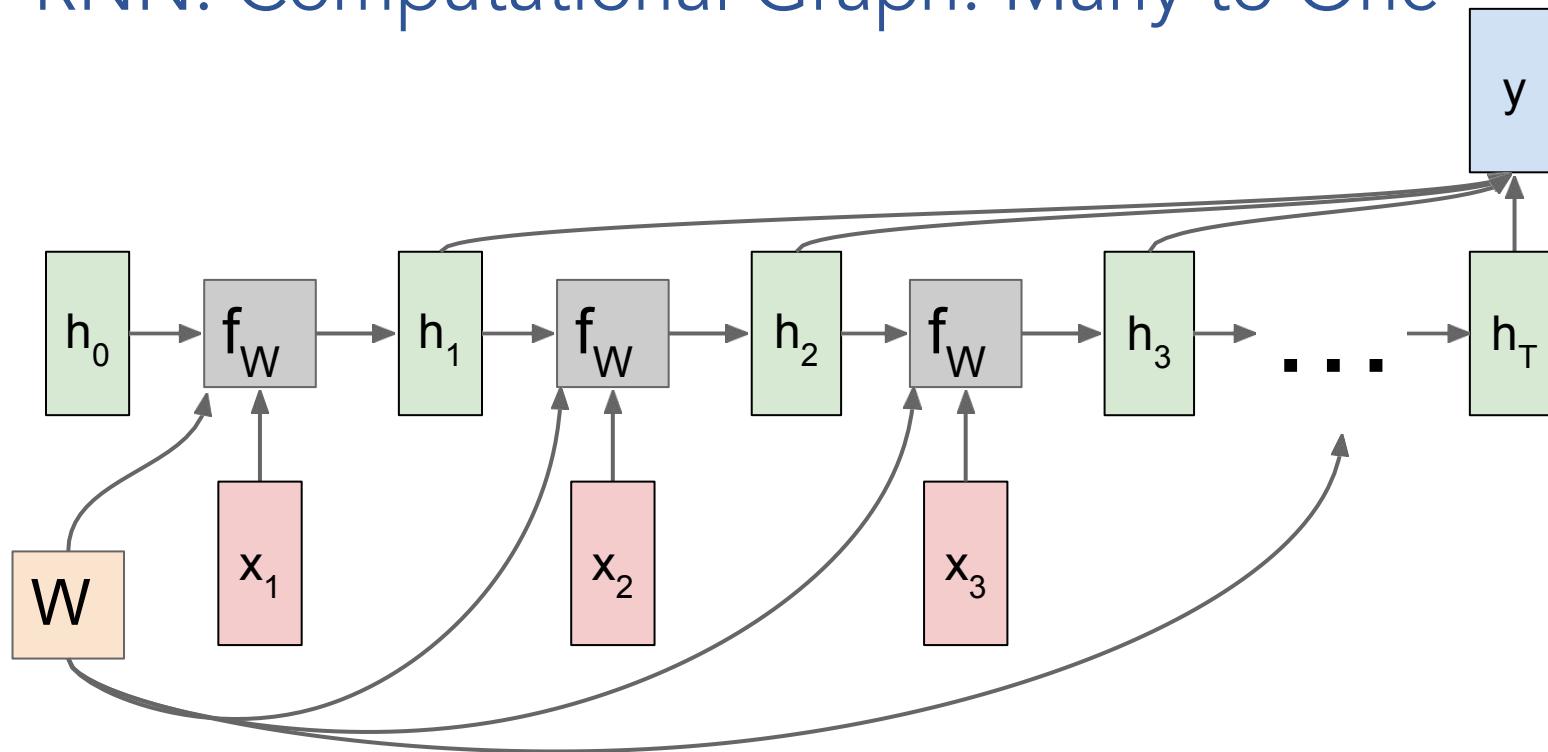
Re-use the same weight matrix at every time-step



