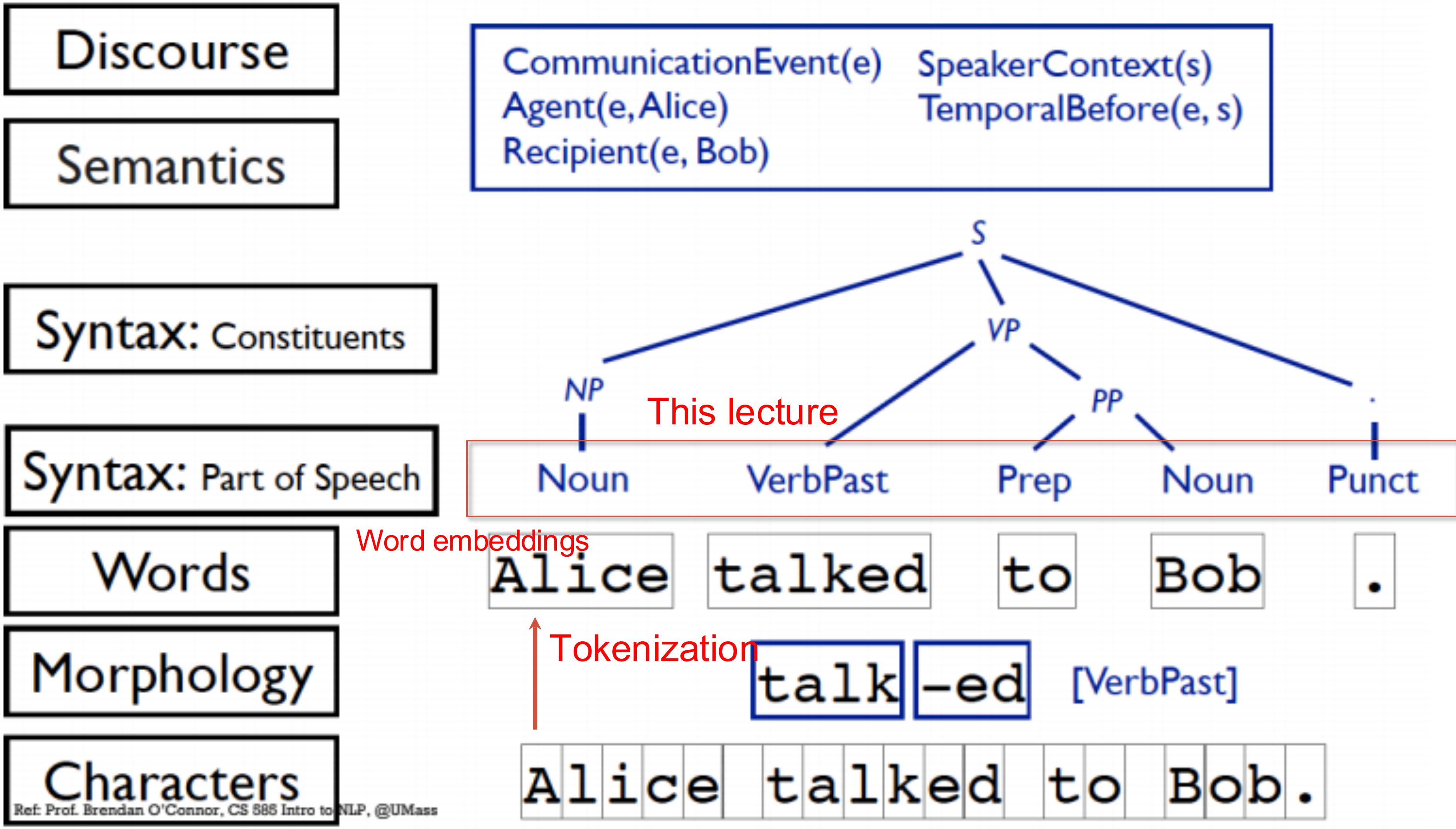
Token Classification

PoS and NER

HMM, CRF, and search

Token Classification

- A broad term for classifying tokens: PoS, NER



Ref: Prof. Brendan O'Connor, CS 585 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 *there*
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	*to*

https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html

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*

Example from MIT 6.864

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 คุกกี้ เสี่ยงทาย
    - [ลุงพล]/person ร้อง เพลง [คุกกี้เสี่ยงทาย]/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, โคโรน่า, โค้ก, 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	นี่, นั่น, 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

