

# Recurrent Neural Networks

Armand Joulin

Facebook AI research

# Introduction

- How to train a deep network:
  - **Forward pass:** apply your network to your data
  - **Evaluation:** Compute the error given by your loss function
  - **Backward pass:** propagate this error through the network
  - **Update** the parameters with gradient descent
- What happens when your data are **sequences?**
  - Example: language modelling, weather forecast, videos...
  - Take advantage of the structure of the data to learn better model?

# Sequence modeling

- At each time step  $t$ , we receive an input  $x(t)$  and we want to learn the probability of the output  $y(t)$
- We suppose that the output  $y(t)$  depends on the past inputs, i.e.  $x(1), \dots, x(t)$
- In other words we are interested in modeling the probability :  
$$P(y(t) \mid x(1), \dots, x(t))$$

# Examples

- Language modelling:

- Predicting the distribution of a sentence  $S$

$$P(S) = P(w_1, \dots, w_{|S|})$$

- Since a sentence is a sequence, this means:

$$P(S) = \prod_t P(w_t \mid w_{t-1}, \dots, w_1)$$

- Action classification in a video:

- Predicting if action is happening based on the past frames:

$$P(a_t \mid f_{t-1}, \dots, f_1)$$

# Example: Language modelling

- One-hot encoding (or 1-of-K encoding):
  - Vocabulary = {"cat", "in", "is", "room", "the", ".}"
  - “cat” = [ 1 0 0 0 0 ]
  - “in” = [ 0 1 0 0 0 ]
  - “is” = [ 0 0 1 0 0 ]
  - “room” = [ 0 0 0 1 0 ]
  - “the” = [ 0 0 0 0 1 ]
  - “.” = [ 0 0 0 0 0 1 ]

# Example: Language modelling

- N-grams models are very popular to model a sequence:
  - Bigram:

$$P(\text{"the cat is in the room."}) = P(\text{"the"})P(\text{"cat" | "the"})P(\text{"is" | "cat"})P(\text{"in" | "is"}) \\ \times P(\text{"the" | "in"})P(\text{"room" | "the"})P(\text{". | "room"})$$

- Trigram:

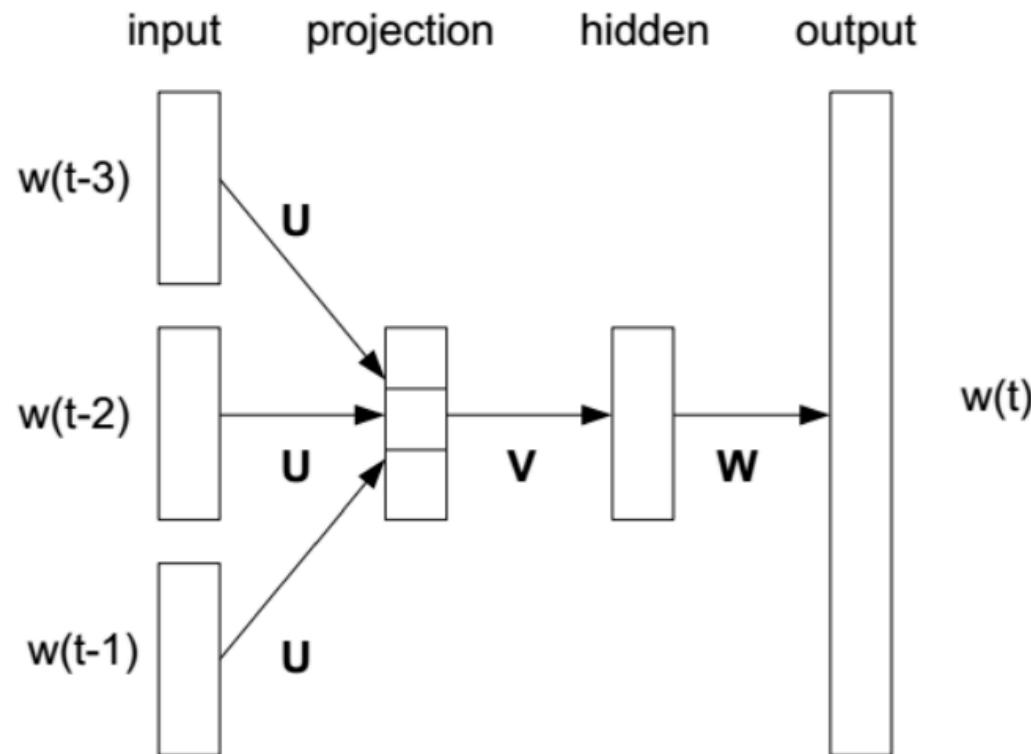
$$P(\text{"the cat is in the room."}) = P(\text{"the"})P(\text{"cat" | "the"})P(\text{"is" | "cat", "the"})P(\text{"in" | "is", "cat"}) \\ \times P(\text{"the" | "in", "is"})P(\text{"room" | "the", "in"})P(\text{". | "room", "the"})$$

- ...

# Example: Language modelling

- N-grams can be modeled by feed-forward networks (Bengio et al. 2003):

- 4-grams:



# Example in Torch

```
require "nn"
local i2w = {"ball", "is", "red", "the"}
local w2i = {ball = 1, is = 2, red = 3, the = 4}
local ngram = 3
local nhid = 10
local nproj = (ngram-1) * nhid
-- Bengio et al. 2003 for 3-grams:
local feedforward = nn.Sequential()
feedforward:add(nn.LookupTable(#i2w, nhid))
feedforward:add(nn.View(nproj))
feedforward:add(nn.Sigmoid())
feedforward:add(nn.Linear(nproj, nhid))
feedforward:add(nn.Sigmoid())
feedforward:add(nn.Linear(nhid, #i2w))
local criterion = nn.CrossEntropyCriterion()

feedforward:forward(torch.LongTensor{w2i["ball"], w2i["is"]})
criterion:forward(feedforward.output, w2i["red"])
```

