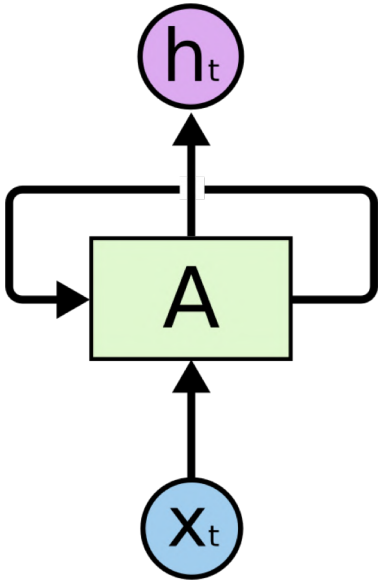
# Recurrent Neural Networks
# +
# Attention Models




Adapted from: Stanford CS231 , Illinois cs498 , Jeremy Jordan , Arun Malya, Justin Johnson

# Recurrent Neural Network (RNN)

A **recurrent neural network** (**RNN**) is a class of artificial neural networks that deal with **sequential data**.

- RNNs allow us to formulate the learning task in a manner which considers the sequential order of individual observations.
- Example - consider the following two sentences:

I'm sorry... it's not you, it's me.

It's not me, it's you... I'm sorry.

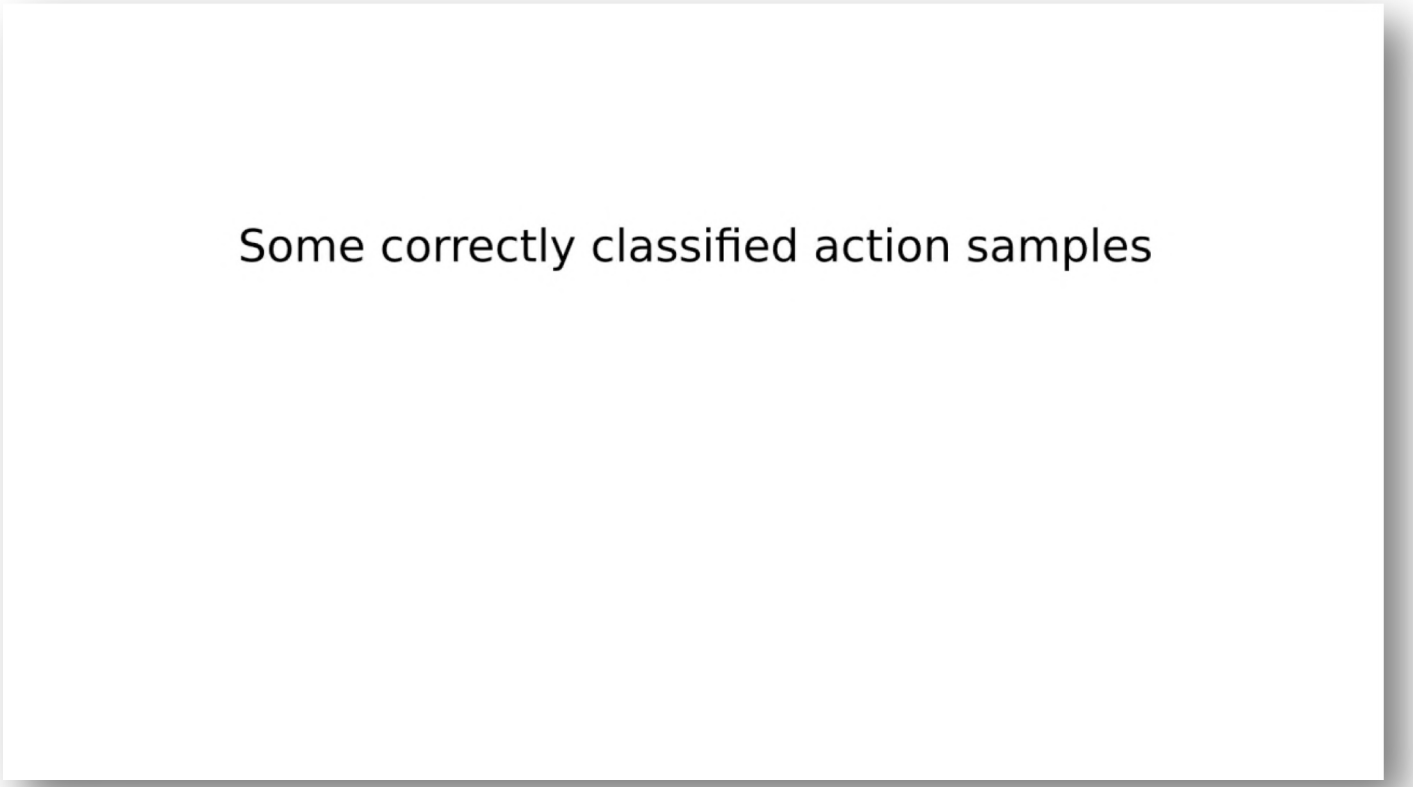
- The two sentences communicate different messages, but this can only be interpreted when considering the sequential order of the words.



Adapted from Jeremy Jordan

# RNN for Video Understanding

Another example: action recognition is video.

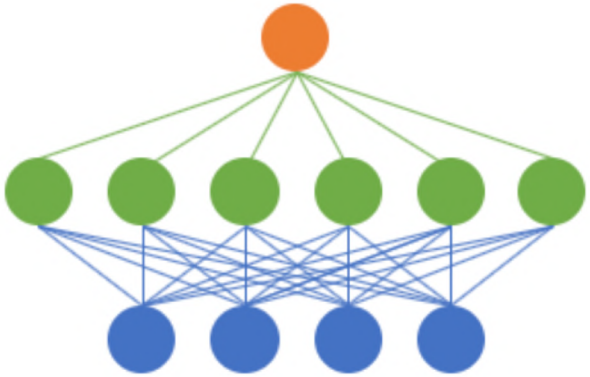


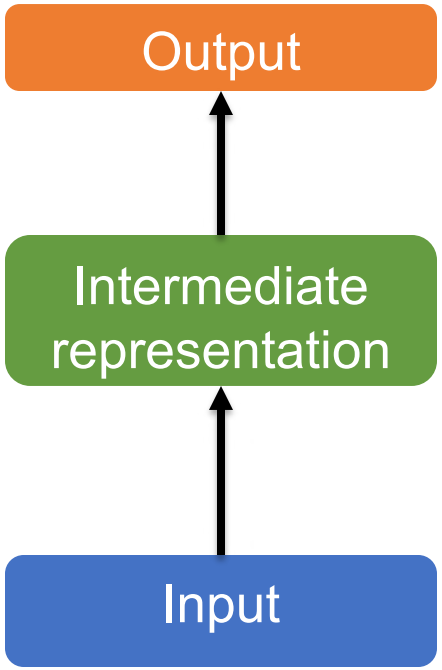



Two-Stream RNN/CNN for Action Recognition in 3D Videos, 2017.

# RNN Intuition

- Recall that DNN performs a series of layer-by-layer transformations in order to embed the input into an ***intermediate representation***.
- This representation makes it easier to solve our task..



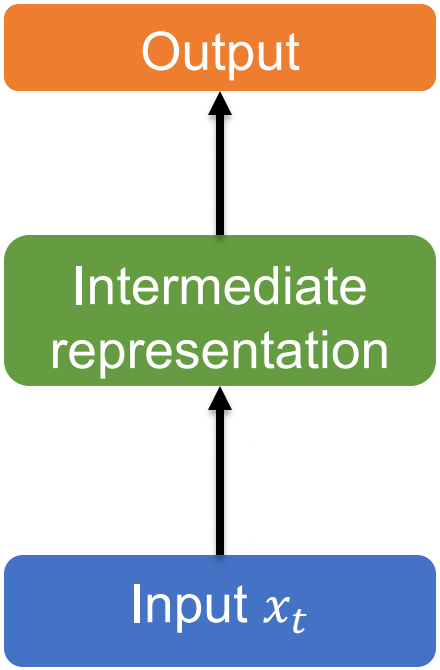





Adapted from Jeremy Jordan

# RNN Intuition

- The same idea applies in a sequential data.
- In the intermediate representation, we want to use information we've already extracted in the previous steps.

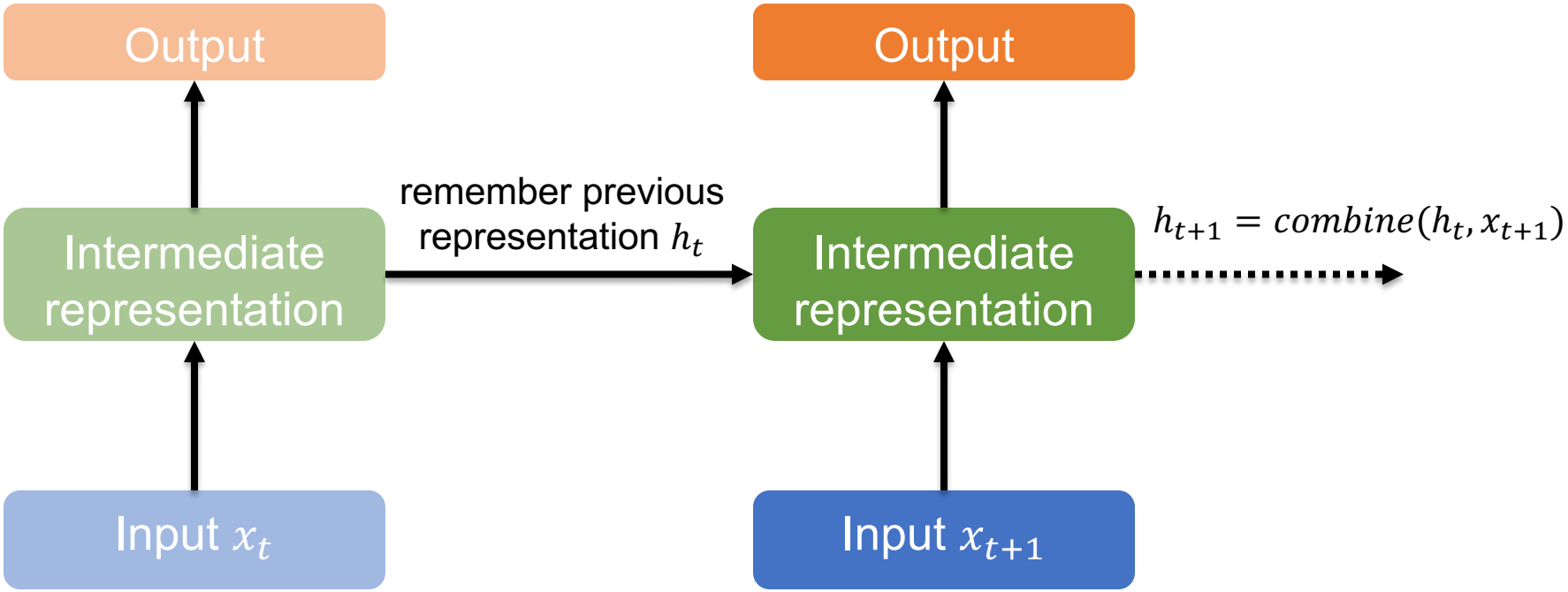





Adapted from Jeremy Jordan

# RNN Intuition

- The same idea applies in a sequential data.
- In the intermediate representation, we want to use information we've already extracted in the previous steps.

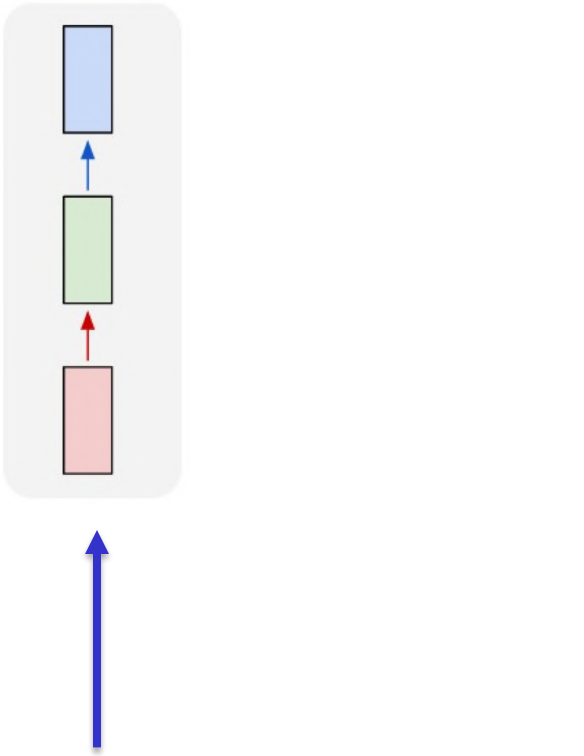





Adapted from Jeremy Jordan

# Common RNN Structures

one to one



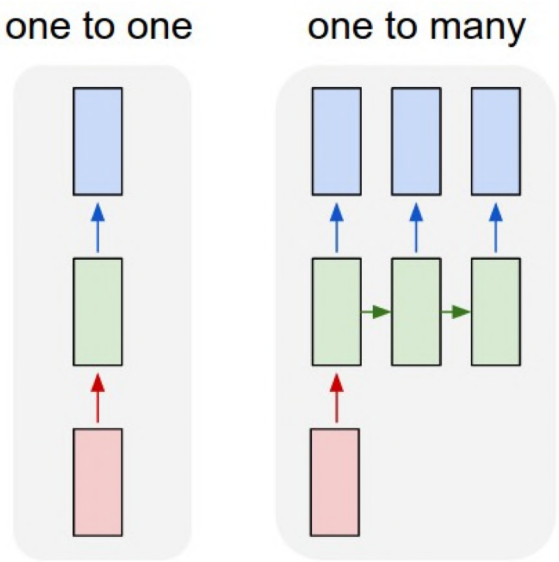
Standard vanilla network

**Image classification**:
image → class



Adapted from Stanford cs231n

# Common RNN Structures

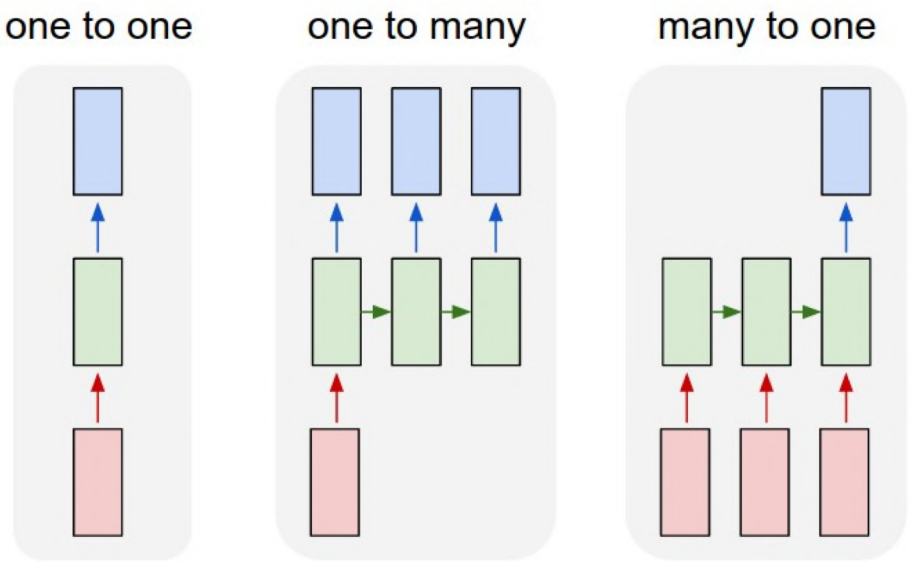



One to many
**Image Captioning**:
image → sequence of words



Adapted from Stanford cs231n

# Common RNN Structures

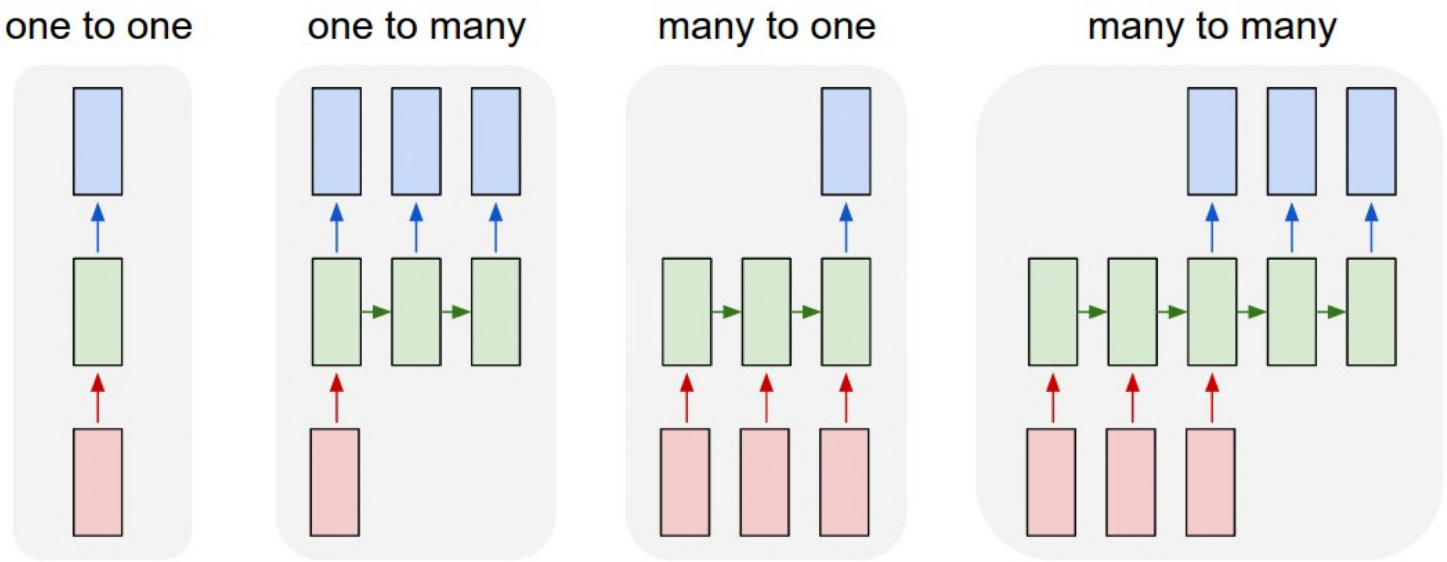



Many to one
**Sentiment Classification:**
sequence of words → sentiment



Adapted from Stanford cs231n

# Common RNN Structures

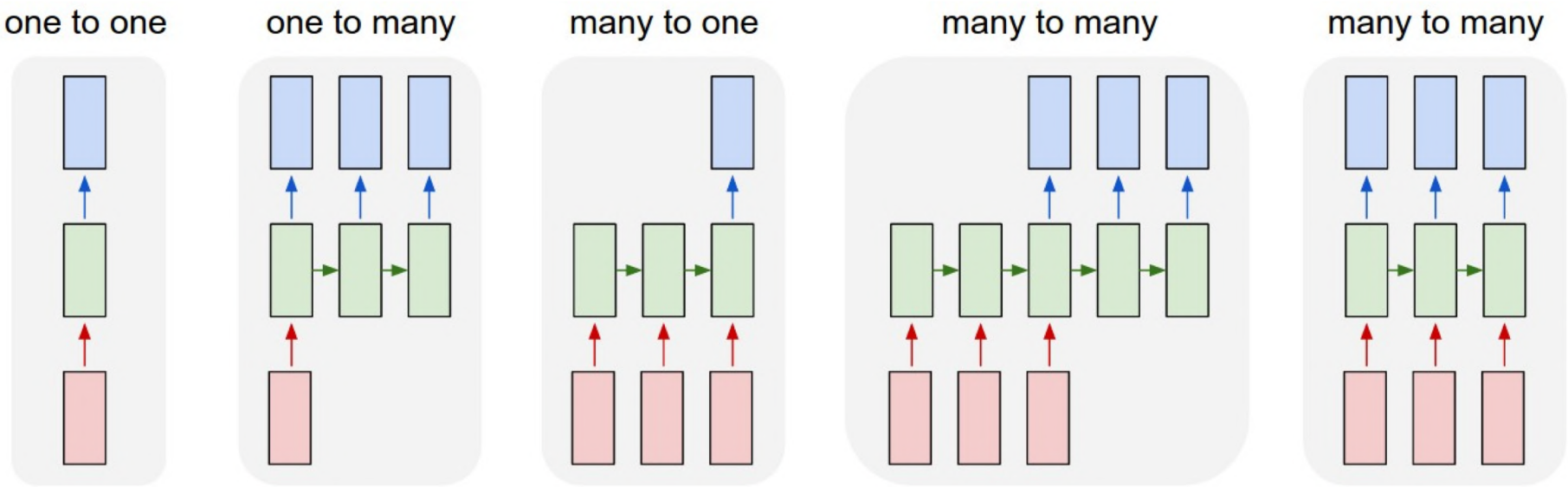



Many to many
**Machine Translation:**
seq of words → seq of words



Adapted from Stanford cs231n

# Common RNN Structures

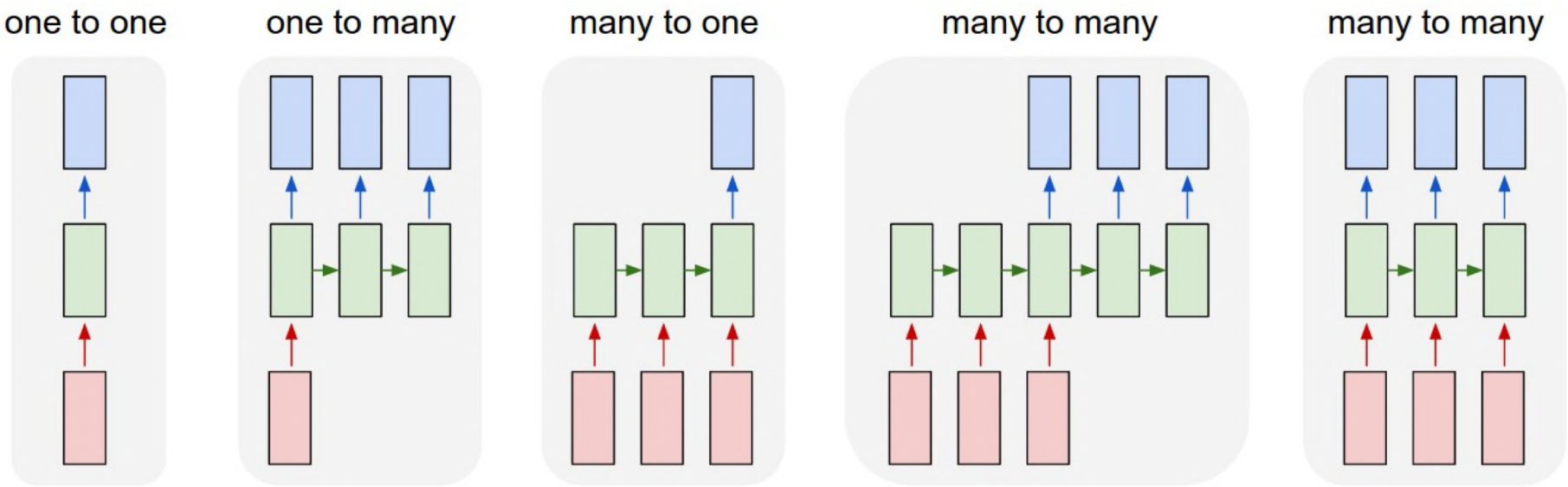



Many to many
**Video Frame Classification:**
video frame → class



Adapted from Stanford cs231n

# Common RNN Structures



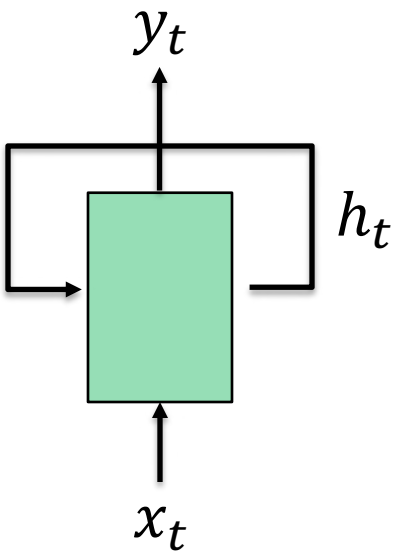

- One of the benefits of recurrent neural networks is the ability to **handle arbitrary length inputs and outputs**.
- This flexibility allows us to define a broad range of tasks.



Adapted from Stanford cs231n

# Recurrent Neural Network

- Key idea: RNN has an "**internal state**" that is updated at each time state.
- The state consists of of a single "hidden" vector $h_t$ :



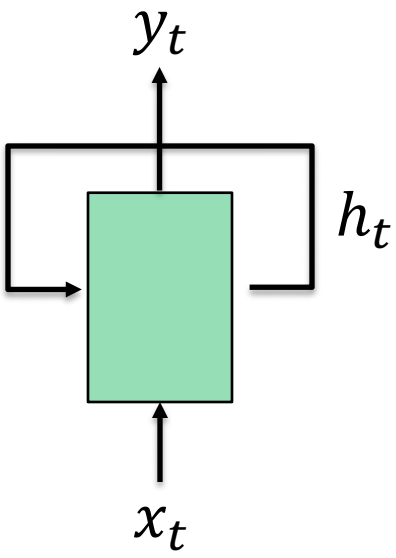

$$h_t = f_W(x_t, h_{t-1})$$

New state

Source: Justin Johnson

# Recurrent Neural Network

- Key idea: RNN has an "internal state" that is updated at each time state.
- The state consists of of a single "hidden" vector $h_t$ :




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

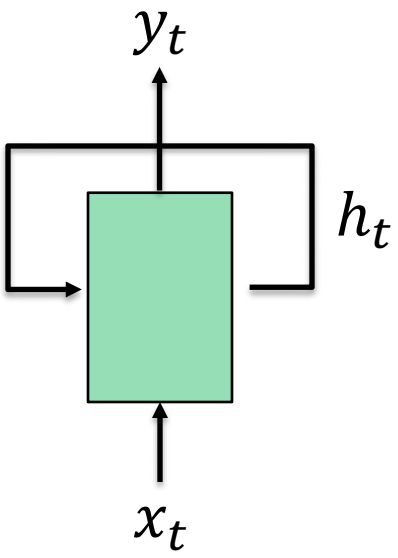
Some function with parameter $W$

Note, $W$ is time independent

Source: Justin Johnson

# Recurrent Neural Network

- Key idea: RNN has an "internal state" that is updated at each time state.
- The state consists of of a single "hidden" vector $h_t$ :



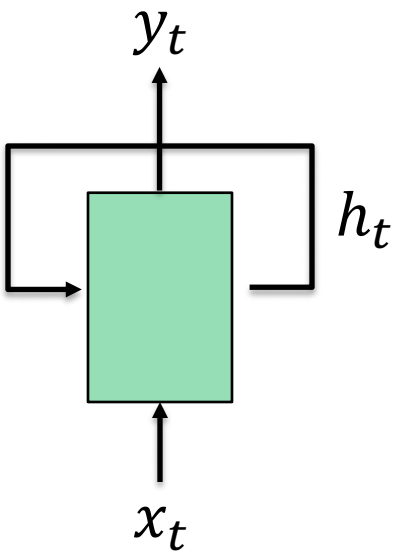

$$h_t = f_W(x_t, h_{t-1})$$

Input vector at time $t$

Source: Justin Johnson

# Recurrent Neural Network

- Key idea: RNN has an “internal state” that is updated at each time state.
- The state consists of of a single “hidden” vector $h_t$ :



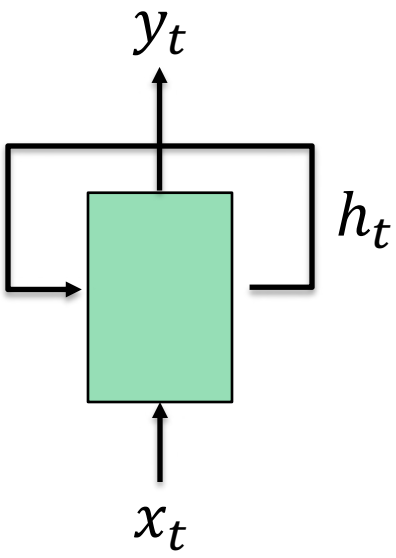

$$h_t = f_W(x_t, h_{t-1})$$

Old state

Source: Justin Johnson

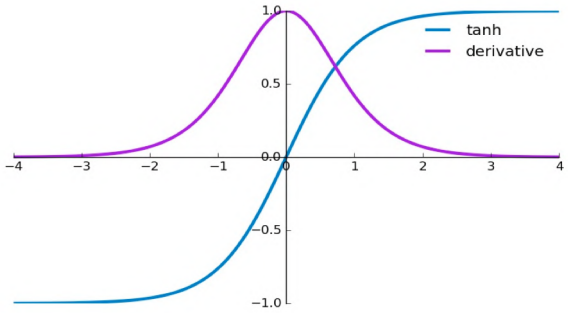
# Recurrent Neural Network

- Key idea: RNN has an “internal state” that is updated at each time state.
- The state consists of of a single “hidden” vector $h_t$ :




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

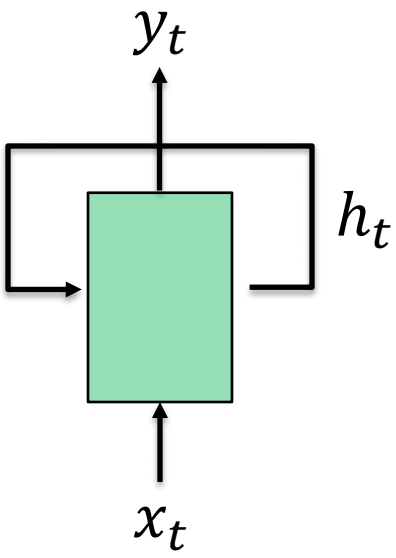
$$= \tanh(W_{xh}x_t + W_{hh}h_{t-1})$$




Source: Justin Johnson

# Recurrent Neural Network

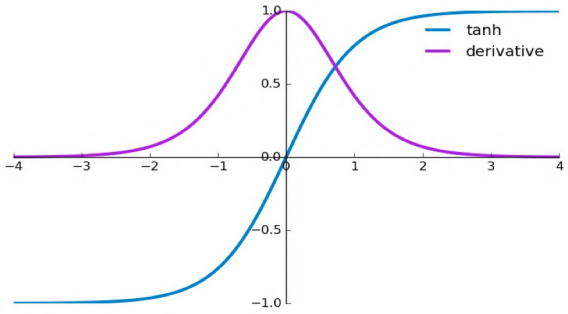
- Key idea: RNN has an "internal state" that is updated at each time state.
- The state consists of of a single "hidden" vector $h_t$ :

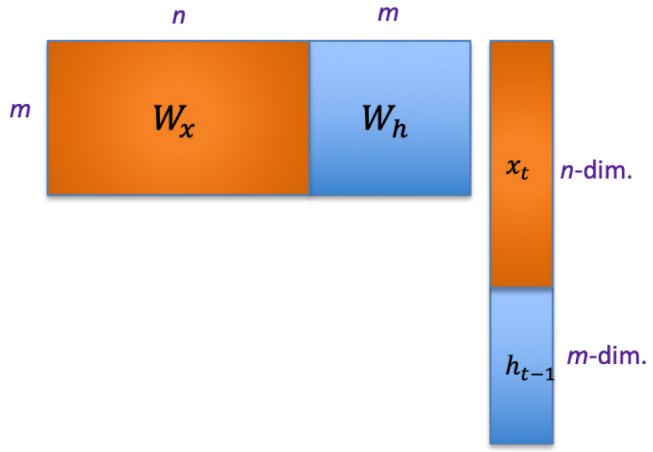



$$h_t = f_W(x_t, h_{t-1})$$

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

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

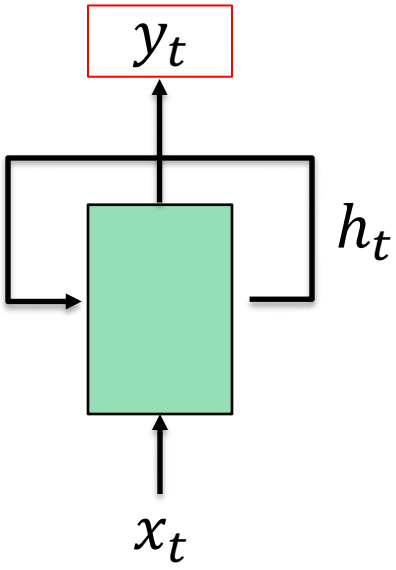



Source: Justin Johnson

# Recurrent Neural Network

- Key idea: RNN has an "internal state" that is updated at each time state.
- The state consists of of a single "hidden" vector $h_t$
- Each step can have an output $y_t$:




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

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

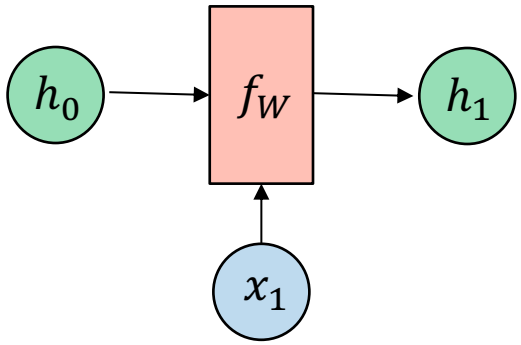
$$= \tanh\left(W \begin{bmatrix} x_t \\ h_{t-1} \end{bmatrix}\right)$$

$$y_t = \tanh(W_{hy}h_t)$$

Source: Justin Johnson

# RNN Computational Graph

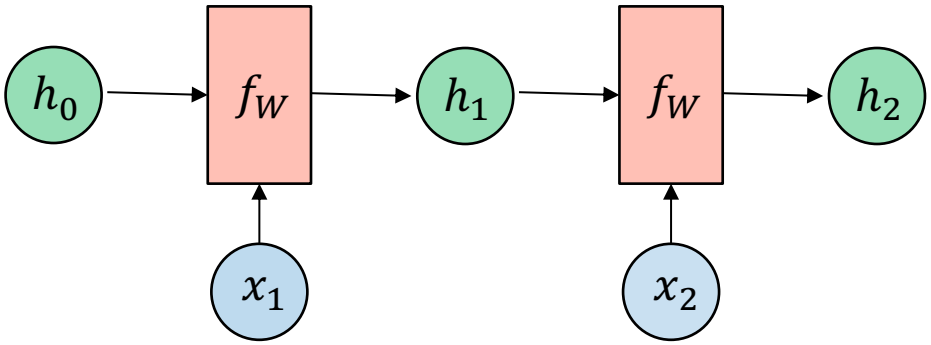
- A recurrent neural network can be thought of as multiple copies of the same network.
- We can roll the copies of the RNN into a single network.






Source: Justin Johnson

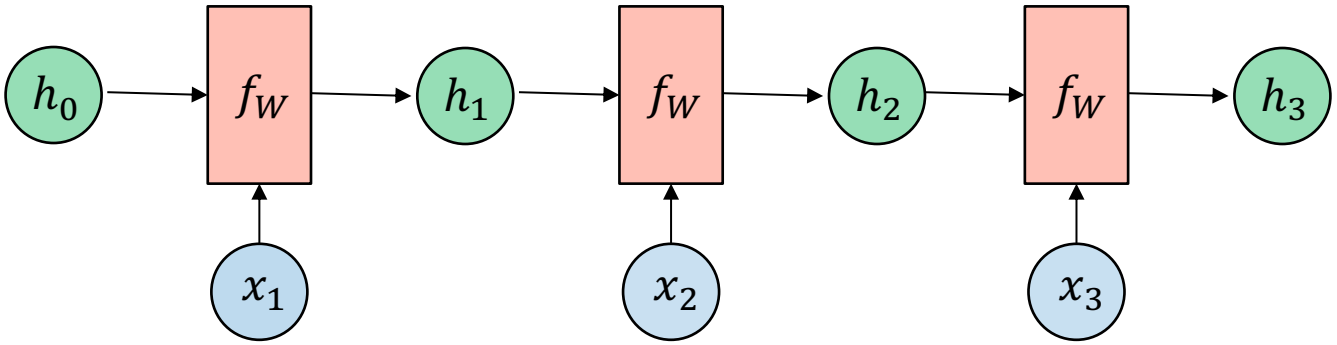
# RNN Computational Graph






Source: Justin Johnson

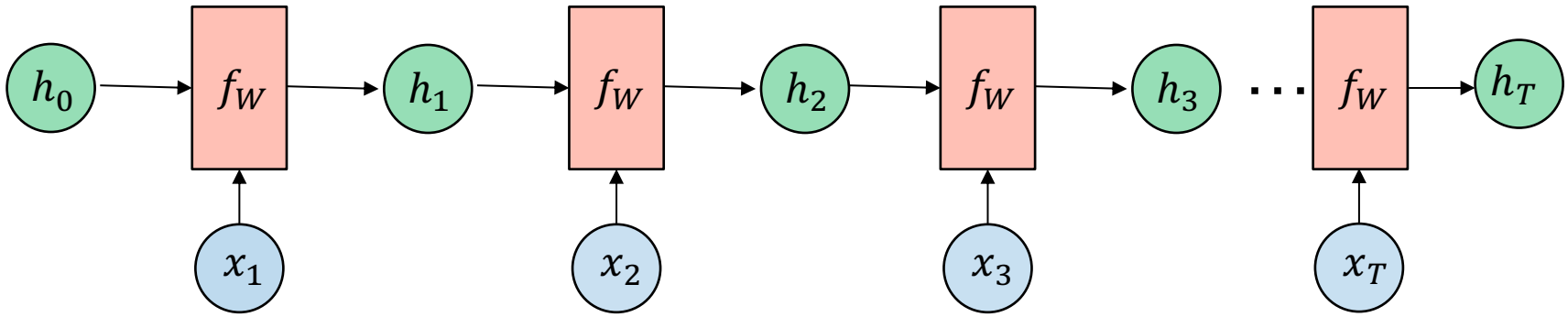
# RNN Computational Graph






Source: Justin Johnson

# RNN Computational Graph

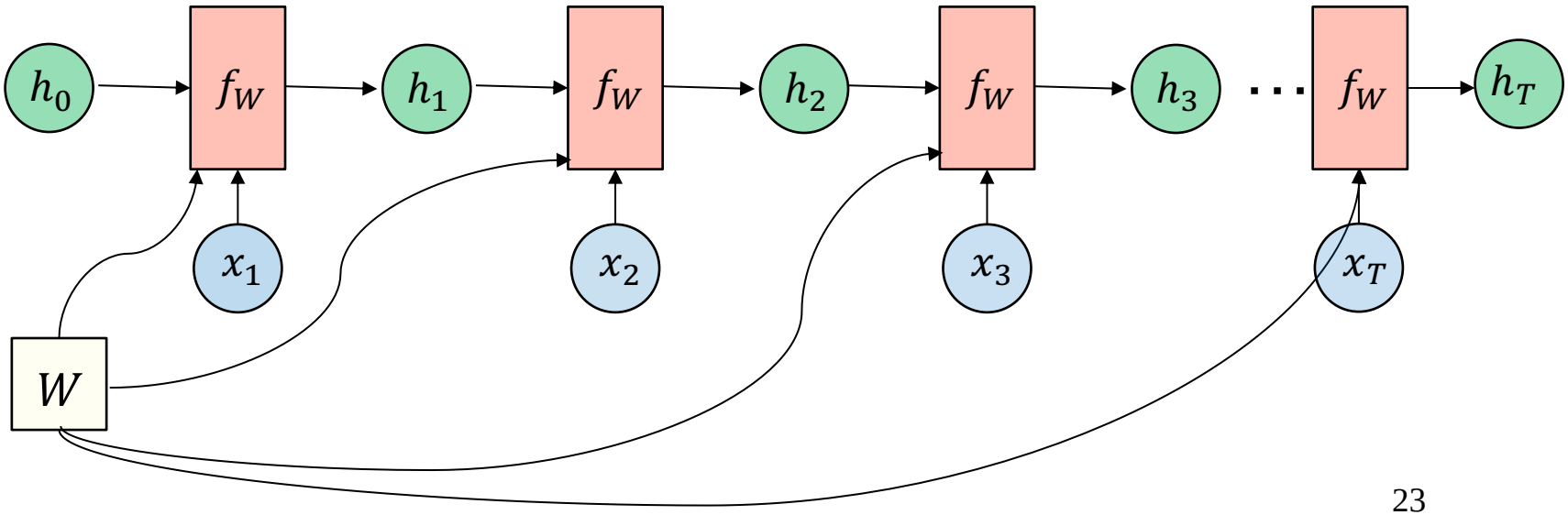





Source: Justin Johnson

# RNN Computational Graph

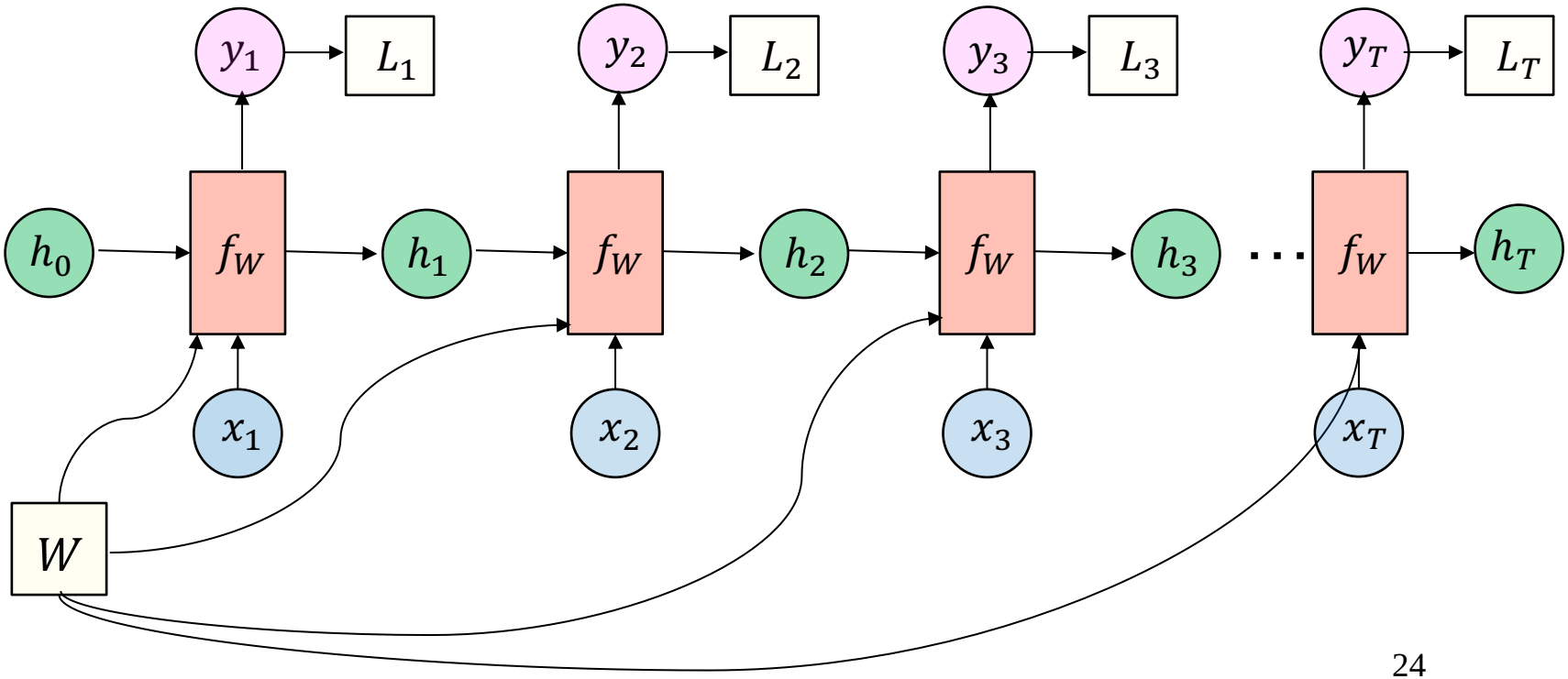
- The weight matrix is identical for all time-steps
- What if the computation graph has fan-out > 1?