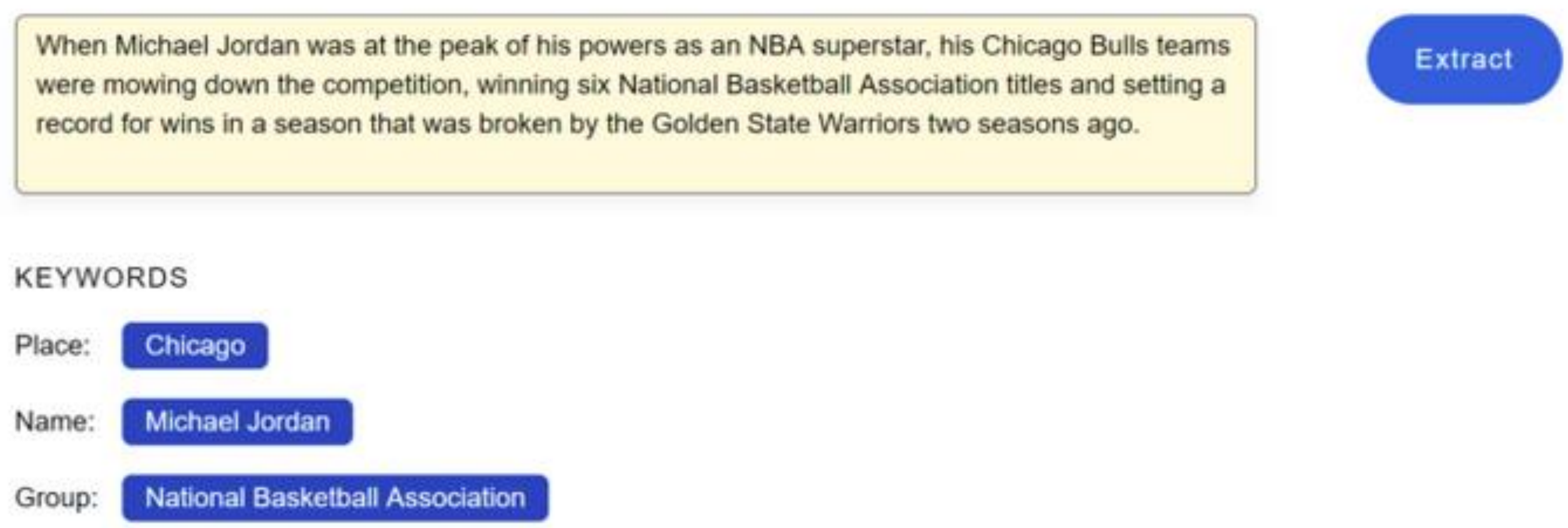
- Name-Entity Recognition wants to extract all name entities in a sentence and classify into types
- Can tag groups of words into the same category
  - ลุง พล ร้อง เพลง คุกกี้ เสี่ยงทาย
  - [ลุงพล]/person ร้อง เพลง [คุกกี้เสี่ยงทาย]/title

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



https://blog.paralleldots.com/product/applications-named-entity-recognition-api/

Labeling standard (IOB format)

- Consider the following sentence and NE tags

ลุง พล ป๋า เบิร์ด ร้อง เพลง ชาติ ไทย

Name	Name	Name	Name	Other	Song	Song	Song

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

NameB	NameI	NameB	NameI	Other	SongB	SongI	SongI

ลุง พล ป๋า เบิร์ด ร้อง เพลง ชาติ ไทย

Thai Nested NER

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

https://medium.com/airesearch-in-th/thai-n-ner-thai-nested-named-entity-recognition-1969f8fe91f0

