CS194/294-129: Designing, Visualizing and Understanding Deep Neural Networks

John Canny

Spring 2018

Lecture 9: Recurrent Networks, LSTMs and Applications

Last time: Localization and Detection

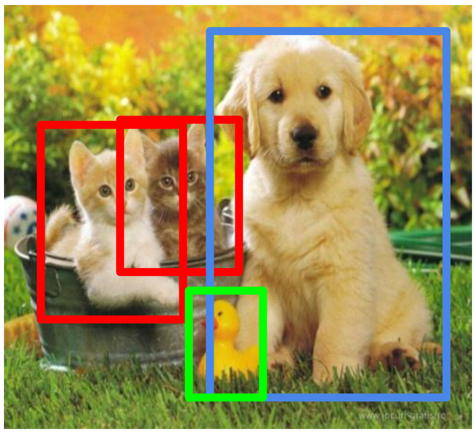
Classification

CAT

Classification + Localization

CAT

Object Detection



CAT, DOG, DUCK

Instance Segmentation

CAT, DOG, DUCK

Single object

Multiple objects

Updates

Please get your project proposal in asap:

- Submit now even if your team is not complete.
- We will try to merge small teams with related topics.

Assignment 2 is out, due 3/5 at 11pm. You’re encouraged but not required to use an EC2 virtual machine.

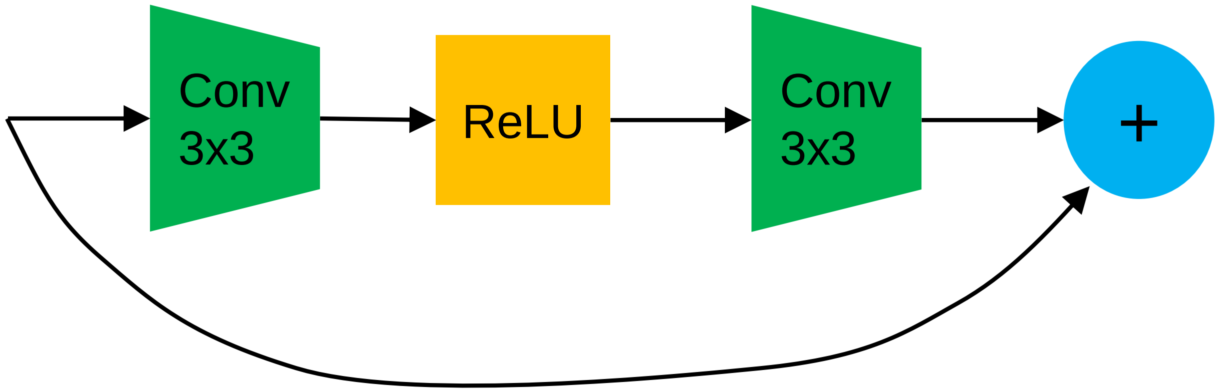
This week’s (starting tomorrow) discussion sections will focus on Tensorflow.

Midterm 1 is coming up on 2/26. Next week’s sections will be midterm preparation.

Neural Network structure

Standard Neural Networks are DAGs (Directed Acyclic Graphs). That means they have a topological ordering.

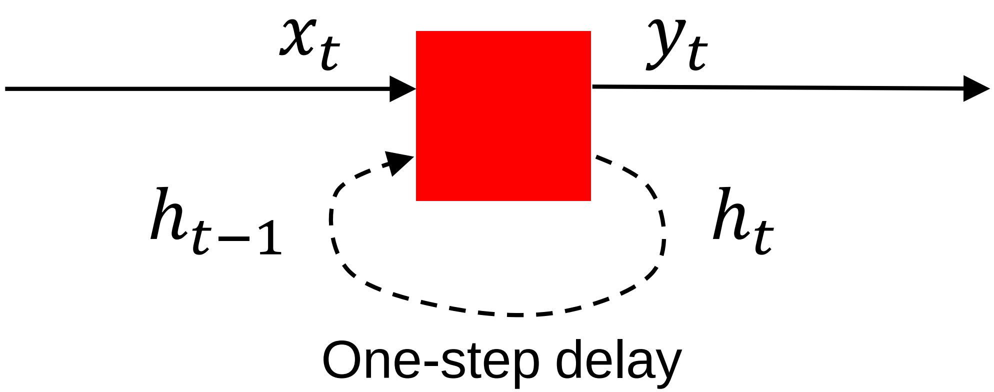
- The topological ordering is used for activation propagation, and for gradient back-propagation.



- These networks process one input minibatch at a time.

Recurrent Neural Networks (RNNs)

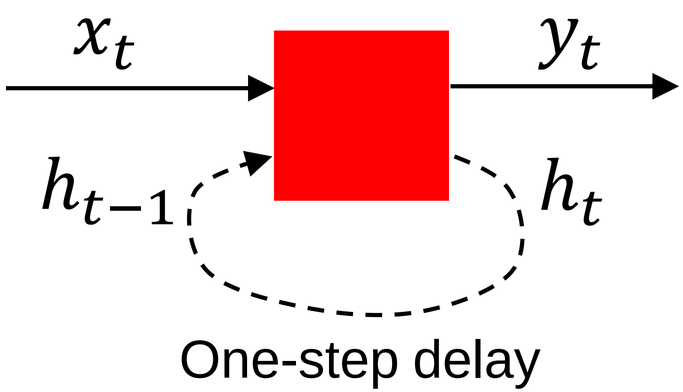
Recurrent networks introduce cycles and a notion of time.

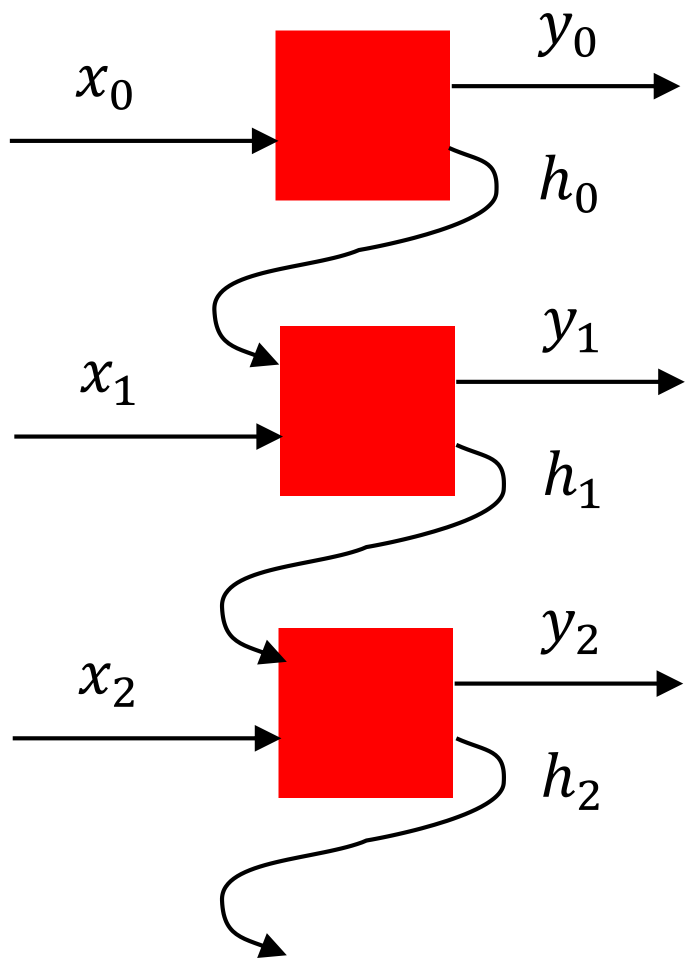


- They are designed to process sequences of data $x_1, \dots, x_n$ and can produce sequences of outputs $y_1, \dots, y_m$.

Unrolling RNNs

RNNs can be unrolled across multiple time steps.



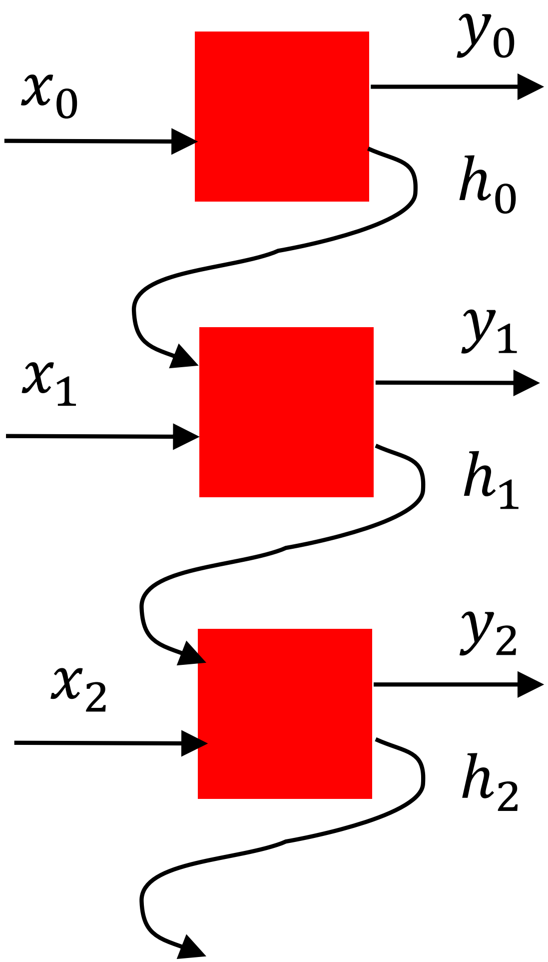


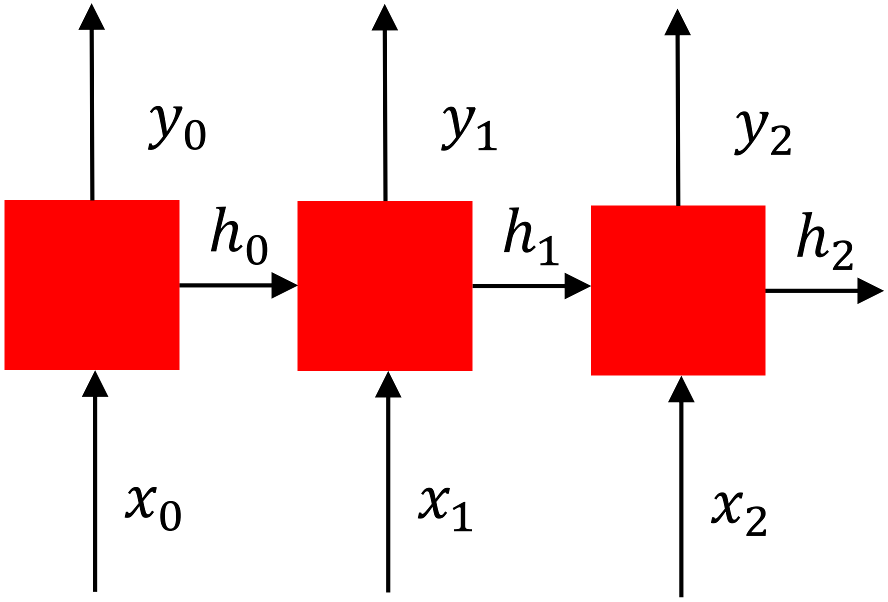
This produces a DAG which supports backpropagation.

But its size depends on the input sequence length.

Unrolling RNNs

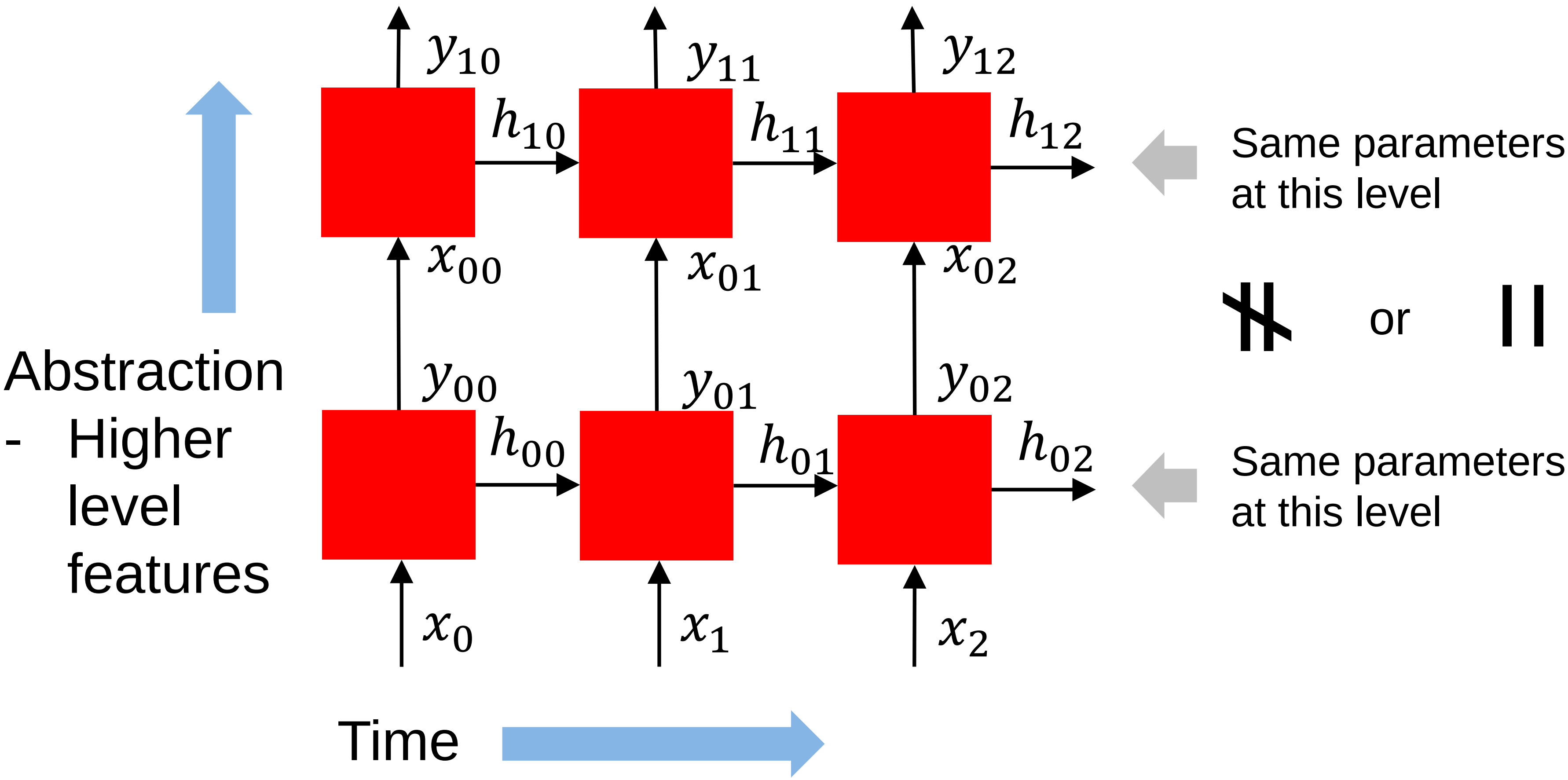
Usually drawn as:





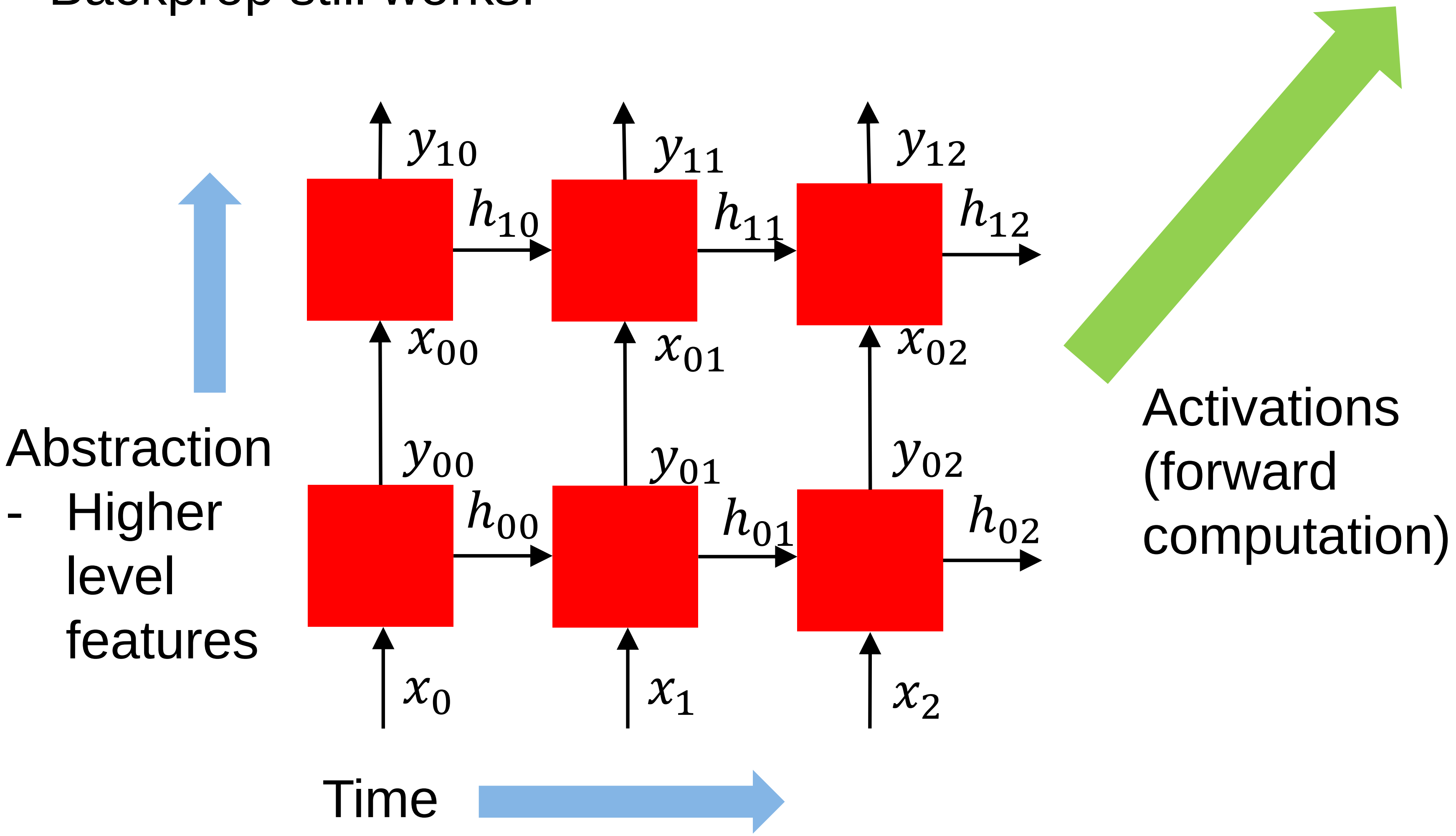
RNN structure

Often layers are stacked vertically (deep RNNs):



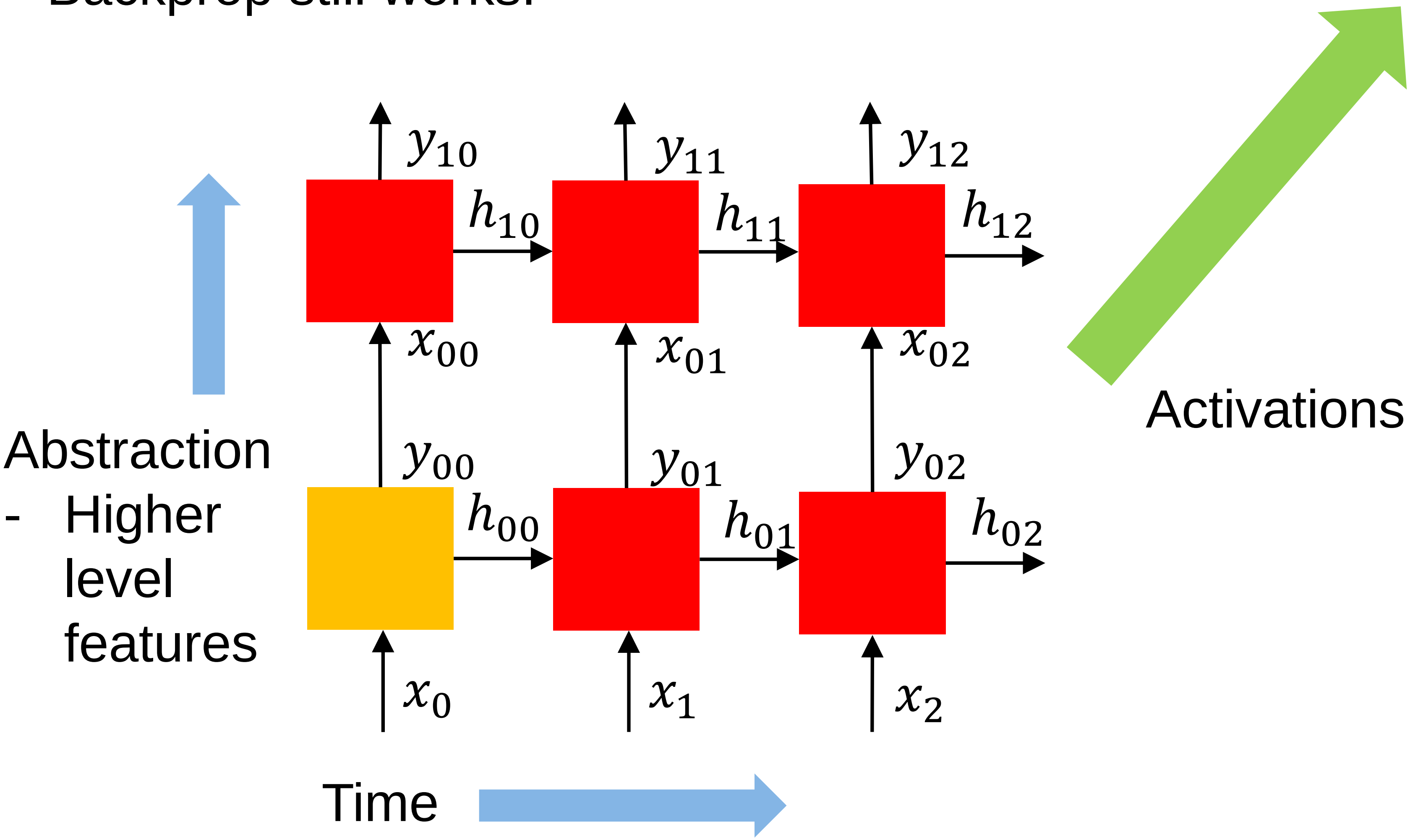
RNN structure

Backprop still works:



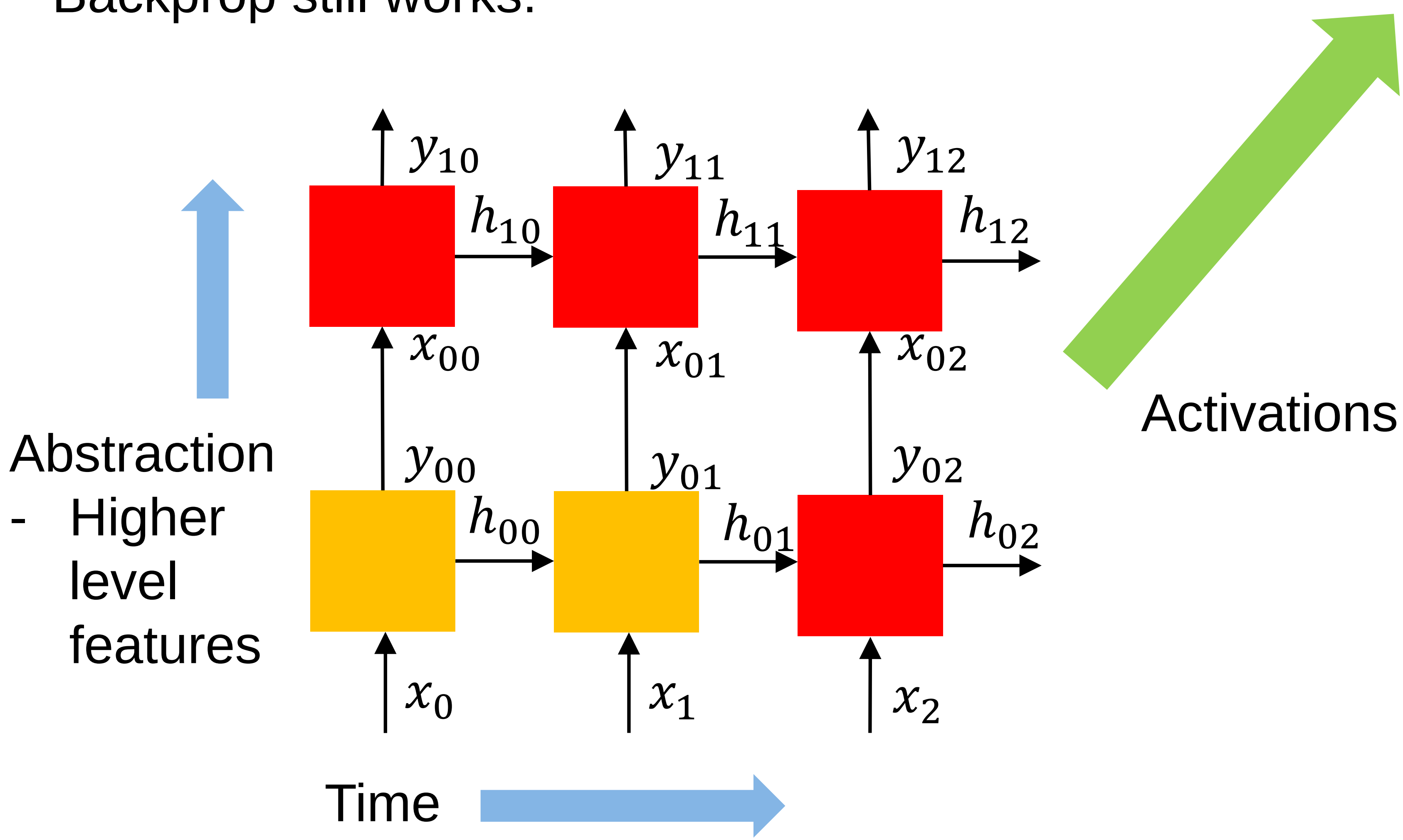
RNN structure

Backprop still works:



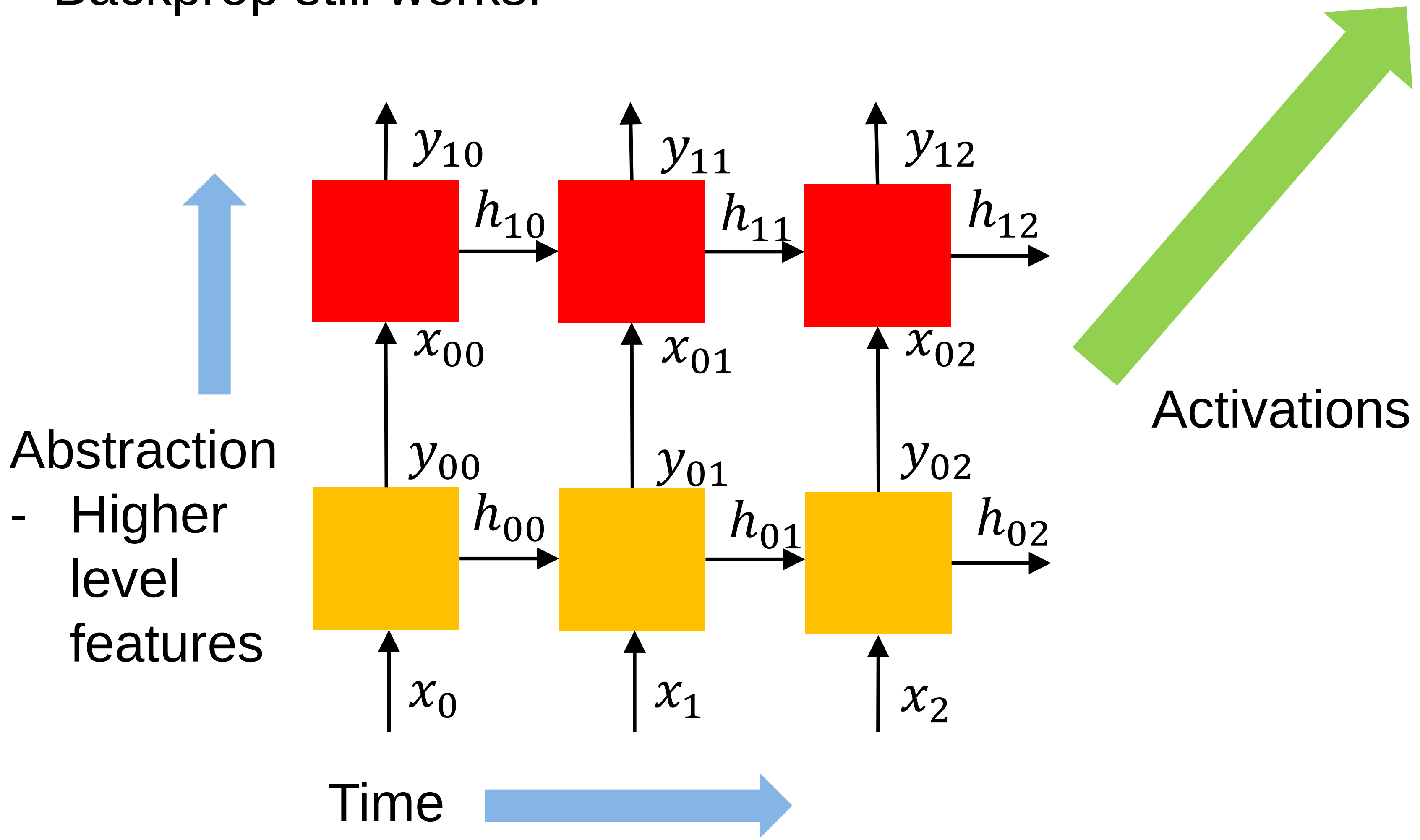
RNN structure

Backprop still works:



