Recurrent Neural Networks Part 1

Anantharaman Narayana Iyer

Narayana dot Anantharaman at gmail dot com

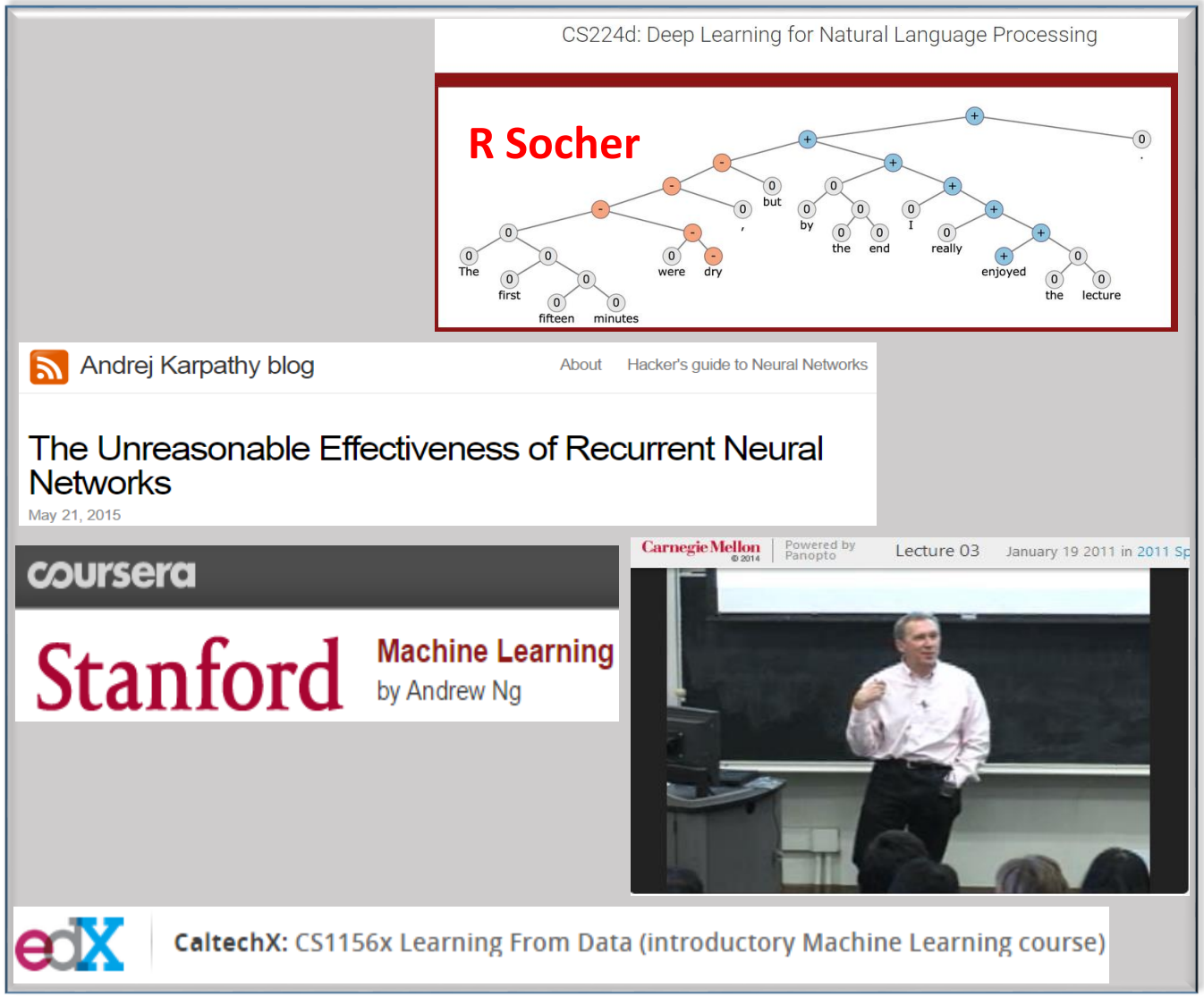
18 Sep 2015

Objective

- Overview of Neural Networks
- Recurrent Neural Networks (RNN)
- Bidirectional Recurrent Neural Networks (BRNN)
- Differences between Recursive and Recurrent Neural Networks
- Challenges in implementing RNN: Vanishing Gradient Problem
- Gated Recurrent Units (GRUs)
- Long Short Term Memory (LSTM)
- Applications

References (Abridged list)

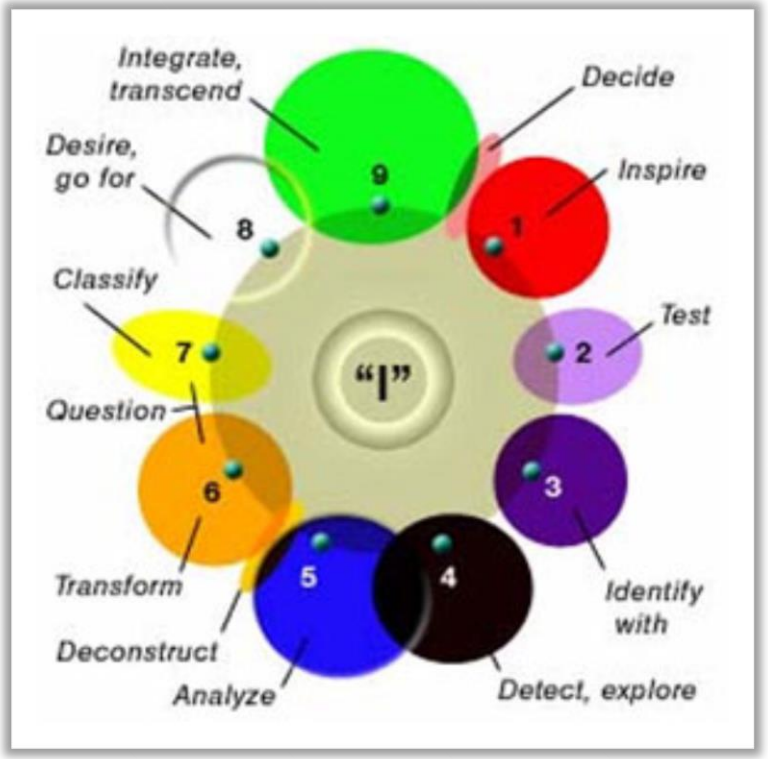
- Machine Learning, T Mitchell
- MOOC Courses offered by Prof Andrew Ng, Prof Yaser Mustafa, Geoff Hinton
- CMU Videos Prof T Mitchell
- Alex Graves: Supervised Sequence Labelling with Recurrent Neural Networks
- Andrej Karpathy's Blogs
- Stanford course CS224d: Socher
- Recurrent Neural Networks based Language Models: Mikolov etal
- Annotating Expressions of Opinions and Emotions in Language: Wiebe etal



