Midterm Review

COS 484 - Part 1

Austin Wang, with slides from Howard Chen, Danqi Chen

Basics

Basics: Probability

$\Pr[A] = P(\text{all outcomes in } A)$
$\Pr[\bar{A}] = 1 - \Pr[A]$

Addition rule:
$\Pr[A \cup B] = \Pr[A] + \Pr[B] - \Pr[A \cap B]$

Chain rule:
$\Pr[AB] = \Pr[B]\Pr[A \mid B]$

For k events:
$\Pr[A_1A_2\ldots A_k] = \Pr[A_1]\Pr[A_2 \mid A_1]\Pr[A_3 \mid A_1A_2]\cdots\Pr[A_k \mid A_1A_2\ldots A_{k-1}]$

Events A, B are independent if $\Pr[A \cap B] = \Pr[A] \cdot \Pr[B]$
Independence also implies $\Pr[A \mid B] = \Pr[A]$ and $\Pr[B \mid A] = \Pr[B]$

Bayes rule:

$$\Pr[A \mid B] = \frac{\Pr[B \mid A]\Pr[A]}{\Pr[B]}$$

Law of total Probability:

$$\Pr[B] = \sum_i \Pr[B \mid A_i]\Pr[A_i]$$

If $\sum_i \Pr[A_i] = 1$

Basics: Exponents, Logs and Sums

Exponential Laws	Logarithm Laws
$x^a \cdot x^b = x^{a+b}$	$\log(ab) = \log(a) + \log(b)$
$\dfrac{x^a}{x^b} = x^{a-b}$	$\log\left(\dfrac{a}{b}\right) = \log(a) - \log(b)$
$(x^a)^b = x^{ab}$	$\log(a^b) = b \cdot \log(a)$
$x^{-a} = \dfrac{1}{x^a}$	$\log_x\left(\dfrac{1}{x^a}\right) = -a$
$x^0 = 1$	$\log_x 1 = 0$

$$e^{\log_e x} = x$$

$$\sum_i (x_i + y_i) = \sum_i x_i + \sum_i y_i$$

$$\sum_i \sum_j x_{ij} = \sum_j \sum_i x_{ij}$$

$$\sum_{i=1}^{n} x_i = \sum_{i\ odd} x_i + \sum_{i\ even} x_i$$

Language Models

Definition: A **language model** is a probabilistic model over sequences of words (tokens). $P(w_1, w_2, \ldots, w_n)$

Language Models

Definition: A **language model** is a probabilistic model over sequences of words (tokens). $P(w_1, w_2, \ldots, w_n)$

We can decompose this using the **chain rule**:

$$P(w_1, w_2, \ldots, w_n) = P(w_1) \cdot P(w_2 \mid w_1) \cdot P(w_3 \mid w_1, w_2) \cdot \ldots \cdot P(w_n \mid w_1, \ldots, w_{n-1})$$

Language Models

Definition: A **language model** is a probabilistic model over sequences of words (tokens). $P(w_1, w_2, \ldots, w_n)$

We can decompose this using the **chain rule**:

$$P(w_1, w_2, \ldots, w_n) = P(w_1) \cdot P(w_2 \mid w_1) \cdot P(w_3 \mid w_1, w_2) \cdot \ldots \cdot P(w_n \mid w_1, \ldots, w_{n-1})$$

To make estimating these probabilities tractable, we use **Markov assumption** (e.g. bigram)

$$P(w_1, w_2, \ldots, w_n) \approx P(w_1)P(w_2 \mid w_1)\ldots P(w_n \mid w_{n-1}) = \prod_{i=1}^{n} P(w_i \mid w_{i-1})$$

Language Models

Definition: A **language model** is a probabilistic model over sequences of words (tokens). $P(w_1, w_2, \ldots, w_n)$

We can decompose this using the **chain rule**:

$$P(w_1, w_2, \ldots, w_n) = P(w_1) \cdot P(w_2 \mid w_1) \cdot P(w_3 \mid w_1, w_2) \cdot \ldots \cdot P(w_n \mid w_1, \ldots, w_{n-1})$$

To make estimating these probabilities tractable, we use **Markov assumption** (e.g. bigram)

$$P(w_1, w_2, \ldots, w_n) \approx P(w_1)P(w_2 \mid w_1) \ldots P(w_n \mid w_{n-1}) = \prod_{i=1}^{n} P(w_i \mid w_{i-1})$$

We set these probabilities to **maximize the probability of the training corpus (MLE).** For trigram:

$$P(w_3 \mid w_1, w_2) \leftarrow \frac{\text{Count}(w_1, w_2, w_3)}{\text{Count}(w_1, w_2)}$$

Language Models

Definition: A **language model** is a probabilistic model over sequences of words (tokens). $P(w_1, w_2, \ldots, w_n)$

We can decompose this using the **chain rule**:

$$P(w_1, w_2, \ldots, w_n) = P(w_1) \cdot P(w_2 \mid w_1) \cdot P(w_3 \mid w_1, w_2) \cdot \ldots \cdot P(w_n \mid w_1, \ldots, w_{n-1})$$

To make estimating these probabilities tractable, we use **Markov assumption** (e.g. bigram)

$$P(w_1, w_2, \ldots, w_n) \approx P(w_1)P(w_2 \mid w_1)\ldots P(w_n \mid w_{n-1}) = \prod_{i=1}^{n} P(w_i \mid w_{i-1})$$

We set these probabilities to **maximize the probability of the training corpus (MLE).** For trigram:

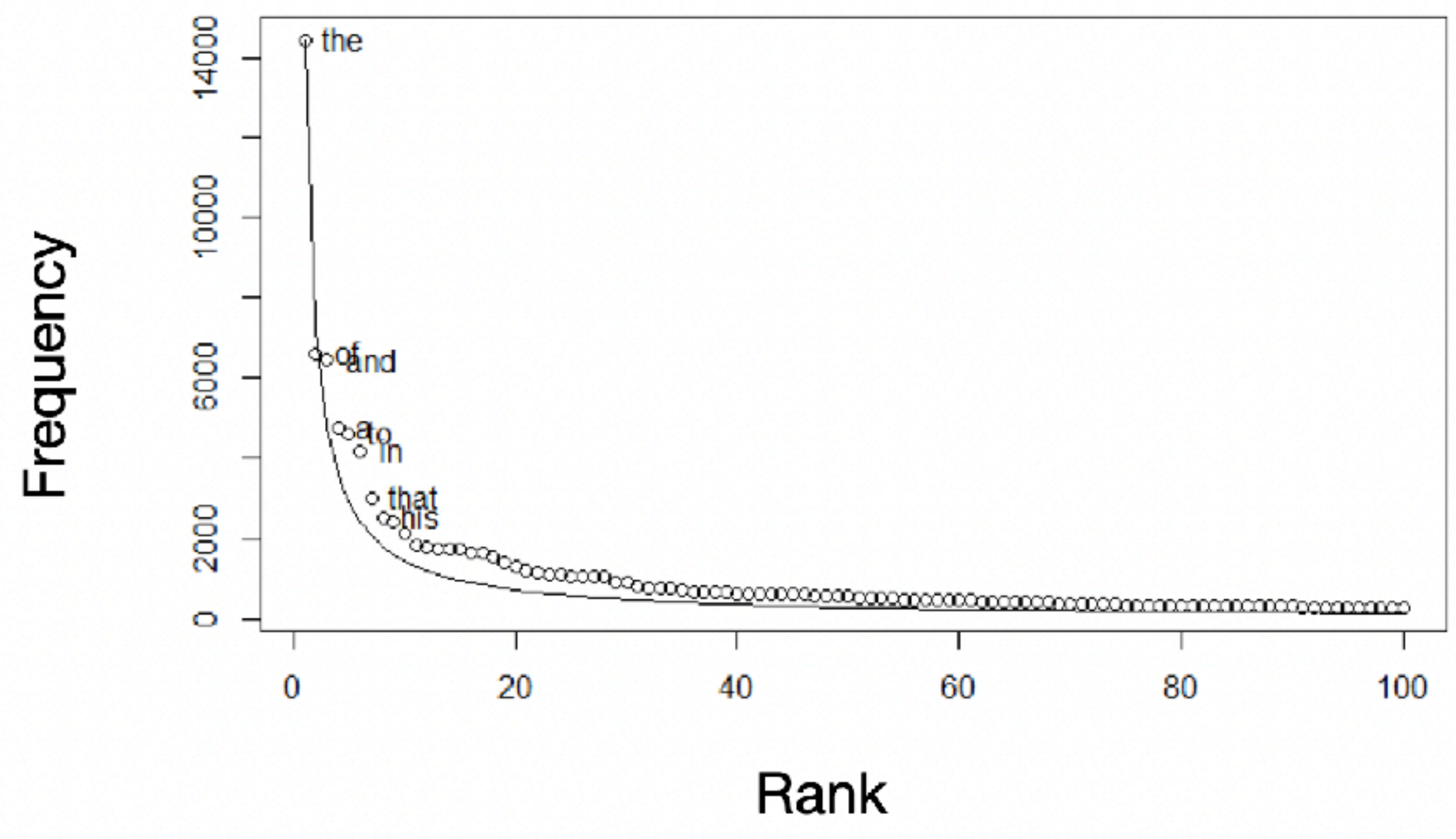
$$P(w_3 \mid w_1, w_2) \leftarrow \frac{\text{Count}(w_1, w_2, w_3)}{\text{Count}(w_1, w_2)}$$

We evaluate using **perplexity:**

$$\text{ppl}(S) = P(w_1, \ldots, w_n)^{-1/n} = \exp\Big(-\frac{1}{n}\sum_{i=1}^{n} \log P(w_i \mid w_1, \ldots, w_{i-1})\Big)$$

Smoothing

We want our models to accurately describe our languages. But, languages have a **long tail** and we have **finite data** → **Not all n-grams will be observed in the training data!**



$$freq \propto \frac{1}{rank}$$

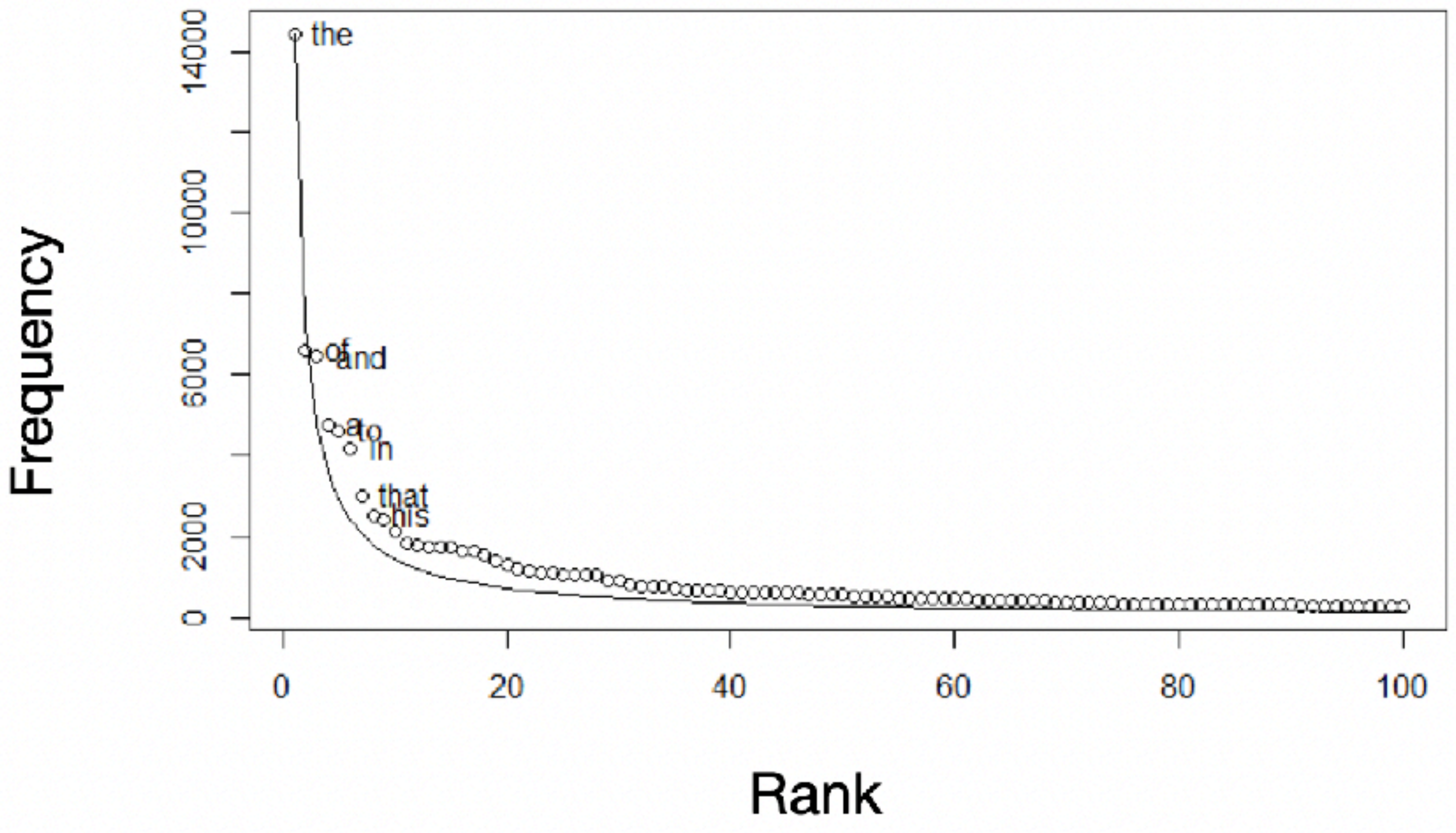
Zipf's Law

Smoothing

We want our models to accurately describe our languages. But, languages have a **long tail** and we have **finite data → Not all n-grams will be observed in the training data!**

How can we help our models compensate for this sparsity? **Smoothing!**

- Additive
- Discounting
- Back-off
- Interpolation



$$freq \propto \frac{1}{rank}$$

Zipf's Law

Smoothing

Additive smoothing (Laplace): add a small count to each n-gram

- Simplest form of smoothing: Just add α to all counts and renormalize!

- Max likelihood estimate for bigrams:

$$P(w_i|w_{i-1}) = \frac{C(w_{i-1}, w_i)}{C(w_{i-1})}$$

- After smoothing:

$$P(w_i|w_{i-1}) = \frac{C(w_{i-1}, w_i) \boxed{+ \alpha}}{C(w_{i-1}) \boxed{+ \alpha|V|}}$$

Naive Bayes: Summary

Naive Bayes: Summary

1. Given a document $d = w_1, \ldots, w_K$ and a set of classes $\mathscr{C} = \{c_1, \ldots, c_m\}$, we want to find the class c that maximizes $P(c \mid d)$.

Naive Bayes: Summary

1. Given a document $d = w_1, \ldots, w_K$ and a set of classes $\mathscr{C} = \{c_1, \ldots, c_m\}$, we want to find the class c that maximizes $P(c \mid d)$.
2. We don't know $P(c \mid d)$, but we know how to estimate $P(d \mid c)$ using a simple LM! → we can get $P(c \mid d)$ using Bayes' rule!
   1. $P(c \mid d) \propto P(d \mid c)P(c)$

Naive Bayes: Summary

1. Given a document $d = w_1, \ldots, w_K$ and a set of classes $\mathscr{C} = \{c_1, \ldots, c_m\}$, we want to find the class c that maximizes $P(c \mid d)$.

2. We don't know $P(c \mid d)$, but we know how to estimate $P(d \mid c)$ using a simple LM! → we can get $P(c \mid d)$ using Bayes' rule!

    1. $P(c \mid d) \propto P(d \mid c)P(c)$

3. Bayes rule requires us to estimate $P(c)$, we can do this just by counting the proportion of documents that are class c

Naive Bayes: Summary

1. Given a document $d = w_1, \ldots, w_K$ and a set of classes $\mathscr{C} = \{c_1, \ldots, c_m\}$, we want to find the class c that maximizes $P(c \mid d)$.
2. We don't know $P(c \mid d)$, but we know how to estimate $P(d \mid c)$ using a simple LM! $\rightarrow$ we can get $P(c \mid d)$ using Bayes' rule!
    1. $P(c \mid d) \propto P(d \mid c)P(c)$
3. Bayes rule requires us to estimate $P(c)$, we can do this just by counting the proportion of documents that are class c
4. To estimate $P(d \mid c)$ let's be lazy and choose the simplest possible LM that assume (Naively) that each word is independent - the unigram

Naive Bayes: Summary

1. Given a document $d = w_1, \ldots, w_K$ and a set of classes $\mathscr{C} = \{c_1, \ldots, c_m\}$, we want to find the class c that maximizes $P(c \mid d)$.
2. We don't know $P(c \mid d)$, but we know how to estimate $P(d \mid c)$ using a simple LM! → we can get $P(c \mid d)$ using Bayes' rule!
    1. $P(c \mid d) \propto P(d \mid c)P(c)$
3. Bayes rule requires us to estimate $P(c)$, we can do this just by counting the proportion of documents that are class c
4. To estimate $P(d \mid c)$ let's be lazy and choose the simplest possible LM that assume (Naively) that each word is independent - the unigram
5. Combine 3 + 4 and you can find the MAP estimate: $c_{MAP} = \arg\max_{c \in C} P(d \mid c)P(c)$

Logistic Regression: Intuition

Given a document $d = w_1, \ldots, w_K$ and a set of classes $\mathscr{C} = \{c_1, \ldots, c_m\}$, we want to find the class c_i that maximizes $P(c \mid d)$

Logistic Regression: Intuition

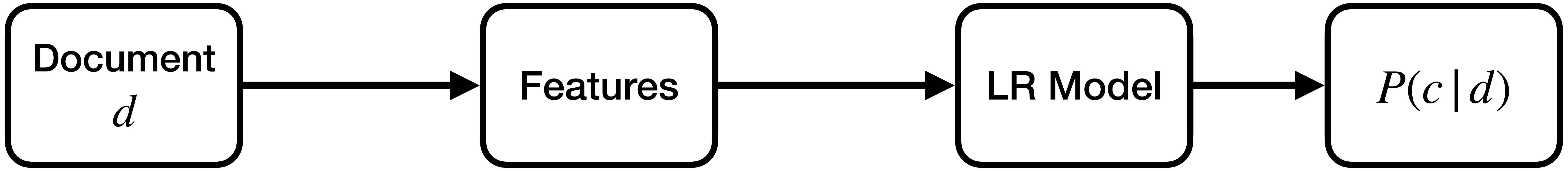
Given a document $d = w_1, \ldots, w_K$ and a set of classes $\mathscr{C} = \{c_1, \ldots, c_m\}$, we want to find the class c_i that maximizes $P(c \mid d)$

Compared to NB, with LR we take a more direct approach: directly compute $P(c \mid d)$ given a set of features constructed from the input document d.

Logistic Regression: Intuition

Given a document $d = w_1, \ldots, w_K$ and a set of classes $\mathscr{C} = \{c_1, \ldots, c_m\}$, we want to find the class c_i that maximizes $P(c \mid d)$

Compared to NB, with LR we take a more direct approach: directly compute $P(c \mid d)$ given a set of features constructed from the input document d.



Logistic Regression: Features

Given a document $d = w_1, \ldots, w_K$ and a set of classes $\mathscr{C} = \{c_1, \ldots, c_m\}$, we want to find the class c_i that maximizes $P(c \,|\, d)$

Compared to NB, with LR we take a more direct approach: directly compute $P(c \,|\, d)$ given a set of features constructed from the input document d.

Document d → Features → LR Model → $P(c \,|\, d)$

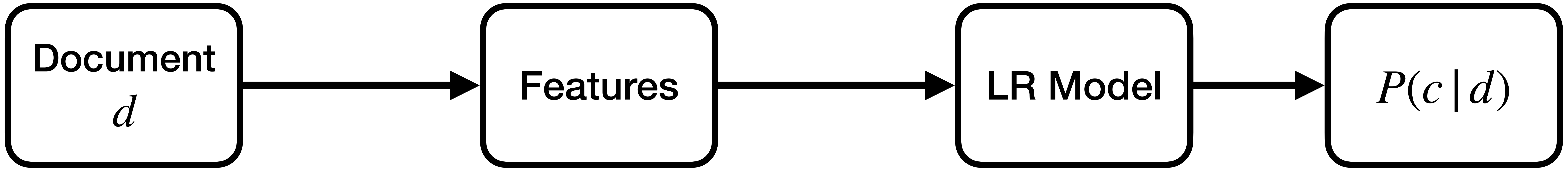
Var	Definition	Value
x_1	count(positive lexicon) $\in$ doc)	3
x_2	count(negative lexicon) $\in$ doc)	2
x_3	$\begin{cases} 1 & \text{if “no”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	1
x_4	count(1st and 2nd pronouns $\in$ doc)	3
x_5	$\begin{cases} 1 & \text{if “!”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	0
x_6	log(word count of doc)	$\ln(64) = 4.15$

This is the feature vector x_d for some input document d

Logistic Regression: Features

Given a document $d = w_1, \ldots, w_K$ and a set of classes $\mathscr{C} = \{c_1, \ldots, c_m\}$, we want to find the class c_i that maximizes $P(c \mid d)$

Compared to NB, with LR we take a more direct approach: directly compute $P(c \mid d)$ given a set of features constructed from the input document d.



