

# Recurrent Neural Network

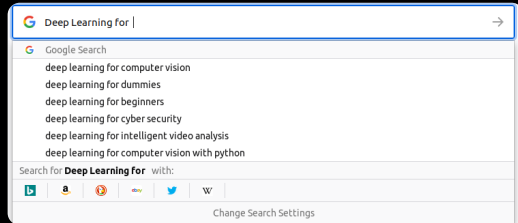
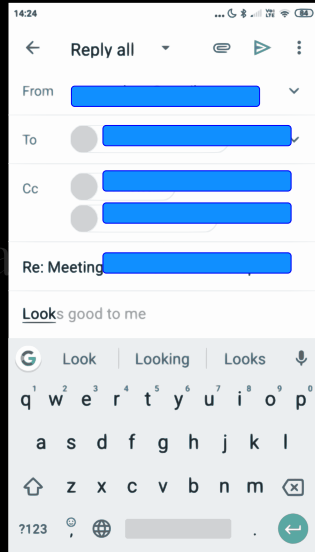
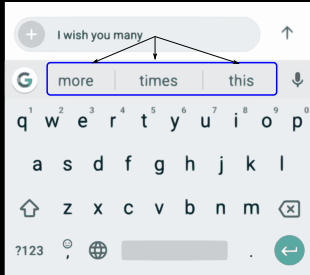
Ramaseshan Ramachandran

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

- ① Recurrent Neural Network
  - Language Model - Recap
  - Limitations of Word2Vec
  - Is standard ANN good enough?
  - Sequence Learning
  - Recurrent Neural Network
  - Recurrent Neuron
  - Unrolled RNN
  - Character based LM - RNN
  - Language Model - RNN
  - Training
  - RNN Training- Back propagation Through Time
- ② Back Propagation Through Time

- Back Propagation Through Time
- Derivatives of Activation Functions
- Perplexity
- Exploding/Vanishing Gradient
- Gradient Clipping
- ③ Long Short Term Memory
  - Introduction
  - LSTM Cell
  - LSTM Forward Pass
  - Truncated BPTT
  - Kinematics Problem Generation
- ④ Gated Recurrent Unit
  - Introduction
  - GRU Forward pass

# NLP APPLICATIONS IN ACTION



# PROBABILISTIC LANGUAGE MODEL

---

**Goal:** Compute the probability of a sequence of words  $P(W) = P(w_1, w_2, w_3, \dots, w_n)$

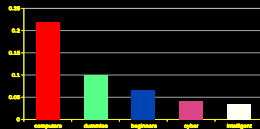
Chain rule converts it into product of conditional probabilities  $P(W) = \prod_{k=1}^n P(w_k | w_1^{k-1})$

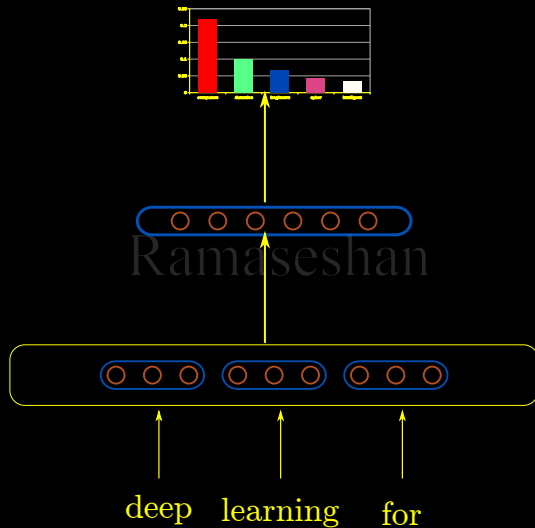
Using Markov-approximation, this could be written as  $P(W) \approx \prod_{k=1}^n P(w_k | w_{k-K+1}^{k-1})$

The next word is predicted using n-grams is

$$P(w_{k+1} | w_k) = \frac{\text{Count of n-gram}}{\text{Count of (n-1)gram}}$$

$$P(w | \text{Deep Learning for}) = \frac{\text{Count}(\text{Deep Learning for } w)}{\text{Count}(\text{Deep Learning for})}$$





# LIMITATIONS OF FIXED INPUT NEURAL NETWORKS

---

- ▶ Embeddings are learned based on a small local window surrounding words
  - ▶ *good* and *bad* share the almost the same embedding
- ▶ Does not address polysemy
  - ▶ The boys play cricket on the banks of a river
  - ▶ The boys play cricket near a national bank
- ▶ Does not use frequencies of term co-occurrences
- ▶ Word embedding provide distributed vectors for words
  - ▶ How about phrases? "India Today", Indian Express, The Sun News,
  - ▶ Can we encode a sentence as a distributed vector - Sentence vectors?
  - ▶ How about paragraphs?

## LIMITATIONS...

---

- ▶ Memory less and does not bother where the words and context come from
- ▶ Handle variable length text..
- ▶ Some NLP tasks require semantic modeling over the whole sentence
  - ▶ Machine translation
  - ▶ Question answering, chat-bots
  - ▶ Text summarization
- ▶ The data is considered as static - does not depend on a sequence or time-
- ▶ They are location invariant
- ▶ Some important tasks depend on the sequence of data  
( $y(t+1) = f(x(t), x(t-1), x(t-2) \dots x(t-n))$ )

Sequence learning is the study of machine learning algorithms designed for applications that require sequential data or temporal data

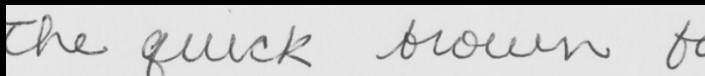
Karnateshan



# APPLICATIONS

---

- ▶ Named Entity Recognition
- ▶ Paraphrase detection - identifying semantically equivalent questions
- ▶ Language Generation
- ▶ Machine Translation
- ▶ Speech recognition
  - ▶ Wreck a nice beach or recognize speech
- ▶ Automatically generating subtitles for a video
- ▶ Spell Checking
- ▶ Predictive typing
- ▶ Chat-bots/Dialog understanding
- ▶ Generate/correct Hand-written text

A rectangular image showing a snippet of handwritten text in cursive script. The text is 'The quick brown fox' and is written on a white background. The handwriting is dark and fluid.