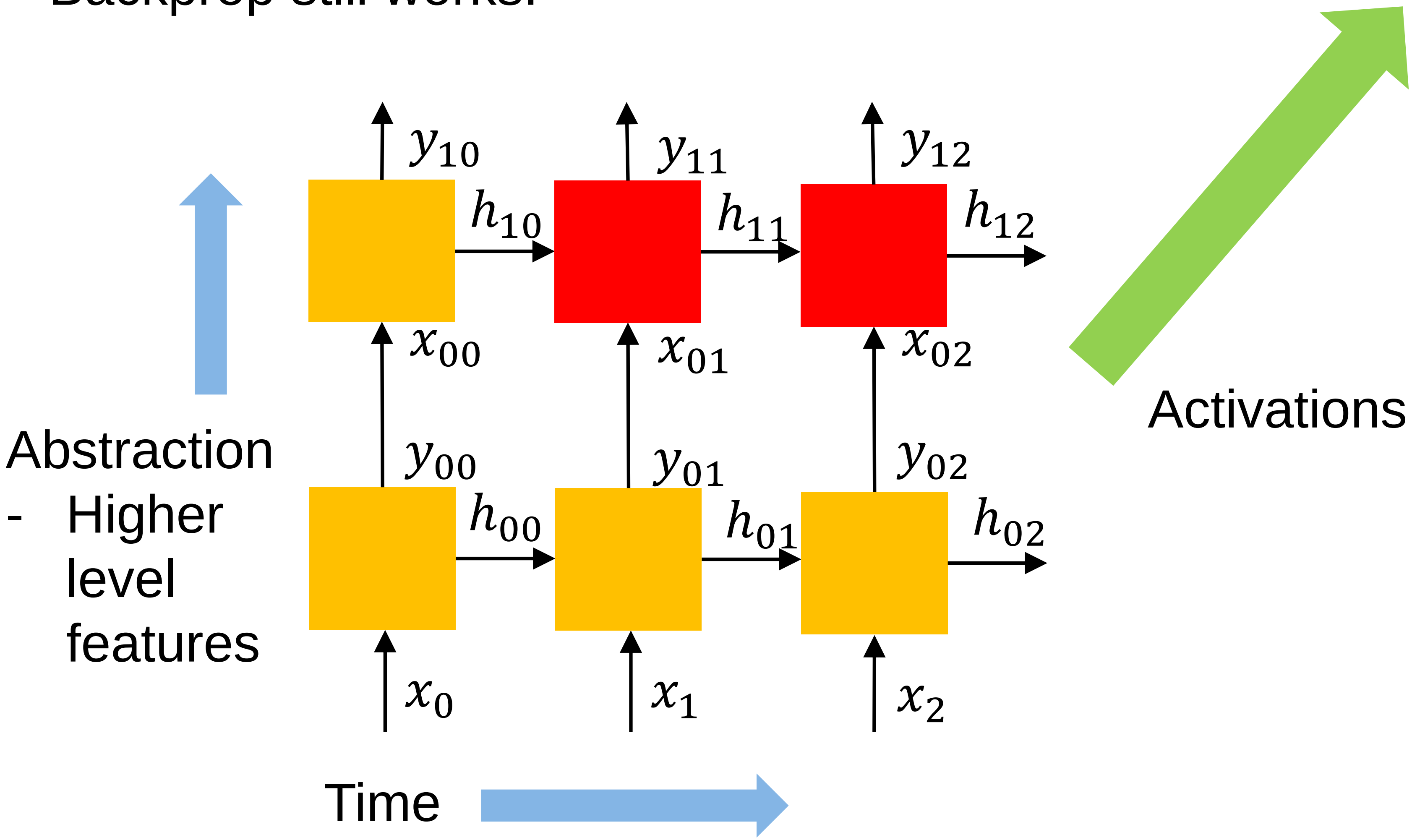
RNN structure

Backprop still works:



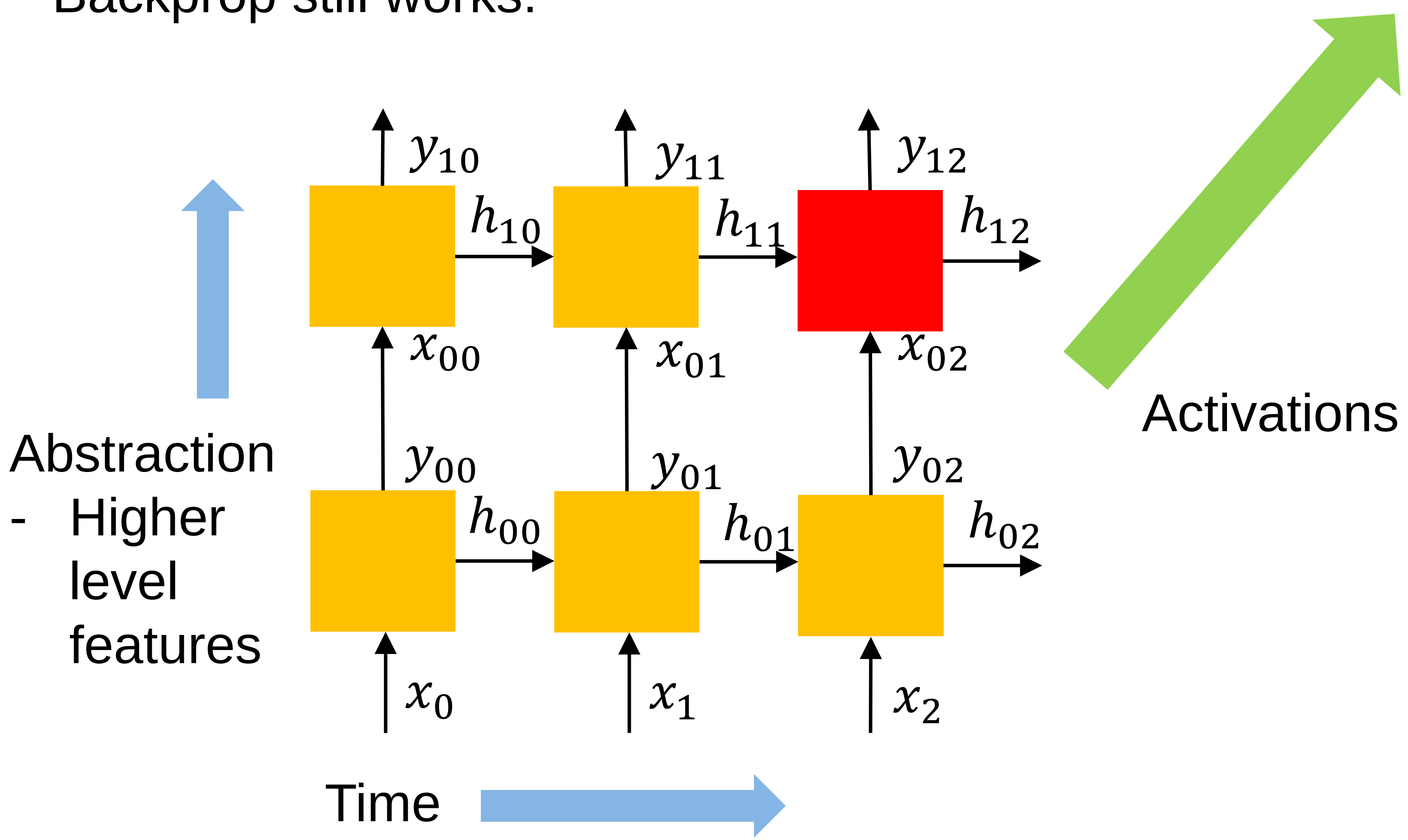
RNN structure

Backprop still works:



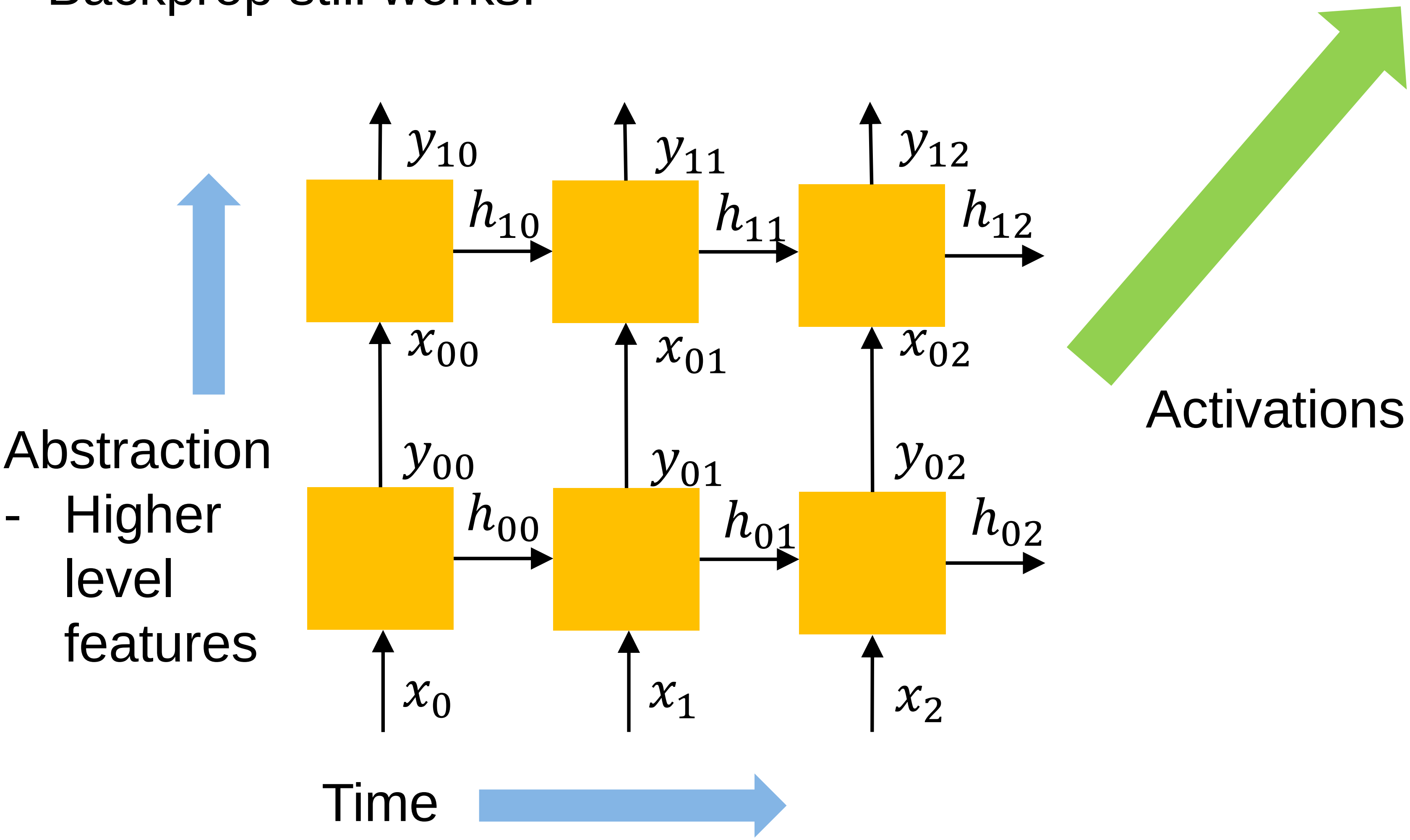
RNN structure

Backprop still works:



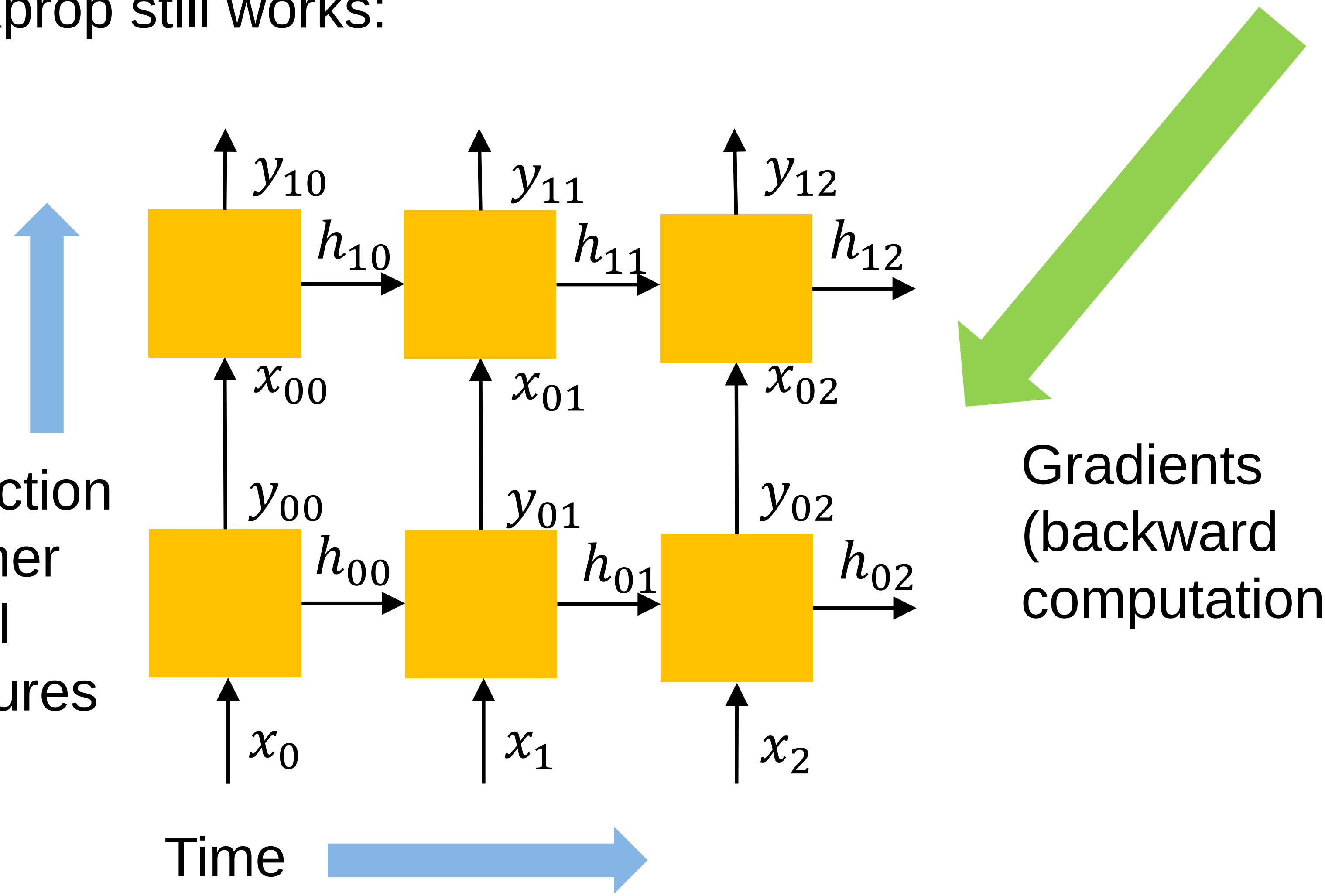
RNN structure

Backprop still works:



RNN structure

Backprop still works:

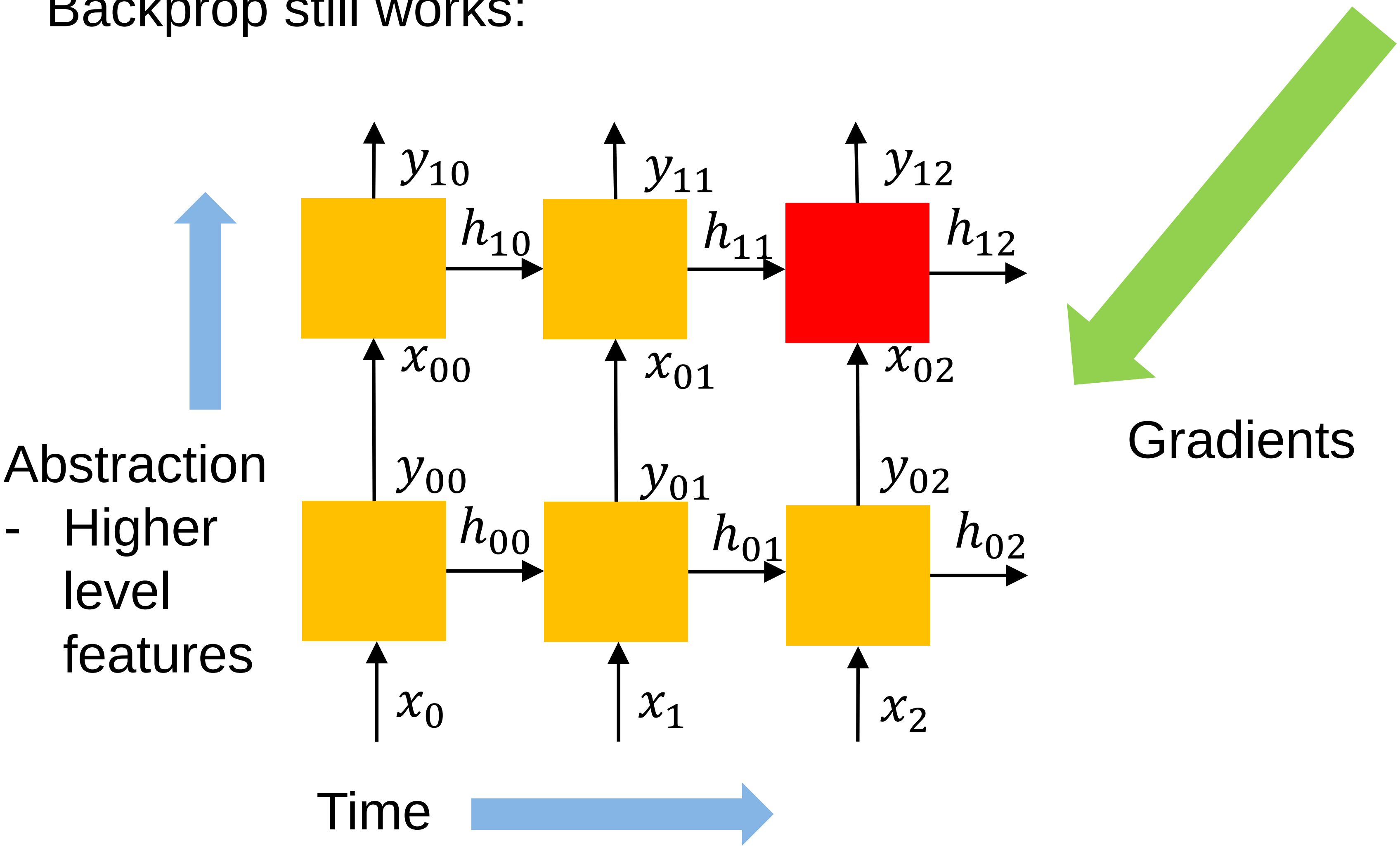


Abstraction
- Higher level features

Gradients (backward computation)

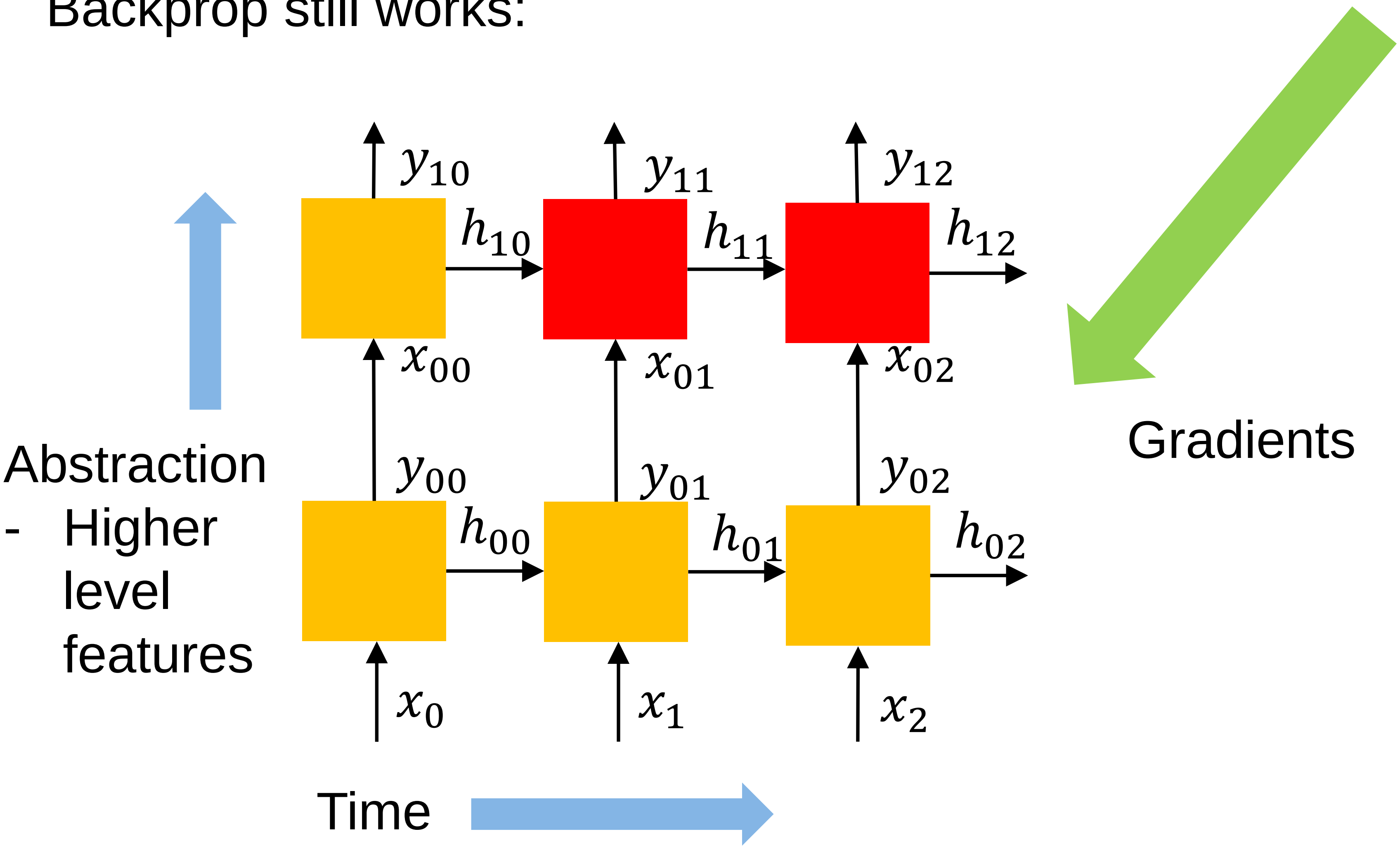
RNN structure

Backprop still works:



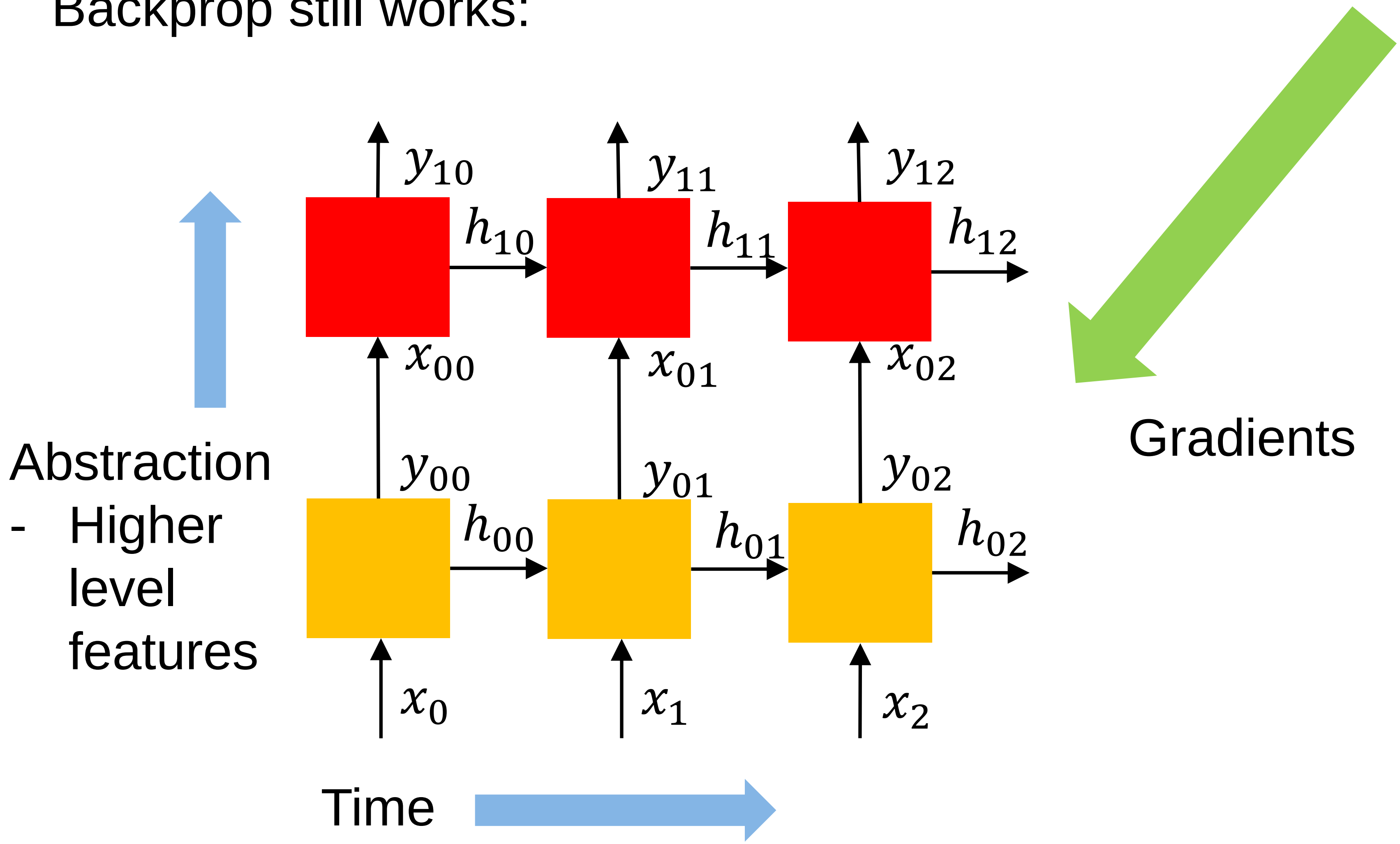
RNN structure

Backprop still works:



