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

EnCo-DeCo, A*, Viterbi, Beam Search, Neural Decoding

Pushpak Bhattacharyya

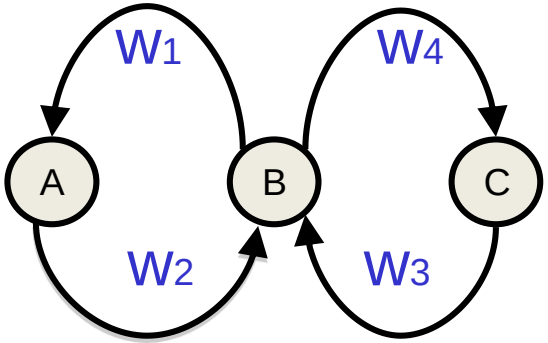
Computer Science and Engineering Department

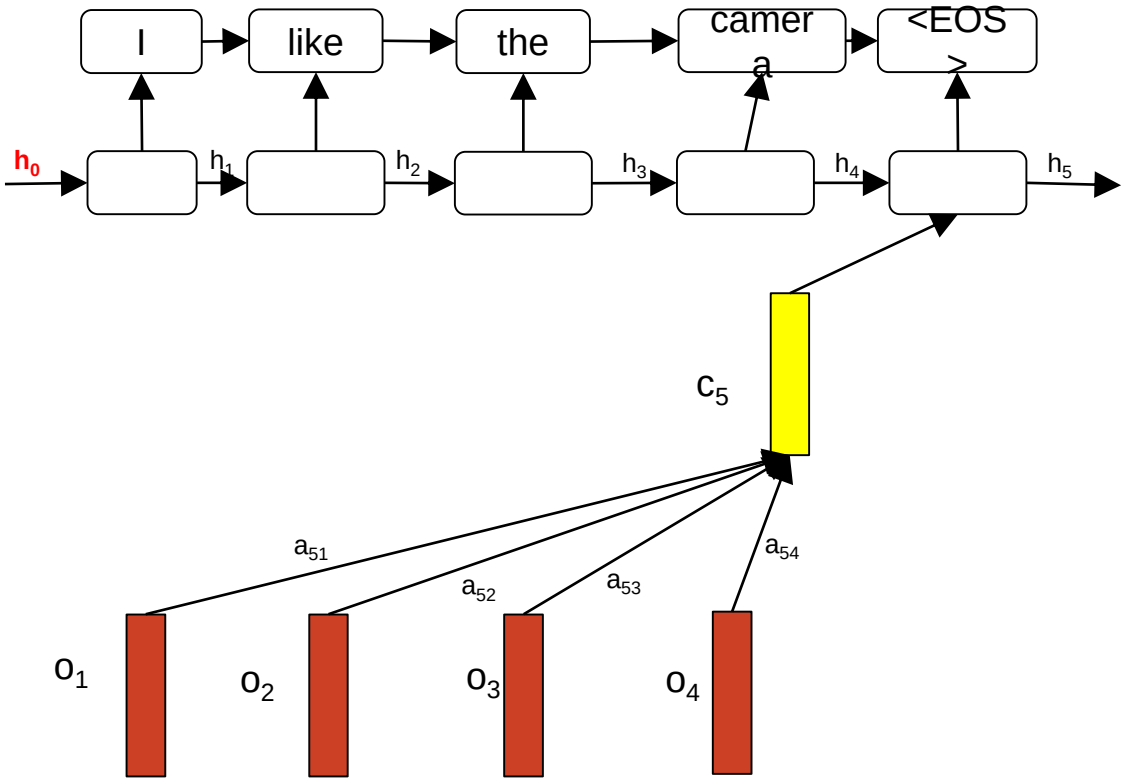
IIT Bombay

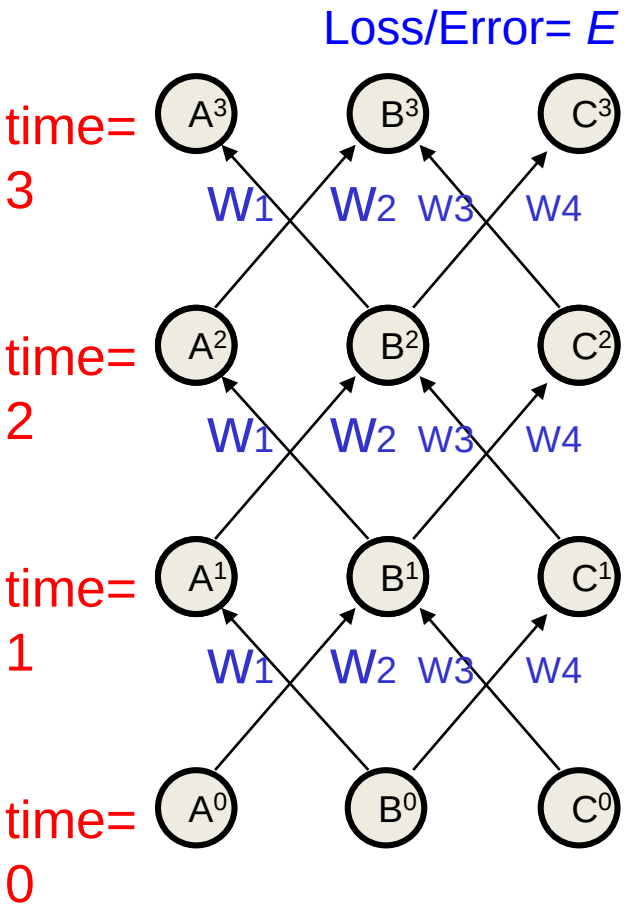
Week 8 of 19feb24

1-slide recap

- EnCo-DeCo
- BPTT
- Foundations of representation learning
- Nittie gritties of BP







Sequence Labelling Task

Input Sequence: $(x_1\ x_2\ x_3\ x_4 \ldots\ldots x_i \ldots\ldots x_N)$

Output Sequence: $(y_1\ y_2\ y_3\ y_4 \ldots\ldots y_i \ldots\ldots y_N)$

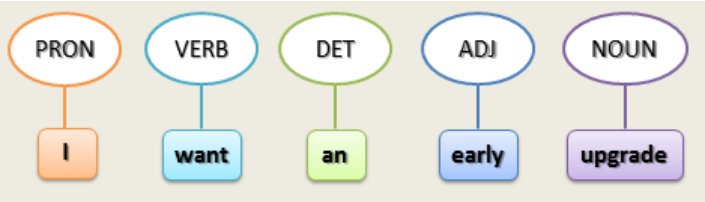
Input and output sequences have the same length

Variable length input

Output contains categorical labels

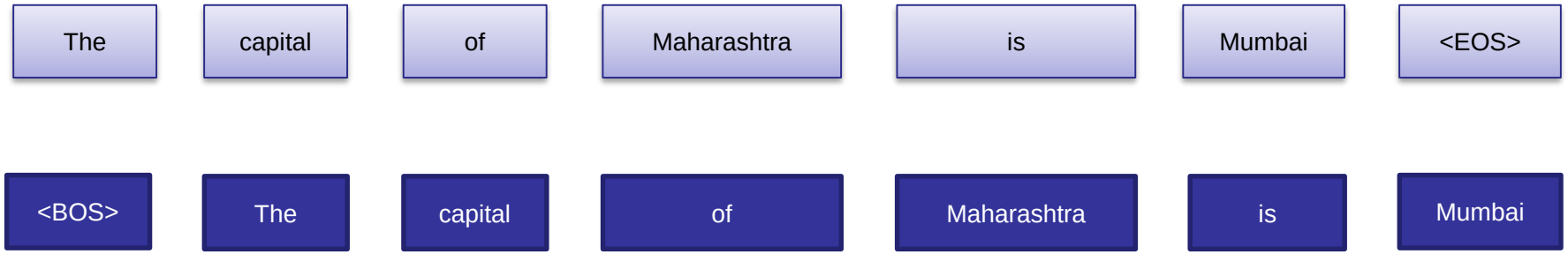
Output at any time-step typically depends on neighbouring output labels and input elements

Part-of-speech tagging



Recurrent Neural Network is a powerful model to learn sequence labelling tasks

How do we model language modeling as a sequence labeling task?



The output sequence is one-time step ahead of the input sequence

Evaluating Language Models

How do we evaluate quality of language models?

Evaluate the ability to predict the next word given a context

Evaluate the probability of a testset of sentences

Standard test sets exist for evaluating language models: Penn Treebank, Billion Word Corpus, WikiText

Model	Perplexity
Interpolated Kneser-Ney 5-gram (Chelba et al., 2013)	67.6
RNN-1024 + MaxEnt 9-gram (Chelba et al., 2013)	51.3
RNN-2048 + BlackOut sampling (Ji et al., 2015)	68.3
Sparse Non-negative Matrix factorization (Shazeer et al., 2015)	52.9
LSTM-2048 (Jozefowicz et al., 2016)	43.7
2-layer LSTM-8192 (Jozefowicz et al., 2016)	30
Ours small (LSTM-2048)	43.9
Ours large (2-layer LSTM-2048)	39.8

https://research.fb.com/building-an-efficient-neural-language-model-over-a-billion-words/

RNN models outperform n-gram models

A special kind of RNN network – LSTM- does even Better

Long distance dependencies: Sentence-1

- Ram who is a good student and lives in London which is a large metro, will go to the University for higher studies.
- राम जो एक अच्छा छात्र है और लंदन में रहता है जो एक बड़ी मेट्रो है, उच्च अध्ययन के लिए विश्वविद्यालय जाएगा।

Sentence-2

- Sita who is a good student and lives in London which is a large metro, will go to the University for higher studies.
- सीता जो एक अच्छी छात्रा है और लंदन में रहती है जो एक बड़ी मेट्रो है, उच्च अध्ययन के लिए विश्वविद्यालय जाएगी।

Long distance dependency: WSD

The *bank*

Long distance dependency: WSD

The *bank* that Ram

Long distance dependency: WSD

The *bank* that Ram used to visit

Long distance dependency: WSD

The *bank* that Ram used to visit 30 years before

Long distance dependency: WSD

The *bank* that Ram used to visit 30 years before was closed

Long distance dependency: WSD

The *bank* that Ram used to visit 30 years before was closed due to

Long distance dependency: WSD

The *bank* that Ram used to visit 30 years before was closed due to the lockdown

Long distance dependency: WSD

The *bank* that Ram used to visit 30 years before was closed due to the lockdown with the Govt

Long distance dependency: WSD

The *bank* that Ram used to visit 30 years before was closed due to the lockdown with the Govt. getting worried that

Long distance dependency: WSD

The *bank* that Ram used to visit 30 years before was closed due to the lockdown with the Govt. getting worried that crowding of people

Long distance dependency: WSD

The *bank* that Ram used to visit 30 years before was closed due to the lockdown with the Govt. getting worried that crowding of people during the

Long distance dependency: WSD

The *bank* that Ram used to visit 30 years before was closed due to the lockdown with the Govt. getting worried that crowding of people during the immersion ceremony

Long distance dependency: WSD

The *bank* that Ram used to visit 30 years before was closed due to the lockdown with the Govt. getting worried that crowding of people during the immersion ceremony on the river will aggravate the situation.

