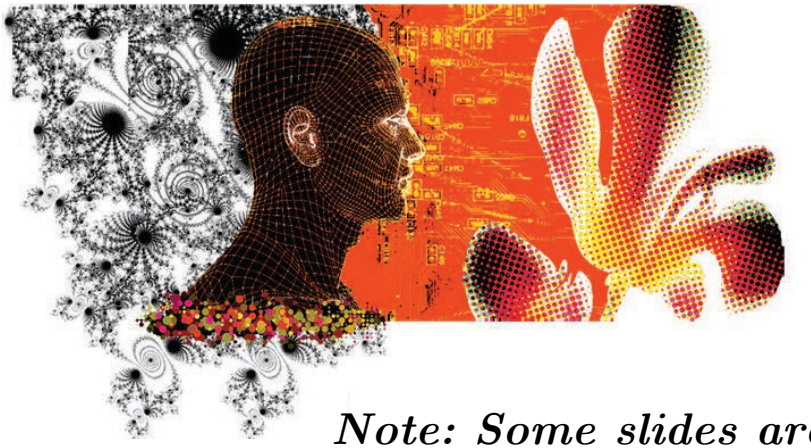
Data Science Training

November 2017

Recurrent Neural Networks

Xavier Bresson

Data Science and AI Research Centre
NTU, Singapore



http://data-science-optum17.tk

Note: Some slides are from Li, Karpathy, Johnson's course on Deep Learning and Le's summer school.

Outline

- Neural Networks for Data Sequence
- Recurrent Neural Networks
- Long Short-Term Memory (LSTM)
- Deep RNNs
- Recent Architectures
- Applications
- Attention Networks
- Conclusion

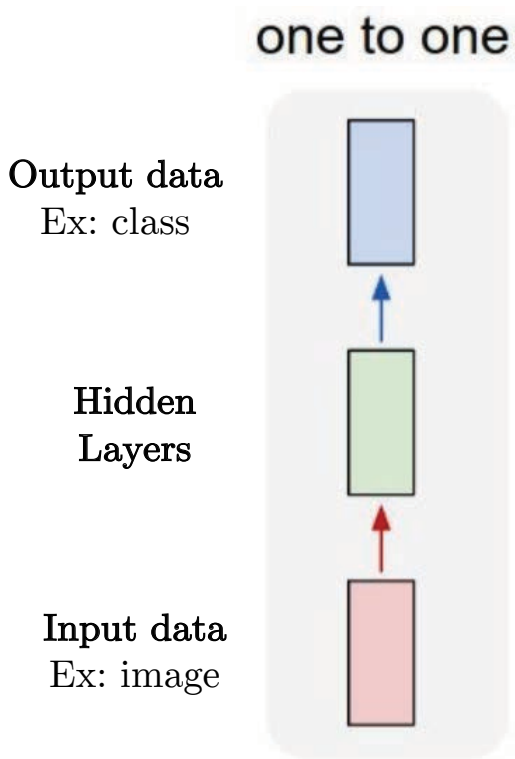
Outline

- **Neural Networks for Data Sequence**
- Recurrent Neural Networks
- Long Short-Term Memory (LSTM)
- Deep RNNs
- Recent Architectures
- Applications
- Attention Networks
- Conclusion

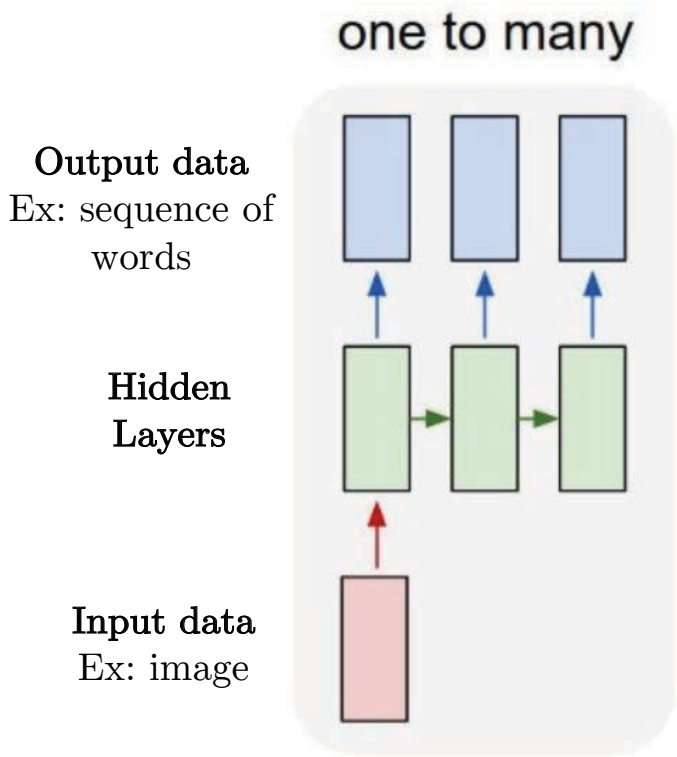
RNN for Data Sequence

- **Recurrent Neural Networks** (RNNs) learn the (non-linear) dynamics of **ordered sequences** of input data. *Exs:* Text, financial series, videos, robot motion, etc.

- RNNs are highly **flexible** to learn:

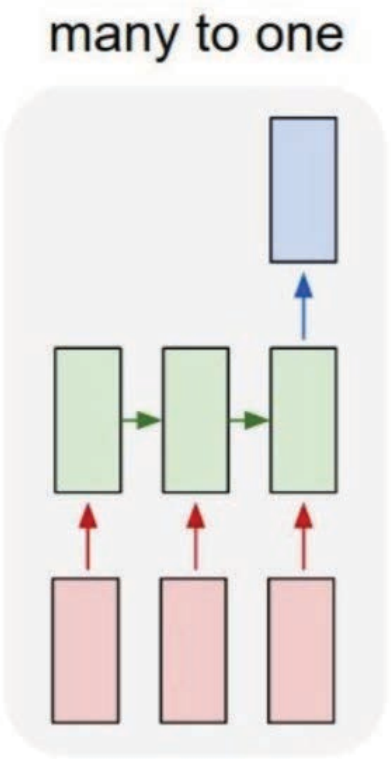


Vanilla NNs
One input vector maps to one output vector.
Ex: **CNNs** (image to class)



RNNs
One input vector maps to multiple output vectors.
Ex: **Image captioning**

RNN for Data Sequence

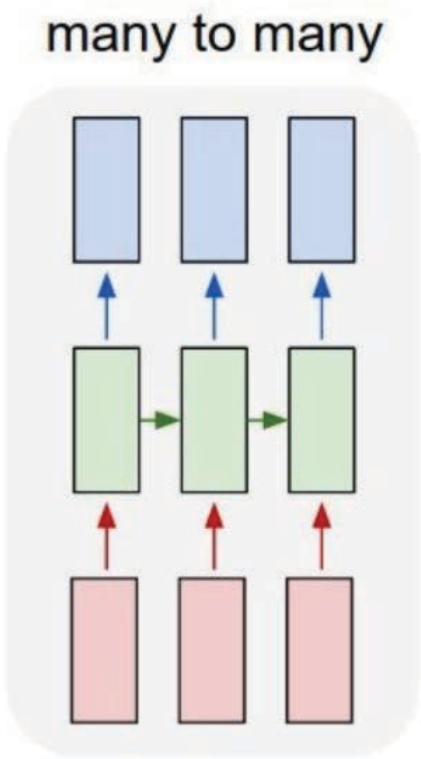


Output data
Ex: class

Hidden Layers

Input data
Ex: sequence of images

RNNs
Multiple input vectors map to an output vector.
Ex: **Video classification**
Sentiment analysis



Output data
Ex: sequence of words

Hidden Layers

Input data
Ex: sequence of words

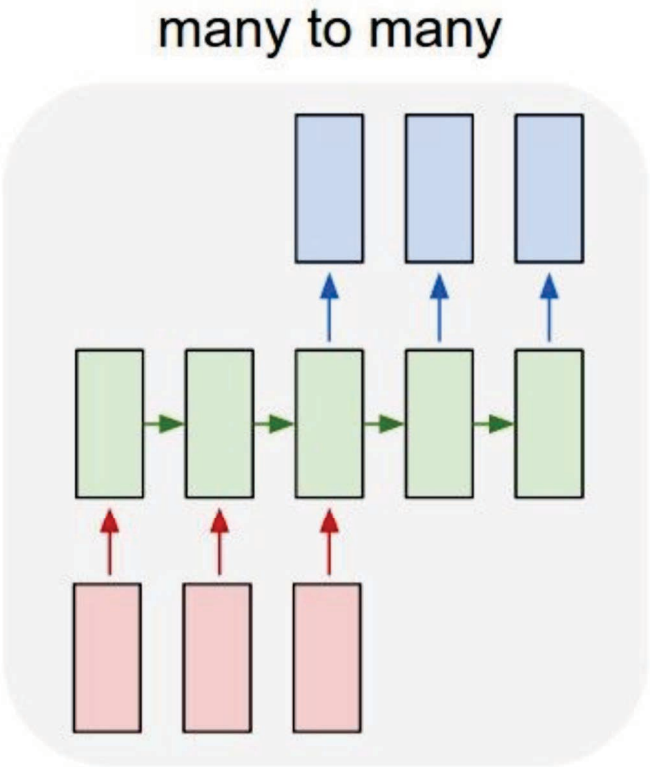
RNNs
Multiple input vectors map to multiple output vectors.
Ex: **Word prediction**

RNN for Data Sequence

Output data
Ex: sequence of words

Hidden Layers

Input data
Ex: sequence of words



RNNs
Multiple input vectors map to multiple output vectors.
Exs: **Machine Translation**
Q&As
Speech-to-Text

Outline

- Neural Networks for Data Sequence
- **Recurrent Neural Networks**
- Long Short-Term Memory (LSTM)
- Deep RNNs
- Recent Architectures
- Applications
- Attention Networks
- Conclusion

Generic RNN

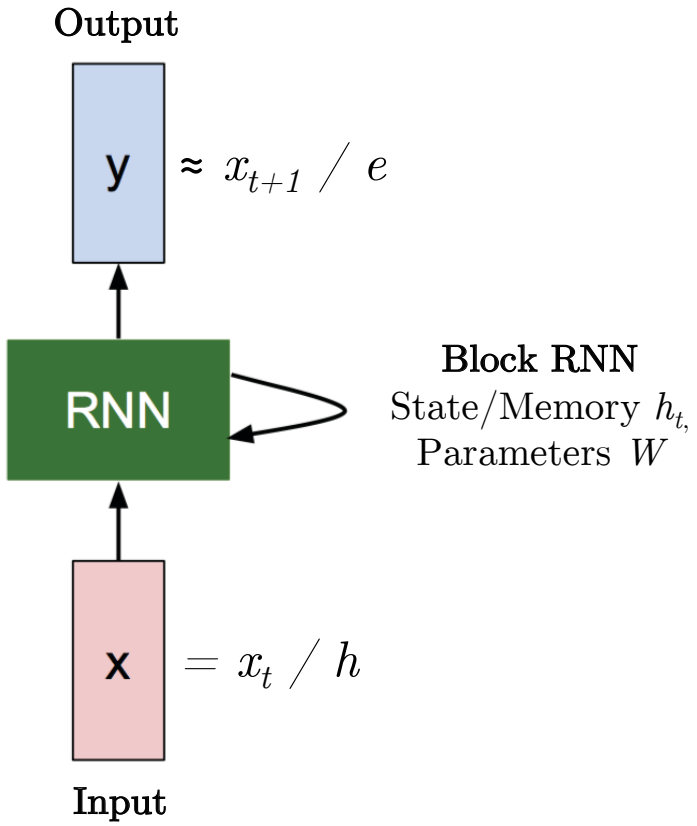
- **RNNs** are **recurrent** learning machine:

  1. RNNs receive at each time t an input vector $x=x_t$, and **learn the output** y **to predict the next input vector** x_{t+1}:

     Example: hello

     x y

  2. RNNs are parametrized by a **state/memory** $\boldsymbol{h_t}$ that summarizes the past. This memory can be modified by changing the **parameters** $\boldsymbol{W}$.



Recurrence Formula

- Memory of RNNs is updated/learned to make successful future prediction with a **recurrence formula**:

Recurrence function

Previous memory

Updated memory

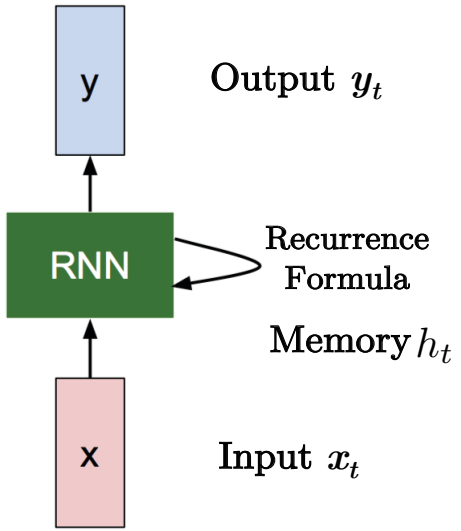
$$h_t = f_W(h_{t-1}, x_t)$$

Weights/ Parameters of memory

Input vector at current time

Output vector at current time

$$y_t = g_W(h_t)$$



- **Notes:**
  1. Recurrence and output functions are **independent of time *t***, i.e. same functions f, g is used at every time step.
  2. Changing W updates the **dynamic behavior prediction** of RNNs.
  3. Weights W are learned by **backpropagation**.

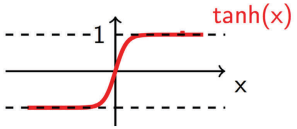
Vanilla RNNs

- **Simplest** RNNs:

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

Linear classifier/ template matching

Non-linear activation



$$h_t = \tanh(W_h h_{t-1} + W_x x_t + b_h)$$
$$y_t = \text{Softmax}(W_y h_t + b_y)$$

Probability

Linear classifier

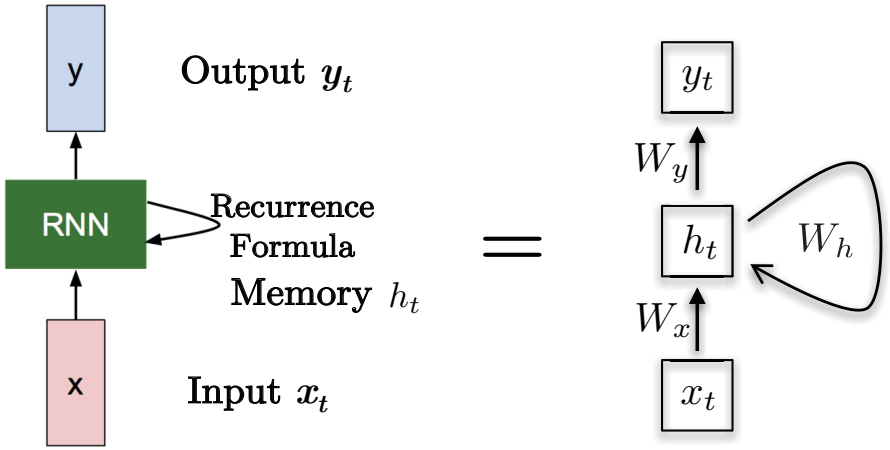
Input sentence: last year...