- ▶ Sequential data prediction is considered as a key problem in machine learning and artificial intelligence
- ▶ Unlike images where we look at the entire image, we read text documents sequentially to understand the content.
- ▶ The likelihood of any sentence can be determined from everyday use of language.
- ▶ The earlier sequence of words (int time) is important to predict the next word, sentence, paragraph or chapter
- ▶ If a word occurs twice in a sentence, but could not be accommodated in the sliding window, then the word is learned twice
- ▶ An architecture that does not impose a fixed-length limit on the prior context

- ▶ States are important in the reading exercise. The previous state definitely affects the next state
- ▶ In order to use the previous state, we need to store it or remember it
- ▶ Traditional Neural networks were not designed as a state machine as anything outside the context window has no impact on the decision being made.
- ▶ Traditional Neural networks do not accept arbitrary input length.
- ▶ Inherent ability to model sequential input
- ▶ Handle variable length inputs without the use of arbitrary fixed-sized windows
- ▶ Use its own output as input
- ▶ RNNs encode not only attributional similarities between words, but also similarities between pairs of words item Analogy - *Chennai : Tamil :: London : English* or *go and went* is same as *run and Ran* or *queen  $\approx$  king - man + woman*

# A SIMPLE RECURRENT NEURAL NETWORK

---

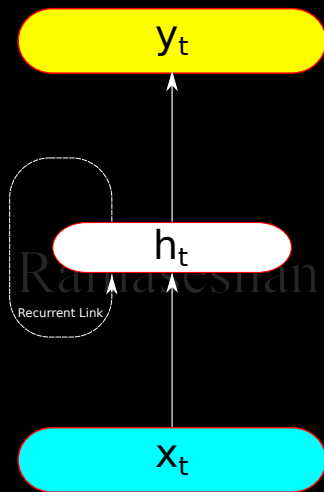
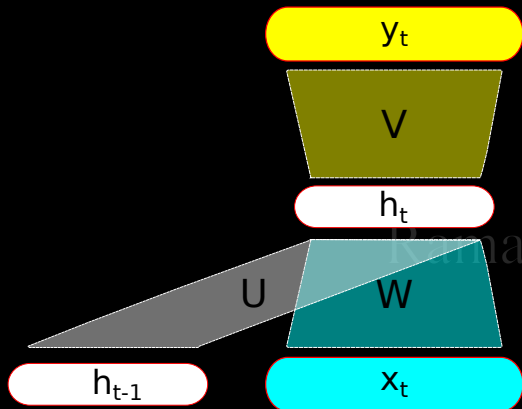


Figure: A simple Recurrent Neural Network

# RNN - AN EXTENSION OF A FEED-FORWARD NETWORK



- ▶ the memory includes the information

from the start of the sentence with no imposition of window size

- ▶ The hidden weights  $U$  from the time-stamp  $h_{t-1}$  is the significant addition to RNN
- ▶ The past weights from the previous time-stamp determines memory of the network

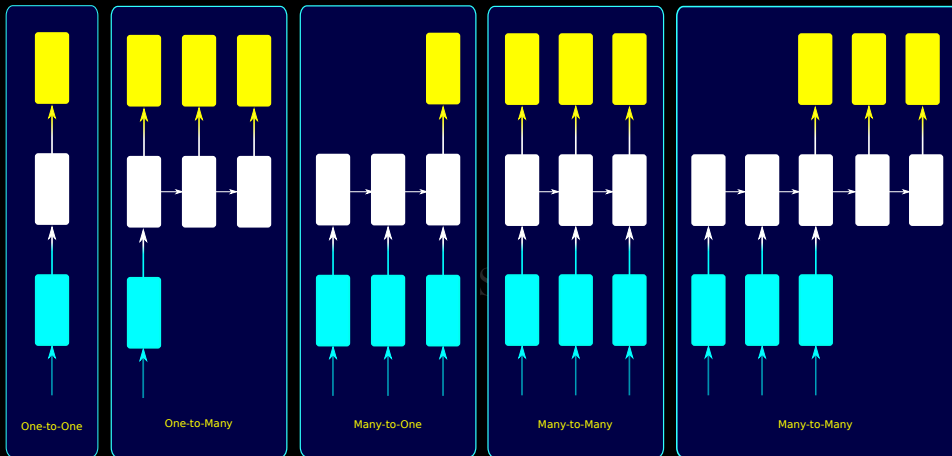
$$h_t = f(Uh_{t-1} + Wx_t)$$

$$y_t = Vh_t$$

$x_t$  : Input at time  $t$

$h_{t-1}$  : State of hidden weights at time  $t - 1$

# MULTIPLE ARCHITECTURES OF RNN



One-to-One: Classification

One-to-Many: Image captioning and image description

Many-to-One: Sentiment Analysis

Many-to-Many: Machine translation

Many-to-Many: Synced sequence input and output - frame by frame labelling

Source: <http://karpathy.github.io/2015/05/21/rnn-effectiveness/>

# FEED FORWARD ALGORITHM

---

[H]

$h \leftarrow 0; t \leftarrow 0;$

$t < \text{len}(x) \quad h_t \leftarrow g(Uh_{t-1} + Wx_t)$

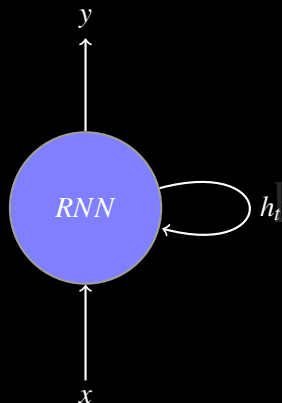
$y_t \leftarrow f(Vh_t)$

$t \leftarrow t + 1 \quad y$

Ramaseshan

# RECURRENT NEURON

---



We can process a sequence of vectors  $x$  by applying a recurrence formula at every time step:

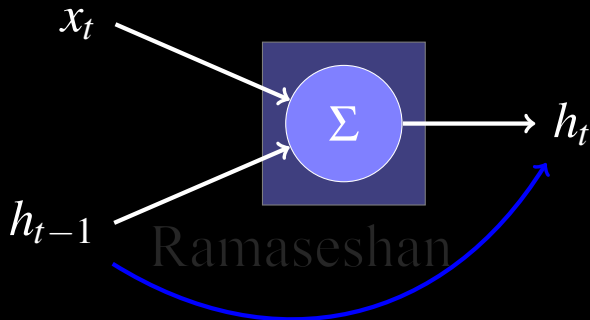
$$h_t = f_w(h_{t-1}, x_t), \text{ where}$$

$h_t$  is the new state,

$h_{t-1}$  is the old state and

$x_t$  is the input at state  $t$  or at time  $t$



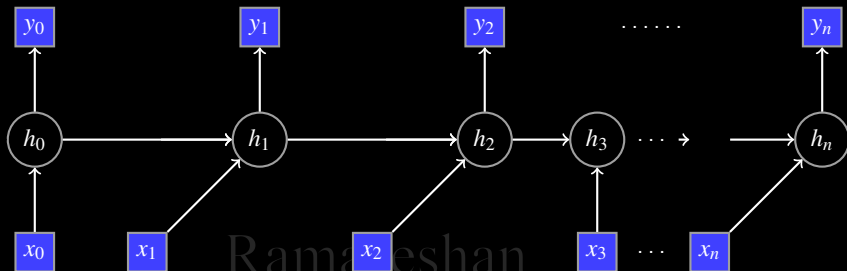


$$h_t = f(U * h_{t-1} + W * x_t)$$

$x_t$  : Input at time

$h_{t-1}$  : State of neuron at time  $t - 1$

# UNROLLED RNN



## Parameters for RNN

- ▶  $W$  - input to hidden weights
- ▶  $U$  - hidden to hidden weights
- ▶  $V$  - the hidden to output.

All  $W, U$  and  $V$  are shared.

$$h_0 = \sigma(Wx_0) \quad (1)$$

$$h_1 = \sigma(Uh_0 + Wx_1) \quad (2)$$

$$\dots \quad (3)$$

$$h_n = f(Uh_{n-1} + Wx_n), \forall n \quad (4)$$

$$y_n = V * h_n, \text{ where } n = 1, 2, \dots, N \quad (5)$$

# RNN UNROLLED IN TIME

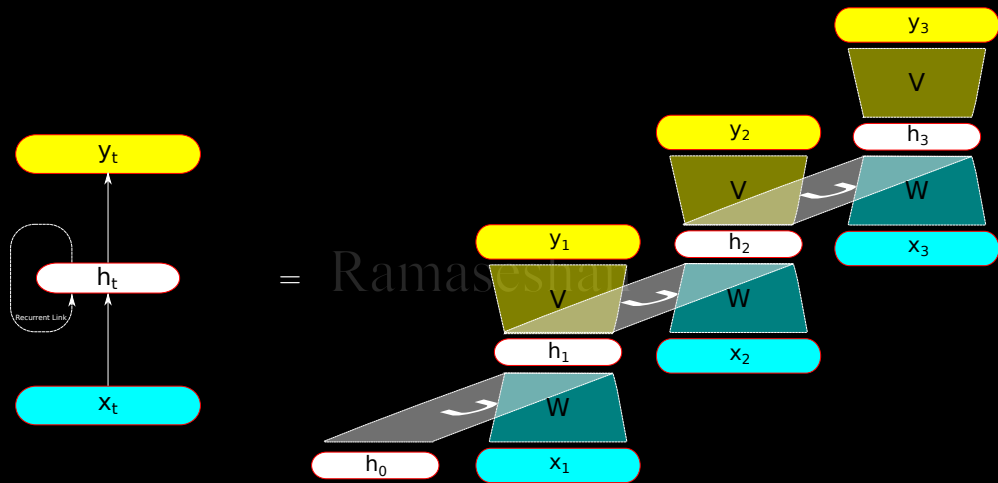
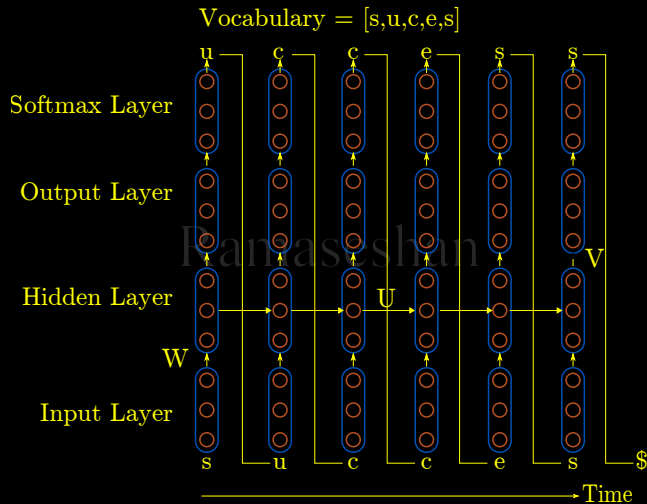
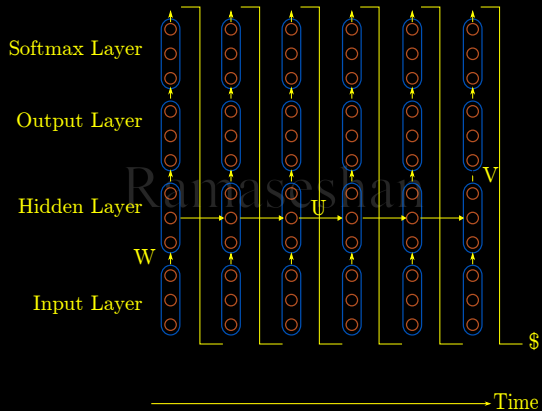


