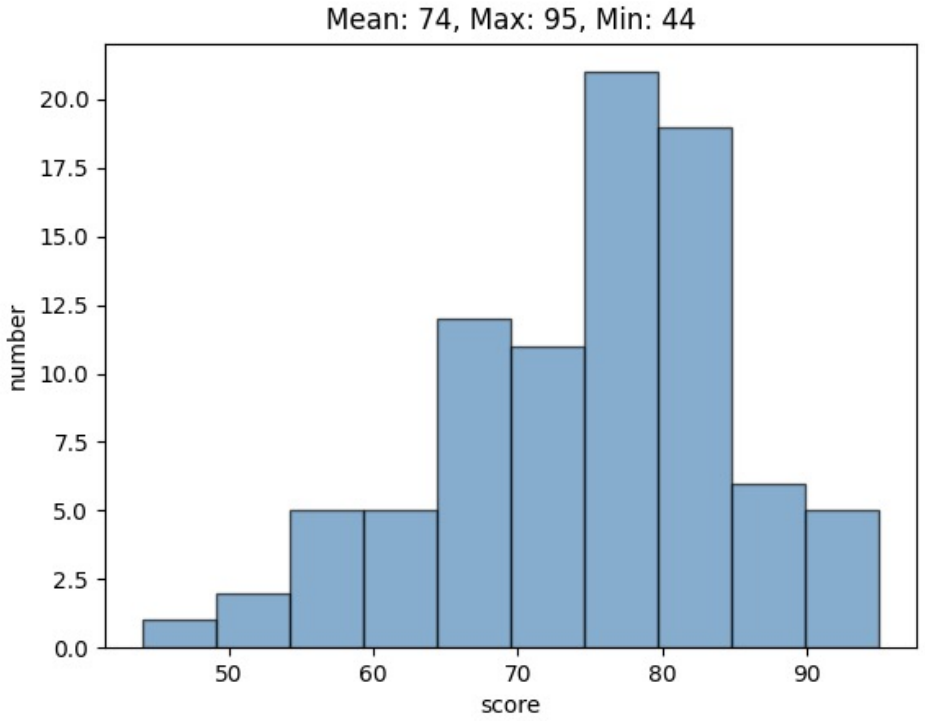
Introduction to Computer Vision

Lecture 10 - Temporal Analysis II

Prof. He Wang

Logistics

- Midterm



- TA session
  - Time: May 12, 4:00PM - 5:30PM
  - Location: Jingyuan Court#5, Room 104

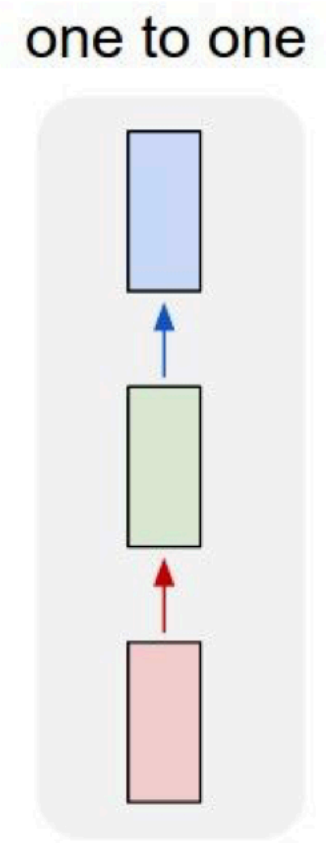
Logistics

- Assignment 3: released on 4/28 11:59PM, due on 5.12 11:59PM

RNN

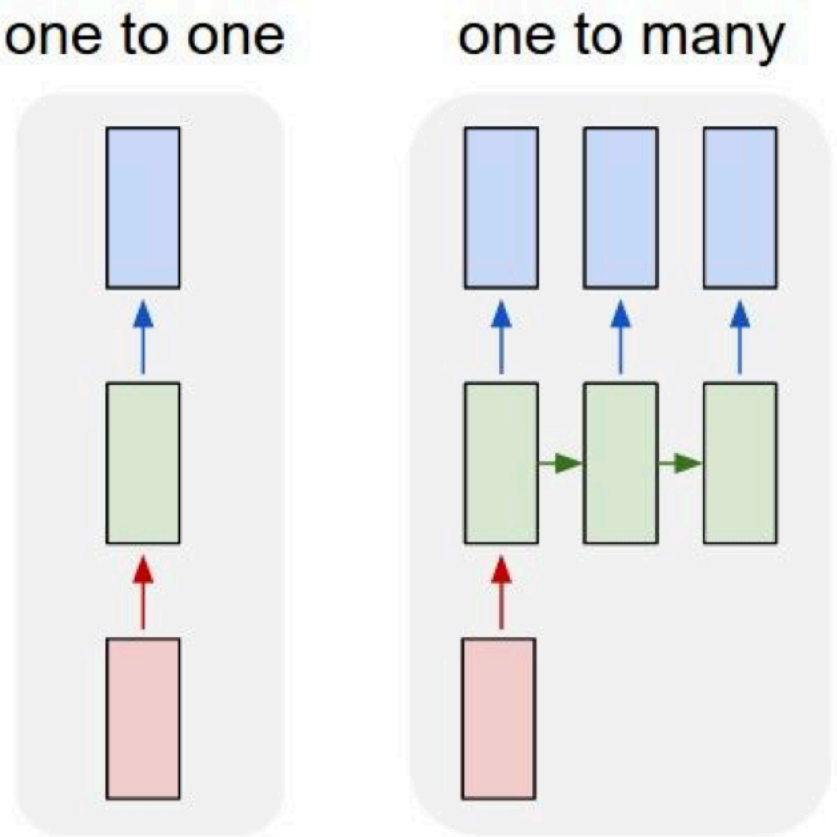
Some slides are borrowed from Stanford CS231N

Single-Frame Neural Network



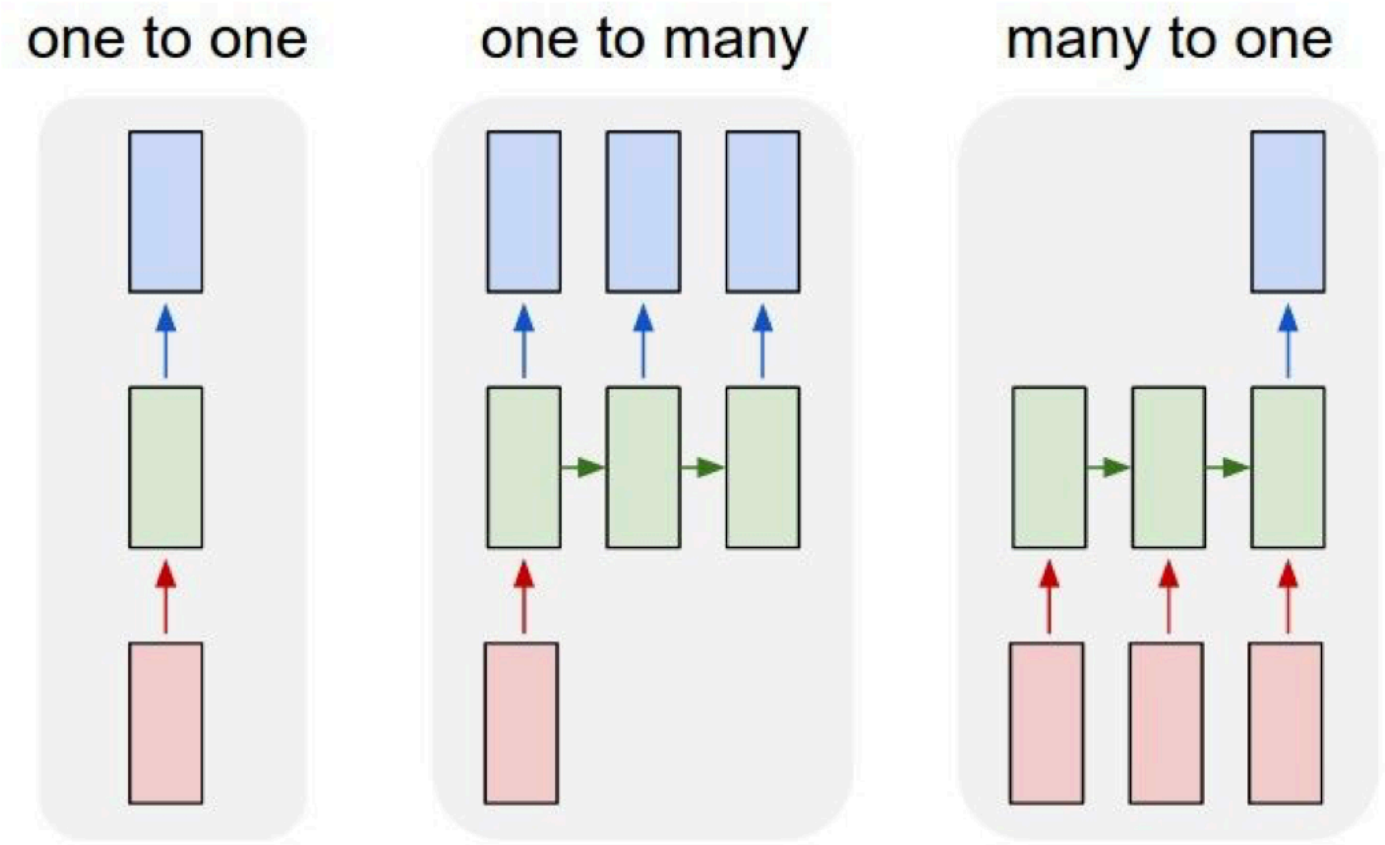
Vanilla Neural Networks

Recurrent Neural Network: Process Sequential Data



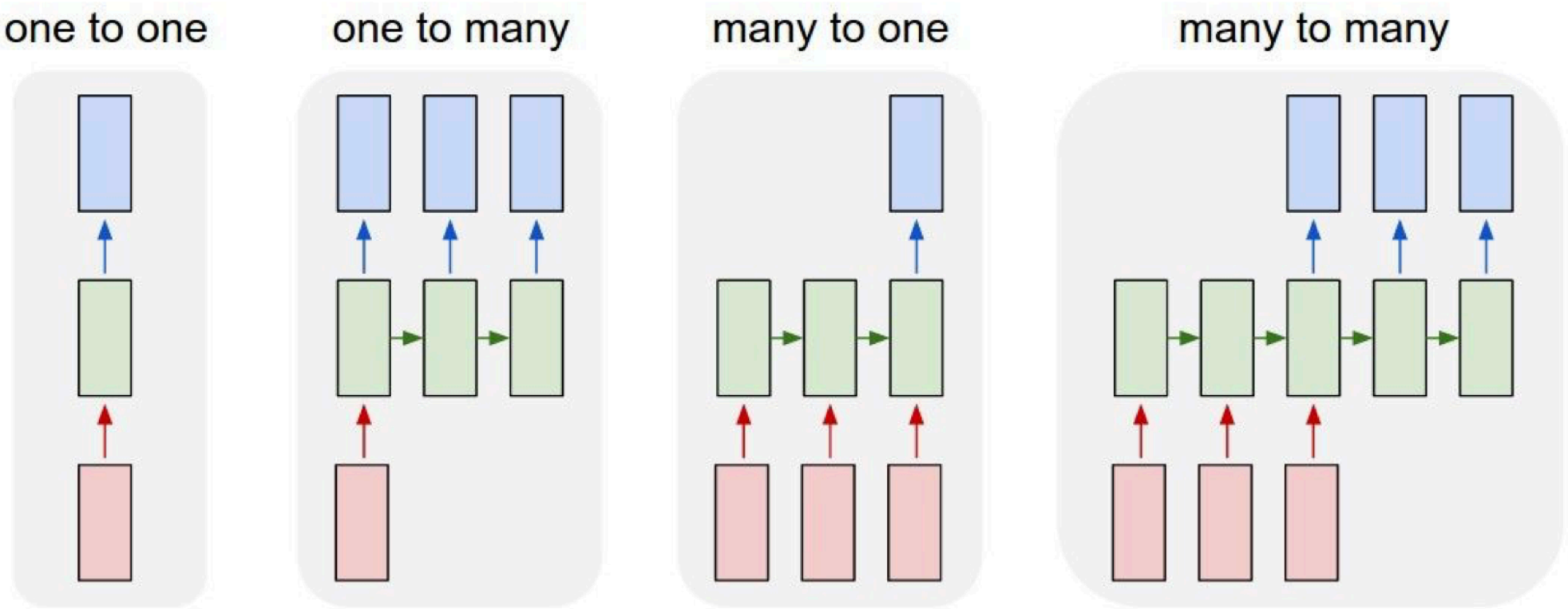
e.g. **Image Captioning**
image -> sequence of words

Recurrent Neural Network: Process Sequential Data



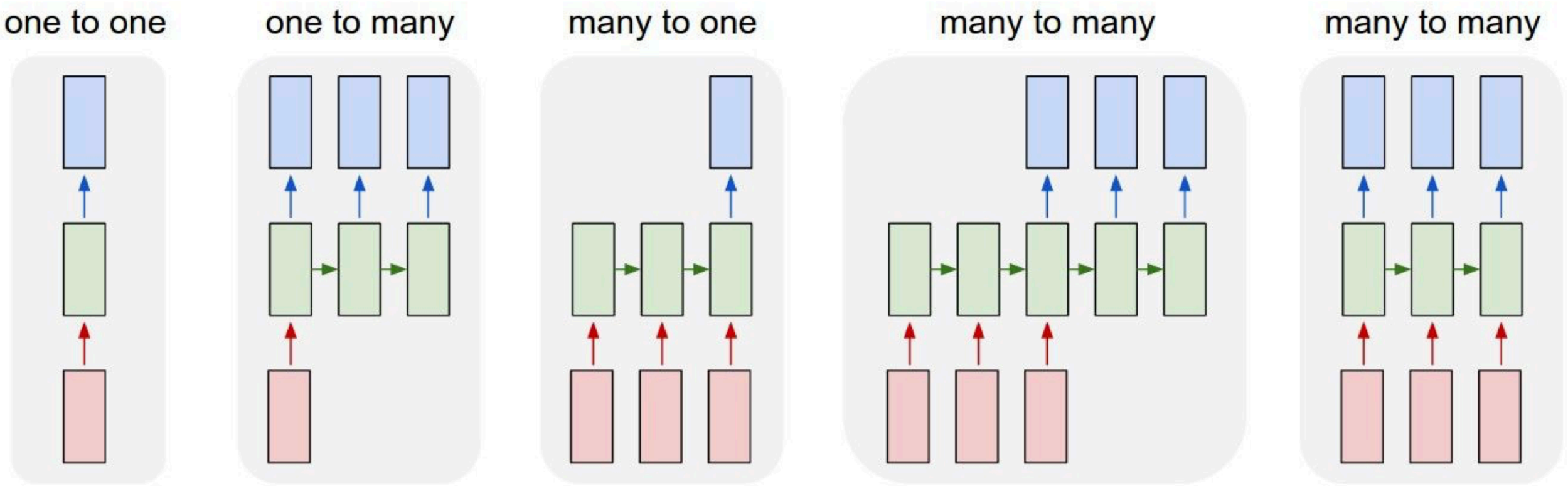
e.g. **action prediction**
sequence of video frames -> action class

Recurrent Neural Network: Process Sequential Data



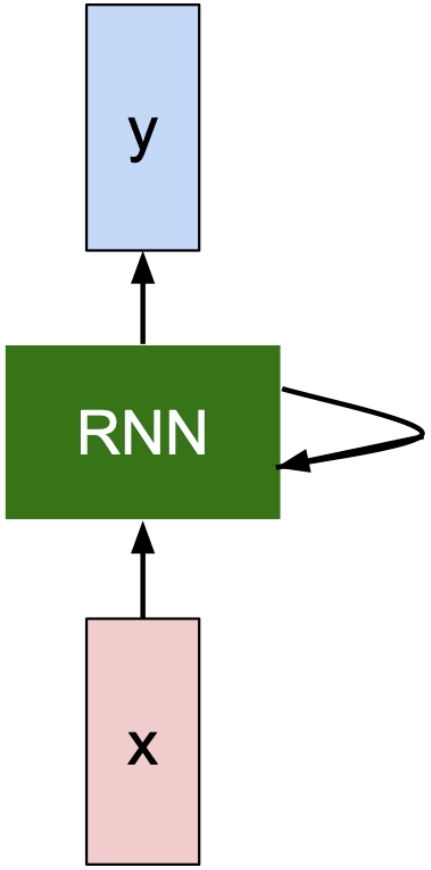
E.g. **Video Captioning**
Sequence of video frames -> caption

Recurrent Neural Network: Process Sequential Data

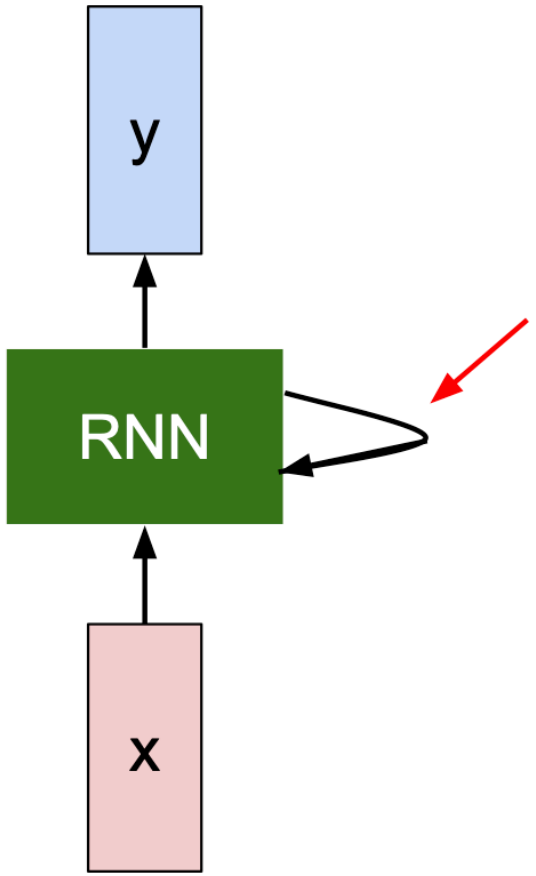


e.g. **Video classification on frame level**

Recurrent Neural Network

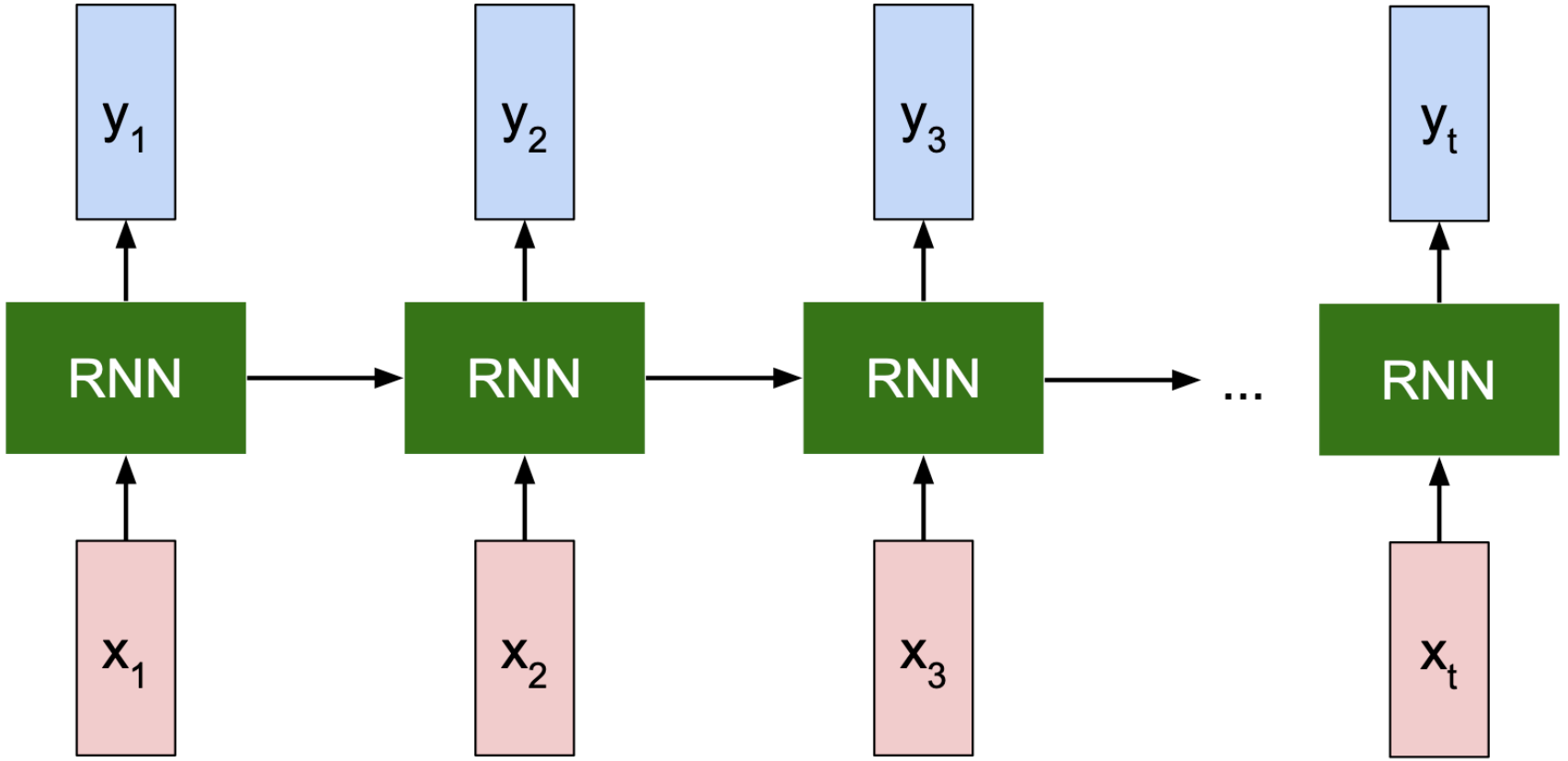


Recurrent Neural Network



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

Unrolled RNN



RNN Hidden Sate Update

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

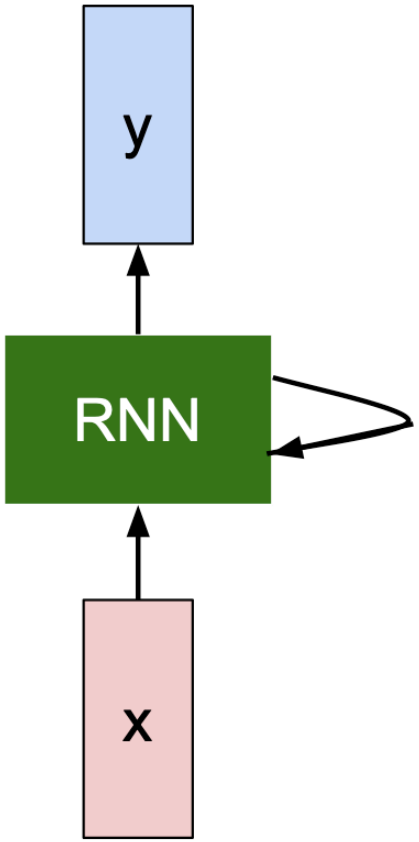
$$h_t = f_W(h_{t-1}, x_t)$$

new state

some function with parameters W

old state

input vector at some time step



RNN Output

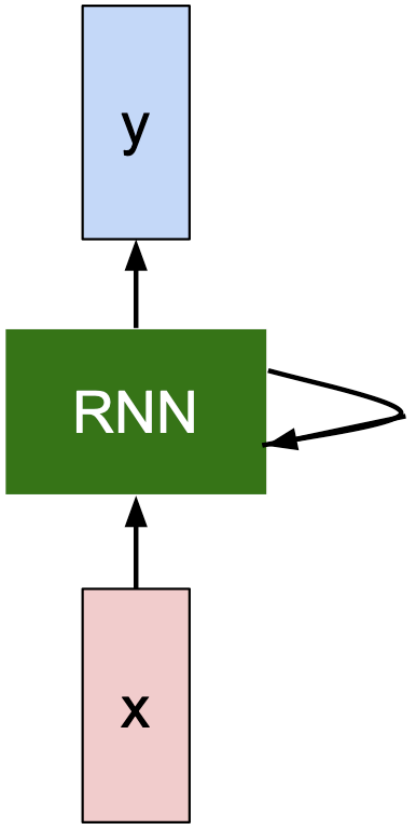
We can process a sequence of vectors **x** by applying a **recurrence formula** at every time step:

