Session 3

Recurrent Neural Network

Acknowledgements

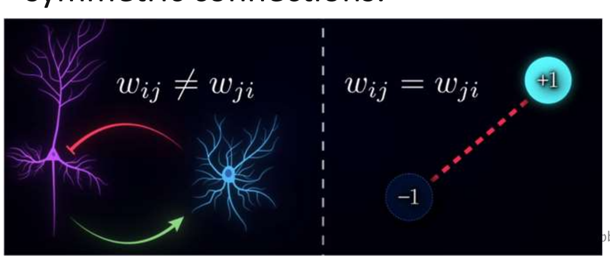
- The materials majorly derived from Prof. Fabien Moutarde. Some slides come from the online classes.
  - **Fei-Fei Li + J.Johnson + S.Yeung: slides on "*Recurrent Neural Networks*" from the** "*Convolutional Neural Networks for Visual Recognition*" **course at Stanford**
    http://cs231n.stanford.edu/slides/2019/cs231n_2019_lecture10.pdf
  - **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/

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
  - BAM
  - Associative memory networks, A.K.A. Hopfield [John Hopfield, 1982]
  - Boltzmann machine
  - Recurrent backpropagation nets [Hecht-Nielsen, 1990].
- From the computational perspective, a dynamic neural network that contains the feedback loop 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.

Hopfield network (1)

- It is a foundational model of associative memory that underlies many important ideas in neuroscience and machine learning, such as Boltzmann machines and Dense associative memory
- Neurons are binary units (on or off)
- Unlike biological system with asymmetric connections of the synapse, the Hopfield network simplifies and stabilizes the calculation to symmetric connections.



$x_i x_j > 0: agree\ with\ each\ other$

$x_i x_j < 0: disagree\ with\ each\ other$

$happiness\ of\ edge_{ij} = w_{ij} x_i x_j$

$$Network\ happiness = \sum_{ij}^{edges} w_{ij} x_i x_j$$

Degree of agreement between states and connection weights: goal is to maximin it

Hopfield network (2)

- The energy is the opposite of happiness and the goal becomes to minimize the energy
  - Adjusting weights: sculping energy landscape, creating local minima at memory locations ➔ learning
  - Adjusting states: evolve the system towards local minima by descending along the surface ➔ inference
- The energy of state vector, v, is defined as

$$E(v) = -\left(\sum_{i} s_i^v b_i + \sum_{i<j} s_i^v s_j^v w_{ij}\right)$$

  - w_{ij} : weight on the connection between unit I and j.
  - s_j^v: given state vector v, 1 if unit j is on and 0 otherwise
  - b_i : bias of unit i

Hopfield network (3)

- Inference is like a voting that the neuron are updated based on the voting rule.
- Randomly pick neuron x and calculate the result of voting from all the connection: $h_i = \sum_{j \neq i} w_{ij} x_i$
- Update: $x_i = \{\begin{matrix} +1 & if\ h_i > 0 \\ -1 & if\ h_i < 0 \end{matrix}$
- These gradually decrease the energy ➔ expected to reach the minimum energy (steady state) when no neuron is flipped
- Is this steady state guarantee the global energy descend? Mathematically, it will be when the connection w_{ij} is symmetric

HC0

Slide 6

HC0 Cite paper

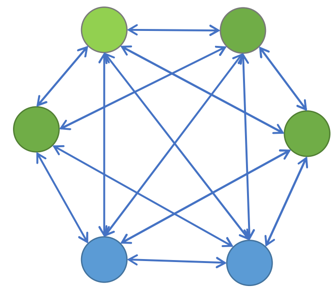
HsiuWen Chang, 2024-11-22T16:16:27.120

Boltzman Machine (1)

- BM is a network of symmetrically connected, neuron-like units that make stochastic decisions about whether to be on or off

$$z_i = b_i + \sum_j s_j w_{ij}$$

$$prob(s_i = 1) = \frac{1}{1 + e^{-z_i}}$$



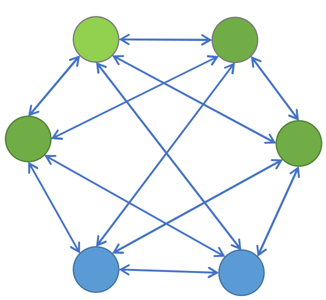
- Undirected models, where the connection goes both ways.
- When there is no more change during the update of units in any order, stationary distribution is reached

$$P(v) = \frac{e^{-E(v)}}{\sum_u e^{-E(u)}}$$

- Purpose is to optimize the solution: search problem (fixed weights as cost function, find best state) and learning problem (use set of data vectors and learn weights
- Discover features from datasets composed of binary vectors

Boltzman Machine (2)

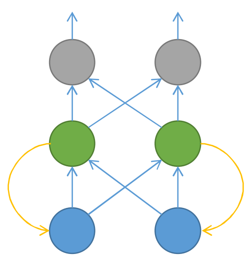
- Restricted Boltzman machine removes the connections between hidden neurons and those between input neurons.