Figure: RNN Unrolled in time

# CHARACTER BASED LM - RNN



# LANGUAGE MODEL - RNN



- ▶ FF network is static - does not worry about the sequence of the or order of the patterns, it does not matter where they occur
- ▶ The sequence must be preserved
- ▶ Two kinds of Training
  - ▶ back propagation through time
  - ▶ real time recurrent learning

Ramaseshan

# RNN TRAINING- BACK PROPAGATION THROUGH TIME

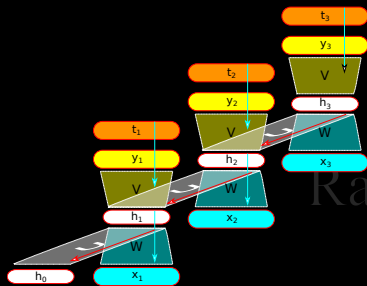


Figure: RNN Training using backpropagation

$$z^{[1]} = Wx \quad (6)$$

$$a^{[1]} = g(z^{[1]}) \quad (7)$$

$$z^{[2]} = Ua^{[1]} \quad (8)$$

$$a^{[2]} = g(z^{[2]}) \quad (9)$$

$$y = f(Va^{[2]}) \quad (10)$$

$$\frac{\partial L}{\partial V} = \frac{\partial L}{\partial a} \frac{\partial a}{\partial z} \frac{\partial z}{\partial V} \quad (11)$$

$$\delta h = g'(z)V\delta_{out} + \delta_{next} \quad (12)$$

# RNN - TRAINING - BACKWARD PASS

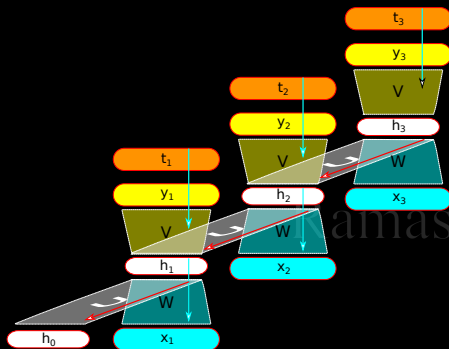


Figure: RNN Training using backpropagation

$$\delta_{out} = \frac{\partial L}{\partial a} \frac{\partial a}{\partial z} \quad (13)$$

$$\frac{\partial L}{\partial V} = \delta_{out} h_t \quad (14)$$

$$\frac{\partial L}{\partial W} = \delta h x_t \quad (15)$$

$$\frac{\partial L}{\partial U} = \delta h h_{t-1} \quad (16)$$



- ▶ Theoretically, it is possible to store all historical information in the RNN
- ▶ Vanishing gradient problem - The diminishing value of  $\delta$  makes it difficult to capture the long term memory as we move down the memory lane or layers of hidden nodes
- ▶ What is the solution?

# LONG SHORT-TERM MEMORY (LSTM)

---

Learning to store information over extended time intervals via recurrent takes a very long time, mostly due to insufficient, decaying error back flow

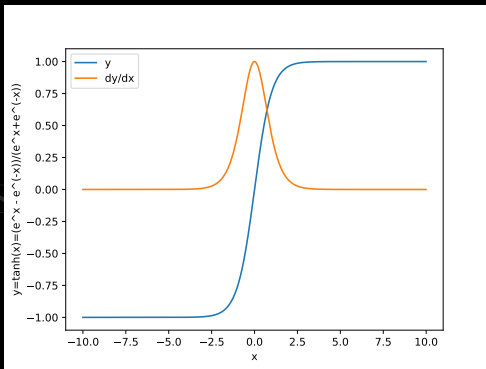
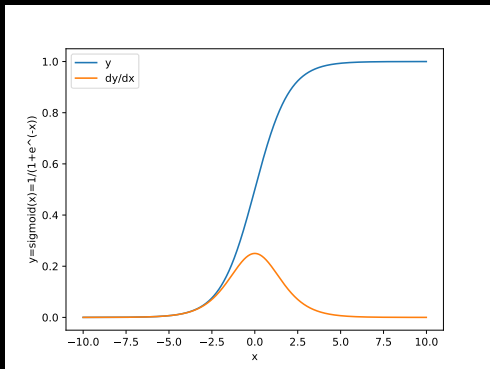
[Hochreiter:1997:LSM:1246443.1246450]

- ▶ Vanilla RNNs are good at learning from sequential recency rather than from long term dependency
- ▶ The temporal evolution of the back-propagated error exponentially depends on the size of the weights
- ▶ The gradients tend to either (1) explode or (2) vanish:
  - ▶ If it explodes up, then the learning may lead to oscillating weights
  - ▶ If it vanishes, then either takes a lot of time to learn or fails

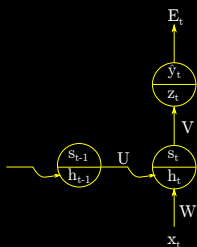
# DERIVATIVES OF ACTIVATION FUNCTIONS

Activation Function	Derivative
$y = \left(\frac{1}{1 + e^{-x}}\right)$	$\frac{dy}{dx} = \left(\frac{1}{1 + e^{-x}}\right) \left(1 - \frac{1}{1 + e^{-x}}\right) = \sigma(x)(1 - \sigma(x))$
$y = \tanh(x) = \frac{\sinh(x)}{\cosh(x)}$	$\frac{dy}{dx} = \frac{\cosh(x) \cosh(x) - \sinh(x) \sinh(x)}{\cosh^2(x)} = (1 - \tanh^2(x))$
$E = -y \log_{10}(\hat{y})$	$\frac{dE}{d\hat{y}} = -\frac{y_j}{\hat{y}_j}$
$E = \frac{1}{2} \sum_j (y_j - \hat{y}_j)^2$	$\frac{dE}{d\hat{y}_t} = -(y - \hat{y})$

