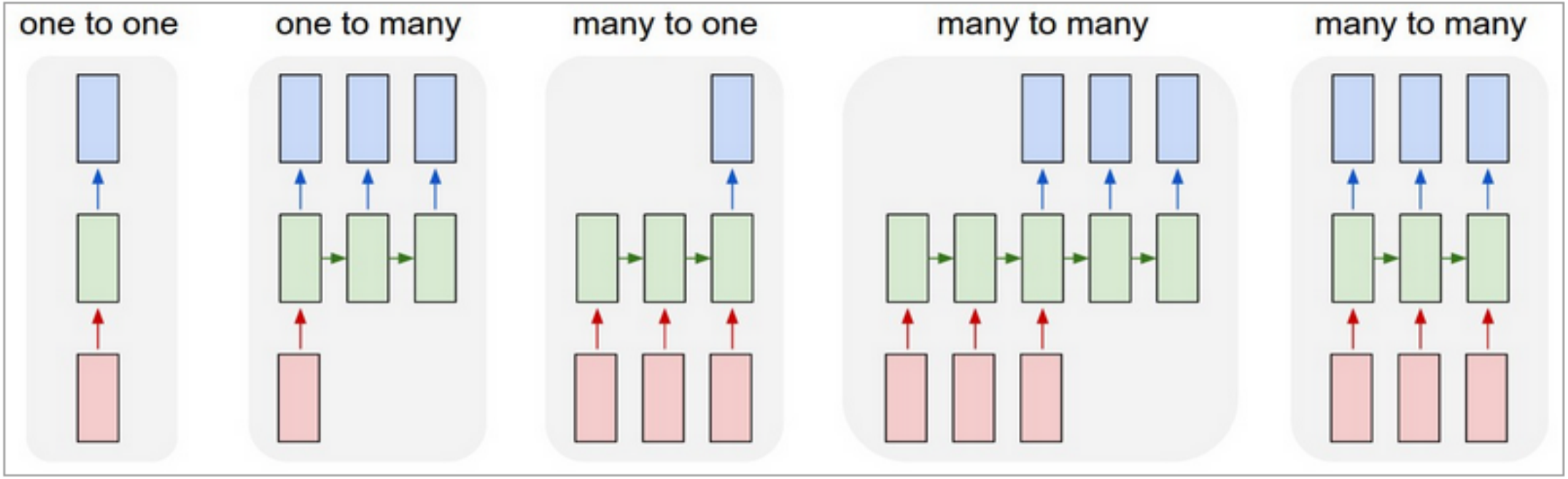
Time Series and Sequence Classification

Palacode Narayana Iyer Anantharaman

22 May 2018

RNN (Courtesy: A Karpathy's Blog)



Each rectangle is a vector and arrows represent functions (e.g. matrix multiply). Input vectors are in red, output vectors are in blue and green vectors hold the RNN's state (more on this soon). From left to right: **(1)** Vanilla mode of processing without RNN, from fixed-sized input to fixed-sized output (e.g. image classification). **(2)** Sequence output (e.g. image captioning takes an image and outputs a sentence of words). **(3)** Sequence input (e.g. sentiment analysis where a given sentence is classified as expressing positive or negative sentiment). **(4)** Sequence input and sequence output (e.g. Machine Translation: an RNN reads a sentence in English and then outputs a sentence in French). **(5)** Synced sequence input and output (e.g. video classification where we wish to label each frame of the video). Notice that in every case are no pre-specified constraints on the lengths sequences because the recurrent transformation (green) is fixed and can be applied as many times as we like.

Building Blocks To Process Text

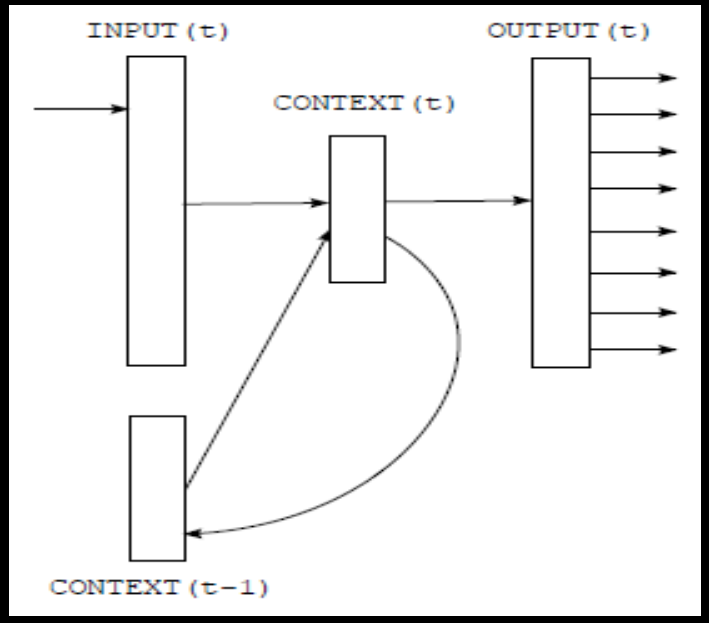
- Characters
  - Character Representation – e.g one hot
- Words
  - Word Representation
- Sentences
  - RNN, Seq2Seq (Encoder-Decoder)
- Text
  - Memory networks

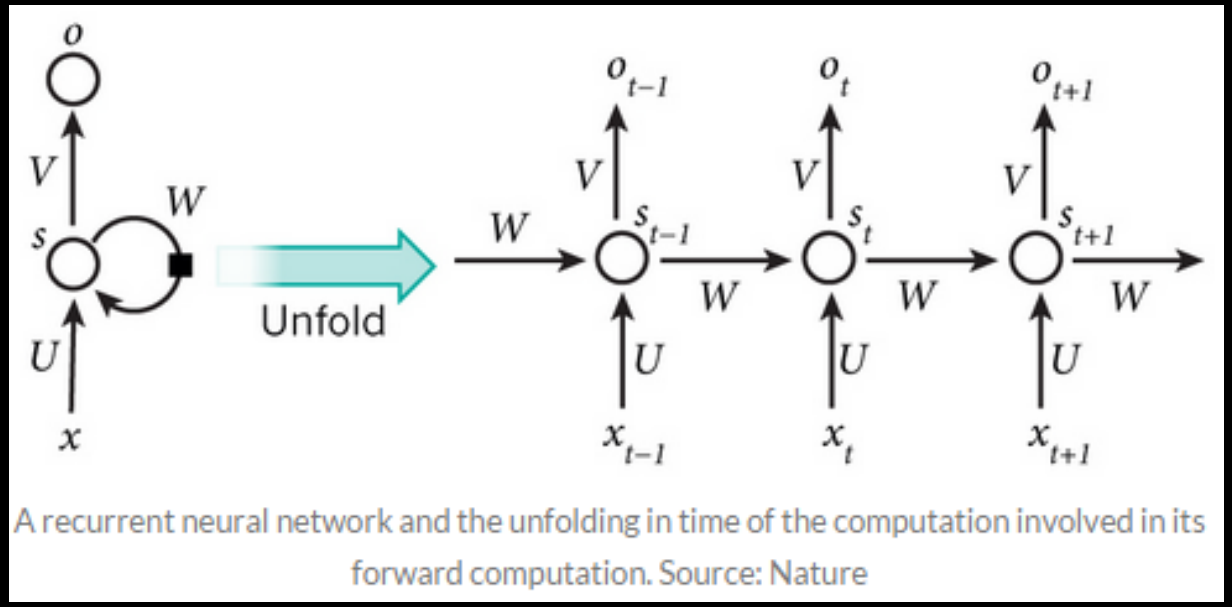
RNN Model with sigmoid nonlinearity and softmax output

$$h_t = \sigma(W^{hh} h_{t-1} + W^{hx} x_t)$$

$$y_t = softmax(W^S h_t)$$

$$Loss\ Function\ at\ a\ step\ t = J^t(\theta) = -\sum_{j=1}^{K} t_{t,j} \log y_{t,j}$$





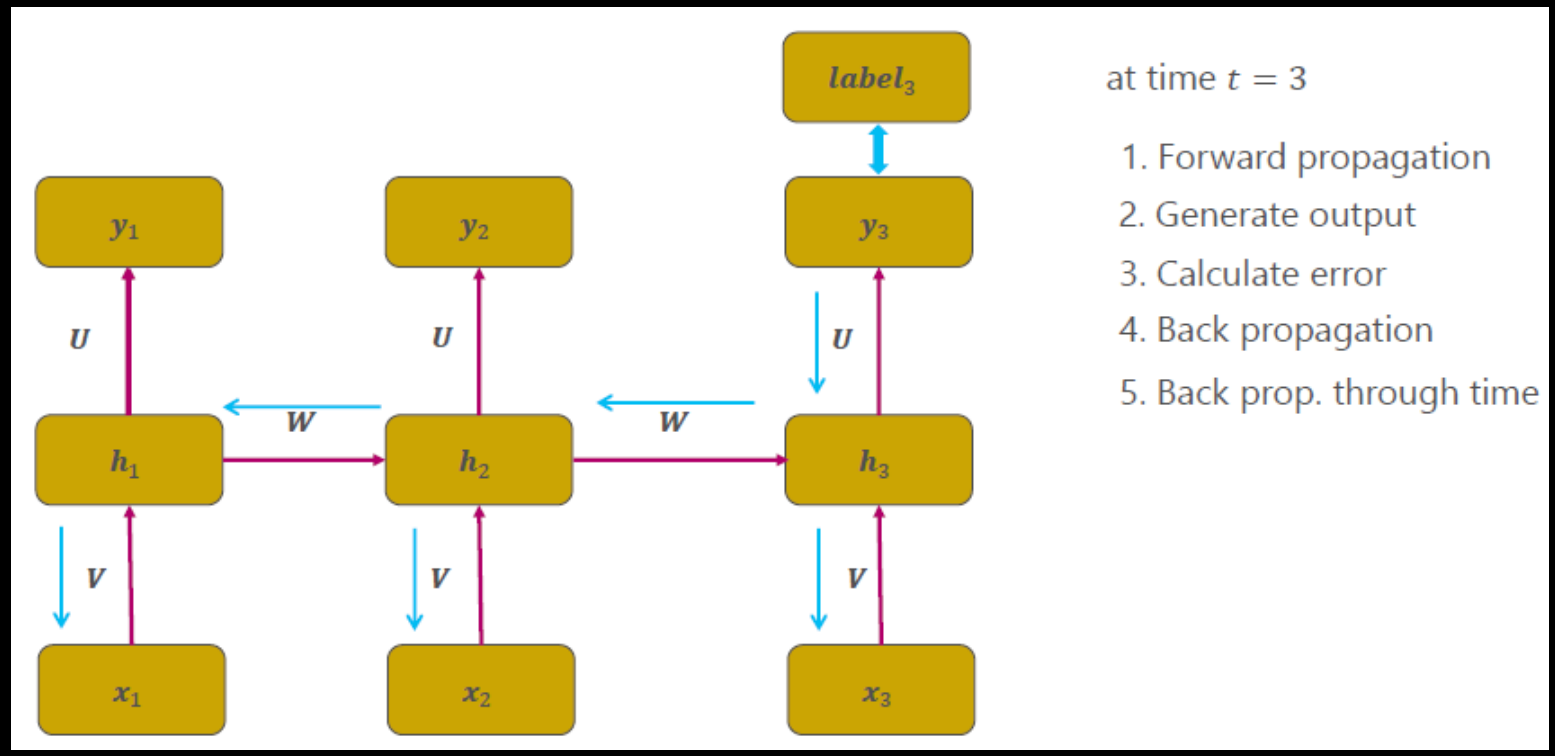
A recurrent neural network and the unfolding in time of the computation involved in its forward computation. Source: Nature

