

# Token Classification

---

PoS and NER

HMM, CRF, and search

# Token Classification

- A broad term for classifying tokens: PoS, NER

Discourse

Semantics

CommunicationEvent(e) SpeakerContext(s)  
Agent(e, Alice) TemporalBefore(e, s)  
Recipient(e, Bob)

Syntax: Constituents

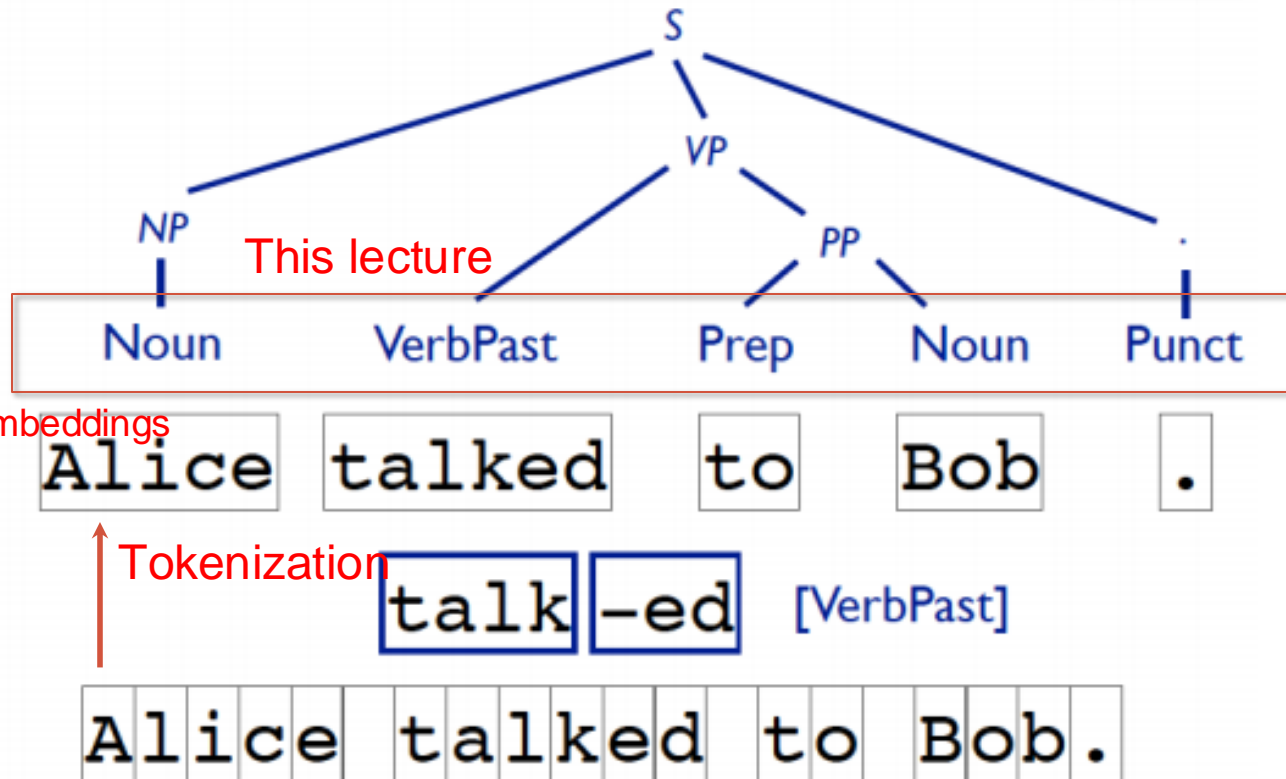
Syntax: Part of Speech

Words

Morphology

Characters

Ref: Prof. Brendan O'Connor, CS 885 Intro to NLP, @UMass



# Part-Of-Speech tagging

- Categorize words into similar **grammatical** properties (syntax)
  - Examples: Nouns, Verbs, Adjectives
- Actual applications often use more granular PoS labels
- PoS tags are often
  - Language specific
  - Application/corpus specific

Number	Tag	Description
1.	CC	Coordinating conjunction
2.	CD	Cardinal number
3.	DT	Determiner
4.	EX	Existential <i>there</i>
5.	FW	Foreign word
6.	IN	Preposition or subordinating conjunction
7.	JJ	Adjective
8.	JJR	Adjective, comparative
9.	JJS	Adjective, superlative
10.	LS	List item marker
11.	MD	Modal
12.	NN	Noun, singular or mass
13.	NNS	Noun, plural
14.	NNP	Proper noun, singular
15.	NNPS	Proper noun, plural
16.	PDT	Predeterminer
17.	POS	Possessive ending
18.	PRP	Personal pronoun
19.	PRP\$	Possessive pronoun
20.	RB	Adverb
21.	RBR	Adverb, comparative
22.	RBS	Adverb, superlative
23.	RP	Particle
24.	SYM	Symbol
25.	TO	<i>to</i>

# Part-Of-Speech tagging

- Input
- They refuse to permit us to obtain the refuse permit.
- Output
- They/*PRP* refuse/*VBP* to/*To* permit/*VB* us/*PRP* to/*TO*  
obtain/*VB* the/*DT* refuse/*NN* permit/*NN*

# PoS usage

- Word disambiguation
  - Different word vectors for different PoS of the same words
- Helps other NLP tasks
- PoS provides additional information that helps other tasks
  - **Tokenization**
  - **Name-Entity Recognition**
    - Identify group of words that refer to the same entity
    - ลุง พล ร้อง เพลง คุณก็ เสียหาย
    - [ลุงพล]/person ร้อง เพลง [คุณก็เสียหาย]/title
  - **Parsing** Ex parsing name and address from a sentence
  - **Search** Ex disambiguation in keywords
  - **Text-to-speech** Ex Nice, France vs nice french toast.

# Thai PoS standards

- <https://pythainlp.org/dev-docs/api/tag.html>
- LST20 and Orchid are most famous Thai datasets
- There is also Universal POS tags (not used much for Thai)

# Orchid PoS corpus

- Building A Thai Part-Of-Speech Tagged Corpus (ORCHID) (1999)  
<http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.34.3496>
  - 47 tags

No.	POS	Description	Example
1	NPRP	Proper noun	วินโดวส์ 95, โคโรนา, ไค้ก, พระอาทิตย์
2	NCNM	Cardinal number	หนึ่ง, สอง, สาม, 1, 2, 3
3	NONM	Ordinal number	ที่หนึ่ง, ที่สอง, ที่สาม, ที่1, ที่2, ที่3
4	NLBL	Label noun	1, 2, 3, 4, ก, ข, a, b
5	NCMN	Common noun	หนังสือ, อาหาร, อาคาร, คน
6	NTTL	Title noun	ดร., พลเอก
7	PPRS	Personal pronoun	คุณ, เขา, ฉัน
8	PDMN	Demonstrative pronoun	นี้, นั้น, ที่นั่น, ที่นี่
9	PNTR	Interrogative pronoun	ใคร, อะไร, อย่างไร
10	PREL	Relative pronoun	ที่, ซึ่ง, อัน, ผู้
11	VACT	Active verb	ทำงาน, ร้องเพลง, กิน
12	VSTA	Stative verb	เห็น, รู้, คือ
13	VATT	Attributive verb	อ้วน, ดี, สวย
14	XVBM	Pre-verb auxiliary, before negator “ไม่”	เกิด, เกือบ, กำลัง
15	XVAM	Pre-verb auxiliary, after negator “ไม่”	ค่อย, น่า, ได้
16	XVMM	Pre-verb, before or after negator “ไม่”	ควร, เคย, ต้อง
17	XVBB	Pre-verb auxiliary, in imperative mood	กรุณา, จง, เชิญ, อย่า, ห้าม
18	XVAE	Post-verb auxiliary	ไป, มา, ขึ้น



# NER

- Name-Entity Recognition wants to extract all name entities in a sentence and classify into types
- Can tag groups of words into the same category
  - ลุง พล ร้อง เพลง คุณก็ เสียหาย
  - [ลุงพล]/person ร้อง เพลง [คุณก็เสียหายน]/title

Biomedical publication mining

**IL-2 gene** expression and **NF-kappa B** activation through **CD28** requires reactive oxygen production by **5-lipoxygenase**.

# NER in applications

- Classifying/tagging content
  - Information retrieval
  - Trends analysis

When Michael Jordan was at the peak of his powers as an NBA superstar, his Chicago Bulls teams were mowing down the competition, winning six National Basketball Association titles and setting a record for wins in a season that was broken by the Golden State Warriors two seasons ago.

Extract

## KEYWORDS

Place: Chicago

Name: Michael Jordan

Group: National Basketball Association

# Labeling standard (IOB format)

- Consider the following sentence and NE tags

ลง พล ป่า เบิร์ด ร้อง เพลง ชาติ  
Name Name Name Name Other Song Song Song  
ไทย

- The two names are merged into a single entity. To separate the two entities, we sometimes add B

NameB b NameI i NameB b NameI i ns Other to SongB ag SongI SongI  
ลง พล ป่า เบิร์ด ร้อง เพลง ชาติ  
ไทย

# Thai Nested NER

- Thai NER has multiple levels/layers and can be nested



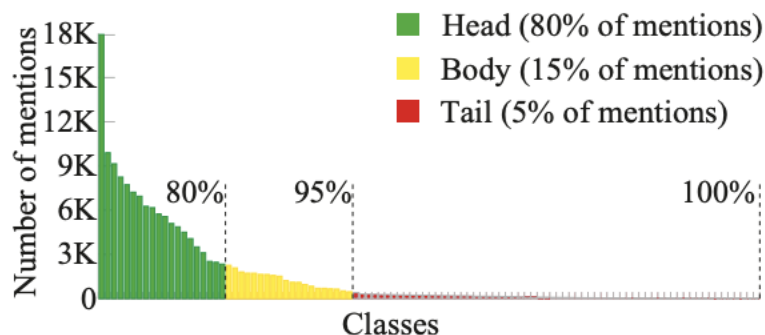
# Thai Nested NER

	Models	Head			Body			Tail			All		
		P	R	F1	P	R	F1	P	R	F1	P	R	F1
Baseline	CRF model	86.06	66.46	75.00	78.30	44.88	57.06	74.07	29.46	42.15	84.60	60.59	70.61
	WangchanBERTa-sp	<b>90.70</b>	77.66	83.67	<b>81.55</b>	55.90	<b>66.33</b>	78.02	26.09	39.10	<b>89.04</b>	70.89	78.94
	WangchanBERTa-sh	90.51	79.24	84.50	81.37	55.09	65.70	78.33	30.79	44.20	88.87	72.25	79.70
	XLM-R-sp	90.27	77.39	83.34	80.45	52.71	63.69	75.42	33.04	45.95	88.42	70.56	78.48
	XLM-R-sh	89.45	79.72	84.31	77.29	<b>58.06</b>	66.31	71.80	<b>39.73</b>	<b>51.16</b>	86.93	<b>73.66</b>	<b>79.75</b>
SOTA	Second-best-learning	87.57	<b>81.78</b>	<b>84.58</b>	80.12	54.85	65.12	<b>79.05</b>	19.41	31.16	86.41	73.49	79.43
	Pyramid	87.59	80.33	83.81	76.07	53.72	62.97	74.20	23.92	36.18	85.65	72.45	78.50
	Locate and label	77.60	80.38	78.97	64.42	56.21	60.04	77.43	18.86	30.33	75.57	72.61	74.06

1 model per layer  
1 model all layer

Table 4: Experimental results nested-NER models divided into head, body, tail, and overall in our dataset

Training set statistics



<https://aclanthology.org/2022.findings-acl.116/>  
<https://github.com/vistec-AI/Thai-NNER>

Figure 3: The distribution of classes sorted by frequency shows that rarer classes consist of more than 20% of all instances.

# Overview

- What is Token Classification?
- Traditional methods
  - Sequence methods
    - HMM
    - CRF
      - Viterbi and beam search
- Neural network methods

# Sequence methods

- They refuse to permit us to obtain the refuse permit.
- They/*PRP* refuse/*VBP* to/*To* permit/*VB* us/*PRP* to/*TO* obtain/*VB* the/*DT* refuse/*NN* permit/*NN*
- Determining the PoS tag depends on the decision of the words around it
  - A sequence problem

# Problem setup

- Sequence of words

- $W := \{w_1, w_2, w_3, \dots, w_n\}$

- Sequence of tags

- $T := \{t_1, t_2, t_3, \dots, t_n\}$

$P(a,b)$  joint distribution

$P(a|b)$  conditional distribution

$P(a)$  marginal distribution

$$P(a) = \sum_b P(a,b)$$

- Given  $W$  predict  $T$

- $\operatorname{argmax}_T P(T|W)$  Discriminative Model

- Or

- $\operatorname{argmax}_T \frac{P(T,W)}{P(W)} = \operatorname{argmax}_T P(T,W)$  Generative Model

$P(W)$  is constant does not affect the argmax



# Modeling $P(T, W)$

- $P(T, W) = P(w_1, w_2, w_3, \dots, w_n, t_1, t_2, t_3, \dots, t_n)$ 
  - Is there a problem with this?

# Modeling $P(T,W)$

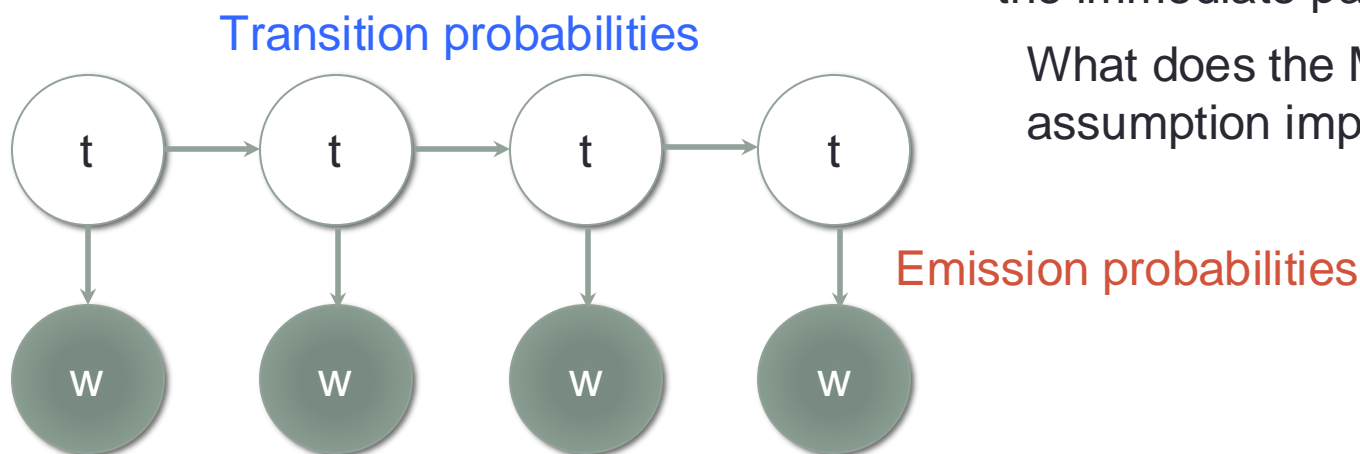
- $P(T,W) = P(w_1, w_2, w_3, \dots, w_n, t_1, t_2, t_3, \dots, t_n)$ 
  - Is there a problem with this? **Curse of dimensionality**
- Language modeling
  - $P(w_t)$  requires  $N$  table values
  - $P(w_t|w_{t-1})$  requires  $N^2$  table values
  - $P(w_t|w_{t-1}, w_{t-2})$  requires  $N^3$  table values
  - Many values have 0 counts (needs many tricks)
- We can use **Markov Assumptions**
  - Or more generally, we use **independence assumptions (conditional independence)** to simplify the distribution to model

# Hidden Markov Model

## Markov assumption

Current value only depends on the immediate pass

What does the Markov assumption imply?