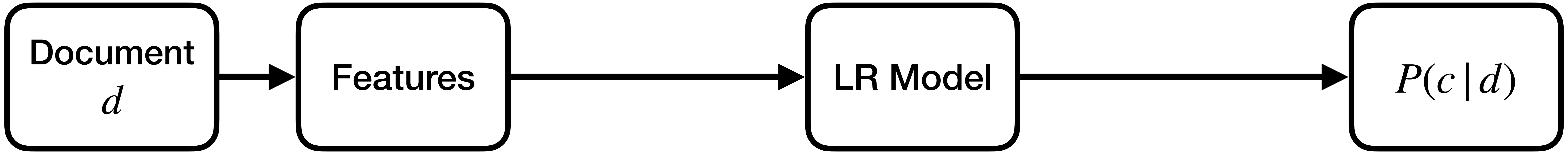
Var	Definition	Value
x_1	count(positive lexicon) $\in$ doc)	3
x_2	count(negative lexicon) $\in$ doc)	2
x_3	$\begin{cases} 1 & \text{if "no"} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	1
x_4	count(1st and 2nd pronouns $\in$ doc)	3
x_5	$\begin{cases} 1 & \text{if "!"} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	0
x_6	log(word count of doc)	$\ln(64) = 4.15$

The features to use is a design decision. A natural default is to use a vector $x_d \in \mathbb{R}^{|V|}$ where each dim is the counts of one word in the vocabulary. (BOW)

Logistic Regression: LR Model

Given a document $d = w_1, \ldots, w_K$ and a set of classes $\mathscr{C} = \{c_1, \ldots, c_m\}$, we want to find the class c_i that maximizes $P(c \mid d)$

Now given some feature vector x_d how do we turn this to a probability?

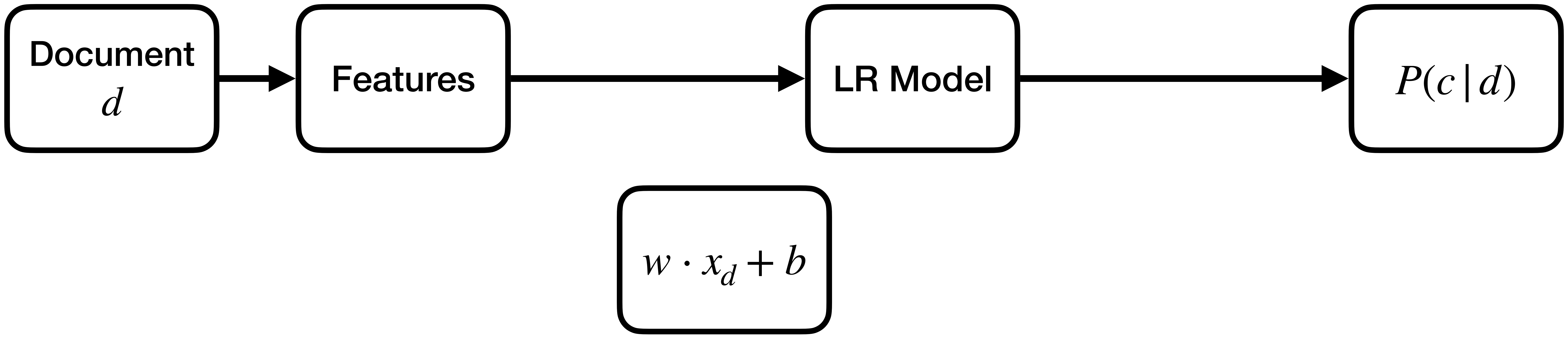


Logistic Regression: LR Model

Given a document $d = w_1, ..., w_K$ and a set of classes $\mathscr{C} = \{c_1, ..., c_m\}$, we want to find the class c_i that maximizes $P(c \mid d)$

Now given some feature vector x_d how do we turn this to a probability?

1. Convert the features to a number. The higher the number, the more confident we are that the document belongs to a class. We call these numbers **logits.**

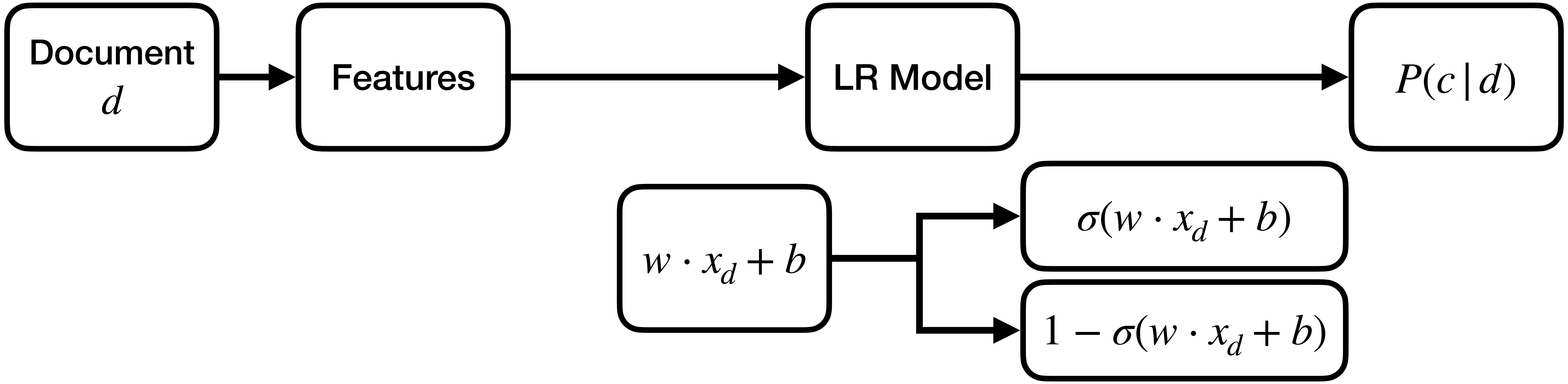


Logistic Regression: LR Model

Given a document $d = w_1, ..., w_K$ and a set of classes $\mathscr{C} = \{c_1, ..., c_m\}$, we want to find the class c_i that maximizes $P(c \mid d)$

Now given some feature vector x_d how do we turn this to a probability?

1. Convert the features to a number. The higher the number, the more confident we are that the document belongs to a class. We call these numbers **logits.**
2. Normalize the logits using sigmoid so we get a well-defined probability distribution.
    1. For more than 2 classes we use the softmax, which is the m > 2 generalization of sigmoid



Logistic Regression: LR Model

Given a document $d = w_1, \ldots, w_K$ and a set of classes $\mathscr{C} = \{c_1, \ldots, c_m\}$, we want to find the class c_i that maximizes $P(c \,|\, d)$

Now given some feature vector x_d how do we turn this to a probability?

1. Convert the features to a number. The higher the number, the more confident we are that the document belongs to a class. We call these numbers **logits.**
2. Normalize the logits using sigmoid so we get a well-defined probability distribution.
   1. For more than 2 classes we use the softmax, which is the m > 2 generalization of sigmoid (multinomial logistic regression)

$$P(c \mid d) = \frac{\exp(w_c \cdot x_d + b_c)}{\sum_{c' \in Y} \exp(w_{c'} \cdot x_d + b_{c'})}$$

Logistic Regression: Summary

Logistic Regression: Summary

1. Given a document $d = w_1, \ldots, w_K$ and a set of classes $\mathscr{C} = \{c_1, \ldots, c_m\}$, we want to find the class c that maximizes $P(c \mid d)$. Let's say we estimating $P(d \mid c)$ reliably is hard, we will need to estimate $P(c \mid d)$ directly.

Logistic Regression: Summary

1. Given a document $d = w_1, \ldots, w_K$ and a set of classes $\mathscr{C} = \{c_1, \ldots, c_m\}$, we want to find the class c that maximizes $P(c \mid d)$. Let's say we estimating $P(d \mid c)$ reliably is hard, we will need to estimate $P(c \mid d)$ directly.
2. Want to turn d into a vector x because then we can operate on it more conveniently.
   1. We can use a BOW, where each dim in $x \in \mathbb{R}^{|V|}$ is the # of times a word in V appears
   2. We can also be creative and add additional features we think are important (e.g. # of emojis in text)

Logistic Regression: Summary

1. Given a document $d = w_1, \ldots, w_K$ and a set of classes $\mathscr{C} = \{c_1, \ldots, c_m\}$, we want to find the class c that maximizes $P(c \mid d)$. Let's say we estimating $P(d \mid c)$ reliably is hard, we will need to estimate $P(c \mid d)$ directly.
2. Want to turn d into a vector x because then we can operate on it more conveniently.
    1. We can use a BOW, where each dim in $x \in \mathbb{R}^{|V|}$ is the # of times a word in V appears
    2. We can also be creative and add additional features we think are important (e.g. # of emojis in text)
3. Somehow we need to turn x into a single number, because $P(c \mid d)$ is a single number.
    1. Let's be as lazy as possible and just take a linear combination of the features: $w \cdot x + b$

Logistic Regression: Summary

1. Given a document $d = w_1, \ldots, w_K$ and a set of classes $\mathscr{C} = \{c_1, \ldots, c_m\}$, we want to find the class c that maximizes $P(c \mid d)$. Let's say we estimating $P(d \mid c)$ reliably is hard, we will need to estimate $P(c \mid d)$ directly.
2. Want to turn d into a vector x because then we can operate on it more conveniently.
    1. We can use a BOW, where each dim in $x \in \mathbb{R}^{|V|}$ is the # of times a word in V appears
    2. We can also be creative and add additional features we think are important (e.g. # of emojis in text)
3. Somehow we need to turn x into a single number, because $P(c \mid d)$ is a single number.
    1. Let's be as lazy as possible and just take a linear combination of the features: $w \cdot x + b$
4. Oh no! The linear combination might not be in $[0,1]$, so we normalize using sigmoid: $\sigma(x) = (1 + e^{-x})^{-1}$
    1. The probability for one class is $\sigma(w \cdot x + b)$, so the other class must have prob $1 - \sigma(w \cdot x + b)$

Logistic Regression: Summary

1. Given a document $d = w_1, \ldots, w_K$ and a set of classes $\mathscr{C} = \{c_1, \ldots, c_m\}$, we want to find the class c that maximizes $P(c \mid d)$. Let's say we estimating $P(d \mid c)$ reliably is hard, we will need to estimate $P(c \mid d)$ directly.
2. Want to turn d into a vector x because then we can operate on it more conveniently.
    1. We can use a BOW, where each dim in $x \in \mathbb{R}^{|V|}$ is the # of times a word in V appears
    2. We can also be creative and add additional features we think are important (e.g. # of emojis in text)
3. Somehow we need to turn x into a single number, because $P(c \mid d)$ is a single number.
    1. Let's be as lazy as possible and just take a linear combination of the features: $w \cdot x + b$
4. Oh no! The linear combination might not be in $[0,1]$, so we normalize using sigmoid: $\sigma(x) = (1 + e^{-x})^{-1}$
    1. The probability for one class is $\sigma(w \cdot x + b)$, so the other class must have prob $1 - \sigma(w \cdot x + b)$
5. Given our model, we can estimate the probability of a train set under the model $P(\mathscr{D})$
    1. We will set w, b so that $P(\mathscr{D})$ = $\Pi_i P(c_i \mid d_i)$ is maximal (MLE principle)
    2. For stability and convenience we can take the $\log$ to minimize $-\sum_i \log P(c_i \mid d_i)$ this is CE loss

Logistic Regression: Summary

1. Given a document $d = w_1, \ldots, w_K$ and a set of classes $\mathscr{C} = \{c_1, \ldots, c_m\}$, we want to find the class c that maximizes $P(c \mid d)$. Let's say we estimating $P(d \mid c)$ reliably is hard, we will need to estimate $P(c \mid d)$ directly.
2. Want to turn d into a vector x because then we can operate on it more conveniently.
    1. We can use a BOW, where each dim in $x \in \mathbb{R}^{|V|}$ is the # of times a word in V appears
    2. We can also be creative and add additional features we think are important (e.g. # of emojis in text)
3. Somehow we need to turn x into a single number, because $P(c \mid d)$ is a single number.
    1. Let's be as lazy as possible and just take a linear combination of the features: $w \cdot x + b$
4. Oh no! The linear combination might not be in $[0,1]$, so we normalize using sigmoid: $\sigma(x) = (1 + e^{-x})^{-1}$
    1. The probability for one class is $\sigma(w \cdot x + b)$, so the other class must have prob $1 - \sigma(w \cdot x + b)$
5. Given our model, we can estimate the probability of a train set under the model $P(\mathscr{D})$
    1. We will set w, b so that $P(\mathscr{D}) = \Pi_i P(c_i \mid d_i)$ is maximal (MLE principle)
    2. For stability and convenience we can take the $\log$ to minimize $-\sum_i \log P(c_i \mid d_i)$ this is CE loss
6. We can then use GD to minimize the CE loss! Since the function is convex, we will converge to the optimum.

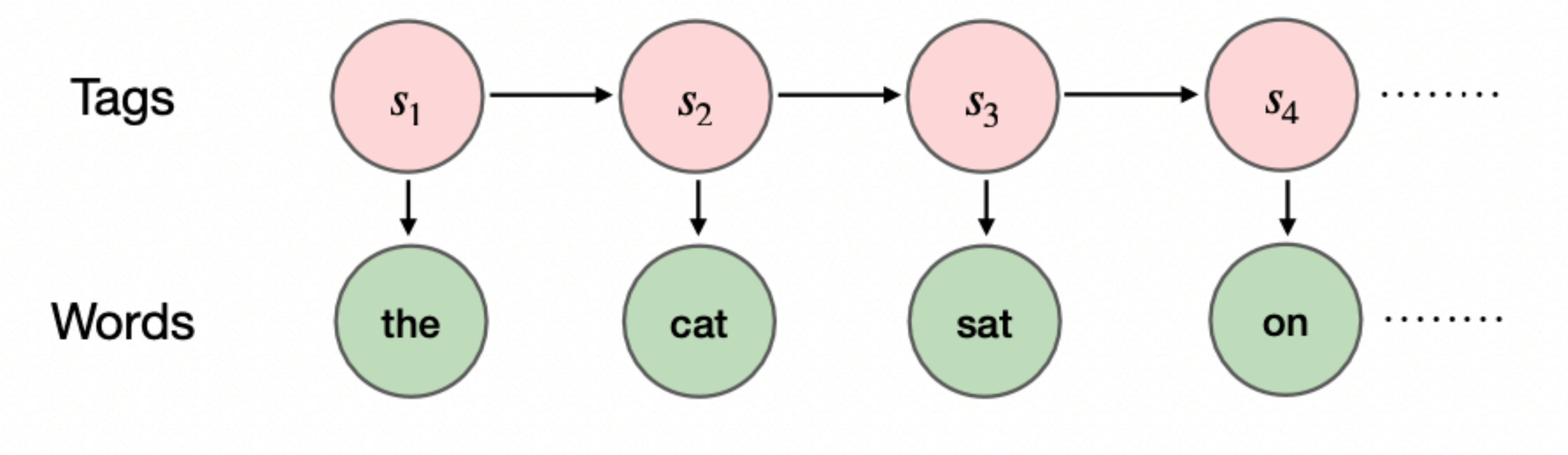
Agenda

- HMM
- Viterbi Algorithm
- MEMM

Agenda

- HMM
- Viterbi Algorithm
- MEMM

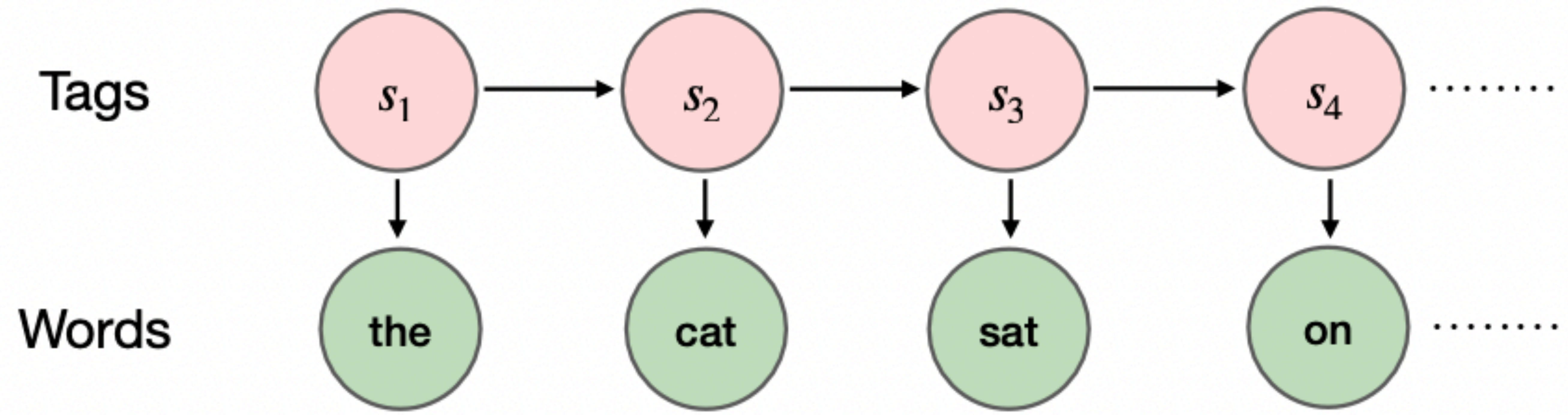
NB vs HMM



Generative

	Generative
Text classification	Naive Bayes: $P(c)P(d \mid c)$
Sequence prediction	HMM: $P(s_1, \ldots, s_n)P(o_1, \ldots, o_n \mid s_1, \ldots, s_n)$

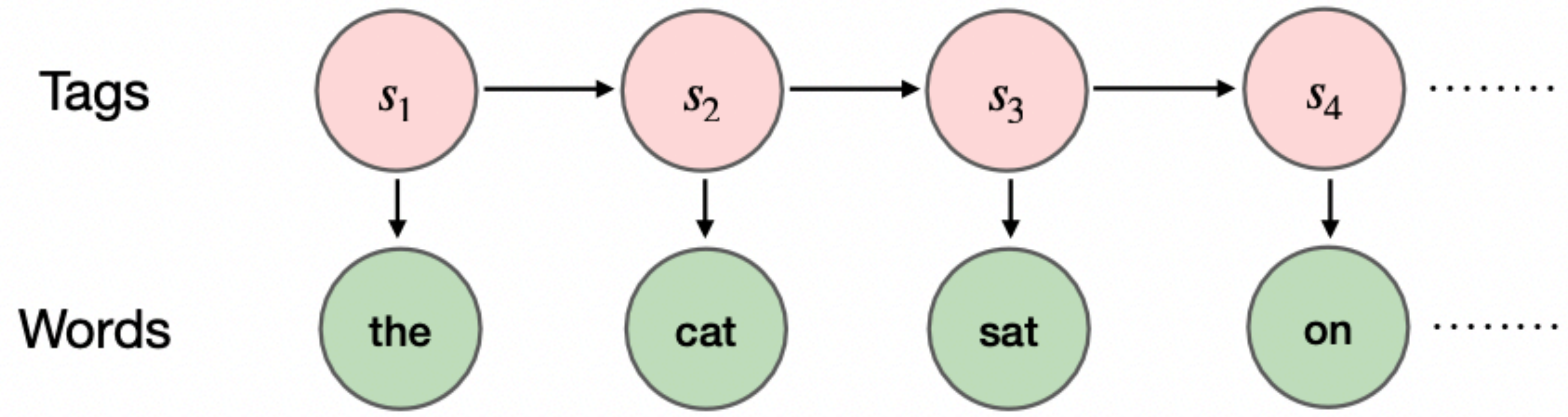
NB vs HMM



$$c_{\text{MAP}} = \text{argmax}_{c \in C} P(c \mid d)$$

$$\hat{S} = \arg\max_{S} P(S \mid O)$$

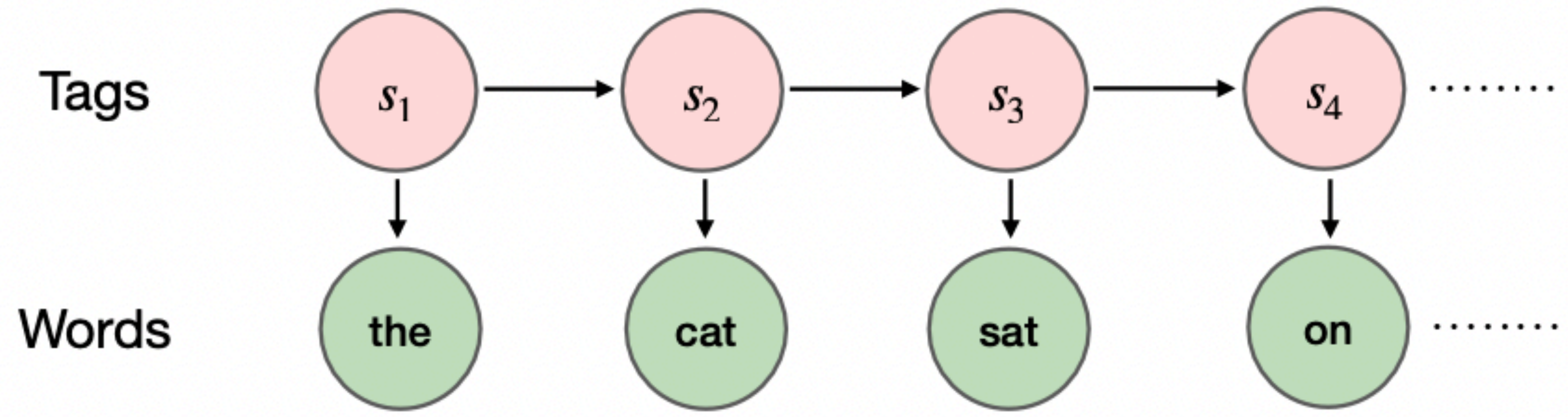
NB vs HMM



$$c_{\text{MAP}} = \text{argmax}_{c \in C} P(c \mid d)$$
$$= \text{argmax}_{c \in C} \frac{P(d \mid c)P(c)}{P(d)}$$

$$\hat{S} = \arg\max_S P(S \mid O)$$
$$= \arg\max_S \frac{P(O \mid S)P(S)}{P(O)}$$