Thai Nested NER

	Models	Head P	Head R	Head F1	Body P	Body R	Body F1	Tail P	Tail R	Tail F1	All P	All R	All 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
Baseline	WangchanBERTa–sp	**90.70**	77.66	83.67	**81.55**	55.90	**66.33**	78.02	26.09	39.10	**89.04**	70.89	78.94
Baseline	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
Baseline	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
Baseline	XLM-R–sh	89.45	79.72	84.31	77.29	**58.06**	66.31	71.80	**39.73**	**51.16**	86.93	**73.66**	**79.75**
SOTA	Second-best-learning	87.57	**81.78**	**84.58**	80.12	54.85	65.12	**79.05**	19.41	31.16	86.41	73.49	79.43
SOTA	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
SOTA	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

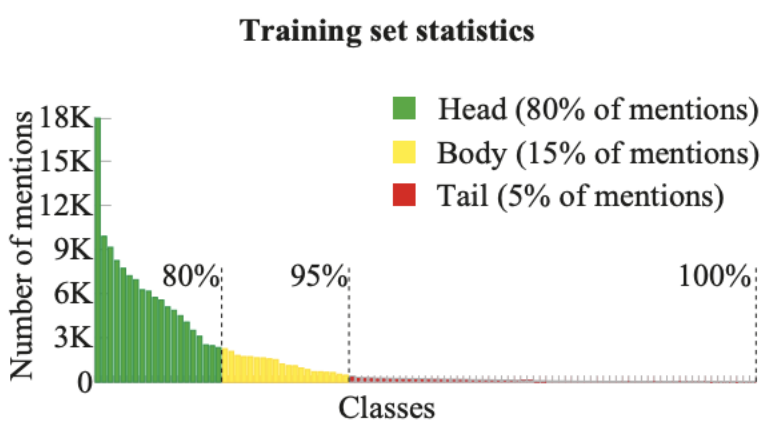


Figure 3: The distribution of classes sorted by frequency shows that rarer classes consist of more than 20% of all instances.

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

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
  - $W := \{w_1, w_2, w_3, \ldots, w_n\}$
- Sequence of tags
  - $T := \{t_1, t_2, t_3, \ldots t_n\}$

P(a,b) joint distribution
P(a|b) conditional distribution
P(a) marginal distribution
$P(a) = \Sigma_b P(a,b)$

- Given W predict T
  - $\boxed{\text{argmax}_T P(T|W)}$ Discriminative Model
    - Or
  - $\text{argmax}_T \frac{P(T,W)}{P(W)} = \boxed{\text{argmax}_T P(T,W)}$ Generative Model

P(W) is constant does not affect the argmax

Modeling P(T,W)

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

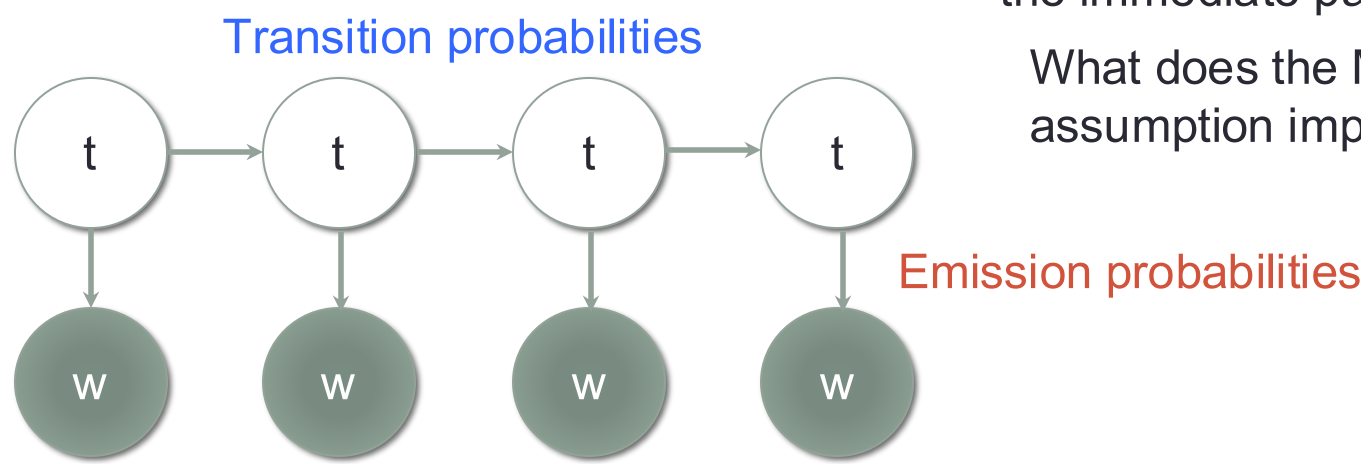
Modeling P(T,W)

