

# **CS 4644-DL / 7643-A: LECTURE 12**

## **DANFEI XU**

Recurrent Neural Networks (RNN)

Long Short-Term Memory (LSTM)

# Recap: Second-Order Optimization

second-order Taylor expansion:

$$f(\mathbf{x}) = f(\mathbf{a}) + (\mathbf{x} - \mathbf{a})^T \nabla f + \frac{1}{2} (\mathbf{x} - \mathbf{a})^T H (\mathbf{x} - \mathbf{a})$$

Solving for the critical point we obtain the Newton parameter update:

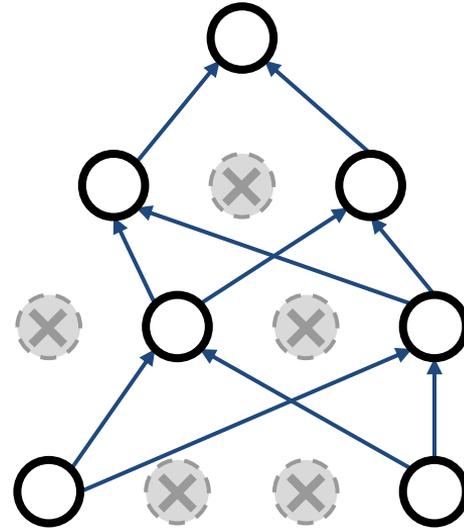
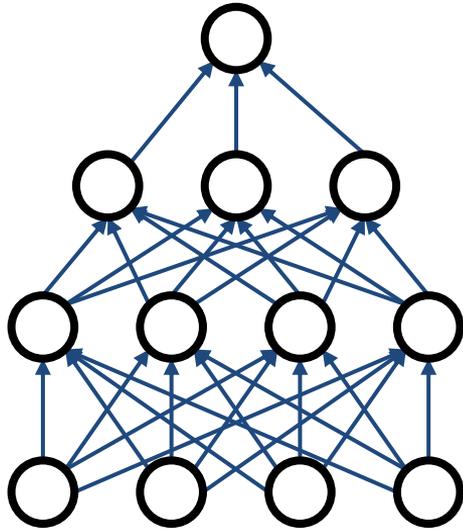
$$\mathbf{x}^* = \mathbf{a} - H^{-1} \nabla f$$

Hessian has  $O(N^2)$  elements  
Inverting takes  $O(N^3)$   
N = Millions

Q: Why is this bad for deep learning?

# Regularization: Dropout

In each forward pass, randomly set some neurons to zero  
Probability of dropping is a hyperparameter; 0.5 is common

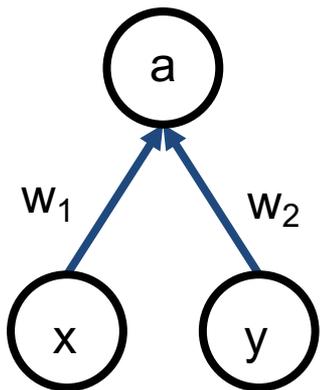


# Dropout: Test time

Compute the expectation

$$y = f(x) = E_z [f(x, z)] = \int p(z) f(x, z) dz$$

Consider a single neuron.



Without dropout:

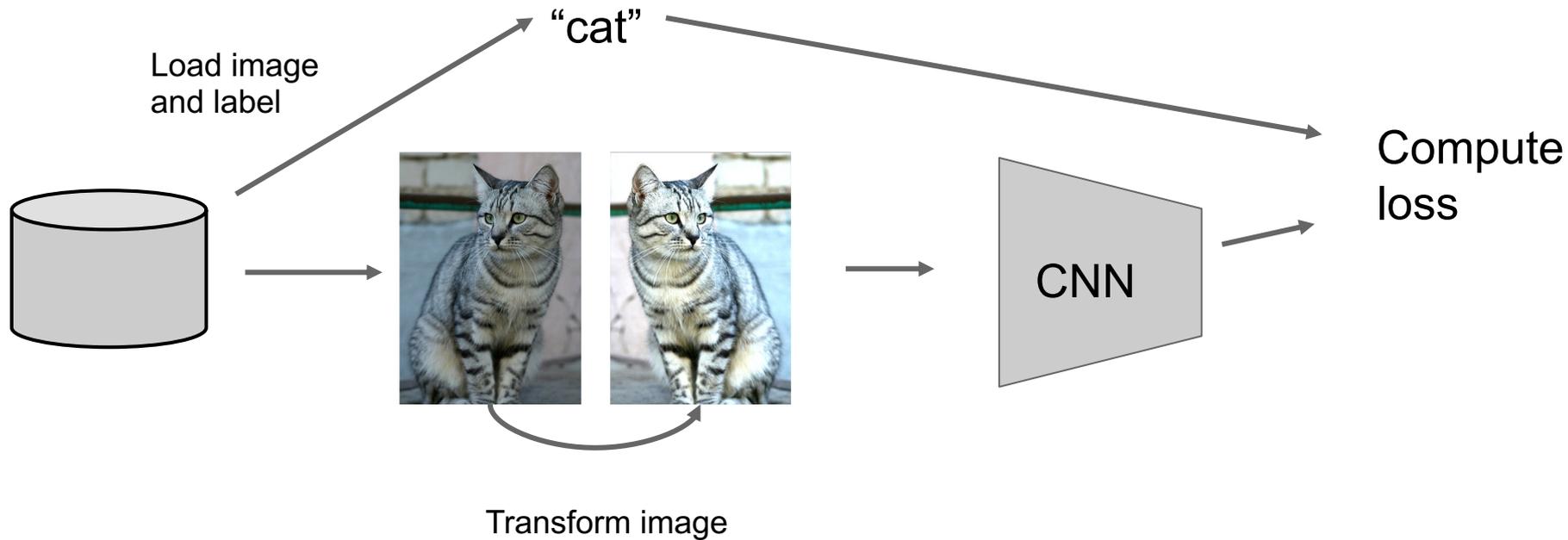
$$E[a] = w_1 x + w_2 y$$

With dropout we have:

$$\begin{aligned} E[a] &= \frac{1}{4}(w_1 x + w_2 y) + \frac{1}{4}(w_1 x + 0y) \\ &\quad + \frac{1}{4}(0x + 0y) + \frac{1}{4}(0x + w_2 y) \\ &= \frac{1}{2}(w_1 x + w_2 y) \end{aligned}$$

**At test time, multiply  
by dropout probability**

# Regularization: Data Augmentation



# Gradient clipping: prevent large gradient step

Large gradient step will likely destabilize training (gradients are noisy!)

Large gradient update can be caused by many issues, e.g., large weights, large input, bad loss function / activation function, ...

Should always first try to fix the root cause (normalization, better loss / activation function, etc.)

But if all things fail ... just clip the gradient

$$g_{new} = \min\left(1, \frac{\lambda}{\|g\|}\right) \times g$$

$g$ : original gradient

$\lambda$ : clipping threshold

```
# Zero the gradients.
optimizer.zero_grad()

# Perform forward pass.
outputs = model(inputs)

# Compute the loss.
loss = loss_function(outputs, targets)

# Perform backward pass (compute gradients).
loss.backward()

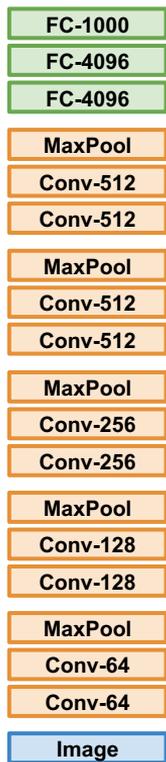
# Clip the gradients.
torch.nn.utils.clip_grad_norm_(model.parameters(), max_norm=1.0)

# Update the model parameters.
optimizer.step()
```

# Transfer Learning with CNNs

Donahue et al, "DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition", ICML 2014  
Razavian et al, "CNN Features Off-the-Shelf: An Astounding Baseline for Recognition", CVPR Workshops 2014

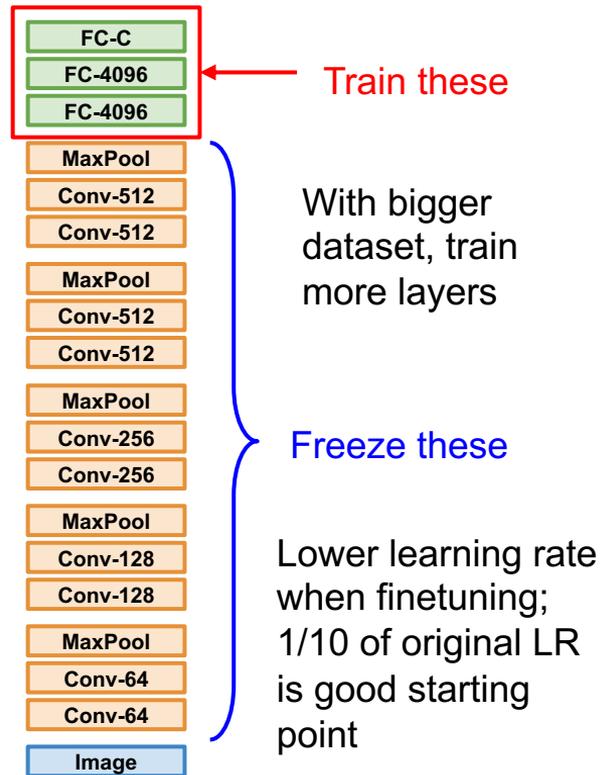
## 1. Train on Imagenet



## 2. Small Dataset (C classes)

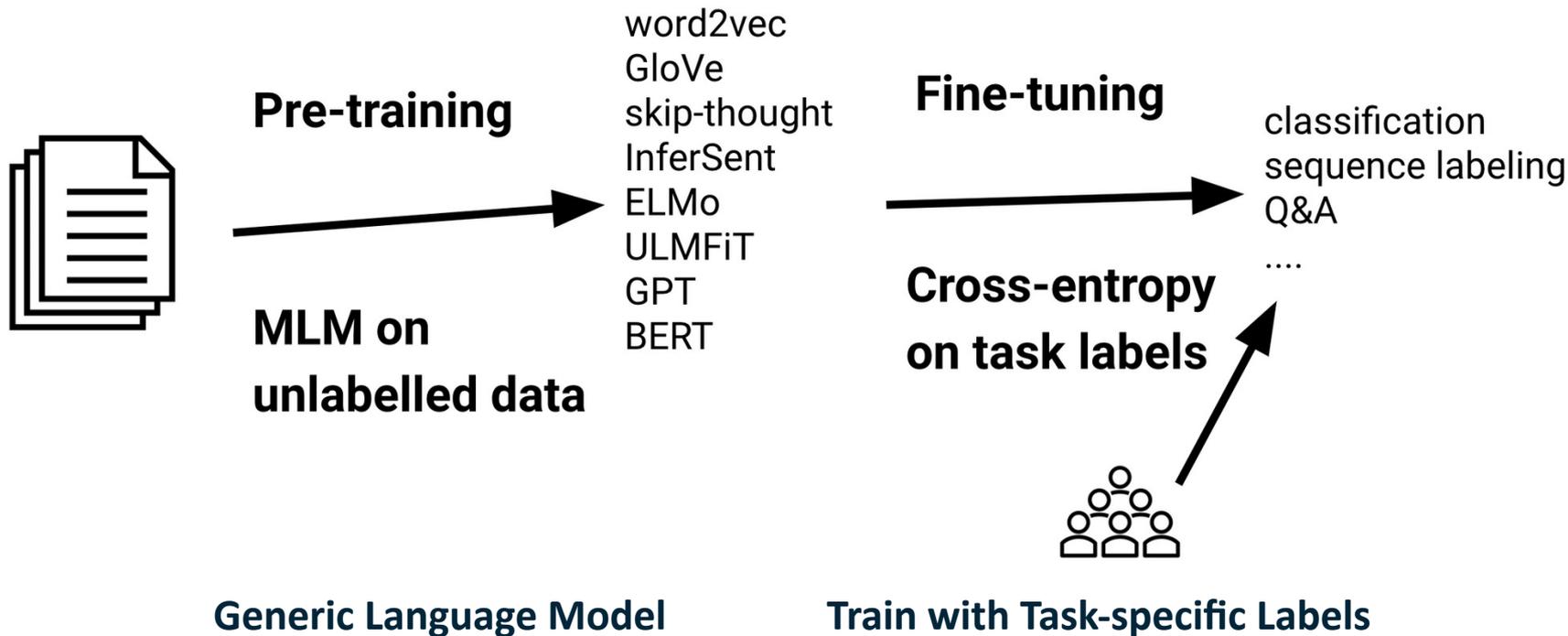


## 3. Bigger dataset



# Transfer learning is pervasive...

(it's the norm, not an exception)



# Diagnose your training

(without tons of GPUs)

# Diagnose your training

## Step 1: Check initial loss

Turn off weight decay, sanity check loss at initialization  
e.g.  $\log(C)$  for softmax with  $C$  classes

Reminder:  $L = -\log p = -\log(1/C) = \log(C)$

# Diagnose your training

**Step 1:** Check initial loss

**Step 2:** **Overfit a small sample**

Try to train to 100% training accuracy on a small sample of training data (~5-10 minibatches); fiddle with architecture, learning rate, weight initialization

Loss not going down? LR too low, bad initialization, bug in code or errors in training labels

Loss explodes to Inf or NaN? LR too high, bad initialization, bug in code

# Diagnose your training

**Step 1:** Check initial loss

**Step 2:** Overfit a small sample

**Step 3:** Find LR that makes loss go down

Use the architecture from the previous step, use all training data, turn on small weight decay, find a learning rate that makes the loss drop significantly within  $\sim 100$  iterations

Good learning rates to try:  $1e-3$ ,  $3e-4$ ,  $1e-4$

# Diagnose your training

**Step 1:** Check initial loss

**Step 2:** Overfit a small sample

**Step 3:** Find LR that makes loss go down

**Step 4:** Coarse grid, train for ~1-5 epochs

Choose a few values of learning rate and weight decay around what worked from Step 3, train a few models for ~1-5 epochs.

Good weight decay to try:  $1e-4$ ,  $1e-5$ , 0

# Diagnose your training

**Step 1:** Check initial loss

**Step 2:** Overfit a small sample

**Step 3:** Find LR that makes loss go down

**Step 4:** Coarse grid, train for ~1-5 epochs

**Step 5:** Refine grid, train longer

Pick best models from Step 4, train them for longer (~10-20 epochs) without learning rate decay

# Diagnose your training

**Step 1:** Check initial loss

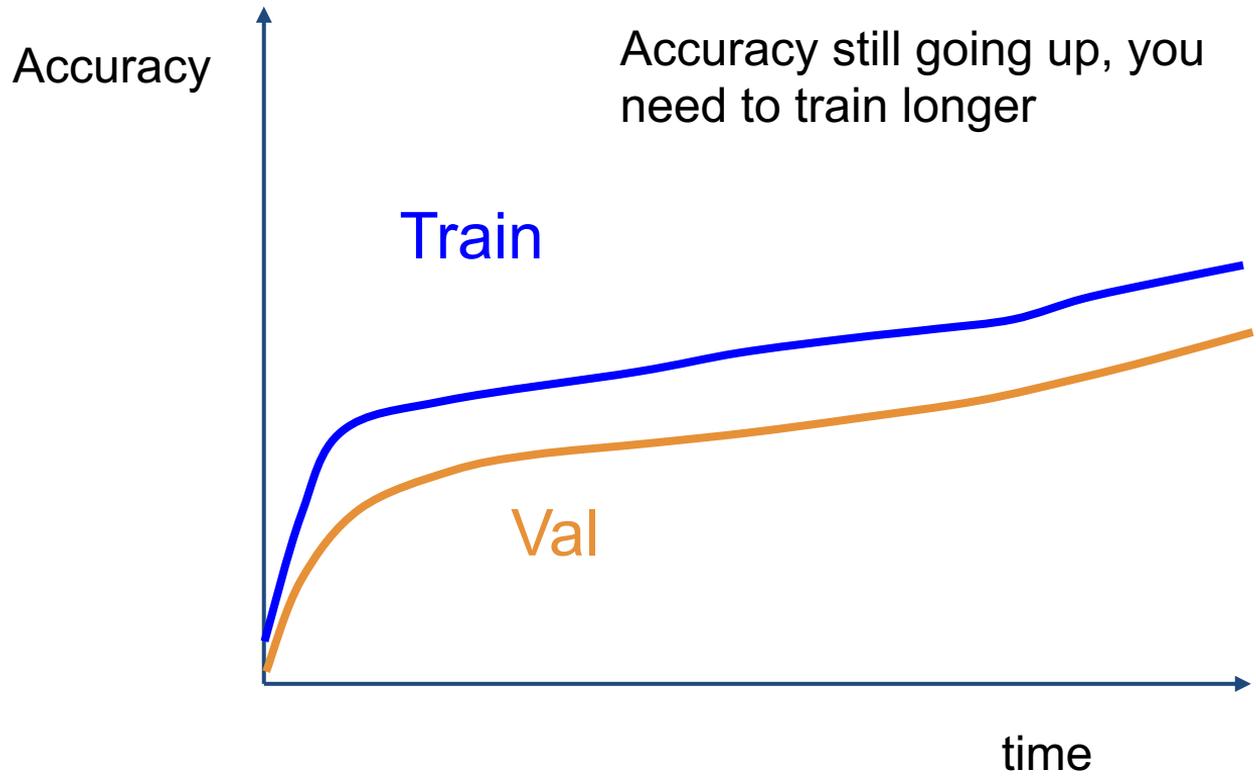
**Step 2:** Overfit a small sample

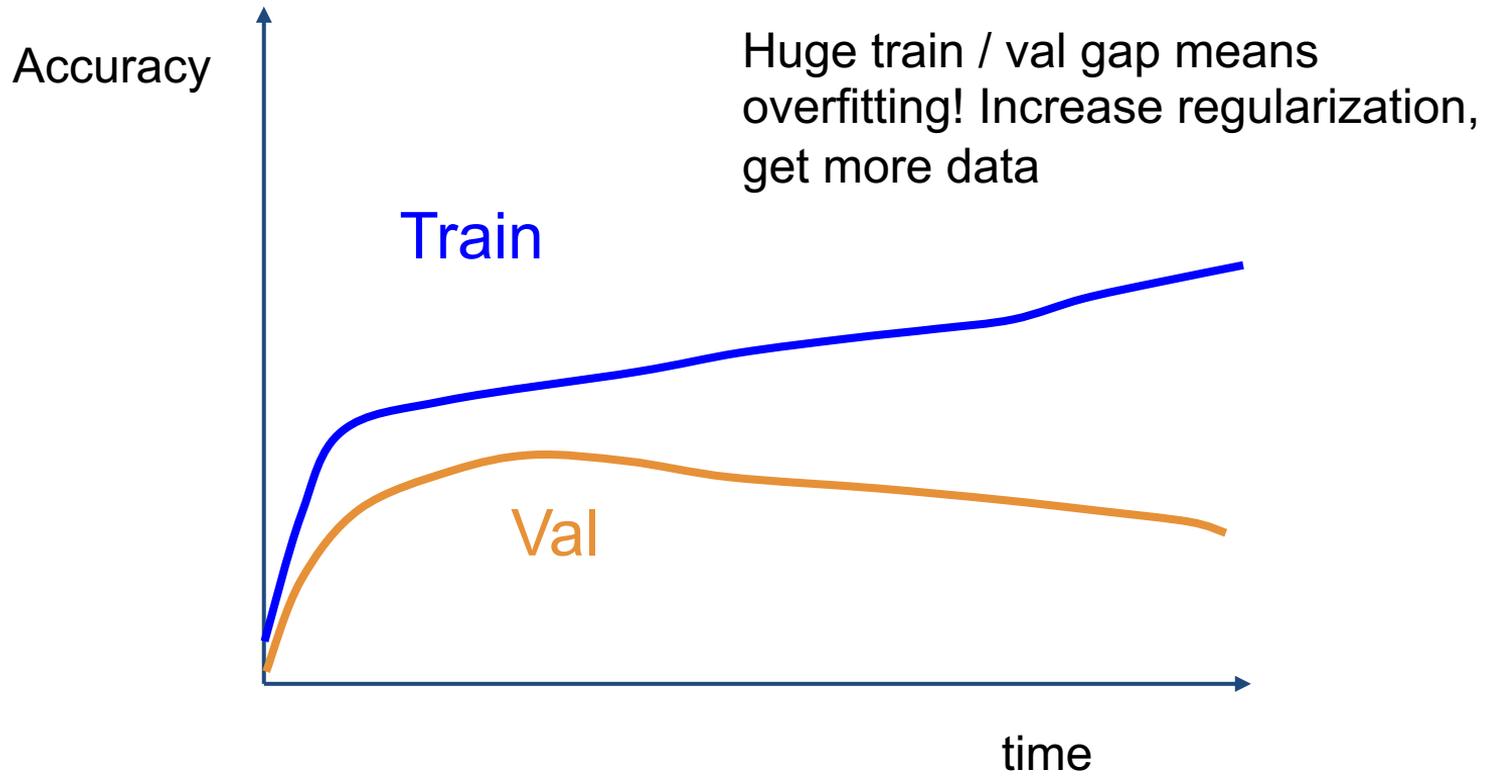
**Step 3:** Find LR that makes loss go down

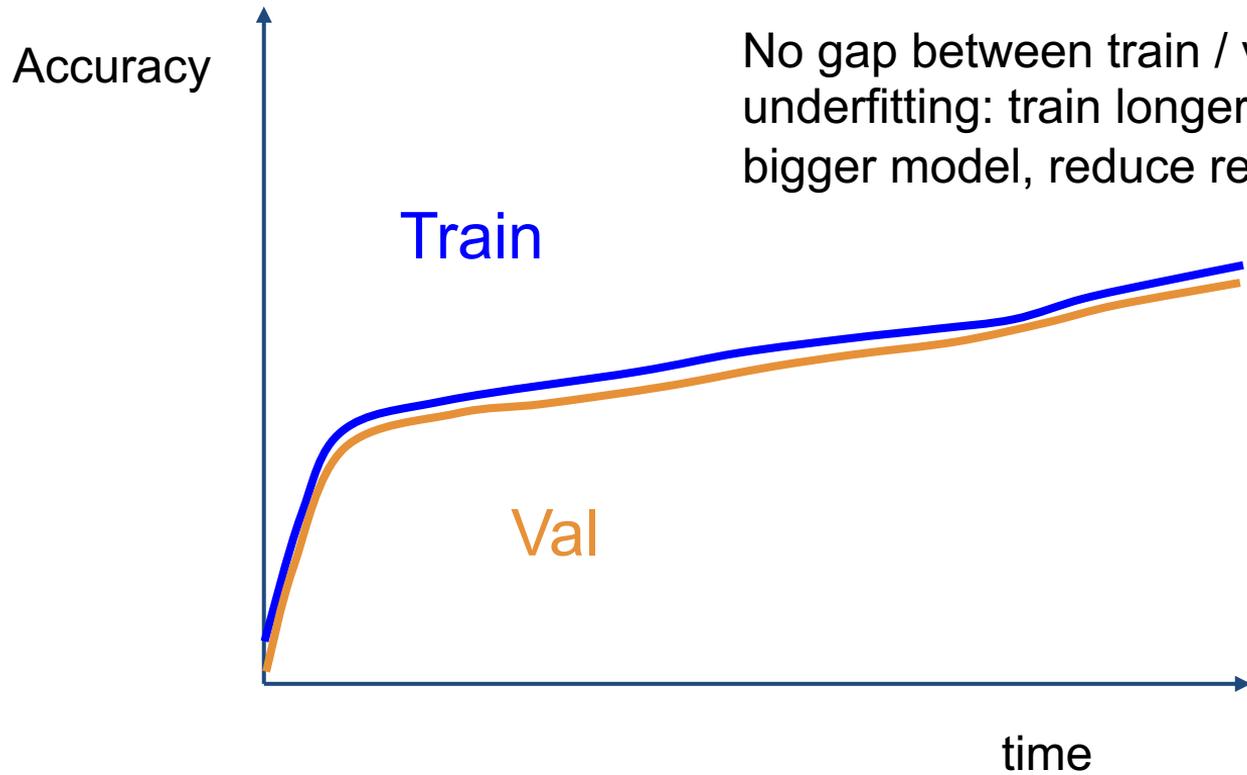
**Step 4:** Coarse grid, train for ~1-5 epochs

**Step 5:** Refine grid, train longer

**Step 6:** Look at loss and accuracy curves







No gap between train / val means underfitting: train longer, use a bigger model, reduce regularization