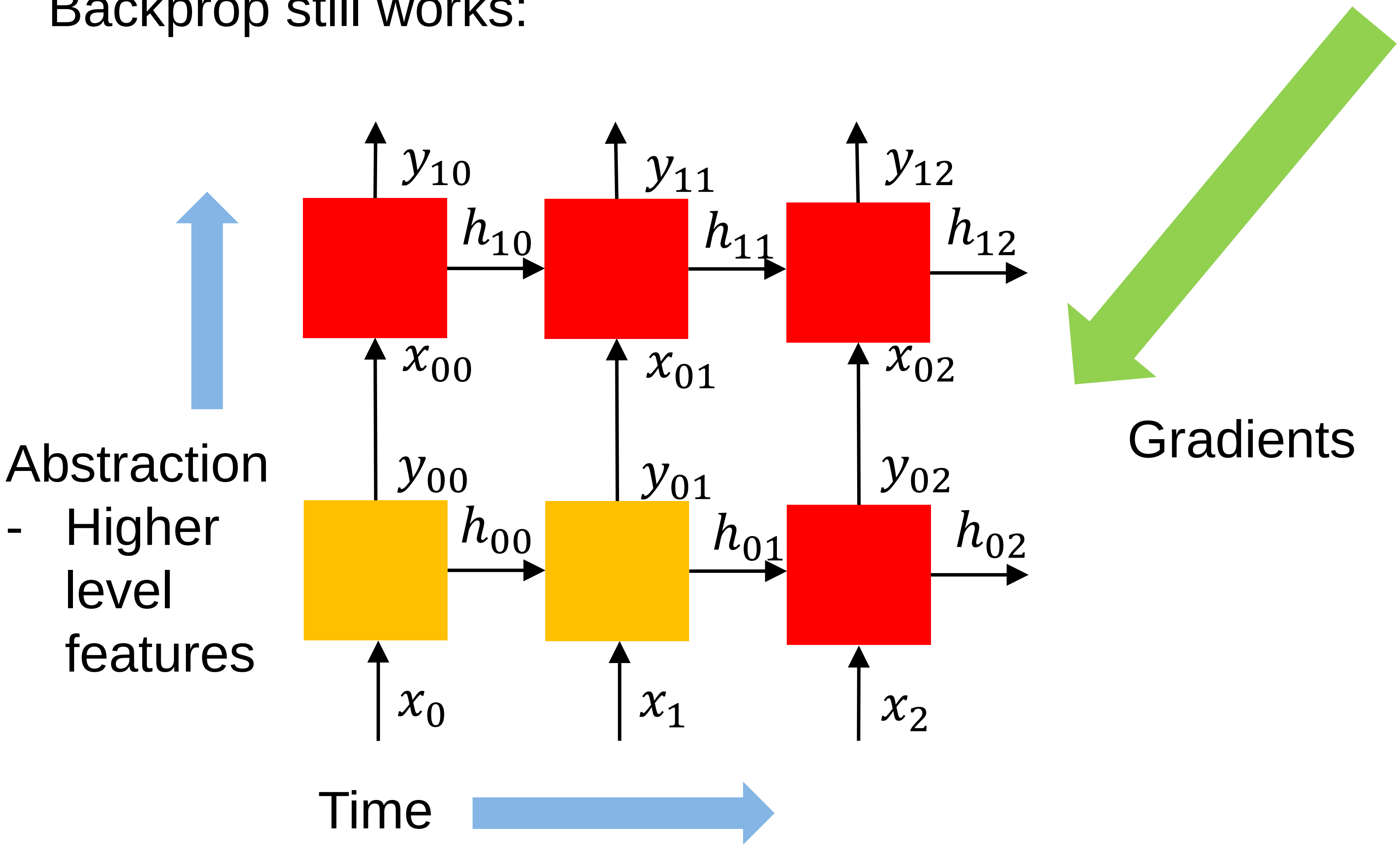
RNN structure

Backprop still works:



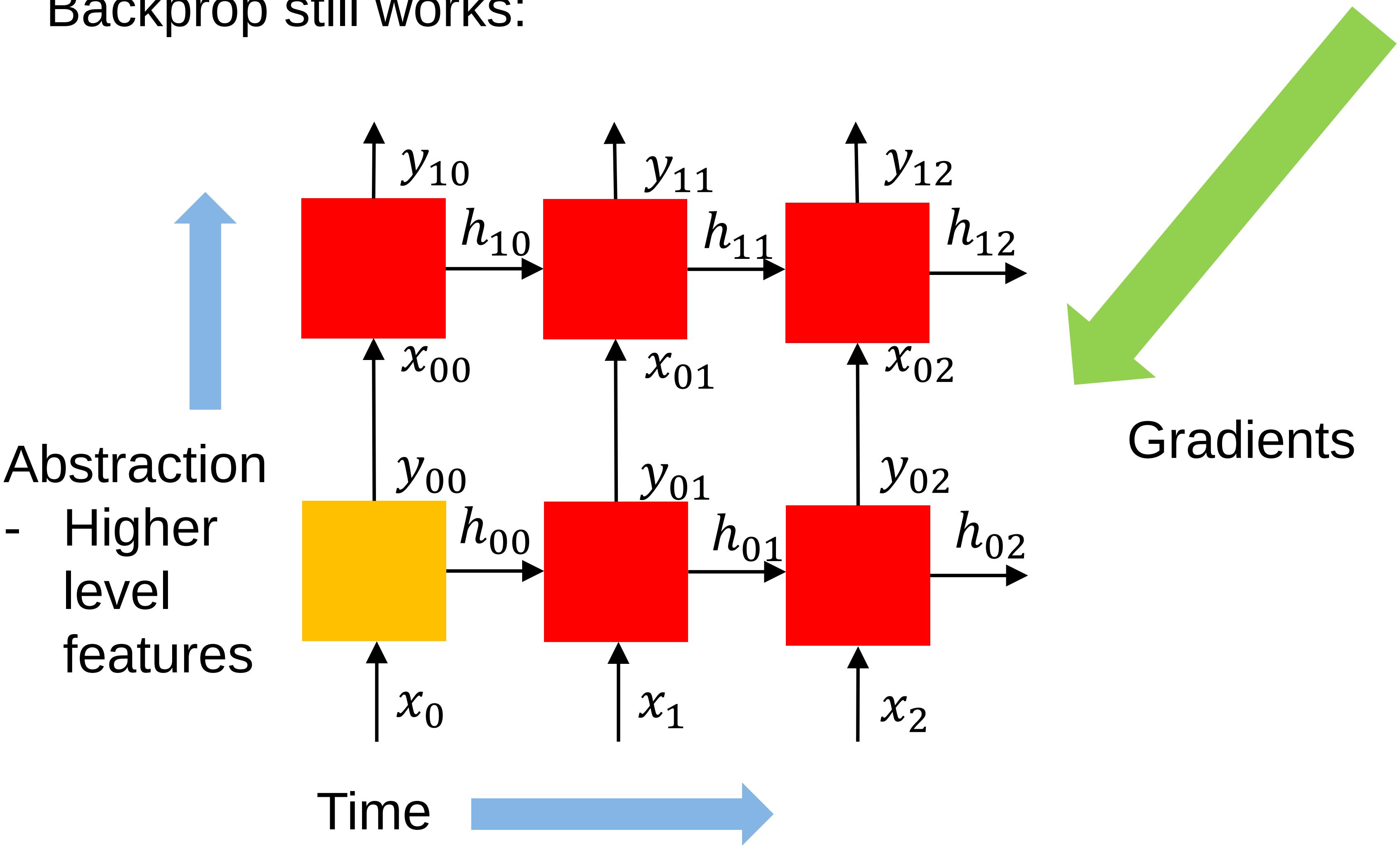
RNN structure

Backprop still works:



RNN structure

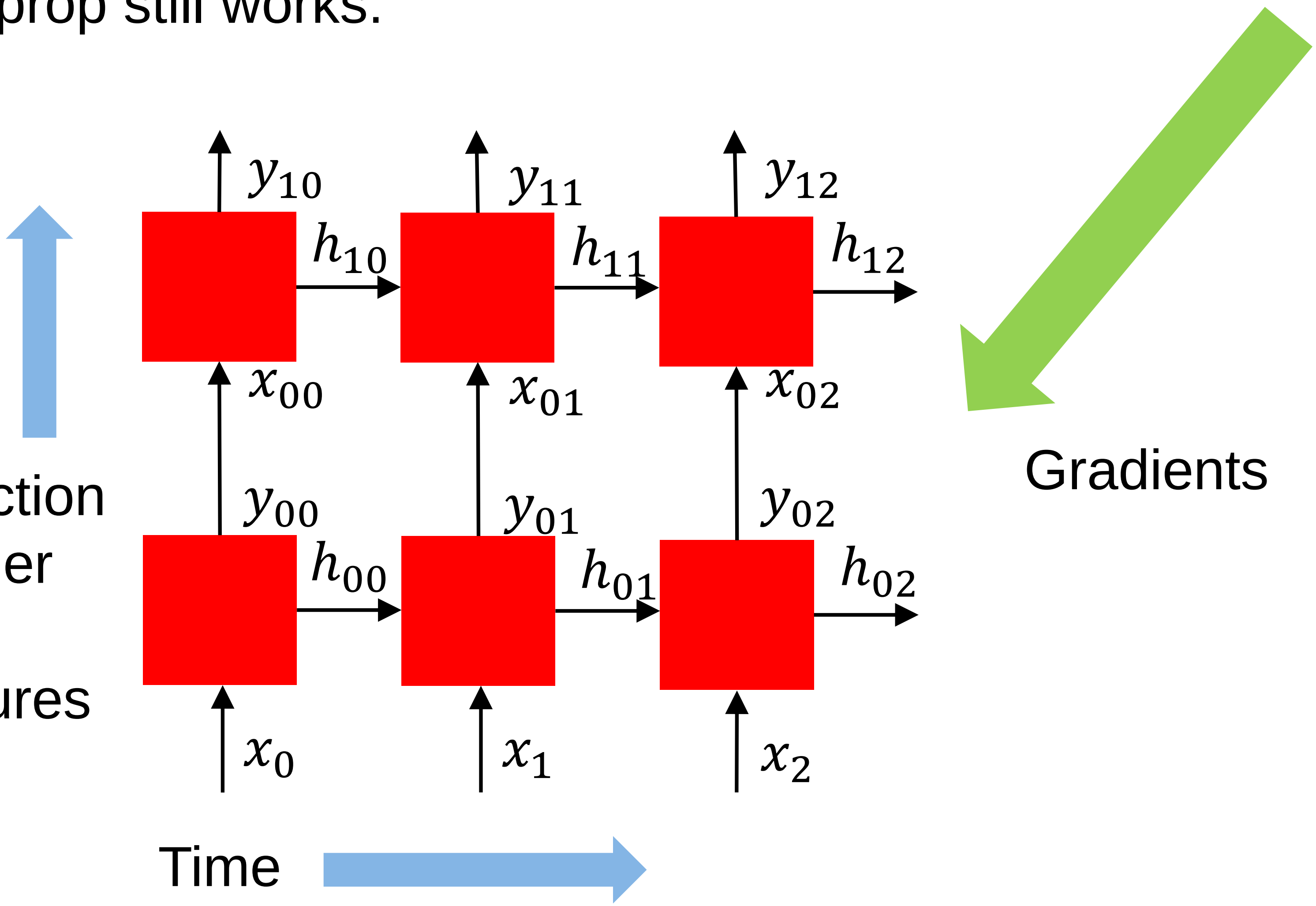
Backprop still works:



RNN structure

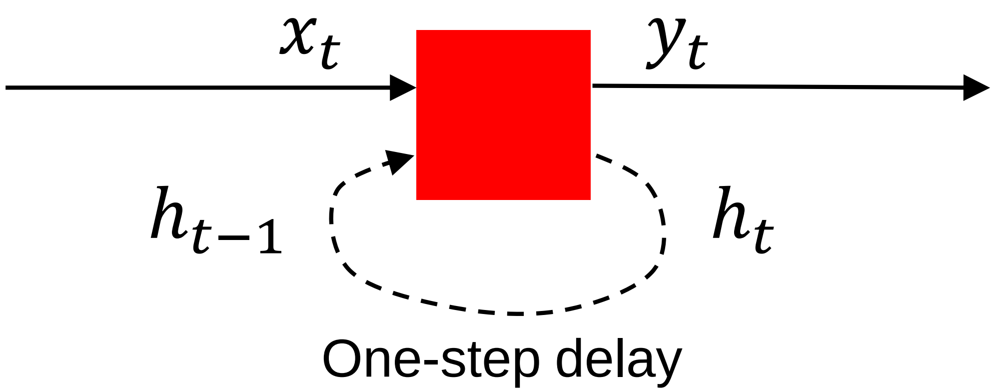
Backprop still works:

Abstraction
- Higher level features



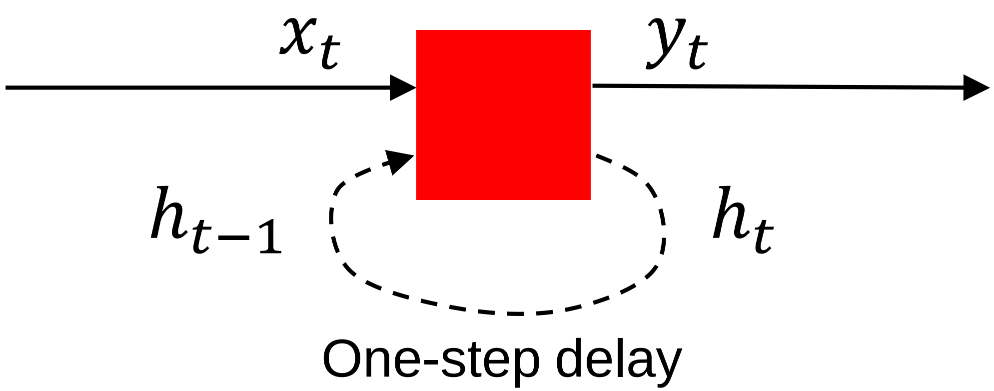
RNN unrolling

Question: Can you run forward/backward inference on an unrolled RNN, i.e. keeping only one copy of its state?



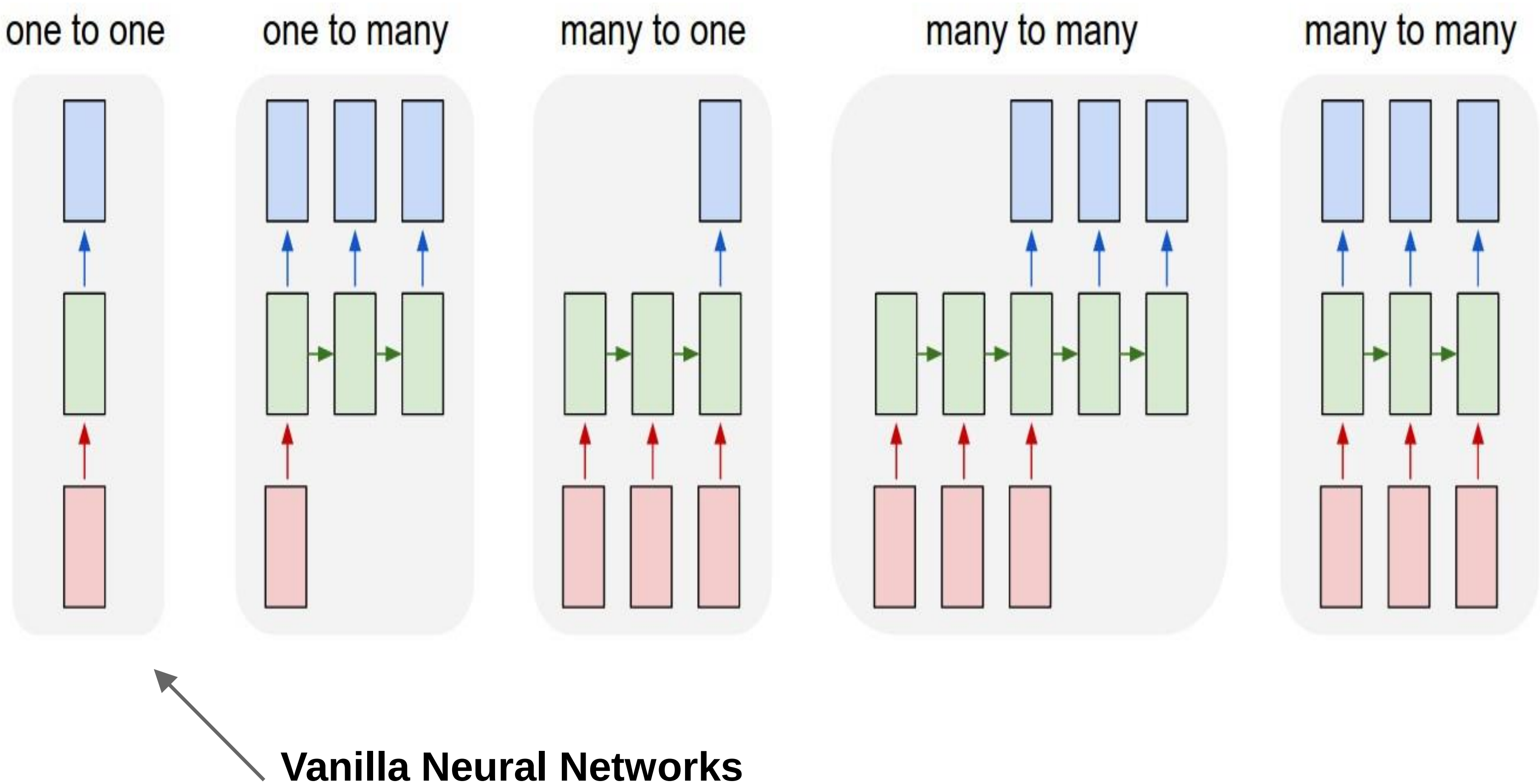
RNN unrolling

Question: Can you run forward/backward inference on an unrolled RNN, i.e. keeping only one copy of its state?



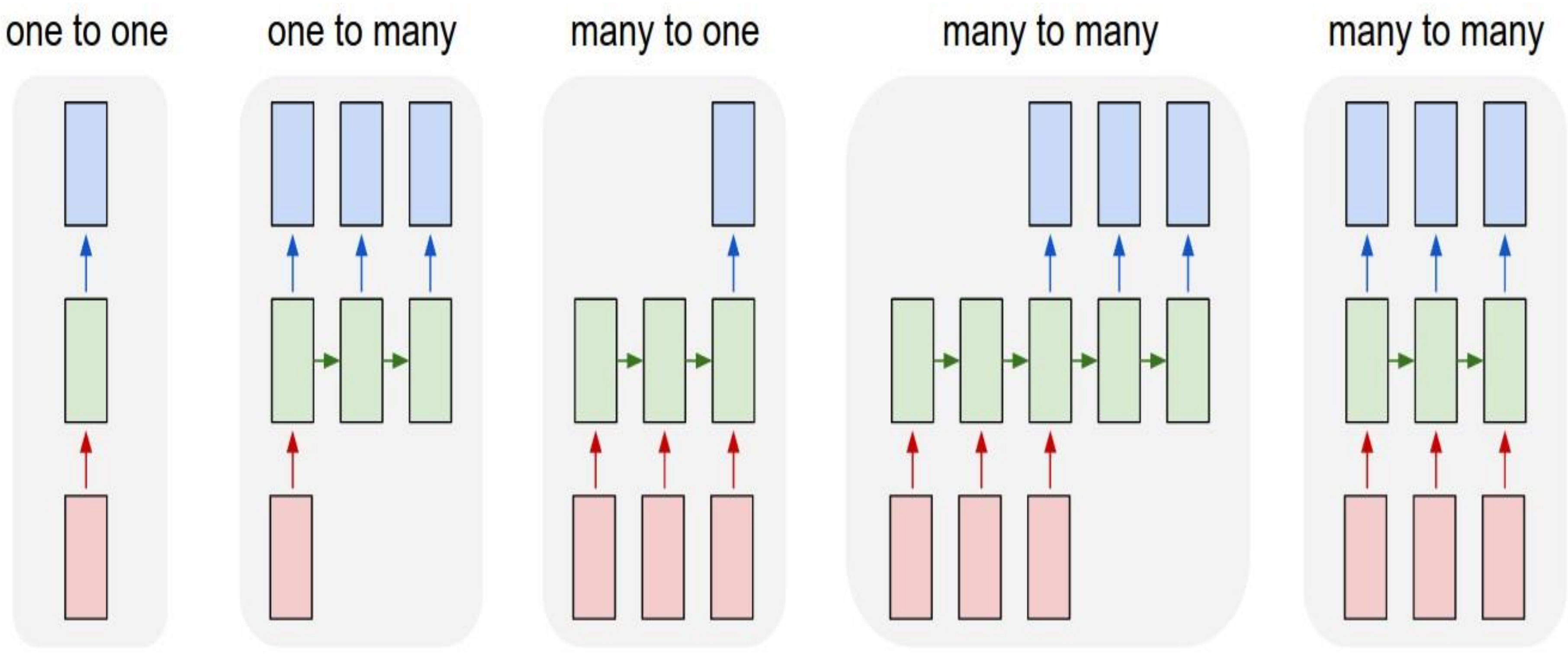
Forward: Yes
Backward: No

Recurrent Networks offer a lot of flexibility:



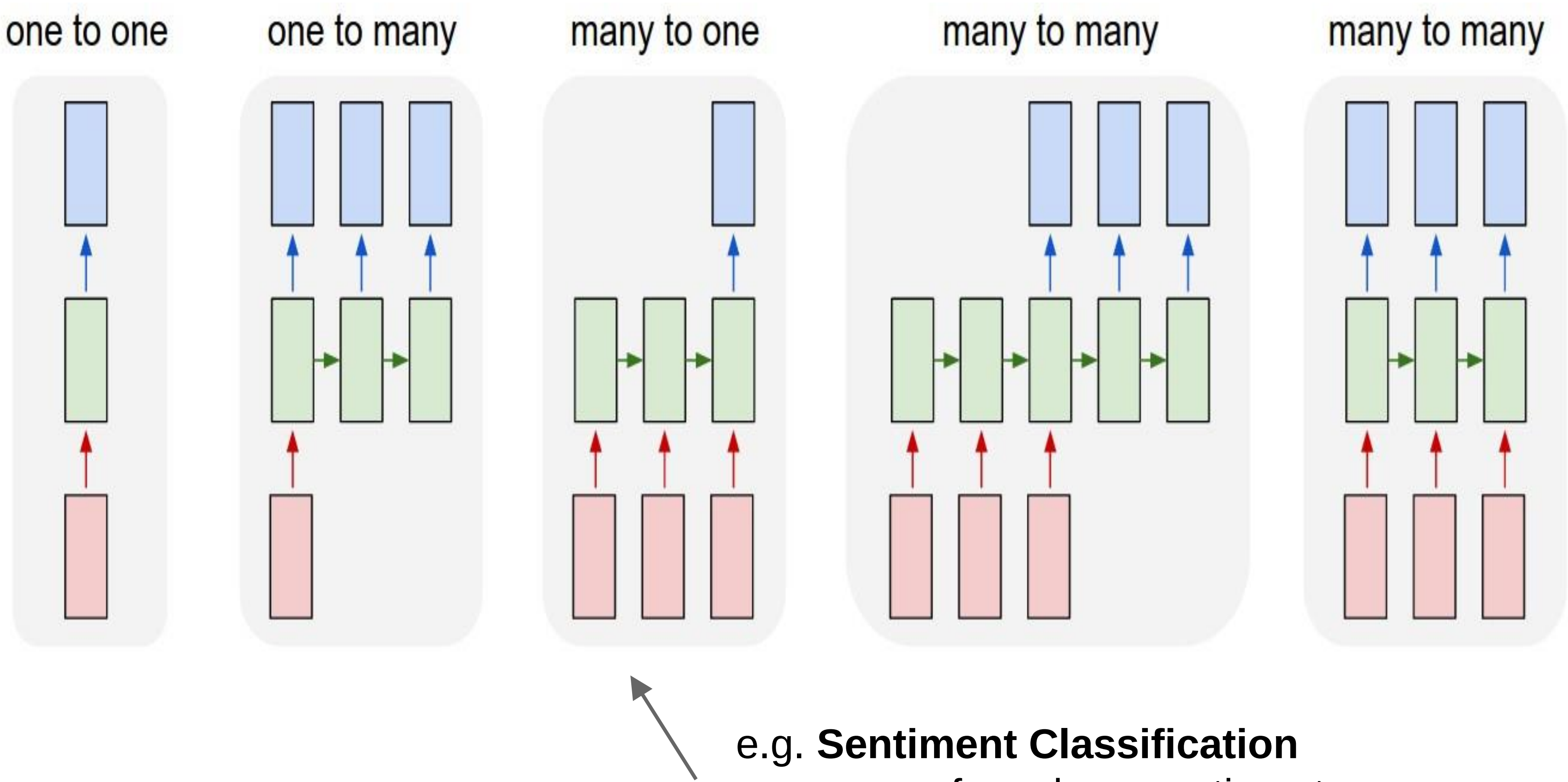
Vanilla Neural Networks

Recurrent Networks offer a lot of flexibility:



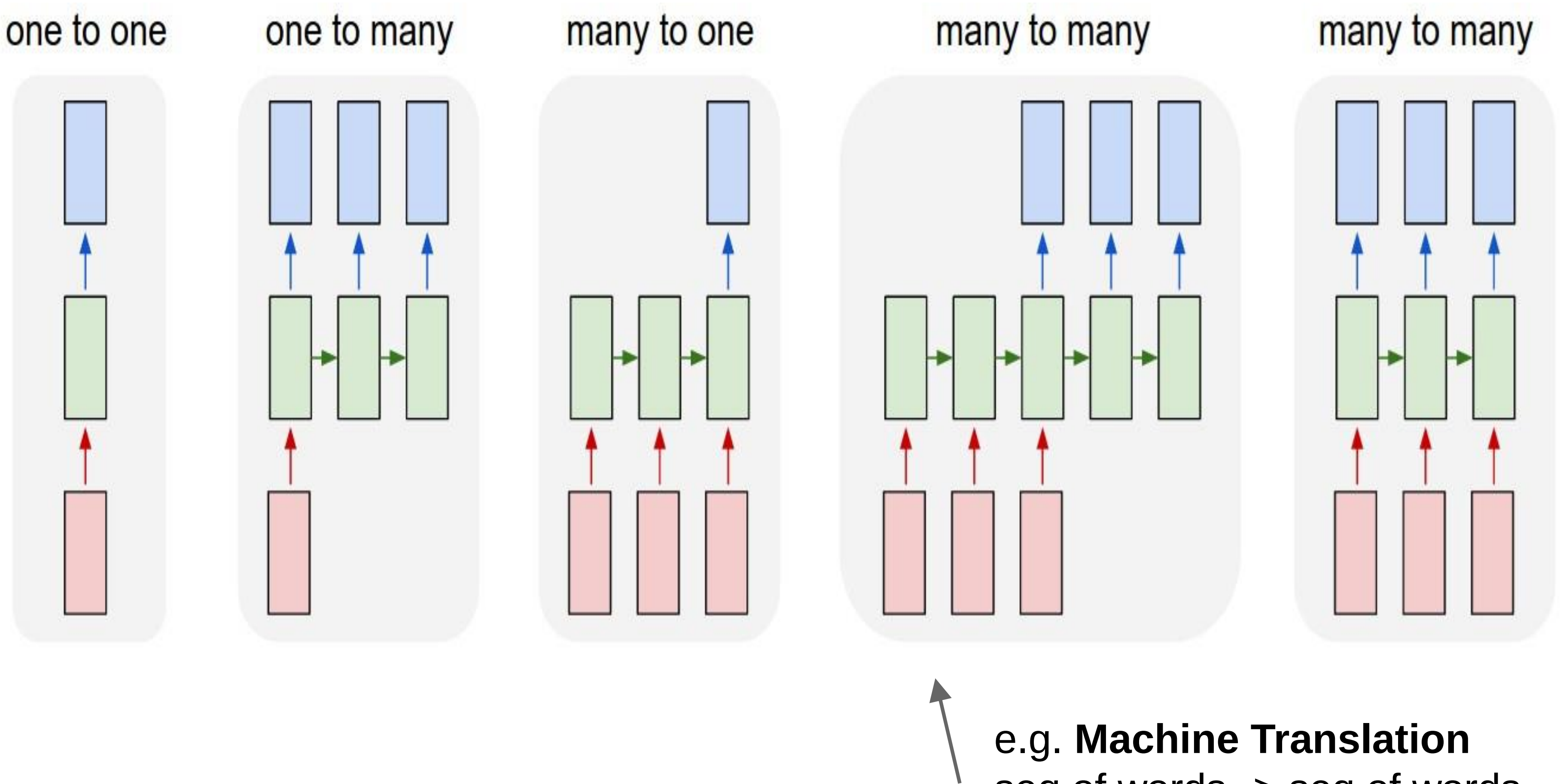
e.g. **Image Captioning**
image -> sequence of words

