CS60050
Recurrent Neural Networks

Somak Aditya
Assistant Professor
Department of CSE, IIT Kharagpur

Slide Courtesy: Prof. Sudeshna Sarkar (IIT KGP), Prof. Li Fei Fei, Prof. Yunzhu Li, Prof. Ruohan Gao (Stanford Univ)

Recurrent Neural Network: A Bit of History

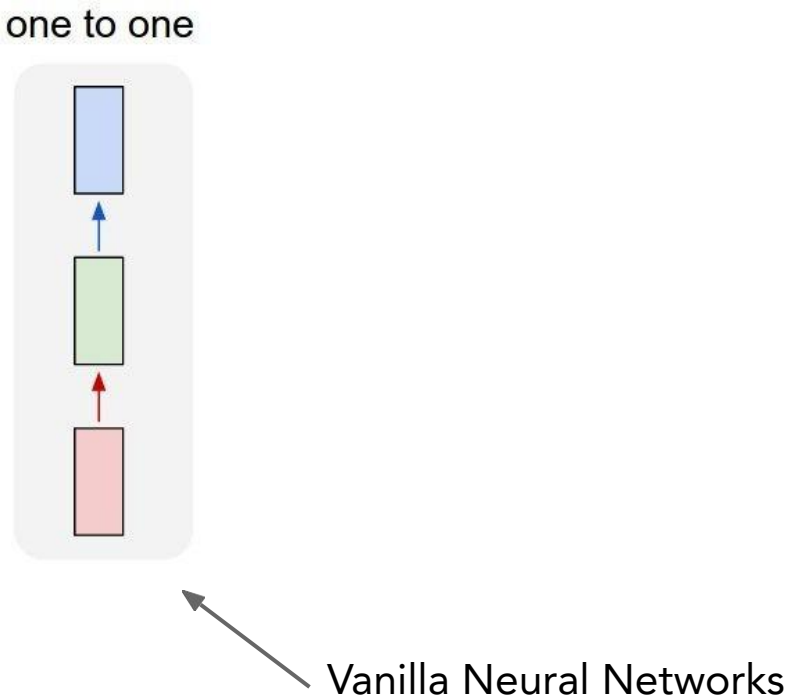
It is quite fundamental and was mentioned in rudimentary form in many papers:

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*

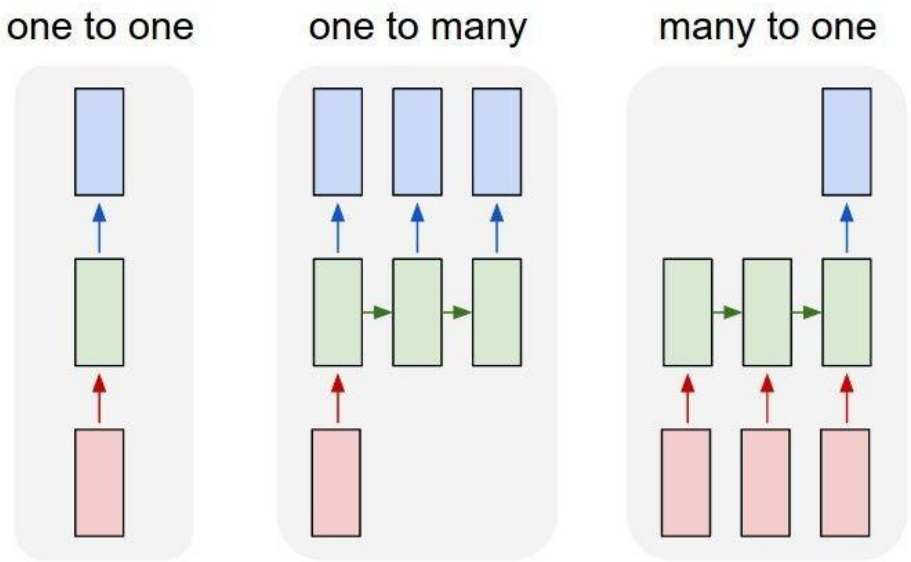
Many interesting theoretical results:

- 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



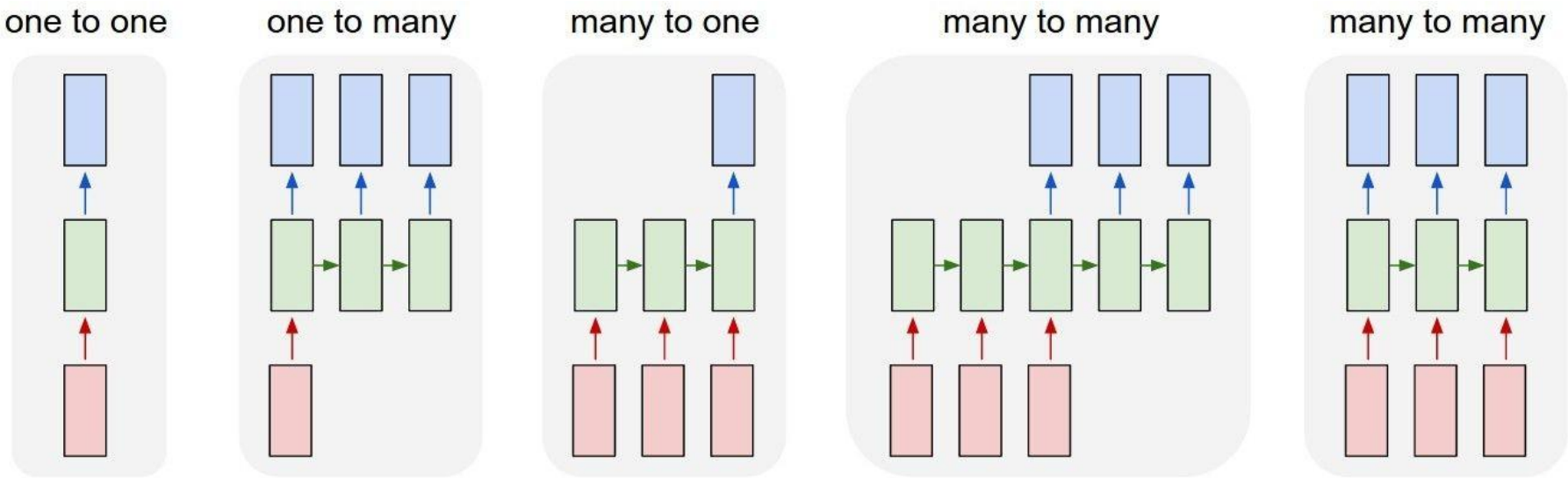
Recurrent Neural Networks: Process Sequences



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



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

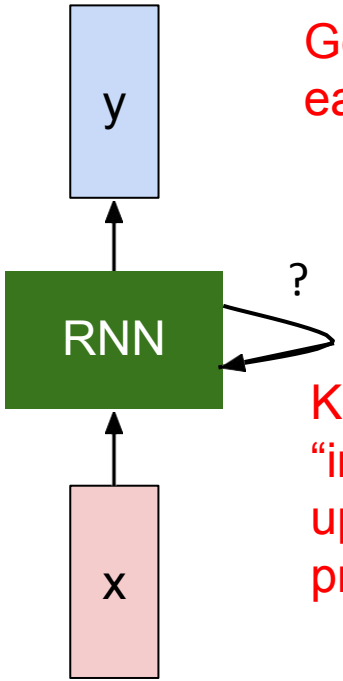
e.g. Sentence
Parts-of-speech tagging

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.

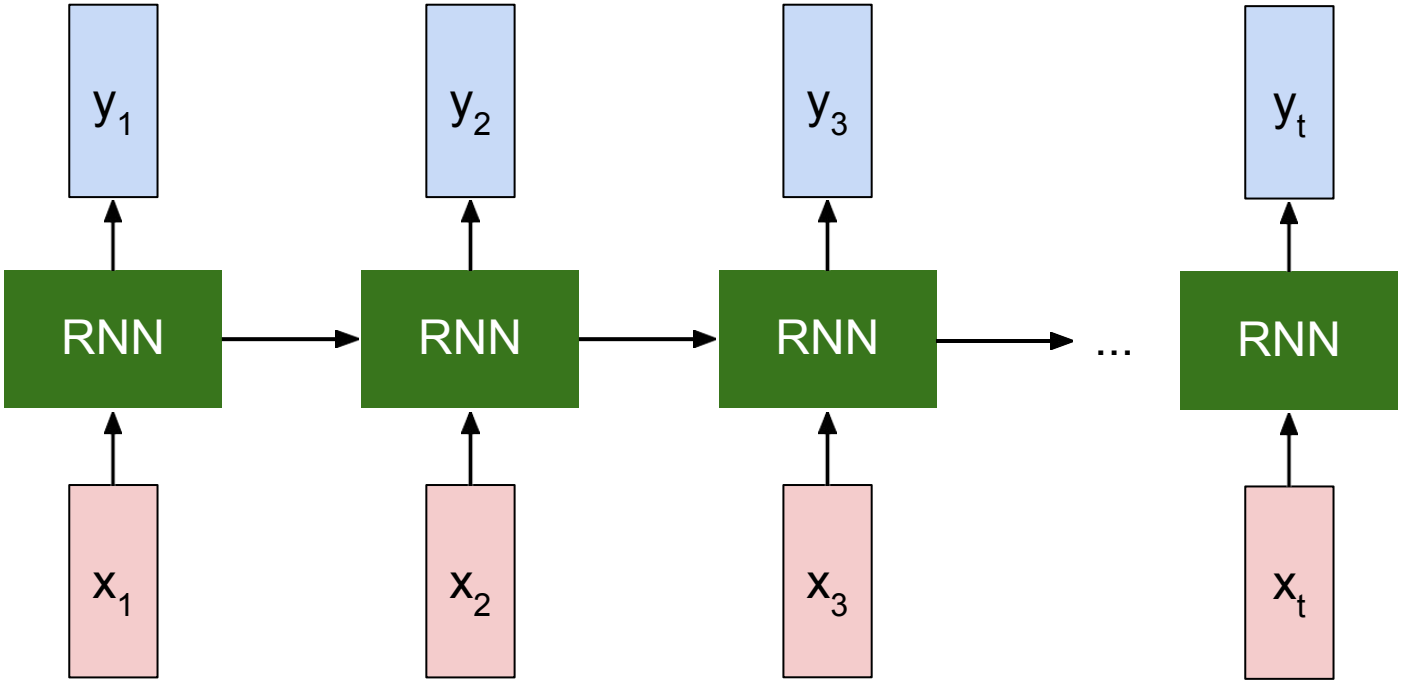
RNN Output Generation



Generally a vector at each (or some) timesteps

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

Unrolled RNN



RNN hidden state update

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

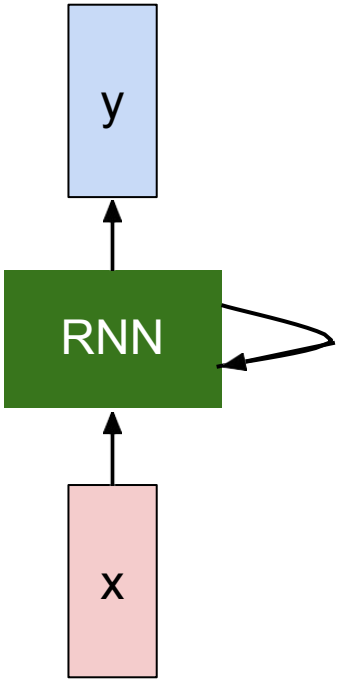
$$h_t = f_W(h_{t-1}, x_t)$$

new state

some function with parameters W

old state

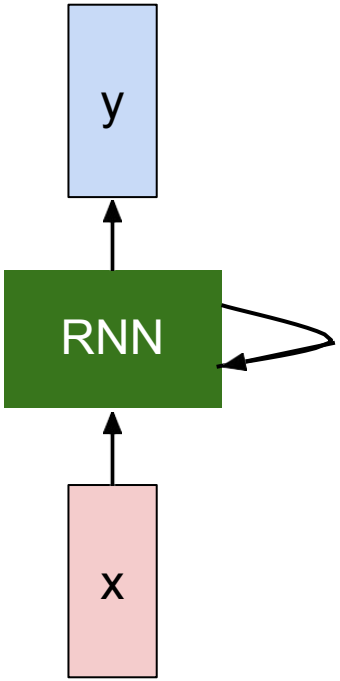
input vector at some time step



Recurrent Neural Network

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

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



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

RNN output generation

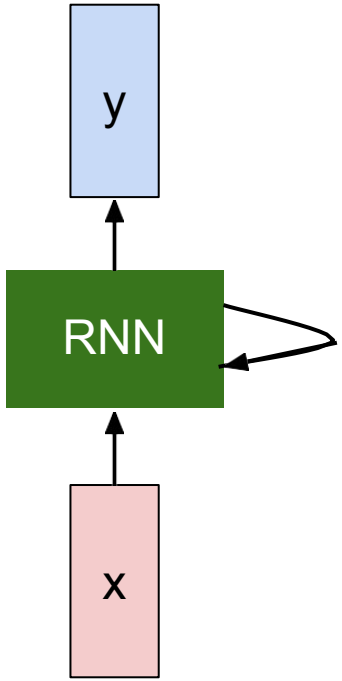
We can process a sequence of vectors **x** by applying a **recurrence formula** at every time step:

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

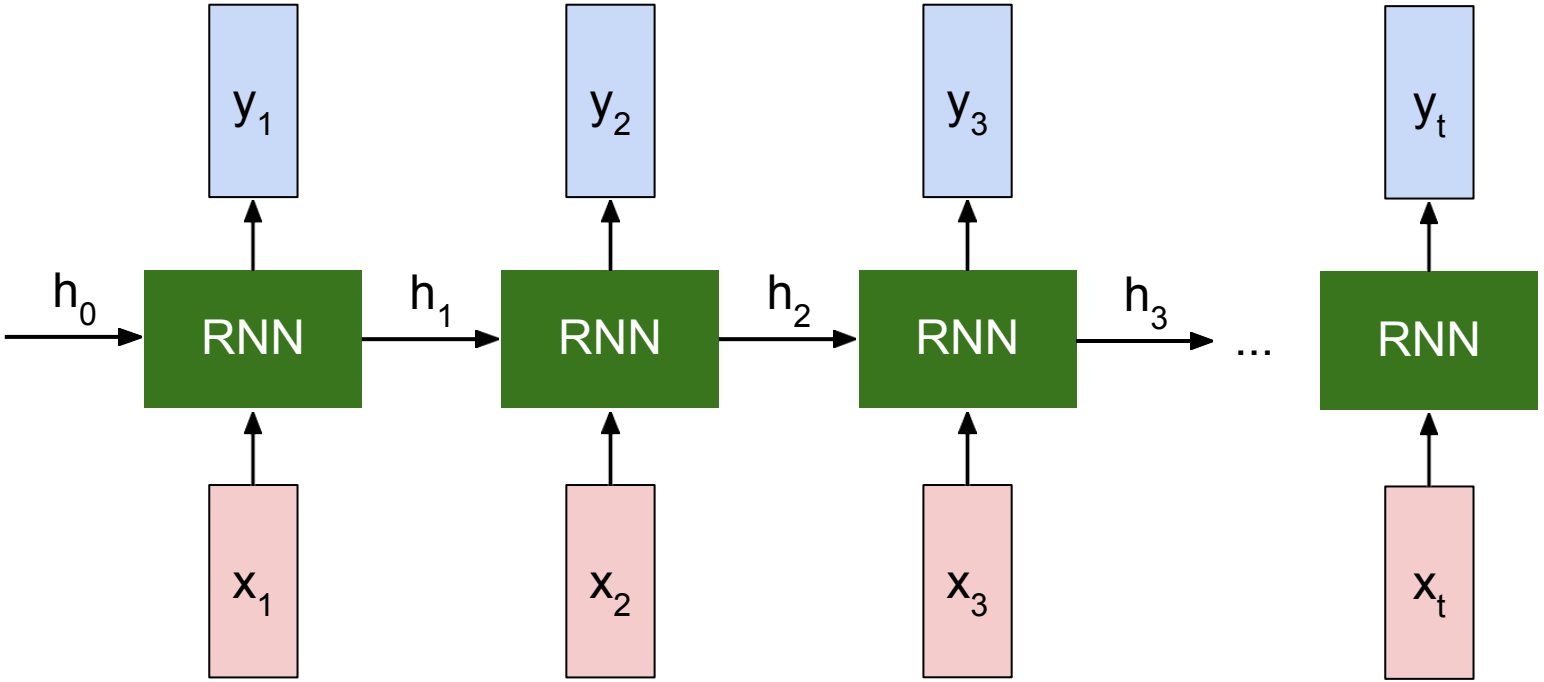
output

new state

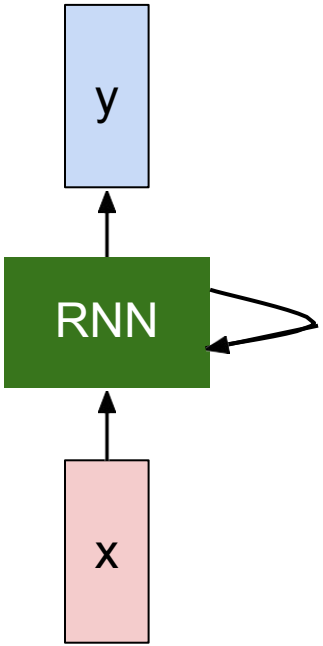
another function with parameters W_o



Recurrent Neural Network



(Vanilla) Recurrent Neural Network



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

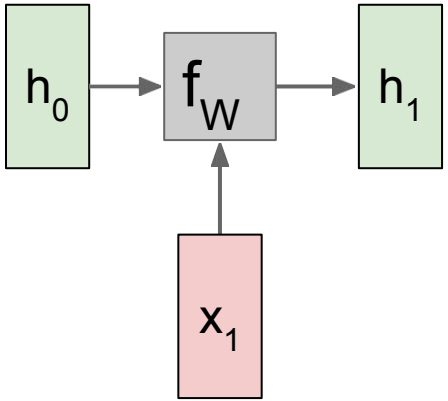
$$h_t = f_W(h_{t-1}, x_t)$$

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

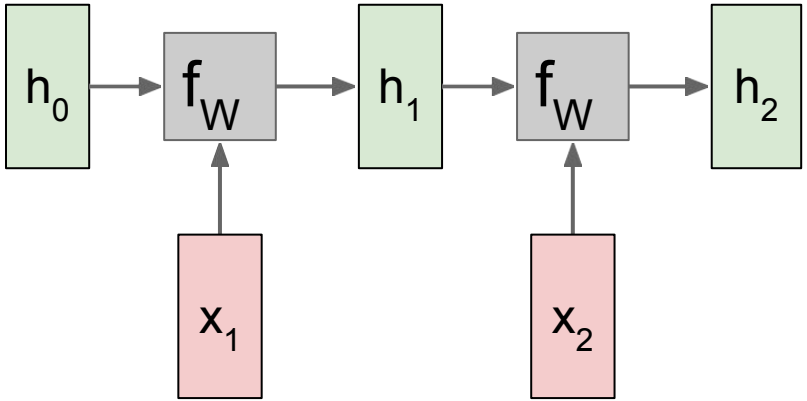
$$y_t = W_{hy}h_t$$

