CS772: Deep Learning for Natural Language Processing (DL-NLP)

RNN, Encoder-Decoder, A*, CNN

Pushpak Bhattacharyya
Computer Science and Engineering Department
IIT Bombay

Week 7 of 13^{th} Feb, 2023

Re-cap

PCA: Example

49 birds: 21 survived in a storm and 28 died.

5 body characteristics given

X_1: body length; X_2: alar extent; X_3: beak and head length

X_4: humerus length; X_5: keel length

Could we have predicted the fate from the body characteristic

$$
\begin{array}{cc}
 & \begin{array}{ccccc} X_1 & X_2 & X_3 & X_4 & X_5 \end{array} \\
R = & \begin{bmatrix}
1.000 & & & & \\
0.735 & 1.000 & & & \\
0.662 & 0.674 & 1.000 & & \\
0.645 & 0.769 & 0.763 & 1.000 & \\
0.605 & 0.529 & 0.526 & 0.607 & 1.000
\end{bmatrix}
\begin{array}{c} X_1 \\ X_2 \\ X_3 \\ X_4 \\ X_5 \end{array}
\end{array}
$$

Eigenvalues and Eigenvectors of *R*

Eigenvalues: 3.612, 0.532, 0.386, 0.302, 0.165

First Eigen-vector: V_1	V_2	V_3	V_4	V_5
0.452	0.462	0.451	0.471	0.398
-0.051	0.300	0.325	0.185	-0.877
0.691	0.341	-0.455	-0.411	-0.179
-0.420	0.548	-0.606	0.388	0.069
0.374	-0.530	-0.343	0.652	-0.192

Which principal components are important?

- Total variance in the data=

  $\lambda_1 + \lambda_2 + \lambda_3 + \lambda_4 + \lambda_5$

  = sum of diagonals of $R = 5$
- First eigenvalue= 3.616 ≈ 72% of total variance 5
- Second ≈ 10.6%, Third ≈ 7.7%, Fourth ≈ 6.0% and Fifth ≈ 3.3%
- ***First PC is the most important and sufficient for studying the classification***

Forming the PCs

- $Z_1 = 0.451X_1+0.462X_2+0.451X_3+0.471X_4+0.398X_5$
- $Z_2 = -0.051X_1+0.300X_2+0.325X_3+0.185X_4 -0.877X_5$
- For all the 49 birds find the first two principal components
- This becomes the new data
- Classify using them

For the first bird

X_1=156, X_2=245, X_3=31.6, X_4=18.5, X_5=20.5

After standardizing

Y_1=(156-157.98)/3.65=-0.54,

Y_2=(245-241.33)/5.1=0.73,

Y_3=(31.6-31.5)/0.8=0.17,

Y_4=(18.5-18.46)/0.56=0.05,

Y_5=(20.5-20.8)/0.99=-0.33

PC_1 *for the first bird=*

Z_1= 0.45X(-0.54)+ 0.46X(0.725)+0.45X(0.17)+0.47X(0.05)+0.39X(-0.33)

=0.064

Similarly, Z_2= 0.602

Reduced Classification Data

- Instead of

X_1	X_2	X_3	X_4	X_5
		49 rows		

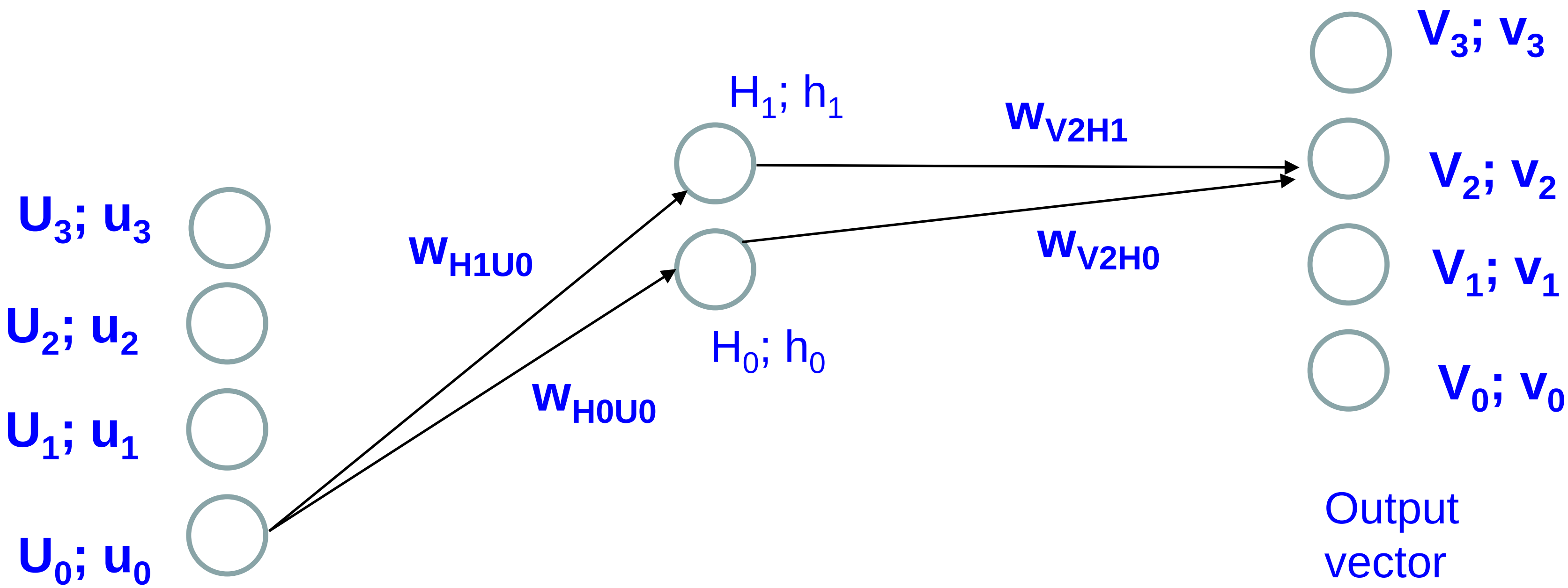
- Use

Z_1	Z_2
49	rows

Working out a simple case of word2vec

Word2vec n/w

Capital letter for NAME of neuron; small letter for output from the same neuron



Output vector V

Input vector U

Weights go from all neurons to all neurons in the next layer; shown For only one input and output

Computing Δw_{V2H0}

$$\Delta w_{V_2H_0} = -\eta \frac{\delta E}{\delta w_{V_2H_0}}$$

$$E = -net_{V_2} + \log(e^{net_{V_0}} + e^{net_{V_1}} + e^{net_{V_2}} + e^{net_{V_3}})$$

$$= -W_{U_0}W_{V_2}^T + \log(e^{net_{V_0}} + e^{net_{V_1}} + e^{net_{V_2}} + e^{net_{V_3}})$$

$$W_{U_0}W_{V_2}^T = w_{V_2H_0}w_{H_0U_0} + w_{V_2H_1}w_{H_1U_0}$$

$$\frac{\delta E}{\delta w_{V_2H_0}} = -w_{H_0U_0} + \frac{e^{W_{V_2}.W_{U_0}}}{e^{W_{V_0}.W_{U_0}} + e^{W_{V_1}.W_{U_0}} + e^{W_{V_2}.W_{U_0}} + e^{W_{V_3}.W_{U_0}})}.w_{H_0U_0}$$

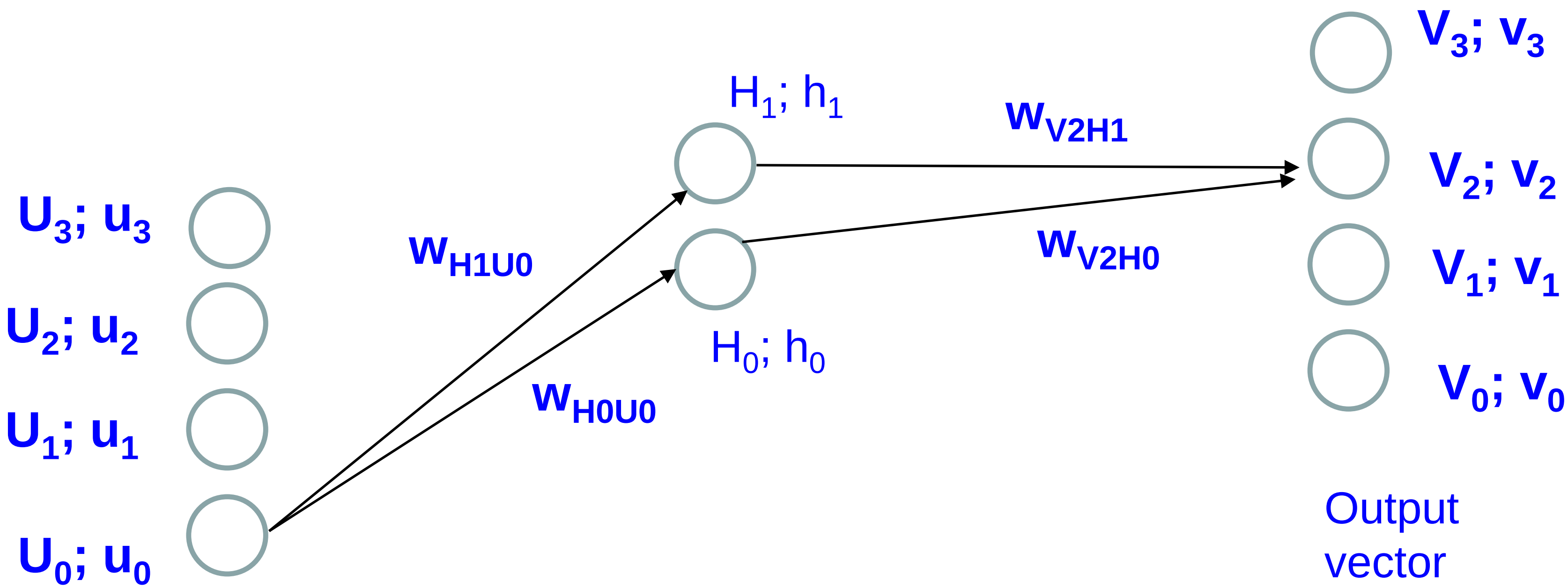
$$= -w_{H_0U_0} + v_2.w_{H_0U_0}$$

$$\Rightarrow \Delta w_{V_2H_0} = \eta(1 - v_2).w_{H_0U_0} = \eta(1 - v_2)o_{H_0}$$

o/p of hidden neuron H_0

Word2vec n/w

Capital letter for NAME of neuron; small letter for output from the same neuron



Output vector V

Input vector U

Weights go from all neurons to all neurons in the next layer; shown For only one input and output

Change in other weights to output layer, say, *V_1*, due to input *U_0*

$$\Delta w_{V_1 H_0} = -\eta \frac{\delta E}{\delta w_{V_1 H_0}}$$

$$E = -net_{V_2} + \log(e^{net_{V0}} + e^{net_{V1}} + e^{net_{V2}} + e^{net_{V3}})$$

$$= -W_{U_0} W_{V_2}^T + \log(e^{net_{V0}} + e^{net_{V1}} + e^{net_{V2}} + e^{net_{V3}})$$

$$W_{U_0} W_{V_2}^T = w_{V_2 H_0} w_{H_0 U_0} + w_{V_2 H_1} w_{H_1 U_0}$$

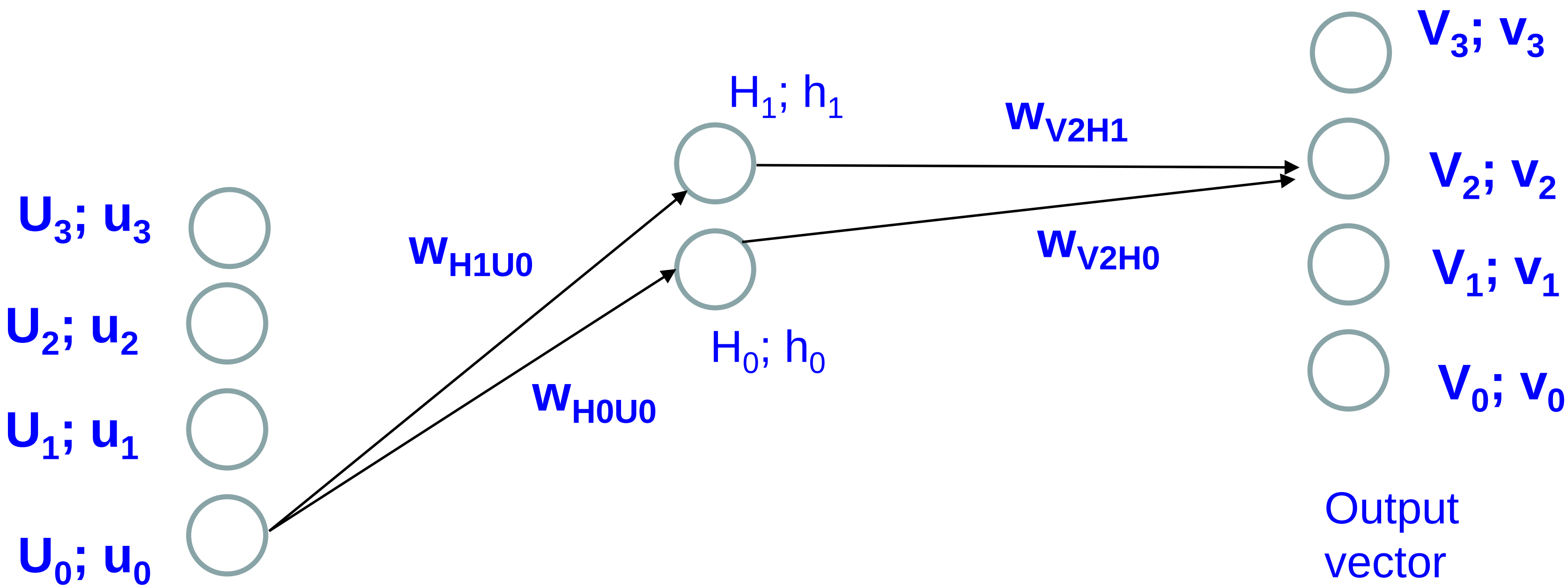
$$\frac{\delta E}{\delta w_{V_1 H_0}} = -0 + \frac{e^{W_{V1} . W_{U0}}}{e^{W_{V0} . W_{U0}} + e^{W_{V1} . W_{U0}} + e^{W_{V2} . W_{U0}} + e^{W_{V3} . W_{U0}})} . w_{H_0 U_0}$$

$$= v_1 . w_{H_0 U_0}$$

$$\Rightarrow \Delta w_{V_1 H_0} = -\eta v_1 w_{H_0 U_0} = -\eta v_1 o_{H_0}$$

Word2vec n/w

Capital letter for NAME of neuron; small letter for output from the same neuron



Output vector V

Input vector U

Weights go from all neurons to all neurons in the next layer; shown For only one input and output

Cntd: Weight change for input to hidden layer, say, w_{H0U0}

$$\frac{\delta E}{\delta w_{H_0U_0}}$$

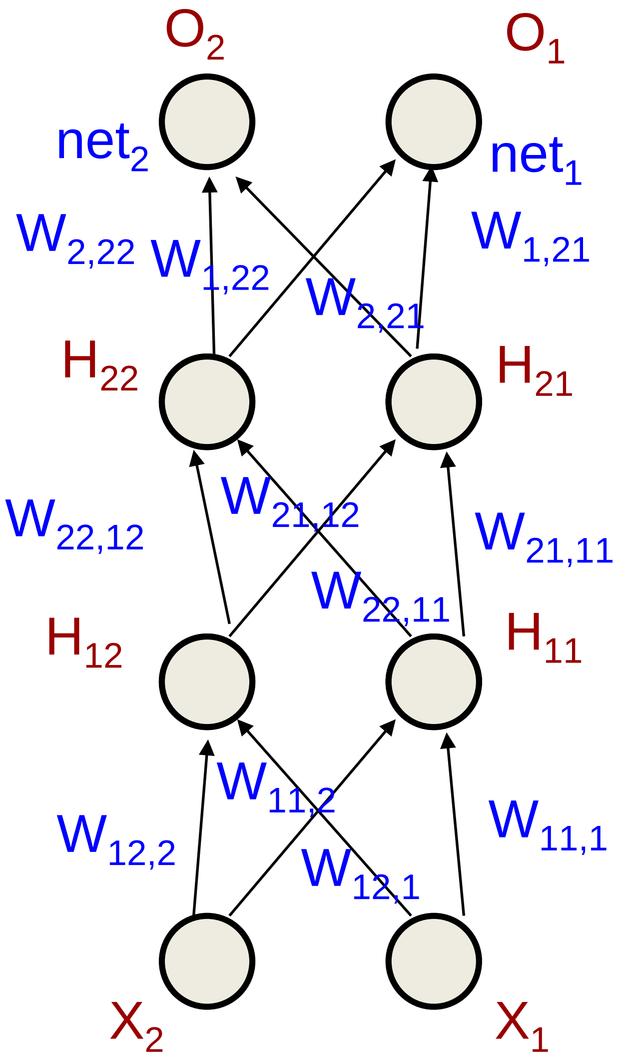
$$= -w_{V_2H_0} + \frac{w_{V_0H_0}e^{W_{V_0}.W_{U_0}} + w_{V_1H_0}e^{W_{V_1}.W_{U_0}} + w_{V_2H_0}e^{W_{V_2}.W_{U_0}} + w_{V_3H_0}e^{W_{V3}.W_{U_0}}}{e^{W_{V_0}.W_{U_0}} + e^{W_{V_1}.W_{U_0}} + e^{W_{V_2}.W_{U_0}} + e^{W_{V_3}.W_{U_0}}}$$

$$= -w_{V_2H_0} + w_{V_0H_0}v_0 + w_{V_1H_0}v_1 + w_{V_2H_0}v_2 + w_{V_3H_0}v_3$$

$$\Rightarrow \Delta w_{H_0U_0} = \eta[(1-v_2)w_{V_2H_0} - w_{V_0H_0}v_0 - w_{V_1H_0}v_1 - w_{V_3H_0}v_3]$$

Softmax, Cross Entropy and RELU

FFNN with O_1-O_2 softmax, all hidden neurons RELU, Cross Entropy Loss



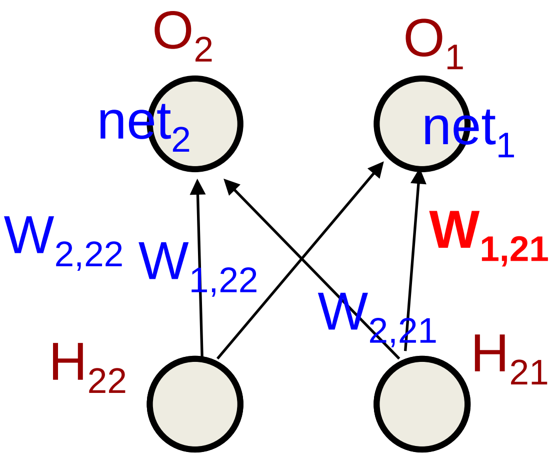
We will apply the $\Delta w_{ji}=\eta\delta_j o_i$ rule