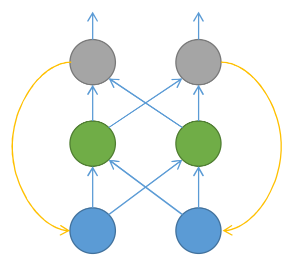


MINES ParisTech | PSL

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

9

9

MINES ParisTech | PSL

Flexibility

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

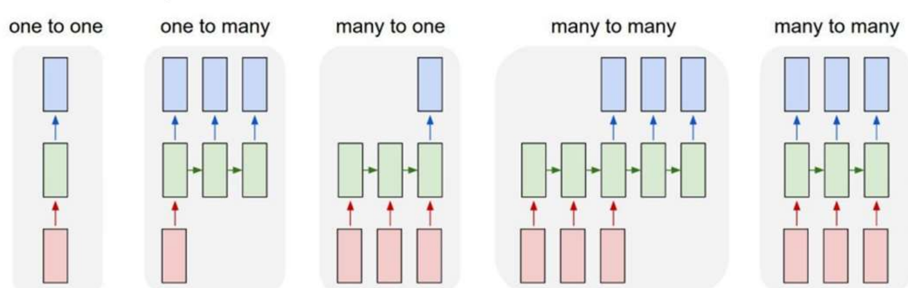


Image captioning
Image->seq of words

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

10

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

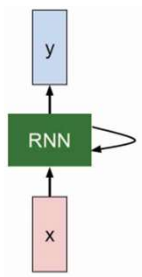
New hidden state

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

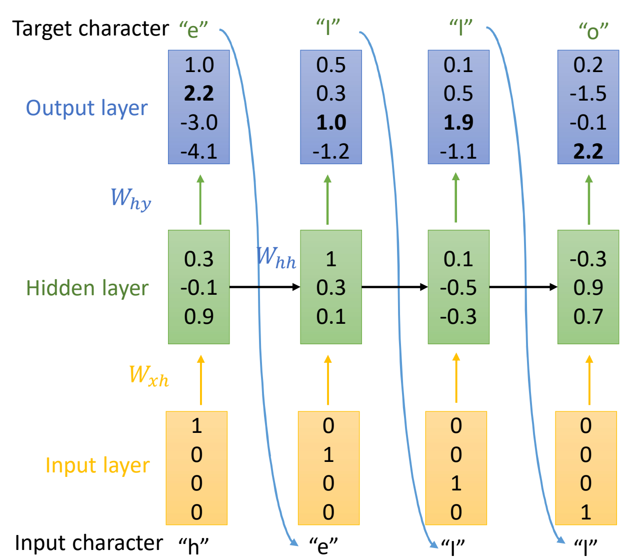
13

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.