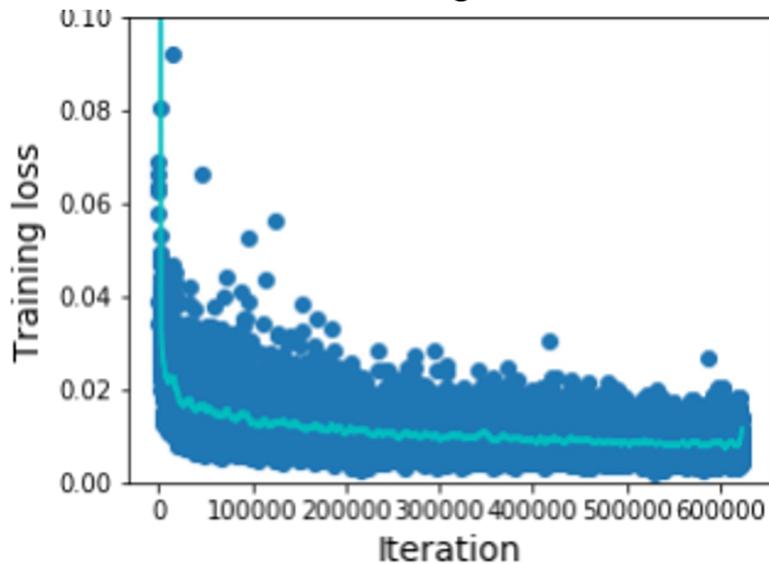
Train

Val

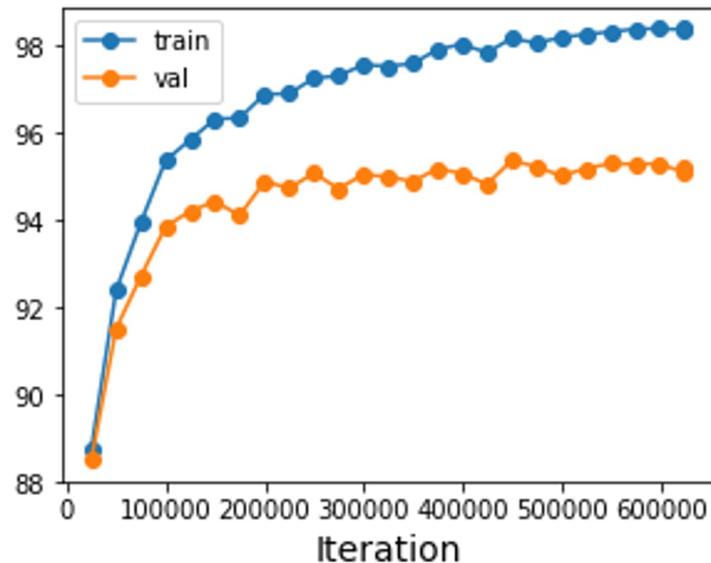
time

# Look at learning curves!

## Training Loss



## Train / Val Accuracy

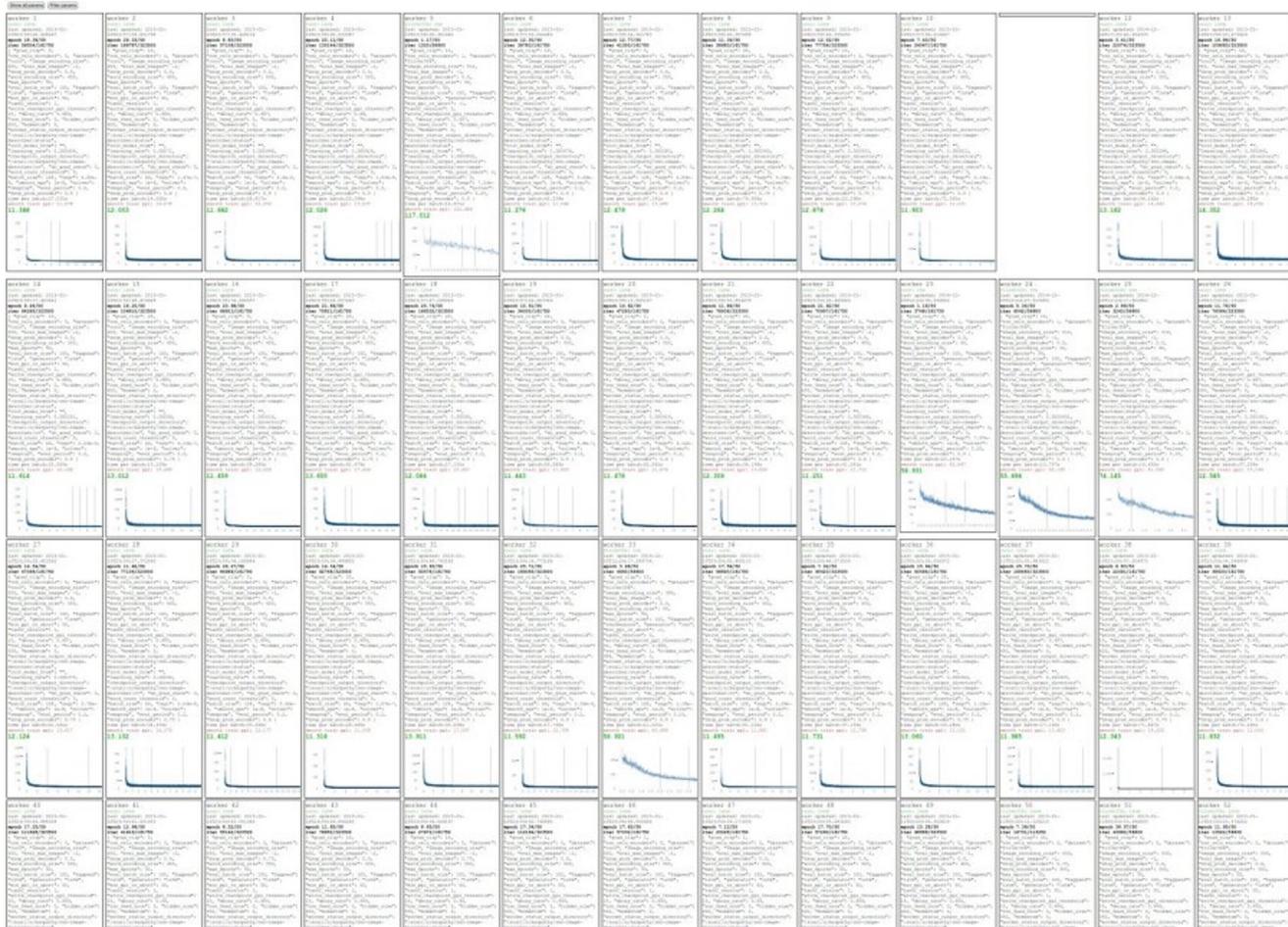


Losses may be noisy, use a scatter plot and also plot moving average to see trends better

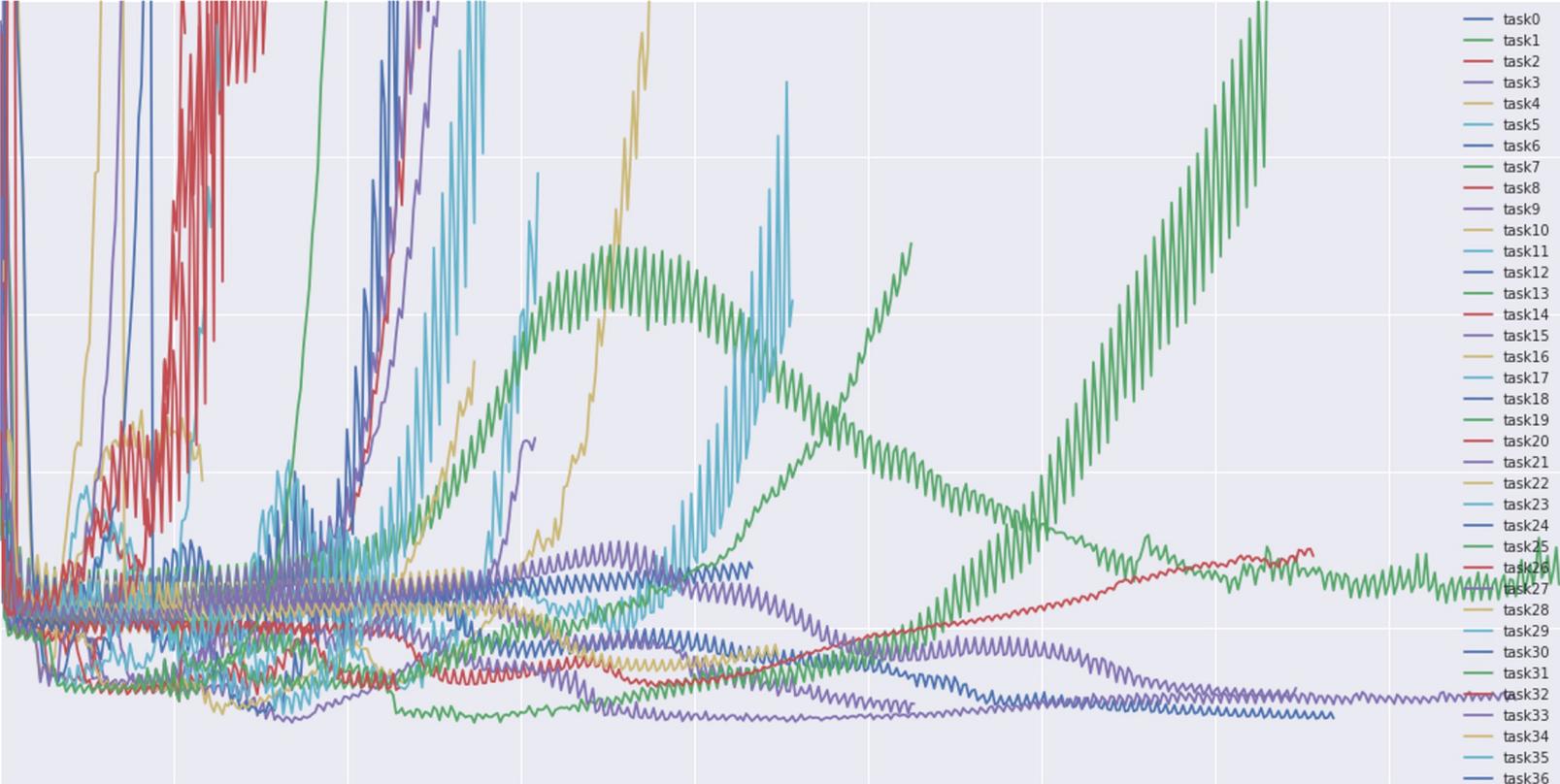
# Cross-validation

We develop "command centers" to visualize all our models training with different hyperparameters

check out [weights and biases](#)



You can plot all your loss curves for different hyperparameters on a single plot



Don't look at accuracy or loss curves for too long!



# Choosing Hyperparameters

**Step 1:** Check initial loss

**Step 2:** Overfit a small sample

**Step 3:** Find LR that makes loss go down

**Step 4:** Coarse grid, train for ~1-5 epochs

**Step 5:** Refine grid, train longer

**Step 6:** Look at loss and accuracy curves

**Step 7:** GOTO step 5

# Hyperparameters to play with:

- network architecture
- learning rate, its decay schedule, update type
- regularization (L1/L2/Dropout strength)

# Summary

- Improve your training error:
  - Optimizers
  - Learning rate schedules
- Improve your test error:
  - Regularization
  - Choosing Hyperparameters

# Summary

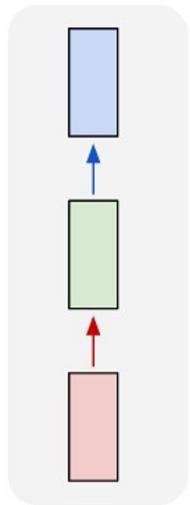
## **Training** Deep Neural Networks

- Details of the non-linear activation functions
- Data normalization
- Weight Initialization
- Batch Normalization
- Advanced Optimization
- Regularization
- Data Augmentation
- Transfer learning
- Hyperparameter Tuning

# Sequence Modeling: Recurrent Neural Networks

# “Vanilla” Neural Network

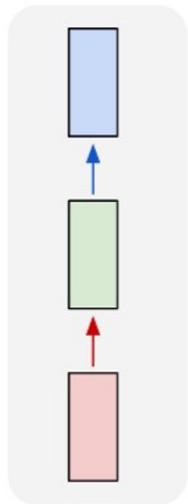
one to one



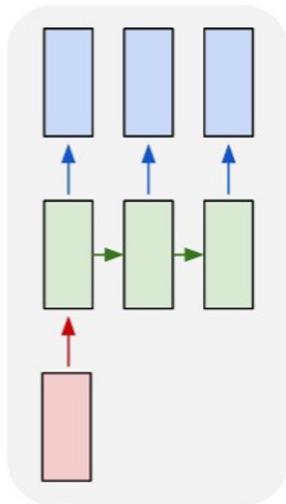
**Vanilla Neural Networks**

# Recurrent Neural Networks: Process Sequences

one to one



one to many

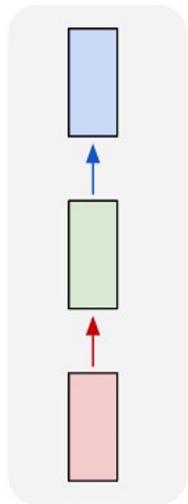


e.g. **Image Captioning**

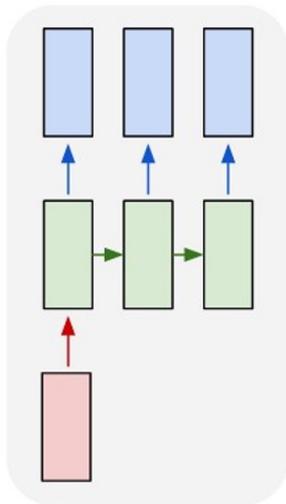
image -> sequence of words

# Recurrent Neural Networks: Process Sequences

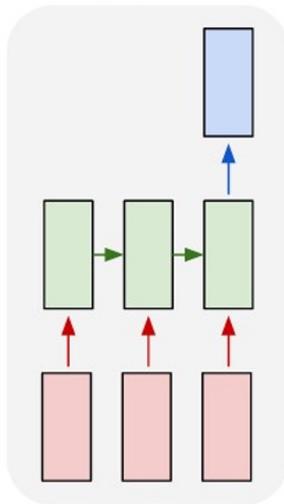
one to one



one to many



many to one

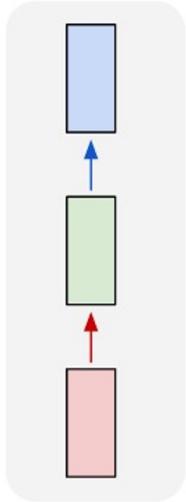


e.g. **sentiment analysis**

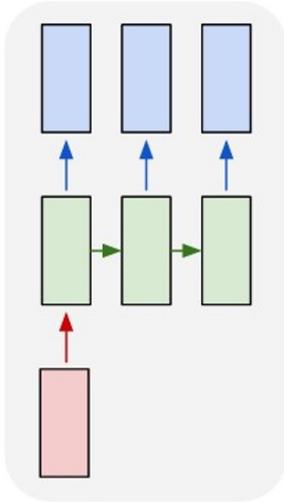
sequence of words -> sentiment label

# Recurrent Neural Networks: Process Sequences

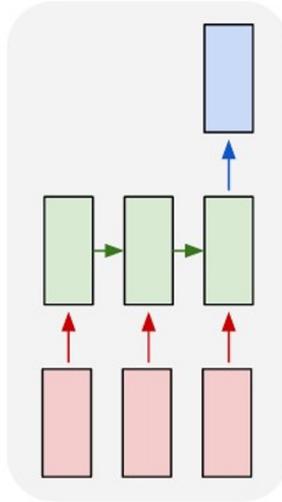
one to one



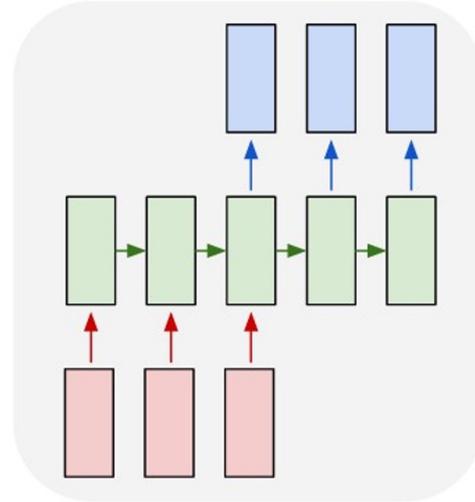
one to many



many to one



many to many

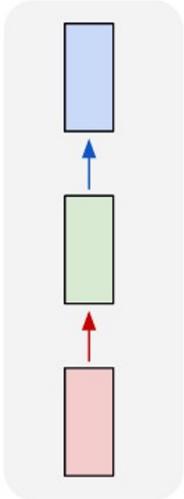


**E.g. Translation, Q&A,  
Conversation**

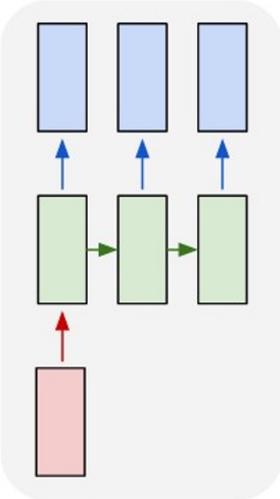
Sequence of words ->  
sequence of words

# Recurrent Neural Networks: Process Sequences

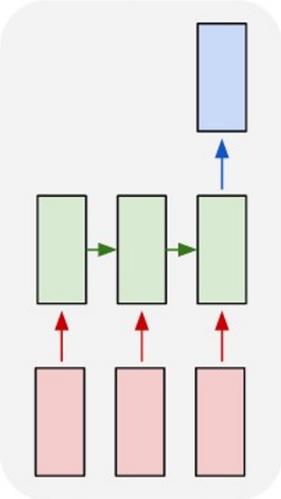
one to one



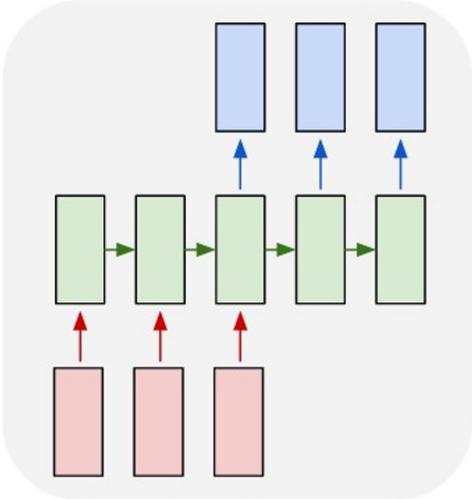
one to many



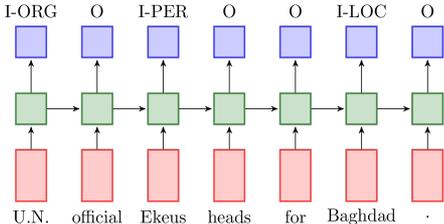
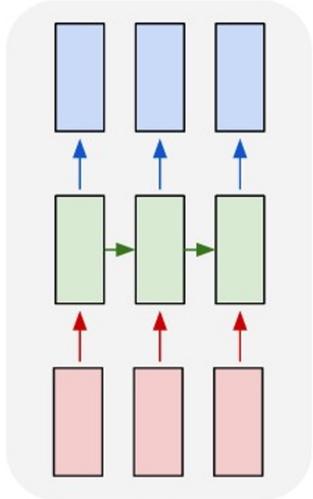
many to one



many to many



many to many



e.g. **Language entity recognition**



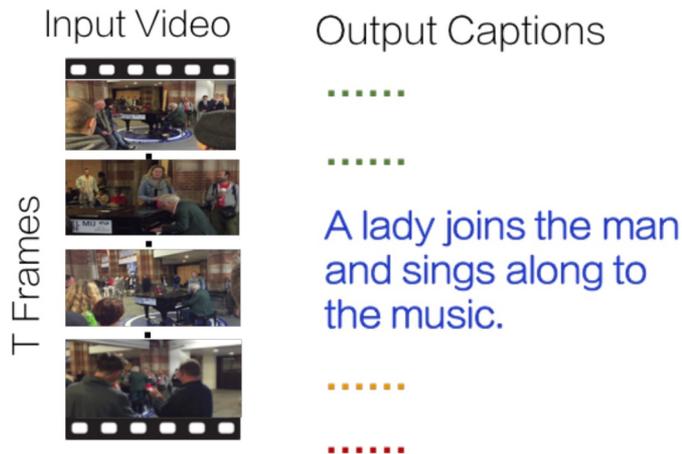
# Why are existing convnets insufficient?

Variable sequence length inputs and outputs!

**Example task:** video captioning

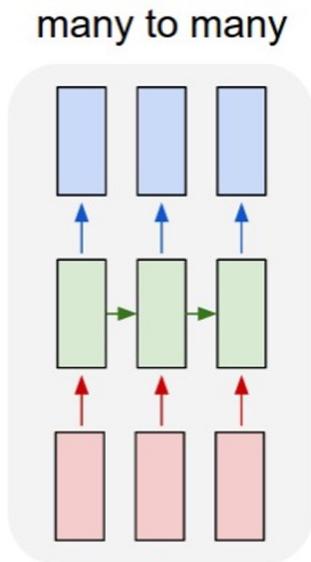
**Input** video can have variable number of frames

**Output** captions can be variable length.



# Let's start with a setting that takes a variable input and produces an output at every step

Example: Video activity labeling



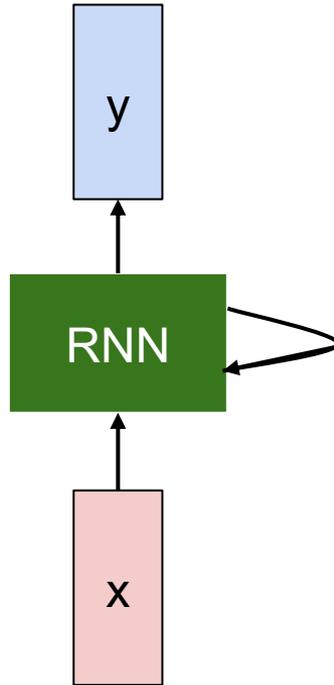
Huang et al., 2016

**Input:** video frame; **Output:** activity label at each frame  
Recognizing an activity requires looking at more than one frame!

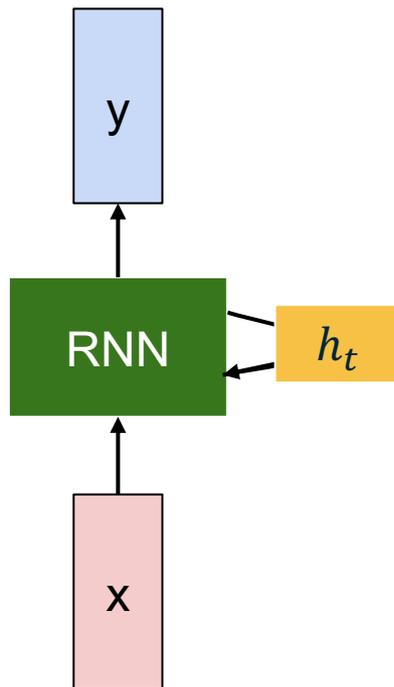
Want: a model that can make prediction for each frame based on the past frames.

We need a model that can *memorize* what it has seen so far!

# Recurrent Neural Network

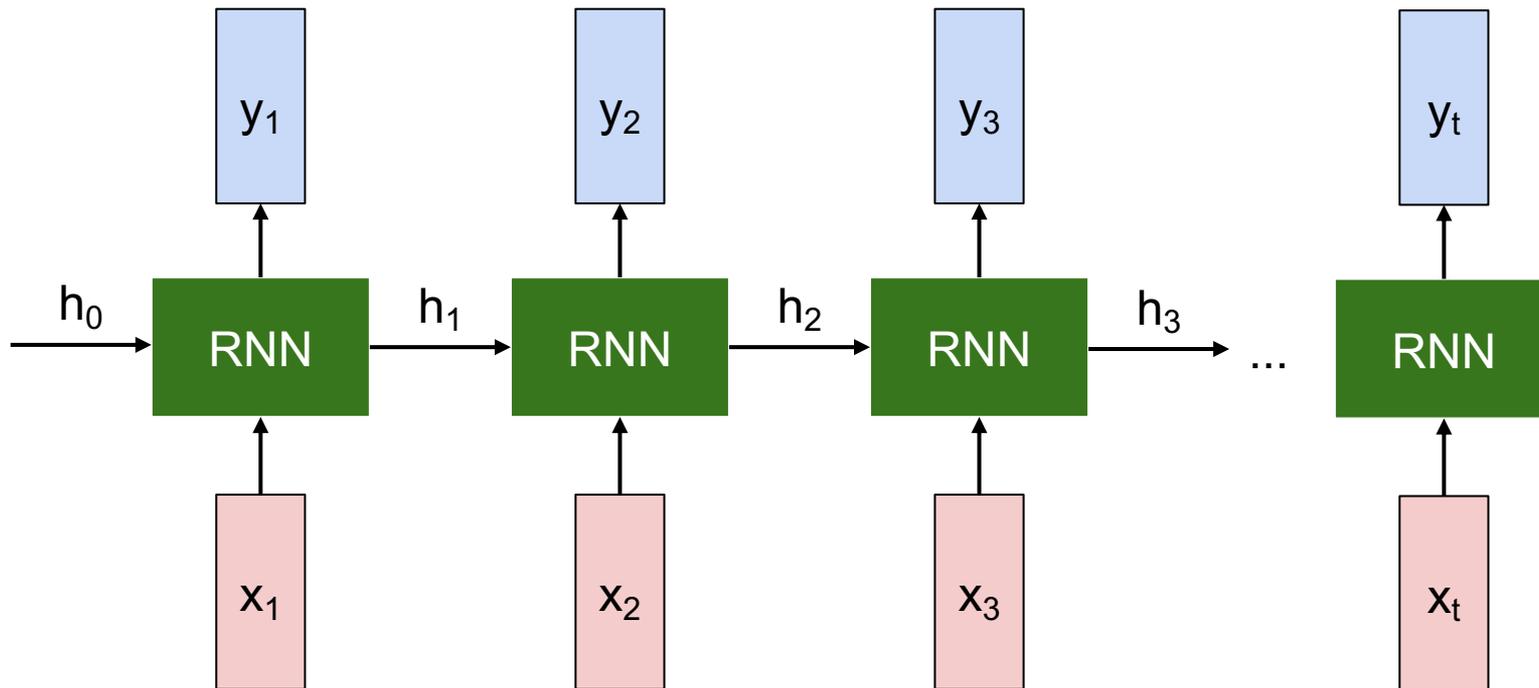


# Recurrent Neural Network

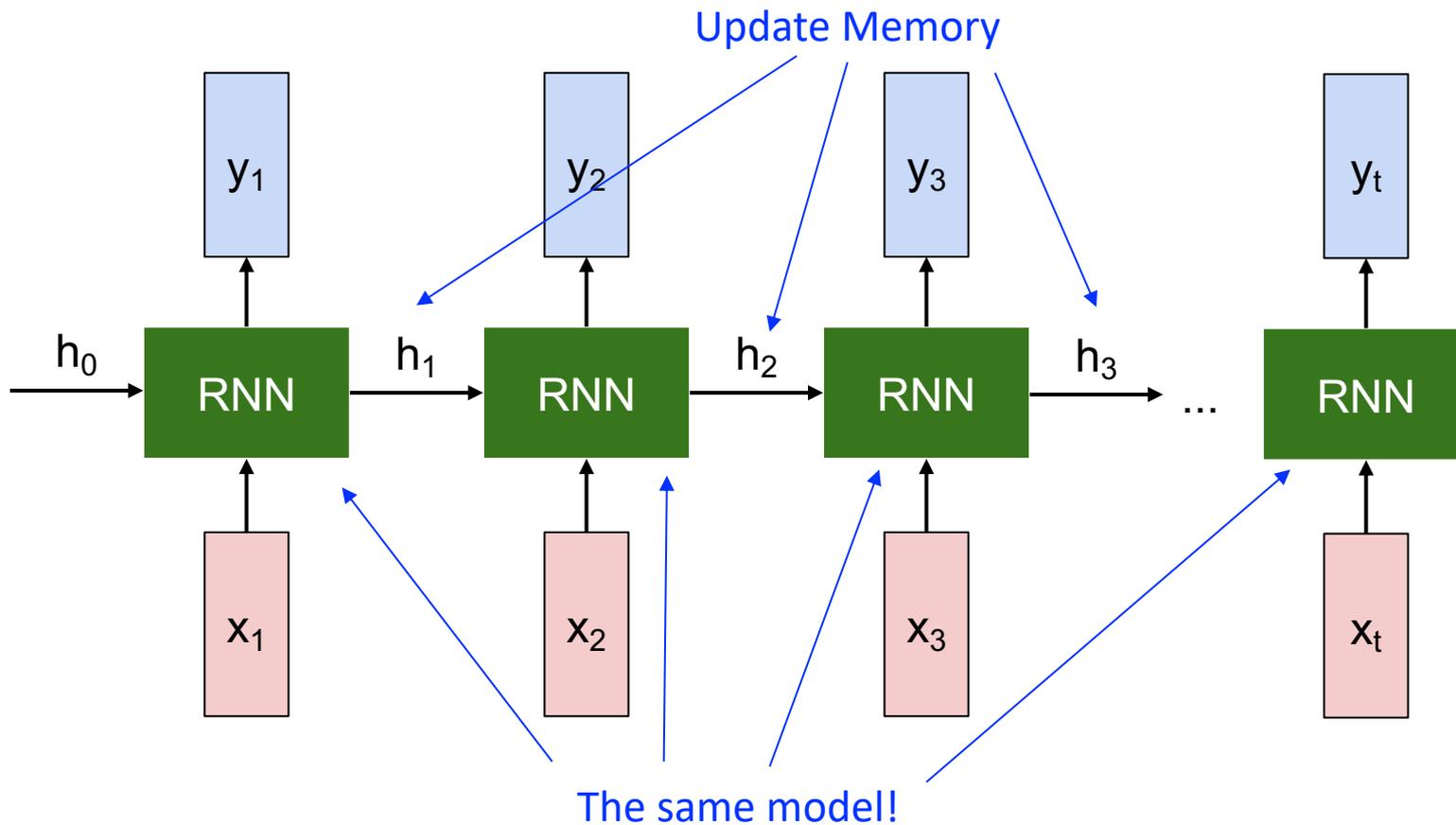


Key idea: RNNs have an “internal state” that is updated as a sequence is processed. You can think of it as “memory”.

