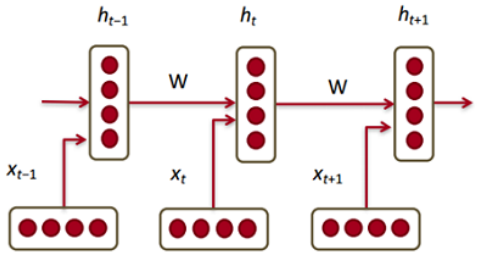
Natural Language Processing & RNN



$x_{shirt} - x_{clothing} \approx x_{chair} - x_{furniture}$

$x_{king} - x_{man} \approx x_{queen} - x_{woman}$

$$\log p(o|c) = \log \frac{\exp\left(u_o^T v_c\right)}{\sum_{w=1}^{W} \exp\left(u_w^T v_c\right)}$$

Sequence Data

- Alphabet 을 z부터 a까지 거꾸로 외워봅시다

- Traditional multilayer perceptron neural networks make the assumption that all inputs are independent of each other
- This assumption breaks down in the case of sequence data

Language Model

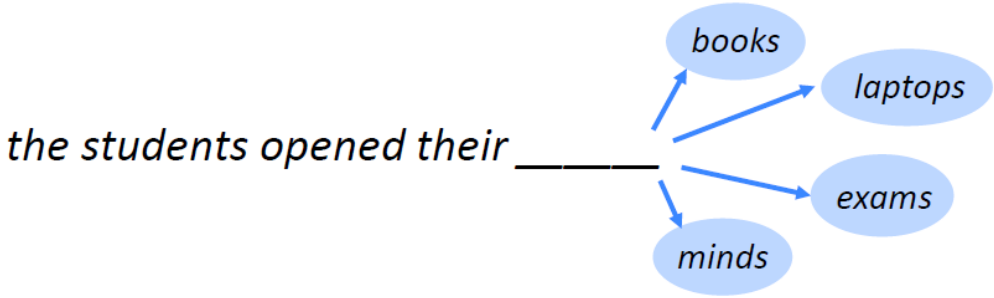
what is the

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**

Google Search I'm Feeling Lucky

Language Modeling

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



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

$$P(\boldsymbol{x}^{(t+1)} | \boldsymbol{x}^{(t)}, \ldots, \boldsymbol{x}^{(1)})$$

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

- 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 $x^{(1)}, x^{(2)}, \ldots, x^{(T)}$, then the probability of this text(according to the Language Model) is :

$$P(\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(T)}) = P(\boldsymbol{x}^{(1)}) \times P(\boldsymbol{x}^{(2)} | \boldsymbol{x}^{(1)}) \times \cdots \times P(\boldsymbol{x}^{(T)} | \boldsymbol{x}^{(T-1)}, \ldots, \boldsymbol{x}^{(1)})$$

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

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.

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

$$= \frac{P(\boldsymbol{x}^{(t+1)},\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(t-n+2)})}{P(\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(t-n+2)})} \quad \text{(definition of conditional prob)}$$

prob of a n-gram

prob of a (n-1)-gram

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

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

N-gram Language Models

- Suppose we are learning 4-gram Language Model.

~~as the proctor started the clock, the~~ *students opened their* _____

discard

condition on this

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

For example, suppose that in the corpus:

- "students opened their" occurred 1000 times
- "students opened their books" occurred 400 times
  - → P(books | students opened their) = 0.4
- "students opened their exams" occurred 100 times
  - → P(exams | 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 probabilitv o!

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

- Inability to capture long-term dependencies.

~~*as the proctor started the clock, the*~~ *students opened their* _____

discard

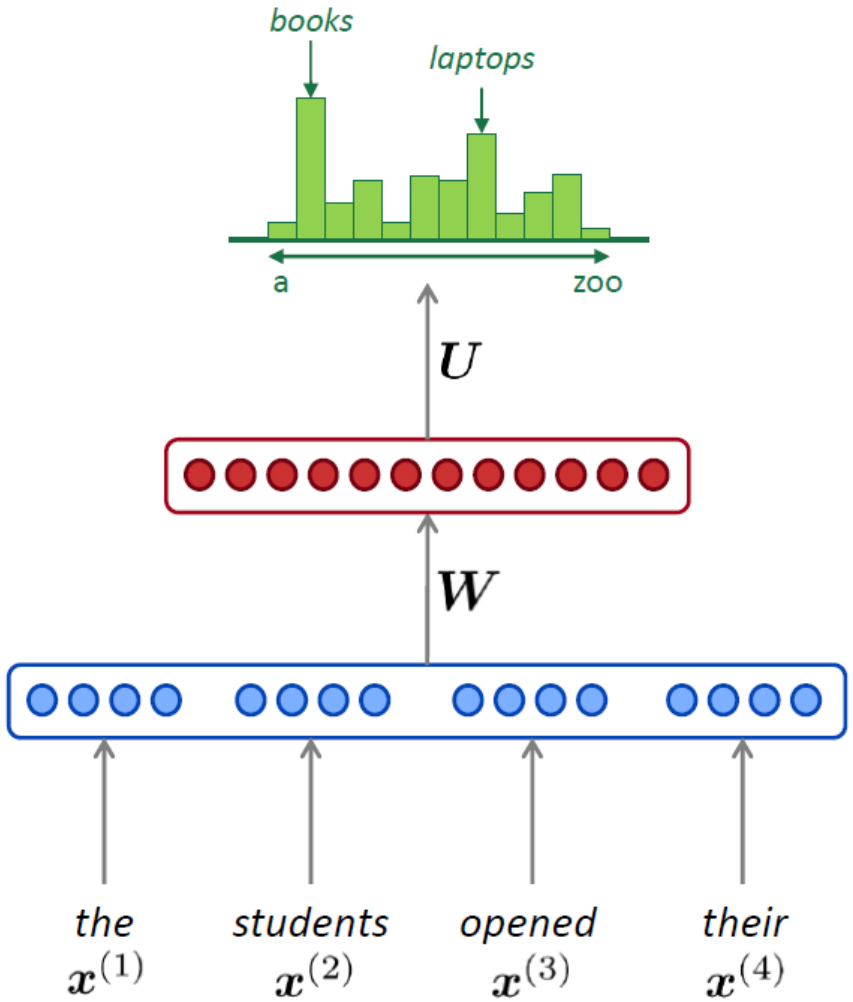
condition on this

- Need to store count for all n-grams you saw in the corpus. Increasing n or increasing corpus increases model size.

How to Build a Neural Language Model?

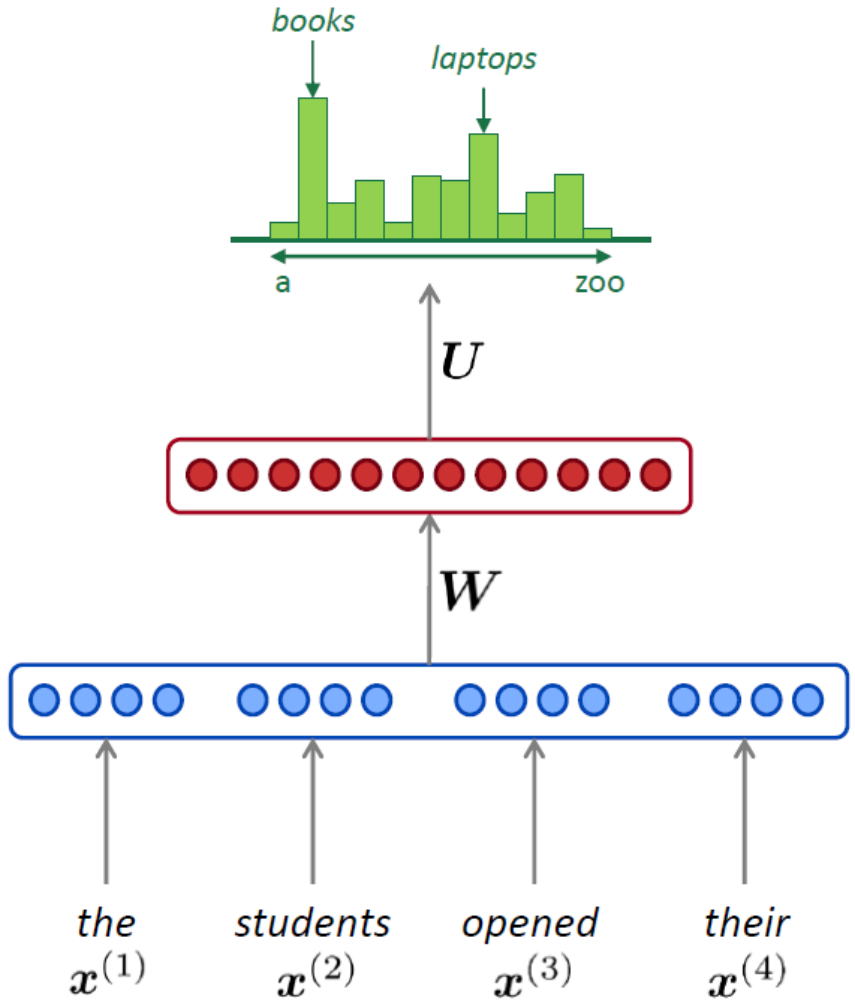
- Recall the Language Modeling task:
  - Input: sequence of words $x^{(1)}, x^{(2)}, \ldots, x^{(t)}$
  - Output: prob dist of the next word $P\left(x^{(t+1)} \middle| x^{(t)}, \ldots, x^{(1)}\right)$

- How about a window-based neural model?



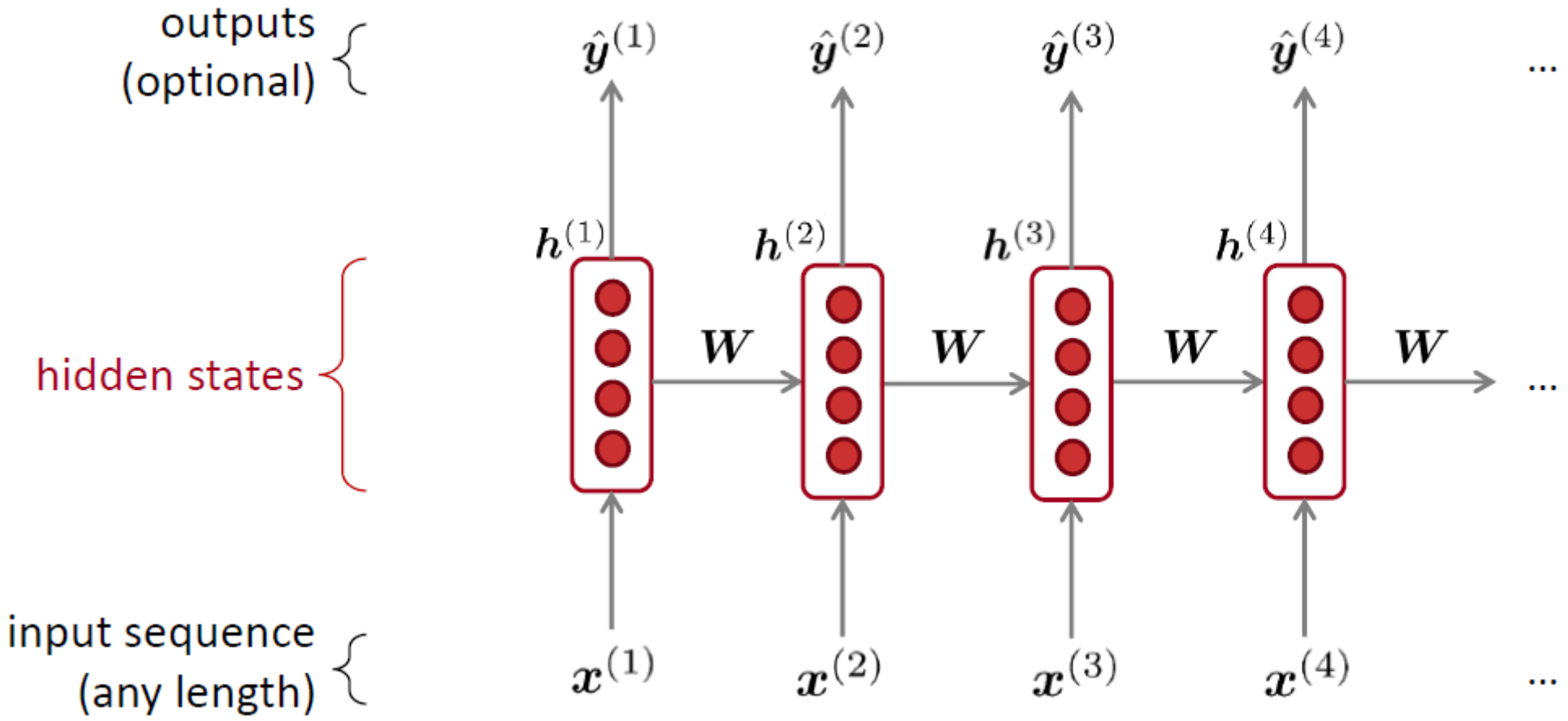
A Fixed-Window Neural Language Model

- 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

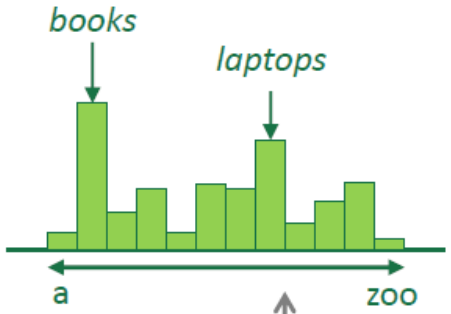


$$\hat{\boldsymbol{y}}^{(4)} = P(\boldsymbol{x}^{(5)}|\text{the students opened their})$$

A RNN Language Model

output distribution

$$\hat{\boldsymbol{y}}^{(t)} = \text{softmax}\left(\boldsymbol{U}\boldsymbol{h}^{(t)} + \boldsymbol{b}_2\right) \in \mathbb{R}^{|V|}$$



hidden states

$$\boldsymbol{h}^{(t)} = \sigma\left(\boldsymbol{W}_h\boldsymbol{h}^{(t-1)} + \boldsymbol{W}_e\boldsymbol{e}^{(t)} + \boldsymbol{b}_1\right)$$

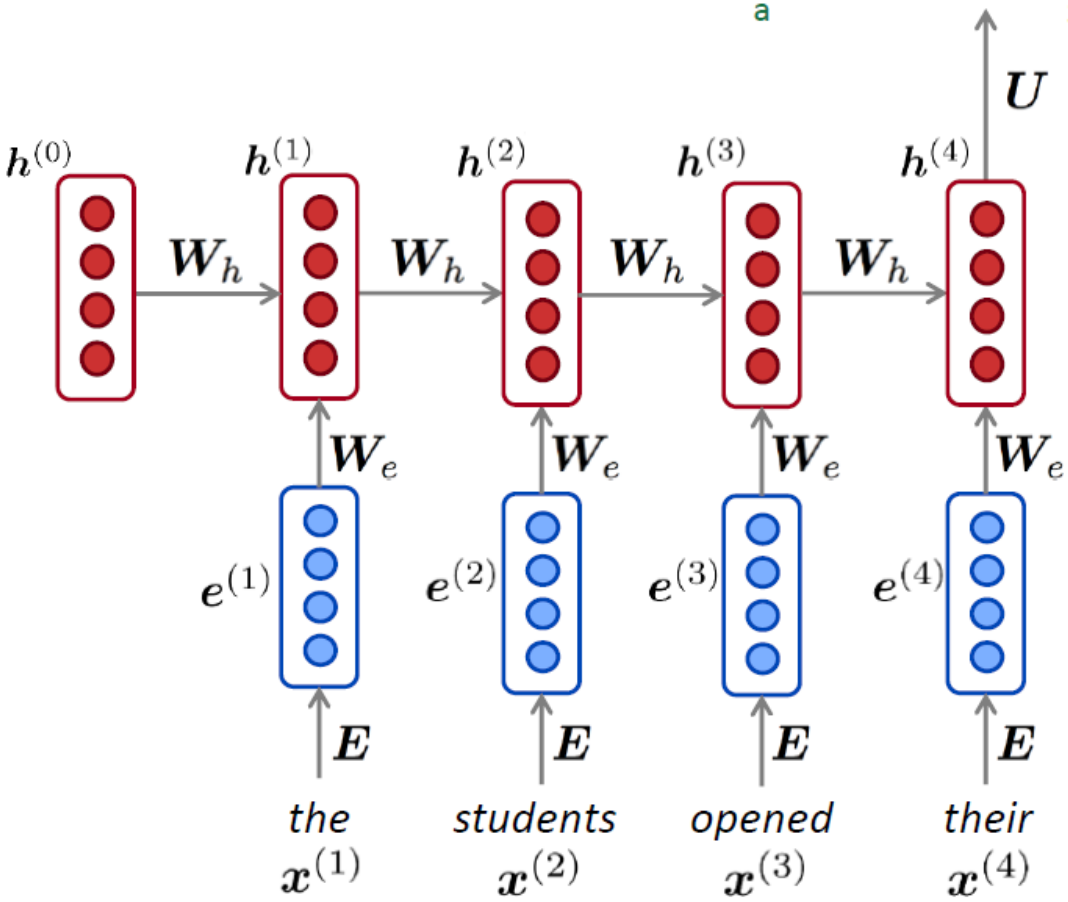
$\boldsymbol{h}^{(0)}$ is the initial hidden state

word embeddings

$$\boldsymbol{e}^{(t)} = \boldsymbol{E}\boldsymbol{x}^{(t)}$$

words / one-hot vectors

$$\boldsymbol{x}^{(t)} \in \mathbb{R}^{|V|}$$



***Note**: this input sequence could be much longer, but this slide doesn't have space!*

A RNN Language Model

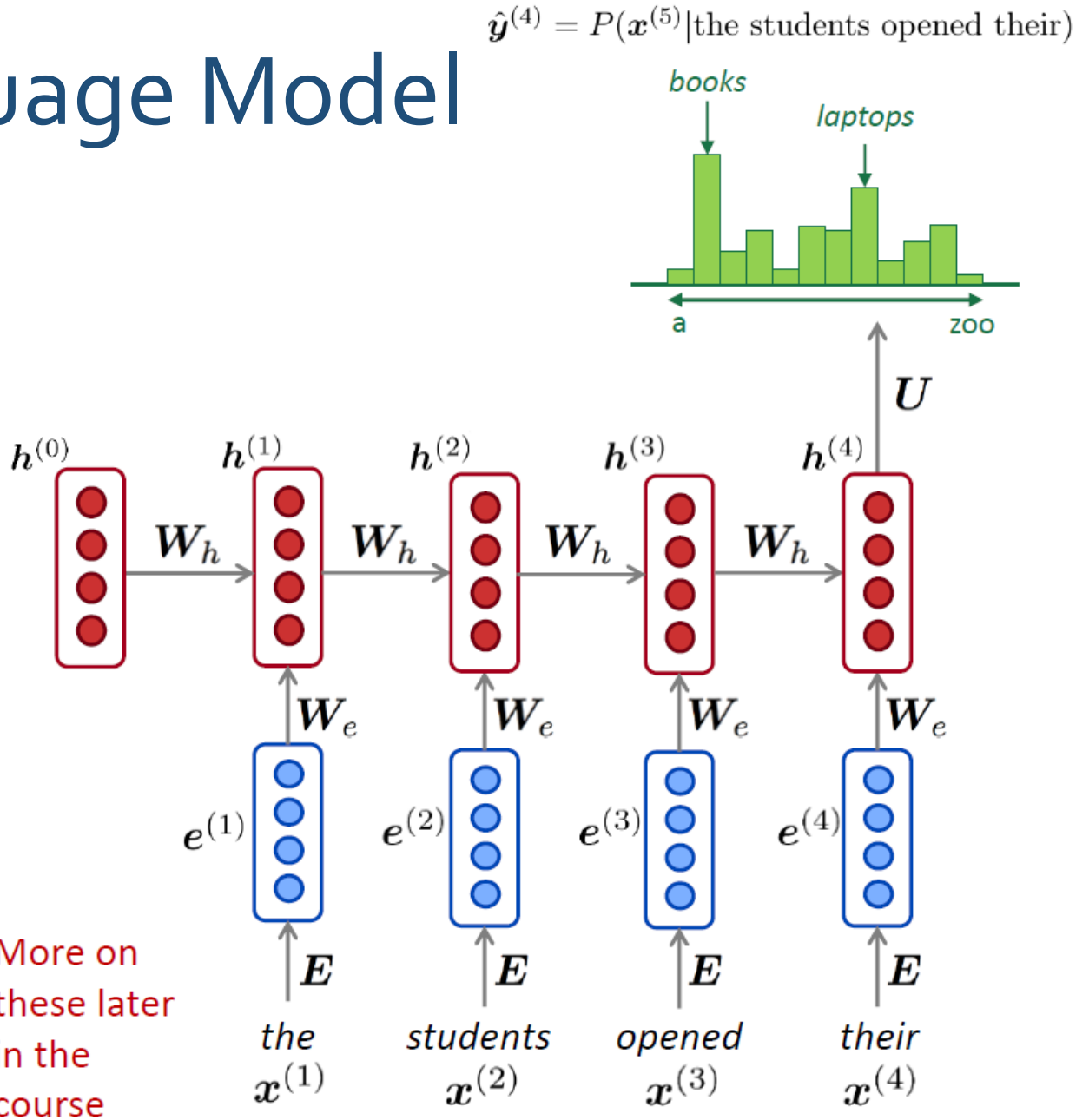
RNN **Advantages**:

- 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.

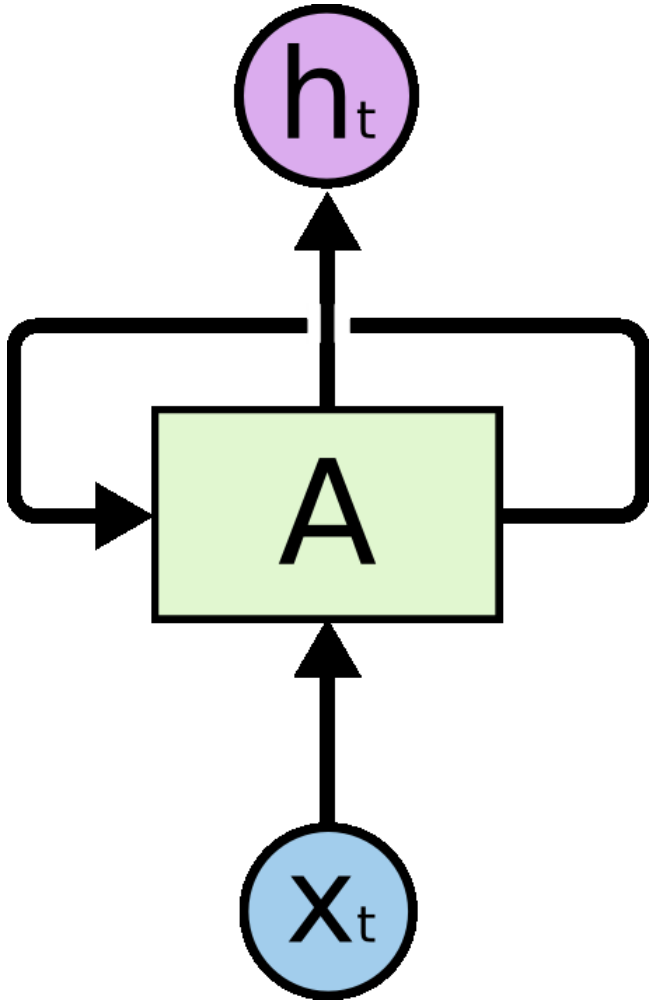
RNN **Disadvantages**:

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

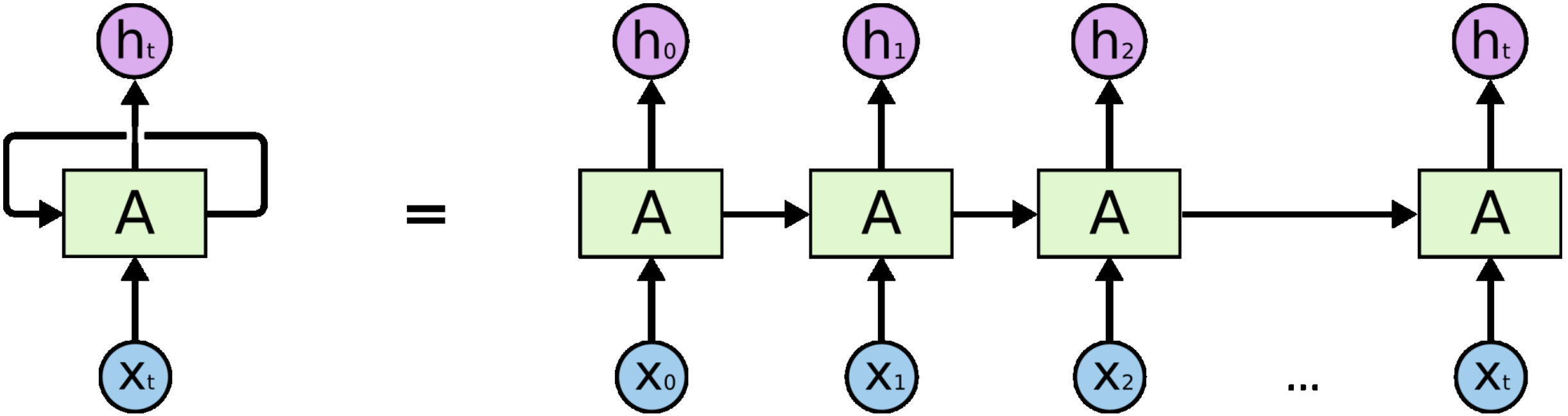
More on these later in the course



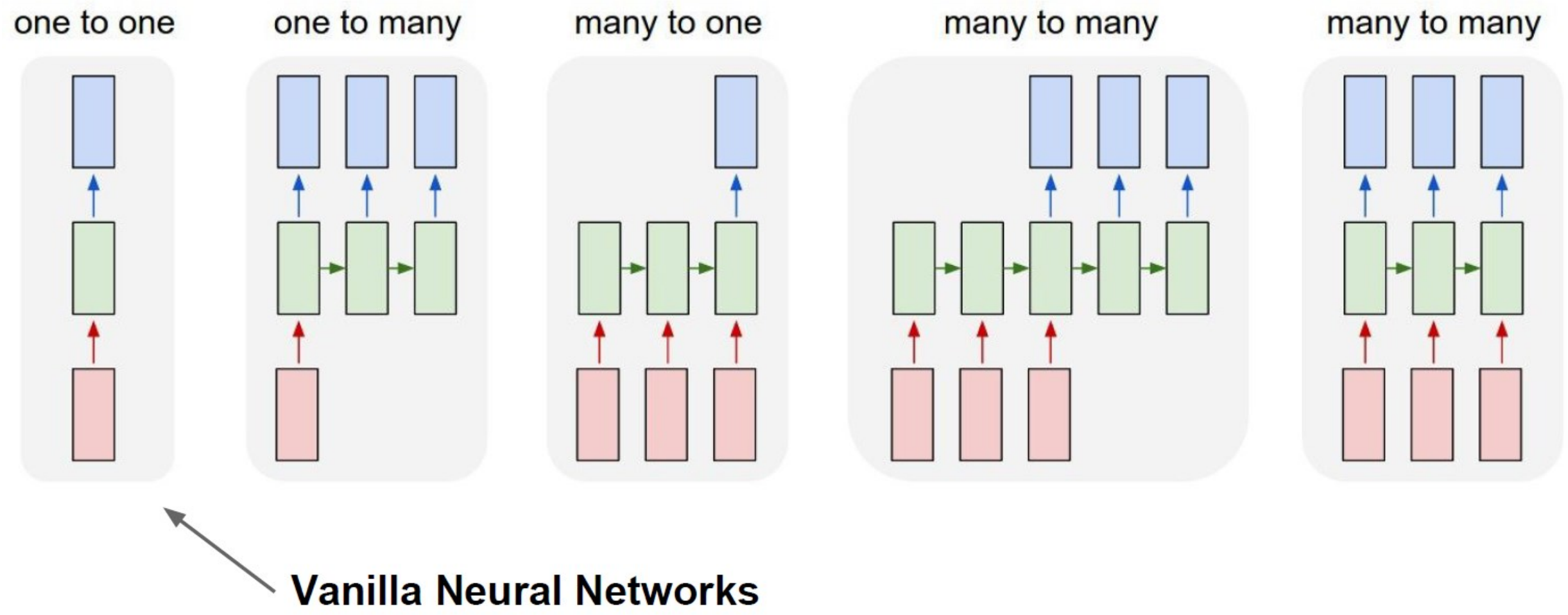
Recurrent Neural Network



Recurrent Neural Network



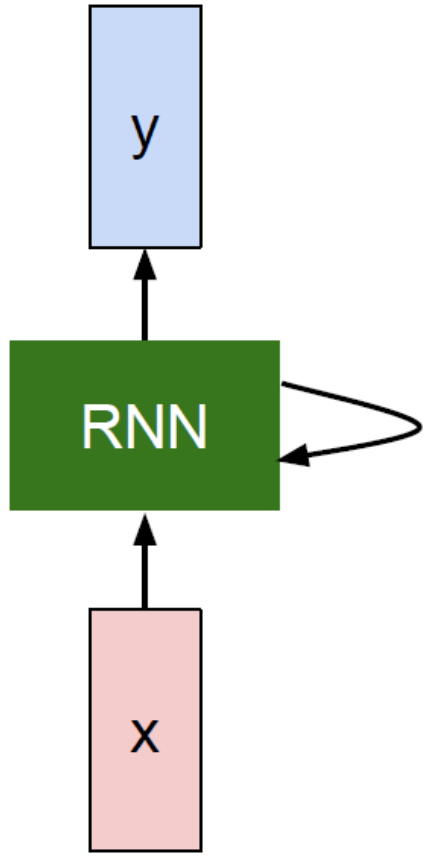
RNN's flexibility



Recurrent Neural Network

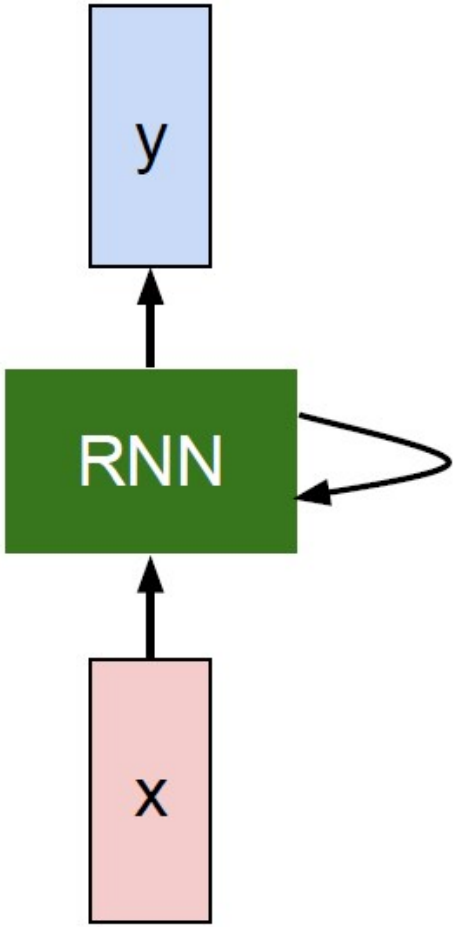
We can process a sequence of vectors **x** by applying a recurrence formula at every time step:

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



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

Recurrent Neural Network



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

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

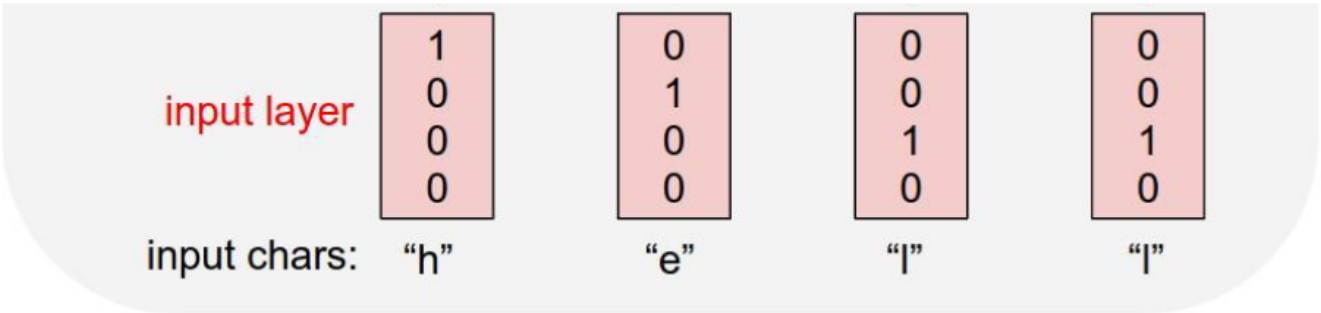
$$y_t = W_{hy}h_t$$

RNN

Character-level language model example

Vocabulary:
[h,e,l,o]

Example training sequence:
“hello”



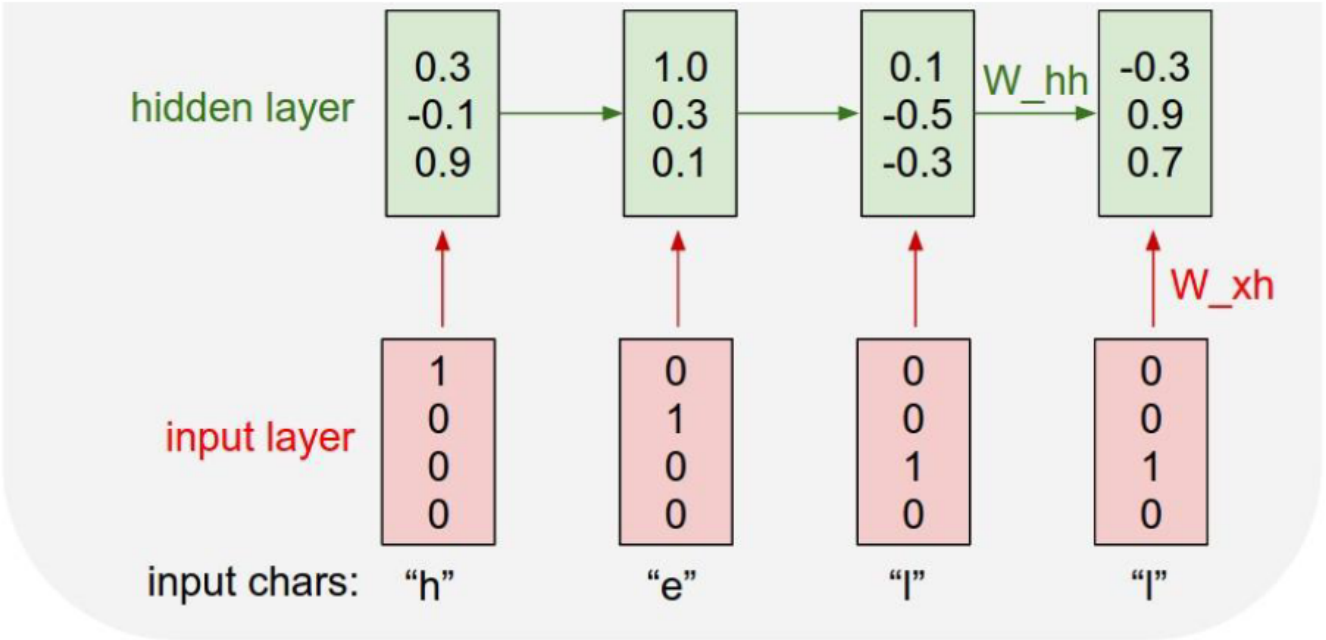
RNN

Character-level language model example

Vocabulary:
[h,e,l,o]