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

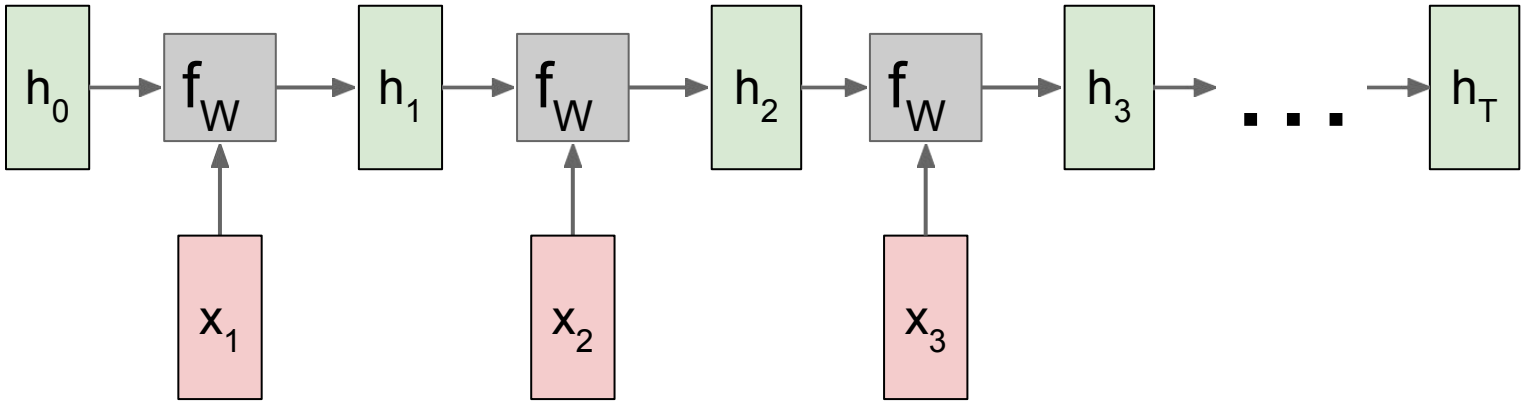
RNN: Computational Graph



RNN: Computational Graph

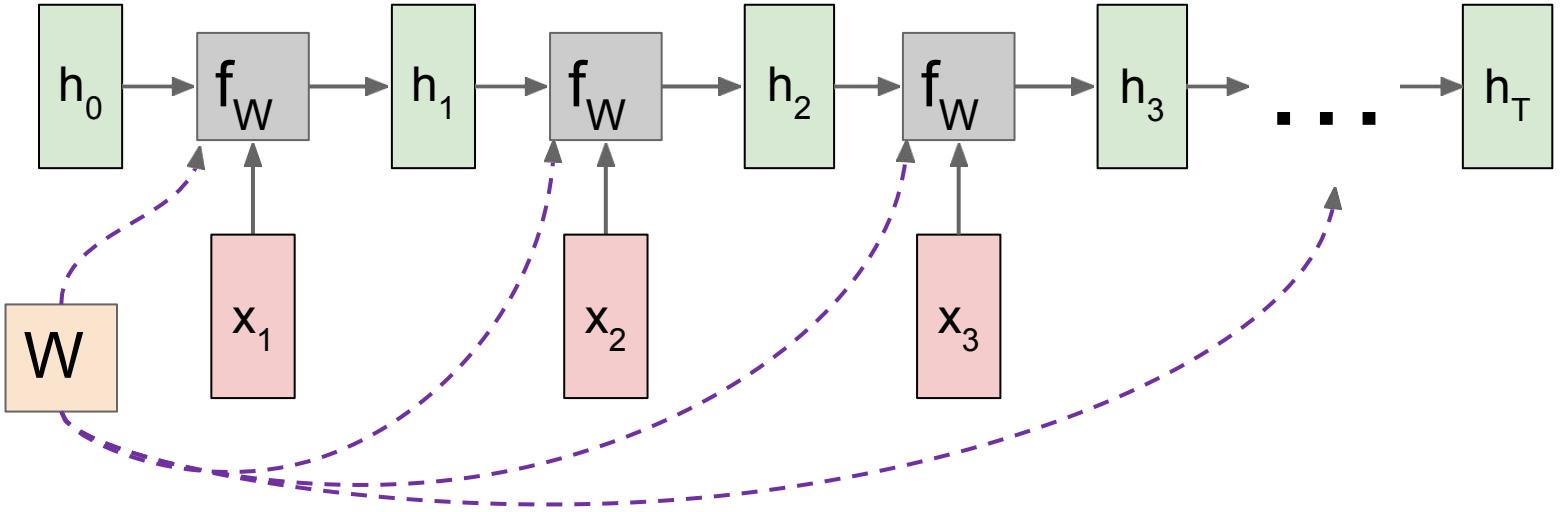


RNN: Computational Graph

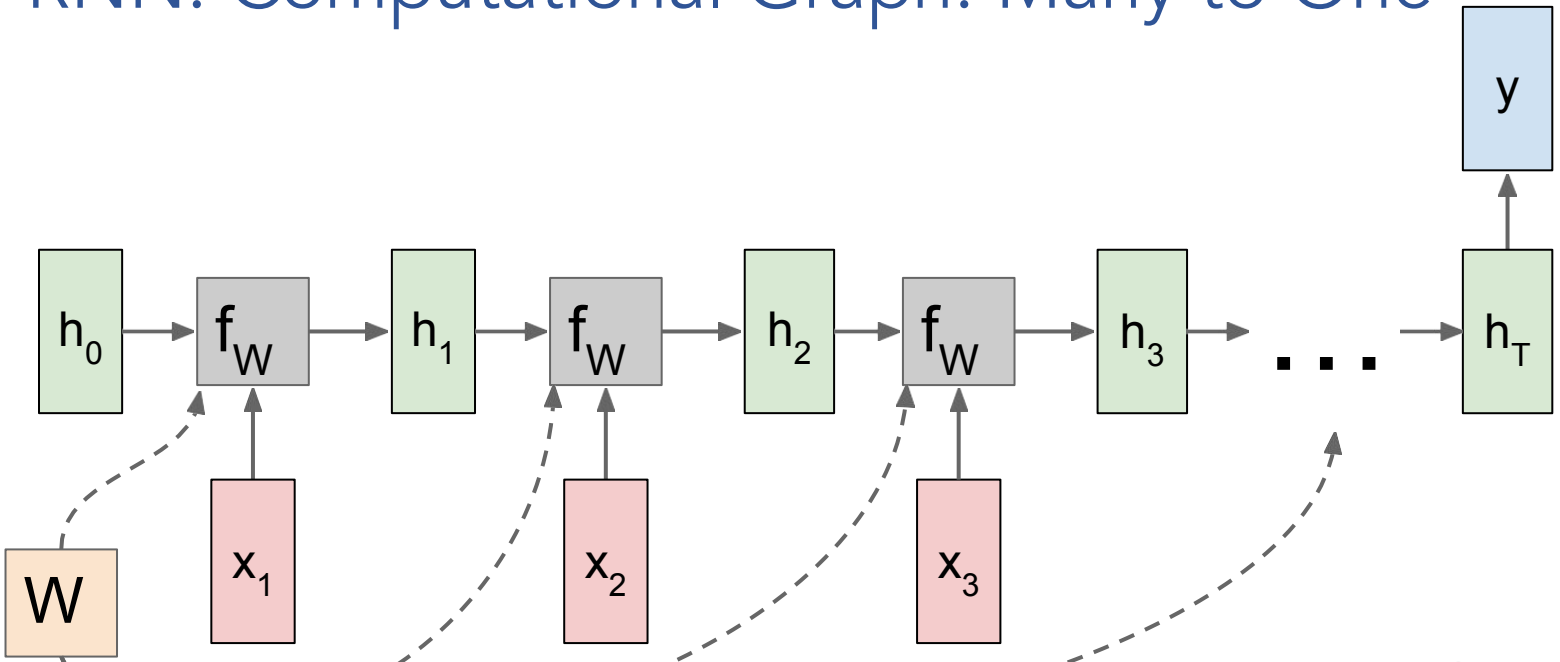


RNN: Computational Graph

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



RNN: Computational Graph: Many to One

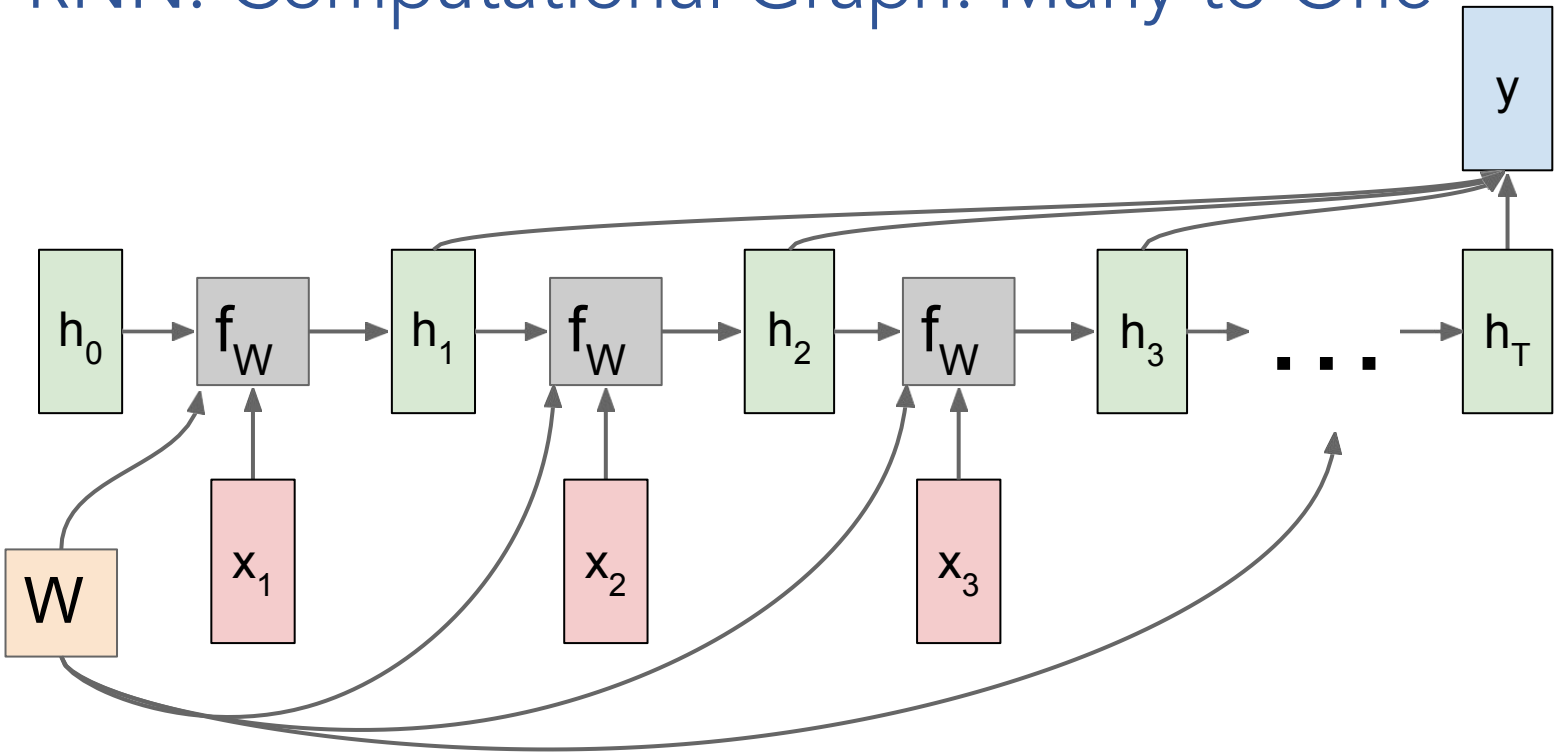


Sentiment Classification.

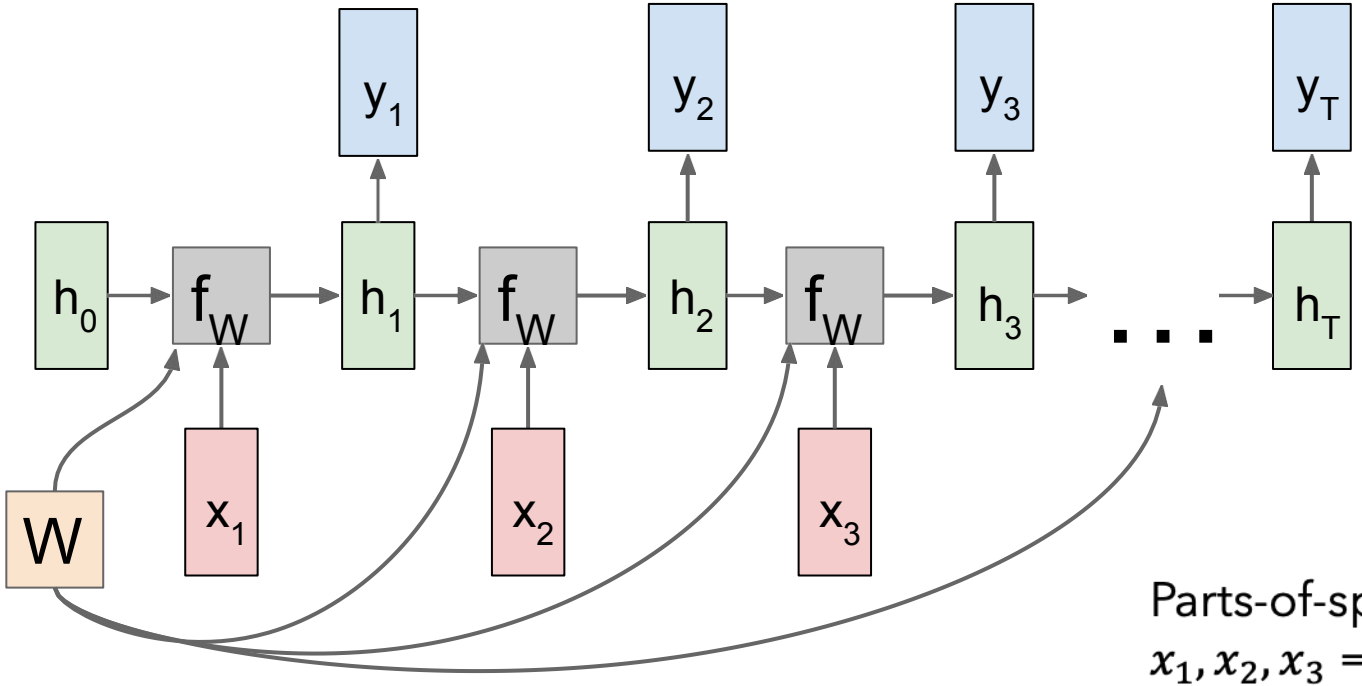
x_1, x_2, x_3 = {I, loved, the, movie}

y_1 = positive.

