CS626: Speech, NLP and the Web

RNN, Seq2seq, Data Driven Machine Translation (SMT and NMT)

Pushpak Bhattacharyya

Computer Science and Engineering Department

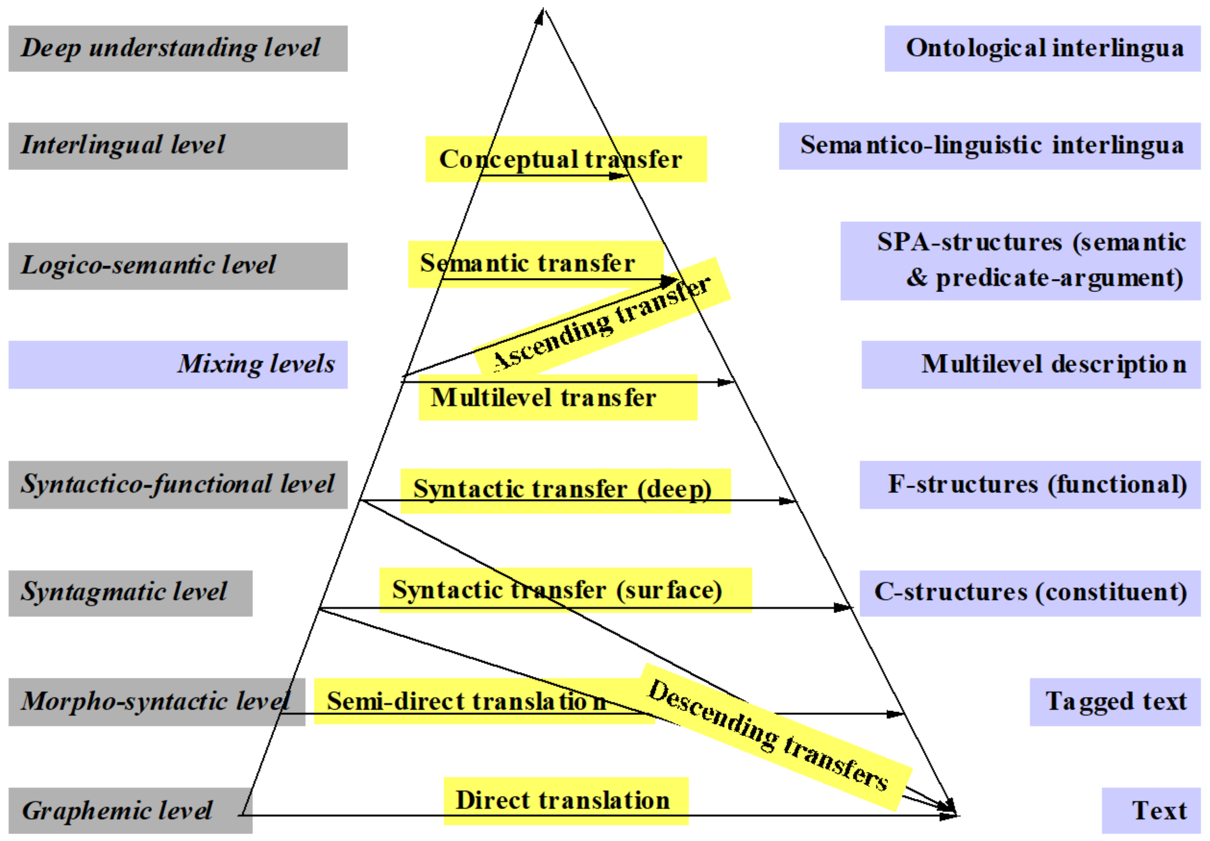
IIT Bombay

Week of 16th November, 2020

Vauquois Triangle

Kinds of MT Systems

(point of entry from source to the target text)



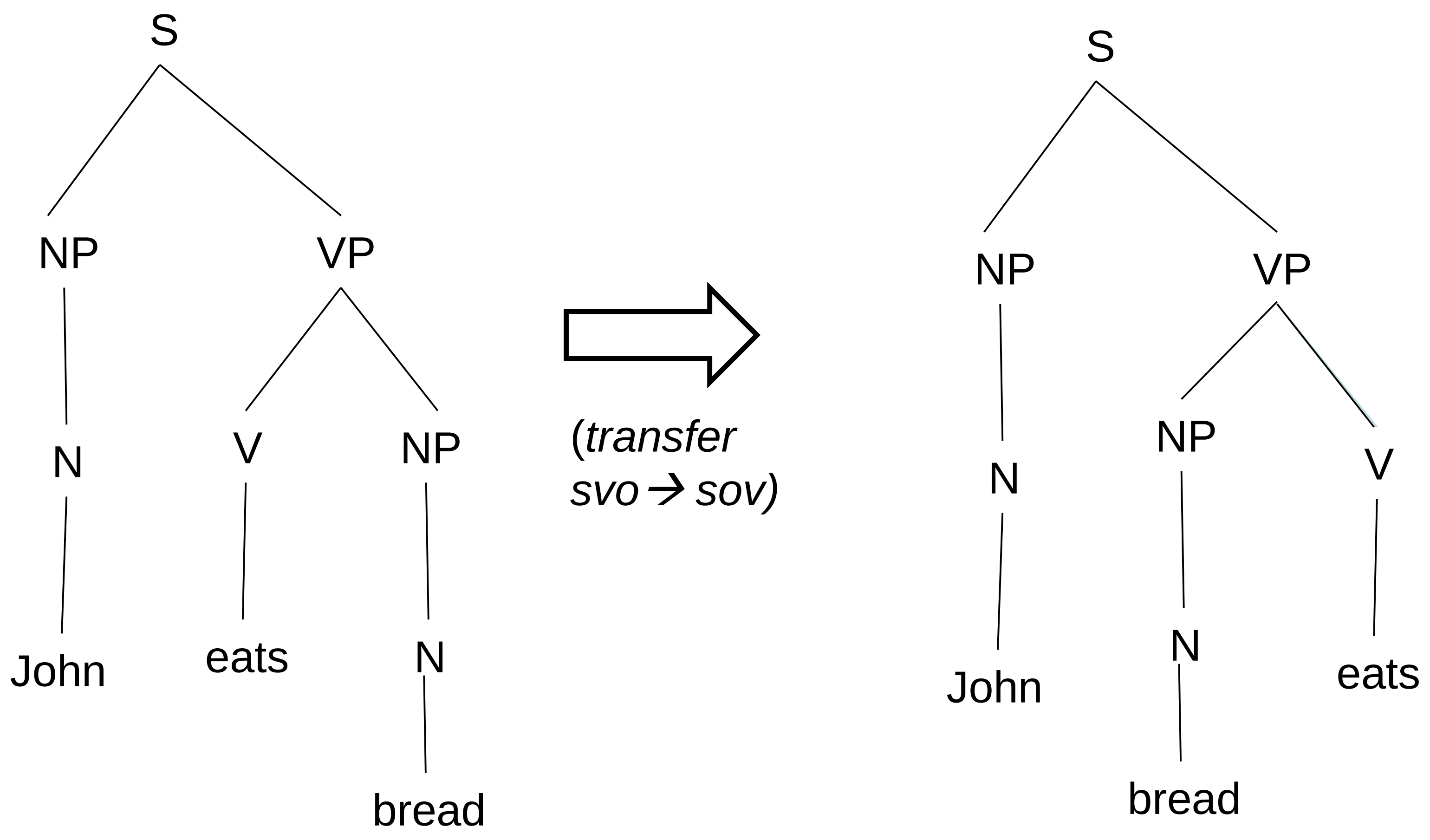
Pair of languages decides the level of analysis: Syncretism in Bengali languages

- Syncretism: overloading in the functionality of morphemes
- Bengali has more **syncretism** than hindi
- It is more challenging to get morpheme mapping
- Example
  - Bai**be**: **will** carry
  - will: Morpheme “be” in bengali

Full Ambiguity resolution is not always needed: for translation

- Example: Semantic role ambiguity
    - *Mujhe apko mithai khilani padegi*
        - Ambiguous sentence
        - Semantic role ambiguity, who is the agent and who the beneficiary
        - Who is giving the sweets to whom
- For translation to
    - English
        - Ambiguity resolution is necessary
    - Bengali/Marathi/Gujrati/Assamese
        - Ambiguity resolution is not necessary

Illustration of transfer SVO→SOV



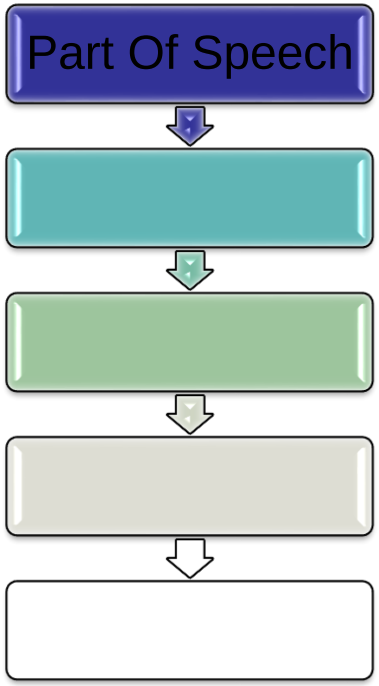
Fundamental processes in Machine Translation

- **Analysis**
  - Analysis of the source language to represent the source language in more disambiguated form
    - Morphological segmentation, POS tagging, chunking, parsing, discourse resolution, pragmatics etc.
- **Transfer**
  - Representation transfer from one language to another
  - Example: SOV to SVO conversion
- **Generation**
  - Generate the final target sentence
  - Final output is text, intermediate representations can include F-structures, C-structures, tagged text etc.

Issues to handle

Sentence: *I went with my friend, John, to the bank to withdraw some money but was disappointed to find it closed.*

ISSUES



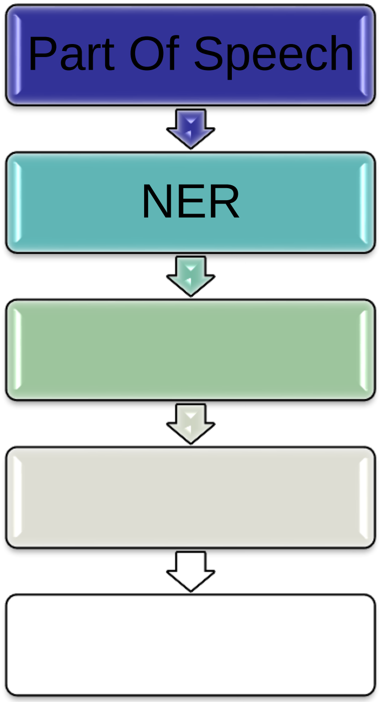
Noun or Verb

Issues to handle

Sentence: *I went with my friend, John, to the bank to withdraw some money but was disappointed to find it closed.*

John is the name of a PERSON

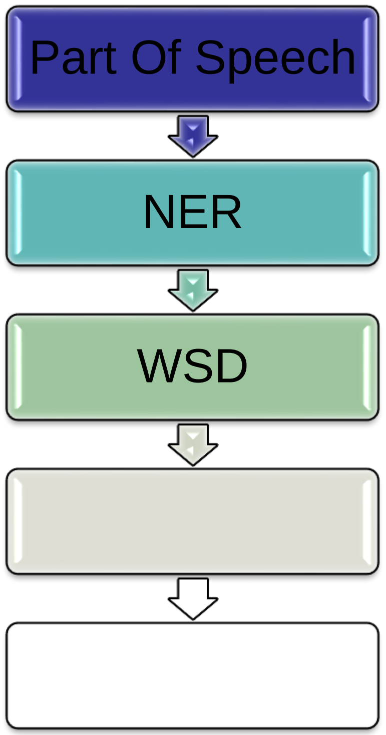
ISSUES



Issues to handle

Sentence: *I went with my friend, John, to the bank to withdraw some money but was disappointed to find it closed.*

ISSUES

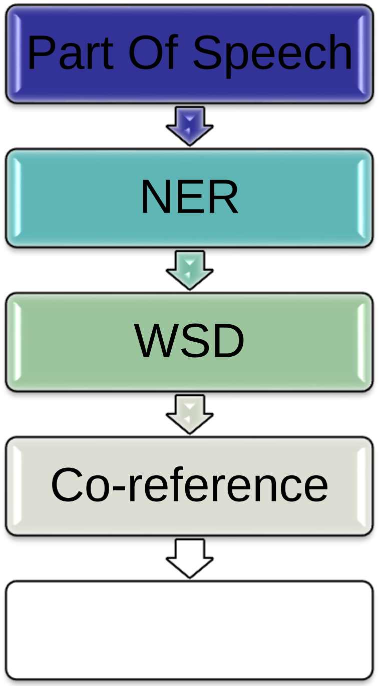


Financial bank or River bank

Issues to handle

Sentence: *I went with my friend, John, to the bank to withdraw some money but was disappointed to find it closed.*

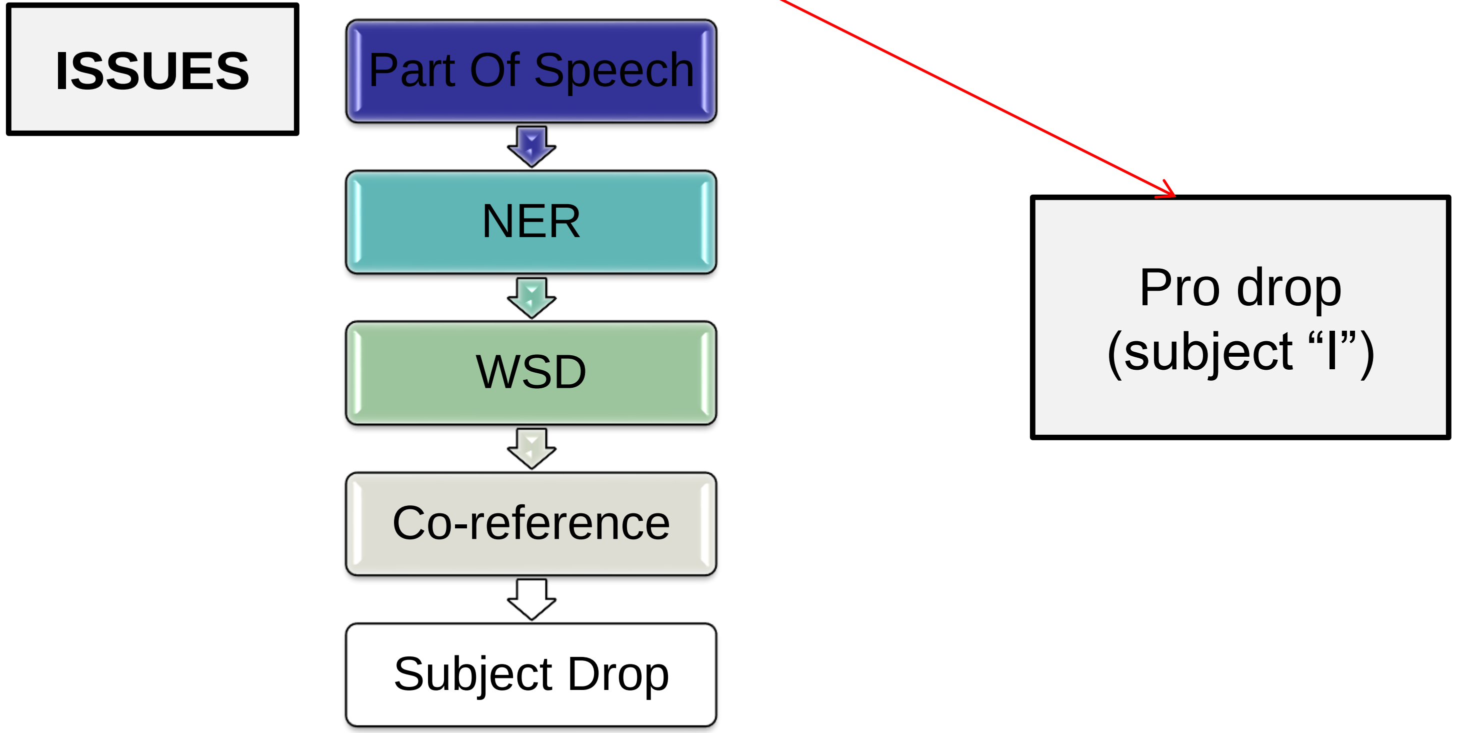
ISSUES



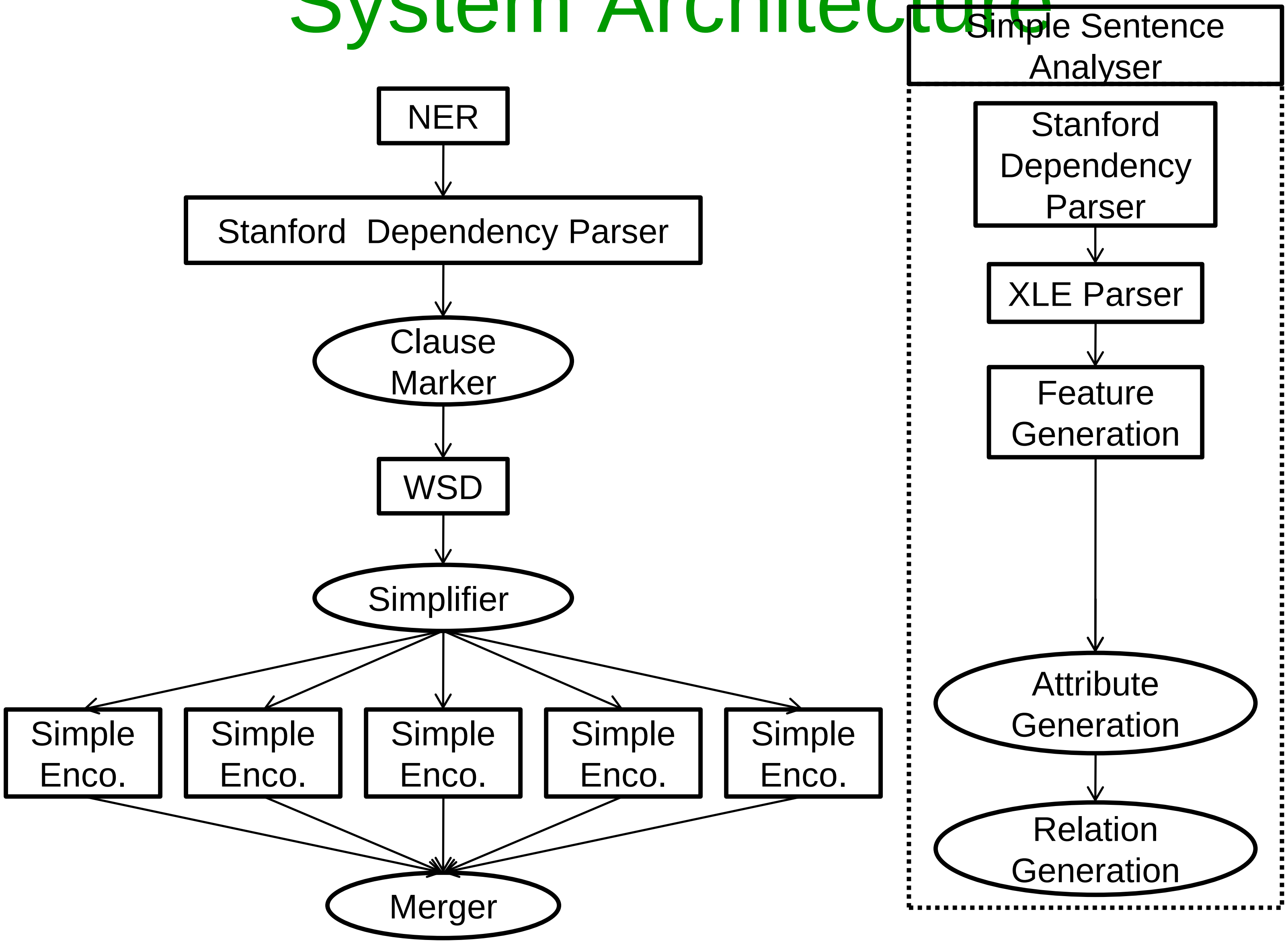
“it” → “bank” .

Issues to handle

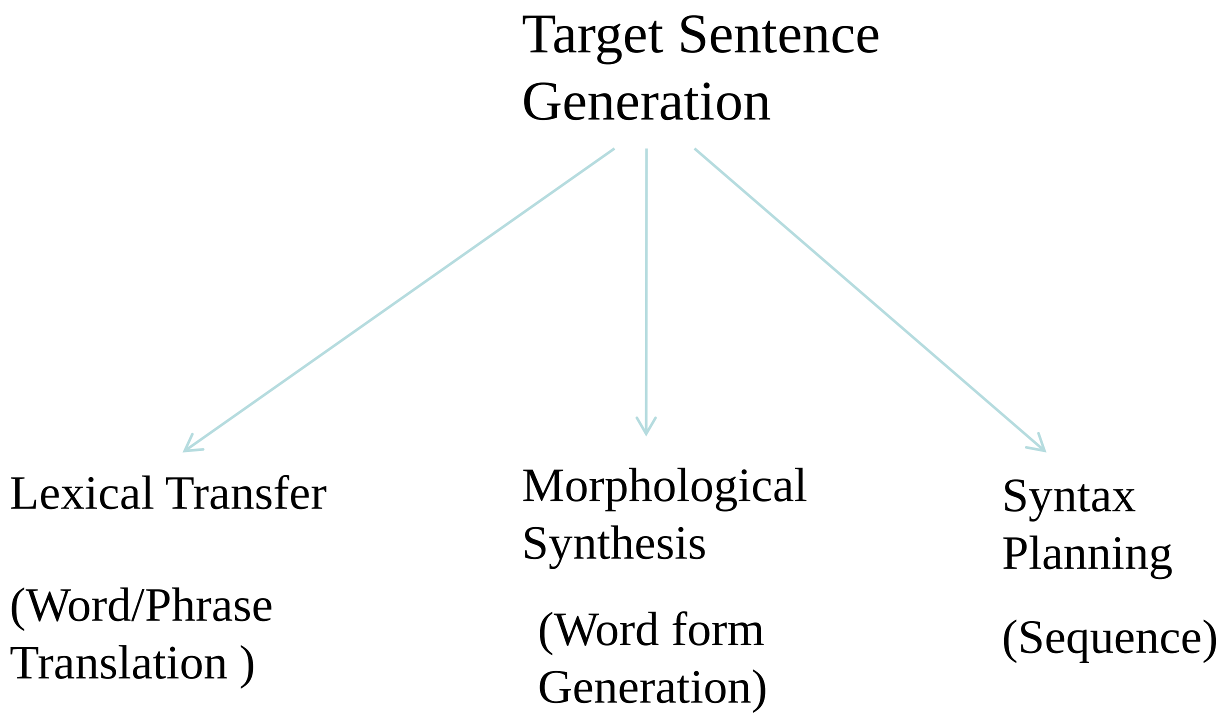
Sentence: *I went with my friend, John, to the bank to withdraw some money but was disappointed to find it closed.*



System Architecture

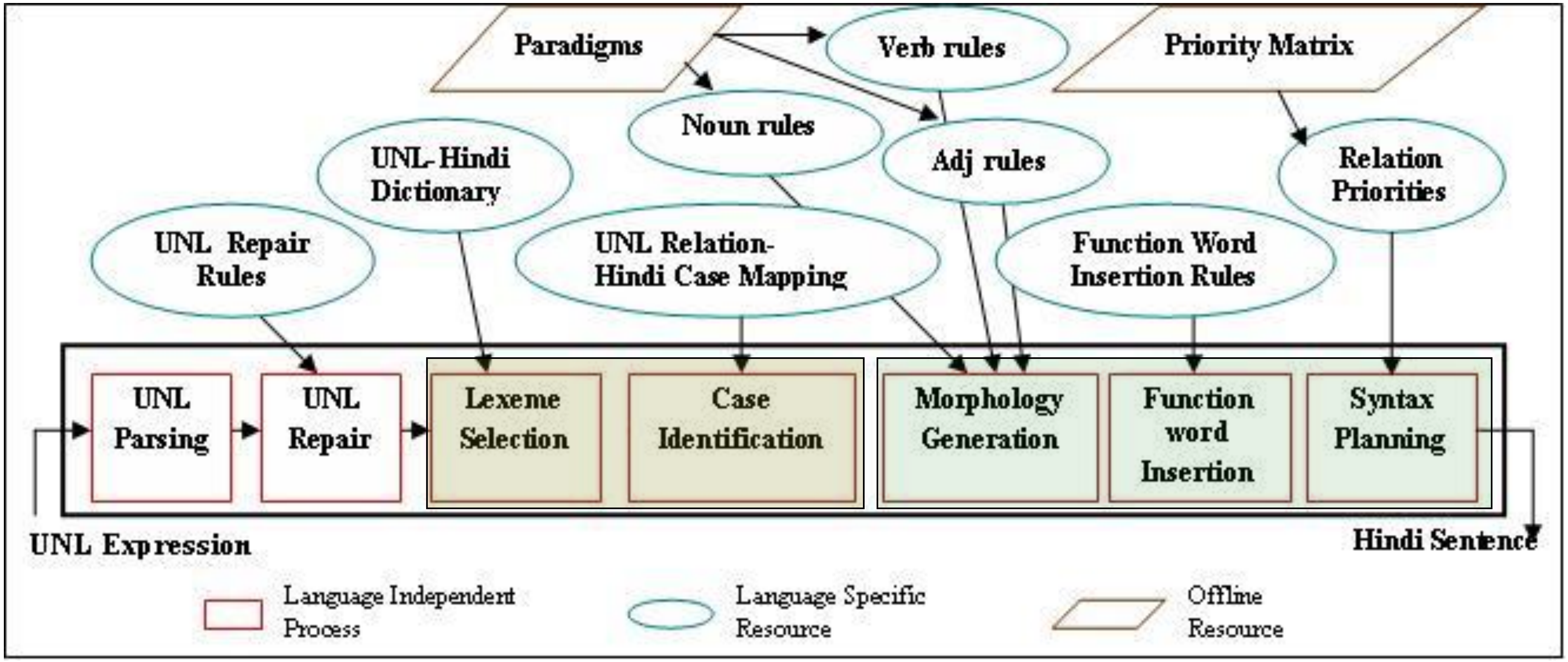


Target Sentence Generation from interlingua

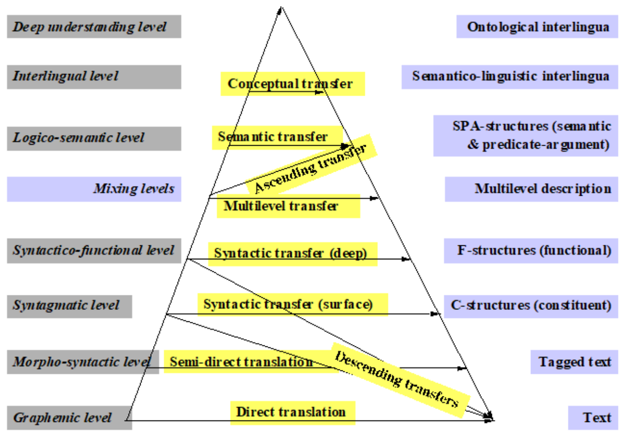


Generation Architecture

Deconversion = Transfer + Generation



Statistical Machine Translation



Czeck-English data

• [nesu]	“I carry”
• [ponese]	“He will carry”
• [nese]	“He carries”
• [nesou]	“They carry”
• [yedu]	“I drive”
• [plavou]	“They swim”

To translate …

- I will carry.
- They drive.
- He swims.
- They will drive.