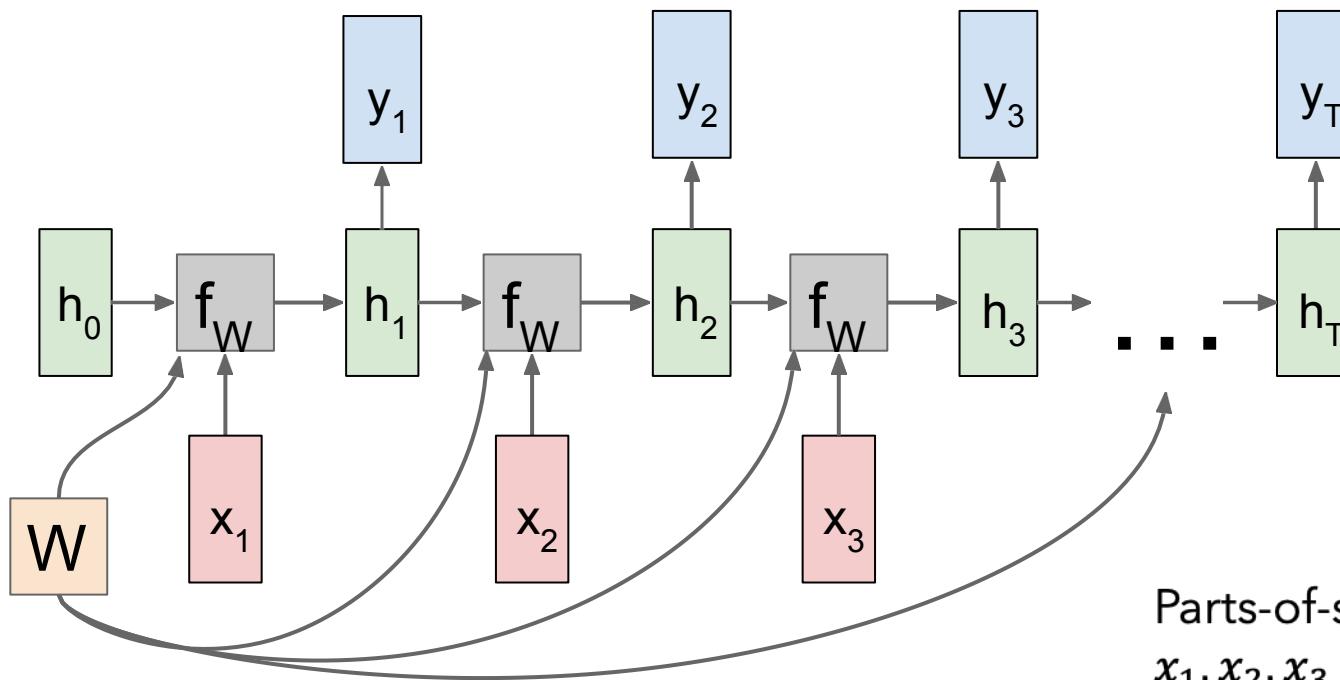
RNN: Computational Graph: Many to One



RNN: Computational Graph: Many to One

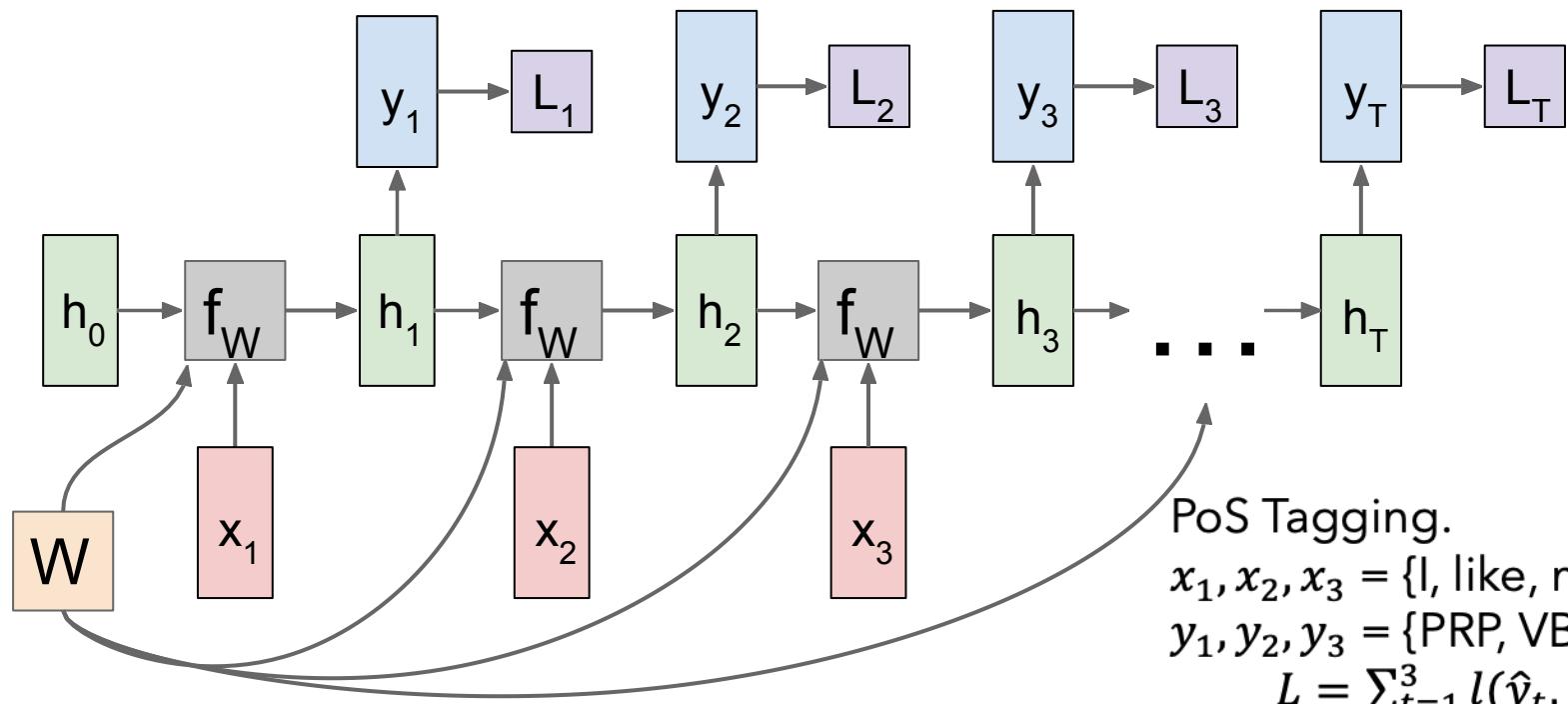


RNN: Computational Graph: Many to Many

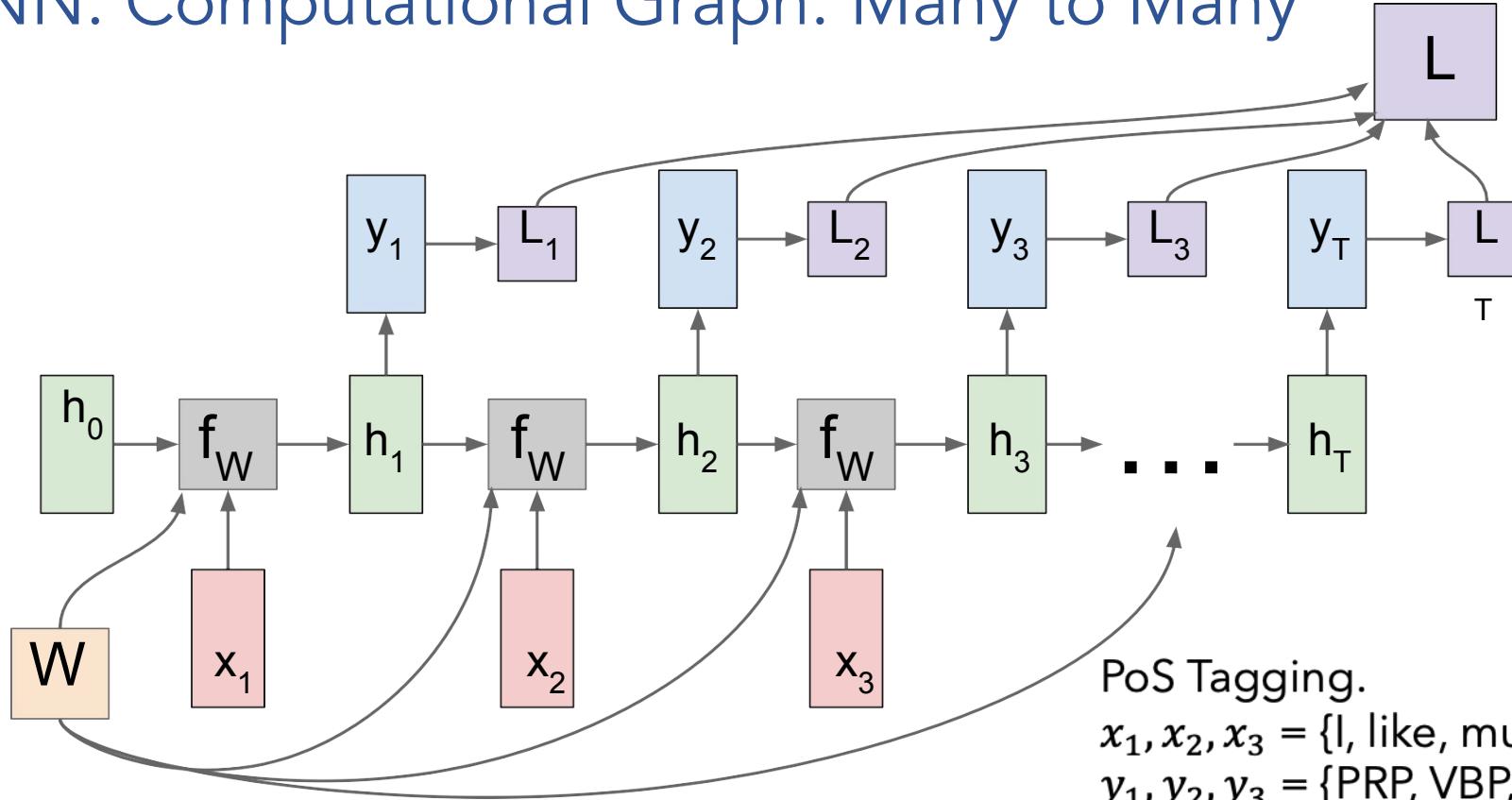


Parts-of-speech Tagging.
 $x_1, x_2, x_3 = \{\text{l, like, music}\}$.
 $y_1, y_2, y_3 = \{\text{PRP, VBP, NN}\}$.

