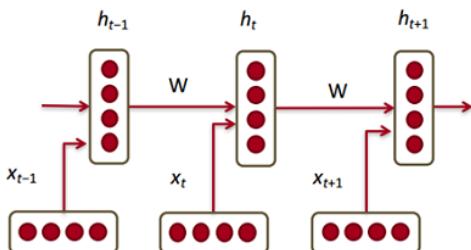
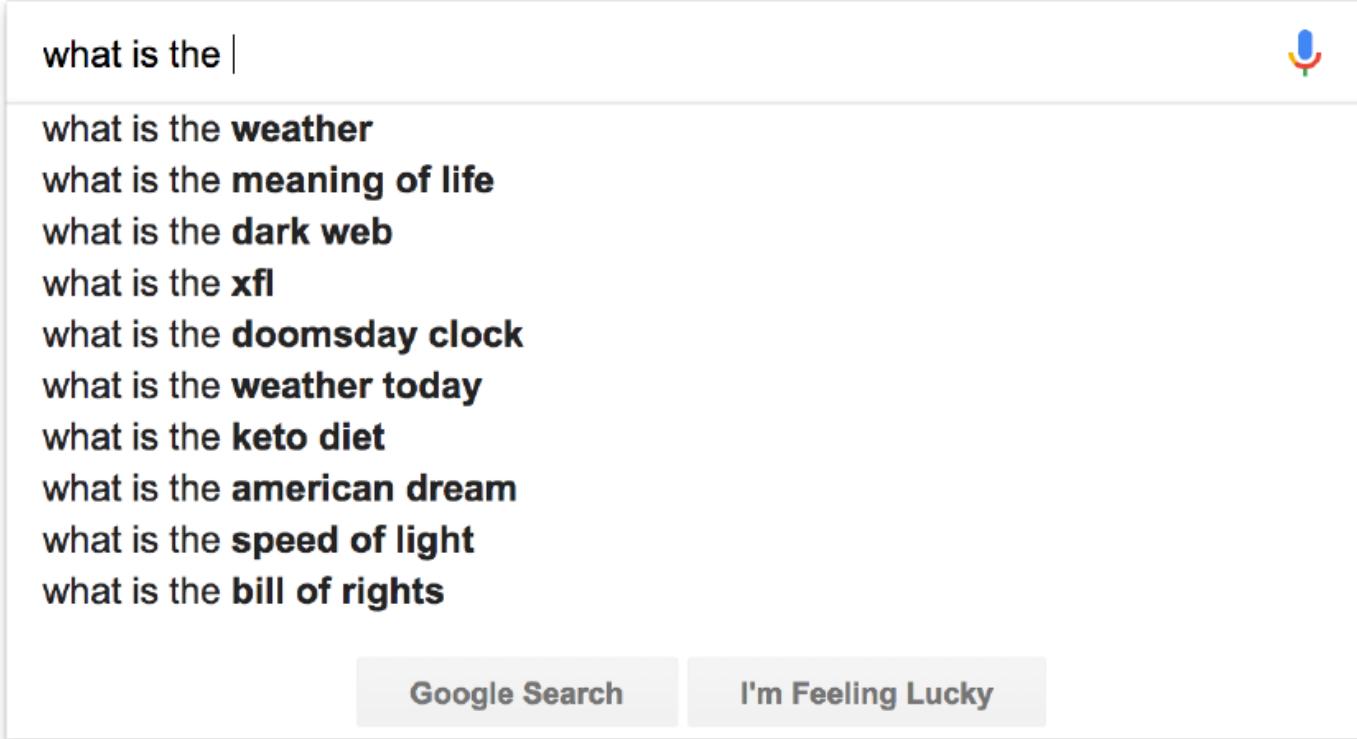


Recurrent Neural Networks



$$x_{shirt} - x_{clothing} \approx x_{chair} - x_{furniture} \quad \log p(o|c) = \log \frac{\exp(u_o^T v_c)}{\sum_{w=1}^W \exp(u_w^T v_c)}$$
$$x_{king} - x_{man} \approx x_{queen} - x_{woman}$$

Language Model



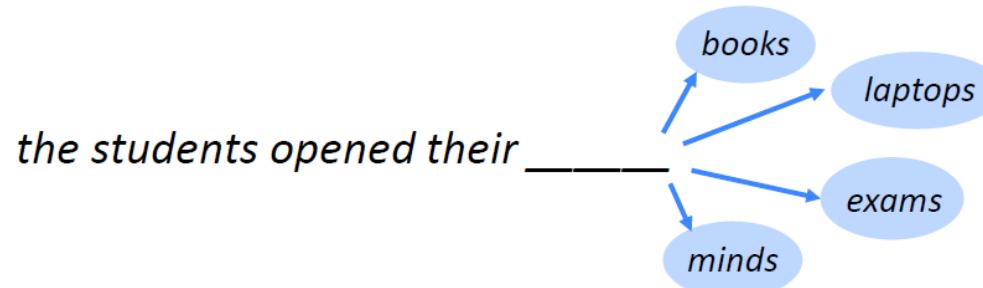
A screenshot of a Google search interface. The search bar at the top contains the text "what is the |". To the right of the search bar is a microphone icon. Below the search bar, a list of suggested search queries is displayed, each preceded by a small blue square icon. The suggestions are:

- what is the **weather**
- what is the **meaning of life**
- what is the **dark web**
- what is the **xfl**
- what is the **doomsday clock**
- what is the **weather today**
- what is the **keto diet**
- what is the **american dream**
- what is the **speed of light**
- what is the **bill of rights**

At the bottom of the interface are two buttons: "Google Search" and "I'm Feeling Lucky".

Language Modeling

- Language Modeling is the task of predicting what word comes next.



- More formally: given a sequence of words $x^{(1)}, x^{(2)}, \dots, x^{(t)}$, compute the probability distribution of the next word $x^{(t+1)}$:

$$P(\mathbf{x}^{(t+1)} | \mathbf{x}^{(t)}, \dots, \mathbf{x}^{(1)})$$

where $x^{(t+1)}$ can be any word in the vocabulary $V = \{w_1, \dots, w_{|V|}\}$

- A system that does this is called a Language Model.

Language Modeling

- You can also think of a Language Model as a system that **assigns probability to a piece of text.**
- For example, if we have some text $x^{(1)}, x^{(2)}, \dots, x^{(T)}$, then the probability of this text(according to the Language Model) is :

$$P(\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(T)}) = P(\mathbf{x}^{(1)}) \times P(\mathbf{x}^{(2)} | \mathbf{x}^{(1)}) \times \dots \times P(\mathbf{x}^{(T)} | \mathbf{x}^{(T-1)}, \dots, \mathbf{x}^{(1)})$$

$$= \prod_{t=1}^T P(\mathbf{x}^{(t)} | \mathbf{x}^{(t-1)}, \dots, \mathbf{x}^{(1)})$$



This is what our LM provides

N-gram Language Models

The students opened their _____

- Definition: A n-gram is a chunk of n consecutive words.
 - Unigrams: “the”, “students”, “opened”, “their”
 - Bigrams: “the students”, “students opened”, “opened their”
 - Trigrams: “the students opened”, “students opened their”
 - 4-grams: “the students opened their”
- Idea: Collect statistics about how frequent different n-grams are, and use these to predict next word.

N-gram Language Models

- First we make a simplifying assumption: $x^{(t+1)}$ depends only on the preceding $n-1$ words.

$$P(\mathbf{x}^{(t+1)} | \mathbf{x}^{(t)}, \dots, \mathbf{x}^{(1)}) = P(\mathbf{x}^{(t+1)} | \overbrace{\mathbf{x}^{(t)}, \dots, \mathbf{x}^{(t-n+2)}}^{n-1 \text{ words}}) \quad (\text{assumption})$$

$$\begin{aligned} & \text{prob of a n-gram} \rightarrow P(\mathbf{x}^{(t+1)}, \mathbf{x}^{(t)}, \dots, \mathbf{x}^{(t-n+2)}) \\ & \text{prob of a (n-1)-gram} \rightarrow P(\mathbf{x}^{(t)}, \dots, \mathbf{x}^{(t-n+2)}) \end{aligned} \quad \mid \quad (\text{definition of conditional prob})$$

- Question:** How do we get these n -gram and $(n-1)$ -gram probabilities?
- Answer:** By **counting** them in some large corpus of text!

$$\approx \frac{\text{count}(\mathbf{x}^{(t+1)}, \mathbf{x}^{(t)}, \dots, \mathbf{x}^{(t-n+2)})}{\text{count}(\mathbf{x}^{(t)}, \dots, \mathbf{x}^{(t-n+2)})} \quad (\text{statistical approximation})$$

N-gram Language Models

- Suppose we are learning 4-gram Language Model.

~~as the proctor started the clock, the students opened their~~
discard students opened their
condition on this

$$P(\mathbf{w}|\text{students opened their}) = \frac{\text{count(students opened their } \mathbf{w}\text{)}}{\text{count(students opened their)}}$$

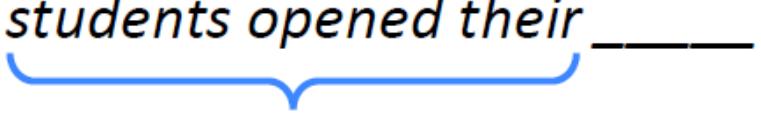
For example, suppose that in the corpus:

- “students opened their” occurred 1000 times
 - “students opened their **books**” occurred **400** times
 - $\rightarrow P(\text{books} \mid \text{students opened their}) = 0.4$
 - “students opened their **exams**” occurred **100** times
 - $\rightarrow P(\text{exams} \mid \text{students opened their}) = 0.1$

Should we have discarded the “proctor” context?

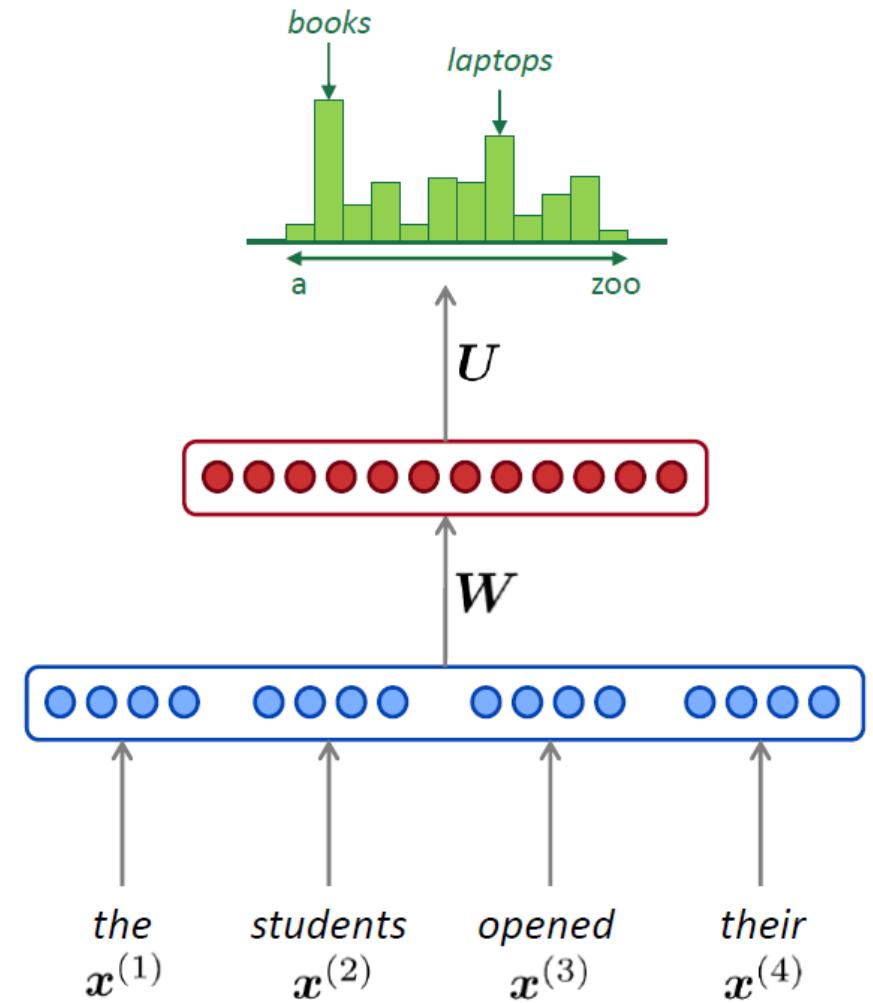
Problems of a N-gram Language Models

- Sparsity Problems
 - What if “students open their (w)” never occurred in data? Then (w) has probability 0!
- Inability to capture long-term dependencies.

~~as the proctor started the clock, the students opened their~~ _____
discard 
condition on this
- Need to store count for all n-grams you saw in the corpus. Increasing n or increasing corpus increases model size.

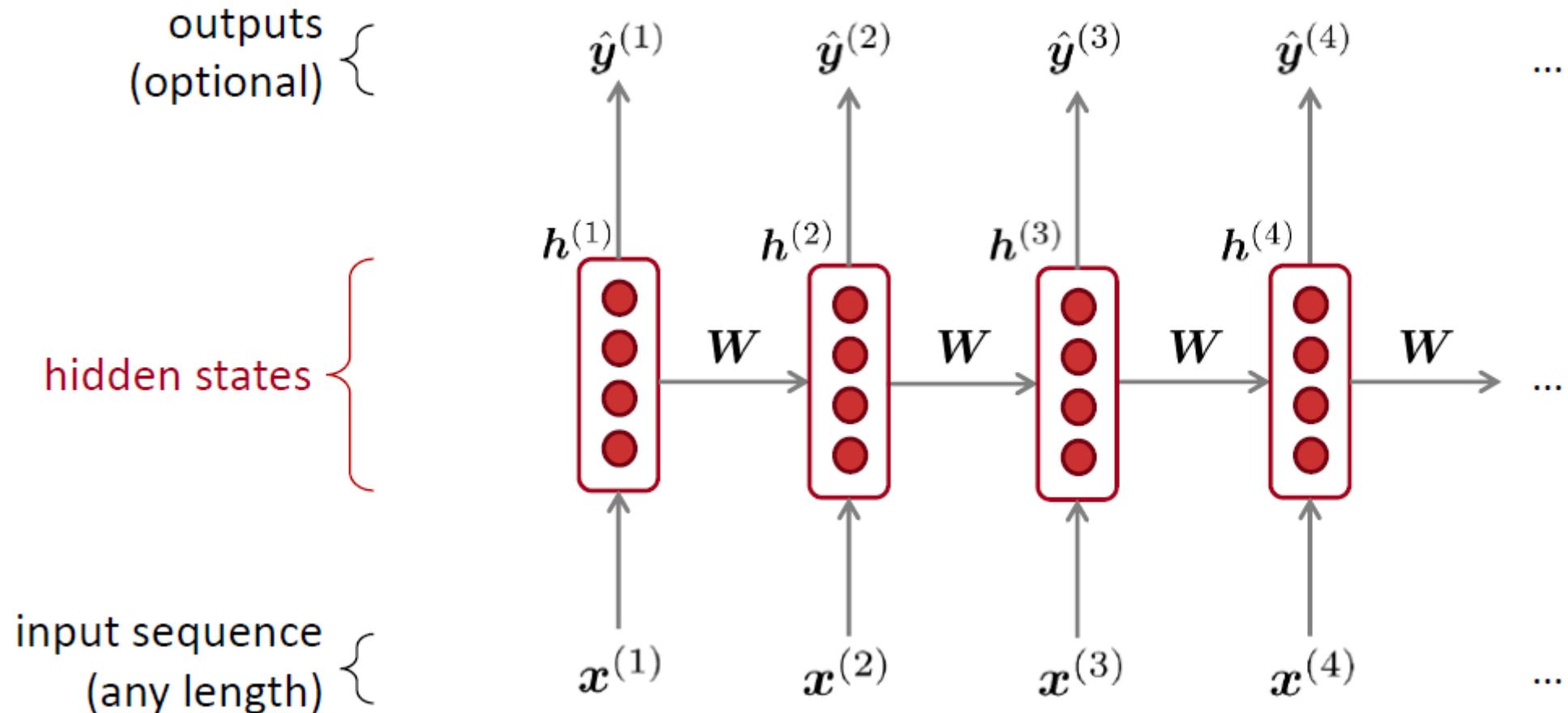
A Fixed-Window Neural Language Model

- Improvement over n-gram LM:
 - No sparsity problem
 - Don't need to store all observed n-grams
- Problems
 - Fixed window is too small
 - Long term dependencies cannot be captured
 - Enlarging window enlarges W