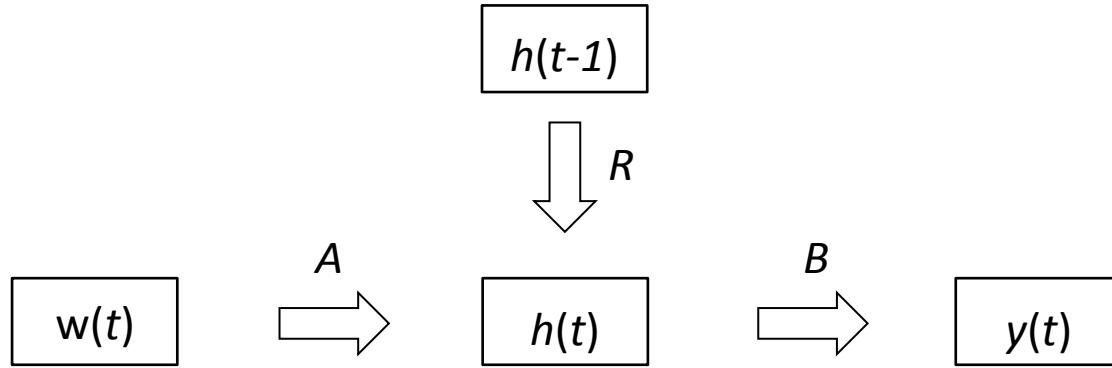
# Example: Language modeling

- Limitation of Feed-forward networks:
  - How do I set the size of the n-gram?
  - For each prediction, I need to process the past again which is inefficient
- Also we are making the strong assumption that:
$$P(w_t \mid w_1, \dots, w_{t-1}) = p(w_t \mid w_{t-1}, w_{t-2})$$
- Instead of hard-coding the important information, can we make the network learn to “remember” what is important?

# Recurrent neural network

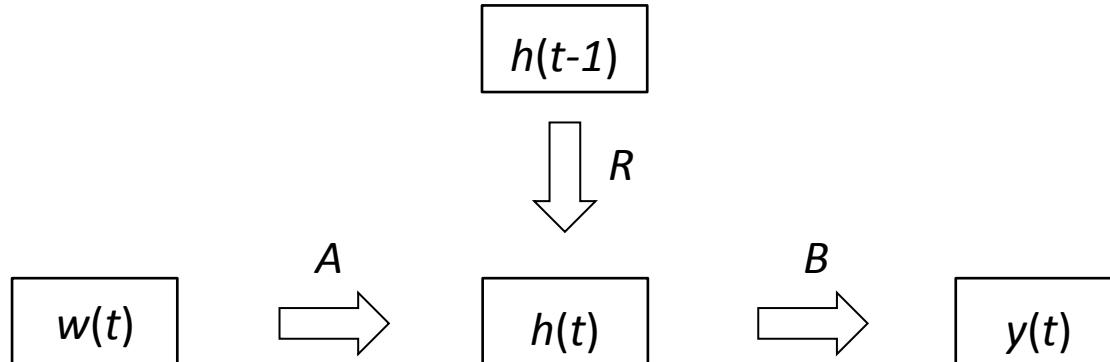
- **Main idea:** Keep a “memory” of the past in the hidden layers.
- That is, by making the previous state of the hidden layers influence the current one
- What does it mean?
  - In simple 1-layer neural network, the state  $h$  of the hidden layer only depends on the input:
$$h(t) = f(w(t))$$
  - In a recurrent model:
$$h(t) = f(w(t), h(t-1))$$

# Simple Recurrent Network



- The simple RNN contains 1 hidden layer which possesses a “Recurrent” connection
- This network is called the Elman Network (from Elman, 1990)

# Simple Recurrent Network



- The simple RNN:

$$h_t = \sigma(Aw_t + Rh_{t-1})$$

where:  $y_t = f(Bh_t)$

- $A$ ,  $B$  and  $R$  are matrices,
- Sigmoid:  $\sigma(z) = 1/(1 + \exp(-z))$
- Softmax  $f(z)_k = \exp(z_k)/(\sum_i \exp(z_i))$

# Back to the example

Vocabulary = {"cat", "in", "is", "room", "the", ".")}

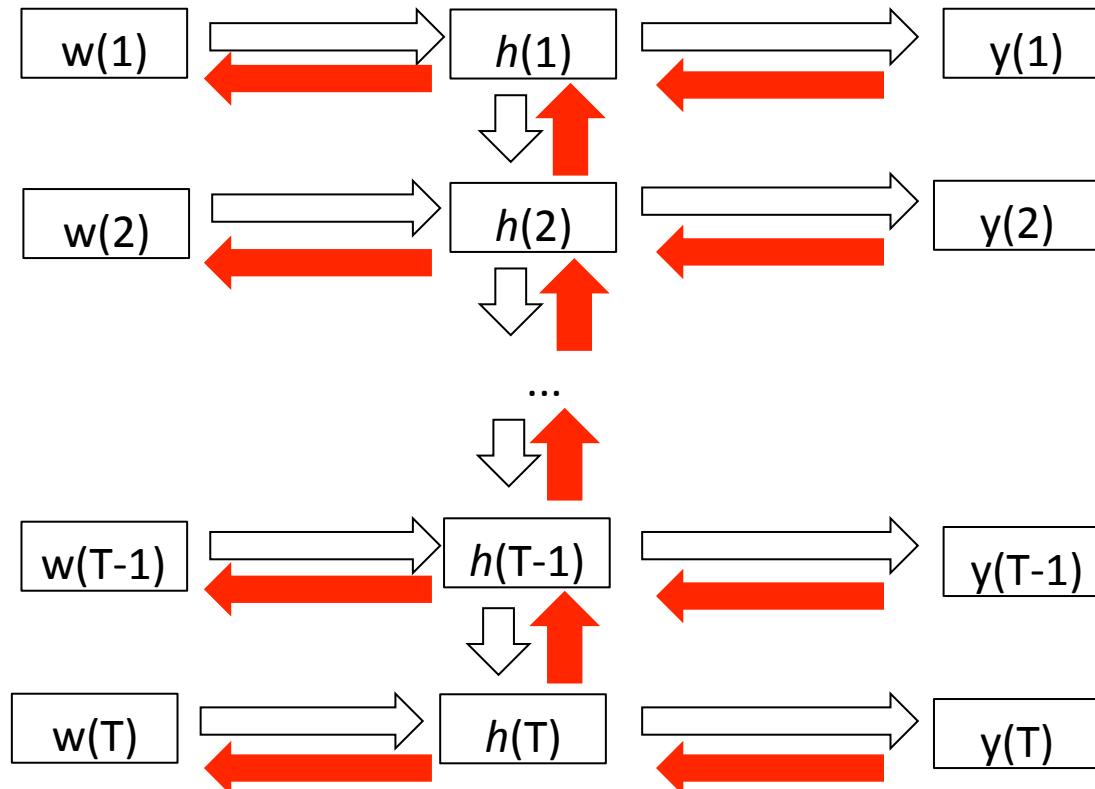
time	Input	Model	output	target
t = 1	"the"	$h(1)$	[ .5 0 0 0.5 0 0 ]	"cat" = [ 1 0 0 0 0 0 ]
t = 2	"cat"	$h(2)$	[ 0 0.4 0.4 0 0 .2 ]	"is" = [ 0 0 1 0 0 0 ]
t = 3	"is"	$h(3)$	[ 0 0.3 0 0 0.5 0.2 ]	"in" = [ 0 1 0 0 0 0 ]
t = 4	"in"	$h(4)$	[ 0 0 0 0 .8 0.2 ]	"the" = [ 0 0 0 0 1 0 ]