RNN: Computational Graph: Many to Many



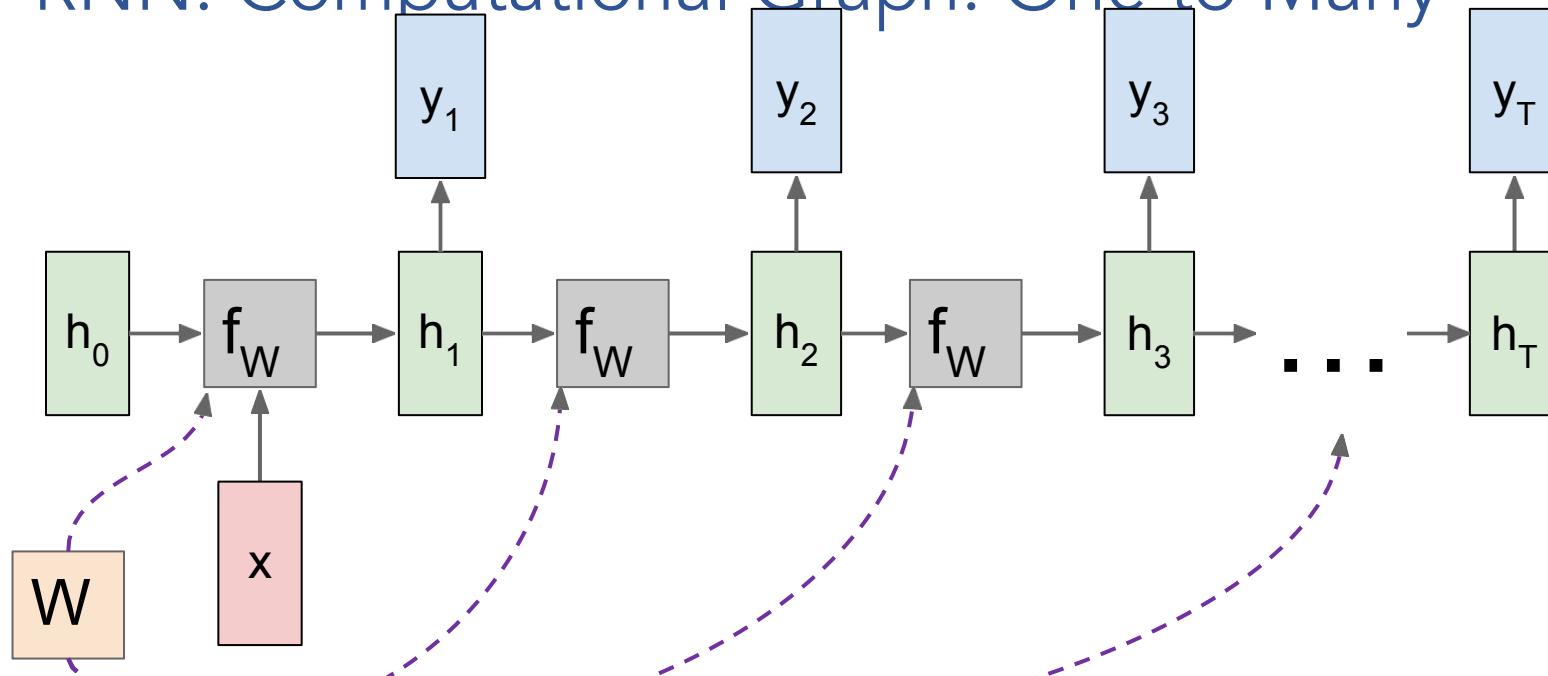
RNN: Computational Graph: Many to Many



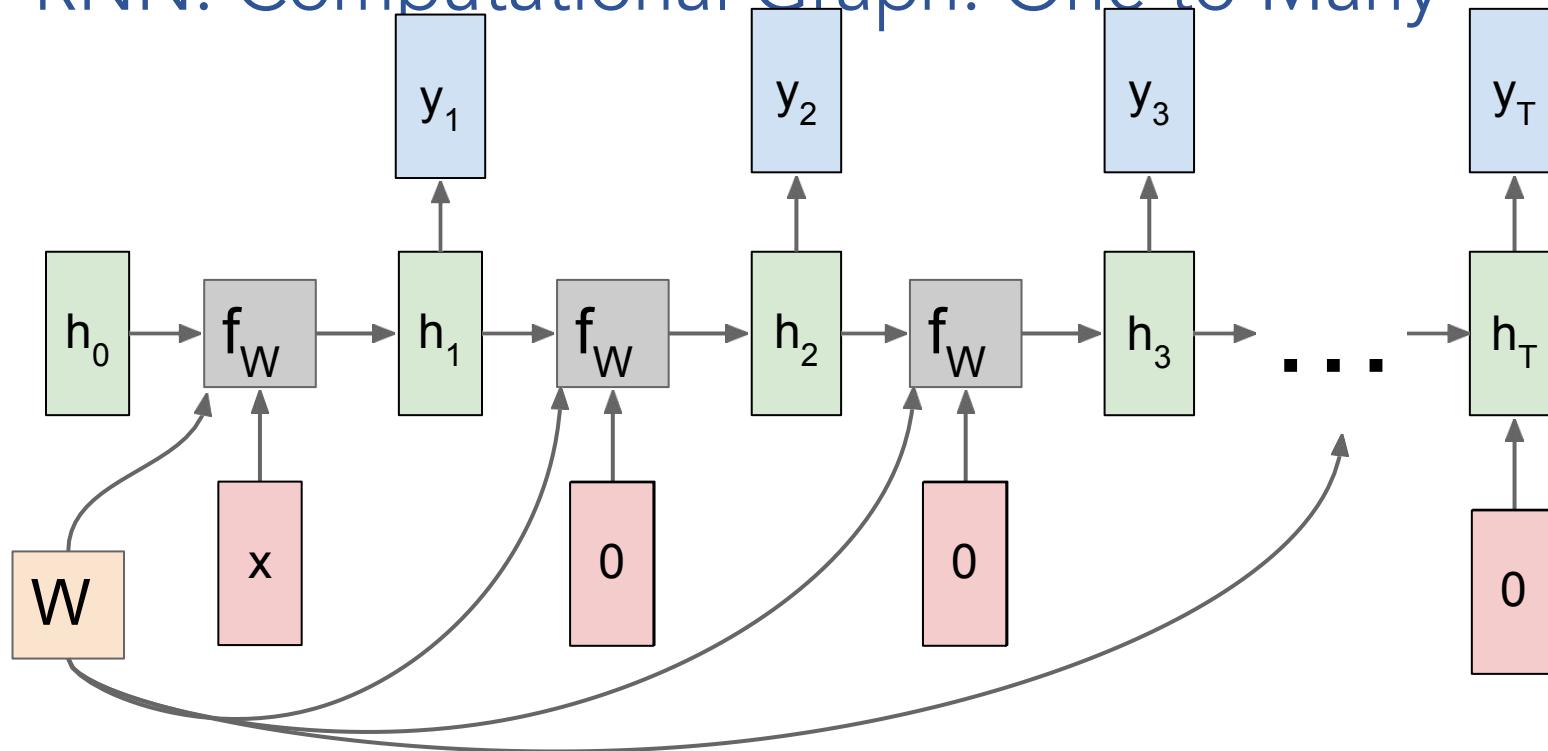
PoS Tagging.

$$\begin{aligned}
 x_1, x_2, x_3 &= \{\text{I, like, music}\}. \\
 y_1, y_2, y_3 &= \{\text{PRP, VBP, NN}\}. \\
 L &= \sum_{t=1}^3 l(\hat{y}_t, y_t)
 \end{aligned}$$

