Session 4

Recurrent Neural Network

Dr. Hsiu-Wen (Kelly) Chang Joly, Center for Robotics, Mines Paristech, PSL

1

Acknowledgements

- The materials majorly derived from Prof. Fabien Moutarde. Some slides come from the online classes.
  - **Fei-Fei Li + J.Johnson + S.Yeung: slides on "*Recurrent Neural Networks*" from the** "*Convolutional Neural Networks for Visual Recognition*" **course at Stanford**
    http://cs231n.stanford.edu/slides/2019/cs231n_2019_lecture10.pdf
  - **Yingyu Liang: slides on "*Recurrent Neural Networks*" from the** "*Deep Learning Basics*" **course at Princeton**
    https://www.cs.princeton.edu/courses/archive/spring16/cos495/slides/DL_lecture9_RNN.pdf
  - **Arun Mallya: slides "*Introduction to RNNs*" from the** "Trends in Deep Learning and Recognition" **course of Svetlana LAZEBNIK at University of Illinois at Urbana-Champaign**
    http://slazebni.cs.illinois.edu/spring17/lec02_rnn.pdf
  - **Tingwu Wang: slides on "*Recurrent Neural Network*" for a course at University of Toronto**
    https://www.cs.toronto.edu/%7Etingwuwang/rnn_tutorial.pdf
  - **Christopher Olah: online tutorial "*Understanding LSTM Networks*"**
    https://colah.github.io/posts/2015-08-Understanding-LSTMs/

2

Introduction

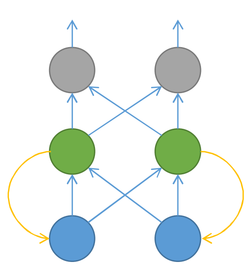
- Recurrent neural networks have been an important focus of research and development during the 1990's. ➔ It is much older than ConvNet!
- They are designed to learn sequential or time varying patterns.
- A recurrent net is a neural network with feedback (closed loop) connections [Fausett, 1994]. Examples include BAM, Hopfield, Boltzmann machine, and recurrent backpropagation nets [Hecht-Nielsen, 1990].
- A dynamic neural network can be defined as a neural networks that consists of inter-layer feedback loops (i.e., from output layer to input layer) and intra-layer feedback loops (i.e., between different neurons within the same layer) or self-feedback loops.
- From the computational perspective, a dynamic neural network that contains the feedback loop that may provide more computational advantages than a static neural network, which contains only feed-forward architecture
- Applications: natural language processing (NLP), forecasting, signal processing and control require the treatment of dynamics associated with the unknown model.

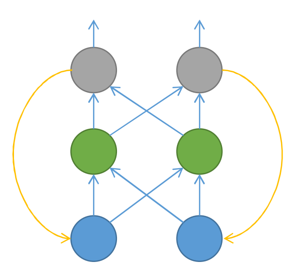
3

3

Old style of RNN

- Elman introduced feedback from the hidden layer to the context portion of the input layer.
  - This approach pays more attention to the sequence of input values.
- Jordan recurrent neural networks use feedback from the output layer to the context nodes of the input layer and give more emphasis to the sequence of output values.





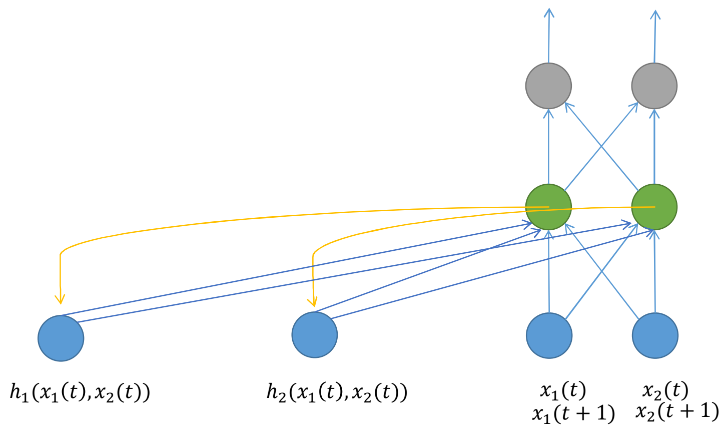
However, these methods did not succeed in bigger data set due to the design of gradient flow

4

4

Recurrent neural networks

- Flow in animation



5

5

Flexibility

- In some context of machine learning, we want to have flexibility of input and output

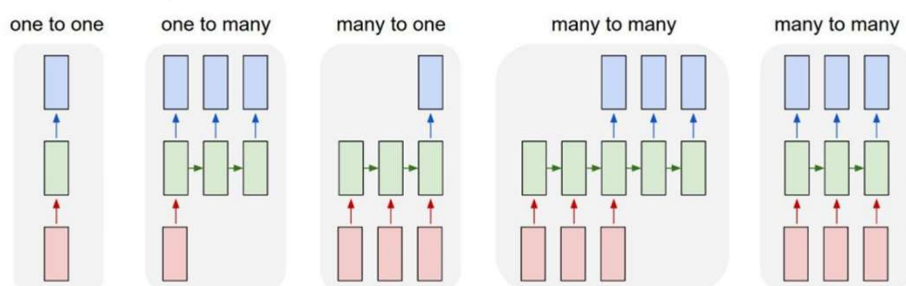


Image captioning
Image->seq of words

Sentiment classification
seq of words -> sentiment

Language translation
Seq of words -> seq of words

Video classification on frame level

Dr. Hsiu-Wen (Kelly) Chang Joly, Center for Robotics, Mines Paristech, PSL

6

Gradient flow

- Gradient flow is very important in network
- We already saw a lots in the last section
- Risk to have feed-back connection:
  - stability
  - Controllability
  - Observability

Advantages of RNN

- The hidden state s of the RNN builds a kind of lossy summary of the past
- RNN totally adapted to processing SEQUENTIAL data (same computation formula applied at each time step, but modulated by the evolving "memory" contained in state s)
- Universality of RNNs: any function computable by a Turing Machine can be computed by a finite-size RNN (Siegelmann and Sontag, 1995)