RNN: Computational Graph: Many to One

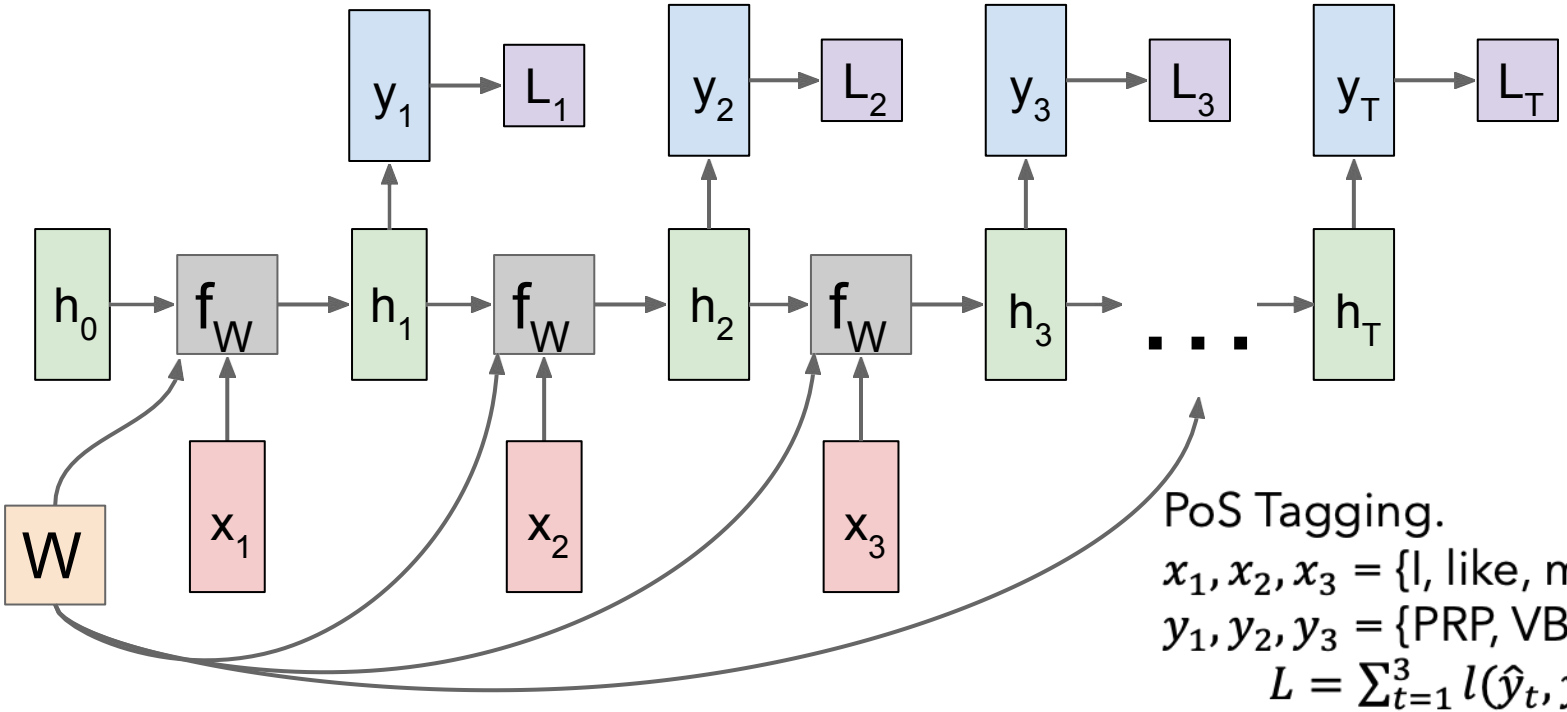


RNN: Computational Graph: Many to Many



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

RNN: Computational Graph: Many to Many



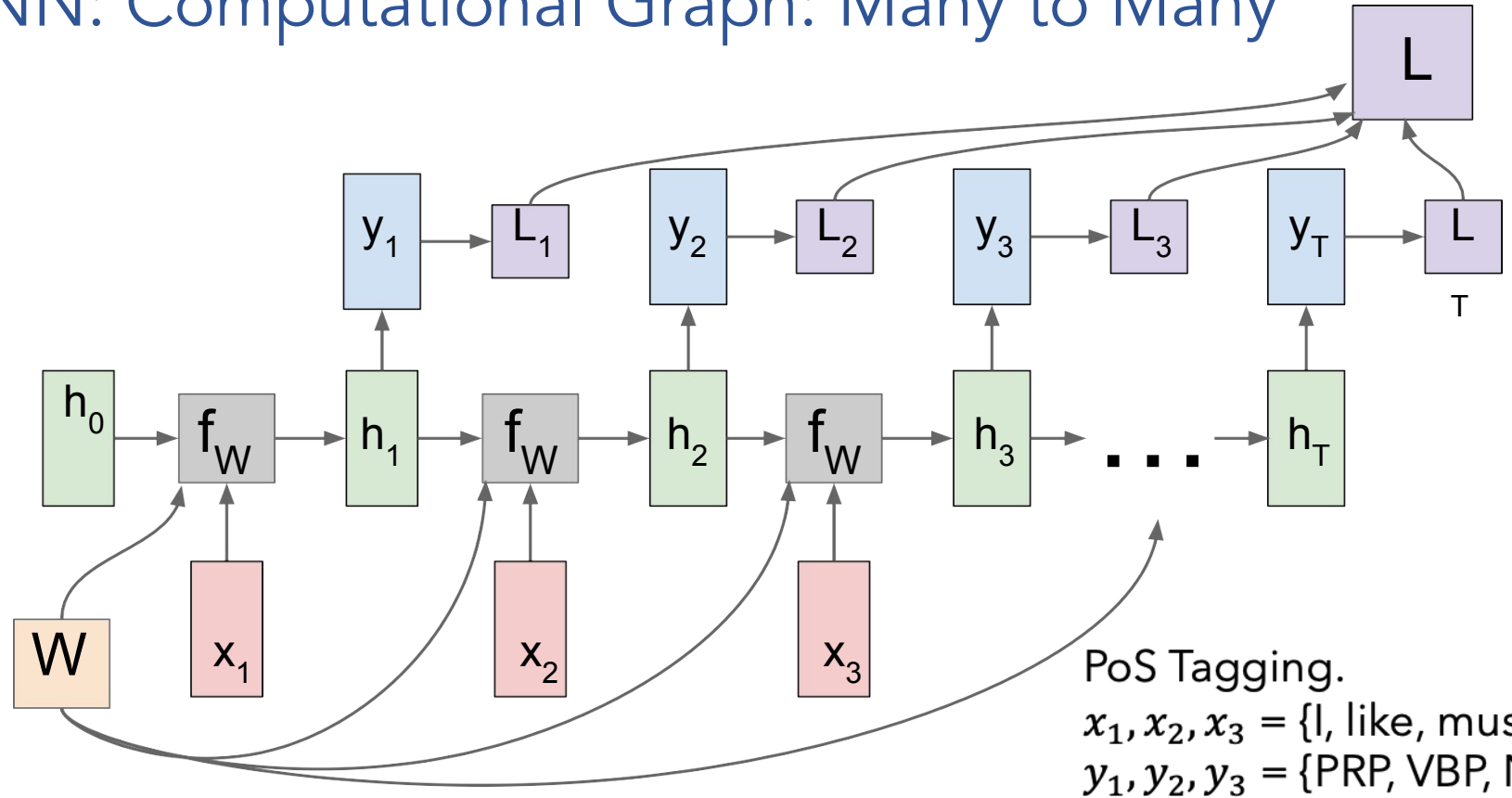
PoS Tagging.

x_1, x_2, x_3 = {I, like, music}.

y_1, y_2, y_3 = {PRP, VBP, NN}.

$$L = \sum_{t=1}^{3} l(\hat{y}_t, y_t)$$

RNN: Computational Graph: Many to Many



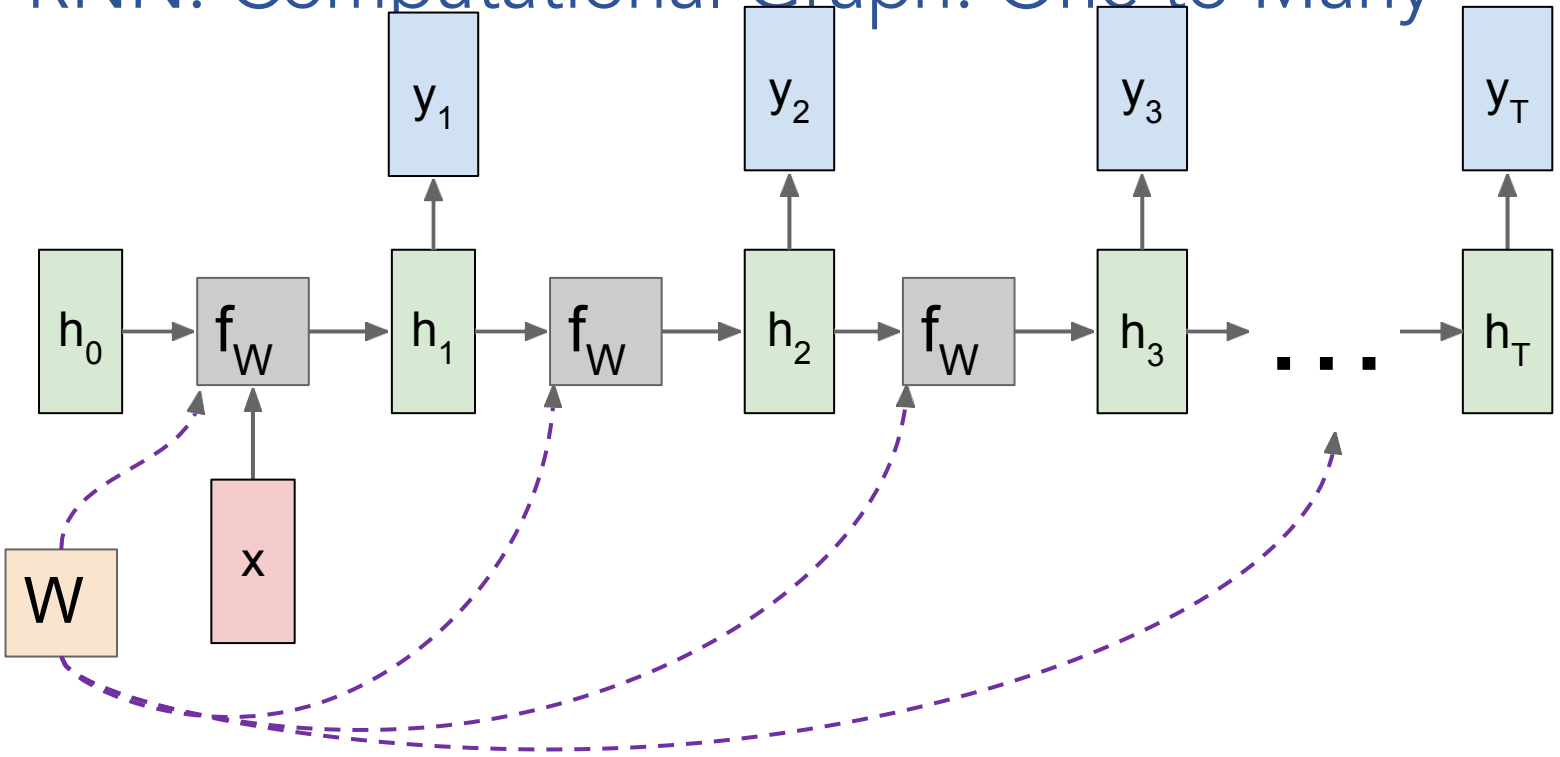
PoS Tagging.

x_1, x_2, x_3 = {I, like, music}.

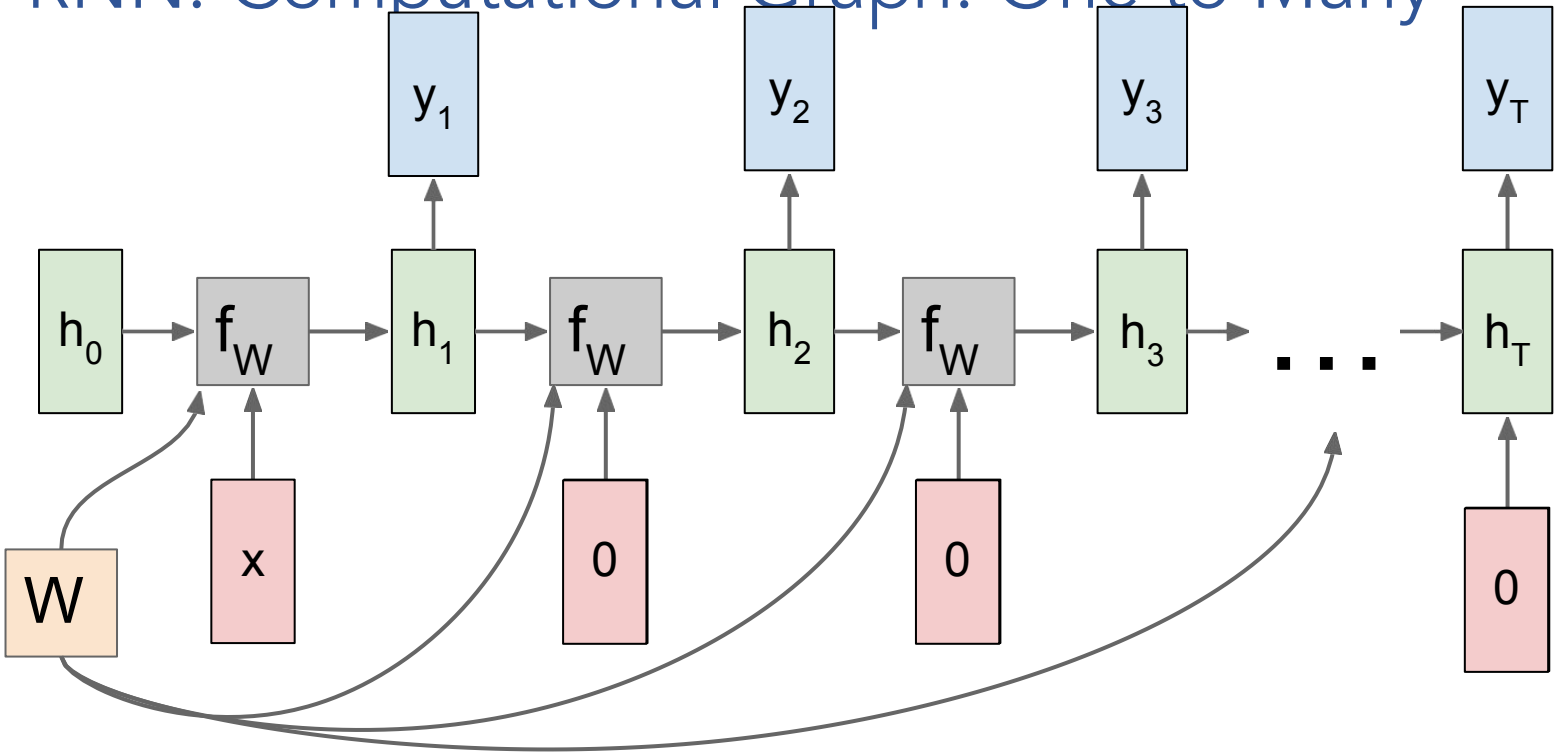
y_1, y_2, y_3 = {PRP, VBP, NN}.