NB vs HMM



$$c_{\text{MAP}} = \operatorname{argmax}_{c \in C} P(c \mid d)$$

$$= \operatorname{argmax}_{c \in C} \frac{P(d \mid c)P(c)}{P(d)}$$

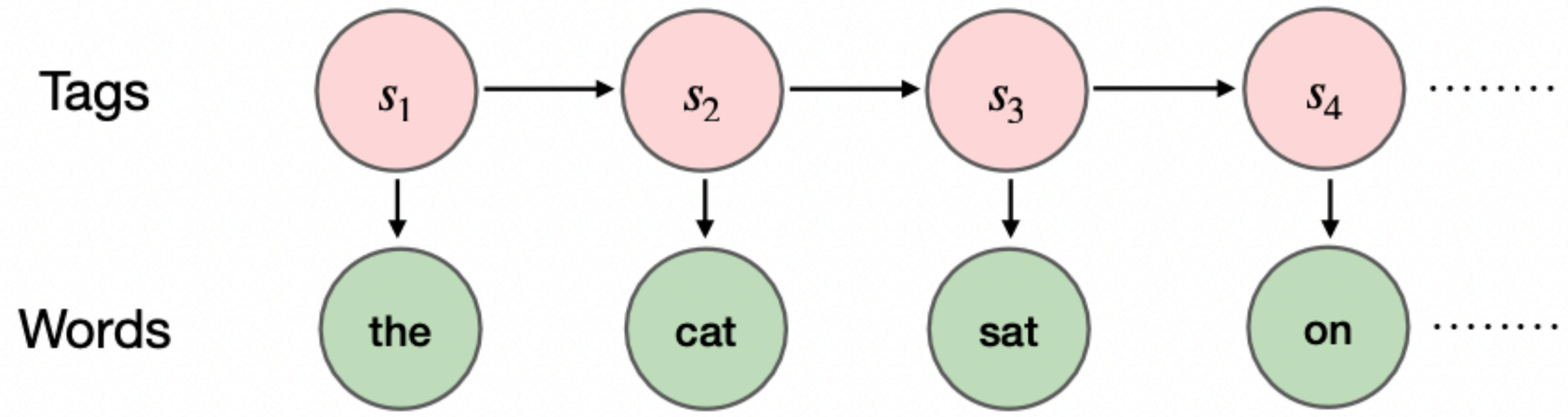
$$= \operatorname{argmax}_{c \in C} P(d \mid c)P(c)$$

$$\hat{S} = \arg\max_{S} P(S \mid O)$$

$$= \arg\max_{S} \frac{P(O \mid S)P(S)}{P(O)}$$

$$= \arg\max_{S} P(O \mid S)P(S)$$

NB vs HMM



$$c_{\text{MAP}} = \text{argmax}_{c \in C} P(c \mid d)$$

$$= \text{argmax}_{c \in C} \frac{P(d \mid c)P(c)}{P(d)}$$

$$= \text{argmax}_{c \in C} P(d \mid c)P(c)$$

$$= \text{argmax}_{c \in C} P(c) \prod_{i=1}^{K} P(w_i \mid c)$$

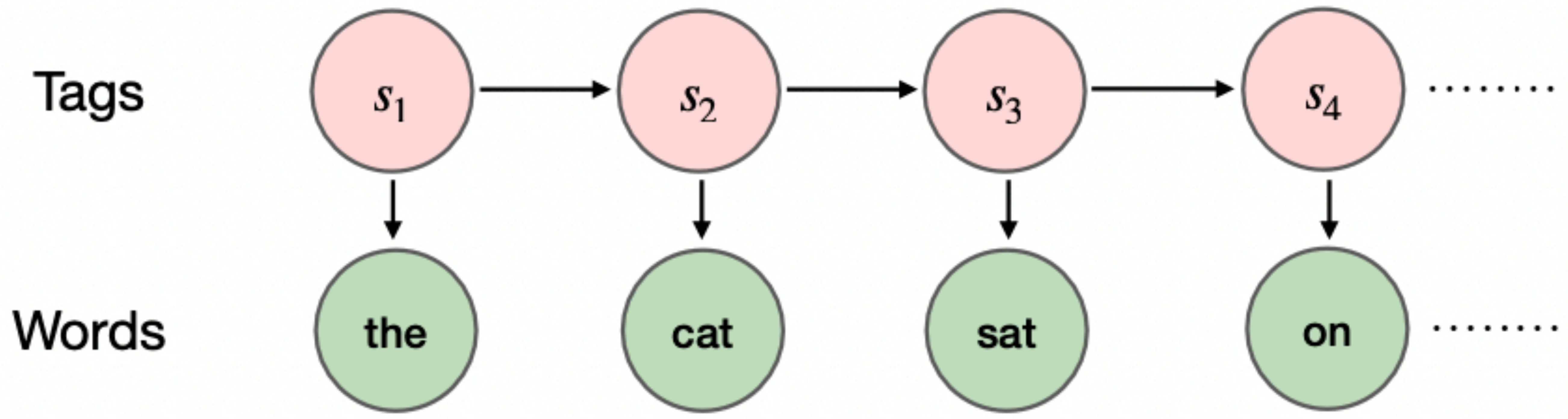
$$\hat{S} = \arg\max_{S} P(S \mid O)$$

$$= \arg\max_{S} \frac{P(O \mid S)P(S)}{P(O)}$$

$$= \arg\max_{S} P(O \mid S)P(S)$$

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

NB vs HMM



$$c_{\text{MAP}} = \text{argmax}_{c \in C} P(c \mid d)$$

$$= \text{argmax}_{c \in C} \frac{P(d \mid c)P(c)}{P(d)}$$

$$= \text{argmax}_{c \in C} P(d \mid c)P(c)$$

$$= \text{argmax}_{c \in C} P(c) \prod_{i=1}^{K} P(w_i \mid c)$$

$$\hat{S} = \arg\max_{S} P(S \mid O)$$

$$= \arg\max_{S} \frac{P(O \mid S)P(S)}{P(O)}$$

$$= \arg\max_{S} P(O \mid S)P(S)$$

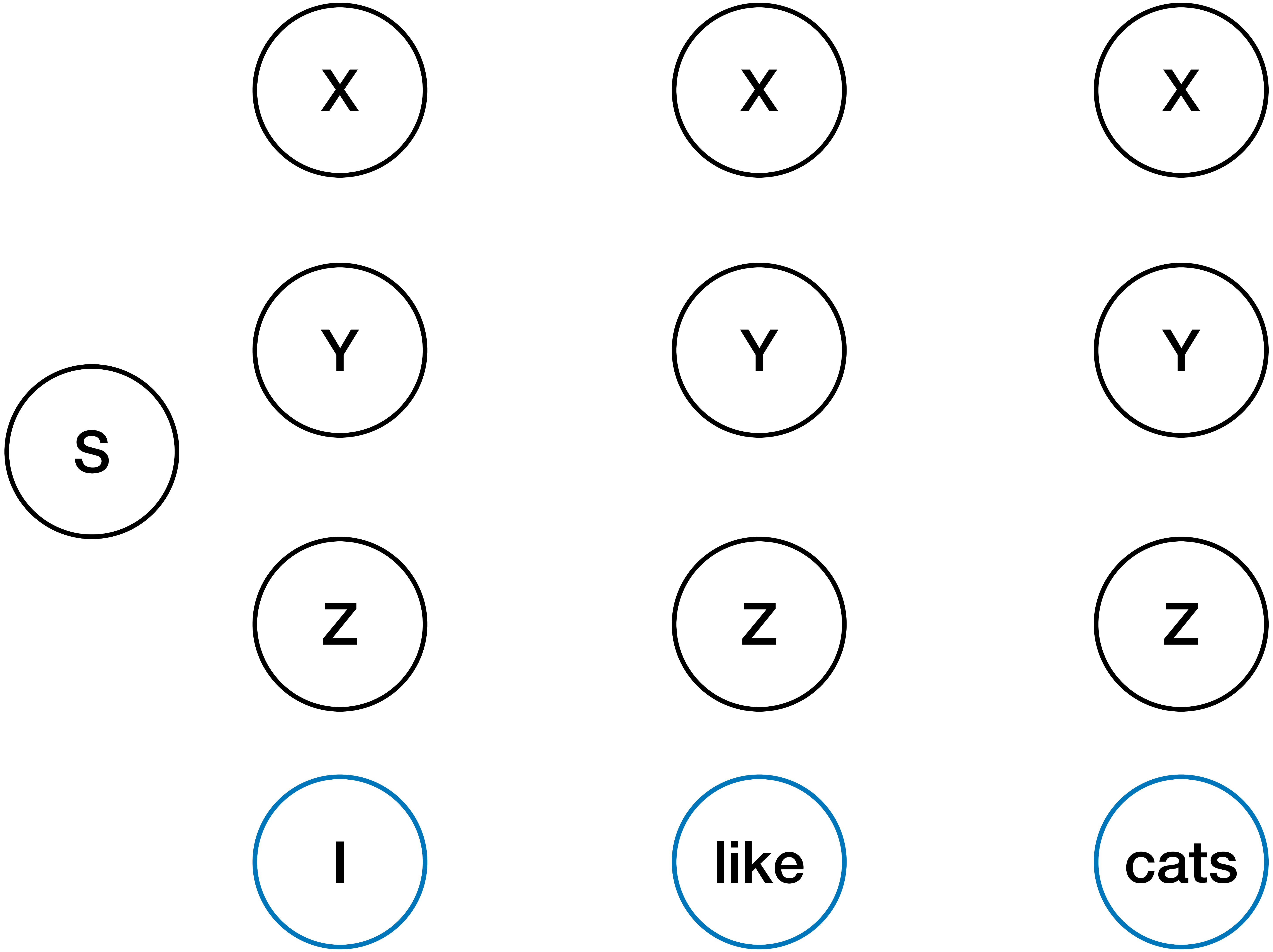
$$= \arg\max_{\boxed{s_1,\ldots,s_n}} \prod_{i=1}^{n} P(s_i \mid s_{i-1})P(o_i \mid s_i)$$

