CS 7643: Deep Learning

Topics:

- Recurrent Neural Networks (RNNs)
- BackProp Through Time (BPTT)
- Vanishing / Exploding Gradients

Dhruv Batra
Georgia Tech

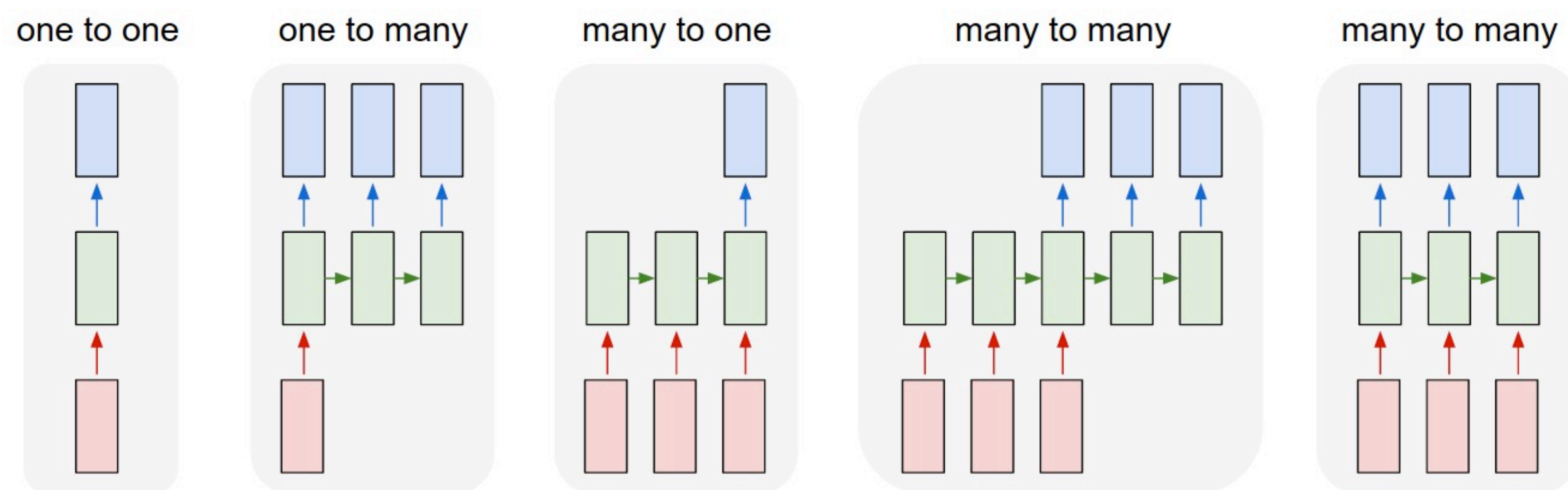
Administrativia

- HW3 + PS3 out
  - Due 10/28
  - Last PS and HW
  - Focus on projects after that

Plan for Today

- Model
  - Recurrent Neural Networks (RNNs)
- Learning
  - BackProp Through Time (BPTT)
  - Vanishing / Exploding Gradients

New Topic: RNNs



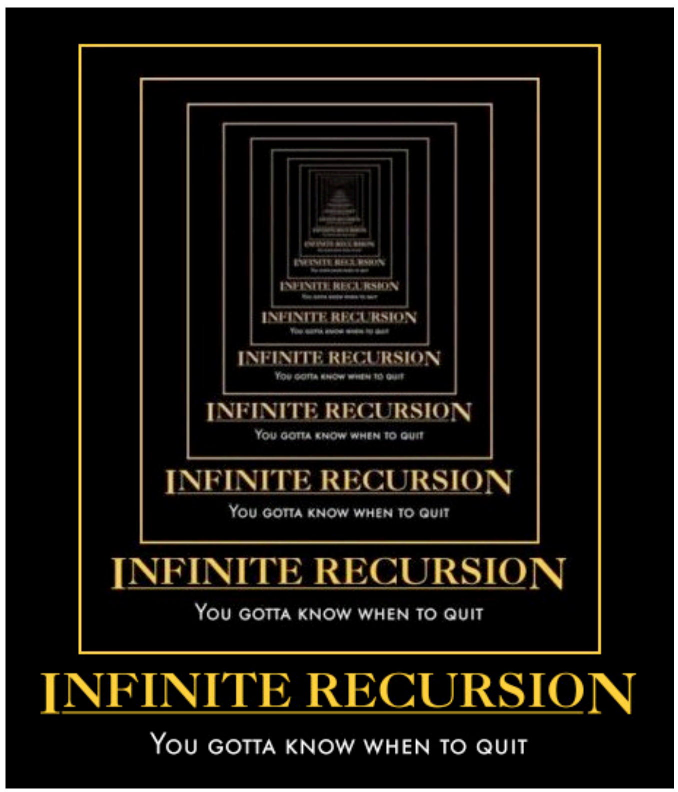


Image Credit: Andrej Karpathy

Synonyms

- Recurrent Neural Networks (RNNs)
- Recursive Neural Networks
  - General family; think graphs instead of chains
- Types:
  - "Vanilla" RNNs
  - Long Short Term Memory (LSTMs)
  - Gated Recurrent Units (GRUs)
  - …
- Algorithms
  - BackProp Through Time (BPTT) Recurrent
  - BackProp Through Structure (BPTS)

What's wrong with MLPs?

- Problem 1: Can't model sequences
  - Fixed-sized Inputs & Outputs
  - No temporal structure

- Problem 2: Pure feed-forward processing
  - No "memory", no feedback

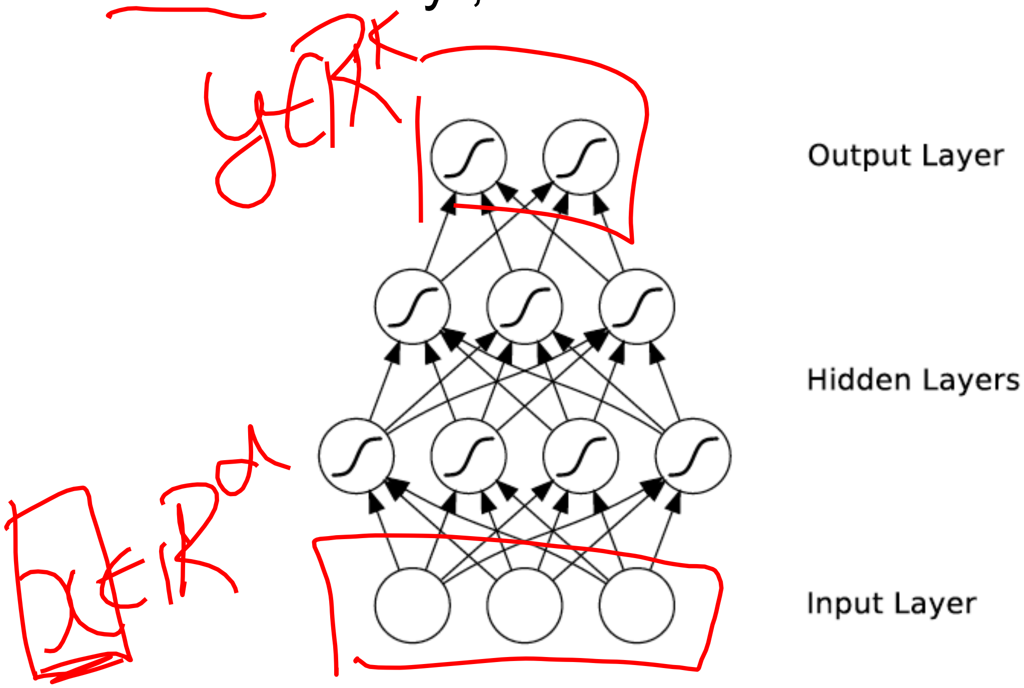
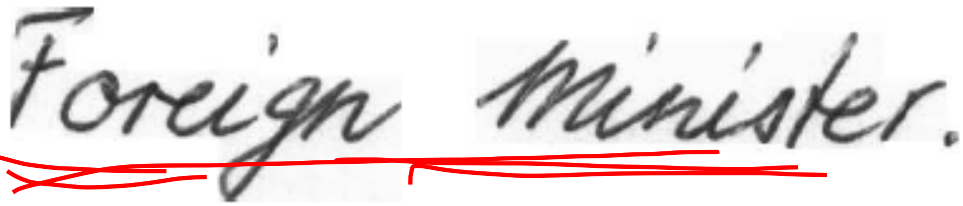
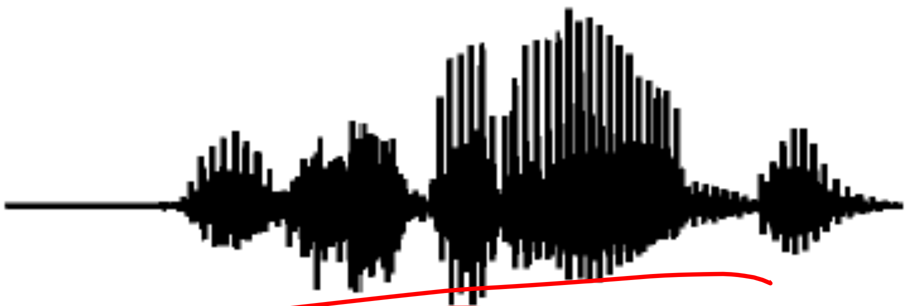


Image Credit: Alex Graves, book

Sequences are everywhere…



FOREIGN MINISTER.



THE SOUND OF

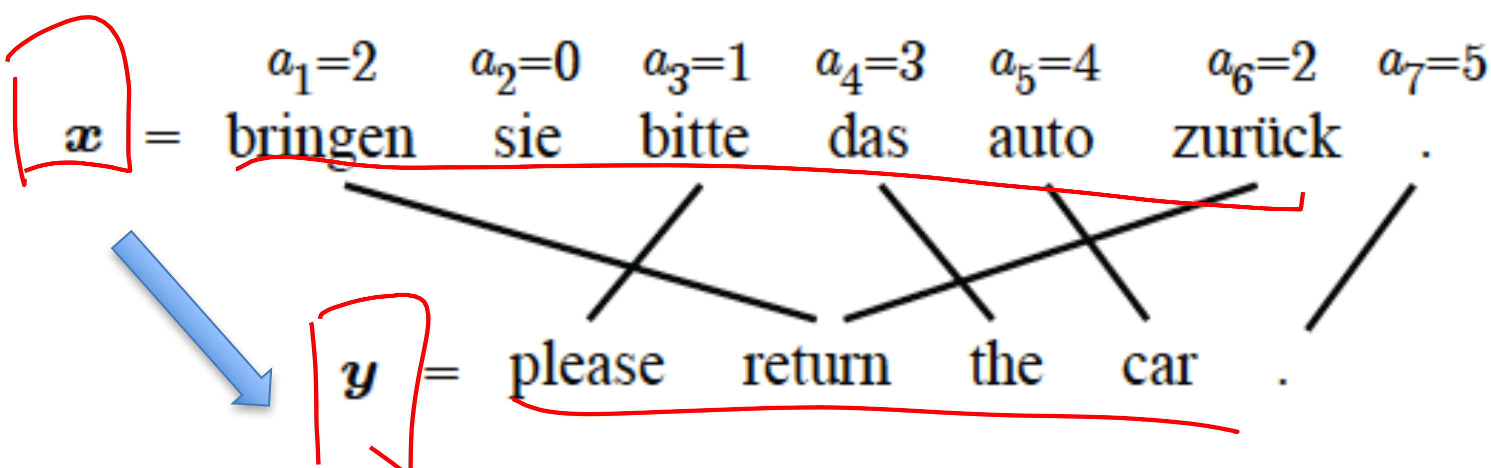
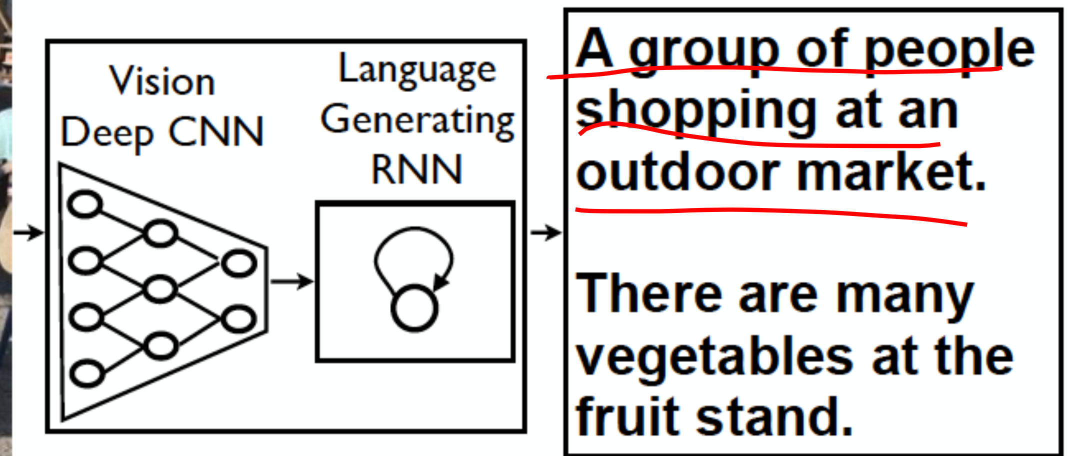
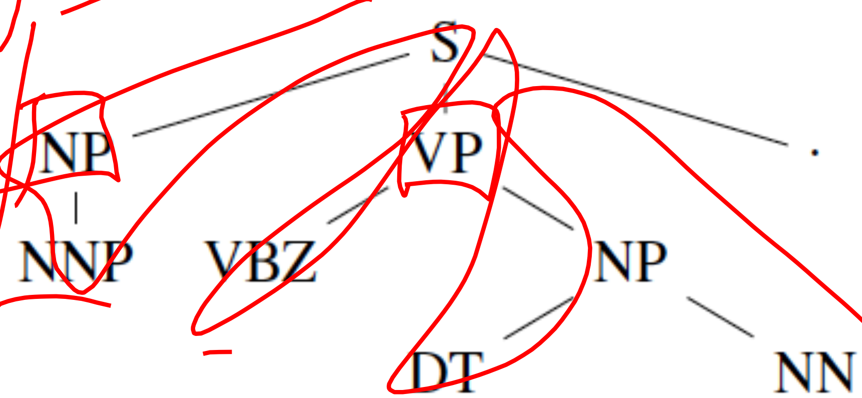


Image Credit: Alex Graves and Kevin Gimpel

Even where you might not expect a sequence…



John has a dog . →



John has a dog . → (S (NP NNP $)_{NP}$ (VP VBZ (NP DT NN $)_{NP}$ $)_{VP}$. $)_{S}$

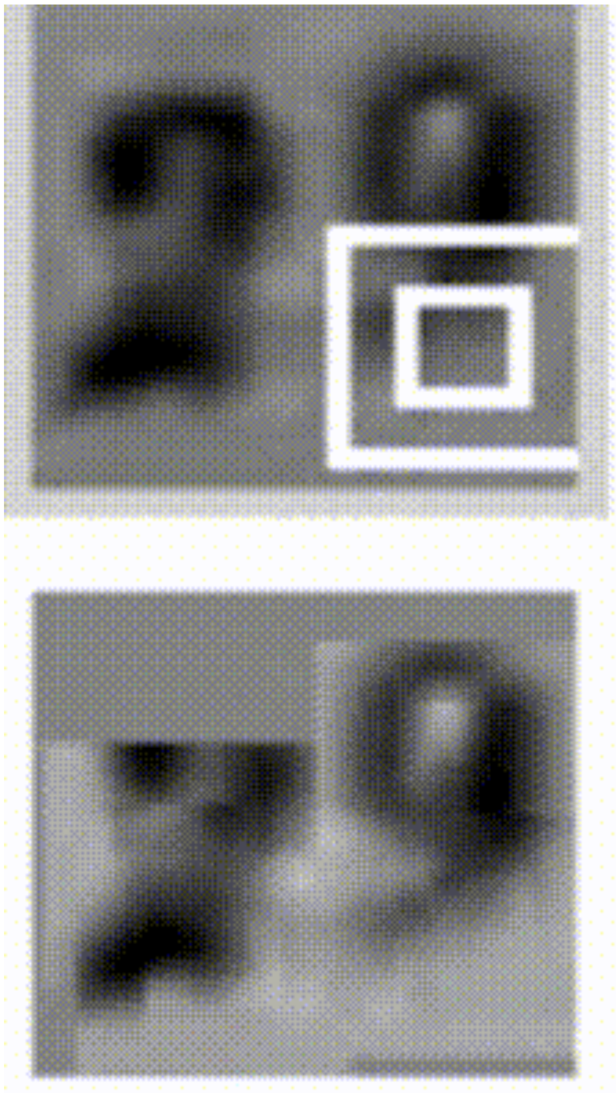
Even where you might not expect a sequence…

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

Even where you might not expect a sequence…

- Output ordering = sequence



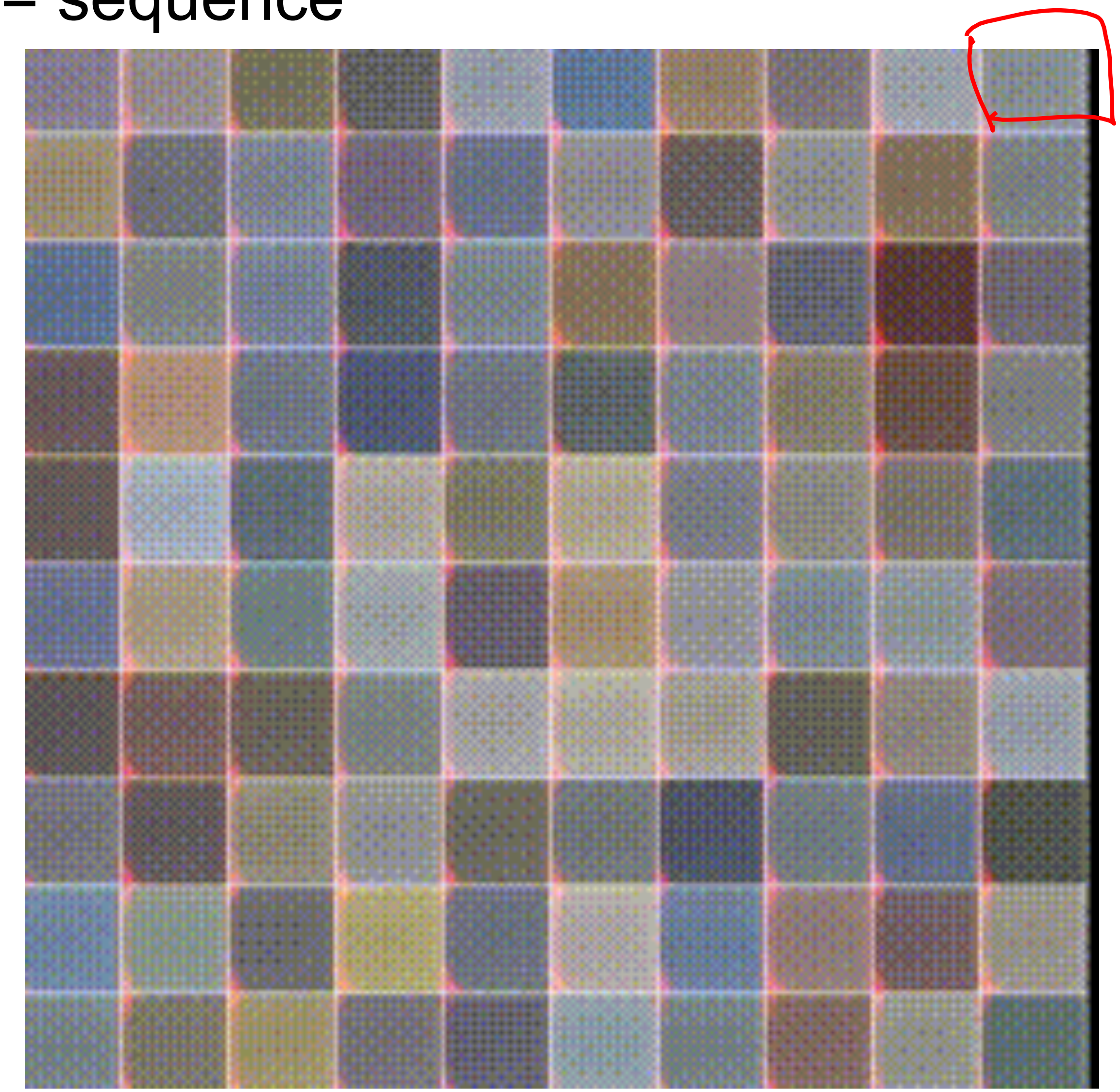


Image Credit: Ba et al.; Gregor et al

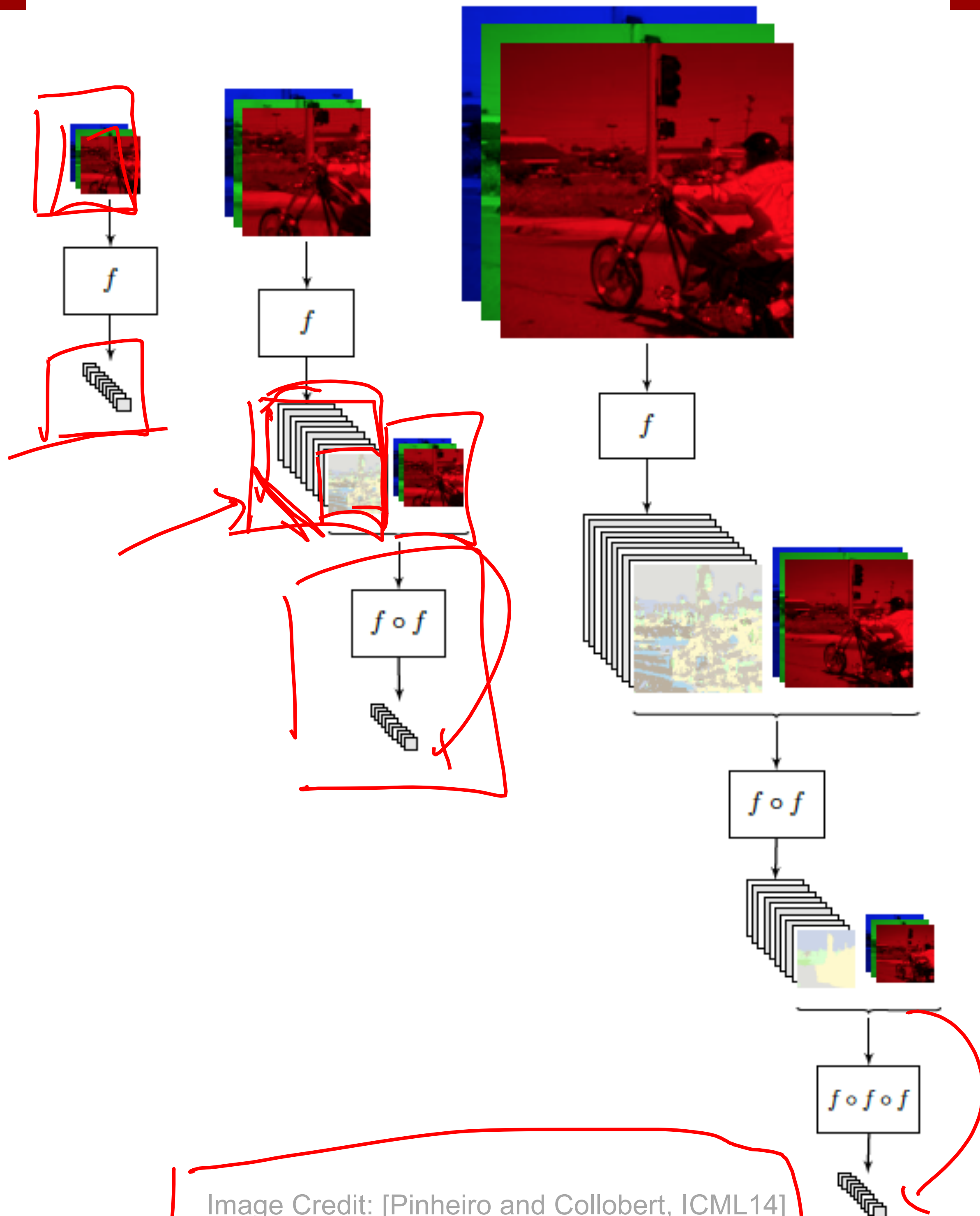


Image Credit: [Pinheiro and Collobert, ICML14]

Why model sequences?

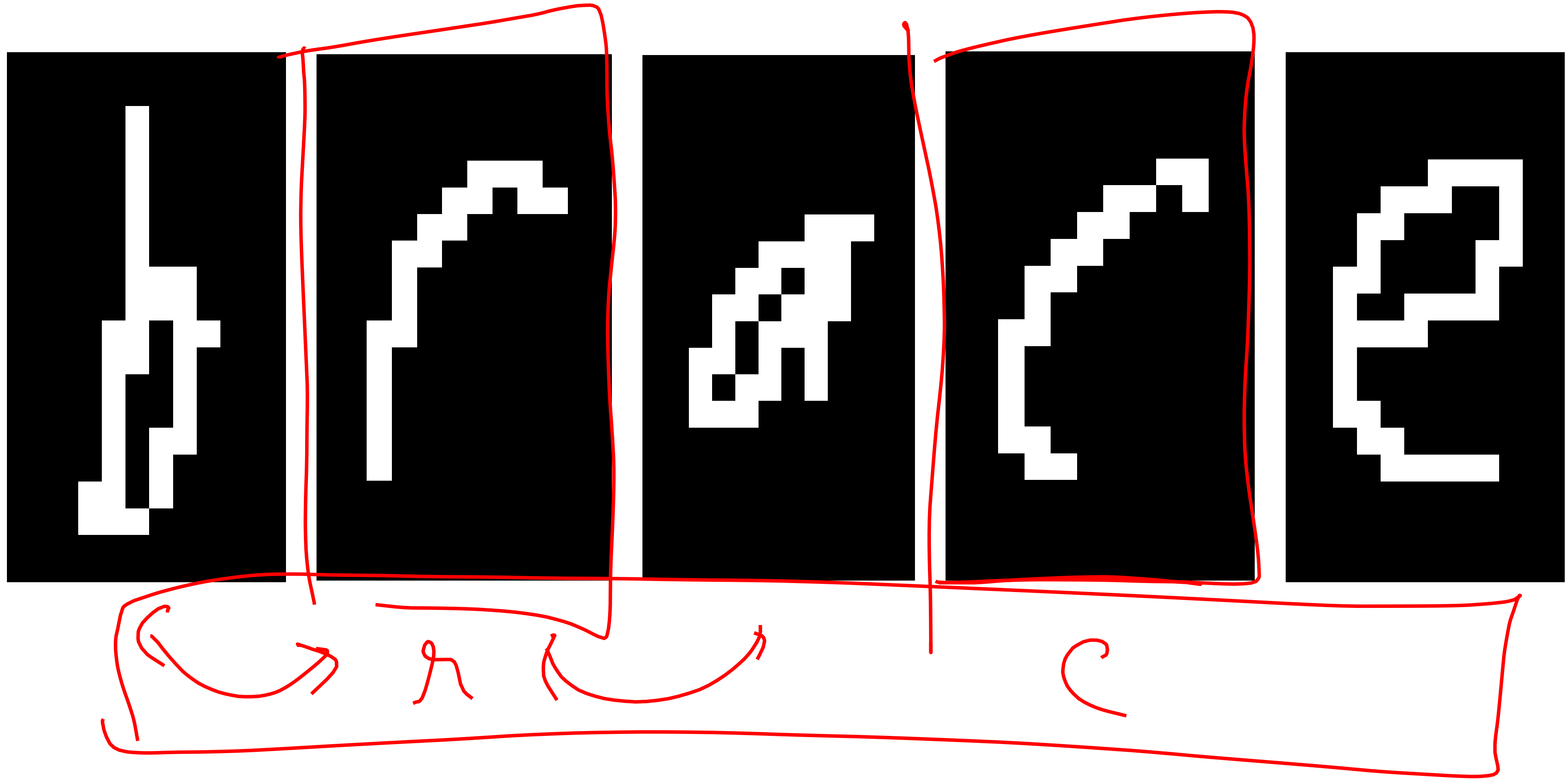


Figure Credit: Carlos Guestrin

Why model sequences?