Training Algorithm (Fig: Xiodong He etal, Microsoft Research)

- Different training procedures exist, we will use Back Propagation Through Time (BPTT)

- Similar to standard backpropagation, BPTT involves using chain rule repeatedly and bakpropagating the deltas

- However one key subtlety is that, for RNNs, the cost function depends on the activation of hidden layer not only through its influence on output layer but also through its influence on hidden layer of the next time step



$$\delta_h^t = \theta'(a_h^t)\left(\sum_{k=1}^{K}\delta_k^t w_{hk} + \sum_{h'=1}^{H}\delta_{h'}^{t+1} w_{hh'}\right)$$

where

$$\delta_j^t \stackrel{\text{def}}{=} \frac{\partial \mathcal{L}}{\partial a_j^t}$$

A sketch of implementation – Forward pass

Forward Propagation – Key steps

for t from 1 to T

1. Compute hidden activations of time t with current input and hidden activations for (t-1)
2. For all j in the output units compute the net_j (dot product of W^S with h_t)
3. Apply the softmax function on the net_j and get the probability distribution for time t

```
for t in xrange(len(inputs)):
  xs[t] = np.array(inputs[t]).reshape(len(inputs[t]), 1)
  hs[t] = np.tanh(np.dot(self.Wxh, xs[t]) + np.dot(self.Whh, hs[t-1]) + self.bh)
  ys[t] = np.dot(self.Why, hs[t]) + self.by # unnormalized log probabilities
  ps[t] = np.exp(ys[t]) / np.sum(np.exp(ys[t])) # softmax probabilities
```

A sketch of implementation – Backpropagation

Backpropagation for RNN – Key steps

for t from T down to 1

1. compute the delta at the output (dy)
2. Compute Δw_{ji} where w is the (softmax) weight matrix W^S
3. Determine the bias terms
4. Backpropagate and compute delta for hidden layer (dhraw)
5. Compute the updates to weight matrix W^{hh} and W^{hx}
6. Perform BPTT by computing the error to be propagated to the previous layer (dbnext).

```
for t in reversed(xrange(len(inputs))):
  dy = np.copy(ps[t])
  # the following works for binary targ
  ind = targets[t].index(1)
  dy[ind] -= 1 # backprop into y
  dWhy += np.dot(dy, hs[t].T)
  dby += dy
  dh = np.dot(self.Why.T, dy) + dhnext
  dhraw = (1 - hs[t] * hs[t]) * dh # ba
  dbh += dhraw
  dWxh += np.dot(dhraw, xs[t].T)
  dWhh += np.dot(dhraw, hs[t-1].T)
  dhnext = np.dot(self.Whh.T, dhraw)
```

Sample Applications

- Language Model (Mikolov etal)
  - Input at a time t is the corresponding word vector
  - Output is the predicted next word
- Handwriting Recognition
- Image captioning and description
- Language translation
- Slot filling (see next slide)
- Character LM (Andrej Karpathy)
- Speech recognition
- Question Answering Systems
- NER
- And many more sequence based applications

What is a Language Model?

- Language Model assigns probability to a sequence of words P(w1, w2.., wn) formed out of the vocabulary V of a language

- Language Models can also generate text using a probabilistic approach (Shannon's work)

The Language Model Problem

- The LM problem is to assign a probability to a word sequence
  - $P(w_1, w_2,\ldots, w_n)$
  - Example: P(tomorrow, is, a, holiday)

- This may also be stated as a problem of computing the probability of each word given its preceding context.
  - $P(w_i \mid w_1, w_2 \ldots w_{i-1})$
  - Example: P(holiday | tomorrow, is, a)

Semantic Slot Filling Application Example

Many problems in Information extraction require generating a data structure from a natural language input

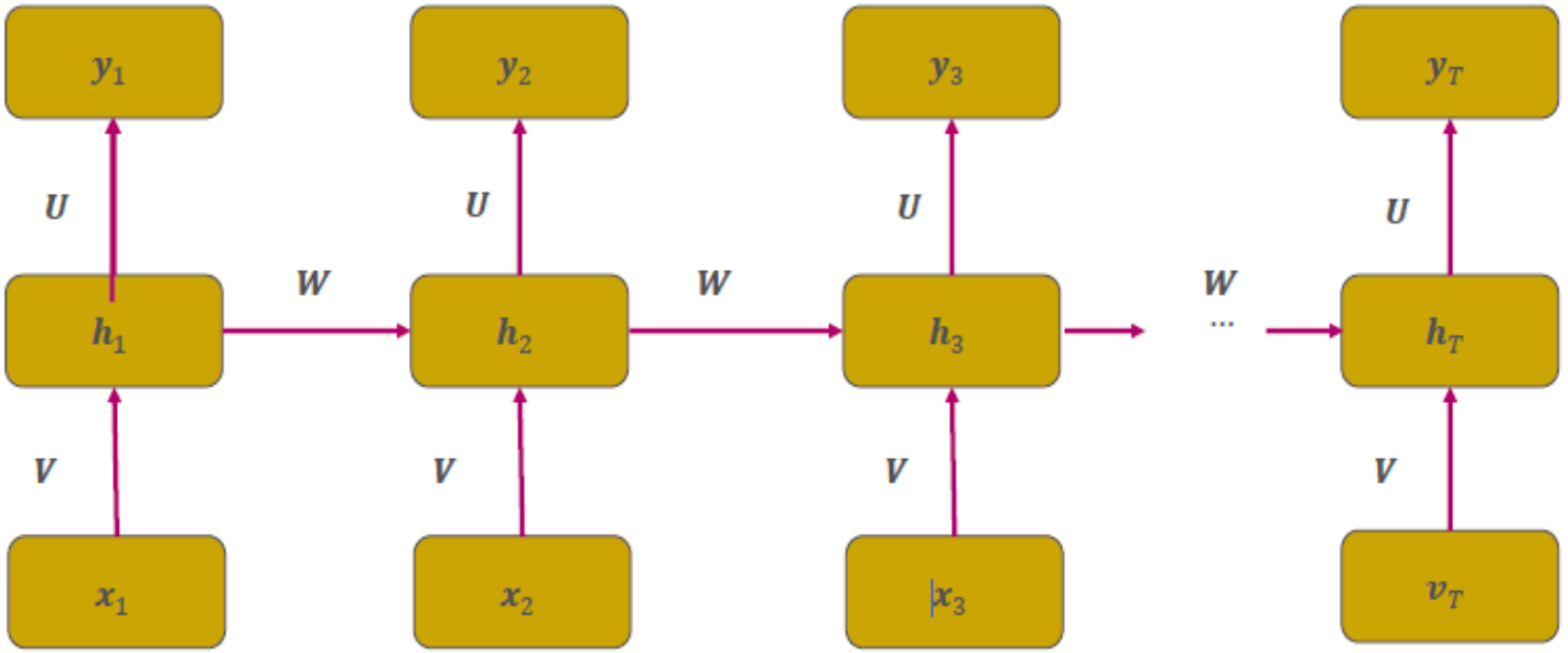
One possible way to cast this problem is to treat this as a slot filling exercise.

This can be viewed as a sequential tagging problem and use an RNN for tagging

h_t is the hidden layer that carries the information from time $0 \sim t$

where x_t: *the input word* , y_t: *the output tag*

$y_t = SoftMax(U \cdot h_t), where\ h_t = \sigma(W \cdot h_{t-1} + V \cdot x_t)$



[Mesnil, He, Deng, Bengio, 2013; Yao, Zweig, Hwang, Shi, Yu, 2013]

	show	flights	from	boston	to	new	york	today
Slots	O	O	O	B-dept	O	B-arr	I-arr	B-date

Building an NER with RNN

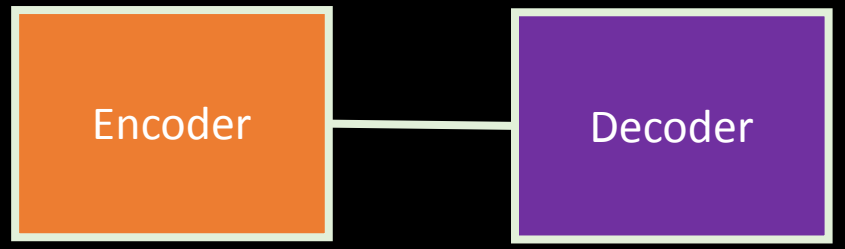
- The traditional MEMM or CRF based NER design techniques require domain expertise when designing the feature vector
- RNN based NER's don't need feature engineering and with some minimum text preprocessing (such as removing infrequent words), one can build an NER that provides comparable performance
- Steps:
  - Preprocess the words: tokenization and some simple task dependent preprocessing as needed
  - Get word vectors (this helps reducing the dimensionality)
  - Form the training dataset
  - Train the NER
  - Predict

Encoder Decoder Design

- Use 2 RNN's, one for encoding and the other decoding
  - Example: Machine Translation

- The activations of the final stage of the encoder is fed to the decoder

- Useful when the output sequence is of variable length and the entire input sequence can be processed before generating the output



Bidirectional RNNs

- Key idea:
  - Output at a step t not only depends on the past steps (t-1…t1) but also depends on future steps (t+1, …T).
  - The forward pass abstracts and summarizes the context in the forward direction while the backward pass does the same from the reverse direction
- Examples: Fill in the blanks below
  - I want _____ buy a good book _____ NLP
  - I want _____ Mercedes
- Let’s illustrate bidirectional RNNs with an application example from: Opinion Mining with Deep Recurrent Nets by Irsoy and Cardie 2014

