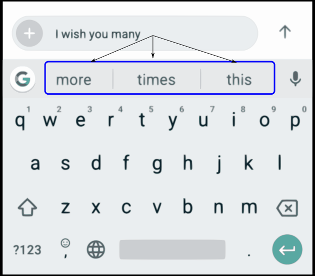
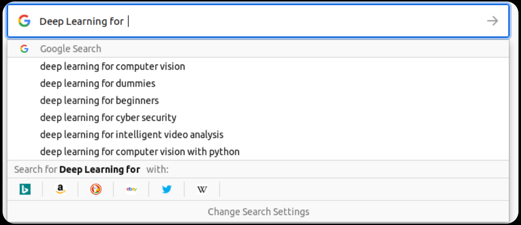
Recurrent Neural Network

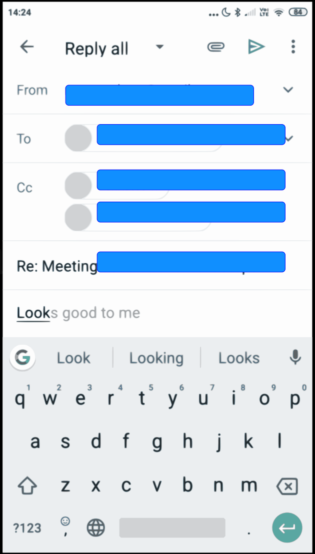
Ramaseshan Ramachandran

Ramaseshan

NLP APPLICATIONS IN ACTION







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

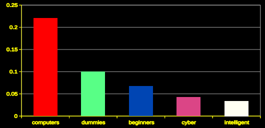
Chain rule converts it into product of conditional probabilities $P(W) = \prod_{k=1}^{n} P(w_k | w_1^{k-1})$

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

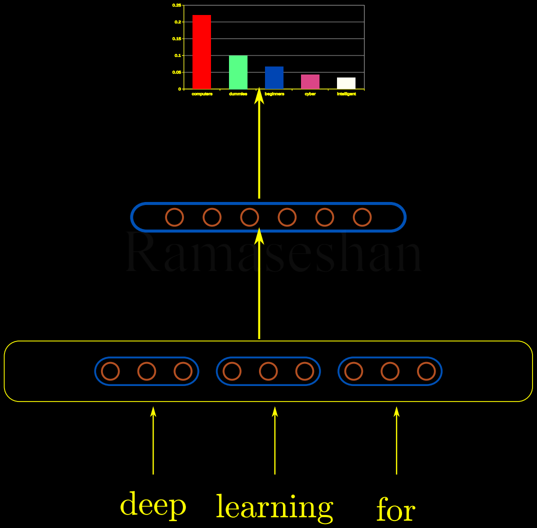
The next word is predicted using n-grams is

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

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



ANN FOR LM


0.25
0.2
0.15
0.1
0.05
0
computers
dummies
beginners
cyber
intelligent
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, char-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 $(y(t+1) = f(x(t), x(t-1), x(t-2) \ldots x(t-n)))$

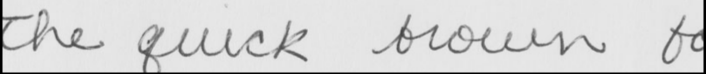
SEQUENCE LEARNING

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

Recurrent Neural Network

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



RECURRENT NEURAL NETWORK

- 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

RNN

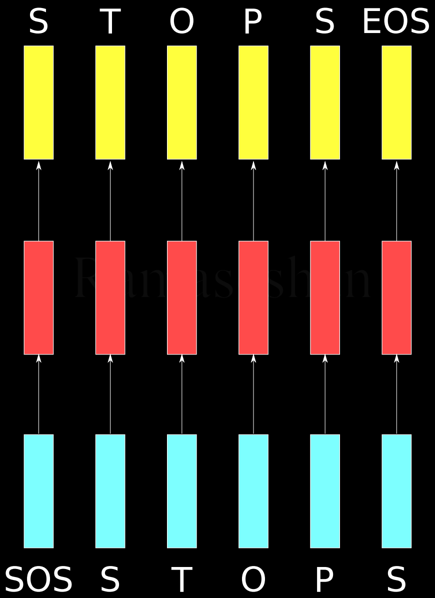
- States are important in the reading exercise. The previous state definitely affects the next state
- In order to use the previous state, we need to store it or remember it
- Traditional Neural networks were not designed as a state machine as anything outside the context window has no impact on the 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
- RNNs are dynamic models
- Analogy - $Chennai : Tamil :: London : English$ or $go\,and\,went$ is same as $run\,and\,Ran$ or $queen \approx king - man + woman$

IMPORTANT INGREDIENTS

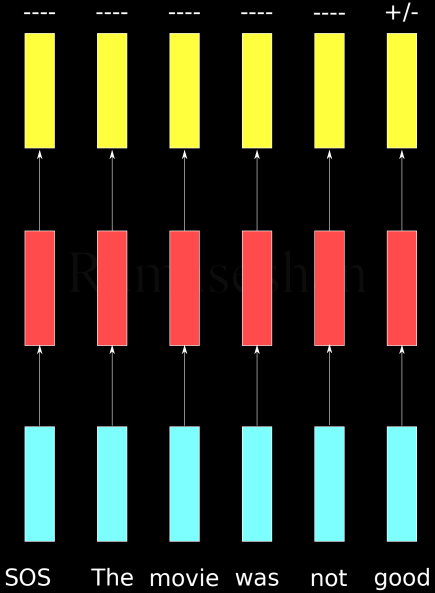
- Develop a model to handle variable input sequence size
- Manage the inter-dependency of input
- Perform the same task for every input