Movement of probability mass for “bank”

- Seeing “closed”, probability mass edges toward “financial” sense, because of strong association between “bank” and “closed/open”
- “lockdown” pushed this probability mass towards “river bank”
- Push further strengthened by arrival of “crowding”, “immersion” and “river” one after the other; “river” closes the case!

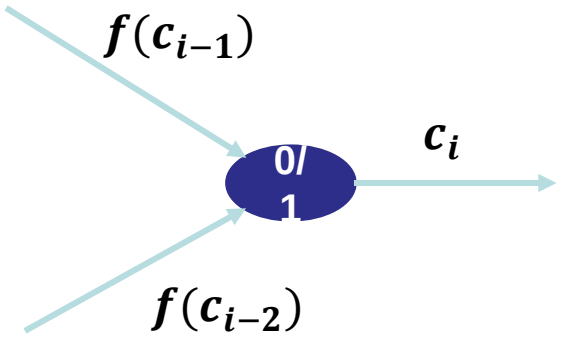
LSTM

Vanishing Gradient Problem ➔ Intuition

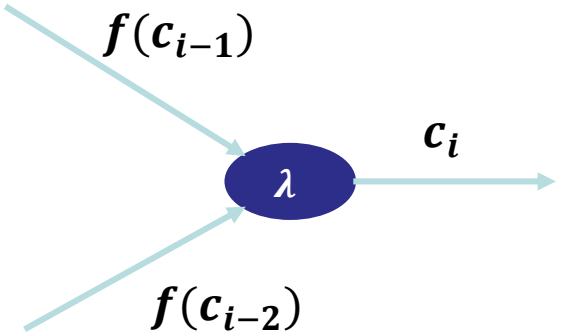
- The product of gradients can become small if the some of these quantities are small
- Gradient signal can become small for positions that are farther away from a position under consideration ➔ can't learn long term dependency
- Truncated BPTT ➔ restrict the product to fewer terms
- Exploding Gradient: Gradient can also become large ➔ clip the gradient before

Selecting hidden states that affect current hidden state

- What if there was a switch to select which previous hidden state should impact current hidden state?

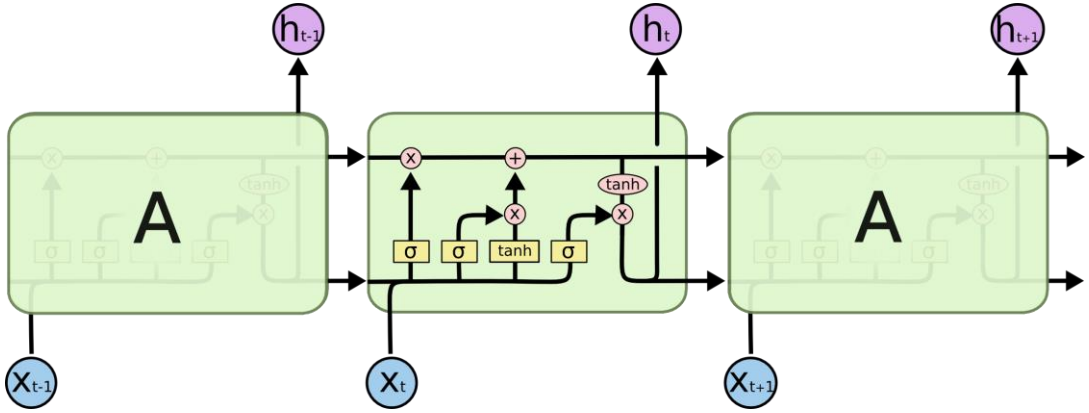


- Better still, we had a soft-switch

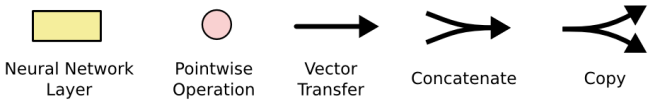


Addressing VG: Long Short-Term Memory (LSTM)

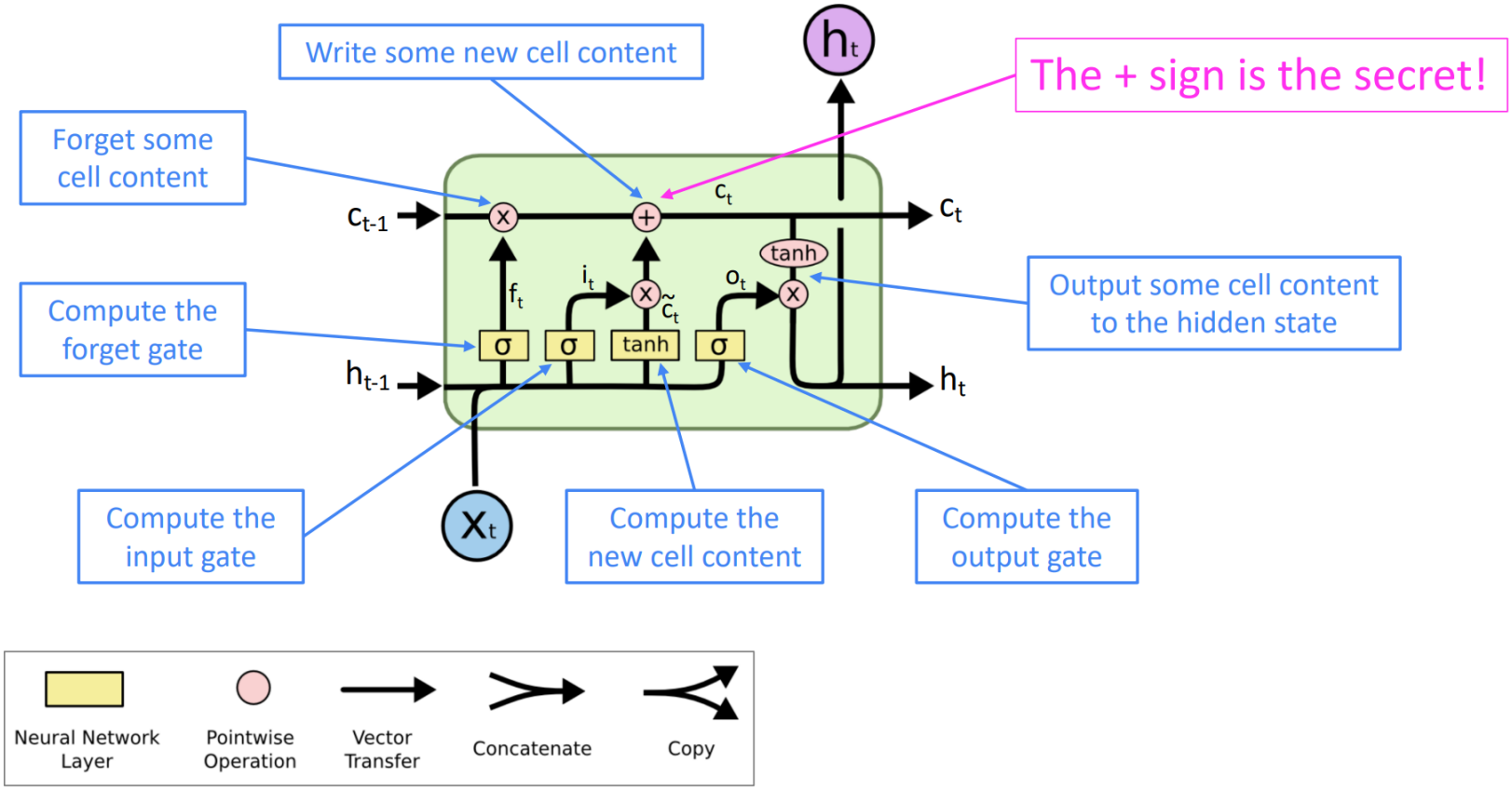
- Formalization of the ideas just discussed
- A special kind of RNN-cell that enables selective read/write/erasure