Human Cognition

- Most Common human cognitive tasks (such as understanding and speaking natural language, recognizing objects etc) are highly non linear.
- Human cognition tasks are often hierarchical
- Though our brain receives low level sensory inputs as impulses, we process the input as a whole, recognizing several patterns as opposed to looking at micro level data
- Humans learn continuously often unaided or unsupervised
- For the same object or pattern that we recognize we see them in different perspectives.
  - E.g. We may know “home” and “residence” as two separate words but we can interpret them in different contexts. Home maker versus residence address.

What does it take to build an autonomous car that can drive itself in Bangalore traffic?



Quick recap of last lecture

- ML attempts to approximate real world applications by mathematical models
- The underlying process behind the given real world application (that we are trying to model) is called the unknown target function
- Linear models approximate the real world using a linear function.
- Most of the real world applications are non-linear and are hierarchical
- Artificial Neural networks (ANN) are non linear models and are effective for certain class of applications.
- Each hidden layer represents a particular level of abstraction
- ANNs are commonly trained using backpropagation algorithms
- The model parameters are tunable knobs that determine the output of the machine and signify the degrees of freedom
- More the parameters, more easily we can fit the training data but impacts the generalization. Regularization keeps the model parameters under check
- Traditional ANNs with a large number of hidden layers are hard to train: Problems of local minima and vanishing/exploding gradients
- Deep learning techniques are breakthroughs that enable realization of deep architectures
- Recurrent Neural Networks (RNN), Recursive Neural Networks and Convolutional Neural Networks are specializations of the ANN architecture to handle different nature of problems.
  - For instance RNNs are effective for predicting time series problems
- For a brief 5 slide refresher on DNNs see: http://www.slideshare.net/ananth/deep-learningprimer-7june2014

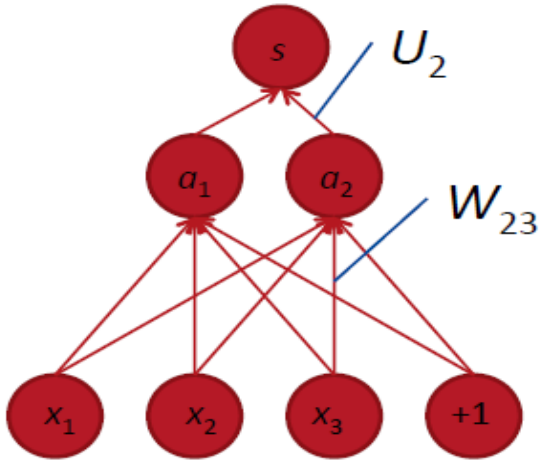
Non linear Models: Neural Networks

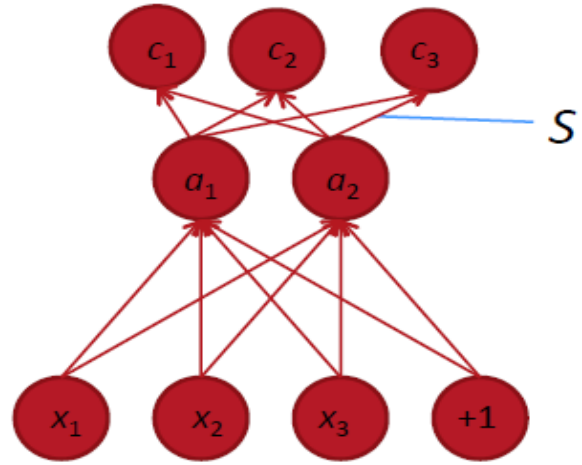
- Motivation
  - A large number of classification tasks involve inherently highly non linear target functions – for example, face recognition
  - Though we can transform the input vector in to a non linear form and perform classification with linear models, the model becomes very complex quickly.
    - For example:
      - Consider a 10 dimensional input vector that needs to be transformed in to a polynomial with degree 3. $O(n^3)$
      - Consider the problem of looking at the image of a building and identifying it (say: 100 by 100 pixels)
  - Over fitting problems are common when we train more complex models
- Illustration (on black board)
  - Boolean functions AND, OR can be effectively modelled by Linear Models
  - A single logistic regression unit can't model more complex Boolean functions such as XOR
- Cascading logistic regression units can classify complex Boolean target functions effectively
  - It is shown that with 2 layers of logistic regression units, one can model many complex Boolean expressions effectively

Neural Networks

(Fig: courtesy R Socher)

Neural Networks can be built for different input, output types.





- Outputs can be:
  - Linear, single output (Linear)
  - Linear, multiple outputs (Linear)
  - Single output binary (Logistic)
  - Multi output binary (Logistic)
  - 1 of k Multinomial output (Softmax)

- Inputs can be:
  - A scalar number
  - Vector of Real numbers
  - Vector of Binary

Goal of training: Given the training data (inputs, targets) and the architecture, determine the model parameters.

Model Parameters for a 3 layer network:

- Weight matrix from input layer to the hidden (W_{jk})
- Weight matrix from hidden layer to the output (W_{kj})
- Bias terms for hidden layer
- Bias terms for output layer

Our strategy will be:

- Compute the error at the output
- Determine the contribution of each parameter to the error by taking the differential of error wrt the parameter
- Update the parameter commensurate with the error it contributed.