Gradient Descent Rule and the General Weight Change Equation

$$\Delta w_{1,21} = \eta \delta_{o_1} h_{21}$$

$$\delta_{o_1} = -\frac{\delta E}{\delta net_1}$$

$$E = -t_2 \log o_2 - t_1 \log o_1$$



$$\frac{\partial E}{\partial net_1} = \frac{\partial E}{\partial o_1} \cdot \frac{\partial o_1}{\partial net_1} + \frac{\partial E}{\partial o_2} \cdot \frac{\partial o_2}{\partial net_1}$$

$$= -\frac{t_1}{o_1} o_1(1-o_1) + (-\frac{t_2}{o_2})(-o_1 o_2)$$

$$= -t_1(1-o_1) + t_2 o_1$$

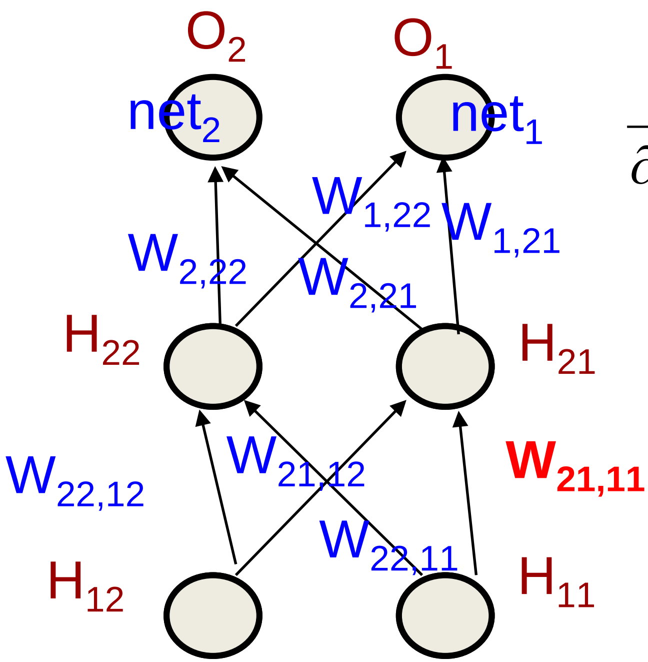
$$= -t_1 o_2 + t_2 o_1 = -(t_1 - o_1)$$

$$\Rightarrow \delta_{O_1} = (t_1 - o_1)$$

$$Similarly, \delta_{O_2} = (t_2 - o_2)$$

$$\boxed{\Delta W_{1,21} = \eta(t_1 - o_1) h_{21}}$$

Weight Change for Hidden Layer, $W_{21,11}$



$$\Delta w_{21,11} = -\eta \frac{\partial E}{\partial w_{21,11}} = \eta \delta_{H_{21}} h_{11}$$

$$\delta_{H_{21}} = -\frac{\partial E}{\partial net_{H_{21}}}$$

$$\frac{\partial E}{\partial net_{H_{21}}} = \frac{\partial E}{\partial h_{21}} . \frac{\partial h_{21}}{\partial net_{H_{21}}} ; h_{21} = output(H_{21})$$

$$= \frac{\partial E}{\partial h_{21}} .r'(H_{21}); \quad r' = derivative\_RELU(H_{21})$$

$$\frac{\partial E}{\partial h_{21}} = \frac{\partial E}{\partial net_1} . \frac{\partial net_1}{\partial h_{21}} + \frac{\partial E}{\partial net_2} . \frac{\partial net_2}{\partial h_{21}}$$

$$= (-\delta_{o_1}).W_{1,21} + (-\delta_{o2}).W_{2,21}$$

$$\Rightarrow \delta_{H_{21}} = (\delta_{o_1}.W_{1,21} + \delta_{o2}.W_{2,21}).r'(H_{21})$$

$$= backpropagated\_delta.RELU\_derivative$$

$$\boxed{\Delta W_{21,11} = \eta[(t_2 - o_2)W_{2,21} + (t_1 - o_1)W_{1,21}].r'(H_{21}).h_{11}}$$

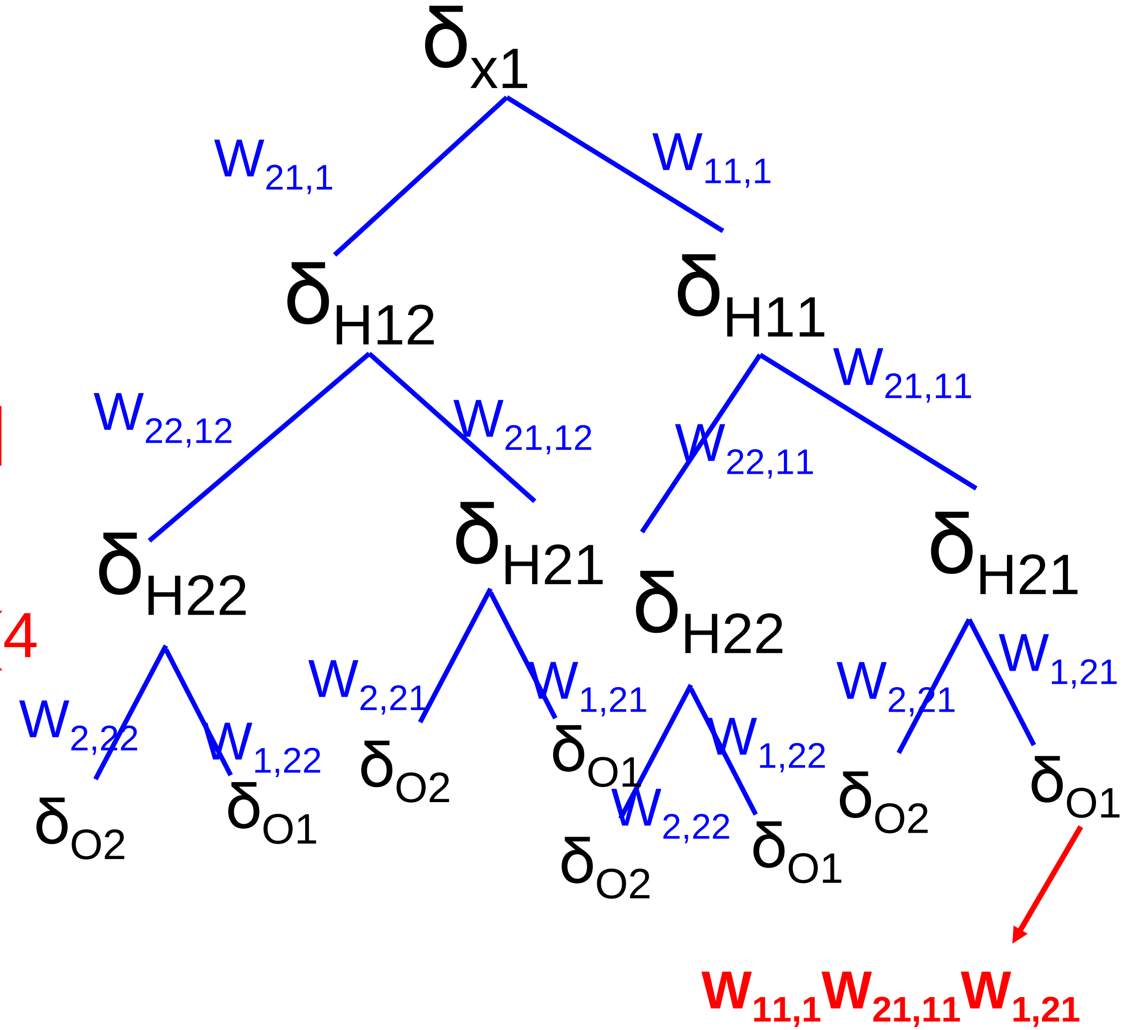
Vanishing/Exploding Gradient

$\delta_{x1}=W_{11,1}\delta_{H11}+W_{21,1}\delta_{H12}$ [2 terms]

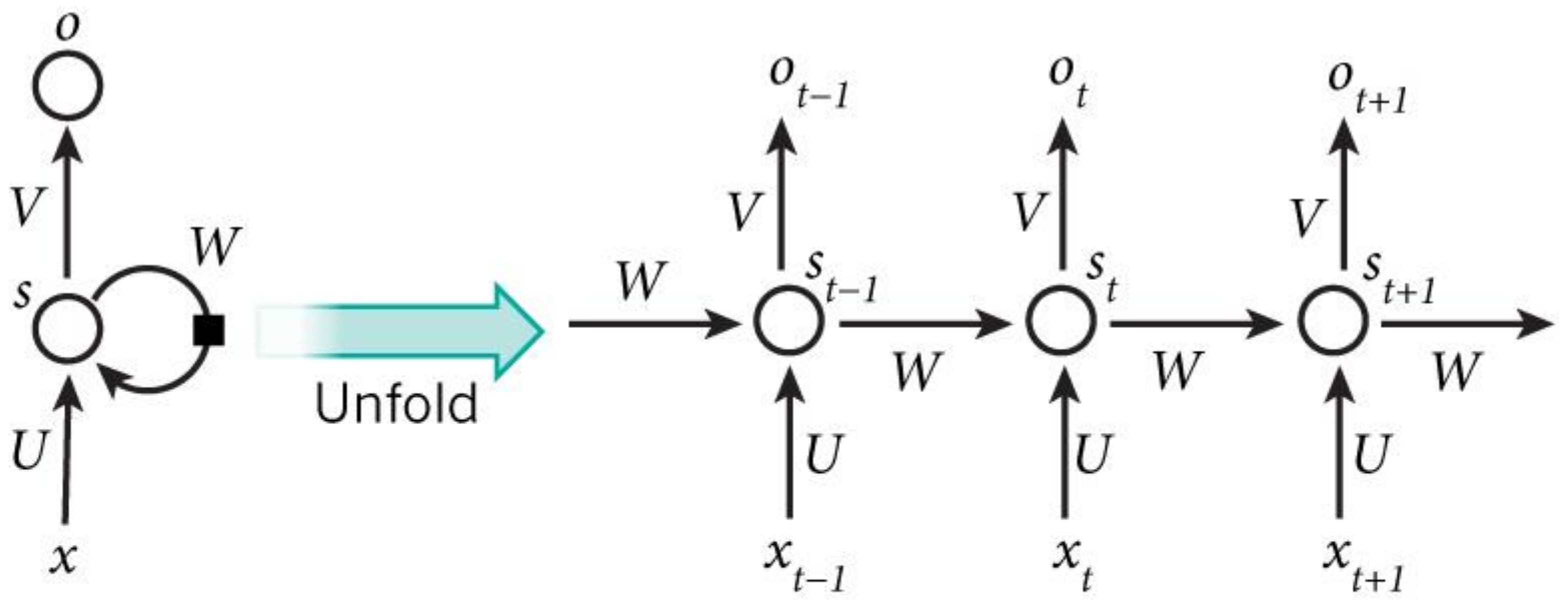
$=W_{11,1}(W_{21,11}\delta_{H21}+W_{22,11}\delta_{H22}).r'(H_{11})+W_{21,1}(W_{21,12}\delta_{H21}+W_{22,12}\delta_{H22}).\ r'(H_{12})$ [4 terms]

= (4 terms involving δ_{o1}) + (4 terms involving δ_{o2})

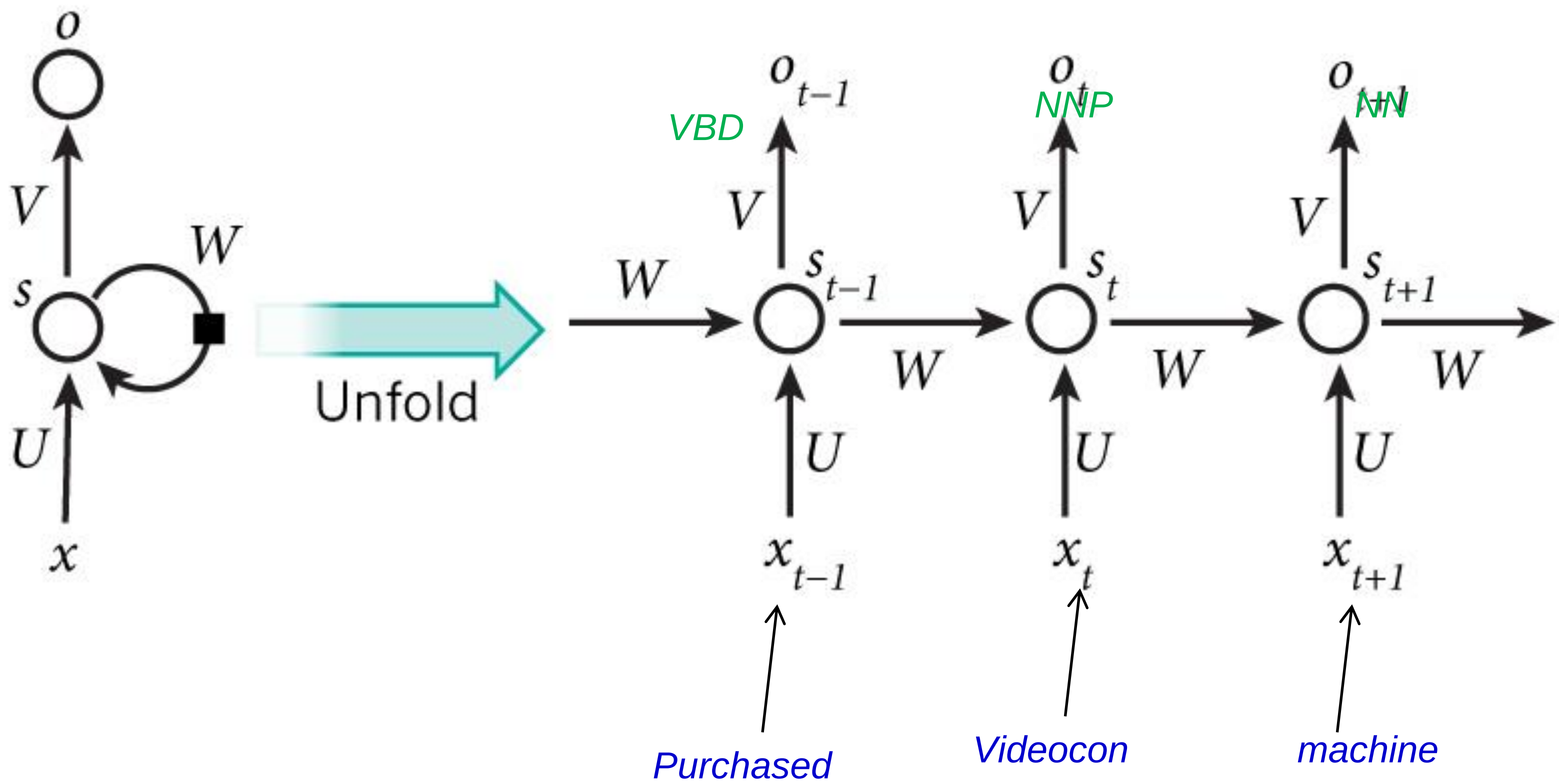
δs get multiplied by derivatives of RELU which are 1 or 0; hence δs from the output layer pass as such or as 0



RNN: Sequence processing m/c



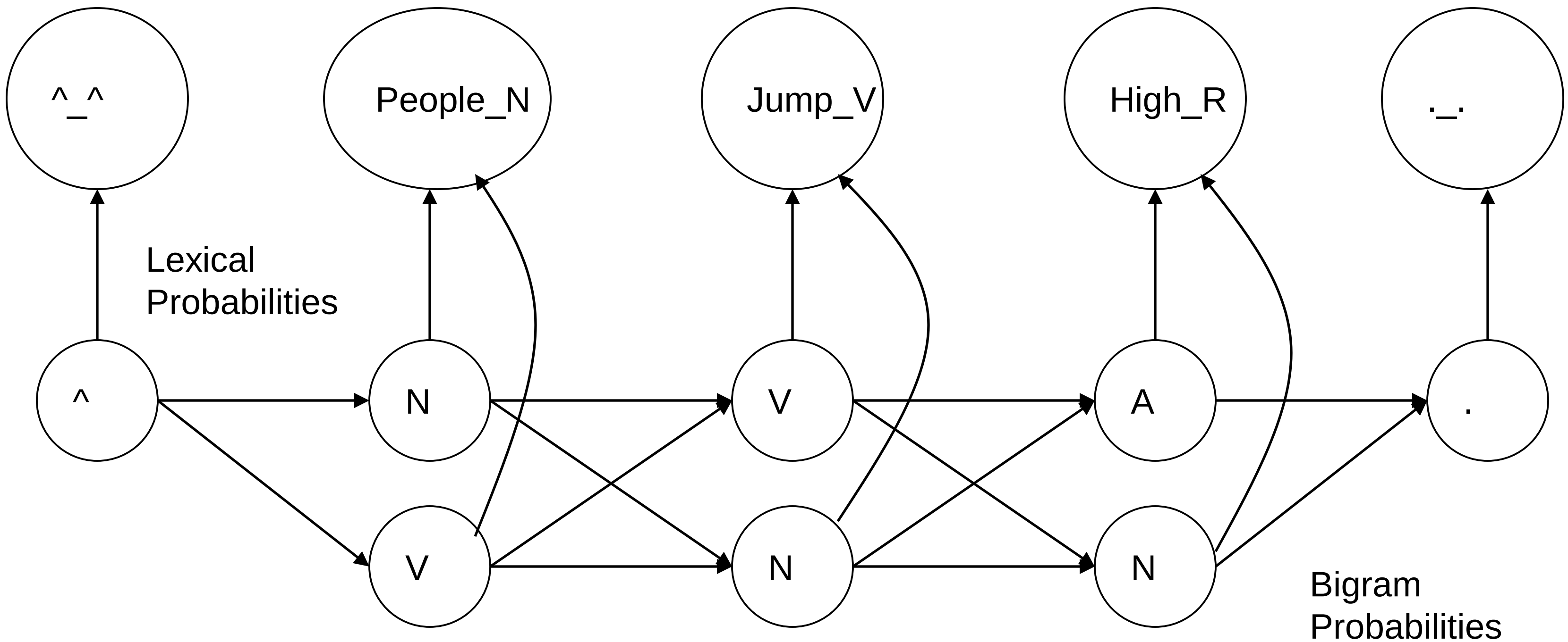
POS Tagging with RNN



Note that POS of "purchased" is ambiguous with possibilities as VBD or VBN or JJ

"I purchased Videocon machine" vs. "my purchased Videocon machine is running well"