Recurrent Networks offer a lot of flexibility:



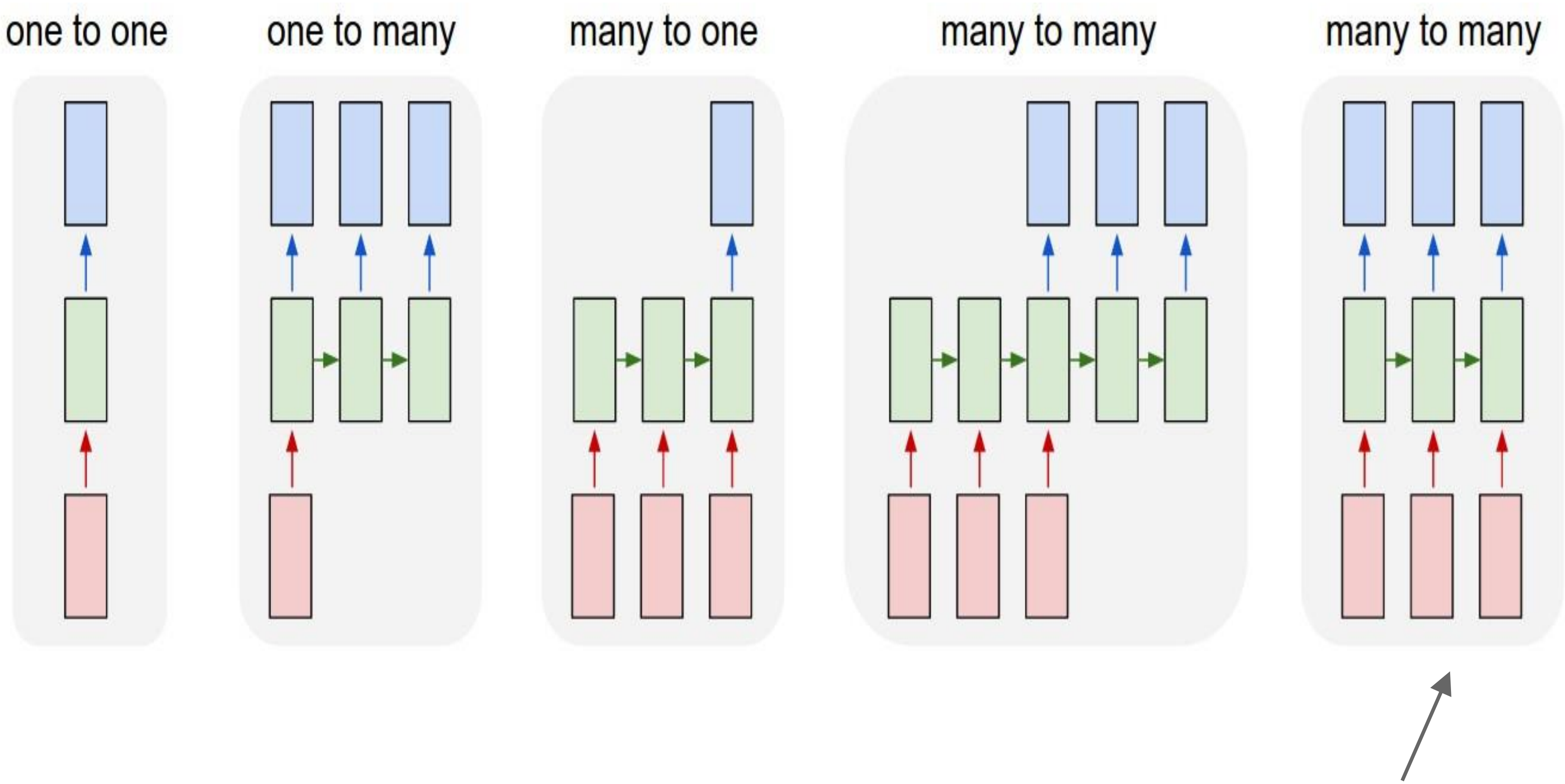
e.g. **Sentiment Classification**
sequence of words -> sentiment

Recurrent Networks offer a lot of flexibility:



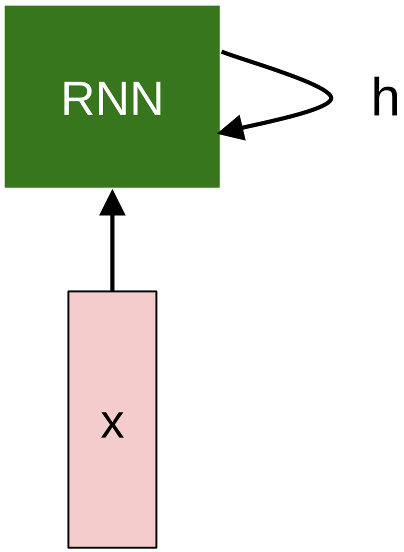
e.g. **Machine Translation**
seq of words -> seq of words

Recurrent Networks offer a lot of flexibility:

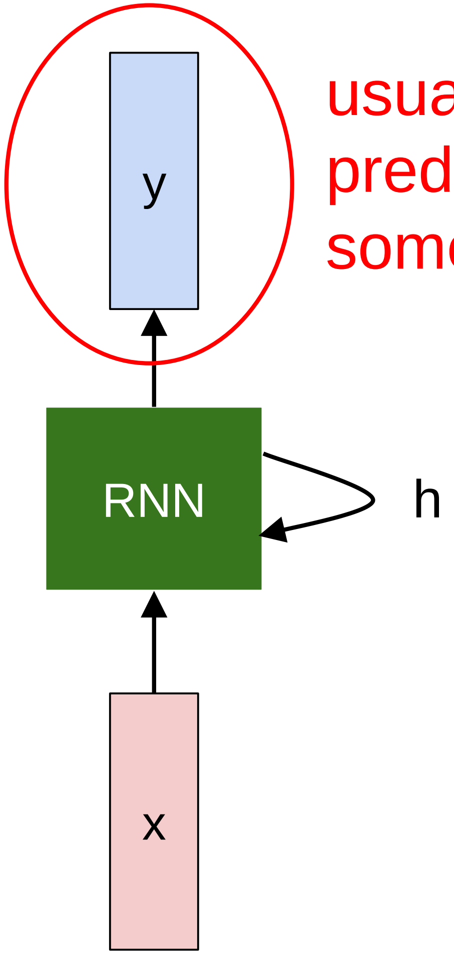


e.g. **Video classification on frame level**

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:

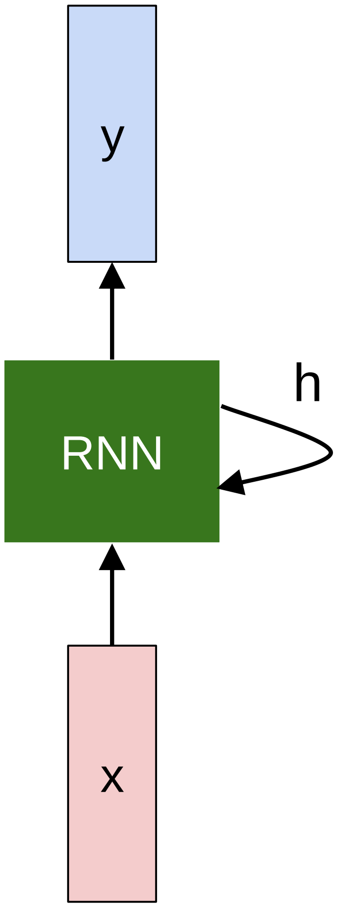
$$\boxed{h_t} = \boxed{f_W}(\boxed{h_{t-1}}, \boxed{x_t})$$

new state

old state

input vector at some time step

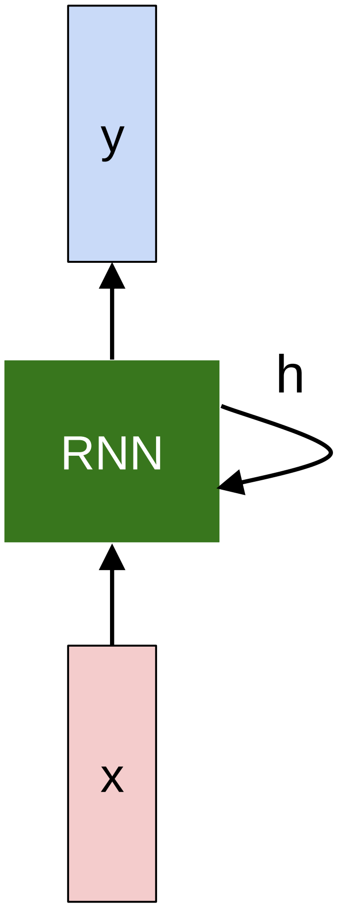
some function with parameters W



Recurrent Neural Network

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

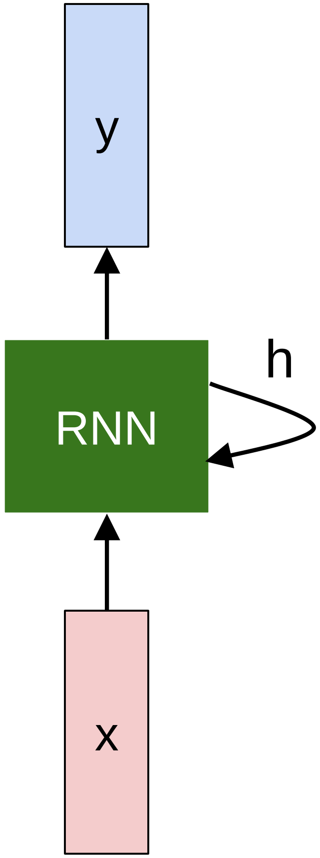
$$h_t = f_W(h_{t-1}, x_t)$$



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

(Vanilla) Recurrent Neural Network

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



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

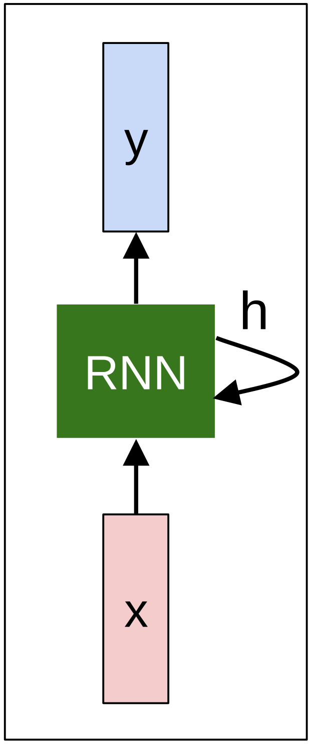
$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

$$y_t = W_{hy}h_t$$

Character-level language model example

Vocabulary:
[h,e,l,o]

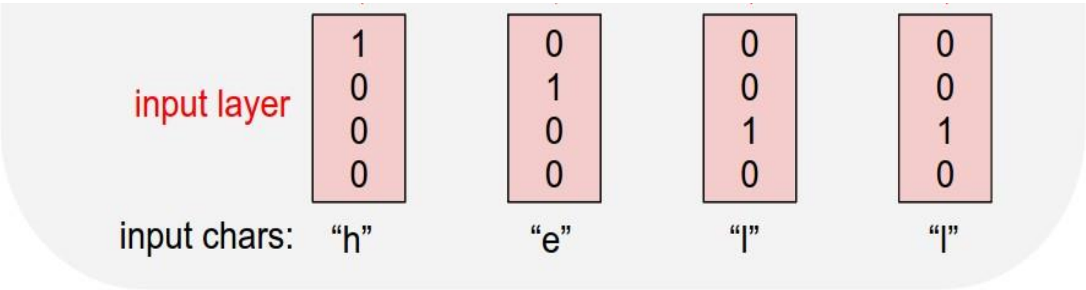
Example training sequence:
“hello”



Character-level language model example

Vocabulary:
[h,e,l,o]

Example training sequence:
“hello”

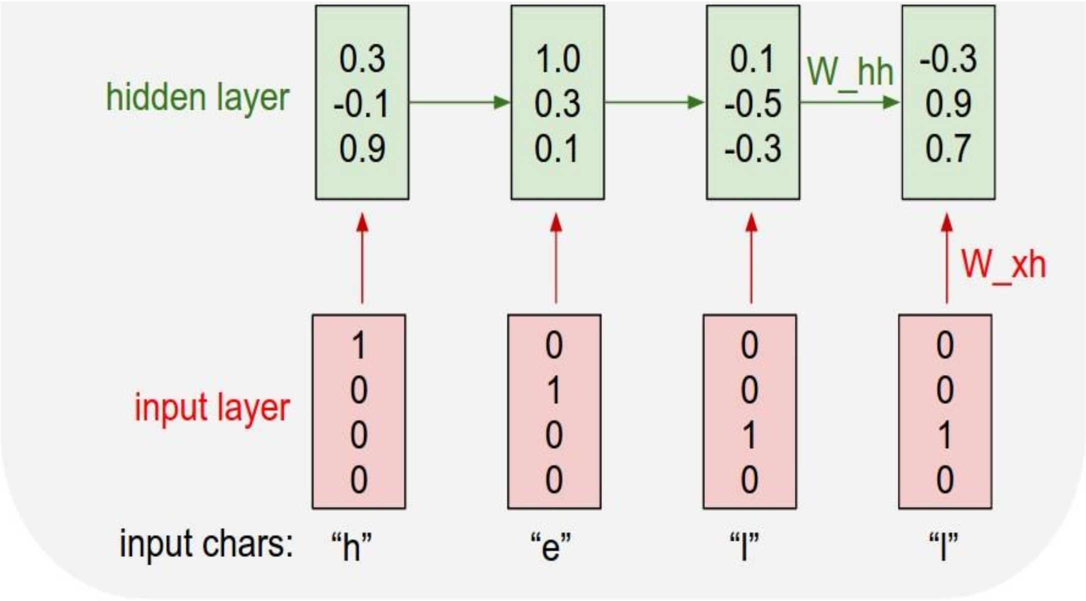


Character-level anguage model example

Vocabulary:
[h,e,l,o]

Example training sequence:
“hello”

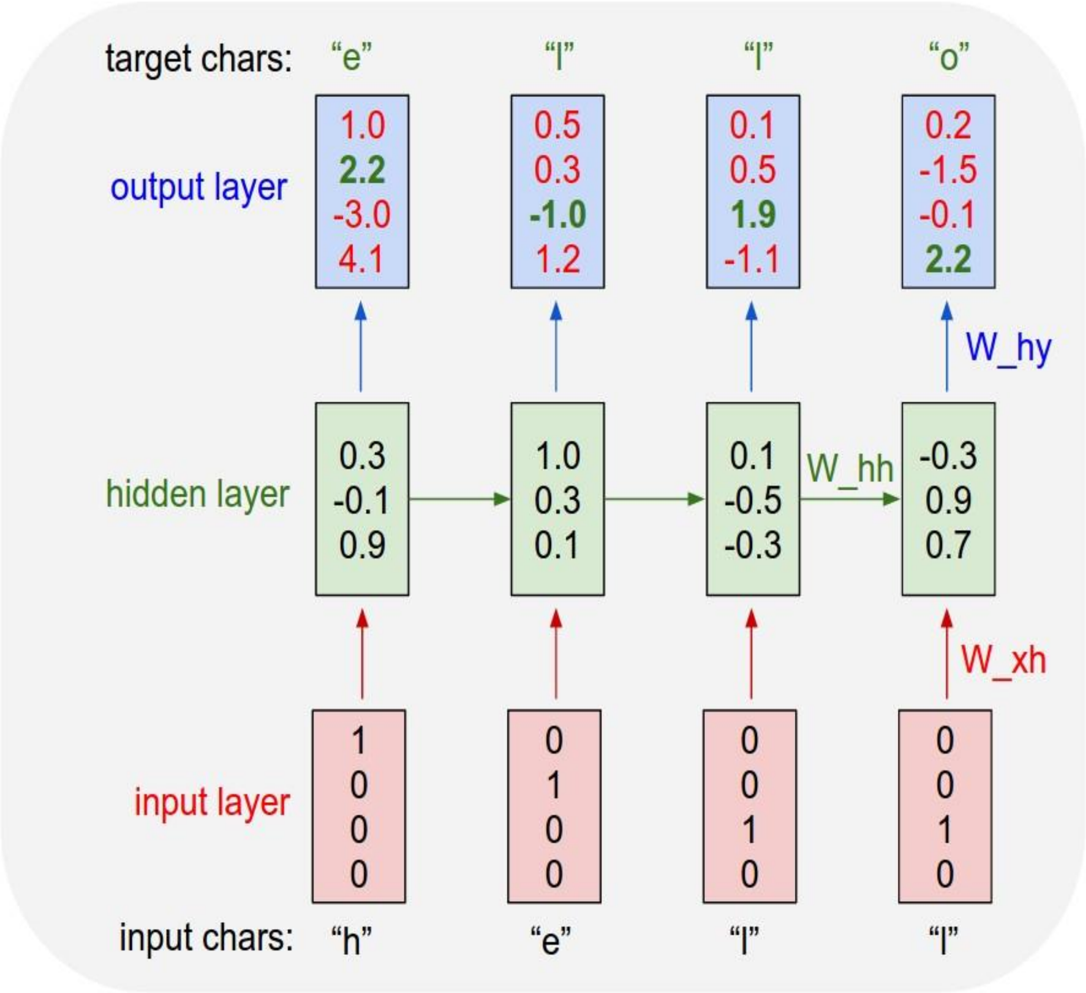
$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$



Character-level language model example

Vocabulary:
[h,e,l,o]

Example training sequence:
“hello”

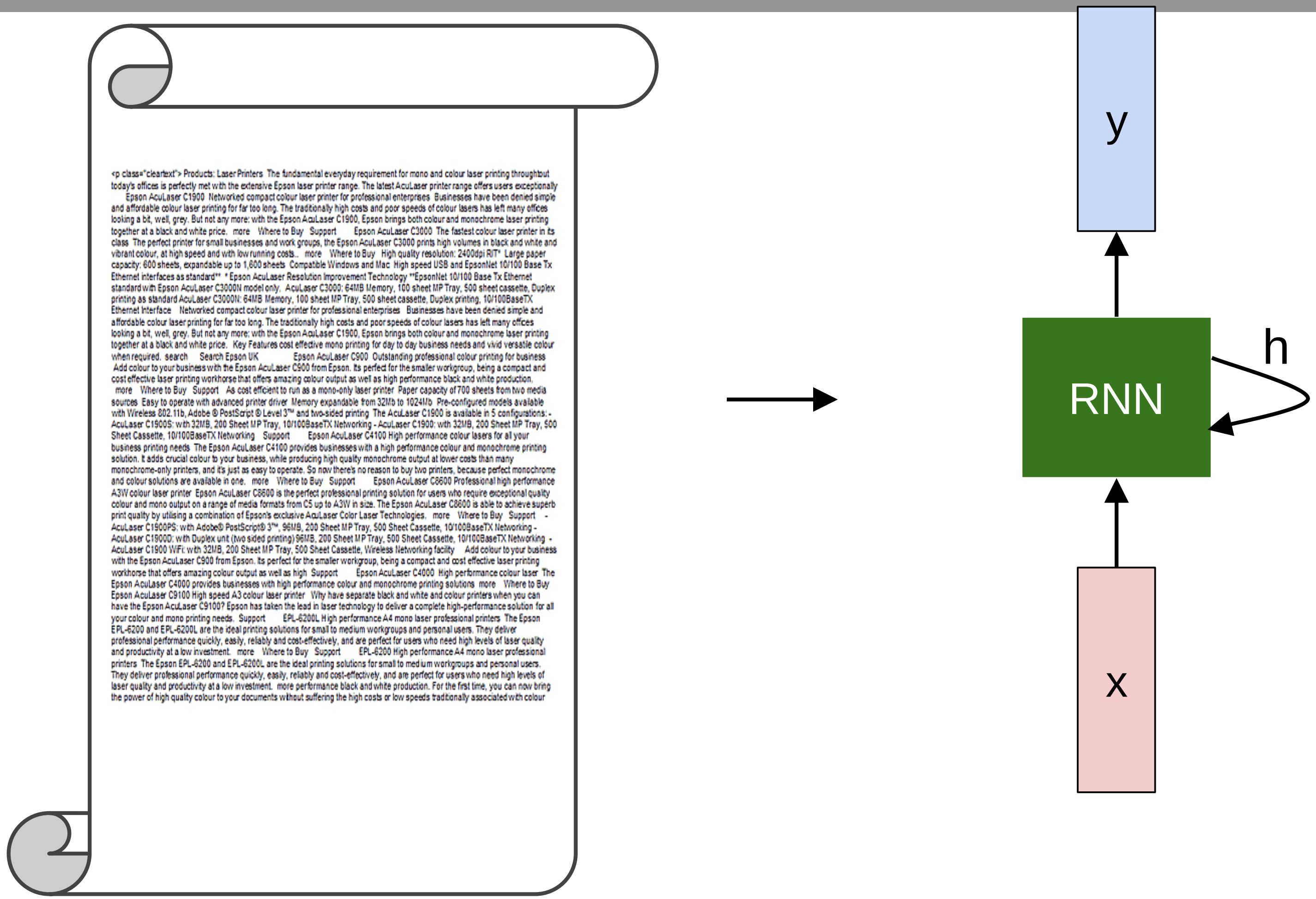


min-char-rnn.py gist: 112 lines of Python

```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)

Applications - Poetry



http://karpathy.github.io/2015/05/21/rnn-effectiveness/

(Vanilla) Recurrent Neural Network

Sonnet 116 – Let me not ...

by William Shakespeare

Let me not to the marriage of true minds
    Admit impediments. Love is not love
Which alters when it alteration finds,
    Or bends with the remover to remove:
O no! it is an ever-fixed mark
    That looks on tempests and is never shaken;
It is the star to every wandering bark,
    Whose worth's unknown, although his height be taken.
Love's not Time's fool, though rosy lips and cheeks
    Within his bending sickle's compass come:
Love alters not with his brief hours and weeks,
    But bears it out even to the edge of doom.
If this be error and upon me proved,
    I never writ, nor no man ever loved.

http://karpathy.github.io/2015/05/21/rnn-effectiveness/

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

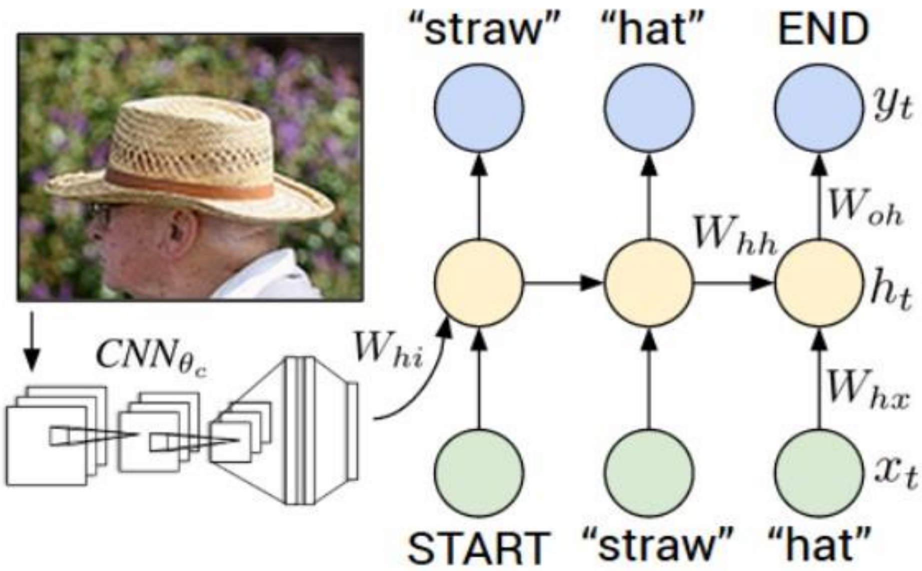
```
"Why do what that day," replied Natasha, and wishing to himself the fact the
princess, Princess Mary was easier, fed in had oftened him.
Pierre aking his soul came to the packs and drove up his father-in-law women.
```

And later:

```
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.
```

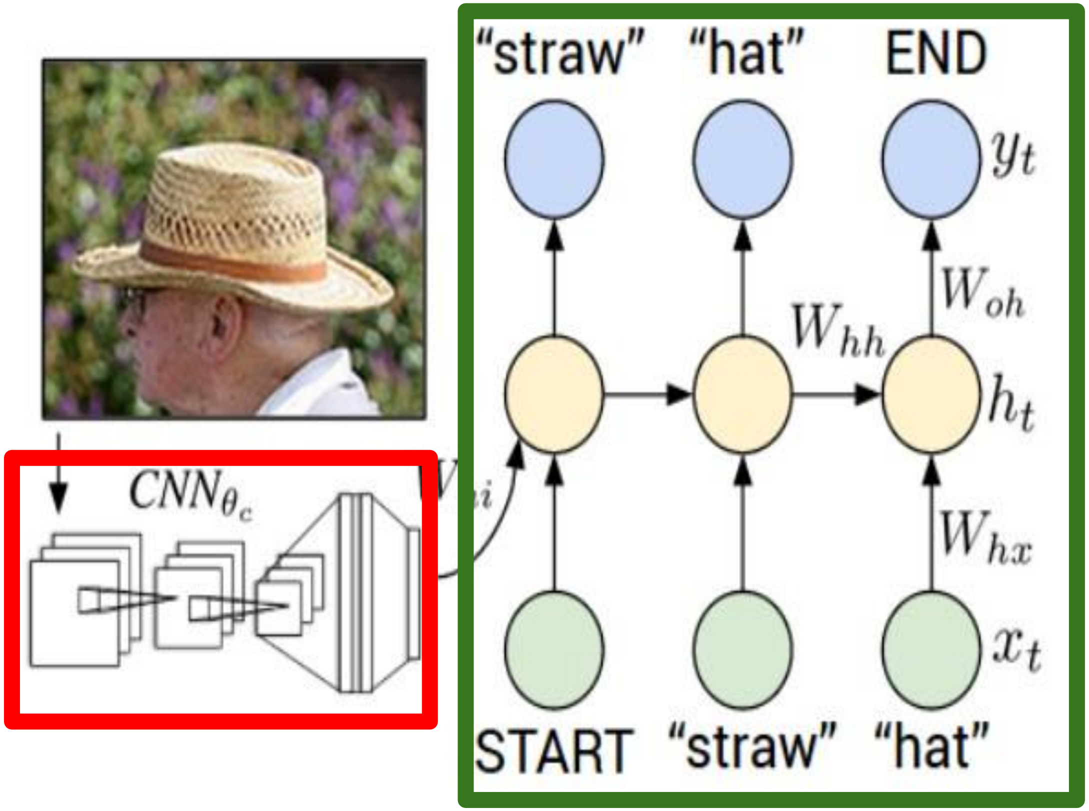
```
VIOLA:
Why, Salisbury must find his flesh and thought
That which I am not aps, not a man and in fire,
To show the reining of the raven and the wars
To grace my hand reproach within, and not a fair are hand,
That Caesar and my goodly father's world;
When I was heaven of presence and our fleets,
We 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