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

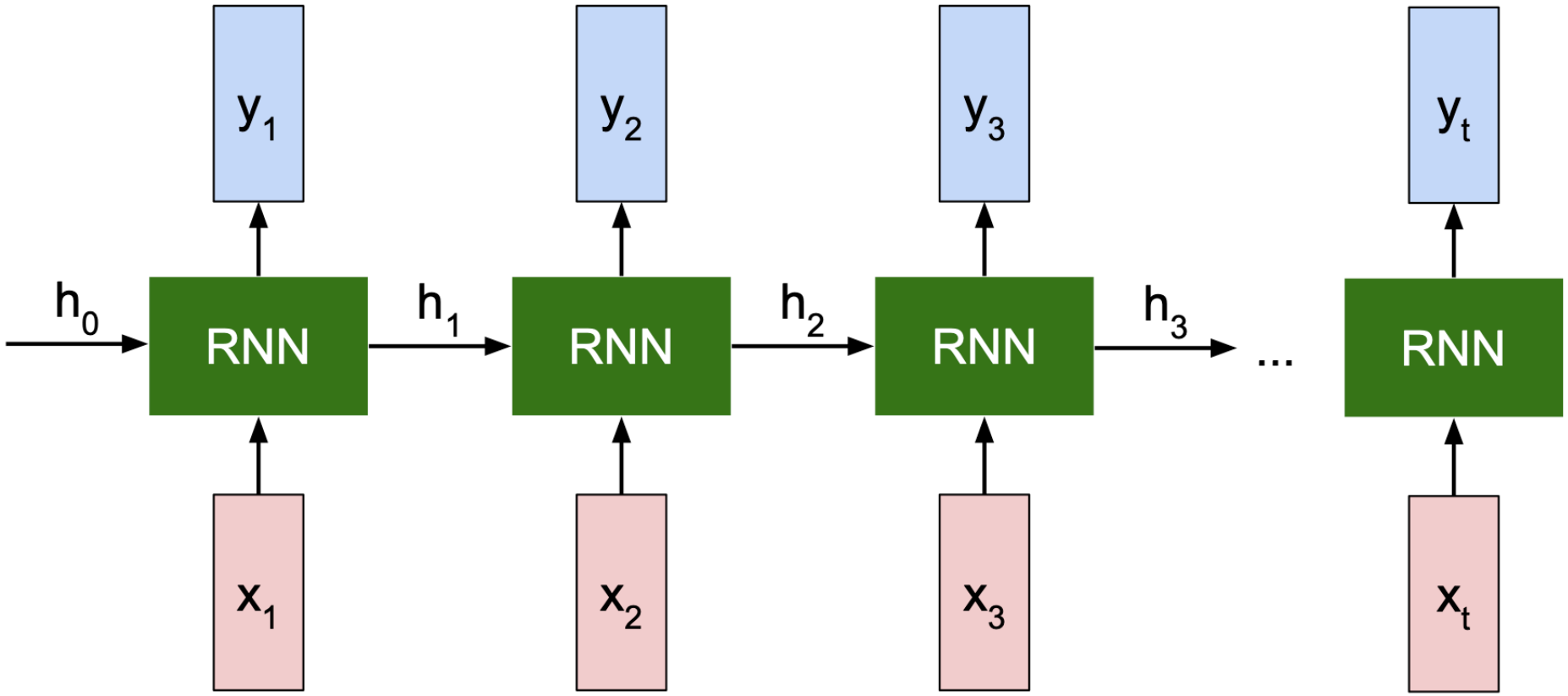
output

new state

another function with parameters W_o



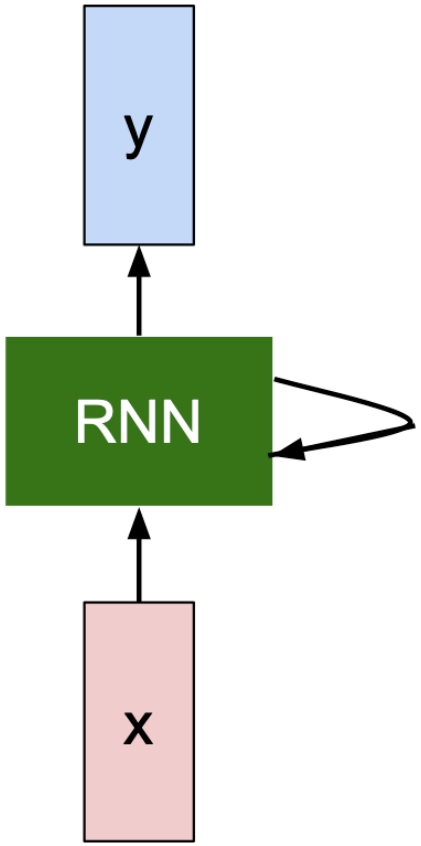
Recurrent Neural Network



Recurrent Neural Network

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

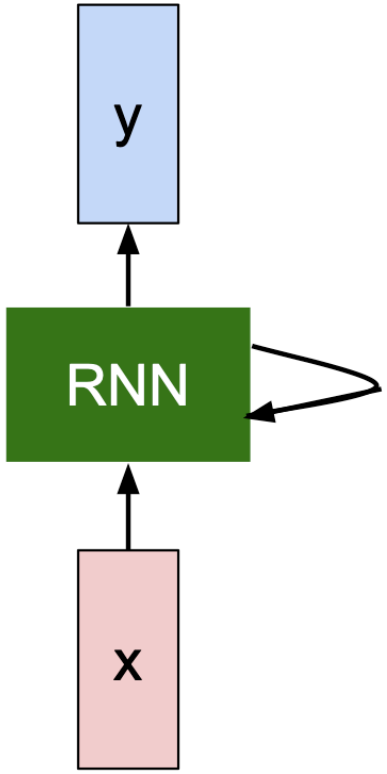
$$h_t = f_W(h_{t-1}, x_t)$$



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

Vanilla RNN

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



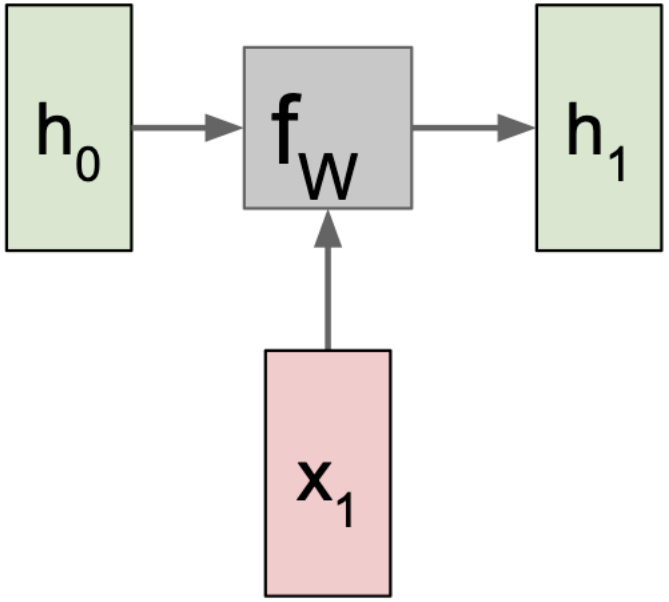
$$h_t = f_W(h_{t-1}, x_t)$$

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

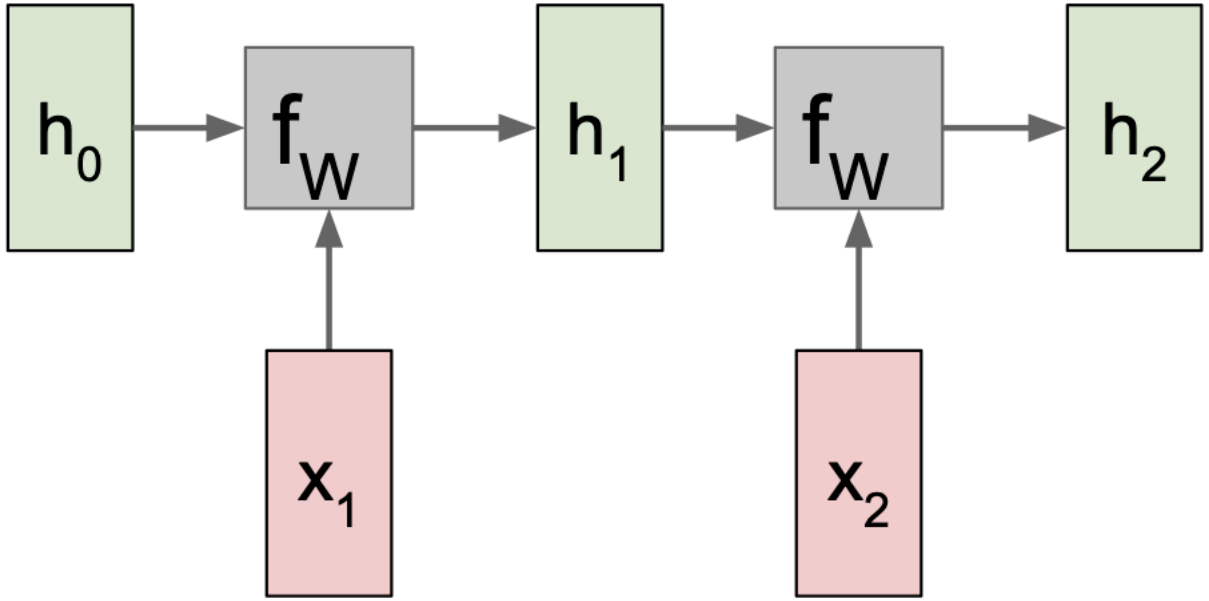
$$y_t = W_{hy}h_t$$

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

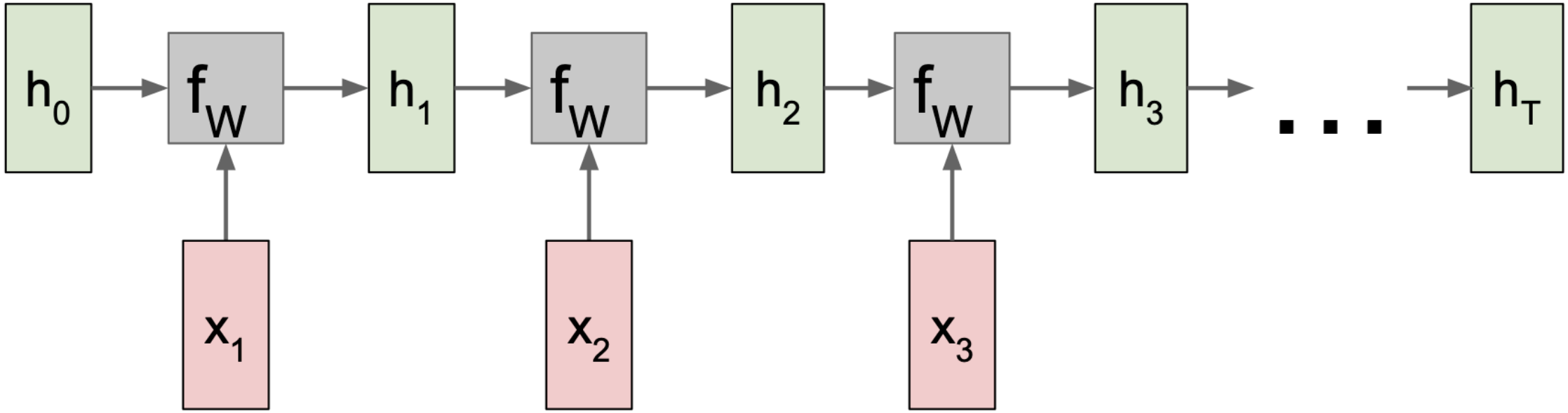
RNN: Computational Graph



RNN: Computational Graph

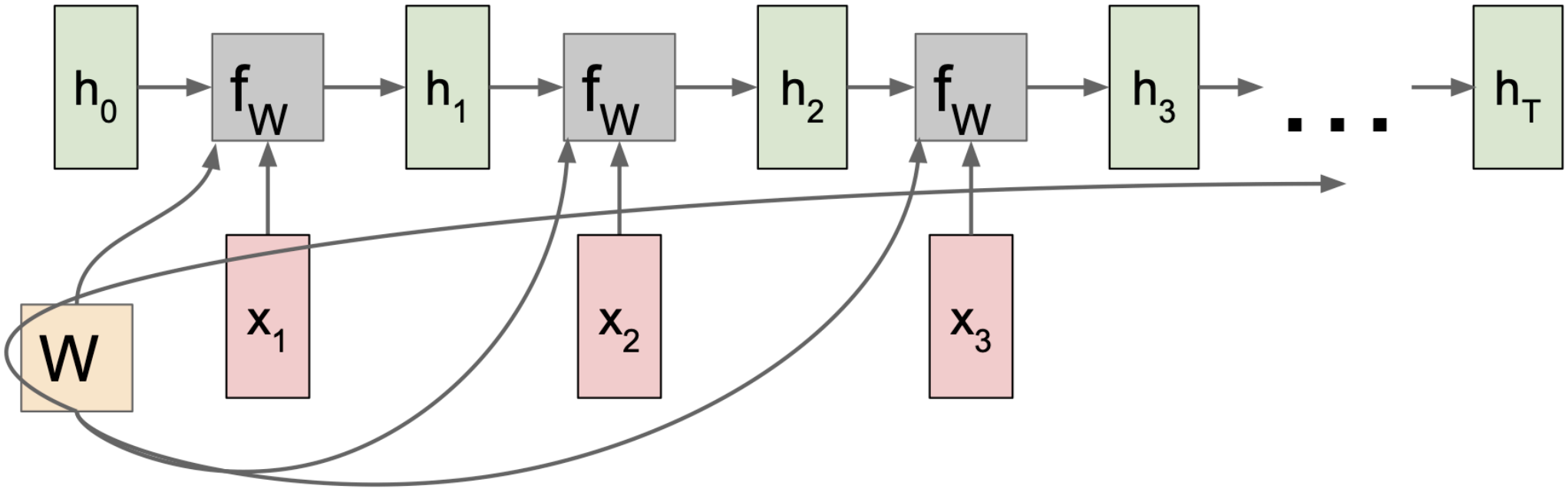


RNN: Computational Graph



RNN: Computational Graph

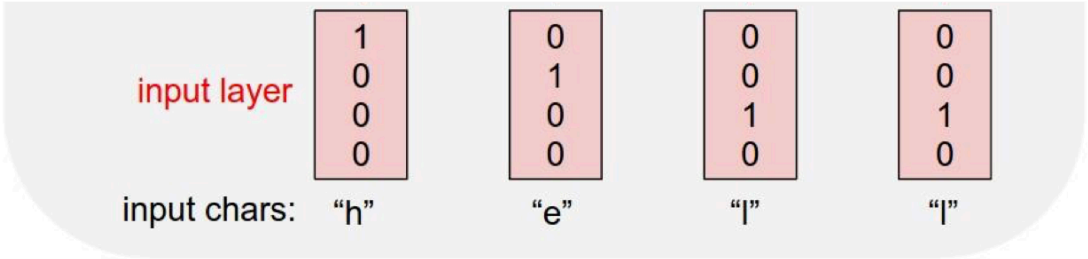
Re-use the same weight matrix at every time-step



Character-Level Language Model

Vocabulary:
[h,e,l,o]

Example training sequence:
“hello”

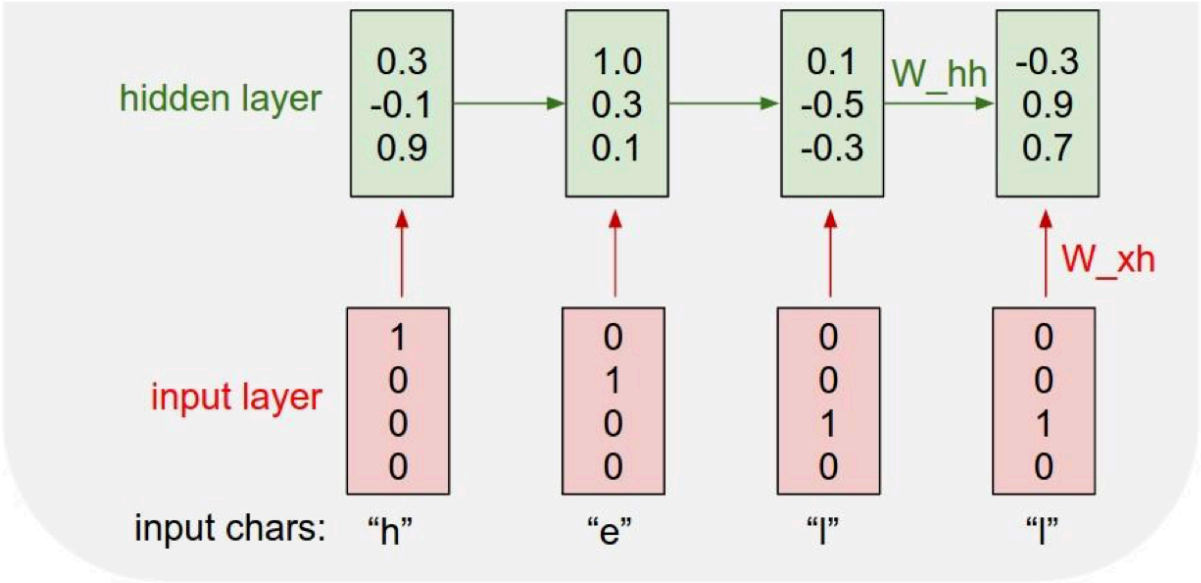


Character-Level Language Model

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

Vocabulary:
[h,e,l,o]

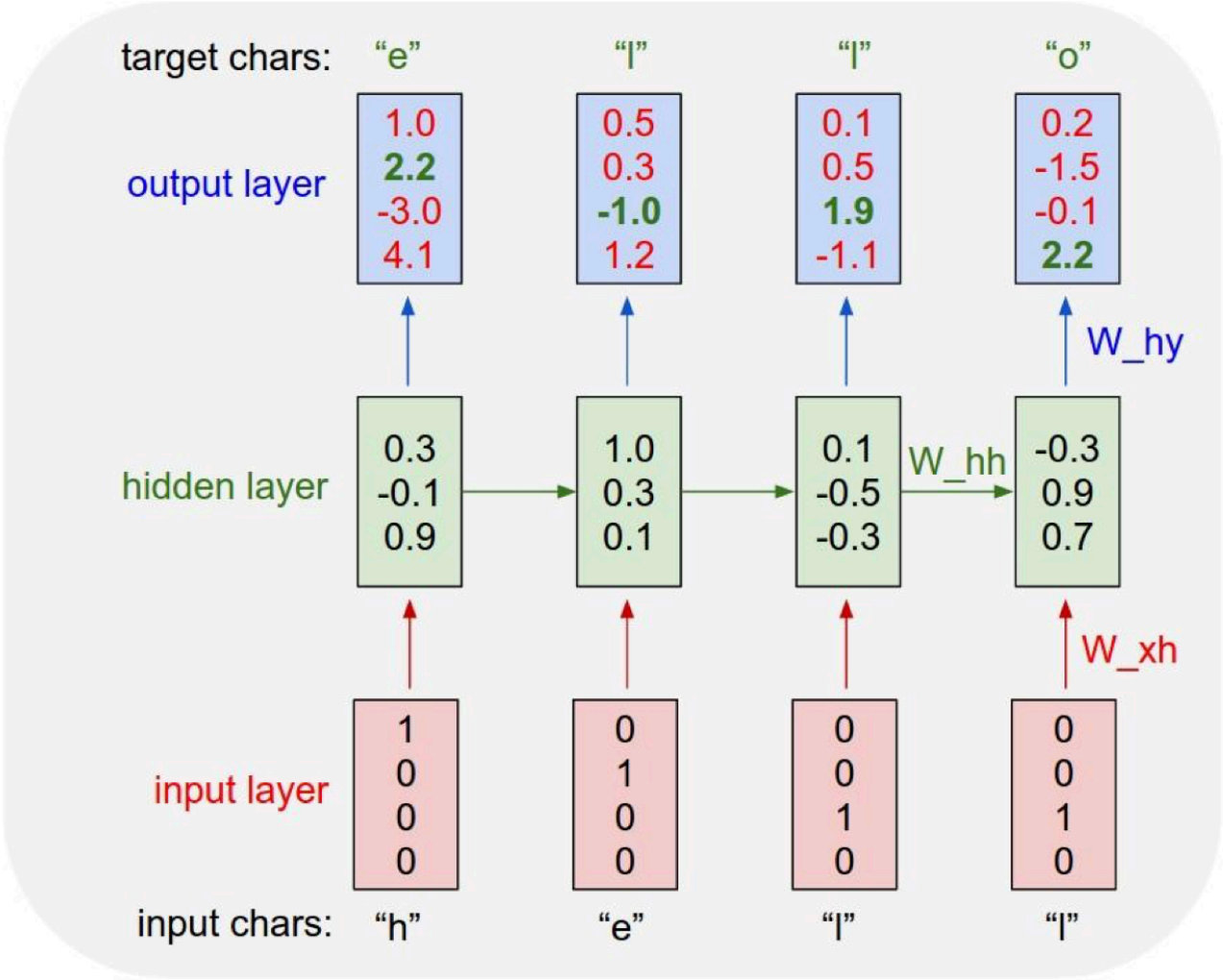
Example training sequence:
“hello”



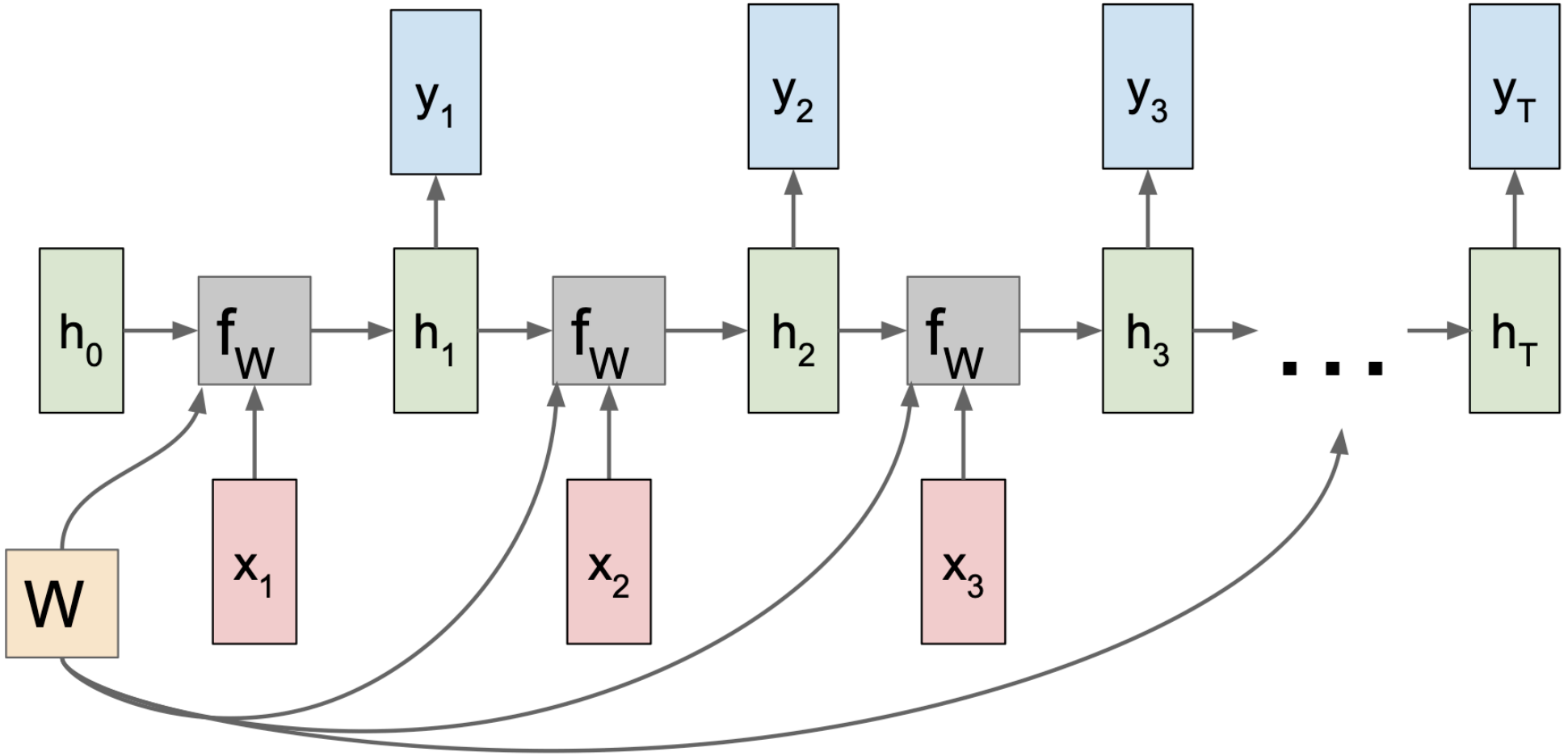
Character-Level Language Model

Vocabulary:
[h,e,l,o]

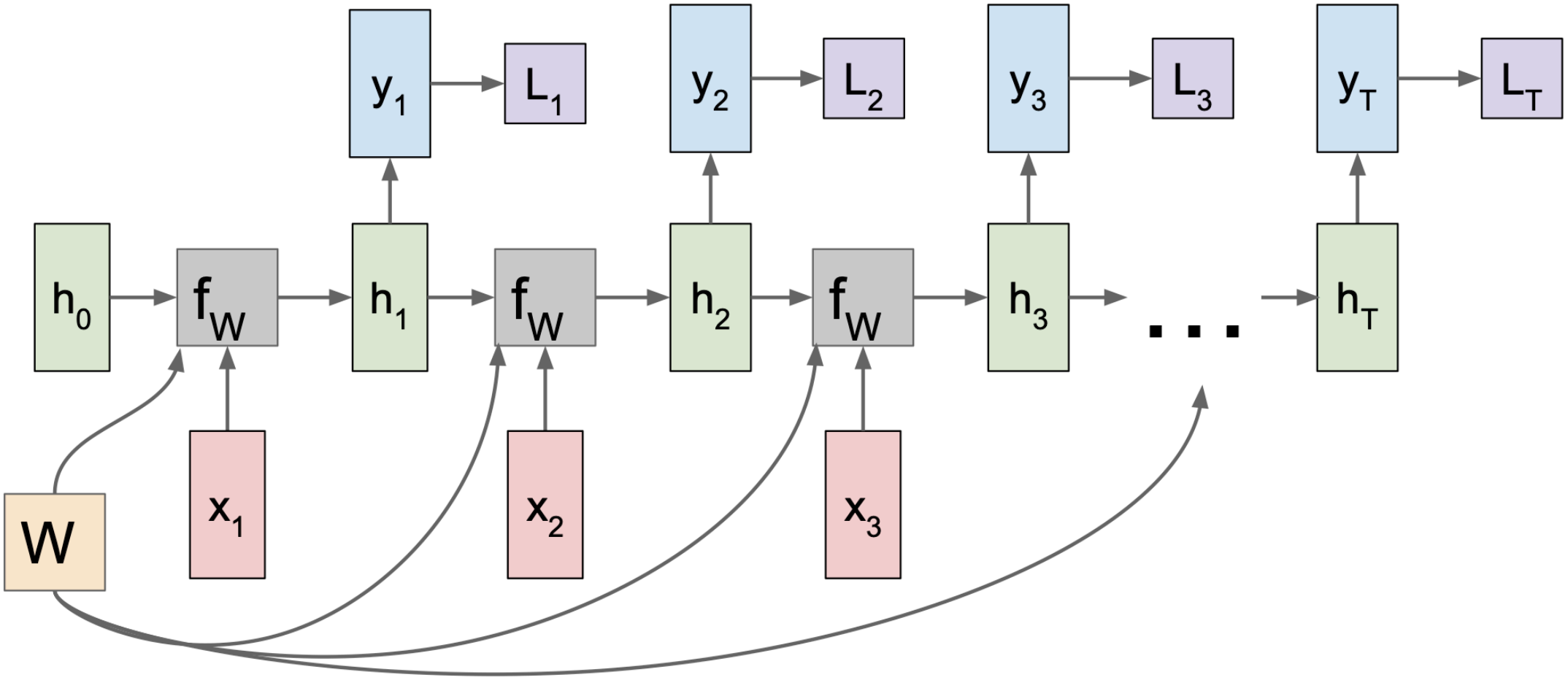
Example training sequence:
“hello”



