Neural Language Model

Ramaseshan Ramachandran

CURSE OF DIMENSIONALITY

In probalistic Language Model, we took advantage of the idea - ***temporally closer words in the word sequence are statistically more dependent***

- A fundamental problem that makes language modelling and other learning problems difficult is the curse of dimensionality
- It is particularly obvious in the case when one wants to model the joint distribution between many discrete random variable
- If one wants to estimate the joint probability distribution of n-grams words in a language with a $|V|$ words as vocabulary, then we need to estimate a maximum of $|V|^n$ free parameters

DISADVANTAGES OF PLM

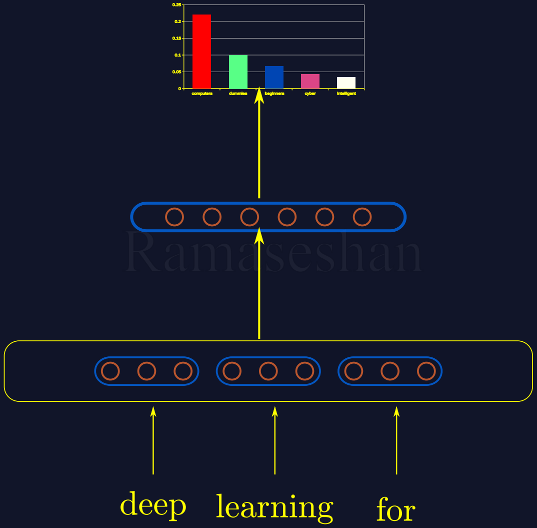
- Scalability wrt increase in context window size
- Markov assumption. It is difficult to predict a word in a longer sequence

  As he is from Tamil Nadu, it is most probable that his mother tongue is ....
- Free parameter size increases exponentially with the context window size
- The context words closer to the word to be estimated are given more importance
- It does not use any semantic or syntactic similarity for the estimation

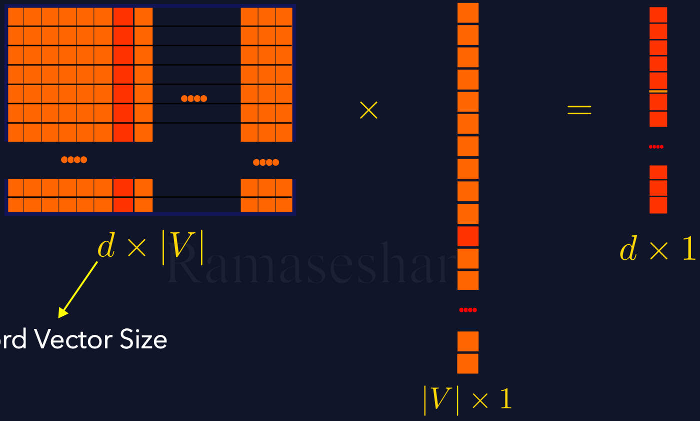
IDEAL LEARNING OF A SEQUENCE

- Associate every word in the vocabulary as a distributed feature vector
- Let the feature vectors represent the joint probability of the n-gram sequences
- Learn the features as parameters to represent the sequence

ANN FOR LM


0.25
0.2
0.15
0.1
0.05
0
computers
dummies
beginners
cyber
intelligent
deep
learning
for

EMBEDDING LAYER CREATION



NEURAL LM MODEL

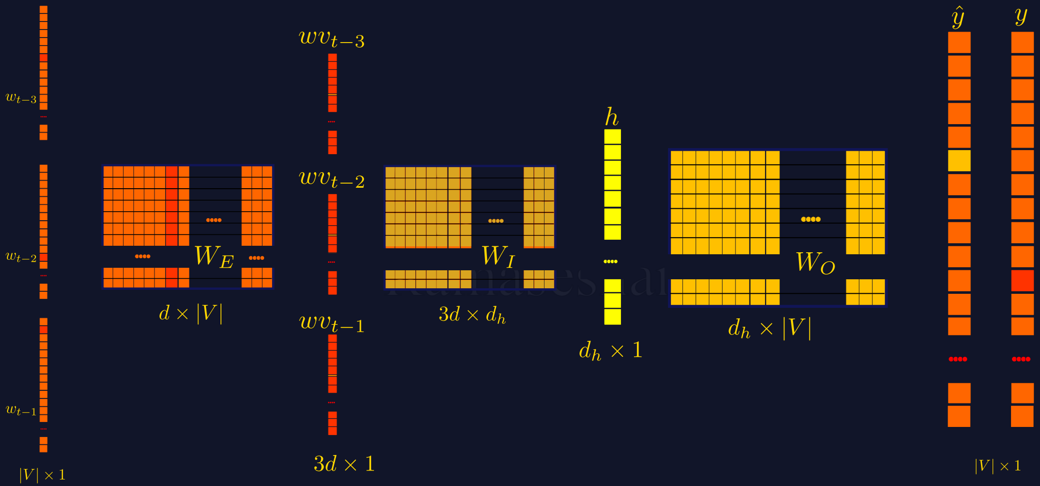


Figure: Adopted from https://web.stanford.edu/~jurafsky/slp3/7.pdf - Chapter 7

FORWARD PASS

- Forward pass is used to compute the probability distribution over the possible outputs.
- Word embedding matrix is used
- One-hot vector is used to get the index of the input word to get the corresponding word vector from the embedding matrix
- Hidden layer values are computed using word vectors of the context words and the input weights W_I
- The output $\hat{y}$ is computed using the hidden layer and the output weights W_O
- Embedding weights W_E is shared among all the context words
- h depends on W_E and W_I
- The parameters of this model are W_E, W_I, h, W_O

LOSS FUNCTION

The loss function measures how much the output $\hat{y}$ differs from the true observation, y. The loss function $\mathscr{E}$ is with respect to the correct output y is a maximum likelihood estimate. The parameters of the models are chosen to maximize the probability given the input-output relationships.

$$\hat{y} = p(y|\text{context words}) \tag{1}$$

We can use cross-entropy as loss function if the estimated result and the target are probability distributions. Since we know in this case that there is only one correct outcome, given the context words, the above equation can take the form[1]

$$p(x) = \begin{cases} p & \text{if } x = 1 \\ 1 - p & \text{if } x = 0 \end{cases} \tag{2}$$

Now, $p(y|\text{context words})$ in equation(1) can be written as

$$p(y|\text{context words}) = \hat{y}^y - (1 - \hat{y})^{1-y} \tag{3}$$

[1]Since there are only two discrete outcomes (1 or 0), this is a Bernoulli distribution

LOSS FUNCTION

Taking log on both sides of equation(3)

$$\begin{aligned}\log(p(y|\text{context words})) &= \log(\hat{y}^y - (1-\hat{y})^{1-y}) \\ &= y\log(\hat{y}) + (1-y)\log(1-\hat{y}) \qquad (4)\end{aligned}$$

By incorporating equation(1), the loss function will be a cross entropy loss function which we need to minimize during the learning process

$$\mathscr{E}(W_E, W_I, h, W_O) = -y\log(\hat{y}) - (1-y)\log(1-\hat{y}) \qquad (5)$$

