



Introduction to Computer Vision

Lecture 11 Sequential Modeling I

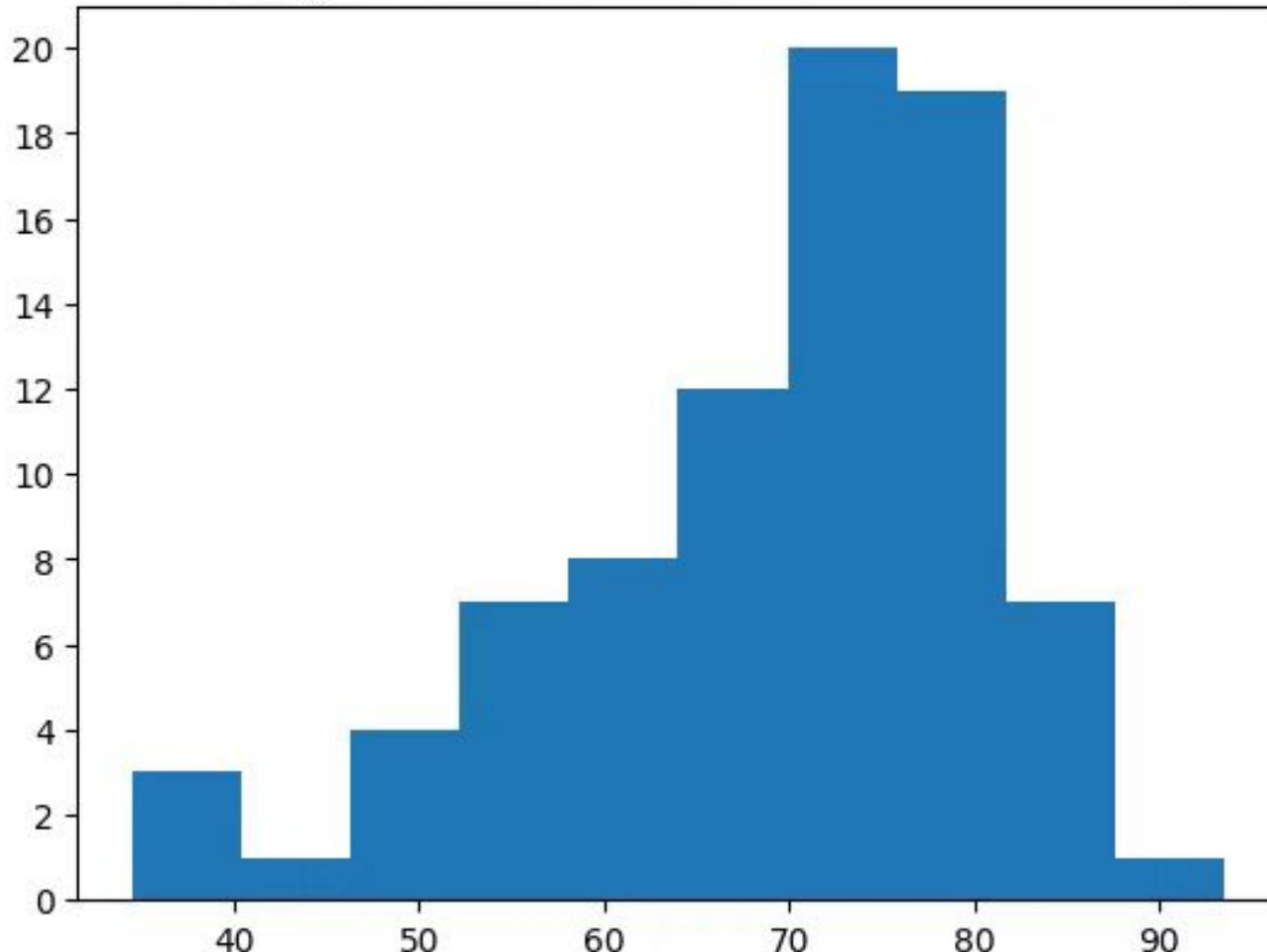
Prof. He Wang

Logistics

- Assignment 3:
 - due on 5/12 11:59PM (Sunday)
- If 1 day (0 - 24 hours) past the deadline, 15% off
- If 2 day (24 - 48 hours) past the deadline, 30% off
- Zero credit if more than 2 days.

Midterm

Mid origin median:71.000 mean:68.860 std:11.767

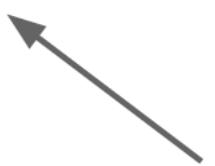
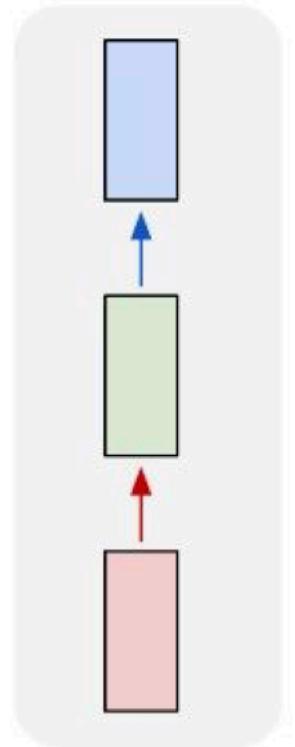


RNN

Some slides are borrowed from Stanford CS231N

Single-Frame Neural Network

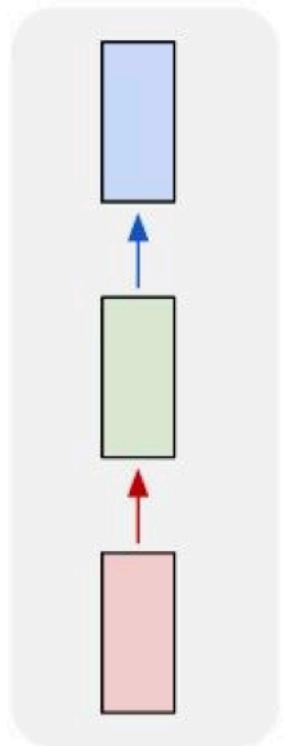
one to one



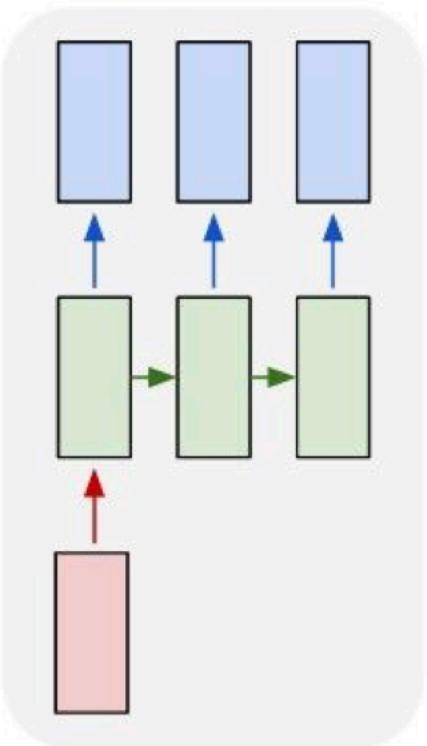
Vanilla Neural Networks

Recurrent Neural Network: Process Sequential Data

one to one



one to many



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

Image Captioning



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

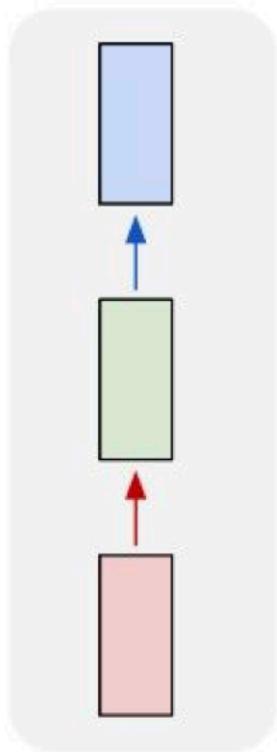


A man riding a dirt bike on a dirt track

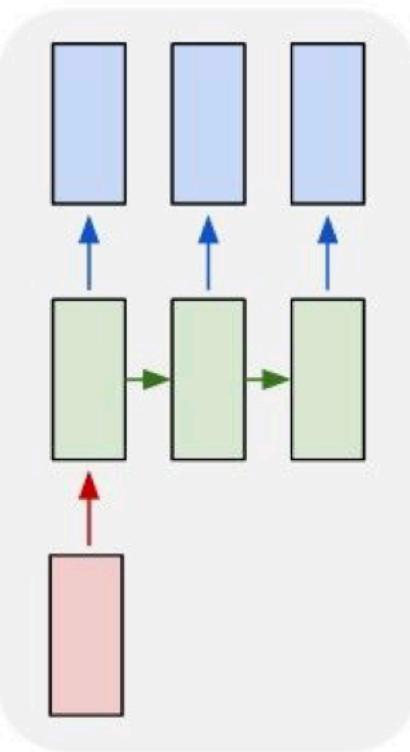
Captions generated using [neuraltalk2](#).

Recurrent Neural Network: Process Sequential Data

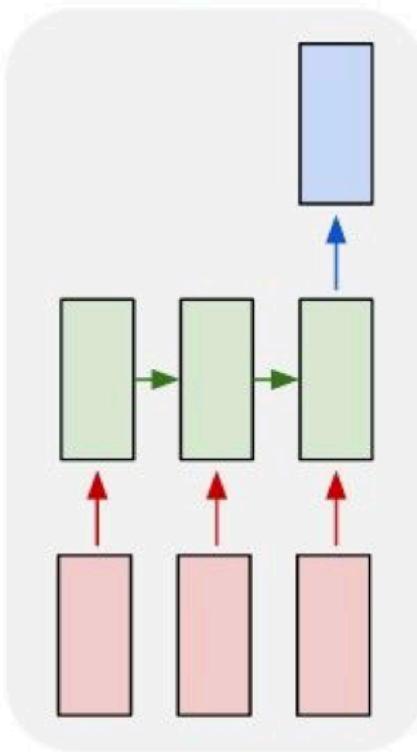
one to one



one to many



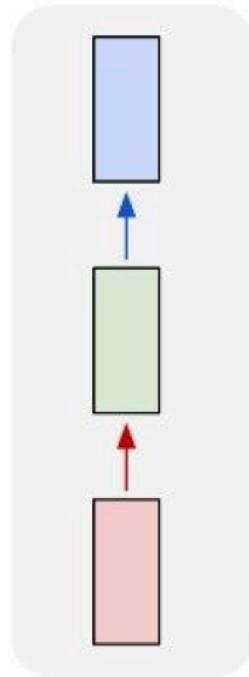
many to one



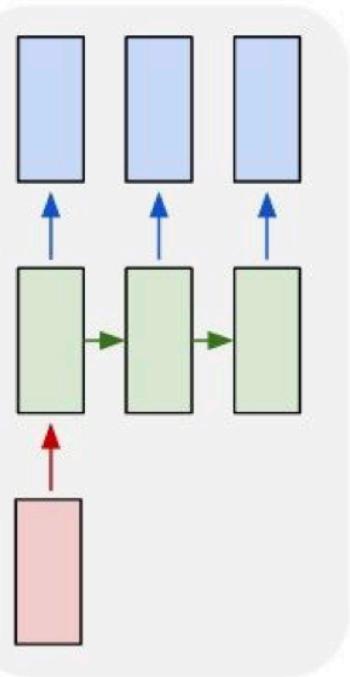
e.g. **action prediction**
sequence of video frames -> action class

Recurrent Neural Network: Process Sequential Data

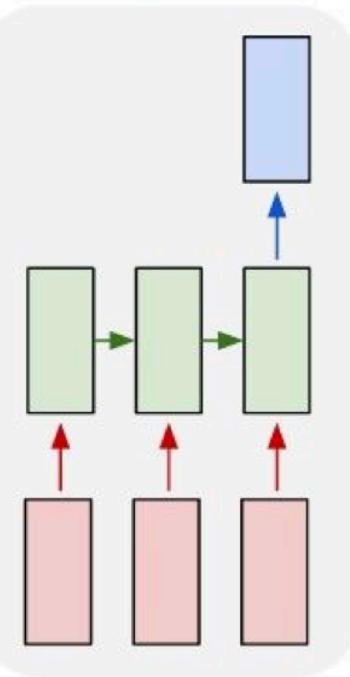
one to one



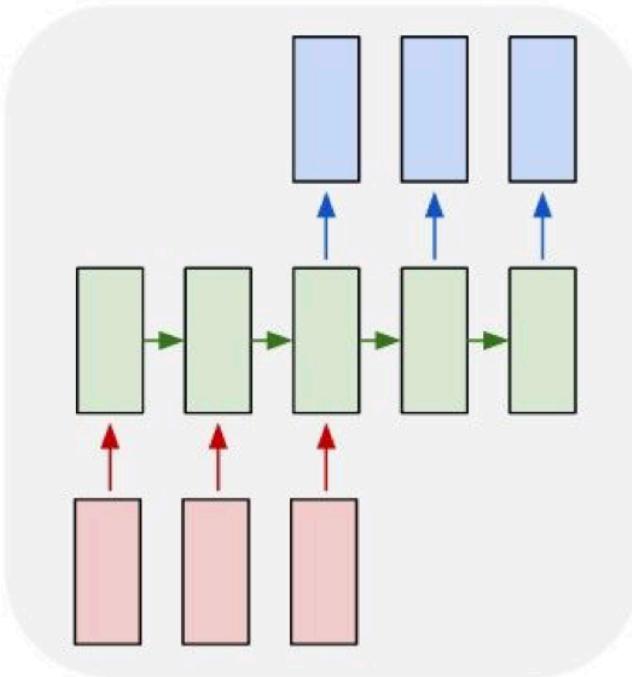
one to many



many to one

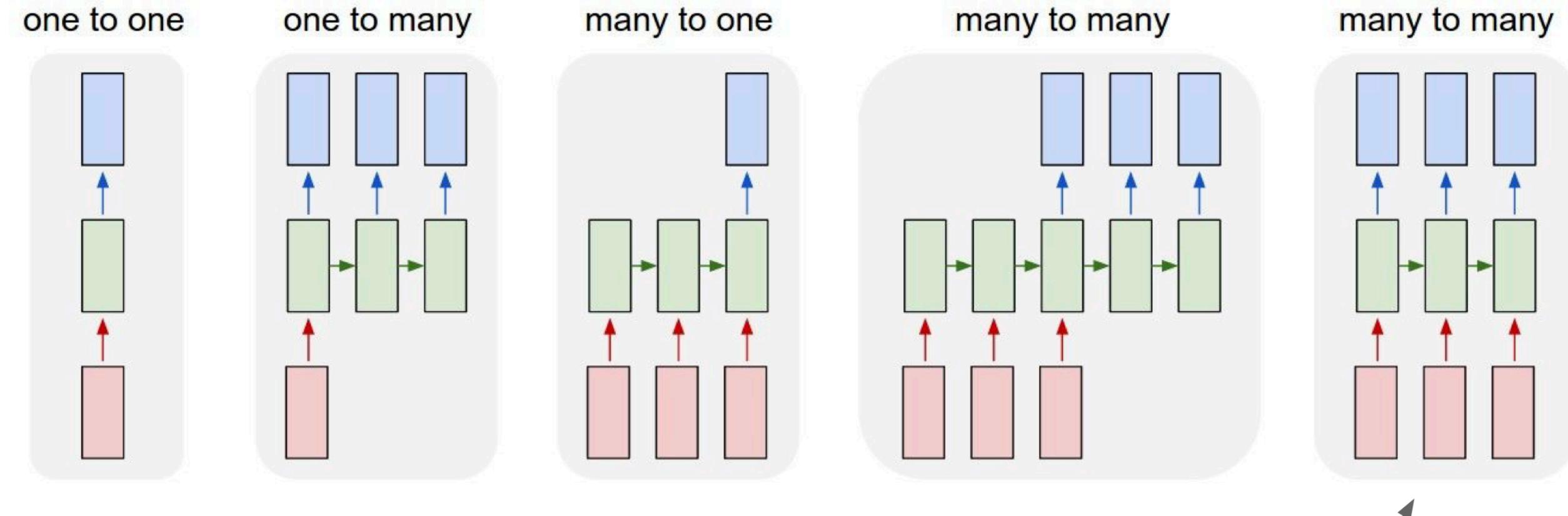


many to many



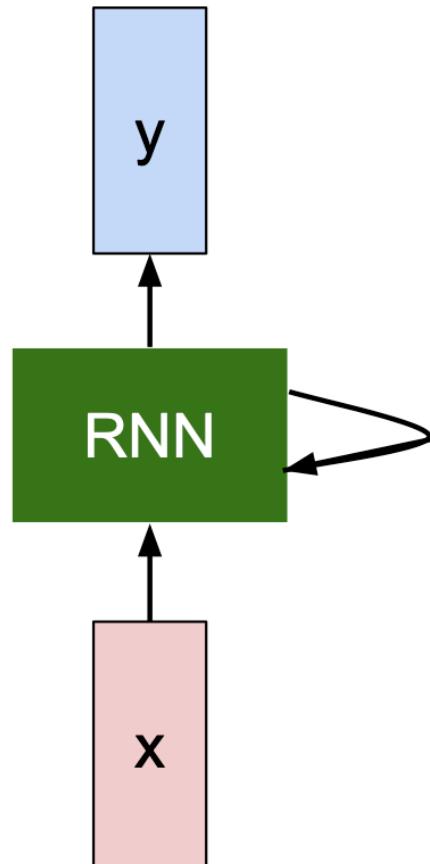
E.g. **Video Captioning**
Sequence of video frames -> caption

Recurrent Neural Network: Process Sequential Data

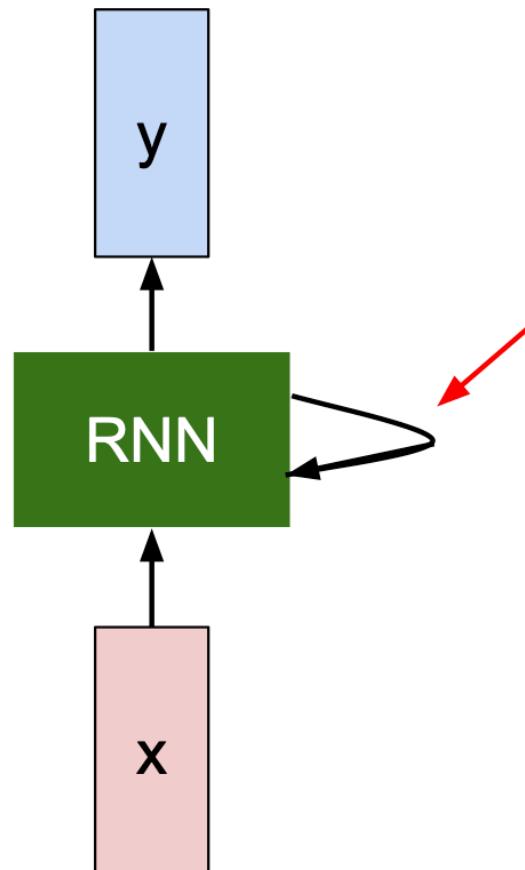


e.g. **Video classification on frame level**

Recurrent Neural Network

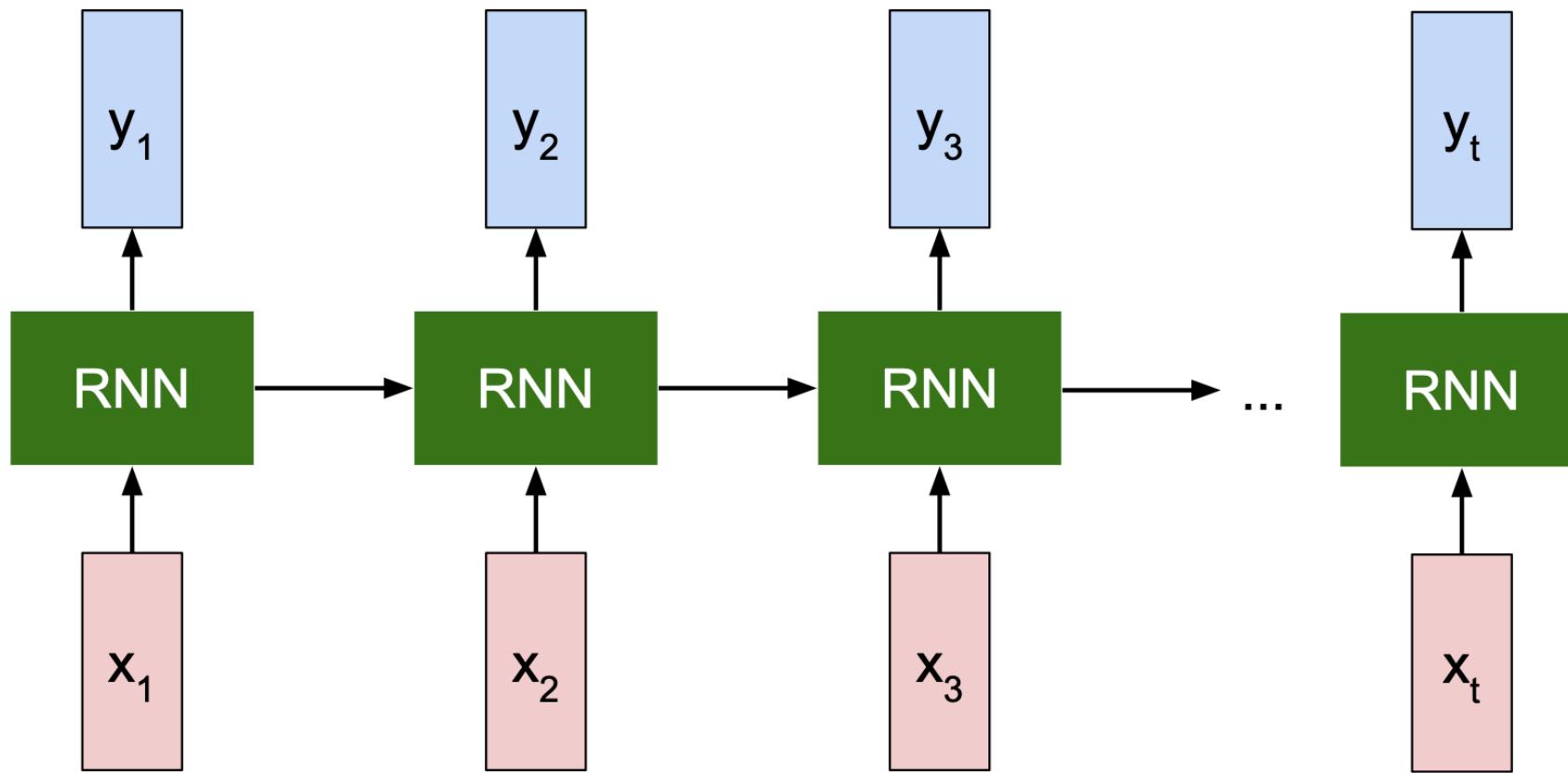


Recurrent Neural Network



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

Unrolled RNN

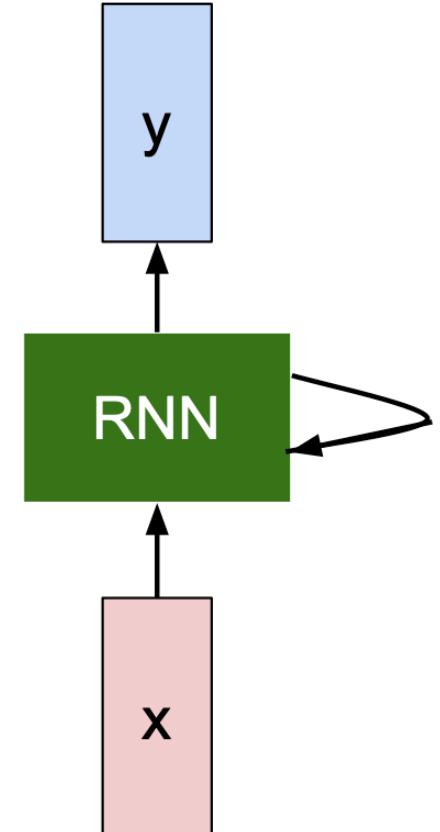


RNN Hidden State Update

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

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

new state old state input vector at
 / some time step
 some function
 with parameters W

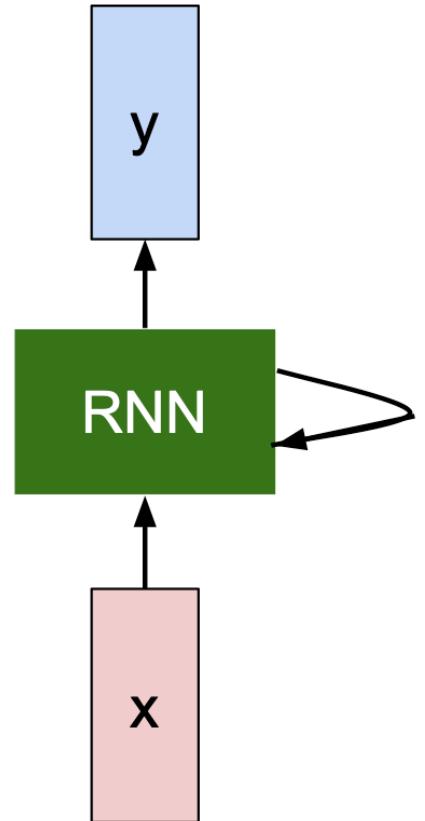


RNN Output

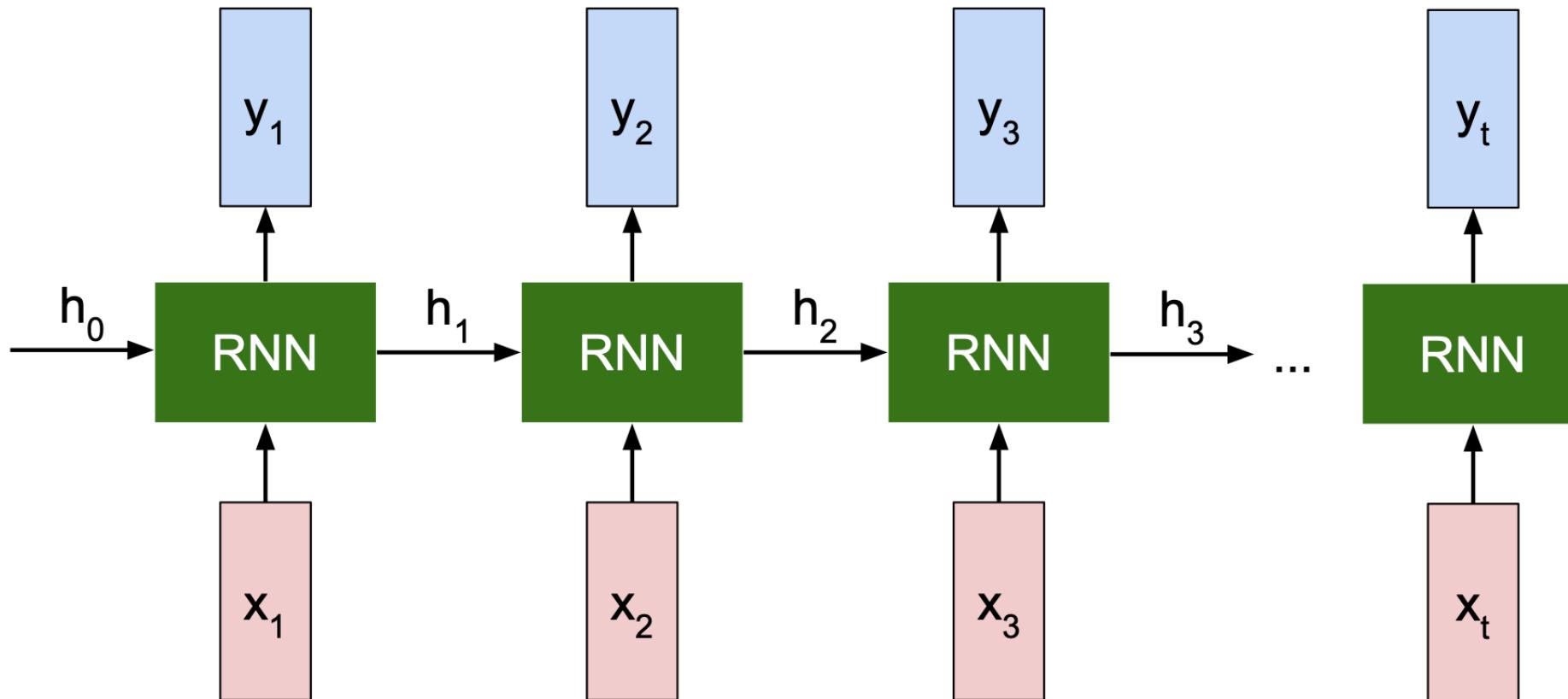
We can process a sequence of vectors x by applying a **recurrence formula** at every time step:

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

output new state
another function
with parameters W_o



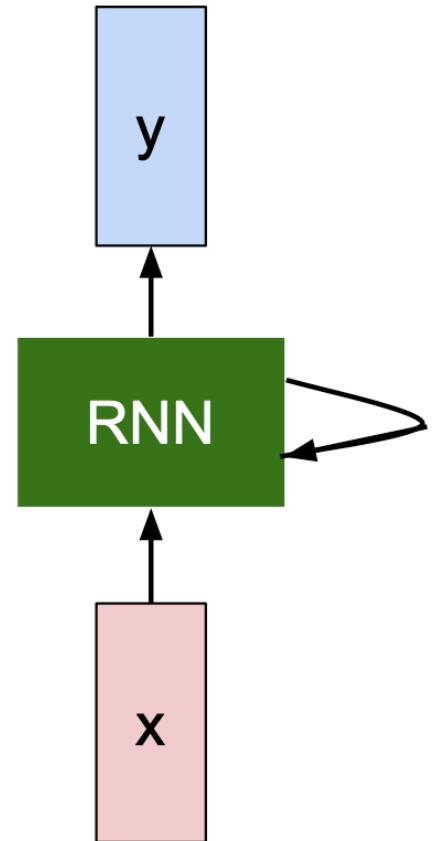
Recurrent Neural Network



Recurrent Neural Network

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

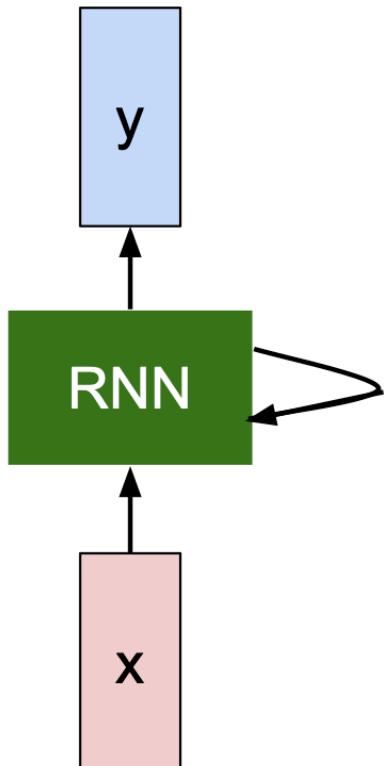
$$h_t = f_W(h_{t-1}, x_t)$$



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

Vanilla RNN

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



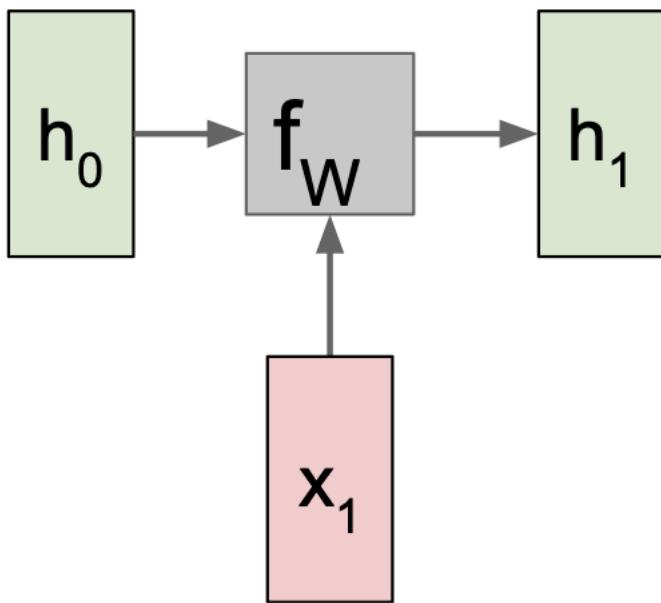
$$h_t = f_W(h_{t-1}, x_t)$$

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

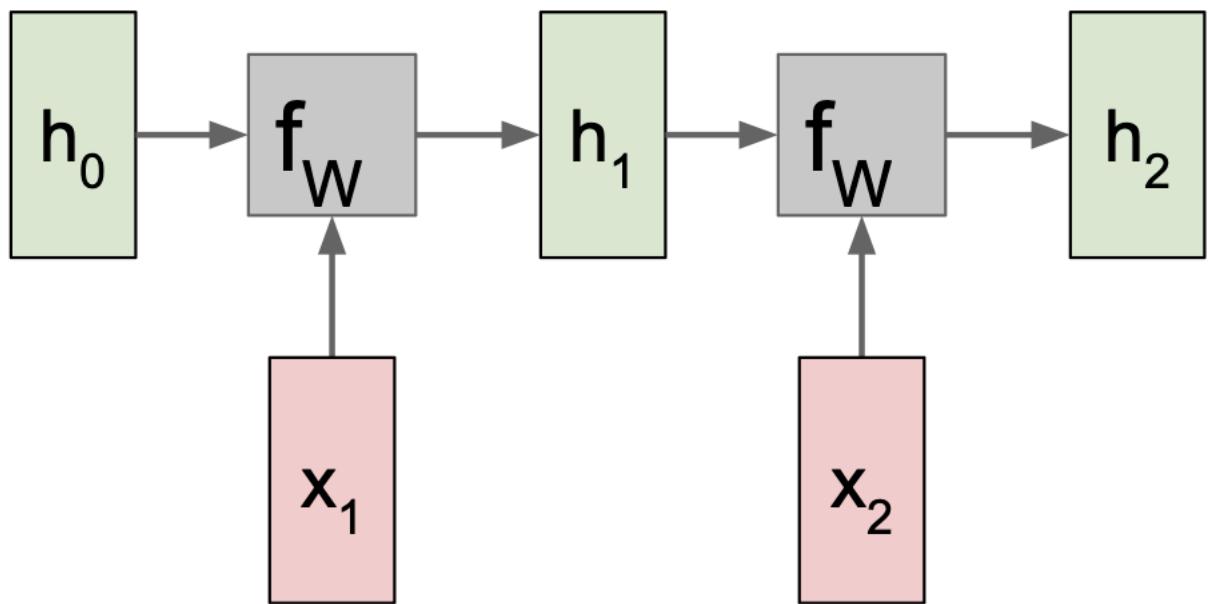
$$y_t = W_{hy}h_t$$

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

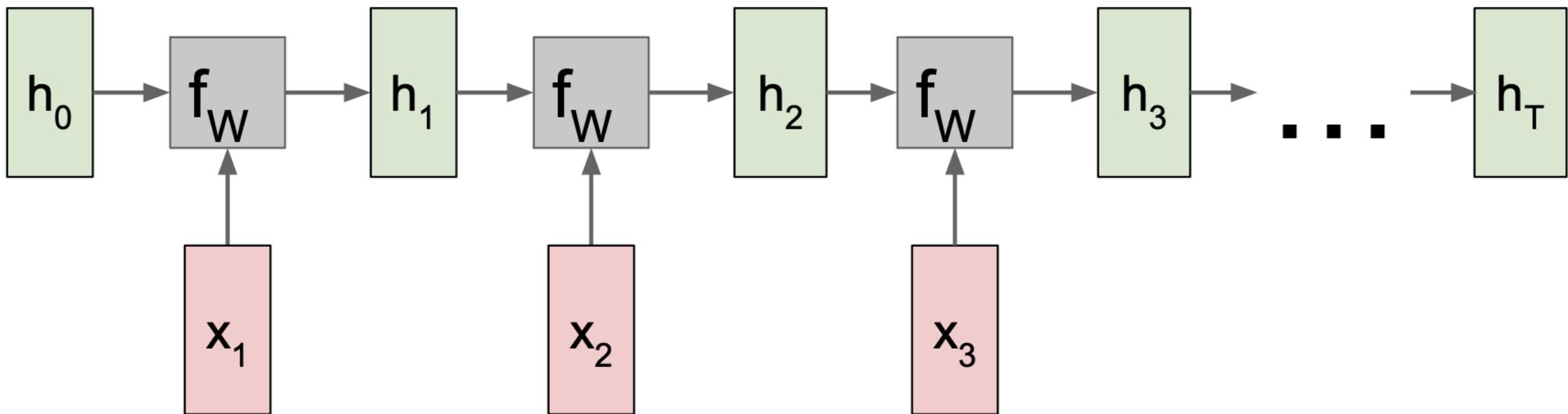
RNN: Computational Graph



RNN: Computational Graph

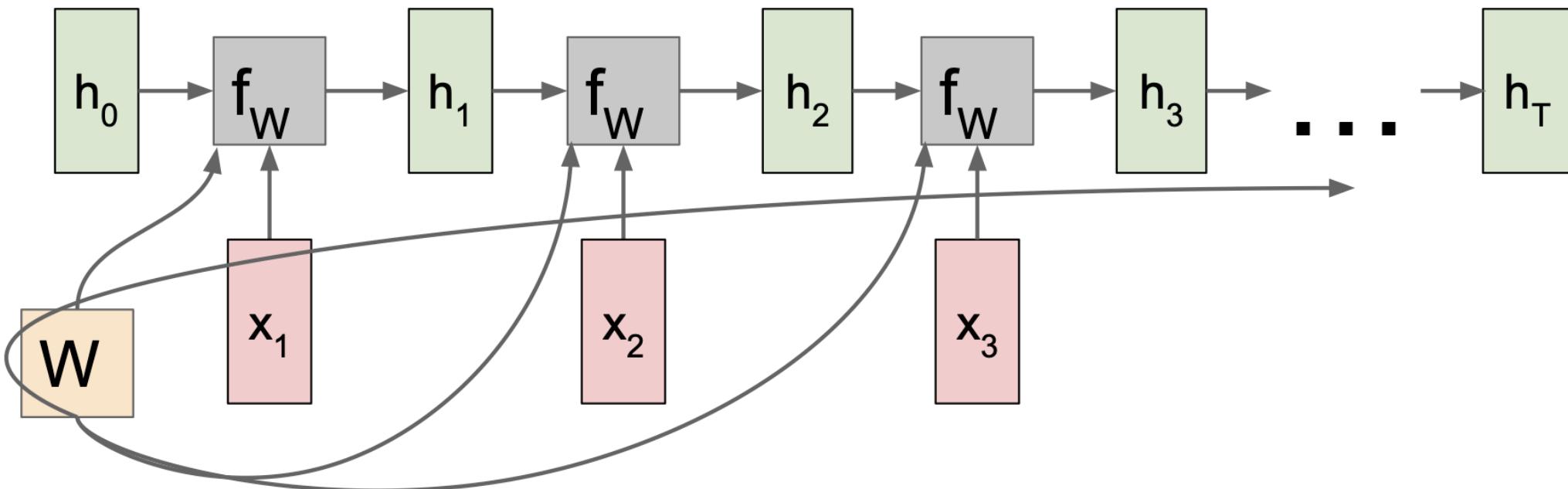


RNN: Computational Graph



RNN: Computational Graph

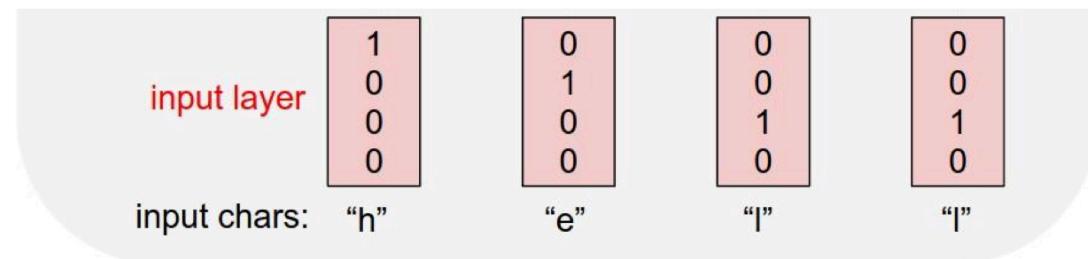
Re-use the same weight matrix at every time-step



Character-Level Language Model

Vocabulary:
[h,e,l,o]

Example training
sequence:
“hello”

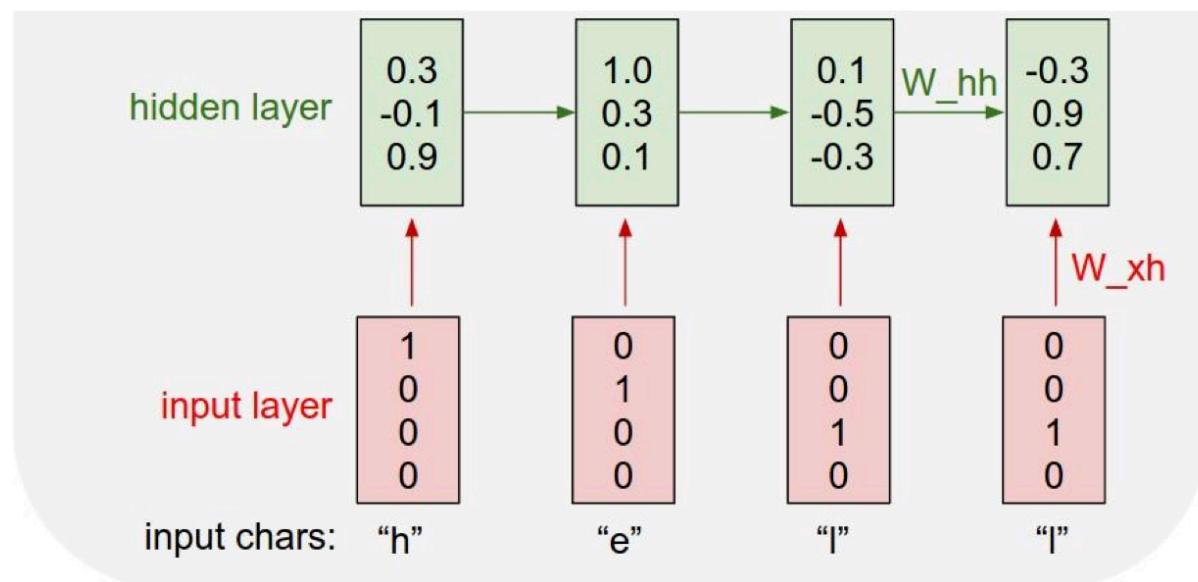


Character-Level Language Model

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

Vocabulary:
[h,e,l,o]

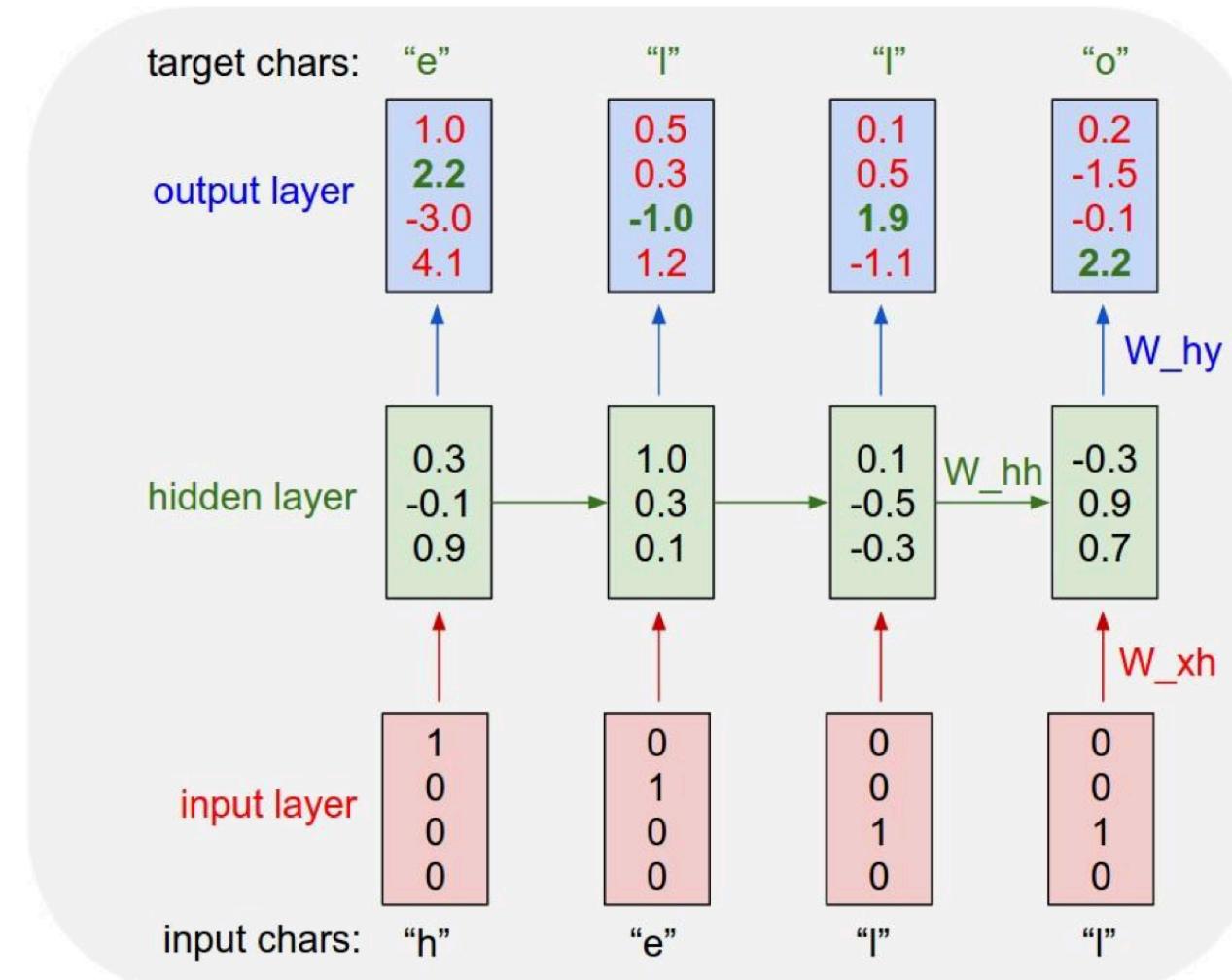
Example training
sequence:
“hello”



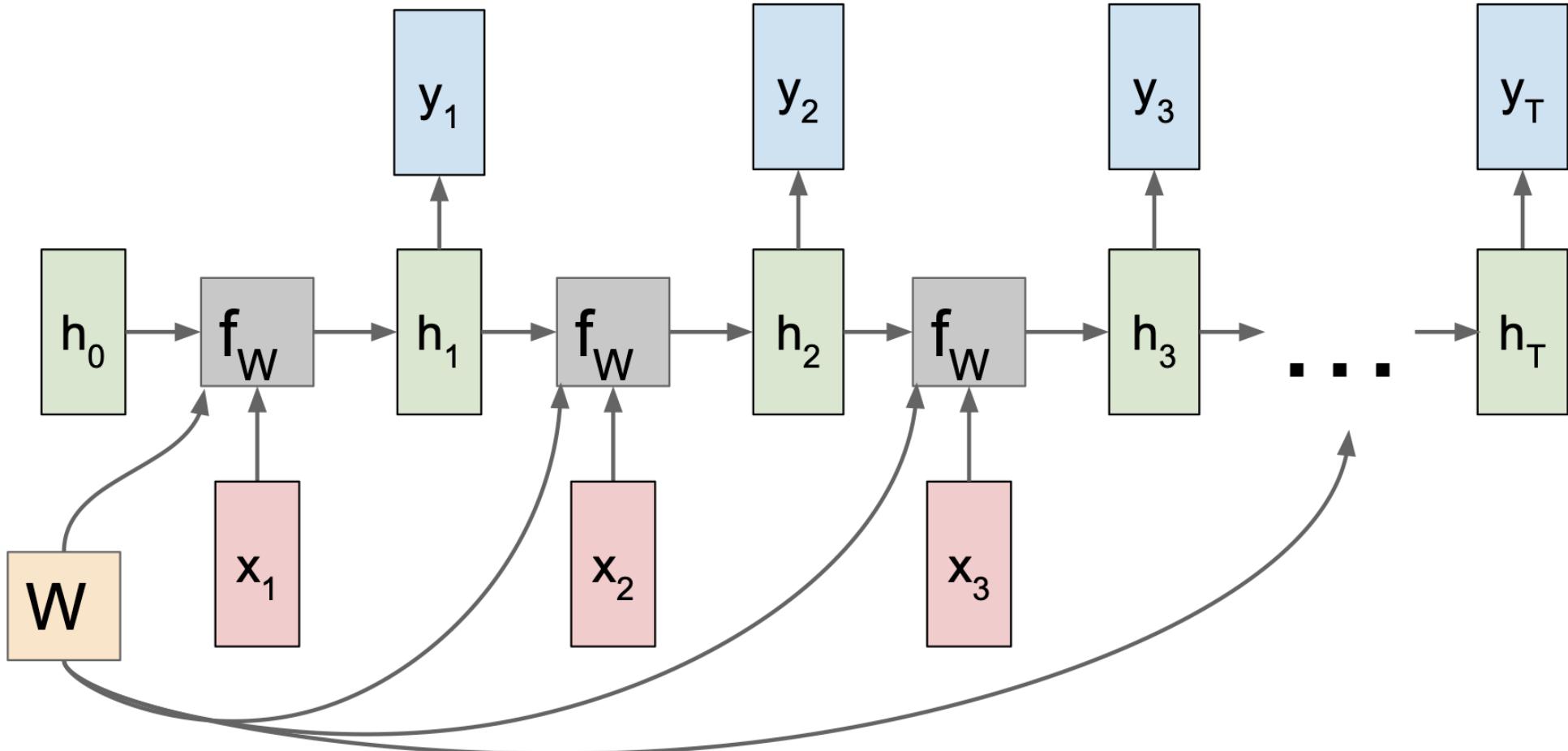
Character-Level Language Model

Vocabulary:
[h,e,l,o]

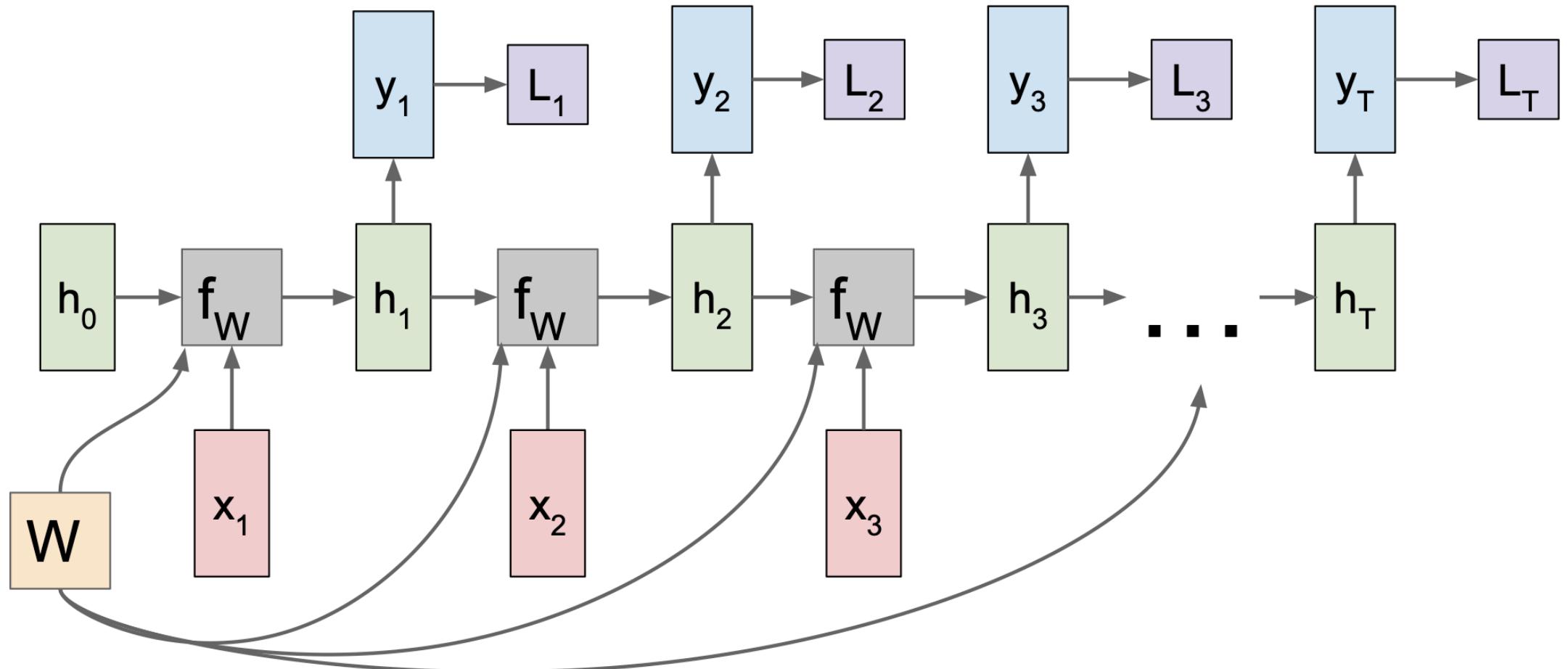
Example training
sequence:
“hello”



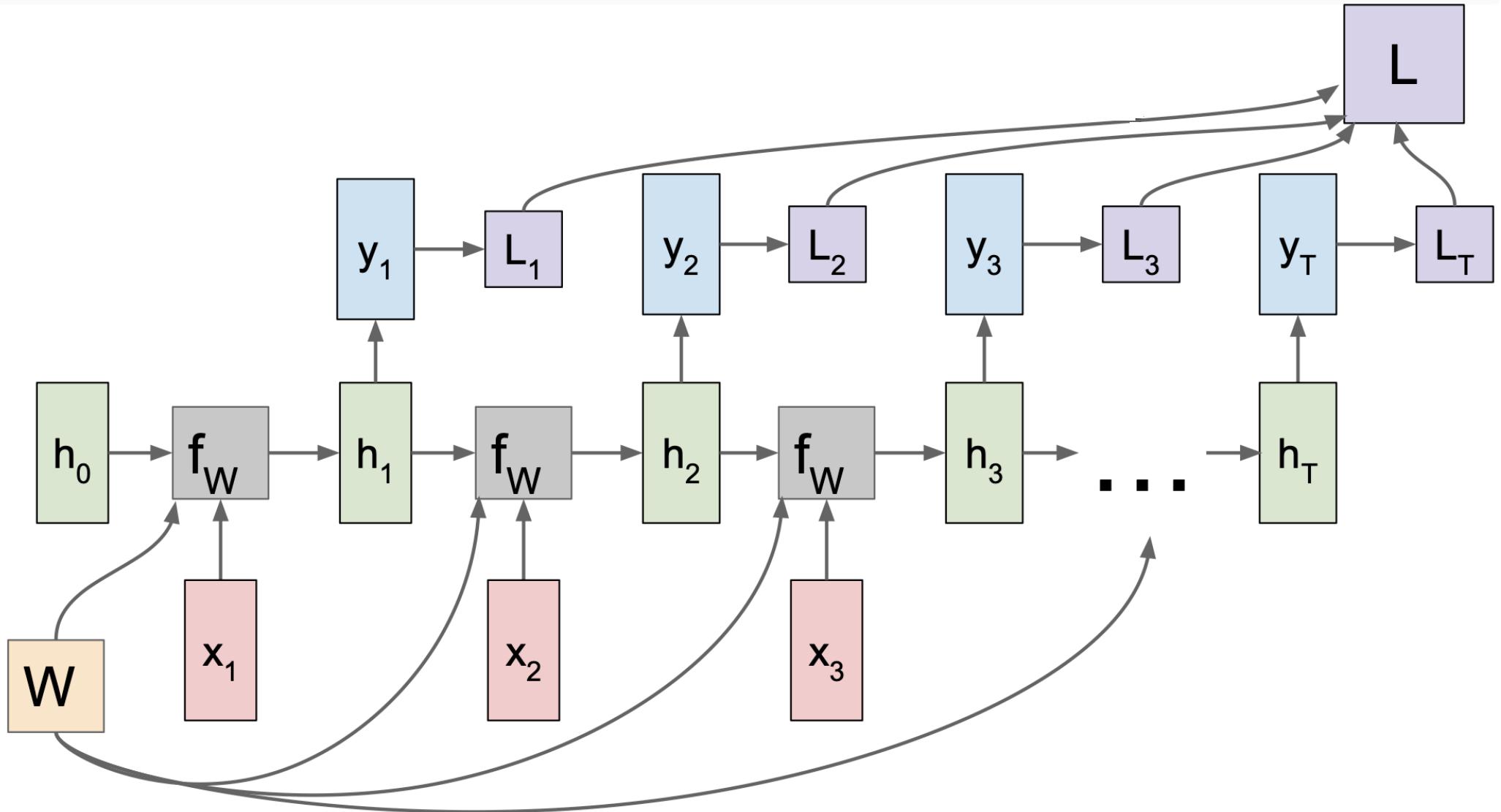
Many-to-Many RNN: Computational Graph



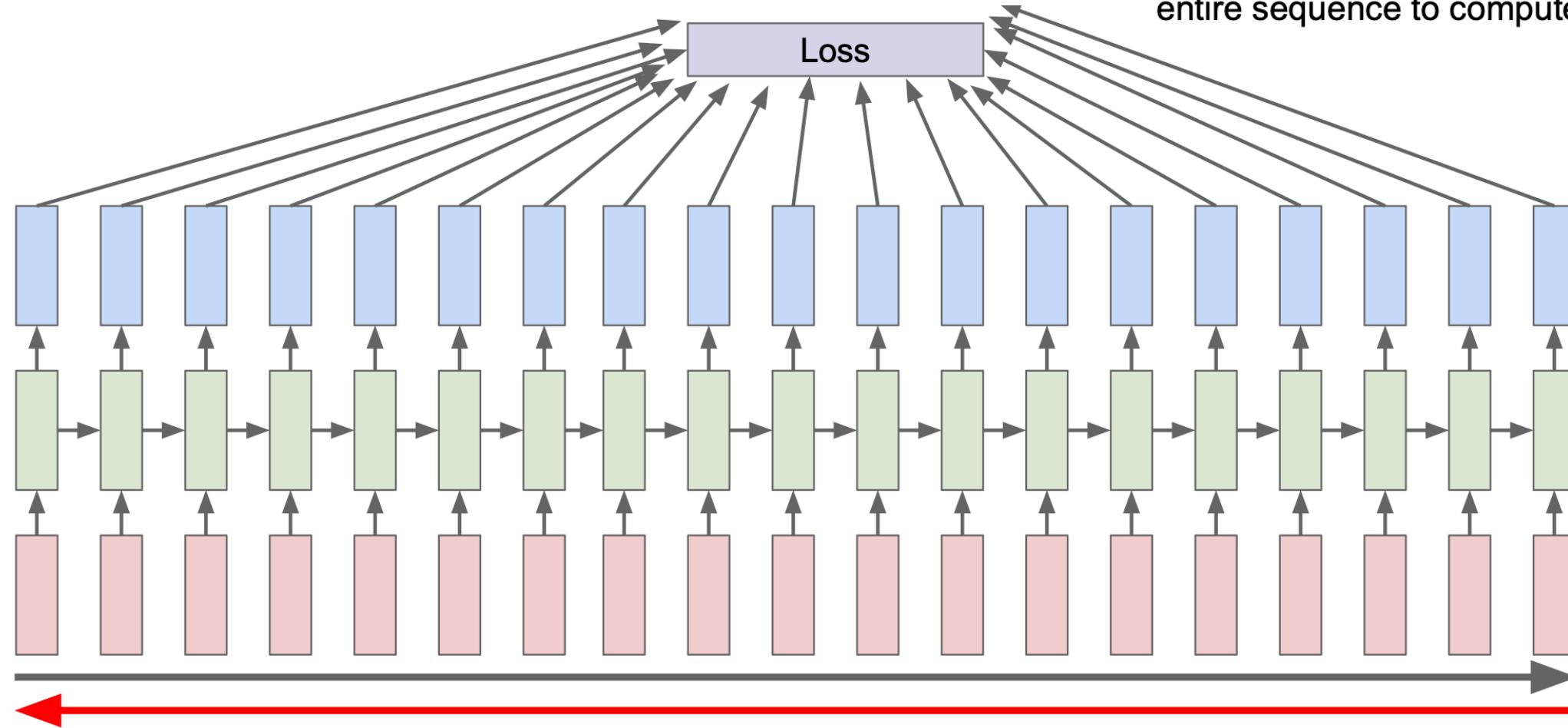
Many-to-Many RNN: Computational Graph



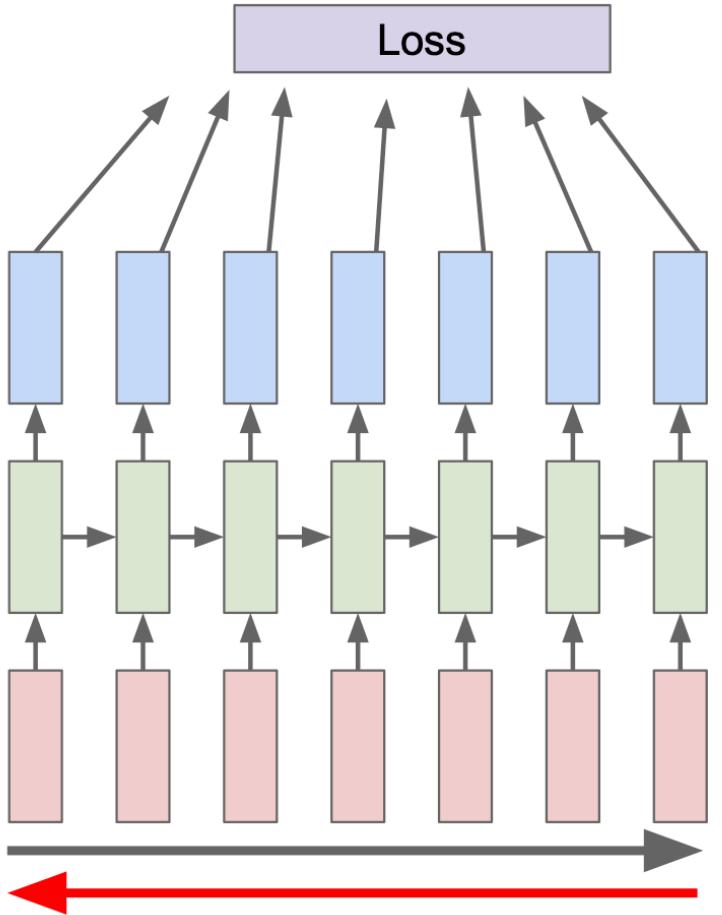
Many-to-Many RNN: Computational Graph



Backpropagation through Time

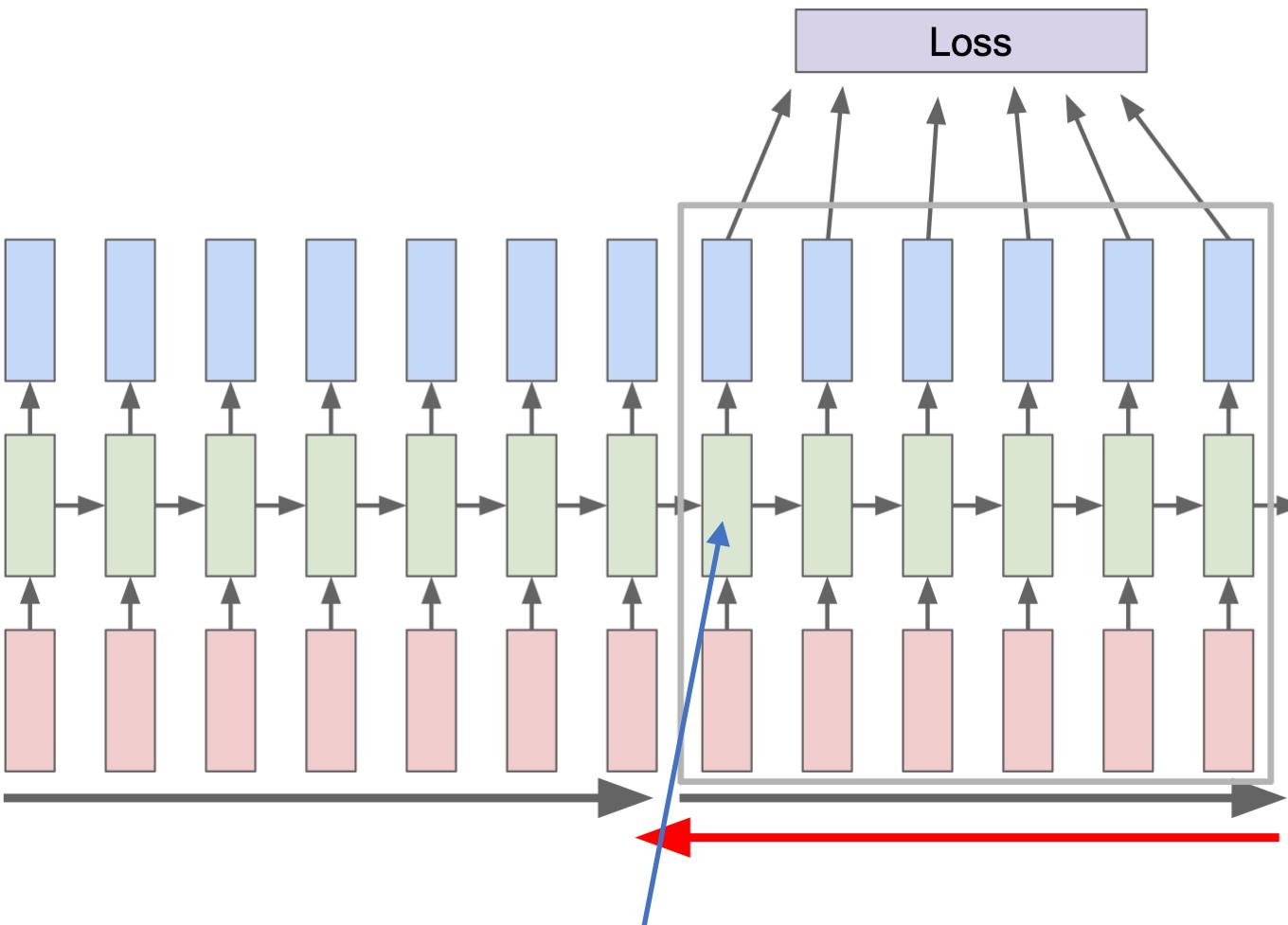


Truncated Backpropagation through Time



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

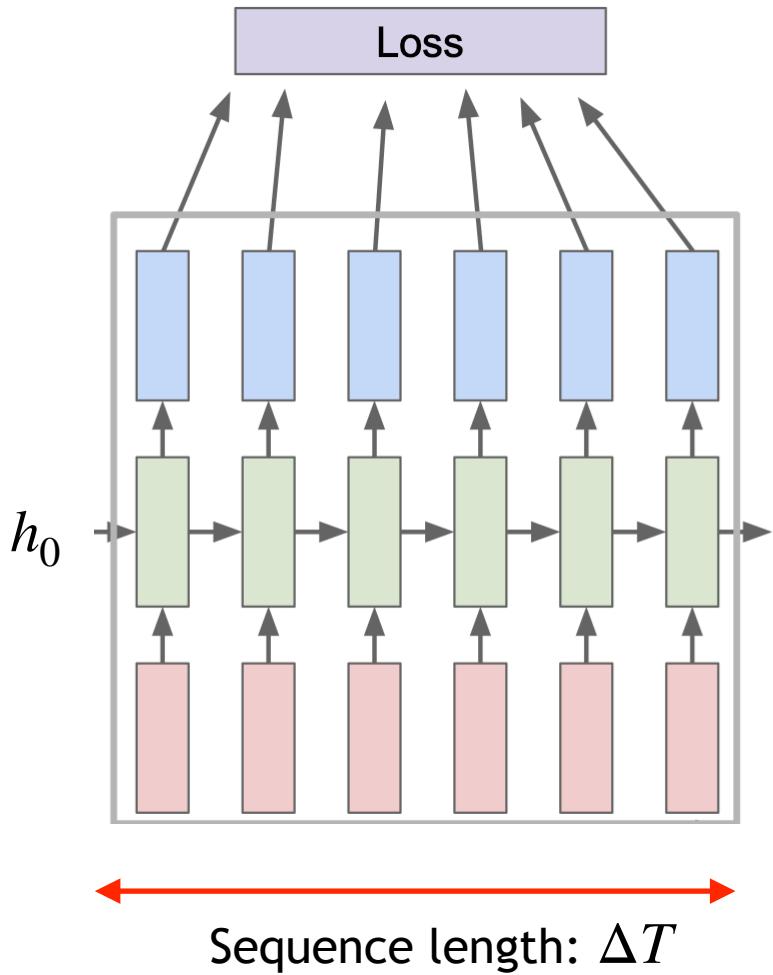
Truncated Backpropagation through Time



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**.