# Unrolled RNN



# Unrolled RNN



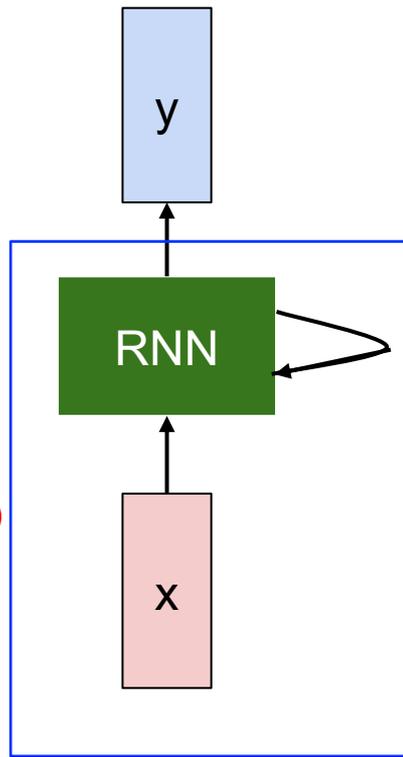
# RNN hidden state update

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

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

new state (vector)      old state (vector)      input vector at some time step

some model with parameters  $W$



Can set initial state  $h_0$  to all 0's

# RNN output generation

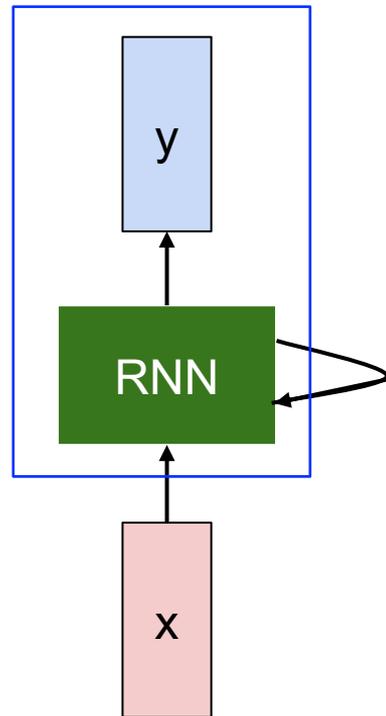
“Read out” the prediction by passing the hidden state through a network (e.g., a few FC layers)

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

output

new state

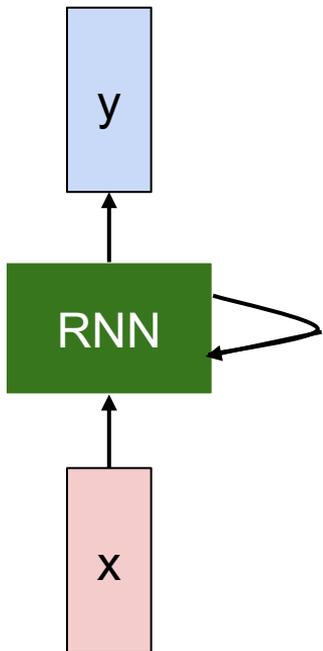
another model  
with parameters  $W_{hy}$



The prediction network is often shared across timestep.

# (Simple) Recurrent Neural Network

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



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

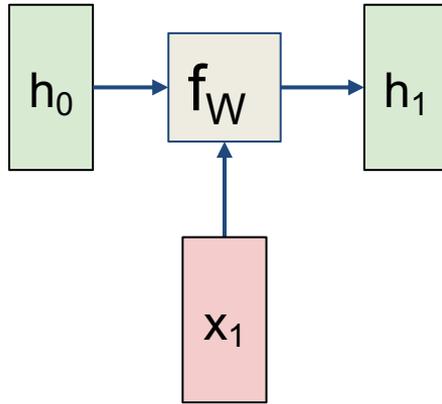


$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

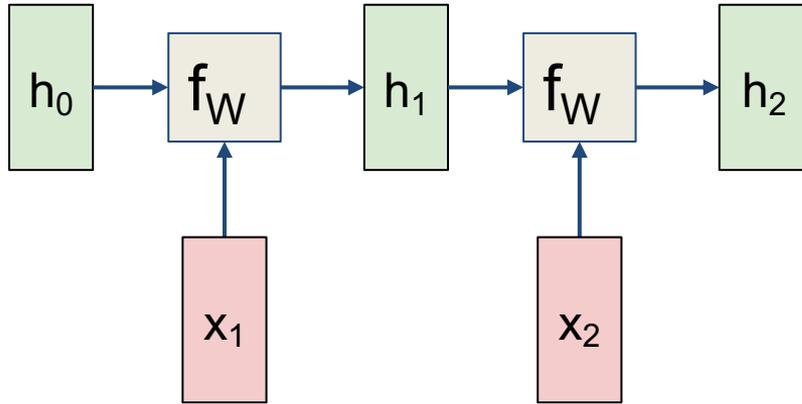
$$y_t = W_{hy}h_t$$

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

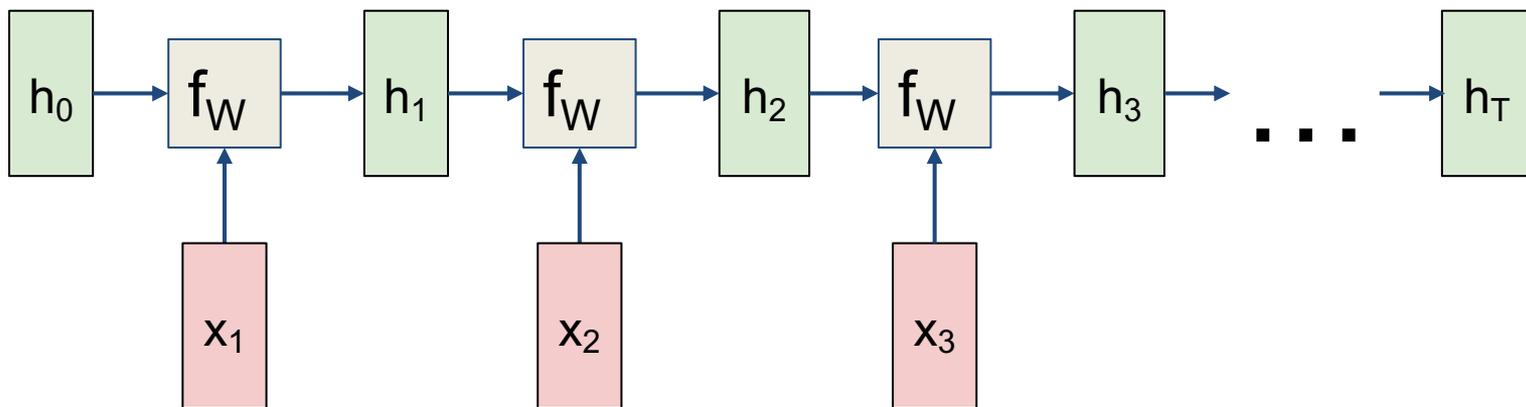
# RNN: Computational Graph



# RNN: Computational Graph

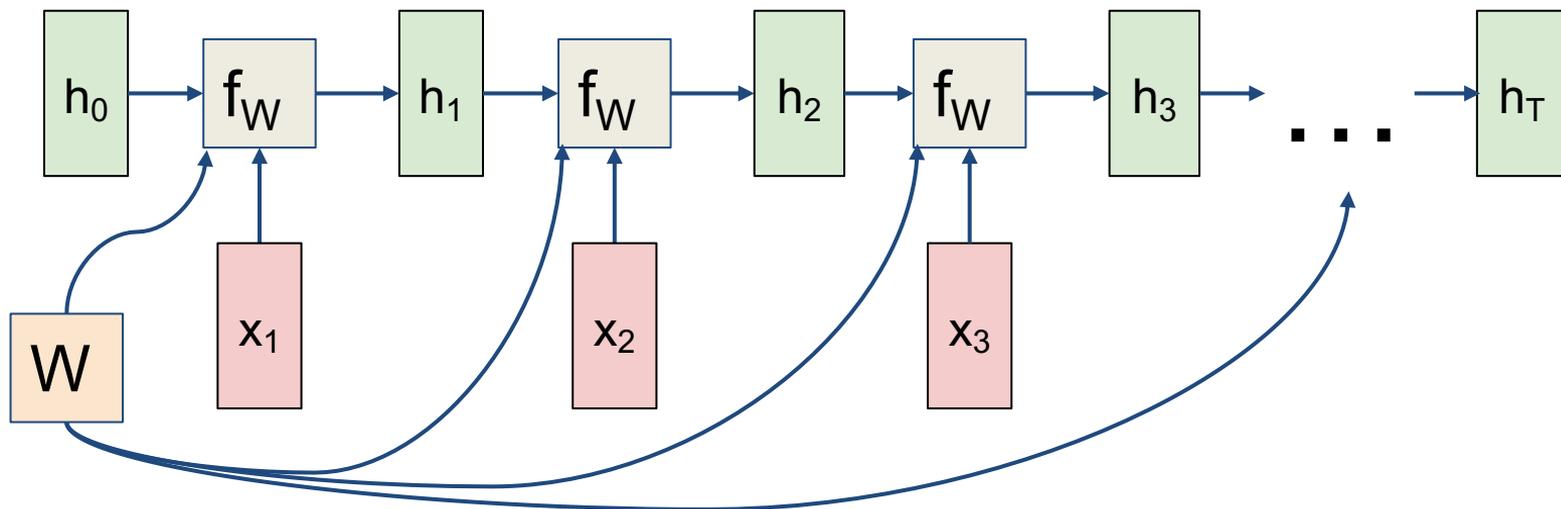


# RNN: Computational Graph

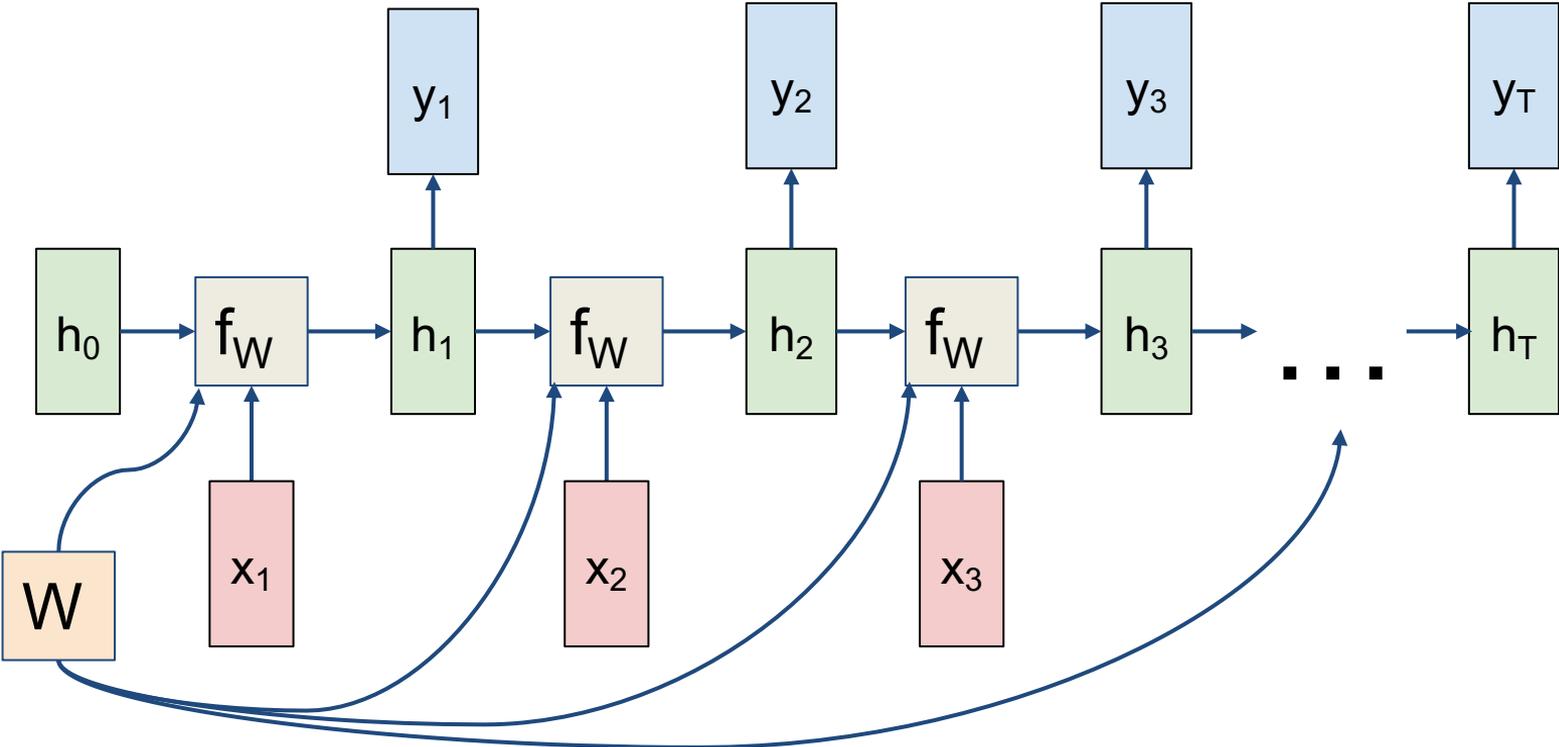


# RNN: Computational Graph

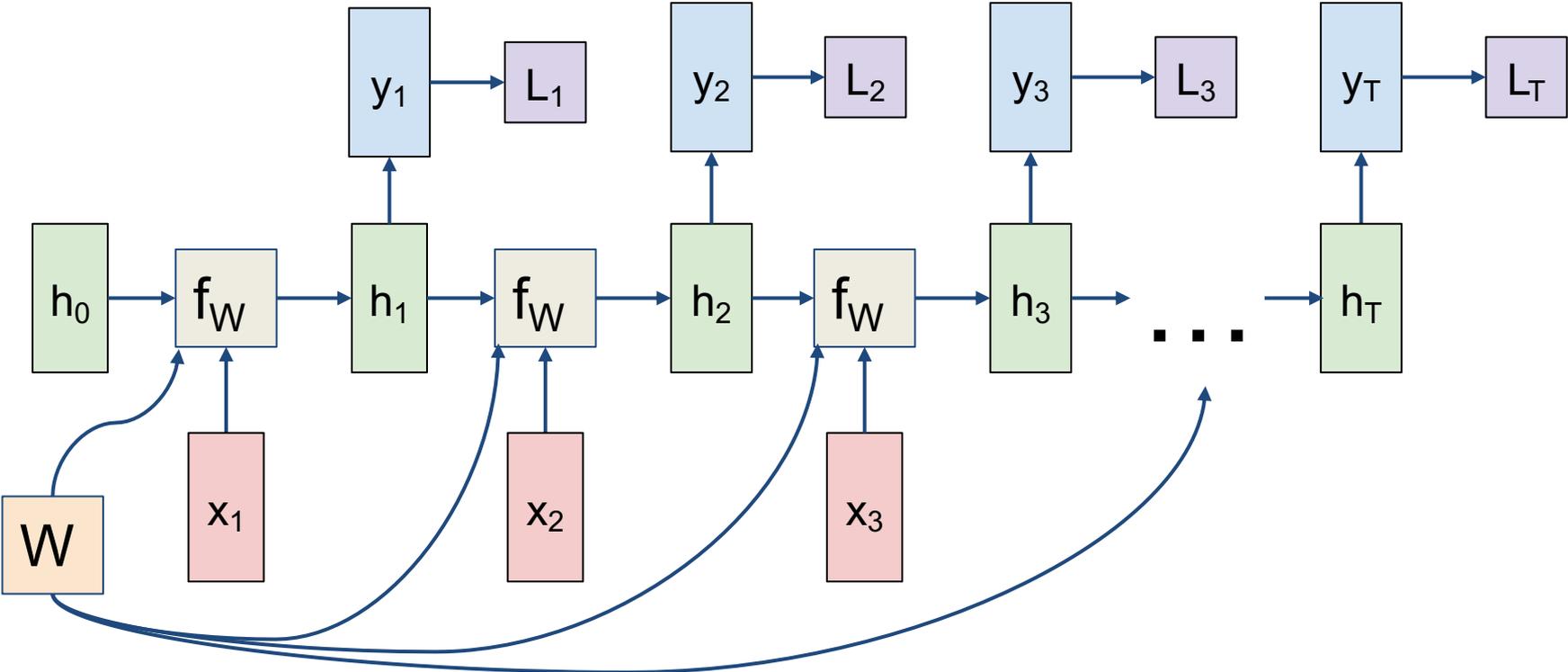
Re-use the same weight matrix at every time-step



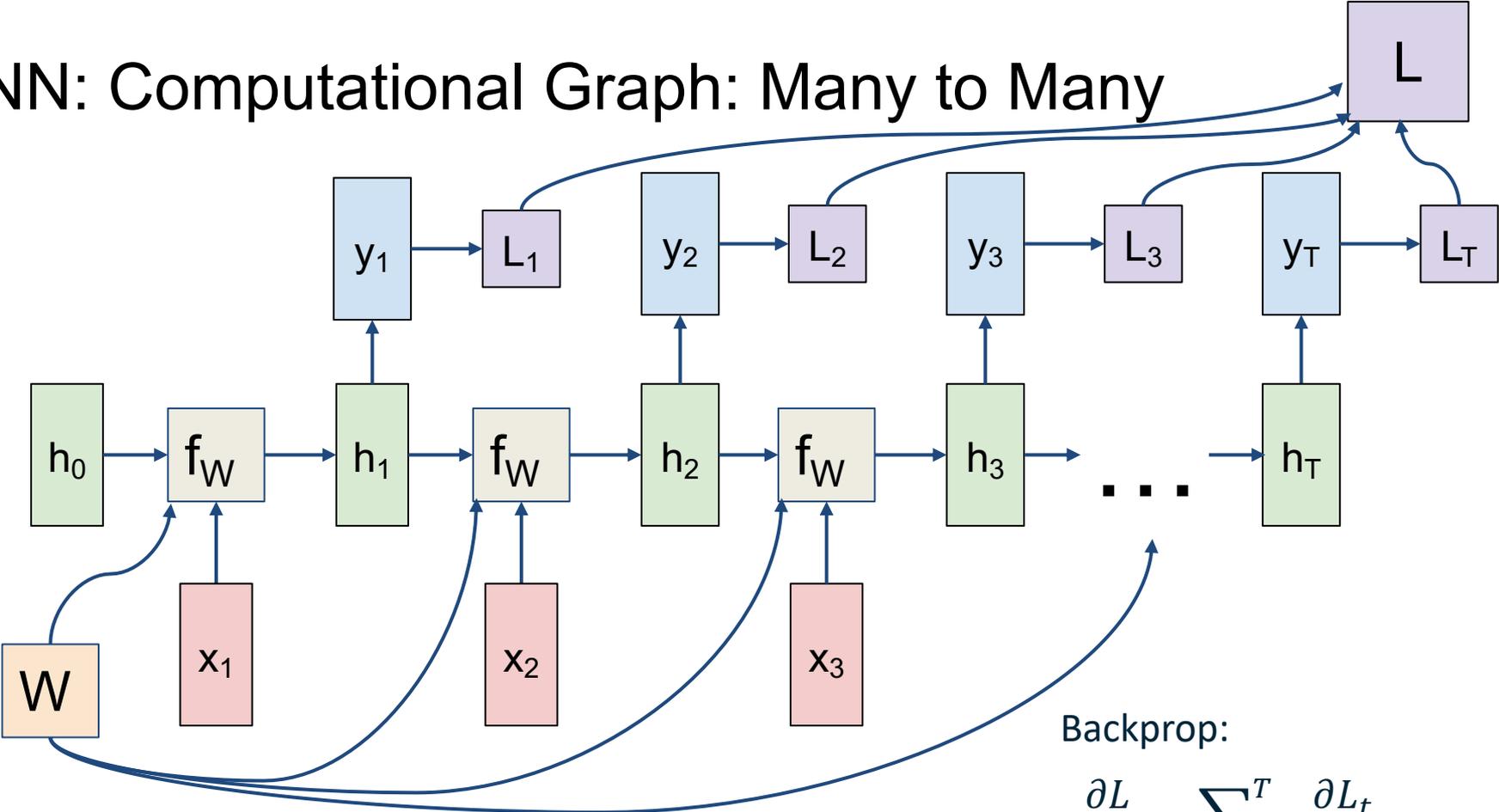
# RNN: Computational Graph: Many to Many



# RNN: Computational Graph: Many to Many



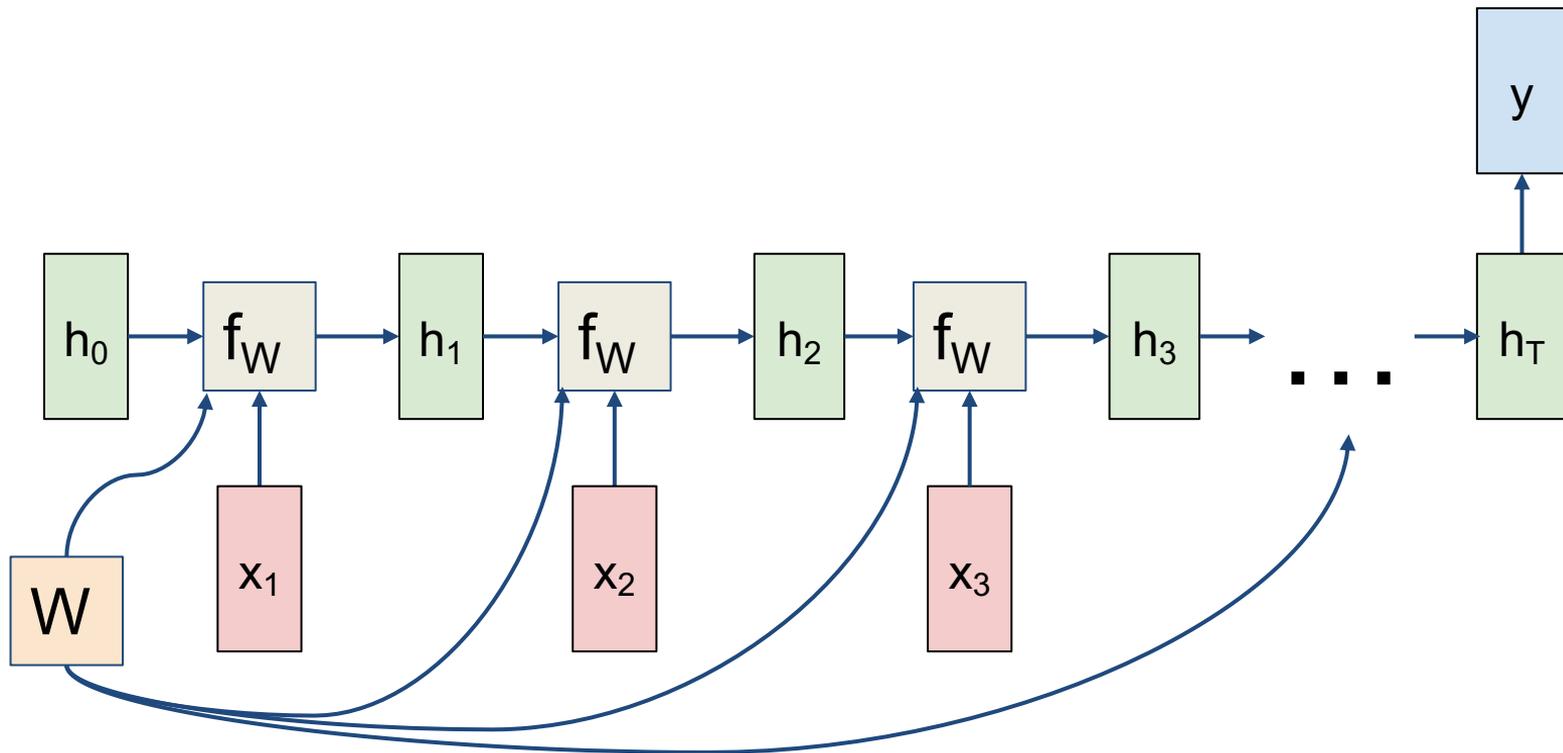
# RNN: Computational Graph: Many to Many



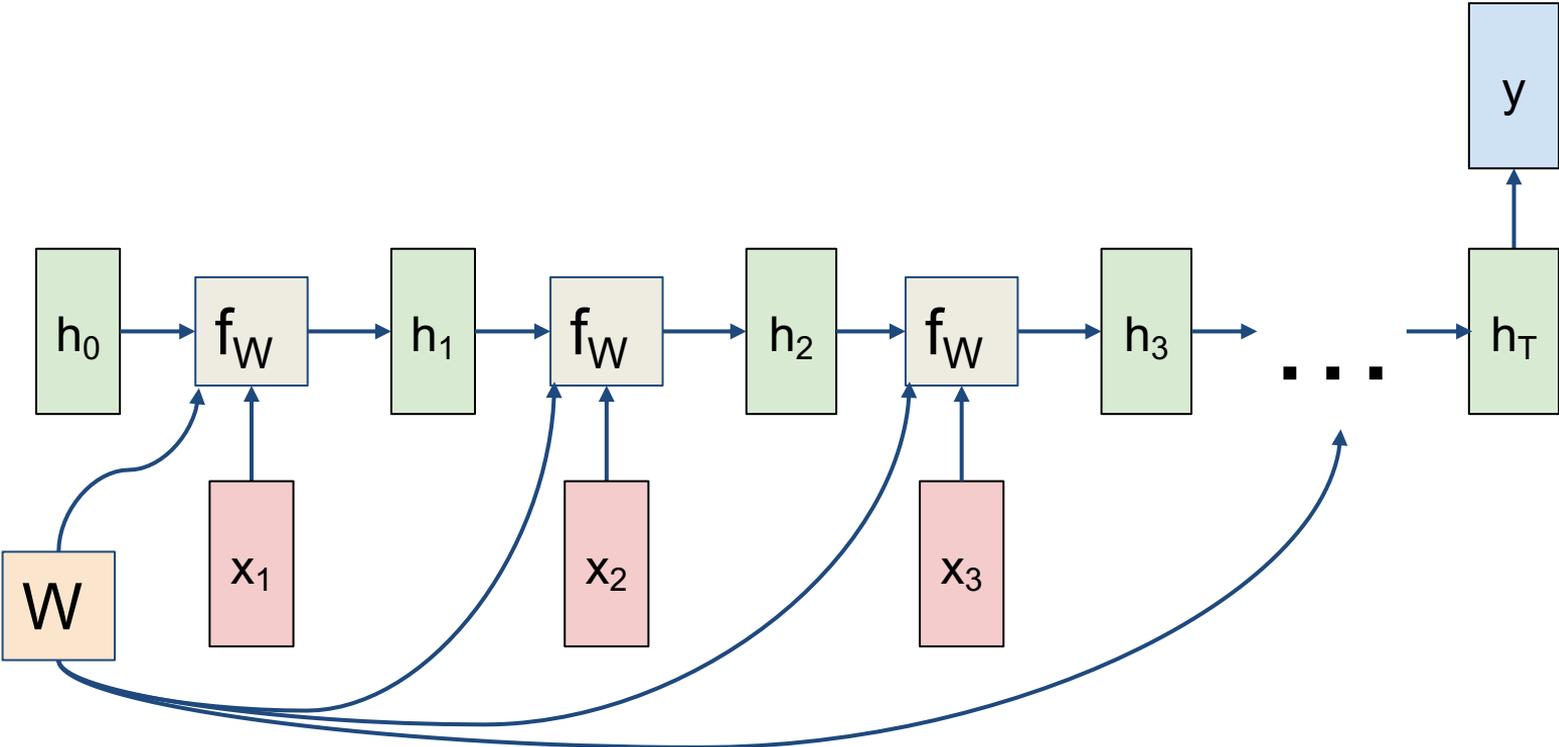
Backprop:

$$\frac{\partial L}{\partial W_i} = \sum_{t=i}^T \frac{\partial L_t}{\partial W_i}$$