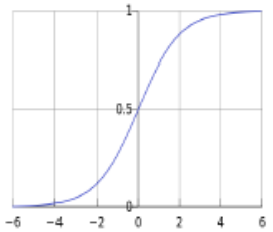
Design Choices

- When building a neural network, the designer would choose the following hyper parameters and non linearities based on the application characteristics:
  - Number of hidden layers
  - Number of hidden units in each layer
  - Learning rate
  - Regularization coefft
  - Number of outputs
  - Type of output (linear, logistic, softmax)
  - Choice of Non linearity at the output layer and hidden layer (See next slide)
  - Input representation and dimensionality

Commonly used non linearities (fig: courtesy Socher)

logistic ("sigmoid")

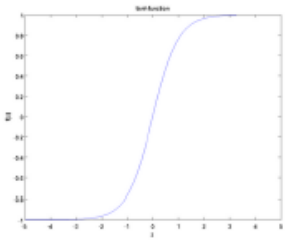
$$f(z) = \frac{1}{1+\exp(-z)}.$$



$$f'(z) = f(z)(1-f(z))$$

tanh

$$f(z) = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}},$$

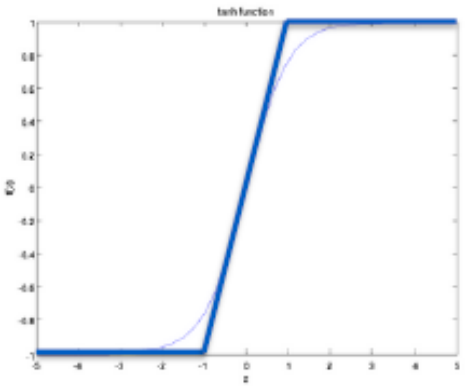


$$f'(z) = 1 - f(z)^2$$

$$\tanh(z) = 2\text{logistic}(2z) - 1$$

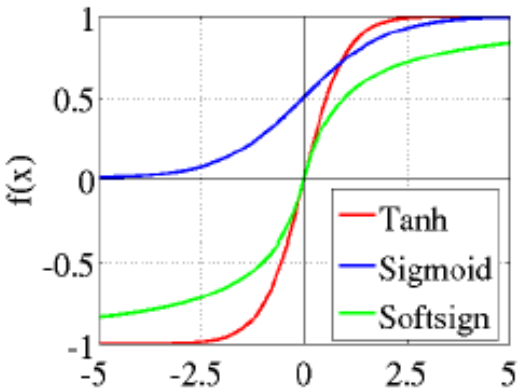
hard tanh

$$\text{HardTanh}(x) = \begin{cases} -1 & \text{if } x < -1 \\ x & \text{if } -1 <= x <= 1 \\ 1 & \text{if } x > 1 \end{cases}$$



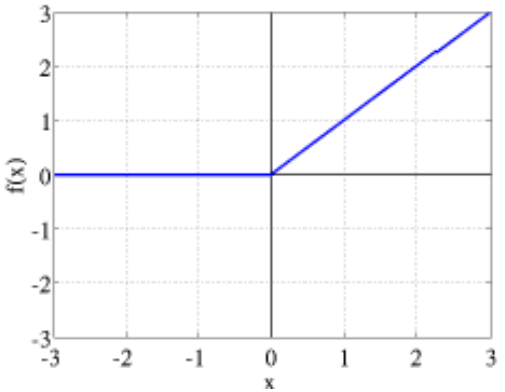
soft sign

$$\text{softsign}(z) = \frac{a}{1+|a|}$$



rectified linear (ReLu)

$$\text{rect}(z) = \max(z, 0)$$



Objective Functions and gradients (derivation of gradient on the board)

- Linear – Mean squared error
  - $E(w) = \frac{1}{2N}\sum_1^N (t_n - y_n)^2$
- Logistic with binary classifications: Cross Entropy Error
- Logistic with k outputs: k > 2: Cross Entropy Error
- Softmax: 1 of K multinomial classification: Cross Entropy Error, minimize NLL
- In all the above cases we can show that the gradient is: $(y_k - t_k)$ where y_k is the predicted output for the output unit k and t_k is the corresponding target

High Level Backpropagation Algorithm

- Apply the input vector to the network and forward propagate. This will yield the activations for hidden layer(s) and the output layer
  - $net_j = \sum_i w_{ji} z_i$,
  - $z_j = h(net_j)$ where h is your choice of non linearity. Usually it is sigmoid or tanh. Rectified Linear Unit (RelU) is also used.
- Evaluate the error δ_k for all the output units

  $\delta_k = o_k - t_k$ where o_k is the output produced by the model and t_k is the target provided in the training dataset
- Backpropagate the δ's to obtain δ_j for each hidden unit j