Simply RNN: Vanilla

State vector s ← → vector h of hidden neurons

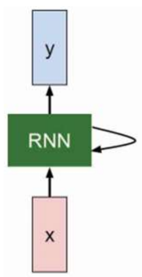
New hidden state

Input vector at time t

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

Old hidden state

Some function with parameters W



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

$$y_t = W_{hy}h_t$$

Or $y_t = softmax(W_{hy}h_t)$

Dr. Hsiu-Wen (Kelly) Chang Joly, Center for Robotics, Mines Paristech, PSL

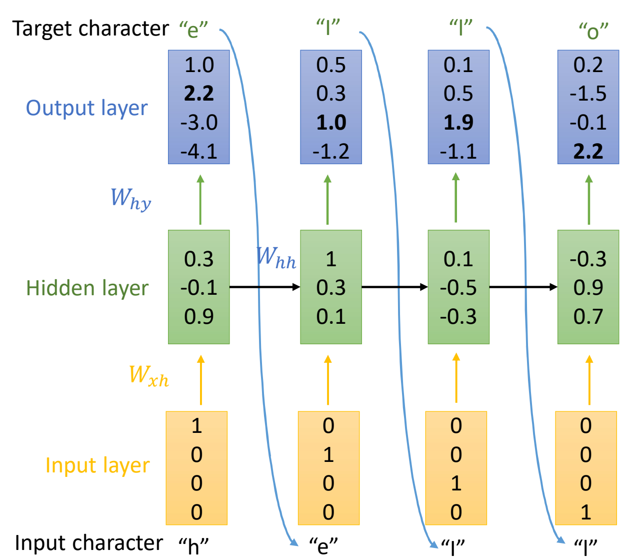
9

Character-level Language Model

- Given four letters [h,e,l,o] as input vector
- First we transfer this into one-hot vector
- Then we randomize weights W_{xh}, W_{hy}
- Then we can train a sequence to predict "hello"

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

Be aware that weights are shared in the sequence processing.

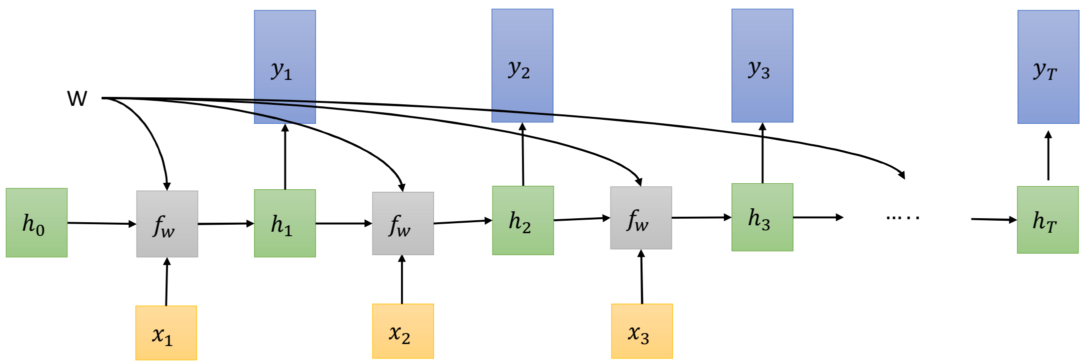


Dr. Hsiu-Wen (Kelly) Chang Joly, Center for Robotics, Mines Paristech, PSL

10

RNN: computation graph M2M

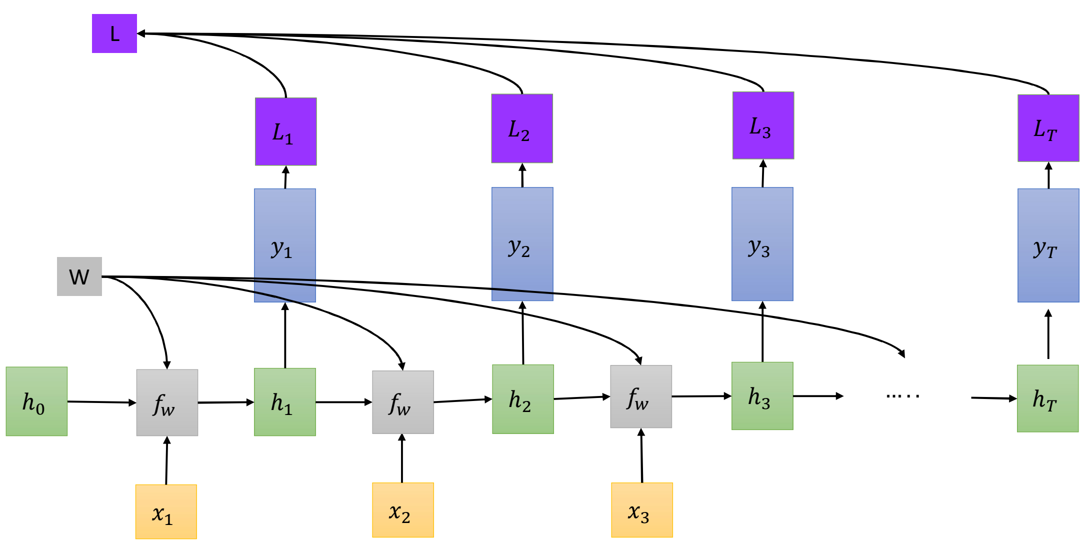
- RNN forward pass: Many to many: (language translation)



Dr. Hsiu-Wen (Kelly) Chang Joly, Center for Robotics, Mines Paristech, PSL

11

RNN: computation graph M2M training

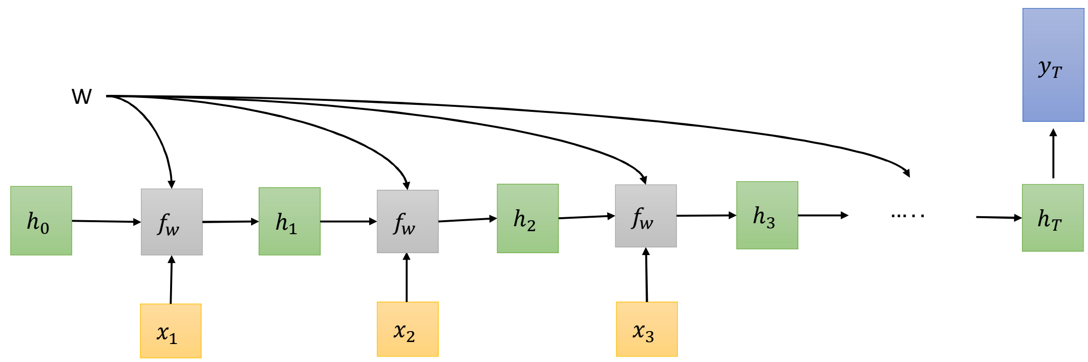


Dr. Hsiu-Wen (Kelly) Chang Joly, Center for Robotics, Mines Paristech, PSL

12

RNN: computation graph M2O

- RNN forward pass: Many to one: (semantic judge)