1. **Memory neurons** W_h get activated when "last" appears, then high probability to get "year", "month", "week", etc

2. Input vector is **embedded** with W_x into a meaningful compressed representation.

$$\text{Softmax}\left(\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}\right) = \begin{bmatrix} \frac{e^{x_1}}{e^{x_1}+e^{x_2}+e^{x_3}} \\ \frac{e^{x_2}}{e^{x_1}+e^{x_2}+e^{x_3}} \\ \frac{e^{x_3}}{e^{x_1}+e^{x_2}+e^{x_3}} \end{bmatrix}$$

Ex. $\text{Softmax}\left(\begin{bmatrix} -1.5 \\ 3 \\ 1 \end{bmatrix}\right) = \begin{bmatrix} .01 \\ .87 \\ .12 \end{bmatrix}$



Unfolding RNNs

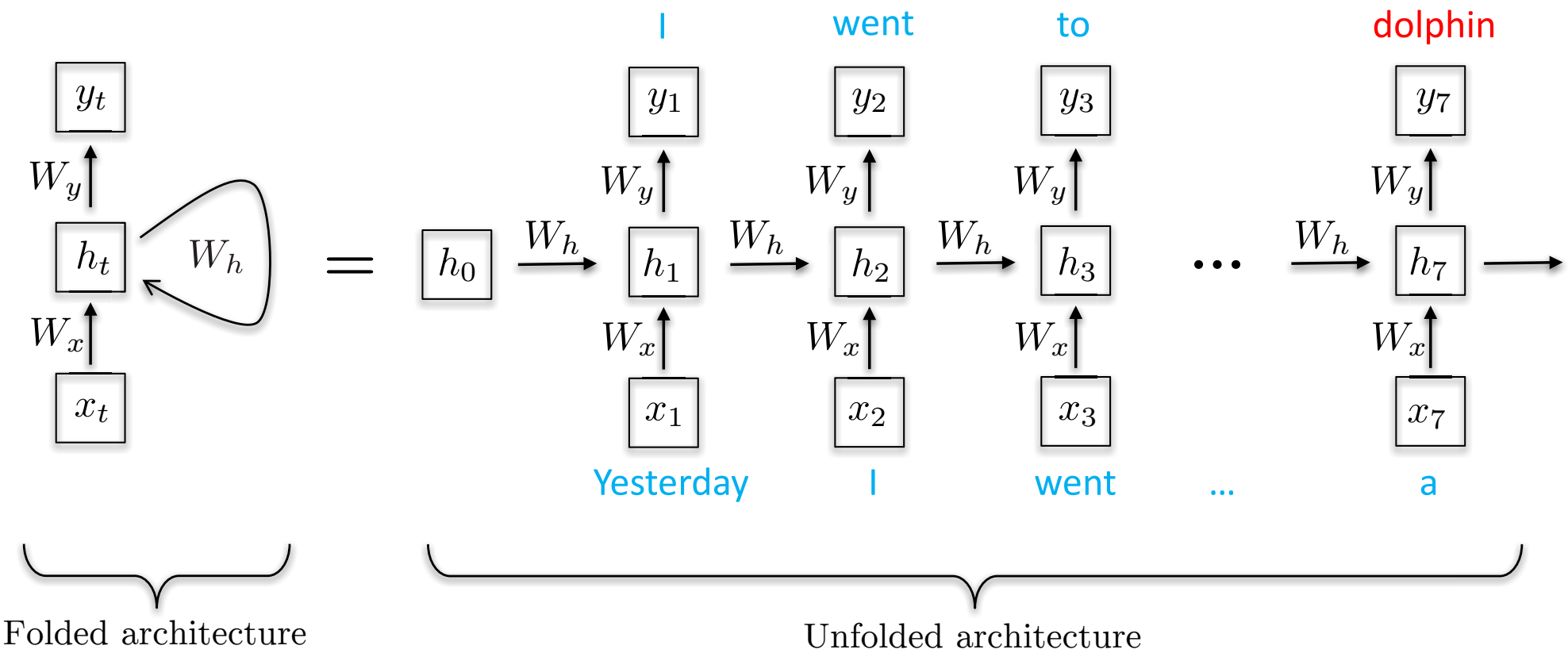
- **NLP example:**

  Input: A sequence of words

  Yesterday I went to the beach and I saw a …

  Output: Probability of the next word

next word	probability
dolphin	11.5%
boat	8.2%
⋮	
elephant	0.1%
tiger	0.1%
⋮	
cooking	0.0%
with	0.0%



Unfolding RNNs

- **Character-Level NLP**

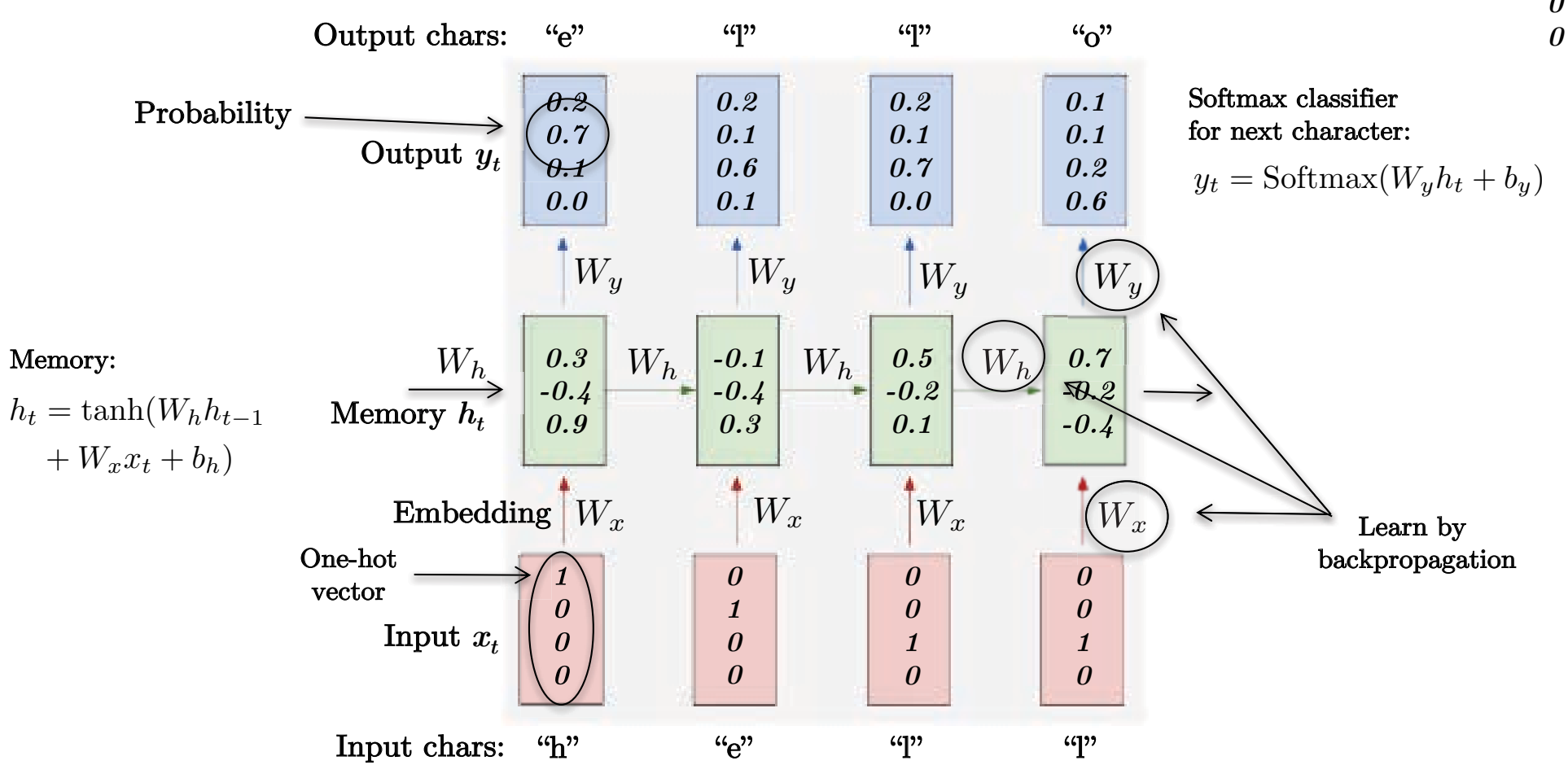
  Input: A sequence of characters

  hello

  Output: Probability of the next character

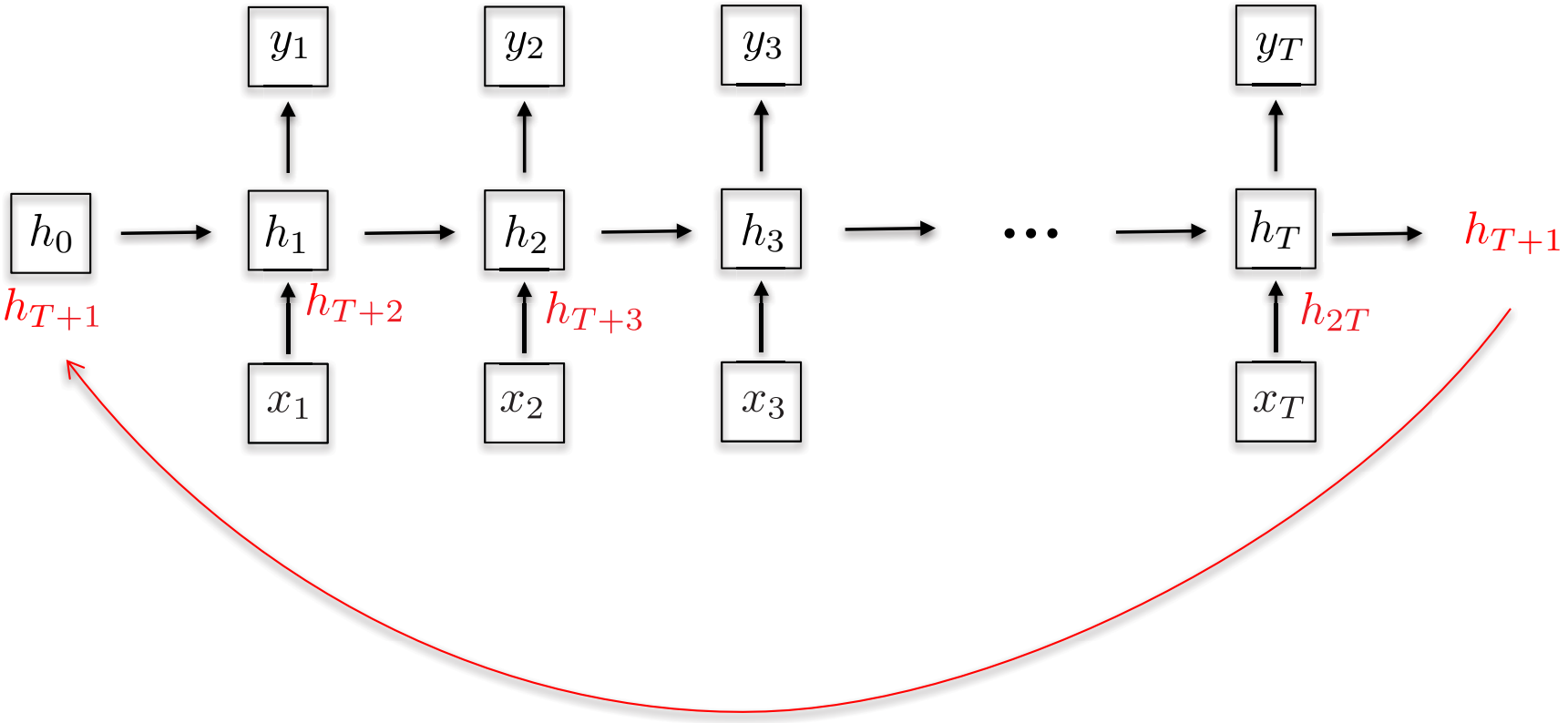
Vocabulary set: {h,e,l,o}
In RNN, **inputs** are usually encoded by **one-hot vector**.

"e" = 1 0 0 0



Long-Term Dependencies

- VRNNs can process **any length** of data sequence by **batch of size T**:



- **Limitation:** VRNN are not able to learn **long-term dependencies**.

Long-Term Dependencies

- **Short-term dependency:**

  Input: The clouds are in the ?

  Output: sky

- **Long-term dependency:**

  Input: I grew up in France… blabla… blabla… I speak fluent ?

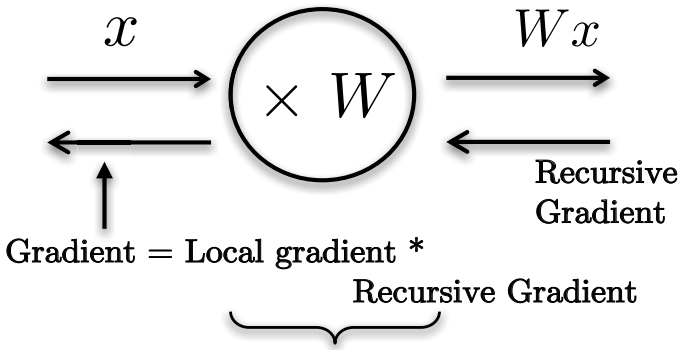
  Output: French

- Theoretical issue: **Vanishing gradient problem in backpropagation.**

  1. At each layer, output gradient is **smaller**.

  2. We **chain** all recursive gradients by backpropagation:

  W x W x W x … x W ⇒ **Exponential decay** (e.g. W=0.1^{10})



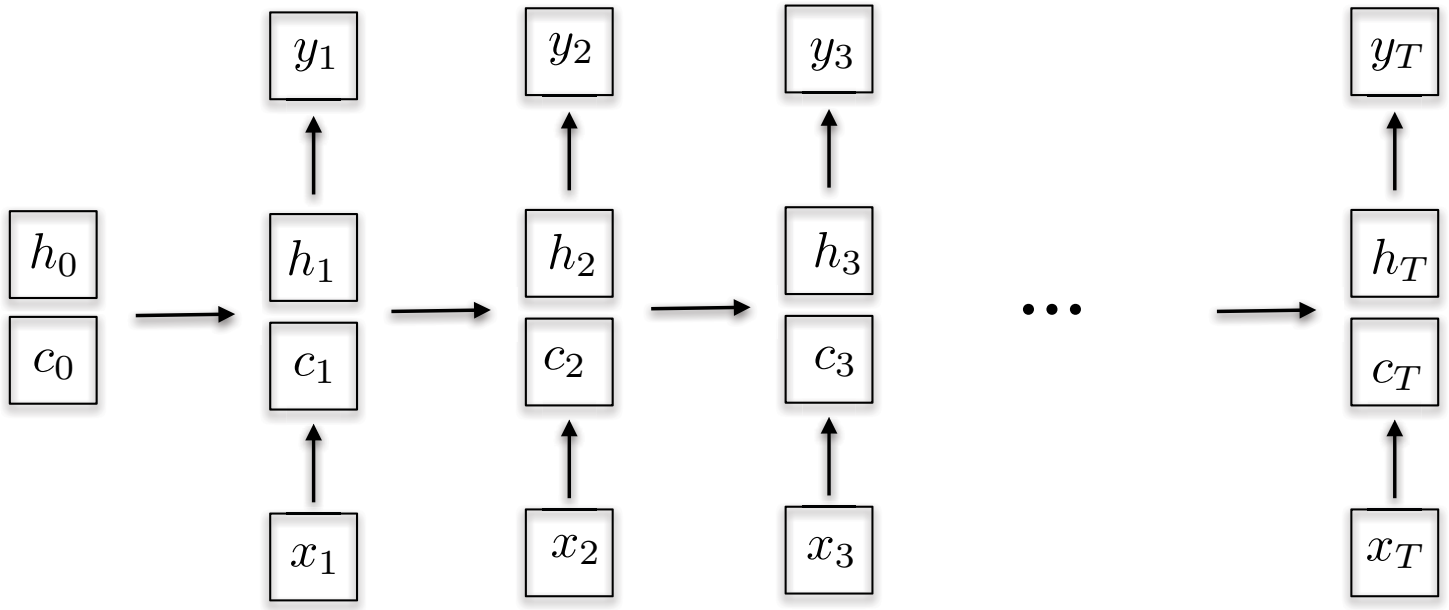
Gradient descent: $\frac{\partial W}{\partial t} = -\tau \frac{L(W)}{\partial x}$ ⟶ **Gradient:** $\frac{\partial Wx}{\partial x} = W$