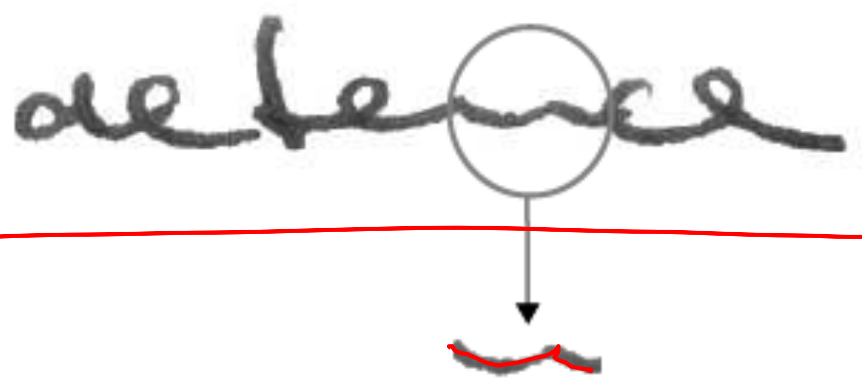
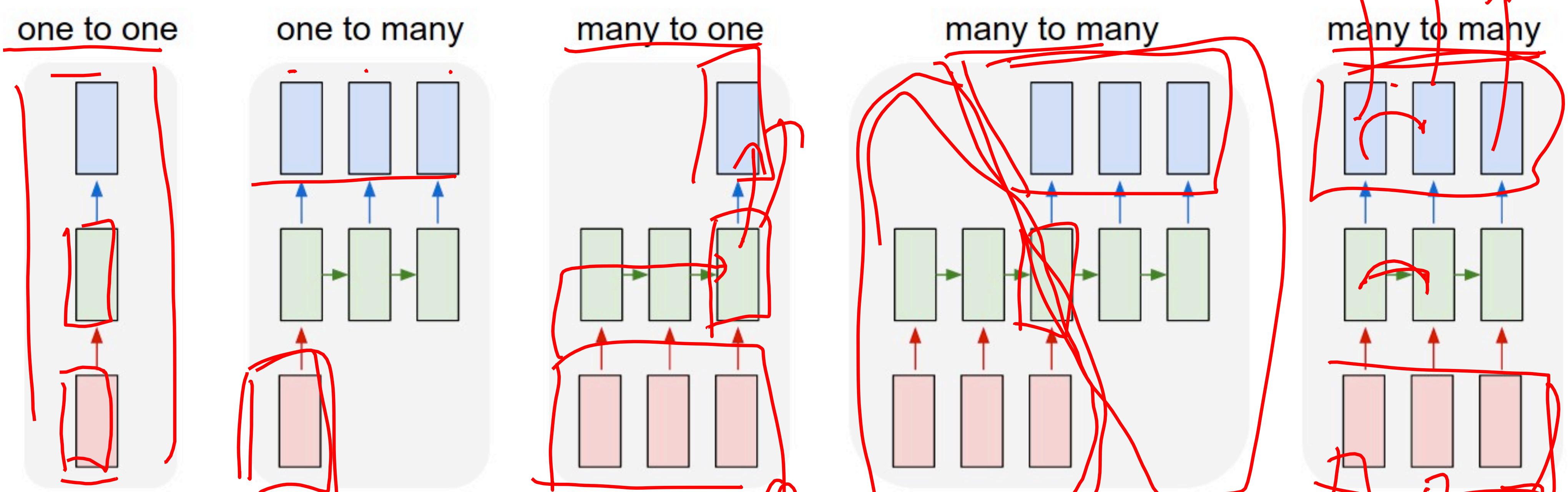


Image Credit: Alex Graves

Sequences in Input or Output?

- It's a spectrum…



Input: No sequence

Output: No sequence

Example: "standard" classification / regression problems

Input: No sequence

Output: Sequence

Example: Im2Caption

Input: Sequence

Output: No sequence

Example: sentence classification, multiple-choice question answering

Input: Sequence

Output: Sequence

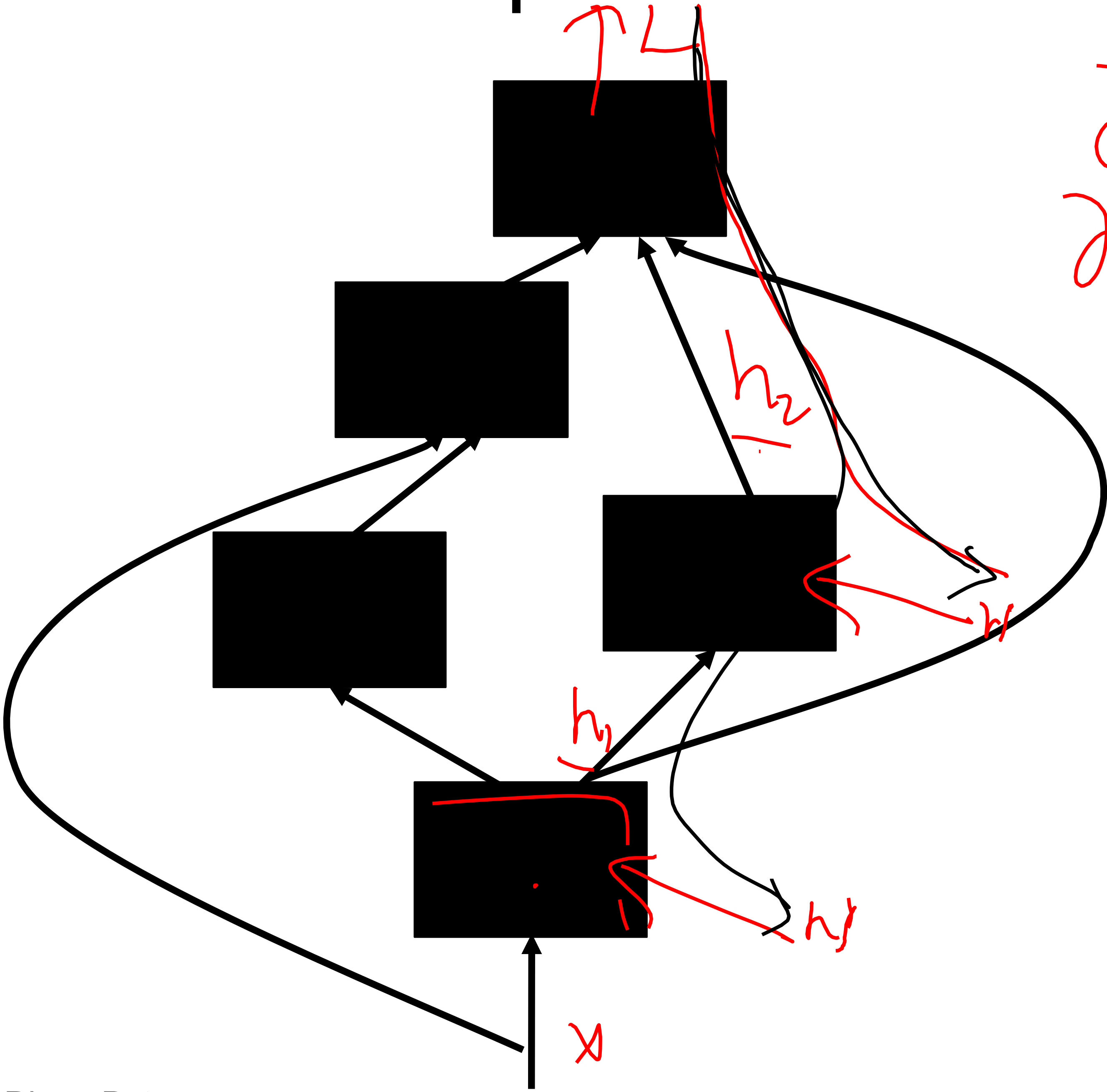
Example: machine translation, video captioning, open-ended question answering, video question answering

Image Credit: Andrej Karpathy

2 Key Ideas

- Parameter Sharing
  - in computation graphs = adding gradients

Computational Graph



$$\frac{\partial L}{\partial w} =$$

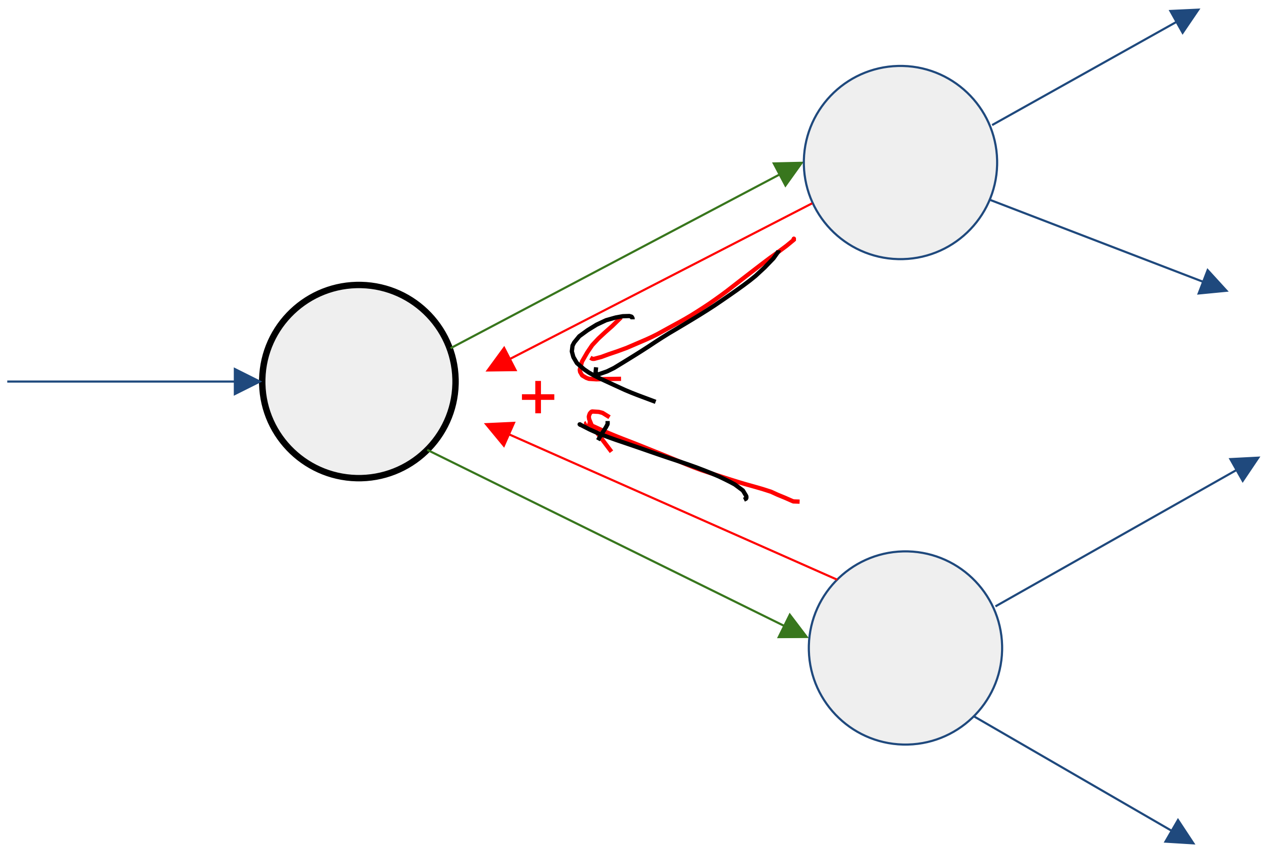
$$f(\underline{w_1}^{(t)}, \underline{w_2}^{(t)})$$

$$\frac{\partial f}{\partial w_1} = \lim_{h \to 0} \frac{f(w_1 + h, w_2)}{h}$$

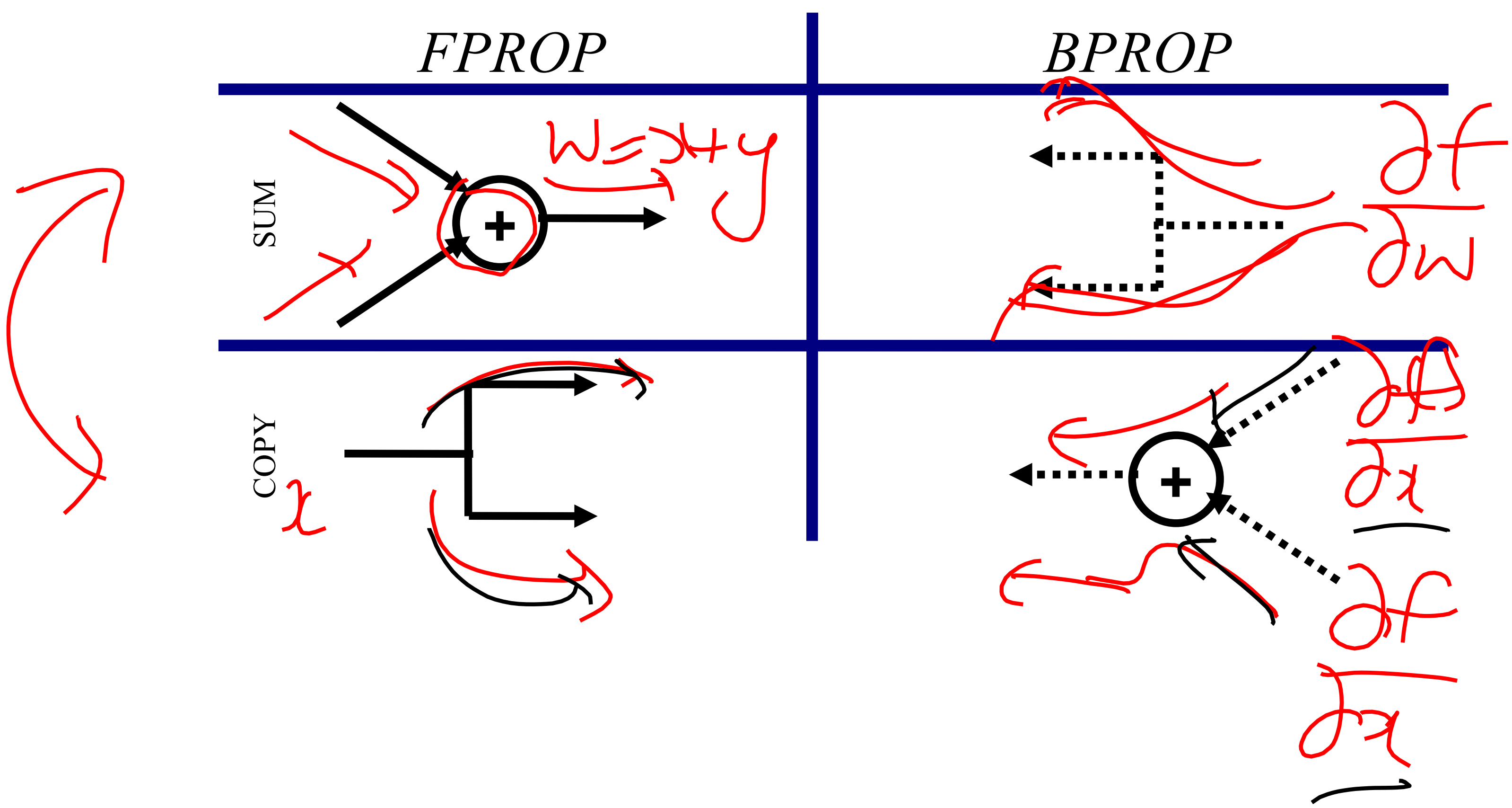
$$\frac{df}{dt} = \underbrace{\frac{\partial f}{\partial w_1}} \cancel{\frac{\partial w_1}{\partial t}} + \underbrace{\frac{\partial f}{\partial w_2}} \cancel{\frac{\partial w_2}{\partial t}}$$

$$\cancel{\frac{\partial f}{\partial w}} \quad \frac{df}{dw}$$

Gradients add at branches



Duality in Fprop and Bprop

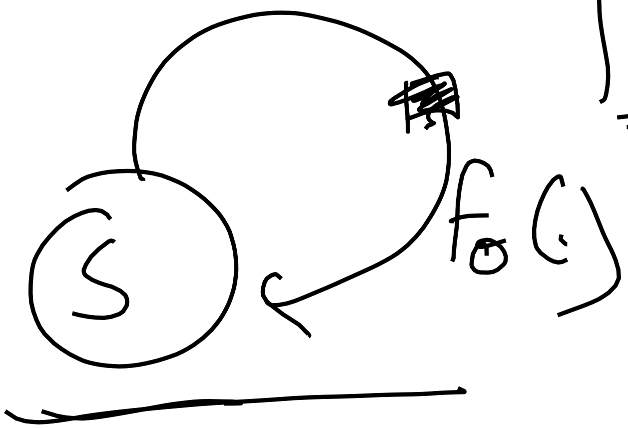


2 Key Ideas

- Parameter Sharing
  - in computation graphs = adding gradients
- "Unrolling"
  - in computation graphs with parameter sharing

How do we model sequences?

- No input



$$\boxed{s_t = f_\theta(s_{t-1})}$$

How do we model sequences?

- No input

$$s_t = f_\theta(s_{t-1})$$

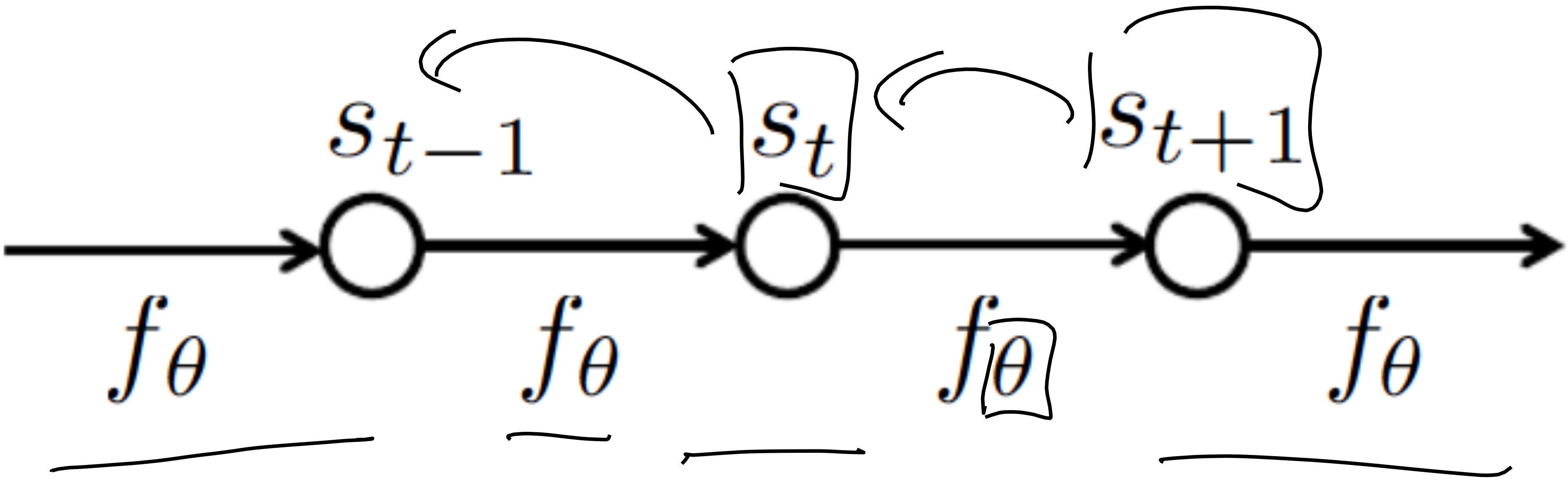


Image Credit: Bengio, Goodfellow, Courville

How do we model sequences?

- With inputs

$$s_t = f_\theta(s_{t-1}, x_t)$$

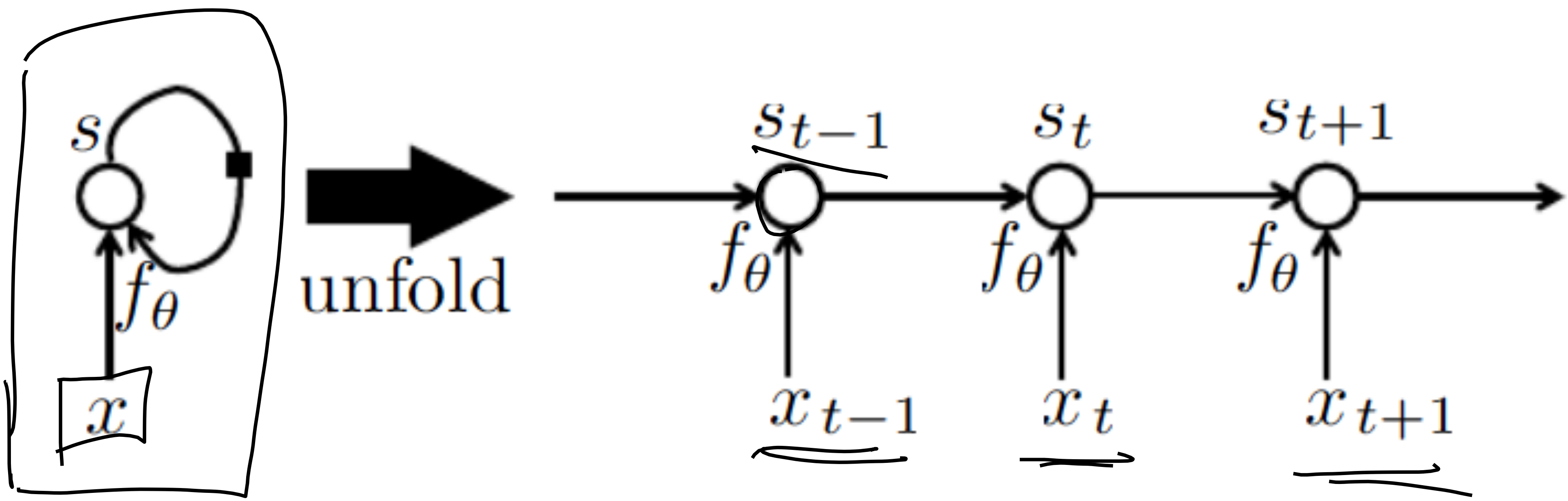
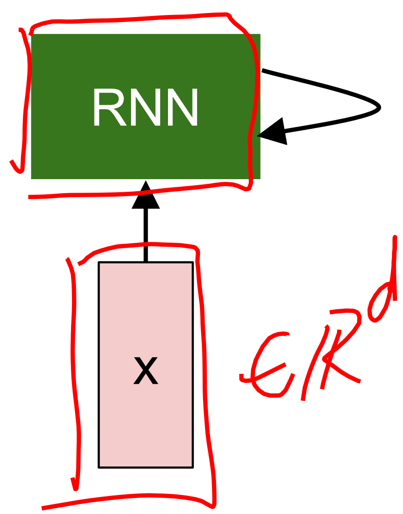


Image Credit: Bengio, Goodfellow, Courville

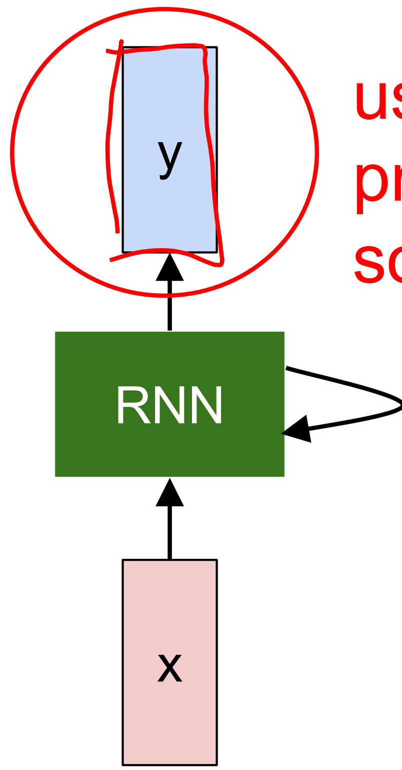
2 Key Ideas

- Parameter Sharing
  - in computation graphs = adding gradients
- "Unrolling"
  - in computation graphs with parameter sharing
- Parameter sharing + Unrolling
  - Allows modeling arbitrary sequence lengths!
  - Keeps numbers of parameters in check