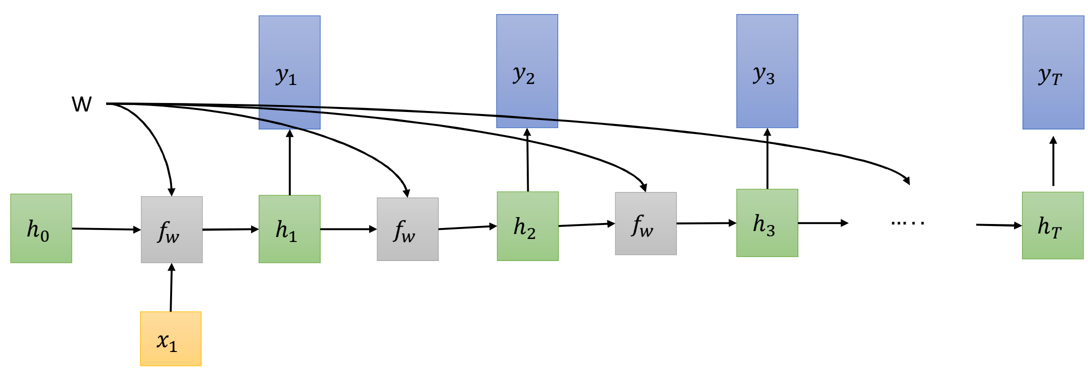


Dr. Hsiu-Wen (Kelly) Chang Joly, Center for Robotics, Mines Paristech, PSL

13

RNN: computation graph O2M

- RNN forward pass: One to many: (captioning)



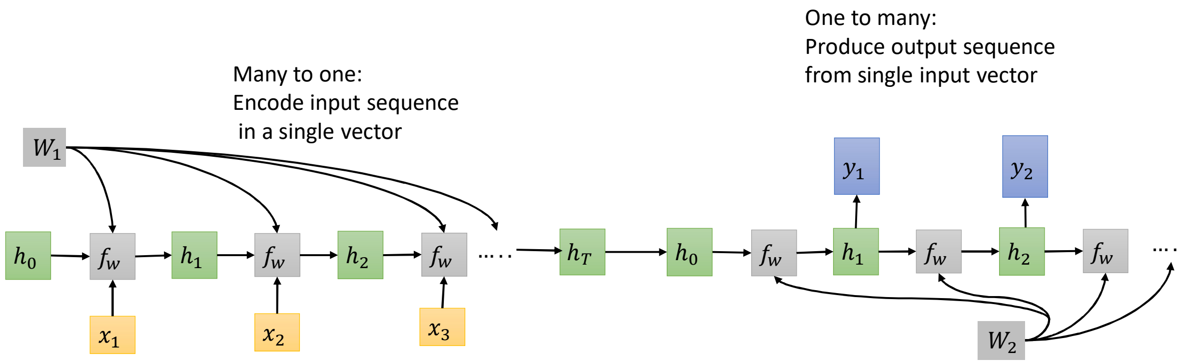
Dr. Hsiu-Wen (Kelly) Chang Joly, Center for Robotics, Mines Paristech, PSL

14

MINES ParisTech | PSL

Sequence to Sequence: O2M+M2O

- O2M+M2O



MINES ParisTech | PSL

Backpropagation through time

- Applying backpropagation in RNNs is called ***backpropagation through time*** [Werbos.1990].
- This procedure requires us to expand (or unroll) the computational graph of an RNN one time step at a time.
- The unrolled RNN is essentially a feedforward neural network with the spec ial property that the same parameters are repeated throughout the unrolle d network, appearing at each time step.
- Then, like in feedforward neural network, we apply the chain rule to backpropagate gradients through the unrolled net.
- The gradient with respect to each parameter must be summed across all places that the parameter occurs in the unrolle d net.
- Handling such weight tying should be familiar from our chapters on convol utional neural networks.

Werbos, P. J. (1990). Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, *78*(10), 1550–1560.

Analysis of gradients

$$h_t = f(x_t, h_{t-1}, w_h)$$
$$o_t = g(h_t, w_o)$$

- where **f** and **g** are transformations of the hidden layer and the output layer, respectively.
- Hence, we have a chain of values that depend on each other via recurrent computation

$$\{\dots, (x_{t-1}, h_{t-1}, o_{t-1}), (x_t, h_t, o_t), \dots\}$$

The forward pass of this model is to loop through the (x_t, h_t, o_t) triples one time step at a time. The discrepancy between output o_t and the desired target y_t is then evaluated by an objective function across all the T time steps

$$L(x_1, \dots, x_T, y_1, \dots, y_T, w_h, w_o) = \frac{1}{T}\sum_{t=1}^{T} l(y_t, o_t)$$

Dr. Hsiu-Wen (Kelly) Chang Joly, Center for Robotics, Mines Paristech, PSL

17

Analysis of gradients

- Here are the tricky way to derive the gradients regarding the parameters w_h of the objective function L.

$$\frac{\partial L}{\partial w_h} = \frac{1}{T}\sum_{t=1}^{T} \frac{\partial l(y_t, o_t)}{\partial w_h}$$
$$= \frac{1}{T}\sum_{t-1}^{T} \frac{\partial l(y_t, o_t)}{\partial o_t}\frac{\partial (o_t)}{\partial w_h} = \frac{1}{T}\sum_{t=1}^{T} \frac{\partial l(y_t, o_t)}{\partial o_t}\frac{\partial g(h_t, w_o)}{\partial h_t}\boxed{\frac{\partial h_t}{\partial w_h}}$$