Outline

- Neural Networks for Data Sequence
- Recurrent Neural Networks
- **Long Short-Term Memory (LSTM)**
- Deep RNNs
- Recent Architectures
- Applications
- Attention Networks
- Conclusion

Long Short-Term Memory (LSTM)
[Hochreiter-Schmidhuber'97]

- **LSTM architecture**

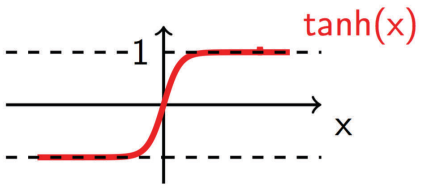


x_t : Input at t
h_t : *Short-term* memory vector at t
c_t : *Long-term* memory vector at t
y_t : Probability of next word at t

Non-Linear Dynamics

- **Iterative scheme:**

$$c_t = f_t \odot c_{t-1} + i_t \odot \tanh(z_t)$$
$$h_t = o_t \odot \tanh(c_t)$$
$$y_t = \text{softmax}(W_y h_t)$$



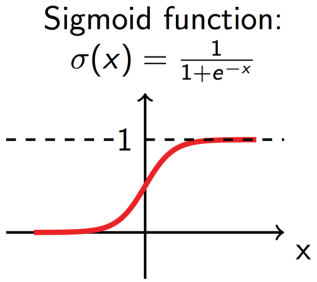
with

$$i_t = \sigma(W_h^i h_t + W_x^i x_t)$$ **Input gate**

$$f_t = \sigma(W_h^f h_t + W_x^f x_t)$$ **Forget gate (*f*=*0,1*)**

$$o_t = \sigma(W_h^o h_t + W_x^o x_t)$$ **Output gate**

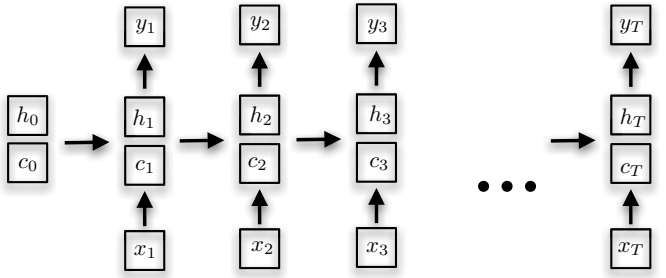
$$z_t = \sigma(W_h^z h_t + W_x^z x_t)$$



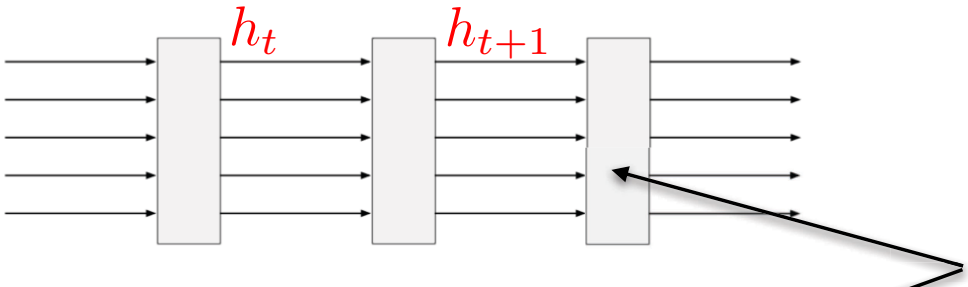
Long-Term Dependencies

- **Long-Term Dependencies:**
  1. **Always initialize with Forget gate *f=1***
     $\Rightarrow$ Long-term memory c is active.
  2. Observe **c_t is directly copied from c_{t-1}**
     $\Rightarrow$ Gradient does not loose any amplitude during backpropagation.

$$c_t = f_t \odot c_{t-1} + i_t \odot \tanh(z_t)$$
$$h_t = o_t \odot \tanh(c_t)$$
$$y_t = \text{softmax}(W_y h_t)$$

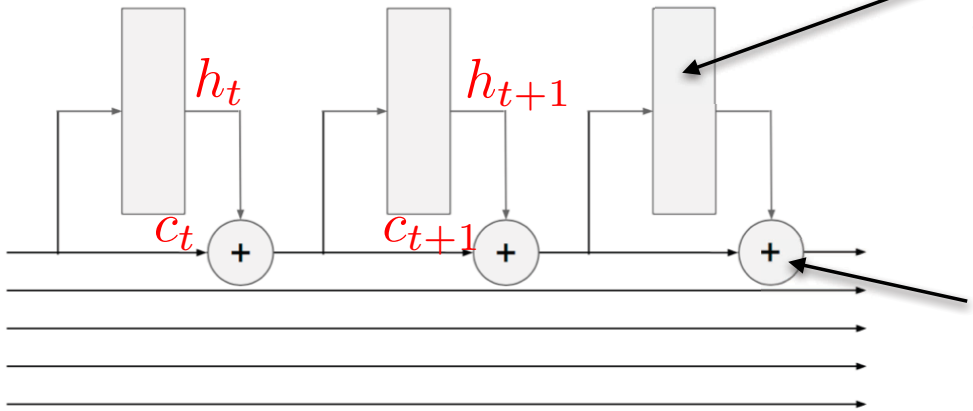


Vanilla RNN



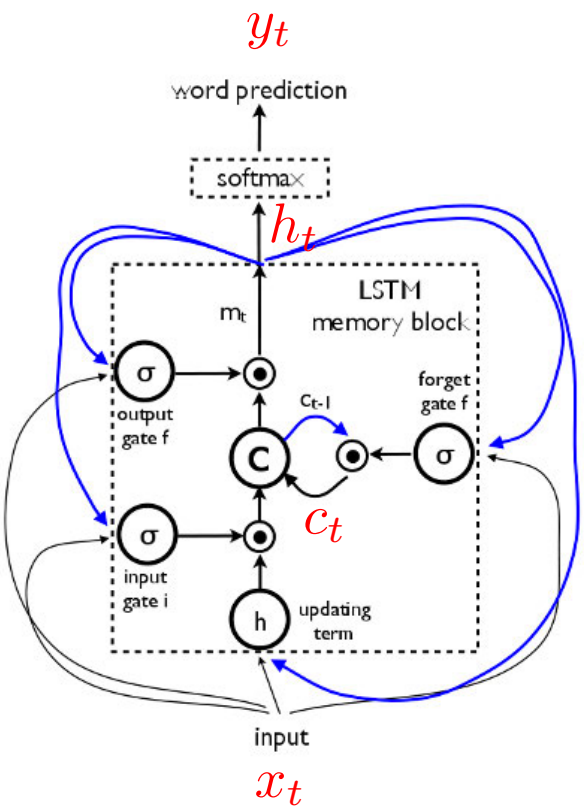
Linear operator + non-linear activation $\Rightarrow$ Decrease ∇h_t amplitude at each gate ≈ 0.5.

LSTM

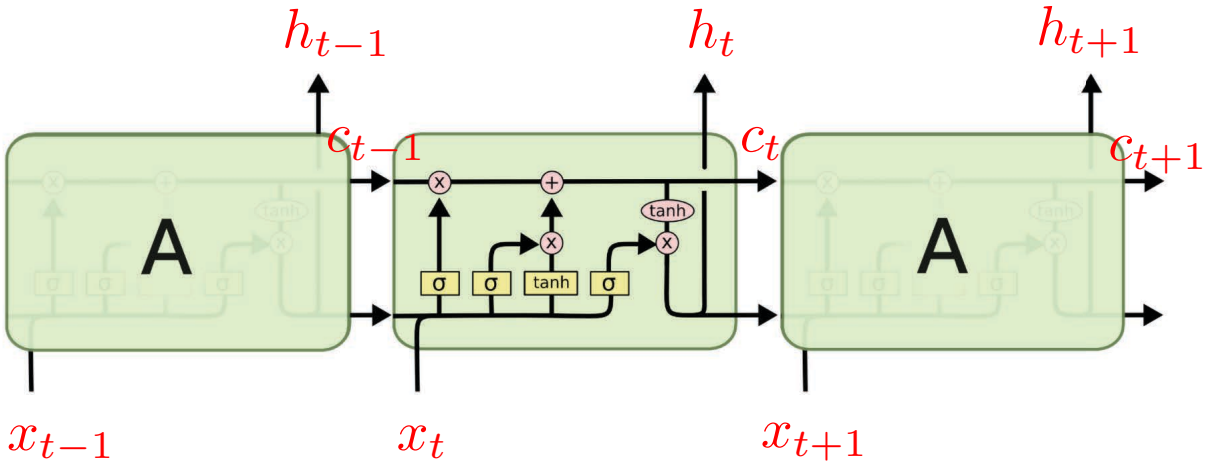


The (+) gates distributes the gradient equally during backpropagation.

LSTM



One LSTM cell



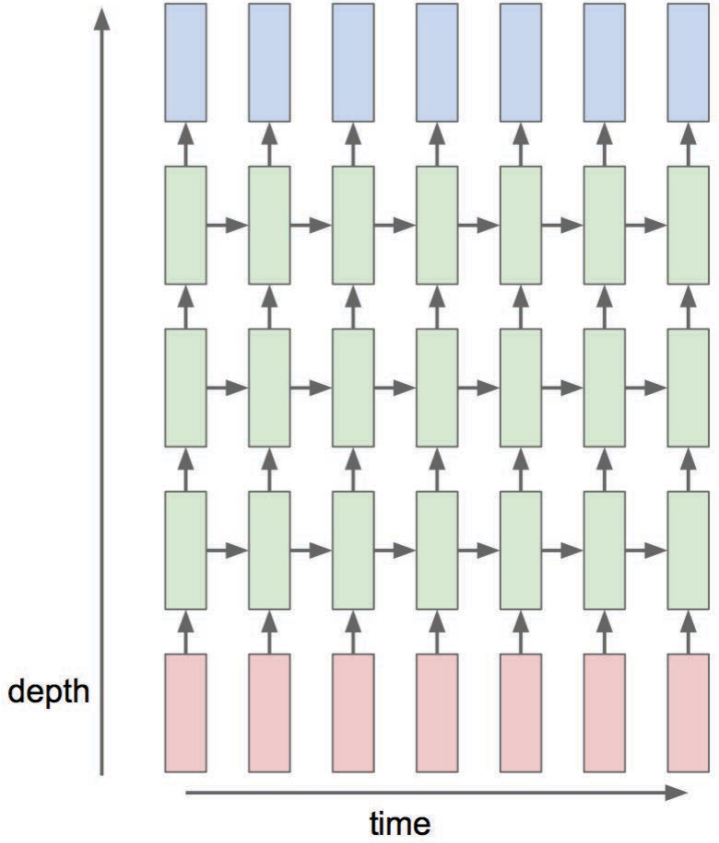
Cascade of LSTM cells to do various learning tasks

Outline

- Neural Networks for Data Sequence
- Recurrent Neural Networks
- Long Short-Term Memory (LSTM)
- **Deep RNNs**
- Recent Architectures
- Applications
- Attention Networks
- Conclusion

Deep RNNs

- There are 2 ways to make RNNs **deep**:
  1. Along **depth** dimension
  2. Along **time** dimension



- **Depth/Multilayer RNNs:**

depth level

$$h_t^0 = \tanh(W_h^0 h_{t-1}^0 + W_x^0 x_t + b_h^0)$$
$$h_t^1 = \tanh(W_h^1 h_{t-1}^1 + W_x^1 h_t^0 + b_h^1)$$

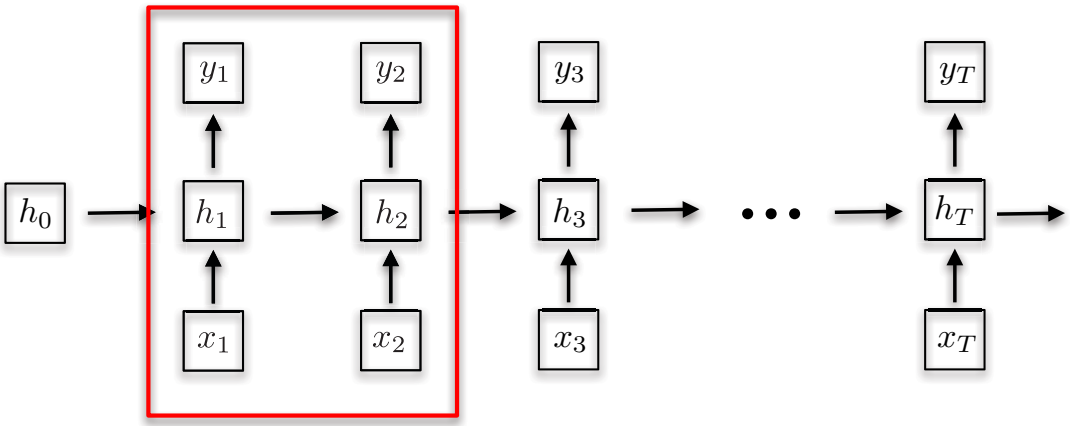
ResNet trick

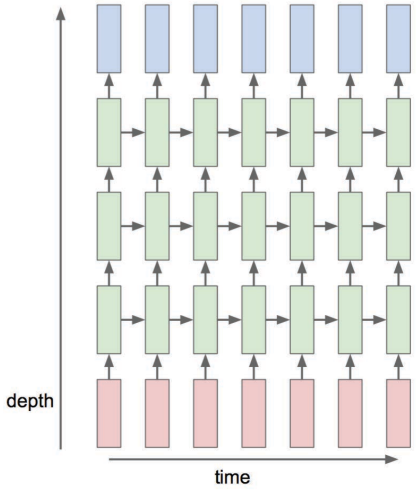
$$h_t^2 = h_t^1 + \tanh(W_h^2 h_{t-1}^2 + W_x^2 h_t^0 + b_h^2)$$
$$h_t^3 = h_t^2 + \tanh(W_h^3 h_{t-1}^3 + W_x^3 h_t^0 + b_h^3)$$
$$\dots$$
$$y_t = \text{Softmax}(W_y h_t^l + b_y)$$

Application: **Google Translation machine** since Nov. 2016

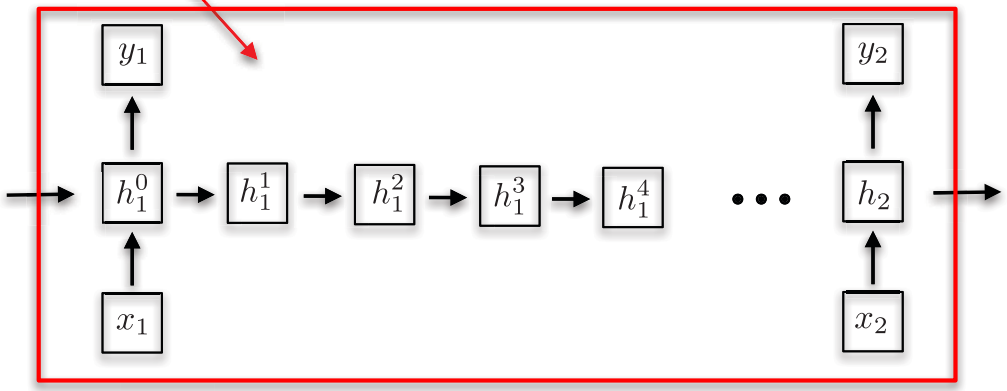
Deep RNNs

- **Regular RNN:**





- **Deep Temporal RNN:**



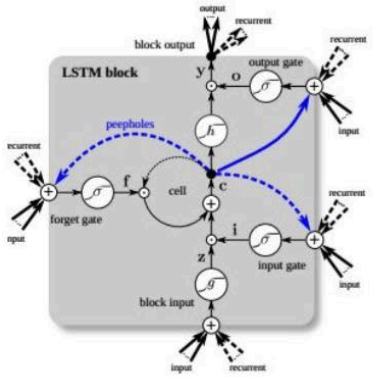
$$h_t^0 = \tanh(W_h^2 \tanh(W_h^1 \underbrace{\tanh(W_h^0 h_{t-1}^0 + W_x^0 x_t)}_{h_1^1}))$$

$$\underbrace{\qquad\qquad\qquad\qquad}_{h_1^2}$$

Outline

- Neural Networks for Data Sequence
- Recurrent Neural Networks
- Long Short-Term Memory (LSTM)
- Deep RNNs
- **Recent Architectures**
- Applications
- Attention Networks
- Conclusion