Replacing $\hat{y} = f(\mathbf{w_O}.\mathbf{h} + b)$ in equation we get

$$\mathscr{E}(W_E, W_I, h, W_O) = -y\log(f(\mathbf{w_O}.\mathbf{h} + b)) - (1-y)\log(1 - f(\mathbf{w_O}.\mathbf{h} + b)) \qquad (6)$$

SIGNIFICANCE OF THE CROSS-ENTROPY LOSS FUNCTION

As we are dealing with one-hot vector as the true output, the equation(5) becomes

$$\mathscr{E}(W_E, W_I, h, W_O) = -y \log(\hat{y})$$

$$= -y \log(f(\mathbf{w_O}.\mathbf{h} + b)) \text{ or}$$

$$\mathcal{E}(W_E, W_I, h, W_O) = -\sum_{d=1}^{|V|} \mathbf{y}_d \log(\hat{\mathbf{y}}_d) \quad (7)$$

$$= -\sum_{d=1}^{|V|} \log(\hat{\mathbf{y}}_d) \quad (8)$$

LEARNING USING CROSS-ENTROPY

During the learning process, if

$$\mathscr{E} = \begin{cases} \text{very small , then the classifier is good - parameters are learned} \\ \text{large, then the classifier is bad or the entropy is large} \\ \text{- Lack of predictability - Needs more training} \end{cases}$$

Since $\log(\hat{y}) \in (0, \inf)$, the loss function in (5) which is a cross-entropy between y and the estimated $\hat{y}$, ensures that the context word to be predicted gets higher probability and the incorrect one gets a minimum probability.

The idea is to

1. Learn the weights for all context and target words
2. Minimize the loss function $\mathscr{E}$
3. Generalize over all training samples (or maximize the input-output relationship)

DRAWBACKS OF TRADITIONAL ANN

- Memory-less and does not bother where the words and context come from
- Traditional Neural networks do not accept arbitrary input length
- Architecture is fixed with respect to the context window or input
- Every context is considered in isolation
- Some important tasks depend on the entire sequence of data
  $y(t+1) = f(x(t), x(t-1), x(t-2) \ldots x(t-n))$

DRAWBACKS OF TRADITIONAL ANN

- Traditional Neural networks are not designed as a state machine
- Anything outside the context window has no impact on the decision being made
- Some NLP tasks require semantic modeling over the whole sentence
- Temporal dependencies are important and are not captured
- Do not or struggle to capture correlations between the temporal elements in sequences

SEQUENCE LEARNING

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

- Helps in modeling complex temporal patterns in the data
- Captures the dependencies and correlations between the different elements in the sequence
- Maintains internal state that captures the context of the previous occurrences of words

Application space

- Machine translation
- Question answering, char-bots
- Text summarization
- Setiment Analysis
- Predicting future values in financial time-series data
- Detecting anomalies in sensor data

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 missing text
- Correct Hand-written text

BASIC IDEAS FROM QUANTUM COMPUTING

- Quantum computing uses the principles of quantum mechanics to process information.
- Quantum computers use ***quantum bits or qubits***.
- Qubit can be in a state of superposition - it can be both 0 and 1 at the same time.
- **Entanglement** is aother fundamental concept
- Two or more quantum systems are correlated if the state of one system is dependent on the state of the other(s), even when separated by large distances.

Can this be used in handling temporal information?
Can we represent homonyms/polysemous/capitonyms words using a qubit vectors?

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 next state depends on the previous states
- In order to use the previous state, we need to store it or remember it
- 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
  - Analogy - $Chennai : Tamil :: London : English$ or $go : went :: run : Ran$ or $queen \approx king - man + woman$

A SIMPLE RECURRENT NEURAL NETWORK

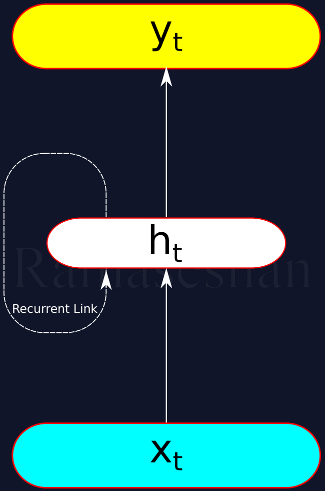
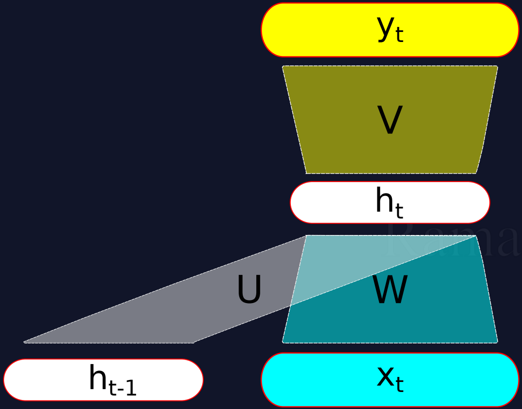


Figure: A simple Recurrent Neural Network

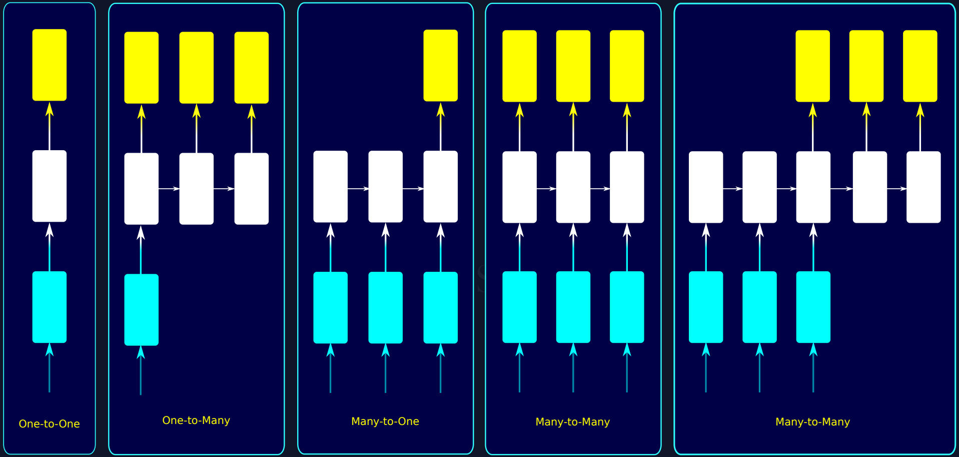
RNN - AN EXTENSION OF A FEED-FORWARD NETWORK