T are called the hidden states. Usually comes from a finite set of possibilities

Ex:  $t \in \{\text{Noun, Adv, Adj, Verb}\}$

$$P(T, W) = \underbrace{P(t_1)}_{\text{Initial state prob}} \underbrace{P(t_2|t_1)P(t_3|t_2)P(t_4|t_3)}_{\text{Transition probabilities}} \underbrace{P(w_1|t_1)P(w_2|t_2)P(w_3|t_3)P(w_4|t_4)}_{\text{Emission probabilities}}$$

Initial state prob   Transition probabilities   Emission probabilities

# Hidden Markov Model

N = Noun

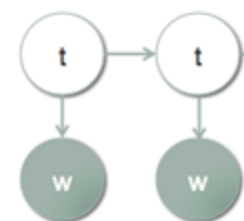
NN = Not Noun

$i, j$  – state (tag) index

$k$  – emission (word) index

- Defining HMM requires
  - Starting state probability  $p_0 = [0.7 \ 0.3]$
  - Transition probability,  $A_{ij}$

$A_{ij}$	To N	To NN
From N	0.6	0.4
From NN	0.5	0.5



- Emission probability,  $B_{ik}$
- If emits discrete values
  - Discrete HMM
- If emits continuous values
  - Continuous HMM

$B_{ik}$	I	eat	Chinese
State N	0.8	0.01	0.19
State NN	0.1	0.45	0.45

Question: What's the probability of  $P([I \text{ eat chinese}], [N \text{ NN N}])$  ?

$$= p_0(N) * A_{12} * A_{21} * B_{11} * B_{22} * B_{13}$$

$$= 0.7 * 0.4 * 0.5 * 0.8 * 0.45 * 0.19$$

# Hidden Markov Model

- How to estimate A and B?

$A_{ij}$	To N	To NN
From N	0.6	0.4
From NN	0.5	0.5

- Counts!

$$P(A_{11}) = \frac{\text{Count}(\text{from N to N})}{\text{Count}(\text{N})}$$

$B_{ik}$	I	eat	Chinese
State N	0.8	0.01	0.19
State NN	0.1	0.45	0.45

# Hidden Markov Model

- How to estimate A and B?

$A_{ij}$	To N	To NN
From N	0.6	0.4
From NN	0.5	0.5

- Counts!

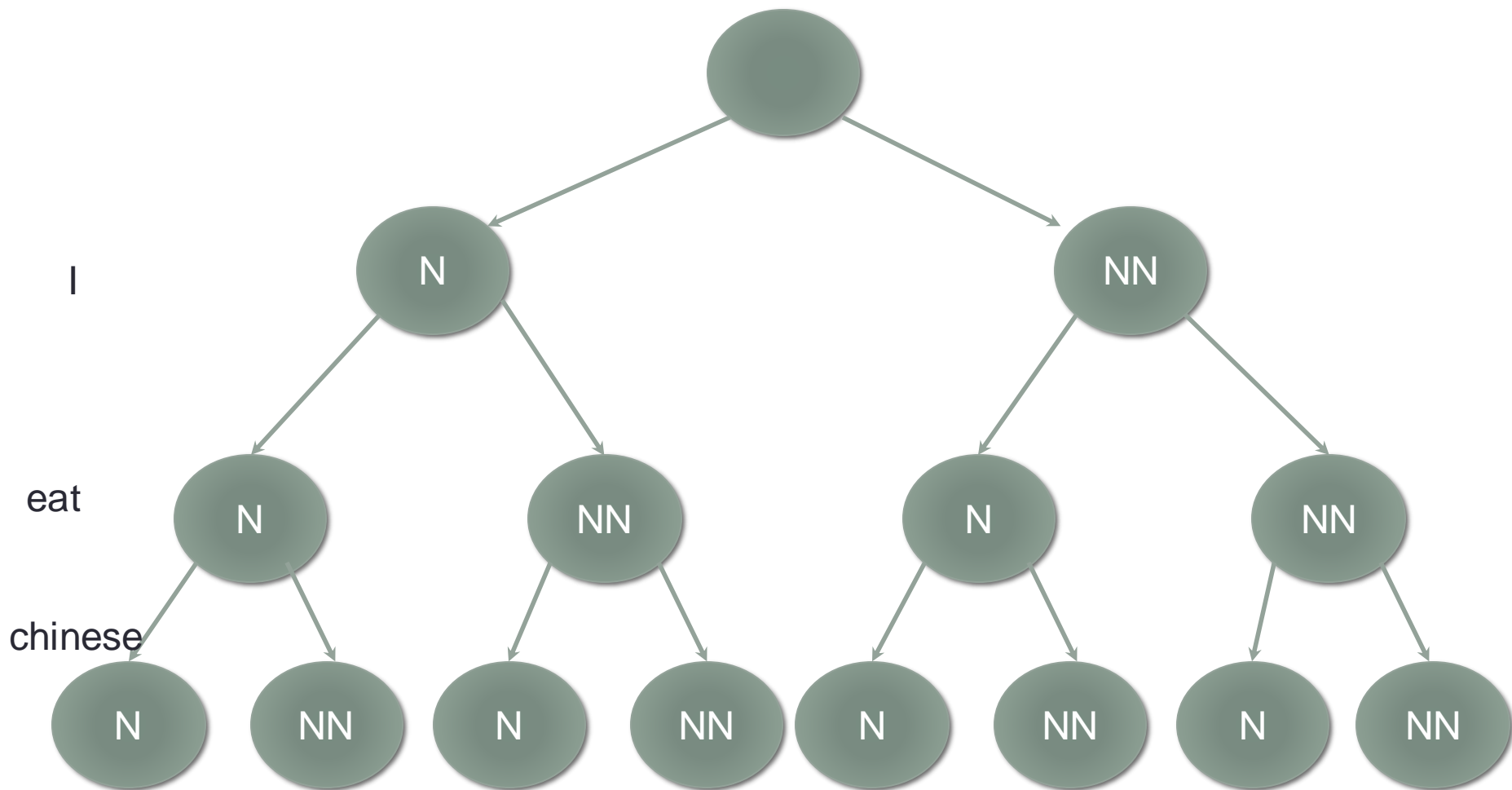
$$P(B_{11}) = \frac{\text{Count}(\text{"I"}, N)}{\text{Count}(N)}$$

$B_{ik}$	I	eat	Chinese
State N	0.8	0.01	0.19
State NN	0.1	0.45	0.45

# Decoding

- Recall we want to find the sequence of tags that maximizes the joint probability
  - $\operatorname{argmax}_T P(T,W)$
- How to do this?
  - Brute force
    - Find all possible sequence of T, calculate  $P(T,W)$  and compare
    - Length N words, B possible tags
    - Big O = ?
  - Depth first search?
  - Breadth first search?

# Breadth first/depth first search



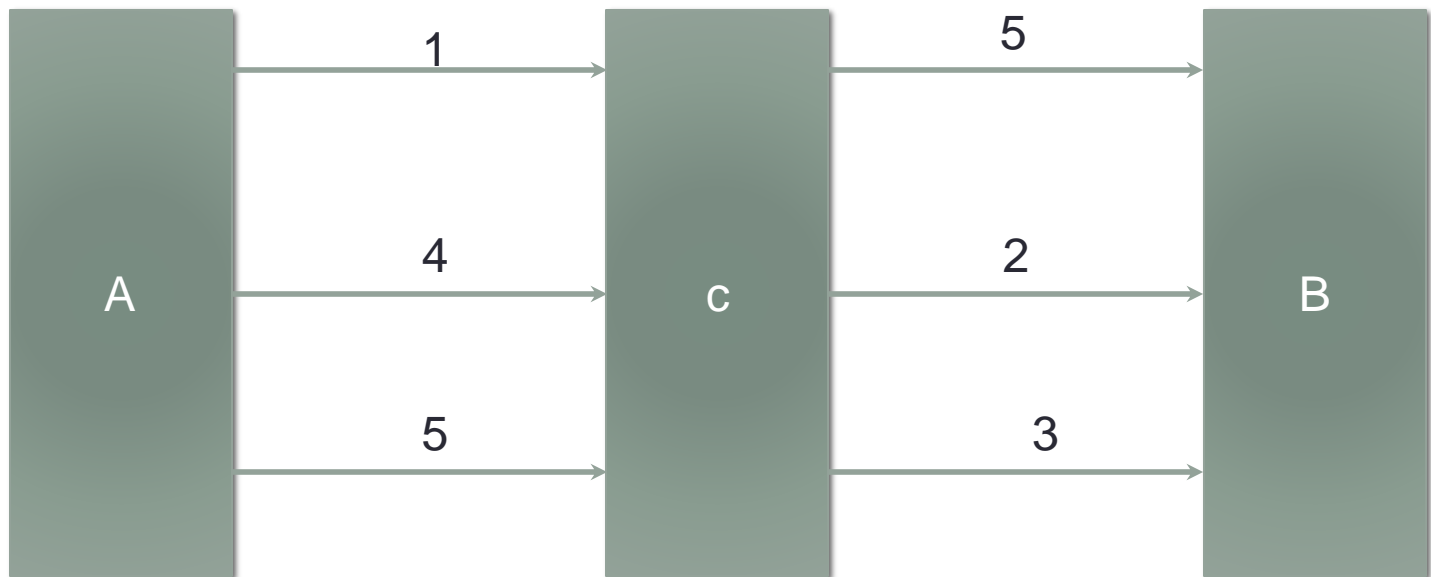


# The Viterbi Algorithm (Dynamic programming)

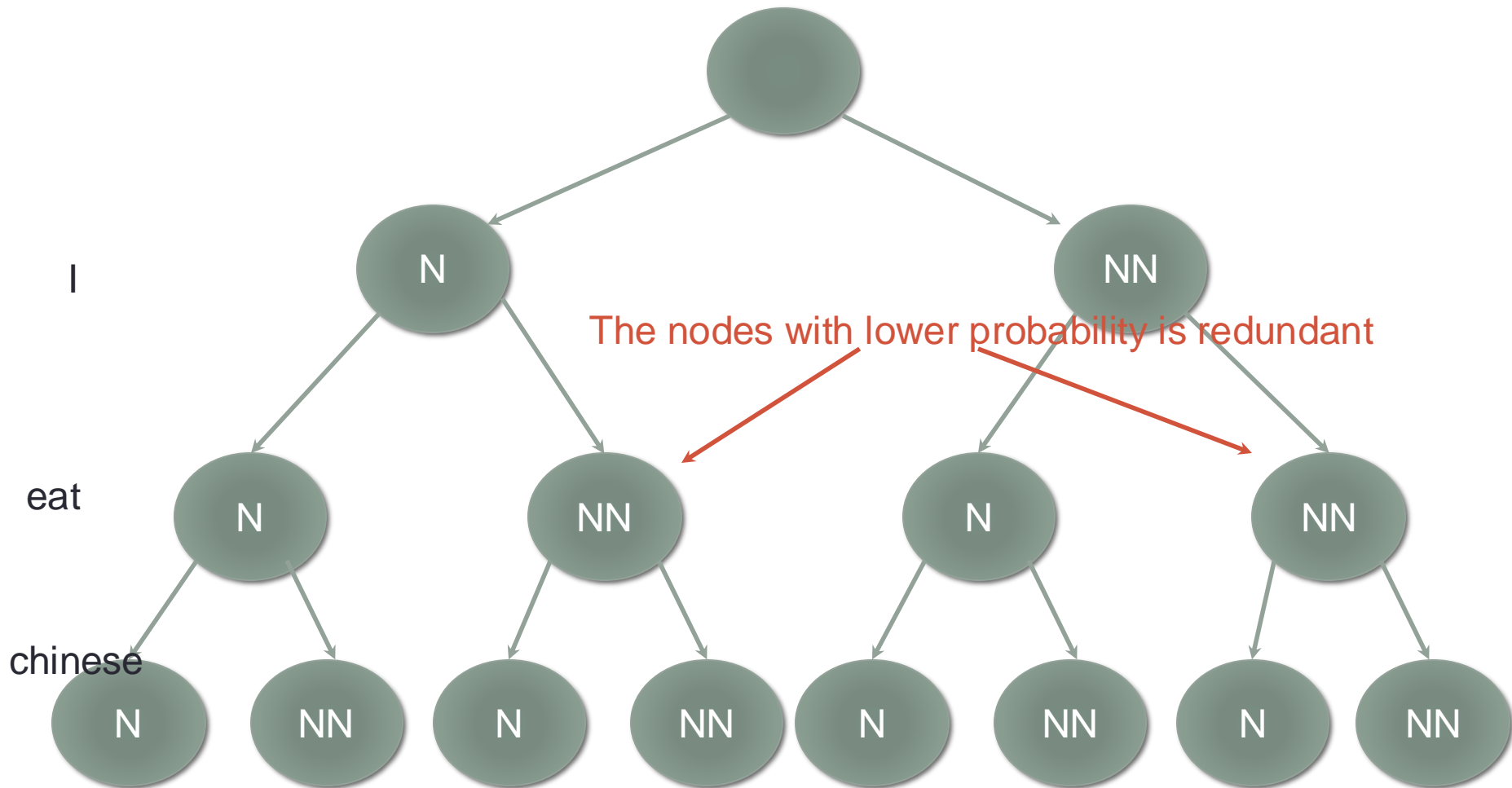
- Some computation are redundant
  - We can save computation from previous steps

# Dynamic programming

- Saving computation for future use. How?
- Example: Find best route from A to B



# Redundancy



# The Viterbi Algorithm (Dynamic programming)

- Some computation are redundant
  - We can save computation from previous steps
- Creates two matrices
  - $\pi[i, t]$  saves the best probability at word position  $i$  for hidden state  $t$
  - $B[i, t]$  saves the previous hidden state that maximize this current state probability