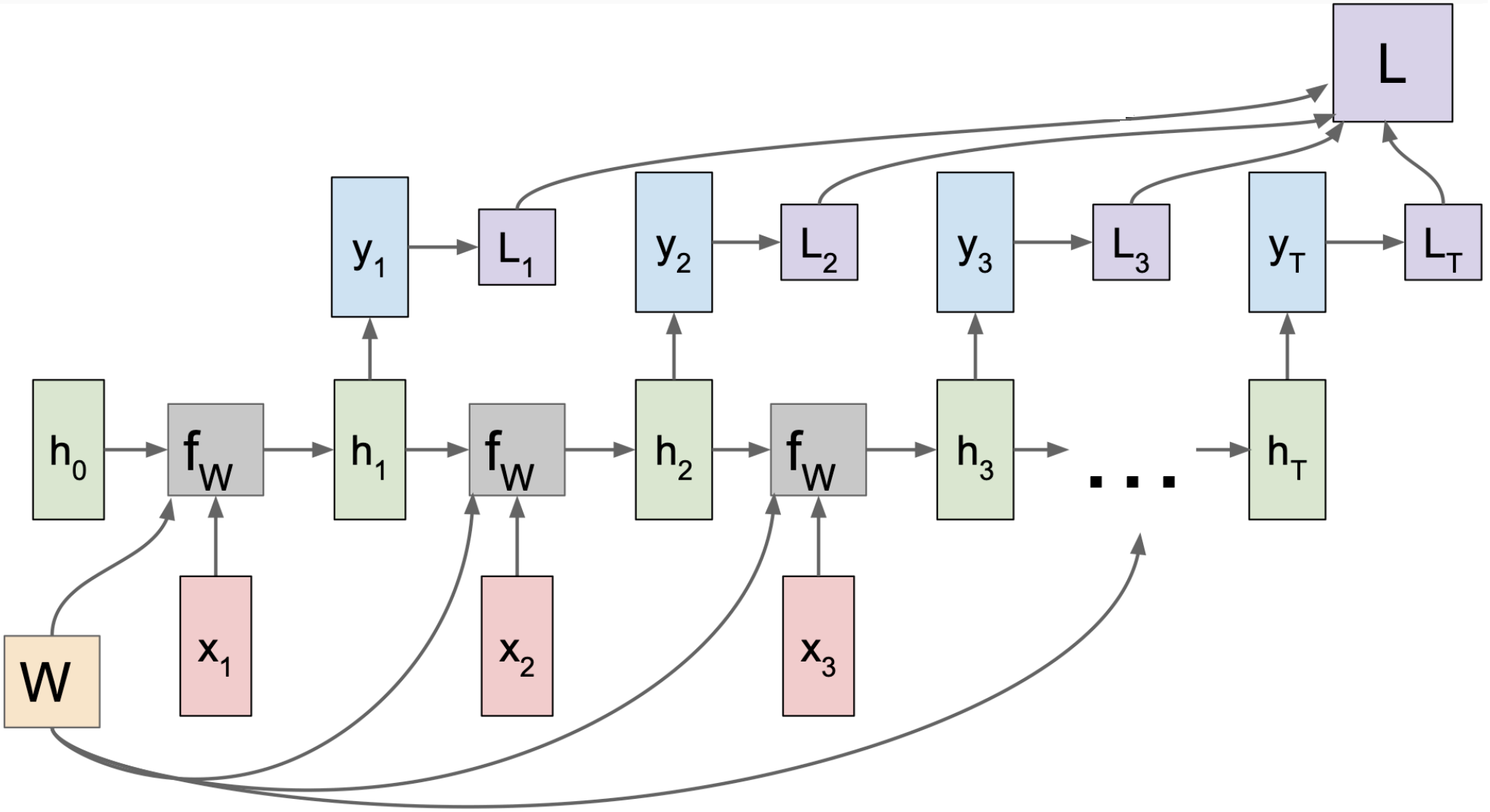
Many-to-Many RNN: Computational Graph



Many-to-Many RNN: Computational Graph

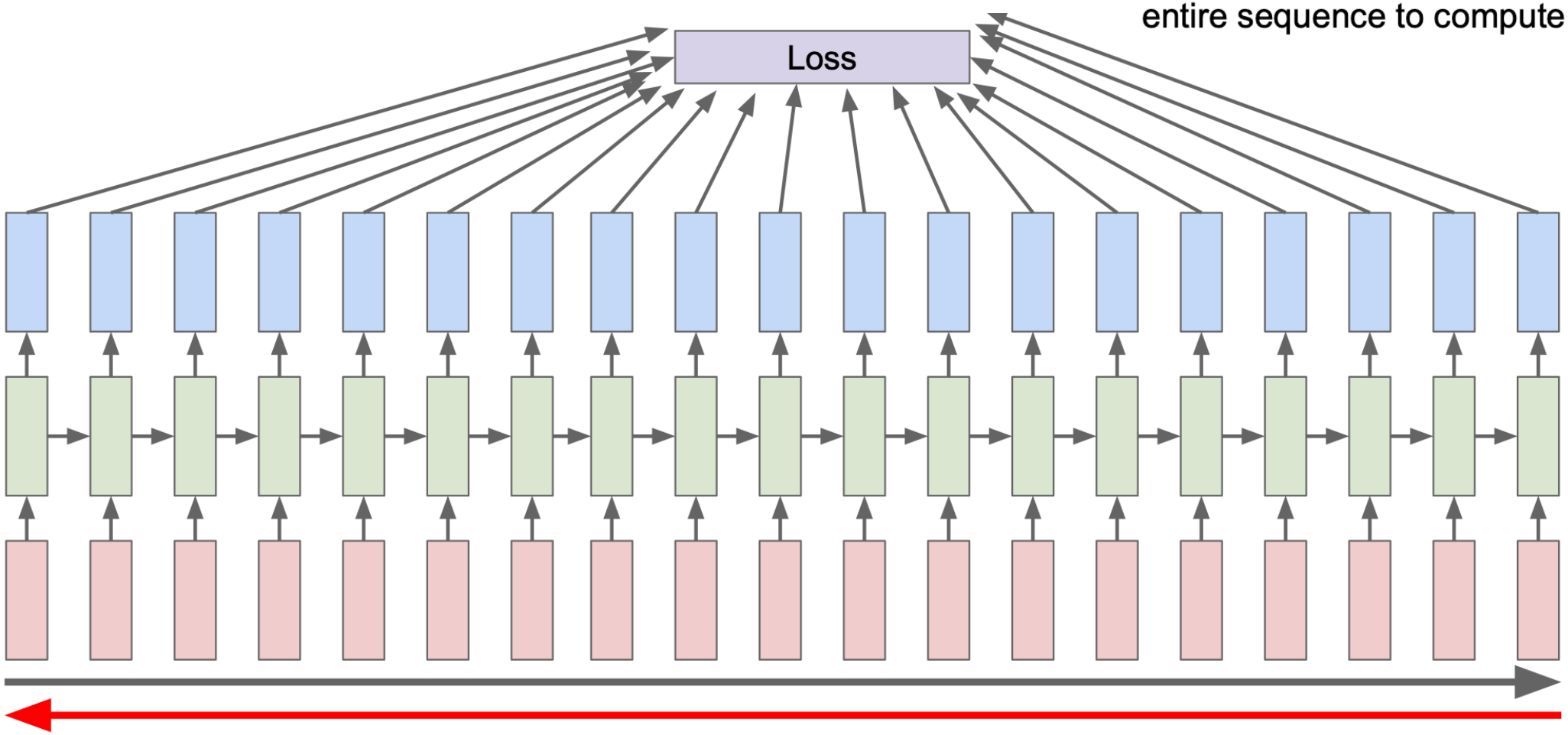


Many-to-Many RNN: Computational Graph

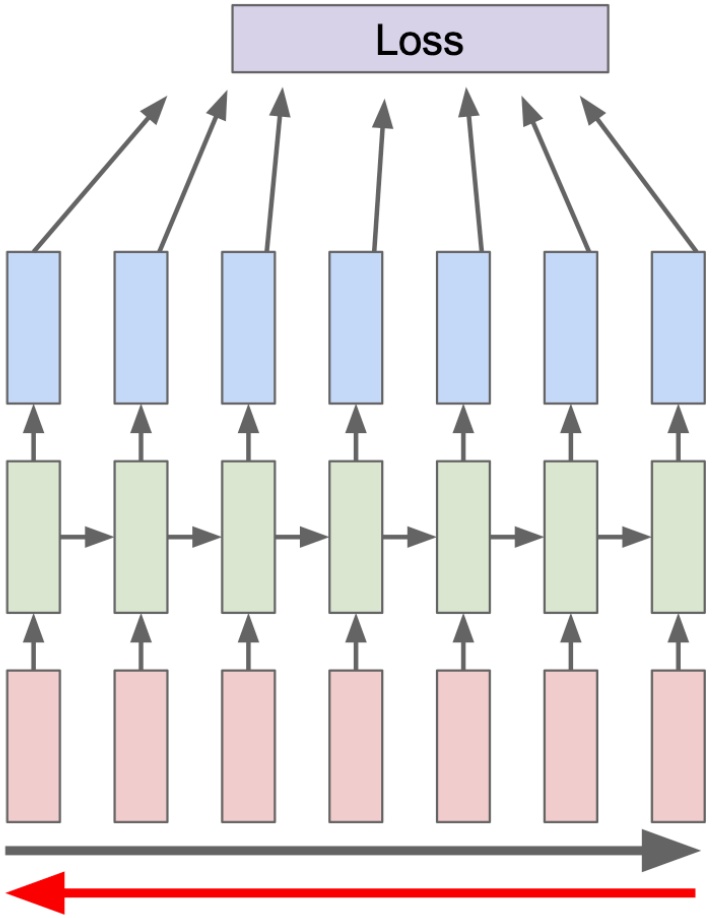


Backpropagation through Time

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

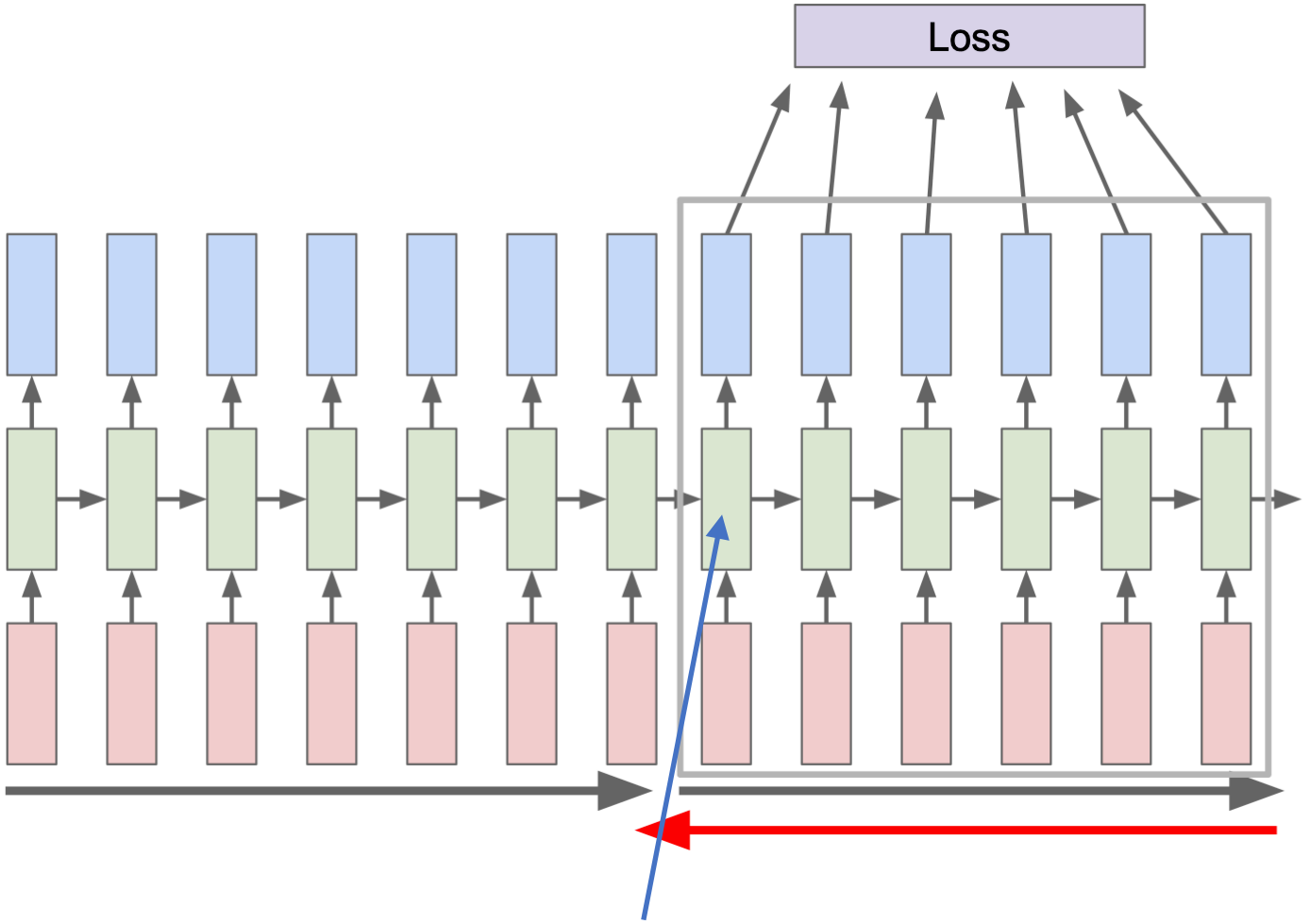


Truncated Backpropagation through Time



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

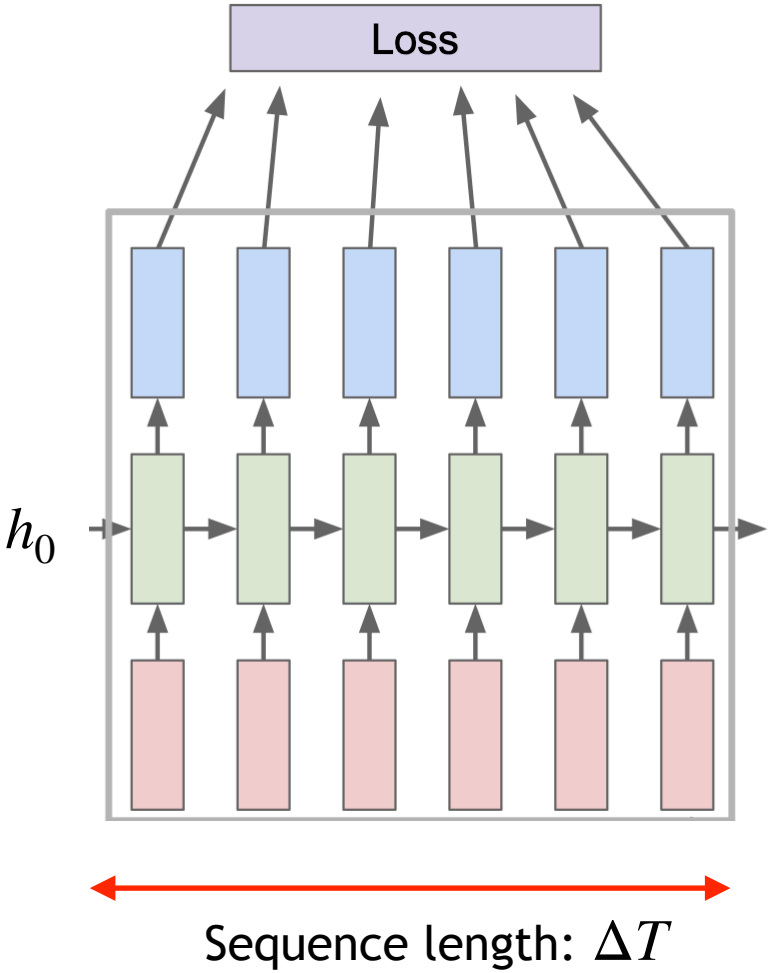
Truncated Backpropagation through Time



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

h_t or in practice can use h_0 if t is so very large such that we even don't want to compute h_t during training

Truncated Backpropagation through Time



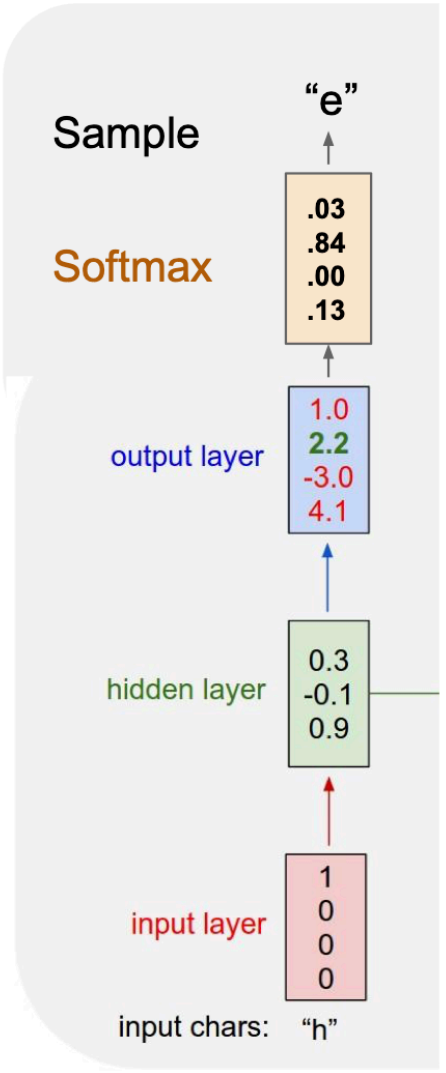
If use h_0, then training only enforces the temporal relationship inside the window, whose length is sequence length.

Long-term relationship longer than ΔT will probably not be learned.

Character-Level Language Model Sampling

Vocabulary:
[h,e,l,o]

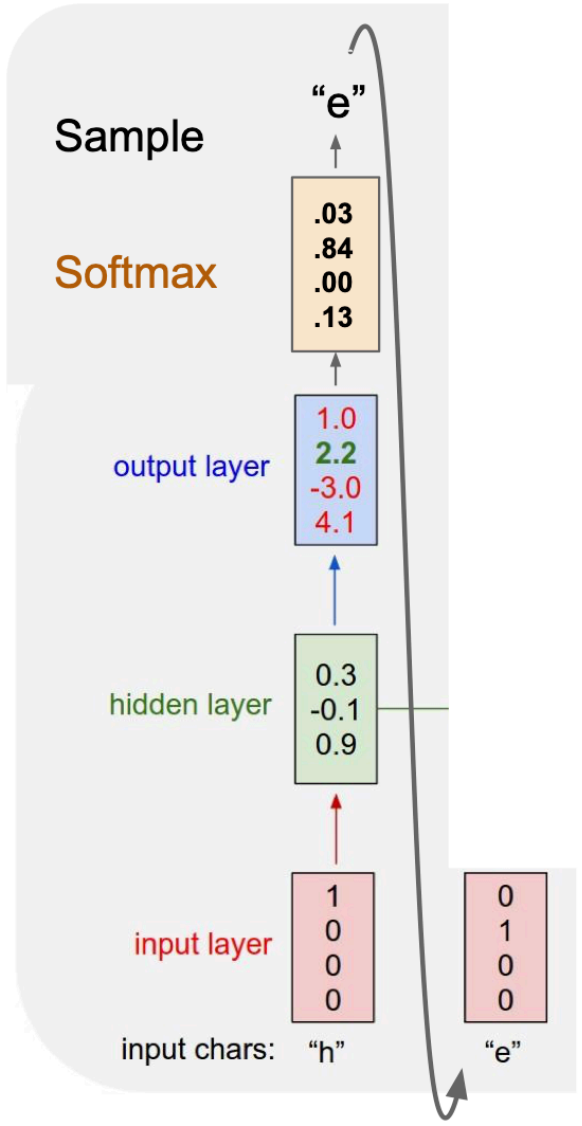
At test-time sample characters one at a time, feed back to model



Character-Level Language Model Sampling

Vocabulary:
[h,e,l,o]

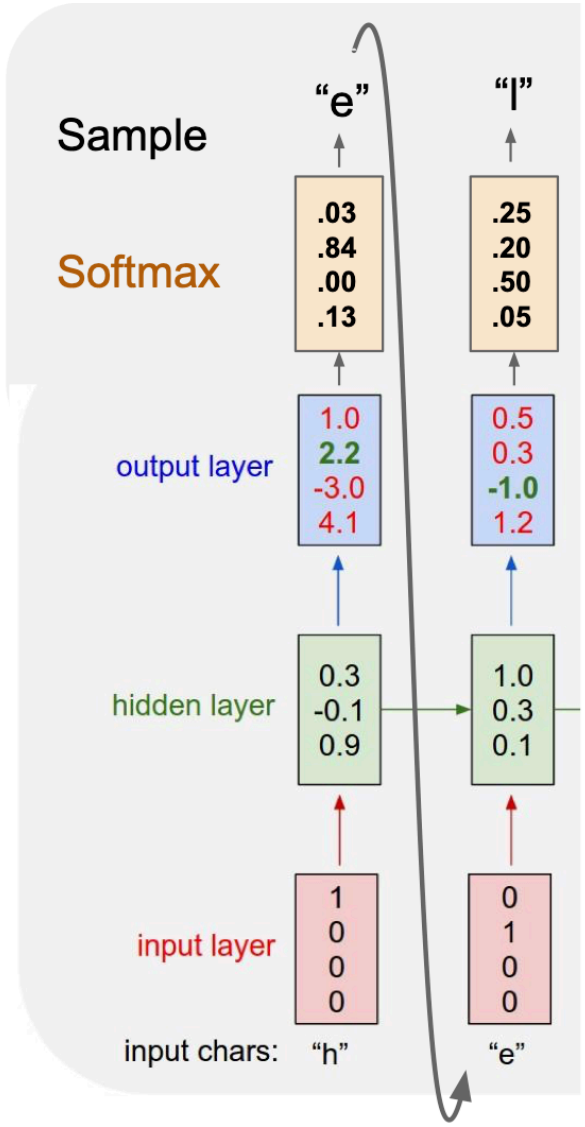
At test-time sample characters one at a time, feed back to model



Character-Level Language Model Sampling

Vocabulary:
[h,e,l,o]

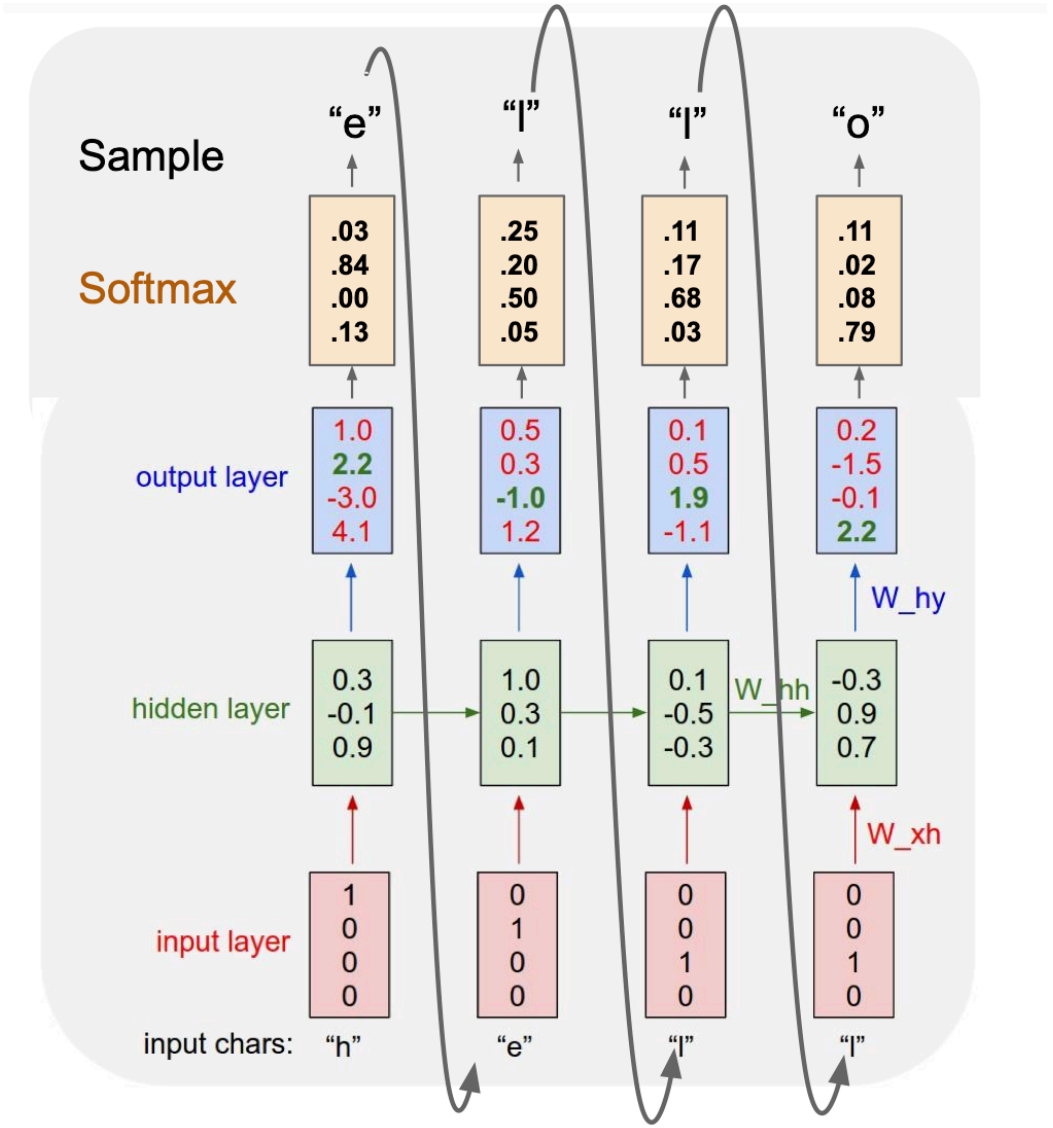
At test-time sample characters one at a time, feed back to model



Character-Level Language Model Sampling

Vocabulary:
[h,e,l,o]

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

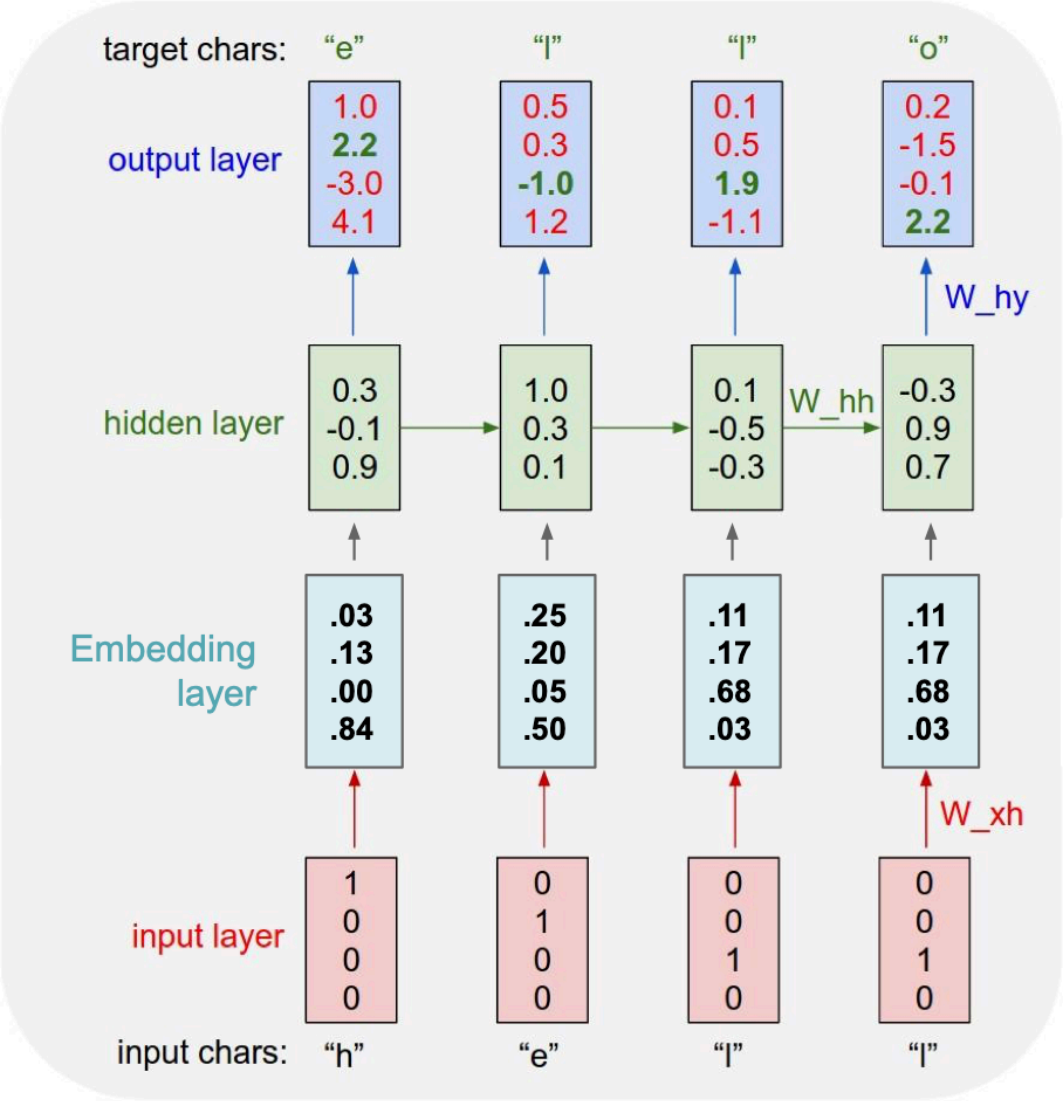


Character-Level Language Model Sampling

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

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

Sometimes embedding layer is fixed, e.g. use a pretrained word embedding.



Different Sampling Strategies

- Greedy sampling: always takes the highest prob.
  - Issue: deterministic, can only generate one sequence given the initial token and hidden states.

- Weighted sampling: sample the next token according to the predicted probability distribution
  - Advantage: can generate diverse sequences.
  - Issue: can accidentally obtain wrong token and screw up the generation.

Exhaustive Search

- Ideally, we may want to find a (length T) sequence that maximizes the probability

$$P(y|x) = P(y_1|x)\, P(y_2|y_1, x)\, P(y_3|y_1, y_2, x) \ldots, P(y_T|y_1, \ldots, y_{T-1}, x)$$
$$= \prod_{t=1}^{T} P(y_t|y_1, \ldots, y_{t-1}, x)$$

- We could try computing all possible sequences and find the one with the maximum probability
  - However, too expensive, $O(V^T)$ complexity where V is the vocabulary size.

Beam Search

- On each timestep, keep track of the k most probable partially generated sequences
  - k is the beam size.

- Beam search is not guaranteed to find optimal solution. However, more efficient than exhaustive search.

The Unreasonable Effectiveness of Recurrent Neural Networks

Andrej Karpathy blog

The Unreasonable Effectiveness of Recurrent Neural Networks

May 21, 2015

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

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

```
"""
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
```

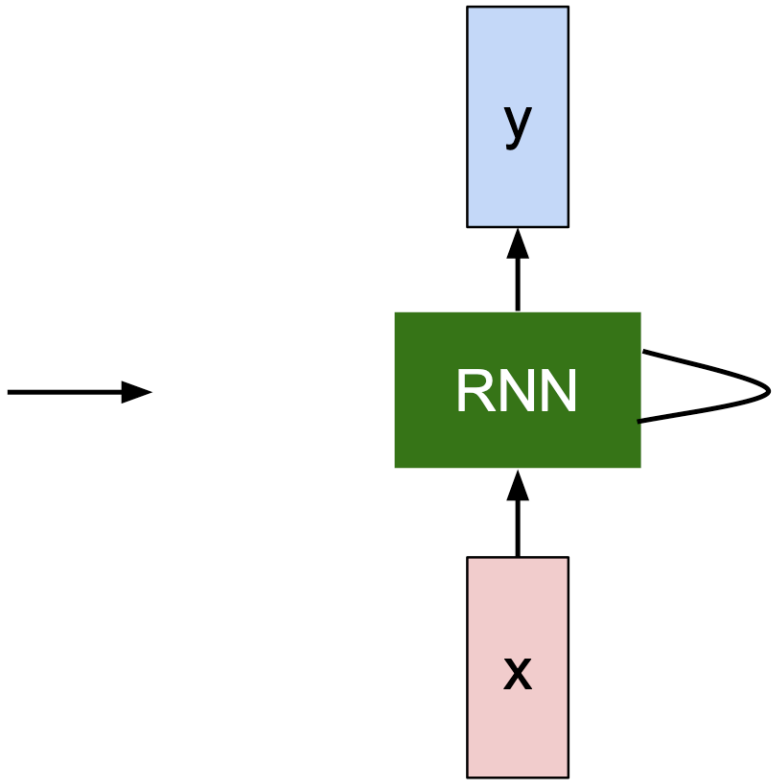
Character-Level Language Model: Shakespeare

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.