- $P(T,W) = P(w_1,w_2,w_3,\ldots,w_n,t_1,t_2,t_3,\ldots t_n)$
    - 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: $t \in$ {Noun, Adv, Adj, Verb}

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

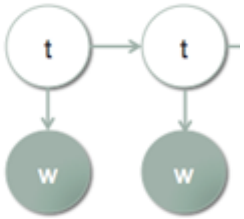
Initial state prob Transition probabilities Emission probabilities

Hidden Markov Model

N = Noun
NN = Not Noun
i,j – state (tag) index
k – emission (word) index

- Defining HMM requires
- 1. Starting state probability p_0 = [0.7 0.3]
- 2. Transition probability, A_{ij}

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



- 3. Emission probably, 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 eat chinese] , [N NN N]) ?
= p_0(N) * A12 * A21 * B11 * B22 * B13
= 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!

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

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

Hidden Markov Model

- How to estimate A and B?
- Counts!

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

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

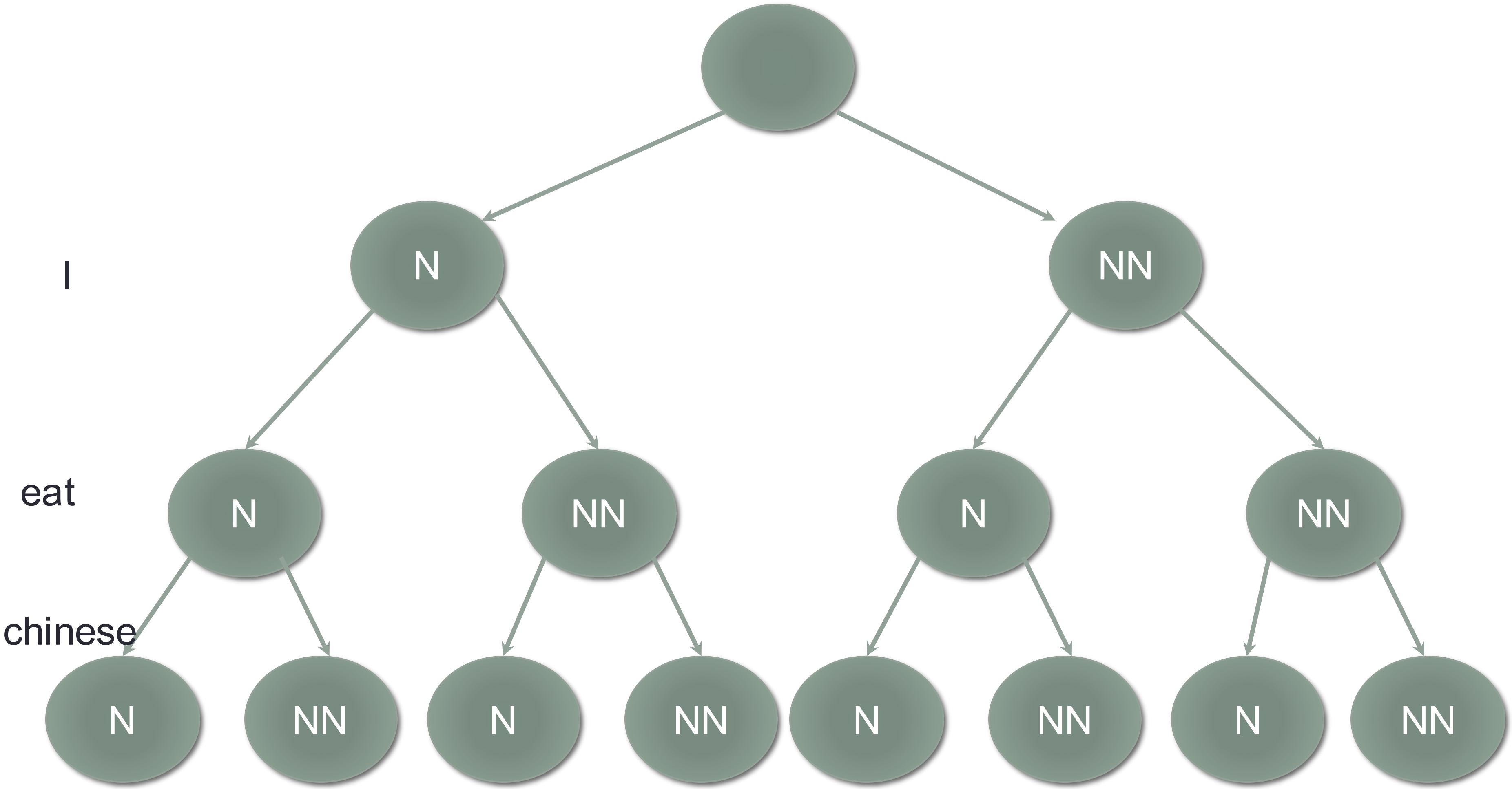
B_{ik}	I	eat	Chinese
State N	0.8	0.01	0.19
State NN	0.1	0.45	0.45