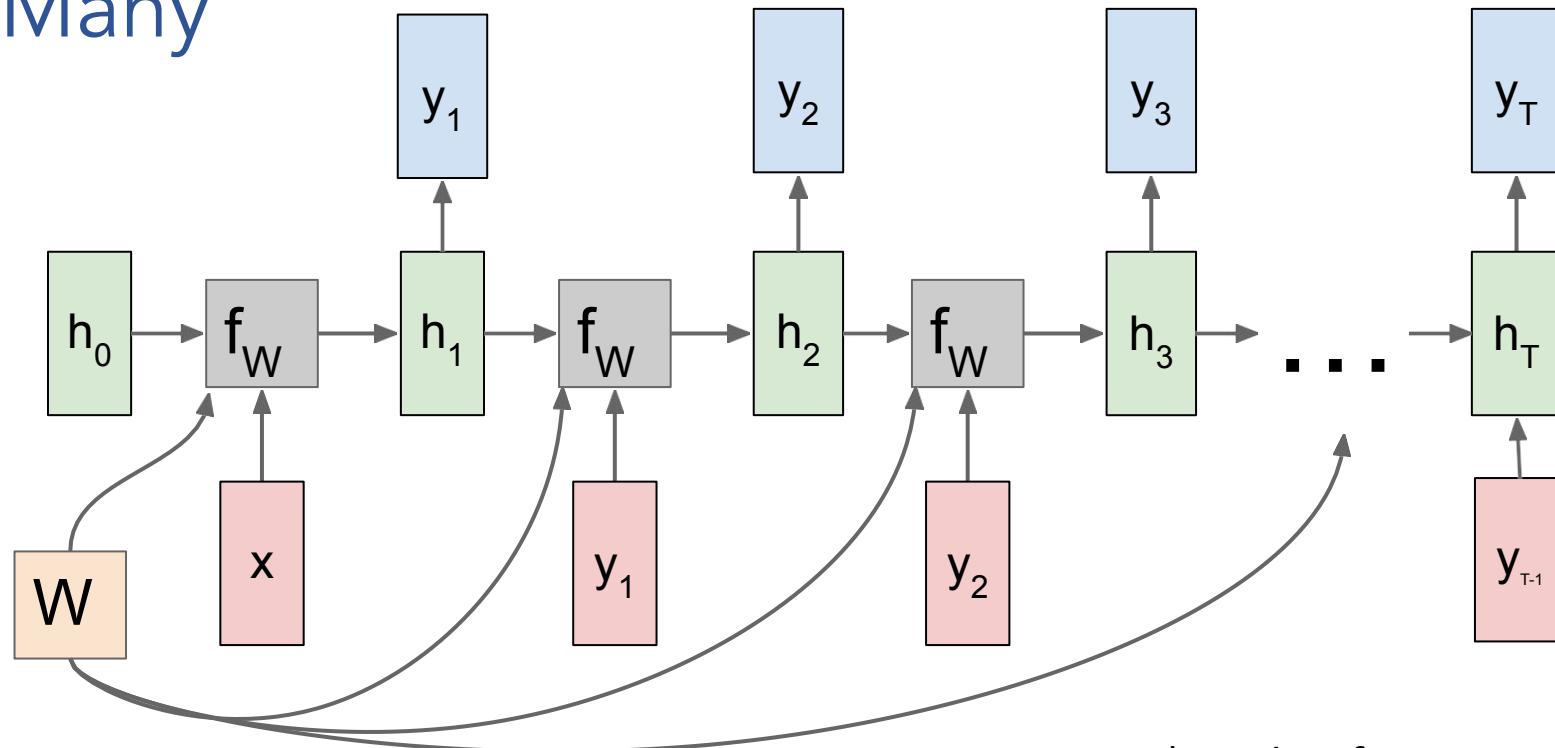
RNN: Computational Graph: One to Many



RNN: Computational Graph: One to Many



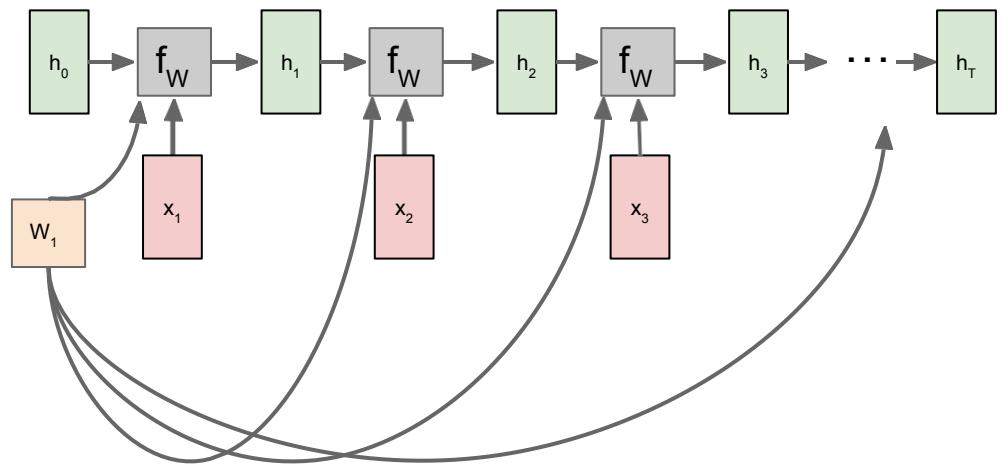
RNN: Computational Graph: One to Many



*Teacher-forcing
– putting the groundtruth

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

Many to one: Encode input sequence in a single vector

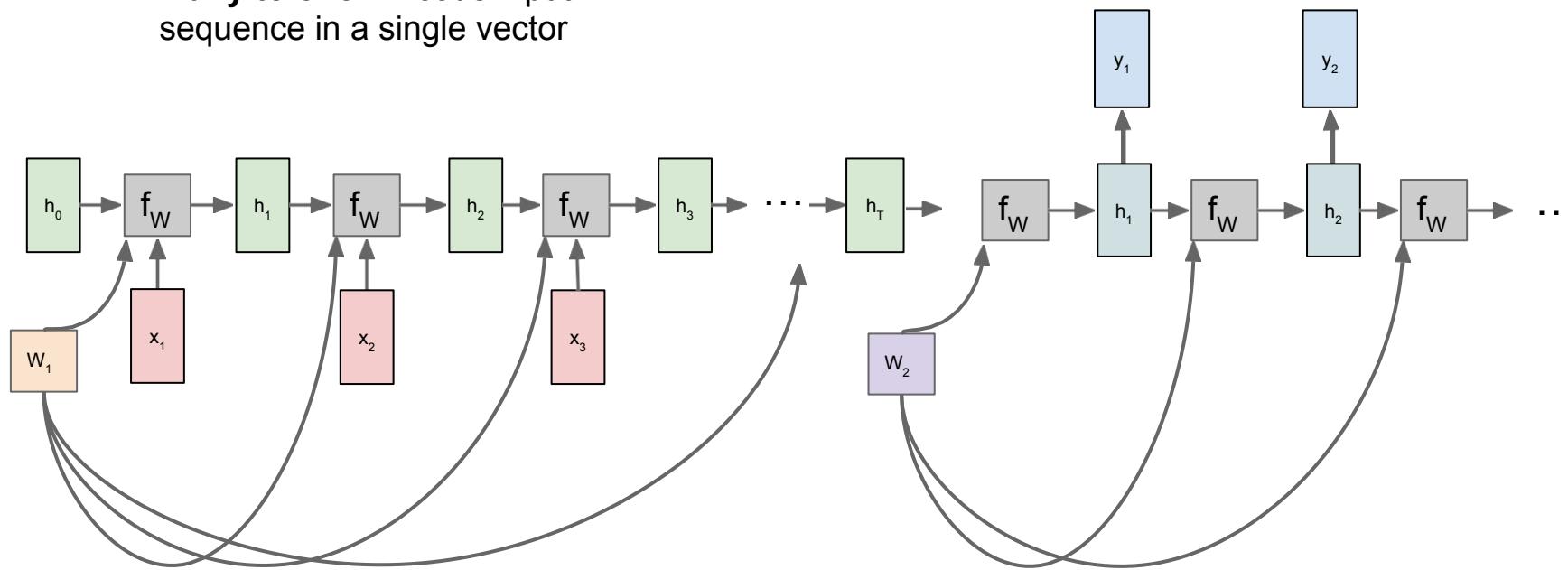


Sutskever et al, "Sequence to Sequence Learning with Neural Networks", NIPS 2014

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



Forward Pass Example: RNN Character Language Model

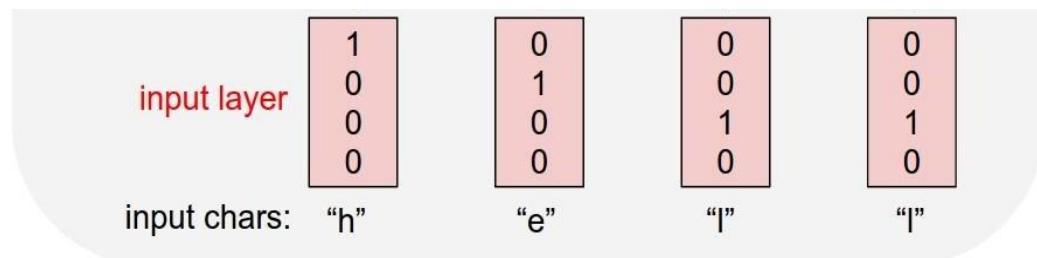
Example: Character-level Language Model

Vocabulary:
[h,e,l,o]

Example training
sequence: "hello"

Task: Predict the next
character

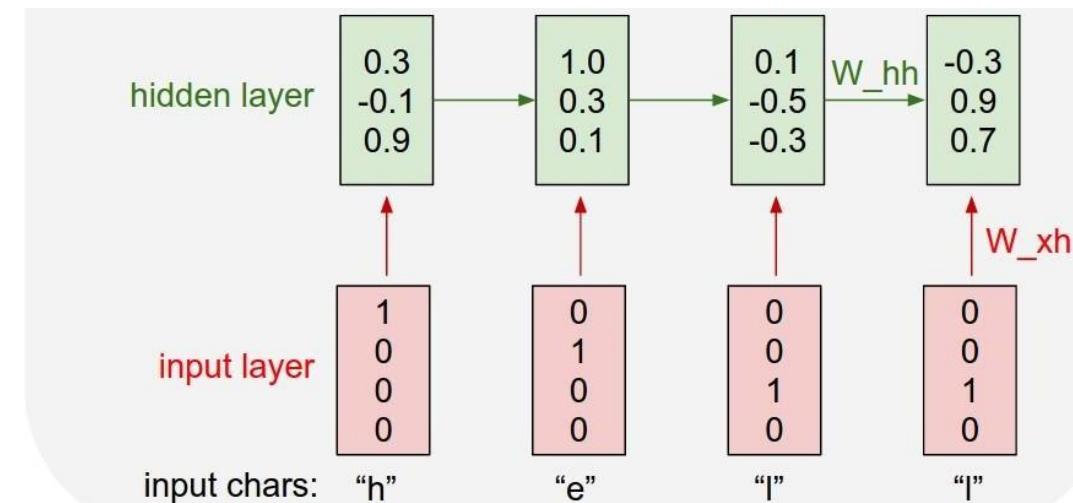
Why? At test time, we
can use this ability to
generate words,
sentences, even
paragraphs



Example: Character-level Language Model

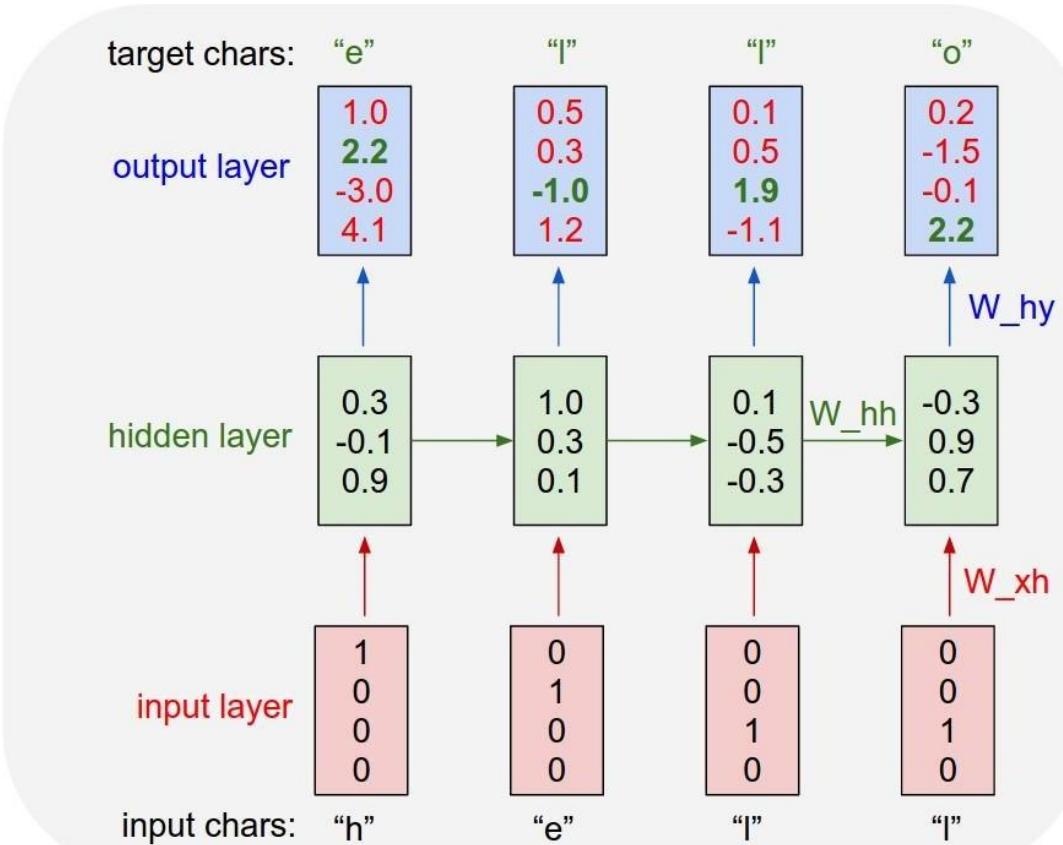
- Vocabulary:
[h,e,l,o]
- Example training sequence: “hello”
- Task: Predict the next character
 - Why? At test time, we can use this ability to generate words, sentences, even paragraphs

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



Example: Character-level Language Model

- Vocabulary:
[h,e,l,o]
- Example training
sequence: "hello"

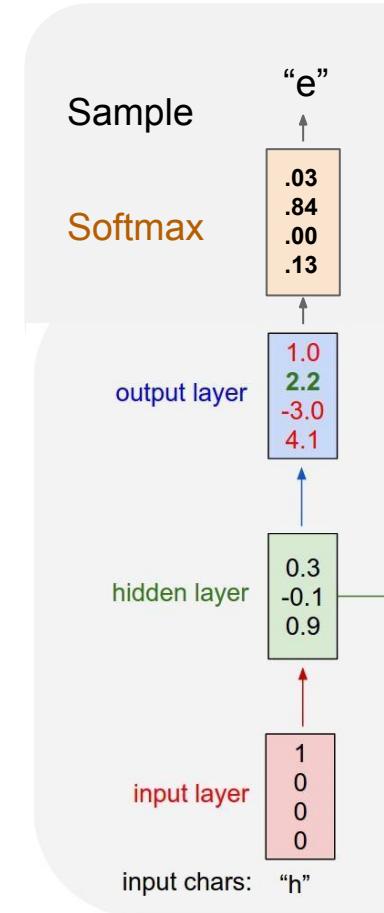


Example: Character-level Language Model

Vocabulary:

[h,e,l,o]

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

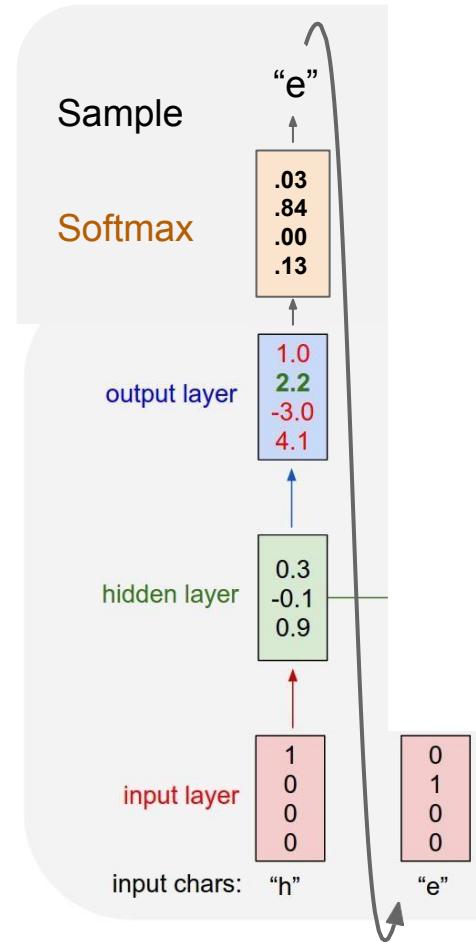


Example: Character-level Language Model

Sampling:

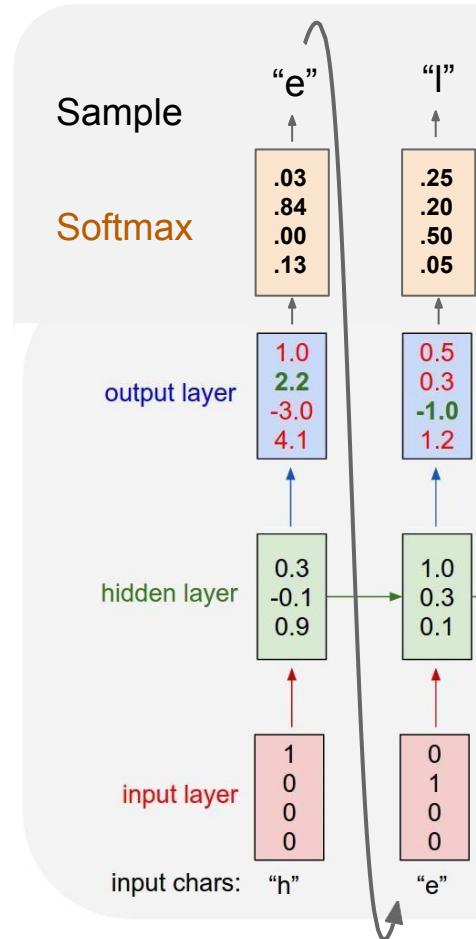
[h,e,l,o]

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



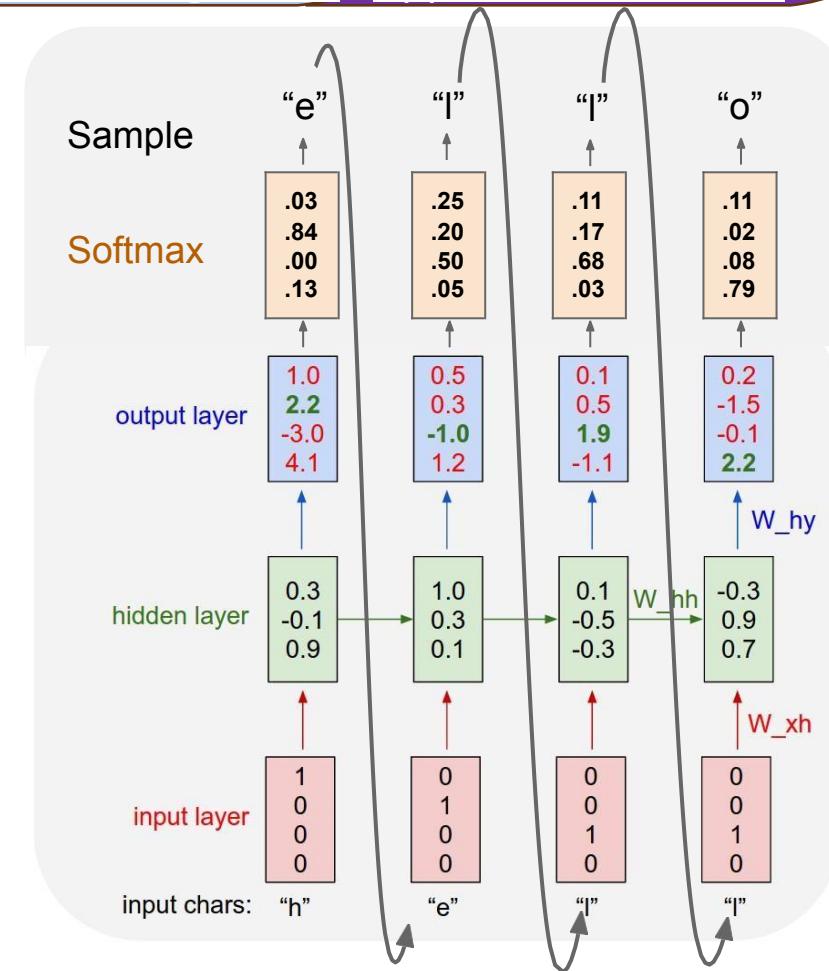
Example: Character-level Language Model Sampling: Vocabulary: [h,e,l,o]

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



Example: Character-level Language Model Sampling: Vocabulary: [h,e,l,o]

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

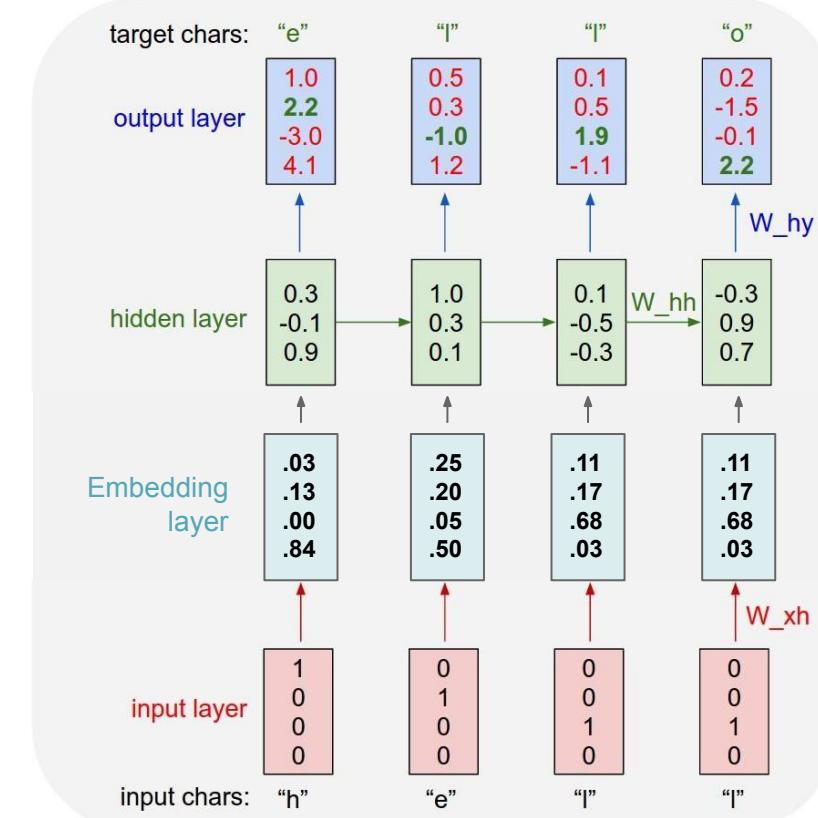


Example:

Character-level Language Model Sampling

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

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

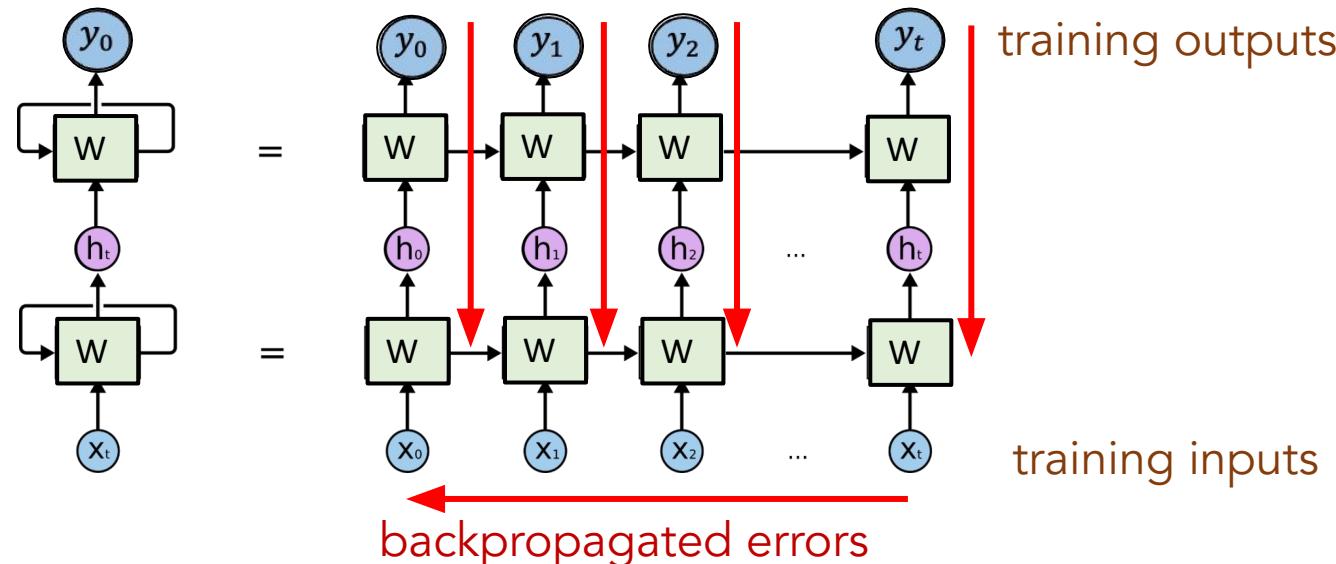


Backpropagation through Time

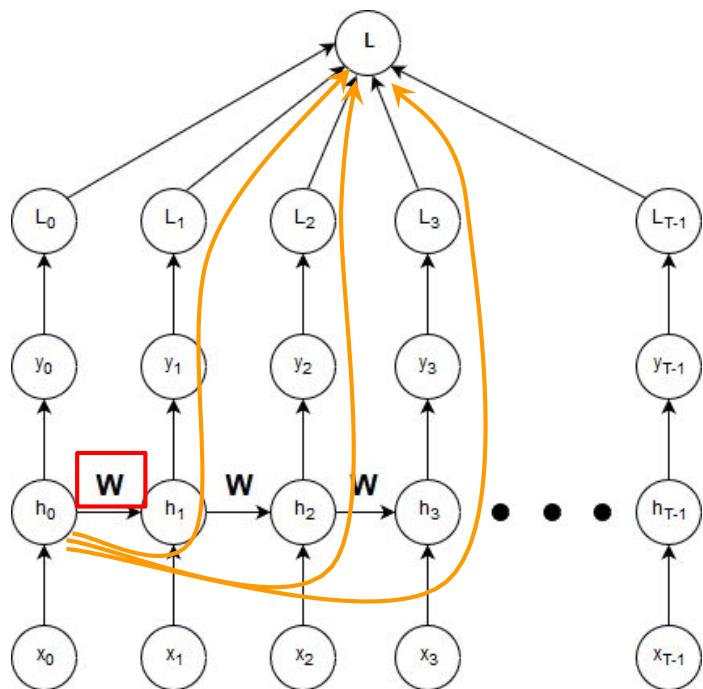
Training RNN's

RNNs can be trained using “backpropagation through time.”