Recurrent Neural Network



Recurrent Neural Network



usually want to predict a vector at some time steps

Recurrent Neural Network

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

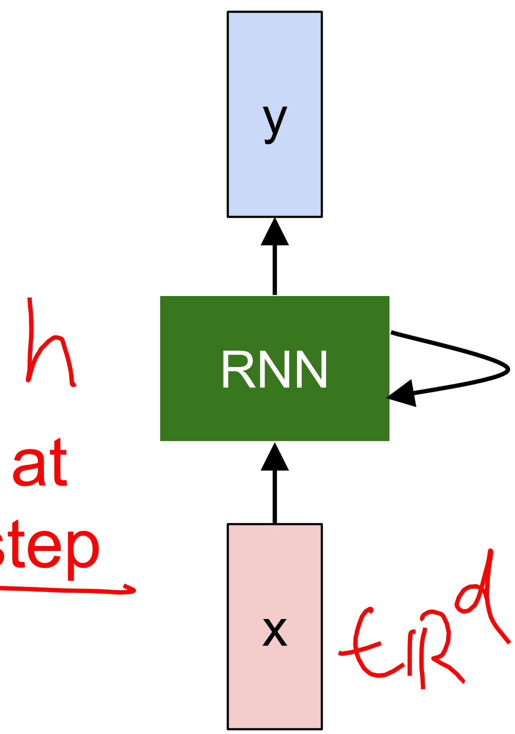
$$\boxed{h_t} = \boxed{f_W}(\boxed{h_{t-1}}, \boxed{x_t})$$

new state

some function with parameters W

old state

input vector at some time step

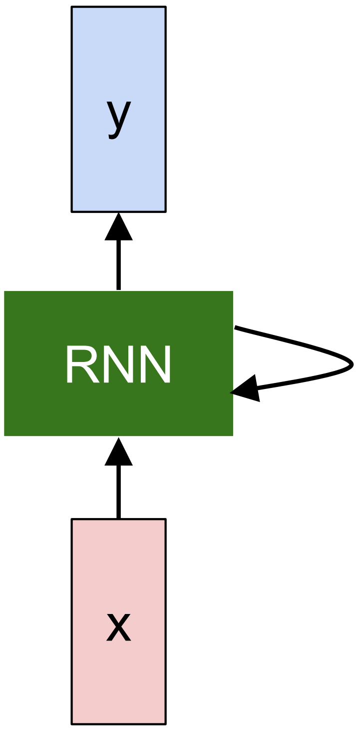


Recurrent Neural Network

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

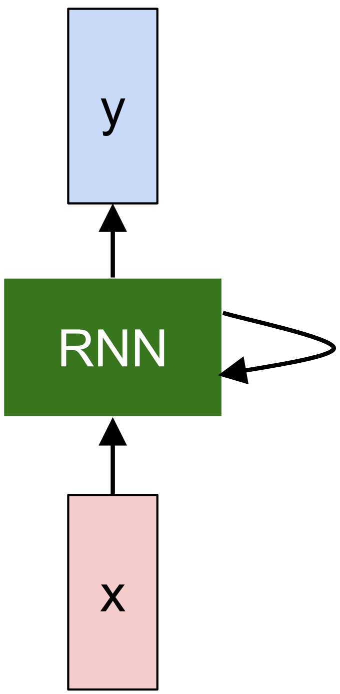
$$h_t = \underline{f_W(h_{t-1}, x_t)}$$

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



(Vanilla) Recurrent Neural Network

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



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

$d_2 \times (d_1 + d_2)$

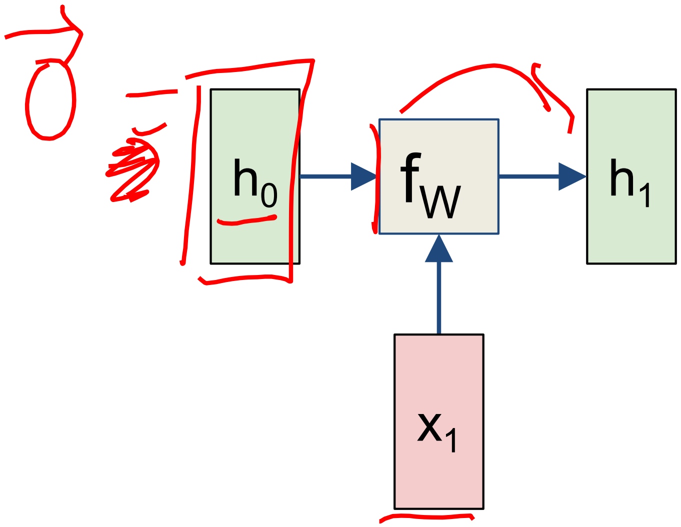
$\tanh\left(W \begin{pmatrix} \vec{h}_{t-1} \\ \vec{x}_t \end{pmatrix} + \vec{b}\right)$ d_2 d_1

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

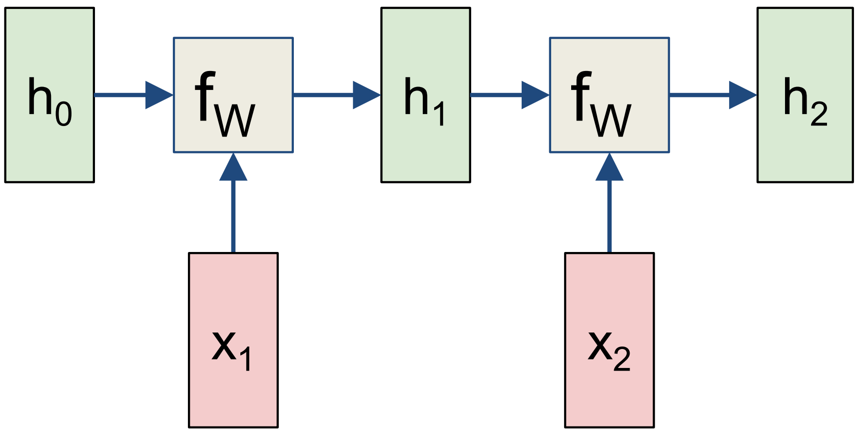
$$\vec{h}_t = \boxed{\tanh}(\boxed{W_{hh}} \vec{h}_{t-1} + \boxed{W_{xh}} \vec{x}_t + \vec{b}_h)$$

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

RNN: Computational Graph



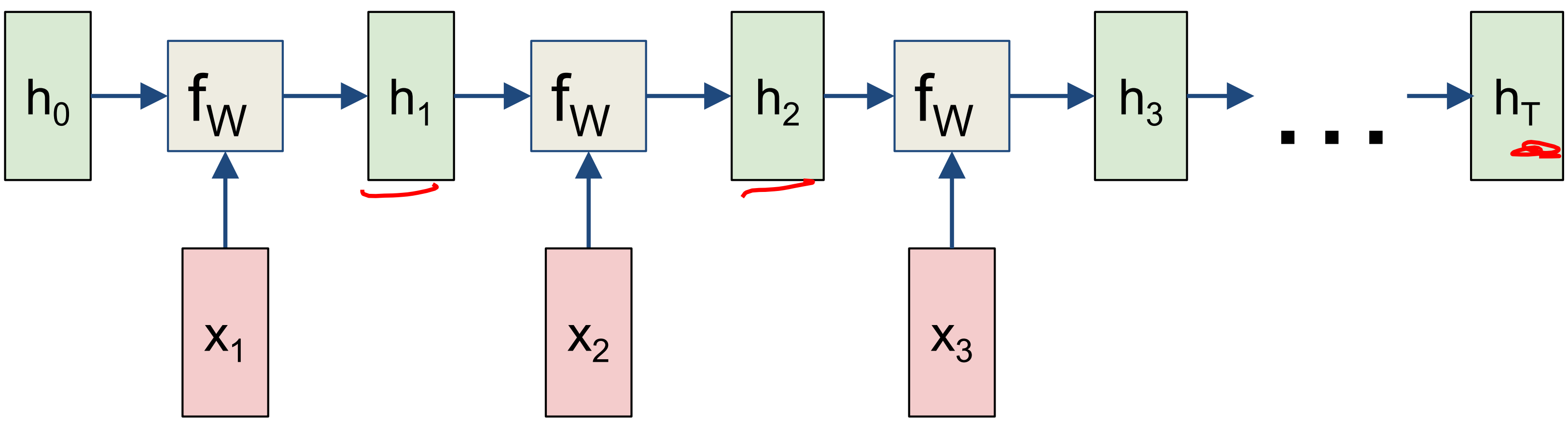
RNN: Computational Graph



Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

RNN: Computational Graph

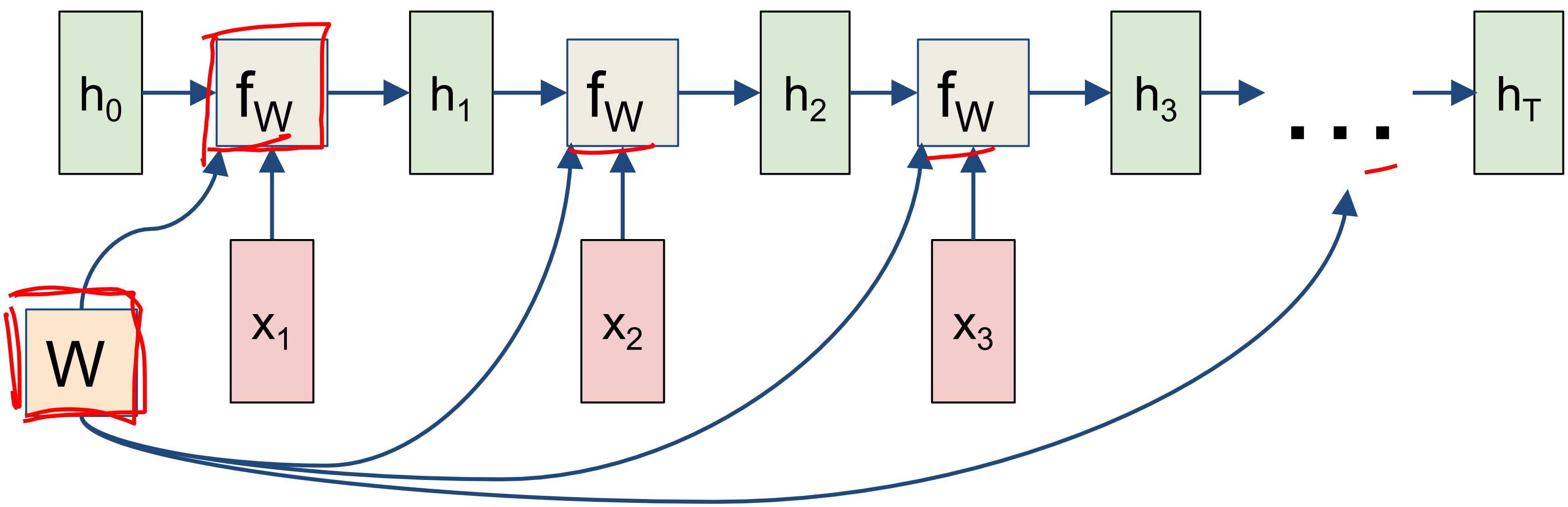
$$h_2 = f_W(h_1, x_2)$$



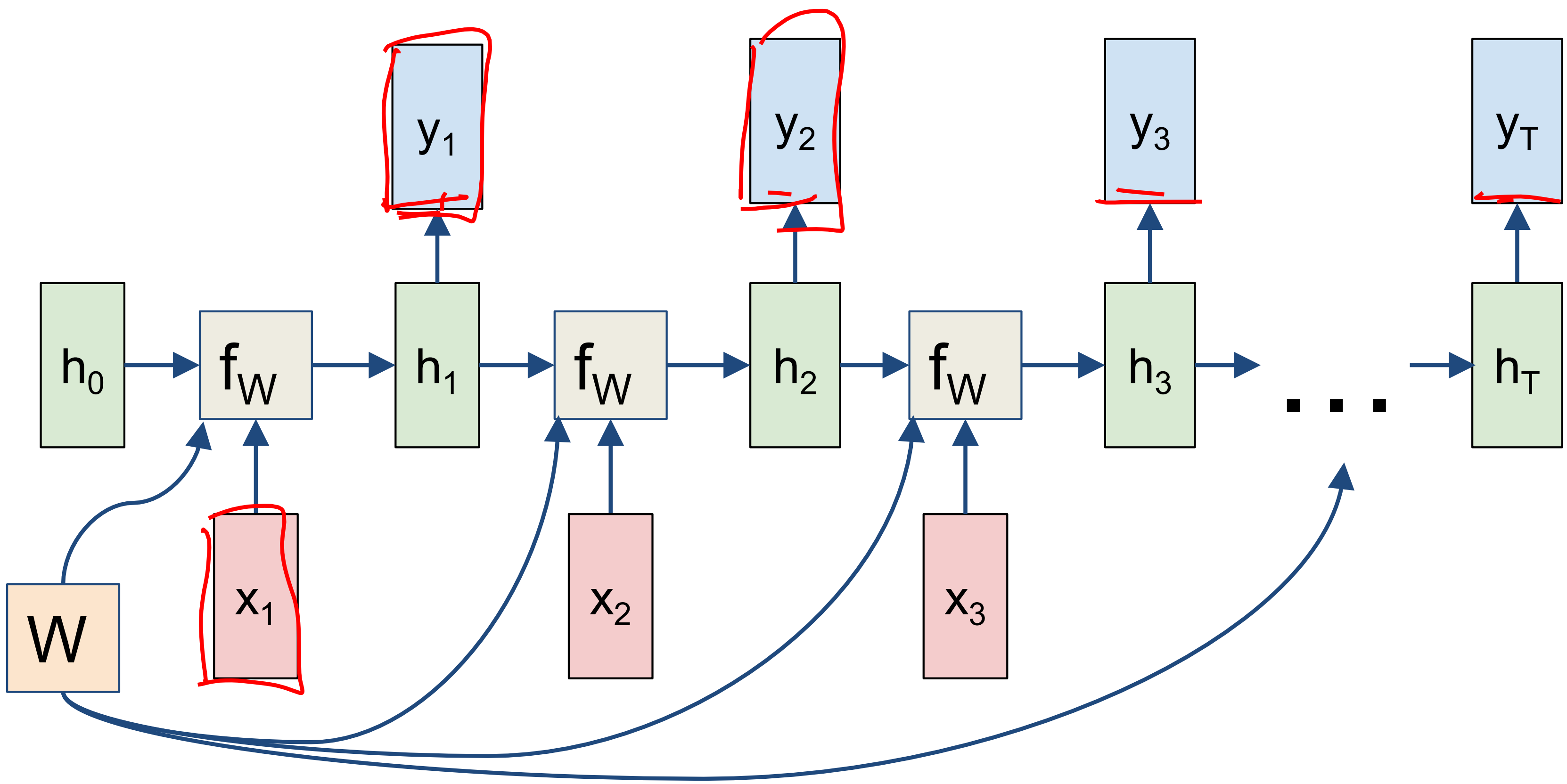
Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

RNN: Computational Graph

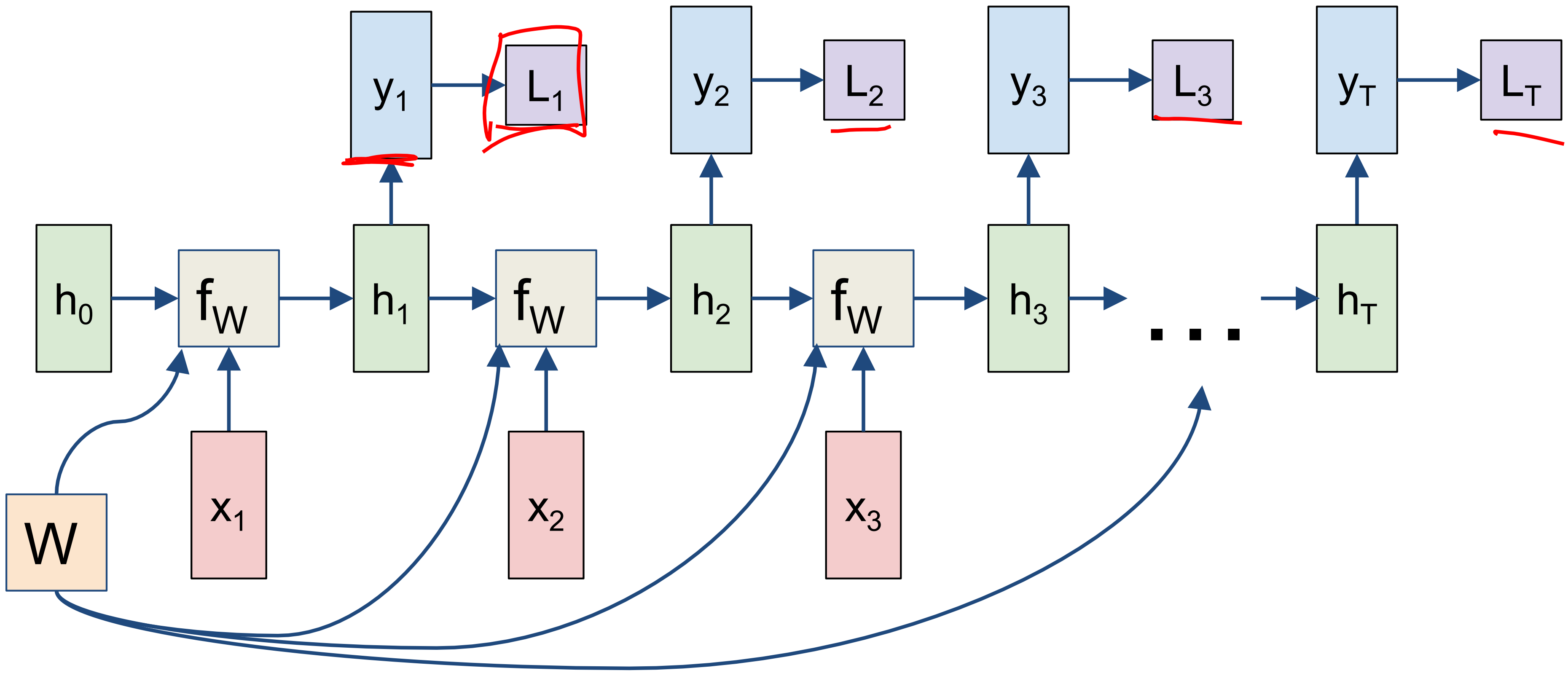
Re-use the same weight matrix at every time-step