Recurrent Neural Network

- Core idea: Apply the same weights W repeatedly



$$\hat{\mathbf{y}}^{(4)} = P(\mathbf{x}^{(5)} | \text{the students opened their})$$

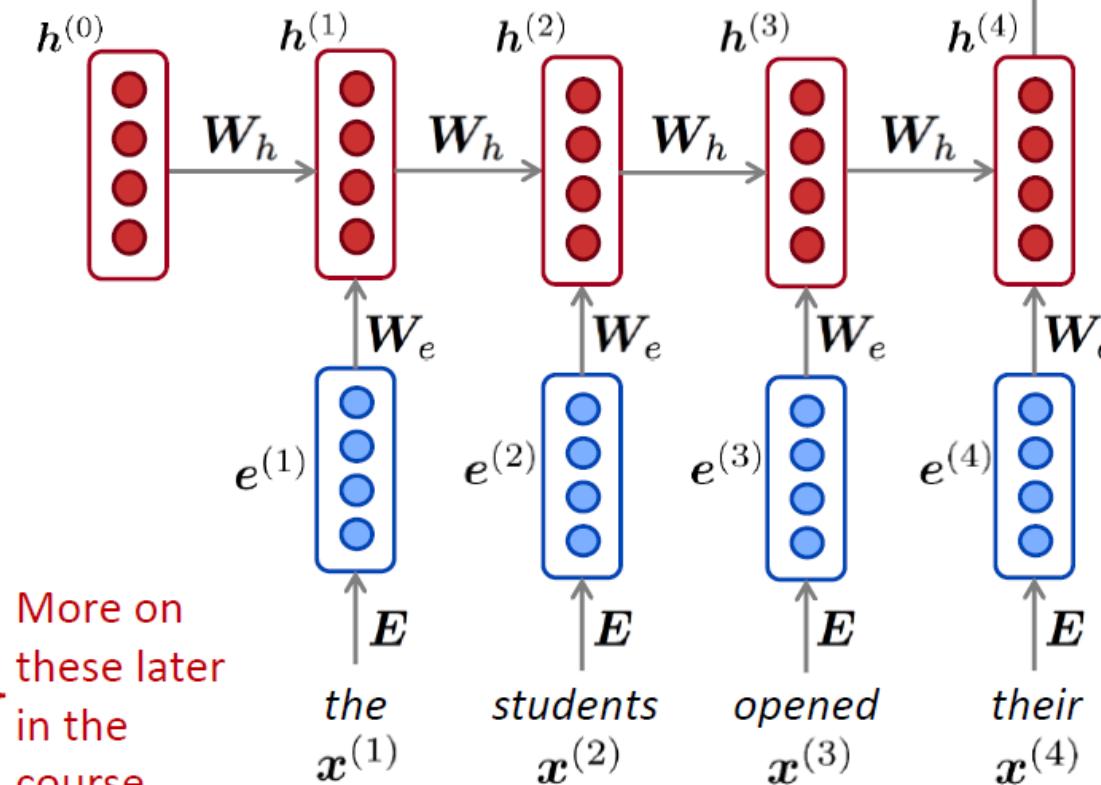
A RNN Language Model

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.

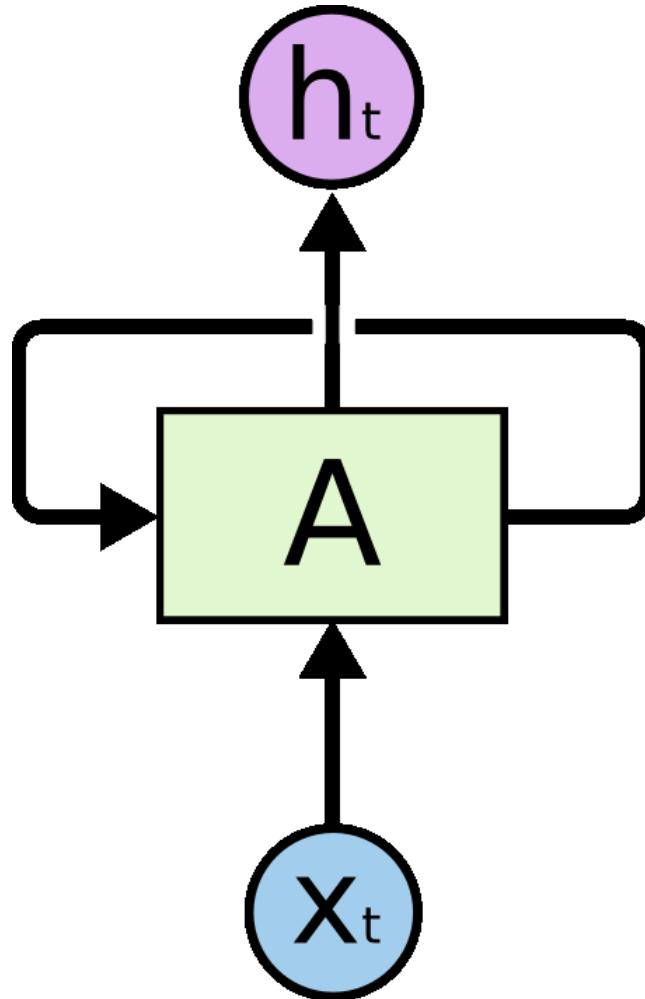
RNN Disadvantages:

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

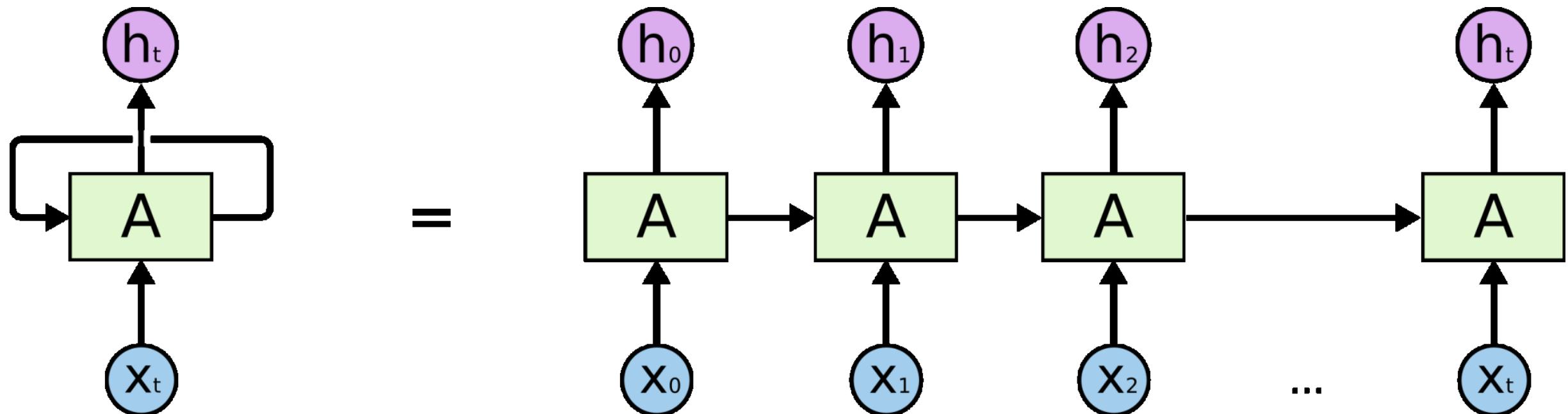


More on
these later
in the
course

Recurrent Neural Network

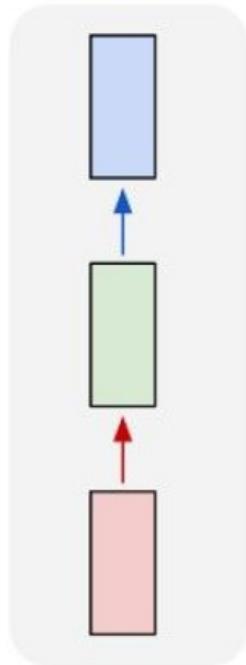


Recurrent Neural Network

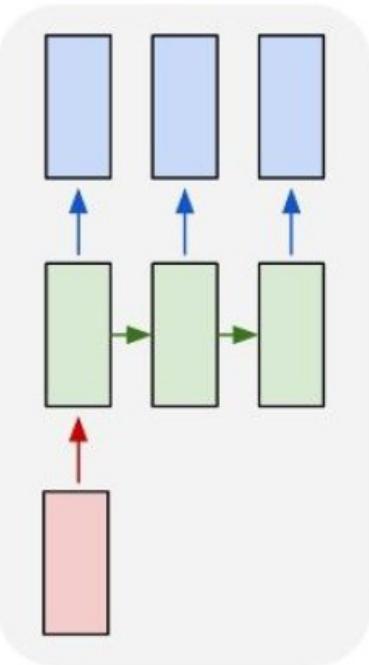


RNN's flexibility

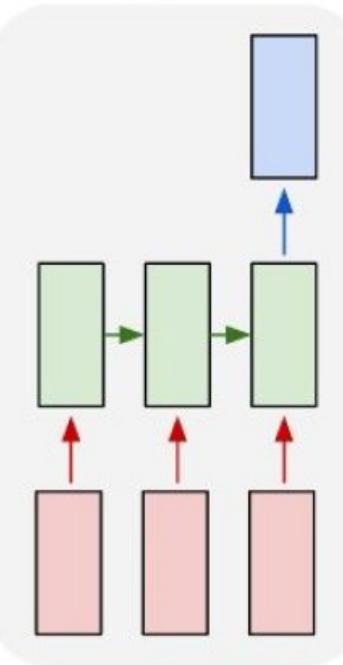
one to one



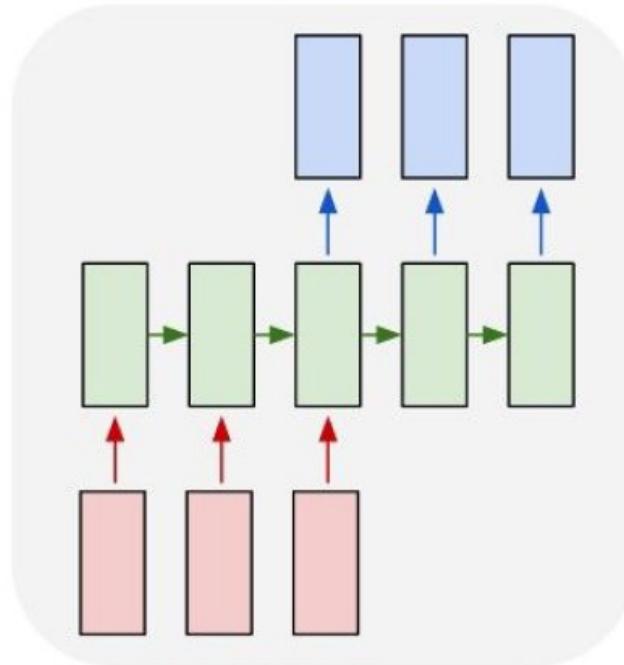
one to many



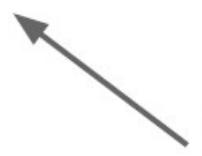
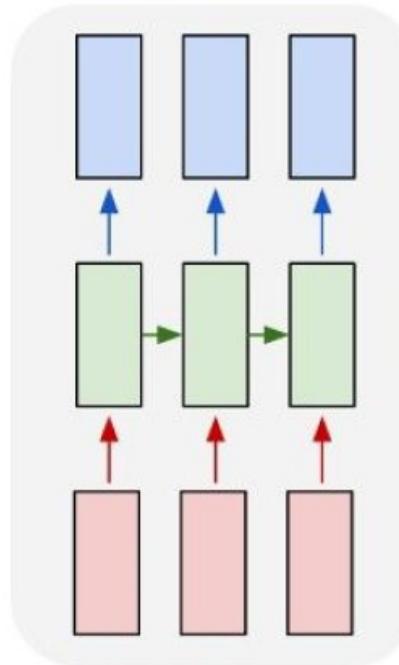
many to one



many to many

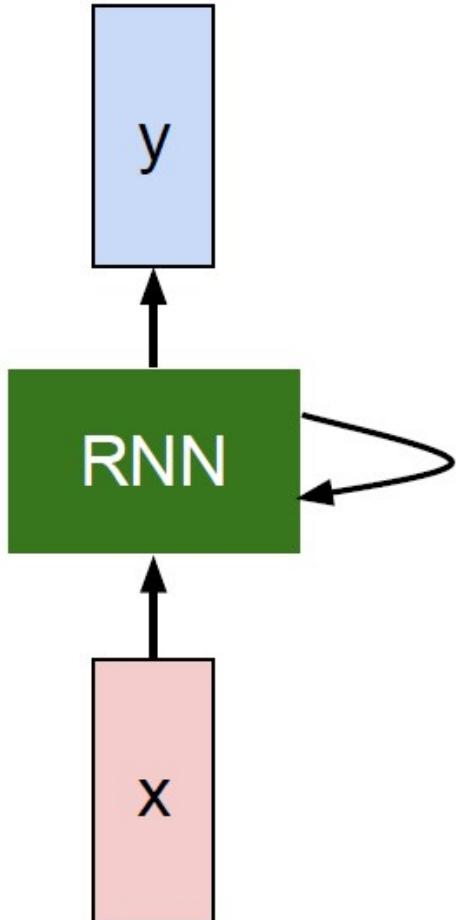


many to many



Vanilla Neural Networks

Recurrent Neural Network



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



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

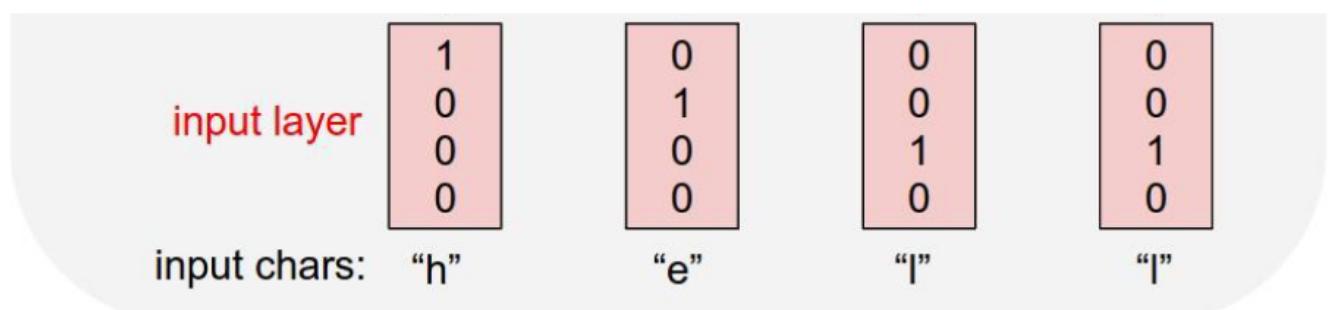
$$y_t = W_{hy}h_t$$

RNN

**Character-level
language model
example**

Vocabulary:
[h,e,l,o]