Example training sequence:
“hello”

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

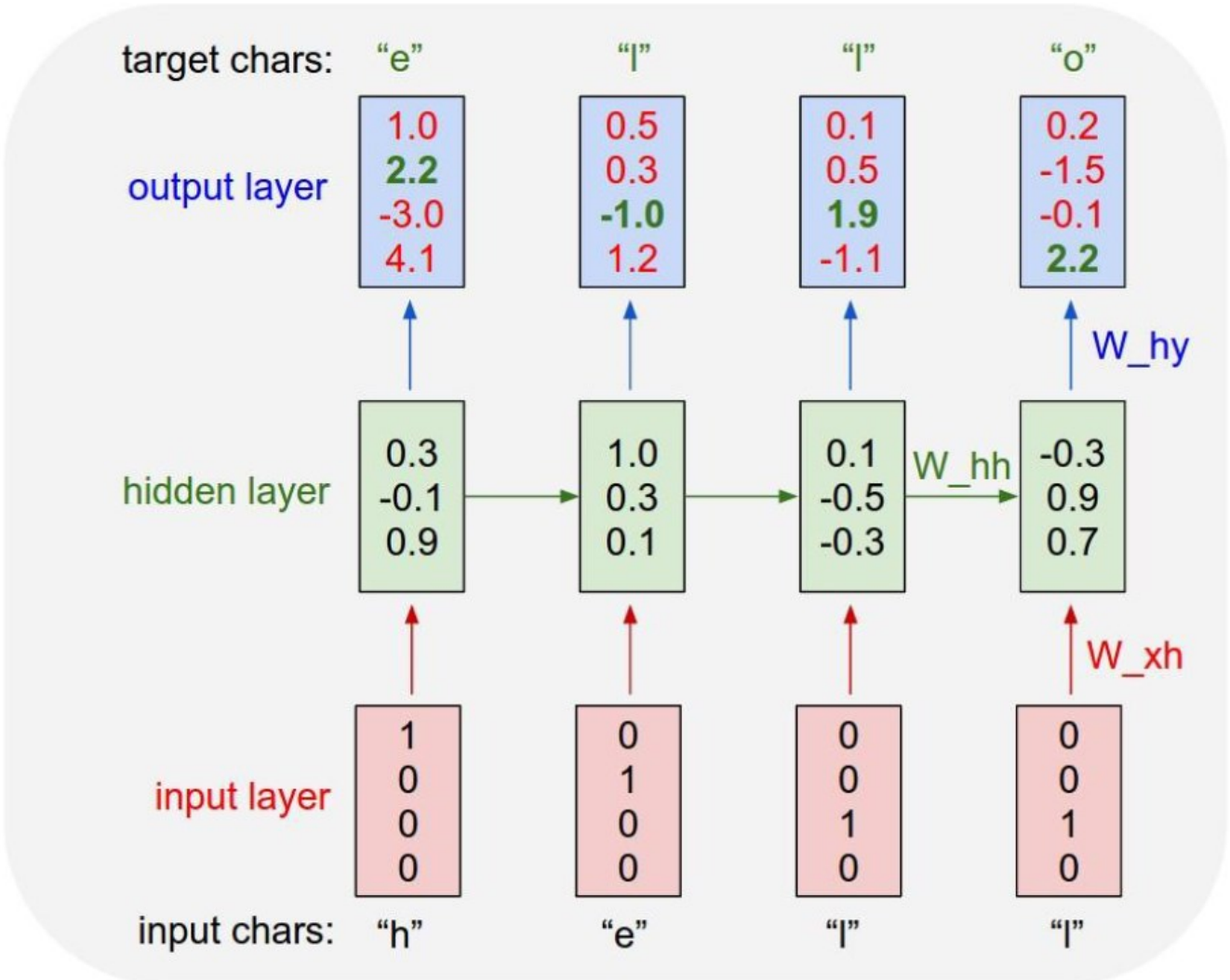


RNN

Character-level language model example

Vocabulary:
[h,e,l,o]

Example training sequence:
“hello”



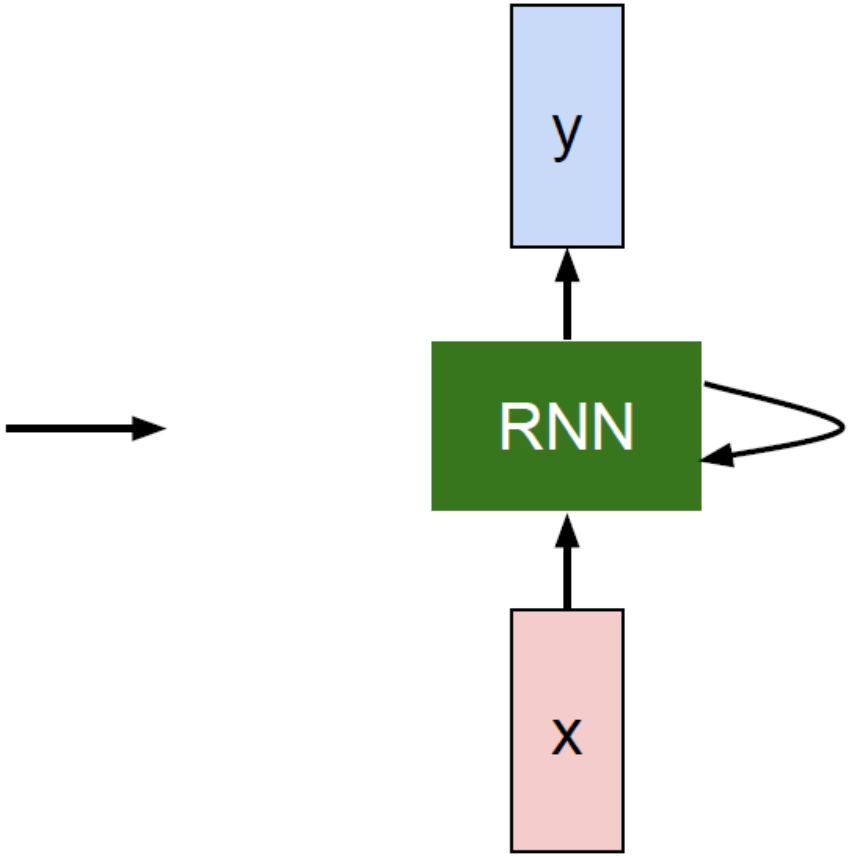
Language Model

THE SONNETS

by William Shakespeare

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

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



Language Model

at first:

```
tyntd-iafhatawiaoihrdemot  lytdws  e ,tfti, astai f ogoh eoase rrranbyne 'nhthnee e
plia tklrgd t o idoe ns,smtt   h ne etie h,hregtrs nigtike,aoaenns lng
```

train more

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

train more

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

train more

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

Language Model

```
PANDARUS:
Alas, I think he shall be come approached and the day
When little srain would be attain'd into being never fed,
And who is but a chain and subjects of his death,
I should not sleep.

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

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

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

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

VIOLA:
I'll drink it.
```

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

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

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 $X_{\acute{e}tale}$ we have

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

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

Lemma 0.2. *This is an integer $\mathcal{Z}$ is injective.*

Proof. See Spaces, Lemma ??. □

Lemma 0.3. *Let S be a scheme. Let X be a scheme and X is an affine open covering. Let $\mathcal{U} \subset \mathcal{X}$ be a canonical and locally of finite type. Let X be a scheme. Let X be a scheme which is equal to the formal complex.*

The following to the construction of the lemma follows.

Let X be a scheme. Let X be a scheme covering. Let

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

be a morphism of algebraic spaces over S and Y.

Proof. Let X be a nonzero scheme of X. Let X be an algebraic space. Let $\mathcal{F}$ be a quasi-coherent sheaf of $\mathcal{O}_X$-modules. The following are equivalent

(1) $\mathcal{F}$ is an algebraic space over S.
(2) If X is an affine open covering.

Consider a common structure on X and X the functor $\mathcal{O}_X(U)$ which is locally of finite type. □

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

$$\begin{array}{ccc} \mathcal{S} & \longrightarrow & \\ \downarrow & & \\ \xi & \longrightarrow & \mathcal{O}_{X'} \\ & & \uparrow \\ \mathrm{gor}_s & & \end{array} \qquad \begin{array}{ccc} & & \\ & \searrow & \\ & & \end{array}$$

$$\begin{array}{ccc} = \alpha' & \longrightarrow & \\ \updownarrow & & \\ = \alpha' & \longrightarrow & \alpha \end{array} \qquad \begin{array}{c} X \\ \downarrow \end{array}$$

$$\mathrm{Spec}(K_\psi) \qquad \mathrm{Mor}_{Sets} \quad \mathrm{d}(\mathcal{O}_{\mathcal{X}_{X/k}}, \mathcal{G})$$

is a limit. Then $\mathcal{G}$ is a finite type and assume S is a flat and $\mathcal{F}$ and $\mathcal{G}$ is a finite type f_*. This is of finite type diagrams, and

- the composition of $\mathcal{G}$ is a regular sequence,
- $\mathcal{O}_{X'}$ is a sheaf of rings.

□

Proof. We have see that $X = \mathrm{Spec}(R)$ and $\mathcal{F}$ is a finite type representable by algebraic space. The property $\mathcal{F}$ is a finite morphism of algebraic stacks. Then the cohomology of X is an open neighbourhood of U. □

Proof. This is clear that $\mathcal{G}$ is a finite presentation, see Lemmas ??.
A *reduced above* we conclude that U is an open covering of $\mathcal{C}$. The functor $\mathcal{F}$ is a "field

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

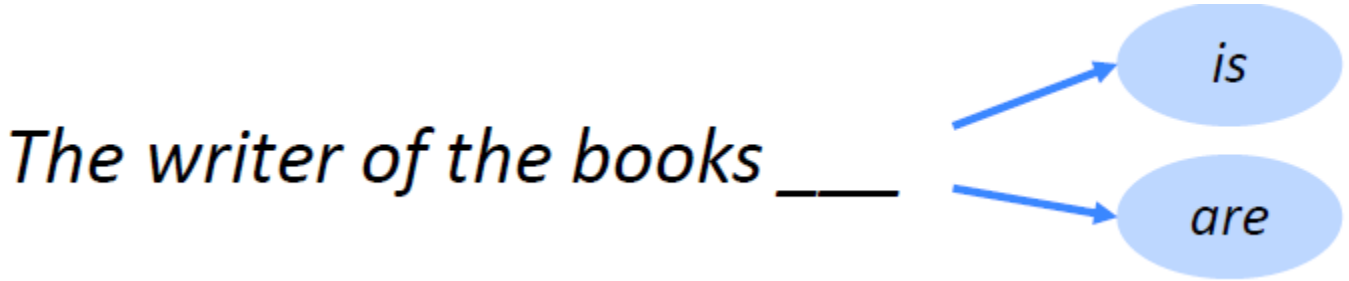
is an isomorphism of covering of $\mathcal{O}_{X_t}$. If $\mathcal{F}$ is the unique element of $\mathcal{F}$ such that X is an isomorphism.
The property $\mathcal{F}$ is a disjoint union of Proposition ?? and we can filtered set of presentations of a scheme $\mathcal{O}_X$-algebra with $\mathcal{F}$ 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.

Language Model

```
#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;
}
```

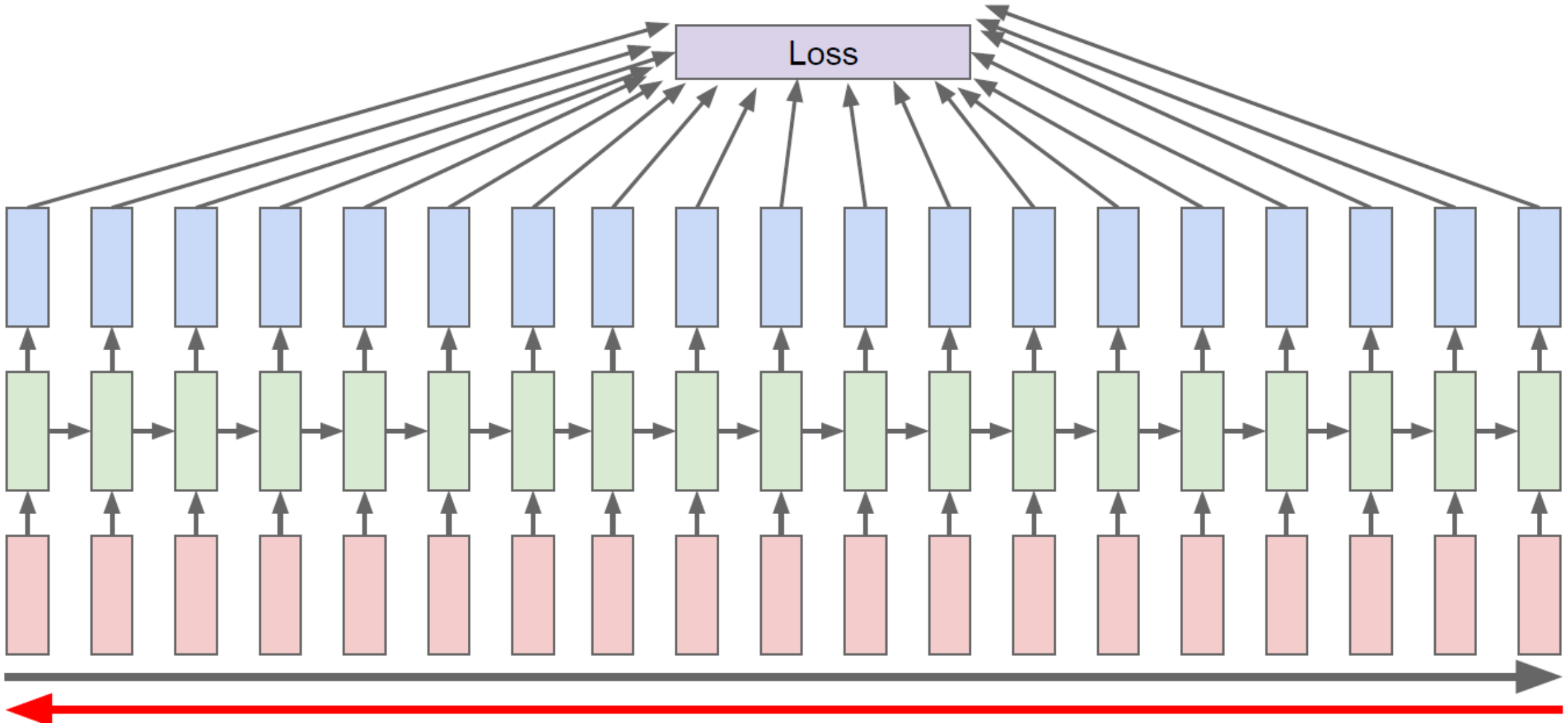
Is Long Term Dependency Problem Resolved?



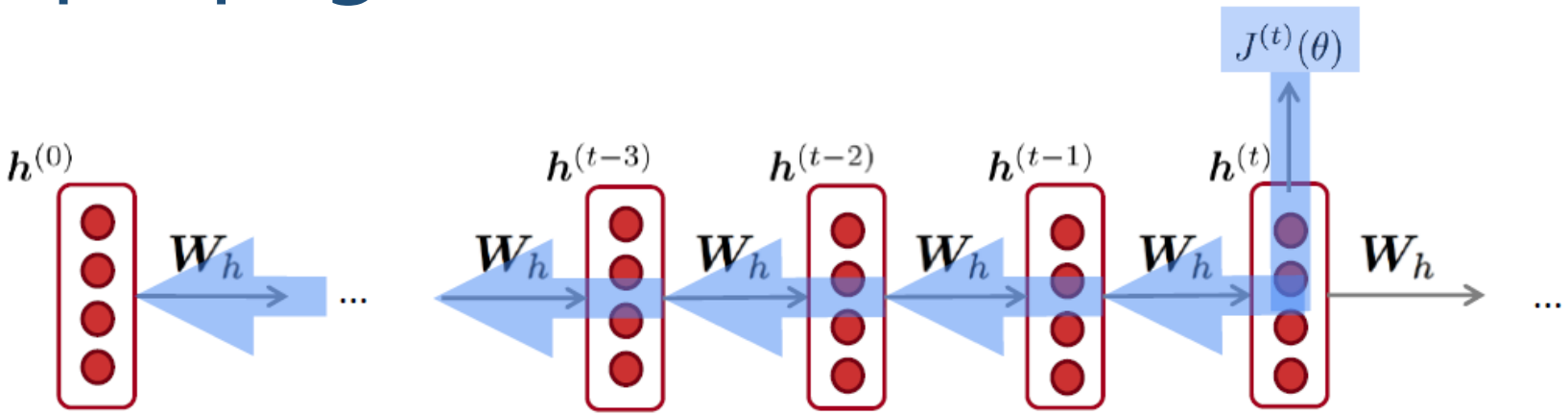
- RNN language model can choose “are”.

- Why? How can we train RNN?

Back Propagation Through Time



Backpropagation of RNNs

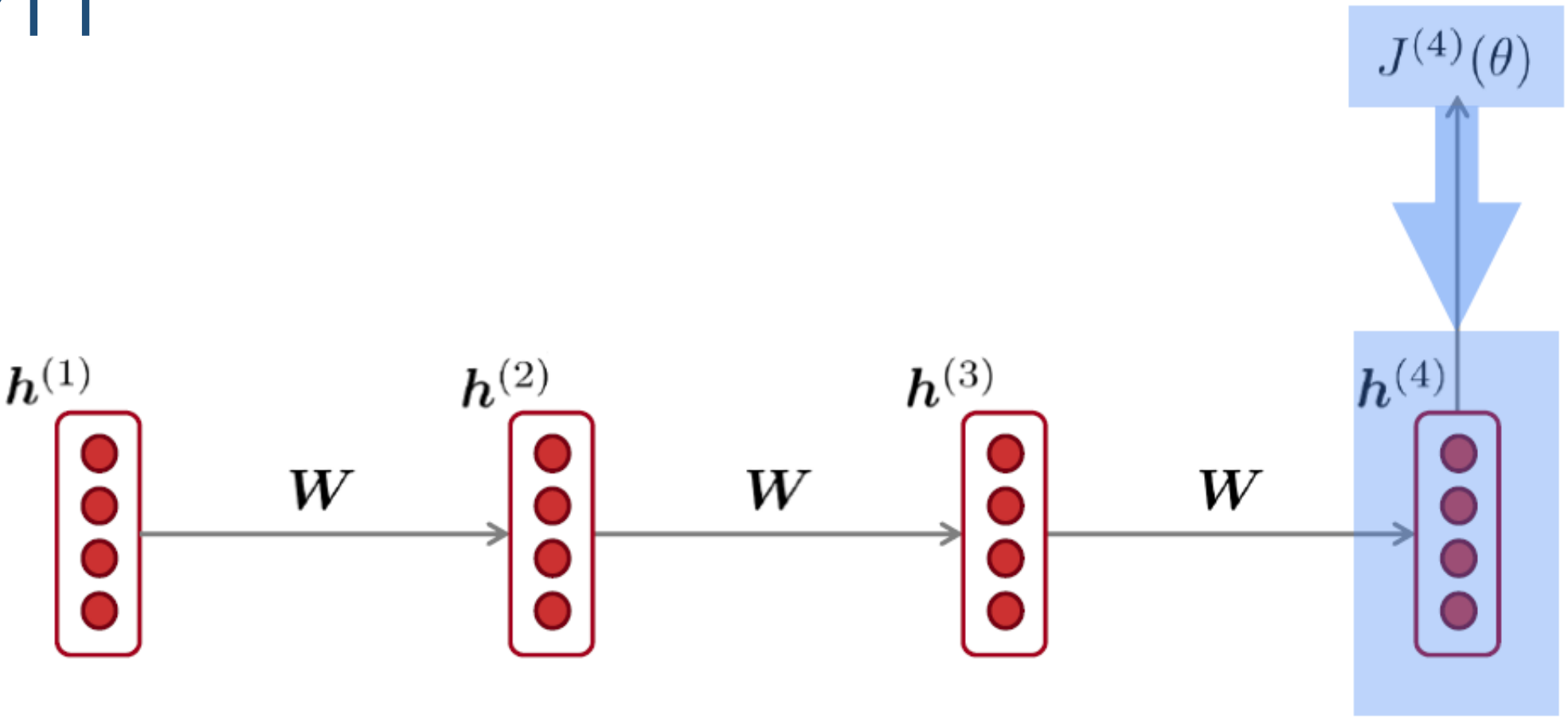


$$\frac{\partial J^{(t)}}{\partial \boldsymbol{W_h}} = \sum_{i=1}^{t} \frac{\partial J^{(t)}}{\partial \boldsymbol{W_h}}\Big|_{(i)}$$

Question: How do we calculate this?

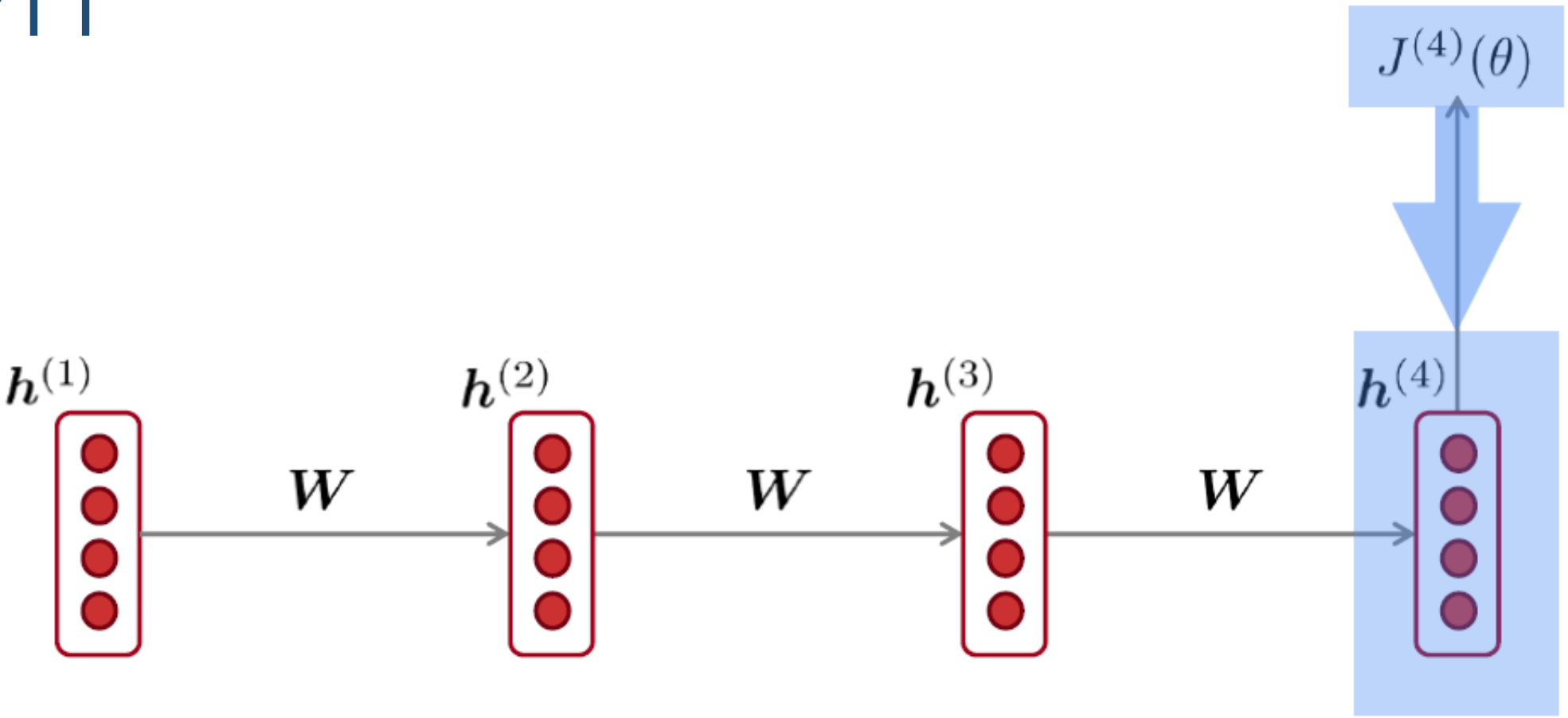
Answer: Backpropagate over timesteps *i*=*t*,…,0, summing gradients as you go.
This algorithm is called
"backpropagation through time"

BPTT



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

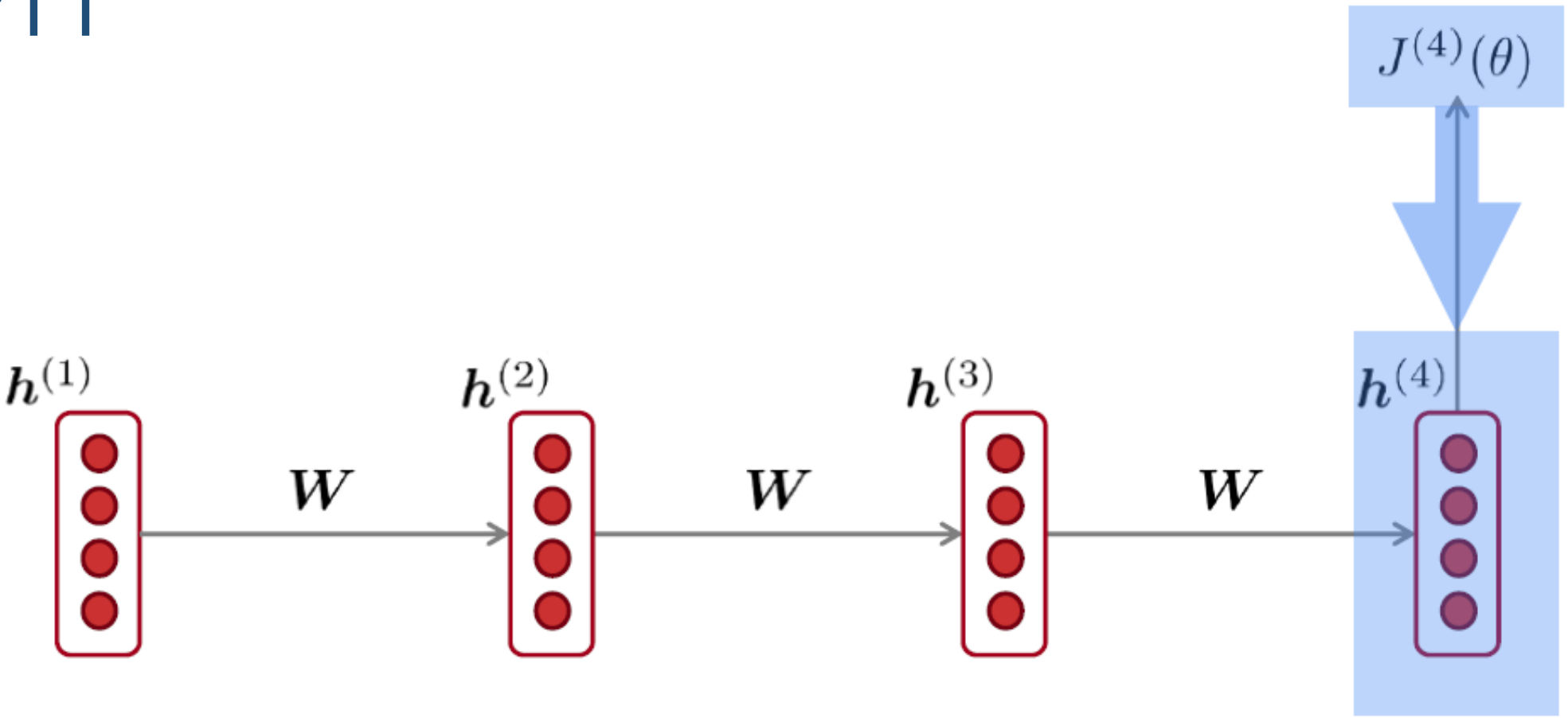
BPTT



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

chain rule!

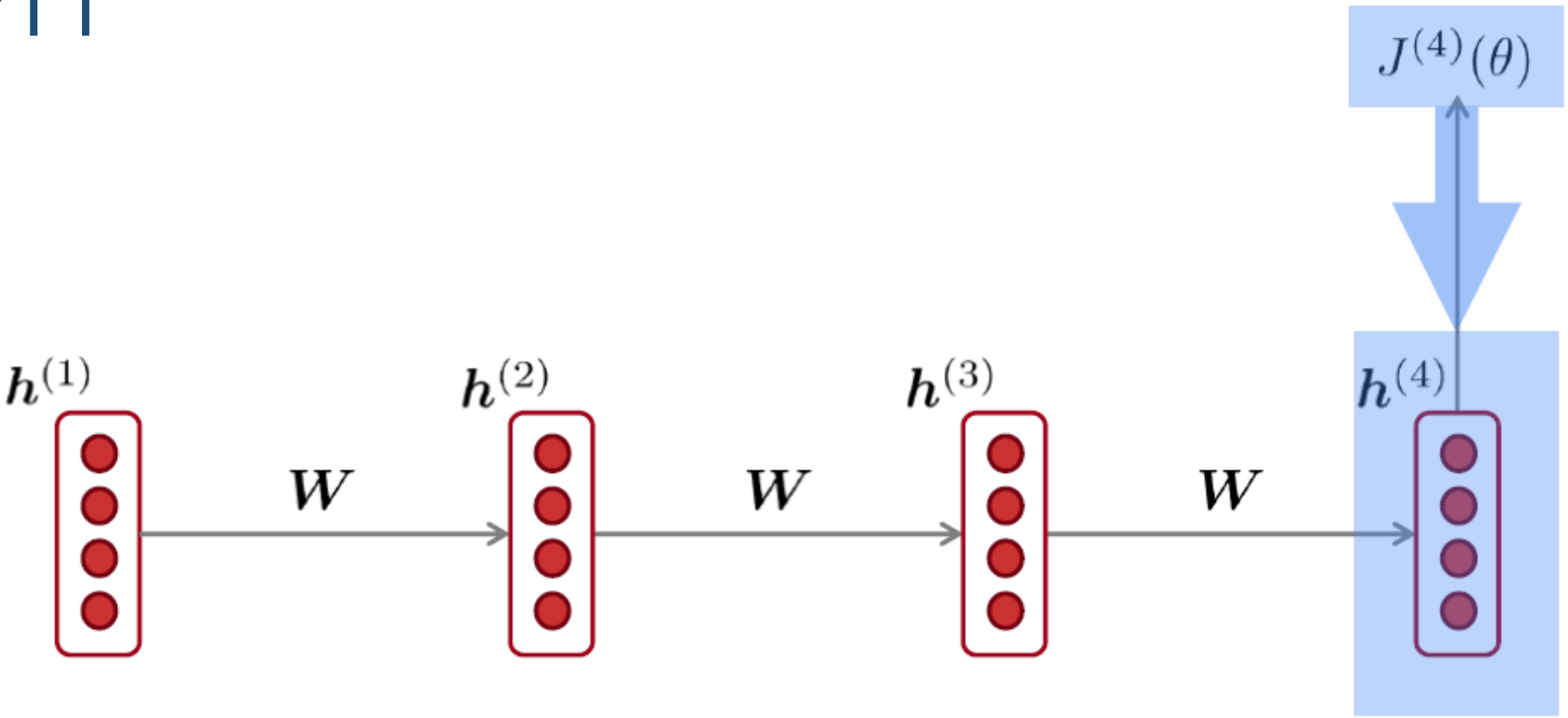
BPTT



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

chain rule!

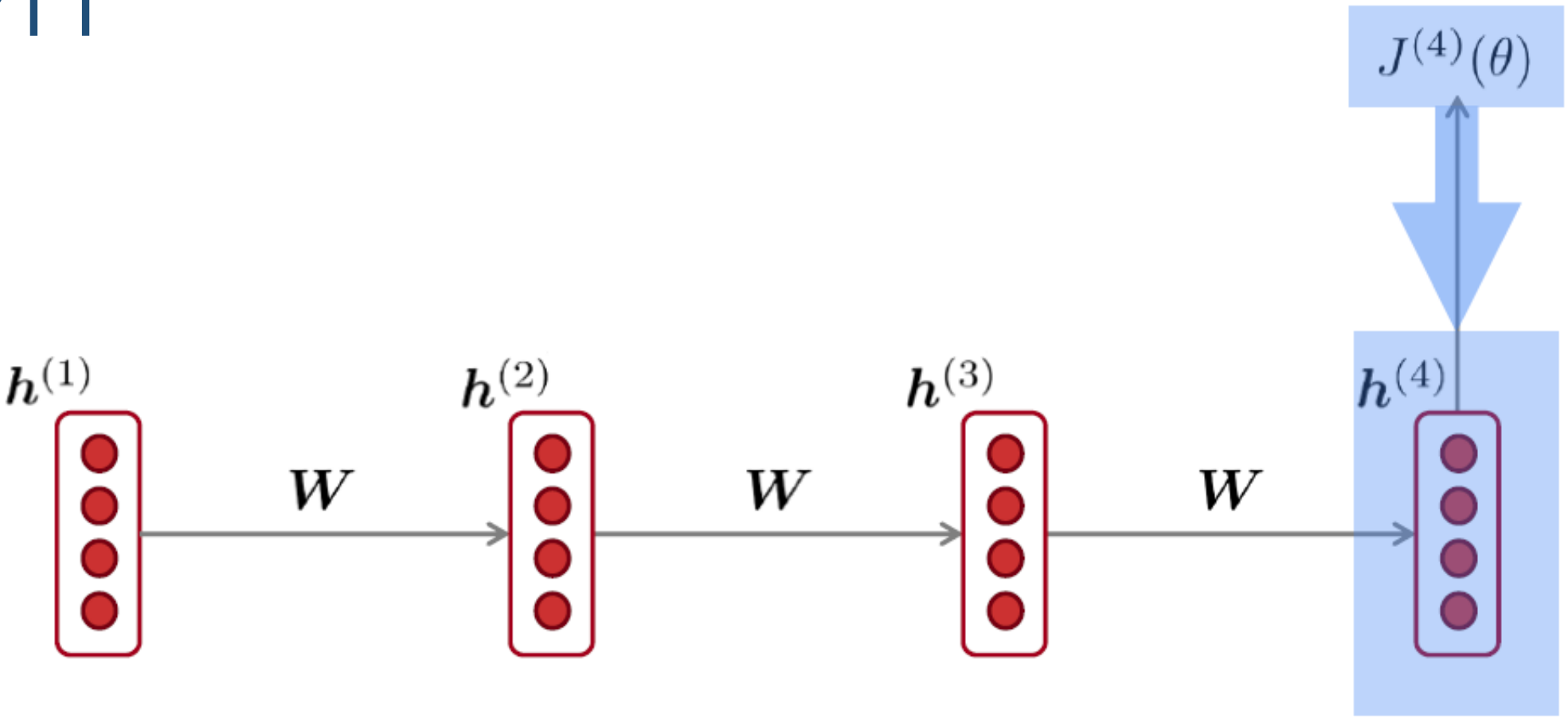
BPTT



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

chain rule!