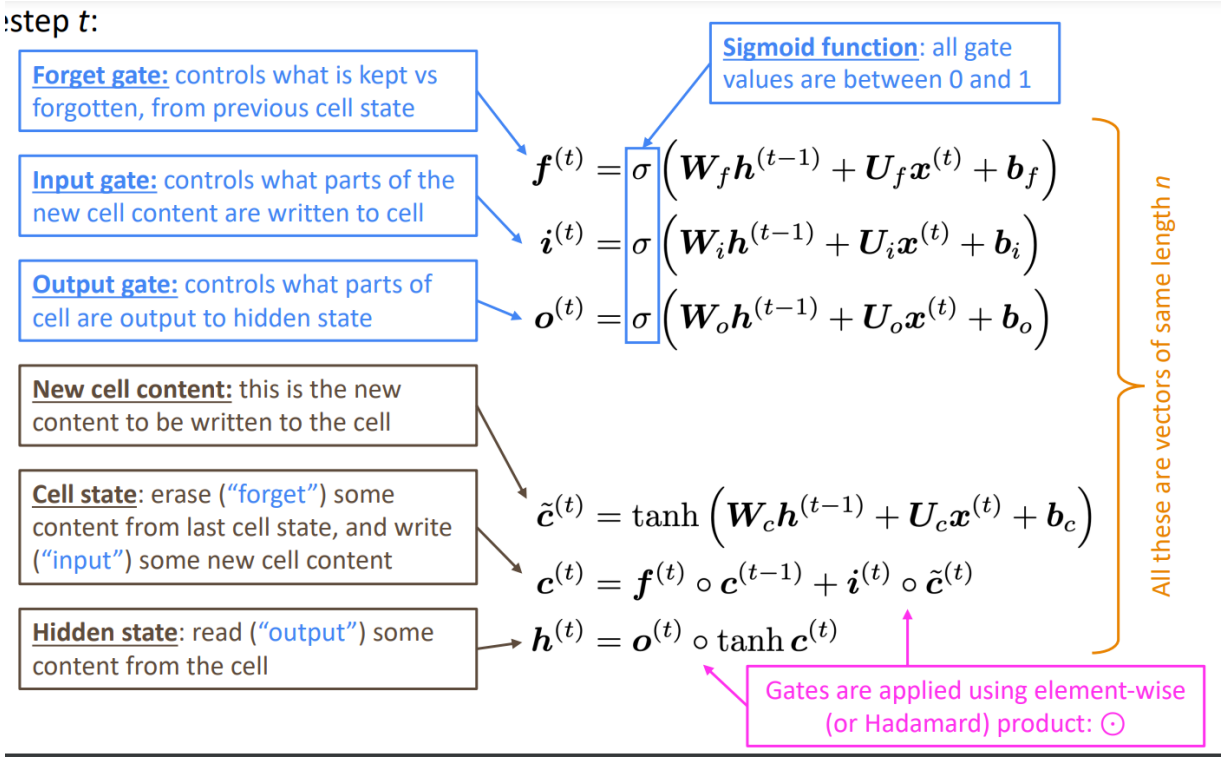
Different gates (shown with crosses)control flow of information



http://colah.github.io/posts/2015-08-Understanding-LSTMs



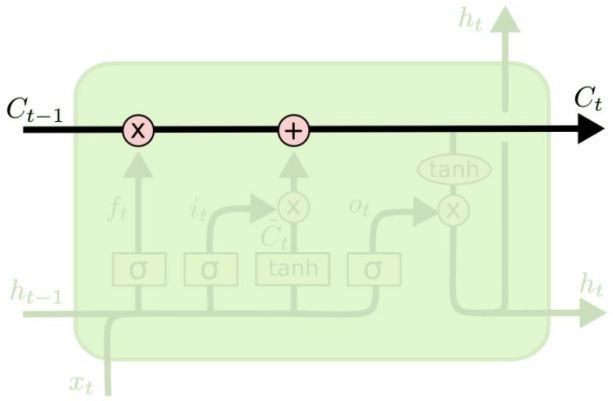
http://web.stanford.edu/class/cs224n/slides/cs224n-2021-lecture06-fancy-rnn.pdf



http://web.stanford.edu/class/cs224n/slides/cs224n-2021-lecture06-fancy-rnn.pdf

LSTMs solve vanishing gradient problem

- Intuition: “The cell state is kind of like a conveyor belt. It runs straight down the entire chain, with only some minor linear interactions. It’s very easy for information to just flow along it unchanged.”



- You can show that the gradient of loss with respect to cell state depends on forget gate values along the path
- For a detailed explanation see this: http://www.cse.iitm.ac.in/~miteshk/CS7015/Slides/Teaching/pdf/Lecture15.pdf

Other Noteworthy Points

- Bi-directional RNNs
  - All NLU applications typically use this to incorporate context in both directions
  - Run separate RNNs in forward and backward directions
  - Concatenate hidden states at each time step for bidirectional context vector
- Deep RNNs: RNNS layers can be stacked
- Gated Recurrent Units (GRU): a simpler variant of LSTM
- LSTM-CRF network: A LSTM-variant where dependencies between output variables can be modeled

Suggested Reading

- N-gram Language Models (textbook chapter)
- Smoothing techniques: http://nrs.harvard.edu/urn-3:HUL.InstRepos:25104739
- FF-Neural LM: https://www.jmlr.org/papers/volume3/bengio03a/bengio03a.pdf
- The Unreasonable Effectiveness of Recurrent Neural Networks (nice blog post overview)
- Sequence Modeling: Recurrent and Recursive Neural Nets (Sections 10.1 and 10.2)
- On Chomsky and the Two Cultures of Statistical Learning
- Sequence Modeling: Recurrent and Recursive Neural

Suggested Reading

- On the difficulty of training Recurrent Neural Networks (proof of vanishing gradient problem)
- Vanishing Gradients Jupyter Notebook (demo for feedforward networks)
- Understanding LSTM Networks (must read, blog post overview)
- https://r2rt.com/written-memories-understanding-deriving-and-extending-the-lstm.html

Suggested Reading

- Original LSTM papers:
  - http://www.bioinf.jku.at/publications/older/2604.pdf (the original paper)
  - ftp://ftp.idsia.ch/pub/juergen/TimeCount-IJCNN2000.pdf (peephole variant that is most widely used)
- Pretrained Models
  - ELMo
  - ULMFit
- Other lectures on RNNs
  - https://github.com/oxford-cs-deepnlp-2017/lectures (Lectures 3 & 4)
  - https://www.cse.iitm.ac.in/~miteshk/CS7015.html (Lectures 14 & 15)
  - http://web.stanford.edu/class/cs224n/ (Lectures 5 & 6)

Decoding- A* Algorithm

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	

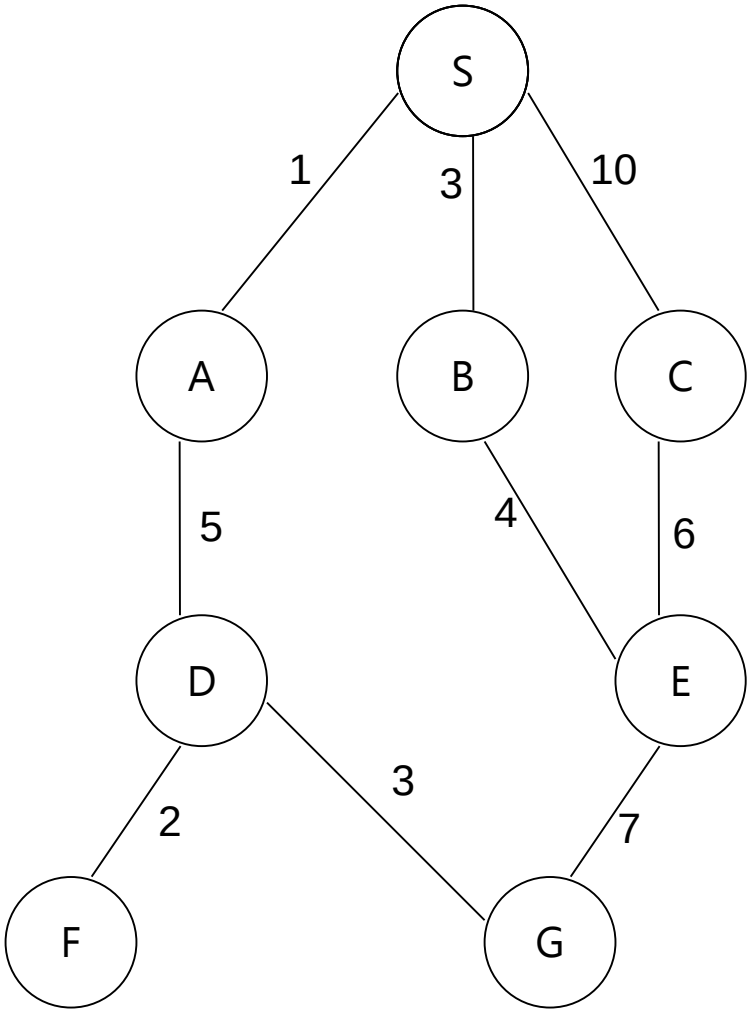
G

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

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

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

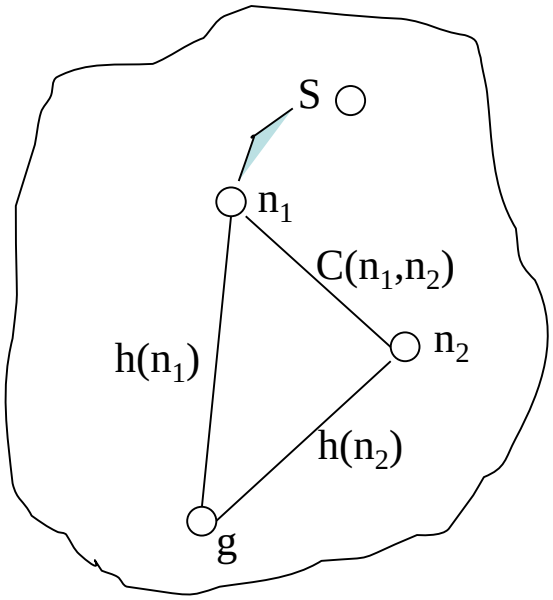
Visit that node whose f-value is the smallest.

OL is a queue (BFS)

OL is stack (DFS)

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

→ $h = 0$ → estimate is same for all the nodes.