Character-Level Language Model: Shakespeare

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.
```

Word-Level Language Model: Shakespeare

```
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.
```

Due to limited sequence length, the text generated during test time can't capture long-term relationship longer than sequence length and there becomes locally plausible but globally meaningless.

Word-Level Language Model: Math Book (Latex)

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 compariccoly 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. □

Word-Level Language Model: Math Book (Latex)

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

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

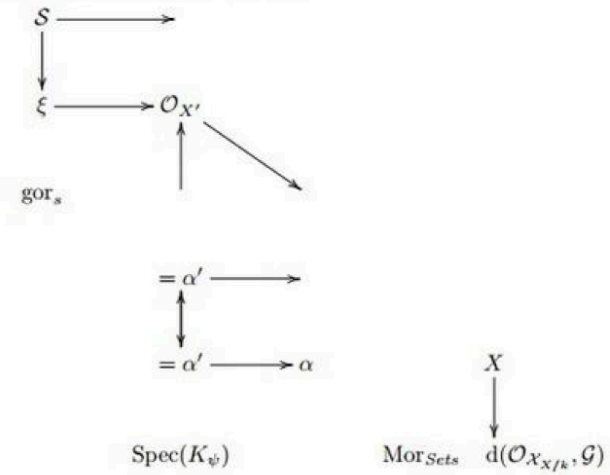
be a morphism of algebraic spaces over S and Y.

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

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

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.

Character-Level Language Model: 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 & 0x00000000ffffffff8) & 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

RNN Tradeoffs

RNN Advantages:

- Can process any length input
- Computation for step *t* can (in theory) use information from many steps back
- Model size doesn't increase for longer input
- Same weights applied on every timestep, so there is symmetry in how inputs are processed.

RNN Disadvantages:

- Recurrent computation is slow
- In practice, difficult to access information from many steps back

Image Captioning

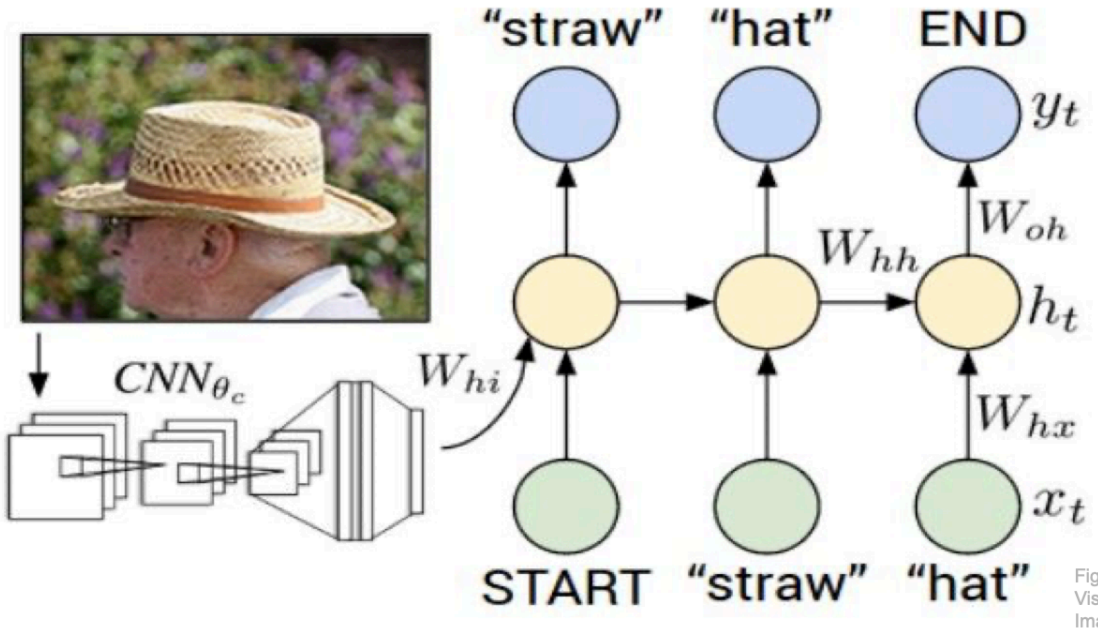


Figure from Karpathy et a, "Deep Visual-Semantic Alignments for Generating Image Descriptions", CVPR 2015; figure copyright IEEE, 2015.
Reproduced for educational purposes.

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

Image Captioning

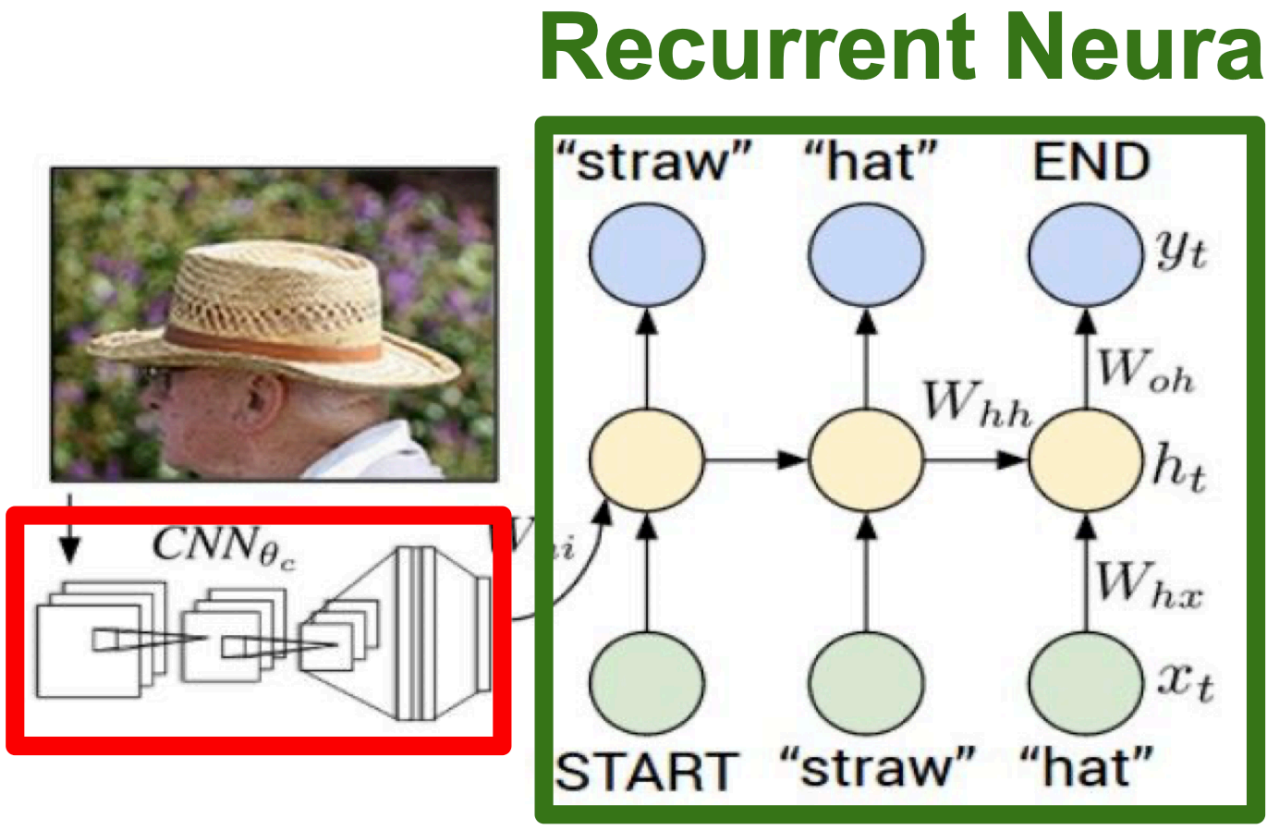


Image Captioning: Example Results

A cat sitting on a suitcase on the floor

A cat is sitting on a tree branch

A dog is running in the grass with a frisbee

A white teddy bear sitting in the grass

Two people walking on the beach with surfboards

A tennis player in action on the court

Two giraffes standing in a grassy field

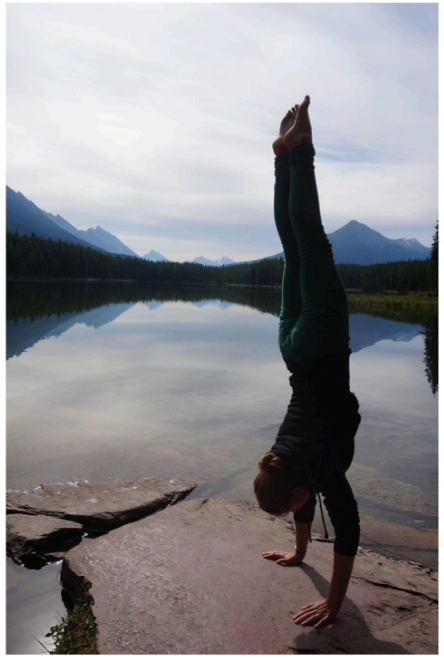
A man riding a dirt bike on a dirt track

Captions generated using neuraltalk2.

Image Captioning: Failure Cases

A woman is holding a cat in her hand

A person holding a computer mouse on a desk



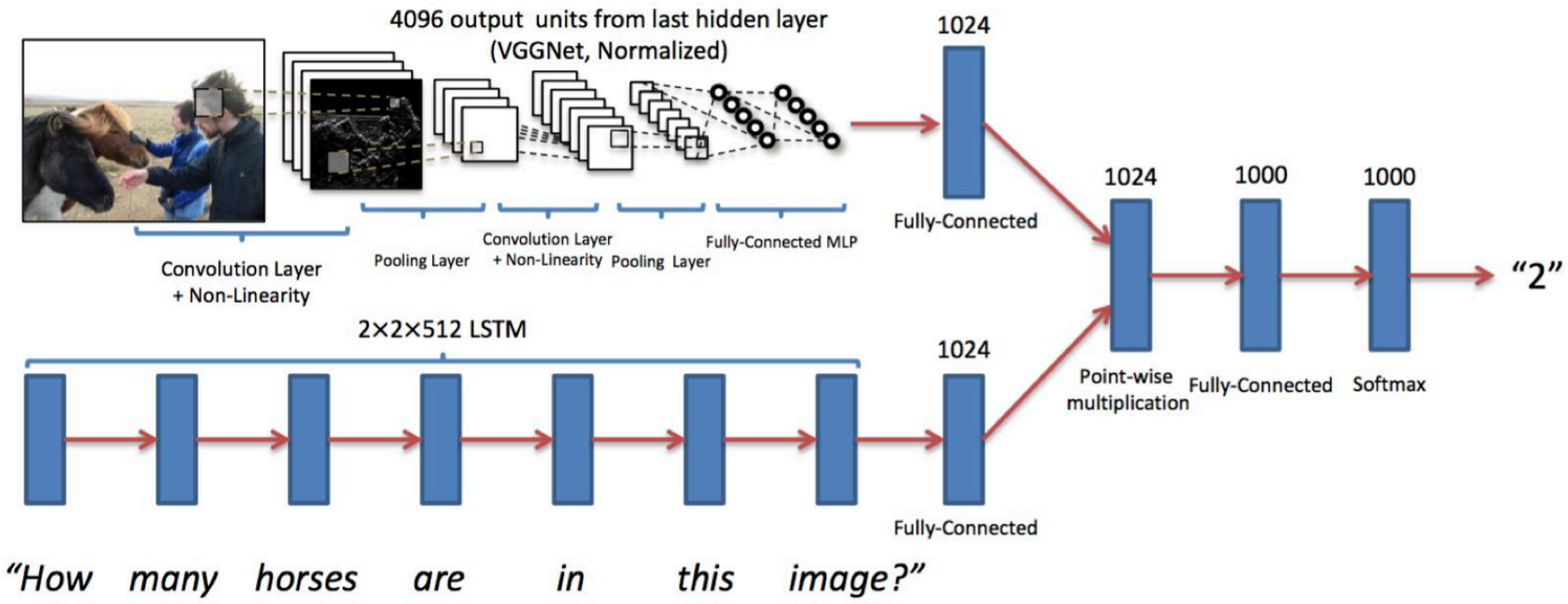
A woman standing on a beach holding a surfboard

A bird is perched on a tree branch

A man in a baseball uniform throwing a ball

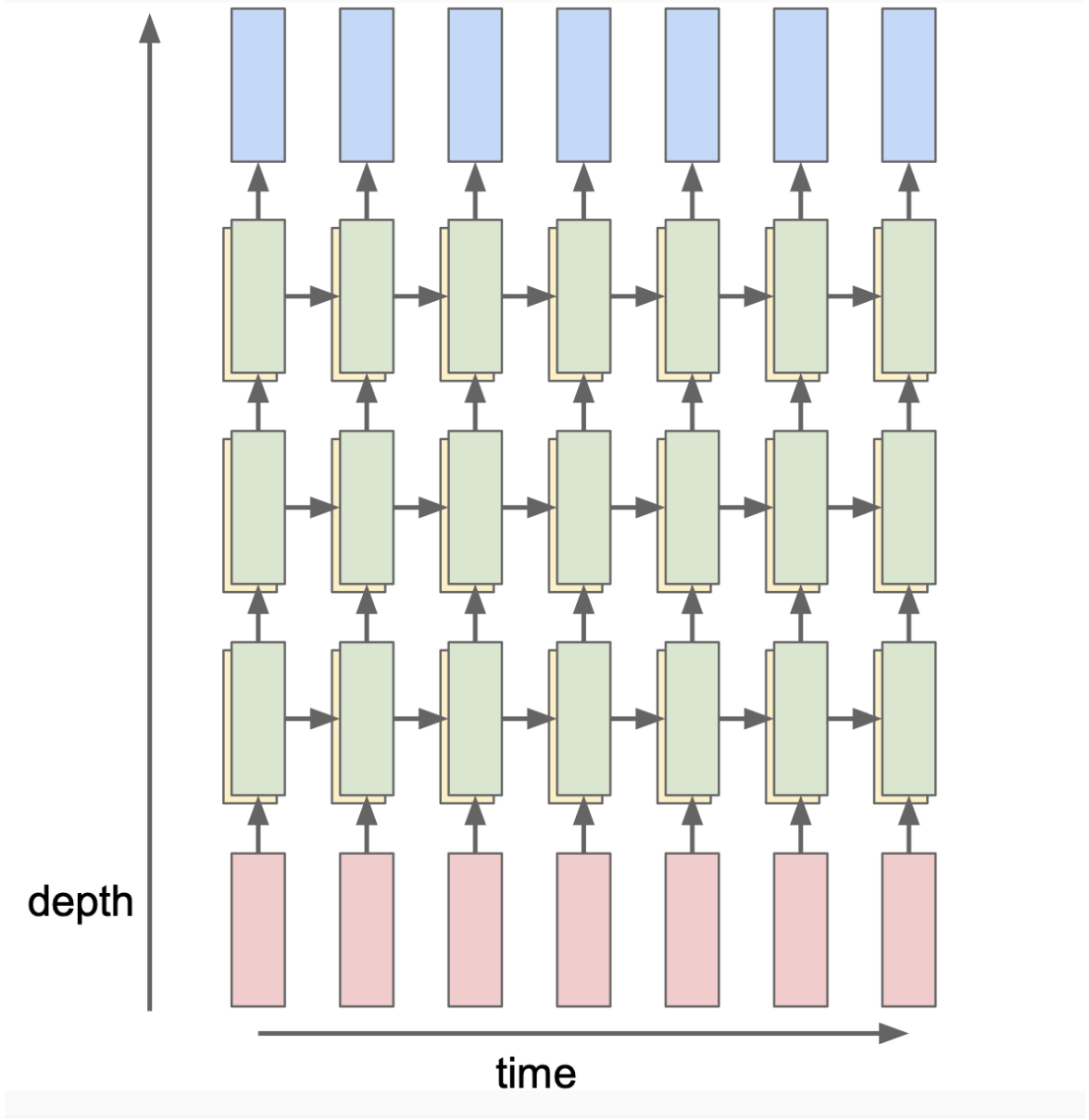
Captions generated using neuraltalk2.

Visual Question Answering (VQA)



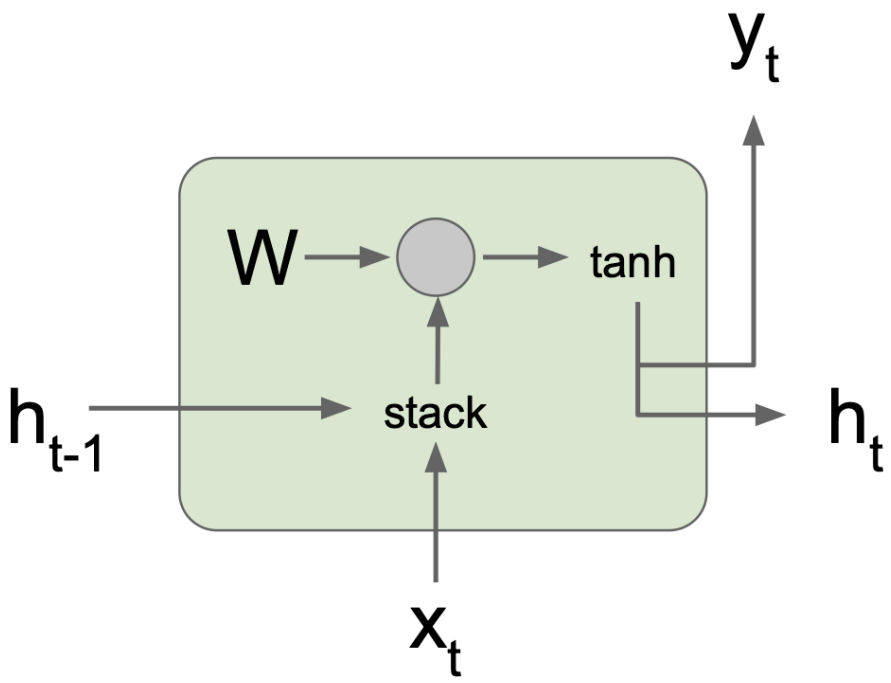
Agrawal et al, "Visual 7W: Grounded Question Answering in Images", CVPR 2015

Multilayer RNNs



Vanilla RNN Gradient Flow

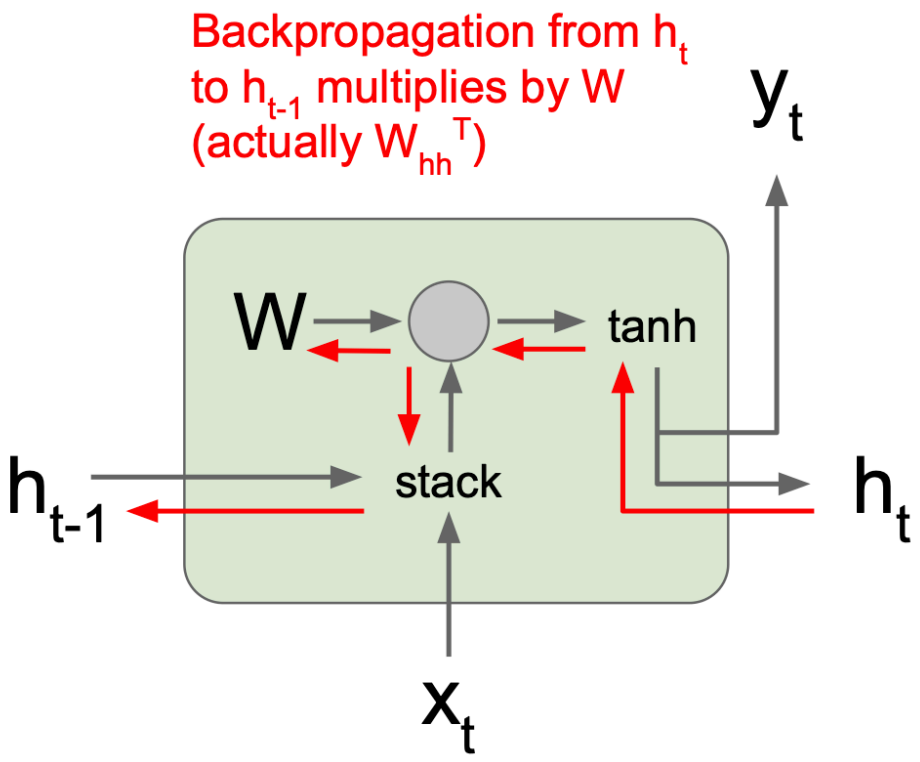
Bengio et al, “Learning long-term dependencies with gradient descent is difficult”, IEEE Transactions on Neural Networks, 1994
Pascanu et al, “On the difficulty of training recurrent neural networks”, ICML 2013



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

Vanilla RNN Gradient Flow

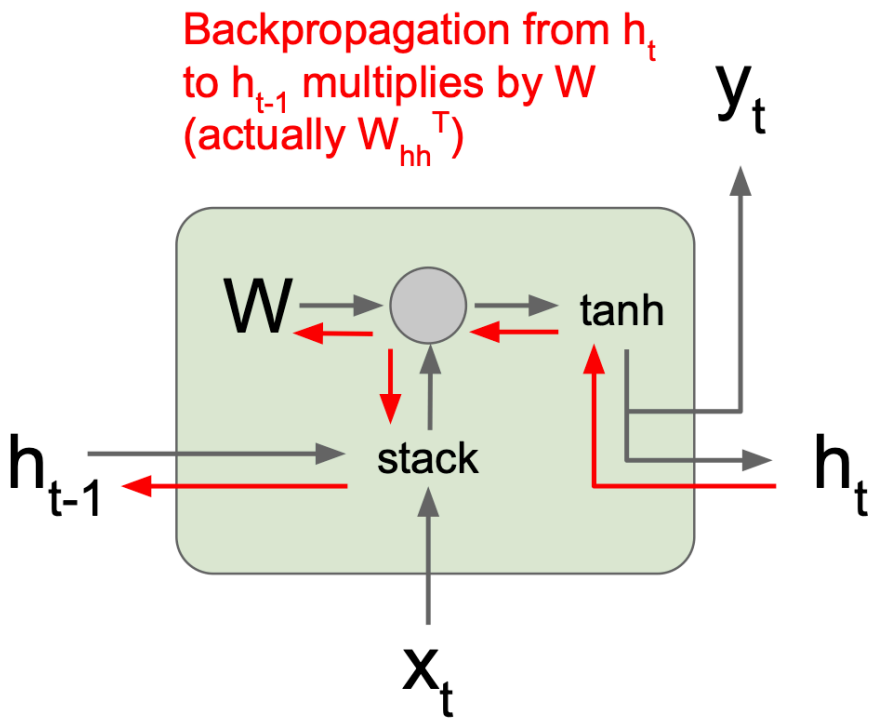
Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013



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

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

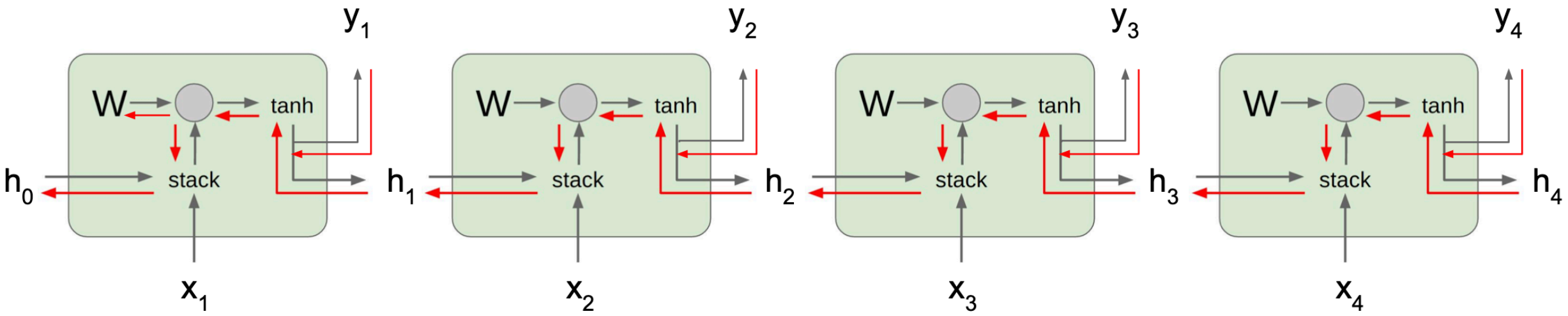


$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$
$$= \tanh\left((W_{hh} \quad W_{hx})\begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$
$$= \tanh\left(W\begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$

$$\frac{\partial h_t}{\partial h_{t-1}} = tanh'(W_{hh}h_{t-1} + W_{xh}x_t)W_{hh}$$

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

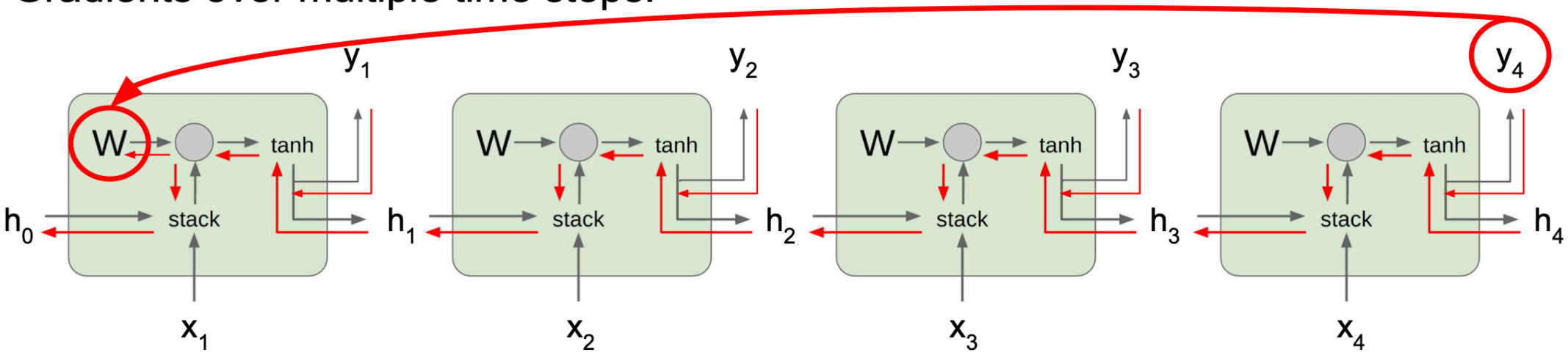


$$\frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W}$$

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

Gradients over multiple time steps:



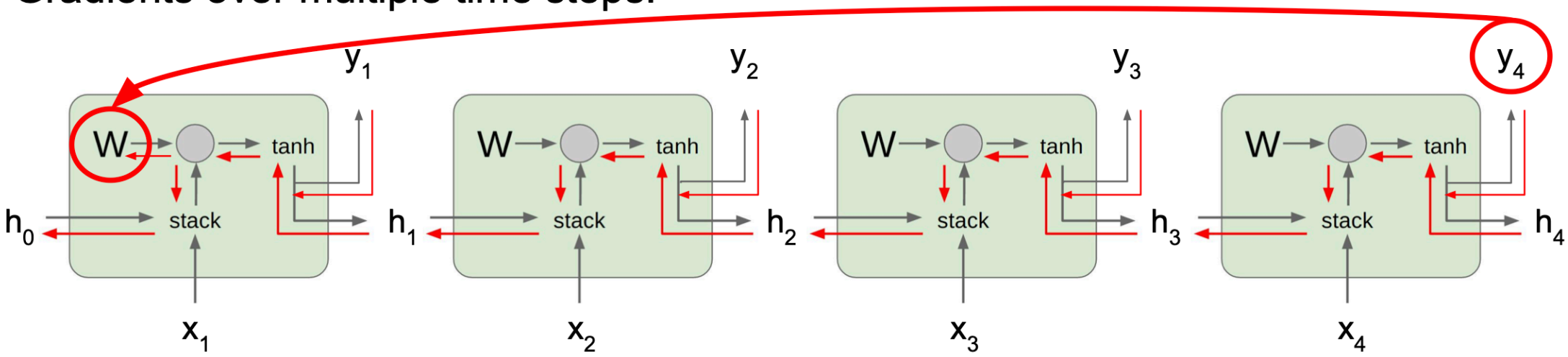
$$\frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W}$$

$$\frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial h_T} \frac{\partial h_t}{\partial h_{t-1}} \cdots \frac{\partial h_1}{\partial W}$$

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

Gradients over multiple time steps:



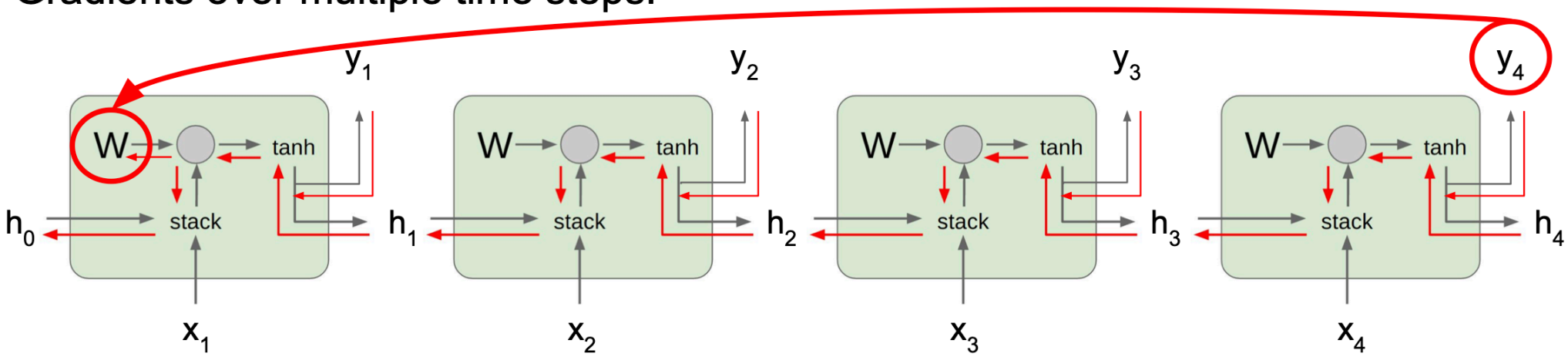
$$\frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W}$$

$$\frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial h_T} \frac{\partial h_t}{\partial h_{t-1}} \cdots \frac{\partial h_1}{\partial W} = \frac{\partial L_T}{\partial h_T} \left(\prod_{t=2}^{T} \frac{\partial h_t}{\partial h_{t-1}} \right) \frac{\partial h_1}{\partial W}$$

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

Gradients over multiple time steps:



$$\frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W}$$

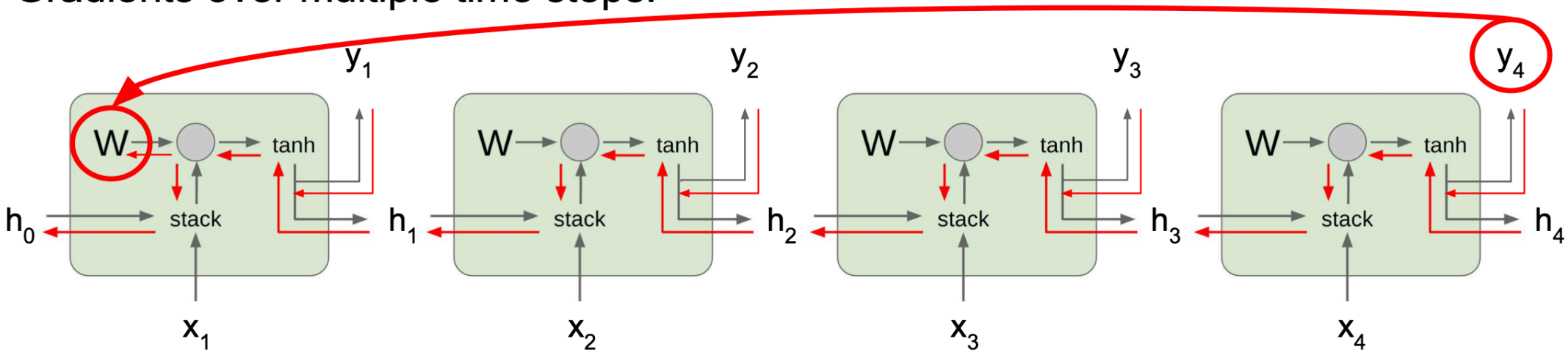
$$\frac{\partial h_t}{\partial h_{t-1}} = tanh'(W_{hh} h_{t-1} + W_{xh} x_t) W_{hh}$$

$$\frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial h_T} \frac{\partial h_t}{\partial h_{t-1}} \cdots \frac{\partial h_1}{\partial W} = \frac{\partial L_T}{\partial h_T} \left(\prod_{t=2}^{T} \frac{\partial h_t}{\partial h_{t-1}} \right) \frac{\partial h_1}{\partial W}$$

