Precept 3: HMMs, Viterbi, RNNs

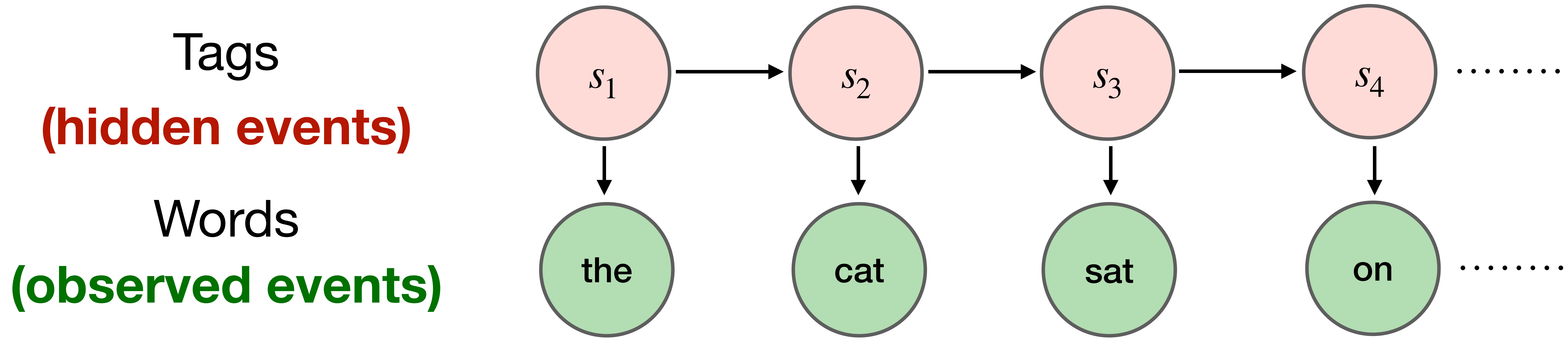
COS 484

Lucy He (slides adapted from Tyler Zhu and Colin Wang)

Today’s Plan

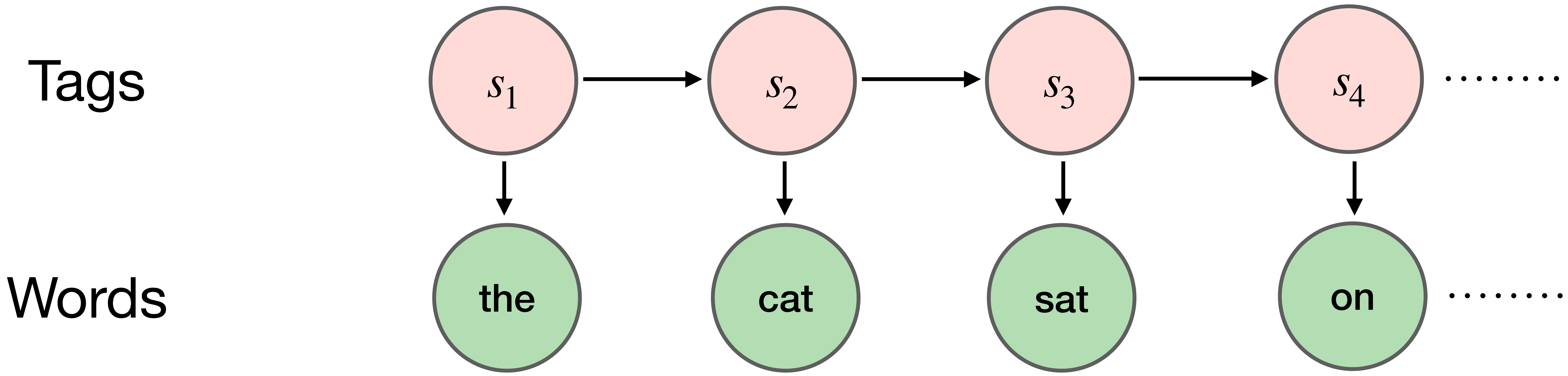
1. Hidden Markov Models
2. Viterbi
3. RNNs

Hidden Markov Model (HMM)



- We don't normally see sequences of POS tags in text
- However, we do observe the words!
- The HMM allows us to *jointly reason* over both **hidden** and **observed** events.
    - Assume that each position has a tag that generates a word

HMMs: Assumptions



What are the key assumptions?

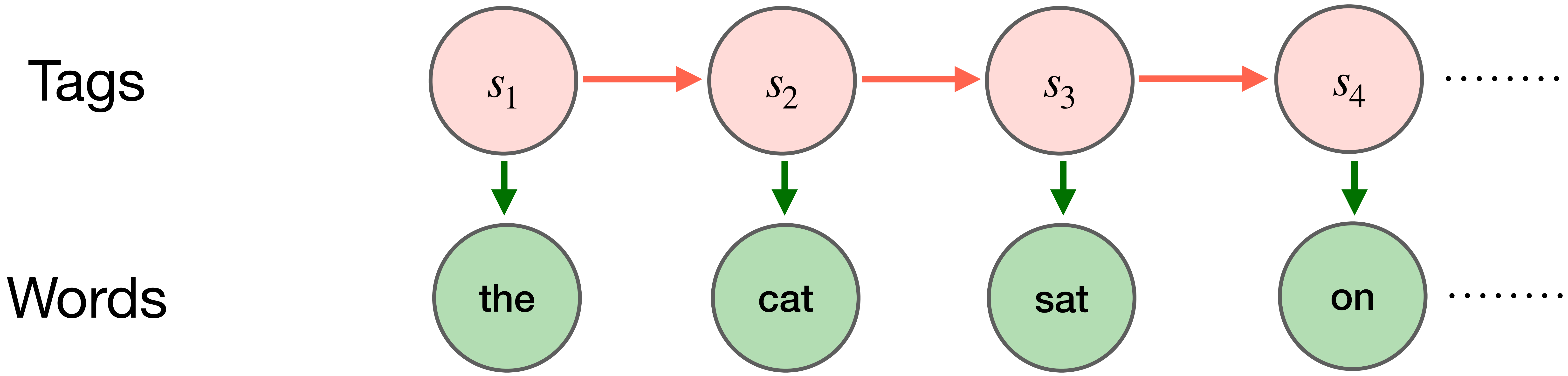
1. Markov assumption:

$$P(s_t \mid s_1, \ldots, s_{t-1}) \approx P(s_t \mid s_{t-1})$$

2. Output independence:

$$P(o_t \mid s_1, \ldots, s_t) \approx P(o_t \mid s_t)$$

HMMs: Assumptions



What are the key assumptions?

1. Markov assumption:

$$P(s_t \mid s_1, \ldots, s_{t-1}) \approx P(s_t \mid s_{t-1})$$

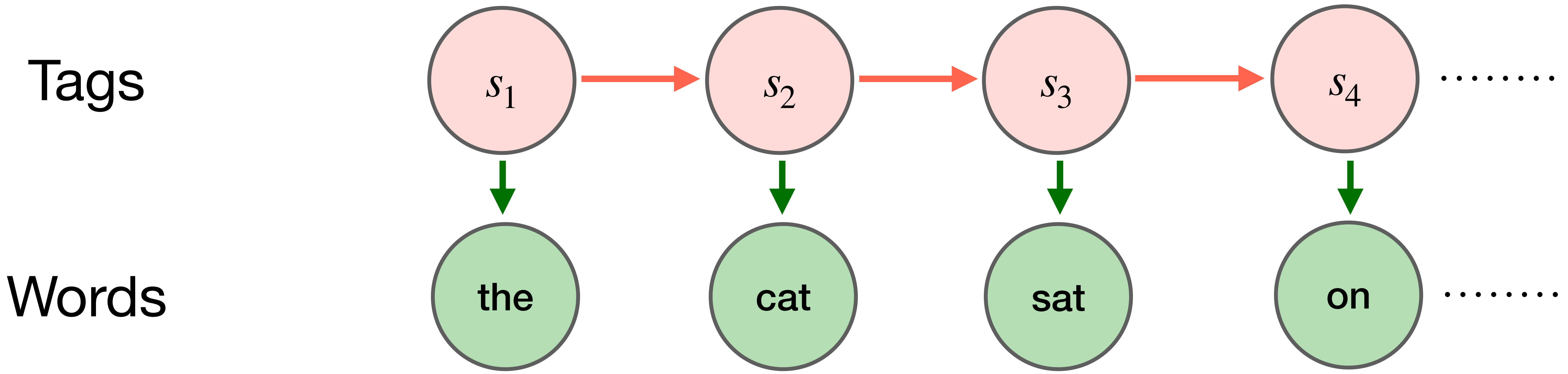
Transition probabilities

2. Output independence:

$$P(o_t \mid s_1, \ldots, s_t) \approx P(o_t \mid s_t)$$

Emission probabilities

HMMs: Training



What do we train and how?

Transition probabilities

$$P(s_t \mid s_1, \ldots, s_{t-1}) \approx P(s_t \mid s_{t-1})$$

	DT	NN	VB
∅	0.5	0.3	0.2
DT	0.1	0.5	0.4
NN	0.2	0.3	0.5
VB	0.4	0.3	0.3

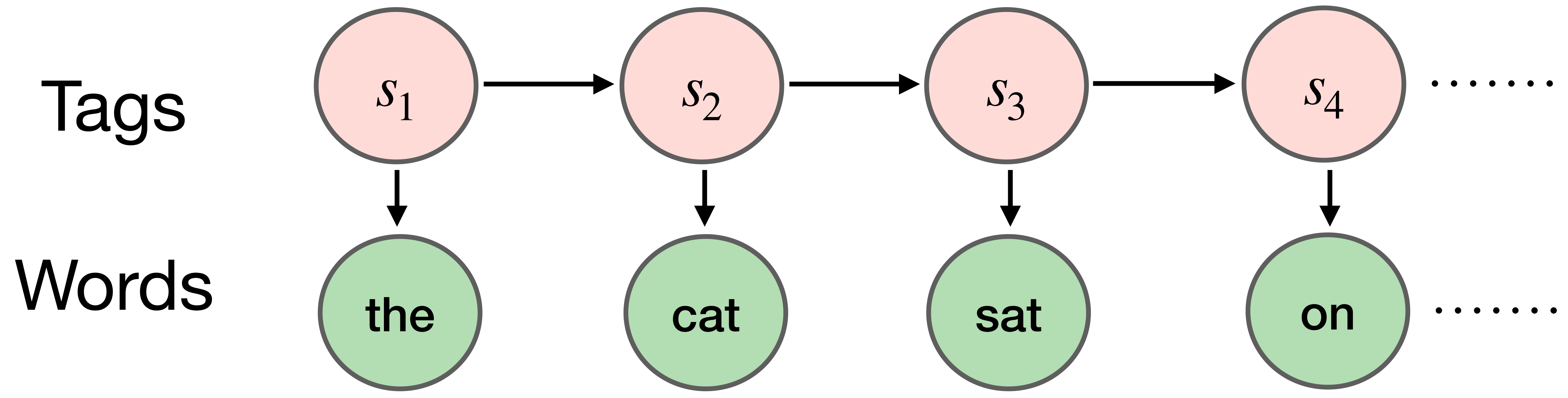
From training data!

Emission probabilities

$$P(o_t \mid s_1, \ldots, s_t) \approx P(o_t \mid s_t)$$

	the	cat	runs
DT	0.4	0.5	0.1
NN	0.5	0.4	0.1
VB	0.2	0.3	0.5

HMMs: Inference



Task: Find the most probable sequence of states $S = s_1, s_2, \ldots, s_n$ given the observations $O = o_1, o_2, \ldots, o_n$

$$\hat{S} = \arg\max_S P(S \mid O) = \arg\max_S \frac{P(O \mid S)P(S)}{P(O)} \quad \text{[Bayes' rule]}$$

$$= \arg\max_S P(O \mid S)P(S) \quad [P(O) \text{ doesn't depend on } S!]$$

$$= \arg\max_{s_1, s_2, \ldots s_n} \prod_{i=1}^{n} P(o_i \mid s_i)P(s_i \mid s_{i-1}) \quad \text{[Markov assumption]}$$

How can we maximize this?
Search over all state sequences?

[SP23 Midterm] Problem 5: Lie Detection

You are building a lie detector taking as input a stream of recorded behaviors:

- $x_t \in \{a, b\}$, i.e., face touching (a) or blinking (b)

Detector at every moment then predicts one of four labels:

- $y_t \in \{N, U, L, H\}$, i.e., Neutral (N), Unclear (U), Lying (L), and Honest (H)

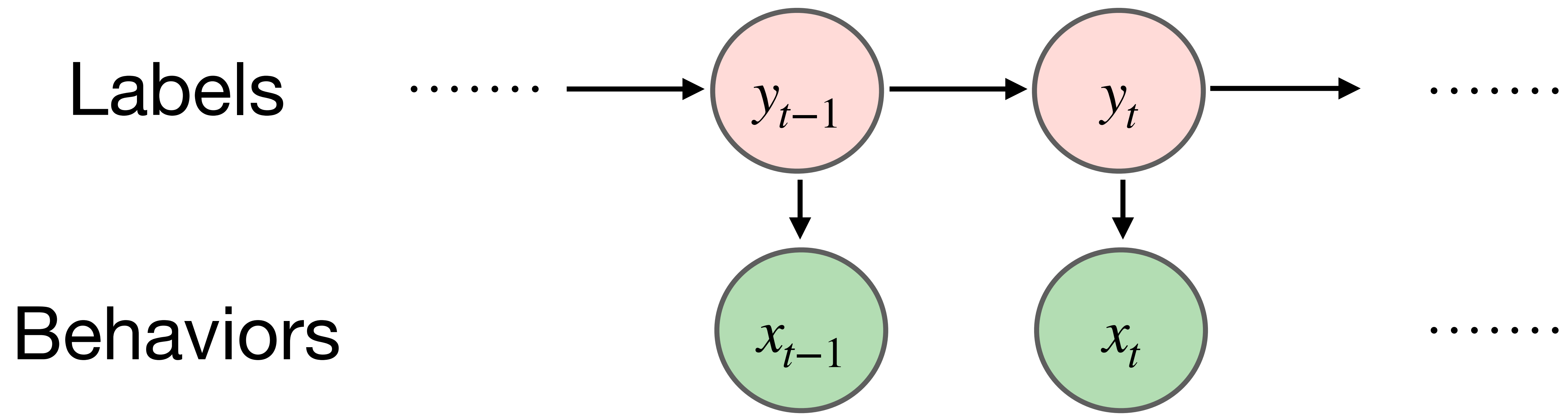
Dataset is triplets (y_{t-1}, y_t, x_t): 9x of (N, L, a), 9x of (U, L, b), 1x of (N, H, b)

- Labels y_t come from body language experts' annotation

Q1: You use the above data to build an HMM. What is $P(x_t = b \,|\, y_{t-1} = N)$?

[SP23 Midterm] Problem 5: Lie Detection

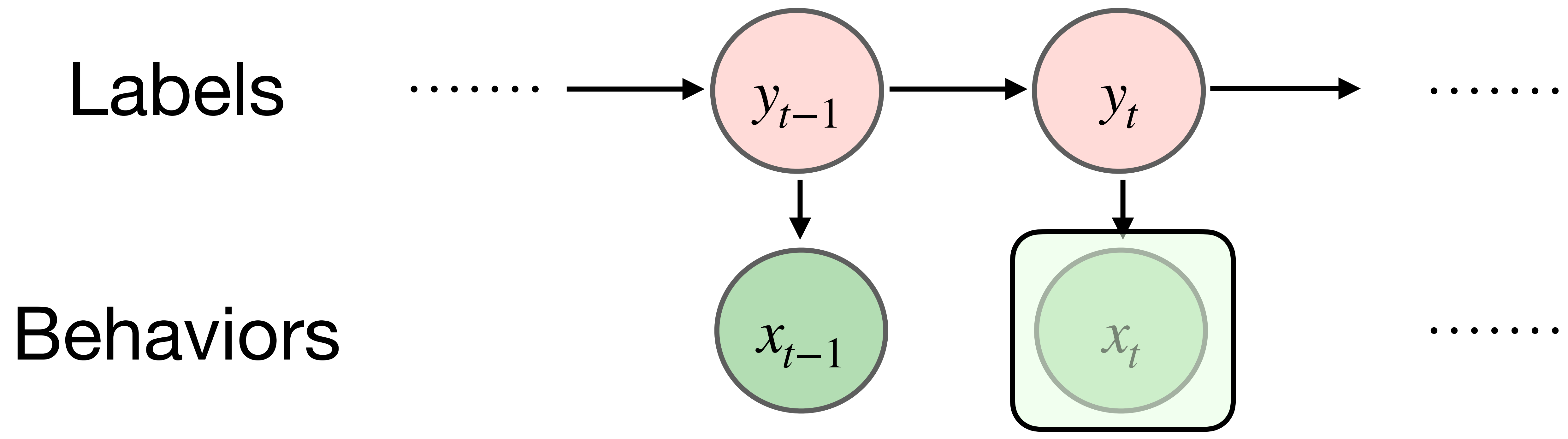
Q1: You use the above data to build an HMM. What is $P(x_t = b \mid y_{t-1} = N)$?



Simplified model of the HMM we care about.

[SP23 Midterm] Problem 5: Lie Detection

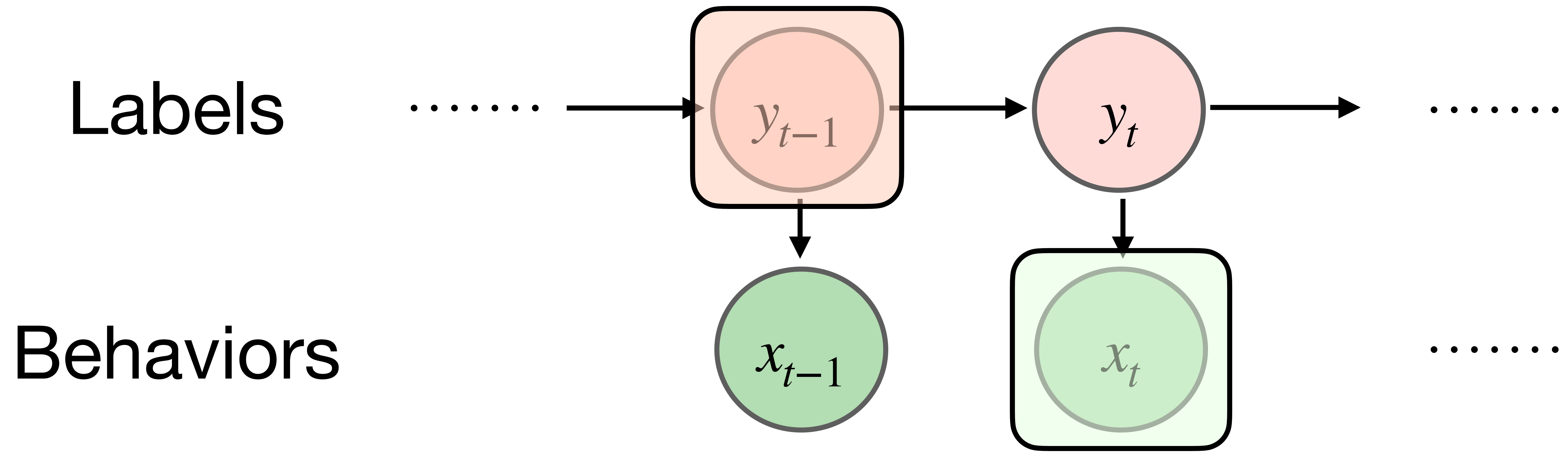
Q1: You use the above data to build an HMM. What is $P(x_t = b \,|\, y_{t-1} = N)$?



Simplified model of the HMM we care about.

[SP23 Midterm] Problem 5: Lie Detection

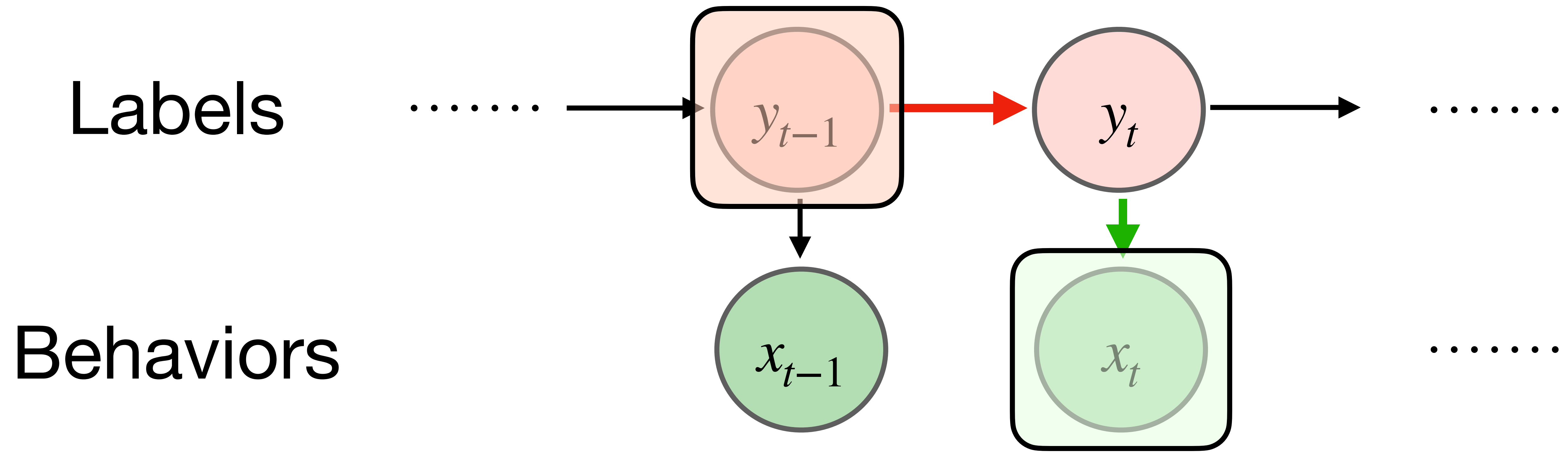
Q1: You use the above data to build an HMM. What is $P(x_t = b \mid y_{t-1} = N)$?



Simplified model of the HMM we care about.