# RNN: Computational Graph: Many to One

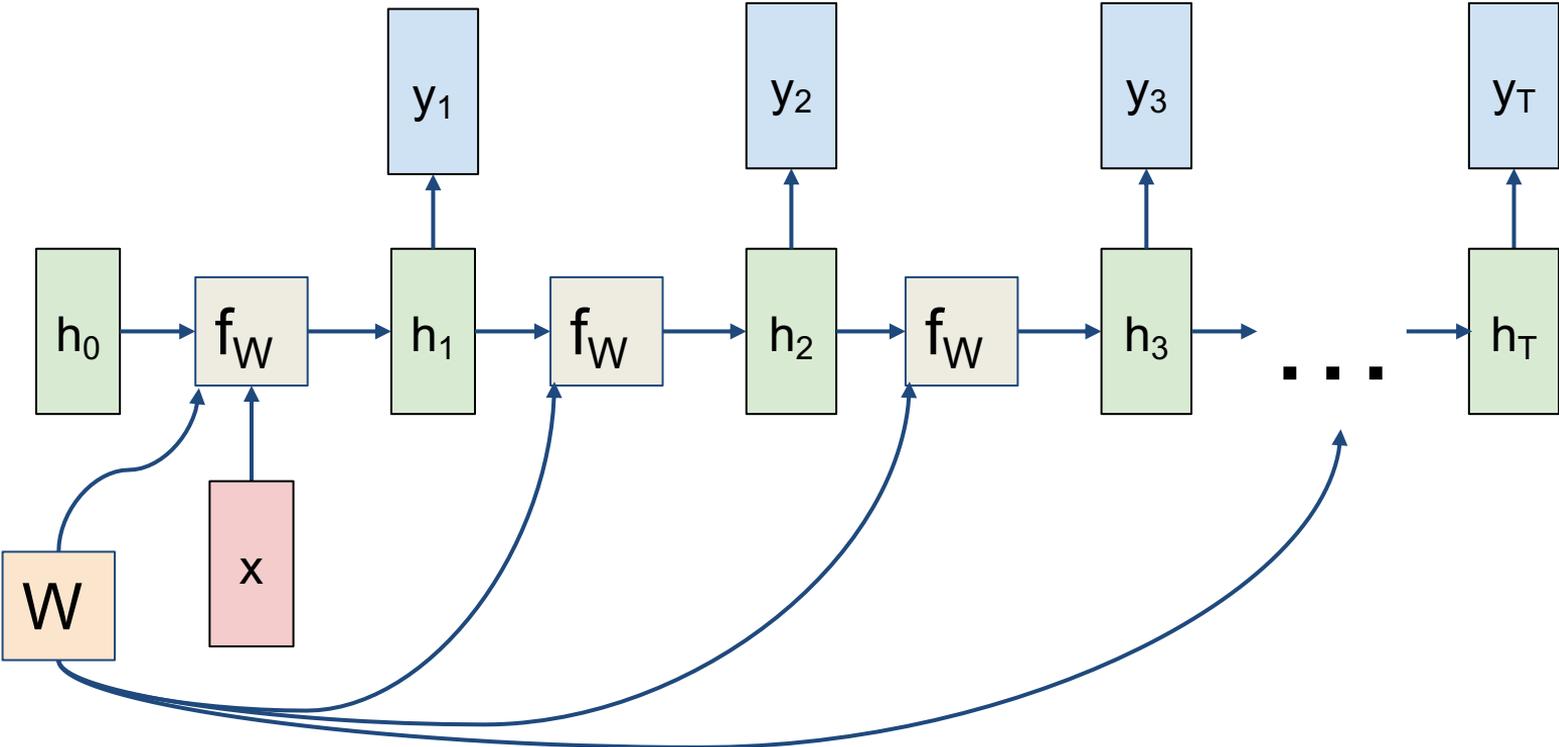


# RNN: Computational Graph: Many to One



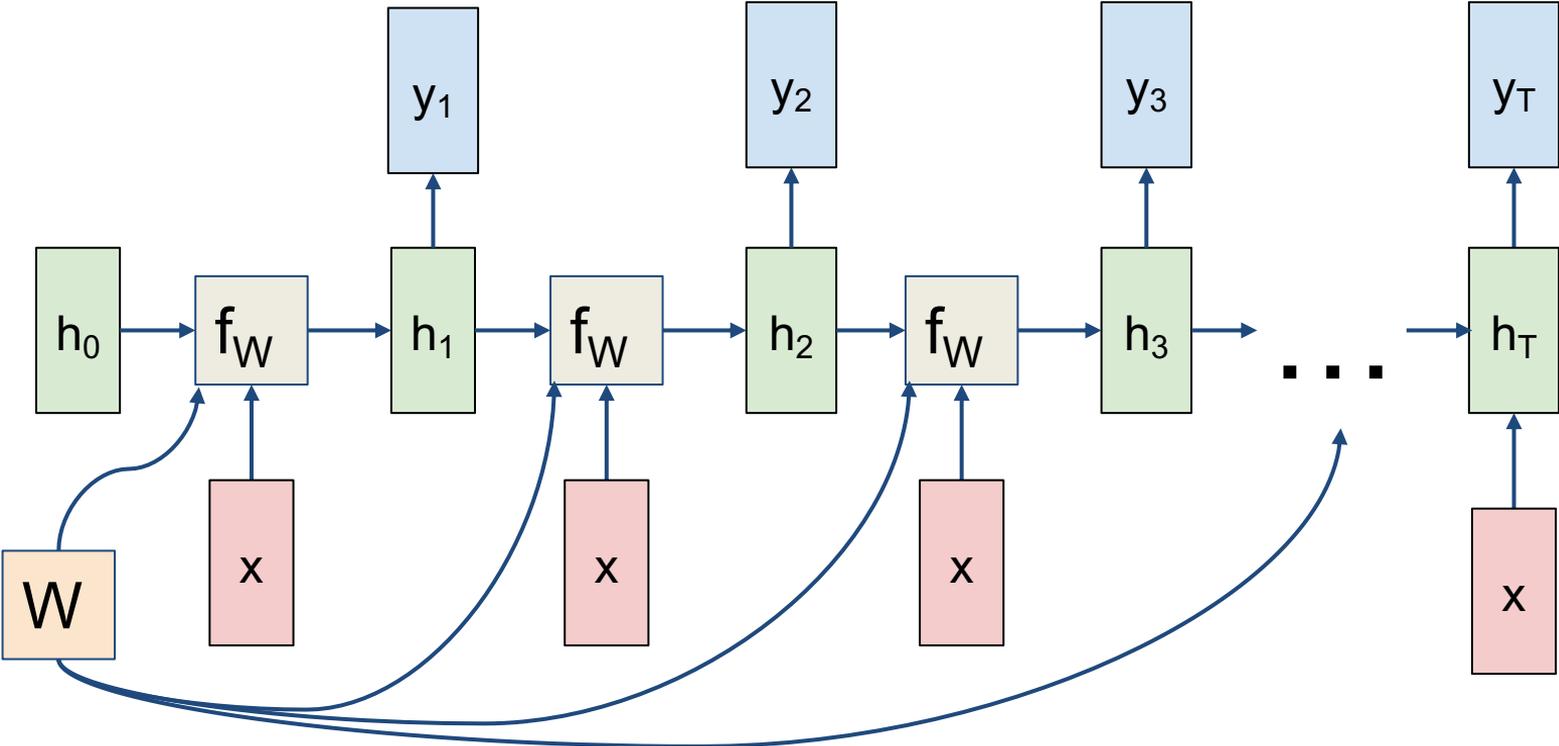
Example: sentence classification

# RNN: Computational Graph: One to Many



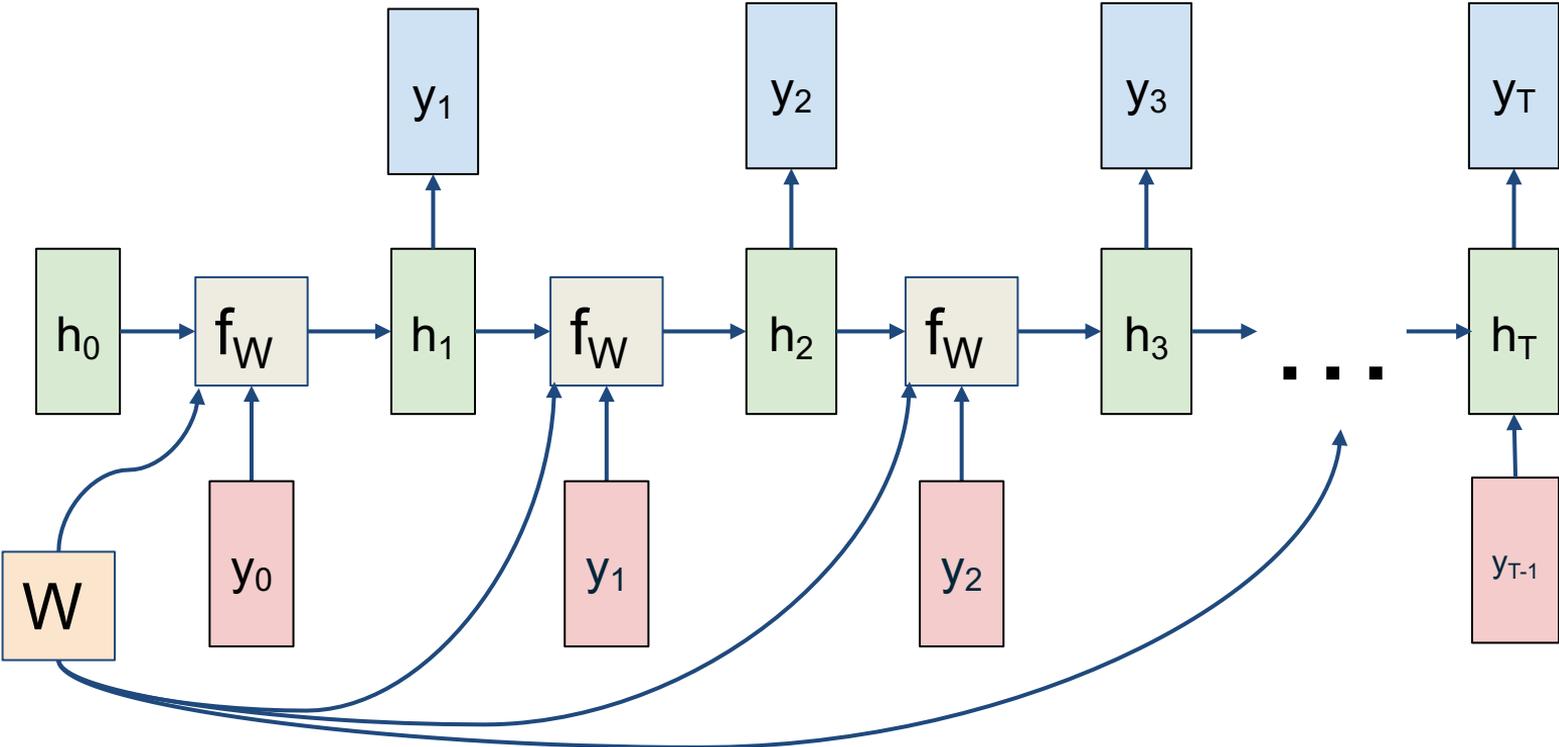
Example: image captioning

# RNN: Computational Graph: One to Many



Example: image captioning

# RNN: Computational Graph: One to Many

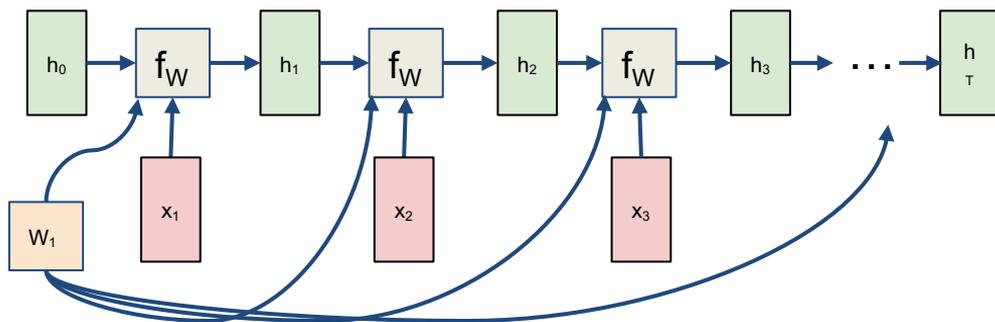


Example: text generation

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

**Many to one:** Encode input sequence in a single vector

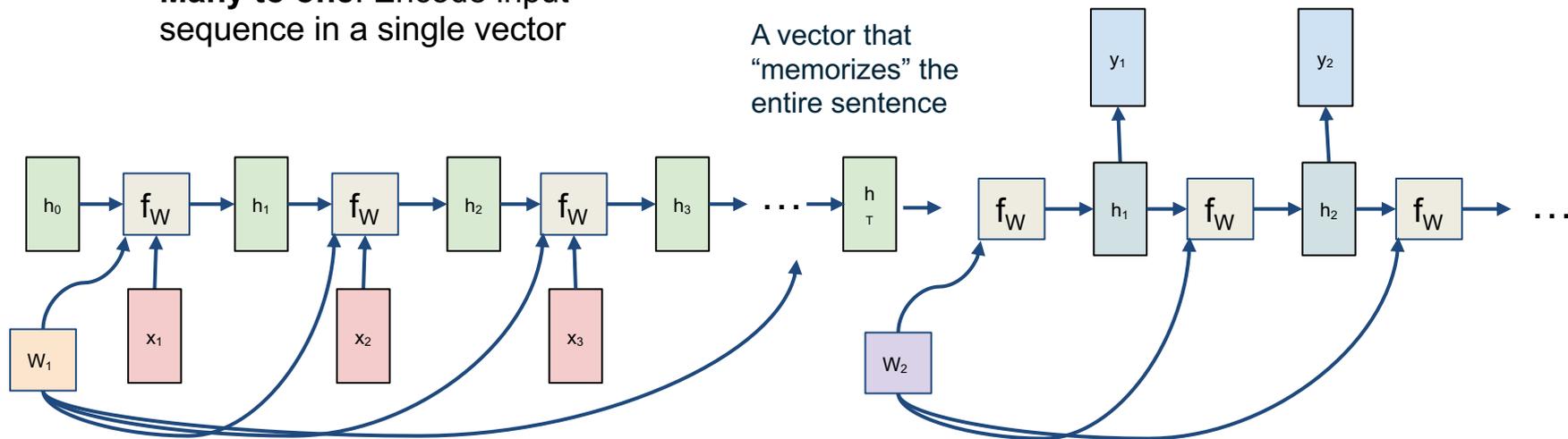
A vector that  
“memorizes” the  
entire sentence



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

**Many to one:** Encode input sequence in a single vector

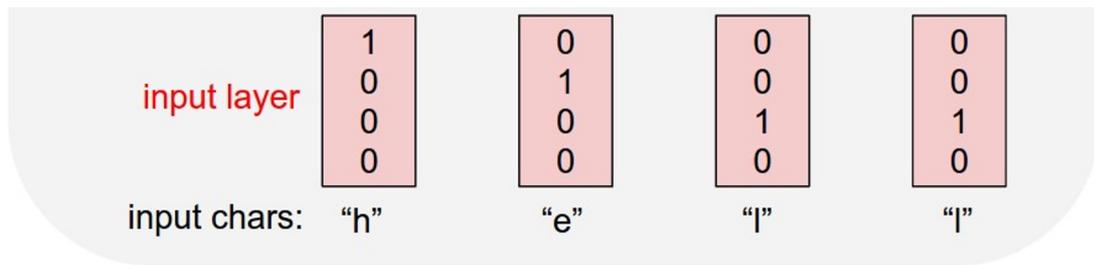
**One to many:** Produce output sequence from single input vector



# Example: Character-level Language Model

Vocabulary:  
[h,e,l,o]

Example training  
sequence:  
“hello” with  
**one-hot encoding**

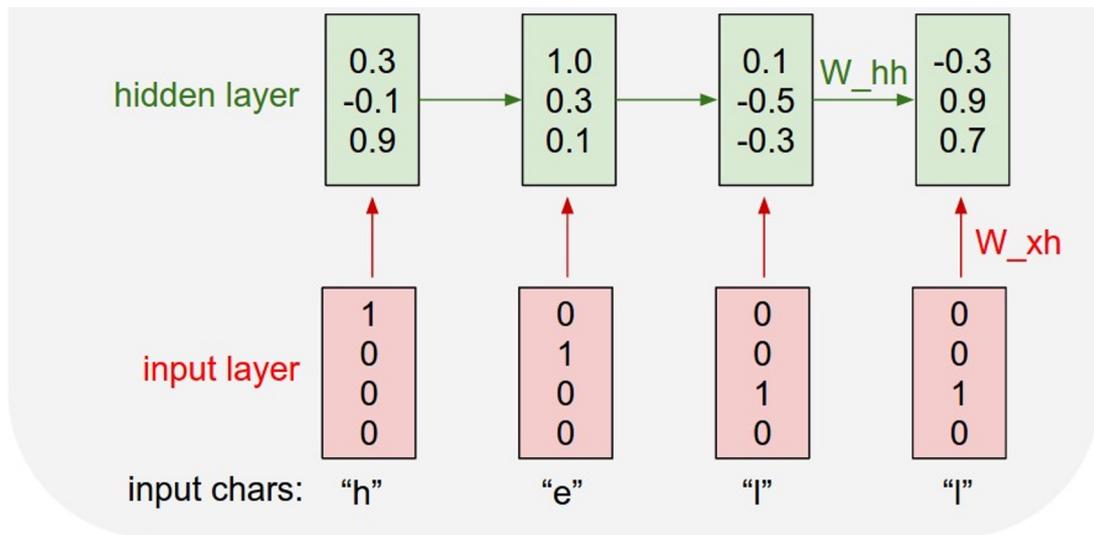


# Example: Character-level Language Model

Vocabulary:  
[h,e,l,o]

Example training  
sequence:  
“**hello**” with  
one-hot encoding

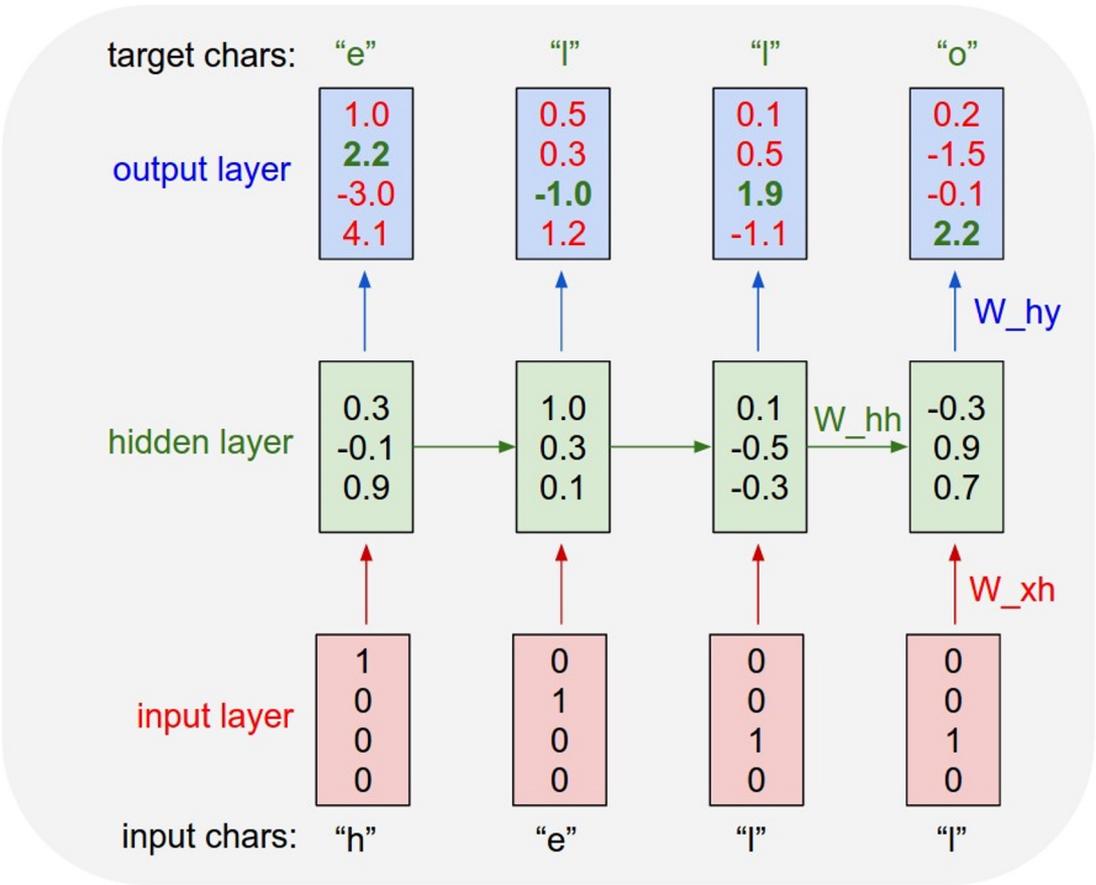
$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$



# Example: Character-level Language Model

Vocabulary:  
[h,e,l,o]

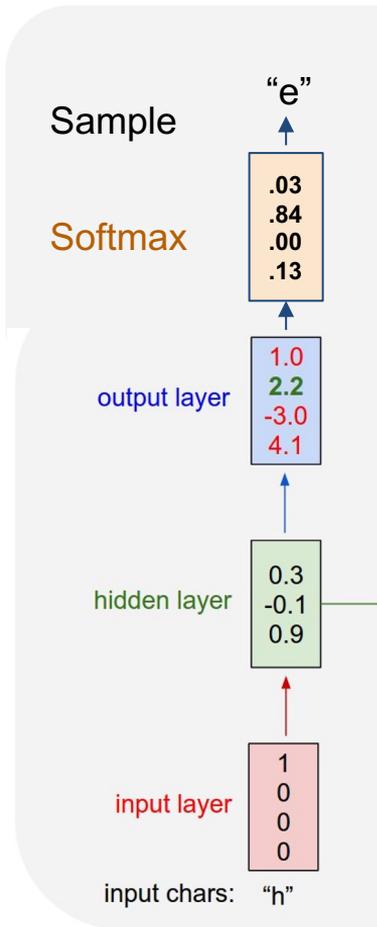
Example training  
sequence:  
“**hello**” with  
one-hot encoding



# Example: Character-level Language Model Sampling

Vocabulary:  
[h,e,l,o]

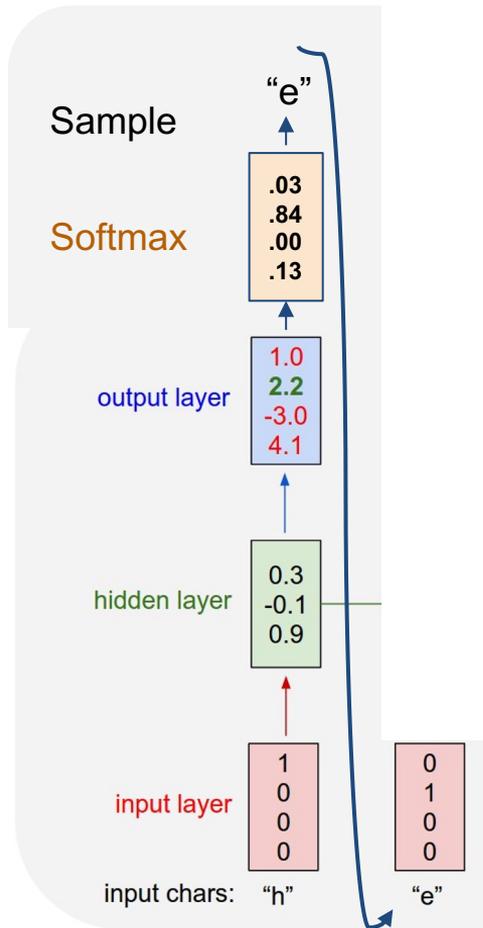
At test-time sample  
characters one at a time,  
feed back to model



# Example: Character-level Language Model Sampling

Vocabulary:  
[h,e,l,o]

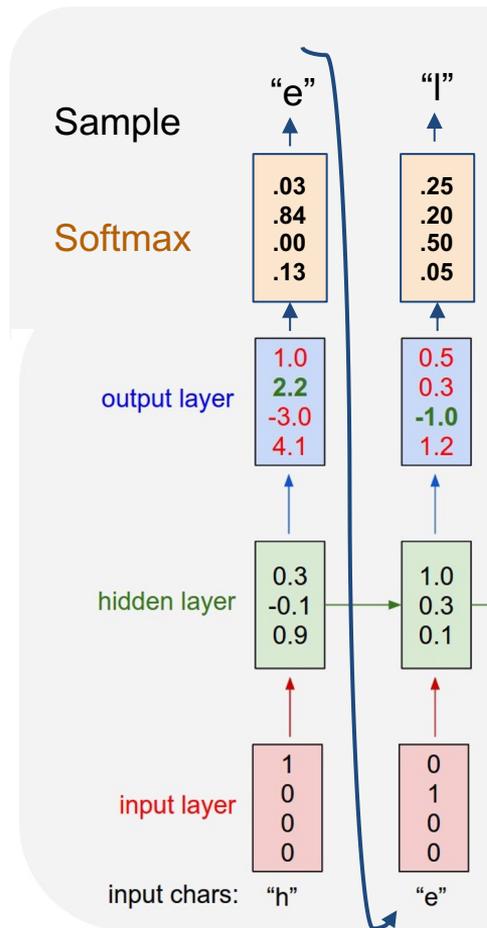
At test-time sample characters one at a time, feed back to model



# Example: Character-level Language Model Sampling

Vocabulary:  
[h,e,l,o]

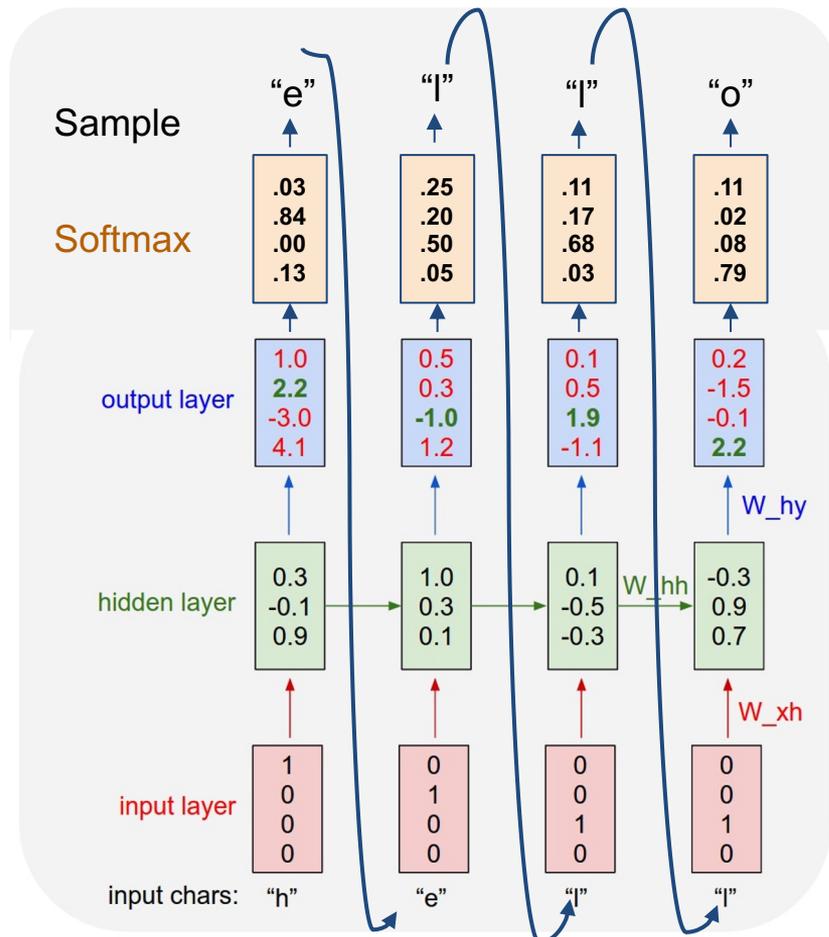
At test-time sample characters one at a time, feed back to model