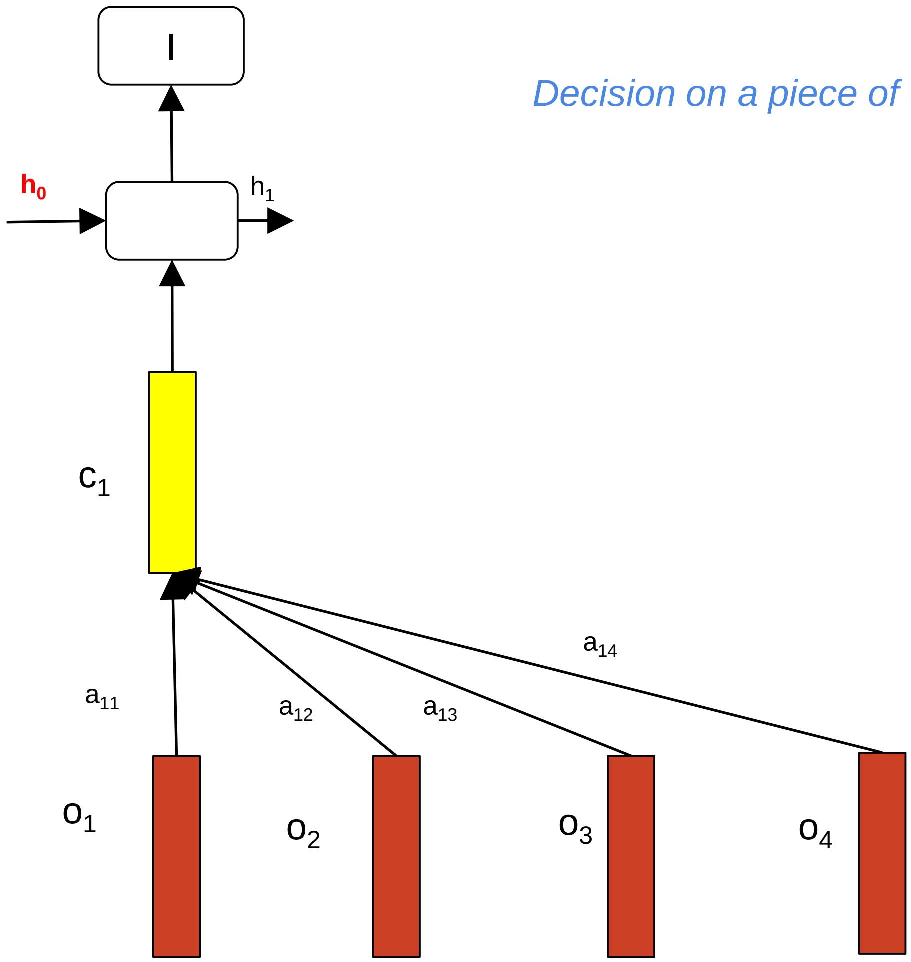
HMM: Generative Model

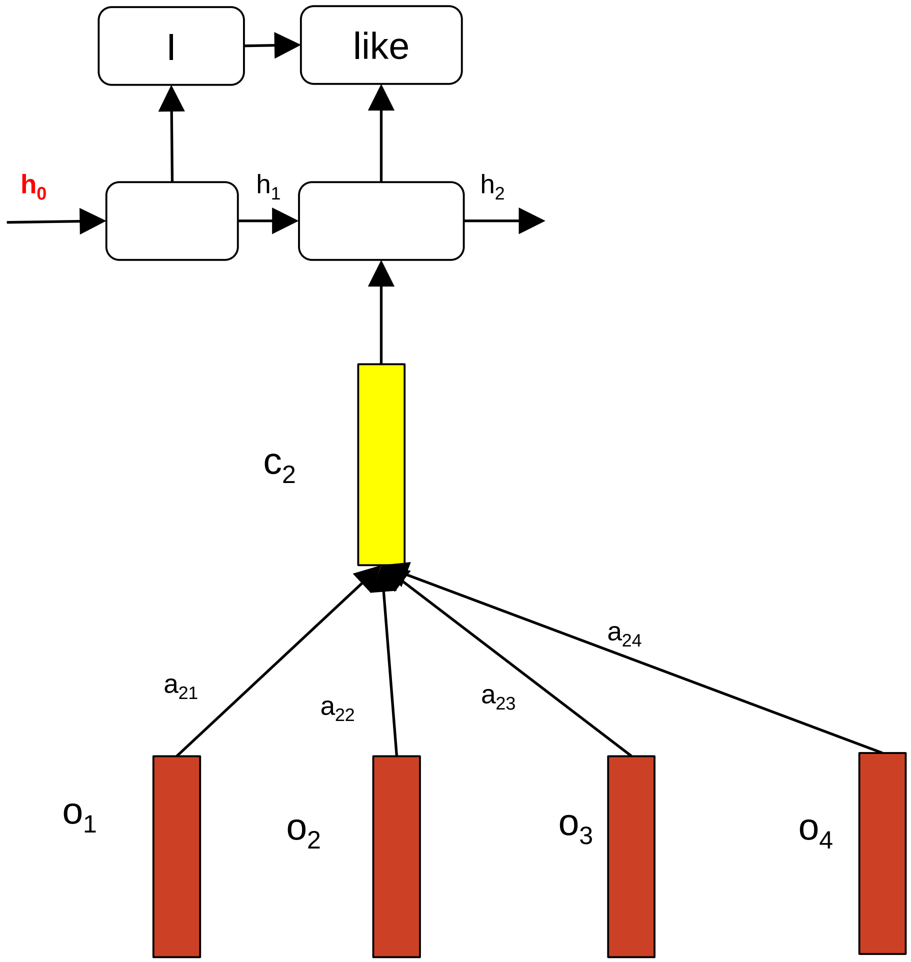


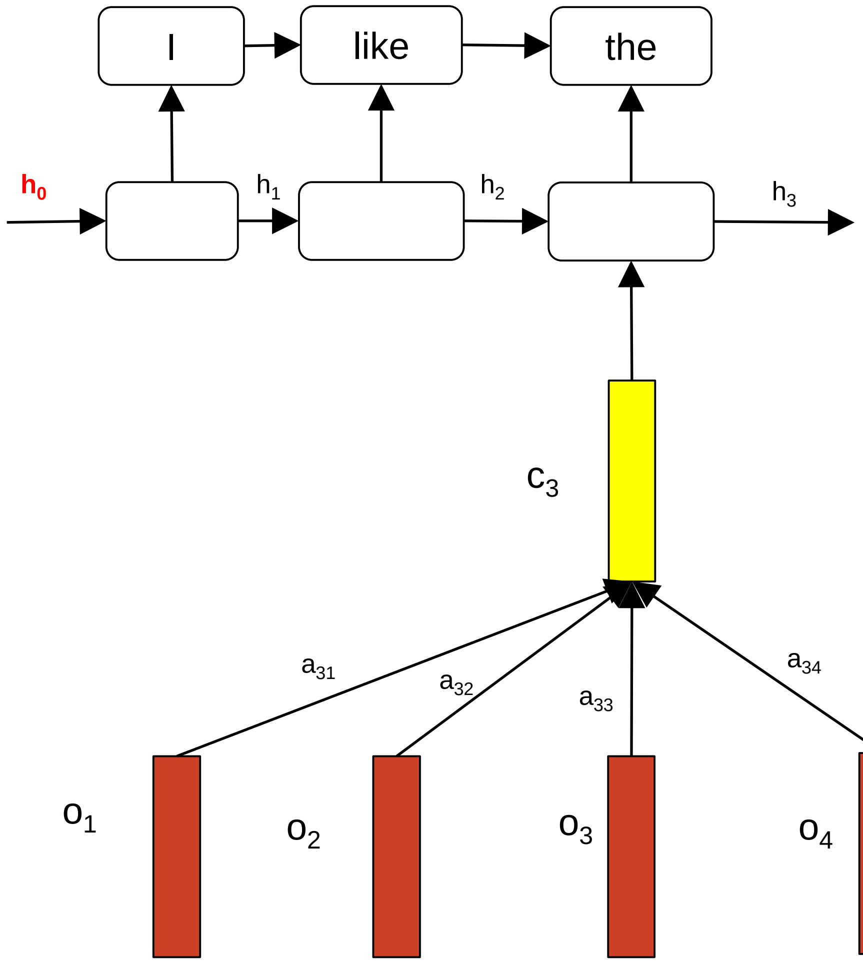
This model is called Generative model.
Here words are observed from tags as states.
This is similar to HMM.

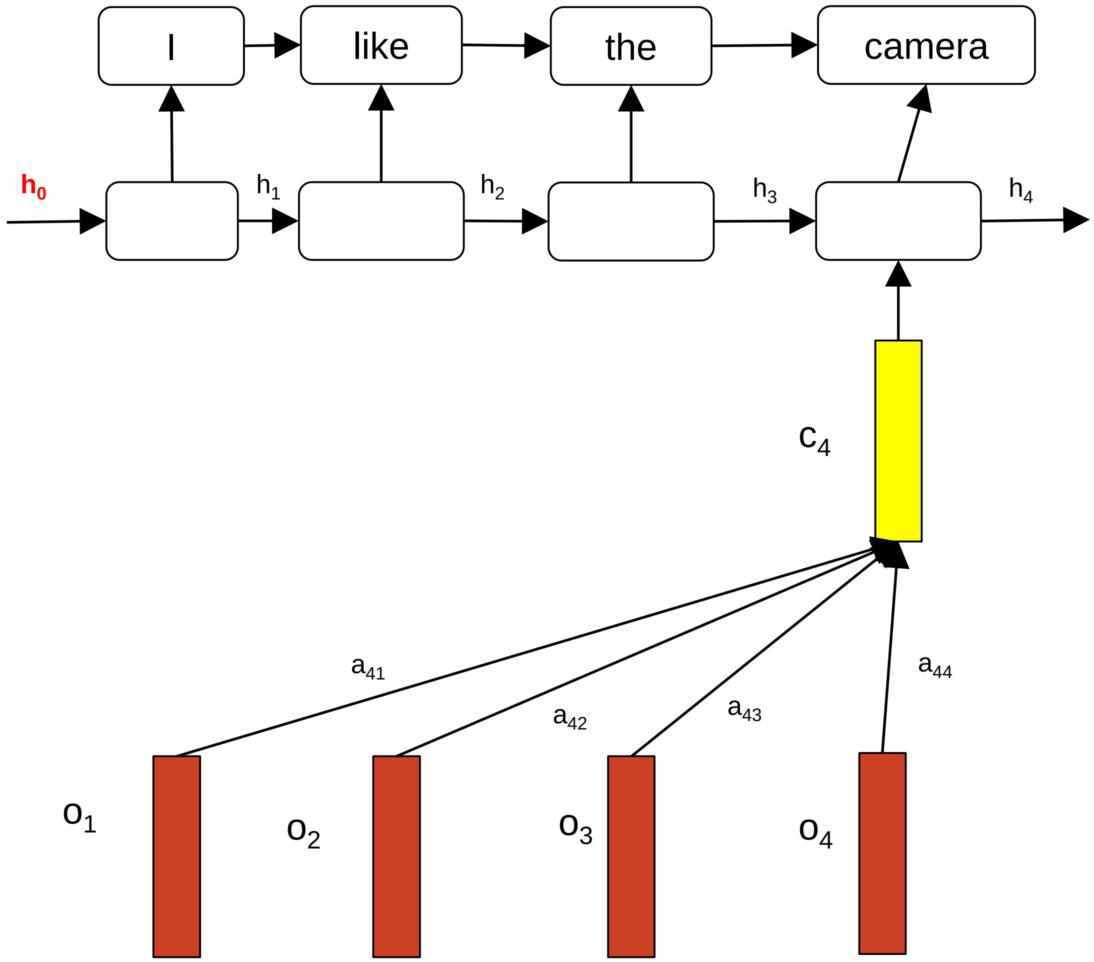
E.g. Sentiment Analysis

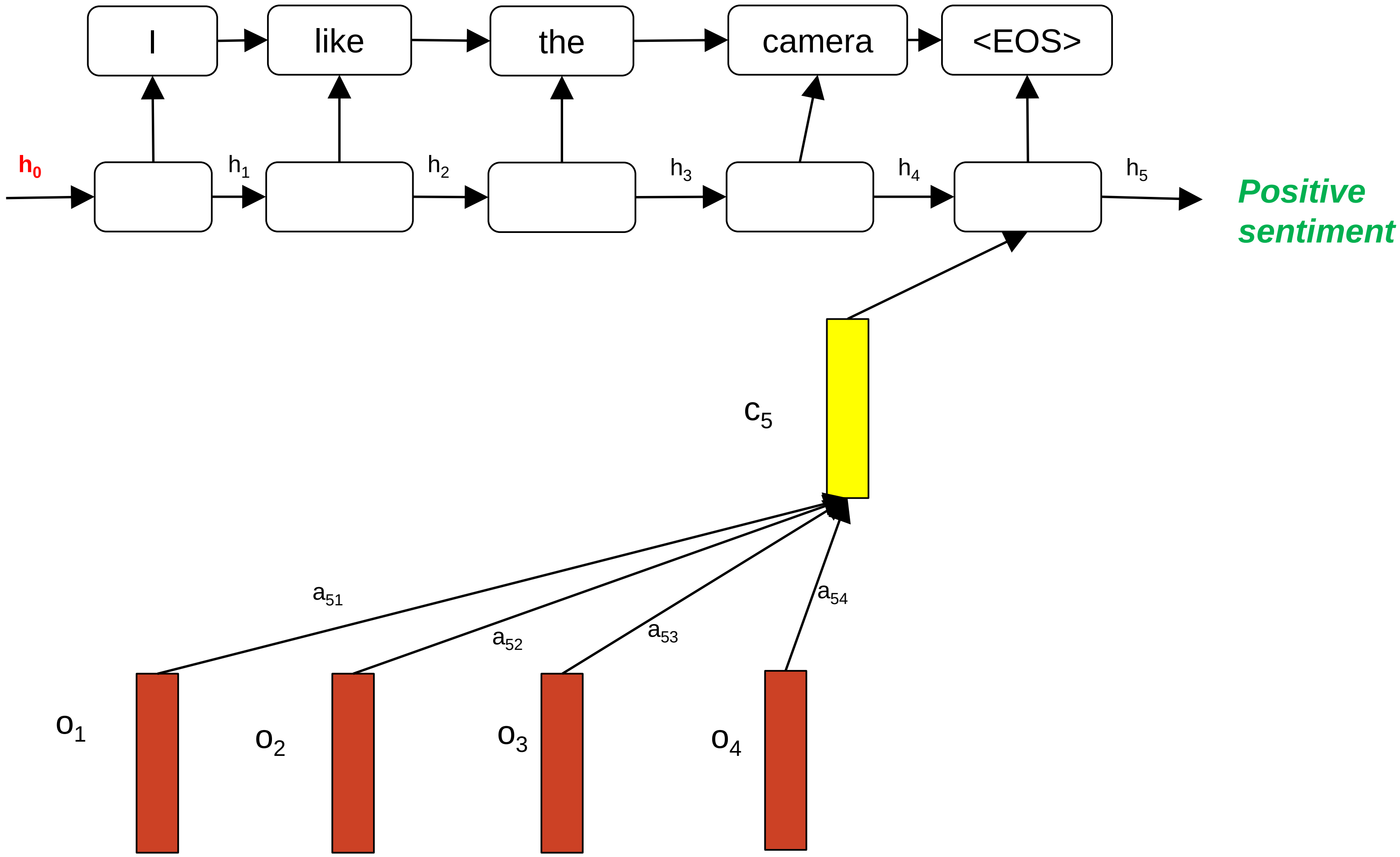
Decision on a piece of text




I
like
h0
h1
h2
c2
a21
a22
a23
a24
o1
o2
o3
o4


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
h0
h1
h2
h3
h4
c4
a41
a42
a43
a44
o1
o2
o3
o4


I
like
the
camera
<EOS>
h_0
h_1
h_2
h_3
h_4
h_5
Positive sentiment
c_5
a_{51}
a_{52}
a_{53}
a_{54}
o_1
o_2
o_3
o_4

Neural Search and Decoding

Two approaches to Search

- Rule/Knowledge based:
  - BFS, DFS, Djikstra, A, A*

- Data and ML based:
  - Viterbi, Beam

A* Algorithm

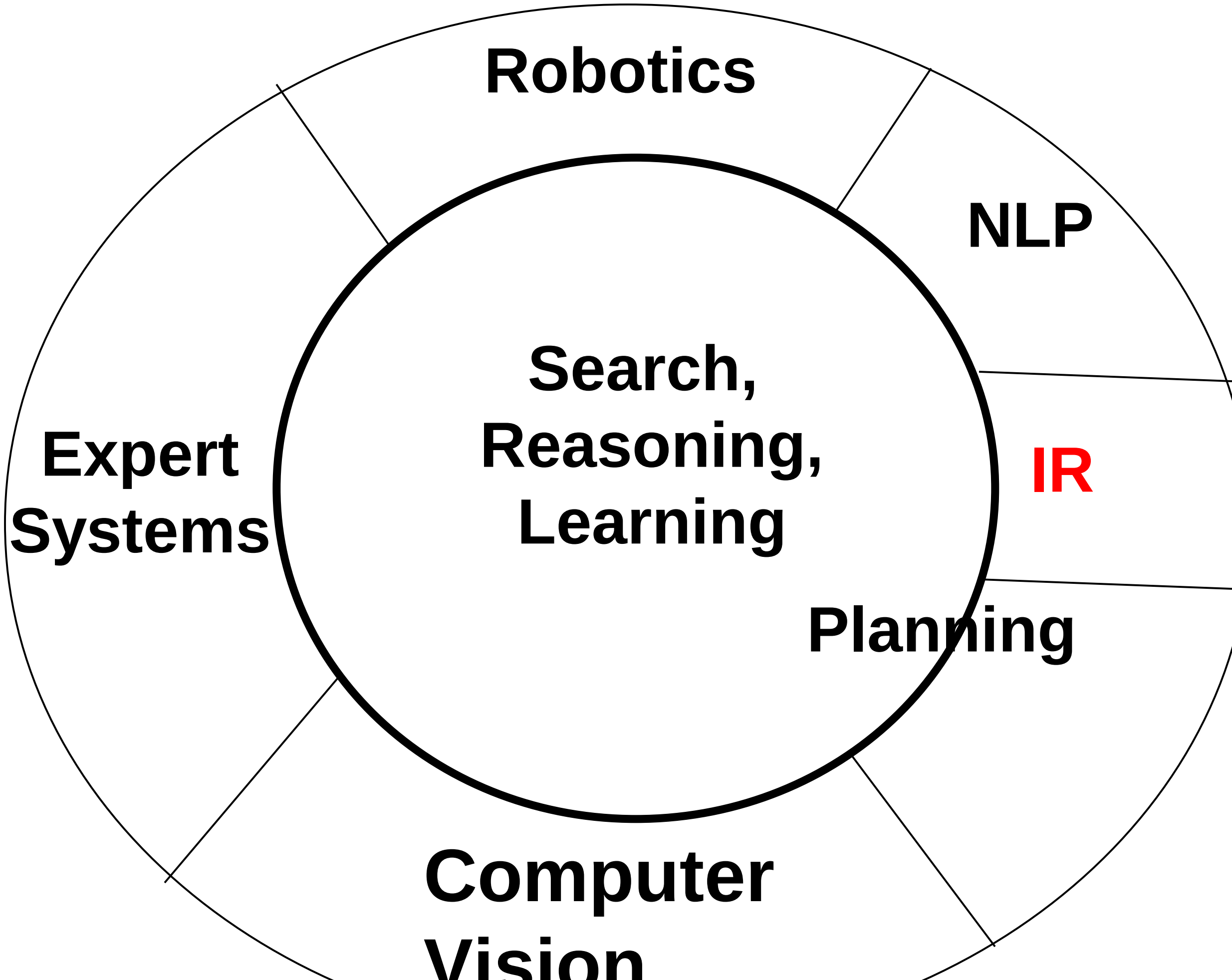
What is AI?

- The science and technology of making computers good at tasks that living beings perform effortlessly
  - E.g., understanding scenes, language
  - Driving a car
  - Identifying a person from picture even if half done
  - Diagnosing problems etc.

Modern AI is highly data driven

- “Data is the new oil”
- Cannot give a theory
- Let the data give a model
- E.g. maps.google.com

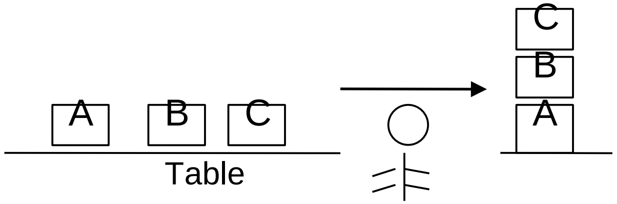
AI Perspective (post-web)



Search: Everywhere

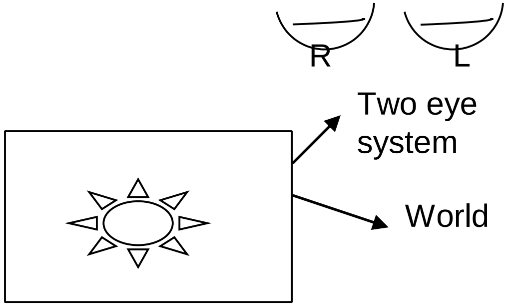
Planning

- (a) which block to *pick*, (b) which to *stack*, (c) which to *unstack*, (d) whether to *stack* a block or (e) whether to *unstack* an already stacked block. These options have to be searched in order to arrive at the right sequence of actions.