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

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

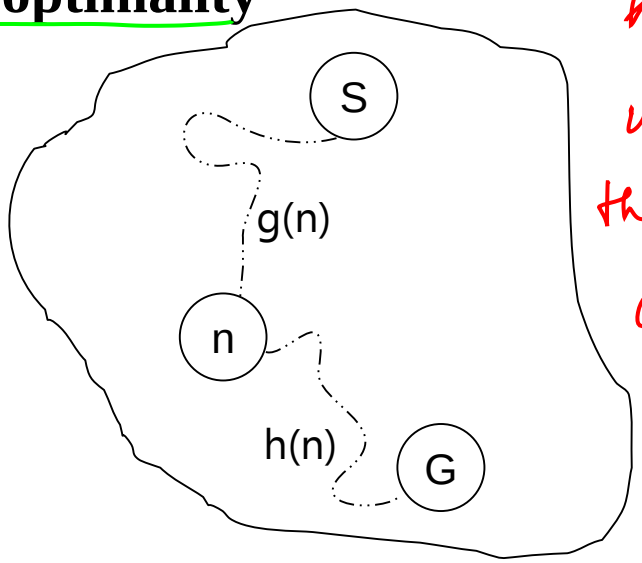
Important

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**"

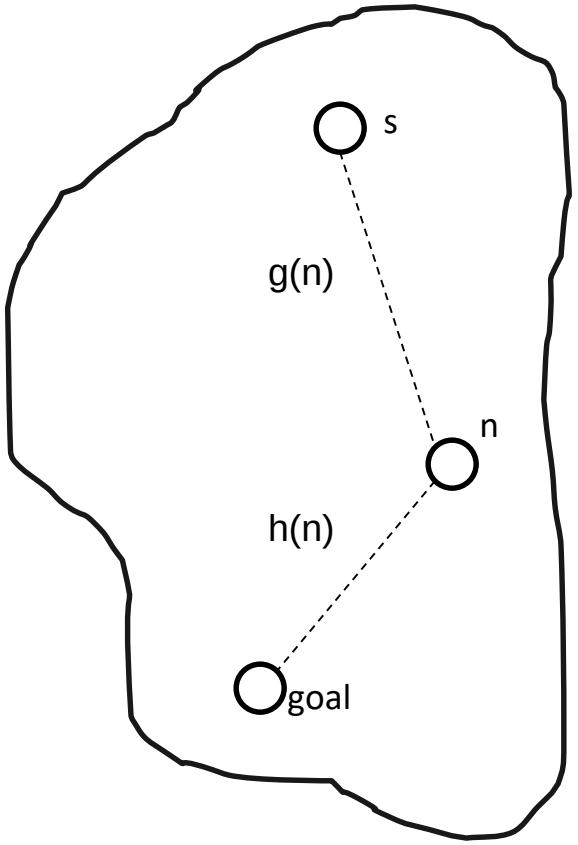
leads to optimality.



h = heuristic function, if we choose h properly then lot of unnecessary computations can be avoided.

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

Manhattan Dist.
1 → 0, 2 → 2
3 → 2, 4 → 0
5 → 2, 7 → 4
6 → 1
8 → 1

6 heuristics value, since 6 coins has to be moved.

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

these coins stays constant.

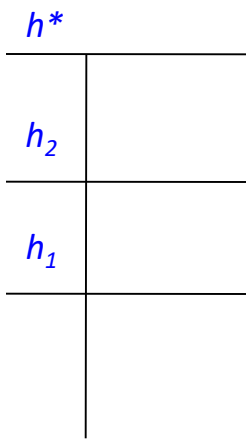
$h^*(n)$ = actual no. of moves to transform n to g

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.

streets are horizontal, avenues are vertical.

$h_1(n) \leq h^*(n)$ and $h_1(n) \leq h^*(n)$

Grid structure, x, y → Manhattan distance → estimating easier.



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.

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

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.

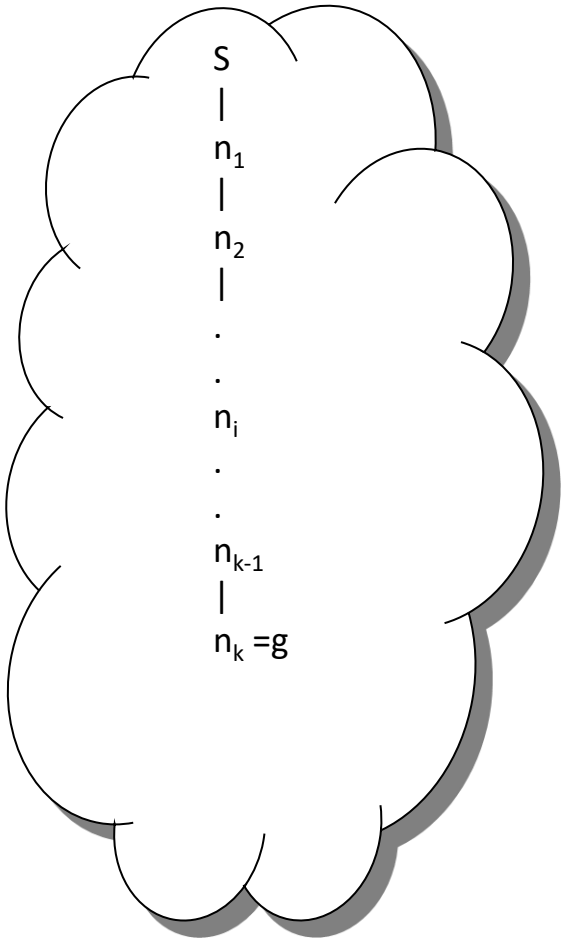
This means,

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

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

Key point about A* search

BFS and DFS has ← h=0 = ignorant algos, brute force algos.



Statement:

Let S -n_1-n_2-n_3…n_i…-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,…,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 $s \rightarrow n_1 \rightarrow n_2 \rightarrow \ldots \rightarrow g$,
  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:

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

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

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

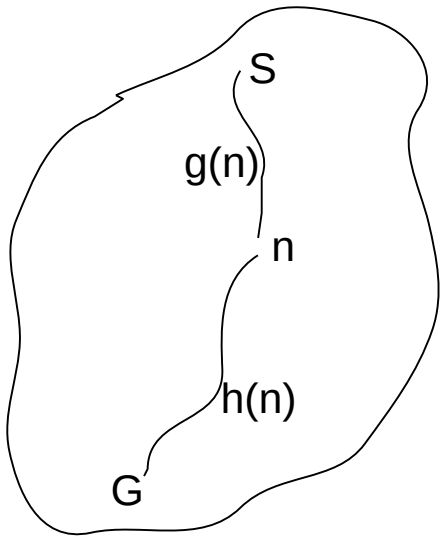
$$h^*(n_{i+1}) = h^*(n_i) - c(n_i, n_{i+1})$$

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.

Intuition



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

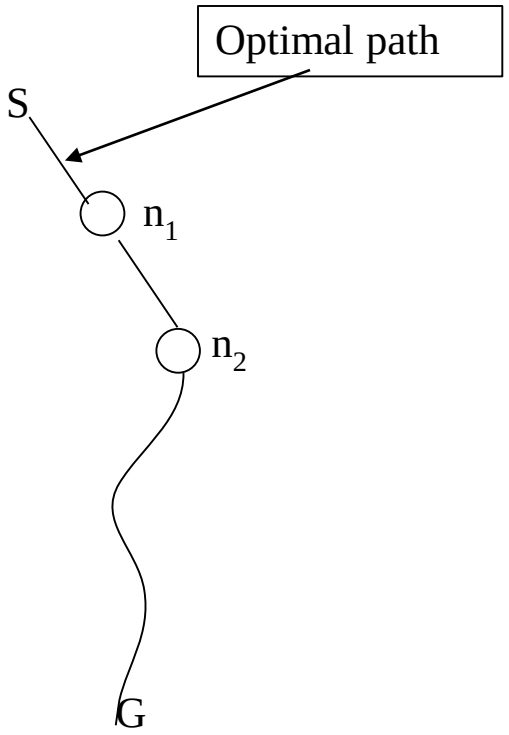
(2) If A* does not terminate, the *f* value of the nodes expanded become unbounded.

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,

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

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

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

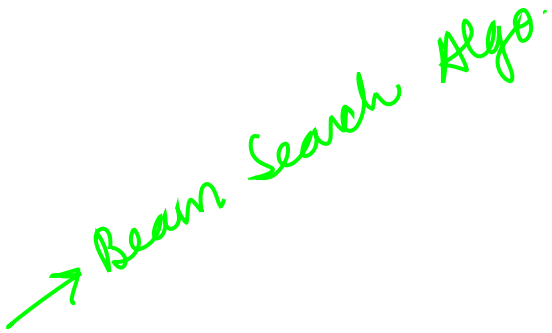
If the *h* value for every node over-estimates the *h** value of the corresponding node by a constant, then the path found need not be costlier than the optimal path by that constant. Prove this.

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 → Finite state Automaton
- Viterbi Search on Probabilistic FSA
- Hill Climbing
- Simulated Annealing → probabilistic search
- Gradient Descent
- Stack Based Search
- Genetic Algorithms → stochastic search / not based on prob. based on mutation & cross over.
- Memetic Algorithms



Viterbi Decoding

Illustration with POS tagging

Sentence: "*People Dance*"