- 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 $t-1$

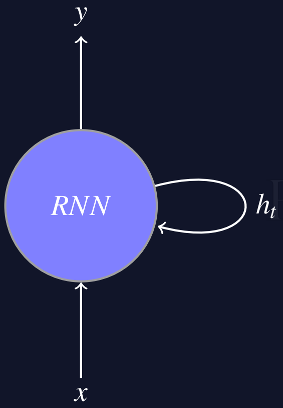
MULTIPLE ARCHITECTURES OF RNN



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--effec:veness/

RECURRENT NEURON



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

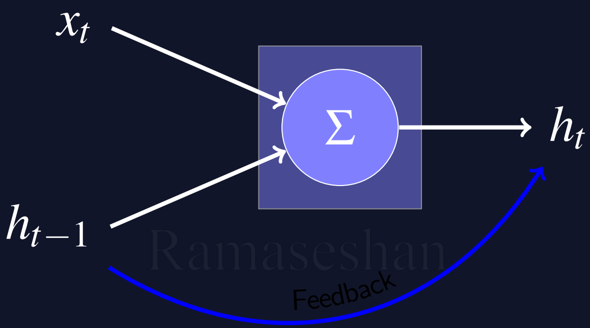
$h_t = f_W(h_{t_1}, x_t)$, where

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

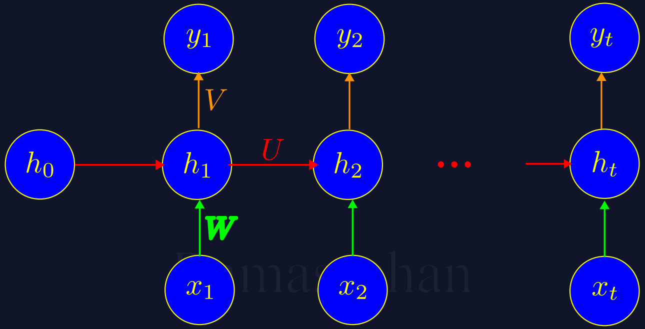


$$h_t = f(U * h_{t-1} + W * x_t)$$

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.

$$h_0 = \sigma(Wx_0) \tag{9}$$

$$h_1 = \sigma(Uh_0 + Wx_1) \tag{10}$$

$$\dots \tag{11}$$

$$h_n = f(Uh_{n-1} + Wx_n), \forall n \tag{12}$$

$$y_n = V * h_n, \text{ where } n = 1, 2, \dots N \tag{13}$$

RNN UNROLLED IN TIME

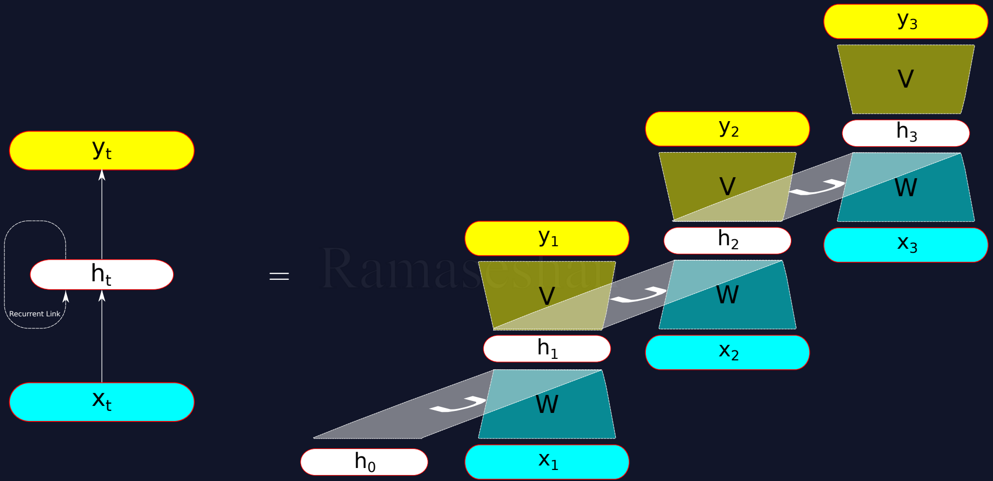
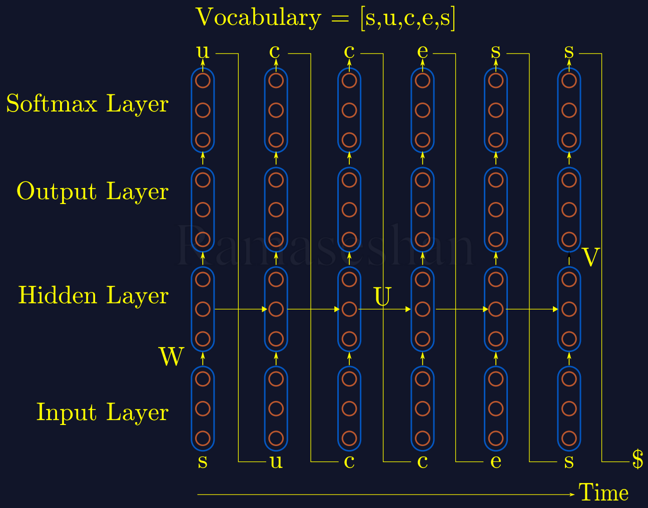
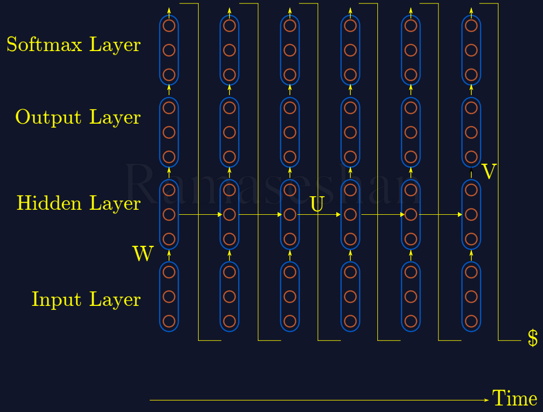


Figure: RNN Unrolled in time

CHARACTER BASED LM - RNN



LANGUAGE MODEL - RNN



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

RNN TRAINING- BACK PROPAGATION THROUGH TIME

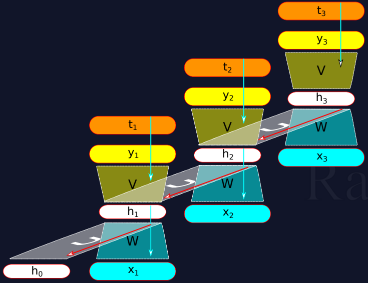


Figure: RNN Training using backpropagation

$$z^{|1|} = Wx \tag{14}$$

$$a^{|1|} = g(z^{1}) \tag{15}$$

$$z^{|2|} = Ua^{|1|} \tag{16}$$

$$a^{|2|} = g(z^{|2|}) \tag{17}$$

$$y = f(Va^{|2|}) \tag{18}$$

$$\frac{\partial L}{\partial V} = \frac{\partial L}{\partial a}\frac{\partial a}{\partial z}\frac{\partial z}{\partial V} \tag{19}$$