# DERIVATIVES OF SIGMOID AND TANH



# FORWARD PASS - NETWORK EQUATIONS



## Forward pass

$$h_t = Wx_t + Uh_{t-1} \quad (17)$$

$$s_t = \tanh(h_t) \quad (18)$$

$$z_t = Vs_t \quad (19)$$

$$\hat{y}_t = \text{softmax}(z_t) \quad (20)$$

$$E = - \sum_t y_t \log(\hat{y}_t) \quad (21)$$

If the corpus contains  $T$  words, then  $(x_1, x_2, x_3, \dots, x_T)$  are the corresponding word vectors

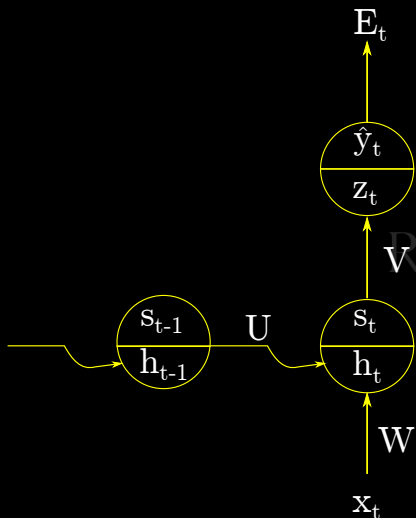
- ▶  $x_t \in R^{D_w}$  represents the input word at time  $t$  and  $D_w$  is the dimension of the word vector. If one-hot vector, it will be  $x^{D_v}$
- ▶  $W \in R^{D_w \times D_h}$  is the weight matrix that conditions the input vector
- ▶  $U \in R^{D_h \times D_h}$  matrix that keeps the dependency of the word sequence
- ▶  $V \in R^{|V| \times R^{D_h}}$
- ▶  $s_{t-1}$  is the output of the non-linear function ( $\tanh$ ) of the time step  $t - 1$
- ▶  $\hat{y}_t^t \in R^{|V|}$  is the probability distribution of the predicted word at time step  $t$  for the given context of  $x_1, x_2, x_3, \dots, x_t$ , where  $|V|$  is the size of the vocabulary

## SIZE OF THE RNN NETWORK

---

If we assume the size of the word vector as 100 and the number of the hidden neurons as 500, and  $|V| = 10000$ , then

Parameter	Size
Word Vector	100
$W$	$500 \times 100$
$h_t, s_t$	500
$U$	$500 \times 500$
$V$	$500 \times 10000$
$\hat{y}_t$	10000



$$h_t = Wx_t + Uh_{t-1}$$

$$s_t = \tanh(h_t)$$

$$z_t = Vs_t$$

$$\hat{y}_t = \text{softmax}(z_t)$$

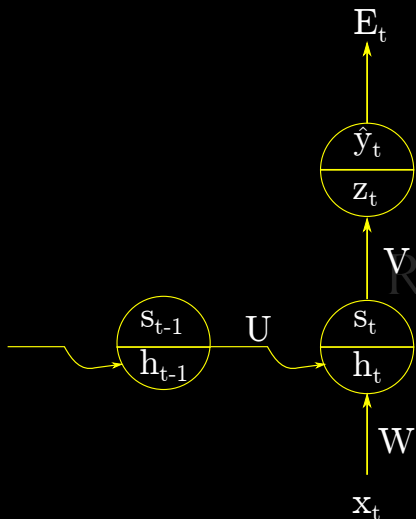
$$E_t = -y_t \log(\hat{y}_t)$$

$$\frac{\partial E_t}{\partial V} = \frac{\partial E_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial z_t} \frac{\partial z_t}{\partial V} \quad (22)$$

$$\text{Let } \delta_{out}^t = \frac{\partial E_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial z_t}$$

$$\frac{\partial E_t}{\partial V} = \delta_{out}^t s_t \quad (23)$$

Here  $\delta_{out}^t$  is the loss for each of the units in the output layer

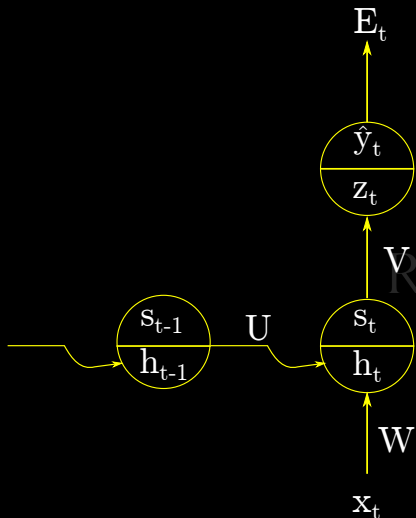


$$\frac{\partial E_t}{\partial W} = \underbrace{\frac{\partial E_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial z_t} \frac{\partial z_t}{\partial s_t} \frac{\partial s_t}{\partial h_t} \frac{\partial h_t}{\partial W}}_{\text{Ramasesha}} \quad (24)$$

$$= \delta_{out}^t V \sigma'(h_t) x_t \quad (25)$$

Since the hidden layer activation depends on the previous time state, we have another similar term  $\delta_{t-1}$  that get added to (25)