$$\delta_j = h'(z_j) \sum_k w_{kj} \delta_k$$

- Evaluate the required derivatives

$$\frac{\partial E}{\partial W_{ji}} = \delta_j z_i$$

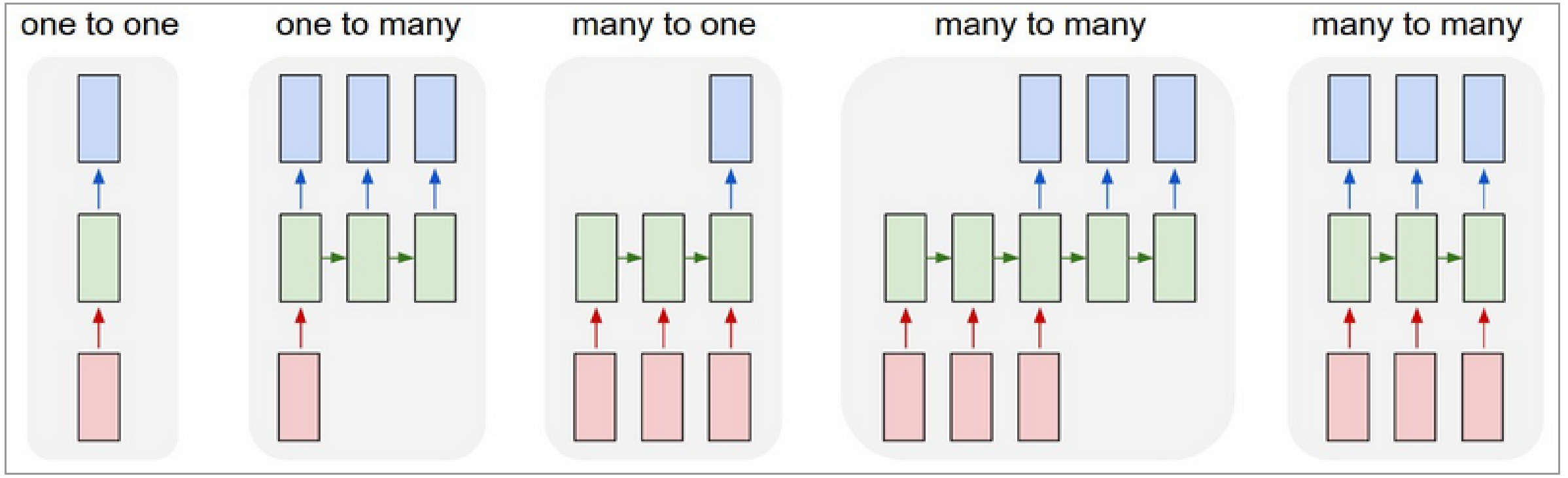
Backpropagation Training

- To be explained on the board

RNN – Some toy applications to evaluate the system

- Often times some toy applications, even if they are contrived, serve the following purposes:
  - Test the correctness of the implementation of the model
  - Compare the performance of the new model with respect to the old ones
- Example applications for verifying the performance of RNN:
  - Arithmetic progression (will be demo'd now)
  - Process an input of the form: $a^n b^j$ and return true if n = j
  - Count the number of words in a sequence ignoring the words that are enclosed in parenthesis
  - Perform XOR of bits of a sequence up to a time step t

RNN (Courtesy: A Karpathy's Blog)



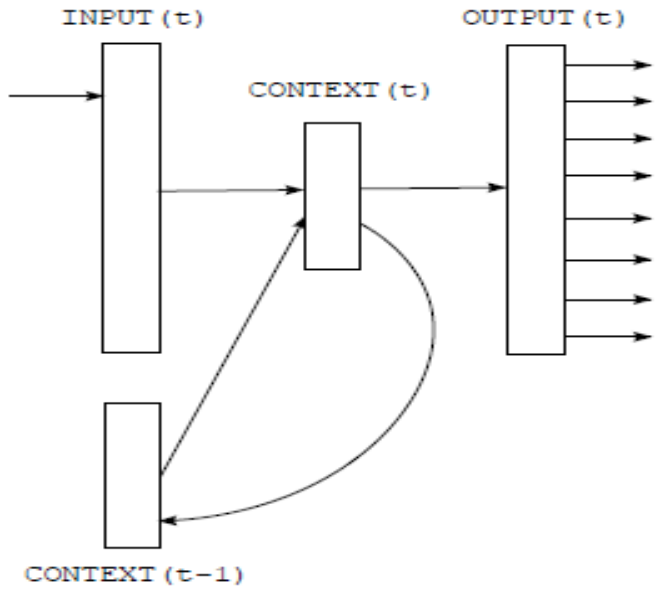
Each rectangle is a vector and arrows represent functions (e.g. matrix multiply). Input vectors are in red, output vectors are in blue and green vectors hold the RNN's state (more on this soon). From left to right: **(1)** Vanilla mode of processing without RNN, from fixed-sized input to fixed-sized 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.

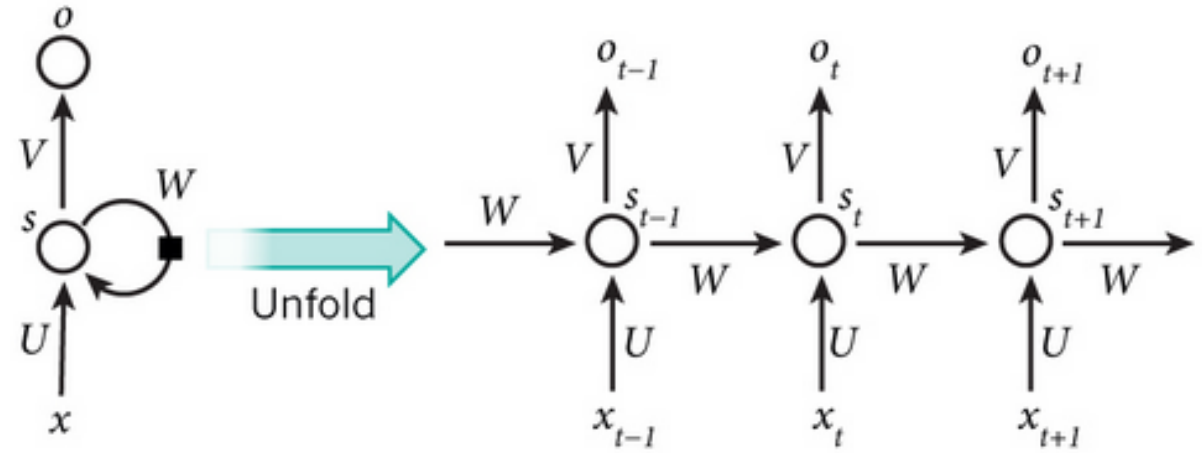
RNN Model with sigmoid nonlinearity and softmax output

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

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

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





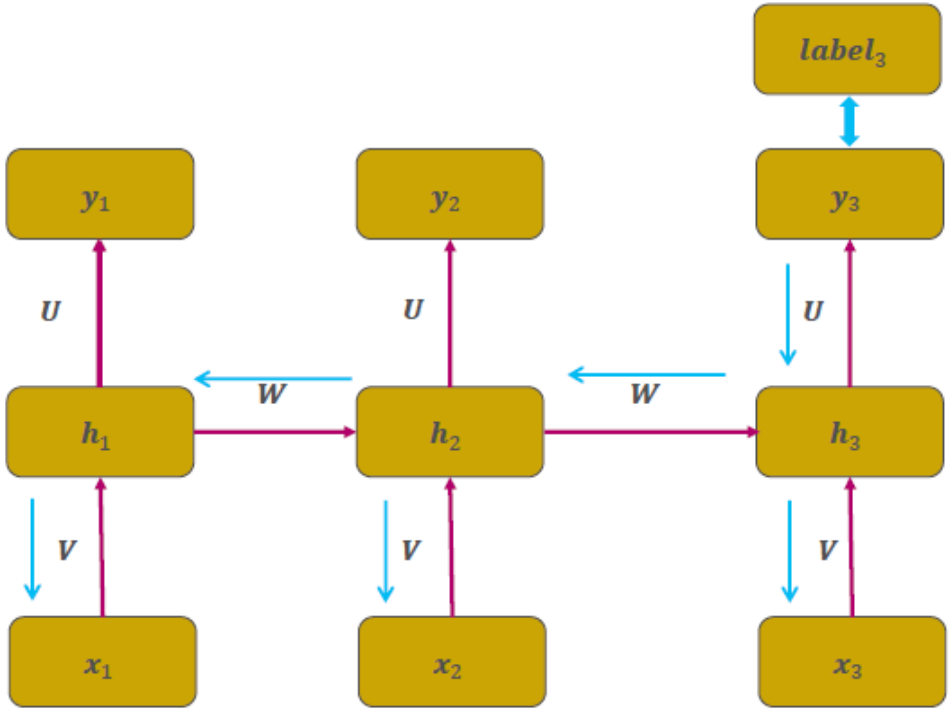
A recurrent neural network and the unfolding in time of the computation involved in its forward computation. Source: Nature