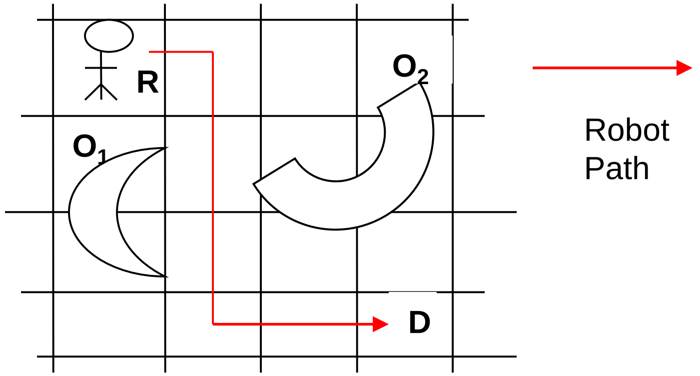
Vision

- A search needs to be carried out to find which point in the image of *L* corresponds to which point in *R*. Naively carried out, this can become an *O(n2)* process where *n* is the number of points in the retinal images.



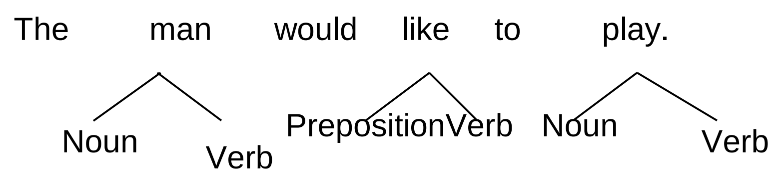
Robot Path Planning

- searching amongst the options of moving ***L***eft, ***R***ight, ***U***p or ***D***own. Additionally, each movement has an associated cost representing the relative difficulty of each movement. The search then will have to find the *optimal*, *i.e.*, the *least cost* path.



Natural Language Processing

- search among many combinations of parts of speech on the way to deciphering the meaning. This applies to every level of processing- *syntax, semantics, pragmatics* and *discourse*.



Expert Systems

Search among rules, many of which can apply to a situation:

If-conditions

the infection is primary-bacteremia
AND the site of the culture is one of the sterile sites
AND the suspected portal of entry is the gastrointestinal tract

THEN

there is suggestive evidence (0.7) that infection is bacteroid

(from MYCIN)

Search building blocks

- State Space : Graph of states (Express constraints and parameters of the problem)
- Operators : Transformations applied to the states.
- Start state : S0 (Search starts from here)
- Goal state : {G} - Search terminates here.
- Cost : Effort involved in using an operator.
- Optimal path : Least cost path

Examples

Problem 1 : 8 – puzzle

4	3	6
2	1	8
7		5

S

1	2	3
4	5	6
7	8	

G

Tile movement represented as the movement of the blank space.

Operators:

L : Blank moves left

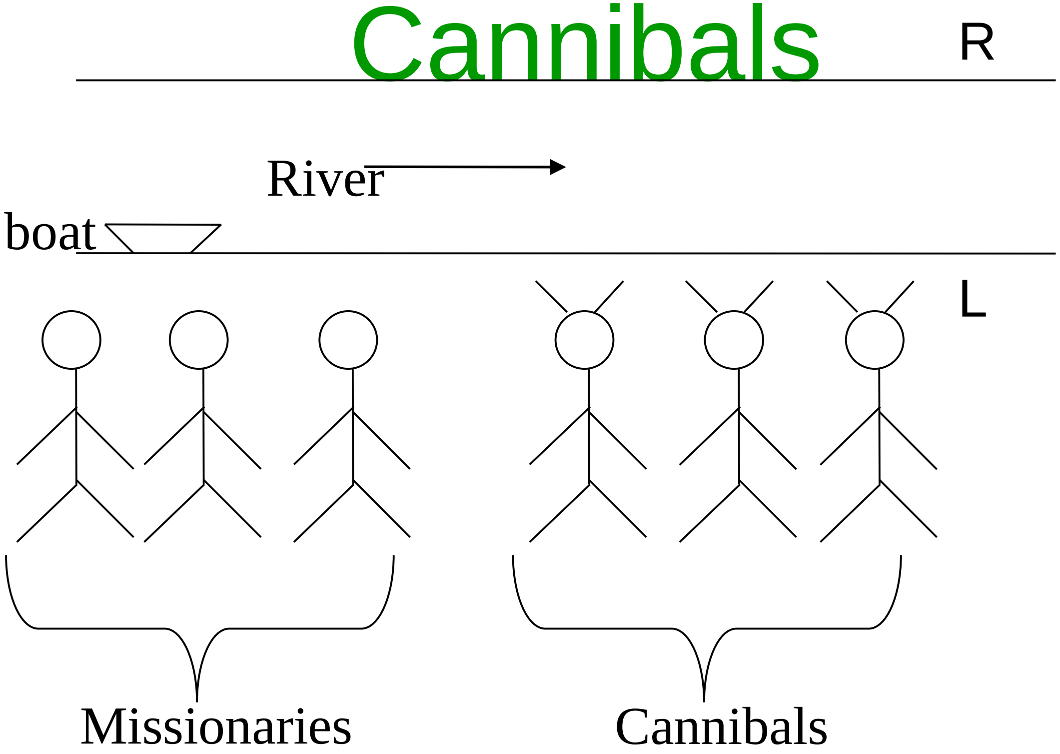
R : Blank moves right

U : Blank moves up

D : Blank moves down

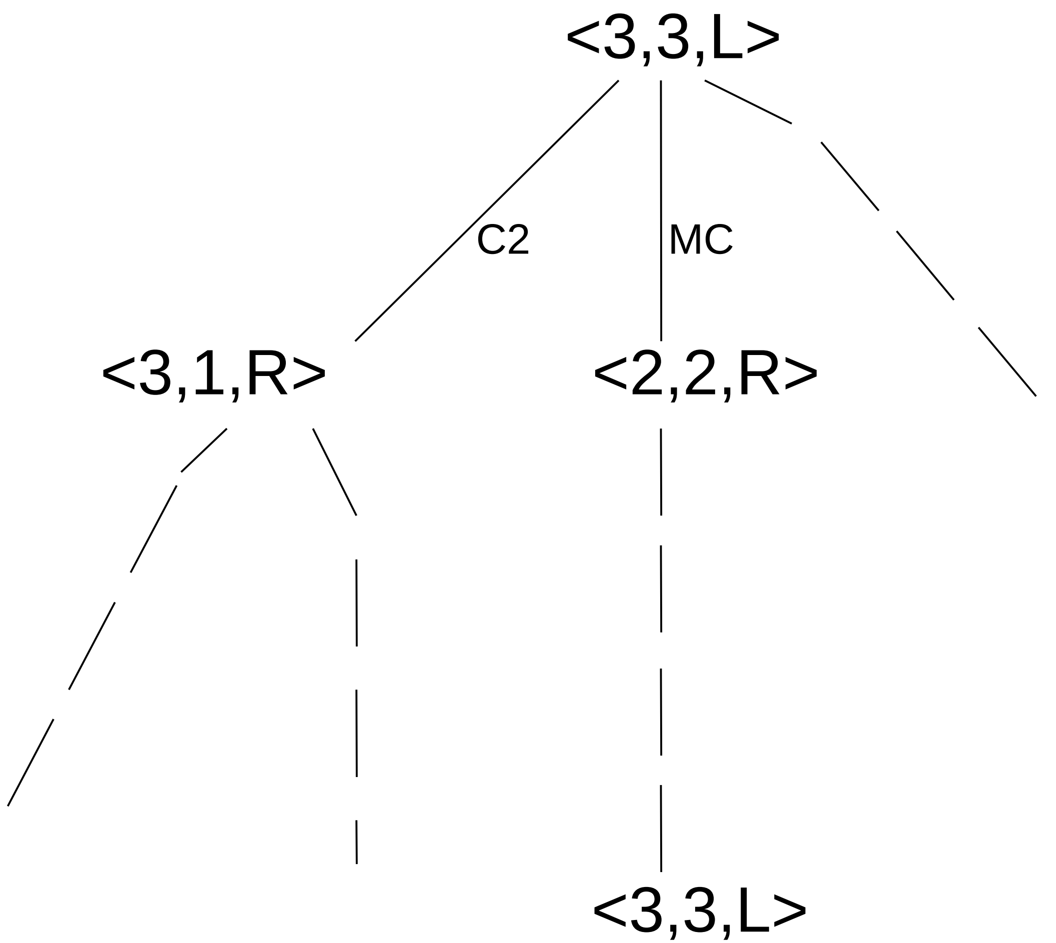
$$C(L) = C(R) = C(U) = C(D) = 1$$

Problem 2: Missionaries and Cannibals



- Constraints: (i) The boat can carry at most 2 people, (ii) On no bank should the cannibals outnumber the missionaries

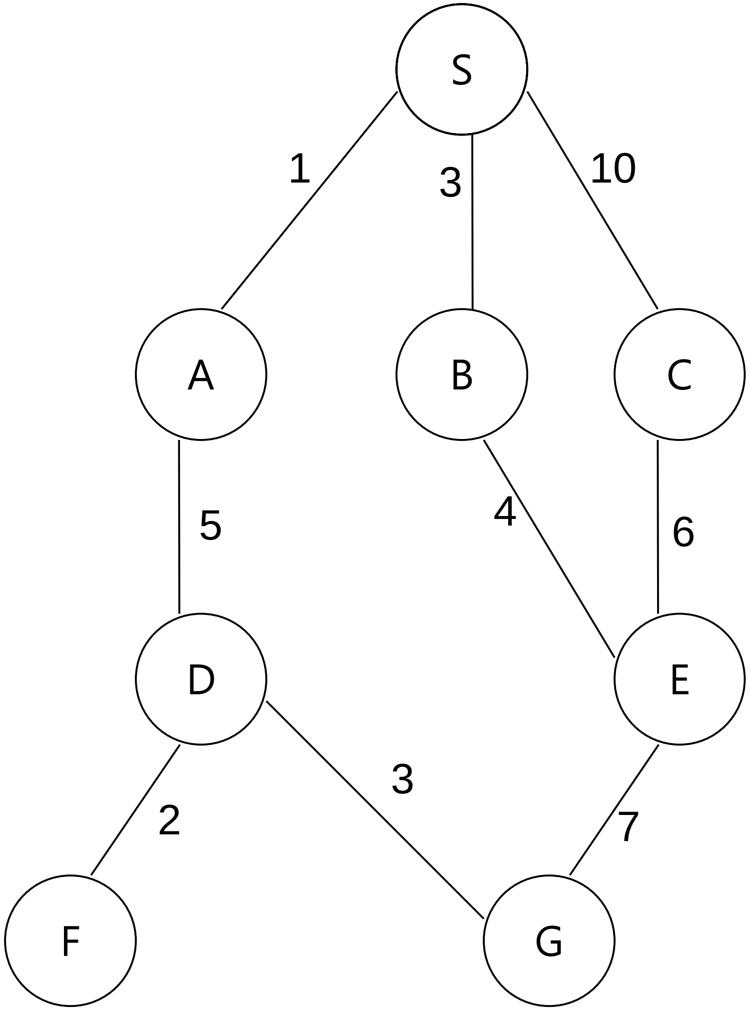
- State : <#M, #C, P>
- #M = Number of missionaries on bank L
- #C = Number of cannibals on bank L
- P = Position of the boat, S0 = <3, 3, L>, G = < 0, 0, R >
- Operations
- M2 = Two missionaries take boat, M1 = One missionary takes boat
- C2 = Two cannibals take boat, C1 = One cannibal takes boat
- MC = One missionary and one cannibal takes boat



Partial search tree

Algorithmics of Search

General Graph search Algorithm



Graph G = (V,E)

1) Open List : S $^{(Ø, 0)}$

Closed list : Ø

2) OL : A$^{(S,1)}$, B$^{(S,3)}$, C$^{(S,10)}$

CL : S

3) OL : B$^{(S,3)}$, C$^{(S,10)}$, D$^{(A,6)}$

CL : S, A

4) OL : C$^{(S,10)}$, D$^{(A,6)}$, E$^{(B,7)}$

CL: S, A, B

5) OL : D$^{(A,6)}$, E$^{(B,7)}$

CL : S, A, B , C

6) OL : E$^{(B,7)}$, F$^{(D,8)}$, G$^{(D, 9)}$

CL : S, A, B, C, D

7) OL : F$^{(D,8)}$, G$^{(D,9)}$

CL : S, A, B, C, D, E

8) OL : G$^{(D,9)}$

CL : S, A, B, C, D, E, F

9) OL : Ø

CL : S, A, B, C, D, E,
F, G

Steps of GGS
(*principles of AI, Nilsson,)*

- 1. Create a search graph *G*, consisting solely of the start node *S*; put *S* on a list called *OPEN.*
- *2.* Create a list called *CLOSED* that is initially empty.
- 3. Loop: if *OPEN* is empty, exit with **FAILURE**.
- 4. Select the first node on *OPEN*, **remove from *OPEN* and put on *CLOSED***, call this node *n*.
- 5. if *n* is the goal node, exit with **SUCCESS** with the solution obtained by tracing a path along the pointers from *n* to *s* in *G*. (pointers are established in step 7).
- 6. **Expand** node *n*, generating the set *M* of its successors that are not ancestors of *n*. Install these memes of *M* as successors of *n* in *G*.

GGS steps (contd.)

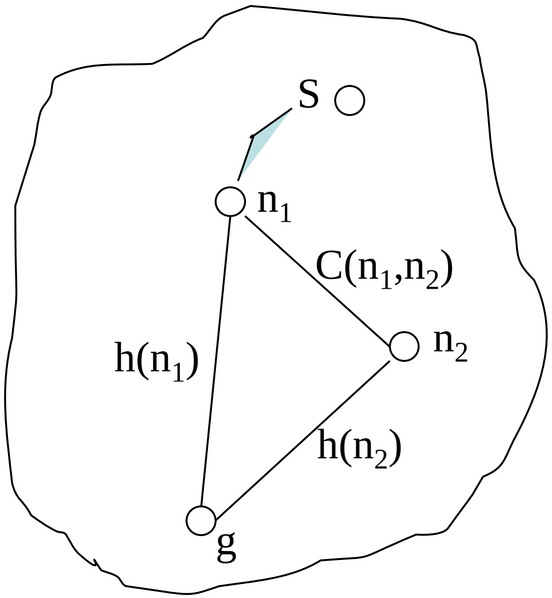
- 7. **Maintain the least cost path and node in OPEN**: to Establish a pointer to *n* from those members of *M* that were not already in *G* (*i.e.*, not already on either *OPEN* or *CLOSED*). Add these members of *M* to *OPEN*. For each member of *M* that was already on *OPEN* or *CLOSED*, decide whether or not to redirect its pointer to *n*. For each member of M already on *CLOSED*, decide for each of its descendants in *G* whether or not to redirect its pointer.
- 8. Reorder the list *OPEN* using some strategy.
- 9. Go *LOOP.*

GGS is a general umbrella

OL is a queue (BFS)

OL is stack (DFS)

OL is accessed by using a functions $f = g + h$ (Algorithm A)



$$h(n_1) \leq C(n_1, n_2) + h(n_2)$$

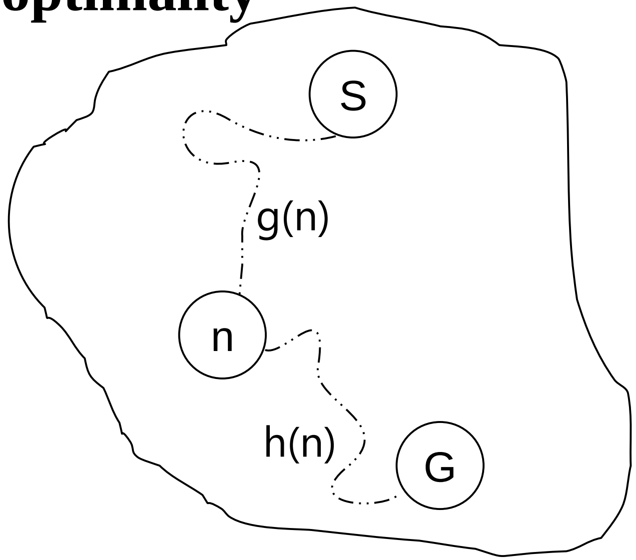
Algorithm A

- A function f is maintained with each node $f(N) = g(N) + h(N)$, N is the node in the open list
- Node chosen for expansion is the one with least f value
- BFS: $h = 0$, g = number of edges in the path to S
- DFS: $h = 0$, g =(1/no. of edges)
- Djikstra: g=path cost from S to N
- A*: h <= h*, h*=actual path cost from N to G the goal

Algorithm A*

- One of the most important advances in AI
- $g(n)$ = least cost path to n from S found so far
- $h(n) <= h^*(n)$ where $h^*(n)$ is the actual cost of optimal path to G(node to be found) from n

"**Optimism leads to optimality**"

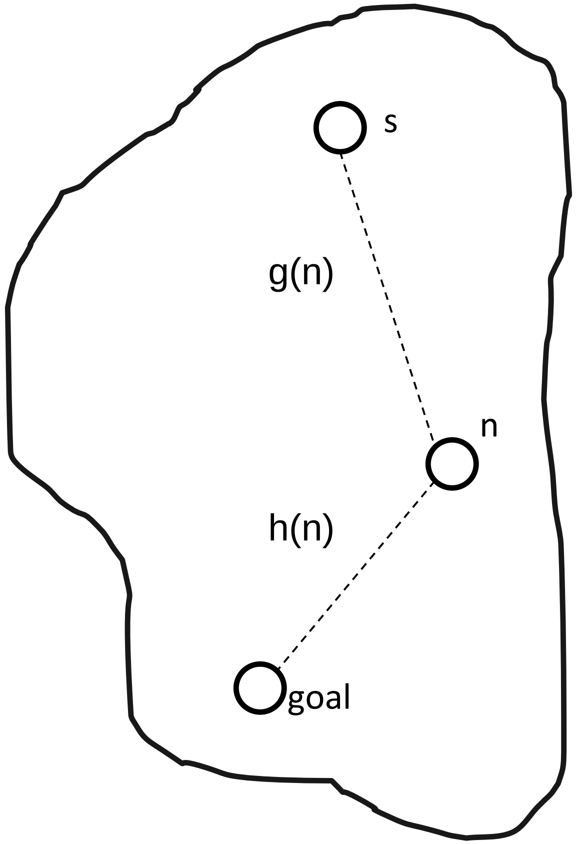


A* Algorithm – Definition and Properties

- $f(n) = g(n) + h(n)$
- The node with the least value of *f* is chosen from the *OL*.
- $f^*(n) = g^*(n) + h^*(n)$, where,
  $g^*(n)$ = actual cost of the optimal path (s, n)
  $h^*(n)$ = actual cost of optimal path (n, g)

- $g(n) \geq g^*(n)$

- By definition, $h(n) \leq h^*(n)$



State space graph G

8-puzzle: heuristics

Example: 8 puzzle

2	1	4
7	8	3
5	6	

s

1	6	7
4	3	2
5		8

n

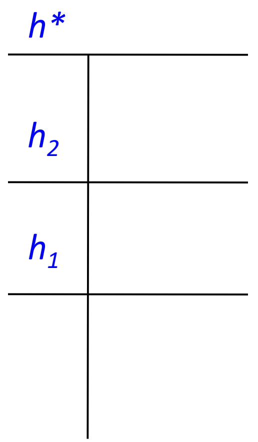
1	2	3
4	5	6
7	8	

g

$h^*(n)$ = actual no. of moves to transform n to g

1. $h_1(n)$ = no. of tiles displaced from their destined position.
2. $h_2(n)$ = sum of Manhattan distances of tiles from their destined position.

$$h_1(n) \leq h^*(n) \text{ and } h_1(n) \leq h^*(n)$$



Comparison

A* critical points

- **Goal**
  1. Do we know the goal?
  2. Is the distance to the goal known?
  3. Is there a path (known?) to the goal?