Viterbi Algorithm

Agenda

- HMM
- Viterbi Algorithm
- MEMM

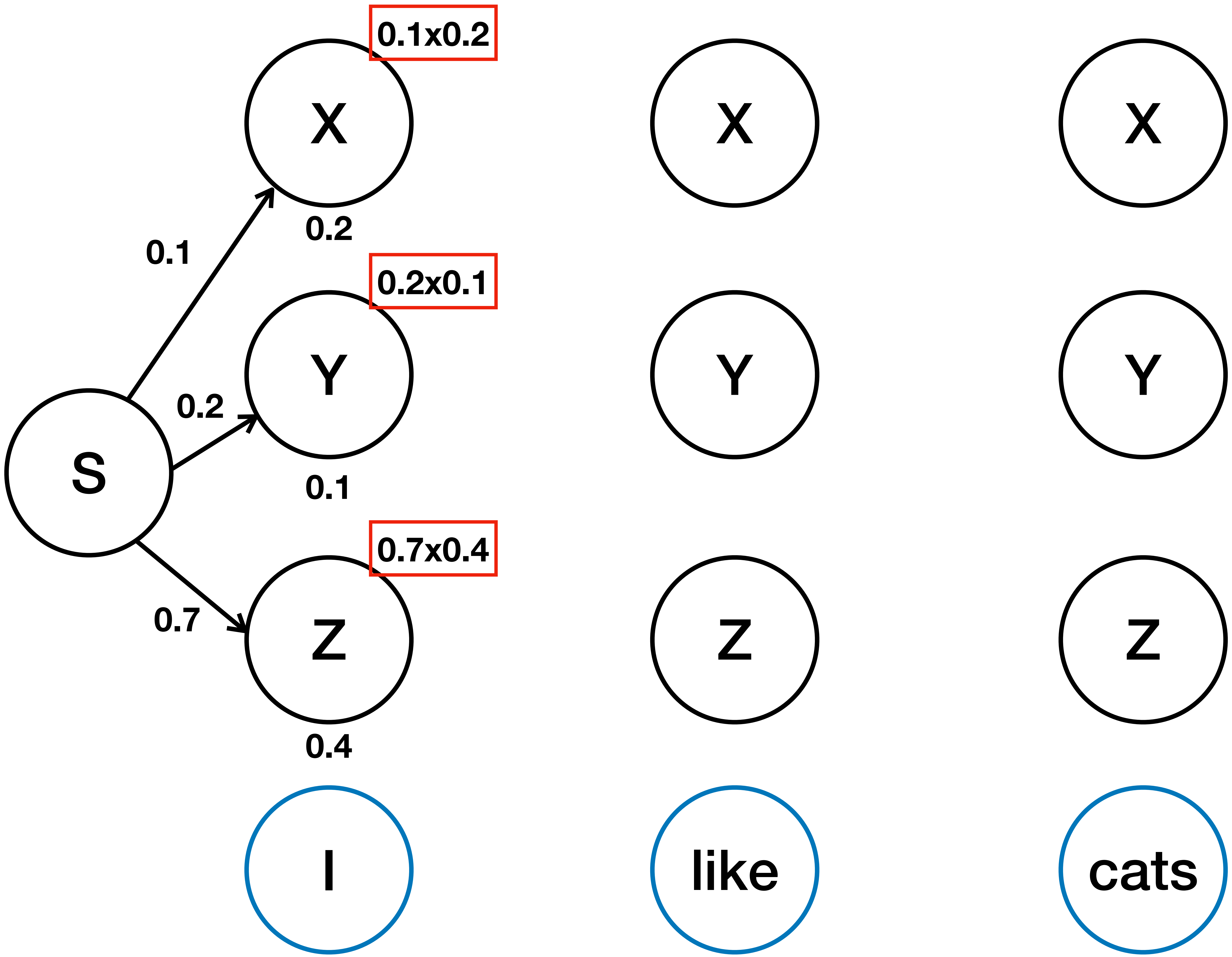
Viterbi Algorithm



	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

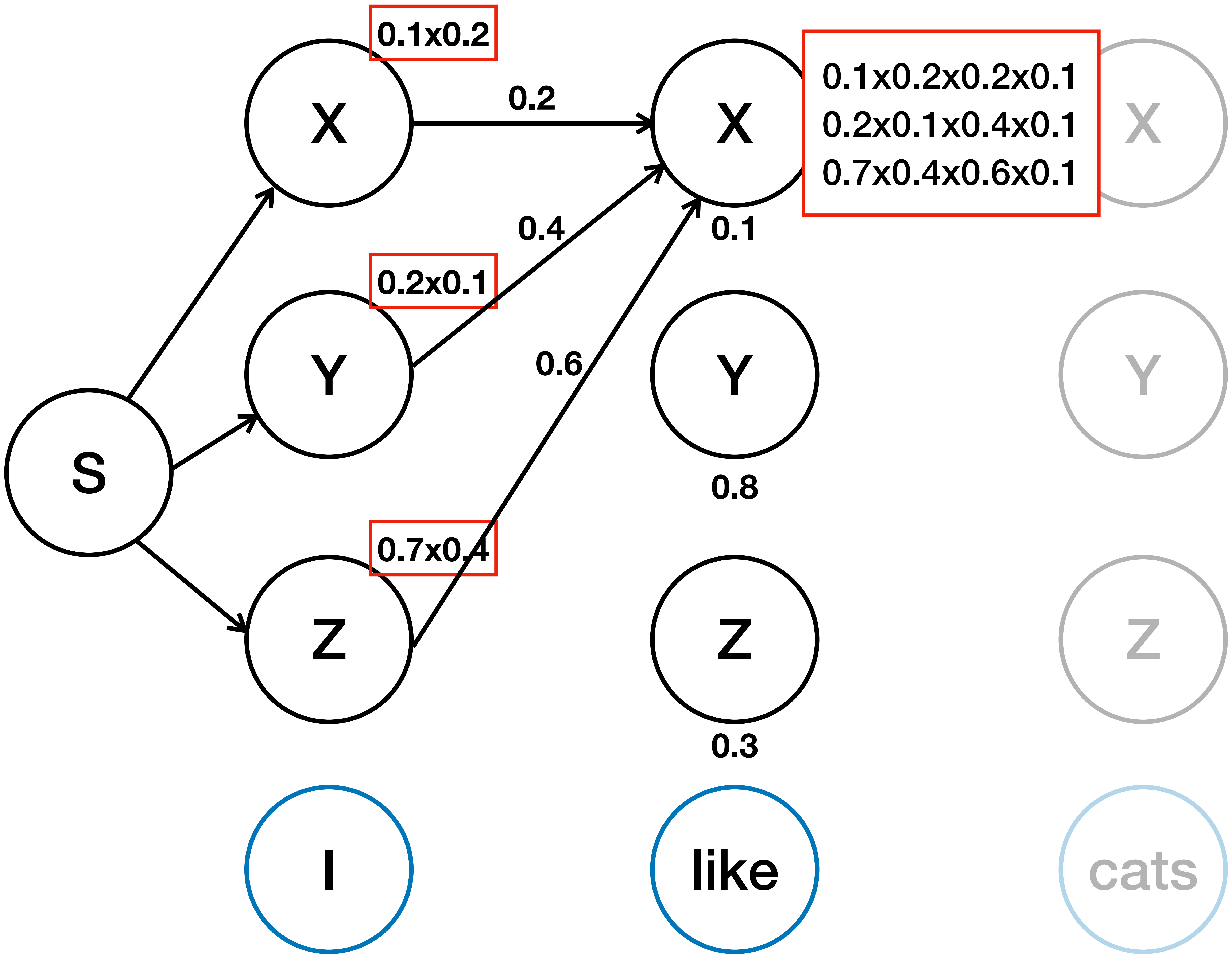
Viterbi Algorithm



	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

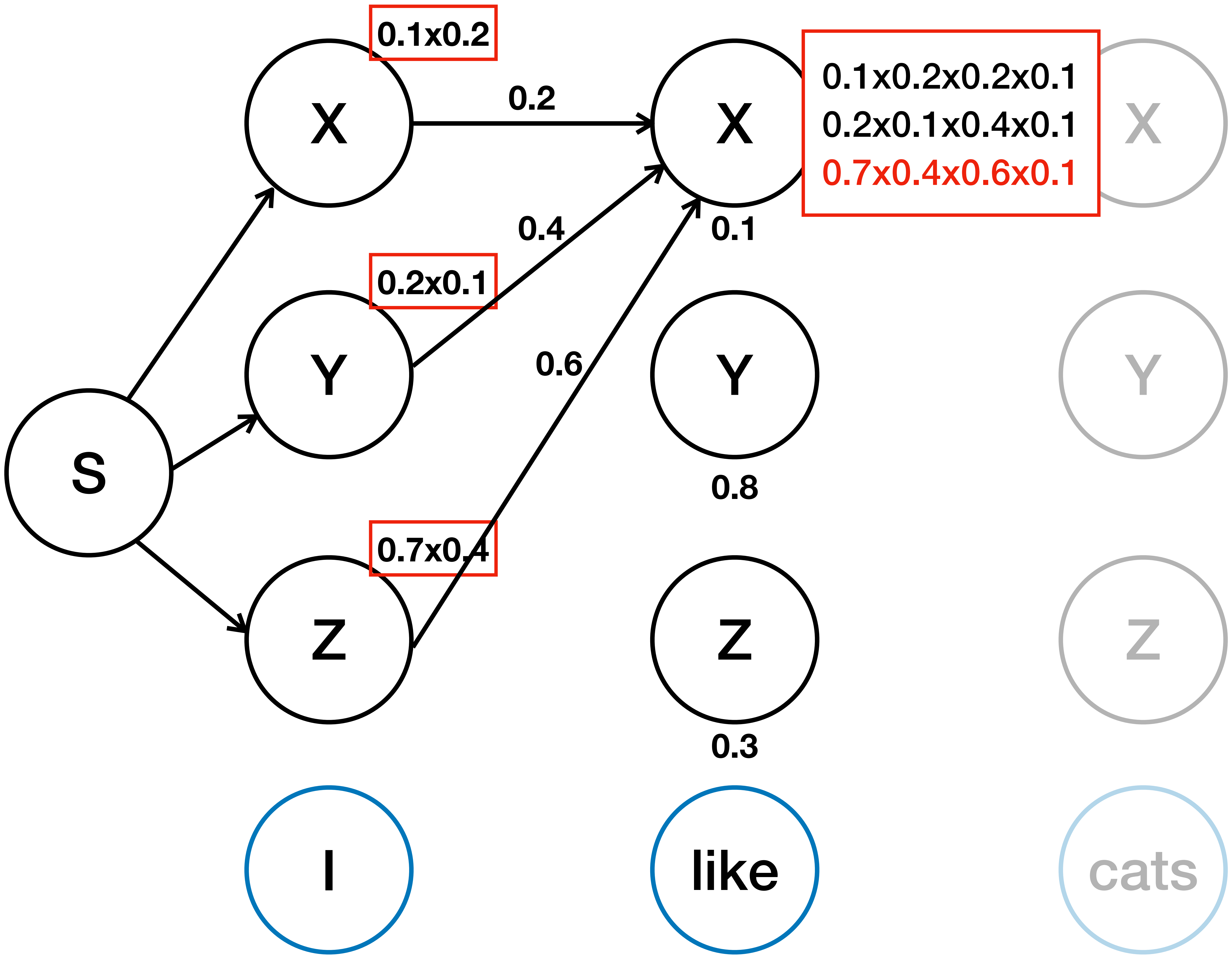
Viterbi Algorithm



	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

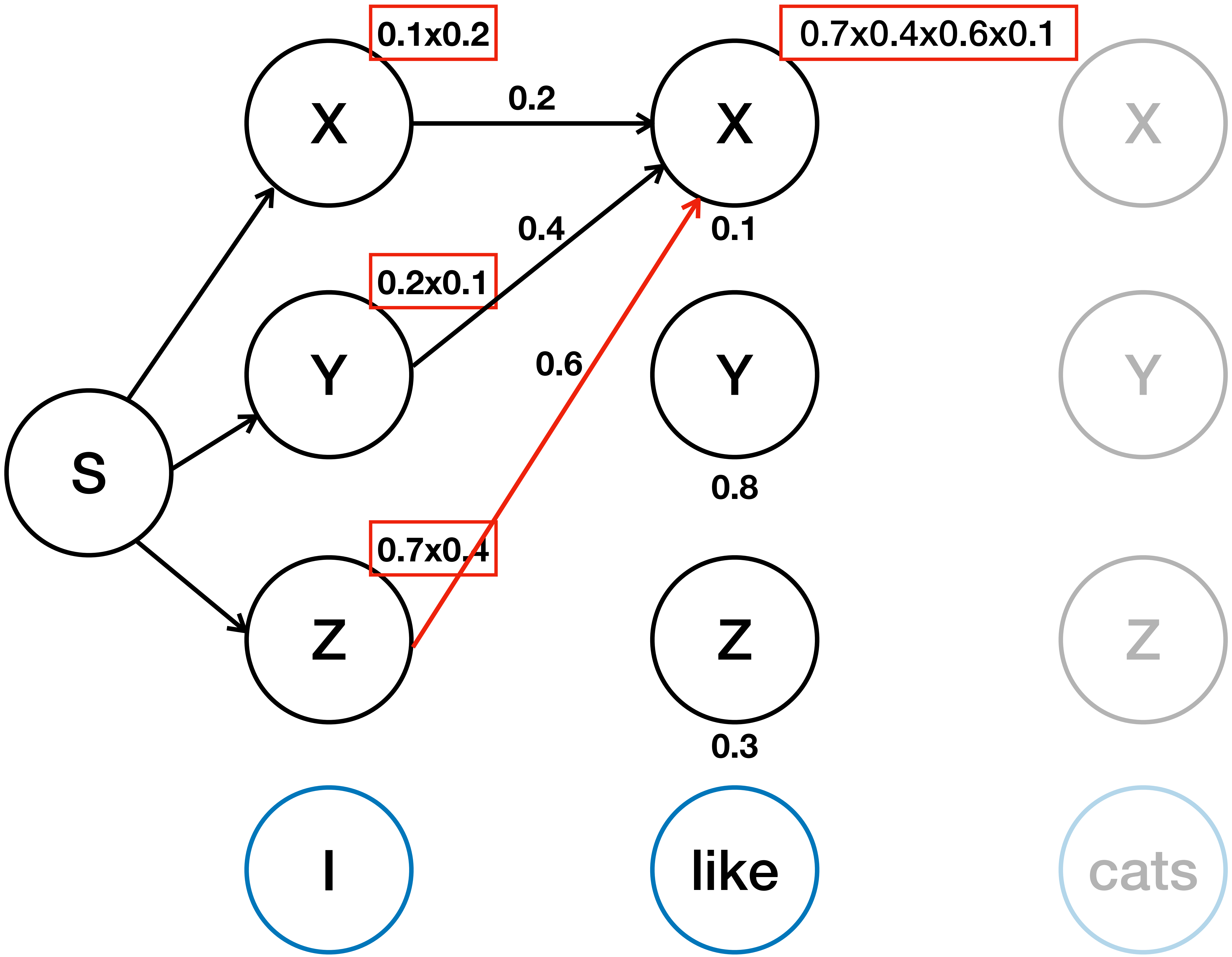
Viterbi Algorithm



	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

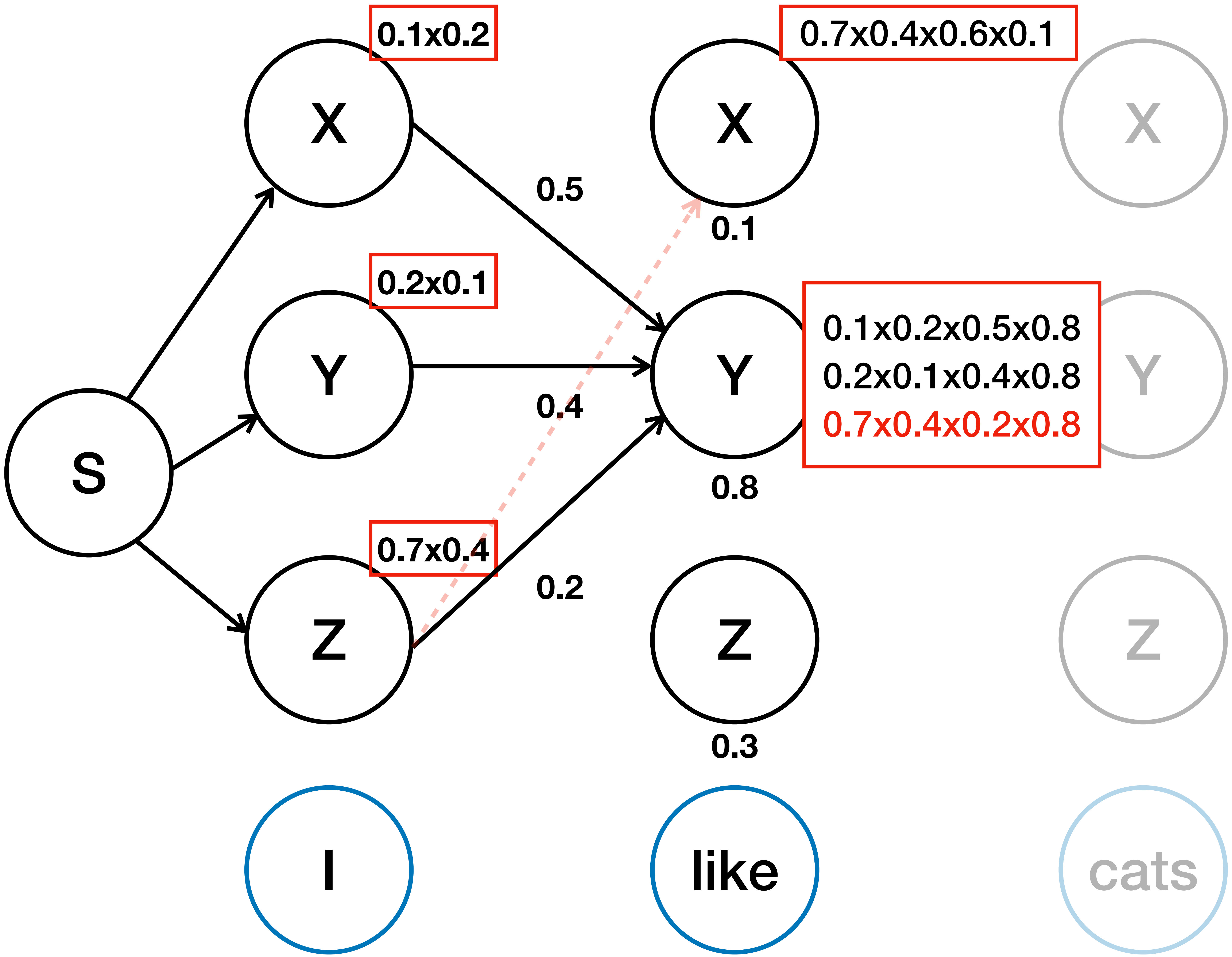
Viterbi Algorithm



	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

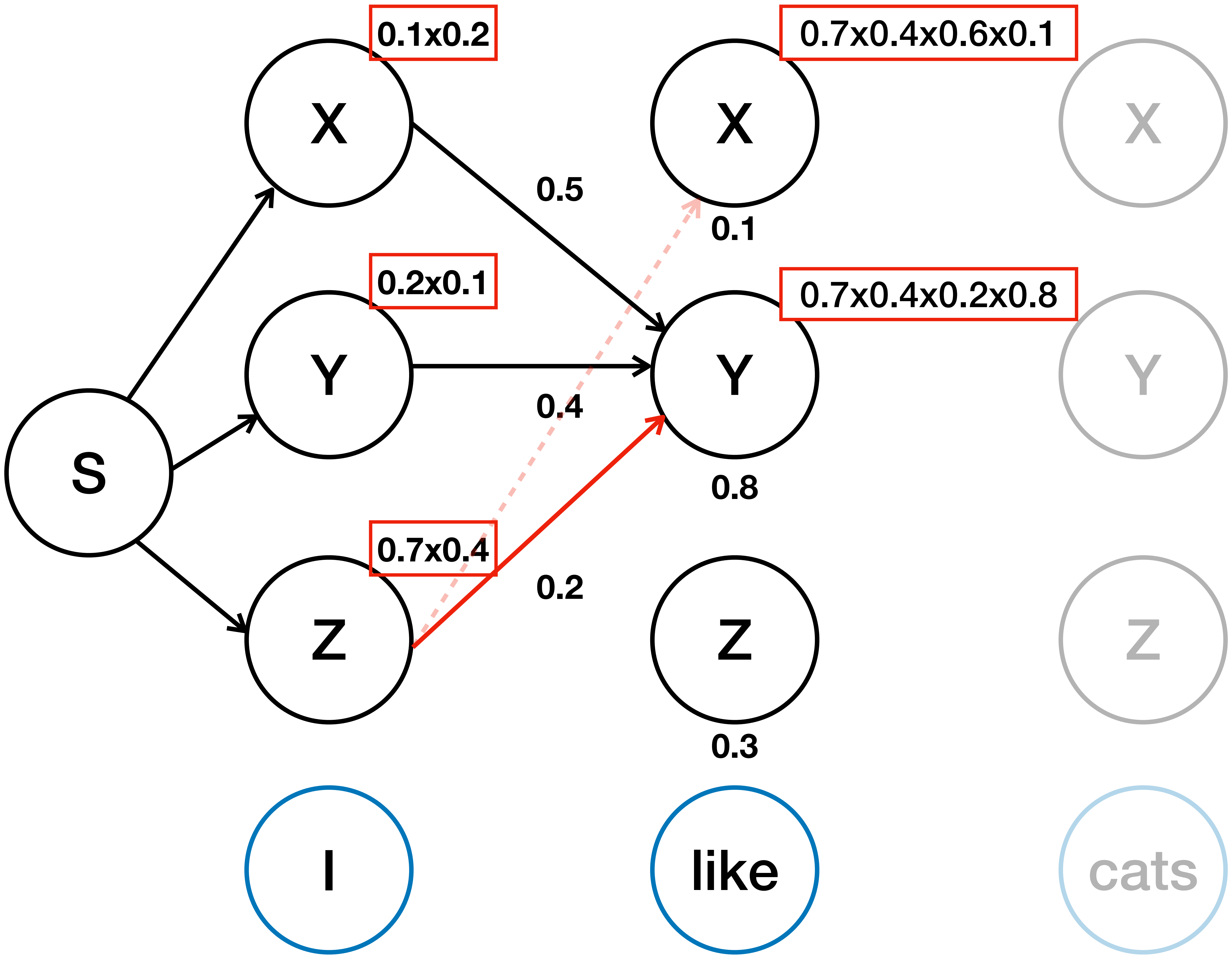
Viterbi Algorithm



	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

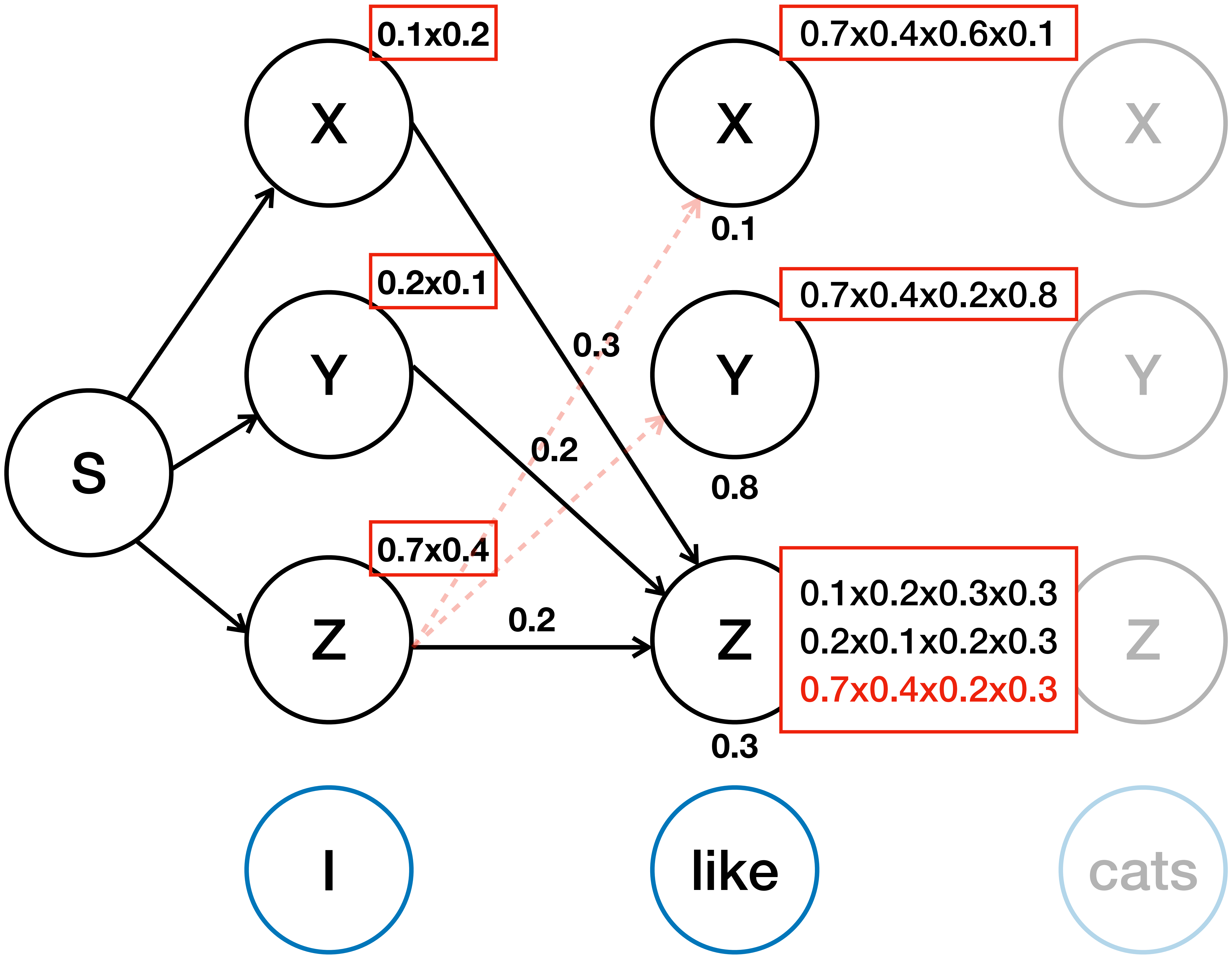
Viterbi Algorithm



	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