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

Gradients over multiple time steps:

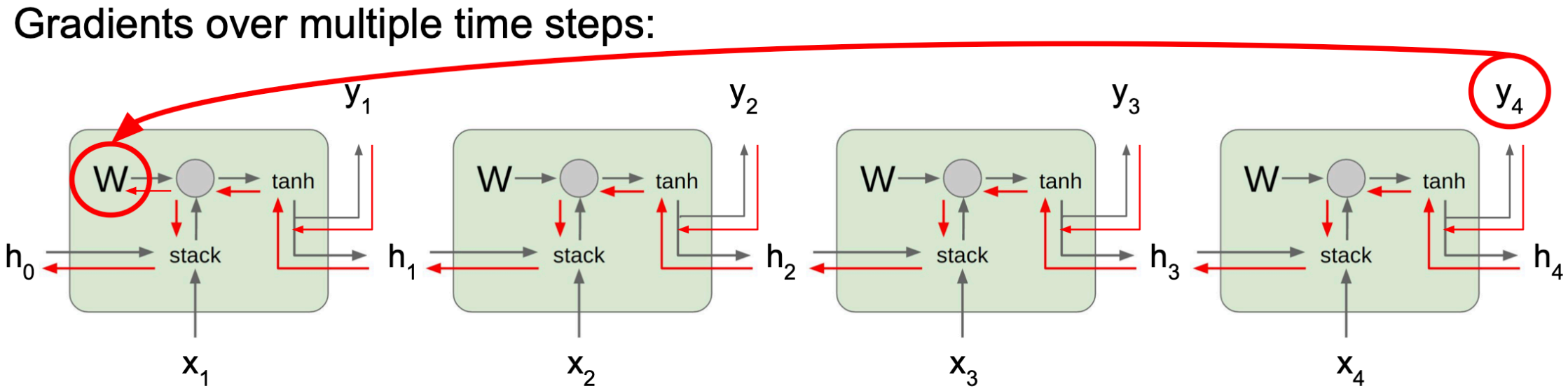


$$\frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W}$$

Almost always < 1
Vanishing gradients

$$\frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial h_T} \left(\prod_{t=2}^{T} tanh'\left(W_{hh} h_{t-1} + W_{xh} x_t\right) \right) W_{hh}^{T-1} \frac{\partial h_1}{\partial W}$$

Why is Vanishing Gradient a Problem?



- Gradient signal from far away is lost because it's much smaller than gradient signal from close-by.
- Model weights are updated only with respect to near effects, not long-term effects.

Slides credit: Stanford CS 224

Effect of Vanishing Gradient on RNN Language Model

- **Language model task:** When she tried to print her tickets, she found that the printer was out of toner. She went to the stationery store to buy more toner. It was very overpriced. After installing the toner into the printer, she finally printed her _____ .

Slides credit: Stanford CS 224

Effect of Vanishing Gradient on RNN Language Model

- **Language model task:** When she tried to print her tickets, she found that the printer was out of toner. She went to the stationery store to buy more toner. It was very overpriced. After installing the toner into the printer, she finally printed her _____ .

- To learn from this training example, the RNN-LM needs to model the dependency between *"tickets"* on the 7th step and the target word *"tickets"* at the end.

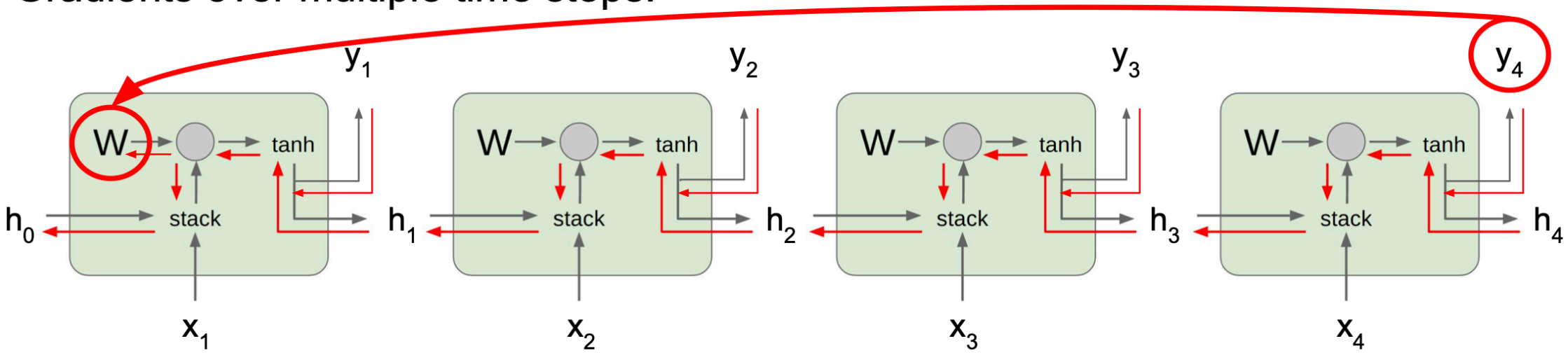
- But if the gradient is small, the model can't learn this dependency
  - then the model will be unable to predict similar long-distance dependencies at test time

Slides credit: Stanford CS 224

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

Gradients over multiple time steps:



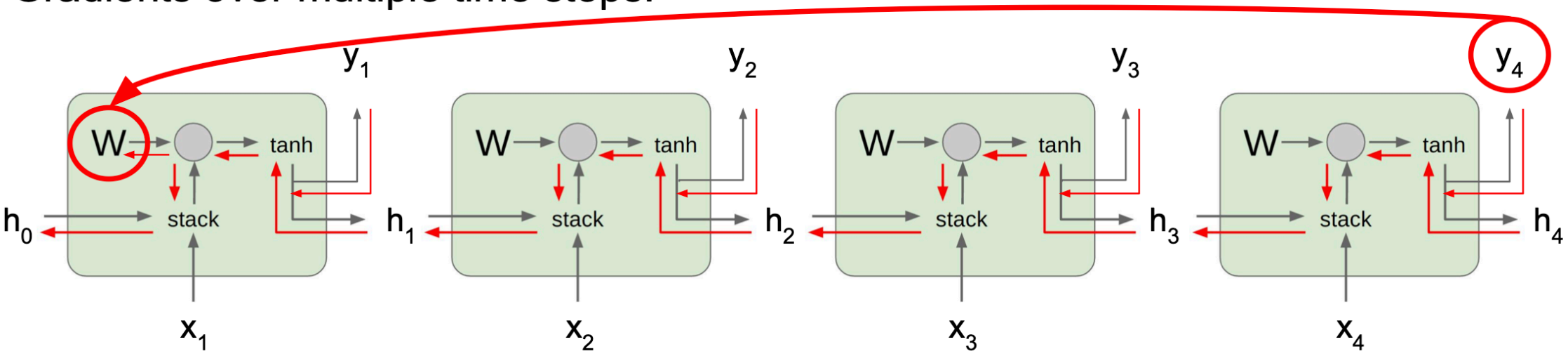
$$\frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W}$$

What if we assumed no non-linearity?

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

Gradients over multiple time steps:



What if we assumed no non-linearity?

$$\frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W}$$

$$\frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial h_T} \boxed{W_{hh}^{T-1}} \frac{\partial h_1}{\partial W}$$

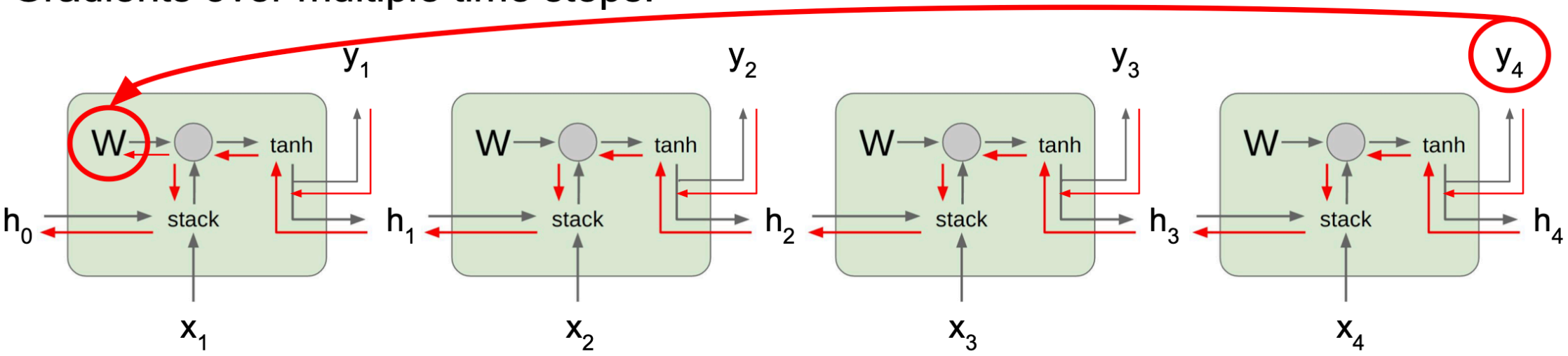
Largest singular value > 1:
Exploding gradients

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

Gradients over multiple time steps:



What if we assumed no non-linearity?

$$\frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W}$$

$$\frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial h_T} W_{hh}^{T-1} \frac{\partial h_1}{\partial W}$$

Largest singular value > 1: **Exploding gradients** → **Gradient clipping**: Scale gradient if its norm is too big

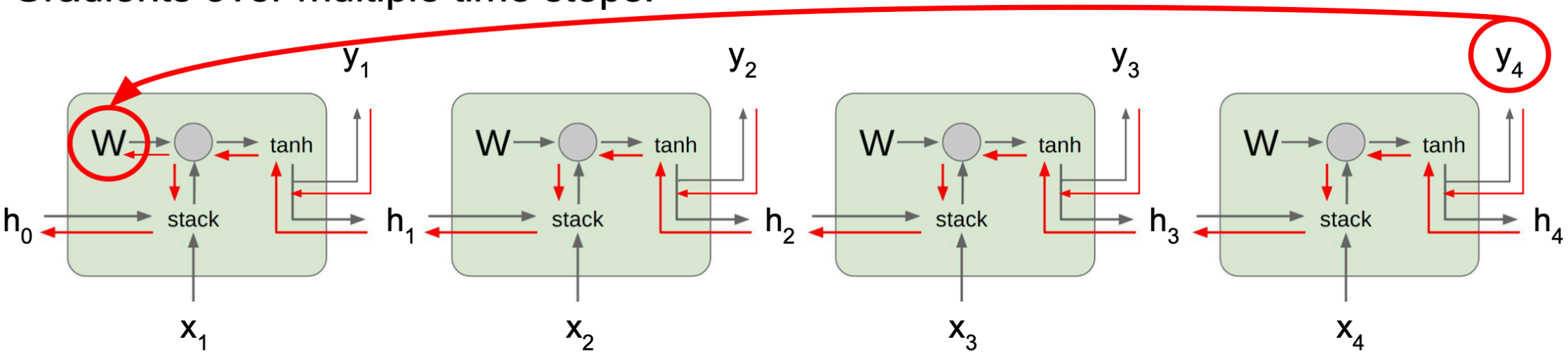
Largest singular value < 1: **Vanishing gradients**

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

Vanilla RNN Gradient Flow

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

Gradients over multiple time steps:



What if we assumed no non-linearity?

$$\frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W}$$

$$\frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial h_T} \boxed{W_{hh}^{T-1}} \frac{\partial h_1}{\partial W}$$

Largest singular value > 1:
Exploding gradients

Largest singular value < 1:
Vanishing gradients

→ Change RNN architecture

How to Fix the Vanishing Gradient Problem?

- The main problem is that it's too difficult for the RNN to learn to preserve information over many timesteps.

- In a vanilla RNN, the hidden state is constantly being rewritten

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

- How about an RNN with separate memory which is added to?

Slides credit: Stanford CS 224

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

Long Short Term Memory (LSTM)

LSTM

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

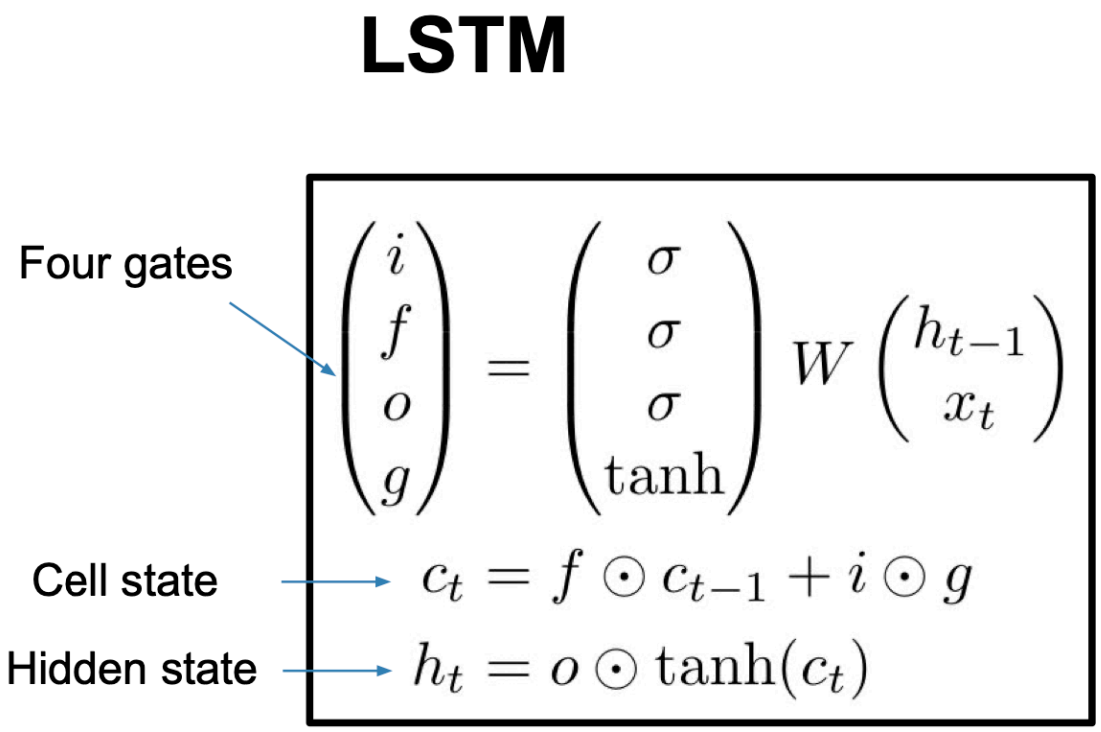
$$c_t = f \odot c_{t-1} + i \odot g$$

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

- On step t, there is a hidden state h_t and a cell state c_t
  - Both of them are of same length
- The cell state stores long-term information
- The LSTM can read, erase, and write information from the cell
  - This cell becomes conceptually rather like RAM in a computer

Slides credit: Stanford CS 224

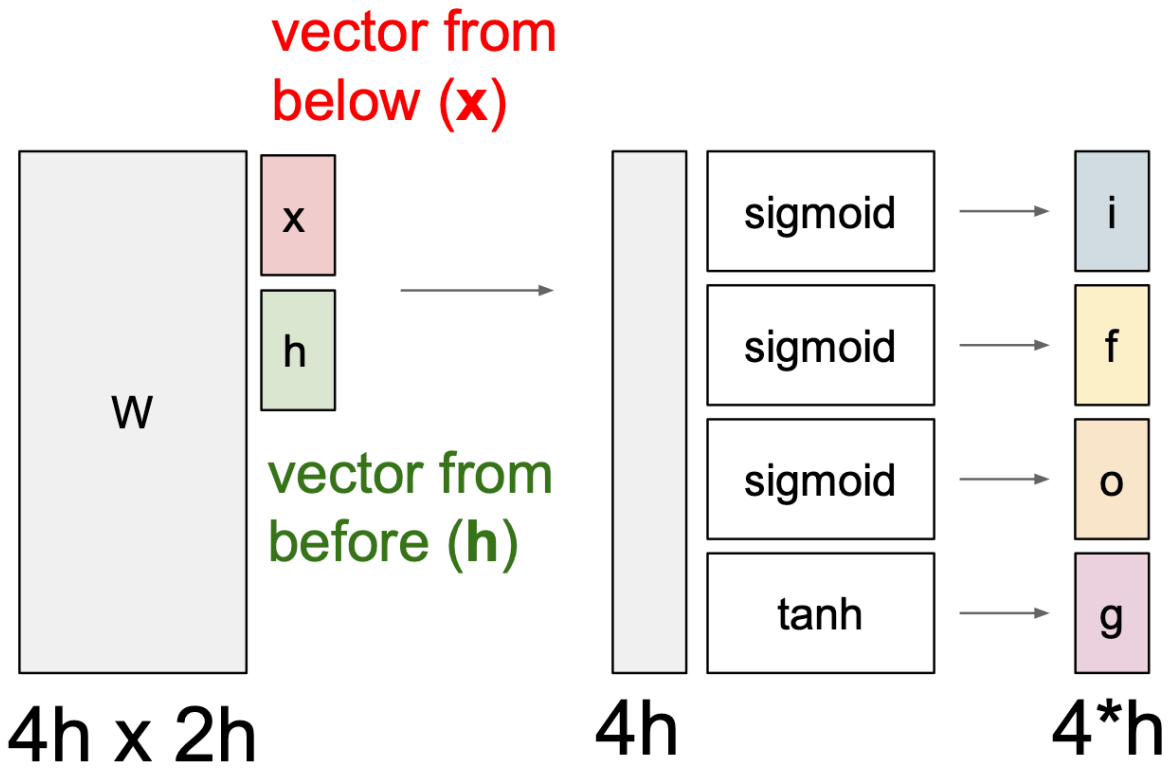
Long Short Term Memory (LSTM)



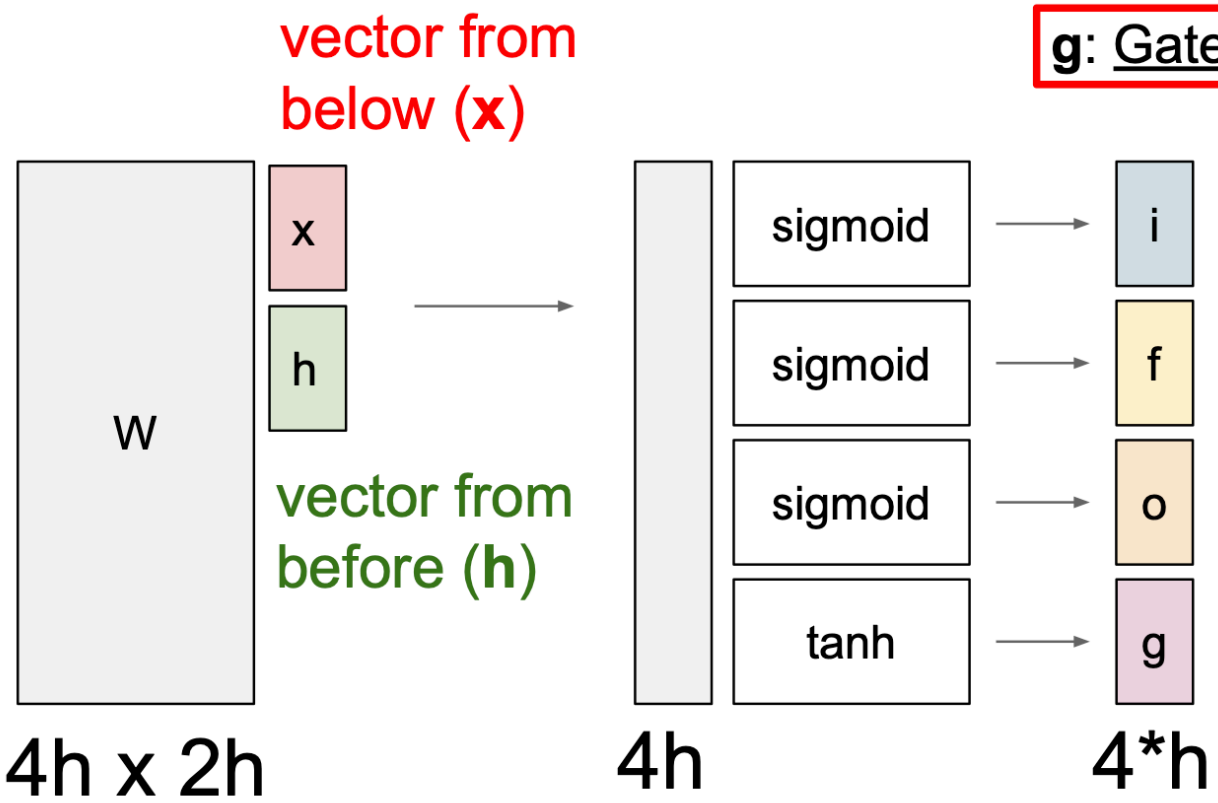
- The selection of which information is erased/written/read is controlled by three corresponding gates
- The gates are also vectors of length *n*
- On each timestep,each element of the gates can be **open**(1), **closed**(0),or somewhere in-between
- The gates are dynamic: their value is computed based on the current context

Slides credit: Stanford CS 224

Long Short Term Memory (LSTM)



Long Short Term Memory (LSTM)



g: Gate gate (?), How much to write to cell

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

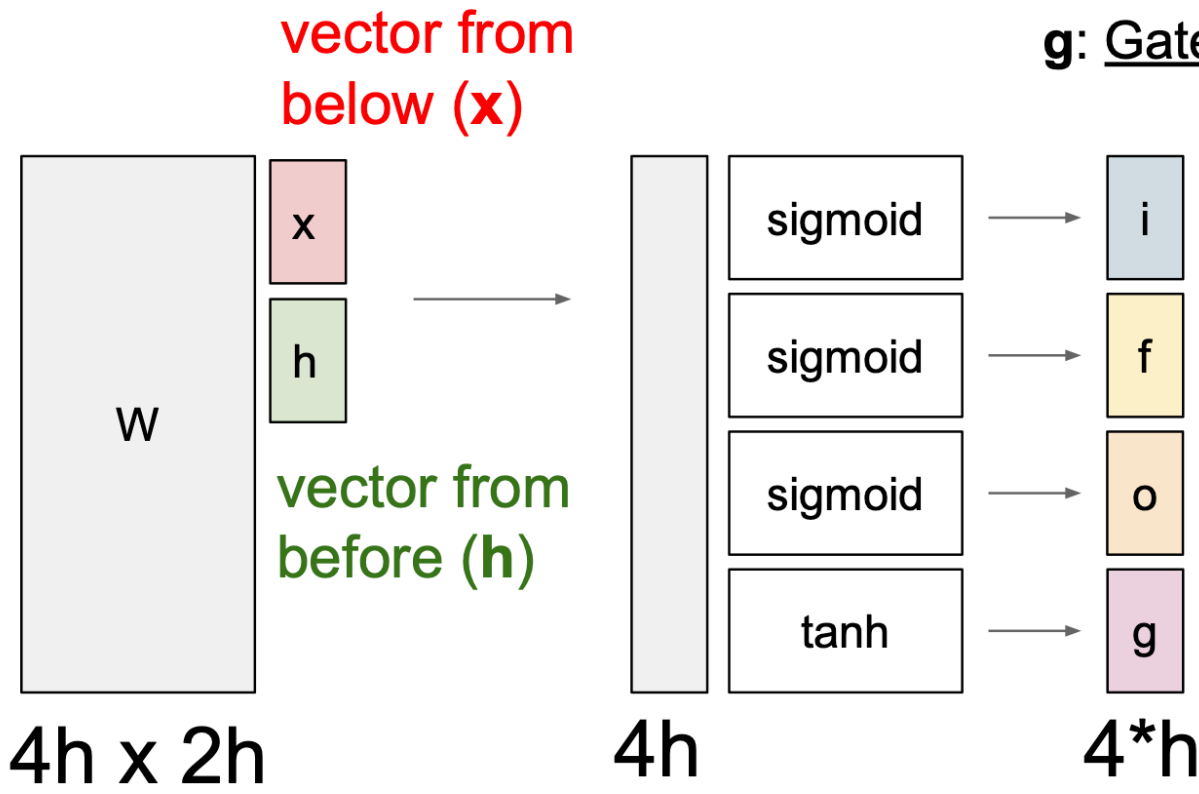
$$c_t = f \odot c_{t-1} + i \odot g$$

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

Long Short Term Memory (LSTM)