[SP23 Midterm] Problem 5: Lie Detection

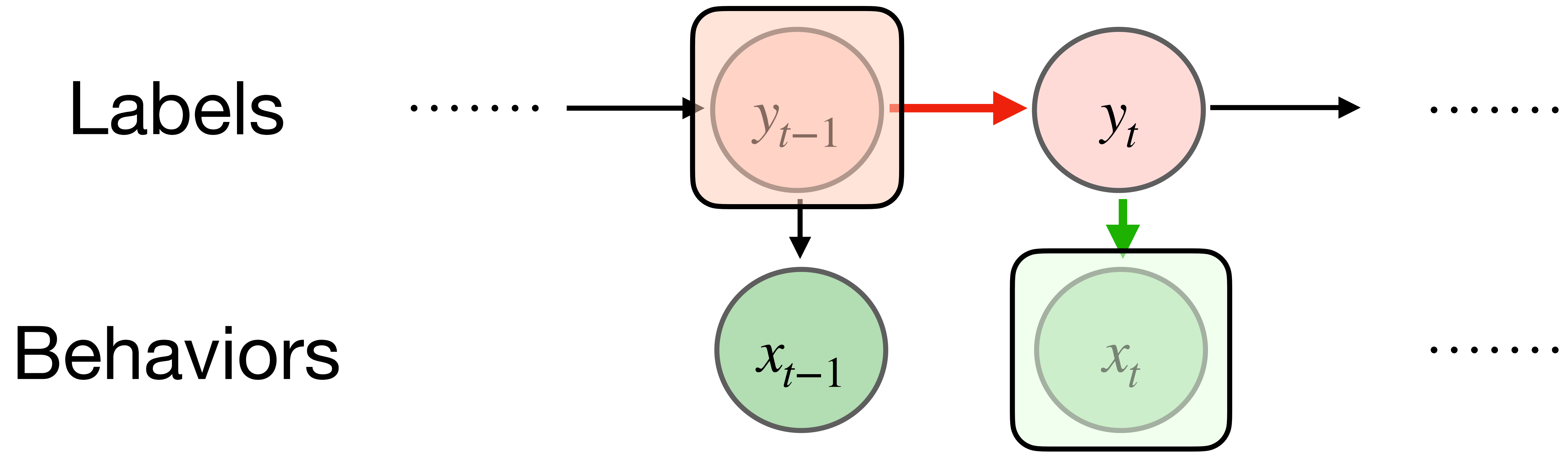
Q1: You use the above data to build an HMM. What is $P(x_t = b \mid y_{t-1} = N)$?



Simplified model of the HMM we care about.

[SP23 Midterm] Problem 5: Lie Detection

Q1: You use the above data to build an HMM. What is $P(x_t = b \,|\, y_{t-1} = N)$?

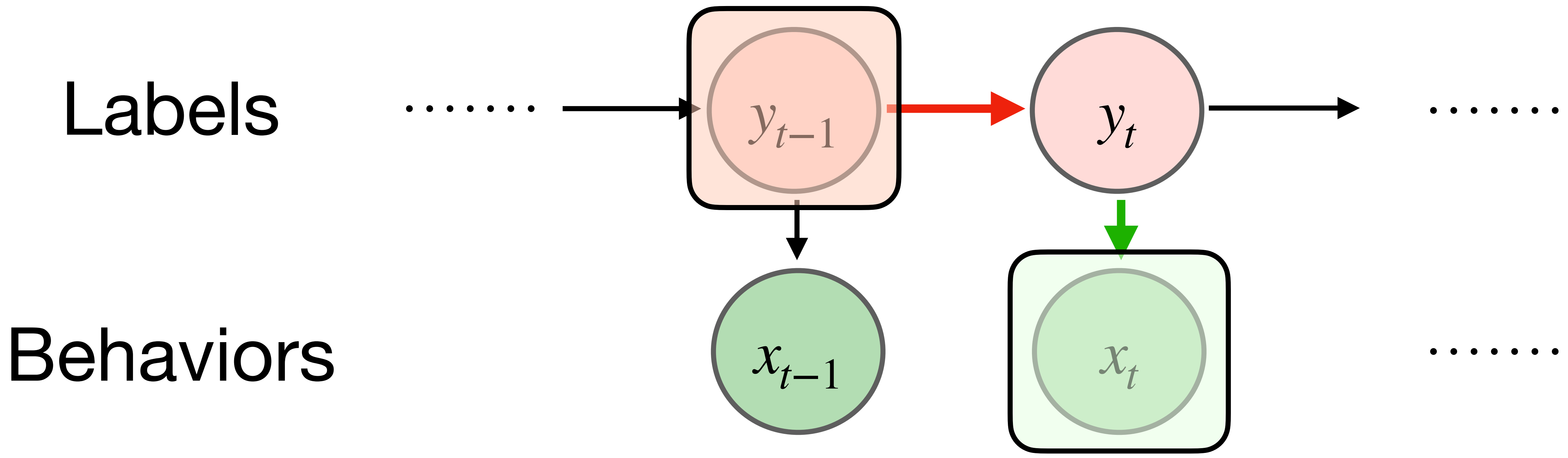


Simplified model of the HMM we care about.

We need to condition over y_t using our emission and transition probabilities.

[SP23 Midterm] Problem 5: Lie Detection

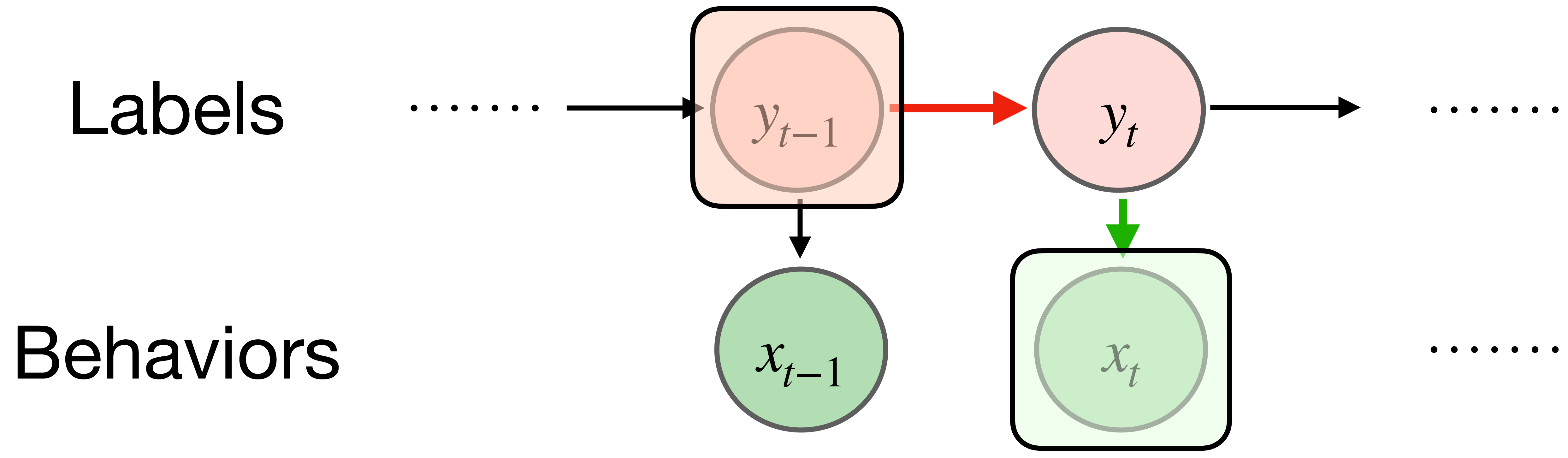
Q1: You use the above data to build an HMM. What is $P(x_t = b \mid y_{t-1} = N)$?



$$P(x_t = b \mid y_{t-1} = N) = \sum_{y_t} P(x_t = b, y_t \mid y_{t-1} = N)$$

[SP23 Midterm] Problem 5: Lie Detection

Q1: You use the above data to build an HMM. What is $P(x_t = b \mid y_{t-1} = N)$?

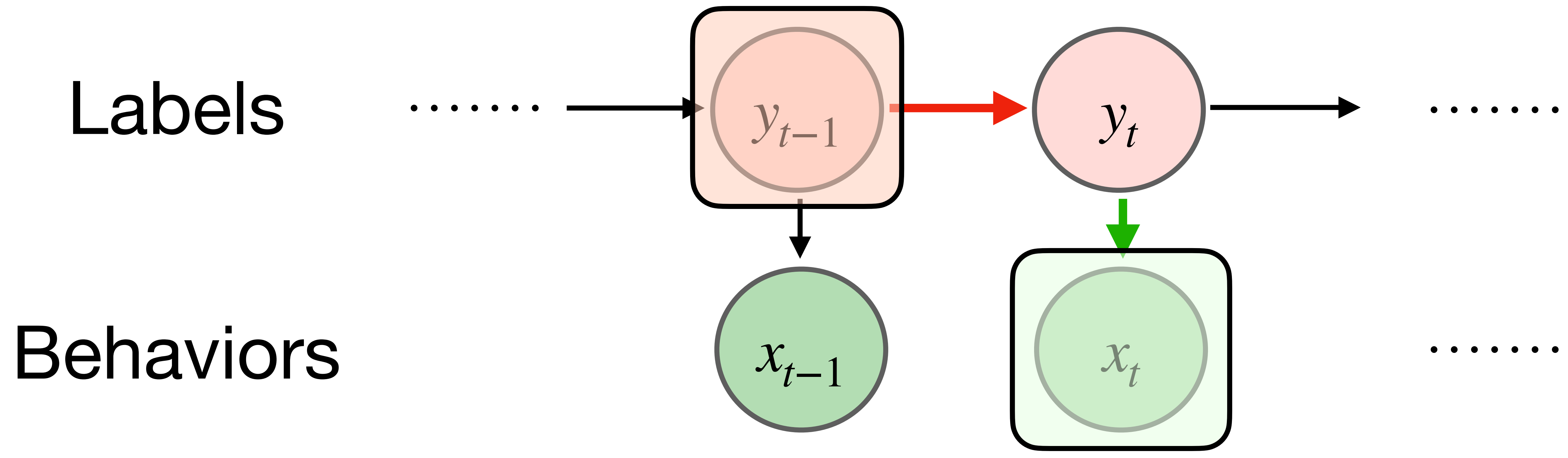


$$P(x_t = b \mid y_{t-1} = N) = \sum_{y_t} P(x_t = b, y_t \mid y_{t-1} = N)$$

$$= \sum_{y_t} \underbrace{P(x_t = b \mid y_t)}_{\text{emission}} \underbrace{P(y_t \mid y_{t-1} = N)}_{\text{transition}}$$

[SP23 Midterm] Problem 5: Lie Detection

Q1: You use the above data to build an HMM. What is $P(x_t = b \,|\, y_{t-1} = N)$?



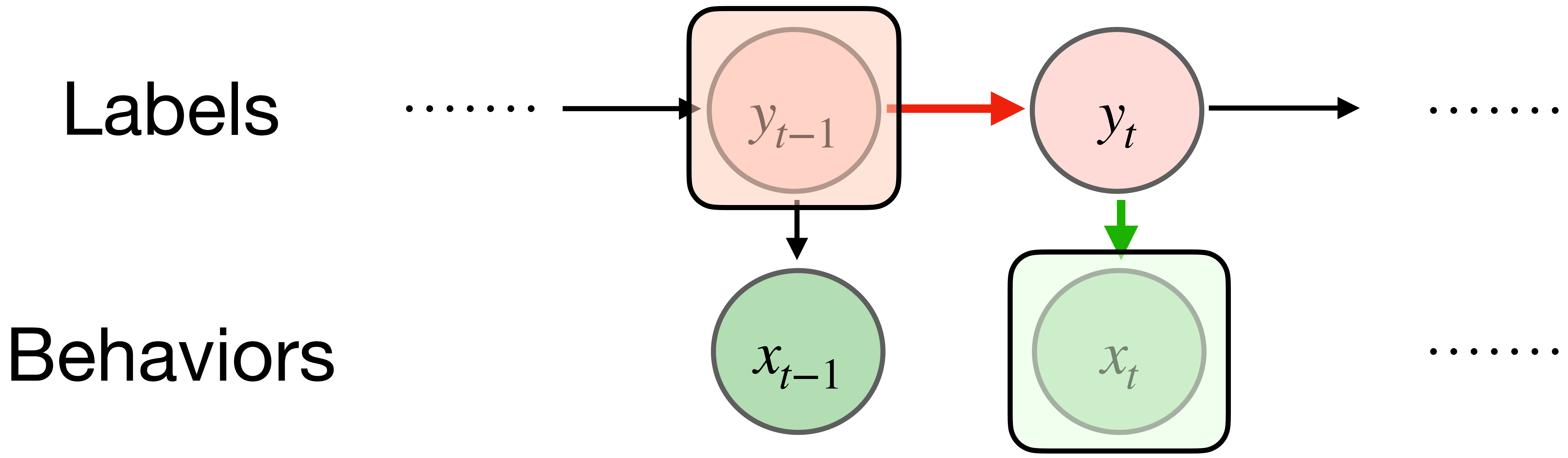
$$P(x_t = b \,|\, y_{t-1} = N) = \sum_{y_t} P(x_t = b, y_t \,|\, y_{t-1} = N)$$

$$= \sum_{y_t} P(x_t = b \,|\, y_t) P(y_t \,|\, y_{t-1} = N)$$

$$= P(b \,|\, L) P(L \,|\, N) + P(b \,|\, H) P(H \,|\, N)$$

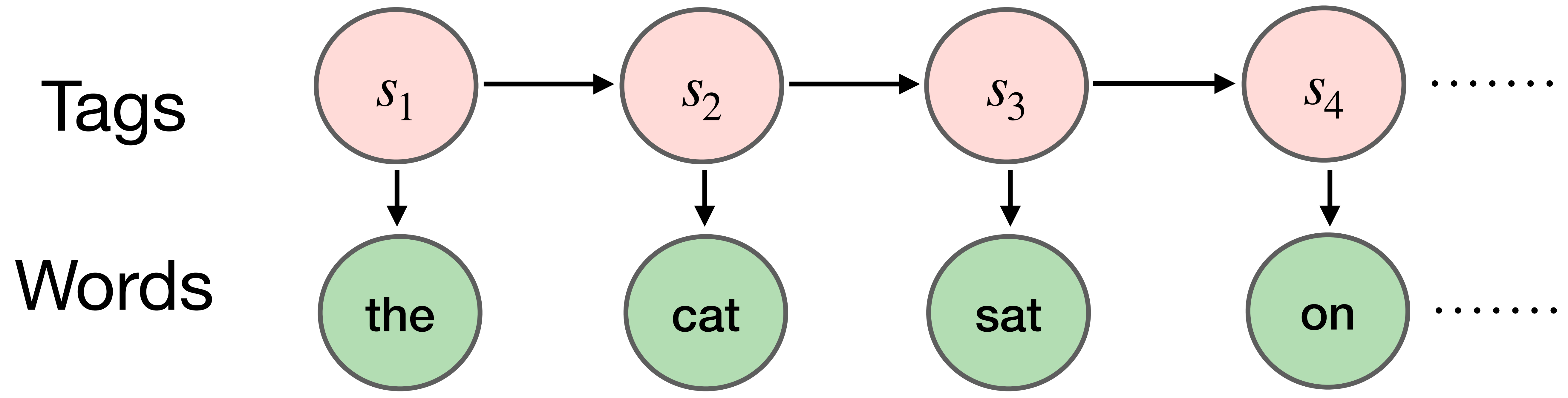
[SP23 Midterm] Problem 5: Lie Detection

Q1: You use the above data to build an HMM. What is $P(x_t = b \mid y_{t-1} = N)$?



$$
\begin{aligned}
P(x_t = b \mid y_{t-1} = N) &= \sum_{y_t} P(x_t = b, y_t \mid y_{t-1} = N) \\
&= \sum_{y_t} P(x_t = b \mid y_t) P(y_t \mid y_{t-1} = N) \\
&= P(b \mid L) P(L \mid N) + P(b \mid H) P(H \mid N) \\
&= \frac{1}{2} \times \frac{9}{10} + 1 \times \frac{1}{10} = \frac{11}{20}
\end{aligned}
$$

HMMs: Efficient Inference

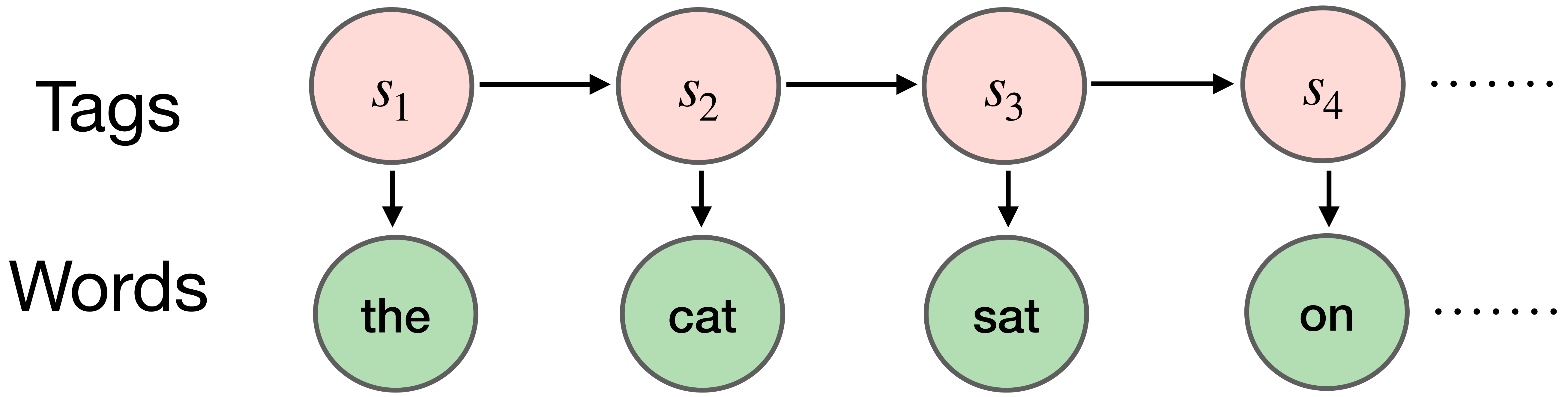


Task: Find the most probable sequence of states $S = s_1, s_2, \ldots, s_n$ given the observations $O = o_1, o_2, \ldots, o_n$

$$\hat{S} = \arg\max_{S} P(S \mid O) = \arg \max_{s_1, s_2, \ldots s_n} \prod_{i=1}^{n} P(o_i \mid s_i) P(s_i \mid s_{i-1})$$

How can we maximize this?
Search over all state sequences?

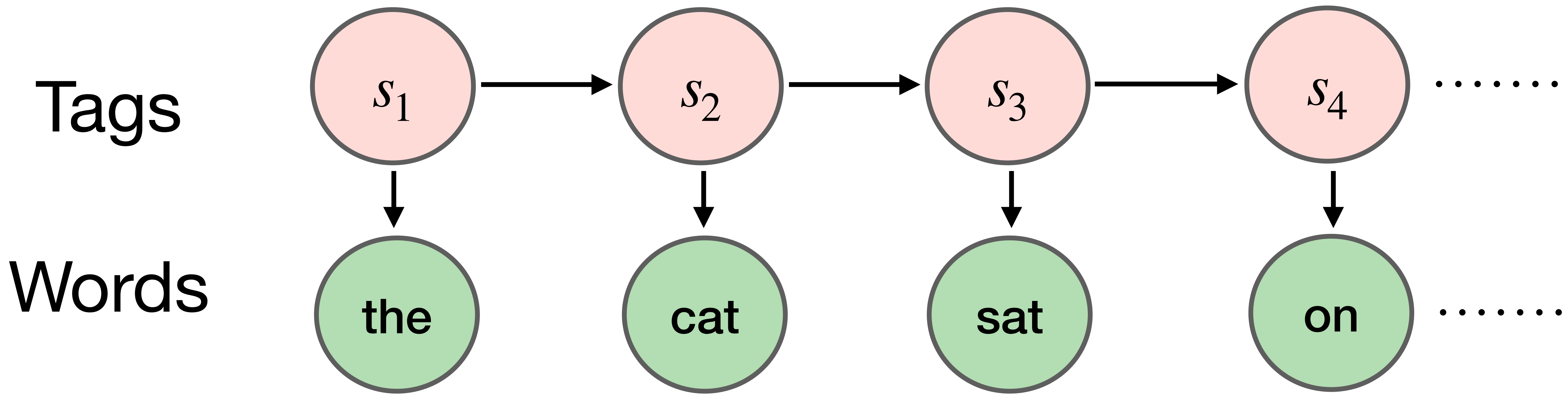
HMMs: Efficient Inference



Task: Find the most probable sequence of states $S = s_1, s_2, \ldots, s_n$ given the observations $O = o_1, o_2, \ldots, o_n$

$$\hat{S} = \arg\max_{S} P(S \mid O) = \arg \max_{s_1, s_2, \ldots s_n} \prod_{i=1}^{n} P(o_i \mid s_i) P(s_i \mid s_{i-1})$$

HMMs: Efficient Inference

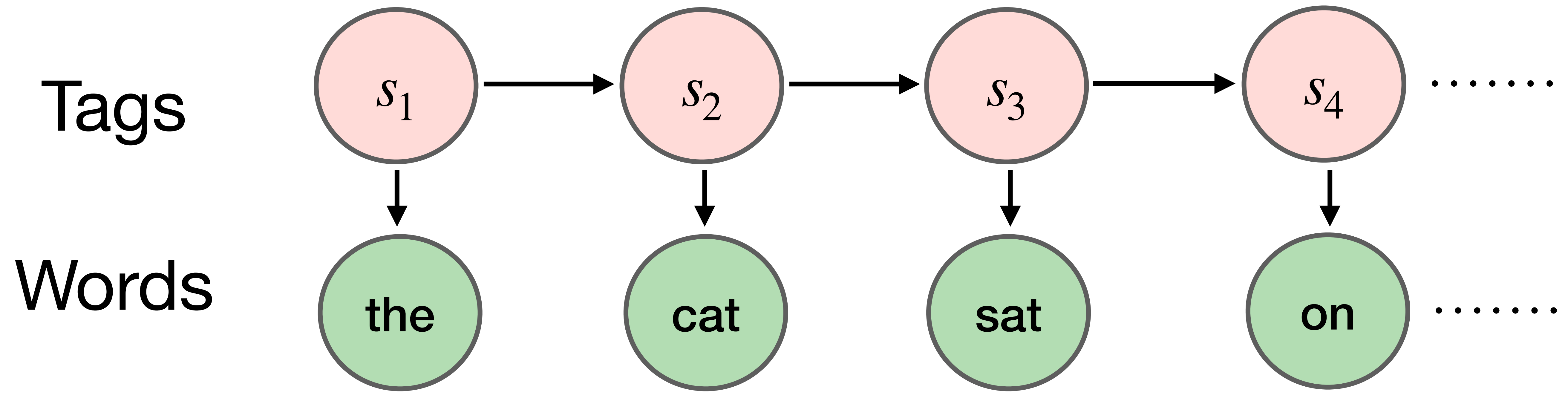


Task: Find the most probable sequence of states $S = s_1, s_2, \ldots, s_n$ given the observations $O = o_1, o_2, \ldots, o_n$

$$\hat{S} = \arg\max_{S} P(S \mid O) = \arg\max_{s_1, s_2, \ldots s_n} \prod_{i=1}^{n} P(o_i \mid s_i) P(s_i \mid s_{i-1})$$

Find the state sequence that gives the biggest product of emission probability × transition probability at every step.