$$L = \sum_{t=1}^{3} l(\hat{y}_t, y_t)$$

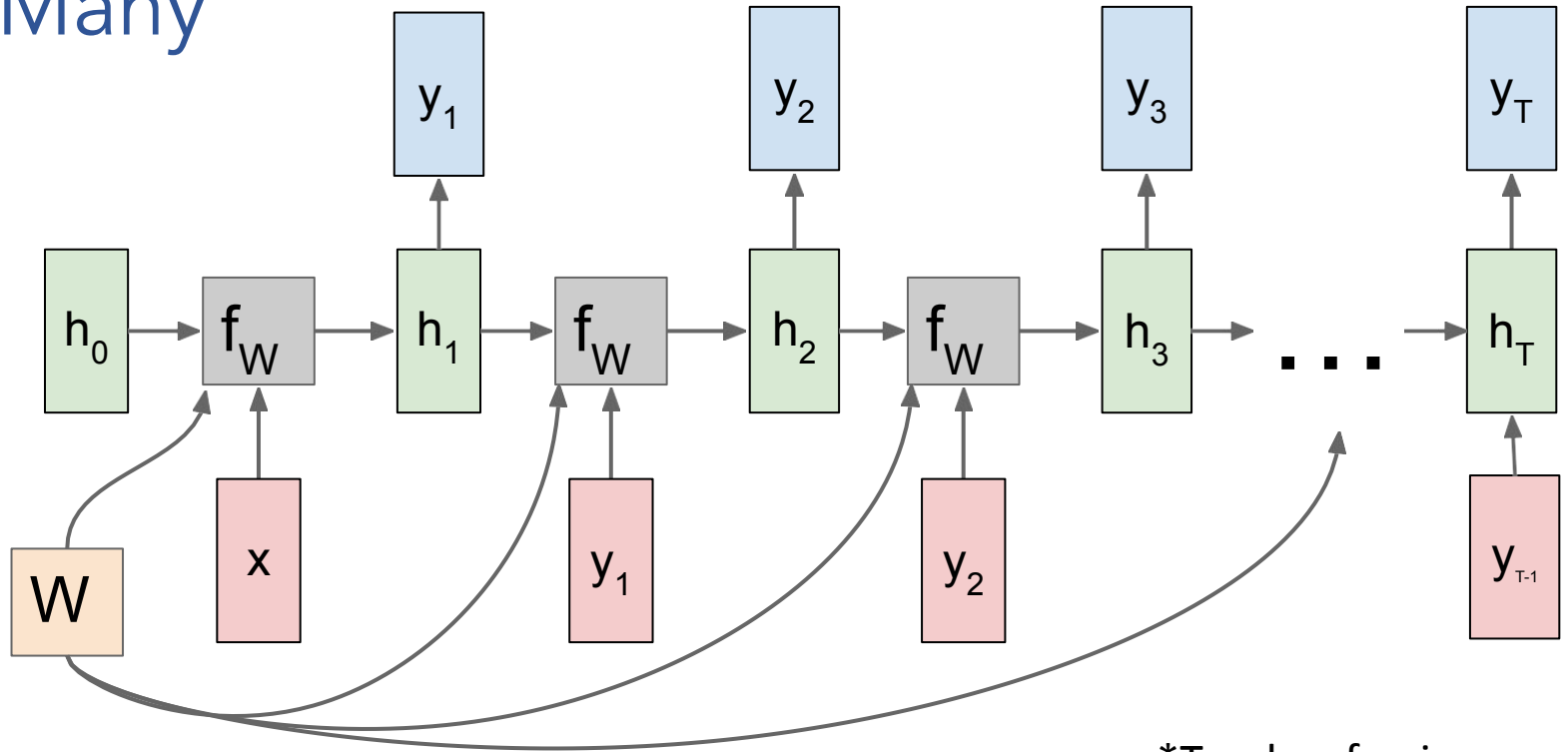
RNN: Computational Graph: One to Many



RNN: Computational Graph: One to Many



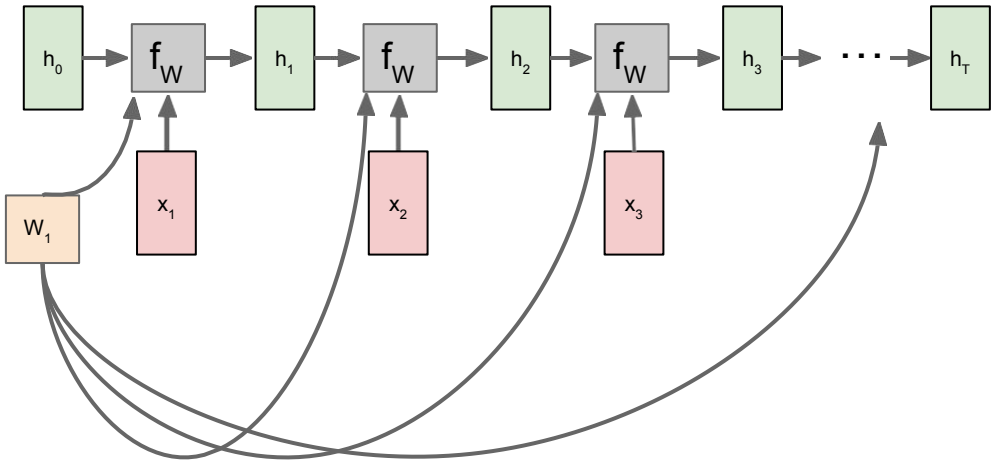
RNN: Computational Graph: One to Many



*Teacher-forcing
– putting the groundtruth

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

Many to one: Encode input sequence in a single vector

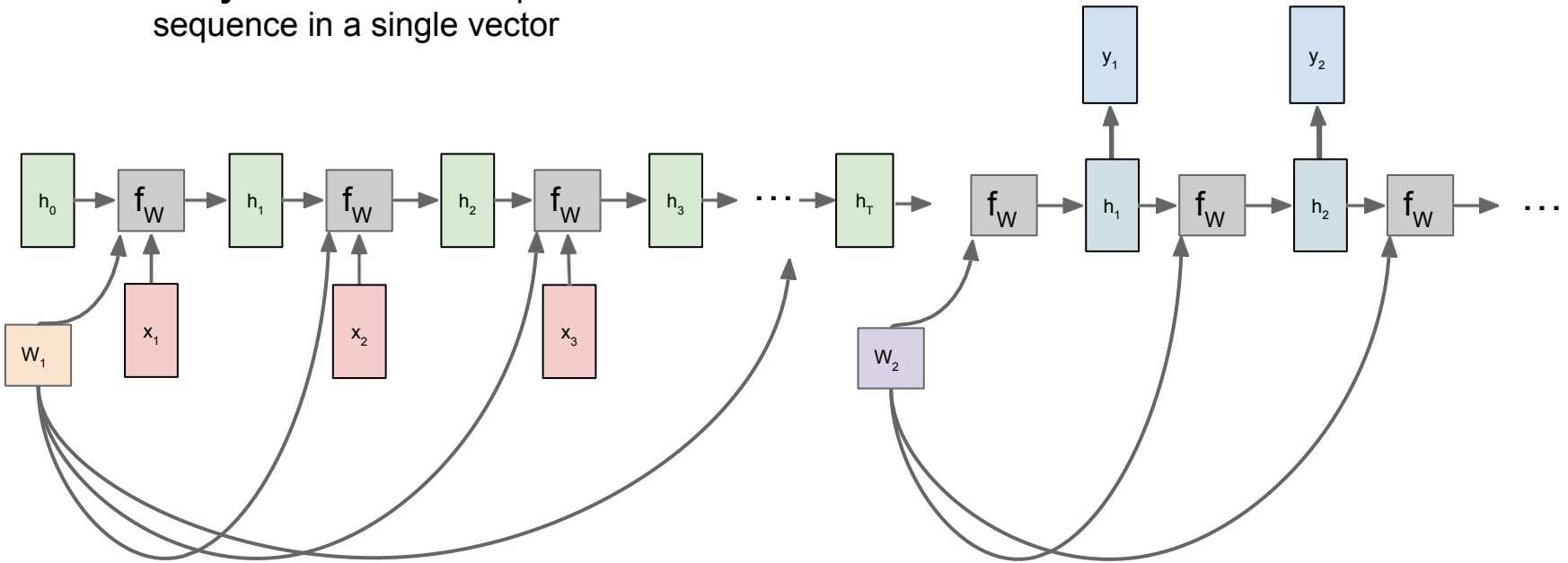


Sutskever et al, “Sequence to Sequence Learning with Neural Networks”, NIPS 2014

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



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

Forward Pass Example: RNN Character Language Model

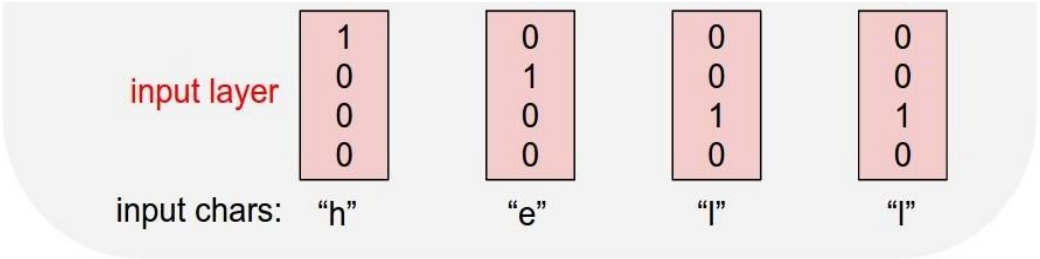
Example: Character-level Language Model

Vocabulary:
[h,e,l,o]

Example training sequence: “hello”

Task: Predict the next character

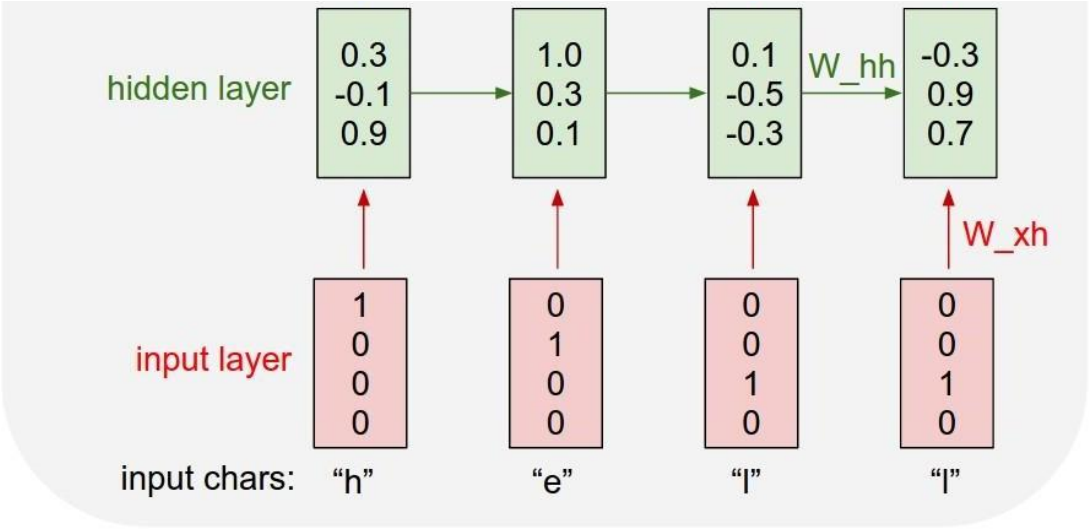
- Why? At test time, we can use this ability to generate words, sentences, even paragraphs



Example: Character-level Language Model

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

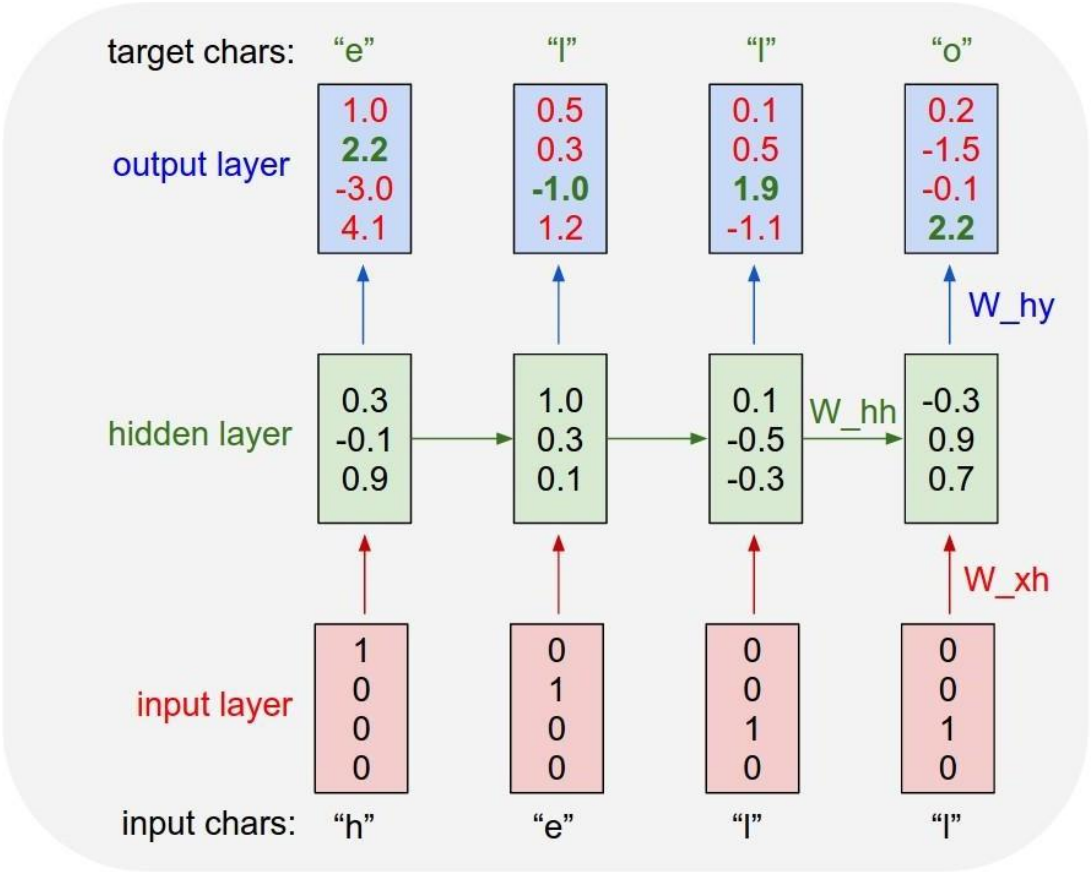
- Vocabulary: [h,e,l,o]
- Example training sequence: "hello"
- Task: Predict the next character
  - Why? At test time, we can use this ability to generate words, sentences, even paragraphs



Example: Character-level Language Model

- Vocabulary: [h,e,l,o]