RNN: Computational Graph: Many to Many

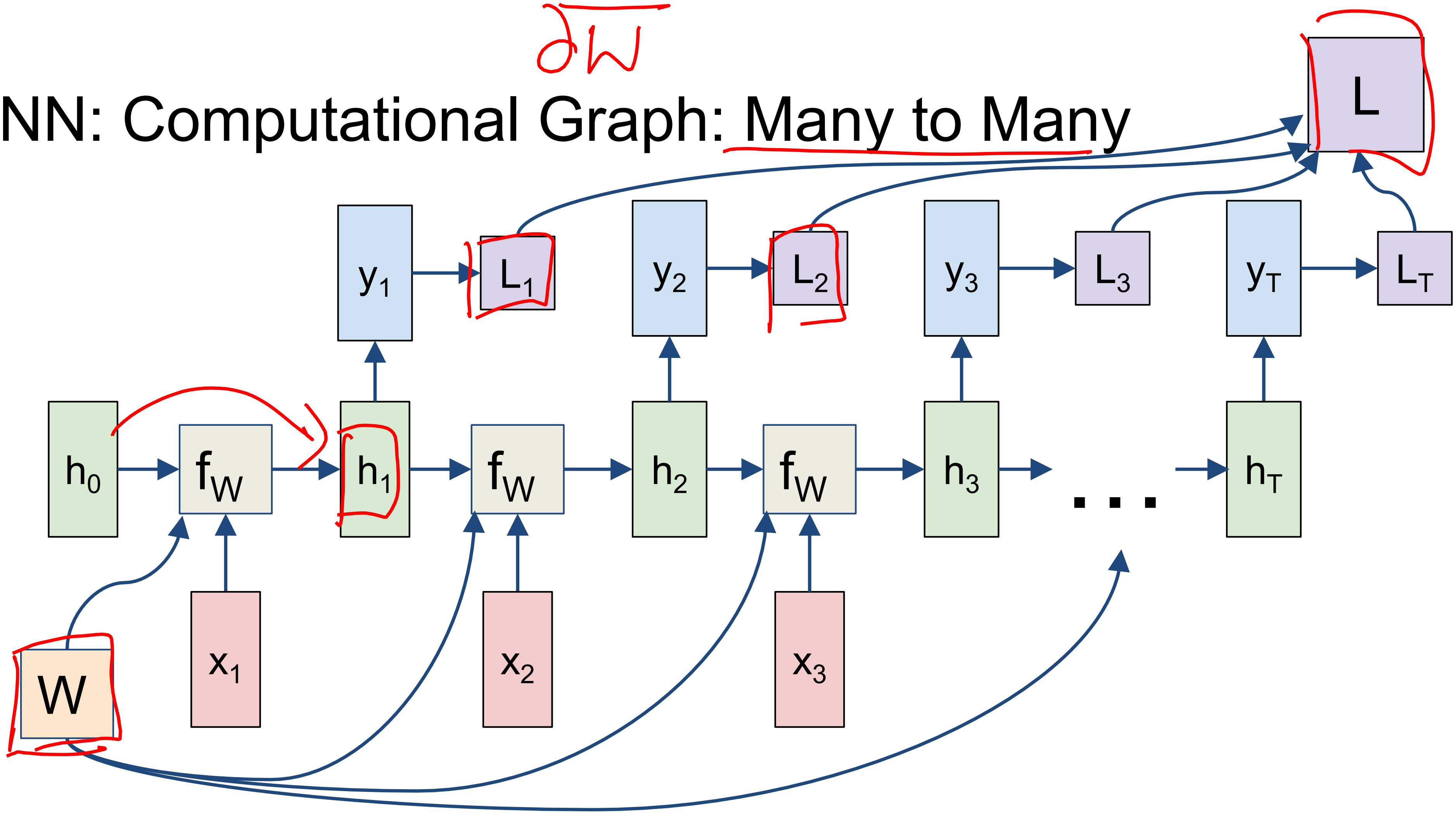


RNN: Computational Graph: Many to Many

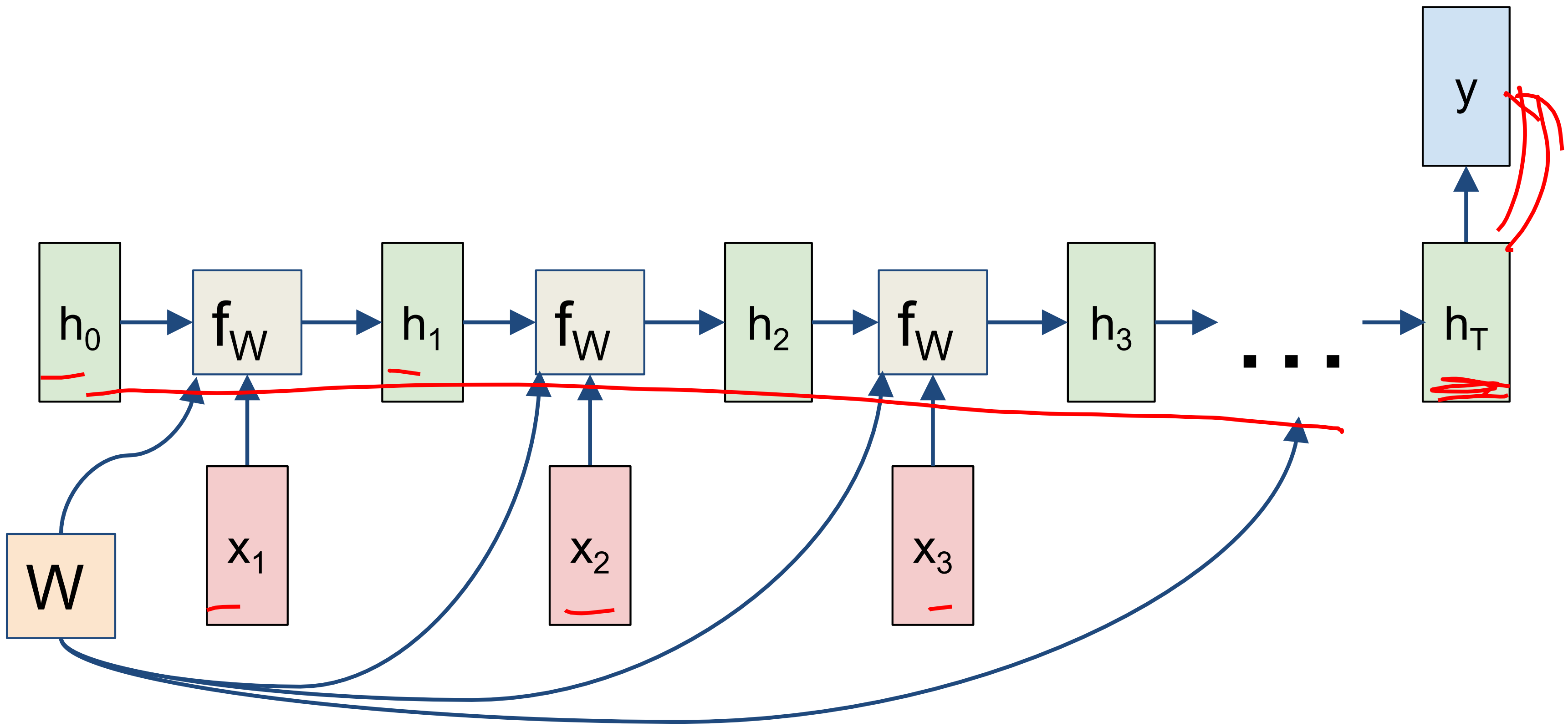


$\frac{\partial L}{\partial W}$

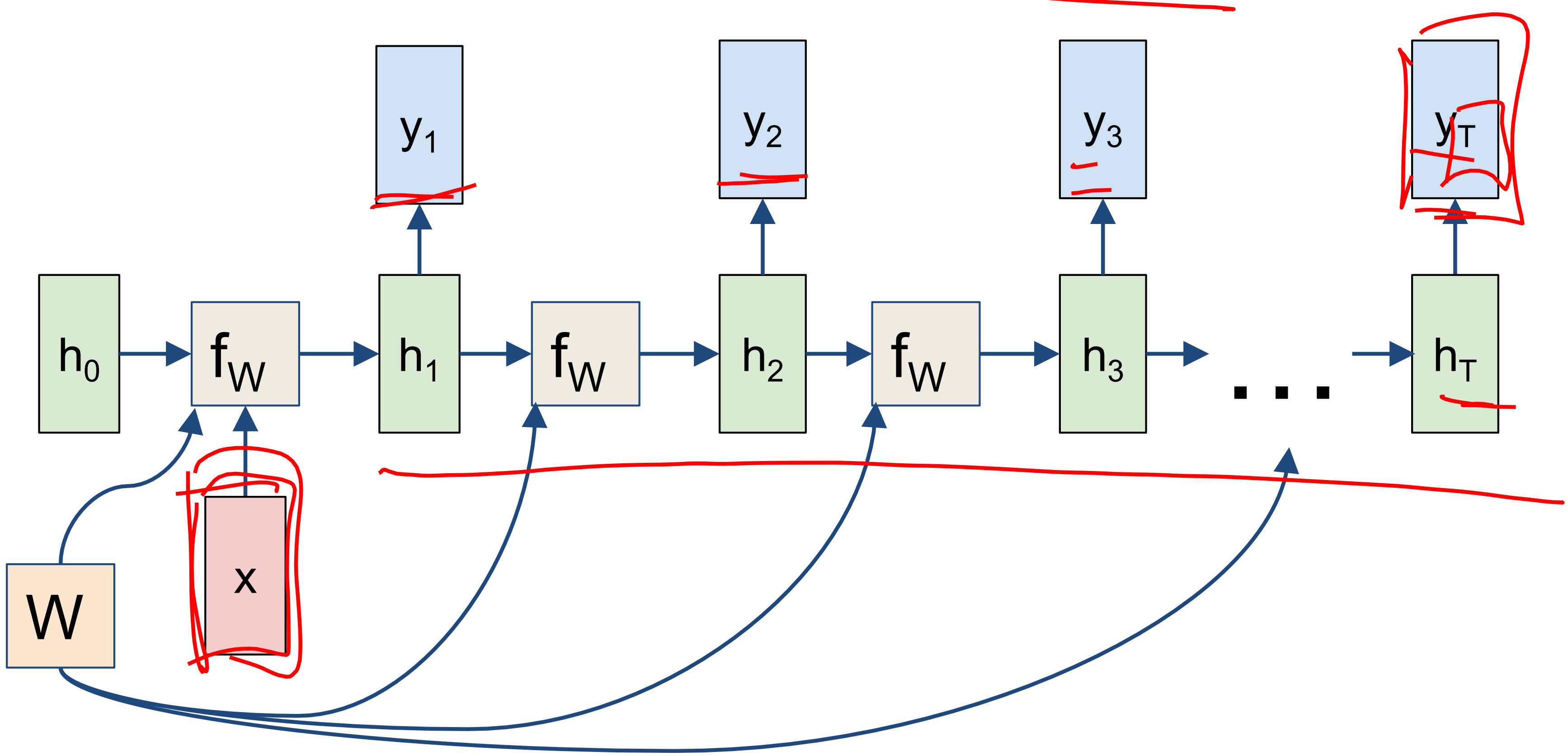
RNN: Computational Graph: Many to Many



RNN: Computational Graph: Many to One

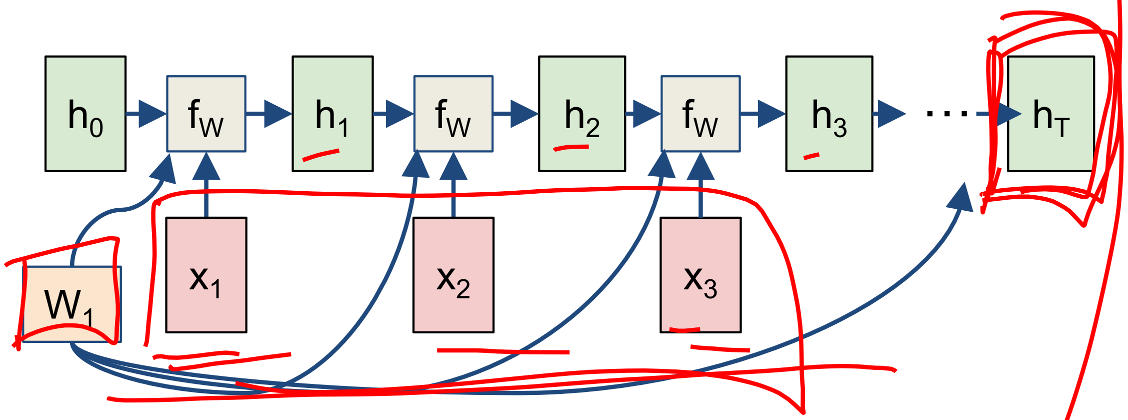


RNN: Computational Graph: One to Many



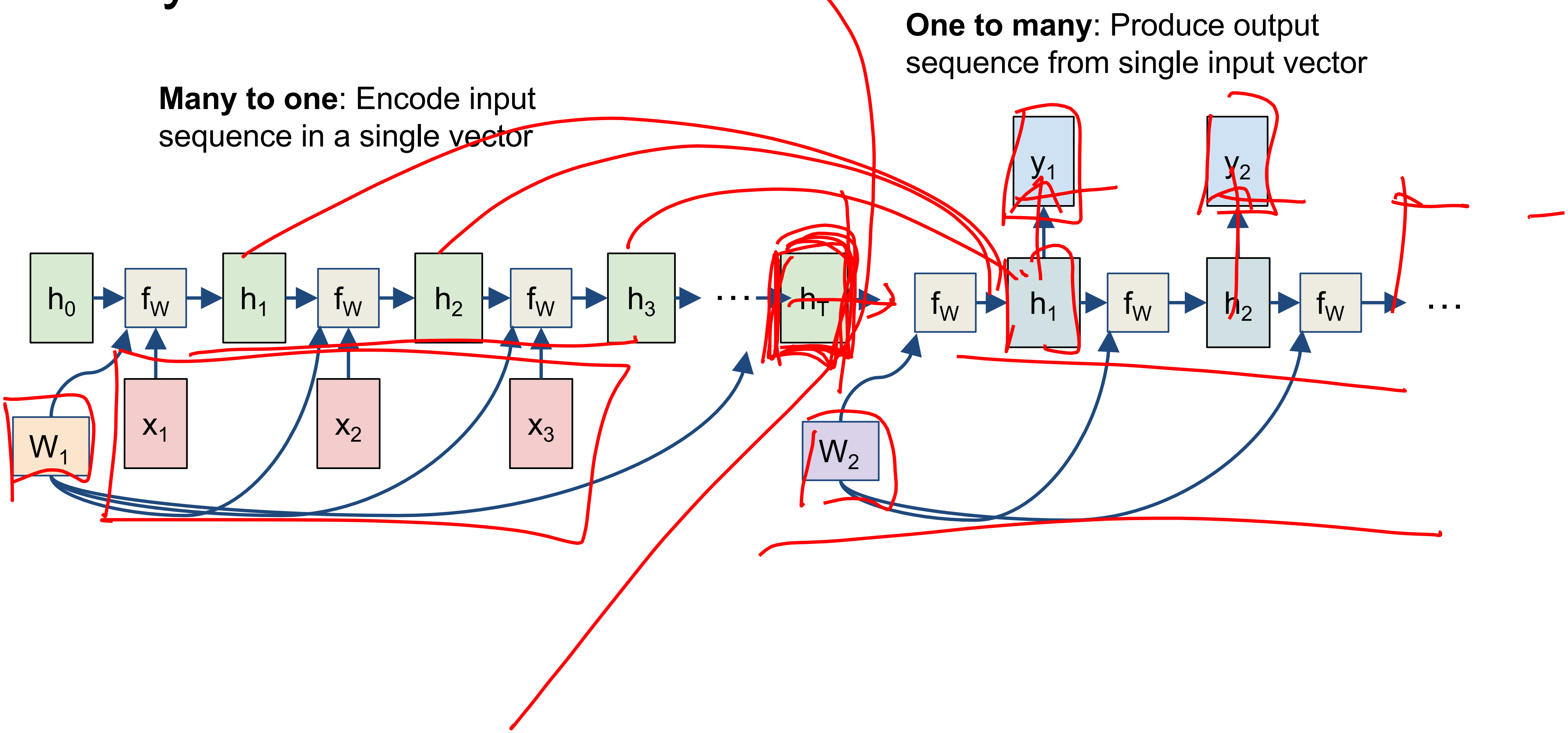
Sequence to Sequence: Many-to-one + one-to-many

Many to one: Encode input sequence in a single vector



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

$\min_{\theta} L(W_1, W_2)$

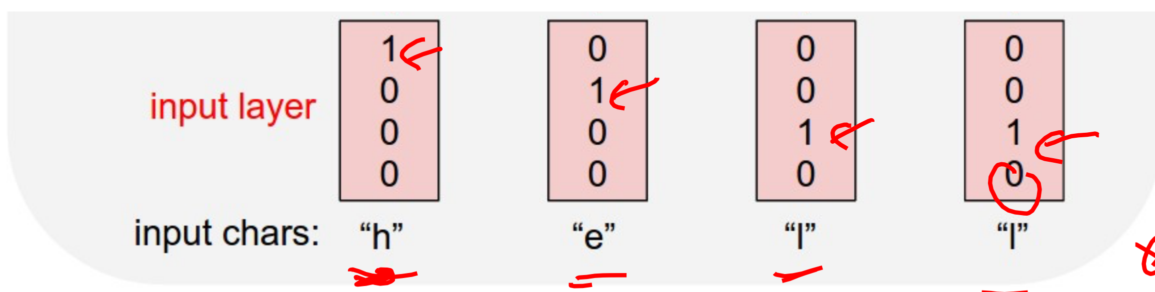


$P(x_1, x_2, x_3, x_4)$

Example:
Character-level Language Model

Vocabulary:
[h,e,l,o]

Example training sequence:
“hello”



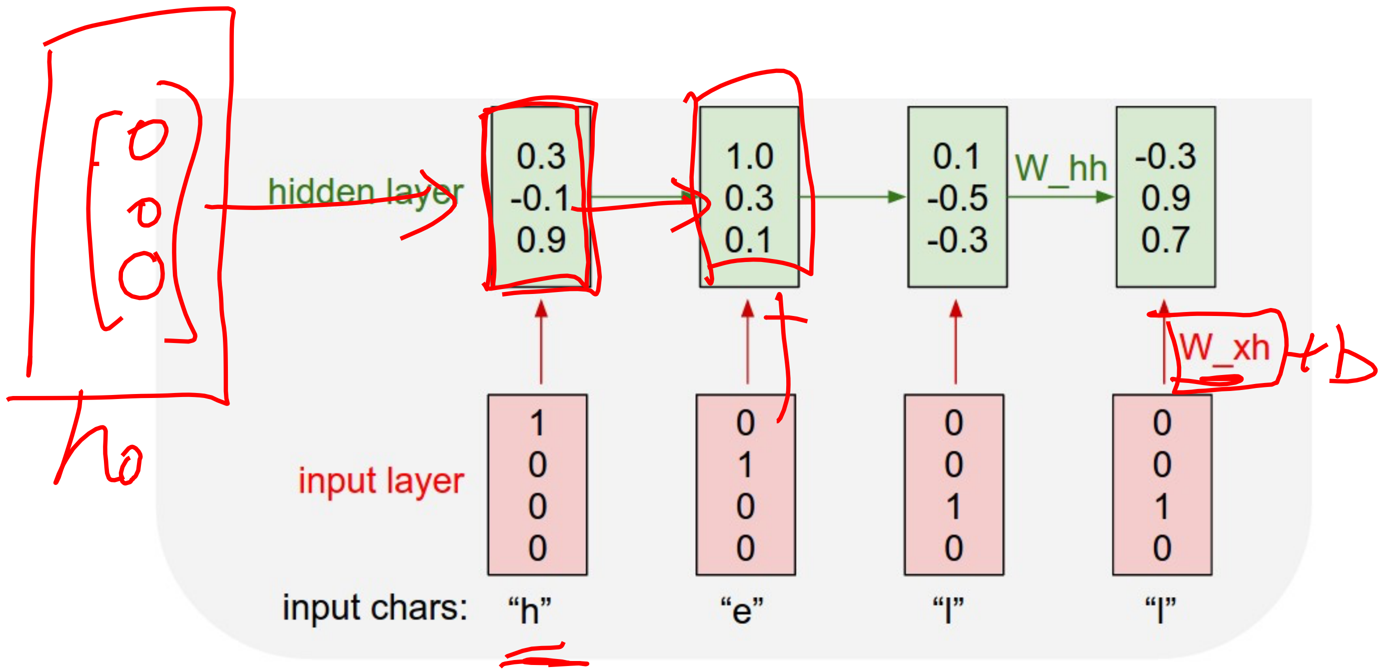
x_1 x_2 x_3 x_4

Example:
Character-level Language Model

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

Vocabulary:
[h,e,l,o]

Example training sequence:
“hello”



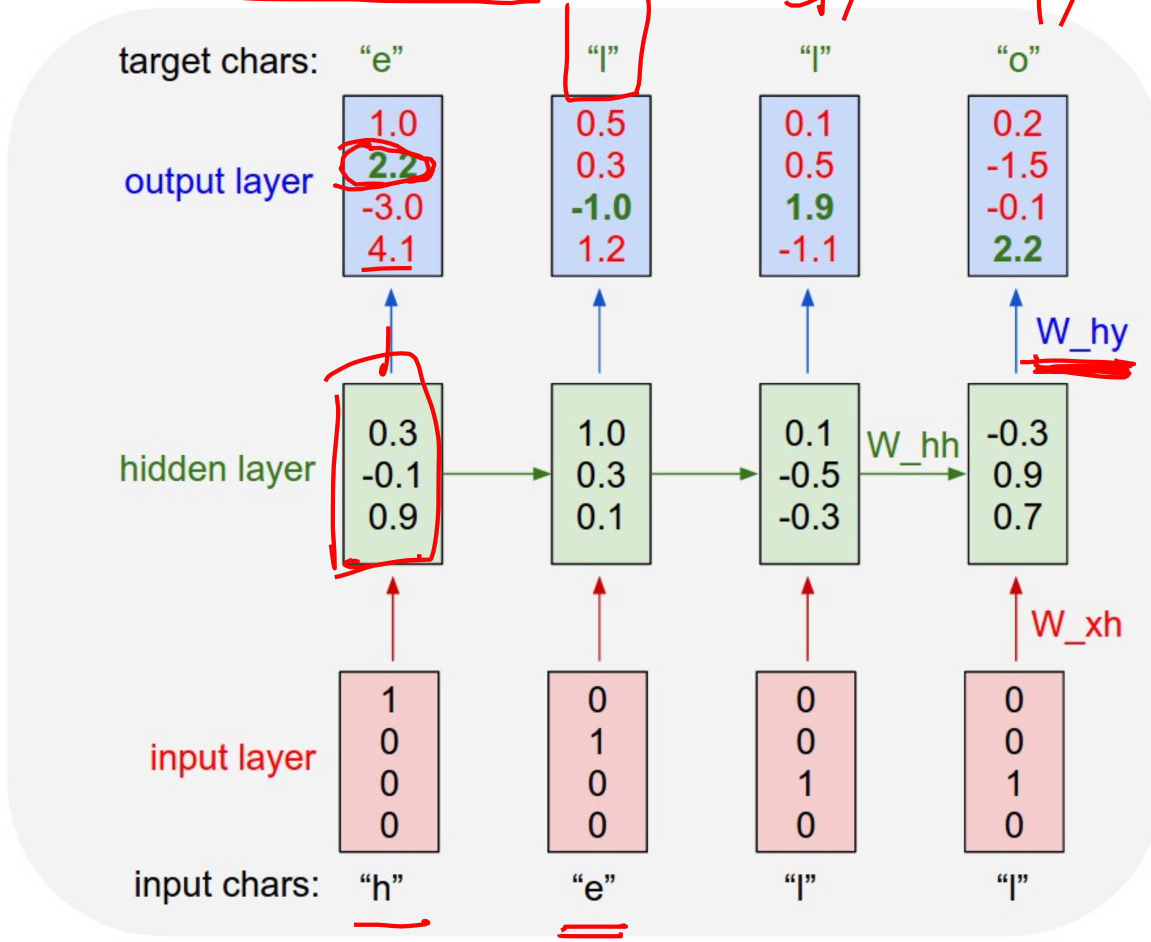
Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

$$\arg\max_{w} \sum \log \boxed{P(x_t \mid x_1 \ldots x_{t-1})}$$

Example: Character-level Language Model

Vocabulary: [h,e,l,o]

Example training sequence: **“hello”**

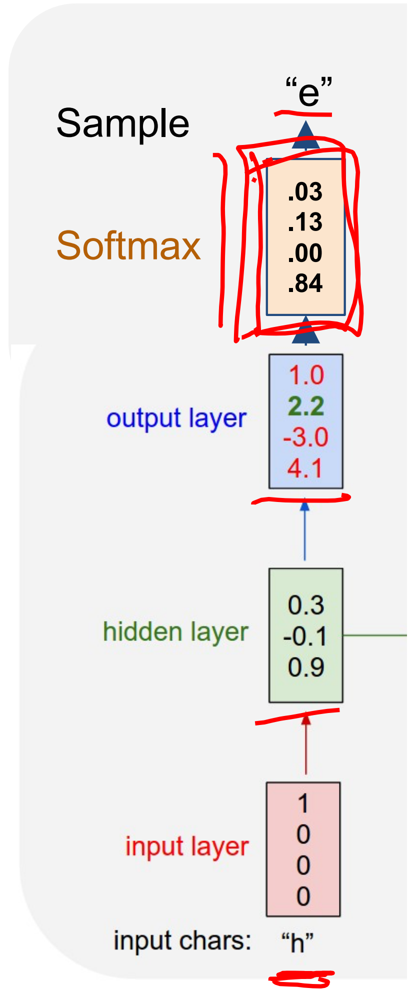


$$\hat{x}_2 \sim P(X_2 \mid X_1 = x_1, W)$$

Example: Character-level Language Model Sampling

Vocabulary:
[h,e,l,o]

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



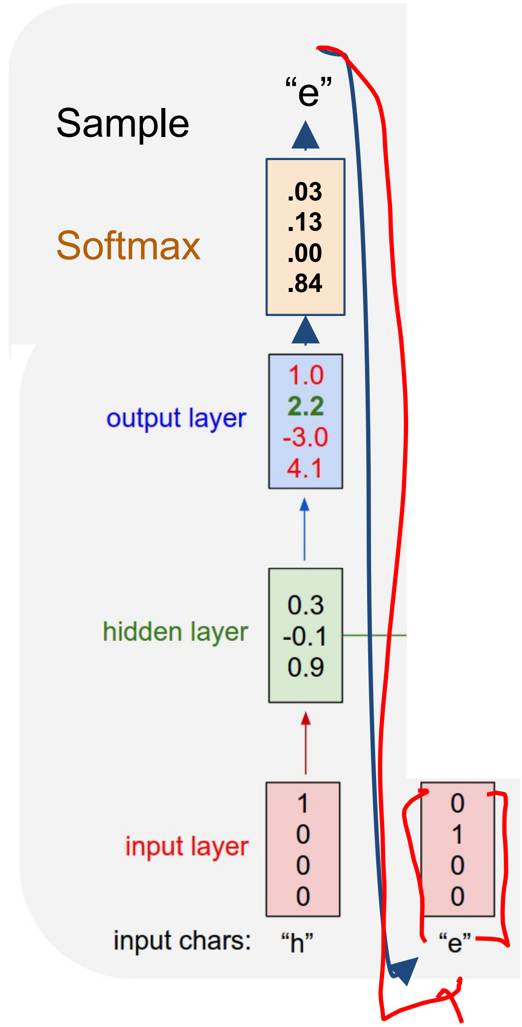
x_1 x_2

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Example: Character-level Language Model Sampling

Vocabulary: [h,e,l,o]

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

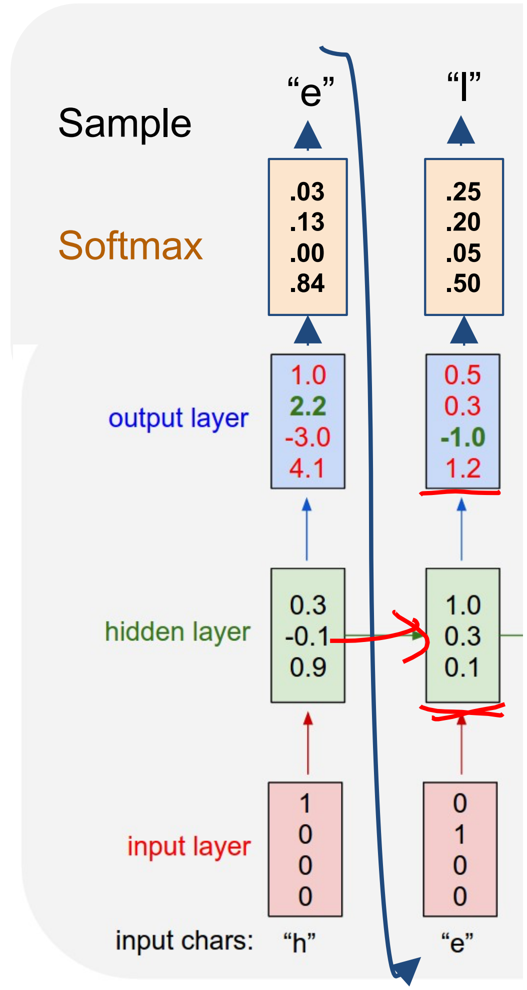


$x_3 \sim P(X_3 \mid X_1 = x_1, X_2 = x_2, W)$

Example:
Character-level
Language Model
Sampling

Vocabulary:
[h,e,l,o]

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

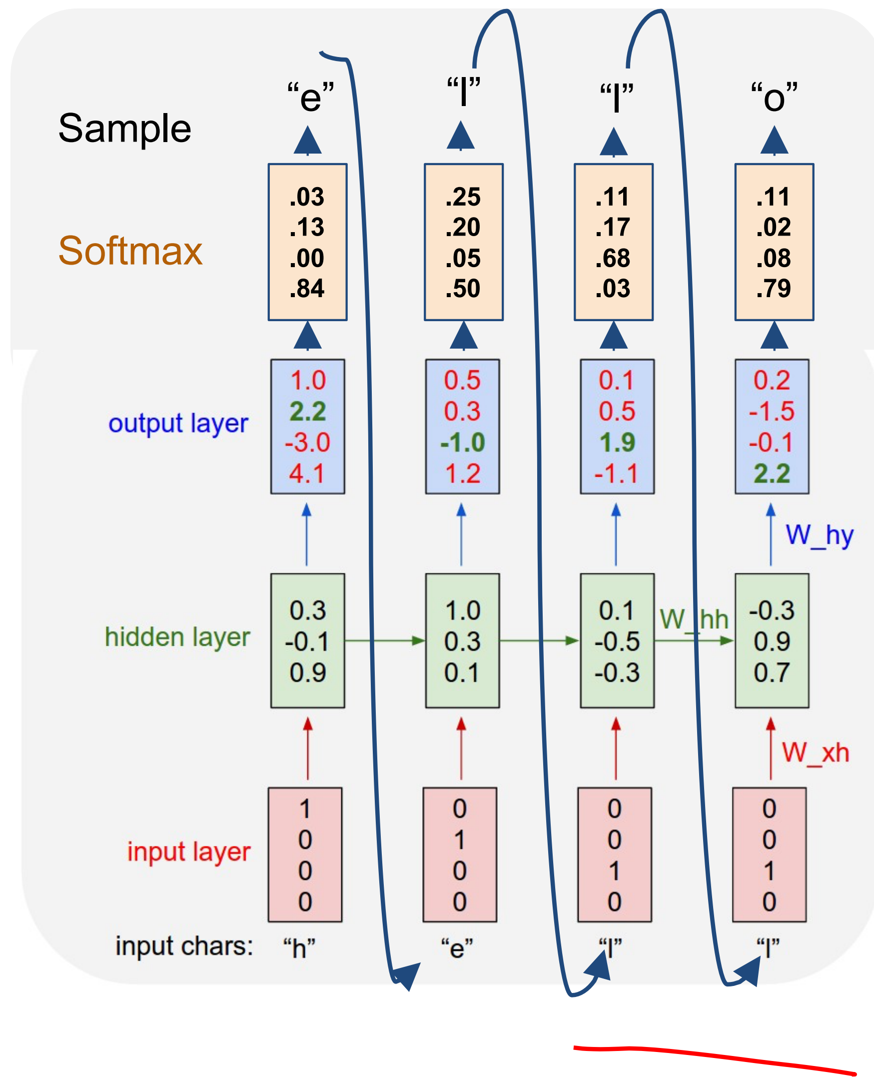


Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Example: Character-level Language Model Sampling

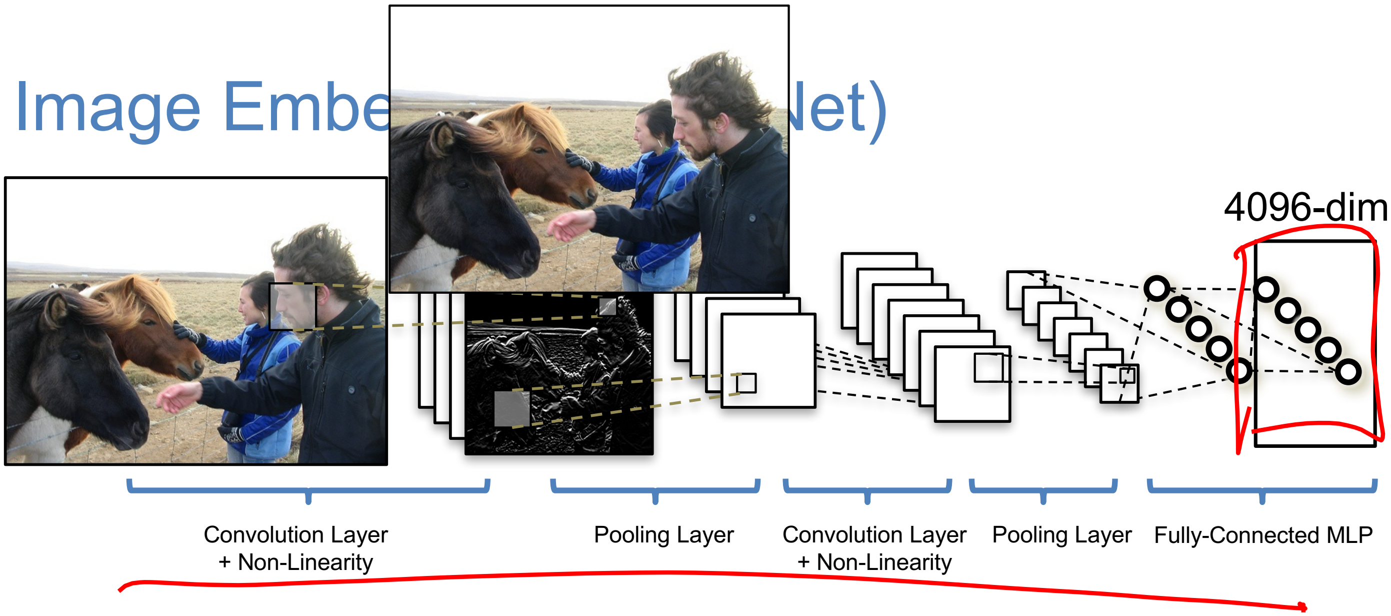
Vocabulary: [h,e,l,o]