i: Input gate, whether to write to cell

g: Gate gate (?), How much to write to cell



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

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

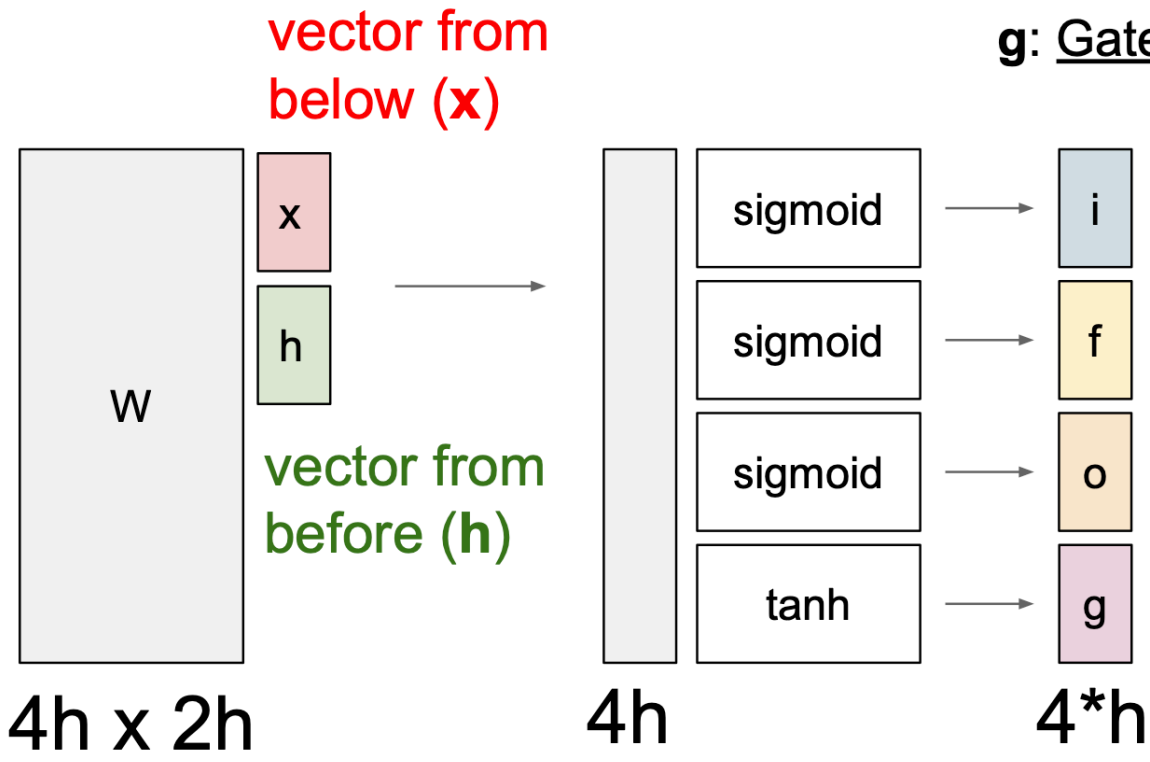
$$h_t = o \odot \tanh(c_t)$$

Long Short Term Memory (LSTM)

i: Input gate, whether to write to cell

f: Forget gate, Whether to erase cell

g: Gate gate (?), How much to write to cell



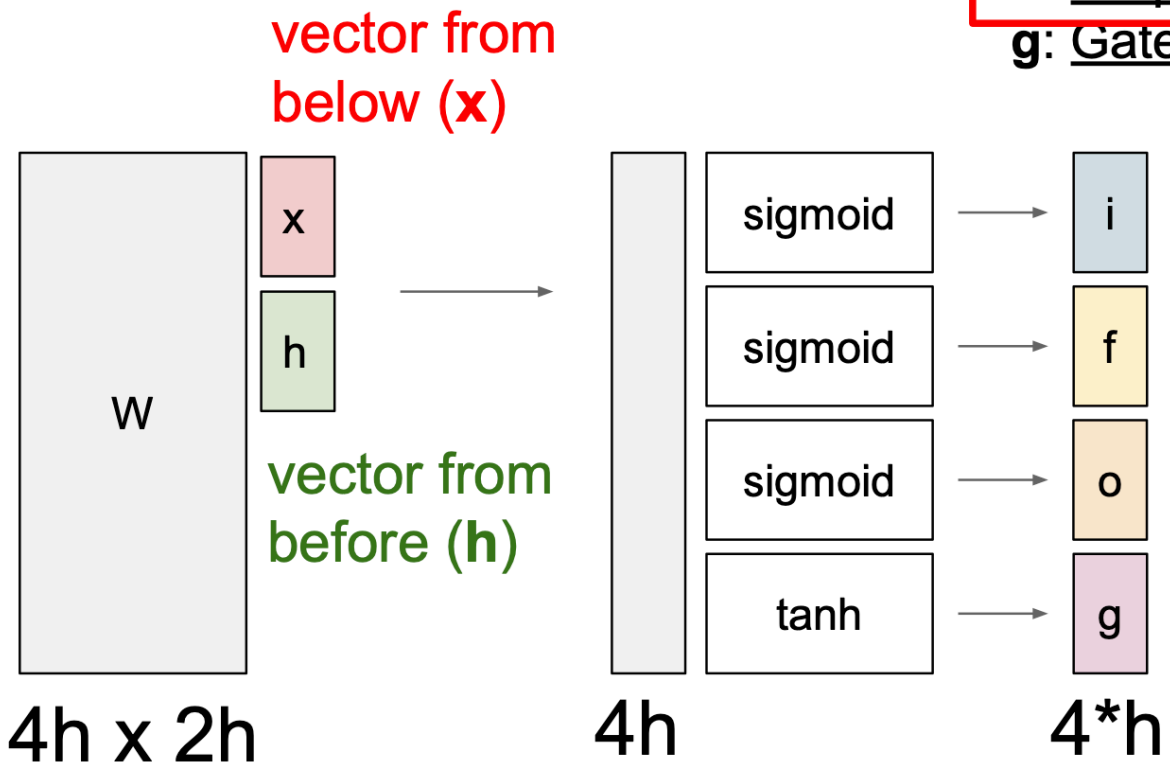
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

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

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

Long Short Term Memory (LSTM)

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

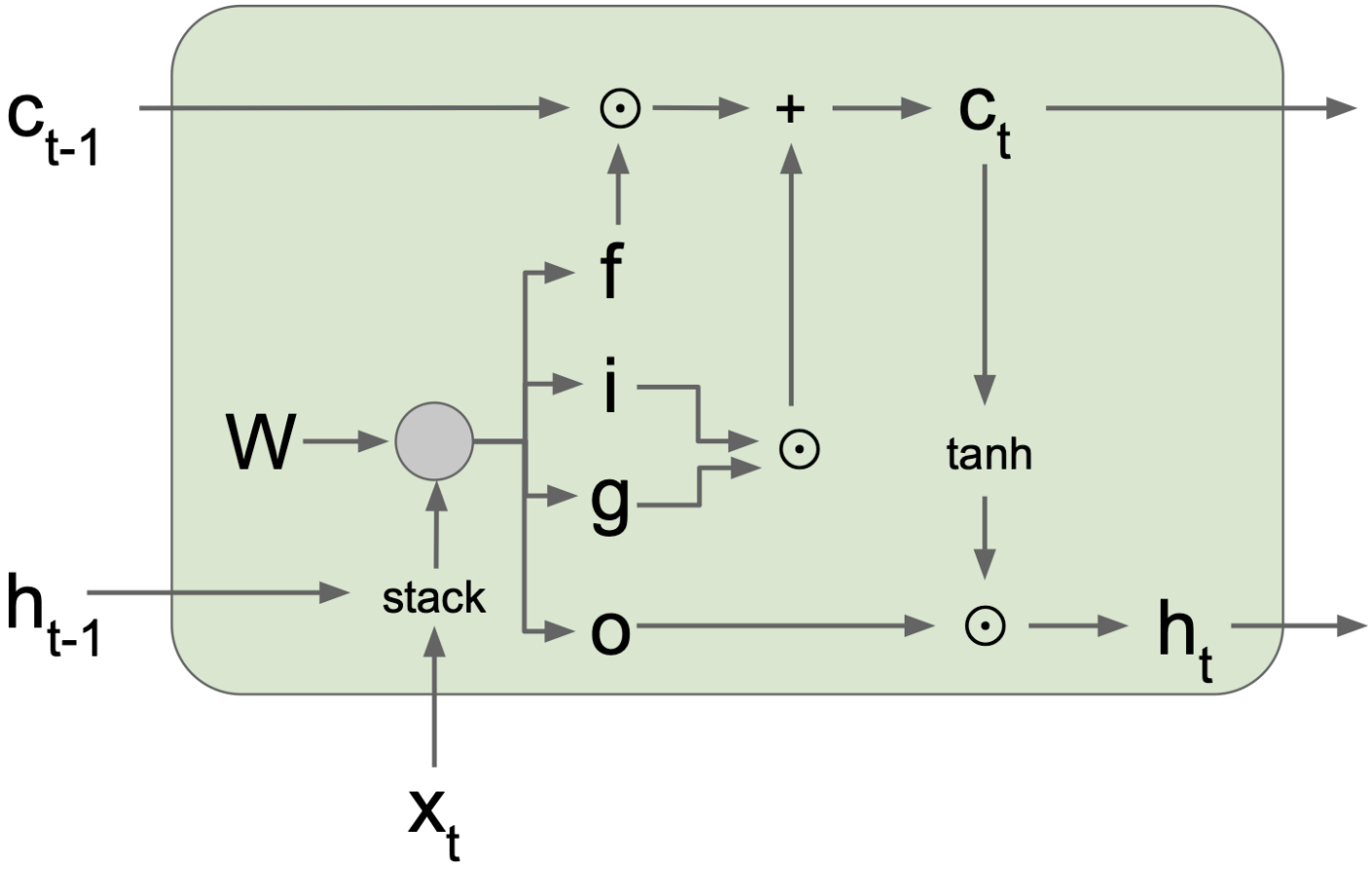


$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

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

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

Long Short Term Memory (LSTM)

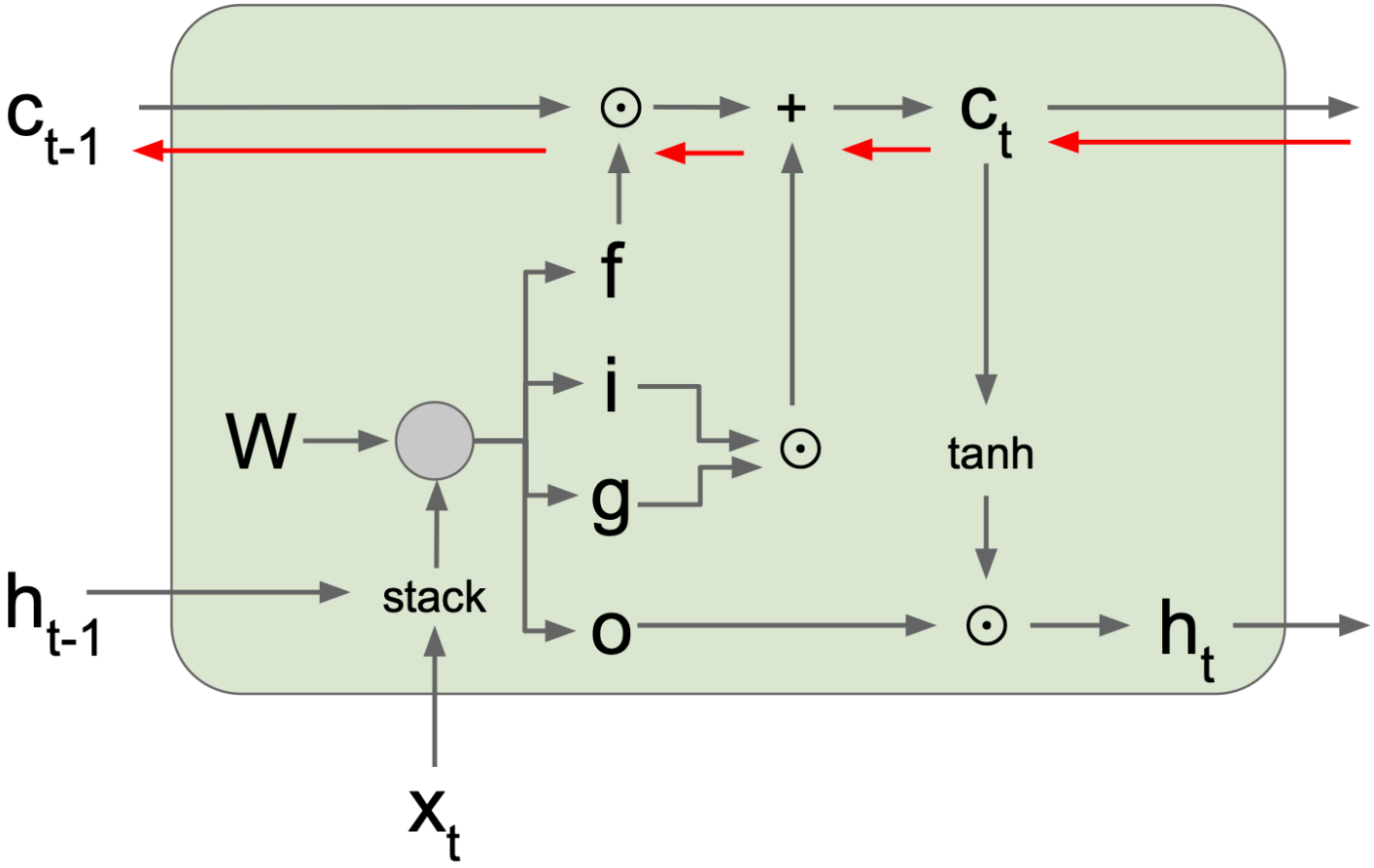


$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

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

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

LSTM Gradient Flow



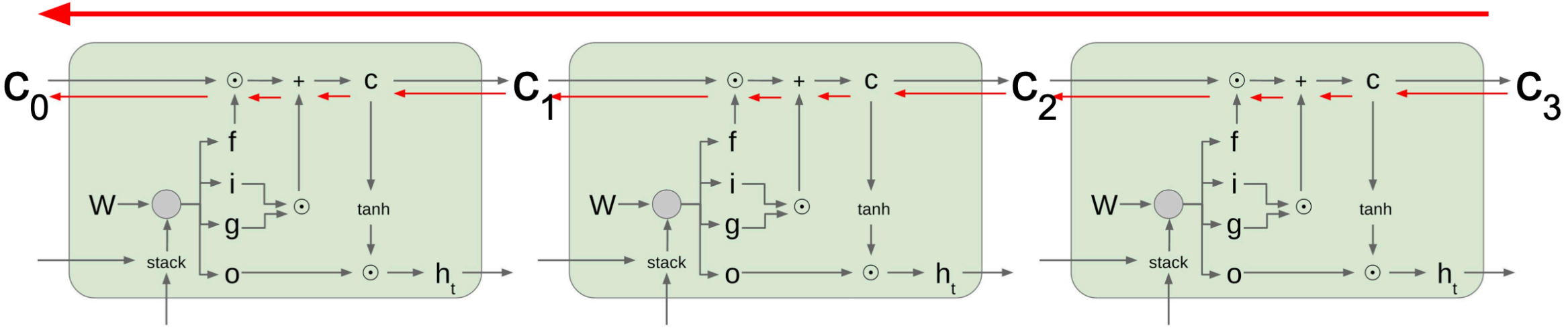
Backpropagation from c_t to c_{t-1} only elementwise multiplication by f, no matrix multiply by W

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

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

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

LSTM Gradient Flow



Notice that the gradient contains the **f** gate's vector of activations

- allows better control of gradients values, using suitable parameter updates of the forget gate.

Also notice that are added through the **f, i, g,** and **o** gates

- better balancing of gradient values

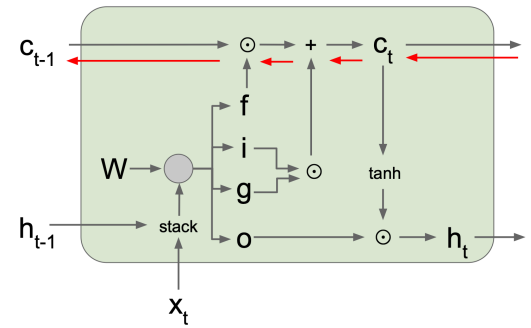
Do LSTMs Solve the Vanishing Gradient Problem?

The LSTM architecture makes it easier for the RNN to preserve information over many timesteps

- e.g. **if the f = 1 and the i = 0**, then the information of that cell is preserved indefinitely.
- By contrast, it's harder for vanilla RNN to learn a recurrent weight matrix Wh that preserves info in hidden state

LSTM **doesn't guarantee** that there is no vanishing/exploding gradient, but it does provide an easier way for the model to learn long-distance dependencies

[Hochreiter et al., 1997]

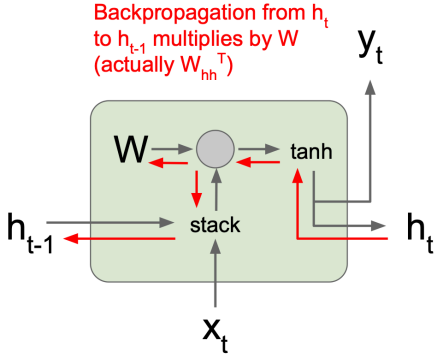


Backpropagation from c_t to c_{t-1} only elementwise multiplication by f, no matrix multiply by W

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

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

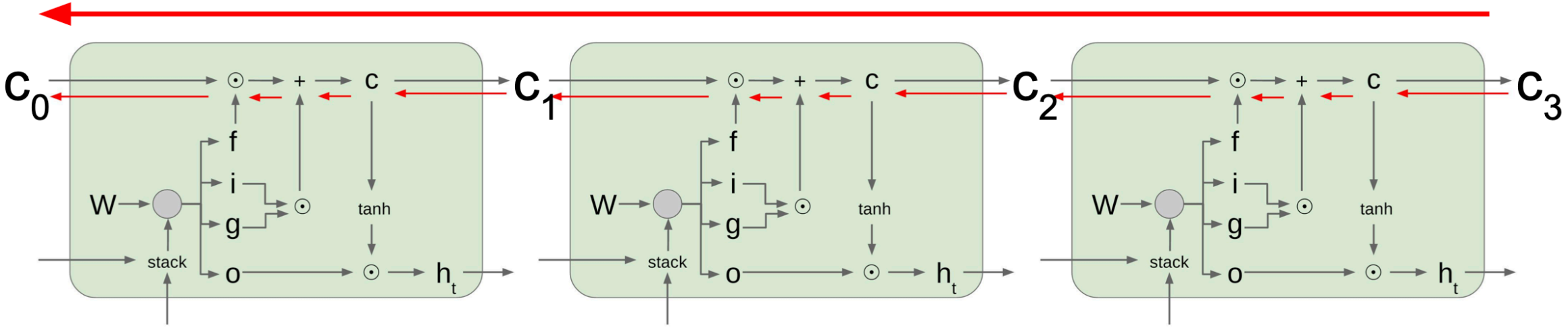
$$h_t = o \odot \tanh(c_t)$$



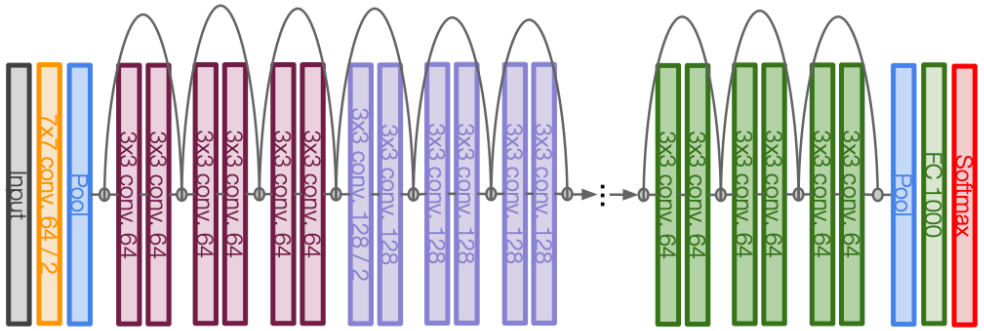
$$\begin{aligned} h_t &= \tanh(W_{hh} h_{t-1} + W_{xh} x_t) \\ &= \tanh\left(\begin{pmatrix} W_{hh} & W_{hx} \end{pmatrix} \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \right) \\ &= \tanh\left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \right) \end{aligned}$$

LSTM Gradient Flow

Uninterrupted gradient flow!



Similar to ResNet!



Other RNN Variants: Gated Recurrent Unit (GRU)

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

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

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

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

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

Summary of RNN and LSTM

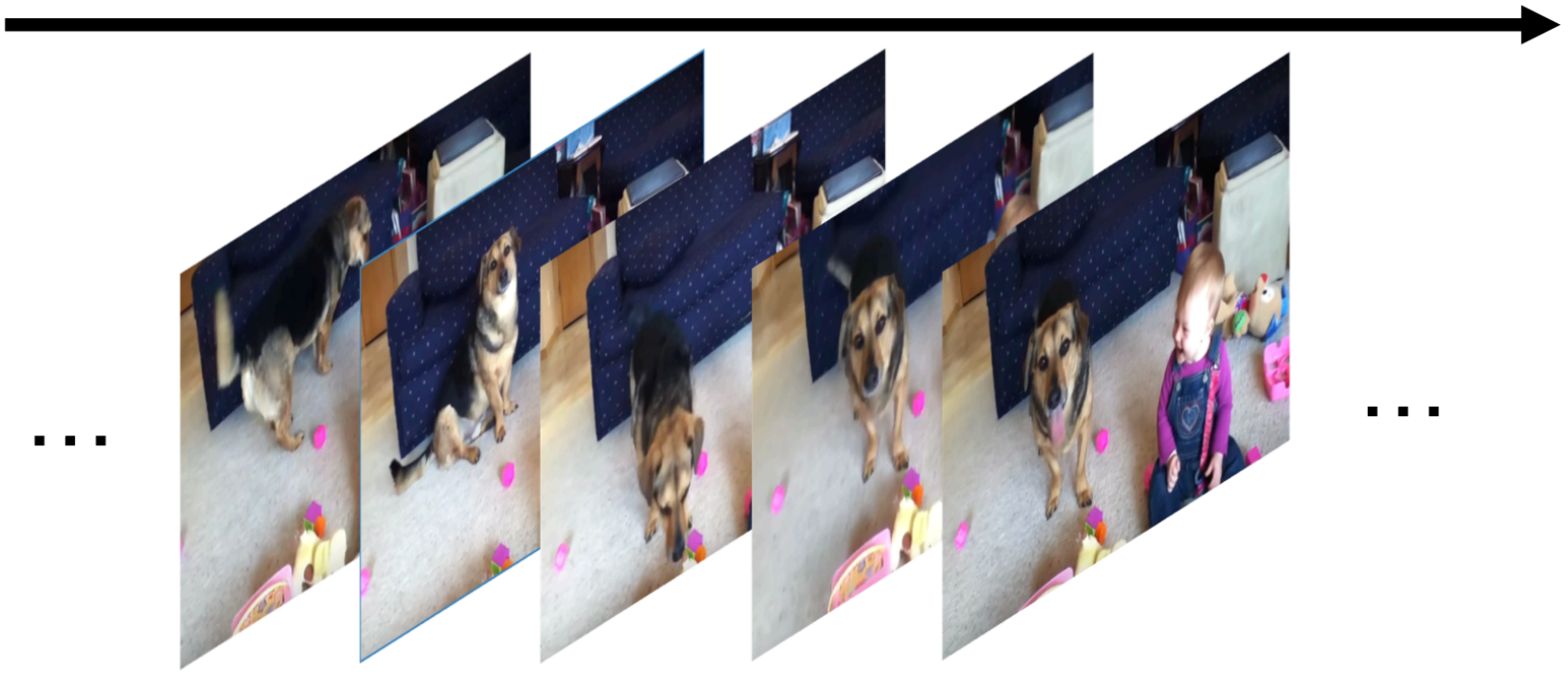
- RNNs allow a lot of flexibility in architecture design
- Vanilla RNNs are simple but don't work very well
- Common to use LSTM or GRU: their additive interactions improve gradient flow
- Backward flow of gradients in RNN can explode or vanish. Exploding is controlled with gradient clipping. Vanishing is controlled with additive interactions (LSTM)
- Better/simpler architectures are a hot topic of current research, as well as new paradigms for reasoning over sequences
- Better understanding (both theoretical and empirical) is needed.

Video Analysis

Slides are borrowed from Stanford CS231N

Video = 2D + Time

A video is a **sequence** of images
4D tensor: T x 3 x H x W
(or 3 x T x H x W)



Example Task: Video Classification

Input video:
T x 3 x H x W

Swimming
Running
Jumping
Eating
Standing

Slide credit: Justin Johnson

Example Task: Video Classification

Images: Recognize **objects**

Dog
Cat
Fish
Truck

Videos: Recognize **actions**

Swimming
Running
Jumping
Eating
Standing

Slide credit: Justin Johnson

Problems: Videos are Big!

Input video:
T x 3 x H x W

Videos are ~30 frames per second (fps)

Size of uncompressed video
(3 bytes per pixel):

SD (640 x 480): **~1.5 GB per minute**
HD (1920 x 1080): **~10 GB per minute**

Slide credit: Justin Johnson

Problems: Videos are Big!

Input video:
T x 3 x H x W

Videos are ~30 frames per second (fps)

Size of uncompressed video
(3 bytes per pixel):

SD (640 x 480): **~1.5 GB per minute**
HD (1920 x 1080): **~10 GB per minute**

Solution: Train on short **clips:**
low fps and low spatial resolution
e.g. T = 16, H=W=112
(3.2 seconds at 5 fps, 588 KB)

Slide credit: Justin Johnson

Training on Clips

Raw video: Long, high FPS

Slide credit: Justin Johnson

Training on Clips

Raw video: Long, high FPS