Hindi-English data

• [DhotA huM]	“I carry”
• [DhoegA]	“He will carry”
• [DhotA hAi]	“He carries”
• [Dhote hAi]	“They carry”
• [chalAtA huM]	“I drive”
• [tErte hEM]	“They swim”

Bangla-English data

• [bai]	“I carry”
• [baibe]	“He will carry”
• [bay]	“He carries”
• [bay]	“They carry”
• [chAlAi]	“I drive”
• [sAMtrAy]	“They swim”

To translate … (repeated)

- I will carry.
- They drive.
- He swims.
- They will drive.

Foundation

- Data driven approach
- Goal is to find out the English sentence *e* given foreign language sentence *f* whose *p(e|f)* is maximum.

$$\tilde{e} = \underset{e \in e^*}{\operatorname{argmax}}\, p(e|f) = \underset{e \in e^*}{\operatorname{argmax}}\, p(f|e)p(e)$$

- Translations are generated on the basis of statistical model
- Parameters are estimated using bilingual parallel corpora

SMT: Language Model

- To detect *good* English sentences
- Probability of an English sentence $w_1 w_2 \ldots\ldots w_n$ can be written as

$$Pr(w_1 w_2 \ldots\ldots w_n) = Pr(w_1) * Pr(w_2|w_1) * \ldots * Pr(w_n|w_1\, w_2 \ldots w_{n-1})$$

- Here $Pr(w_n|w_1\, w_2 \ldots w_{n-1})$ is the probability that word w_n follows word string $w_1\, w_2 \ldots w_{n-1}$.
  - N-gram model probability
- Trigram model probability calculation

$$p(w_3|w_1 w_2) = \frac{count(w_1 w_2 w_3)}{count(w_1 w_2\,)}$$

SMT: Translation Model

- *P*(*f*|*e*): Probability of some *f* given hypothesis English translation *e*
- How to assign the values to *p*(*e*|*f*) ?

$$p(f|e) = \frac{count(f,e)}{count(e)} \longleftarrow$$

Sentence level

  - Sentences are infinite, not possible to find pair(e,f) for all sentences

- Introduce a hidden variable ***a***, that represents alignments between the individual words in the sentence pair

$$\Pr(\boldsymbol{f}|\boldsymbol{e}) = \sum_{\boldsymbol{a}} \Pr(\boldsymbol{f}, \boldsymbol{a}|\boldsymbol{e}) \longleftarrow$$

Word level

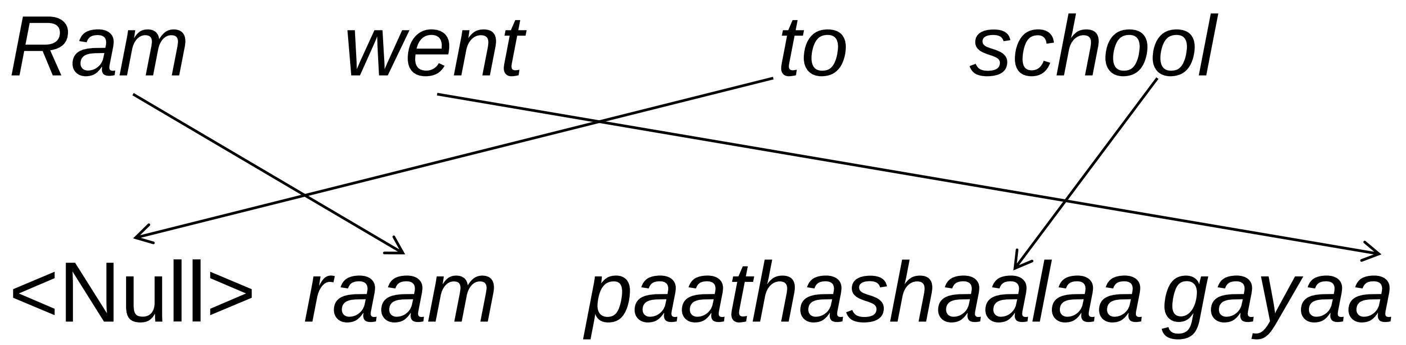
Alignment

- If the string, $e= e_1^l= e_1\ e_2\ \ldots e_l$, has l words, and the string, $f= f_1^m=f_1 f_2 \ldots f_m$, has m words,
- then the alignment, a, can be represented by a series, $\boldsymbol{a_1^m= a_1 a_2 \ldots a_m}$, of m values, each between 0 and l such that if the word in position j of the f-string is connected to the word in position i of the e-string, then
  - $\boldsymbol{a_j= i}$, and
  - if it is not connected to any English word, then $\boldsymbol{a_j= O}$

Example of alignment

English: *Ram went to school*

Hindi: *raam paathashaalaa gayaa*



Translation Model: Exact expression

$$\Pr(\boldsymbol{f}, \boldsymbol{a} | \boldsymbol{e}) = \Pr(m | \boldsymbol{e}) \prod_{j=1}^{m} \Pr\left(a_j \middle| a_1^{j-1}, f_1^{j-1}, m, \boldsymbol{e}\right) \Pr\left(f_j \middle| a_1^{j}, f_1^{j-1}, m, \boldsymbol{e}\right)$$

Choose the length of foreign language string given *e*

Choose alignment given *e* and *m*

Choose the identity of foreign word given *e, m, a*

- Five models for estimating parameters in the expression [2]
- Model-1, Model-2, Model-3, Model-4, Model-5

Proof of Translation Model: Exact expression

$$\Pr(f \mid e) = \sum_a \Pr(f, a \mid e)$$; marginalization

$$\Pr(f, a \mid e) = \sum_m \Pr(f, a, m \mid e)$$; marginalization

$$\Pr(f, a, m \mid e) = \sum_m \Pr(m \mid e) \Pr(f, a \mid m, e)$$

$$= \sum_m \Pr(m \mid e) \Pr(f, a \mid m, e)$$

$$= \sum_m \Pr(m \mid e) \prod_{j=1}^{m} \Pr(f_j, a_j \mid a_1^{j-1}, f_1^{j-1}, m, e)$$

$$= \sum_m \Pr(m \mid e) \prod_{j=1}^{m} \Pr(a_j \mid a_1^{j-1}, f_1^{j-1}, m, e) \Pr(f_j \mid a_1^{j}, f_1^{j-1}, m, e)$$

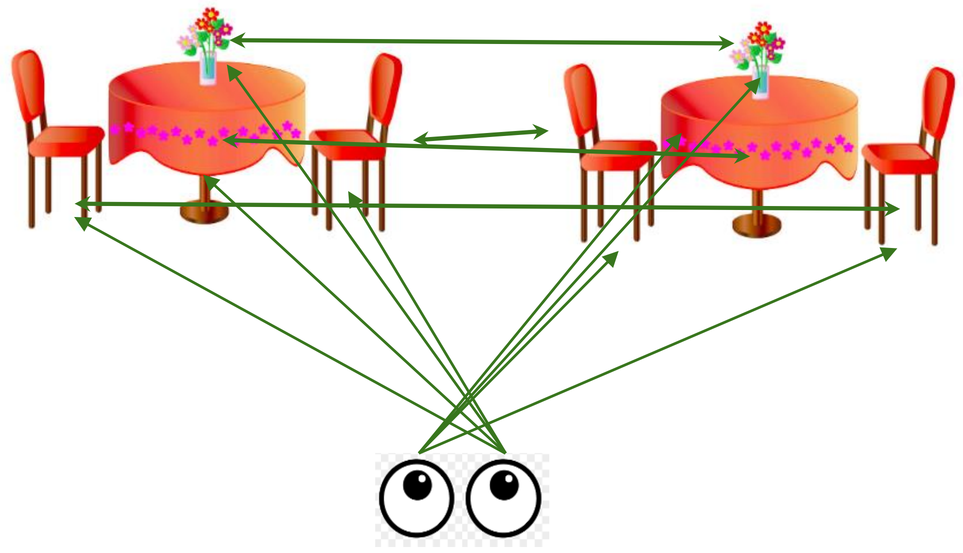
m is fixed for a particular *f*, hence

$$\Pr(f, a, m \mid e) = \Pr(m \mid e) \prod_{j=1}^{m} \Pr(a_j \mid a_1^{j-1}, f_1^{j-1}, m, e) \Pr(f_j \mid a_1^{j}, f_1^{j-1}, m, e)$$

Alignment

How to build part alignment from whole alignment

- Two images are in alignment: images on the two retina
- Need to find alignment of parts of it



Fundamental and ubiquitous

- Spell checking
- Translation
- Transliteration
- Speech to text
- Text to speeh

EM for word alignment from sentence alignment: example

English	French
(1) three rabbits	(1) trois lapins
a b	w x
(2) rabbits of Grenoble	(2) lapins de Grenoble
b c d	x y z

Initial Probabilities:
each cell denotes *t(a←→w), t(a←→x) etc.*

	a	b	c	d
w	1/4	1/4	1/4	1/4
x	1/4	1/4	1/4	1/4
y	1/4	1/4	1/4	1/4
z	1/4	1/4	1/4	1/4

Example of expected count

C[w←→a; (a b)←→(w x)]

$$= \frac{t(w\leftrightarrow a)}{t(w\leftrightarrow a)+t(w\leftrightarrow b)} \times \#(a \text{ in 'a b'}) \times \#(w \text{ in 'w x'})$$

$$= \frac{1/4}{1/4+1/4} \times 1 \times 1 = 1/2$$

"counts"

a b ←→ ***w x***	a	b	c	d
w	1/2	1/2	0	0
x	1/2	1/2	0	0
y	0	0	0	0
z	0	0	0	0

b c d ←→ ***x y z***	a	b	c	d
w	0	0	0	0
x	0	1/3	1/3	1/3
y	0	1/3	1/3	1/3
z	0	1/3	1/3	1/3

Revised probability: example

$$t_{revised}(a \leftrightarrow w)$$

$$= \frac{1/2}{(1/2+1/2+0+0)_{(a\ b)\leftrightarrow(w\ x)} + (0+0+0+0)_{(b\ c\ d)\leftrightarrow(x\ y\ z)}}$$

Revised probabilities table

	a	b	c	d
w	1/2	1/2	0	0
x	1/4	5/12	1/6	1/6
y	0	1/3	1/3	1/3
z	0	1/3	1/3	1/3

“revised counts”

a b* ←→ *w x	a	b	c	d
w	1/2	3/8	0	0
x	1/2	5/8	0	0
y	0	0	0	0
z	0	0	0	0

b c d* ←→ *x y z	a	b	c	d
w	0	0	0	0
x	0	5/9	1/3	1/3
y	0	2/9	1/3	1/3
z	0	2/9	1/3	1/3

Re-Revised probabilities table

	a	b	c	d
w	1/2	1/2	0	0
x	3/16	**85/144**	1/9	1/9
y	0	1/3	1/3	1/3
z	0	1/3	1/3	1/3

Continue until convergence; notice that (b,x) binding gets progressively stronger; b=rabbits, x=lapins

Derivation of EM based Alignment Expressions

$$V_E = \text{vocalbulary of language } L_1 \text{ (Say English)}$$

$$V_F = \text{vocabulary of language } L_2 \text{ (Say Hindi)}$$

E1 *what is in a name ?*
नाम में क्या है ?
naam meM kya hai ?
F1 *name in what is ?*

E2 *That which we call rose, by any other name will smell as sweet.*
जिसे हम गुलाब कहते हैं, और भी किसी नाम से उसकी कुशबू समान मीठा होगी
F2 *Jise hum gulab kahte hai, aur bhi kisi naam se uski khushbu samaan mitha hogii*
That which we rose say , any other name by its smell as sweet
That which we call rose, by any other name will smell as sweet.

Vocabulary mapping

Vocabulary

V_E	V_F
what , is , in, a , name , that, which, we , call ,rose, by, any, other, will, smell, as, sweet	*naam, meM, kya, hai, jise, ham, gulab, kahte, aur, bhi, kisi, bhi, uski, khushbu, saman, mitha, hogii*

Key Notations

English vocabulary : V_E
French vocabulary : V_F
No. of observations / sentence pairs : S
Data D which consists of S observations looks like,

$$e^1{}_1, e^1{}_2, \dots, e^1{}_{l^1} \Leftrightarrow f^1{}_1, f^1{}_2, \dots, f^1{}_{m^1}$$

$$e^2{}_1, e^2{}_2, \dots, e^2{}_{l^2} \Leftrightarrow f^2{}_1, f^2{}_2, \dots, f^2{}_{m^2}$$

.....

$$e^s{}_1, e^s{}_2, \dots, e^s{}_{l^s} \Leftrightarrow f^s{}_1, f^s{}_2, \dots, f^s{}_{m^s}$$

.....

$$e^S{}_1, e^S{}_2, \dots, e^S{}_{l^S} \Leftrightarrow f^S{}_1, f^S{}_2, \dots, f^S{}_{m^S}$$

No. words on English side in s^{th} sentence : l^s
No. words on French side in s^{th} sentence : m^s
$index_E(e^s{}_p)$ =Index of English word $e^s{}_p$in English vocabulary/dictionary
$index_F(f^s{}_q)$ =Index of French word $f^s{}_q$in French vocabulary/dictionary

(Thanks to Sachin Pawar for helping with the maths formulae processing)

Hidden variables and parameters

Hidden Variables (Z) :
Total no. of hidden variables = $\sum_{s=1}^{S} l^s \, m^s$ where each hidden variable is as follows:
$z_{pq}^{s} = 1$, if in s^{th} sentence, p^{th} English word is mapped to q^{th} French word.
$z_{pq}^{s} = 0$, otherwise

Parameters (*Θ*) :
Total no. of parameters = $|\, V_E| \times |V_F|$, where each parameter is as follows:
$P_{i,j}$ = Probability that i^{th} word in English vocabulary is mapped to j^{th} word in French vocabulary

Likelihoods

Data Likelihood *L(D; Θ)* :

$$L(D;\Theta) = \prod_{s=1}^{S}\prod_{p=1}^{l^s}\prod_{q=1}^{m^s}\left(P_{index_E(e_p^s),index_F(f_q^s)}\right)^{z_{pq}^s}$$

Data Log-Likelihood LL(D; Θ) :

$$LL(D;\Theta) = \sum_{s=1}^{S}\sum_{p=1}^{l^s}\sum_{q=1}^{m^s} z_{pq}^s \log\left(P_{index_E(e_p^s),index_F(f_q^s)}\right)$$

Expected value of Data Log-Likelihood E(LL(D; Θ)) :

$$E(LL(D;\Theta)) = \sum_{s=1}^{S}\sum_{p=1}^{l^s}\sum_{q=1}^{m^s} E(z_{pq}^s)\log\left(P_{index_E(e_p^s),index_F(f_q^s)}\right)$$

Constraint and Lagrangian

$$\sum_{j=1}^{|V_F|} P_{i,j} = 1 \ , \forall i$$

$$\sum_{s=1}^{S} \sum_{p=1}^{l^s} \sum_{q=1}^{m^s} E(z_{pq}^s) \, log\left(P_{index_E(e_p^s), index_F(f_q^s)}\right) - \sum_{i=1}^{|V_E|} \lambda_i \left(\sum_{j=1}^{|V_F|} P_{i,j} - 1 \right)$$

Differentiating wrt P_{ij}

$$\sum_{s=1}^{S}\sum_{p=1}^{l^s}\sum_{q=1}^{m^s}\delta_{index_E(e_p^s),i}\,\delta_{index_F(f_q^s),j}\left(\frac{E(z_{pq}^s)}{P_{i,j}}\right)-\lambda_i=0$$

$$P_{i,j}=\frac{1}{\lambda_i}\sum_{s=1}^{S}\sum_{p=1}^{l^s}\sum_{q=1}^{m^s}\delta_{index_E(e_p^s),i}\,\delta_{index_F(f_q^s),j}\,E(z_{pq}^s)$$

$$\sum_{j=1}^{|V_F|}P_{i,j}=1=\sum_{j=1}^{|V_F|}\frac{1}{\lambda_i}\sum_{s=1}^{S}\sum_{p=1}^{l^s}\sum_{q=1}^{m^s}\delta_{index_E(e_p^s),i}\,\delta_{index_F(f_q^s),j}\,E(z_{pq}^s)$$

Final E and M steps

M-step

$$P_{i,j} = \frac{\sum_{s=1}^{S}\sum_{p=1}^{l^s}\sum_{q=1}^{m^s} \delta_{index_E(e_p^s),i}\,\delta_{index_F(f_q^s),j} E(z_{pq}^s)}{\sum_{j=1}^{|V_F|}\sum_{s=1}^{S}\sum_{p=1}^{l^s}\sum_{q=1}^{m^s} \delta_{index_E(e_p^s),i}\,\delta_{index_F(f_q^s),j} E(z_{pq}^s)}, \forall i,j$$

E-step

$$E(z_{pq}^s) = \frac{P_{index_E(e_p^s),index_F(f_q^s)}}{\sum_{q'=1}^{m^s} P_{index_E(e_p^s),index_F(f_{q'}^s)}}, \forall s,p,q$$

Recurrent Neural Network

Acknowledgement:

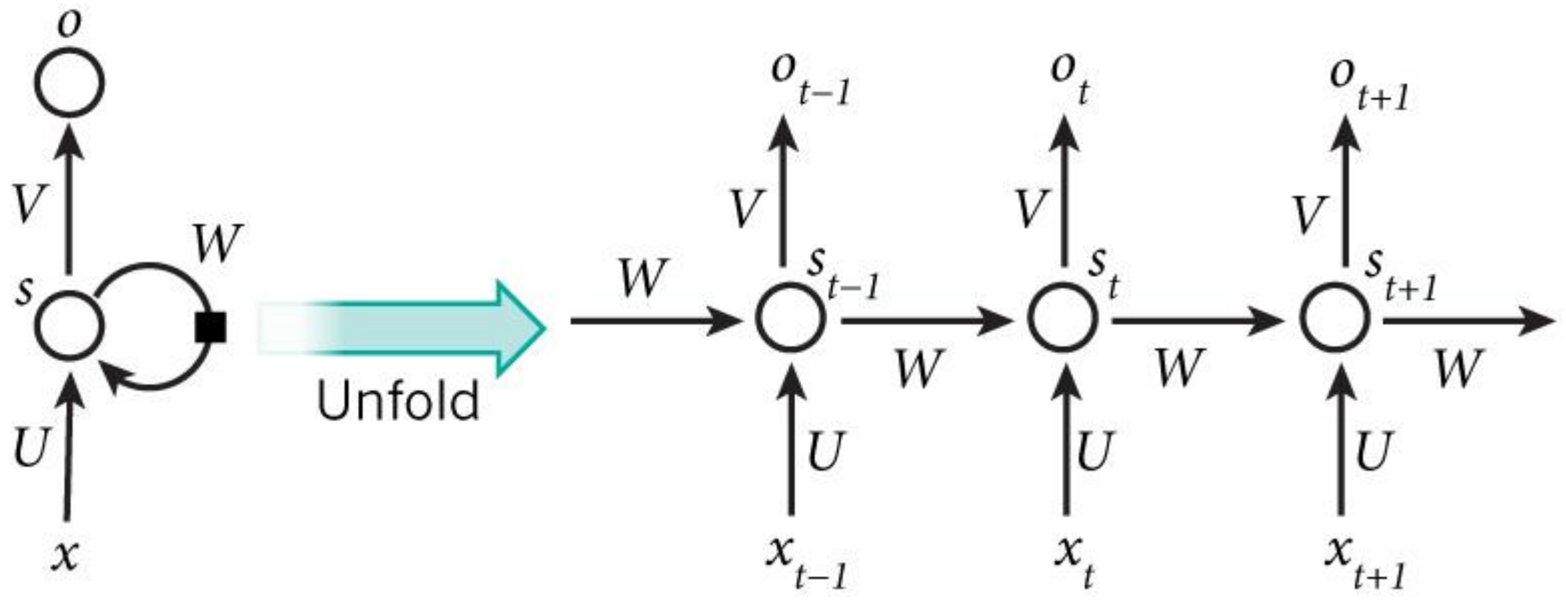
1. http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/

By Denny Britz

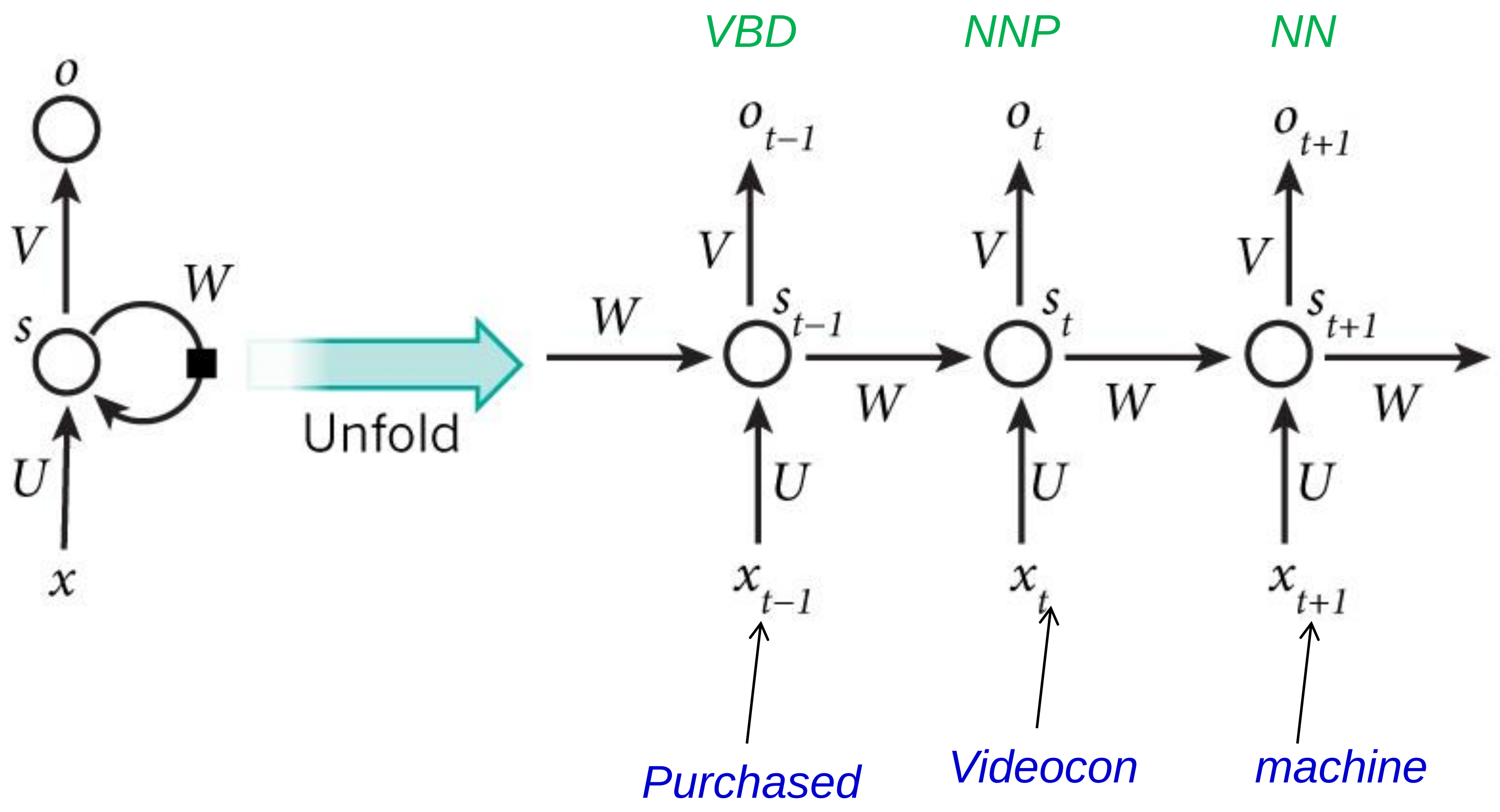
2. Introduction to RNN by Jeffrey Hinton