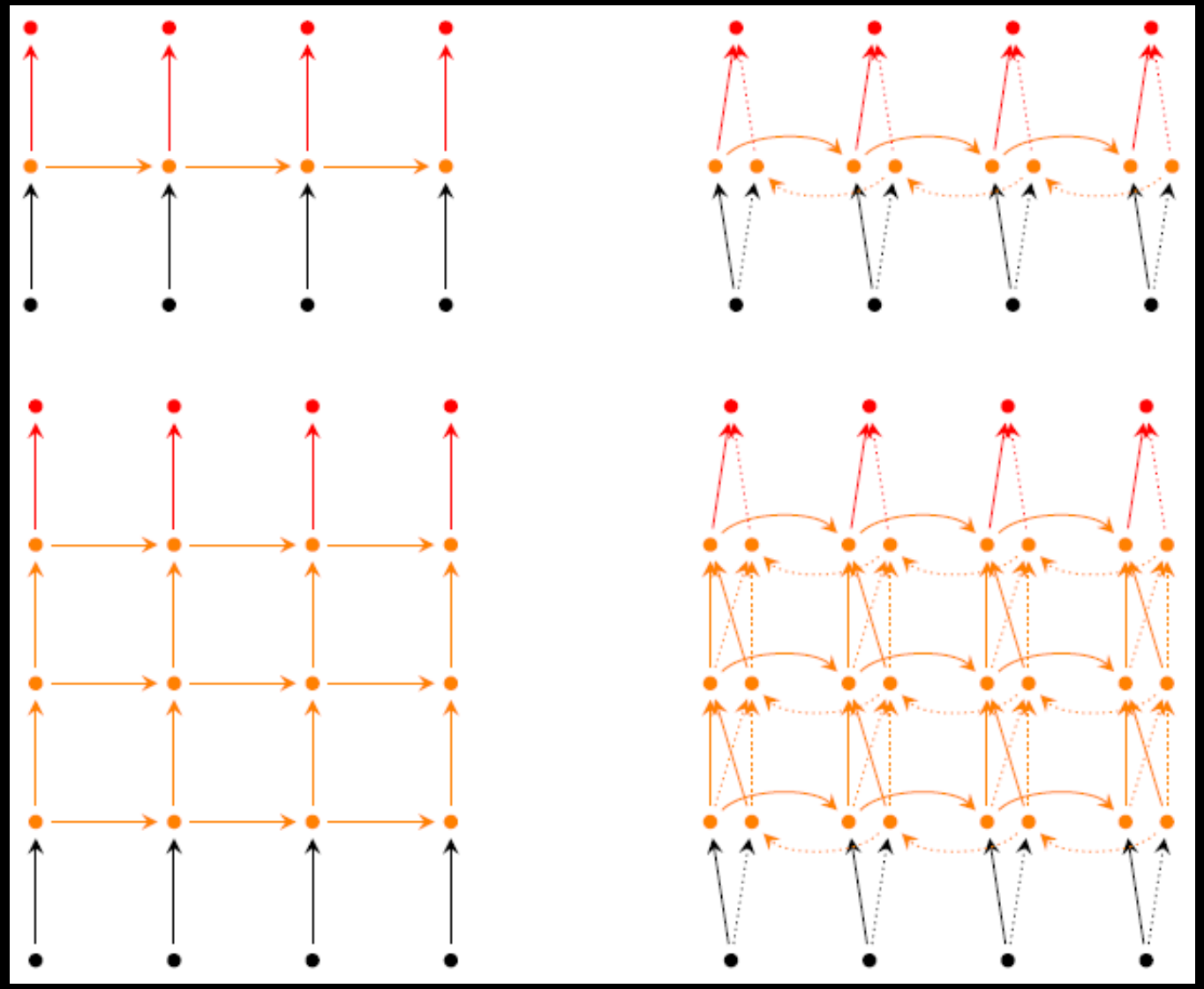
Problem Statement: Ref Irsoy and Cardie 2014

- Given a sentence, classify each word in to one of the tags: {O, B-ESE, I-ESE, B-DSE, I-DSE}
- Definitions
  - Direct Subjective Expressions (DSE): explicit mentions of private states or speech events expressing private states
  - Expressive Subjective Expressions (ESE): Expressions that indicate sentiment, emotion, etc., without explicitly conveying them.

Fine-grained opinion analysis aims to detect the subjective expressions in a text (e.g. "hate") and to characterize their intensity (e.g. strong) and sentiment (e.g. negative) as well as to identify the opinion holder (the entity expressing the opinion) and the target, or topic, of the opinion (i.e. what the opinion is about) (Wiebe et al., 2005). Fine-grained opinion analysis is important for a variety of NLP tasks including opinion-oriented question answering and opinion summarization. As a result, it has been studied extensively in recent years.

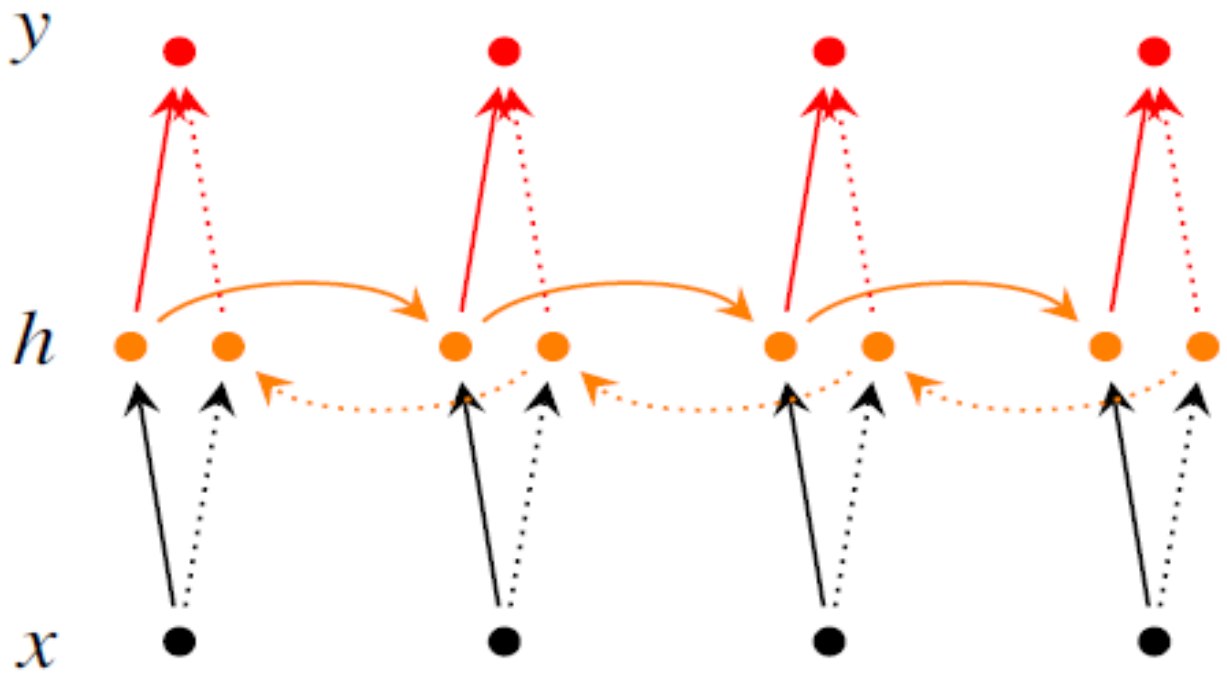
Bidirectional RNN Model

- Input: A sequence of words. At each time step t a single token (represented by its word vector) is input to the RNN. (Black dots)
- Output: At each time step t one of the possible tags from the tagset is output by the RNN (Red dots)
- Memory: This is the hidden unit that is computed from current word and the past hidden values. It summarizes the sentence up to that time. (Orange dots)



The model with 1 hidden layer

Problem: For classification you want to incorporate information from words both preceding and following



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

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

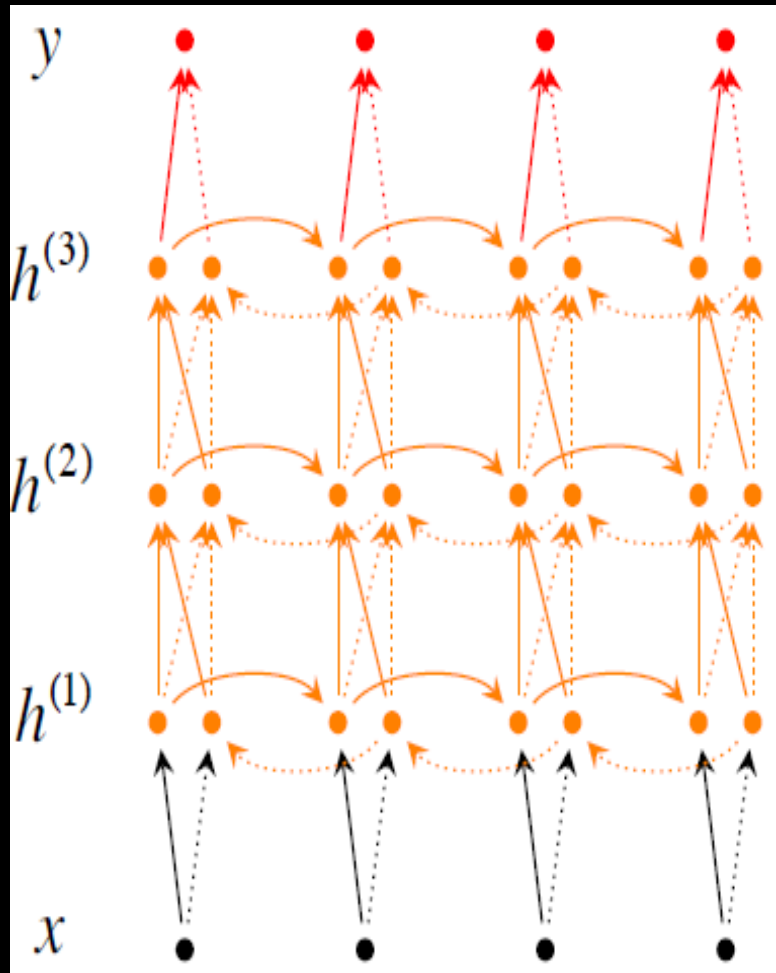
$$y_t = g(U[\vec{h}_t; \overleftarrow{h}_t] + c)$$

$$y_t = g(U_{\rightarrow}\vec{h}_t^{(L)} + U_{\leftarrow}\overleftarrow{h}_t^{(L)} + c)$$