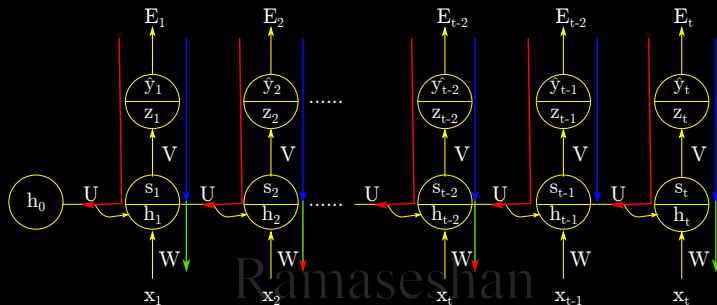




$$\frac{\partial E_t}{\partial U} = \underbrace{\frac{\partial E_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial z_t} \frac{\partial z_t}{\partial s_t} \frac{\partial s_t}{\partial h_t}}_{\text{Ramaseshan}} \frac{\partial h_t}{\partial U} \quad (26)$$

$$= \delta_{out}^t V \sigma'(h_t) h_{t-1} \quad (27)$$

Since we are back propagating the error from the current state to the previous state,  $\delta_{next} = \sigma(h_t) U \delta_{out}^t V \sigma'(h_t)$  needs to be added



The error for the entire duration of  $T$  for all the vocabulary is the sum of all the error across the layers

$$E(\theta) = -\frac{1}{T} \sum_{t=1}^T \sum_{j=1}^{|V|} y_{t,j} \log(\hat{y}_{t,j}) \quad (28)$$

This term (28) is known as the perplexity. Lower the value of  $2^{E(\theta)}$  better is the confidence of the network in predicting the next word

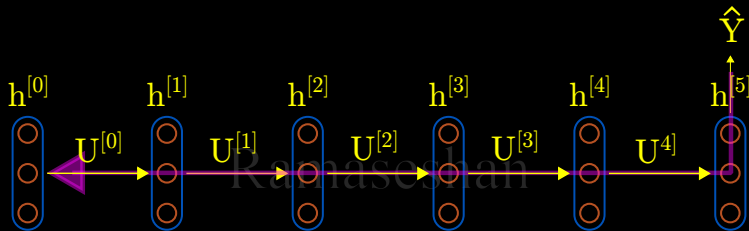
Perplexity is a measurement of how well a model predicts a sample. Perplexity is defined as

$$\text{For bigram model, } PP(W_N) = \sqrt[N]{\prod_{i=1}^N \frac{1}{P(w_i|w_{i-1})}} \quad (29)$$

$$\text{For trigram model } PP(W_N) = \sqrt[N]{\prod_{i=1}^N \frac{1}{P(w_i|w_{i-1}w_{i-2})}} \quad (30)$$

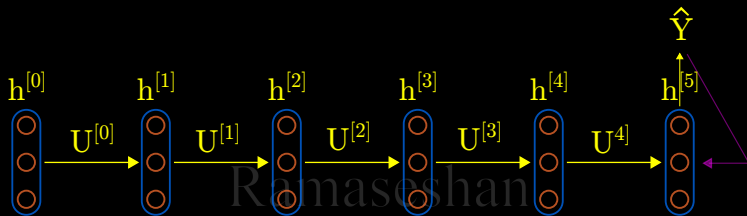
A good model gives maximum probability to a sentence or minimum perplexity to a sentence

# EXPLODING/VANISHING GRADIENT

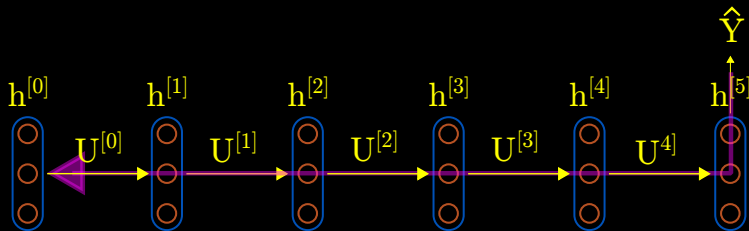


Let us assume that  $\hat{y}_t = f(U, h)$ . We want to propagate the error through backpropagation

# EXPLODING/VANISHING GRADIENT



The error  $E$  at 5th time state depends on  $h^{[5]}$ . Hence we need to estimate  $\frac{\partial E}{\partial h^{[5]}}$ .  $h^{[5]}$  depends on  $h^{[4]}$ ,  $h^{[4]}$  depends on  $h^{[3]}$ ,  $h^{[3]}$  depends on  $h^{[2]}$ ,  $h^{[2]}$  depends on  $h^{[1]}$ , and  $h^{[1]}$  depends on  $h^{[0]}$ .



Ramaseshan

$$\frac{\partial E^{[5]}}{\partial h^{[0]}} = \frac{\partial E^{[5]}}{\partial h^{[5]}} \times \frac{\partial h^{[5]}}{\partial h^{[4]}} \times \frac{\partial h^{[4]}}{\partial h^{[3]}} \times \frac{\partial h^{[3]}}{\partial h^{[2]}} \times \frac{\partial h^{[2]}}{\partial h^{[1]}} \times \frac{\partial h^{[1]}}{\partial h^{[0]}} = \frac{\partial E^{[5]}}{\partial h^{[5]}} \prod_{t=1}^{t=5} \frac{\partial h^{[t]}}{\partial h^{[t-1]}} \quad (31)$$

Generalizing

$$\frac{\partial E^{[\tau]}}{\partial h^{[0]}} = \frac{\partial E^{[\tau]}}{\partial h^{[\tau]}} \prod_{t=1}^{t=\tau} \frac{\partial h^{[t]}}{\partial h^{[t-1]}}, \text{ where } \tau \text{ represents depth of the layers} \quad (32)$$

$$\frac{\partial E^{[\tau]}}{\partial h^{[0]}} = \frac{\partial E^{[\tau]}}{\partial h^{[\tau]}} \prod_{t=1}^{t=\tau} \frac{\partial h^{[t]}}{\partial h^{[t-1]}}$$

$$h^{[t]} = \sigma(WX^{[t]} + Uh^{[t-1]}) \quad (33)$$

$$\frac{\partial h^{[t]}}{\partial h^{[t-1]}} = \text{diag}(\sigma'(WX^{[t-1]} + Uh^{[t-1]}))U \quad (34)$$

where  $\sigma'$  computes element-wise the derivative of  $\sigma$

$$\frac{\partial h^{[t]}}{\partial h^{[t-1]}} \text{ is a Jacobian} \quad (35)$$

$$\therefore \frac{\partial E^{[\tau]}}{\partial h^{[0]}} = \frac{\partial E^{[\tau]}}{\partial h^{[\tau]}} U^{\tau} \prod_{t=1}^{t=\tau} \text{diag}(\sigma'(WX^{[t-1]} + Uh^{[t-1]})) \quad (36)$$