$$\delta h = g^{'}(z)V\delta_{out} + \delta_{next} \tag{20}$$

RNN - TRAINING - BACKWARD PASS

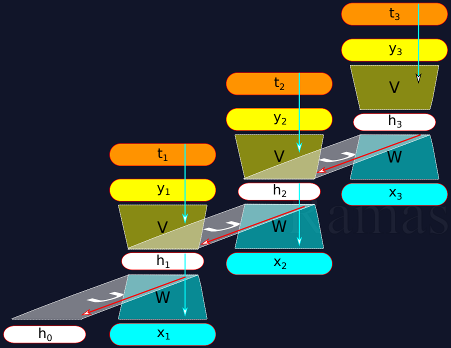


Figure: RNN Training using backpropagation

$$\delta_{out} = \frac{\partial L}{\partial a}\frac{\partial a}{\partial z} \tag{21}$$

$$\frac{\partial L}{\partial V} = \delta_{out} h_t \tag{22}$$

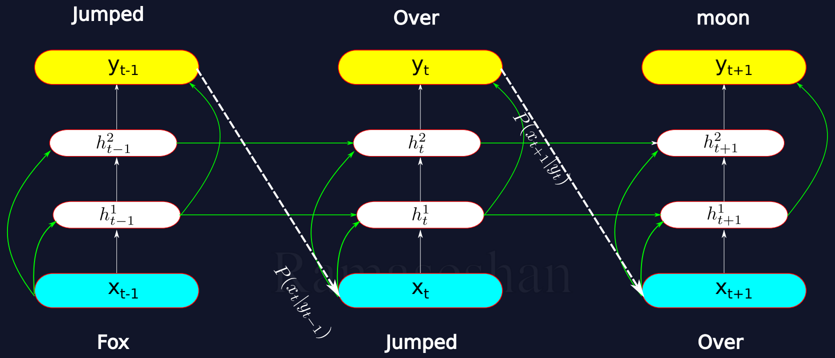
$$\frac{\partial L}{\partial W} = \delta h x_t \tag{23}$$

$$\frac{\partial L}{\partial U} = \delta h h_{t-1} \tag{24}$$

ADVANTAGES AND TROUBLE WITH RNN

- Theoretically, it is possible to store all historical information in the RNN
- Vanishing gradient problem - The diminishing value of δ 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?

RNN FOR PREDICTING THE NEXT WORD



The input vector sequence $\mathbf{x} = (x_1, x_2, x_3, ..., x_T)$
The output vector sequence $\mathbf{y} = (y_1, y_2, y_3, \ldots, y_T)$
The hidden vector sequence $\mathbf{h}^n = (h^n_1, h^n_2, h^n_3, \ldots, h^n_T)$

$$h^1_t = g(W_{ih^1} x_t + W h^1 h^1 h^1_{t-1} + b^1_h)$$
$$h^n_t = g(W_{ih^n} x_t + W h^{n-1} h^n h^{n-1}_{t-1} + W h^n h^n h^n_{t-1} + b^n_h)$$

$$\hat{y} = b_y + \sum_{n=1}^{N} W_{h^n y} h^n_t$$
$$y_t = f(\hat{y}_t)$$

$$P(\mathbf{x}) = \prod_{t=1}^{T} P(x_{t+1}|y_t)$$
$$L(\mathbf{x}) = -\sum_{t=1}^{T} \log P(x_{t+1}|y_t)$$

LONG SHORT-TERM MEMORY (LSTM)

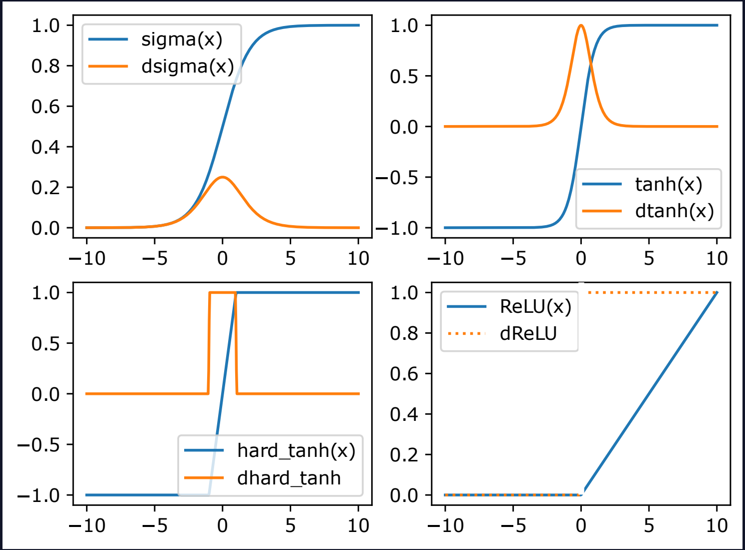
Learning to store information over extended time intervals via recurrent takes a very long time, mostly due to insufficient, decaying error back flow [1]

- 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
$y = \left(\frac{1}{1+e^{-x}}\right)$	$\frac{dy}{dx} = \left(\frac{1}{1+e^{-x}}\right)\left(1 - \frac{1}{1+e^{-x}}\right) = \sigma(x)(1-\sigma(x))$
$y = \tanh(x) = \frac{sinh(x)}{cosh(x)}$	$\frac{dy}{dx} = \frac{\cosh(x)\cosh(x) - \sinh(x)\sinh(x)}{\cosh^2(x)} = (1 - \tanh^2(x))$
$y = \max(c,x)$	$\frac{dy}{dx} = \begin{cases} 1, & if\ x > 0 \\ c, & if\ x \leq 0 \end{cases}$
	For RELU, $c = 0$ and $\frac{dy}{dx}$ does not exist at $x = 0$

DERIVATIVES OF SIGMOID, TANH, HARD TANH, RELU



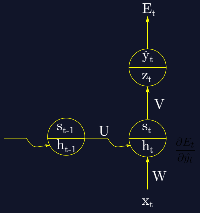
WHY tanh?

tanh is symmetrical around the origin.

- Zero -centring of $tanh$ allows faster convergence during training process
- $\partial(\tanh)$ is steep at the origin or gradients are larger when compared to sigmoid function
- Allows efficient backpropagation of errors during the training
- Speeds up the learning process or updates weights faster than sigmoid
- $\partial(\sigma)$ has a range $[0, 0.25]$. Changes slowly at the origin - may result in very slow learning at around the origin
- $\partial(tanh)$ should be able to capture small changes
- $\partial(\tanh)$ has a range $[0, 1]$. The rapid change may minimize the problem of vanishing gradient

Is Sigmoid function symmetrical around origin?