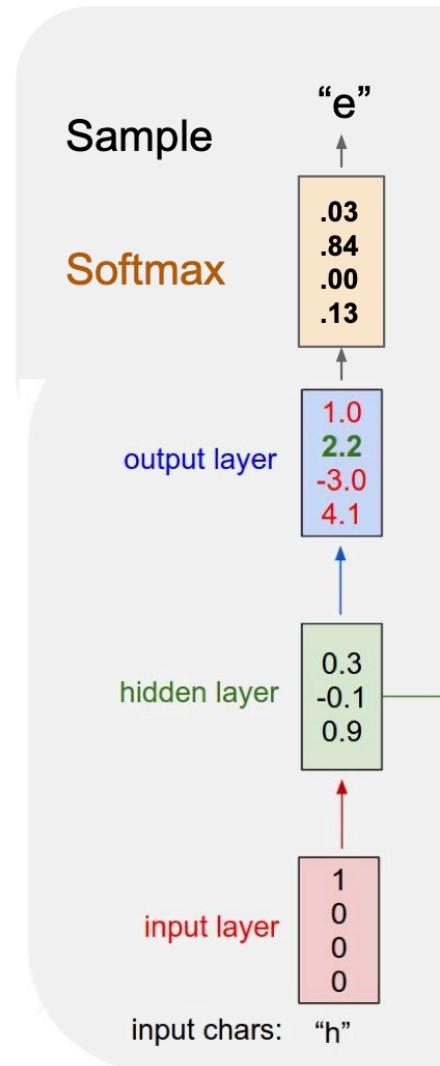
Long-term relationship longer than ΔT will probably not be learned.

Character-Level Language Model Sampling

Vocabulary:

[h,e,l,o]

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

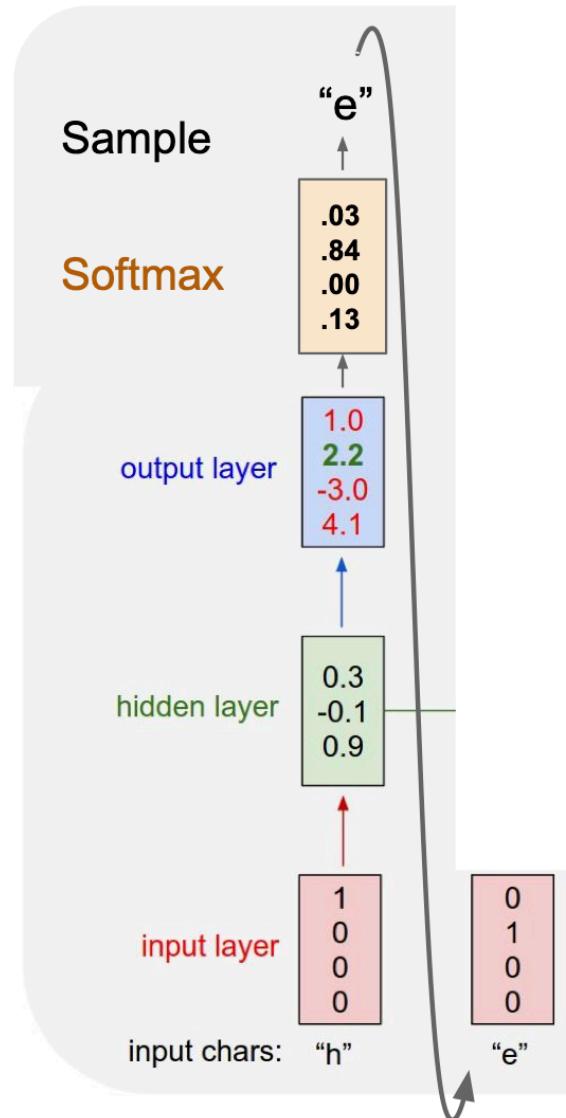


Character-Level Language Model Sampling

Vocabulary:

[h,e,l,o]

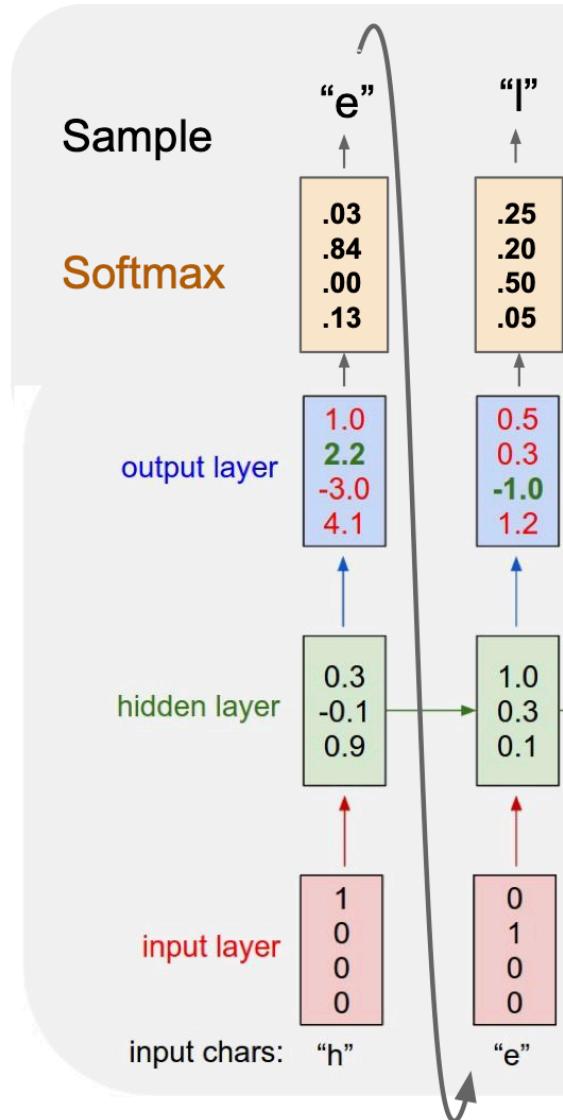
At test-time sample characters one at a time, feed back to model



Character-Level Language Model Sampling

Vocabulary:
[h,e,l,o]

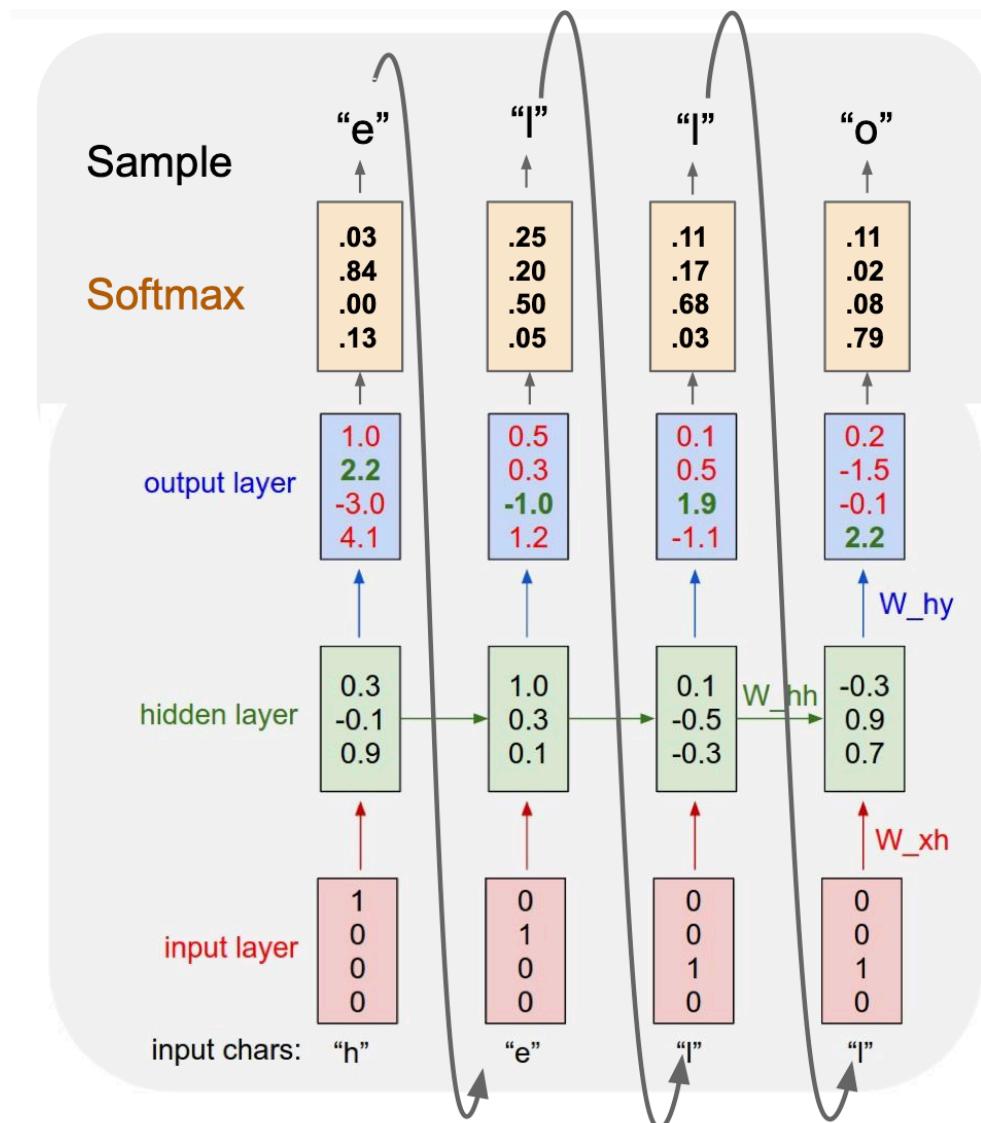
At test-time sample characters
one at a time, feed back to
model



Character-Level Language Model Sampling

Vocabulary:
[h,e,l,o]

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

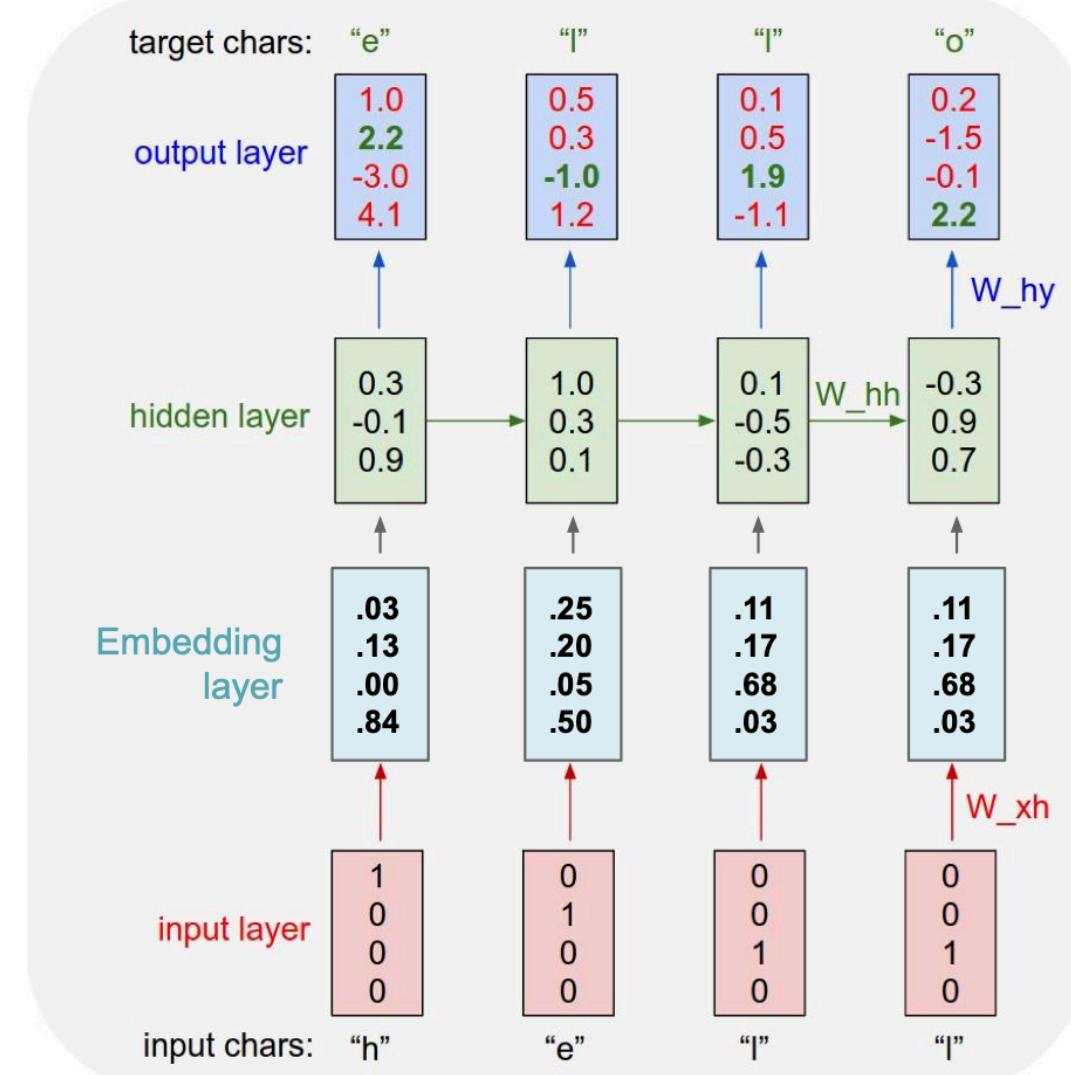


Character-Level Language Model Sampling

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

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

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

- 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



The Unreasonable Effectiveness of Recurrent Neural Networks

May 21, 2015

```
"""
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
wkh = 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(wkh, 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
        dwhx, dwhh, dwhy = np.zeros_like(wkh), 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
            dhy = np.dot(dy, hs[t].T)
            dby += dhy
            dy = np.dot(why.T, dy) + dhnext # backprop into h
            dhrw = (1 - hs[t] * hs[t]) * dy # backprop through tanh nonlinearity
            dbh += dhrw
            dwhx += np.dot(dhrw, xs[t].T)
            dwhh += np.dot(dhrw, hs[t-1].T)
            dhnext = np.dot(whh.T, dhrw)
            for dparam in [dwhx, dwhh, dwhy, dbh, dby]:
                np.clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients
            return loss, dwhx, dwhh, dwhy, dbh, dby, hs[len(inputs)-1]
```

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

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

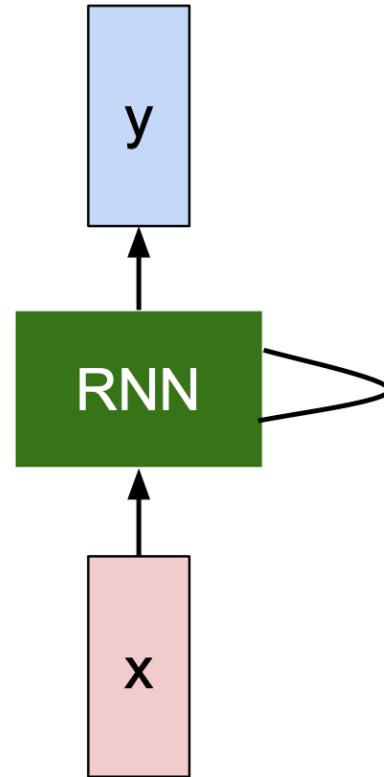
Character-Level Language Model: Shakespeare

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 deserved 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-iafhatawiaoahrdemot lytdws e ,tfti, astai f ogoh eoase rrranbyne 'nhthnee e
plia tkldrgd t o idoe ns,smtt h ne etie h,hregtrs nigtike,aoaenns lng

↓ train more

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

↓ train more

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

↓ 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 strain would be attain'd into being never fed,
And who is but a chain and subjects of his death,
I should not sleep.

Second Senator:

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

DUKE VINCENTIO:

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

Second Lord:

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

Clown:

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

VIOLA:

I'll drink it.

VIOLA:

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

KING LEAR:

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

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 $\bigoplus_{n=1,\dots,m}$ 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 $U \rightarrow T$ is a separated algebraic space.

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

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

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

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

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

is a unique morphism of algebraic stacks. Note that

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

and

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

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. \square

The result for prove any open covering follows from the less of Example ???. It may replace S by $X_{\text{spaces},\text{etale}}$ 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 $\underline{\text{Proj}}_X(\mathcal{A}) = \text{Spec}(B)$ over U compatible with the complex

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

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

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

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

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 $i_0 \leq p$ is a subset of $\mathcal{J}_{n,0} \circ \overline{A}_2$ works.

Lemma 0.3. In Situation ???. Hence we may assume $q' = 0$.

Proof. We will use the property we see that p is the next 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 . \square

Word-Level Language Model: Math Book (Latex)

Proof. Omitted. \square

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 $X_{\text{étale}}$ we have

$$\mathcal{O}_X(\mathcal{F}) = \{\text{morph}_1 \times_{\mathcal{O}_X} (\mathcal{G}, \mathcal{F})\}$$

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

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

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

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. \square

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

$$\begin{array}{ccccc}
 S & \longrightarrow & & & \\
 \downarrow & & & & \\
 \xi & \longrightarrow & \mathcal{O}_{X'} & \nearrow & \\
 \text{gor}_s & & \uparrow & & \\
 & & = \alpha' \longrightarrow & & X \\
 & & \downarrow & & \downarrow \\
 \text{Spec}(K_\psi) & & \text{Mor}_{\text{Sets}} & & \text{d}(\mathcal{O}_{X/k}, \mathcal{G})
 \end{array}$$

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.

\square

Proof. We have see that $X = \text{Spec}(R)$ 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 . \square

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 C . The functor \mathcal{F} is a “field”

$$\mathcal{O}_{X,x} \rightarrow \mathcal{F}_{\bar{x}} \rightarrow (\mathcal{O}_{X_{\text{étale}}})^{-1}(\mathcal{O}_{X_{\bar{x}}}^{-1}\mathcal{O}_{X_{\bar{x}}}(\mathcal{O}_{X_{\bar{x}}}))$$

is an isomorphism of covering of $\mathcal{O}_{X_{\bar{x}}}$. 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. \square

If \mathcal{F} is a finite direct sum \mathcal{O}_{X_λ} 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 & 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

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

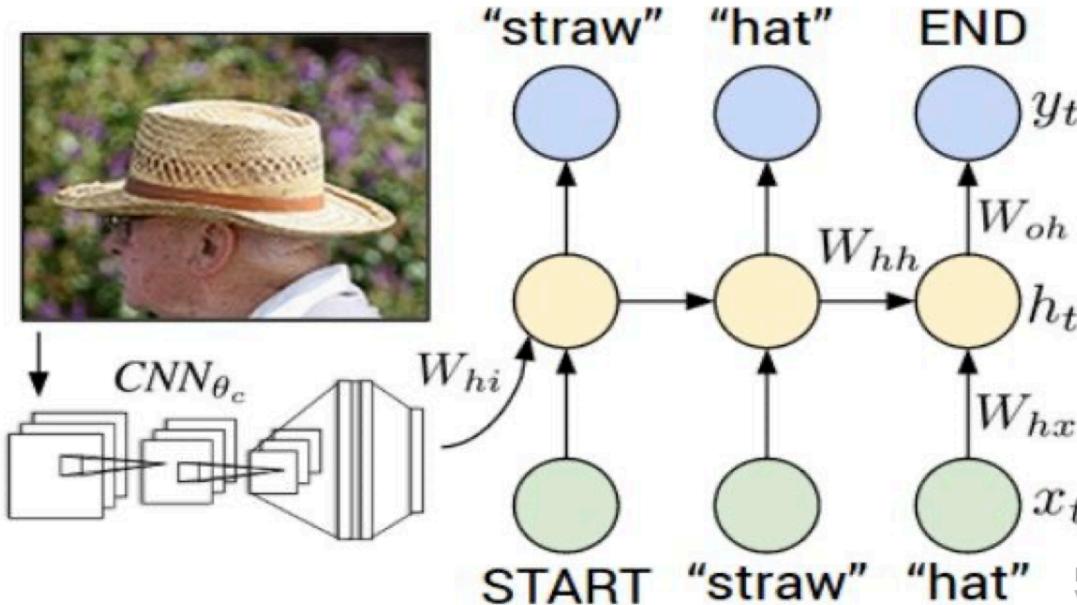


Figure from Karpathy et al, "Deep Visual-Semantic Alignments for Generating Image Descriptions", CVPR 2015; figure copyright IEEE, 2015.
Reproduced for educational purposes.

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

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

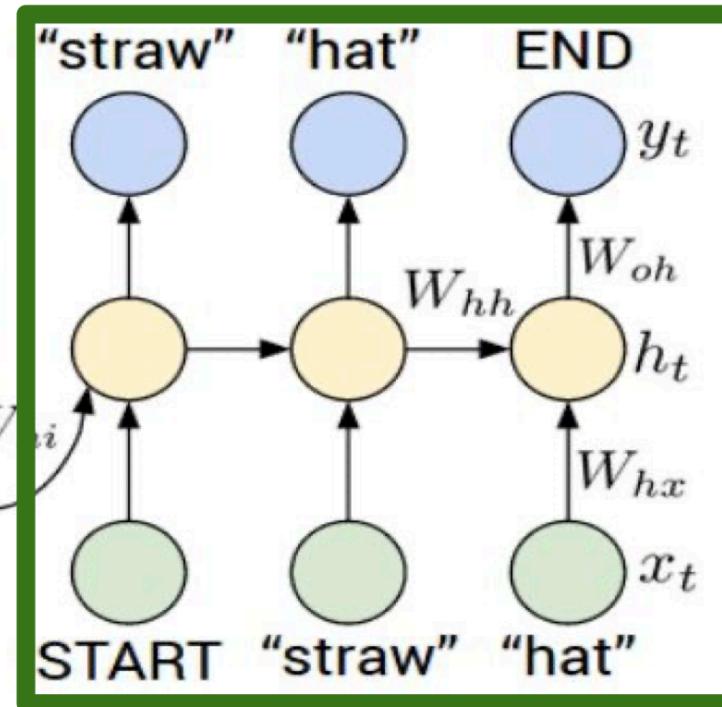
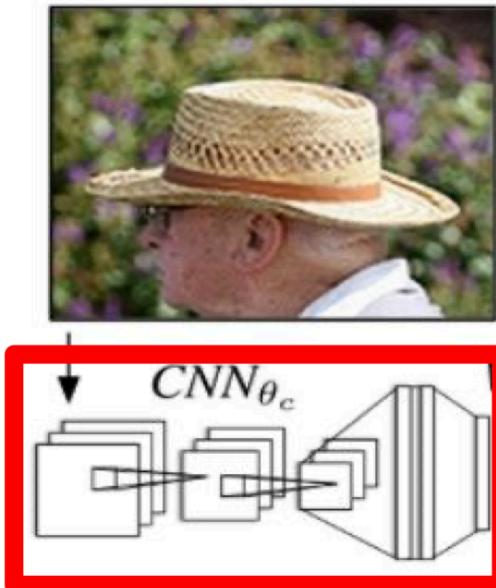
Show and Tell: A Neural Image Caption Generator, Vinyals et al.

Long-term Recurrent Convolutional Networks for Visual Recognition and Description, Donahue et al.

Learning a Recurrent Visual Representation for Image Caption Generation, Chen and Zitnick

Image Captioning

Recurrent Neural Network



Convolutional Neural Network

Image Captioning: Example Results



A cat sitting on a suitcase on the floor



A cat is sitting on a tree branch



A dog is running in the grass with a frisbee



A white teddy bear sitting in the grass



Two people walking on the beach with surfboards



A tennis player in action on the court



Two giraffes standing in a grassy field



A man riding a dirt bike on a dirt track

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



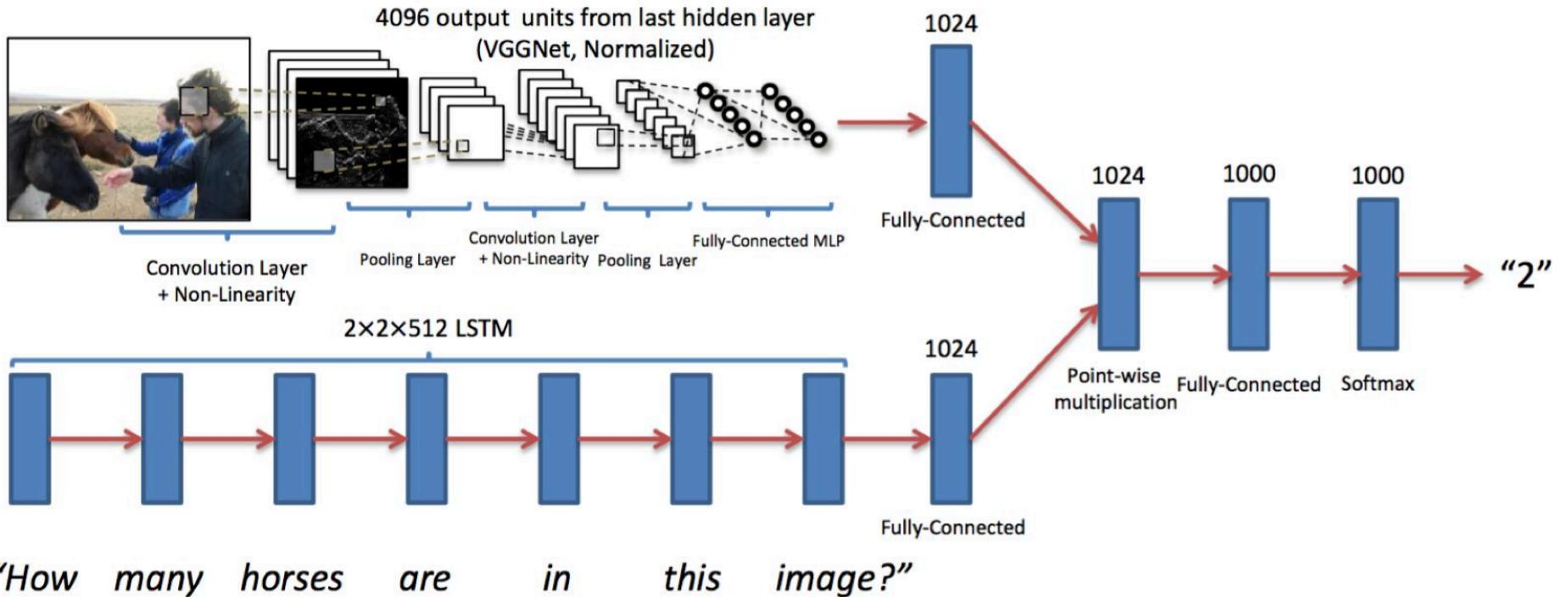
A bird is perched on a tree branch



A man in a baseball uniform throwing a ball

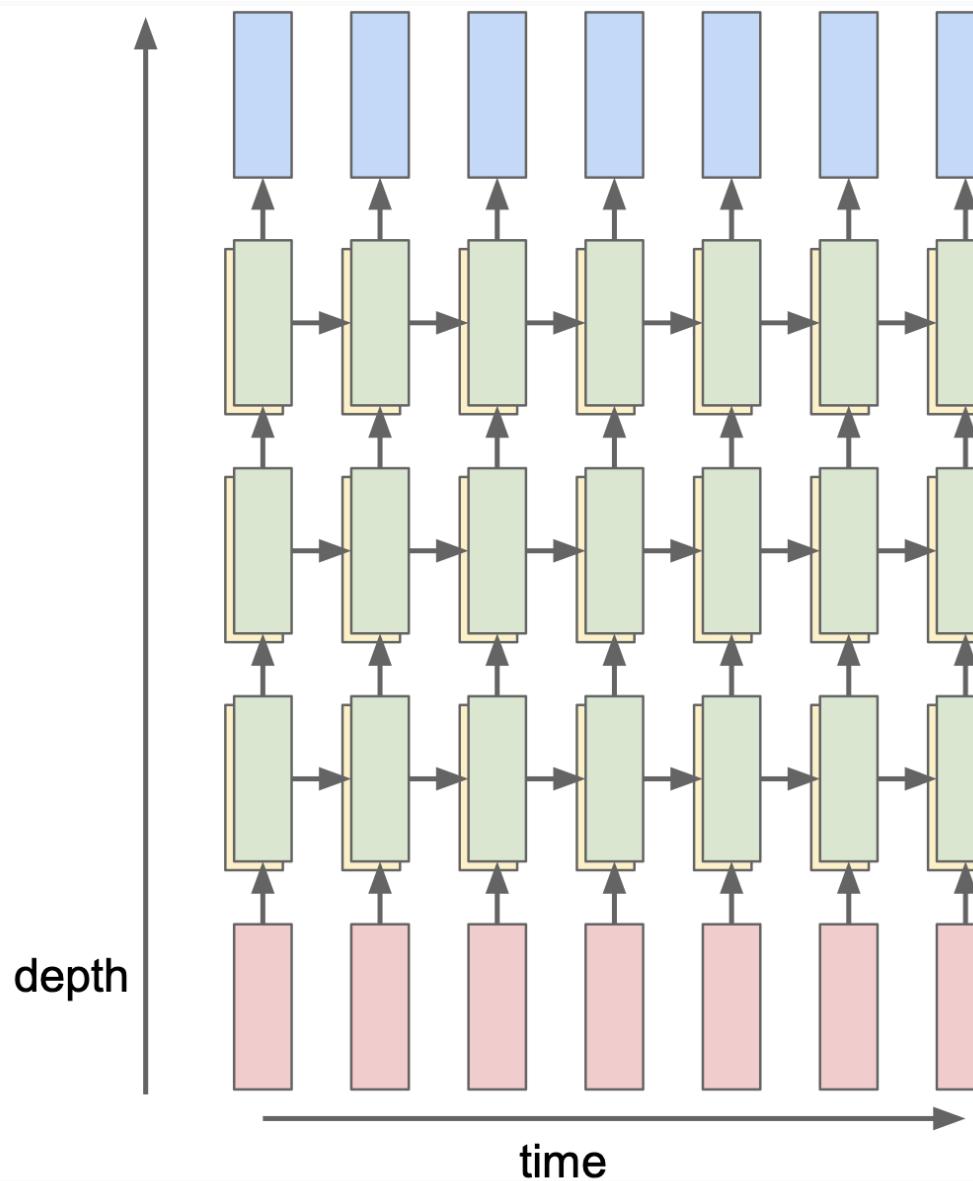
Captions generated using [neuraltalk2](#).