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



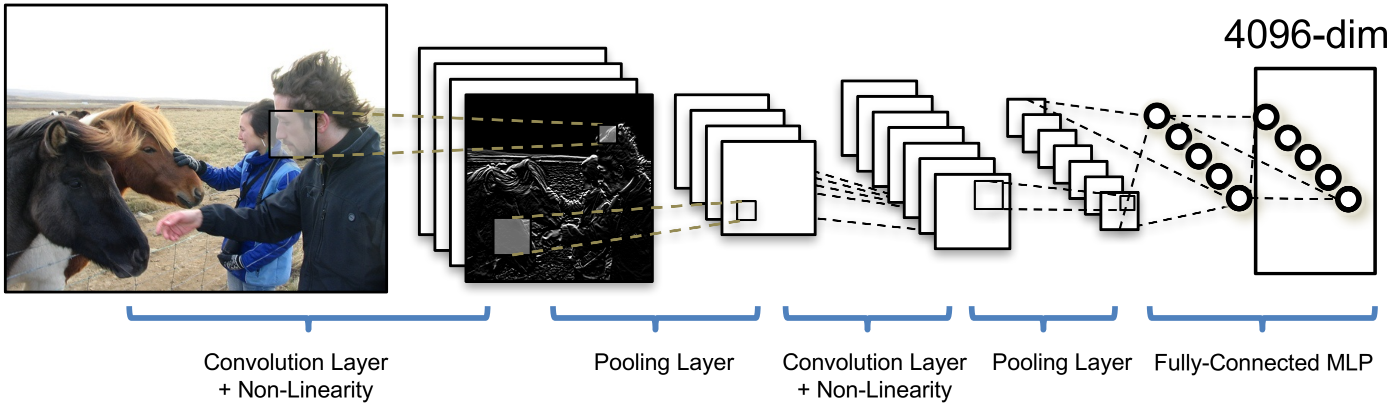
Neural Image Captioning

Image Embe Net)

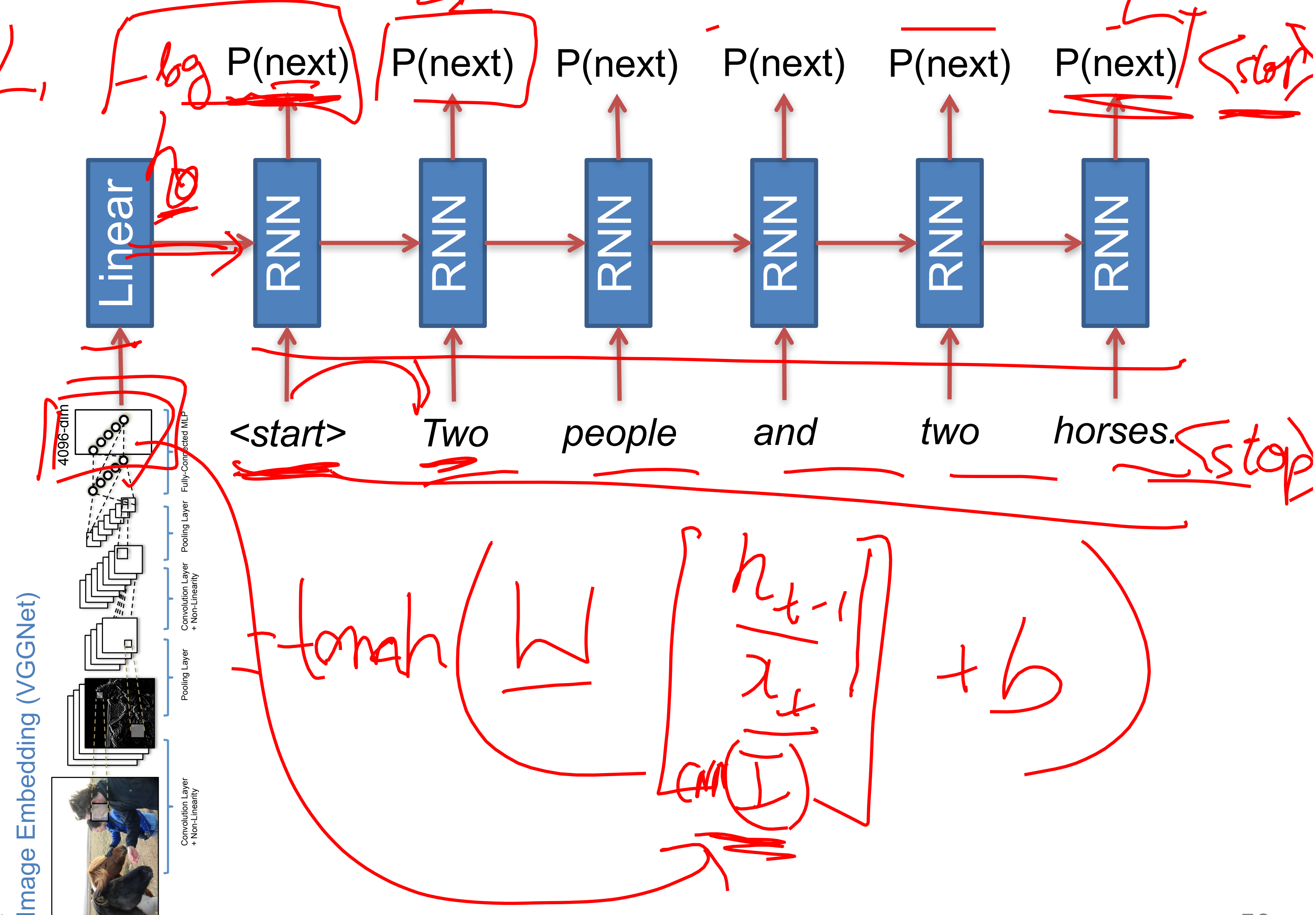


Neural Image Captioning

Image Embedding (VGGNet)

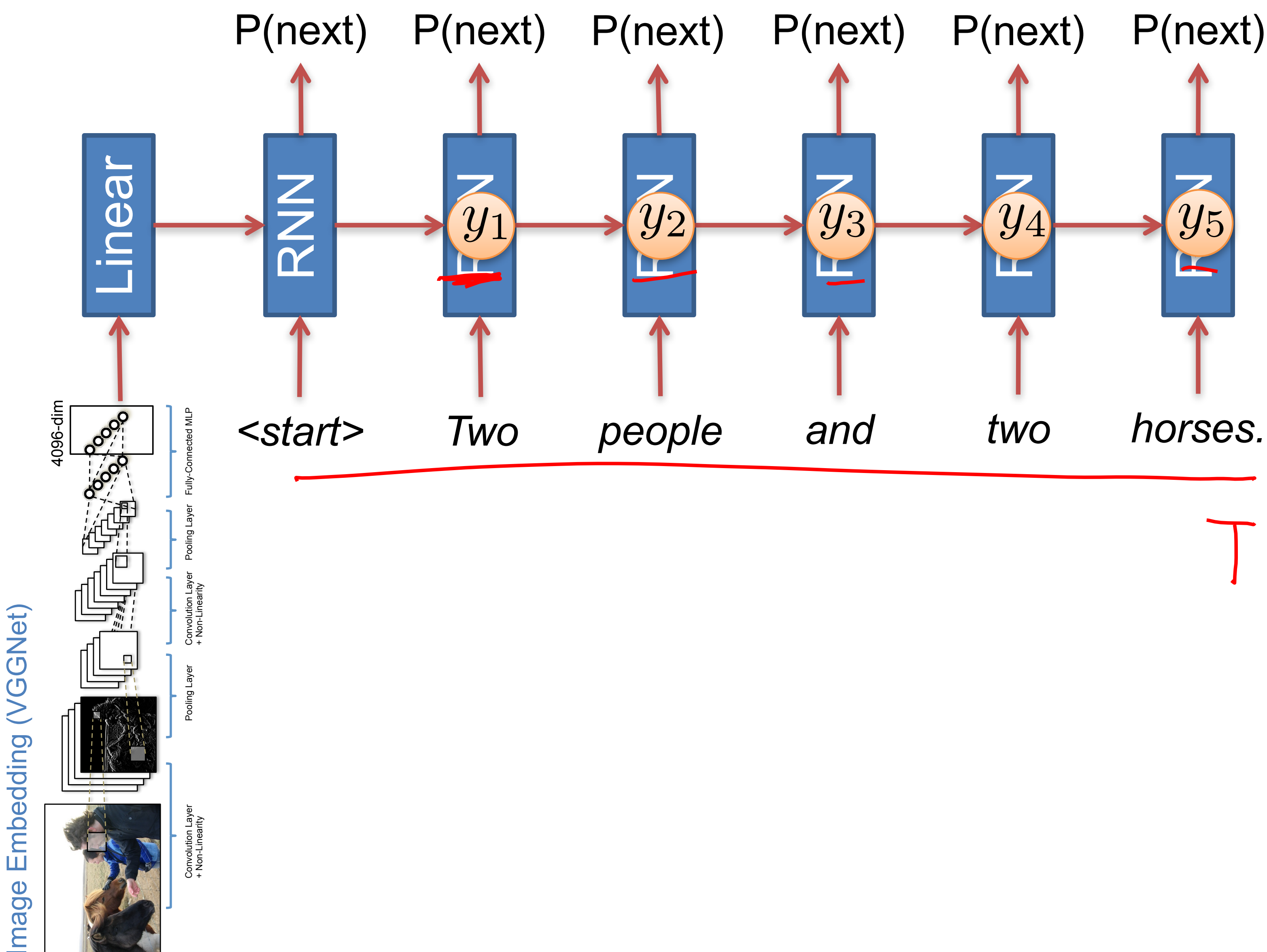


Neural Image Captioning

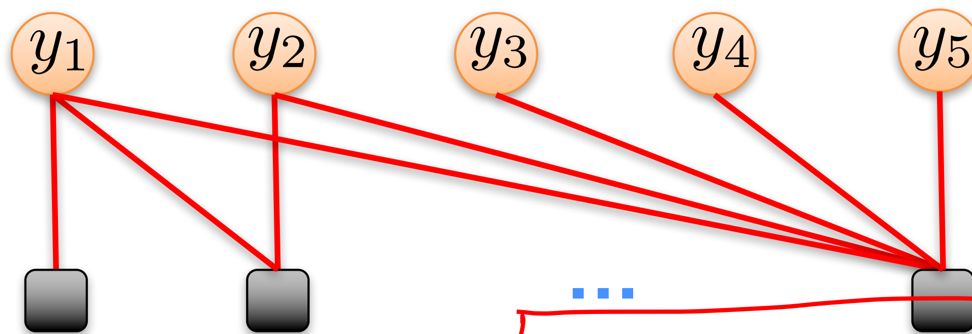


$$= \tanh\left(W \begin{bmatrix} h_{t-1} \\ x_t \\ cnn(I) \end{bmatrix} + b \right)$$

Neural Image Captioning



Sequence Model Factor Graph



$P(y_t \mid y_1, \ldots, y_{t-1})$

MAP

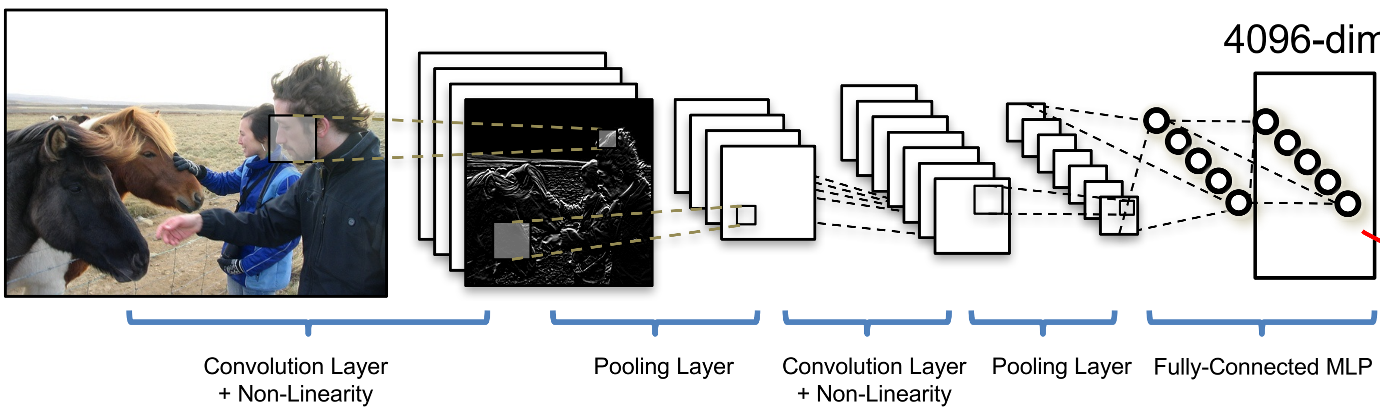
$$\underset{y_1 \ldots y_T}{\text{argmax}} \; P(y_1, \ldots, y_T \mid x, w)$$

Beam Search Demo

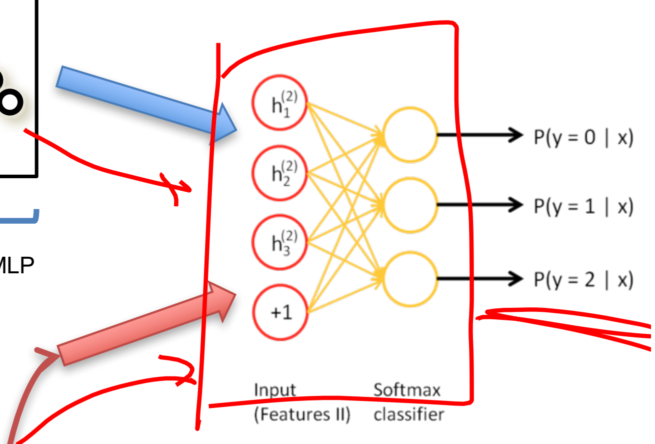
- http://dbs.cloudcv.org/captioning&mode=interactive

Typical VQA Models

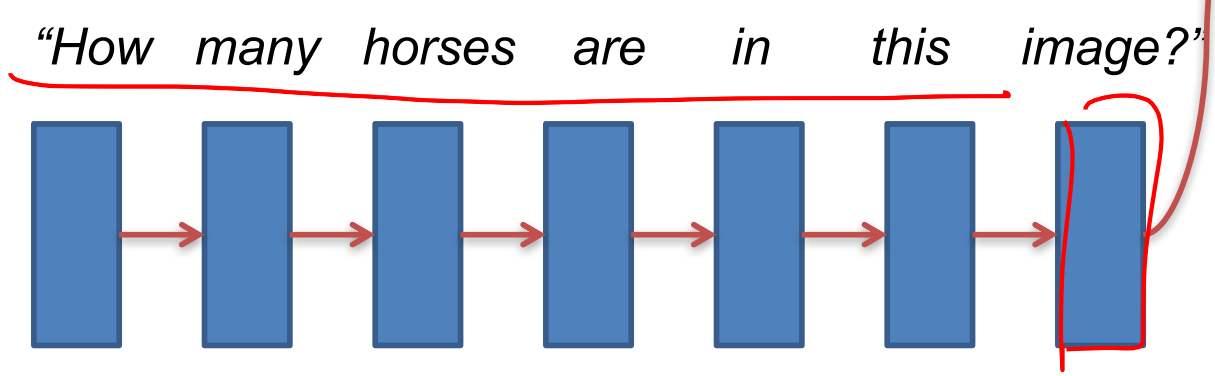
Image Embedding (VGGNet)



Neural Network Softmax over top K answers

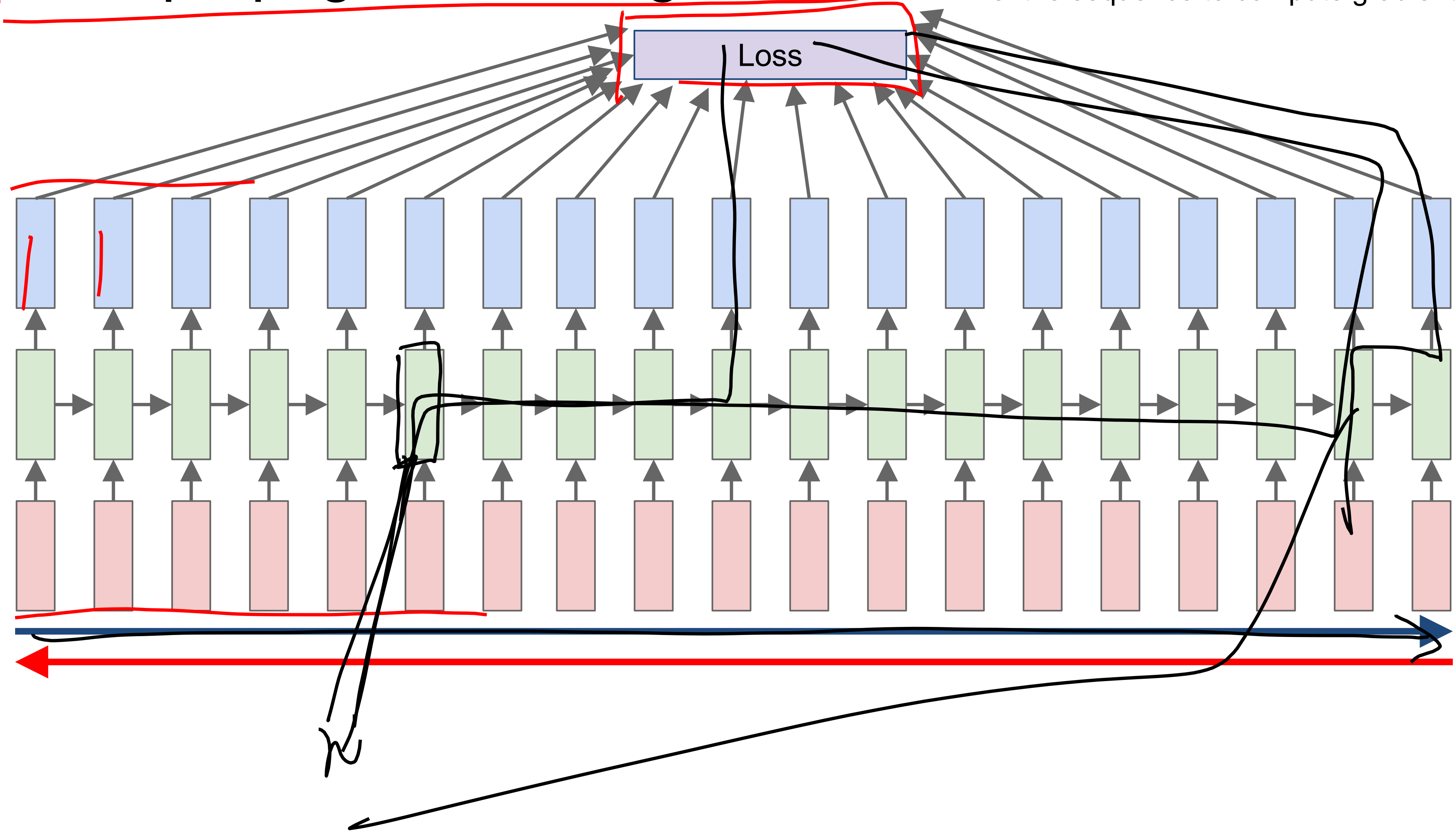


Question Embedding (LSTM)

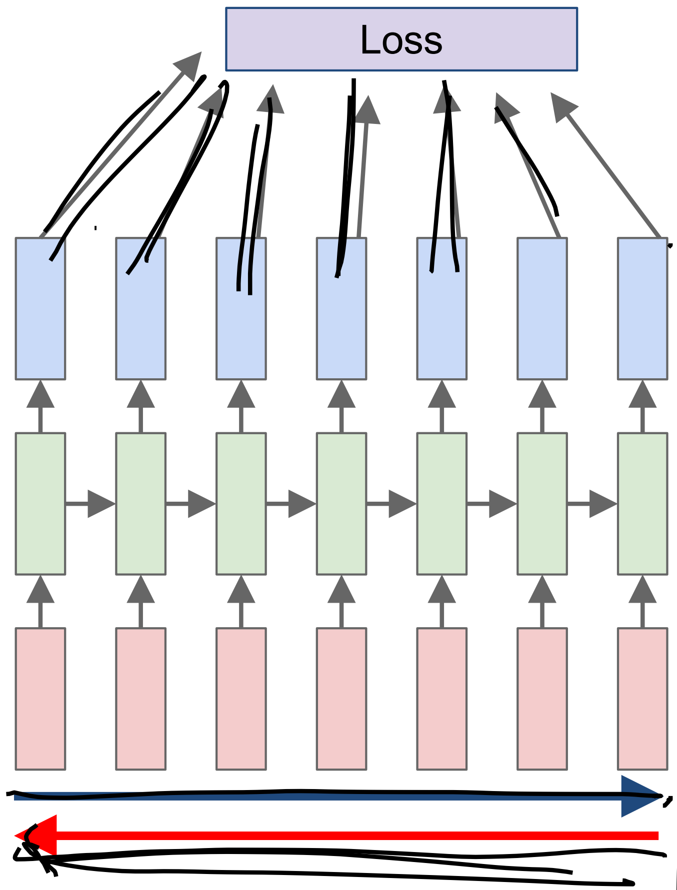


Backpropagation through time

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

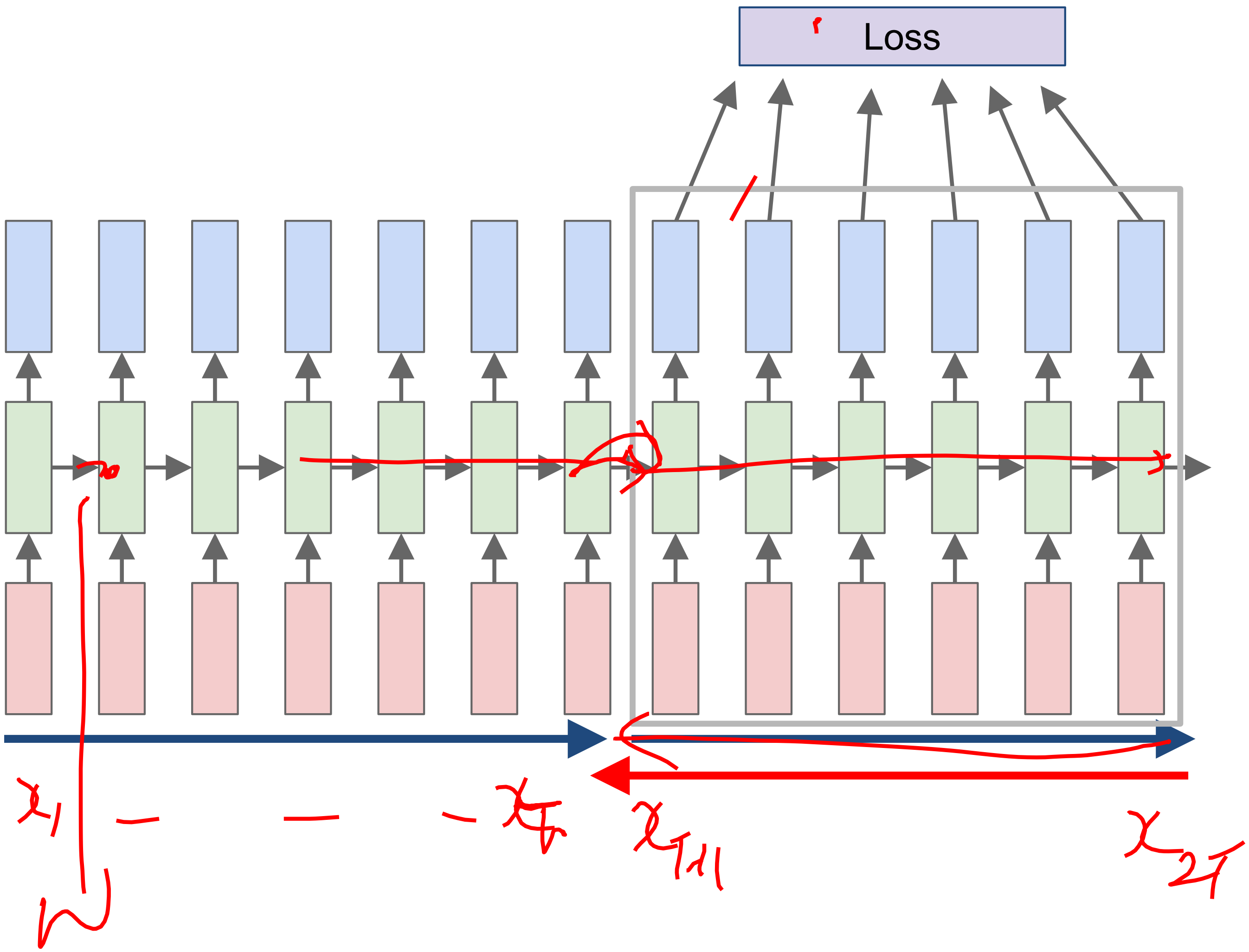


Truncated Backpropagation through time



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

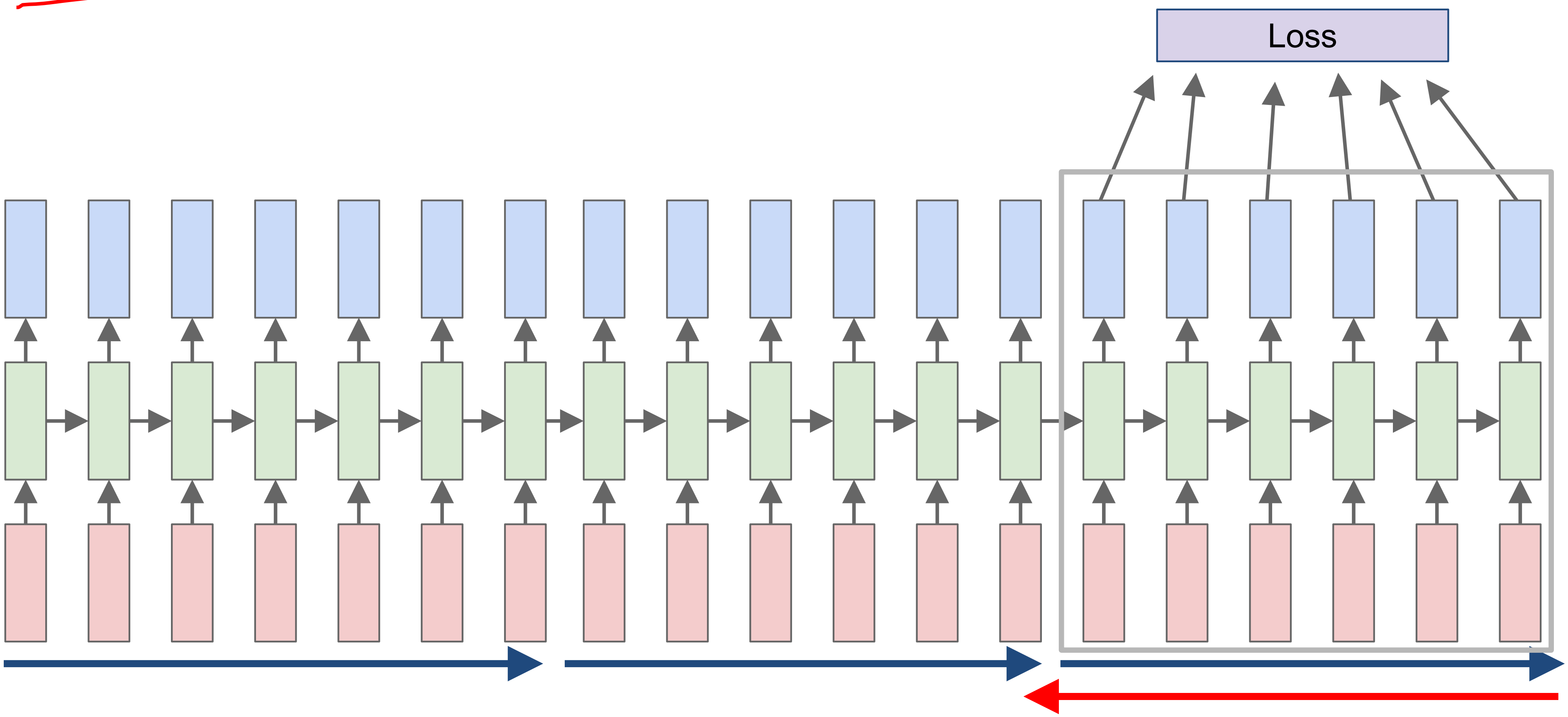
Truncated Backpropagation through time



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

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Truncated Backpropagation through time



Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

[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]

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
```

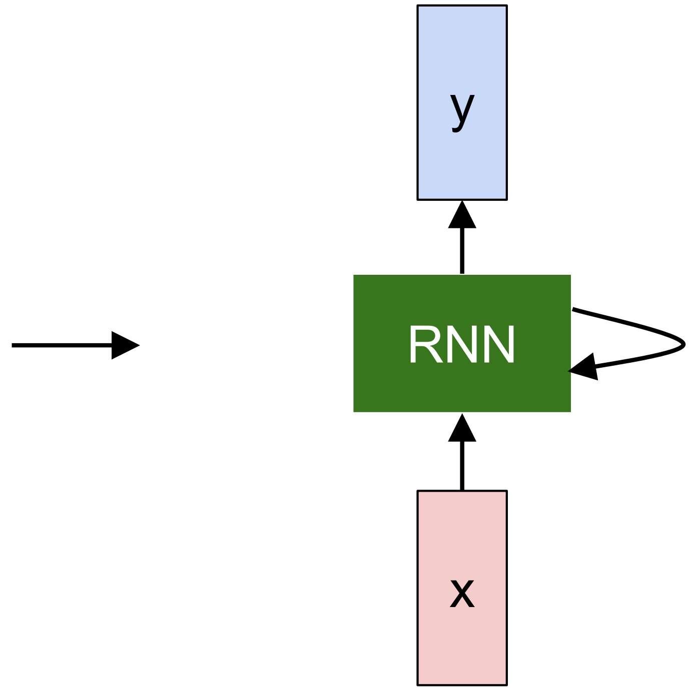
(https://gist.github.com/karpathy/d4dee566867f8291f086)

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 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.
```

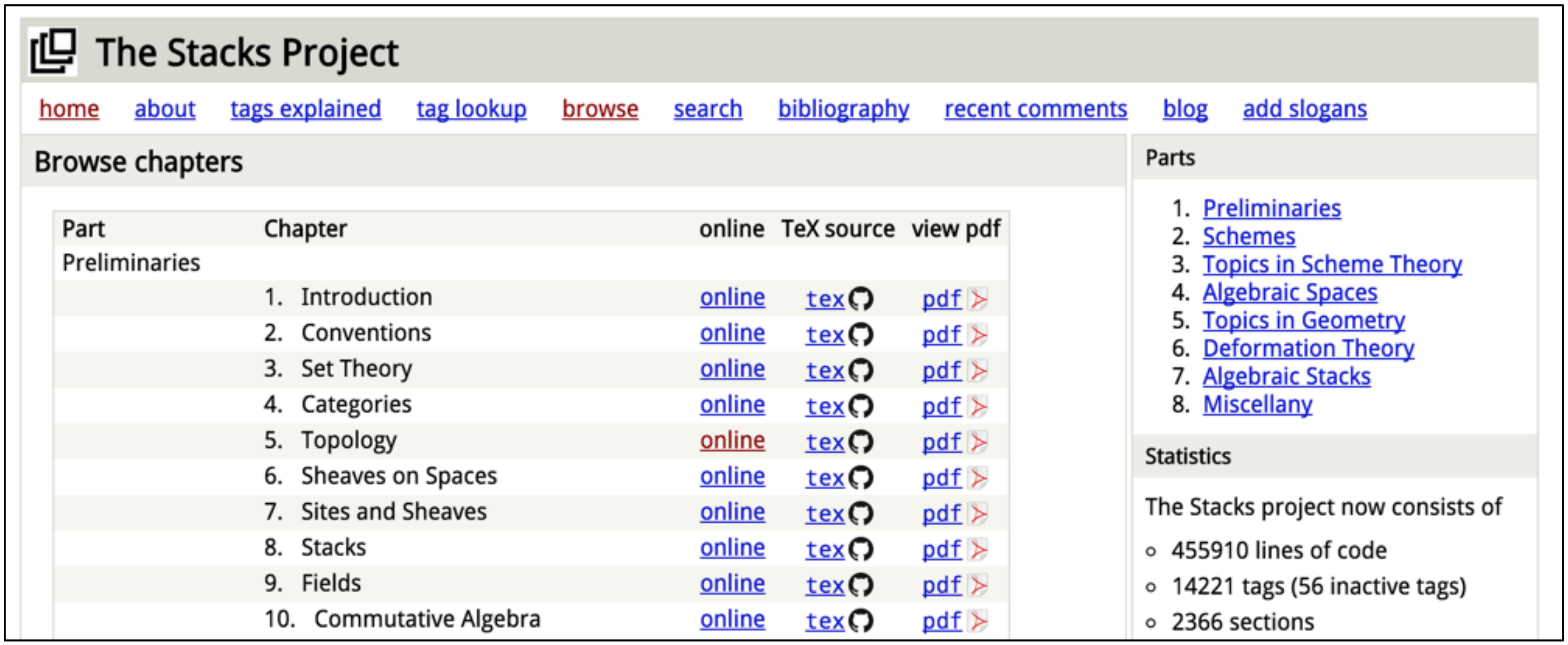
train more

```
"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.
```

The Stacks Project: open source algebraic geometry textbook



Latex source

http://stacks.math.columbia.edu/

For $\bigoplus_{n=1,\ldots,m}$ where $\mathcal{L}_{m_\bullet} = 0$, 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 $U \to T$ is a separated algebraic space.

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

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

and the comparicoly in the fibre product covering we have to prove the lemma generated by $\coprod Z \times_U U \to V$. Consider the maps M along the set of points Sch_{fppf} and $U \to U$ 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 $Sh(G)$ such that $\mathrm{Spec}(R') \to S$ 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 $\mathcal{O}_{X,x'} \to \mathcal{O}'_{X',x'}$ is separated. By Algebra, Lemma ?? we can define a map of complexes $\mathrm{GL}_{S'}(x'/S'')$ and we win. □

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

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

is a unique morphism of algebraic stacks. Note that

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

and

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

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. □

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

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

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

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

When in this case of to show that $\mathcal{Q} \to \mathcal{C}_{Z/X}$ 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 $S = \mathrm{Spec}(R)$ and $Y = \mathrm{Spec}(R)$.*

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

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

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

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

Proof. We will use the property we see that $\mathfrak{p}$ is the mext functor (??). On the other hand, by Lemma ?? we see that

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

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

Proof. Omitted. □

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

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

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

.

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

$$\mathcal{O}_X(\mathcal{F}) = \{morph_1 \times_{\mathcal{O}_X} (\mathcal{G}, \mathcal{F})\}$$

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

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

Proof. See Spaces, Lemma ??. □

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

The following to the construction of the lemma follows.

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

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

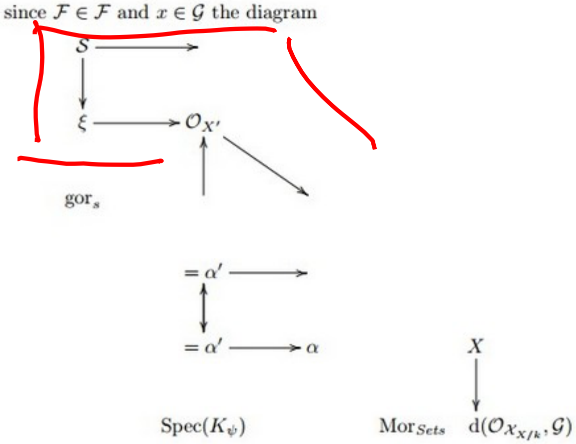
be a morphism of algebraic spaces over S and Y.

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

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

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

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



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

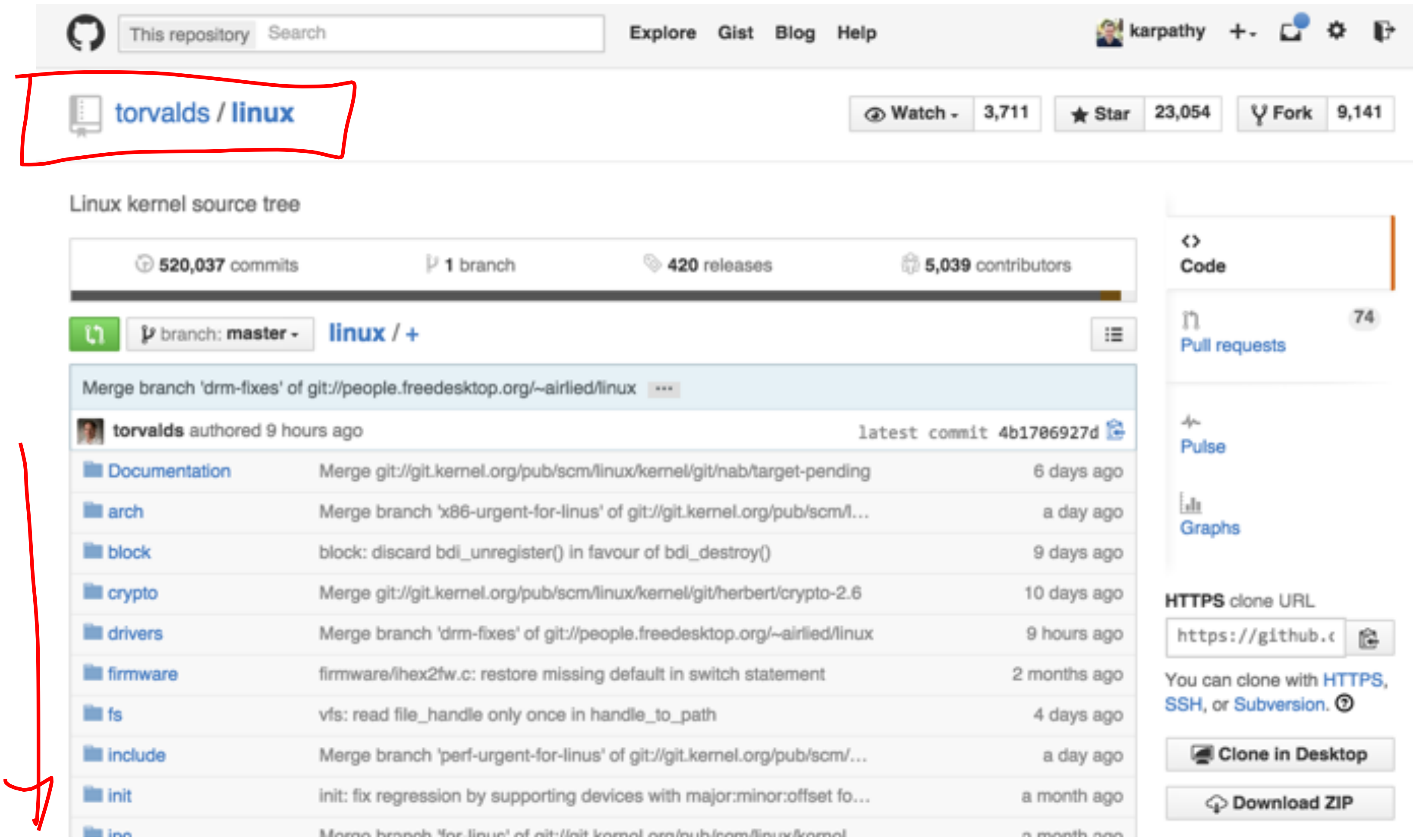
Proof. This is clear that $\mathcal{G}$ is a finite presentation, see Lemmas ??.
A *reduced above* we conclude that U is an open covering of $\mathcal{C}$. The functor $\mathcal{F}$ is a "field

$$\mathcal{O}_{X,x} \longrightarrow \mathcal{F}_{\overline{x}} \text{ -1}(\mathcal{O}_{X_{\acute{e}tale}}) \longrightarrow \mathcal{O}_{X_\ell}^{-1} \mathcal{O}_{X_\lambda}(\mathcal{O}_{X_\eta}^{\overline{v}})$$

is an isomorphism of covering of $\mathcal{O}_{X_i}$. If $\mathcal{F}$ is the unique element of $\mathcal{F}$ such that X is an isomorphism.
The property $\mathcal{F}$ is a disjoint union of Proposition ?? and we can filtered set of presentations of a scheme $\mathcal{O}_X$-algebra with $\mathcal{F}$ are opens of finite type over S.
If $\mathcal{F}$ is a scheme theoretic image points. □

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

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n



Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

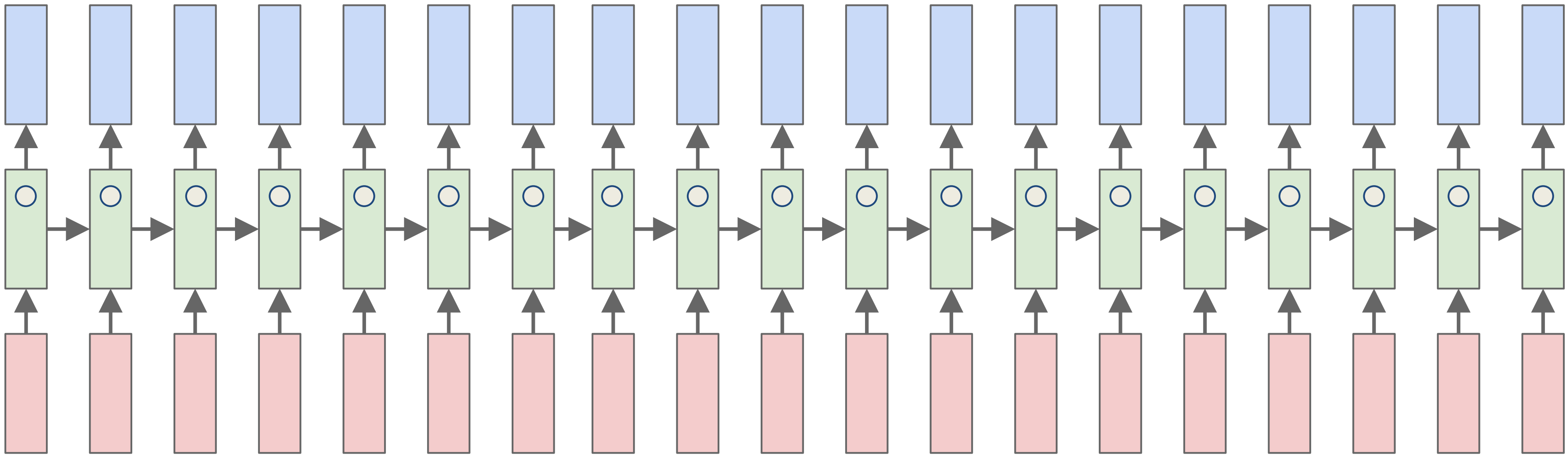
```c
static void do_command(struct seq_file *m, void *v)
{
  int column = 32 << (cmd[2] & 0x80);
  if (state)
    cmd = (int)(int_state ^ (in_8(&ch->ch_flags) & Cmd) ? 2 : 1);
  else
    seq = 1;
  for (i = 0; i < 16; i++) {
    if (k & (1 << 1))
      pipe = (in_use & UMXTHREAD_UNCCA) +
        ((count & 0x00000000fffffff8) & 0x000000f) << 8;
    if (count == 0)
      sub(pid, ppc_md.kexec_handle, 0x20000000);
    pipe_set_bytes(i, 0);
  }
  /* Free our user pages pointer to place camera if all dash */
  subsystem_info = &of_changes[PAGE_SIZE];
  rek_controls(offset, idx, &soffset);
  /* Now we want to deliberately put it to device */
  control_check_polarity(&context, val, 0);
  for (i = 0; i < COUNTER; i++)
    seq_puts(s, "policy ");
}
```