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

Deep Bidirectional RNNs

- RNNs are deep networks with depth in time.
- When unfolded, they are multi layer feed forward neural networks, where there are as many hidden layers as input tokens.
- However, this doesn't represent the hierarchical processing of data across time units as we still use same U, V, W
- A stacked deep learner supports hierarchical computations, where each hidden layer corresponds to a degree of abstraction.
- Stacking a simple RNN on top of others has the potential to perform hierarchical computations moving over the time axis



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

$$\overleftarrow{h}_t^{(i)} = f(\overleftarrow{W}^{(i)} h_t^{(i-1)} + \overleftarrow{V}^{(i)} \overleftarrow{h}_{t+1}^{(i)} + \overleftarrow{b}^{(i)})$$

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

$$y_t = g(U_{\rightarrow} \overrightarrow{h}_t^{(L)} + U_{\leftarrow} \overleftarrow{h}_t^{(L)} + c)$$

Training the BRNN (ref: Alex Graves: Supervised Sequence Labelling with Recurrent Neural Networks)

Forward Pass

for t = 1 to T do

Forward pass for the forward hidden layer, storing activations at each time step

for t = T to 1 do

Forward pass for the backward hidden layer, storing activations at each time step

for all t, in any order do

Forward pass for the output layer, using the stored activations from both hidden layers

Backward Pass

for all t, in any order do

Backward pass for the output layer, storing terms at each time step

for t = T to 1 do

BPTT backward pass for the forward hidden layer, using the stored terms from the output layer

for t = 1 to T do

BPTT backward pass for the backward hidden layer, using the stored terms from the output layer

Vanishing Gradients: Why is this an issue?

- Recall: RNN's are temporally deep networks. Hence the problem of vanishing (or exploding) gradients are possible

- Long term dependencies in the sequence could still be a problem due to vanishing gradients issue
  - Example: Ananth was setting up the projector. Participants walked in. It was a pleasant Monday morning after a refreshing weekend. The participants warmly greeted --------- ?

Workaround: Clipping

- Key Idea: Avoid the vanishing/exploding gradient problem by looking at a threshold and clip the gradient to that threshold.

- While this is a simple workaround to address the issue, it is crude and might hamper the performance

- Better solutions: LSTMs and its variants like GRUs

Motivation for advanced architectures

- Consider the cases below, where a customer is interested in iPhone 6s plus and he needs to gift it to his father on his birthday on Oct 2. He goes through a review that reads as below:
  - Review 1: Apple has unveiled the iPhone 6s and iPhone 6s Plus - described by CEO Tim Cook as the "most advanced phones ever" - at a special event in San Francisco on Wednesday. Pre-orders for the new iPhone models begin this Saturday and they have a launch date (start shipping) in twelve countries on September 25. The price for the iPhone 6s and iPhone 6s Plus remain unchanged compared to their predecessors: $649 for the 16GB iPhone 6s, $749 for the 64GB iPhone 6s and 16GB iPhone 6s Plus, $849 for 128GB iPhone 6s and 64GB iPhone 6s Plus, and $949 for the 128GB iPhone 6s Plus (all US prices). **There's no word yet on India price or launch date**
- How would we design a RNN that advises him: Buy/No Buy?
- Suppose the customer doesn’t have the time constraint as above but has a price constraint, where his budget is around Rs 50K, what would be our decision?
- Suppose there is another review article that reads as below:
  - Review 2: **Priced at INR 75K for the low end model**, Apple iPhone boasts of an ultra slim device with an awesome camera. Apple’s CEO while showcasing the device at San Francisco, announced its availability on 12 countries including India. This is the best phone that one can flaunt if he can afford it!

Observations from the case studies

- A product review may have the aspects of our interest at various places in the text.
- Must have features: If a customer needs an item within a few days, he can't wait indefinitely. If he has a budget constraint, he can't buy the item even if it meets other requirements.
- If we find a sentence that implies that a must have feature can't be met, rest of the sentences don't contribute to the buying decision
- Hence the context plays a vital role in the classification decision.
- In a large text (say a 5 page product review) just the first sentence alone may contribute to the decision.
- While an RNN can carry the context, there are 2 limitations:
  - Due to the vanishing gradient problem, RNN's effectiveness is limited when it needs to go deep in to the context.
  - There is no finer control over which part of the context needs to be carried forward and how much of the past needs to be "forgotten"

Gated Recurrent Units (GRU)

- More sophisticated hidden units that support capturing long distance dependencies

- A model that allows error messages to flow through different units at different strengths based on the inputs

Standard RNN computes hidden layer at next time step directly: $h_t = f\left(W^{(hh)}h_{t-1} + W^{(hx)}x_t\right)$

GRU first computes an update **gate** (another layer) based on current input word vector and hidden state

$$z_t = \sigma\left(W^{(z)}x_t + U^{(z)}h_{t-1}\right)$$

Compute reset gate similarly but with different weights

$$r_t = \sigma\left(W^{(r)}x_t + U^{(r)}h_{t-1}\right)$$

Ref Socher's course

GRU Architecture

- A hidden unit in a neural network usually computes a simple activation function such as sigmoid or tanh or RelU

- GRU is a more sophisticated hidden unit that has 2 additional gates along with a cell state that generates the hidden activation

Update gate $z_t = \sigma\left(W^{(z)}x_t + U^{(z)}h_{t-1}\right)$

Reset gate $r_t = \sigma\left(W^{(r)}x_t + U^{(r)}h_{t-1}\right)$

New memory content: $\tilde{h}_t = \tanh\left(Wx_t + r_t \circ Uh_{t-1}\right)$

If reset gate unit is ~0, then this ignores previous memory and only stores the new word information

Final memory at time step combines current and previous time steps: $h_t = z_t \circ h_{t-1} + (1 - z_t) \circ \tilde{h}_t$

Ref Socher's course

Long Short Term Memory (LSTM)

- LSTM's are more complex hidden units that can learn long distance dependencies
- Most of the advanced RNN based applications use LSTMs or some variant of it such as GRUs
- The key element of a LSTM is the cell state that is created using a combination of part of current input and part of the context.
- A part of the cell state is exposed as hidden activation

The five key Architectural Elements of LSTM

LSTM elements

- Input Gate
- Forget Gate
- Cell
- Output Gate
- Hidden state output

$$i_t = \sigma\left(W^{(i)}x_t + U^{(i)}h_{t-1}\right)$$

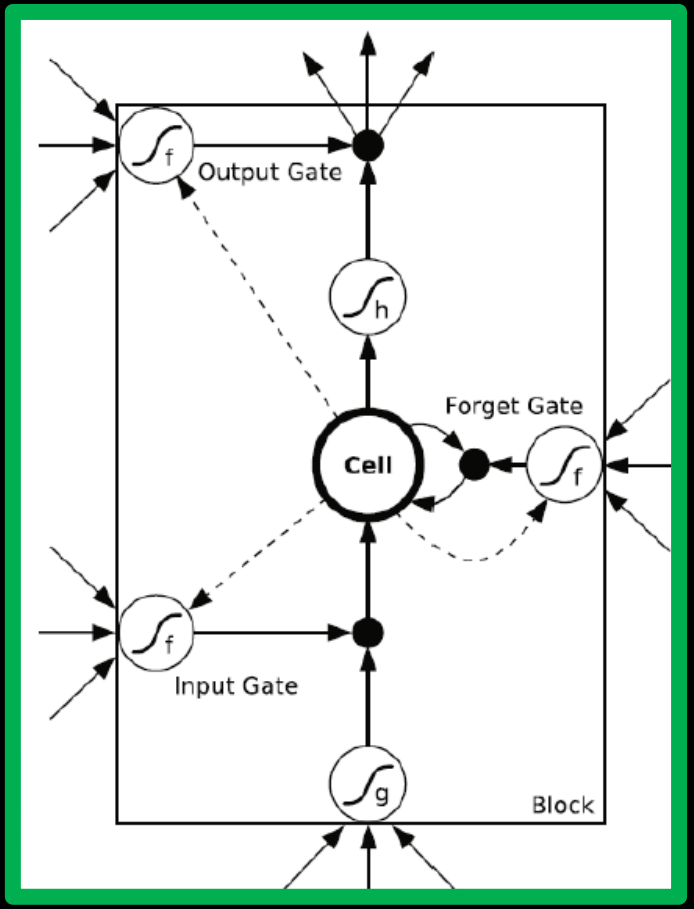
$$f_t = \sigma\left(W^{(f)}x_t + U^{(f)}h_{t-1}\right)$$