Example training
sequence:
“hello”



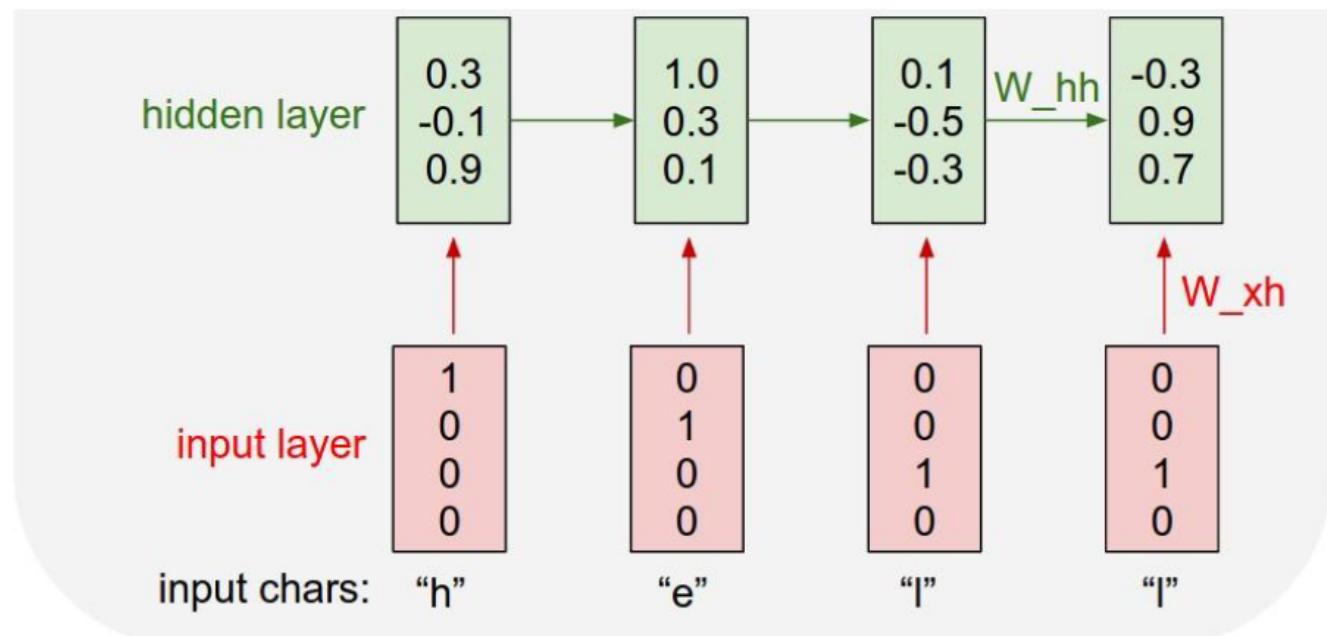
RNN

Character-level language model example

Vocabulary:
[h,e,l,o]

Example training
sequence:
“hello”

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

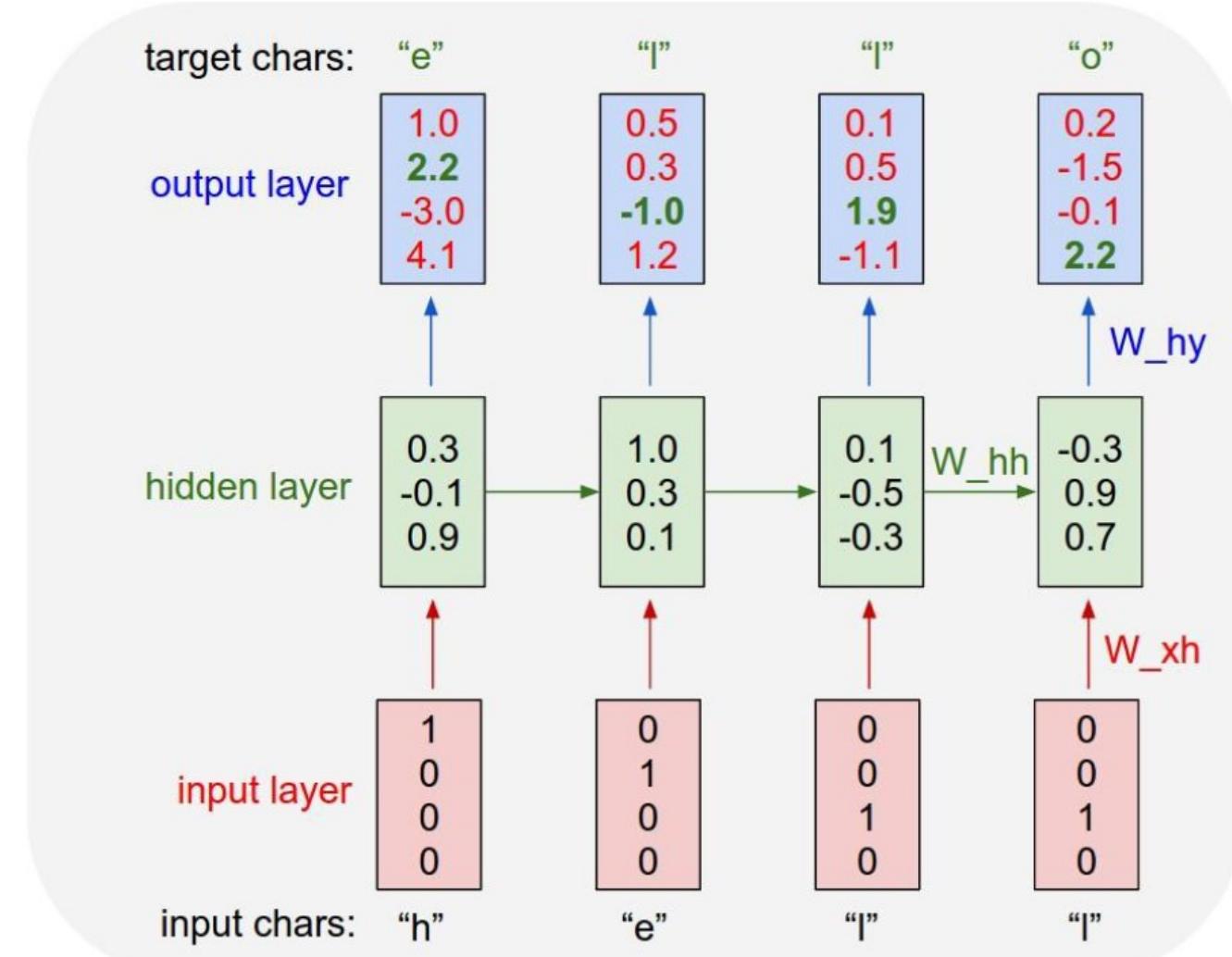


RNN

Character-level language model example

Vocabulary:
[h,e,l,o]

Example training
sequence:
“hello”



Language Model

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.

Language Model

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 $X_{\text{étale}}$ we have

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

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

Lemma 0.2. *This is an integer 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 $\mathcal{U} \subset X$ 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 X .

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

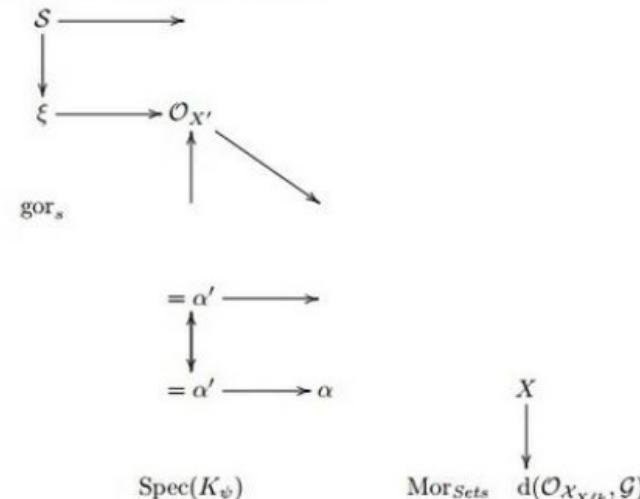
be a morphism of algebraic spaces over S and Y .

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



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

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

is an isomorphism of covering of \mathcal{O}_{X_i} . If \mathcal{F} is the unique element of \mathcal{F} such that X is an isomorphism.

The property \mathcal{F} is a disjoint union of Proposition ?? and we can filtered set of presentations of a scheme \mathcal{O}_Y -algebra with \mathcal{F} are opens of finite type over S .

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.

Language Model

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

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

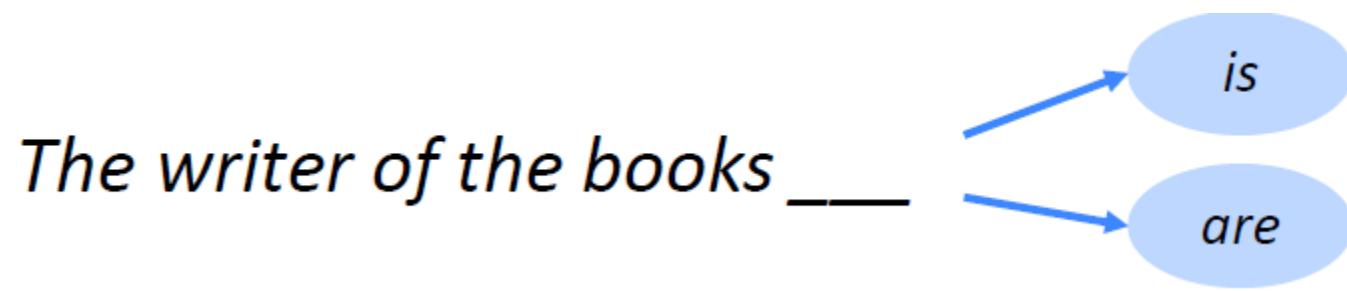
#define SWAP_ALLOCATE(nr)      (e)
#define emulate_sigs() arch_get_unaligned_child()
#define access_rw(TST) asm volatile("movd %%esp, %0, %3" : : "r" (0)); \
    if (__type & DO_READ)