Source: Justin Johnson

# RNN Computational Graph (many-to-many)

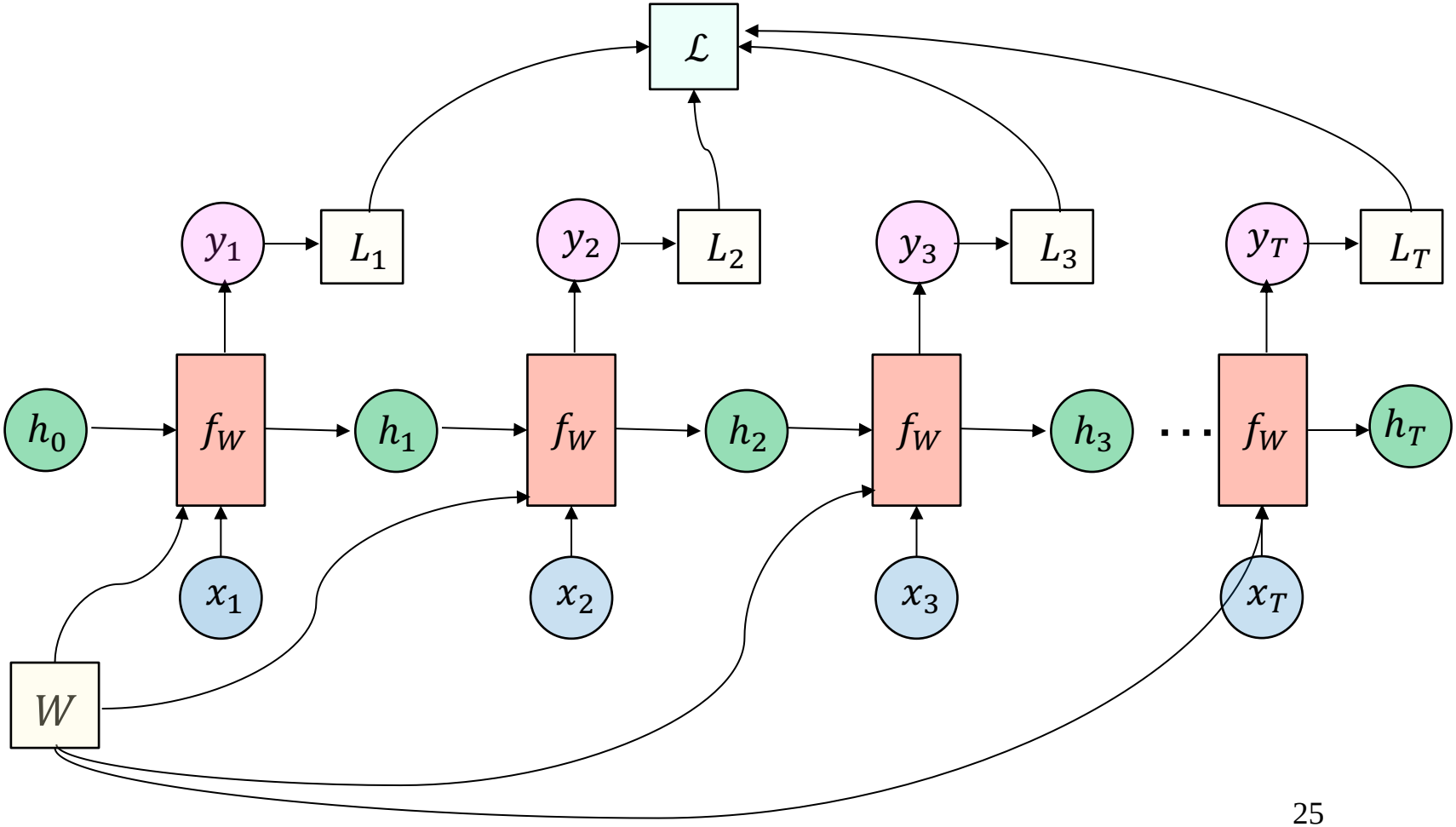





Source: Justin Johnson

# RNN Computational Graph (many-to-many)

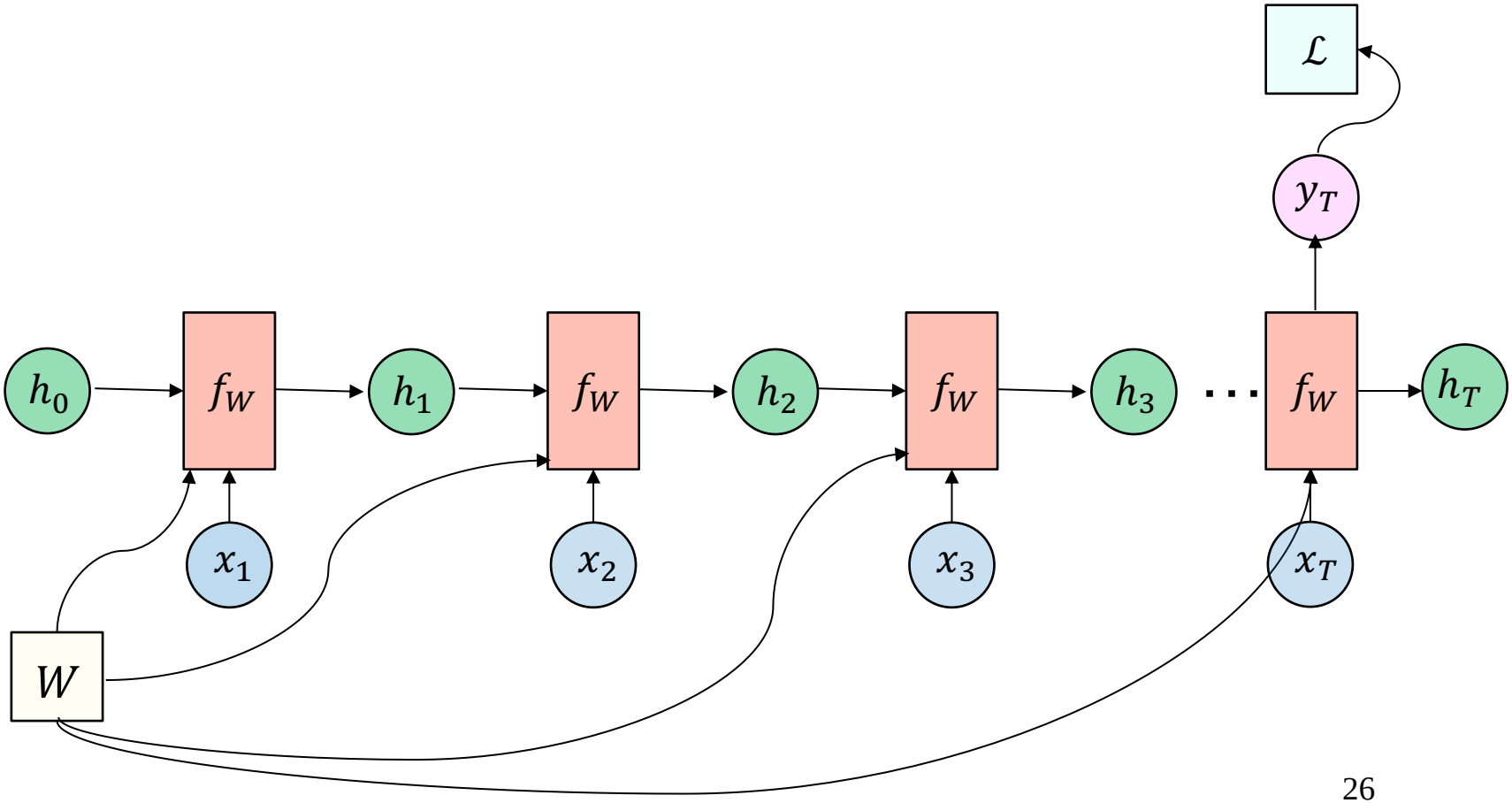
- The total loss is a sum of all step-loss






Source: Justin Johnson

# RNN Computational Graph (many-to-one)

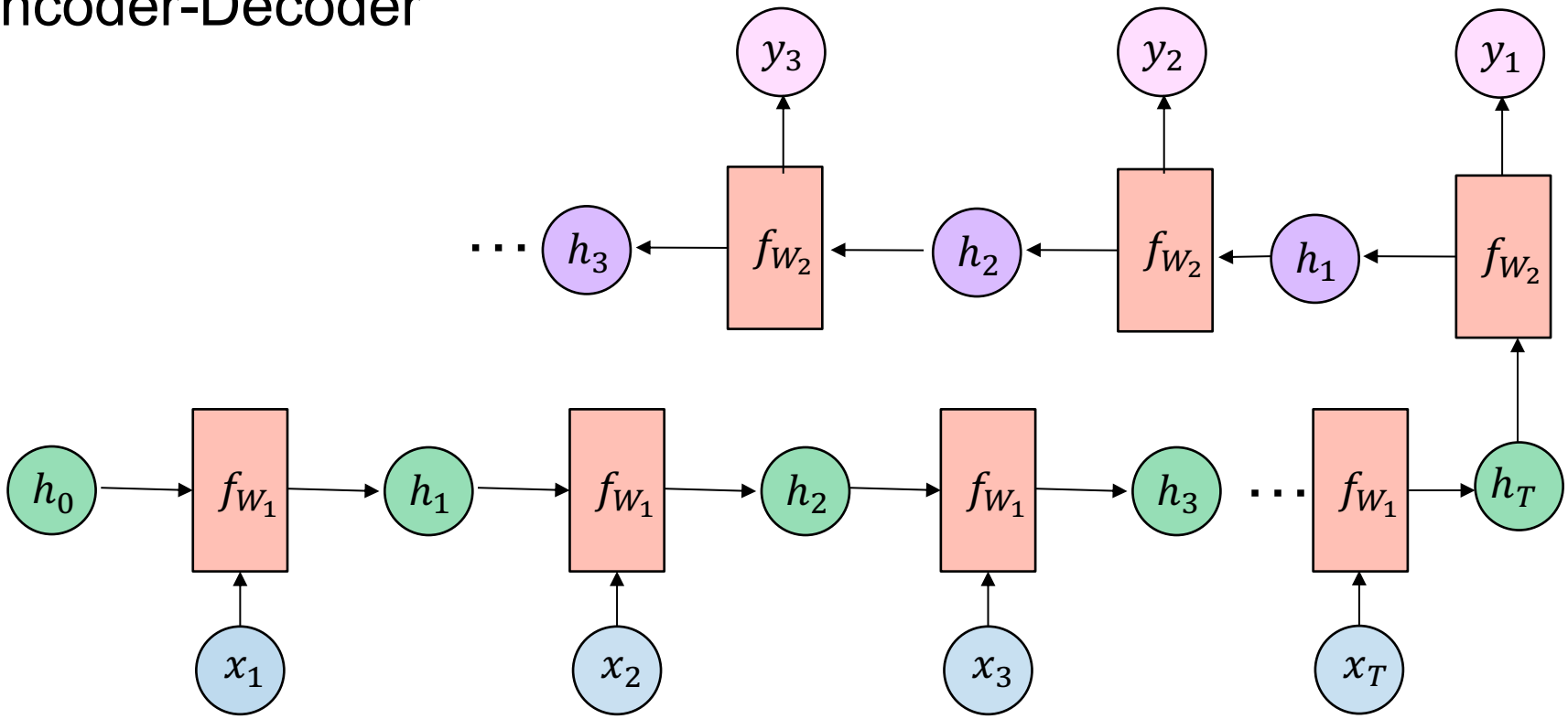





Source: Justin Johnson

# RNN Sequence to sequence (seq. to seq.)

- (many-to-one)+(one-to-many)
- Encoder-Decoder

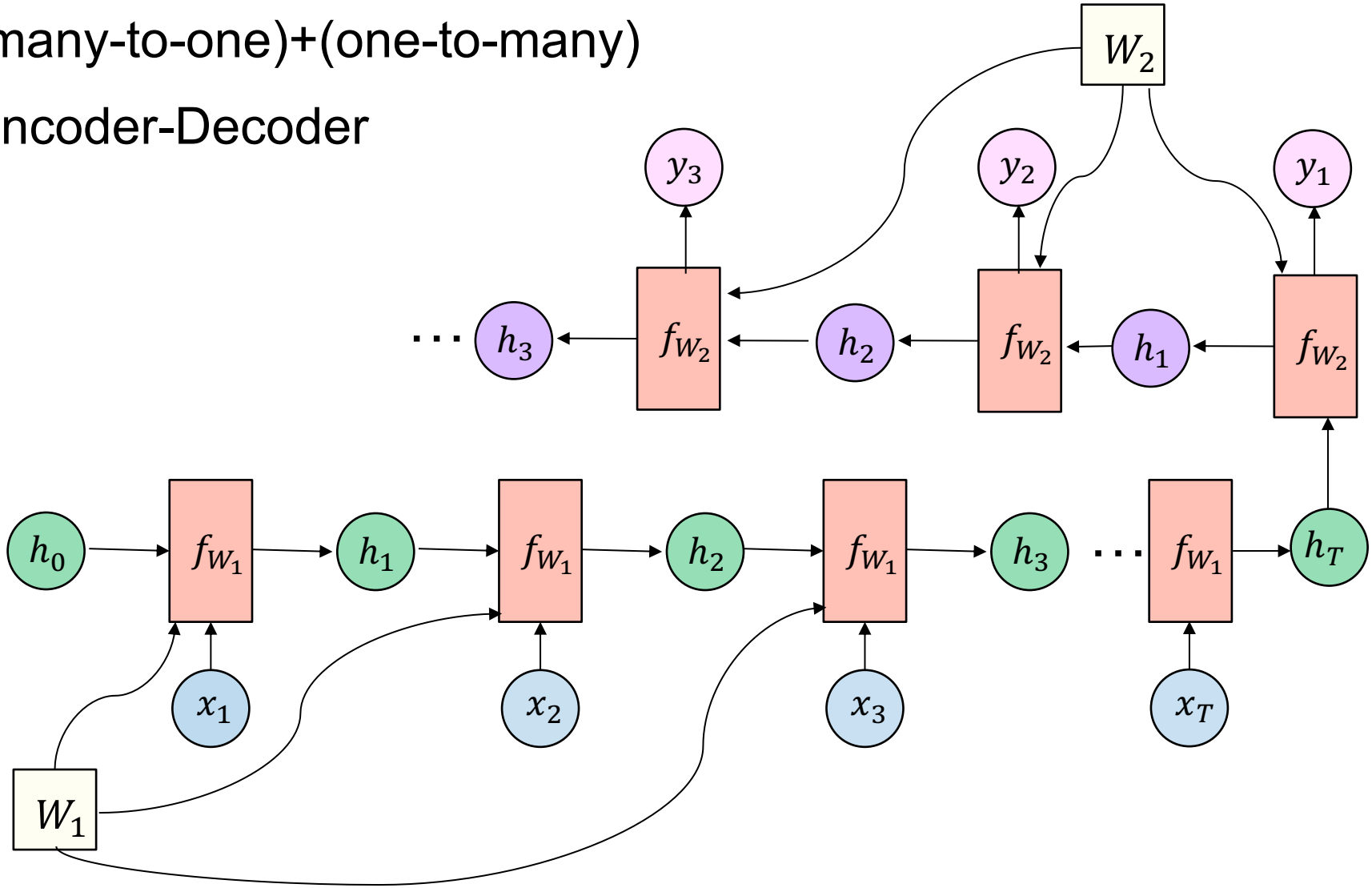





Source: Justin Johnson

# RNN Sequence to sequence (seq. to seq.)

- (many-to-one)+(one-to-many)
- Encoder-Decoder






Source: Justin Johnson

# RNN Example: Sentiment Analysis

Example: **Sentiment classification**

**Input:** A written review about restaurant or movie

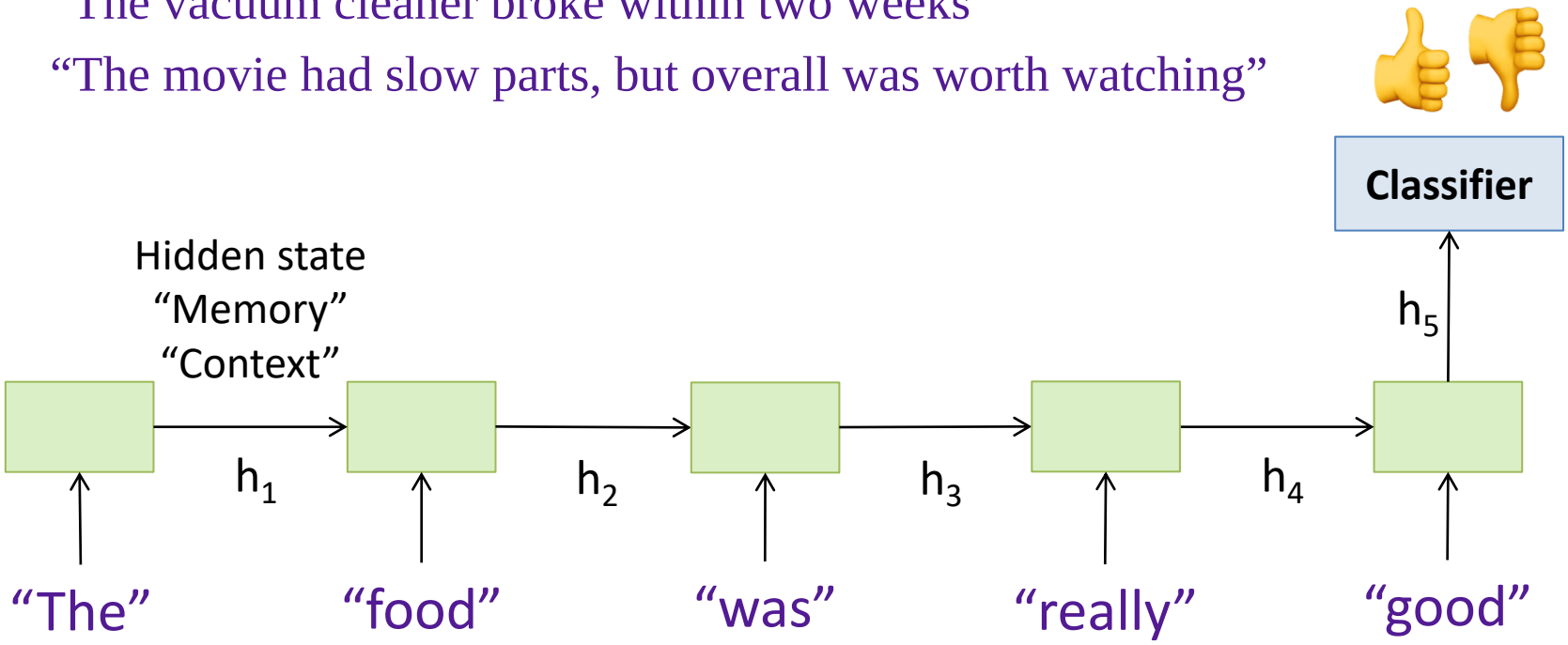
**Output**: Recommendation

Examples:

"The food was really good"

"The vacuum cleaner broke within two weeks"

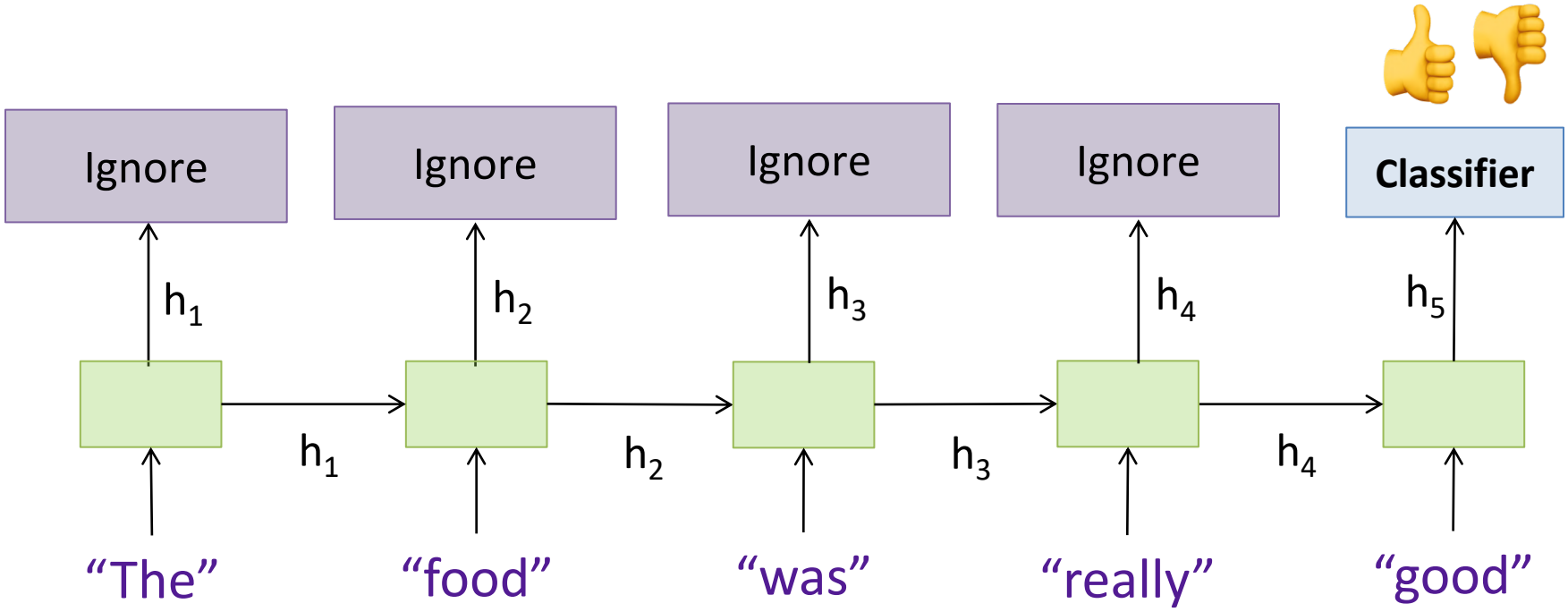
"The movie had slow parts, but overall was worth watching"



# RNN Example: Sentiment Analysis

Example: **Sentiment classification**

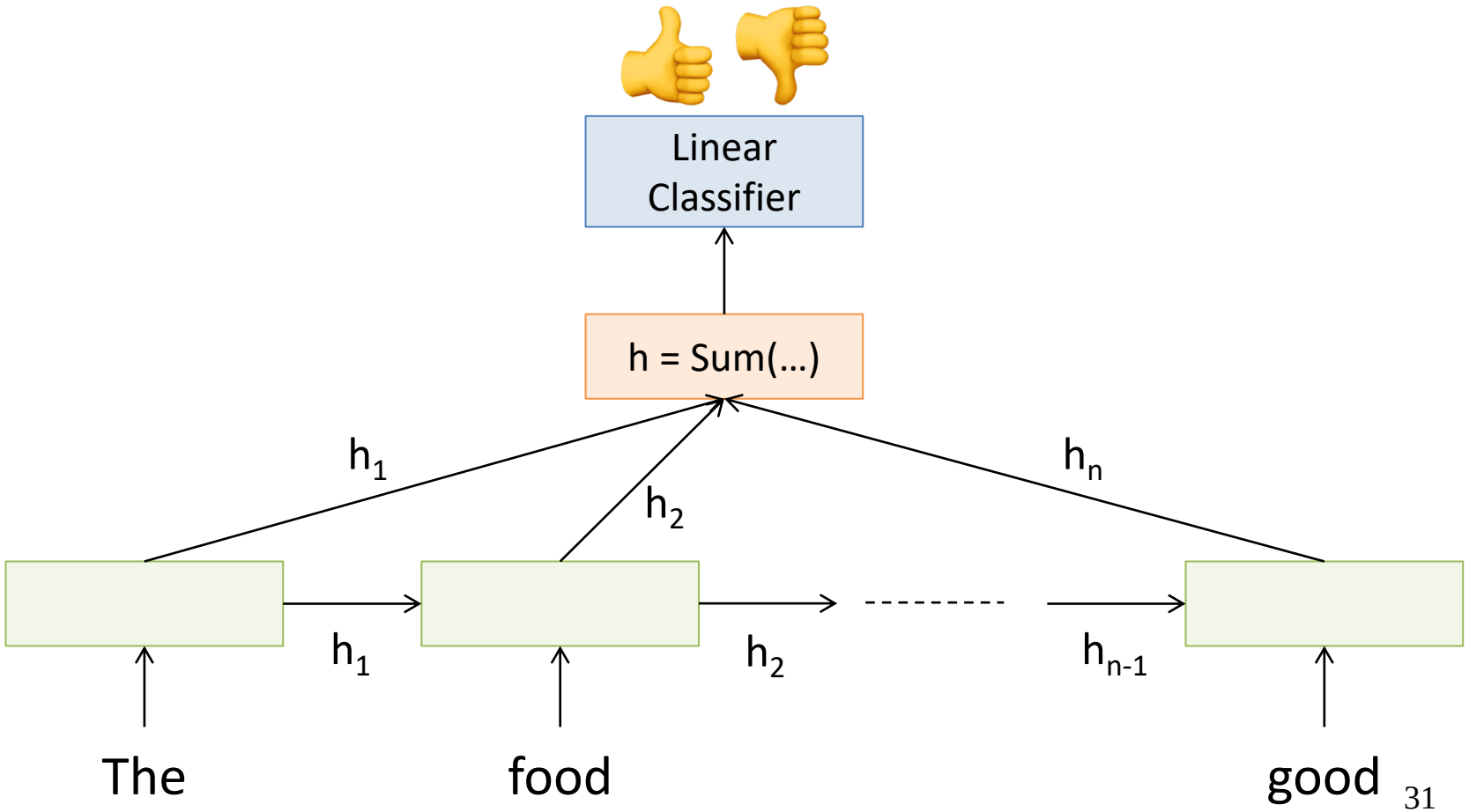
- Many-to-one, ignore all outputs but the last one.
- Word inputs are one-hot vectors or word2Vec
- How do we know we reached the last word?



# RNN Example: Sentiment Analysis

Example: **Sentiment classification**

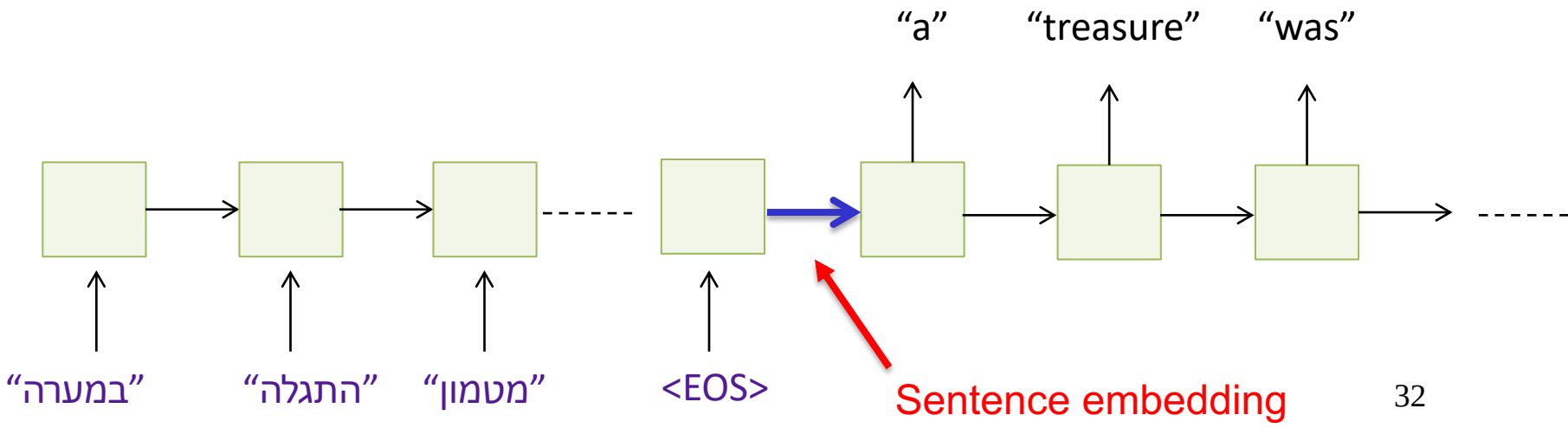
Another possibility of the outputs:



# RNN Example: Seq. to Seq.

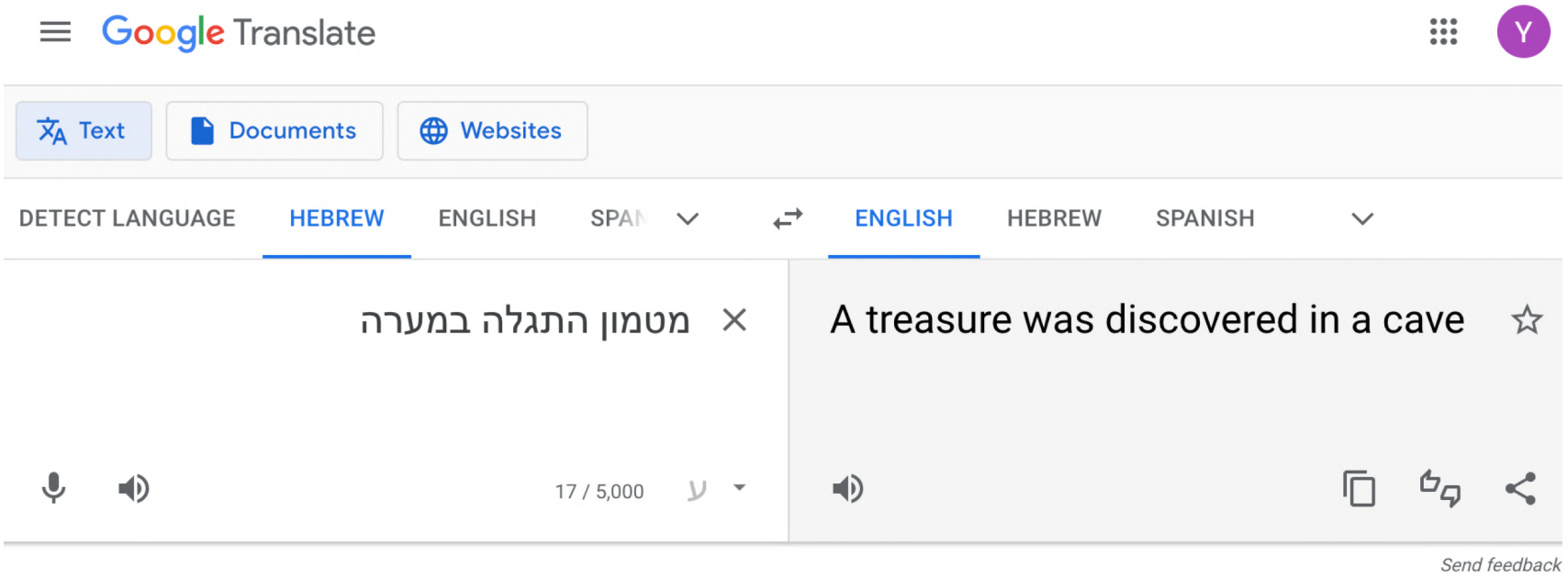
Example: **Machine Translation**

- Multiple input – multiple output (sequence-to-sequence)
- Word inputs are **one-hot vectors** or **word2Vec** of a given word vocabulary
- Word outputs are probability vectors into the vocabulary
- Termed an **encoder-decoder** RNN or **seq-to-seq**



# RNN Example: Seq. to Seq.

- Example: **Machine Translation**



# RNN Example: Language Modeling

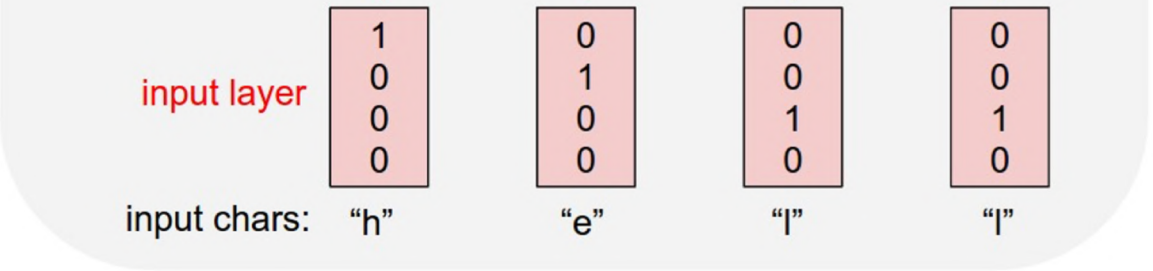
Example: **Language Modeling – Char RNN**

**Input:** Characters $c_1, c_2, \ldots, c_{t-1}$

**Output:** predict $c_t$

Training sequence: “hello”

Vocabulary: [h,e,l,o]




Source: Justin Johnson

# RNN Example: Language Modeling

Example: **Language Modeling – Char RNN**

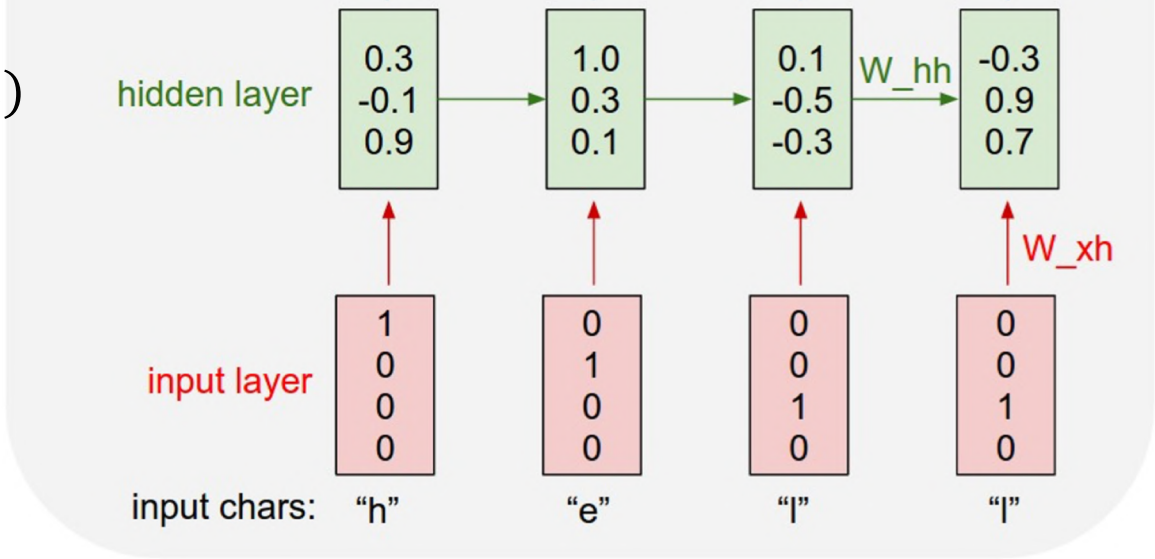
**Input:** Characters $c_1, c_2, \ldots, c_{t-1}$

**Output:** predict $c_t$

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

Training sequence: “hello”

Vocabulary: [h,e,l,o]




Source: Justin Johnson

# RNN Example: Language Modeling

Example: **Language Modeling – Char RNN**

**Input:** Characters $c_1, c_2, \ldots, c_{t-1}$

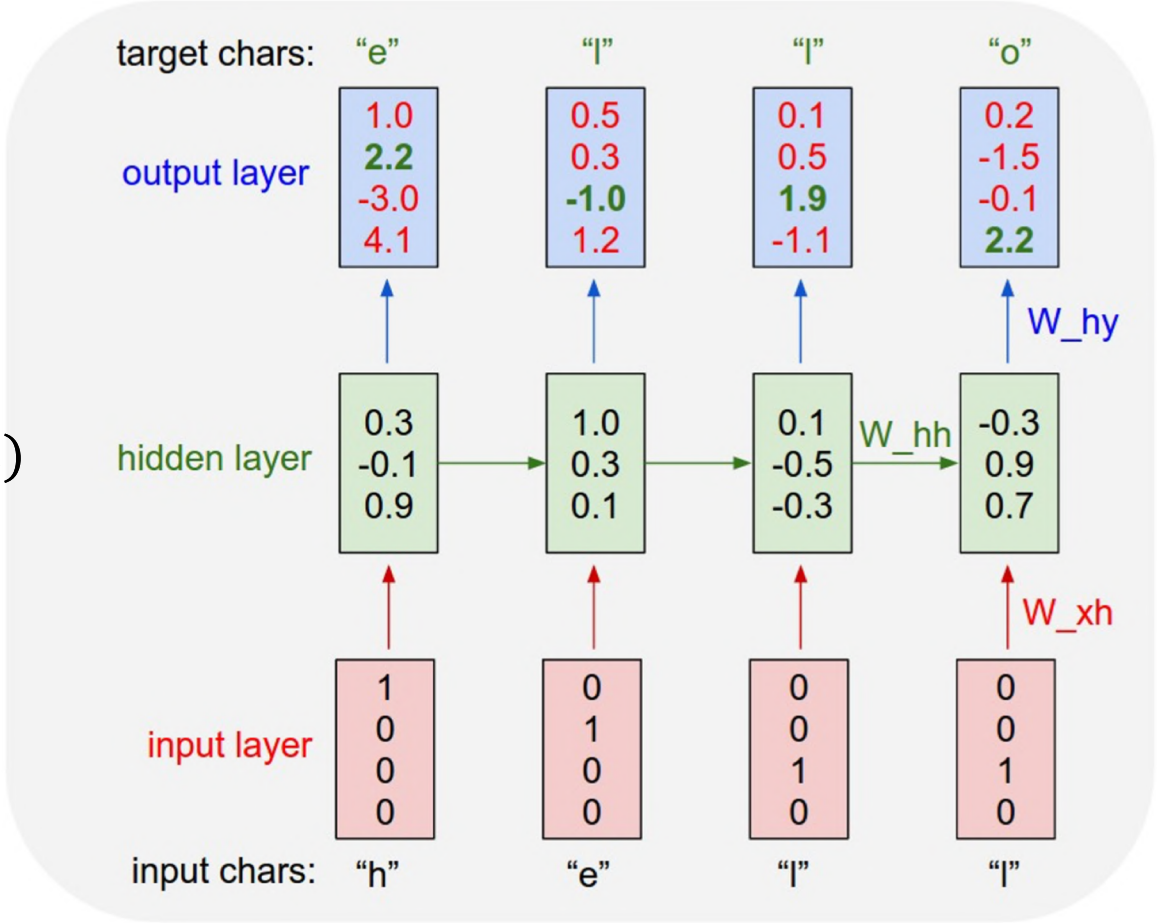
**Output:** predict $c_t$

$$y_t = \tanh(W_{hy} h_t)$$

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

Training sequence: “hello”

Vocabulary: [h,e,l,o]



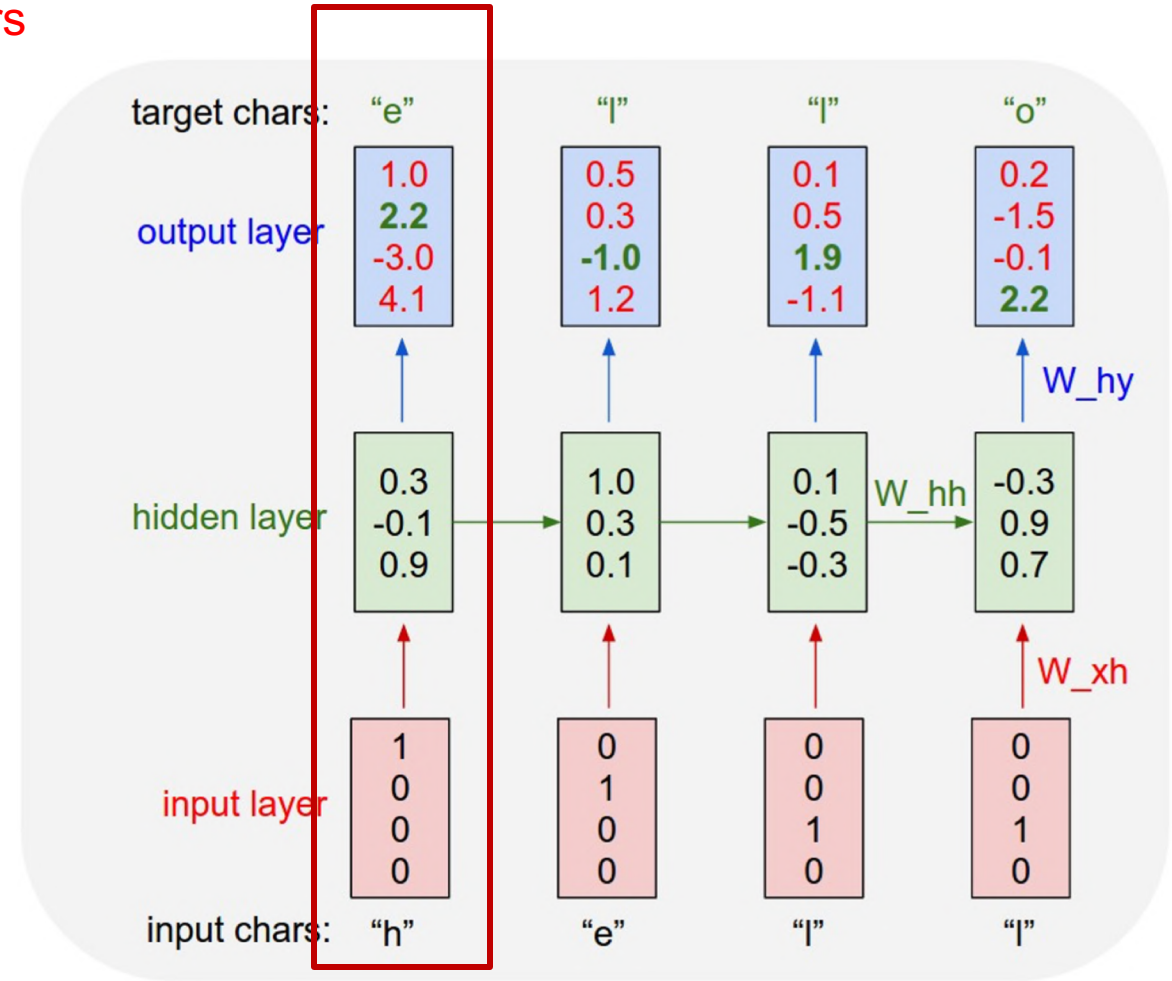

Source: Justin Johnson

# RNN Example: Language Modeling

**Training phase**:

We want the green numbers to be high and red numbers to be low.