# How to train a RNN?

- **Problem:** each output depends on all the states of the hidden layers since  $t = 1$
- To compute the gradient: backpropagate until the beginning of the sequence
- This is called backpropagation through time (BPTT)
- RNN can be seen as very deep neural networks with weight sharing

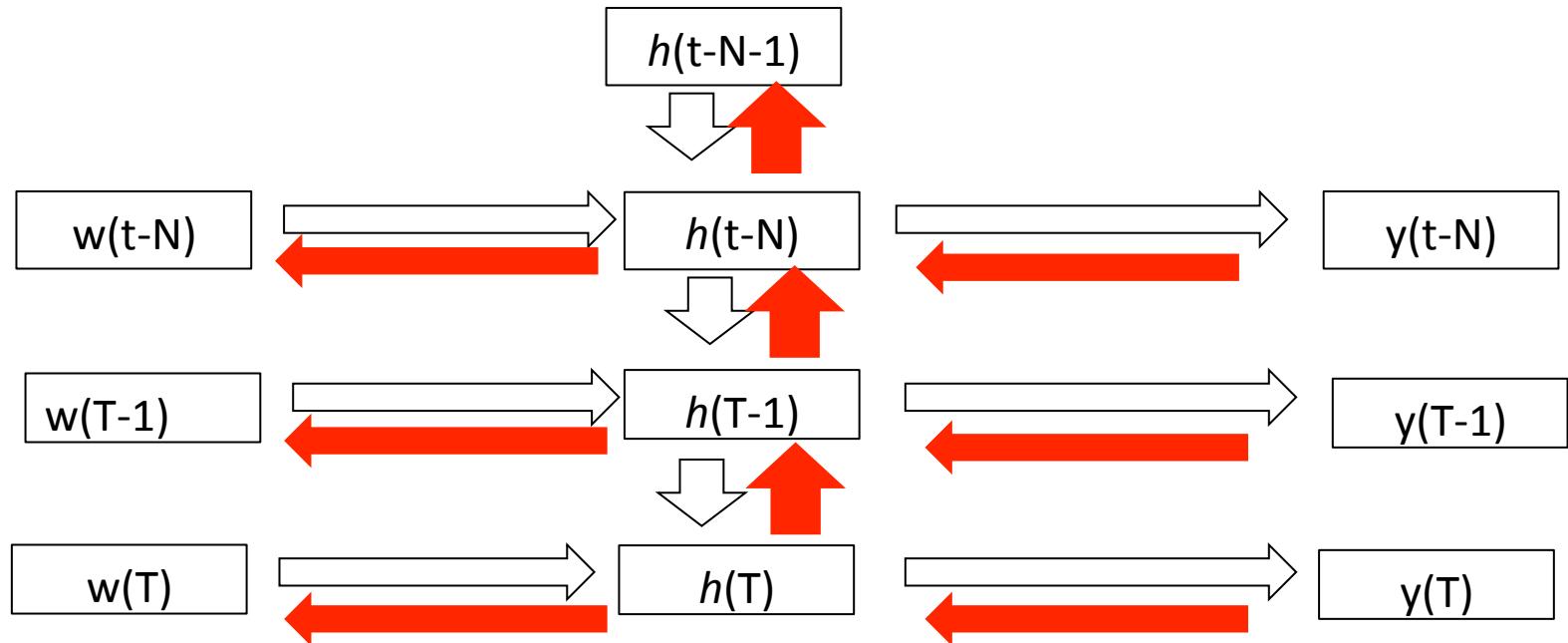
# Backpropagation through time

- Unrolling the model since the beginning
- This means that computing a gradient is  $O(T)$
- backward propagation of error in red:



# Backpropagation through time

- Trick to make it practical :
  - unfold the network for N fixed step  $\rightarrow O(N)$
  - compute the gradient every N steps
  - $\rightarrow$  gradient can be computed efficiently **online**



# Backpropagation through time

- Depending on the eigenvalues of the recurrent matrix  $R$ , two possible issues (Bengio, 94):
  - $\max(\text{eigenvalue}) > 1 \rightarrow \text{Exploding gradient}$
  - $\max(\text{eigenvalue}) < 1 \rightarrow \text{Vanishing gradient}$

# Backpropagation through time

- **max(eigenvalue) > 1 → Exploding gradient:**  
As you backpropagate through time, you multiply the gradient over and over by  $R$ , making the norm of the gradient go to infinity
- Clipping or normalizing the gradient fix that problem (Mikolov et al. 2010)

# Backpropagation through time

- **$\max(\text{eigenvalue}) < 1 + \text{nonlinearity} \rightarrow \text{Vanishing gradient:}$**
- The magnitude of the gradient decreases when backpropagate through time:
  - Hard to keep very long dependency
  - Solutions have been proposed to counter that effect :
    - *Exponential trace memory* (Jordan 1987, Mozer 1989)
    - *Long Short-term Memory* (Hochreiter & Schmidhuber, 1997)

# RNN in torch

```
local sentence, ws = {"the", "ball", "is", "red"}, {}
for i,v in ipairs(sentence) do ws[i] = torch.LongTensor{w2i[v]} end

-- Parameters of the RNN (cf. slides)
local R = nn.Linear(nhid, nhid)
local A = nn.LookupTable(#i2w, nhid)
local B = nn.Linear(nhid, #i2w)

-- one column [h(t-1) ,w(t) ] -> [h(t), y(t)]
local m      = nn.Sequential()
local encoder = nn.ParallelTable():add(R):add(A)
local decoder = nn.ConcatTable():add(nn.Identity()):add(B)
m:add(encoder):add(nn.CAddTable()):add(nn.Sigmoid()):add(decoder)

-- create the RNN by cloning gradInput and output
local rnn      = {}
for t = 1, #ws- 1 do
    rnn[t] = m:clone{'weight','gradWeight','bias','gradBias'}
end
```