$$o_t = \sigma\left(W^{(o)}x_t + U^{(o)}h_{t-1}\right)$$

$$\tilde{c}_t = \tanh\left(W^{(c)}x_t + U^{(c)}h_{t-1}\right)$$

$$c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c}_t$$

$$h_t = o_t \circ \tanh(c_t)$$

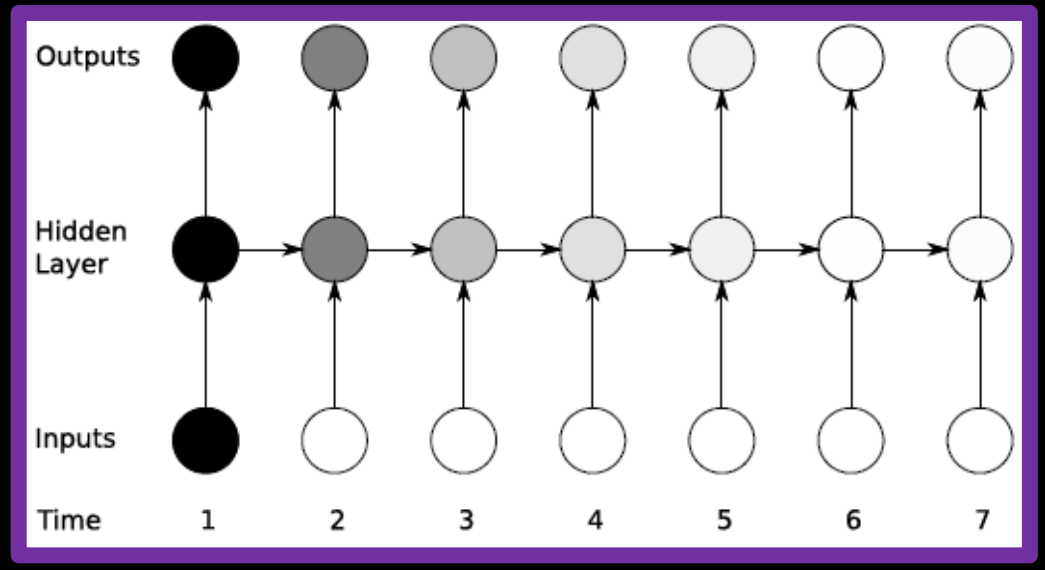


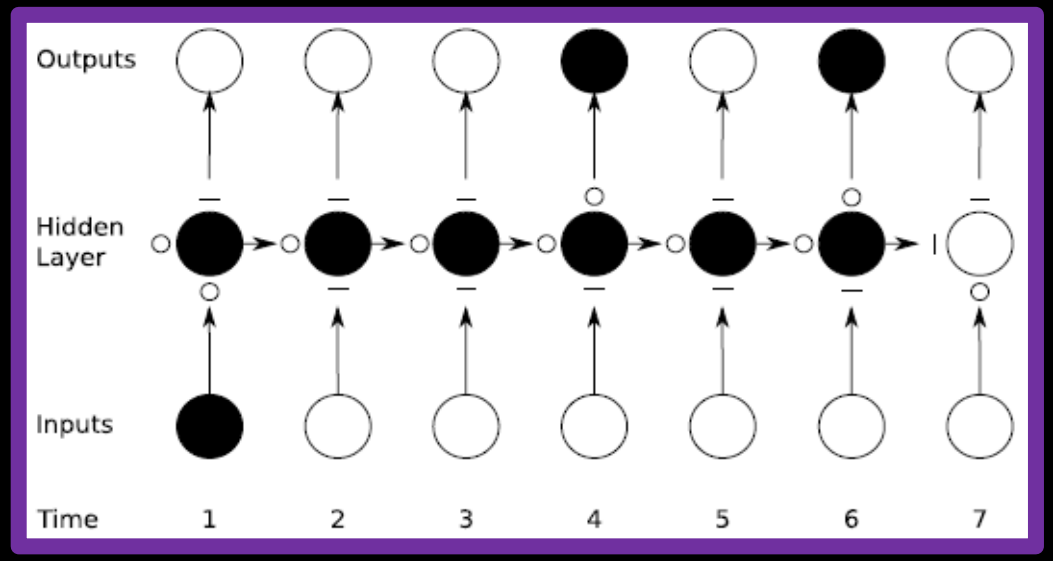
LSTM Architectural pieces

- Input Gate: Determines how much the current input matters
- Forget gate: Can be used to control how much of past memory we need to retain
-
- Output Gate: Determines how much of the computed cell state to expose to other layers outside LSTM
- Cell output: This is the internal output produced by LSTM
- Hidden state output: This is a fraction of cell output exposed to other layers. Output gate determines this proportion.

Effect of LSTM on sensitivity (Ref: Graves)

- In a simple RNN with sigmoid or tanh neuron units, the later output nodes of the network are less sensitive to the input at time t = 1. This happens due to the vanishing gradient problem
- An LSTM allows the preservation of gradients. The memory cell remembers the first input as long as the forget gate is open and the input gate is closed.
- The output gate provides finer control to switch the output layer on or off without altering the cell contents.





Implementing an LSTM: Notes for practitioners

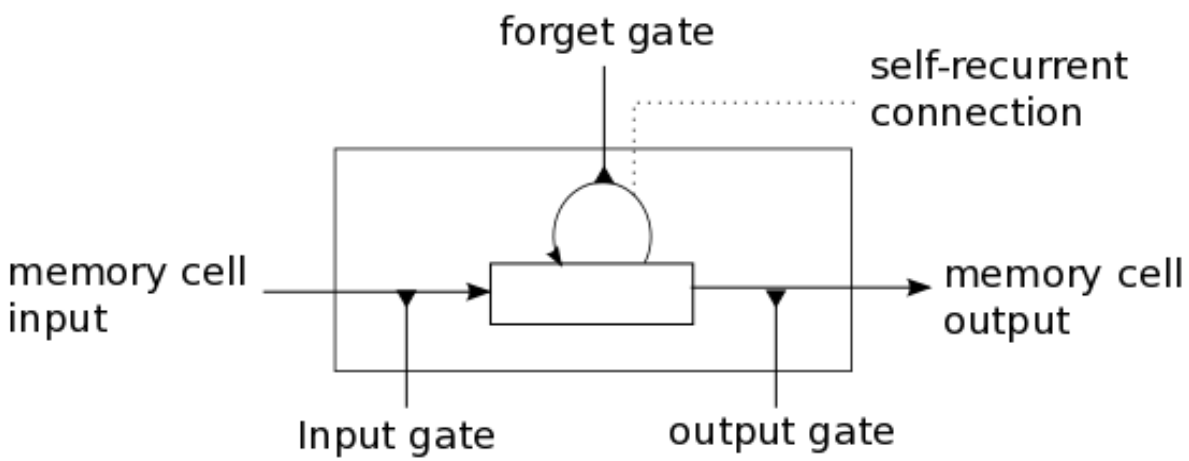
- Some points to take in to account while choosing an LSTM architecture:
  - LSTM has many variants compared to the original architecture
  - The LSTM initially didn't have forget gate, it was later added.
  - Most of the current implementations are based on the 3 gate LSTM model
  - Some variants adopt a simpler version. E.g. peephole connections omitted
  - Training is a bit complex compared to feedforward ANN
  - Many training techniques are reported.

- LSTMs can be stacked vertically to create a deep LSTM network

Long Short-Term Memory

Sepp Hochreiter
Fakultät für Informatik, Technische Universität München, 80290 München, Germany

Jürgen Schmidhuber
IDSIA, Corso Elvezia 36, 6900 Lugano, Switzerland



Beam Search in Seq2Seq models

Network Equations (Ref Alex Graves)

Input Gates

$$a_\iota^t = \sum_{i=1}^{I} w_{i\iota} x_i^t + \sum_{h=1}^{H} w_{h\iota} b_h^{t-1} + \sum_{c=1}^{C} w_{c\iota} s_c^{t-1}$$

$$b_\iota^t = f(a_\iota^t)$$

Forget Gates

$$a_\phi^t = \sum_{i=1}^{I} w_{i\phi} x_i^t + \sum_{h=1}^{H} w_{h\phi} b_h^{t-1} + \sum_{c=1}^{C} w_{c\phi} s_c^{t-1}$$

$$b_\phi^t = f(a_\phi^t)$$

Cells

$$a_c^t = \sum_{i=1}^{I} w_{ic} x_i^t + \sum_{h=1}^{H} w_{hc} b_h^{t-1}$$

$$s_c^t = b_\phi^t s_c^{t-1} + b_\iota^t g(a_c^t)$$

Output Gates

$$a_\omega^t = \sum_{i=1}^{I} w_{i\omega} x_i^t + \sum_{h=1}^{H} w_{h\omega} b_h^{t-1} + \sum_{c=1}^{C} w_{c\omega} s_c^t$$

$$b_\omega^t = f(a_\omega^t)$$

Cell Outputs

$$b_c^t = b_\omega^t h(s_c^t)$$

$$\epsilon_c^t \stackrel{\text{def}}{=} \frac{\partial \mathcal{L}}{\partial b_c^t} \qquad \epsilon_s^t \stackrel{\text{def}}{=} \frac{\partial \mathcal{L}}{\partial s_c^t}$$

Cell Outputs

$$\epsilon_c^t = \sum_{k=1}^{K} w_{ck}\delta_k^t + \sum_{h=1}^{H} w_{ch}\delta_h^{t+1}$$

Output Gates

$$\delta_\omega^t = f'(a_\omega^t)\sum_{c}^{C} h(s_c^t)\epsilon_c^t$$

States

$$\epsilon_s^t = b_\omega^t h'(s_c^t)\epsilon_c^t + b_\phi^{t+1}\epsilon_s^{t+1} + w_{c\iota}\delta_\iota^{t+1} + w_{c\phi}\delta_\phi^{t+1} + w_{c\omega}\delta_\omega^t$$

Cells

$$\delta_c^t = b_\iota^t g'(a_c^t)\epsilon_s^t$$

Forget Gates

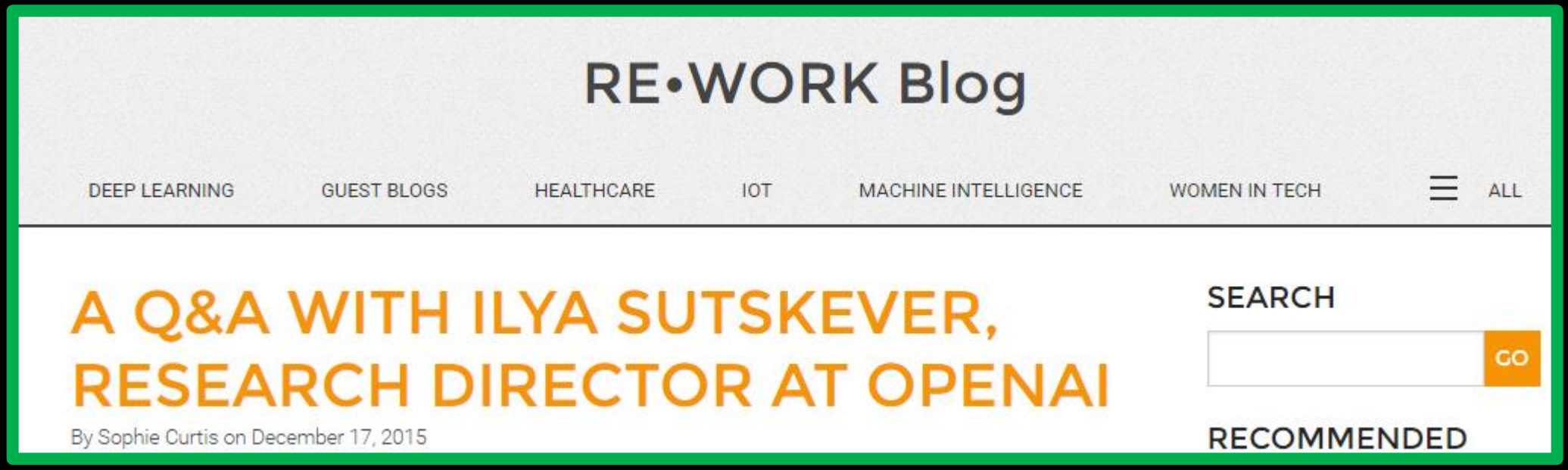
$$\delta_\phi^t = f'(a_\phi^t)\sum_{c=1}^{C} s_c^{t-1}\epsilon_s^t$$