# Example: Character-level Language Model Sampling

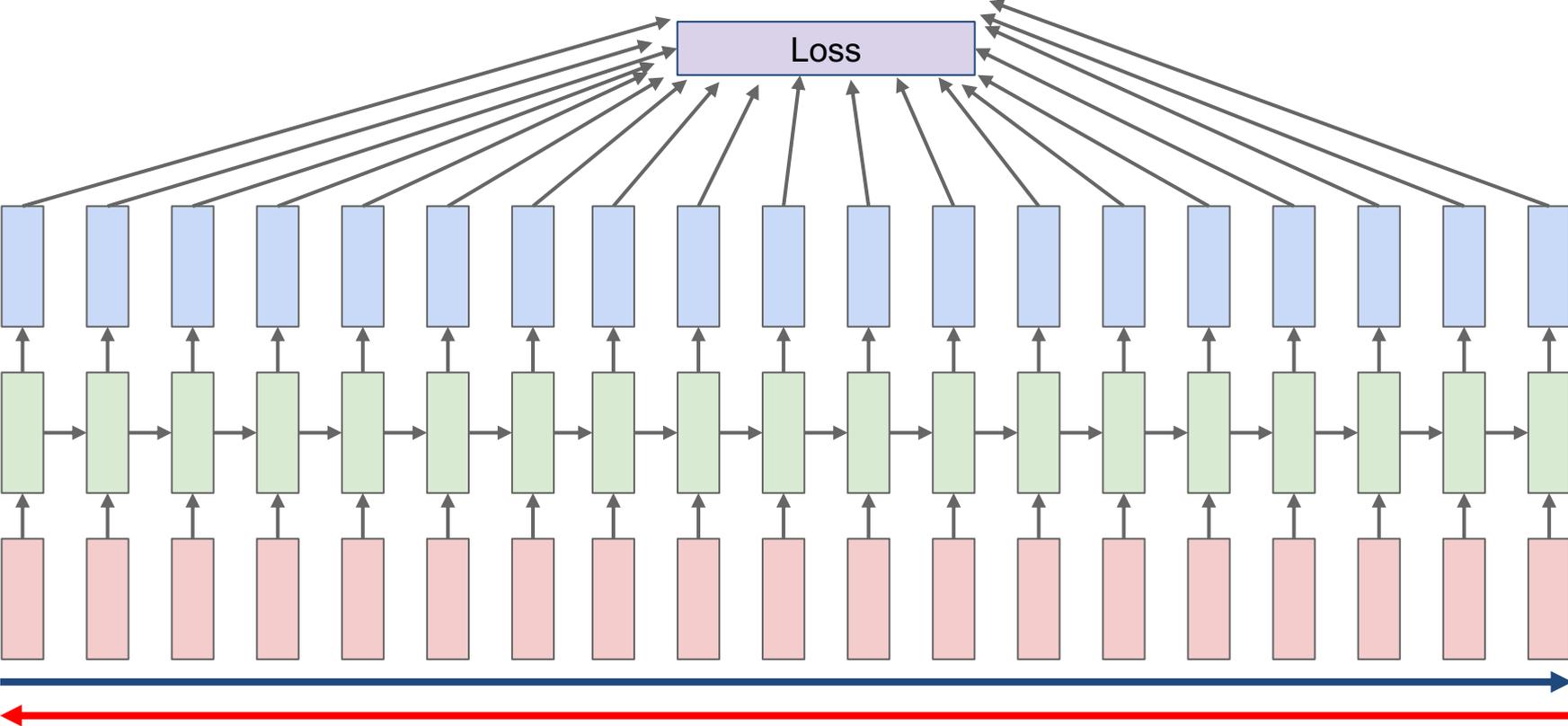
Vocabulary:  
[h,e,l,o]

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



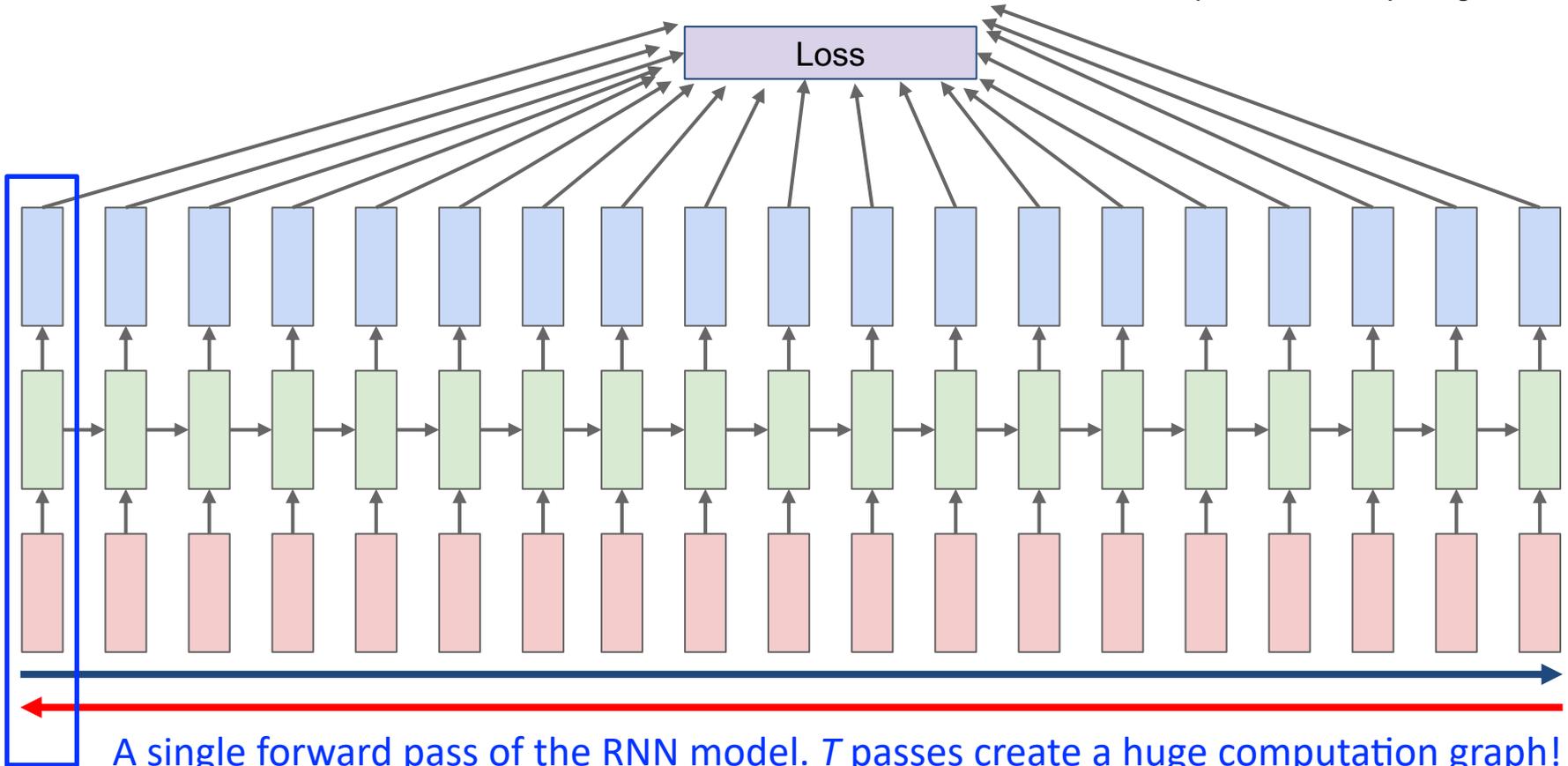
# Backpropagation through time

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



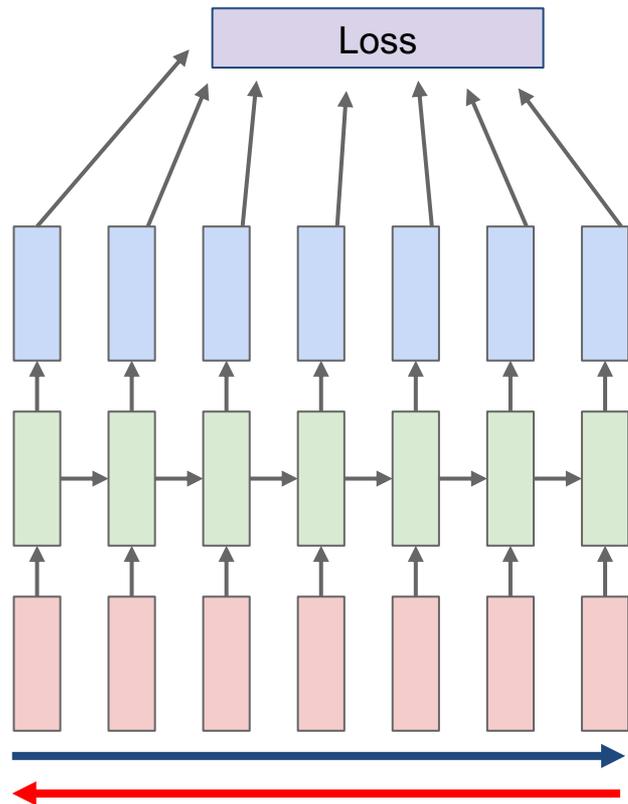
# Backpropagation through time

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



A single forward pass of the RNN model.  $T$  passes create a huge computation graph!

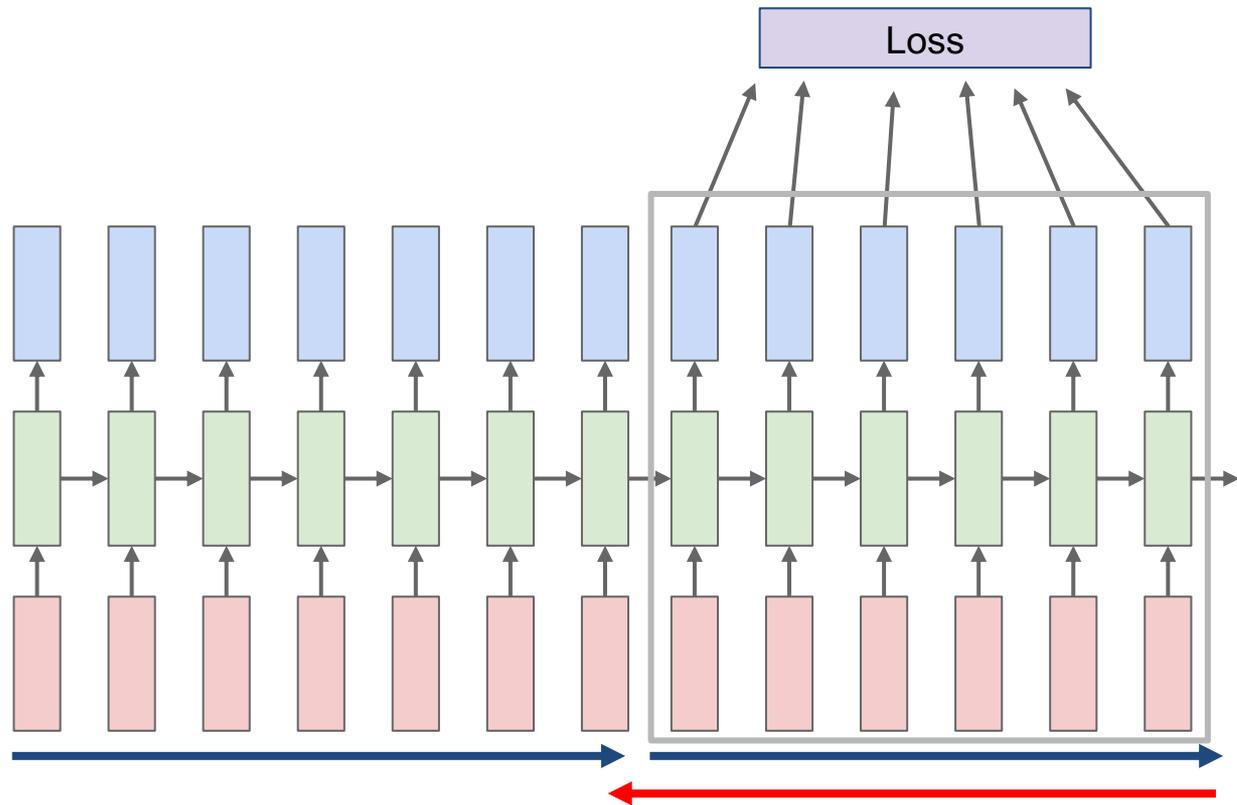
# Truncated Backpropagation through time



Run forward and backward through chunks (length  $k$ ) of the sequence instead of whole sequence, do parameter update, clear gradient cache

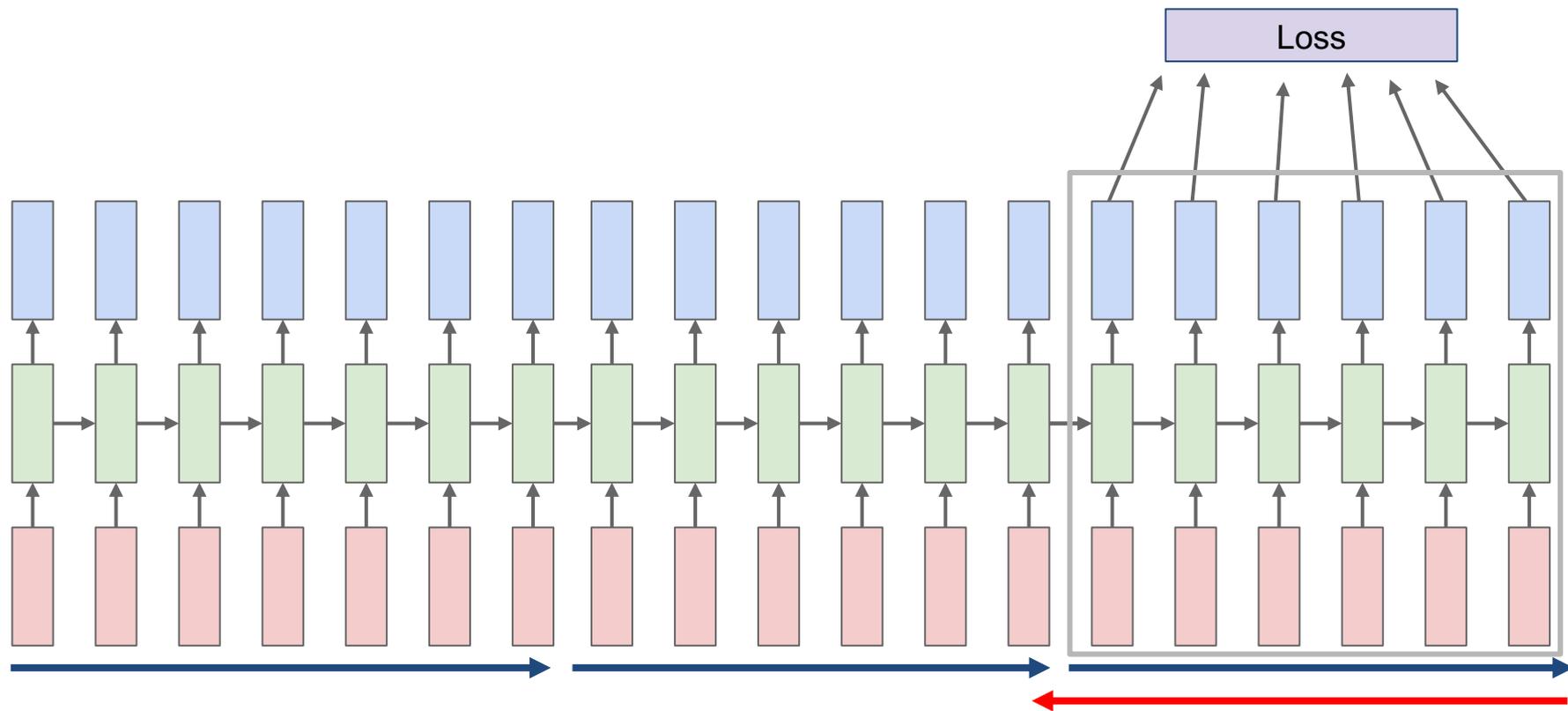
Save the last hidden state!

# Truncated Backpropagation through time



Carry hidden states forward in time for  $k$  steps, backprop, update parameter, clear gradient ...

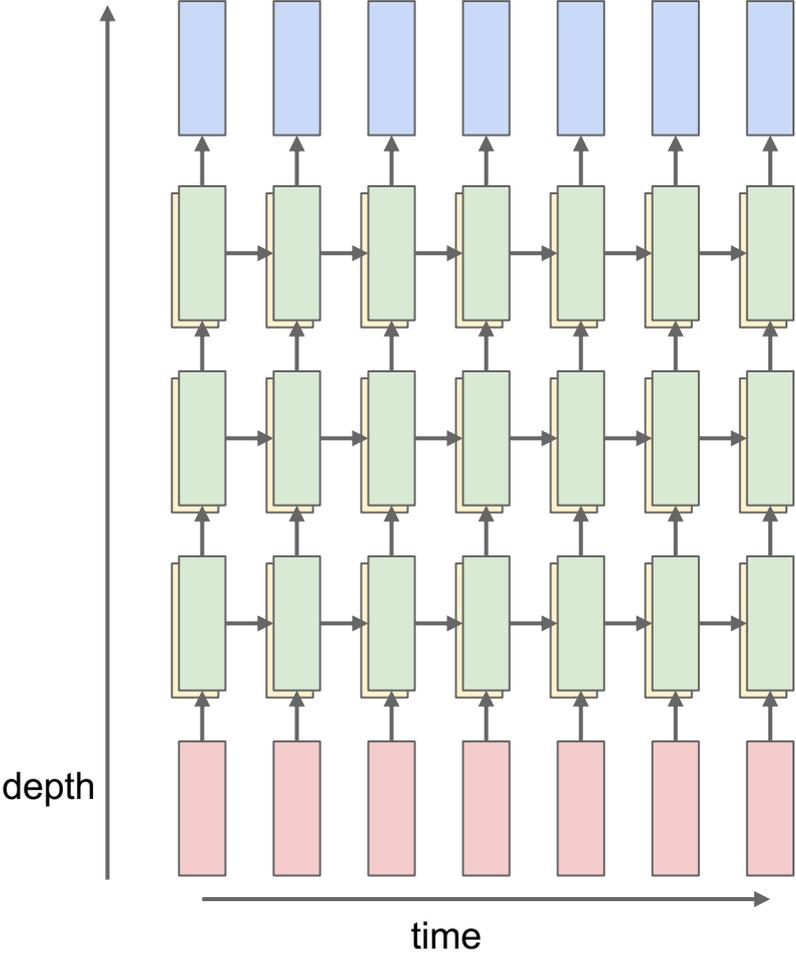
# Truncated Backpropagation through time



# Truncated Backpropagation through time

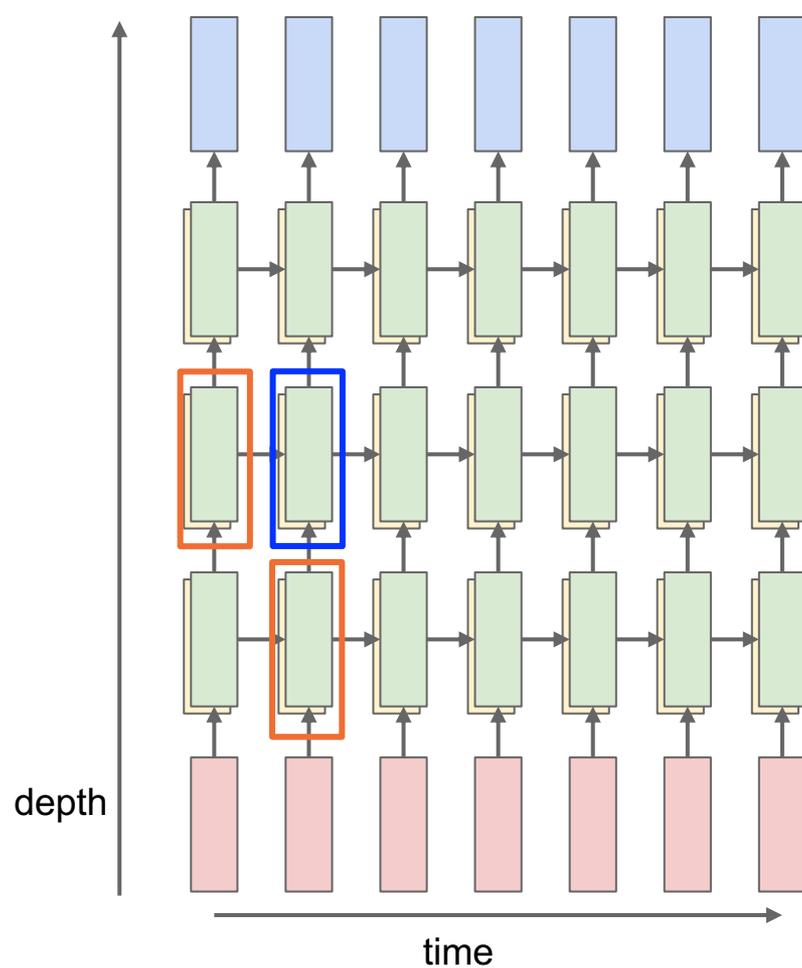
```
1 # pseudocode for Truncated BPTT
2 h = h_0
3 for c_i in num_chunk:
4     start_i, end_i = c_i * chunk_size, (c_i + 1) * chunk_size
5     for i in range(start_i, end_i):
6         h = f_h(x[i], h)
7         y_pred[i] = f_y(h)
8     L = compute_loss(y_pred[start_i:end_i], y[start_i:end_i])
9     backprop_update(L, [f_h, f_y])
10    clear_gradient([f_h, f_y])
```

# Multilayer RNNs



# Multilayer RNNs

Each RNN layer takes as input (1) previous hidden state from the same layer and (2) the output of the previous layer at the same timestep (or the input).

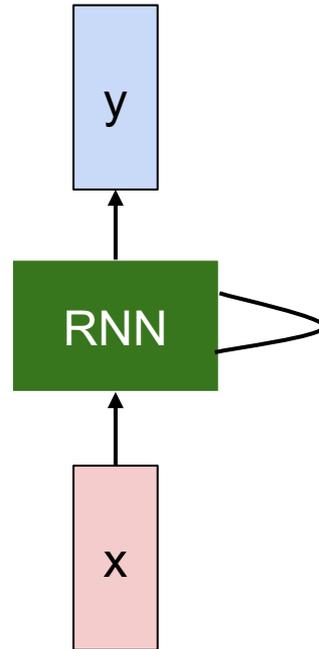


# THE SONNETS

by William Shakespeare

From fairest creatures we desire increase,  
That thereby beauty's rose might never die,  
But as the ripper should by time decease,  
His tender heir might bear his memory:  
But thou, contracted to thine own bright eyes,  
Feed'st thy light's flame with self-substantial fuel,  
Making a famine where abundance lies,  
Thyself thy foe, to thy sweet self too cruel:  
Thou that art now the world's fresh ornament,  
And only herald to the gaudy spring,  
Within thine own bud buriest thy content,  
And tender churl mak'st waste in niggarding:  
    Pity the world, or else this glutton be,  
    To eat the world's due, by the grave and thee.

When forty winters shall besiege thy brow,  
And dig deep trenches in thy beauty's field,  
Thy youth's proud livery so gazed on now,  
Will be a tatter'd weed of small worth held:  
Then being asked, where all thy beauty lies,  
Where all the treasure of thy lusty days;  
To say, within thine own deep sunken eyes,  
Were an all-eating shame, and thriftless praise.  
How much more praise deserv'd thy beauty's use,  
If thou couldst answer 'This fair child of mine  
Shall sum my count, and make my old excuse,'  
Proving his beauty by succession thine!  
    This were to be new made when thou art old,  
    And see thy blood warm when thou feel'st it cold.



at first:

tyntd-iafhatawiaoihrdemot lytdws e ,tfti, astai f ogoh eoase rrranbyne 'nhtnee e  
plia tklrgrd t o idoe ns,smtt h ne etie h,hregtrs nigtkie,aoaenns lng



train more

"Tmont thithey" fomesscerliund  
Keushey. Thom here  
sheulke, anmerenith ol sivh I lalterthend Bleipile shuw y 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 offer.



train more

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

PANDARUS:

Alas, I think he shall be come approached and the day  
When little 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 nudes 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.

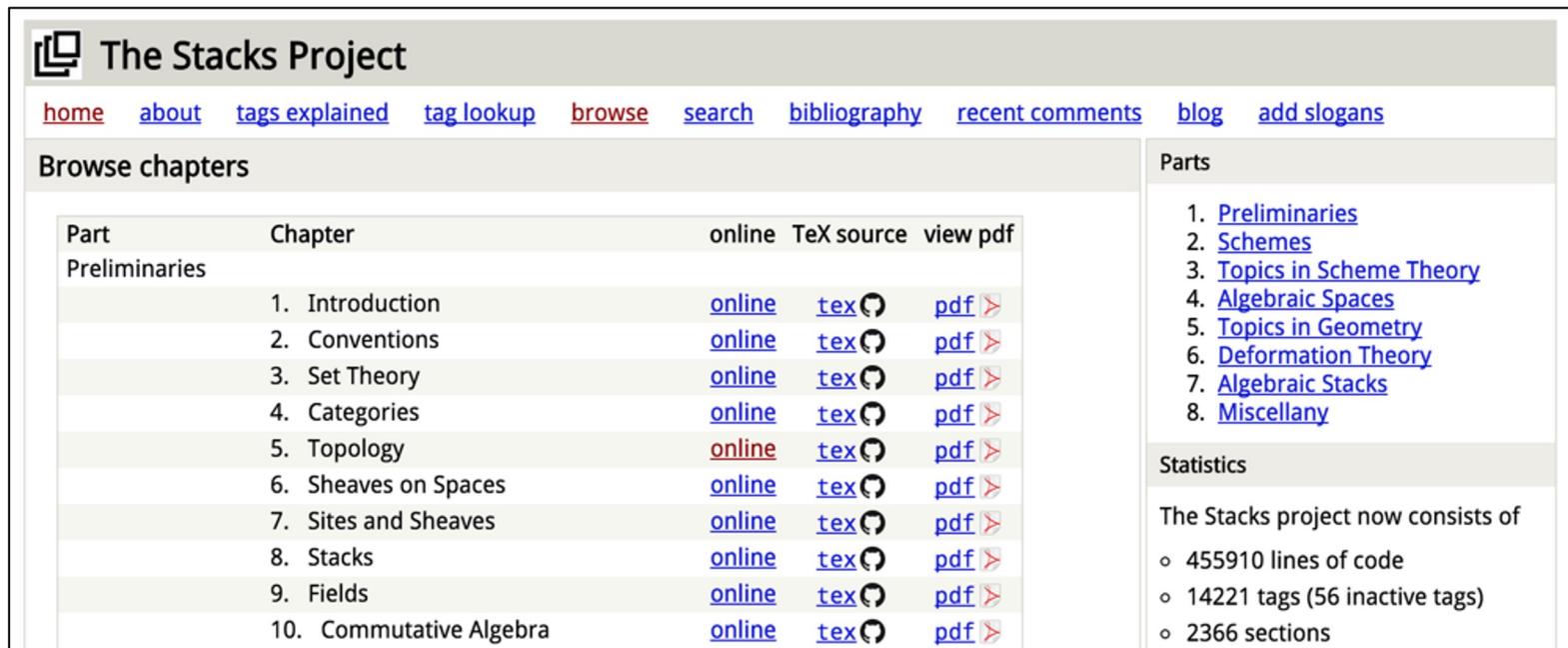
VIOLA:

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

KING LEAR:

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

# The Stacks Project: open source algebraic geometry textbook



The Stacks Project

[home](#) [about](#) [tags explained](#) [tag lookup](#) [browse](#) [search](#) [bibliography](#) [recent comments](#) [blog](#) [add slogans](#)

**Browse chapters**

Part	Chapter	online	TeX source	view pdf
Preliminaries				
	1. Introduction	<a href="#">online</a>	<a href="#">tex</a>	<a href="#">pdf</a>
	2. Conventions	<a href="#">online</a>	<a href="#">tex</a>	<a href="#">pdf</a>
	3. Set Theory	<a href="#">online</a>	<a href="#">tex</a>	<a href="#">pdf</a>
	4. Categories	<a href="#">online</a>	<a href="#">tex</a>	<a href="#">pdf</a>
	5. Topology	<a href="#">online</a>	<a href="#">tex</a>	<a href="#">pdf</a>
	6. Sheaves on Spaces	<a href="#">online</a>	<a href="#">tex</a>	<a href="#">pdf</a>
	7. Sites and Sheaves	<a href="#">online</a>	<a href="#">tex</a>	<a href="#">pdf</a>
	8. Stacks	<a href="#">online</a>	<a href="#">tex</a>	<a href="#">pdf</a>
	9. Fields	<a href="#">online</a>	<a href="#">tex</a>	<a href="#">pdf</a>
	10. Commutative Algebra	<a href="#">online</a>	<a href="#">tex</a>	<a href="#">pdf</a>

**Parts**

- [Preliminaries](#)
- [Schemes](#)
- [Topics in Scheme Theory](#)
- [Algebraic Spaces](#)
- [Topics in Geometry](#)
- [Deformation Theory](#)
- [Algebraic Stacks](#)
- [Miscellany](#)

**Statistics**

The Stacks project now consists of

- 455910 lines of code
- 14221 tags (56 inactive tags)
- 2366 sections

Latex source



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

The stacks project is licensed under the [GNU Free Documentation License](#)

For  $\bigoplus_{n=1, \dots, m}$  where  $\mathcal{L}_{m_*} = 0$ , 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  $\text{Sch}_{fppf}$  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

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

and

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

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

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

Suppose  $X = \lim |X|$  (by the formal open covering  $X$  and a single map  $\text{Proj}_X(\mathcal{A}) = \text{Spec}(B)$  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  $\mathcal{F}_{x_0} = \mathcal{F}_{x_0} = \mathcal{F}_{X, \dots, 0}$ .