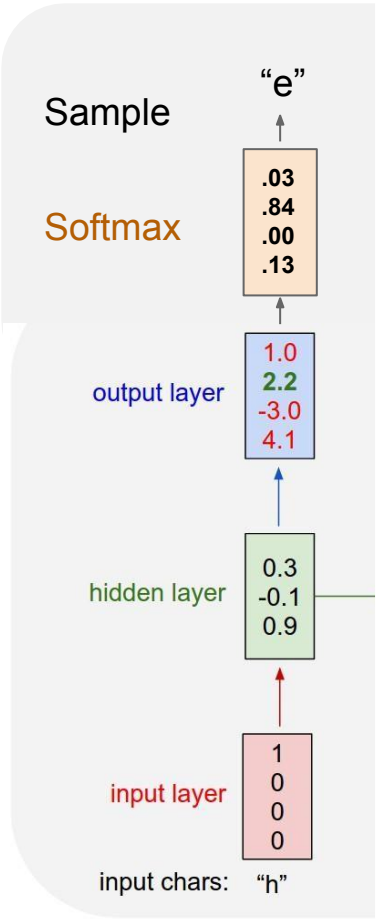
- Example training sequence: “hello”



Example: Character-level Language Model Sampling

Vocabulary:
[h,e,l,o]

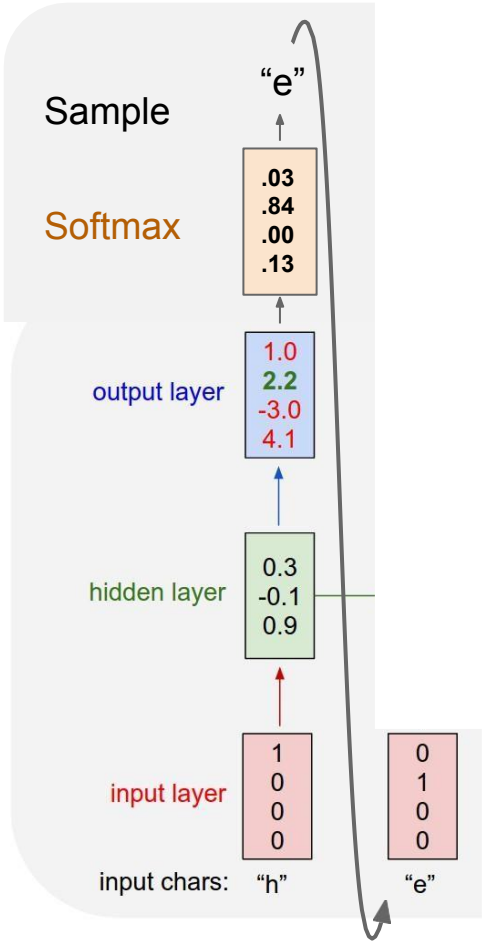
At test-time sample characters one at a time, feed back to model



Example: Character-level Language Model Sampling

Vocabulary:
[h,e,l,o]

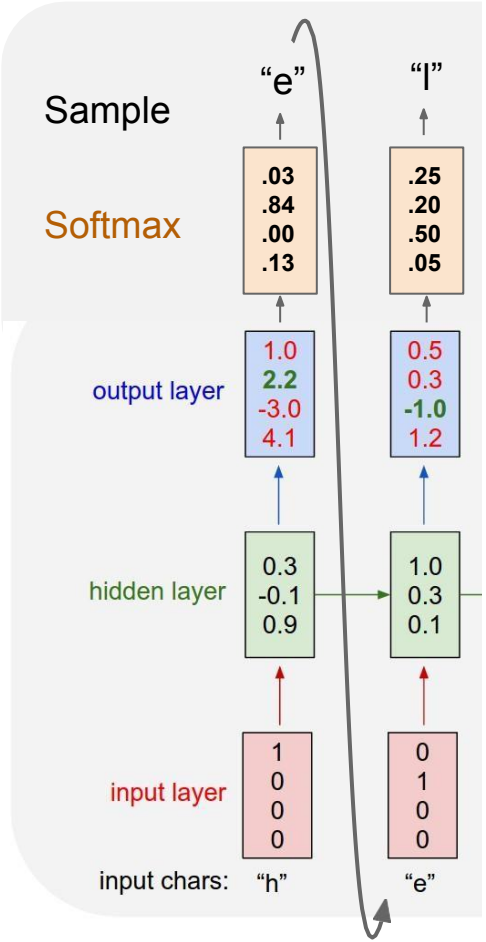
At test-time sample characters one at a time, feed back to model



Example: Character-level Language Model Sampling

Vocabulary:
[h,e,l,o]

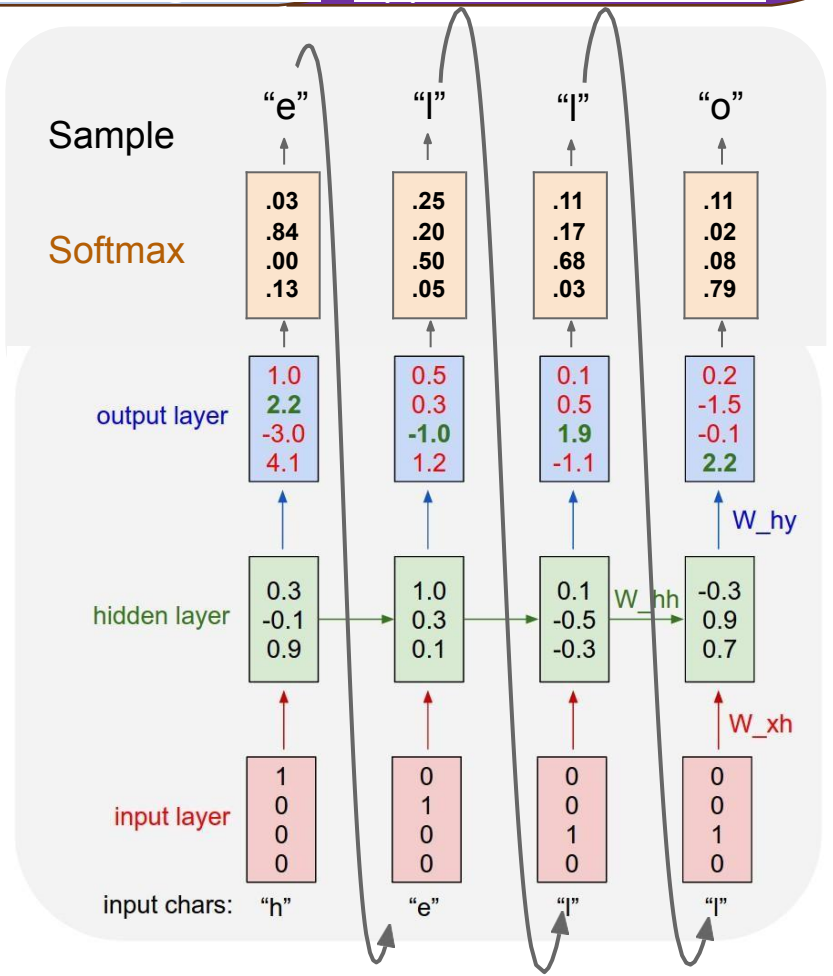
At test-time sample characters one at a time, feed back to model



Example: Character-level Language Model Sampling

Vocabulary:
[h,e,l,o]

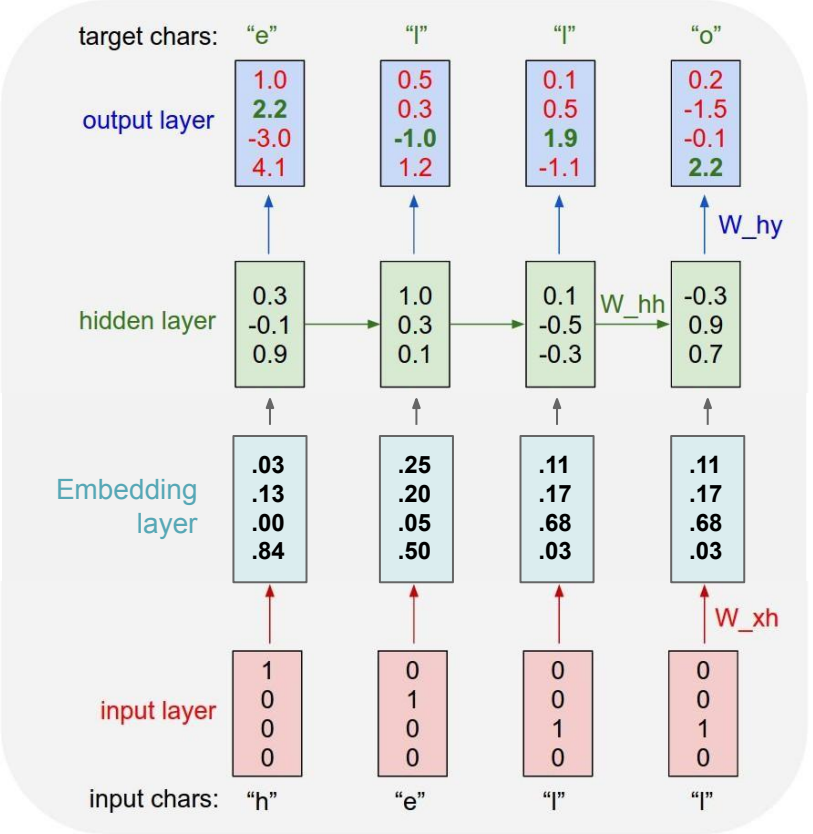
At test-time sample characters one at a time, feed back to model



Example: Character-level Language Model Sampling

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

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.

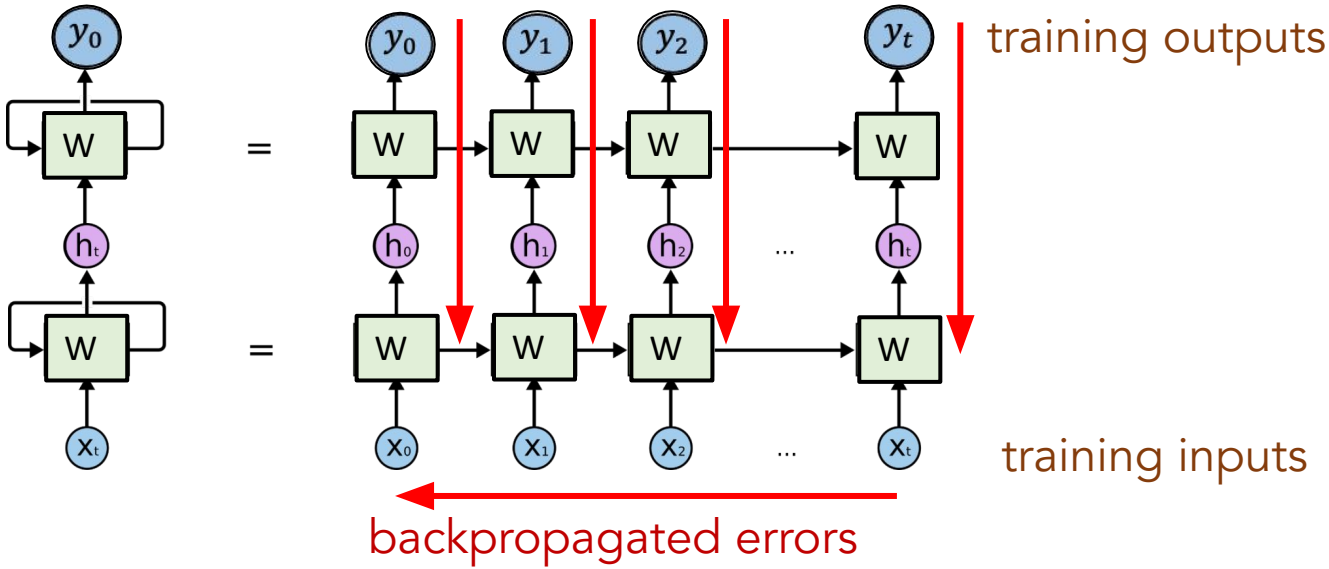


Backpropagation through Time

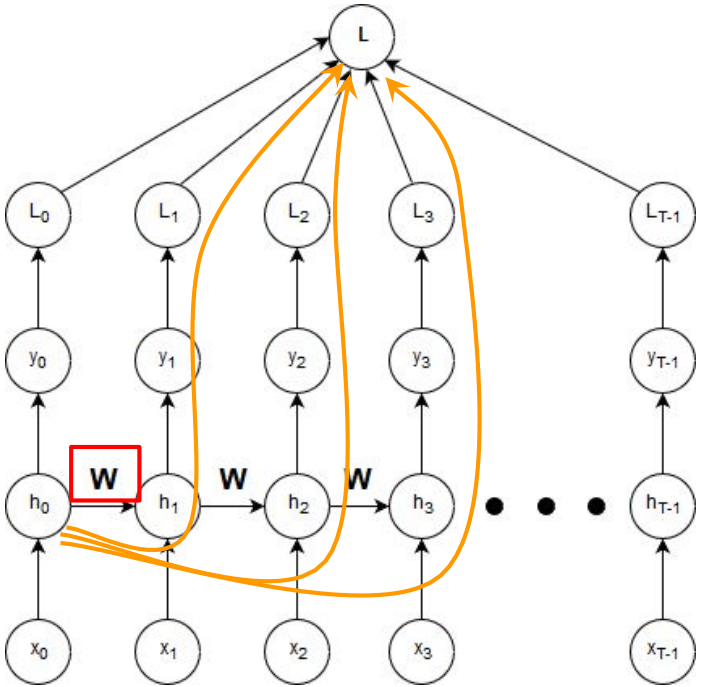
Training RNN's

RNNs can be trained using "backpropagation through time."