- The third term in this equation is tricky because the computation of h_t depends on both h_{t-1}, w_h where computation of h_{t-1} also depends on w_h. Applying total differential of $df(x, y) = f_x dx + f_y dy$, we have:

$$\frac{\partial h_t}{\partial w_h} = \frac{\partial f(x_t, h_{t-1}, w_h)}{\partial w_h} + \frac{\partial f(x_t, h_{t-1}, w_h)}{\partial h_{t-1}}\frac{\partial h_{t-1}}{\partial w_h}$$

Dr. Hsiu-Wen (Kelly) Chang Joly, Center for Robotics, Mines Paristech, PSL

18

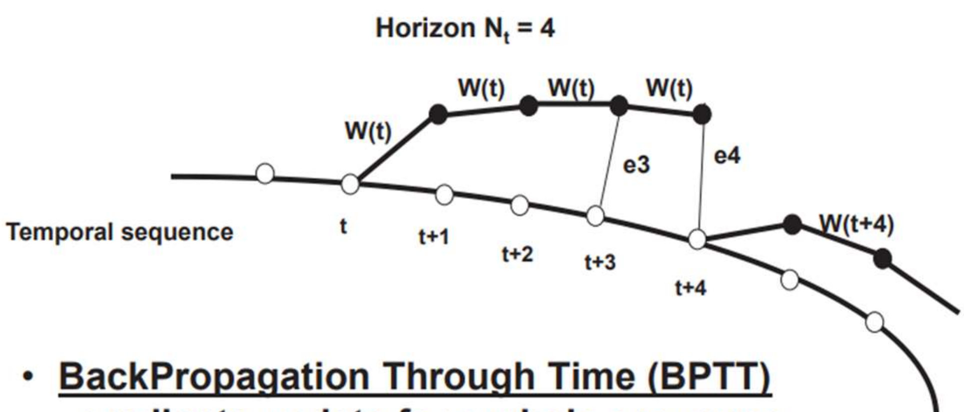
Analysis of gradients

$$\frac{\partial h_t}{\partial w_h} = \frac{\partial f(x_t, h_{t-1}, w_h)}{\partial w_h} + \sum_{i=1}^{t-1}\left(\prod_{j=i+1}^{t} \frac{\partial f(x_j, h_{j-1}, w_h)}{\partial h_{j-1}}\right)\frac{\partial f(x_i, h_{i-1}, w_h)}{\partial w_h}$$

- While we can use the chain rule to compute $\frac{\partial h_t}{\partial w_h}$ recursively, this chain can get very long whenever t is large.
  - Full computation: very slow and gradients can blow up, since subtle changes in the initial conditions can potentially affect the outcome a lot
  - Truncating time steps [Jaeger, 2002]
  - Randomized Truncation [Tallec and Ollivier, 2017]

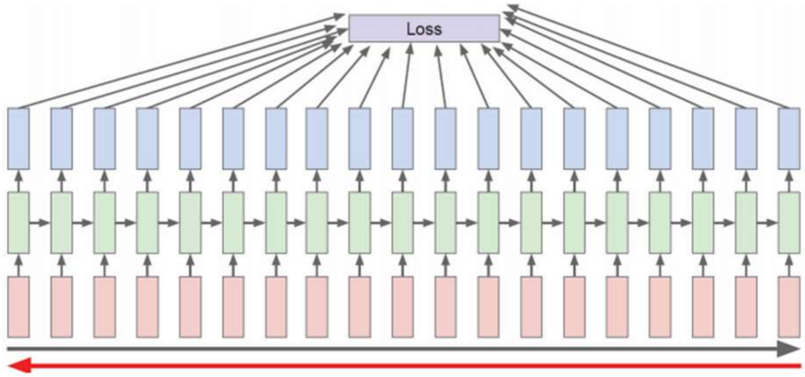
Jaeger, H. (2002). *Tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the" echo state network" approach*. Vol. 5. GMD-Forschungszentrum Informationstechnik Bonn
Tallec, C., & Ollivier, Y. (2017). Unbiasing truncated backpropagation through time. *arXiv preprint arXiv:1705.08209*.

RNN training



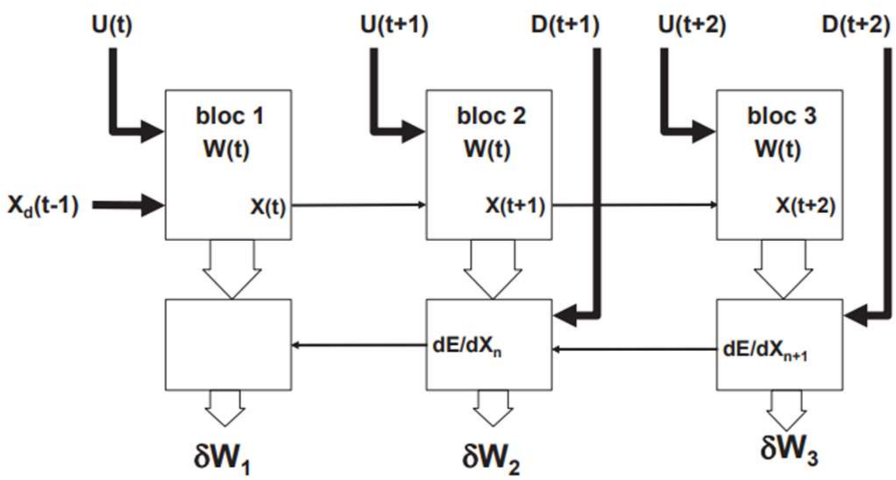
- <u>BackPropagation Through Time (BPTT)</u>
  gradients update for a whole sequence
- **or Real Time Recurrent Learning (RTRL)**
  gradients update for each frame in a sequence

BackPropagation THROUGH TIME (BPTT)



- **Forward through entire sequence to compute SUM of losses at ALL (or part of) time steps**
- **Then backprop through ENTIRE sequence to compute gradients**

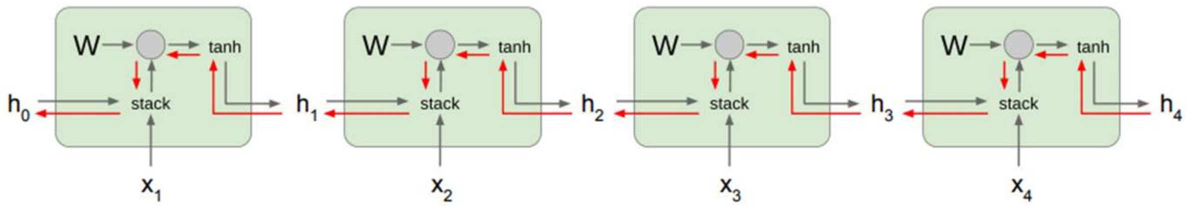
BPTT computation principle



$$\delta W = \delta W_1 + \delta W_2 + \delta W_3$$

Vanilla RNN Gradient Flow

Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013



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

Largest singular value > 1:
Exploding gradients

Largest singular value < 1:
Vanishing gradients

Gradient clipping: Scale gradient if its norm is too big

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

Change architecture!