A* critical points

- **About the path**

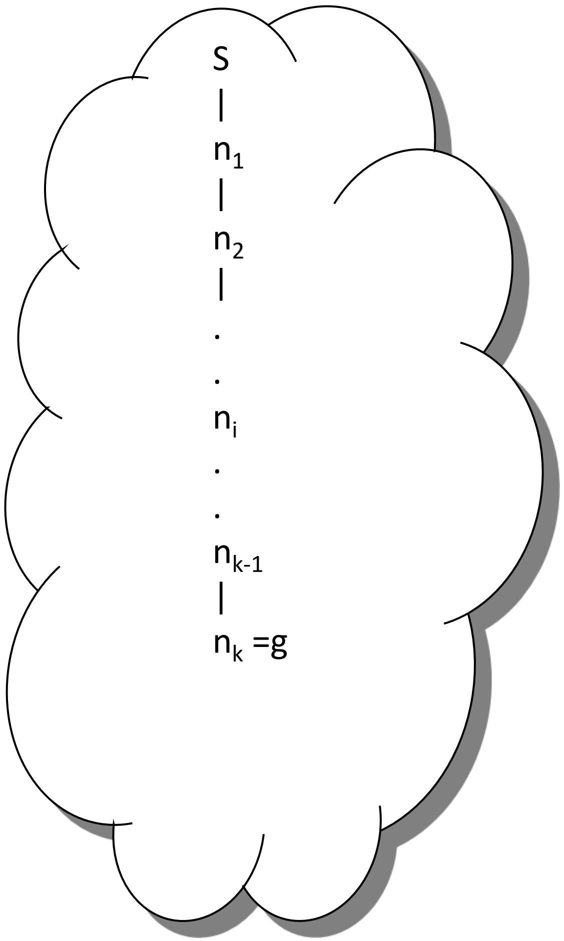
  Any time before A* terminates there exists on the OL, a node from the optimal path all whose ancestors in the optimal path are in the CL.

This means,

There exists in the OL always a node 'n' s.t.

$$g(n) = g^*(n)$$

Key point about A* search



<u>Statement</u>:

Let S $-n_1-n_2-n_3\ldots n_i\ldots -n_{k-1}-n_k$(=G) be an optimal path.

At any time during the search:

1. There is a node n_i from the optimal path in the OL
2. For n_i all its ancestors S,$n_1,n_2,\ldots,n_{i-1}$ are in CL
3. $g(n_i) = g^*(n_i)$

Proof of the statement

Proof by induction on iteration no. j

Basis : j = 0, S is on the OL, S satisfies the statement

Hypothesis : Let the statement be true for j = p (p^{th} iteration)

Let n_i be the node satisfying the statement

Proof (continued)

Induction : Iteration no. $j = p+1$

Case 1 : n_i is expanded and moved to the closed list

Then, n_{i+1} from the optimal path comes to the OL

Node n_{i+1} satisfies the statement

(note: if n_{i+1} is in CL, then n_{i+2} satisfies the property)

Case 2 : Node $x \neq n_i$ is expanded

Here, n_i satisfies the statement

A* Algorithm- Properties

- **Admissibility**: An algorithm is called admissible if it always terminates and terminates in optimal path
- **Theorem**: A* is admissible.
- **Lemma**: Any time before A* terminates there exists on *OL* a node *n* such that $f(n) <= f^*(s)$
- **Observation**: For optimal path $s \longrightarrow n_1 \longrightarrow n_2 \longrightarrow \ldots \longrightarrow g$,

  1. $h^*(g) = 0$, $g^*(s)=0$ and
  2. $f^*(s) = f^*(n_1) = f^*(n_2) = f^*(n_3) \ldots = f^*(g)$

A* Properties (contd.)

$f^*(n_i) = f^*(s)$, $n_i \neq s$ and $n_i \neq g$

Following set of equations show the above equality:

$$f^*(n_i) = g^*(n_i) + h^*(n_i)$$

$$f^*(n_{i+1}) = g^*(n_{i+1}) + h^*(n_{i+1})$$

$$g^*(n_{i+1}) = g^*(n_i) + c(n_i , n_{i+1})$$

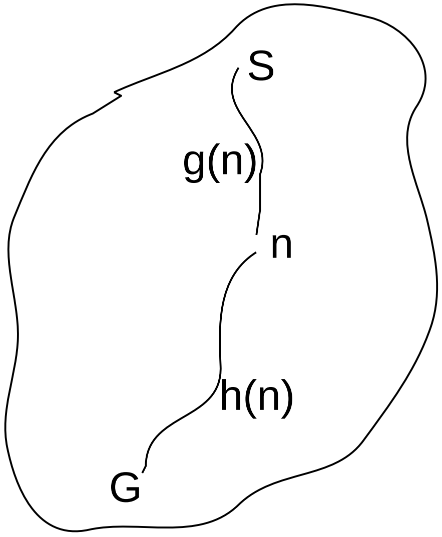
$$h^*(n_{i+1}) = h^*(n_i) - c(n_i , n_{i+1})$$

Above equations hold since the path is optimal.

Admissibility of A*

A* always terminates finding an optimal path to the goal if such a path exists.

<u>Intuition</u>



(1) In the open list there always exists a node *n* such that $f(n) <= f^*(S)$.

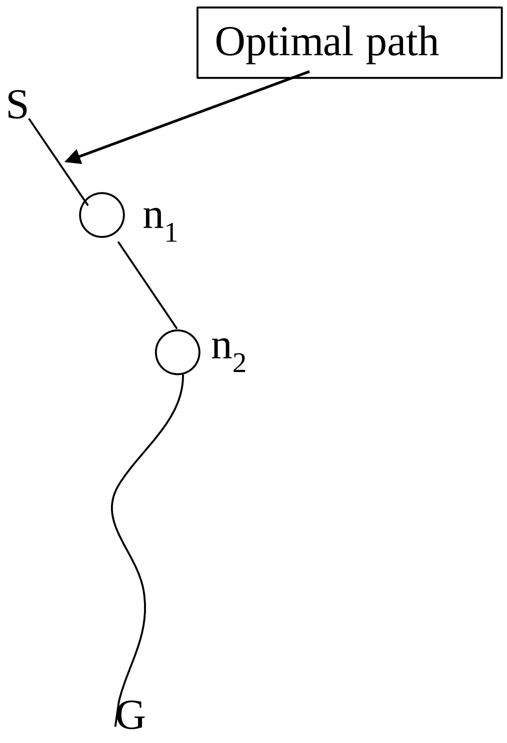
(2) If A* does not terminate, the *f* value of the nodes expanded become unbounded.

1) and 2) are together inconsistent

Hence A* must terminate

Lemma

Any time before A* terminates there exists in the open list a node n' such that $f(n') <= f^*(S)$



For any node n_i on optimal path,

$f(n_i) = g(n_i) + h(n_i)$

$\quad <= g^*(n_i) + h^*(n_i)$

Also $f^*(n_i) = f^*(S)$

Let n' be the first node in the optimal path that is in OL. Since all parents of n' in the optimal have gone to CL,

$g(n') = g^*(n')$ *and* $h(n') <= h^*(n')$

$=> f(n') <= f^*(S)$

If A* does not terminate

Let e be the least cost of all arcs in the search graph.

Then $g(n) >= e.l(n)$ where $l(n)$ = # of arcs in the path from S to n found so far. If A* does not terminate, $g(n)$ and hence $f(n) = g(n) + h(n)$ $[h(n) >= 0]$ will become unbounded.

This is not consistent with the lemma. So A* has to terminate.

2nd part of admissibility of A*

The path formed by A* is optimal when it has terminated

Proof

Suppose the path formed is not optimal

Let G be expanded in a non-optimal path.

At the point of expansion of G,

$f(G) = g(G) + h(G)$

$= g(G) + 0$

$> g^*(G) = g^*(S) + h^*(S)$

$= f^*(S)$ [$f^*(S)$ = cost of optimal path]

This is a contradiction

So path should be optimal

Key Points on Admissibility

- 1\. A* algorithm halts
- *2.* A* algorithm finds optimal path
- 3\. If $f(n) < f^*(S)$ then node n has to be expanded before termination
- 4\. If A* does not expand a node n before termination then $f(n) >= f^*(S)$

Exercise-1

Prove that if the distance of every node from the goal node is “known”, then no “search:” is necessary

Ans:

- For every node *n*, $h(n)=h^*(n)$. The algo is A*.
- Lemma proved: any time before A* terminates, there is a node *m* in the OL that has $f(m) <= f^*(S)$, *S*= start node (*m* is the node on the optimal path all whose ancestors in the optimal path are in the closed list).
- For *m*, $g(m)=g^*(m)$ and hence $f(m)=f^*(S)$.
- Thus at every step, the node with $f=f^*$ will be picked up, and the journey to the goal will be completely directed and definite, with no “search” at all.
- Note: when $h=h^*$, *f* value of any node on the OL can never be less than $f^*(S)$.

Exercise-2

If the *h* value for every node over-estimates the *h** value of the corresponding node by a constant, then the path found need not be costlier than the optimal path by that constant. Prove this.

Ans:

- Under the condition of the problem, $h(n) <= h^*(n) + c.$
- Now, any time before the algo terminates, there exists on the OL a node *m* such that $f(m) <= f^*(S)+c.$
- The reason is as follows: let *m* be the node on the optimal path all whose ancestors are in the CL (there *has to be* such a node).
- Now, $f(m)= g(m)+h(m)=g^*(m)+h(m) <= g^*(m)+h^*(m)+c = f^*(S)+c$
- When the goal *G* is picked up for expansion, it must be the case that
- $f(G)<= f^*(S)+c=f^*(G)+c$
- *i.e.,* $g(G)<= g^*(G)+c,$ since $h(G)=h^*(G)=0$.

A list of AI Search Algorithms

- A*
  - AO*
  - IDA* (Iterative Deepening)
- Minimax Search on Game Trees
- Viterbi Search on Probabilistic FSA
- Hill Climbing
- Simulated Annealing
- Gradient Descent
- Stack Based Search
- Genetic Algorithms
- Memetic Algorithms

Viterbi Decoding

Illustration with POS tagging

Sentence: "*People Dance*"

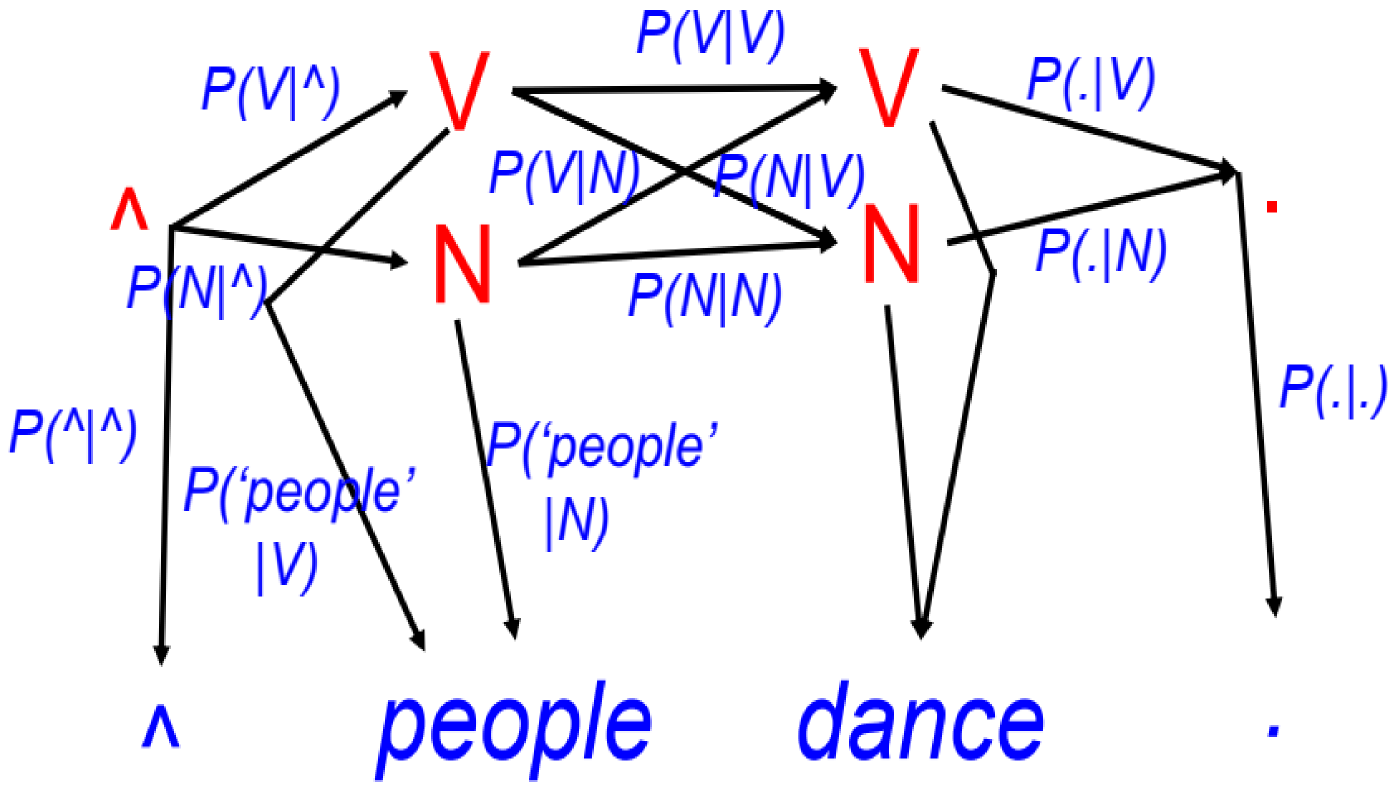
- '*people*' and '*dance*' can both be both nouns and verbs, as in
  - "*old_JJ people_NNS*" ('*people*' as noun)
  - "*township_NN peopled_VBN with soldiers_NNS*" ('*people*' as verb)
- as well as
  - "*rules_NNS of_IN classical_JJ dance_NN*" ('*dance*' as noun)
  - "*will*_VAUX *dance_VB well_RB*" ('*dance*' as verb)

Possible Tags: “^ *people dance .*”

- for simplicity we take single letter tags- *N: noun, V: verb*:
  - ^ *N N .*
  - ^ *N V .*
  - ^ *V N .*
  - ^ *V V .*
- We know that out of these, the second option ^ *N V.* is the correct one. How do we get this sequence?

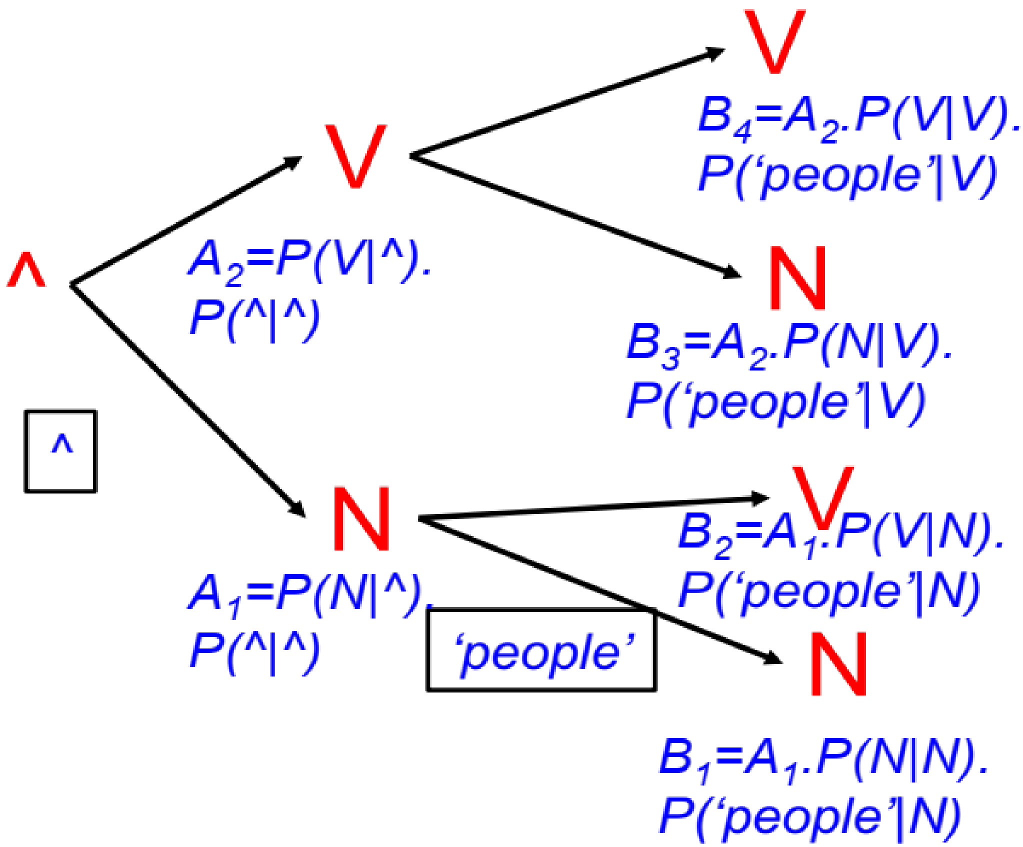
Step-1: Trellis

Columns of tags on each input word with transition arcs going from tags (states) to tags in consecutive columns and output arcs going from tags to words (observations)



Aim: select the highest probability path

From 4 possibilities; *A*s and *B*s are accumulated probabilities



RNN vs. HMM

- RNN is an infinite memory machine (ideally) and is more general than a k-order HMM
- HMM combines lexical and transition probabilities through the product operation (Markov independence assumption) while the Softmax operation in the RNN encompasses both these probabilities

Some numerical values: hypothetical but not unrealistic

- Calculations:
- When it comes to the start of the sentence, most sentences start with a noun. So lets have

  $P(N|\text{^})=0.8$, $P(V|\text{^})=0.2$ and of course $P(\text{‘^’}|\text{^})=1.0$
- Then

$$A_1=0.8,\ A_2=0.2$$

Encounter “people”: more probabilities (1/2)

- Transition from *N* to *N* is less common than to *V*.
- Transition from *V* to *V*- as in auxiliary verb to main verb- is quite common (e.g., *is going*).
- *V* to *N* too is common- as in case of a nominal object following the verb *(going home).*
- Following plausible transition probabilities:
  - $P(N|N)=0.2$, $P(V|N)=0.8$, $P(V|V)=0.4$, $P(N|V)=0.6$
- We also need lexical probabilities. ‘people’ appearing as verb is much less common than its appearing as noun. So let us have

Encounter “people”: more probabilities (2/2)