LSTM Variants



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

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

$$
\begin{aligned}
r_t &= \text{sigm}\left(W_{\text{xr}}x_t + W_{\text{hr}}h_{t-1} + b_{\text{r}}\right) \\
z_t &= \text{sigm}(W_{\text{xz}}x_t + W_{\text{hz}}h_{t-1} + b_{\text{z}}) \\
\tilde{h}_t &= \tanh(W_{\text{xh}}x_t + W_{\text{hh}}(r_t \odot h_{t-1}) + b_{\text{h}}) \\
h_t &= z_t \odot h_{t-1} + (1 - z_t) \odot \tilde{h}_t
\end{aligned}
$$

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

MUT1:

$$
\begin{aligned}
z &= \text{sigm}(W_{\text{xz}}x_t + b_{\text{z}}) \\
r &= \text{sigm}(W_{\text{xr}}x_t + W_{\text{hr}}h_t + b_{\text{r}}) \\
h_{t+1} &= \tanh(W_{\text{hh}}(r \odot h_t) + \tanh(x_t) + b_{\text{h}}) \odot z \\
&+ h_t \odot (1 - z)
\end{aligned}
$$

MUT2:

$$
\begin{aligned}
z &= \text{sigm}(W_{\text{xz}}x_t + W_{\text{hz}}h_t + b_{\text{z}}) \\
r &= \text{sigm}(x_t + W_{\text{hr}}h_t + b_{\text{r}}) \\
h_{t+1} &= \tanh(W_{\text{hh}}(r \odot h_t) + W_{xh}x_t + b_{\text{h}}) \odot z \\
&+ h_t \odot (1 - z)
\end{aligned}
$$

MUT3:

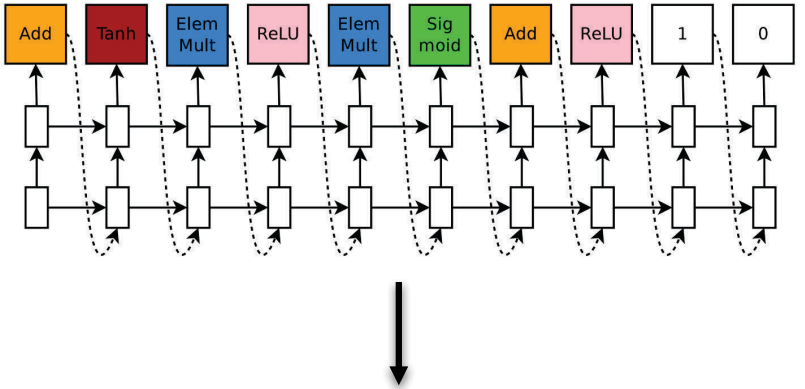
$$
\begin{aligned}
z &= \text{sigm}(W_{\text{xz}}x_t + W_{\text{hz}}\tanh(h_t) + b_{\text{z}}) \\
r &= \text{sigm}(W_{\text{xr}}x_t + W_{\text{hr}}h_t + b_{\text{r}}) \\
h_{t+1} &= \tanh(W_{\text{hh}}(r \odot h_t) + W_{xh}x_t + b_{\text{h}}) \odot z \\
&+ h_t \odot (1 - z)
\end{aligned}
$$

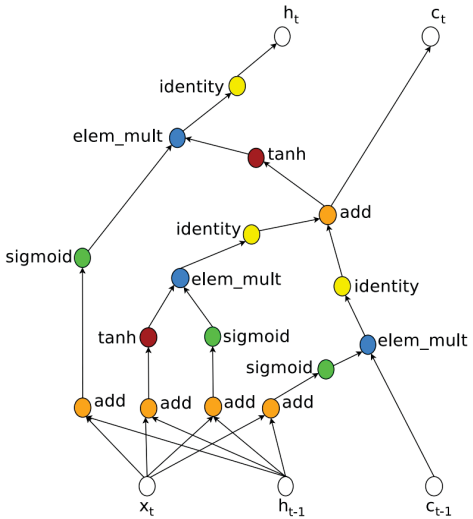
- At the end of the day, LSTM gives the best performances over many possible experimental conditions.
- LSTM works very well: It is used by **all big IT companies**: Facebook, Google, Microsoft, Apple, IBM, etc.

Learn-to-Learn LSTM-based Architecture

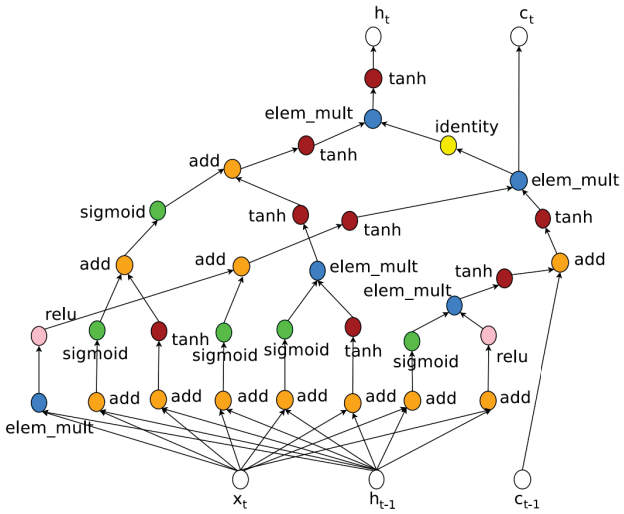
[Zoph-V.Le'17]

- Learn the **weights** of the architecture (by backpropagation), and also the **architecture** (by reinforcement learning):





Standard LSTM



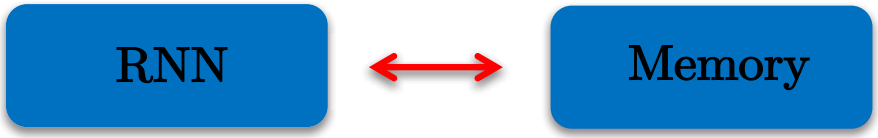
Learned architecture

- Issue: **800 GPUs** are needed for reinforcement learning.

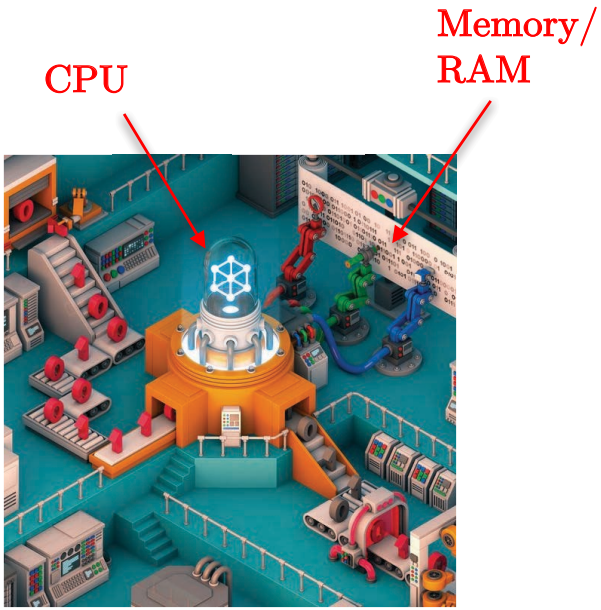
Augmenting RNN with Memory Unit

- **The ugly truth:** LSTM does **not** have long-term dependencies (at most 50 time steps).

- **It's okay.** The **cortex** only remembers things for **20 seconds.** The **hippocampus** (separate module) stores long-term memory information.



- Models:
  1. Memory networks [Facebook'14]
  2. Neural Turing machines [DeepMind'16]
  3. Dynamic memory networks [Salesforce'16]



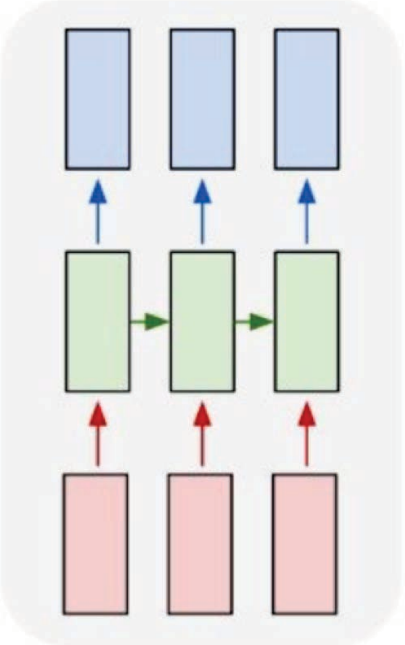
Nature

Outline

- Neural Networks for Data Sequence
- Recurrent Neural Networks
- Long Short-Term Memory (LSTM)
- Deep RNNs
- Recent Architectures
- **Applications**
- Attention Networks
- Conclusion

Applications

- **Many-to-many:**
  1. *Char-to-char prediction*
  2. *Word-to-word prediction (NLP)*



Applications

- Shakespeare-like:

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.
```

Demo

- Run **code01.ipynb**

Code 1 : Vanilla Recurrent Neural Networks

```
# Import functions in lib folder
import sys
sys.path.insert(0, 'lib/')
```

```
# Run RNNs on shakespeare
%run lib/min-char-rnn
```

```
data has 1115394 characters, 65 unique.
learning_rate= 0.1
----
 XTiyHbrO,aIf;bCMWsIazANgDR&Cc&oTdjoM
xWp$V!mPa$Hx?Zhhvp&;FakOX,N!Dh&aC
rb$!lLNbRnPtG3rnXtQkLSzpJXJKZYcWC-WTFkraDAWgGenzTKYKiR$jSlLgMQ
Ddokl:'RFRCDwALK
----
iter 0, loss: 104.359688
----
```

100 Python lines

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

Applications

- **Math-like** using Open source textbooks on algebraic geometric

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

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

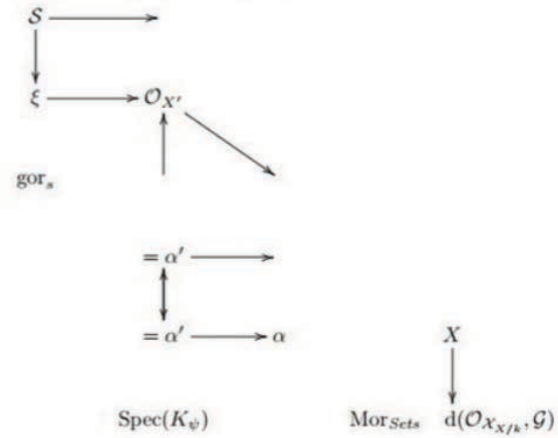
be a morphism of algebraic spaces over S and Y.

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 $f_\bullet$. 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.

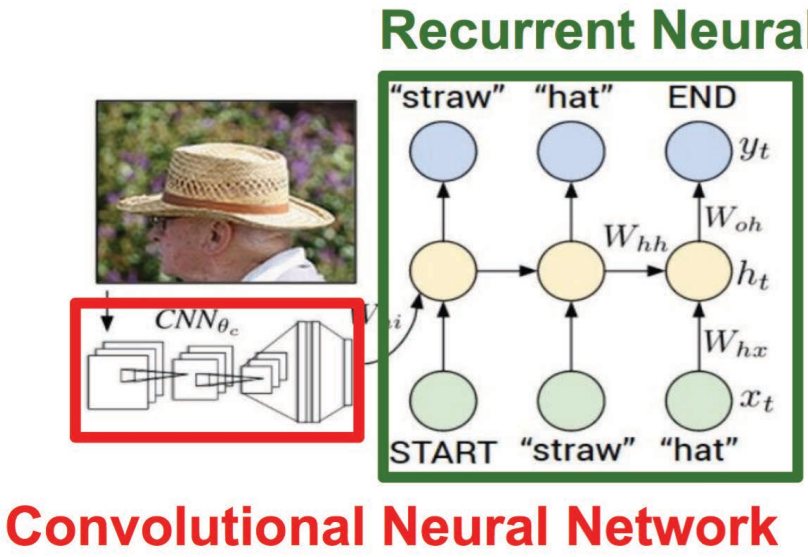
Applications

- **Code-like** using Linux 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;
}
```