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;
}

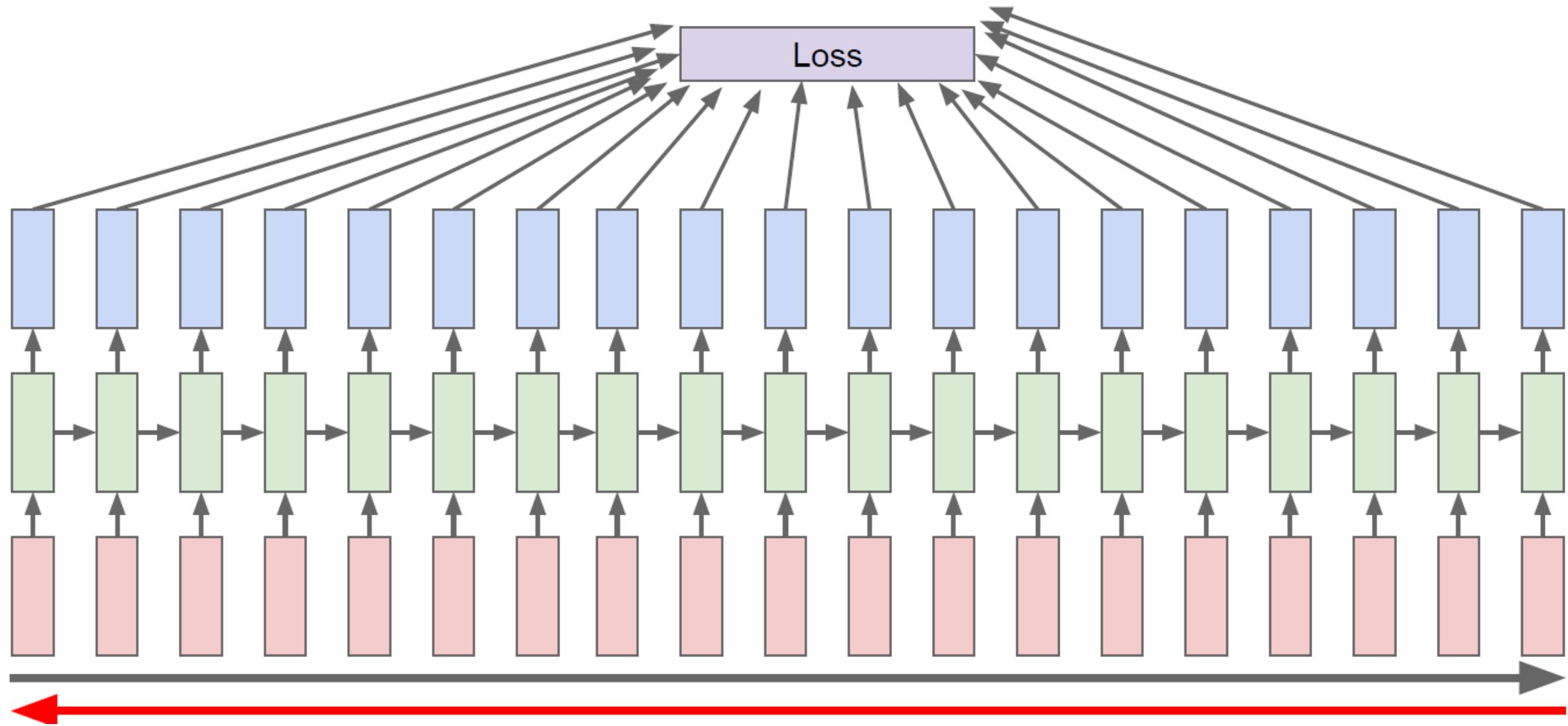
```

Is Long Term Dependency Problem Resolved?

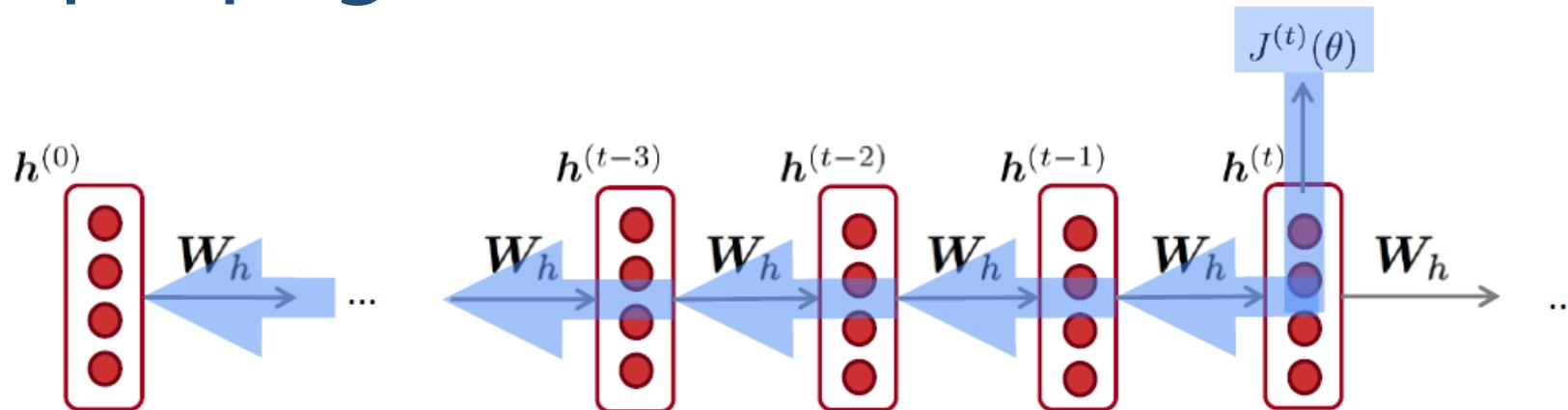


- RNN language model can choose “are”.
- Why? How can we train RNN?

Back Propagation Through Time



Backpropagation of RNNs

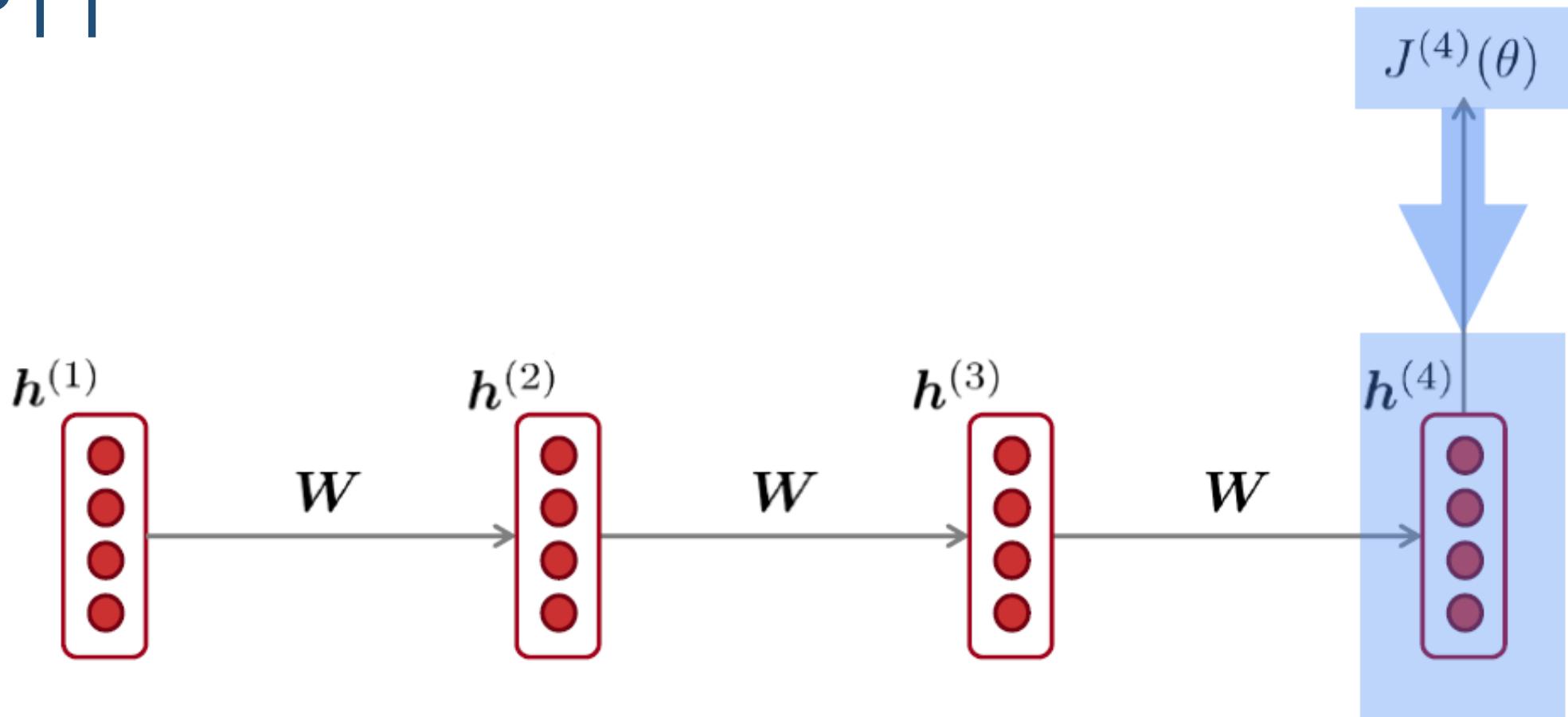


$$\frac{\partial J^{(t)}}{\partial \mathbf{W}_h} = \sum_{i=1}^t \frac{\partial J^{(t)}}{\partial \mathbf{W}_h} \Big|_{(i)}$$

Question: How do we calculate this?

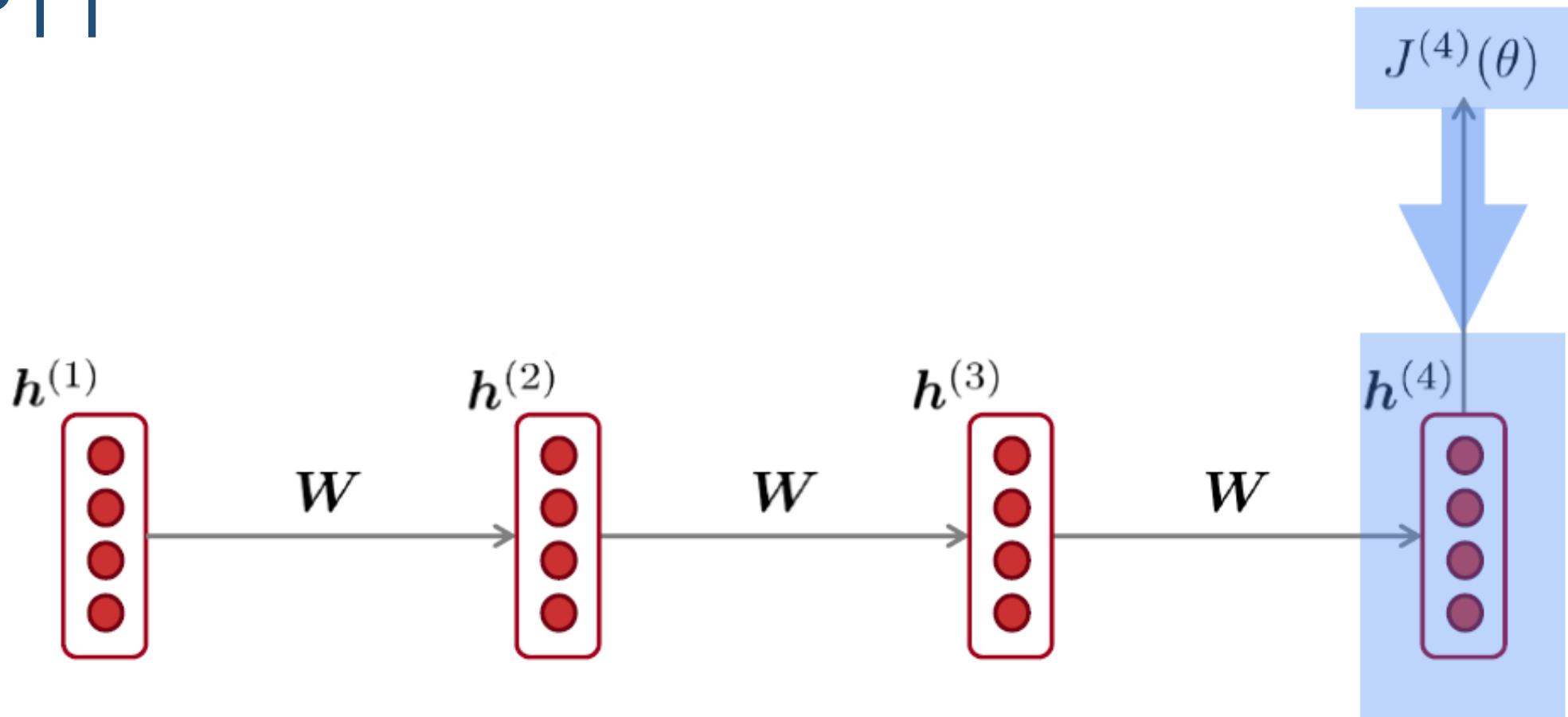
Answer: Backpropagate over timesteps $i=t, \dots, 0$, summing gradients as you go.
This algorithm is called “backpropagation through time”

BPTT



$$\frac{\partial J^{(4)}}{\partial h^{(1)}} = ?$$

BPTT

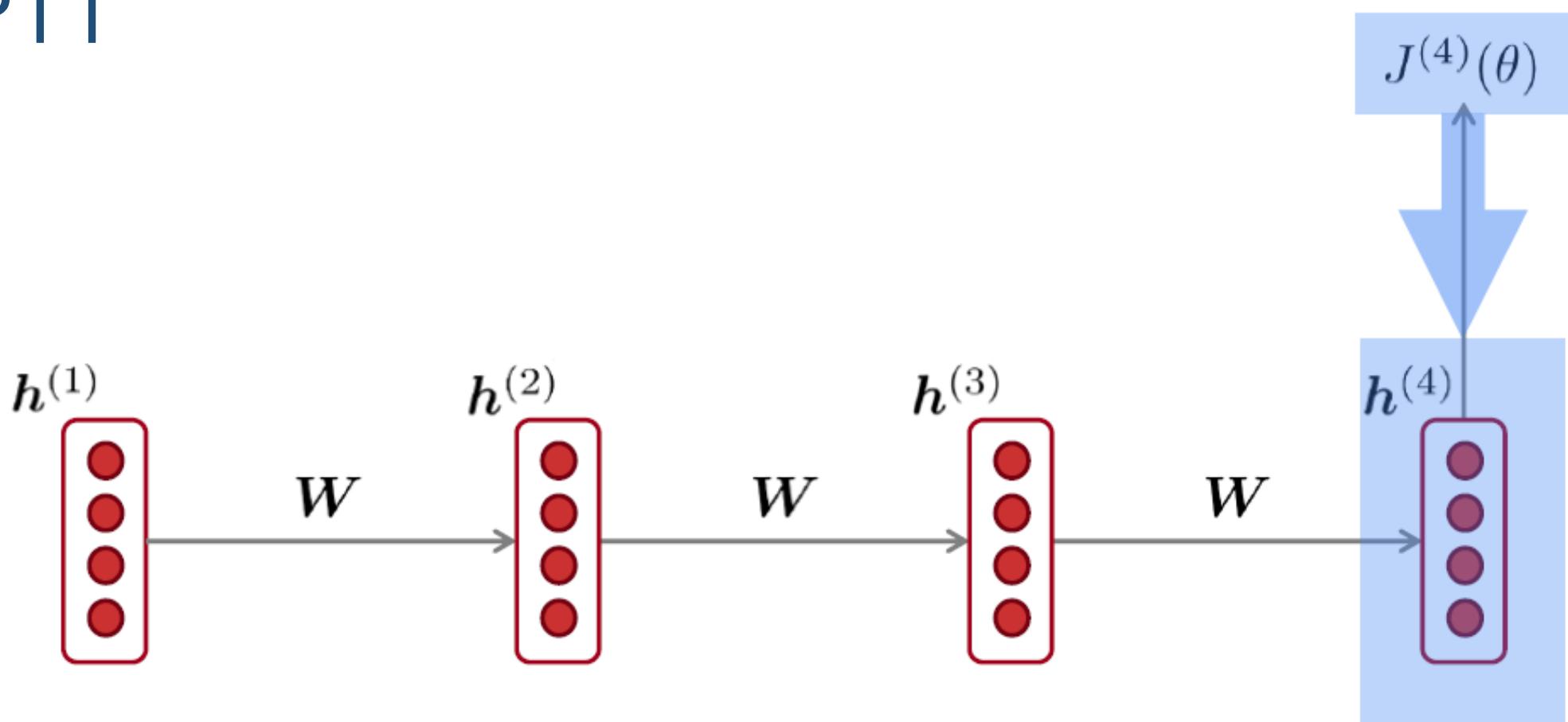


$$\frac{\partial J^{(4)}}{\partial \mathbf{h}^{(1)}} =$$

$$\frac{\partial \mathbf{h}^{(4)}}{\partial \mathbf{h}^{(3)}} \times \frac{\partial J^{(4)}}{\partial \mathbf{h}^{(4)}}$$

chain rule!

BPTT



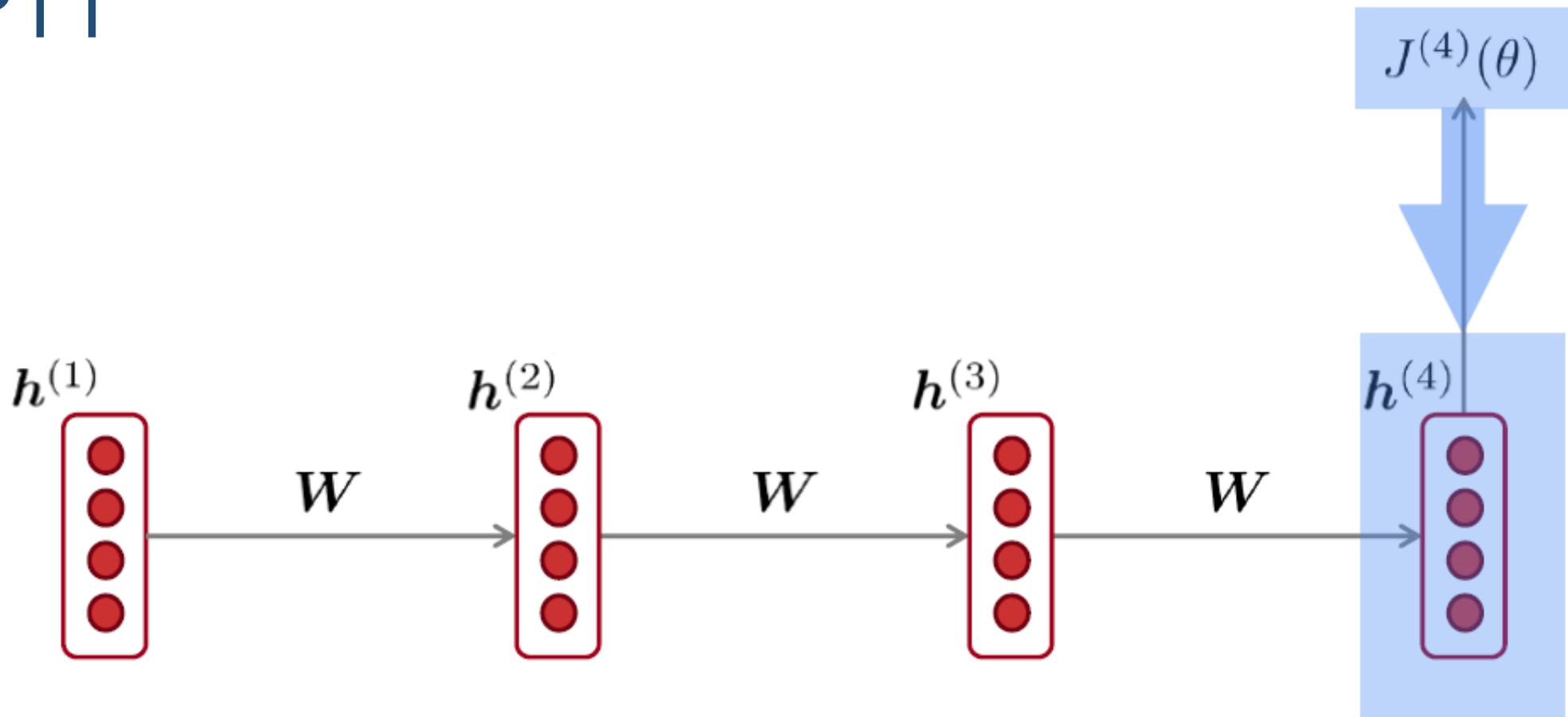
$$\frac{\partial J^{(4)}}{\partial \mathbf{h}^{(1)}} =$$

$$\frac{\partial \mathbf{h}^{(3)}}{\partial \mathbf{h}^{(2)}} \times$$

$$\frac{\partial \mathbf{h}^{(4)}}{\partial \mathbf{h}^{(3)}} \times \frac{\partial J^{(4)}}{\partial \mathbf{h}^{(4)}}$$

chain rule!

BPTT



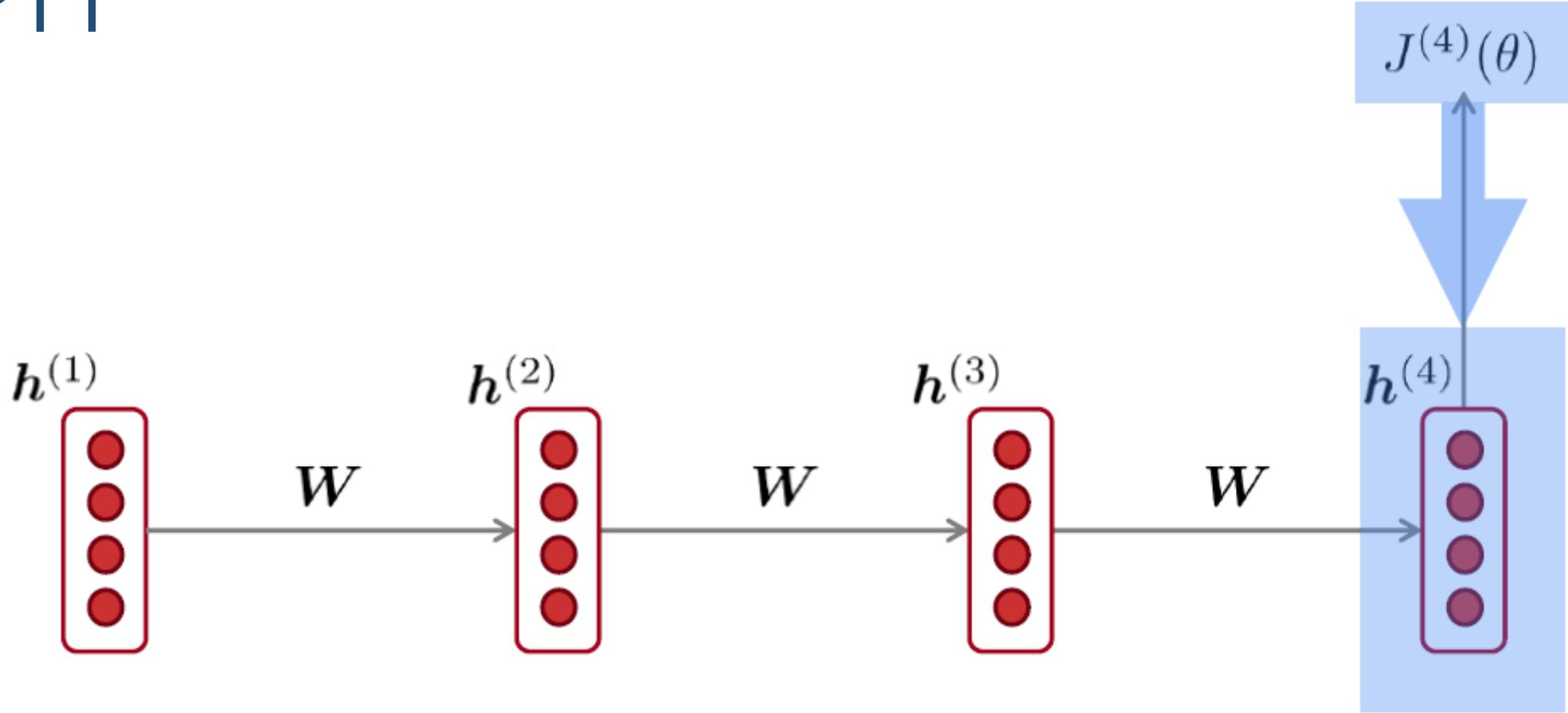
$$\frac{\partial J^{(4)}}{\partial \mathbf{h}^{(1)}} = \frac{\partial \mathbf{h}^{(2)}}{\partial \mathbf{h}^{(1)}} \times \dots$$

$$\frac{\partial \mathbf{h}^{(3)}}{\partial \mathbf{h}^{(2)}} \times$$

$$\frac{\partial \mathbf{h}^{(4)}}{\partial \mathbf{h}^{(3)}} \times \frac{\partial J^{(4)}}{\partial \mathbf{h}^{(4)}}$$

chain rule!

BPTT