# Viterbi

## Base Step:

$$\pi[0, < S >] = \log 1 = 0$$

$$\pi[0, t] = \log 0 = -\infty, \text{ if } t \neq < S >$$

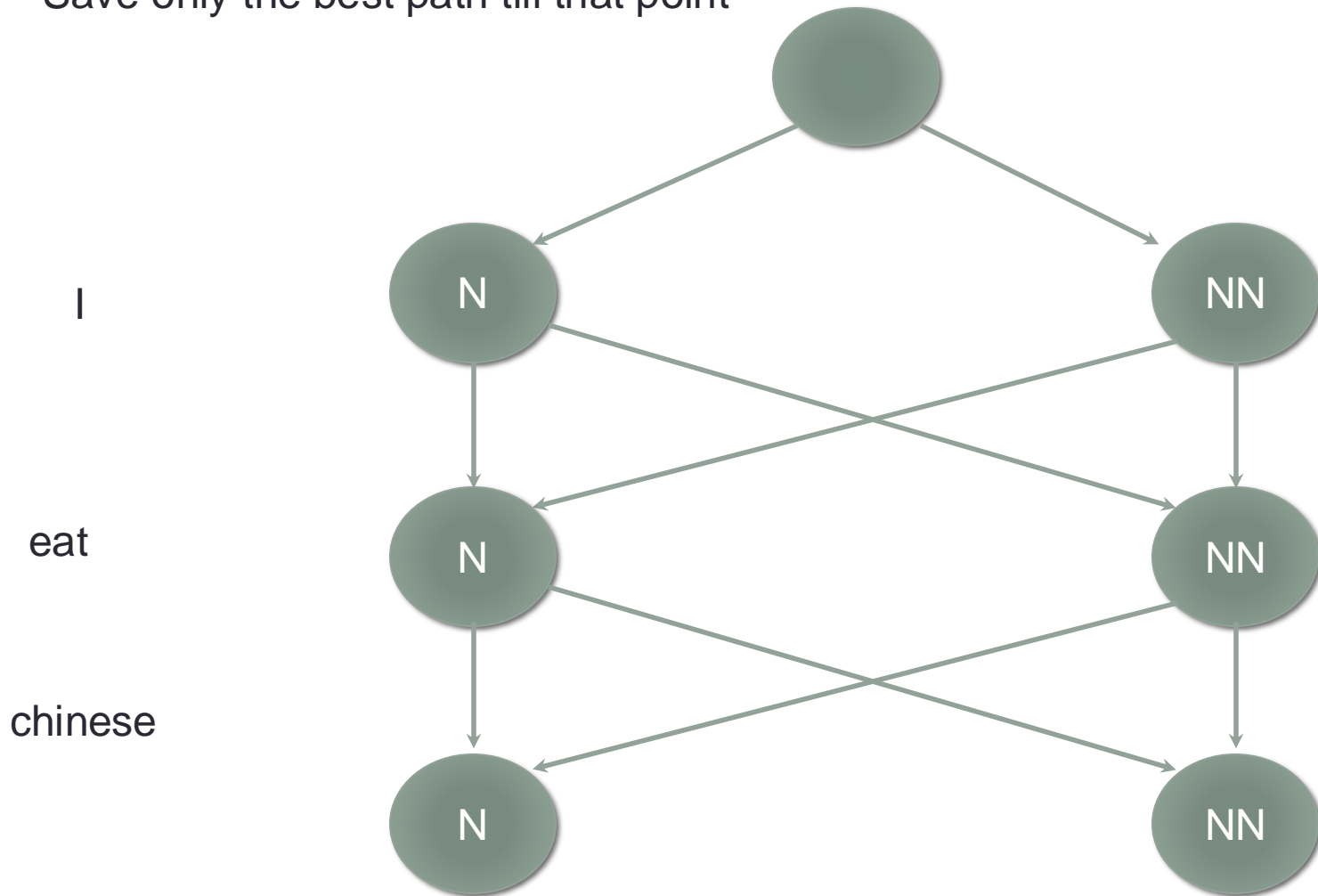
where  $< S >$  is the start symbol.

## Recursive Step:

$$\pi[i, t] = \max_{t'} \{ \pi[i-1, t'] + \log P(t|t') + \log P(w_i|t) \}$$

# Viterbi

Save only the best path till that point



# Decoding example

$A_{ij}$	To N	To NN
From N	0.6	0.4
From NN	0.5	0.5

$B_{ik}$	I	eat	Chinese
State N	0.8	0.01	0.19
State NN	0.1	0.45	0.45

We ignore the log to do easy compute  
 We also ignoring initial state probability.  
 What if we want to include it?

$\pi[i, t]$	I	eat	Chinese
State N	0.8		
State NN	0.1		

$B[i, t]$	I	eat	Chinese
State N	-		
State NN	-		

**Recursive Step:**

$$\pi[i, t] = \max_{t'} \{ \pi[i-1, t'] + \log P(t|t') + \log P(w_i|t) \}$$

# Decoding example

$A_{ij}$	To N	To NN
From N	0.6	0.4
From NN	0.5	0.5

$B_{ik}$	I	eat	Chinese
State N	0.8	0.01	0.19
State NN	0.1	0.45	0.45

$\pi[i, t]$	I	eat	Chinese
State N	0.8	0.005	
State NN	0.1		

$B[i, t]$	I	eat	Chinese
State N	-	N	
State NN	-		

$0.8 * 0.6 * 0.01$  vs  $0.1 * 0.5 * 0.01$   
 $0.0048$  vs  $0.0005$

**Recursive Step:**

$$\pi[i, t] = \max_{t'} \{ \pi[i-1, t'] + \log P(t|t') + \log P(w_i|t) \}$$



# Decoding example

$A_{ij}$	To N	To NN
From N	0.6	0.4
From NN	0.5	0.5

$B_{ik}$	I	eat	Chinese
State N	0.8	0.01	0.19
State NN	0.1	0.45	0.45

$\pi[i, t]$	I	eat	Chinese
State N	0.8	0.005	0.014
State NN	0.1	0.144	0.032

$B[i, t]$	I	eat	Chinese
State N	-	N	NN
State NN	-	N	NN

**Recursive Step:**

$$\pi[i, t] = \max_{t'} \{ \pi[i-1, t'] + \log P(t|t') + \log P(w_i|t) \}$$

# Decoding example (backtrack)

$A_{ij}$	To N	To NN
From N	0.6	0.4
From NN	0.5	0.5

$B_{ik}$	I	eat	Chinese
State N	0.8	0.01	0.19
State NN	0.1	0.45	0.45

$\pi[i, t]$	I	eat	Chinese
State N	0.8	0.005	0.014
State NN	0.1	0.144	0.032

$B[i, t]$	I	eat	Chinese
State N	-	N	NN
State NN	-	N	NN

N, NN, NN

# Decoding (Reconstructing T)

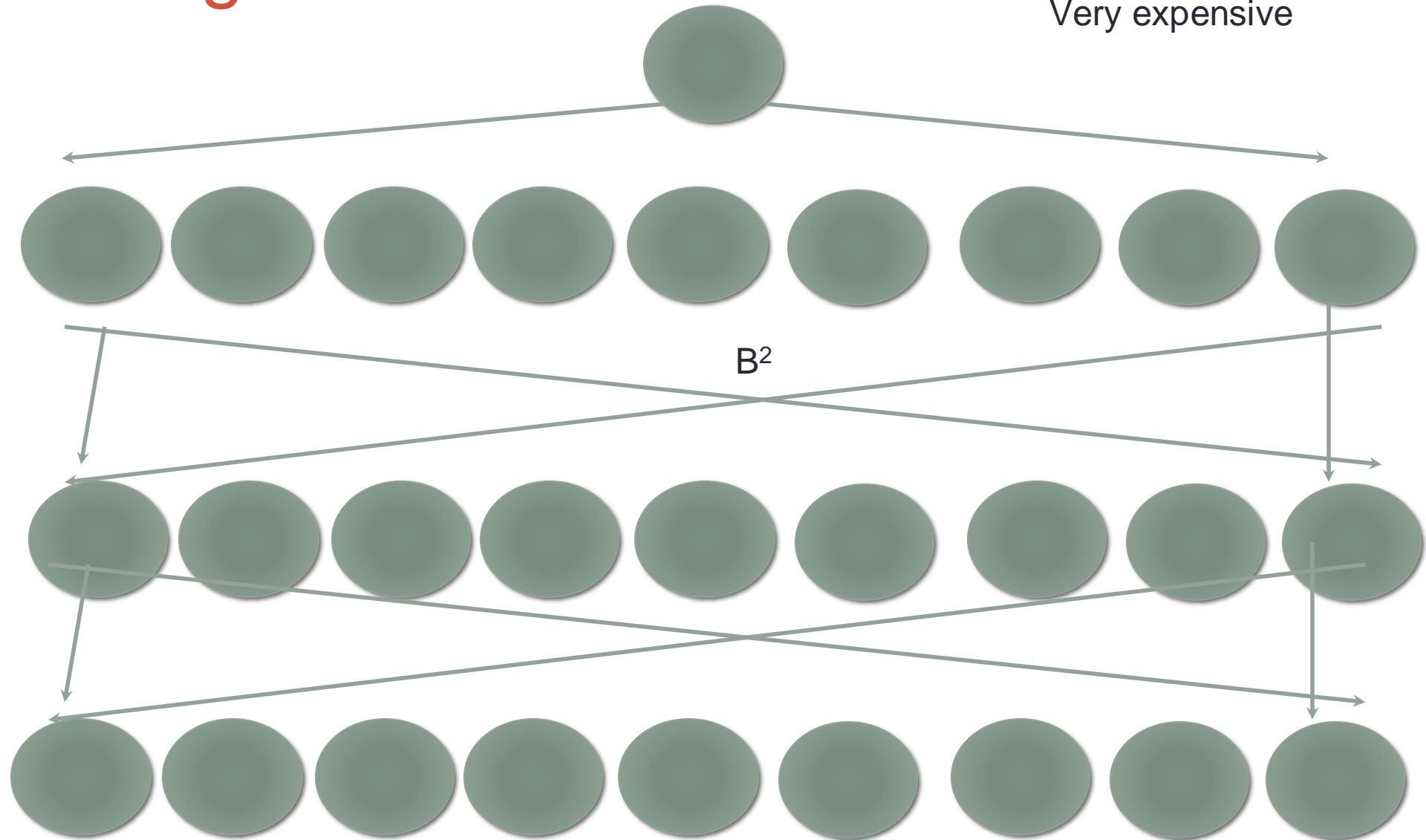
- We can find the best T by
- Find that best probability at the end
- Backtrack according to  $B[i,t]$

$$t_N = \arg \max_t \pi[N, t]$$

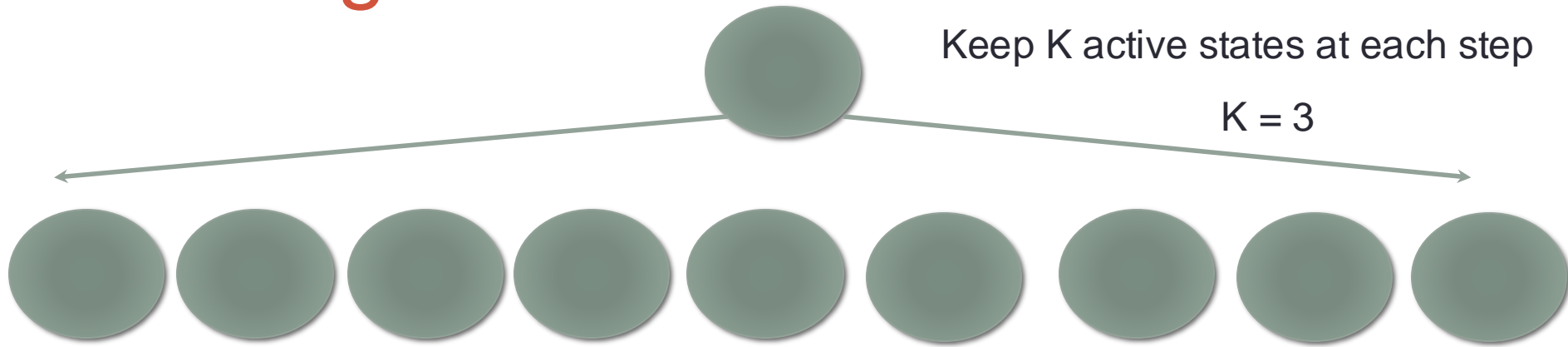
- This gives a big O of
  - We need to compute and create a table of size  $O(B N)$
  - For each value we need to perform  $B$  computations
  - $O(B^2 N)$ , Space complexity of  $O(2 B N)$
- What happens if  $B$  is big?