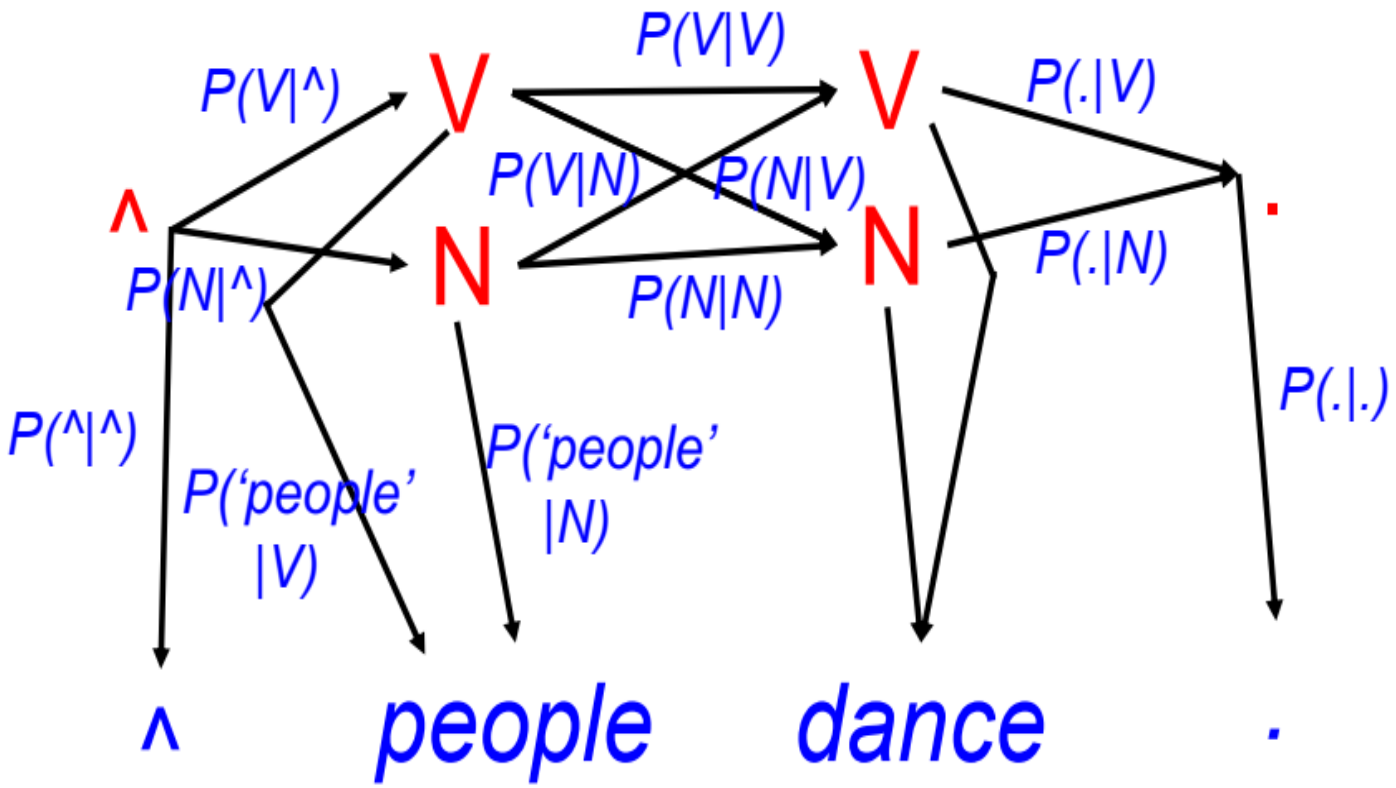
- '*people*' and '*dance*' can both be both nouns and verbs, as in
  - "*old_JJ people_NNS*" ('*people*' as noun)
  - "*township_NN peopled_VBN with soldiers_NNS*" ('*people*' as verb)
- as well as
  - "*rules_NNS of_IN classical_JJ dance_NN*" ('*dance*' as noun)
  - "*will*_VAUX *dance_VB well_RB*"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

Possible Tags: "^ *people dance .*"

- for simplicity we take single letter tags- *N: noun, V: verb*:
  - ^ *N N .*
  - ^ *N V .* → correct
  - ^ *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?

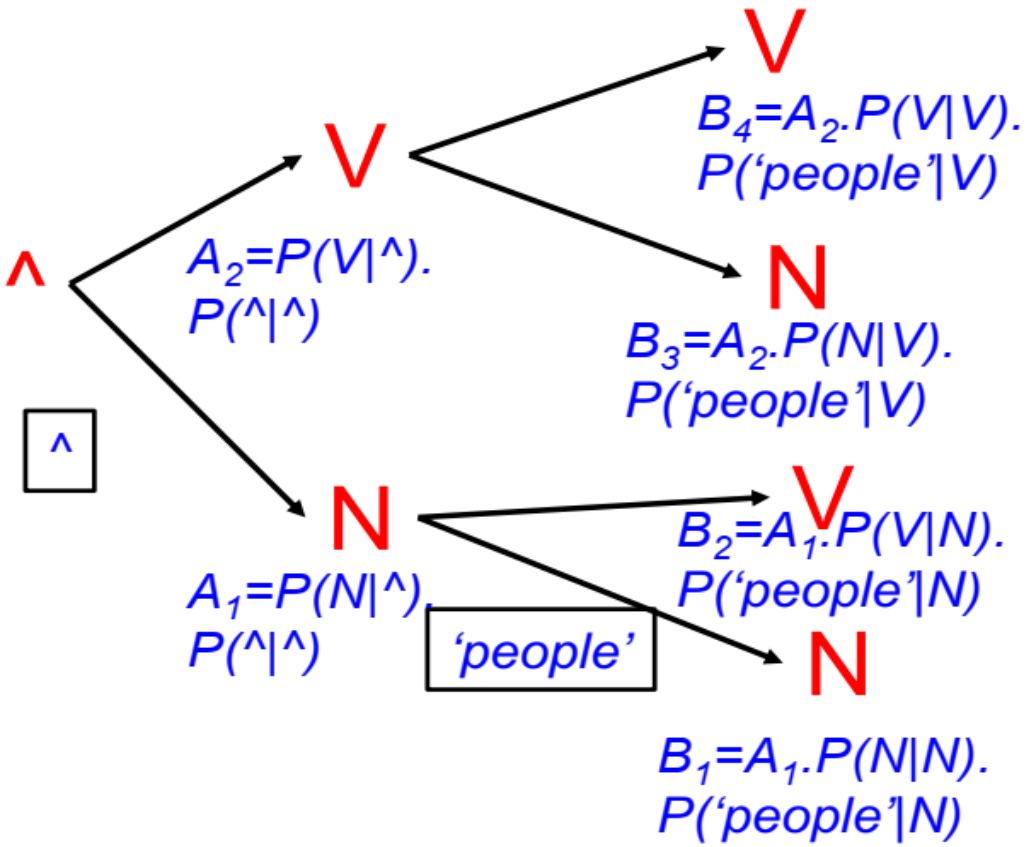
Step-1: Trellis

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

  $P(N|\hat{})=0.8$, $P(V|\hat{})=0.2$ and of course $P(\text{“}\hat{}\text{”}|\hat{})=1.0$
- 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

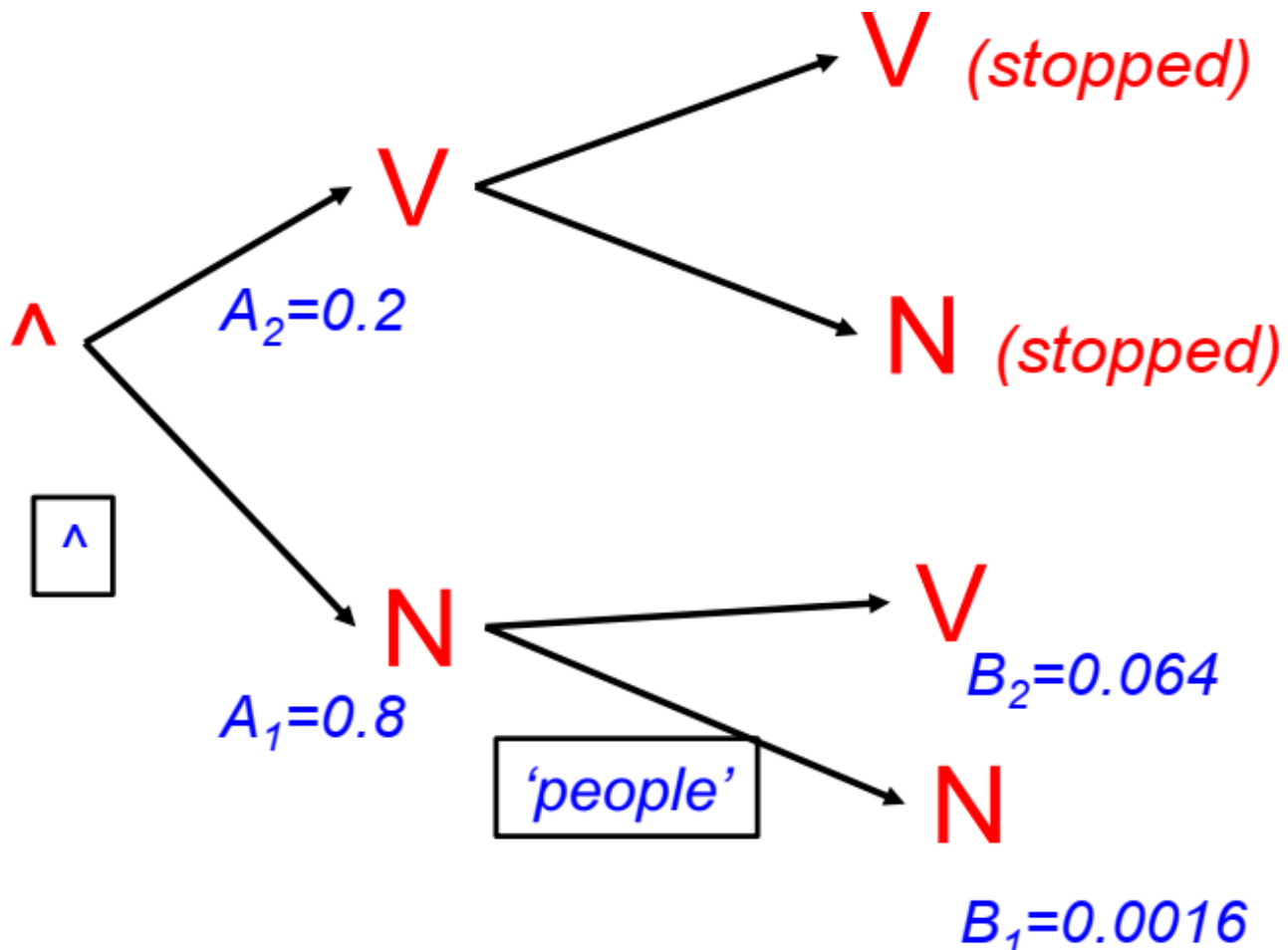
  - *P(‘people’|N)=0.01, P(‘people’|V)=0.001*
- Note: *N N*: golf club, cricket bat, town people- ambiguity “The town people visited was deserted”/”town people will not be able to live here”
- V V combination: Hindi- *has padaa* (laughed suddenly), Bengali- *chole gelo* (went away)