BPTT



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

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: $h^{(t)} = \tanh\left(W_h h^{(t-1)} + W_x x^{(t)} + b\right)$
- $\frac{\partial h^{(t)}}{\partial h^{(t-1)}} = W_h \cdot \tanh'\left(W_h h^{(t-1)} + W_x x^{(t)} + b\right)$
- Therefore: $\frac{\partial J^{(4)}}{\partial h^{(1)}} = \frac{\partial J^{(4)}}{\partial h^{(4)}} \cdot {W_h}^3 \cdot \tanh'^3$

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

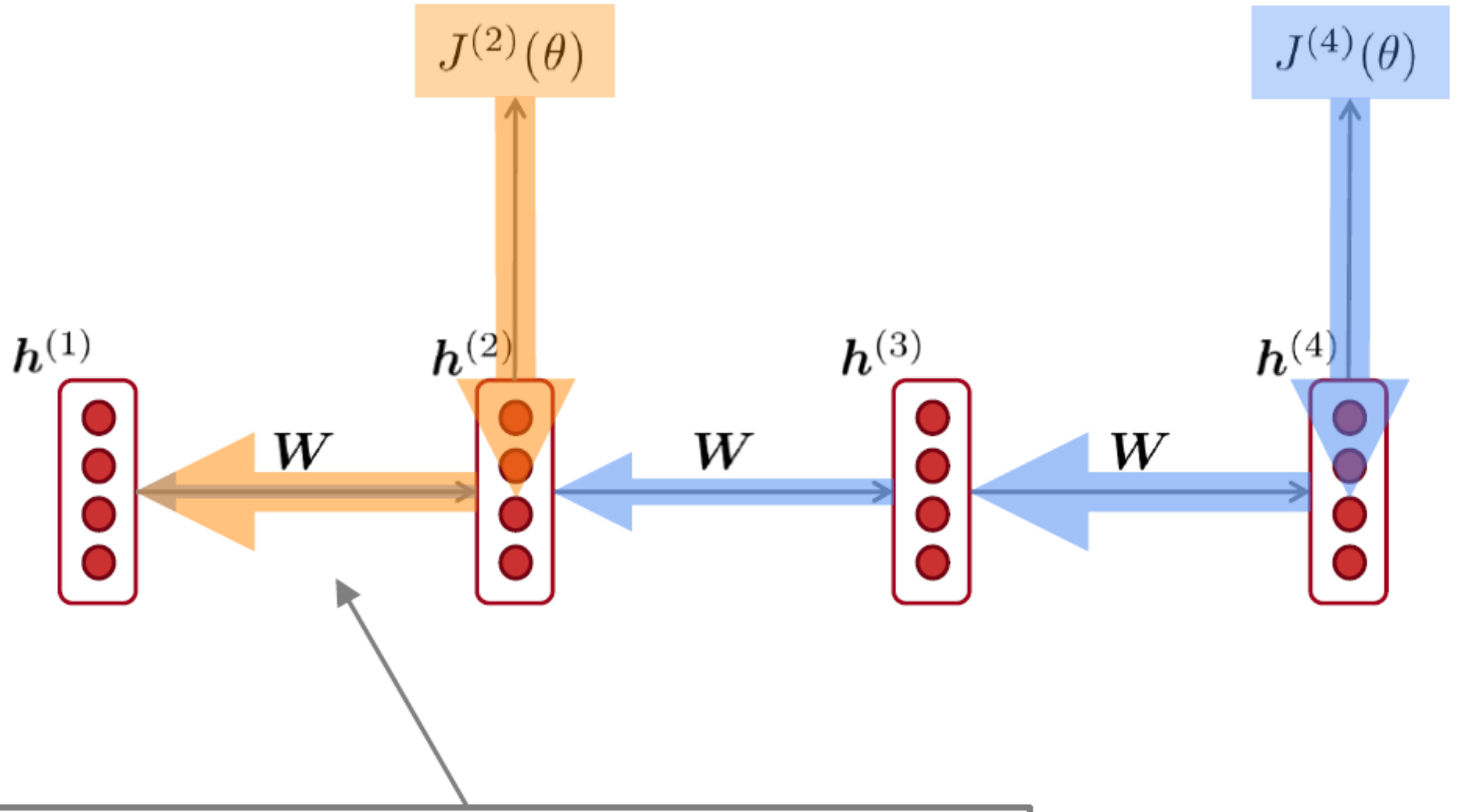
What happens if (n-m) is getting larger?

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 → vanishing gradient
- If the largest eigenvalue of W_h < 1 → vanishing gradient
- If the largest eigenvalue of W_h > 1 → 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.

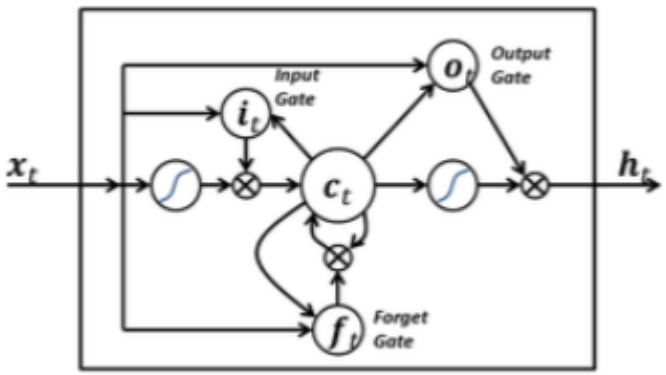
So model weights are only updated only with respect to near effects, not long-term effects.

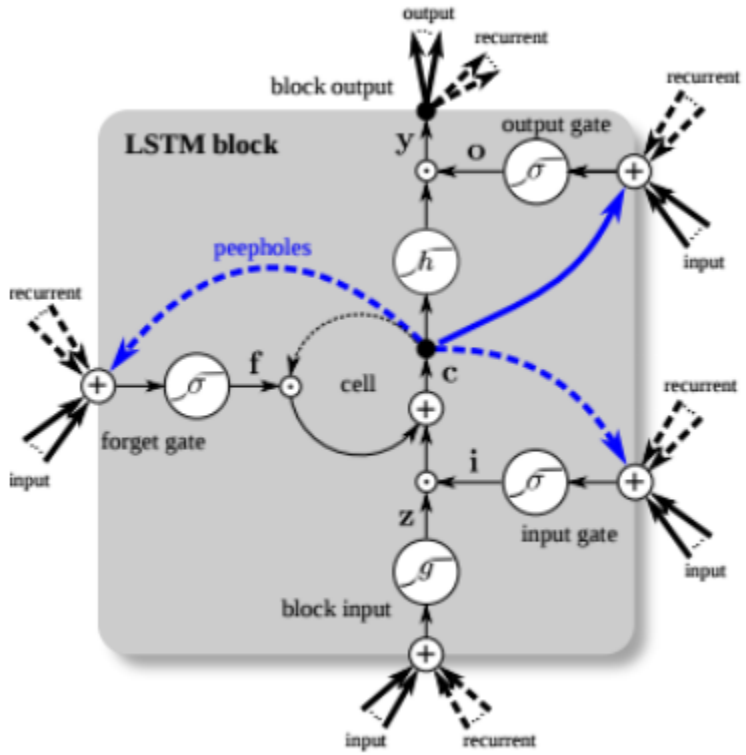
Why is Vanishing Gradient a Problem?

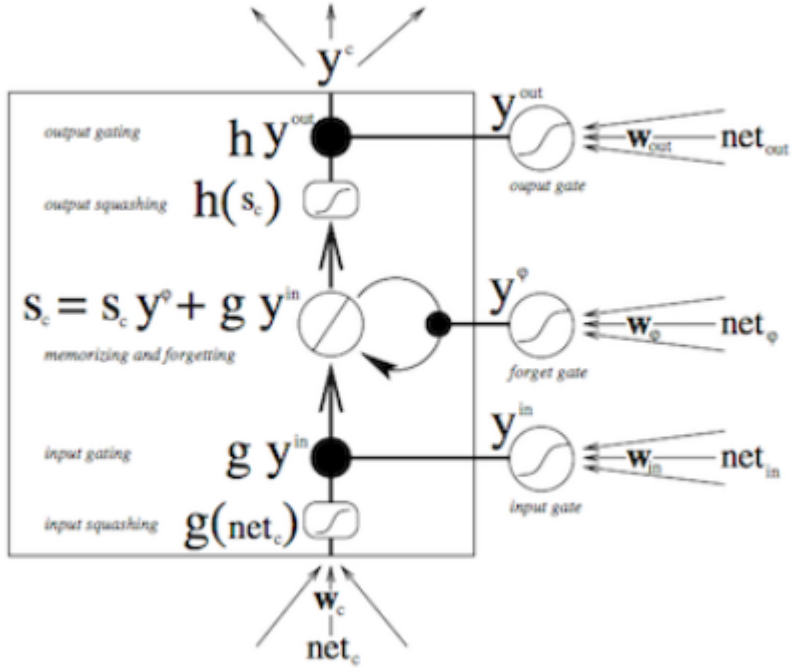
- Another explanation: Gradient can be viewed as a measure of the effect of the past on the future

- 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

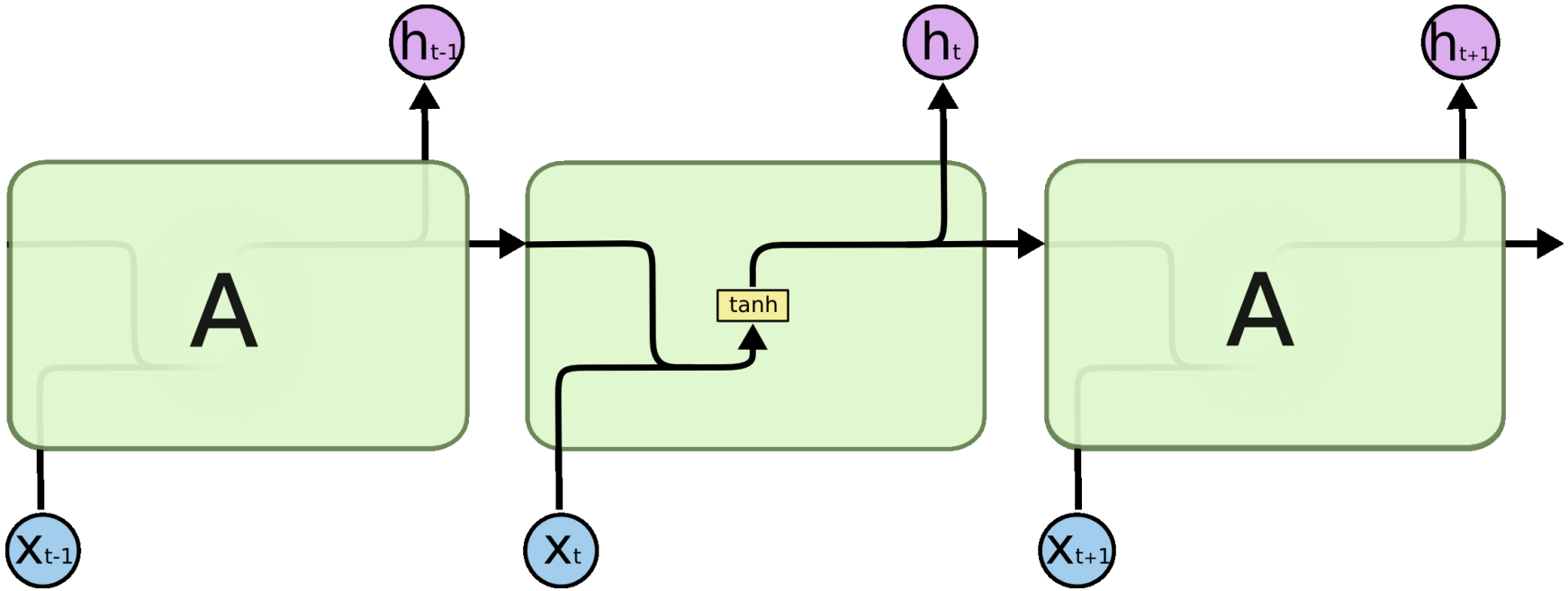






LSTM

Long Short Term Memory

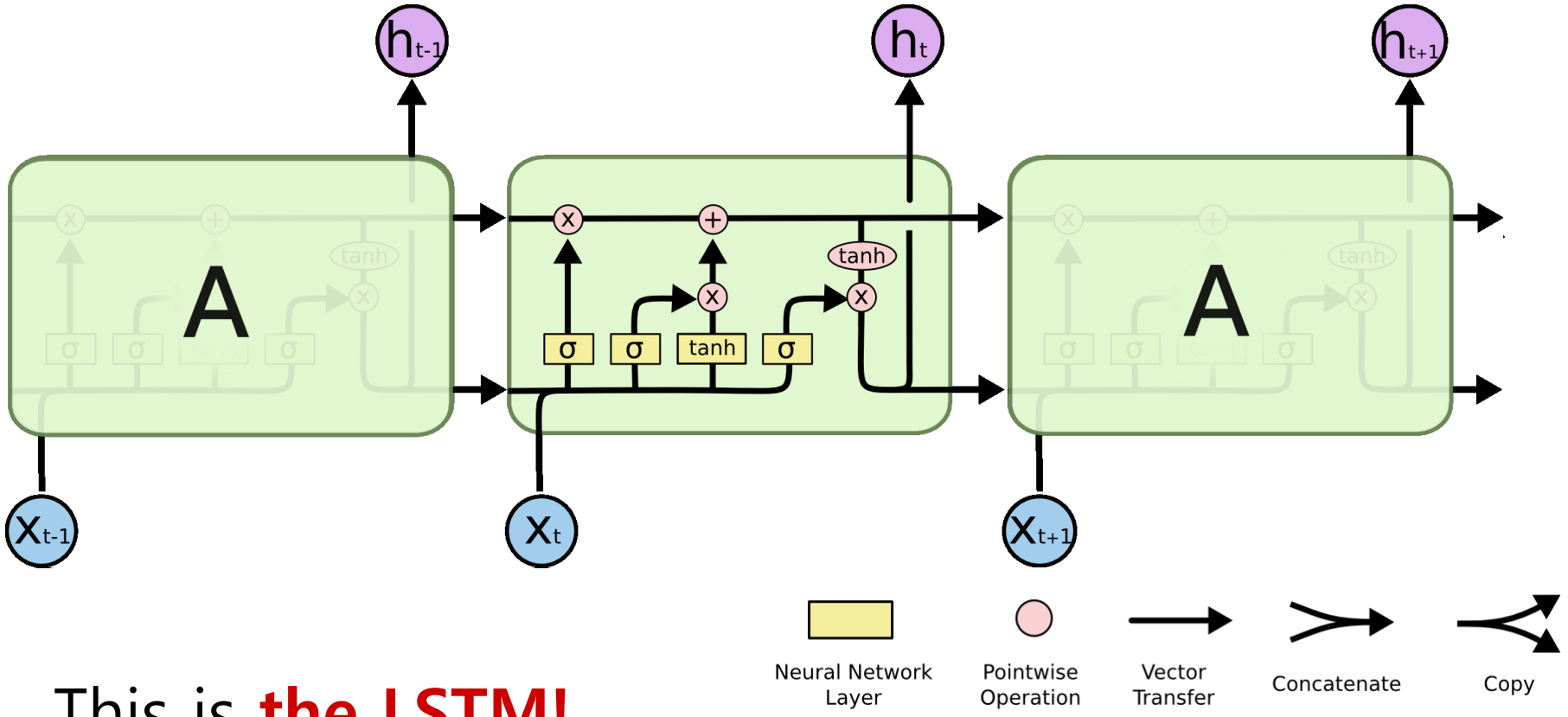


This is just a standard RNN.

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTM

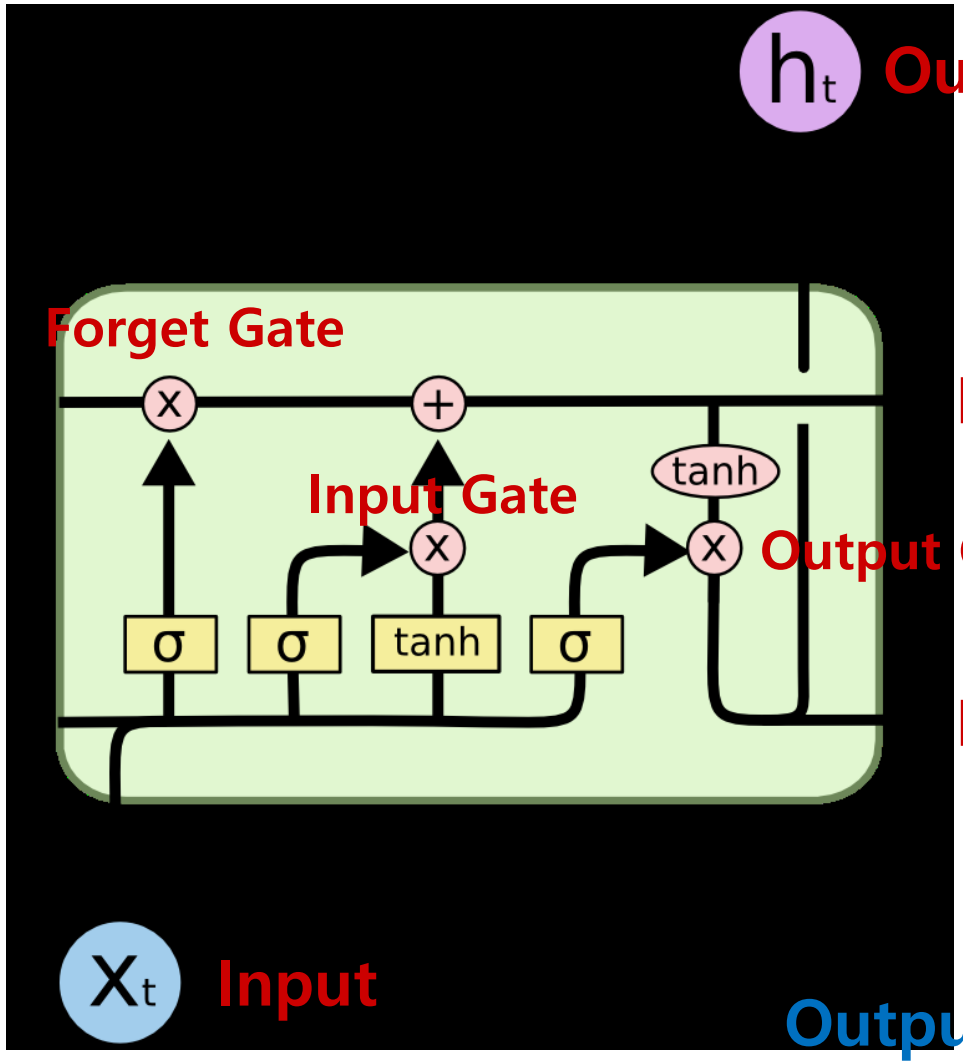
Long Short Term Memory



This is **the LSTM!**

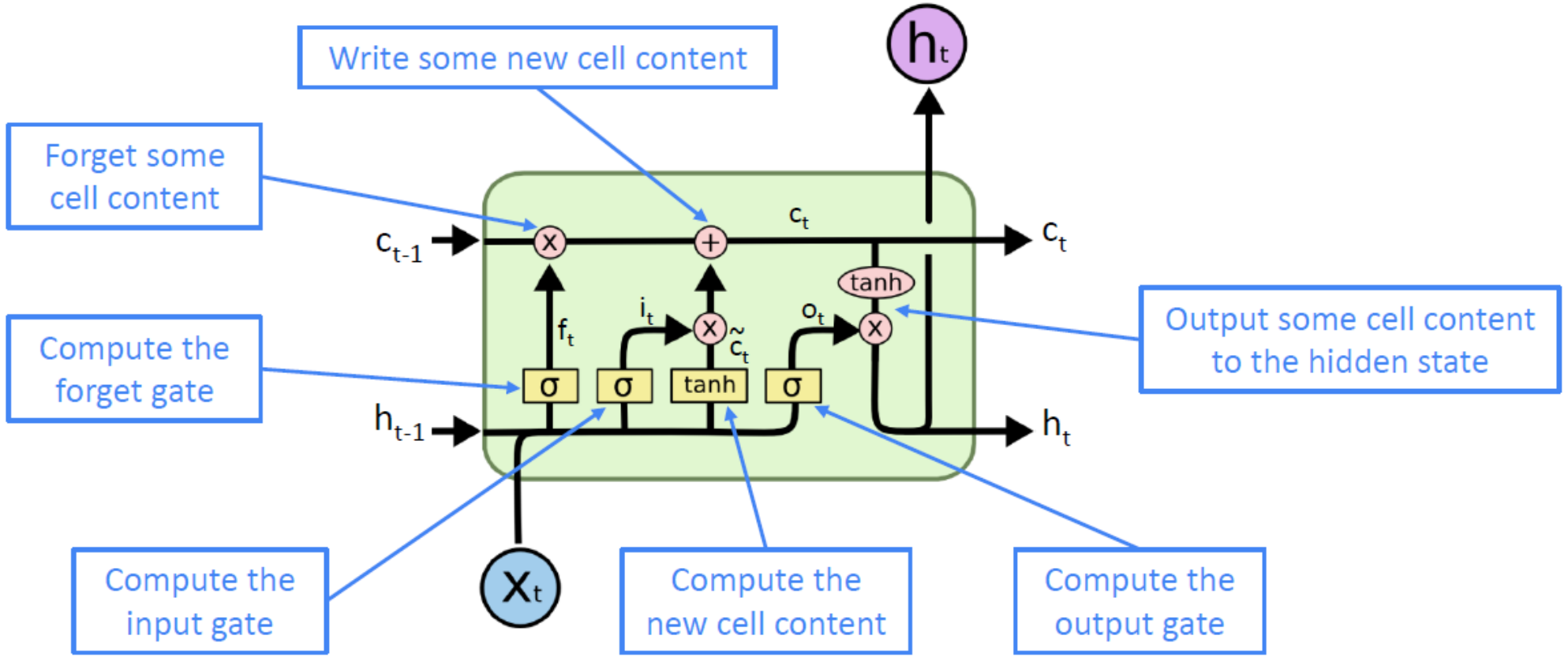
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Overall Architecture



http://colah.github.io/posts/2015-08-Understanding-LSTMs/

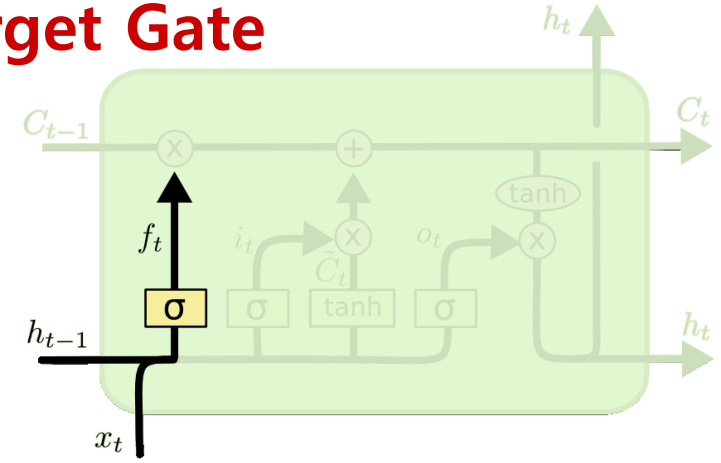
Overall Architecture



http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Forget Gate & Input Gate

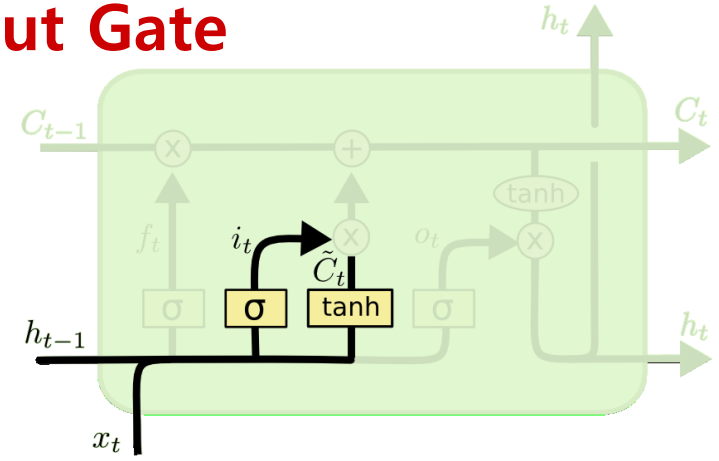
Forget Gate



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

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

Input Gate



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

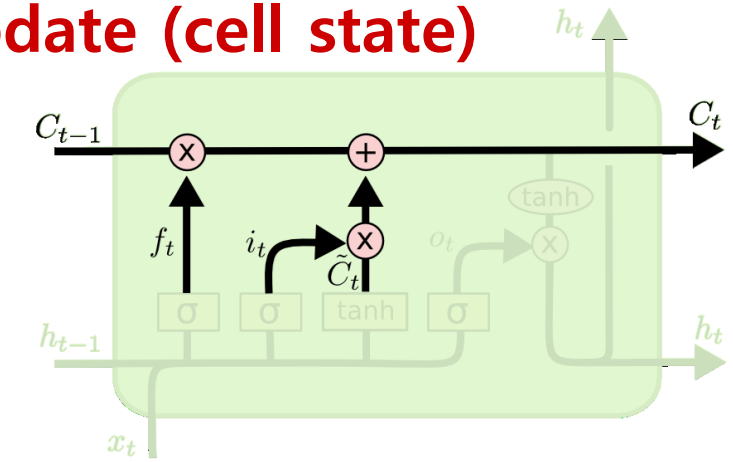
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

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

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Update Cell State & Output Gate

Update (cell state)

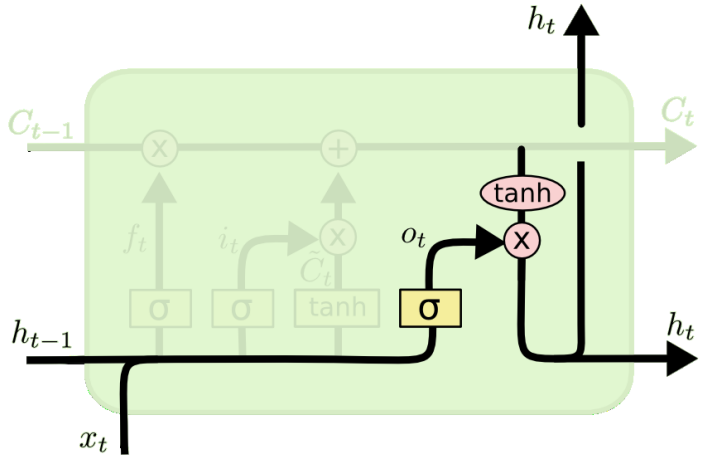


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Update, scaled by how much we decide to update

: input_gate*curr_state + forget_gate*prev_state

Output Gate (hidden state)



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

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

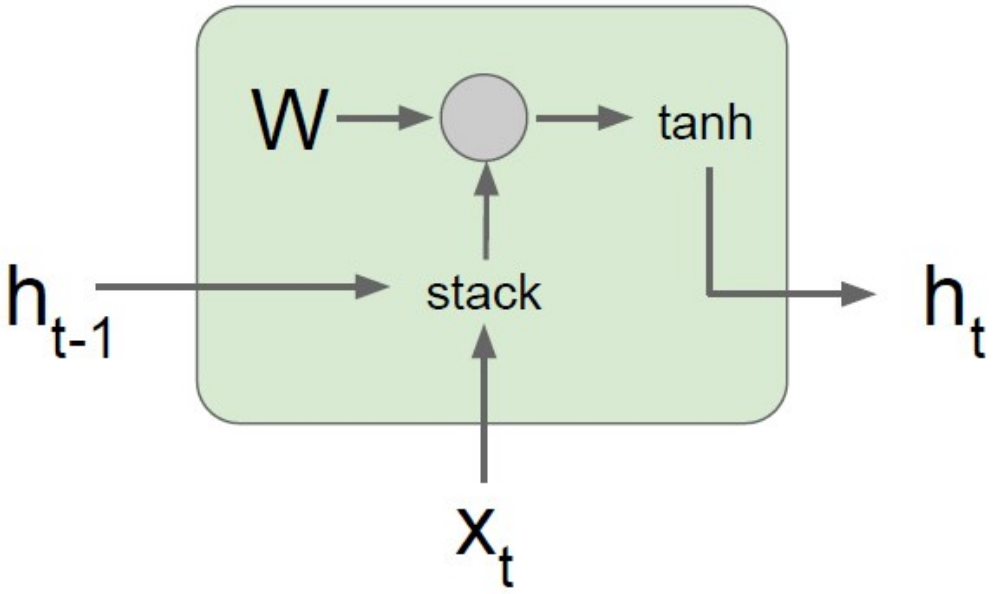
Output based on the updated state

: output_gate*updated_state

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Is LSTM free from Vanishing Gradient Problems?

Vanilla RNN Gradient Flow

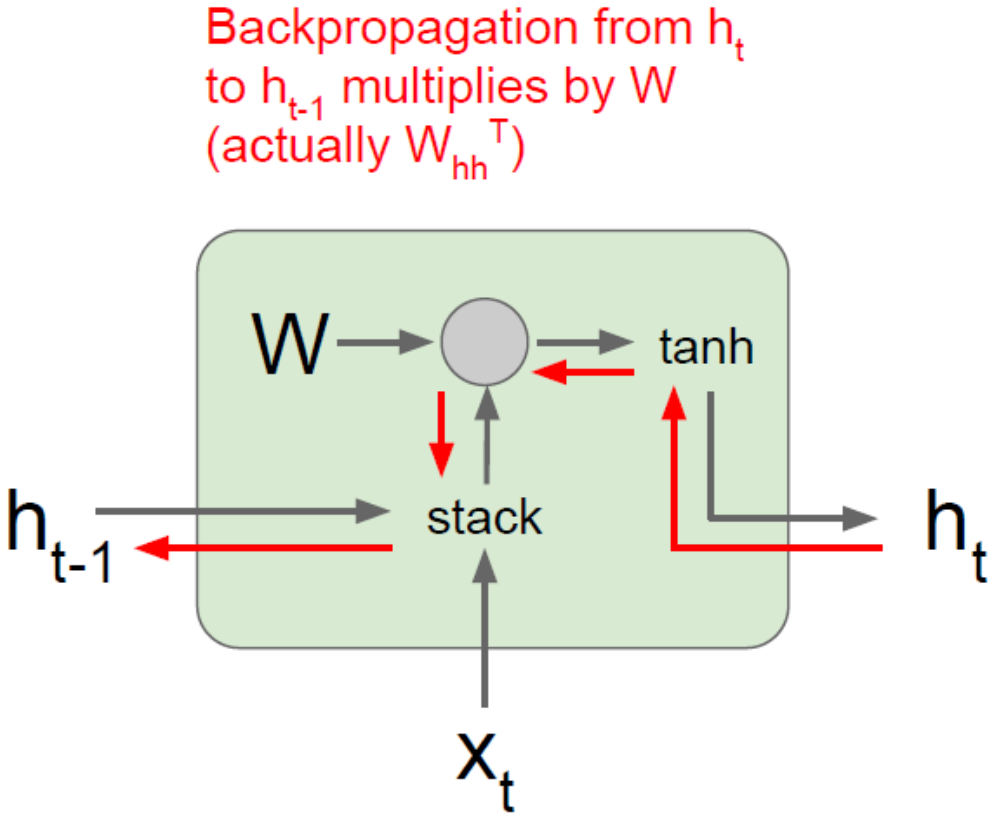


$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

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

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

Vanilla RNN Gradient Flow

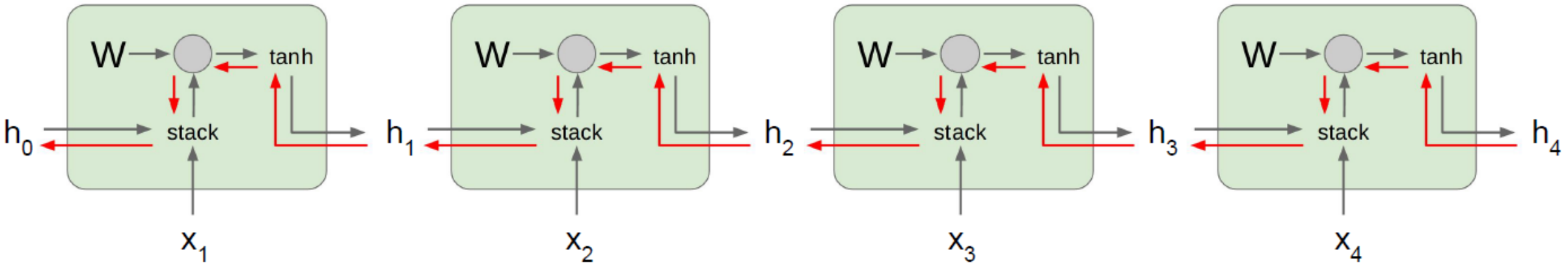


$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

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

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

Vanilla RNN Gradient Flow



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

Largest singular value > 1:
Exploding gradients

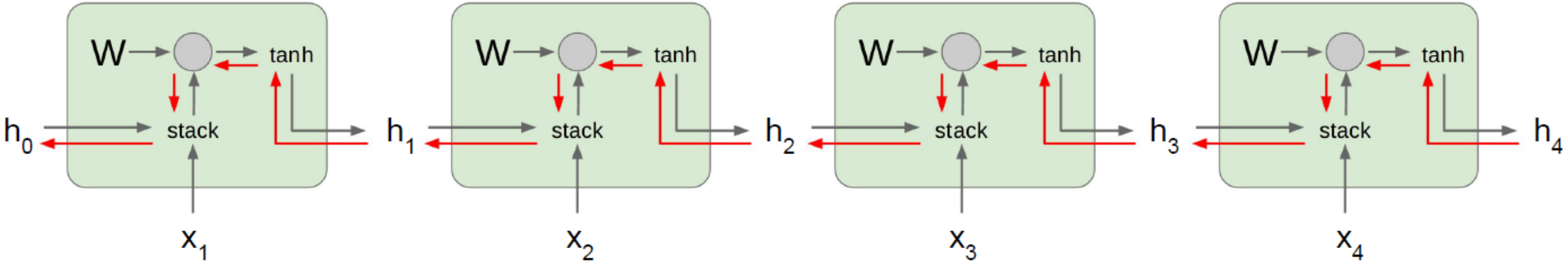
→ **Gradient clipping**: Scale gradient if its norm is too big