Recurrent Neural Network

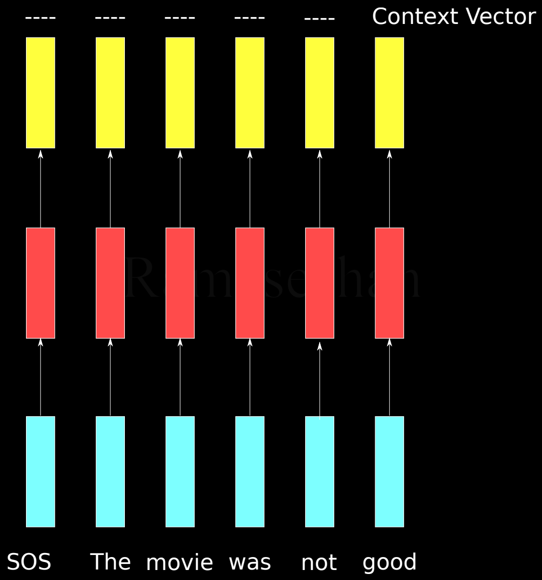
CHARACTER-BASED MODEL



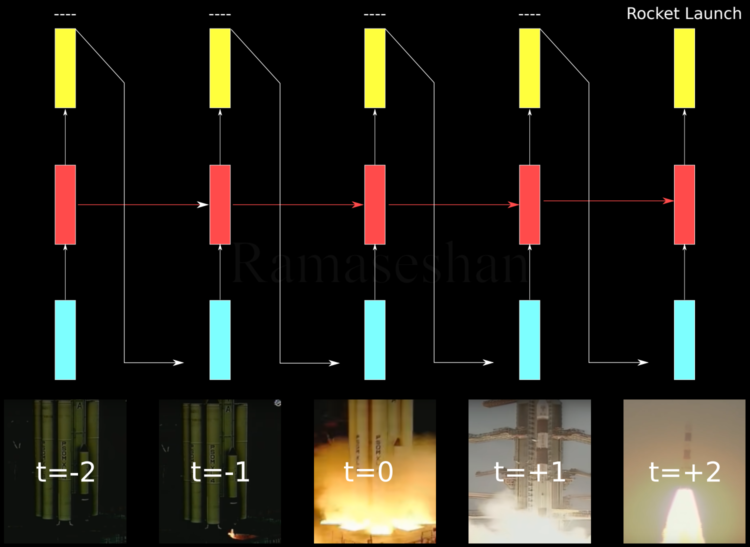
WORD-BASED MODEL



CONTEXT VECTOR MODEL



ANOTHER EXAMPLE TO ILLUSTRATE THE TIME SERIES



A SIMPLE RECURRENT NEURAL NETWORK

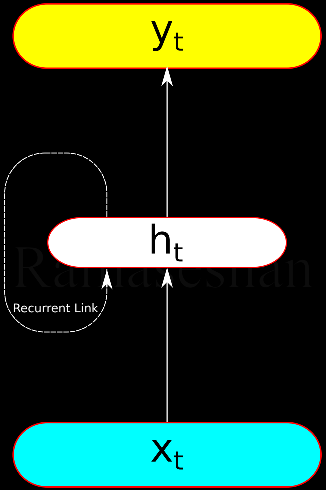
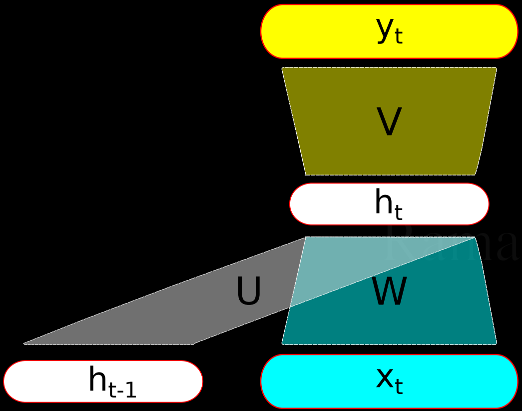


Figure: A simple Recurrent Neural Network

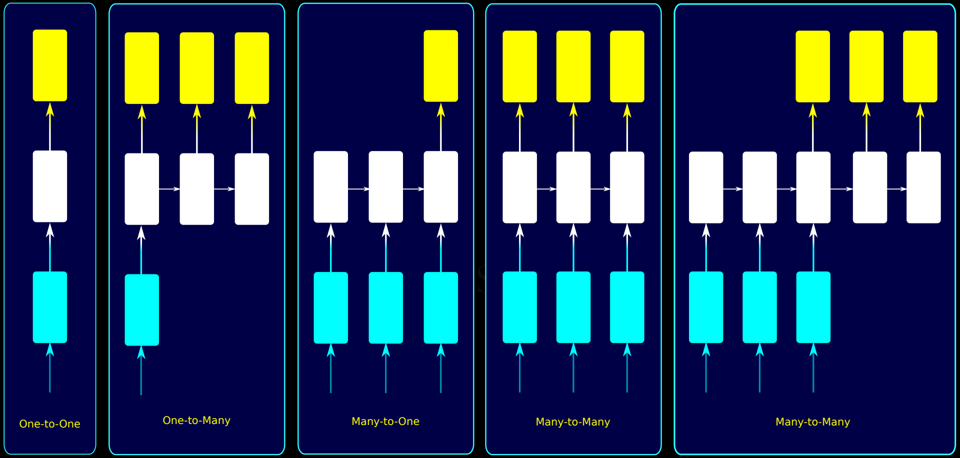
RNN - AN EXTENSION OF A FEED-FORWARD NETWORK