23

Vanishing/exploding gradient problem

- If eigenvalues of Jacobian matrix >1, then gradients tend to EXPLODE
  ➔Learning will never converge.
- Conversely, if eigenvalues of Jacobian matrix < 1, then gradients tend to VANISH
  ➔Error signals can only affect small time lags
  ➔short-term memory.
- Possible solutions for exploding gradient: CLIPPING trick (limited values in an array, see numpy.clip), truncated.
- Possible solutions for vanishing gradient:
  - use ReLU instead of tanh
  - change what is inside the RNN!

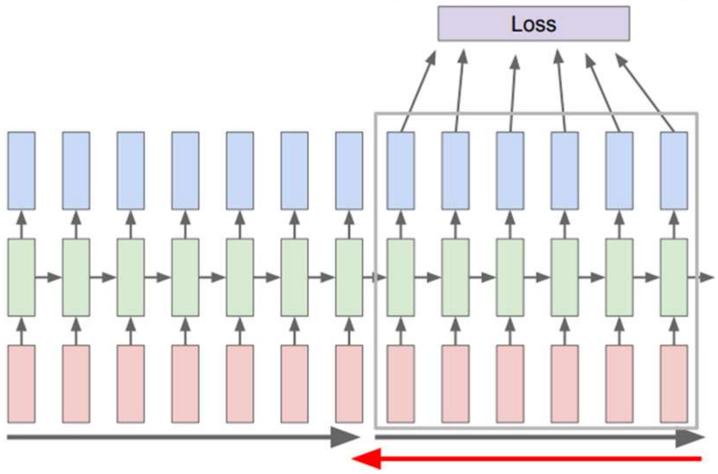
Recommended code to read for better understand this slide: https://gist.github.com/karpathy/d4dee566867f8291f086

Dr. Hsiu-Wen (Kelly) Chang Joly, Center for Robotics, Mines Paristech, PSL

24

Truncated tricks

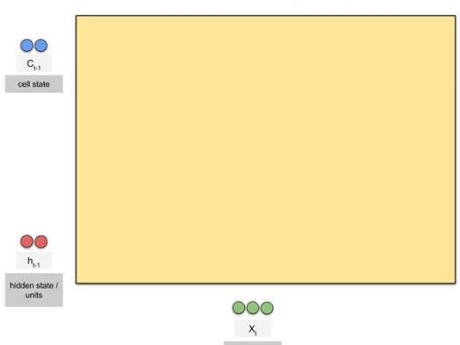
Truncated Backpropagation through time

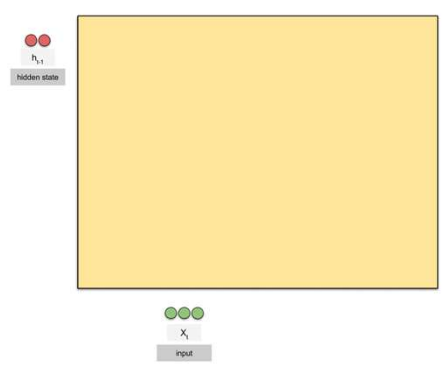


Carry hidden states forward in time forever, but only backpropagate for some smaller number of steps

Modern ways in RNN

- To solve the gradient issue, a better design of the architecture is important.
- Here are two successful networks that improve the performance by the way it design the gradient flow
  - Long Short-Term Memory (LSTM)
  - Gated Recurrent Unit (GRU)





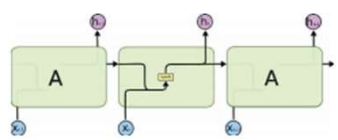
[Created by Raimi Karim, https://towardsdatascience.com/animated-rnn-lstm-and-gru-ef124d06cf45]

Long short-term memory

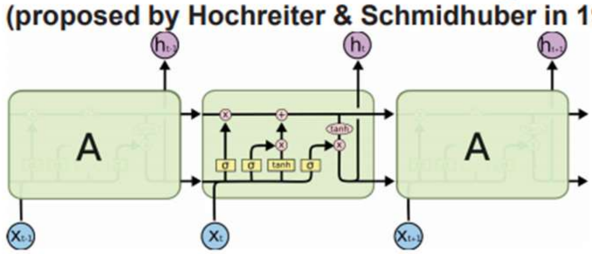
- The term "long short-term memory" comes from the following intuition. Simple recurrent neural networks have *long-term memory* in the form of weights. The weights change slowly during training, encoding general knowledge about the data. They also have *short-term memory* in the form of ephemeral activations, which pass from each node to successive nodes. The LSTM model introduces an intermediate type of storage via the memory cell. A memory cell is a composite unit, built from simpler nodes in a specific connectivity pattern, with the novel inclusion of multiplicative nodes
- Gated memory cell is equipped with an internal state and a number of multiplicative gates that determine
  - a given input should impact the internal state (the *input gate*): $i \in [0,1]$
  - Weather the internal state should be flushed to 0 (the *forget gate*): $f \in [0,1]$
  - Weather the internal state of a given neuron should be allowed to impact the cell's output (the *output* gate): $o \in [0,1]$

LSTM

Problem of *standard* RNNs = no actual LONG-TERM memory



LSTM = RNN variant for solving this issue
(proposed by Hochreiter & Schmidhuber in 1997)



[Figures from https://colah.github.io/posts/2015-08-Understanding-LSTMs/]