**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  $i_0 \leq \mathfrak{p}$  is a subset of  $\mathcal{J}_{n,0} \circ \mathcal{A}_2$  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 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  $\delta_{n+1}$  is a scheme over  $S$ .  $\square$

*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

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

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

**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

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

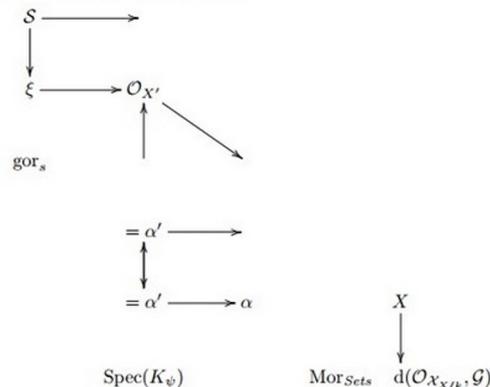
be a morphism of algebraic spaces over  $S$  and  $Y$ .

*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  $\mathcal{O}_X(\mathcal{U})$  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  $X = \text{Spec}(R)$  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  $U$ . □

*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"

$$\mathcal{O}_{X,x} \rightarrow \mathcal{F}_{\bar{x}}^{-1}(\mathcal{O}_{X_{\acute{e}tale}}) \rightarrow \mathcal{O}_{X_t}^{-1} \mathcal{O}_{X_\lambda}(\mathcal{O}_{X_\eta}^{\bar{v}})$$

is an isomorphism of covering of  $\mathcal{O}_{X_t}$ . 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.



This repository Search

Explore Gist Blog Help



karpthy



torvalds / linux

Watch -

3,711

★ Star

23,054

Fork

9,141

## Linux kernel source tree

520,037 commits

1 branch

420 releases

5,039 contributors



branch: master -

linux / +



Merge branch 'drm-fixes' of git://people.freedesktop.org/~airlied/linux



torvalds authored 9 hours ago

latest commit 4b1706927d

Documentation

Merge git://git.kernel.org/pub/scm/linux/kernel/git/hab/target-pending

6 days ago

arch

Merge branch 'x86-urgent-for-linus' of git://git.kernel.org/pub/scm/l...

a day ago

block

block: discard bdi\_unregister() in favour of bdi\_destroy()

9 days ago

crypto

Merge git://git.kernel.org/pub/scm/linux/kernel/git/herbert/crypto-2.6

10 days ago

drivers

Merge branch 'drm-fixes' of git://people.freedesktop.org/~airlied/linux

9 hours ago

firmware

firmware/ihex2fw.c: restore missing default in switch statement

2 months ago

fs

vfs: read file\_handle only once in handle\_to\_path

4 days ago

include

Merge branch 'perf-urgent-for-linus' of git://git.kernel.org/pub/scm/...

a day ago

init

init: fix regression by supporting devices with major:minor:offset fo...

a month ago

ipc

Merge branch 'for-linus' of git://git.kernel.org/pub/scm/linux/kernel...

a month ago



Code



Pull requests

74



Pulse



Graphs

HTTPS clone URL

https://github.c



You can clone with [HTTPS](#), [SSH](#), or [Subversion](#).



Clone in Desktop



Download ZIP

```

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 << i))
            pipe = (in_use & UMXTHREAD_UNCCA) +
                ((count & 0x00000000fffffff8) & 0x0000000f) << 8;
        if (count == 0)
            sub(pid, ppc_md.kexec_handle, 0x20000000);
        pipe_set_bytes(i, 0);
    }
    /* Free our user pages pointer to place camera if all dash */
    subsystem_info = &of_changes[PAGE_SIZE];
    rek_controls(offset, idx, &soffset);
    /* Now we want to deliberately put it to device */
    control_check_polarity(&context, val, 0);
    for (i = 0; i < COUNTER; i++)
        seq_puts(s, "policy ");
}

```

# Generated C code

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

```
#include <linux/kexec.h>
#include <linux/errno.h>
#include <linux/io.h>
#include <linux/platform_device.h>
#include <linux/multi.h>
#include <linux/ckevent.h>
```

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

```

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

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

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

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

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

```

# Image Captioning

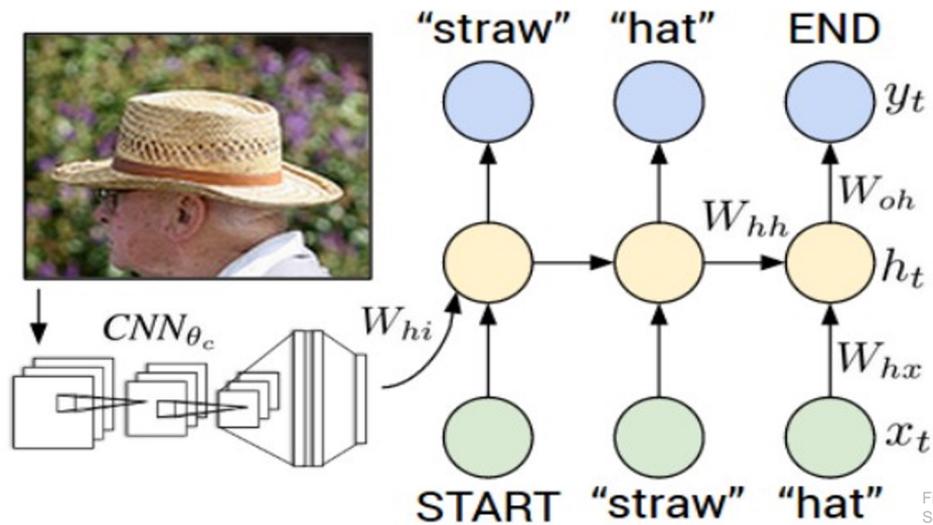


Figure from Karpathy et al, "Deep Visual-Semantic Alignments for Generating Image Descriptions", CVPR 2015; figure copyright IEEE, 2015.

Reproduced for educational purposes.

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

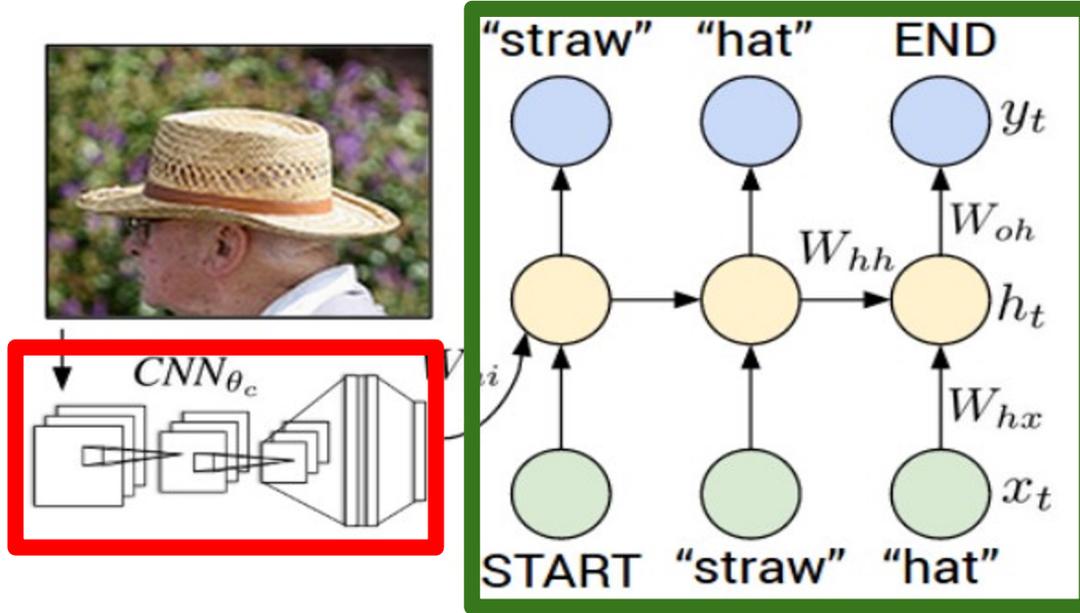
Deep Visual-Semantic Alignments for Generating Image Descriptions, Karpathy and Fei-Fei

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

# Recurrent Neural Network



# Convolutional Neural Network



test image

[This image](#) is [CC0 public domain](#)

image

conv-64

conv-64

maxpool

conv-128

conv-128

maxpool

conv-256

conv-256

maxpool

conv-512

conv-512

maxpool

conv-512

conv-512

maxpool

FC-4096

FC-4096

FC-1000

softmax



test image

image

conv-64

conv-64

maxpool

conv-128

conv-128

maxpool

conv-256

conv-256

maxpool

conv-512

conv-512

maxpool

conv-512

conv-512

maxpool

FC-4096

FC-4096

FC-1000

softmax



test image





test image

image

conv-64

conv-64

maxpool

conv-128

conv-128

maxpool

conv-256

conv-256

maxpool

conv-512

conv-512

maxpool

conv-512

conv-512

maxpool

FC-4096

FC-4096

x0  
<START>

image

conv-64

conv-64

maxpool

conv-128

conv-128

maxpool

conv-256

conv-256

maxpool

conv-512

conv-512

maxpool

conv-512

conv-512

maxpool

FC-4096

FC-4096

V



test image

y0

h0

x0  
<START>

W<sub>ih</sub>

**before:**

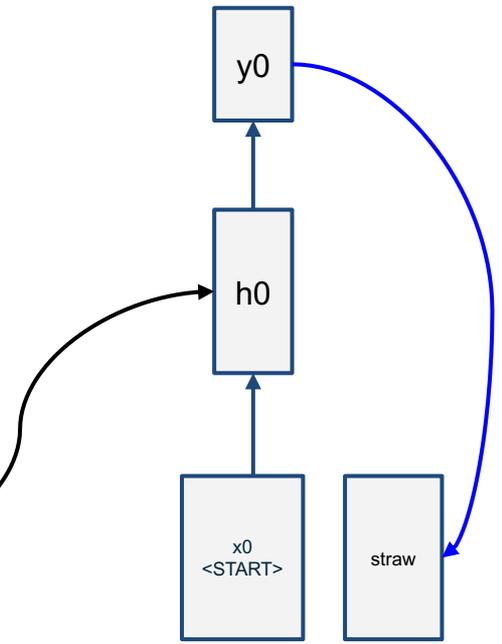
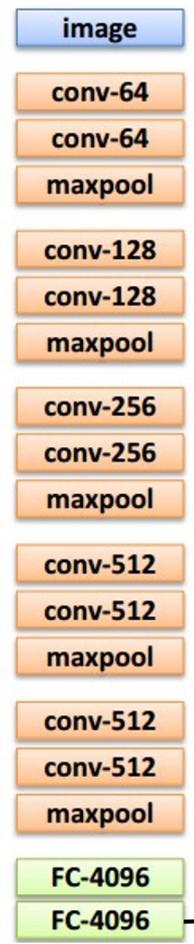
$$h = \tanh(W_{xh} * x + W_{hh} * h)$$

**now:**

$$h = \tanh(W_{xh} * x + W_{hh} * h + W_{ih} * v)$$



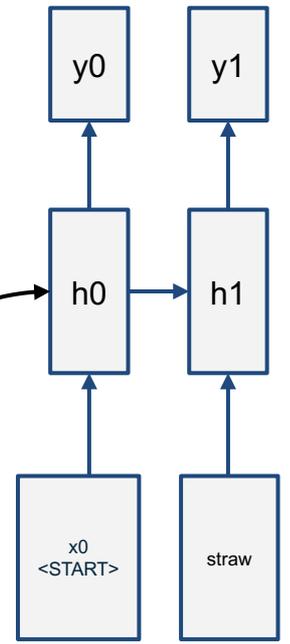
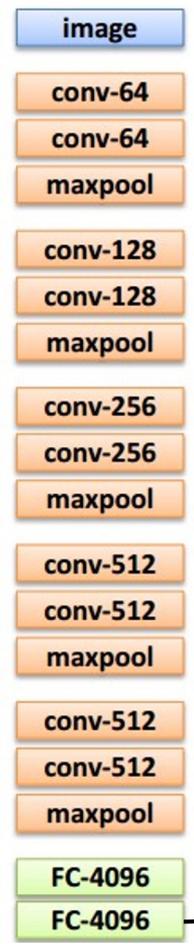
test image



sample!



test image



image

conv-64

conv-64

maxpool

conv-128

conv-128

maxpool

conv-256

conv-256

maxpool

conv-512

conv-512

maxpool

conv-512

conv-512

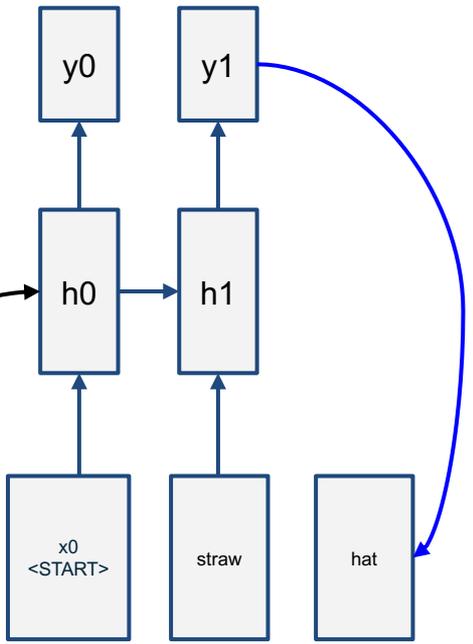
maxpool

FC-4096

FC-4096

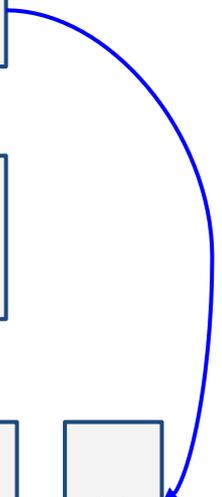
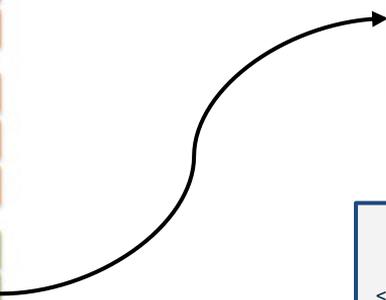


test image



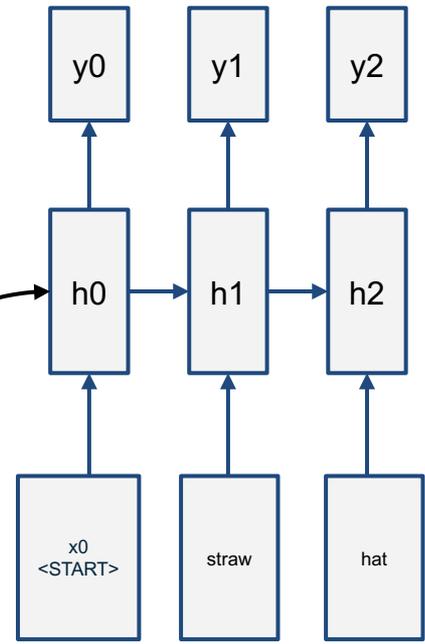
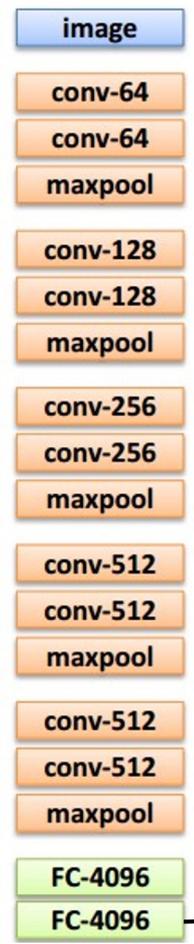
sample!

←





test image



image

conv-64

conv-64

maxpool

conv-128

conv-128

maxpool

conv-256

conv-256

maxpool

conv-512

conv-512

maxpool

conv-512

conv-512

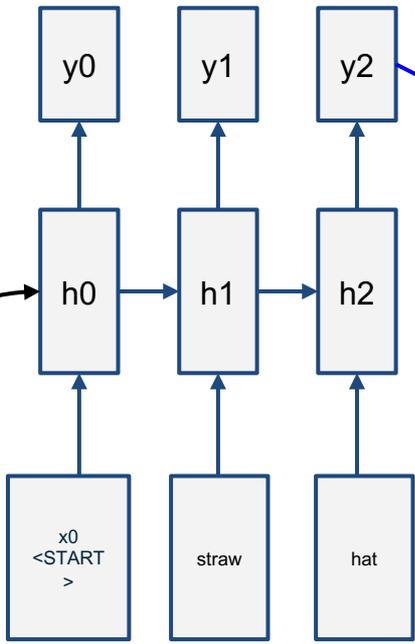
maxpool

FC-4096

FC-4096



test image



sample  
<END> token  
=> finish.

←

→

→

# Image Captioning: Example Results

Captions generated using  
[neuraltalk2](#)  
All images are [CC0 Public domain](#)  
[cat](#) [suitcase](#) [cat](#) [tree](#) [dog](#) [bear](#)  
[surfers](#) [tennis](#) [giraffe](#) [motorcycle](#)



*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*

# Image Captioning: Failure Cases

Captions generated using [neuraltalk2](#)  
All images are [CC0 Public domain](#): [fur coat](#), [handstand](#), [spider web](#), [baseball](#)



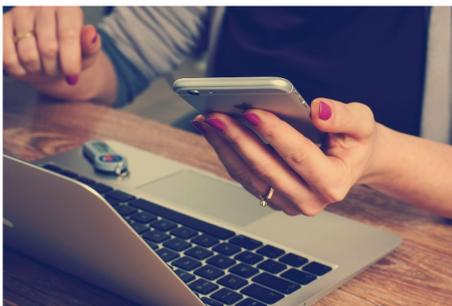
*A woman is holding a cat in her hand*



*A woman standing on a beach holding a surfboard*



*A bird is perched on a tree branch*



*A person holding a computer mouse on a desk*



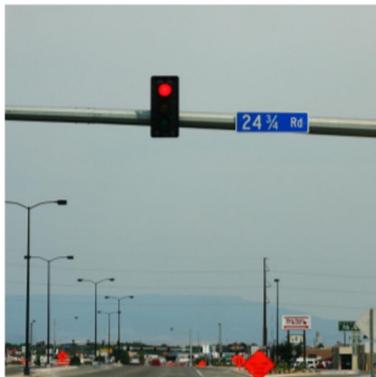
*A man in a baseball uniform throwing a ball*

# Visual Question Answering (VQA)



**Q: What endangered animal is featured on the truck?**

- A: A bald eagle.**
- A: A sparrow.
- A: A humming bird.
- A: A raven.



**Q: Where will the driver go if turning right?**

- A: Onto 24 3/4 Rd.**
- A: Onto 25 3/4 Rd.
- A: Onto 23 3/4 Rd.
- A: Onto Main Street.



**Q: When was the picture taken?**

- A: During a wedding.**
- A: During a bar mitzvah.
- A: During a funeral.
- A: During a Sunday church service



**Q: Who is under the umbrella?**

- A: Two women.**
- A: A child.
- A: An old man.
- A: A husband and a wife.

# Visual Dialog: Conversations about images

**Visual Dialog**



A cat drinking water out of a coffee mug.

What color is the mug?

White and red

Are there any pictures on it?

No, something is there can't tell what it is

Is the mug and cat on a table?

Yes, they are

Are there other items on the table?

Yes, magazines, books, toaster and basket, and a plate

Start typing question here ...

# Visual Language Navigation: Go to the living room

Agent encodes instructions in language and uses an RNN to generate a series of movements as the visual input changes after each move.

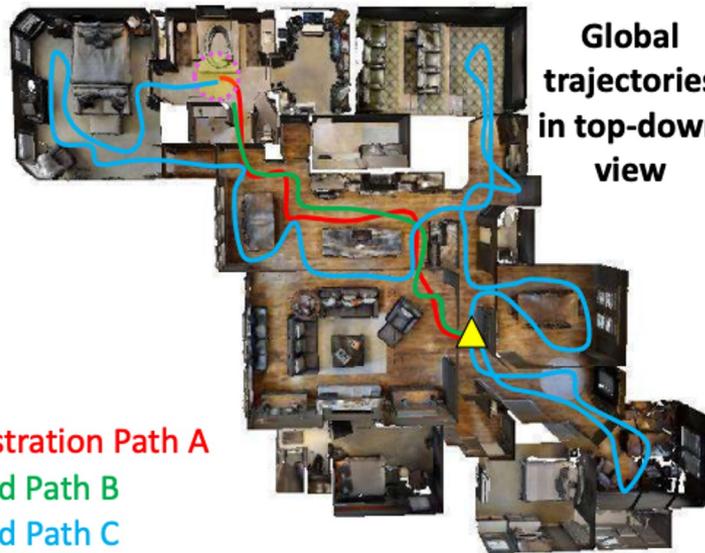
## Instruction

Turn right and head towards the *kitchen*. Then turn left, pass a *table* and enter the *hallway*. Walk down the hallway and turn into the *entry way* to your right *without doors*. Stop in front of the *toilet*.

## Local visual scene



## Global trajectories in top-down view



 Initial Position

 Target Position

 Demonstration Path A

 Executed Path B

 Executed Path C

# RNN tradeoffs

## RNN Advantages:

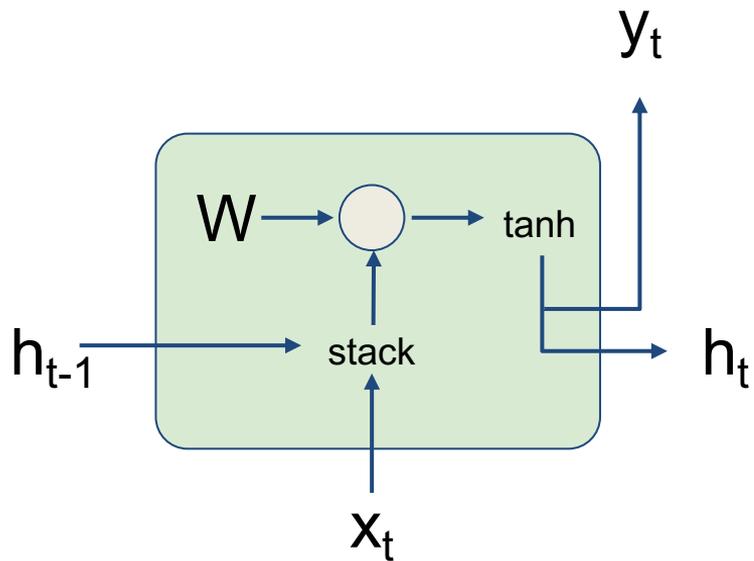
- Can process any length input
- Computation for step  $t$  can (in theory) use information from many steps back
- Model size doesn't increase for longer input
- Same weights applied on every timestep, so there is symmetry in how inputs are processed.