# RNN in torch

```
-- Forward:  
local hs, ys, out = {[0] = torch.Tensor(nhid):zero()}, {}  
for t = 1, #ws - 1 do  
    out = rnn[t]:forward{hiddens[t-1], ws[t]}  
    table.insert(hs, out[1])  
    table.insert(ys, out[2])  
end  
  
-- Backward:  
local gradh, grad, err = torch.Tensor(nhid):zero()  
for t = #ws - 1, 1, -1 do  
    criterion:forward(ys[t], ws[t+1])  
    err = criterion:backward(ys[t], ws[t+1])  
    grad = rnn[t]:backward({hs[t-1], ws[t]}, {gradh, err})  
    gradh = grad[1]  
    gradh:div(math.max(10, gradh:norm(2))) -- clipping  
end
```

# RNN in practice

Model	Perplexity
Kneser-Ney 5-gram	141
Maxent 5-gram	142
Random forest	132
Feedforward NNLM	140
Recurrent NNLM	125

- Penn TreeBank (~1M words, vocabulary size = 10K)
- RNNLM (Mikolov toolbox, 2010) show a significant improvement compared to standard models

# Dealing with large vocabulary

- Hierarchical Softmax:
  - Cluster words in meta-class to speed-up decoding
  - speed up (x10) but slightly worse than full softmax
- Character level RNNs:
  - When the dictionary is very large, character level RNNs may be better
  - No need for text normalization
  - They are very slow because needs to process every character

*Strategies for Training Large Vocabulary Neural Language Models; Chen et al. 2015*

*Generating text with recurrent neural networks; Sutskever et al. 2011*

*Alternative structures for character-level RNNs; Bojanovski et al. 2015*

# More complex RNN

- To capture long term dependency, need for more complex architectures:
  - Longer Short Term Memory (LSTM)
  - Gated Recurrent Network (Cho et al, 2014)
  - Linear Unit Network (Mikolov et al. 2014)
- All of these networks are trained in the same way as standard RNNs

# Long Short Term Memory (LSTM)

- General idea:
  - Each hidden units possess a “**memory cell**” which can be preserve, modify or erase
  - The state of the hidden is simply a consequence of the current state of the memory cell.
  - In practice this means adding “**gating**” mechanisms to allow:
    - Preserving the value of the hidden layers
    - or “forgetting” its previous value

# Gating mechanism

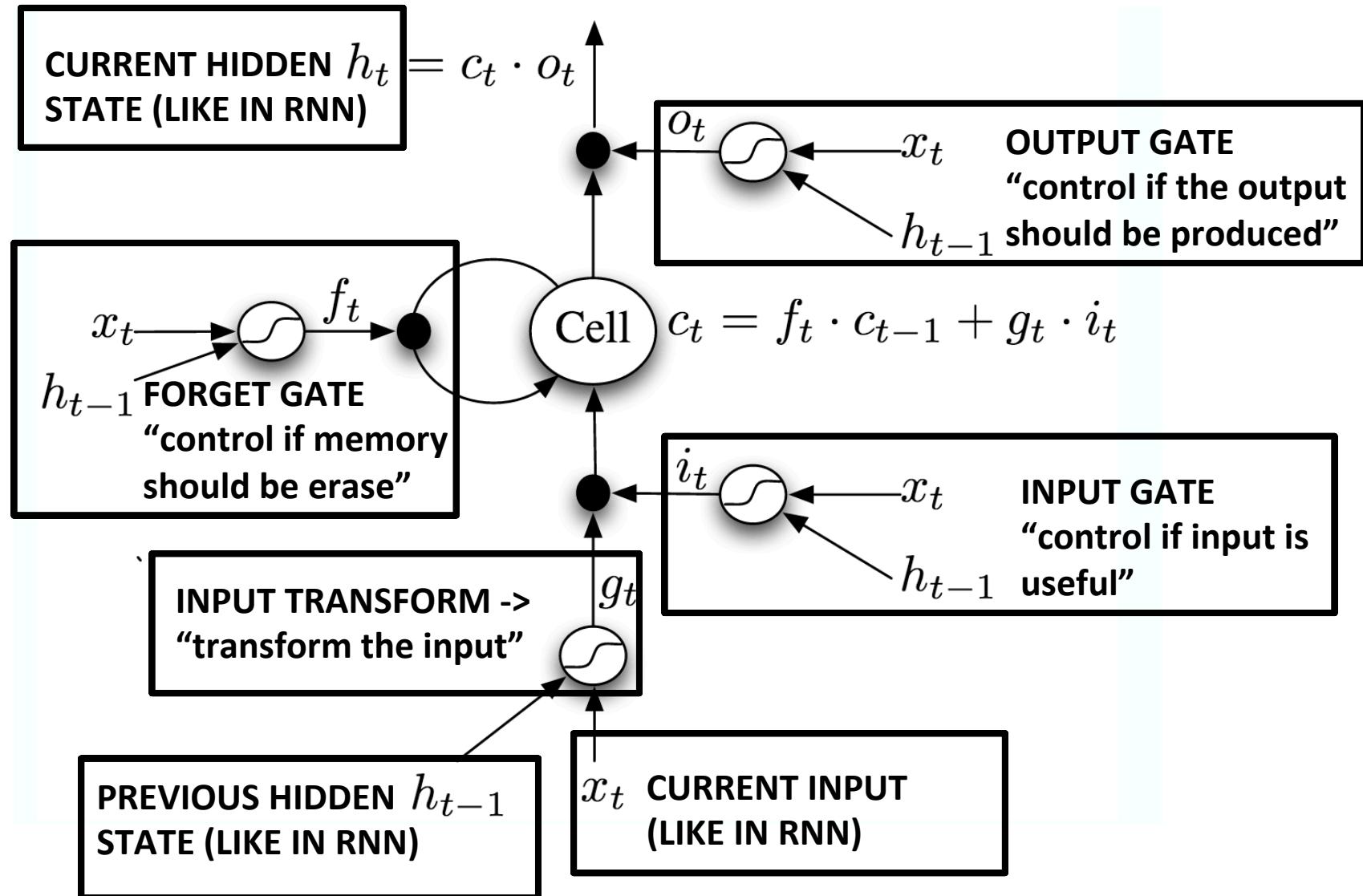
- A gating is a variable going from 0 to 1 which allows or block a signal.
- For example:

$$y = \sigma(x)a + (1 - \sigma(x))b$$

where  $\sigma$  is the sigmoid function (in [0,1]).