$$\frac{\partial J^{(4)}}{\partial h^{(1)}} = \boxed{\frac{\partial h^{(2)}}{\partial h^{(1)}}} \times \boxed{\frac{\partial h^{(3)}}{\partial h^{(2)}}} \times \boxed{\frac{\partial h^{(4)}}{\partial h^{(3)}}} \times \frac{\partial J^{(4)}}{\partial h^{(4)}}$$

What happens if these are smaller than 1 or larger than 1?

Vanishing/Exploding Gradient Problem

- $\frac{\partial J^{(4)}}{\partial h^{(1)}} = \frac{\partial J^{(4)}}{\partial h^{(4)}} \cdot \prod_{i=2}^4 \frac{\partial h^{(t)}}{\partial h^{(t-1)}}$
- Recall: $h^{(t)} = \tanh(W_h h^{(t-1)} + W_x x^{(t)} + b)$
- $\frac{\partial h^{(t)}}{\partial h^{(t-1)}} = W_h \cdot \tanh'(W_h h^{(t-1)} + W_x x^{(t)} + b)$
- Therefore: $\frac{\partial J^{(4)}}{\partial h^{(1)}} = \frac{\partial J^{(4)}}{\partial h^{(4)}} \cdot W_h^3 \cdot \tanh'^3$
- Generally, $\frac{\partial J^{(n)}}{\partial h^{(m)}} = \frac{\partial J^{(n)}}{\partial h^{(n)}} \cdot W_h^{(n-m)} \cdot \tanh'^{(n-m)}$

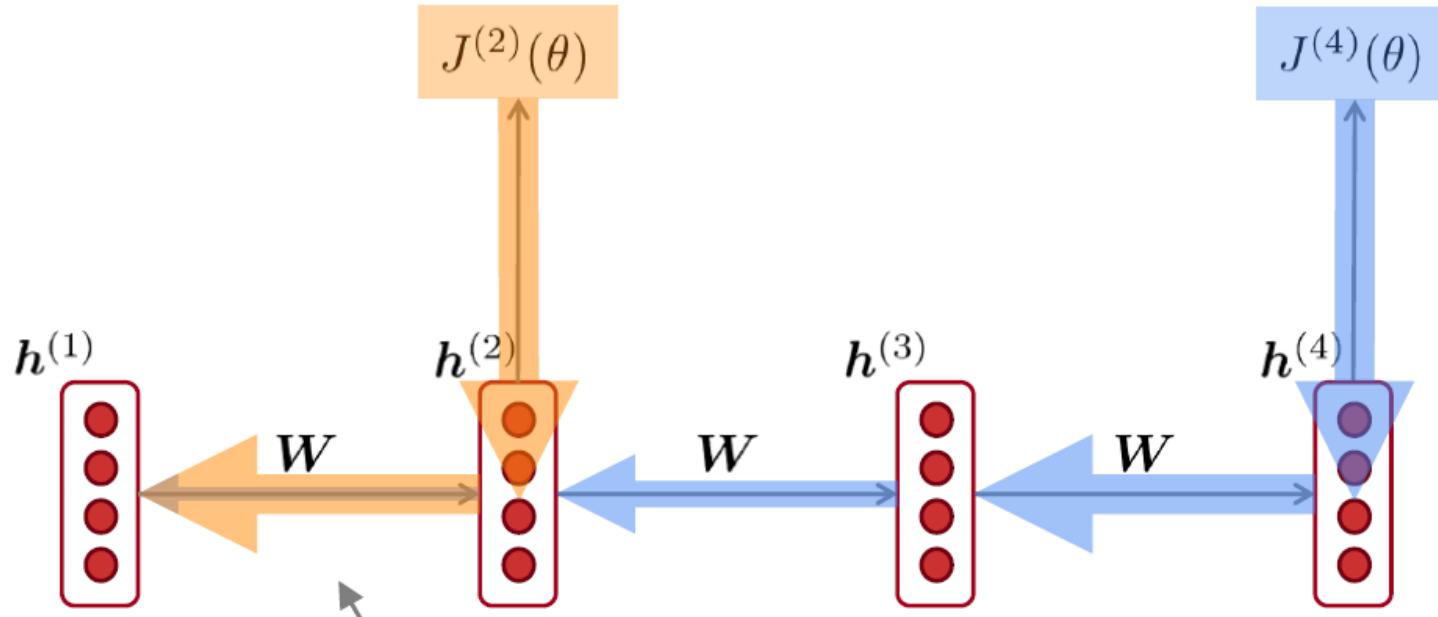
What happens if $(n-m)$ is getting larger?

Vanishing/Exploding Gradient Problem

$$\frac{\partial J^{(n)}}{\partial h^{(m)}} = \frac{\partial J^{(n)}}{\partial h^{(n)}} \cdot W_h^{(n-m)} \cdot \tanh'(n-m)$$

- \tanh' is always less than equal to 1 \rightarrow vanishing gradient
- If the largest eigenvalue of $W_h < 1$ \rightarrow vanishing gradient
- If the largest eigenvalue of $W_h > 1$ \rightarrow exploding gradient

Why is Vanishing Gradient a Problem?



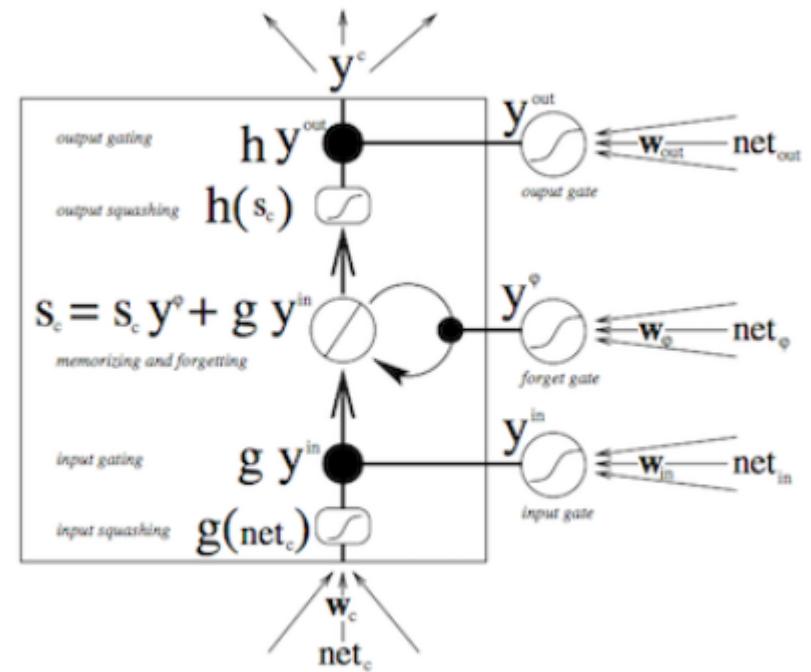
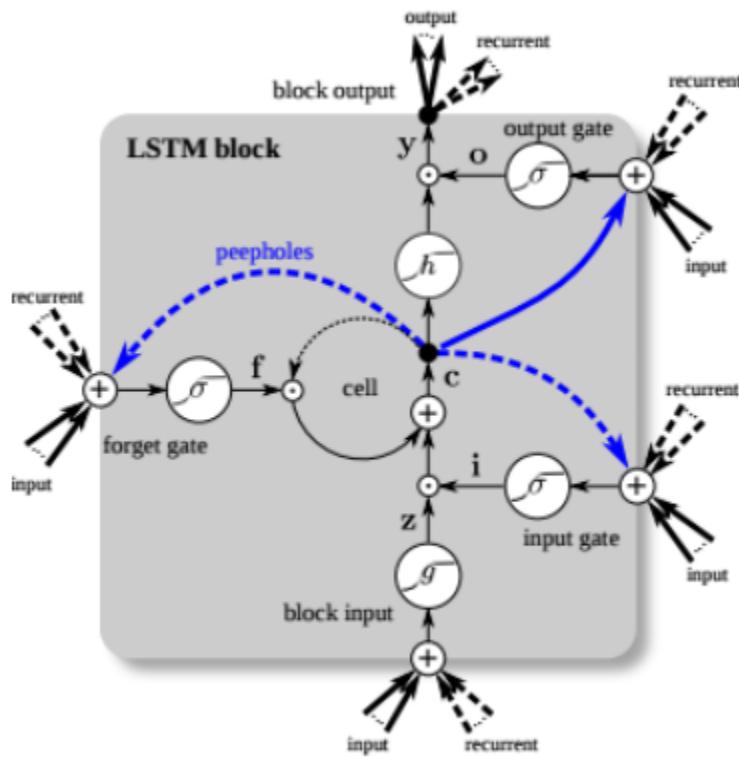
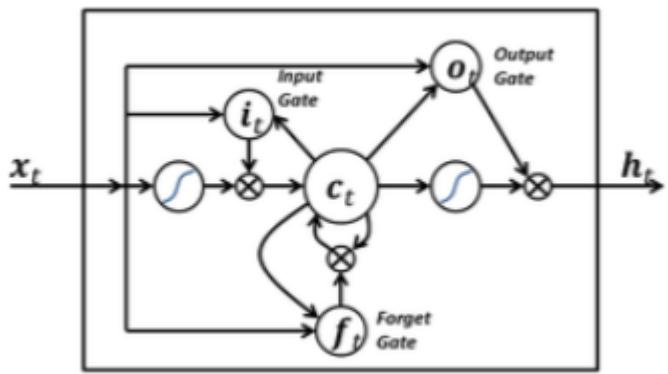
Gradient signal from faraway is lost because it's much smaller than gradient signal from close-by.

So model weights are only updated only with respect to near effects, not long-term effects.

Why is Vanishing Gradient a Problem?