Visual Question Answering (VQA)



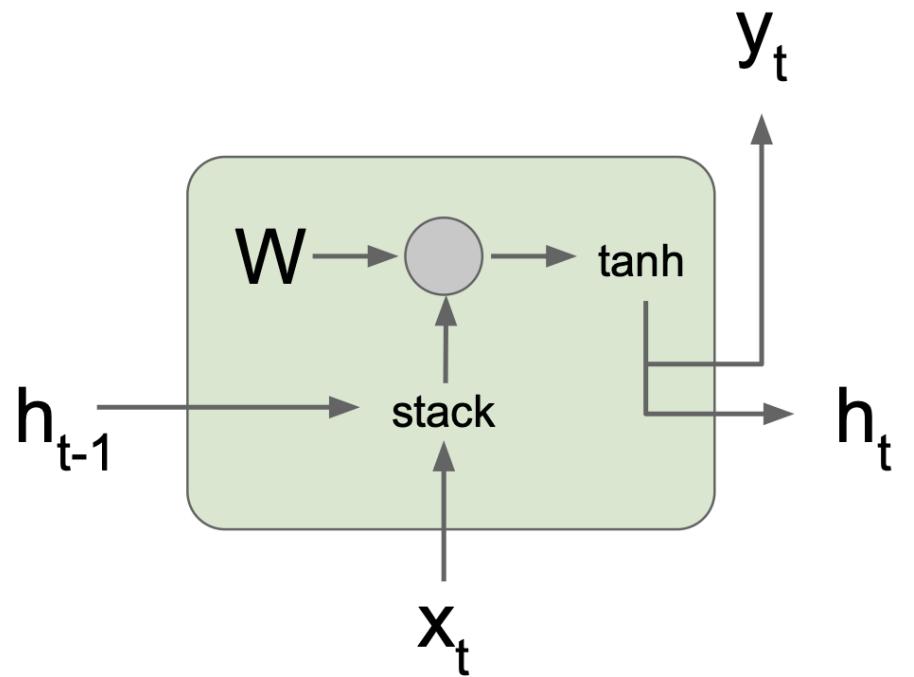
Agrawal et al, “Visual 7W: Grounded Question Answering in Images”, CVPR 2015
Figures from Agrawal et al, copyright IEEE 2015. Reproduced for educational purposes.

Multilayer RNNs



Vanilla RNN Gradient Flow

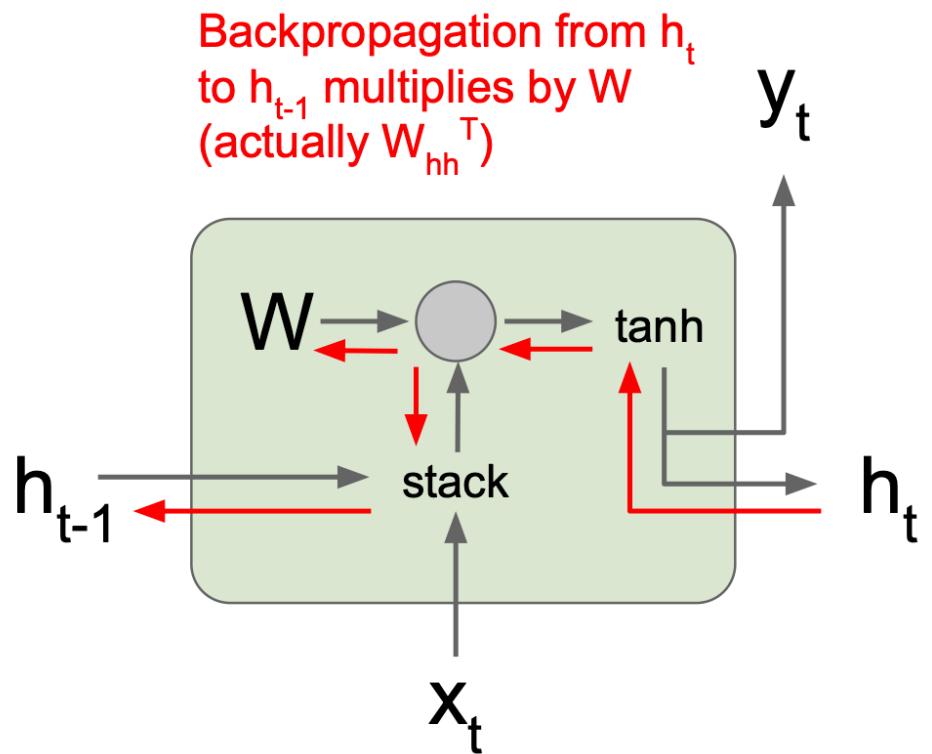
Bengio et al, “Learning long-term dependencies with gradient descent is difficult”,
IEEE Transactions on Neural Networks, 1994
Pascanu et al, “On the difficulty of training recurrent neural networks”, ICML 2013



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

Vanilla RNN Gradient Flow

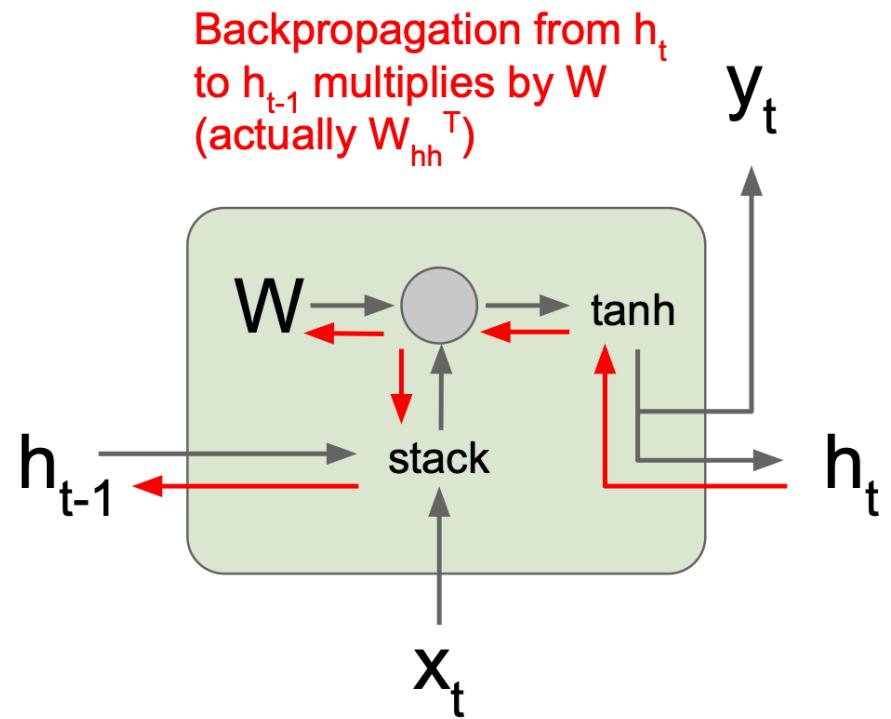
Bengio et al, “Learning long-term dependencies with gradient descent is difficult”,
IEEE Transactions on Neural Networks, 1994
Pascanu et al, “On the difficulty of training recurrent neural networks”, ICML 2013



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

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

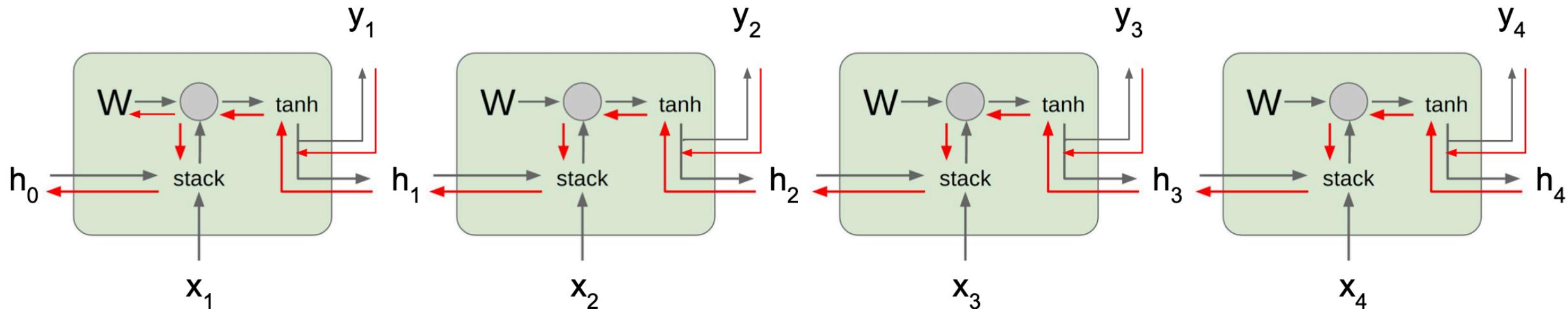


$$\begin{aligned} h_t &= \tanh(W_{hh}h_{t-1} + W_{xh}x_t) \\ &= \tanh \left(\begin{pmatrix} W_{hh} & W_{hx} \end{pmatrix} \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \right) \\ &= \tanh \left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \right) \end{aligned}$$

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

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

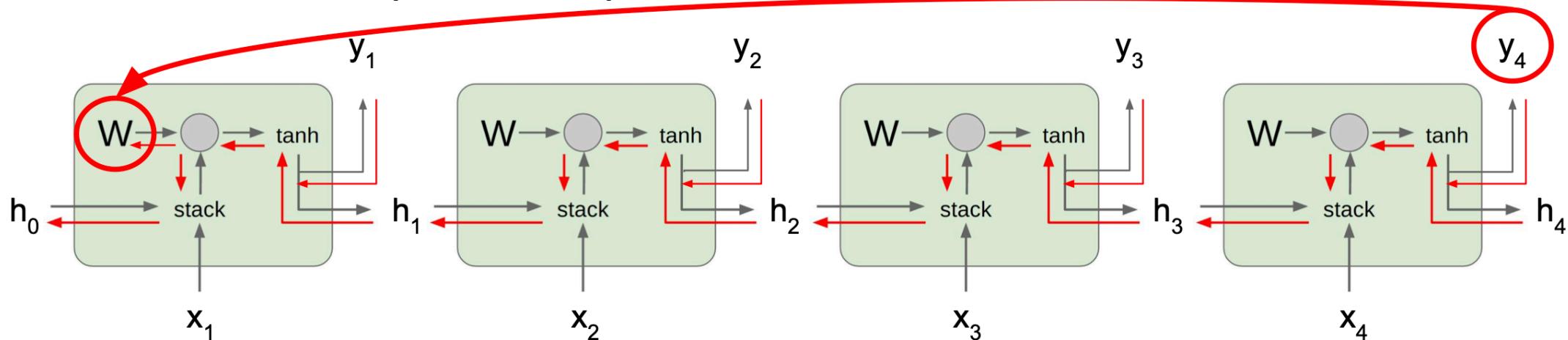


$$\frac{\partial L}{\partial W} = \sum_{t=1}^T \frac{\partial L_t}{\partial W}$$

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:



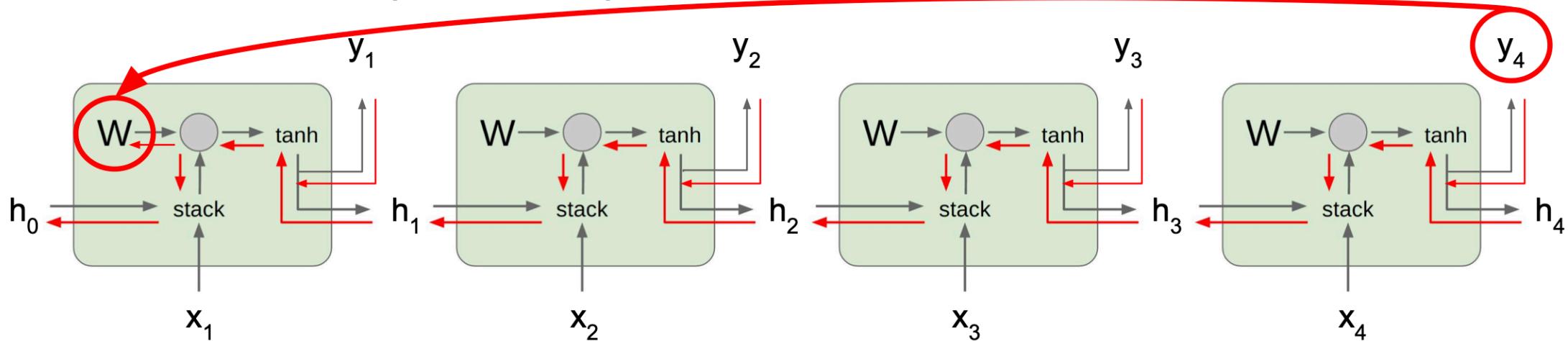
$$\frac{\partial L}{\partial W} = \sum_{t=1}^T \frac{\partial L_t}{\partial W}$$

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

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:



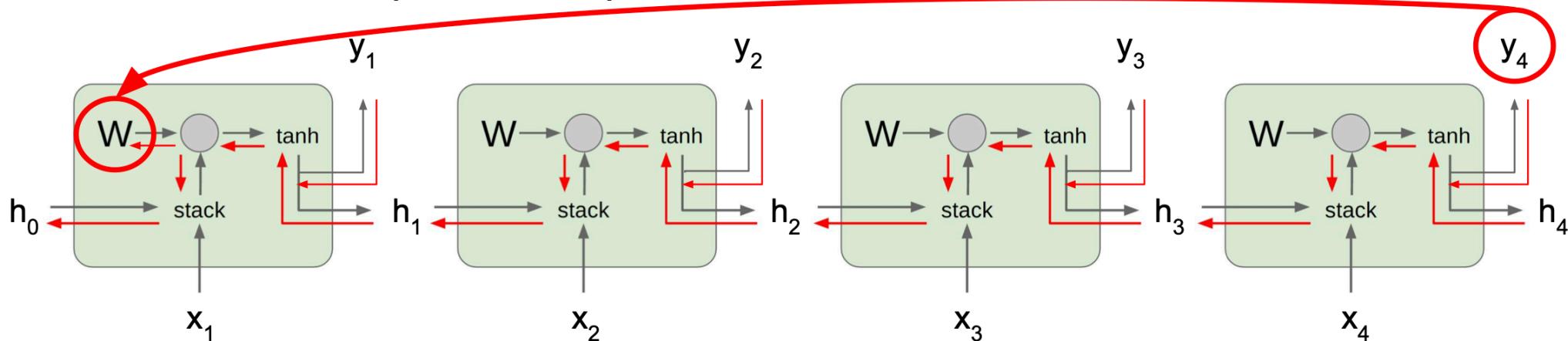
$$\frac{\partial L}{\partial W} = \sum_{t=1}^T \frac{\partial L_t}{\partial W}$$

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

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:



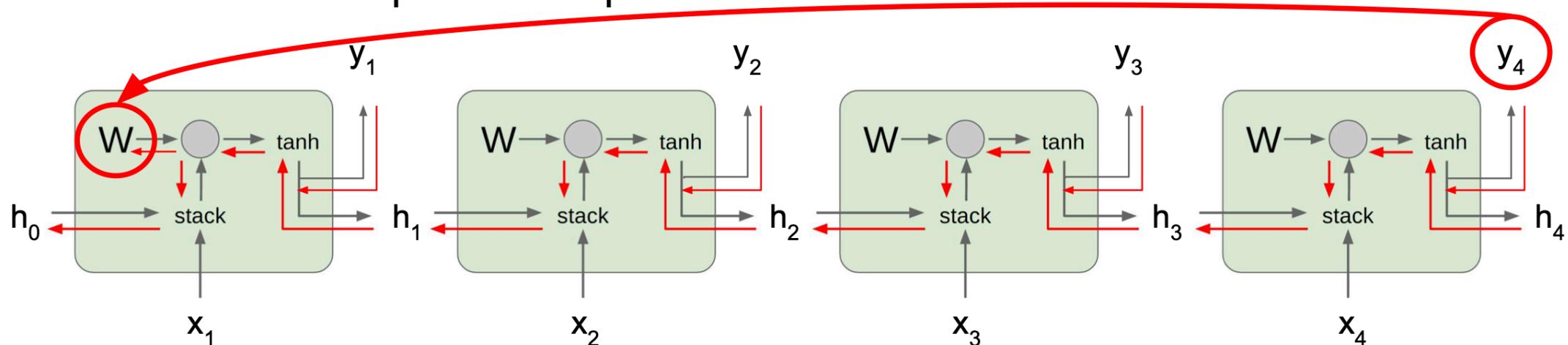
$$\frac{\partial L}{\partial W} = \sum_{t=1}^T \frac{\partial L_t}{\partial W} \quad \boxed{\frac{\partial h_t}{\partial h_{t-1}} = \tanh'(W_{hh} h_{t-1} + W_{xh} x_t) W_{hh}}$$

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

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:



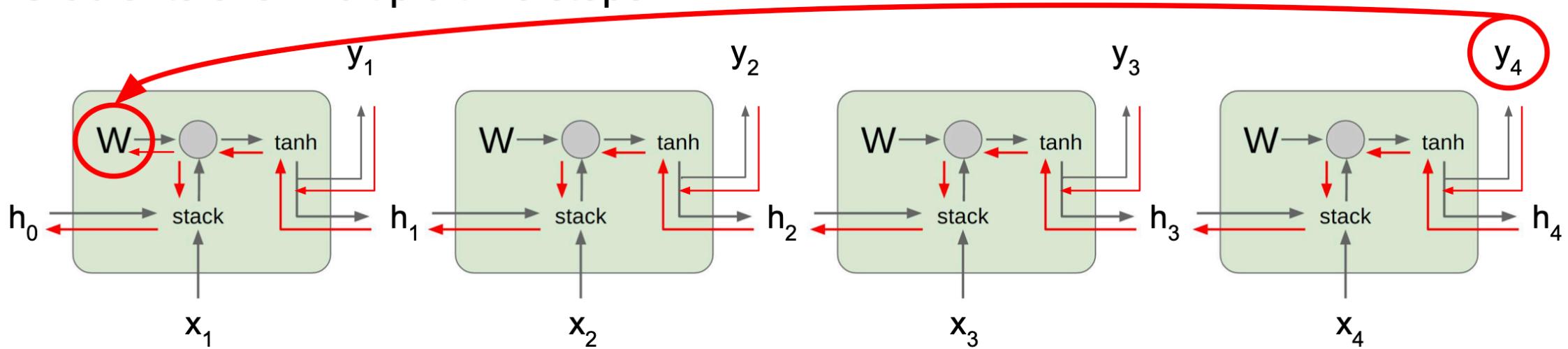
$$\frac{\partial L}{\partial W} = \sum_{t=1}^T \frac{\partial L_t}{\partial W}$$

Almost always < 1
Vanishing gradients

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

Why is Vanishing Gradient a Problem?

Gradients over multiple time steps:



- 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.

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 _____.

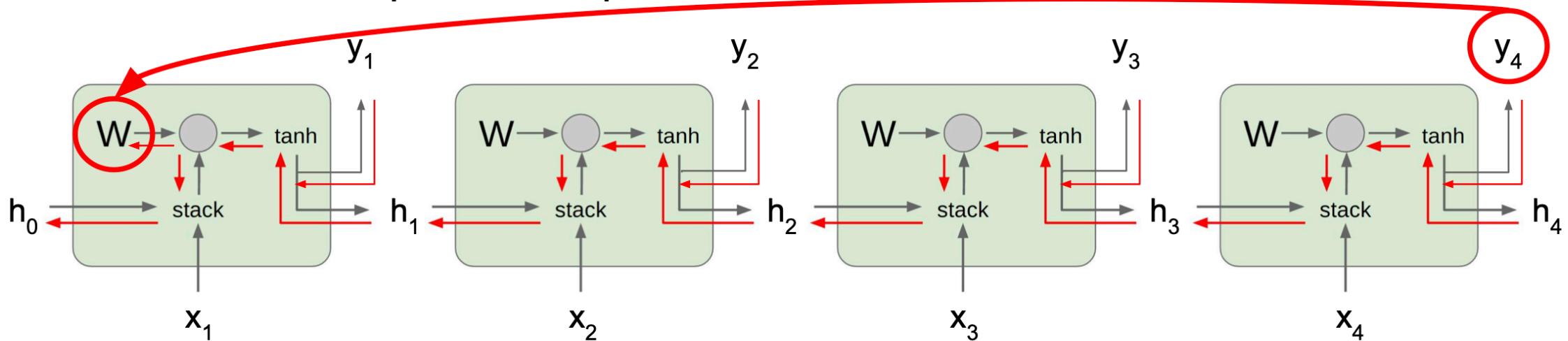
Effect of Vanishing Gradient on RNN Language Model

- **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**

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:

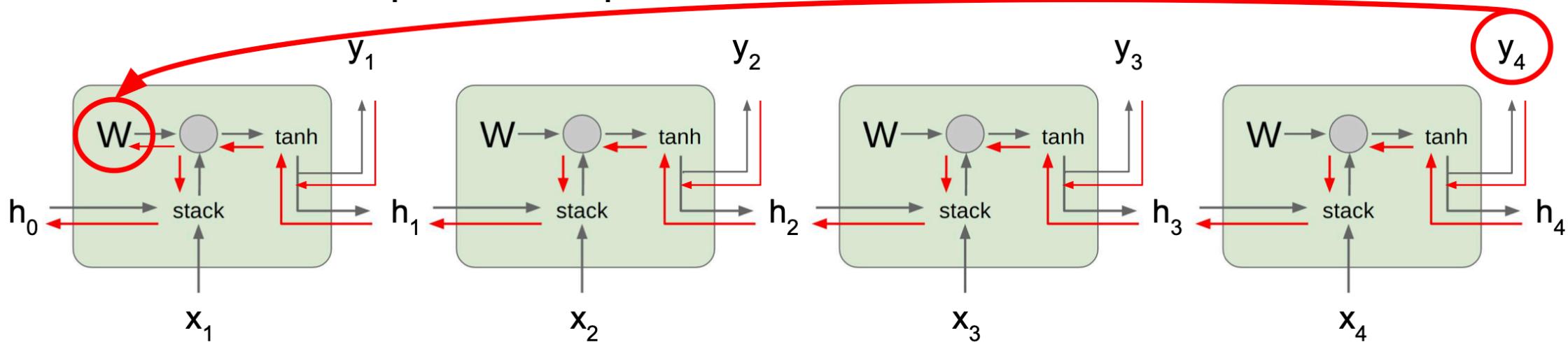


$$\frac{\partial L}{\partial W} = \sum_{t=1}^T \frac{\partial L_t}{\partial W} \quad \text{What if we assumed no non-linearity?}$$

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?

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

Largest singular value > 1 :
Exploding gradients

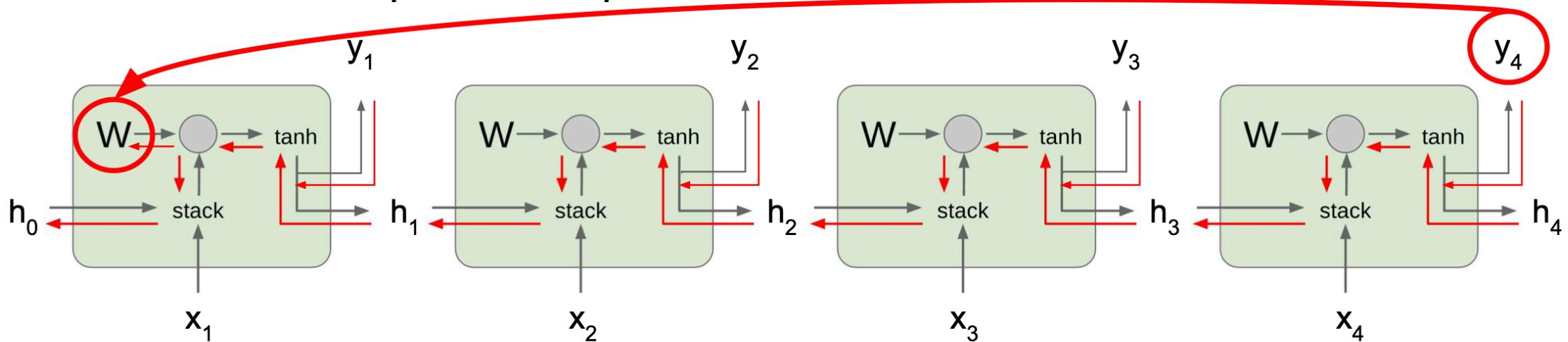
$$\frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial h_T} \boxed{W_{hh}^{T-1}} \frac{\partial h_1}{\partial W}$$

Largest singular value < 1 :
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

Largest singular value < 1:
Vanishing gradients

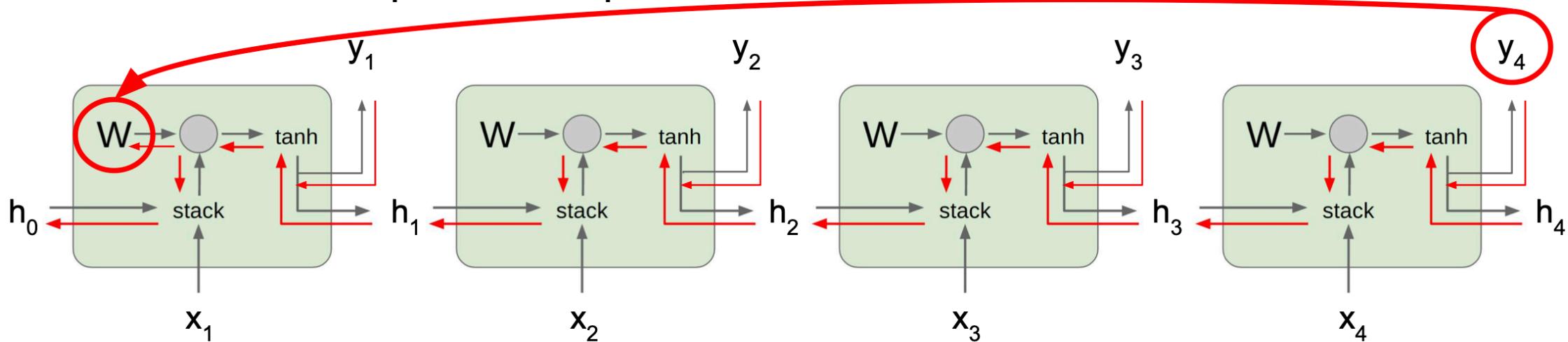
→ **Gradient clipping:**
Scale gradient if its norm is too big

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

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?

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

Largest singular value > 1 :
Exploding gradients

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

Largest singular value < 1 :
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?

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

Long Short Term Memory (LSTM)

LSTM

Four gates

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

Cell state

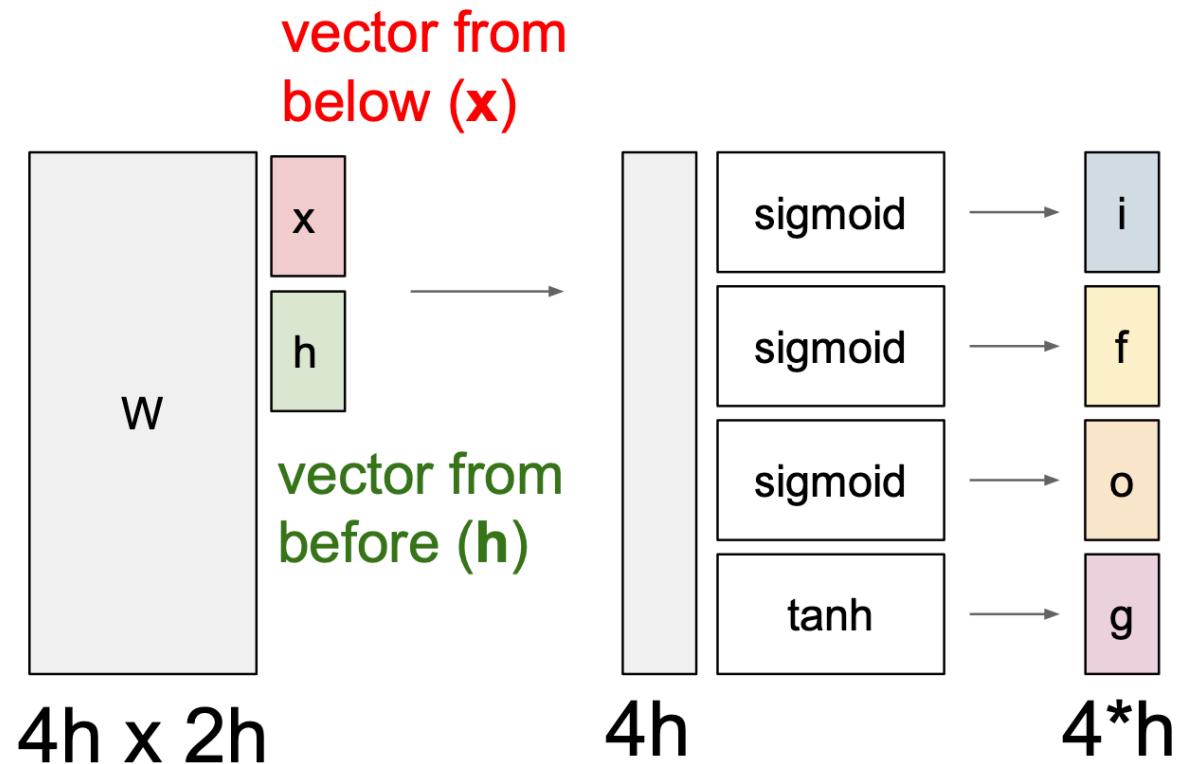
$$c_t = f \odot c_{t-1} + i \odot g$$

Hidden state

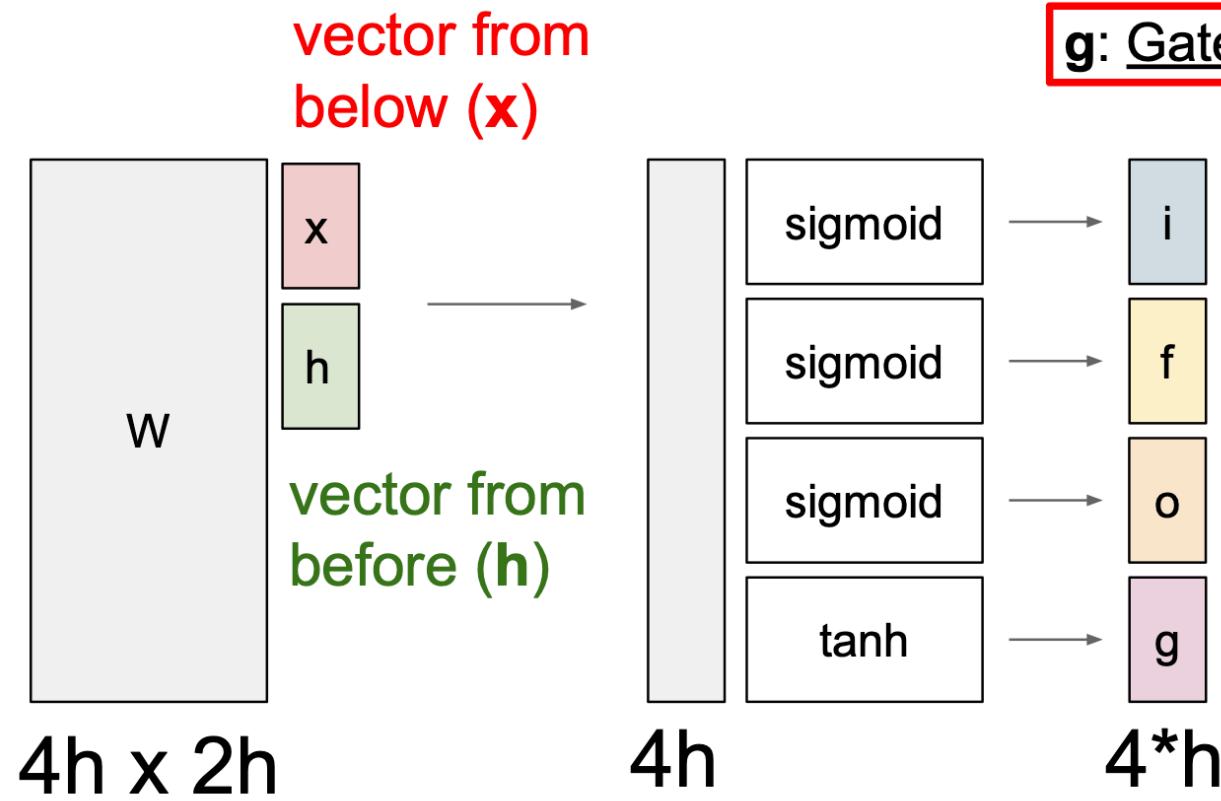
$$h_t = o \odot \tanh(c_t)$$

- 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

Long Short Term Memory (LSTM)



Long Short Term Memory (LSTM)