Given “h” predict “e”



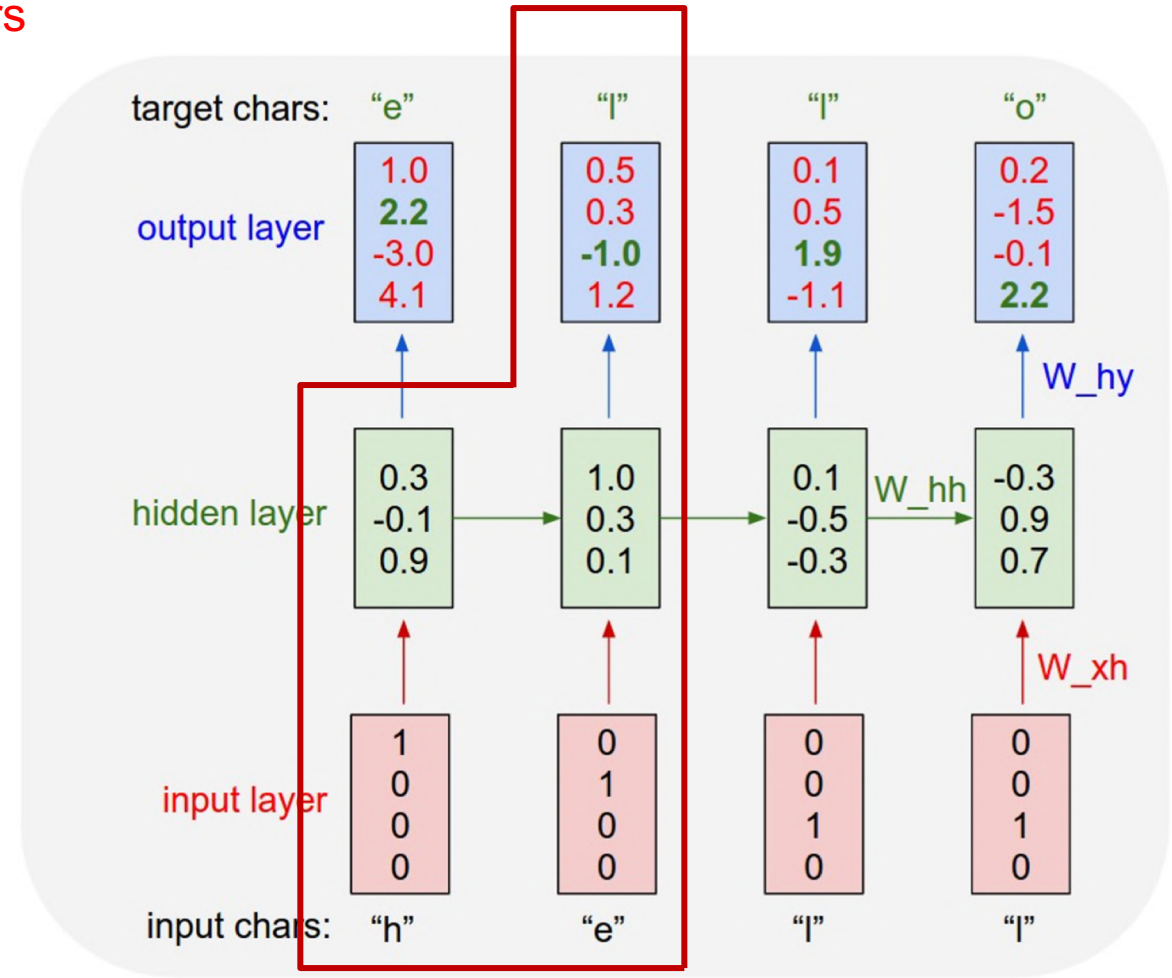

Source: Justin Johnson

# RNN Example: Language Modeling

**Training phase**:

We want the green numbers to be high and red numbers to be low.

Given “he” predict “l”



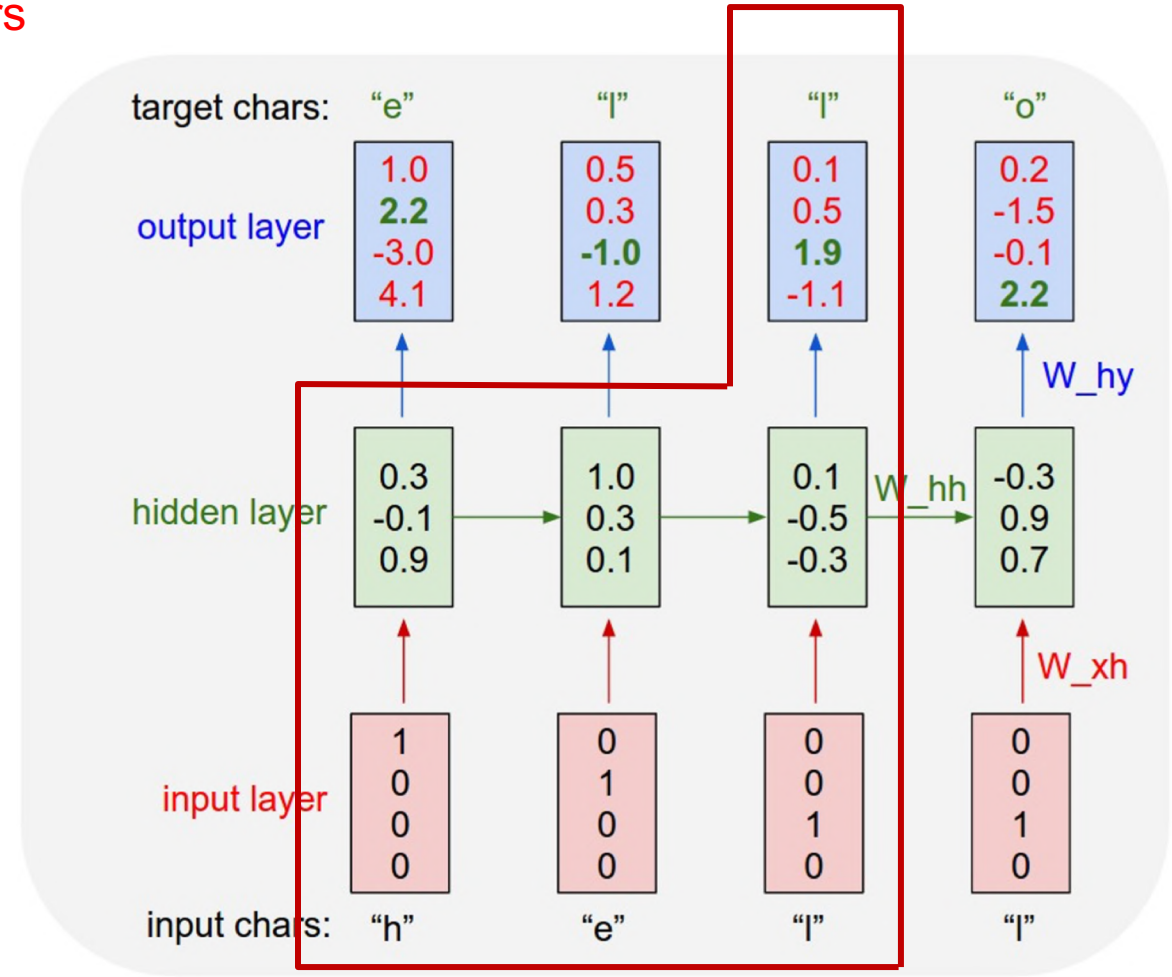

Source: Justin Johnson

# RNN Example: Language Modeling

**Training phase**:

We want the green numbers to be high and red numbers to be low.

Given “hel” predict “l”



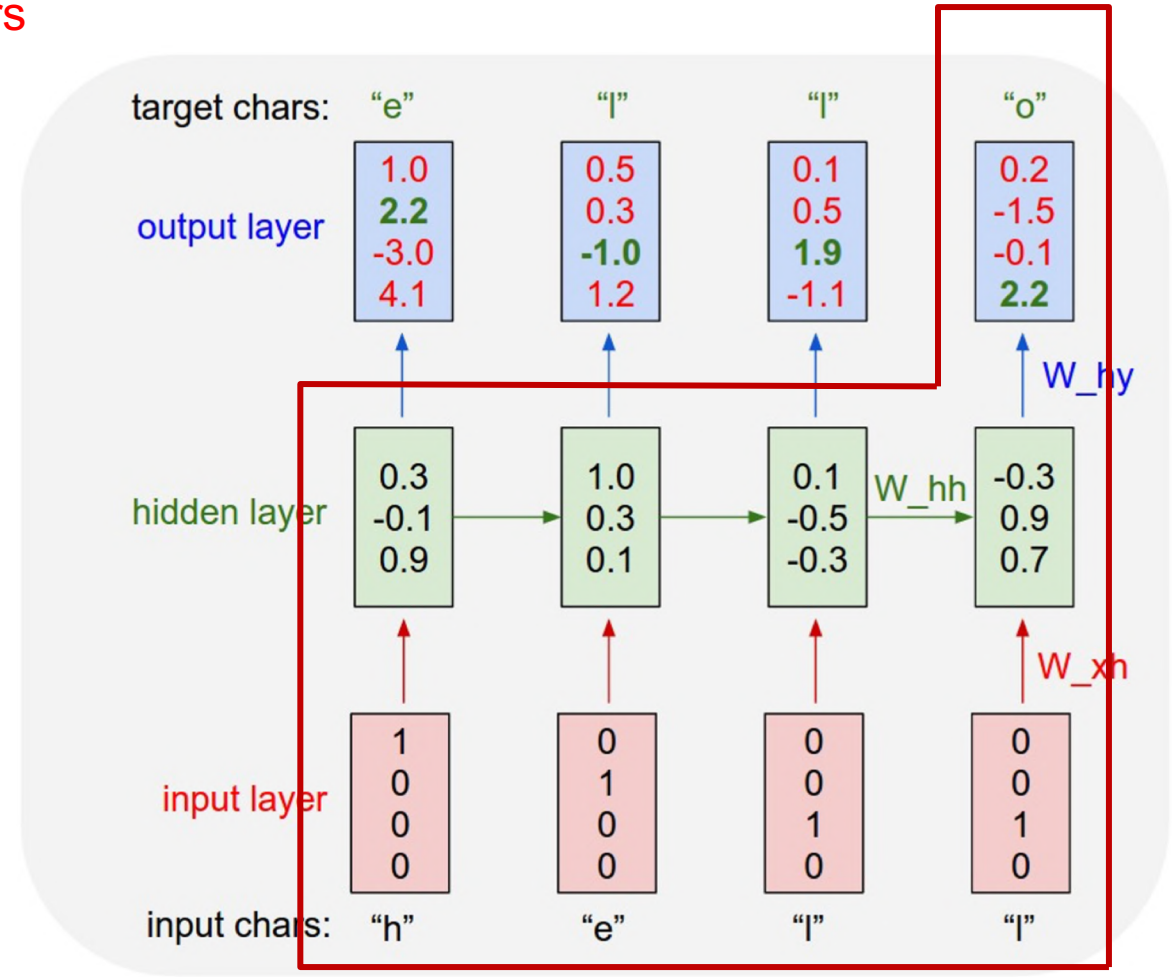

Source: Justin Johnson

# RNN Example: Language Modeling

**Training phase**:

We want the green numbers to be high and red numbers to be low.

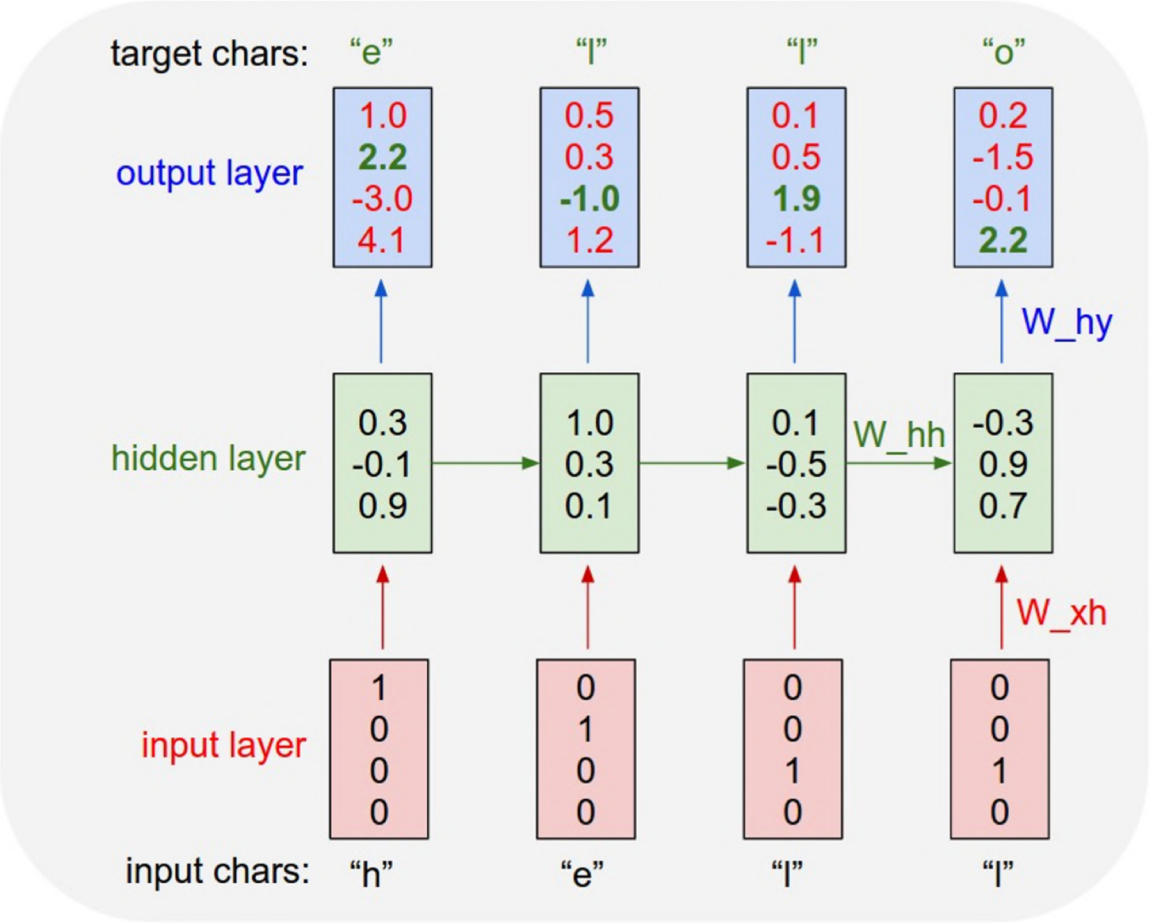
Given “hell” predict “o”




Source: Justin Johnson

# RNN Example: Language Modeling

$$p(y_1, y_2, \ldots, y_n)$$
$$= p(y_1)p(y_2|y_1)p(y_3|y_1, y_2) \ldots$$
$$= \prod_{i=1}^{n} p(y_i|y_1, \ldots, y_{i-1})$$
$$\approx \prod_{i=1}^{n} P_W(y_i|h_i)$$

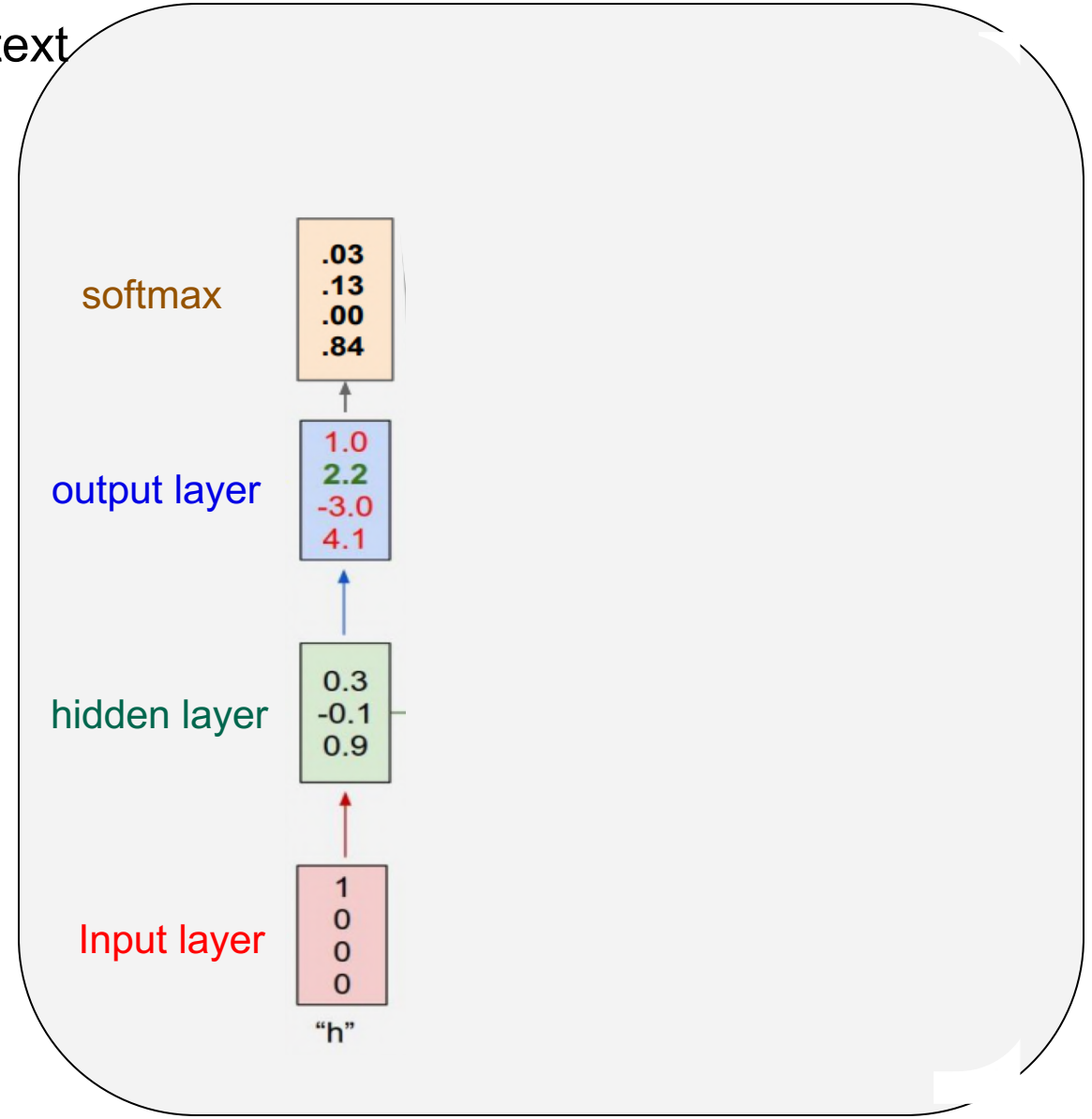



Source: Justin Johnson

# RNN Example: Language Modeling

**At test time**: Generate new text

- One char at a time
- feed back to the model



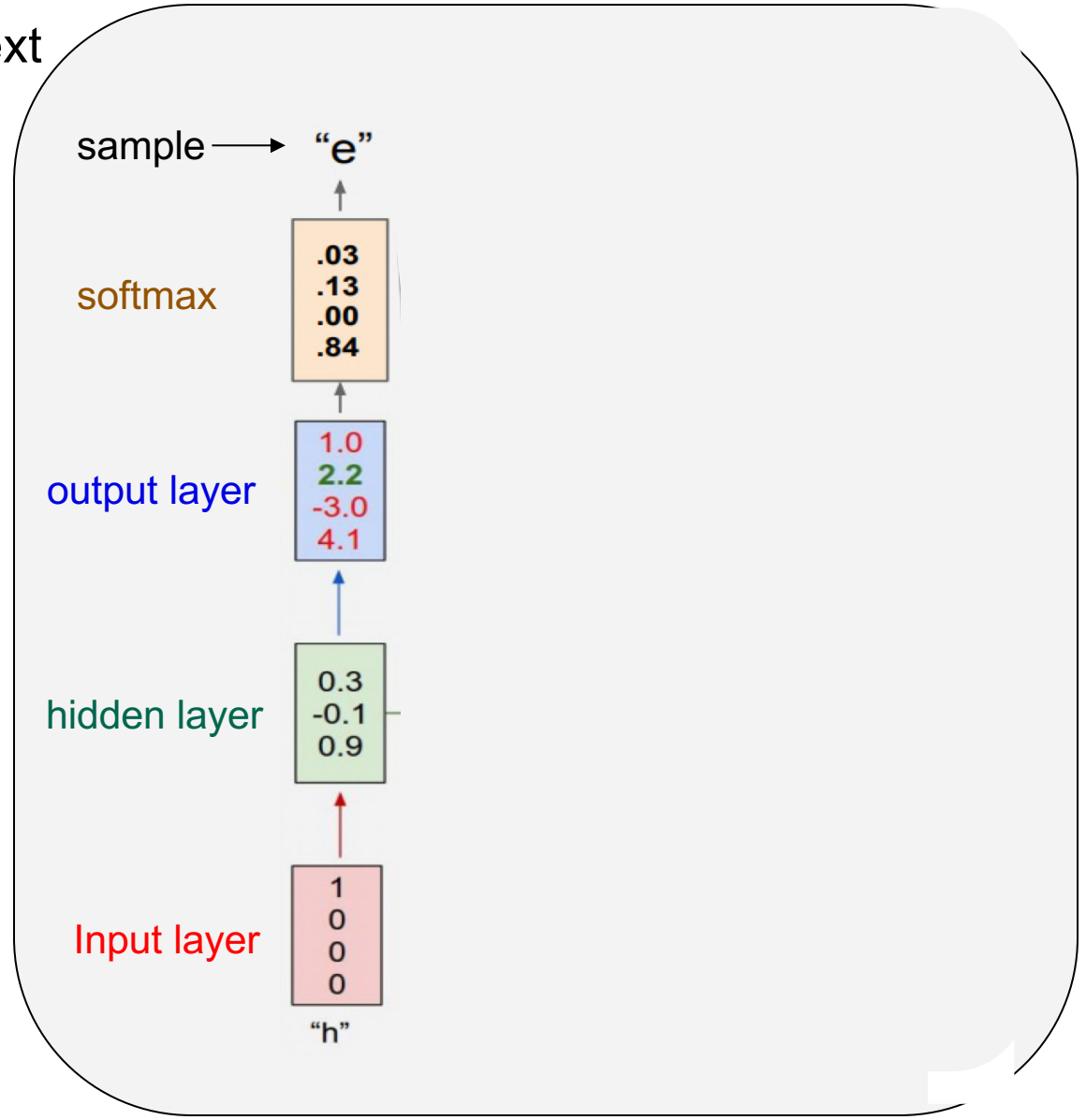

Vocabulary: [h,e,l,o]

Source: Justin Johnson

# RNN Example: Language Modeling

At test time: Generate new text

- One char at a time
- feed back to the model



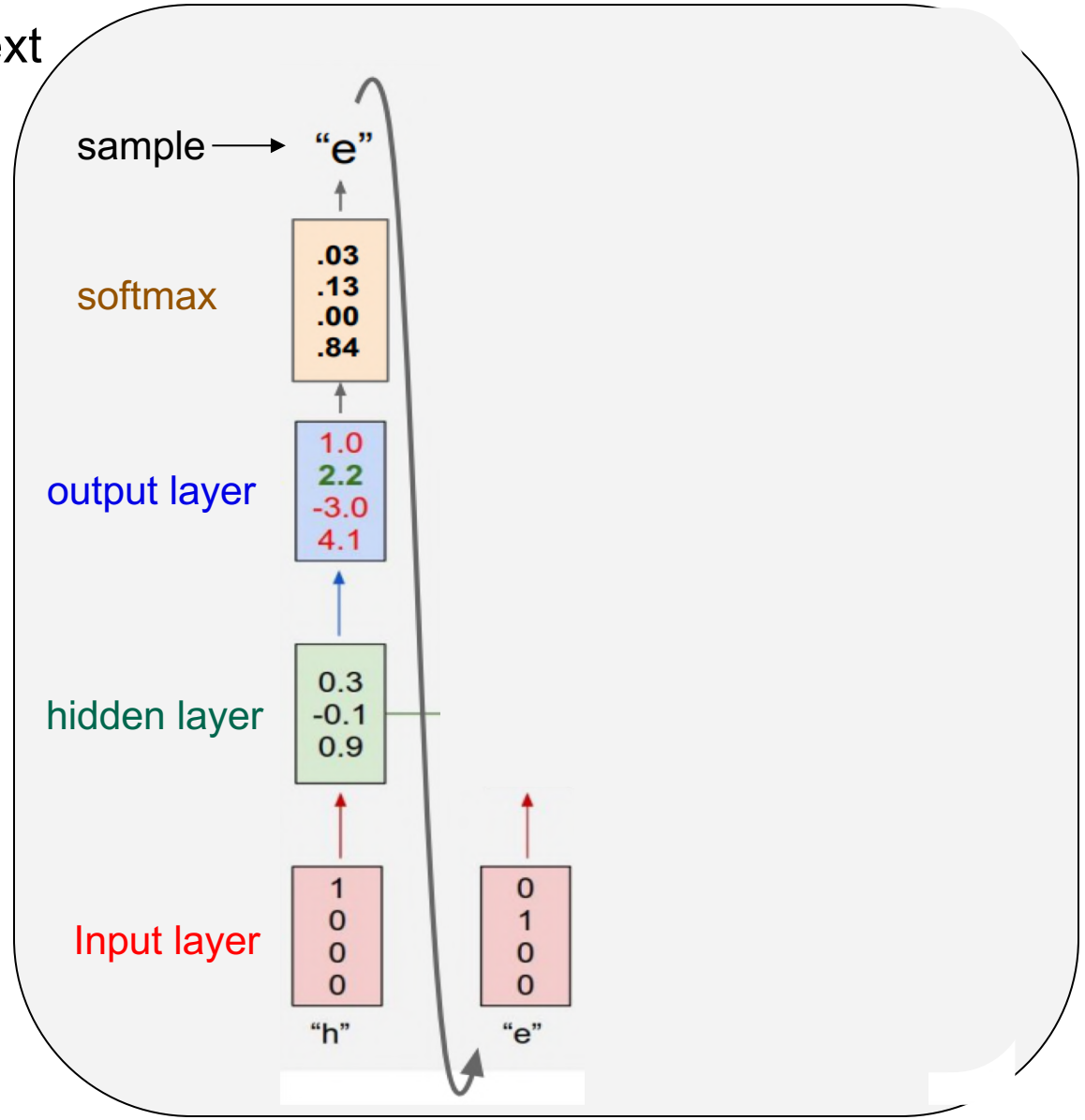

Vocabulary: [h,e,l,o]

Source: Justin Johnson

# RNN Example: Language Modeling

At test time: Generate new text

- One char at a time
- feed back to the model



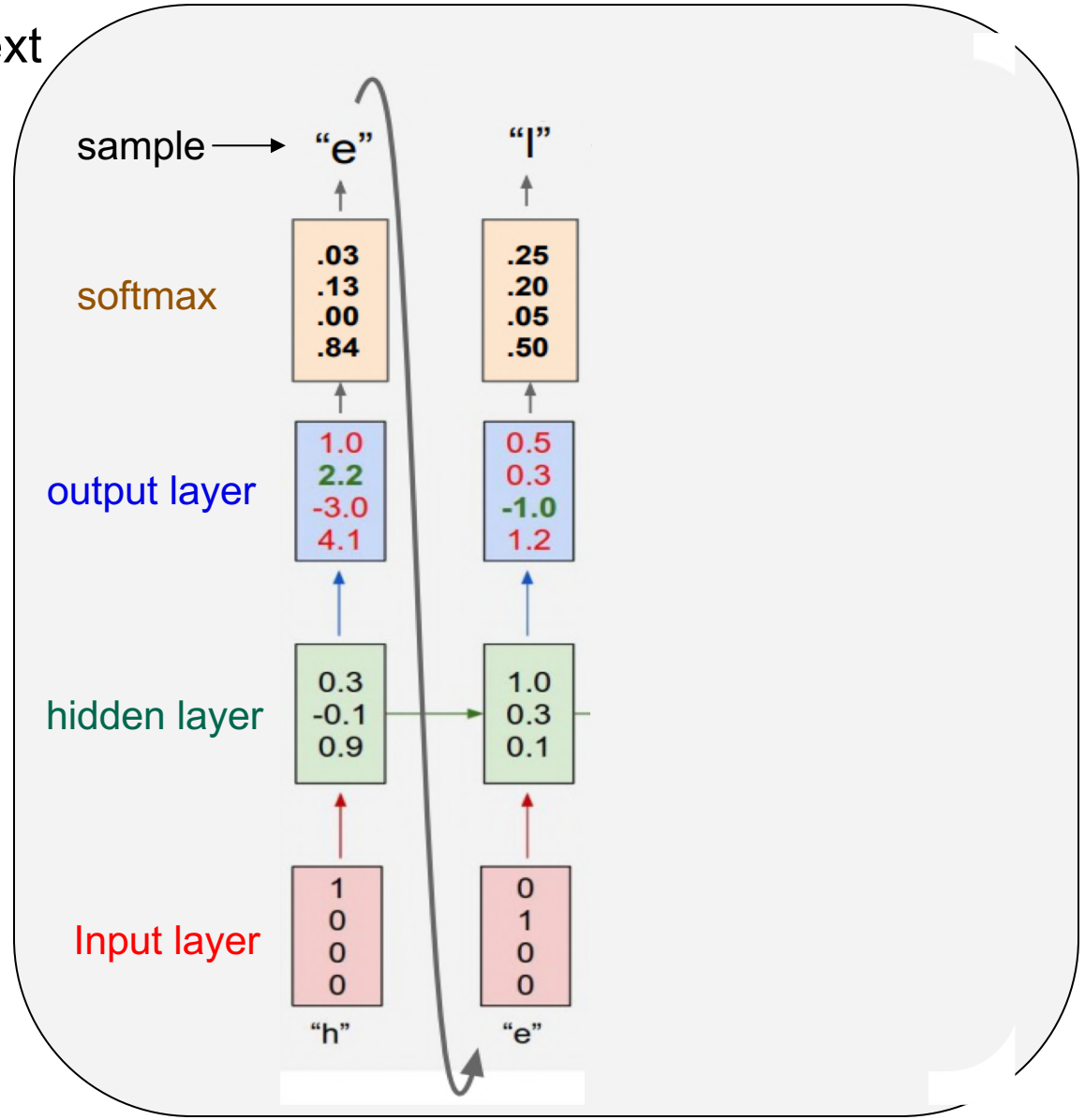

Vocabulary: [h,e,l,o]

Source: Justin Johnson

# RNN Example: Language Modeling

At test time: Generate new text

- One char at a time
- feed back to the model



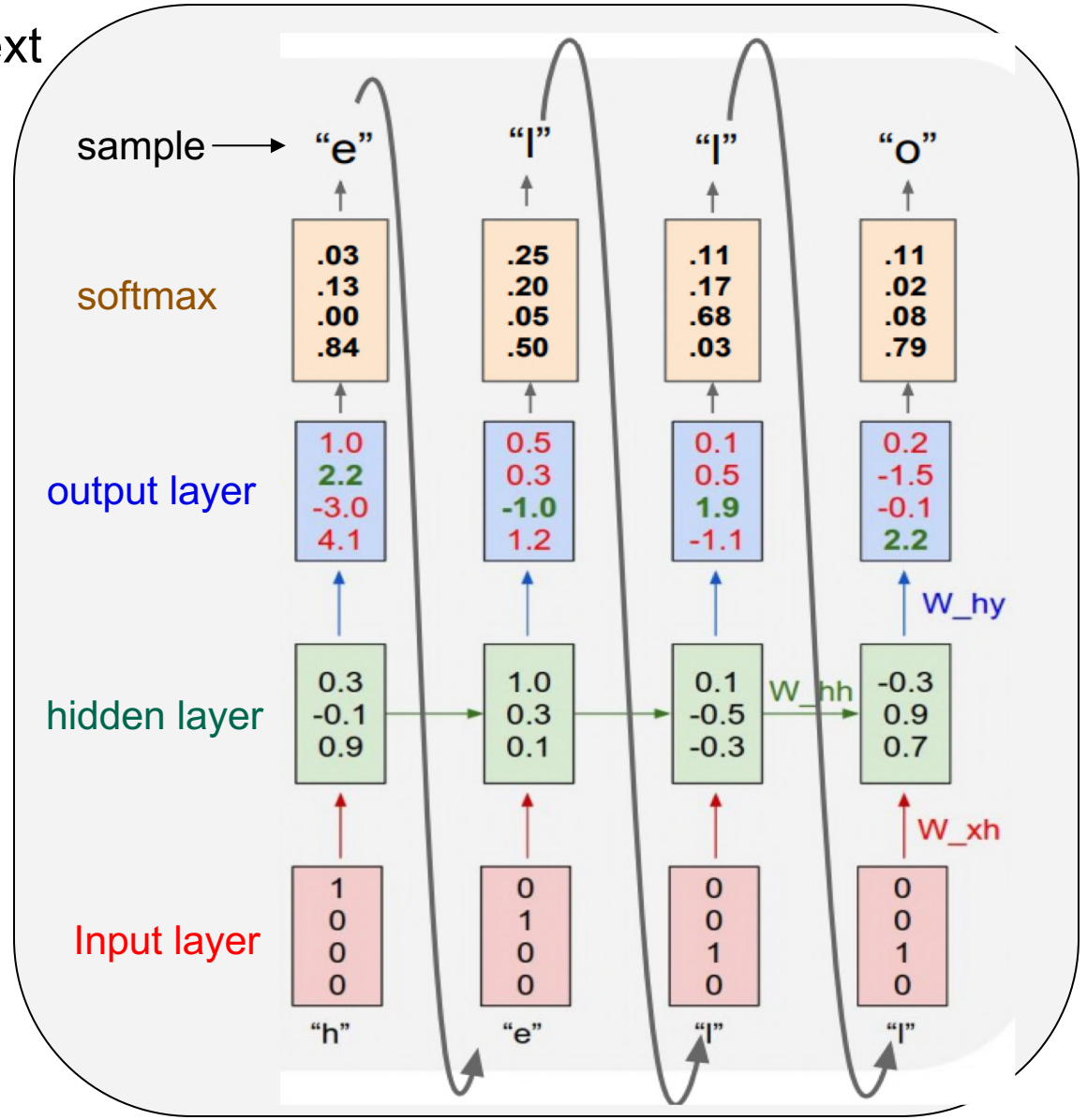

Vocabulary: [h,e,l,o]

Source: Justin Johnson

# RNN Example: Language Modeling

At test time: Generate new text

- One char at a time
- feed back to the model

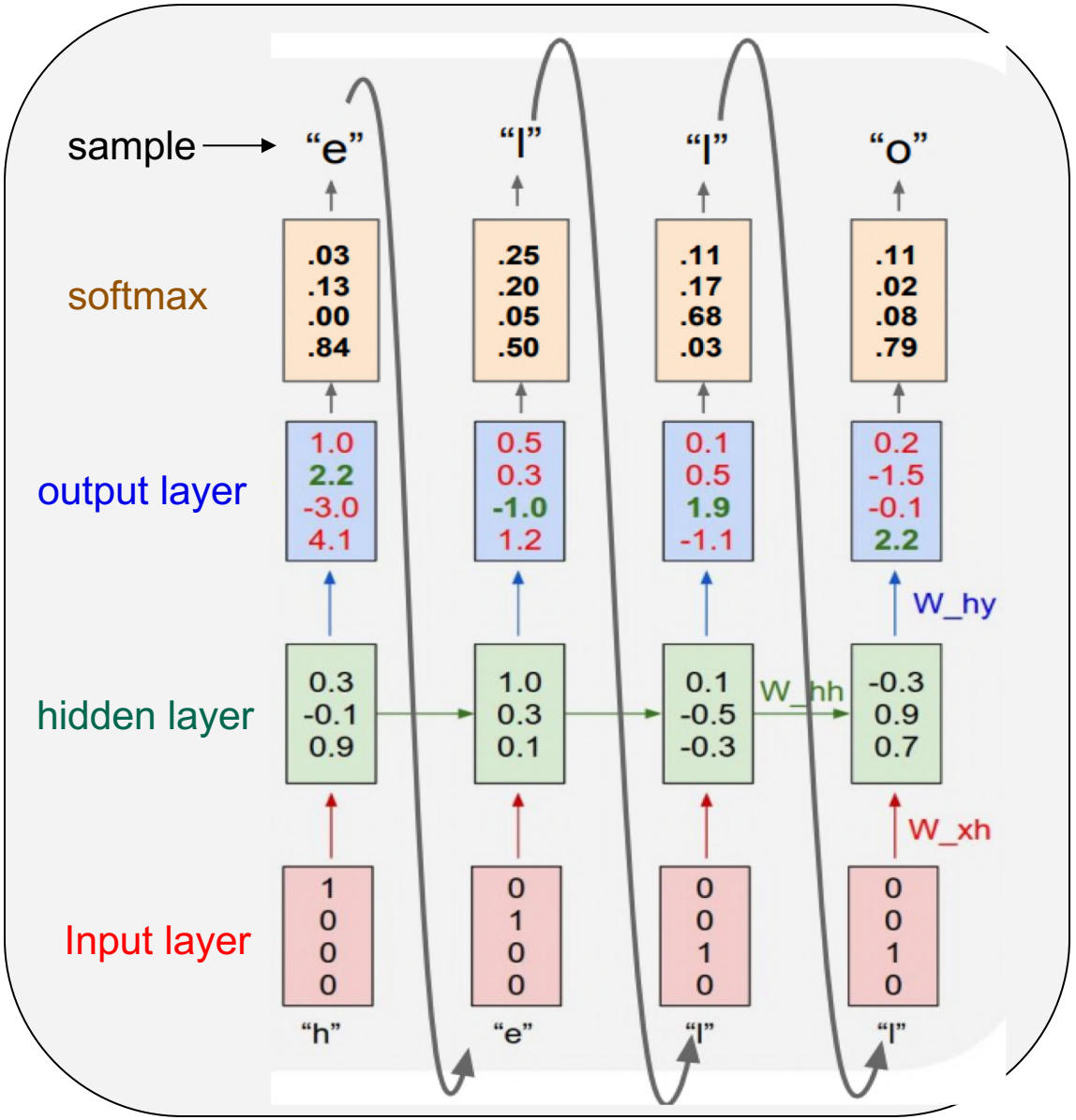



Vocabulary: [h,e,l,o]

Source: Justin Johnson

# RNN Example: Language Modeling

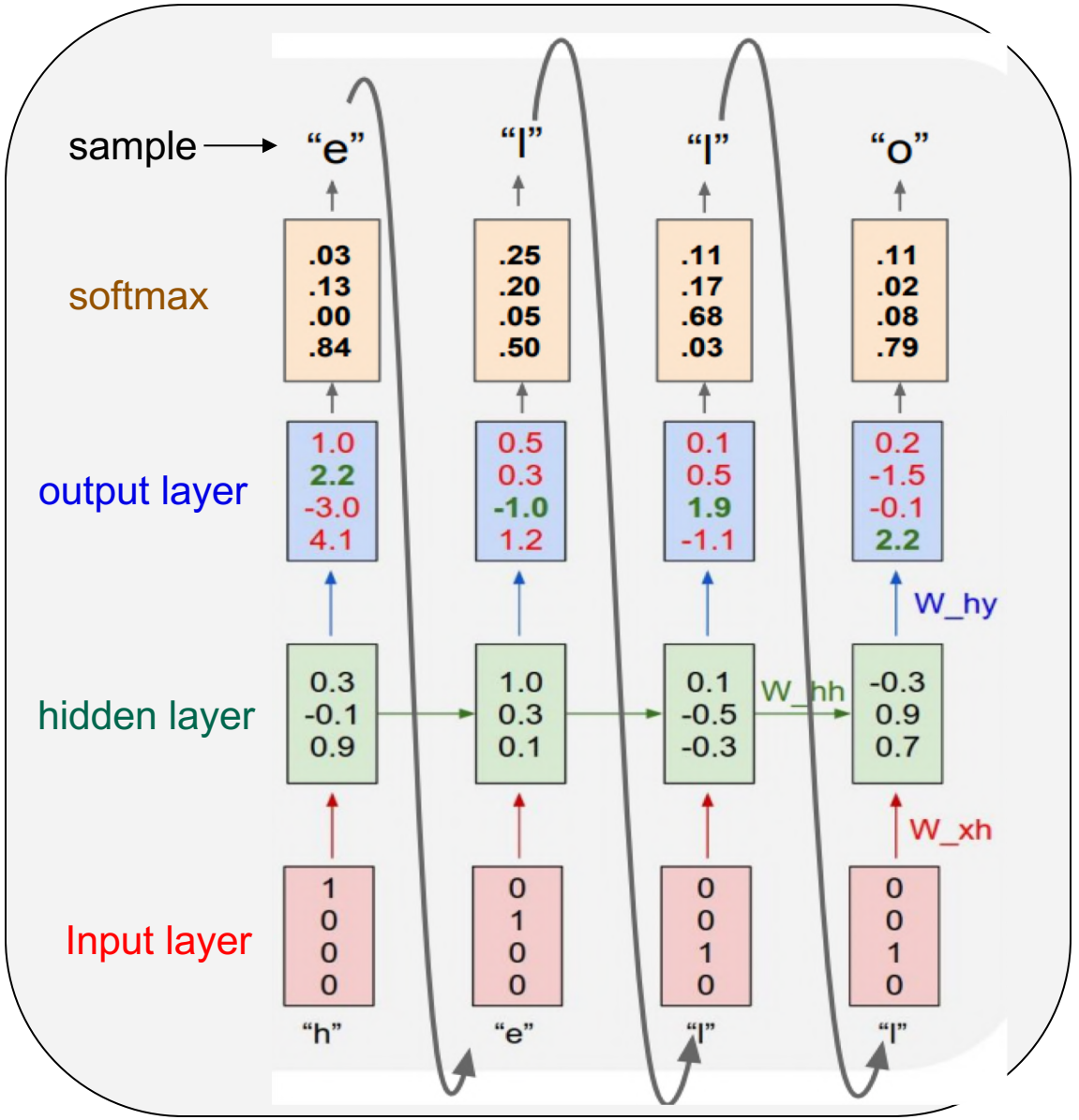
$$p(y_1, y_2, \ldots, y_n)$$
$$= p(y_1)p(y_2|y_1)p(y_3|y_1, y_2) \ldots$$
$$= \prod_{i=1}^{n} p(y_i|y_1, \ldots, y_{i-1})$$
$$\approx \prod_{i=1}^{n} P_W(y_i|h_i)$$




Source: Justin Johnson

# RNN Example: Language Modeling

So far: inputs were encodes as **one-hot vector**.