http://www.cs.toronto.edu/~hinton/csc2535/lectures.html

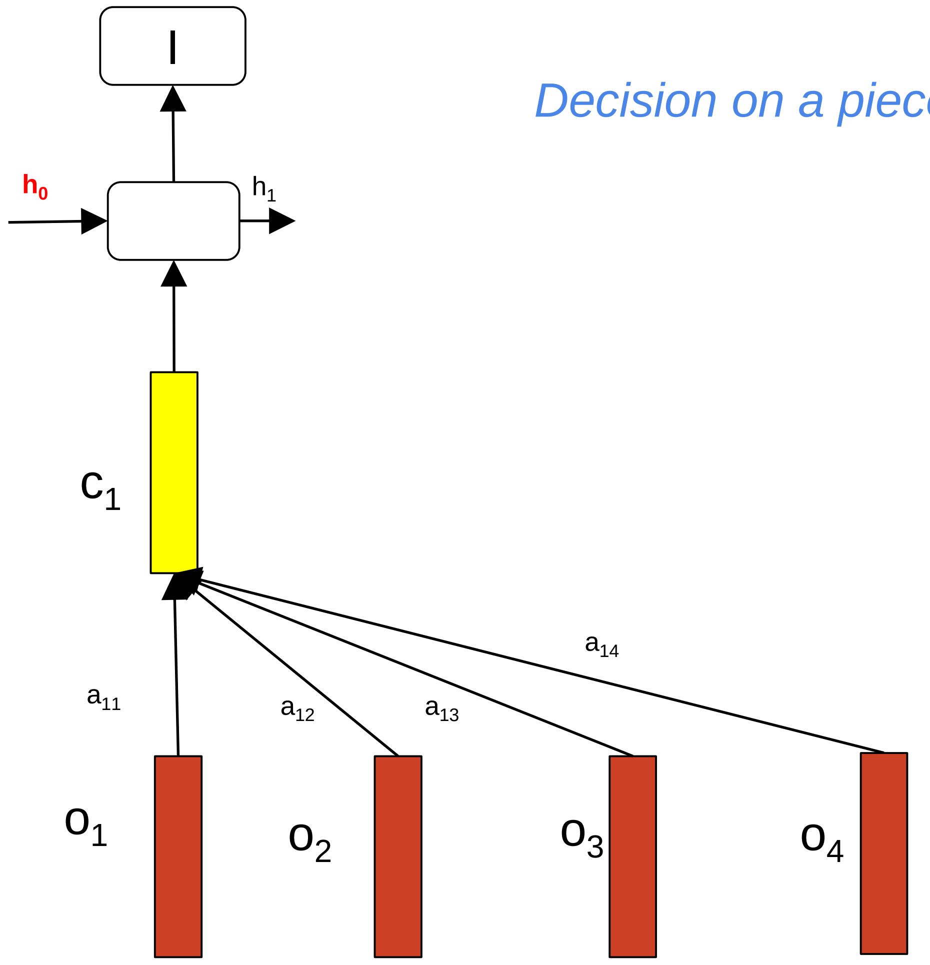
Sequence processing m/c



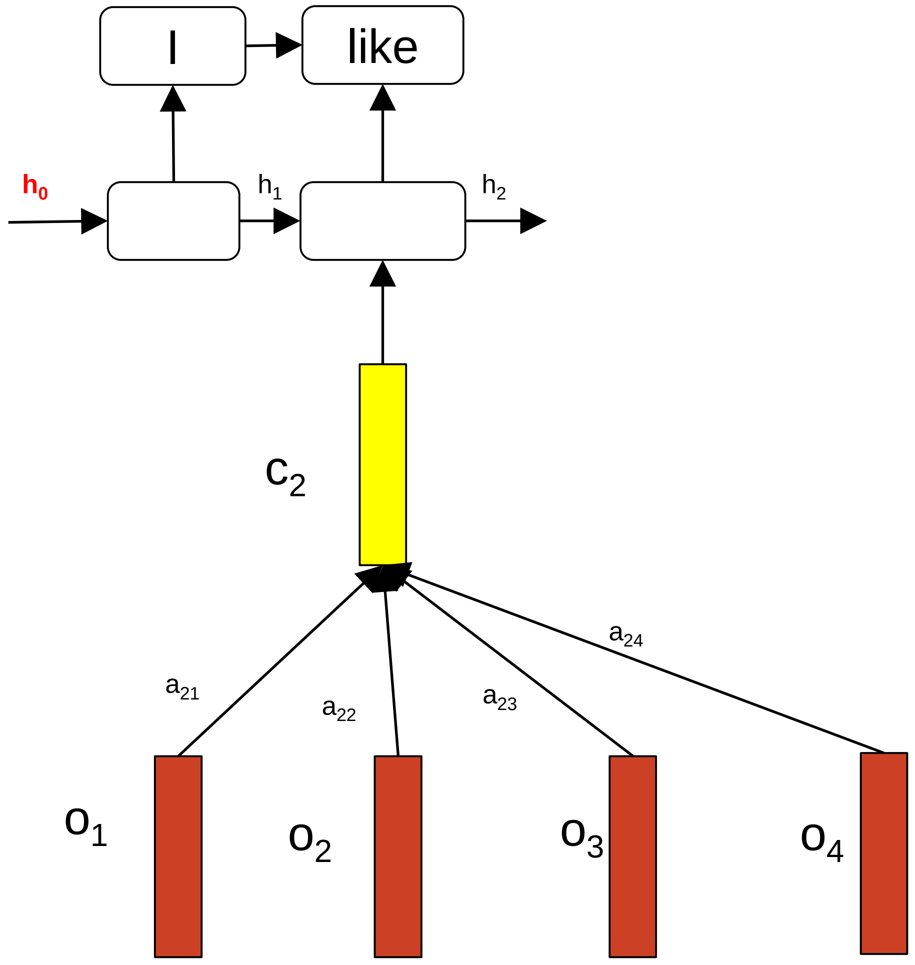
E.g. POS Tagging

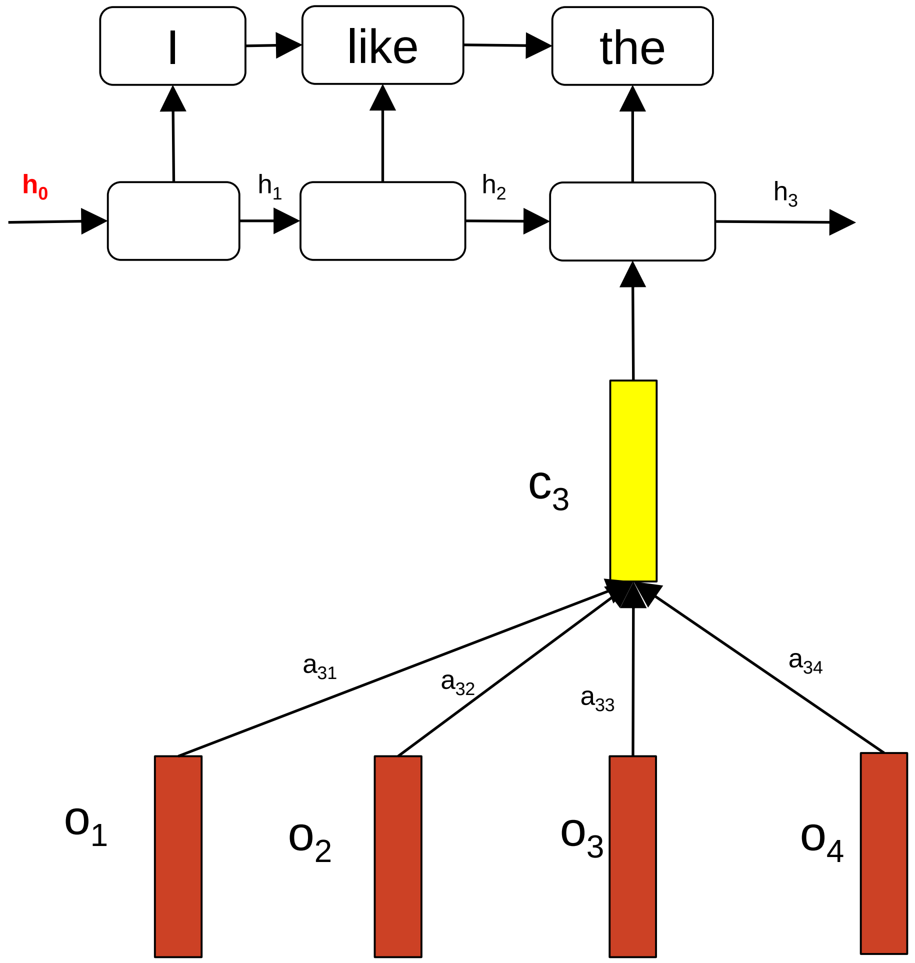


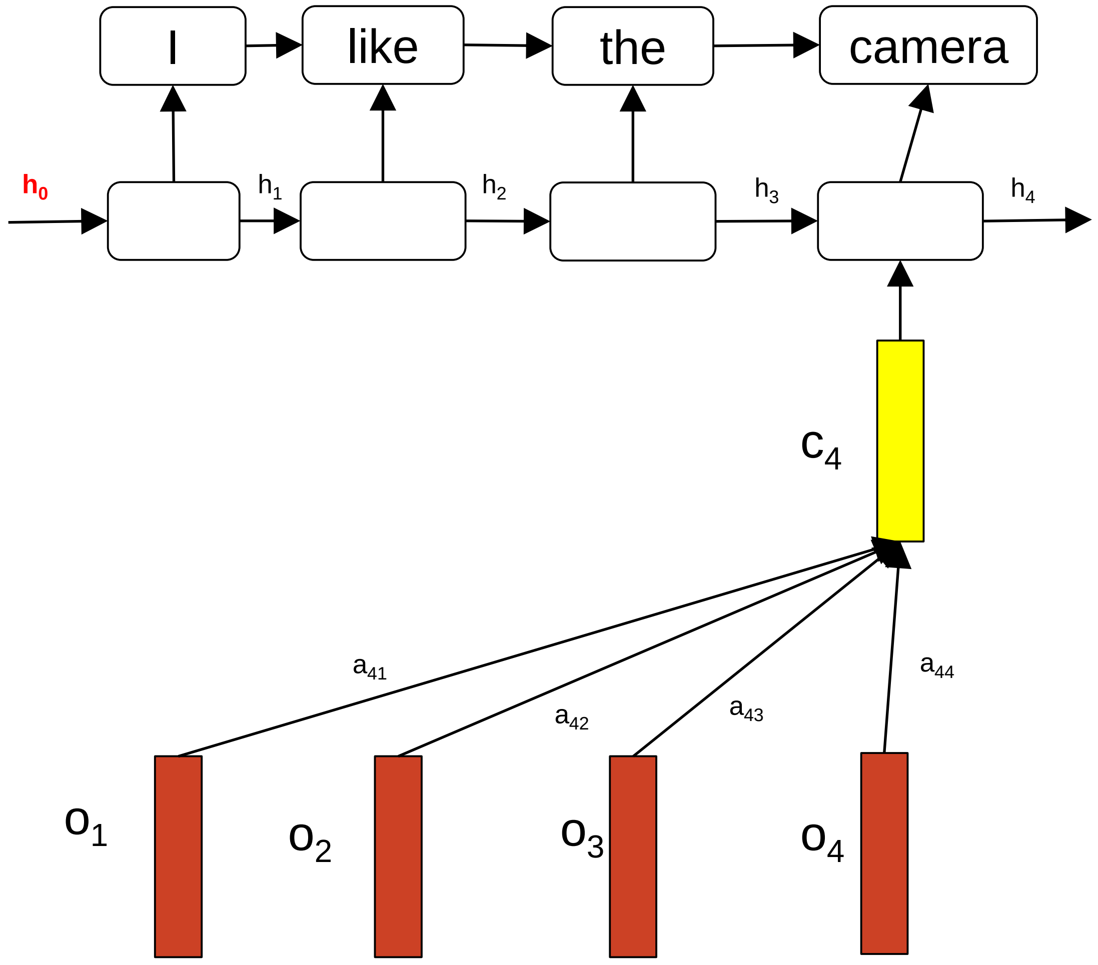
E.g. Sentiment Analysis

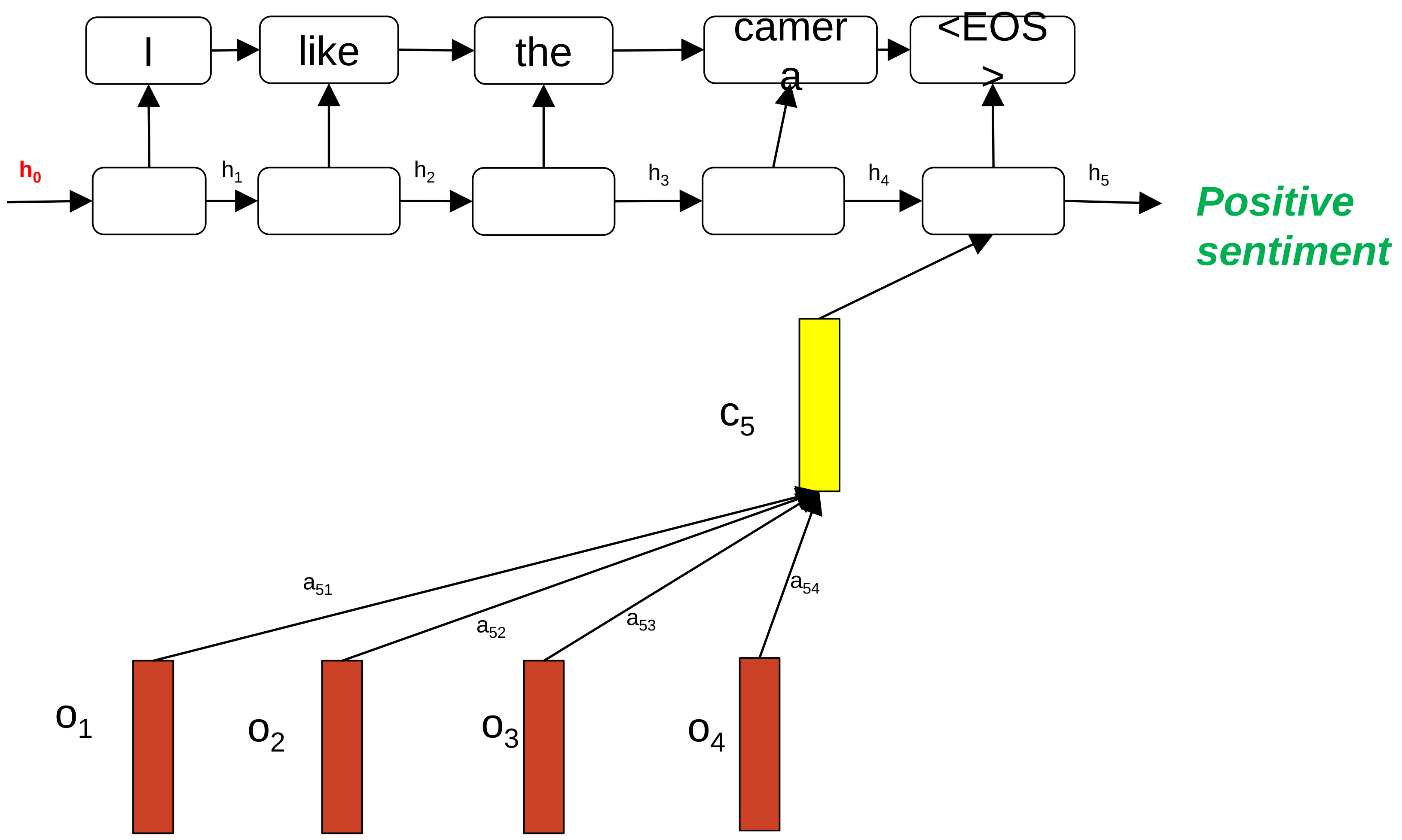


Decision on a piece of text

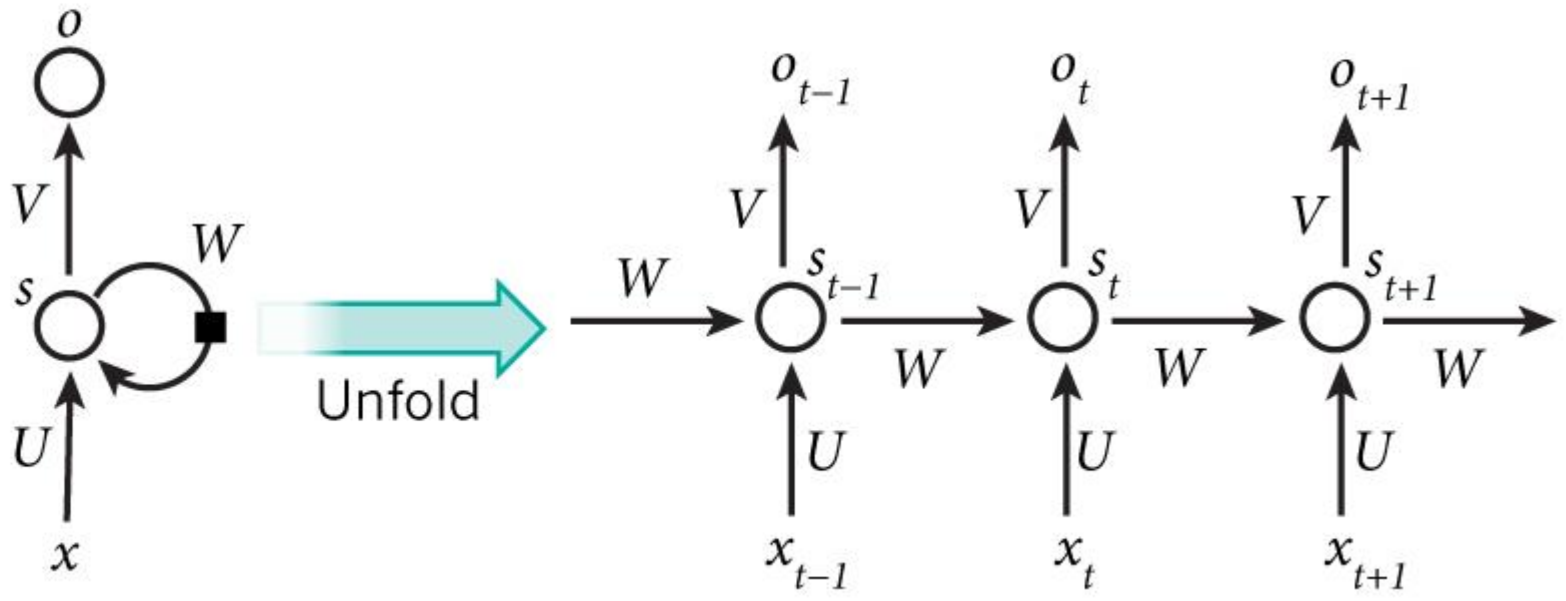

I
like
h_0
h_1
h_2
c_2
a_{21}
a_{22}
a_{23}
a_{24}
o_1
o_2
o_3
o_4


I
like
the
h_0
h_1
h_2
h_3
c_3
a_{31}
a_{32}
a_{33}
a_{34}
o_1
o_2
o_3
o_4


I
like
the
camera
h_0
h_1
h_2
h_3
h_4
c_4
a_{41}
a_{42}
a_{43}
a_{44}
o_1
o_2
o_3
o_4


I
like
the
camer a
<EOS >
h0
h1
h2
h3
h4
h5
Positive sentiment
c5
a51
a52
a53
a54
o1
o2
o3
o4

Back to RNN model



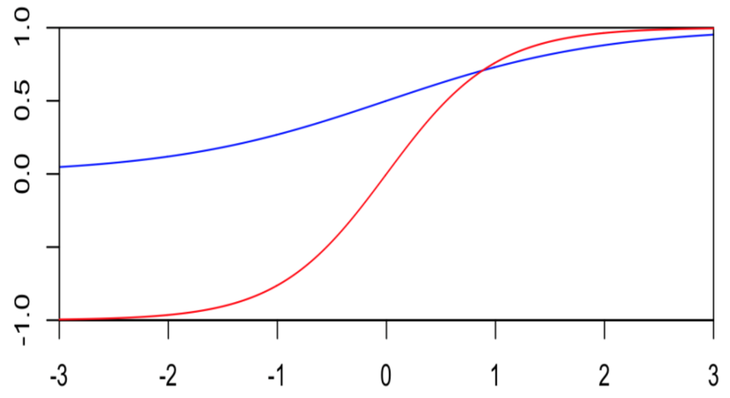
Notation: input and state

- x_t is the input at time step *t*. For example, could be a one-hot vector corresponding to the second word of a sentence.
- s_t is the hidden state at time step *t*. It is the “memory” of the network.
- $s_t = f(U.x_t + Ws_{t-1})$ ***U* and *W* matrices are learnt**
- *f* is a function of the input and the previous state
- Usually *tanh* or *ReLU* (approximated by *softplus*)

Tanh, *ReLU* (rectifier linear unit) and Softplus

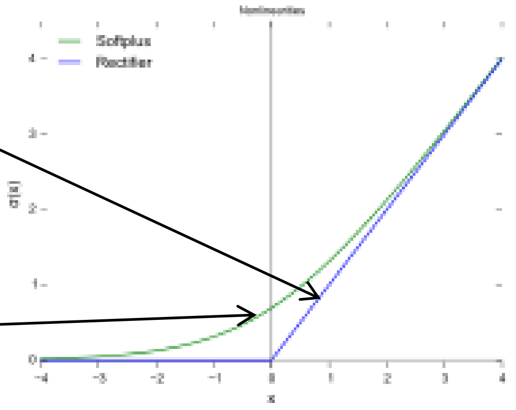
$$\tanh = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

$\tanh =$



$$f(x) = \max(0, x)$$

$$g(x) = \ln(1 + e^{x})$$



Notation: output

- o_t is the output at step t

- For example, if we wanted to predict the next word in a sentence it would be a vector of probabilities across our vocabulary

- $o_t = softmax(V.s_t)$

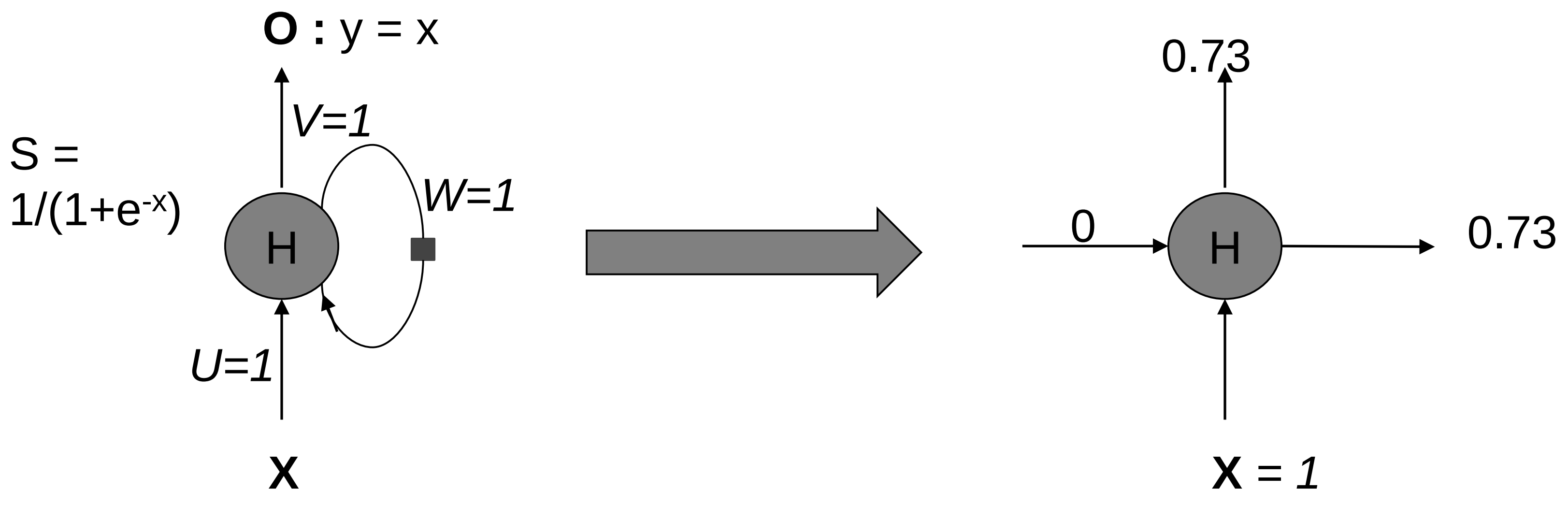
Operation of RNN

- RNN shares the same parameters (*U*, *V*, *W*) across all steps
- Only the input changes
- Sometimes the output at each time step is not needed: e.g., in sentiment analysis
- Main point: the hidden states !!