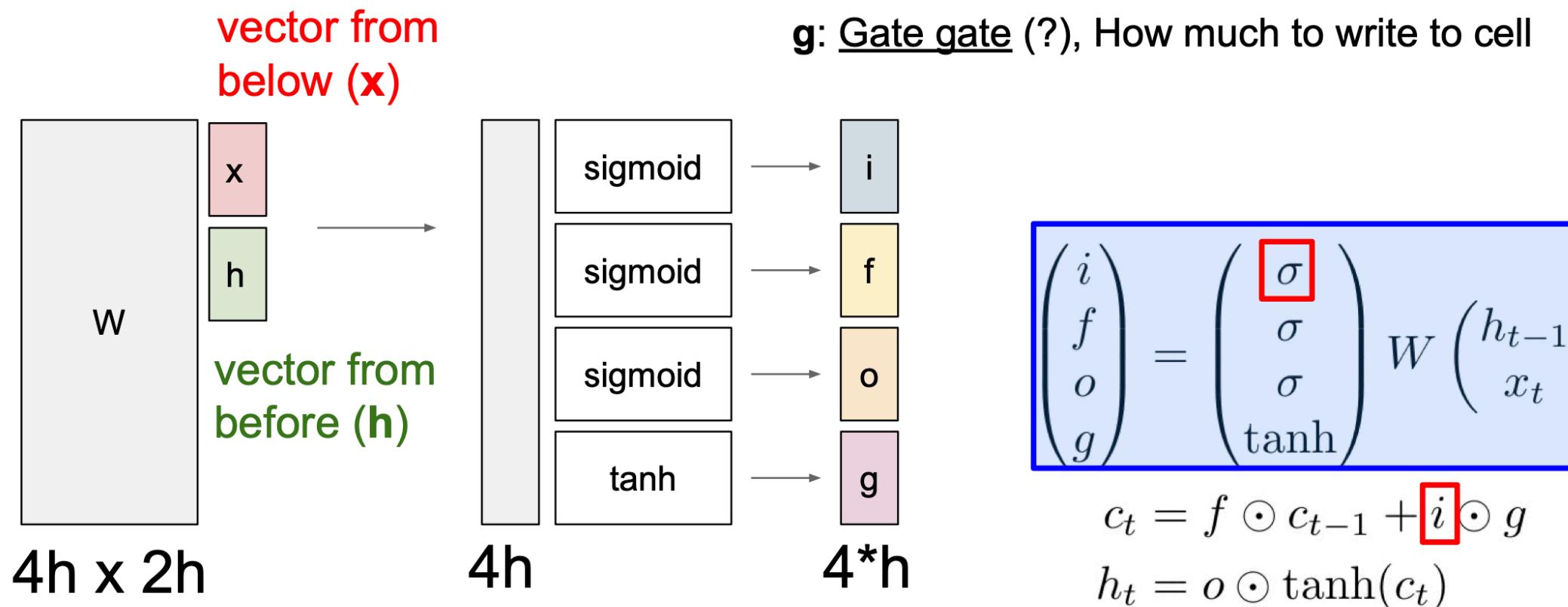
g : Gate (?), How much to write to cell

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

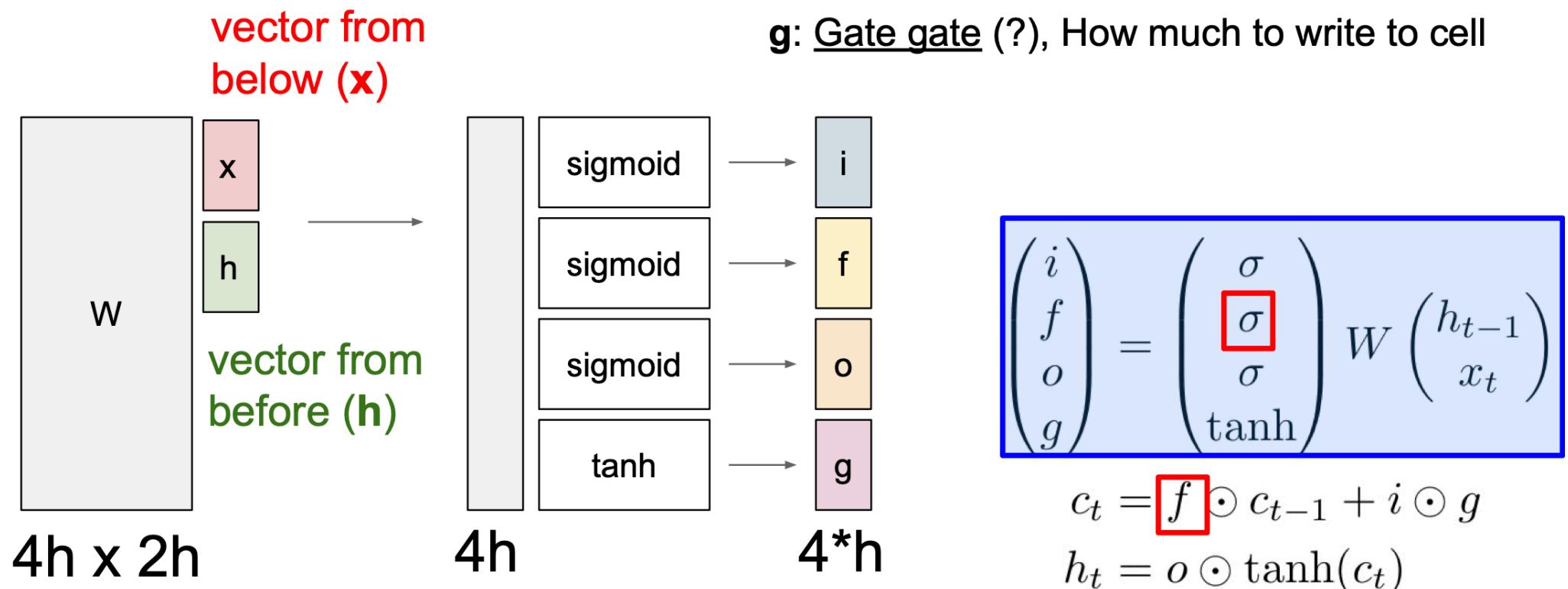
$$c_t = f \odot c_{t-1} + i \odot g$$

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

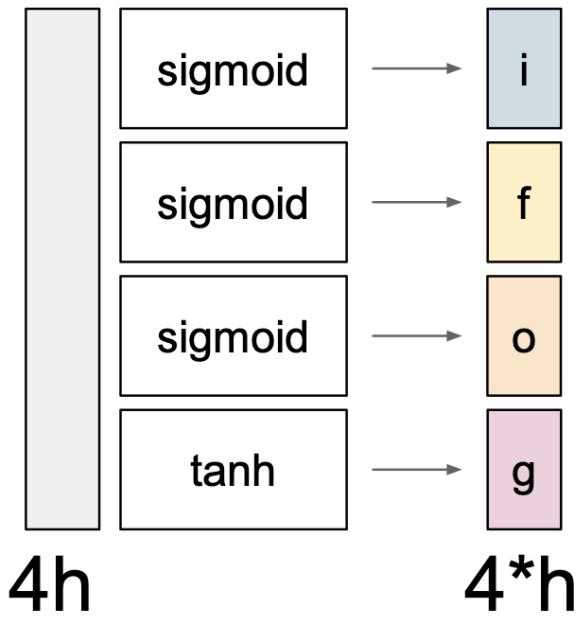
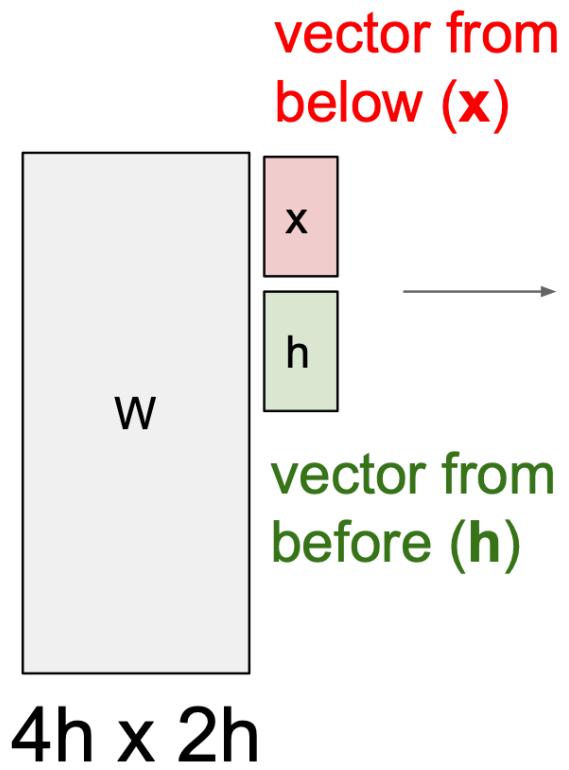
Long Short Term Memory (LSTM)



Long Short Term Memory (LSTM)



Long Short Term Memory (LSTM)

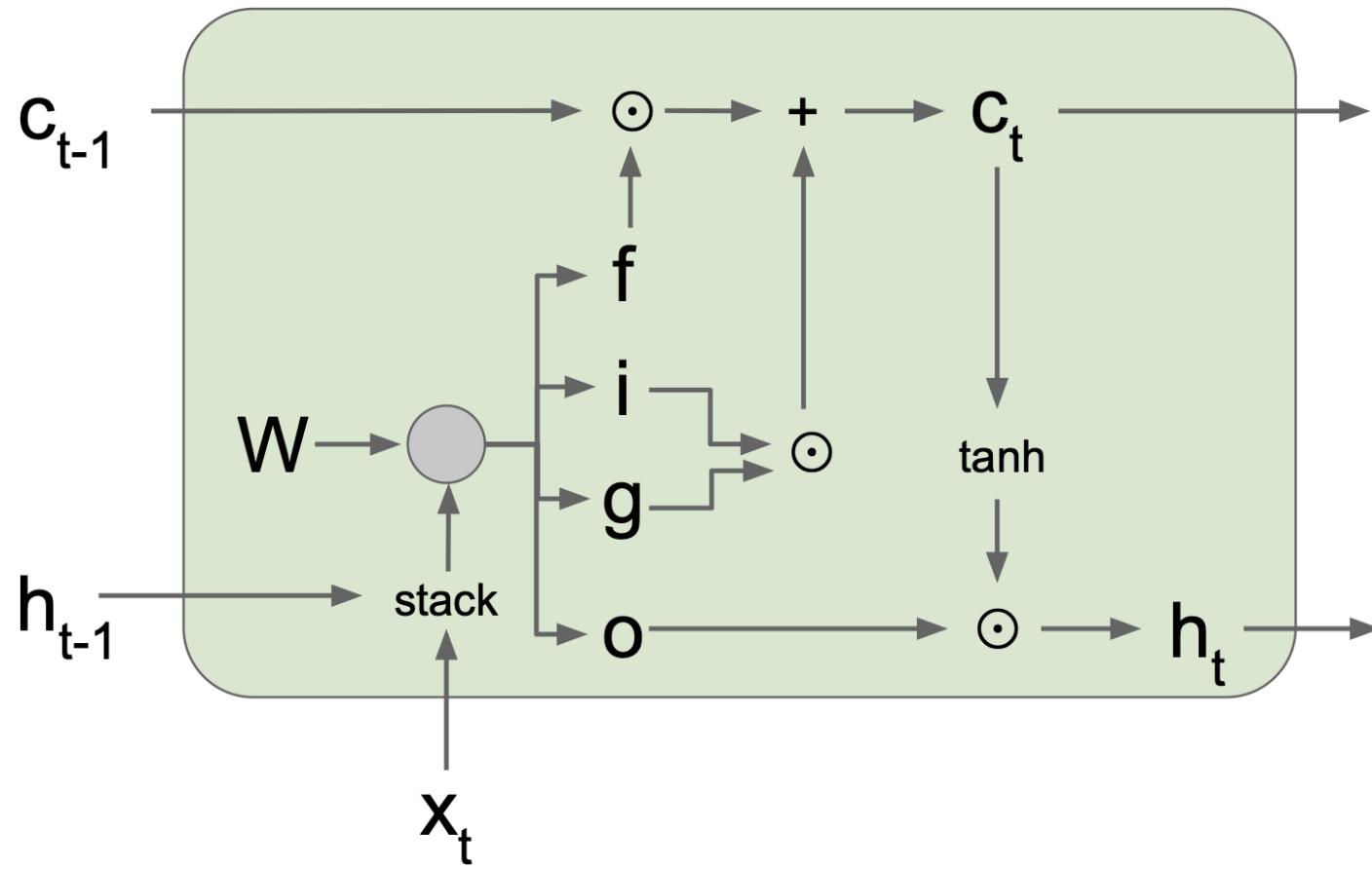


i: Input gate, whether to write to cell
f: Forget gate, Whether to erase cell
o: Output gate, How much to reveal cell
g: Gate gate (?), How much to write to cell

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

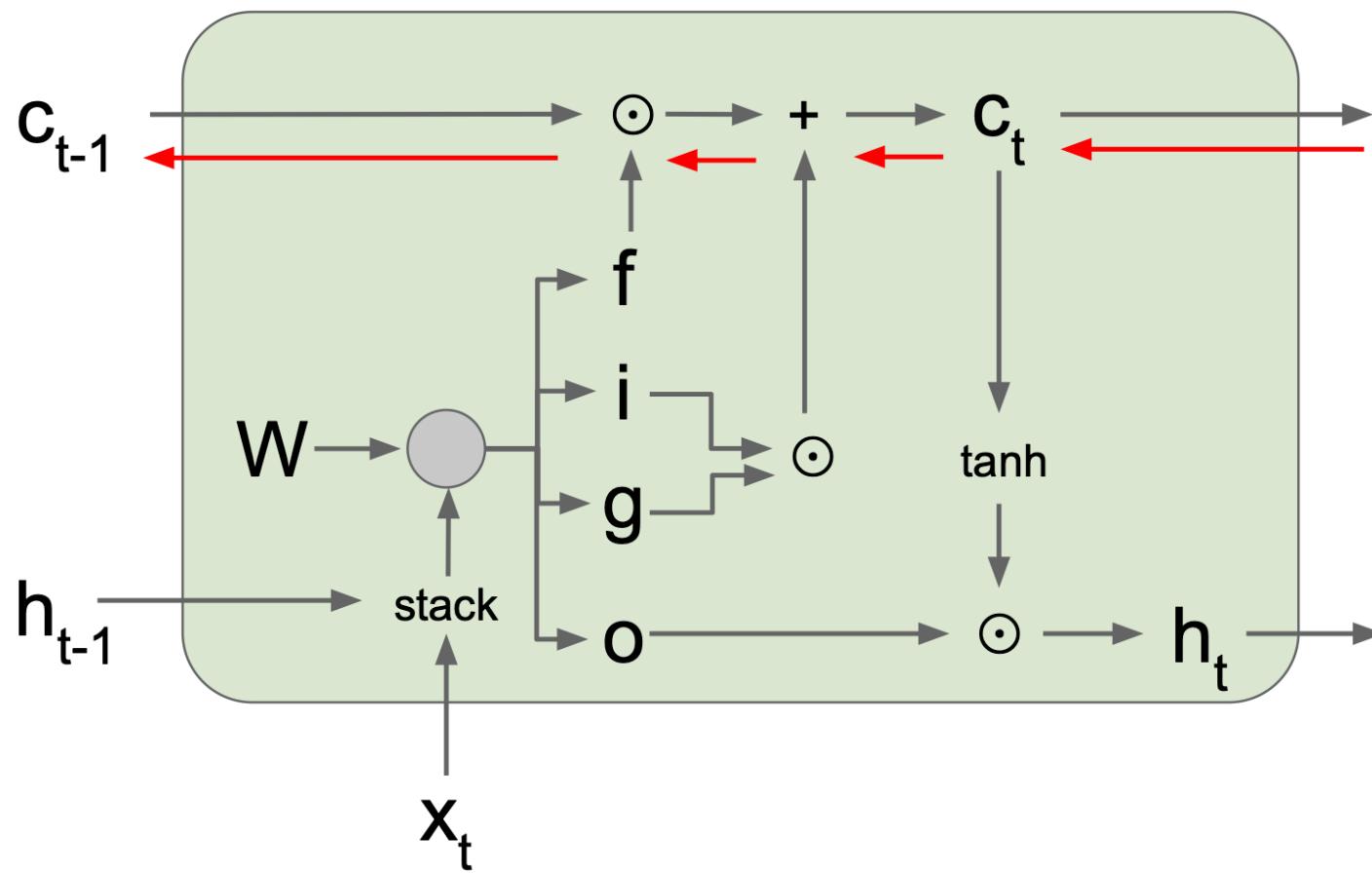
$$c_t = f \odot c_{t-1} + i \odot g$$
$$h_t = o \odot \tanh(c_t)$$

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

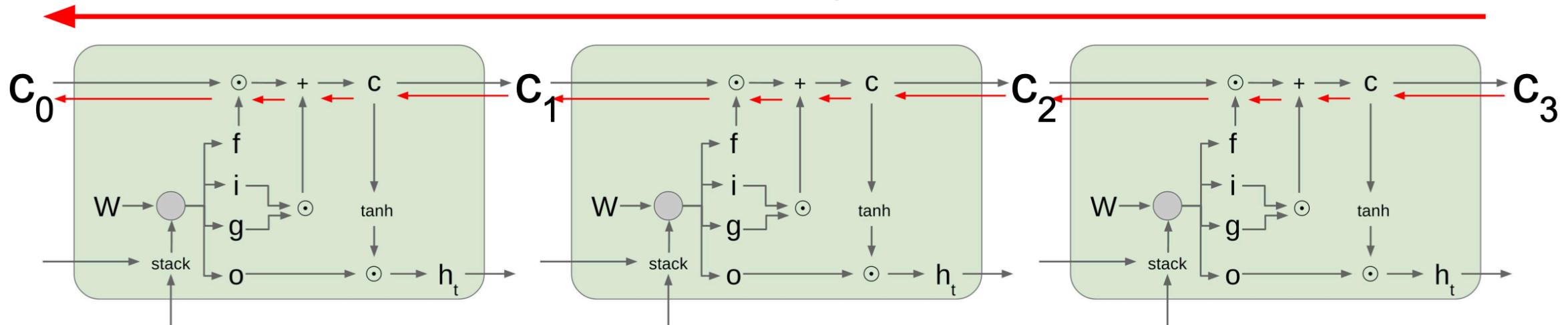
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

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

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

LSTM Gradient Flow

Uninterrupted 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

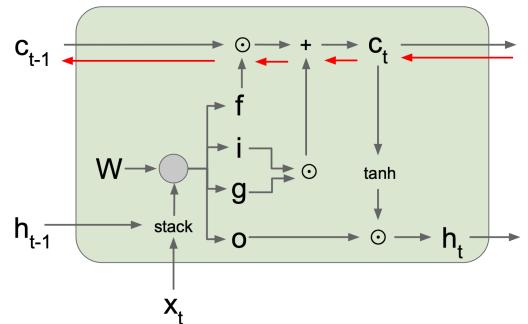
Do LSTMs Solve the Vanishing Gradient Problem?

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 W_h that preserves info in hidden state

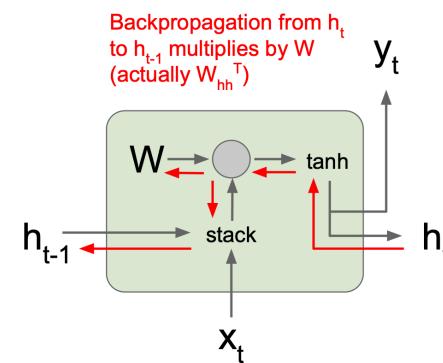
LSTM **doesn't guarantee** that there is no vanishing/exploding gradient, but it does provide an easier way for the model to learn long-distance dependencies

[Hochreiter et al., 1997]



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

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$
$$c_t = f \odot c_{t-1} + i \odot g$$
$$h_t = o \odot \tanh(c_t)$$

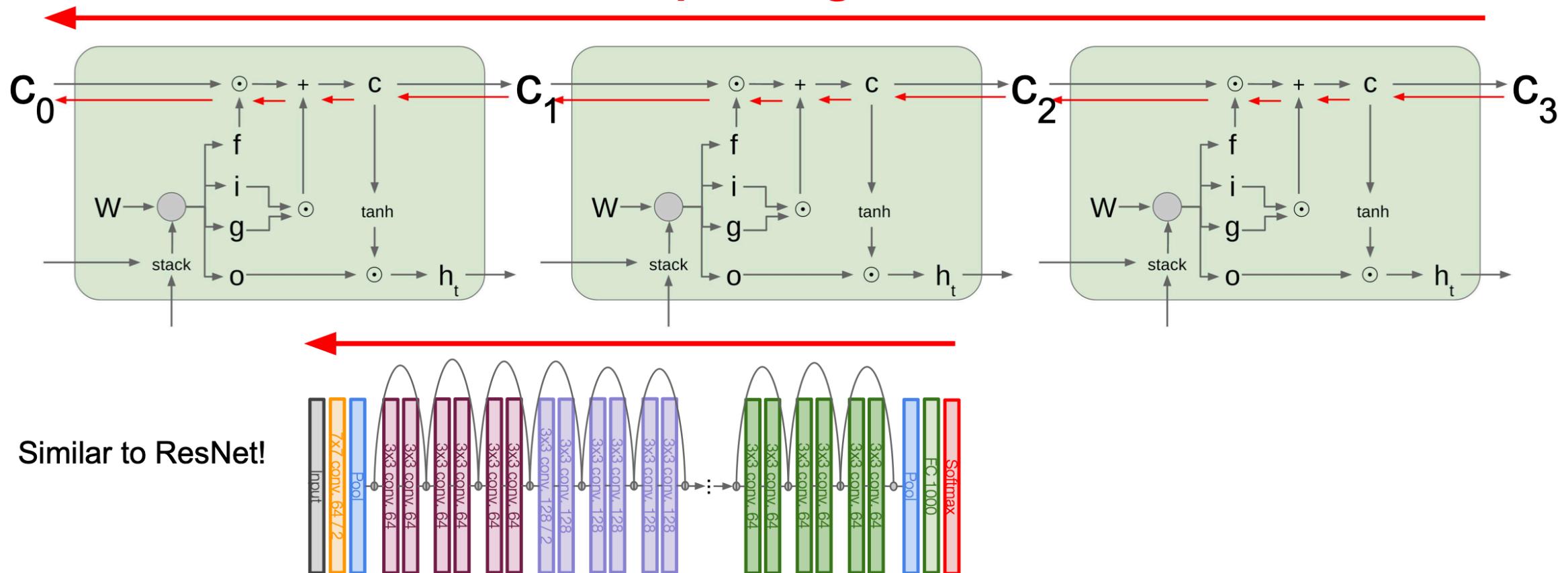


Backpropagation from h_t to h_{t-1} multiplies by W (actually W_{hh}^T)

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

LSTM Gradient Flow

Uninterrupted gradient flow!



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

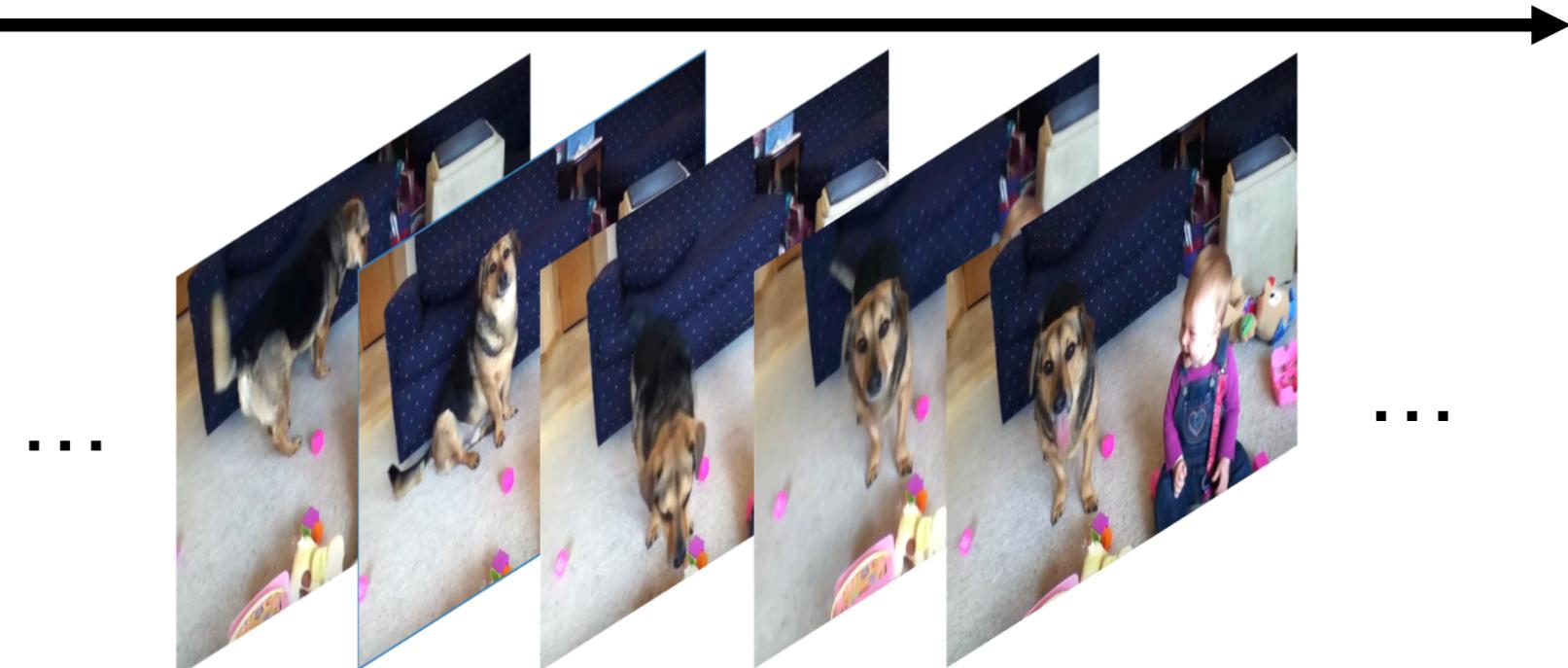
Slides are borrowed from Stanford CS231N

Video = 2D + Time

A video is a **sequence** of images

4D tensor: $T \times 3 \times H \times W$

(or $3 \times T \times H \times W$)



Example Task: Video Classification



Input video:
 $T \times 3 \times H \times W$



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!

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**

Input video:
 $T \times 3 \times H \times W$

Slide credit: Justin Johnson

Problems: Videos are Big!



Input video:
 $T \times 3 \times H \times W$

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



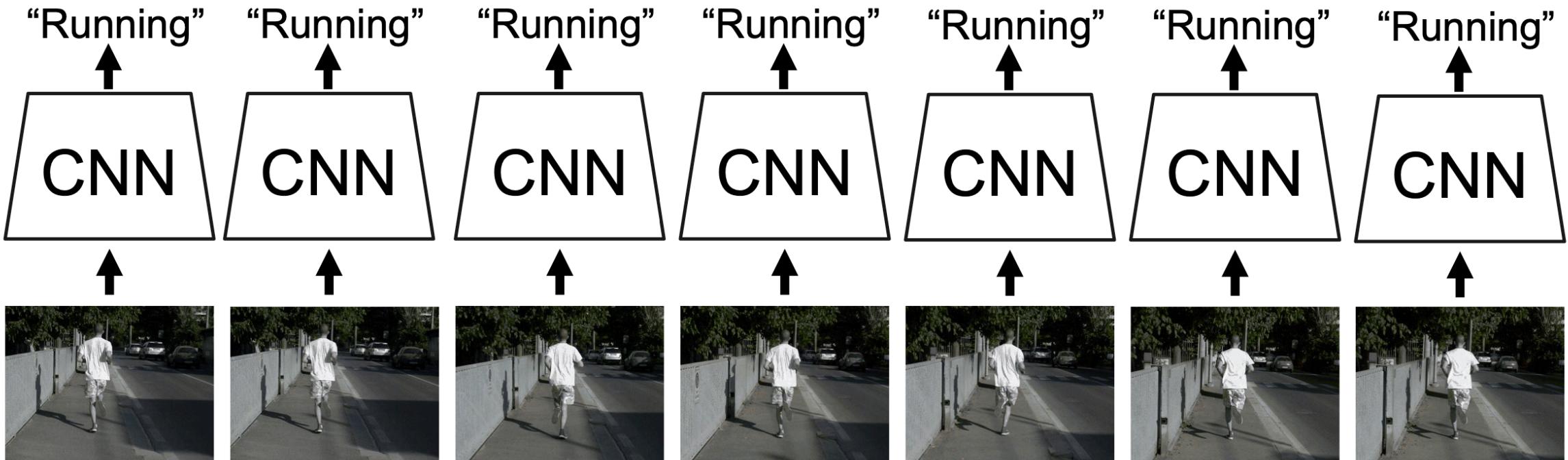
Testing: Run model on different clips, average predictions



Slide credit: Justin Johnson

Video Classification: Single-Frame CNN

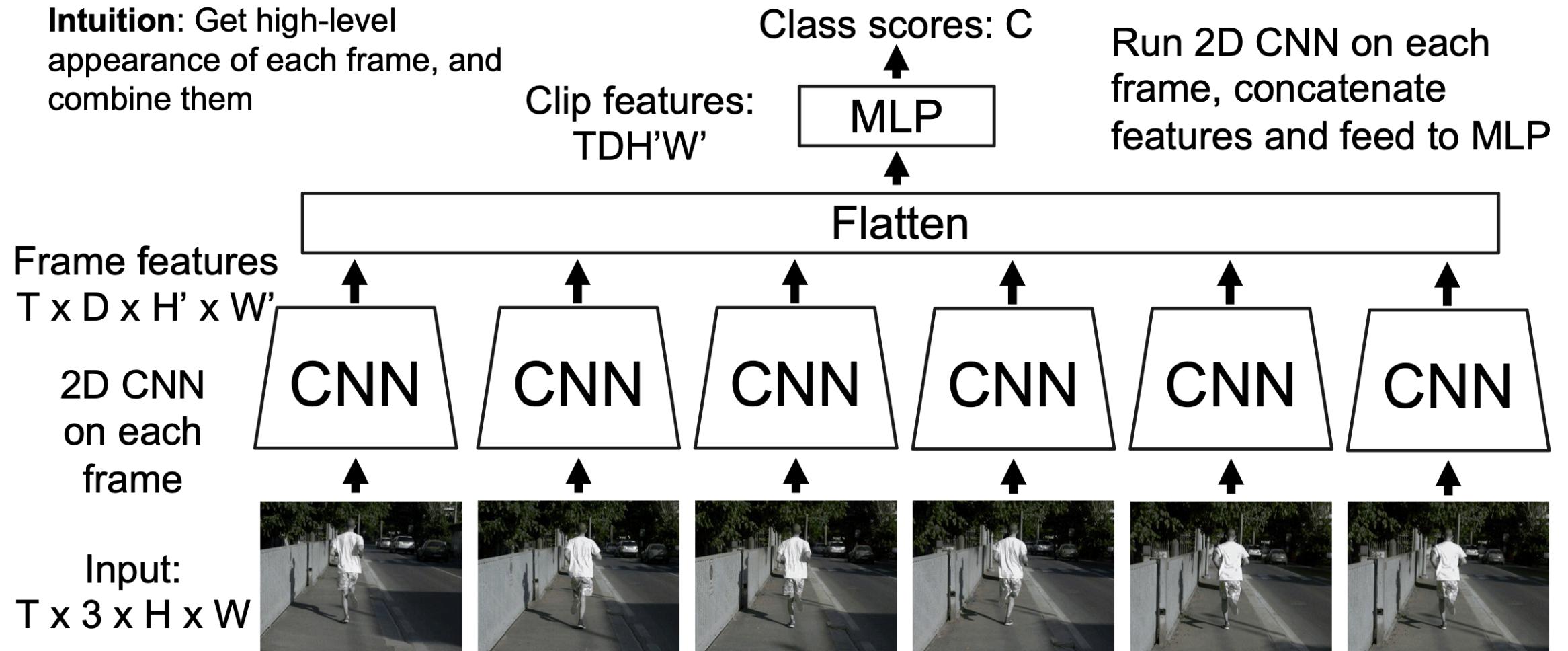
Simple idea: train normal 2D CNN to classify video frames independently!
(Average predicted probs at test-time)
Often a **very** strong baseline for video classification