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

Recurrent Neural Network



Convolutional Neural Network

test image


image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
conv-256
conv-256
maxpool
conv-512
conv-512
maxpool
conv-512
conv-512
maxpool
FC-4096
FC-4096
FC-1000
softmax

test image

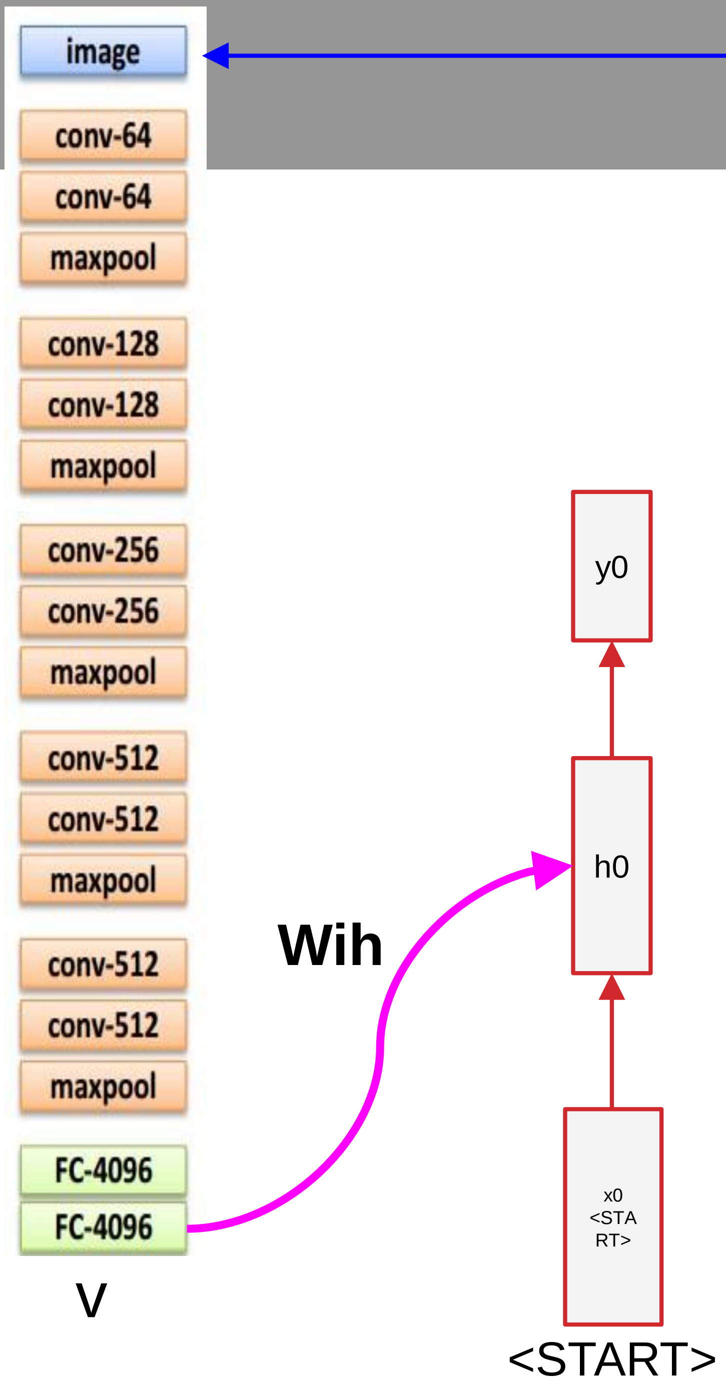
image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
conv-256
conv-256
maxpool
conv-512
conv-512
maxpool
conv-512
conv-512
maxpool
FC-4096
FC-4096
FC-1000

test image

test image

x0
<START>

<START>



before:

h = tanh(Wxh * x + Whh * h)

now:

h = tanh(Wxh * x + Whh * h **+ Wih * v**)

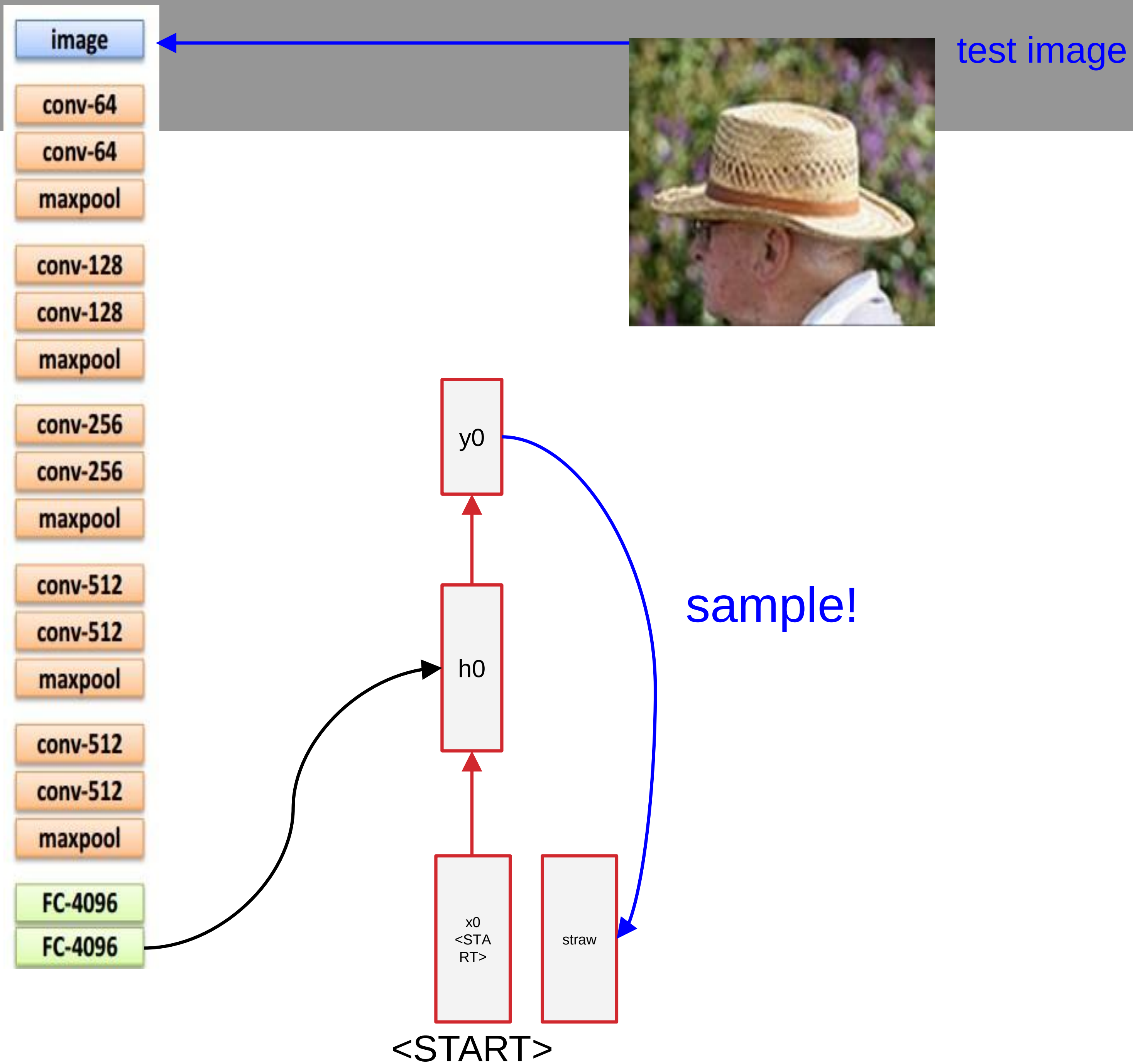
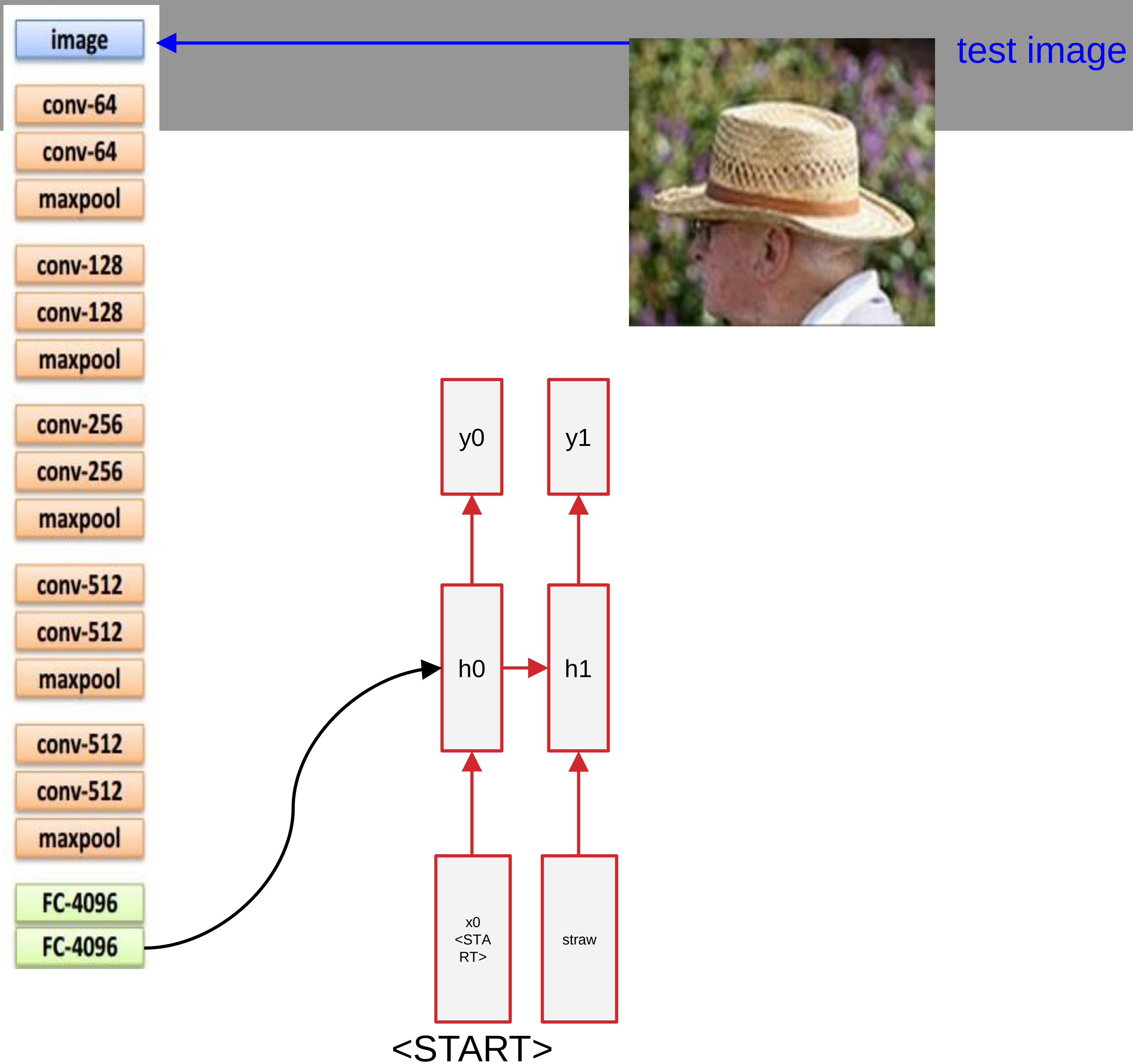


image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
conv-256
conv-256
maxpool
conv-512
conv-512
maxpool
conv-512
conv-512
maxpool
FC-4096
FC-4096
test image
y0
h0
sample!
x0
<STA
RT>
straw
<START>


test image
image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
conv-256
conv-256
maxpool
conv-512
conv-512
maxpool
conv-512
conv-512
maxpool
FC-4096
FC-4096
y0
y1
h0
h1
x0 <STA RT>
straw
<START>

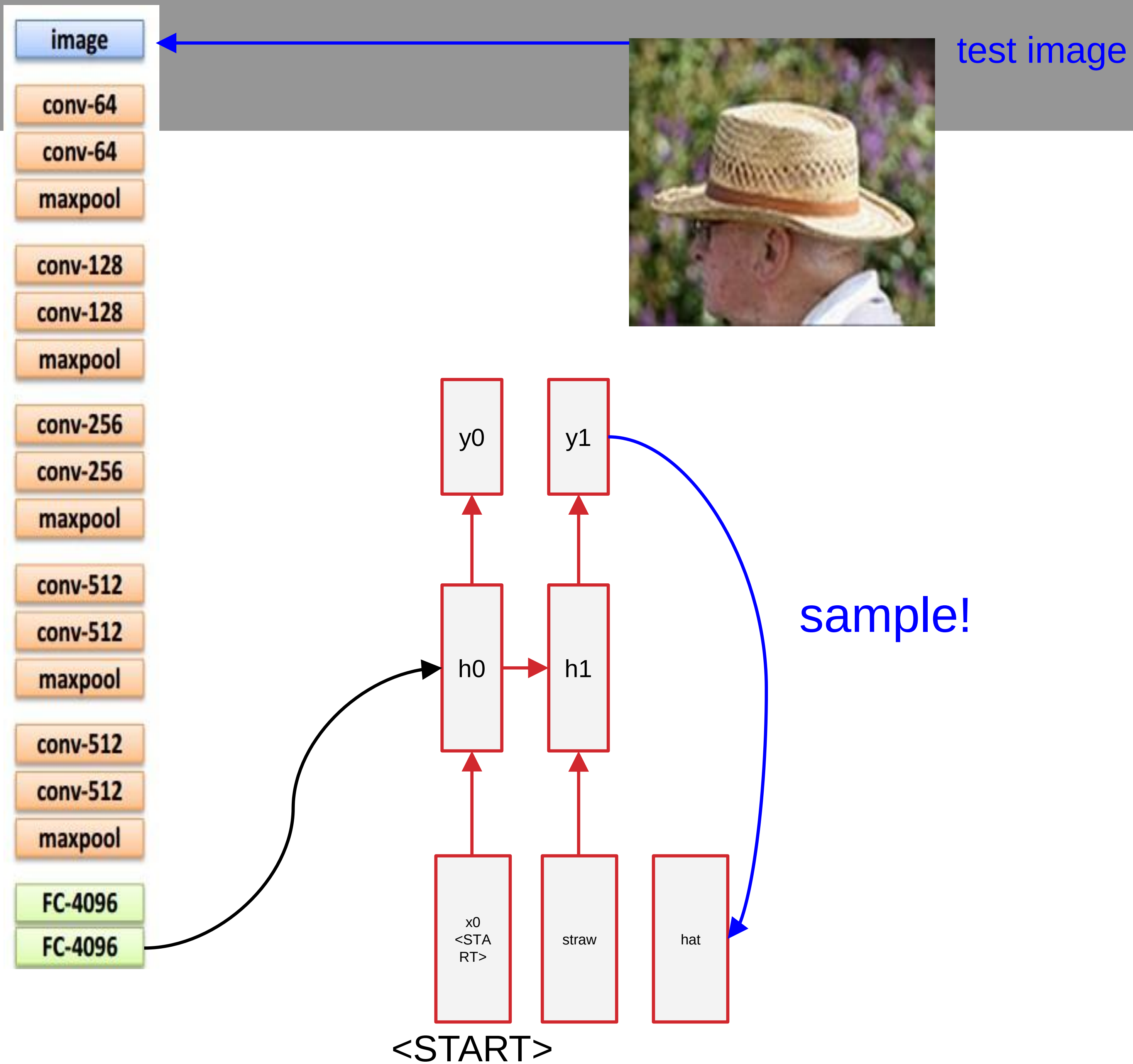


image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
conv-256
conv-256
maxpool
conv-512
conv-512
maxpool
conv-512
conv-512
maxpool
FC-4096
FC-4096
test image
y0
y1
h0
h1
sample!
x0 <START>
straw
hat
<START>

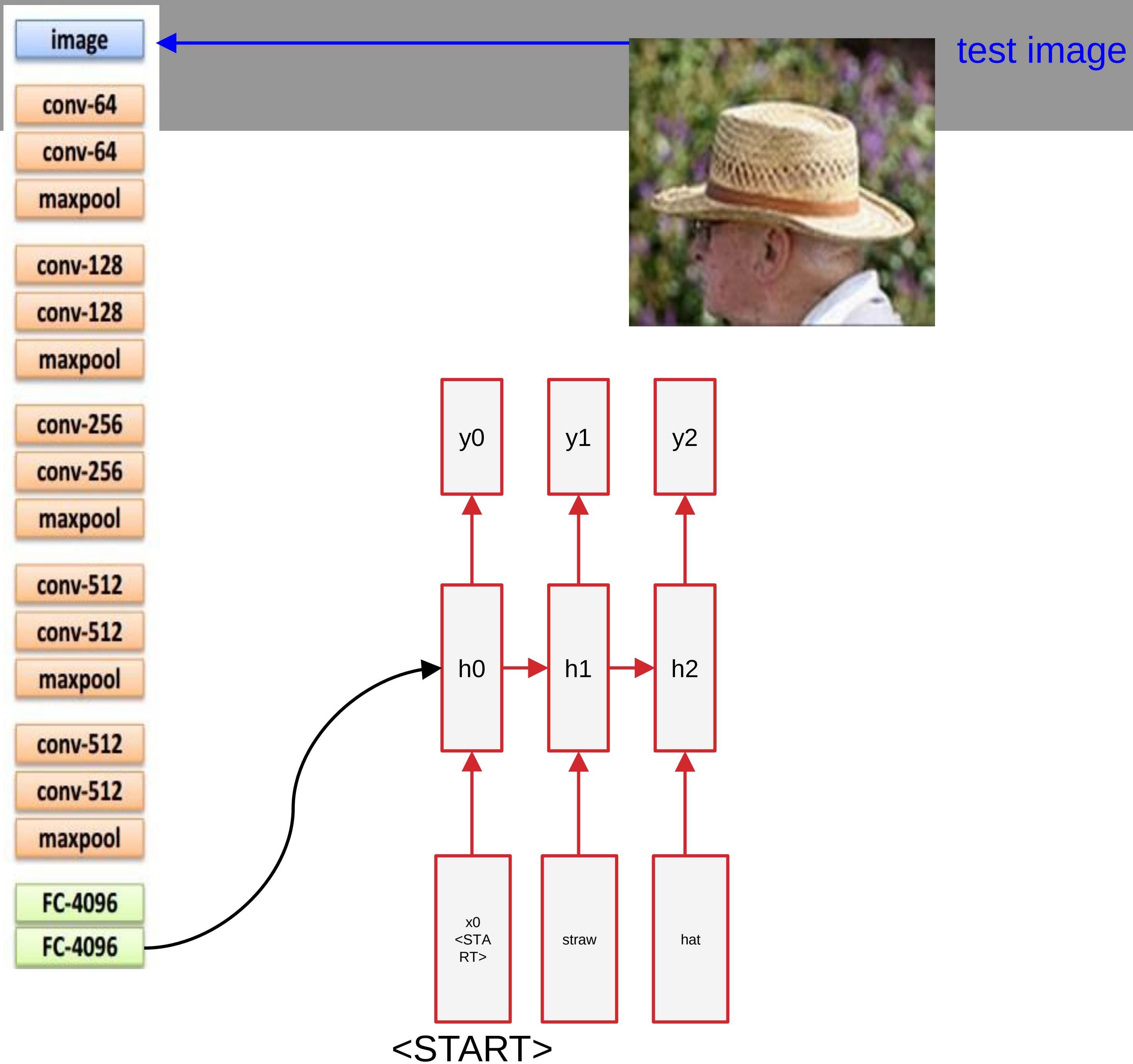


image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
conv-256
conv-256
maxpool
conv-512
conv-512
maxpool
conv-512
conv-512
maxpool
FC-4096
FC-4096
test image
y0
y1
y2
h0
h1
h2
x0
<STA
RT>
straw
hat
<START>

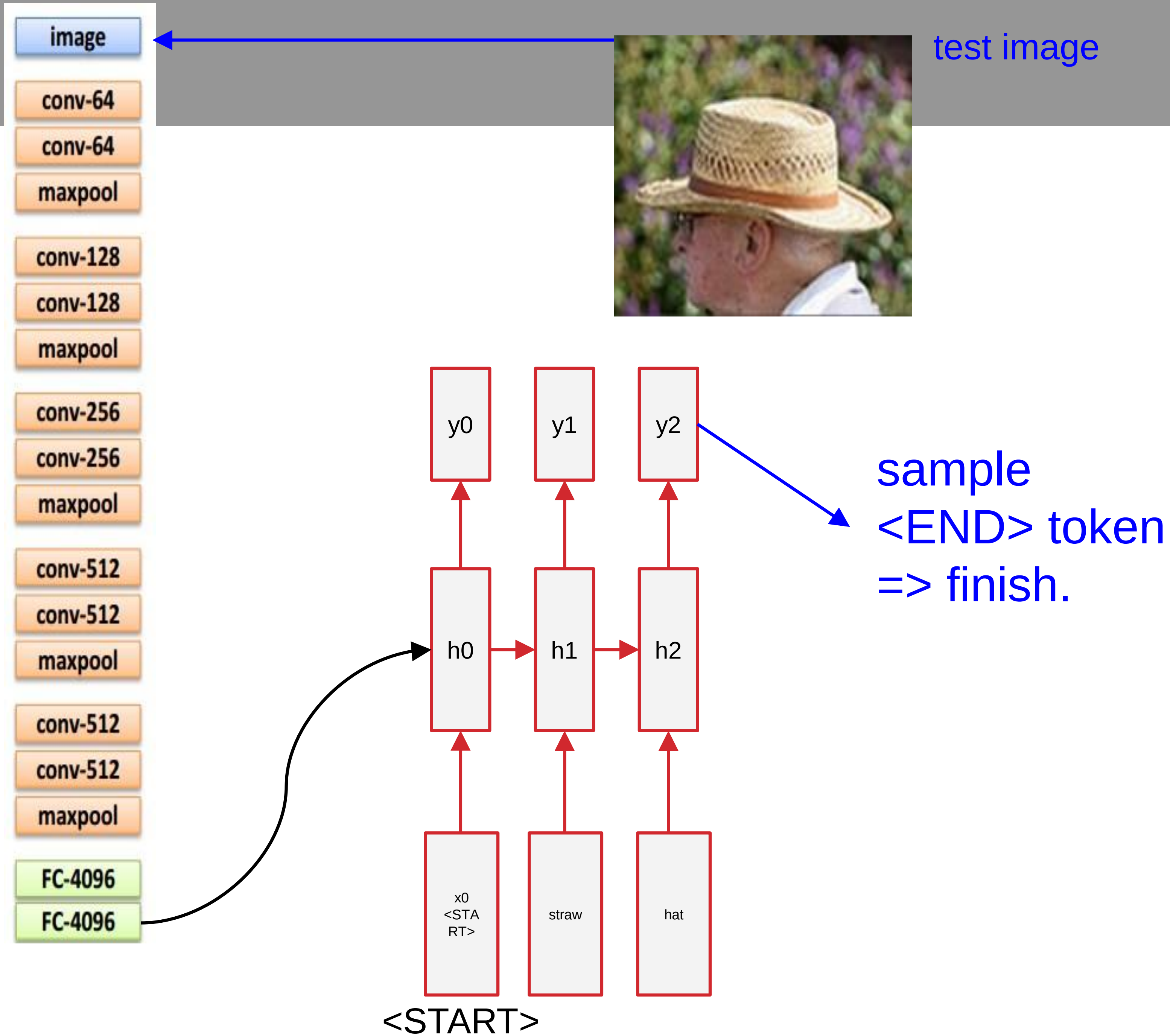


image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
conv-256
conv-256
maxpool
conv-512
conv-512
maxpool
conv-512
conv-512
maxpool
FC-4096
FC-4096
test image
y0
y1
y2
h0
h1
h2
x0 <STA RT>
straw
hat
<START>
sample <END> token => finish.

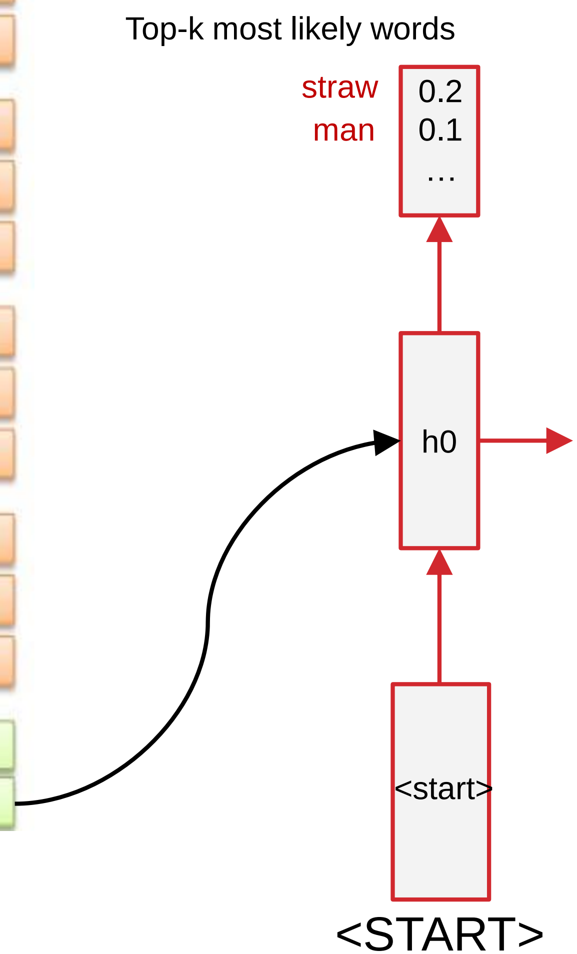
RNN sequence generation

Greedy (most likely) symbol generation is not very effective.

Typically, the top-k sequences generated so far are remembered and the top-k of their one-symbol continuations are kept for the next step (beam search)

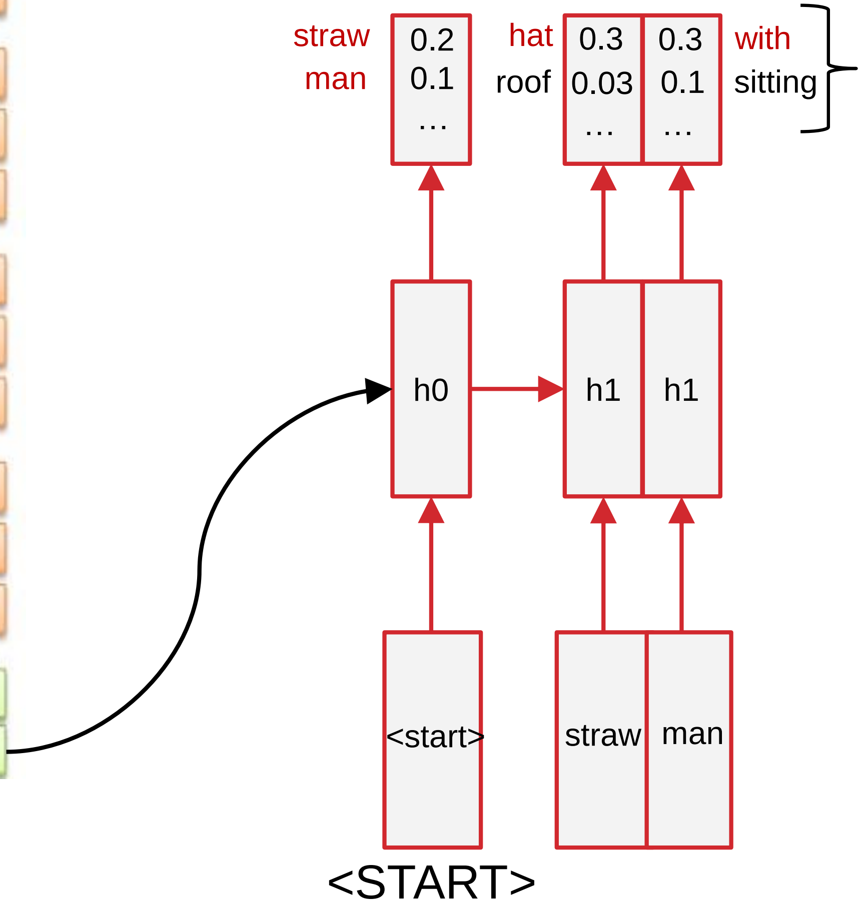
However, k does not have to be very large (7 was used in Karpathy and Fei-Fei 2015).