Decoding

- Recall we want to find the sequence of tags that maximizes the joint probability
  - 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

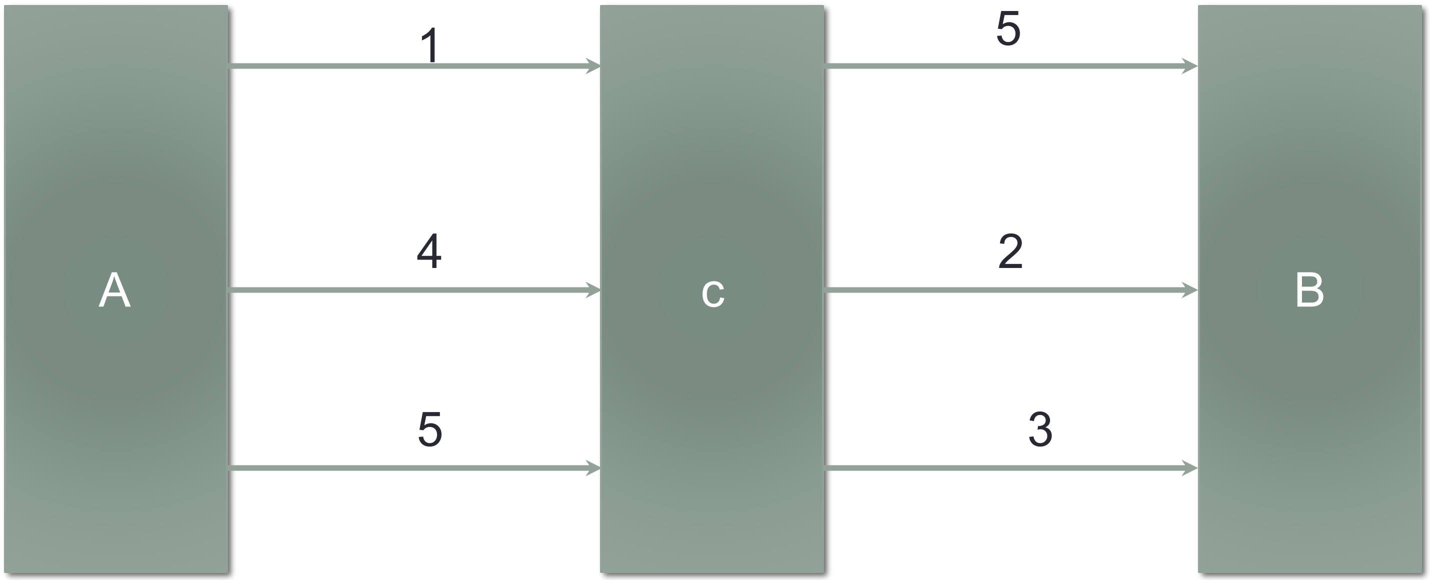


The Viterbi Algorithm (Dynamic programming)