→  $\sigma(x)$  is a **gate** controlled by  $x$ , which decides if  $y$  should take the value  $a$  or  $b$ .

# Long Short Term Memory (LSTM)



# LSTM unit in Torch

from **Wojciech Zaremba** github ([wojzaremba/lstm](https://github.com/wojzaremba/lstm)) + nngraph:

```
function lstmcell(input, prevh)
local i2h    = nn.Linear(nhid, 4 * nhid)(input)
local h2h    = nn.Linear(nhid, 4 * nhid)(prevh)
local gates = nn.CAddTable()({i2h, h2h})
gates        = nn.Reshape(4,nhid)(gates)
gates        = nn.SplitTable(2)(gates)

local ingate      = nn.Sigmoid()(nn.SelectTable(1)(gates))
local intransform = nn.Tanh()(  nn.SelectTable(2)(gates))
local forgetgate  = nn.Sigmoid()(nn.SelectTable(3)(gates))
local outgate     = nn.Sigmoid()(nn.SelectTable(4)(gates))

local nextc = nn.CAddTable()({
  nn.CMulTable()({forgetgate, prevc}),
  nn.CMulTable()({ingate,      intransform})
})
local nexth = nn.CMulTable()({outgate, nn.Tanh()(nextc)})
return {nextc, nexth}
end
```

# LSTM for speech recognition

**Table 1.** TIMIT Phoneme Recognition Results. ‘Epochs’ is the number of passes through the training set before convergence. ‘PER’ is the phoneme error rate on the core test set.

NETWORK	WEIGHTS	EPOCHS	PER
CTC-3L-500H-TANH	3.7M	107	37.6%
CTC-1L-250H	0.8M	82	23.9%
CTC-1L-622H	3.8M	87	23.0%
CTC-2L-250H	2.3M	55	21.0%
CTC-3L-421H-UNI	3.8M	115	19.6%
CTC-3L-250H	3.8M	124	18.6%
CTC-5L-250H	6.8M	150	18.4%
TRANS-3L-250H	4.3M	112	18.3%
<b>PRETRANS-3L-250H</b>	<b>4.3M</b>	<b>144</b>	<b>17.7%</b>

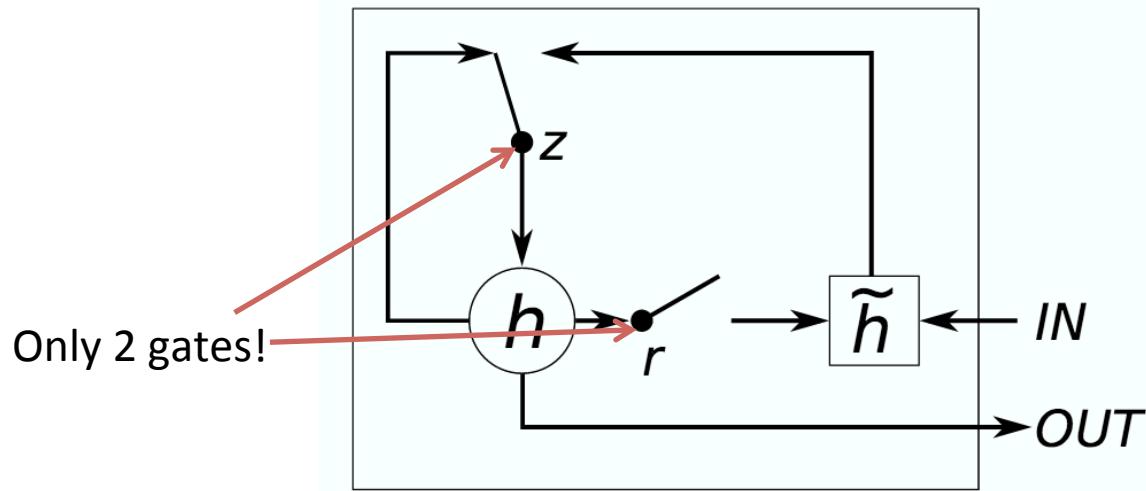
Graves et. al., 2014: *Speech Recognition with Deep Recurrent Neural Networks*

# Gated Recurrent Network (GRU)

- LSTM is very popular and works well in practice.
  - However it seems unnecessarily complicated
- GRU is a simplified version of LSTM which performs as well in practice

*Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling; Chung et al., 2014*

# Gated Recurrent Network



2 gates:

- $r(t)$  : control if the hidden should be reset
- $z(t)$  : control if the hidden should be updated