Training: Train model to classify short **clips** with low FPS

Slide credit: Justin Johnson

Training on Clips

Raw video: Long, high FPS

Training: Train model to classify short **clips** with low FPS

Testing: Run model on different clips, average predictions

Slide credit: Justin Johnson

Training on Clips

Raw video: Long, high FPS

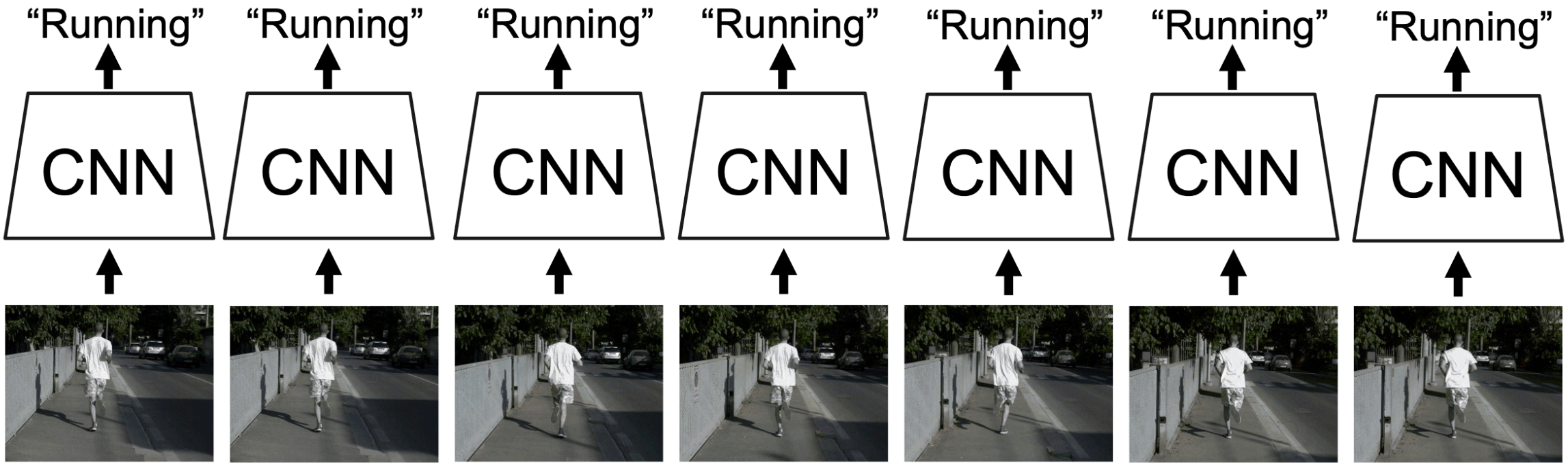
Training: Train model to classify short **clips** with low FPS

Testing: Run model on different clips, average predictions

Slide credit: Justin Johnson

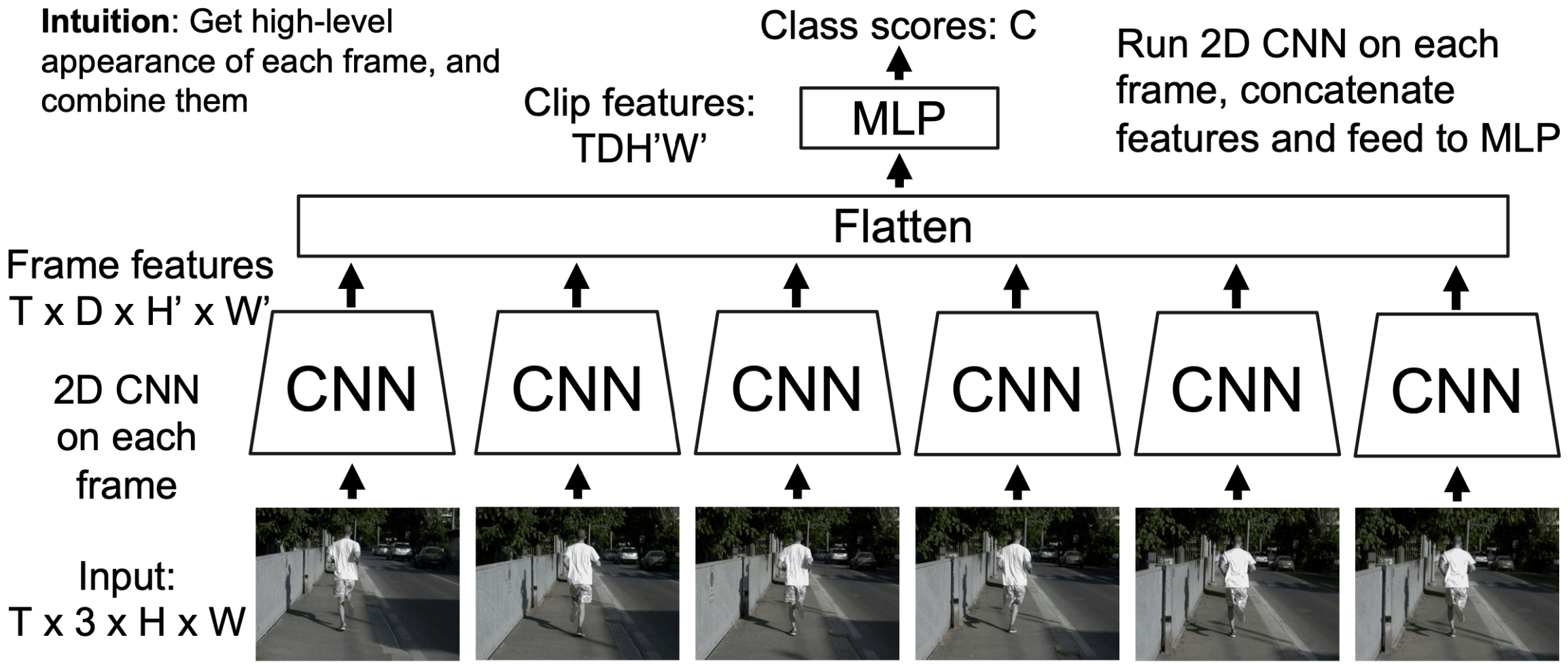
Video Classification: Single-Frame CNN

Simple idea: train normal 2D CNN to classify video frames independently!
(Average predicted probs at test-time)
Often a **very** strong baseline for video classification



Slide credit: Justin Johnson

Video Classification: Late Fusion with FC

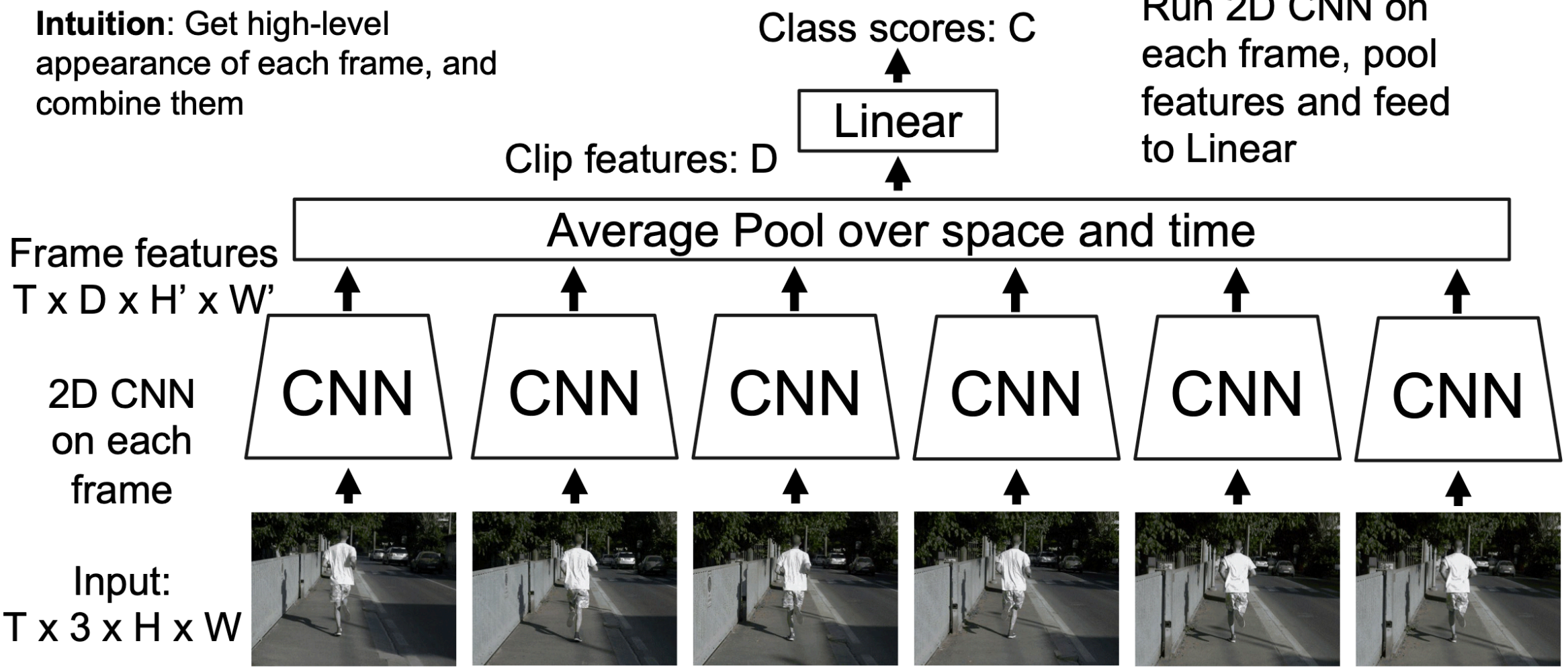


Slide credit: Justin Johnson

Karpathy et al, "Large-scale Video Classification with Convolutional Neural Networks", CVPR 2014

Video Classification: Late Fusion with Pooling

Intuition: Get high-level appearance of each frame, and combine them



Slide credit: Justin Johnson

Karpathy et al, "Large-scale Video Classification with Convolutional Neural Networks", CVPR 2014

Video Classification: Late Fusion with Pooling

Intuition: Get high-level appearance of each frame, and combine them

Problem: Hard to compare low level motion between frames

Class scores: C

Linear

Clip features: D

Run 2D CNN on each frame, pool features and feed to Linear

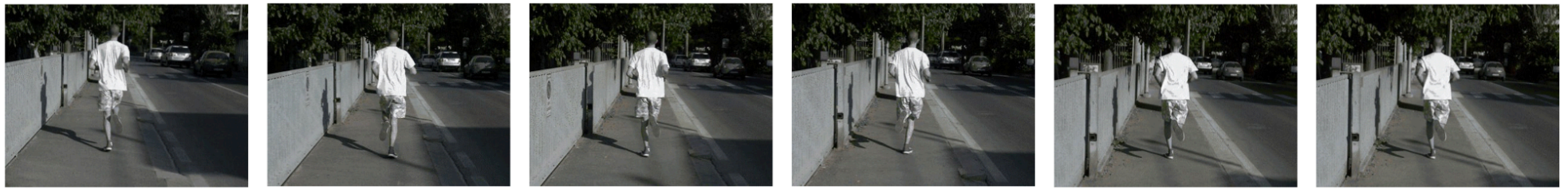
Average Pool over space and time

Frame features
T x D x H' x W'

2D CNN on each frame

CNN CNN CNN CNN CNN CNN

Input:
T x 3 x H x W

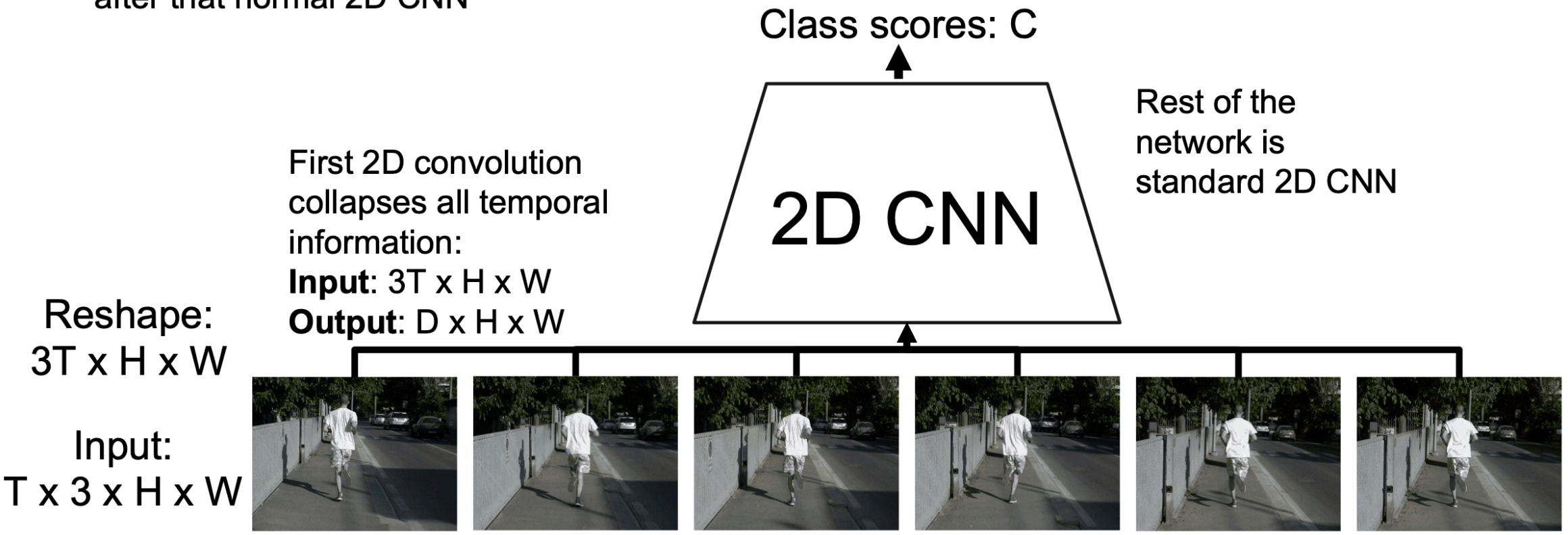


Slide credit: Justin Johnson

Karpathy et al, "Large-scale Video Classification with Convolutional Neural Networks", CVPR 2014

Video Classification: Early Fusion

Intuition: Compare frames with very first conv layer, after that normal 2D CNN



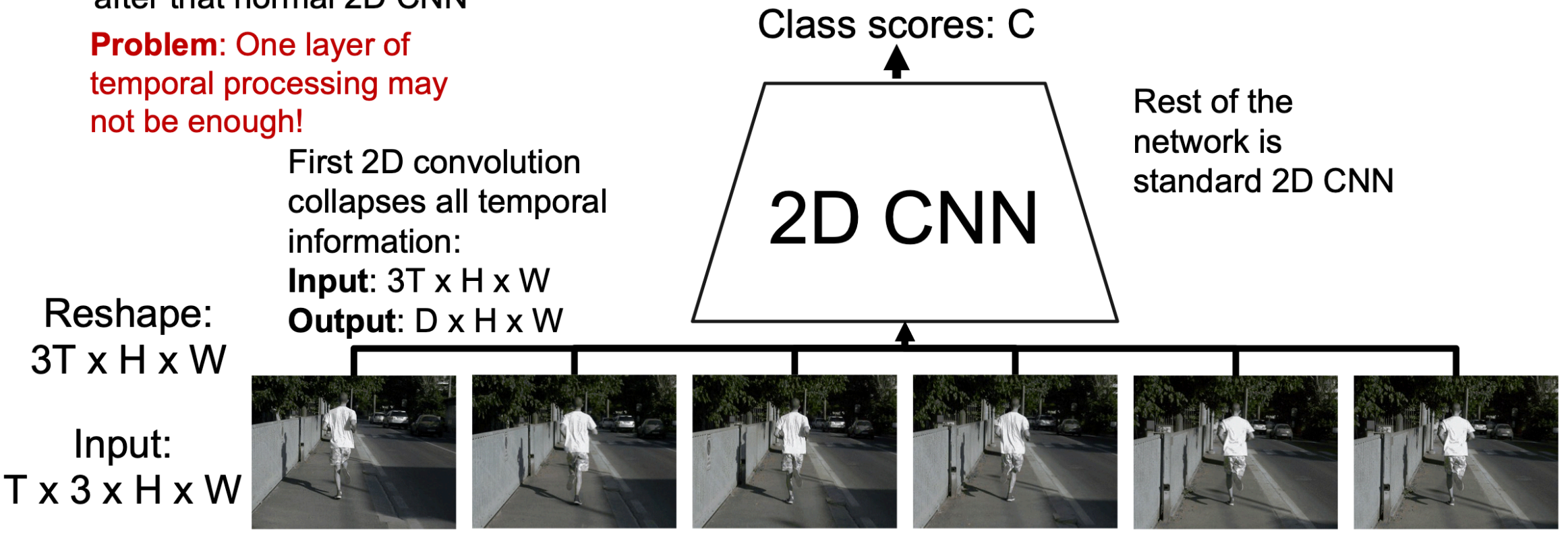
Slide credit: Justin Johnson

Karpathy et al, "Large-scale Video Classification with Convolutional Neural Networks", CVPR 2014

Video Classification: Early Fusion

Intuition: Compare frames with very first conv layer, after that normal 2D CNN

Problem: One layer of temporal processing may not be enough!



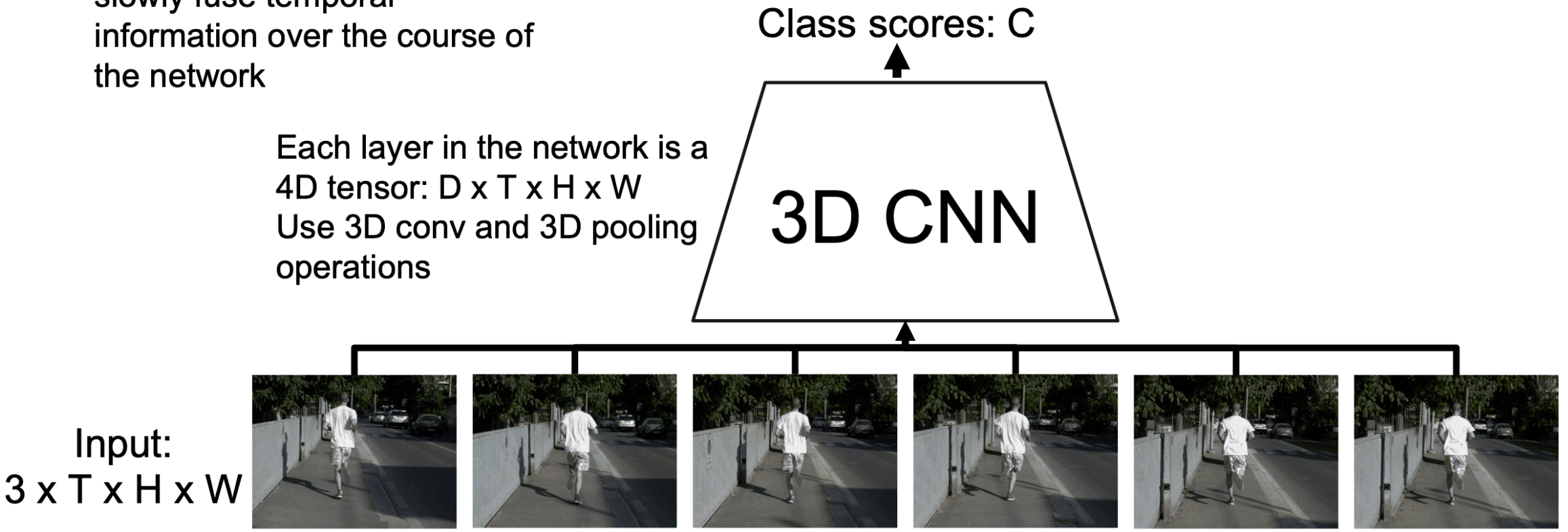
Slide credit: Justin Johnson

Karpathy et al, "Large-scale Video Classification with Convolutional Neural Networks", CVPR 2014

Video Classification: 3D CNN

Intuition: Use 3D versions of convolution and pooling to slowly fuse temporal information over the course of the network

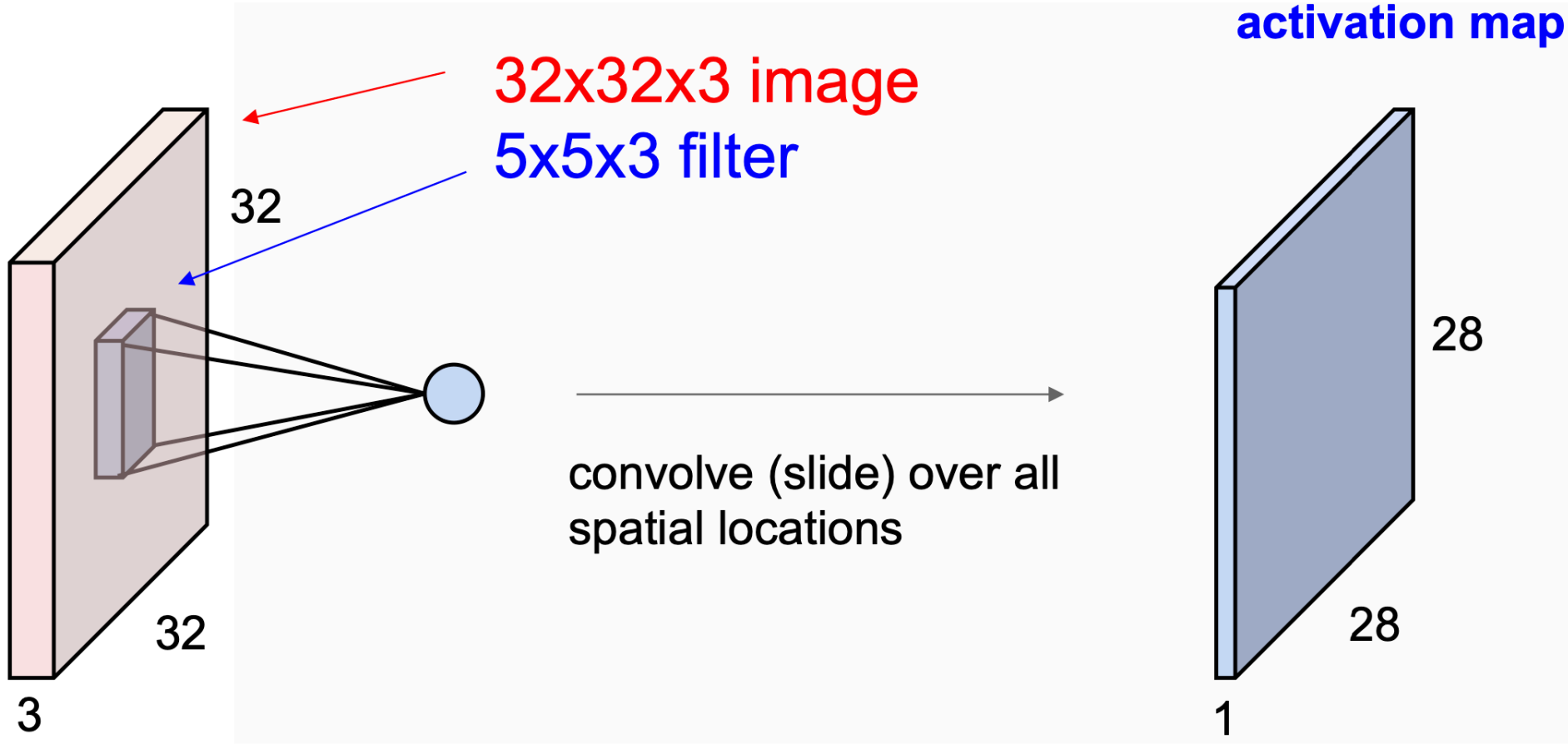
Each layer in the network is a 4D tensor: D x T x H x W
Use 3D conv and 3D pooling operations



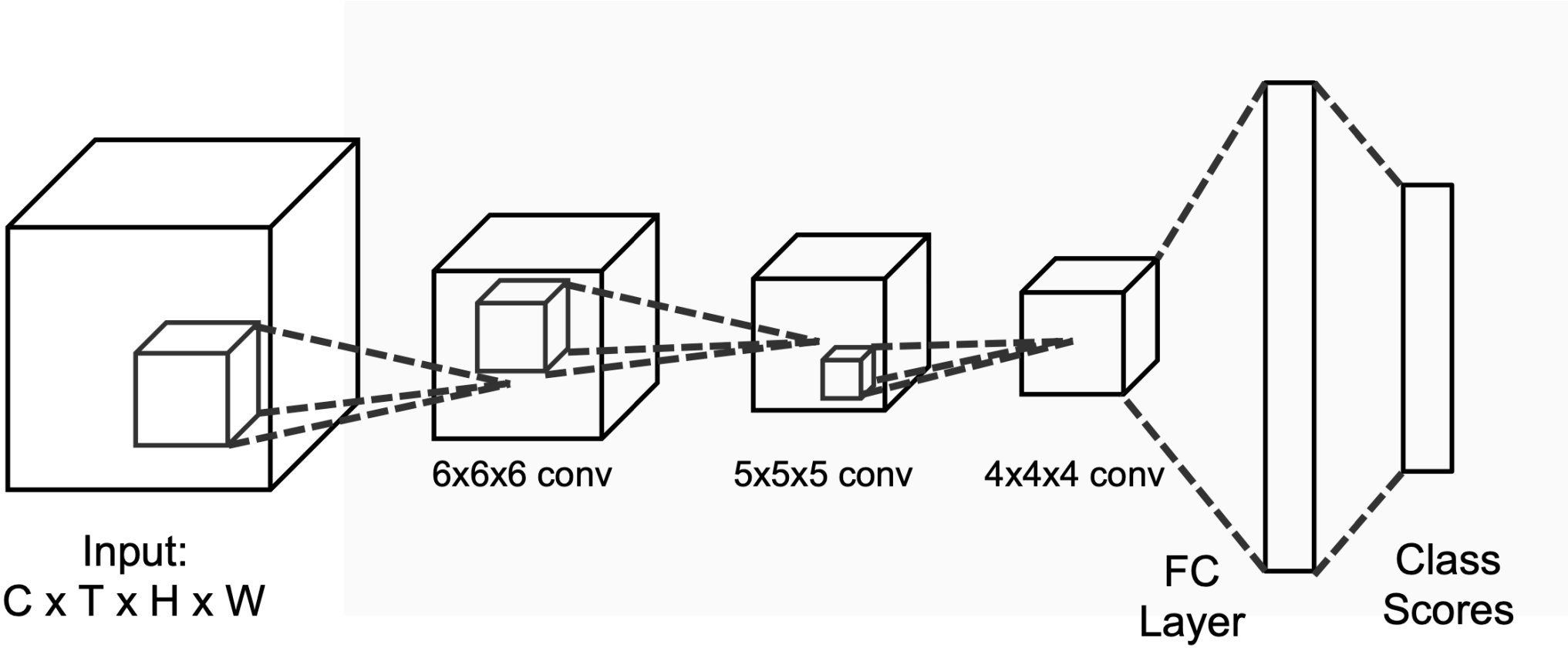
Ji et al, "3D Convolutional Neural Networks for Human Action Recognition", TPAMI 2010 ;
Karpathy et al, "Large-scale Video Classification with Convolutional Neural Networks", CVPR 2014

Slide credit: Justin Johnson

Conv Layer


activation map
32x32x3 image
5x5x3 filter
32
32
3
convolve (slide) over all
spatial locations
28
28
1

3D Convolution



Early Fusion vs. Late Fusion vs. 3DCNN

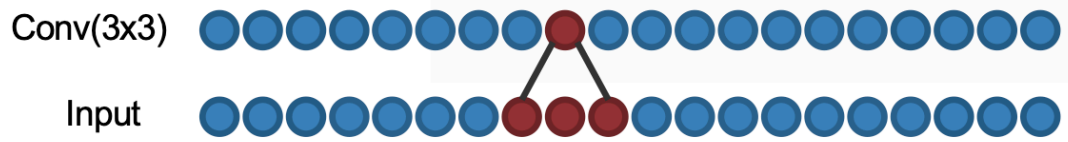
Late Fusion

Layer	Size (C x T x H x W)	Receptive Field (T x H x W)
Input	3 x 20 x 64 x 64	
Conv2D(3x3, 3->12)	12 x 20 x 64 x 64	1 x 3 x 3

Early Fusion vs. Late Fusion vs. 3DCNN

Late Fusion

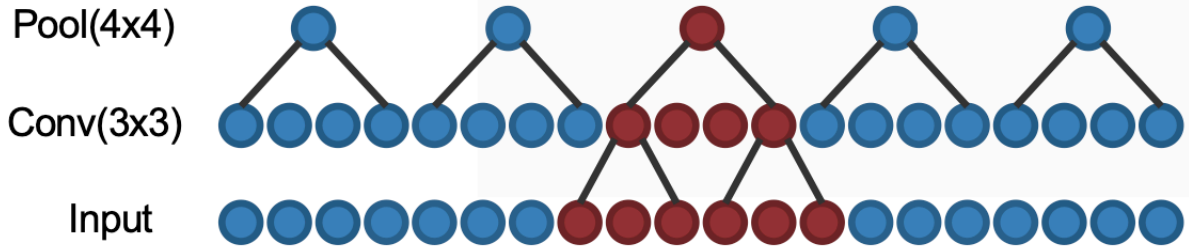
Layer	Size (C x T x H x W)	Receptive Field (T x H x W)
Input	3 x 20 x 64 x 64	
Conv2D(3x3, 3->12)	12 x 20 x 64 x 64	1 x 3 x 3



Early Fusion vs. Late Fusion vs. 3DCNN

Late Fusion

Layer	Size (C x T x H x W)	Receptive Field (T x H x W)
Input	3 x 20 x 64 x 64	
Conv2D(3x3, 3->12)	12 x 20 x 64 x 64	1 x 3 x 3
Pool2D(4x4)	12 x 20 x 16 x 16	1 x 6 x 6

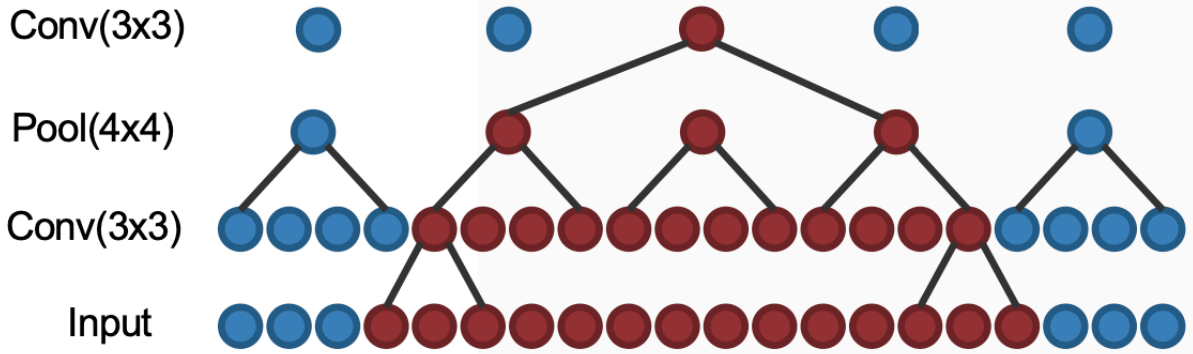


Early Fusion vs. Late Fusion vs. 3DCNN

Late Fusion

Layer	Size (C x T x H x W)	Receptive Field (T x H x W)
Input	3 x 20 x 64 x 64	
Conv2D(3x3, 3->12)	12 x 20 x 64 x 64	1 x 3 x 3
Pool2D(4x4)	12 x 20 x 16 x 16	1 x 6 x 6
Conv2D(3x3, 12->24)	24 x 20 x 16 x 16	1 x 14 x 14

Build slowly in space

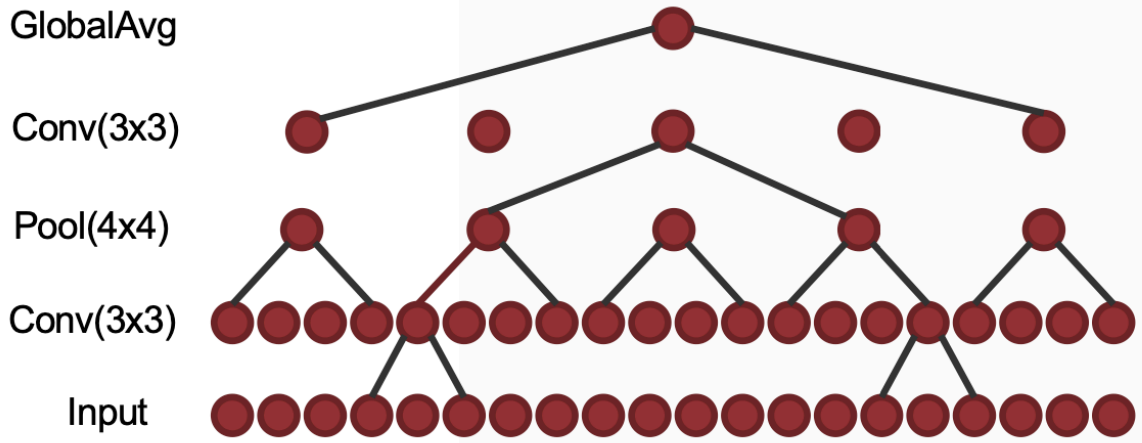


Early Fusion vs. Late Fusion vs. 3DCNN

Late Fusion

Layer	Size (C x T x H x W)	Receptive Field (T x H x W)
Input	3 x 20 x 64 x 64	
Conv2D(3x3, 3->12)	12 x 20 x 64 x 64	1 x 3 x 3
Pool2D(4x4)	12 x 20 x 16 x 16	1 x 6 x 6
Conv2D(3x3, 12->24)	24 x 20 x 16 x 16	1 x 14 x 14
GlobalAvgPool	24 x 1 x 1 x 1	20 x 64 x 64

Build slowly in space,
All-at-once in time at end



Early Fusion vs. Late Fusion vs. 3DCNN

	Layer	Size (C x T x H x W)	Receptive Field (T x H x W)	
Late Fusion	Input	3 x 20 x 64 x 64		
	Conv2D(3x3, 3->12)	12 x 20 x 64 x 64	1 x 3 x 3	Build slowly in space, All-at-once in time at end
	Pool2D(4x4)	12 x 20 x 16 x 16	1 x 6 x 6	
	Conv2D(3x3, 12->24)	24 x 20 x 16 x 16	1 x 14 x 14	
	GlobalAvgPool	24 x 1 x 1 x 1	20 x 64 x 64	
Early Fusion	Input	3 x 20 x 64 x 64		
	Conv2D(3x3, 3*20->12)	12 x 64 x 64	20 x 3 x 3	Build slowly in space, All-at-once in time at start
	Pool2D(4x4)	12 x 16 x 16	20 x 6 x 6	
	Conv2D(3x3, 12->24)	24 x 16 x 16	20 x 14 x 14	
	GlobalAvgPool	24 x 1 x 1	20 x 64 x 64	

Early Fusion vs. Late Fusion vs. 3DCNN