Vocabulary: [h,e,l,o]

Source: Justin Johnson

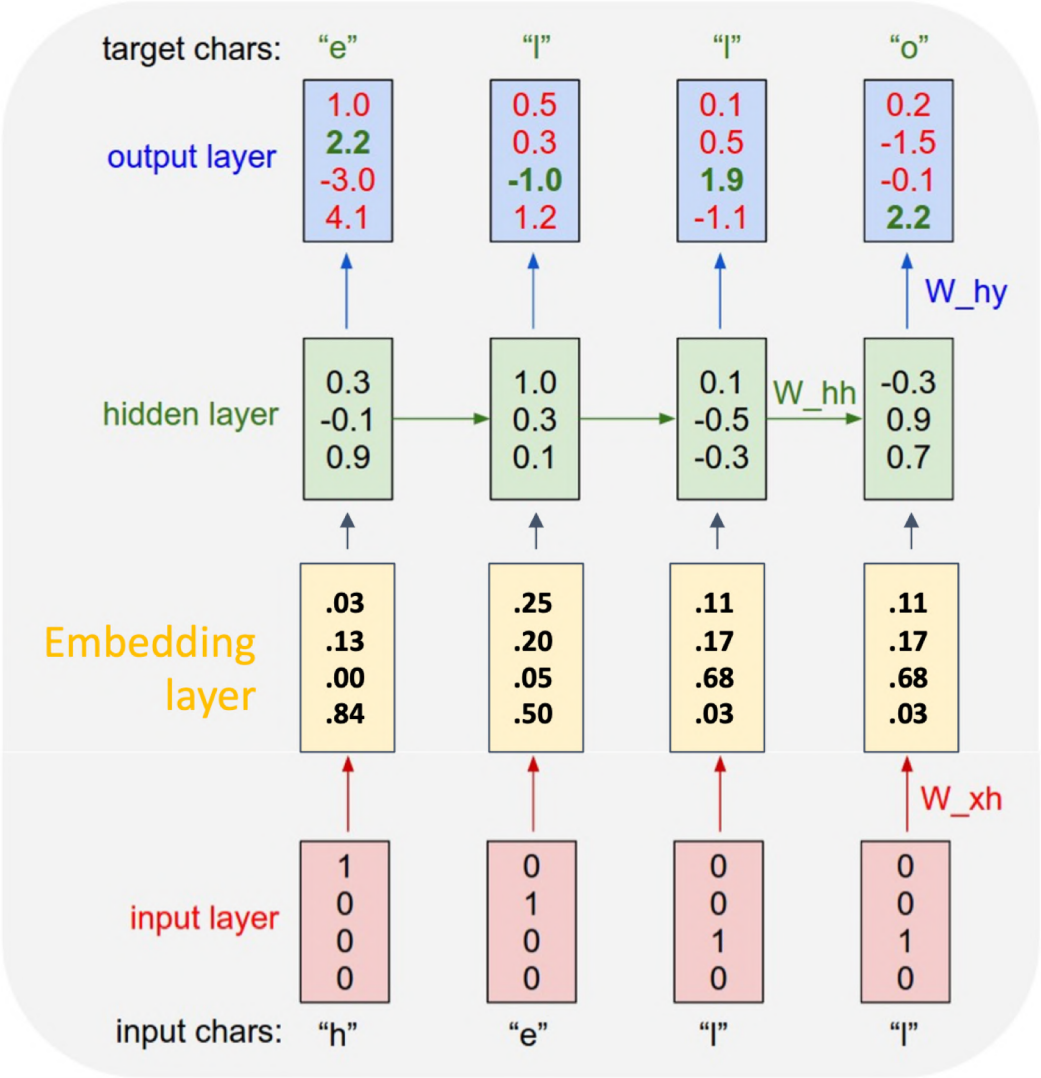
# RNN Example: Language Modeling

So far: inputs were encodes as **one-hot vector**.

Commonly, inputs are encoded as **word2vec** (char2vec)

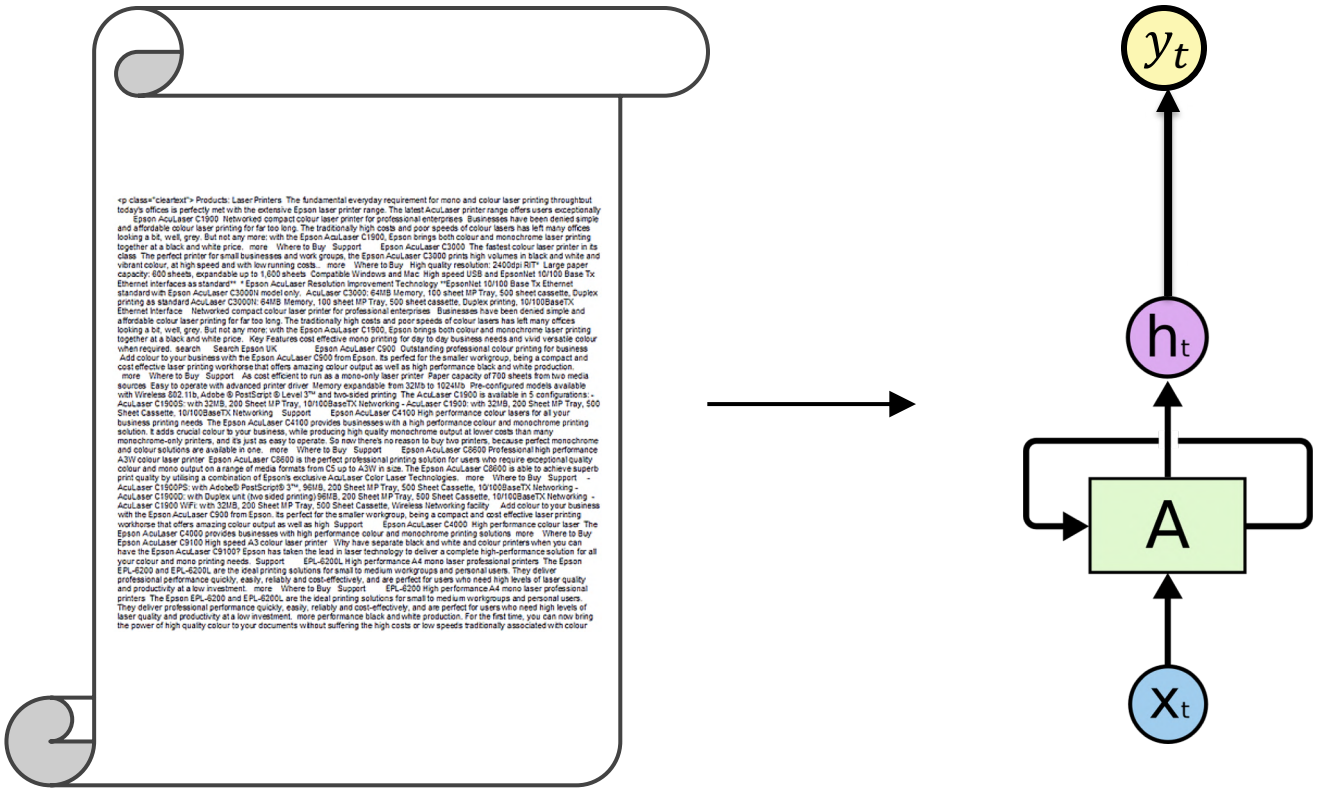
$$\begin{bmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ w_{21} & w_{22} & w_{23} & w_{24} \\ w_{31} & w_{32} & w_{33} & w_{34} \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} w_{12} \\ w_{22} \\ w_{32} \end{bmatrix}$$

**Word2vec** embedding is trained so that ”close” words will be embedded close to each other in the embedding space.



# RNN Example

## Char RNN

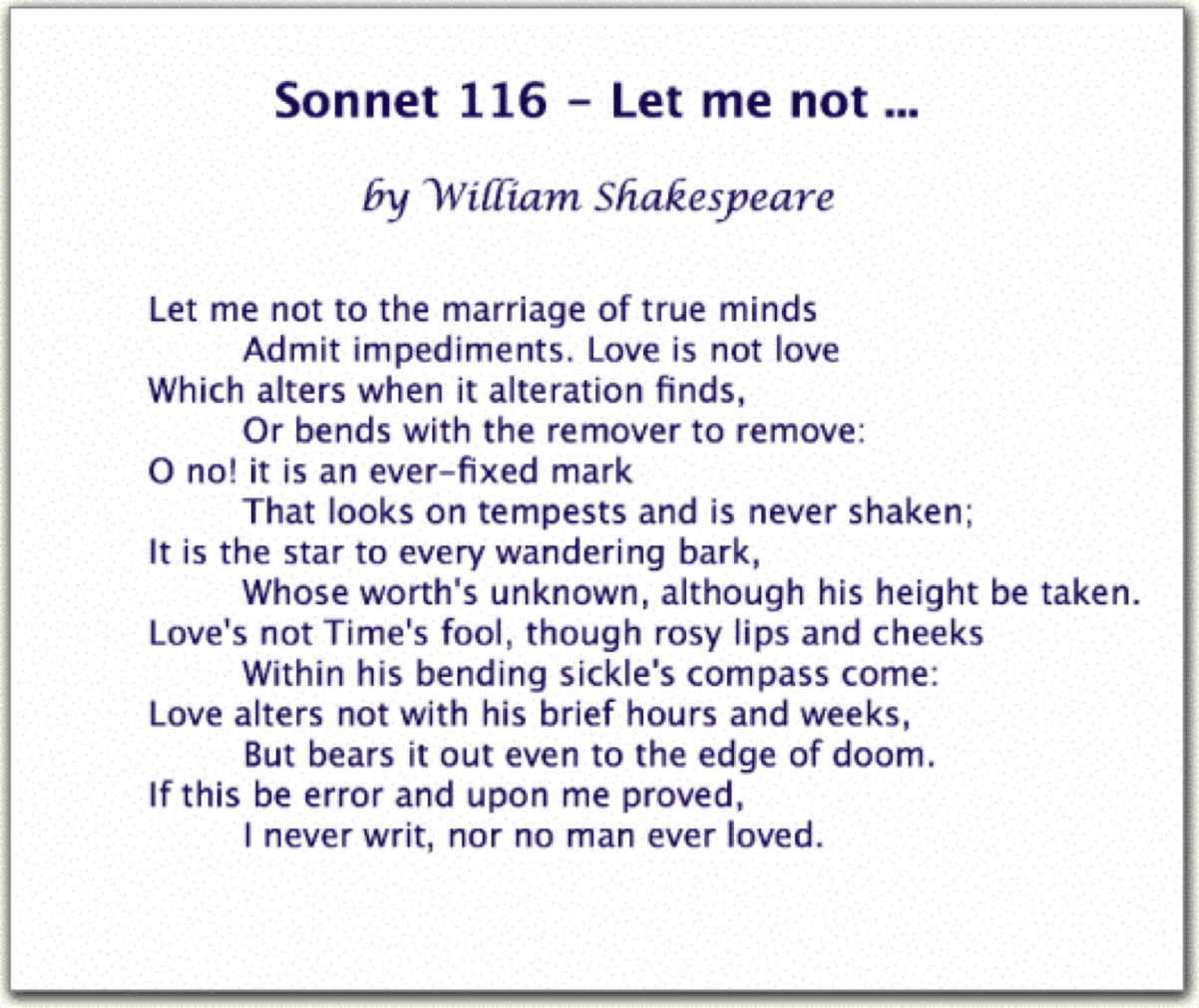





From A. Karpathy [Stanford cs231n]

# RNN Example

**Char RNN trained on William Shakespeare**



**Sonnet 116 – Let me not ...**

*by William Shakespeare*

Let me not to the marriage of true minds
Admit impediments. Love is not love
Which alters when it alteration finds,
Or bends with the remover to remove:
O no! it is an ever-fixed mark
That looks on tempests and is never shaken;
It is the star to every wandering bark,
Whose worth's unknown, although his height be taken.
Love's not Time's fool, though rosy lips and cheeks
Within his bending sickle's compass come:
Love alters not with his brief hours and weeks,
But bears it out even to the edge of doom.
If this be error and upon me proved,
I never writ, nor no man ever loved.



A. Karpathy Stanford cs231n

# RNN Example

## Char RNN trained on William Shakespeare

at first:

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

train more

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

train more

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

train more

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



A. Karpathy Stanford cs231n

# RNN Example

## Char RNN trained on William Shakespeare

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

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

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

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

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

VIOLA:
I'll drink it.
```

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

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

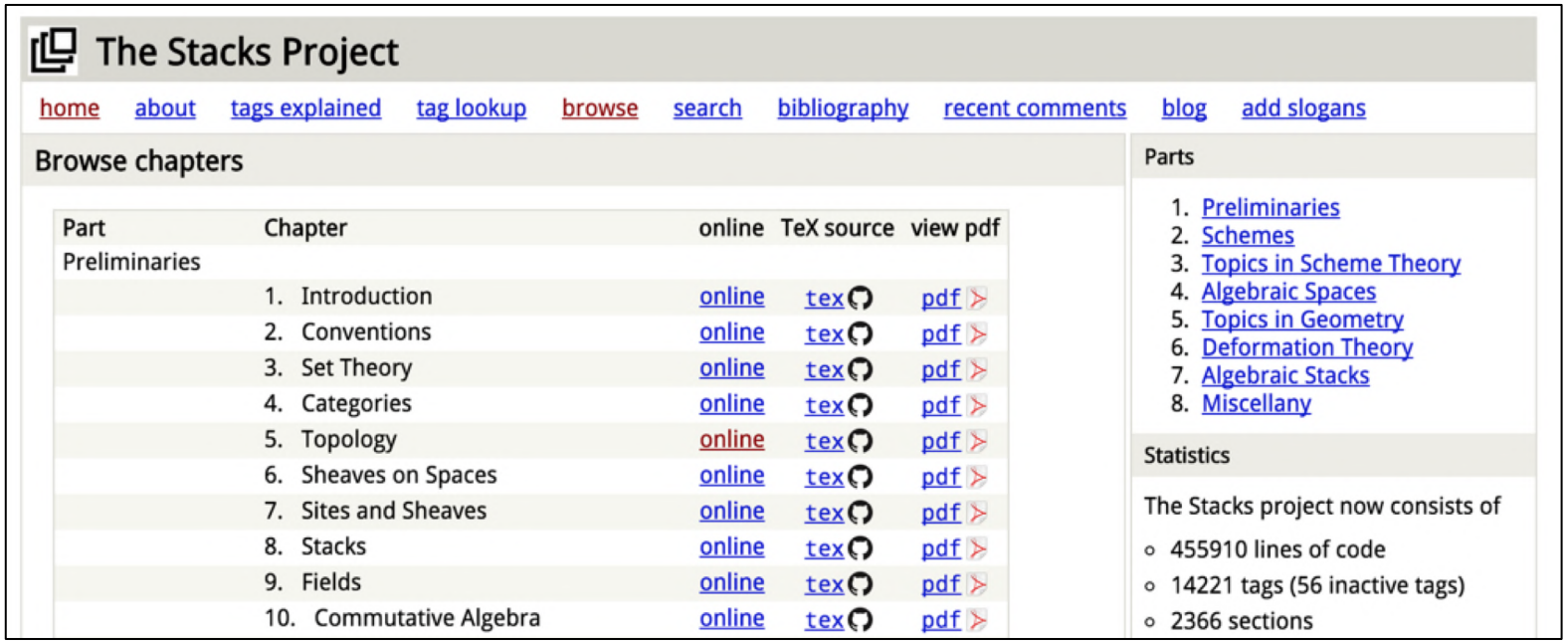


A. Karpathy [Stanford cs231n]

# RNN Example

**Char RNN trained on Algebraic Geometry Latex book**

open source textbook on algebraic geometry




Latex source

# RNN Example

## Char RNN trained on Algebraic Geometry Latex book

For $\bigoplus_{n=1,\ldots,m}$ where $\mathcal{L}_{m_\bullet} = 0$, hence we can find a closed subset $\mathcal{H}$ in $\mathcal{H}$ and any sets $\mathcal{F}$ on $X$, $U$ is a closed immersion of $S$, then $U \to T$ is a separated algebraic space.

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

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

and the compariciely in the fibre product covering we have to prove the lemma generated by $\coprod Z \times_U U \to V$. Consider the maps $M$ along the set of points $Sch_{fppf}$ and $U \to U$ is the fibre category of $S$ in $U$ in Section, ?? and the fact that any $U$ affine, see Morphisms, Lemma ??. Hence we obtain a scheme $S$ and any open subset $W \subset U$ in $Sh(G)$ such that $\mathrm{Spec}(R') \to S$ is smooth or an

$$U = \bigcup U_i \times_{S_i} U_i$$

which has a nonzero morphism we may assume that $f_i$ is of finite presentation over $S$. We claim that $\mathcal{O}_{X,x}$ is a scheme where $x, x', s'' \in S'$ such that $\mathcal{O}_{X,x'} \to \mathcal{O}'_{X',x'}$ is separated. By Algebra, Lemma ?? we can define a map of complexes $\mathrm{GL}_{S'}(x'/S'')$ and we win. □

To prove study we see that $\mathcal{F}|_U$ is a covering of $\mathcal{X}'$, and $\mathcal{T}_i$ is an object of $\mathcal{F}_{X/S}$ for $i > 0$ and $\mathcal{F}_p$ exists and let $\mathcal{F}_i$ be a presheaf of $\mathcal{O}_X$-modules on $\mathcal{C}$ as a $\mathcal{F}$-module. In particular $\mathcal{F} = U/\mathcal{F}$ we have to show that

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

is a unique morphism of algebraic stacks. Note that

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

and

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

is an open subset of $X$. Thus $U$ is affine. This is a continuous map of $X$ is the inverse, the groupoid scheme $S$.

*Proof.* See discussion of sheaves of sets. □

The result for prove any open covering follows from the less of Example ??. It may replace $S$ by $X_{spaces,étale}$ which gives an open subspace of $X$ and $T$ equal to $S_{Zar}$, see Descent, Lemma ??. Namely, by Lemma ?? we see that $R$ is geometrically regular over $S$.

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

*Suppose* $X = \lim |X|$ *(by the formal open covering* $X$ *and a single map* $\underline{Proj}_X(\mathcal{A}) =$ $\mathrm{Spec}(B)$ *over* $U$ *compatible with the complex*

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

*When in this case of to show that* $\mathcal{Q} \to \mathcal{C}_{Z/X}$ *is stable under the following result in the second conditions of (1), and (3). This finishes the proof. By Definition ?? (without element is when the closed subschemes are catenary. If* $T$ *is surjective we may assume that* $T$ *is connected with residue fields of* $S$*. Moreover there exists a closed subspace* $Z \subset X$ *of* $X$ *where* $U$ *in* $X'$ *is proper (some defining as a closed subset of the uniqueness it suffices to check the fact that the following theorem*

(1) $f$ *is locally of finite type. Since* $S = \mathrm{Spec}(R)$ *and* $Y = \mathrm{Spec}(R)$.

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

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

**Lemma 0.2.** *Let* $X$ *be a locally Noetherian scheme over* $S$, $E = \mathcal{F}_{X/S}$. *Set* $\mathcal{I} = \mathcal{J}_1 \subset \mathcal{I}'_n$. *Since* $\mathcal{I}^n \subset \mathcal{I}^n$ *are nonzero over* $i_0 \leq \mathfrak{p}$ *is a subset of* $\mathcal{J}_{n,0} \circ \overline{A}_2$ *works.*

**Lemma 0.3.** *In Situation ??. Hence we may assume* $\mathfrak{q}' = 0$.

*Proof.* We will use the property we see that $\mathfrak{p}$ is the mext functor (??). On the other hand, by Lemma ?? we see that

$$D(\mathcal{O}_{X'}) = \mathcal{O}_X(D)$$

where $K$ is an $F$-algebra where $\delta_{n+1}$ is a scheme over $S$. □

A. Karpathy Stanford cs231n

# RNN Example

## Char RNN trained on Algebraic Geometry Latex book

*Proof.* Omitted. □

**Lemma 0.1.** *Let $\mathcal{C}$ be a set of the construction.*

*Let $\mathcal{C}$ be a gerber covering. Let $\mathcal{F}$ be a quasi-coherent sheaves of $\mathcal{O}$-modules. We have to show that*

$$\mathcal{O}_{\mathcal{O}_X} = \mathcal{O}_X(\mathcal{L})$$

.

*Proof.* This is an algebraic space with the composition of sheaves $\mathcal{F}$ on $X_{\acute{e}tale}$ we have

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

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

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

*Proof.* See Spaces, Lemma ??. □

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

*The following to the construction of the lemma follows.*

*Let $X$ be a scheme. Let $X$ be a scheme covering. Let*

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

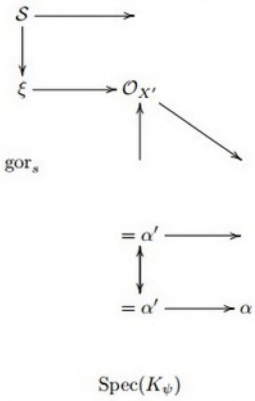
*be a morphism of algebraic spaces over $S$ and $Y$.*

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

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

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

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




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

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

□

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

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

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

is an isomorphism of covering of $\mathcal{O}_{X_i}$. If $\mathcal{F}$ is the unique element of $\mathcal{F}$ such that $X$ is an isomorphism.
The property $\mathcal{F}$ is a disjoint union of Proposition ?? and we can filtered set of presentations of a scheme $\mathcal{O}_X$-algebra with $\mathcal{F}$ are opens of finite type over $S$.
If $\mathcal{F}$ is a scheme theoretic image points. □

If $\mathcal{F}$ is a finite direct sum $\mathcal{O}_{X_\lambda}$ is a closed immersion, see Lemma ??. This is a sequence of $\mathcal{F}$ is a similar morphism.



A. Karpathy Stanford cs231n

# RNN Example

## Char RNN trained on C-code

```
static void do_command(struct seq_file *m, void *v)
{
  int column = 32 << (cmd[2] & 0x80);
  if (state)
    cmd = (int)(int_state ^ (in_8(&ch->ch_flags) & Cmd) ? 2 : 1);
  else
    seq = 1;
  for (i = 0; i < 16; i++) {
    if (k & (1 << 1))
      pipe = (in_use & UMXTHREAD_UNCCA) +
        ((count & 0x00000000fffffff8) & 0x000000f) << 8;
    if (count == 0)
      sub(pid, ppc_md.kexec_handle, 0x20000000);
    pipe_set_bytes(i, 0);
  }
  /* Free our user pages pointer to place camera if all dash */
  subsystem_info = &of_changes[PAGE_SIZE];
  rek_controls(offset, idx, &soffset);
  /* Now we want to deliberately put it to device */
  control_check_polarity(&context, val, 0);
  for (i = 0; i < COUNTER; i++)
    seq_puts(s, "policy ");
}
```

Generated C code



A. Karpathy Stanford cs231n

# RNN Example

## Char RNN trained on C-code

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

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

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



A. Karpathy Stanford cs231n

# RNN Example

## Char RNN trained on C-code

```
#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;
}
```

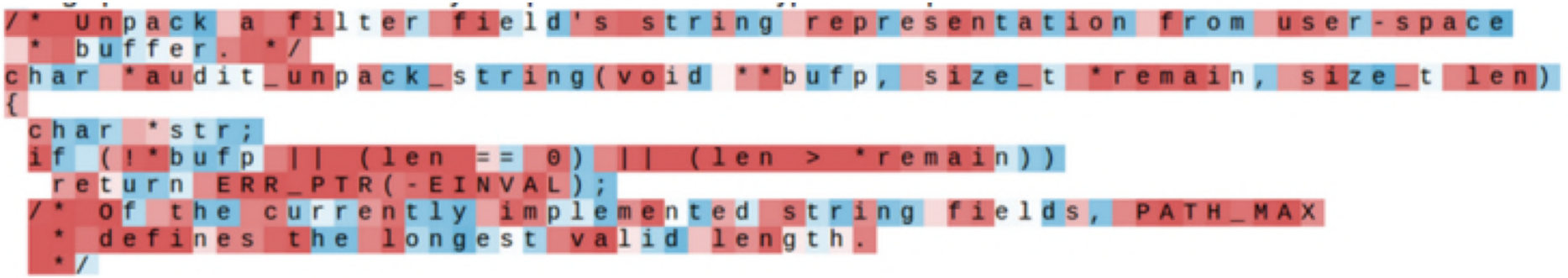


A. Karpathy Stanford cs231n

# RNN Example

**Char RNN trained on C-code:**

Searching for interpretable cells



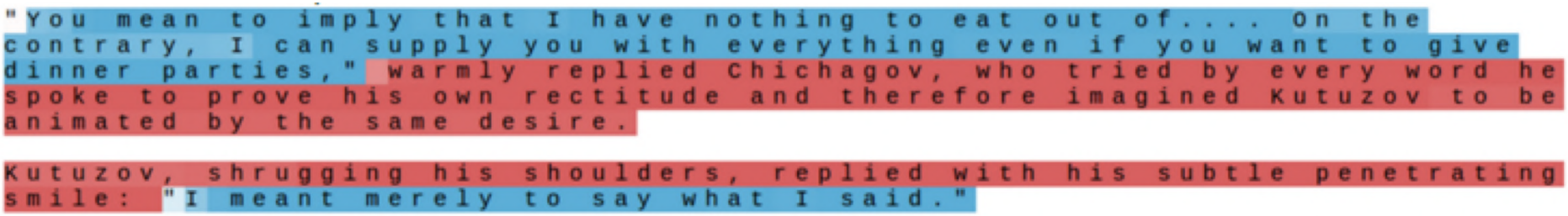
```
/* Unpack a filter field's string representation from user-space
 * buffer. */
char *audit_unpack_string(void **bufp, size_t *remain, size_t len)
{
  char *str;
  if (!*bufp || (len == 0) || (len > *remain))
    return ERR_PTR(-EINVAL);
  /* Of the currently implemented string fields, PATH_MAX
   * defines the longest valid length.
   */
```

*[Visualizing and Understanding Recurrent Networks, Andrej Karpathy*, Justin Johnson*, Li Fei-Fei]*

# RNN Example

**Char RNN trained on C-code:**

Searching for interpretable cells




quote detection cell

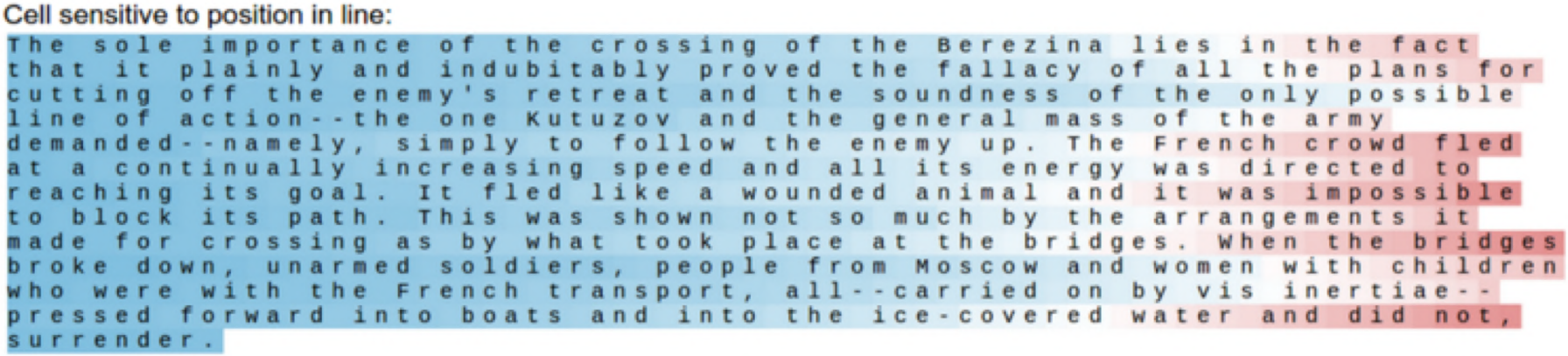


A. Karpathy Stanford cs231n

# RNN Example

**Char RNN trained on C-code:**

Searching for interpretable cells




line length tracking cell



A. Karpathy Stanford cs231n

# RNN Example

**Char RNN trained on C-code:**

Searching for interpretable cells

```
static int __dequeue_signal(struct sigpending *pending, sigset_t *mask,
        siginfo_t *info)
{
  int sig = next_signal(pending, mask);
  if (sig) {
    if (current->notifier) {
      if (sigismember(current->notifier_mask, sig)) {
        if (!(current->notifier)(current->notifier_data)) {
          clear_thread_flag(TIF_SIGPENDING);
          return 0;
        }
      }
    }
    collect_signal(sig, pending, info);
  }
  return sig;
}
```

if statement cell



A. Karpathy Stanford cs231n

# RNN Example

**Char RNN trained on C-code:**

Searching for interpretable cells

Cell that turns on inside comments and quotes:

```
/* Duplicate LSM field information.  The lsm_rule is opaque, so
 * re-initialized. */
static inline int audit_dupe_lsm_field(struct audit_field *df,
                                       struct audit_field *sf)
{
  int ret = 0;
  char *lsm_str;
  /* our own copy of lsm_str */
  lsm_str = kstrdup(sf->lsm_str, GFP_KERNEL);
  if (unlikely(!lsm_str))
    return -ENOMEM;
  df->lsm_str = lsm_str;
  /* our own (refreshed) copy of lsm_rule */
  ret = security_audit_rule_init(df->type, df->op, df->lsm_str,
                                 (void **)&df->lsm_rule);
  /* Keep currently invalid fields around in case they
   * become valid after a policy reload. */
  if (ret == -EINVAL) {
    pr_warn("audit rule for LSM \'%s\' is invalid\n",
            df->lsm_str);
    ret = 0;
  }
  return ret;
}
```

quote/comment cell

# RNN Example

**Char RNN trained on C-code:**

Searching for  interpretable cells