FORWARD PASS - NETWORK EQUATIONS



Forward pass

$$h_t = Wx_t + Uh_{t-1} \quad (25)$$
$$s_t = \tanh(h_t) \quad (26)$$
$$z_t = Vs_t \quad (27)$$
$$\hat{y}_t = softmax(z_t) \quad (28)$$
$$E = -\sum_t y_t \log(\hat{y}_t) \quad (29)$$

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

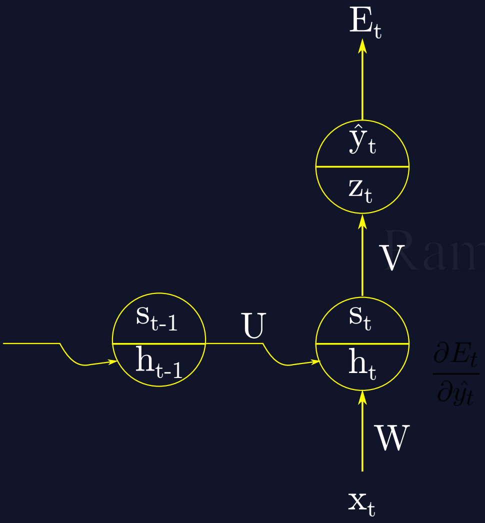
- $x_t \in R^{D_{|w|}}$ represents the input word at time t and D_w is the dimension of the word vector. If one-hot vector, it will be $x^{D_{|V|}}$
- $W \in R^{D_w \times D_h}$ is the weight matrix that conditions the input vector
- $U \in R^{D_h \times D_h}$ matrix that keeps the dependency of the 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

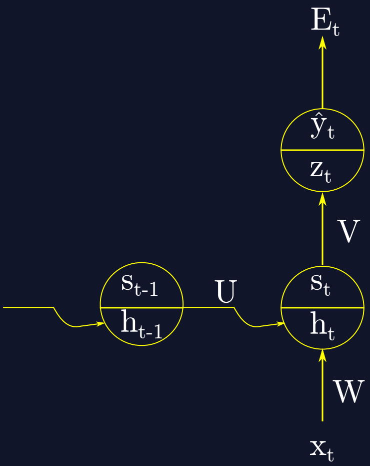
BPTT - DERIVATIVE FOR V



$$h_t = Wx_t + Uh_{t-1}$$
$$s_t = \tanh(h_t)$$
$$z_t = Vs_t$$
$$\hat{y}_t = softmax(z_t)$$
$$E_t = -y_t \log(\hat{y}_t)$$
$$\frac{\partial E_t}{\partial V} = \frac{\partial E_t}{\partial \hat{y}_t}\frac{\partial \hat{y}}{\partial z_t}\frac{\partial z_t}{\partial V} \quad (30)$$
$$\text{Let } \delta^t_{out} = \frac{\partial E_t}{\partial \hat{y}_t}\frac{\partial \hat{y}}{\partial z_t}$$
$$\frac{\partial E_t}{\partial V} = \delta^t_{out} s_t \quad (31)$$

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

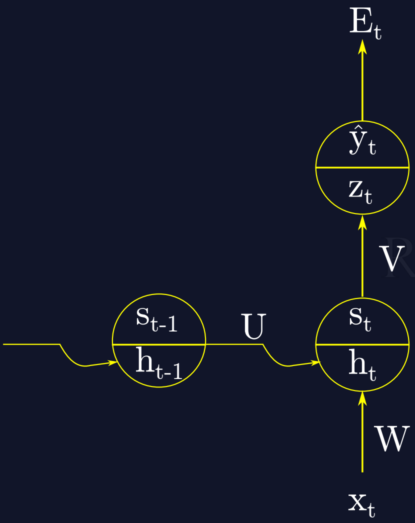
BPTT - DERIVATIVE FOR W



$$\frac{\partial E_t}{\partial W} = \underbrace{\frac{\partial E_t}{\partial \hat{y}_t}\frac{\partial \hat{y}}{\partial z_t}}\frac{\partial z_t}{\partial s_t}\frac{\partial s_t}{\partial h_t}\frac{\partial h_t}{\partial W} = \delta_{out}^{t} V \sigma'(h_t) x_t \tag{32}$$

Since the hidden layer activation depends on the previous time state, we have another similar term δ_{t-1} that get added to (32)

BPTT - DERIVATIVE FOR U



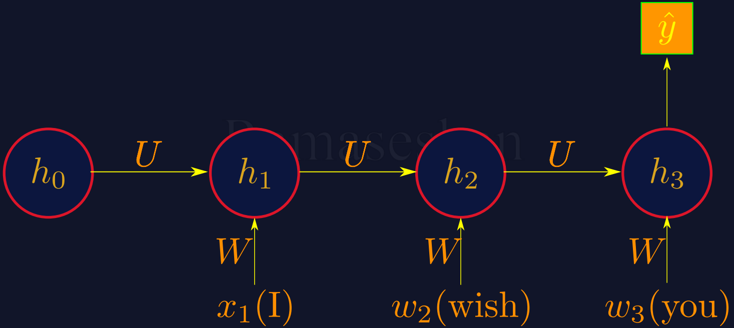
$$\frac{\partial E_t}{\partial U} = \underbrace{\frac{\partial E_t}{\partial \hat{y}_t}\frac{\partial \hat{y}}{\partial z_t}\frac{\partial z_t}{\partial s_t}\frac{\partial s_t}{\partial h_t}}\frac{\partial h_t}{\partial U} \quad (33)$$

$$= \delta^t_{out} V \sigma^{'}(h_t) h_{t-1} \quad (34)$$

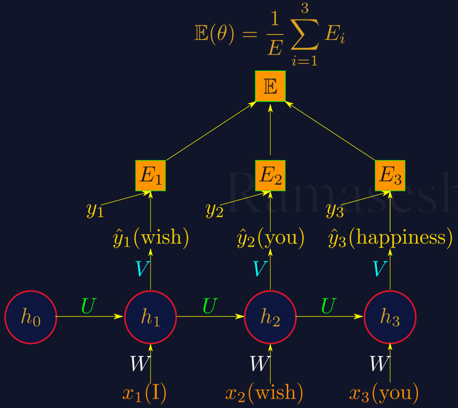
Since we are back propagating the error from the current state to the previous state, $\delta_{next} = \sigma(h_t) U \delta^t_{out} V \sigma^{'}(h_t)$ needs to be added

ERROR COMPUTATION - 1

$$\mathbb{E} = y - \hat{y}$$



ERROR COMPUTATION - 2



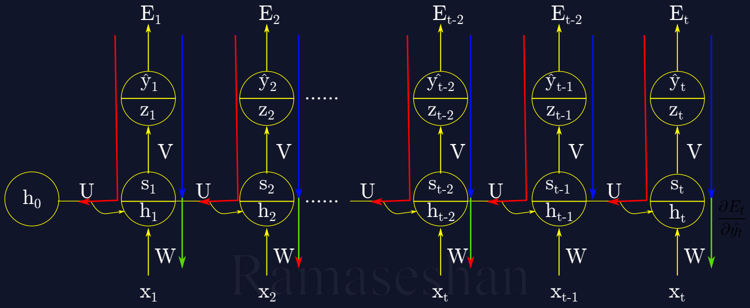
$$\mathbb{E}(\theta) = -\frac{1}{T}\sum_{1}^{T}\sum_{j=1}^{|V|} y_{t,j}\log(\hat{y}_{t,j})$$

We need to compute the following derivatives

$$\frac{\partial \mathbb{E}}{\partial \hat{y}_t}, \frac{\partial \mathbb{E}}{\partial V}, \frac{\partial \mathbb{E}}{\partial W}, \frac{\partial \mathbb{E}}{\partial h_t}, \text{ and } \frac{\partial \mathbb{E}}{\partial U}$$

$$h_t \bowtie h_1^{t-1}$$

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

$$\mathbb{E}(\theta) = -\frac{1}{T}\sum_{1}^{T}\sum_{j=1}^{|V|} y_{t,j}\log(\hat{y}_{t,j}) \tag{35}$$