Illustration of operation

RNN Sequence Processing Example

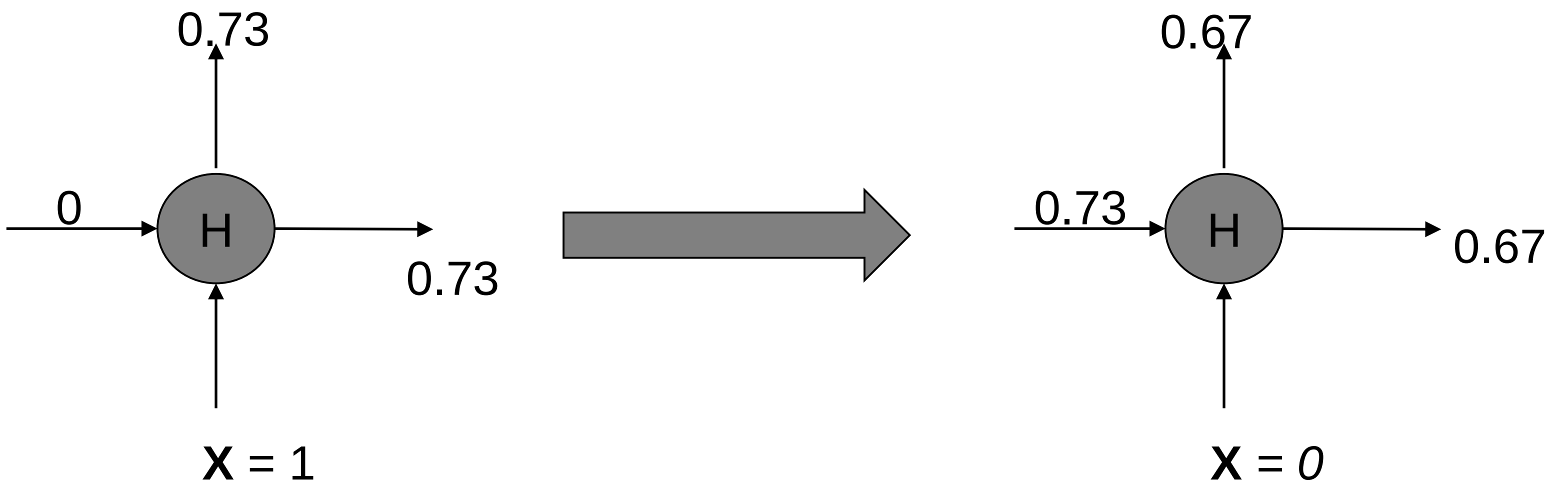
Input Sequence: 1 0 0 0 1 0



T = 1

RNN Sequence Processing Example

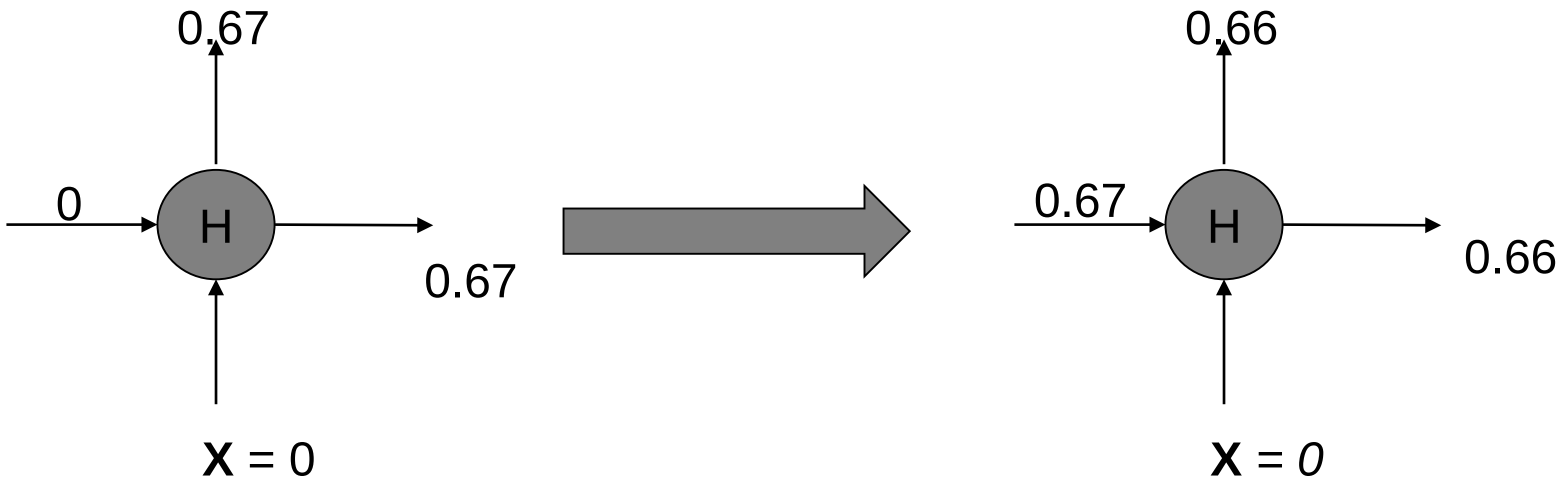
Input Sequence: 1 **0** 0 0 1 0



T = 2

RNN Sequence Processing Example

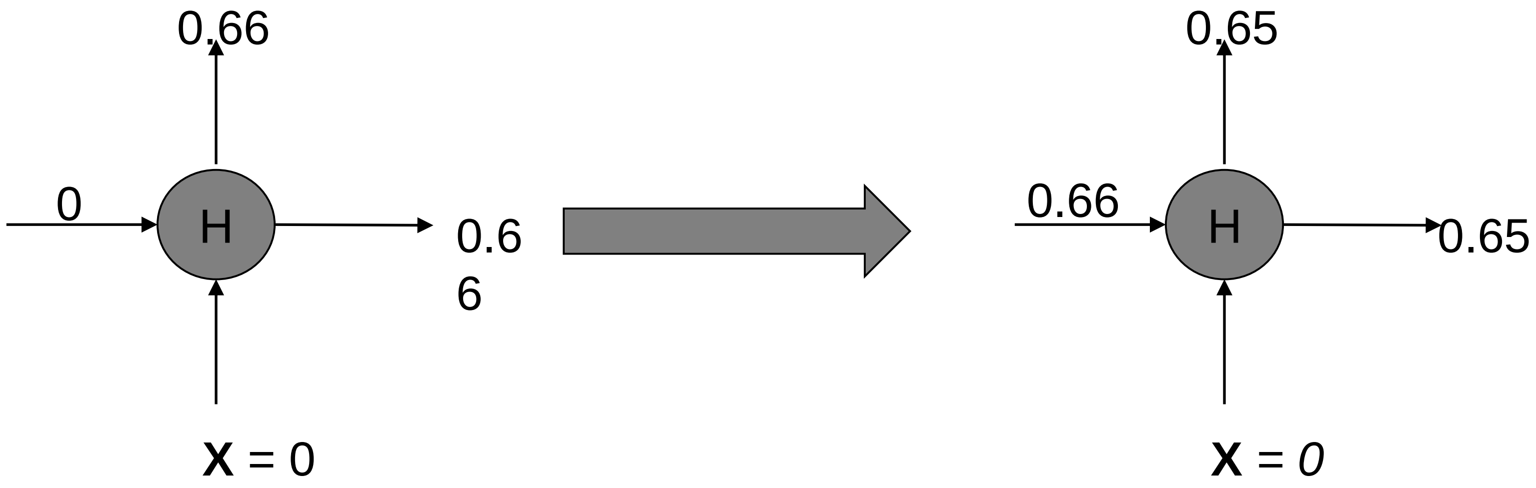
Input Sequence: 1 0 **0** 0 1 0



T = 3

RNN Sequence Processing Example

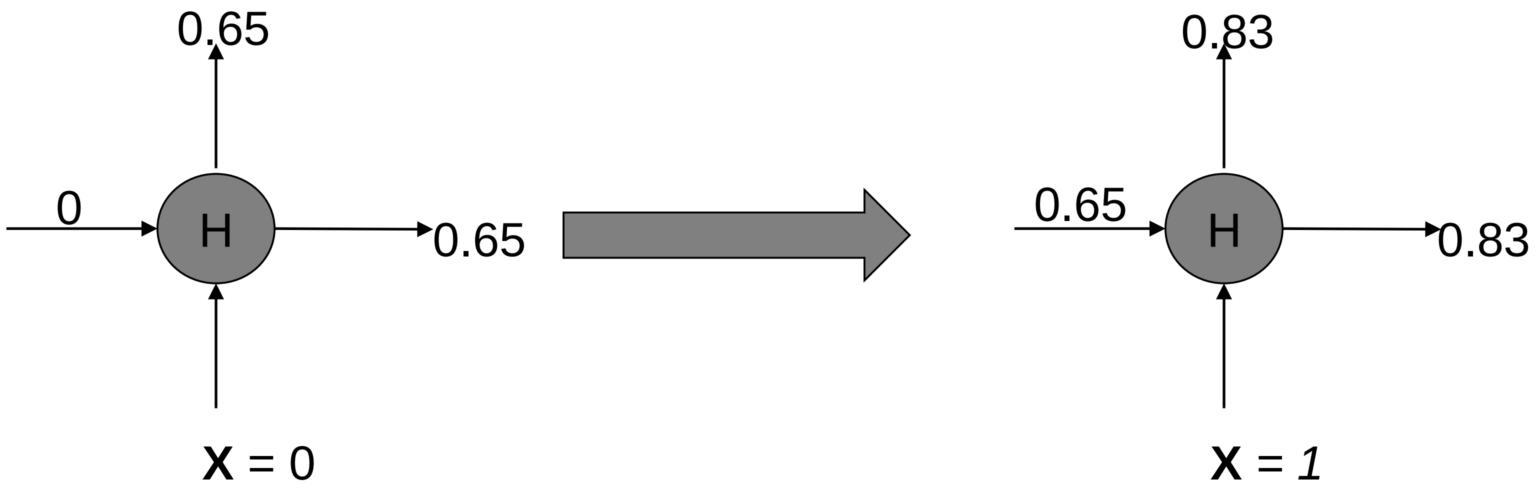
Input Sequence: 1 0 0 **0** 1 0



T = 4

RNN Sequence Processing Example

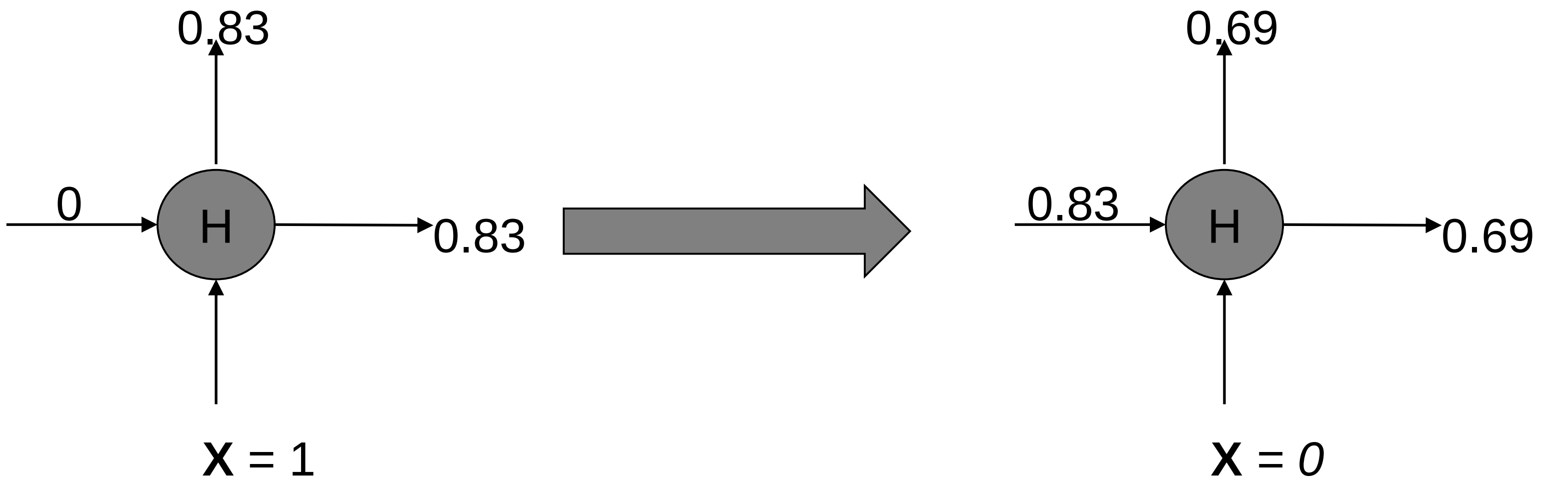
Input Sequence: 1 0 0 0 **1** 0



T = 5

RNN Sequence Processing Example

Input Sequence: 1 0 0 0 1 **0**

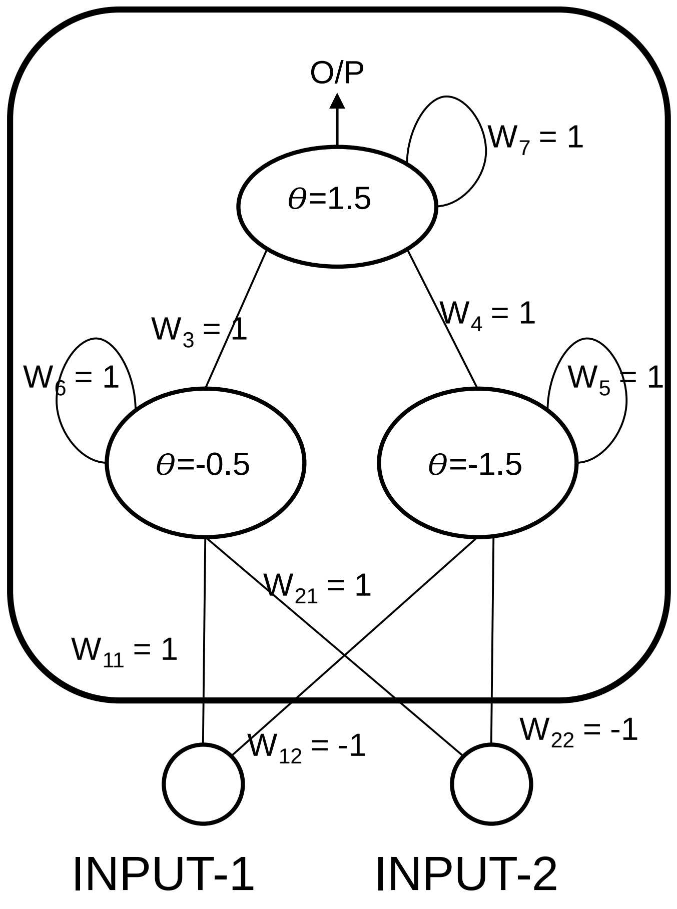


T = 6

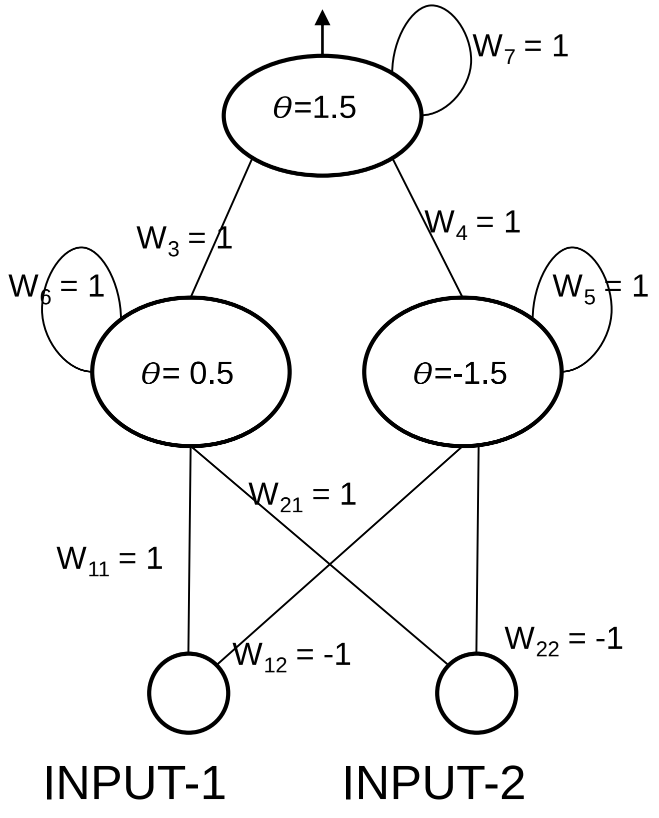
Final o/p seq: 0.73 0.67 0.66 0.65 0.83 0.69

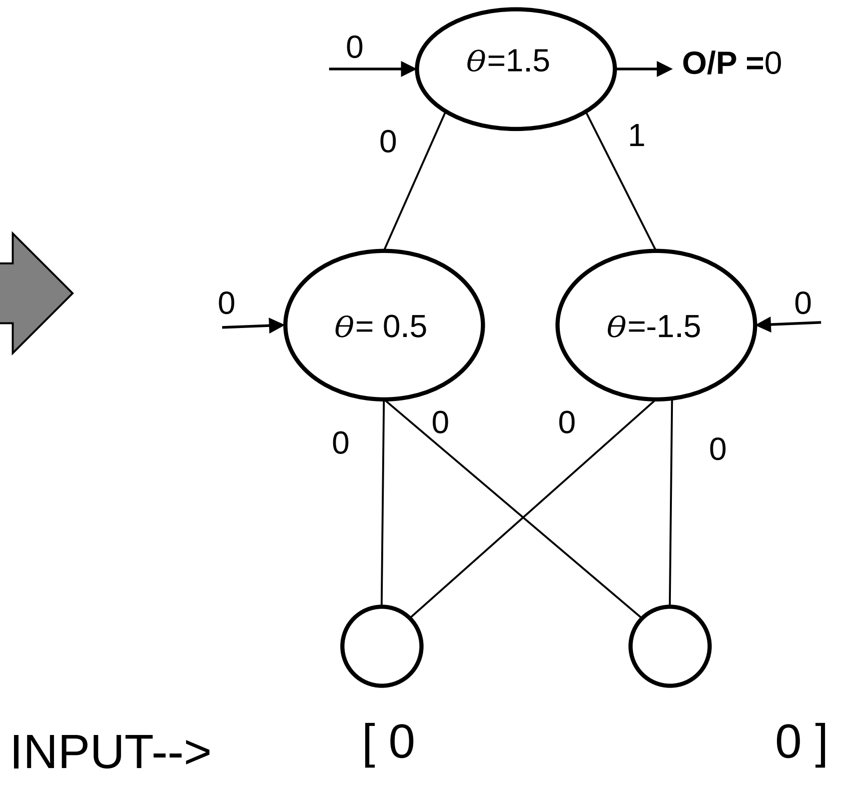
XOR N/W acting on a bit-string: 2 bits at a time

XOR RNN unit

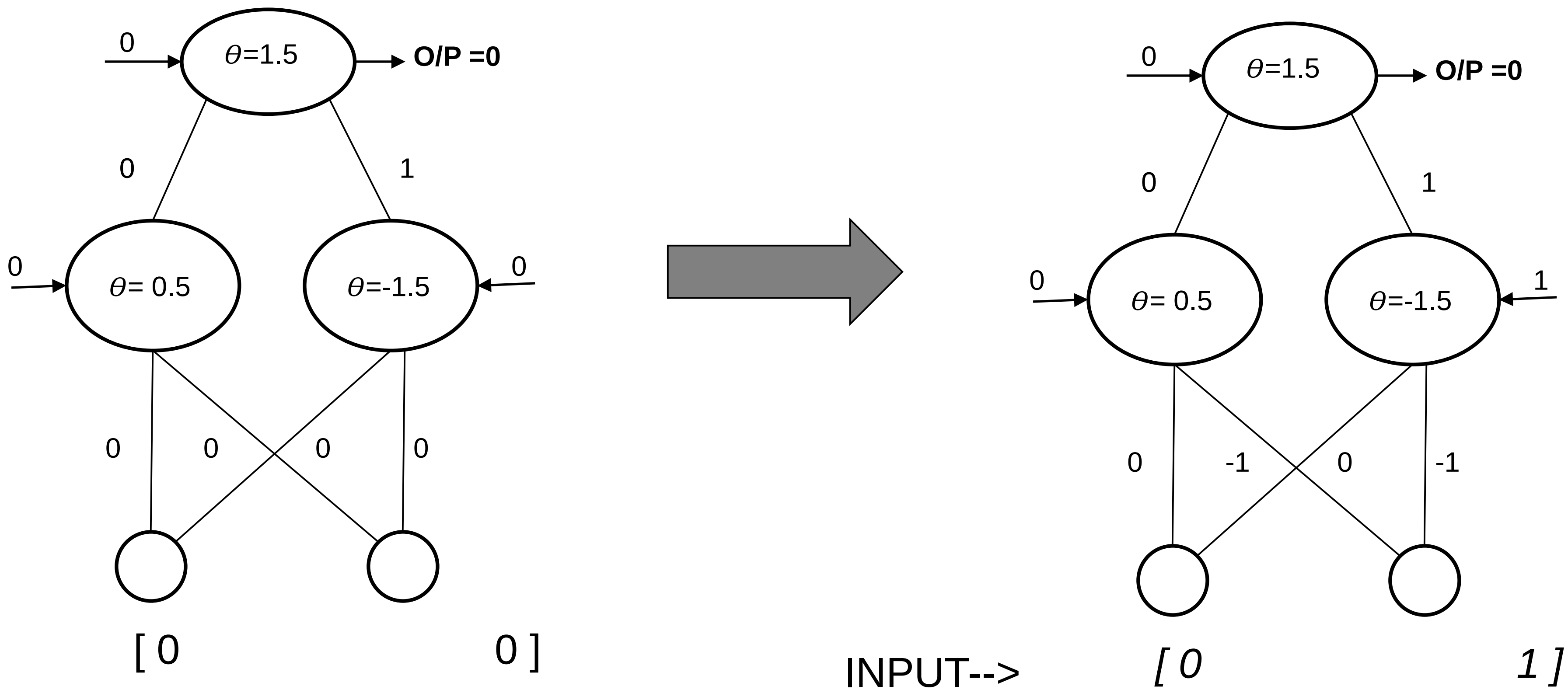

O/P
$W_7 = 1$
θ=1.5
$W_3 = 1$
$W_4 = 1$
$W_6 = 1$
$W_5 = 1$
θ=-0.5
θ=-1.5
$W_{21} = 1$
$W_{11} = 1$
$W_{12} = -1$
$W_{22} = -1$
INPUT-1
INPUT-2

XOR RNN unit (all feedback wt= 1); values adjacent to connections are o/p coming from the source neurons

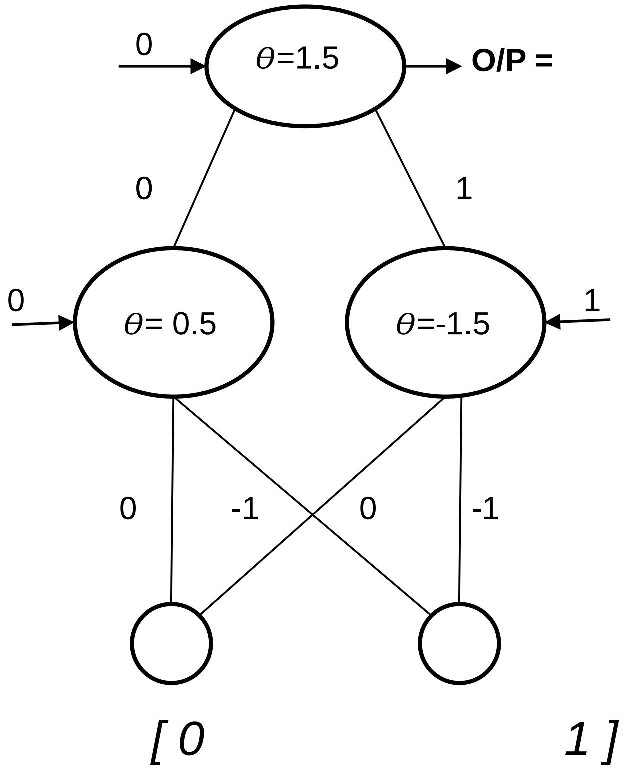




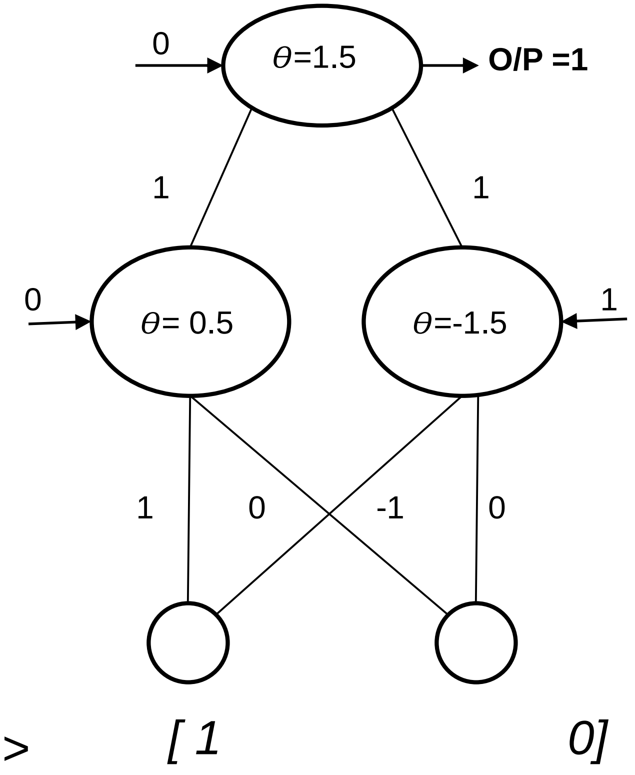
XOR RNN unit; (all feedback wt= 1); values adjacent to connections are o/p coming from the source neurons



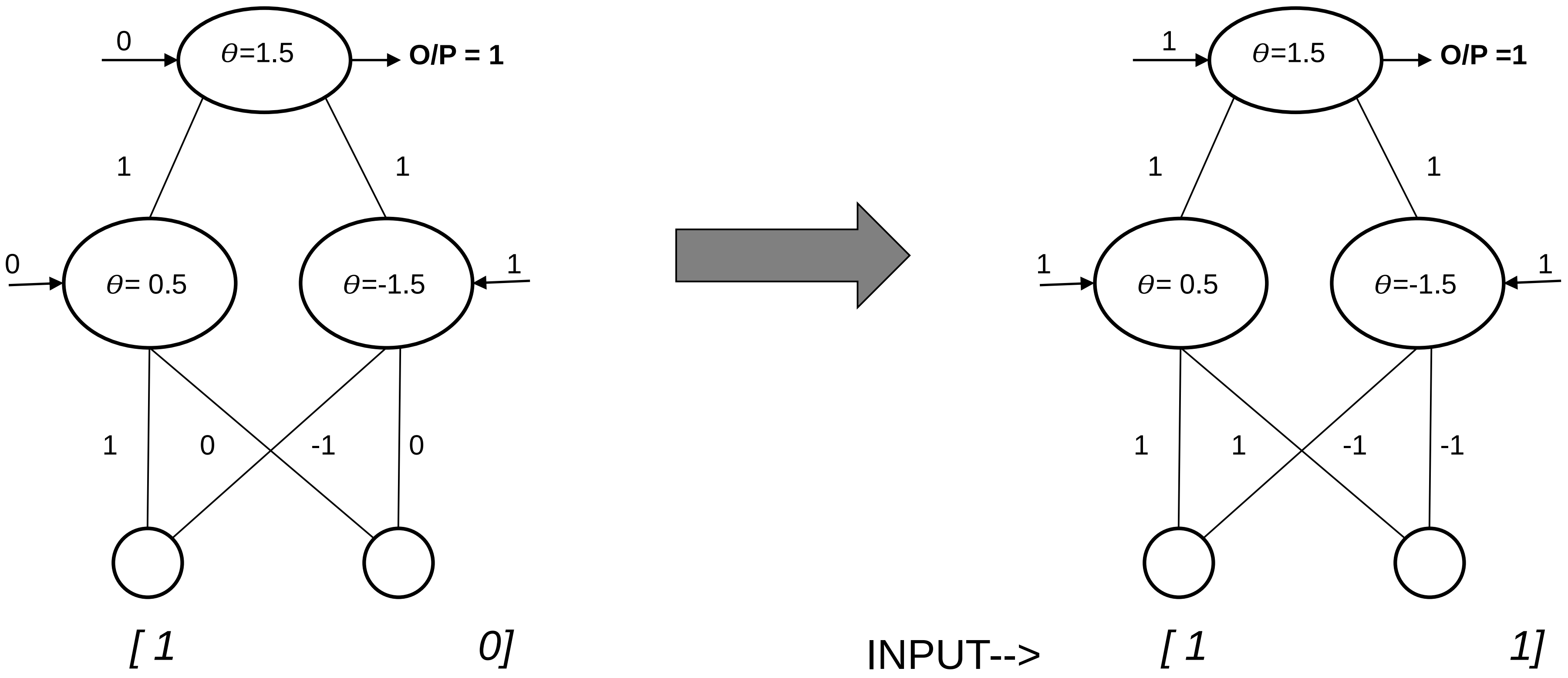
XOR RNN unit; (all feedback wt= 1); values adjacent to connections are o/p coming from the source neurons