Beam search with k = 2



Beam search with k = 2

Top-k most likely words

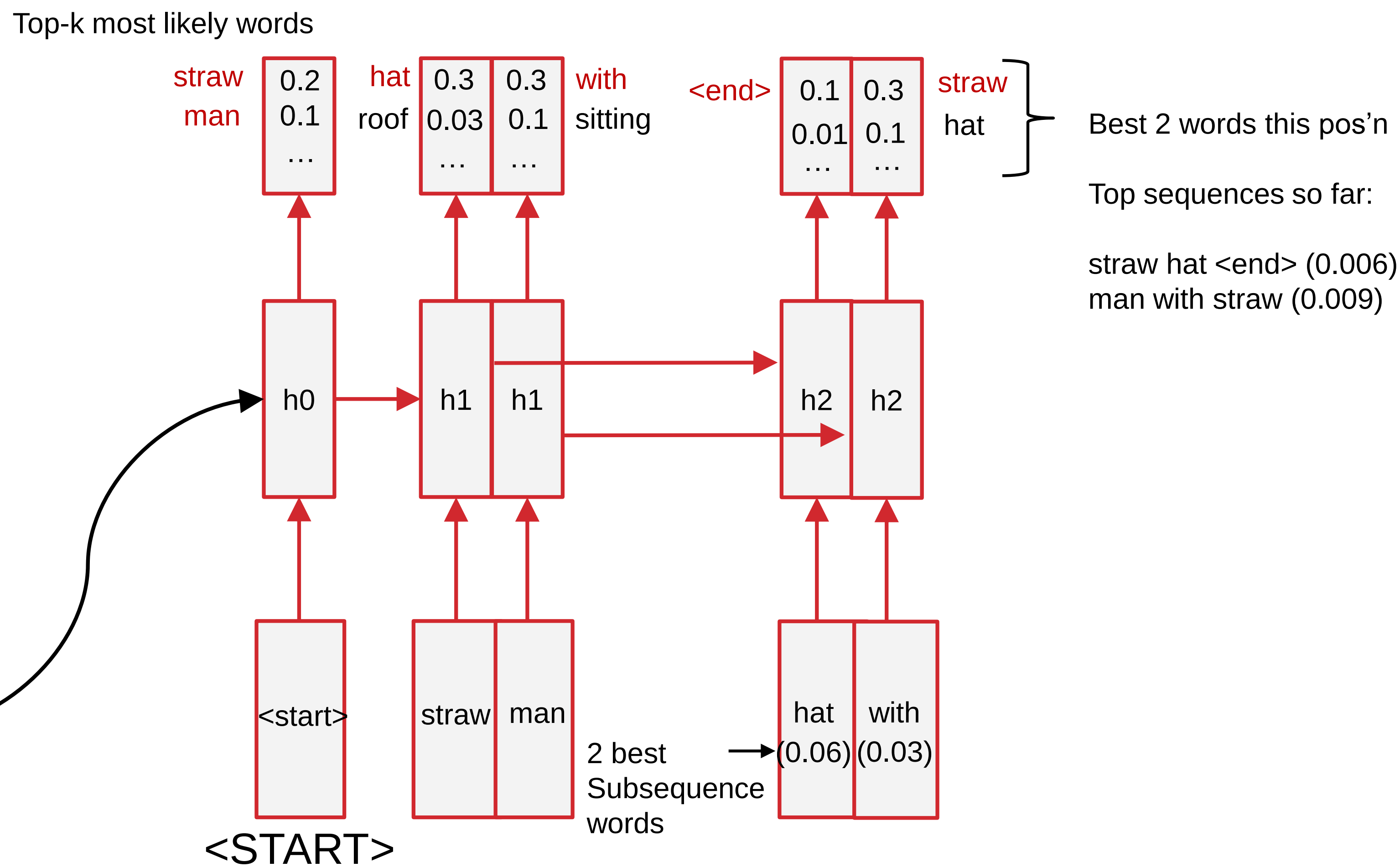


Need Top-k most likely words
Because they might be part of
the top-K most likely subsequences

Top sequences so far are:

straw hat (p = 0.06)
man with (p = 0.03)

Beam search with k = 2



Beam search with k = 2

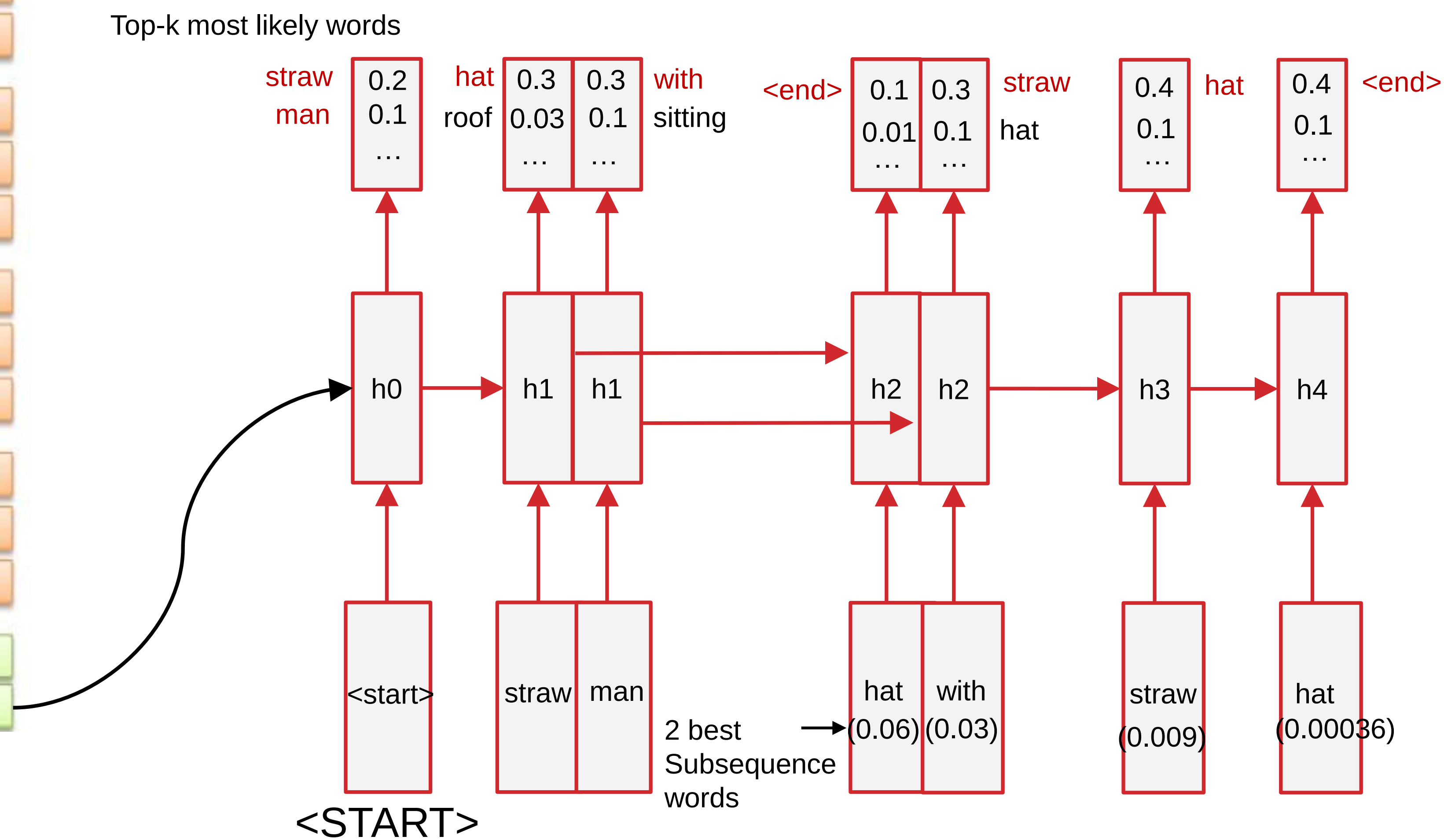


Image Sentence Datasets

a man riding a bike on a dirt path through a forest.
bicyclist raises his fist as he rides on desert dirt trail.
this dirt bike rider is smiling and raising his fist in triumph.
a man riding a bicycle while pumping his fist in the air.
a mountain biker pumps his fist in celebration.

Microsoft COCO

[Tsung-Yi Lin et al. 2014]

mscoco.org

currently:

~120K images

~5 sentences each

"man in black shirt is playing guitar."

"construction worker in orange safety vest is working on road."

"two young girls are playing with lego toy."

"boy is doing backflip on wakeboard."

"man in black shirt is playing guitar."

"construction worker in orange safety vest is working on road."

"two young girls are playing with lego toy."

"boy is doing backflip on wakeboard."

"a young boy is holding a baseball bat."

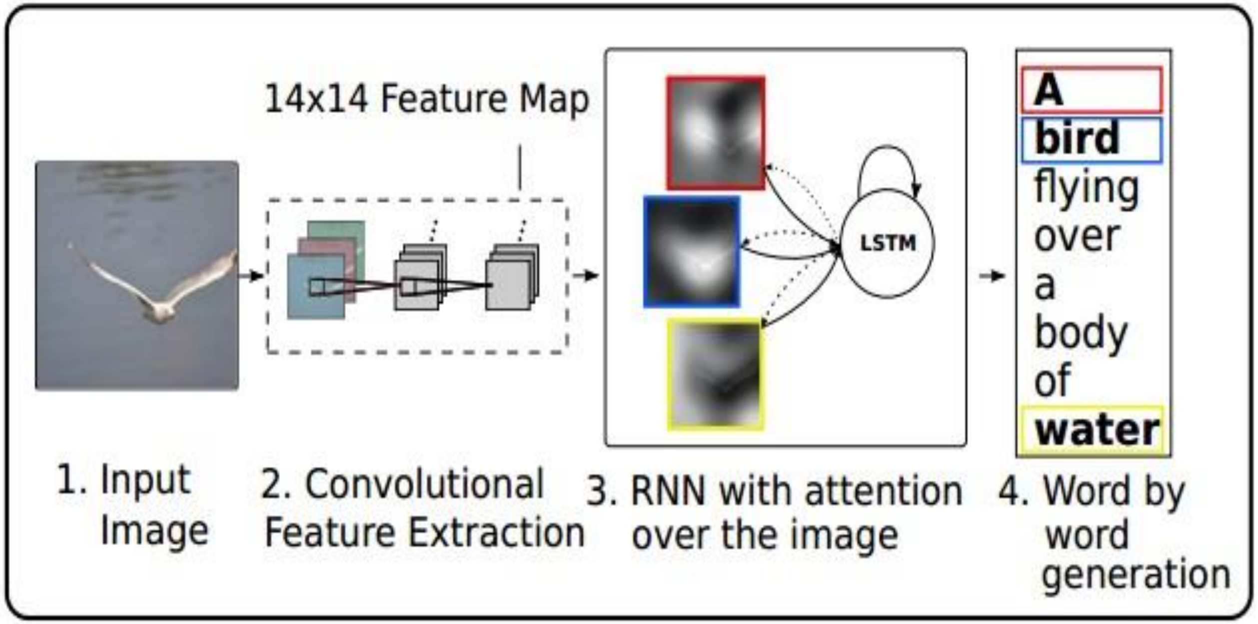
"a cat is sitting on a couch with a remote control."

"a woman holding a teddy bear in front of a mirror."

"a horse is standing in the middle of a road."

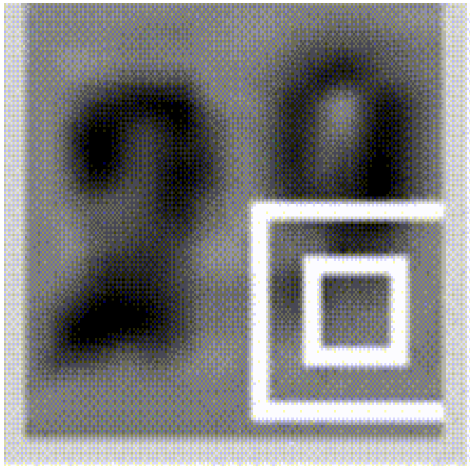
RNN attention networks

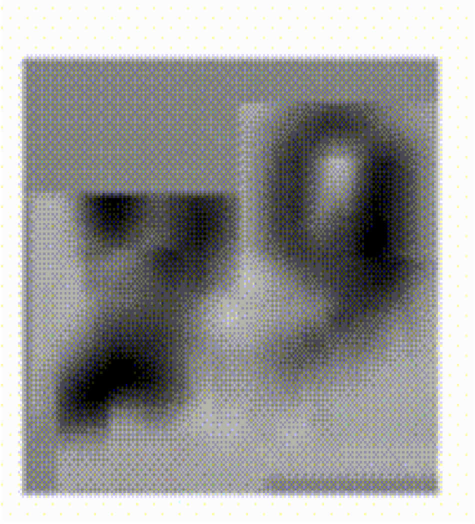
RNN attends spatially to different parts of images while generating each word of the sentence:



Show Attend and Tell, Xu et al., 2015

Sequential Processing of fixed inputs

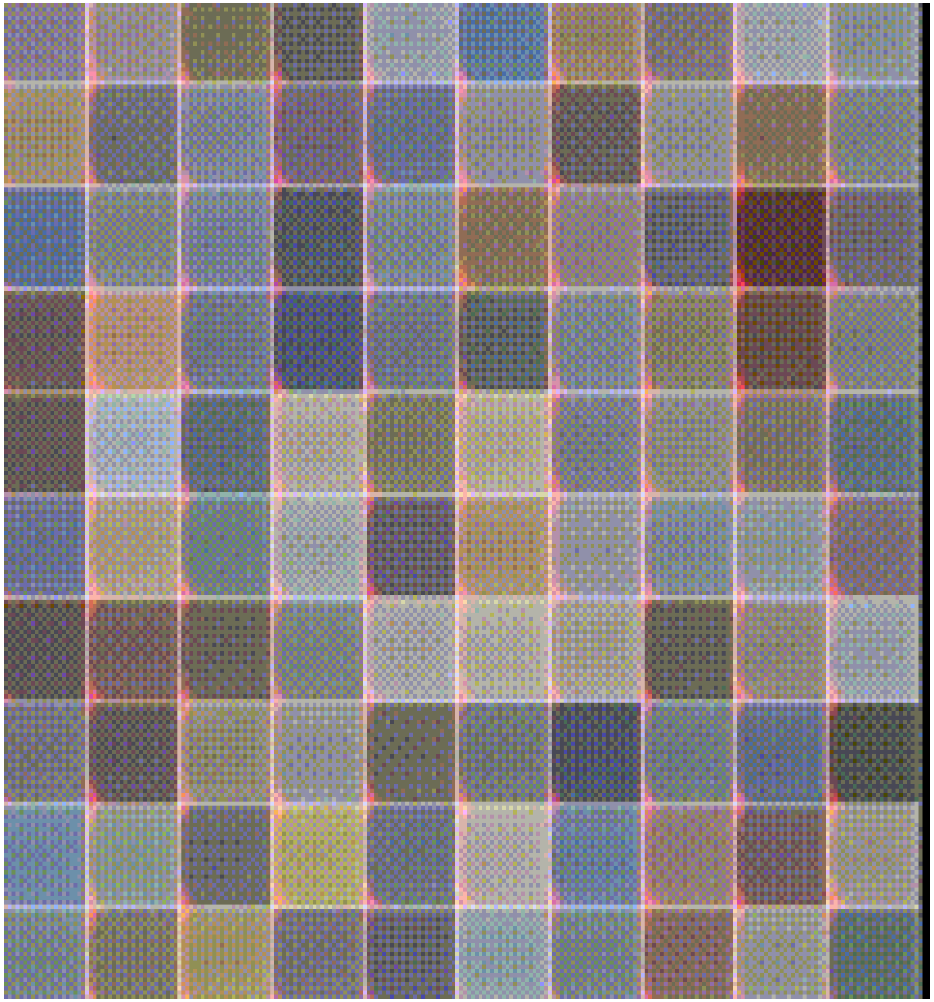




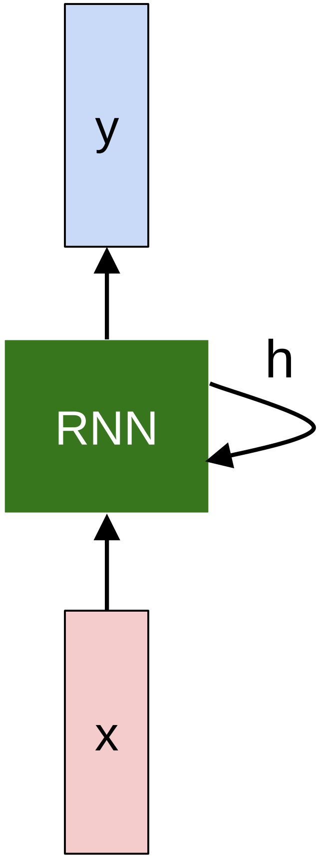
Multiple Object Recognition with Visual Attention, Ba et al. 2015

Sequential Processing of fixed inputs

DRAW: A Recurrent Neural Network For Image Generation, Gregor et al.



Vanilla RNN: Exploding/Vanishing Gradients



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

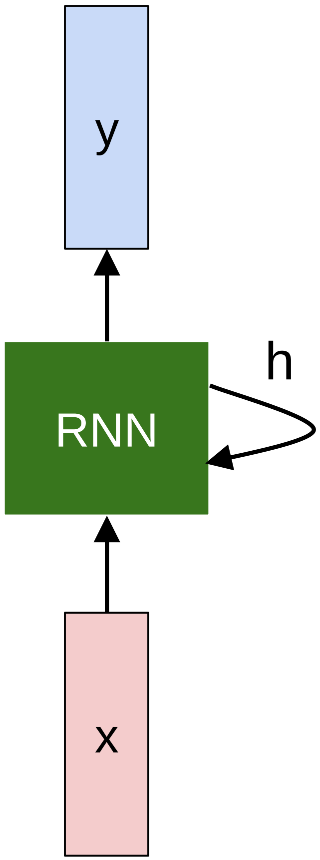
$$y_t = W_{hy}h_t$$

Using Jacobians:

$$J_L(h_{t-1}) = J_L(h_t)\ J_{h_t}(h_{t-1})$$

$$J_{h_t}(h_{t-1}) = \frac{1}{\cosh^2 \hat{h}} W_{hh}$$

Vanilla RNN: Exploding/Vanishing Gradients



Using Jacobians:

$$J_L(h_{t-1}) = J_L(h_t)\ J_{h_t}(h_{t-1})$$

$$J_{h_t}(h_{t-1}) = \frac{1}{\cosh^2 \hat{h}} W_{hh}$$

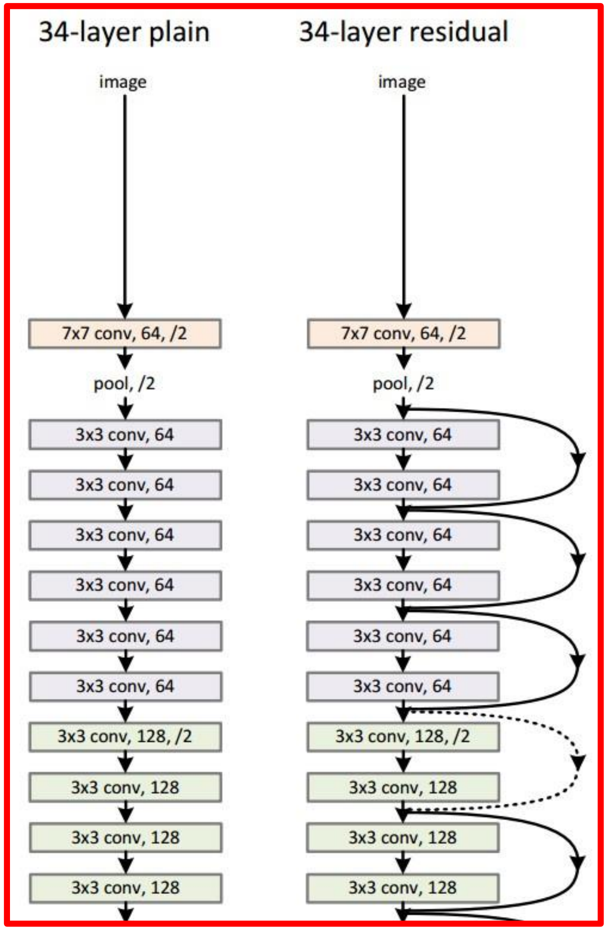
Now $\frac{1}{\cosh^2 \hat{h}}$ is ≤ 1, and W_{hh} can be arbitrarily large/small.

W_{hh} is multiplied by the gradient at the next step, so the gradient across time steps is roughly a power of W_{hh}

If the largest eigenvalue of $W_{hh} > 1$, the gradients will grow exponentially with time (exploding).

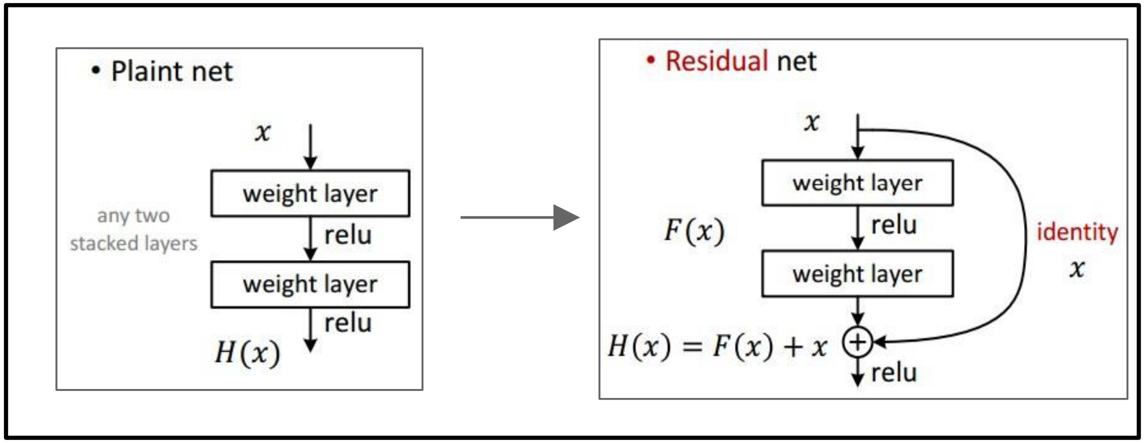
If the largest eigenvalue of $W_{hh} < 1$, the gradients will shrink exponentially with time (vanishing).

Better RNN Memory: LSTMs



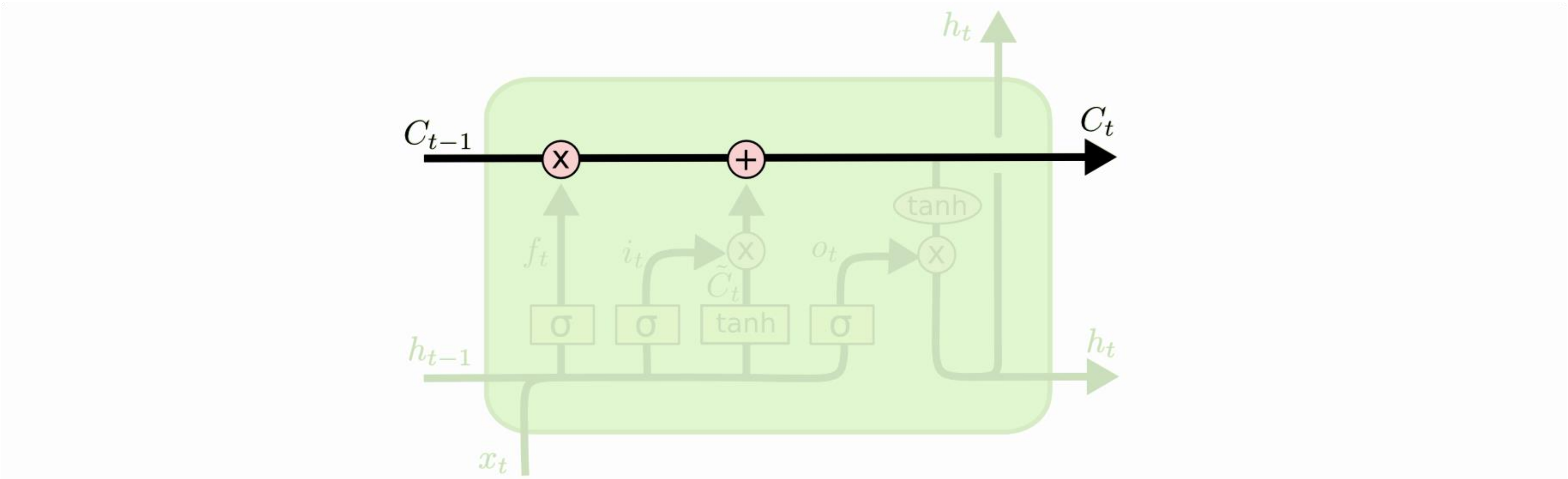
Recall: “PlainNets” vs. ResNets

ResNet are very deep networks. They use residual connections as “hints” to approximate the identity fn.



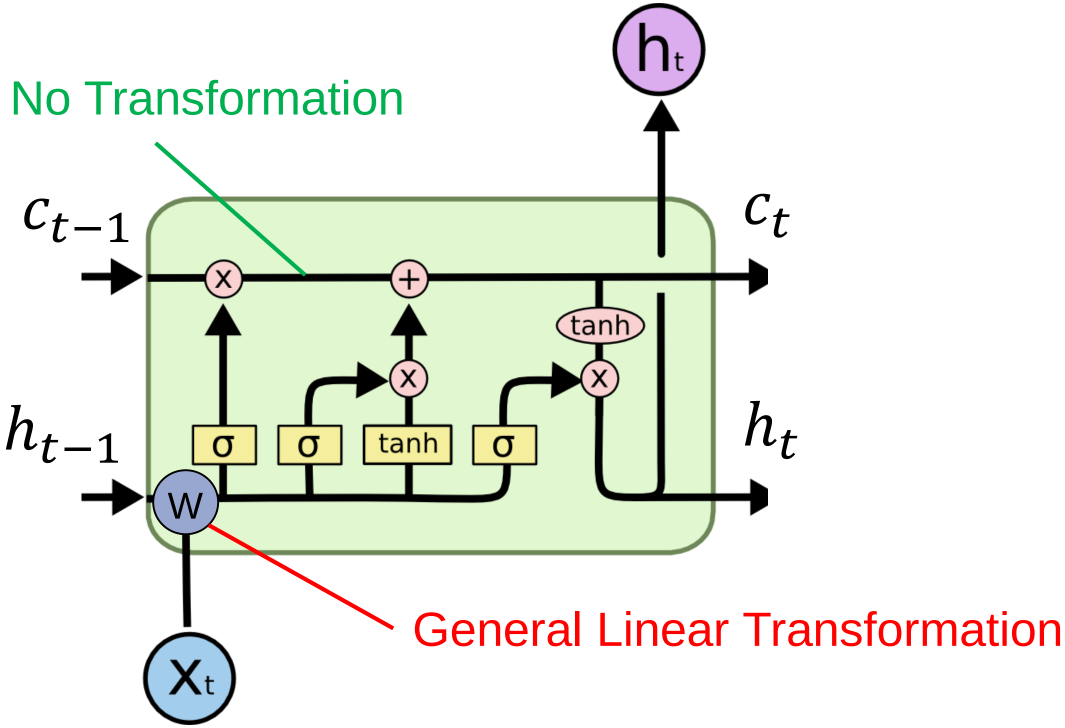
Better RNN Memory: LSTMs

LSTMs (Long Short-Term Memory) units have a memory cell c_i which is gated element-wise (no linear transform) from one time step to the next.



Better RNN Memory: LSTMs

They also have non-linear, linearly transformed hidden states like a standard RNN.