HMMs: Efficient Inference

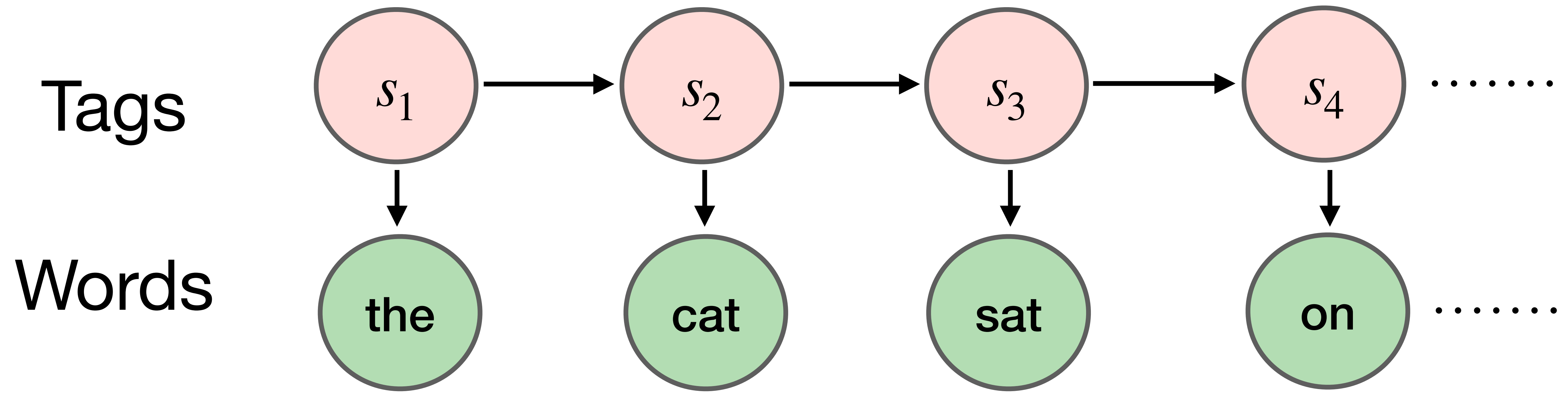


Task: Find the most probable sequence of states $S = s_1, s_2, \ldots, s_n$ given the observations $O = o_1, o_2, \ldots, o_n$

$$\hat{S} = \arg\max_{S} P(S \mid O) = \arg \max_{s_1, s_2, \ldots s_n} \prod_{i=1}^{n} P(o_i \mid s_i) P(s_i \mid s_{i-1})$$

The best path to time i ending in state s_i must go through the best path to time i–1 ending in some previous state.

HMMs: Efficient Inference

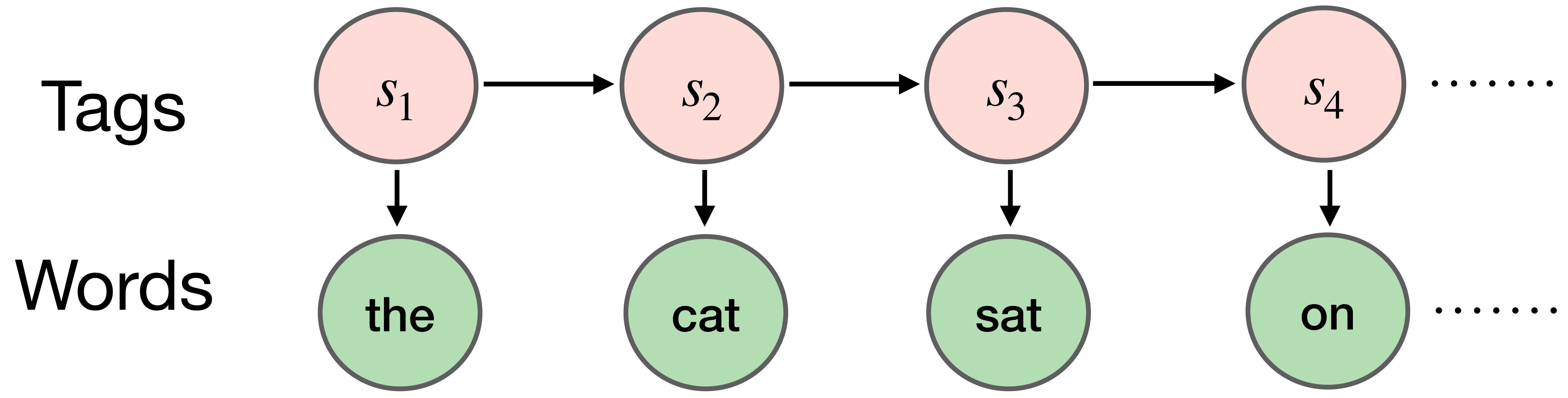


Task: Find the most probable sequence of states $S = s_1, s_2, \ldots, s_n$ given the observations $O = o_1, o_2, \ldots, o_n$

$$\hat{S} = \arg\max_{S} P(S \mid O) = \arg \max_{s_1, s_2, \ldots s_n} \prod_{i=1}^{n} P(o_i \mid s_i) P(s_i \mid s_{i-1})$$

Makes dynamic programming possible, so we only need to keep the “best” for each previous time step to decode the next (instead of more history). Define score as the probability of the best path so far that ends in state s_i at time i

HMMs: Efficient Inference

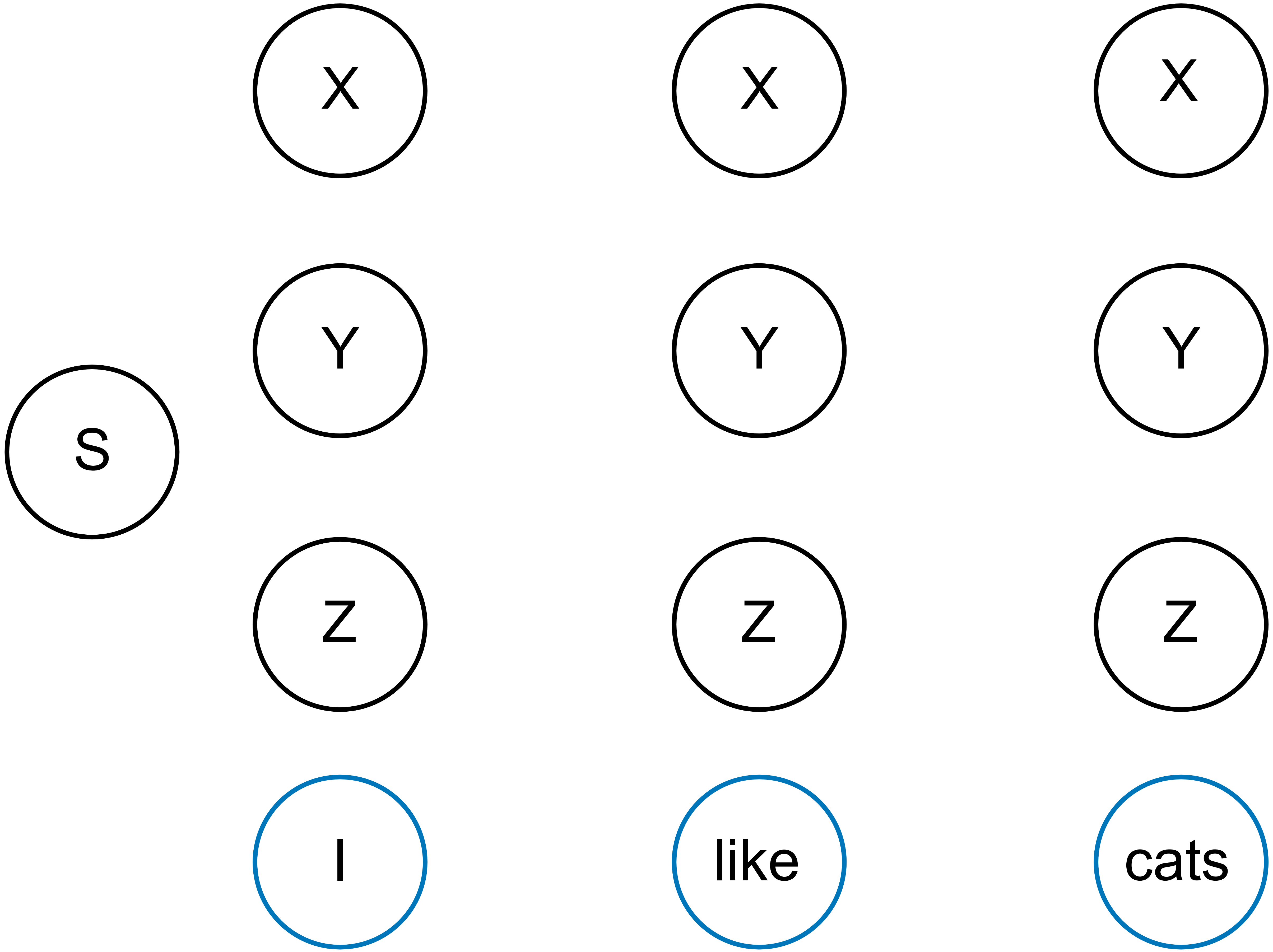


$$\text{score}[i][s_i] = P(o_i \mid s_i) \times \max_{s_{i-1}} \left(\text{score}[i-1][s_{i-1}] \times P(s_i \mid s_{i-1})\right)$$

Viterbi:

- When deciding the best state at time i, you don’t just look at how well it explains the current observation, you also include the best path leading up to it so far.
- LHS- “the probability of the most likely path that ends in state s_i at time I”.
- First term - See how well it explains the current observation.
- Second term- Look back and choose the one with the highest accumulated probability so far. The P(s_i|s_{i-1}) term captures how likely we move from previous state to this one.