$$\tilde{h}_t = \tanh(Wx_t + U\text{Diag}(r_t)h_{t-1})$$

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

# GRU units in Torch

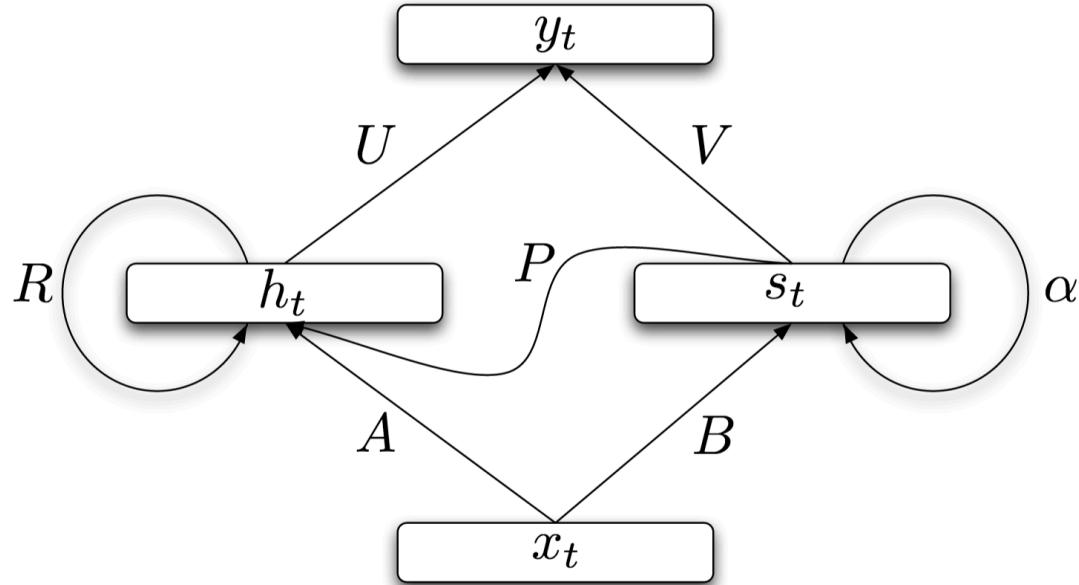
```
function grucell(input, prevh)
local i2h    = nn.Linear(nhid, 3 * nhid)(input)
local h2h    = nn.Linear(nhid, 3 * nhid)(prevh)
local gates = nn.CAddTable()({
    nn.Narrow(2, 1, 2 * nhid)(i2h),
    nn.Narrow(2, 1, 2 * nhid)(h2h),
})
gates = nn.SplitTable(2)(nn.Reshape(2, nhid)(gates))
local resetgate = nn.Sigmoid()(nn.SelectTable(1)(gates))
local updategate = nn.Sigmoid()(nn.SelectTable(2)(gates))
local output = nn.Tanh()(nn.CAddTable()({
    nn.Narrow(2, 2 * nhid+1, nhid)(i2h),
    nn.CMulTable()({resetgate,
        nn.Narrow(2, 2 * nhid+1, nhid)(h2h),})
}))
local nexth = nn.CAddTable()({ prevh,
    nn.CMulTable()({ updategate,
        nn.CSubTable()({output, prevh,}),}),})
return nexth
end
```

# Structurally constrained Network (Mikolov et al, 2014)

- A even simpler model than GRU
- Key idea:
  - Instead of using gating, why not having two types of hidden variable:
    - regular ones (similar to RNN)
    - Linear ones (able to capture long term dependencies)

Mikolov et.al., 2014: *Learning Longer Memory in Recurrent Neural Networks*

# Structurally constrained network



- This is the full model
- No gating:

$$s_t = (1 - \alpha)Bx_t + \alpha s_{t-1},$$

$$h_t = \sigma(Ps_t + Ax_t + Rh_{t-1}),$$

$$y_t = f(Uh_t + Vs_t)$$

# SCRN in torch

```
require 'nngraph'
local a      = 0.9
local nhid  = 100
local nstr  = 40

local input  = nn.Identity()()
local prevh = nn.Identity()()
local prevs = nn.Identity()()

local h2h = nn.Linear(nhid, nhid, false)(prevh)
local s2h = nn.Linear(nstr, nhid, false)(prevs)
local i2h = nn.LookupTable(ninput, nhid)(input)
local i2s = nn.LookupTable(ninput, nstr)(input)
local h = nn.Sigmoid()(nn.CAddTable()({i2h, h2h, s2h}))
local s = nn.CAddTable(){nn.Mul(1-a)(i2s), nn.Mul(a)(prevs)}
local y = nn.Linear(nhid + nstr, nvoc, false)(nn.ConcatTable{h,s})

scrn = nn.gModule({prevh, prevs, input}, {h, s, y})
```

# SCRN in practice

MODEL	# hidden units	Perplexity
N-gram	-	141
N-gram + cache	-	125
SRN	100	129
LSTM	100 (x4 parameters)	115
SCRN	100 + 40	115

- Performs as well as LSTM on Penn TreeBank
- Perform a bit worse on text8 (~ 5 ppl worse)  
→ Suggests that most of the long term dependency captured by LSTM is same as with **RNN + bag of words.**

# Extensions to other domains

- So far, we mostly talked about language modeling
- How to apply them to other domains?
  - Machine translation
  - Image captioning
  - Frame prediction in videos

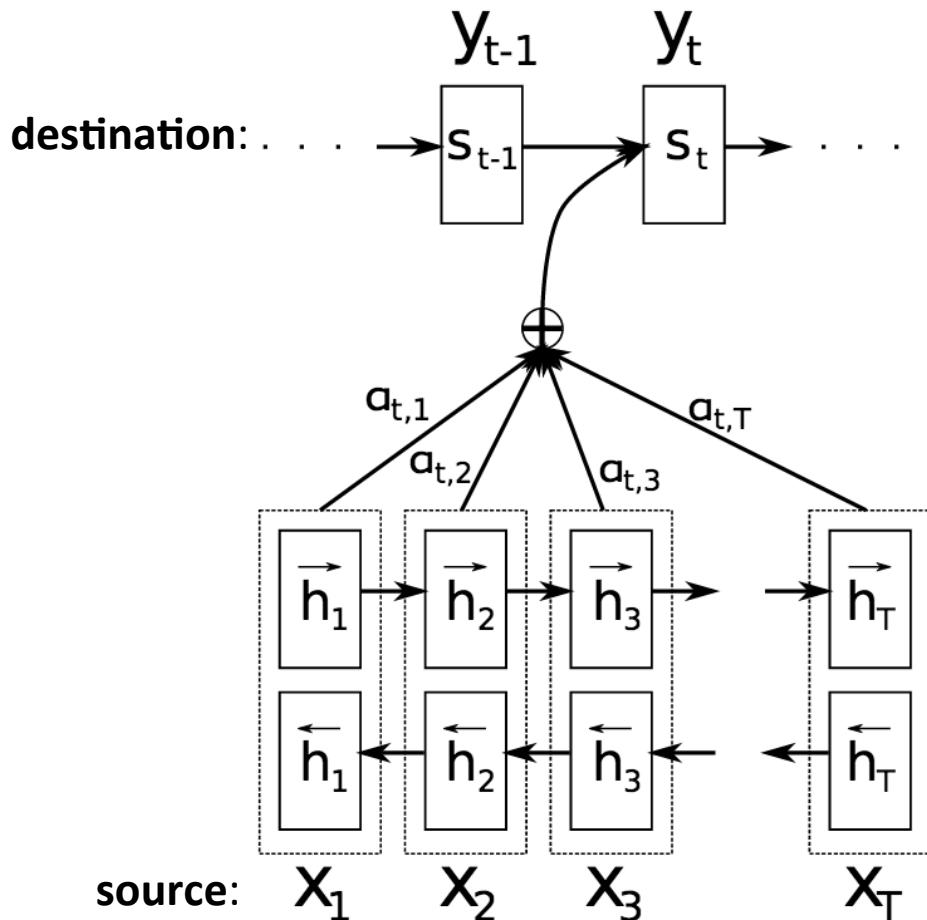
# Machine Translation

- Learn from example how to translate sentences between a *source* language and a *destination* language
- The two sentences may have different length
  - Cannot apply RNN directly
- 2 solutions:
  - Condition prediction of *each word* in destination sentence by the source sentence (Bahdanau et al. 2014)
  - Condition prediction of *the whole* destination sentence by the source sentence (Sutskever et al. 2014)