```
#ifdef CONFIG_AUDITSYSCALL
static inline int audit_match_class_bits(int class, u32 *mask)
{
  int i;
  if (classes[class]) {
    for (i = 0; i < AUDIT_BITMASK_SIZE; i++)
      if (mask[i] & classes[class][i])
        return 0;
  }
  return 1;
}
```

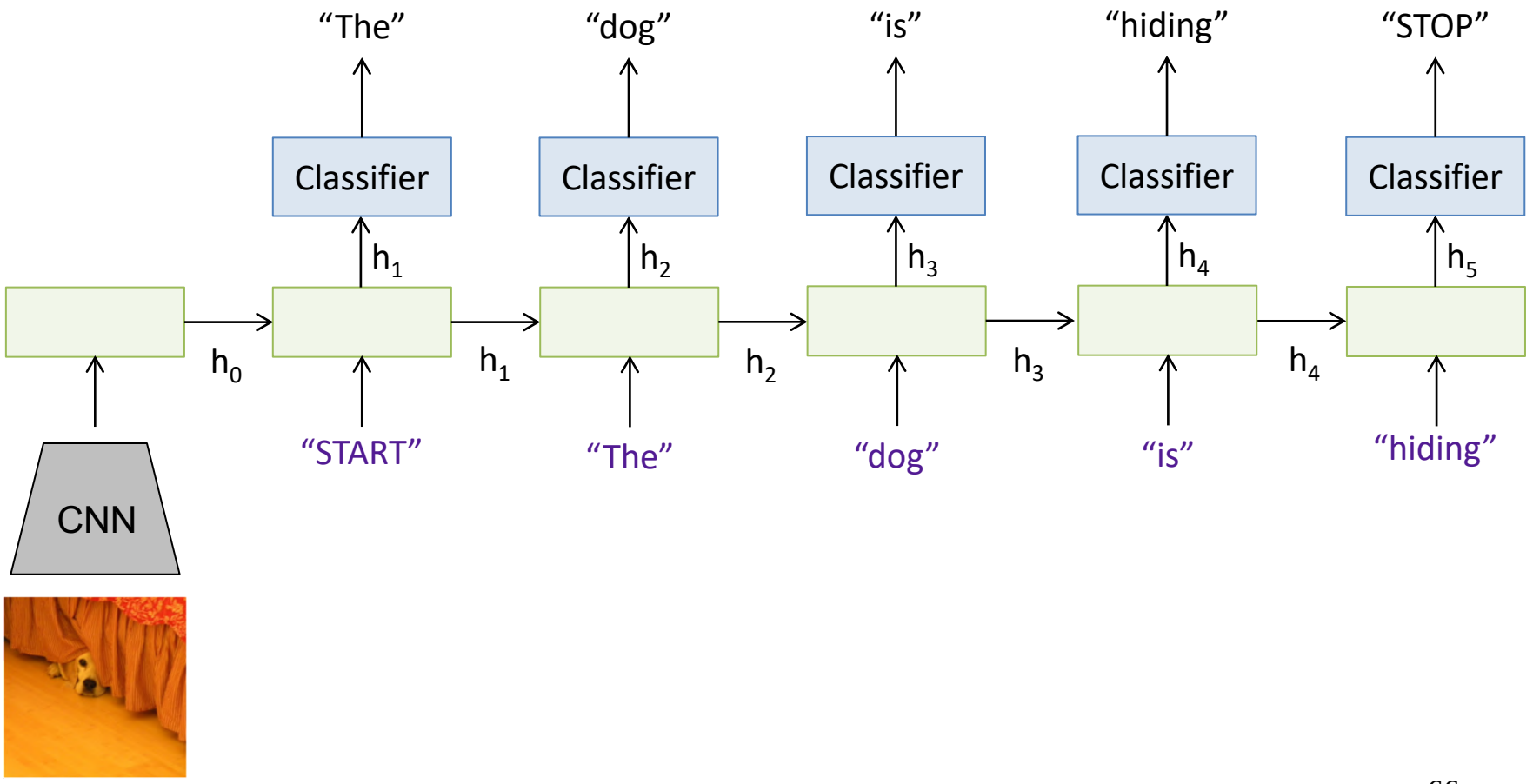
code depth cell



A. Karpathy Stanford cs231n

# RNN Example

- Example: **Image Caption Generation**



# RNN Outputs: Image Captions

A person riding a motorcycle on a dirt road.

Two dogs play in the grass.

A herd of elephants walking across a dry grass field.

A group of young people playing a game of frisbee.

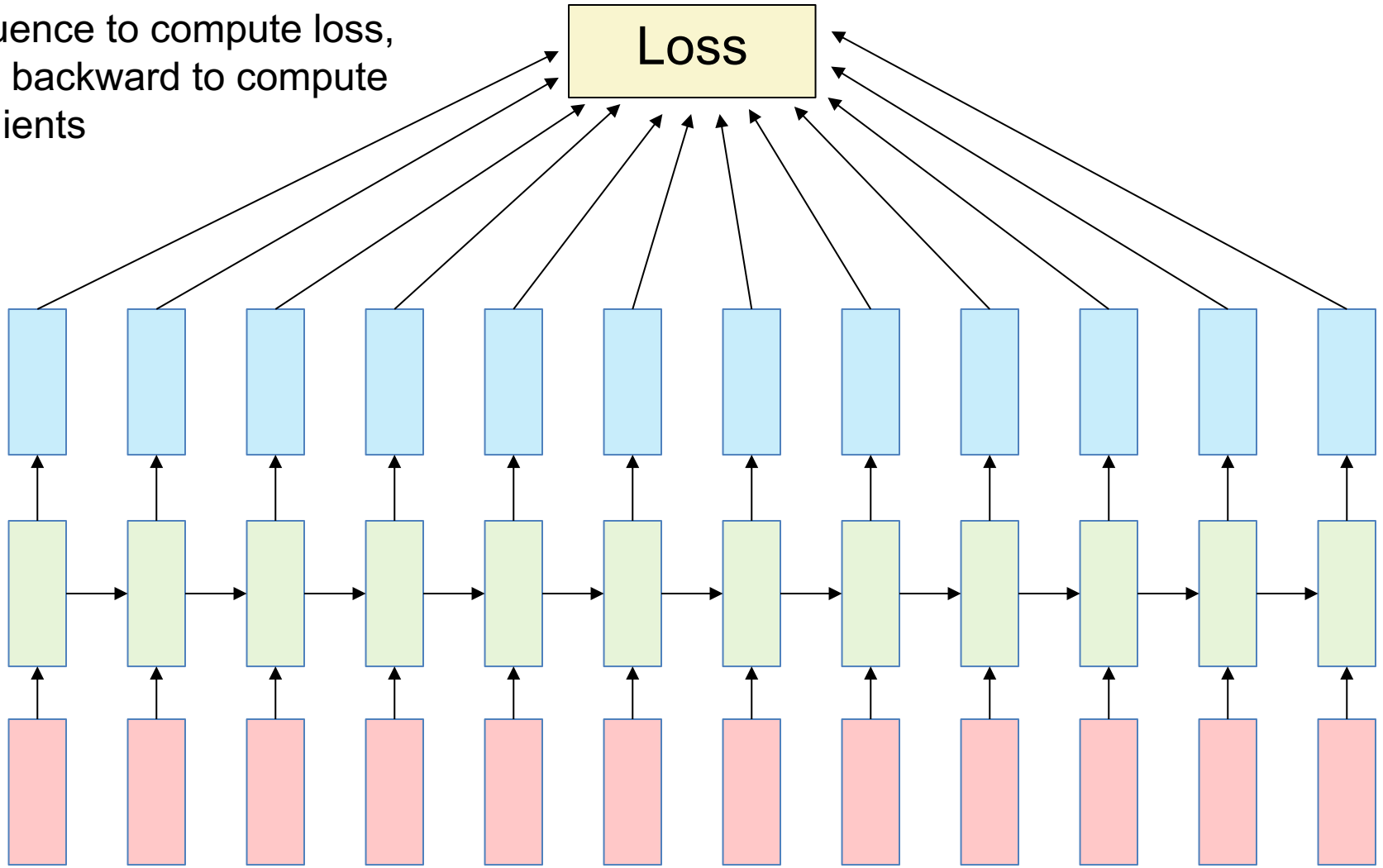
Two hockey players are fighting over the puck.

A close up of a cat laying on a couch.

O. Vinyals, A. Toshev, S. Bengio, D. Erhan, Show and Tell: A Neural Image Caption Generator, CVPR 2015

# Backpropagation Through Time

Forward the entire sequence to compute loss, then backward to compute gradients



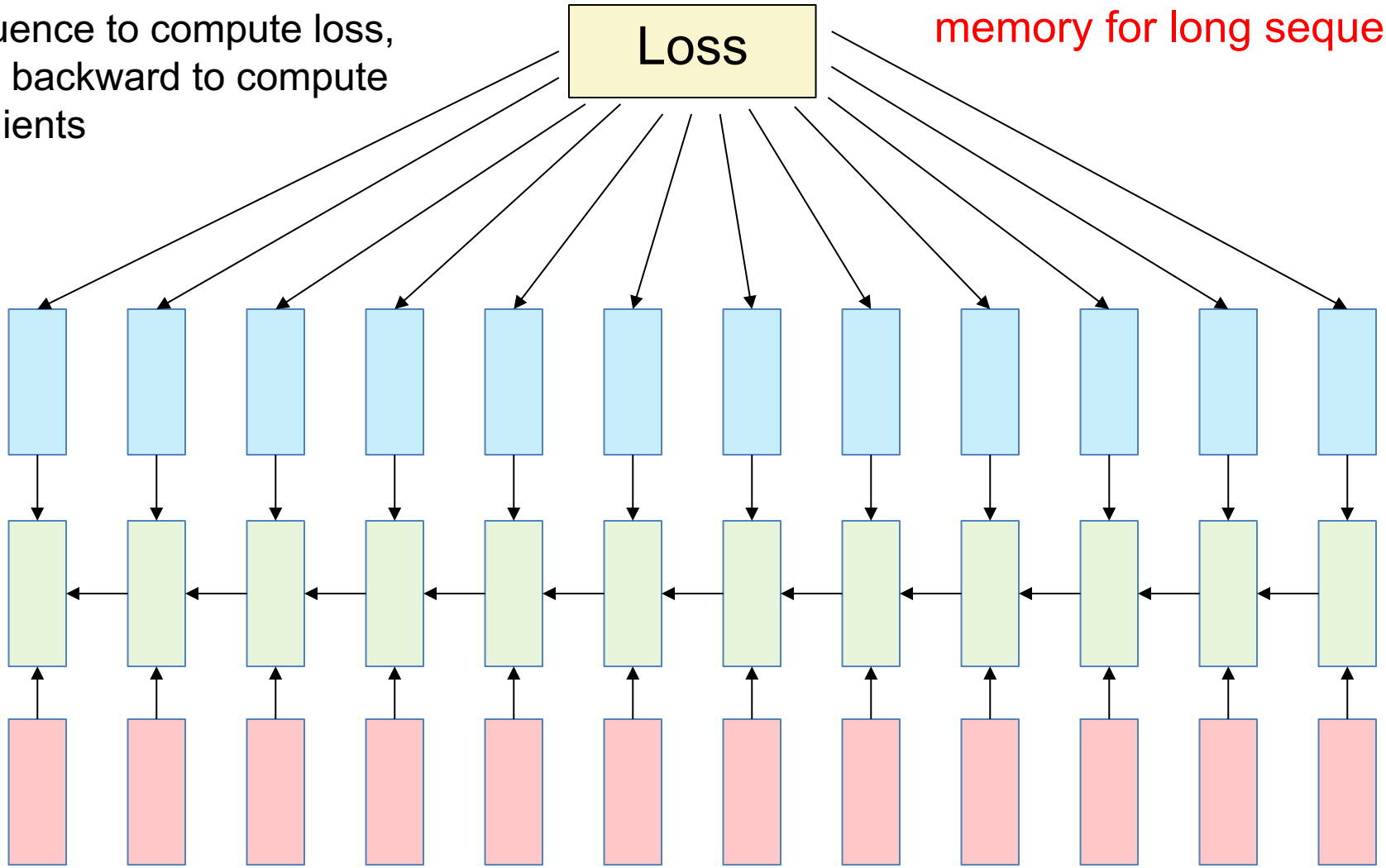



Source: Justin Johnson

# Backpropagation Through Time

Forward the entire sequence to compute loss, then backward to compute gradients

**Problem**: takes a lot of memory for long sequence.

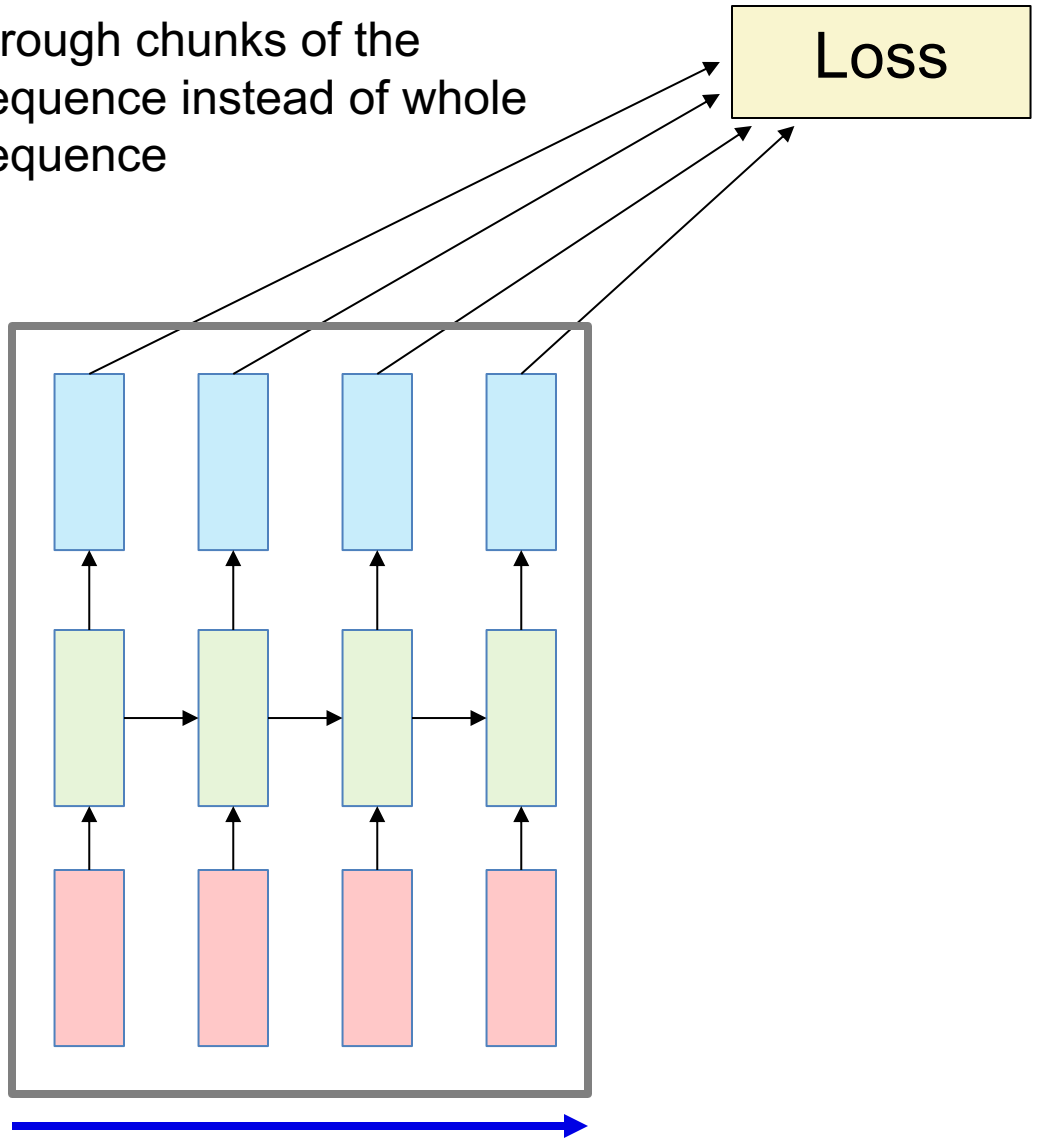





Source: Justin Johnson

# Backpropagation Through Time

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

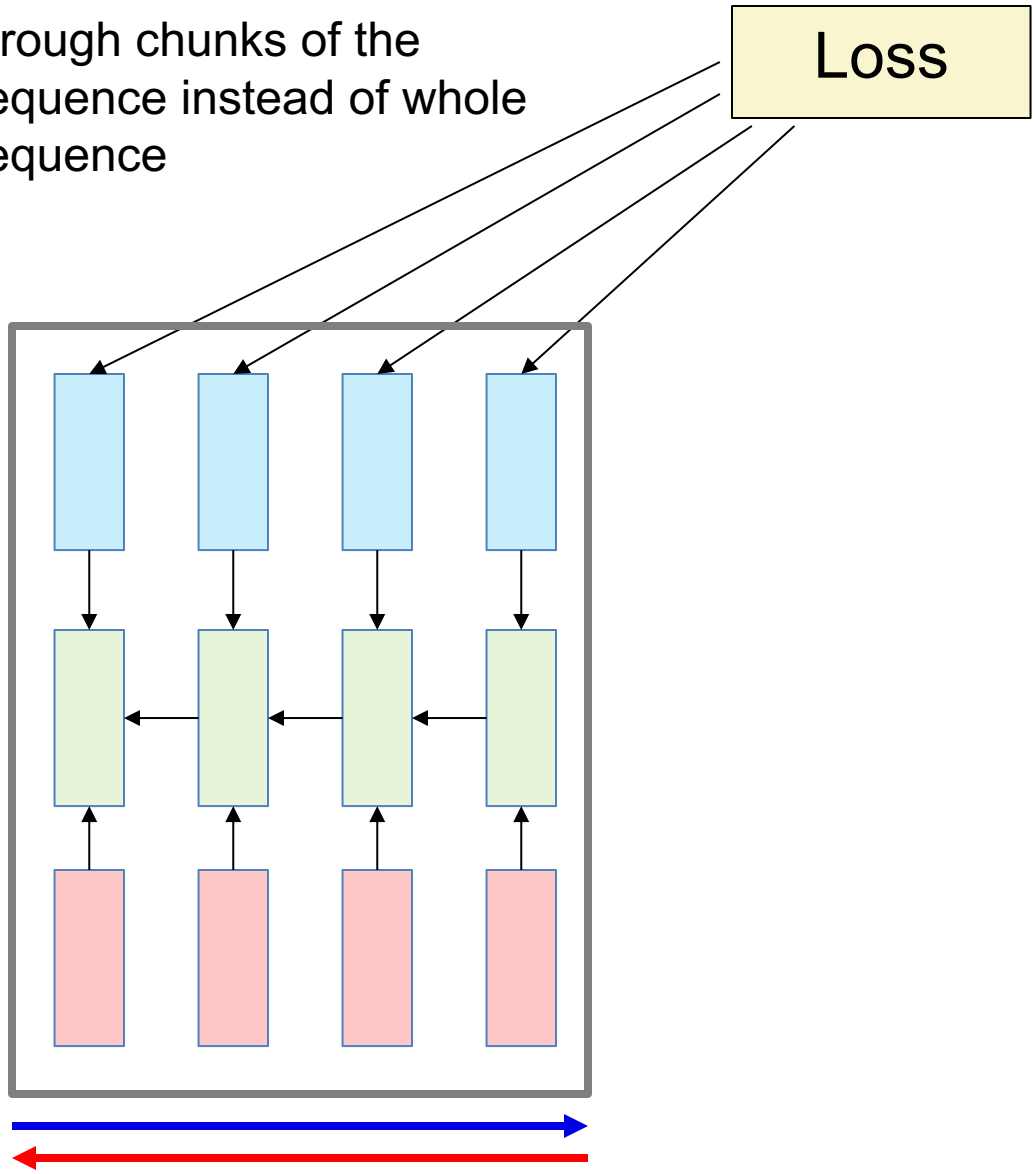





Source: Justin Johnson

# Backpropagation Through Time

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

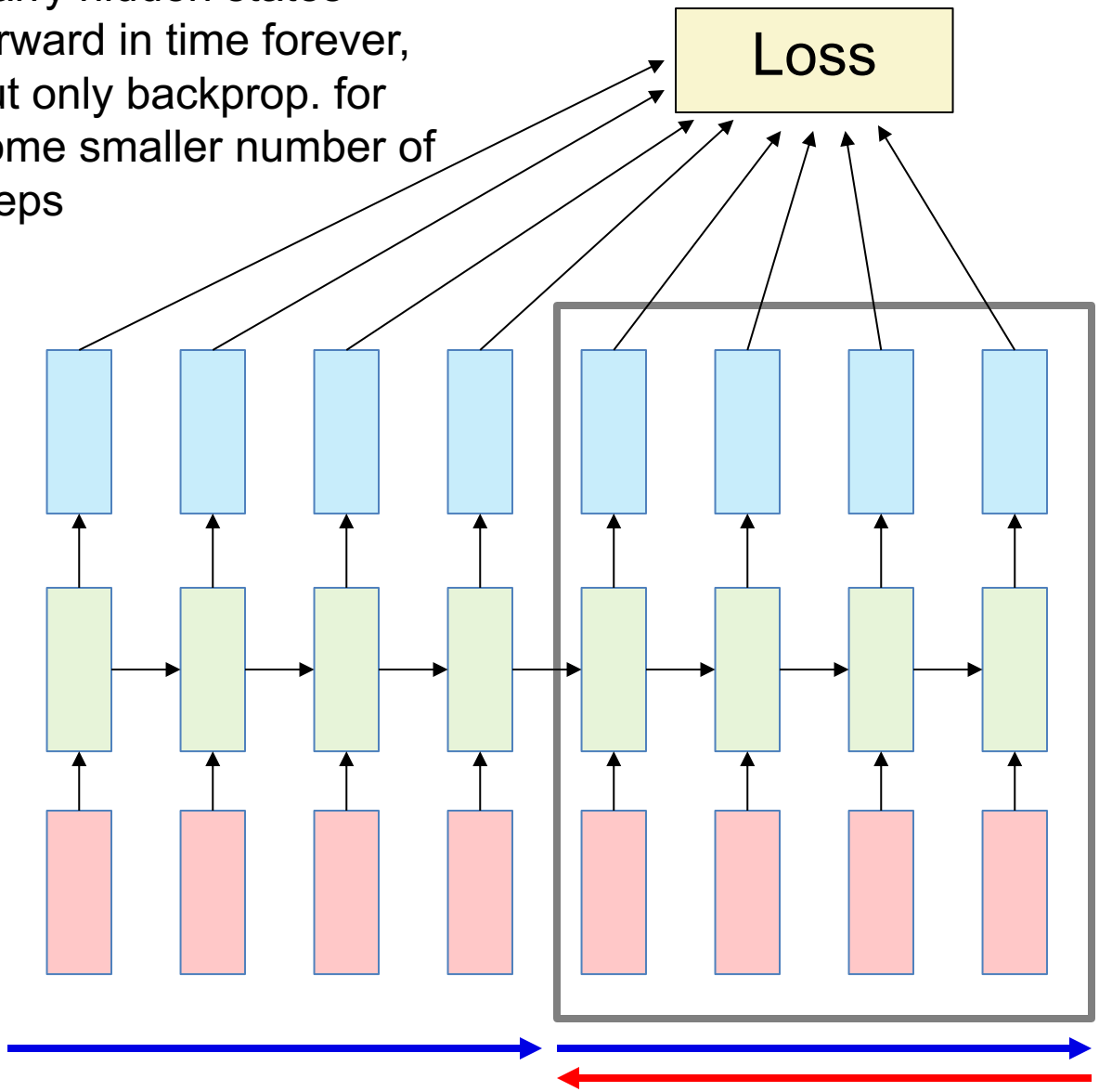





Source: Justin Johnson

# Backpropagation Through Time

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

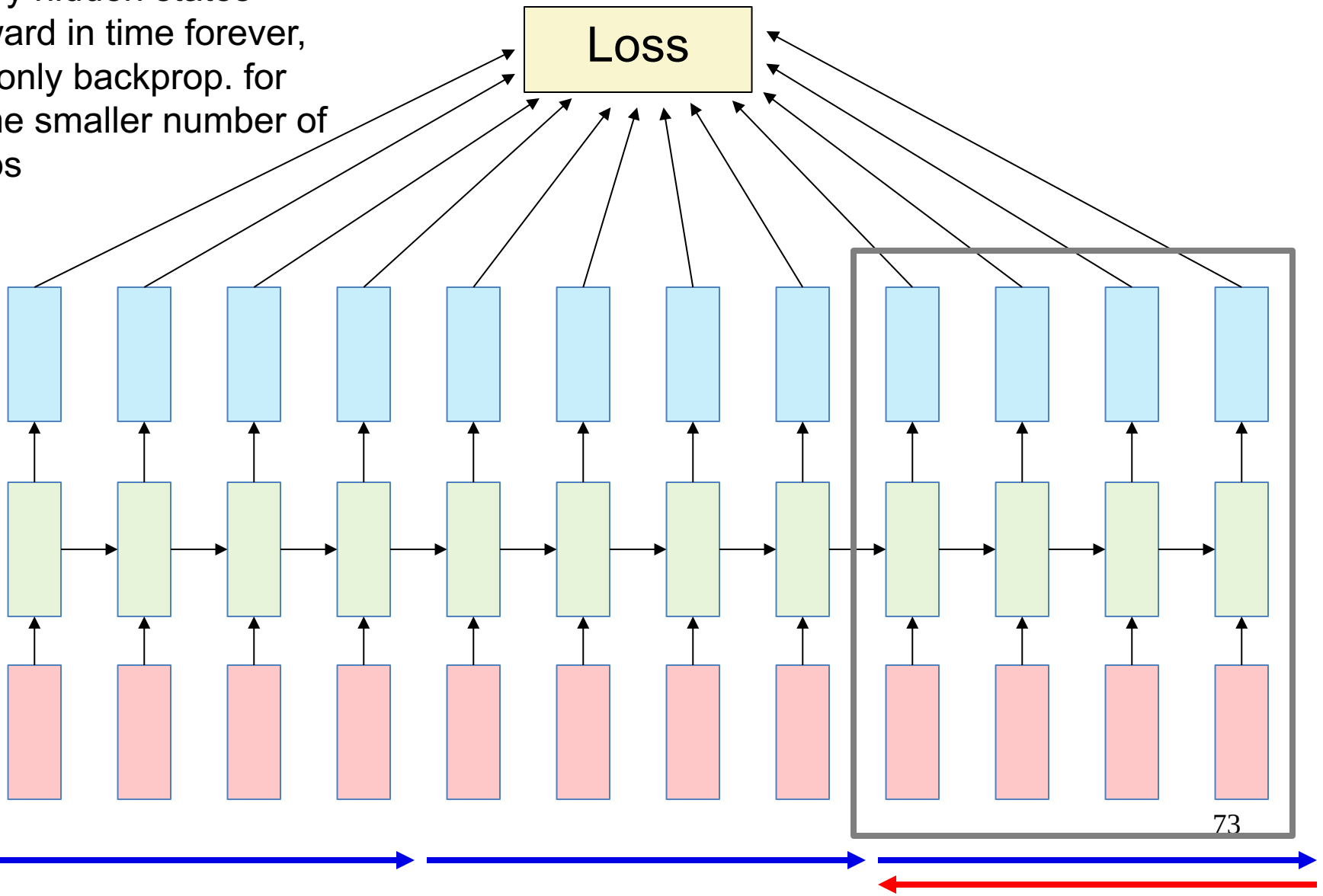




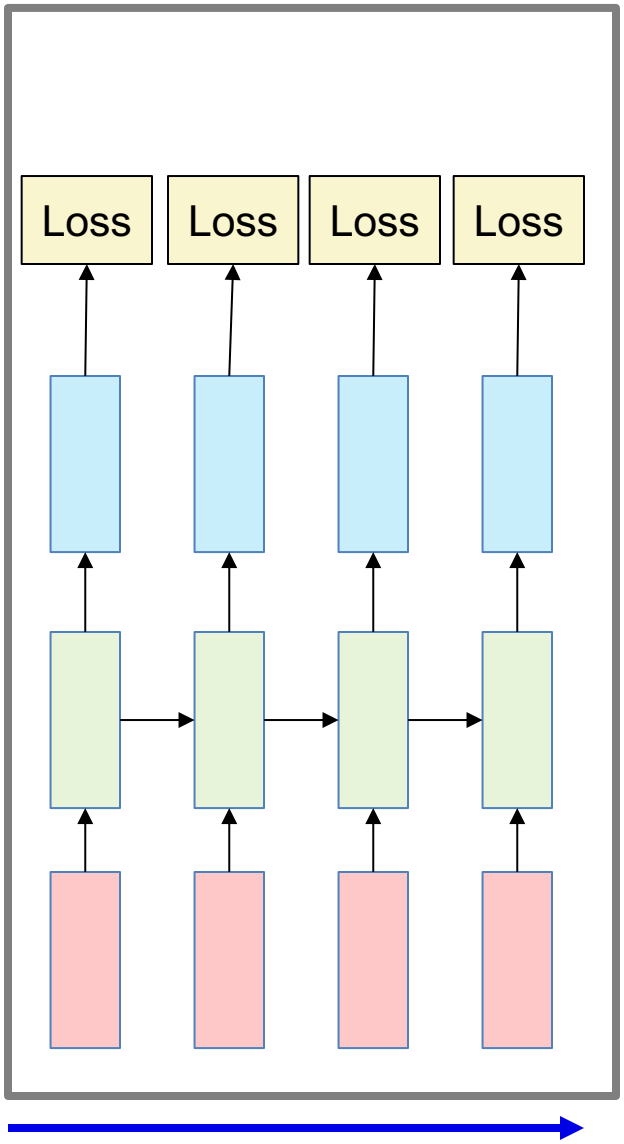

Source: Justin Johnson

# Backpropagation Through Time

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



# Backpropagation Through Time

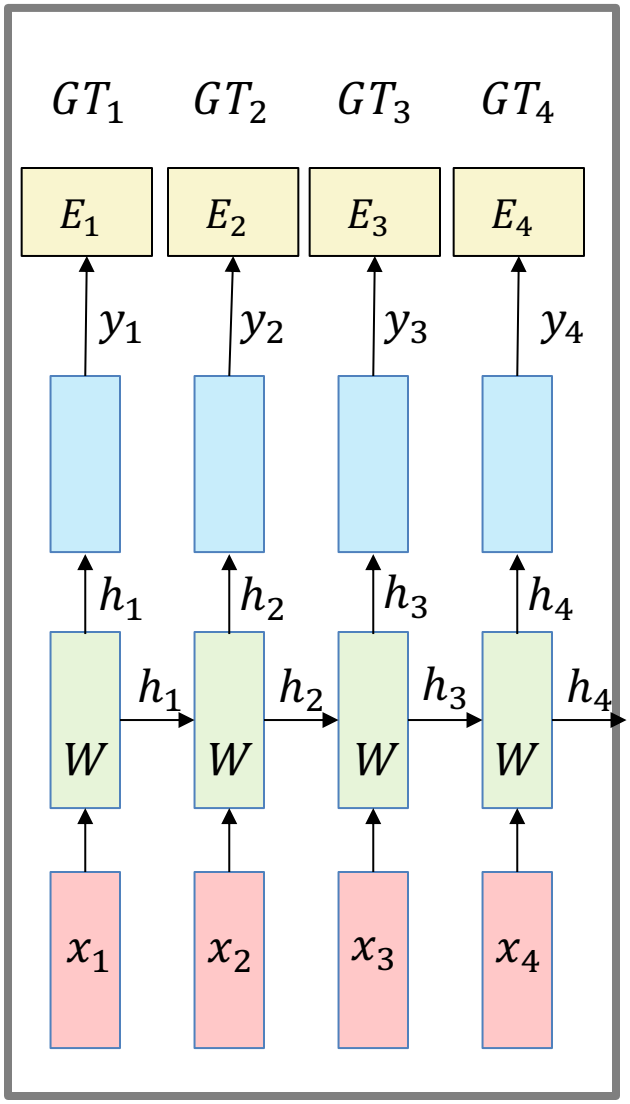



The weight updates are computed for each copy in the unfolded network, then summed (or averaged) and applied to the RNN weights.



Source: Justin Johnson

# Backpropagation Through Time




The weight updates are computed for each copy in the unfolded network, then summed (or averaged) and applied to the RNN weights.

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

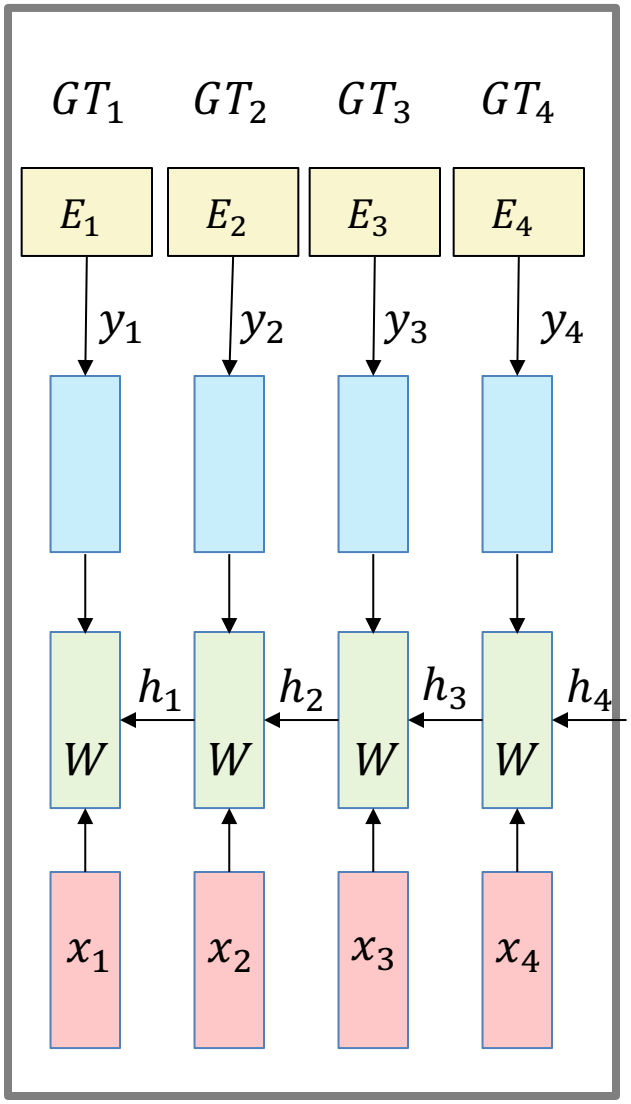
$$y_t = \text{softmax}(W_{hy}h_t)$$

$$E_t = -\log(y_t(GT_t))$$



Source: Justin Johnson

# Backpropagation Through Time

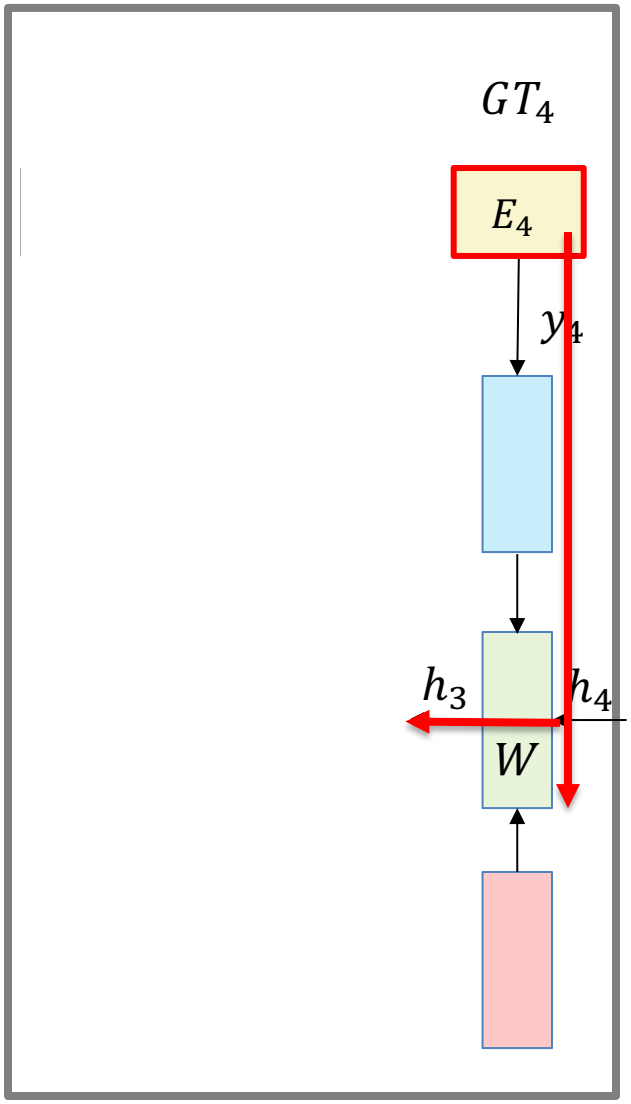



The weight updates are computed for each copy in the unfolded network, then summed (or averaged) and applied to the RNN weights.



Source: Justin Johnson

# Backpropagation Through Time

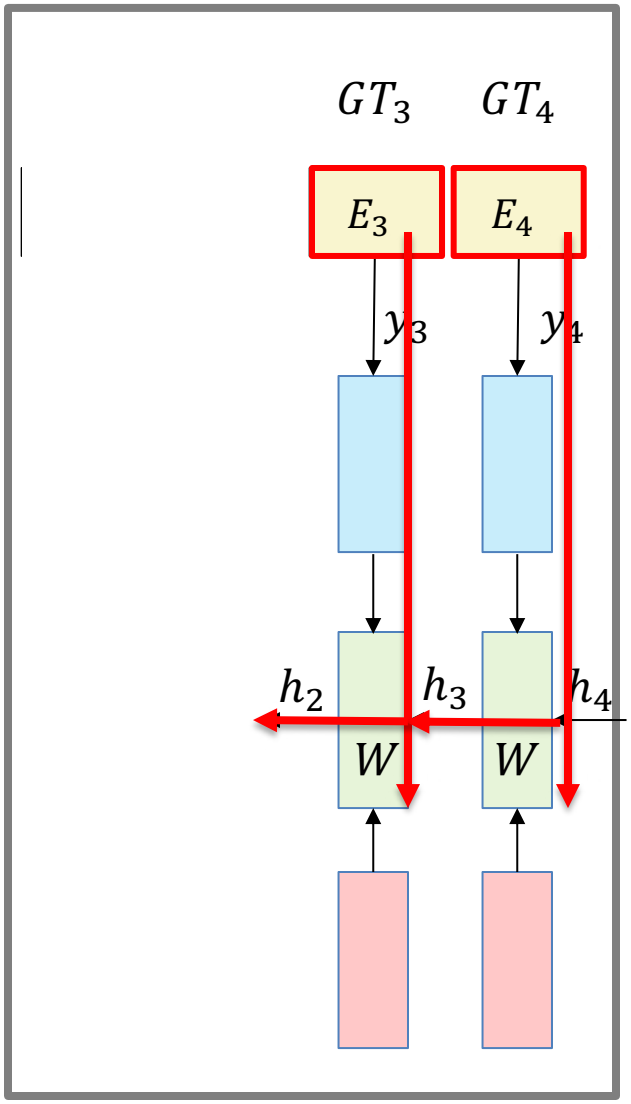



The weight updates are computed for each copy in the unfolded network, then summed (or averaged) and applied to the RNN weights.



Source: Justin Johnson

# Backpropagation Through Time

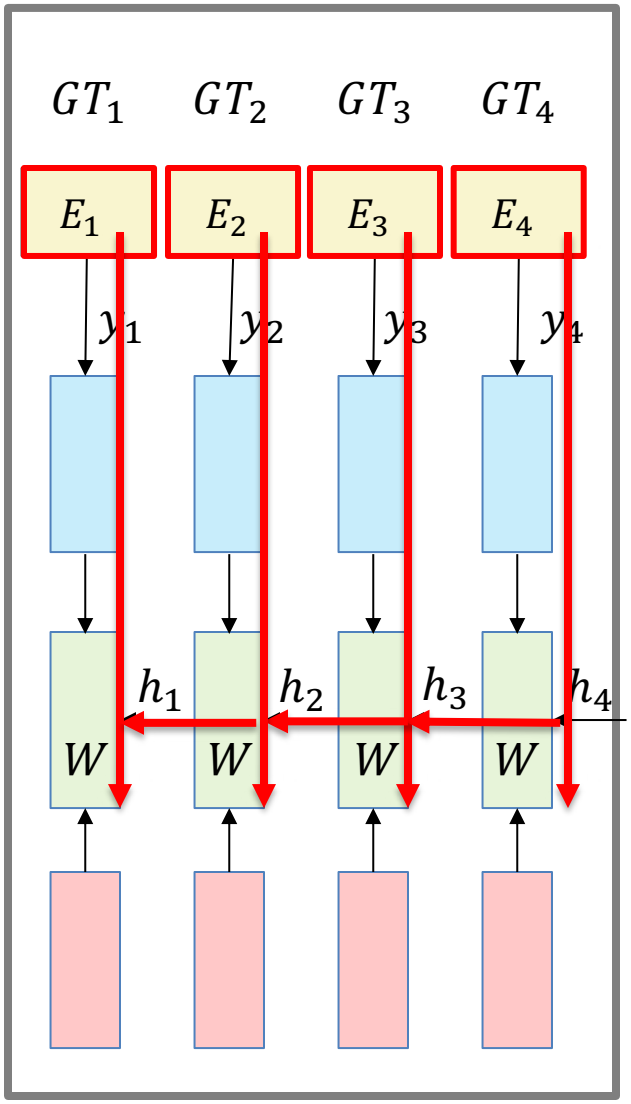



The weight updates are computed for each copy in the unfolded network, then summed (or averaged) and applied to the RNN weights.



Source: Justin Johnson

# Backpropagation Through Time

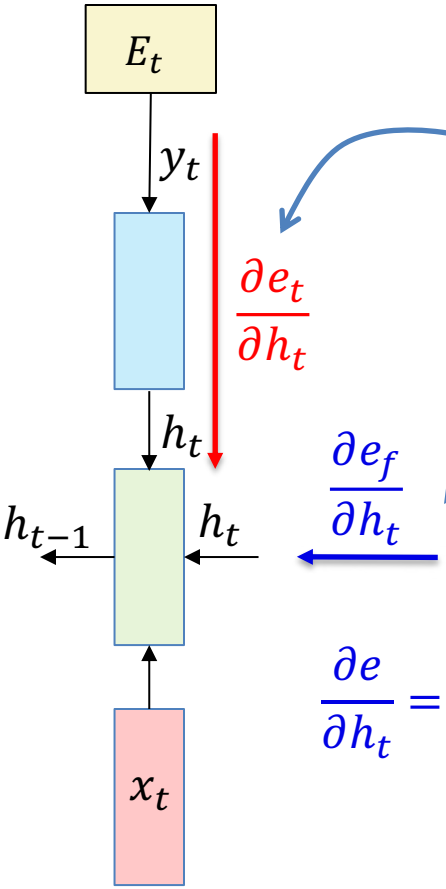



The weight updates are computed for each copy in the unfolded network, then summed (or averaged) and applied to the RNN weights.



Source: Justin Johnson

# Backpropagation Through Time



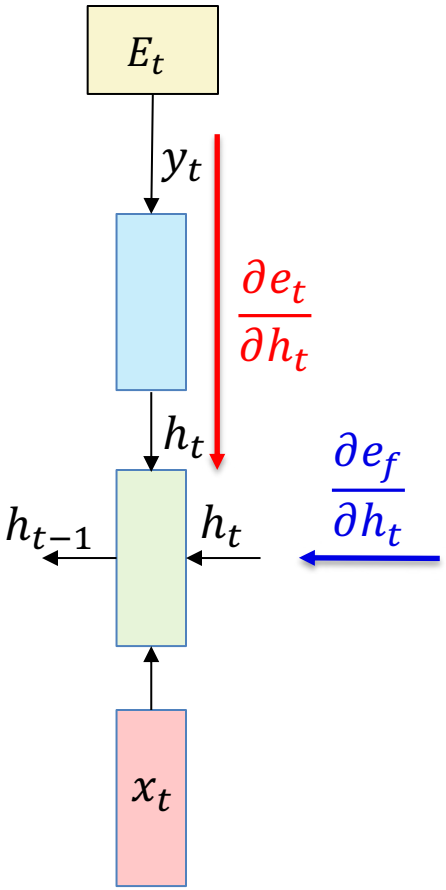

Error from prediction at the current step

Error from predictions at future steps

$\frac{\partial e_f}{\partial h_t}$

$$\frac{\partial e}{\partial h_t} = \left(\frac{\partial e_f}{\partial h_t} + \frac{\partial e_t}{\partial h_t}\right)$$

# Backpropagation Through Time




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

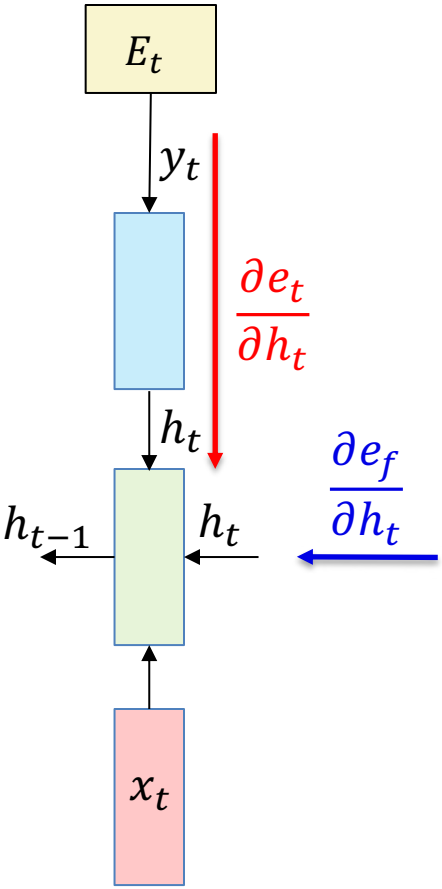
$$\frac{\partial e}{\partial h_t} = \left(\frac{\partial e_f}{\partial h_t} + \frac{\partial e_t}{\partial h_t}\right)$$

TO what is the derivative off tanh

The derivative of the hyperbolic tangent function (tanh) is given by:

derivative of tanh(x) = 1 - tanh^2(x)

# Backpropagation Through Time




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

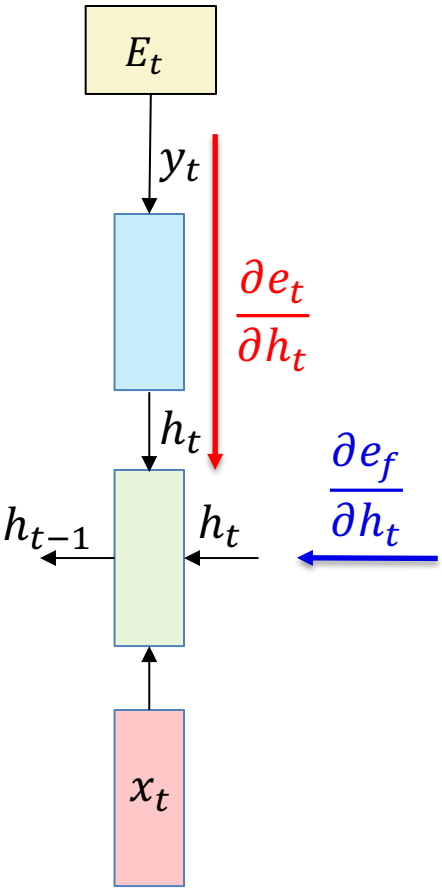
$$\frac{\partial e}{\partial h_t} = \left(\frac{\partial e_f}{\partial h_t} + \frac{\partial e_t}{\partial h_t}\right)$$

$$\frac{\partial e}{\partial W_{hh}} = \frac{\partial e}{\partial h_t} \cdot \left(1 - \tanh^2(W_{xh}x_t + W_{hh}h_{t-1})\right) h_{t-1}^T$$

$$\frac{\partial e}{\partial W_{xh}} = \frac{\partial e}{\partial h_t} \cdot \left(1 - \tanh^2(W_{xh}x_t + W_{hh}h_{t-1})\right) x_t^T$$

$$\frac{\partial e}{\partial h_{t-1}} = \frac{\partial e}{\partial h_t} \cdot W_{hh}^T\left(1 - \tanh^2(W_{xh}x_t + W_{hh}h_{t-1})\right)$$