	Layer	Size (C x T x H x W)	Receptive Field (T x H x W)	
Late Fusion	Input	3 x 20 x 64 x 64		
	Conv2D(3x3, 3->12)	12 x 20 x 64 x 64	1 x 3 x 3	Build slowly in space, All-at-once in time at end
	Pool2D(4x4)	12 x 20 x 16 x 16	1 x 6 x 6	
	Conv2D(3x3, 12->24)	24 x 20 x 16 x 16	1 x 14 x 14	
	GlobalAvgPool	24 x 1 x 1 x 1	20 x 64 x 64	
Early Fusion	Input	3 x 20 x 64 x 64		
	Conv2D(3x3, 3*20->12)	12 x 64 x 64	20 x 3 x 3	Build slowly in space, All-at-once in time at start
	Pool2D(4x4)	12 x 16 x 16	20 x 6 x 6	
	Conv2D(3x3, 12->24)	24 x 16 x 16	20 x 14 x 14	
	GlobalAvgPool	24 x 1 x 1	20 x 64 x 64	
3D CNN	Input	3 x 20 x 64 x 64		
	Conv3D(3x3x3, 3->12)	12 x 20 x 64 x 64	3 x 3 x 3	Build slowly in space, Build slowly in time "Slow Fusion"
	Pool3D(4x4x4)	12 x 5 x 16 x 16	6 x 6 x 6	
	Conv3D(3x3x3, 12->24)	24 x 5 x 16 x 16	14 x 14 x 14	
	GlobalAvgPool	24 x 1 x 1	20 x 64 x 64	

Early Fusion vs. Late Fusion vs. 3DCNN

	Layer	Size (C x T x H x W)	Receptive Field (T x H x W)
Late Fusion	Input	3 x 20 x 64 x 64	
	Conv2D(3x3, 3->12)	12 x 20 x 64 x 64	1 x 3 x 3
	Pool2D(4x4)	12 x 20 x 16 x 16	1 x 6 x 6
	Conv2D(3x3, 12->24)	24 x 20 x 16 x 16	1 x 14 x 14
	GlobalAvgPool	24 x 1 x 1 x 1	20 x 64 x 64
Early Fusion	Input	3 x 20 x 64 x 64	
	Conv2D(3x3, 3*20->12)	12 x 64 x 64	20 x 3 x 3
	Pool2D(4x4)	12 x 16 x 16	20 x 6 x 6
	Conv2D(3x3, 12->24)	24 x 16 x 16	20 x 14 x 14
	GlobalAvgPool	24 x 1 x 1	20 x 64 x 64
3D CNN	Input	3 x 20 x 64 x 64	
	Conv3D(3x3x3, 3->12)	12 x 20 x 64 x 64	3 x 3 x 3
	Pool3D(4x4x4)	12 x 5 x 16 x 16	6 x 6 x 6
	Conv3D(3x3x3, 12->24)	24 x 5 x 16 x 16	14 x 14 x 14
	GlobalAvgPool	24 x 1 x 1	20 x 64 x 64

What is the difference?

Late Fusion: Build slowly in space, All-at-once in time at end

Early Fusion: Build slowly in space, All-at-once in time at start

3D CNN: Build slowly in space, Build slowly in time "Slow Fusion"

(Small example architectures, in practice much bigger)

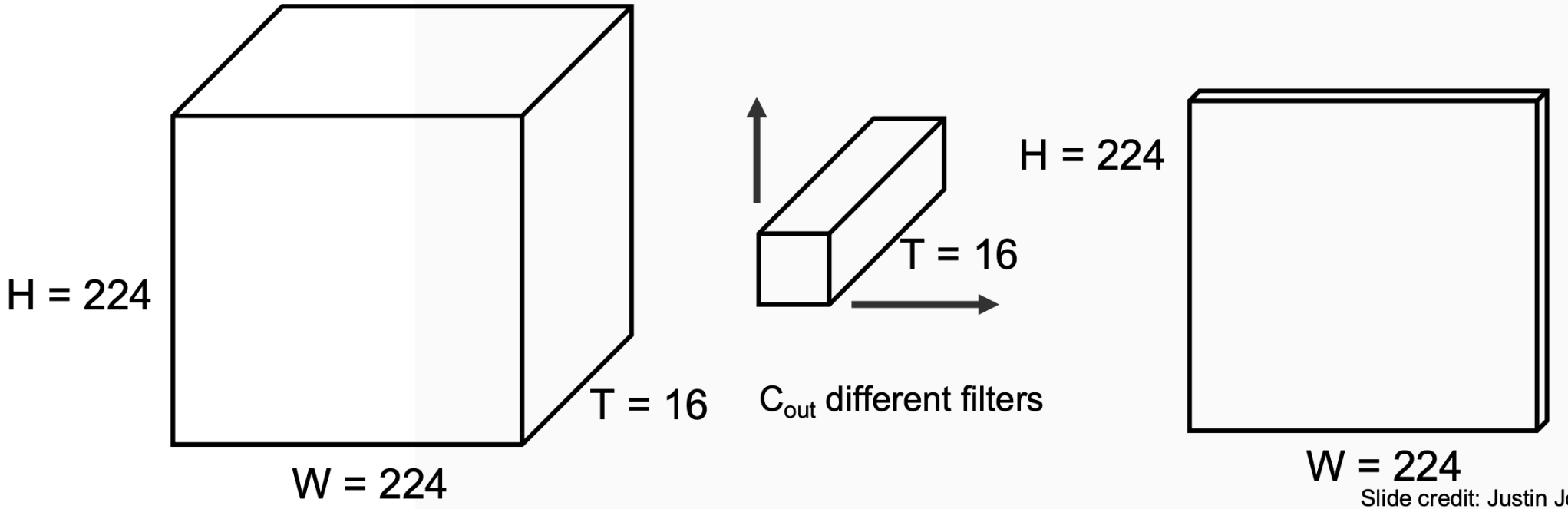
Slide credit: Justin Johnson

2D Conv (Early Fusion) vs 3D Conv (3D CNN)

Input: C_{in} x T x H x W
(3D grid with C_{in}-dim feat at each point)

Weight:
C_{out} x C_{in} x T x 3 x 3
Slide over x and y

Output:
C_{out} x H x W
2D grid with C_{out} – dim feat at each point



Slide credit: Justin Johnson

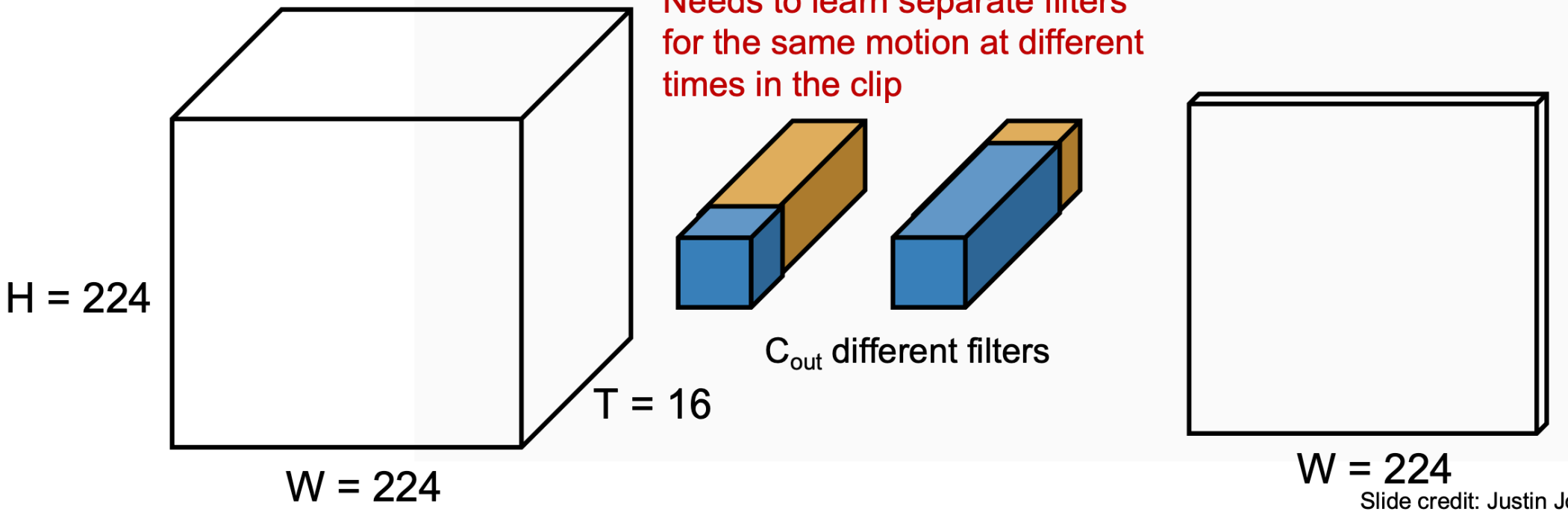
2D Conv (Early Fusion) vs 3D Conv (3D CNN)

Input: C_{in} x T x H x W
(3D grid with C_{in}-dim feat at each point)

Weight:
C_{out} x C_{in} x T x 3 x 3
Slide over x and y

No temporal shift-invariance!
Needs to learn separate filters for the same motion at different times in the clip

Output:
C_{out} x H x W
2D grid with C_{out} –dim feat at each point



Slide credit: Justin Johnson

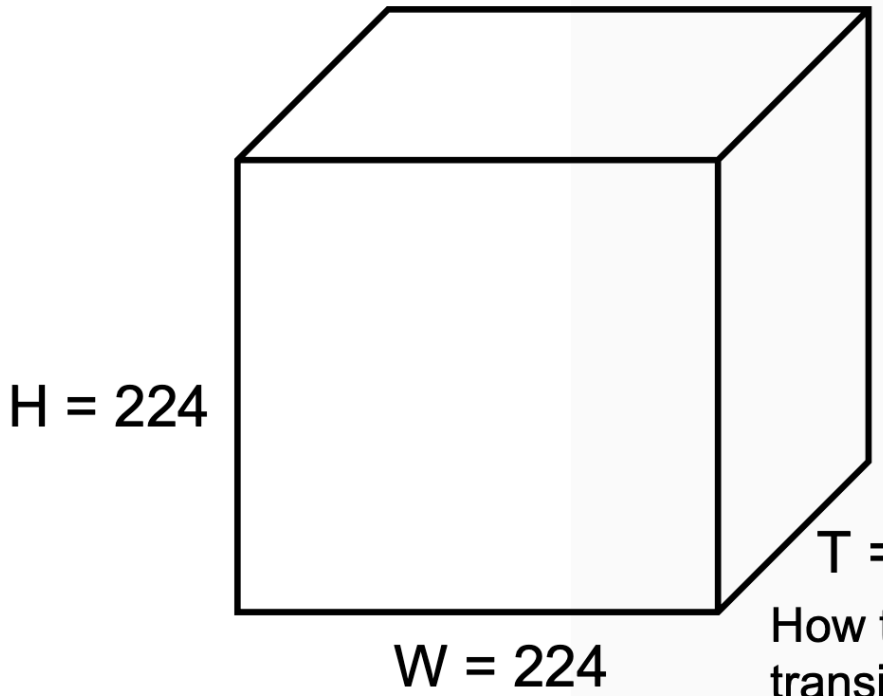
2D Conv (Early Fusion) vs 3D Conv (3D CNN)

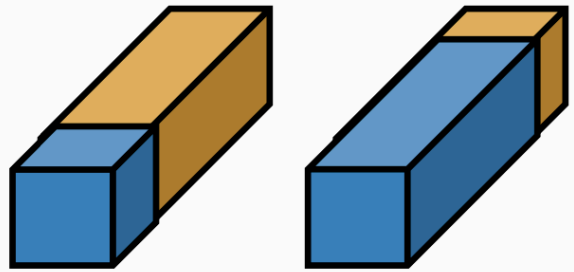
Input: C_{in} x T x H x W
(3D grid with C_{in}-dim feat at each point)

Weight:
C_{out} x C_{in} x T x 3 x 3
Slide over x and y

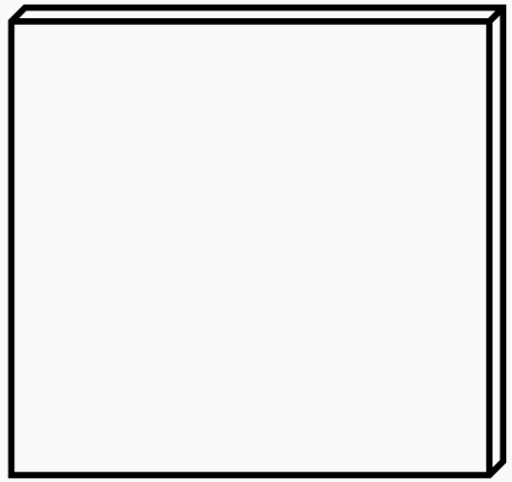
No temporal shift-invariance!
Needs to learn separate filters for the same motion at different times in the clip

Output:
C_{out} x H x W
2D grid with C_{out} –dim feat at each point





C_{out} different filters



W = 224

How to recognize **blue** to **orange** transitions anywhere in space and time?

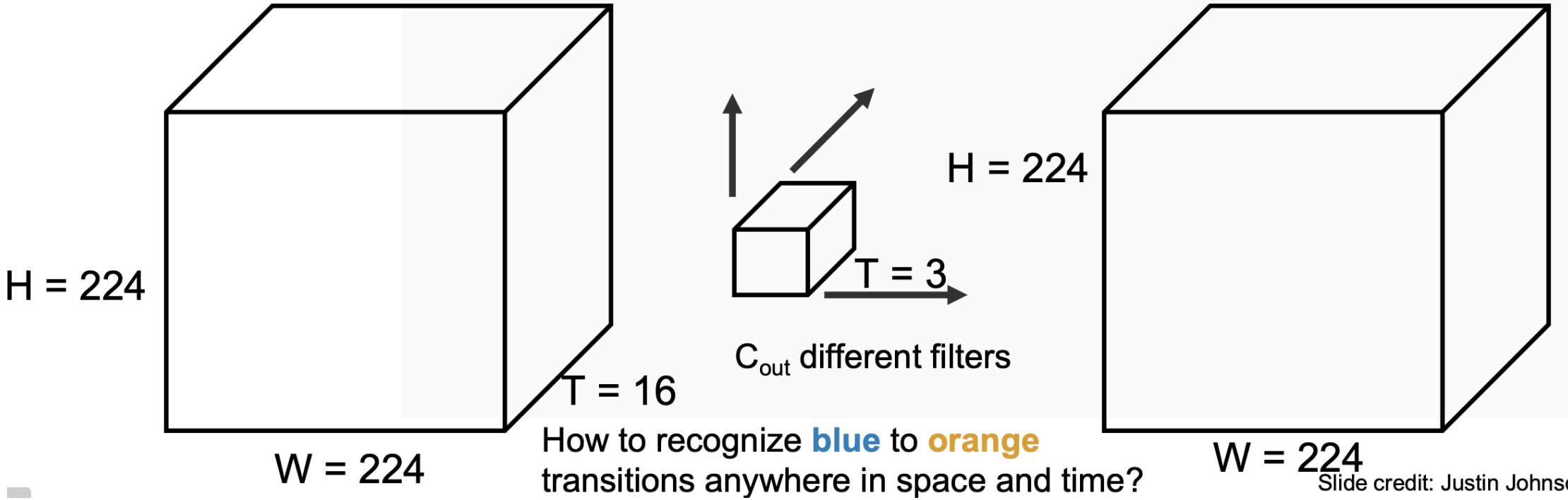
Slide credit: Justin Johnson

2D Conv (Early Fusion) vs 3D Conv (3D CNN)

Input: C_{in} x T x H x W
(3D grid with C_{in}-dim feat at each point)

Weight:
C_{out} x C_{in} x 3 x 3 x 3
Slide over x and y

Output:
C_{out} x T x H x W
3D grid with C_{out}–dim feat at each point



C_{out} different filters

How to recognize **blue** to **orange** transitions anywhere in space and time?

Slide credit: Justin Johnson

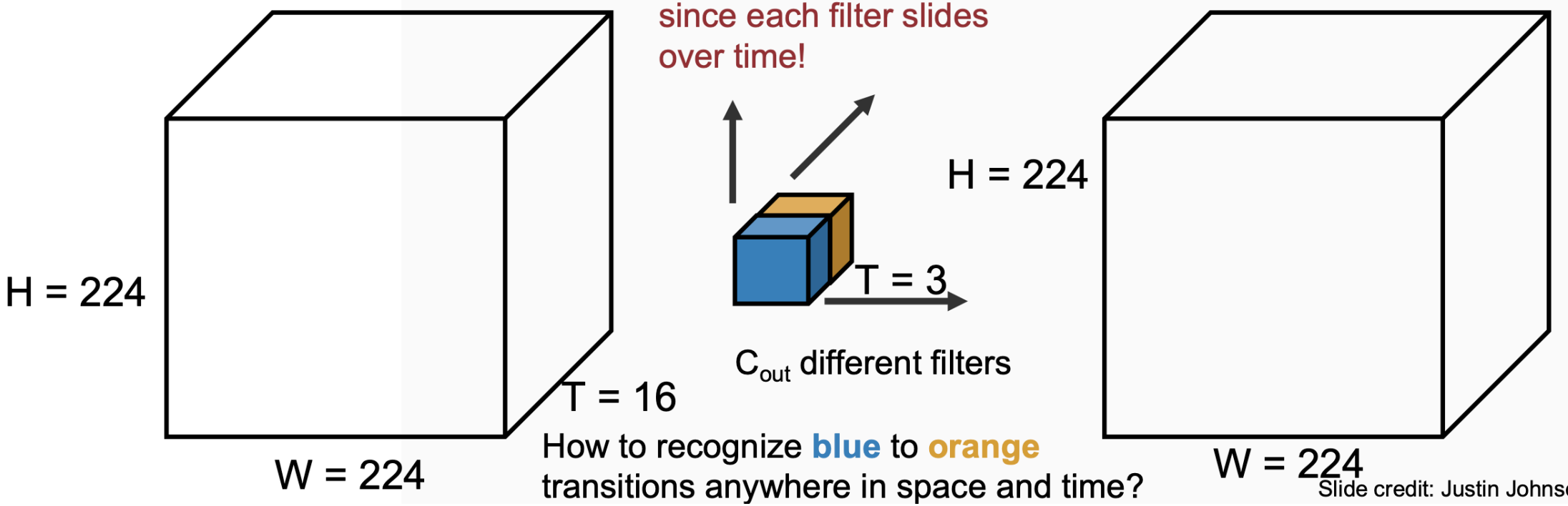
2D Conv (Early Fusion) vs 3D Conv (3D CNN)

Input: C_{in} x T x H x W
(3D grid with C_{in}-dim feat at each point)

Weight:
C_{out} x C_{in} x 3 x 3 x 3
Slide over x and y

Output:
C_{out} x T x H x W
3D grid with C_{out}–dim feat at each point

Temporal shift-invariant since each filter slides over time!



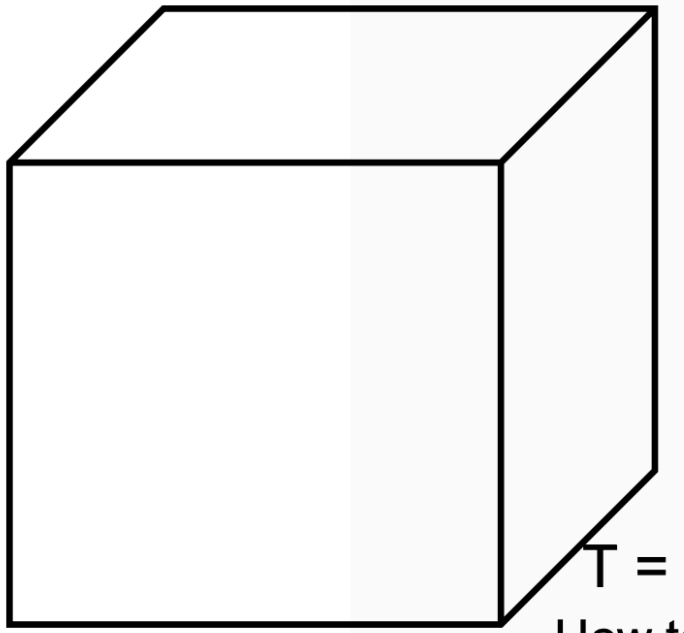
C_{out} different filters

How to recognize **blue** to **orange** transitions anywhere in space and time?

Slide credit: Justin Johnson

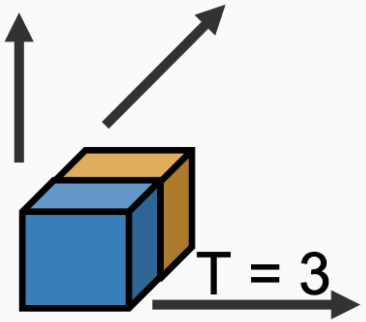
2D Conv (Early Fusion) vs 3D Conv (3D CNN)

Input: C_{in} x T x H x W
(3D grid with C_{in}-dim feat at each point)



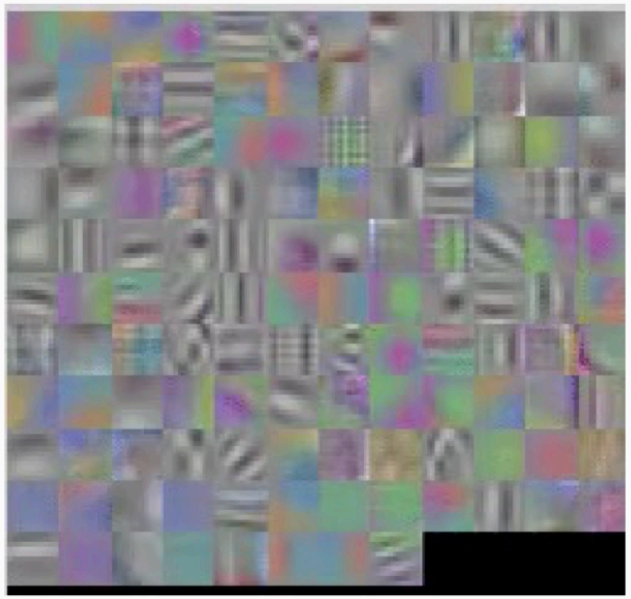
Weight:
C_{out} x C_{in} x 3 x 3 x 3
Slide over x and y

Temporal shift-invariant since each filter slides over time!



C_{out} different filters

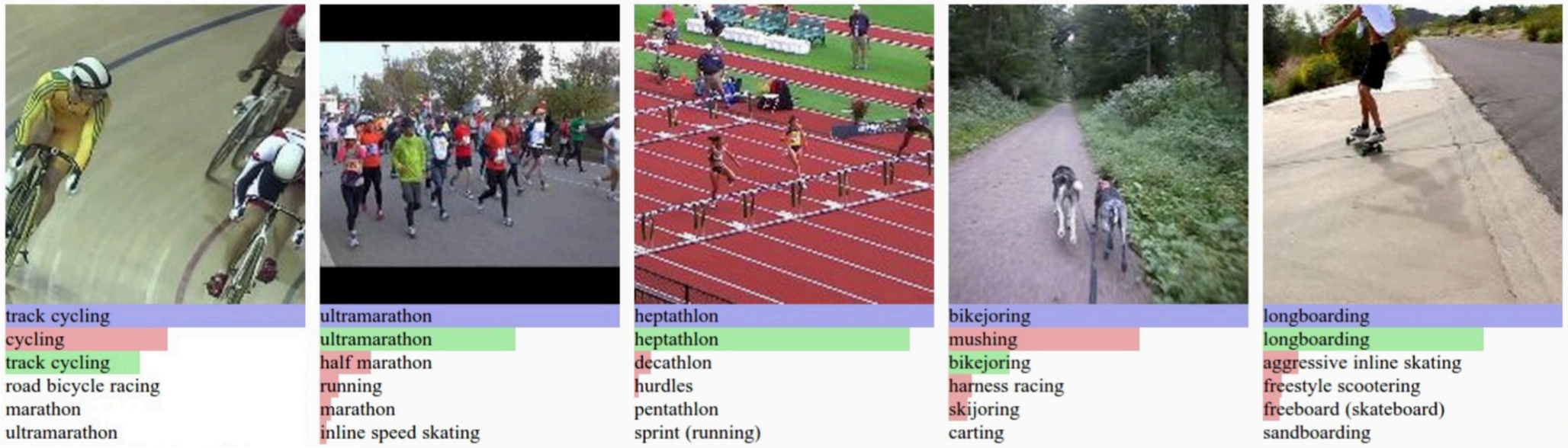
First-layer filters have shape 3 (RGB) x 4 (frames) x 5 x 5 (space)
Can visualize as video clips!



How to recognize **blue** to **orange** transitions anywhere in space and time?

Slide credit: Justin Johnson

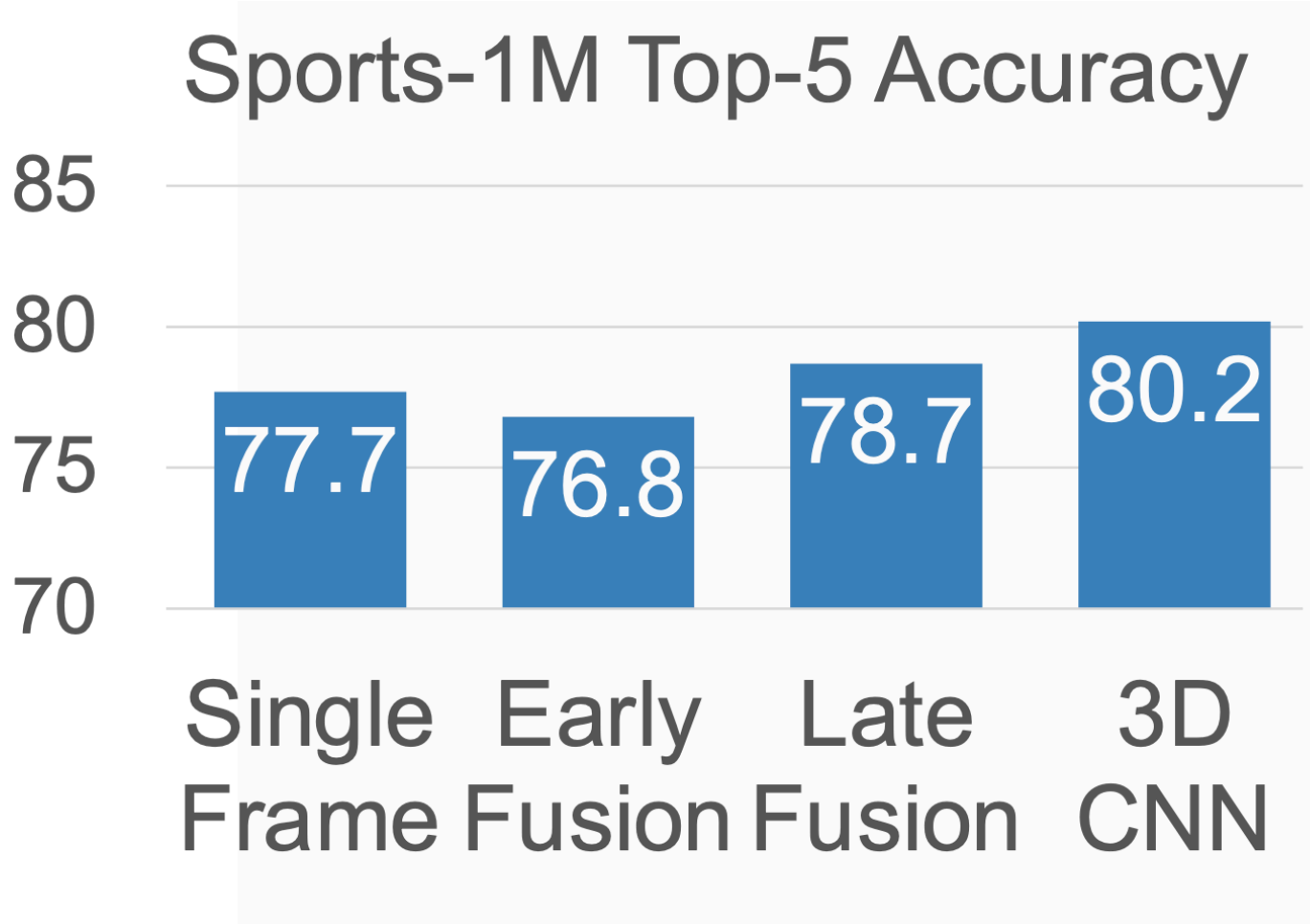
Example Video Dataset: Sports-1M



1 million YouTube videos annotated with labels for 487 different types of sports

Ground Truth
Correct prediction
Incorrect prediction

Early Fusion vs. Late Fusion vs. 3DCNN



Single Frame model works well – always try this first!

3D CNNs have improved a lot since 2014!

Introduction to Computer Vision

Next lecture: Lecture 11,
Temporal Analysis II