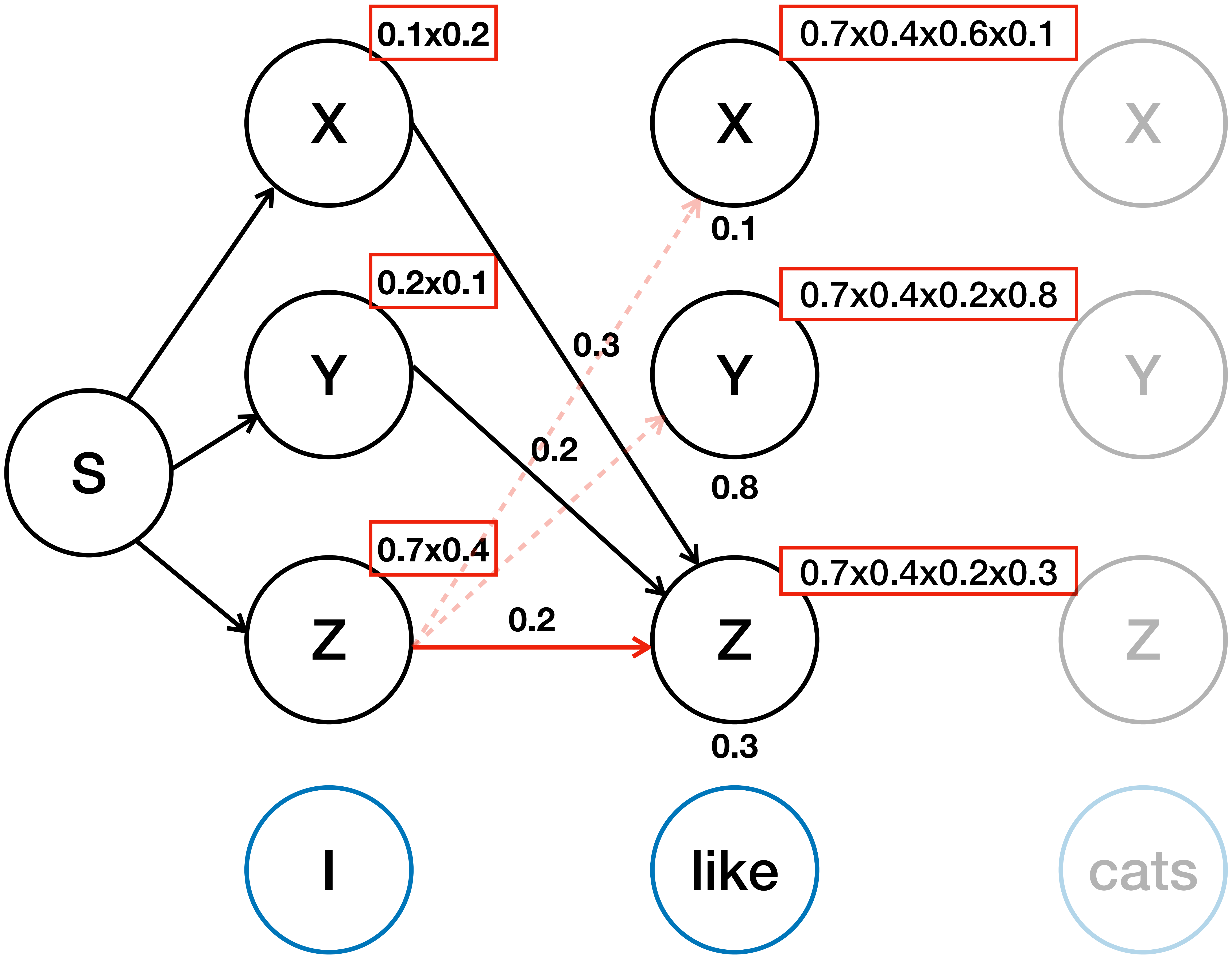
Viterbi Algorithm



	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

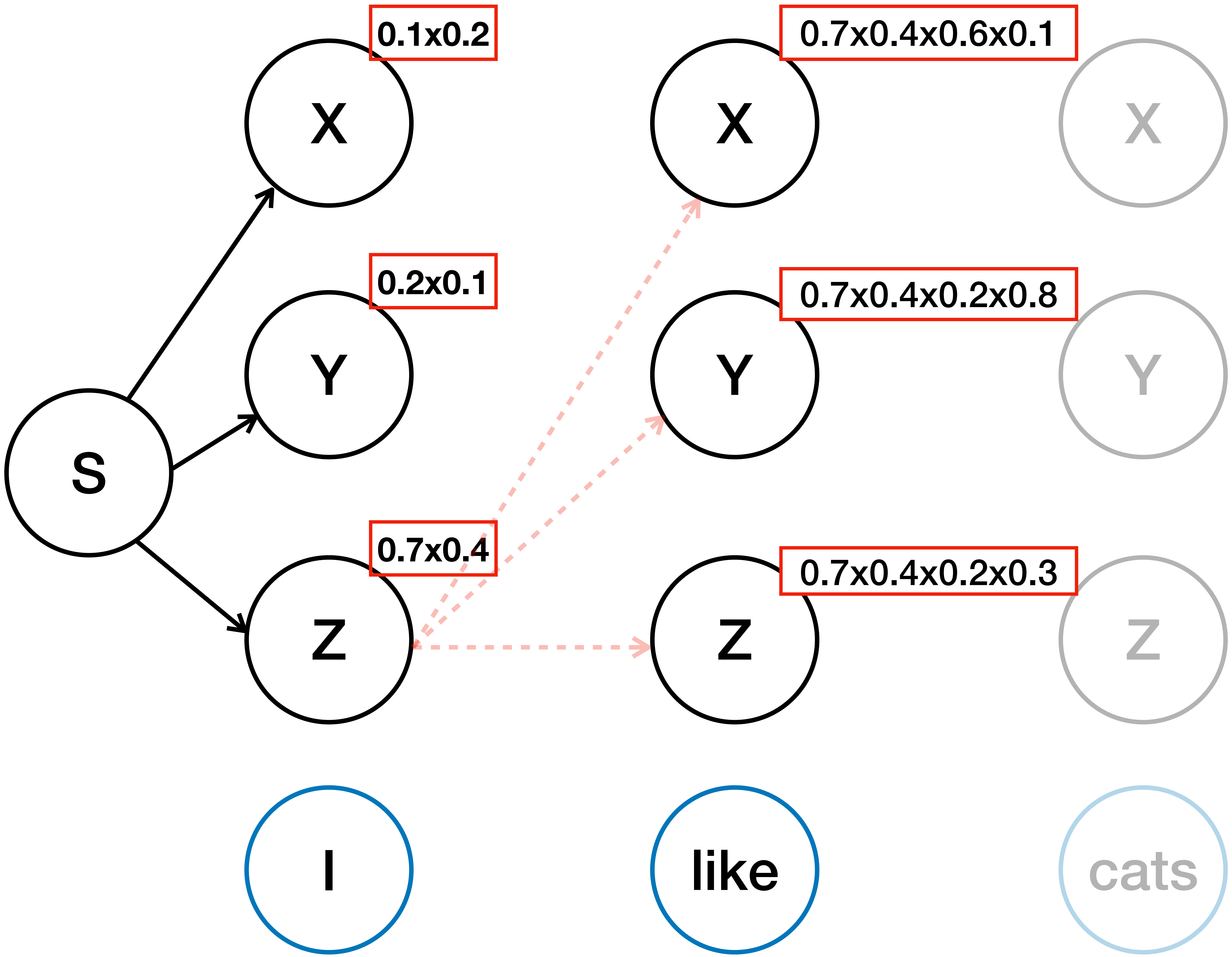
Viterbi Algorithm



	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

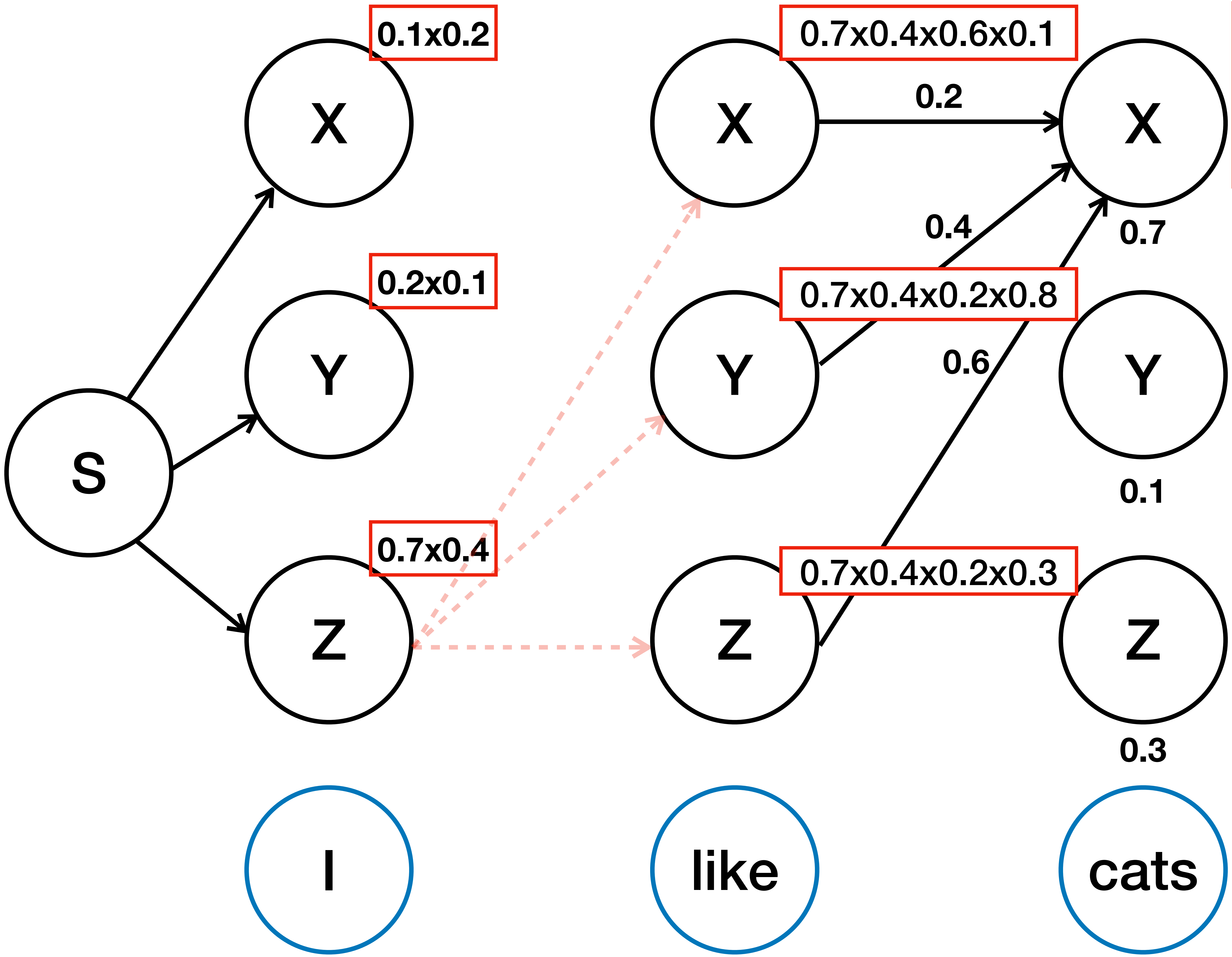
Viterbi Algorithm



	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

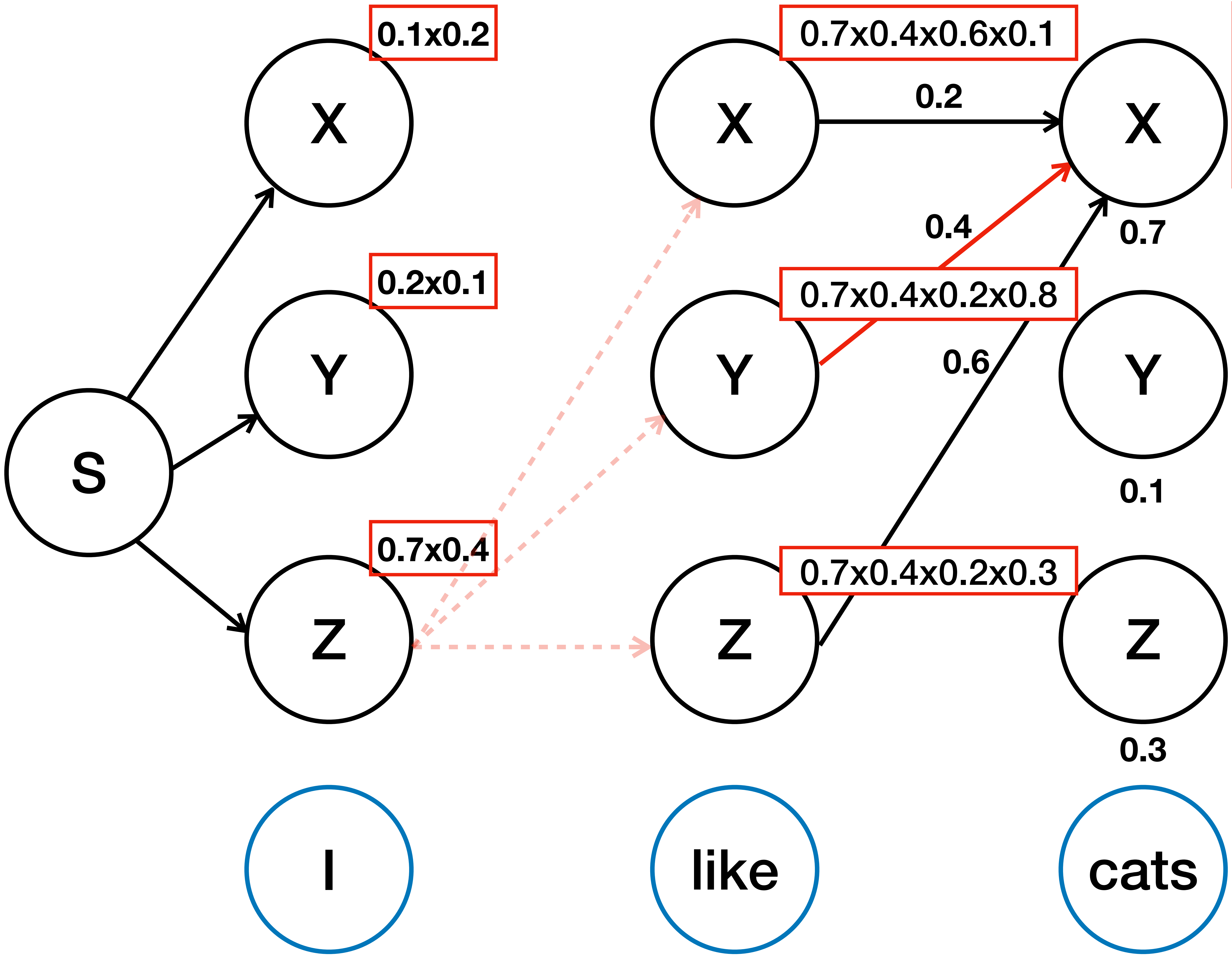


0.7x0.4x0.6x0.1x0.2x0.7 = 0.0023
0.7x0.4x0.2x0.8x0.4x0.7 = 0.0125
0.7x0.4x0.2x0.3x0.6x0.7 = 0.0070

		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

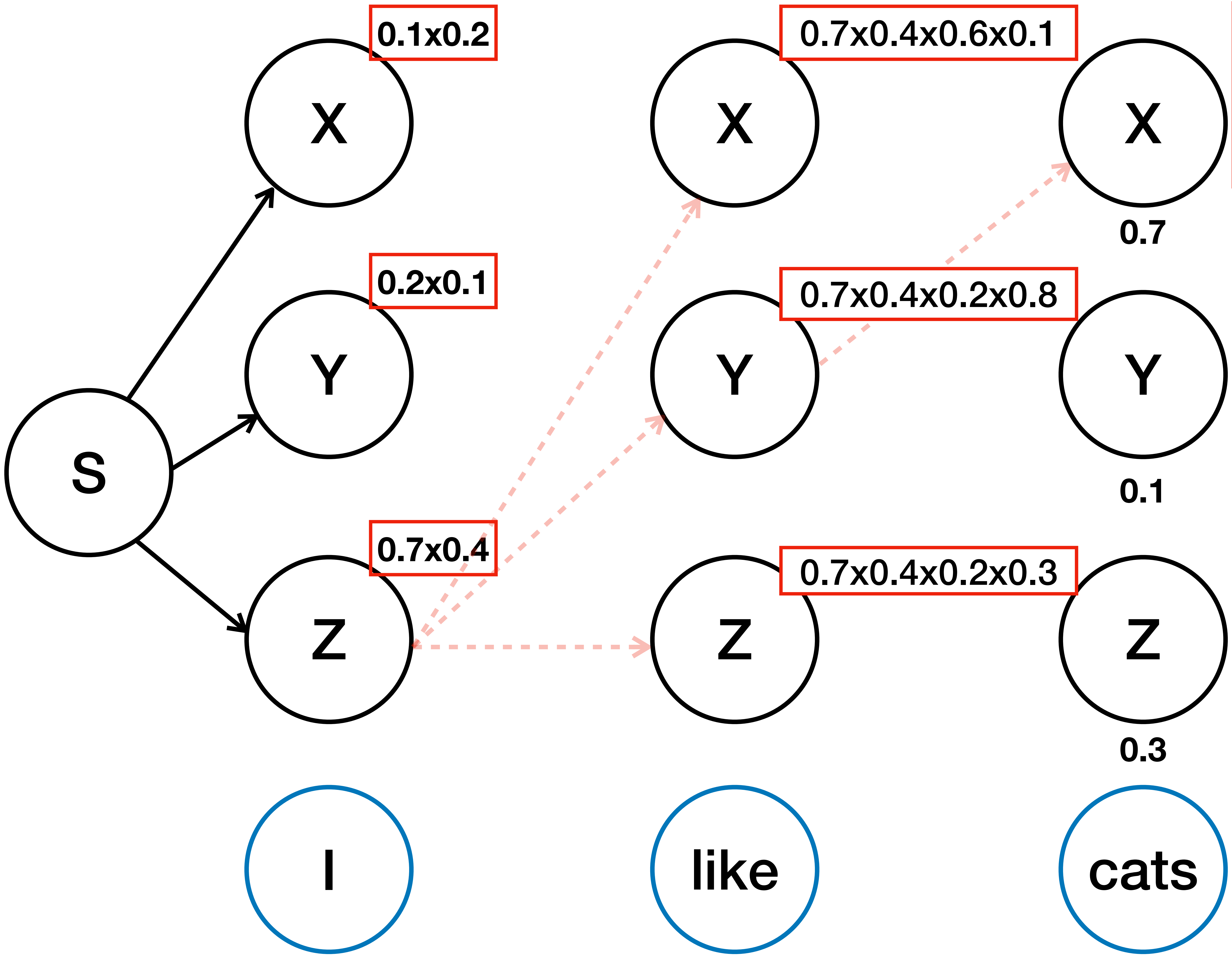


0.7x0.4x0.6x0.1x0.2x0.7 = 0.0023
0.7x0.4x0.2x0.8x0.4x0.7 = 0.0125
0.7x0.4x0.2x0.3x0.6x0.7 = 0.0070

		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

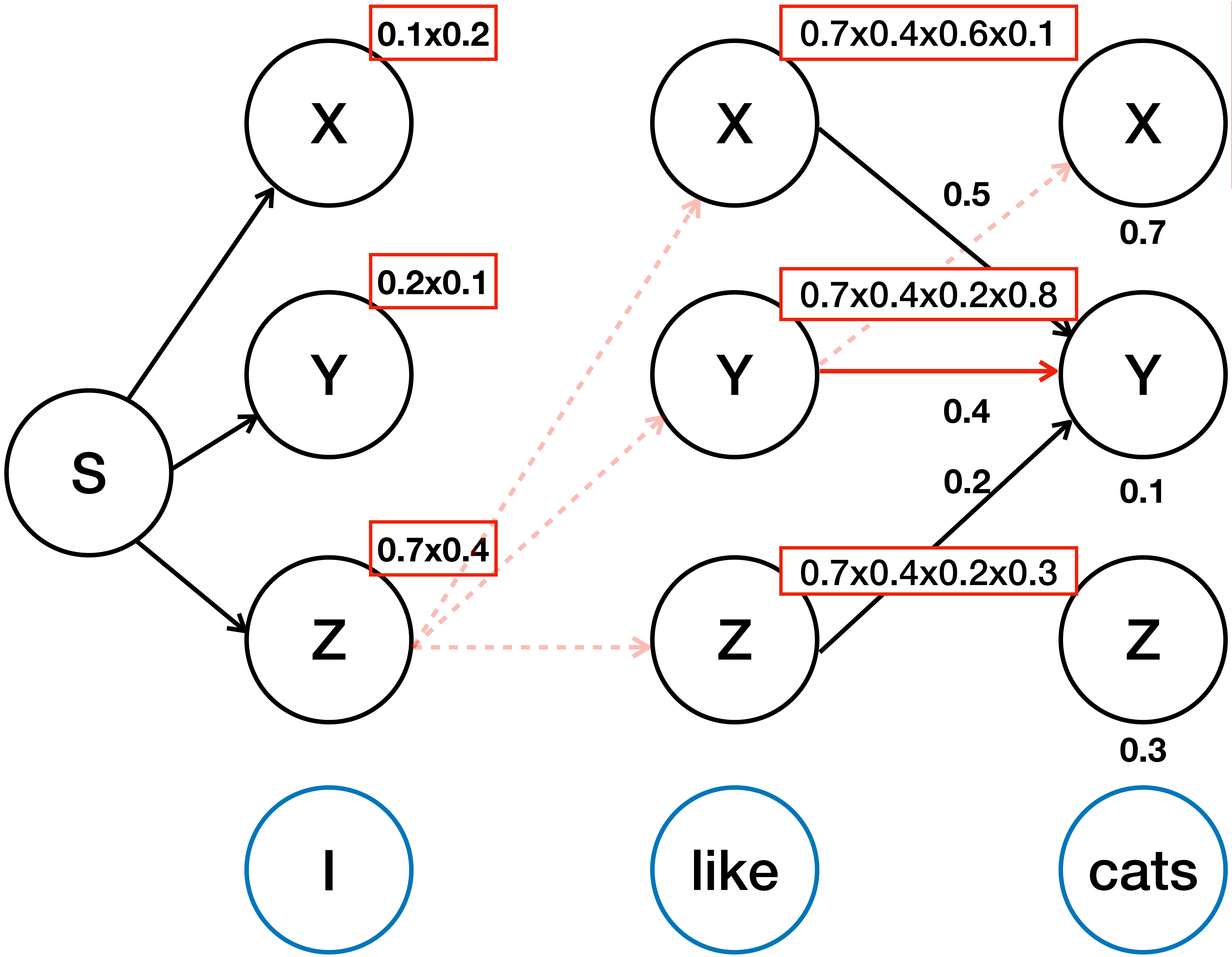


0.7x0.4x0.6x0.1x0.2x0.7 = 0.0023
0.7x0.4x0.2x0.8x0.4x0.7 = 0.0125
0.7x0.4x0.2x0.3x0.6x0.7 = 0.0070