# Large hidden states

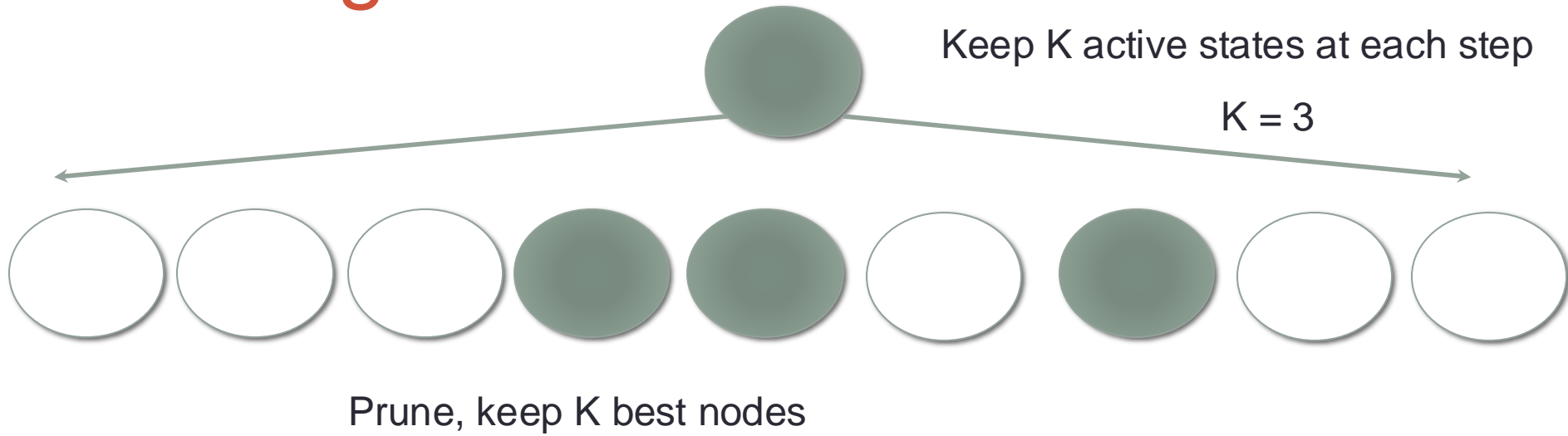
Very expensive



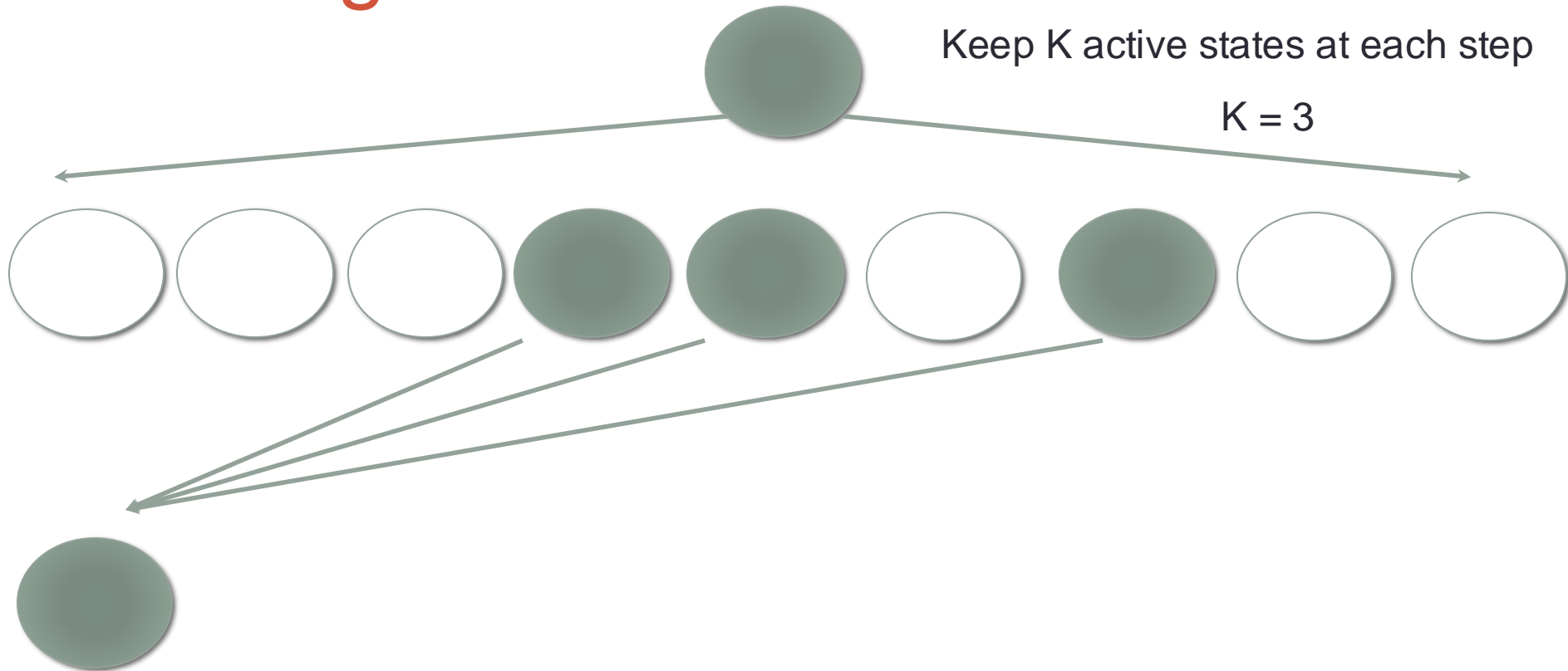
# Pruning with Beamsearch



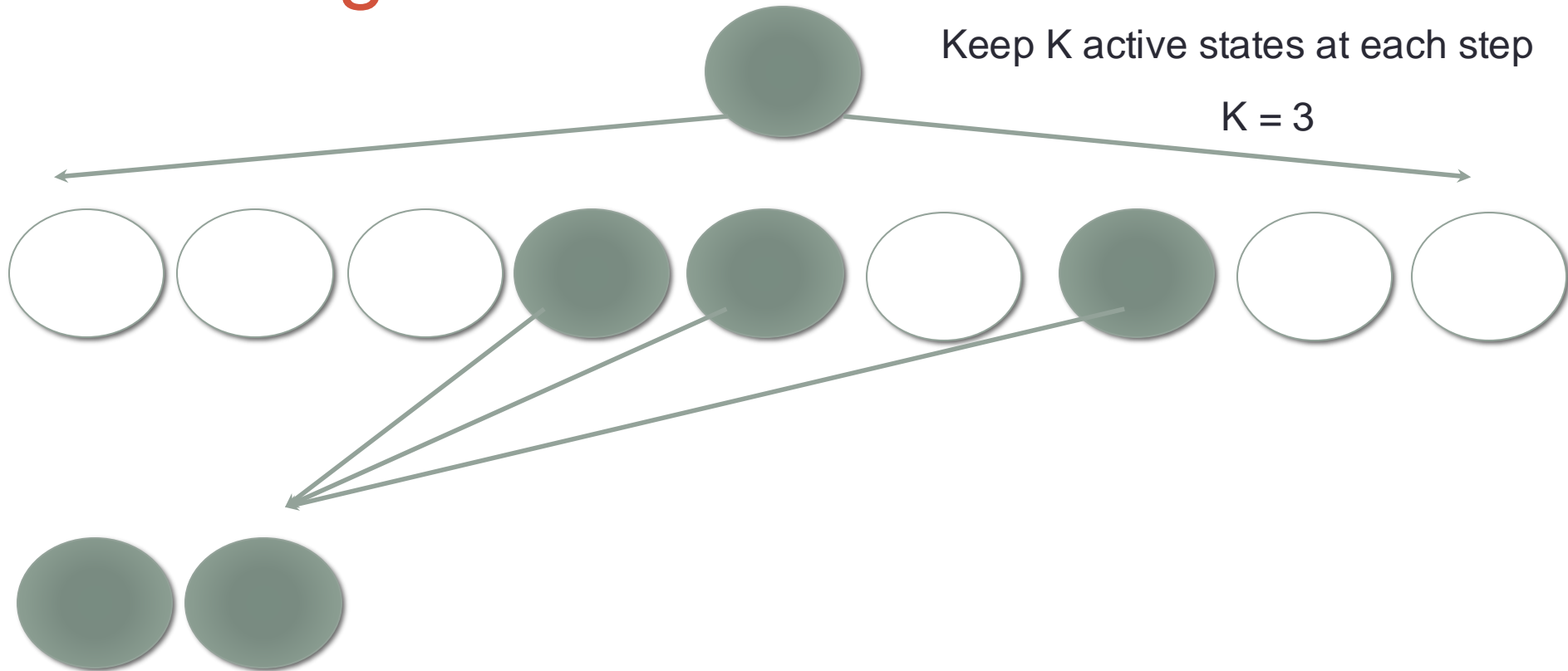
# Pruning with Beamsearch



# Pruning with Beamsearch

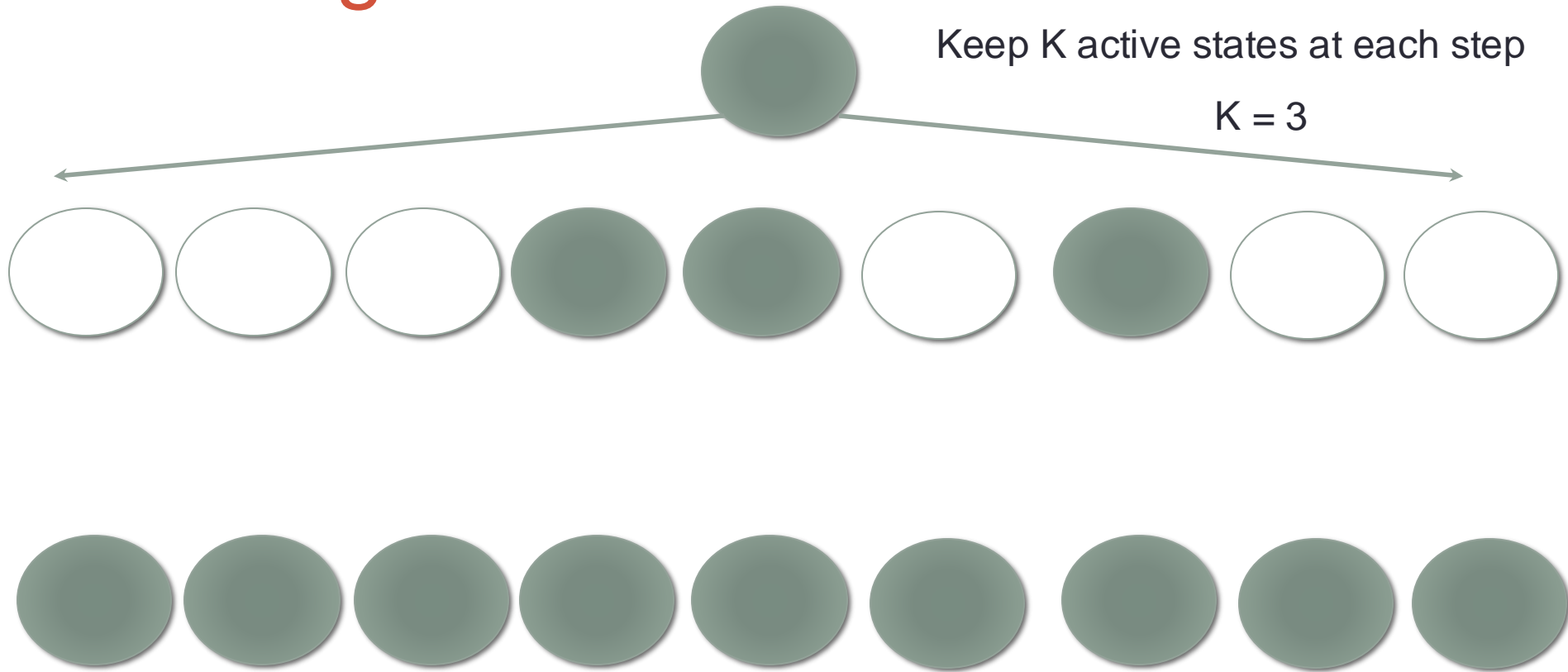


# Pruning with Beamsearch

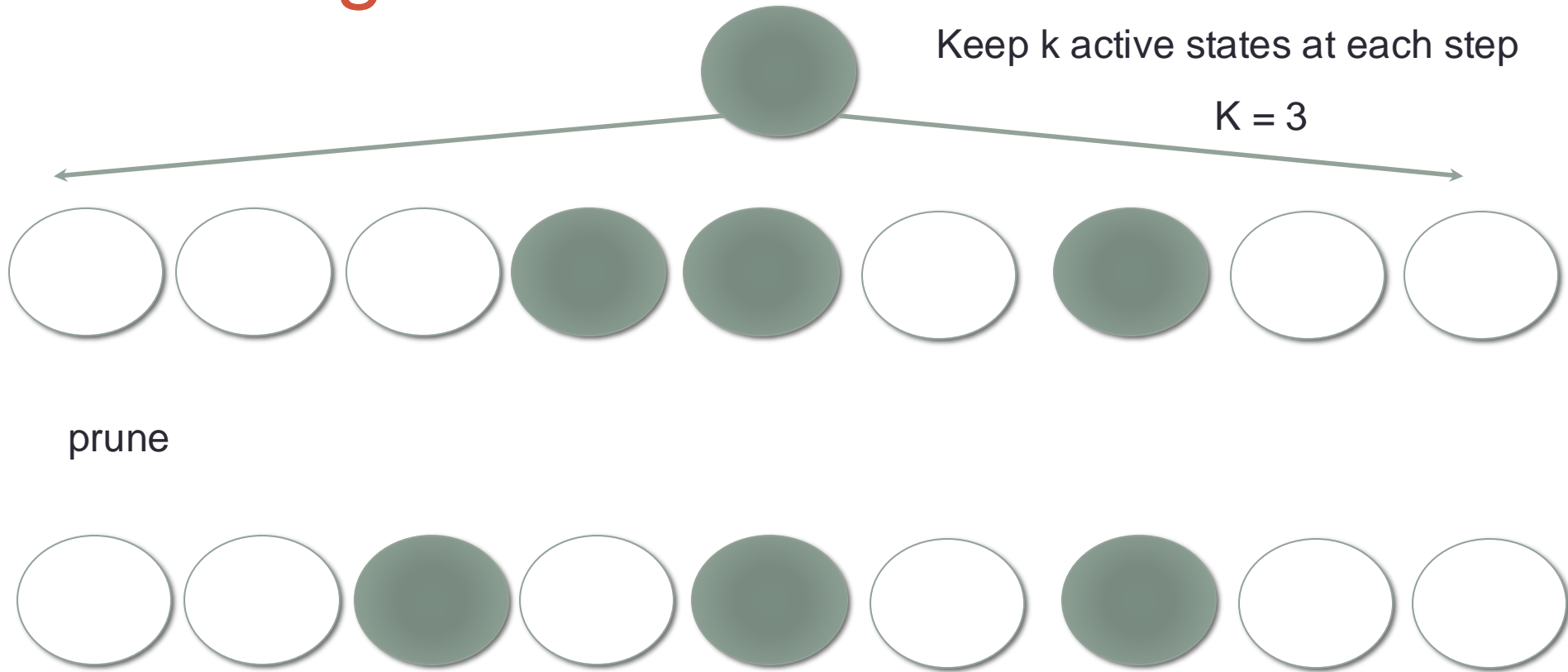




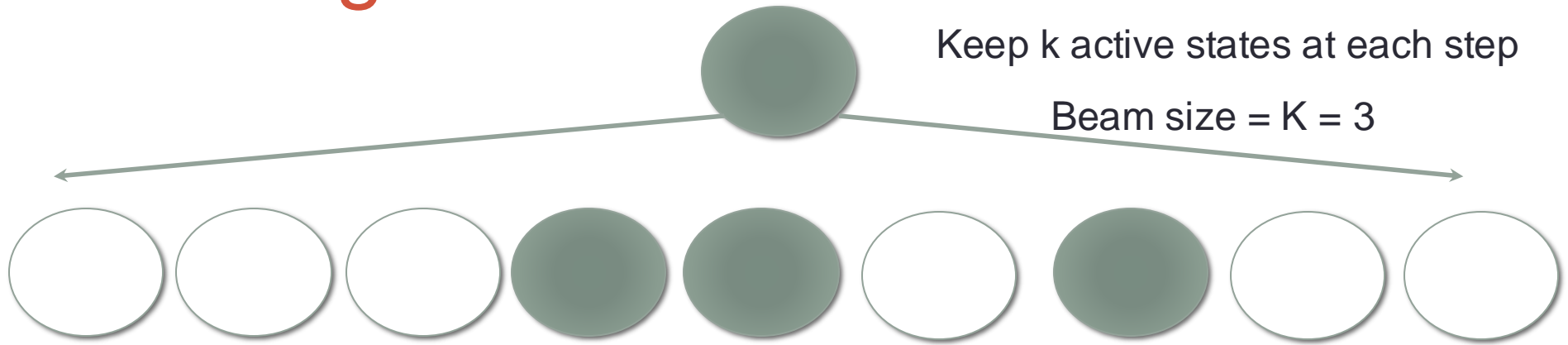
# Pruning with Beamsearch



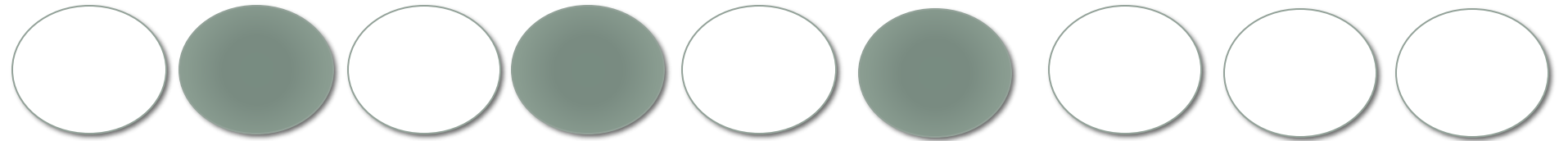
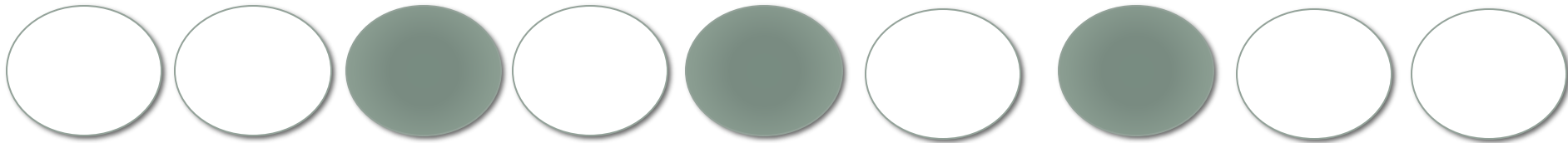
# Pruning with Beamsearch