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

Largest singular value < 1:
Vanishing gradients

Vanilla RNN Gradient Flow

- Gradient가 흘러갈 때 같은 숫자(W, tanh의 미분)가 계속 곱해짐

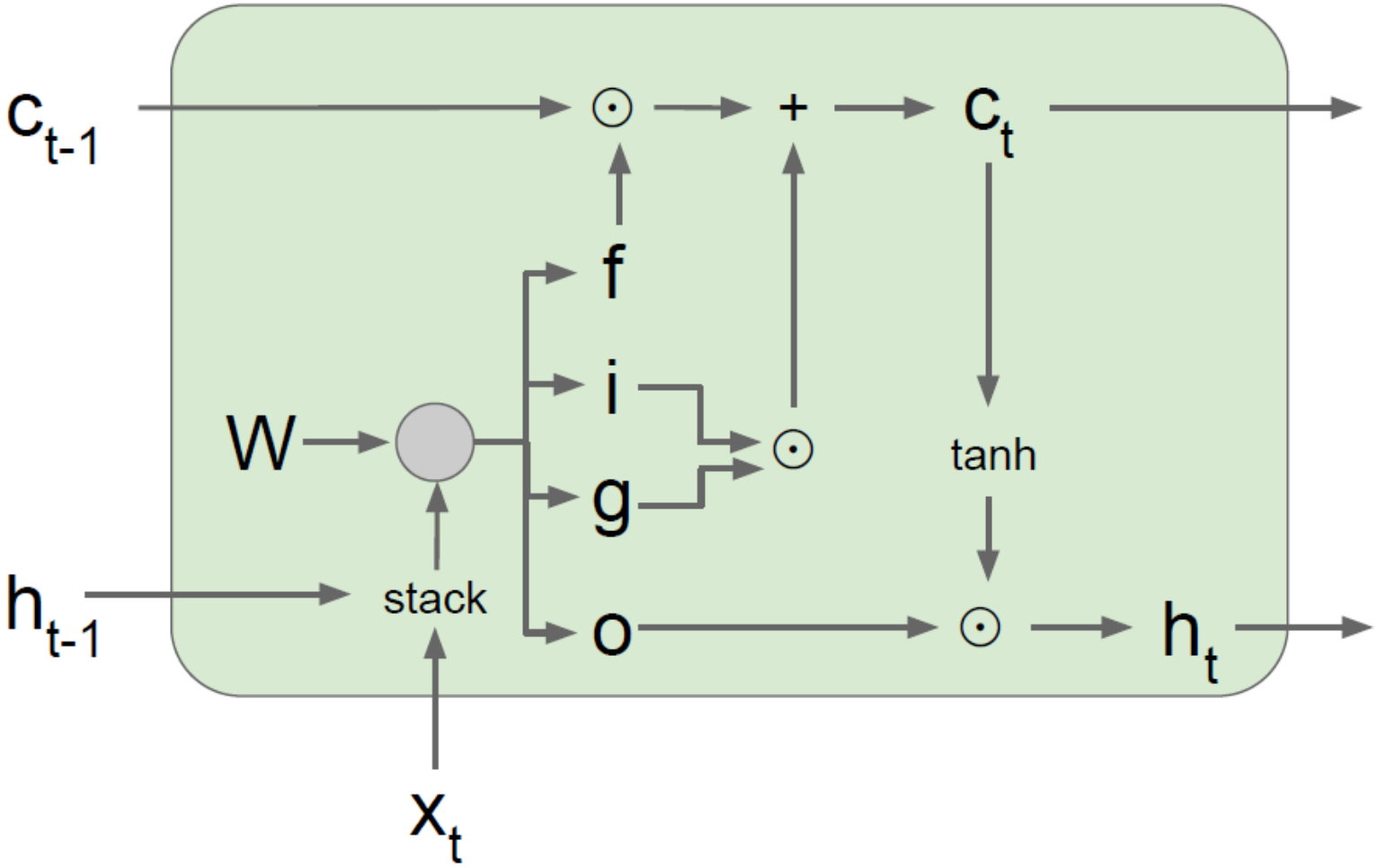


Computing gradient of h_0 involves many factors of W (and repeated tanh)

Largest singular value > 1:
Exploding gradients

Largest singular value < 1:
Vanishing gradients → Change RNN architecture

LSTM Gradient Flow



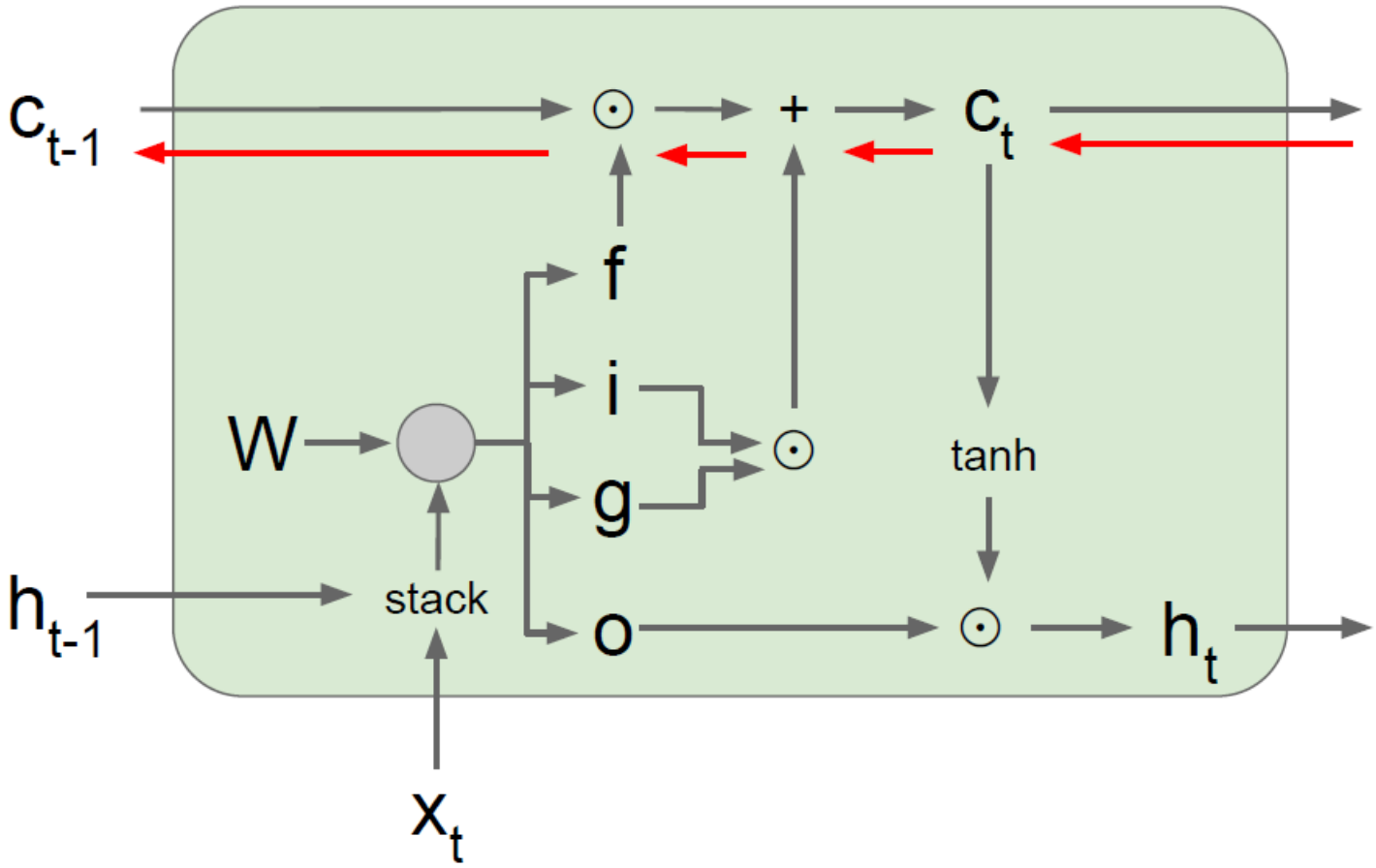
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

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

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

LSTM Gradient Flow

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



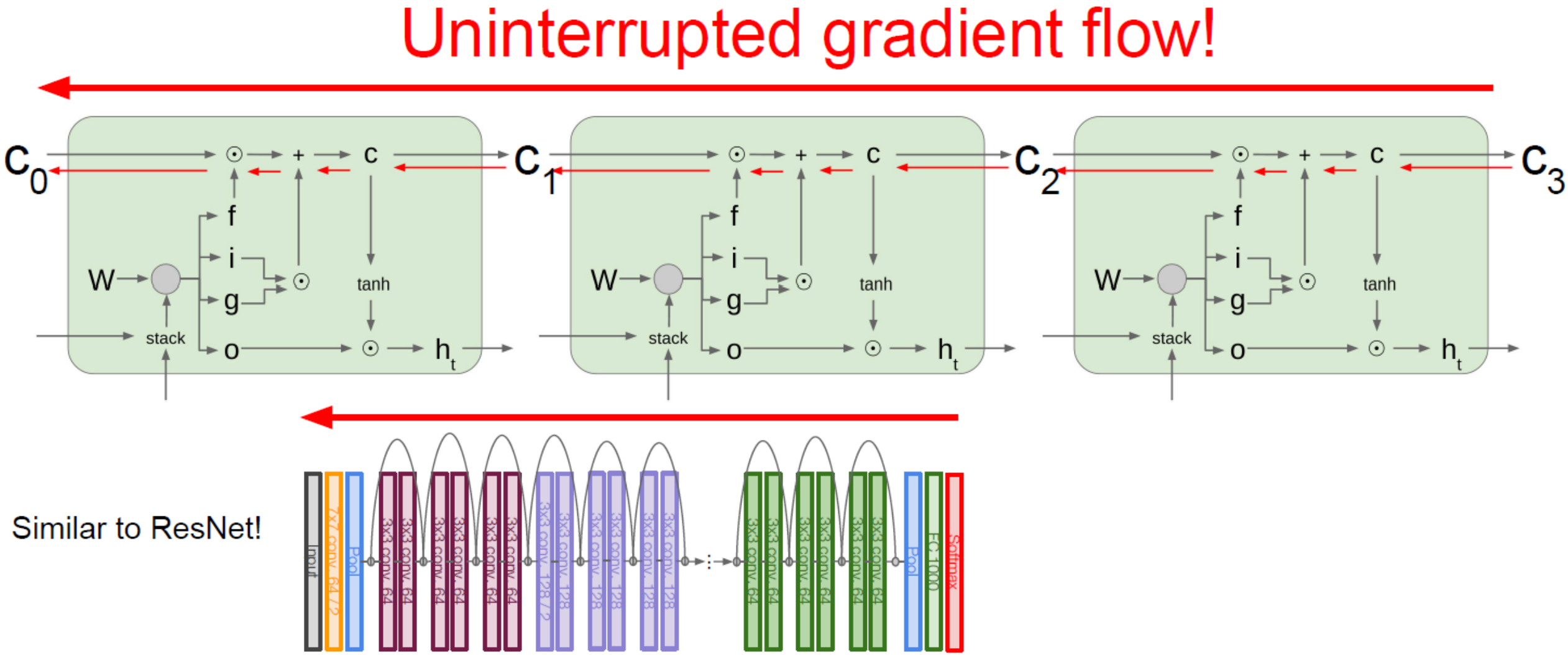
Backpropagation from c_t to c_{t-1} only elementwise multiplication by f, no matrix multiply by W

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

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

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

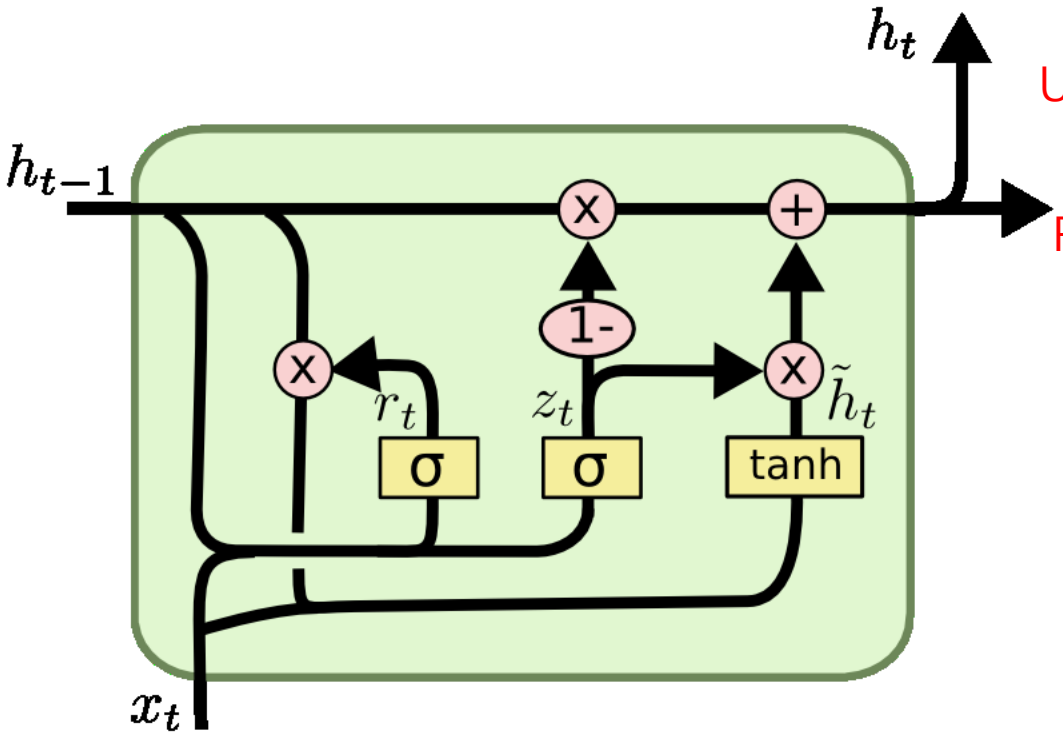
LSTM Gradient Flow



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

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

Reset gate

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

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

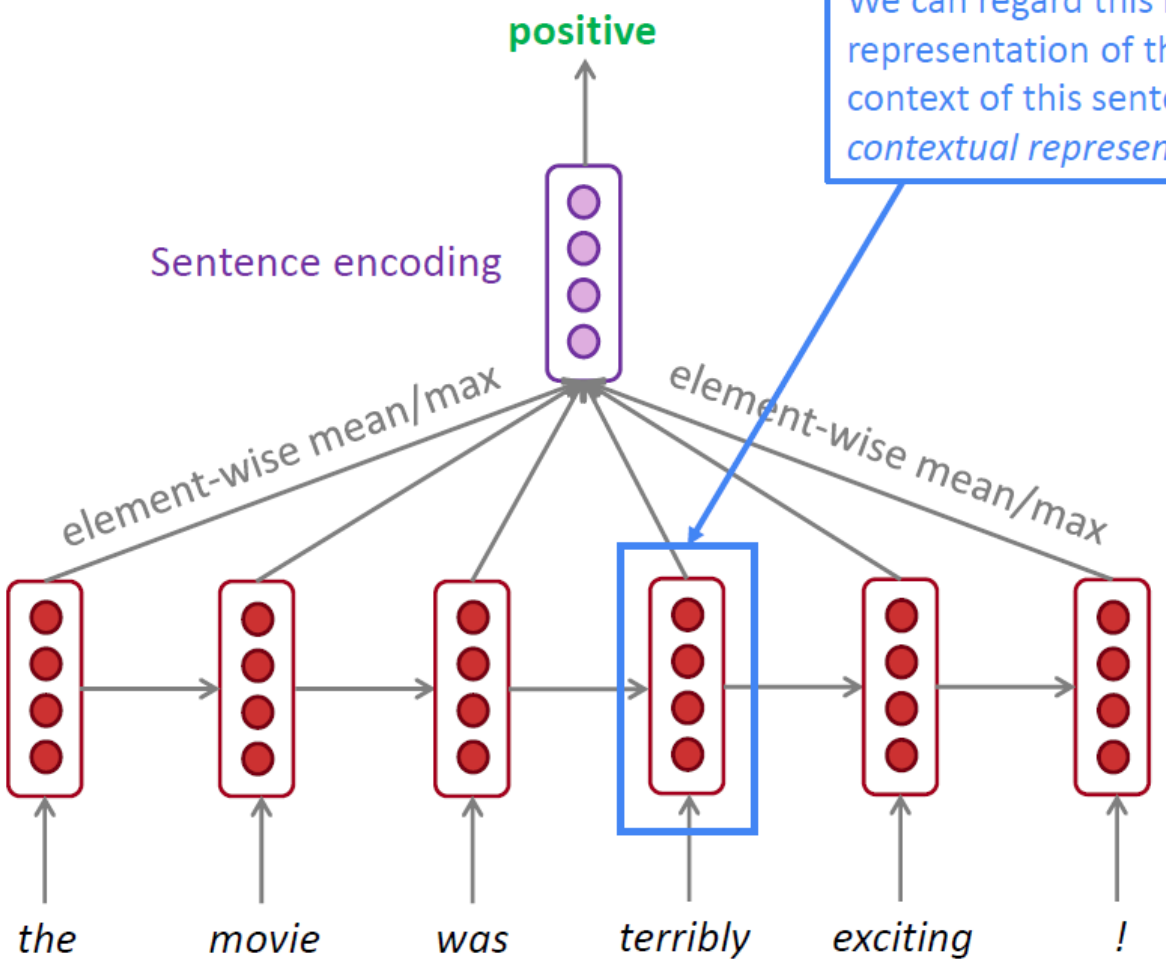
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

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



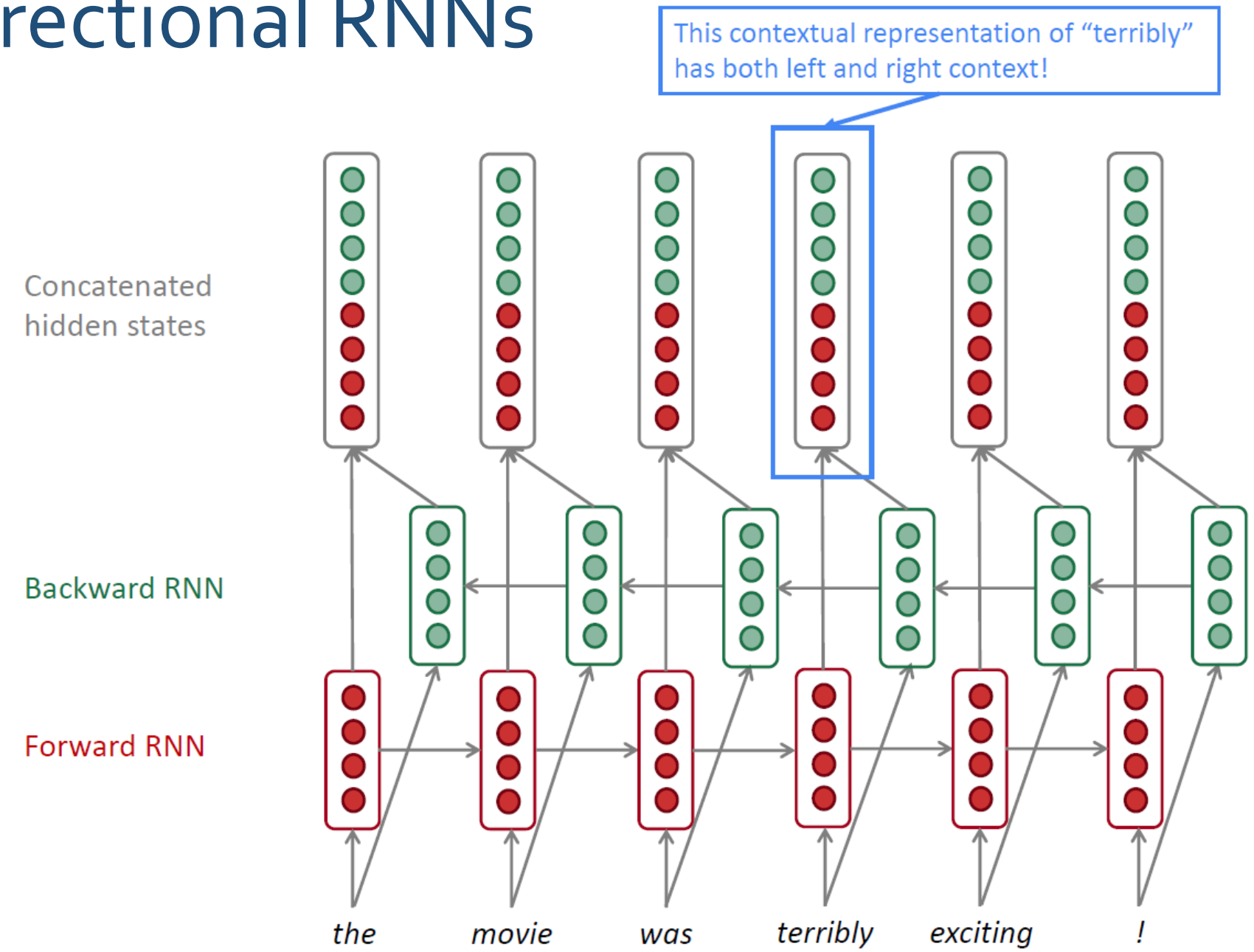
We can regard this hidden state as a representation of the word *"terribly"* in the context of this sentence. We call this a *contextual representation.*

These contextual representations only contain information about the *left* context (e.g. *"the movie was"*).

What about *right* context?

In this example, *"exciting"* is in the right context and this modifies the meaning of *"terribly"* (from negative to positive)

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.

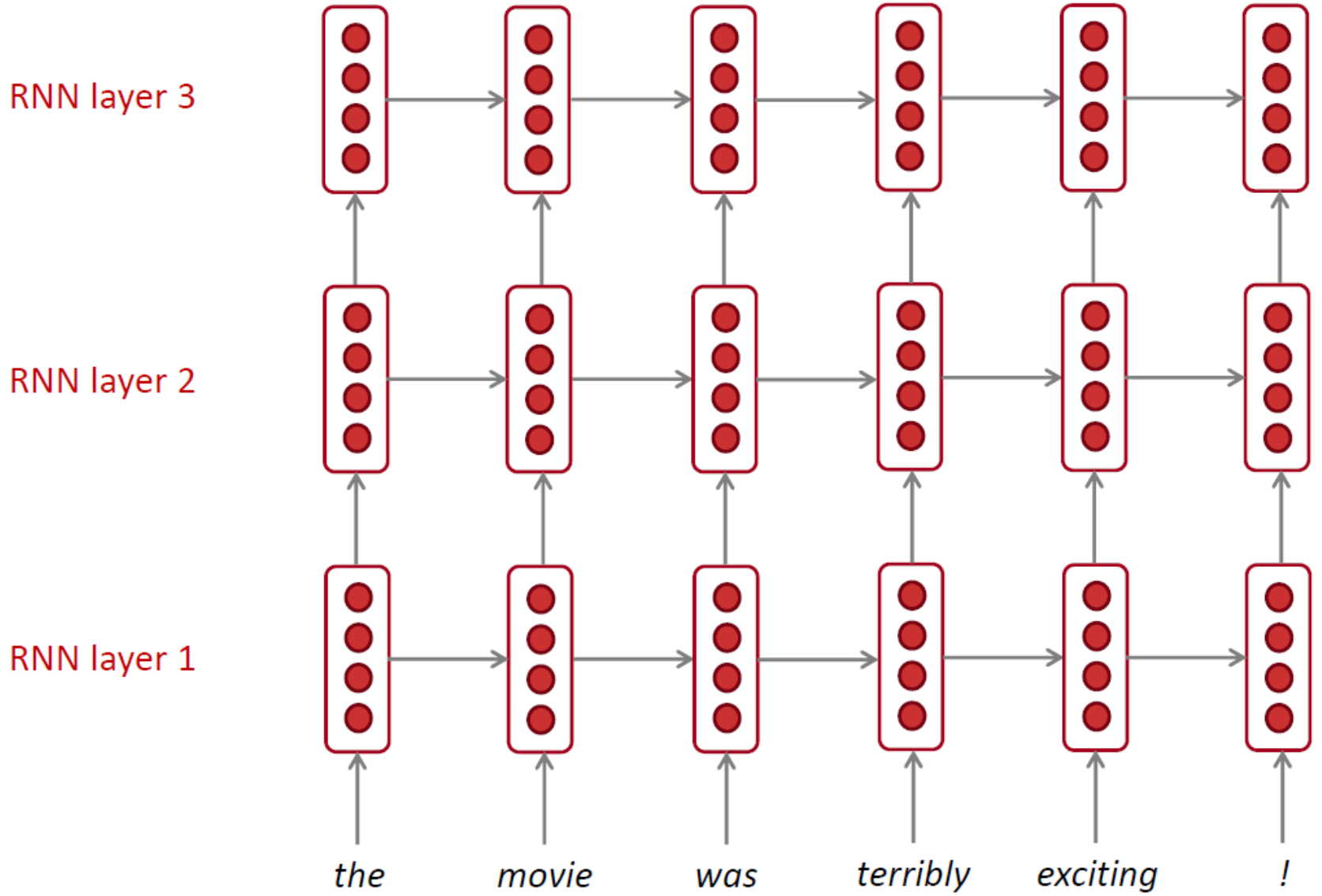
- If you do have entire input sequence (e.g. any kind of encoding), bidirectionality is powerful (you should use it by default).

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*



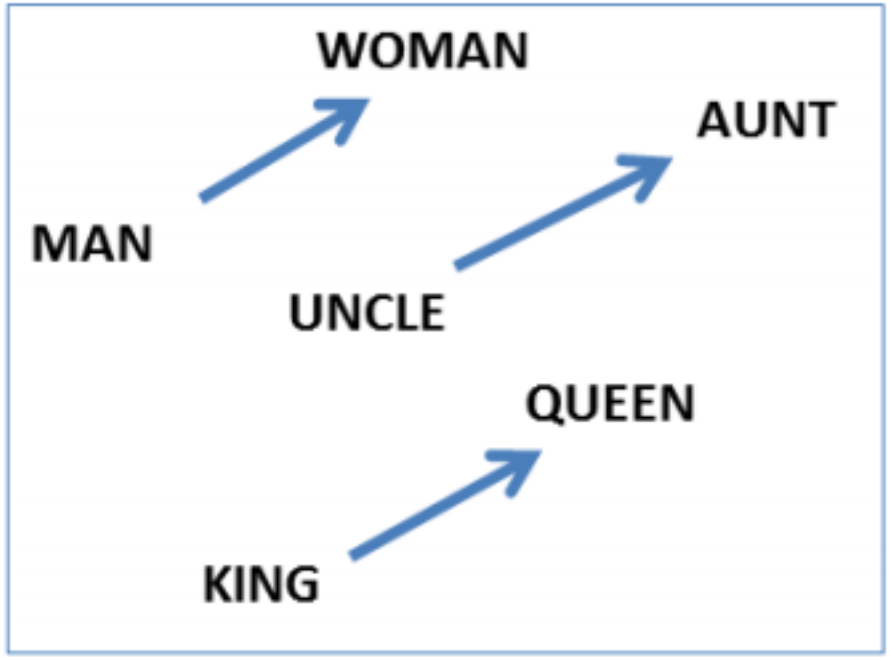
Natural Language Processing

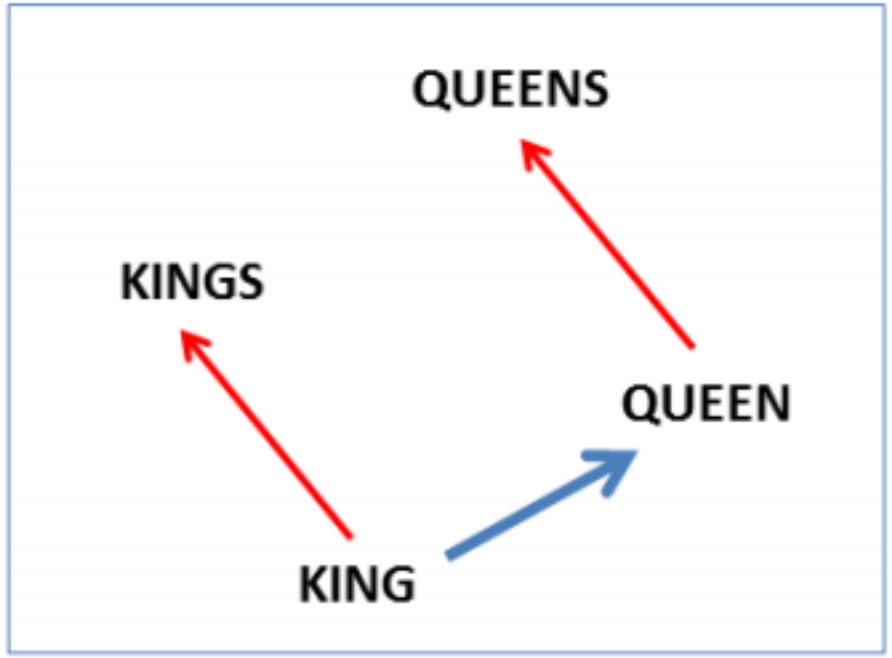
- Token Representation – Word2Vec
- Sentence Representation – CBOW, RN, CNN, Self Attention, RNN
- Language Model – N-Gram LM, NNLM
- Neural Machine Translation

How to Represent a Token

- Intuitive embedding – one hot encoding
  - Apple, Strawberry, Dog 세 단어가 있을 때,
  - Apple → [1, 0, 0]
  - Strawberry → [0, 1, 0]
  - Dog → [0, 0, 1]
- 장점
  - Easy!
- 단점
  - 단어들 간의 의미관계를 파악할 수 없음(apple과 strawberry, apple과 dog)
  - 단어가 많아지면?

We Want…



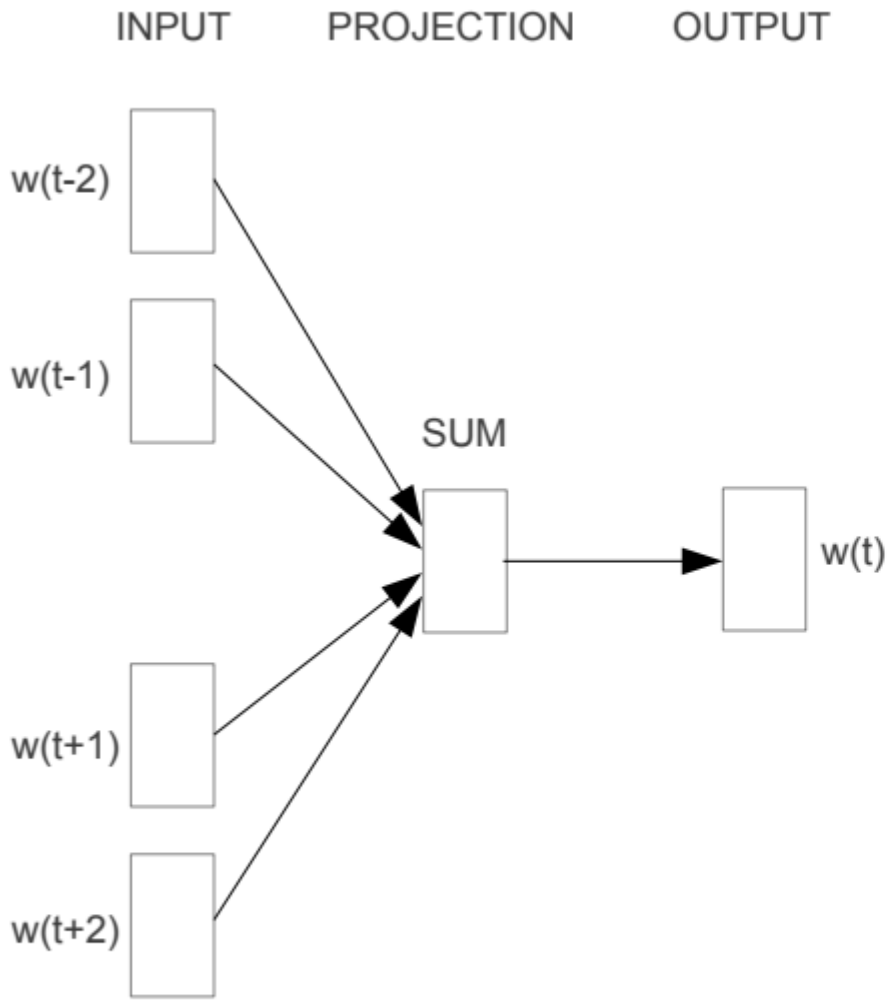


(Mikolov et al., NAACL HLT, 2013)

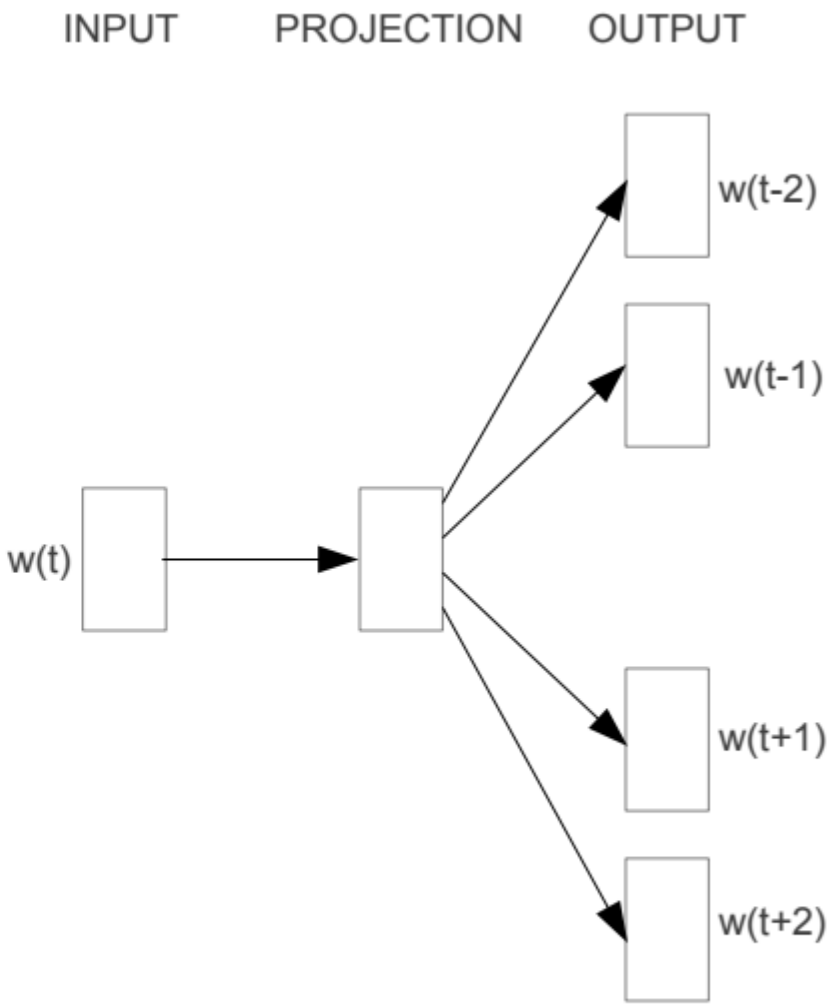
Let’s Try It

- http://w.elnn.kr/search/

CBOW & Skip-gram



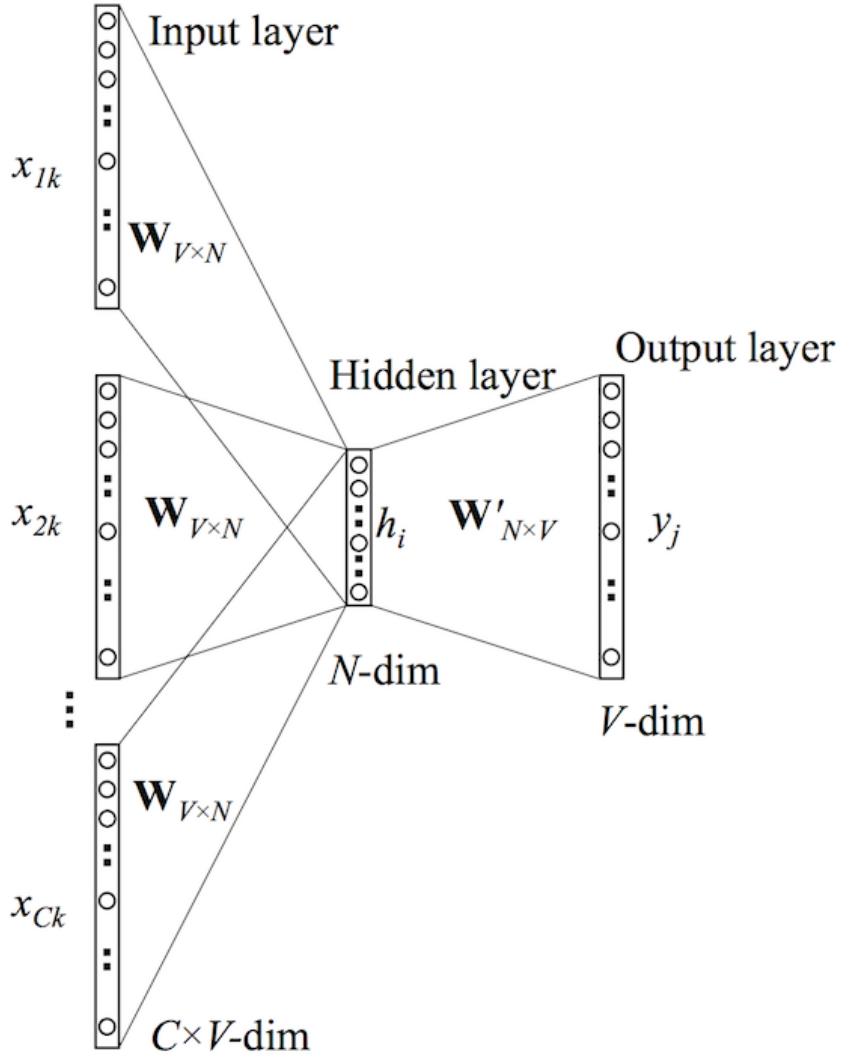
CBOW



Skip-gram

CBOW – Continuous Bag of Words

- Fill the blank
  - 아이스크림을 사 먹었는데, ___ 시려서 먹기가 힘들었다.
- 앞 뒤로 C/2개의 단어를 input으로 하여 center 단어를 맞추도록 학습
- Input은 one-hot encoding
- Input → Hidden layer는 linear mapping(avg(Wx_{ik}))
- Hidden → Output layer는 Softmax($W'h_i$)