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



at time $t = 3$

1. Forward propagation
2. Generate output
3. Calculate error
4. Back propagation
5. Back prop. through time

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

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)
```

Applications

- Language Model (Mikolov etal)
  - Input at a time t is the corresponding word vector
  - Output is the predicted next word
- Language translation
- Slot filling (see next slide)
- Character LM (Andrej Karpathy)
- Image captioning and description
- Speech recognition
- Question Answering Systems (We are doing a special topic project on this)
- Semantic Role Labeling (We are doing a special topic project on this)
- NER (demo done last week!)
- And many more sequence based applications

Semantic Slot Filling Application Example

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

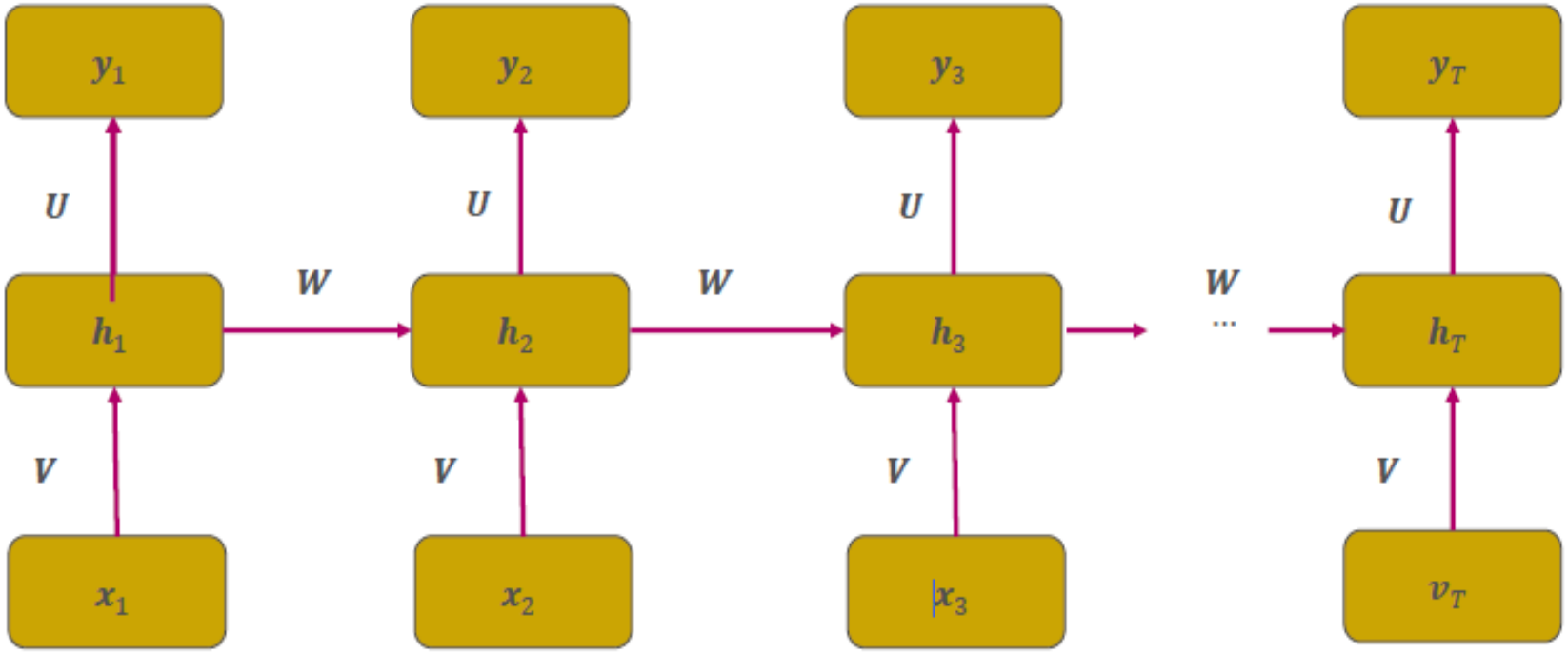
One possible way to cast this problem is to treat this as a slot filling exercise.

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

- Example: Machine Translation
- Use 2 RNN’s, one for encoding and the other decoding
- The activations of the final stage of the encoder is fed to the decoder
- This is useful when the output sequence is of variable length and if the entire input sequence can be processed before generating the output