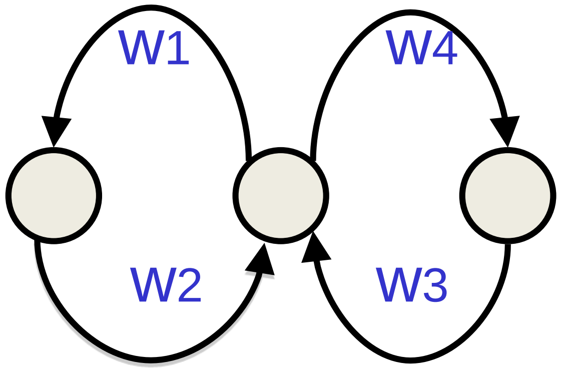
INPUT-->



XOR RNN unit; (all feedback wt= 1); values adjacent to connections are o/p coming from the source neurons

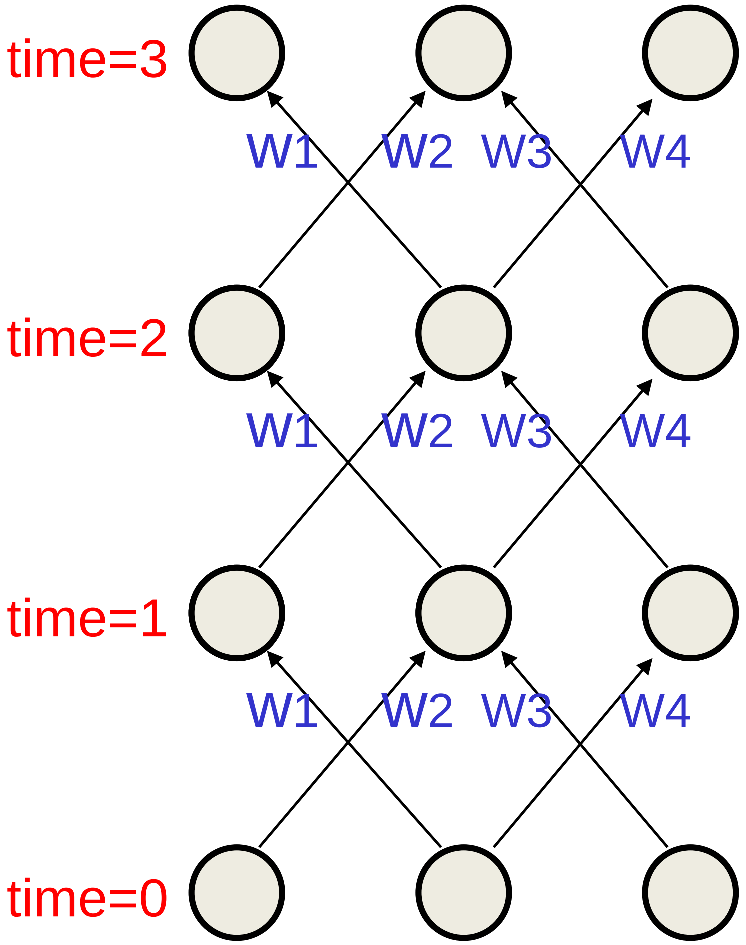


The equivalence between feedforward nets and recurrent nets



Assume that there is a time delay of 1 in using each connection.

The recurrent net is just a layered net that keeps reusing the same weights.



BPTT- BP through time- Backpropagation with weight constraints

- **Linear constraints between the weights.**
- **Compute the gradients as usual**
- **Then modify the gradients so that they satisfy the constraints.**
- **So if the weights started off satisfying the constraints, they will continue to satisfy them.**

Example

$$To\ \ constrain: \quad w_1 = w_2$$

$$we\ \ need: \quad \Delta w_1 = \Delta w_2$$

$$compute: \quad \frac{\partial E}{\partial w_1} \quad and \quad \frac{\partial E}{\partial w_2}$$

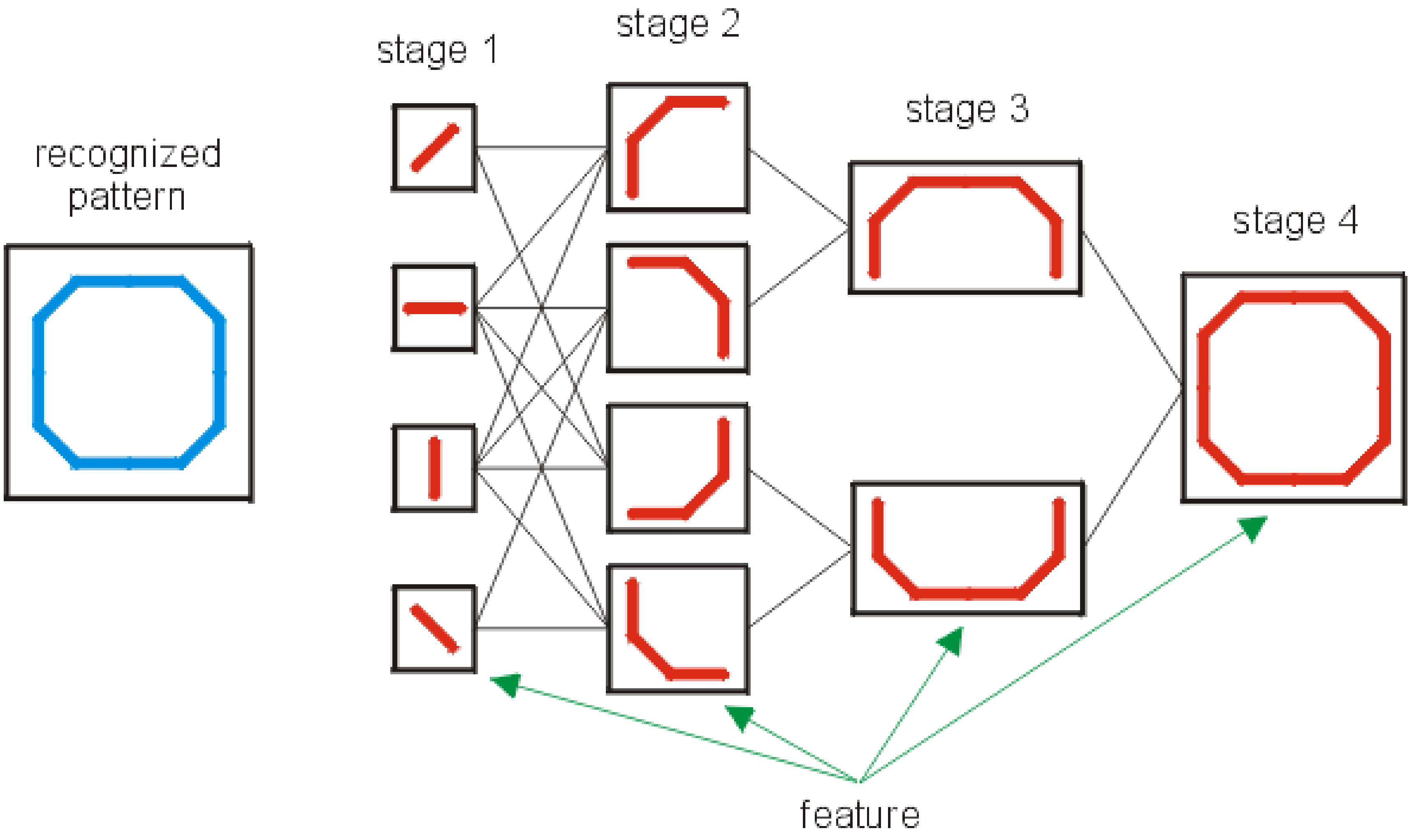
$$use \quad \frac{\partial E}{\partial w_1} + \frac{\partial E}{\partial w_2} \quad for\ w_1\ and\ w_2$$

Convolutional Neural Network (CNN)

CNN= feedforward + recurrent!

- Whatever we learnt so far in FF-BP is useful to understand CNN
- So also is the case with RNN (and LSTM)
- Input divided into regions and fed forward
- Window slides over the input: input changes, but ‘filter’ parameters remain same
- That is RNN

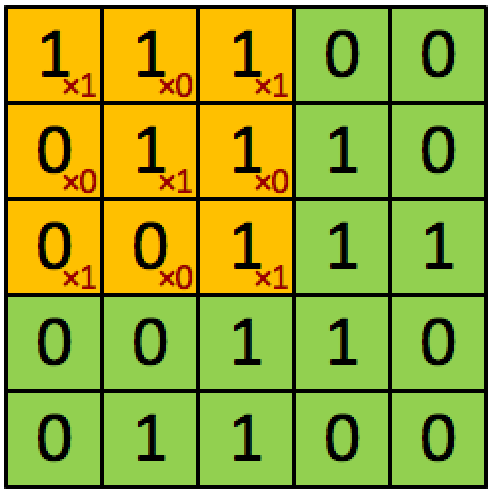
Genesis: Neocognitron (Fukusima, 1980)



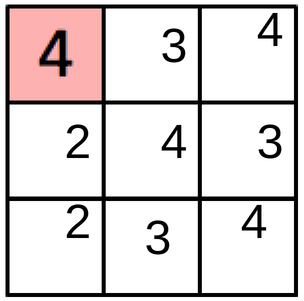
Convolution

1	**0**	**1**
0	1	0
1	0	1

kernel



Image



Convolved Feature

- Matrix on the left represents an black and white image.
- Each entry corresponds to one pixel, 0 for black and 1 for white (typically it's between 0 and 255 for grayscale images).
- The sliding window is called a *kernel, filter,* or *feature detector.*
- Here we use a 3×3 filter, multiply its values element-wise with the original matrix, then sum them up.
- To get the full convolution we do this for each element by sliding the filter over the whole matrix.

CNN architecture

- Several layers of convolution with *tanh* or *ReLU* applied to the results
- In a traditional feedforward neural network we connect each input neuron to each output neuron in the next layer. That's also called a fully connected layer, or affine layer.
- In CNNs we use convolutions over the input layer to compute the output.
- This results in local connections, where each region of the input is connected to a neuron in the output

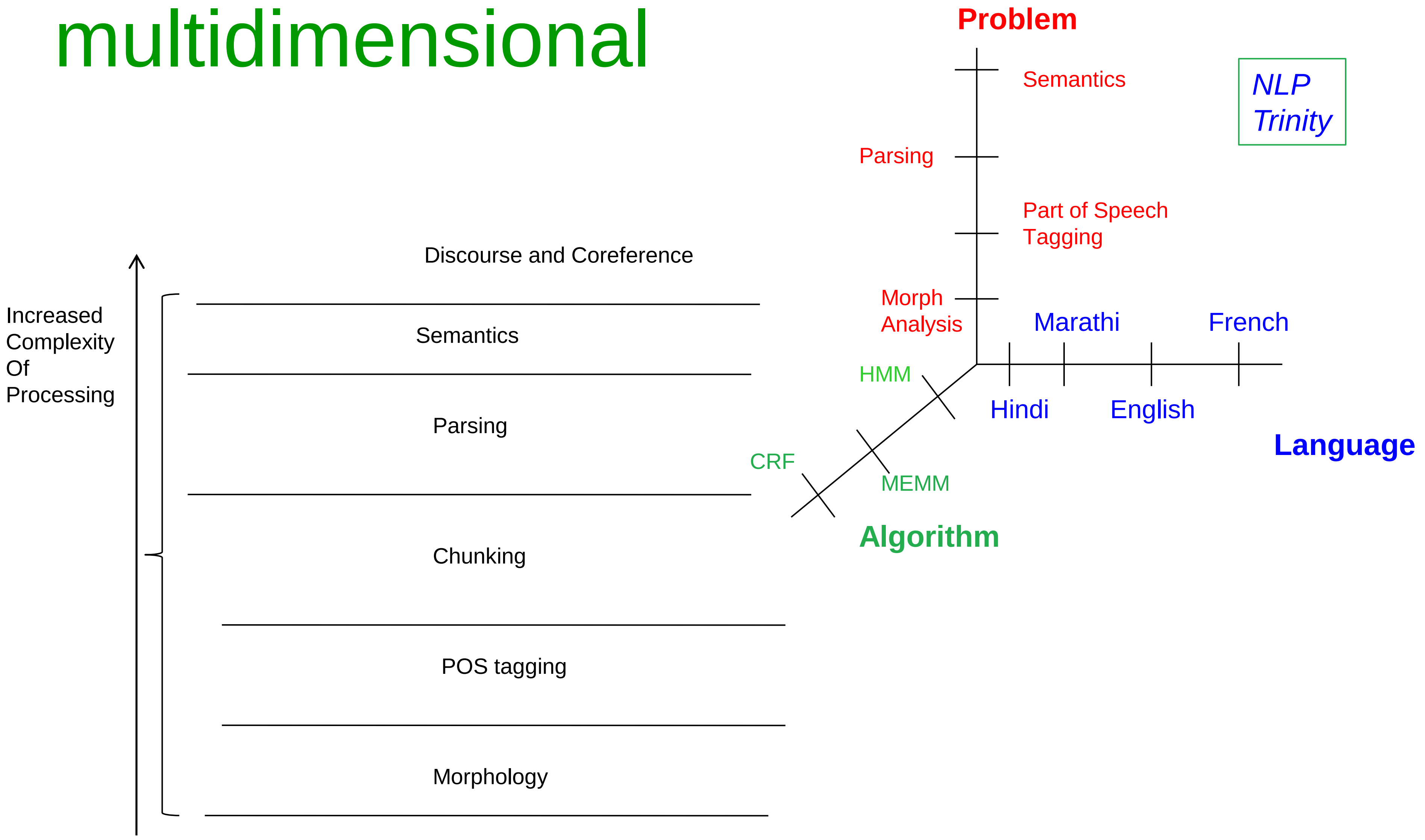
Learning in CNN

- **Automatically learns the values of its filters**
- For example, in Image Classification learn to
  - detect edges from raw pixels in the first layer,
  - then use the edges to detect simple shapes in the second layer,
  - and then use these shapes to deter higher-level features, such as facial shapes in higher layers.
  - The last layer is then a classifier that uses these high-level features.

What about NLP and CNN?

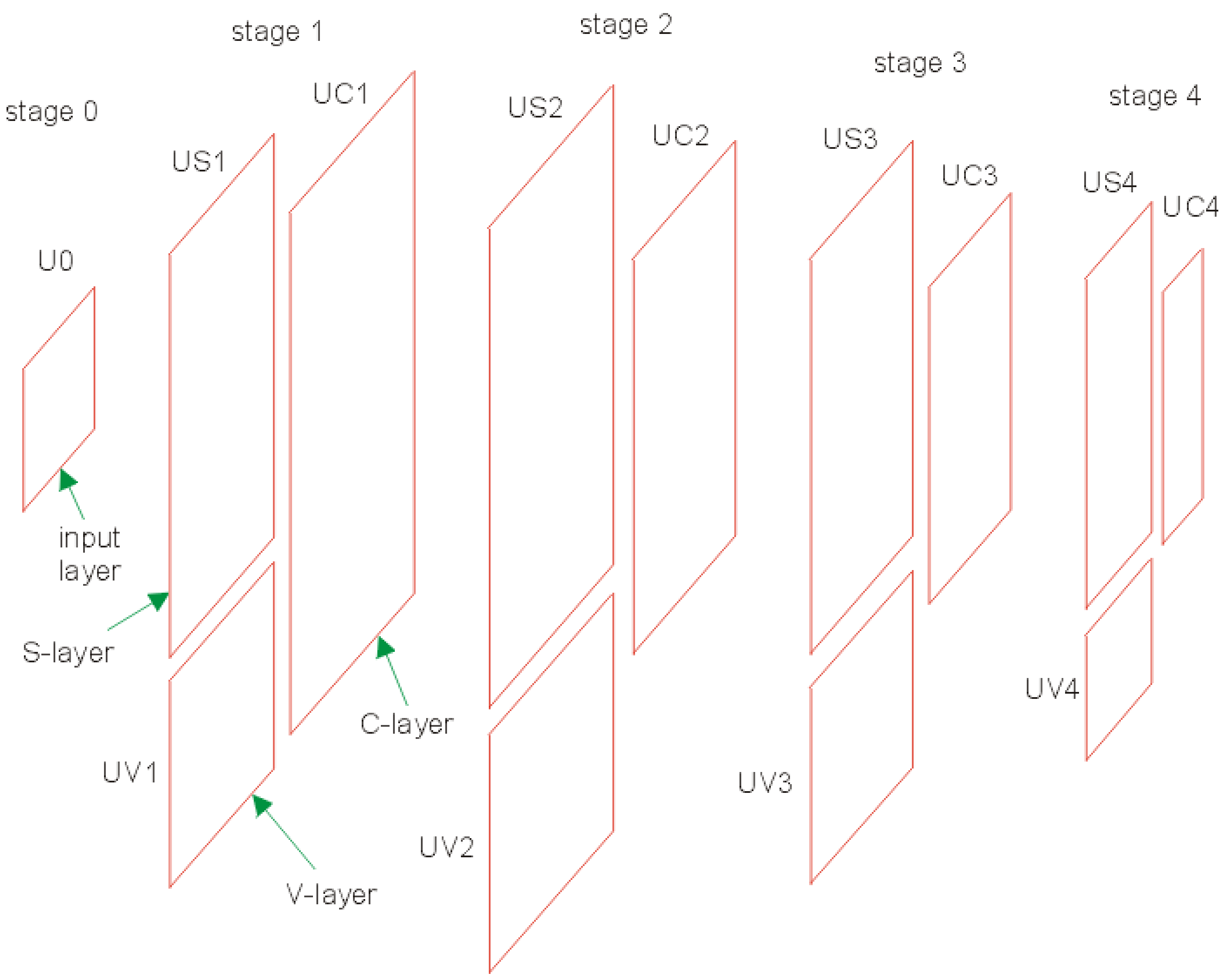
- Natural Match!
- NLP happens in layers

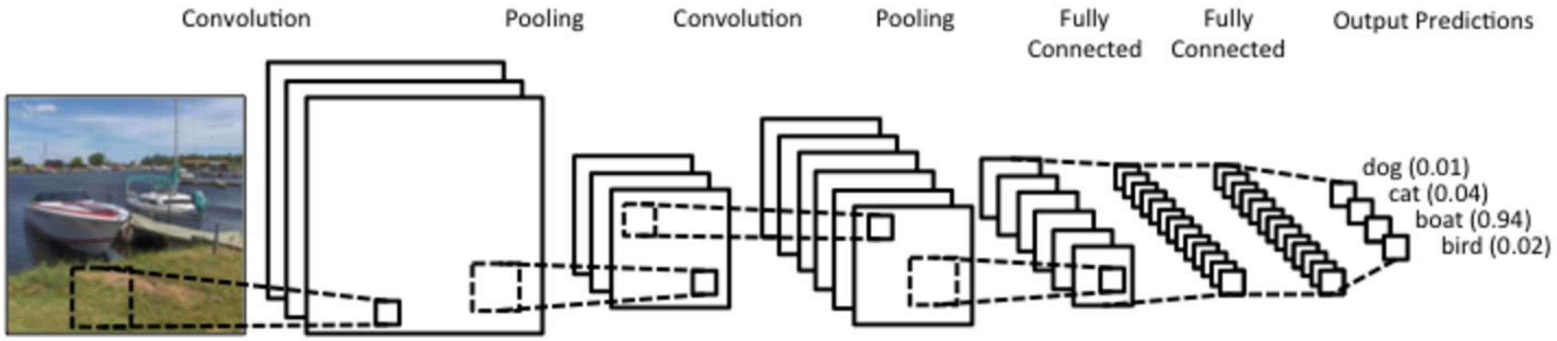
NLP: multilayered, multidimensional



NLP layers and CNN

- Morph layer →
- POS layer →
- Parse layer →
- Semantics layer


stage 0
stage 1
stage 2
stage 3
stage 4
U0
US1
UC1
US2
UC2
US3
UC3
US4
UC4
UV1
UV2
UV3
UV4
input layer
S-layer
C-layer
V-layer



http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/

Pooling

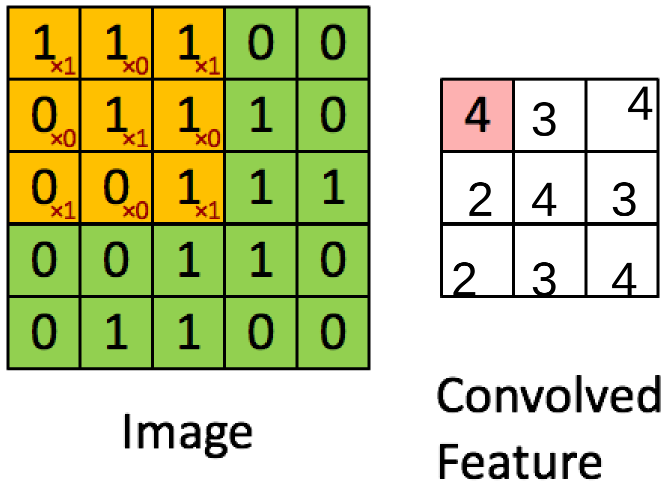
- Gives invariance in translation, rotation and scaling
- Important for image recognition
- Role in NLP?

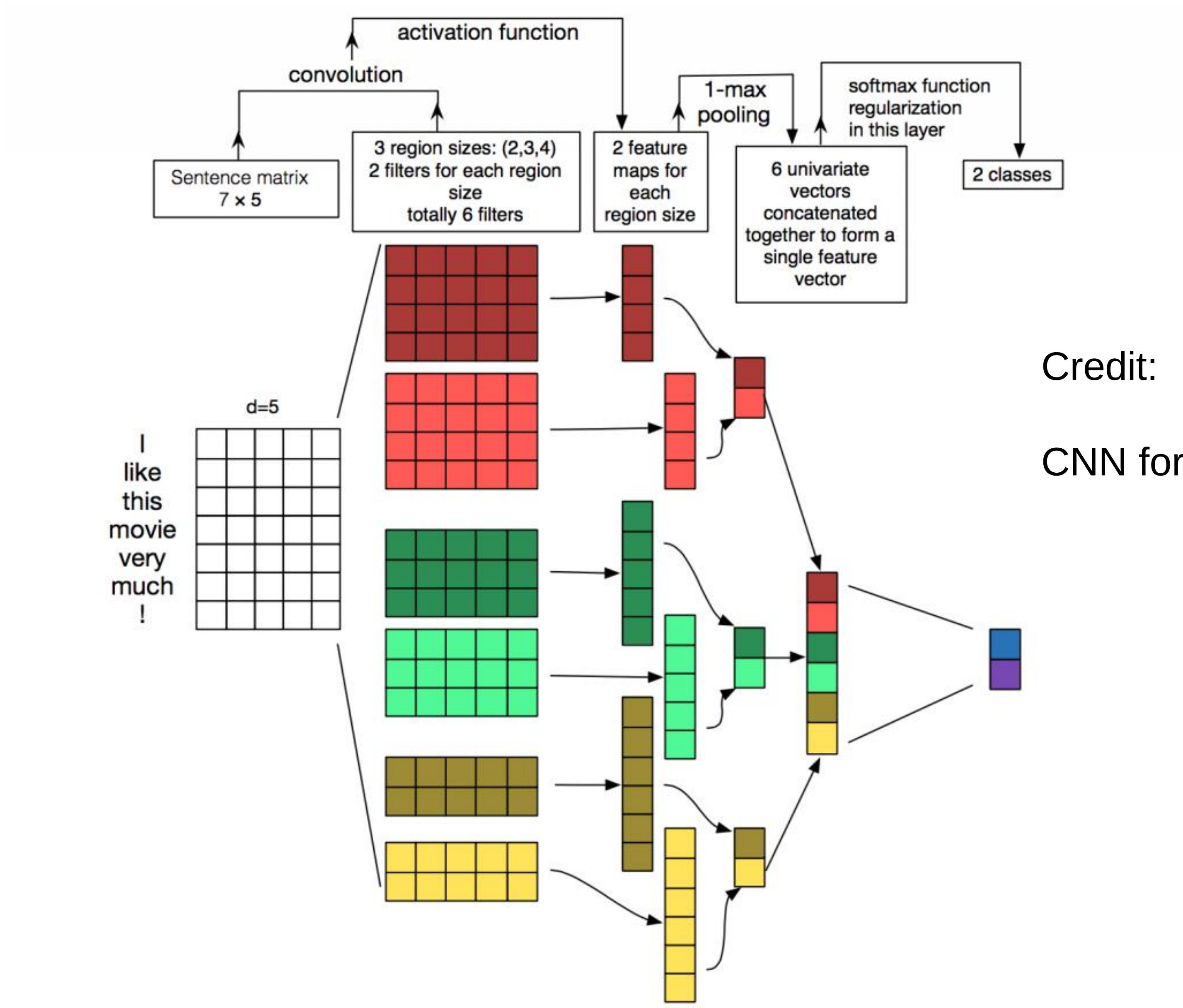
Input matrix for CNN: NLP

▪“image” for NLP ←→ word vectors
▪in the rows

▪For a 10 word sentence using a 100-dimensional Embedding,

▪we would have a 10×100 matrix as our input




activation function
convolution
Sentence matrix 7 × 5
3 region sizes: (2,3,4) 2 filters for each region size totally 6 filters
2 feature maps for each region size
1-max pooling
6 univariate vectors concatenated together to form a single feature vector
softmax function regularization in this layer
2 classes
d=5
I
like
this
movie
very
much
!