Skip-gram

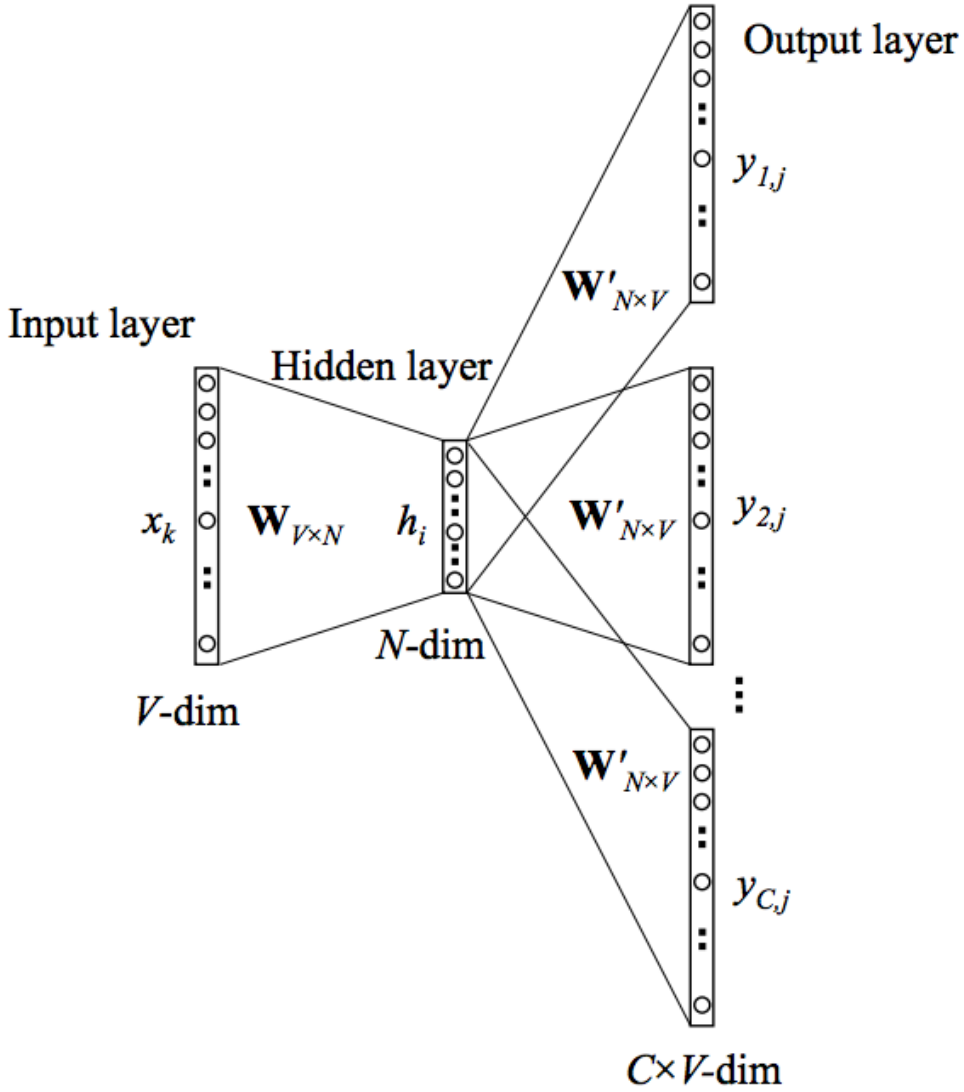
- CBOW와 반대로 중심 단어를 주고 주변 단어들에 대한 확률 값을 출력함
- Window 내에 있는 단어의 확률이 최대가 되도록 학습
- Objective function
  - Maximize

$$J'(\theta) = \prod_{t=1}^{T} \prod_{-C/2 \le j \le C/2, j \ne 0} P(x_{t+j} | x_t ; \theta)$$

→ Negative log likelihood

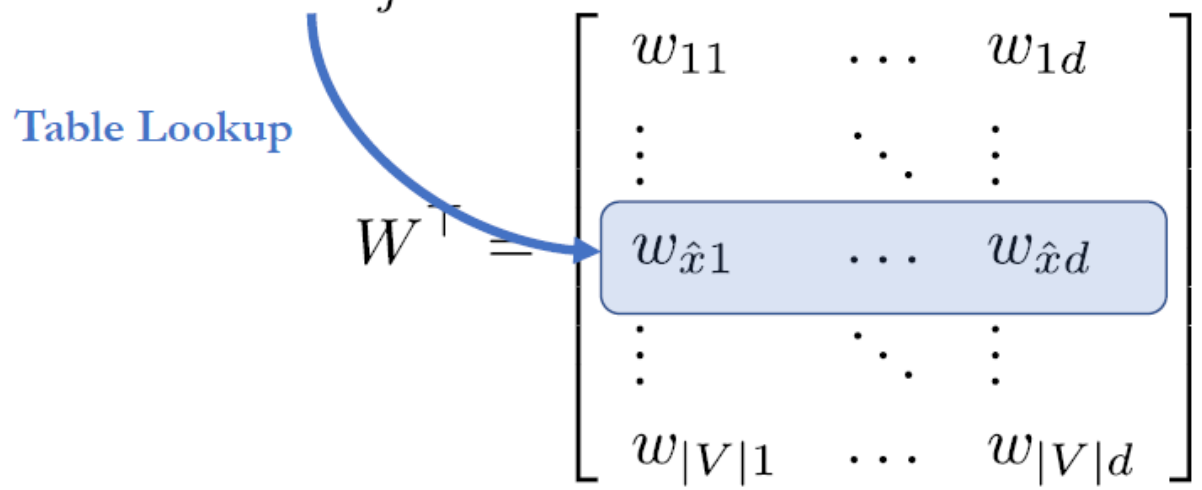
  - Minimize

$$J(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{-C/2 \le j \le C/2, j \ne 0} log P(x_{t+j} | x_t ; \theta)$$



How to Represent a Token

- How do should we represent a token so that it reflects its "meaning"?
- First, we assume nothing is known: use an one-hot encoding.
- Second, the neural network capture the token's meaning as a vector.
- This is done by a simple matrix multiplication:
  $Wx = W[\hat{x}]$, if x is one-hot,
  where $\hat{x} = \arg\max_j x_j$ is the token's index in the vocabulary.

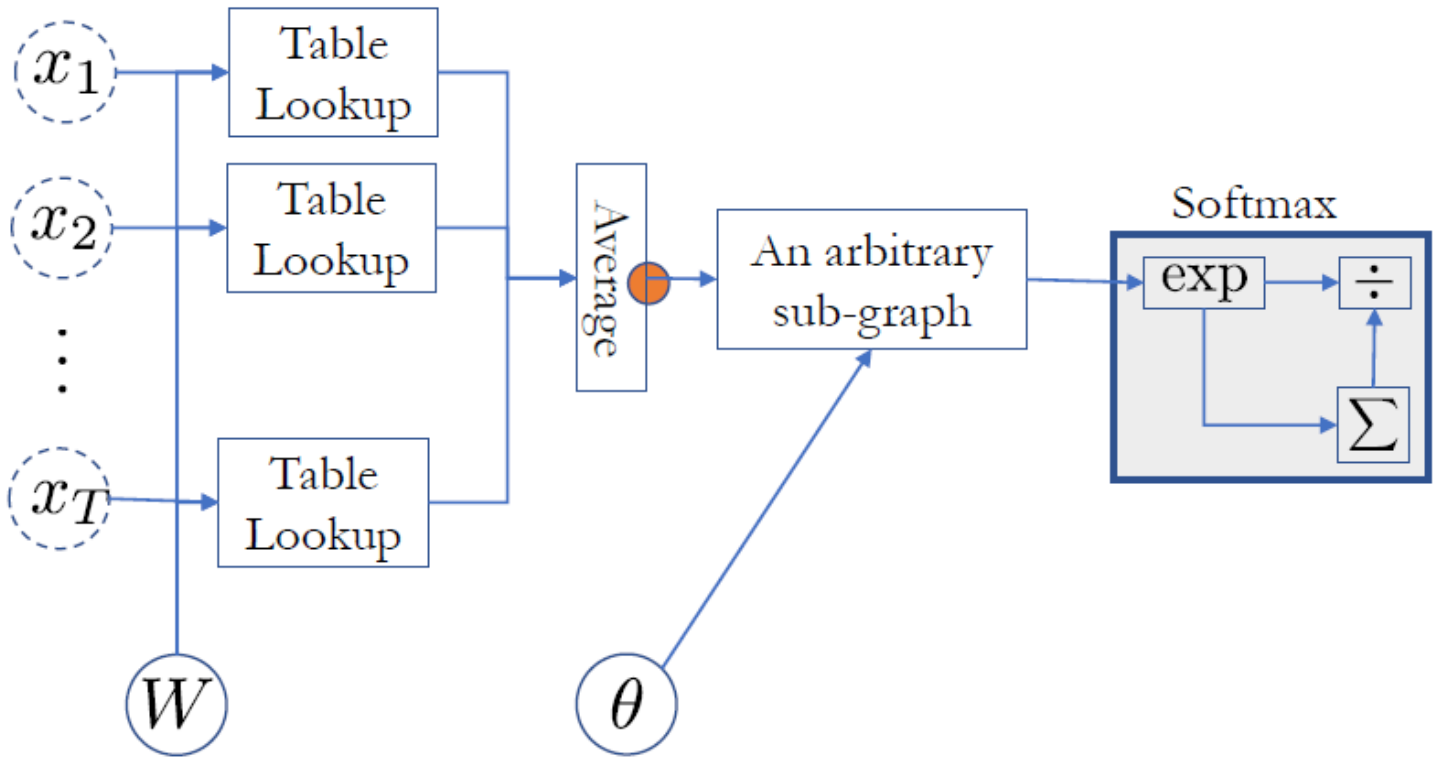


How to Represent a Sentence

- 단어는 Word2Vec을 사용하면 될텐데, 문장은 어떻게 해야할까?
- 문장 = 단어들의 sequence
- Sequence의 길이가 문장마다 모두 다름 → fixed length vector로 표현하는 방법을 찾아야 함

How to Represent a Sentence - CBOW

- Continuous bag-of-words based multi-class text classifier



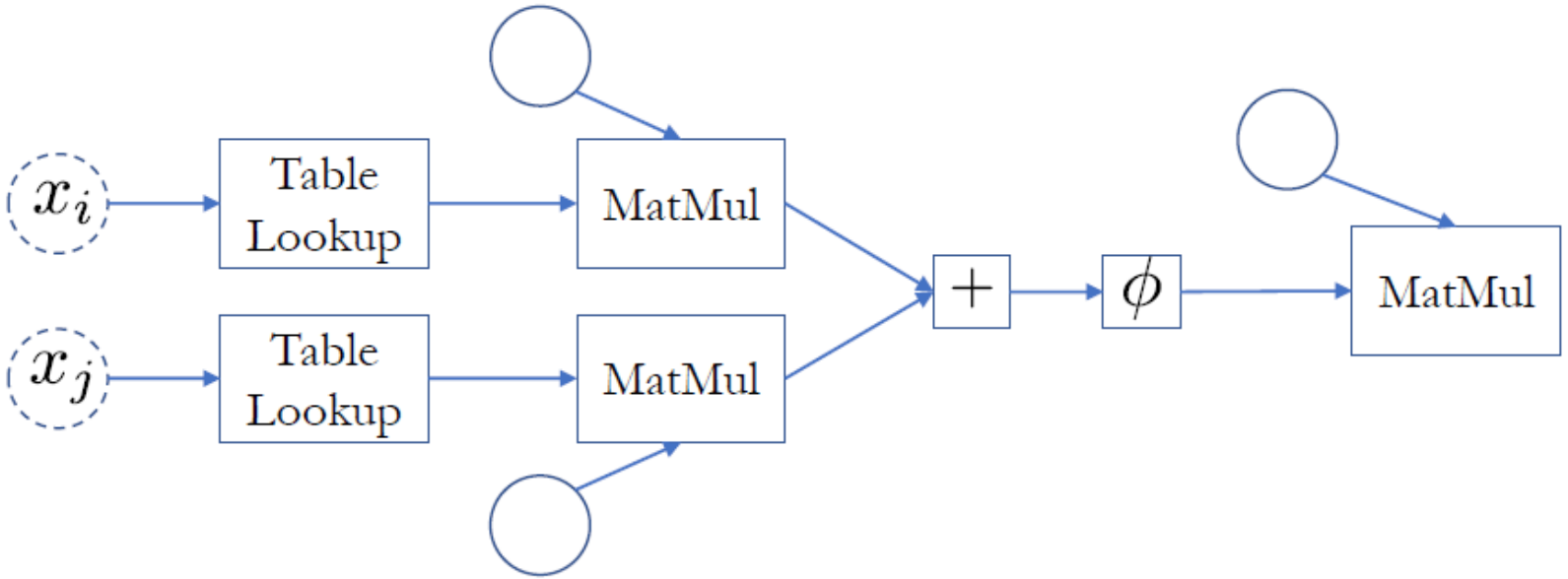
- With this DAG, you use automatic backpropagation and stochastic gradient descent to train the classifier.

How to Represent a Sentence - RN

- Relation Network [Santoro et al., 2017]: Skip Bigrams
  - Consider all possible pairs of tokens: $(x_i, x_j), \forall i \neq j$
  - Combine two token vectors with a neural network for each pair

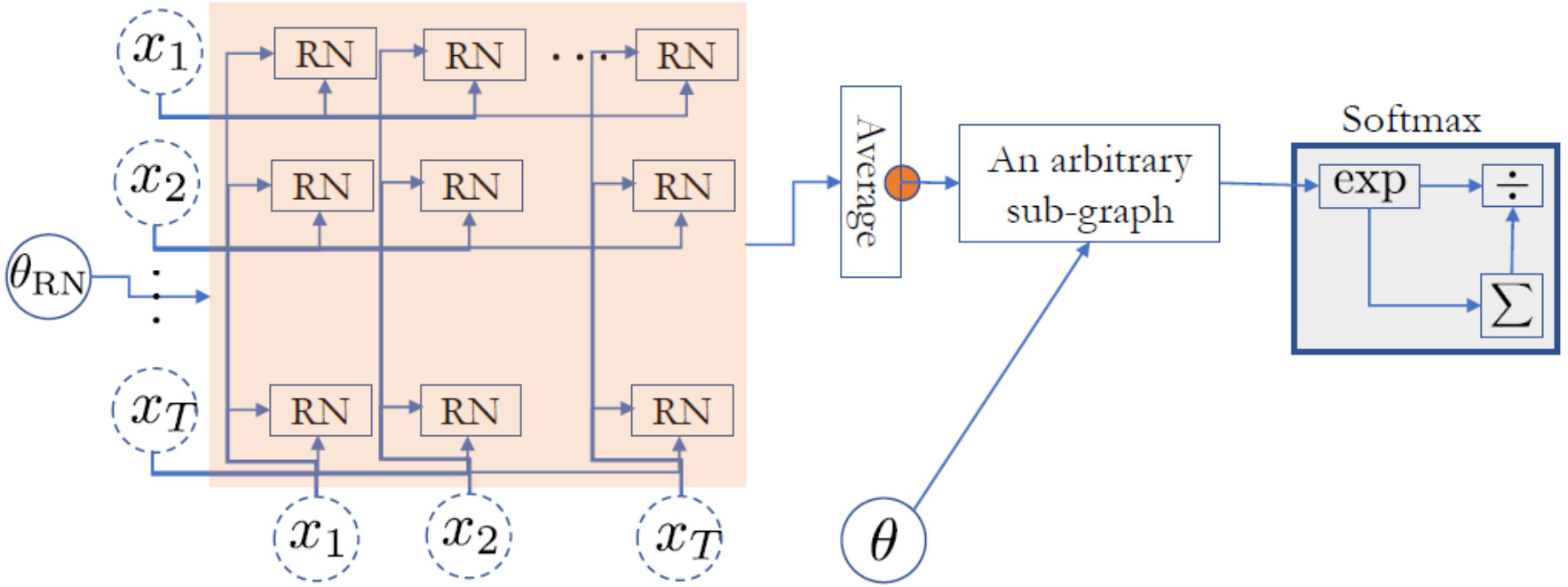
    $$f(x_i, x_j) = W\phi(U_{\text{left}}e_i + U_{\text{right}}e_j)$$

    - ϕ is a element-wise nonlinear function, such as tanh or ReLU ($\max(0, a)$)
    - One subgraph in the DAG.



How to Represent a Sentence - RN

- Relation Network: Skip Bigrams
  - Considers all possible pairs of tokens: $(x_i, x_j), \forall i \neq j$
    $f(x_i, x_j) = W\phi(U_{\text{left}} e_i + U_{\text{right}} e_j)$
  - Considers the pair-wise "relation"ship $\text{RN}(X) = \frac{1}{2N(N-1)} \sum_{i=1}^{T-1} \sum_{j=i+1}^{T} f(x_i, x_j)$
  - Averages all these relationship vectors

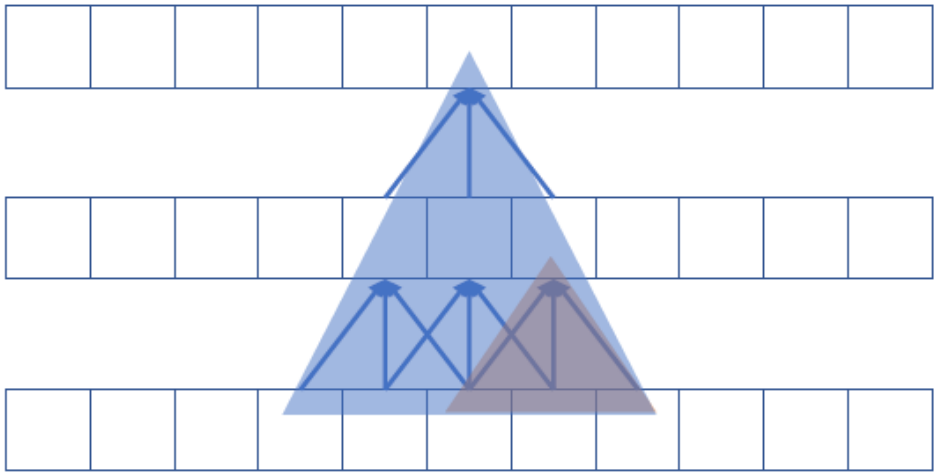


How to Represent a Sentence - CNN

- Convolutional Networks [Kim, 2014; Kalchbrenner et al., 2015]
  - Captures *k*-grams hierarchically
  - One 1-D convolutional layer: considers all *k*-grams

$$h_t = \phi\left(\sum_{\tau=-k/2}^{k/2} W_\tau e_{t+\tau}\right), \text{ resulting in } H = (h_1, h_2, \ldots, h_T).$$

  - Stack more than one convolutional layers: progressively-growing window
  - Fits our intuition of how sentence is understood: **tokens→multi-word expressions→phrases→sentence**



How to Represent a Sentence – Self Attention

- Can we combine and generalize the relation network and the CNN?
- Relation Network:
  - Each token's representation is computed against all the other tokens

$$h_t = f(x_t, x_1) + \cdots + f(x_t, x_{t-1}) + f(x_t, x_{t+1}) + \cdots + f(x_t, x_T)$$

- CNN:
  - Each token's representation is computed against neighbouring tokens

$$h_t = f(x_t, x_{t-k}) + \cdots + f(x_t, x_t) + \cdots + f(x_t, x_{t+k})$$

- RN considers the entire sentence vs. CNN focuses on the local context.

How to Represent a Sentence – Self Attention

- Can we combine and generalize the relation network and the CNN?
- CNN as a weighted relation network:
  - Original: $h_t = f(x_t, x_{t-k}) + \cdots + f(x_t, x_t) + \cdots + f(x_t, x_{t+k})$
  - Weighted:

$$h_t = \sum_{t'=1}^{T} \mathbb{I}(|t' - t| \leq k) f(x_t, x_{t'})$$

where $\mathbb{I}(S) = 1$, if S is true, and 0, otherwise .

- Can we compute those weights instead of fixing them to 0 or 1?

How to Represent a Sentence – Self Attention

- Can we compute those weights instead of fixing them to 0 or 1?
- That is, compute the weight of each pair $(x_t, x_{t'})$

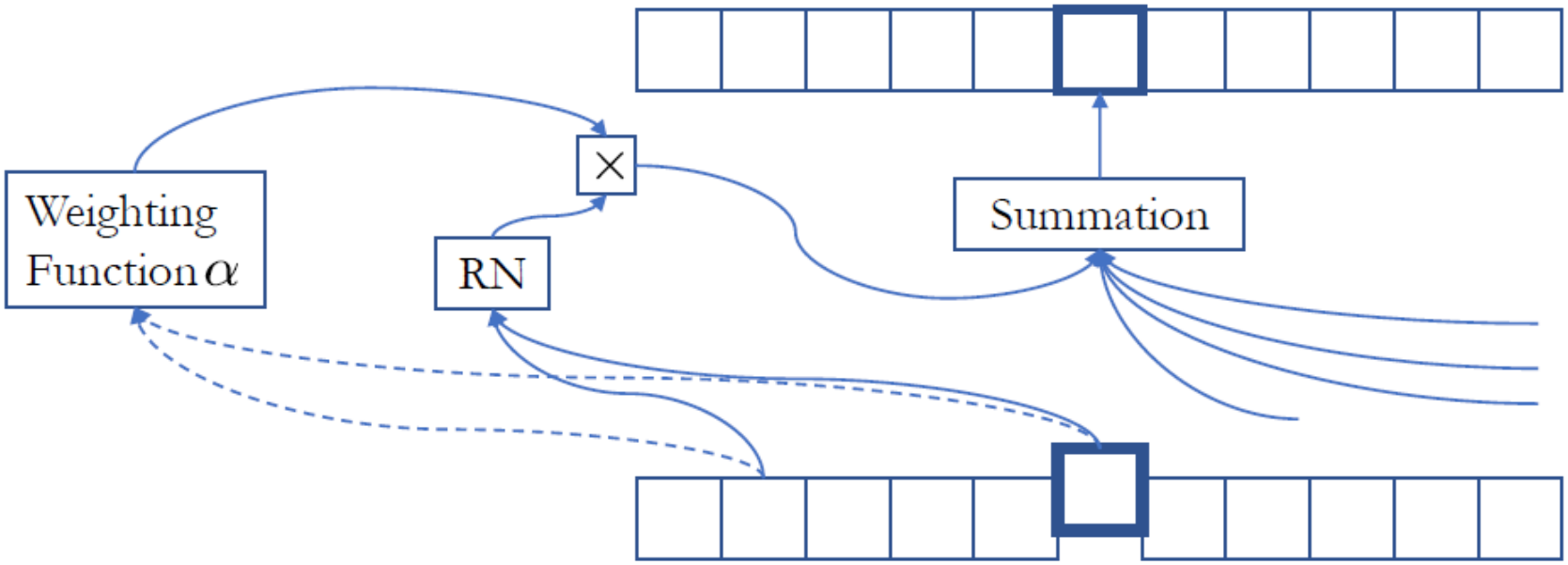
$$h_t = \sum_{t'=1}^{T} \alpha(x_t, x_{t'}) f(x_t, x_{t'})$$

- The weighting function could be yet another neural network
  - Just another subgraph in a DAG: easy to use!
    $\alpha(x_t, x_{t'}) = \sigma(\mathrm{RN}(x_t, x_{t'})) \in [0, 1]$
  - Perhaps we want to normalize them so that the weights sum to one

$$\alpha(x_t, x_{t'}) = \frac{\exp(\beta(x_t, x_{t'}))}{\sum_{t''=1}^{T} \exp(\beta(x_t, x_{t''}))}, \text{where } \beta(x_t, x_{t'}) = \mathrm{RN}(x_t, x_{t'}))$$

How to Represent a Sentence – Self Attention

- Self-Attention: a generalization of CNN and RN.
- Able to capture long-range dependencies within a single layer.
- Able to ignore irrelevant long-range dependencies.

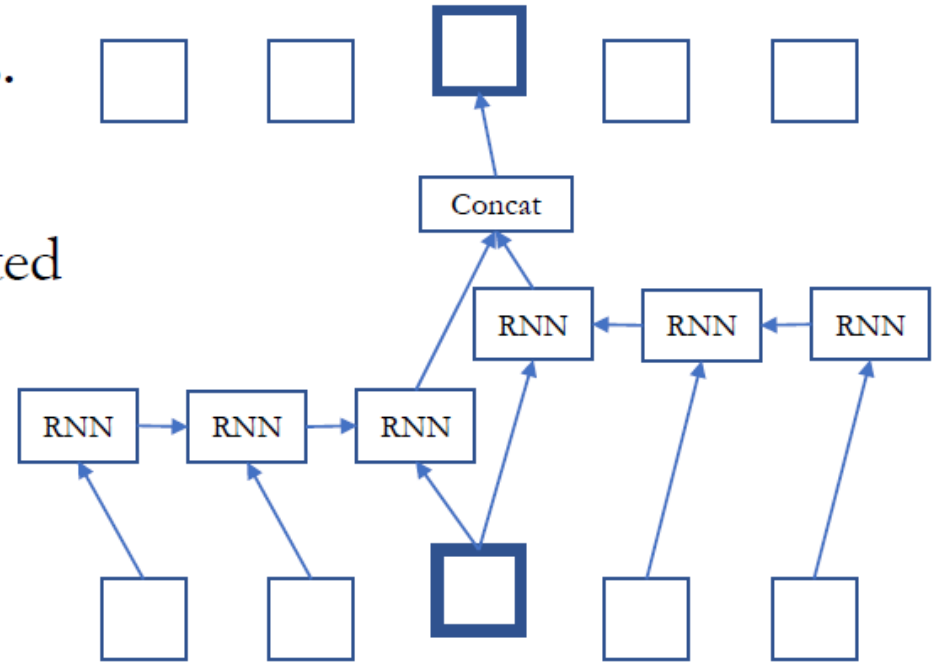


How to Represent a Sentence – RNN

- Recurrent neural network: online compression of a sequence $O(T)$

  $h_t = \text{RNN}(h_{t-1}, x_t)$, where $h_0 = 0$.

- Bidirectional RNN to account for both sides.
- Inherently sequential processing
  - Less desirable for modern, parallelized, distributed computing infrastructure.
- LSTM [Hochreiter&Schmidhuber, 1999] and GRU [Cho et al., 2014] have become de facto standard
  - All standard frameworks implement them.
  - Efficient GPU kernels are available.



Language Model

- Input: a sentence
- Output: the probability of the input sentence
- A language model captures the distribution over all possible sentences.
  $p(X) = p((x_1, x_2, \ldots, x_T))$
- Unlike text classification, it is *unsupervised learning*.
  - We will however turn the problem into a *sequence of supervised learning*.

Autoregressive Language Model

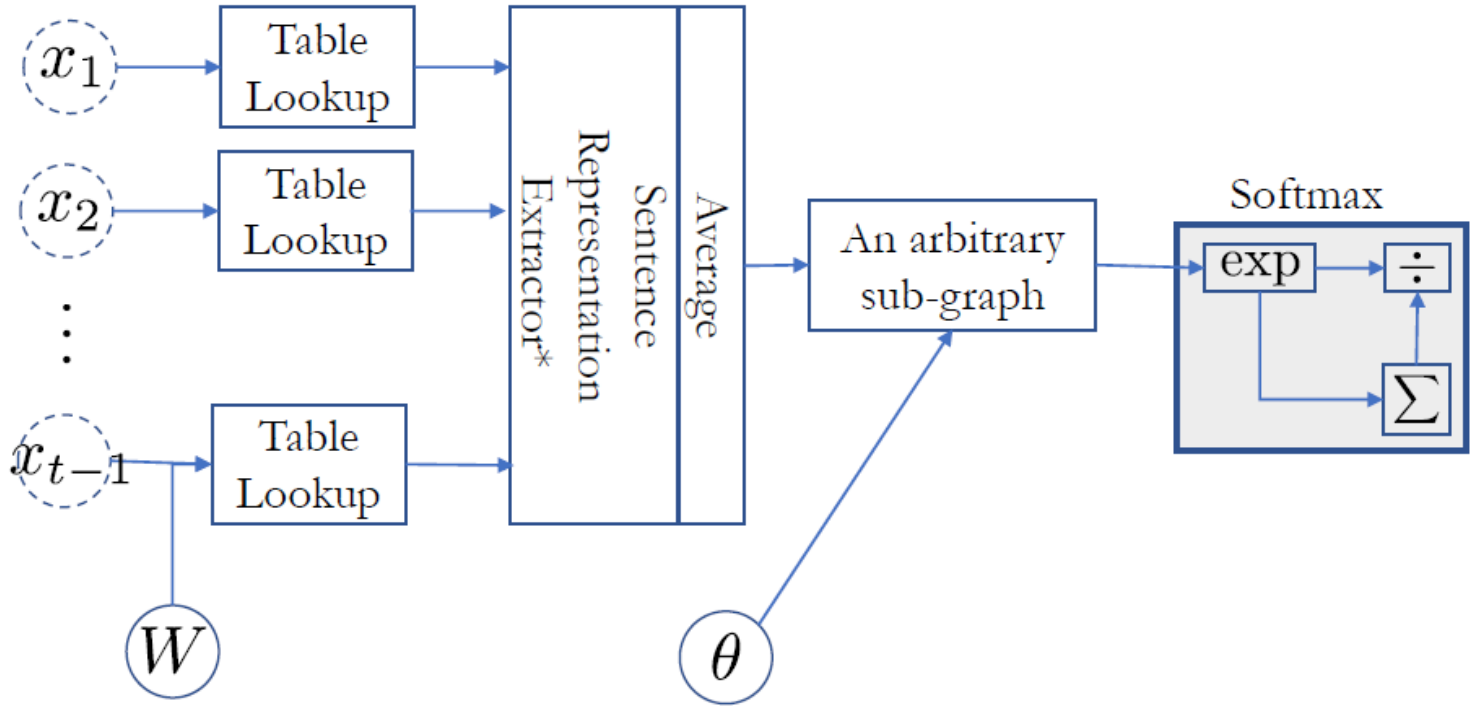
- Autoregressive sequence modelling
  - The distribution over the next token is based on all the previous tokens.
    $p(X) = p(x_1)p(x_2|x_1) \cdots p(x_T|x_1, \ldots, x_{T-1})$
  - This equality holds exactly due to the def. of conditional distribution*
- Unsupervised learning becomes a set of supervised problems.
  - Each conditional is a neural network classifier.
  - Input is all the previous tokens (a partial sentence).
  - Output is the distribution over all possible next tokens (classes).
  - It is a **text classification** problem.

Autoregressive Language Model

- Autoregressive sequence modelling
  - The distribution over the next token is based on all the previous tokens.

$$p(X) = p(x_1)p(x_2|x_1)\cdots p(x_T|x_1,\ldots,x_{T-1})$$

- Each conditional is a sentence classifier:



N-Gram Language Model

- Let's back up a little…
- What would we do *without* a neural network?
- We need to estimate *n*-gram probabilities: $p(x|x_{-N}, x_{-N+1}, \ldots, x_{-1})$
- Recall the def. of conditional and marginal probabilities:

$$p(x|x_{-N}, x_{-N+1}, \ldots, x_{-1}) = \frac{p(x_{-N}, x_{-N+1}, \ldots, x_{-1}, x)}{p(x_{-N}, x_{-N+1}, \ldots, x_{-1})}$$

$$= \frac{p(x_{-N}, x_{-N+1}, \ldots, x_{-1}, x)}{\sum_{x \in V} p(x_{-N}, x_{-N+1}, \ldots, x_{-1}, x)}$$

  - V : all possible tokens (=vocabulary)

N-Gram Language Model

- We need to estimate *n*-gram probabilities:

$$p(x|x_{-N}, x_{-N+1}, \ldots, x_{-1}) = \frac{p(x_{-N}, x_{-N+1}, \ldots, x_{-1}, x)}{p(x_{-N}, x_{-N+1}, \ldots, x_{-1})}$$

- Estimation:

$$p(x|x_{-N}, x_{-N+1}, \ldots, x_{-1}) = \frac{p(x_{-N}, x_{-N+1}, \ldots, x_{-1}, x)}{\sum_{x \in V} p(x_{-N}, x_{-N+1}, \ldots, x_{-1}, x)}$$

$$\approx \frac{c(x_{-N}, x_{-N+1}, \ldots, x_{-1}, x)}{\sum_{x' \in V} c(x_{-N}, x_{-N+1}, \ldots, x_{-1}, x')}$$

- *Do you see why this makes sense?*

N-Gram Language Model

- We need to estimate n-gram probabilities:

$$p(x|x_{-N}, x_{-N+1}, \ldots, x_{-1}) = \frac{p(x_{-N}, x_{-N+1}, \ldots, x_{-1}, x)}{\sum_{x \in V} p(x_{-N}, x_{-N+1}, \ldots, x_{-1}, x)}$$

$$\approx \frac{c(x_{-N}, x_{-N+1}, \ldots, x_{-1}, x)}{\sum_{x' \in V} c(x_{-N}, x_{-N+1}, \ldots, x_{-1}, x')}$$

- How likely is "University" given "New York"?
  - Count all "New York University"
  - Count all "New York ?": e.g., "New York State", "New York City", "New York Fire", "New York Police", "New York Bridges", …
  - How often "New York University" happens among these?