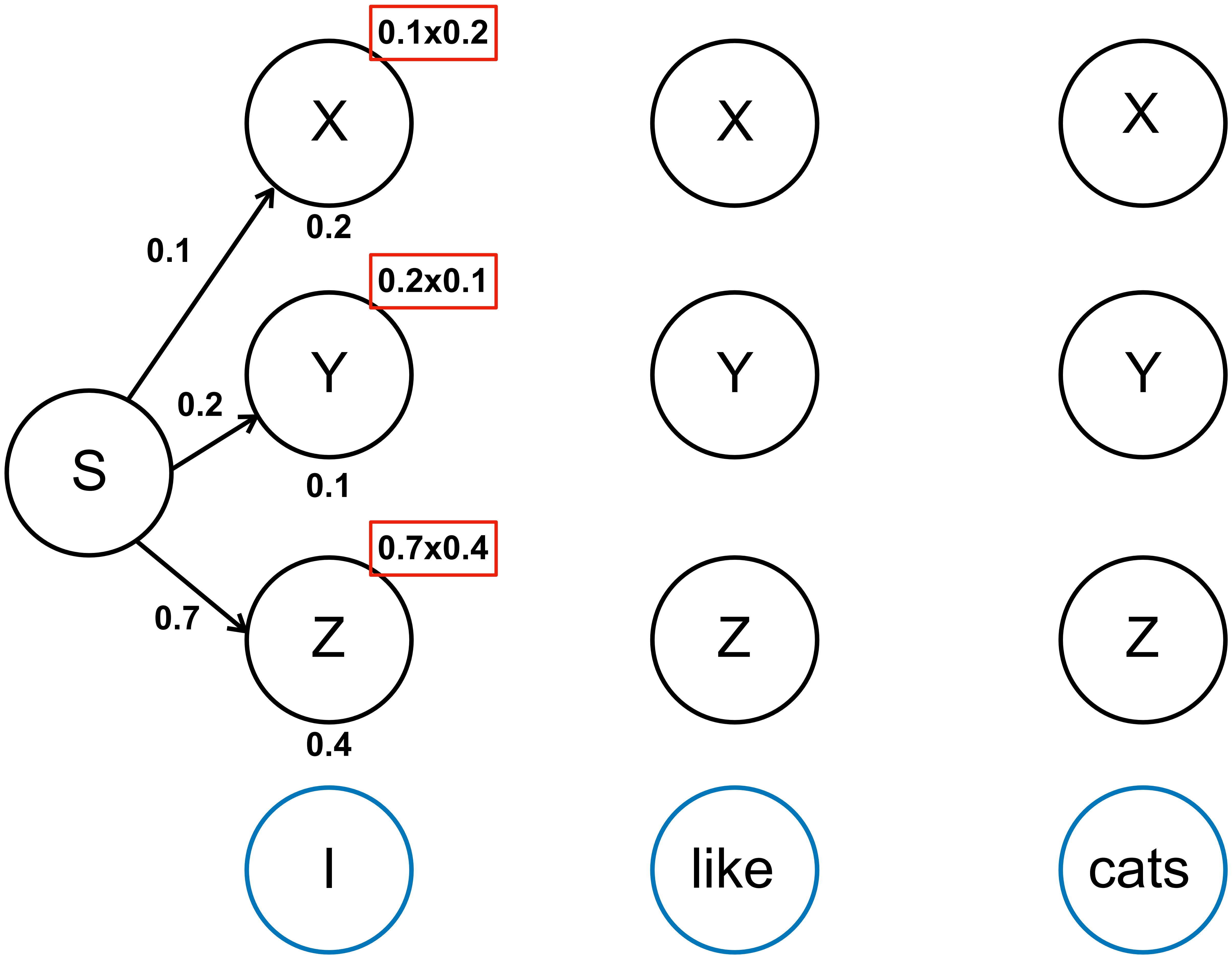
Viterbi Algorithm



	X	Y	Z
S	0.1	0.2	0.7
X	0.2	0.5	0.3
Y	0.4	0.4	0.2
Z	0.6	0.2	0.2

	I	like	cats
X	0.2	0.1	0.7
Y	0.1	0.8	0.1
Z	0.4	0.3	0.3

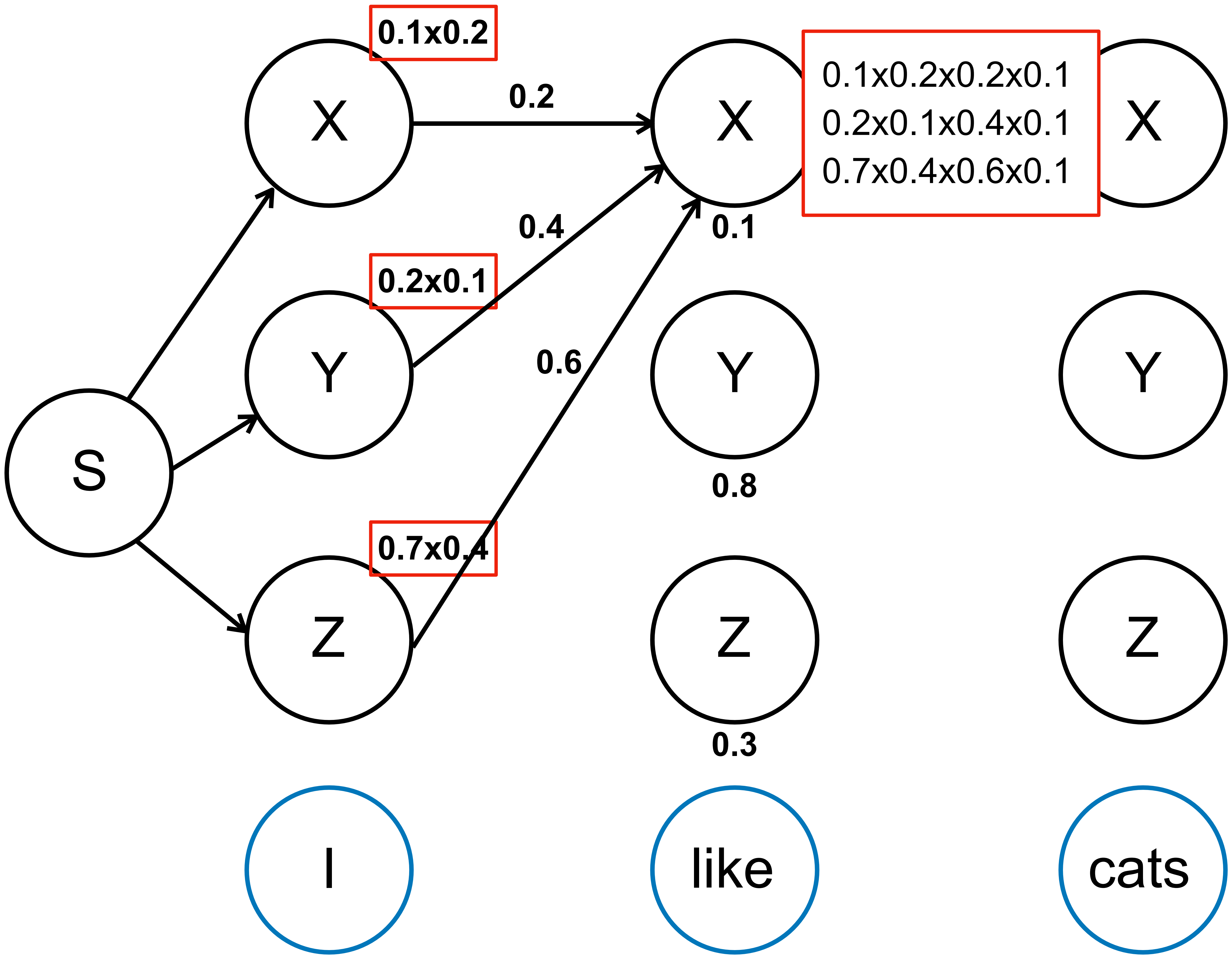
Viterbi Algorithm



	X	Y	Z
S	0.1	0.2	0.7
X	0.2	0.5	0.3
Y	0.4	0.4	0.2
Z	0.6	0.2	0.2

	I	like	cats
X	0.2	0.1	0.7
Y	0.1	0.8	0.1
Z	0.4	0.3	0.3

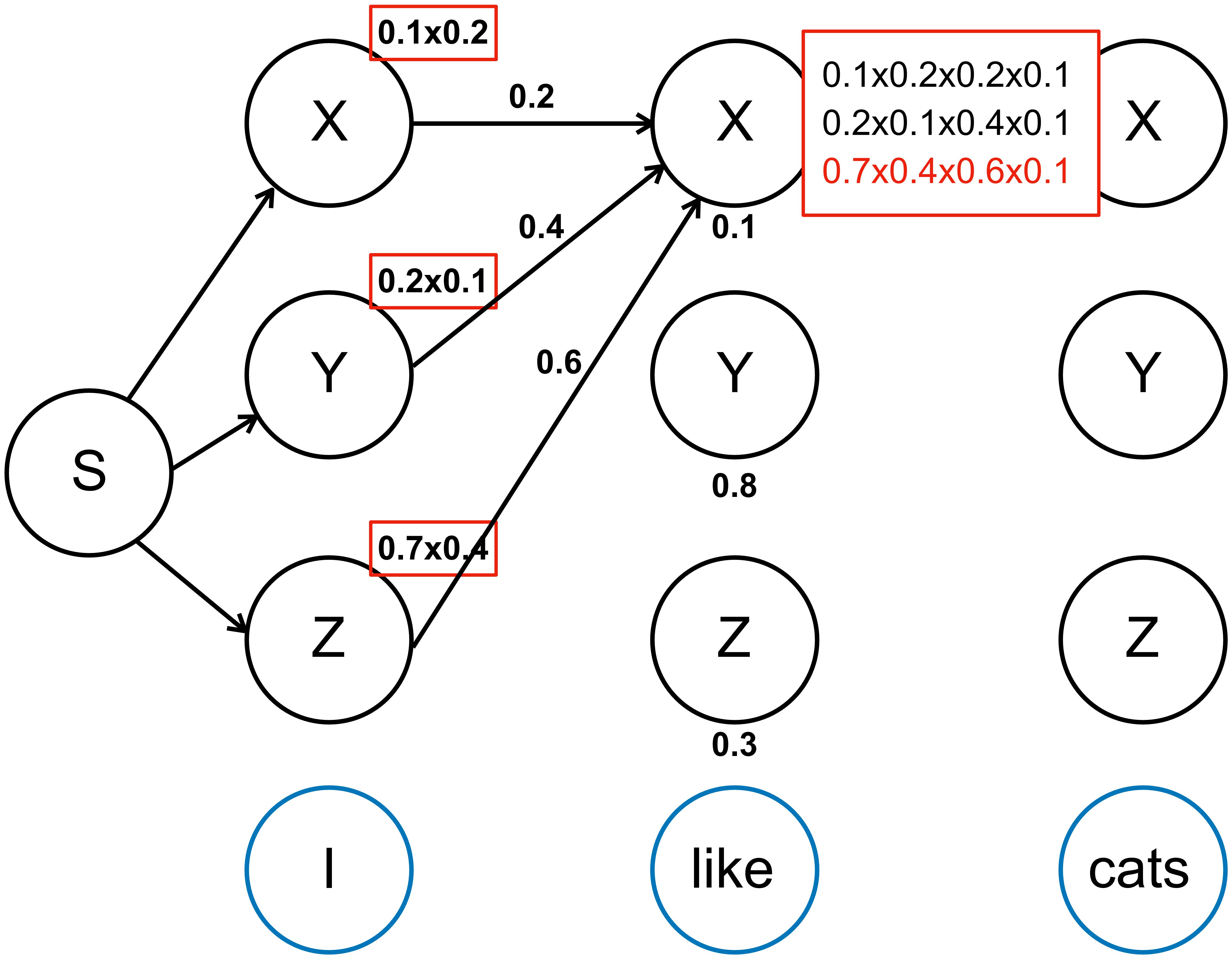
Viterbi Algorithm



	X	Y	Z
S	0.1	0.2	0.7
X	0.2	0.5	0.3
Y	0.4	0.4	0.2
Z	0.6	0.2	0.2

	I	like	cats
X	0.2	0.1	0.7
Y	0.1	0.8	0.1
Z	0.4	0.3	0.3

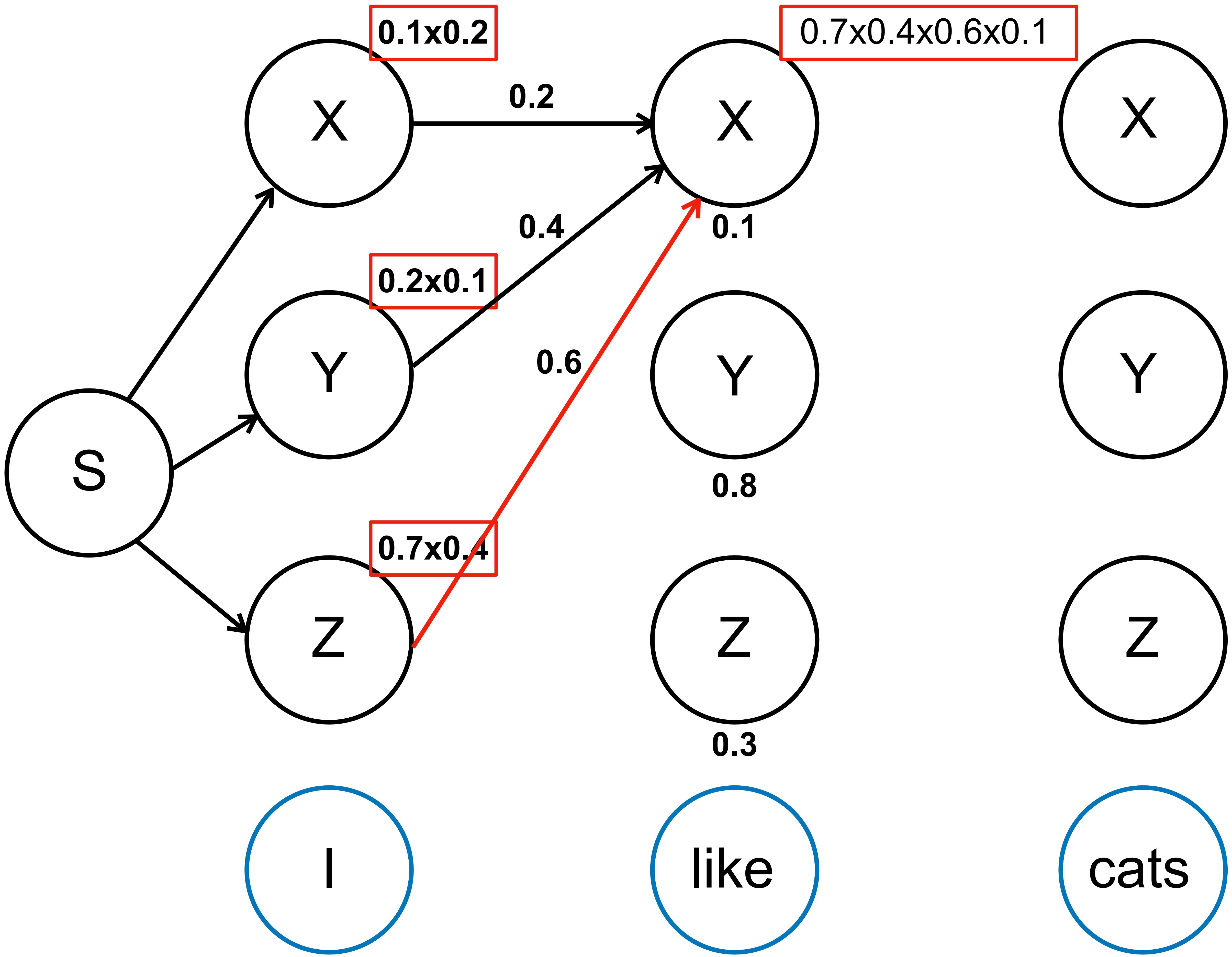
Viterbi Algorithm



	X	Y	Z
S	0.1	0.2	0.7
X	0.2	0.5	0.3
Y	0.4	0.4	0.2
Z	0.6	0.2	0.2

	I	like	cats
X	0.2	0.1	0.7
Y	0.1	0.8	0.1
Z	0.4	0.3	0.3

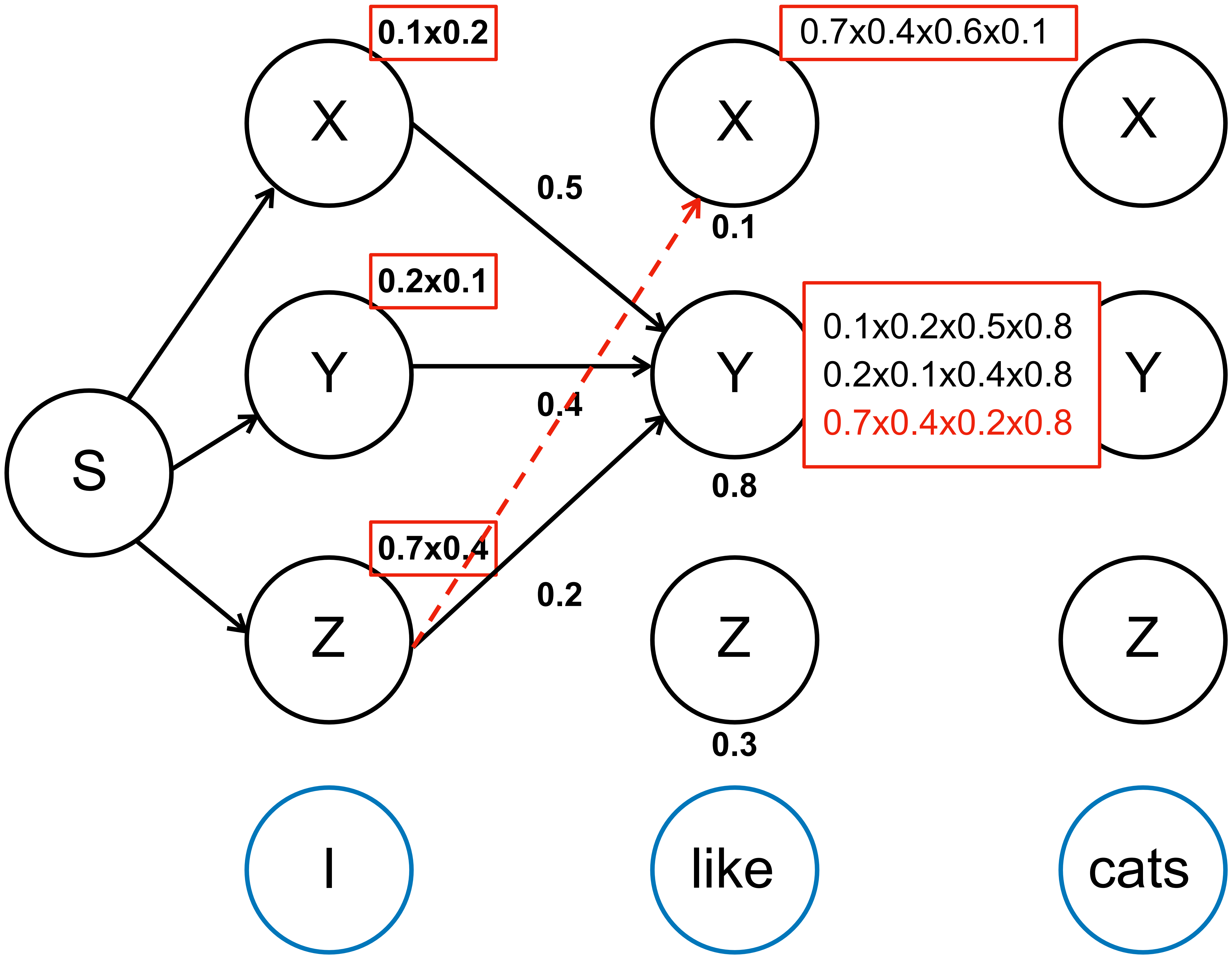
Viterbi Algorithm



	X	Y	Z
S	0.1	0.2	0.7
X	0.2	0.5	0.3
Y	0.4	0.4	0.2
Z	0.6	0.2	0.2

	I	like	cats
X	0.2	0.1	0.7
Y	0.1	0.8	0.1
Z	0.4	0.3	0.3

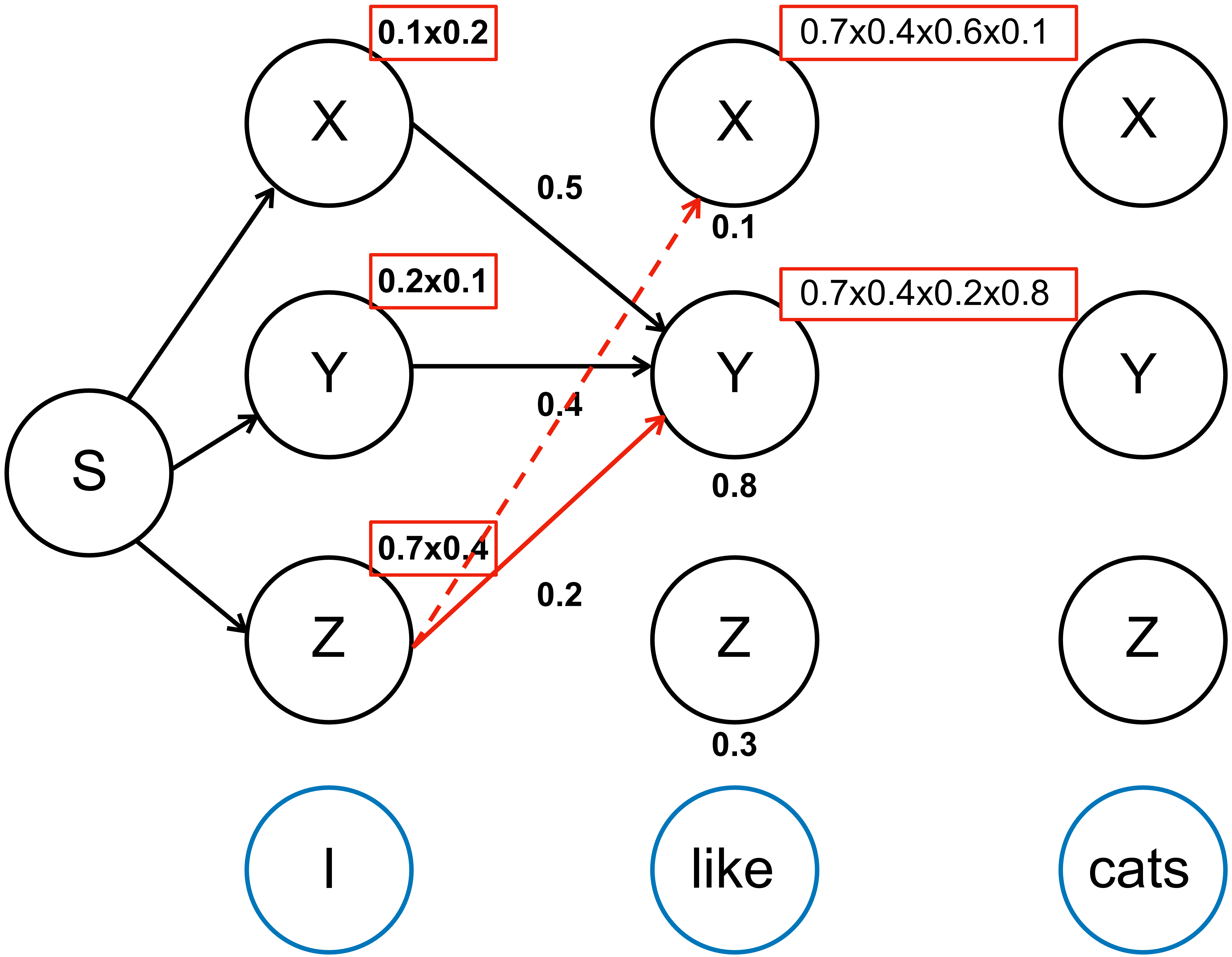
Viterbi Algorithm



	X	Y	Z
S	0.1	0.2	0.7
X	0.2	0.5	0.3
Y	0.4	0.4	0.2
Z	0.6	0.2	0.2

	I	like	cats
X	0.2	0.1	0.7
Y	0.1	0.8	0.1
Z	0.4	0.3	0.3

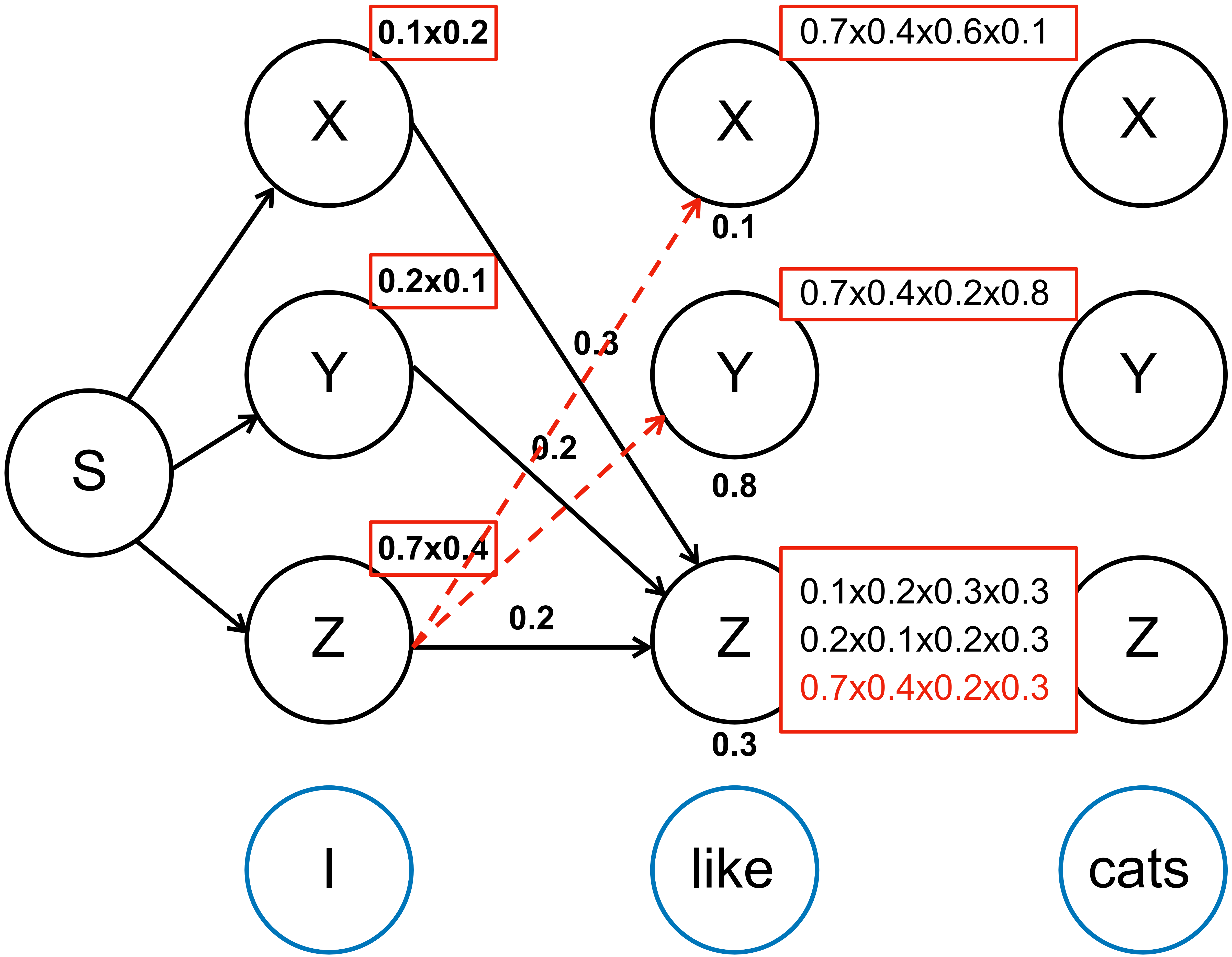
Viterbi Algorithm



	X	Y	Z
S	0.1	0.2	0.7
X	0.2	0.5	0.3
Y	0.4	0.4	0.2
Z	0.6	0.2	0.2

	I	like	cats
X	0.2	0.1	0.7
Y	0.1	0.8	0.1
Z	0.4	0.3	0.3

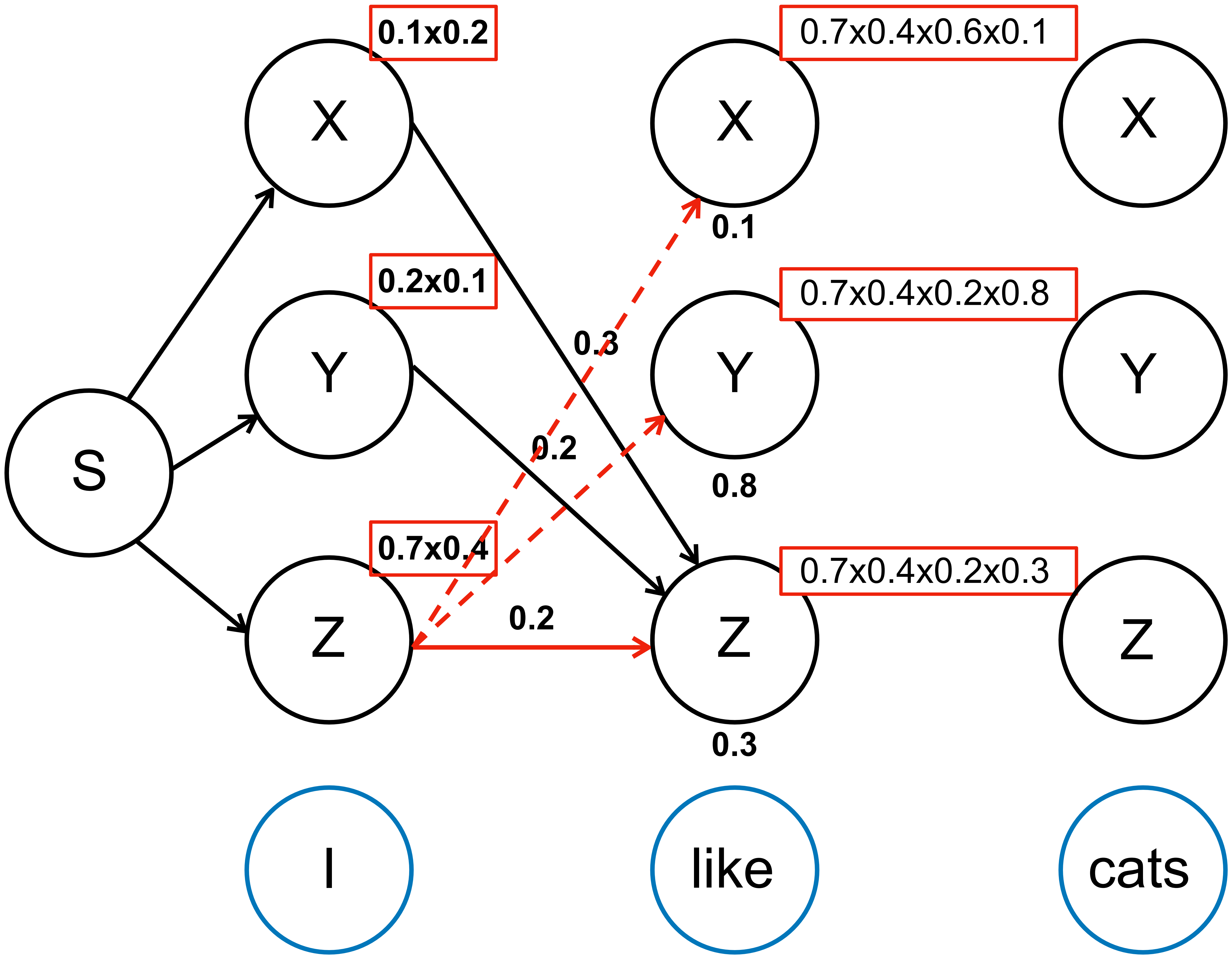
Viterbi Algorithm



	X	Y	Z
S	0.1	0.2	0.7
X	0.2	0.5	0.3
Y	0.4	0.4	0.2
Z	0.6	0.2	0.2

	I	like	cats
X	0.2	0.1	0.7
Y	0.1	0.8	0.1
Z	0.4	0.3	0.3

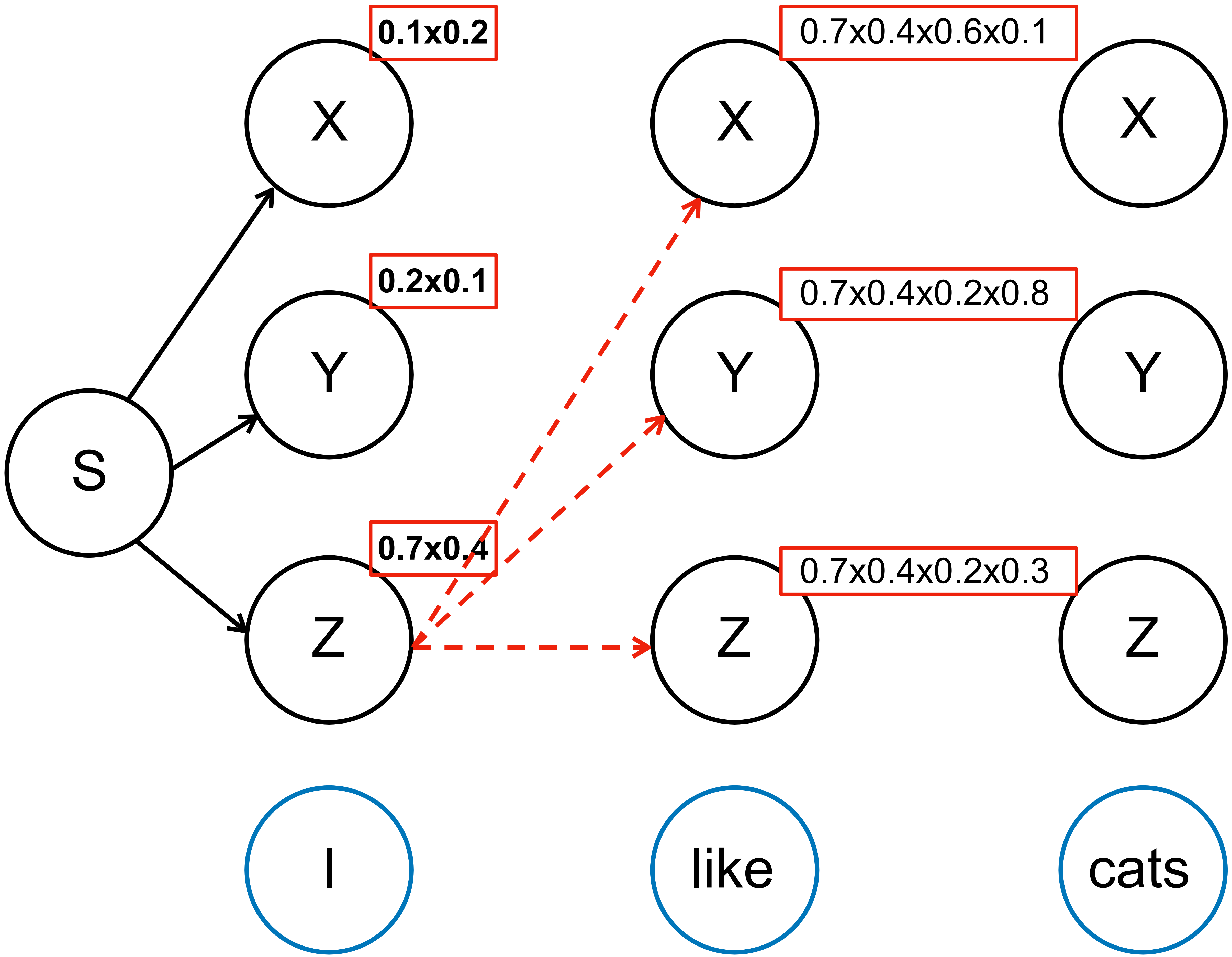