Slide credit: Justin Johnson

Video Classification: Late Fusion with FC

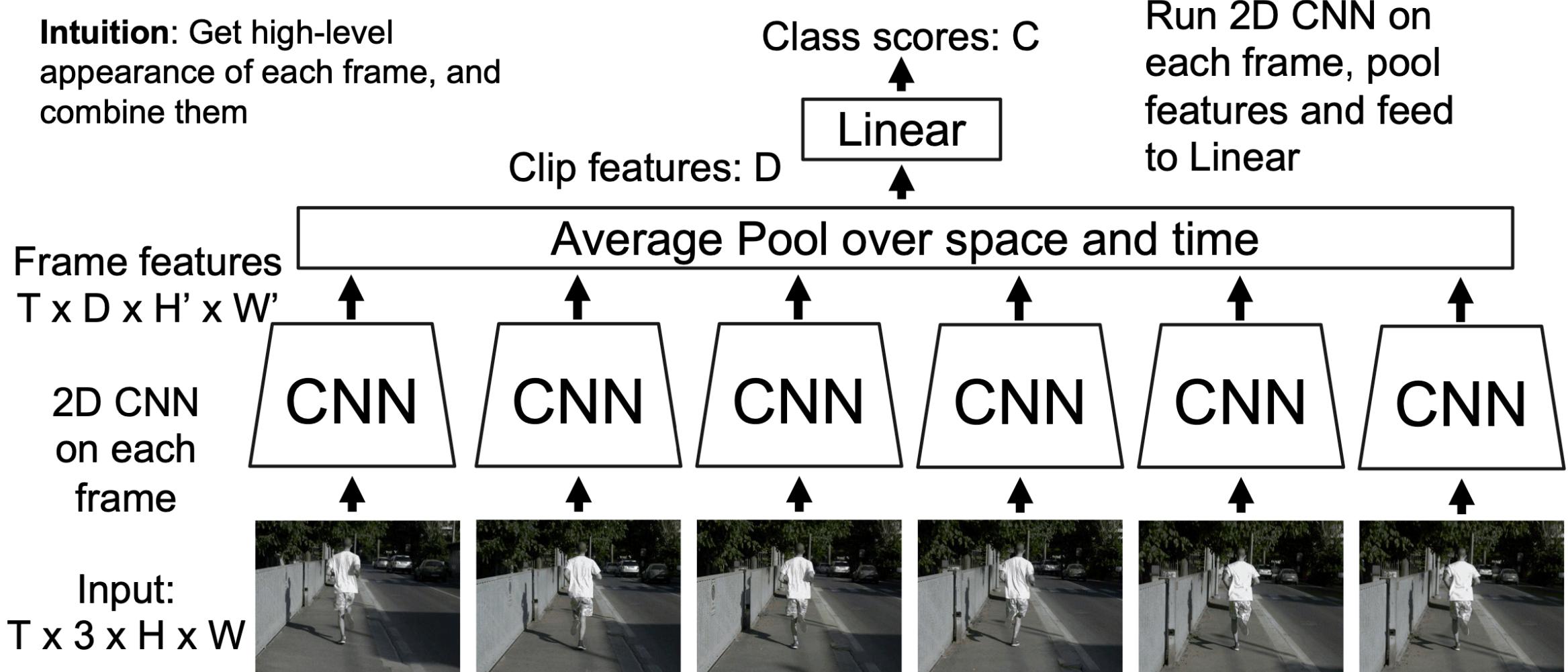
Intuition: Get high-level appearance of each frame, and combine them



Slide credit: Justin Johnson

Video Classification: Late Fusion with Pooling

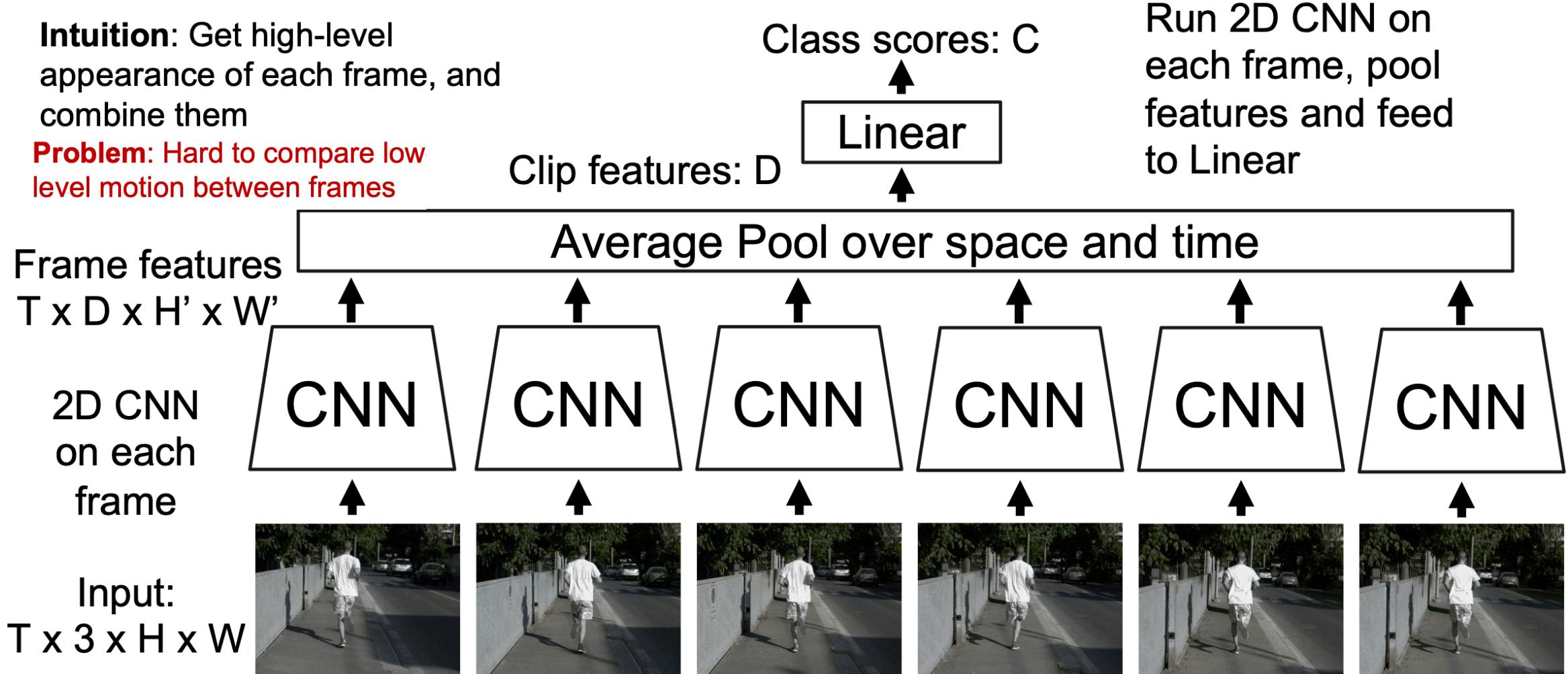
Intuition: Get high-level appearance of each frame, and combine them



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



Video Classification: Early Fusion

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

Reshape:
 $3T \times H \times W$

Input:
 $T \times 3 \times H \times W$

First 2D convolution collapses all temporal information:
Input: $3T \times H \times W$
Output: $D \times H \times W$



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!

Reshape:
 $3T \times H \times W$

Input:
 $T \times 3 \times H \times W$

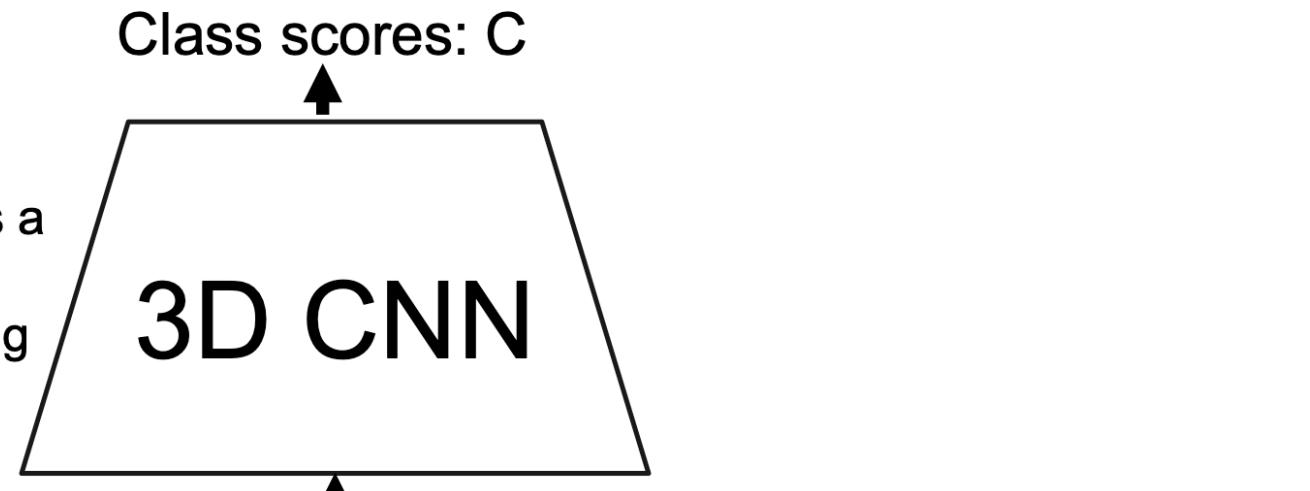


Video Classification: 3D CNN

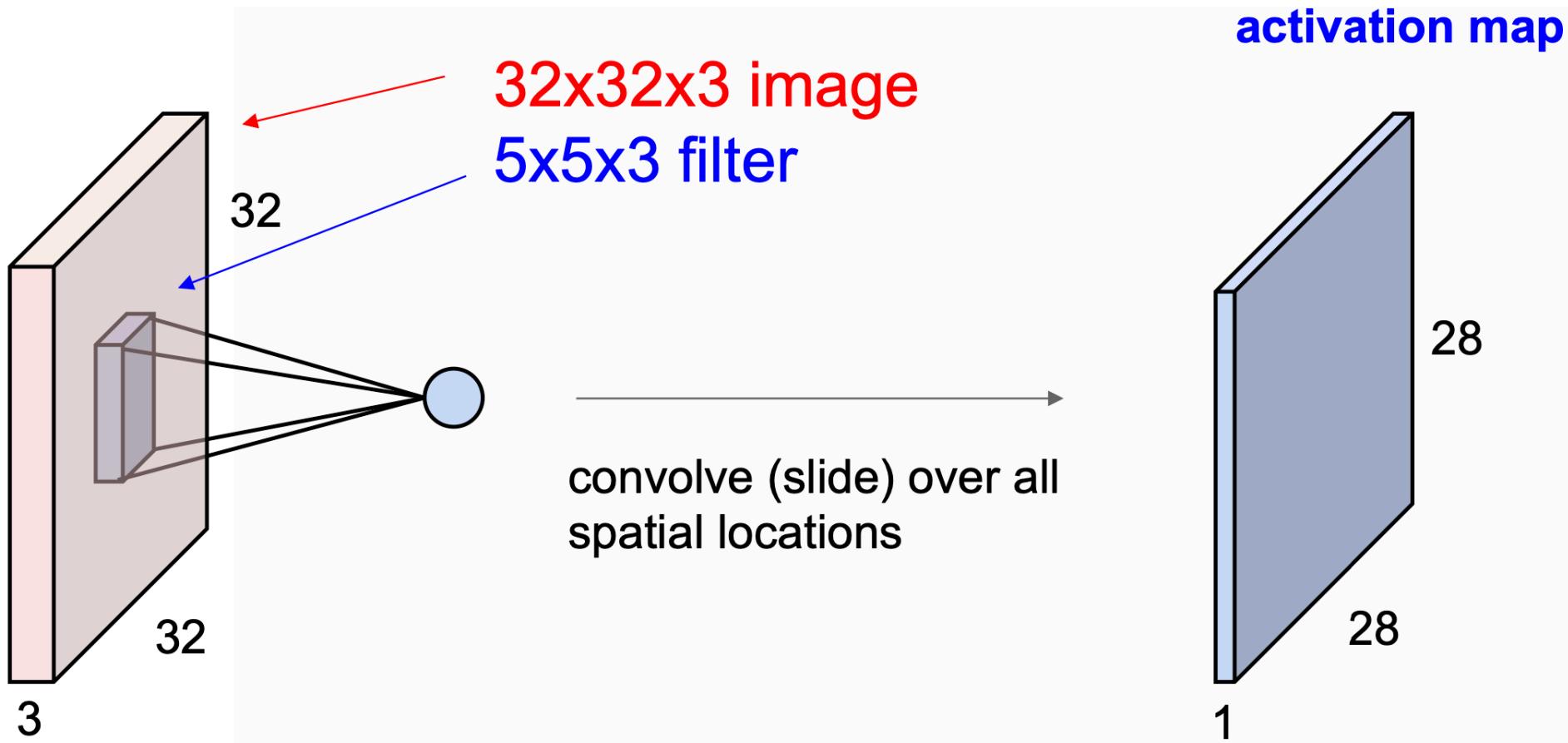
Intuition: Use 3D versions of convolution and pooling to slowly fuse temporal information over the course of the network

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

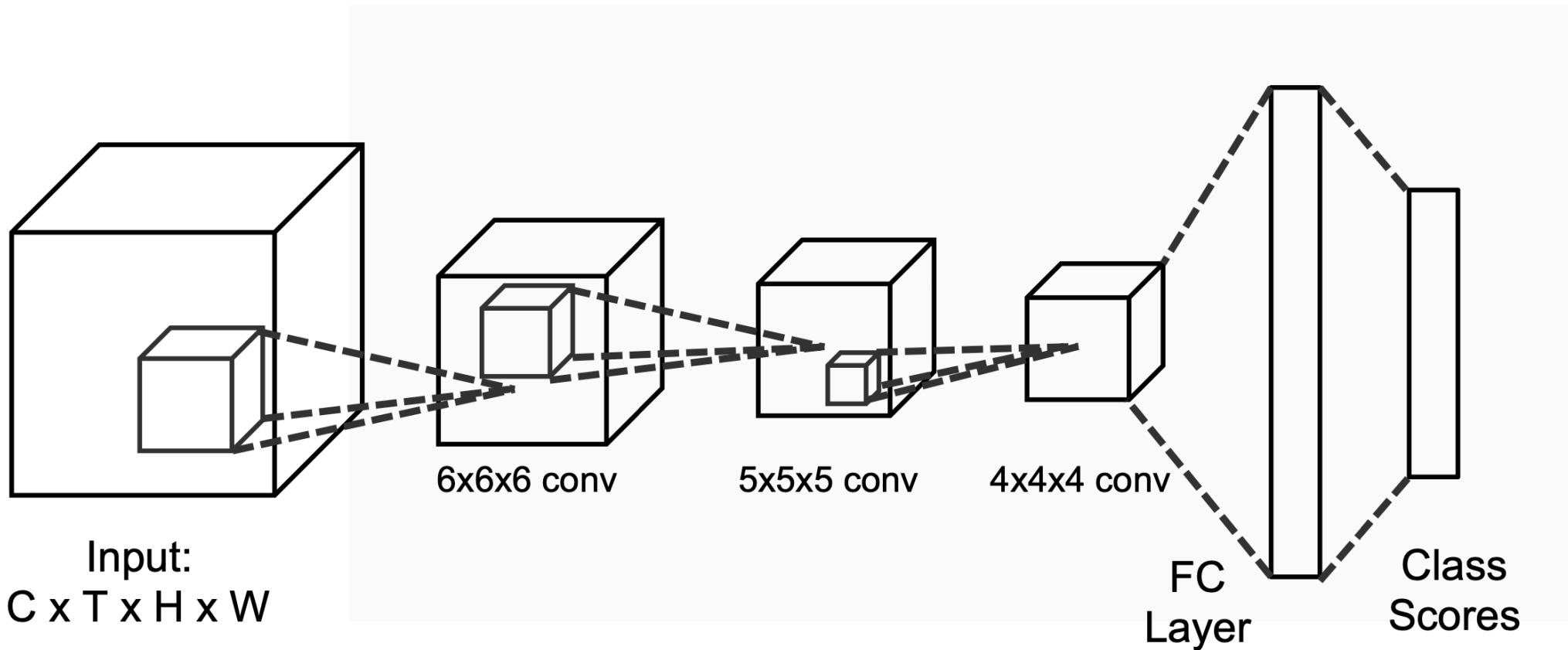
Input:
 $3 \times T \times H \times W$



Conv Layer



3D Convolution



Early Fusion vs. Late Fusion vs. 3DCNN

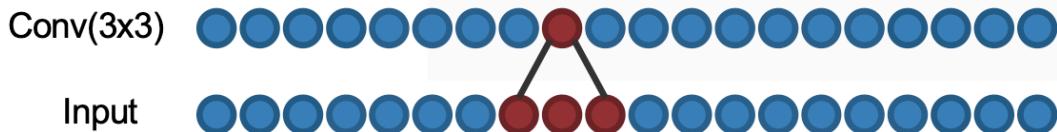
Late
Fusion

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

Early Fusion vs. Late Fusion vs. 3DCNN

Late
Fusion

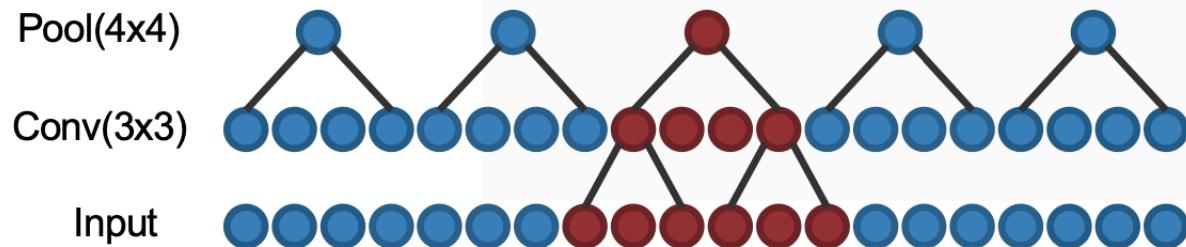
Layer	Size (C x T x H x W)	Receptive Field (T x H x W)
Input	3 x 20 x 64 x 64	
Conv2D(3x3, 3->12)	12 x 20 x 64 x 64	1 x 3 x 3



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	$3 \times 20 \times 64 \times 64$	
Conv2D(3x3, 3->12)	$12 \times 20 \times 64 \times 64$	$1 \times 3 \times 3$
Pool2D(4x4)	$12 \times 20 \times 16 \times 16$	$1 \times 6 \times 6$

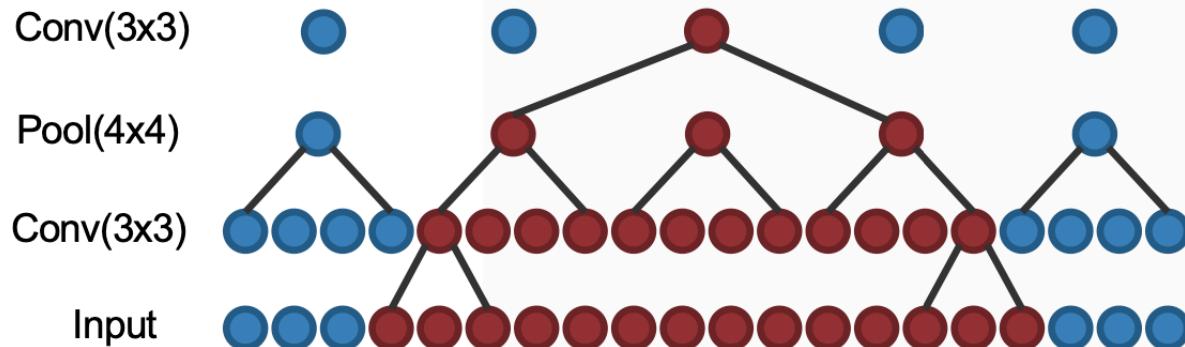


Early Fusion vs. Late Fusion vs. 3DCNN

Late
Fusion

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

Build slowly in space

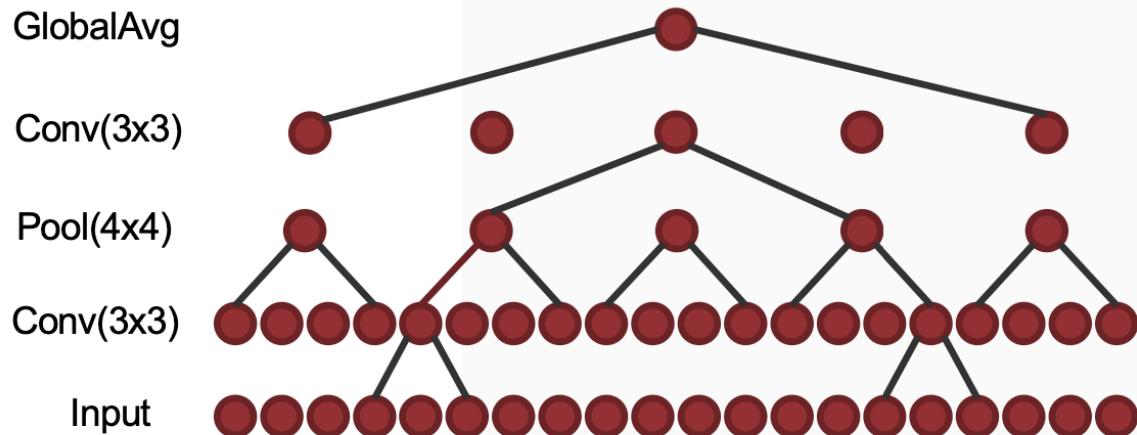


Early Fusion vs. Late Fusion vs. 3DCNN

Late
Fusion

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

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



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
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

Early
Fusion

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

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

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	$3 \times 20 \times 64 \times 64$	
Conv2D(3x3, 3->12)	$12 \times 20 \times 64 \times 64$	$1 \times 3 \times 3$
Pool2D(4x4)	$12 \times 20 \times 16 \times 16$	$1 \times 6 \times 6$
Conv2D(3x3, 12->24)	$24 \times 20 \times 16 \times 16$	$1 \times 14 \times 14$
GlobalAvgPool	$24 \times 1 \times 1 \times 1$	$20 \times 64 \times 64$

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

Early
Fusion

Input	$3 \times 20 \times 64 \times 64$	
Conv2D(3x3, 3*20->12)	$12 \times 64 \times 64$	$20 \times 3 \times 3$
Pool2D(4x4)	$12 \times 16 \times 16$	$20 \times 6 \times 6$
Conv2D(3x3, 12->24)	$24 \times 16 \times 16$	$20 \times 14 \times 14$
GlobalAvgPool	$24 \times 1 \times 1$	$20 \times 64 \times 64$

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

3D
CNN

Input	$3 \times 20 \times 64 \times 64$	
Conv3D(3x3x3, 3->12)	$12 \times 20 \times 64 \times 64$	$3 \times 3 \times 3$
Pool3D(4x4x4)	$12 \times 5 \times 16 \times 16$	$6 \times 6 \times 6$
Conv3D(3x3x3, 12->24)	$24 \times 5 \times 16 \times 16$	$14 \times 14 \times 14$
GlobalAvgPool	$24 \times 1 \times 1$	$20 \times 64 \times 64$

Build slowly in space,
Build slowly in time
"Slow Fusion"

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	$3 \times 20 \times 64 \times 64$	
Conv2D(3x3, 3->12)	$12 \times 20 \times 64 \times 64$	$1 \times 3 \times 3$
Pool2D(4x4)	$12 \times 20 \times 16 \times 16$	$1 \times 6 \times 6$
Conv2D(3x3, 12->24)	$24 \times 20 \times 16 \times 16$	$1 \times 14 \times 14$
GlobalAvgPool	$24 \times 1 \times 1 \times 1$	$20 \times 64 \times 64$

Early
Fusion

Input	$3 \times 20 \times 64 \times 64$	
Conv2D(3x3, 3*20->12)	$12 \times 64 \times 64$	$20 \times 3 \times 3$
Pool2D(4x4)	$12 \times 16 \times 16$	$20 \times 6 \times 6$
Conv2D(3x3, 12->24)	$24 \times 16 \times 16$	$20 \times 14 \times 14$
GlobalAvgPool	$24 \times 1 \times 1$	$20 \times 64 \times 64$

3D
CNN

Input	$3 \times 20 \times 64 \times 64$	
Conv3D(3x3x3, 3->12)	$12 \times 20 \times 64 \times 64$	$3 \times 3 \times 3$
Pool3D(4x4x4)	$12 \times 5 \times 16 \times 16$	$6 \times 6 \times 6$
Conv3D(3x3x3, 12->24)	$24 \times 5 \times 16 \times 16$	$14 \times 14 \times 14$
GlobalAvgPool	$24 \times 1 \times 1$	$20 \times 64 \times 64$

What is the difference?

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

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

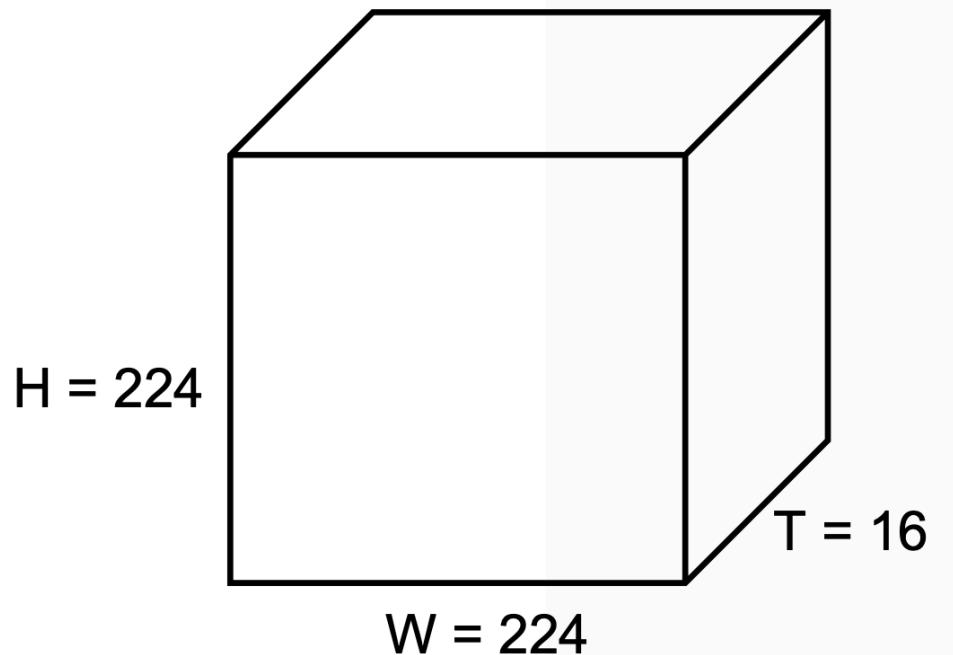
Build slowly in space,
Build slowly in time
"Slow Fusion"

(Small example architectures, in practice much bigger)

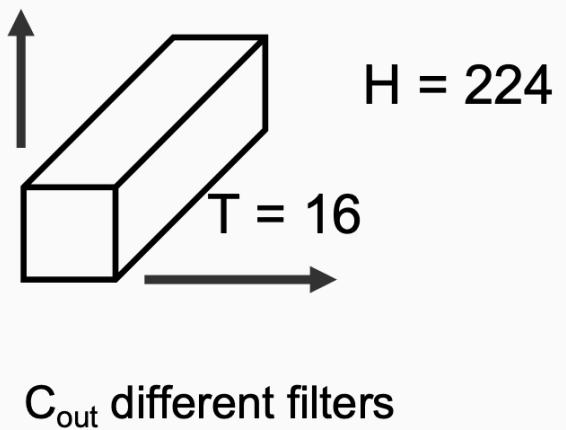
Slide credit: Justin Johnson

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

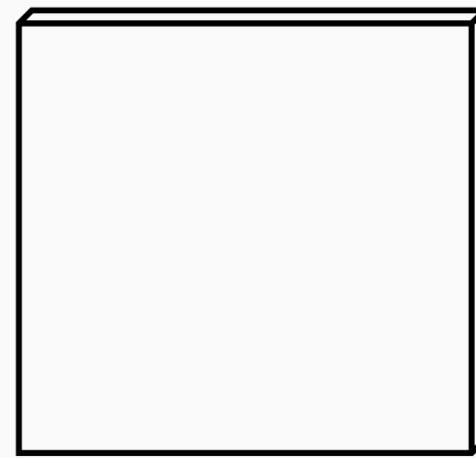
Input: $C_{in} \times T \times H \times W$
(3D grid with C_{in} -dim
feat at each point)



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



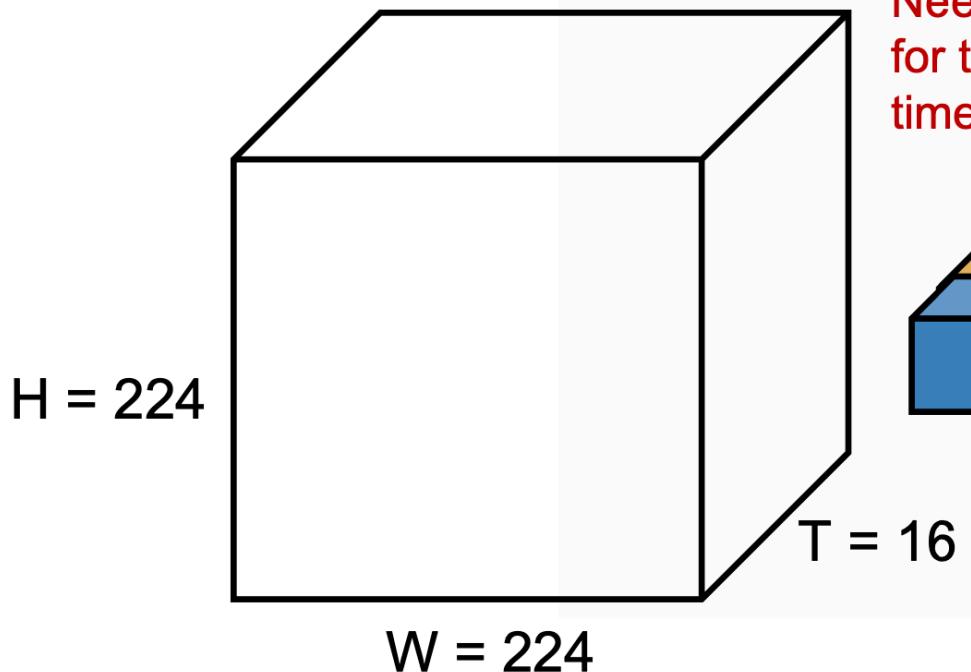
Output:
 $C_{out} \times H \times W$
2D grid with C_{out} -dim
feat at each point



Slide credit: Justin Johnson

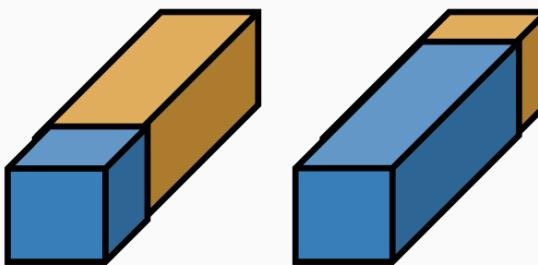
2D Conv (Early Fusion) vs 3D Conv (3D CNN)

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



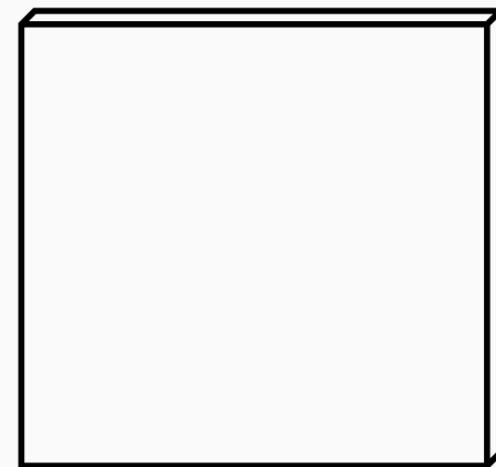
Weight:
 $C_{out} \times C_{in} \times T \times 3 \times 3$
Slide over x and y

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



C_{out} different filters

Output:
 $C_{out} \times H \times W$
2D grid with C_{out} -dim
feat at each point

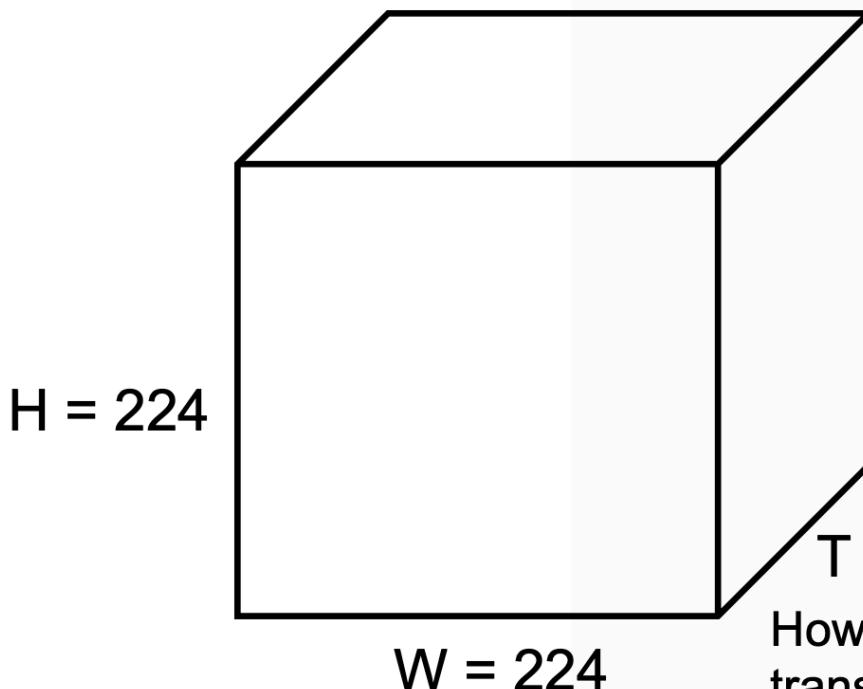


$W = 224$

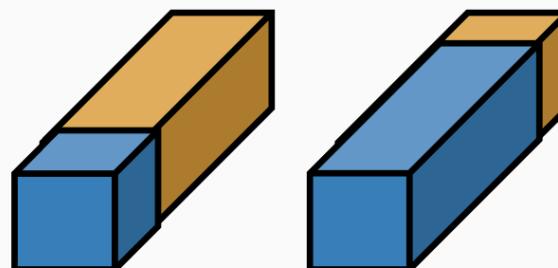
Slide credit: Justin Johnson

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

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



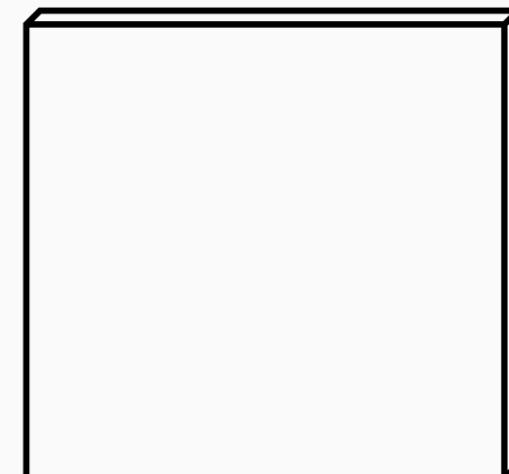
Weight:
 $C_{out} \times C_{in} \times T \times 3 \times 3$
Slide over x and y
No temporal shift-invariance!
Needs to learn separate filters
for the same motion at different
times in the clip