- 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--effec:veness/

FEED FORWARD ALGORITHM

Algorithm 1: Feed forward algorithm

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

while $t < len(x)$ **do**

    $h_t \leftarrow g(Uh_{t-1} + Wx_t)$

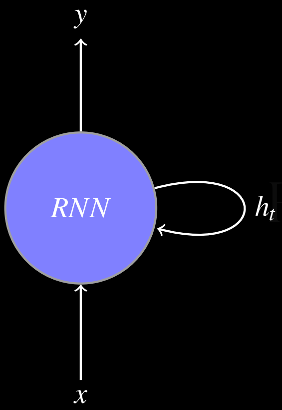
    $y_t \leftarrow f(Vh_t)$

    $t \leftarrow t + 1$

end

Result: y

RECURRENT NEURON



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

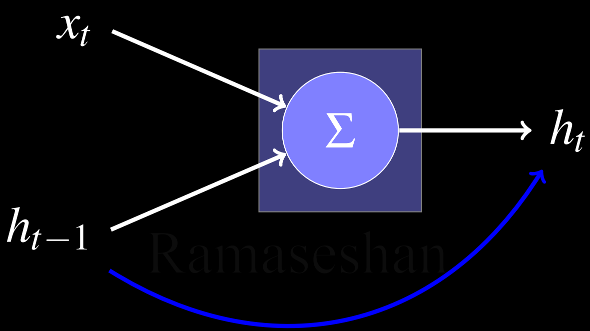
$h_t = f_w(h_{t_1}, x_t)$, where

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

RECURRENT NEURON

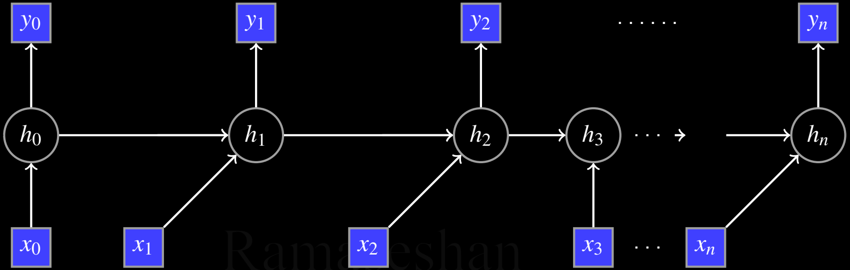


$$h_t = f(U * h_{t-1} + W * x_t)$$

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.

$$h_0 = \sigma(Wx_0) \tag{1}$$

$$h_1 = \sigma(Uh_0 + Wx_1) \tag{2}$$

$$\ldots \tag{3}$$

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

$$y_n = V * h_n, \text{ where } n = 1, 2, \ldots N \tag{5}$$

RNN UNROLLED IN TIME

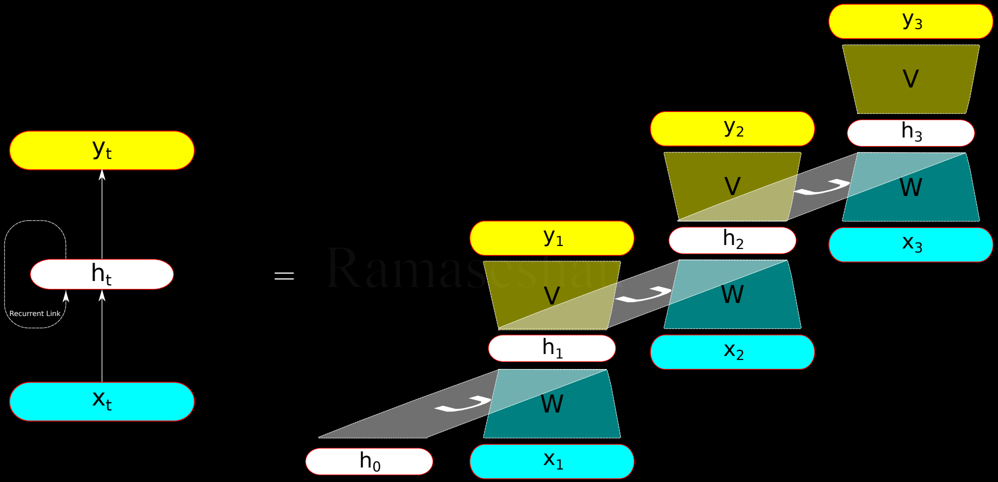
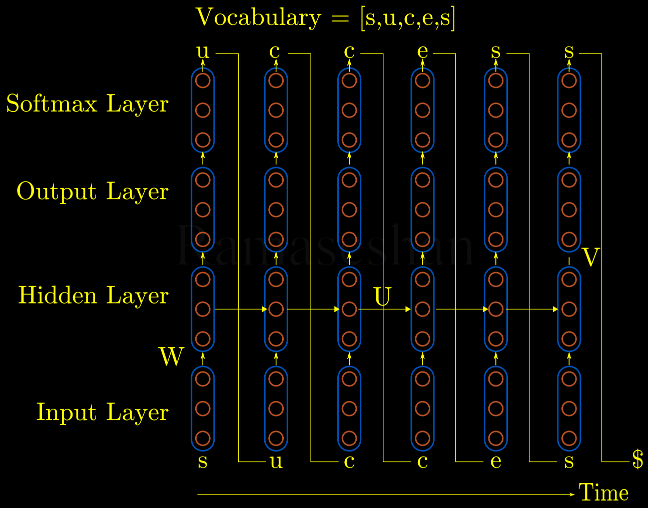
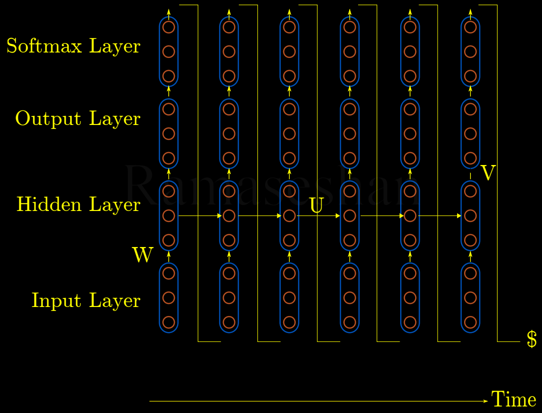