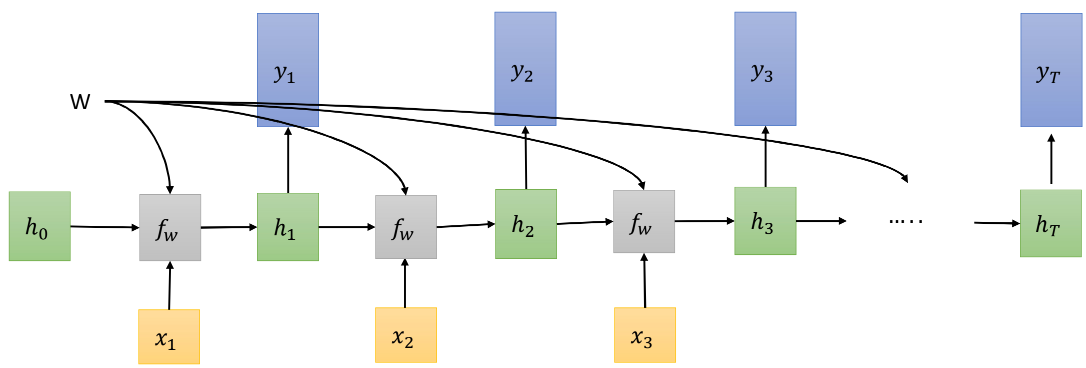


Dr. Hsiu-Wen (Kelly) Chang Joly, Center for Robotics, Mines Paristech, PSL

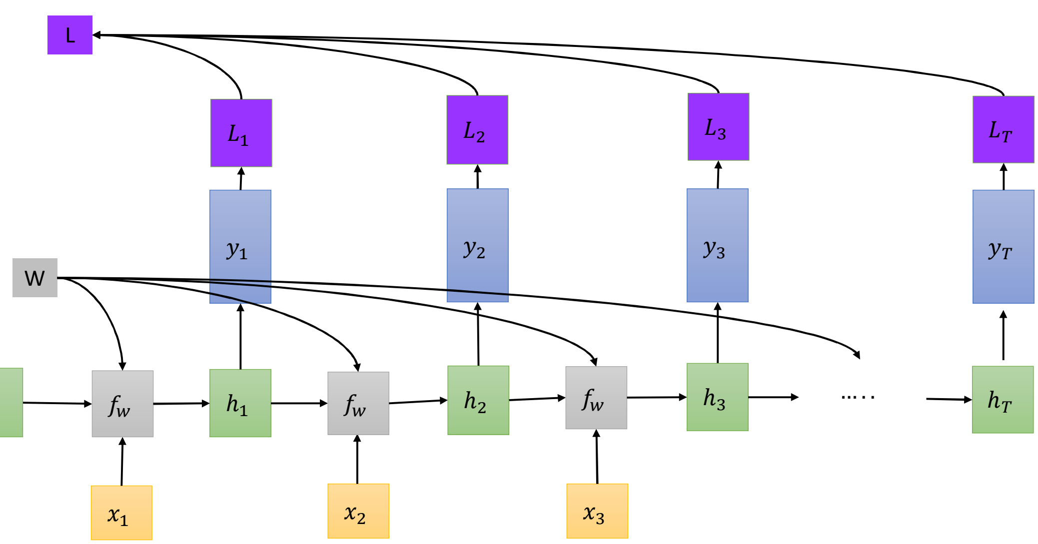
14

RNN: computation graph M2M

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



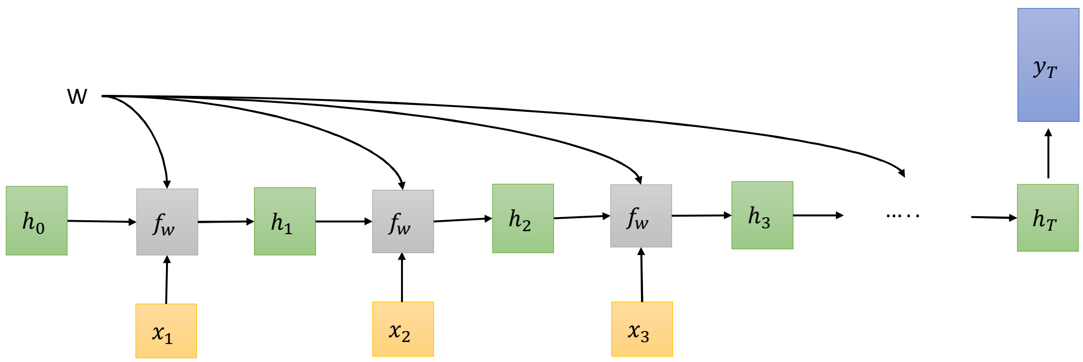
RNN: computation graph M2M training



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

RNN: computation graph M2O

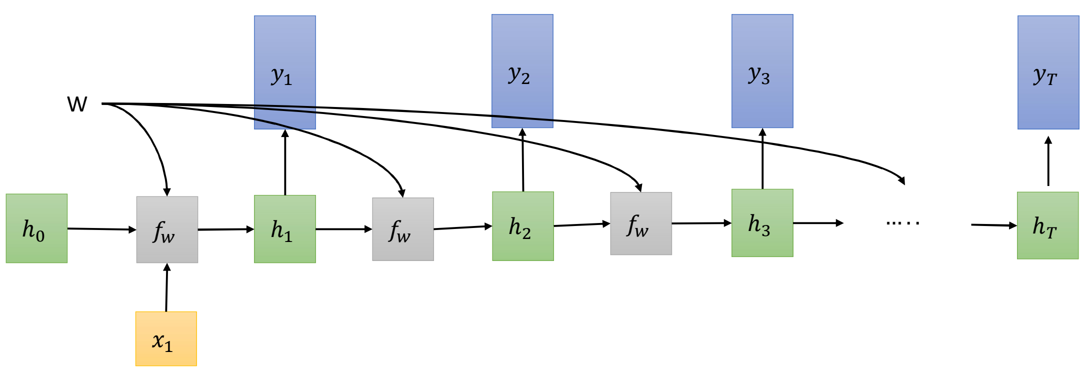
- RNN forward pass: Many to one: (semantic judge)



17

RNN: computation graph O2M

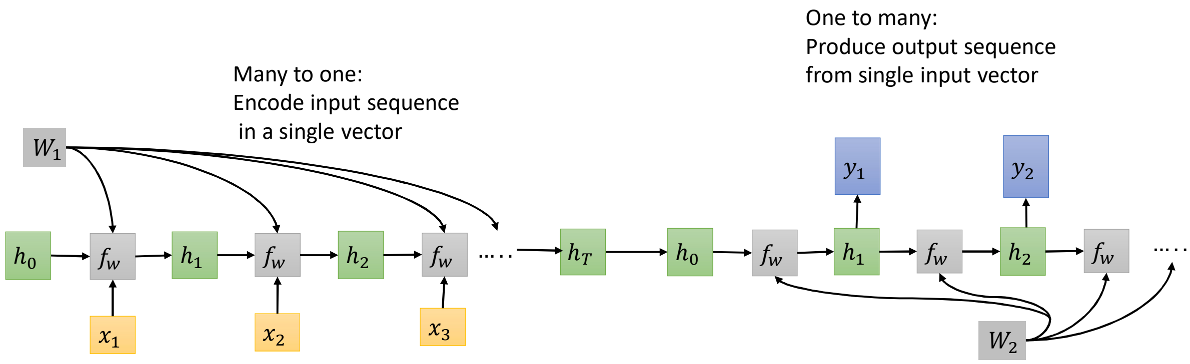
- RNN forward pass: One to many: (captioning)



18

Sequence to Sequence: O2M+M2O

- O2M+M2O



19

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.

20

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

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

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:

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

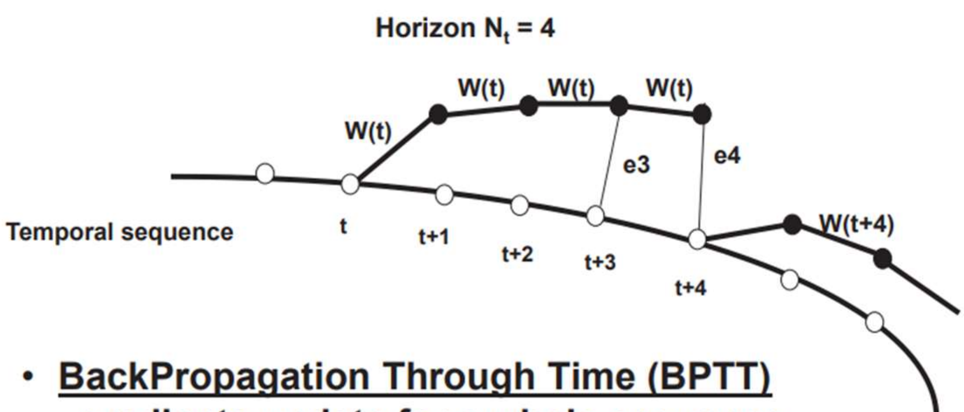
Analysis of gradients

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

- 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]