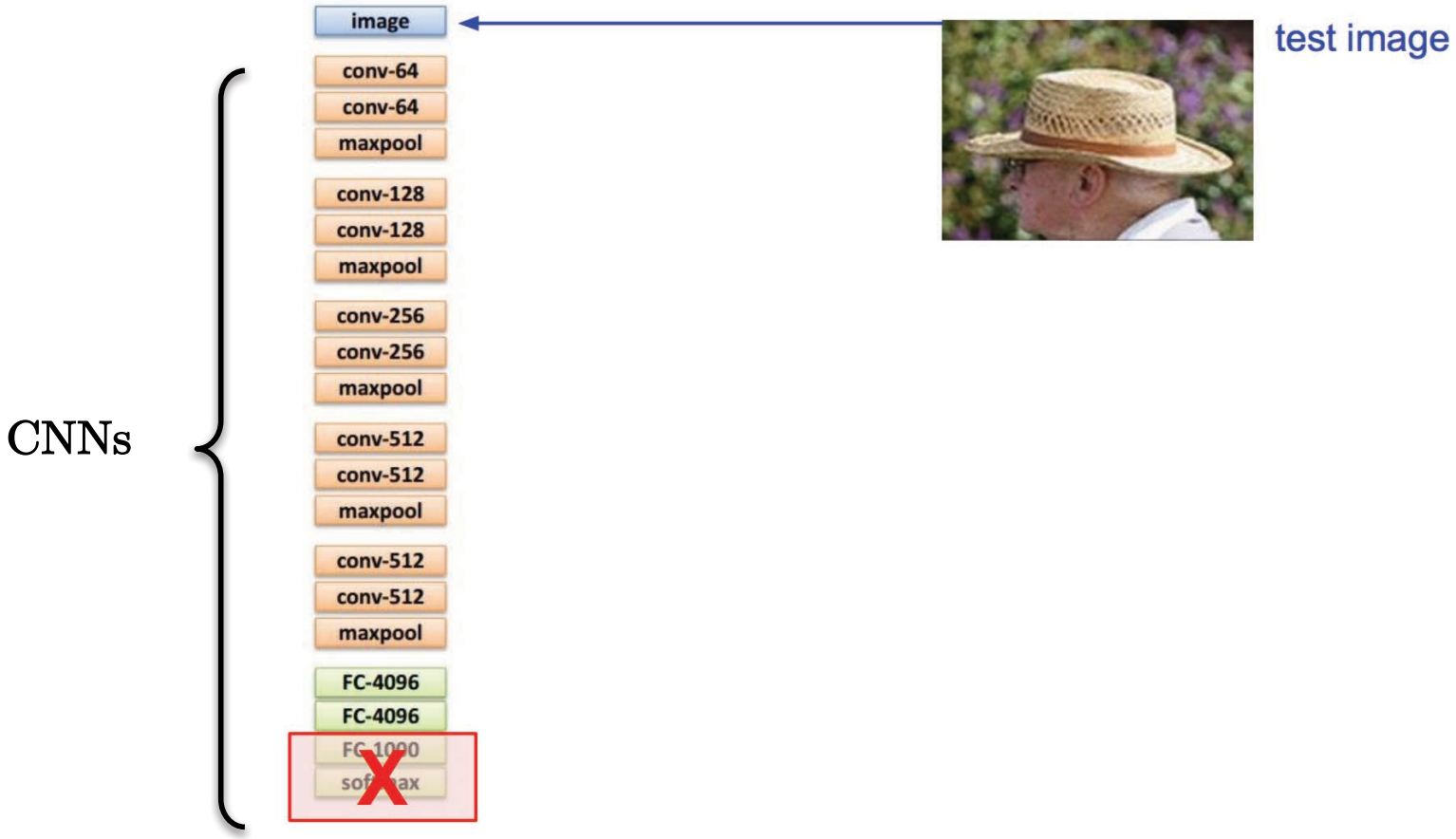
Applications

- **Image Captioning** using RNNs and CNNs



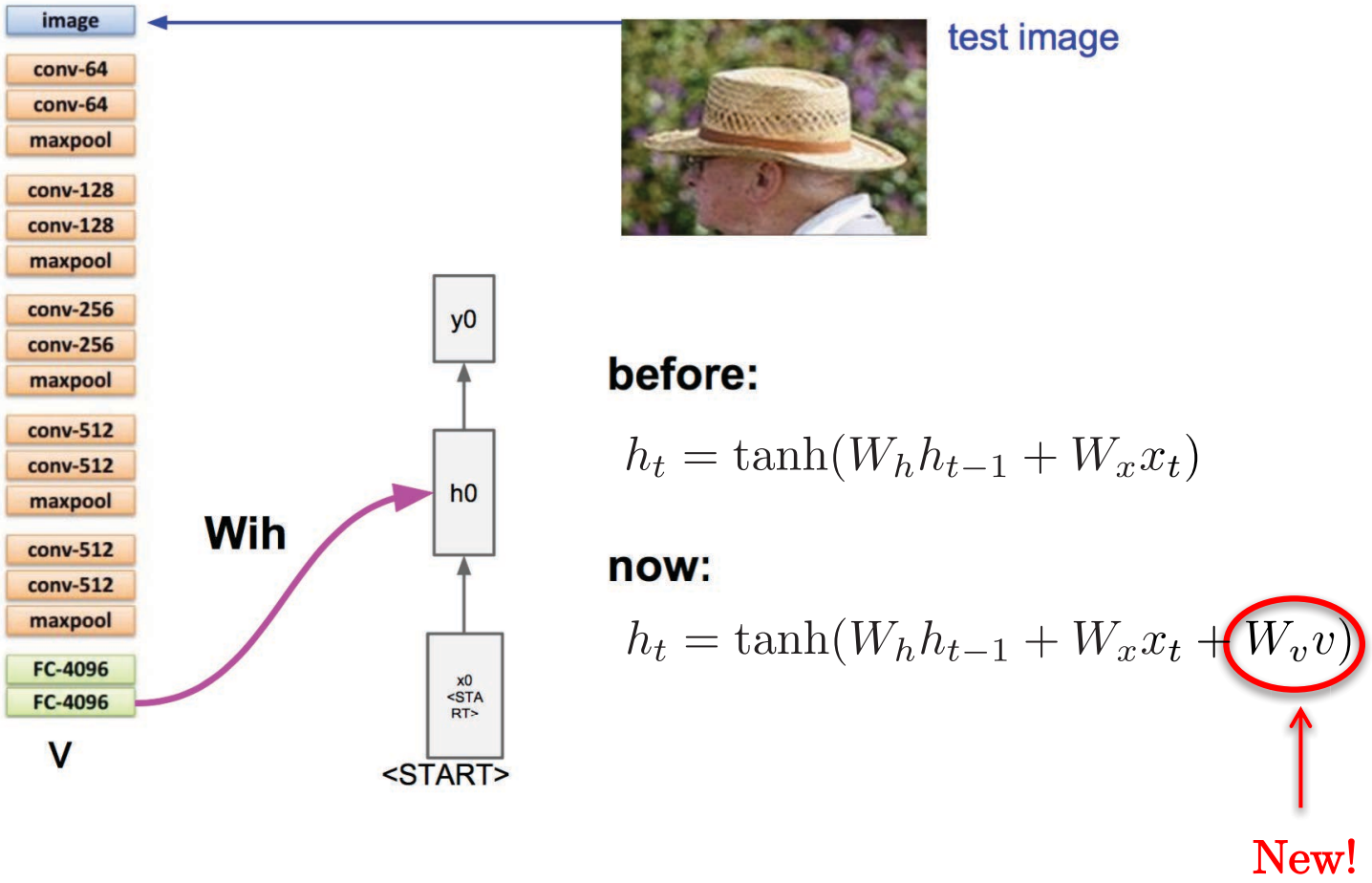
Design

- **Step 1:** Remove the last FC layer and softmax classifier in CNNs (classification is not needed, only visual feature extractors).



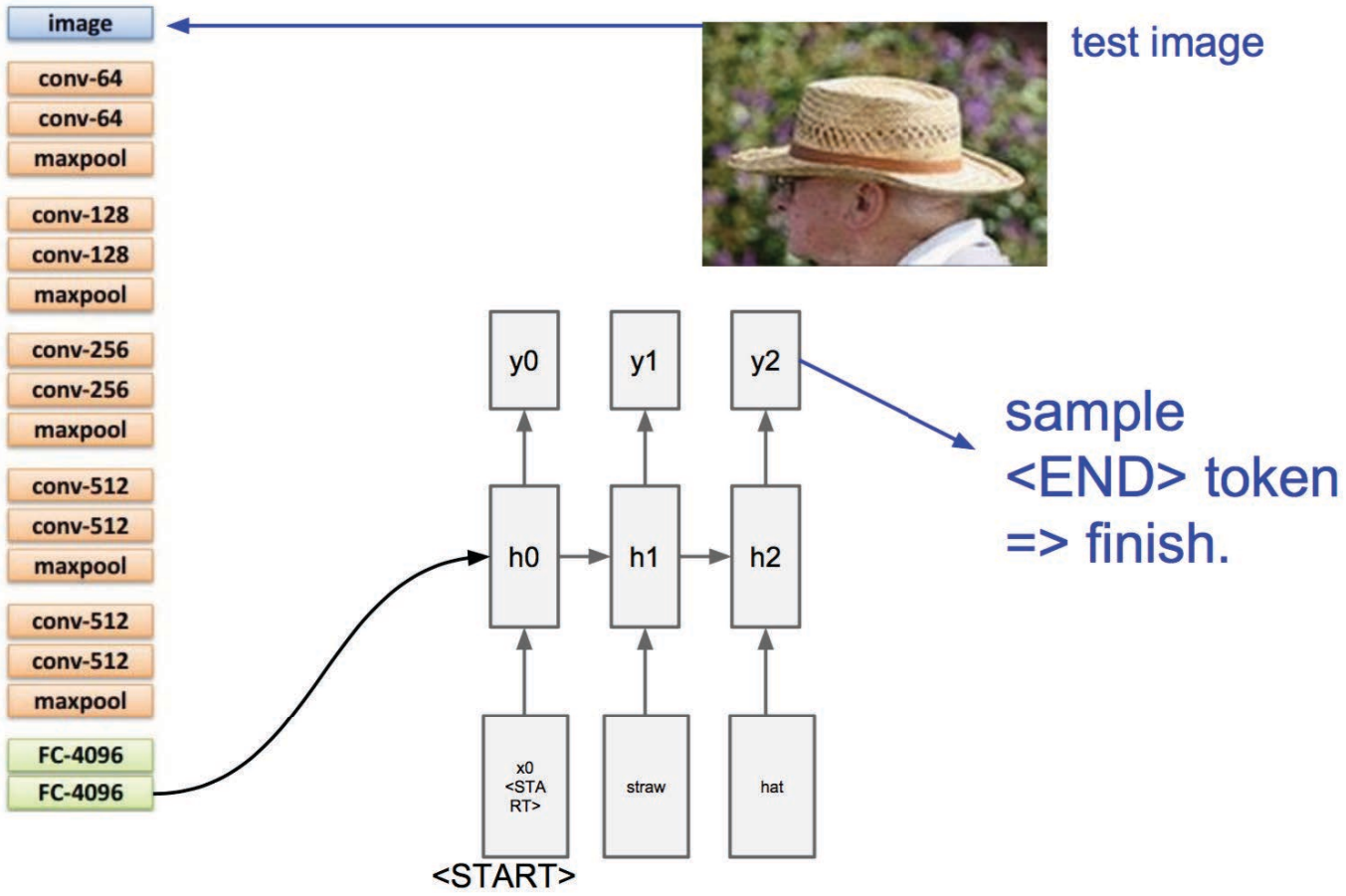
Design

- **Step 2:** Connect CNN output to RNN.



Design

- **Step 3:** Final architecture. Learn the CNN and RNN weights by backpropagation.



Results

"man in black shirt is playing guitar."

"construction worker in orange safety vest is working on road."

"two young girls are playing with lego toy."

"boy is doing backflip on wakeboard."

"a young boy is holding a baseball bat."

"a cat is sitting on a couch with a remote control."

"a woman holding a teddy bear in front of a mirror."

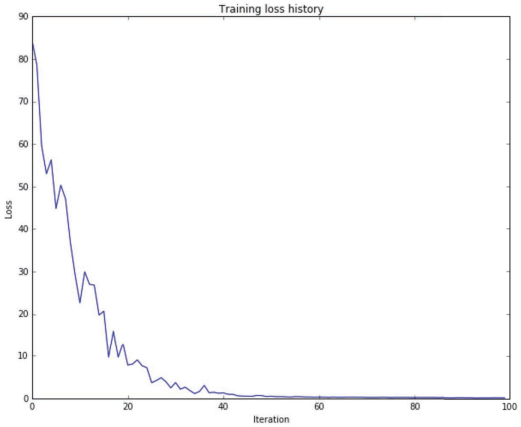
"a horse is standing in the middle of a road."

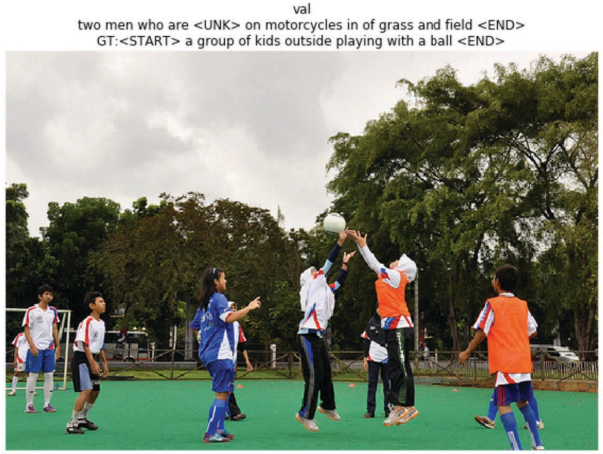
Demo

- VRNN: Run **code02.ipynb**
- LSTM: Run **code03.ipynb**

Microsoft COCO

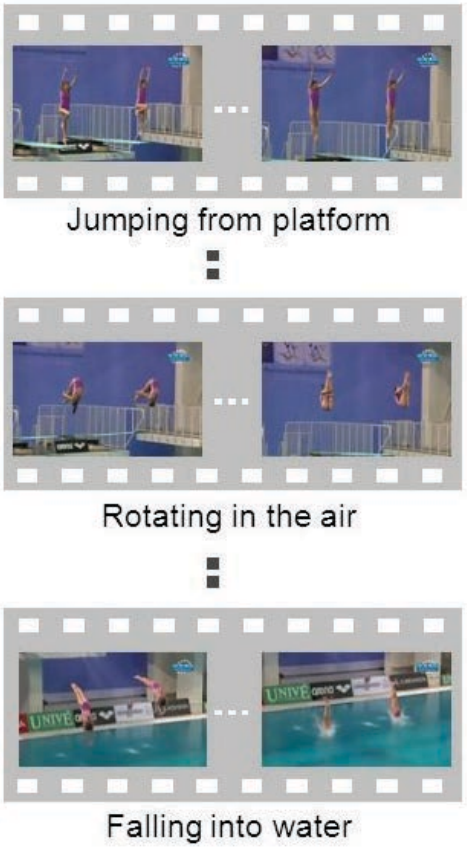
For this exercise we will use the 2014 release of the Microsoft COCO dataset consists of 80,000 training images and 40,000 validation images, each annota

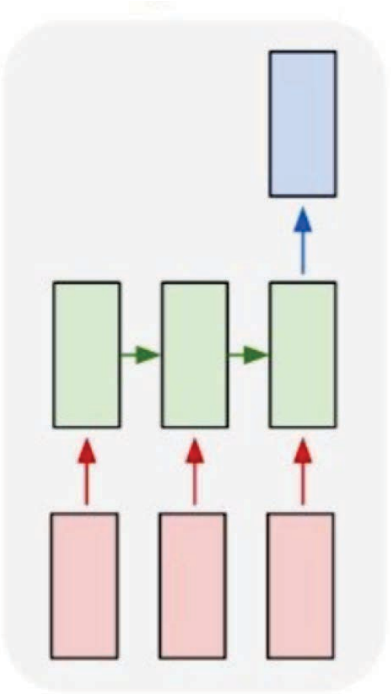




Applications

- **Many-to-one:**
  1. *Sentiment analysis*
  2. *Video classification*





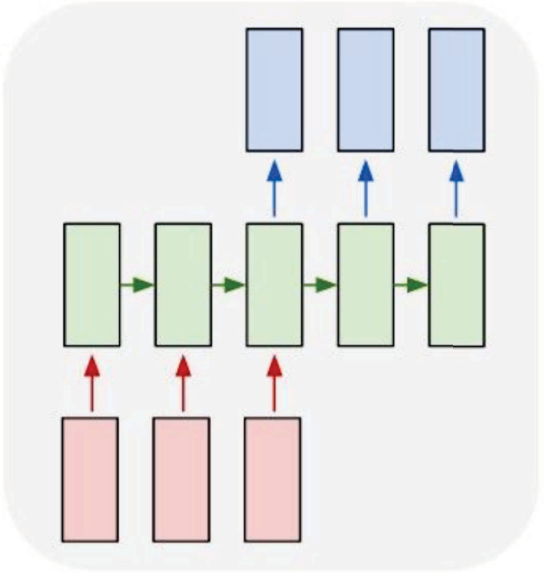
Prod:
The hotel is really beautiful. Moviestar feeling and decadence from yesterday. The pool is designed by Johnny Weissmuller. So it was a trendy pool. The food at the restaurant was really good. Very nice and helpful service at the frontfesk.

Cons: this is what made my grade a 3 instead of 4. We had problems to get the wi-fi working. If you're not depend this is not interesting. We talked several times with the front desk.

When we're there they had party event in the pool area between noon and 5 PM. The pool area was occupied with young party animals. So the area wasn't fun for UD.

Applications

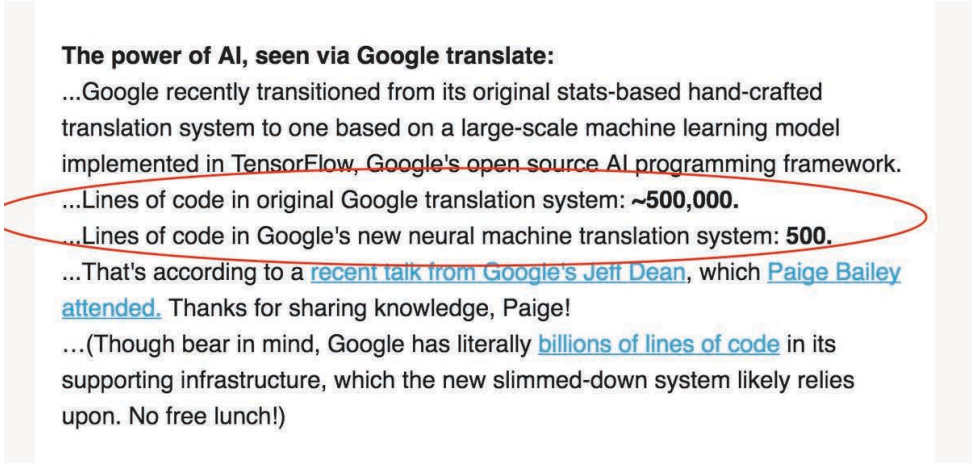
- **Many-to-many:**
  1. *Seq-to-seq: Machine translation*
  2. *Seq-to-seq: Q&A's*
  3. *Speech recognition: Speech-to-text*



Machine Translation

- Machine translation (NLP)

  Google's Neural Machine Translation system (Sept 27, 2016)



The power of AI, seen via Google translate:

...Google recently transitioned from its original stats-based hand-crafted translation system to one based on a large-scale machine learning model implemented in TensorFlow, Google's open source AI programming framework.