- applying normal backprop to the unrolled network.



Backpropagation Through Time (BPTT)

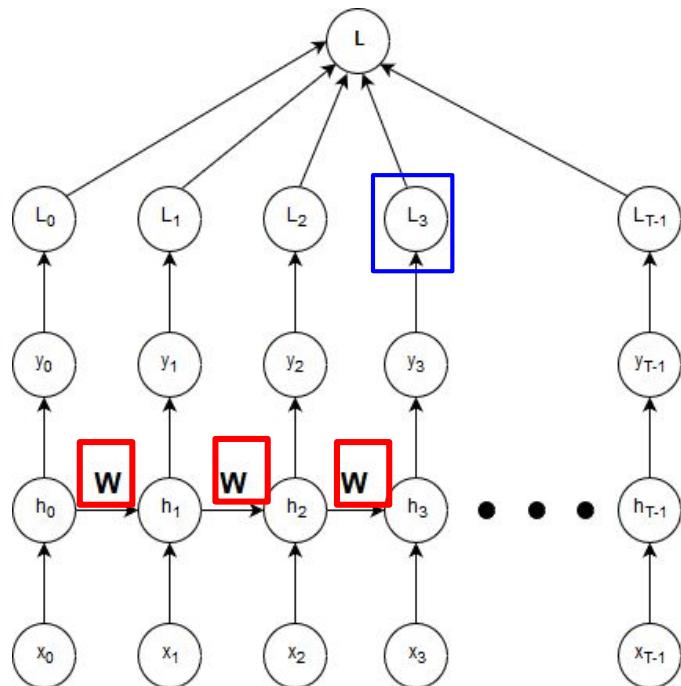


Update the weight matrix:

$$\mathbf{W} \leftarrow \mathbf{W} - \alpha \frac{\partial L}{\partial \mathbf{W}}$$

- Issue: \mathbf{W} occurs each timestep
- **Every** path from \mathbf{W} to L is one dependency
- Find all paths from \mathbf{W} to L !

Backpropagation as two summations



First summation over L

$$\frac{\partial L}{\partial \mathbf{W}} = \sum_{j=0}^{T-1} \frac{\partial L_j}{\partial \mathbf{W}}$$

Second summation over h :

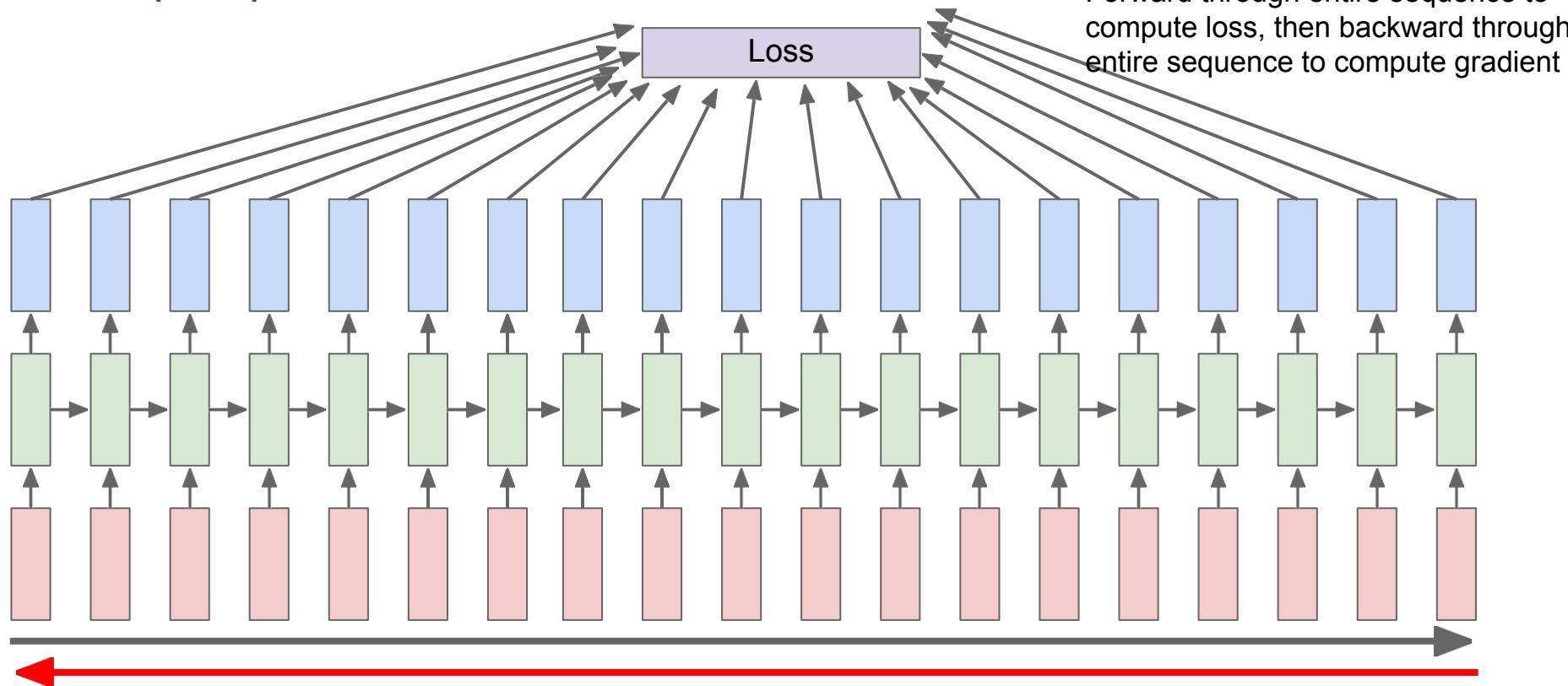
Each L_j depends on the weight matrices *before it*

$$\frac{\partial L_j}{\partial \mathbf{W}} = \sum_{k=1}^j \frac{\partial L_j}{\partial h_k} \frac{\partial h_k}{\partial \mathbf{W}}$$

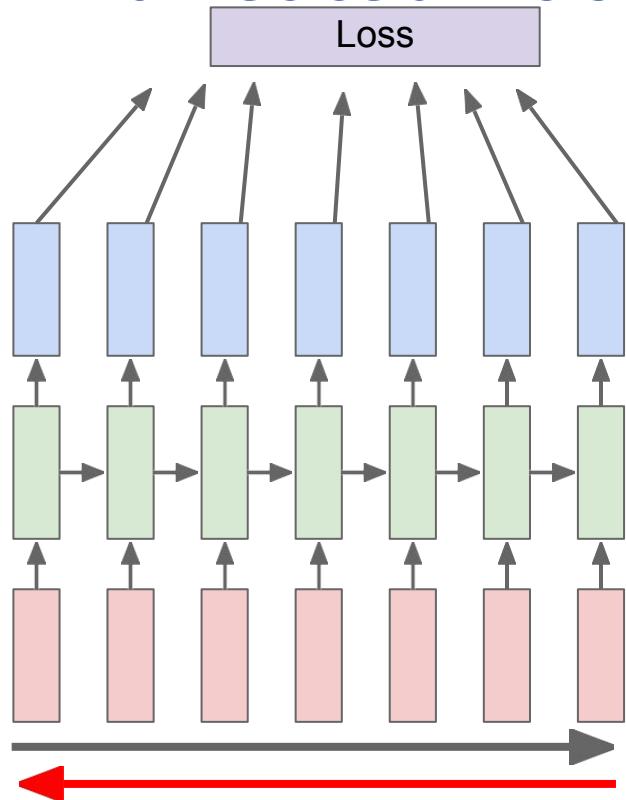


L_j depends on all h_k before it.