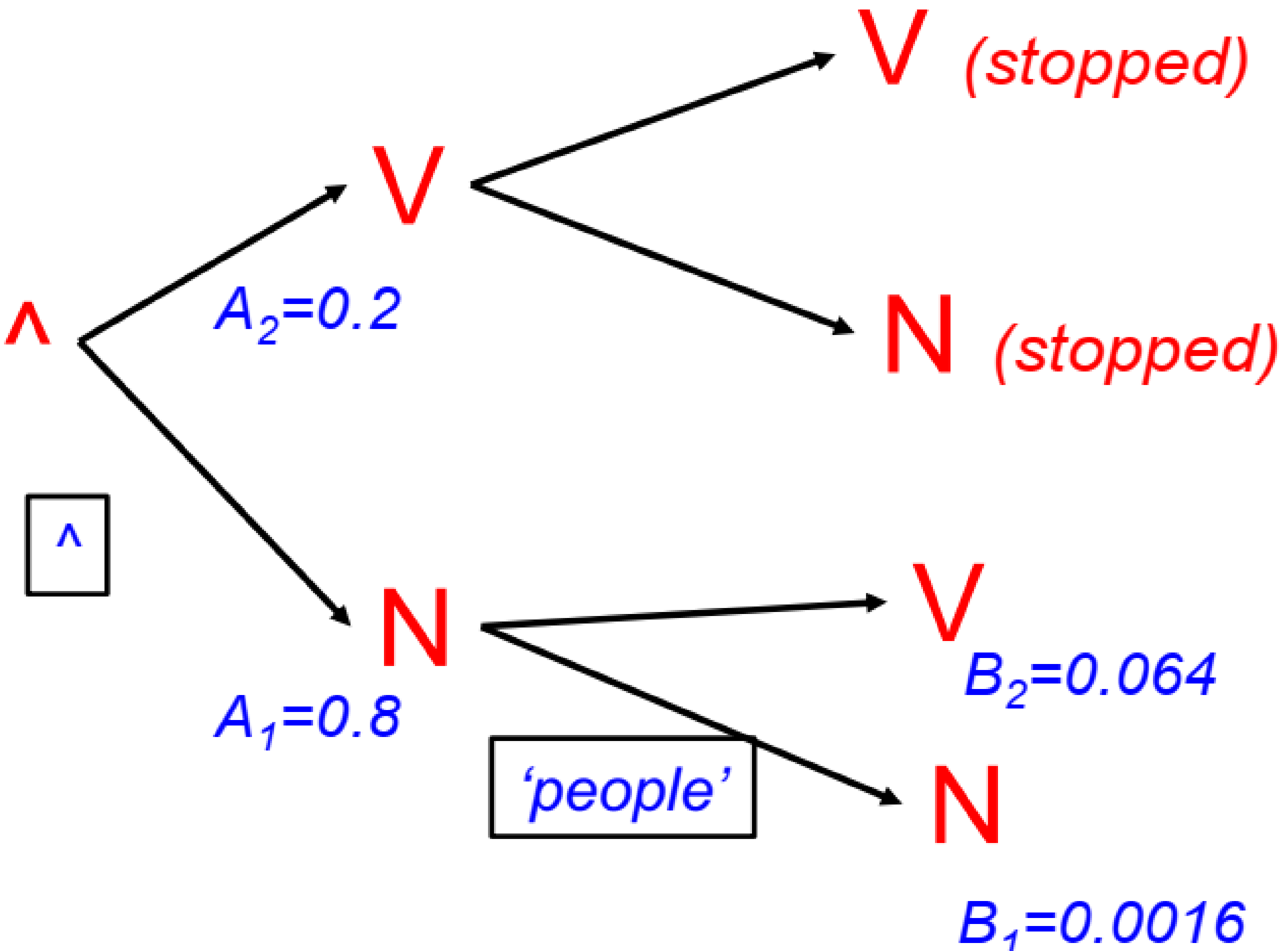
- We also need lexical probabilities. ‘people’ appearing as verb is much less common than its appearing as noun. So let us have
  - *P(‘people’|N)=0.01, P(‘people’|V)=0.001*
- Note: *N N*: golf club, cricket bat, town people- ambiguity “The town people visited was deserted”/”town people will not be able to live here”
- V V combination: Hindi- *has padaa* (laughed suddenly), Bengali- *chole gelo* (went away)

Calculate *B*s

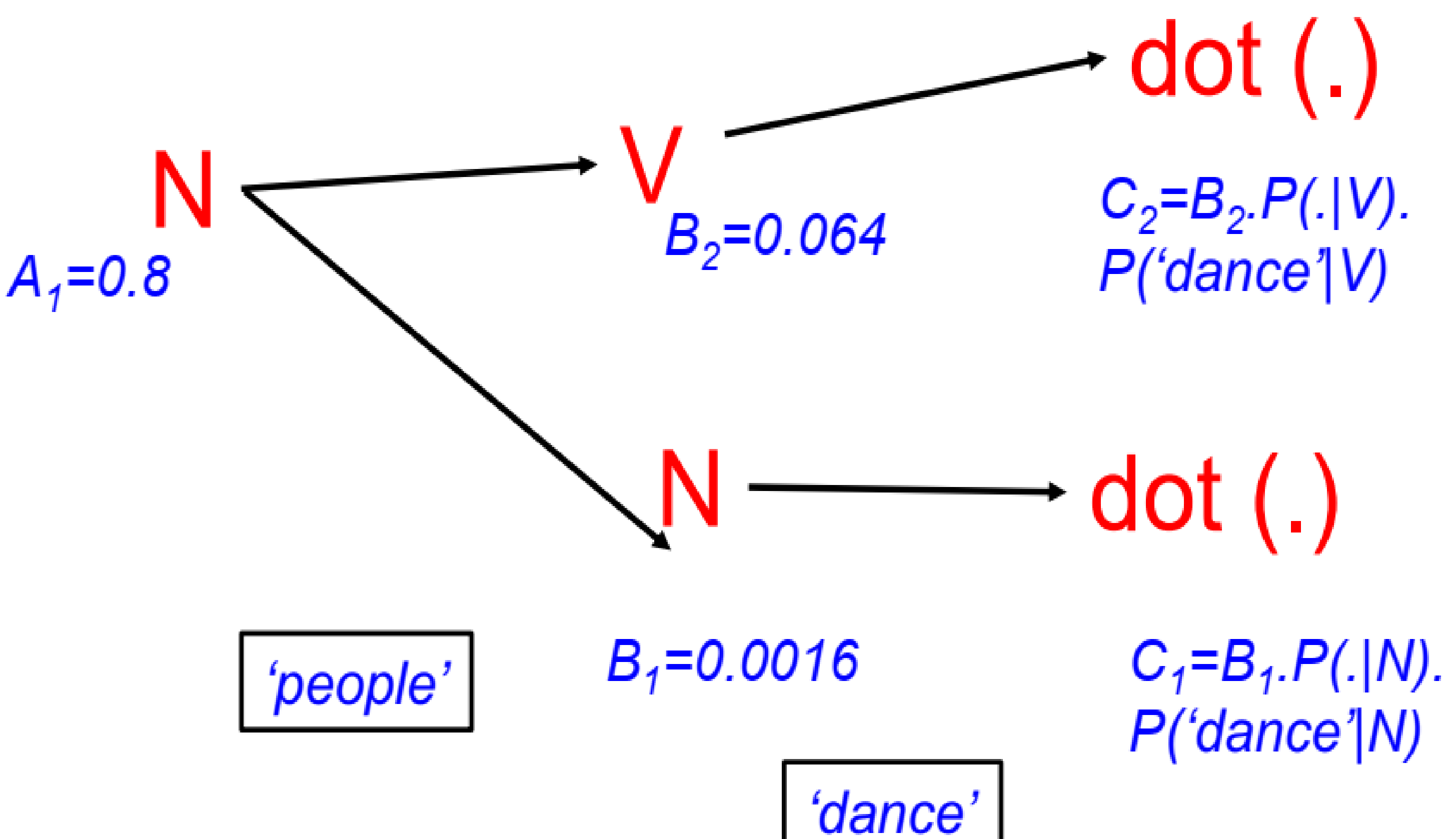
- $B_1=0.8.0.2.0.01=0.0016$ *(approx.)*
- $B_2=0.8.0.8.0.01=0.064$ *(approx.)*
- $B_3=0.2.0.6.0.001=0.00012$
- $B_4=0.2.0.4.0.001=0.00008$

Reduced Viterbi Configuration

- Heart of Decoding → linear time



Next word: '*dance*'



More probabilities needed

- We can give equal probabilities to sentences ending in noun and verb. Also, ‘dance’ as verb is more common than noun.

$$P(.|N)=0.5=P(.|V)$$

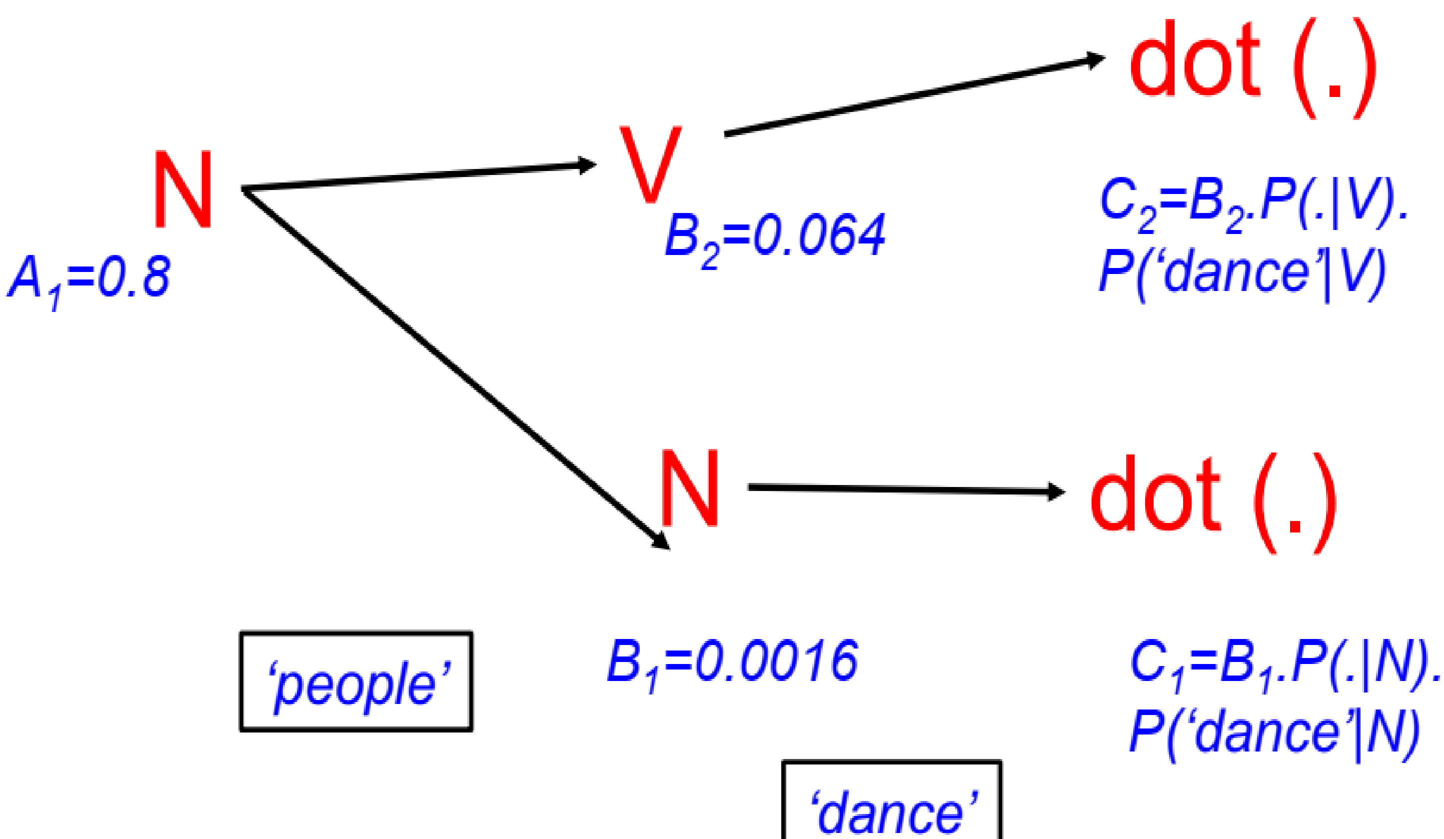
$$P(\text{‘}dance\text{’}|N)=0.001$$

$$P(\text{‘}dance\text{’}|V)=0.01$$

Best Path: ^ N V .

C_1=0.0016.0.5.0.001=0.0000008

C_2=0.064.0.5.0.01=0.00032



Beam Search Based Decoding

Motivation

- HMM based POS tagging cannot handle “free word order” and “agglutination” well
- If *adjective after noun* is equally likely as *adjective before noun*, the transition probability is no better than uniform probability which has high entropy and is uninformative.
- When the words are long strings of many morphemes, POS tagging w/o morph features is highly inaccuarte.

Modelling in Discriminative POS Tagging

- *T** is the best possible tag sequence
- Summation dropped, because given *W* and feature engineering, *F* is unique; also *P(F|T)=1*
- The final independence assumption is that the tag at any position *i* depends only on the feature vector at that position

$$T^* = \arg\max_T P(T \mid W)$$

$$P(T \mid W) = \sum_F P(T, F \mid W) = P(T, F \mid W)$$

$$= P(F \mid W).P(T \mid F, W)$$

$$= 1.P(T \mid F) = P(T \mid F)$$

$$P(T \mid F) = \prod_{i=0}^{n+1} [P(t_i \mid F_i)]$$

Feature Engineering

- Running example: ***^ brown foxes jumped over the fence .***
- *A. Word-based features*

  f_{21} – dictionary index of the current word (‘foxes’): integer

  f_{22} – -do- of the previous word (‘brown’): integer

  f_{23} – -do- of the next word (‘jumped’): integer

B. Part of Speech (POS) tag-based feature

  f_{24} – index of POS of previous word (here JJ): integer

Feature engineering cntd.

- ***^ brown foxes jumped over the fence .***

- *C. Morphology-based features*
  - f_{25}– does the current word (‘foxes’) have a noun suffix, like ‘s’, ‘es’, ‘ies’, etc.: 1/0- here the value is
  - f_{26}– does the current word (‘foxes’) have a verbal suffix, like ‘d’, ‘ed’, ‘t’, etc.: 1/0- 0
  - f_{27} and f_{28} for ‘brown’ like for ‘foxes
  - f_{29} and $f_{2,10}$ for ‘jumped’ like for ‘foxes; here $f_{2,10}$ is 1 (jumped has ‘ed’ as suffix)

An Aside: word vectors

- These features are opaquely represented in word vectors created from huge corpora
- Word vectors are vectors of numbers representing words
- It is not possible to tell which component in the word vector does what

Modelling Equations

$\boldsymbol{W}$: ^ $w_0\, w_1\, w_2 \ldots w_{n-2}\, w_{n-1}\, w_n$. $\boldsymbol{T}$: ^ $t_0\, t_1\, t_2 \ldots t_{n-2}\, t_{n-1}\, t_n$.

$$P(T) = \prod_{i=0}^{n+1} [P(t_i \mid F_i)]$$

$$P(t_i = t \mid F_i) = \frac{e^{\sum_{j=1.k} \lambda_j f_{ij}}}{\sum_{t' \in S} e^{\sum_{j=1.k} \lambda_j f_{ij}(t')}}$$

Maximum Entropy Markov Model (MEMM)

S: set of tags.

The sequence probability of a tag sequence *T* is the product of $P(t_i|F_i)$, *i* varying over the positions.

Beam Search Based Decoding

- **^ *The brown foxes jumped .***
- Let us assume the following tags for the purpose of the discussion:
  - D- determiner like ‘the’
  - A- adjective like ‘brown’
  - N- noun like ‘foxes’, ‘fence’
  - V- verb like ‘jumped’

- Let the decoder start at the state ‘^’ which denotes start of the sentence.

Step-1

- **^ *The brown foxes jumped .***
- The word '*the*' is encountered. First there are 4 next states possible corresponding to 4 tags, giving rise to 4 possible paths:
- ^ *D* -P_1
- ^ *A* -P_2
- ^ *N* -P_3
- ^ *V* -P_4

Commit to Beam Width

- Beam width is an integer which denotes how many of the possibilities should be kept *open*.
- Let the beam width be 2.
  - **This means that out of all the paths obtained so far we retain only the top 2 in terms of their probability scores.**
- We will assume that the actual linguistically viable sub-sequence appears amongst the top two choices.
  - ‘The’ is a determiner and we get the two highest probability paths for “^ The” as P_1 and P_3.

Step-2

- **^ *The* *brown* *foxes jumped .***
- ‘*brown*’ is the next word. P_1 and P_3 are extended as
- ^ *D D* -P_{11}
- ^ *D A* -P_{12}
- ^ *D N* -P_{13}
- ^ *D V* -P_{14}
- ^ *N D* -P_{31}
- ^ *N A* -P_{32}
- ^ *N N* -P_{33}
- ^ *N V* -P_{34}

Retain two paths

- Keep two possibilities corresponding to correct/almost-correct sub-sequences. '*brown*' is an adjective, but can be noun too (e.g., "*the brown of his eyes*").

^ *D A* -P_{12}

^ *D N* -P_{13}

Step-3

- ***^ The brown foxes jumped .***
- Can be both noun and verb (verb: "*he was foxed by their guile*").
- From P_{12} and P_{13}, we will get 8 paths, but retain only two, as per the beam width.
- We assume only the paths coming from P_{12} survive with 'A' and 'N' extending the paths:

^ *D A A* -P_{122} (this is a wrong path!)

^ *D A N* -P_{123}

Step-4

- ***^ The brown foxes jumped .***

- Can be both a past participial adjective ("*the halted train*") and a verb.

- Retaining only two top probability paths we get

  ^ *D A N A* -P_{1232}

  ^ *D A N V* -P_{1234}

Step-5

- ***^ The brown foxes jumped .***

- Can be both a past participial adjective ("*the halted train*") and a verb.

- Retaining only two top probability paths we get

  ^ *D A N A* -P_{1232}

  ^ *D A N V* -P_{1234}

Step-6: Final Step

- **^ *The brown foxes jumped .***
- On encountering dot, the beam search stops.
- We assume we get the correct path probabilistically in the beam (width 2)

  ^ *D A N V.*

How to fix the beam width (1/2)

- English POS tagging with Penn POS tag set: approximately 40 tags

- Fine categories like NNS for plural NNP for proper noun, VAUX for auxiliary verb, VBD for past tense verb and so on.

- A word can have on an average at most 3 POSs recorded in the dictionary.