N-Gram Language Model – Two Problems

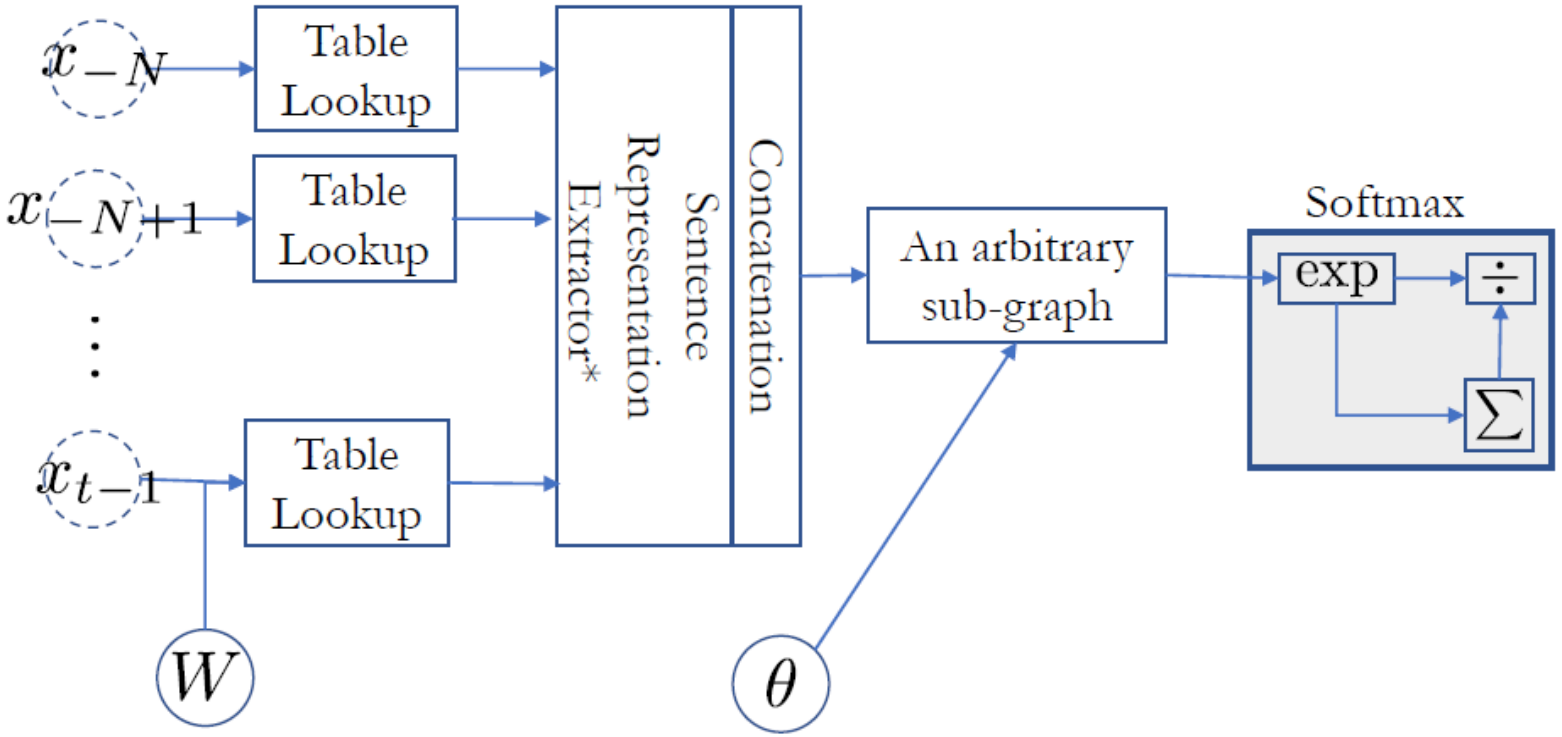
1. Data sparsity: lack of generalization
   - What happens "one" n-gram never happens?

$$p(\text{a lion is chasing a llama}) = p(\text{a}) \times p(\text{lion}|\text{a}) \times p(\text{is}|\text{a lion}) \times p(\text{chasing}|\text{lion is}) \times p(\text{a}|\text{is chasing}) \times \underbrace{p(\text{llama}|\text{chasing a})}_{=0} = 0$$

2. Inability to capture long-term dependencies
   - Each conditional only considers a small window of size *n*.
   - Consider "*the same* ***stump*** *which had impaled the car of many a guest in the past thirty years and which he refused to have* ***removed***"
   - It is impossible to tell "removed" is likely by looking at the four preceding tokens.

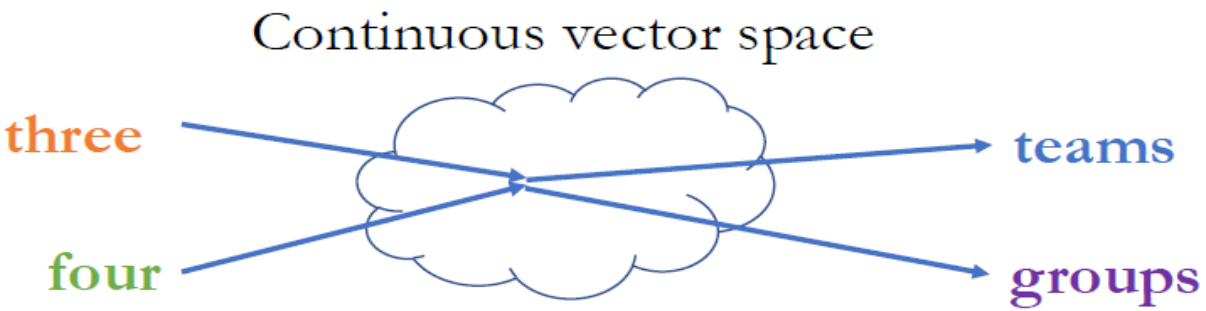
Neural N-Gram Language Model

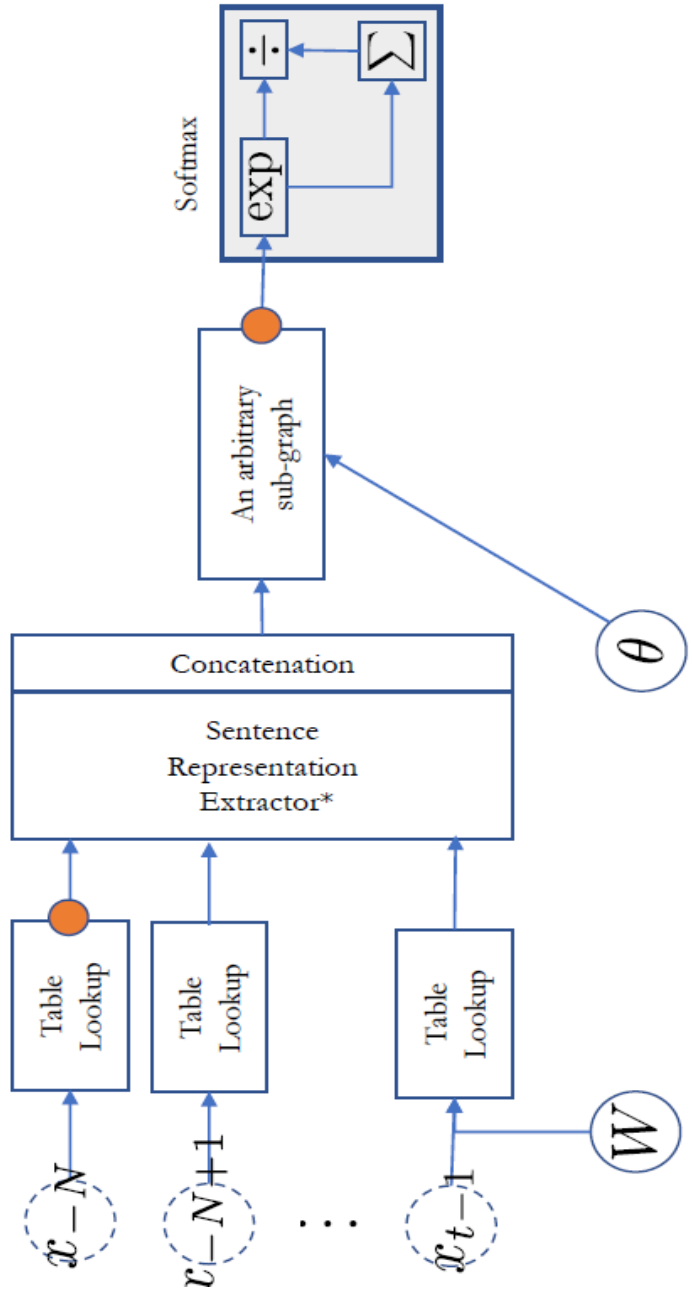
- The first extension of n-gram language models using a neural network



Neural N-Gram Language Model

- Training examples
  - there are **three teams** left for qualification.
  - **four teams** have passed the first round.
  - **four groups** are playing in the field.
- Q: how likely is "groups" followed by "three"?





Neural N-Gram Language Model

- In practice,

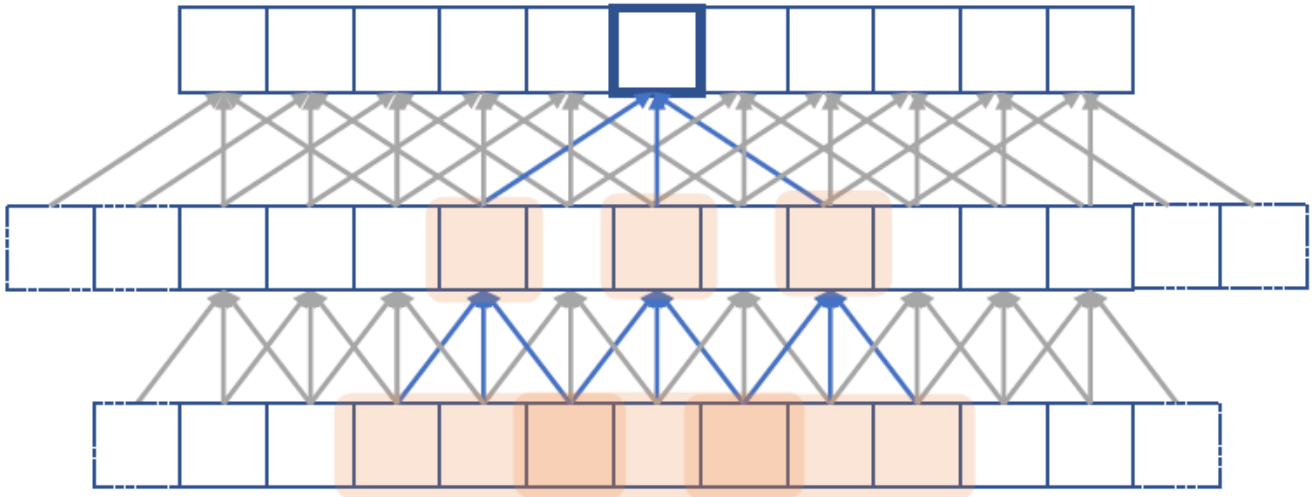
1. Collect all n-grams from the corpus.
2. Shuffle all the n-grams to build a training set
3. Train the neural n-gram language model using stochastic gradient descent on minibatches containing 100-1000 n-grams.
4. Early-stop based on the validation set.
5. Report perplexity on the test set.

$$\text{ppl} = b^{\frac{1}{|D|} \sum_{(x_1,\ldots,x_N) \in D} \log_b p(x_N | x_1,\ldots,x_{N-1})}$$

Increasing the Context Size - Convolutional Language Model

[Kalchbrenner et al., 2015; Dauphin et al., 2016]

- Dilated convolution to rapidly increase the window size
  - Exponential-growth of the window by introducing a multiplicative factor
  - By carefully selecting the multiplicative factor, no loss in the information.

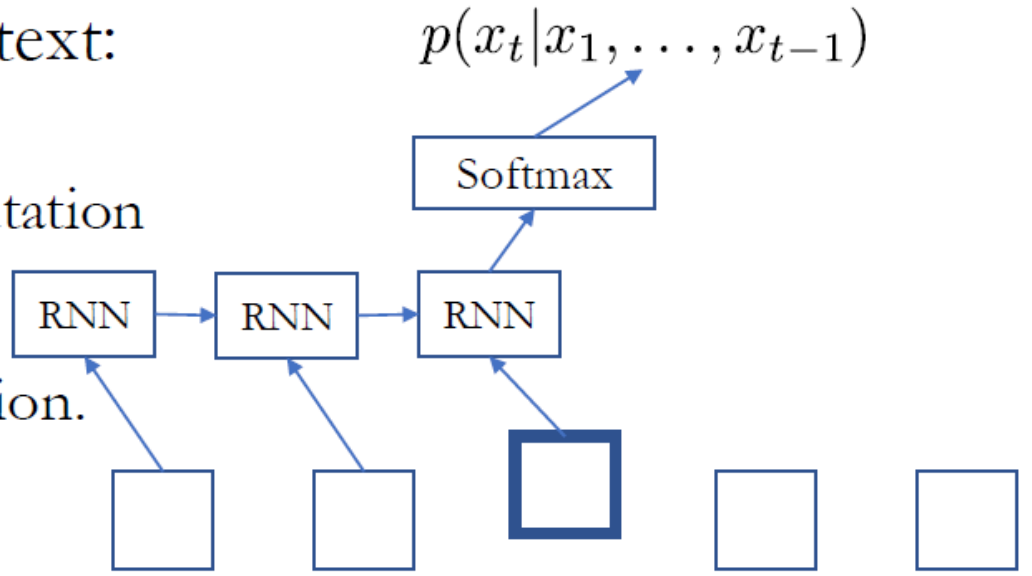


Infinite Context : n → ∞
- CBoW Language Model

- Equivalent to the neural LM after replacing "concat" with "average"
  - "Averaging" allows the model to consider the infinite large context window.
- Extremely efficient, but a weak language model
  - Ignores the order of the tokens in the context windows.
    - Any language with a fixed order cannot be modelled well.
  - Averaging ignores the absolute counts, which may be important:
    - If the context window is larger, "verb" becomes less likely in SVO languages.

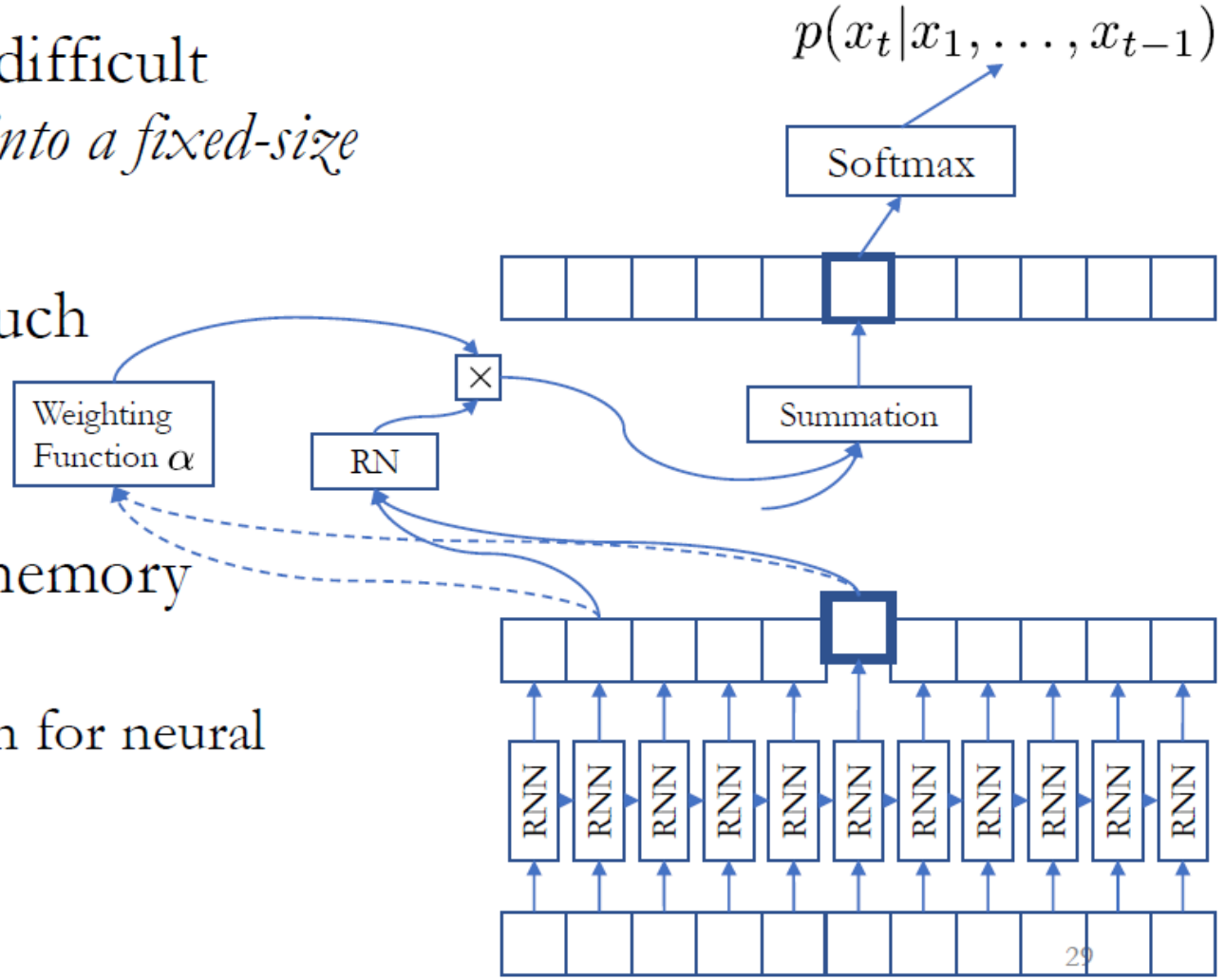
Infinite Context : n → ∞
- Recurrent Language Model

- A recurrent network summarizes all the tokens so far.
- Use the recurrent network's memory to predict the next token.
- Efficient online processing of a streaming text:
  - Constant time per step.
  - Constant memory throughout forward computation
- Useful in practice:
  - Useful for autocomplete and keyword suggestion.
  - Scoring partial hypotheses in generation.



Infinite Context : n → ∞
- Recurrent Language Model

- The **recurrent network** solves a difficult problem: *compress the entire context into a fixed-size memory vector.*
- **Self-attention** does not require such compression but still can capture long-term dependencies.
- Combine these two: a recurrent memory network (RMN) [Tran et al., 2016]
  - RNMT+: a similar, recent extension for neural machine translation



Machine Translation

- Input: a sentence written in a source language L_S
- Output: a corresponding sentence in a target language L_T
- Problem statement:
  - Supervised learning: given the input sentence, output its translation
  - Compute the conditional distribution over all possible translation given the input

    $p(Y = (y_1, \ldots, y_T) | X = (x_1, \ldots, x_{T'}))$

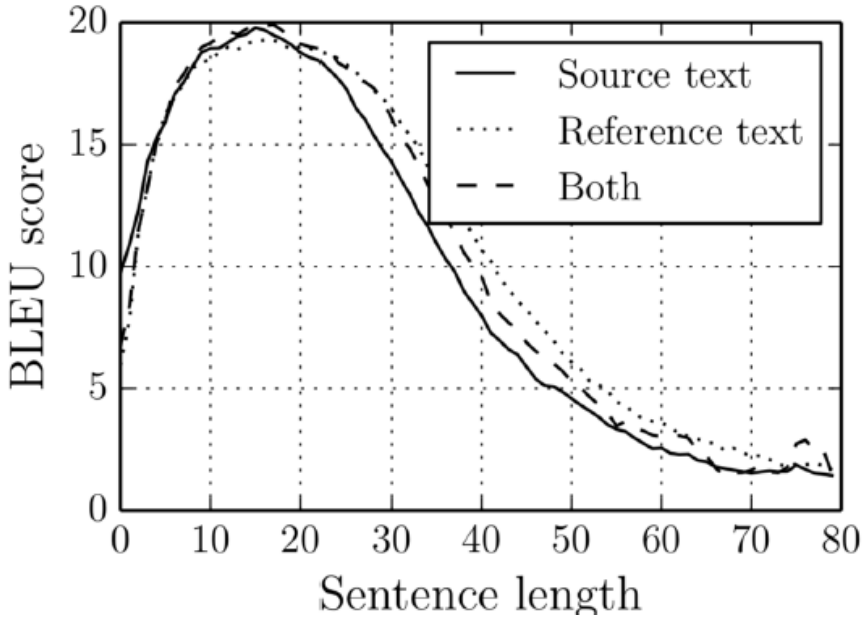
- *We have already learned every necessary ingredient for building a full neural machine translation system.*

Encoder – Source Sentence Representation

- Encode the source sentence into a set of sentence representation vectors
  - # of encoded vectors is proportional to the source sentence length: often same. $H = (h_1, \ldots, h_{T'})$
  - Recurrent networks have been widely used [Cho et al., 2014; Sutskever et al., 2014], but CNN [Gehring et al., 2017; Kalchbrenner&Blunsom, 2013] and self-attention [Vaswani et al., 2017] are used increasingly more often. See Lecture 2 for details.
- We do not want to collapse them into a single vector.
  - Collapsing often corresponds to information loss.
  - Increasingly more difficult to encode the entire source sentence into a single vector, as the sentence length increases [Cho et al., 2014b].
  - We didn't know initially until [Bahdanau et al., 2015].

Encoder – Source Sentence Representation

- Encode the source sentence into a set of sentence representation vectors
- We do not want to collapse them into a single vector.
  - Increasingly more difficult to encode the entire source sentence into a single vector, as the sentence length increases [Cho et al., 2014b].



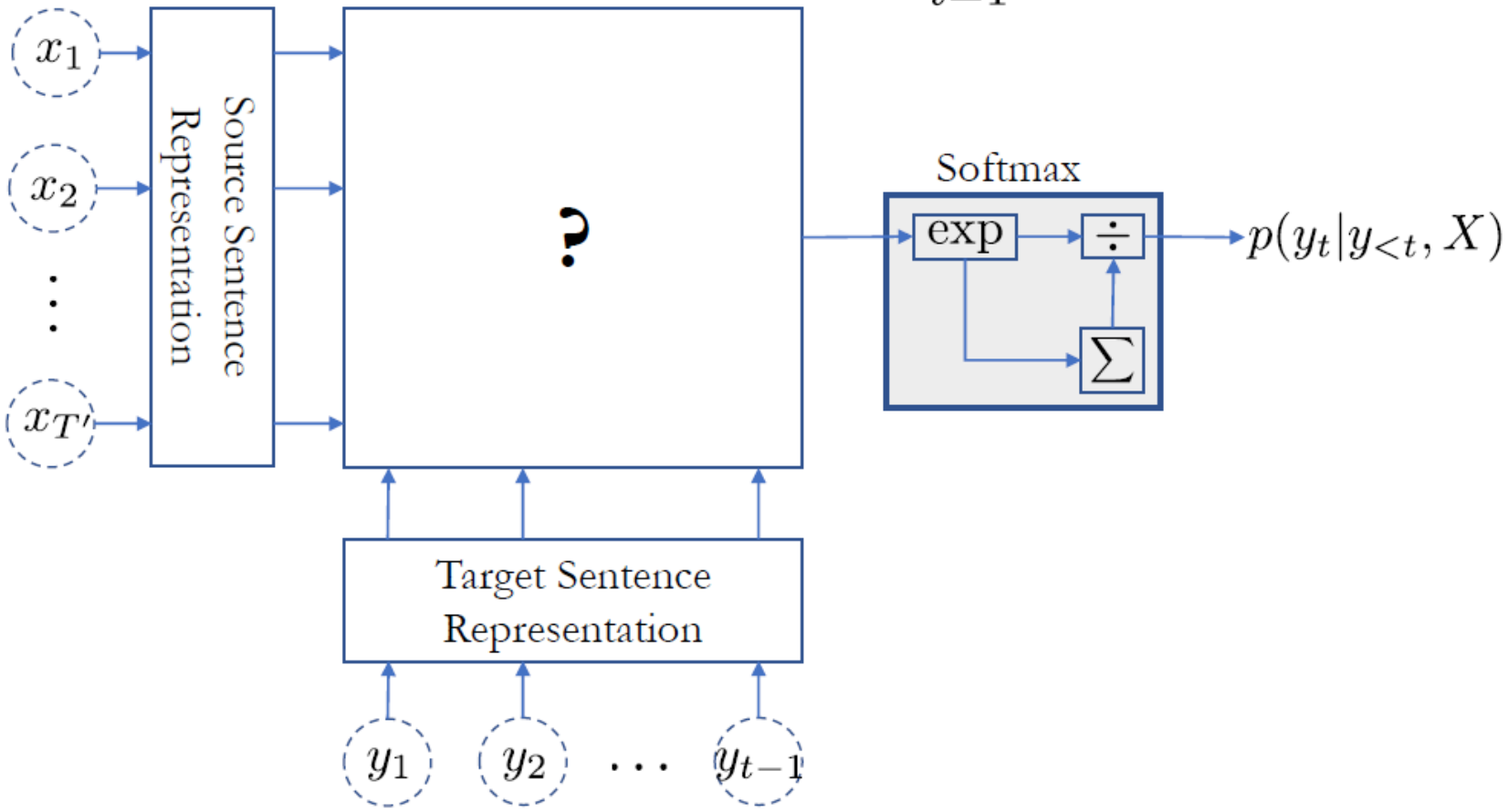
Decoder – Language Modelling

- Autoregressive Language modelling with an infinite context $n \to \infty$
  - Larger context is necessary to generate a coherent sentence.
    - Semantics could be largely provided by the source sentence, but syntactic properties need to be handled by the language model directly.
  - Recurrent networks, self-attention and (dilated) convolutional networks
    - Causal structure must be followed.
    - See Lecture 3.
- **Conditional** Language modelling
  - The context based on which the next token is predicted is **two-fold**

$$p(Y|X) = \prod_{t=1}^{T} p(y_t|y_{<t}, X)$$

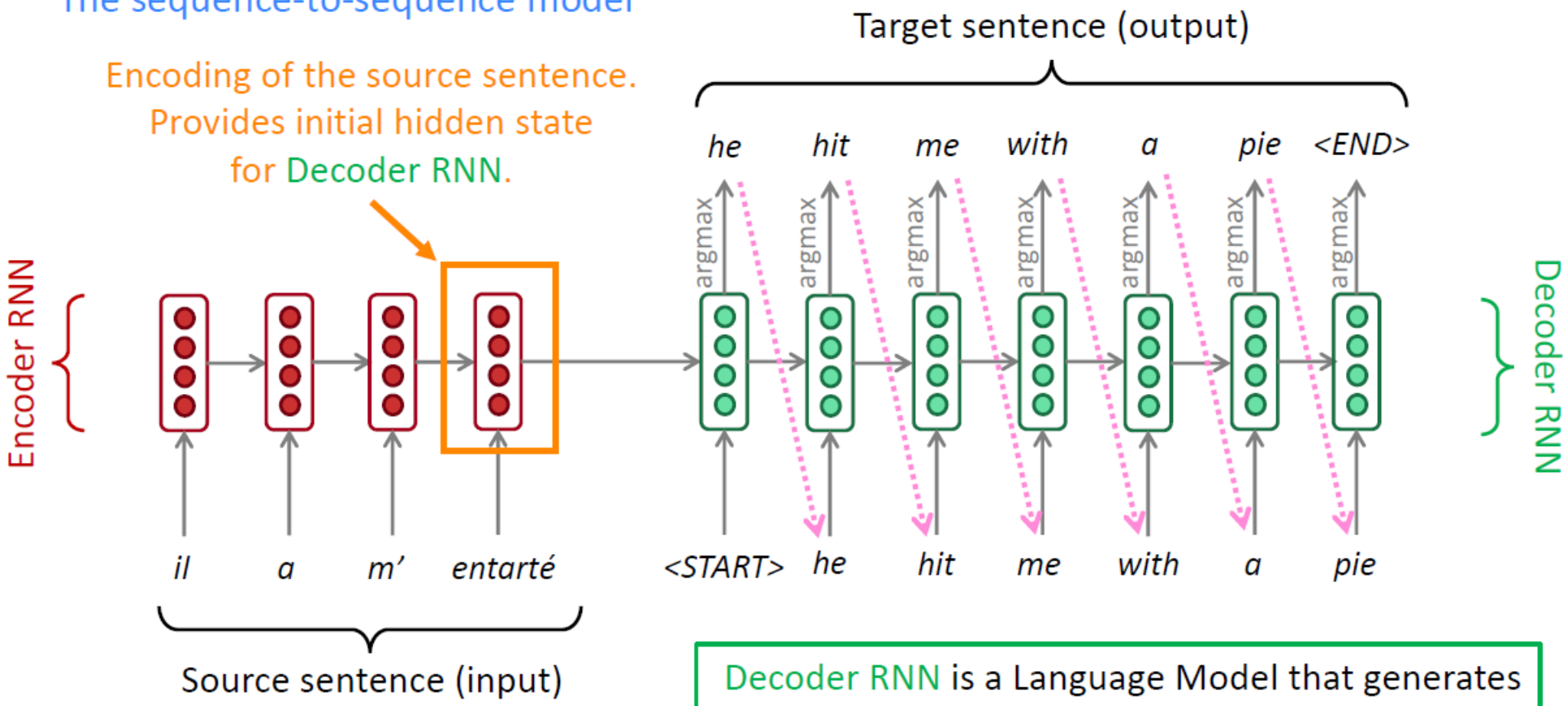
Decoder – Language Modelling

- Conditional Language modelling $p(Y|X) = \prod_{t=1}^{T} p(y_t|y_{<t}, X)$



Neural Machine Translation (NMT)

The sequence-to-sequence model



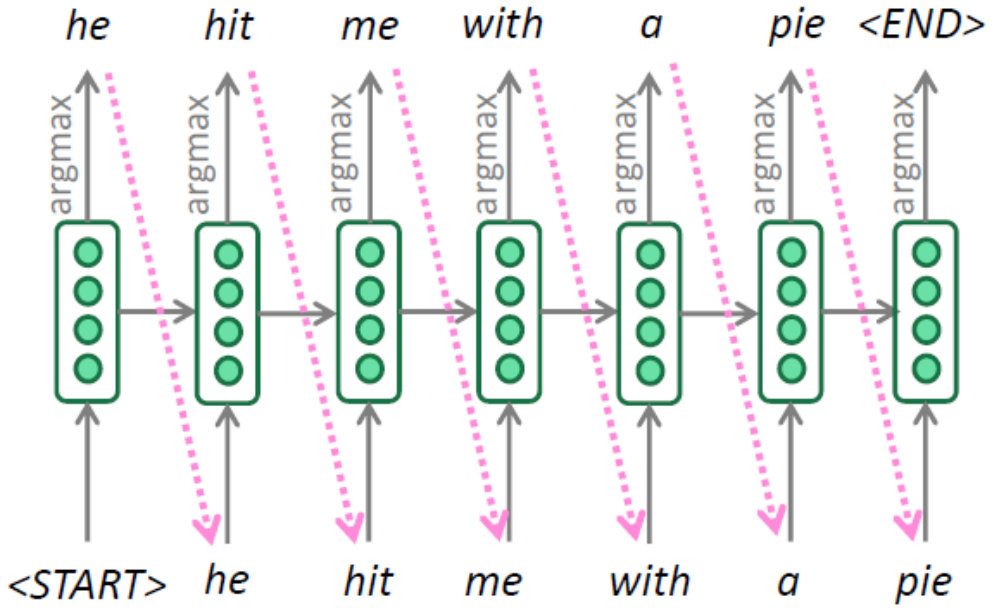
Encoder RNN produces an encoding of the source sentence.

Decoder RNN is a Language Model that generates target sentence, *conditioned on encoding*.

Note: This diagram shows **test time** behavior: decoder output is fed in ⋯→ as next step's input

Greedy Decoding

- We saw how to generate (or "decode") the target sentence by taking argmax on each step of the decoder



- This is greedy decoding (take most probable word on each step)
- **Problems with this method?**

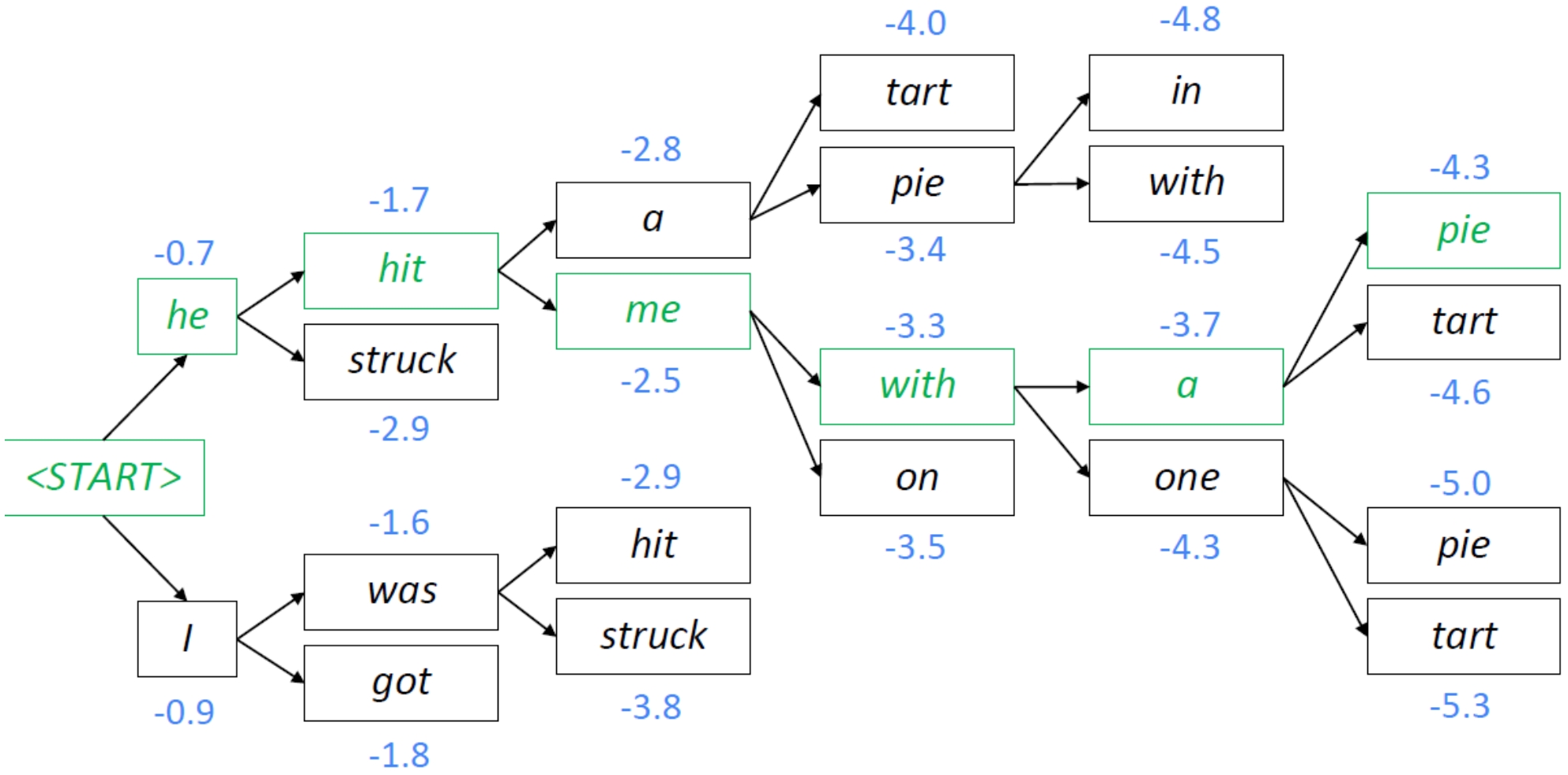
Problems with Greedy Decoding

- Greedy decoding has no way to undo decisions!
  - Input: *il a m'entarté* *(he hit me with a pie)*
  - → *he* ____
  - → *he hit* ____
  - → *he hit a* ____ (whoops! no going back now…)

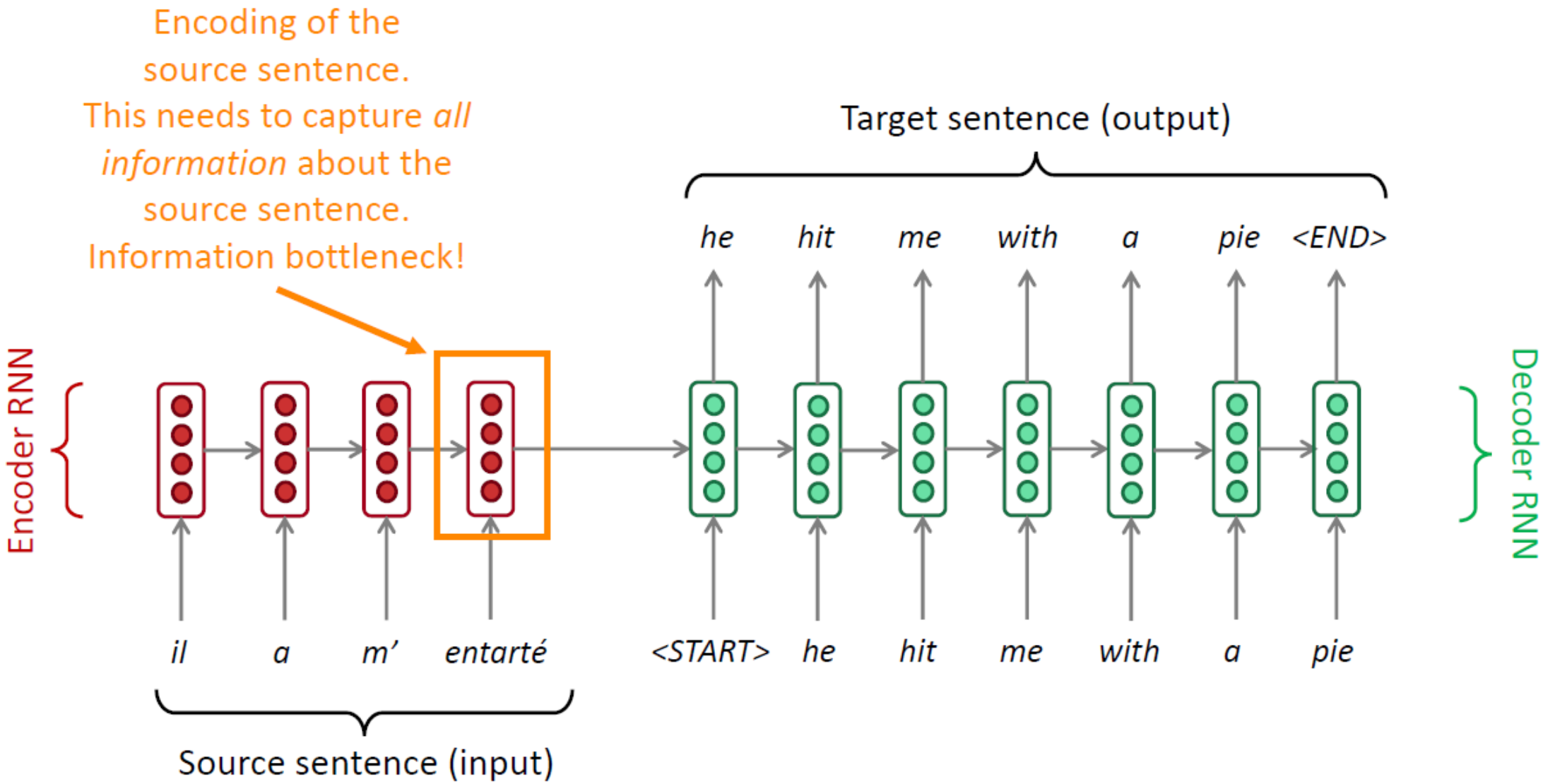
- How to fix this?