Vanishing Gradients

- Refer Socher's presentation

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 (topic of next class!)

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

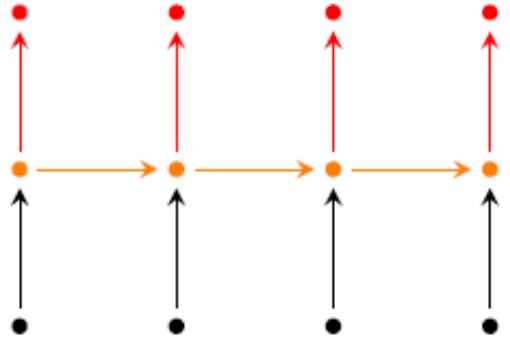
Problem Statement: Ref Irsoy and Cardie 2014

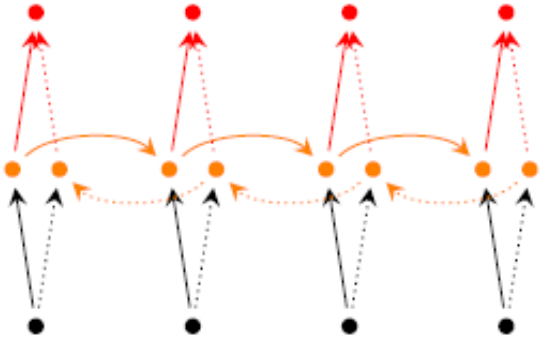
- Given a sentence, classify each word in to one of the tags: {O, B-ESE, I-ESE, B-DSE, I-DSE}
- Definitions
  - Direct Subjective Expressions (DSE): explicit mentions of private states or speech events expressing private states
  - Expressive Subjective Expressions (ESE): Expressions that indicate sentiment,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

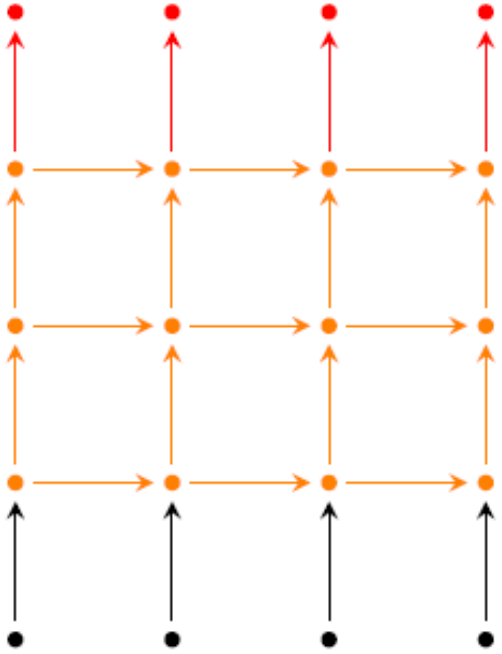
Fine-grained opinion analysis aims to detect the subjective expressions in a text (e.g. "hate") and to characterize their intensity (e.g. strong) and sentiment (e.g. negative) as well as to identify the opinion holder (the entity expressing the opinion) and the target, or topic, of the opinion (i.e. 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.

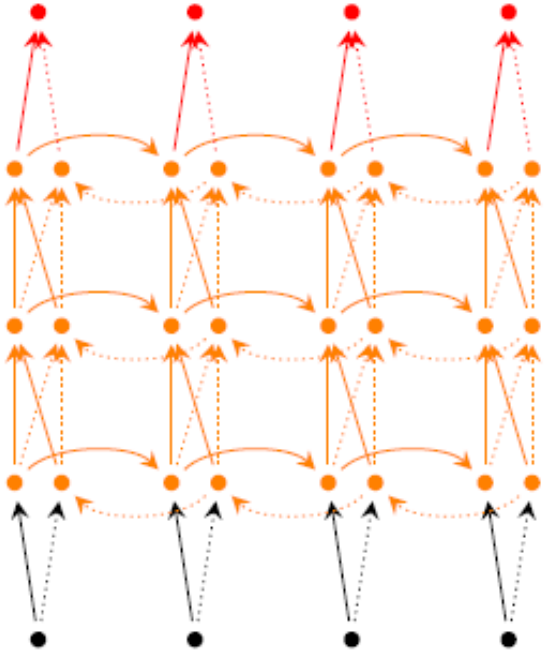
Bidirectional RNN Model

- 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)



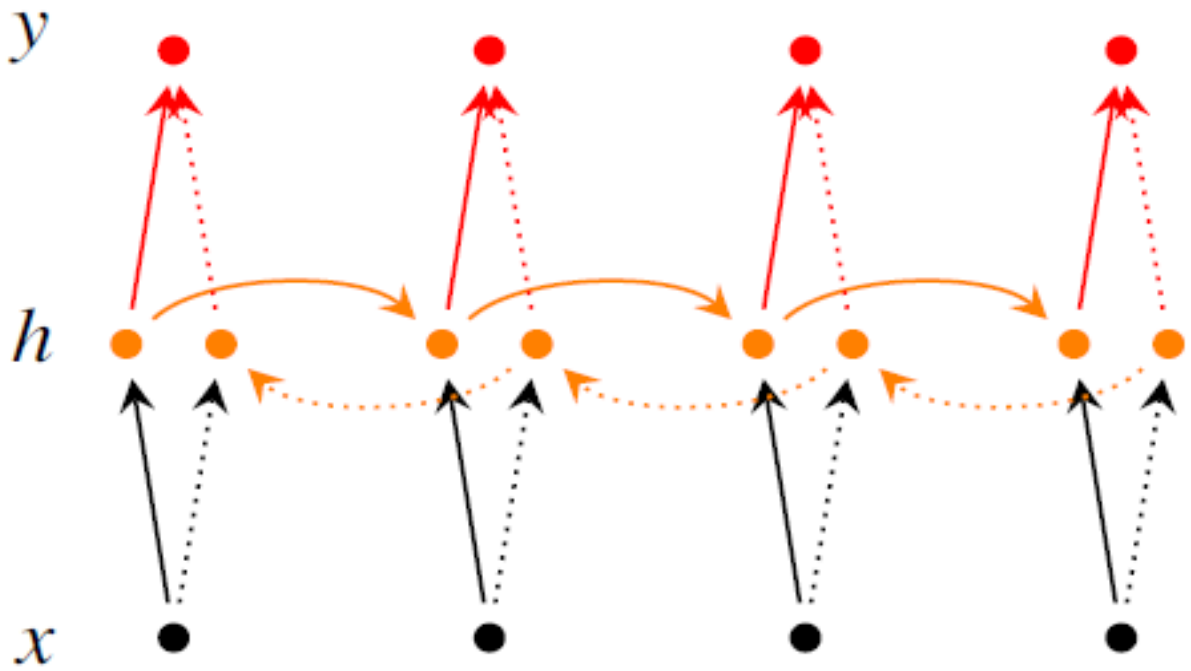






The model with 1 hidden layer