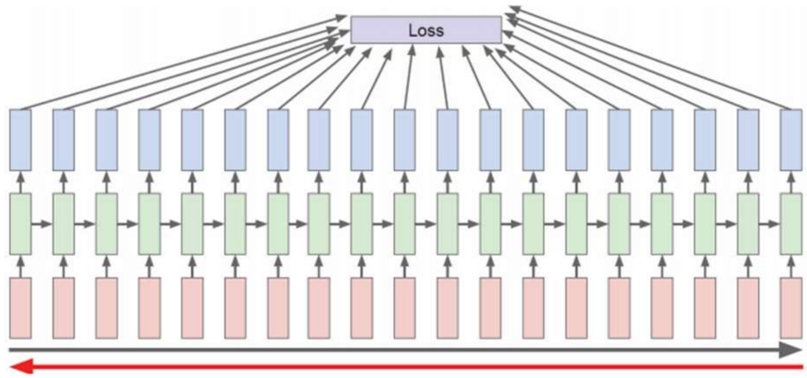
Jaeger, H. (2002). *Tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the" echo state network" approach*. Vol. 5. GMD-Forschungszentrum Informationstechnik Bonn
Tallec, C., & Ollivier, Y. (2017). Unbiasing truncated backpropagation through time. *arXiv preprint arXiv:1705.08209*.

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**

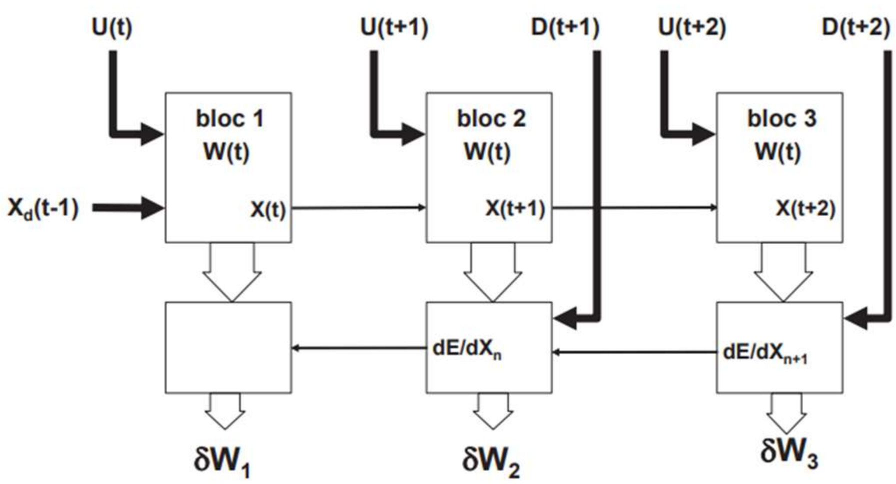
BackPropagation THROUGH TIME (BPTT)



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

25

BPTT computation principle

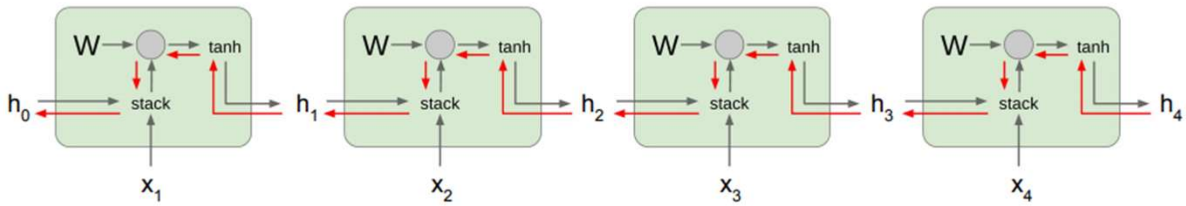


$$\delta W = \delta W_1 + \delta W_2 + \delta W_3$$

26

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!

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

27

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!

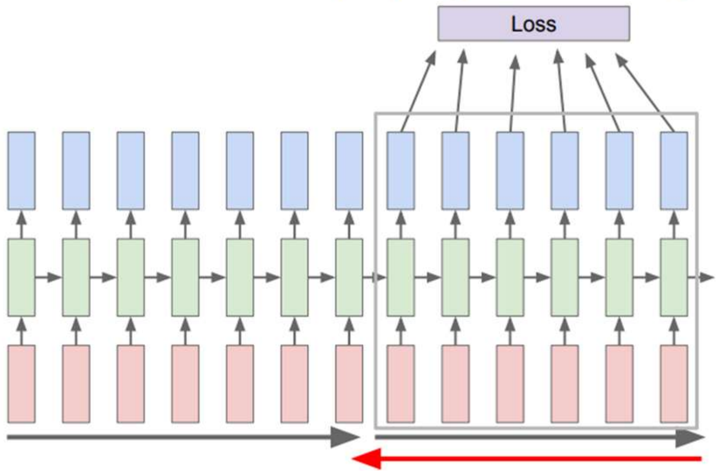
Recommended code to read for better understand this slide: https://gist.github.com/karpathy/d4dee566867f8291f086