## RNN Disadvantages:

- Recurrent computation is slow
- In practice, difficult to access information from many steps back
- **Vanishing gradient / gradient explosion**

# Vanilla RNN Gradient Flow

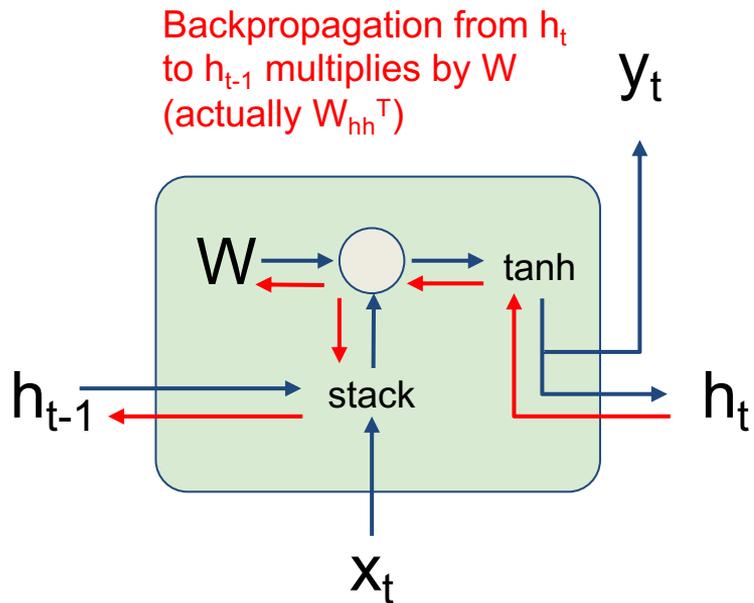
Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994  
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013



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

# Vanilla RNN Gradient Flow

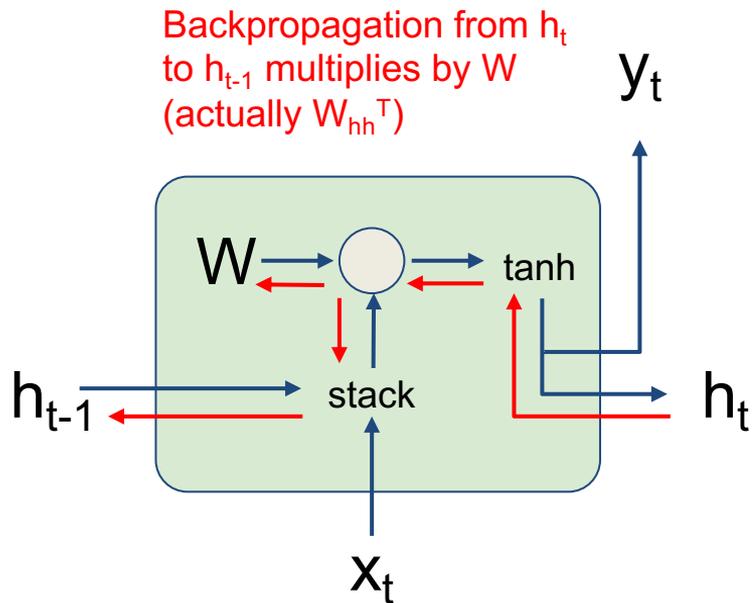
Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994  
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013



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

# 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

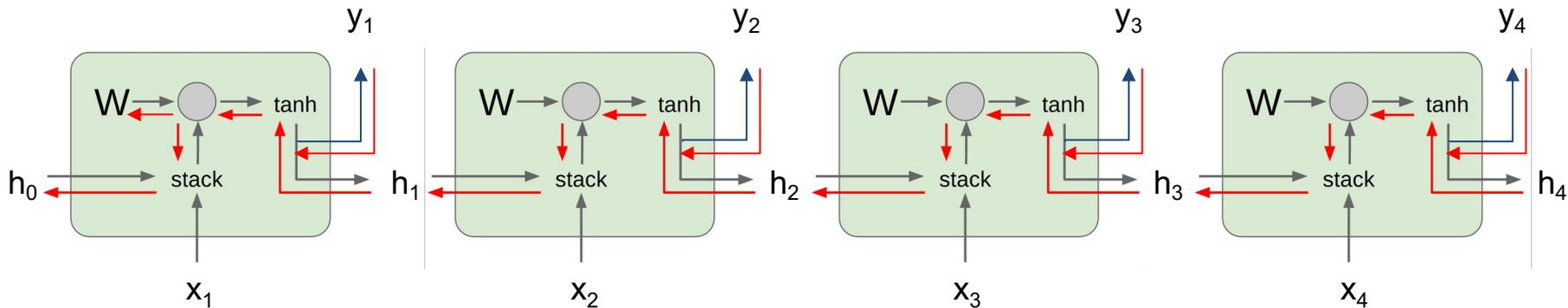


$$\begin{aligned}h_t &= \tanh(W_{hh}h_{t-1} + W_{xh}x_t) \\&= \tanh\left(\begin{pmatrix} W_{hh} & W_{hx} \end{pmatrix} \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right) \\&= \tanh\left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)\end{aligned}$$

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

# 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

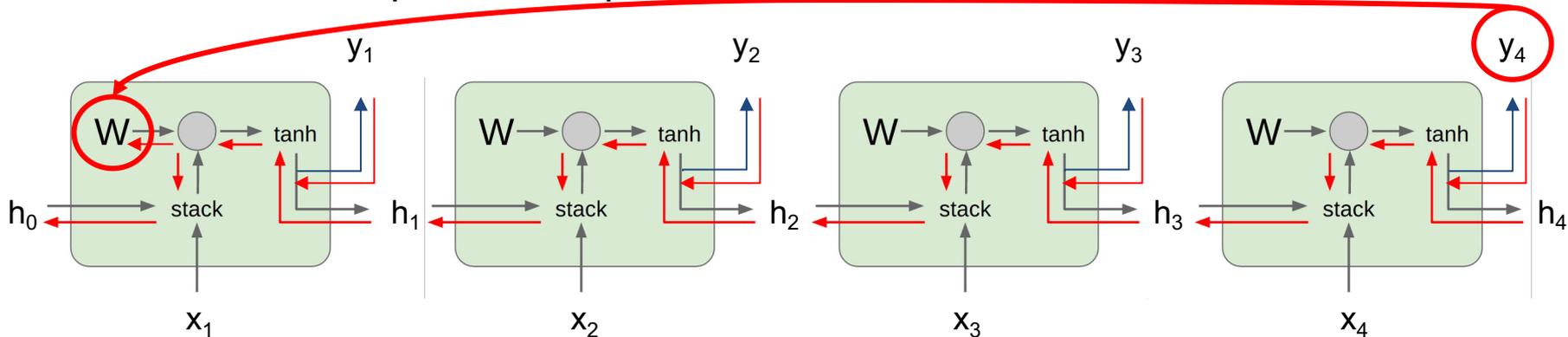


$$\frac{\partial L}{\partial W} = \sum_{t=1}^T \frac{\partial L_t}{\partial W}$$

# 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:



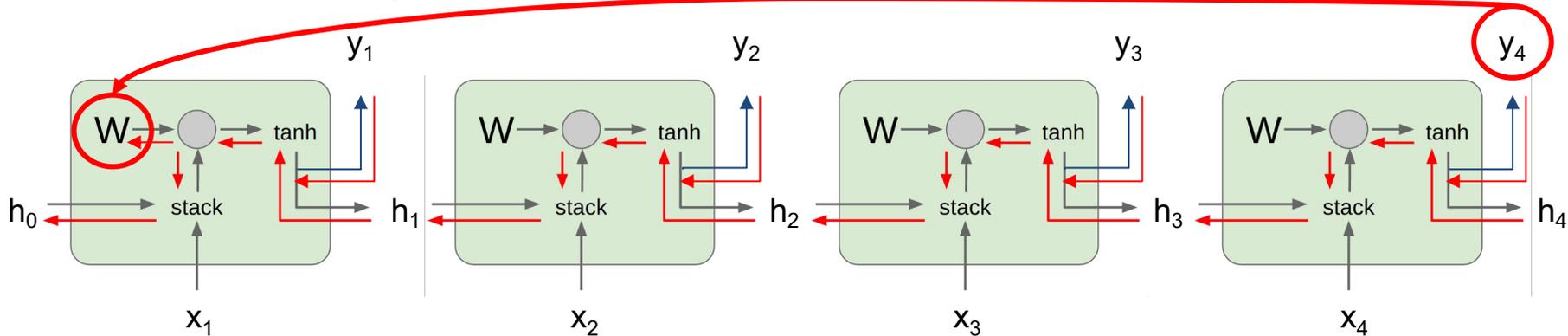
$$\frac{\partial L}{\partial W} = \sum_{t=1}^T \frac{\partial L_t}{\partial W}$$

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

# Vanilla RNN Gradient Flow

Gradients over multiple time steps:

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



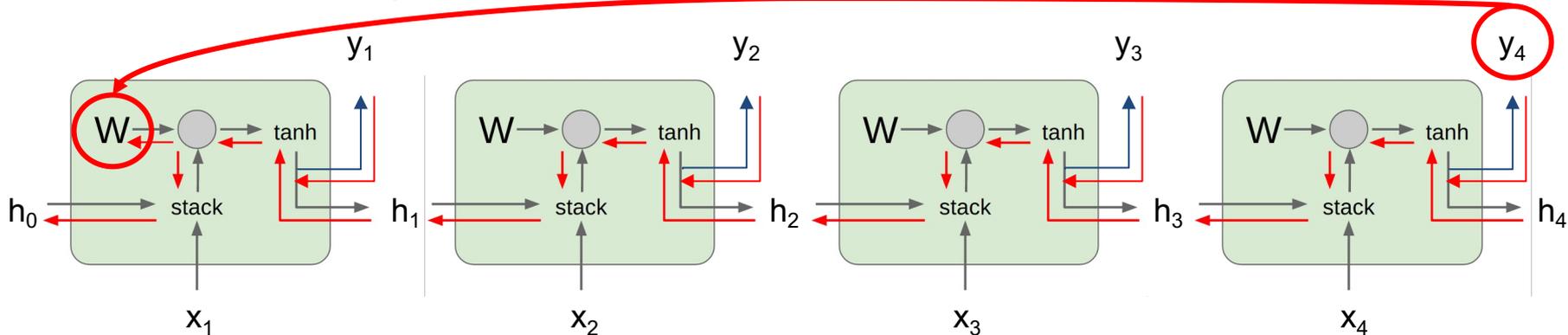
$$\frac{\partial L}{\partial W} = \sum_{t=1}^T \frac{\partial L_t}{\partial W}$$

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

# 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:



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

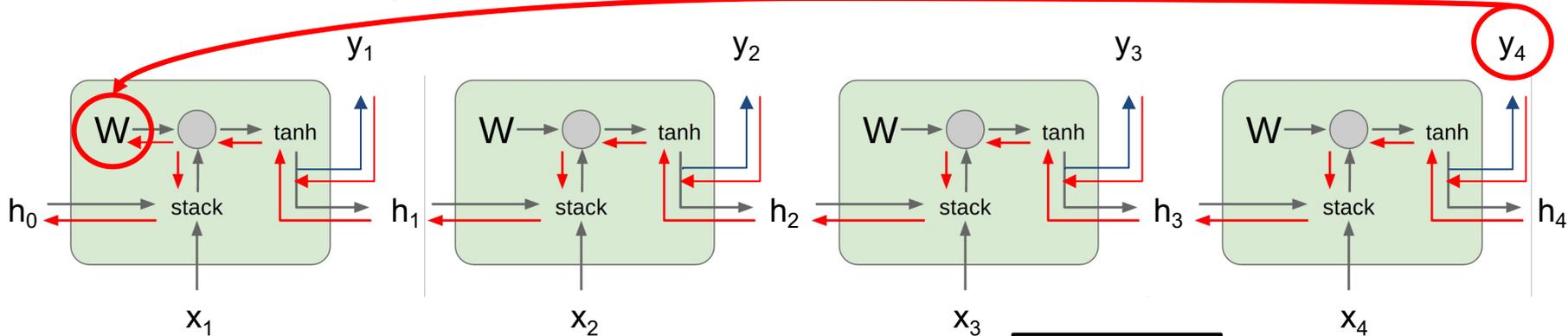
$$\frac{\partial h_t}{\partial h_{t-1}} = \tanh'(W_{hh} h_{t-1} + W_{xh} x_t) W_{hh}$$

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

# Vanilla RNN Gradient Flow

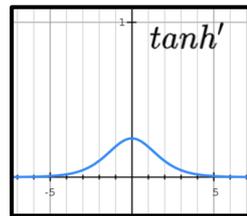
Gradients over multiple time steps:

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



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

Always < 1  
Vanishing gradients

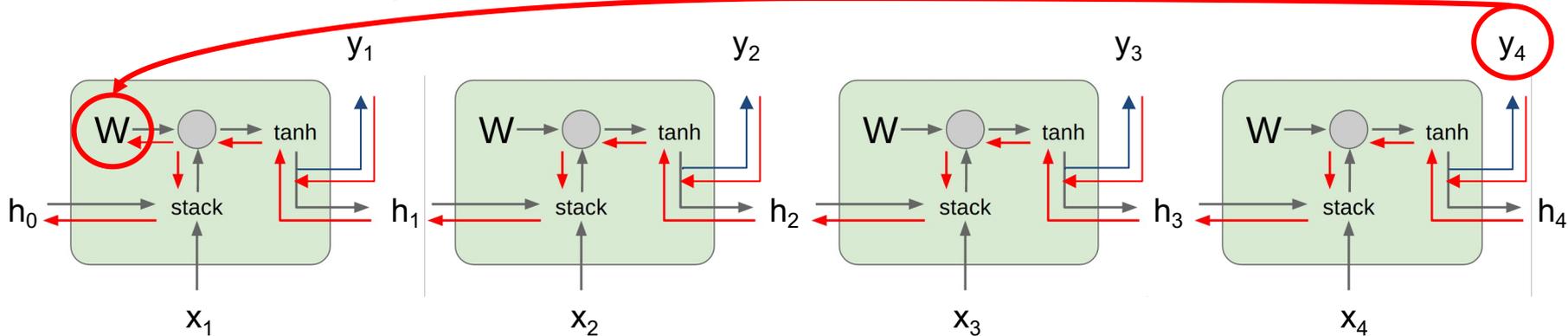


$$\frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial h_T} \left( \prod_{t=2}^T \tanh'(W_{hh} h_{t-1} + W_{xh} x_t) \right) W_{hh}^{T-1} \frac{\partial h_1}{\partial W}$$

# Vanilla RNN Gradient Flow

Gradients over multiple time steps:

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



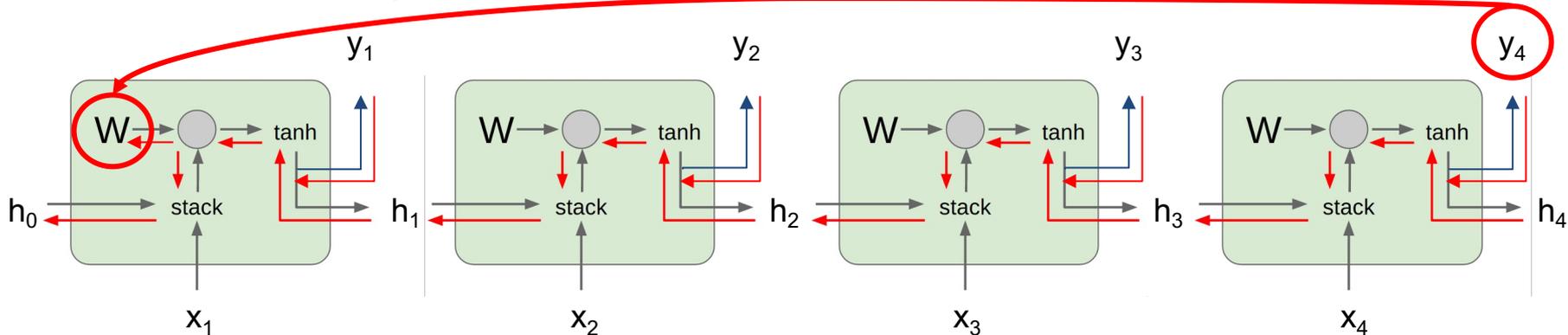
$$\frac{\partial L}{\partial W} = \sum_{t=1}^T \frac{\partial L_t}{\partial W}$$

What if we assumed no non-linearity?

# Vanilla RNN Gradient Flow

Gradients over multiple time steps:

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



What if we assumed no non-linearity?

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

Largest eigen value > 1:  
**Exploding gradients**

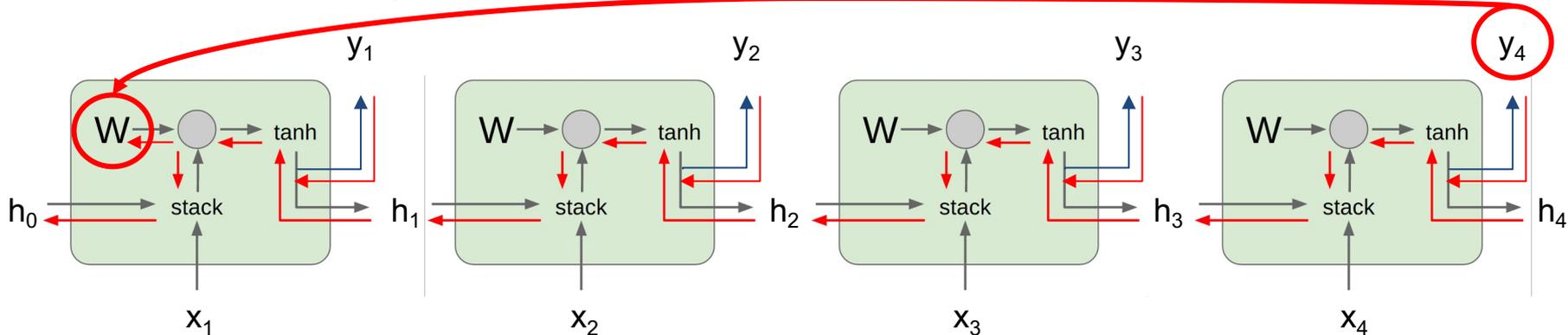
$$\frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial h_T} \boxed{W_{hh}^{T-1}} \frac{\partial h_1}{\partial W}$$

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

# Vanilla RNN Gradient Flow

Gradients over multiple time steps:

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



What if we assumed no non-linearity?

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

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

Largest eigen value > 1:  
**Exploding gradients**

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

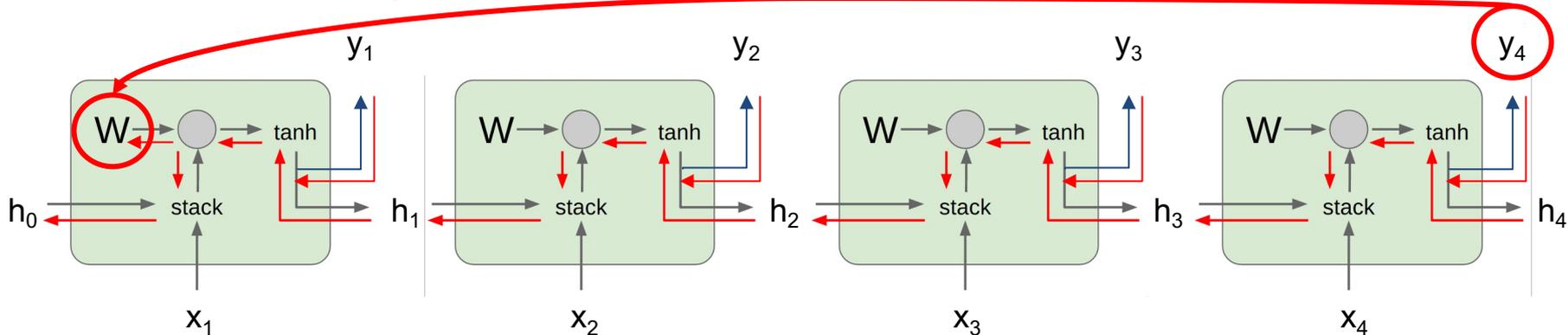
→ **Gradient clipping:**  
 Scale gradient if its norm is too big

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

# Vanilla RNN Gradient Flow

Gradients over multiple time steps:

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



What if we assumed no non-linearity?

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

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

→ We need a new RNN architecture!

# Long Short Term Memory (LSTM)

## Vanilla RNN

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

## LSTM

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

Learn to control information flow from previous state to the next state

# Long Short Term Memory (LSTM)

## Vanilla RNN

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

Long-term memory  $c$  determines how much information should go into the hidden state  $h$  (short-term memory)

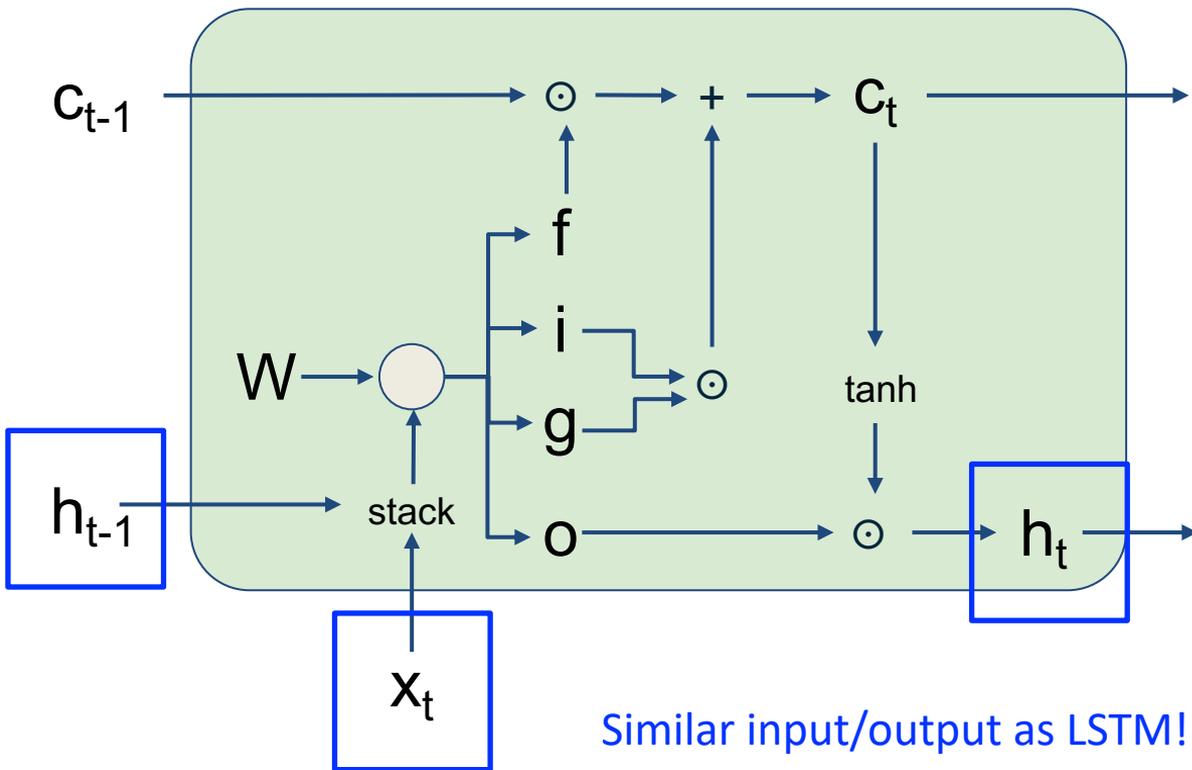
## LSTM

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

Two “memory vectors”

# Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]



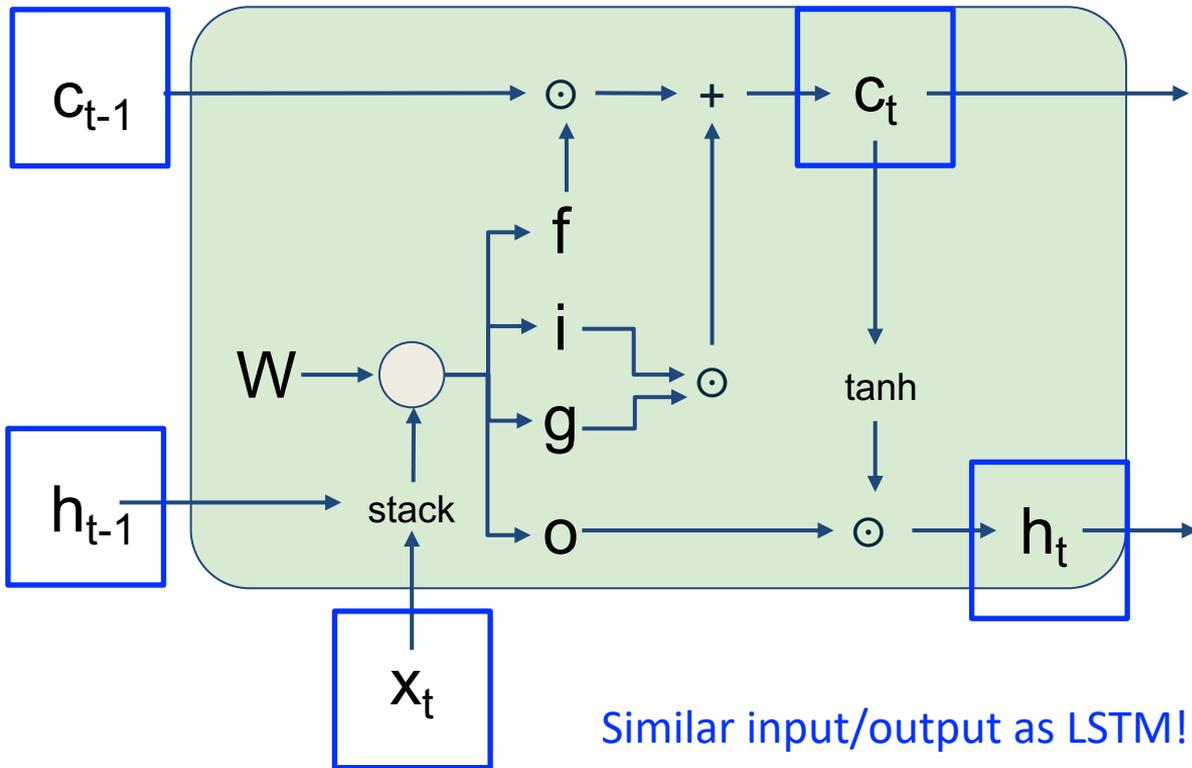
$$c_t = f_t \odot c_{t-1} + i_t \odot g_t$$
$$h_t = o_t \odot \tanh(c_t)$$

Similar input/output as LSTM!

# Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]

Keep long-term memory cell  $c$  in addition to short term memory  $h$

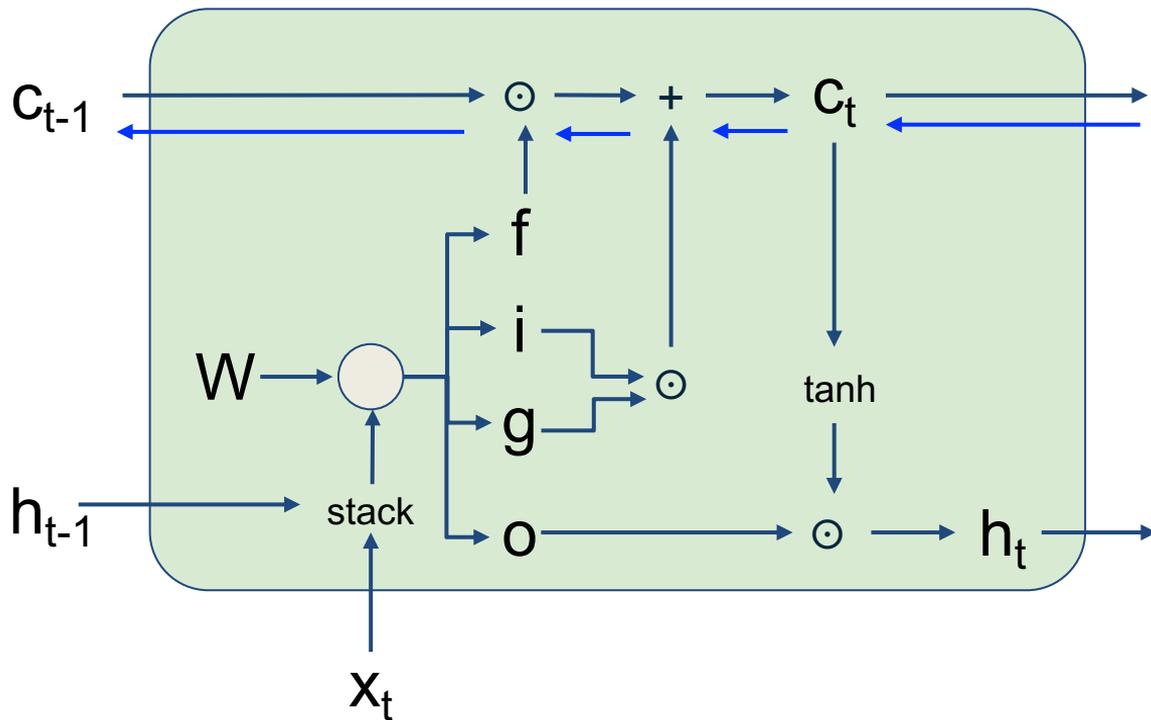


$$c_t = f_t \odot c_{t-1} + i_t \odot g_t$$
$$h_t = o_t \odot \tanh(c_t)$$

Similar input/output as LSTM!