Beam Search Decoding

Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



Sequence-to-Sequence: the Bottleneck Problem



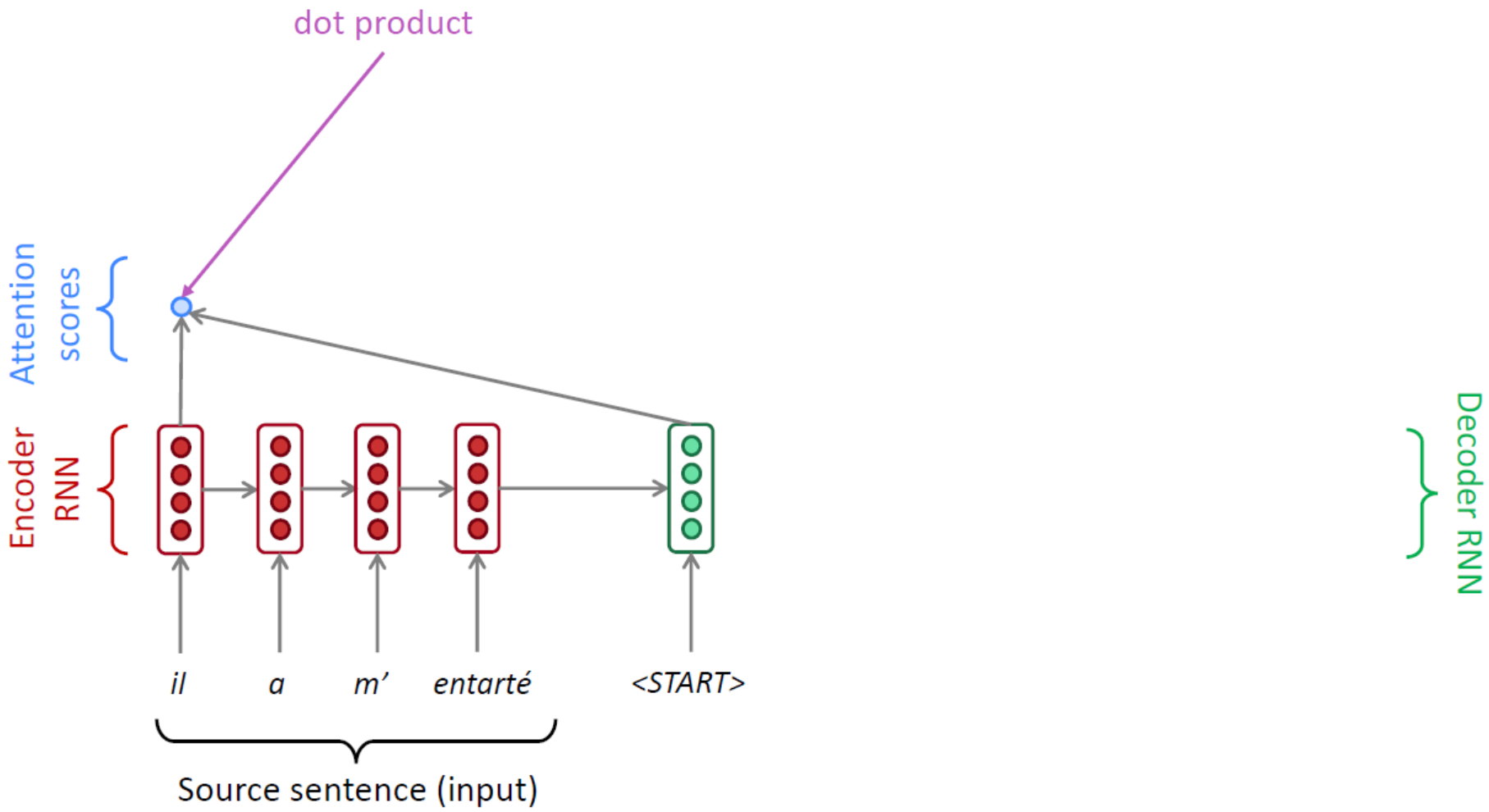
Attention

- **Attention** provides a solution to the bottleneck problem.

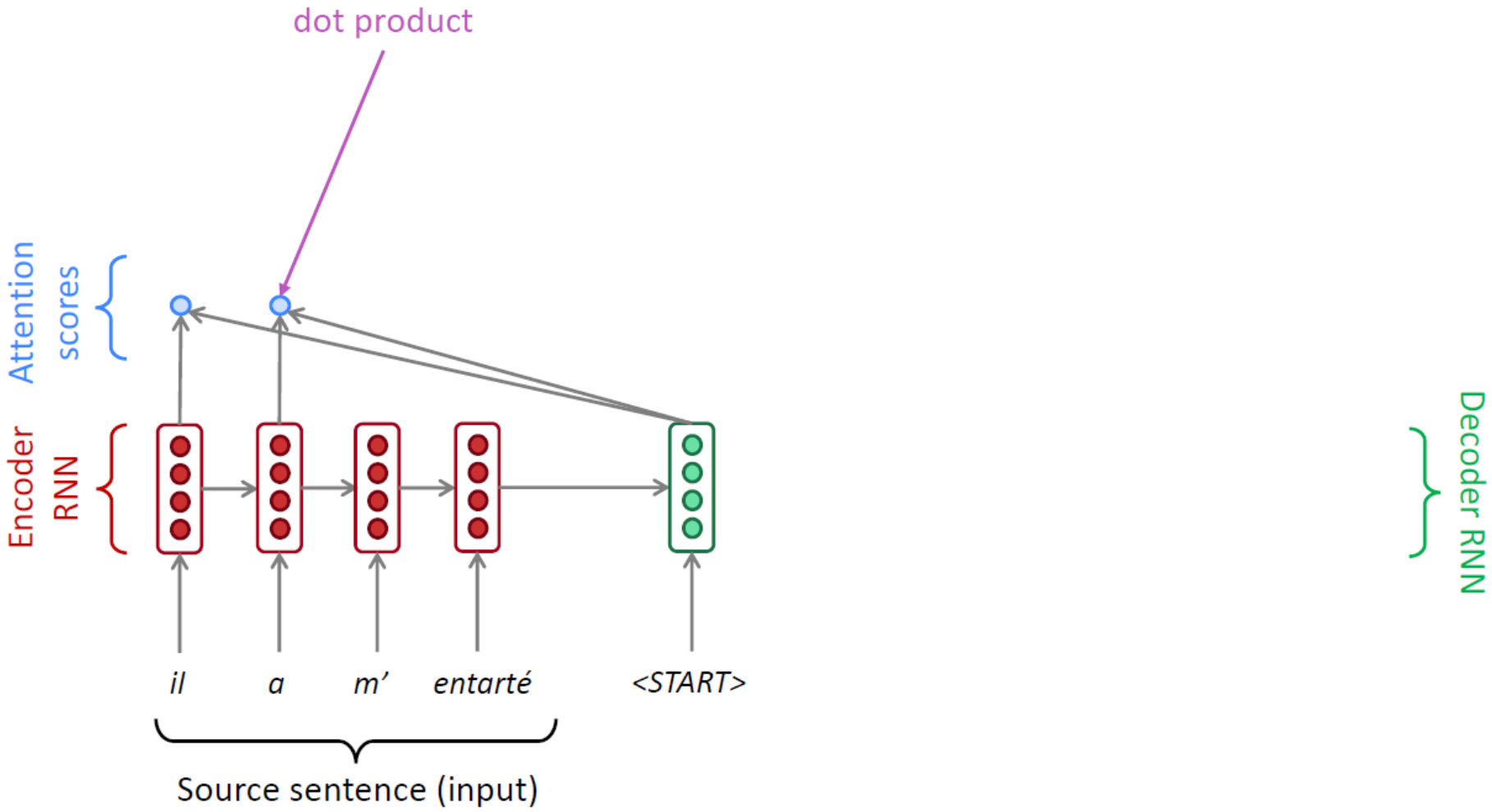
- Core idea: on each step of the decoder, use *direct connection to the encoder* to *focus on a particular part* of the source sequence

- First we will show via diagram (no equations), then we will show with equations

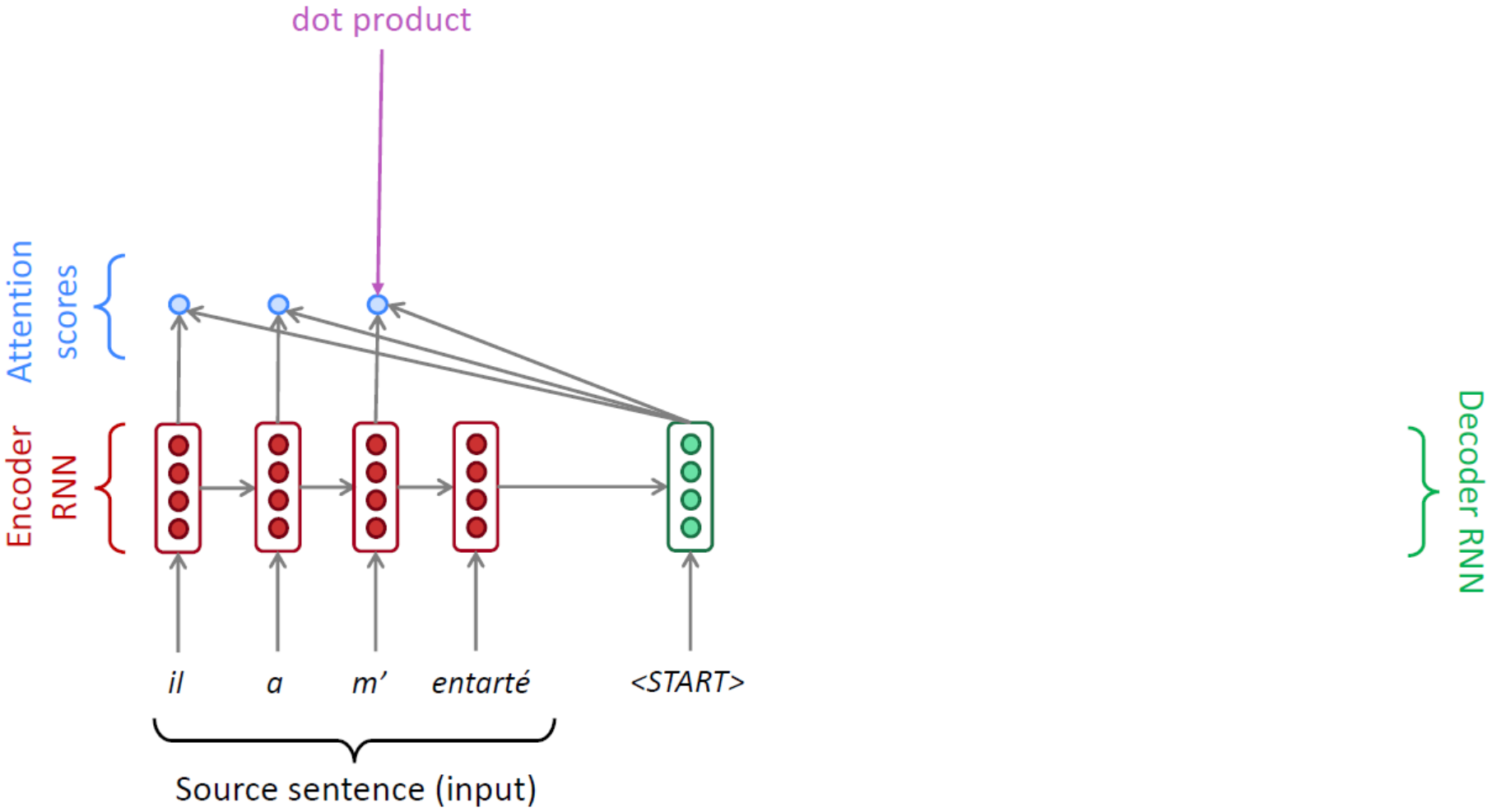
Sequence-to-Sequence with Attention



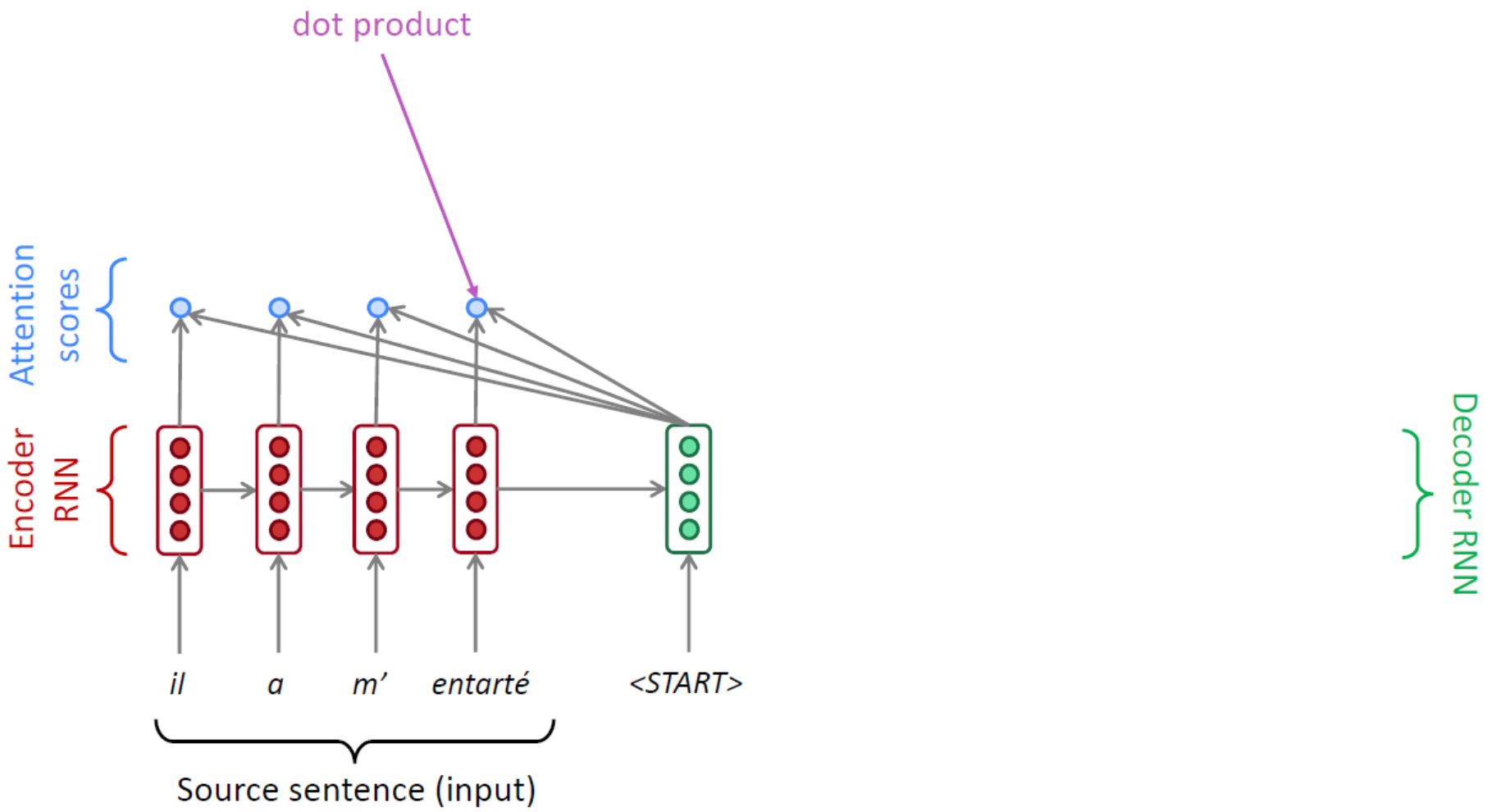
Sequence-to-Sequence with Attention



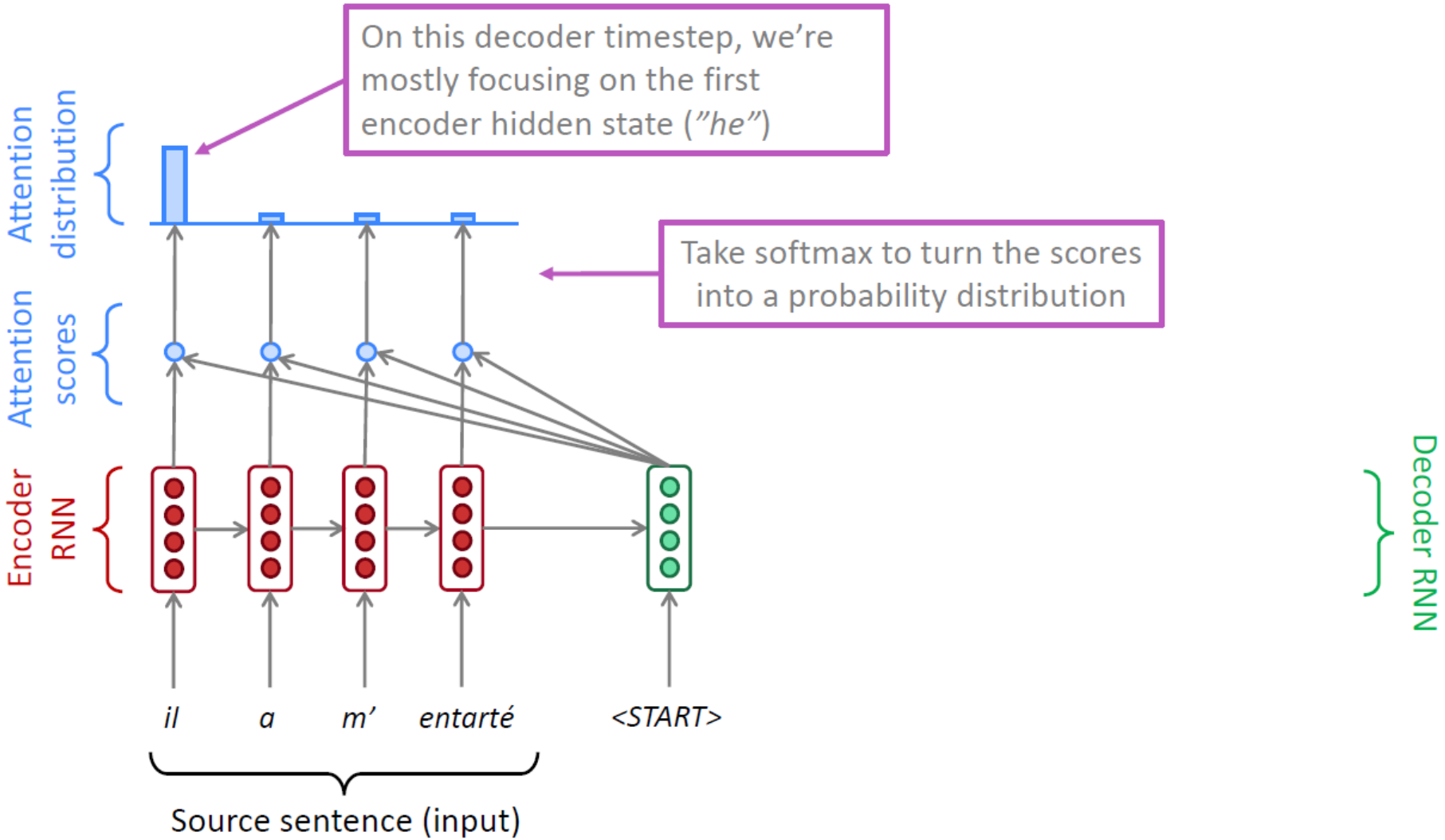
Sequence-to-Sequence with Attention



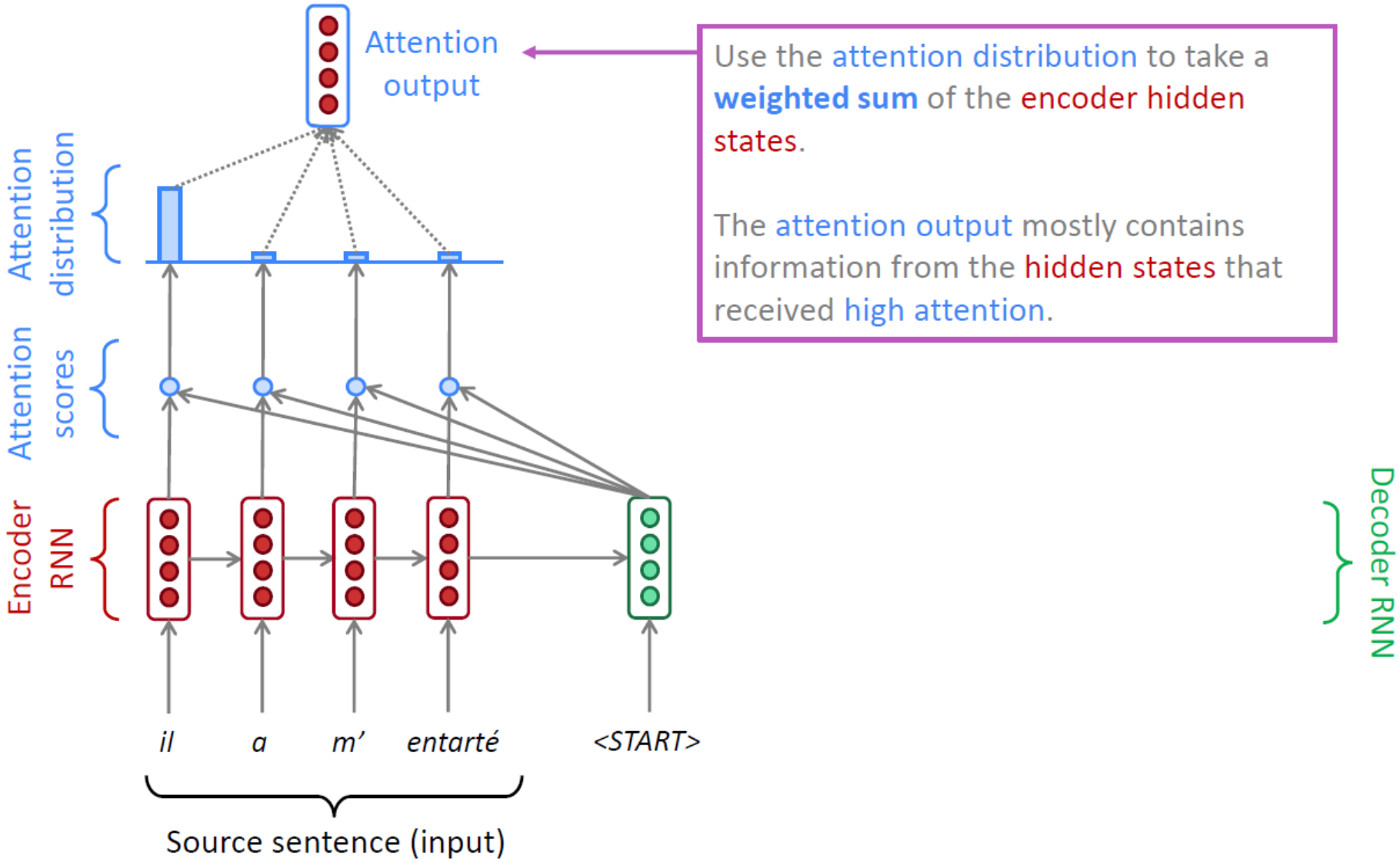
Sequence-to-Sequence with Attention



Sequence-to-Sequence with Attention



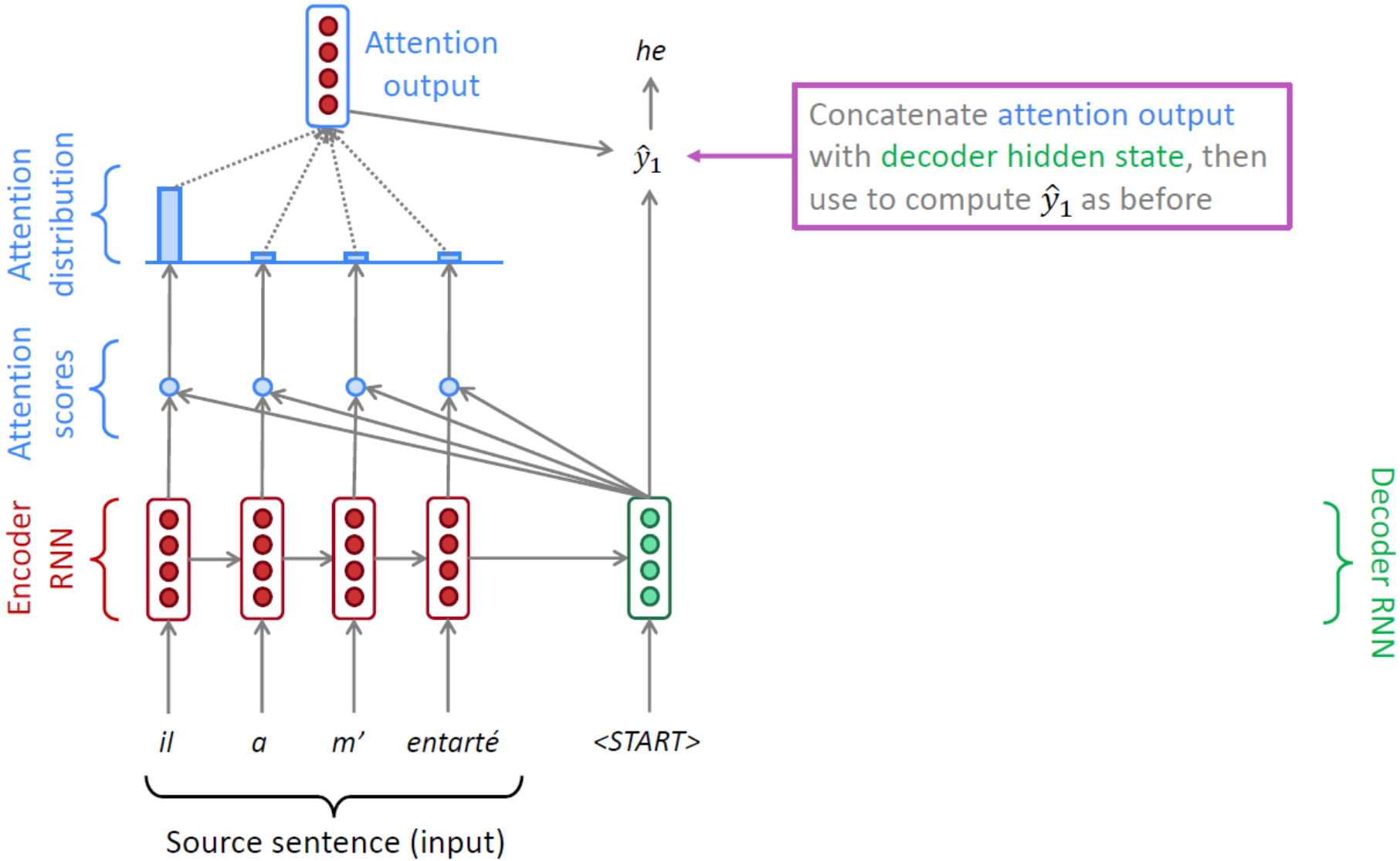
Sequence-to-Sequence with Attention



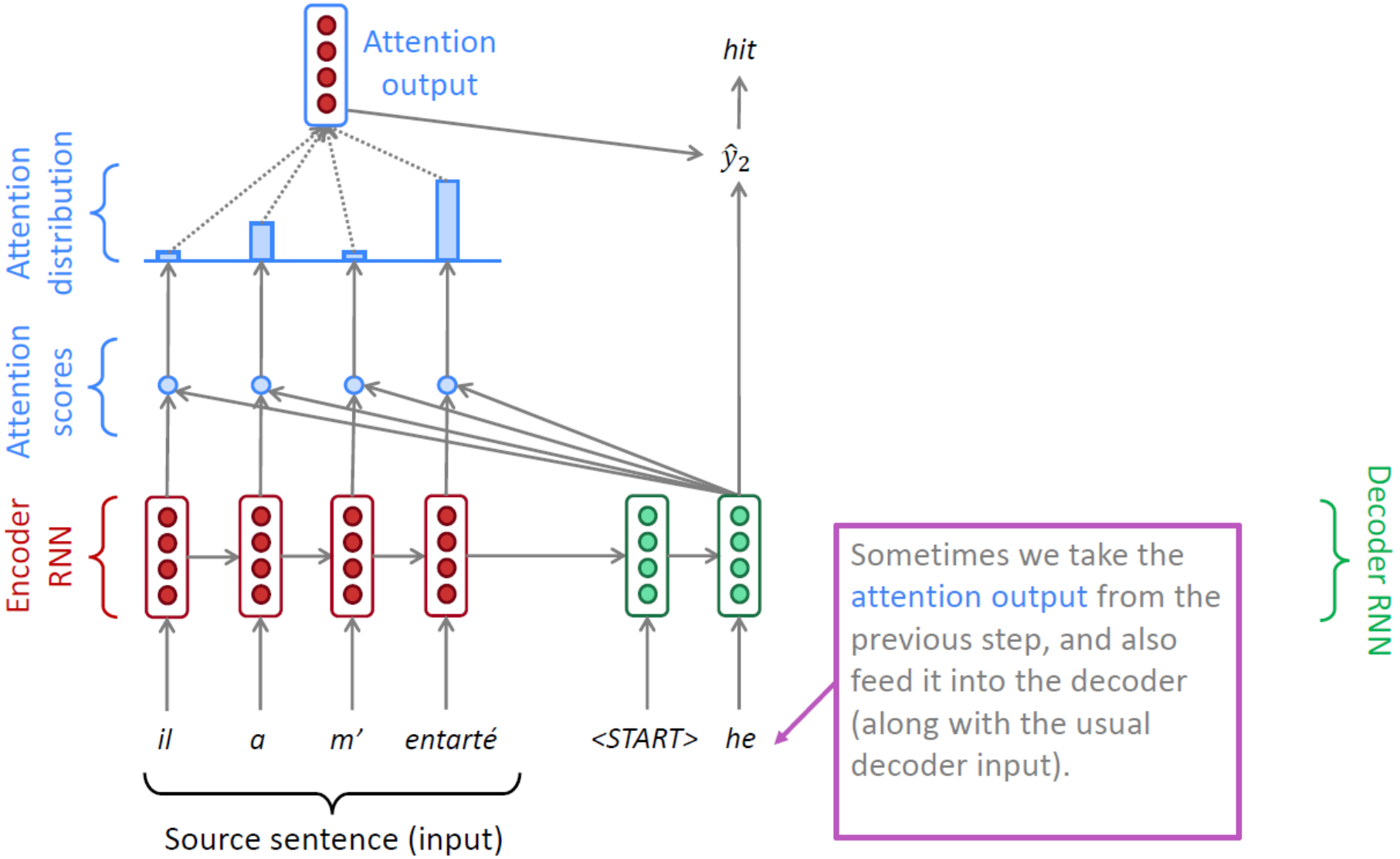
Use the attention distribution to take a **weighted sum** of the encoder hidden states.

The attention output mostly contains information from the hidden states that received high attention.