Figure: RNN Unrolled in time

CHARACTER BASED LM - RNN



LANGUAGE MODEL - RNN



Recurrent Neural Network

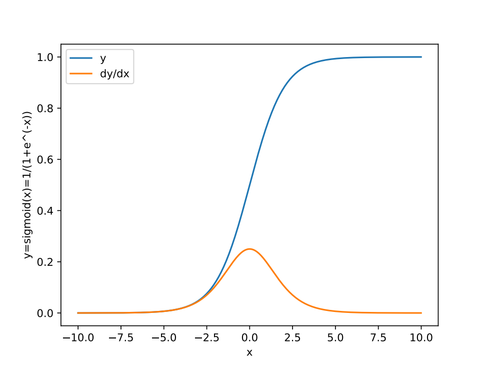
TRAINING

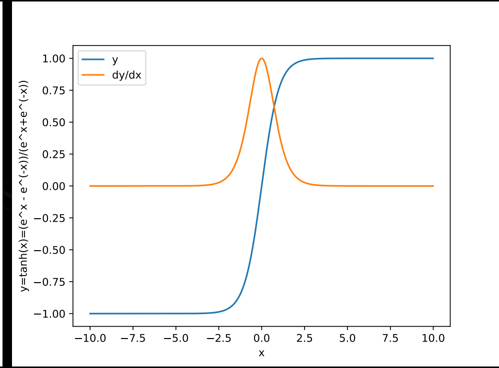
- FF network is static - does not worry about the sequence of the or order of the patterns, it does not matter where they occur
- The sequence must be preserved
- Two kinds of Training
  - back propagation through time
  - real time recurrent learning

DERIVATIVES OF ACTIVATION FUNCTIONS

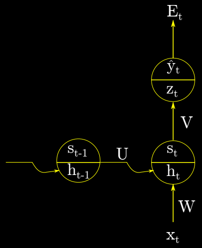
Activation Function	Derivative
$y = (\frac{1}{1+e^{-x}})$	$\frac{dy}{dx} = \left(\frac{1}{1+e^{-x}}\right)\left(1 - \frac{1}{1+e^{-x}}\right) = \sigma(x)(1-\sigma(x))$
$y = \tanh(x) = \frac{sinh(x)}{cosh(x)}$	$\frac{dy}{dx} = \frac{\cosh(x)\cosh(x) - \sinh(x)\sinh(x)}{\cosh^2(x)} = (1-\tanh^2(x))$
$E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

DERIVATIVES OF SIGMOID AND TANH





FORWARD PASS - NETWORK EQUATIONS



Forward pass

$$h_t = Wx_t + Us_{t-1} \tag{6}$$

$$s_t = \tanh(h_t) \tag{7}$$

$$z_t = Vs_t \tag{8}$$

$$\hat{y}_t = softmax(z_t) \tag{9}$$

$$E = -\sum_t y_t \log(\hat{y}_t) \tag{10}$$

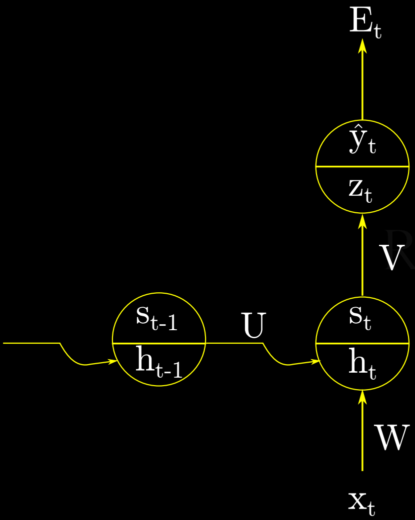
If the corpus contains T words, then $(x_1, x_2, x_3, \ldots, x_T)$ are the corresponding word vectors