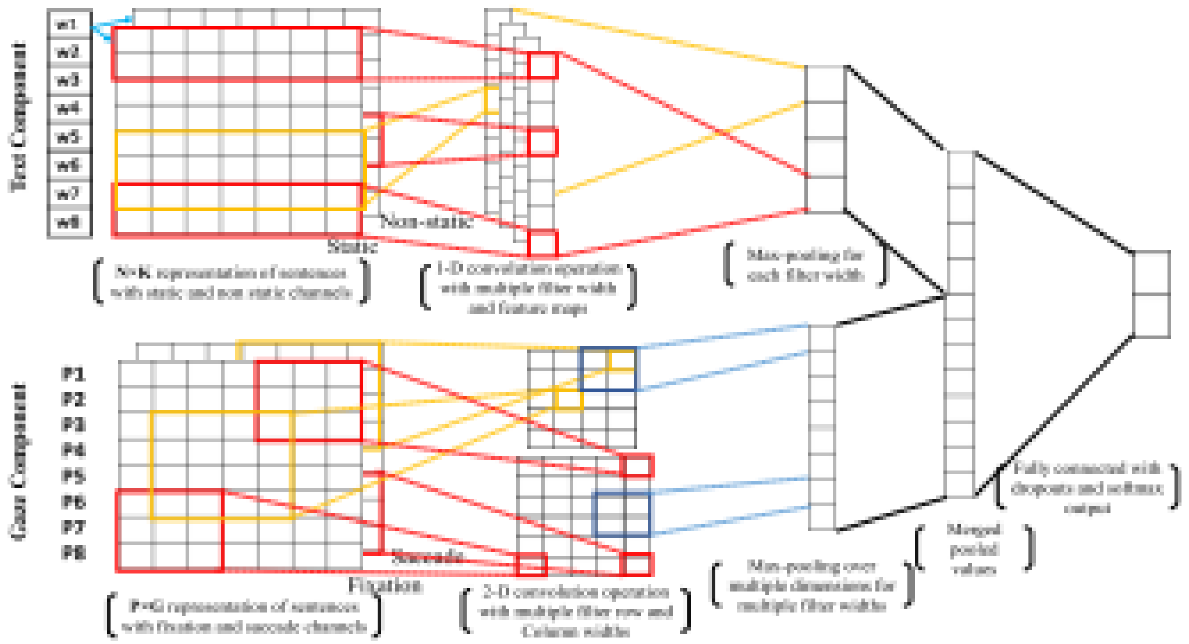
Credit: Denny Britz

CNN for NLP

CNN Hyper parameters

- Narrow width vs. wide width
- Stride size
- Pooling layers
- Channels

Abhijit Mishra, Kuntal Dey and Pushpak Bhattacharyya, *Learning Cognitive Features from Gaze Data for Sentiment and Sarcasm Classification Using Convolutional Neural Network*, **ACL 2017**, Vancouver, Canada, July 30-August 4, 2017.


Text Component
w1
w2
w3
w4
w5
w6
w7
w8
Non-static
Static
N×K representation of sentences with static and non static channels
1-D convolution operation with multiple filter width and feature maps
Max-pooling for each filter width
Gaze Component
P1
P2
P3
P4
P5
P6
P7
P8
Saccade
Fixation
P×G representation of sentences with fixation and saccade channels
2-D convolution operation with multiple filter row and column widths
Max-pooling over multiple dimensions for multiple filter widths
Merged pooled values
Fully connected with dropouts and softmax output

Learning Cognitive Features from Gaze Data for Sentiment and Sarcasm Classification

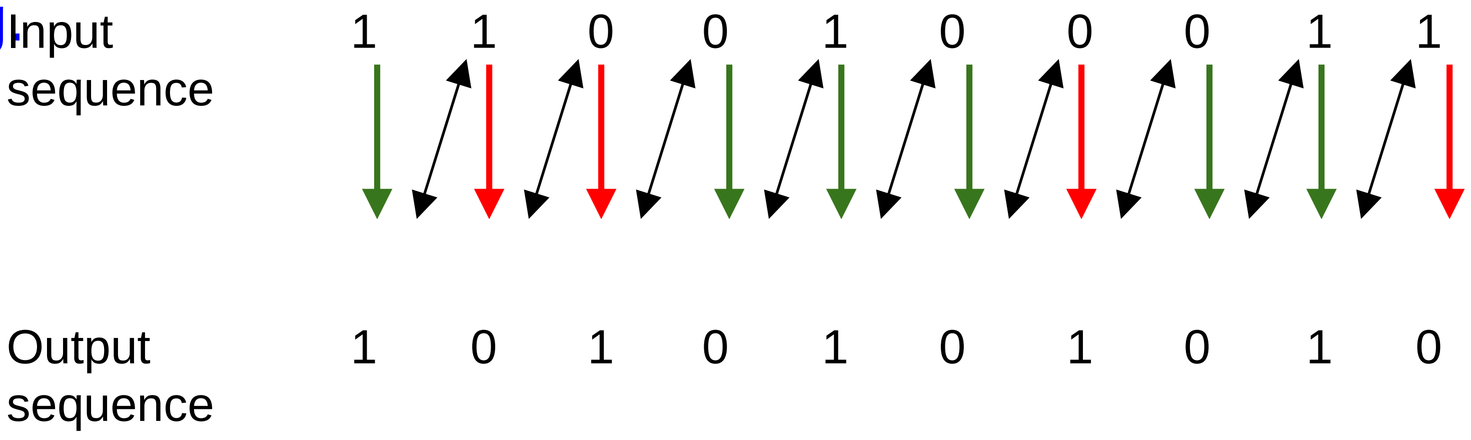
- In complex classification tasks like sentiment analysis and sarcasm detection, even the extraction and choice of features should be delegated to the learning system
- CNN learns features from both gaze and text and uses them to classify the input text

Backup Slides

Problem 1: Bit Reverse

Bit Reverse

- Problem definition:
  - Reverse the bit if the current i/p and previous o/p are same.
- E.g.



Prepare Data

Let

Sequence length

: 10

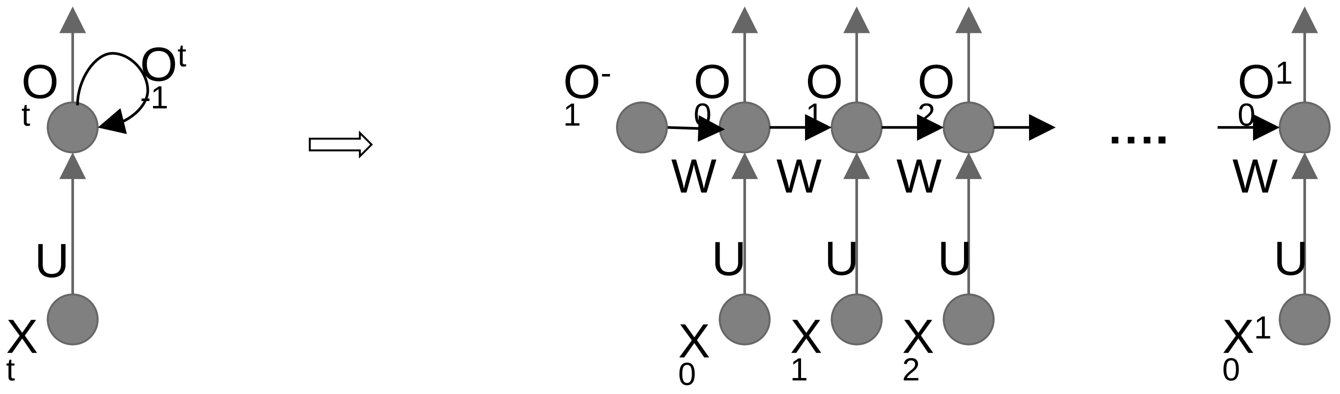
Dimension of each element of I/p sequence (X) : 1 bit

Dimension of each element of O/p sequence (O) :

1 bit

Network Architecture

Number of I/P neurons	: 1
Number of O/P neurons	: 1
Sequence length	: 10



Implementation using Keras 1/8

1. Import necessary libraries

```
     import numpy
     # Numpy for mathematical ops
    import keras
     # Keras main library
     from keras.models import Sequential                    # Model type
    from keras.layers import SimpleRNN                      # Recurrent layer


dimInUnits = numInNeurons = 1
     dimOutUnits = numOutNeurons = 1
     numUnits = seqLen = 10
    numInstances = 4
```

Implementation using Keras 2/8

2. Design network

```
    model = Sequential()                                    # Instantiate
sequential network

# Add a single layer of RNN layer.
# input_shape is required only for the first layer of the network.
# return_sequences should be True, if we require o/p at each time step. It will be
False, if we require single o/p for the entire sequence.
model.add(SimpleRNN(numOutNeurons, input_shape=(seqLen, numInNeurons),
return_sequences=True, activation='sigmoid'))

# If we need to add more layers we have to call model.add() again. Next time
input_shape() is not required.
```

Implementation using Keras 3/8

3. Compile the network

```
model.compile(optimizer='sgd', loss='mse')

# Validate the network. If any issues (dimension mismatch etc.) are found, they
will be reported.
# Optimization algorithm is stochastic gradient descent
# Loss is mean squared error


# At this point network is ready for training
```

Implementation using Keras 4/8

4. Print the the network summary

```
model.summary()            # Print summary of the network
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
simple_rnn_1 (SimpleRNN)     (None, 10, 1)             3
=================================================================
Total params: 3.0
Trainable params: 3
Non-trainable params: 0.0
```

I/p to layer[0] weight : 1
Layer[0] (t-1) to layer[0] (t) weight : 1
I/p bias weight : 1

Implementation using Keras 5/8

5. Load training data

```
X = np.loadtxt(open('x.txt','r'))        # load sequence i/p file
O = np.loadtxt(open('o.txt','r'))        # load sequence o/p file
```

6. Reshape data *w.r.t.* the network

```
X = X.reshape(numInstances, numUnits, dimInUnits)
# Input file has 'numInstances', each instance has 'numUnits' and each unit has
dimension 'dimInUnits'.

O = O.reshape(numInstances, numUnits, dimOutUnits)
# Output file has 'numInstances', each instance has 'numUnits' and each unit has
dimension 'dimOutUnits'.
```

Implementation using Keras 6/8

7. Train the network

```
    model.fit(X, O, epochs=5)                    # Train the network for 5
epochs
```

```
Epoch 1/5
4/4 [==============================] - 0s - loss: 0.0987
Epoch 2/5
4/4 [==============================] - 0s - loss: 0.0987
Epoch 3/5
4/4 [==============================] - 0s - loss: 0.0986
Epoch 4/5
4/4 [==============================] - 0s - loss: 0.0986
Epoch 5/5
 4/4 [==============================] - 0s - loss: 0.0985
```

Implementation using Keras 7/8

8. Print final *weights*

```
     print (model.layers[0].get_weights())    # Print weights of first layer

[
     array([[-0.4387919]], dtype=float32),          # Input to layer[0]
     array([[ 0.99820316]], dtype=float32),         # layer[0](t-1) to layer[0](t)
     array([-0.00290805], dtype=float32)            # Input bias
]
```

Implementation using Keras 8/8

9. Evaluate the network

   a. Prepare the test data

```
   test = np.random.randint(2, size=10)        # Sequence of 1 & 0 of len
10
```

   a. Predict o/p

```
   prediction = model.predict_classes(test)          # predict o/p
sequence
```

   a. Print test and its prediction

```
   print ('Input seq:', test)
   print ('Output seq:', prediction)

           Input seq:      1 1 0 0 1 0 0 0 1 1
           Output seq: 1 0 0 0 1 1 1 1 1 0
```

Implementation using Keras

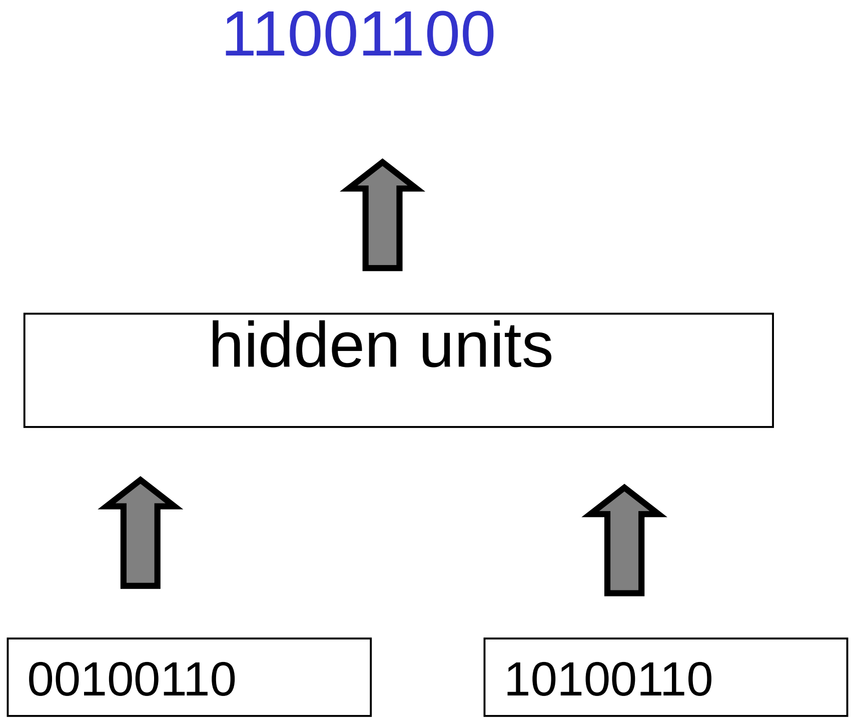
```
  # Import libraries
import numpy
import keras
from keras.models import Sequential
from keras.layers import SimpleRNN

  dimInUnits = numInNeurons = 1
dimOutUnits = numOutNeurons = 1
numUnits = seqLen = 10
numInstances = 4

  # Design network
model = Sequential()
model.add(SimpleRNN(numOutNeurons, input_shape=(seqLen, numInNeurons), return_sequences=True, activation='sigmoid'))
model.compile(optimizer='sgd', loss='mse')
model.summary()

  # Prepare data
X = np.loadtxt(open('','r'))
O = np.loadtxt(open('','r'))
  X = X.reshape(numInstances, numUnits, dimInUnits)
  Y = Y.reshape(numInstances, numUnits, dimOutUnits)

  # Training
model.fit(X, O, epochs=5)
print (model.layers[0].get_weights())

  # Evaluation
test = np.random.randint(2, size=10)
prediction = model.predict_classes(test)
print ('Input seq:', test)
print ('Output seq:', prediction)
```

Backpropagation through time (BPTT algorithm)

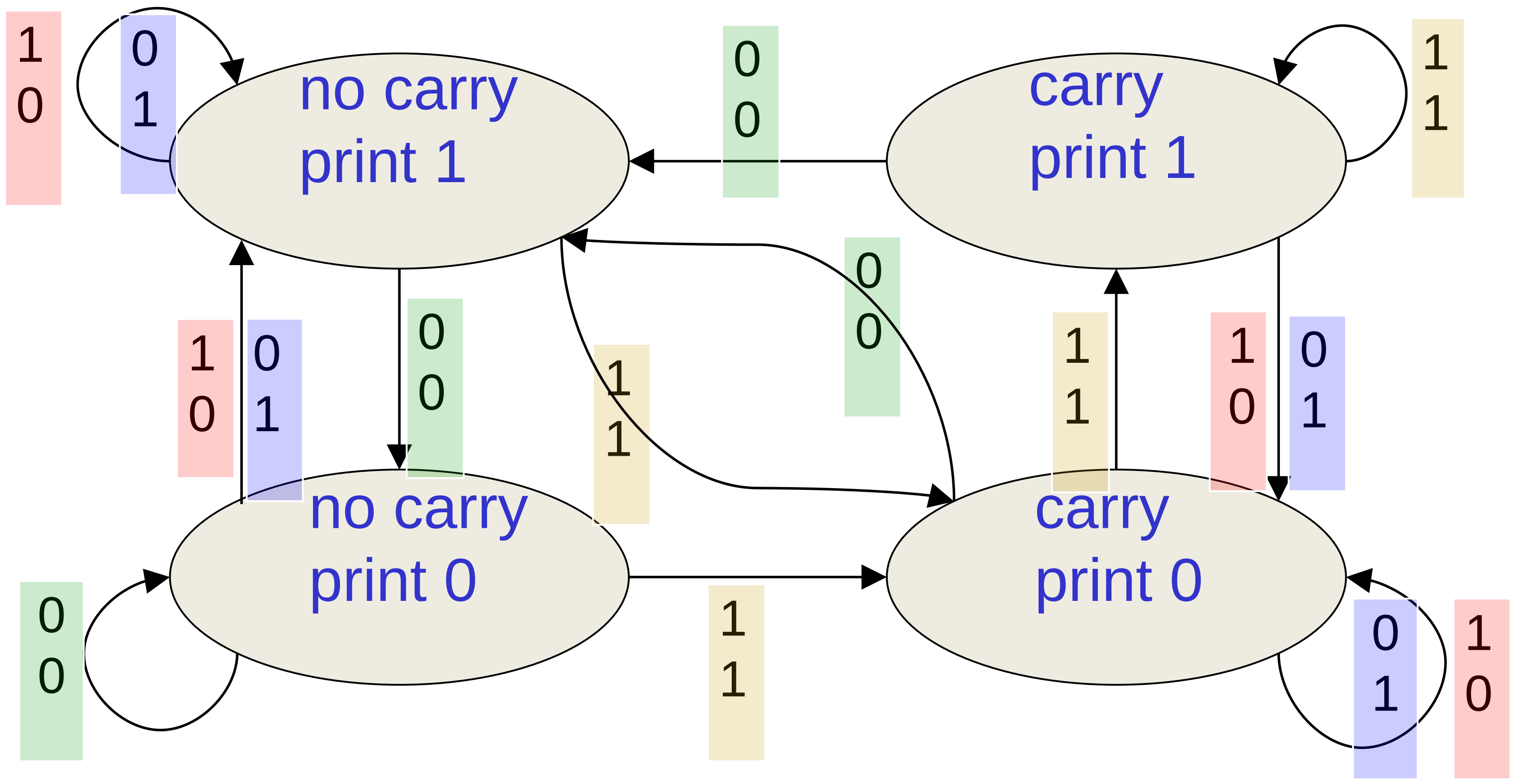
- The forward pass at each time step.
-
- The backward pass computes the error derivatives at each time step.
- After the backward pass we add together the derivatives at all the different times for each weight.

Binary addition using recurrent network (Jeffrey Hinton’s lecture)

- Feed forward n/w
- But problem of variable length input



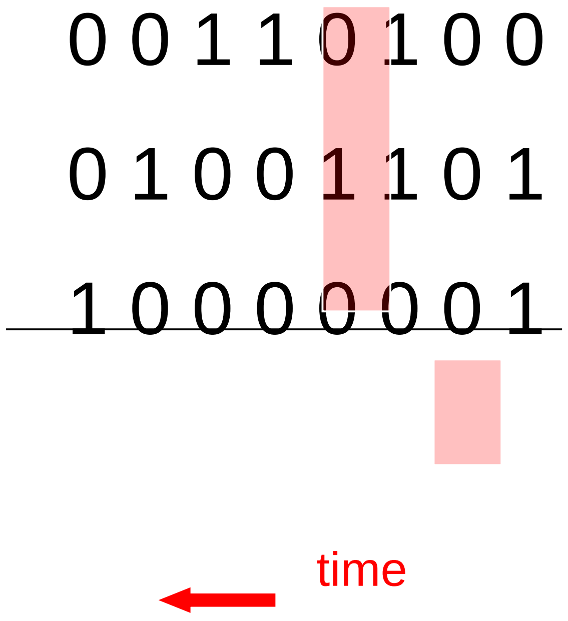
The algorithm for binary addition



This is a finite state automaton. It decides what transition to make by looking at the next column. It prints after making the transition. It moves from right to left over the two input numbers.

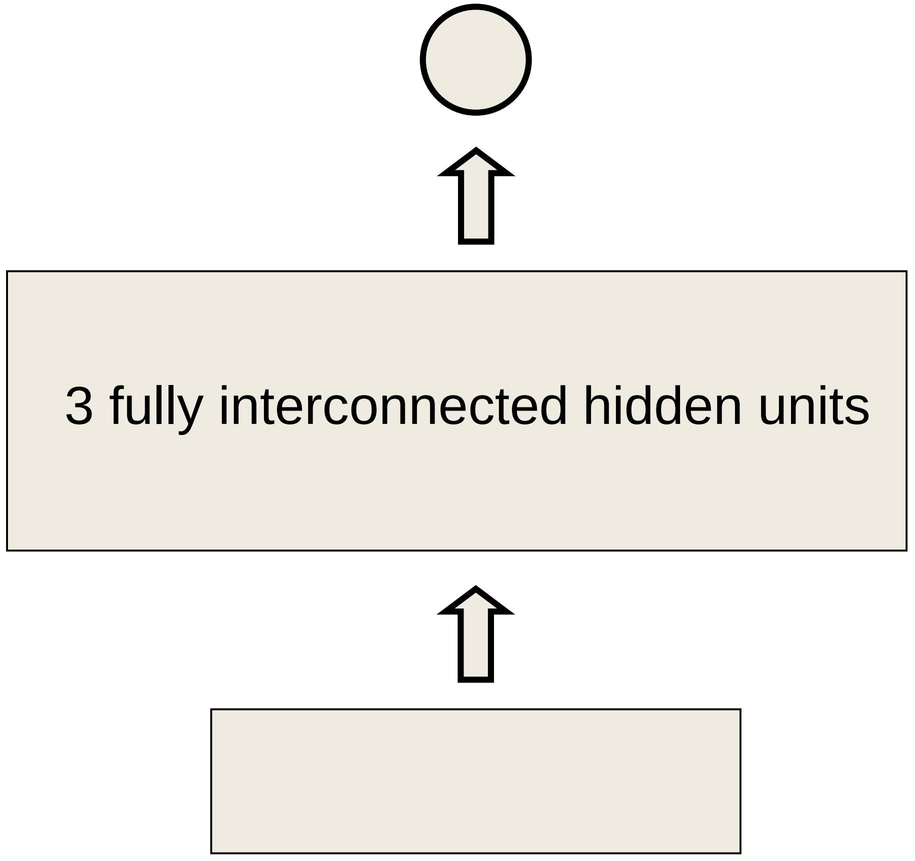
A recurrent net for binary addition

- Two input units and one output unit.
- Given two input digits at each time step.
- The desired output at each time step is the output for the column that was provided as input two time steps ago.
  - It takes one time step to update the hidden units based on the two input digits.
  - It takes another time step for the hidden units to cause the output.



The connectivity of the network

- The input units have feed forward connections
- Allow them to vote for the next hidden activity pattern.

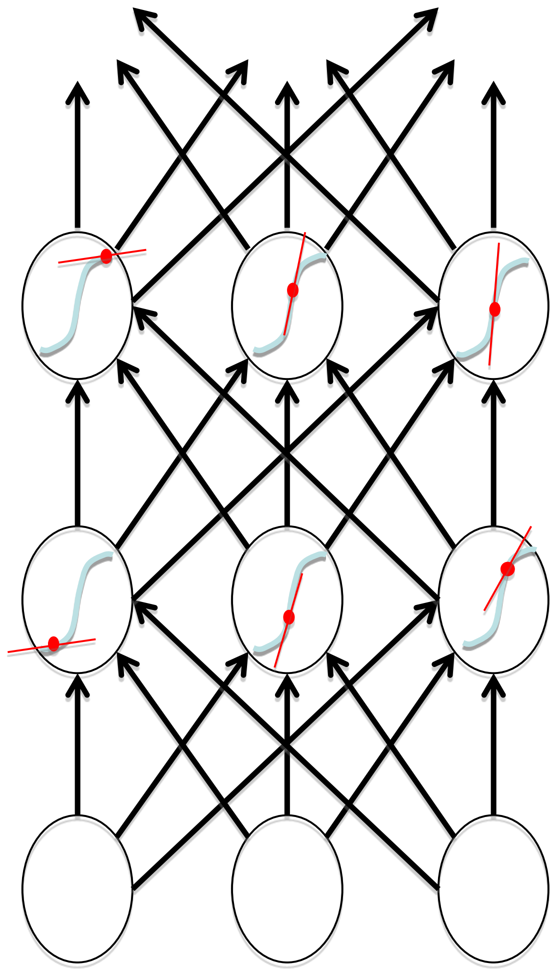


What the network learns

- Learns four distinct patterns of activity for the 3 hidden units.
- Patterns correspond to the nodes in the finite state automaton
- Nodes in FSM are like activity vectors
- The automaton is restricted to be in exactly one state at each time
- The hidden units are restricted to have exactly one vector of activity at each time.

The backward pass is linear

- The backward pass, is completely linear. If you double the error derivatives at the final layer, all the error derivatives will double.
- The forward pass determines the slope of the linear function used for backpropagating through each neuron.



Recall: Backpropagation Rule

- General weight updating rule:

$$\Delta w_{ji} = \eta \delta j o_i$$

- Where

$$\delta_j = (t_j - o_j) o_j (1 - o_j) \quad \text{for outermost layer}$$

$$= \sum_{k \in \text{next layer}} (w_{kj} \delta_k) o_j (1 - o_j) o_i \quad \text{for hidden layers}$$

The problem of exploding or vanishing gradients (1/2)

– If the weights are small, the gradients shrink exponentially

– If the weights are big the gradients grow exponentially.

- Typical feed-forward neural nets can cope with these exponential effects because they only have a few hidden layers.

The problem of exploding or vanishing gradients (2/2)

- In an RNN trained on long sequences (*e.g.* sentence with 20 words) the gradients can easily explode or vanish.
  - We can avoid this by initializing the weights very carefully.
- Even with good initial weights, its very hard to detect that the current target output depends on an input from many time-steps ago.
  - So RNNs have difficulty dealing with long-range dependencies.