Calculate *B*s

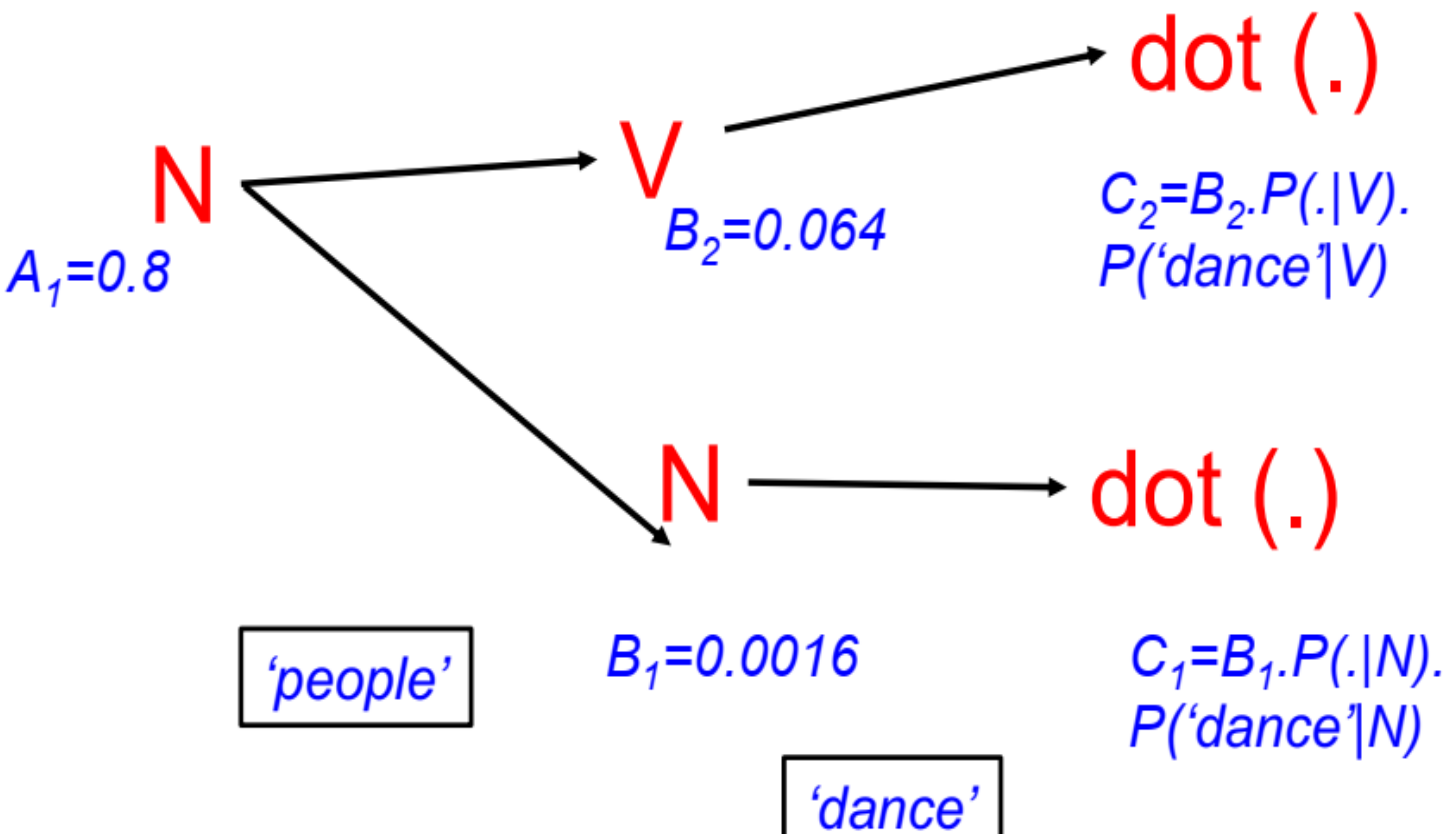
- B_1=0.8.0.2.0.01=0.0016 (approx.)
- B_2=0.8.0.8.0.01=0.064 (approx.)
- B_3=0.2.0.6.0.001=0.00012
- B_4=0.2.0.4.0.001=0.00008

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.

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

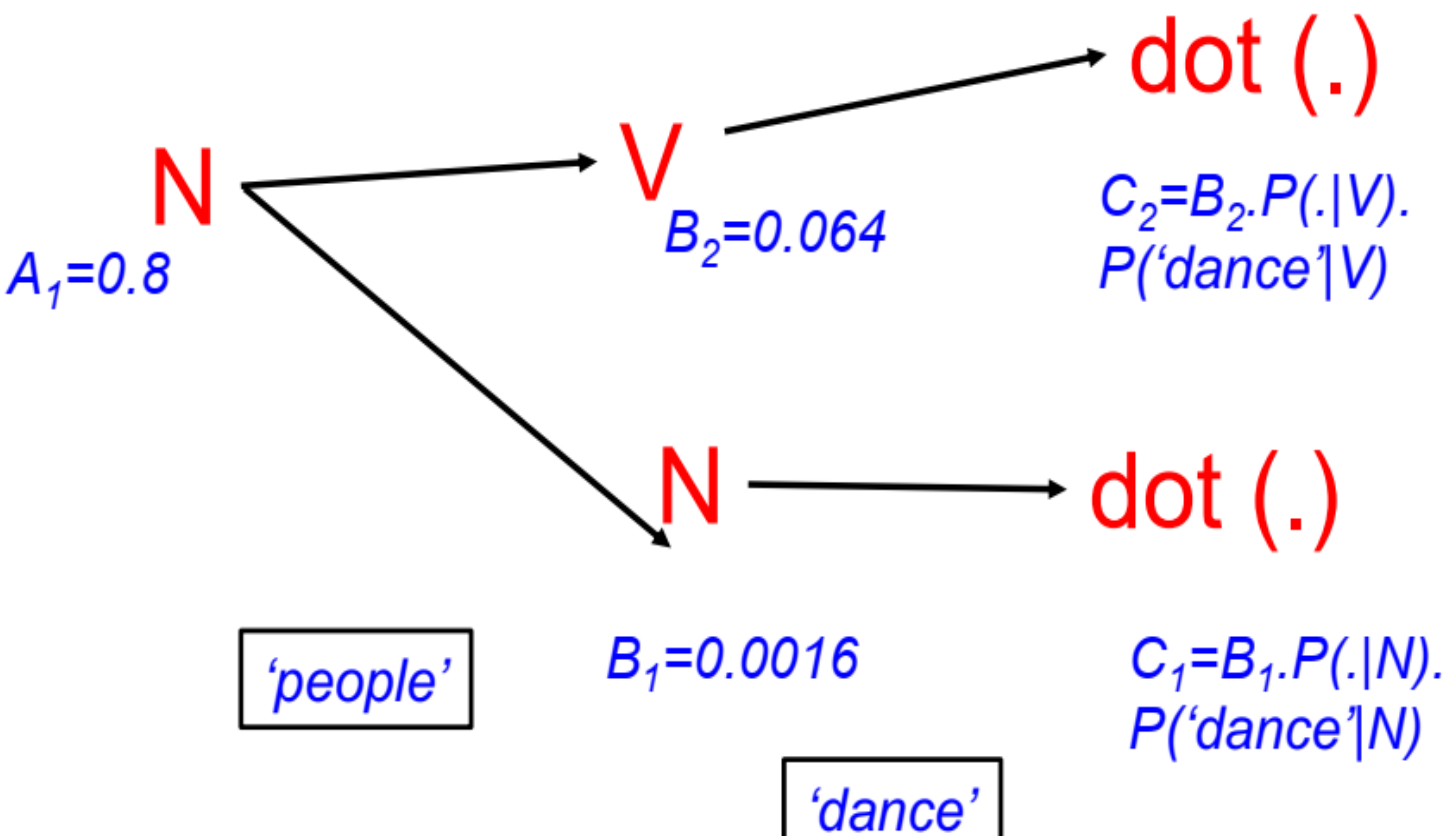
$$P(\text{‘dance’}|N)=0.001$$

$$P(\text{‘dance’}|V)=0.01$$

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

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

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

→ feature of a position, not feature of a word.
↳ word also looked up as a feature at that position.

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

→ Normalizing qty

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.