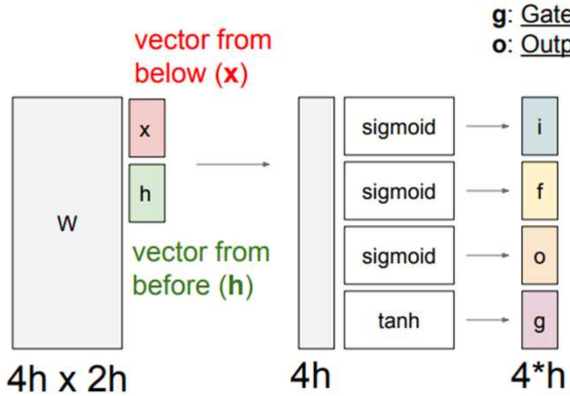
- **Key idea = use "gates" that modulate respective influences of input and memory**

LSTM

Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]

f: Forget gate, Whether to erase cell
i: Input gate, whether to write to cell
g: Gate gate (?), How much to write to cell
o: Output gate, How much to reveal cell



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

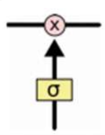
$$c_t = f \odot c_{t-1} + i \odot g$$

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

Dr. Hsiu-Wen (Kelly) Chang Joly, Center for Robotics, Mines Paristech, PSL

29

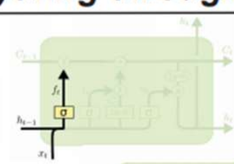
LSTM gates



Gate = *pointwise* multiplication by σ in]0;1[
➔ modulate between "*let nothing through*" and "*let everything through*"

- **FORGET gate**

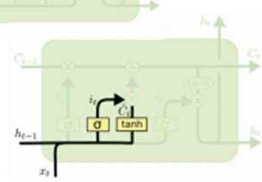
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$



- **INPUT gate**

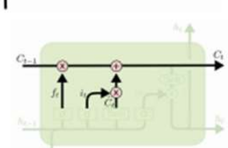
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$

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



➔ next state = mix between pure memory or pure new

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



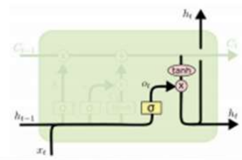
[Figures from https://colah.github.io/posts/2015-08-Understanding-LSTMs/]

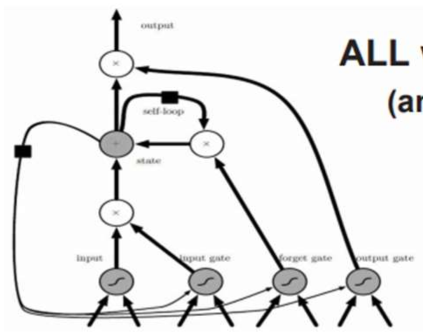
30

LSTM summary

- **OUTPUT gate**

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





ALL weigths $\mathtt{W_f}$, $\mathtt{W_i}$, $\mathtt{W_c}$ and $\mathtt{W_o}$ (and biases) are LEARNT

[Figure from Deep Learning book by I. Goodfellow, Y. Bengio & A. Courville]

31

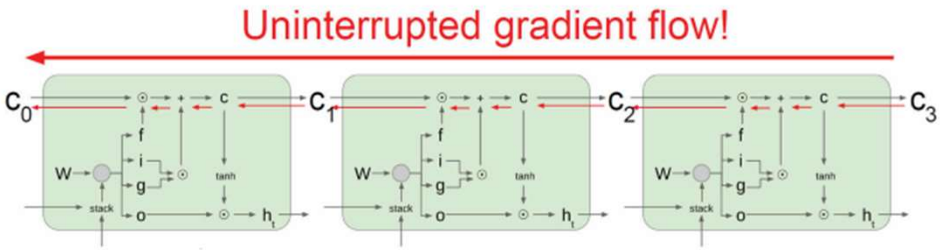
Practice

- Given input $x_t \in R^4$ and the target y_t at time step t. The hidden state $h_t \in R^2$ and the output $o_t \in R^1$.
  - What is the size of the weight when vanilla RNN is applied?
  - What is the size of the weights when LSTM is applied?

Dr. Hsiu-Wen (Kelly) Chang Joly, Center for Robotics, Mines Paristech, PSL

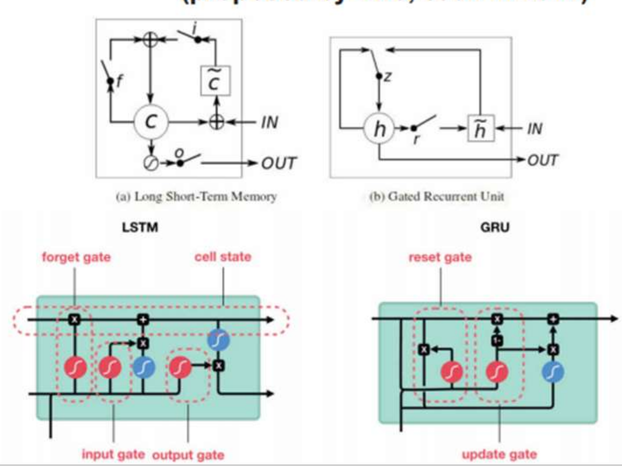
32

Why LSTM avoids vanishing gradients?



Gated Recurrent Unit (GRU)

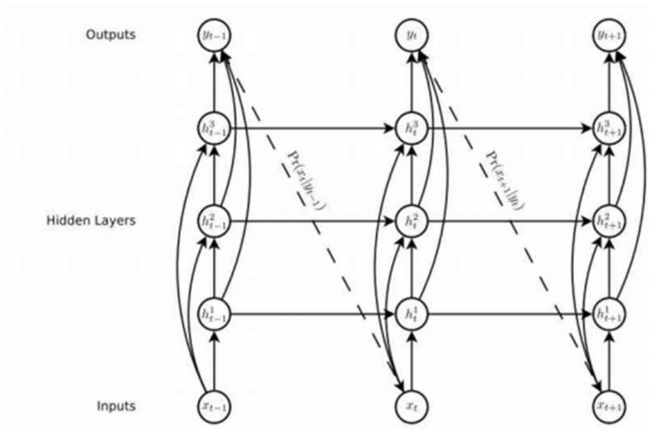
Simplified variant of LSTM, with only 2 gates:
a RESET gate & an UPDATE gate
(proposed by Cho, et al. in 2014)