Input Gates

$$\delta_\iota^t = f'(a_\iota^t)\sum_{c=1}^{C} g(a_c^t)\epsilon_s^t$$

Accessing Memory in Neural Networks

Attention Networks

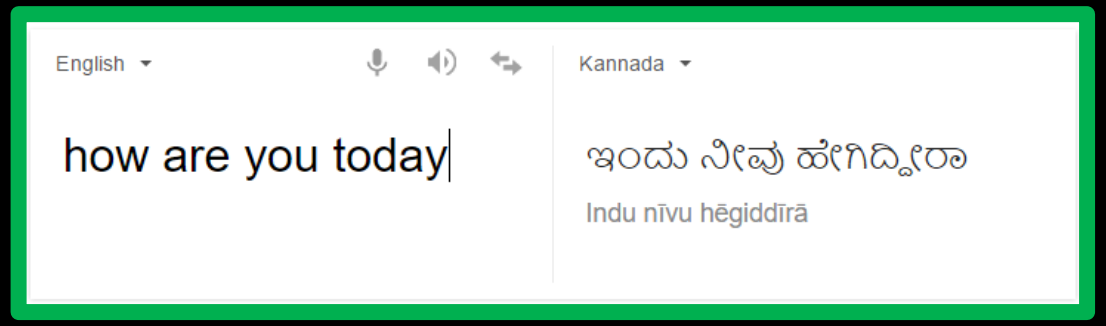


What advancements excite you most in the field?

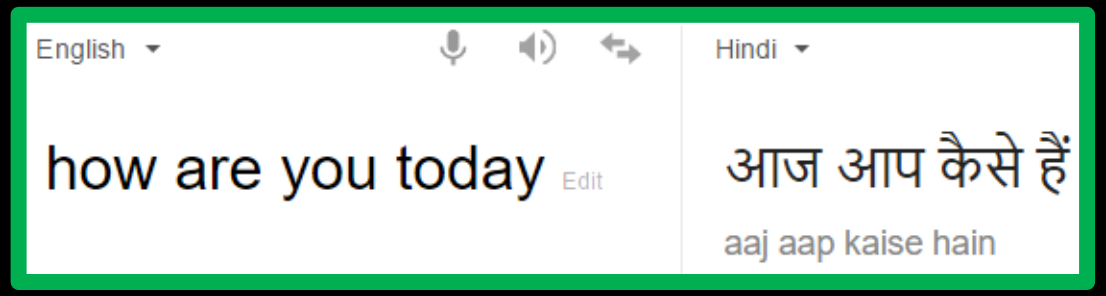
I am very excited by the recently introduced attention models, due to their simplicity and due to the fact that they work so well. Although these models are new, I have no doubt that they are here to stay, and that they will play a very important role in the future of deep learning.

What are attention networks?

- Suppose we have a sequence of input and we are to produce a sequence of outputs for this.
  - E.g. The inputs could be a sequence of words, could also be image segments

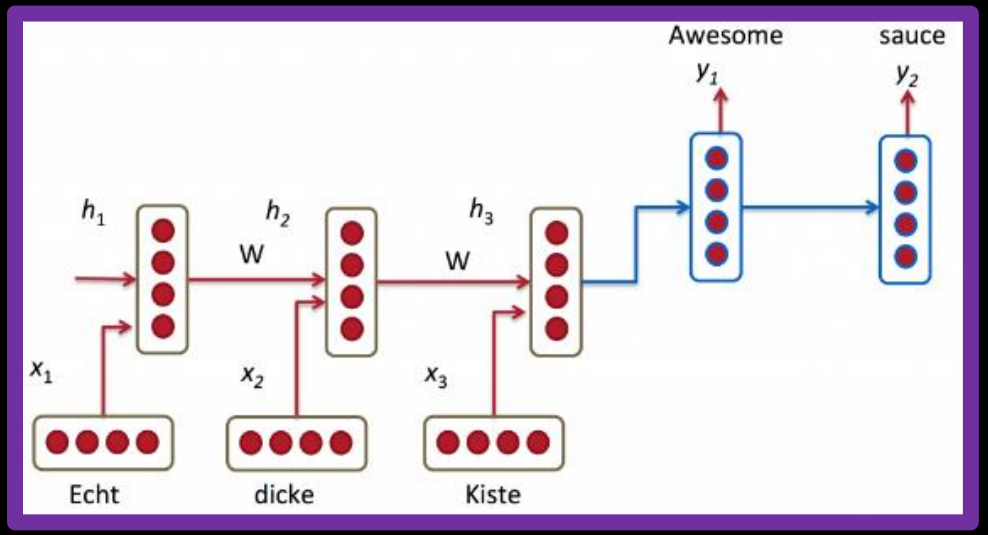


- Attention mechanisms focus on a narrow region of the input when making a classification decision for the output sequence.
  - E.g. In a machine translation, when outputting a word, the attention network might give higher weightage to certain words in the input.



Neural Machine Translation

- Traditional machine translation techniques require domain knowledge to build sophisticated features.
- In NMT, the source sentence is encoded as a single fixed length vector which is then translated in to the target language sentence.
- The encoding process is equivalent to learning a higher level meaning of the sentence, which can't be effectively captured using techniques like n-grams
- However, as the length of input sequence is arbitrary, it is not always effective to encode every one of them with same length vector.
- Attention mechanisms address the issue by varying the focus on input words as the output words are generated one at a time



Attention Mechanism

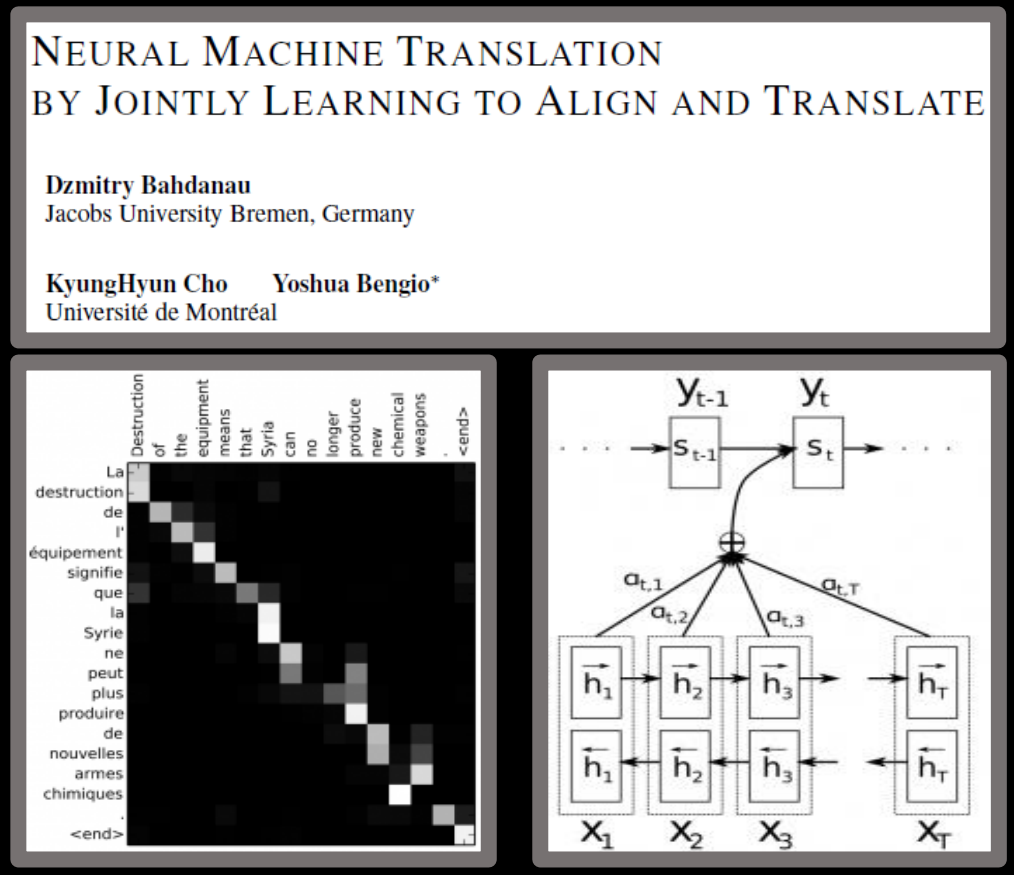
- When using the attention mechanism we make use of the hidden vectors computed at each time step of the encoder and hence the decoding doesn't need to depend on one single hidden vector

- The decoder "attends" to different parts of the source sentence at each step of the output generation.

- The model learns what to attend to based on the input sentence and what it has produced so far.
  - E.g. In the example of English to Kannada translation, the last word "today" translates in to the first word "indu" in Kannada. The attention model learns to focus on "today" when generating "indu"

Attention Mechanism

- X_i are the input words, processed by the encoder. Y_i are the outputs, the translated words
- The output Y_t depends not only on the previous hidden state value but also on the weighted combination of all input states
- α‘s are weights that determine how much of a given input word should be considered in computing the output at a given time step
- α‘s are normalized to 1 using a softmax



Machine Translation – Details

$$p(y_i|y_1, \ldots, y_{i-1}, x) = g(y_{i-1}, s_i, c_i)$$

$$s_i = f(s_{i-1}, y_{i-1}, c_i)$$

The context vector c_i depends on a sequence of annotations h_j

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij}\, h_j$$

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})}$$

Where: $e_{ij} = a(s_{i-1}, h_j)$, a is the alignment model learnt by a neural network