- $x_t \in R^{D_{|w|}}$ 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 $x_1, x_2, x_3, \ldots x_t$, 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

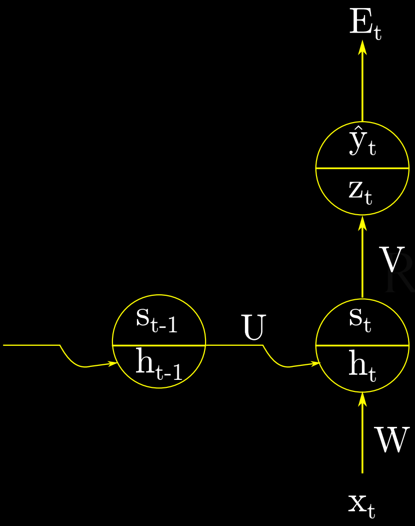
Parameter	Size
Word Vector	100
W	500x100
h_t, s_t	500
U	500x500
V	500x10000
$\hat{y}_t$	10000



$$h_t = Wx_t + Us_{t-1}$$
$$s_t = \tanh(h_t)$$
$$z_t = Vs_t$$
$$\hat{y}_t = softmax(z_t)$$
$$E_t = -y_t \log(\hat{y}_t)$$
$$\frac{\partial E_t}{\partial V} = \frac{\partial E_t}{\partial \hat{y}_t}\frac{\partial \hat{y}}{\partial z_t}\frac{\partial z_t}{\partial V} \quad (11)$$
$$\text{Let } \delta^t_{out} = \frac{\partial E_t}{\partial \hat{y}_t}\frac{\partial \hat{y}}{\partial z_t}$$
$$\frac{\partial E_t}{\partial V} = \delta^t_{out} s_t \quad (12)$$

Here δ^t_{out} is the loss for each of the units in the output layer

BPTT - DERIVATIVE FOR W

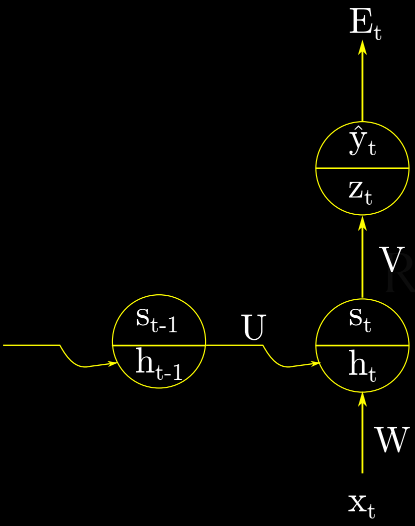


$$\frac{\partial E_t}{\partial W} = \underbrace{\frac{\partial E_t}{\partial \hat{y}_t}\frac{\partial \hat{y}}{\partial z_t}}\frac{\partial z_t}{\partial s_t}\frac{\partial s_t}{\partial h_t}\frac{\partial h_t}{\partial W} \tag{13}$$

$$= \delta^t_{out} V \sigma^{'}(h_t) x_t \tag{14}$$

Since the hidden layer activation depends on the previous time state, we have another similar term δ_{t-1} that get added to (14)

BPTT - DERIVATIVE FOR U

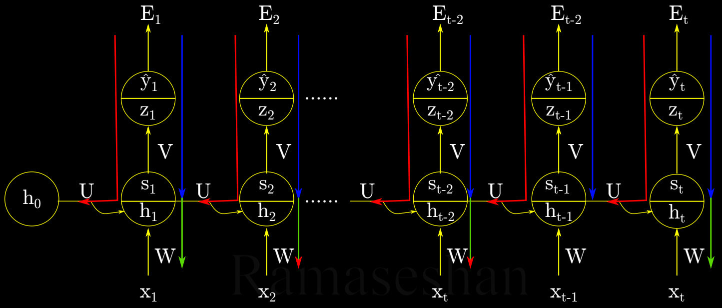


$$\frac{\partial E_t}{\partial U} = \underbrace{\frac{\partial E_t}{\partial \hat{y}_t}\frac{\partial \hat{y}}{\partial z_t}\frac{\partial z_t}{\partial s_t}\frac{\partial s_t}{\partial h_t}}\frac{\partial h_t}{\partial U} \tag{15}$$

$$= \delta^t_{out} V \sigma^{'}(h_t) h_{t-1} \tag{16}$$

Since we are back propagating the error from the current state to the previous state, $\delta_{next} = \sigma(h_t) U \delta^t_{out} V \sigma^{'}(h_t)$ needs to be added

BPTT - UNROLLED RNN



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

$$E(\theta) = -\frac{1}{T}\sum_{1}^{T}\sum_{j=1}^{|V|} y_{t,j}\log(\hat{y_{t,j}}) \tag{17}$$

This term (17) is known as the perplexity. Lower the value of $2^{E(\theta)}$ better is the confidence of the network in predicting the next word

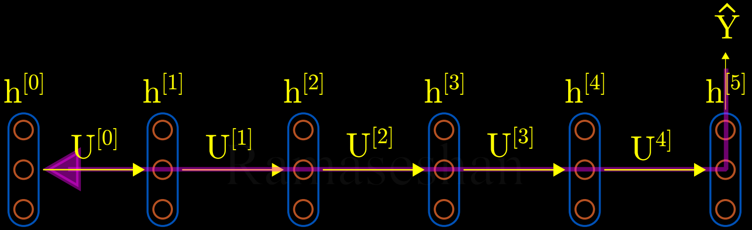
PERPLEXITY

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

$$\text{For bigram model, } PP(W_N) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_{i-1})}} \tag{18}$$