# Pruning with Beamsearch

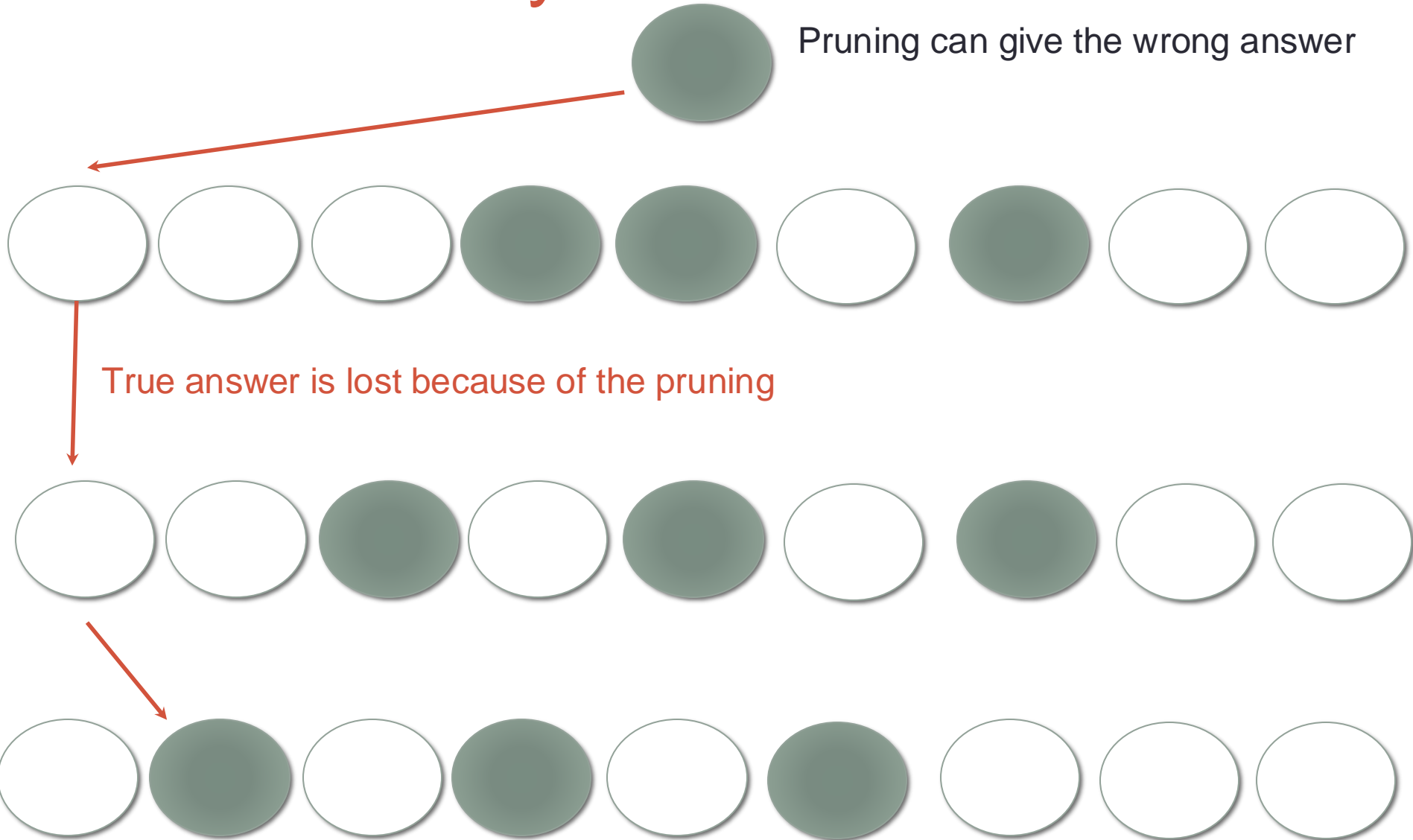


$O(kB)$  per word index



# Inadmissibility

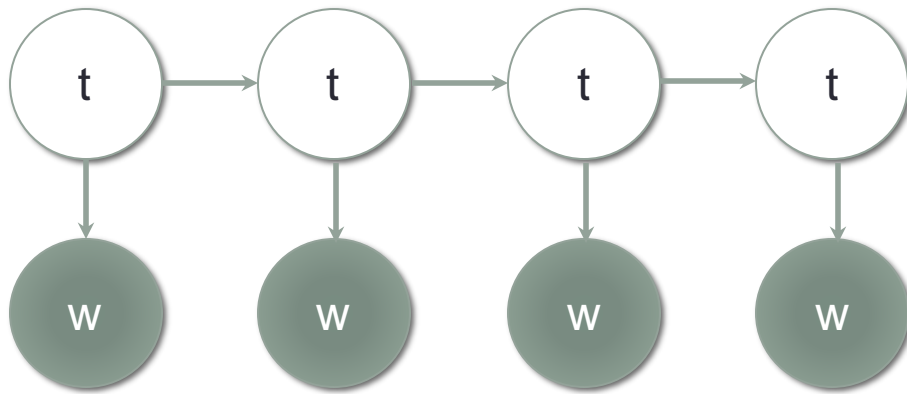
Pruning can give the wrong answer



# Beam search

- Beam search is inadmissible (can be wrong)
- Size of beam affect the quality of the answer
  - Practically still useful even for small size ( $K < 10$  in machine translation)
- We will use beam search again for text generation.

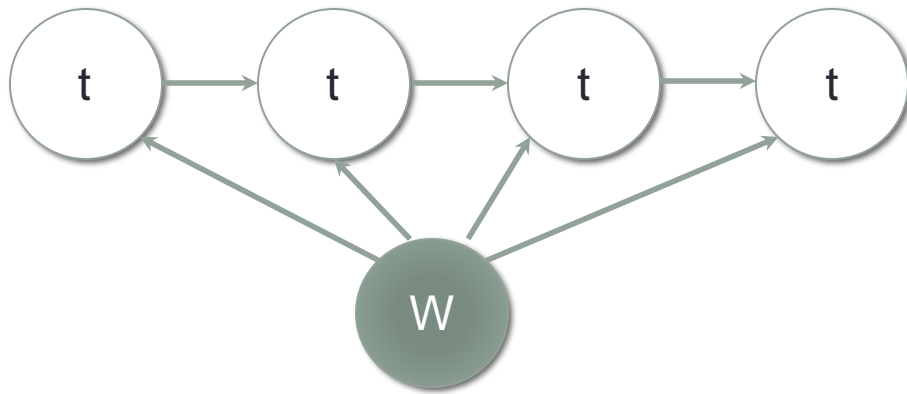
# HMM assumptions and disadvantages



- No dependency across words
- HMM is a generative model  $P(T, W)$ 
  - But we care about  $P(T | W)$ 
    - Mismatch between final objective and model learning objective

# Solution: Conditional Random Fields (CRF)

Random variable  
Random/stochastic process  
Random field



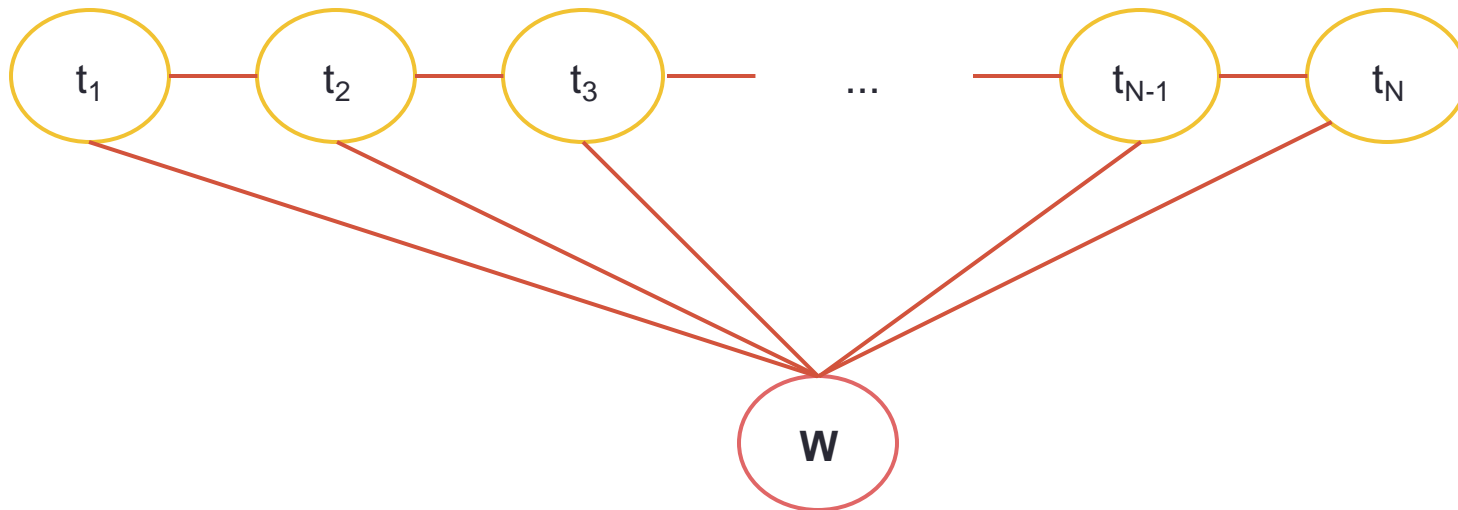
$$P(T|W) = \prod P(t_n | t_{n-1}, W) = P(t_1 | W) P(t_2 | t_1, W) P(t_3 | t_2, W) P(t_4 | t_3, W)$$

- Every point in the chain now depends on the entire sentence
- CRF is a discriminative model

# Linear chain CRF

Linear-chain CRF models with these independent assumption:

- (1) each label  $t_n$  only depends on previous label  $t_{n-1}$
- (2) each label  $t_n$  globally depends on  $\mathbf{x}$



**Problem:** This is a big function (depends on  $n + 2$  things). Hard to estimate using our previous method (counting)



# Workaround

- Probability distribution is a function (with special constraints)
- Find a **function** that will represent “probabilities”
- We can turn functions into probabilities easily.
  - Softmax function normalization
  - We just need to have **a function that give higher values to more likely inputs**

$$P(y = j|x) = \frac{e^{h(y=j|x)}}{\sum_y e^{h(y|x)}}$$

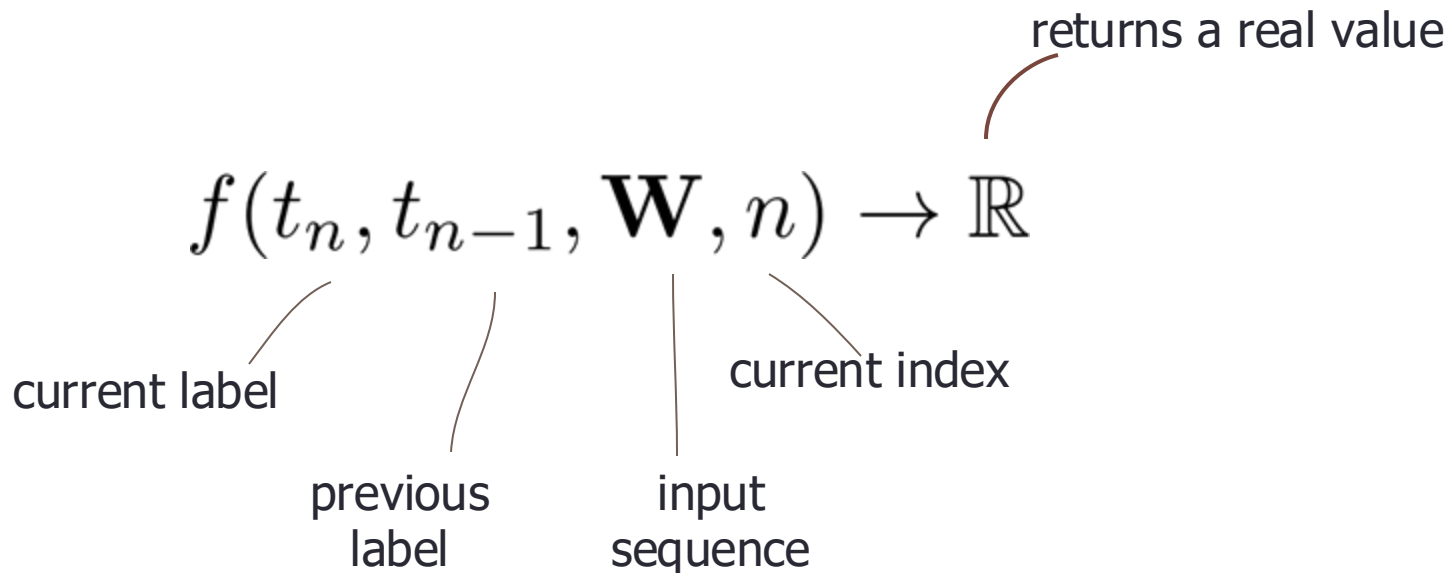
# Goal

- Find a function that will represent “probabilities”
- Turn functions into probabilities
  - Softmax function normalization
  - We just need to have function that give higher to more likely inputs
- Building functions that represent the whole sequence is hard
  - We’ll build by combining pieces
  - But each piece should have the form  $f(t_n, t_{n-1}, \mathbf{W}, n) \rightarrow \mathbb{R}$ 
    - This is from our independence assumption.
  - We call these functions, **feature functions**

# Feature function

At each time step, a feature function  $f(t_n, t_{n-1}, \mathbf{W}, n) \rightarrow \mathbb{R}$  is used to capture some characteristics of current label and the observation.