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

28

Truncated tricks

Truncated Backpropagation through time

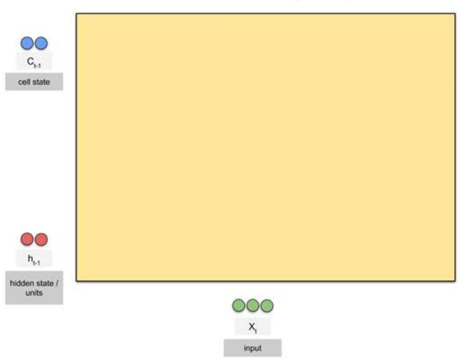


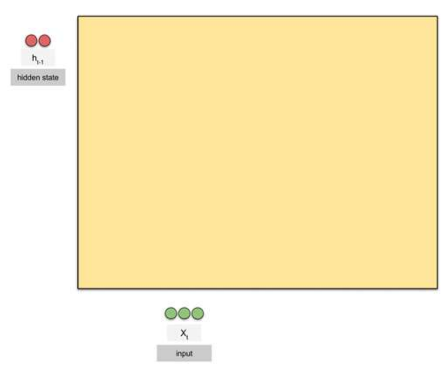
Carry hidden states forward in time forever, but only backpropagate for some smaller number of steps

29

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]

30

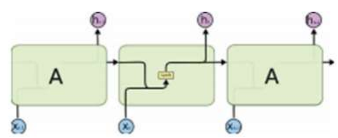
Long short-term memory

- 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): $o \in [0,1]$

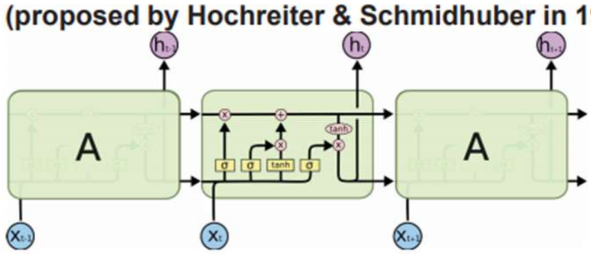
31

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**

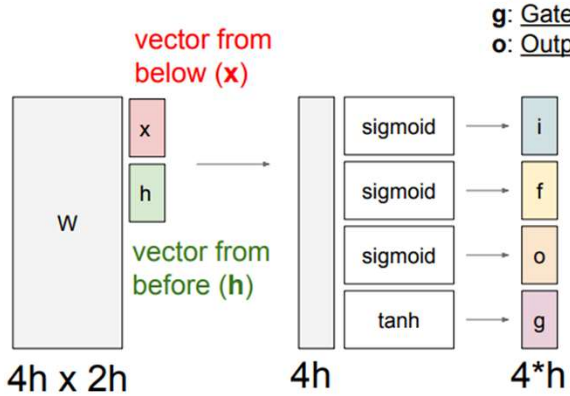
32

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



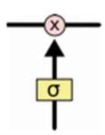
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

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

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

33

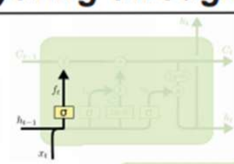
LSTM gates



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

- **FORGET gate**

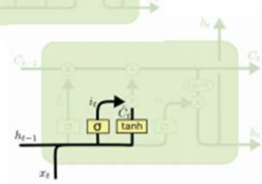
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \;+\; b_f\right)$$



- **INPUT gate**

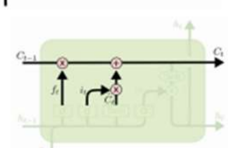
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \;+\; b_i\right)$$

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



➔ next state = mix between pure memory or pure new

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



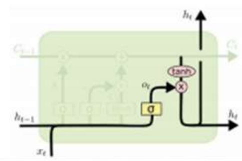
[Figures from https://colah.github.io/posts/2015-08-Understanding-LSTMs/]

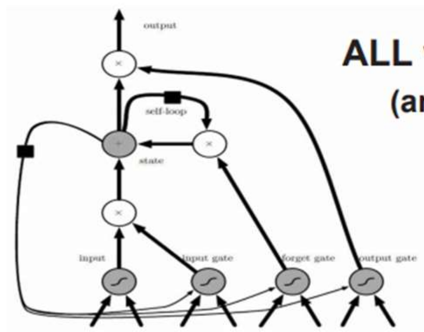
34

LSTM summary

- **OUTPUT gate**

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



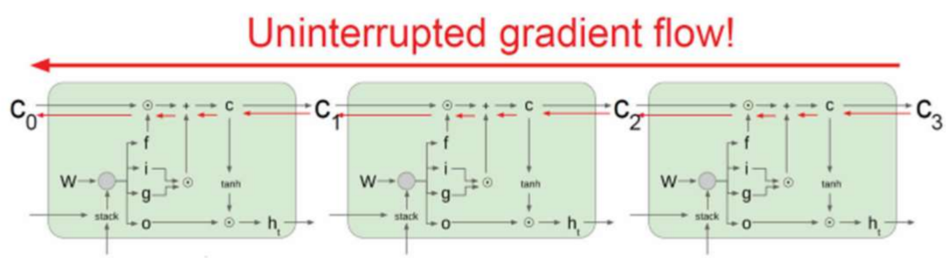


ALL weigths $\mathtt{W_f}$, $\mathtt{W_i}$, $\mathtt{W_c}$ and $\mathtt{W_o}$ (and biases) are LEARNT

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

35

Why LSTM avoids vanishing gradients?