Values of  $U$  become very small when the depth increases<sup>1</sup>

<sup>1</sup>Source: "On the difficulty of training recurrent neural networks", Pascanu et al, 2013 - <http://proceedings.mlr.press/v28/pascanu13.pdf>

## EXPLODING/VANISHING GRADIENT

---

Consider the following sentence:

Raj entered CoffeeDay to meet his partner Dru. Raj said "Hi Dru. In the next few hours they discussed their start-up and devised a plan to develop a product on knowledge management. After a the long discussion and fruitful discussion, Raj said goodbye to his \_\_\_\_\_( $47^{th}$  word).

The target word is **partner**. If the long distance gradient (the gap between  $U^{[7]}$  and  $U^{[\$]}$  is large), then the target word is lost in the gradient as it would be too small to contribute

The decay in the gradient value is proportional to the depth of the network. The deeper the net network, the the chance of getting a smaller value of the gradient towards the end of the backpropagation. If the some of the values are in the range of  $[(0.01, 0.5), (0.03, 0.01)]$ , then the the derivative would vanish to zero -  $0.01^{47} = 1.0e-94$  and  $0.5^{47} = 7.1054274e-15$



- ▶ The gradient is either very large or very small. This can cause the optimizer to converge slowly.
- ▶ To speed up training, clip the gradient at certain values
  - ▶ If  $g < 1$ , or if  $g > 1$ , then  $g = 1$
  - ▶ Or
  - ▶ If  $\|g\| > threshold$ , then  $g \leftarrow \frac{threshold}{\|g\|} g$
- ▶ Clip the gradient if it exceeds a threshold

- ▶ The component of the gradient in directions that correspond to long-term dependencies is small<sup>2</sup>
- ▶ The component of the gradient in directions that correspond to short-term dependencies is large
- ▶ As a result, RNNs can easily learn the short-term but not the long-term dependencies

Ramaseshan

---

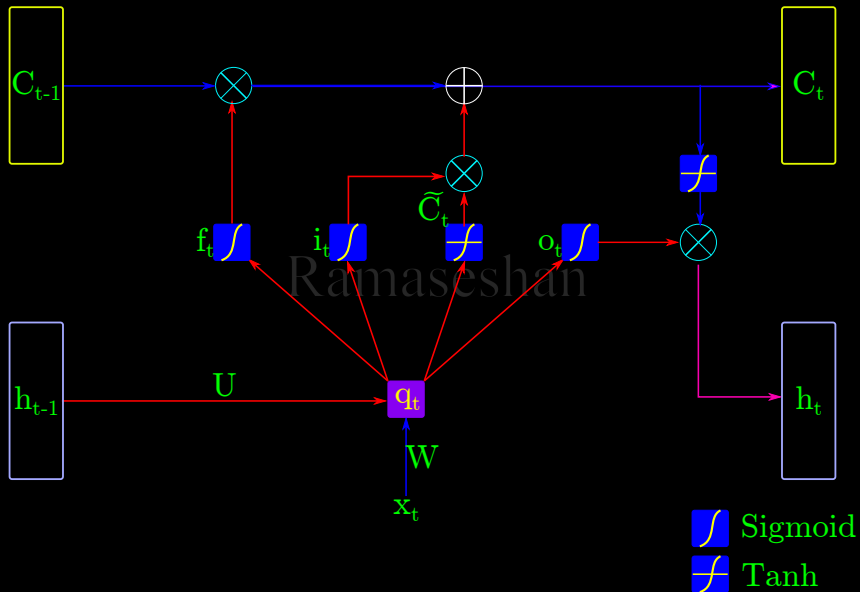
<sup>2</sup>An empirical exploration of recurrent network architectures - <http://dl.acm.org/citation.cfm?id=3045118.3045367>

- ▶ In LSTM network is the same as a standard RNN, except that the summation units in the hidden layer are replaced by memory blocks
- ▶ The multiplicative gates allow LSTM memory cells to store and access information over long periods of time, thereby mitigating the vanishing gradient problem<sup>3</sup>
- ▶ Along with the hidden state vector  $h_t$ , LSTM maintains a memory vector  $C_t$
- ▶ At each time step the LSTM can choose to read from, write to, or reset the cell using explicit gating mechanisms
- ▶ LSTM computes well behaved gradients by controlling the values using the gates

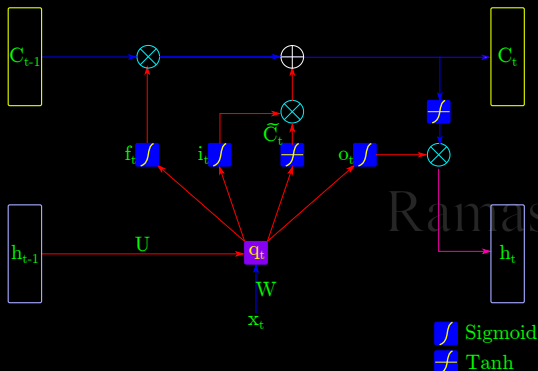
---

<sup>3</sup><http://dblp.uni-trier.de/db/journals/corr/corr1506.html#KarpathyJL15>