Viterbi Algorithm



	X	Y	Z
S	0.1	0.2	0.7
X	0.2	0.5	0.3
Y	0.4	0.4	0.2
Z	0.6	0.2	0.2

	I	like	cats
X	0.2	0.1	0.7
Y	0.1	0.8	0.1
Z	0.4	0.3	0.3

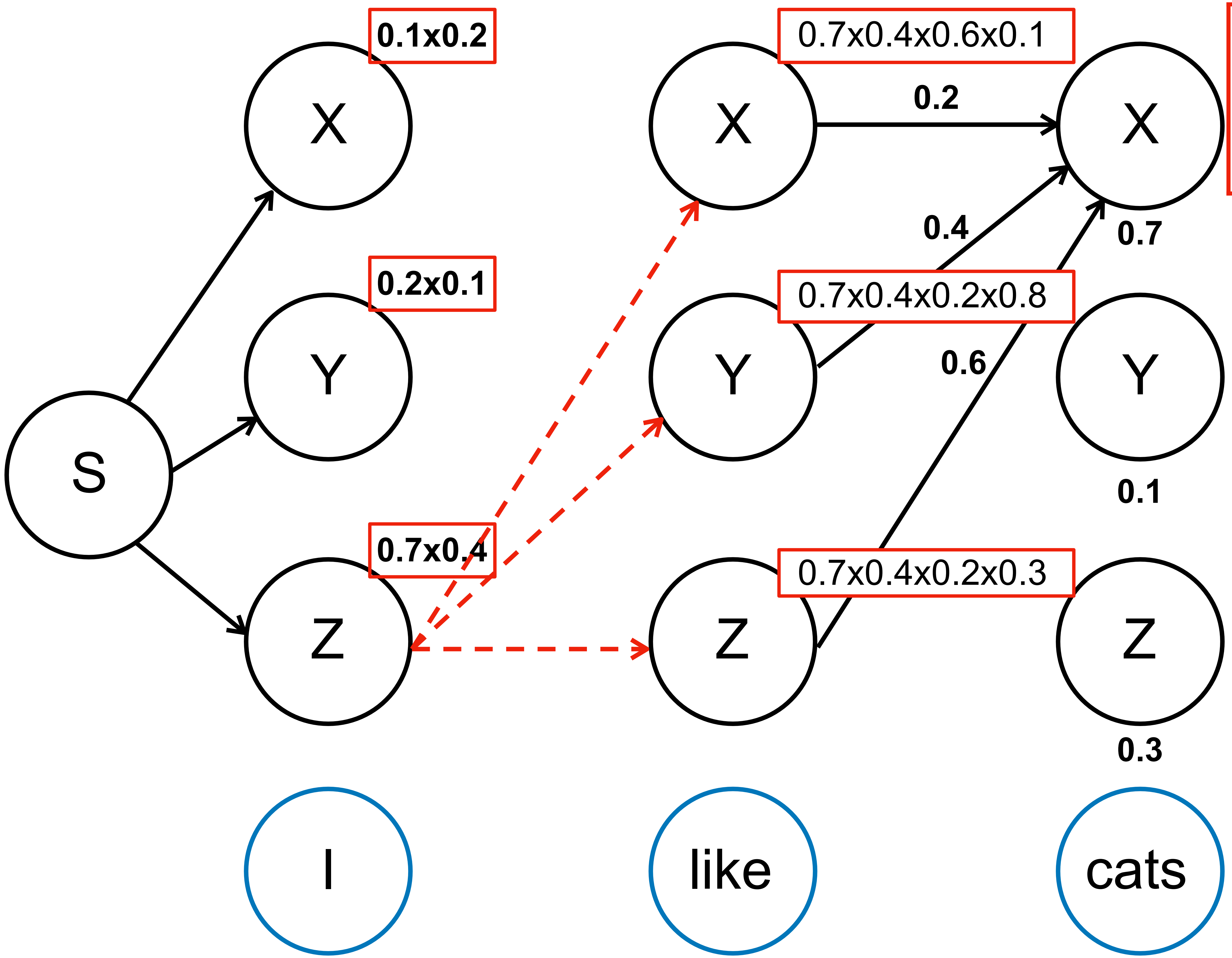
Viterbi Algorithm



	X	Y	Z
S	0.1	0.2	0.7
X	0.2	0.5	0.3
Y	0.4	0.4	0.2
Z	0.6	0.2	0.2

	I	like	cats
X	0.2	0.1	0.7
Y	0.1	0.8	0.1
Z	0.4	0.3	0.3

Viterbi Algorithm

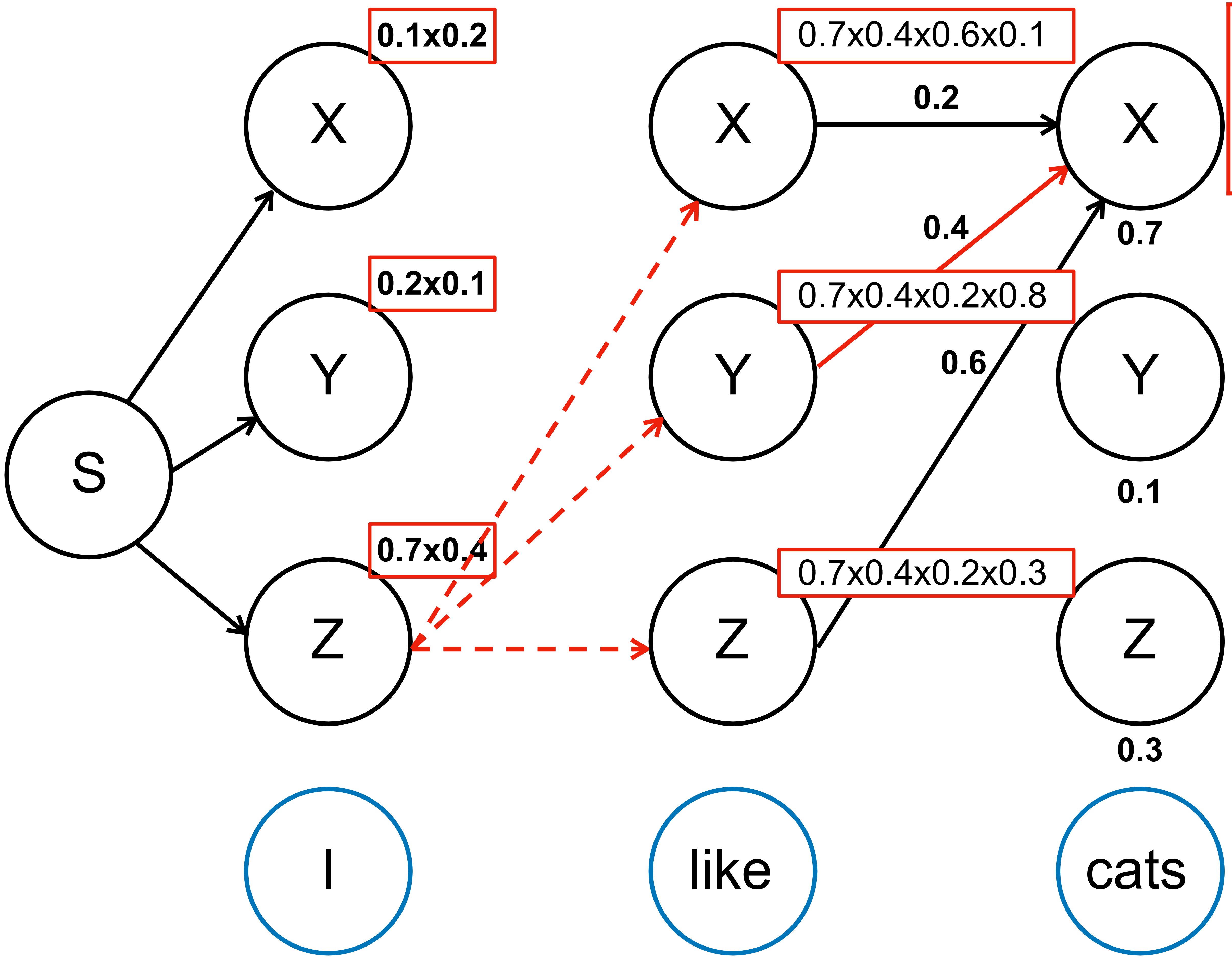


0.7x0.4x0.6x0.1x0.2x0.7 = 0.0023
0.7x0.4x0.2x0.8x0.4x0.7 = 0.0125
0.7x0.4x0.2x0.3x0.6x0.7 = 0.0070

		Y	Z
S	0.1	0.2	0.7
X	0.2	0.5	0.3
Y	0.4	0.4	0.2
Z	0.6	0.2	0.2

	I	like	cats
X	0.2	0.1	0.7
Y	0.1	0.8	0.1
Z	0.4	0.3	0.3

Viterbi Algorithm



0.7x0.4x0.6x0.1x0.2x0.7 = 0.0023

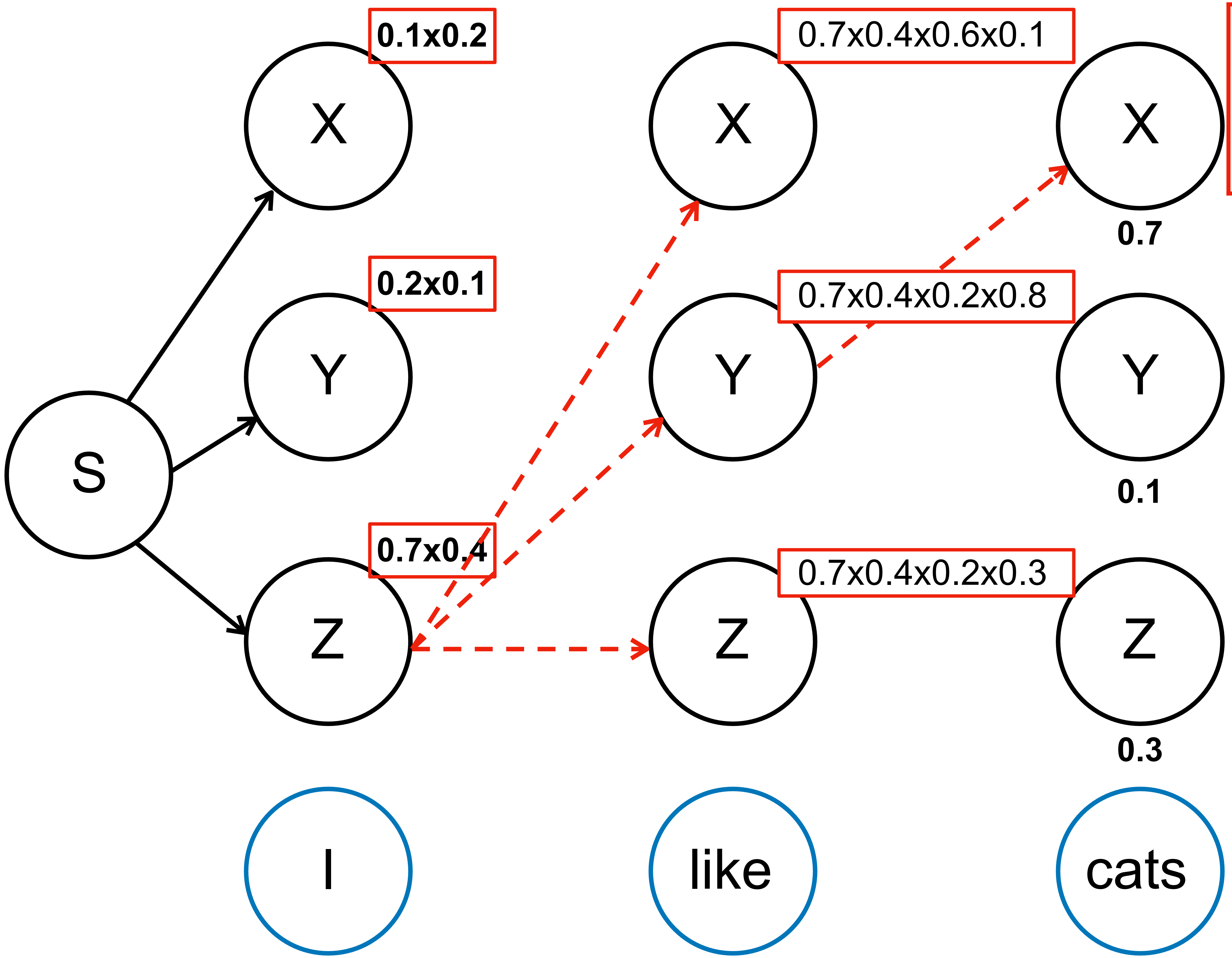
0.7x0.4x0.2x0.8x0.4x0.7 = 0.0125

0.7x0.4x0.2x0.3x0.6x0.7 = 0.0070

		Y	Z
S	0.1	0.2	0.7
X	0.2	0.5	0.3
Y	0.4	0.4	0.2
Z	0.6	0.2	0.2

	I	like	cats
X	0.2	0.1	0.7
Y	0.1	0.8	0.1
Z	0.4	0.3	0.3

Viterbi Algorithm

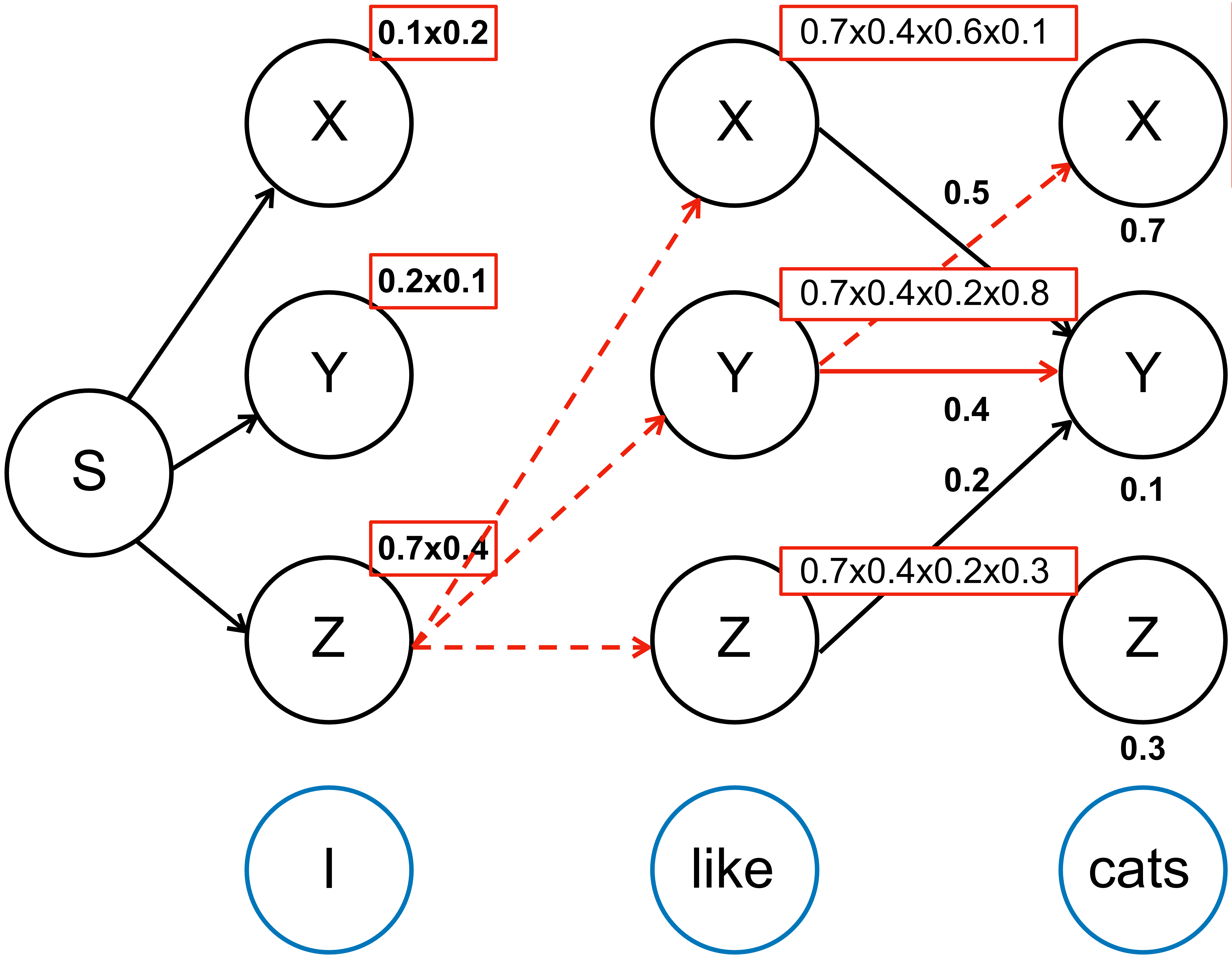


0.7x0.4x0.6x0.1x0.2x0.7 = 0.0023
0.7x0.4x0.2x0.8x0.4x0.7 = 0.0125
0.7x0.4x0.2x0.3x0.6x0.7 = 0.0070

		Y	Z
S	0.1	0.2	0.7
X	0.2	0.5	0.3
Y	0.4	0.4	0.2
Z	0.6	0.2	0.2

	I	like	cats
X	0.2	0.1	0.7
Y	0.1	0.8	0.1
Z	0.4	0.3	0.3

Viterbi Algorithm



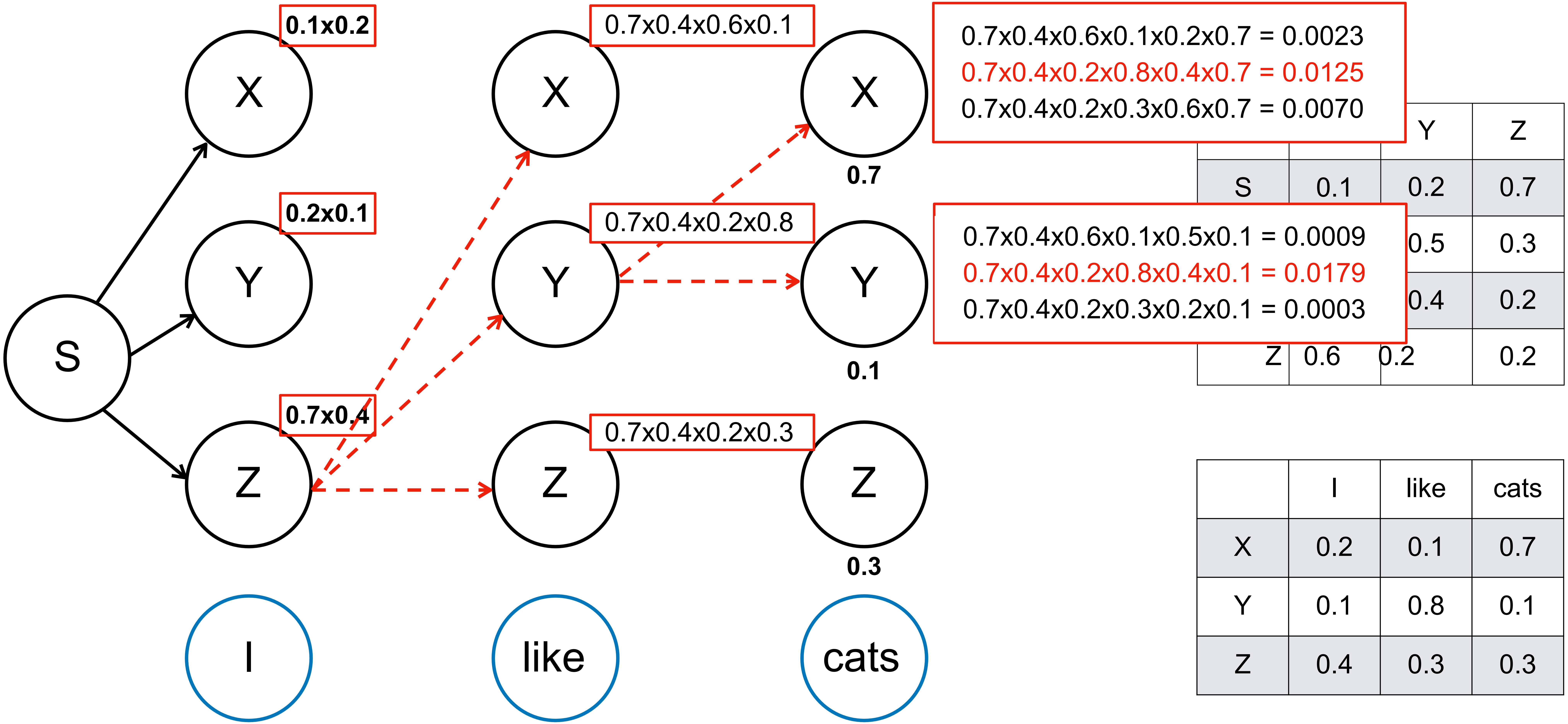
0.7x0.4x0.6x0.1x0.2x0.7 = 0.0023
0.7x0.4x0.2x0.8x0.4x0.7 = 0.0125
0.7x0.4x0.2x0.3x0.6x0.7 = 0.0070

		Y	Z
S 0.1		0.2	0.7
		0.5	0.3
		0.4	0.2
Z 0.6		0.2	0.2

0.7x0.4x0.6x0.1x0.5x0.1 = 0.0009
0.7x0.4x0.2x0.8x0.4x0.1 = 0.0179
0.7x0.4x0.2x0.3x0.2x0.1 = 0.0003