# Backpropagation Through Time




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

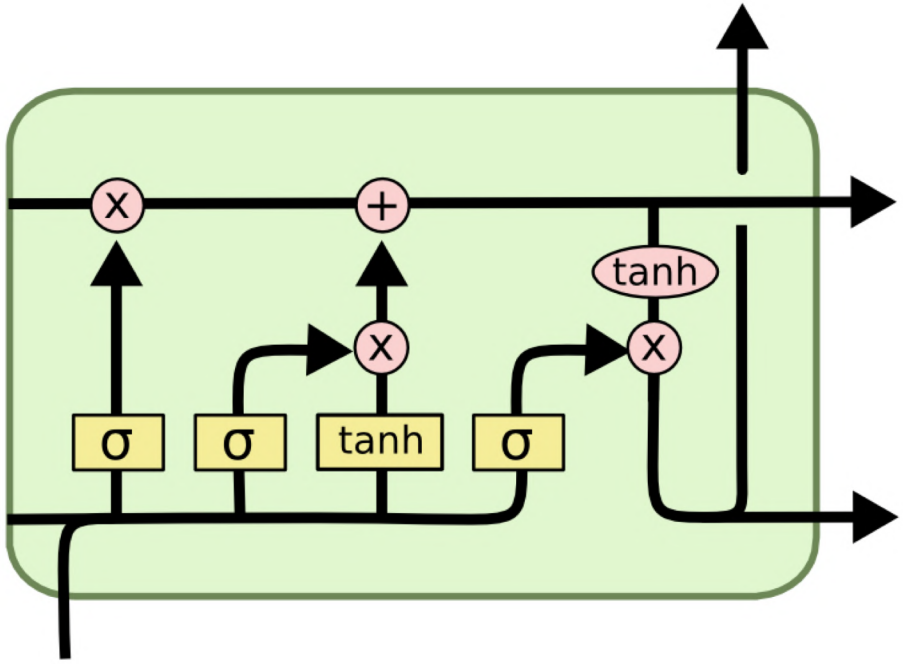
$$\frac{\partial e}{\partial h_{t-1}} = \frac{\partial e}{\partial h_t} \cdot \frac{\partial h_t}{\partial h_{t-1}} =$$

$$= \frac{\partial e}{\partial h_t} \cdot W_{hh}^T\left(1 - \tanh^2(W_{xh}x_t + W_{hh}h_{t-1})\right)$$

$$\frac{\partial e}{\partial h_1} = \left(\frac{\partial e}{\partial h_t}\right)\left(\frac{\partial h_t}{\partial h_{t-1}}\right)\left(\frac{\partial h_{t-1}}{\partial h_{t-2}}\right)\dots\left(\frac{\partial h_2}{\partial h_1}\right)$$

- We have to multiple t times with $W_{hh}$
- Gradients will vanish if largest eigen-value of $W_{hh}$ is less than 1

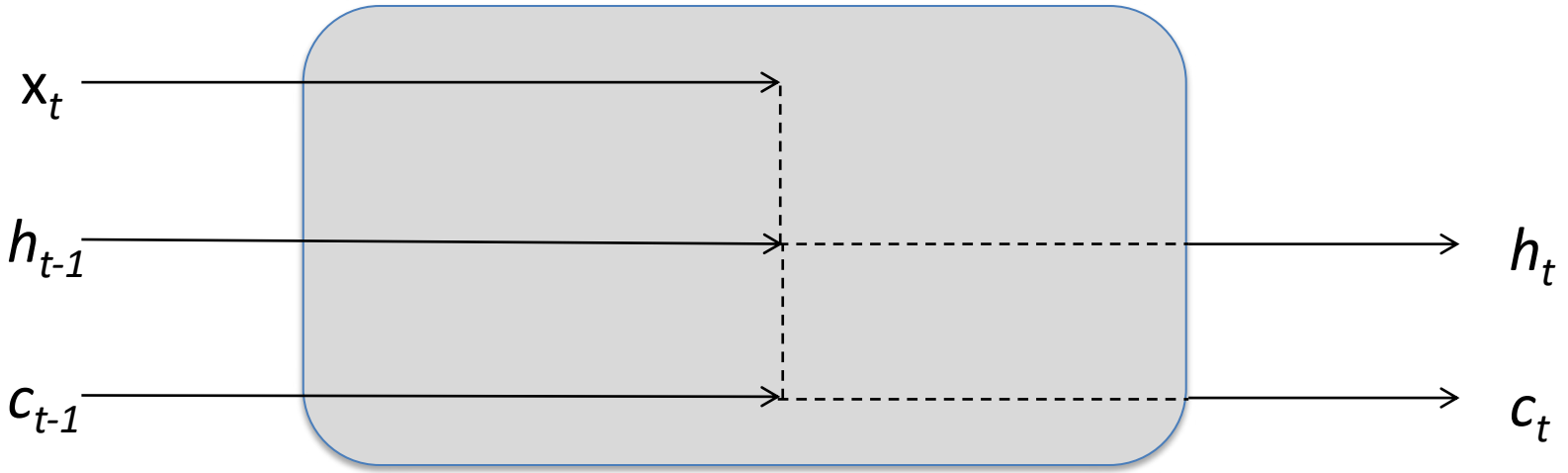
# Long Short-Term Memory (LSTM)



# Long Short-Term Memory (LSTM)

- Add a *memory cell* that is not subject to matrix multiplication or squishing, thereby avoiding gradient decay

- Instead of computing new state as a matrix product with the old state, it rather computes the difference between them. Expressivity is the same, but gradients are better behaved



# Long Short-Term Memory (LSTM)

- Suppose that instead of a matrix multiplication, we had an identity relationship between the hidden states

$$h_t = h_{t-1} + F(x_t)$$

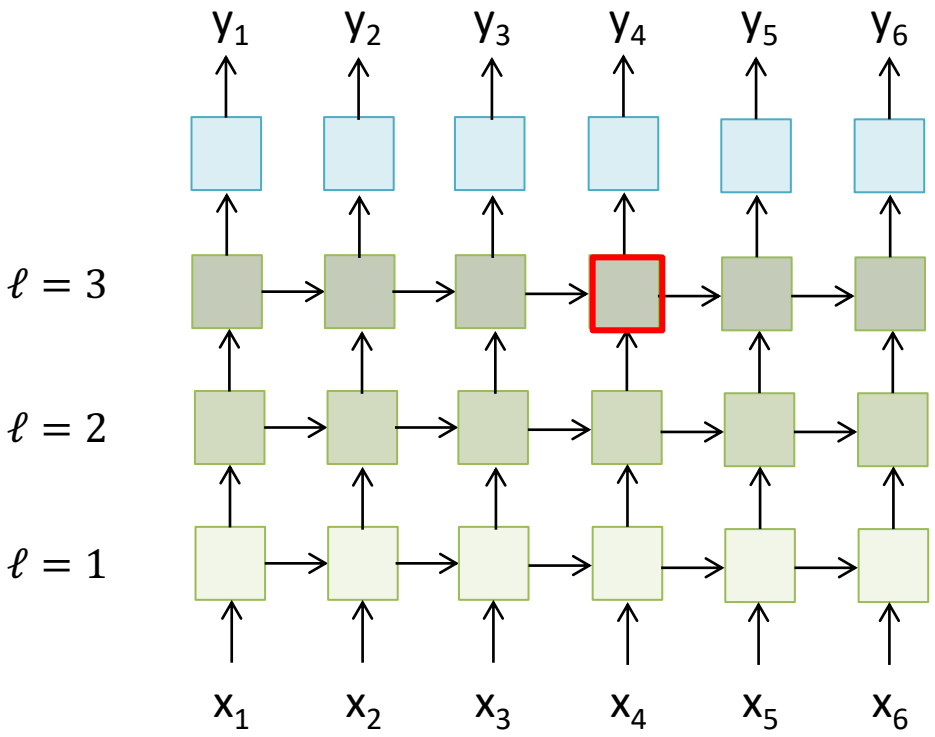
Remember Resnets?

$$\Rightarrow \frac{\partial h_t}{\partial h_{t-1}} = 1$$

- The gradient does not decay as the error is propagated all the way back.
- We will skip this subject for now due to time limitations…

# Multilayer RNN

- We can of course design RNNs with multiple hidden layers



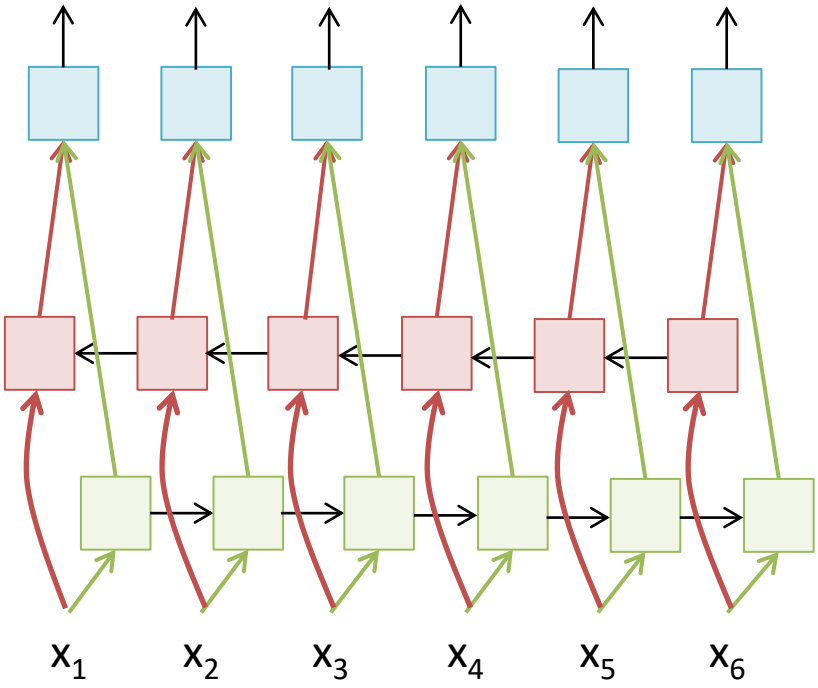

$$h_t^\ell = \tanh\, W^\ell \begin{pmatrix} h_t^{\ell-1} \\ h_{t-1}^\ell \end{pmatrix}$$

$h \in \mathbb{R}^n \qquad W^\ell \in [n \times 2n]$

- Anything goes: skip connections across layers, across time, …

# Bidirectional RNN

- RNNs can process the input sequence in forward and in the reverse direction

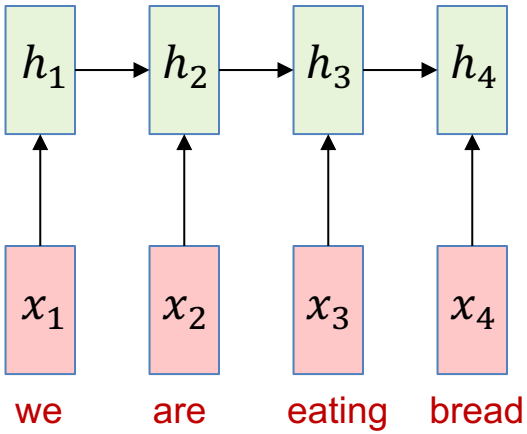



- Popular architecture in speech recognition

# RNN with Attention

# RNN with Attention

- Recall the encoder-decoder RNN for seq-to-seq translation.
- **Input**: Sequence $x_1, x_2, x_3, \ldots, x_T$
- **Output**: Sequence $y_1, y_2, y_3, \ldots, y_{T'}$
- Encoder: $h_t = f_{W_e}(x_t, h_{t-1})$

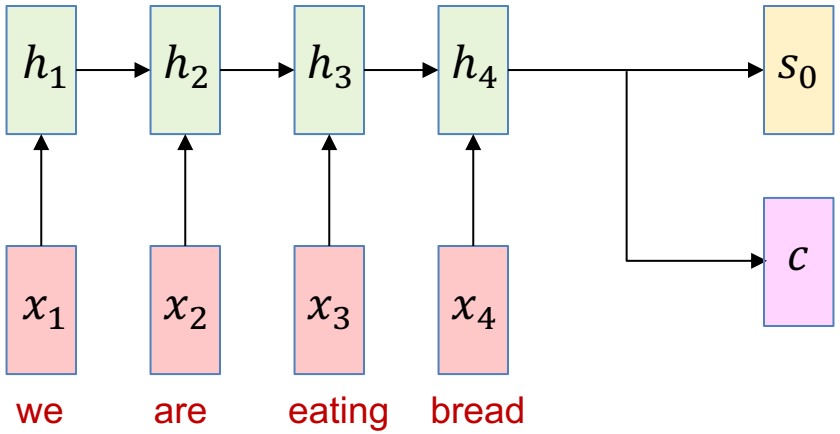



Source: Justin Johnson

# RNN with Attention

- Recall the encoder-decoder RNN for seq-to-seq translation.
- **Input**: Sequence $x_1, x_2, x_3, \ldots, x_T$
- **Output**: Sequence $y_1, y_2, y_3, \ldots, y_{T'}$
- Encoder: $h_t = f_{W_e}(x_t, h_{t-1})$

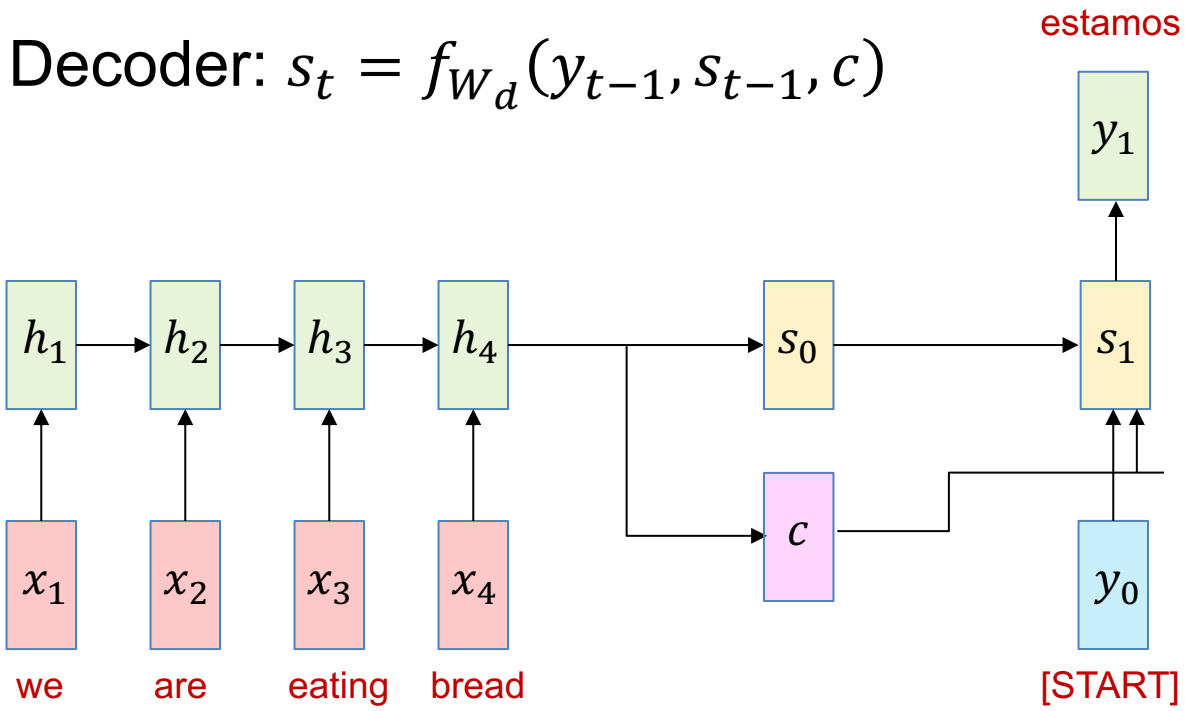
From final hidden state predict:
**Initial decoder state** $s_0$
**Context vector** $c$ (often $c = h_T$)




Source: Justin Johnson

# RNN with Attention

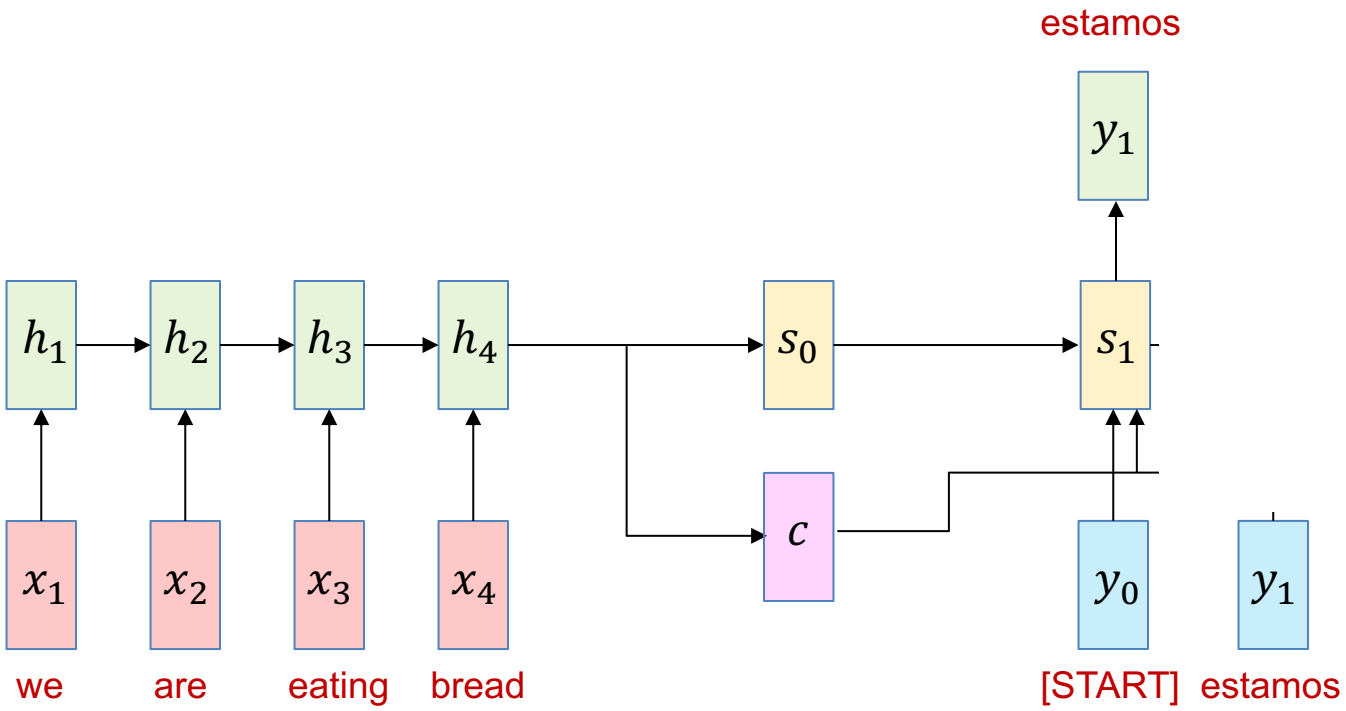
- Recall the encoder-decoder RNN for seq-to-seq translation.
- **Input**: Sequence $x_1, x_2, x_3, \ldots, x_T$
- **Output**: Sequence $y_1, y_2, y_3, \ldots, y_{T'}$
- Encoder: $h_t = f_{W_e}(x_t, h_{t-1})$
- Decoder: $s_t = f_{W_d}(y_{t-1}, s_{t-1}, c)$




Source: Justin Johnson

# RNN with Attention

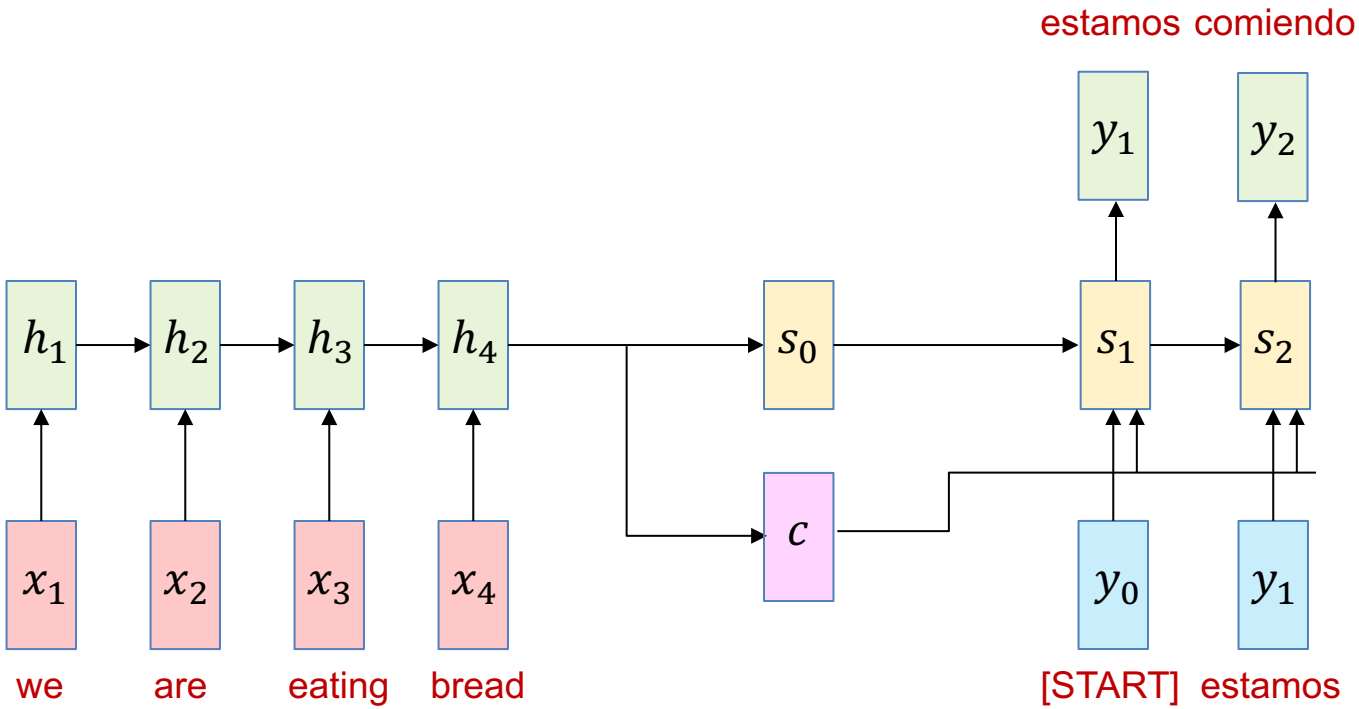
- Recall the encoder-decoder RNN for seq-to-seq translation.
- **Input**: Sequence $x_1, x_2, x_3, \ldots, x_T$
- **Output**: Sequence $y_1, y_2, y_3, \ldots, y_{T'}$
- Decoder: $s_t = f_{W_d}(y_{t-1}, s_{t-1}, c)$




Source: Justin Johnson

# RNN with Attention

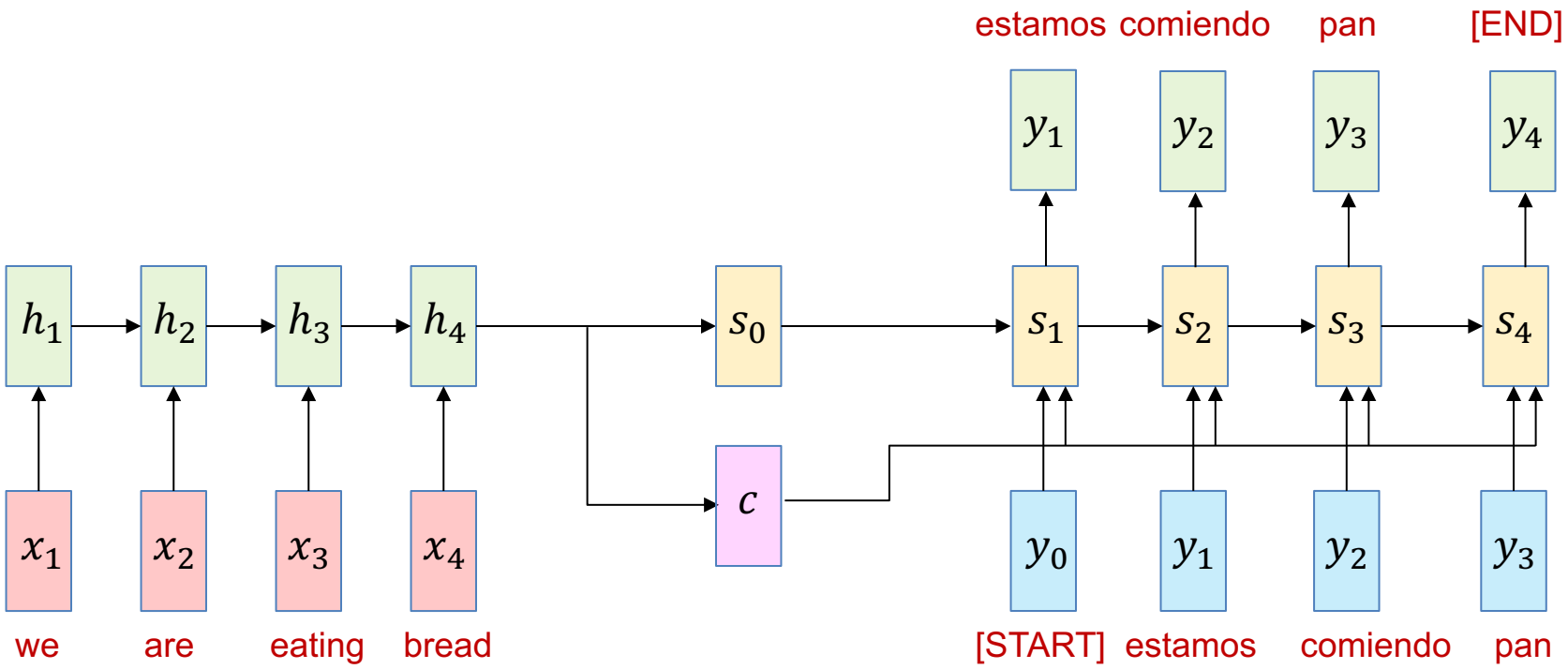
- Recall the encoder-decoder RNN for seq-to-seq translation.
- **Input**: Sequence $x_1, x_2, x_3, \ldots, x_T$
- **Output**: Sequence $y_1, y_2, y_3, \ldots, y_{T'}$
- Decoder: $s_t = f_{W_d}(y_{t-1}, s_{t-1}, c)$




Source: Justin Johnson

# RNN with Attention

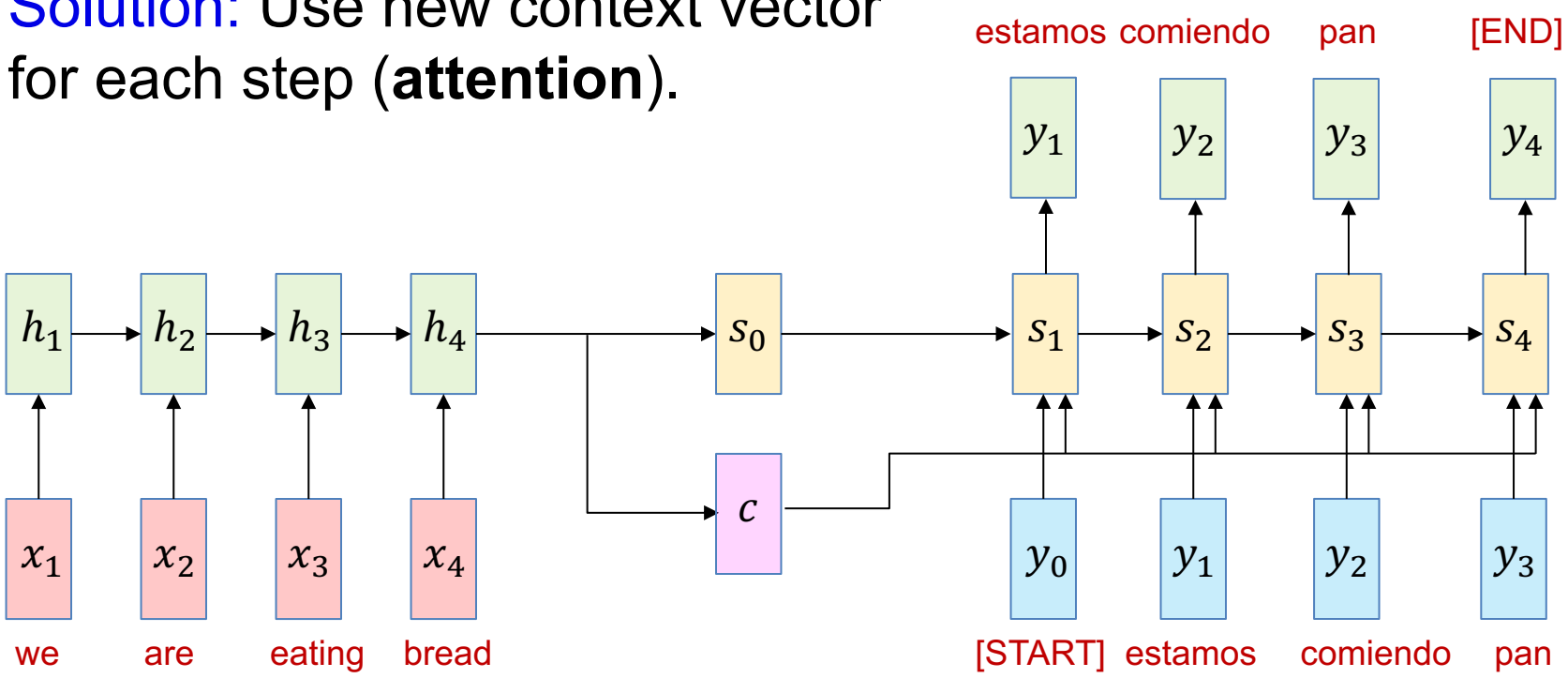
- Recall the encoder-decoder RNN for seq-to-seq translation.
- **Input**: Sequence $x_1, x_2, x_3, \ldots, x_T$
- **Output**: Sequence $y_1, y_2, y_3, \ldots, y_{T'}$
- Decoder: $s_t = f_{W_d}(y_{t-1}, s_{t-1}, c)$




Source: Justin Johnson

# RNN with Attention

- **Problem 1**: The encoder needs to represent the entire input sequence $x_1$, $x_2$, $x_3$, $x_4$ as a single vector c.
- **Problem 2**: the decoder needs to decipher the passed information from this single vector.
- Solution: Use new context vector for each step (**attention**).

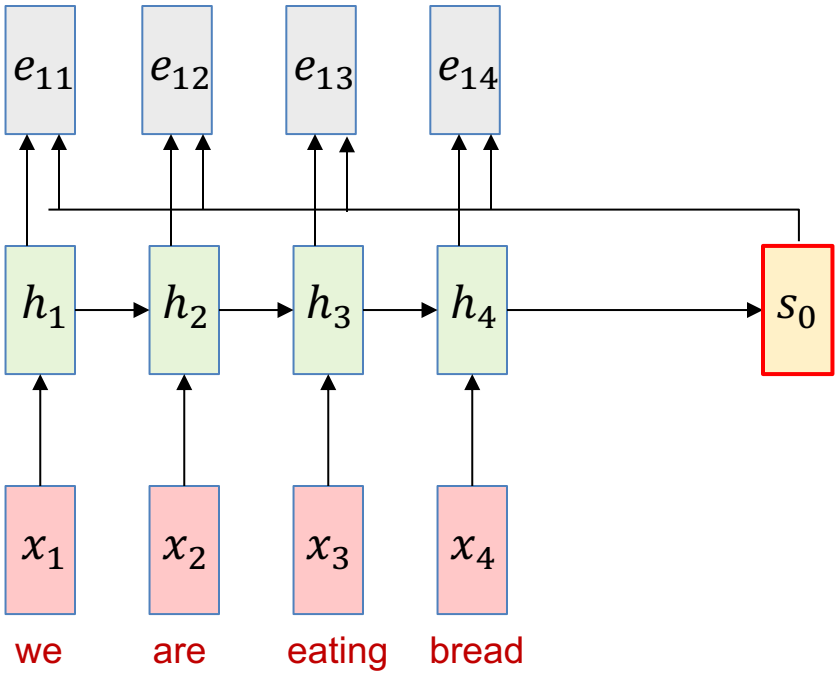



Source: Justin Johnson

# RNN with Attention

**Alignment score**:

$$e_{t,i} = f_{att}(s_{t-1}, h_i)$$

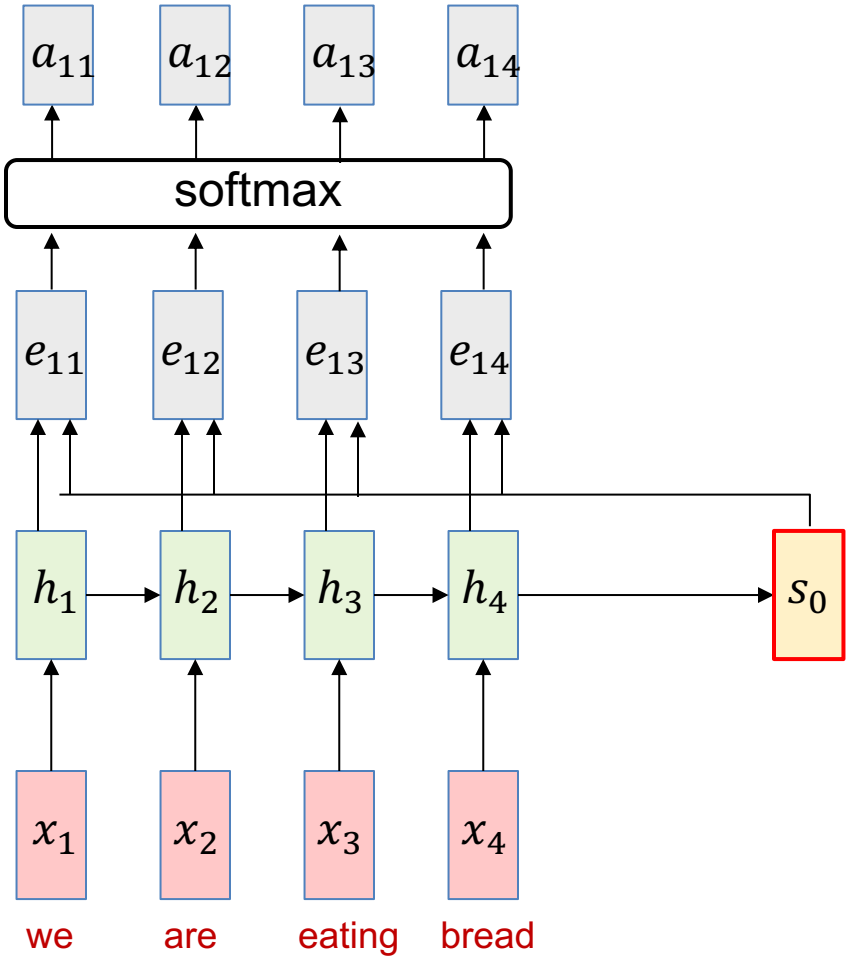



Source: Justin Johnson

# RNN with Attention

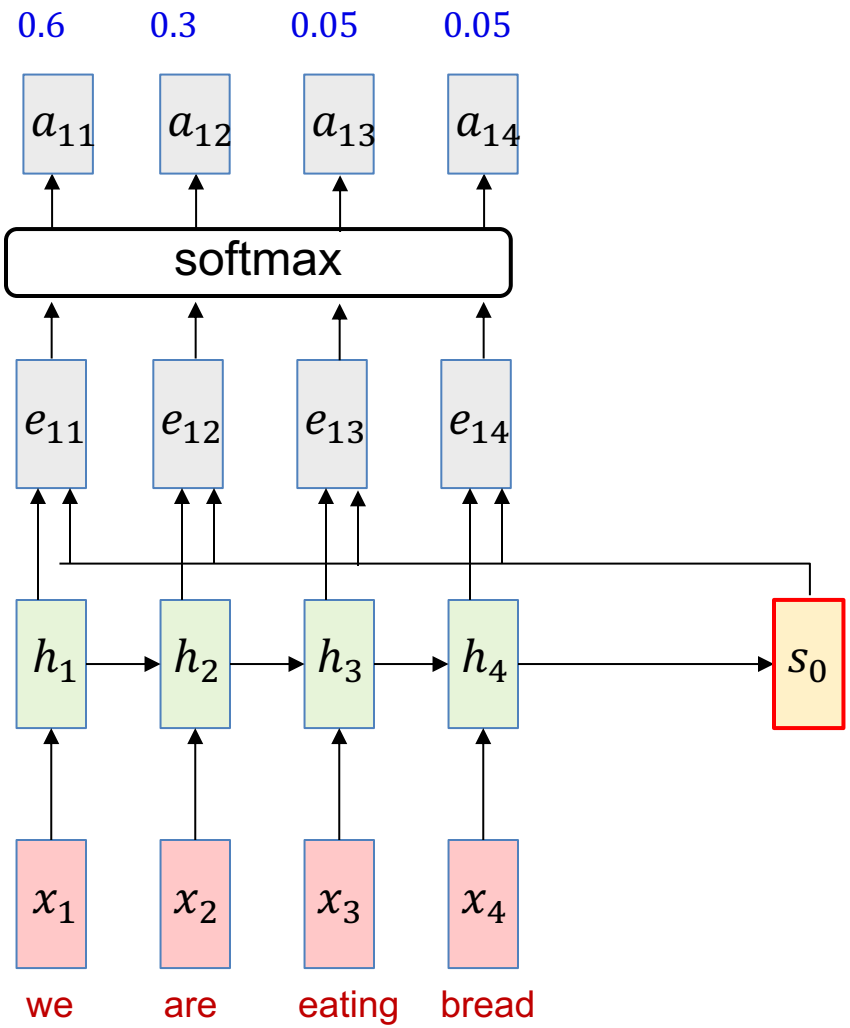
**Alignment score**:

$$e_{t,i} = f_{att}(s_{t-1}, h_i)$$




Source: Justin Johnson

# RNN with Attention



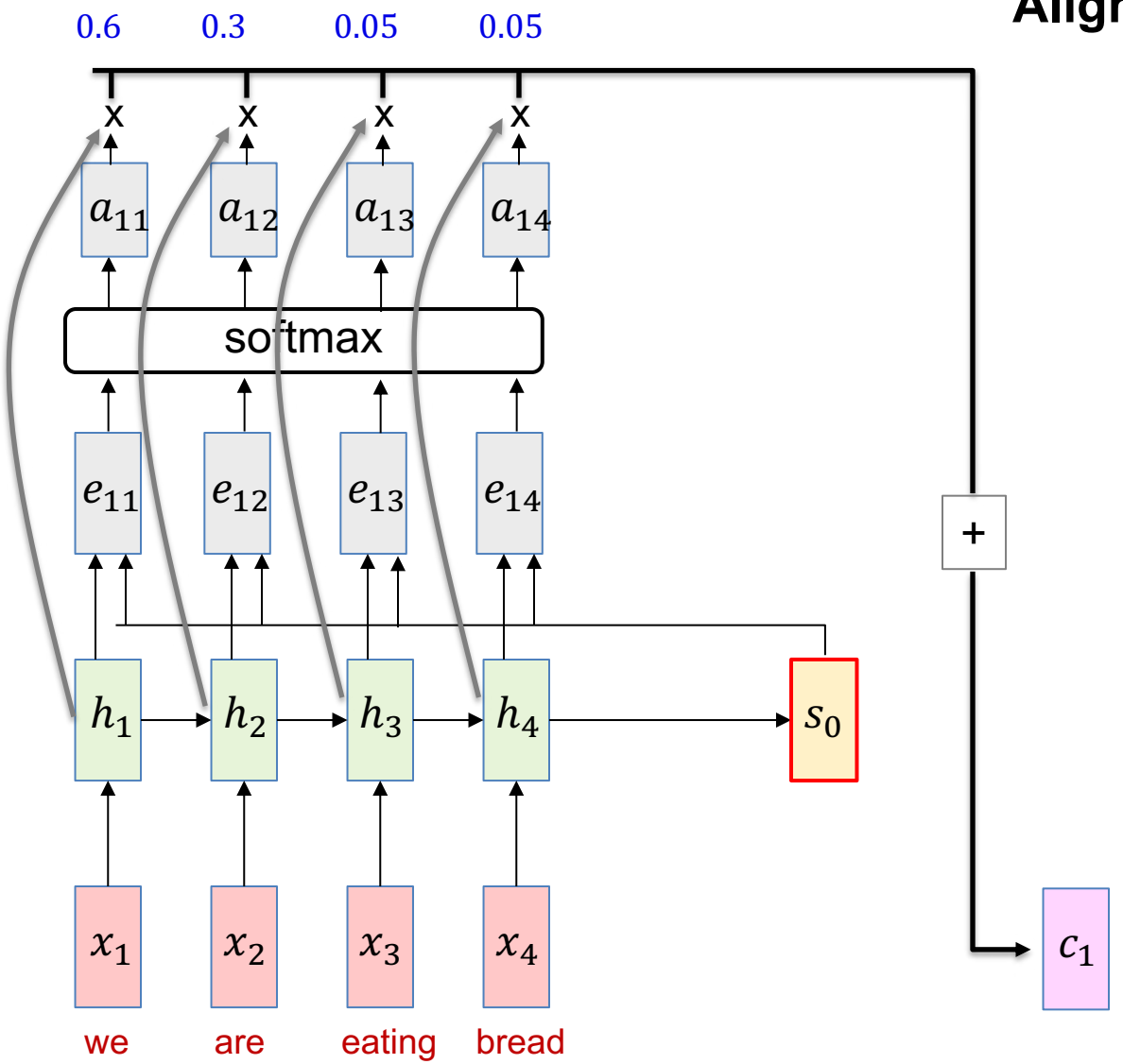

**Alignment score**:

$$e_{t,i} = f_{att}(s_{t-1}, h_i)$$