How to fix the beam width (2/2)

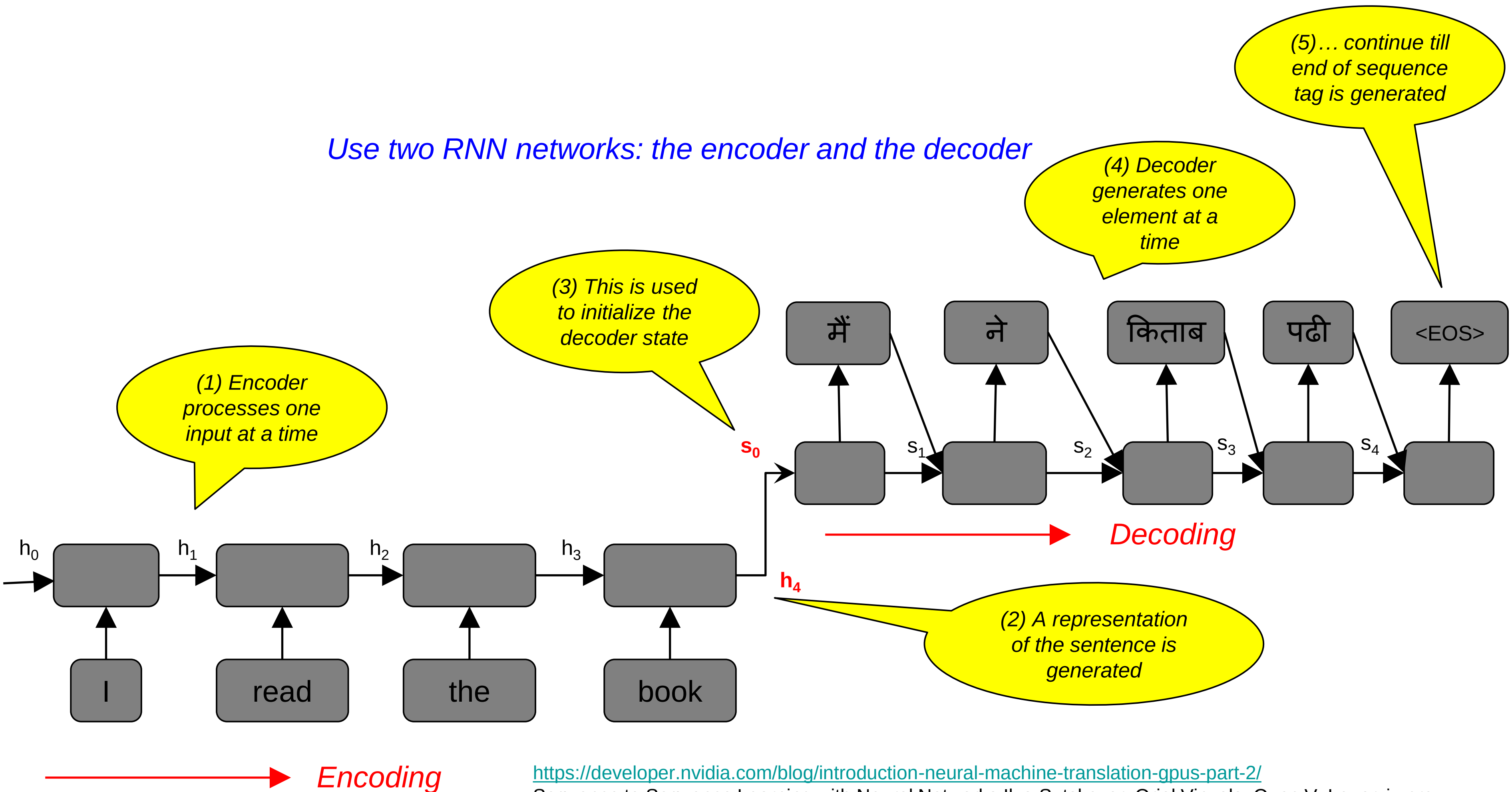
- Allow for 4 finer category POSs under each category and with support from a lexicon that records the broad category POSs,

- A practical beam width for POS tagging for English using Penn tagset could be 12 (=3 X 4). (think and justify)

Neural Decoding

Encode - Decode Paradigm Explained

Use two RNN networks: the encoder and the decoder



https://developer.nvidia.com/blog/introduction-neural-machine-translation-gpus-part-2/
Sequence to Sequence Learning with Neural Networks Ilya Sutskever, Oriol Vinyals, Quoc V. Le. arxiv pre-print [link]

Decoding in seq2seq

- There are 4 influencing factors for conditioning random variables -
  - Input encoding
  - Autoregression
  - Cross attention
  - Self attention

All searching is table lookup!

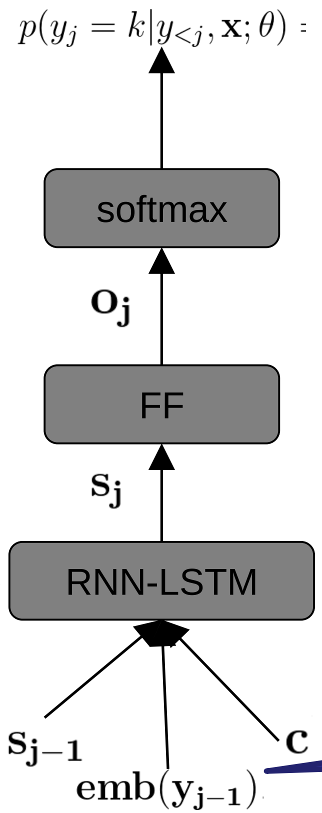
- Table look-up is equivalent to mapping
- Any form of search, is computing a mapping continuously, including the neural networks

Structural AI vs Functional AI

Structural AI	Functional AI
• Concerned with understanding the <u>anatomy</u> of a system	• Concerned with understanding the <u>behaviour</u> of a system
• Analogy to medicine: Doctors use graphs like EEG to understand anatomy of system	• Analogy to medicine: Attributes like facial expression, body language and pain are used to understand behaviour

- 80s and 90s, AI used to get ints inspiration and way forward from biology, neuro-physiology
- Today's AI finds the way forward from DATA

What is the decoder doing at each time-step?



This captures $y_{<j}$

This captures x, $c=h_4$

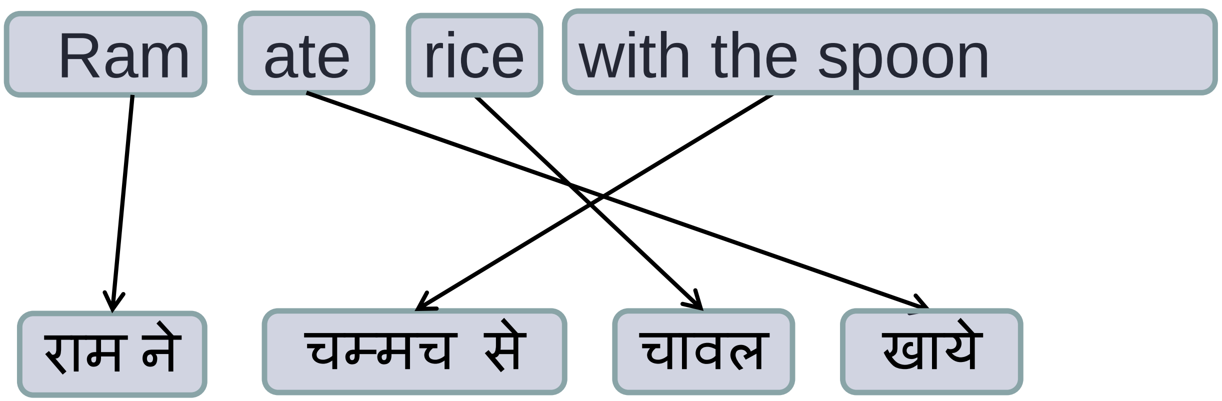
$$softmax(o_{jk}) = \frac{\exp(o_{jk})}{\sum_{m=0}^{m=T} \exp(o_{jm})}$$

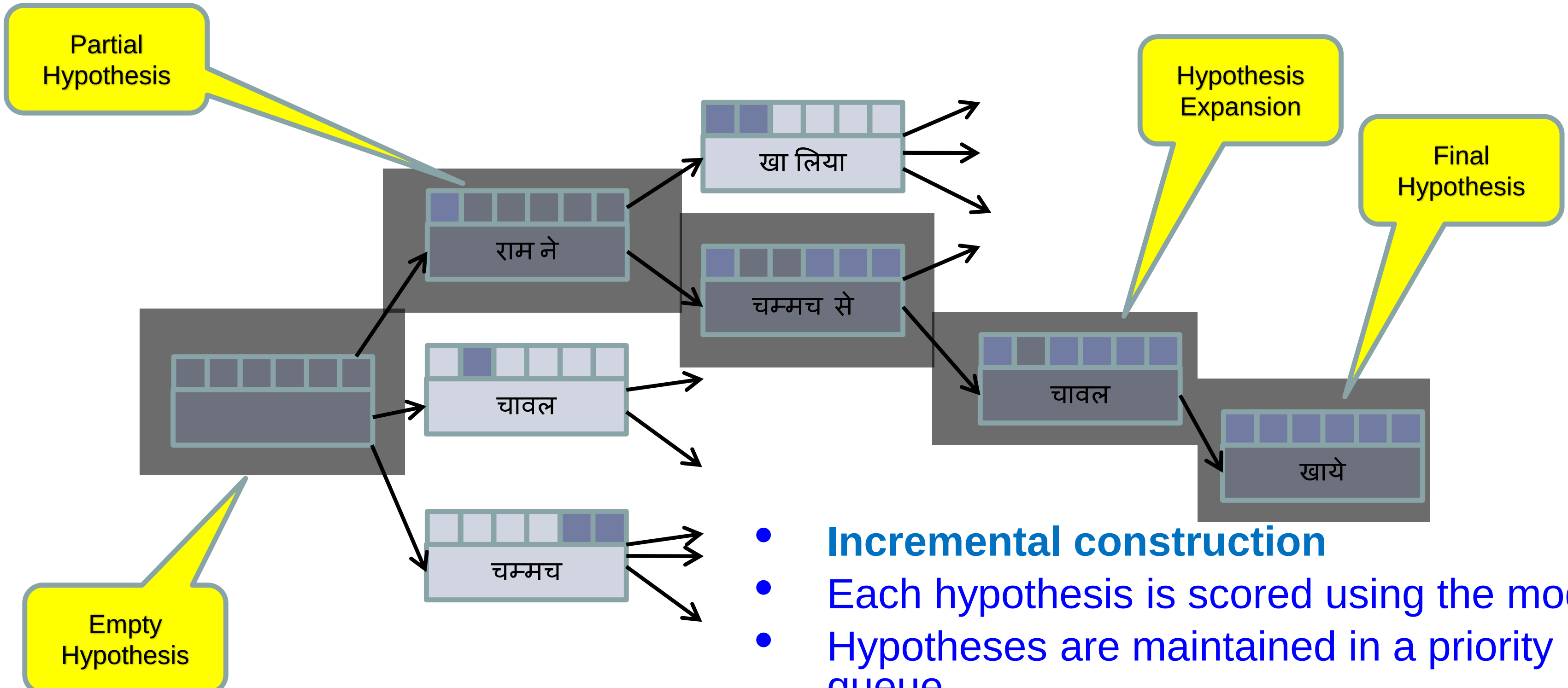
$$\mathbf{o_j} = FF(s_j)$$

$$\mathbf{s_j} = g(\mathbf{s_{j-1}}, \mathbf{emb}(\mathbf{y_{j-1}}), \mathbf{c})$$

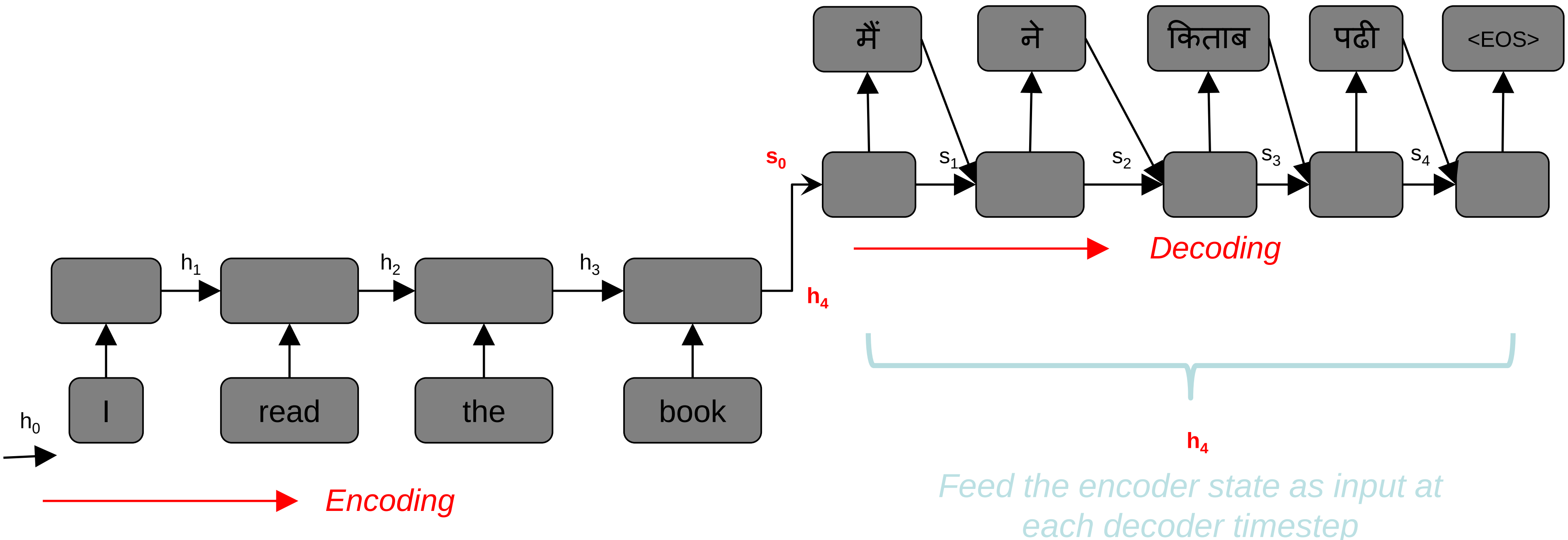
Decoding

Searching for the best translations in the space of all translations





- **Incremental construction**
- Each hypothesis is scored using the model
- Hypotheses are maintained in a priority queue



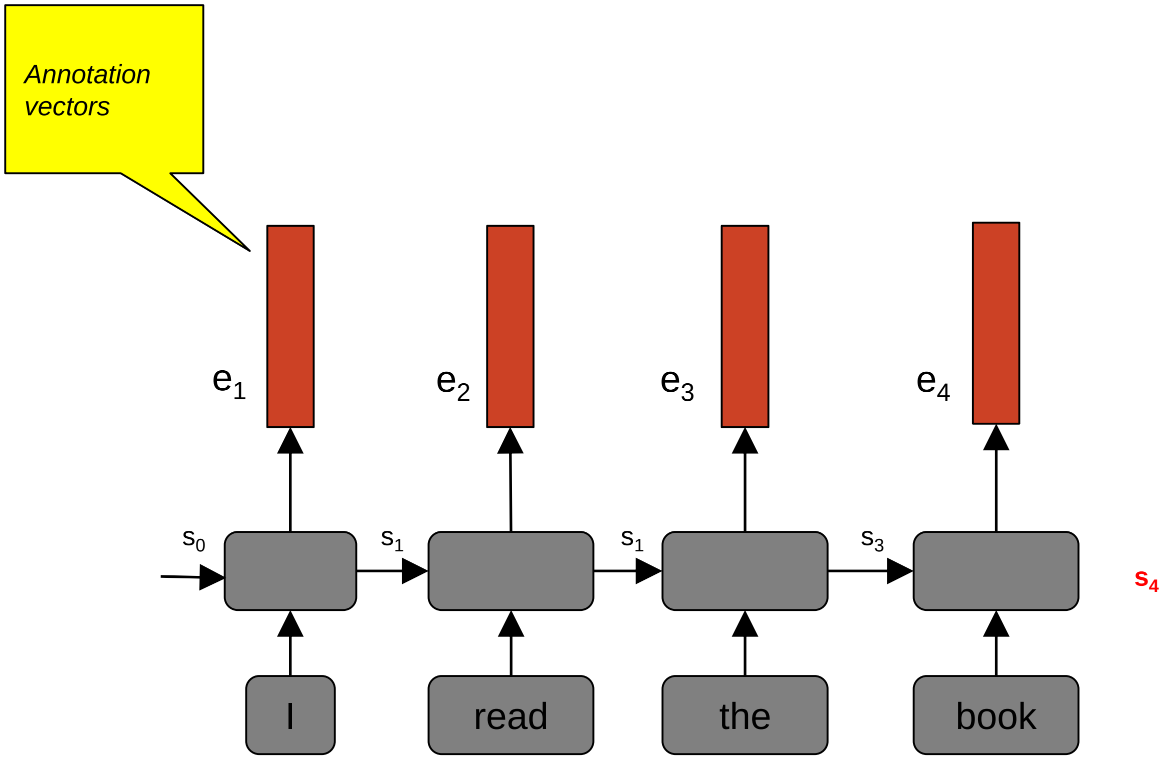
मैं
ने
किताब
पढी
<EOS>
s_0
s_1
s_2
s_3
s_4
Decoding
h_1
h_2
h_3
h_4
h_0
I
read
the
book
Encoding
h_4
Feed the encoder state as input at each decoder timestep

The entire source sentence is represented by a single vector

Problems

- Insufficient to represent to capture all the syntactic and semantic complexities
  - *Solution: Use a richer representation for the sentences*
- Long-term dependencies: Source sentence representation not useful after few decoder time steps
  - *Solution: Make source sentence information when making the next prediction*
  - *Even better, make **RELEVANT** source sentence information available*

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***

Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio. "Neural machine translation by jointly learning to align and translate." *ICLR 2015.*

https://developer.nvidia.com/blog/introduction-neural-machine-translation-gpus-part-3/

CNN

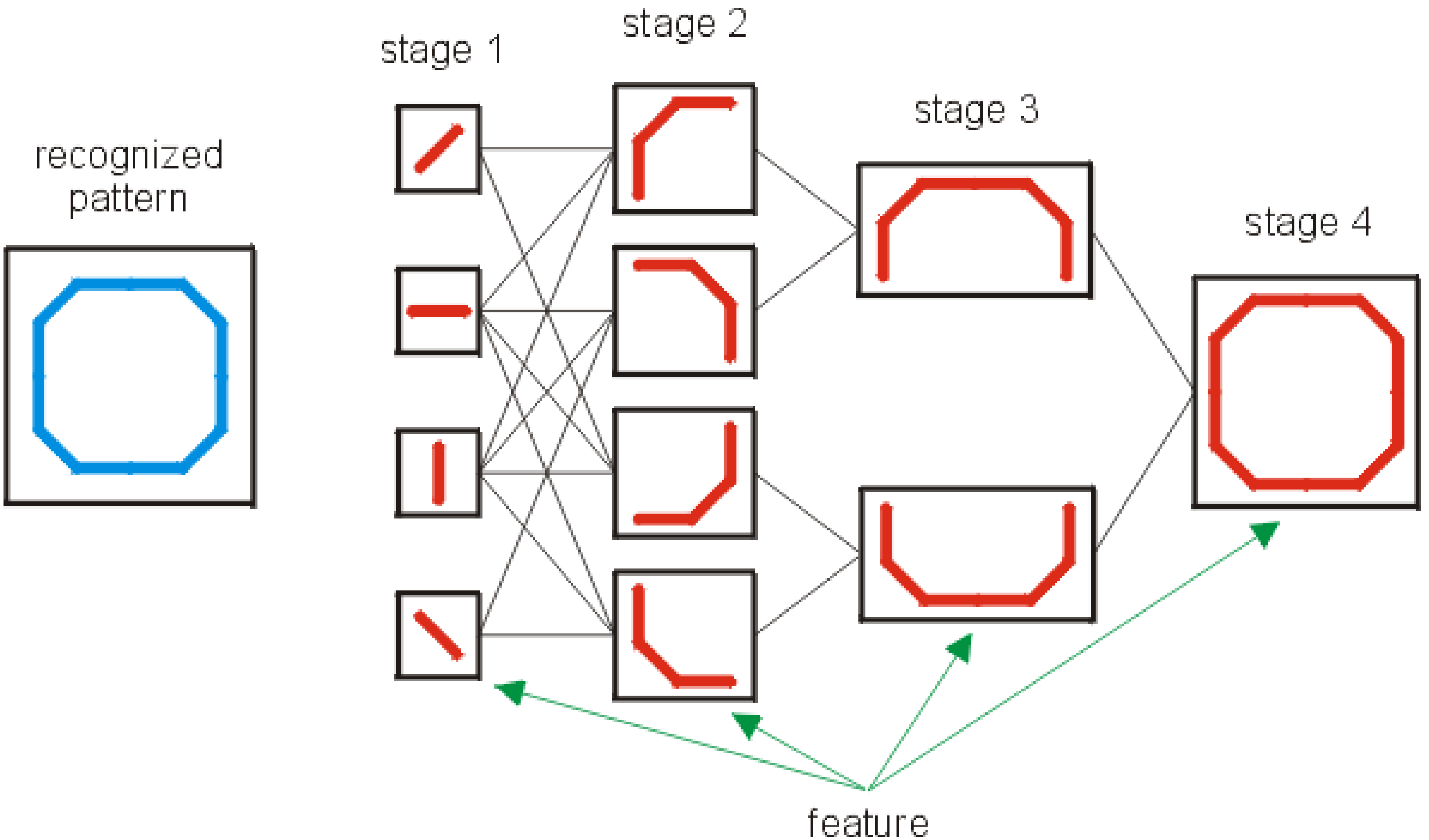
Two motivation points

- 1. Reduced number of parameters
- 2. Stepwise extraction of features
- These two are applicable to any AI situation, and not only vision and image processing

CNN= feedforward like + recurrent like!