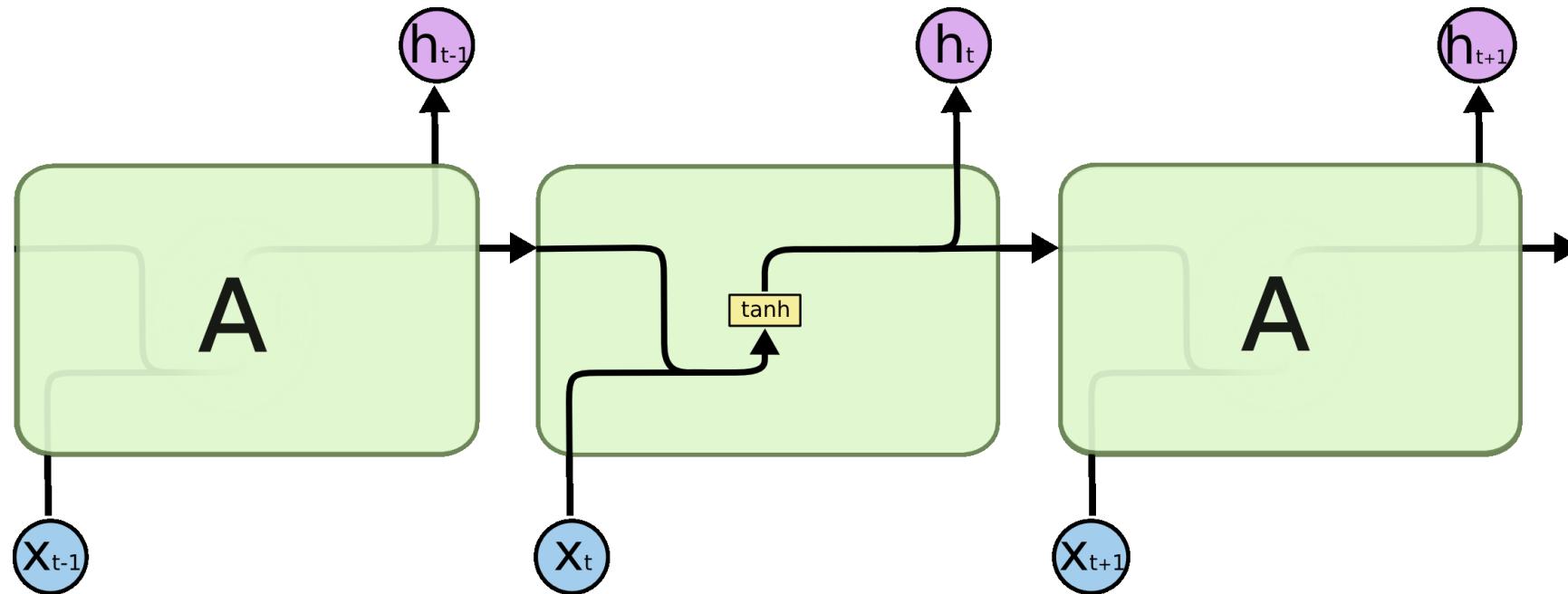
- Another explanation: **Gradient** can be viewed as a measure of **the effect of the past on the future**
- If the gradient becomes vanishingly small over longer distances(step t to step $t+n$). Then we can't tell whether:
 - There is **no dependency** between step t and $t+n$ in the data
 - We have **wrong parameters** to capture the true dependency between t and $t+n$

Long Short Term Memory



LSTM

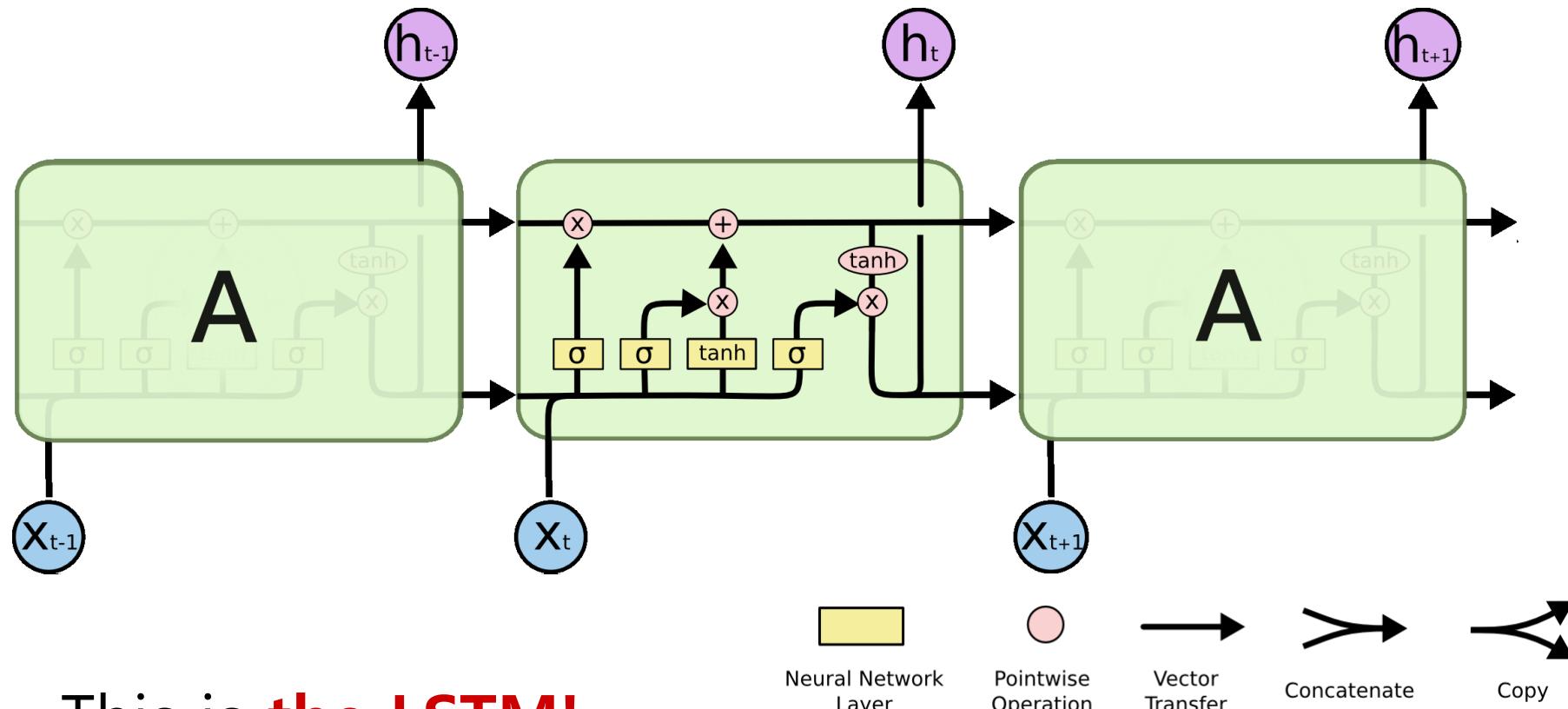
Long Short Term Memory



This is just a standard RNN.

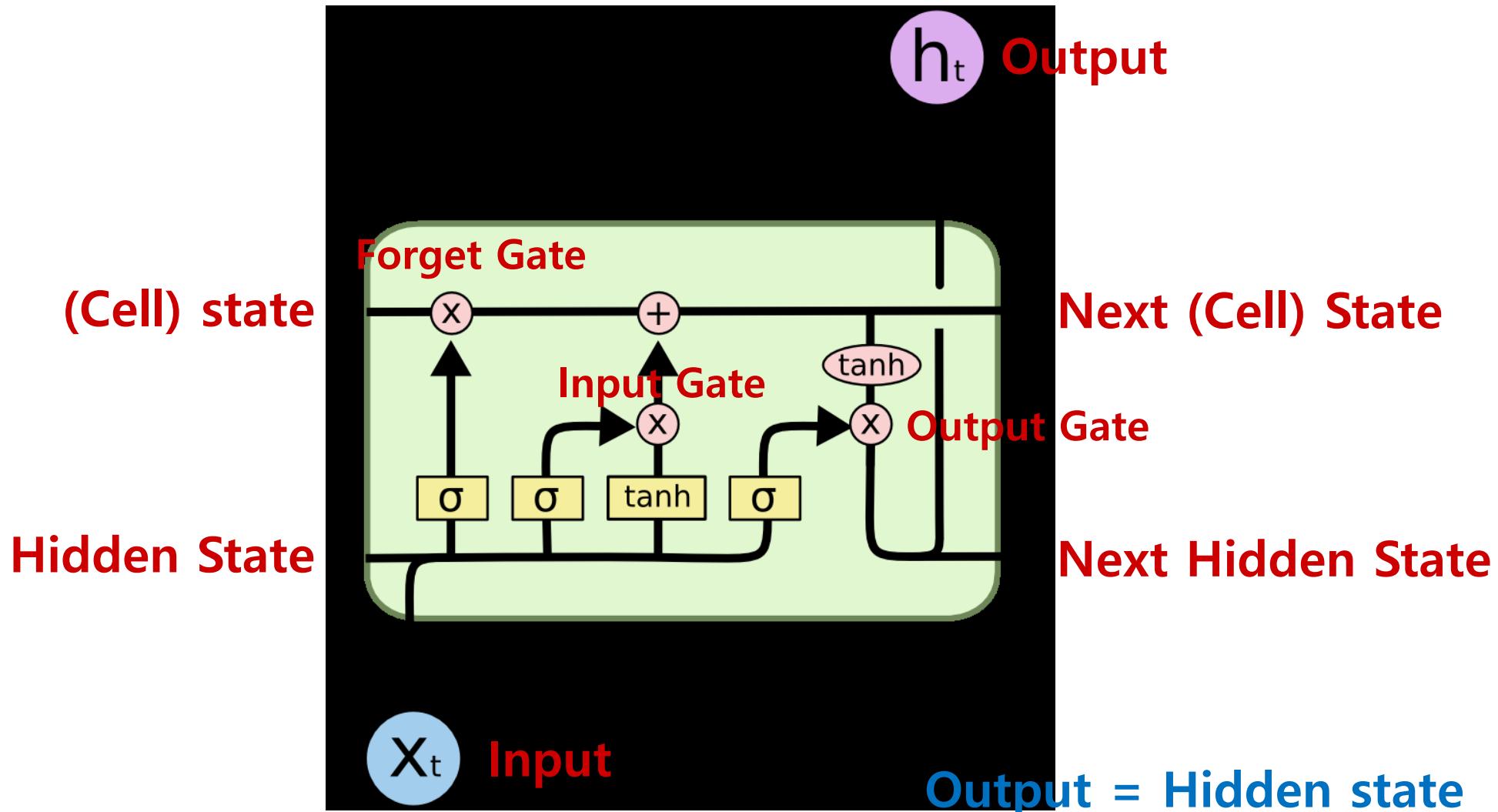
LSTM

Long Short Term Memory

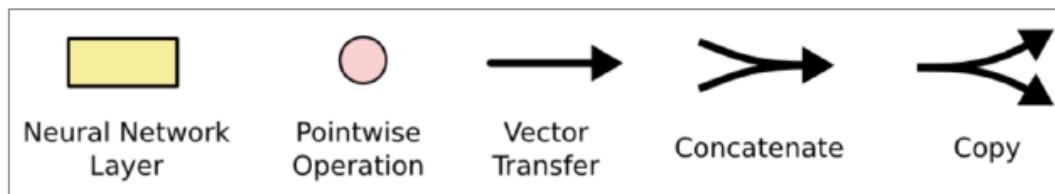
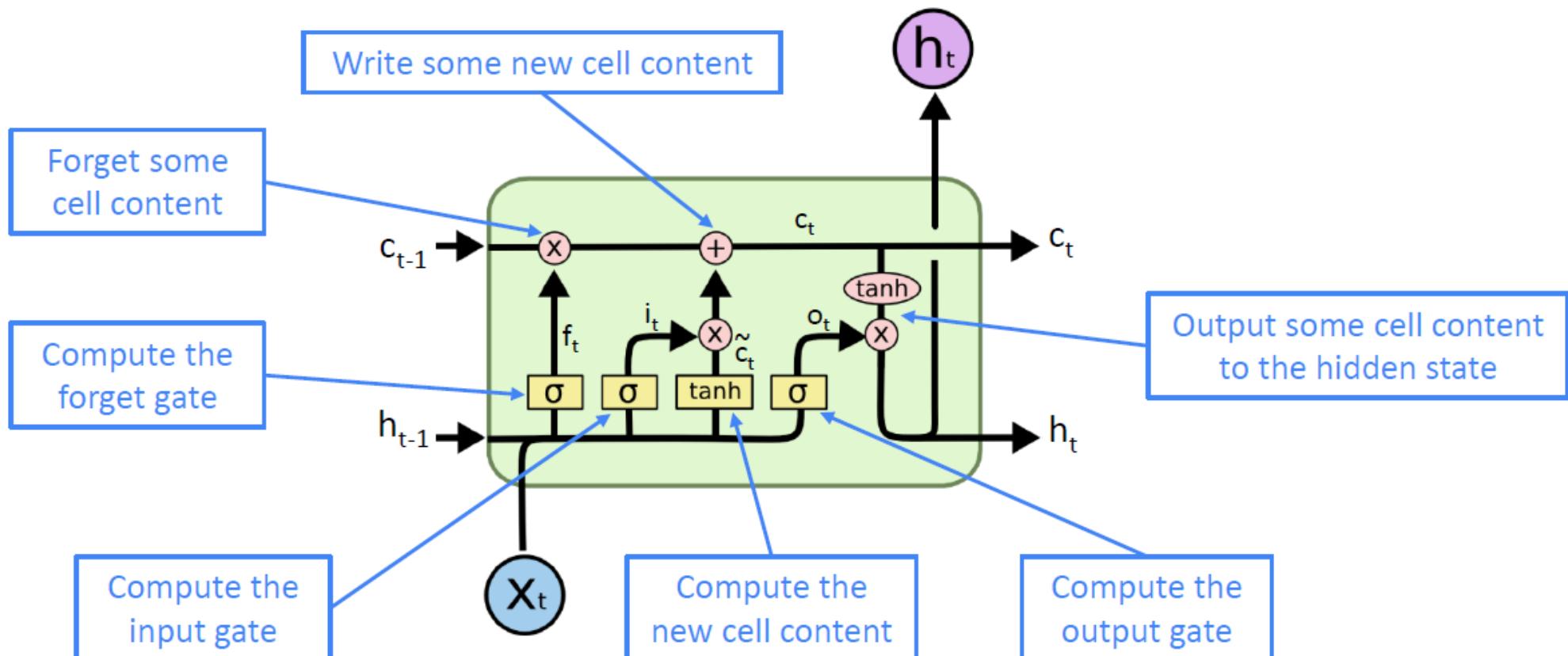


This is **the LSTM!**

Overall Architecture

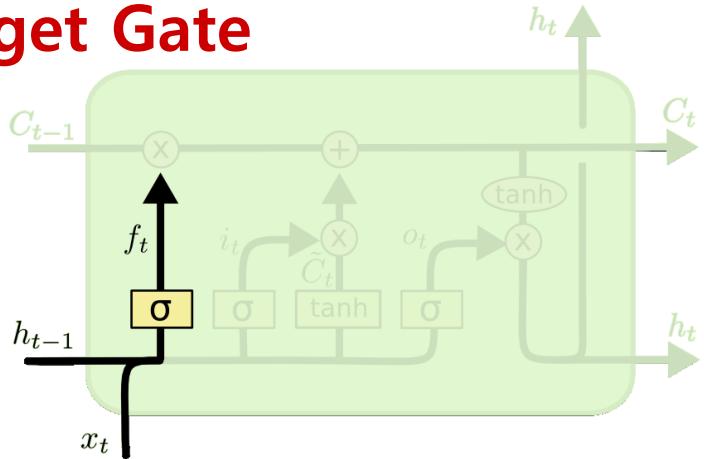


Overall Architecture



Forget Gate & Input Gate

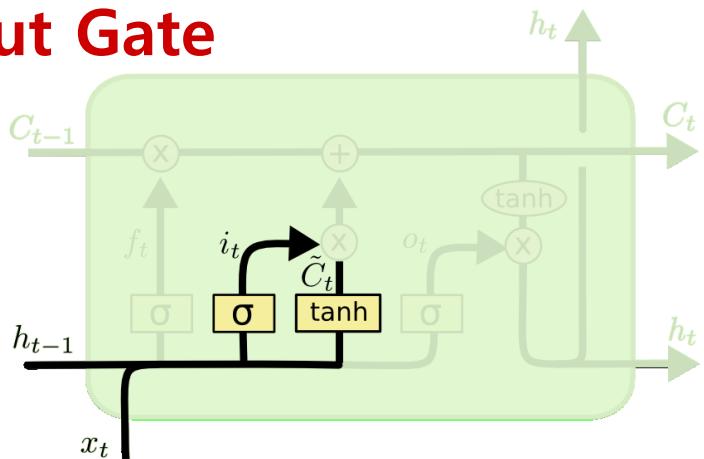
Forget Gate



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

Decide what information we're going to **throw away** from the cell state.

Input Gate



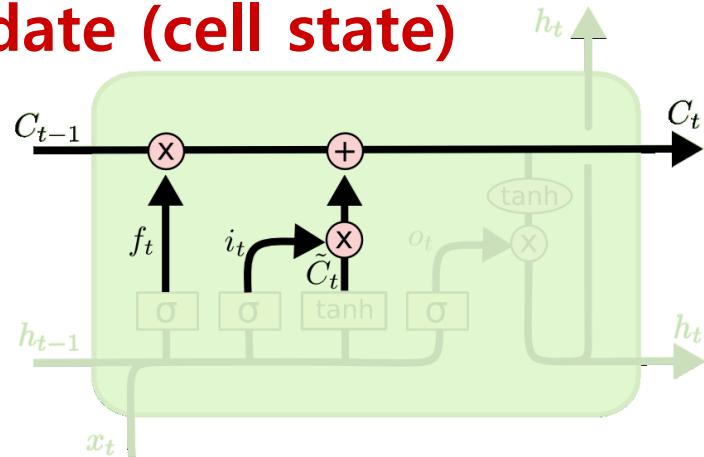
$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

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

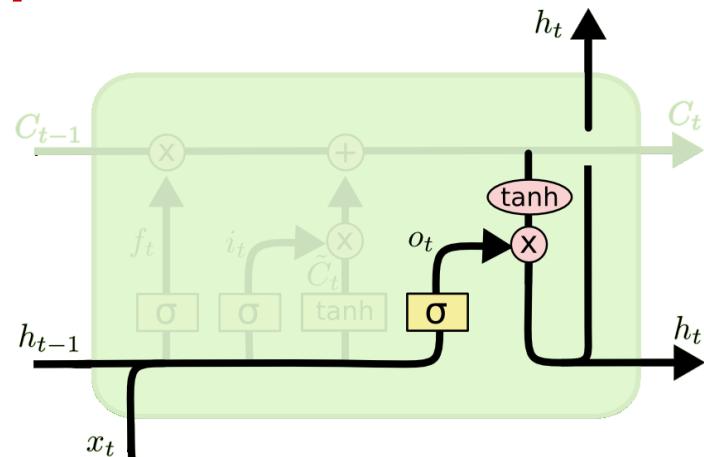
Decide what new information we're going to **store** in the cell state.

Update Cell State & Output Gate

Update (cell state)



Output Gate (hidden state)



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

Update, scaled by how much we decide to update

: `input_gate*curr_state + forget_gate*prev_state`

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

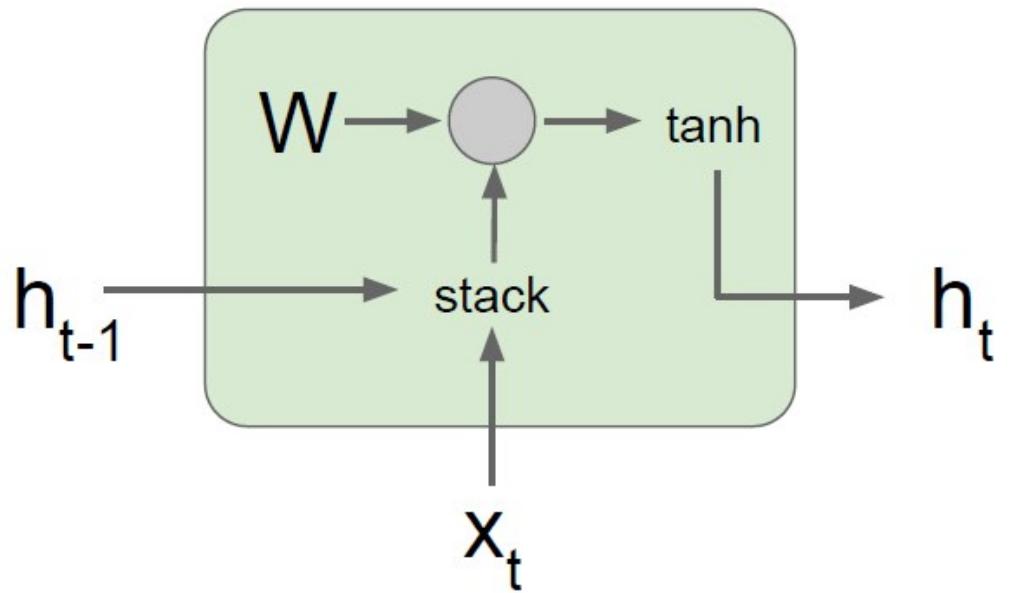
$$h_t = o_t * \tanh(C_t)$$

Output based on the updated state

: `output_gate*updated_state`