LSTM: Long Short-Term Memory

Vanilla RNN:

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

Input from below
Input from left

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

LSTM: (much more widely used)

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

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

Input from below
Input from left

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

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

LSTM: Anything you can do…

Vanilla RNN:

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

Input from below
Input from left

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

LSTM: (much more widely used)

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

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

Input from below
Input from left

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

An LSTM can emulate a simple RNN by remembering with *i* and *o*

LSTM: Anything you can do…

Vanilla RNN:

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

Input from below
Input from left

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

LSTM: (much more widely used)

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

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

Input from below
Input from left

$$c_t^l = \overset{0}{f} \odot c_{t-1}^l + \overset{1}{i} \odot g$$

$$h_t^l = \underset{1}{o} \odot \tanh(c_t^l)$$

An LSTM can emulate a simple RNN by remembering with *i* and *o*

LSTM: Anything you can do…

Vanilla RNN:

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

Input from below
Input from left

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

LSTM: (much more widely used)

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

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

Input from below
Input from left

$$c_t^l = \overset{0}{f} \odot c_{t-1}^l + \overset{1}{i} \odot g$$

$$h_t^l = \underset{1}{o} \odot \tanh(c_t^l)$$

An LSTM can emulate a simple RNN by remembering with *i* and *o, almost*

LSTM

= “gain” nodes [0,1]

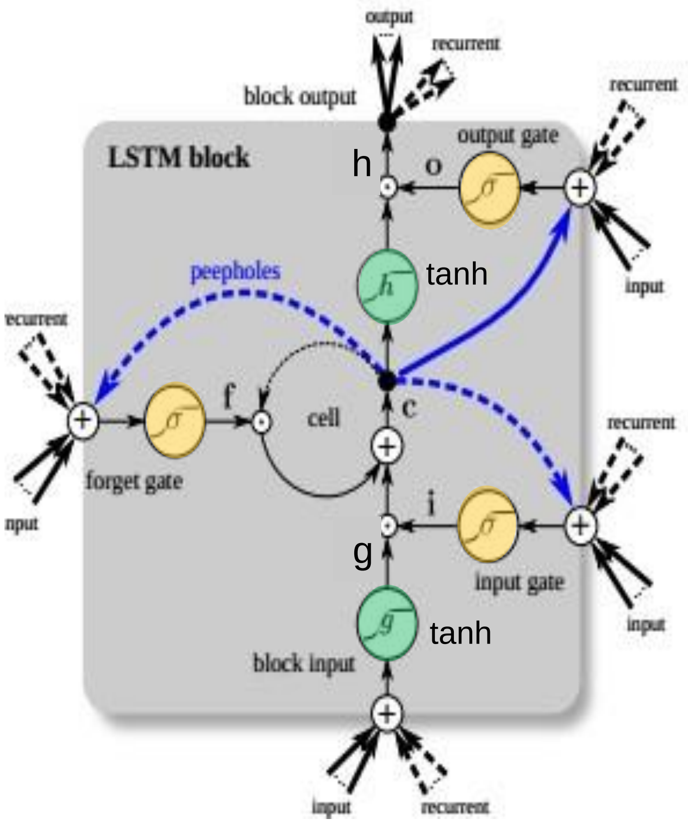
= “data” nodes [-1,1]

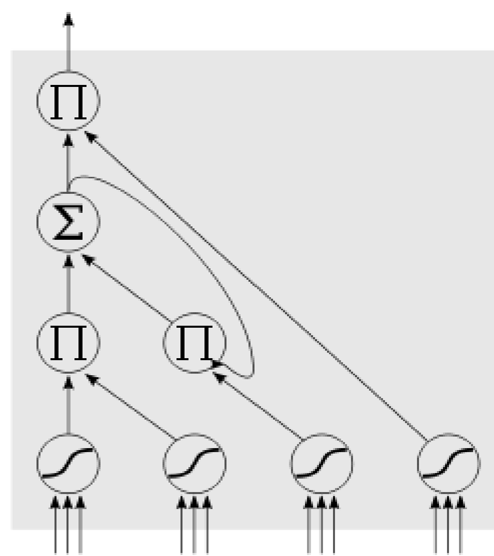
$$\begin{pmatrix} i\\ f\\ o\\ g\end{pmatrix} = \begin{pmatrix}\text{sigm}\\ \text{sigm}\\ \text{sigm}\\ \tanh\end{pmatrix} W^l \begin{pmatrix} h_t^{l-1}\\ h_{t-1}^l\end{pmatrix}$$

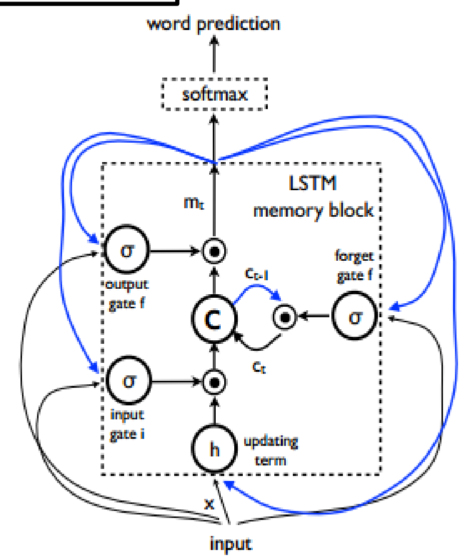
Input from below
Input from left

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

cell Input (left)



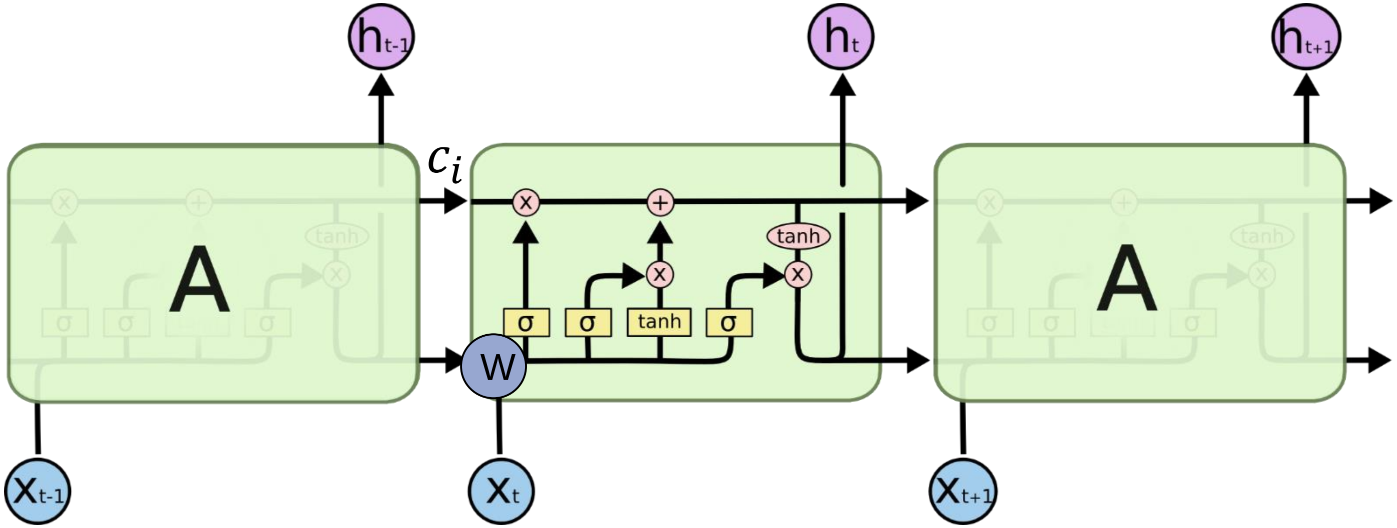




“peepholes” exist in some variations

LSTM Arrays

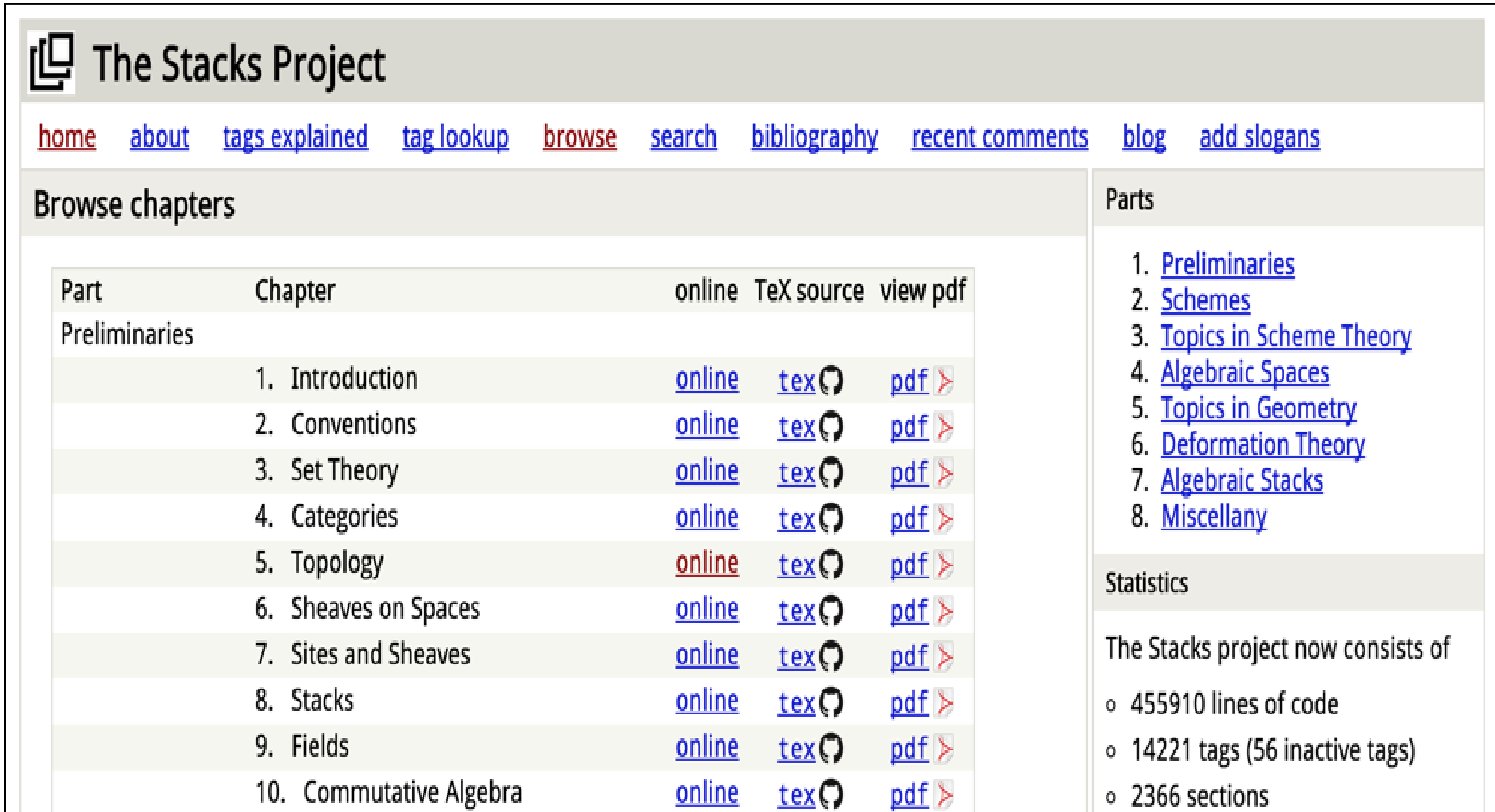
- We now have two recurrent nodes, c_i and h_i.
- h_i plays the role of the output in the simple RNN, and is recurrent.
- The the cell state c_i is the cell's *memory*, it undergoes no transform.
- When we compose LSTMs into arrays, they look like this:



- For stacked arrays, the hidden layers (h_i's) become the inputs (x_i's) for the layer above.

Figure courtesy Chris Olah http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Open source textbook on algebraic geometry



Latex source

http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Generated math

http://karpathy.github.io/2015/05/21/rnn-effectiveness/

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 $X_{spaces,\acute{e}tale}$ which gives an open subspace of X and T equal to S_{Zar}, see Descent, Lemma ??. Namely, by Lemma ?? we see that R is geometrically regular over S.

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

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

Generated math

http://karpathy.github.io/2015/05/21/rnn-effectiveness/

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

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

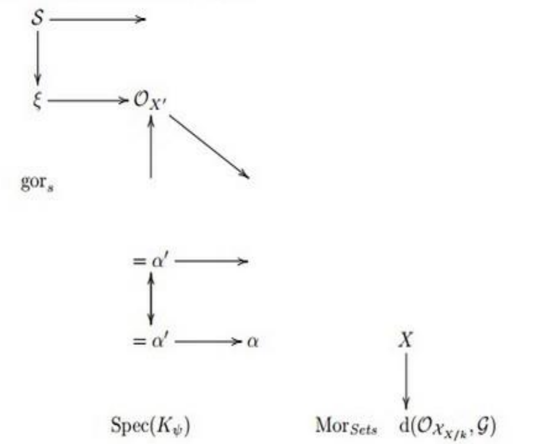
be a morphism of algebraic spaces over S and Y.

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

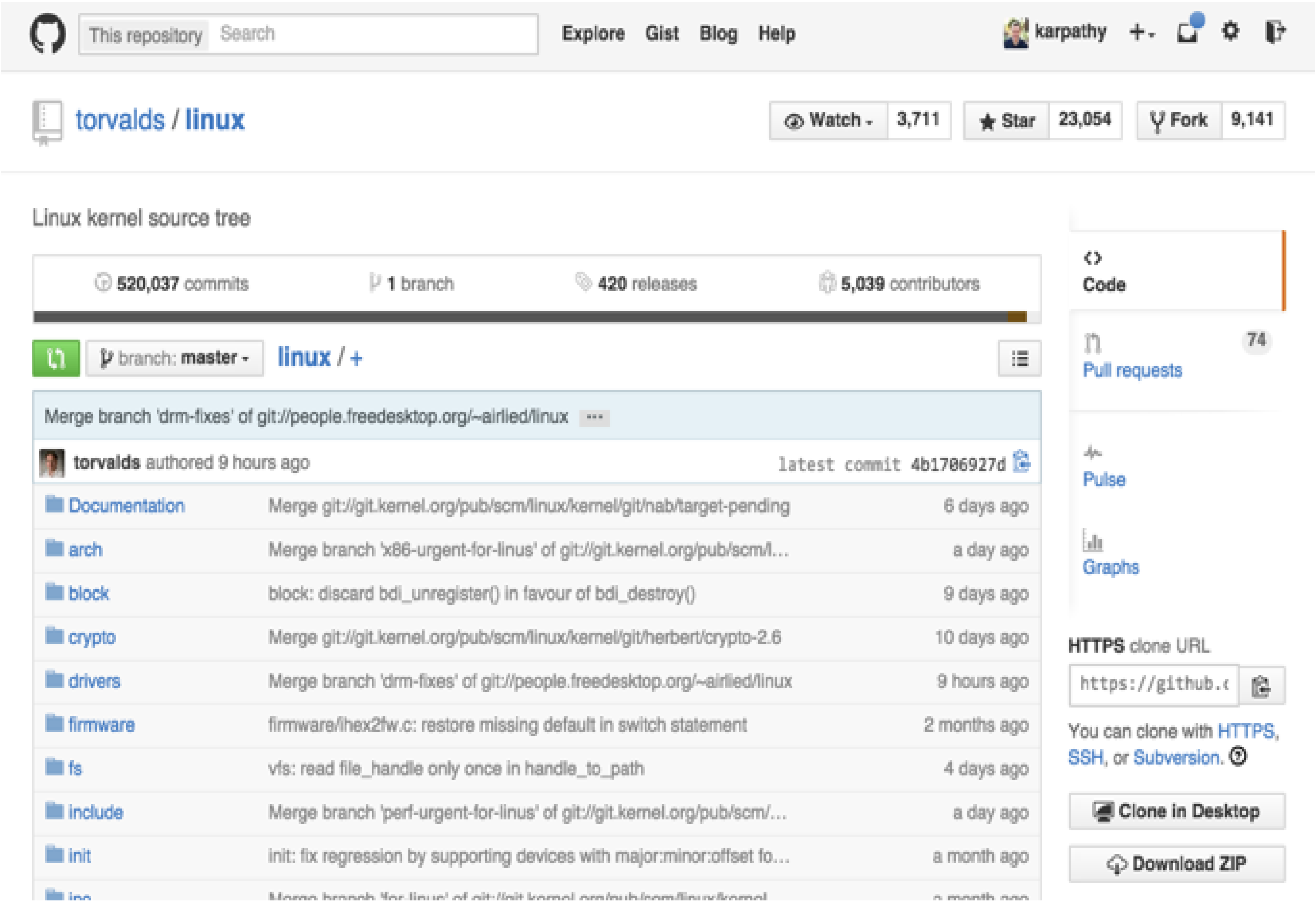
$$\mathcal{O}_{X,x} \longrightarrow \mathcal{F}_{\overline{x}} \ \text{-1}(\mathcal{O}_{X_{\acute{e}tale}}) \longrightarrow \mathcal{O}_{X_\ell}^{-1}\mathcal{O}_{X_\lambda}(\mathcal{O}_{X_\eta}^{\overline{v}})$$

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.

Train on Linux code

http://karpathy.github.io/2015/05/21/rnn-effectiveness/



Generated C code

```
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>
```

Generated C code

```
#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;
}
```

Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]

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

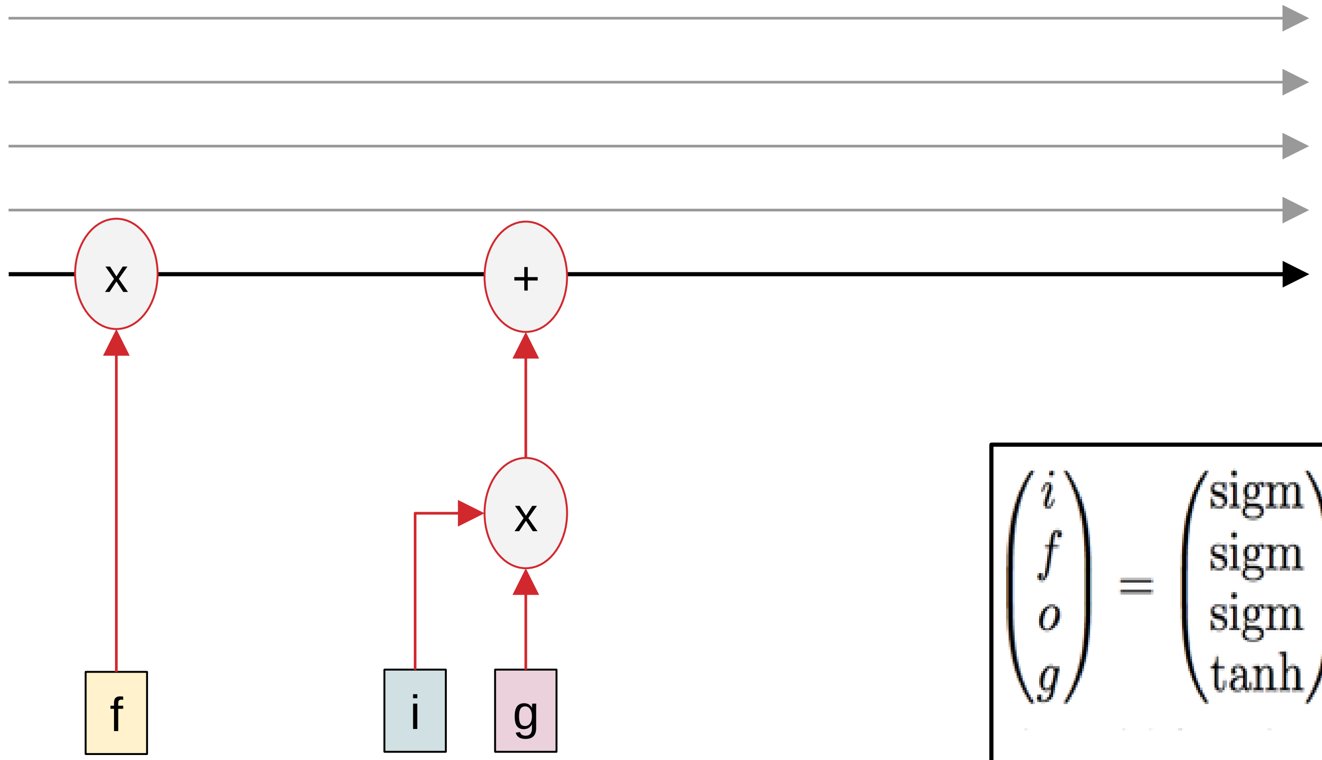
$$c_t^l = \boxed{f \odot c_{t-1}^l} + i \odot g$$

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

Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]

cell state c



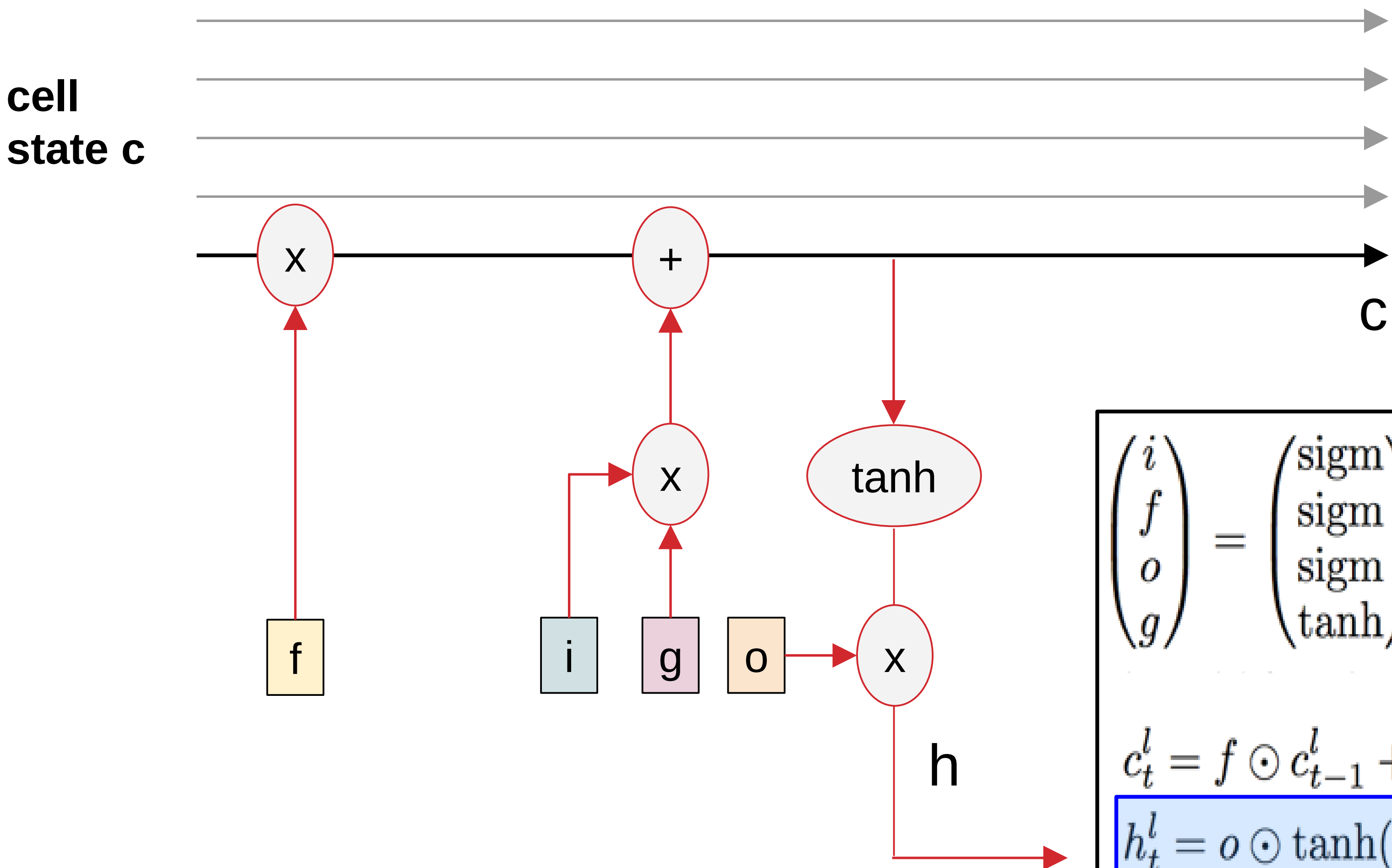
$$\begin{pmatrix} i\\ f\\ o\\ g\end{pmatrix} = \begin{pmatrix}\text{sigm}\\ \text{sigm}\\ \text{sigm}\\ \tanh\end{pmatrix} W^l \begin{pmatrix} h_t^{l-1}\\ h_{t-1}^l\end{pmatrix}$$

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

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

Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]



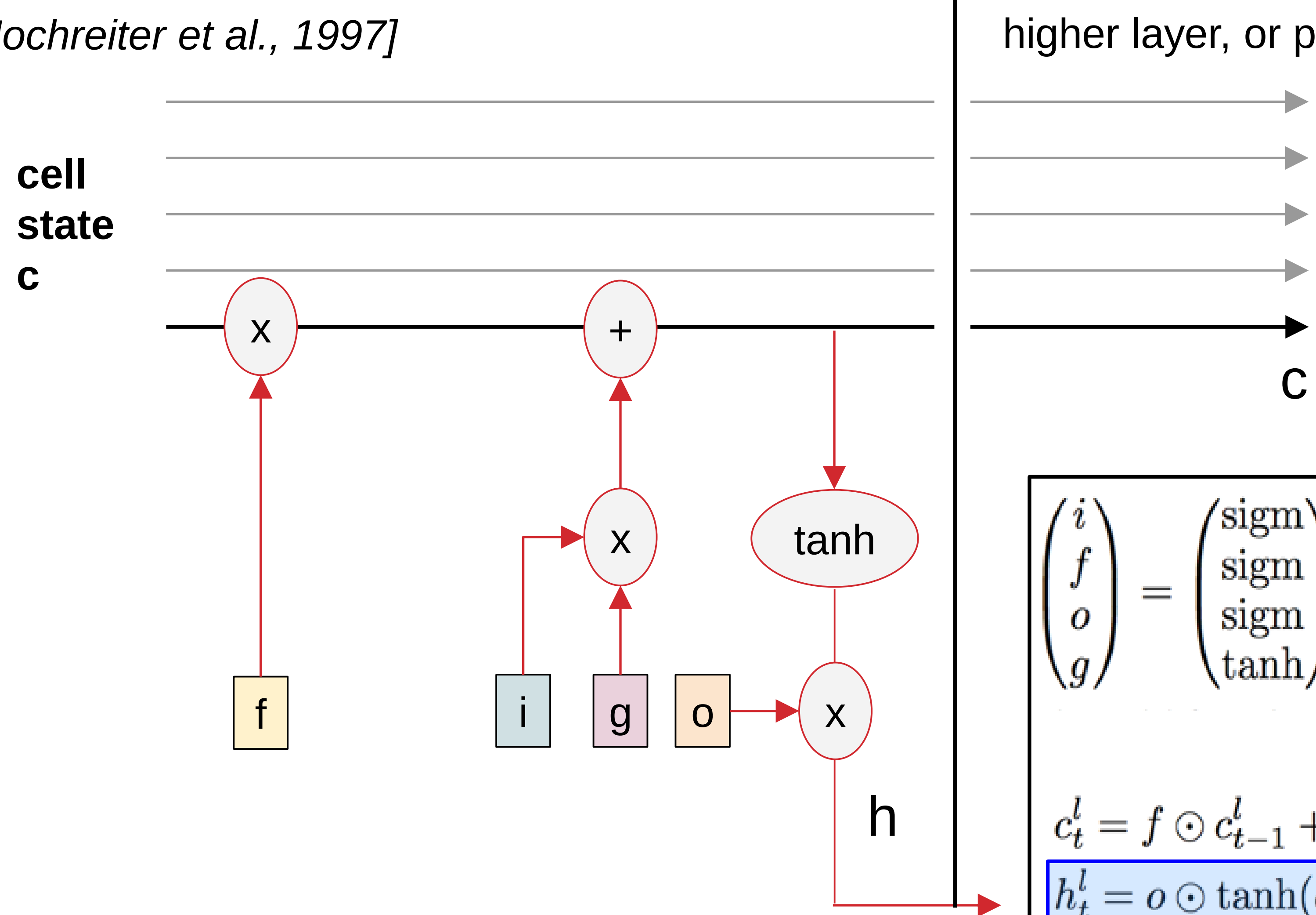
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \tanh \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$

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

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

Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]