Vanishing/Exploding gradient: solution

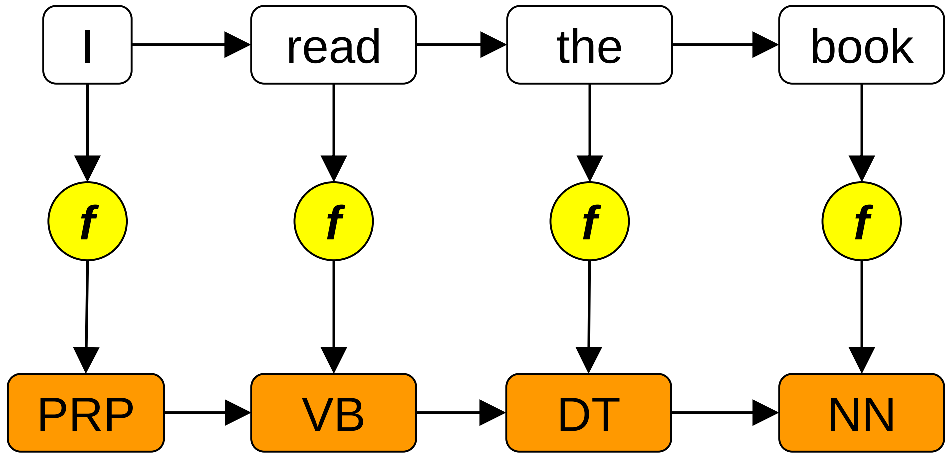
- LSTM
- Error becomes “trapped” in the memory portion of the block
- This is referred to as an "error carousel“
- Continuously feeds error back to each of the gates until they become trained to cut off the value
- (to be expanded)

Attention: DL-POS

Acknowledgement: Anoop Kunchukuttan, IIT Bombay

So far we are seen POS tagging as a sequence labelling task

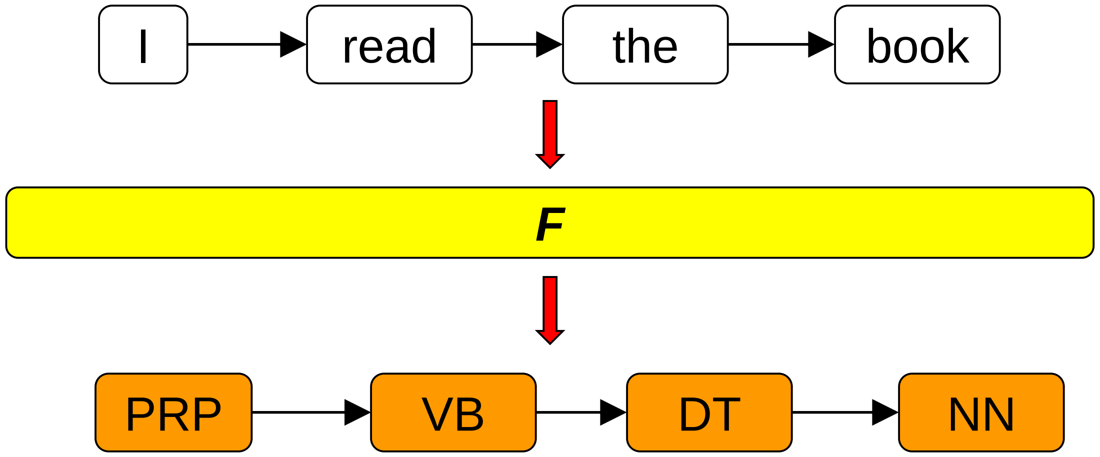
*For every element, predict the tag/label (using function **f**)*



- Length of output sequence is same as input sequence
- Prediction of tag at time *t* can use only the words seen till time *t*

We can also look at POS tagging as a sequence to sequence transformation problem

*Read the entire sequence and predict the output sequence (using function **F**)*



- Length of output sequence need not be the same as input sequence
- Prediction at any time step *t* has access to the entire input
- A more general framework than sequence labelling

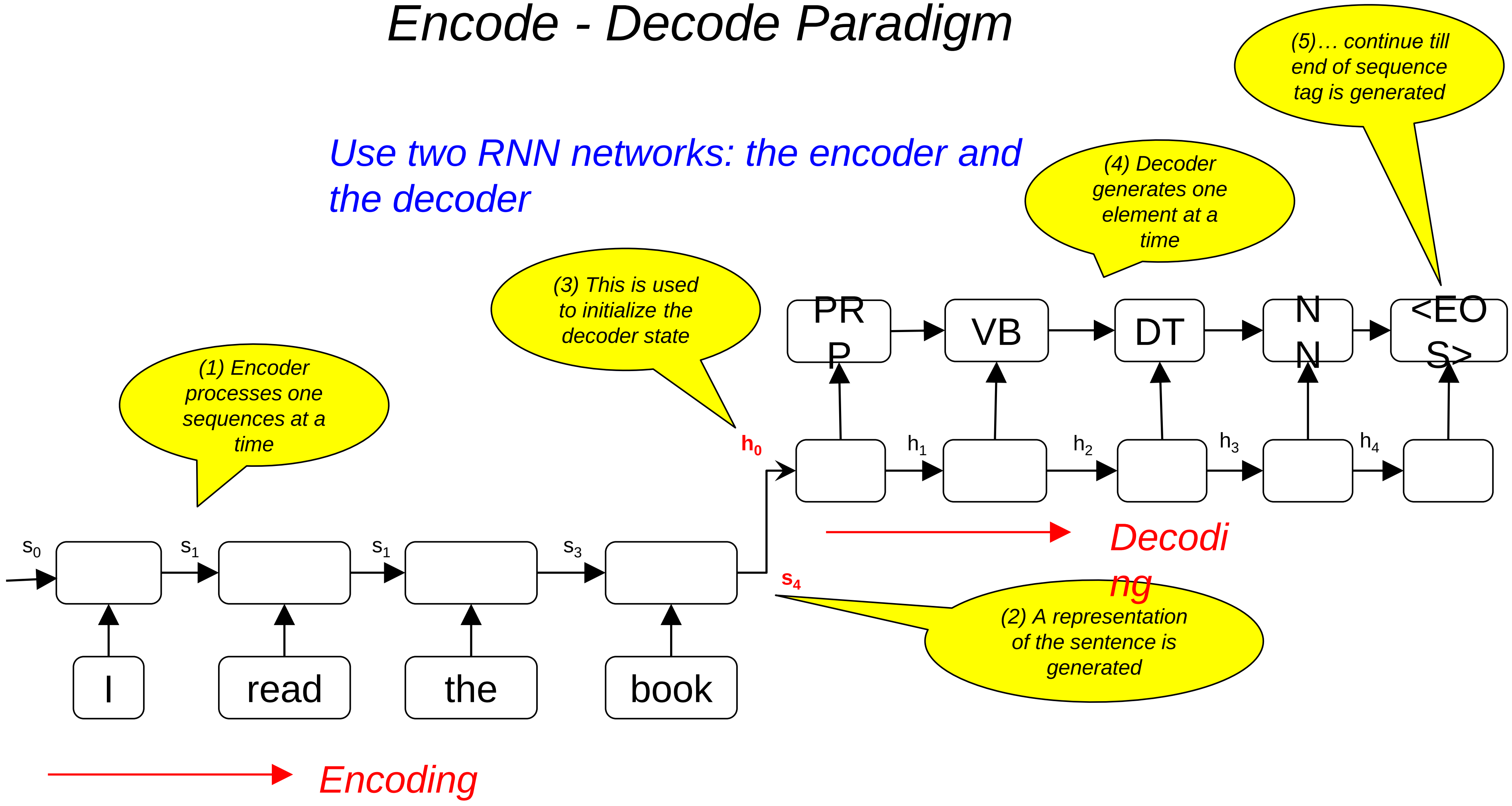
Sequence to Sequence transformation is a more general framework than sequence labelling

- Many other problems can be expressed as sequence to sequence transformation
  - *e.g. machine translation, summarization, question answering, dialog*
- Adds more capabilities which can be useful for problems like MT:
  - many → many mappings: insertion/deletion to words, one-one mappings
  - non-monotone mappings: reordering of words
- For POS tagging, these capabilites are not required

How does a sequence to sequence model work? Let's see two paradigms

Encode - Decode Paradigm

Use two RNN networks: the encoder and the decoder


(1) Encoder processes one sequences at a time
(2) A representation of the sentence is generated
(3) This is used to initialize the decoder state
(4) Decoder generates one element at a time
(5)… continue till end of sequence tag is generated
I
read
the
book
s_0
s_1
s_1
s_3
s_4
h_0
h_1
h_2
h_3
h_4
PRP
VB
DT
NN
<EOS>
Encoding
Decoding

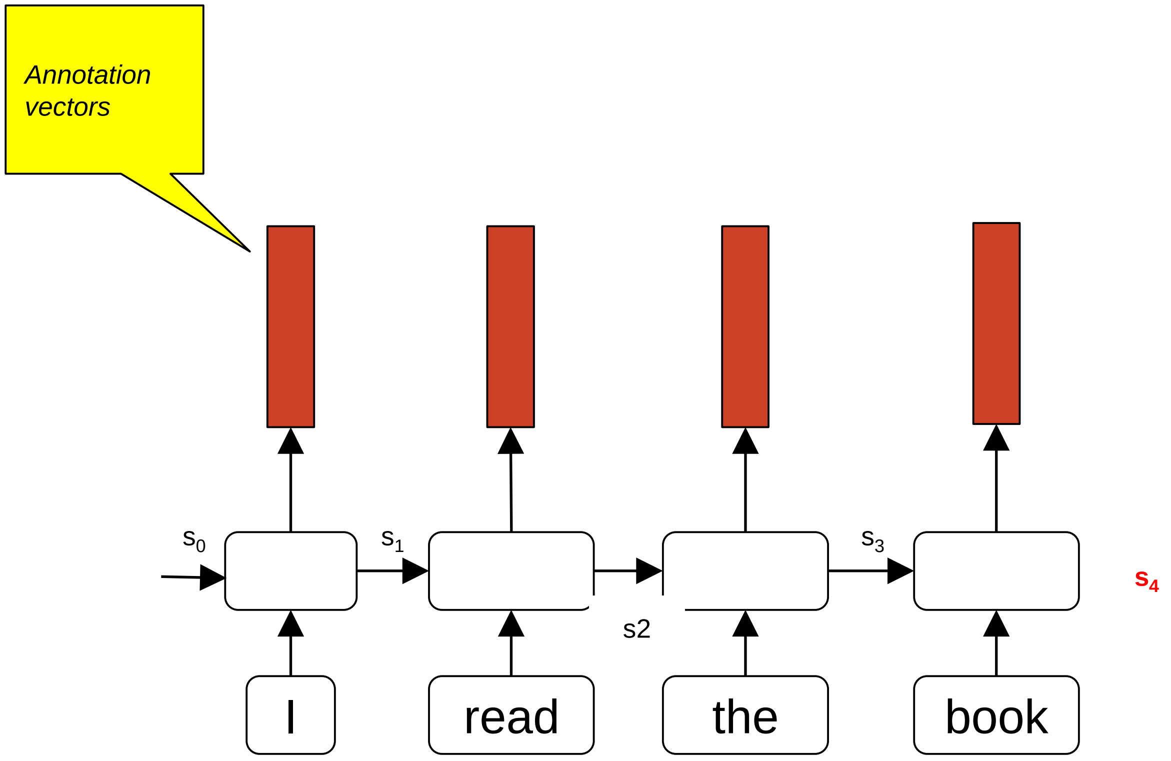
This approach reduces the entire sentence representation to a single vector

Two problems with this design choice:

- This is not sufficient to represent to capture all the syntactic and semantic complexities of a sentence
  - *Solution: Use a richer representation for the sentences*
- Problem of capturing long term dependencies: The decoder RNN will not be able to able to make use of source sentence representation after a few time steps
  - *Solution: Make source sentence information when making the next prediction*
  - *Even better, make **RELEVANT** source sentence information available*

These solutions motivate the next paradigm

Encode - Attend - Decode Paradigm



Represent the source sentence by the **set of output vectors** from the encoder

Each output vector at time *t* is a contextual representation of the input at time *t*

Let’s call these encoder output vectors ***annotation vectors***

How should the decoder use the set of annotation vectors while predicting the next character?

Key Insight:

(1)Not all annotation vectors are equally important for prediction of the next element

(2)The annotation vector to use next depends on what has been generated so far by the decoder

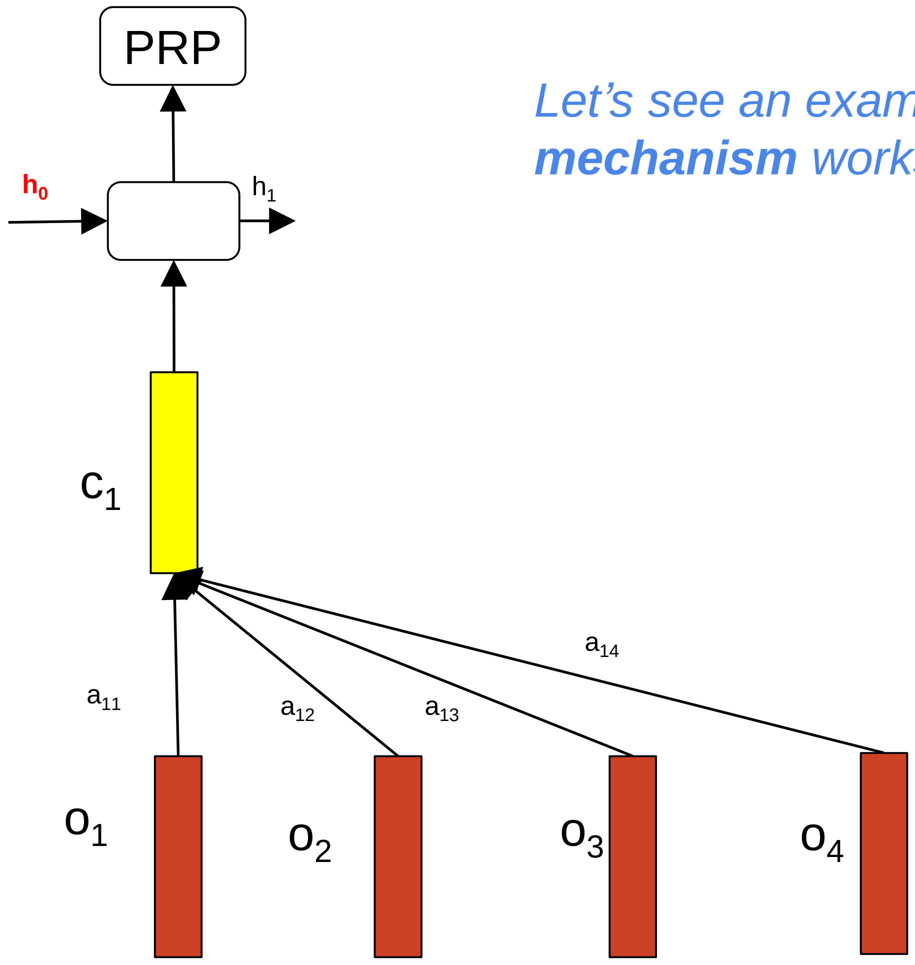
eg. To generate the 3rd POS tag, the 3rd annotation vector (hence 3rd word) is most important

One way to achieve this:
Take a weighted average of the annotation vectors, with more weight to annotation vectors which need more **focus or attention**

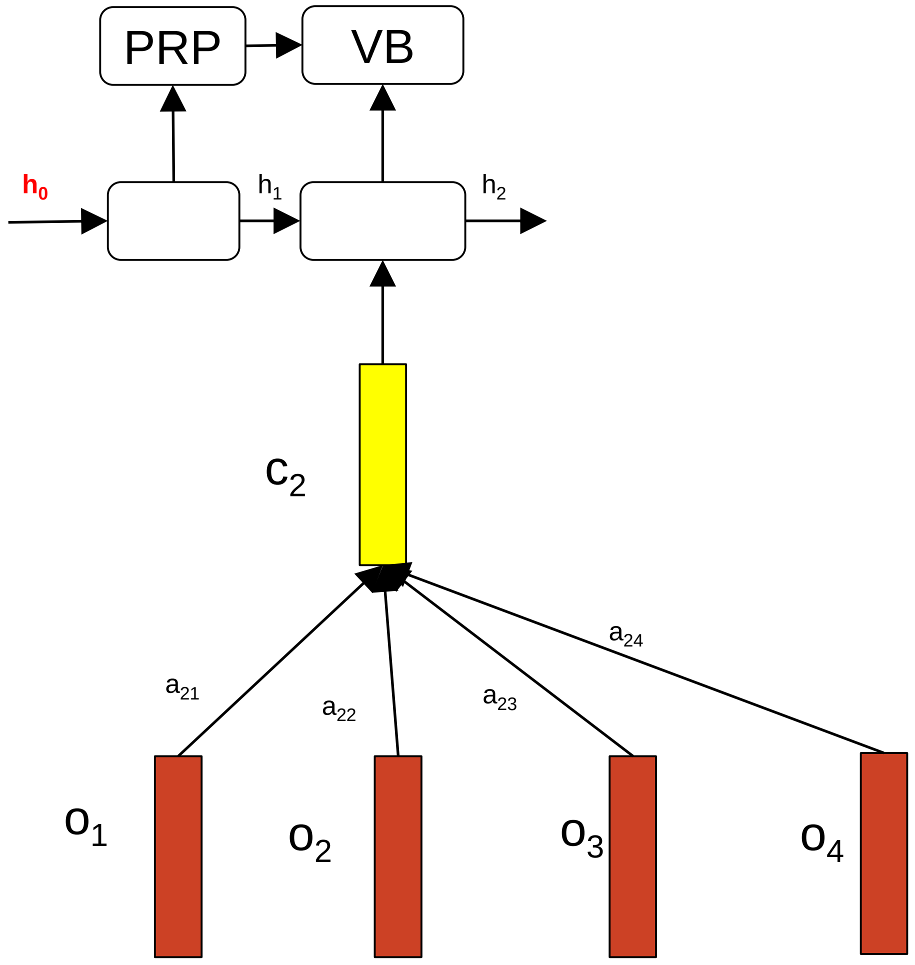
This averaged ***context vector*** is an input to the decoder

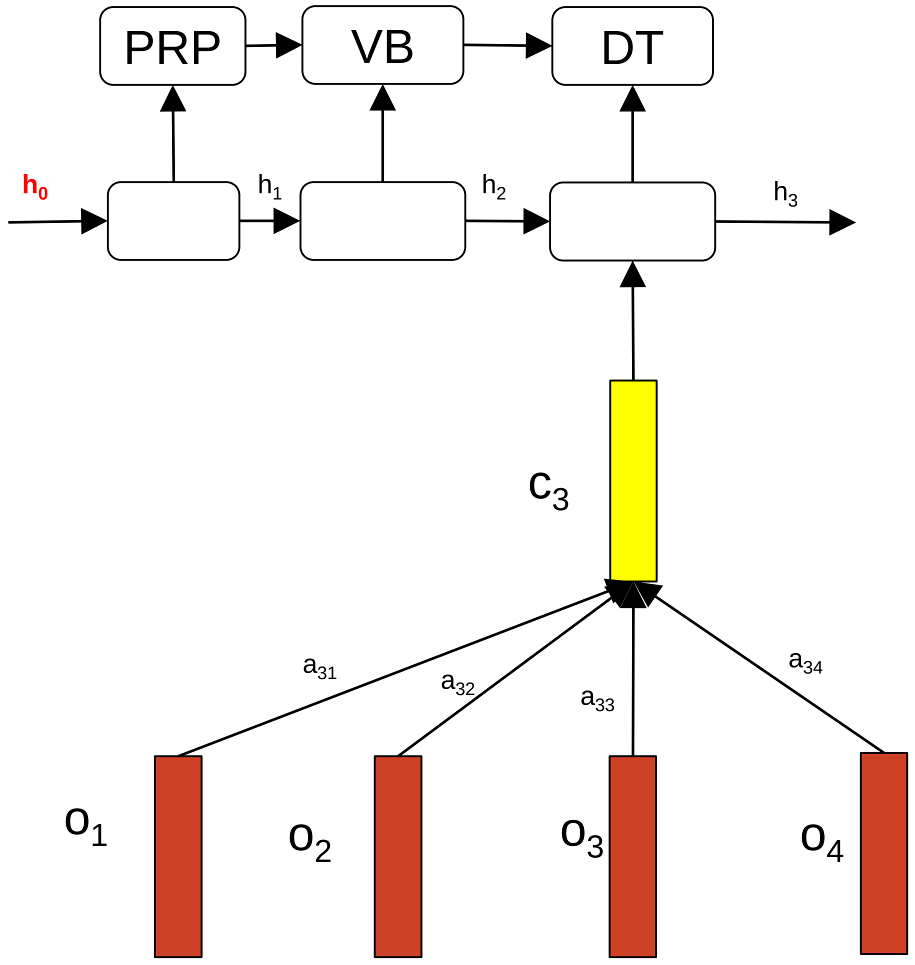
$$c_i = \sum_{j=1}^{n} a_{ij} o_j$$

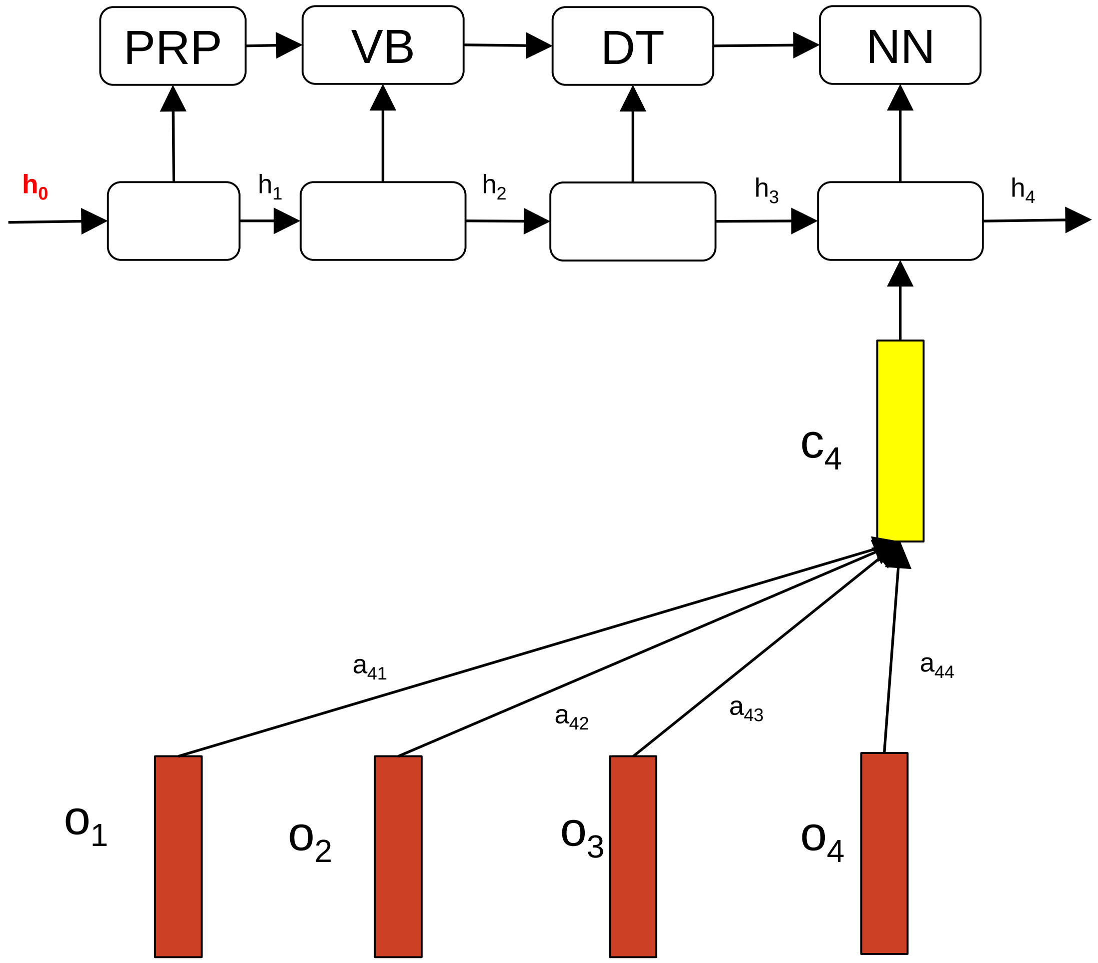
For generation of ith output character:
c_i : context vector
a_{ij} : annotation weight for the jth annotation vector
o_j: jth annotation vector

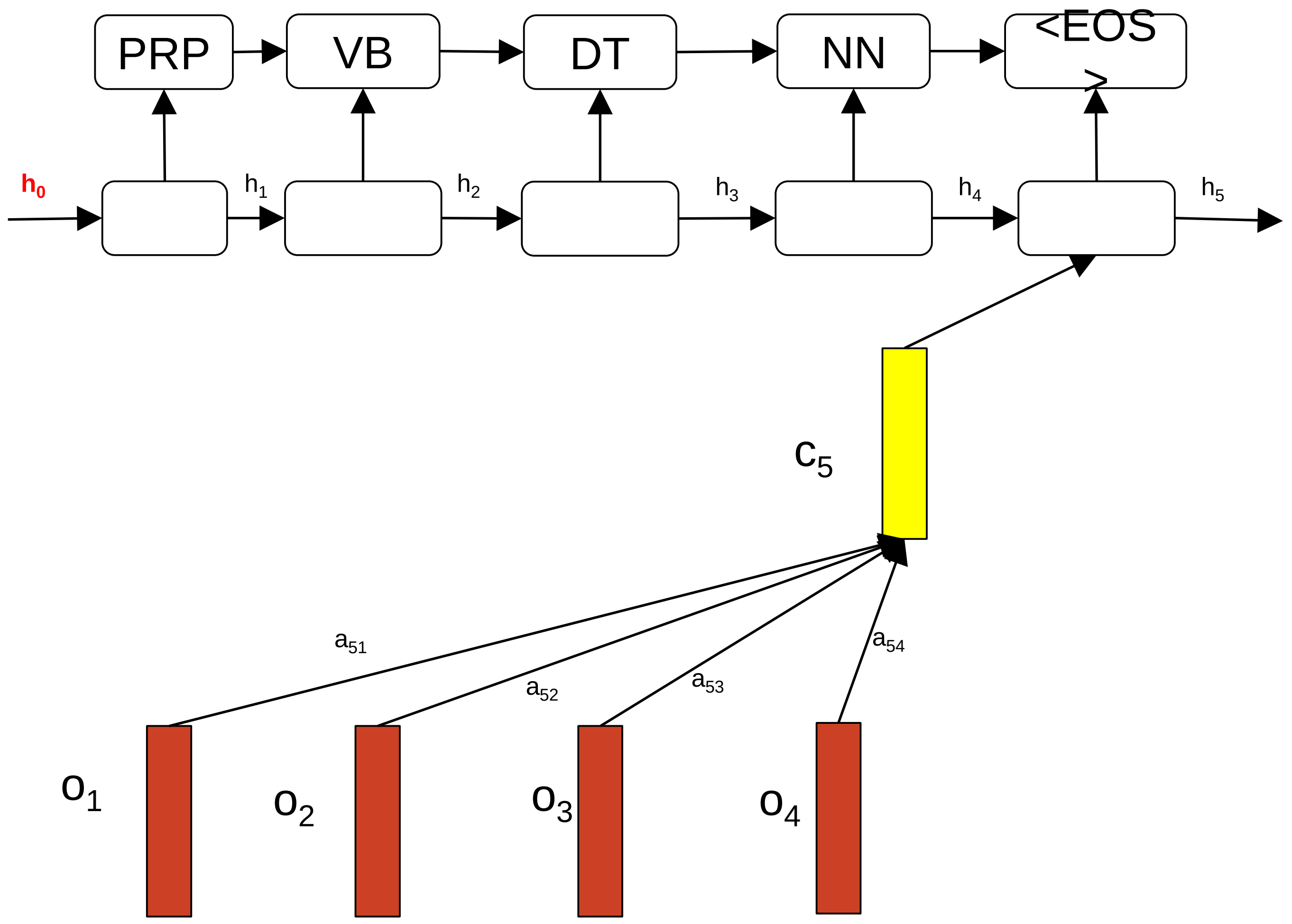


*Let’s see an example of how the **attention mechanism** works*


PRP
VB
h_0
h_1
h_2
c_2
a_{21}
a_{22}
a_{23}
a_{24}
o_1
o_2
o_3
o_4


PRP
VB
DT
h0
h1
h2
h3
c3
a31
a32
a33
a34
o1
o2
o3
o4


PRP
VB
DT
NN
h_0
h_1
h_2
h_3
h_4
c_4
a_{41}
a_{42}
a_{43}
a_{44}
o_1
o_2
o_3
o_4


PRP
VB
DT
NN
<EOS>
h_0
h_1
h_2
h_3
h_4
h_5
c_5
a_{51}
a_{52}
a_{53}
a_{54}
o_1
o_2
o_3
o_4

But we do not know the attention weights?
How do we find them?