If the value of the perplexity, $(2^{\mathbb{E}(\theta)})$ lower, then the confidence of the network in predicting the next word is higher

PERPLEXITY

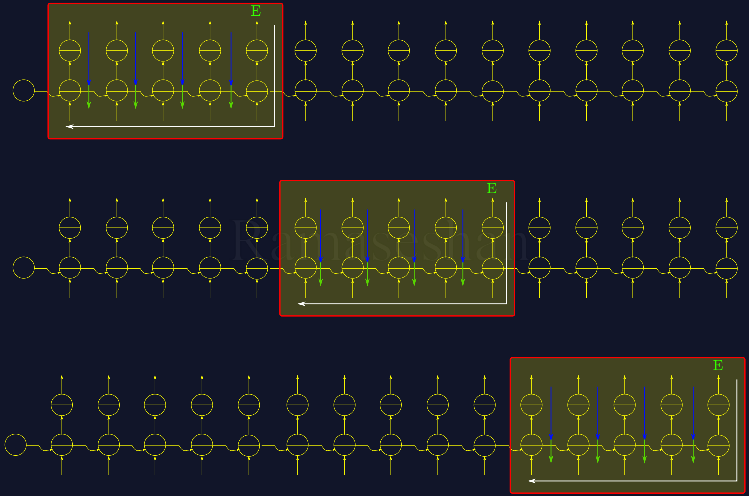
Perplexity is a measurement of how well a model predicts a sample. Perplexity is defined as

$$\text{For bigram model, } PP(W_N) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_{i-1})}} \tag{36}$$

$$\text{For trigram model } PP(W_N) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_{i-1}w_{i-2})}} \tag{37}$$

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

TBPTT



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.

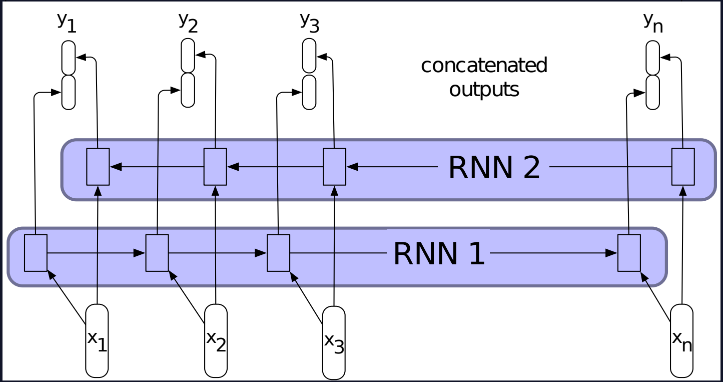
Hyper-parameter

- *batch* - the number of samples or subset of the training dataset
- *Epoch* - complete pass through the entire training dataset

BI-DIRECTIONAL RNN

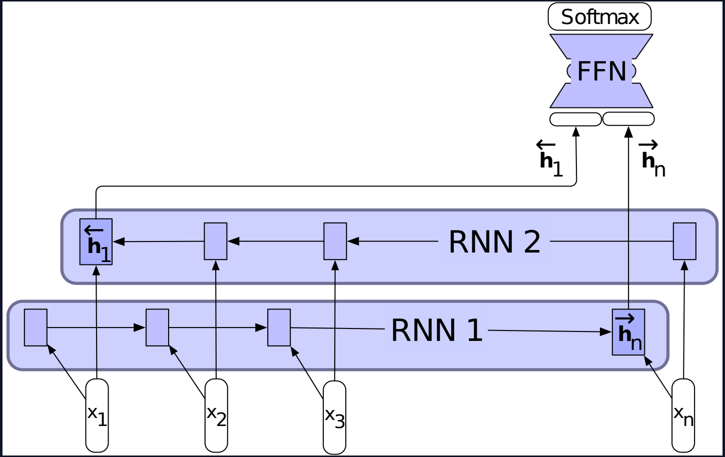
- Capture the word meaning as well as its contextual information to enable different word embeddings for the same word -contextualized word embedding
- The language model does not only have a predict the next word, but also the previous word.

BIDIRECTIONAL RNN WITH MULTIPLE OUTPUT



A bidirectional RNN. Separate models are trained in the forward and backward directions, with the output of each model at each time point concatenated to represent the bidirectional state at that time point.

BIDIRECTIONAL RNN WITH SINGLE OUTPUT


Softmax
FFN
$\overleftarrow{\mathbf{h}}_1$
$\overrightarrow{\mathbf{h}}_n$
RNN 2
$\overleftarrow{\mathbf{h}}_1$
RNN 1
$\overrightarrow{\mathbf{h}}_n$
x_1
x_2
x_3
x_n

A bidirectional RNN for sequence classif cation. The f nal hidden units from the forward and backward passes are combined to represent the entire sequence. This combined representation serves as input to the subsequent classif er.

RNN - KINEMATICS PROBLEM GENERATION

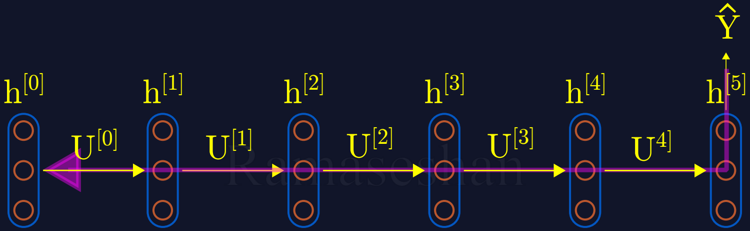
Contains around 270+ problems in Kinematics Divided into 100 characters/sequence
Each sequence is trained and learn to predict the next character (alphabet, punctuations, numbers)

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 ?

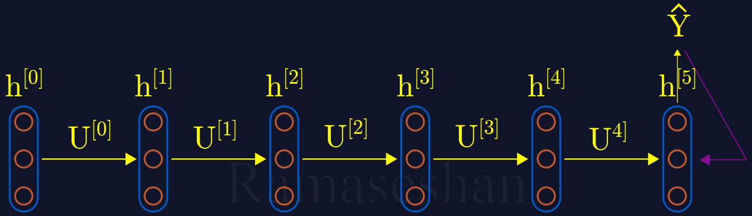
- 5% training - What is the hid acceleration of the car pasking the car ski distance. A croosts from the wate it stop
- 25% training - What is the distance constant reach and aft when it hits the same ball. A ball is thrown out of a velo
- **Recall** - ”determine the time it takes a piece of glass to 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

EXPLODING/VANISHING GRADIENT



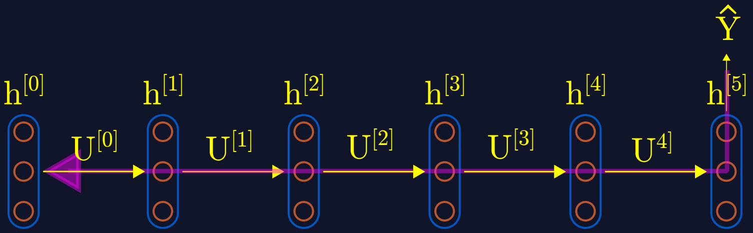
Let us assume that $\hat{y}_t = f(U,h)$. We want to propagate the error through back propagation