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

$$r_t = \sigma(W_{xr}x_t + W_{hr}h_{t-1} + b_r)$$
$$z_t = \sigma(W_{xz}x_t + W_{hz}h_{t-1} + b_z)$$
$$\tilde{h}_t = \tanh(W_{xh}x_t + W_{hh}(r_t \odot h_{t-1}) + b_h)$$
$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot \tilde{h}_t$$

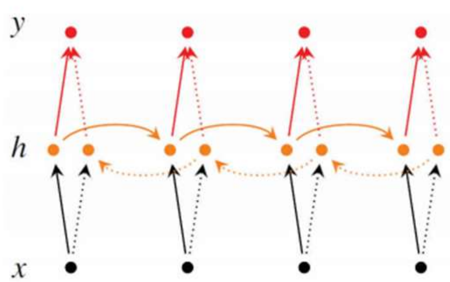
Deeper RNN



Several RNNs stacked (like layers in MLP)

35

Bi-directional RNNs



$$\overrightarrow{h}_t = f(\overrightarrow{W}x_t + \overrightarrow{V}\overrightarrow{h}_{t-1} + \overrightarrow{b})$$

$$\overleftarrow{h}_t = f(\overleftarrow{W}x_t + \overleftarrow{V}\overleftarrow{h}_{t+1} + \overleftarrow{b})$$

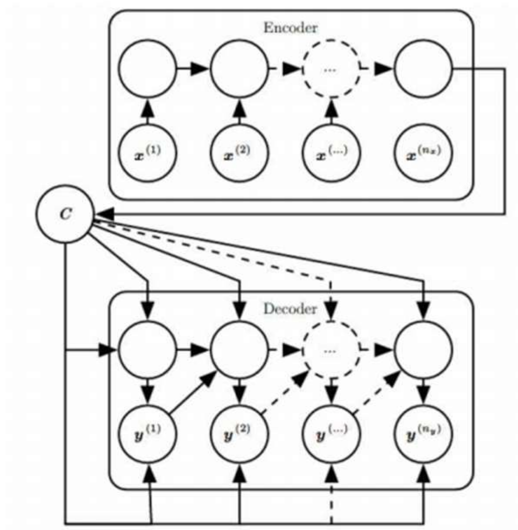
$$y_t = g(U[\overrightarrow{h}_t;\overleftarrow{h}_t] + c)$$

$h = [\overrightarrow{h};\overleftarrow{h}]$ now represents (summarizes) the past and future around a single token.

(e.g. for offline classification of sequence of words)

36

Encoder-decoder RNN



We will play it in practical lesson

37

Recommended reading: RNN variants

- [LSTM: A Search Space Odyssey, Greff et al., 2015] :
  - they play around the LSTM equations, swap out the non linearities at one point, do we need tanh, this paper made a lot of experiences in playing around different design
  - Conclusion is there is no significant difference.
- [An Empirical Exploration of Recurrent Network Architectures, Jozefowicz et al., 2015]:
  - Search over very large number of random RNN architecture, randomly permute these equations to see if there is a better one
  - Conclusion: No significant improvement with one specific version

38

Applications

Image Captioning

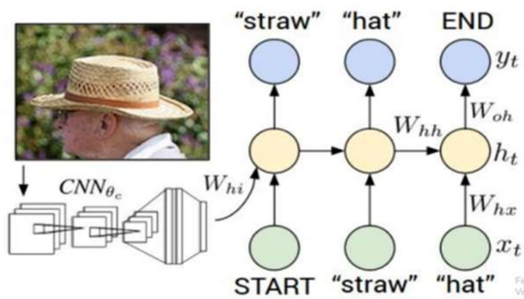


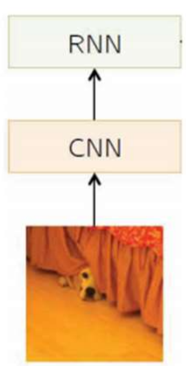
Figure from Karpathy et a, "Deep Visual-Semantic Alignments for Generating Image Descriptions", CVPR 2015; figure copyright IEEE, 2015. Reproduced for educational purposes.

Explain Images with Multimodal Recurrent Neural Networks, Mao et al.
Deep Visual-Semantic Alignments for Generating Image Descriptions, Karpathy and Fei-Fei
Show and Tell: A Neural Image Caption Generator, Vinyals et al.
Long-term Recurrent Convolutional Networks for Visual Recognition and Description, Donahue et al.
Learning a Recurrent Visual Representation for Image Caption Generation, Chen and Zitnick

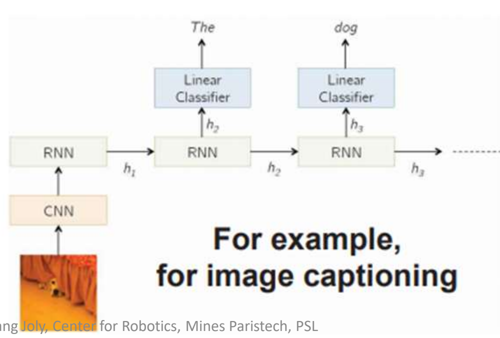
Dr. Hsiu-Wen (Kelly) Chang Joly, Center for Robotics, Mines Paristech, PSL