- 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 like RNN

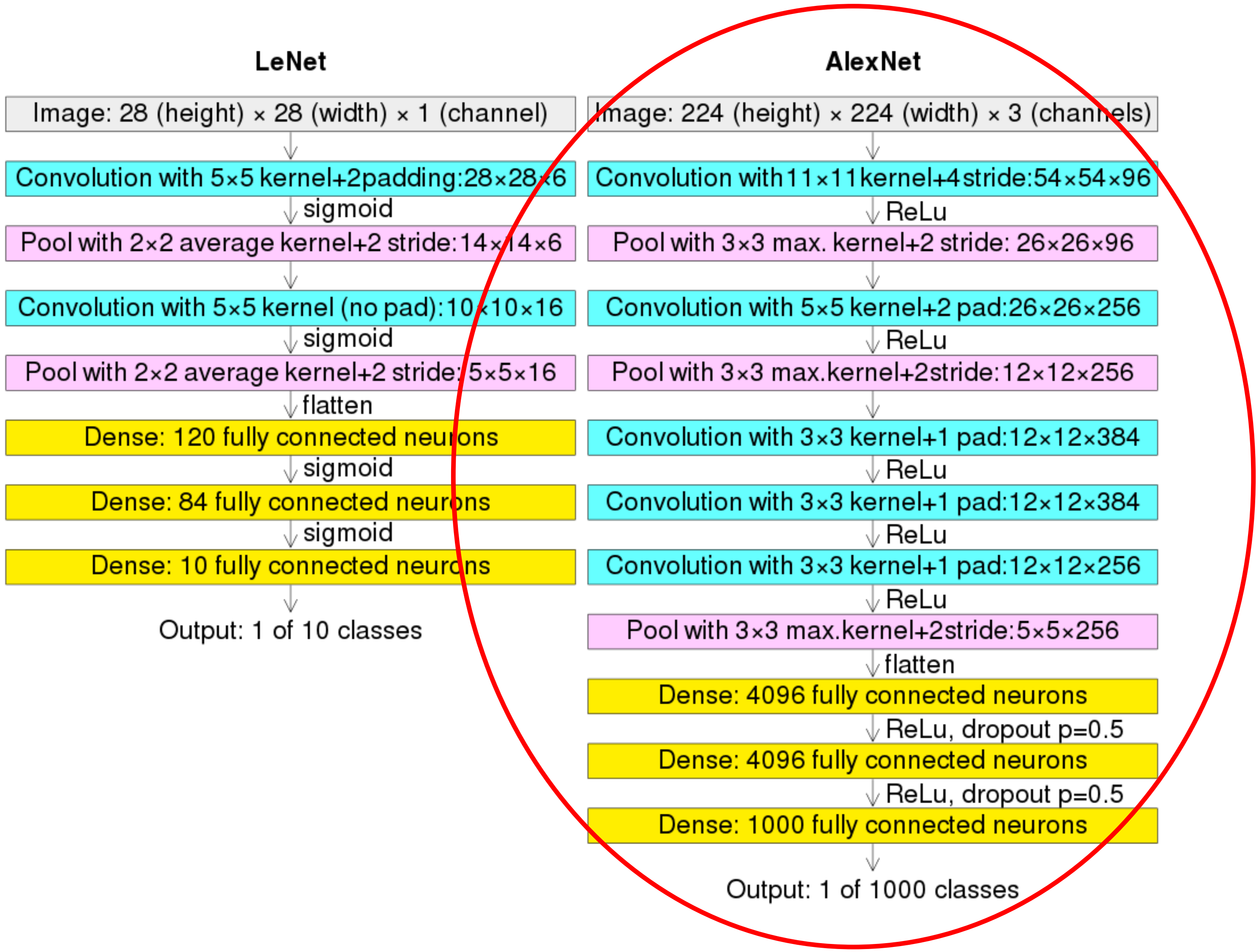
Genesis: Neocognitron (Fukusima, 1980)



Inspiration from biological processes

- Connectivity pattern between neurons resembles the organization of the animal visual cortex
- Individual cortical neurons respond to stimuli only in a restricted region of the visual field known as the receptive field
- Receptive fields of different neurons partially overlap such that they cover the entire visual field

The classic CNN (Wikipedia)



Convolution

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

Filter/kernel/

feature-detector

1 ×1	1 ×0	1 ×1	0	0
0 ×0	1 ×1	1 ×0	1	0
0 ×1	0 ×0	1 ×1	1	1
0	0	1	1	0
0	1	1	0	0

B/W Image

4	3	4
2	4	3
2	3	4

Convolved Feature

4= 1.1+1.0+1.1
+0.0+1.1+1.0
+0.1+0.0+1.1

Convolution basics

Convolution: continuous and discrete

$$(f * g)(t) = \int_{-\infty}^{+\infty} f(\tau)g(t-\tau)d\tau$$

This is the area under the curve $f(\tau)$ **weighted by** $g(t-\tau)$

$$(f * g)[n] = \sum_{m=-\infty}^{+\infty} f(m)g(n-m)$$

Convolution of two vectors

V_1: <0, 1, 2, 3, 4, 5, 6, 7, 8, 9>

V_2: <1, 1, 1>

$V_1 \oplus V_2 =$

<(0.1+1.1+2.1), (1.1+2.1+3.1),
(2.1+3.1+4.1), (3.1+4.1+5.1),
(4.1+5.1+6.1), (5.1+6.1+7.1),
(6.1+7.1+8.1), (7.1+8.1+9.1)>

=<3, 6, 9, 12, 15, 18, 21, 24>

Receptive field and selective emphasis/de-emphasis

- The filter <1,1,1> given equal "emphasis" to constituents of the "receptive field" which means region of interest
- Sliding of the filter corresponds to taking different receptive fields
- By designing the filter as <0,1,0>, we emphasise the center of the receptive field

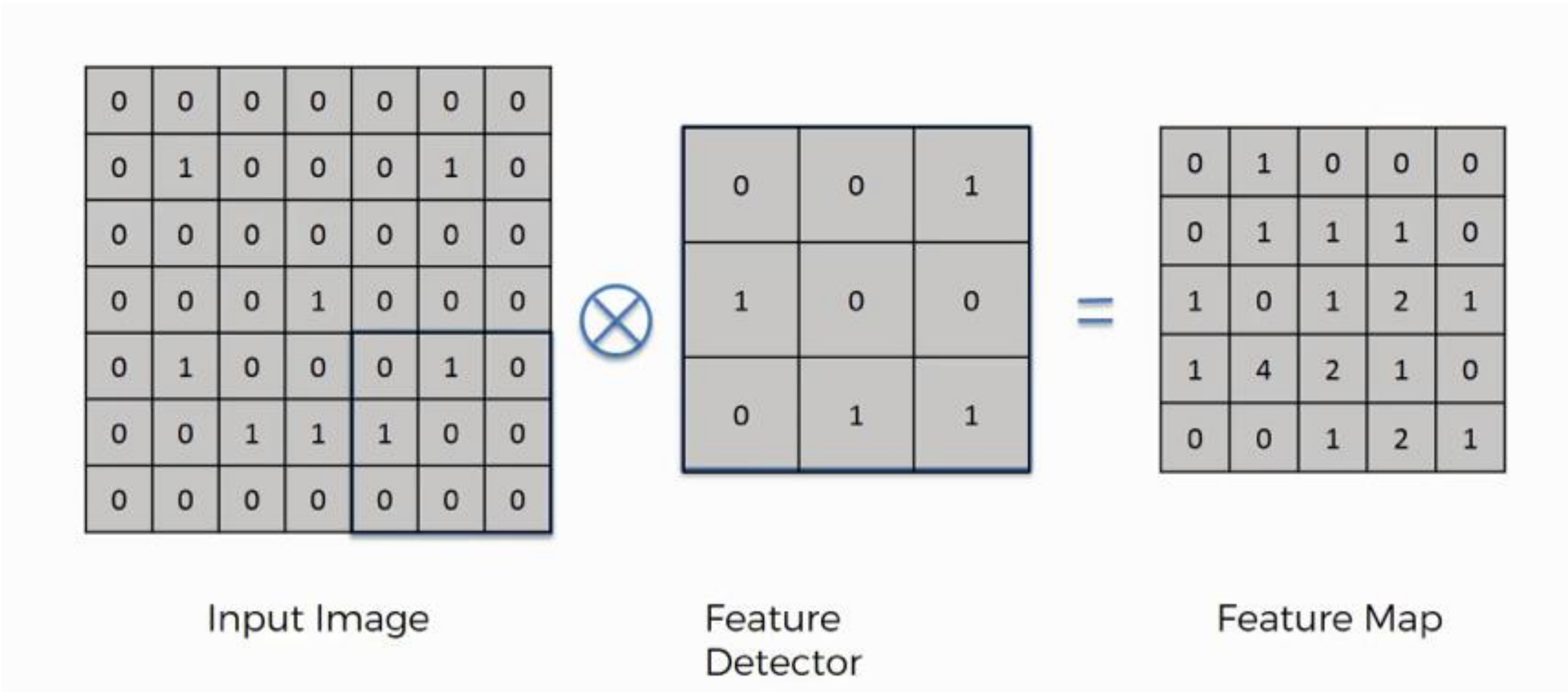
“dog” image and “cat” image

- For dog, the face is of conical shape
- For cat, the shape is round
- So, this distinguishing feature is important for classification
- The filter should have the ability of detecting this kind of feature

Interpretation of convolution

- The filter/kernel/feature_extractor highlights features and obtains those features
- The sliding achieves the effect of focussing on “region” after “region”
- This resembles sequence processing
- The filter components are **LEARNT**

Convolution as feature extractor



CNN architecture

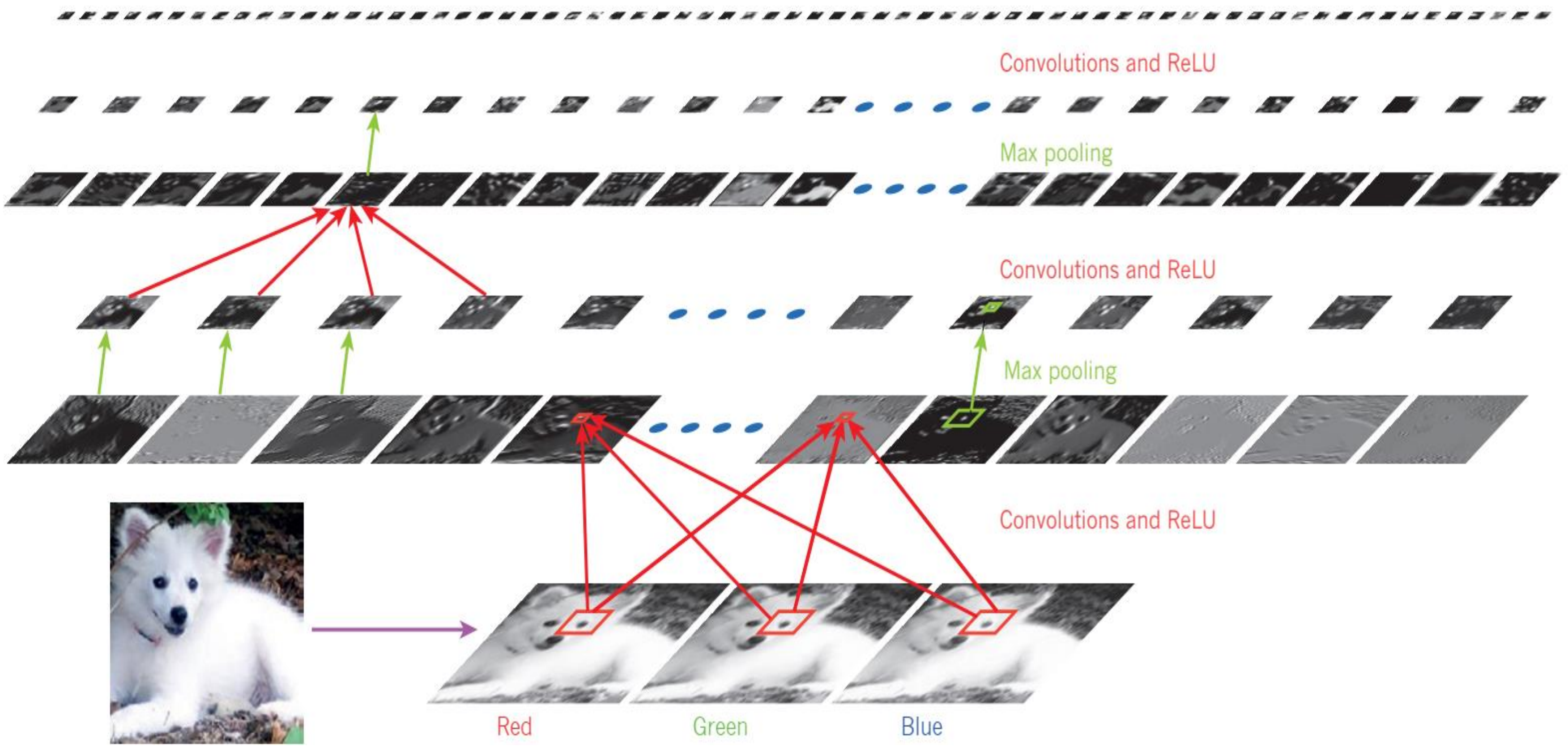
- Several layers of convolution with *tanh* or *ReLU* applied to the results
- In a traditional feedforward neural network we connect each input neuron to each output neuron in the next layer. That's also called a fully connected layer, or affine layer.
- In CNNs we use convolutions over the input layer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

Key Ideas

Four key ideas that take advantage of the properties of natural signals:

- local connections,
- shared weights,
- pooling and
- the use of many layers

A typical ConvNet



Lecun, Bengio, Hinton, Nature, 2015

Why CNN became a rage: image

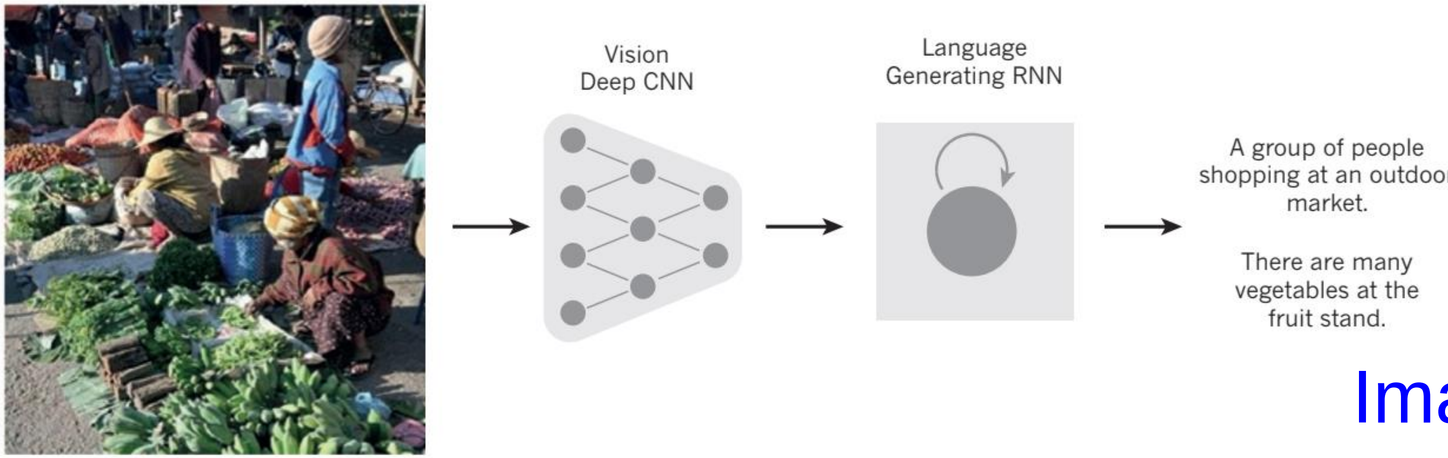


Image Captioning-1

A **stop** sign is on a road with a mountain in the background

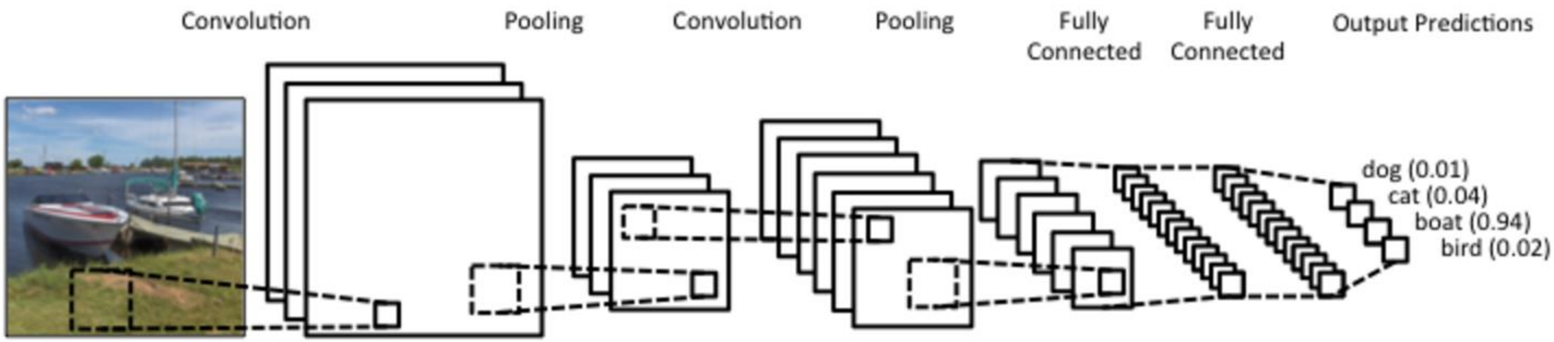
Image Captioning-2

Role of ImageNet

- Million images from the web
- 1,000 different classes
- Spectacular results!
- Almost halving the error rates of the best competing approaches1.

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.



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?