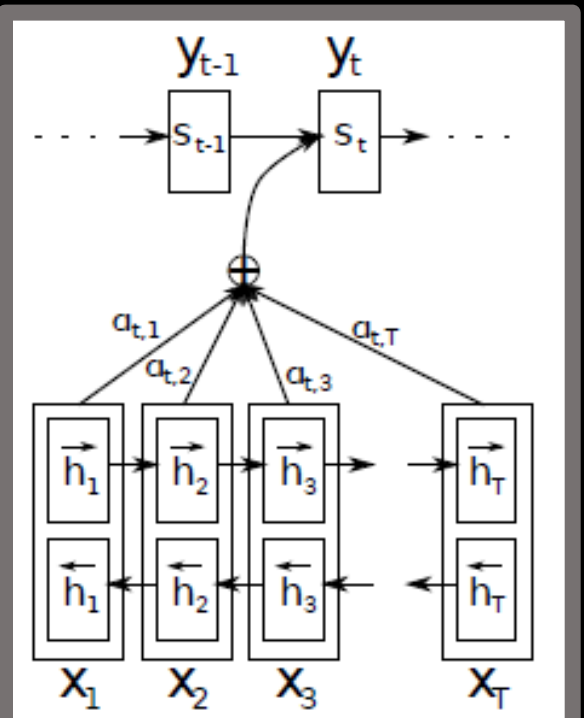
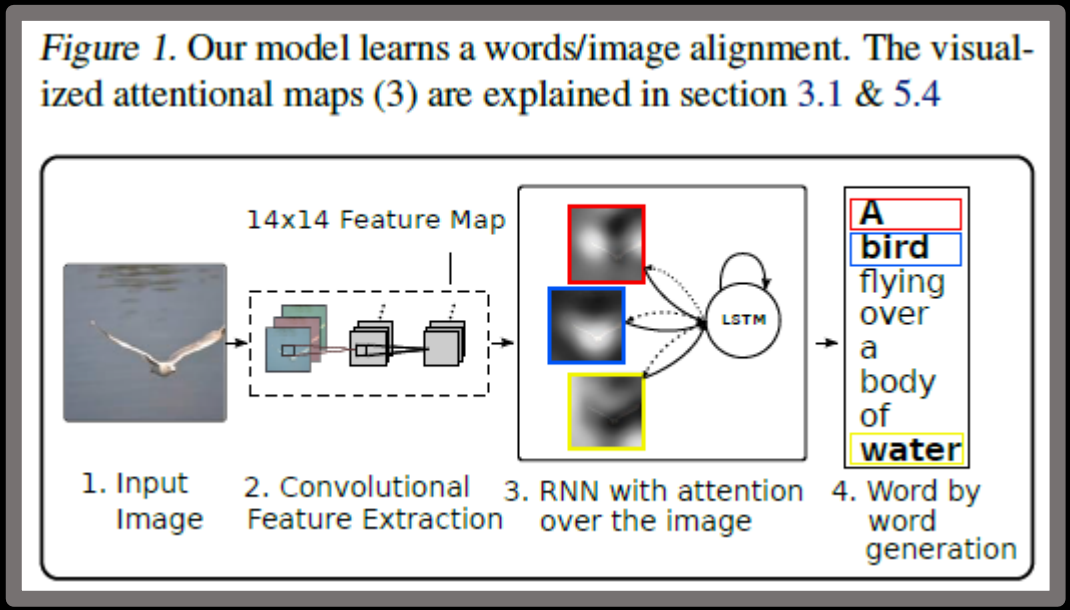
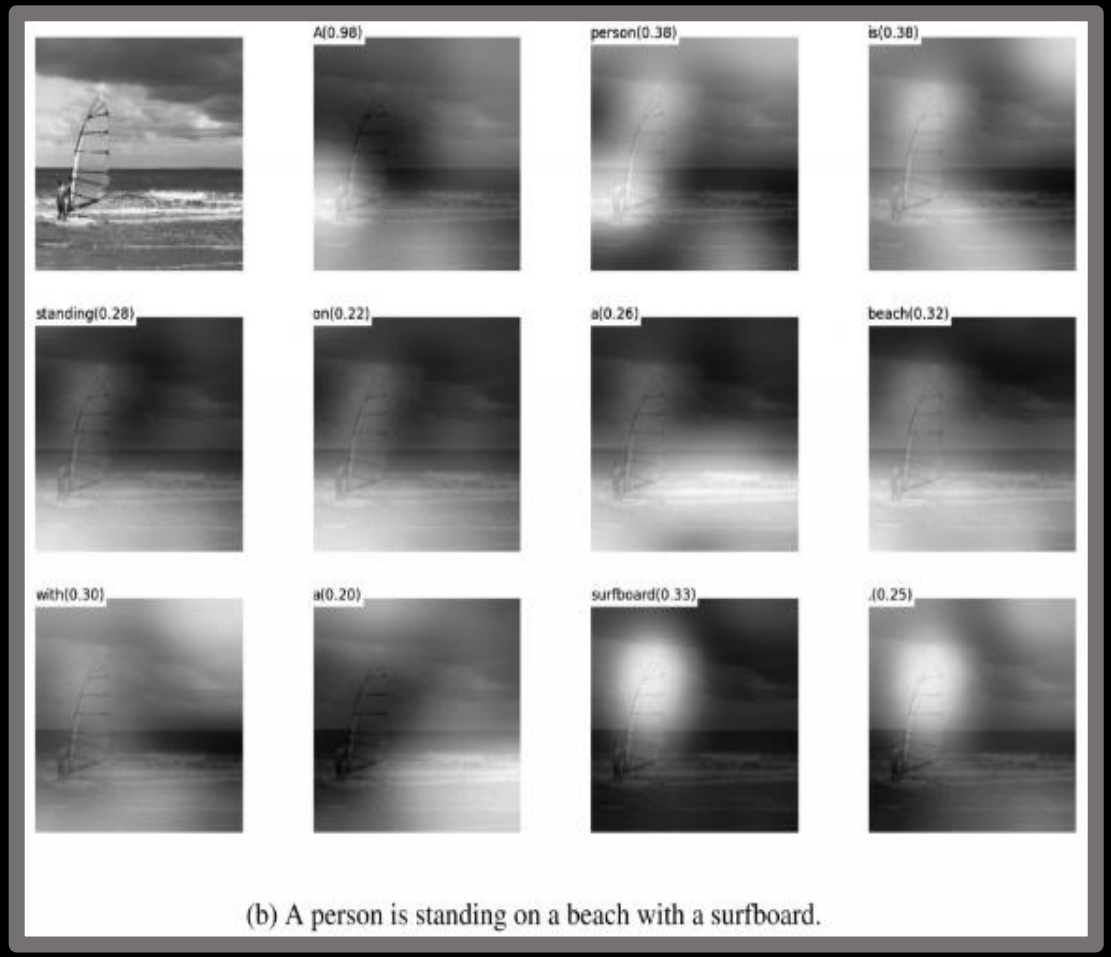


Figure 1: The graphical illustration of the proposed model trying to generate the t-th target word y_t given a source sentence $(x_1, x_2, \ldots, x_T)$.

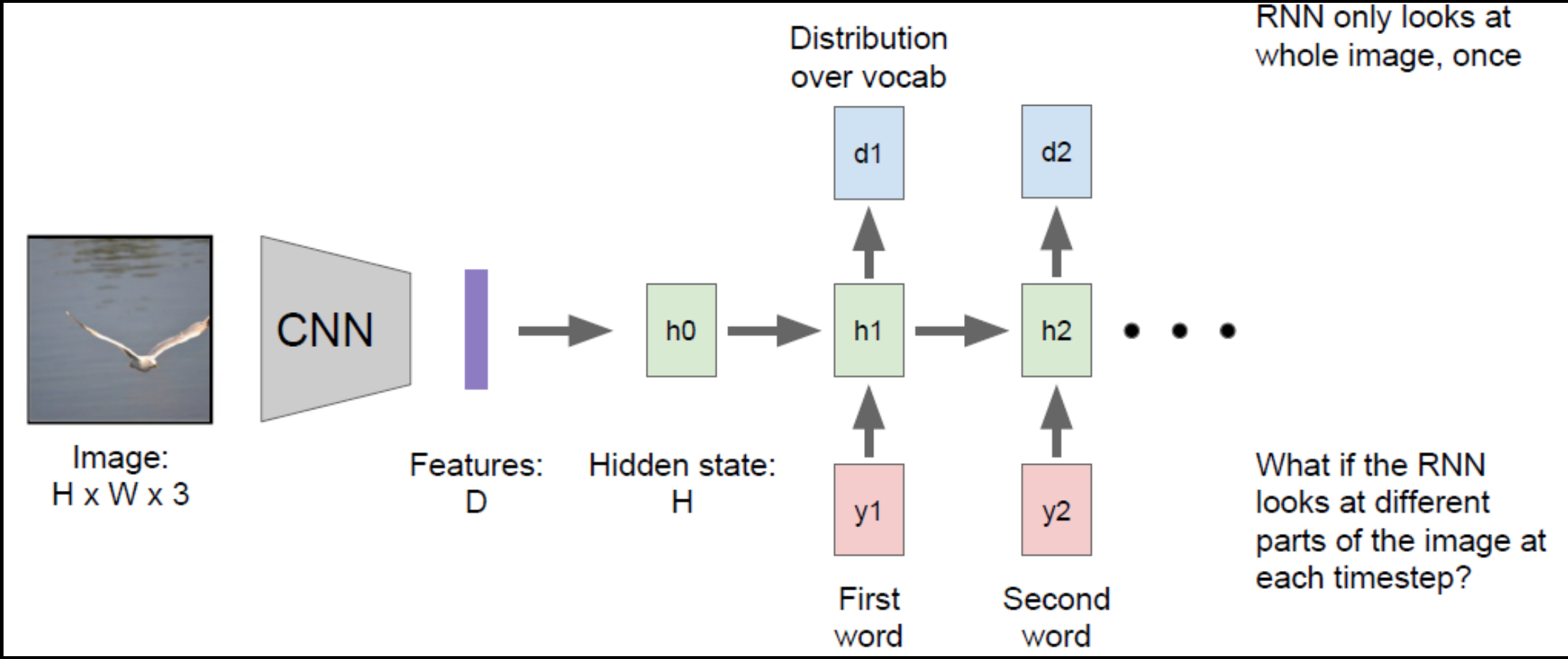
Attention Mechanism for Image Captioning