# Long Short Term Memory (LSTM): Gradient Flow

[Hochreiter et al., 1997]



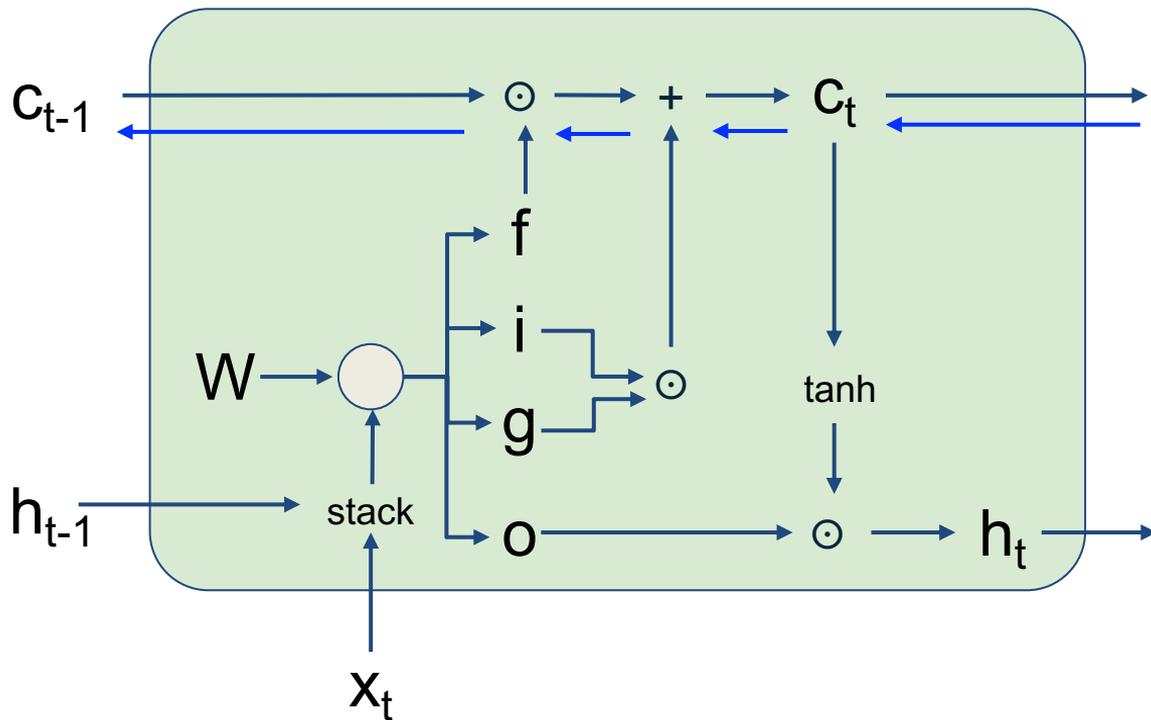
Backpropagation from  $c_t$  to  $c_{t-1}$   
only elementwise multiplication  
by  $f$  (forget gate), no matrix  
multiply by  $W$

$$c_t = f_t \odot c_{t-1} + i_t \odot g_t$$
$$h_t = o_t \odot \tanh(c_t)$$

$$\frac{\partial c_t}{\partial c_{t-1}} = ?$$

# Long Short Term Memory (LSTM): Gradient Flow

[Hochreiter et al., 1997]



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

$$c_t = f_t \odot c_{t-1} + i_t \odot g_t$$
$$h_t = o_t \odot \tanh(c_t)$$

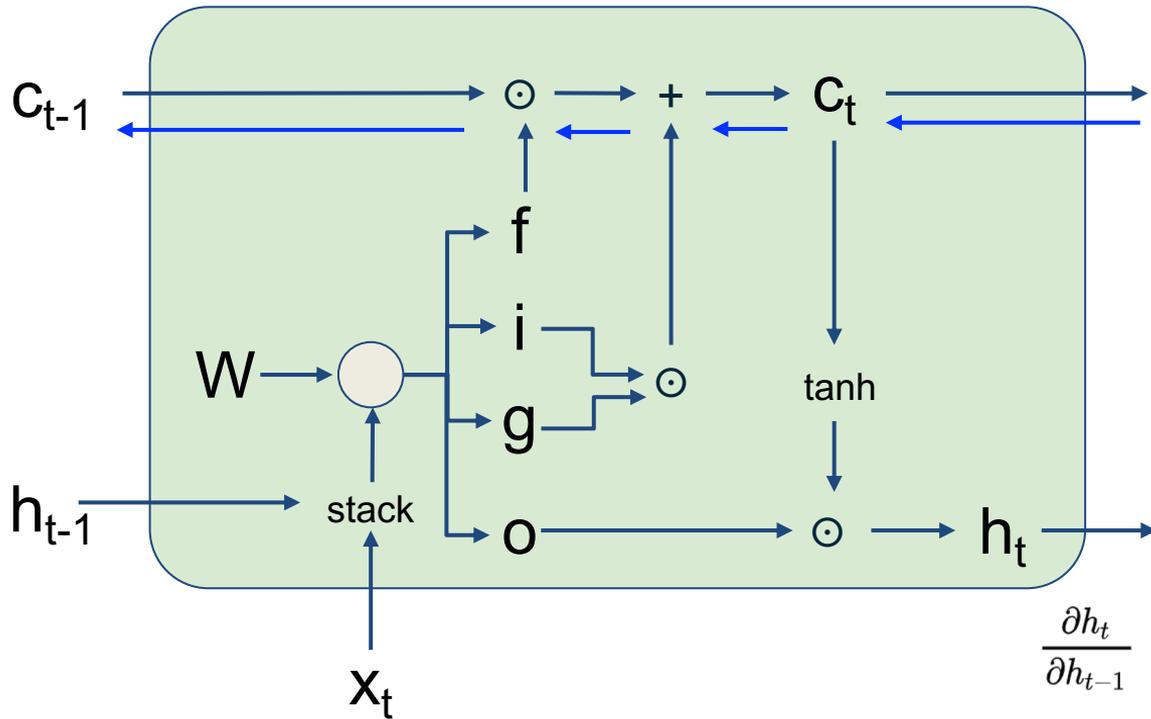
$$\frac{\partial c_t}{\partial c_{t-1}} = f_t \quad (\text{forget gate})$$

Different each step!

When  $f_t$  is close to 1, it allows  
gradient to flow back easily

# Long Short Term Memory (LSTM): Gradient Flow

[Hochreiter et al., 1997]



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

$$c_t = f_t \odot c_{t-1} + i_t \odot g_t$$
$$h_t = o_t \odot \tanh(c_t)$$

$$\frac{\partial c_t}{\partial c_{t-1}} = f_t \quad (\text{forget gate})$$

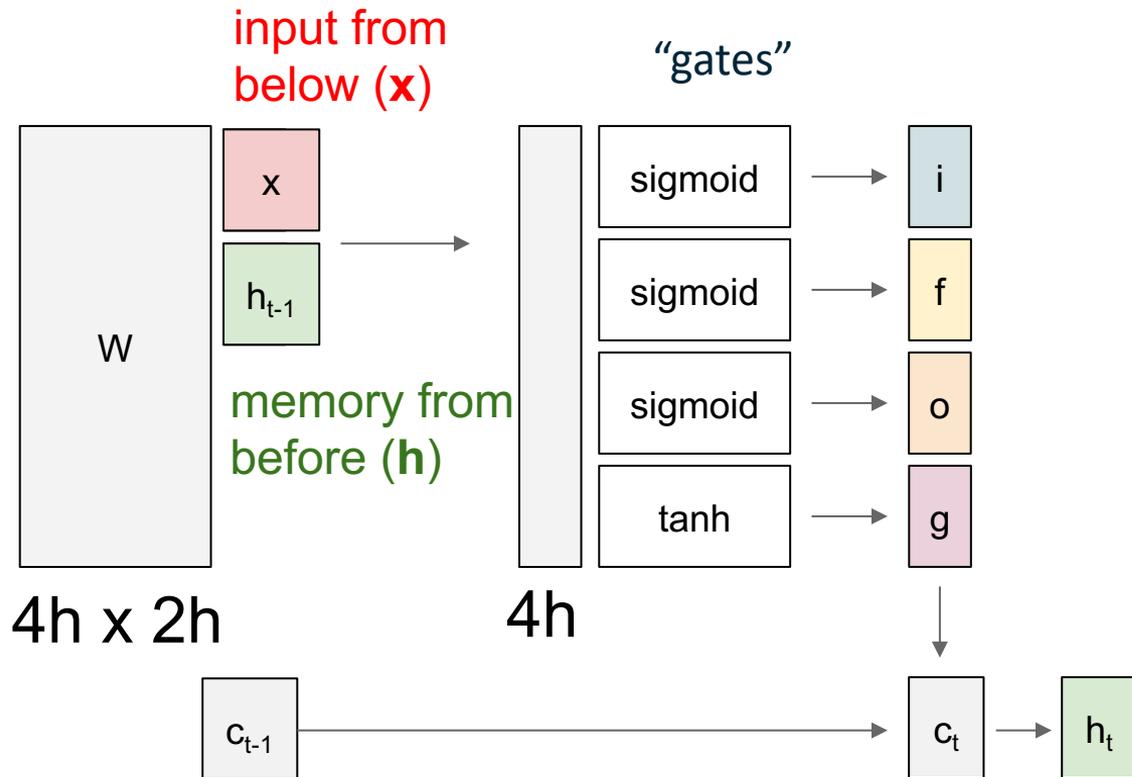
Gradient of RNN:

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

Same matrix every step, causes vanishing gradient

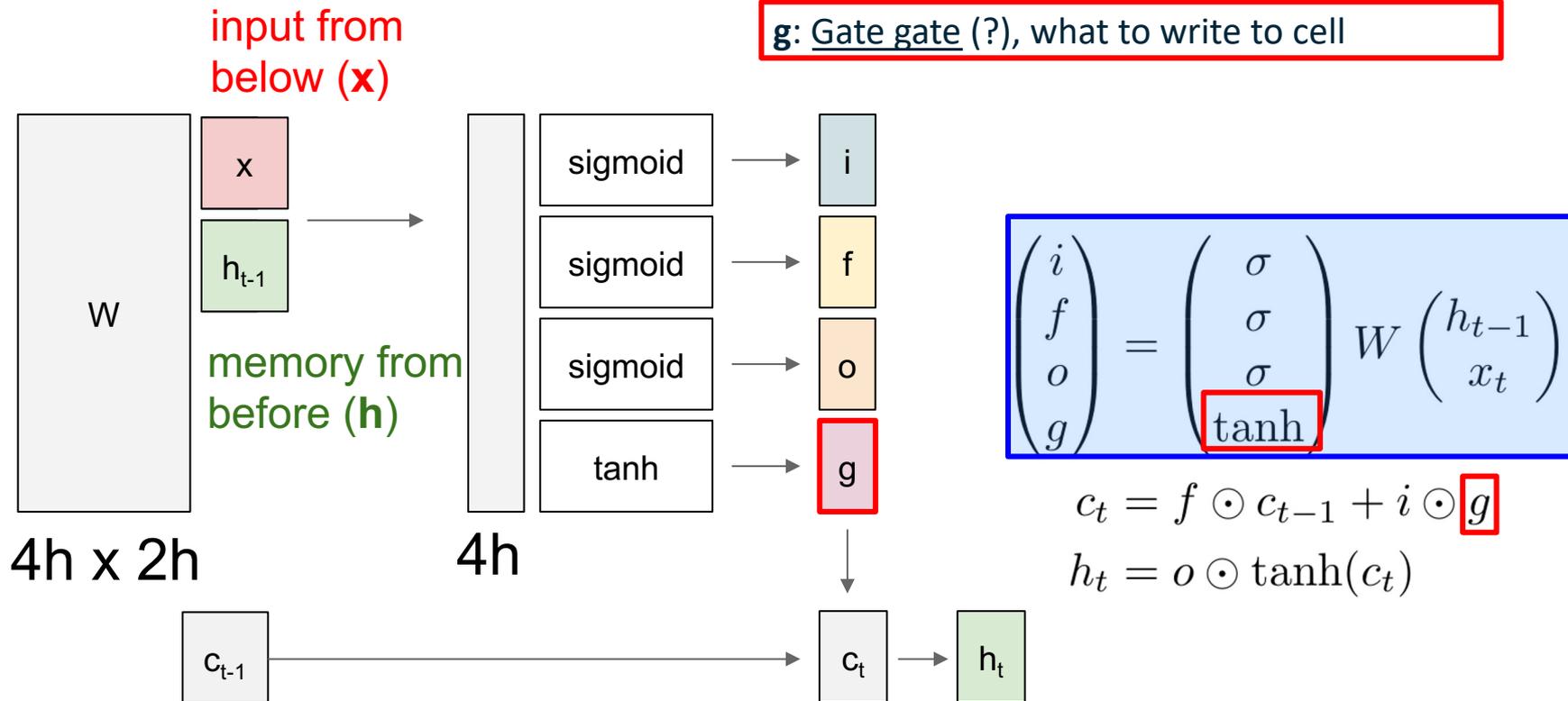
# Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]



# Long Short Term Memory (LSTM)

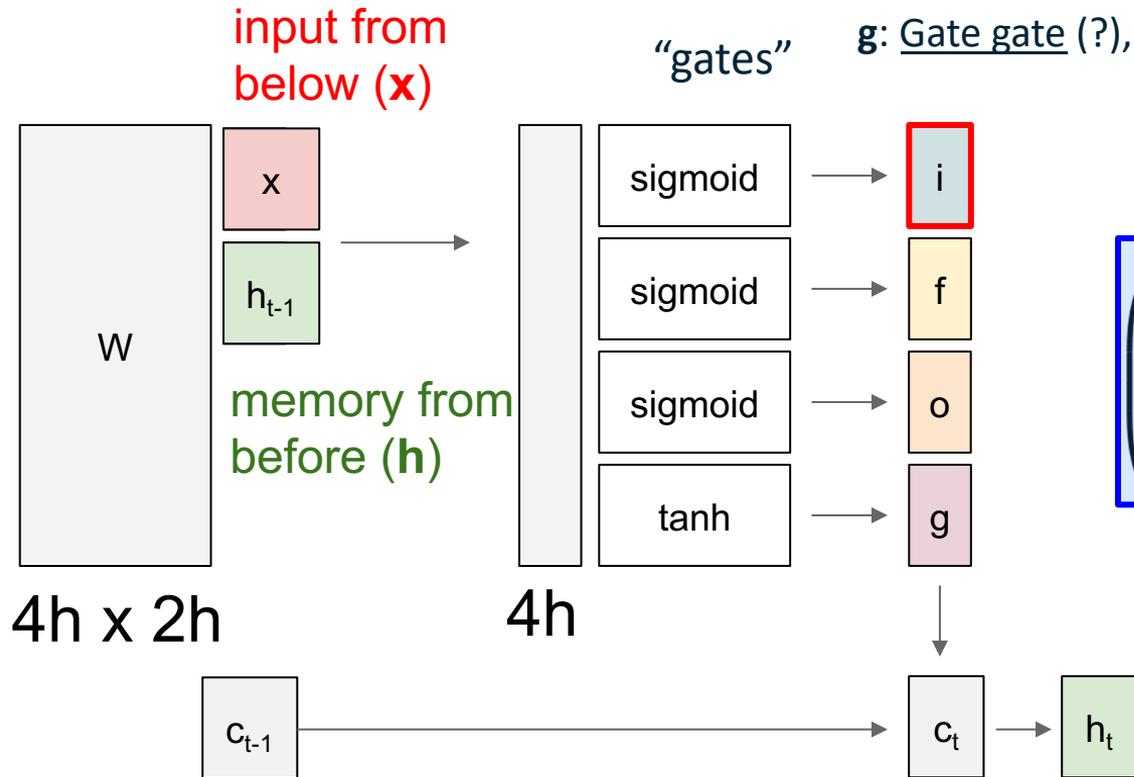
[Hochreiter et al., 1997]



# Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]

**i: Input gate, whether to write to cell**



**g: Gate gate (?), what to write to cell**

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

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

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

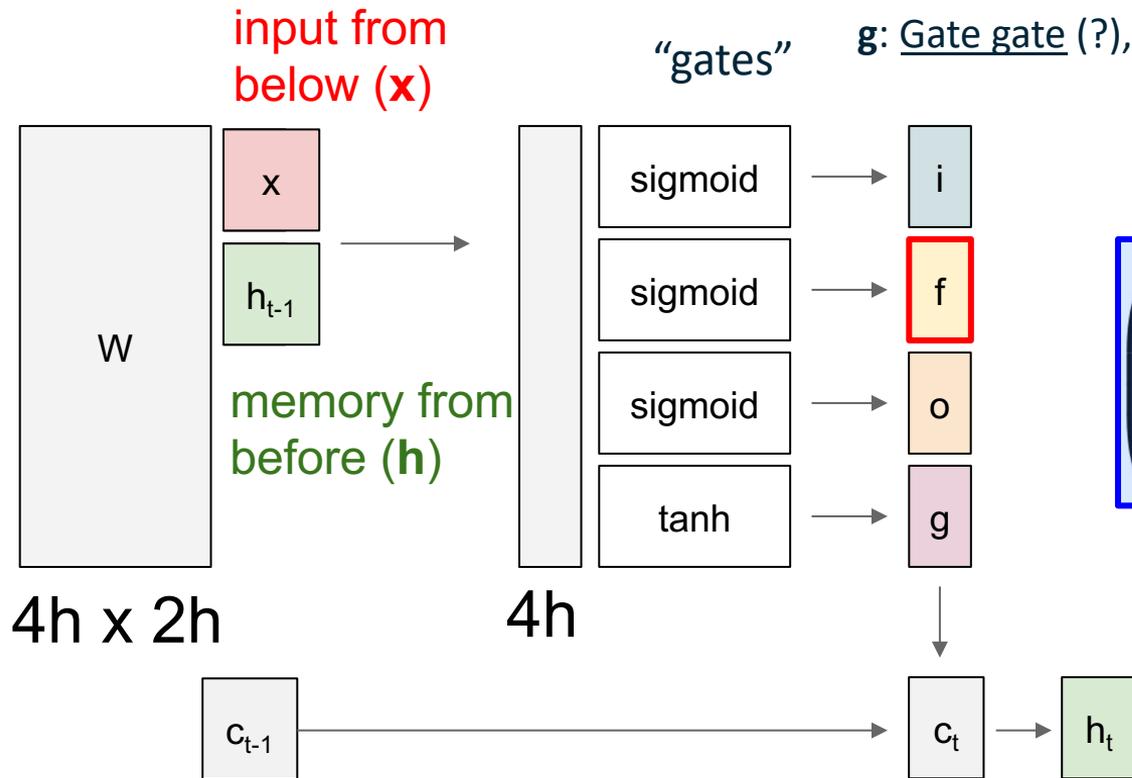
# Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]

i: Input gate, whether to write to cell

f: Forget gate, whether to erase cell

g: Gate gate (?), what 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)

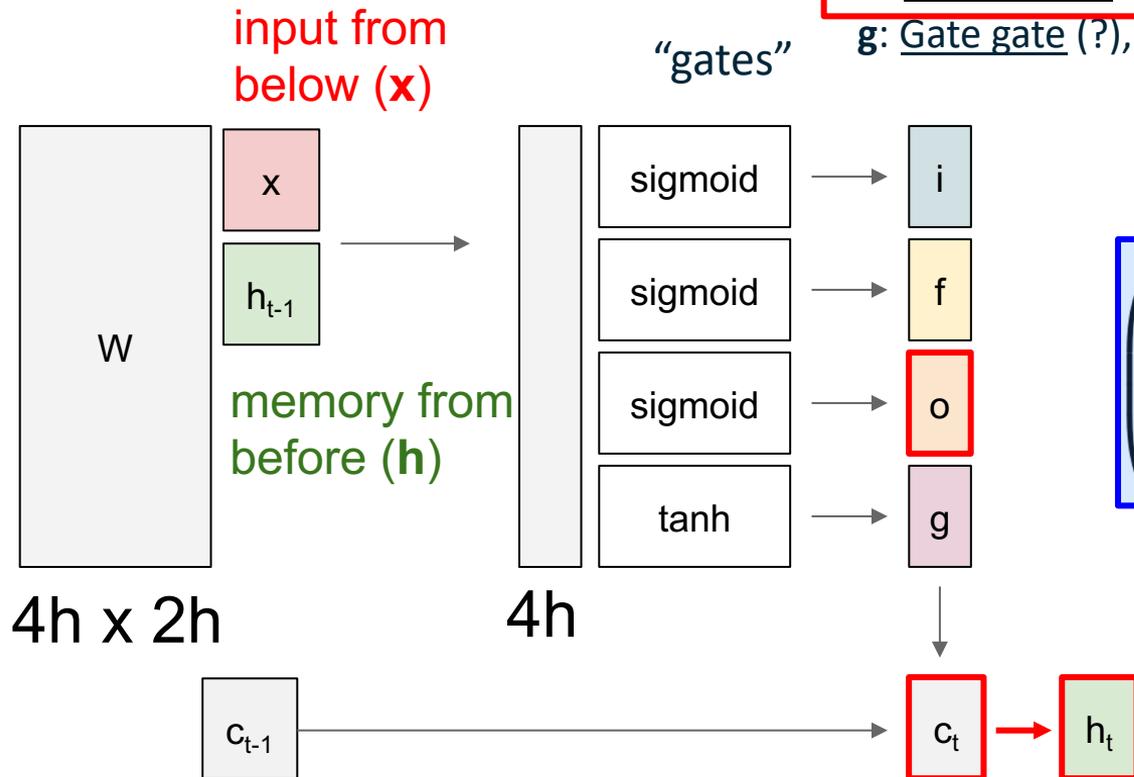
[Hochreiter et al., 1997]

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 (?), what to write to cell



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

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

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

# 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  $f = 1$  and  $i = 0$ , then the information of that cell is preserved indefinitely. Gradient flow back from cell  $c$  easily.
- By contrast, it's harder for vanilla RNN to learn a recurrent weight matrix  $W_h$  that preserves info in hidden state

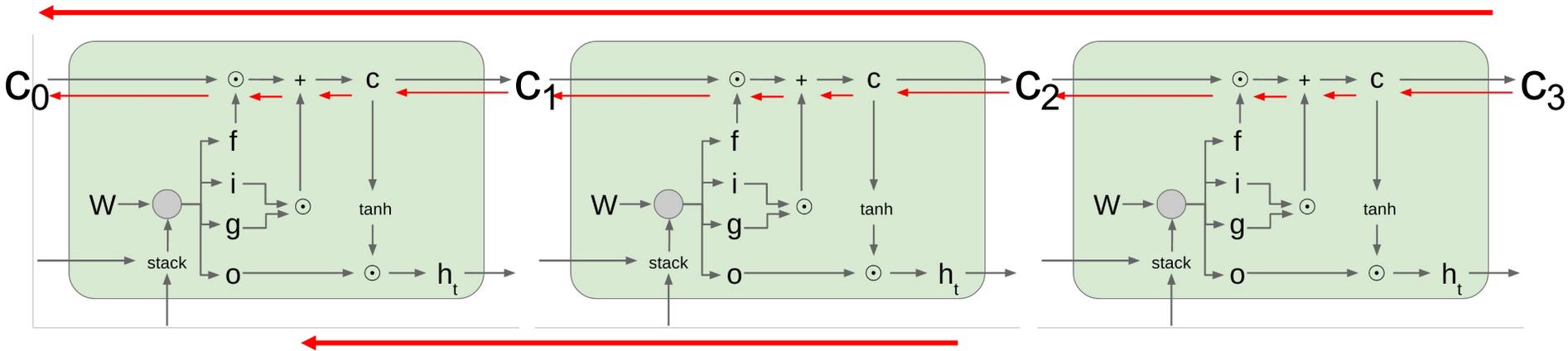
LSTM **doesn't guarantee** that there is no vanishing/exploding gradient, but it does provide an easier way for the model to learn long-distance dependencies.

It is possible to mitigate vanishing / exploding gradient by learning the correct  $f$

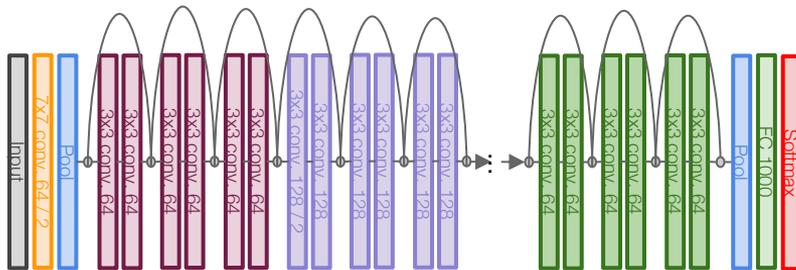
# Long Short Term Memory (LSTM): Gradient Flow

[Hochreiter et al., 1997]

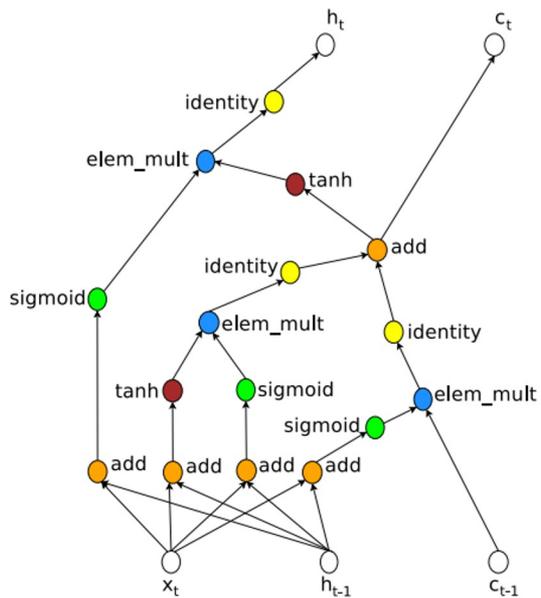
Uninterrupted gradient flow!



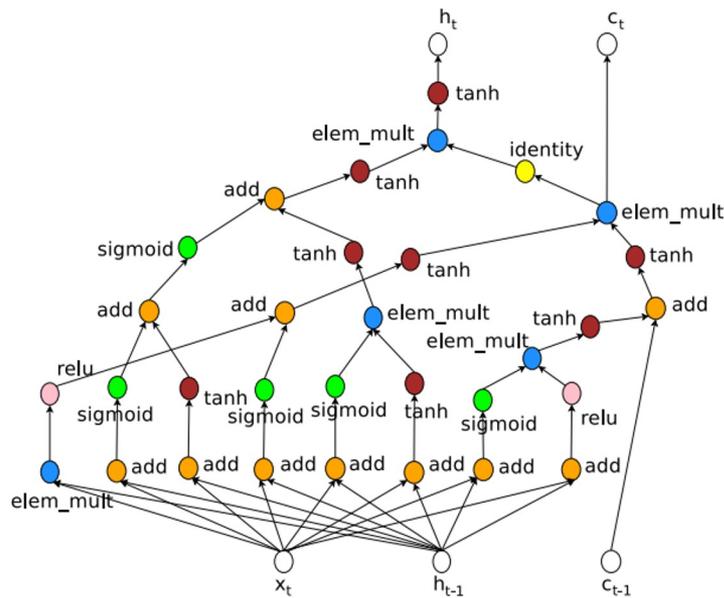
Similar to ResNet!



# Neural Architecture Search for RNN architectures



LSTM cell



Cell they found

# Other RNN Variants

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

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

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

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

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

Simpler than LSTM, but control information flow without cell state.

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

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

MUT1:

$$z = \text{sigm}(W_{xz}x_t + b_z)$$

$$r = \text{sigm}(W_{xr}x_t + W_{hr}h_t + b_r)$$

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

MUT2:

$$z = \text{sigm}(W_{xz}x_t + W_{hz}h_t + b_z)$$

$$r = \text{sigm}(x_t + W_{hr}h_t + b_r)$$

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

MUT3:

$$z = \text{sigm}(W_{xz}x_t + W_{hz} \tanh(h_t) + b_z)$$

$$r = \text{sigm}(W_{xr}x_t + W_{hr}h_t + b_r)$$

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

# Recommendations

- If you want to use RNN-like models, try LSTM
- Use variants like GRU if you want faster compute and less parameters
- New variants of RNNs are still active research topic. Example: RWKV (“Transformer-level performance but with RNN”)

# Problem with Recurrent-style Models (RNN, LSTM, GRU, etc.)

**Learning to memorize** is still hard, especially for ultra-long sequences!

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

Essentially trying to tune  $W$  such that the memory cell  $c$  can retain **important information** for **arbitrary future prediction problems**.

Example (Q&A):

*[... (20-page long transcript)]. Q: What did the CEO say about their competitor company? ...*

*[... (same 20-page transcript)]. Q: How many times did the journalist use the word “interesting”? ...*

Very difficult learning problem!

Next time: Transformer Architecture  
(What memory? Just show me the sequence again)