EXPLODING/VANISHING GRADIENT



The error E at $5th$ time state depends on $h^{[5]}$. Hence we need to estimate $\dfrac{\partial E}{\partial h^{[5]}}$. $h^{[5]}$ 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



$$\frac{\partial E^{[5]}}{\partial h^{[0]}} = \frac{\partial E^{[5]}}{\partial h^{[5]}} \times \frac{\partial h^{[5]}}{\partial h^{[4]}} \times \frac{\partial h^{[4]}}{\partial h^{[3]}} \times \frac{\partial h^{[3]}}{\partial h^{[2]}} \times \frac{\partial h^{[2]}}{\partial h^{[1]}} \times \frac{\partial h^{[1]}}{\partial h^{[0]}} = \frac{\partial E^{[5]}}{\partial h^{[5]}} \prod_{t=1}^{t=5} \frac{\partial h^{[t]}}{\partial h^{[t-1]}} \tag{38}$$

Generalizing

$$\frac{\partial E^{[\tau]}}{\partial h^{[0]}} = \frac{\partial E^{[\tau]}}{\partial h^{[\tau]}} \prod_{t=1}^{t=\tau} \frac{\partial h^{[t]}}{\partial h^{[t-1]}} \tag{39}$$

where τ represents depth of the layers or the size of the time series

EXPLODING/VANISHING GRADIENT

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

$$h^{[t]} = \sigma\left(WX^{[t]} + Uh^{[t-1]}\right) \tag{40}$$

$$\frac{\partial h^{[t]}}{\partial h^{[t-1]}} = diag(\sigma'(WX^{[t-1]} + Uh^{[t-1]}))U \tag{41}$$

where σ' computes element-wise the derivative of σ

$$\frac{\partial h^{[t]}}{\partial h^{[t-1]}} \text{ is a Jacobian} \tag{42}$$

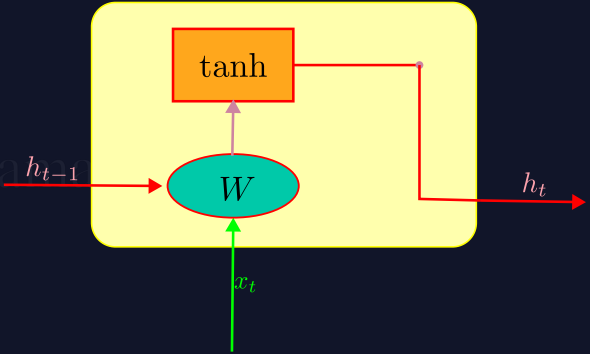
$$\therefore \frac{\partial E^{[\tau]}}{\partial h^{[0]}} = \frac{\partial E^{[\tau]}}{\partial h^{[\tau]}} U^{\tau} \prod_{t=1}^{t=\tau} diag(\sigma'(WX^{[t-1]} + Uh^{[t-1]})) \tag{43}$$

Values of U become very small when the depth increases[2]

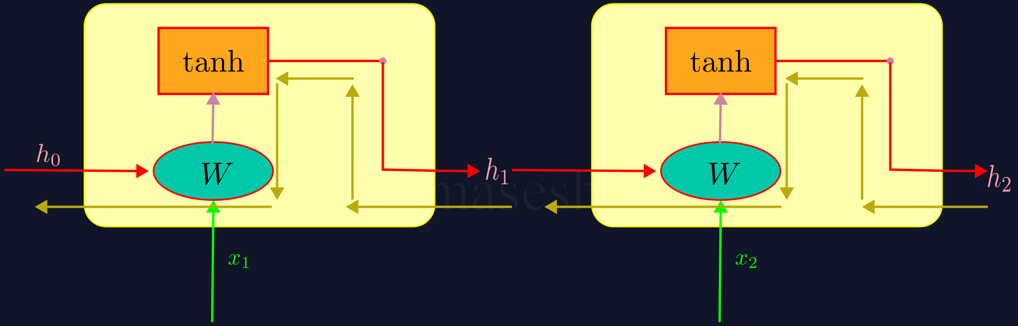
[2]Source: "On the difficulty of training recurrent neural networks", Pascanuet al, 2013 - http://proceedings.mlr.press/v28/pascanu13.pdf

ANOTHER REPRESENTATION

$$
\begin{aligned}
h_t &= \tanh(W_{hh}h_{t-1} + W_{xh}x_t) \\
&= \tanh\left(\begin{pmatrix} W_{hh} & W_{hx} \end{pmatrix} \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right) \\
&= \tanh\left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right) \quad (44)
\end{aligned}
$$



GRADIENT FLOW IN RNN



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 end of the back propagation. 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 $0.5^{47} = 7.1054274e-15$

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

PROBLEMS WITH VANILLA RNN

- The component of the gradient in directions that correspond to long-term dependencies is small[3]
- 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

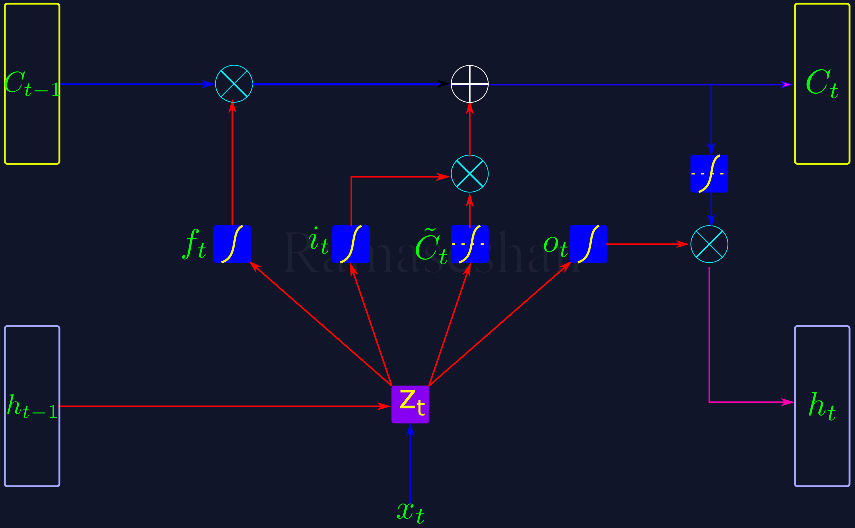
[3] An empirical exploration of recurrent network architectures - http://dl.acm.org/citation.cfm?id=3045118.3045367

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[4]
- 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

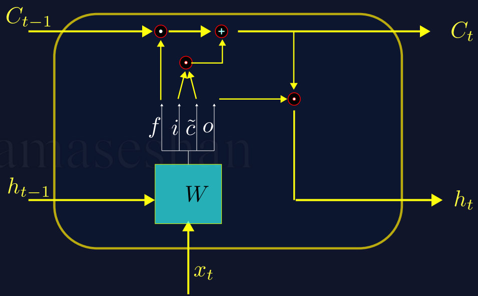
[4] http://dblp.uni-trier.de/db/journals/corr/corr1506.html#KarpathyJL15