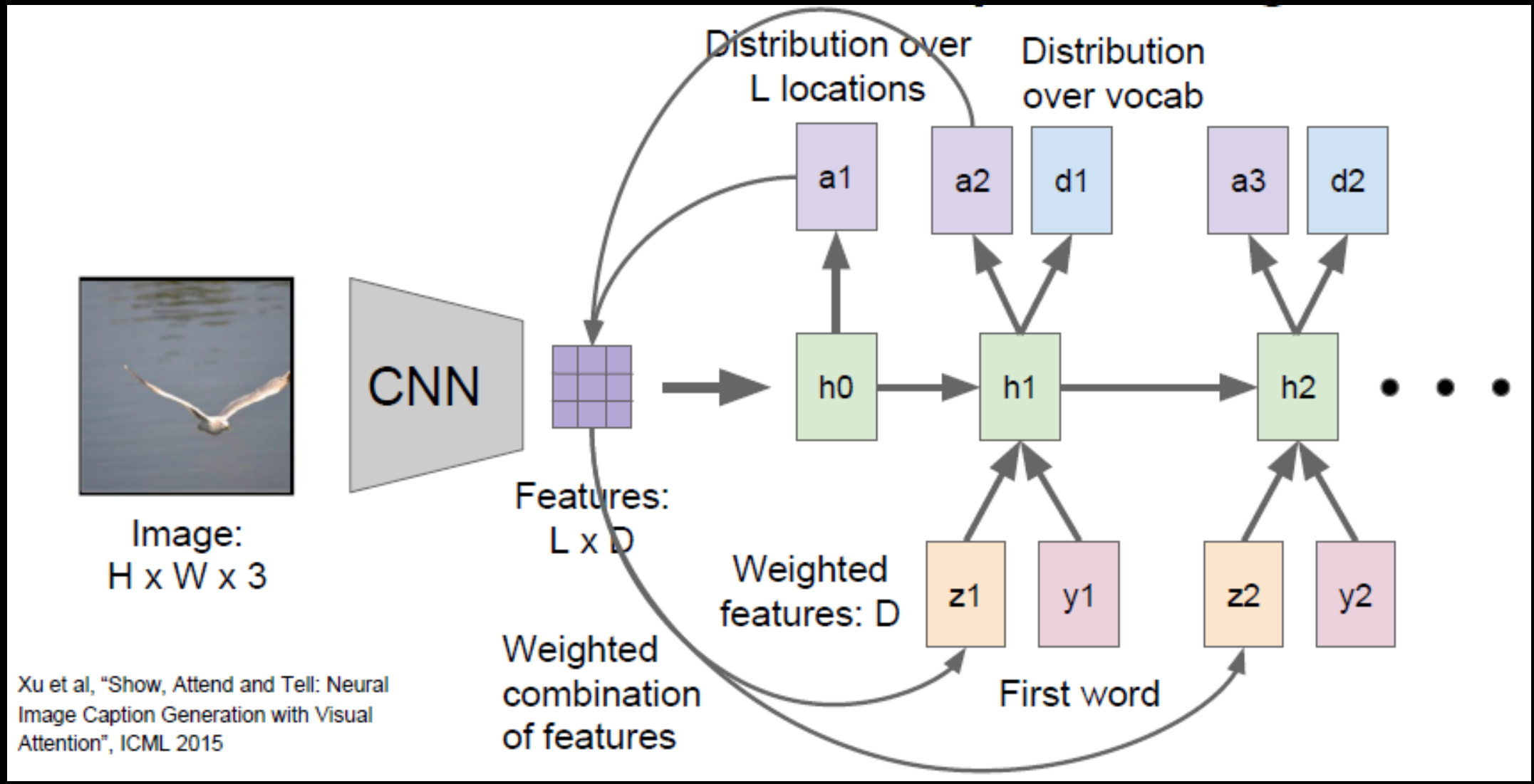
- CNN are used to encode an image and RNN is used with attention mechanism to decode this in to an image caption



Decoder for captioning (Without attention)



Attention based captioning



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

Soft vs Hard Attention

Image:
H x W x 3

CNN

a	b
c	d

Grid of features
(Each D-dimensional)

Context vector z
(D-dimensional)

From RNN:

p_a	p_b
p_c	p_d

Distribution over grid locations
$p_a + p_b + p_c + p_c = 1$

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

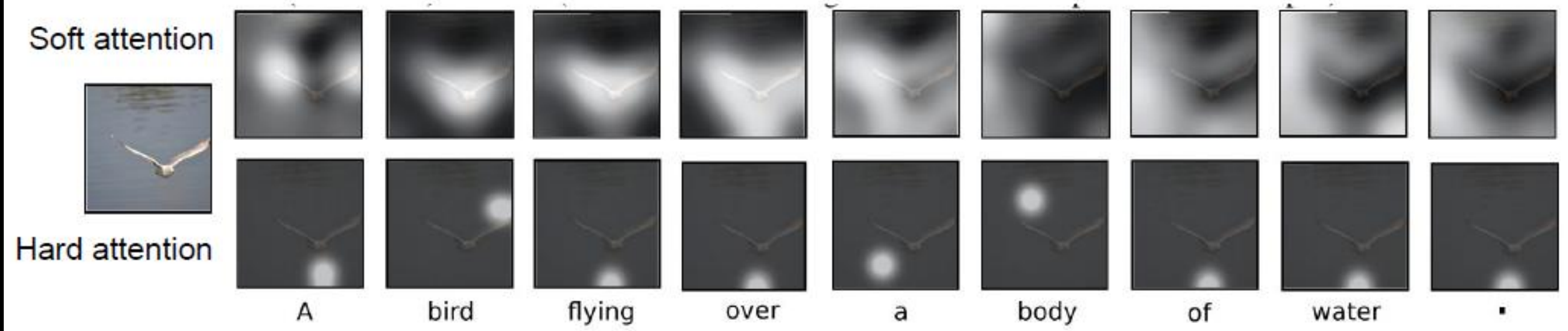
Soft attention:
Summarize ALL locations
$z = p_a a + p_b b + p_c c + p_d d$

Derivative dz/dp is nice!
Train with gradient descent

Hard attention:
Sample ONE location according to p, z = that vector

With argmax, dz/dp is zero almost everywhere …
Can't use gradient descent; need reinforcement learning

Slide Credit A Karpathy CS 231n



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

A woman is throwing a frisbee in a park.

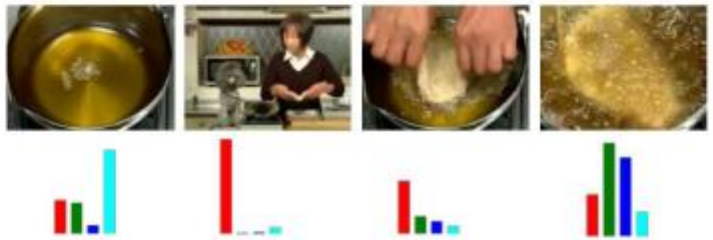
A dog is standing on a hardwood floor.

A stop sign is on a road with a mountain in the background.

Soft Attention for Everything!

Machine Translation, attention over input:
- Luong et al, "Effective Approaches to Attention-based Neural Machine Translation," EMNLP 2015

Speech recognition, attention over input sounds:
- Chan et al, "Listen, Attend, and Spell", arXiv 2015
- Chorowski et al, "Attention-based models for Speech Recognition", NIPS 2015



+Local+Global: Someone is frying a fish in a pot

Video captioning, attention over input frames:
- Yao et al, "Describing Videos by Exploiting Temporal Structure", ICCV 2015

Image, question to answer, attention over image:
- Xu and Saenko, "Ask, Attend and Answer: Exploring Question-Guided Spatial Attention for Visual Question Answering", arXiv 2015
- Zhu et al, "Visual7W: Grounded Question Answering in Images", arXiv 2015

Accessing Memory: Motivation

- The cell state in LSTMs can be considered to be some form of memory and the model parameters allow us to perform different operations on this memory

- However, the amount of content it can hold is quite limited compared to what we can do with external RAMs
  - E.g. The simple task of reading in a sequence and outputting the same sequence by itself is a difficult task, particularly when the sequence length is long
  - Imagine a situation where the classifier needs to watch a movie and answer questions on it. Without support for a powerful memory such applications are not possible

Case Study: Facebook (Refer: Weston's Paper)

Figure 1: Example "story" statements, questions and answers generated by a simple simulation. Answering the question about the location of the milk requires comprehension of the actions "picked up" and "left". The questions also require comprehension of the time elements of the story, e.g., to answer "where was Joe before the office?".

Joe went to the kitchen. Fred went to the kitchen. Joe picked up the milk.
Joe travelled to the office. Joe left the milk. Joe went to the bathroom.
Where is the milk now? A: office
Where is Joe? A: bathroom
Where was Joe before the office? A: kitchen

Components of Memory Networks

- I: (input feature map) – converts the input to the internal feature representation.
- G: (generalization) – updates old memories given the new input. We call this generalization as there is an opportunity for the network to compress and generalize its memories at this stage for some intended future use.
- O: (output feature map) – produces a new output (in the feature representation space), given the new input and the current memory state.
- R: (response) – converts the output into the response format desired. For example, a textual response or an action

Memory Network: Flow of the model

- Suppose the input is a character or word or sentence representation. It can be image or audio MFCC vectors based on the application. Let this be x
- Convert x to an internal feature representation I(x)
- Update memories m_i given the new input: $m_i = G(m_i, I(x), m)\ \forall i$
- Compute output features o given the new input and the memory:
$$o = O(I(x), m)$$
- Decode the output features o to get the final response: $r = R(O)$

Memory Networks – components design

The components I, G, O, R can be designed using any machine learning classifier: SVM, Neural Networks, Decision Trees, etc

I component: Component I can make use of standard pre-processing, e.g., parsing, coreference and entity resolution for text inputs. It could also encode the input into an internal feature representation, e.g., convert from text to a sparse or dense feature vector.

G component: The simplest form of G is to store $I(x)$ in a "slot" in the memory:

$$\mathbf{m}_{H(x)} = I(x), \tag{1}$$

where $H(.)$ is a function selecting the slot. That is, G updates the index $H(x)$ of $\mathbf{m}$, but all other parts of the memory remain untouched. More sophisticated variants of G could go back and update earlier stored memories (potentially, all memories) based on the new evidence from the current input x. If the input is at the character or word level one could group inputs (i.e., by segmenting them into chunks) and store each chunk in a memory slot.

If the memory is huge (e.g., consider all of Freebase or Wikipedia) one needs to organize the memories. This can be achieved with the slot choosing function H just described: for example, it could be designed, or trained, to store memories by entity or topic. Consequently, for efficiency at scale, G (and O) need not operate on all memories: they can operate on only a retrieved subset of candidates (only operating on memories that are on the right topic). We explore a simple variant of this in our experiments.

If the memory becomes full, a procedure for "forgetting" could also be implemented by H as it chooses which memory is replaced, e.g., H could score the utility of each memory, and overwrite the least useful. We have not explored this experimentally yet.

O and R components: The O component is typically responsible for reading from memory and performing inference, e.g., calculating what are the relevant memories to perform a good response. The R component then produces the final response given O. For example in a question answering setup O finds relevant memories, and then R produces the actual wording of the answer, e.g., R could be an RNN that is conditioned on the output of O. Our hypothesis is that without conditioning on such memories, such an RNN will perform poorly.

End To End Memory Networks