- applying normal backprop to the unrolled network.



Backpropagation Through Time (BPTT)

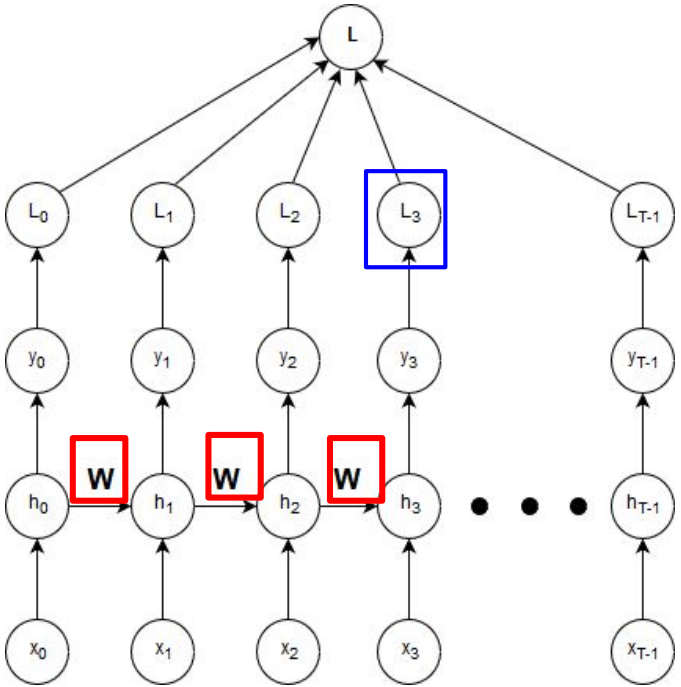


Update the weight matrix:

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

- 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:

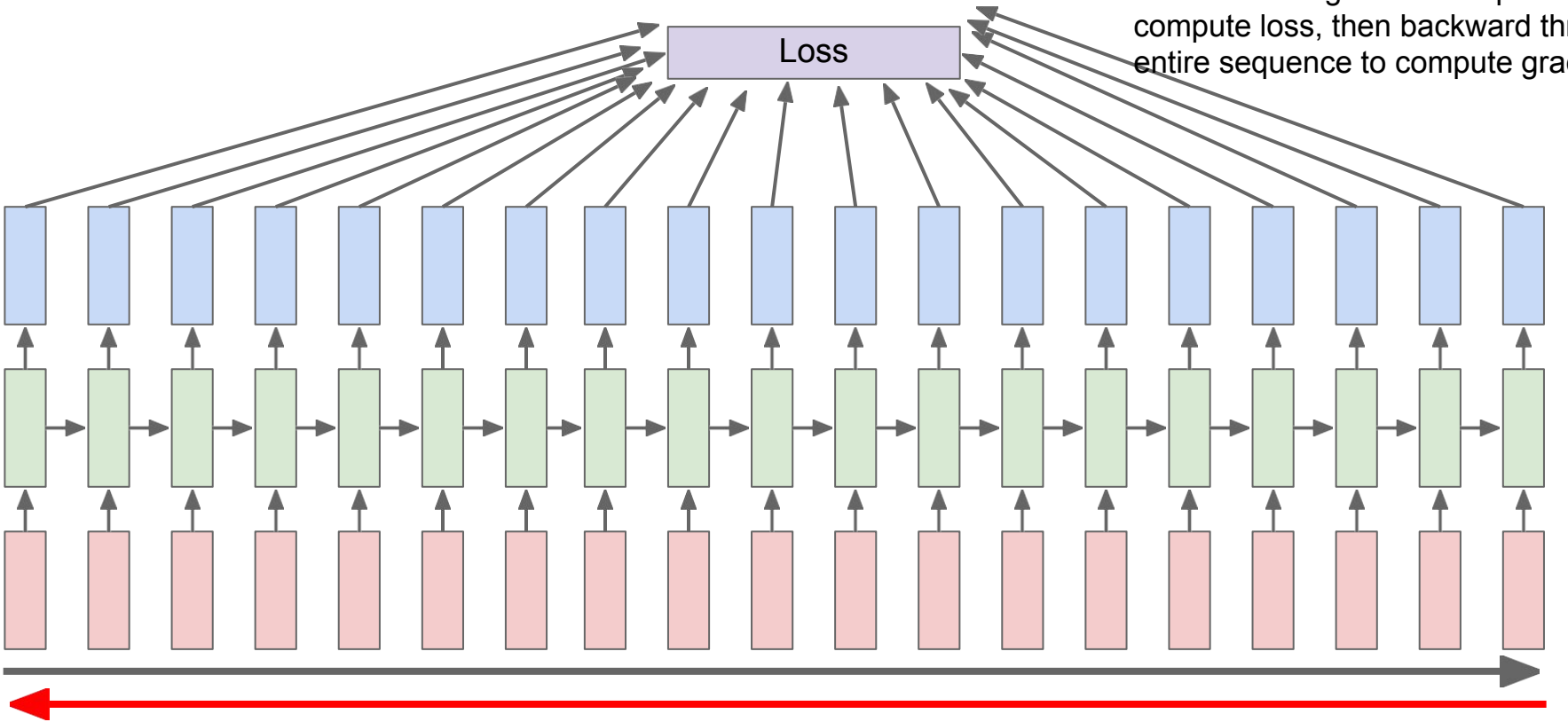
Each L_j depends on the weight matrices *before it*

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

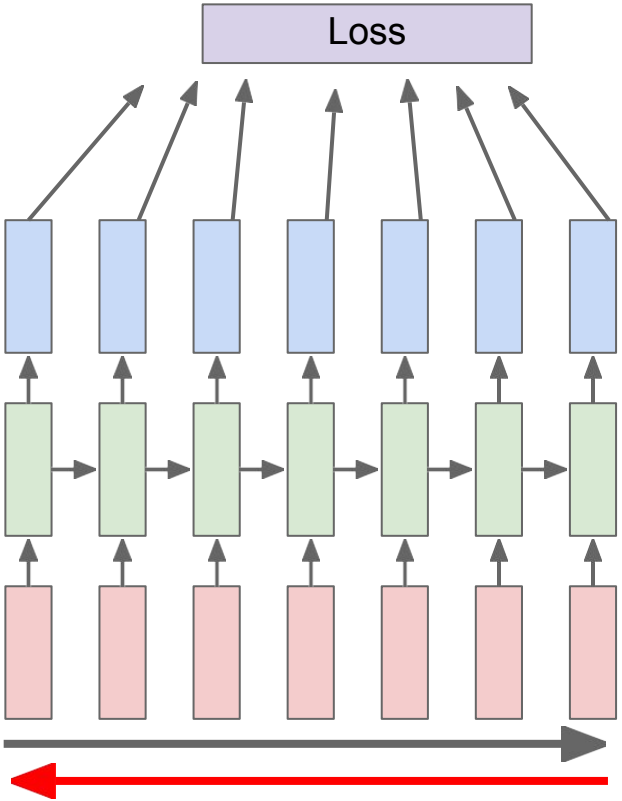
L_j depends on all h_k before it.

Backpropagation through time

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

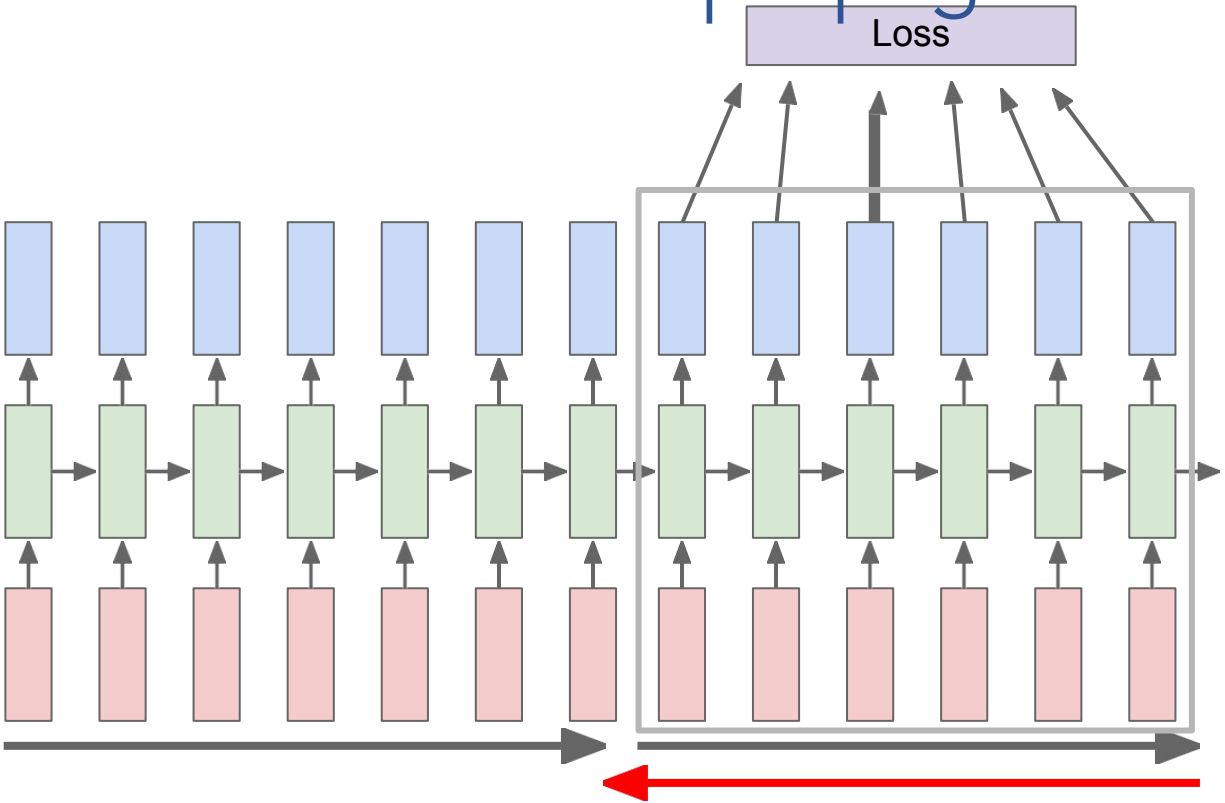


Truncated Backpropagation through time



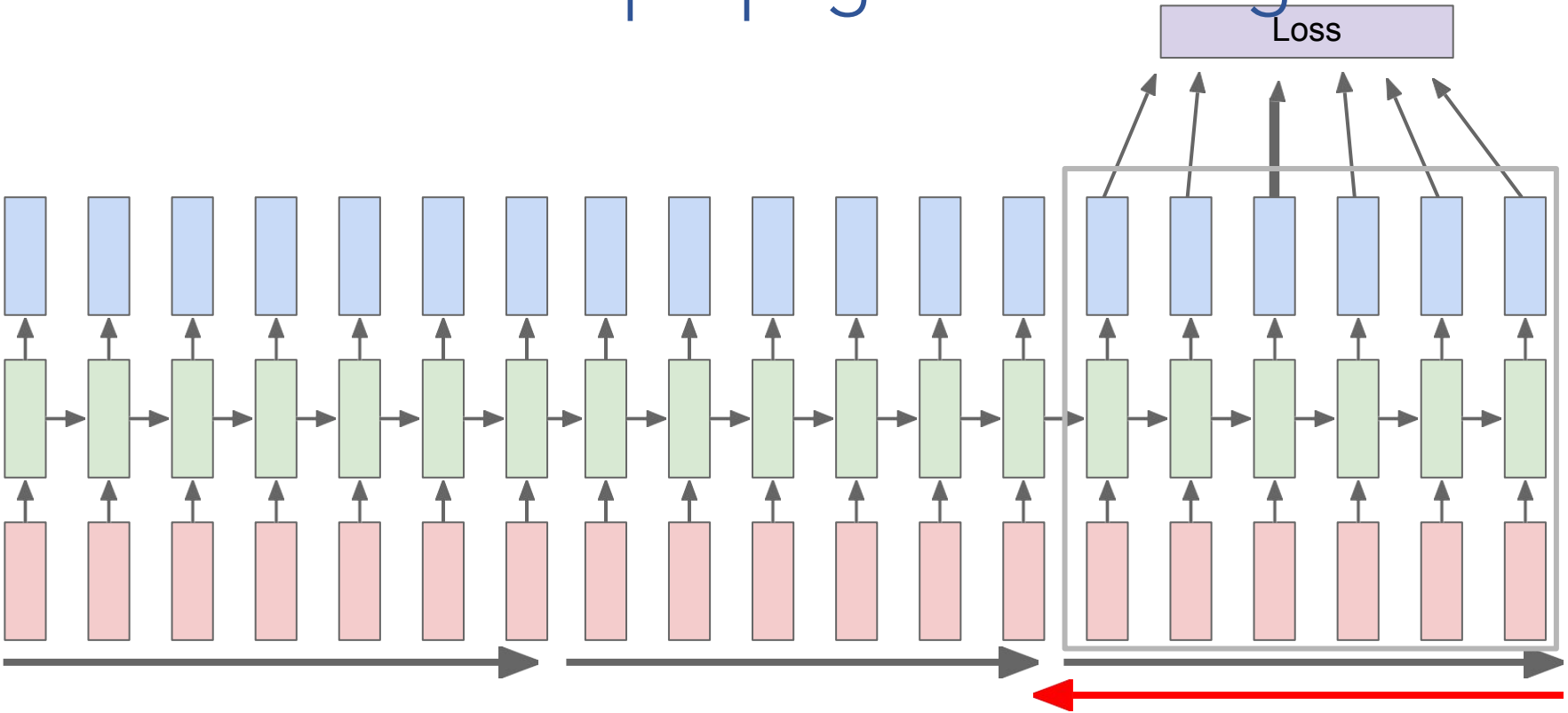
Run forward and backward through chunks of the sequence instead of whole sequence