Generated C code

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

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

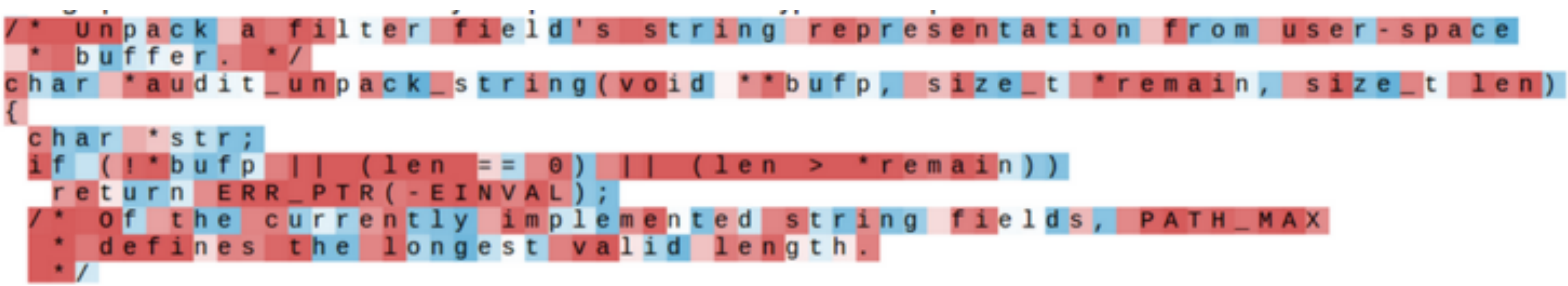
```
#include <asm/io.h>
#include <asm/prom.h>
#include <asm/e820.h>
#include <asm/system_info.h>
#include <asm/setew.h>
#include <asm/pgproto.h>

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

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

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

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

Searching for interpretable cells



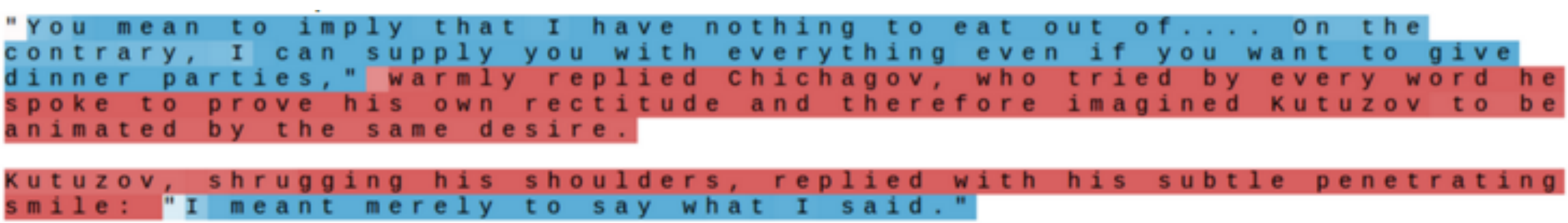
Karpathy, Johnson, and Fei-Fei: Visualizing and Understanding Recurrent Networks, ICLR Workshop 2016

Searching for interpretable cells



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

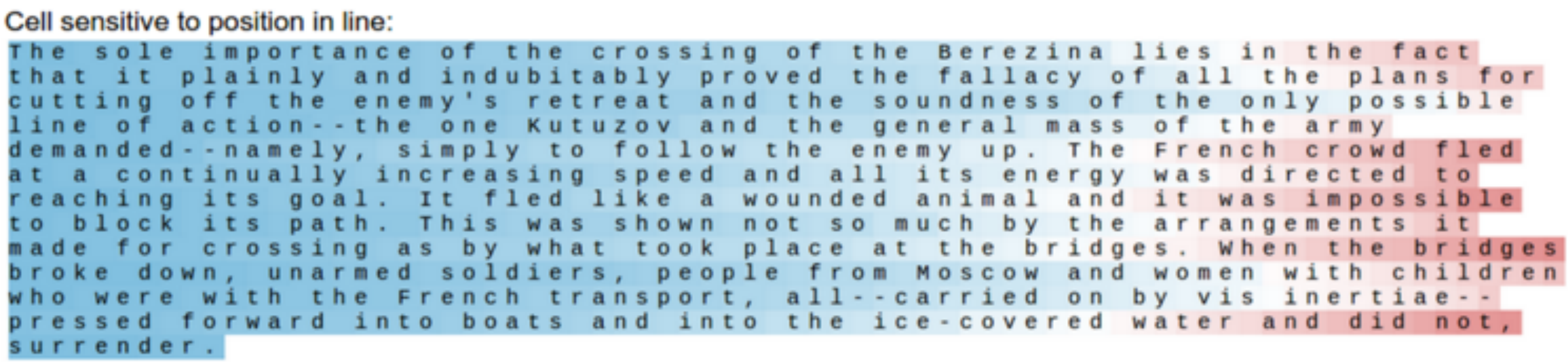
Searching for interpretable cells



quote detection cell

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

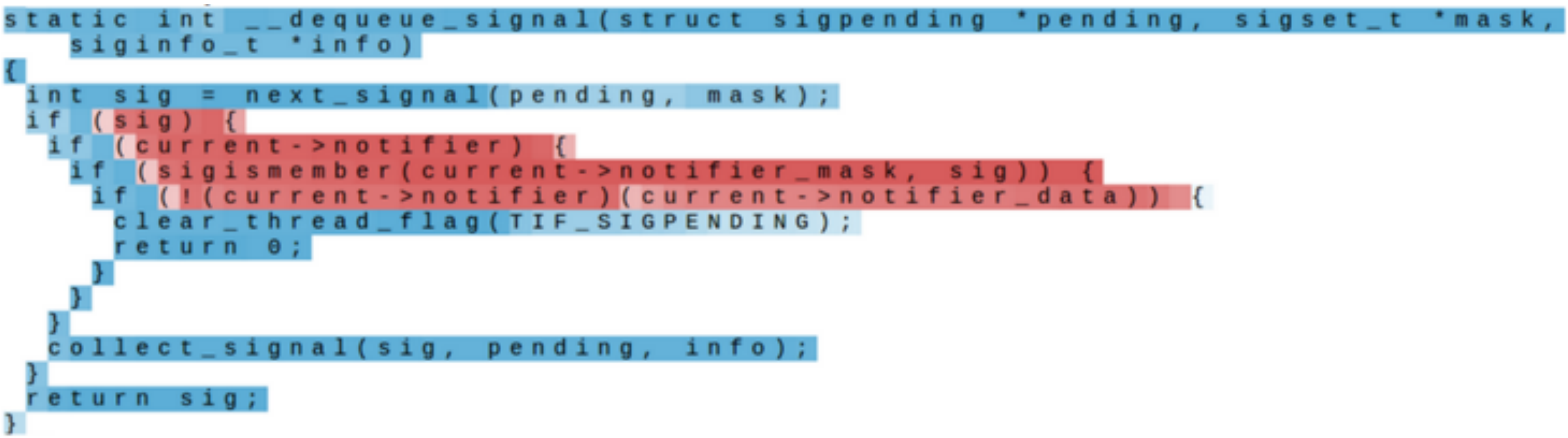
Searching for interpretable cells



line length tracking cell

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

Searching for interpretable cells



if statement cell

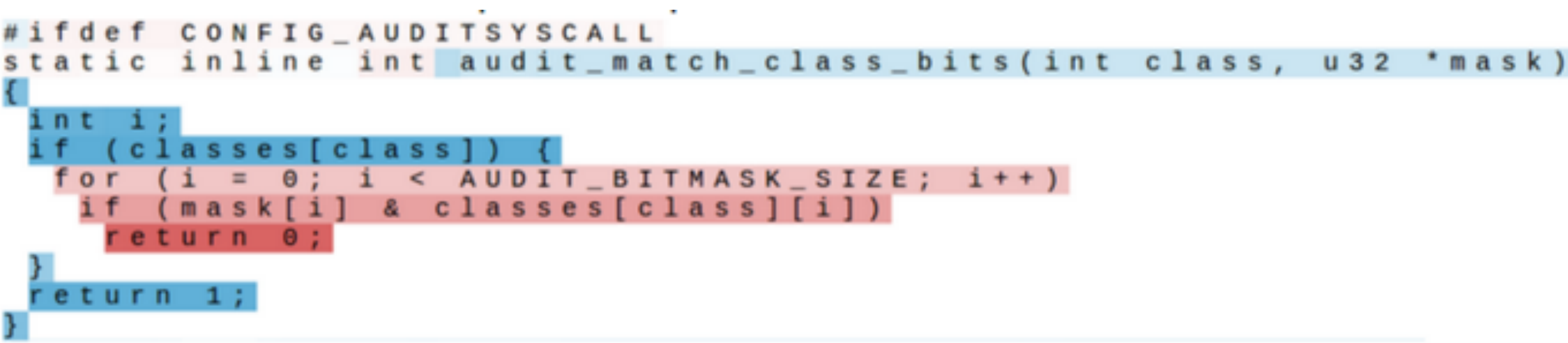
Karpathy, Johnson, and Fei-Fei: Visualizing and Understanding Recurrent Networks, ICLR Workshop 2016

Searching for interpretable cells

Cell that turns on inside comments and quotes:

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

Searching for interpretable cells



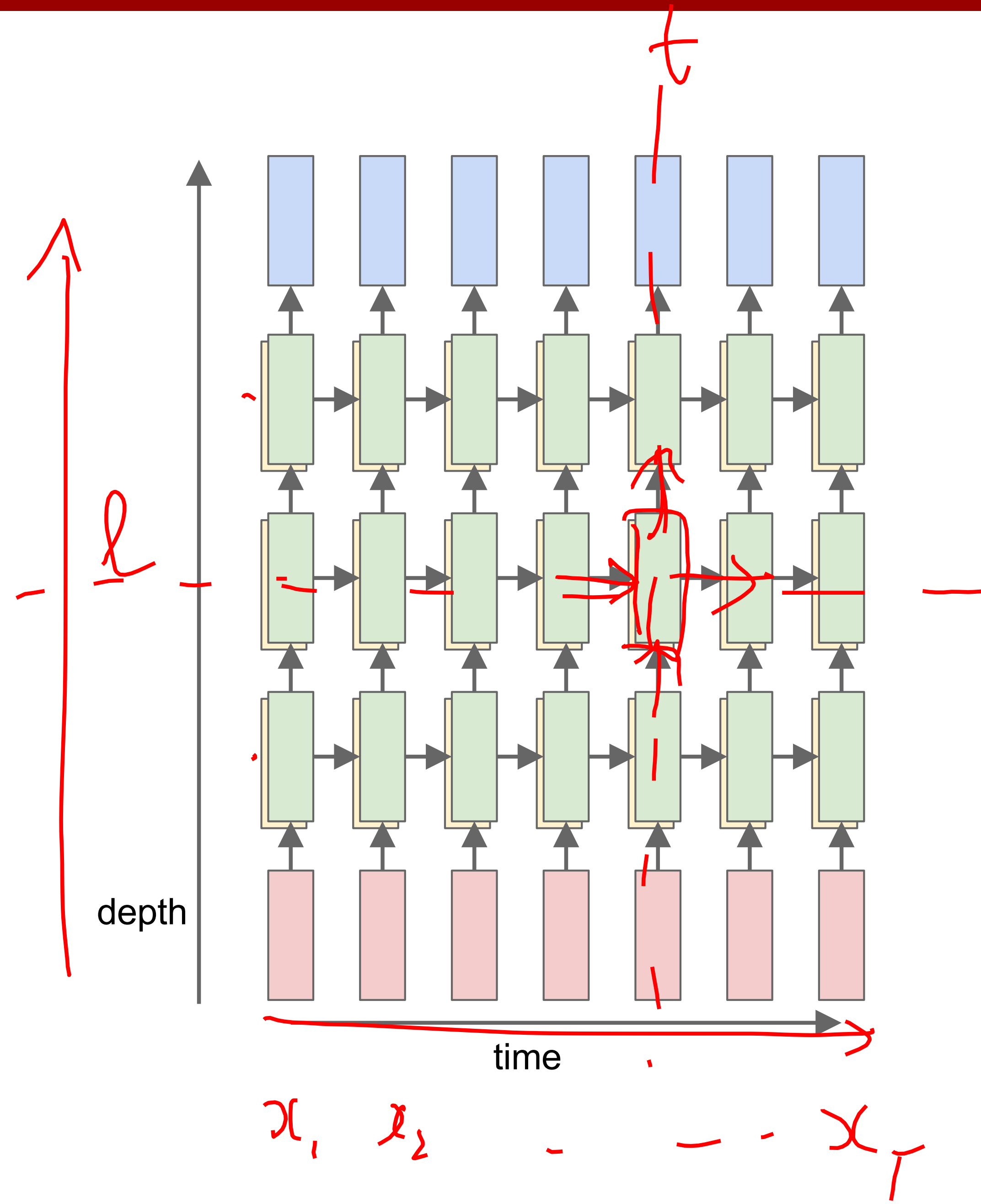
code depth cell

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

Multilayer RNNs

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

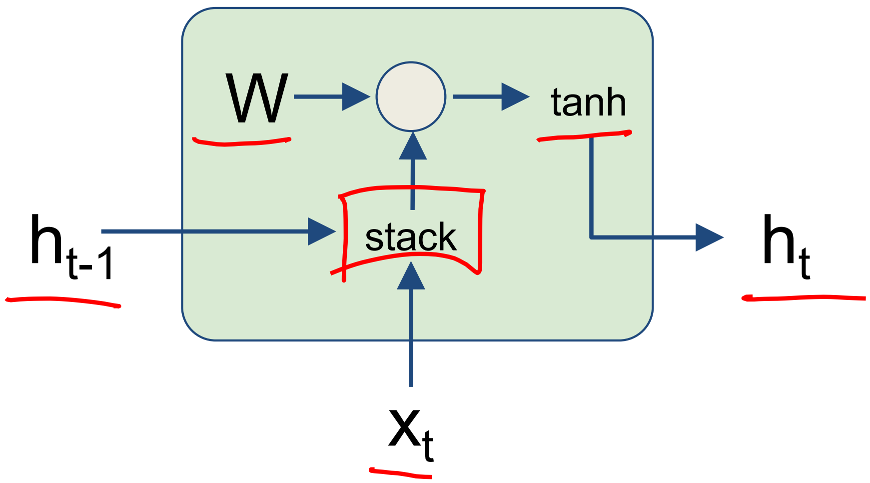
$h \in \mathbb{R}^n$ $\quad W^l \; [n \times 2n]$



Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

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



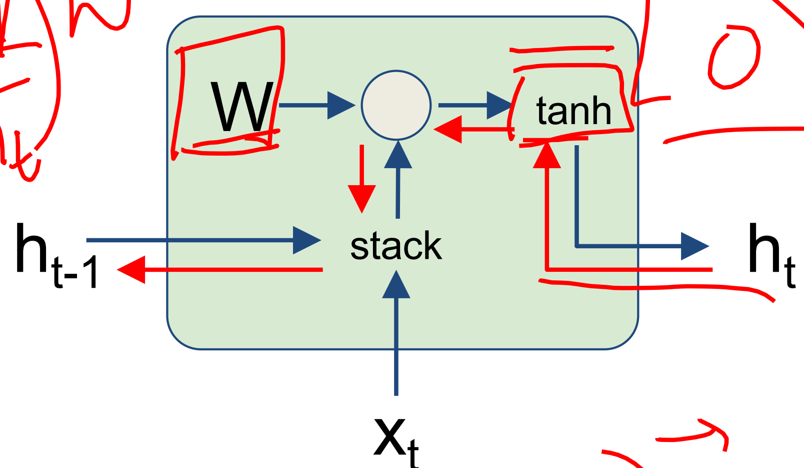
$$
\begin{aligned}
h_t &= \tanh(W_{hh}h_{t-1} + W_{xh}x_t) \\
&= \tanh\left(\begin{pmatrix}W_{hh} & W_{hx}\end{pmatrix}\begin{pmatrix}h_{t-1} \\ x_t\end{pmatrix}\right) \\
&= \tanh\left(W\begin{pmatrix}h_{t-1} \\ x_t\end{pmatrix}\right)
\end{aligned}
$$

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)

$\left(\frac{\partial L}{\partial h_t}\right)^T W$

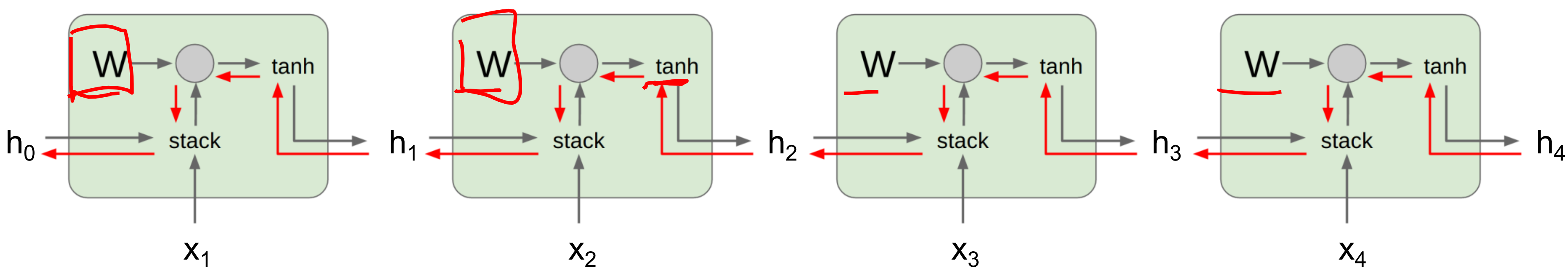


$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$
$$= \tanh\left(\begin{pmatrix} W_{hh} & W_{hx} \end{pmatrix} \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$
$$= \tanh\left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$

$\frac{\partial L}{\partial \vec{h}_{t-1}} = \left(\frac{\partial L}{\partial h_t}\right)^T W D \qquad \frac{\partial \vec{h}_t}{\partial \vec{h}_{t-1}}$

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)

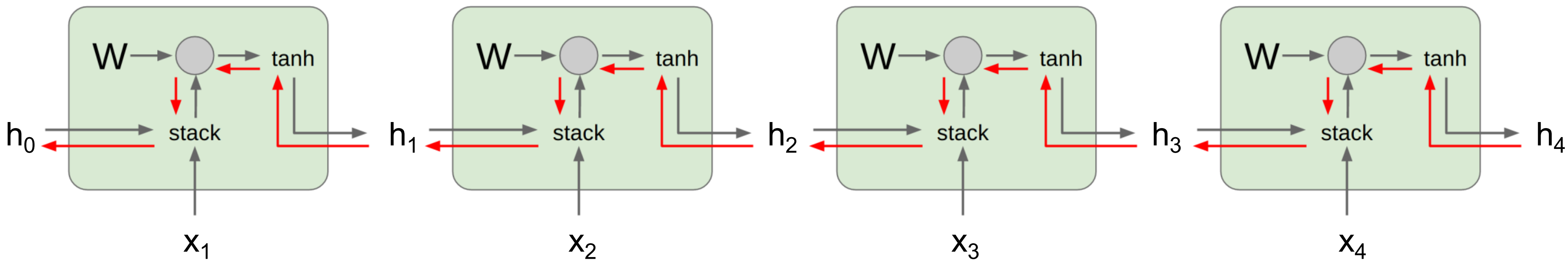
$$\frac{\partial h_4}{\partial h_0} = \frac{\partial h_4}{\partial h_3} \cdots \frac{\partial h_1}{\partial h_0}$$

$$W^4 = Q \Lambda^4 Q^T$$

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

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



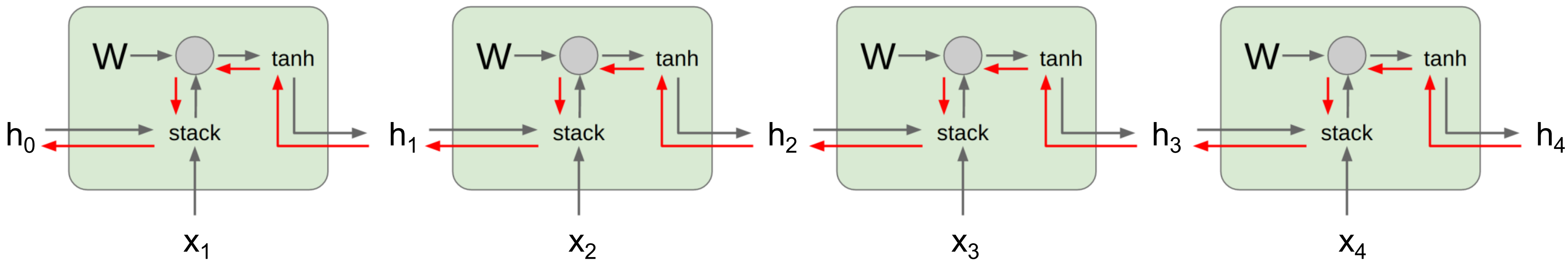
Computing gradient of h_0 involves many factors of W (and repeated tanh)

Largest singular value > 1:
Exploding gradients

Largest singular value < 1:
Vanishing gradients

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

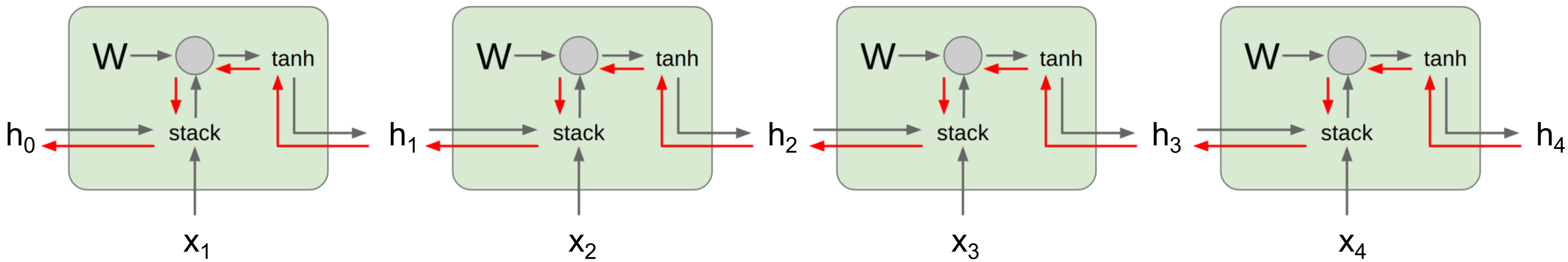
Gradient clipping: Scale gradient if its norm is too big

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

Largest singular value < 1:
Vanishing 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

→ Change RNN architecture

Long Short Term Memory (LSTM)

Vanilla RNN

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

LSTM

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