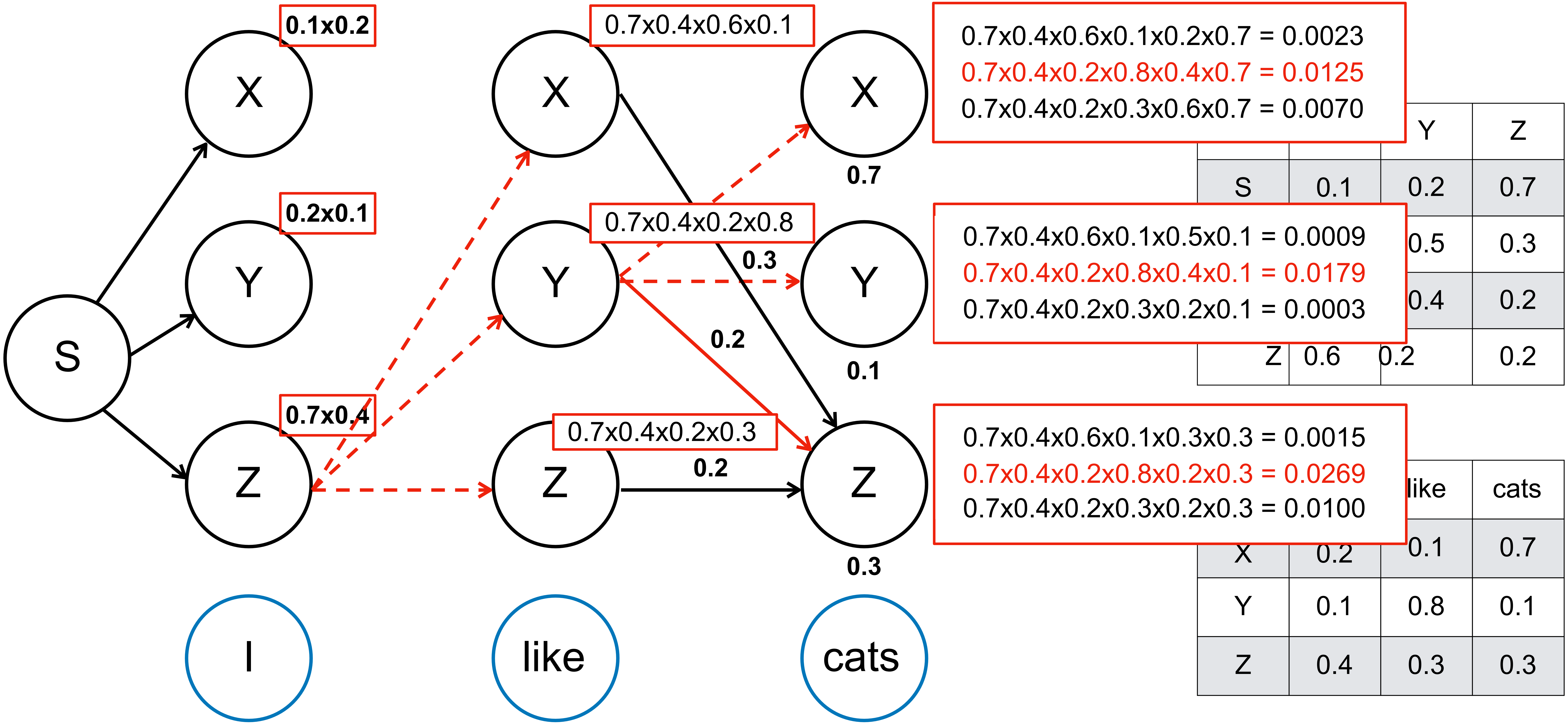
	I	like	cats
X	0.2	0.1	0.7
Y	0.1	0.8	0.1
Z	0.4	0.3	0.3

Viterbi Algorithm

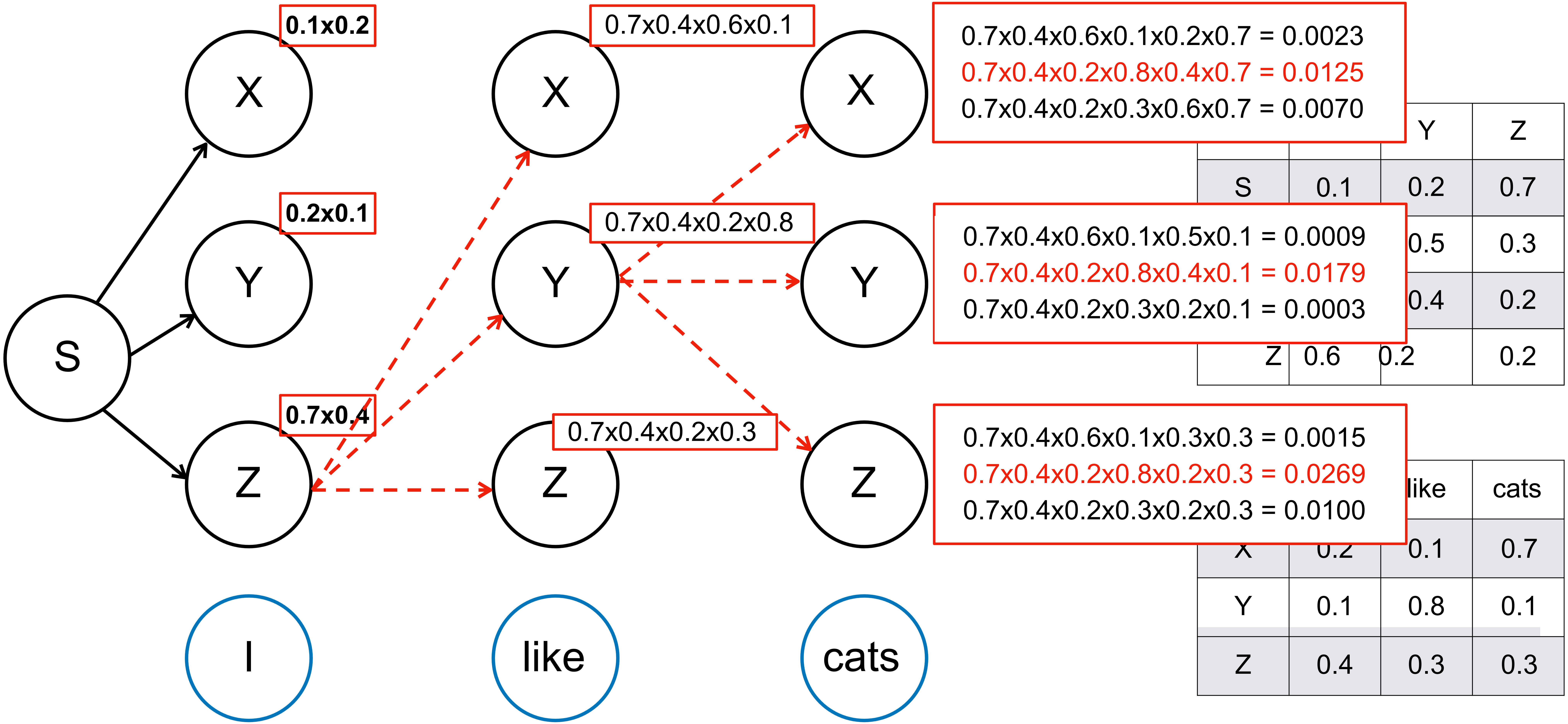


	I	like	cats
X	0.2	0.1	0.7
Y	0.1	0.8	0.1
Z	0.4	0.3	0.3

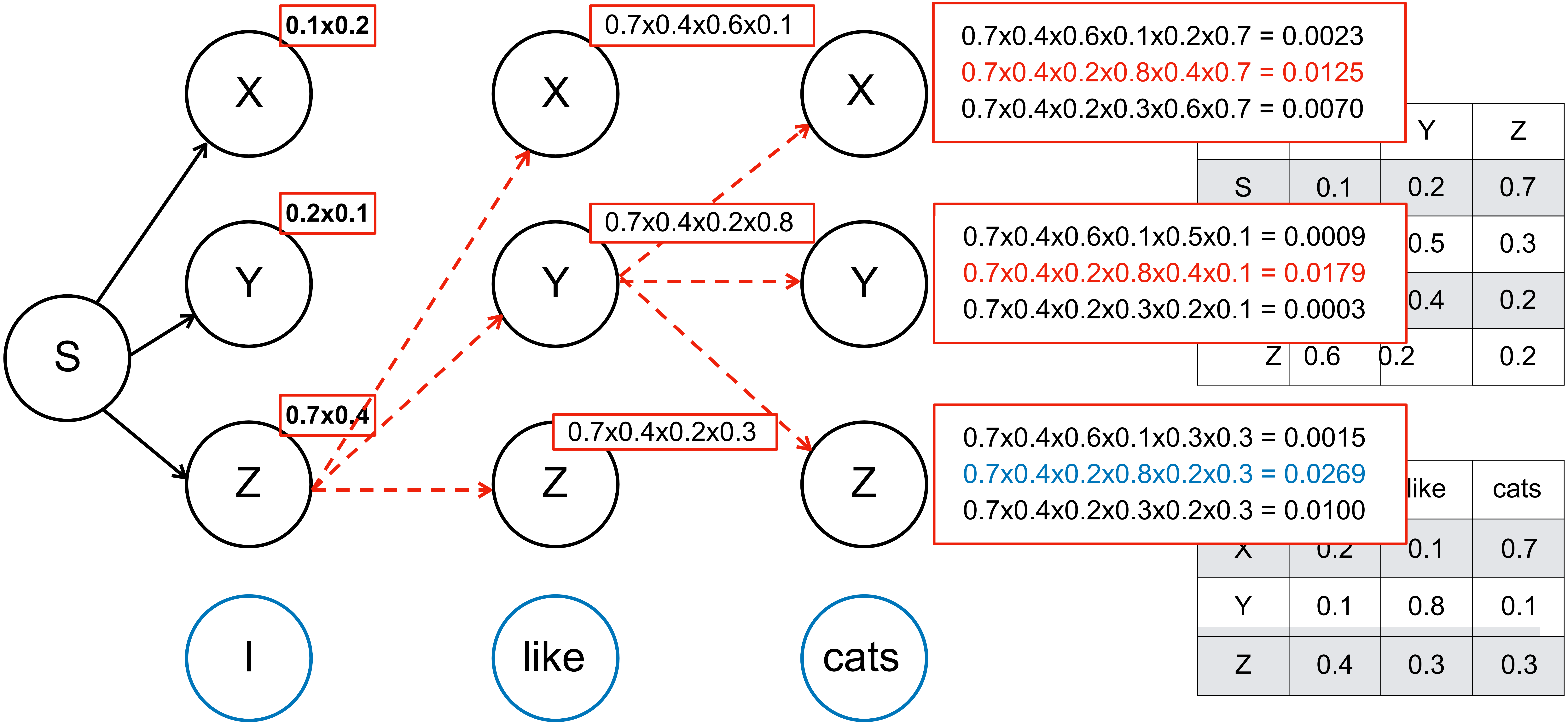
Viterbi Algorithm



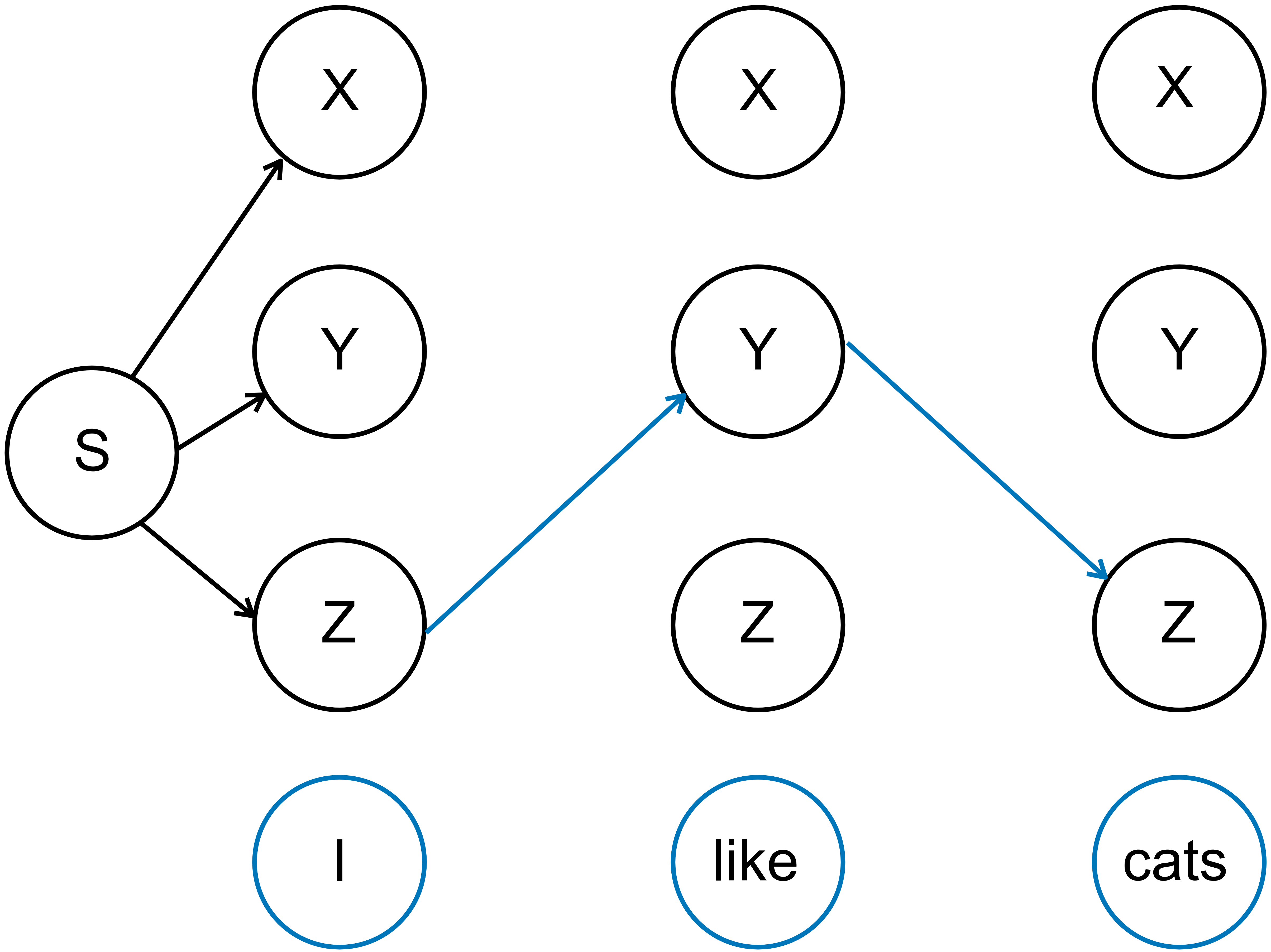
Viterbi Algorithm



Viterbi Algorithm



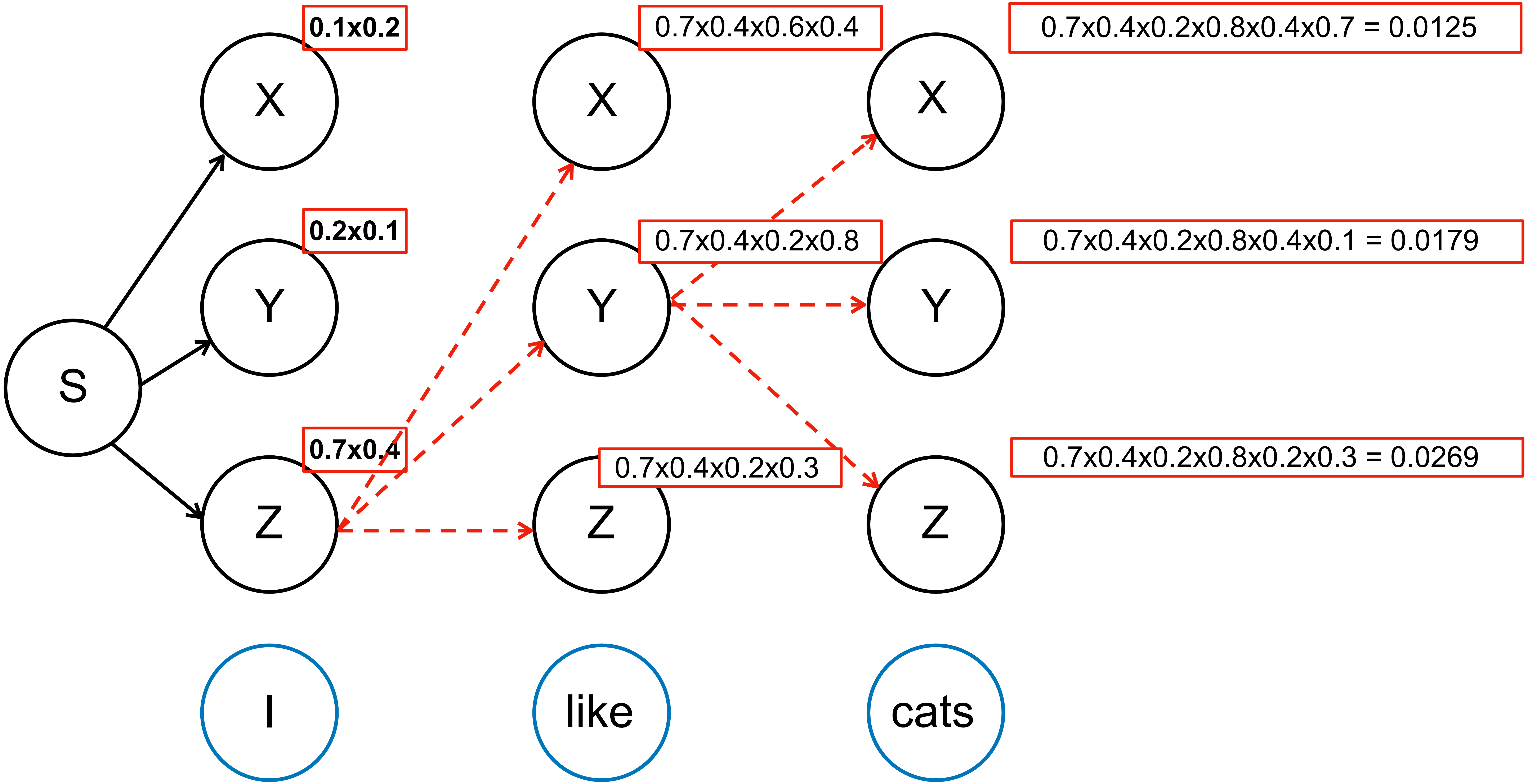
Viterbi Algorithm



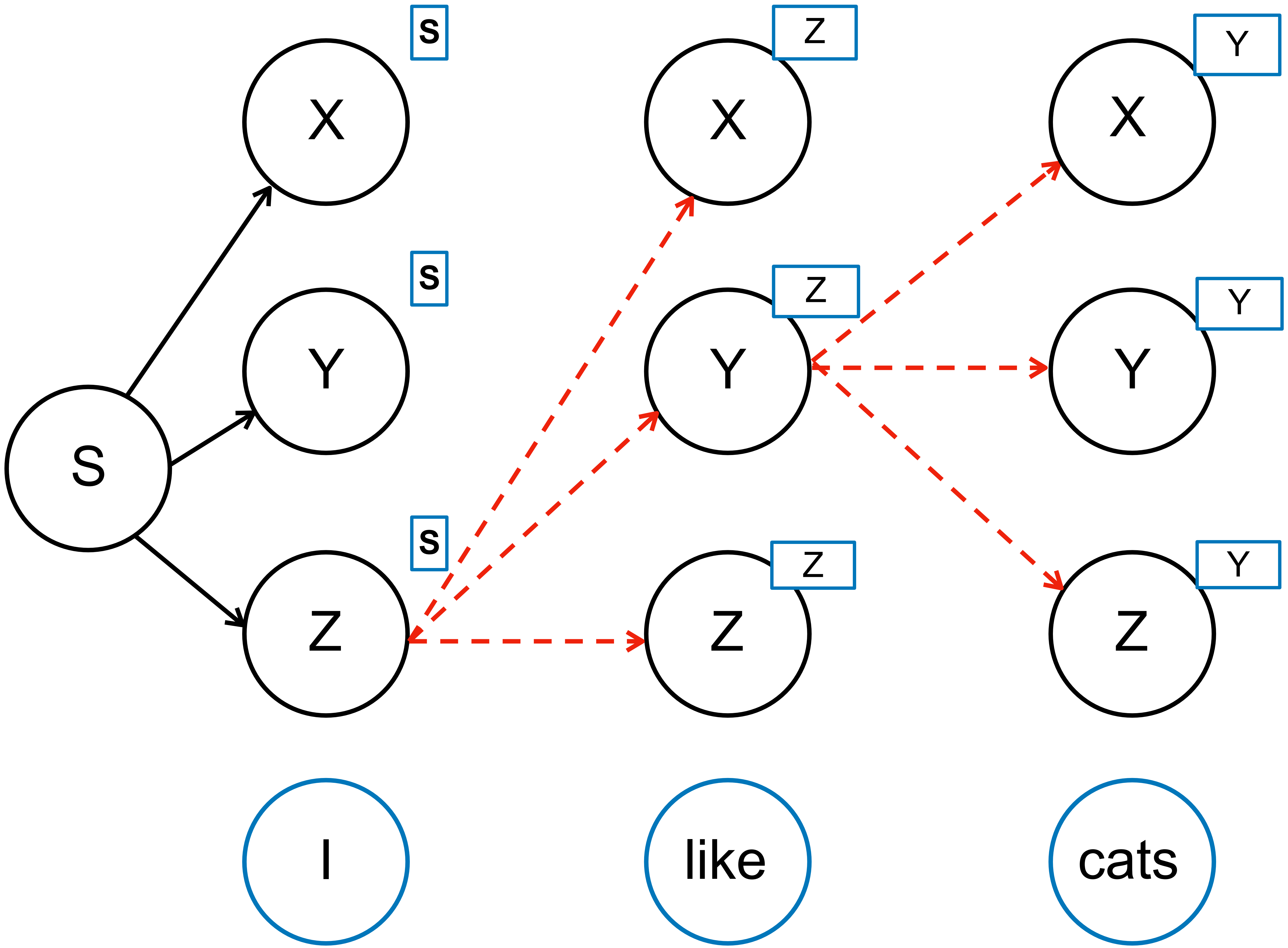
The final tags should be: <Z, Y, Z>

How do we know the path?
Answer: use a backtracking matrix

Viterbi Algorithm



Viterbi Algorithm



The backtracking matrix keeps track of the best node from the previous step.

Viterbi Algorithm

Time complexity:

- At each time step, for **each state**, you compute:

  the best previous state that could transition into it
- O(nk^2) for n time steps and k hidden states.

Applications beyond NLP:

- Speech recognition: converting acoustic signals to text
- Bioinformatics: DNA sequence analysis, gene finding

…

Viterbi Understanding Check

How does Viterbi on a trigram HMM change? What about a 4-gram HMM?

Viterbi Understanding Check

How does Viterbi on a trigram HMM change? What about a 4-gram HMM?

Key: Without Markov, we just need to look **further back** to calculate our likelihood!