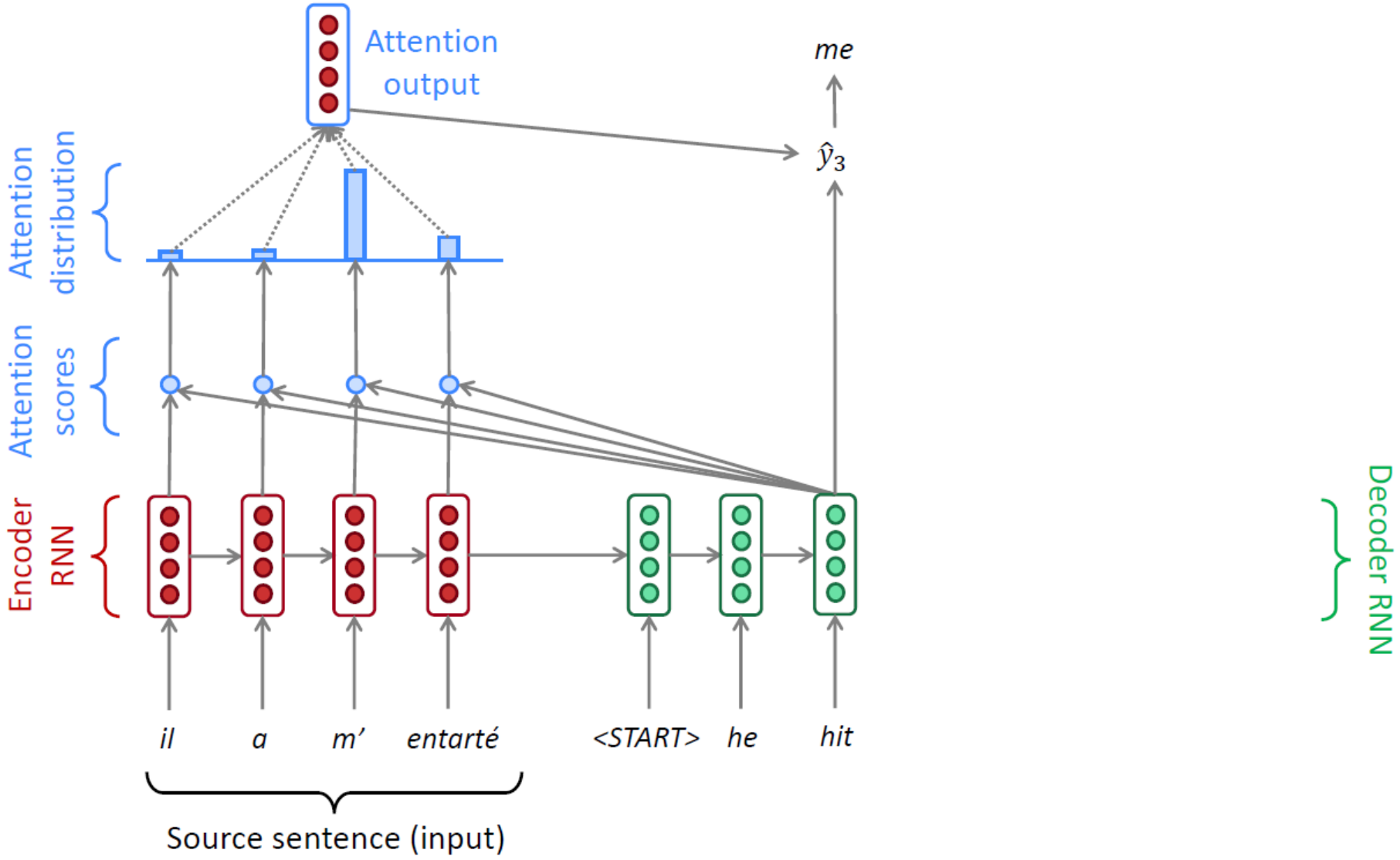
Sequence-to-Sequence with Attention



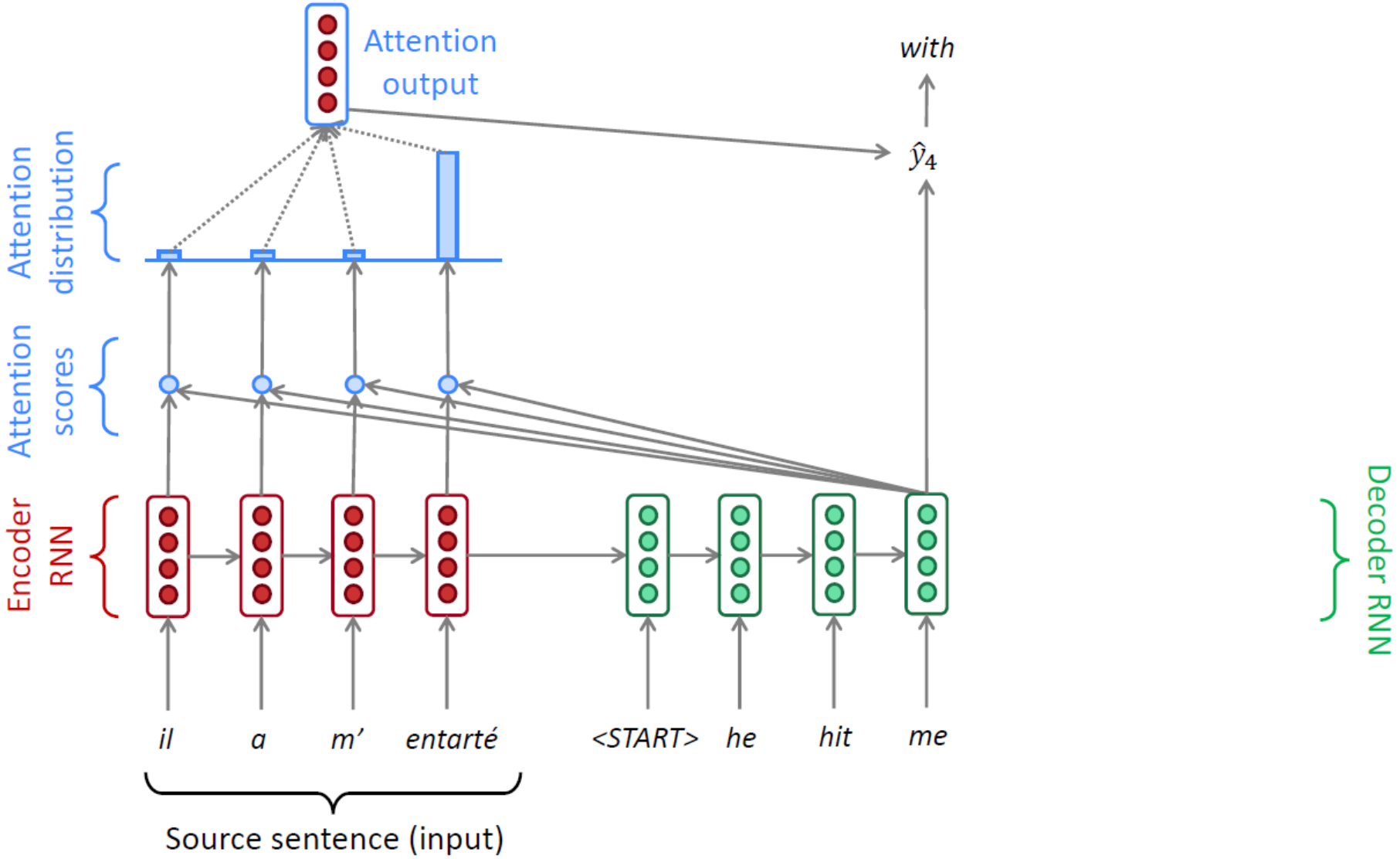
Sequence-to-Sequence with Attention



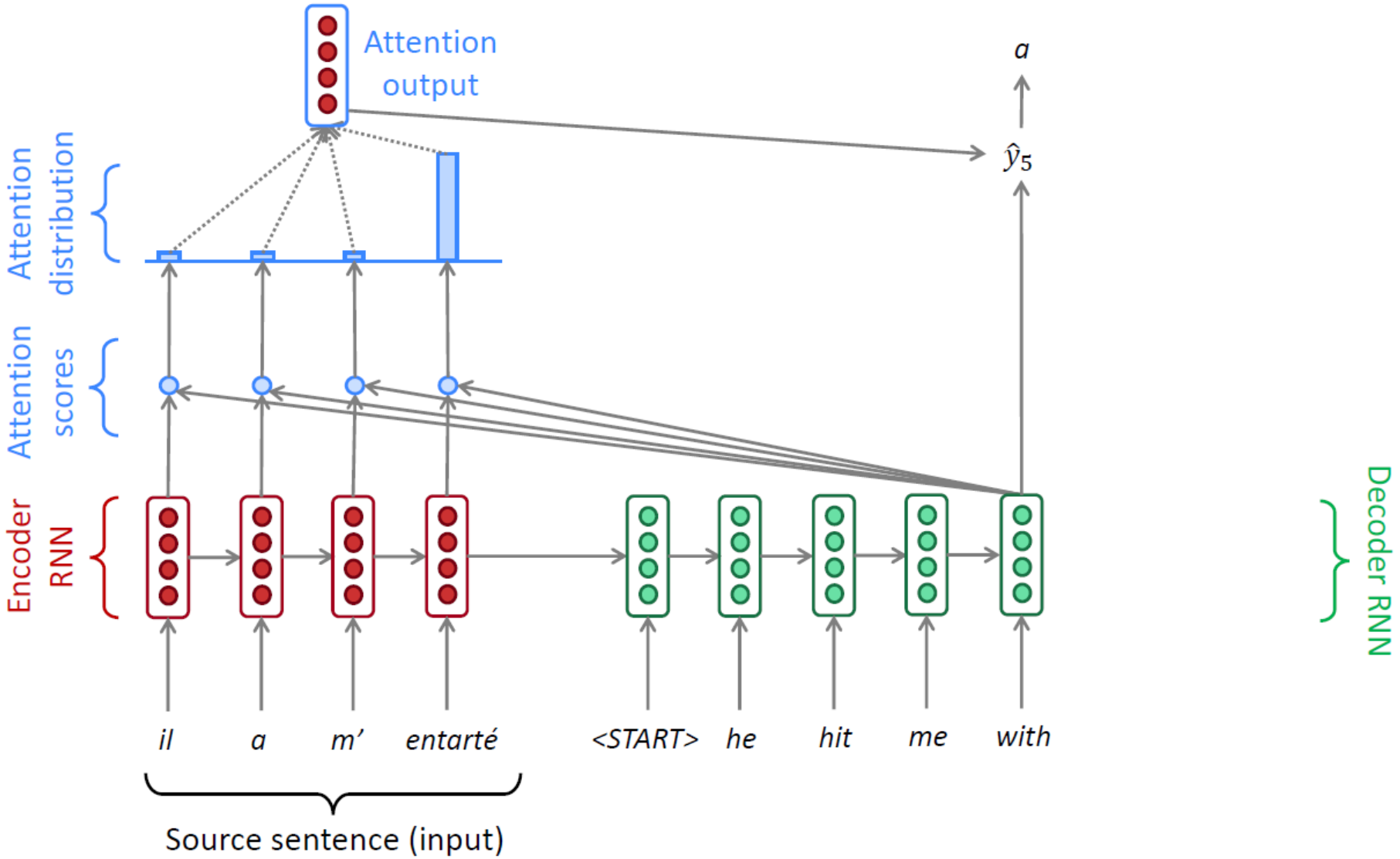
Sequence-to-Sequence with Attention



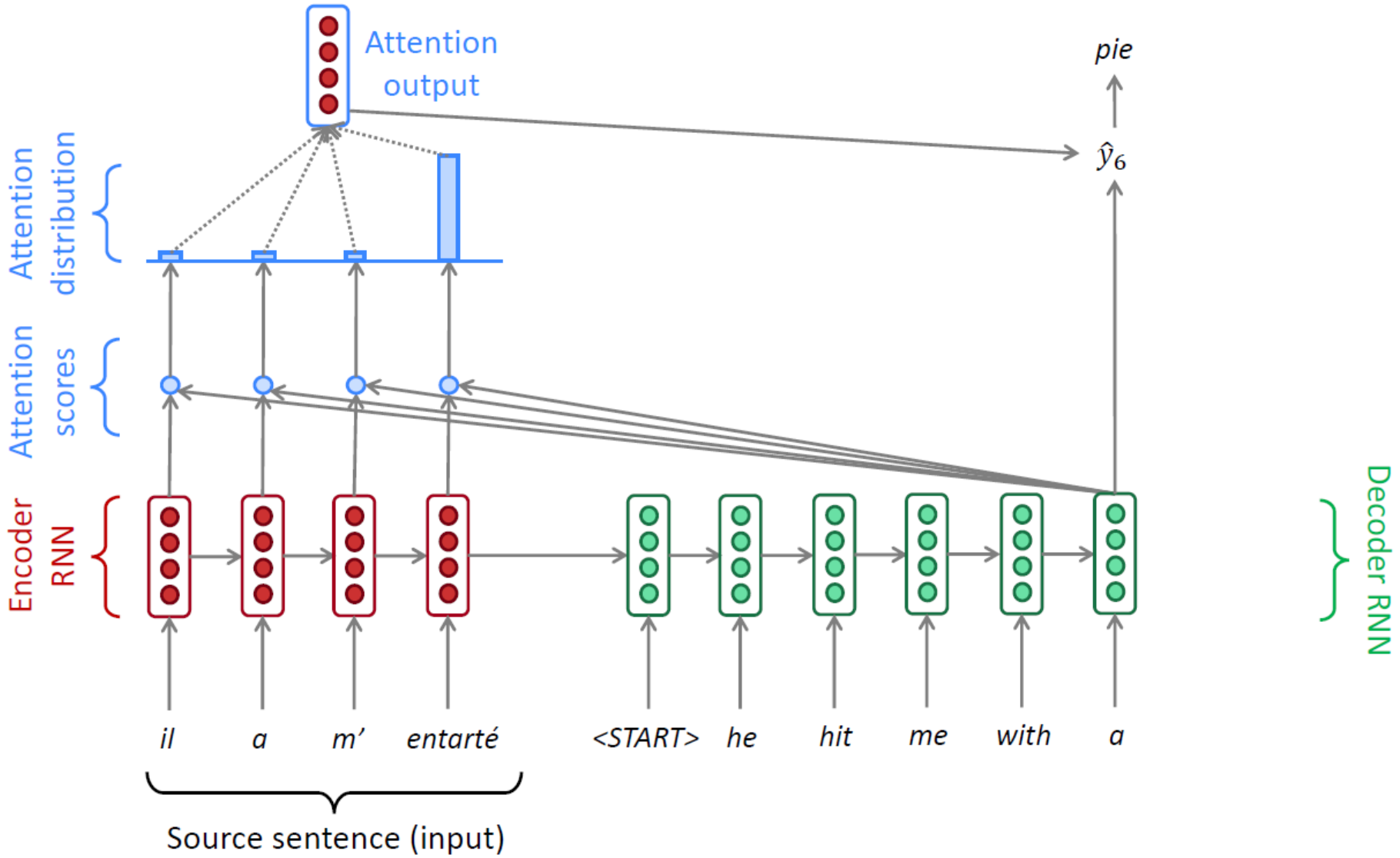
Sequence-to-Sequence with Attention



Sequence-to-Sequence with Attention



Sequence-to-Sequence with Attention



Attention: In Equation

- We have encoder hidden states $h_1, \ldots, h_N \in \mathbb{R}^h$
- On timestep *t*, we have decoder hidden state $s_t \in \mathbb{R}^h$
- We get the attention scores $\boldsymbol{e}^t$ for this step:

$$\boldsymbol{e}^t = [\boldsymbol{s}_t^T \boldsymbol{h}_1, \ldots, \boldsymbol{s}_t^T \boldsymbol{h}_N] \in \mathbb{R}^N$$

- We take softmax to get the attention distribution α^t for this step (this is a probability distribution and sums to 1)

$$\alpha^t = \text{softmax}(\boldsymbol{e}^t) \in \mathbb{R}^N$$

- We use α^t to take a weighted sum of the encoder hidden states to get the attention output $\boldsymbol{a}_t$

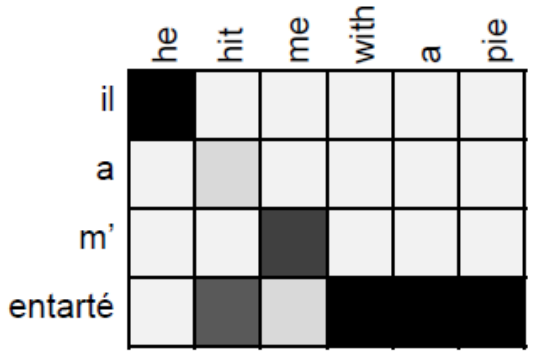
$$\boldsymbol{a}_t = \sum_{i=1}^{N} \alpha_i^t \boldsymbol{h}_i \in \mathbb{R}^h$$

- Finally we concatenate the attention output $\boldsymbol{a}_t$ with the decoder hidden state s_t and proceed as in the non-attention seq2seq model

$$[\boldsymbol{a}_t; \boldsymbol{s}_t] \in \mathbb{R}^{2h}$$

Attention is Great

- Attention significantly improves NMT performance
  - It's very useful to allow decoder to focus on certain parts of the source
- Attention solves the bottleneck problem
  - Attention allows decoder to look directly at source; bypass bottleneck
- Attention helps with vanishing gradient problem
  - Provides shortcut to faraway states
- Attention provides some interpretability
  - By inspecting attention distribution, we can see what the decoder was focusing on
  - We get (soft) alignment for free!
  - This is cool because we never explicitly trained an alignment system
  - The network just learned alignment by itself

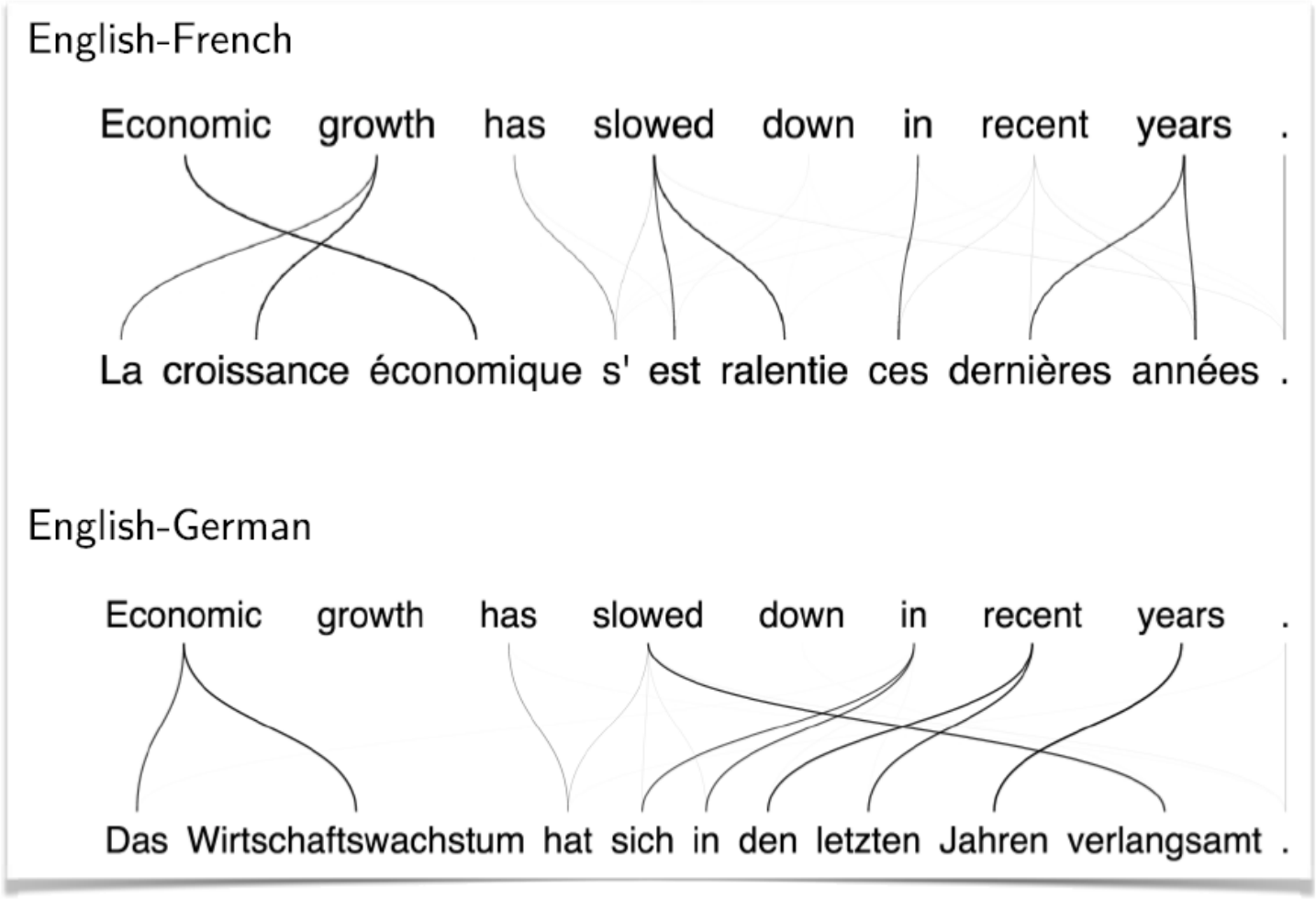


RNN Neural Machine Translation

- **Source**: *An admitting privilege is the right of a doctor to admit a patient to a hospital or a medical centre to carry out a diagnosis or a procedure, based on his status as a health care worker at a hospital.*
- **When collapsed:** *Un privilège d'admission est le droit d'un médecin de reconnaître un patient à l'hôpital ou un centre médical d'un diagnostic ou de prendre un diagnostic en fonction de son état de santé.*
- **RNNSearch**: *Un privilège d'admission est le droit d'un médecin d'admettre un patient à un hôpital ou un centre médical pour effectuer un diagnostic ou une procédure, selon son statut de travailleur des soins de santé à l'hôpital.*

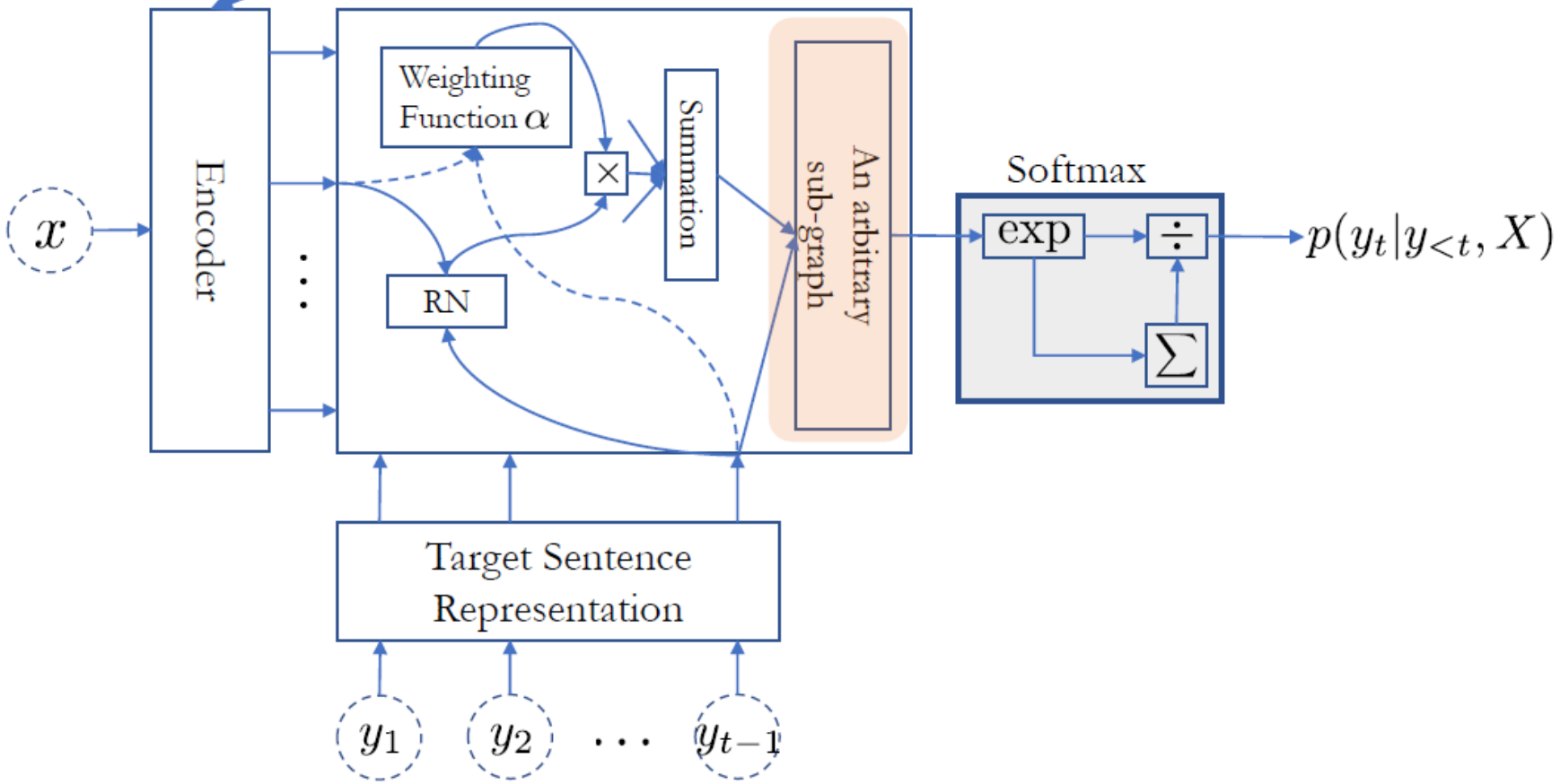
RNN Neural Machine Translation

- Sensible alignment between source and target tokens
- Capture long-range reordering/dependencies
- Without strong supervision on the alignment
  - Weakly supervised learning



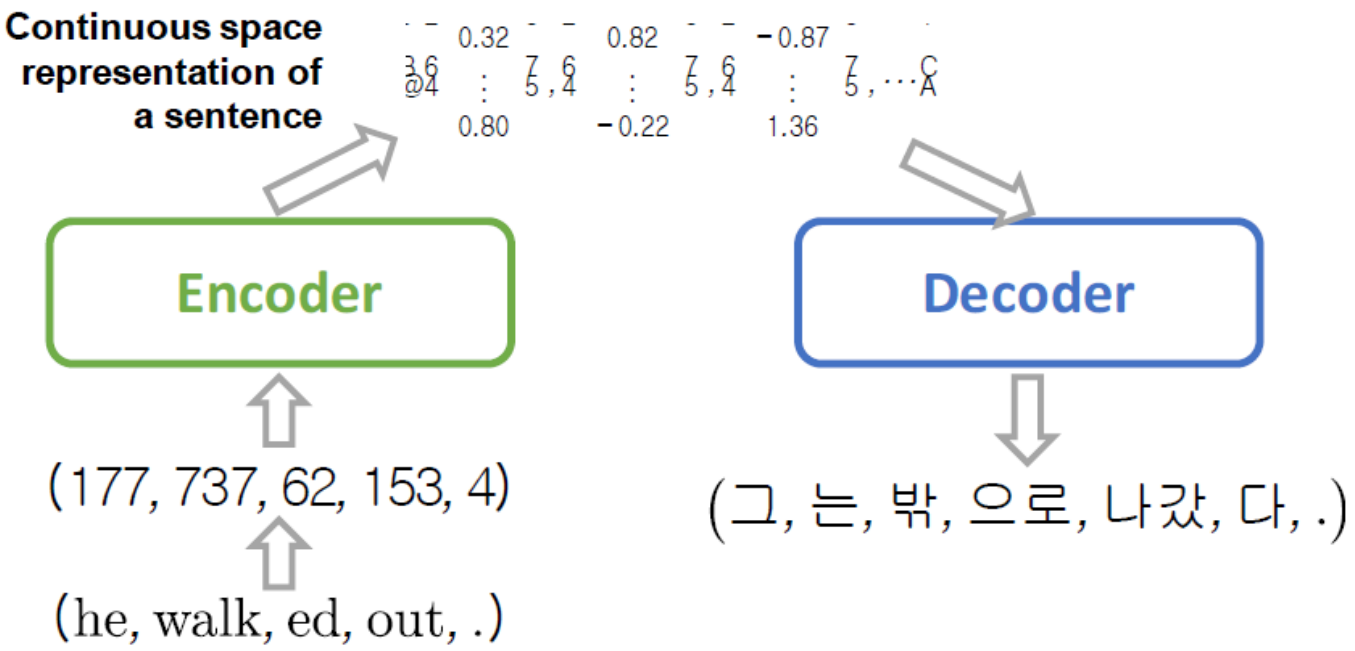
RNN Neural Machine Translation

- Input: **arbitrary as long as encoded into a set of continuous vectors**
- Output: a corresponding sentence in a target language

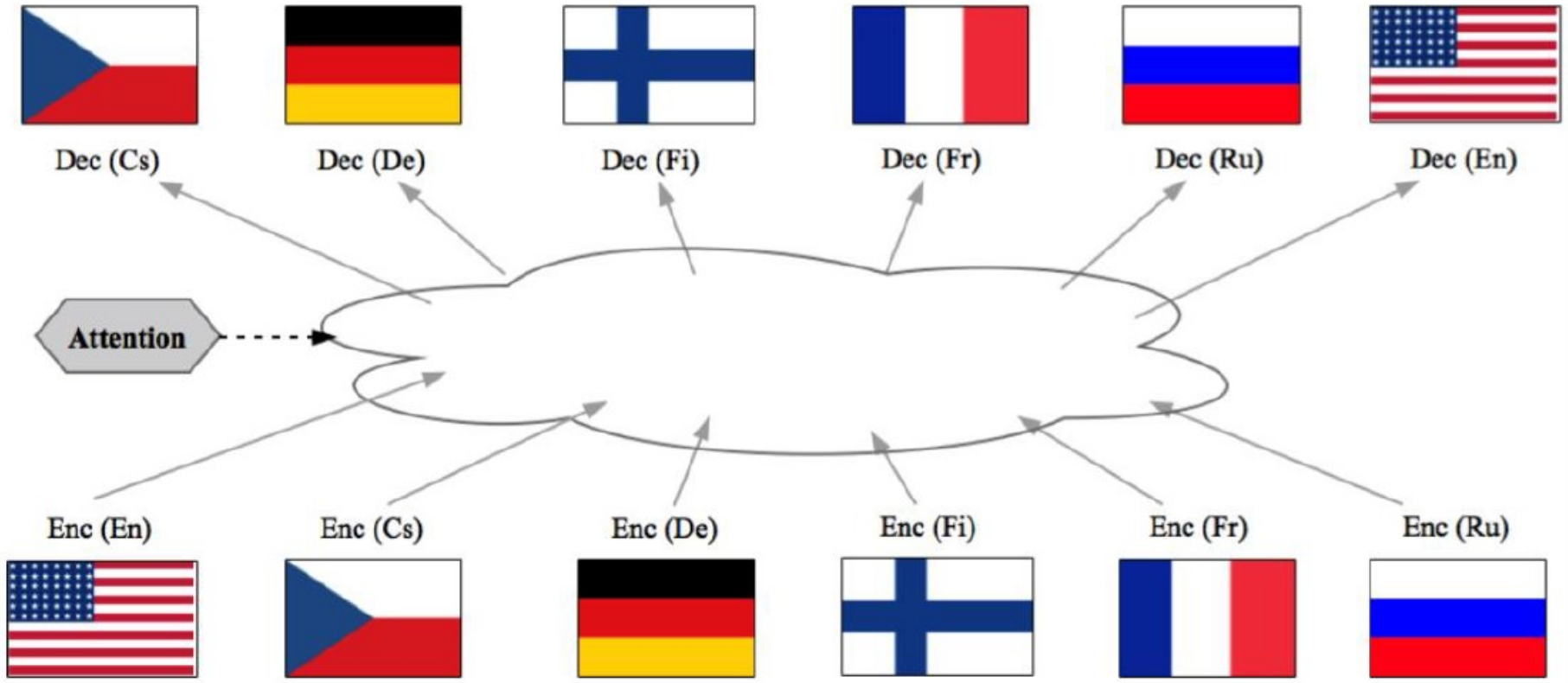


What Does NMT Do?

- Sequence of "discrete" symbols → Set of "continuous" vectors
- Continuous vectors *encode* semantics of discrete symbols
- Continuous vectors are *stripped* of hard, linguistic symbols
- *Can we map multiple languages on a single continuous space?*



Multilingual Neural Machine Translation



- Can this continuous vector space be shared across multiple languages?

Character-Level, Multilingual Translation

- Robust to intra-sentence code switching
- Huge saving in parameters: 4x less parameters without loss in BLEU

(e) Multilingual

Multi src	Bei der Metropolitního výboru pro dopravu für das Gebiet der San Francisco Bay erklärten Beamte , der Kongress könne das Problem банкротство доверительного Фонда строительства шоссейных дорог einfach durch Erhöhung der Kraftstoffsteuer lösen .
EN ref	At the Metropolitan Transportation Commission in the San Francisco Bay Area , officials say Congress could very simply deal with the bankrupt Highway Trust Fund by raising gas taxes .
bpe2char	During the Metropolitan Committee on Transport for San Francisco Bay , officials declared that Congress could solve the problem of bankruptcy by increasing the fuel tax bankrupt .
char2char	At the Metropolitan Committee on Transport for the territory of San Francisco Bay , officials explained that the Congress could simply solve the problem of the bankruptcy of the Road Construction Fund by increasing the fuel tax .

Image Caption Generation

- Input: an image
- Output: an image caption
- Network Architecture
  - Encoder: deep convolution network
  - Decoder: recurrent language model with the attention mechanism.
- Data: image-caption pairs

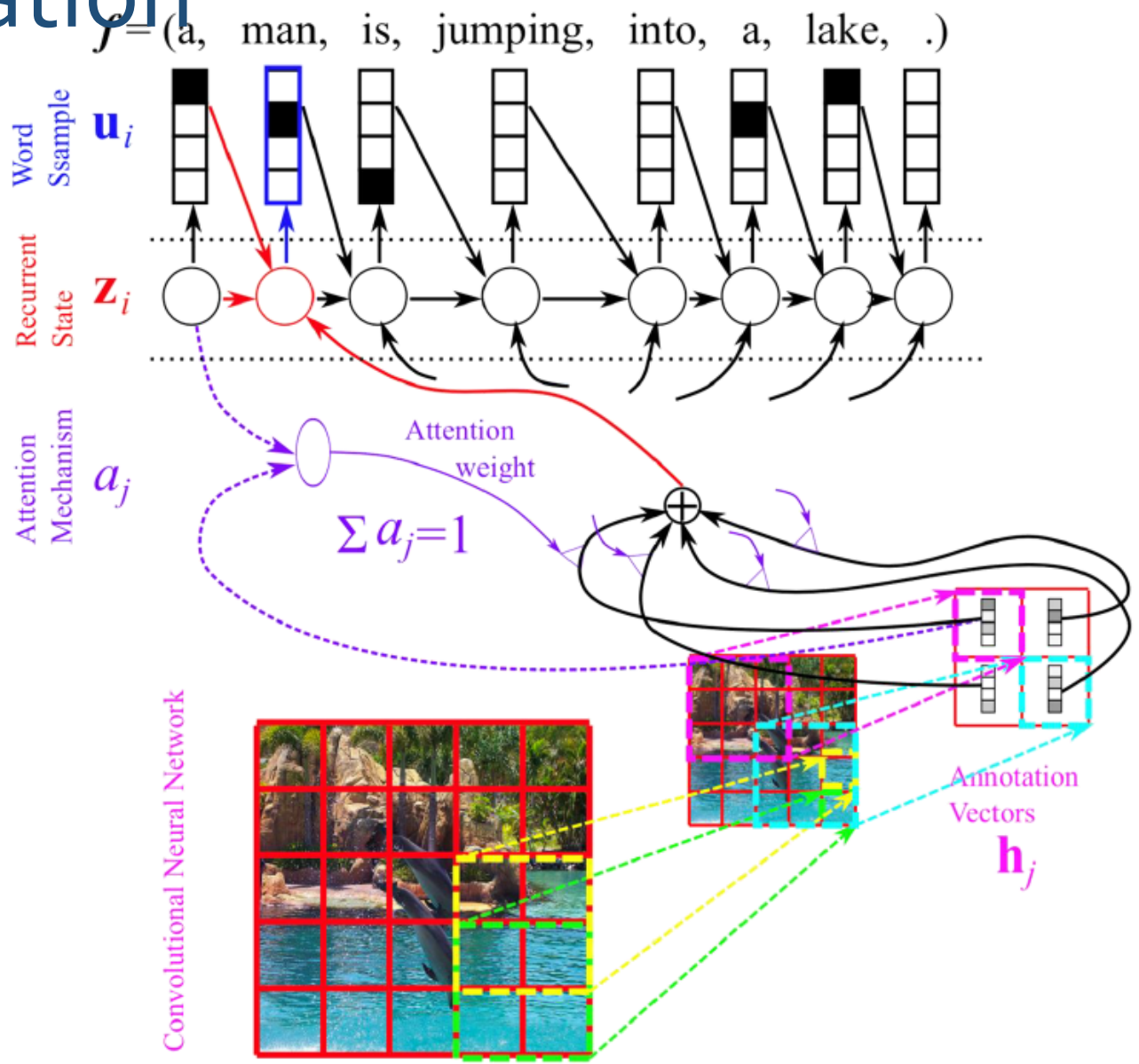


Image Caption Generation

A woman is throwing a frisbee in a park.

A dog is standing on a hardwood floor.

A stop sign is on a road with a mountain in the background.

A little girl sitting on a bed with a teddy bear.

A group of people sitting on a boat in the water.

A giraffe standing in a forest with trees in the background.

Speech Recognition

- Input: Speech
- Output: transcription