Let the training data help you decide!!

Idea: Pick the attention weights that maximize the POS tagging accuracy

(more precisely, decrease training data loss)

Have an attention function that predicts the attention weights:

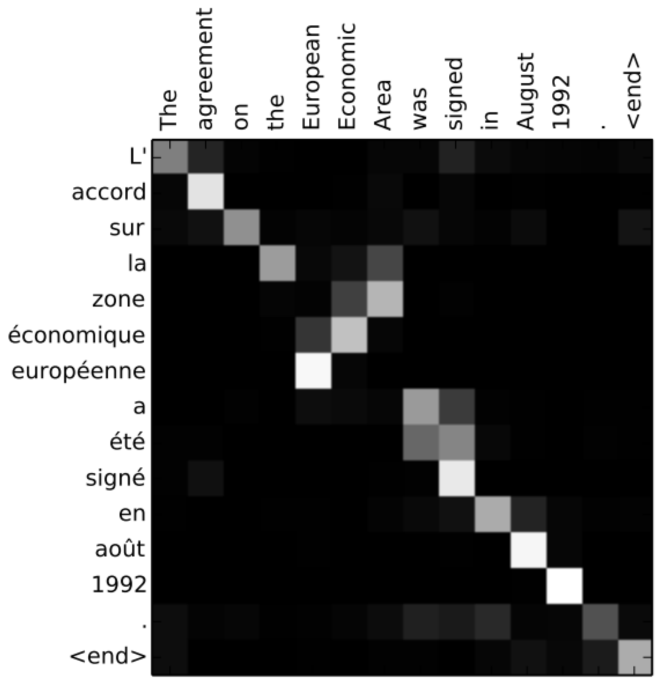
$$a_{ij} = \mathbf{A}(o_j,h_i;\mathbf{o})$$

A could be implemented as a feedforward network which is a component of the overall network

Then training the attention network with the rest of the network ensures that the attention weights are learnt to minimize the translation loss

OK, but do the attention weights actually show focus on certain parts?

Here is an example of how attention weights represent a soft alignment for machine translation



Let’s go back to the encoder. What type of encoder cell should we use there?

- Basic RNN: models sequence history by maintaining state information
  - But, cannot model long range dependencies
- LSTM: can model history and is better at handling long range dependencies

The RNN units model only the sequence seen so far, cannot see the sequence ahead

- Can use a bidirectional RNN/LSTM
- This is just 2 LSTM encoders run from opposite ends of the sequence and resulting output vectors are composed

Both types of RNN units process the sequence sequentially, hence parallelism is limited

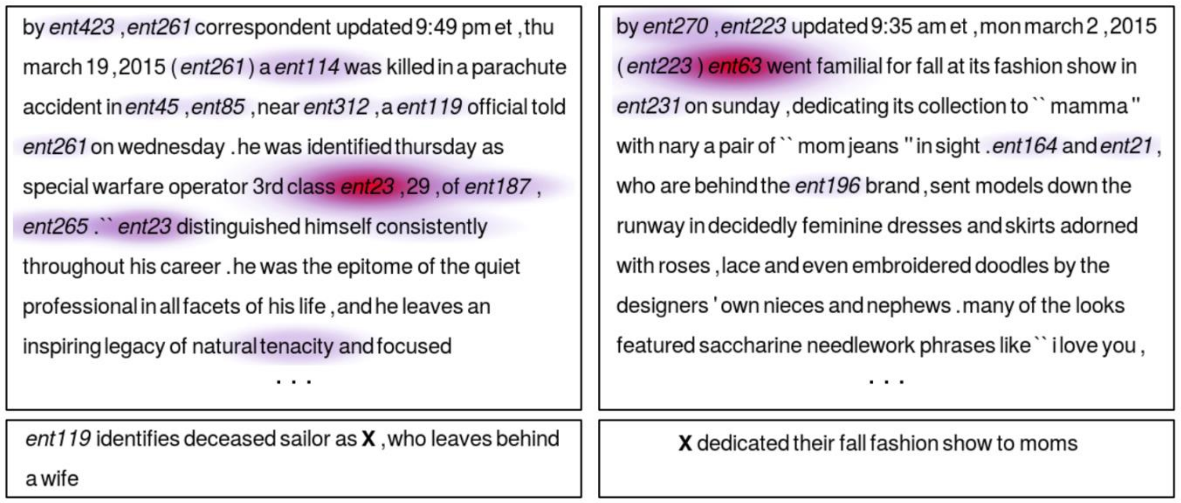
Alternatively, we can use a **CNN**

- Can operate on a sequence in parallel
- However, cannot model entire sequence history
- Model only a short local context. This may be sufficient for some applications or deep CNN layers can overcome the problem

Other applications of Attention

Teaching Machines to Read and Comprehend

Karl Moritz Hermann, Tomáš Kočiský, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, Phil Blunsom, arxiv, 2015



by *ent423* , *ent261* correspondent updated 9:49 pm et , thu march 19 , 2015 (*ent261*) a *ent114* was killed in a parachute accident in *ent45* , *ent85* , near *ent312* , a *ent119* official told *ent261* on wednesday . he was identified thursday as special warfare operator 3rd class *ent23* , 29 , of *ent187* , *ent265* . `` *ent23* distinguished himself consistently throughout his career . he was the epitome of the quiet professional in all facets of his life , and he leaves an inspiring legacy of natural tenacity and focused
. . .

ent119 identifies deceased sailor as **X** , who leaves behind a wife

by *ent270* , *ent223* updated 9:35 am et , mon march 2 , 2015 (*ent223*) *ent63* went familial for fall at its fashion show in *ent231* on sunday , dedicating its collection to `` mamma '' with nary a pair of `` mom jeans '' in sight . *ent164* and *ent21* , who are behind the *ent196* brand , sent models down the runway in decidedly feminine dresses and skirts adorned with roses , lace and even embroidered doodles by the designers ' own nieces and nephews . many of the looks featured saccharine needlework phrases like `` i love you ,
. . .

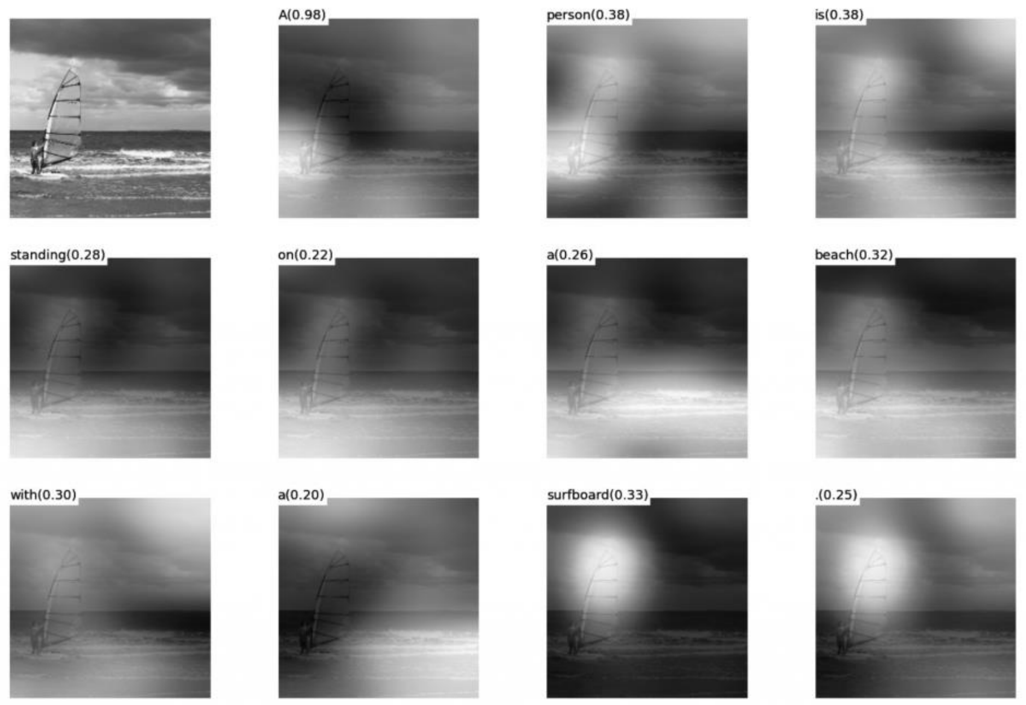
X dedicated their fall fashion show to moms

Used RNN to read a text, read a (synthetically generated) question, and then produce an answer.

By visualizing the attention matrix we can see where the networks “looks” while it tries to find the answer to the question

Show, Attend and Tell: Neural Image Caption Generation with Visual Attention

Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhutdinov, Richard Zemel, Yoshua Bengio



(b) A person is standing on a beach with a surfboard.

Use a Convolutional Neural Network to “encode” the image, and a Recurrent Neural Network with attention mechanisms to generate a description.

By visualizing the attention weights, we interpret what the model is looking at while generating a word

The counts in IBM Model 1

Works by maximizing *P(f|e)* over the entire corpus

For IBM Model 1, we get the following relationship:

$$c(w^f|w^e;f,e)=\frac{t(w^f|w^e)}{t(w^f|w^{e_0})+\square+t(w^f|w^{e_l})}\square.\square$$

$c(w^f|w^e;f,e)$ is the fractional count of the alignment of w^f with w^e in f and e

$t(w^f|w^e)$ is the probability of w^f being the translation of w^e

$\square$ is the count of w^f in f

$\square$ is the count of w^e in e

A PAN Indian SMT Study

PAN Indian SMT

Pan-Indian Language SMT

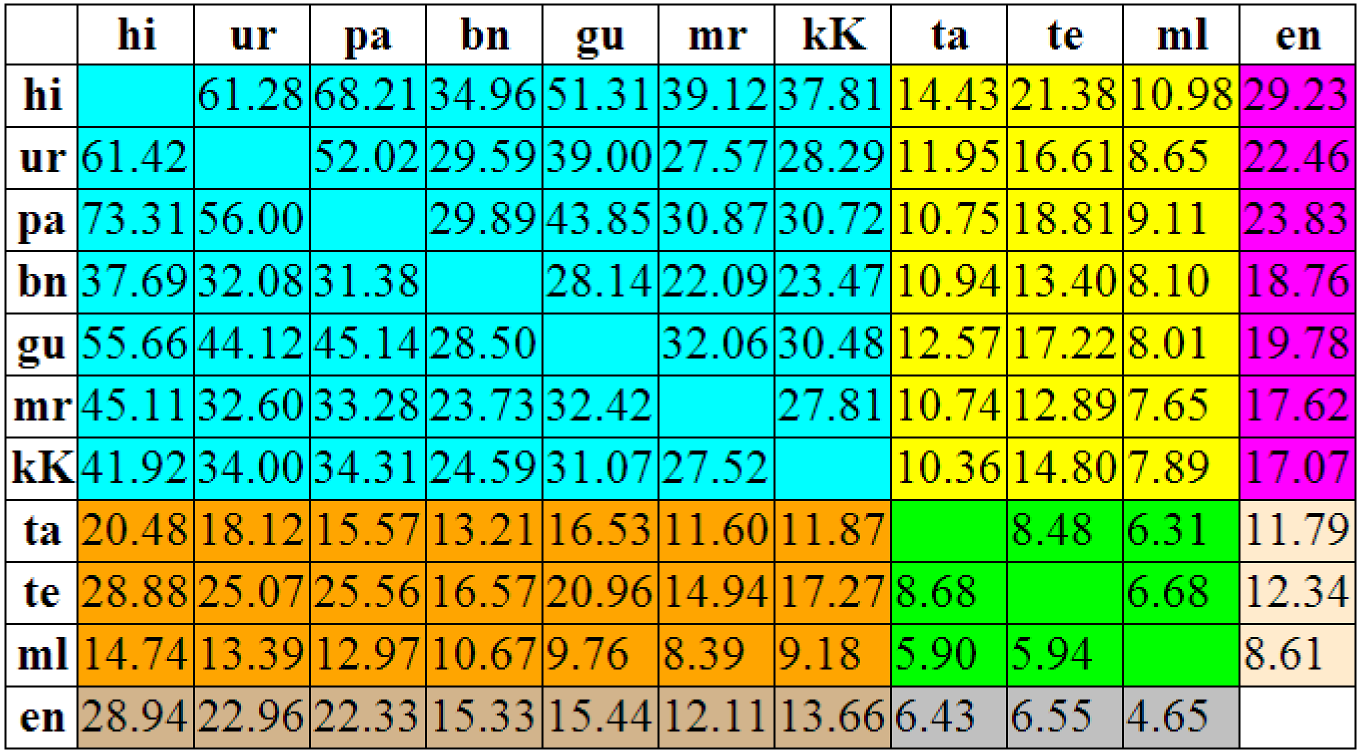
http://www.cfilt.iitb.ac.in/indic-translator

- SMT systems between 11 languages
  - 7 Indo-Aryan: Hindi, Gujarati, Bengali, Oriya, Punjabi, Marathi, Konkani
  - 3 Dravidian languages: Malayalam, Tamil, Telugu
  - English
- Corpus
  - Indian Language Corpora Initiative (ILCI) Corpus
  - Tourism and Health Domains
  - 50,000 parallel sentences
- Evaluation with BLEU
  - METEOR scores also show high correlation with BLEU

SMT Systems Trained

- **Phrase-based** (PBSMT) baseline system (S1)
- E-IL PBSMT with **Source side reordering rules** *(Ramanathan et al., 2008)* (S2)
- E-IL PBSMT with **Source side reordering rules** *(Patel et al., 2013)* (S3)
- IL-IL PBSMT with **transliteration post-editing** (S4)

Natural Partitioning of SMT systems



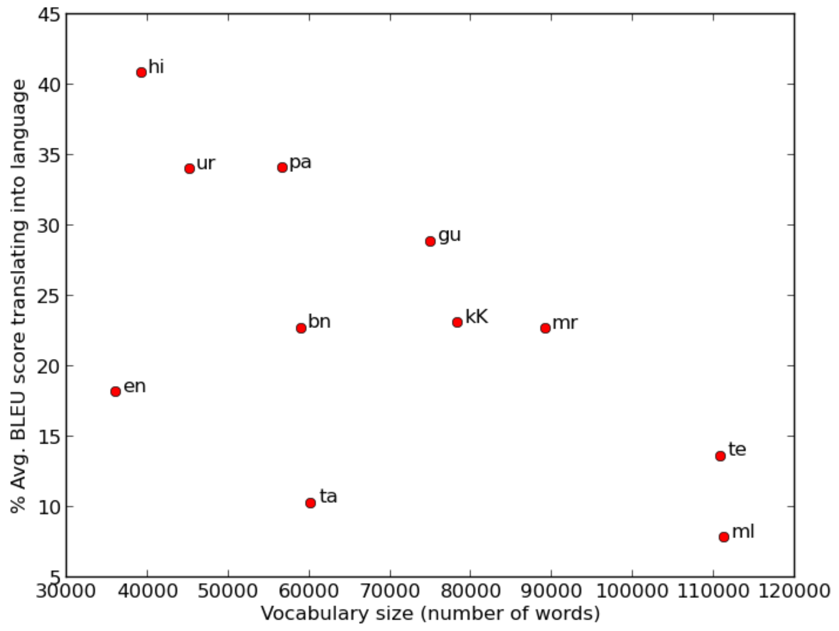
	hi	ur	pa	bn	gu	mr	kK	ta	te	ml	en
hi		61.28	68.21	34.96	51.31	39.12	37.81	14.43	21.38	10.98	29.23
ur	61.42		52.02	29.59	39.00	27.57	28.29	11.95	16.61	8.65	22.46
pa	73.31	56.00		29.89	43.85	30.87	30.72	10.75	18.81	9.11	23.83
bn	37.69	32.08	31.38		28.14	22.09	23.47	10.94	13.40	8.10	18.76
gu	55.66	44.12	45.14	28.50		32.06	30.48	12.57	17.22	8.01	19.78
mr	45.11	32.60	33.28	23.73	32.42		27.81	10.74	12.89	7.65	17.62
kK	41.92	34.00	34.31	24.59	31.07	27.52		10.36	14.80	7.89	17.07
ta	20.48	18.12	15.57	13.21	16.53	11.60	11.87		8.48	6.31	11.79
te	28.88	25.07	25.56	16.57	20.96	14.94	17.27	8.68		6.68	12.34
ml	14.74	13.39	12.97	10.67	9.76	8.39	9.18	5.90	5.94		8.61
en	28.94	22.96	22.33	15.33	15.44	12.11	13.66	6.43	6.55	4.65	

Baseline PBSMT - % BLEU scores (S1)

- **Clear partitioning of translation pairs by language family pairs**, based on translation accuracy.
  - Shared characteristics within language families make translation simpler
  - Divergences among language families make translation difficult

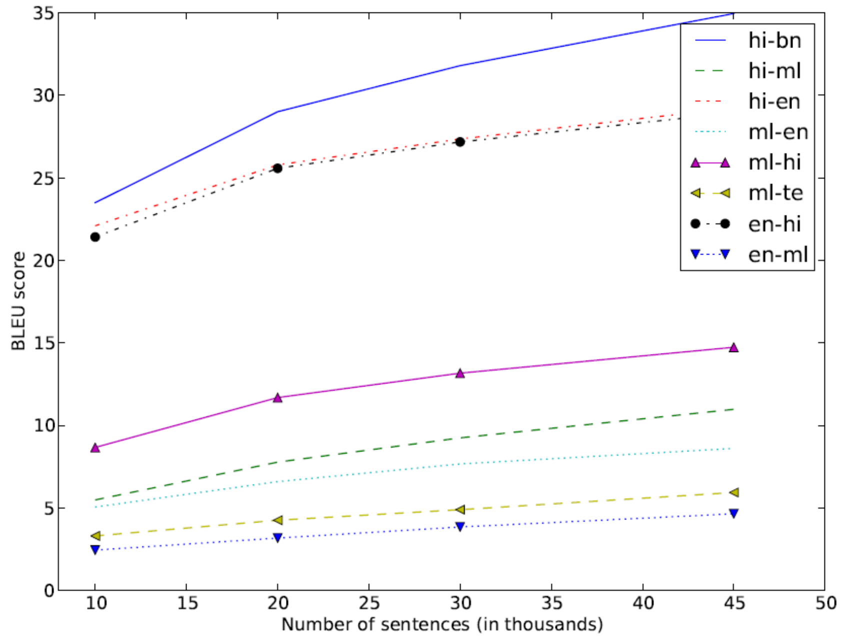
The Challenge of Morphology

Morphological complexity vs BLEU



Vocabulary size is a proxy for morphological complexity
*Note: For Tamil, a smaller corpus was used for computing vocab size

Training Corpus size vs BLEU



- Translation accuracy decreases with increasing morphology
- Even if training corpus is increased, commensurate improvement in translation accuracy is not seen for morphologically rich languages
- ***Handling morphology in SMT is critical***

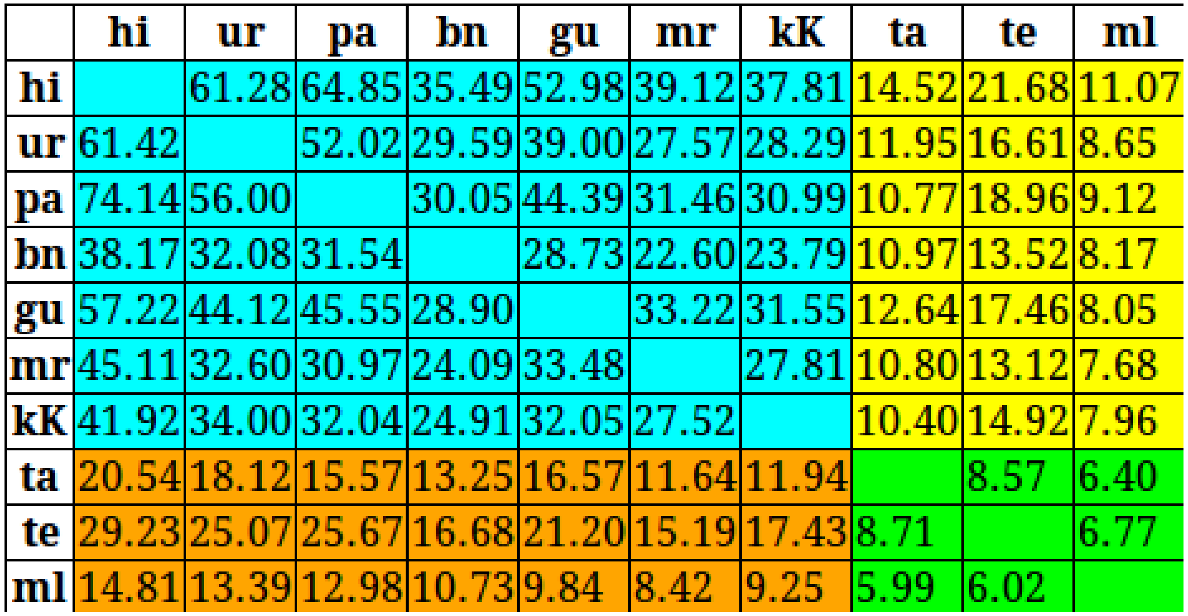
Common Divergences, Shared Solutions

System	hi	ur	pa	bn	gu	mr	kK	ta	te	ml
Baseline PBSMT	28.94	22.96	22.33	15.33	15.44	12.11	13.66	6.43	6.55	4.65
Source Reordering (Generic)	31.41	24.85	24.56	15.89	17.38	13.42	14.55	7.84	8.23	4.95
Source Reordering (Hindi-adapted)	33.54	26.67	26.23	17.86	19.06	14.15	15.56	7.96	8.37	5.30

Comparison of source reordering methods for E-IL SMT - % BLEU scores (S1,S2,S3)

- All Indian languages have similar word order
- The same structural divergence between English and Indian languages *SOV<->SVO, etc.*
- **Common source side reordering rules** improve E-IL translation by 11.4% (generic) and 18.6% (Hindi-adapted)
- ***Common divergences can be handled in a common framework in SMT systems*** (This idea has been used for knowledge based MT systems e.g. *Anglabharati*)

Harnessing Shared Characteristics



	hi	ur	pa	bn	gu	mr	kK	ta	te	ml
hi		61.28	64.85	35.49	52.98	39.12	37.81	14.52	21.68	11.07
ur	61.42		52.02	29.59	39.00	27.57	28.29	11.95	16.61	8.65
pa	74.14	56.00		30.05	44.39	31.46	30.99	10.77	18.96	9.12
bn	38.17	32.08	31.54		28.73	22.60	23.79	10.97	13.52	8.17
gu	57.22	44.12	45.55	28.90		33.22	31.55	12.64	17.46	8.05
mr	45.11	32.60	30.97	24.09	33.48		27.81	10.80	13.12	7.68
kK	41.92	34.00	32.04	24.91	32.05	27.52		10.40	14.92	7.96
ta	20.54	18.12	15.57	13.25	16.57	11.64	11.94		8.57	6.40
te	29.23	25.07	25.67	16.68	21.20	15.19	17.43	8.71		6.77
ml	14.81	13.39	12.98	10.73	9.84	8.42	9.25	5.99	6.02	

PBSMT+ transliteration post-editing for E-IL SMT - % BLEU scores (S4)

- Out of Vocabulary words are transliterated in a post-editing step
- Done using a simple transliteration scheme which harnesses the common phonetic organization of Indic scripts
- Accuracy Improvements of 0.5 BLEU points with this simple approach
- ***Harnessing common characteristics can improve SMT output***

Pubs: http://ww.cse.iitb.ac.in/~pb

Resources and tools:

http://www.cfilt.iitb.ac.in