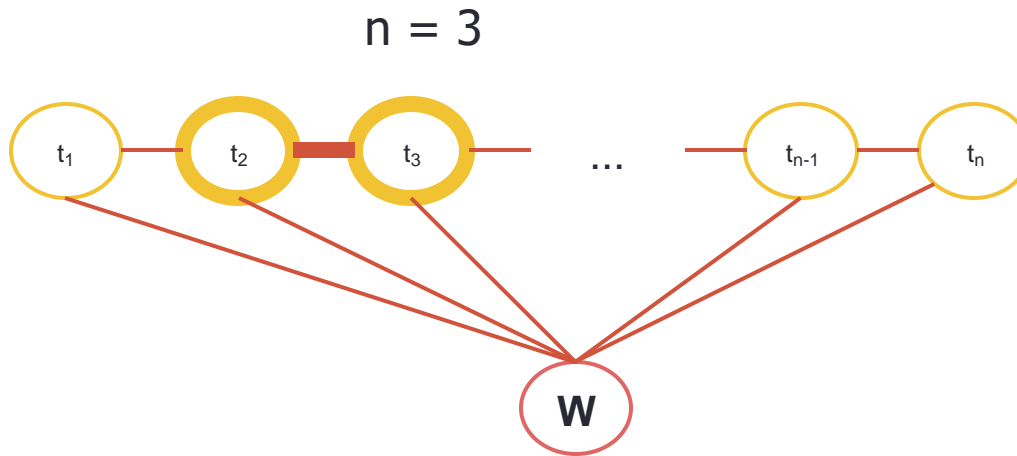
A feature function in linear-CRF:



# Example features

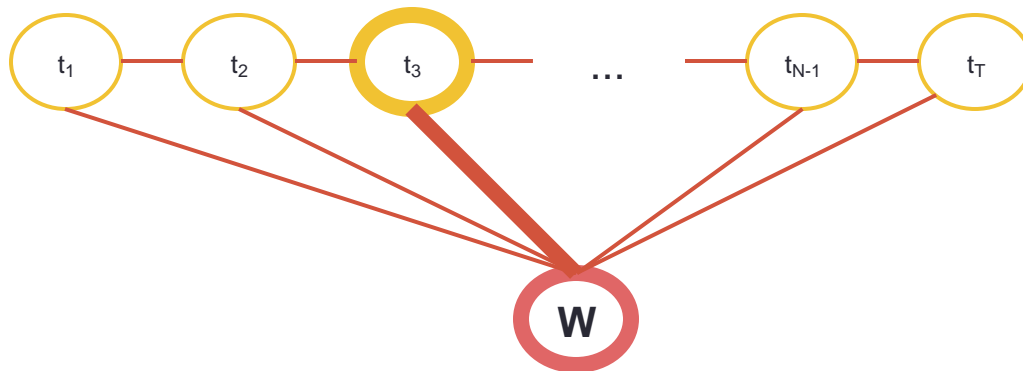
In general, we often define a feature function as a binary function, taking current label and its dependent variable into account. For example:

- transition function  $f_1(t_n, t_{n-1}, \mathbf{W}, n) = \begin{cases} 1, & \text{if } t_n = \text{NOUN and } t_{n-1} = \text{ADJ.} \\ 0, & \text{otherwise.} \end{cases}$



# Feature function: More examples

- state function  $f_2(t_n, t_{n-1}, \mathbf{W}, n) = \begin{cases} 1, & \text{if } t_n = \text{NOUN and } w_n = \text{fox.} \\ 0, & \text{otherwise.} \end{cases}$



- The whole input sequences can be used in a feature function.

$$f_3(t_n, t_{n-1}, \mathbf{W}, n) = \begin{cases} 1, & \text{if } t_n = \text{NOUN and } w_{n-1} = \text{an.} \\ 0, & \text{otherwise.} \end{cases}$$

# Feature function: More example

Other features other than word form can be used too.

$$f_4(t_n, t_{n-1}, \mathbf{W}, n) = \begin{cases} 1, & \text{if } t_n = \text{PROPER NOUN and } w_n \text{ is capitalized.} \\ 0, & \text{otherwise.} \end{cases}$$

$$f_5(t_n, t_{n-1}, \mathbf{W}, n) = \begin{cases} 1, & \text{if } t_n = \text{NOUN and } w_{n-1} \text{ ends with "est".} \\ 0, & \text{otherwise.} \end{cases}$$

# Potential

At each time step, a potential  $\Psi_n(t_n, t_{n-1}, \mathbf{W}) \rightarrow \mathbb{R}^+$  is a function that takes all feature functions into account, by summing their products with the associated weight

number of all feature functions

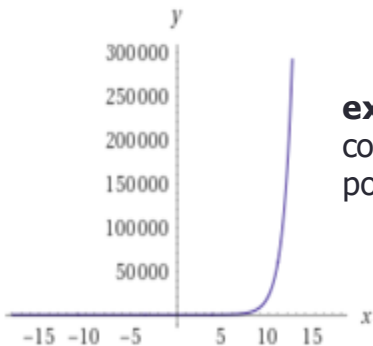
factors at time n

$$\Psi_n(t_n, t_{n-1}, \mathbf{W}) = \exp\left[\sum_k^K \theta_k f_k(t_n, t_{n-1}, \mathbf{W}, n)\right]$$

$k^{\text{th}}$  feature function

**exponential function:** used to convert the summation to the positive range

**weight** associated with each feature function, estimated within training process



# Potential: Example

n	n=1	n=2	n=3	n=4	..
<b>t*</b>	NOUN	VERB	NOUN	VERB	..
<b>w</b>	The	fastest	fox	jumps	..

From feature functions and trained weights on the right, we can compute potentials for the predicted label sequence **t\*** at time step n=3 as following:

$$f_1(t_n, t_{n-1}, \mathbf{W}, n) = \begin{cases} 1, & \text{if } t_n = \text{NOUN and } t_{n-1} = \text{ADJ.} \\ 0, & \text{otherwise.} \end{cases}$$

$$f_2(t_n, t_{n-1}, \mathbf{W}, n) = \begin{cases} 1, & \text{if } t_n = \text{NOUN and } w_n = \text{fox.} \\ 0, & \text{otherwise.} \end{cases}$$

$$f_3(t_n, t_{n-1}, \mathbf{W}, n) = \begin{cases} 1, & \text{if } t_n = \text{NOUN and } w_{n-1} = \text{an.} \\ 0, & \text{otherwise.} \end{cases}$$

$$f_4(t_n, t_{n-1}, \mathbf{W}, n) = \begin{cases} 1, & \text{if } t_n = \text{PROPER NOUN and } w_n \text{ is cap} \\ 0, & \text{otherwise.} \end{cases}$$

$$f_5(t_n, t_{n-1}, \mathbf{W}, n) = \begin{cases} 1, & \text{if } t_n = \text{NOUN and } w_{n-1} \text{ ends with "es"} \\ 0, & \text{otherwise.} \end{cases}$$

$$\theta_1 = 2.54, \theta_2 = 0.13, \theta_3 = 1.12, \theta_4 = 2.01, \theta_5 = 0.97$$

$$\Psi_3(t_3, t_2, \mathbf{W}) = \exp[\sum_1^5 \theta_k f_k(t_3, t_2, \mathbf{W})]$$

$$= \exp\{(0 \times 2.54) + (1 \times 0.13) + (0 \times 1.12) + (0 \times 2.01) + (1 \times 0.97)\}$$

$$= 3.00$$



# Probability of the whole sequence

Joint probability distribution of input and output sequence can be represented as:

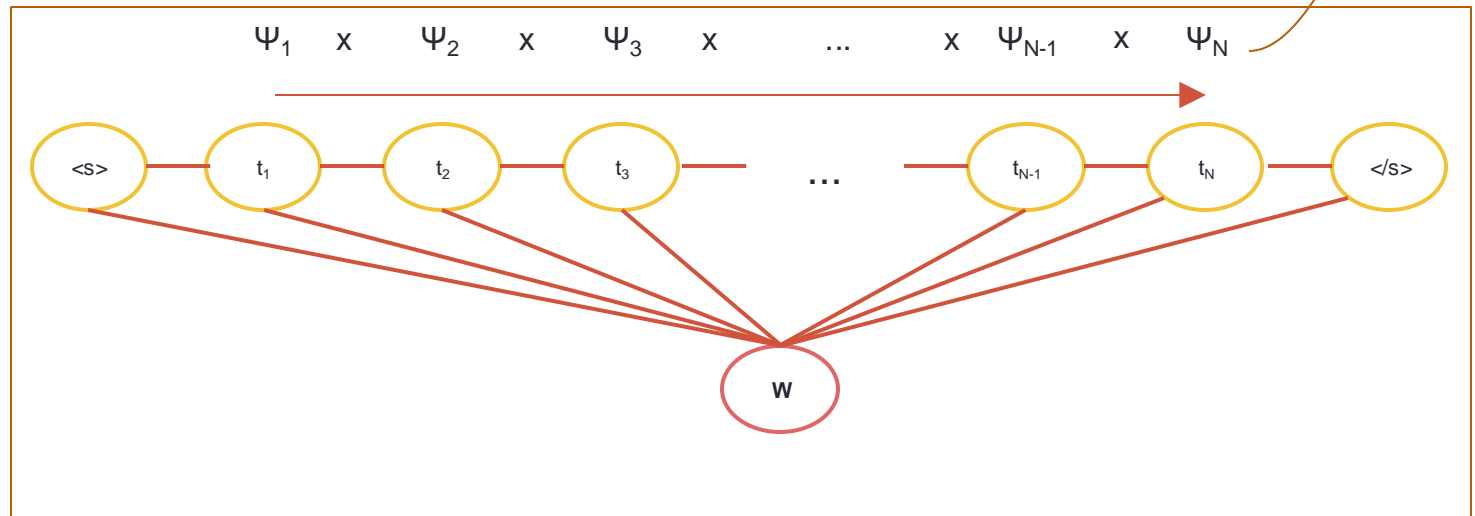
$$P(\mathbf{T}, \mathbf{W}) = \frac{1}{Z} \prod_{n=1}^N \Psi_n(t_n, t_{n-1}, \mathbf{W})$$

With  $p(\mathbf{T}, \mathbf{W})$  we can compare and pick the best  $\mathbf{T}$

Sum of scores for all possible labels with all possible input sequences

$$Z = \sum_{T, W} \prod_{n=1}^N \Psi_n(t_n, t_{n-1}, \mathbf{W})$$

Score of a sequence label  $\mathbf{T}$  for a given sequence input  $\mathbf{W}$

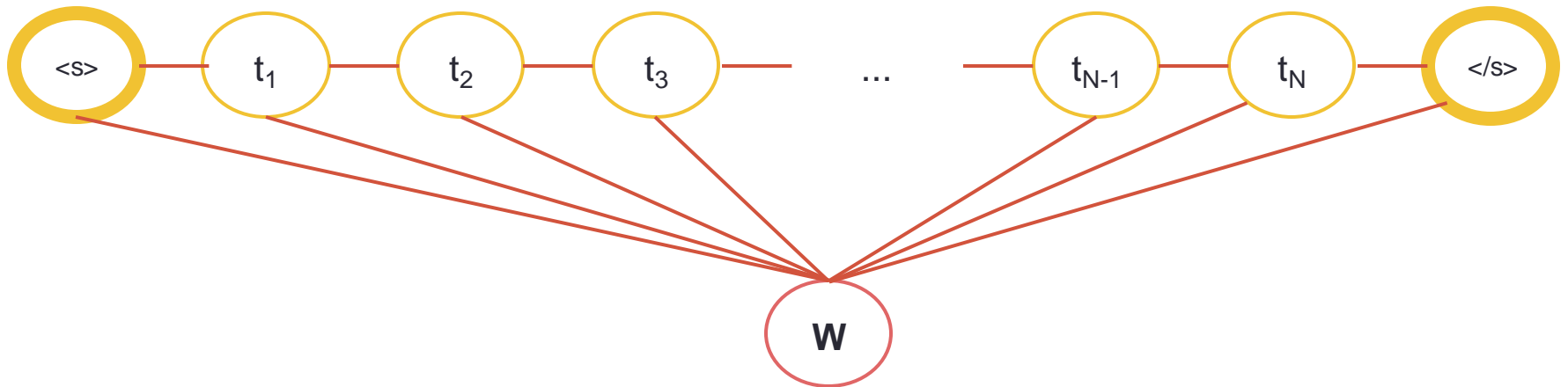


# Special states and characters

To simplify modeling, we add two new special states and characters:

**<s>** indicates the beginning of the sequence

**</s>** indicates the end of the sequence



# Product of sum over feature functions

From the definition of factors, the joint distribution can be represented by

$$P(\mathbf{T}, \mathbf{W}) = \frac{1}{Z} \prod_{n=1}^N \Psi_n(t_n, t_{n-1}, \mathbf{W})$$

$$P(T, W) = \frac{1}{Z} \prod_{n=1}^N \exp\left[\sum_k \theta_k f_k(t_n, t_{n-1}, \mathbf{W}, n)\right]$$

$$Z = \sum_{T, W} \prod_{n=1}^N \Psi_n(t_n, t_{n-1}, \mathbf{W})$$

Computing  $\mathbf{Z}$  is intractable:

Imagine a sentence of 20 words with vocabulary size of 100,000

we have to consider all  $(100000)^{20}$  possible input sequences!

# Linear-chain CRF

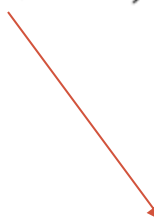
Modeling conditional probability distribution  $P(T|W)$  is enough for classification tasks.

So, in linear-chain CRF, we model the conditional distribution by using these two equations:

$$P(T|W) = \frac{P(T, W)}{P(W)} = \frac{P(T, W)}{\sum_{T'} P(T', W)}$$

$$P(T, W) = \frac{1}{Z} \prod_{n=1}^N \exp\left[\sum_k \theta_k f_k(t_n, t_{n-1}, \mathbf{W}, n)\right]$$

$$Z = \sum_{T, W} \prod_{n=1}^N \Psi_n(t_n, t_{n-1}, \mathbf{W})$$

$$Z(W) = \sum_T \prod_{n=1}^N \exp\left[\sum_k \theta_k f_k(t_n, t_{n-1}, \mathbf{W}, n)\right]$$


# Linear-chain CRF

A linear-chain CRF is a conditional distribution

$$P(T|W) = \frac{1}{Z(W)} \prod_{n=1}^N \exp\left[\sum_k^K \theta_k f_k(t_n, t_{n-1}, \mathbf{W}, n)\right]$$

, where  $Z(\mathbf{x})$  is an instance-specific normalization function

$$Z(W) = \sum_T \prod_{n=1}^N \exp\left[\sum_k^K \theta_k f_k(t_n, t_{n-1}, \mathbf{W}, n)\right]$$

$$Z = \sum_{T, W} \prod_{n=1}^N \Psi_n(t_n, t_{n-1}, \mathbf{W})$$

# Linear-chain CRF

$$P(T|W) = \frac{1}{Z(W)} \prod_{n=1}^N \exp\left[\sum_k^K \theta_k f_k(t_n, t_{n-1}, \mathbf{W}, n)\right]$$

## normalization function

the sum of  
products of all  
possible output  
sequences  
not the same as  $\mathbf{Z}$  in  
the joint distribution

multiply  
over  
all time  
steps

**sum** of weighted  
feature functions at  
one time step, then  
taken to the  
**exponential** function

# Linear-chain CRF big picture

- Wants  $P(T|W)$
- Assumes independence, where we only consider  $P(t_{t-1}, t_t, \mathbf{W})$
- How to model  $P(t_{t-1}, t_t, \mathbf{W})$ ?
  - Still too hard, let's make it into a function where high value means high probability – potential functions  $\Psi_n(t_n, t_{n-1}, \mathbf{W}) \rightarrow \mathbb{R}^+$
  - Still too hard, let's build it from pieces – feature functions  $f(t_n, t_{n-1}, \mathbf{W}, n)$
- We can get  $P(\mathbf{T}, \mathbf{W})$  by multiplying all  $\Psi_n(t_n, t_{n-1}, \mathbf{W}) \rightarrow \mathbb{R}^+$ 
  - This is not a probability, need a normalization
- We can also get  $P(T|W)$  from multiplying all  $\Psi_n(t_n, t_{n-1}, \mathbf{W})$ 
  - Still need a normalization, but easier.
- This is our model, but!
  - How to inference? How to train? What features functions?