		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



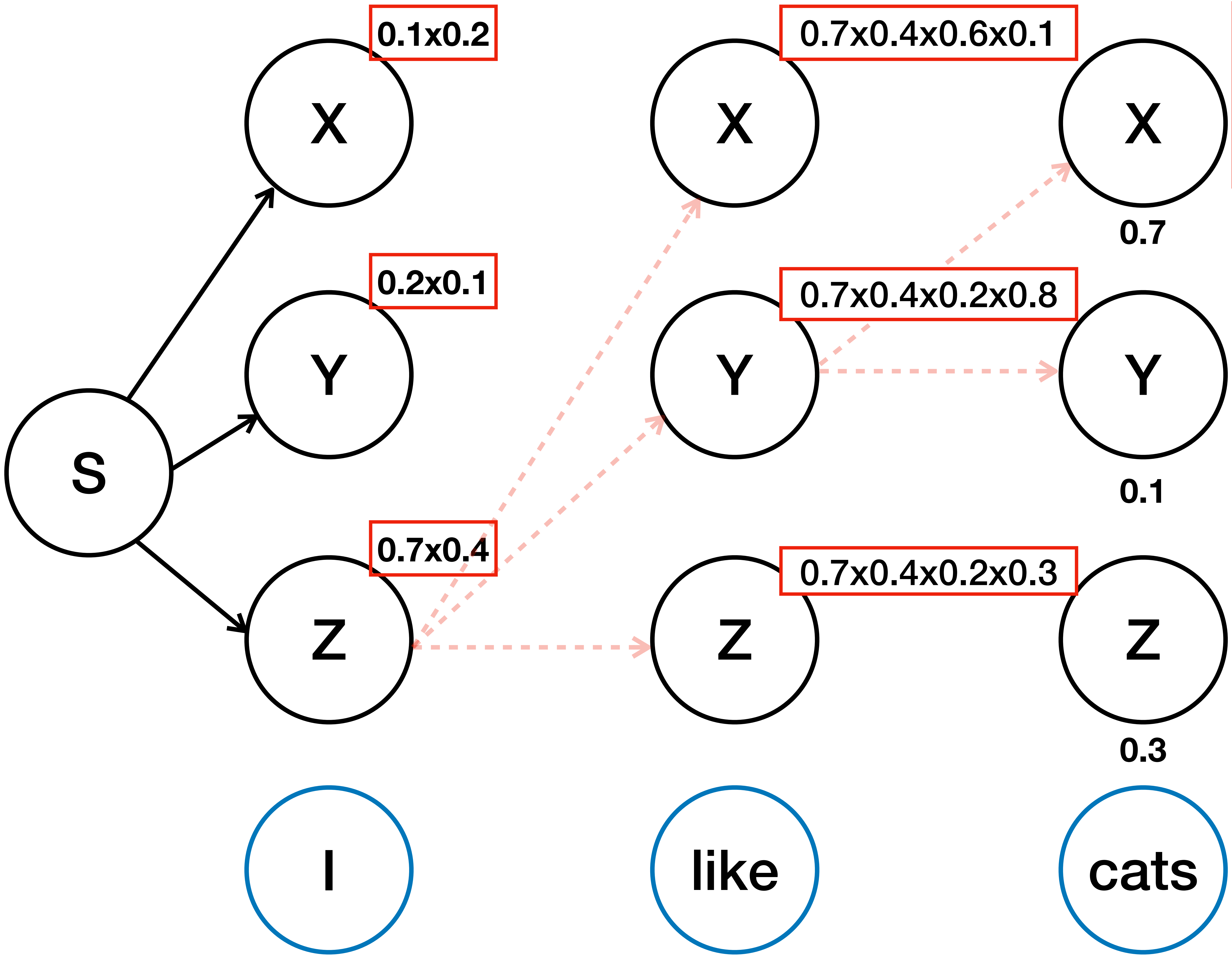
0.7x0.4x0.6x0.1x0.2x0.7 = 0.0023
0.7x0.4x0.2x0.8x0.4x0.7 = 0.0125
0.7x0.4x0.2x0.3x0.6x0.7 = 0.0070

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

		Y	Z
S	0.1	0.2	0.7
		0.5	0.3
		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



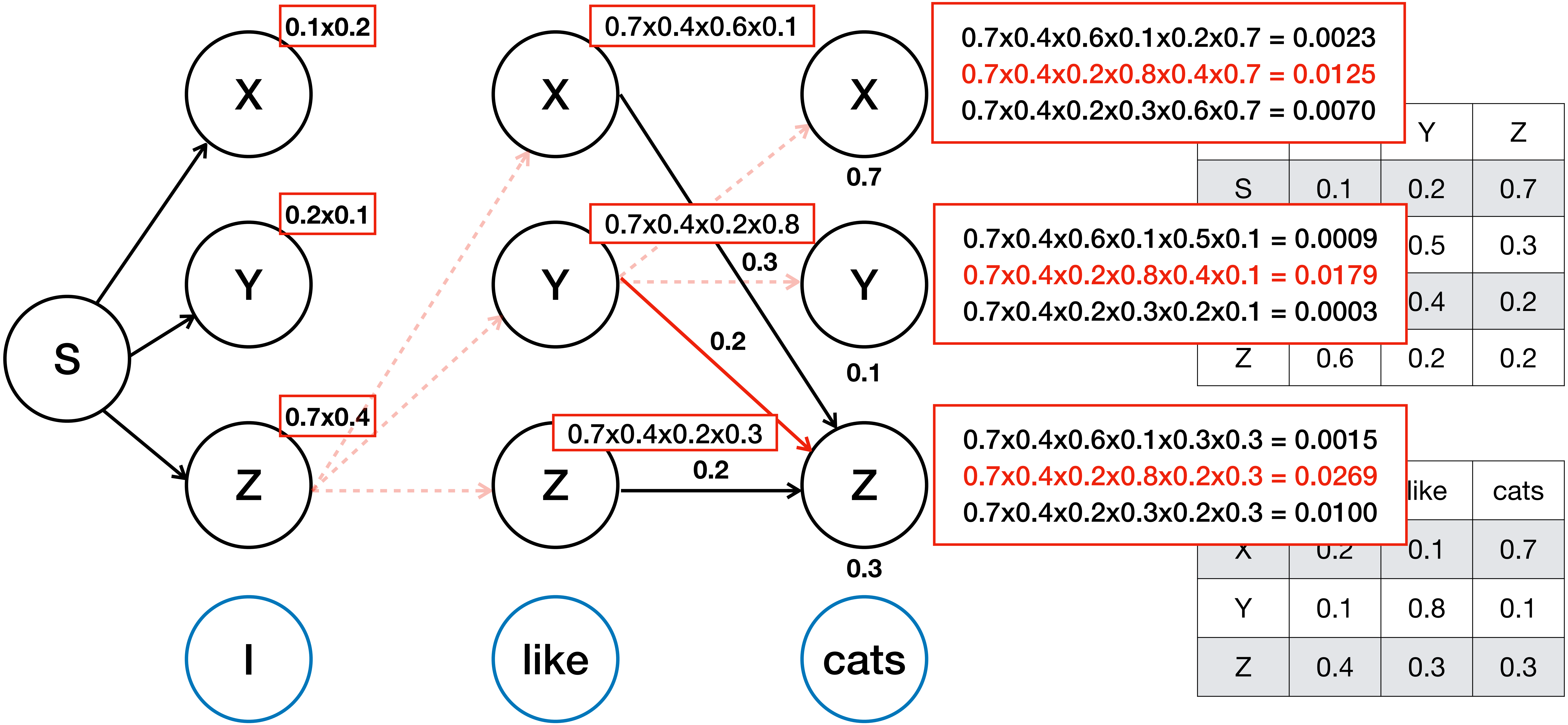
0.7x0.4x0.6x0.1x0.2x0.7 = 0.0023
0.7x0.4x0.2x0.8x0.4x0.7 = 0.0125
0.7x0.4x0.2x0.3x0.6x0.7 = 0.0070

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

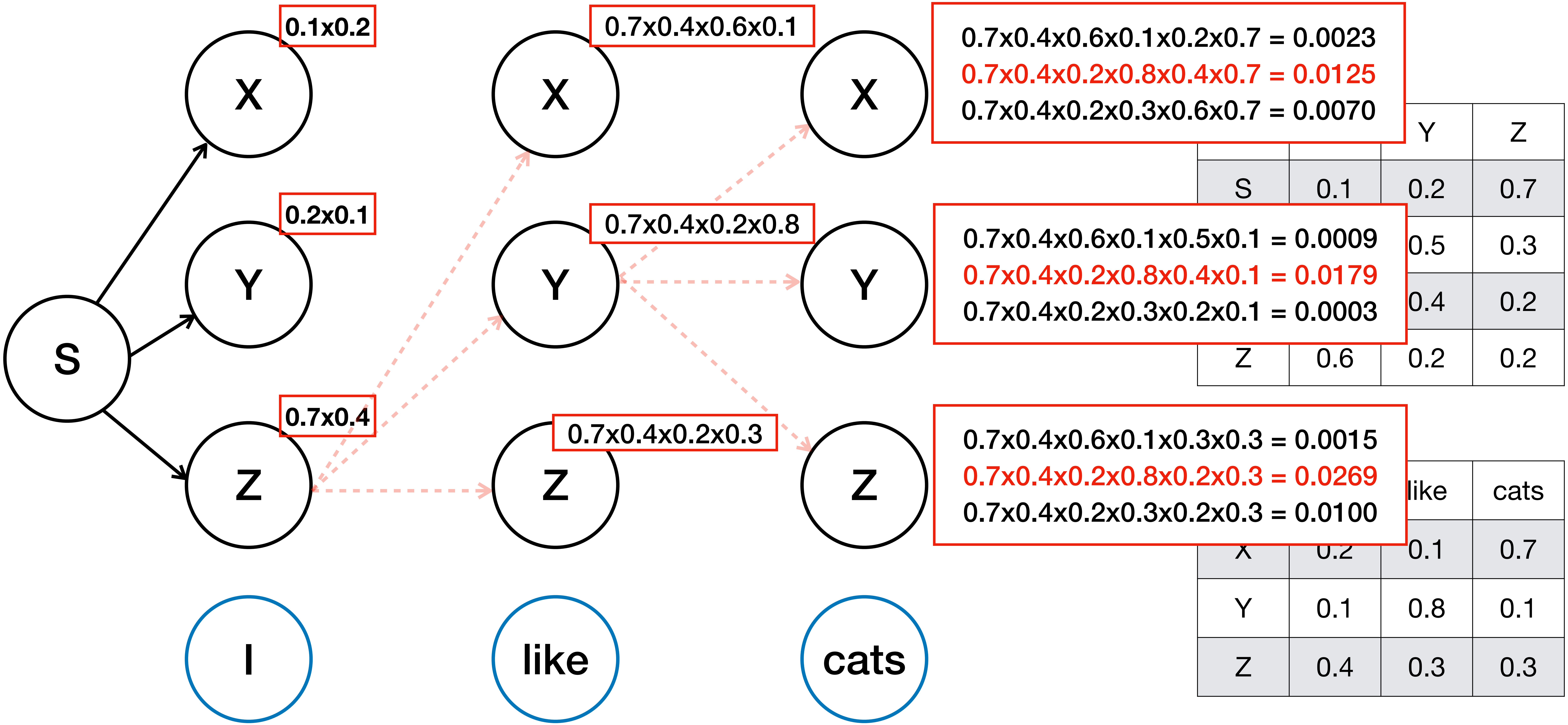
		Y	Z
S	0.1	0.2	0.7
		0.5	0.3
		0.4	0.2
Z	0.6	0.2	0.2