Backpropagation through time

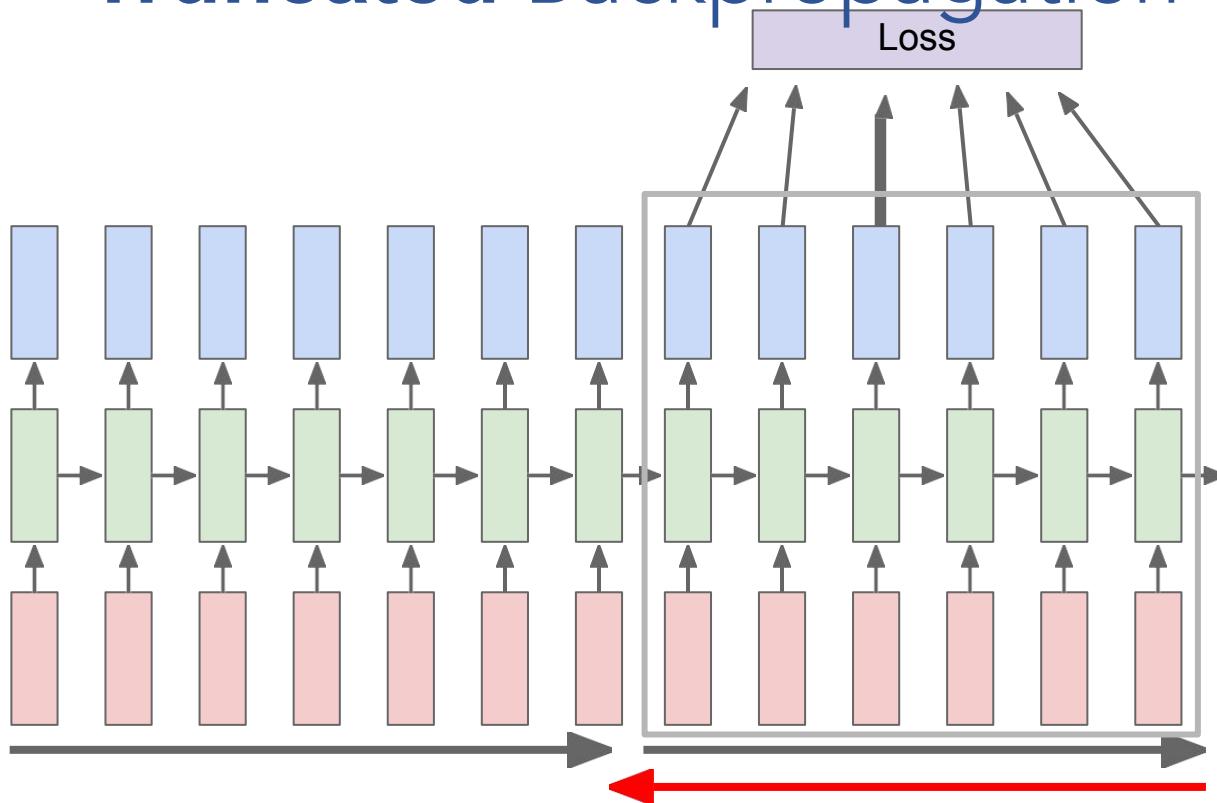


Truncated Backpropagation through time



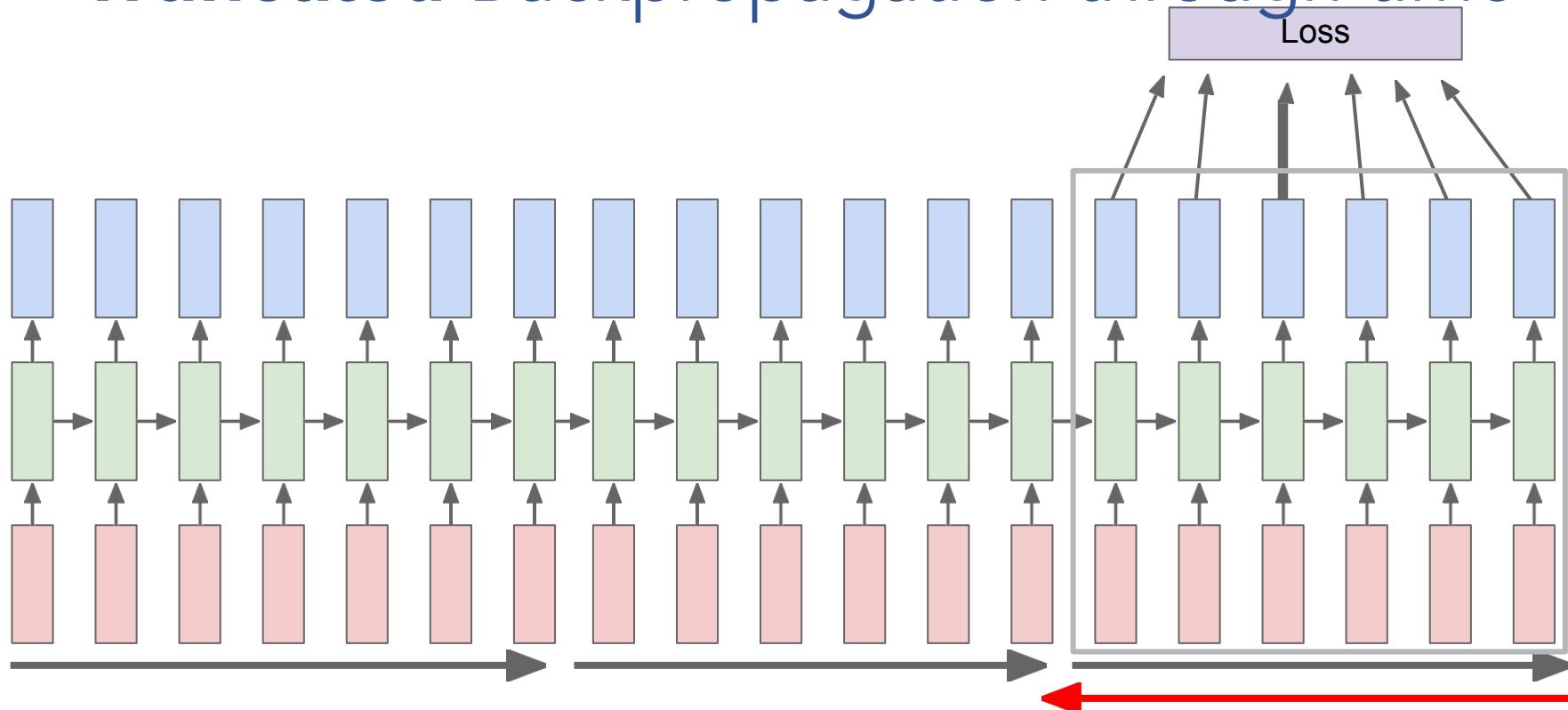
Run forward and backward
through chunks of the
sequence instead of whole
sequence