C_{out} different filters

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

Output:
 $C_{out} \times H \times W$
2D grid with C_{out} -dim
feat at each point

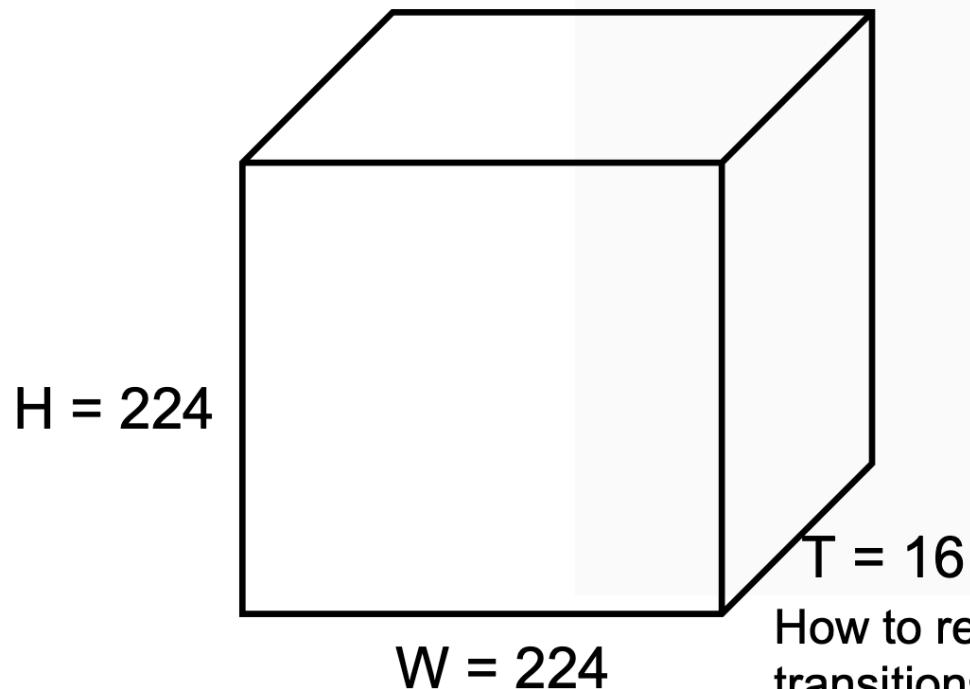


$W = 224$

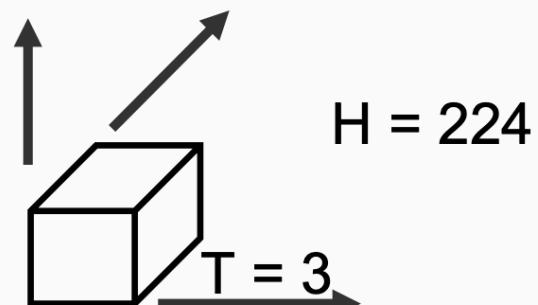
Slide credit: Justin Johnson

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

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

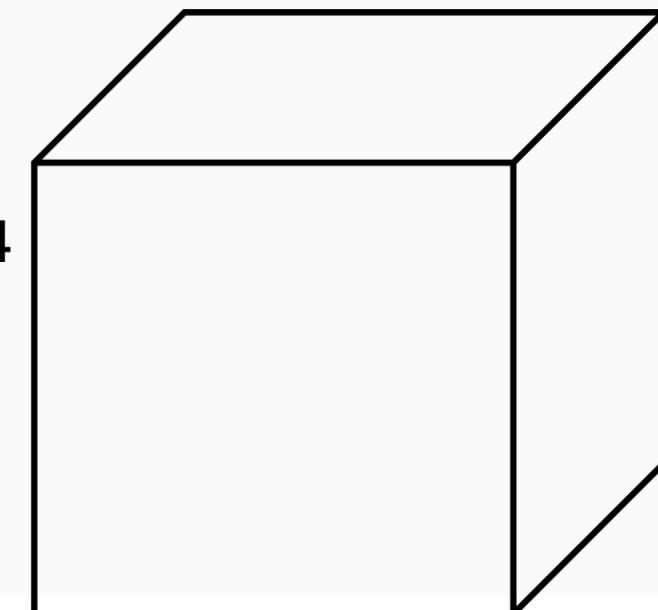


Weight:
 $C_{out} \times C_{in} \times 3 \times 3 \times 3$
Slide over x and y



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

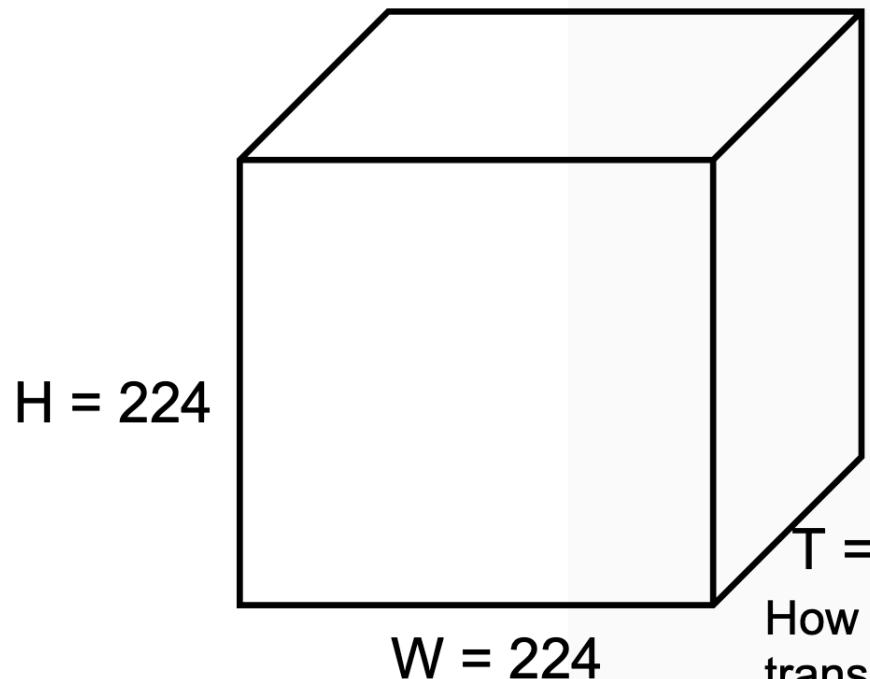
Output:
 $C_{out} \times T \times H \times W$
3D grid with C_{out} -dim
feat at each point



Slide credit: Justin Johnson

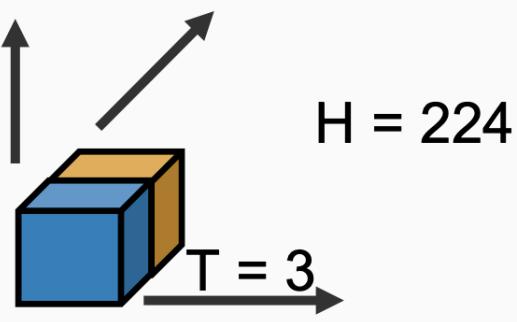
2D Conv (Early Fusion) vs 3D Conv (3D CNN)

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



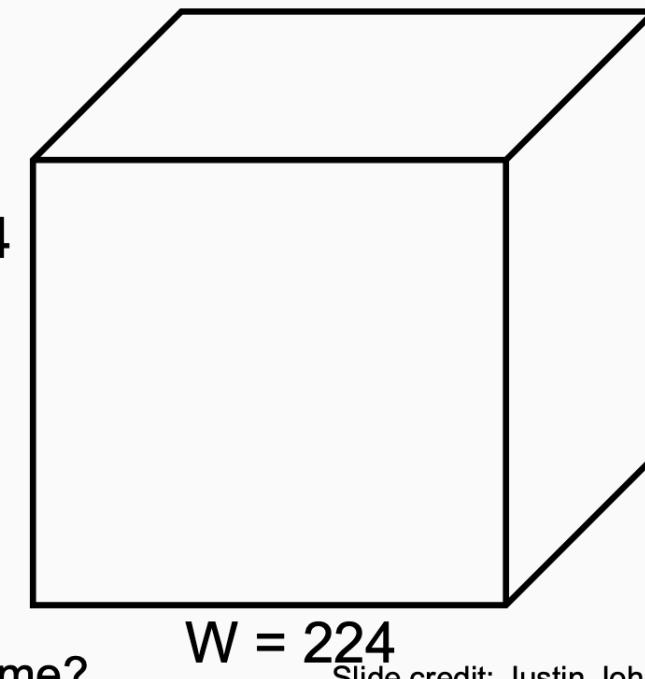
Weight:
 $C_{out} \times C_{in} \times 3 \times 3 \times 3$
Slide over x and y

Temporal shift-invariant
since each filter slides
over time!



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

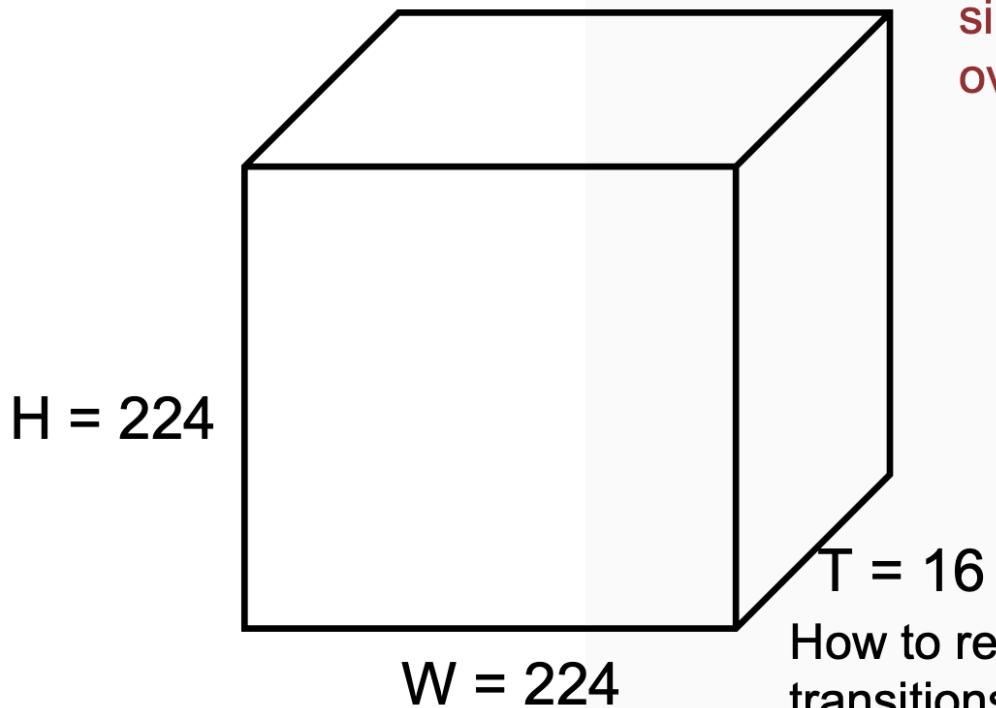
Output:
 $C_{out} \times T \times H \times W$
3D grid with C_{out} -dim
feat at each point



Slide credit: Justin Johnson

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

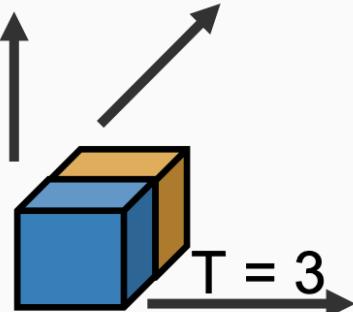
Input: $C_{in} \times T \times H \times W$
(3D grid with C_{in} -dim
feat at each point)



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

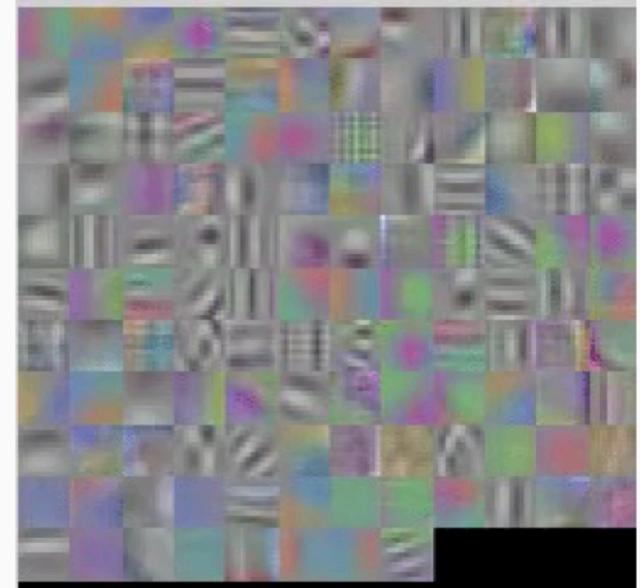
Weight:
 $C_{out} \times C_{in} \times 3 \times 3 \times 3$
Slide over x and y

Temporal shift-invariant
since each filter slides
over time!



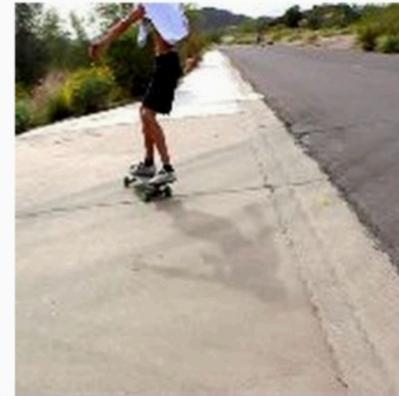
C_{out} different filters

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



Slide credit: Justin Johnson

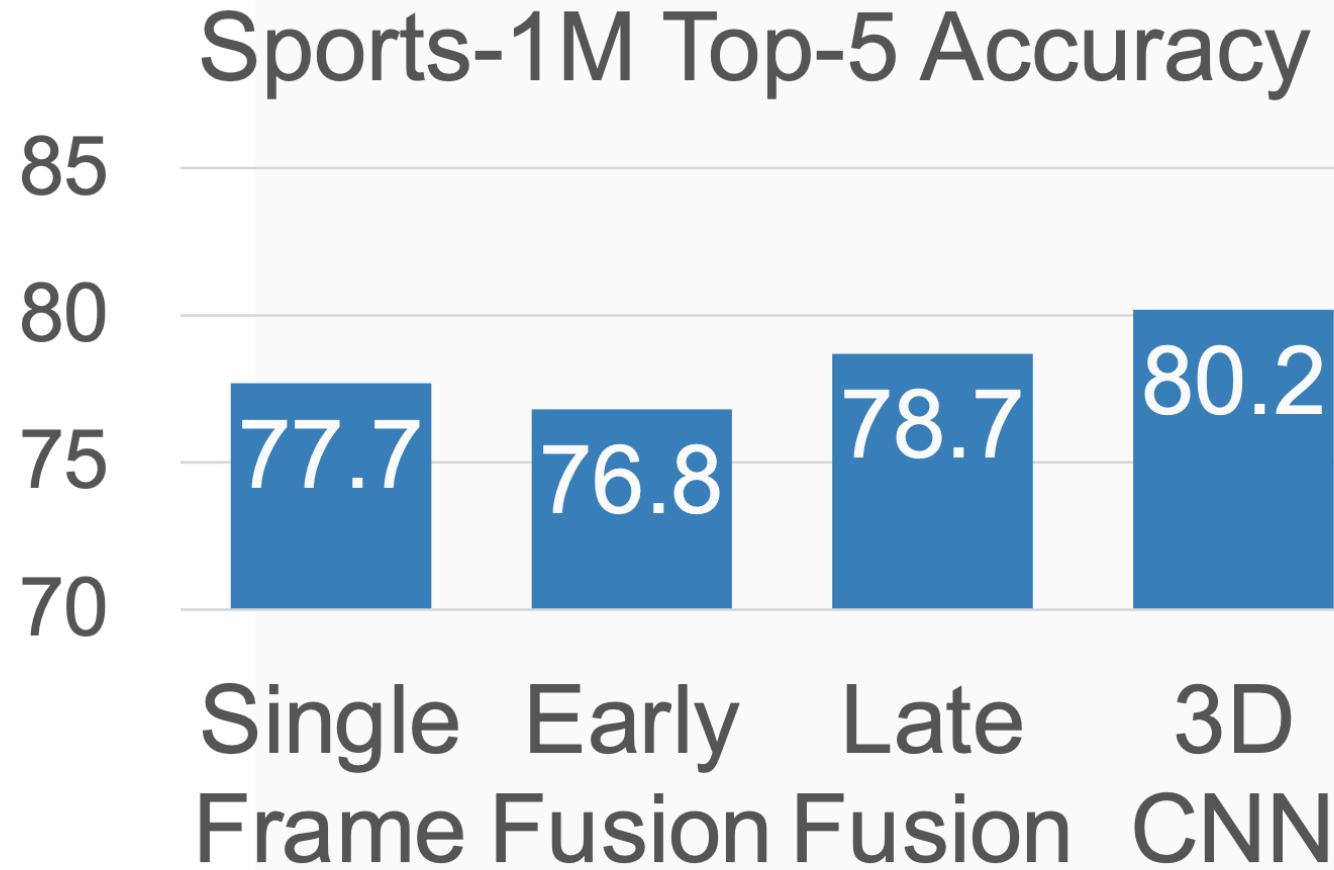
Example Video Dataset: Sports-1M



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!

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	$3 \times 20 \times 64 \times 64$	
Conv2D(3x3, 3->12)	$12 \times 20 \times 64 \times 64$	$1 \times 3 \times 3$
Pool2D(4x4)	$12 \times 20 \times 16 \times 16$	$1 \times 6 \times 6$
Conv2D(3x3, 12->24)	$24 \times 20 \times 16 \times 16$	$1 \times 14 \times 14$
GlobalAvgPool	$24 \times 1 \times 1 \times 1$	$20 \times 64 \times 64$

Early
Fusion

Input	$3 \times 20 \times 64 \times 64$	
Conv2D(3x3, 3*20->12)	$12 \times 64 \times 64$	$20 \times 3 \times 3$
Pool2D(4x4)	$12 \times 16 \times 16$	$20 \times 6 \times 6$
Conv2D(3x3, 12->24)	$24 \times 16 \times 16$	$20 \times 14 \times 14$
GlobalAvgPool	$24 \times 1 \times 1$	$20 \times 64 \times 64$

3D
CNN

Input	$3 \times 20 \times 64 \times 64$	
Conv3D(3x3x3, 3->12)	$12 \times 20 \times 64 \times 64$	$3 \times 3 \times 3$
Pool3D(4x4x4)	$12 \times 5 \times 16 \times 16$	$6 \times 6 \times 6$
Conv3D(3x3x3, 12->24)	$24 \times 5 \times 16 \times 16$	$14 \times 14 \times 14$
GlobalAvgPool	$24 \times 1 \times 1$	$20 \times 64 \times 64$

What is the difference?

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

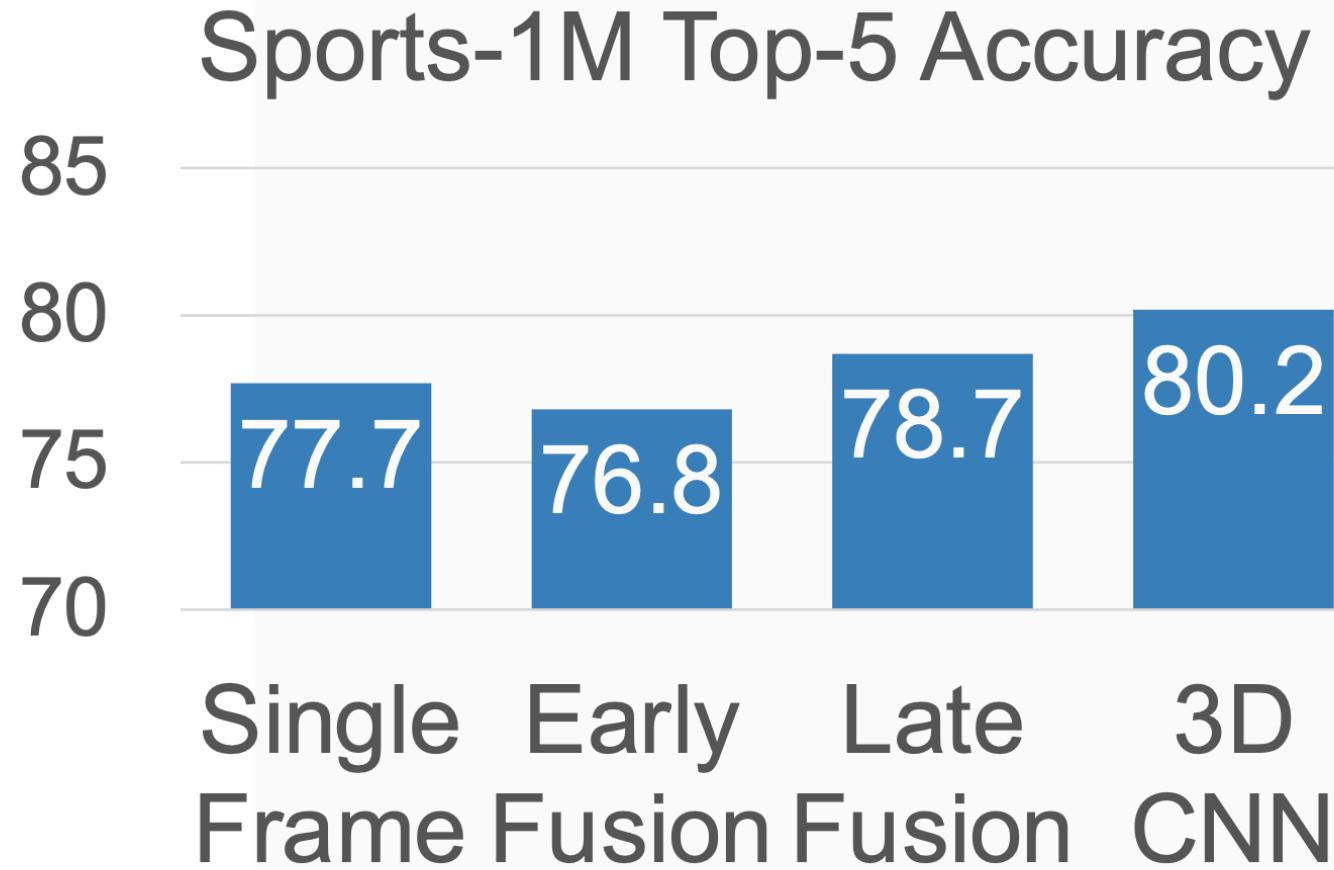
Build slowly in space,
All-at-once in time at start

Build slowly in space,
Build slowly in time
"Slow Fusion"

(Small example architectures, in practice much bigger)

Slide credit: Justin Johnson

Early Fusion vs. Late Fusion vs. 3DCNN



Single Frame
model works well
– always try this
first!

3D CNNs have
improved a lot
since 2014!

C3D: The VGG of 3D CNNs

3D CNN that uses all 3x3x3 conv
and 2x2x2 pooling
(except Pool1 which is 1x2x2)

Released model pretrained on
Sports-1M: Many people used this
as a video feature extractor

Layer	Size
Input	3 x 16 x 112 x 112
Conv1 (3x3x3)	64 x 16 x 112 x 112
Pool1 (1x2x2)	64 x 16 x 56 x 56
Conv2 (3x3x3)	128 x 16 x 56 x 56
Pool2 (2x2x2)	128 x 8 x 28 x 28
Conv3a (3x3x3)	256 x 8 x 28 x 28
Conv3b (3x3x3)	256 x 8 x 28 x 28
Pool3 (2x2x2)	256 x 4 x 14 x 14
Conv4a (3x3x3)	512 x 4 x 14 x 14
Conv4b (3x3x3)	512 x 4 x 14 x 14
Pool4 (2x2x2)	512 x 2 x 7 x 7
Conv5a (3x3x3)	512 x 2 x 7 x 7
Conv5b (3x3x3)	512 x 2 x 7 x 7
Pool5	512 x 1 x 3 x 3
FC6	4096
FC7	4096
FC8	C

C3D: The VGG of 3D CNNs

3D CNN that uses all 3x3x3 conv
and 2x2x2 pooling
(except Pool1 which is 1x2x2)

Released model pretrained on
Sports-1M: Many people used this
as a video feature extractor

Problem: 3x3x3 conv is very
expensive!

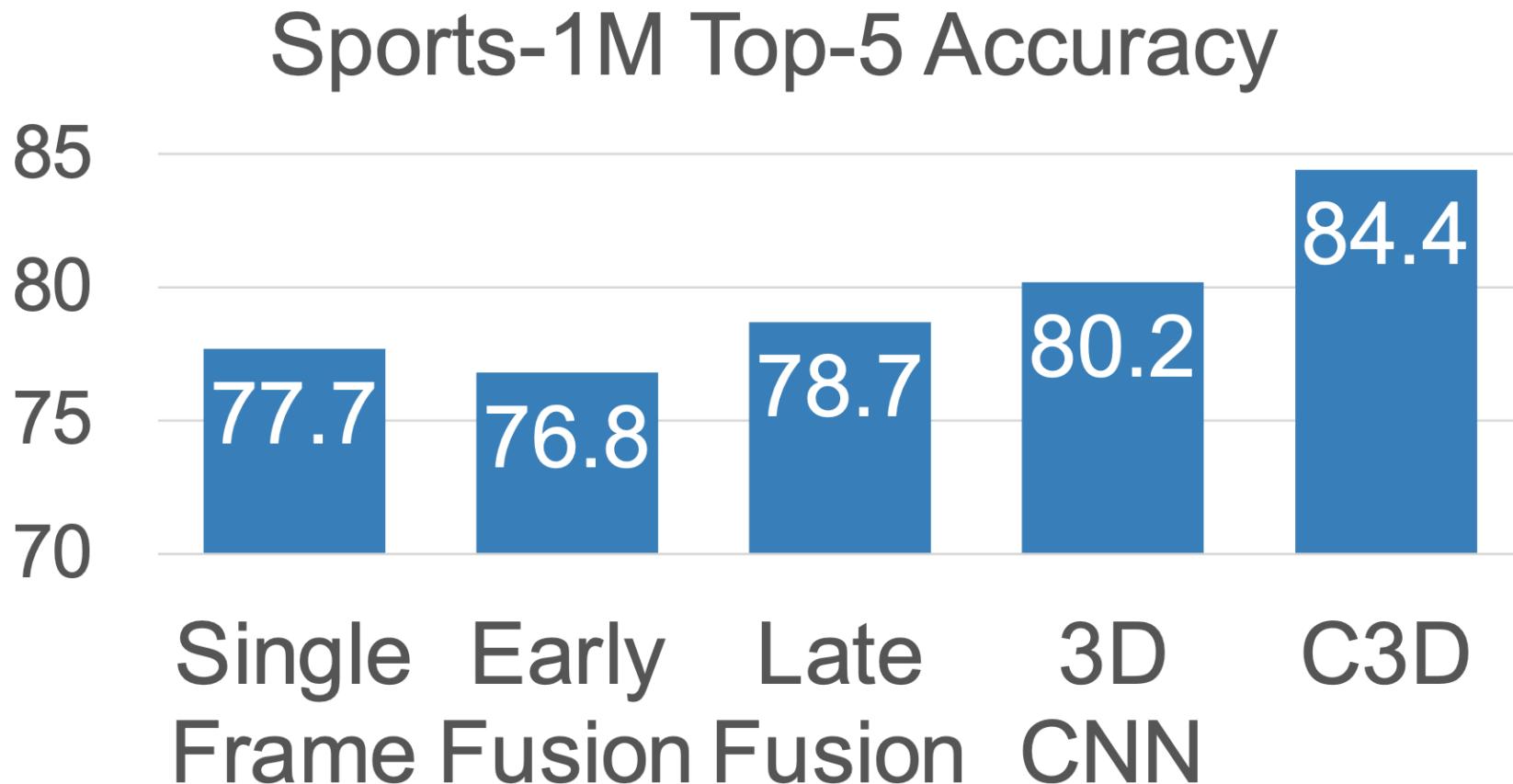
AlexNet: 0.7 GFLOP

VGG-16: 13.6 GFLOP

C3D: **39.5 GFLOP (2.9x VGG!)**

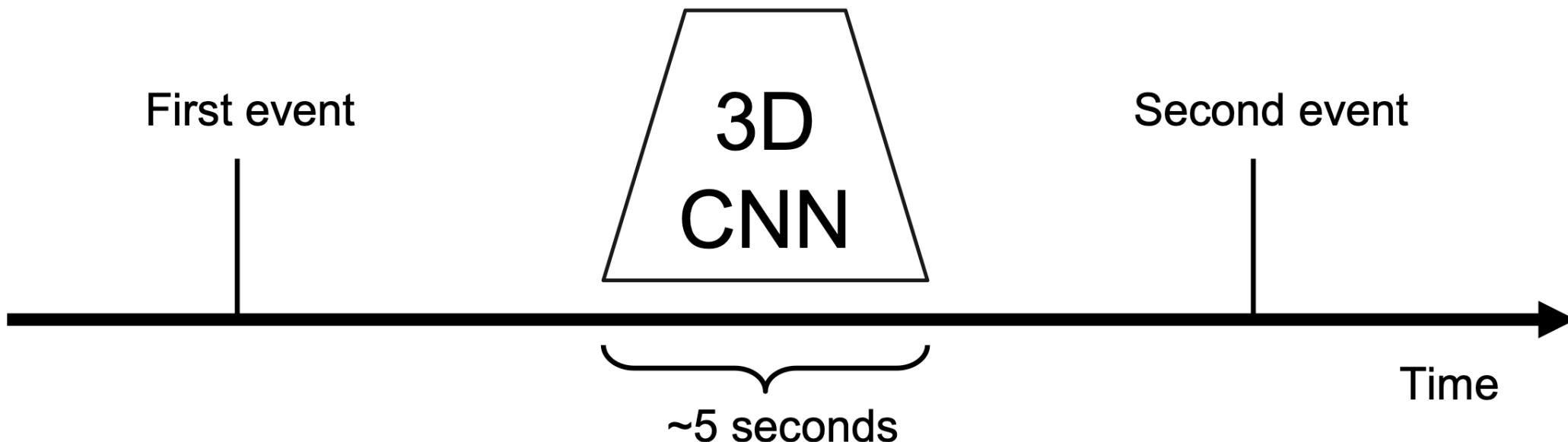
Layer	Size	MFLOPs
Input	$3 \times 16 \times 112 \times 112$	
Conv1 (3x3x3)	$64 \times 16 \times 112 \times 112$	1.04
Pool1 (1x2x2)	$64 \times 16 \times 56 \times 56$	
Conv2 (3x3x3)	$128 \times 16 \times 56 \times 56$	11.10
Pool2 (2x2x2)	$128 \times 8 \times 28 \times 28$	
Conv3a (3x3x3)	$256 \times 8 \times 28 \times 28$	5.55
Conv3b (3x3x3)	$256 \times 8 \times 28 \times 28$	11.10
Pool3 (2x2x2)	$256 \times 4 \times 14 \times 14$	
Conv4a (3x3x3)	$512 \times 4 \times 14 \times 14$	2.77
Conv4b (3x3x3)	$512 \times 4 \times 14 \times 14$	5.55
Pool4 (2x2x2)	$512 \times 2 \times 7 \times 7$	
Conv5a (3x3x3)	$512 \times 2 \times 7 \times 7$	0.69
Conv5b (3x3x3)	$512 \times 2 \times 7 \times 7$	0.69
Pool5	$512 \times 1 \times 3 \times 3$	
FC6	4096	0.51
FC7	4096	0.45
FC8	C	0.05

Early Fusion vs. Late Fusion vs. 3DCNN



Modeling Long-Term Temporal Structure

So far all our temporal CNNs only model local motion between frames in very short clips of ~2-5 seconds. What about long-term structure?