LSTM CELL

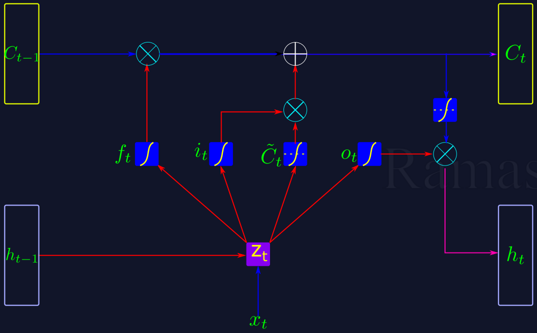


SIMPLE REPRESENTATION

$$\begin{pmatrix} i \\ f \\ \tilde{C} \\ o \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \tanh \\ \sigma \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \tag{45}$$



LSTM - FORWARD PASS



$$f_t = \sigma(W_{ft} q_t + b_f) \tag{46}$$

$$i_t = \sigma(W_{it} q_t + b_i) \tag{47}$$

$$\tilde{C}_t = \tanh(W_{\tilde{C}_t q_t}) \tag{48}$$

$$C_t = (f_t \otimes C_{t-1}) \oplus (i_t \otimes \tilde{C}_t) \tag{49}$$

$$o_t = \sigma(W_{ot} q_t + b_o) \tag{50}$$

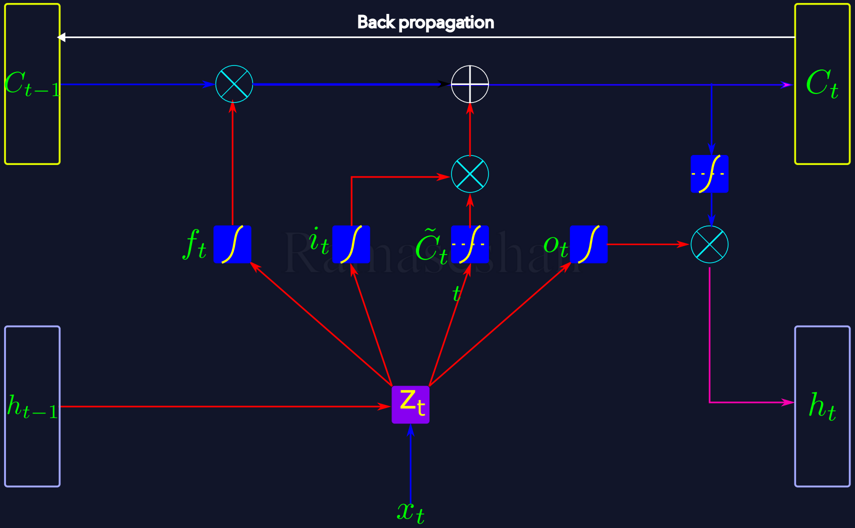
$$h_t = o_t \otimes \tanh(C_t) \tag{51}$$

$$s_t = \tanh(h_t) \tag{52}$$

$$z_t = V z_t \tag{53}$$

$$\hat{y}_t = softmax(z_t) \tag{54}$$

BACK PROPAGATION



WHAT BRAIN RETAINS AND FORGETS?

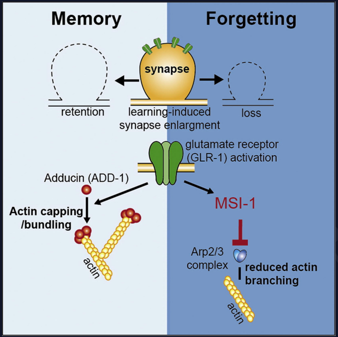


Figure: Musashi protein

- Repetition of events
- Primacy and recency
- Surprise
- Emotional Impact
- Positive or negative actions
- Hypocrisy
- ...

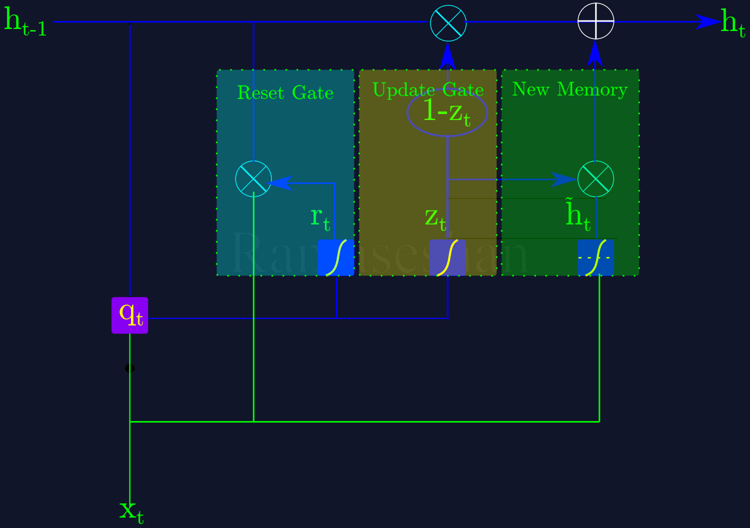
Memory length is regulated cooperatively through the activation of adducin (add-1) and by the inhibitory effect of msi-1. Brain forgets unimportant information in order to remain efficient Can RNN be trained to simulate the actions of adducin and musashi proteins?

GATING MECHANISM

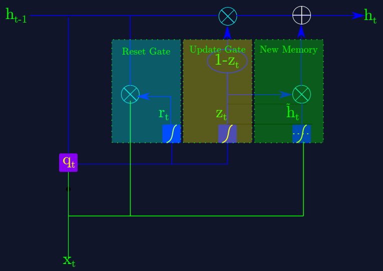
We require a slowly-decaying error propagation. In other words, the update of weights should enhance/retain distributed properties of a sequence.

- Control the flow of information
- Should the new/old information be allowed or dropped?
- Gates are capable of interrupting, or allowing, the passage of activation values among neurons in the hidden layer
- The ability to manage the flow of information may play a key role arresting or setting up a well-behaved gradient during the back-propagation
- Multiplicative gates provide the protection of memory contents from decaying
- Multiplicative input and output gates protect contents from perturbations

INTRODUCTION TO GATED RECURRENT UNIT



GRU FORWARD PASS



$$q_t = f(h_{t-1}, x_t) \tag{55}$$
$$z_t = \sigma(U_z, q_t) \tag{56}$$
$$r_t = \sigma(U_r, q_t) \tag{57}$$
$$\tilde{h}_t = \tanh(W.(r_t, q_t)) \tag{58}$$
$$h_t = (1 - z_t) \otimes h_{t-1} \oplus (z_t \otimes \tilde{h}_t) \tag{59}$$
$$s_t = \tanh(h_t) \tag{60}$$
$$\hat{y}_t = softmax(V s_t) \tag{61}$$

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

[1] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Comput.* 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.