Truncated Backpropagation through time



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

Truncated Backpropagation through time



RNN Applications and Issues

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.randn(hidden_size, vocab_size)*0.01 # input to hidden
22 whh = np.random.randn(hidden_size, hidden_size)*0.01 # hidden to hidden
23 why = np.random.randn(vocab_size, hidden_size)*0.01 # hidden to output
24 bh = np.zeros((hidden_size, 1)) # hidden bias
25 by = np.zeros((vocab_size, 1)) # output bias
26
27 def lossFun(inputs, targets, hprev):
28     """
29     inputs,targets are both list of integers.
30     hprev is Hx1 array of initial hidden state
31     returns the loss, gradients on model parameters, and last hidden state
32     """
33     xs, hs, ys, ps = {}, {}, {}, {}
34     hs[-1] = np.copy(hprev)
35     loss = 0
36     # forward pass
37     for t in xrange(len(inputs)):
38         xs[t] = np.zeros((vocab_size,1)) # encode in 1-of-K representation
39         xs[t][inputs[t]] = 1
40         hs[t] = np.tanh(np.dot(wkh, xs[t]) + np.dot(whh, hs[t-1]) + bh) # hidden state
41         ys[t] = np.exp(hs[t]) / np.sum(np.exp(xs[t])) # 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         dwhh, dwkh = np.zeros_like(whh), np.zeros_like(wkh)
46         dwhy, dbh = np.zeros_like(why), np.zeros_like(bh)
47         dhnext = np.zeros((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             dbh += dy
53             dh = np.dot(why.T, dy) + dhnext # backprop into h
54             dhrw = (1 - hs[t]**2) * dh # backprop through tanh nonlinearity
55             dh += dhrw
56             dwkh += np.dot(dhrw, xs[t].T)
57             dwhy += np.dot(dhrw, hs[t-1].T)
58             dhnext = np.dot(why.T, dhrw)
59             for dparam in [dwkh, dwhh, dwhy, dbh, dy]:
60                 np.clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients
61             return loss, dwkh, dwhh, dwhy, dbh, dy, hs[len(inputs)-1]
62
63 def sample(h, seed_ix, n):
64     """
65     sample a sequence of integers from the model
66     h is memory state, seed_ix is seed letter for first time step
67     """
68     x = np.zeros((vocab_size, 1))
69     x[seed_ix] = 1
70     ixes = []
71     for t in xrange(n):
72         h = np.tanh(np.dot(wkh, x) + np.dot(whh, h) + bh)
73         y = np.dot(why, h) + by
74         p = np.exp(y) / np.sum(np.exp(y))
75         ix = np.random.choice(range(vocab_size), p=p.ravel())
76         x = np.zeros((vocab_size, 1))
77         x[ix] = 1
78         ixes.append(ix)
79     return ixes
80
81 n, p, 0
82 mwkh, mwhh, mwhy = np.zeros_like(wkh), 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[i] for ix in sample_ix)
97         print '----\n%s\n----' % (txt, )
98
99     # forward seq_length characters through the net and fetch gradient
100    loss, dwkh, dwhh, dwhy, dbh, dy, 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 dparam, param, mem in zip([dwkh, dwhh, dwhy, dbh, dy],
106                                 [wkh, whh, why, bh, by],
107                                 [mwkh, mwhh, mwhy, mbh, mby]):
108        mem += dparam * dparam
109        param += -learning_rate * dparam / np.sqrt(mem + 1e-8) # adagrad update
110
111    p += seq_length # move data pointer
112    n += 1 # iteration counter

```

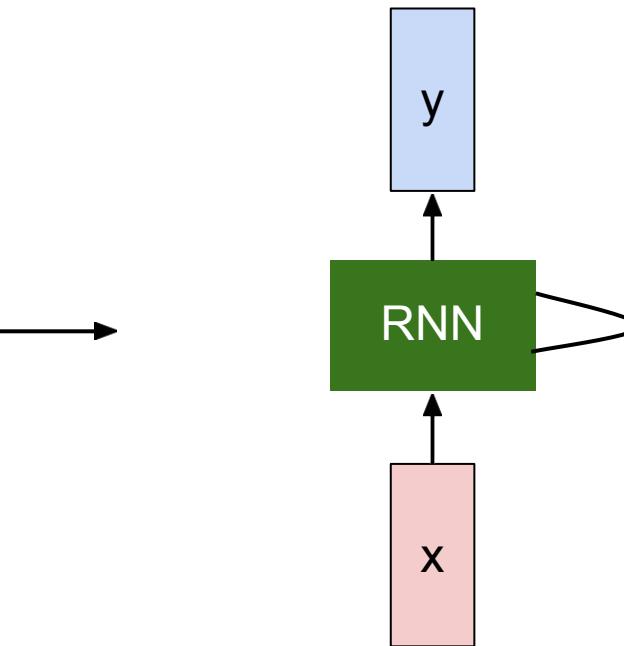
(<https://gist.github.com/karpathy/d4dee566867f8291f086>)

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-iafhatawiaoahrdemot 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" fomescerliund
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.

PANDARUS:

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

Second Senator:

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

DUKE VINCENTIO:

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

Second Lord:

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

Clown:

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

VIOLA:

I'll drink it.

VIOLA:

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

KING LEAR:

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

Image Captioning

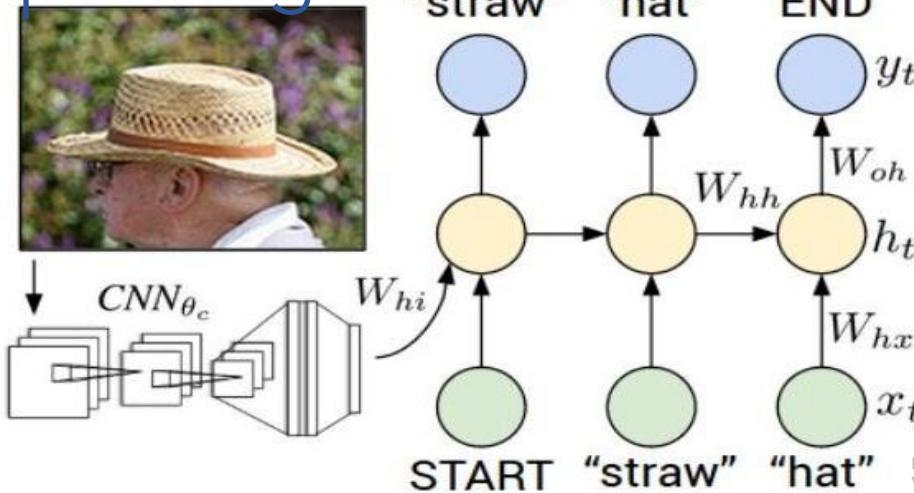


Figure from Karpathy et al, "Deep Visual-Semantic Alignments for Generating Image Descriptions", CVPR 2015; figure copyright IEEE, 2015.
Reproduced for educational purposes.

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

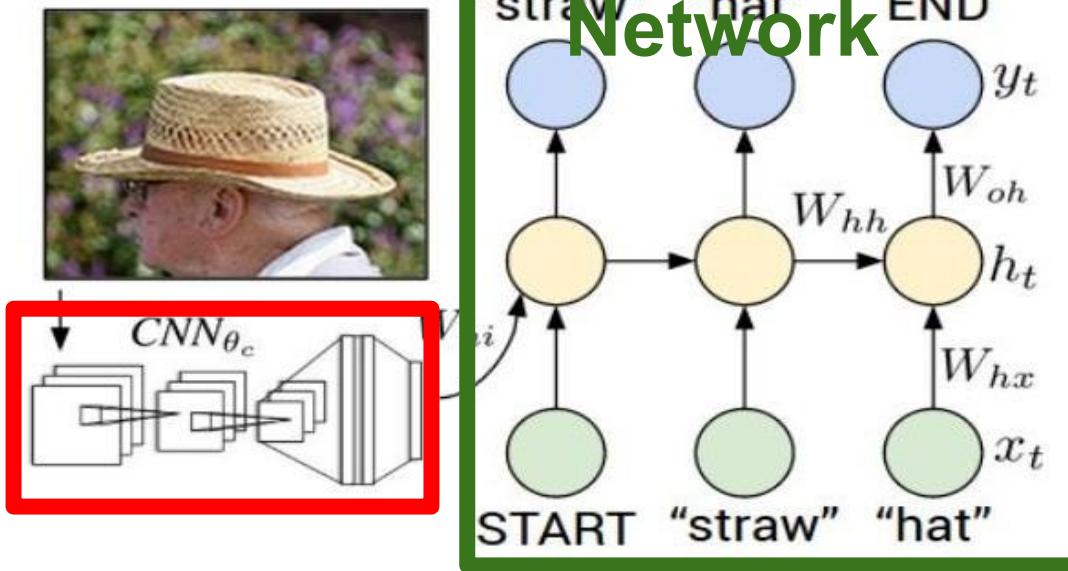
Deep Visual-Semantic Alignments for Generating Image Descriptions, Karpathy and Fei-Fei

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

Image Captioning: Example Results



A cat sitting on a suitcase on the floor



A cat is sitting on a tree branch



A dog is running in the grass with a frisbee



A white teddy bear sitting in the grass



Two people walking on the beach with surfboards



A tennis player in action on the court



Two giraffes standing in a grassy field



A man riding a dirt bike on a dirt track

Image Captioning: Failure Cases



A woman is holding a cat in her hand



A person holding a computer mouse on a desk



A woman standing on a beach holding a surfboard

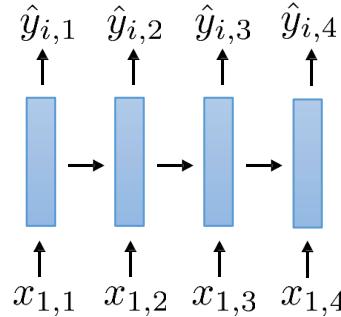


A bird is perched on a tree branch



A man in a baseball uniform throwing a ball

RNNs are extremely deep networks



If sequence length is 100, backpropagate through 100 steps.

vanishing gradients = gradient signal from later steps never reaches the earlier steps

this prevents the RNN from “remembering” things from the beginning!

“vanishing gradients”

$$\frac{d\mathcal{L}}{dW^{(1)}} = \frac{dz^{(1)}}{dW^{(1)}} \frac{da^{(1)}}{dz^{(1)}} \frac{dz^{(2)}}{da^{(1)}} \frac{d\mathcal{L}}{dz^{(2)}}$$

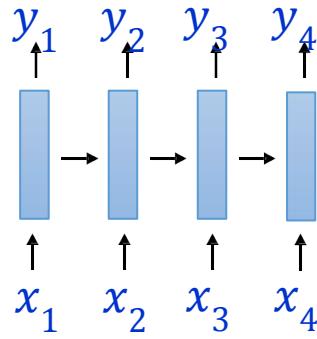
$$\frac{d\mathcal{L}}{dW^{(1)}} = J_1 J_2 J_3 \dots J_n \frac{d\mathcal{L}}{dz^{(n)}}$$

If we multiply many numbers together, what will we get?

- If most of the numbers are < 1 , we get 0
- If most of the numbers are > 1 , we get infinity
- We only get a reasonable answer if the numbers are all close to 1!

“exploding gradients” could fix with gradient clipping

RNNs are difficult to train



Long sequence length

If sequence length is 100, backpropagate through 100 steps.

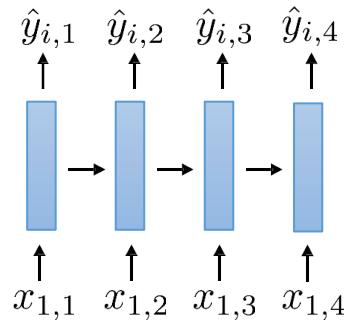
- vanishing gradients
- gradient signal from later steps never reaches the earlier steps.

This prevents the RNN from “remembering” things from the beginning!

$$\frac{d\mathcal{L}}{dW^{(1)}} = \frac{dz^{(1)}}{dW^{(1)}} \frac{da^{(1)}}{dz^{(1)}} \frac{dz^{(2)}}{da^{(1)}} \frac{d\mathcal{L}}{dz^{(2)}}$$

$$\frac{d\mathcal{L}}{dW^{(1)}} = J_1 J_2 J_3 \dots J_n \frac{d\mathcal{L}}{dz^{(n)}}$$

RNNs are extremely deep networks



$$\frac{d\mathcal{L}}{dW^{(1)}} = \frac{dz^{(1)}}{dW^{(1)}} \frac{da^{(1)}}{dz^{(1)}} \frac{dz^{(2)}}{da^{(1)}} \frac{d\mathcal{L}}{dz^{(2)}}$$

$$\frac{d\mathcal{L}}{dW^{(1)}} = J_1 J_2 J_3 \dots J_n \frac{d\mathcal{L}}{dz^{(n)}}$$

If we multiply many many numbers together, what will we get?

- If most of the numbers are < 1 , we get 0 “vanishing gradients” Hard to fix
- If most of the numbers are > 1 , we get infinity “exploding gradients” could fix with gradient clipping
- We only get a reasonable answer if the numbers are all close to 1!

Promoting better gradient flow

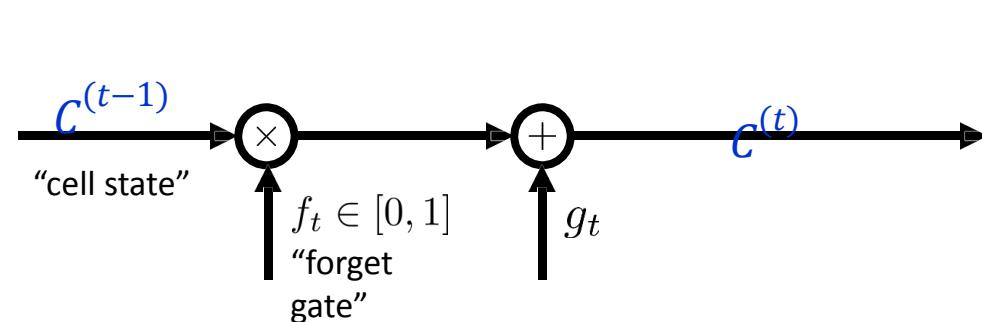
Basic idea: we would really like the gradients to be close to 1

$$C^t = Q(C^{t-1}, x_t)$$

Intuition: want $\frac{dQ}{dC^{(t-1)}} \approx 1$ if we choose to remember $C^{(t-1)}$

for each unit, we have a sort of “logic gate” that decides

- if “remembering,” copy the previous activation
- if “forgetting,” overwrite it with something based on the current input



$$f_t \in [0, 1]$$



$$C^{(t)} = C^{(t-1)} f_t + g_t$$

$$\frac{dQ}{dC^{(t-1)}} = f_t \in [0, 1]$$

Next Class: Long Short Term Memory (LSTM)

Vanilla RNN LSTM

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

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

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

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