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

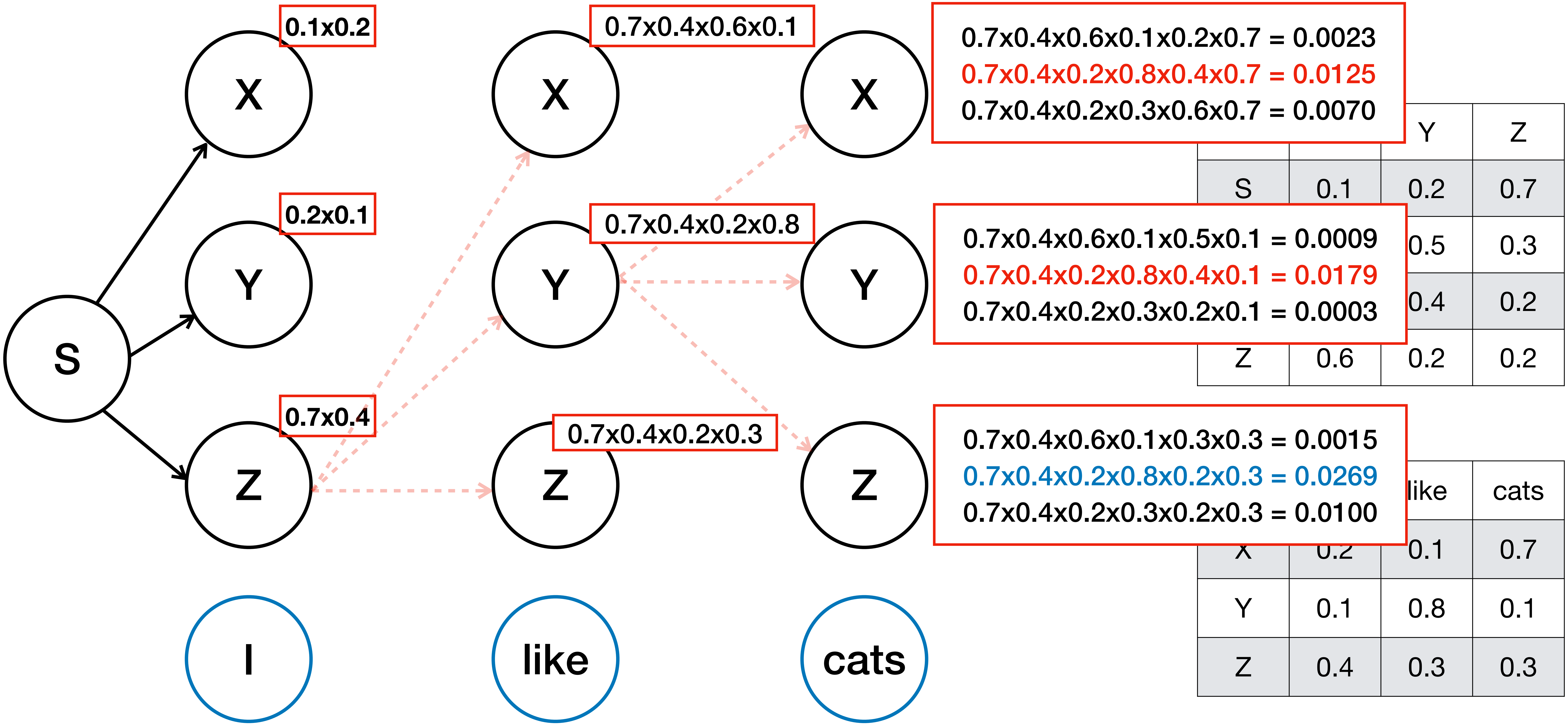
Viterbi Algorithm



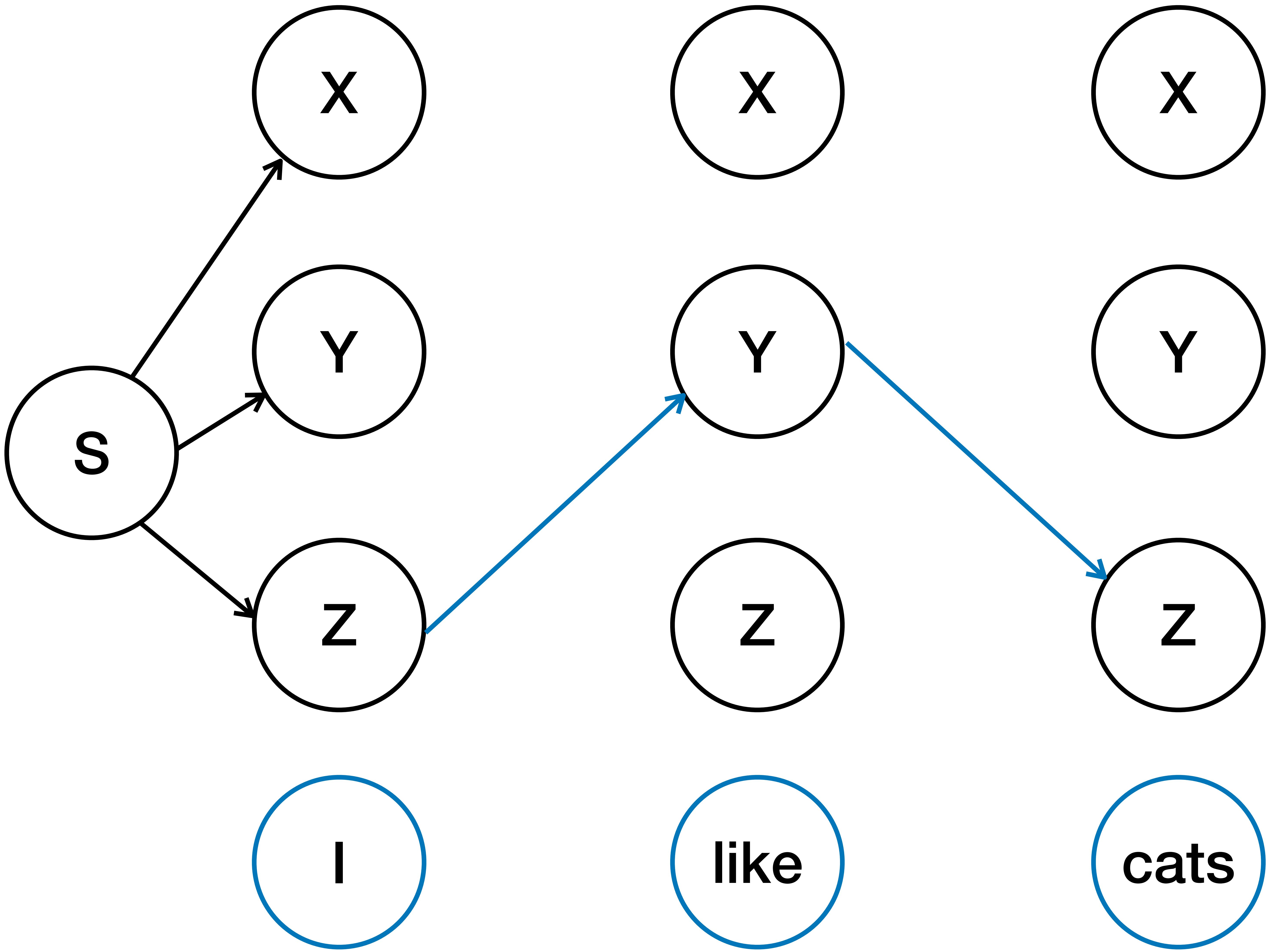
Viterbi Algorithm



Viterbi Algorithm



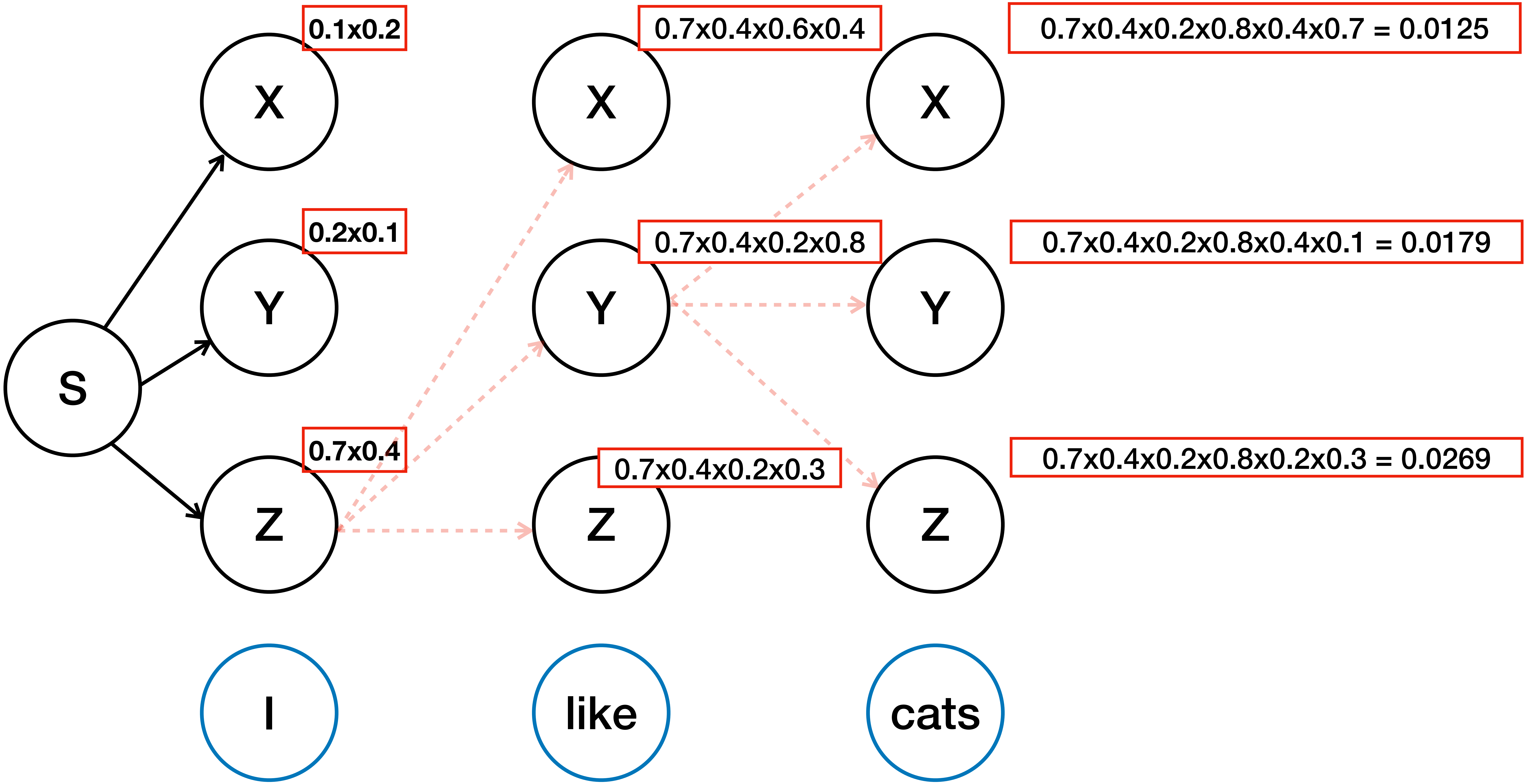
Viterbi Algorithm



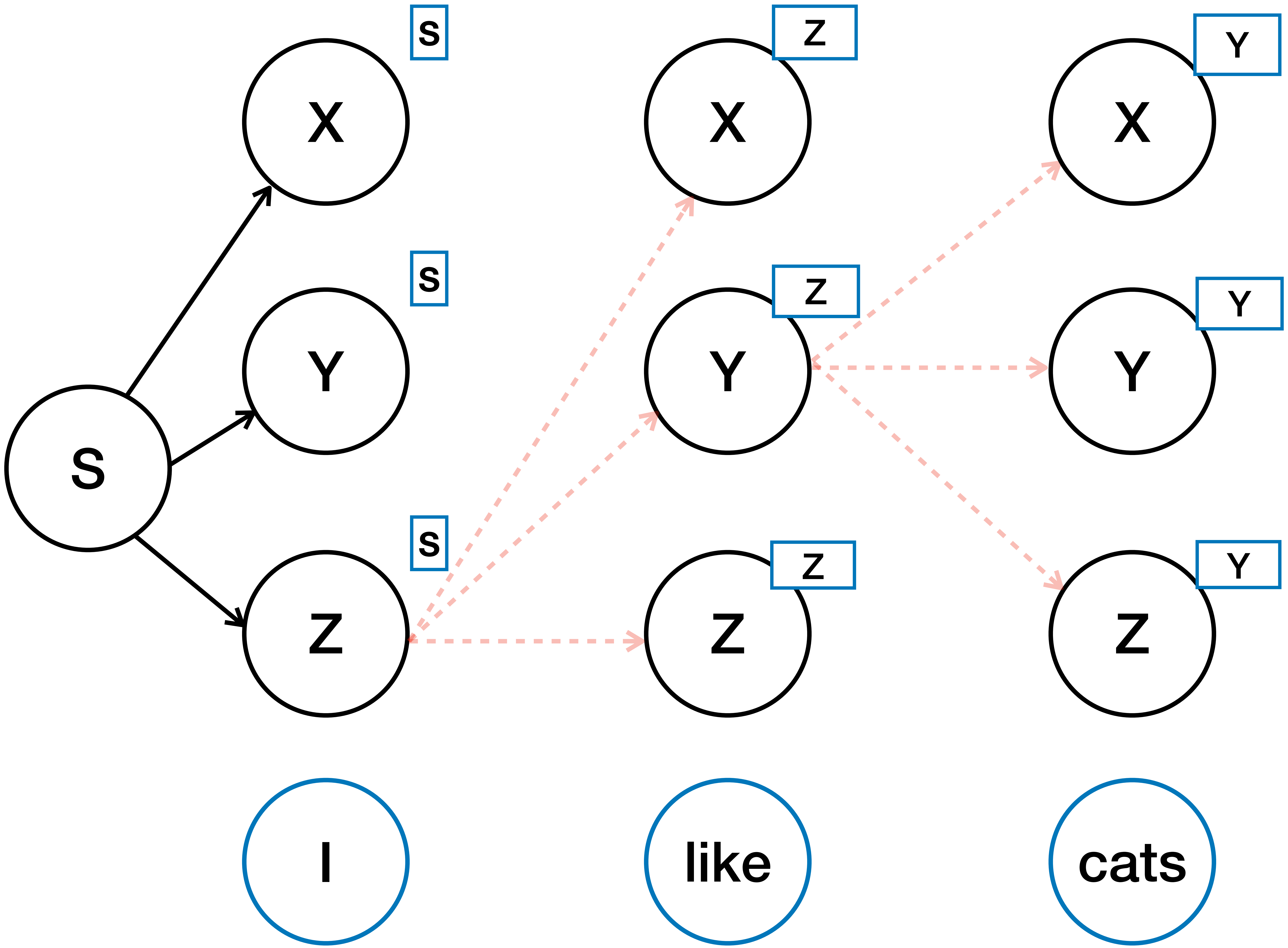
The final tags should be: <Z, Y, Z>

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

Viterbi Algorithm



Viterbi Algorithm

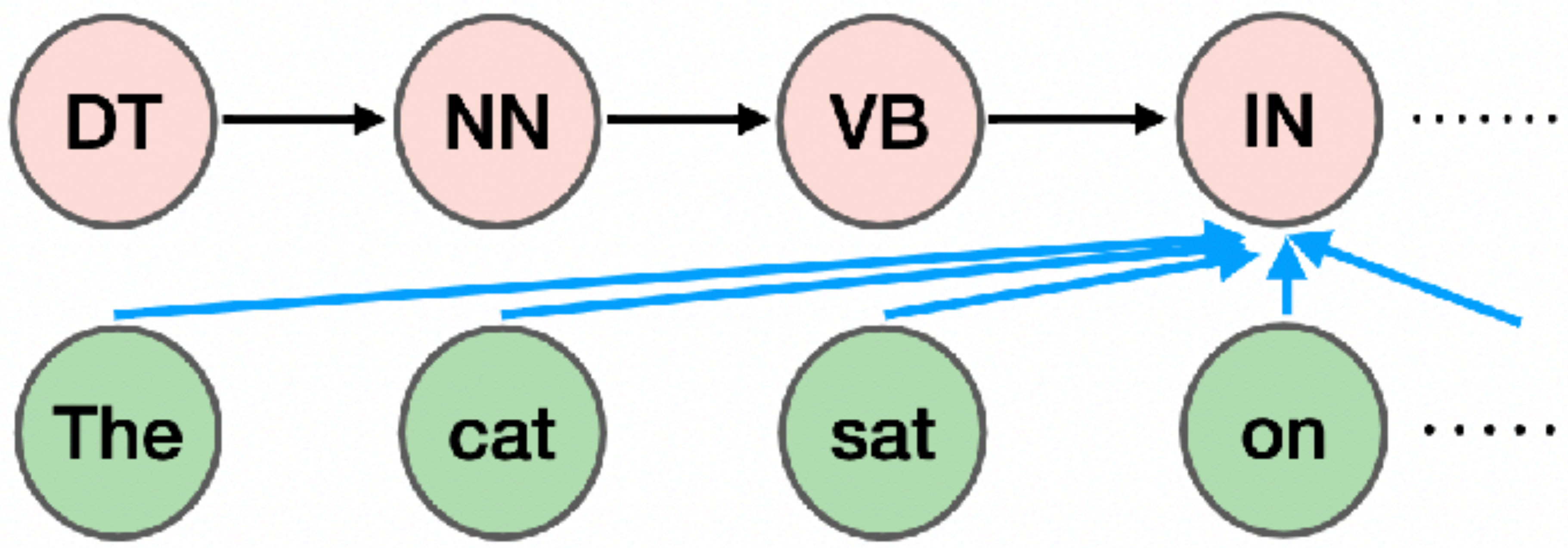


The backtracking matrix keeps track of the best node from the previous step.

Agenda

- HMM
- Viterbi Algorithm
- MEMM

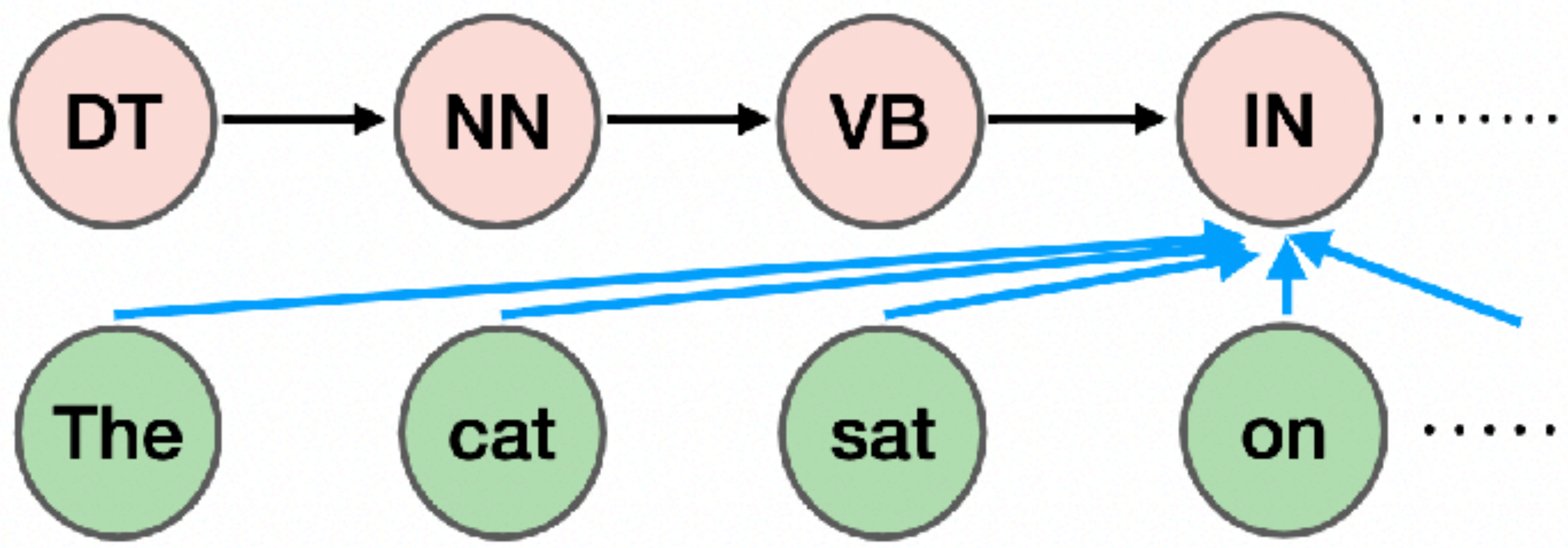
MEMM



MEMM

	Generative	Discriminative
Text classification	Naive Bayes: $P(c)P(d \mid c)$	Logistic Regression: $P(c \mid d)$
Sequence prediction	HMM: $P(s_1, \ldots, s_n)P(o_1, \ldots, o_n \mid s_1, \ldots, s_n)$	MEMM: $P(s_1, \ldots, s_n \mid o_1, \ldots, o_n)$

LR vs MEMM

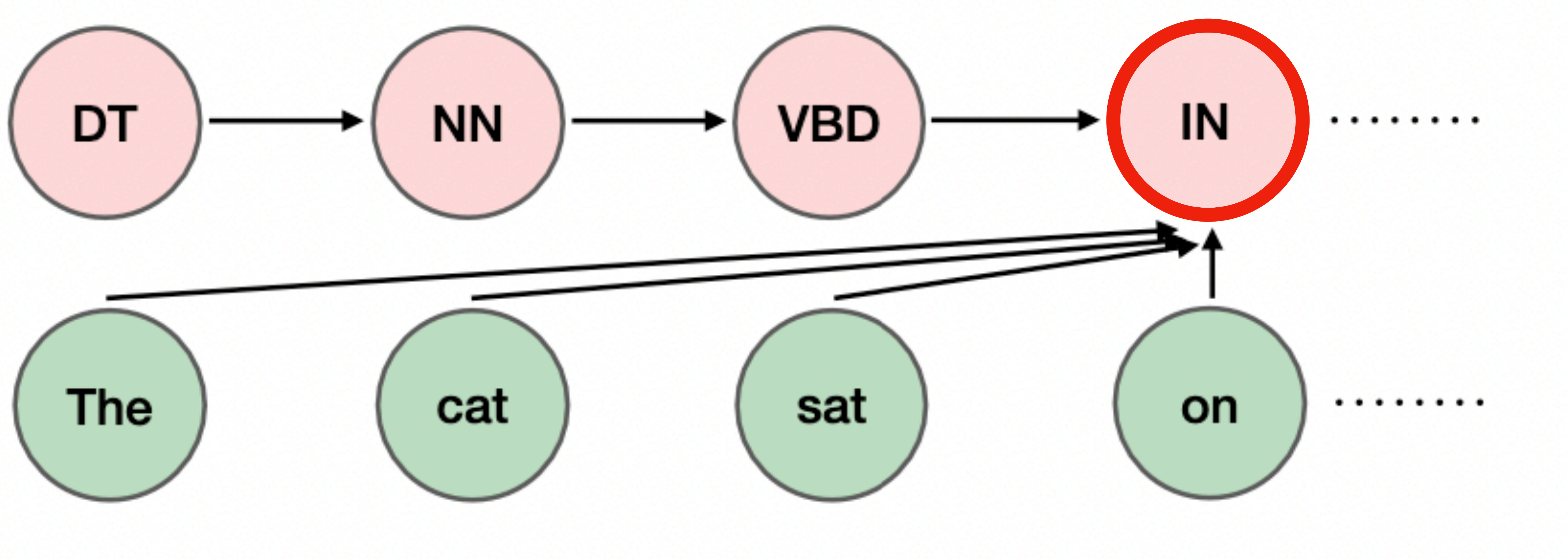


MEMM

$$P(c \mid d) = \frac{\exp(w_c \cdot x_d + b_c)}{\sum_{c' \in Y} \exp(w_{c'} \cdot x_d + b_{c'})}$$

$$P(s_i = s \mid s_{i-1}, O) = \frac{\exp(\mathbf{w} \cdot \mathbf{f}(s_i = s, s_{i-1}, O, i))}{\sum_{s'=1}^{K} \exp(\mathbf{w} \cdot \mathbf{f}(s_i = s', s_{i-1}, O, i))}$$

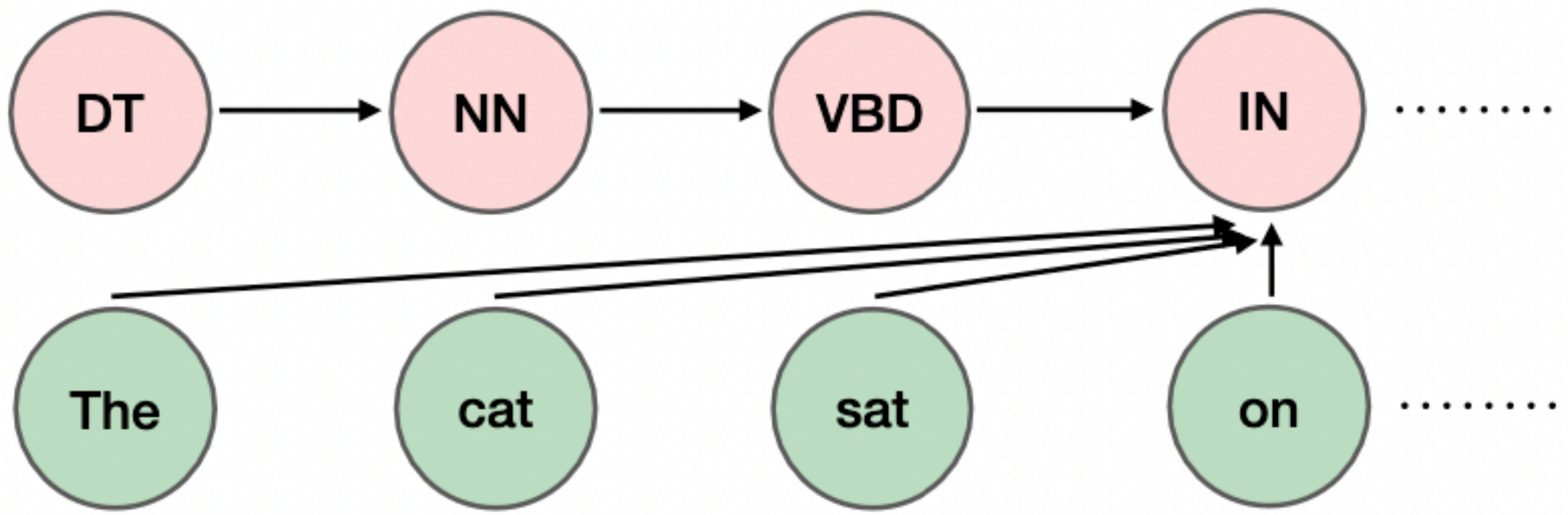
MEMM



- To predict the red node, the 4-gram MEMM conditions on the “prior tags” (DT, NN, VBD, IN) and the observations in the window (The, cat, sat, on)

- Prior tags and observations will be transformed into features (some sort of vector representation)

MEMM



We can design feature templates:

o_{i-2} = animal & s_{i-1} = VBD

s_{i-2} = NN & s_{i-1} = VBD

s_{i-3} = NNP

For predicting the IN tag position, the feature vector would be [1, 1, 0]. In practice, the final feature vector might be more complicated than this — the prior tags might be represented as one-hot vectors in addition to the template feature vectors.

Word Vectors

Word Vectors

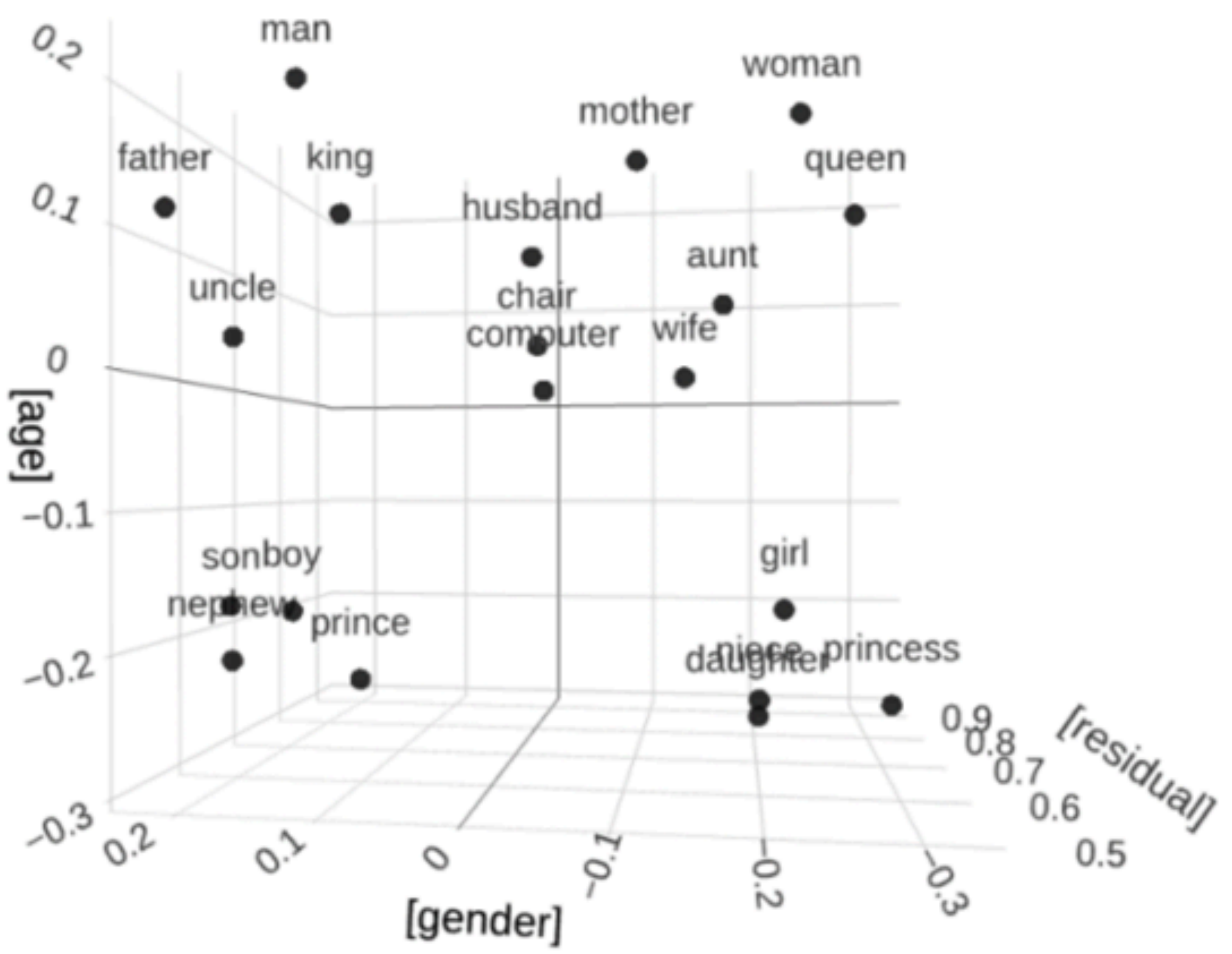
The big idea: model of meaning focusing on similarity

Each word = a vector

$$v_{\text{cat}} = \begin{pmatrix} -0.224 \\ 0.130 \\ -0.290 \\ 0.276 \end{pmatrix} \quad v_{\text{dog}} = \begin{pmatrix} -0.124 \\ 0.430 \\ -0.200 \\ 0.329 \end{pmatrix}$$

$$v_{\text{the}} = \begin{pmatrix} 0.234 \\ 0.266 \\ 0.239 \\ -0.199 \end{pmatrix} \quad v_{\text{language}} = \begin{pmatrix} 0.290 \\ -0.441 \\ 0.762 \\ 0.982 \end{pmatrix}$$

Similar words are "**nearby in the vector space**"



(Bandyopadhyay et al. 2022)

Word Vectors: Counts

First solution: Let's use **word-word co-occurrence counts** to represent the meaning of words!