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



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

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

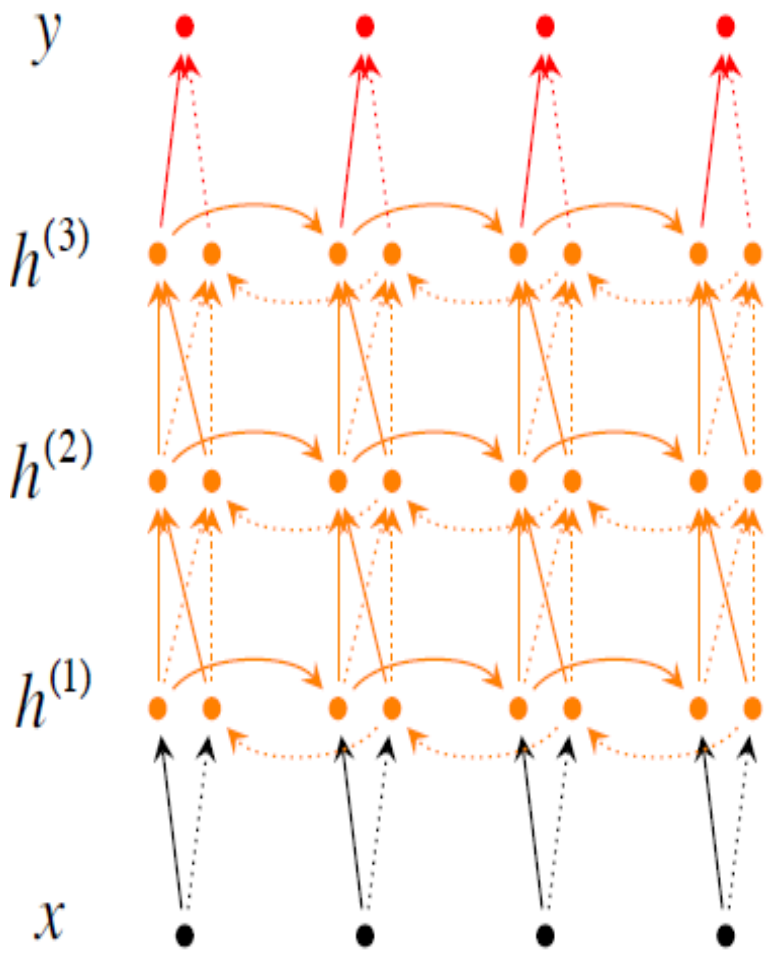
$$y_t = g(U[\vec{h}_t; \overleftarrow{h}_t] + c)$$

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

$h = [\vec{h}; \overleftarrow{h}]$ 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

Long Short Term Memory (LSTM): Motivation 1 of 2

- 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!

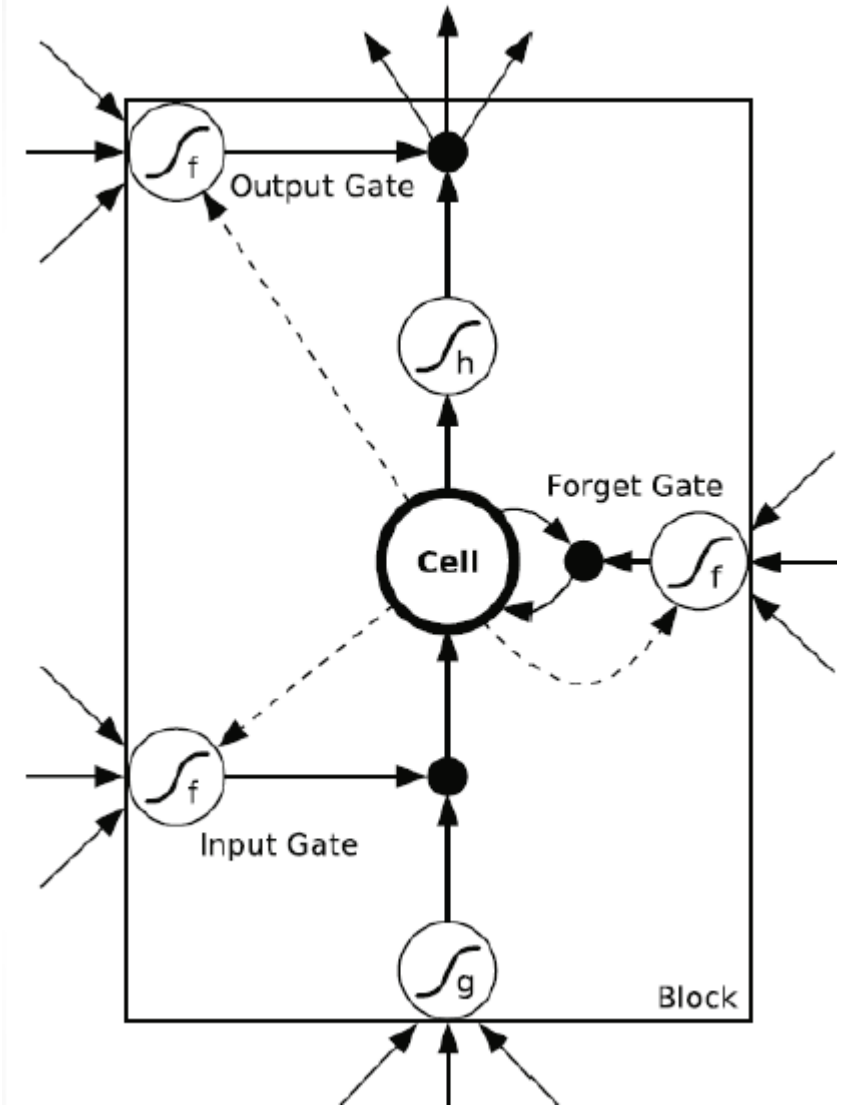
LSTM Motivation 2 of 2

- Observations from the case studies:
- A product review has many sentences and the pieces of information that we may be interested for making our buying decision is found at various places in the text.
- Certain aspects are “must have” for us that can’t be compromised. For instance if a customer needs an item within a few days, he can’t wait for it indefinitely. Similarly if he has a budget constraint, he can’t buy the item even if it is the best fit for his 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) with over 100 sentences, 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 back 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”
- LSTM is proposed as a solution to address this issue

The five key Architectural Elements of LSTM