Is LSTM free from Vanishing Gradient Problems?

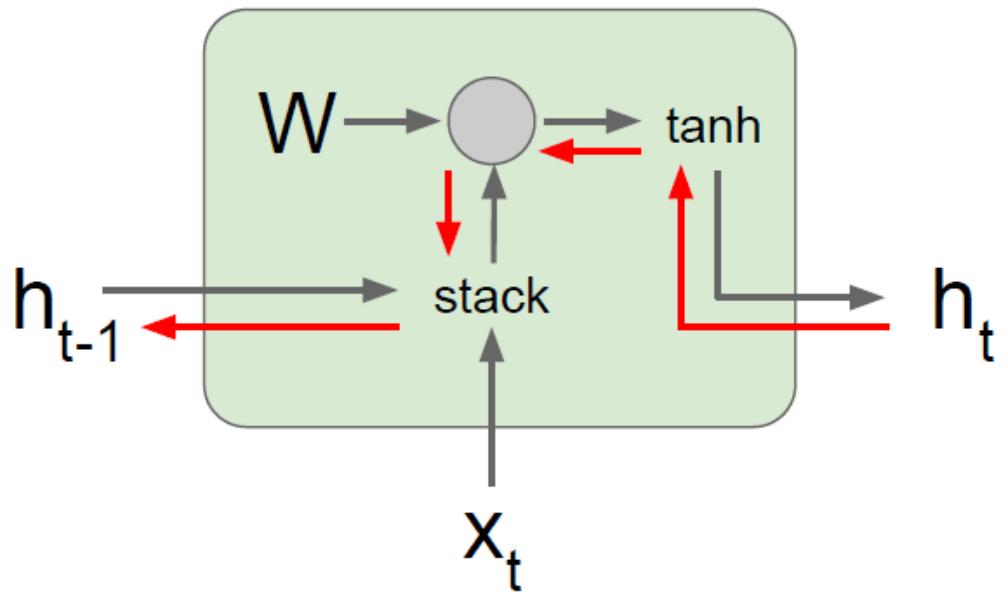
Vanilla RNN Gradient Flow



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

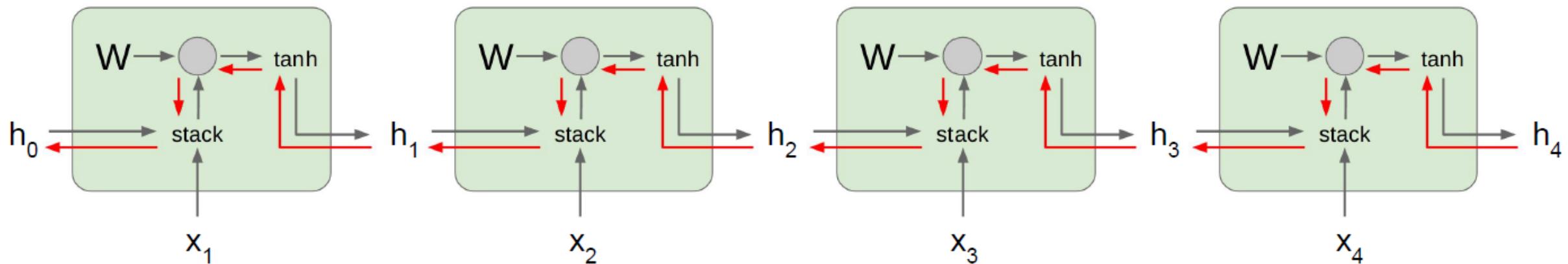
Vanilla RNN Gradient Flow

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



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

Vanilla RNN Gradient Flow



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

Largest singular value > 1 :
Exploding gradients

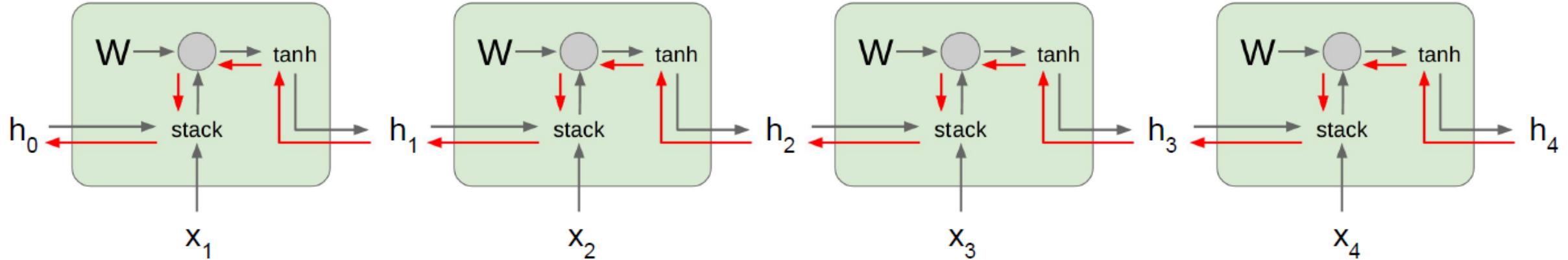
Largest singular value < 1 :
Vanishing gradients

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

- Gradient가 흘러갈 때 같은 숫자(w , \tanh 의 미분)가 계속 곱해짐



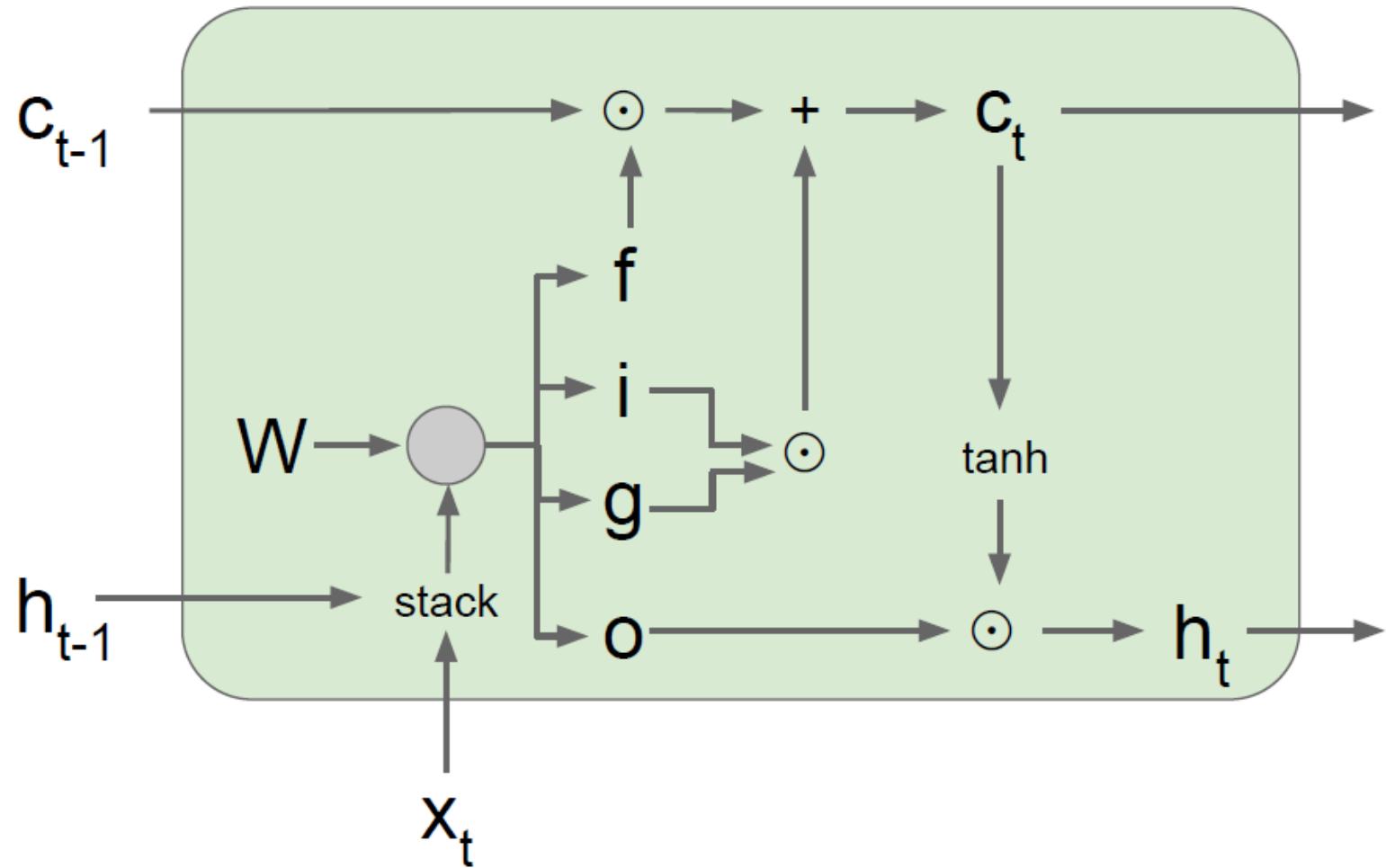
Computing gradient
of h_0 involves many
factors of W
(and repeated \tanh)

Largest singular value > 1 :
Exploding gradients

Largest singular value < 1 :
Vanishing gradients

→ Change RNN architecture

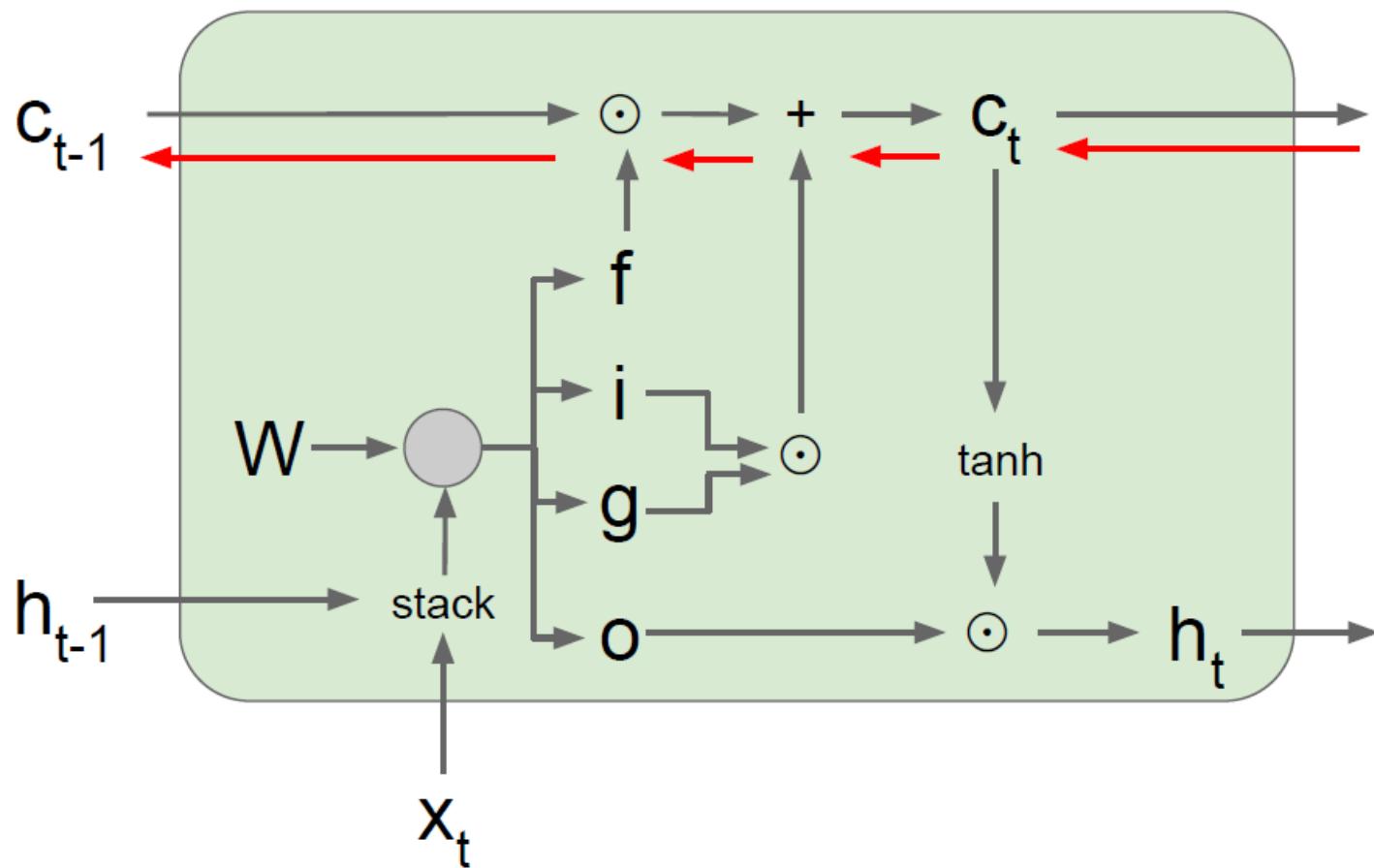
LSTM Gradient Flow



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

LSTM Gradient Flow

- Gradient가 흘러갈 때 forget gate의 값이 계속 곱해짐(매번 다른값)



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

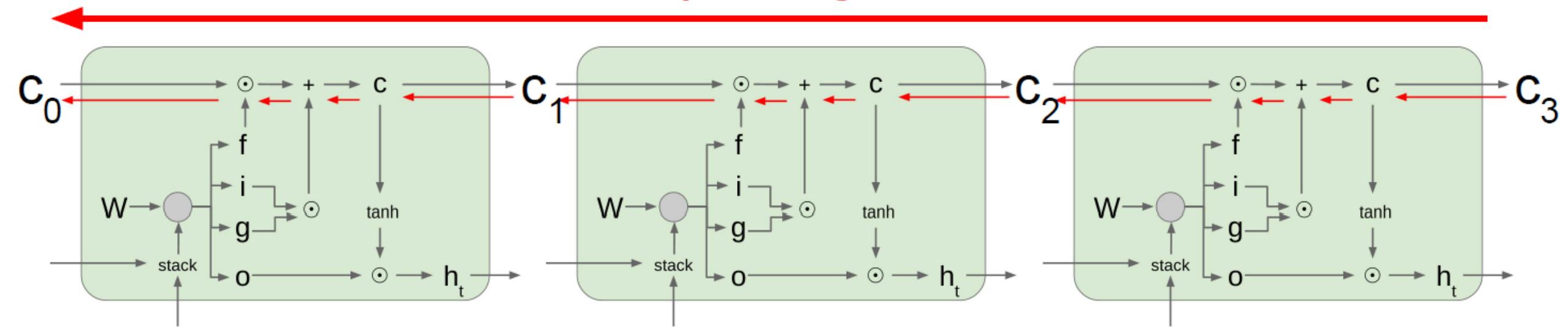
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

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

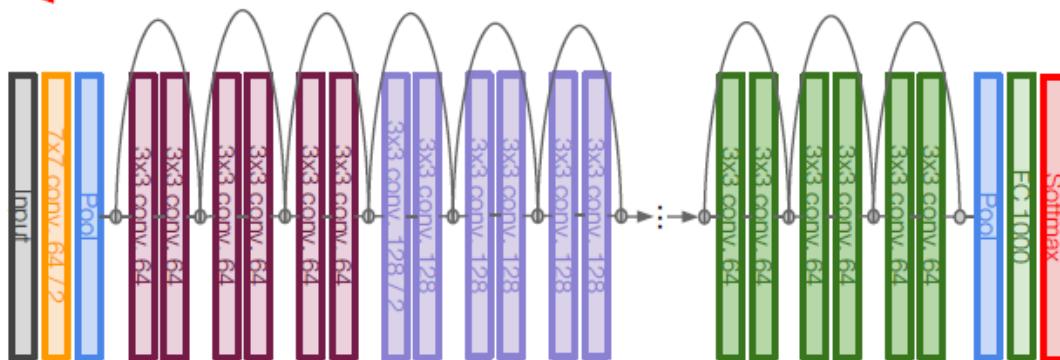
$$h_t = o \odot \tanh(c_t)$$

LSTM Gradient Flow

Uninterrupted gradient flow!



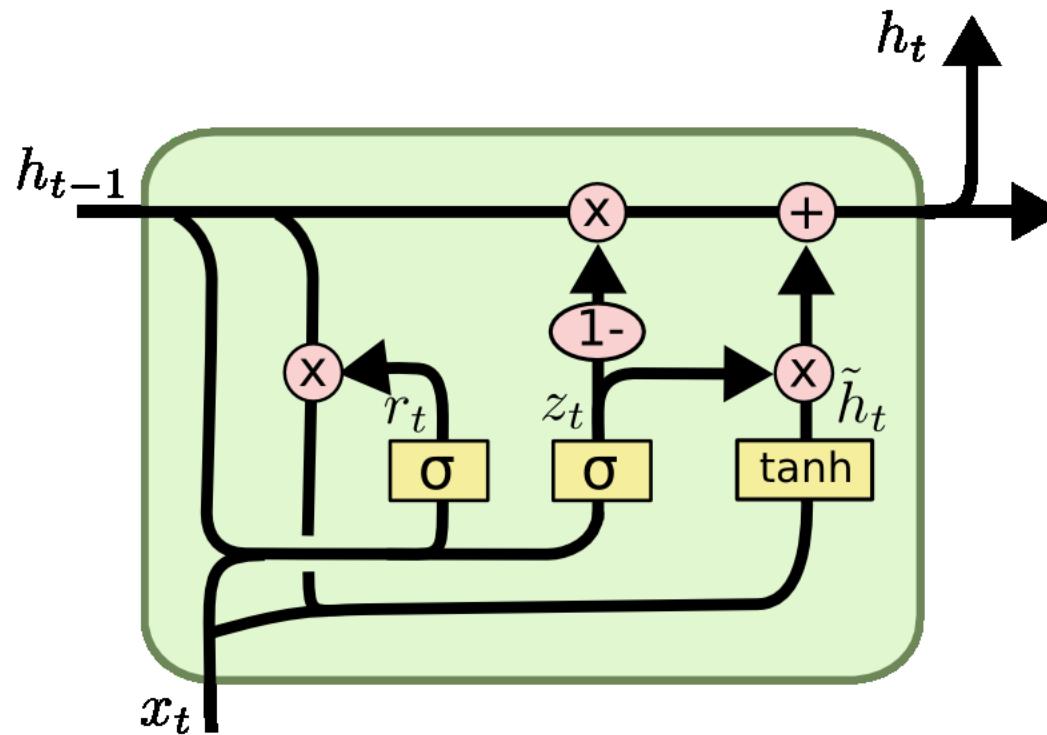
Similar to ResNet!



LSTM's Problems?

- Parameter가 너무 많고 복잡한데,
뭔가 더 줄일 수 있는 여지가 없을까?
- gate를 곱해서 0~1사이의 non-linearity를 주는데,
굳이 따로 activation function이 필요할까?
- Gate 수를 좀 줄여볼 수는 없을까?

Gated Recurrent Unit



Update gate

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

Reset gate