...Lines of code in original Google translation system: **~500,000.**

...Lines of code in Google's new neural machine translation system: **500.**

...That's according to a recent talk from Google's Jeff Dean, which Paige Bailey attended. Thanks for sharing knowledge, Paige!

...(Though bear in mind, Google has literally billions of lines of code in its supporting infrastructure, which the new slimmed-down system likely relies upon. No free lunch!)

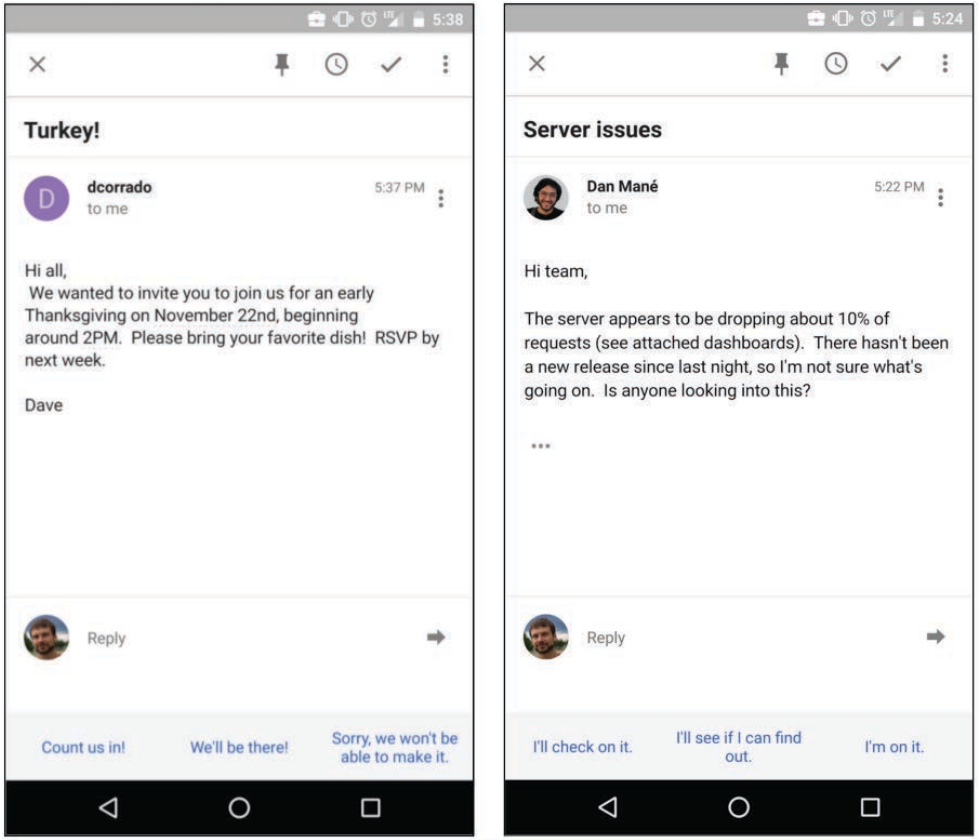
"Today we announce the Google Neural Machine Translation system (GNMT), which utilizes state-of-the-art training techniques to achieve **the largest improvements to date for machine translation quality**."

Input sentence:	*Translation (PBMT):*	*Translation (GNMT):*	*Translation (human):*
李克強此行將啟動中加總理年度對話機制，與加拿大總理杜魯多舉行兩國總理首次年度對話。	Li Keqiang premier added this line to start the annual dialogue mechanism with the Canadian Prime Minister Trudeau two prime ministers held its first annual session.	Li Keqiang will start the annual dialogue mechanism with Prime Minister Trudeau of Canada and hold the first annual dialogue between the two premiers.	Li Keqiang will initiate the annual dialogue mechanism between premiers of China and Canada during this visit, and hold the first annual dialogue with Premier Trudeau of Canada.

Questions-Answers

- Questions-Answers task:

  Gmail Automatic respond to your emails



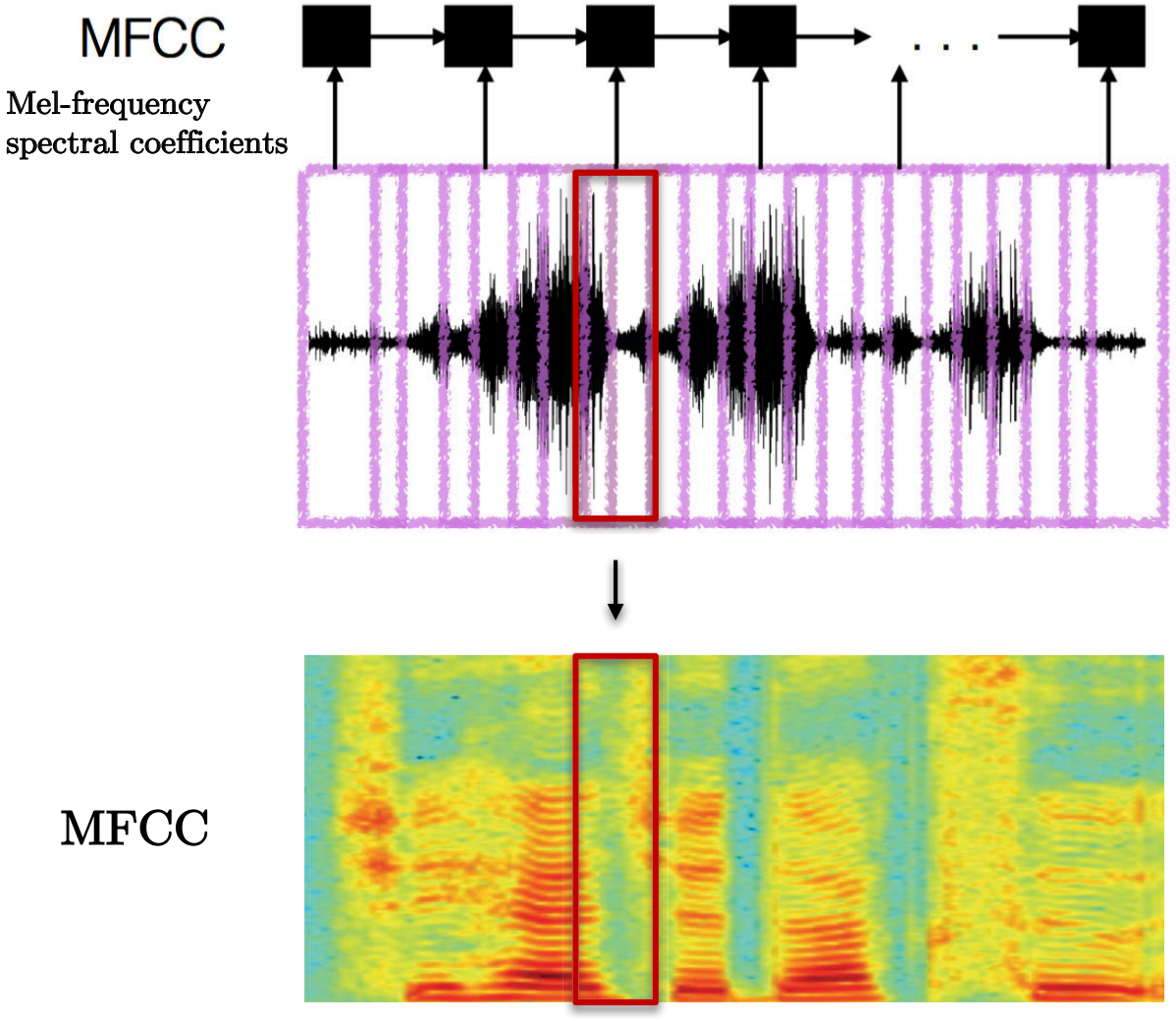
Speech Recognition

- Speech-to-text

- Performances: 5.1% error (Microsoft Research, Xiong-et.al 2017), as good as humans

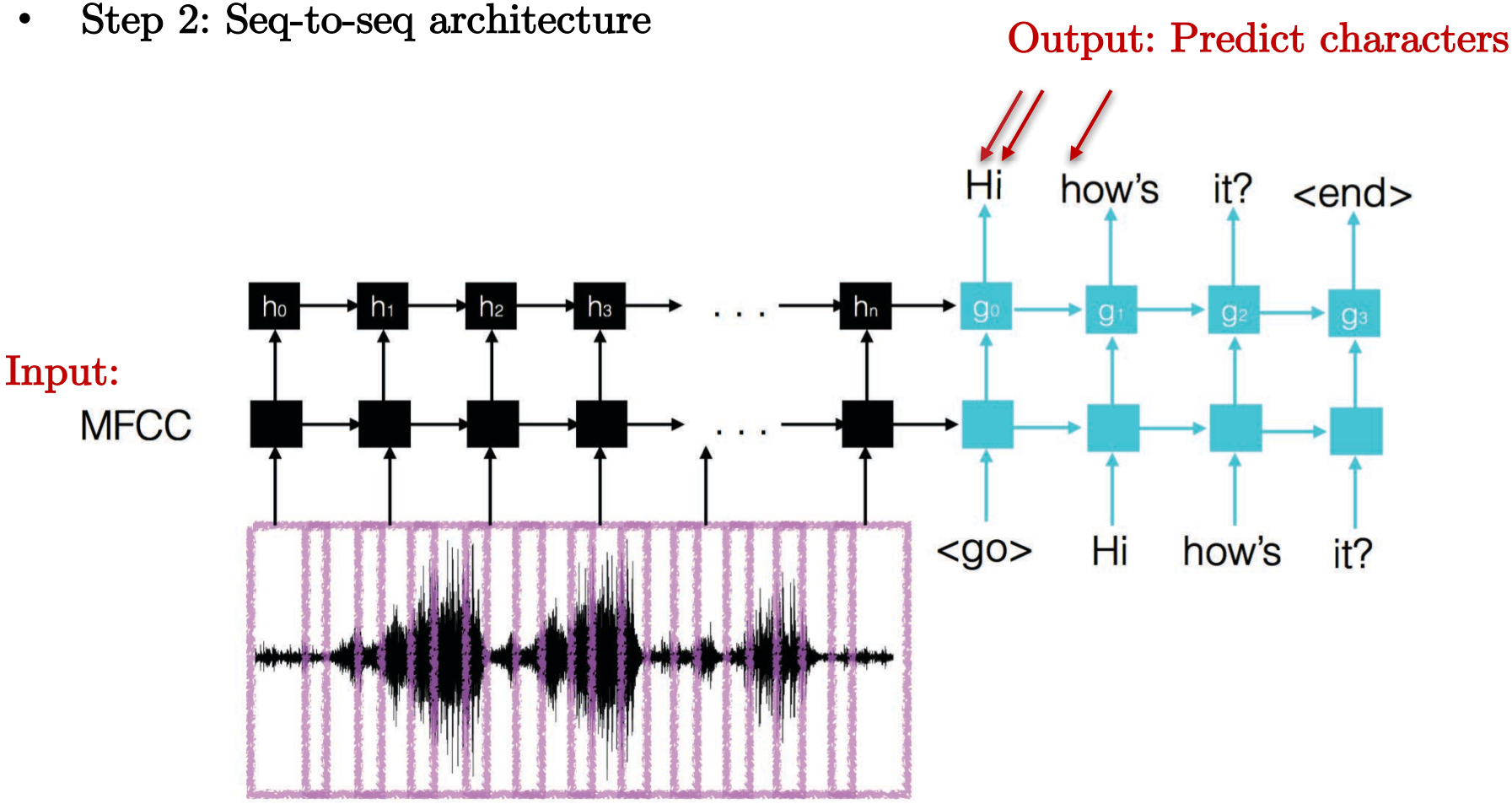
Speech Recognition System

- Step 1/Input: Time-frequency data (spectogram)