- Attention network $f_{att}$ consists of a small MLP
- The attention network is trainable

Source: Justin Johnson

# RNN with Attention




**Alignment score**:

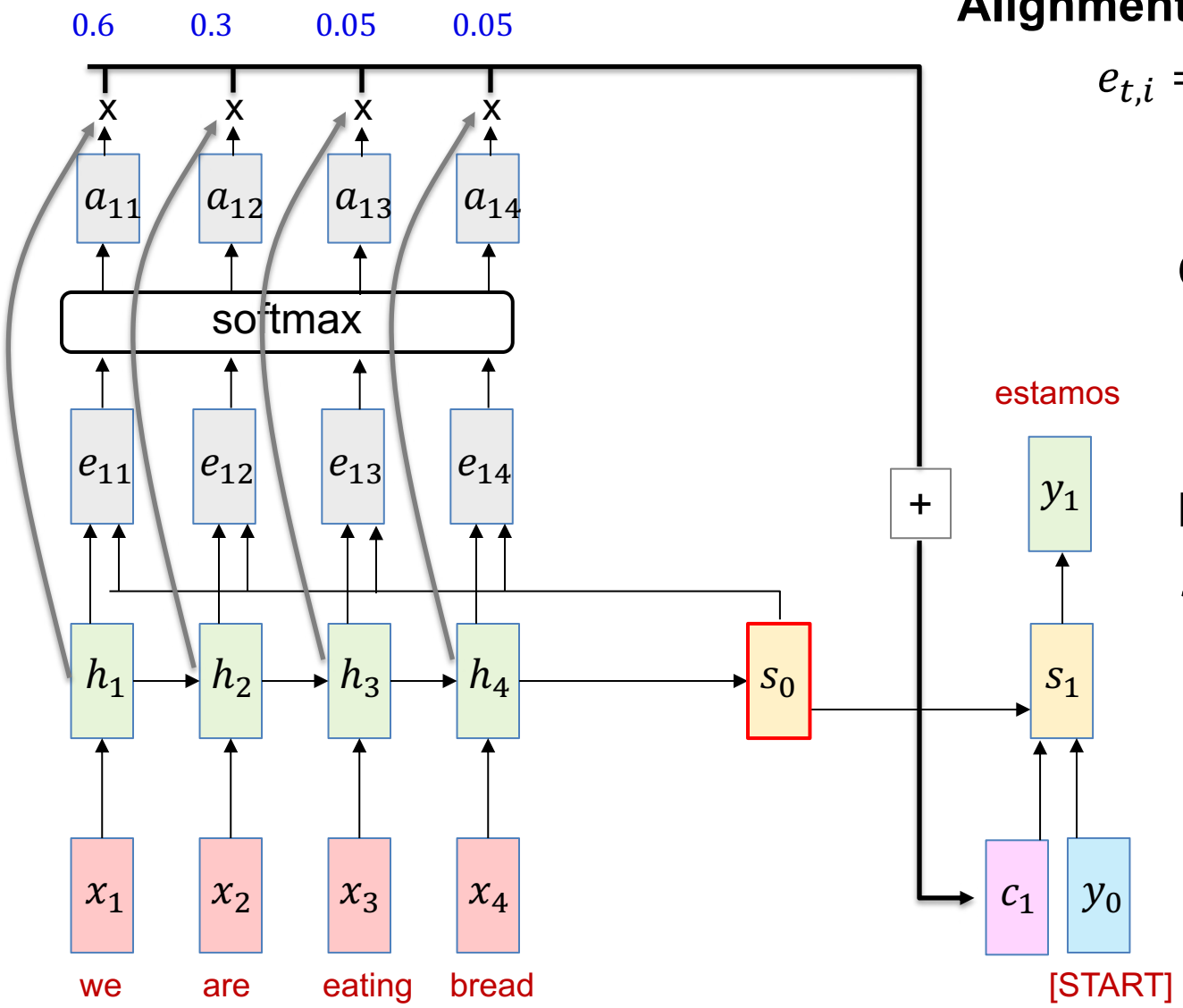
$$e_{t,i} = f_{att}(s_{t-1}, h_i)$$

**Context**:

$$c_t = \sum_i a_{t,i} \cdot h_i$$

Source: Justin Johnson

# RNN with Attention




**Alignment score**:

$$e_{t,i} = f_{att}(s_{t-1}, h_i)$$

**Context**:

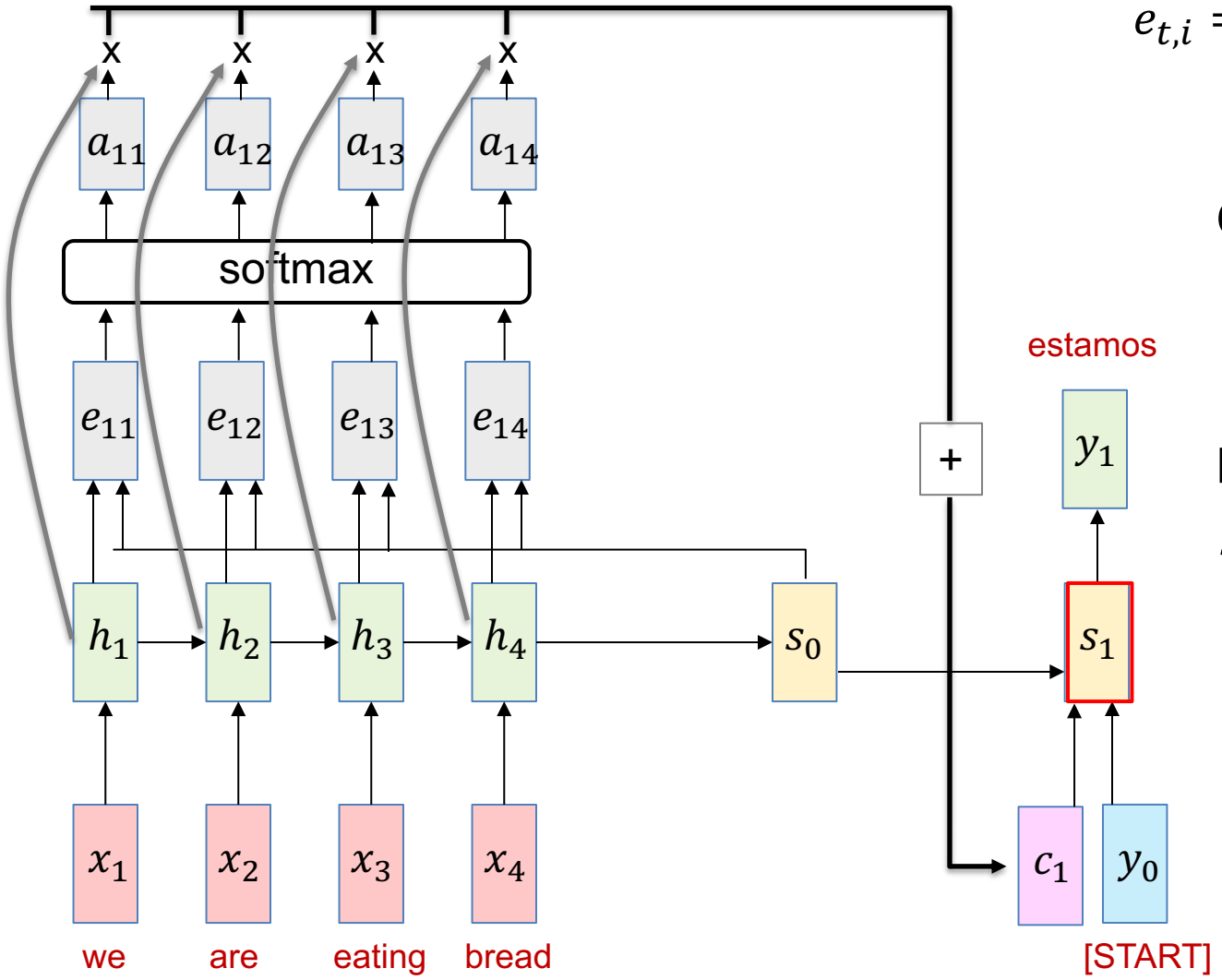
$$c_t = \sum_i a_{t,i} \cdot h_i$$

**Decoder**:

$$s_t = f_{W_d}(y_{t-1}, s_{t-1}, c_t)$$

Source: Justin Johnson

# RNN with Attention




**Alignment score**:

$$e_{t,i} = f_{att}(s_{t-1}, h_i)$$

**Context**:

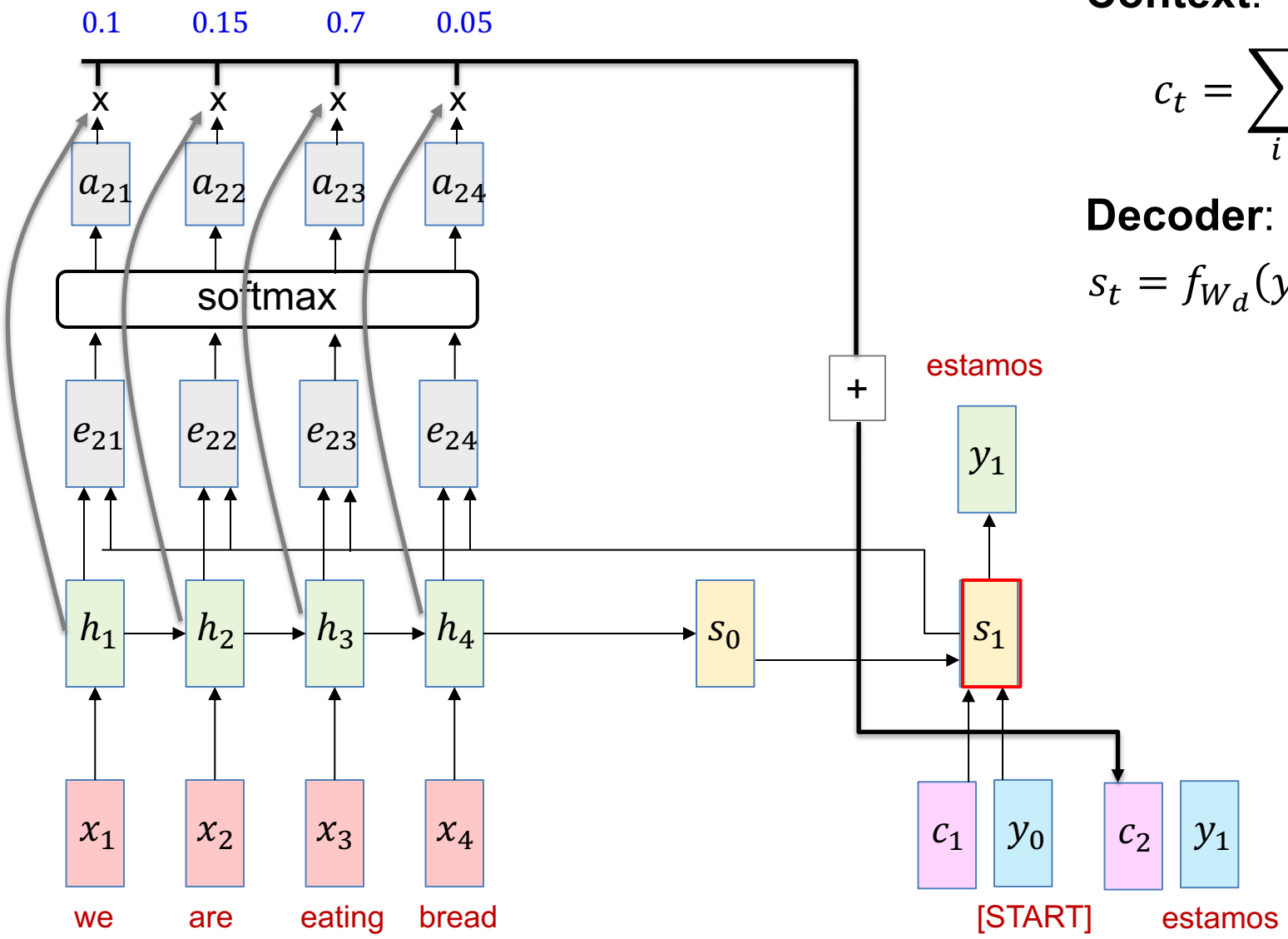
$$c_t = \sum_i a_{t,i} \cdot h_i$$

**Decoder**:

$$s_t = f_{W_d}(y_{t-1}, s_{t-1}, c_t)$$

Source: Justin Johnson

# RNN with Attention




**Context**:

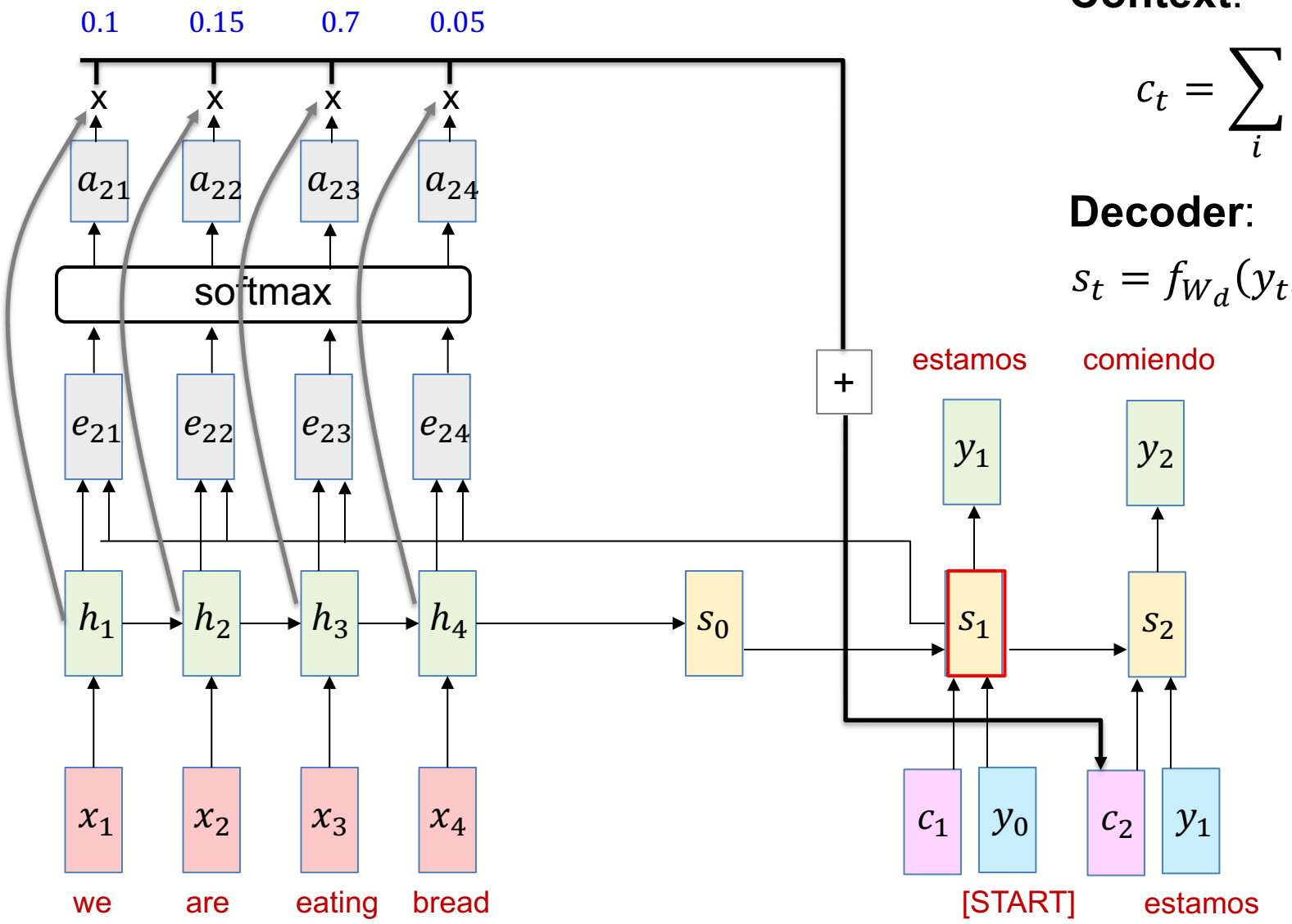
$$c_t = \sum_i a_{t,i} \cdot h_i$$

**Decoder**:

$$s_t = f_{W_d}(y_{t-1}, s_{t-1}, c_t)$$

Source: Justin Johnson

# RNN with Attention




**Context**:

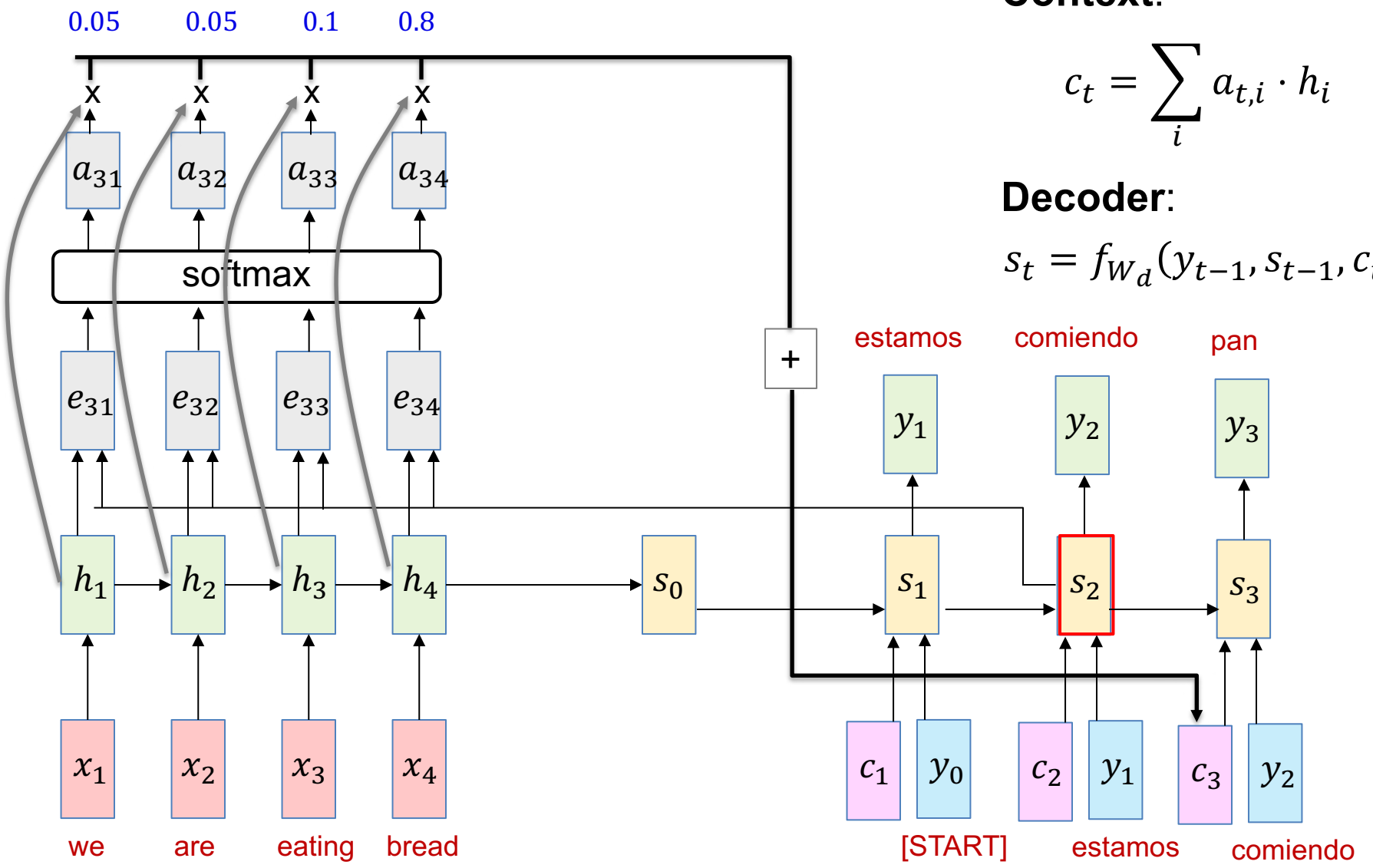
$$c_t = \sum_i a_{t,i} \cdot h_i$$

**Decoder**:

$$s_t = f_{W_d}(y_{t-1}, s_{t-1}, c_t)$$

Source: Justin Johnson

# RNN with Attention



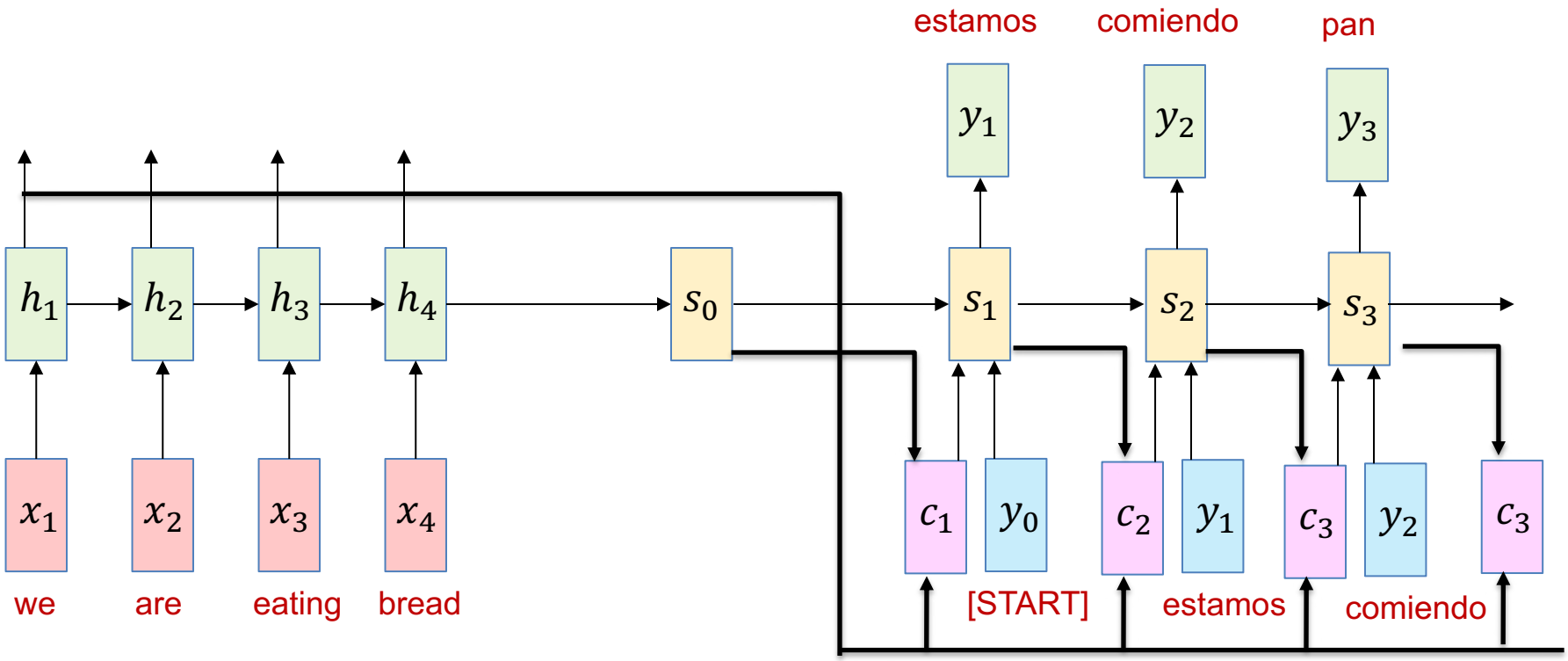

**Context**:

$$c_t = \sum_i a_{t,i} \cdot h_i$$

**Decoder**:

$$s_t = f_{W_d}(y_{t-1}, s_{t-1}, c_t)$$

# RNN with Attention



# RNN with Attention: Summary

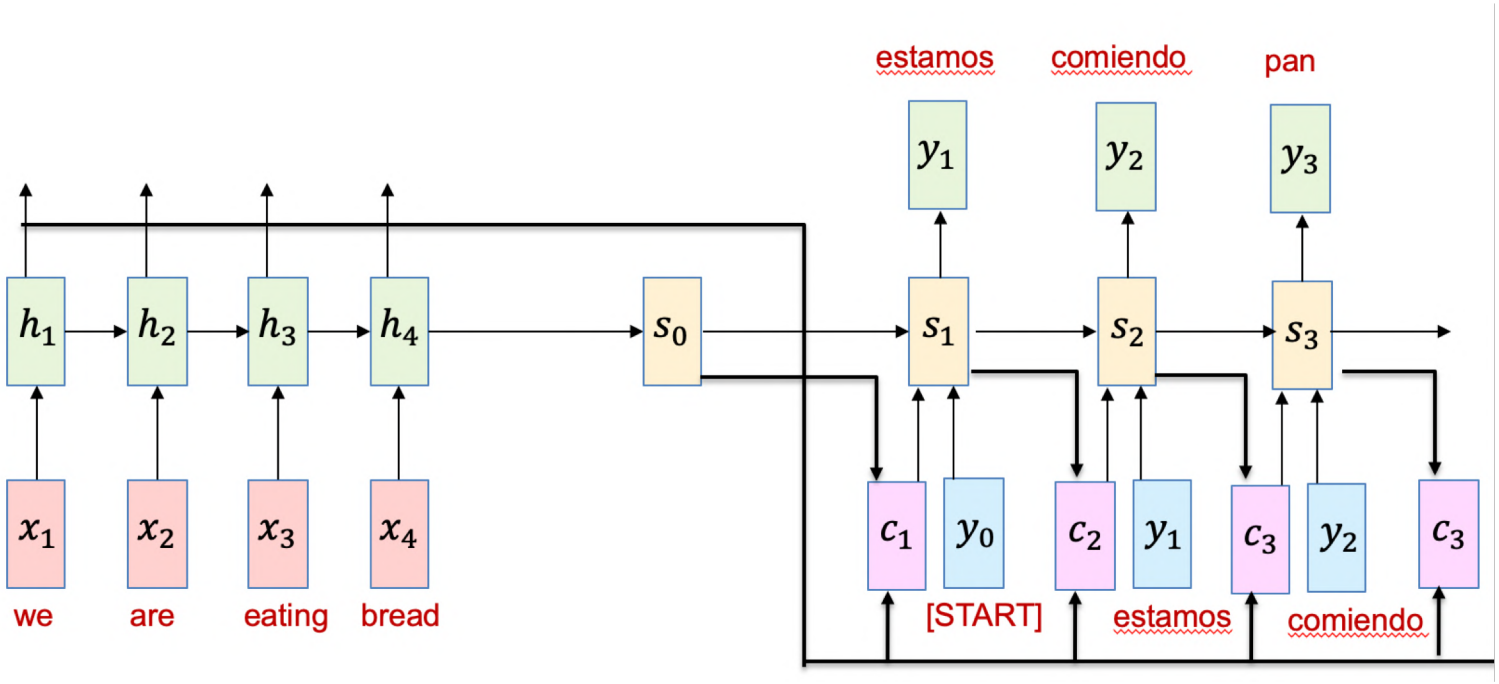
- The encoder's output vectors are $h_1$, $h_2$, ... where:

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

- The decoder outputs are $s_0$, $s_1$, $s_2$, …

$$s_t = f_{W_d}(y_{t-1}, s_{t-1}, c_t)$$

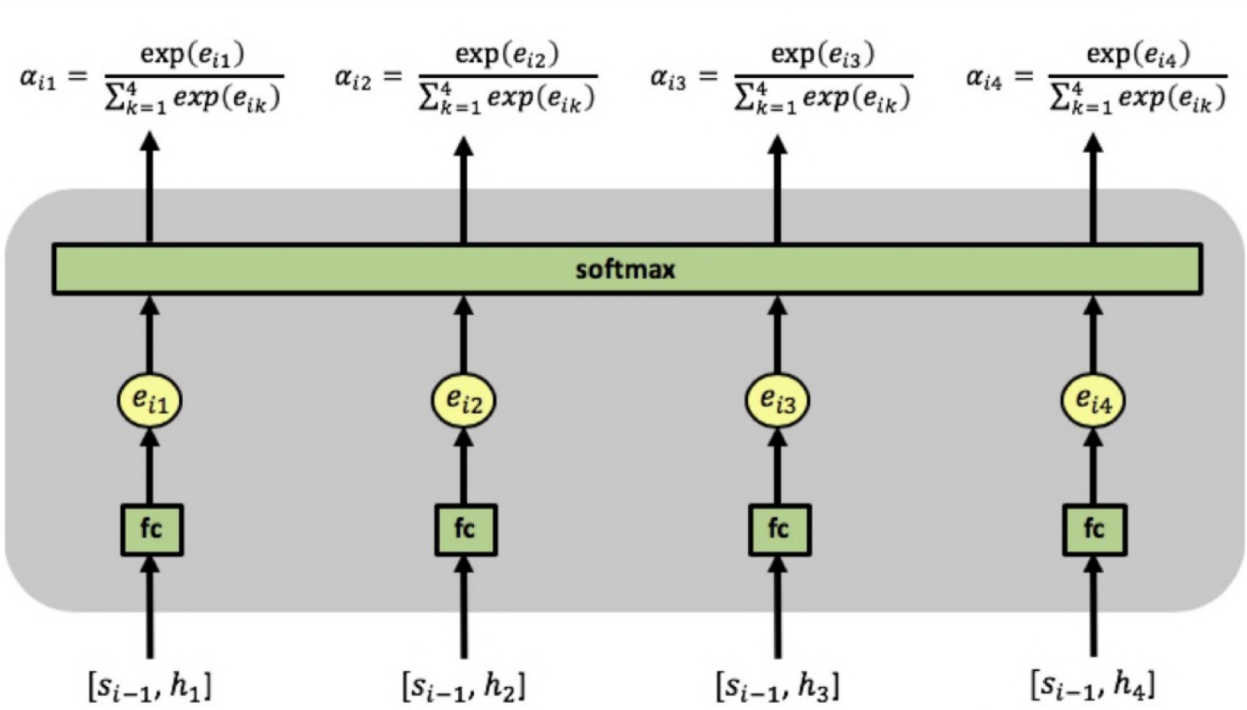
where $c_t$ is an attention input called **context vector**



# RNN with Attention: Summary

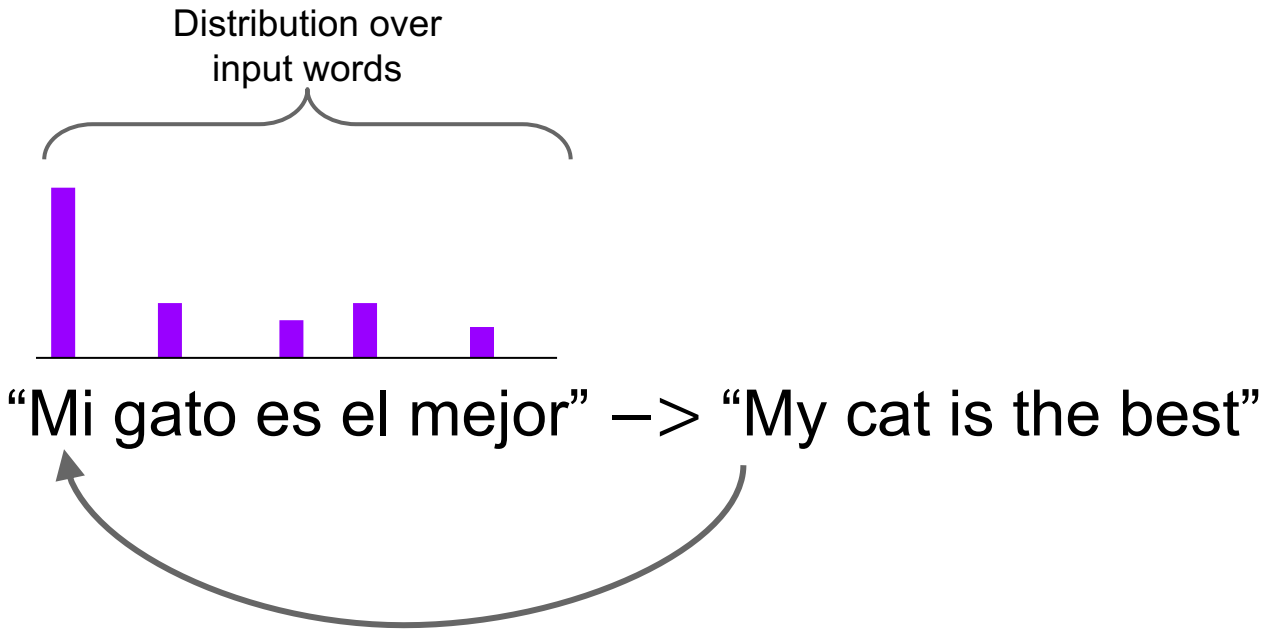
- The attention model consists of alignment scores calculated by a small fully connected network: $e_{t,i} = FC(s_{t-1}, h_i)$
- Alignment scores are normalized using softmax: $\alpha_{t,j} = \frac{\exp e_{t,j}}{\sum_k \exp e_{t,k}}$
- A **context vector** $c_t$ is calculated using the normalized scores:

$$c_t = \sum_{j=1..4} a_{t,j} h_j$$



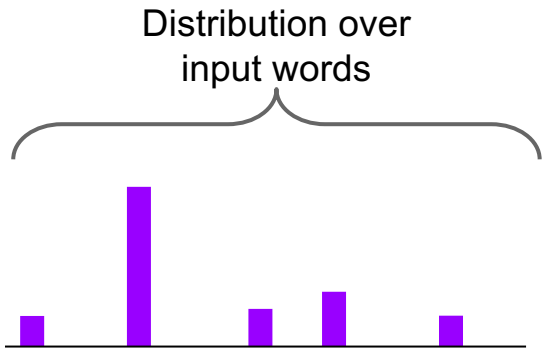
# RNN with Attention

$$\alpha_{t,j} = \frac{\exp e_{i,j}}{\sum_k \exp e_{i,k}} \quad \text{where} \quad e_{t,j} = FC(s_{t-1}, h_j)$$



# RNN with Attention

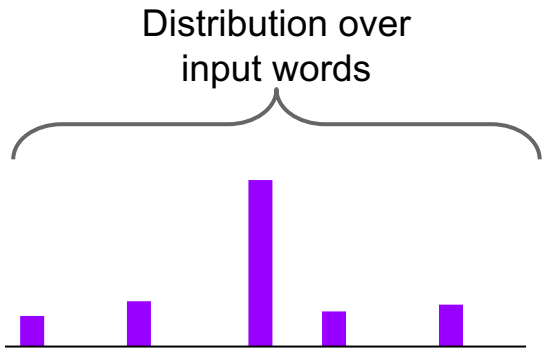
$$\alpha_{t,j} = \frac{\exp e_{i,j}}{\sum_k \exp e_{i,k}} \quad \text{where} \quad e_{t,j} = FC(s_{t-1}, h_j)$$




“Mi gato es el mejor” –> “My cat is the best”

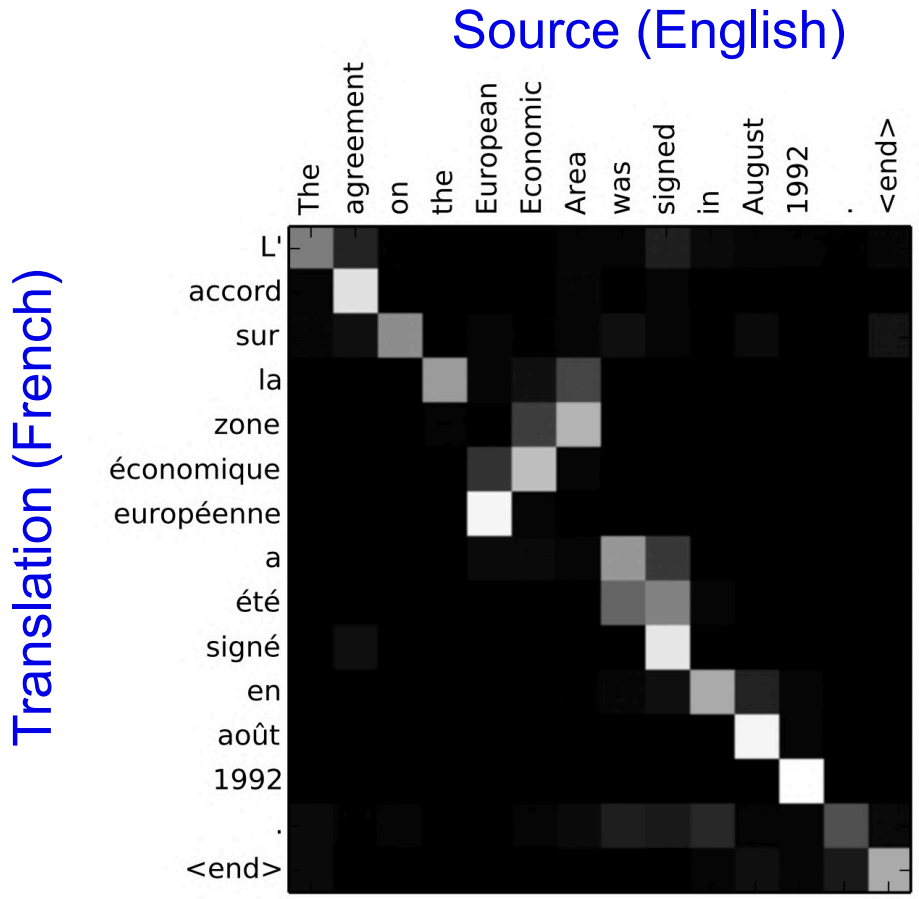
# RNN with Attention

$$\alpha_{t,j} = \frac{\exp e_{i,j}}{\sum_k \exp e_{i,k}} \quad \text{where} \quad e_{t,j} = FC(s_{t-1}, h_j)$$




“Mi gato es el mejor” –> “My cat is the best”

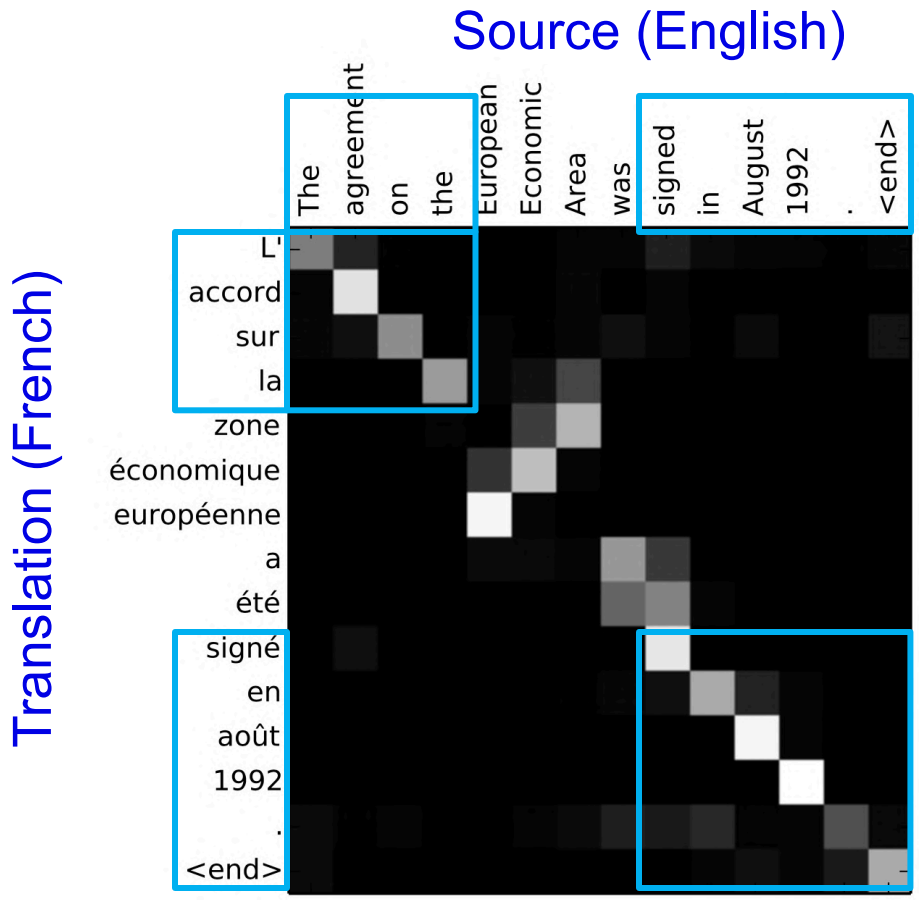
# RNN with Attention




Note, the English "European Economic Area" is reversed in French ("zone économique européenne")

Image from: Jay Alammar

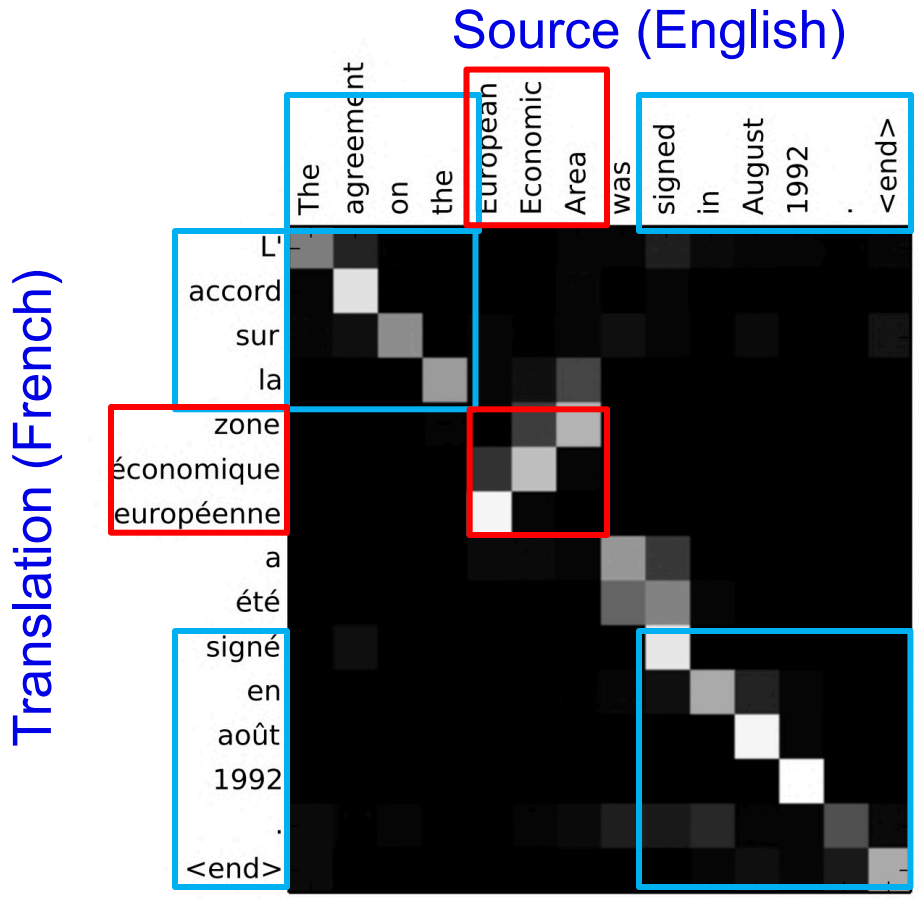
# RNN with Attention




Note, the English "European Economic Area" is reversed in French ("zone économique européenne")

Image from: Jay Alammar

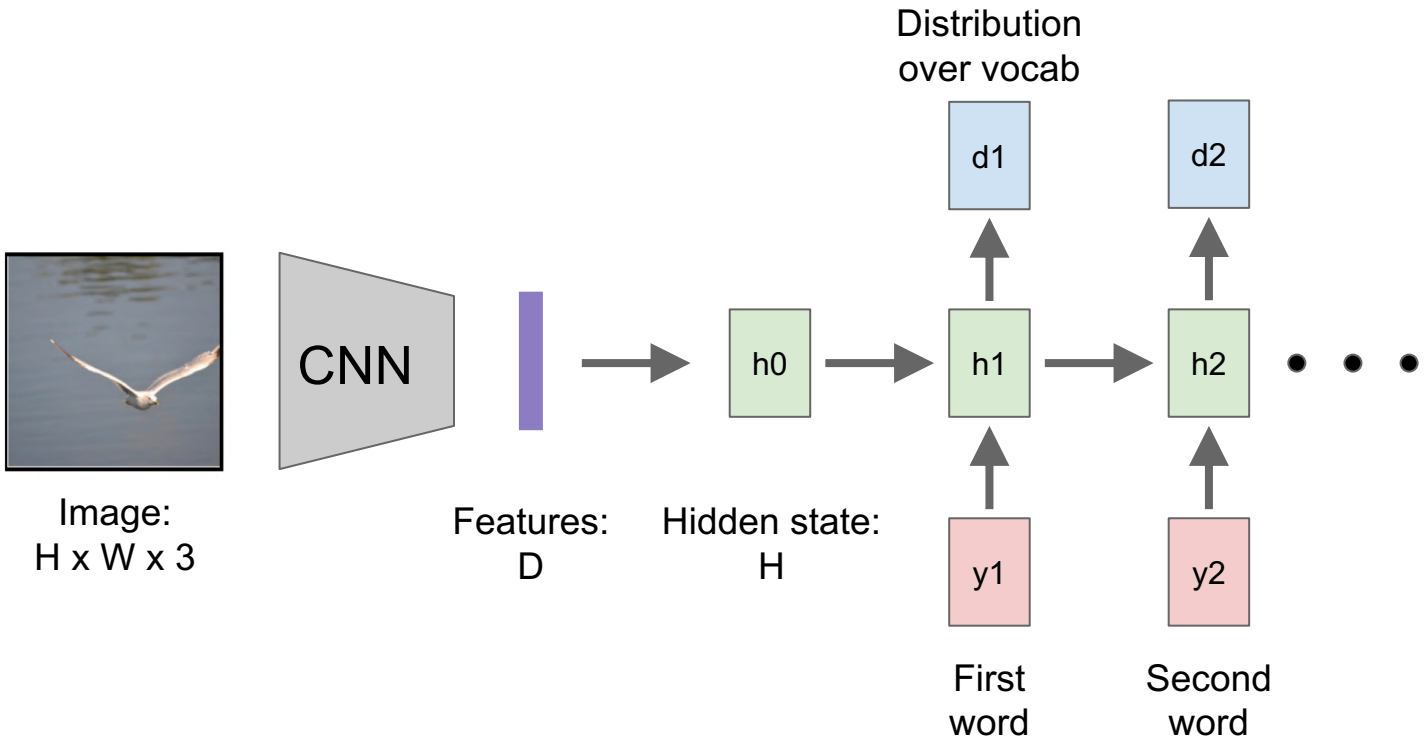
# RNN with Attention




Note, the English "European Economic Area" is reversed in French ("zone économique européenne")

Image from: [Jay Alammar]

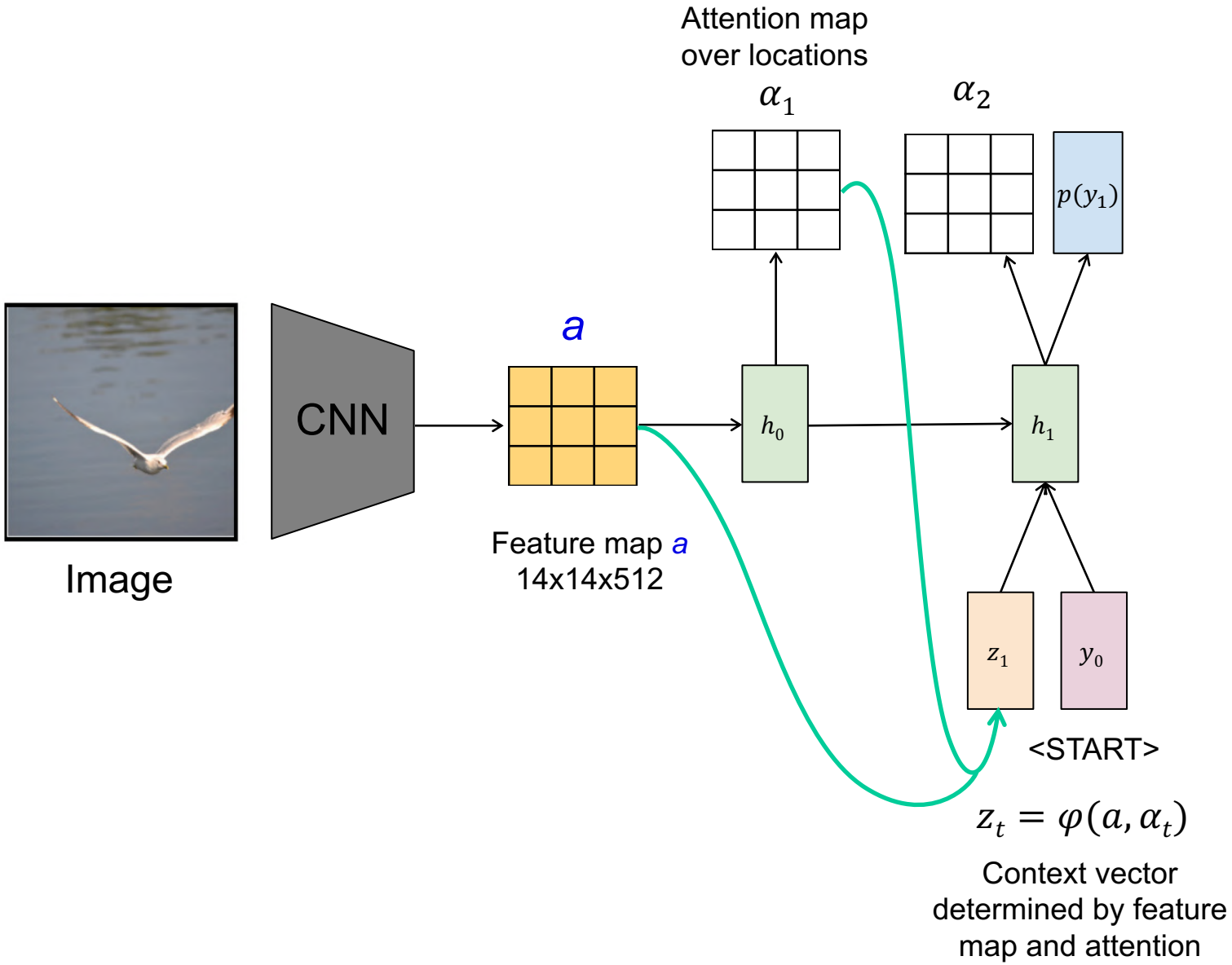
# Attention in Image Captioning




- RNN only looks at whole image, once.
- What if the RNN looks at different parts of the image at each time-step?

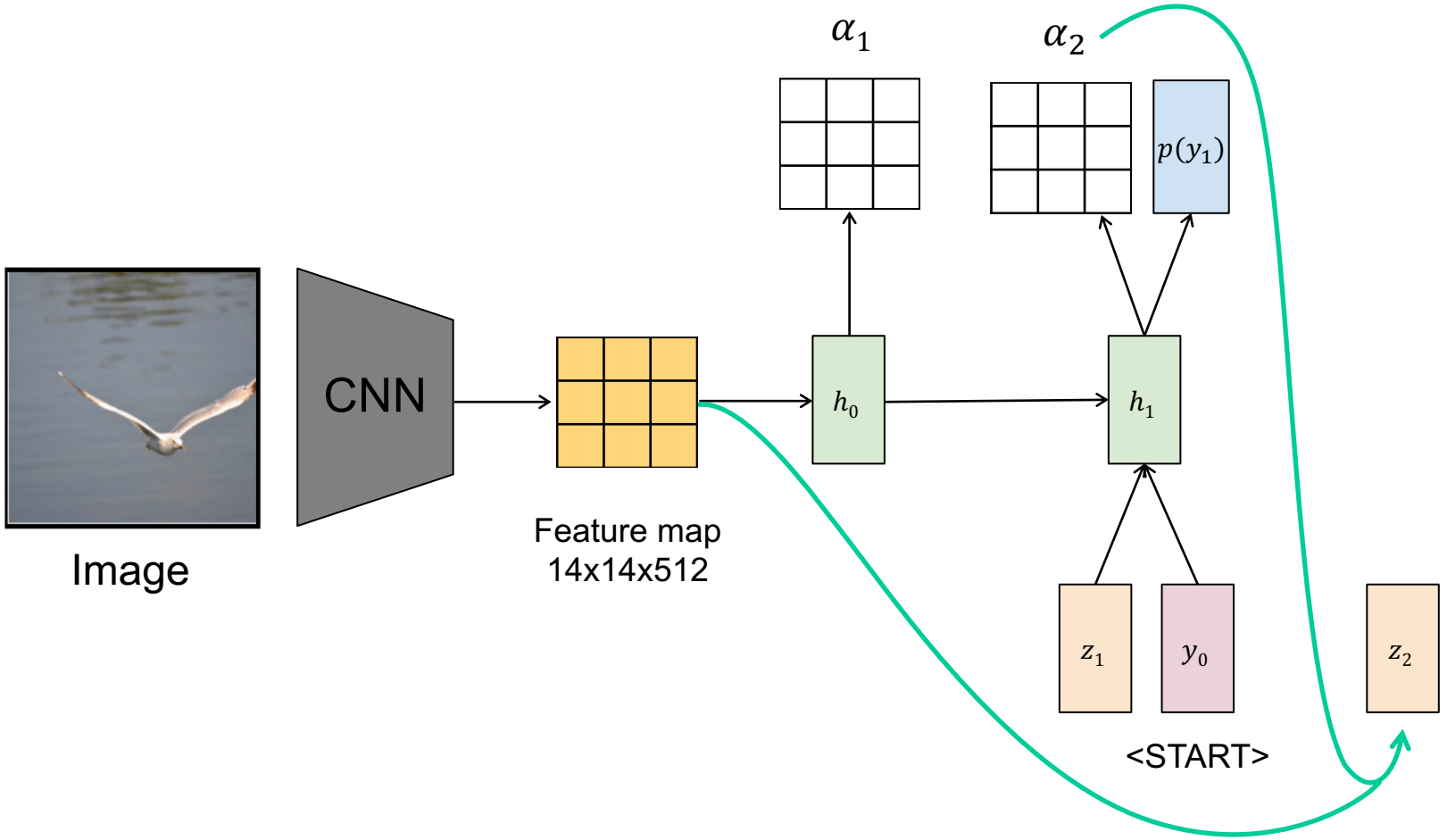
Adapted from Stanford cs231n

# Attention in Image Captioning



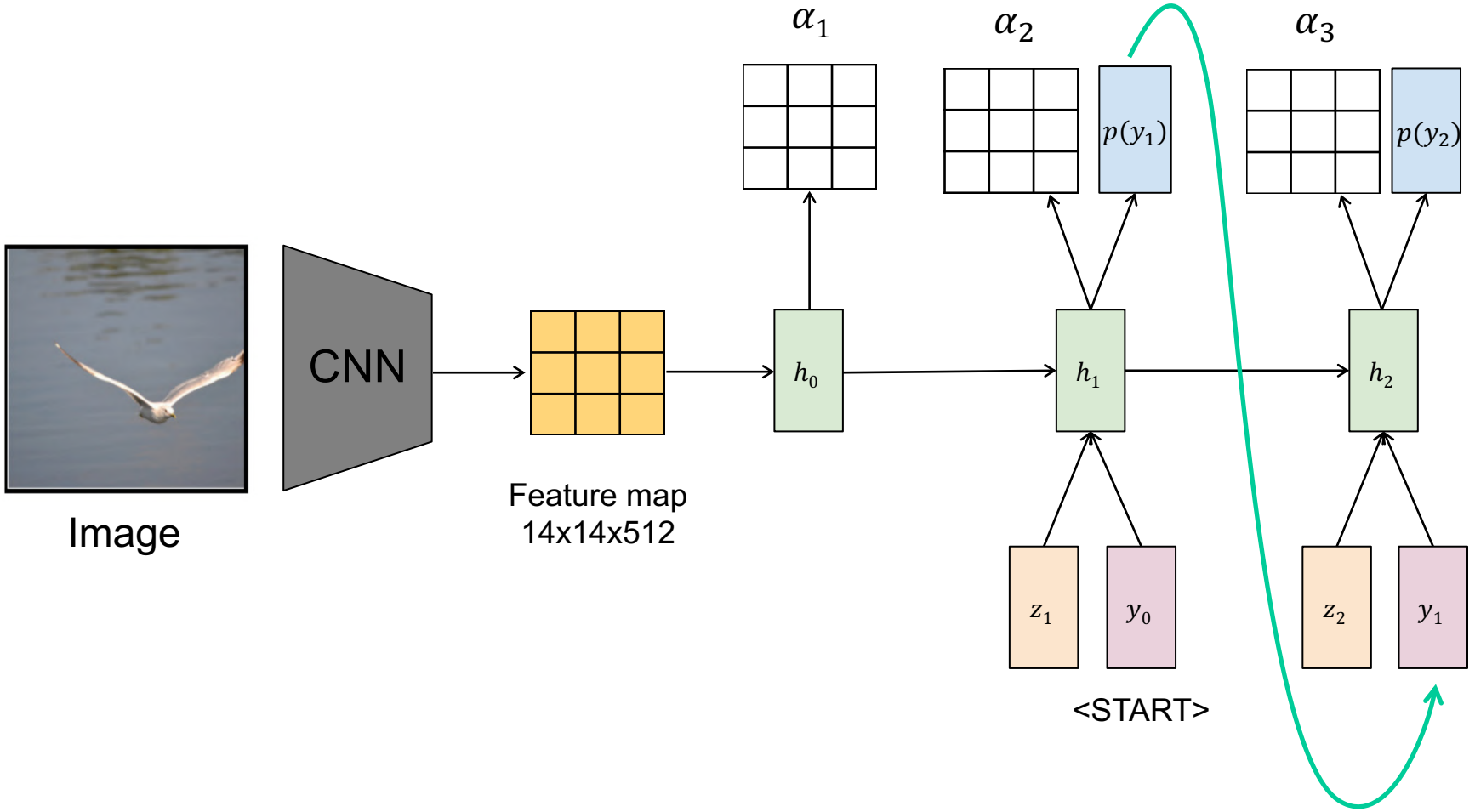

Adapted from Stanford cs231n

# Attention in Image Captioning




Adapted from Stanford cs231n

# Attention in Image Captioning




Adapted from [Stanford cs231n]

# Attention in Image Captioning

Image

CNN

| | | |
|---|---|---|
| $a_1$ | $a_2$ | $a_3$ |
| $a_4$ | $a_5$ | $a_6$ |
| $a_7$ | $a_8$ | $a_9$ |

Grid of features
(Each $a_i$ is D-dimensional)

Context vector $z_t$
(D-dimensional)

From RNN:

| | | |
|---|---|---|
| $\alpha_{t,1}$ | $\alpha_{t,2}$ | $\alpha_{t,3}$ |
| $\alpha_{t,4}$ | $\alpha_{t,5}$ | $\alpha_{t,6}$ |
| $\alpha_{t,7}$ | $\alpha_{t,8}$ | $\alpha_{t,9}$ |

Distribution over grid locations

$$\sum_j \alpha_{t,j} = 1$$

**Soft attention:**
Average over locations of feature map weighted by attention: $z_t = \sum_i \alpha_{t,i}\, a_i$

Train with gradient descent

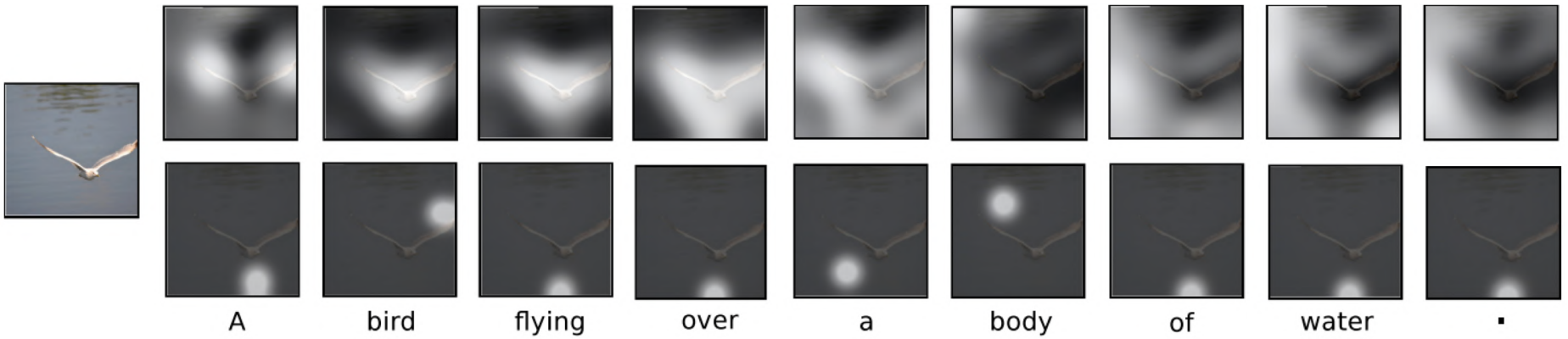
**Hard attention**:
Sample ONE location: $z_t$ is a random variable taking on values $a_i$ with probabilities $\alpha_{t,i}$

Can't use gradient descent; need reinforcement learning

Adapted from Stanford cs231n