Truncated Backpropagation through time



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

Truncated Backpropagation through time



RNN Applications and Issues

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

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

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

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

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

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

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

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

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

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

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

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

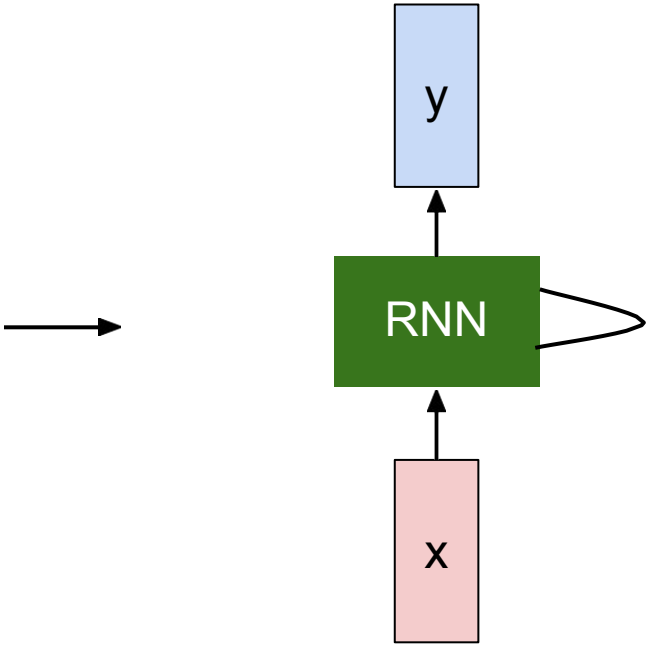
(https://gist.github.com/karpathy/d4dee 566867f8291f086)

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 tklrgd t o idoe ns,smtt   h ne etie h,hregtrs nigtike,aoaenns lng
```

train more

```
"Tmont thithey" fomesscerliund
Keushey. Thom here
sheulke, anmerenith ol sivh I lalterthend Bleipile shuwy fil on aseterlome
coaniogennc Phe lism thond hon at. MeiDimorotion in ther thize."
```

train more

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

train more

```
"Why do what that day," replied Natasha, and wishing to himself the fact the
princess, Princess Mary was easier, fed in had oftened him.
Pierre aking his soul came to the packs and drove up his father-in-law women.
```

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

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

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

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

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

VIOLA:
I'll drink it.
```

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

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

Image Captioning

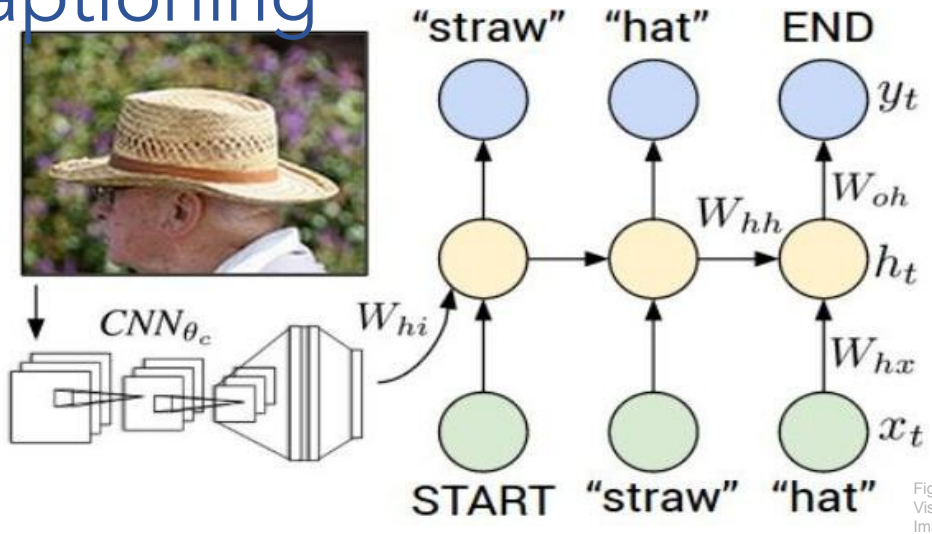
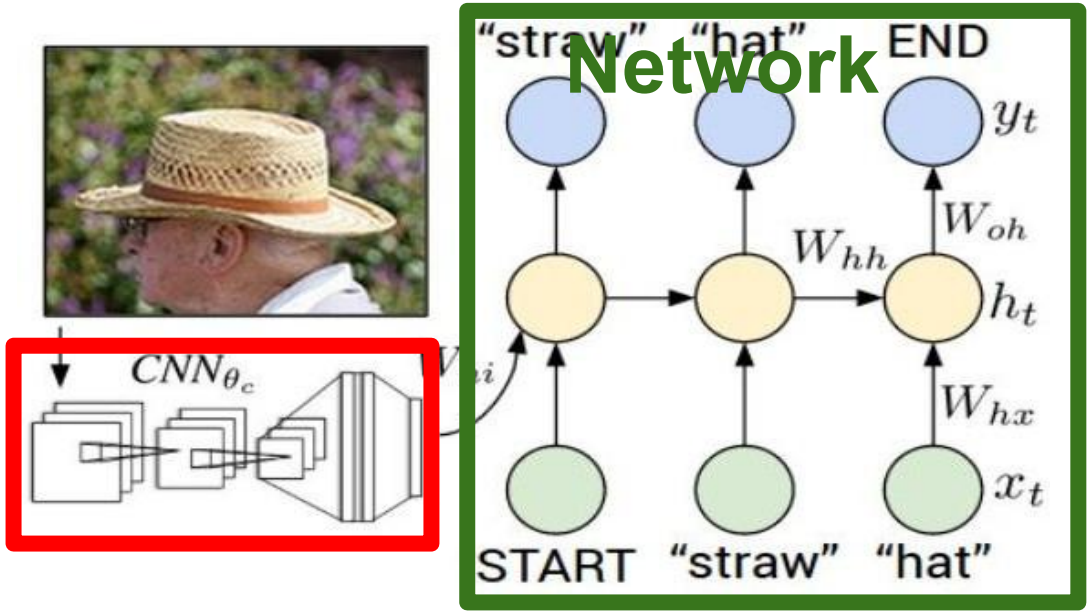


Figure from Karpathy et a, "Deep Visual-Semantic Alignments for Generating Image Descriptions", CVPR 2015; figure copyright IEEE, 2015. Reproduced for educational purposes.

Explain Images with Multimodal Recurrent Neural Networks, Mao et al.
Deep Visual-Semantic Alignments for Generating Image Descriptions, Karpathy and Fei-Fei
Show and Tell: A Neural Image Caption Generator, Vinyals et al.
Long-term Recurrent Convolutional Networks for Visual Recognition and Description, Donahue et al.
Learning a Recurrent Visual Representation for Image Caption Generation, Chen and Zitnick

Recurrent Neural Network



Convolutional Neural Network

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

Image Captioning: Example Results

A cat sitting on a suitcase on the floor

A cat is sitting on a tree branch

A dog is running in the grass with a frisbee

A white teddy bear sitting in the grass

Two people walking on the beach with surfboards

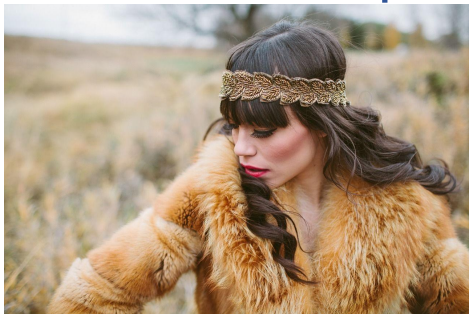
A tennis player in action on the court

Two giraffes standing in a grassy field

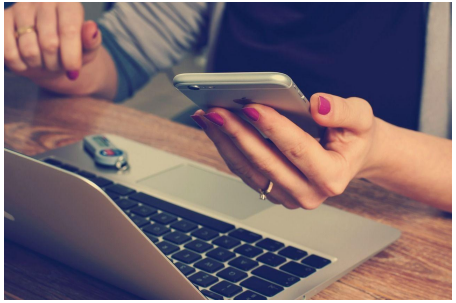
A man riding a dirt bike on a dirt track

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

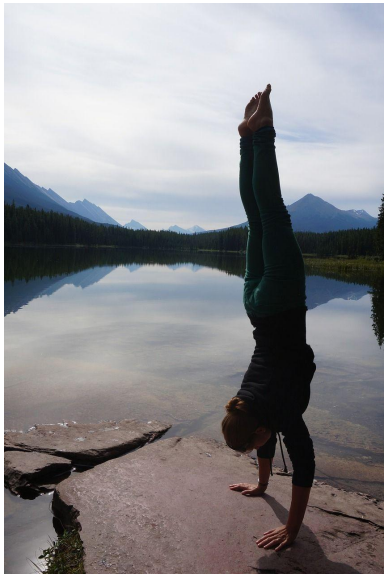
Image Captioning: Failure Cases



A woman is holding a cat in her hand



A person holding a computer mouse on a desk

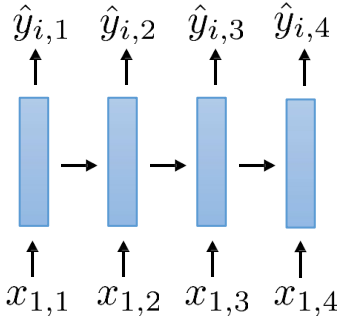


A woman standing on a beach holding a surfboard

A bird is perched on a tree branch

A man in a baseball uniform throwing a ball

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"

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

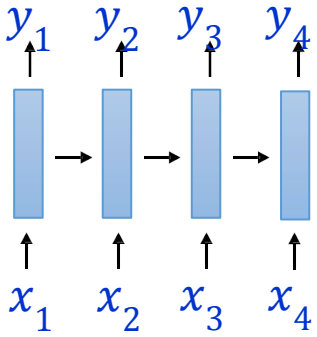
$$\frac{d\mathcal{L}}{dW^{(1)}} = J_1 J_2 J_3 \dots J_n \frac{d\mathcal{L}}{dz^{(n)}}$$

If we multiply many 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.

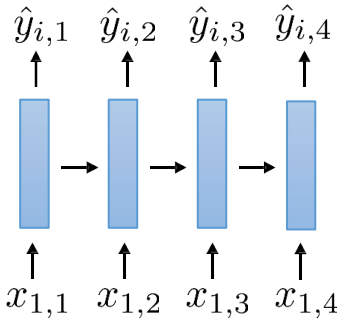
- vanishing gradients
- gradient signal from later steps never reaches the earlier steps.

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

Hard to fix

could fix with gradient clipping

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 $C^{(t-1)}$

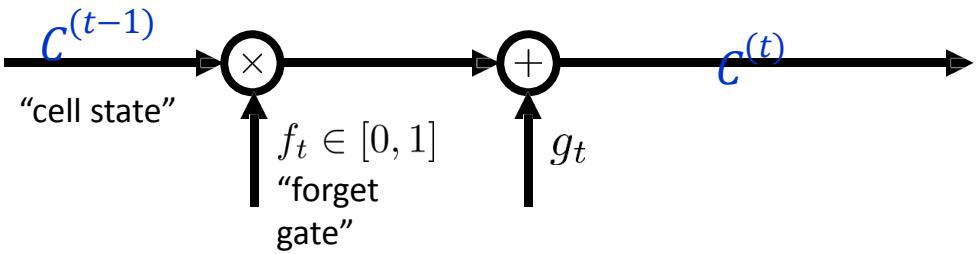
for each unit, we have a sort of "logic gate" that decides

- 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

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

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

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

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

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