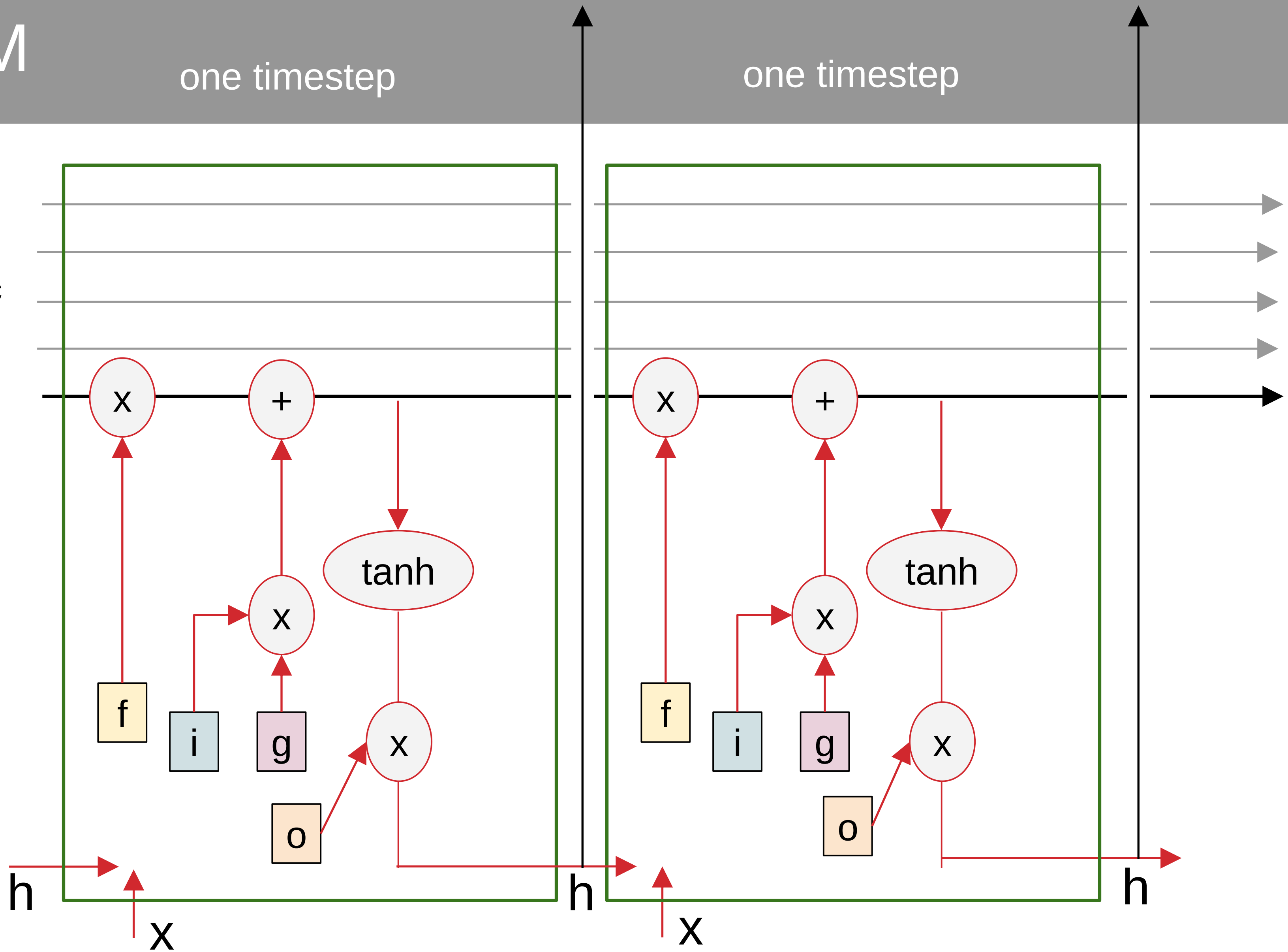


$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \tanh \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$

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

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

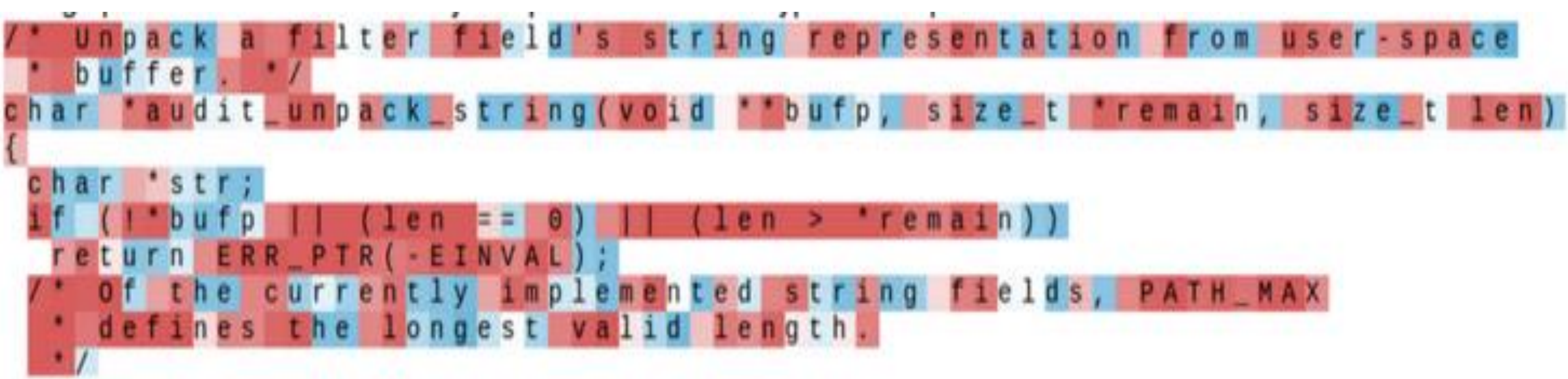
LSTM


one timestep
one timestep
cell
state c
x
+
tanh
x
f
i
g
x
o
h
x
h
x
+
tanh
x
f
i
g
x
o
x
h

LSTM Summary

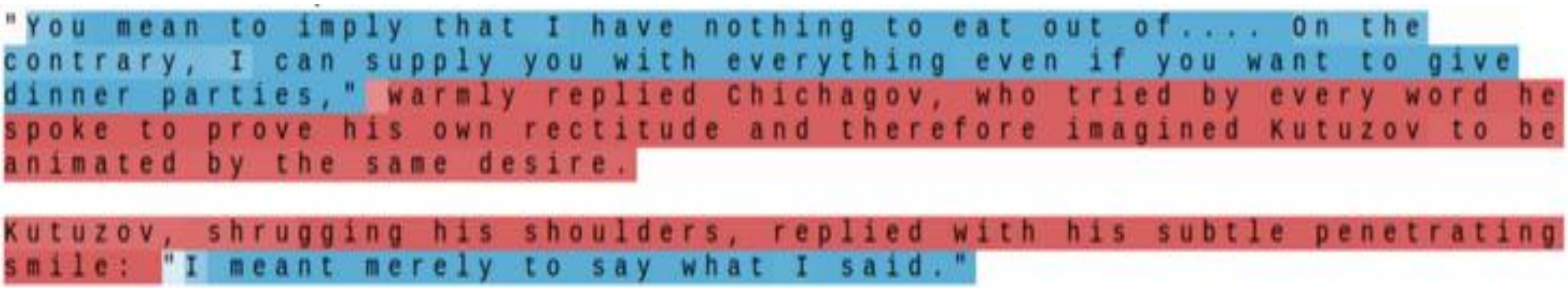
1. Decide what to forget
2. Decide what new things to remember
3. Decide what to output

Searching for interpretable cells



[Visualizing and Understanding Recurrent Networks, Andrej Karpathy, Justin Johnson*, Li Fei-Fei]*

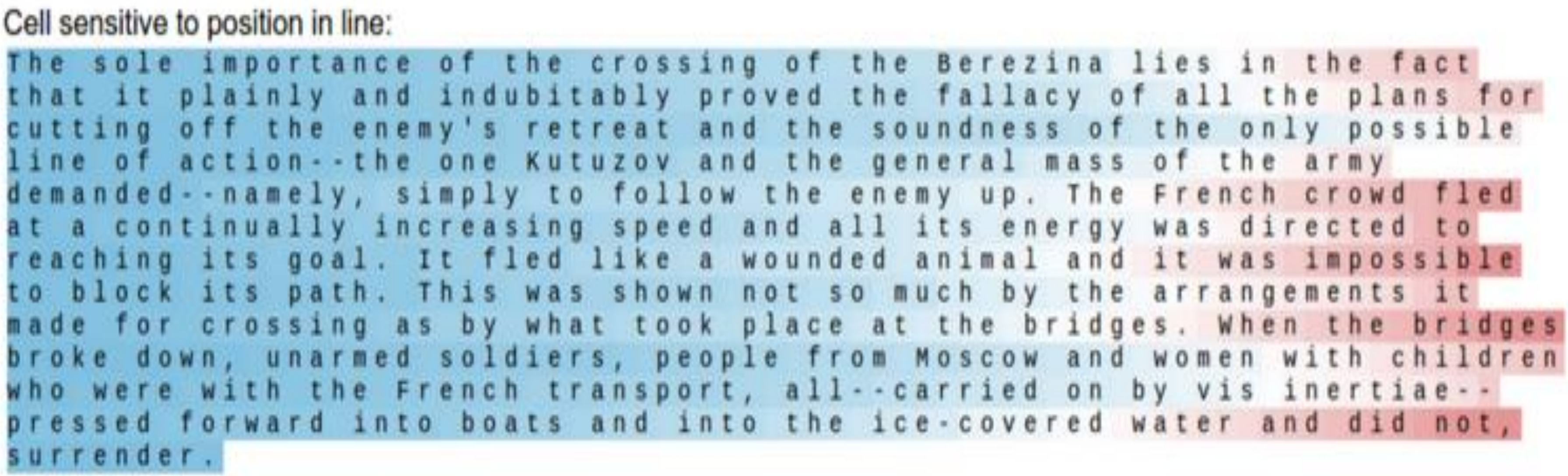
Searching for interpretable cells



quote detection cell

(LSTM, tanh(c), red = -1, blue = +1)

Searching for interpretable cells

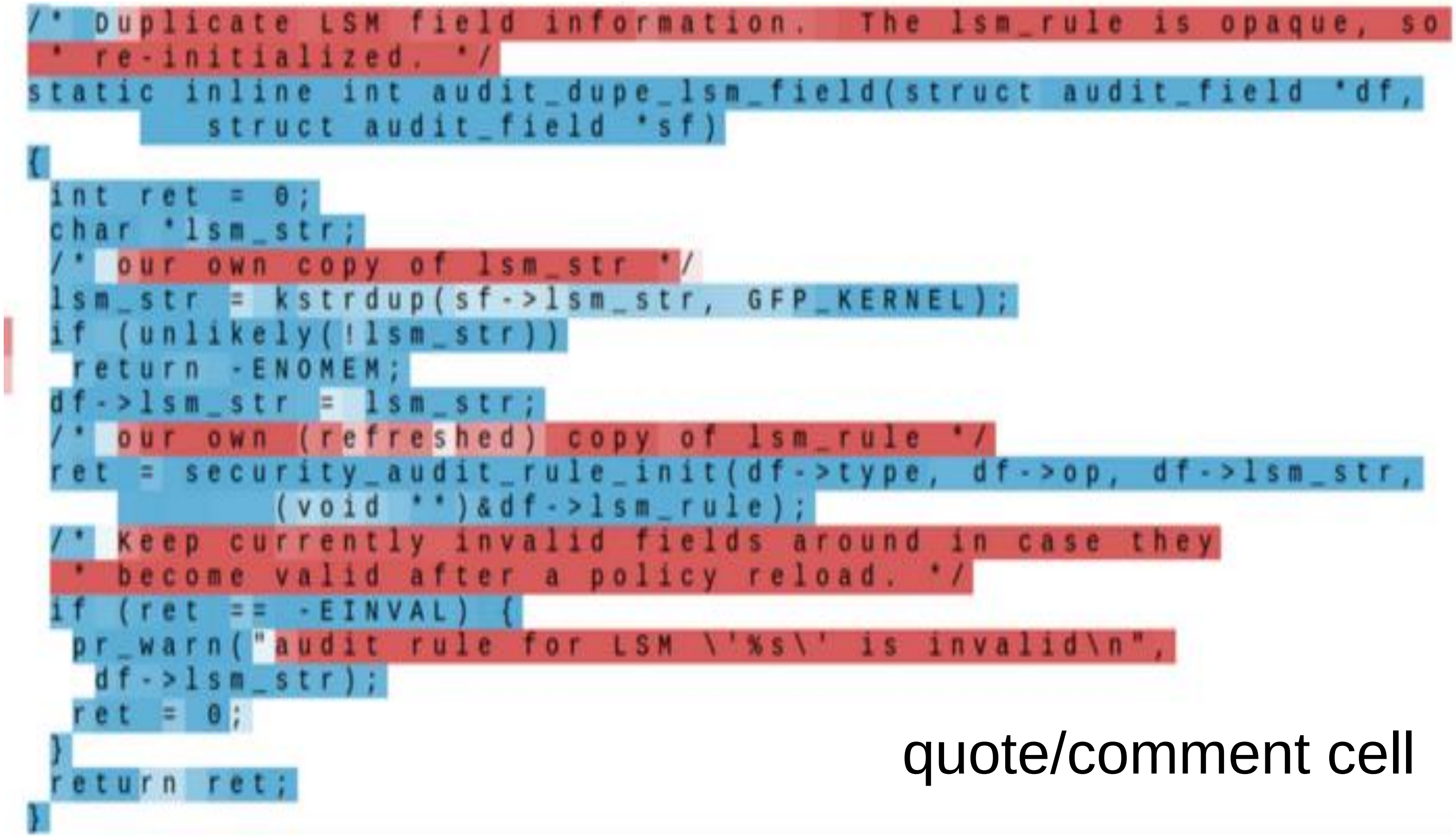


line length tracking cell

Searching for interpretable cells

if statement cell

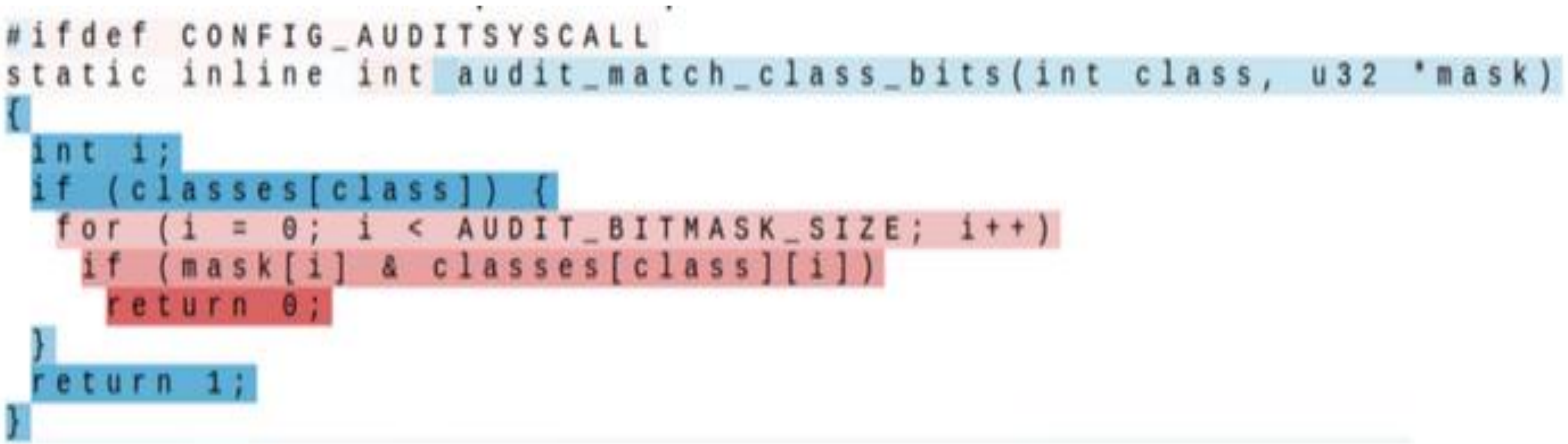
Searching for interpretable cells



quote/comment cell

Based on cs231n by Fei-Fei Li & Andrej Karpathy & Justin Johnson

Searching for interpretable cells



code depth cell

LSTM stability

How fast can the cell value c grow with time?

How fast can the backpropagated gradient of c grow with time?

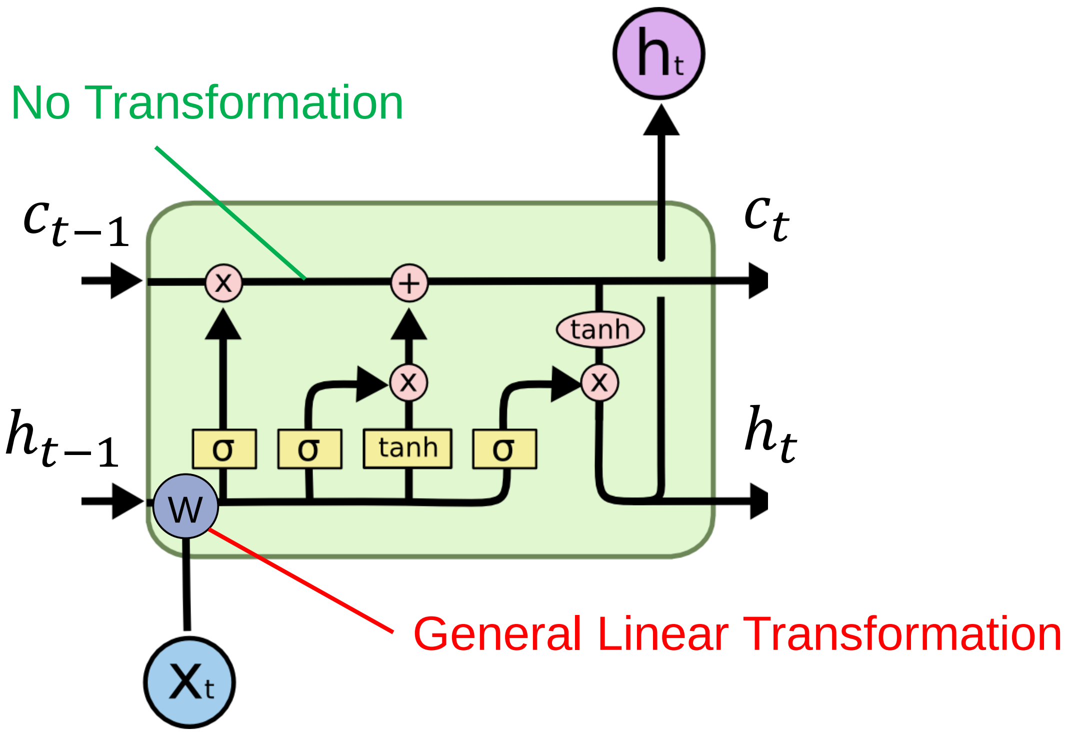
LSTM stability

How fast can the cell value c grow with time?
Linear

How fast can the backpropagated gradient of c grow with time?
Linear

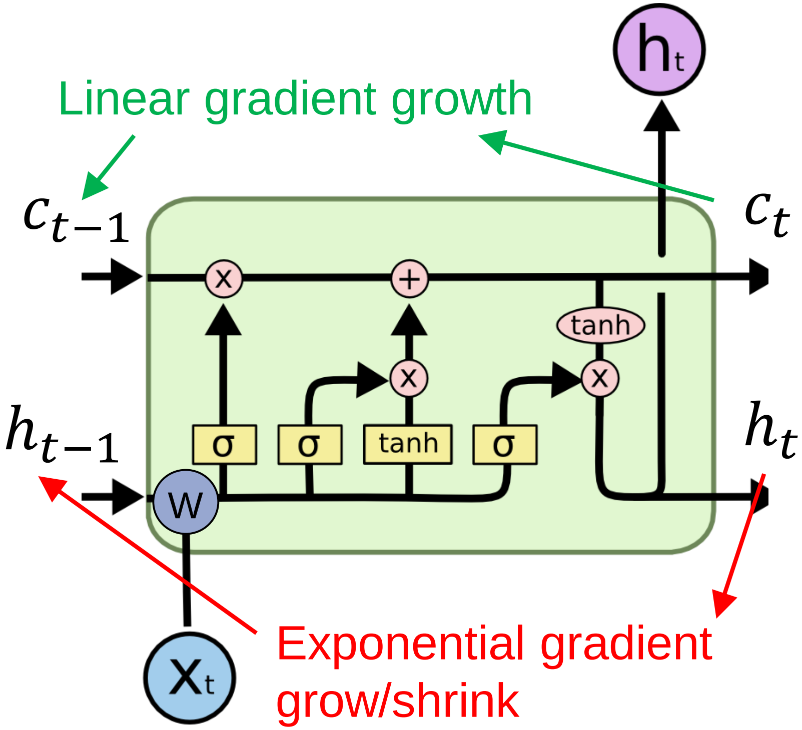
Better RNN Memory: LSTMs

Remember that the h path has a linear transform W_{hh} at each node.



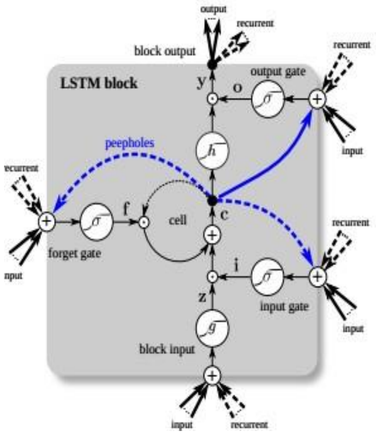
Better RNN Memory: LSTMs

Gradients are well-behaved along the c path but not the h path. Luckily LSTMs learn to rely mostly on c for long-term memory.



LSTM variants and friends

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



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

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

$$\boxed{\begin{aligned} r_t &= \mathrm{sigm}\left(W_{\mathrm{xr}}x_t + W_{\mathrm{hr}}h_{t-1} + b_{\mathrm{r}}\right) \\ z_t &= \mathrm{sigm}(W_{\mathrm{xz}}x_t + W_{\mathrm{hz}}h_{t-1} + b_{\mathrm{z}}) \\ \tilde{h}_t &= \tanh(W_{\mathrm{xh}}x_t + W_{\mathrm{hh}}(r_t \odot h_{t-1}) + b_{\mathrm{h}}) \\ h_t &= z_t \odot h_{t-1} + (1 - z_t) \odot \tilde{h}_t \end{aligned}}$$

MUT1:

$$\begin{aligned} z &= \mathrm{sigm}(W_{\mathrm{xz}}x_t + b_{\mathrm{z}}) \\ r &= \mathrm{sigm}(W_{\mathrm{xr}}x_t + W_{\mathrm{hr}}h_t + b_{\mathrm{r}}) \\ h_{t+1} &= \tanh(W_{\mathrm{hh}}(r \odot h_t) + \tanh(x_t) + b_{\mathrm{h}}) \odot z \\ &+ h_t \odot (1 - z) \end{aligned}$$

MUT2:

$$\begin{aligned} z &= \mathrm{sigm}(W_{\mathrm{xz}}x_t + W_{\mathrm{hz}}h_t + b_{\mathrm{z}}) \\ r &= \mathrm{sigm}(x_t + W_{\mathrm{hr}}h_t + b_{\mathrm{r}}) \\ h_{t+1} &= \tanh(W_{\mathrm{hh}}(r \odot h_t) + W_{xh}x_t + b_{\mathrm{h}}) \odot z \\ &+ h_t \odot (1 - z) \end{aligned}$$

MUT3:

$$\begin{aligned} z &= \mathrm{sigm}(W_{\mathrm{xz}}x_t + W_{\mathrm{hz}}\tanh(h_t) + b_{\mathrm{z}}) \\ r &= \mathrm{sigm}(W_{\mathrm{xr}}x_t + W_{\mathrm{hr}}h_t + b_{\mathrm{r}}) \\ h_{t+1} &= \tanh(W_{\mathrm{hh}}(r \odot h_t) + W_{xh}x_t + b_{\mathrm{h}}) \odot z \\ &+ h_t \odot (1 - z) \end{aligned}$$

LSTM variants and friends

These designs combine the function of c and h, and (similar to ResNet) always include an identity path to the previous h (memory path).

MUT1:

$$
\begin{aligned}
z &= \mathrm{sigm}(W_{\mathrm{xz}} x_t + b_{\mathrm{z}}) \\
r &= \mathrm{sigm}(W_{\mathrm{xr}} x_t + W_{\mathrm{hr}} h_t + b_{\mathrm{r}}) \\
h_{t+1} &= \tanh(W_{\mathrm{hh}}(r \odot h_t) + \tanh(x_t) + b_{\mathrm{h}}) \odot z \\
&+ h_t \odot (1 - z)
\end{aligned}
$$

MUT2:

$$
\begin{aligned}
z &= \mathrm{sigm}(W_{\mathrm{xz}} x_t + W_{\mathrm{hz}} h_t + b_{\mathrm{z}}) \\
r &= \mathrm{sigm}(x_t + W_{\mathrm{hr}} h_t + b_{\mathrm{r}}) \\
h_{t+1} &= \tanh(W_{\mathrm{hh}}(r \odot h_t) + W_{xh} x_t + b_{\mathrm{h}}) \odot z \\
&+ h_t \odot (1 - z)
\end{aligned}
$$

MUT3:

$$
\begin{aligned}
z &= \mathrm{sigm}(W_{\mathrm{xz}} x_t + W_{\mathrm{hz}} \tanh(h_t) + b_{\mathrm{z}}) \\
r &= \mathrm{sigm}(W_{\mathrm{xr}} x_t + W_{\mathrm{hr}} h_t + b_{\mathrm{r}}) \\
h_{t+1} &= \tanh(W_{\mathrm{hh}}(r \odot h_t) + W_{xh} x_t + b_{\mathrm{h}}) \odot z \\
&+ h_t \odot (1 - z)
\end{aligned}
$$

GRU:

$$
\begin{aligned}
r_t &= \mathrm{sigm}\left(W_{\mathrm{xr}} x_t + W_{\mathrm{hr}} h_{t-1} + b_{\mathrm{r}}\right) \\
z_t &= \mathrm{sigm}(W_{\mathrm{xz}} x_t + W_{\mathrm{hz}} h_{t-1} + b_{\mathrm{z}}) \\
\tilde{h}_t &= \tanh(W_{\mathrm{xh}} x_t + W_{\mathrm{hh}}(r_t \odot h_{t-1}) + b_{\mathrm{h}}) \\
h_t &= z_t \odot h_{t-1} + (1 - z_t) \odot \tilde{h}_t
\end{aligned}
$$

LSTM variants and friends

They also combine the functions of f (forgetting gate) and i (input) in a single z gate.

The more you add from the current state (large z) the less $(1-z)$ you remember from previous h, and vice versa.

GRU:

$$\begin{aligned}
r_t &= \mathrm{sigm}\left(W_{\mathrm{xr}}x_t + W_{\mathrm{hr}}h_{t-1} + b_{\mathrm{r}}\right)\\
z_t &= \mathrm{sigm}(W_{\mathrm{xz}}x_t + W_{\mathrm{hz}}h_{t-1} + b_{\mathrm{z}})\\
\tilde{h}_t &= \tanh(W_{\mathrm{xh}}x_t + W_{\mathrm{hh}}(r_t \odot h_{t-1}) + b_{\mathrm{h}})\\
h_t &= z_t \odot h_{t-1} + (1 - z_t) \odot \tilde{h}_t
\end{aligned}$$

MUT1:

$$\begin{aligned}
z &= \mathrm{sigm}(W_{\mathrm{xz}}x_t + b_{\mathrm{z}})\\
r &= \mathrm{sigm}(W_{\mathrm{xr}}x_t + W_{\mathrm{hr}}h_t + b_{\mathrm{r}})\\
h_{t+1} &= \tanh(W_{\mathrm{hh}}(r \odot h_t) + \tanh(x_t) + b_{\mathrm{h}}) \odot z\\
&+ h_t \odot (1 - z)
\end{aligned}$$

MUT2:

$$\begin{aligned}
z &= \mathrm{sigm}(W_{\mathrm{xz}}x_t + W_{\mathrm{hz}}h_t + b_{\mathrm{z}})\\
r &= \mathrm{sigm}(x_t + W_{\mathrm{hr}}h_t + b_{\mathrm{r}})\\
h_{t+1} &= \tanh(W_{\mathrm{hh}}(r \odot h_t) + W_{xh}x_t + b_{\mathrm{h}}) \odot z\\
&+ h_t \odot (1 - z)
\end{aligned}$$

MUT3:

$$\begin{aligned}
z &= \mathrm{sigm}(W_{\mathrm{xz}}x_t + W_{\mathrm{hz}}\tanh(h_t) + b_{\mathrm{z}})\\
r &= \mathrm{sigm}(W_{\mathrm{xr}}x_t + W_{\mathrm{hr}}h_t + b_{\mathrm{r}})\\
h_{t+1} &= \tanh(W_{\mathrm{hh}}(r \odot h_t) + W_{xh}x_t + b_{\mathrm{h}}) \odot z\\
&+ h_t \odot (1 - z)
\end{aligned}$$

Summary

- RNNs are a widely used model for sequential data, including text
- RNNs are trainable with backprop when unrolled over time
- RNNs learn complex and varied patterns in sequential data
- Vanilla RNNs are simple but don’t work very well
  - Backward flow of gradients in RNN can explode or vanish
- Common to use LSTM or GRU. Memory path makes them stably learn long-distance interactions.
- Better/simpler architectures are a hot topic of current research