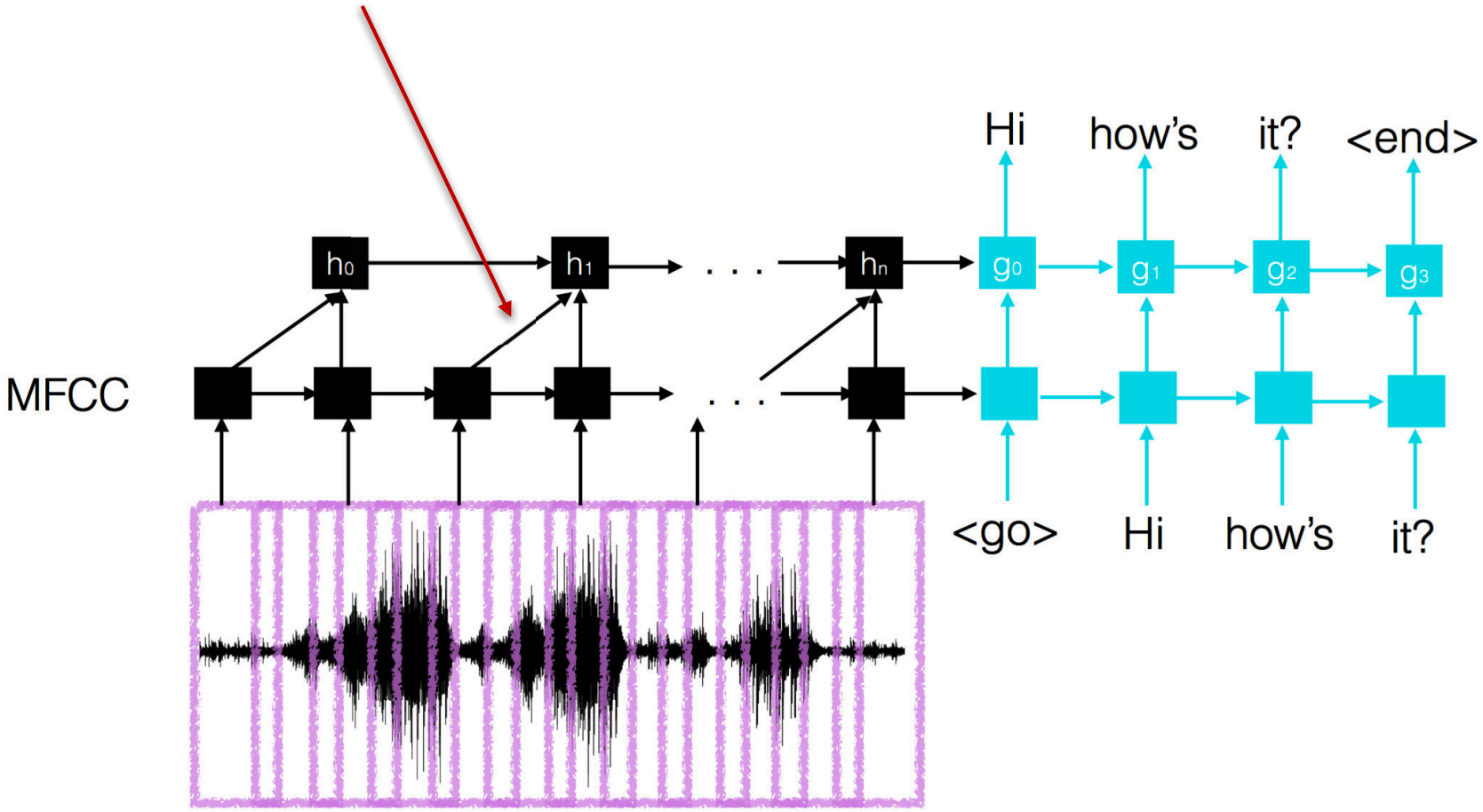
Speech Recognition System

- Step 2: Seq-to-seq architecture



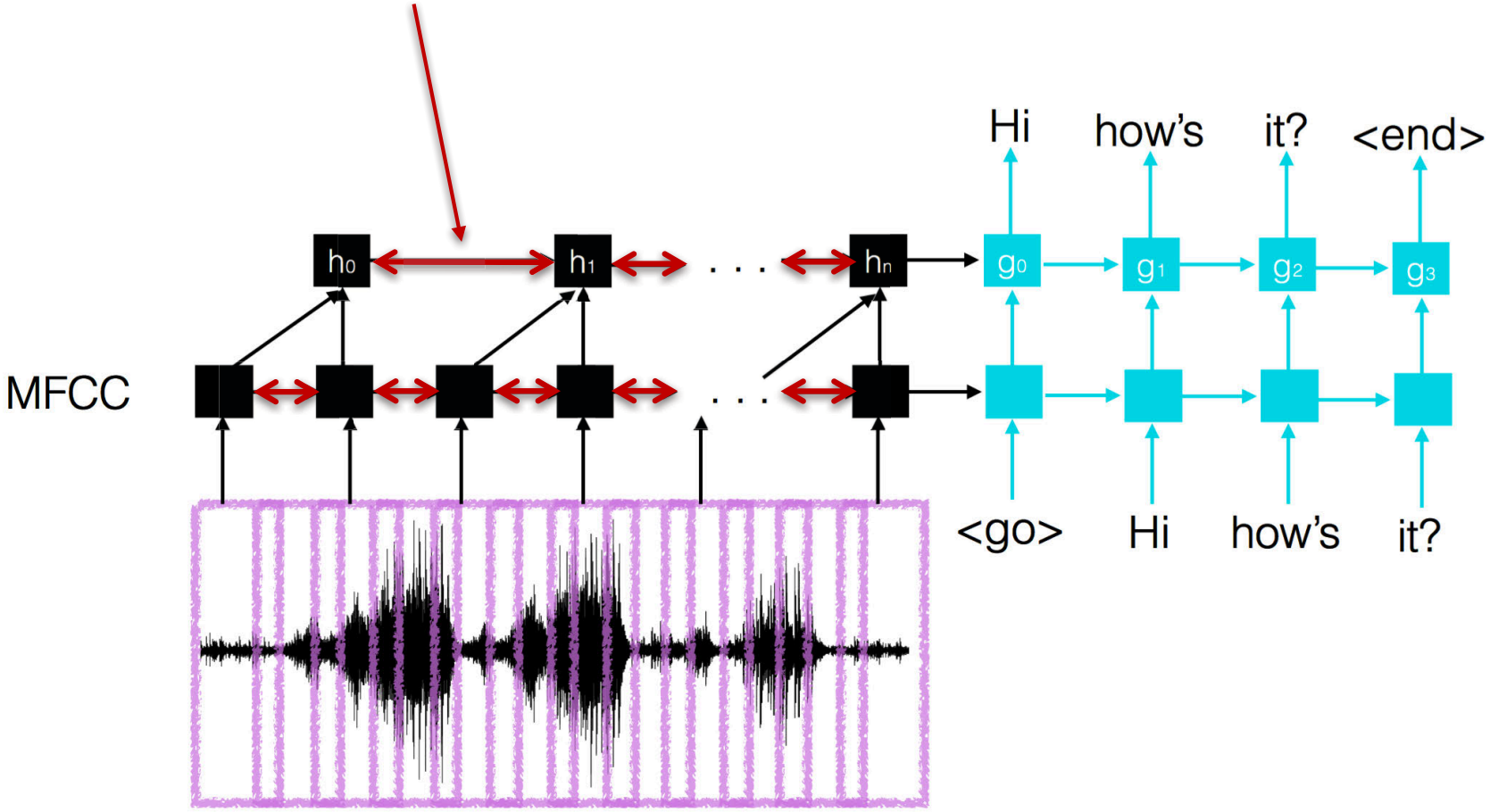
Speech Recognition System

- Step 3: Hierarchical architecture



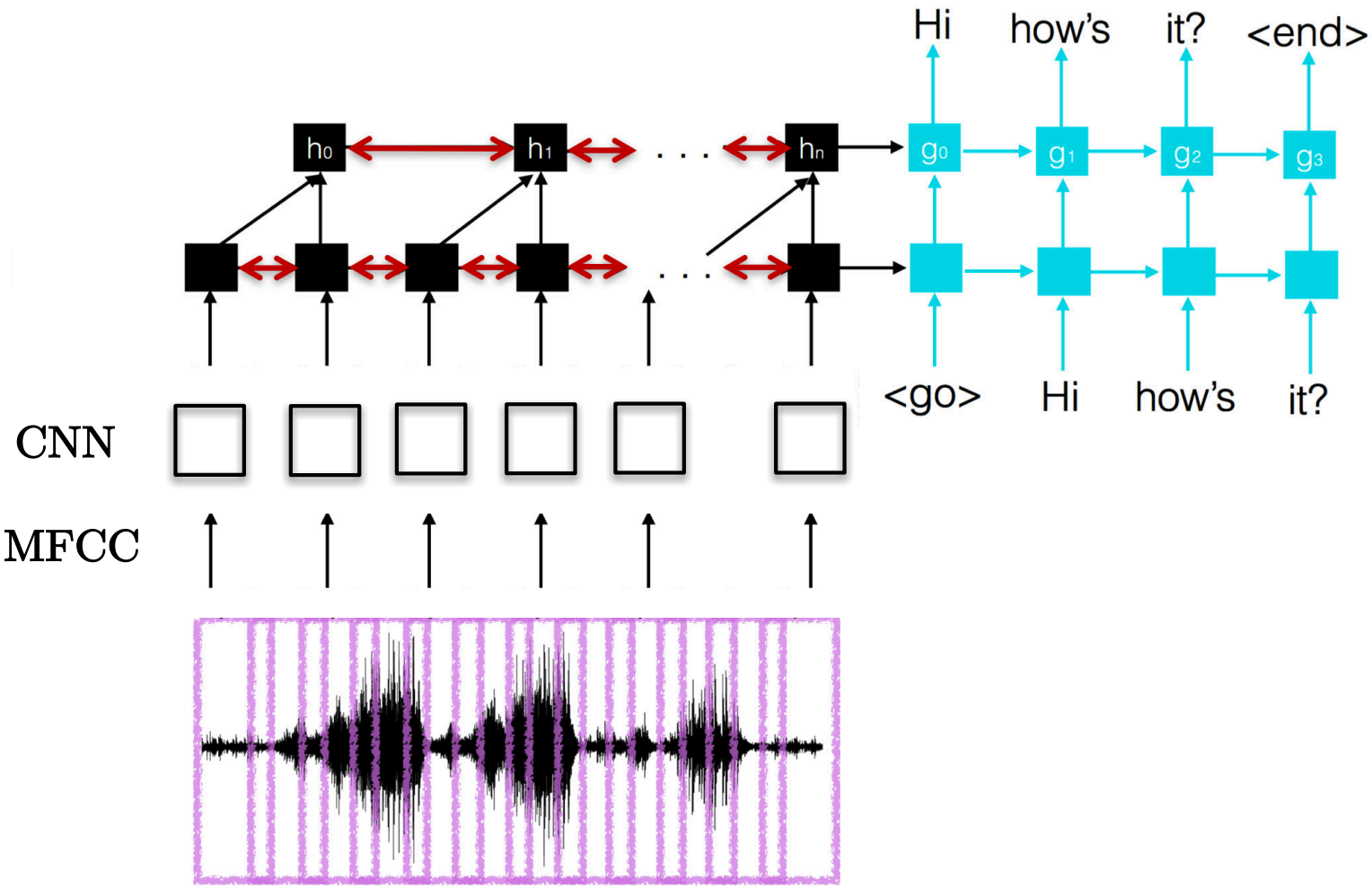
Speech Recognition System

- Step 4: Use bi-directional LSTM



Speech Recognition System

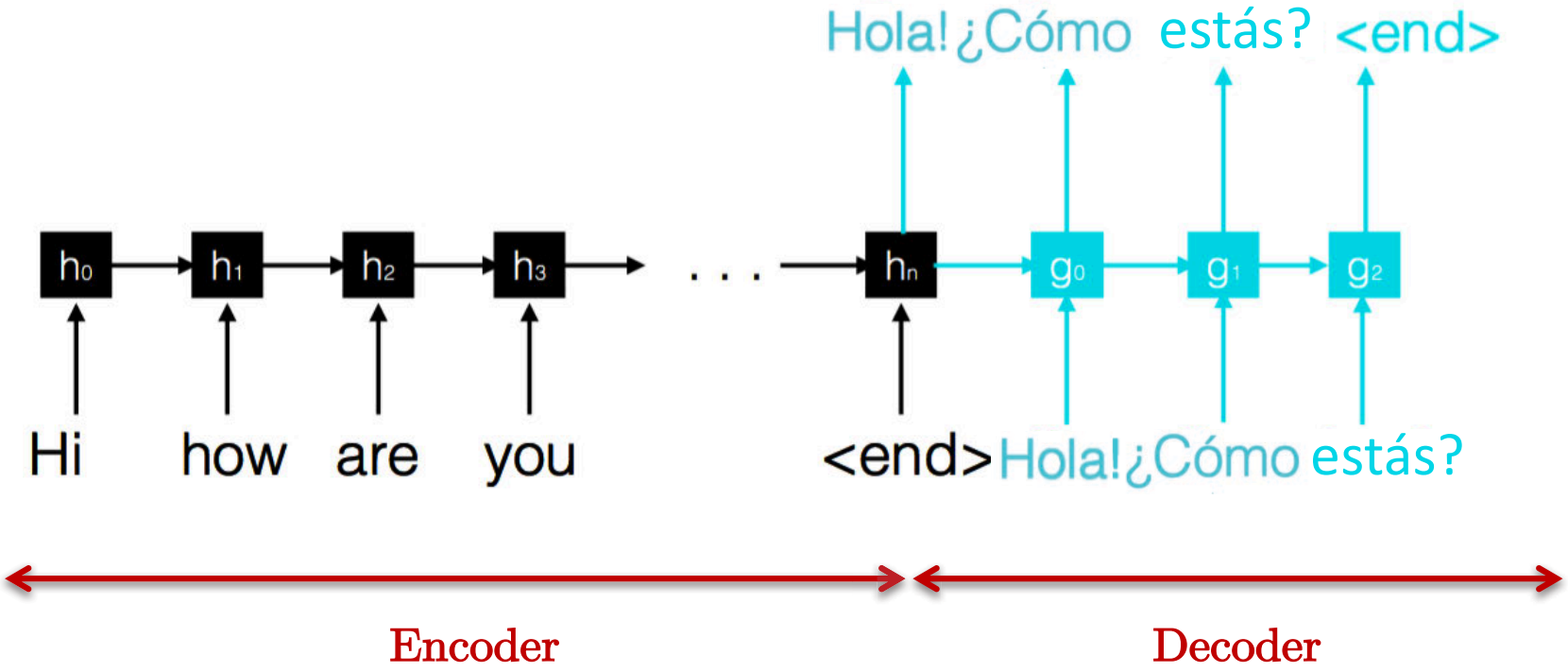
- Step 5: Use CNN on spectrogram before LSTM



Outline

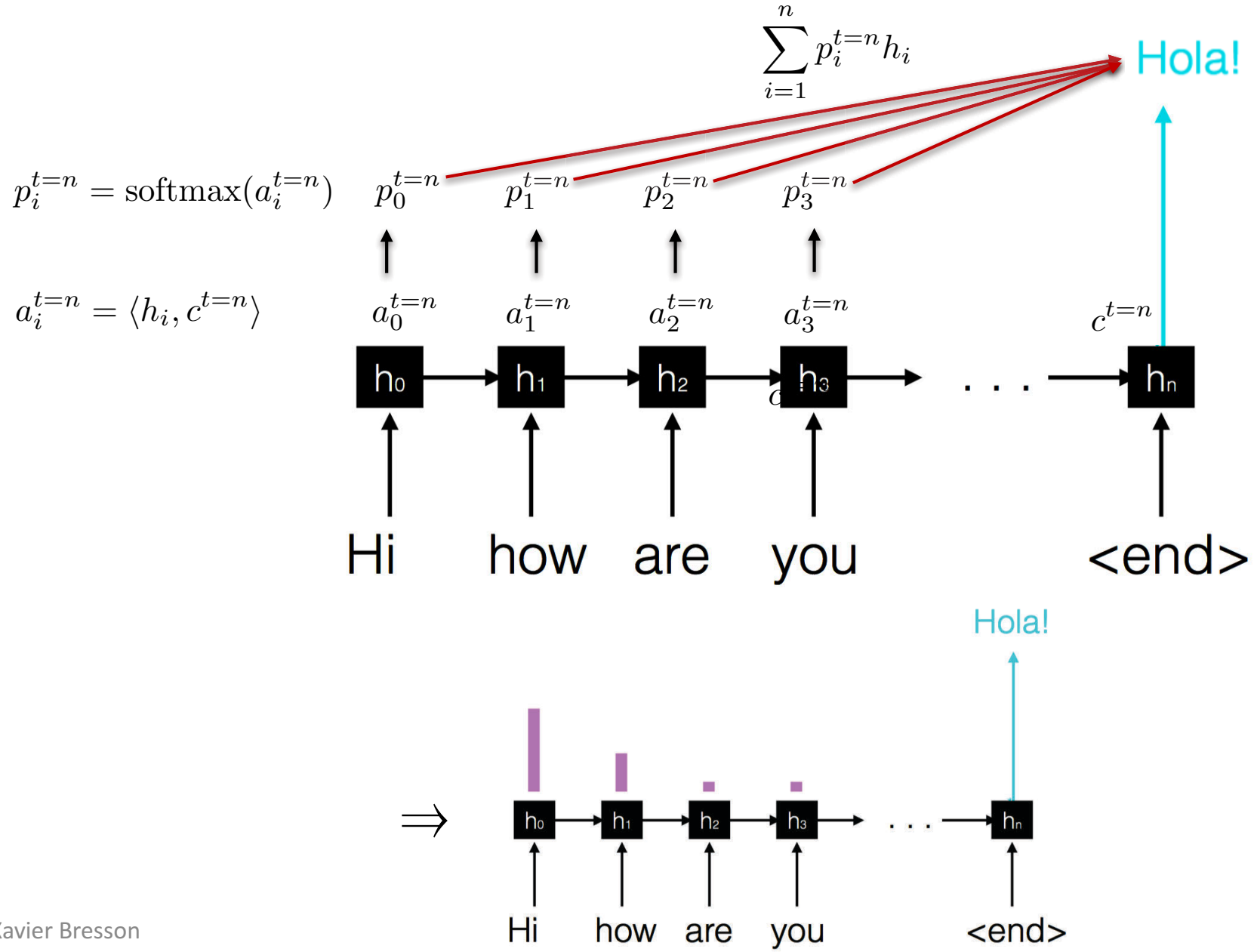
- Neural Networks for Data Sequence
- Recurrent Neural Networks
- Long Short-Term Memory (LSTM)
- Deep RNNs
- Recent Architectures
- Applications
- **Attention Networks**
- Conclusion

Machine Translation with Attention



Attention Mechanism

- **Attention to "Hi" with "Hola" can be learned:**



Attention Mechanism

- Attention to "How" with "Como" can be learned:

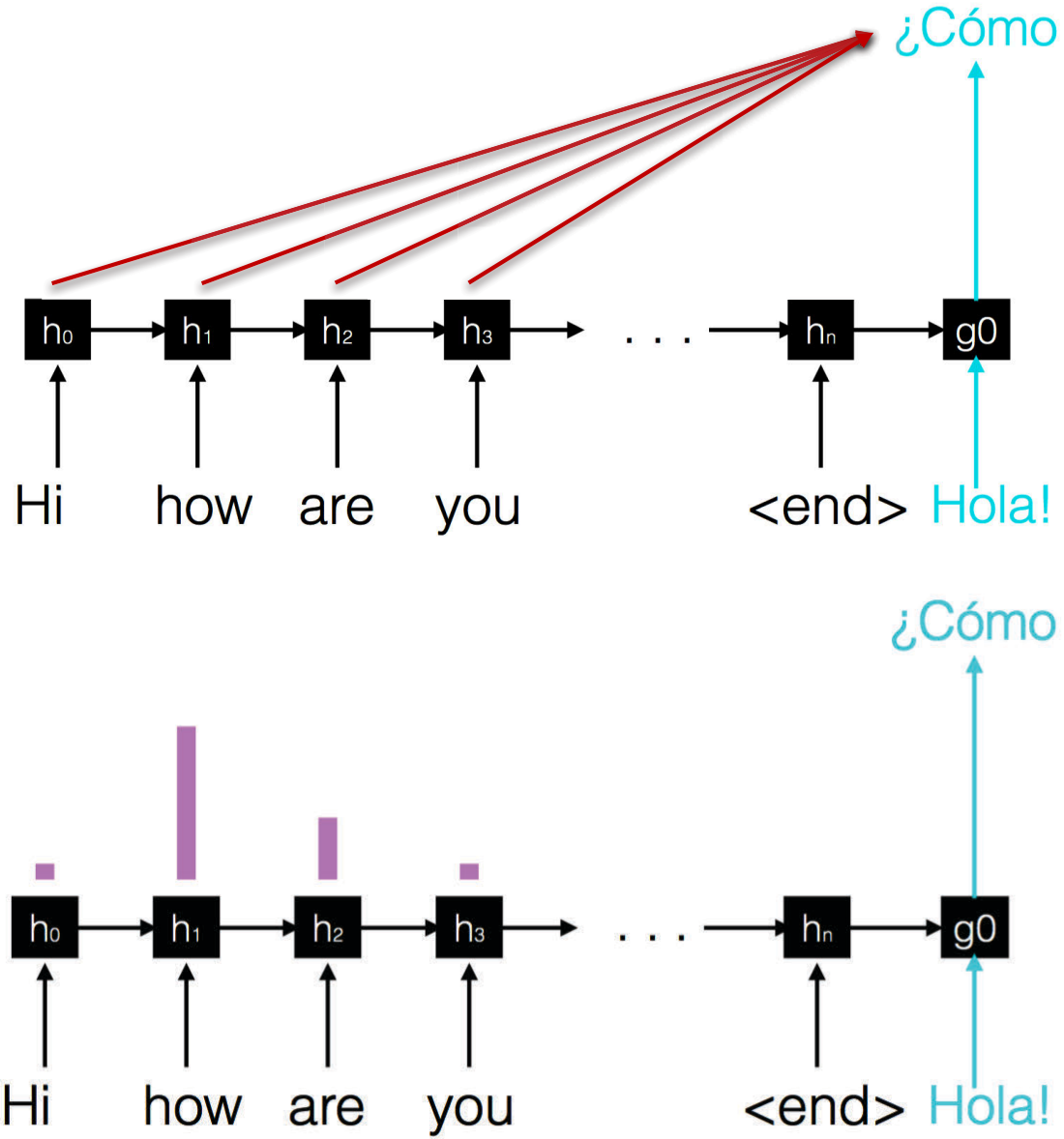
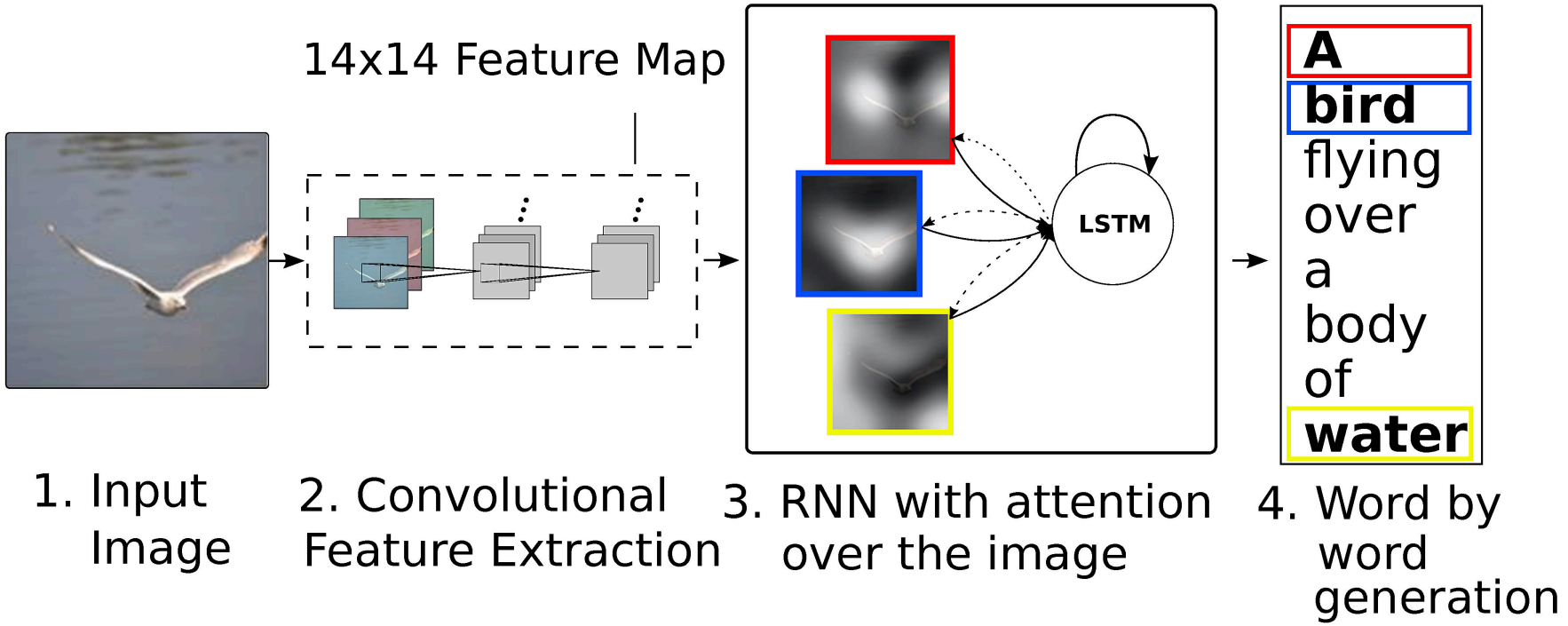
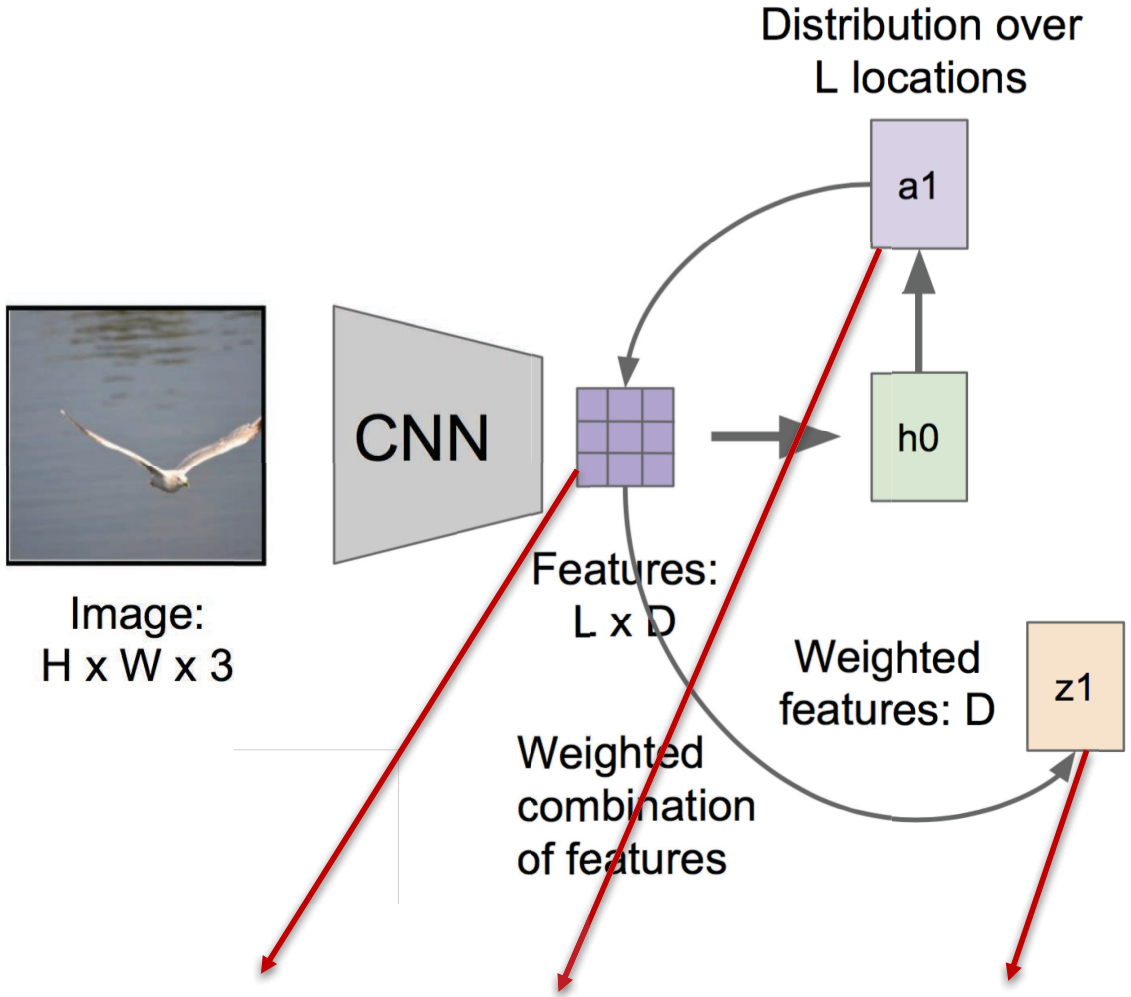


Image Captioning with Attention

- Goal: A RNN focuses its attention at a different spatial location when generating each word [Xu-et.al' 15]

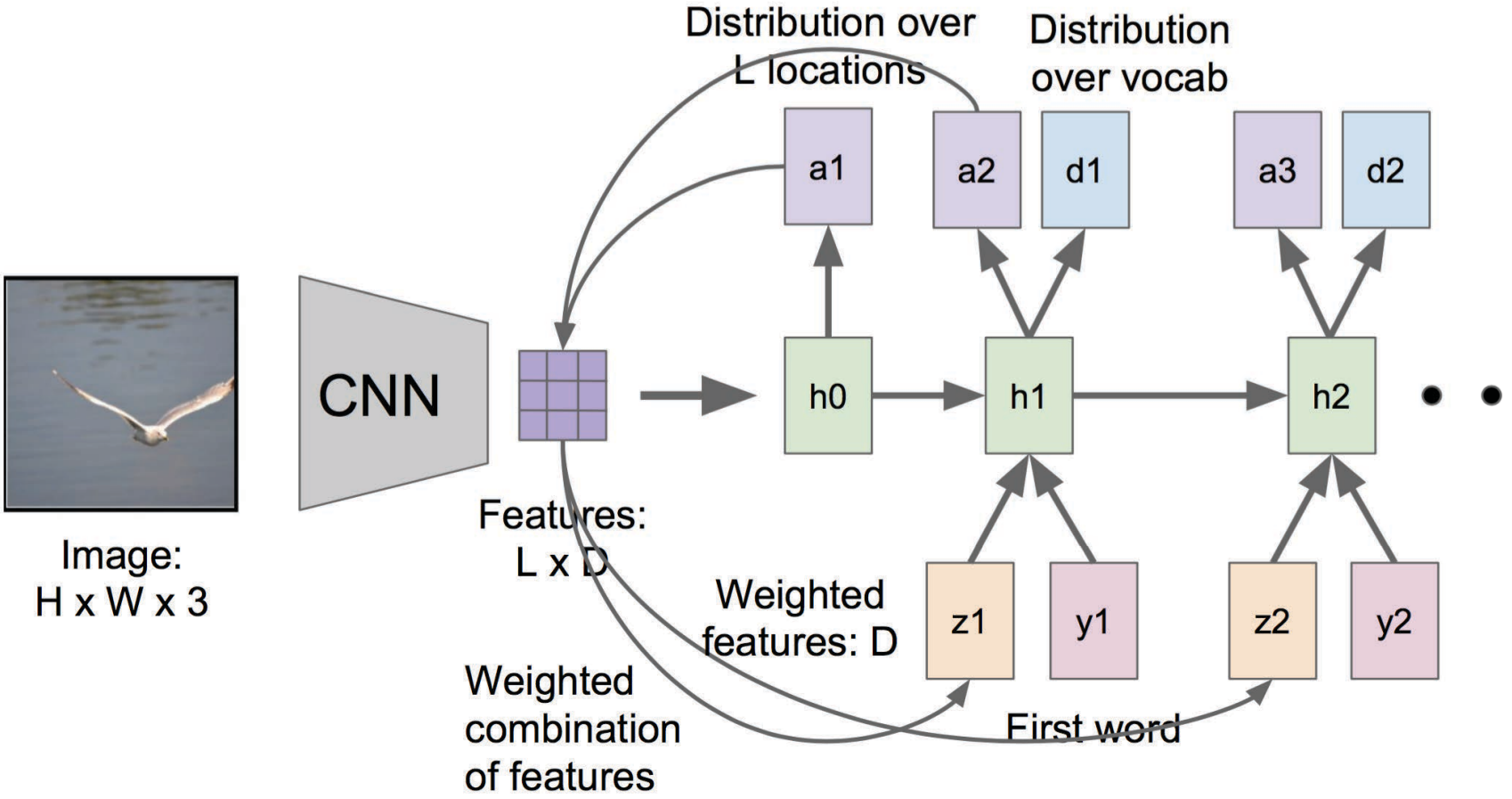


Attention Mechanism



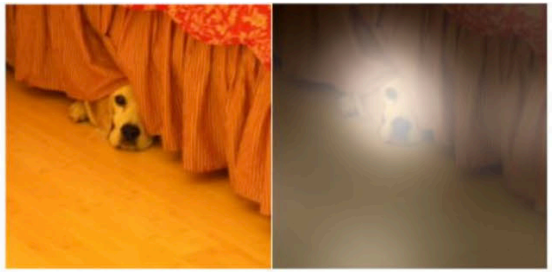
$$p^{t=1} = \text{softmax}\Big(\underbrace{\begin{bmatrix} f_1 & & \\ & & \\ & & f_L \end{bmatrix}}_{f} \times \underbrace{\begin{bmatrix} a_1 & & \\ & & \\ & & a_L \end{bmatrix}}_{a^{t=1}} \Big) \Rightarrow \begin{bmatrix} & & \\ & & \\ & & \end{bmatrix} \quad z^{t=1} = \sum_{i=1}^{L} p_i^{t=1} . f_i$$

Attention Mechanism



Results


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.

Outline

- Neural Networks for Data Sequence
- Recurrent Neural Networks
- Long Short-Term Memory (LSTM)
- Deep RNNs
- Recent Architectures
- Applications
- Attention Networks
- **Conclusion**

Conclusion

- RNNs offer lots of **flexibility** in NN architecture, and so can be applied to **many tasks**.
- RNNs are **not** able to learn very long-term dependencies (even LSTM). Hot research topic.
- **Open questions:**
  1. *Unified architecture design, fundamental principles*
  2. *Better understanding*
  3. *Why (generalization) performances are so good?*

Questions?