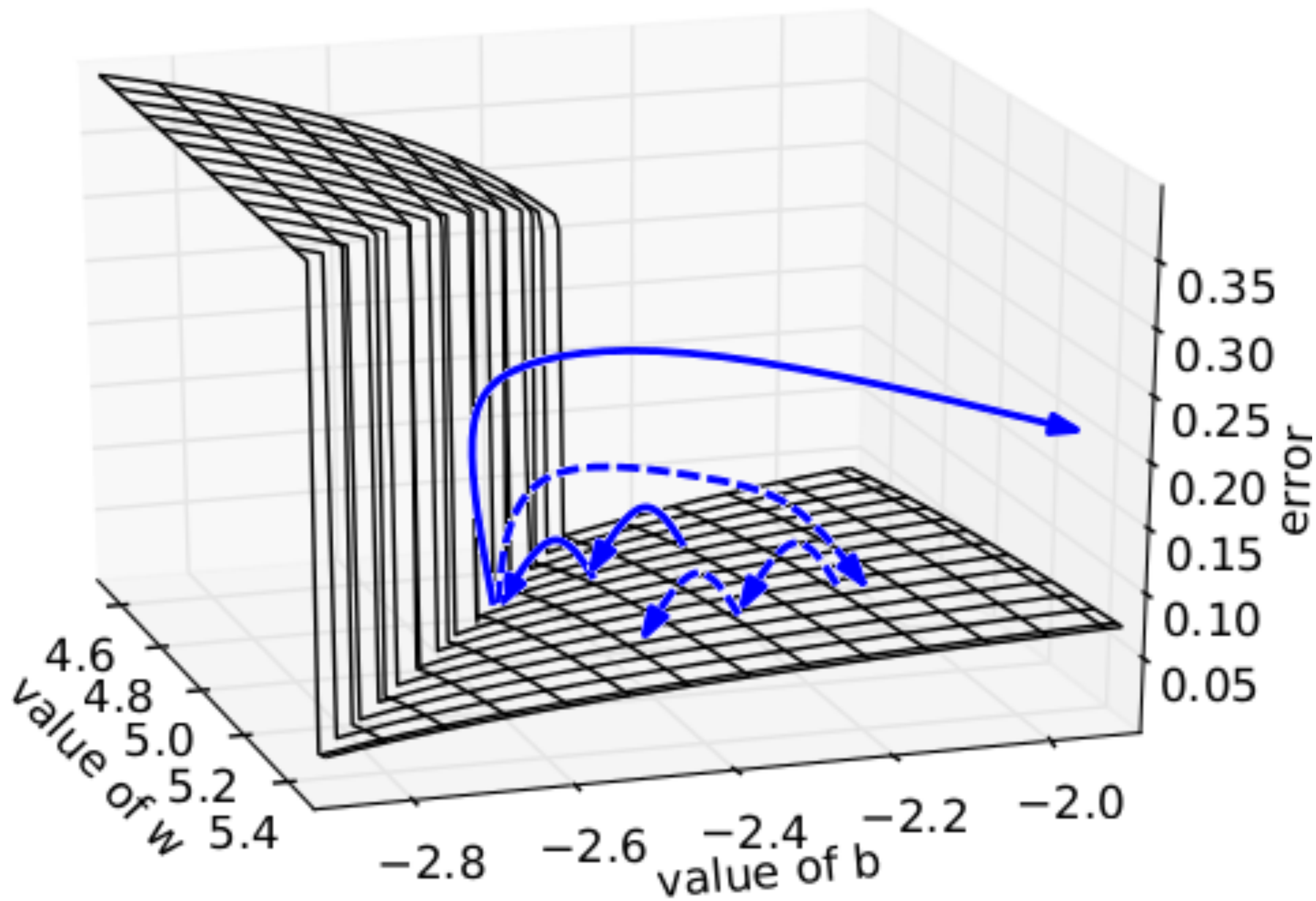
$$c_t = f \odot c_{t-1} + i \odot g$$

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

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

Illustration [Pascanu et al]

- Intuition
  - Error surface of a single hidden unit RNN; High curvature walls
  - Solid lines: standard gradient descent trajectories
  - Dashed lines: gradient rescaled to fix problem



Fix #1

- Pseudocode

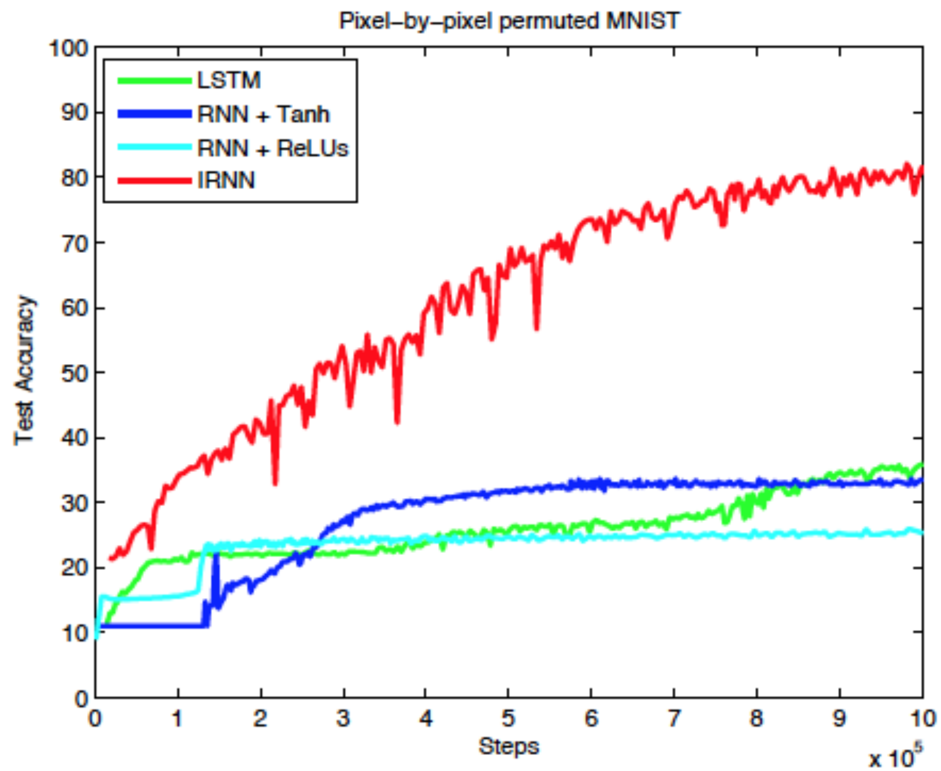
Algorithm 1 Pseudo-code for norm clipping the gradients whenever they explode

$\hat{\mathbf{g}} \leftarrow \frac{\partial \mathcal{E}}{\partial \theta}$
if $\|\hat{\mathbf{g}}\| \geq threshold$ **then**
  $\hat{\mathbf{g}} \leftarrow \frac{threshold}{\|\hat{\mathbf{g}}\|} \hat{\mathbf{g}}$
end if

Image Credit: Richard Socher

Fix #2

- Smart Initialization and ReLus
  – [Socher et al 2013]
  – *A Simple Way to Initialize Recurrent Networks of Rectified Linear* Units, Le et al. 2015



Meet LSTMs

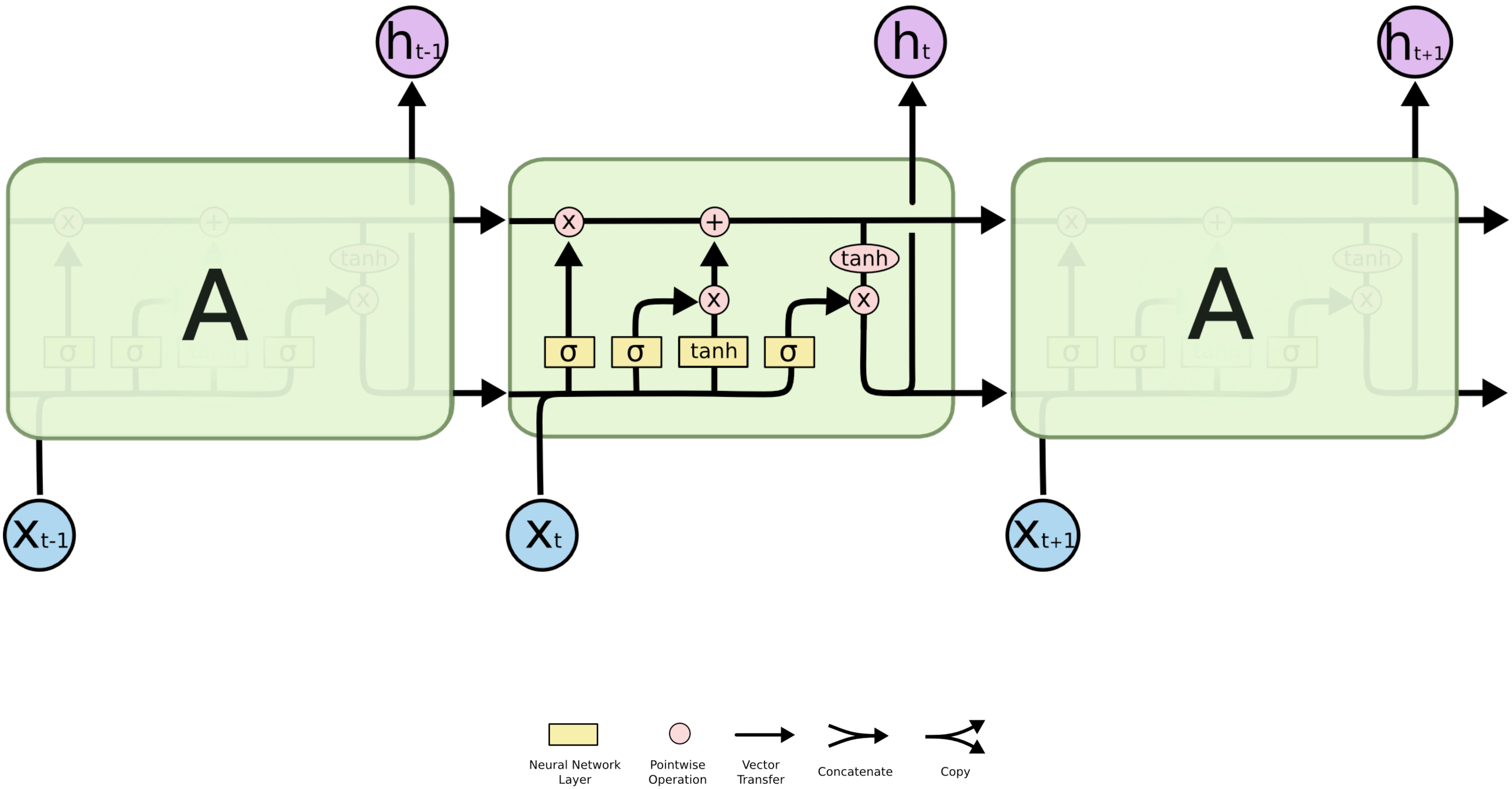


 Image Credit: Christopher Olah (http://colah.github.io/posts/2015-08-Understanding-LSTMs/)

LSTMs Intuition: Memory

- Cell State / Memory

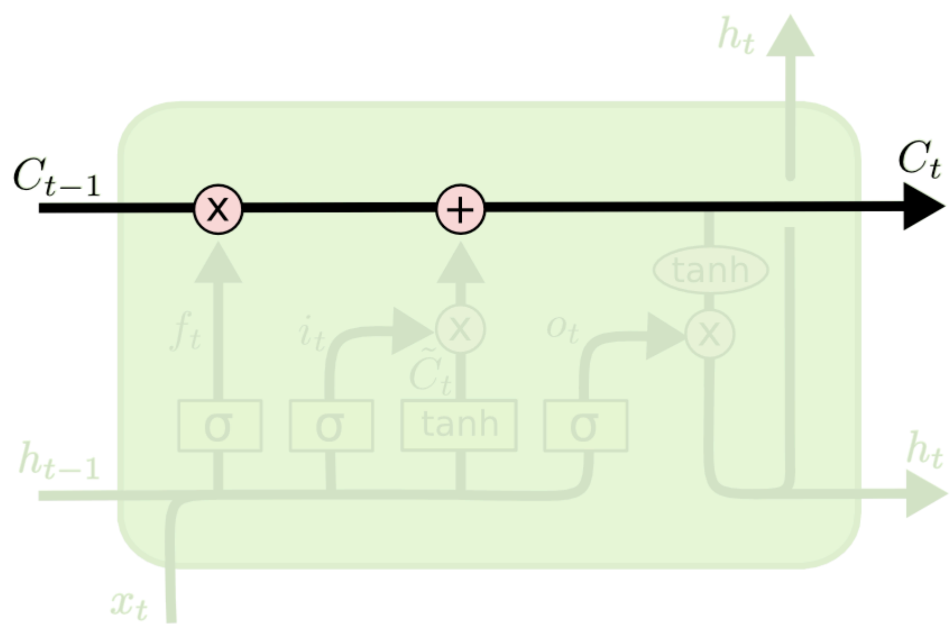
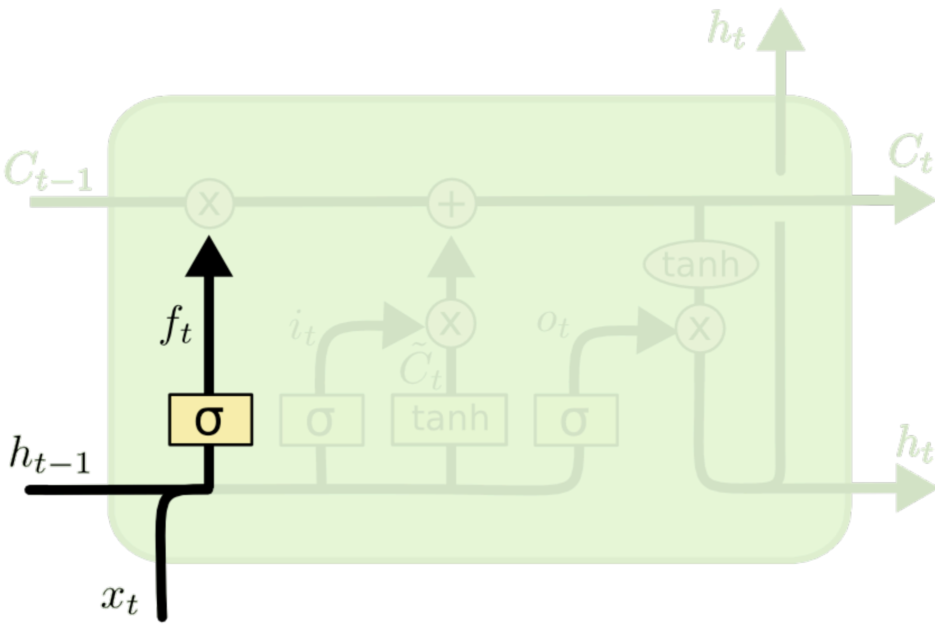


 Image Credit: Christopher Olah (http://colah.github.io/posts/2015-08-Understanding-LSTMs/)

LSTMs Intuition: Forget Gate

- Should we continue to remember this “bit” of information or not?

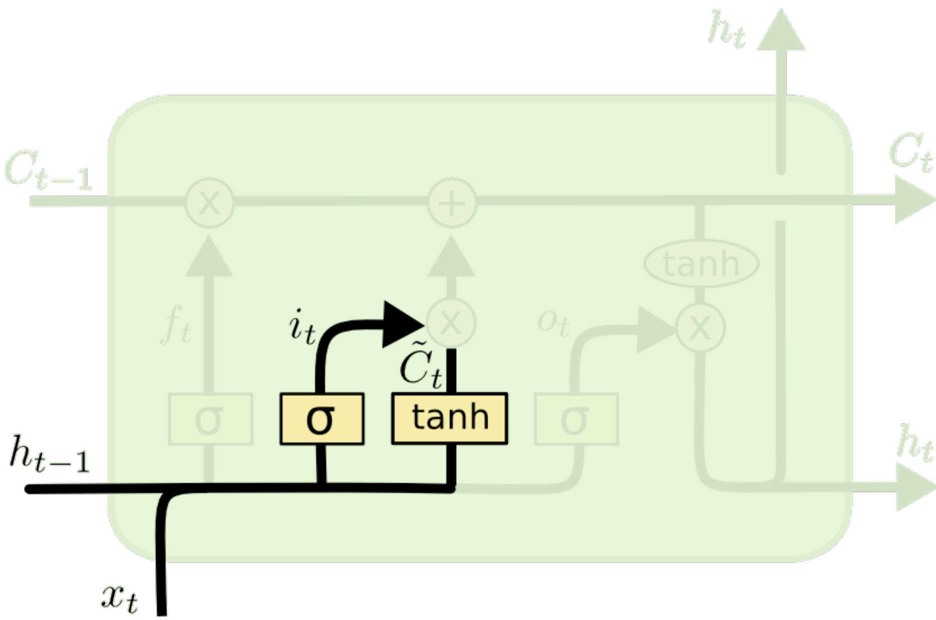


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

 Image Credit: Christopher Olah (http://colah.github.io/posts/2015-08-Understanding-LSTMs/)

LSTMs Intuition: Input Gate

- Should we update this “bit” of information or not?
  - If so, with what?

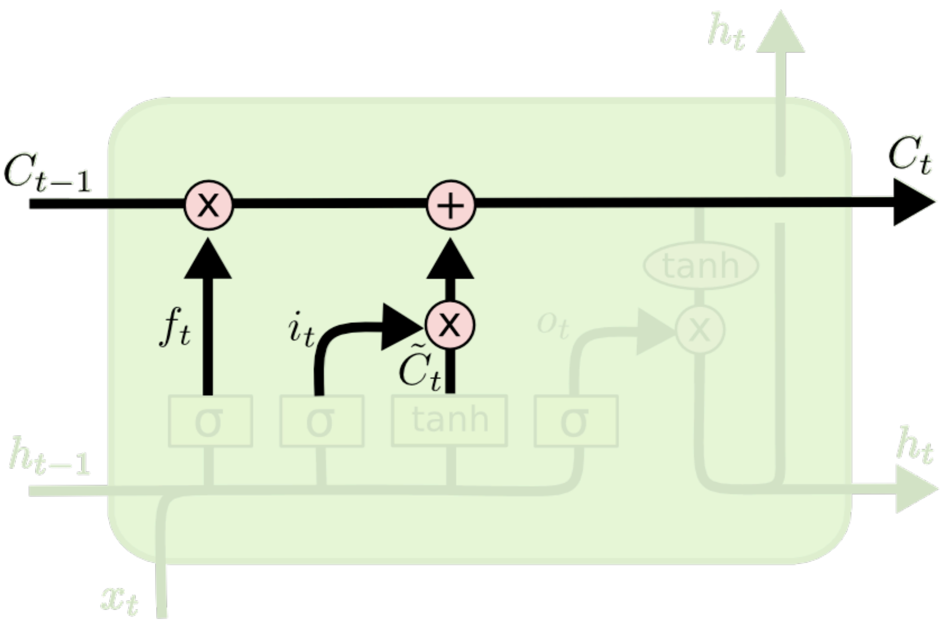


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

LSTMs Intuition: Memory Update

- Forget that + memorize this

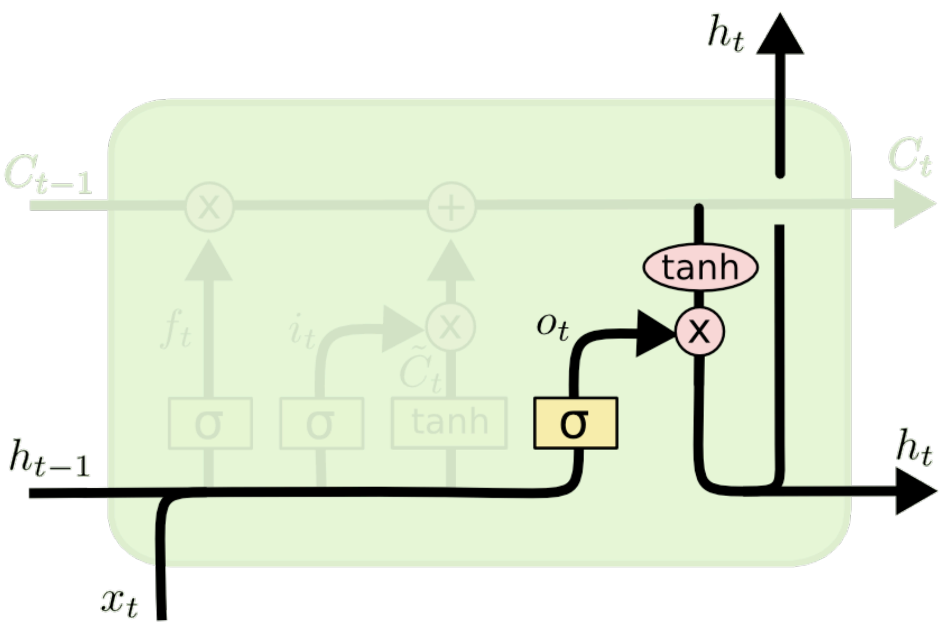


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

 Image Credit: Christopher Olah (http://colah.github.io/posts/2015-08-Understanding-LSTMs/)

LSTMs Intuition: Output Gate

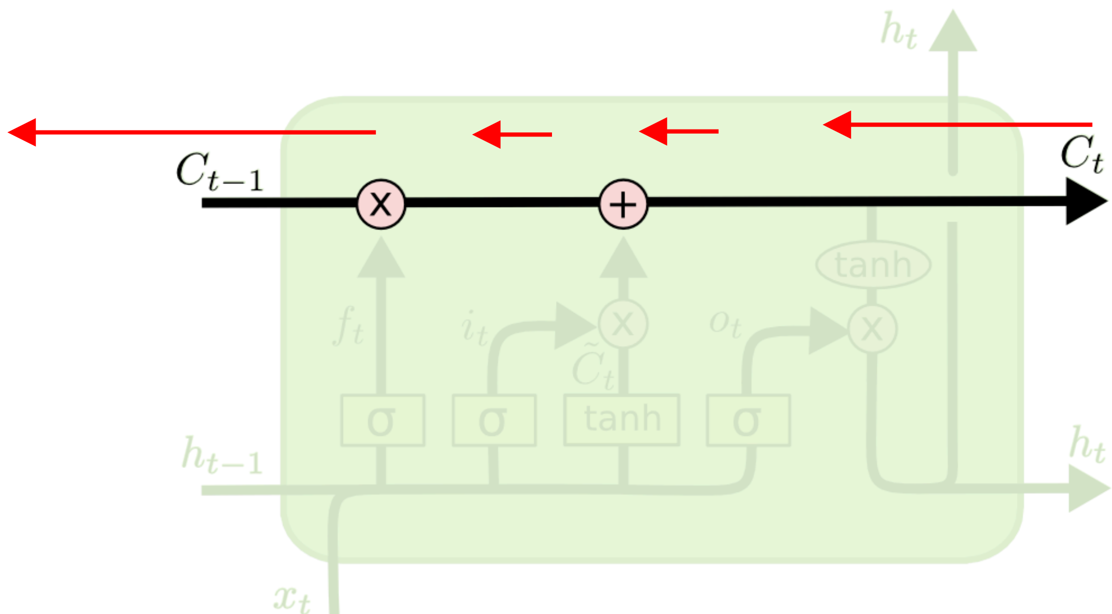
- Should we output this "bit" of information to "deeper" layers?



$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

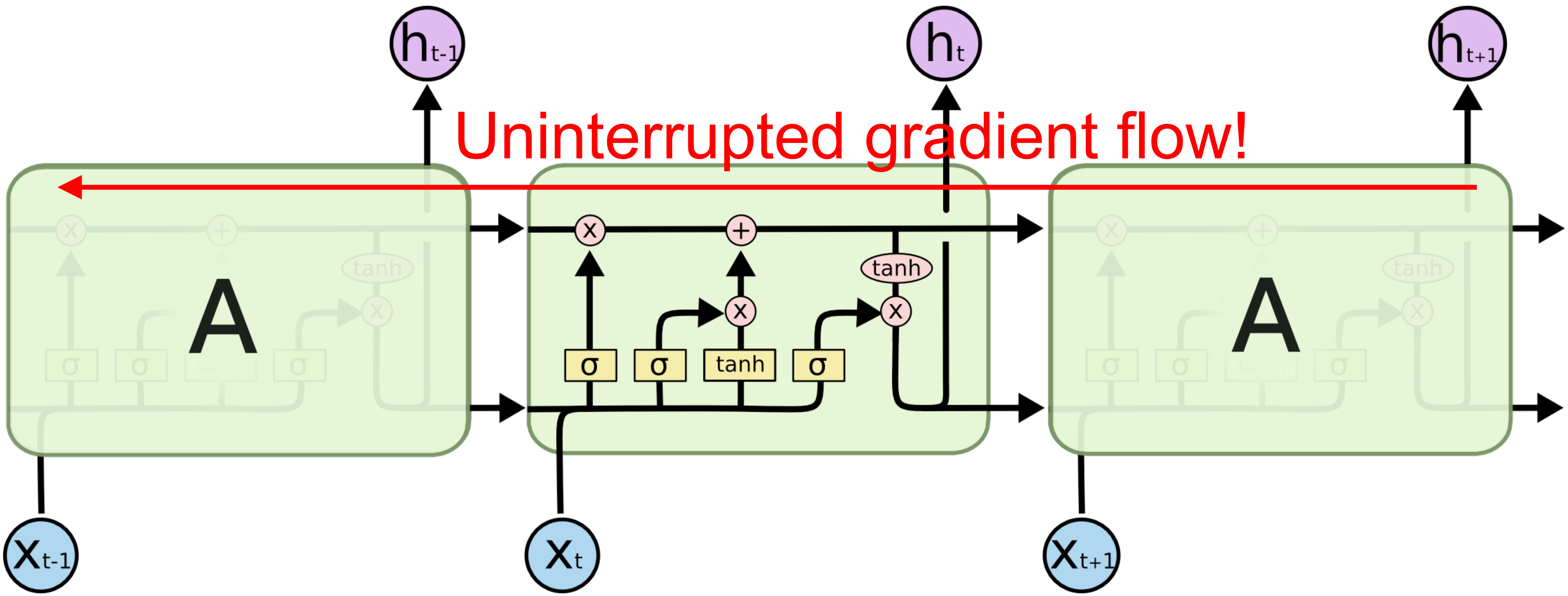
LSTMs Intuition: Additive Updates



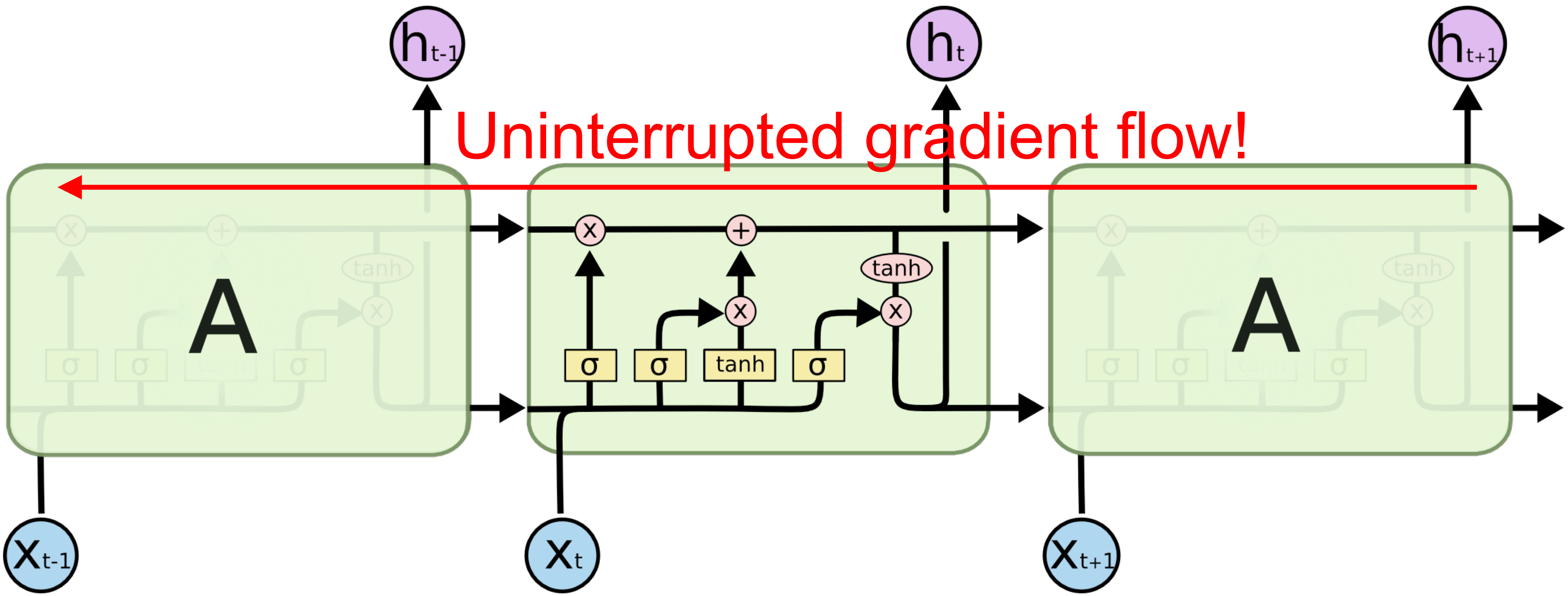
Backpropagation from c_t to c_{t-1} only elementwise multiplication by f, no matrix multiply by W

 Image Credit: Christopher Olah (http://colah.github.io/posts/2015-08-Understanding-LSTMs/)

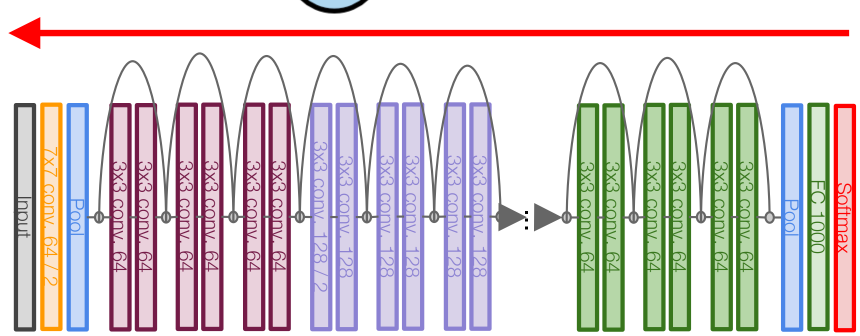
LSTMs Intuition: Additive Updates



LSTMs Intuition: Additive Updates

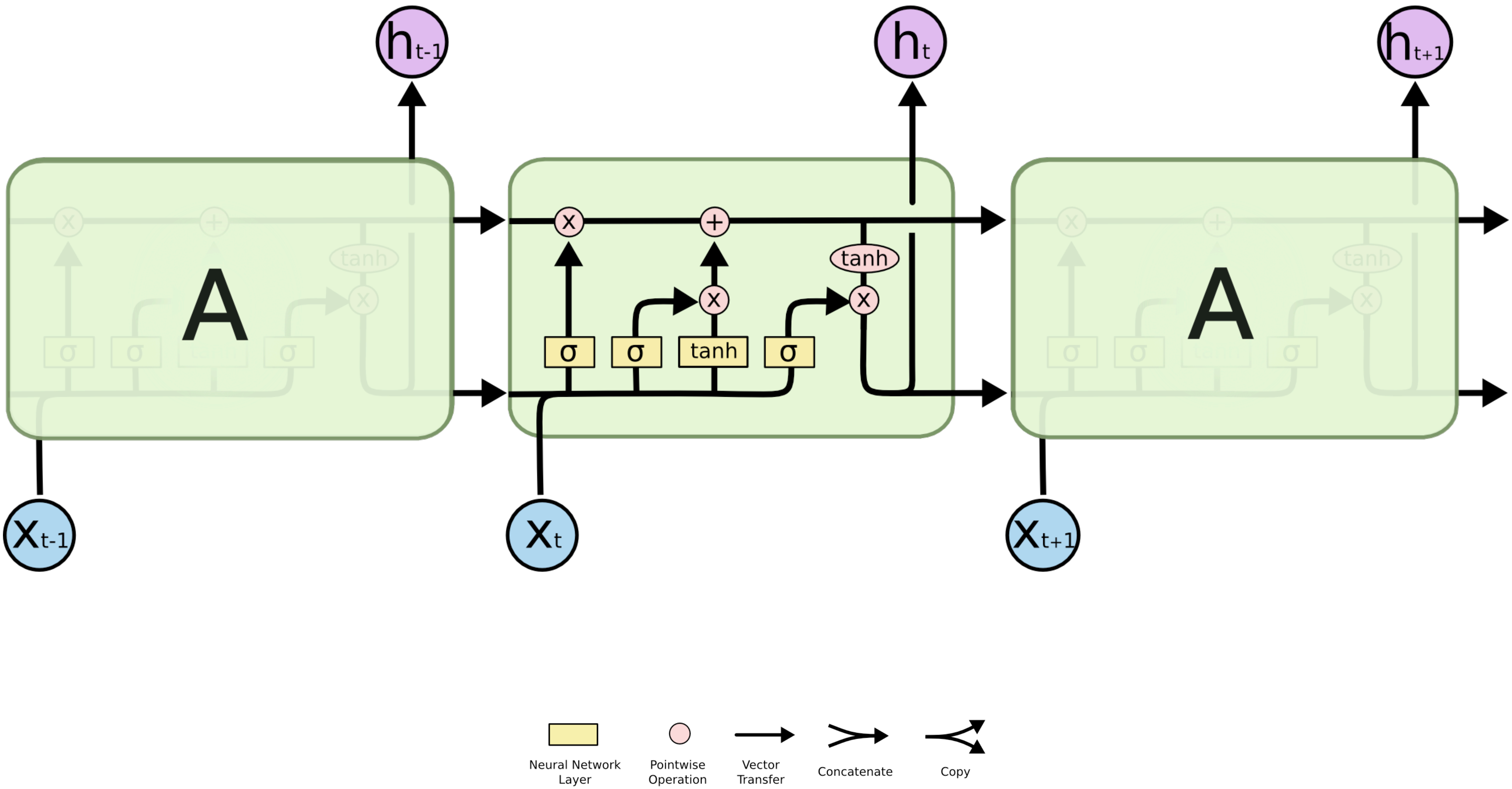


Similar to ResNet!



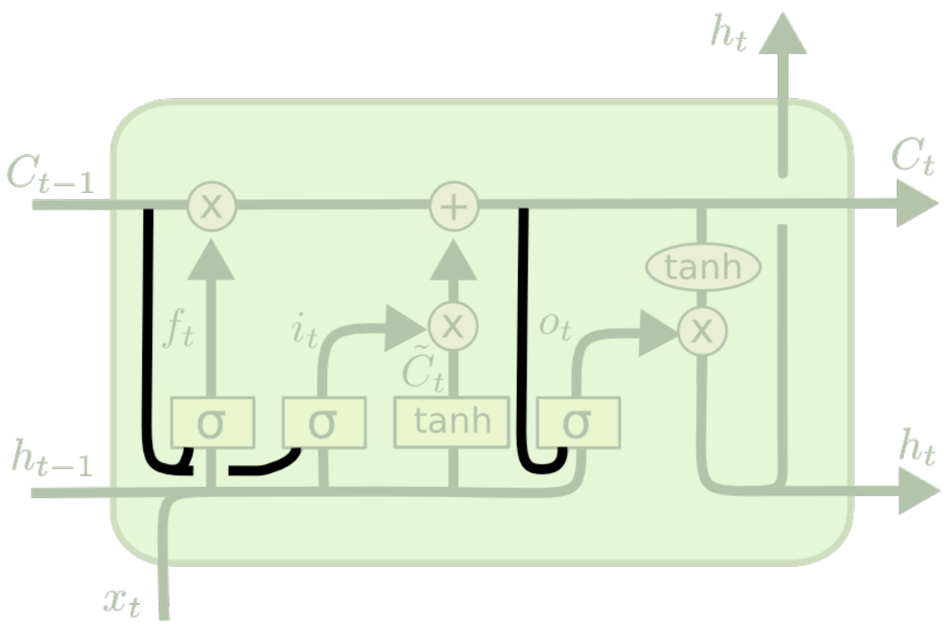
LSTMs

- A pretty sophisticated cell



LSTM Variants #1: Peephole Connections

- Let gates see the cell state / memory

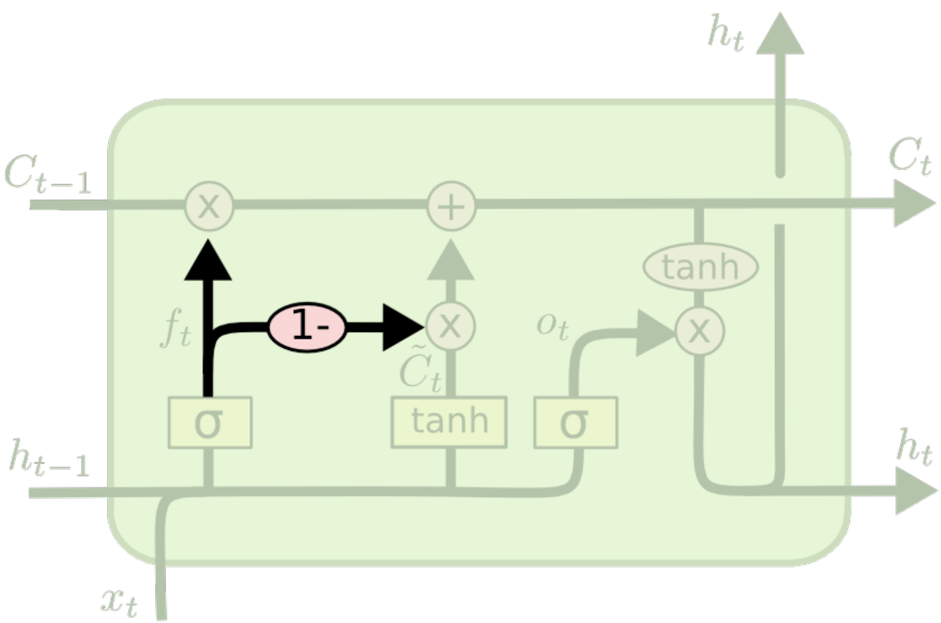


$$f_t = \sigma\left(W_f \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \ + \ b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \ + \ b_i\right)$$
$$o_t = \sigma\left(W_o \cdot [\boldsymbol{C_t}, h_{t-1}, x_t] \ + \ b_o\right)$$

Image Credit: Christopher Olah (http://colah.github.io/posts/2015-08-Understanding-LSTMs/)

LSTM Variants #2: Coupled Gates

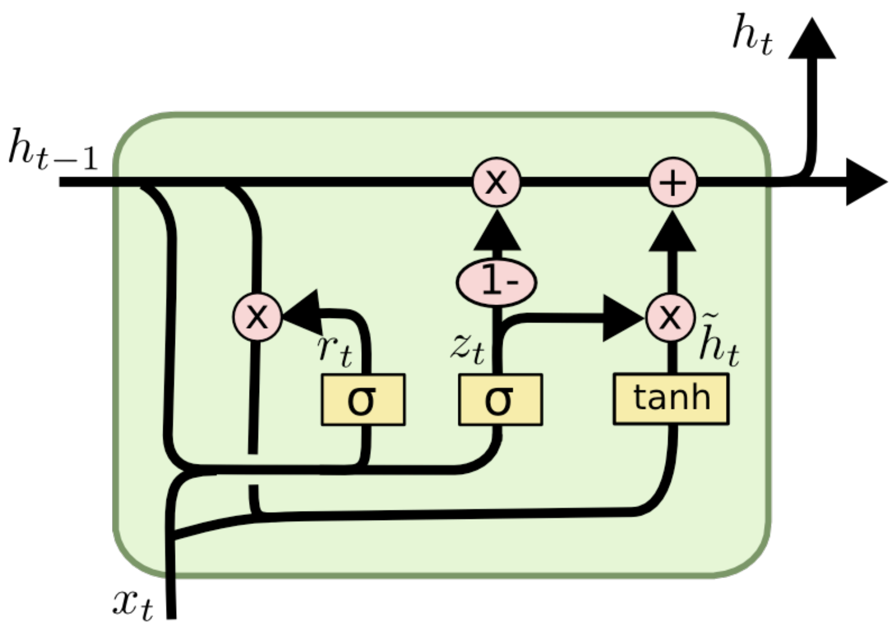
- Only memorize new if forgetting old



$$C_t = f_t * C_{t-1} + (1 - f_t) * \tilde{C}_t$$

LSTM Variants #3: Gated Recurrent Units

- Changes:
  - No explicit memory; memory = hidden output
  - Z = memorize new and forget old



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

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

 Image Credit: Christopher Olah (http://colah.github.io/posts/2015-08-Understanding-LSTMs/)