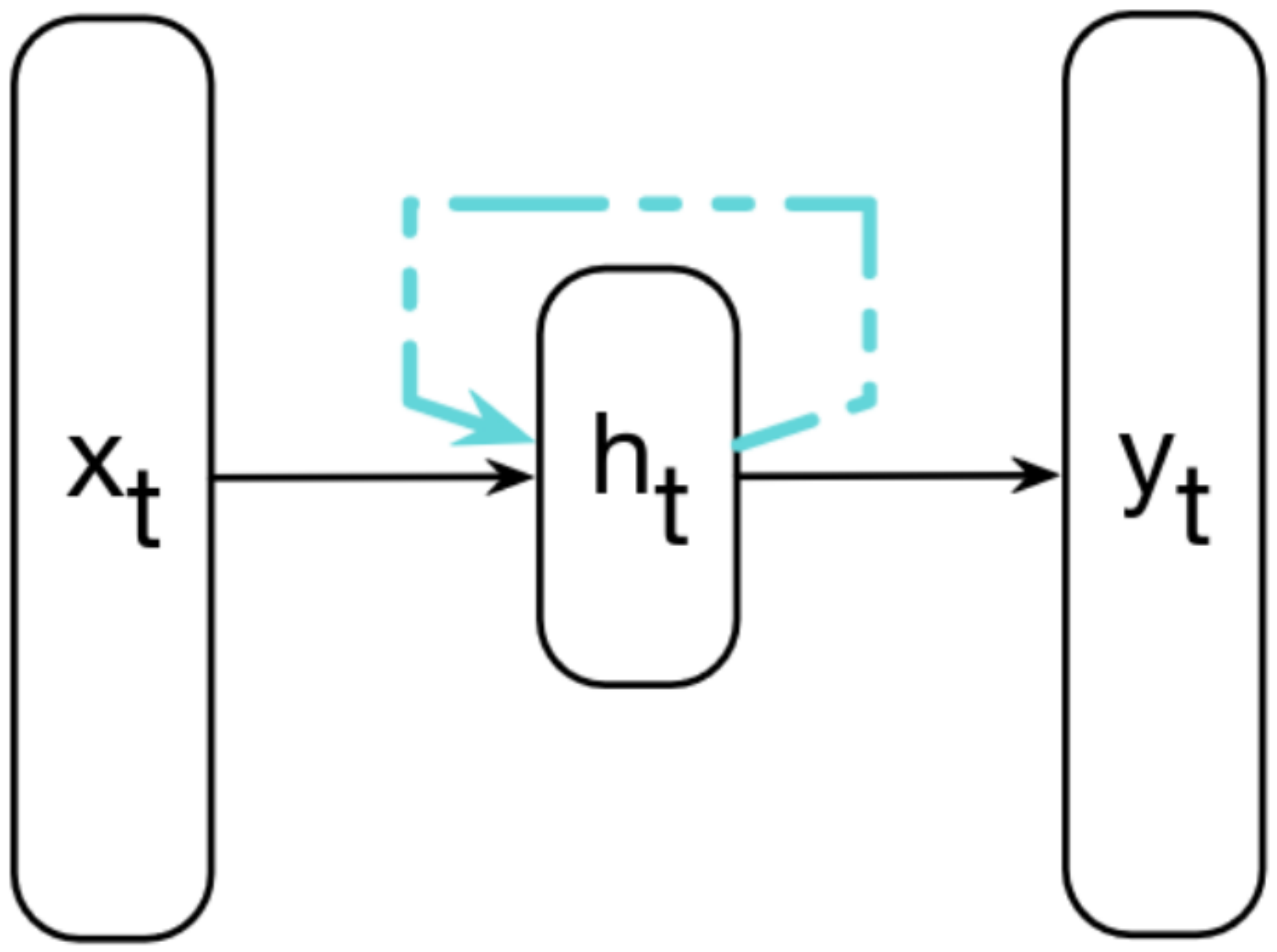
- HMM extended to trigram, 4-gram etc: $P(S, O) = \prod_{i=1}^{n} P(s_i \mid s_{i-1}, s_{i-2}) P(o_i \mid s_i)$
- MLE estimate: $P(s_i \mid s_{i-1}, s_{i-2}) = \dfrac{\text{Count}(s_i, s_{i-1}, s_{i-2})}{\text{Count}(s_{i-1}, s_{i-2})}$
- Viterbi:

$$M[i, j, k] = \max_r M[i-1, k, r]\ P(s_j \mid s_k, s_r)\ P(o_i \mid s_j) \quad 1 \le j, k, r \le K \ \ 1 \le i \le n$$

  - most probable sequence of states ending with state *j* at time *i*, and state *k* at *i-1*
  - Time complexity $= O(nK^3)$

RNN

A **recurrent** neural network is any network that contains a cycle within its network connections.
Unlike a standard feed-forward network that processes each input independently, an RNN has a loop, which lets it carry information forward over time. That’s what we mean by a ‘cycle’ in its connections.



RNN

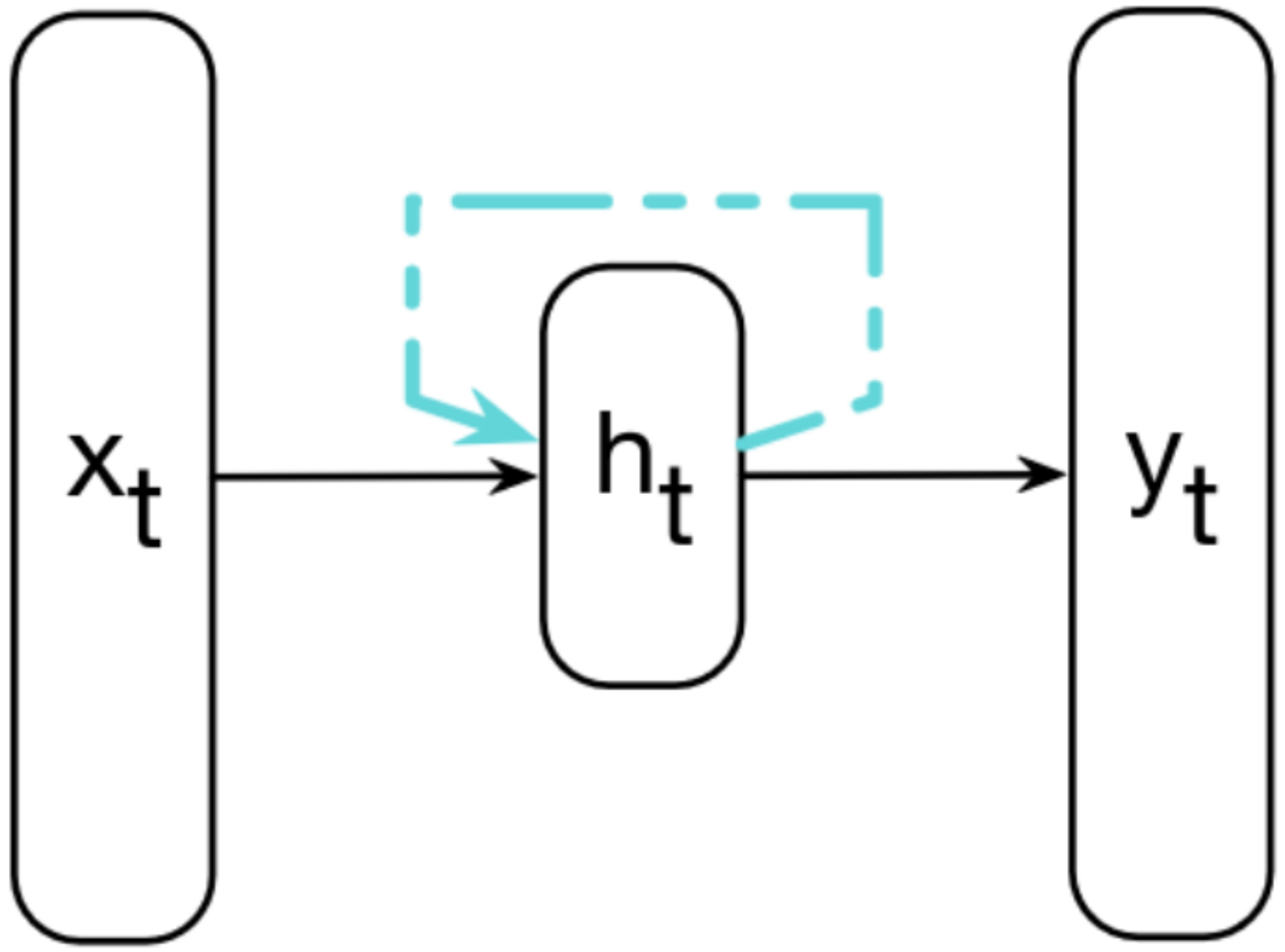
- W transforms the current input, U transforms the previous hidden state, and g( ) is a nonlinearity like tanh or ReLU.
- V maps the hidden state to the output space, and f( ) might be something like softmax for classification.

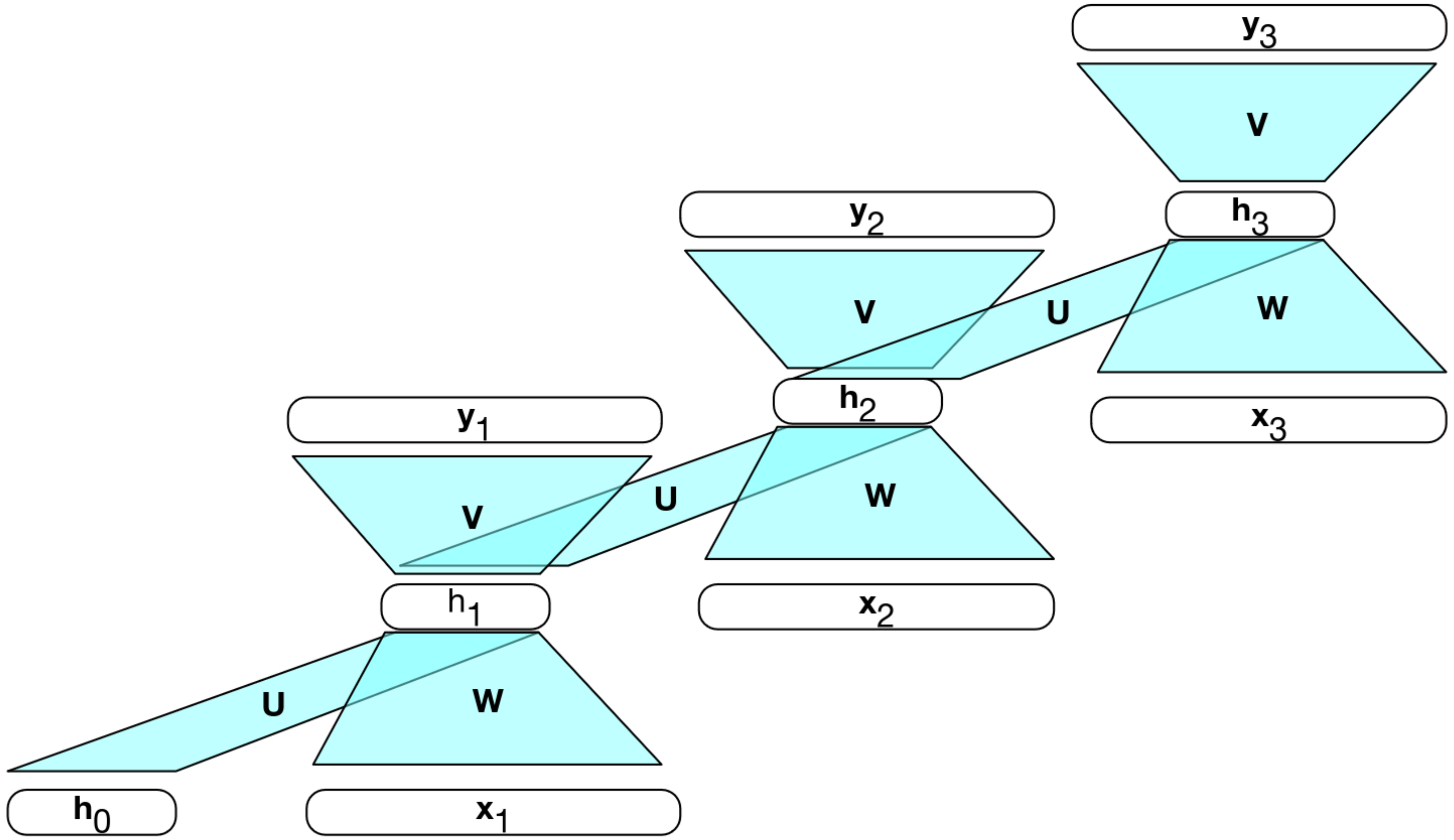
- High-level intuition: *RNN processes one step at a time, and keep a running memory.*

$$
\begin{aligned}
\mathbf{h}_t &= g(\mathbf{U}\mathbf{h}_{t-1} + \mathbf{W}\mathbf{x}_t) \\
\mathbf{y}_t &= f(\mathbf{V}\mathbf{h}_t)
\end{aligned}
$$

RNN

"Unrolled" View of RNNs





RNN

An example of forward pass and loss computation

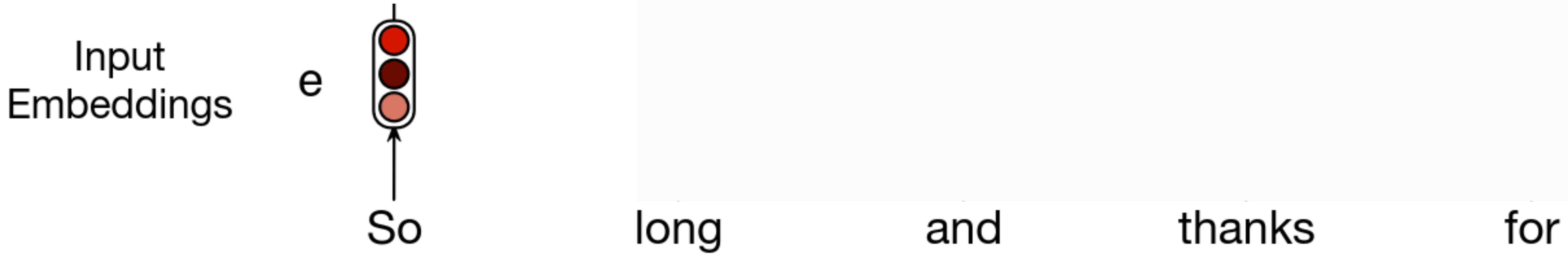
Predict the sentence "*So long and thanks for all the fish***"**

So long and thanks for

RNN

An example of forward pass and loss computation

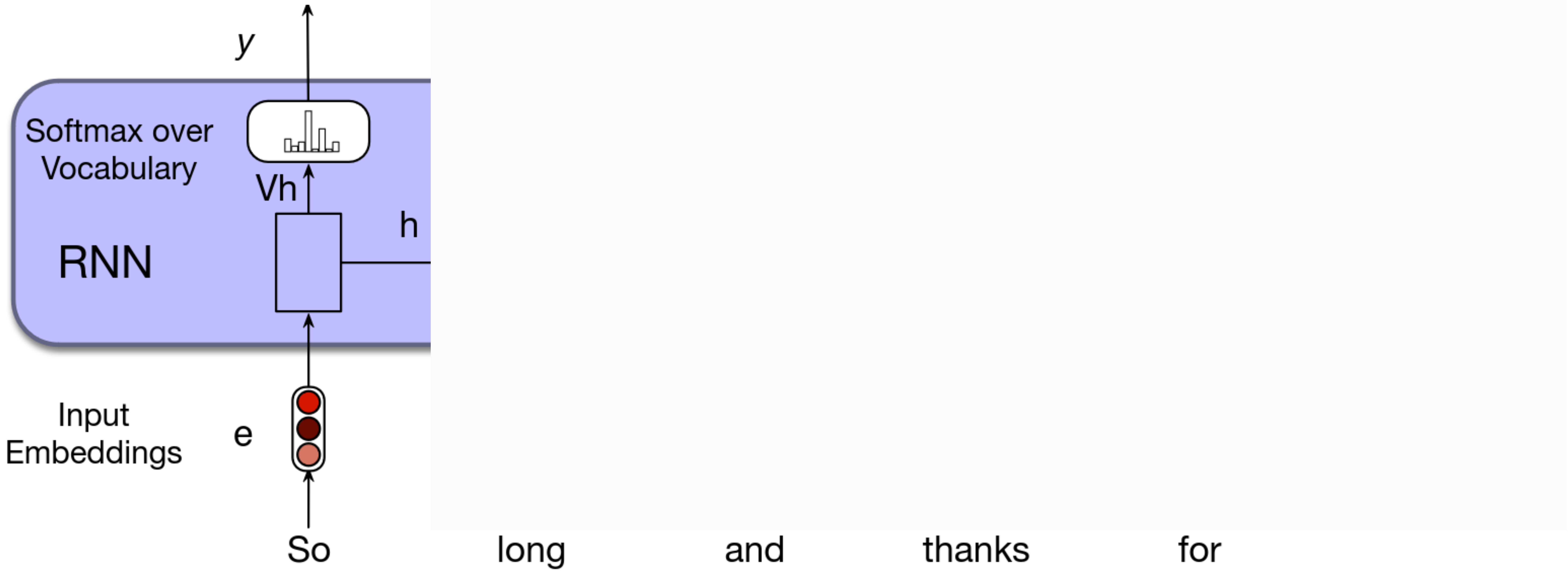
Predict the sentence "*So long and thanks for all the fish*"



RNN

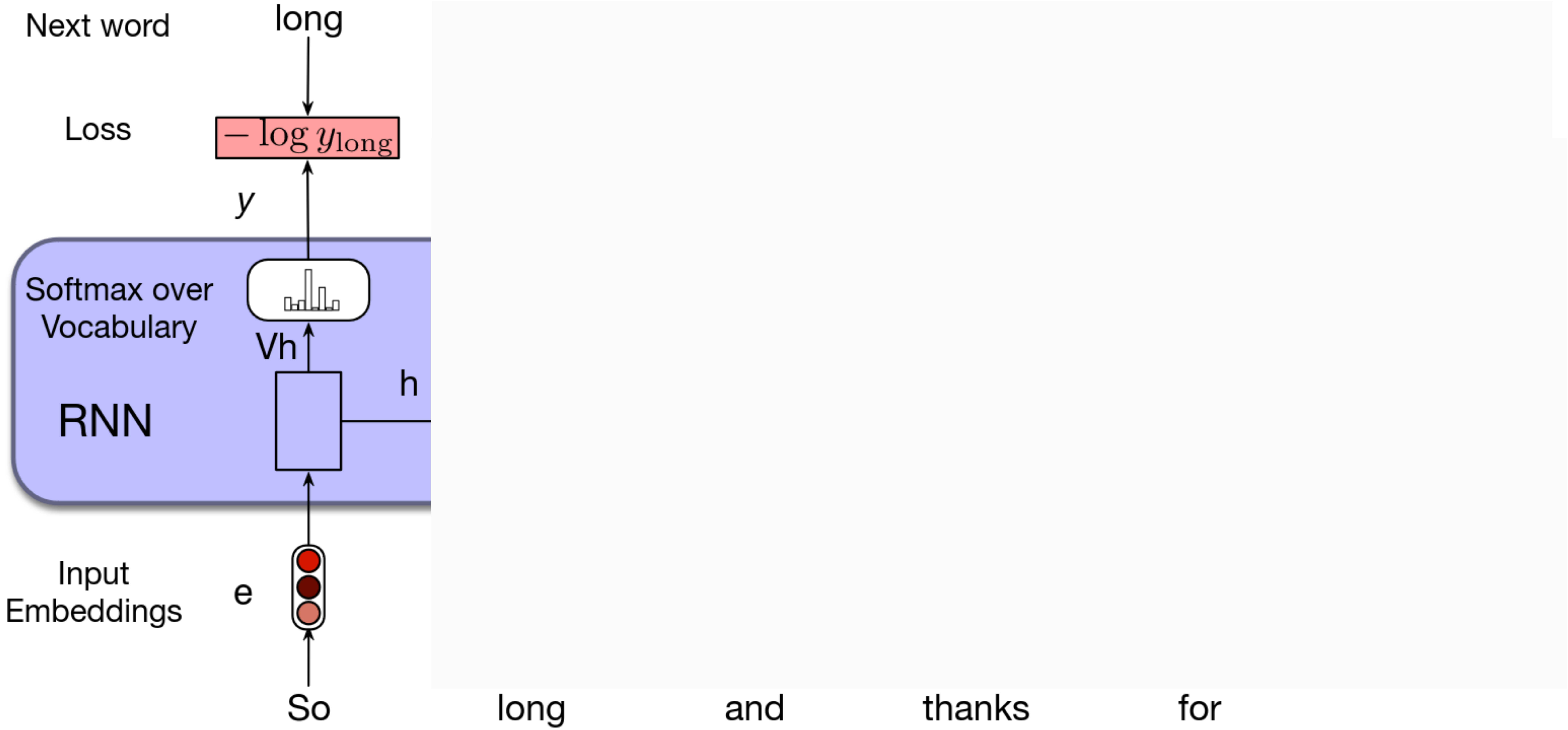
An example of forward pass and loss computation