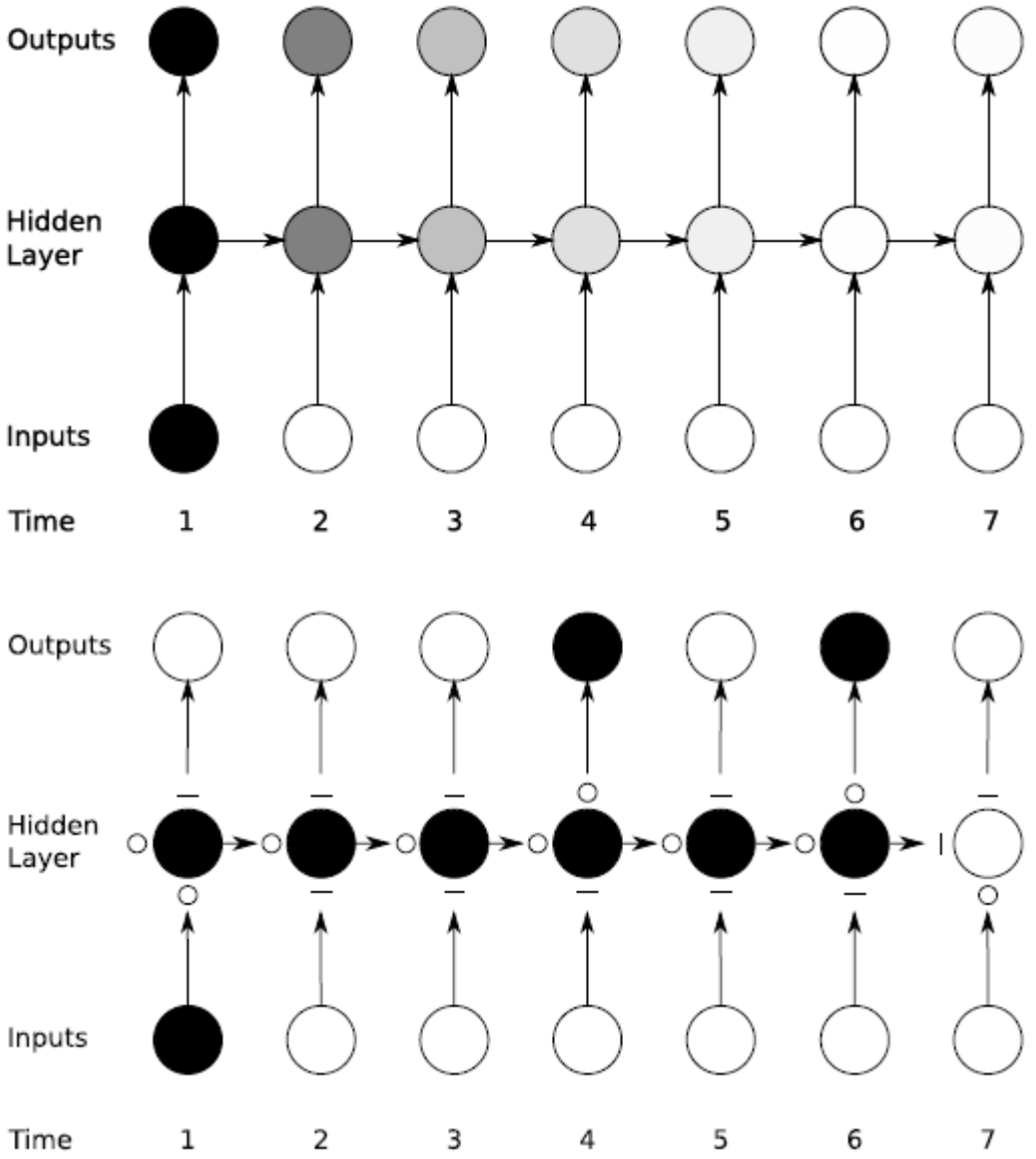
- Input Gate
- Forget Gate
- Cell
- Output Gate
- Hidden state output

$$i_t = \sigma\left(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i\right)$$
$$f_t = \sigma\left(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f\right)$$
$$c_t = f_t c_{t-1} + i_t \tanh\left(W_{xc}x_t + W_{hc}h_{t-1} + b_c\right)$$
$$o_t = \sigma\left(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o\right)$$
$$h_t = o_t \tanh(c_t)$$



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.



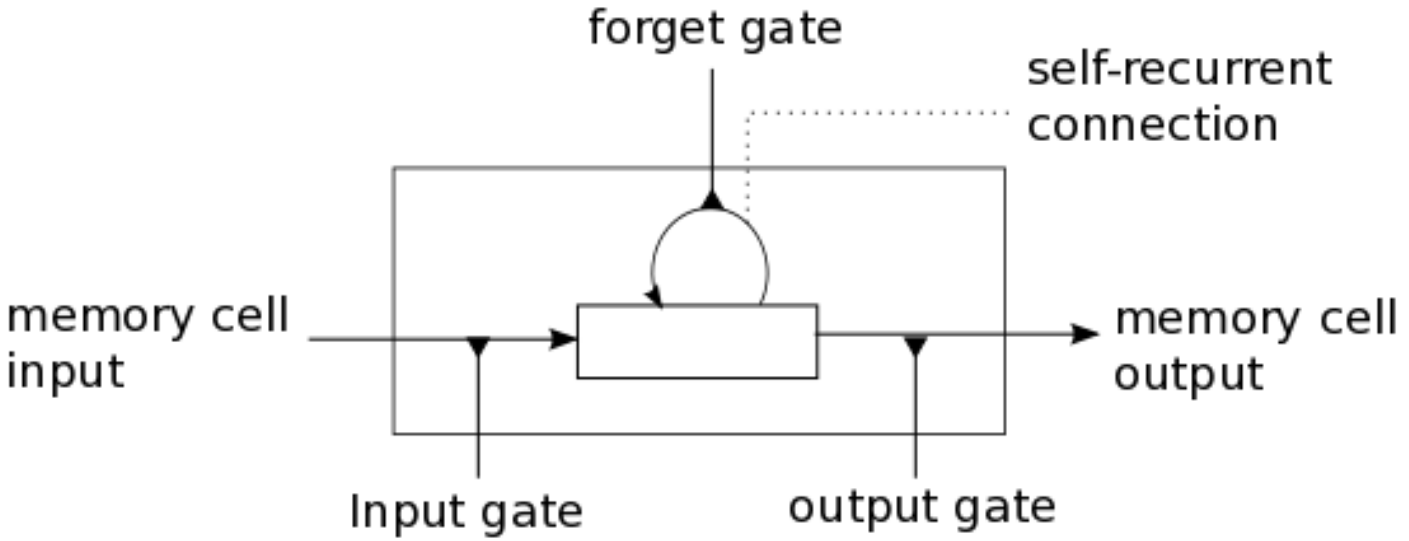
Implementing an LSTM: Notes for practitioners

- Some points to take in to account while choosing an LSTM architecture:
  - LSTM has many variants compared to the architecture proposed in the paper by Sepp Hochreiter and Schmidhuber
  - 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 (input, forget, output).
  - Some variants adopt a simpler version. E.g. peephole connections may be omitted
  - Training is a bit complex compared to feedforward ANN
  - Many training techniques are reported. For BPTT see Alex Graves's thesis
  - See Theano for Python DL library
- 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



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

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

Backup slides