# Word level conditioning

- Simple idea:
  - Use the RNN on the *destination* language
  - Condition each word by the *source* sentence
  - Example: Bag of word
  - Limitation of BoW: every words in the source sentence has the same weight → Learn the weights

# Word level conditioning



Each prediction  $y(t)$  is conditioned by a vector  $c(t)$  such that:

$$c_t = \sum_k \alpha_{tk} h_k$$

$c(t)$  = weighted sum of the representation of source words.

If weights are equal  $\rightarrow$  BoW.

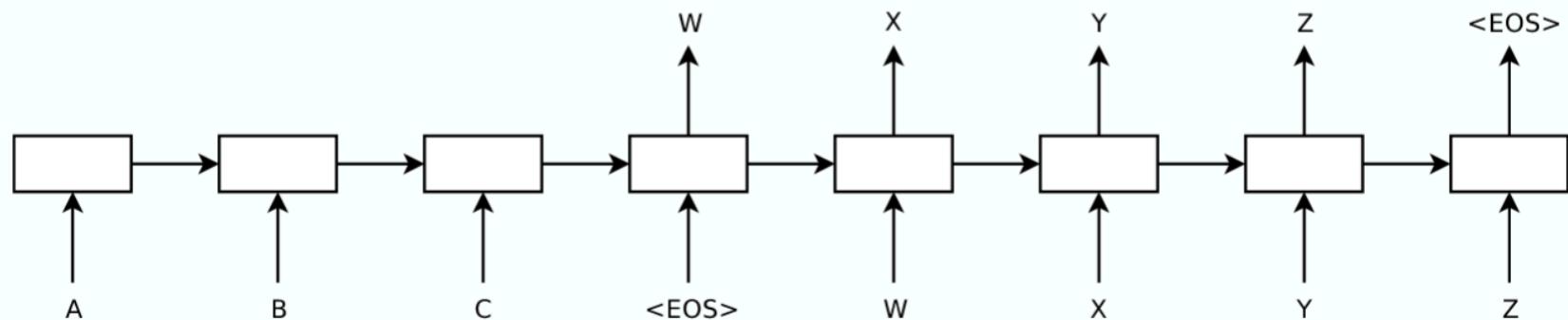
**Attention mechanism:** Make the weights depends on the **relation** between source and destination representations:

$$\alpha_{tk} = \exp(e_{tk}) / \sum_i \exp(e_{ti})$$

$$e_{tk} = f(s_t, h_k)$$

# Sentence level conditioning

- Simple idea:
  - process the source sentence with an RNN
  - use the states of the hidden layers to condition a RNN on the destination sentence
  - Drawback: requires massive RNN to remember information about source sentence



# Machine Translation

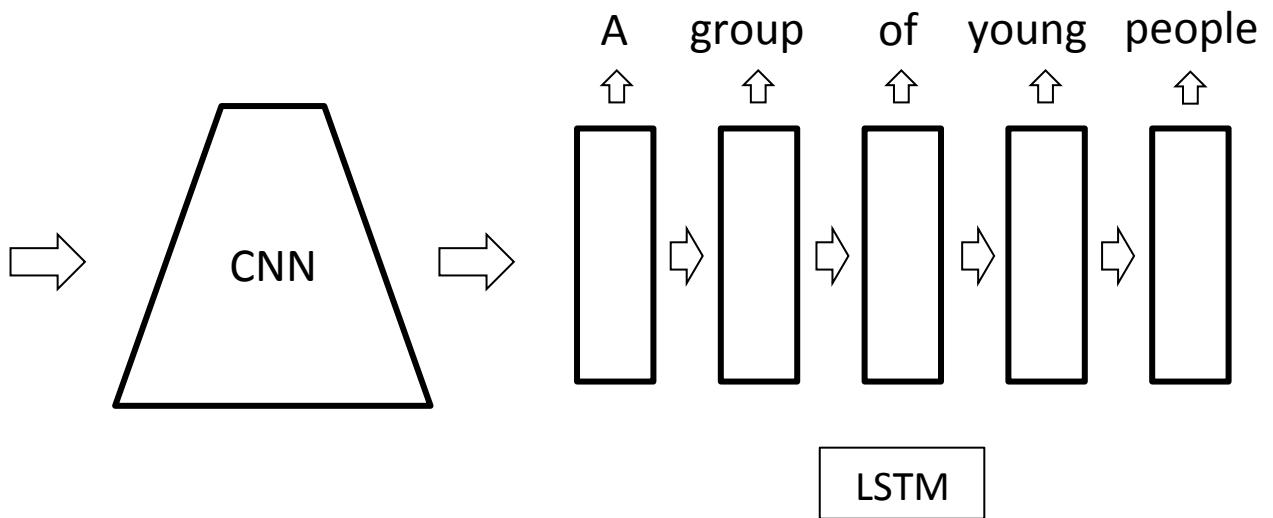
Method	test BLEU score (ntst14)
Bahdanau et al. [2]	28.45
Baseline System [29]	33.30
Single forward LSTM, beam size 12	26.17
Single reversed LSTM, beam size 12	30.59
Ensemble of 5 reversed LSTMs, beam size 1	33.00
Ensemble of 2 reversed LSTMs, beam size 12	33.27
Ensemble of 5 reversed LSTMs, beam size 2	34.50
Ensemble of 5 reversed LSTMs, beam size 12	<b>34.81</b>

**State-of-the-art WMT'14 result: 37.0**

# Image Captioning

- Same idea
- Condition the RNN on an image feature (CNN):
  - either at each word prediction
  - or at the beginning of the sentence
- Lot of papers:
  - Mao et al. 2014,
  - Karpathy et al. 2014,
  - Xu et al. 2015...
- with all possible variants of conditioning

# Example of captioning model



# Image captioning: examples



A square with burning street lamps and a street in the foreground;



Tourists are sitting at a long table with a white table cloth and are eating;



A dry landscape with green trees and bushes and light brown grass in the foreground and reddish-brown round rock domes and a blue sky in the background;



A blue sky in the background;

# Image captioning?

- Issue: The datasets are quite small (flickr8k, flickr30K, COCO80K)
- Larry Zitnick, Organizer of the caption challenge:  
*“35% - 85% of captions are identical to training captions”*
- Is “captioning” simply sentence retrieval?