# How to inference?

- If we are given the model, and  $\mathbf{W}$

$$P(T|W) = \frac{1}{Z(W)} \prod_{n=1}^N \exp\left[\sum_k^K \theta_k f_k(t_n, t_{n-1}, \mathbf{W}, n)\right]$$

- Find  $\mathbf{T}$ 
  - Not so straight forward, many possible  $\mathbf{T}$ 
    - Noun, adjective, verb
    - Noun, noun, verb
    - Verb, noun, noun
    - Too many possibilities to compare
- Solution: Dynamic programming, just like HMM



# Viterbi algorithm

Viterbi algorithm is an algorithm for decoding based on dynamic programming.

From the equation, we can see the  $Z(W)$  is the same for all possible label sequences, so we can consider only the part in the rectangle

$$P(T|W) = \frac{1}{Z(W)} \prod_{n=1}^N \exp\left[\sum_k \theta_k f_k(t_n, t_{n-1}, \mathbf{W}, n)\right]$$

Goes over time step just like HMM viterbi

Find the label sequence  $\mathbf{T}$  that maximize this value

# Viterbi: Structure

$$P(T|W) = \frac{1}{Z(W)} \prod_{n=1}^N \exp\left[\sum_k^K \theta_k f_k(t_n, t_{n-1}, \mathbf{W}, n)\right]$$

Create two 2D arrays: VTB and Var

	0	1	...	N	N+1
ADJ					
ADP					
...					
<S>					
</S>					

**VTB(<s>, T)** stores the highest value a label sequence that  $y_T$  is <s> can take

	0	1	...	N	N+1
ADJ					
ADP					
...					
<S>					
</S>					

**Var(<s>, T)** stores the label  $y_{T-1}$  in the sequence that yields the value in VTB(<s>, T)

# Viterbi: Initialization

$$P(T|W) = \frac{1}{Z(W)} \prod_{n=1}^N \exp\left[\sum_k^K \theta_k f_k(t_n, t_{n-1}, \mathbf{W}, n)\right]$$

**VTB**

	0	1	...	N	N+1
ADJ	0	0.12			
ADP	0	0.03			
...	...	...			
</S>	0	10e-9			
<S>	1	10e-3			

For all other labels, apply the same way as ADJ

$$VTB(ADJ,1) = \Psi_1(ADJ, \langle s \rangle, x) VTB(\langle s \rangle, 0)$$

Factors at time t=1, for  
current label=ADJ,  
previous label=<s>

1

**Var**

	0	1	...	N	N+1
ADJ	-	<S>			
ADP	-	<S>			
...	...	...			
</S>	-	<S>			
<S>	-	<S>			

The first label of the output sequence must be <s>

# Viterbi: Iteration

$$P(T|W) = \frac{1}{Z(W)} \prod_{n=1}^N \exp\left[\sum_k^K \theta_k f_k(t_n, t_{n-1}, \mathbf{W}, n)\right]$$

Iterate from n=2 to n=N+1

**VTB**

**Var**

	0	1	...	N	N+1
ADJ	0	0.12	...	2.32	0.22
ADP	0	0.03	...	0.02	0.10
...	...	...	...	...	...
</S>	0	10e-9	...	0.03	3.09
<S>	1	10e-3	...	1.12	0.02

	0	1	...	N	N+1
ADJ	-	<S>	...	NOUN	PREPO
ADP	-	<S>	...	NOUN	ADJ
...	...	...	...	...	...
</S>	-	<S>	...	VERB	ADJ
<S>	-	<S>	...	X	X

$$VTB(ADJ, n) = \max_{i \in Tags} \underbrace{\Psi_n(ADJ, i, W)}_{\text{Factors at time n}} \underbrace{VTB(i, n-1)}_{\text{Maximum value that label i can take at time n-1}}$$

Find the max among values from all previous label i

Factors at time n

Maximum value that label i can take at time n-1

Var(t, n) Stores the value i that maximize the value of VTB(t, n)

# Viterbi: Finalize

Backtrack from  $\text{Var}(</s>, N+1)$  to get the label sequences that maximize  $P(T|W)$

	Var				
	0	1	...	T	N+1
ADJ	-	<S>	...	NOUN	PREPO
ADP	-	<S>	...	NOUN	ADJ
...	...	...	...	...	...
</S>	-	<S>	...	VERB	ADJ
<S>	-	<S>	...	X	X

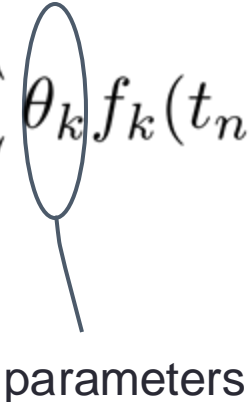
For example:  
output sequence =  
<s>, NOUN, ..., NOUN, ADJ, </s>

# Linear-chain CRF big picture

- Wants  $P(T|W)$
- Assumes independence, where we only consider  $P(t_{t-1}, t_t, \mathbf{W})$
- How to model  $P(t_{t-1}, t_t, \mathbf{W})$ ?
  - Still too hard, let's make it into a function where high value means high probability – potential functions  $\Psi_n(t_n, t_{n-1}, \mathbf{W}) \rightarrow \mathbb{R}^+$
  - Still too hard, let's build it from pieces – feature functions  $f(t_n, t_{n-1}, \mathbf{W}, n)$
- We can get  $P(\mathbf{T}, \mathbf{W})$  by multiplying all  $\Psi_n(t_n, t_{n-1}, \mathbf{W}) \rightarrow \mathbb{R}^+$ 
  - This is not a probability, need a normalization
- We can also get  $P(T|W)$  from multiplying all  $\Psi_n(t_n, t_{n-1}, \mathbf{W})$  and use chain rule.
  - Still need a normalization, but easier.
- This is our model, but!
  - How to inference? Viterbi
  - How to train? What features functions?

# Training Parameters?

Parameters to be learned are weights associated to each feature functions. So, the number of parameters equals the number of feature functions.

$$P(T|W) = \frac{1}{Z(W)} \prod_{n=1}^N \exp\left[\sum_k^K \theta_k f_k(t_n, t_{n-1}, \mathbf{W}, n)\right]$$


parameters

# Training objective

For linear-chain CRF, parameters are trained by **maximum likelihood**.

$$l(\theta) = \sum_{i=1}^N \log P(T^{(i)} | W^{(i)}) \quad (\text{maximize})$$

To clarify, parameters  $\theta$  are trained to maximize the log probability of all pairs of label  $T^{(i)}$  and input  $W^{(i)}$  in the training set.  $(i)$  represents the  $i^{\text{th}}$  training sentence.



# Learning algorithm

To learn parameters from the loss function  $\ell(\theta)$ , several learning algorithm can be used. Some popular learning algorithms for linear-chain CRFs are

- Limited-memory BFGS
- Stochastic Gradient Descent

# Feature functions?

- Anything you can think of, the more the better.
  - The model will learn what is important.

$$f_1(t_n, t_{n-1}, \mathbf{W}, n) = \begin{cases} 1, & \text{if } t_n = \text{NOUN and } t_{n-1} = \text{ADJ.} \\ 0, & \text{otherwise.} \end{cases}$$

$$f_2(t_n, t_{n-1}, \mathbf{W}, n) = \begin{cases} 1, & \text{if } t_n = \text{NOUN and } w_n = \text{fox.} \\ 0, & \text{otherwise.} \end{cases}$$

$$f_3(t_n, t_{n-1}, \mathbf{W}, n) = \begin{cases} 1, & \text{if } t_n = \text{NOUN and } w_{n-1} = \text{an.} \\ 0, & \text{otherwise.} \end{cases}$$

$$f_4(t_n, t_{n-1}, \mathbf{W}, n) = \begin{cases} 1, & \text{if } t_n = \text{PROPER NOUN and } w_n \text{ is capitalized.} \\ 0, & \text{otherwise.} \end{cases}$$

$$f_5(t_n, t_{n-1}, \mathbf{W}, n) = \begin{cases} 1, & \text{if } t_n = \text{NOUN and } w_{n-1} \text{ ends with "est".} \\ 0, & \text{otherwise.} \end{cases}$$

# CRFsuite

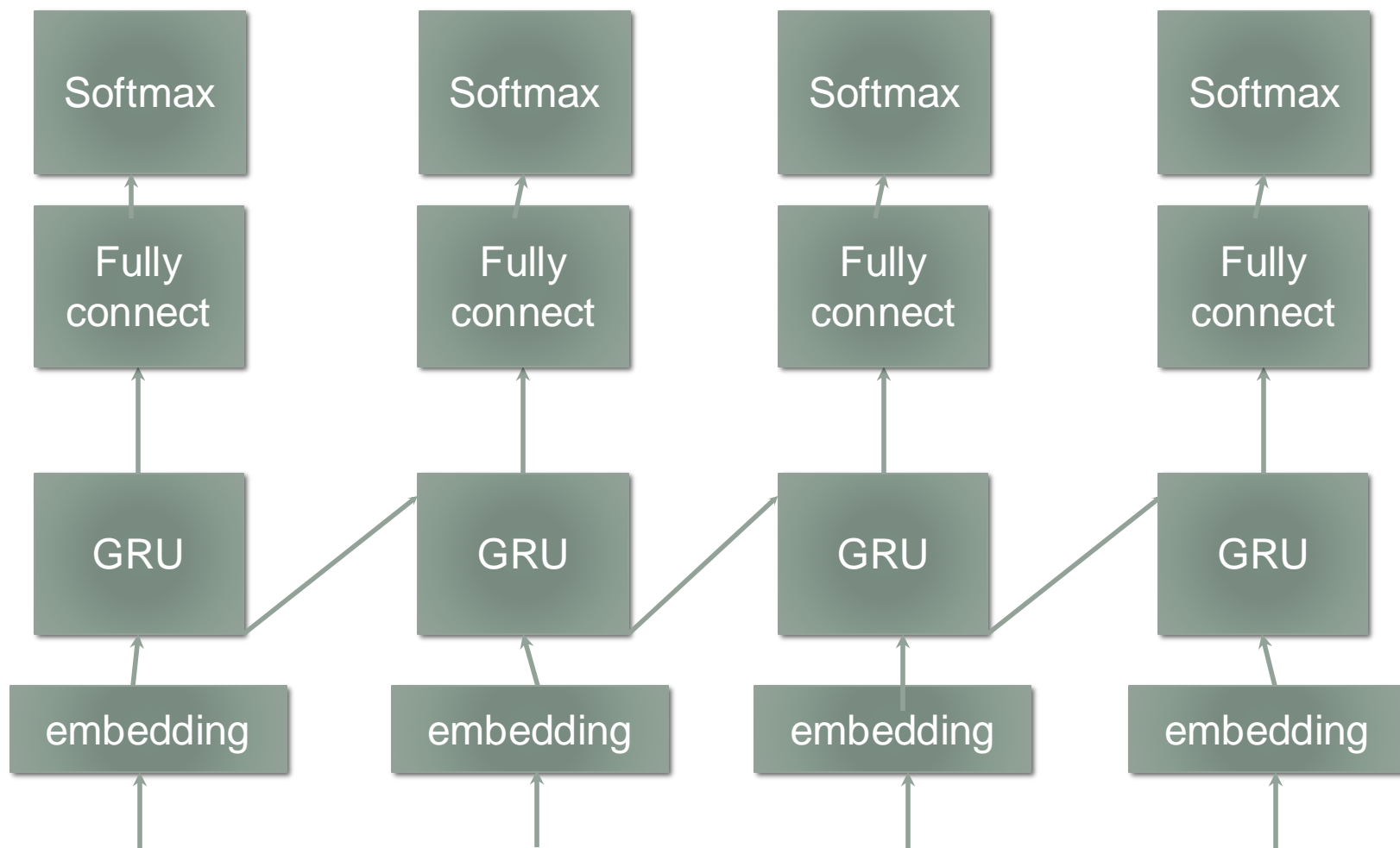
- An implementation of CRFs for labeling sequential data in C++  
SWIG API is provided to be an interface for various languages  
<http://www.chokkan.org/software/crfsuite/>
- python-crfsuite: Python binding for crfsuite  
<https://github.com/scrapinghub/python-crfsuite>
- An example use of python-crfsuite can be found at  
<https://github.com/scrapinghub/python-crfsuite/blob/master/examples/CoNLL%202002.ipynb>

# CRF with neural networks

- Change the softmax layer and loss function
- CRF layer:  $P(y|y_{t-1}, \mathbf{x})$  not just  $P(y_t|\mathbf{x})$