# Attention in Image Captioning

**Soft attention:**
Average over locations of feature map weighted by attention: $z_t = \sum_i \alpha_{i,t}\, a_i$




**Hard attention**:
Sample ONE location: $z_t$ is a random variable taking on values $a_i$ with probabilities $\alpha_{i,t}$

Adapted from Stanford cs231n

# Attention in Image Captioning

- Good captions:

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.

Adapted from Stanford cs231n

# Attention in Image Captioning

- Mistakes

A large white bird standing in a forest.

A woman holding a clock in her hand.

A man wearing a hat and a hat on a skateboard.

A person is standing on a beach with a surfboard.

A woman is sitting at a table with a large pizza.

A man is talking on his cell phone while another man watches.

Adapted from Stanford cs231n

# RNN Summary

- RNNs allow for processing of variable length inputs and outputs by maintaining state information across time steps
- Various Input-Output scenarios are possible (Single/Multiple)
- Exploding gradients are handled by gradient clipping
- RNN attention enables long distance information propagation

- **Attention mechanism** helps in encoder-decoder RNN, where the relevant context from the encoder is adaptively calculated for each step in the decoder.

# Other Useful Resources / References

http://cs231n.stanford.edu/slides/winter1516_lecture10.pdf

http://www.cs.toronto.edu/~rgrosse/csc321/lec10.pdf


R. Pascanu, T. Mikolov, and Y. Bengio, On the difficulty of training recurrent neural networks, ICML 2013

S. Hochreiter, and J. Schmidhuber, Long short-term memory, Neural computation, 1997 9(8), pp.1735-1780

F.A. Gers, and J. Schmidhuber, Recurrent nets that time and count, IJCNN 2000

K. Greff , R.K. Srivastava, J. Koutník, B.R. Steunebrink, and J. Schmidhuber, LSTM: A search space odyssey, IEEE transactions on neural networks and learning systems, 2016

K. Cho, B. Van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, Learning phrase representations using RNN encoder-decoder for statistical machine translation, ACL 2014

R. Jozefowicz, W. Zaremba, and I. Sutskever, An empirical exploration of recurrent network architectures, JMLR 2015

# THE END

# Challenges with sequence Modelling

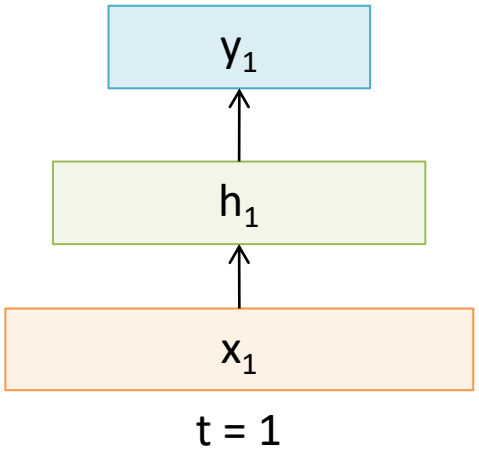
Challenges with RNN:

- Long range dependencies
- Vanishing / explosion gradients
- Large number of training steps
- Recurrence prevents parallel processing

Transformers:

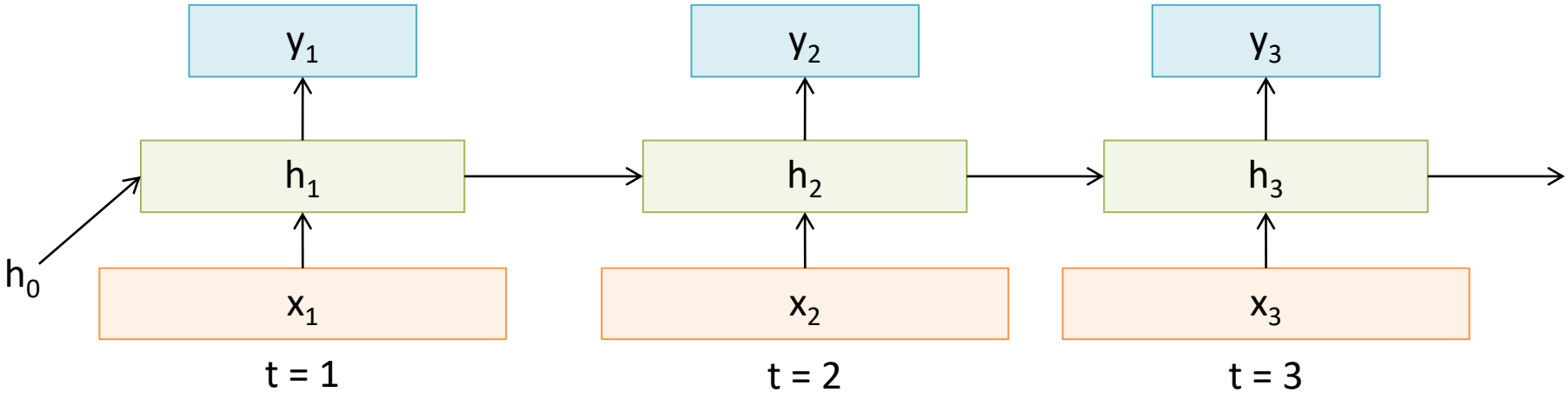
- Facilitates long range dependencies
- No Vanishing / explosion gradients
- Fewer training steps
- No recurrence that facilitates parallel processing

# Vanilla RNN




Adapted from: Arun Malya.

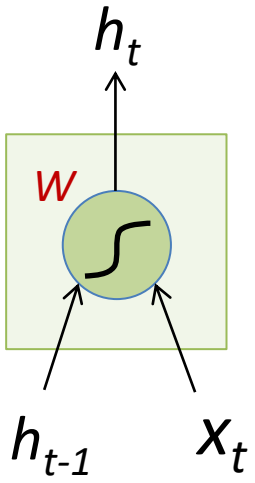
# Vanilla RNN




Adapted from: Arun Malya.

# Vanilla RNN

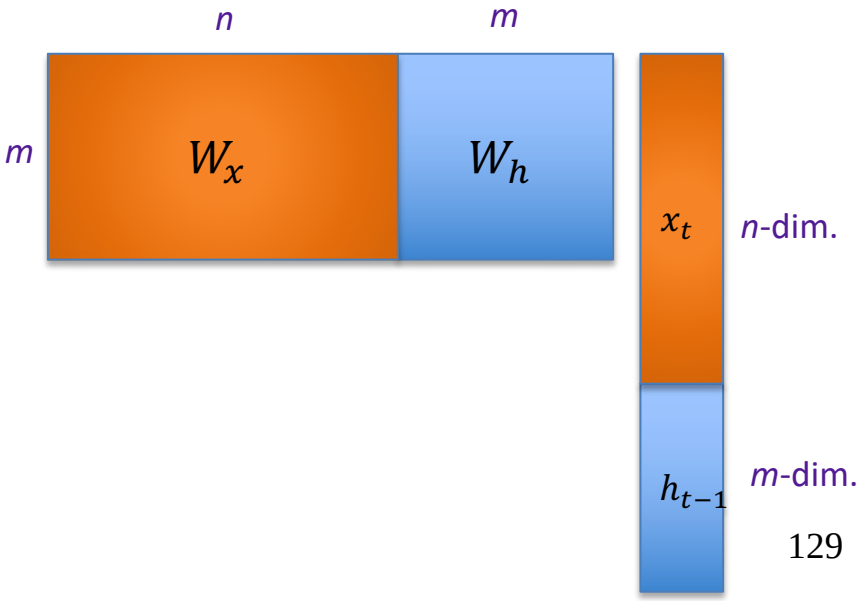
- A single RNN cell




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

$$= \tanh(W_x x_t + W_h h_{t-1})$$

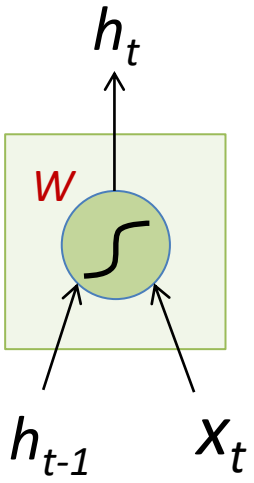
$$= \tanh\left(W \begin{bmatrix} x_t \\ h_{t-1} \end{bmatrix}\right)$$




Adapted from: Arun Malya.

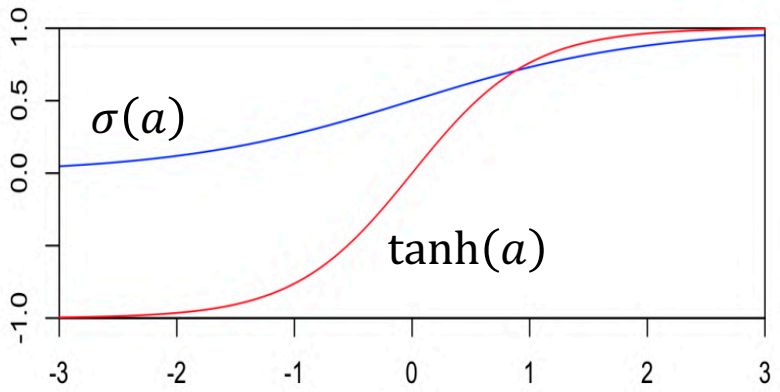
# Vanilla RNN

- A single RNN cell




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

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

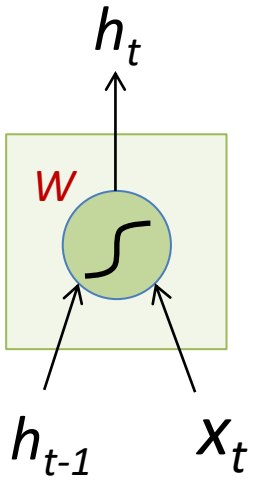



$$\tanh(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}}$$

$$= 2\sigma(2a) - 1$$

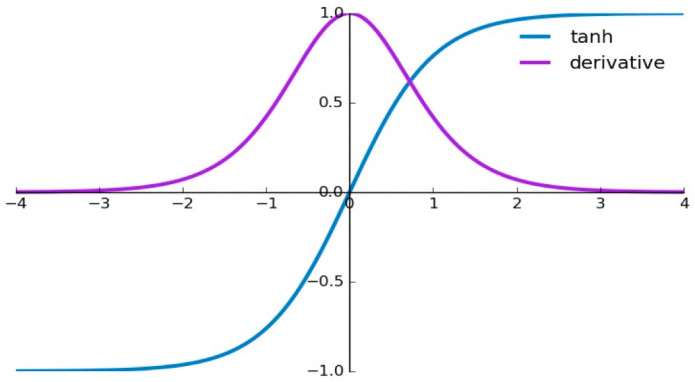
# Vanilla RNN

- A single RNN cell



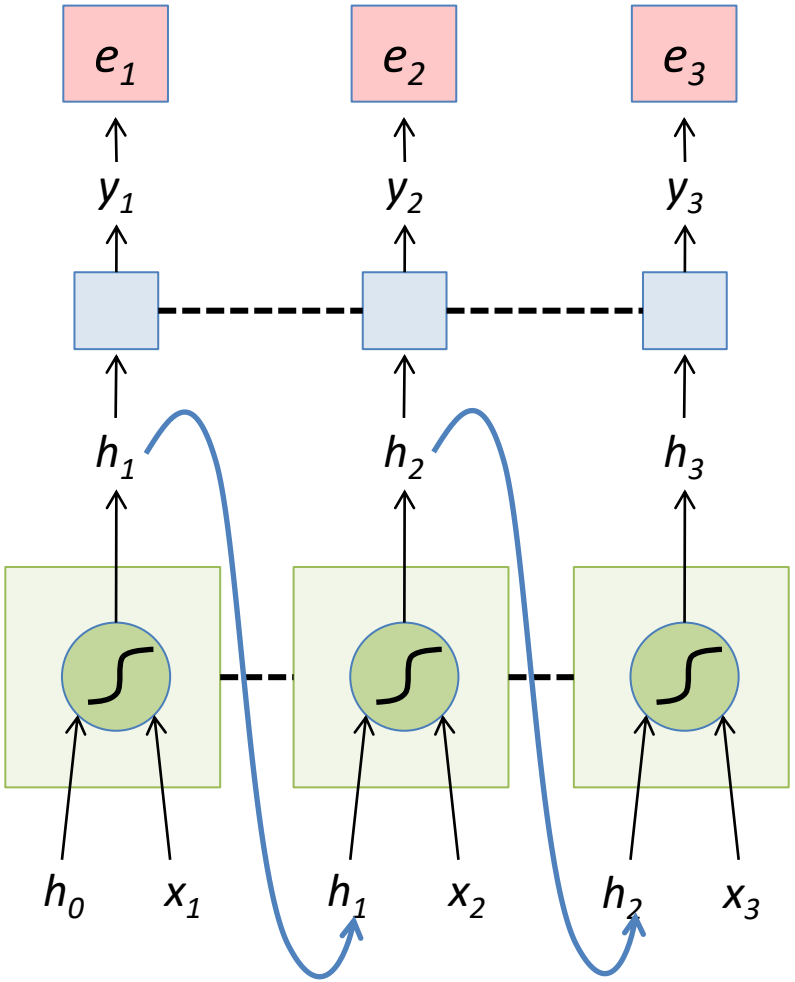

$$h_t = f_W(x_t, h_{t-1})$$

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




$$\frac{d}{da}\tanh(a) = 1 - \tanh^2(a)$$

# RNN Forward Pass




$$e_t = -\log(y_t(GT_t))$$

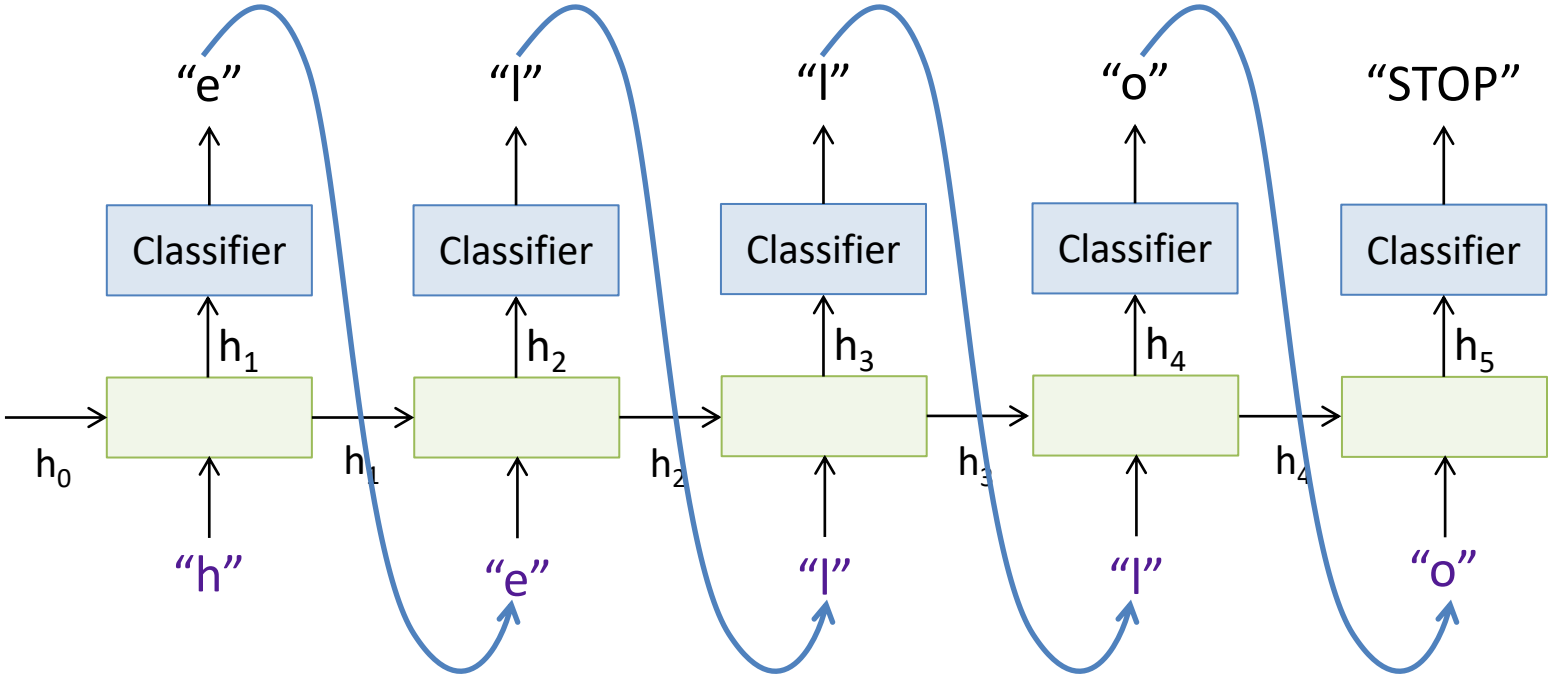
$$y_t = \text{softmax}(W_y h_t)$$

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

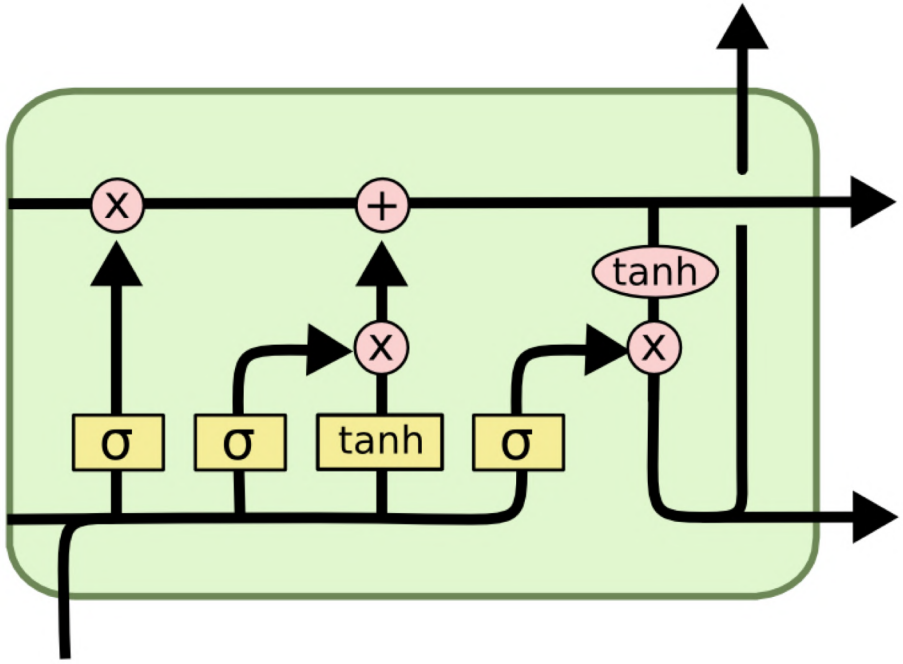
------ shared weights

# RNN Example

Example: **Language Model – Char RNN**
Vocabulary: [h,e,l,o]



# Long Short-Term Memory (LSTM)



# Long Short-Term Memory (LSTM)

- Suppose that instead of a matrix multiplication, we had an identity relationship between the hidden states

$$h_t = h_{t-1} + F(x_t)$$

Remember Resnets?

$$\Rightarrow \frac{\partial h_t}{\partial h_{t-1}} = 1$$

- The gradient does not decay as the error is propagated all the way back.



Adapted from: Arun Malya.

# RNN with Attention

$$\alpha_{i,j} = \frac{\exp e_{i,j}}{\sum_k \exp e_{i,k}} \quad \text{where} \quad e_{i,j} = FC(s_{i-1}, h_j)$$

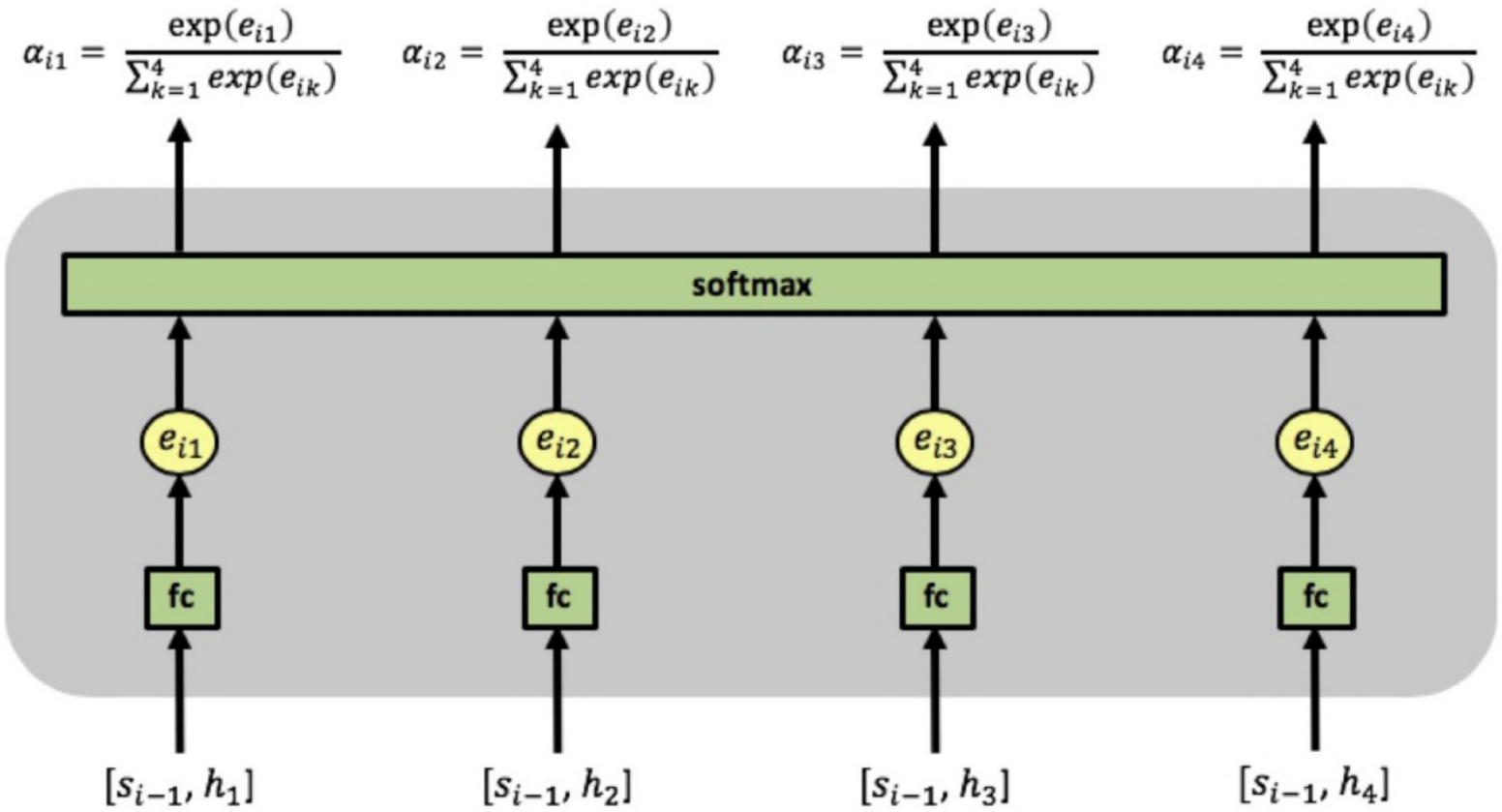
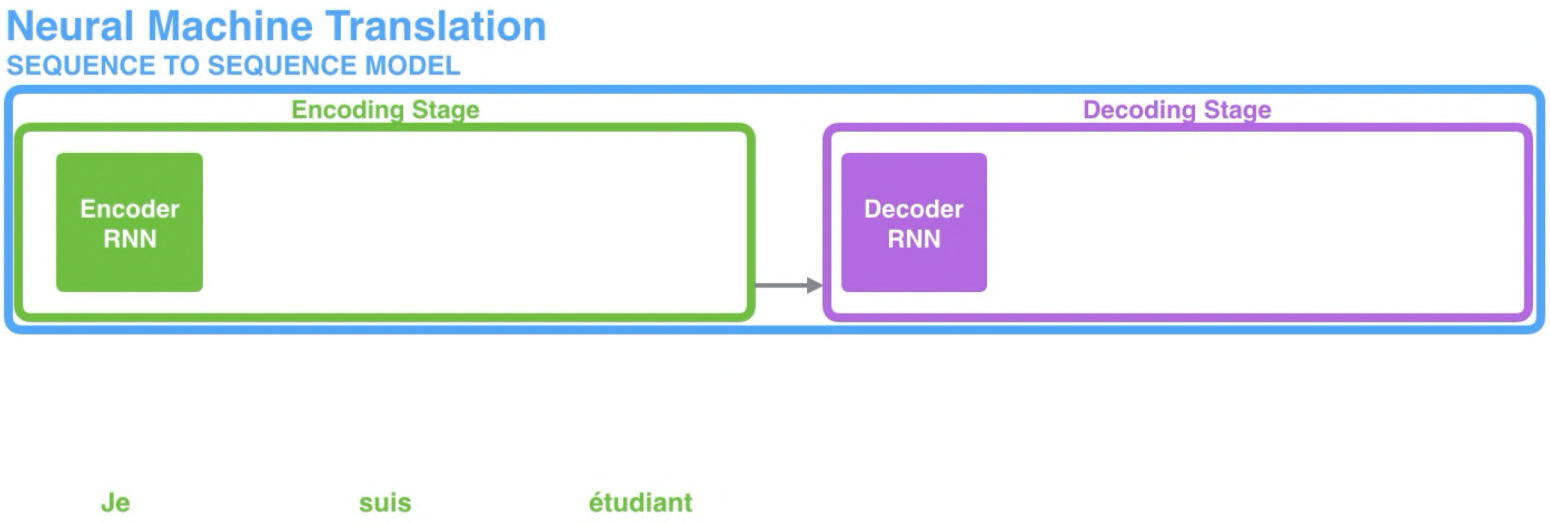


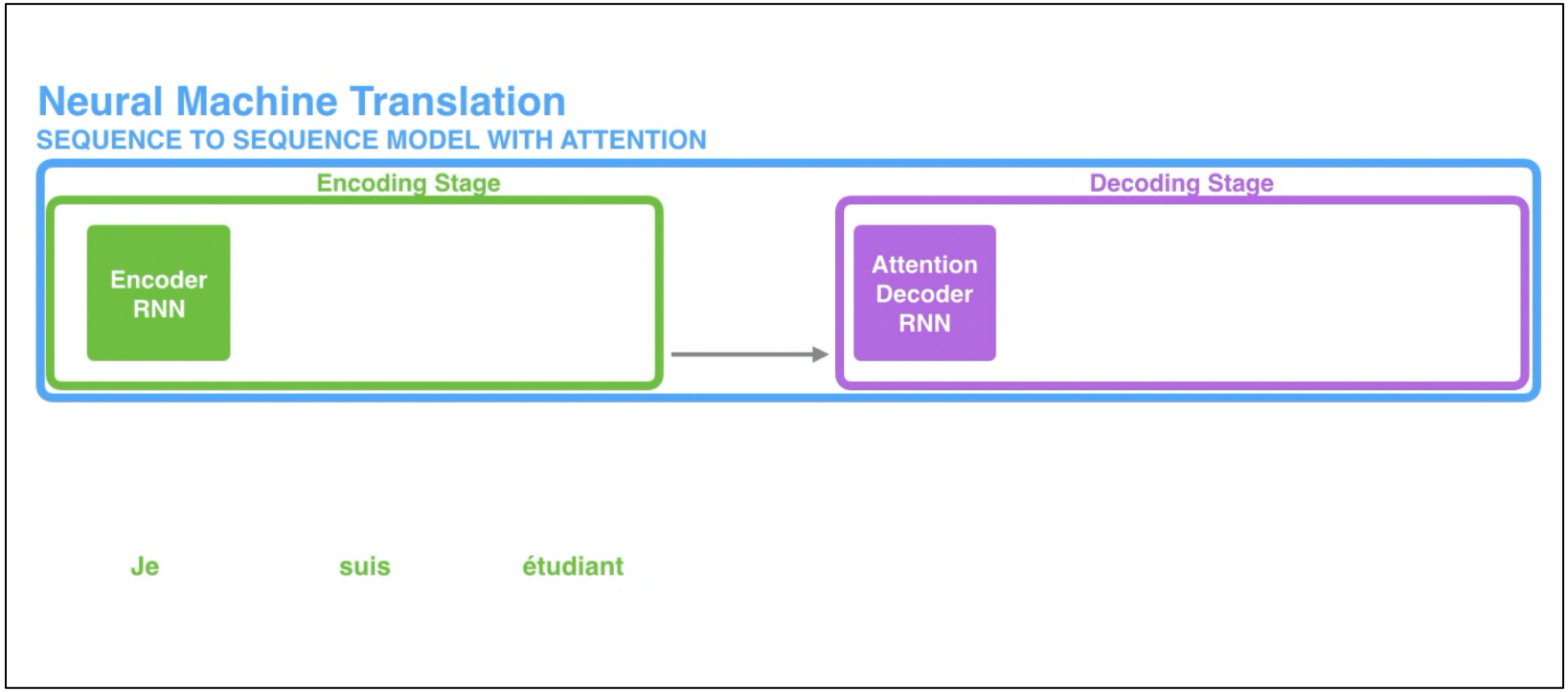

Image from: Eli Arbel

# RNN without Attention




From: [Jay Alammar]

# RNN with Attention




From: [Jay Alammar]

# RNN with Attention


From: Jay Alammar