# Nearest neighbor for captioning

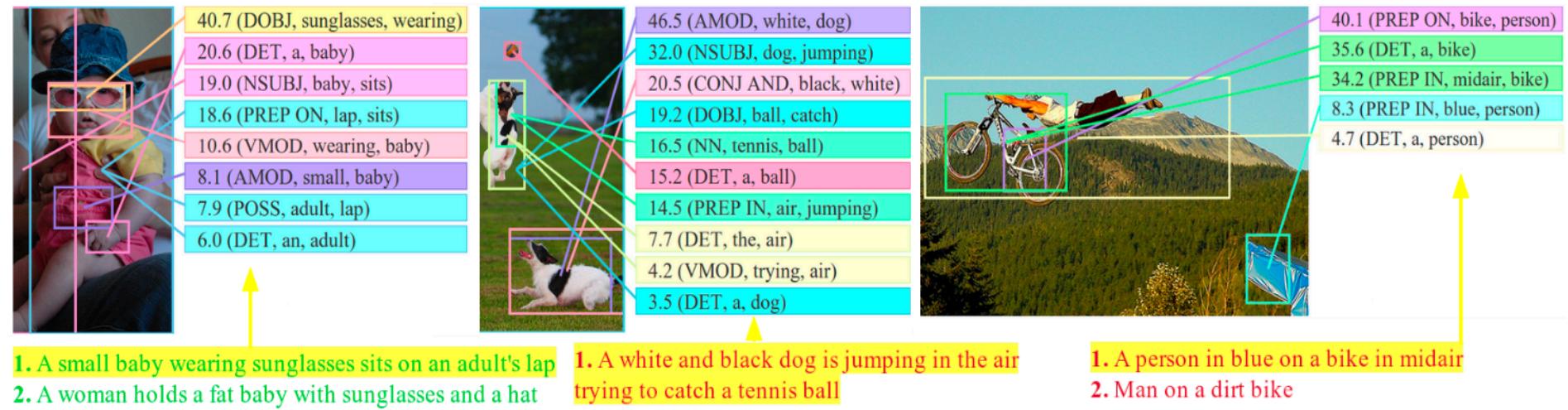


A black and white cat sitting in a bathroom sink.



Two zebras and a giraffe in a field.

# Retrieval with alignment

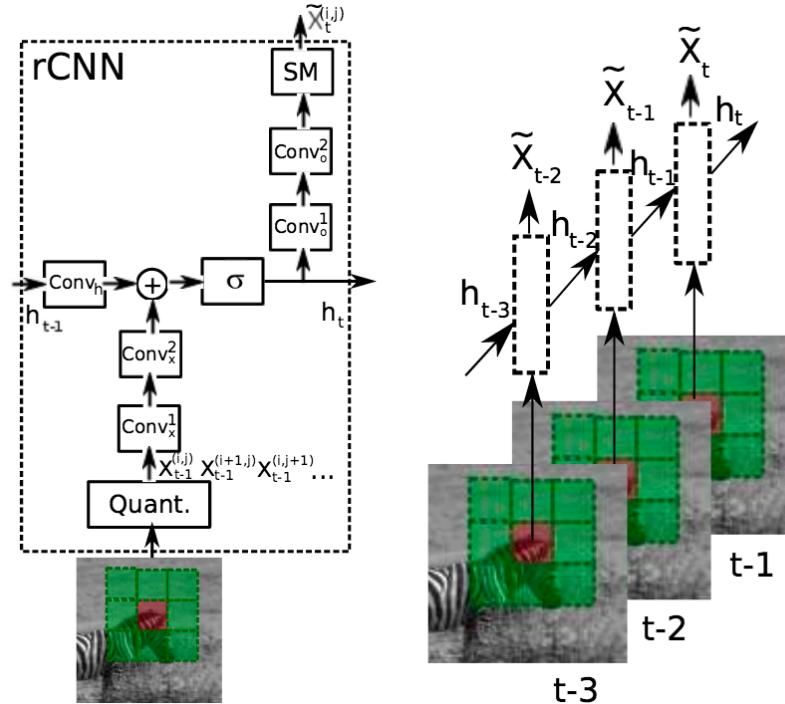


# Frame prediction in videos

- RNN can also be used in the continuous domain
- Example:
  - Frame prediction in videos
  - Allows unsupervised learning of visual features!
  - Consider the video as a sequence and try to predict the next frames (Ranzato et al. 2014, Matthieu et al., 2015)

# Frame prediction in videos

- Extract a CNN feature from a patch in a frame
- Run a RNN on top to predict how it will change



# Summary: RNN

- RNNs are simple sequence prediction models
- They can be trained efficiently on large corpus of data (with GPUs)
- They are state-of-the-art language models
- Successfully applied to speech, machine translation and computer vision

# Limitations

- RNNs only works on tasks where other ML methods would also work a bit. They just work better
- They are mostly compressing the data they have seen during training → No real language understanding

# Promising future research directions

- Evaluating the capacity of these model to perform simple reasoning tasks
- Can we build dialogue agents based on these networks?
- Can we learning algorithms from examples?

# Promising future research directions

- Simple environment to test the capacity of these models:
  - Persistent environment based on language:
    - *A roadmap towards machine intelligence*, Mikolov et al. 2015
    - *MazeBase: A Sandbox for Learning from Games*, Sukhbaatar, 2015
  - Set of simple tasks to test the limit of RNNs:
    - *Towards AI-Complete Question Answering: A Set of Prerequisite Toy Tasks*, Weston et al, 2015

# Promising future research directions

- Learning algorithms form example:
  - *Neural Turing Machine*, Graves et al. 2015
  - *Inferring algorithmic patterns with stack-augmented recurrent nets*, Joulin and Mikolov 2015
  - *Learning Simple Algorithms from Examples*, Zaremba et al. 2015
  - *End-To-End Memory Networks*, Sukhbaatar et al. 2015

Thank you