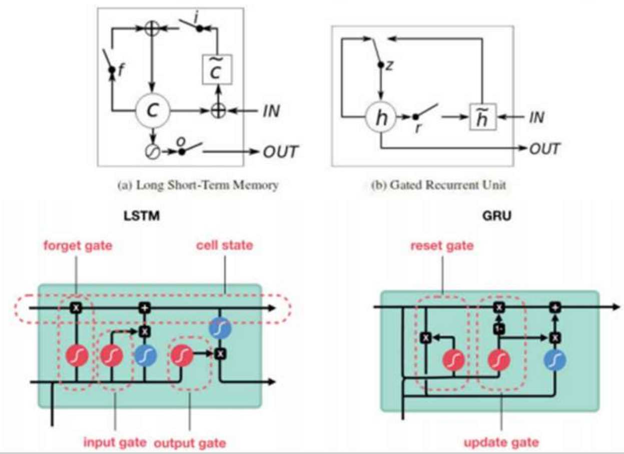


36

MINES ParisTech | PSL

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)



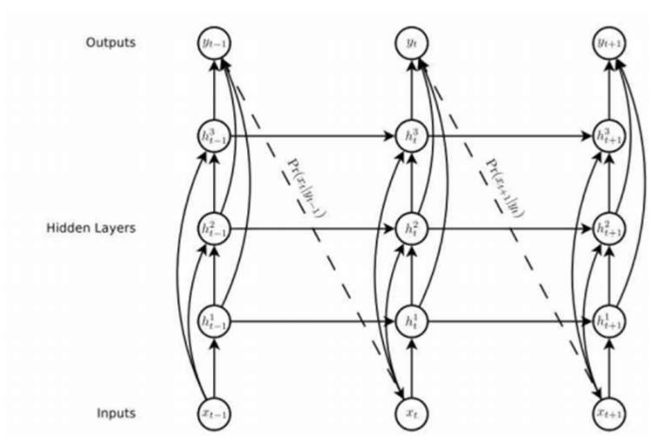
GRU [Learning phrase representations using rnn encoder-decoder for statistical machine translation, Cho et al. 2014]

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

37

MINES ParisTech | PSL

Deeper RNN

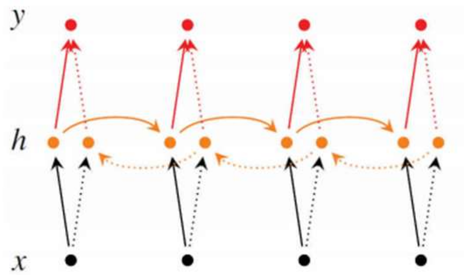


Several RNNs stacked (like layers in MLP)

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

38

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.

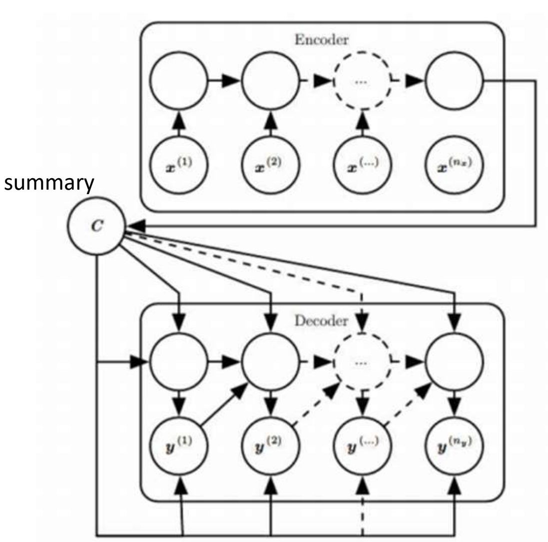
(e.g. for offline classification of sequence of words)

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

39

Cho, K., van Merrienboer, B., Gulcehre, C., Bougares, F., Schwenk, H., and Bengio, Y. (2014a). Learning phrase representations using RNN encoder-decoder for statistical machine translation. In Proceedings of the Empiricial Methods in Natural Language Processing (EMNLP 2014)

Encoder-decoder RNN



An RNN can learn a probability distribution over a sequence by being trained to predict the next symbol in a sequence: $p(x) = \prod_{t-1}^{T} p(x_t|x_{t-1}, \dots, x_1)$

$h(t) = f_1(h_{t-1}, x_t)$

$\mathrm{c} = q(h_{n_x-1}, x_{n_x})$

$h(t) = f_2(h_{t-1}, y_{t-1}, c)$

The conditional distribution of the next output:

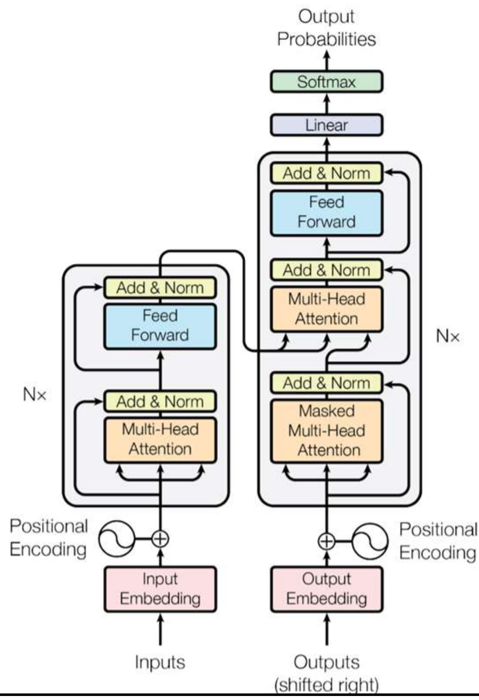
$p(y_t|y_{t-1}, \dots, y_1, c) \ = g(h_{t-1}, y_{t-1}, c)$

The two components of the proposed RNN Encoder–Decoder are jointly trained to maximize the conditional log-likelihood.