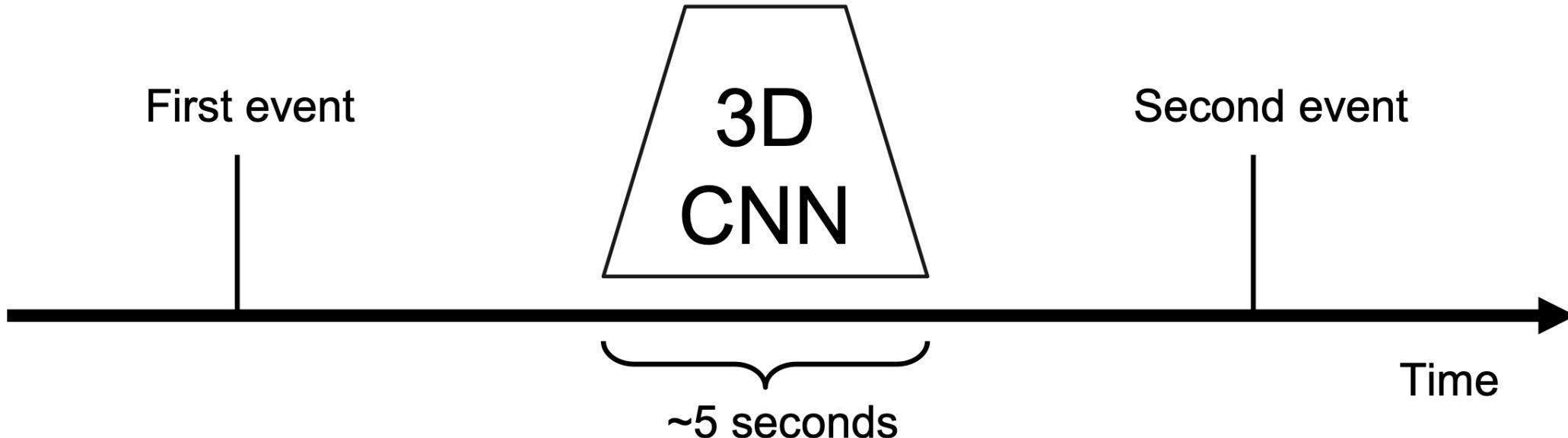


Slide credit: Justin Johnson

Modeling Long-Term Temporal Structure

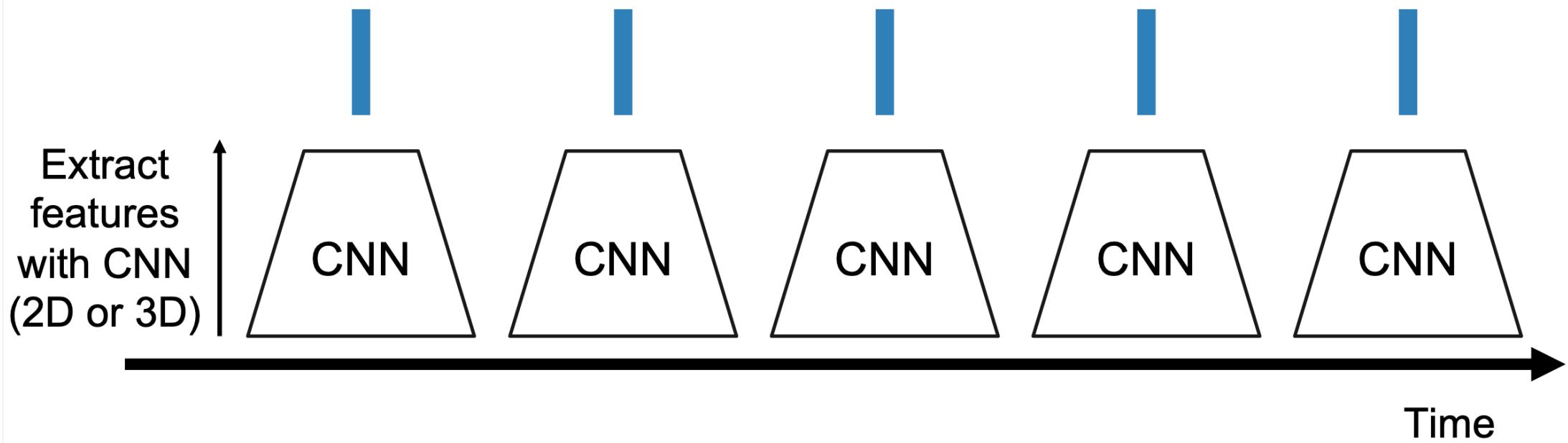
So far all our temporal CNNs only model local motion between frames in very short clips of ~2-5 seconds. What about long-term structure?

We know how to handle sequences! How about recurrent networks?



Slide credit: Justin Johnson

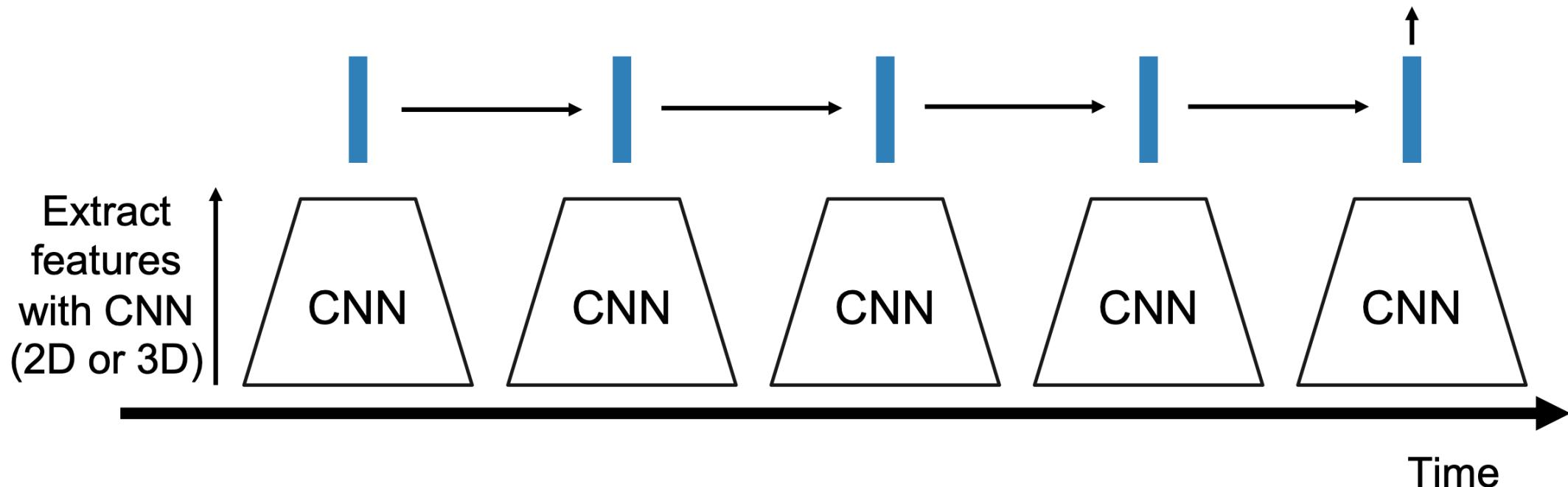
Modeling Long-Term Temporal Structure



Slide credit: Justin Johnson

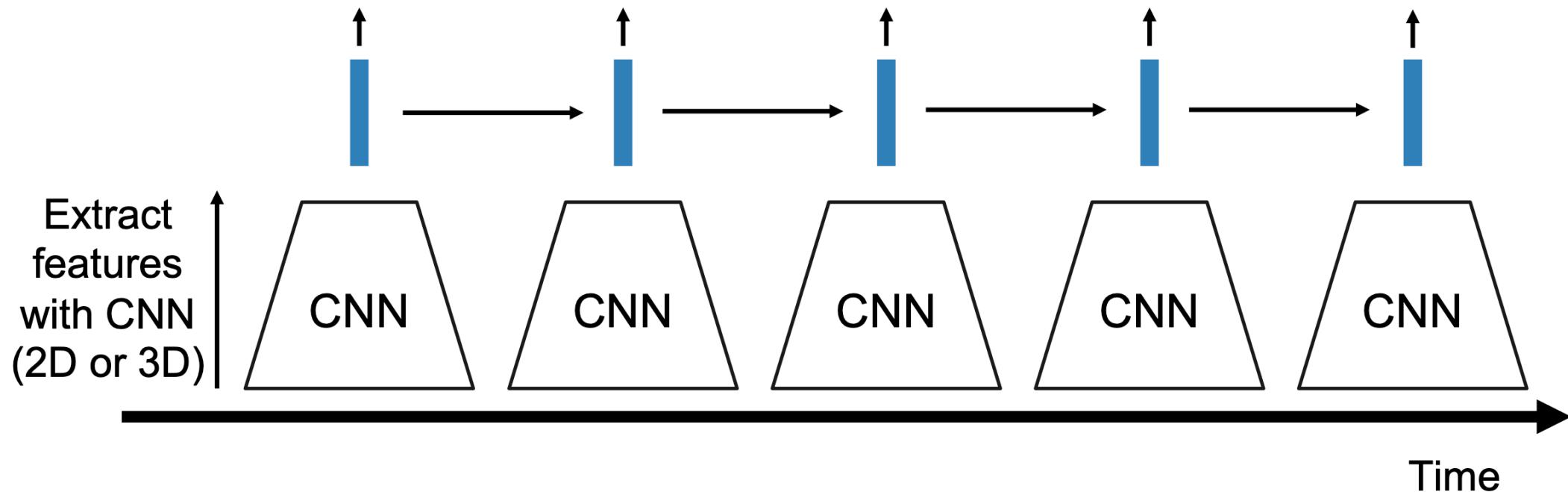
Modeling Long-Term Temporal Structure

Process local features using recurrent network (e.g. LSTM)
Many to one: One output at end of video



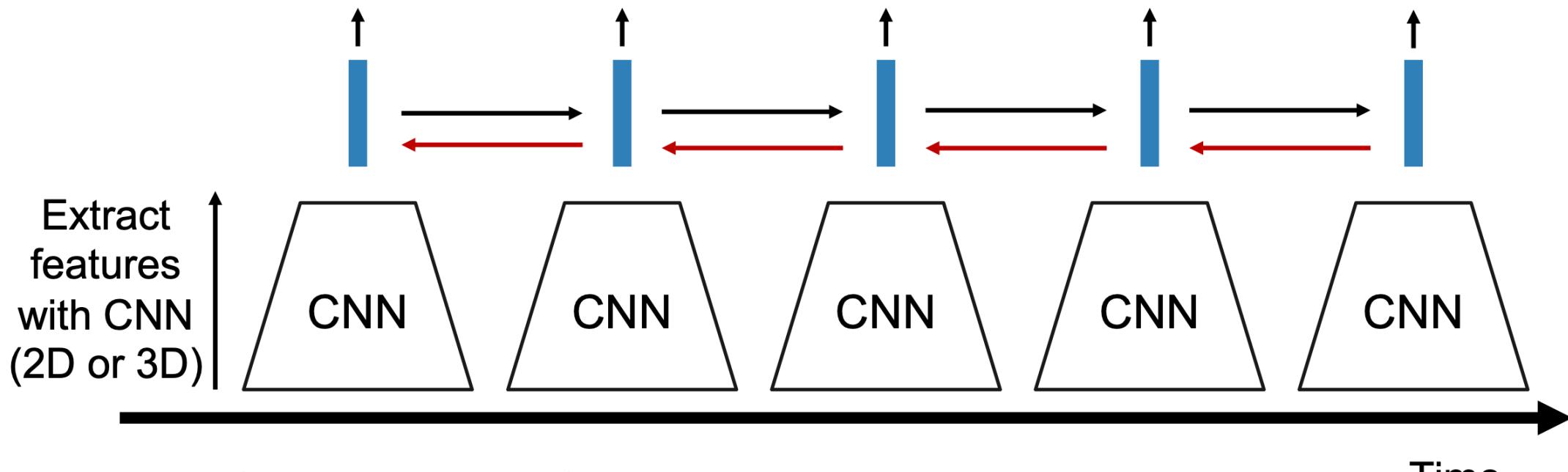
Modeling Long-Term Temporal Structure

Process local features using recurrent network (e.g. LSTM)
Many to many: one output per video frame



Modeling Long-Term Temporal Structure

Sometimes don't backprop to CNN to save memory; pretrain and use it as a feature extractor



Baccouche et al, "Sequential Deep Learning for Human Action Recognition", 2011

Donahue et al, "Long-term recurrent convolutional networks for visual recognition and description", CVPR 2015

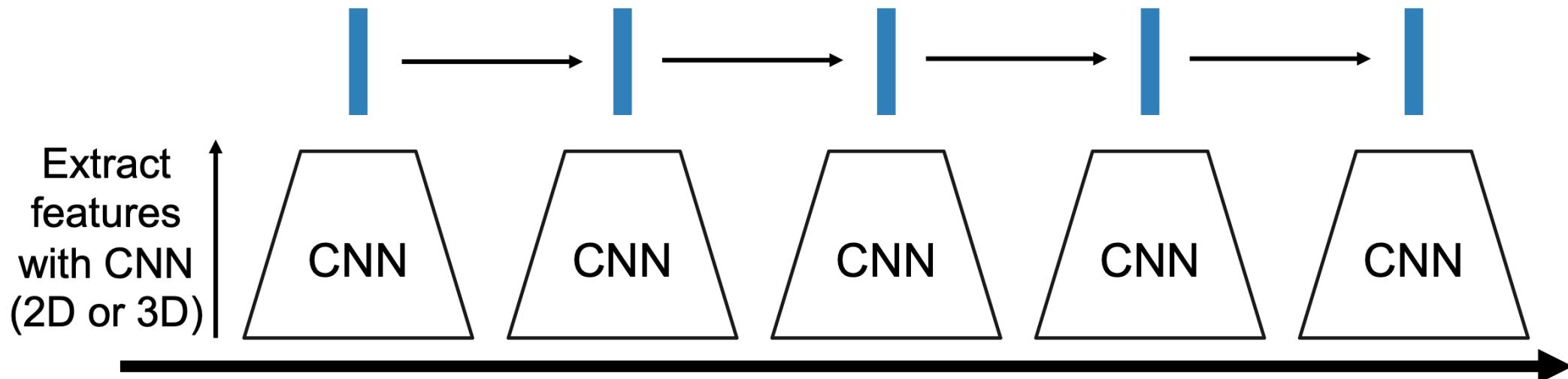
Slide credit: Justin Johnson

Modeling Long-Term Temporal Structure

Inside CNN: Each value is a function of a fixed temporal window (local temporal structure)

Inside RNN: Each vector is a function of all previous vectors (global temporal structure)

Can we merge both approaches?

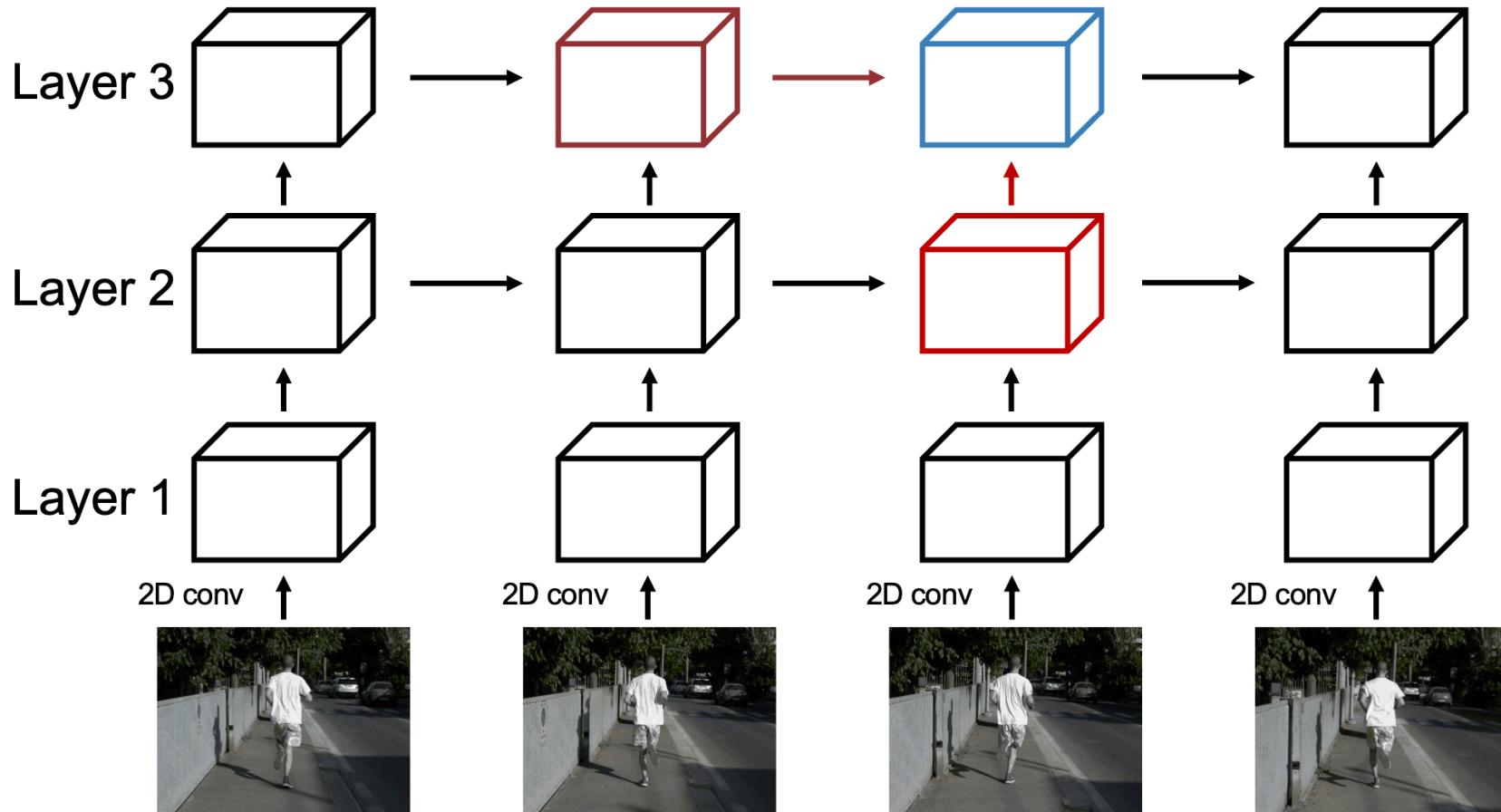


Baccouche et al, "Sequential Deep Learning for Human Action Recognition", 2011

Donahue et al, "Long-term recurrent convolutional networks for visual recognition and description", CVPR 2015

Slide credit: Justin Johnson

Recurrent Convolutional Network



Entire network
uses 2D
feature maps:
 $C \times H \times W$

Each depends
on two inputs:
**1. Same layer,
previous
timestep**
**2. Prev layer,
same timestep**

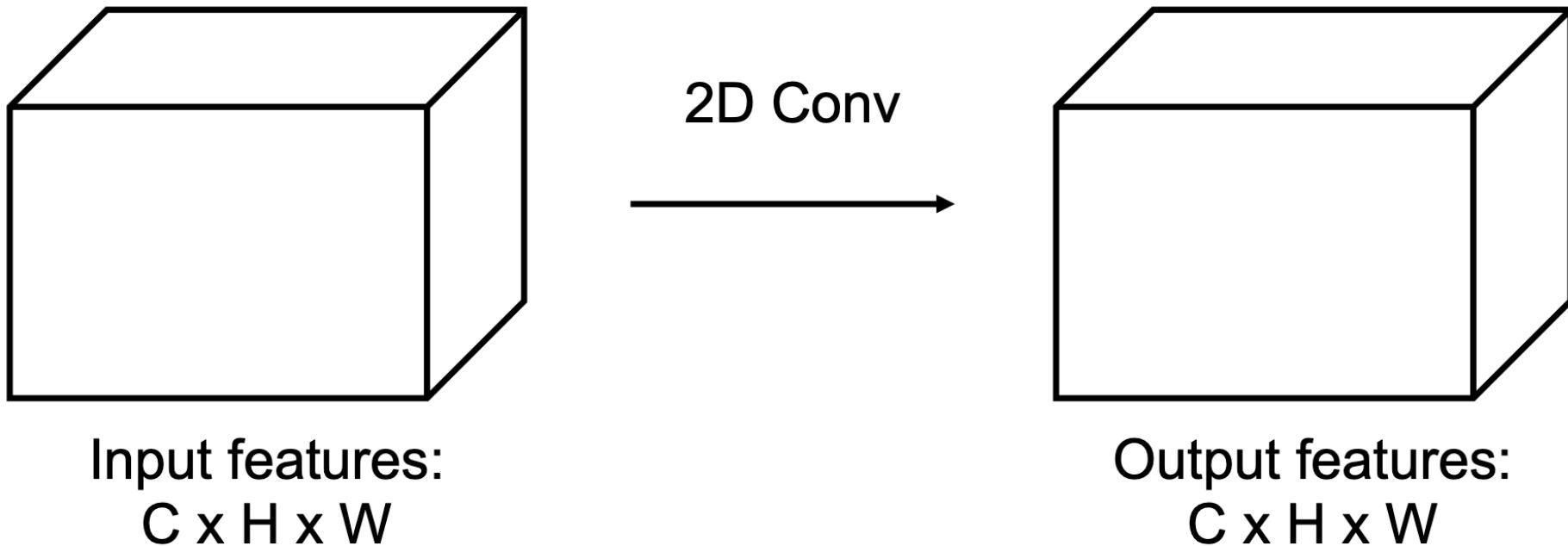
Use different weight
at each layer, share
weights across time

Ballas et al, "Delving Deeper into
Convolutional Networks for Learning
Video Representations", ICLR 2016

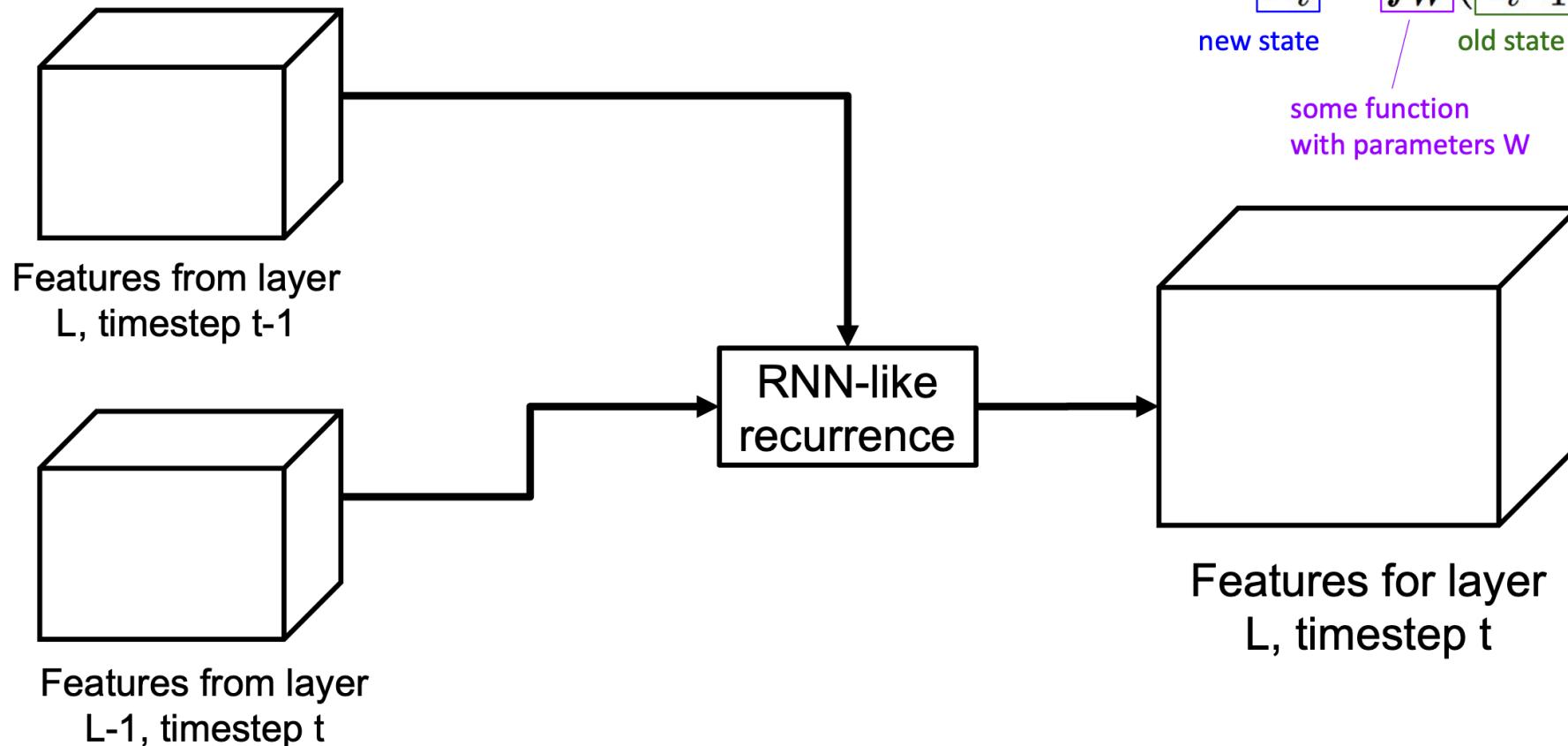
Slide credit: Justin Johnson

Recurrent Convolutional Network

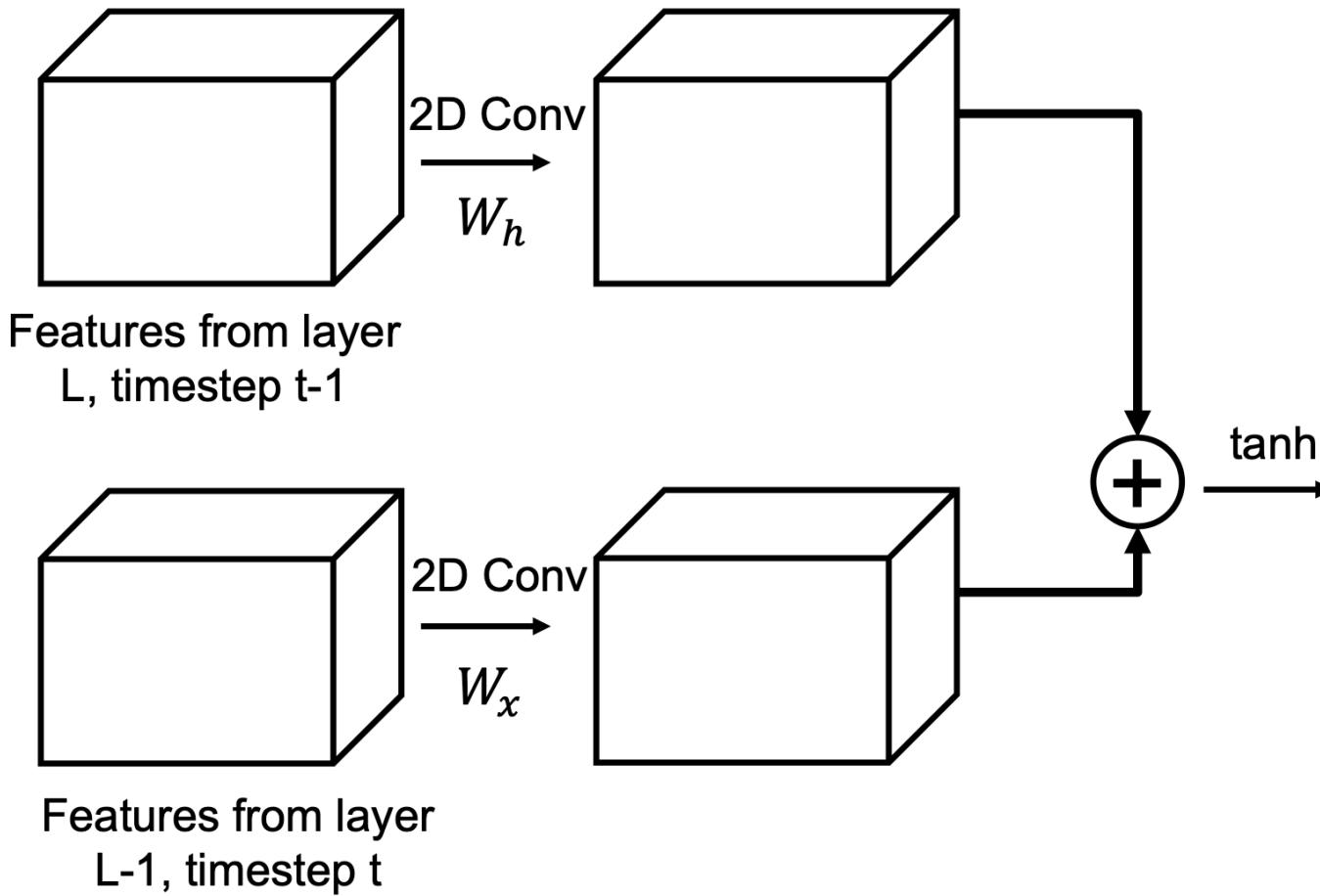
Normal 2D CNN:



Recurrent Convolutional Network



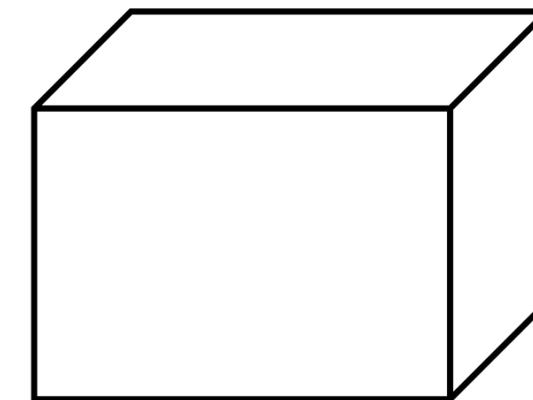
Recurrent Convolutional Network



Recall: Vanilla RNN

$$h_{t+1} = \tanh(W_h h_t + W_x x)$$

Replace all matrix multiply
with 2D convolution!



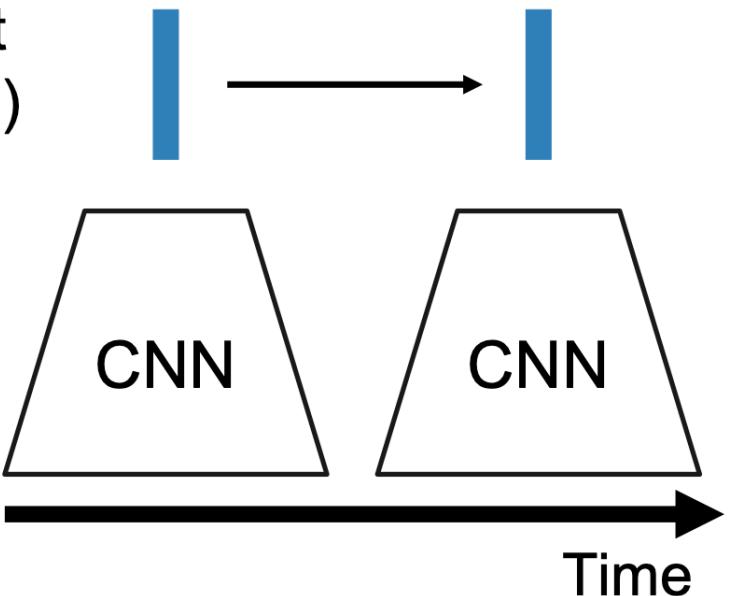
Features for layer
L, timestep t

Modeling Long-Term Temporal Structure

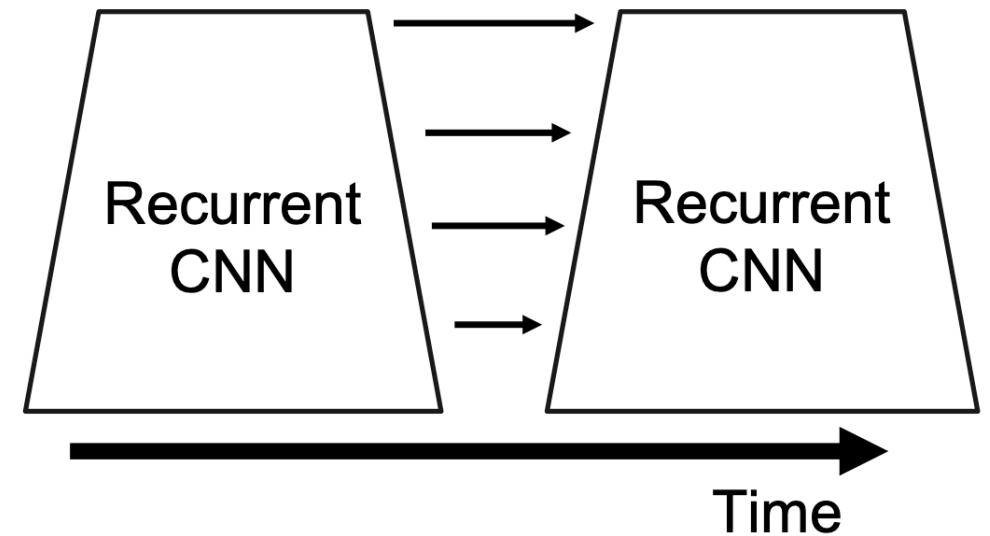
Problem: RNNs are slow for long sequences (can't be parallelized)

RNN: Infinite temporal extent (fully-connected)

CNN: finite temporal extent (convolutional)



Recurrent CNN: Infinite temporal extent (convolutional)



Baccouche et al, "Sequential Deep Learning for Human Action Recognition", 2011
Donahue et al, "Long-term recurrent convolutional networks for visual recognition and description", CVPR 2015

Ballas et al, "Delving Deeper into Convolutional Networks for Learning Video Representations", ICLR 2016

Slide credit: Justin Johnson

A lot of Variations

- ConvLSTM
- ConvGRU
- Direction:
 - Attention
 - Transformer
 - ...



Introduction to Computer Vision

Next week: Lecture 12,
Guest Lecture on Transformer