Predict the sentence "*So long and thanks for all the fish***"**



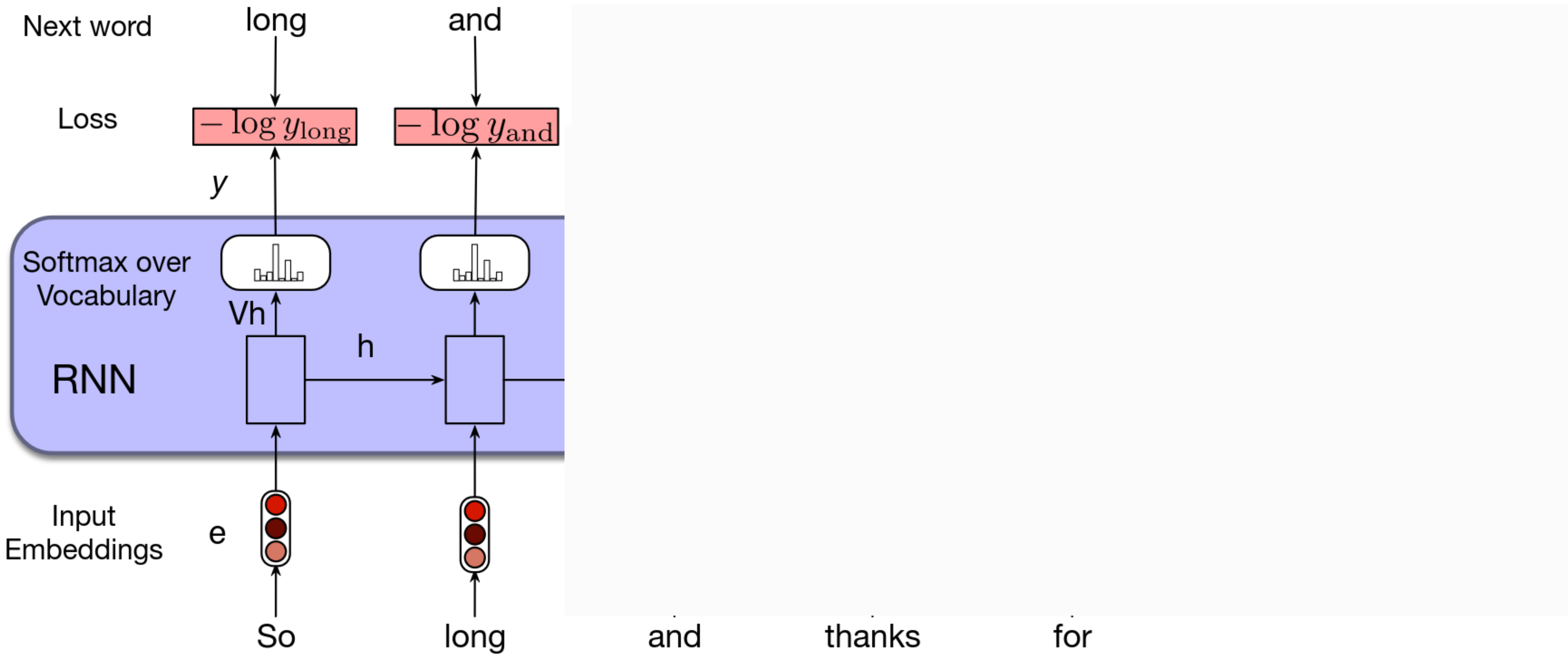
RNN

An example of forward pass and loss computation



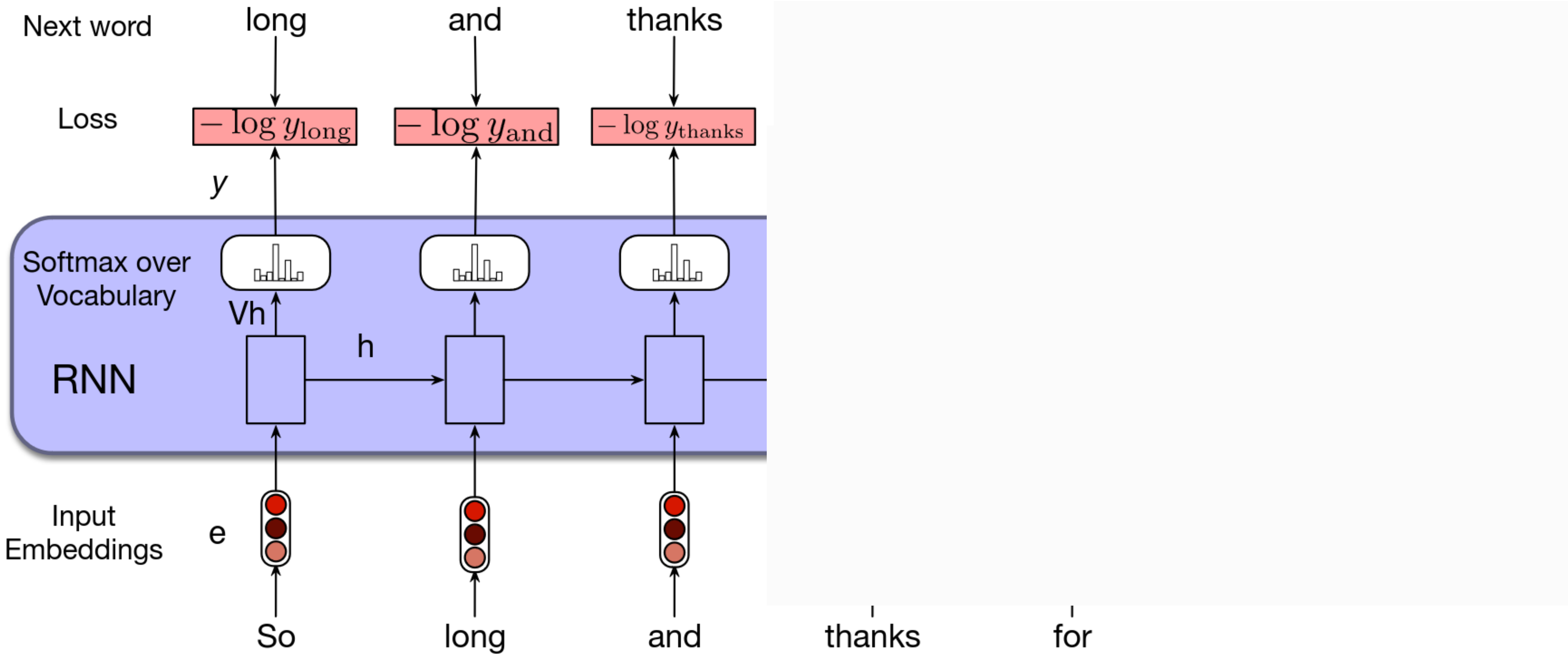
RNN

An example of forward pass and loss computation



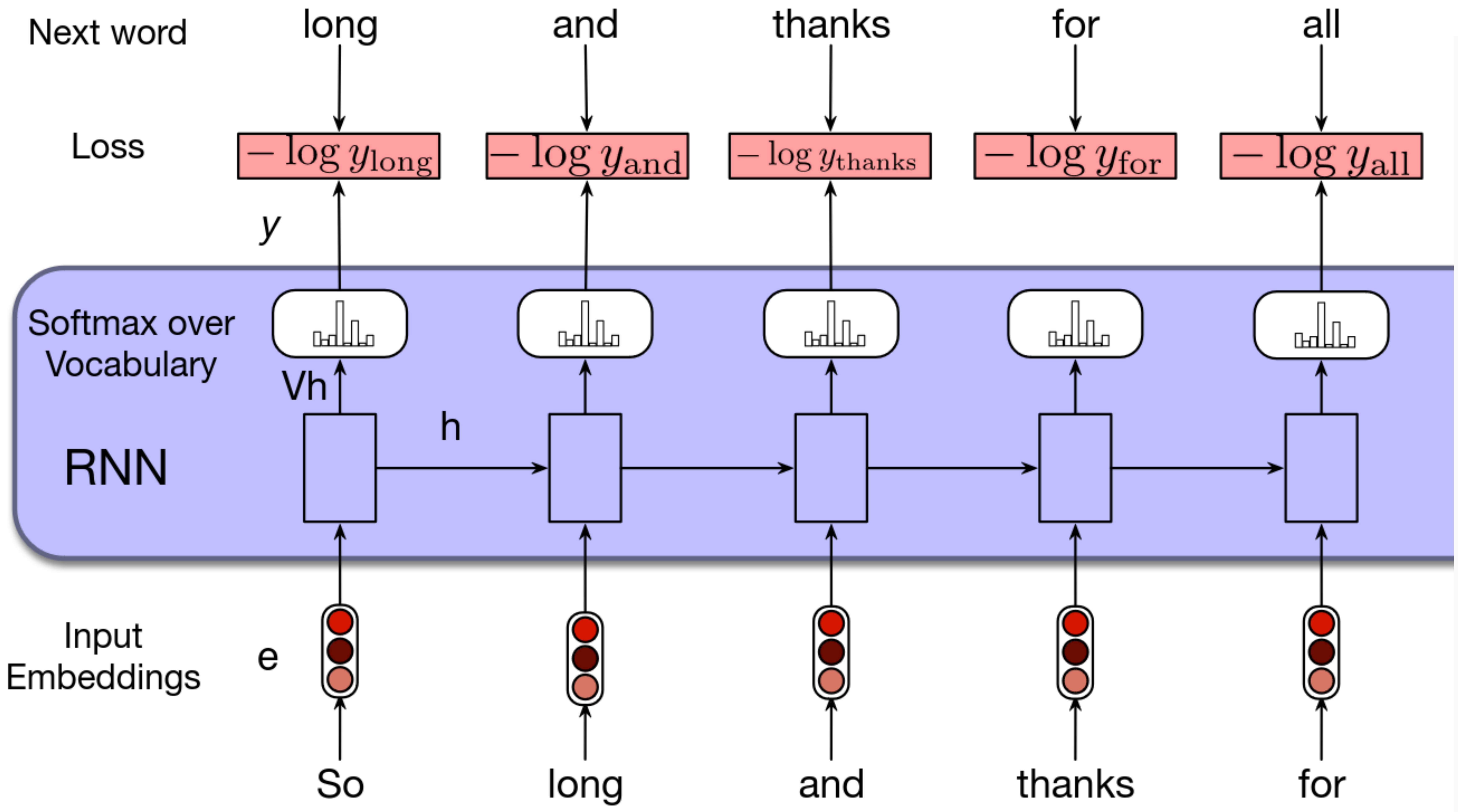
RNN

An example of forward pass and loss computation



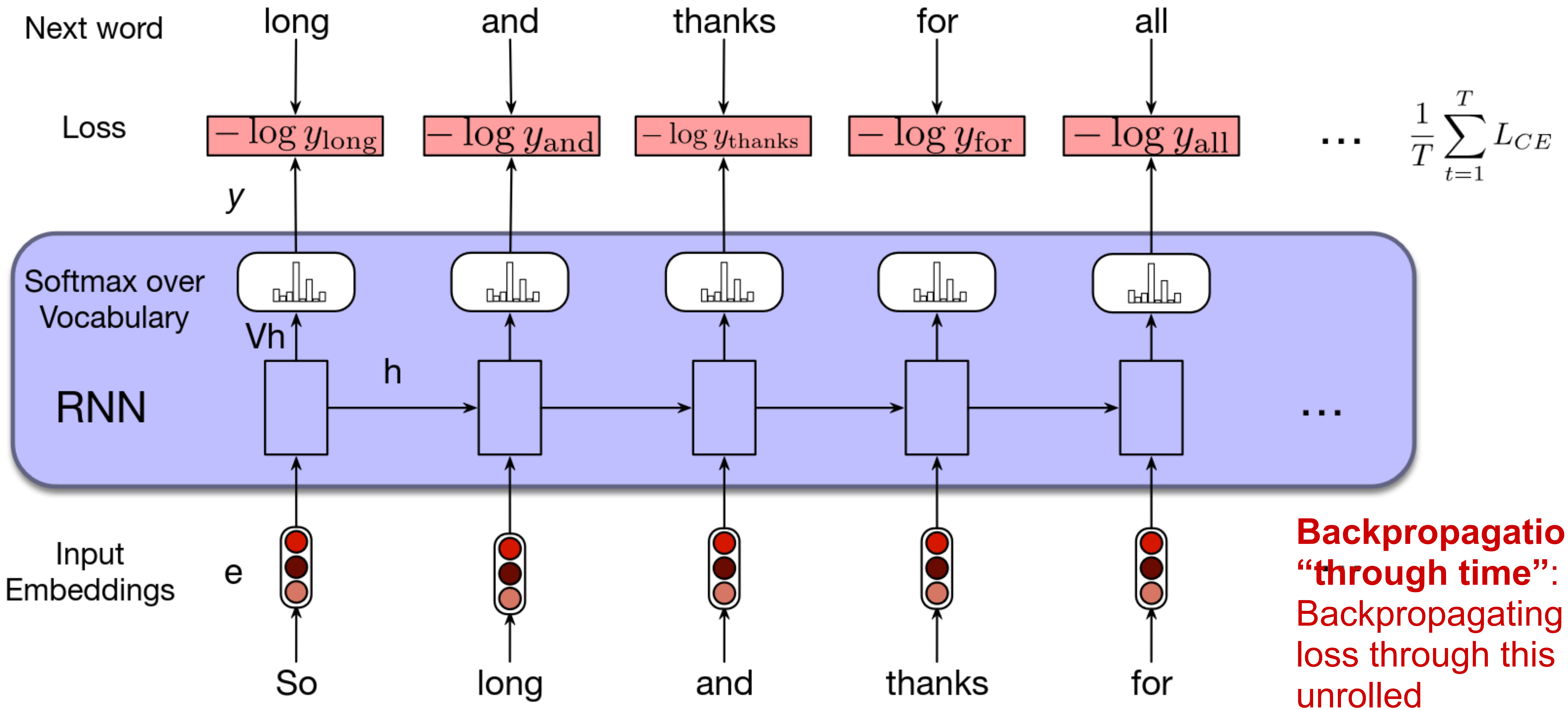
RNN

An example of forward pass and loss computation



RNN

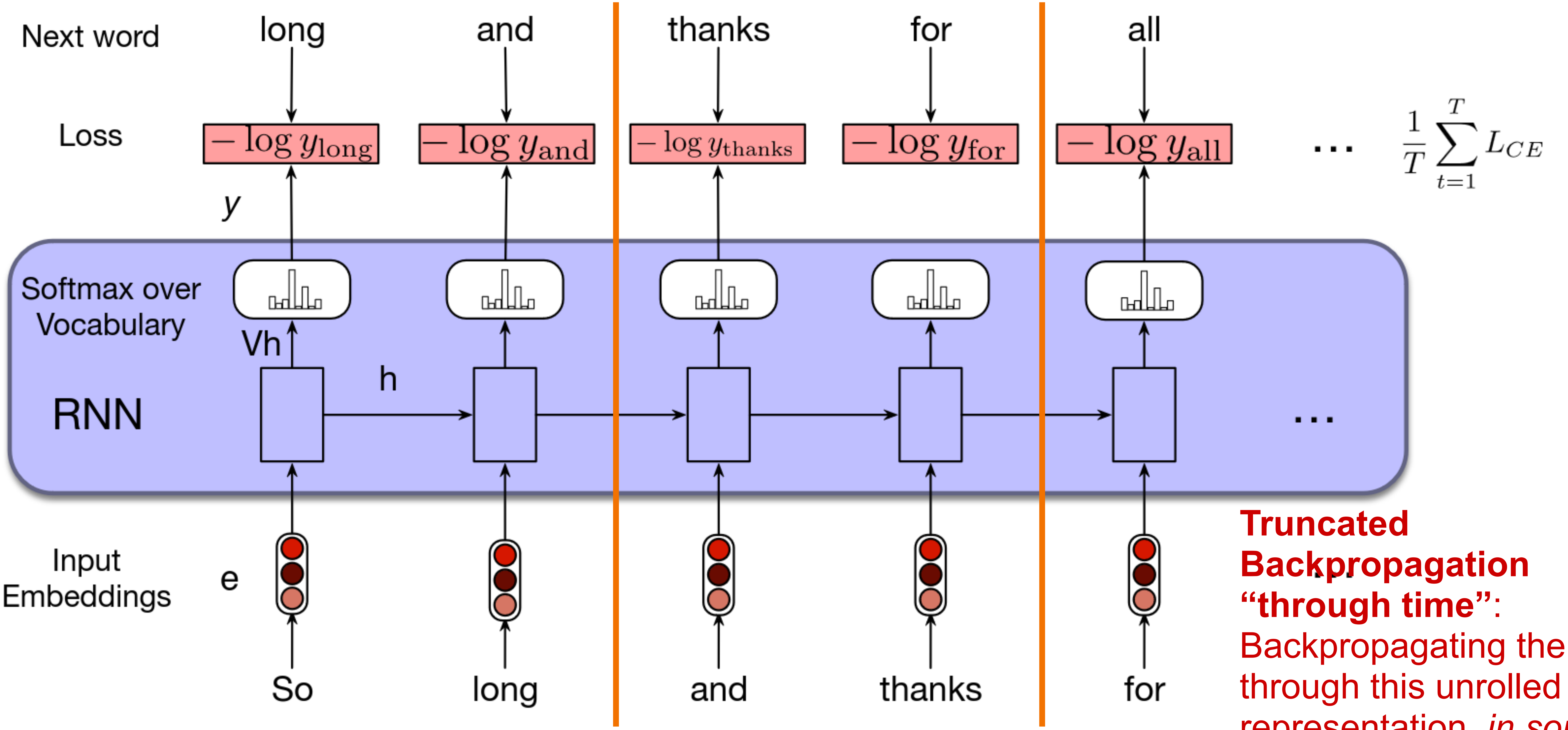
An example of forward pass and loss computation



Backpropagation "through time": Backpropagating the loss through this unrolled representation

RNN

An example of forward pass and loss computation



Truncated Backpropagation "through time": Backpropagating the loss through this unrolled representation, *in some discrete intervals*

RNN

Back propagation through time

$$\mathbf{h}_1 = g(\mathbf{W}\mathbf{h}_0 + \mathbf{U}\mathbf{x}_1 + \mathbf{b})$$
$$\mathbf{h}_2 = g(\mathbf{W}\mathbf{h}_1 + \mathbf{U}\mathbf{x}_2 + \mathbf{b})$$
$$\mathbf{h}_3 = g(\mathbf{W}\mathbf{h}_2 + \mathbf{U}\mathbf{x}_3 + \mathbf{b}) \qquad \hat{\mathbf{y}}_3 = \text{softmax}(\mathbf{W}_o\mathbf{h}_3)$$

$$L_3 = -\log \hat{\mathbf{y}}_3(w_4)$$

First, compute gradient with respect to hidden vector of last time step: $\frac{\partial L_3}{\partial \mathbf{h}_3}$

$$\frac{\partial L_3}{\partial \mathbf{W}} = \frac{\partial L_3}{\partial \mathbf{h}_3}\frac{\partial \mathbf{h}_3}{\partial \mathbf{W}} + \frac{\partial L_3}{\partial \mathbf{h}_3}\frac{\partial \mathbf{h}_3}{\partial \mathbf{h}_2}\frac{\partial \mathbf{h}_2}{\partial \mathbf{W}} + \frac{\partial L_3}{\partial \mathbf{h}_3}\frac{\partial \mathbf{h}_3}{\partial \mathbf{h}_2}\frac{\partial \mathbf{h}_2}{\partial \mathbf{h}_1}\frac{\partial \mathbf{h}_1}{\partial \mathbf{W}}$$

More generally,

$$\boxed{\frac{\partial L}{\partial \mathbf{W}} = -\frac{1}{n}\sum_{t=1}^{n}\sum_{k=1}^{t}\frac{\partial L_t}{\partial \mathbf{h}_t}\left(\prod_{j=k+1}^{t}\frac{\partial \mathbf{h}_j}{\partial \mathbf{h}_{j-1}}\right)\frac{\partial \mathbf{h}_k}{\partial \mathbf{W}}}$$

RNN

Problem with gradients

More generally,

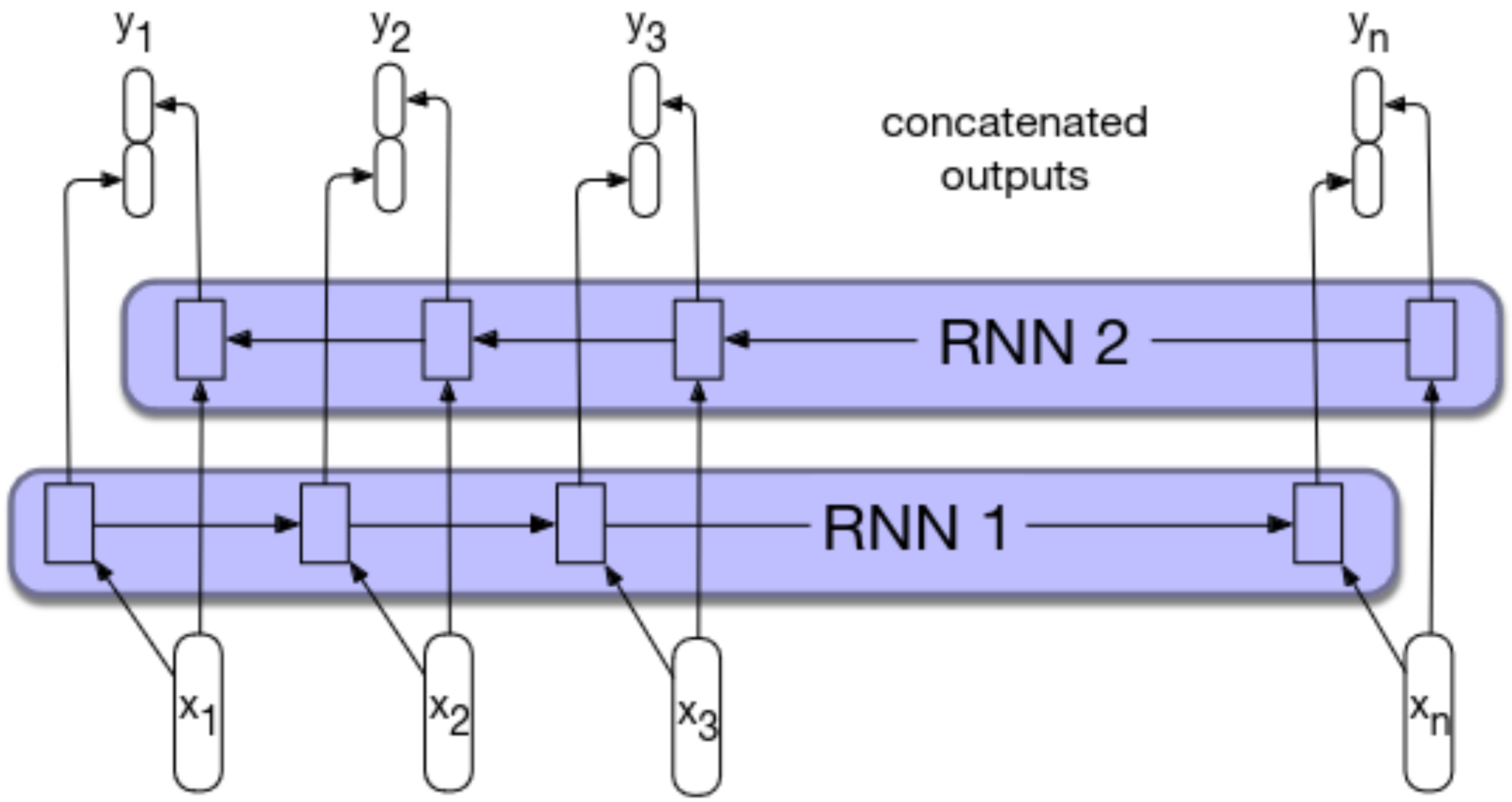
$$\frac{\partial L}{\partial \mathbf{W}} = -\frac{1}{n}\sum_{t=1}^{n}\sum_{k=1}^{t}\frac{\partial L_t}{\partial \mathbf{h}_t}\left(\prod_{j=k+1}^{t}\frac{\partial \mathbf{h}_j}{\partial \mathbf{h}_{j-1}}\right)\frac{\partial \mathbf{h}_k}{\partial \mathbf{W}}$$

1. *Vanishing gradient* $\left\|\frac{\partial h_i}{\partial h_{i-1}}\right\|_2 < 1$
2. *Exploding gradient* $\left\|\frac{\partial h_i}{\partial h_{i-1}}\right\|_2 > 1$

RNN

RNNs can have different variants

- For example, when we have access to an entire sequence x0, …, xn at once, we can improve performance using bidirectional RNNs



RNN

Tradeoffs of RNNs

- Can handle arbitrary length inputs
- Reuse weights to reduce total model parameters

- Suffers from vanishing/exploding gradients
- Doesn’t take full advantage of highly parallel hardware