$$\max_\theta \frac{1}{N}\sum_{n=1}^{N} \log p_\theta(y_n|x_n)$$

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

40



Transformer

) Chang Joly, Center for Robotics, Mines Paristech, PSL

41

What is Embedding

- Embedding layer maps input information from high-dimensional to a lower-dimensional space, allowing the network to learn more about the relationship between inputs and to process the data more efficiently.
- When the data is not numerical values, such as in natural language processing (NLP), finding a meaningful representation and inter-word semantics numeric vectors plays a critical role in machine learning.
- In the previous example, one-hot vectors are high-dimensional and sparse and without the meaning of each word.
  - Text embedding
  - Image embedding
  - Graph embedding and others

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

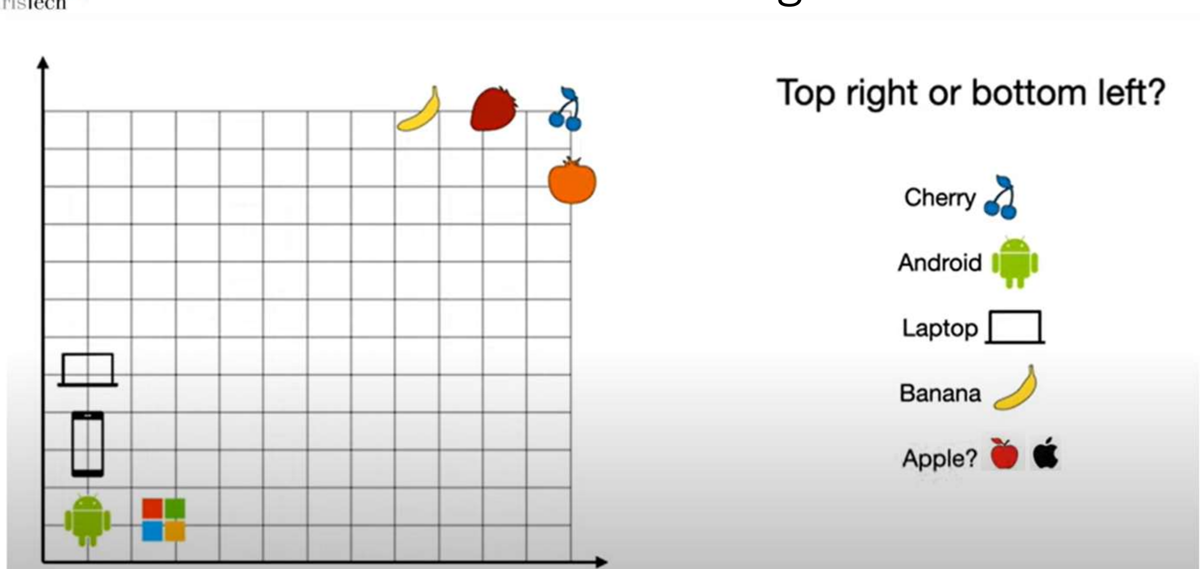
42

Text embedding

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

43

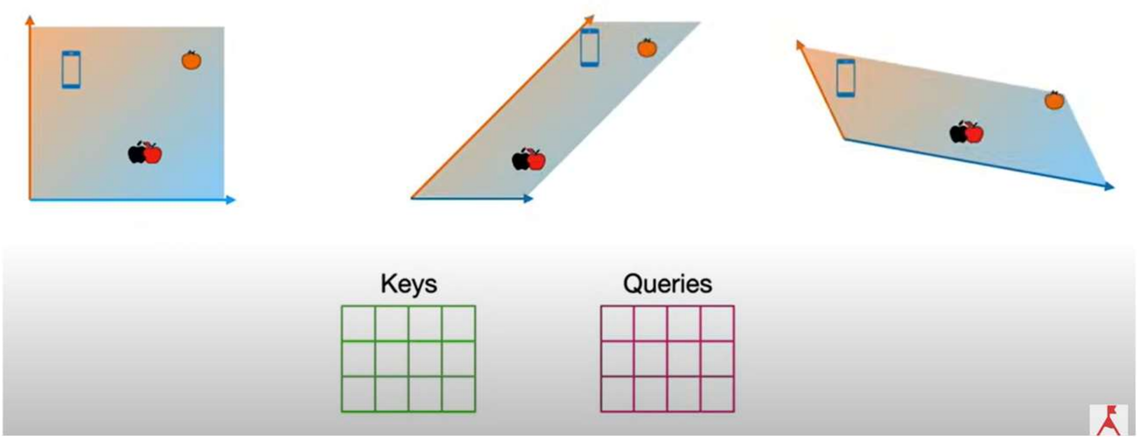
Problem of embedding



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

44

Purpose of Multi-heads attention



Attention mechanism

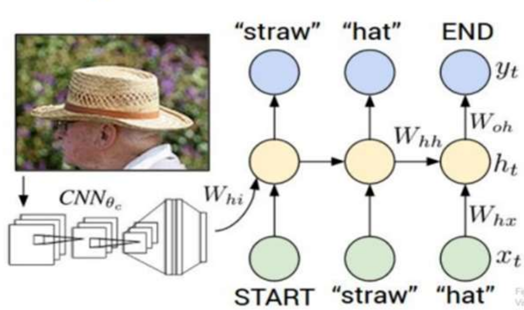
- The attention mechanism is a way to "align" the input and output in natural language process application (NLP) where the RNN is used to build the temporal information in a sequence but can't properly link the component of inputs and outputs in a good way. Vision applications provide the network a way to "pay attention" to specific pixels that are important for the output.
- Generally, there are two kinds of attention mechanisms:
  - A soft attention map is a fully differentiable deterministic mechanism that can be plugged into an existing system, and the gradients are propagated through the attention mechanism at the same time they are propagated through the rest of the network.
  - Hard attention is a stochastic process: instead of using all the hidden states as an input for the decoding, the system samples a hidden state yi with the probabilities si. To propagate a gradient through this process, we estimate the gradient by Monte Carlo sampling.