39

Application mixed vision and text



Input into RNN the features from last convolutional layer

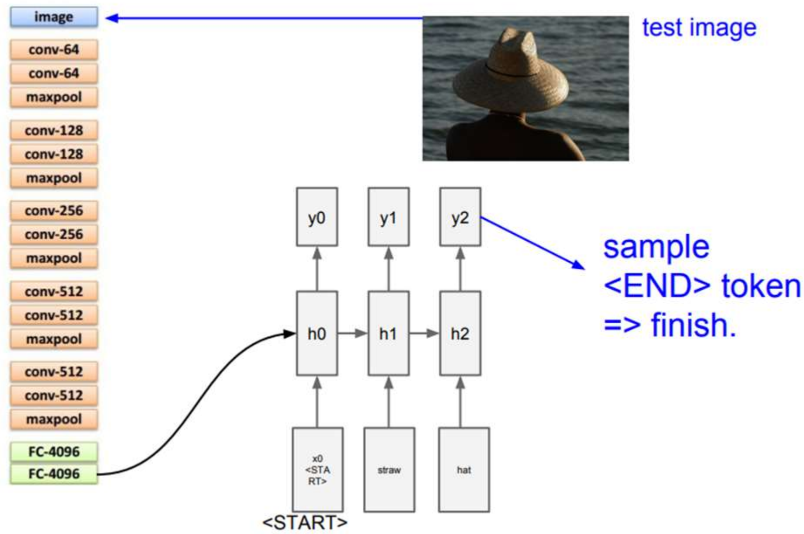


For example, for image captioning

Dr. Hsiu-Wen (Kelly) Chang Joly, Center for Robotics, Mines Paristech, PSL

40

Image captioning



Dr. Hsiu-Wen (Kelly) Chang Joly, Center for Robotics, Mines Paristech, PSL

41

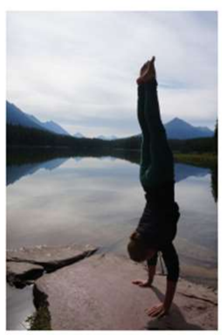
Image Captioning

Image Captioning: Failure Cases

Captions generated using neuraltalk2
All images are CC0 Public domain: fur coat, handstand, spider web, baseball

A woman is holding a cat in her hand

A person holding a computer mouse on a desk



A woman standing on a beach holding a surfboard

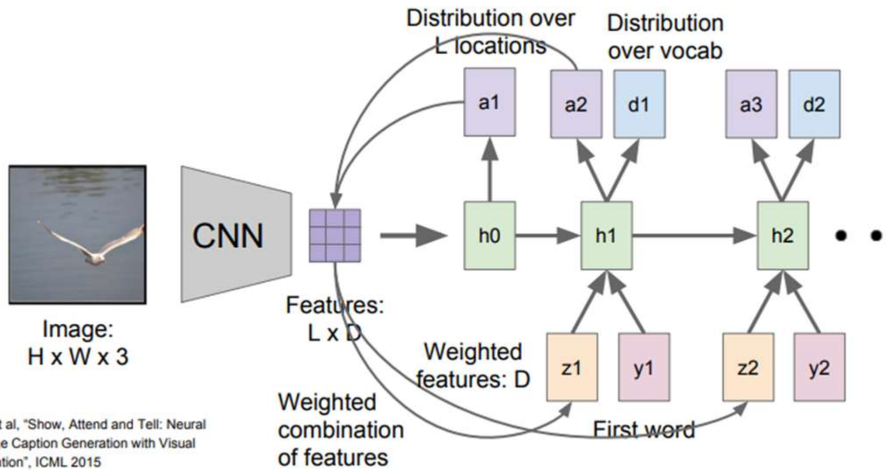
A bird is perched on a tree branch

A man in a baseball uniform throwing a ball

Dr. Hsiu-Wen (Kelly) Chang Joly, Center for Robotics, Mines Paristech, PSL

42

Image captioning with attention

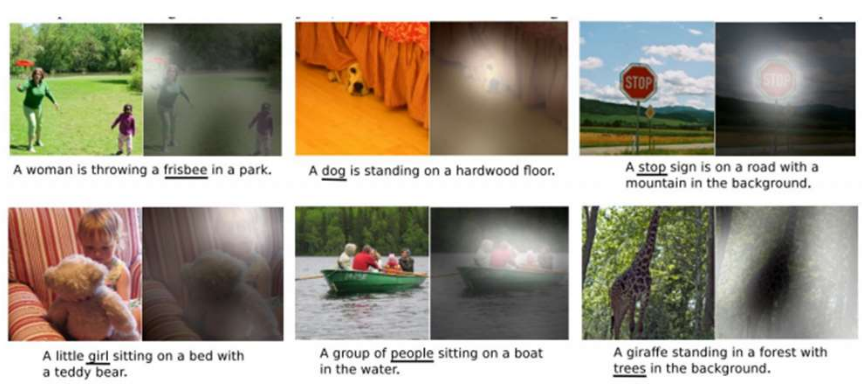


Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

Dr. Hsiu-Wen (Kelly) Chang Joly, Center for Robotics, Mines Paristech, PSL

43

Image captioning with attention



Xu et al, "Show, Attend, and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

Dr. Hsiu-Wen (Kelly) Chang Joly, Center for Robotics, Mines Paristech, PSL

44

Visual Question Answering

Q: What endangered animal is featured on the truck?	Q: Where will the driver go if turning right?	Q: When was the picture taken?	Q: Who is under the umbrella?
A: A bald eagle.	A: Onto 24 ¾ Rd.	A: During a wedding.	A: Two women.
A: A sparrow.	A: Onto 25 ¾ Rd.	A: During a bar mitzvah.	A: A child.
A: A humming bird.	A: Onto 23 ¾ Rd.	A: During a funeral.	A: An old man.
A: A raven.	A: Onto Main Street.	A: During a Sunday church service	A: A husband and a wife.

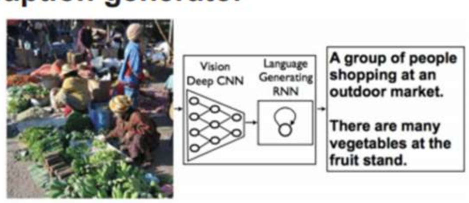
Agrawal et al, "VQA: Visual Question Answering", ICCV 2015
Zhu et al, "Visual 7W: Grounded Question Answering in Images", CVPR 2016
Figure from Zhu et al, copyright IEEE 2016. Reproduced for educational purposes.

45

Applications of RNN/LSTM

Wherever data is intrinsicly SEQUENTIAL

- **Speech recognition**
- ***Natural Language Processing (NLP)***
  - **Machine-Translation**
  - **Image caption generator**



- **Gesture recognition**
- **Music generation**
- ***Potentially any kind of time-series!!***

Check GPT-3 that can be used in a large of different applications

46

Summary

- RNNs allow a lot of flexibility in architecture design
- Vanilla RNNs are simple but don’t work very well
- Common to use LSTM or GRU: their additive interactions improve gradient flow
- Backward flow of gradients in RNN can explode or vanish
- Exploding is controlled with gradient clipping
- Vanishing is controlled with additive interactions (LSTM)
- Better/simpler architectures are a hot topic of current research
- Better understanding (both theoretical and empirical) is needed.