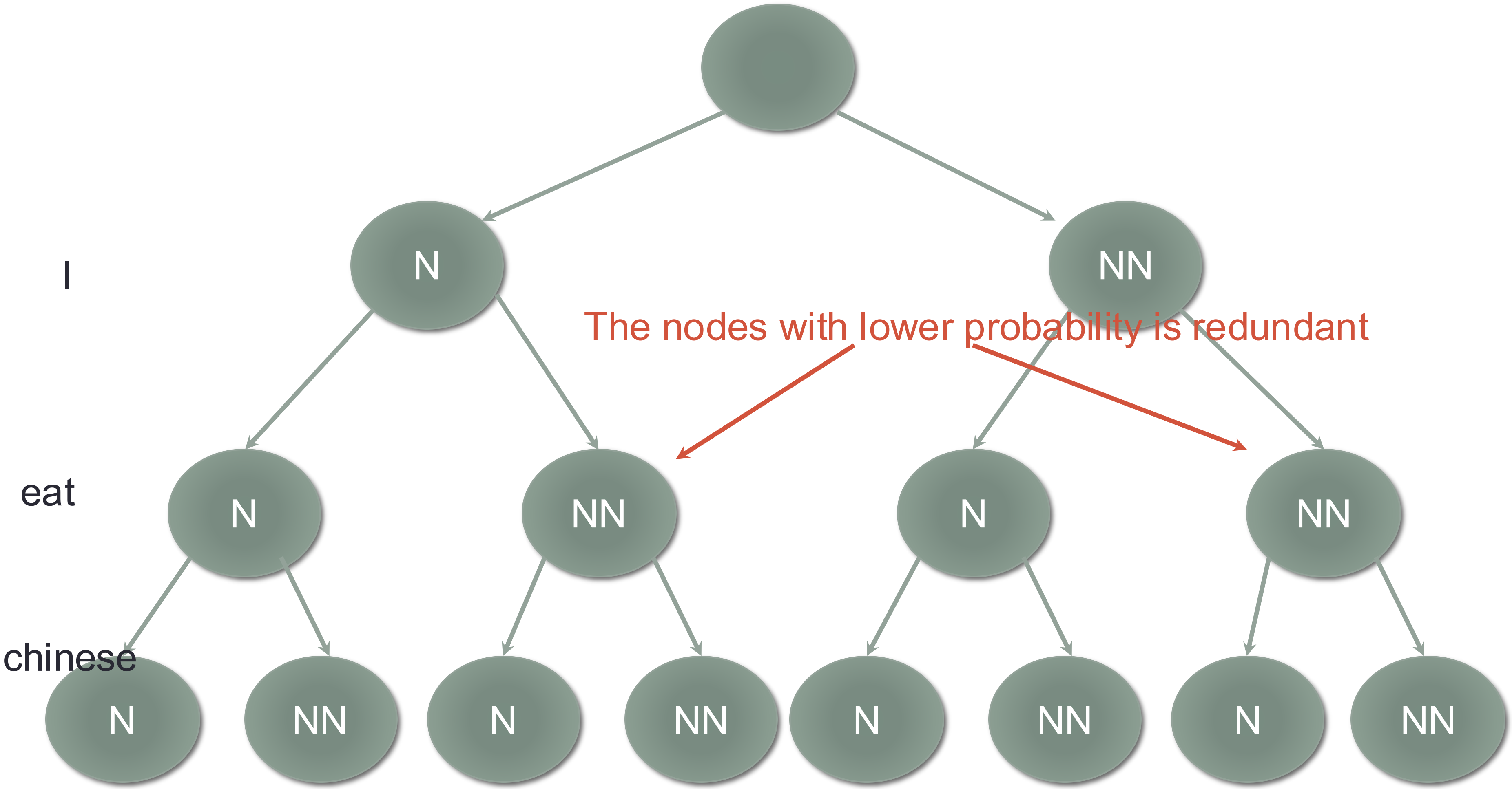
- Some computation are redundant
  - We can save computation from previous steps

Dynamic programing

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



Redundancy



I
eat
chinese
N
NN
N
NN
N
NN
N
NN
N
NN
N
NN
N
NN
The nodes with lower probability is redundant