Each word is represented by the corresponding **row vector**

context words:
4 words to the left +
4 words to the right

	aardvark	...	computer	data	result	pie	sugar	...
cherry	0	...	2	8	9	442	25	...
strawberry	0	...	0	0	1	60	19	...
digital	0	...	1670	1683	85	5	4	...
information	0	...	3325	3982	378	5	13	...

Most entries are 0s $\Longrightarrow$ sparse vectors

Word Vectors: PPMI

- But overly frequent words like "the", "it", or "they" also appear a lot near "cherry". They are not very informative about the context.

Solution: use a **weighted function** instead of raw counts!

Pointwise Mutual Information (PMI):

Do events *x* and *y* co-occur more or less than if they were independent?

$$\text{PMI}(x, y) = \log_2 \frac{P(x, y)}{P(x)P(y)} \qquad \text{PMI}(w = \text{cherry}, c = \text{pie}) = \log_2 \frac{P(w = \text{cherry}, c = \text{pie})}{P(w = \text{cherry})P(c = \text{pie})}$$

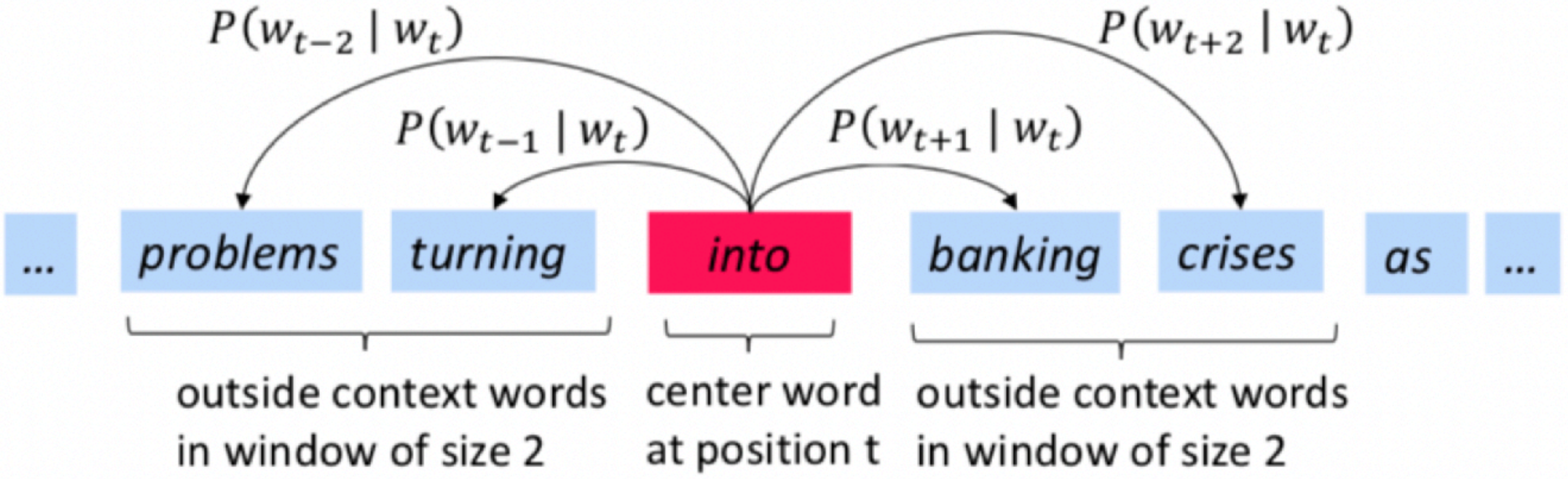
Word Vectors: Dense Vectors

Why dense vectors?

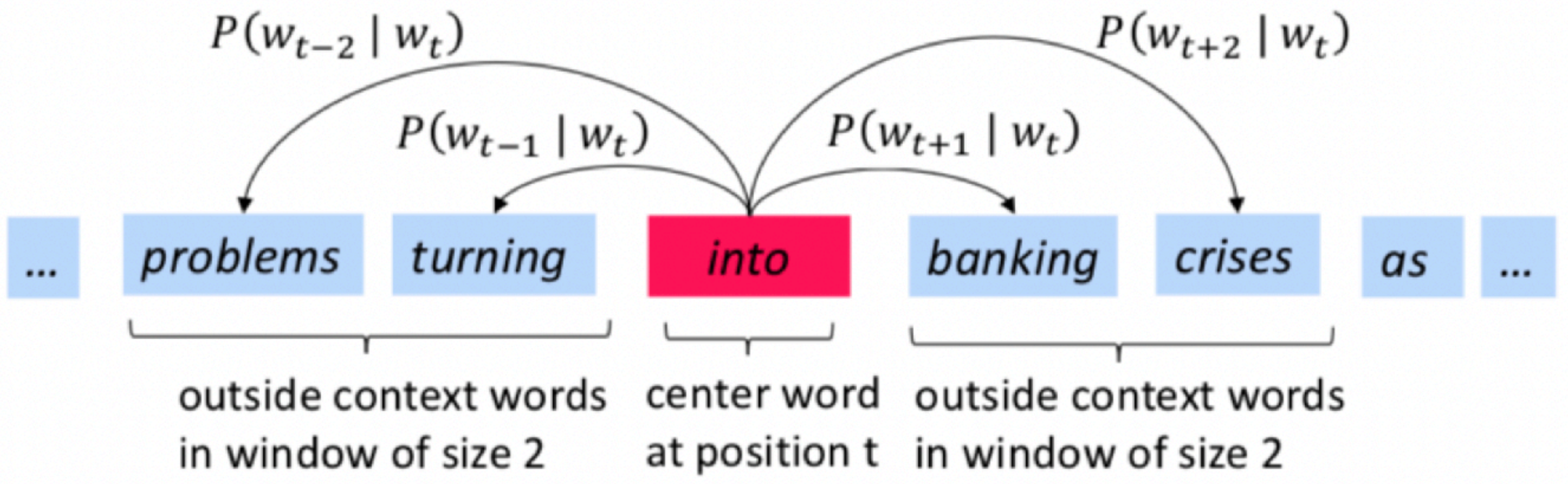
- Short vectors are easier to use as **features** in ML systems
- Dense vectors generalize better than explicit counts (points in real space vs points in integer space)
- Sparse vectors can't capture higher-order co-occurrence
  - w_1 co-occurs with “car”, w_2 co-occurs with “automobile”
  - They should be similar but they aren't because “car” and “automobile” are distinct dimensions
- In practice, they work better!

Word Vectors: Skip-Gram

- Assume that we have a large corpus $w_1, w_2, \ldots, w_T \in V$
- **Key idea:** Use each word to **predict** other words in its context
- Context: a fixed window of size 2*m* (m = 2 in the example)



Word Vectors: Skip-Gram

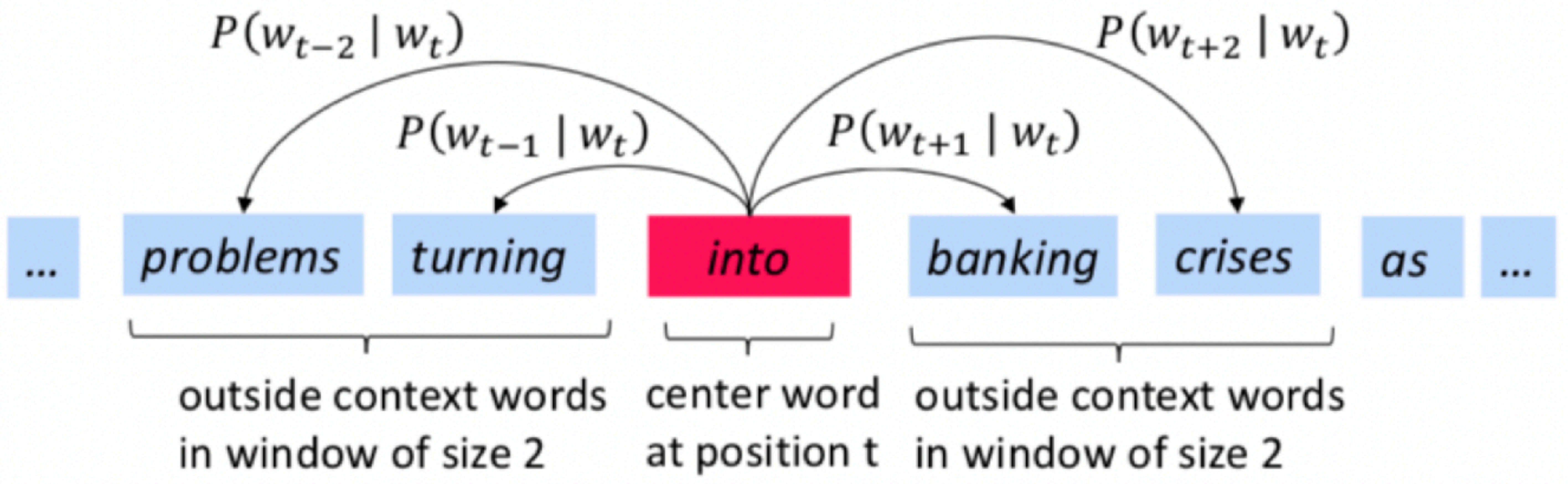


- For each position $t = 1,2,\ldots T$, predict context words within context size m, given center word w_t:

all the parameters to be optimized

$$\mathcal{L}(\theta) = \prod_{t=1}^{T} \prod_{-m \leq j \leq m, j \neq 0} P(w_{t+j} \mid w_t; \theta)$$

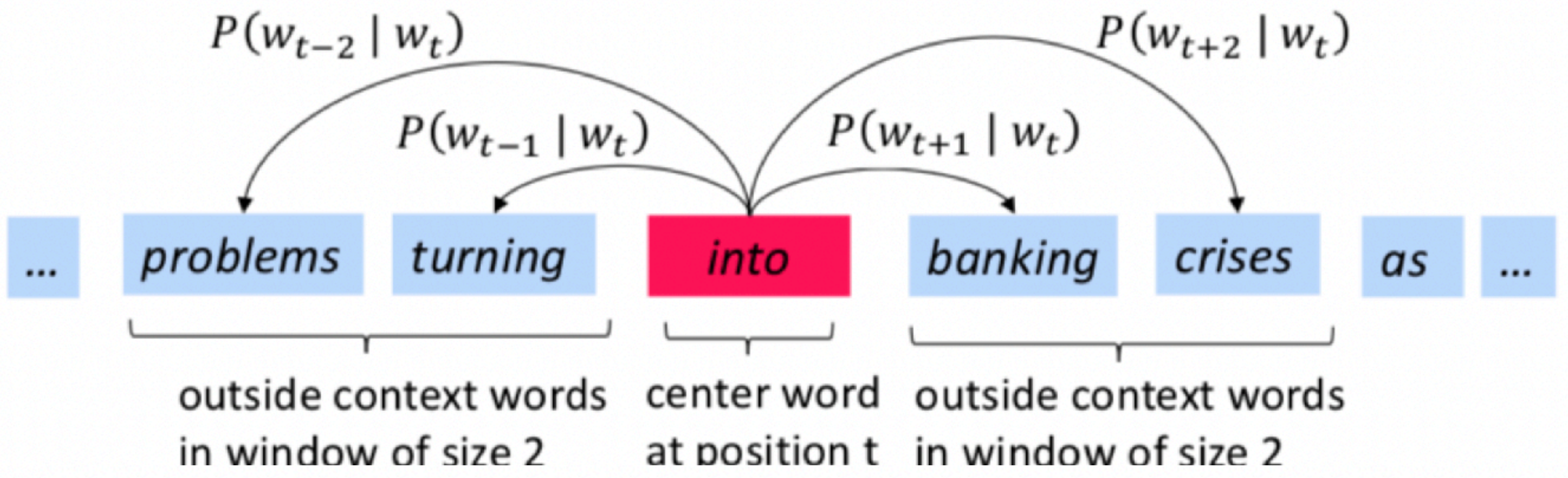
Word Vectors: Skip-Gram



- It is equivalent as minimizing the (average) negative log likelihood:

$$J(\theta) = -\frac{1}{T}\log\mathcal{L}(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{-m\le j\le m, j\neq 0}\log P(w_{t+j} \mid w_t; \theta)$$

Word Vectors: Skip-Gram



- Use inner product $\mathbf{u}_a \cdot \mathbf{v}_b$ to measure how likely word *a* appears with context word *b*

Softmax we have seen in multinomial logistic regression!

$$P(w_{t+j} \mid w_t) = \frac{\exp(\mathbf{u}_{w_t} \cdot \mathbf{v}_{w_{t+j}})}{\sum_{k \in V} \exp(\mathbf{u}_{w_t} \cdot \mathbf{v}_k)}$$

Recall that $P(\cdot \mid a)$ is a probability distribution defined over V…

Word Vectors: Skip-Gram

Problem: every time you get one pair of (t, c), you need to update $\mathbf{v}_k$ with all the words in the vocabulary! This is very expensive computationally.

- Use inner product $\mathbf{u}_a \cdot \mathbf{v}_b$ to measure how likely word *a* appears with context word *b*

Softmax we have seen in multinomial logistic regression!

$$P(w_{t+j} \mid w_t) = \frac{\exp(\mathbf{u}_{w_t} \cdot \mathbf{v}_{w_{t+j}})}{\sum_{k \in V} \exp(\mathbf{u}_{w_t} \cdot \mathbf{v}_k)}$$

Recall that $P(\cdot \mid a)$ is a probability distribution defined over V…

Word Vectors: Skip-Gram

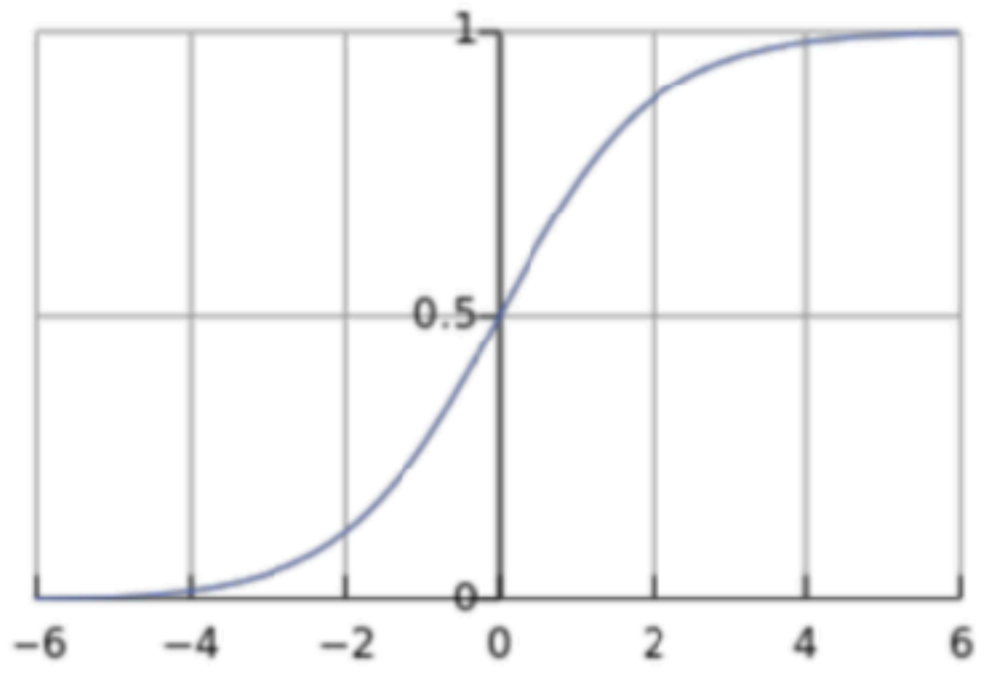
Problem: every time you get one pair of (t, c), you need to update $\mathbf{v}_k$ with all the words in the vocabulary! This is very expensive computationally.

Negative sampling: instead of considering all the words in V, let's randomly sample K (5-20) negative examples.

$$\sigma(x) = \frac{1}{1 + \exp(-x)}$$

softmax: $$y = -\log\left(\frac{\exp(\mathbf{u}_t \cdot \mathbf{v}_c)}{\sum_{k \in V} \exp(\mathbf{u}_t \cdot \mathbf{v}_k)}\right)$$

Negative sampling: $$y = -\log(\sigma(\mathbf{u}_t \cdot \mathbf{v}_c)) - \sum_{i=1}^{K} \mathbb{E}_{j \sim P(w)} \log(\sigma(-\mathbf{u}_t \cdot \mathbf{v}_j))$$

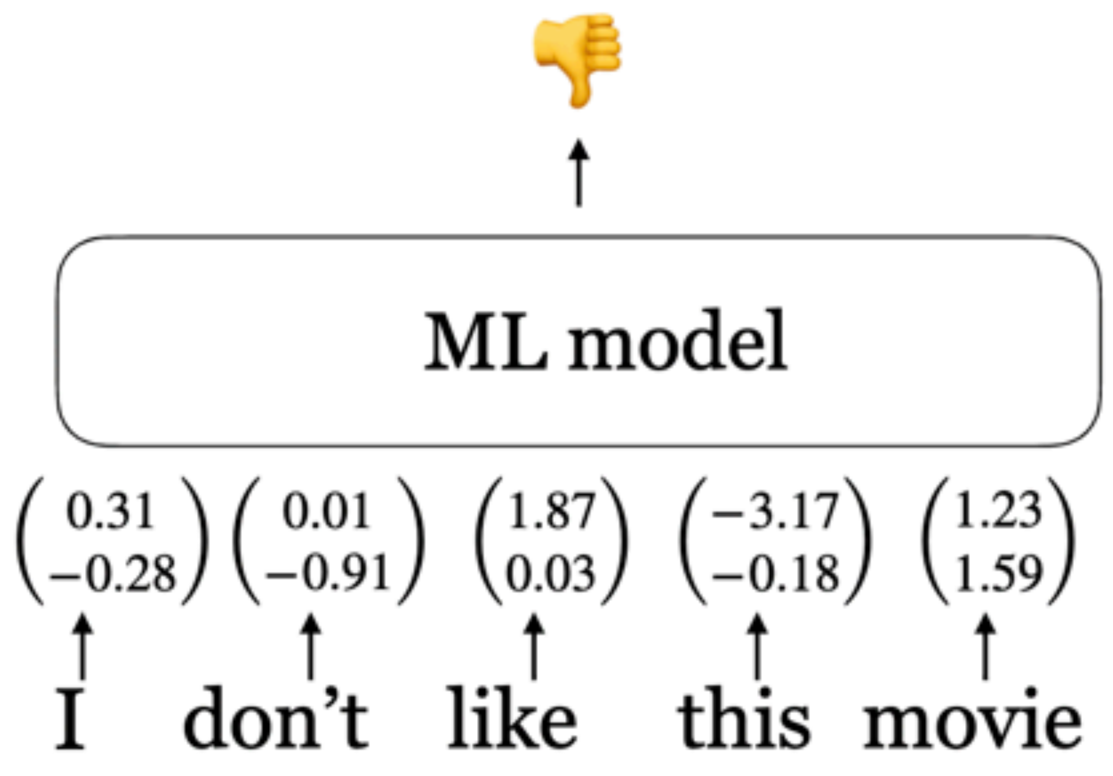


Word Vectors: Evaluation

Extrinsic vs intrinsic evaluation

Extrinsic evaluation

- Let’s plug these word embeddings into a real NLP system and see whether this improves performance
- Could take a long time but still the most important evaluation metric



Intrinsic evaluation

- Evaluate on a specific/intermediate subtask
- Fast to compute
- Not clear if it really helps downstream tasks