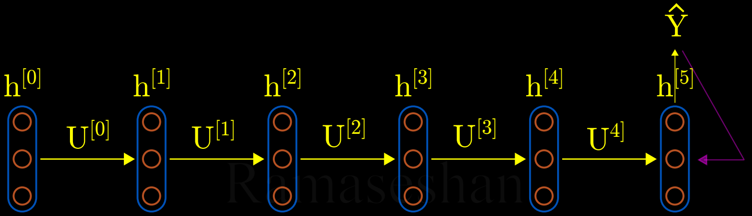
$$\text{For trigram model } PP(W_N) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_{i-1}w_{i-2})}} \tag{19}$$

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



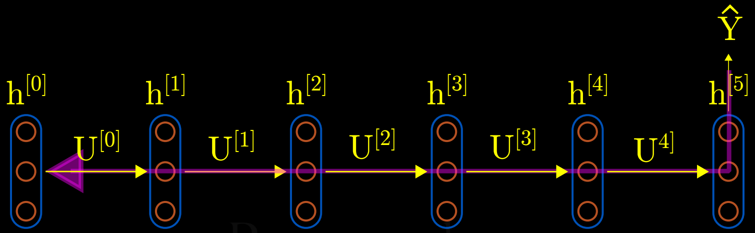
Let us assume that $\hat{y}_t = f(U,h)$. We want to propagate the error through backpropagation

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 $h^{[0]}$.

EXPLODING/VANISHING GRADIENT



$$\frac{\partial E^{[5]}}{\partial h^{[0]}} = \frac{\partial E^{[5]}}{\partial h^{[5]}} \times \frac{\partial h^{[5]}}{\partial h^{[4]}} \times \frac{\partial h^{[4]}}{\partial h^{[3]}} \times \frac{\partial h^{[3]}}{\partial h^{[2]}} \times \frac{\partial h^{[2]}}{\partial h^{[1]}} \times \frac{\partial h^{[1]}}{\partial h^{[0]}} = \frac{\partial E^{[5]}}{\partial h^{[5]}} \prod_{t=1}^{t=5} \frac{\partial h^{[t]}}{\partial h^{[t-1]}} \text{Generalizing}$$

$$\frac{\partial E^{[\tau]}}{\partial h^{[0]}} = \frac{\partial E^{[\tau]}}{\partial h^{[\tau]}} \prod_{t=1}^{t=\tau} \frac{\partial h^{[t]}}{\partial h^{[t-1]}}, \text{ where } \tau \text{ represents depth of the layers} \tag{20}$$

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

$$s^{[t]} = \sigma\left(WX^{[t]} + Us^{[t-1]}\right) \tag{21}$$

$$\frac{\partial s^{[t]}}{\partial s^{[t-1]}} = diag(\sigma'(WX^{[t]} + Us^{[t-1]}))U \tag{22}$$

where σ' computes element-wise the derivative of σ

$$\frac{\partial s^{[t]}}{\partial s^{[t-1]}} \text{ is a Jacobian} \tag{23}$$

$$\therefore \frac{\partial E^{[\tau]}}{\partial s^{[0]}} = \frac{\partial E^{[\tau]}}{\partial s^{[\tau]}} U^{\tau} \prod_{t=1}^{t=\tau} diag(\sigma'(WX^{[t]} + Us^{[t-1]})) \tag{24}$$

U gets very small when the depth increases[1]

[1] Source: "On the difficulty of training recurrent neural networks", Pascanuet 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 to 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 fag end of the backpropagation. If the some of the 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 $0.5^{47} = 7.1054274e - 15$

GRADIENT CLIPPING

- 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

BPTT DERIVATION - 1/2

$$\frac{\partial E_t}{\partial U} = \frac{\partial E_t}{\partial s_t}\frac{\partial s_t}{\partial U} \tag{25}$$

Using explicit and implicit rule and considering only three input parameters (x_1, x_2, x_3, x_4), $\dfrac{\partial s_4}{\partial U}$ can we rewritten as

$$\begin{align}
\frac{\partial s_4}{\partial U} &= \frac{\partial s_4^e}{\partial U} + \frac{\partial s_4}{\partial s_3}\frac{\partial s_3}{\partial U} \tag{26}\\
&= \frac{\partial s_4^e}{\partial U} + \frac{\partial s_4}{\partial s_3}\left[\frac{\partial s_3^e}{\partial s_2} + \frac{\partial s_3}{\partial s_2}\frac{\partial s_2}{\partial U}\right] \tag{27}\\
&= \frac{\partial s_4^e}{\partial U} + \frac{\partial s_4}{\partial s_3}\frac{\partial s_3^e}{\partial s_2} + \frac{\partial s_4}{\partial s_3}\frac{\partial s_3}{\partial s_2}\frac{\partial s_2}{\partial U} \tag{28}\\
&= \frac{\partial s_4^e}{\partial U} + \frac{\partial s_4}{\partial s_3}\frac{\partial s_3^e}{\partial s_2} + \frac{\partial s_4}{\partial s_3}\frac{\partial s_3}{\partial s_2}\left[\frac{\partial s_2^e}{\partial U} + \frac{\partial s_2}{\partial s_1}\frac{\partial s_1^e}{\partial U}\right] \tag{29}\\
&= \frac{\partial s_4^e}{\partial U} + \frac{\partial s_4}{\partial s_3}\frac{\partial s_3^e}{\partial s_2} + \frac{\partial s_4}{\partial s_3}\frac{\partial s_3}{\partial s_2}\left[\frac{\partial s_2^e}{\partial U} + \frac{\partial s_2}{\partial s_1}\frac{\partial s_1^e}{\partial U}\right] \tag{30}
\end{align}$$