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:

$$\pi[0, <S>] = \log 1 = 0$$
$$\pi[0, t] = \log 0 = -\infty,\ if\ t \neq <S>$$

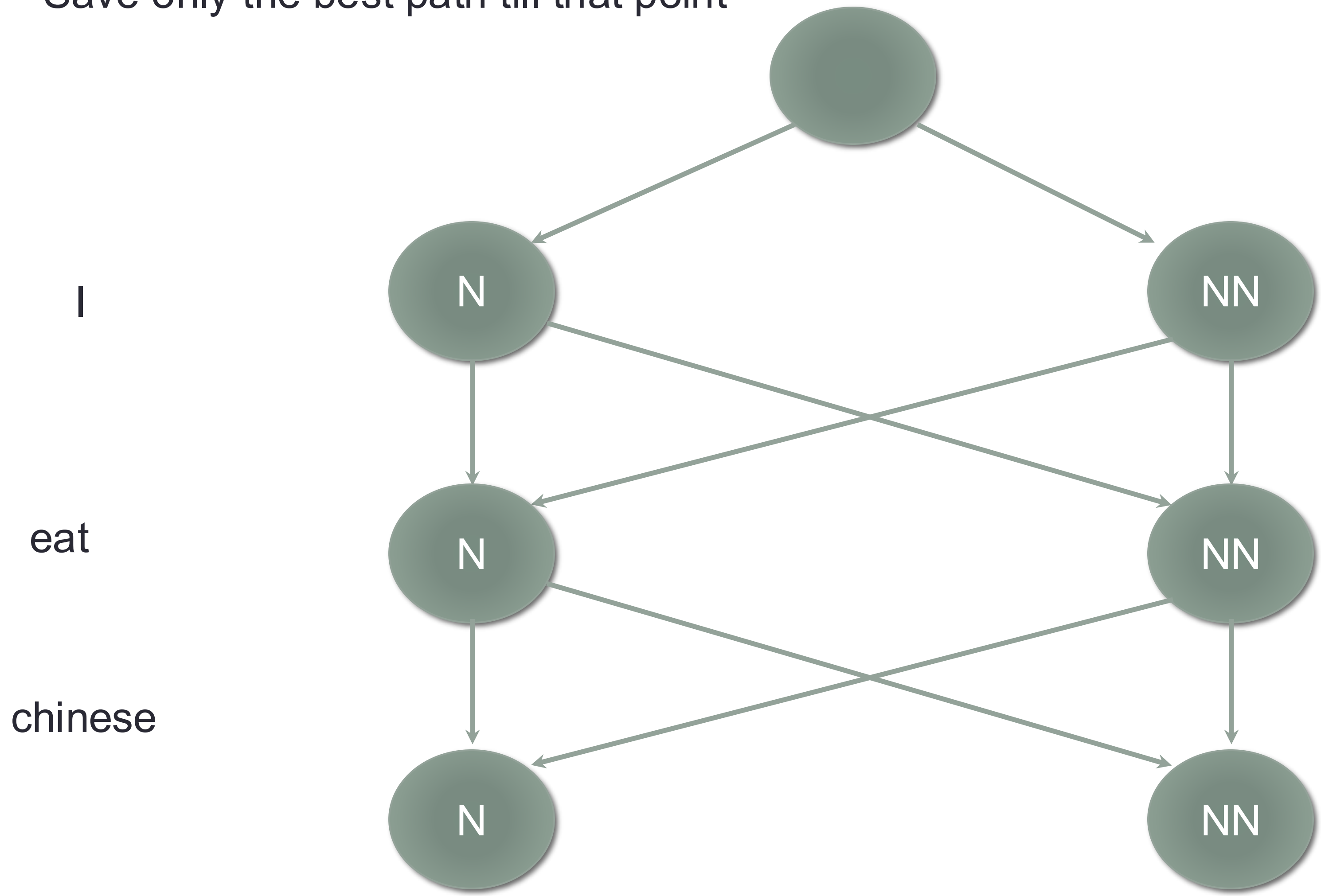
where $< S >$ is the start symbol.