Typical softmax only considers current word label  
not the sequence

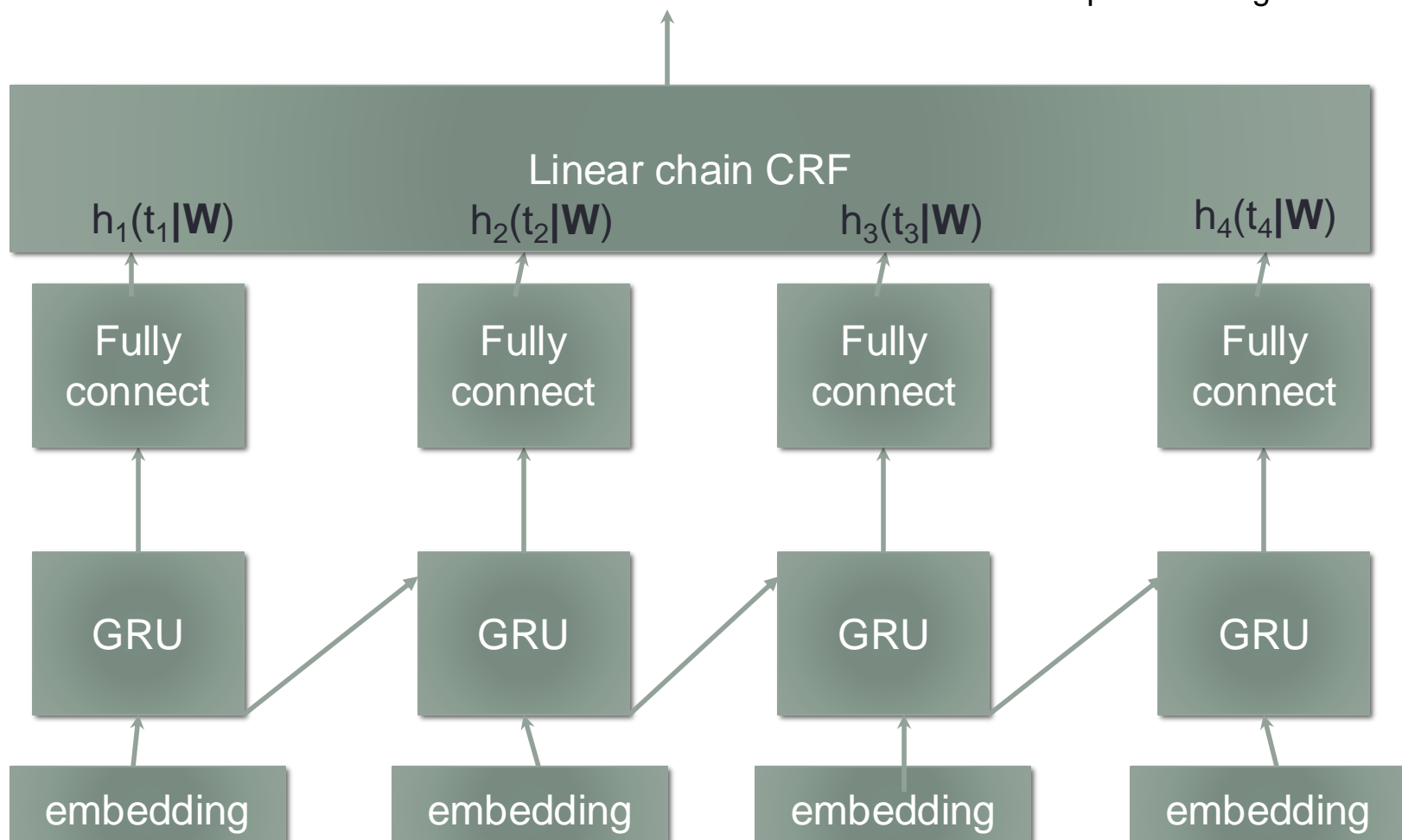
# Neural network for POS



# Neural network for POS with CRF output

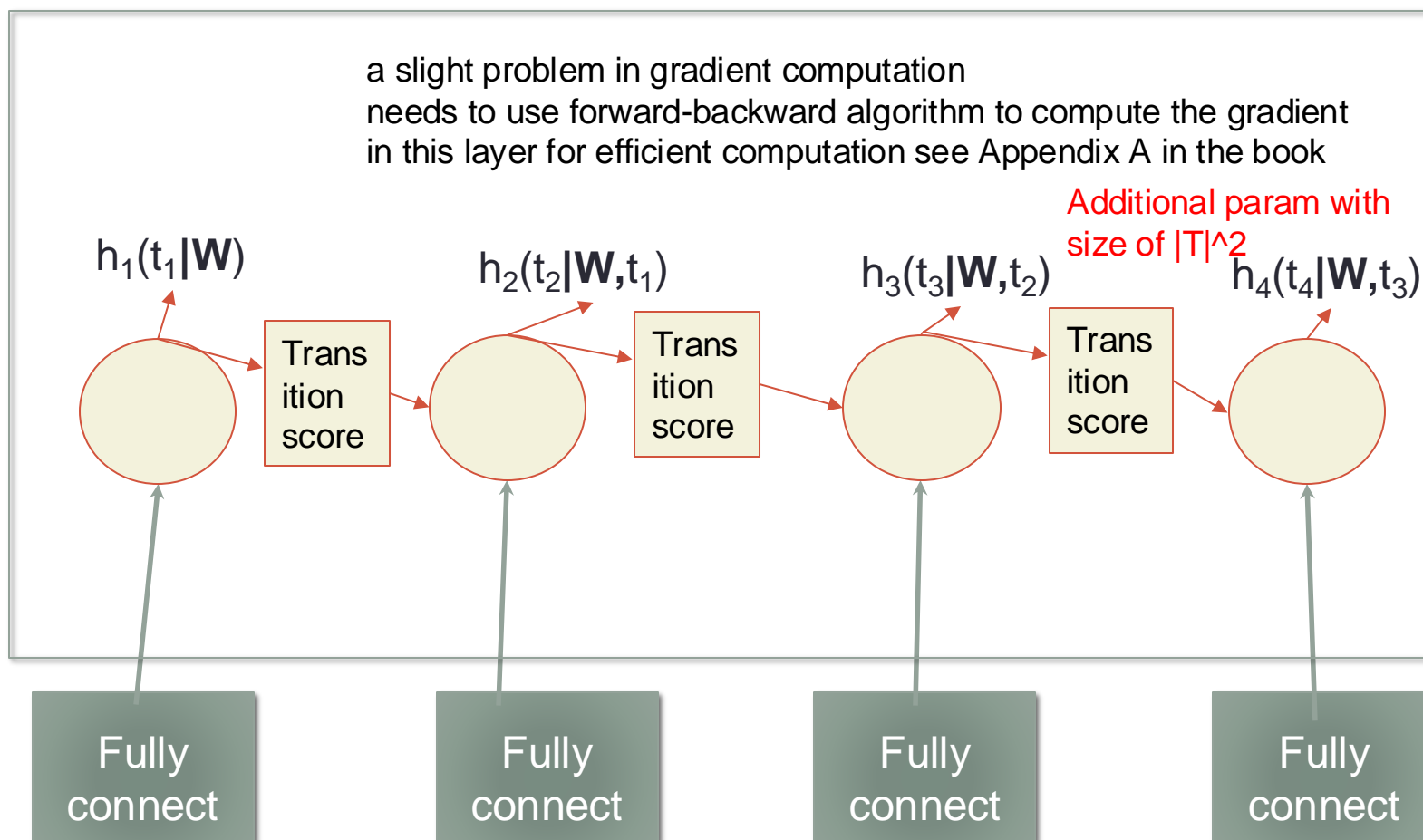
$$P(T|W) = \frac{1}{Z(W)} \prod_{n=1}^N \exp h_n(t_n|\mathbf{W})$$

maximize likelihood of sequence of tags instead



# Neural network for POS with CRF output

$$P(T|W) = \frac{1}{Z(W)} \prod_{n=1}^N \exp [h_n(t_n|\mathbf{W})T(t_n|t_{n-1})]$$



# Neural network for POS with CRF output

- Need to use Viterbi for finding the best sequence
  - Or instead of using the full sequence when decoding, consider each time step instead (marginal inference – faster decoding)
    - This is pretty much a regular softmax during decoding
- Loss function: computed likelihood as a sequence
  - Loss =  $-\log(P(\mathbf{T}^*|\mathbf{W}))$  where  $\mathbf{T}^*$  is the true output

$$P(T|W) = \frac{1}{Z(W)} \prod_{n=1}^N \exp [h_n(t_n|\mathbf{W})T(t_n|t_{n-1})]$$

Example code: [https://pytorch.org/tutorials/beginner/nlp/advanced\\_tutorial.html](https://pytorch.org/tutorials/beginner/nlp/advanced_tutorial.html)



# Performance

Model	Acc.
Giménez and Màrquez (2004)	97.16
Toutanova et al. (2003)	97.27
Manning (2011)	97.28
Collobert et al. (2011) <sup>‡</sup>	97.29
Santos and Zadrozny (2014) <sup>‡</sup>	97.32
Shen et al. (2007)	97.33
Sun (2014)	97.36
Søgaard (2011)	97.50
<b>This paper</b>	<b>97.55</b>

Table 4: POS tagging accuracy of our model on test data from WSJ proportion of PTB, together with top-performance systems. The neural network based models are marked with ‡.

Model	F1
Chieu and Ng (2002)	88.31
Florian et al. (2003)	88.76
Ando and Zhang (2005)	89.31
Collobert et al. (2011) <sup>‡</sup>	89.59
Huang et al. (2015) <sup>‡</sup>	90.10
Chiu and Nichols (2015) <sup>‡</sup>	90.77
Ratinov and Roth (2009)	90.80
Lin and Wu (2009)	90.90
Passos et al. (2014)	90.90
Lample et al. (2016) <sup>‡</sup>	90.94
Luo et al. (2015)	91.20
<b>This paper</b>	<b>91.21</b>

Table 5: NER F1 score of our model on test data set from CoNLL-2003. For the purpose of comparison, we also list F1 scores of previous top-performance systems. ‡ marks the neural models.

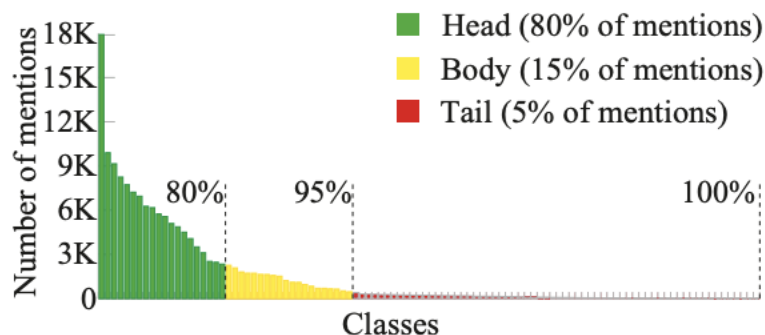
# Thai Nested NER

	Models	Head			Body			Tail			All		
		P	R	F1	P	R	F1	P	R	F1	P	R	F1
Baseline	CRF model	86.06	66.46	75.00	78.30	44.88	57.06	74.07	29.46	42.15	84.60	60.59	70.61
	WangchanBERTa-sp	<b>90.70</b>	77.66	83.67	<b>81.55</b>	55.90	<b>66.33</b>	78.02	26.09	39.10	<b>89.04</b>	70.89	78.94
	WangchanBERTa-sh	90.51	79.24	84.50	81.37	55.09	65.70	78.33	30.79	44.20	88.87	72.25	79.70
	XLM-R-sp	90.27	77.39	83.34	80.45	52.71	63.69	75.42	33.04	45.95	88.42	70.56	78.48
	XLM-R-sh	89.45	79.72	84.31	77.29	<b>58.06</b>	66.31	71.80	<b>39.73</b>	<b>51.16</b>	86.93	<b>73.66</b>	<b>79.75</b>
SOTA	Second-best-learning	87.57	<b>81.78</b>	<b>84.58</b>	80.12	54.85	65.12	<b>79.05</b>	19.41	31.16	86.41	73.49	79.43
	Pyramid	87.59	80.33	83.81	76.07	53.72	62.97	74.20	23.92	36.18	85.65	72.45	78.50
	Locate and label	77.60	80.38	78.97	64.42	56.21	60.04	77.43	18.86	30.33	75.57	72.61	74.06

1 model per layer  
1 model all layer

Table 4: Experimental results nested-NER models divided into head, body, tail, and overall in our dataset

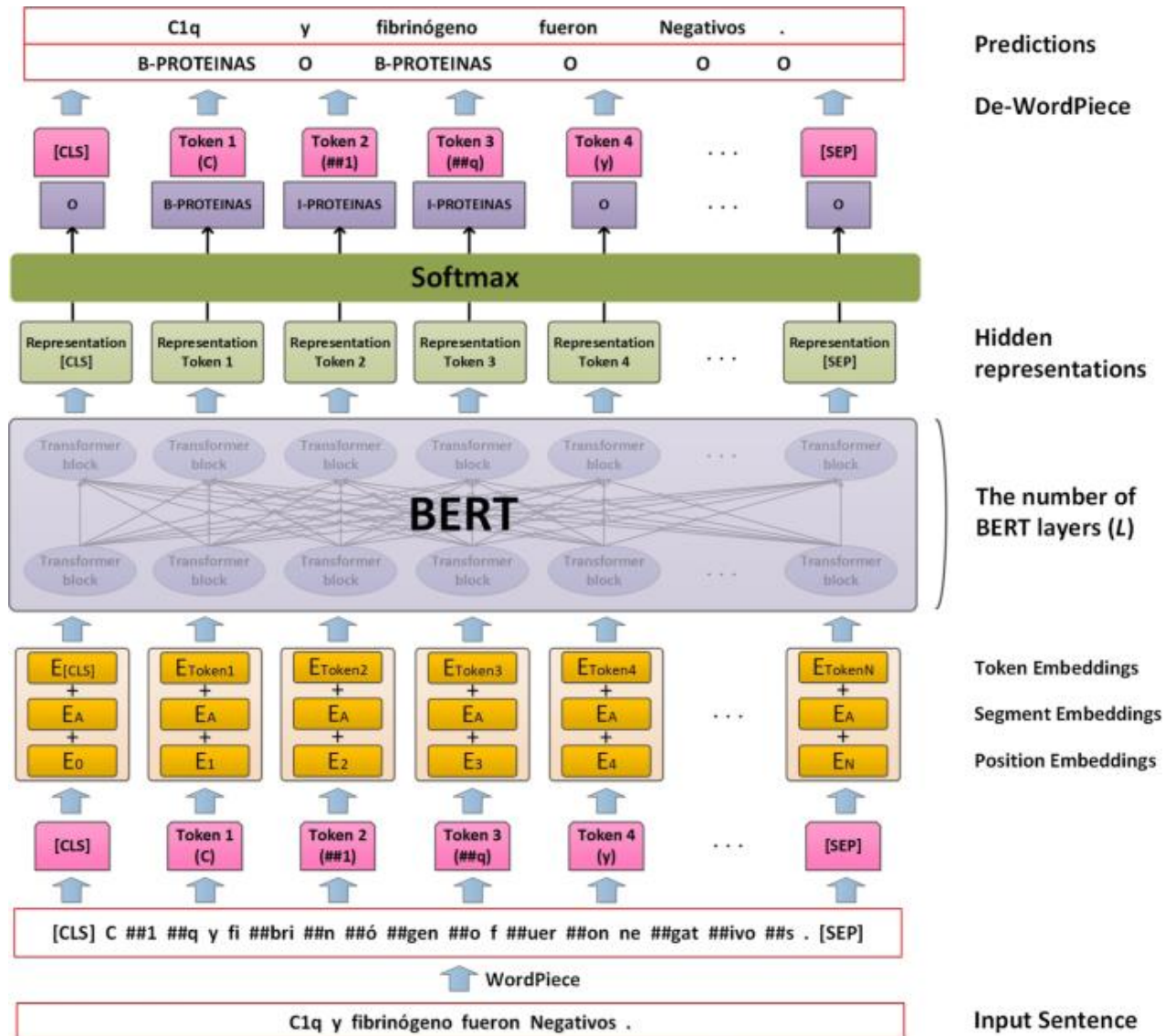
Training set statistics



<https://aclanthology.org/2022.findings-acl.116/>  
<https://github.com/vistec-AI/Thai-NNER>

Figure 3: The distribution of classes sorted by frequency shows that rarer classes consist of more than 20% of all instances.

# BERT and PoS



# Additional reading

- <https://web.stanford.edu/~jurafsky/slp3/>
  - Chap 17 (PoS)
  - HMM (Appendix A)
- Huggingface's token classification tutorial  
[https://huggingface.co/docs/transformers/en/tasks/token\\_classification](https://huggingface.co/docs/transformers/en/tasks/token_classification)

# Conclusion

- What is token classification?
- Traditional methods
  - Sequence methods
    - HMM
    - CRF
      - Viterbi and beamsearch
- Neural network methods