$$= \frac{\partial s_4}{\partial s_4}\frac{\partial s_4^e}{\partial U} + \frac{\partial s_4}{\partial s_3}\frac{\partial s_3^e}{\partial s_2} + \frac{\partial s_4}{\cancel{\partial s_3}}\frac{\cancel{\partial s_3}}{\partial s_2}\frac{\partial s_2^e}{\partial U} + \frac{\partial s_4}{\cancel{\partial s_3}}\frac{\cancel{\partial s_3}}{\cancel{\partial s_2}}\frac{\cancel{\partial s_2}}{\partial s_1}\frac{\partial s_1^e}{\partial U} \tag{31}$$

$$= \sum_{\tau=1}^{4}\frac{\partial s_4}{\partial s_\tau}\frac{\partial s_\tau^e}{\partial U} \tag{32}$$

or in general, this could be written as

$$\frac{\partial s_t}{\partial U} = \sum_{\tau=1}^{t}\frac{\partial s_t}{\partial s_\tau}\frac{\partial s_\tau^e}{\partial U} \tag{33}$$

$$\therefore \frac{\partial E_t}{\partial U} = \frac{\partial E_t}{\partial s_t}\sum_{\tau=1}^{t}\frac{\partial s_t}{\partial s_\tau}\frac{\partial s_\tau^e}{\partial U} \tag{34}$$

From eq.20, $\dfrac{\partial s_t}{\partial s_\tau}$

$$\frac{\partial E_t}{\partial U} = \frac{\partial E_t}{\partial s_t}\sum_{\tau=1}^{t}\prod_{i=\tau+1}^{t}\frac{\partial s_i}{\partial s_{i-1}}\frac{\partial s_\tau^e}{\partial U} \tag{35}$$

PROBLEMS WITH VANILLA RNN

- The component of the gradient in directions that correspond to long-term dependencies is small[2]
- 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

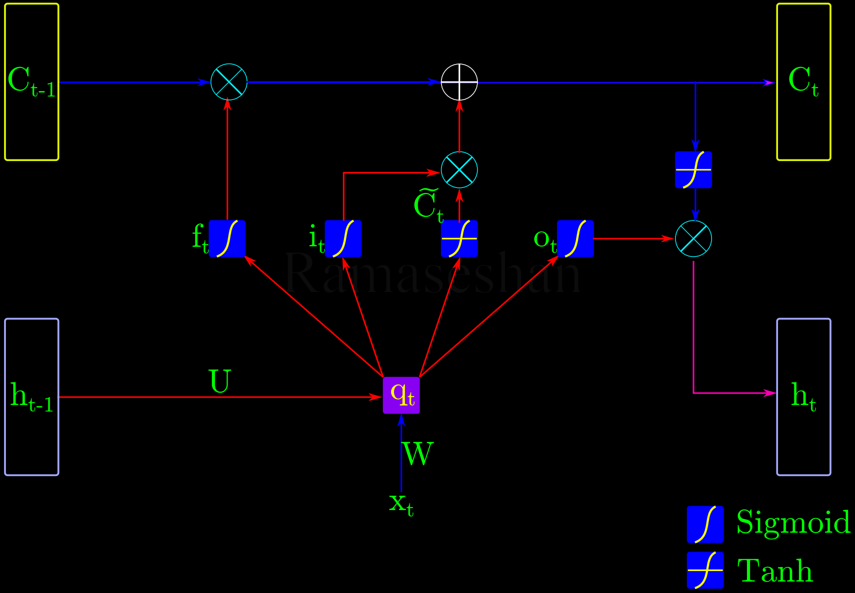
[2] An empirical exploration of recurrent network architectures - http://dl.acm.org/citation.cfm?id=3045118.3045367

LSTM

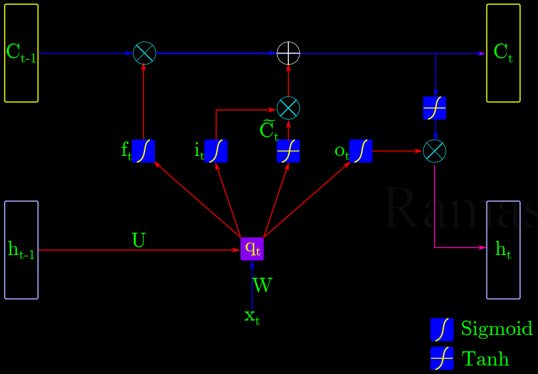
- In LSTM network is the same as a standard RNN, except that the summation units in the hidden layer are replaced by memory blocks
- The multiplicative gates allow LSTM memory cells to store and access information over long periods of time, thereby mitigating the vanishing gradient problem[3]
- Along 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