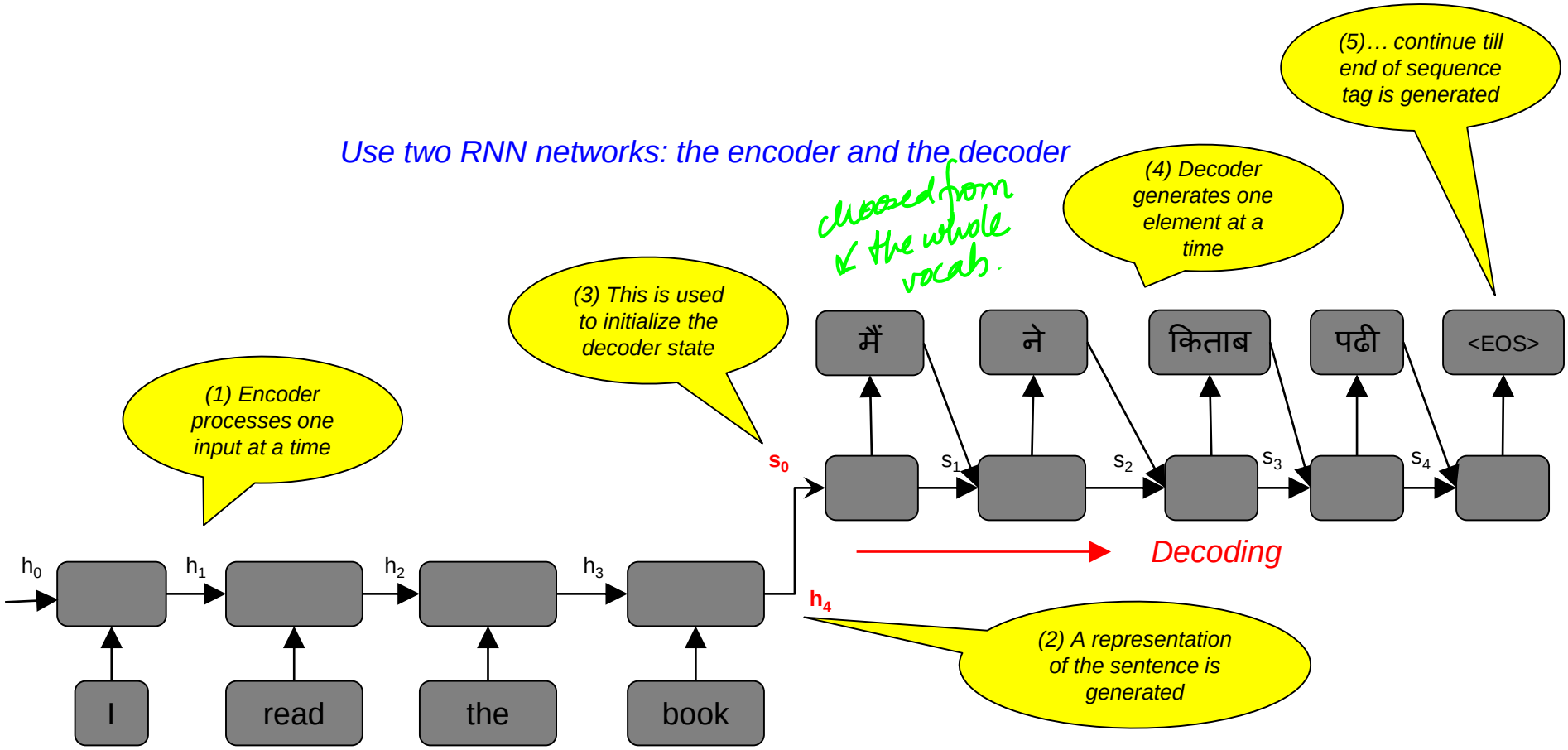
How to fix the beam width (2/2)

- 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



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!

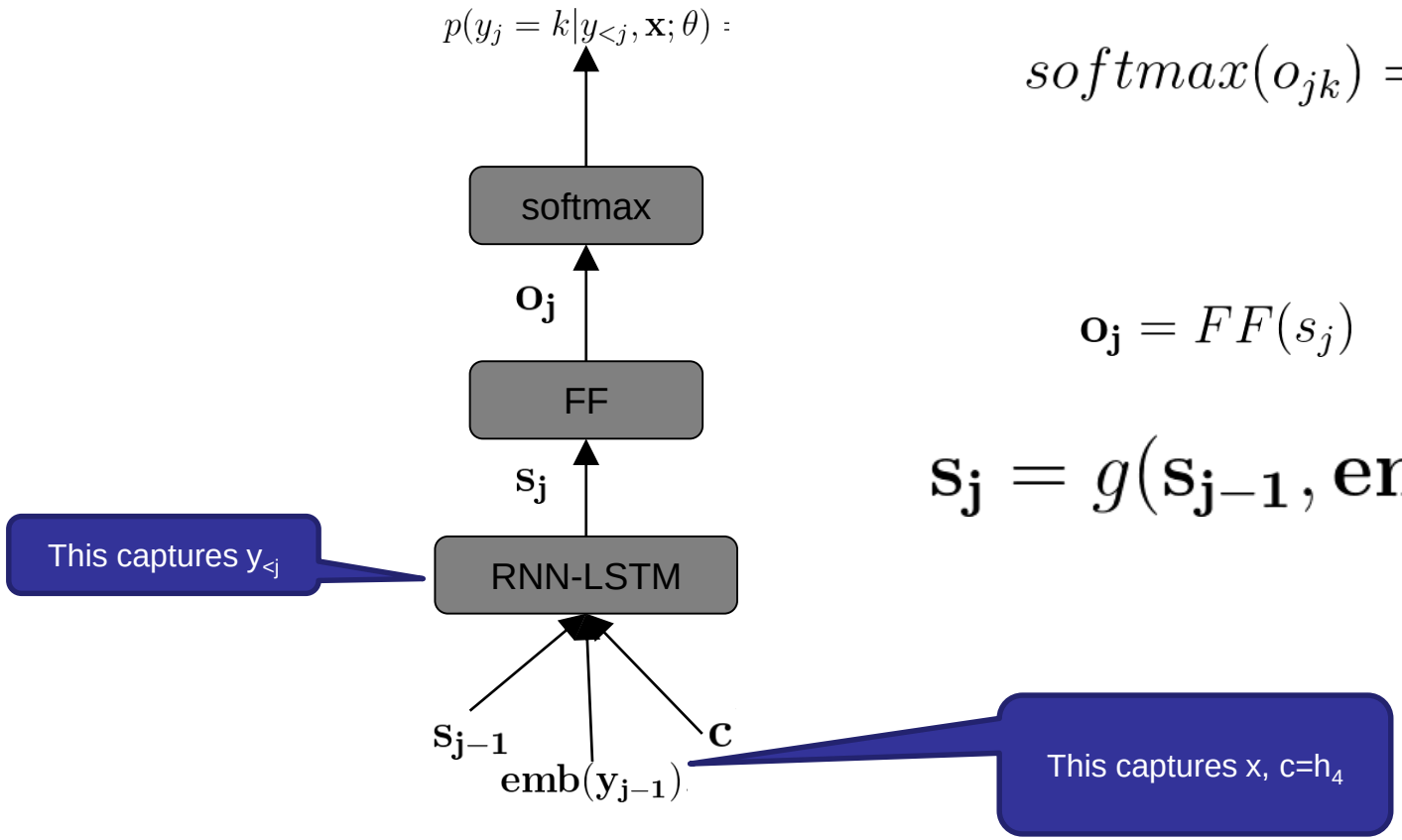
- Table look-up is equivalent to mapping
- Any form of search, is computing a mapping continuously, including the neural networks

Structural AI vs Functional AI

Structural AI	Functional AI
• Concerned with understanding the anatomy of a system	• Concerned with understanding the behaviour 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?



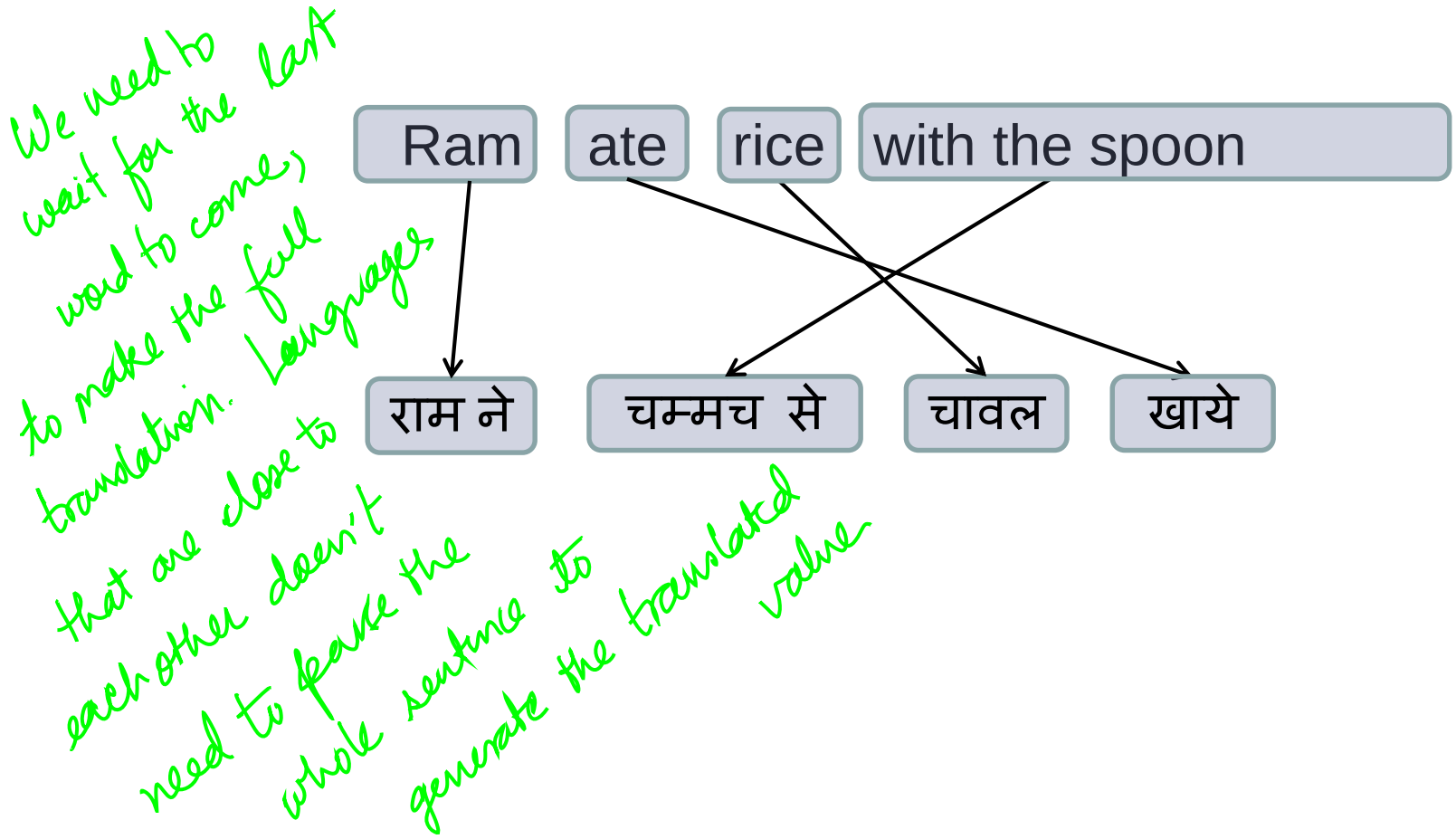
$$softmax(o_{jk}) = \frac{\exp(o_{jk})}{\sum_{m=0}^{m=T} \exp(o_{jm})}$$