Recommended reading: RNN variants

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

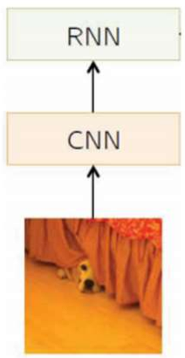
Applications

Image Captioning

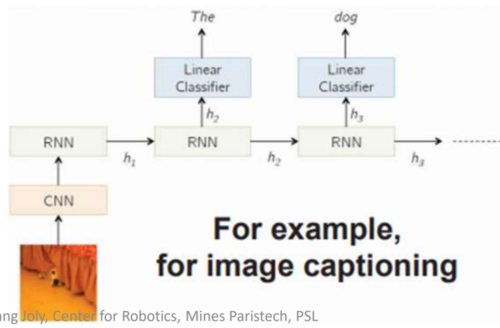


Explain Images with Multimodal Recurrent Neural Networks, Mao et al.
Deep Visual-Semantic Alignments for Generating Image Descriptions, Karpathy and Fei-Fei
Show and Tell: A Neural Image Caption Generator, Vinyals et al.
Long-term Recurrent Convolutional Networks for Visual Recognition and Description,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

Application mixed vision and text



Input into RNN the features from last convolutional layer

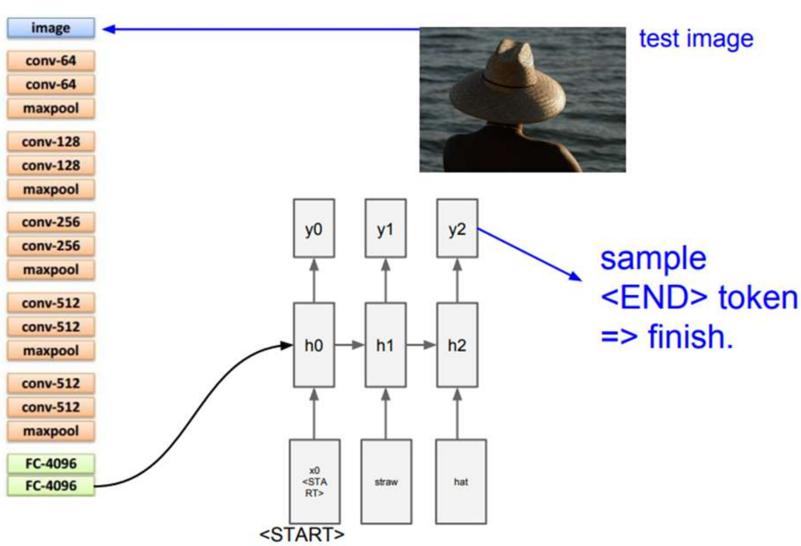


For example, for image captioning

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

49

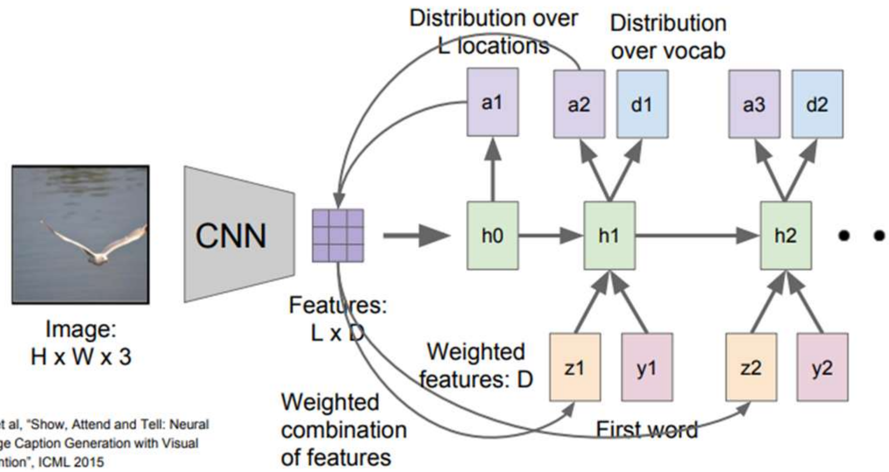
Image captioning



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

50

Image captioning with attention

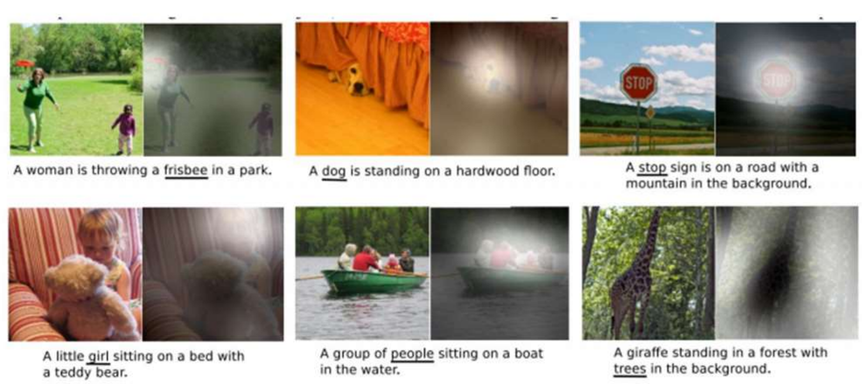


Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

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

51

Image captioning with attention



Xu et al, "Show, Attend, and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015
Figure copyright Kelvin Xu, Jimmy Lei Ba, Jamie Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhutdinov, Richard S. Zemel, and Yoshua Benchio, 2015. Reproduced with permission.

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

52

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.