[3] http://dblp.uni-trier.de/db/journals/corr/corr1506.html#KarpathyJL15

LSTM CELL


C_{t-1}
C_t
$\widetilde{C}_t$
f_t
i_t
o_t
h_{t-1}
U
q_t
W
x_t
h_t
Sigmoid
Tanh
Ramaseshan

LSTM - FORWARD PASS



$$f_t = \sigma(W_{ft}q_t + b_f) \tag{36}$$

$$i_t = \sigma(W_{it}q_t + b_i) \tag{37}$$

$$\tilde{C}_t = \tanh(W_{\tilde{C}_t q_t}) \tag{38}$$

$$C_t = (f_t \otimes C_{t-1}) \oplus (i_t \otimes \tilde{C}_t) \tag{39}$$

$$o_t = \sigma(W_{ot}q_t + b_o) \tag{40}$$

$$h_t = o_t \otimes \tanh(C_t) \tag{41}$$

$$s_t = \tanh(h_t) \tag{42}$$

$$z_t = V z_t \tag{43}$$

$$\hat{y}_t = softmax(z_t) \tag{44}$$

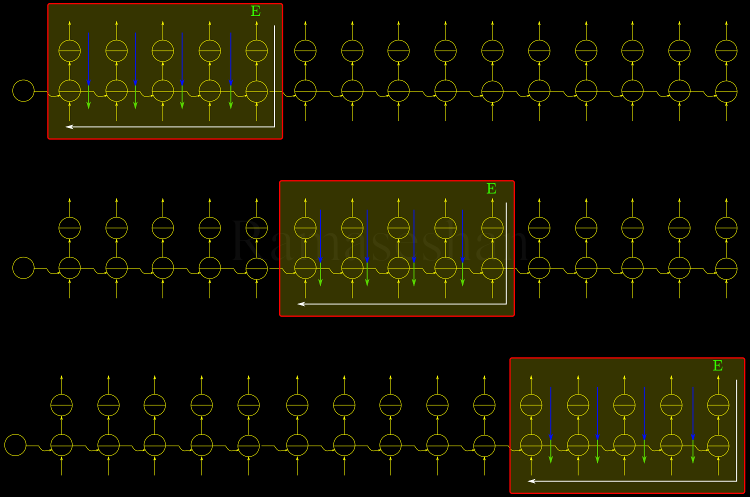
TRUNCATED BPTT

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.

TBPTT



RNN - KINEMATICS PROBLEM GENERATION

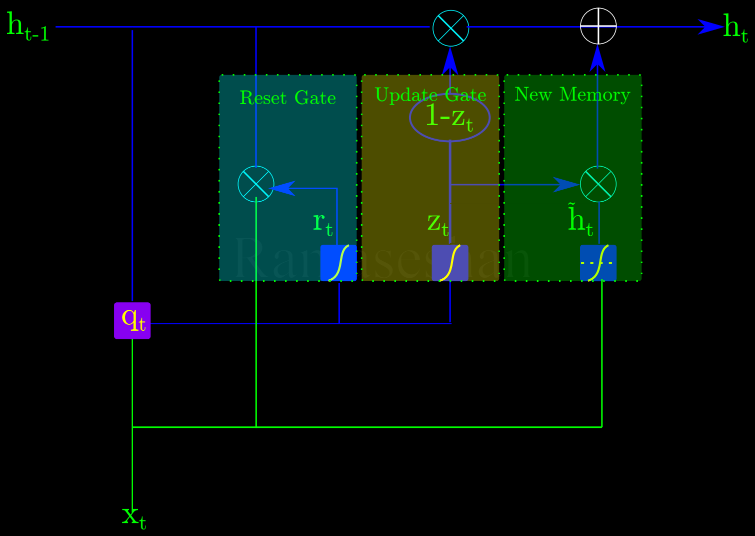
Contains around 270+ problems in Kinematics Divided into 100 characters/sequence
Each sequence is trained and learn to predict the next character (alphabet, punctuations, numbers)

Sample problem

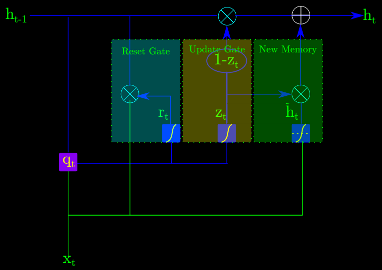
A ball is thrown upward from a bridge with an initial velocity of 5.9 m/s. It strikes water after 2s. If g=9.8m/s2 What is the height of the bridge ?

- 5% training - What is the hid acceleration of the car pasking 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



$$q_t = f(h_{t-1}, x_t) \tag{45}$$

$$z_t = \sigma(U_z, q_t) \tag{46}$$

$$r_t = \sigma(U_r, q_t) \tag{47}$$

$$\tilde{h}_t = \tanh(W.(r_t, q_t)) \tag{48}$$

$$h_t = (1 - z_t) \otimes h_{t-1} \oplus (z_t \otimes \tilde{h}_t) \tag{49}$$

$$s_t = \tanh(h_t) \tag{50}$$

$$\hat{y}_t = softmax(V s_t) \tag{51}$$

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.

REFERENCES

Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural computations* 9.8 (Nov. 1997), pp. 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735. URL: http://dx.doi.org/10.1162/neco.1997.9.8.1735.