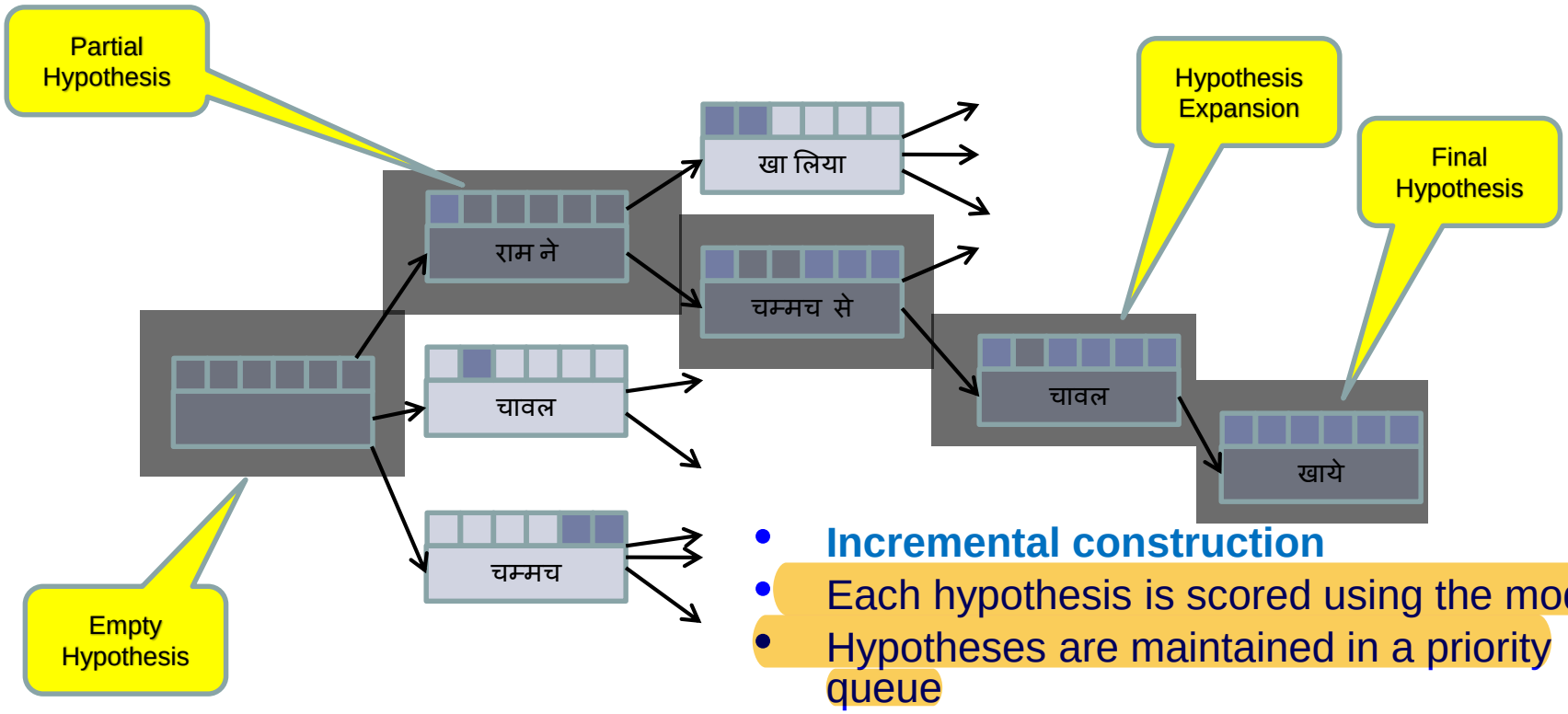
$$\mathbf{o_j} = FF(s_j)$$

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

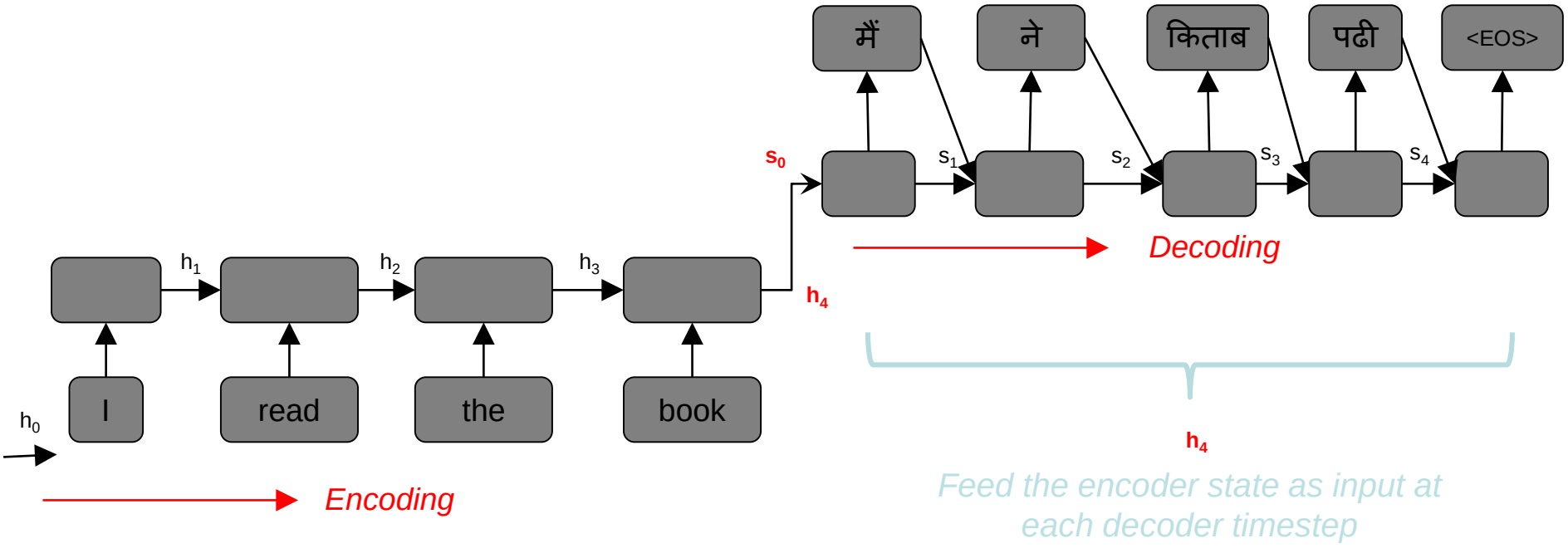
Decoding

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

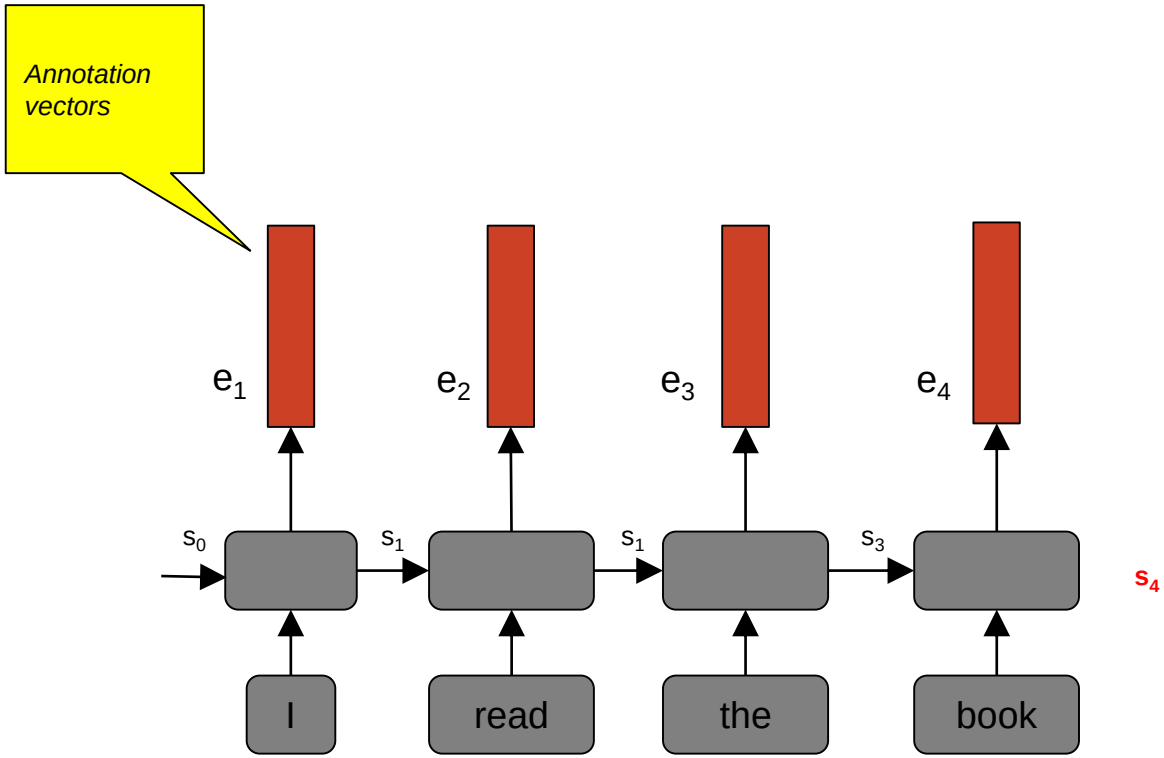

मैं
ने
किताब
पढी
<EOS>
s0
s1
s2
s3
s4
h1
h2
h3
h4
h0
I
read
the
book
Decoding
Encoding
h4
Feed the encoder state as input at each decoder timestep

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/