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

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

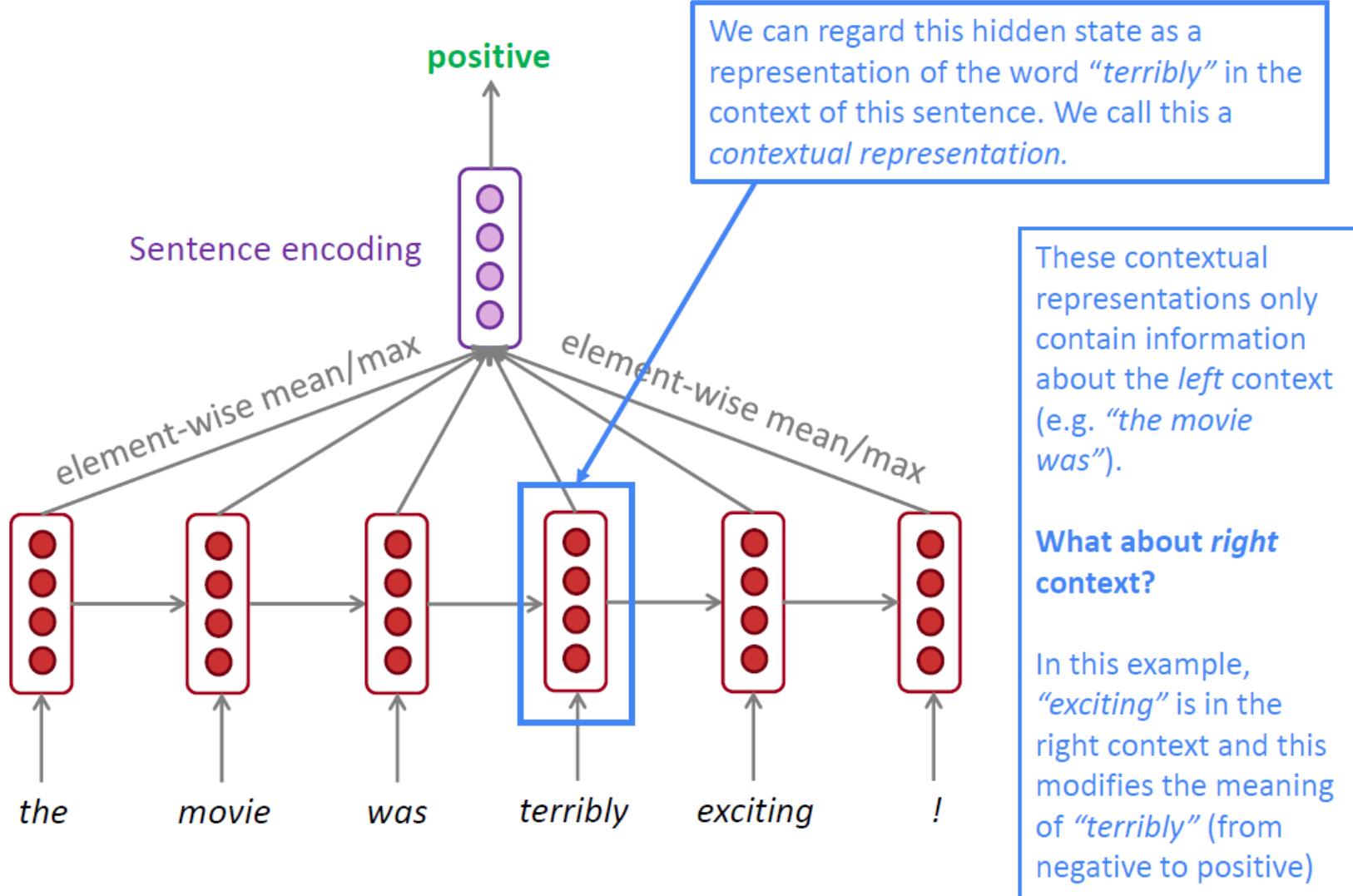
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

LSTM vs GRU

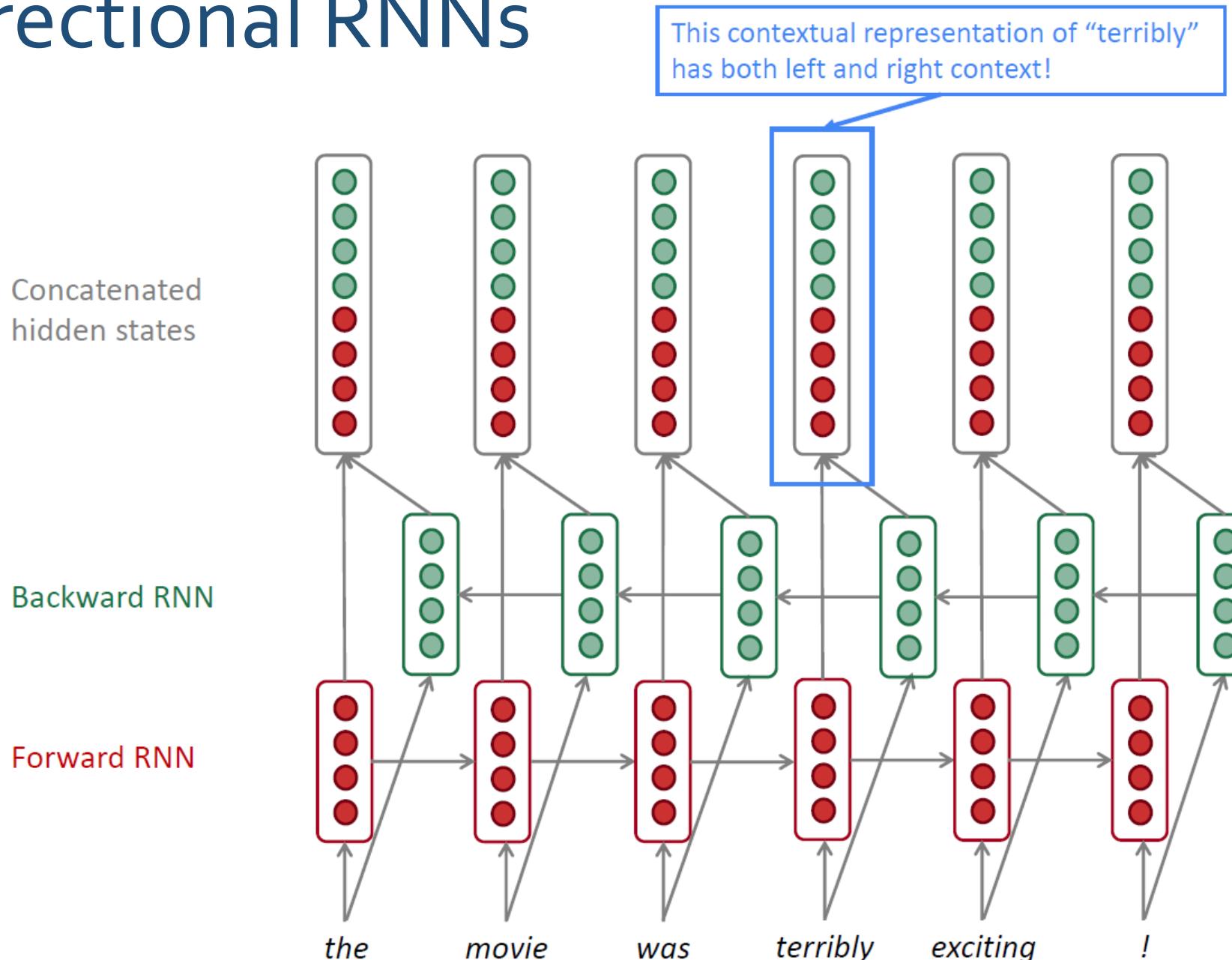
- Researchers have proposed many gated RNN variants, but **LSTM and GRU are the most widely used**
- The biggest difference is that **GRU is quicker to compute** and has fewer parameters
- There is no conclusive evidence that one consistently performs better than the other
- **LSTM is a good default choice** (especially if your data has particularly long dependencies, or you have lots of training data)
- **Rule of thumb** : start with LSTM, but switch to GRU if you want something more efficient

Bidirectional RNNs: Motivation

Task: Sentiment Classification



Bidirectional RNNs



Bidirectional RNNs

- Note: bidirectional RNNs are only applicable if you have access to the **entire input sequence**.
 - They are not applicable to Language Modeling, because in LM you only have left context available.
- If you do have entire input sequence (e.g. any kind of encoding), **bidirectionality is powerful** (you should use it by default).

Multi-layer RNNs

- RNNs are already “deep” on one dimension (they unroll over many timesteps).
- We can also make them “deep” in another dimension by **applying multiple RNNs** – this is a multi-layer RNN.
- This allows the network to compute **more complex representations**.
 - The **lower RNNs** should compute **lower-level features** and the **higher RNNs** should compute **higher-level features**.
- Multi-layer RNNs are also called **stacked RNNs**.

Multi-layer RNNs

The hidden states from RNN layer i
are the inputs to RNN layer $i+1$