# LSTM CELL



# LSTM - FORWARD PASS



$$f_t = \sigma(W_{ft}q_t + b_f) \quad (37)$$

$$i_t = \sigma(W_{it}q_t + b_i) \quad (38)$$

$$\tilde{C}_t = \tanh(W_{\tilde{C}_t}q_t) \quad (39)$$

$$C_t = (f_t \otimes C_{t-1}) \oplus (i_t \otimes \tilde{C}_t) \quad (40)$$

$$o_t = \sigma(W_{ot}q_t + b_o) \quad (41)$$

$$h_t = o_t \otimes \tanh(C_t) \quad (42)$$

$$s_t = \tanh(h_t) \quad (43)$$

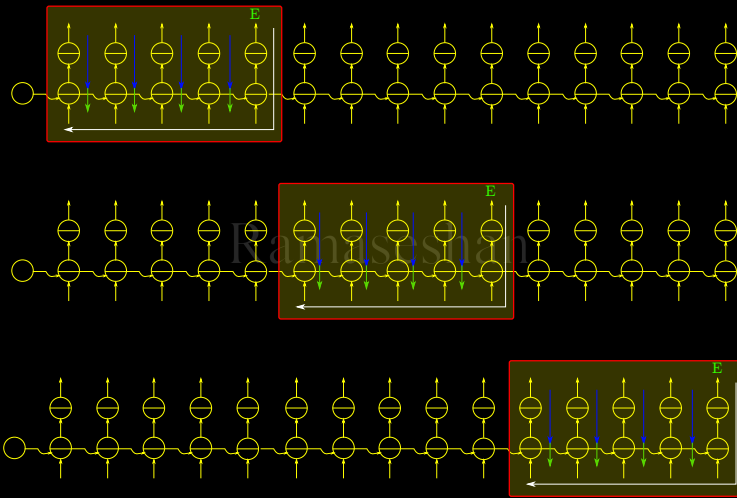
$$z_t = Vz_t \quad (44)$$

$$\hat{y}_t = softmax(z_t) \quad (45)$$

For applications with long sequences, the input is truncated into manageable fixed-sized segments. This approach is called Truncated Backpropagation Through Time (TBPTT).

### Example

Consider a sequence of 5000 samples. We could split this in to 50 sequences of 100 samples each, and the BPTT is computed for each sequence. This works most of the time, but it is blind to temporal dependencies that may span across two sequences. One way to solve this is to have a sentence separator as the conditional BPTT.



# RNN - KINEMATICS PROBLEM GENERATION

---

Contains around 270+ problems in Kinematics Divided into 100 characters/sequence  
Each sequence is trained and learn to predict the next character (alphabet, punctuations, numbers)

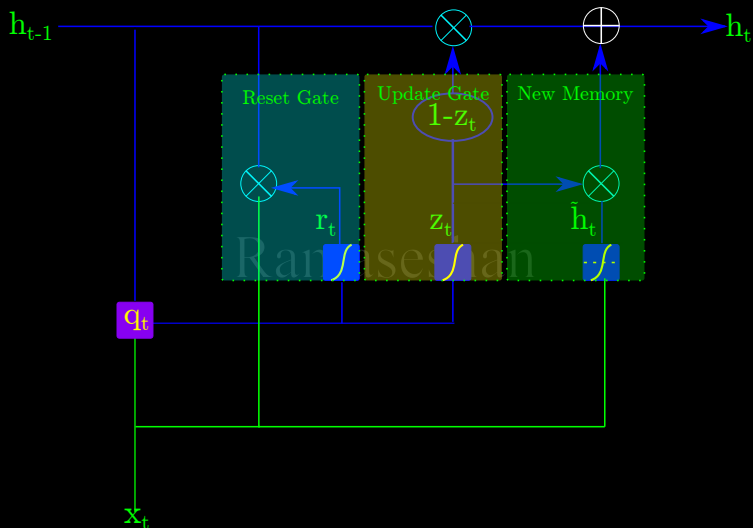
## Sample problem

A ball is thrown upward from a bridge with an initial velocity of 5.9 m/s. It strikes water after 2s. If  $g=9.8\text{m/s}^2$  What is the height of the bridge ?

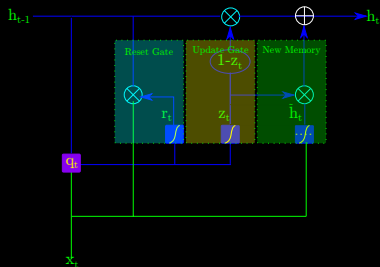
- ▶ 5% training - What is the acceleration of the car passing the car ski distance. A croosts from the wate it stop
- ▶ 25% training - What is the distance constant reach and aft when it hits the same ball. A ball is thrown out of a velo
- ▶ **Recall** - "determine the time it takes a piece of glass to hit the ground? A car drives straight off the edge of a cliff"
- ▶ Epochs = 1500, Hidden units=75, Hidden Layer = 2,  $\eta = 0.01$ , Chunk size=150



# INTRODUCTION TO GATED RECURRENT UNIT



# GRU FORWARD PASS



$$q_t = f(h_{t-1}, x_t) \quad (46)$$

$$z_t = \sigma(U_z, q_t) \quad (47)$$

$$r_t = \sigma(U_r, q_t) \quad (48)$$

$$\tilde{h}_t = \tanh(W \cdot (r_t, q_t)) \quad (49)$$

$$h_t = (1 - z_t) \otimes h_{t-1} \oplus (z_t \otimes \tilde{h}_t) \quad (50)$$

$$s_t = \tanh(h_t) \quad (51)$$

$$\hat{y}_t = \text{softmax}(V s_t) \quad (52)$$

## Intuition

If the reset gate values  $\rightarrow 0$ , previous memory states are faded and new information is stored. If the  $z_t$  is close to 1, the information is copied and retained thereby adjusting the gradient to be alive for the next time step, thereby long-term dependency is stored. BPTT decides the learning of the reset and update gate.