Recursive Step:

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

Viterbi

Save only the best path till that point



Decoding example

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

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

π[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:

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

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

π[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:

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

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

π[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:

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

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

π[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)

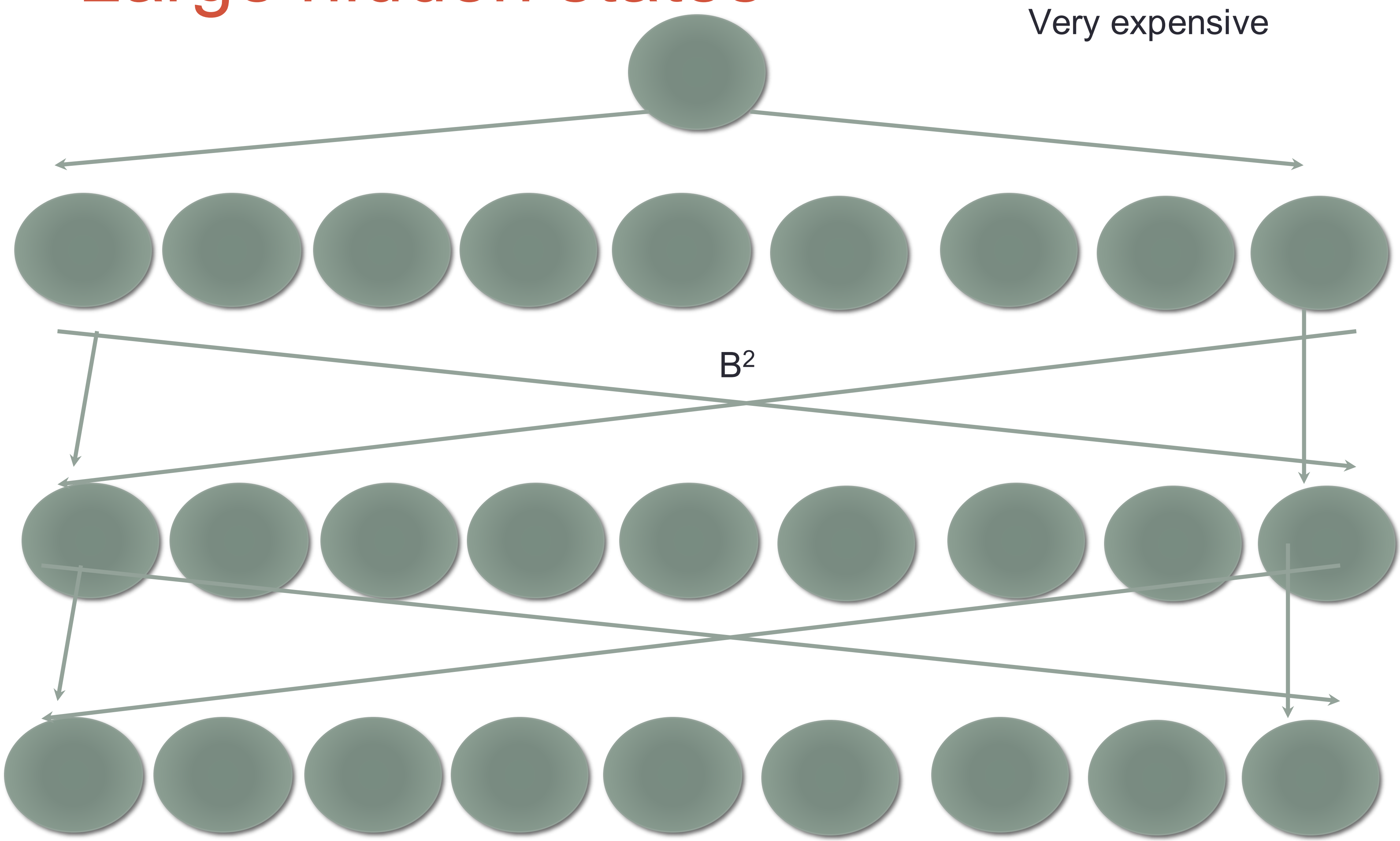
- We can find the best T by
- Find that best probability at the end
- Backtrack according to B[i,t]

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

- 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?

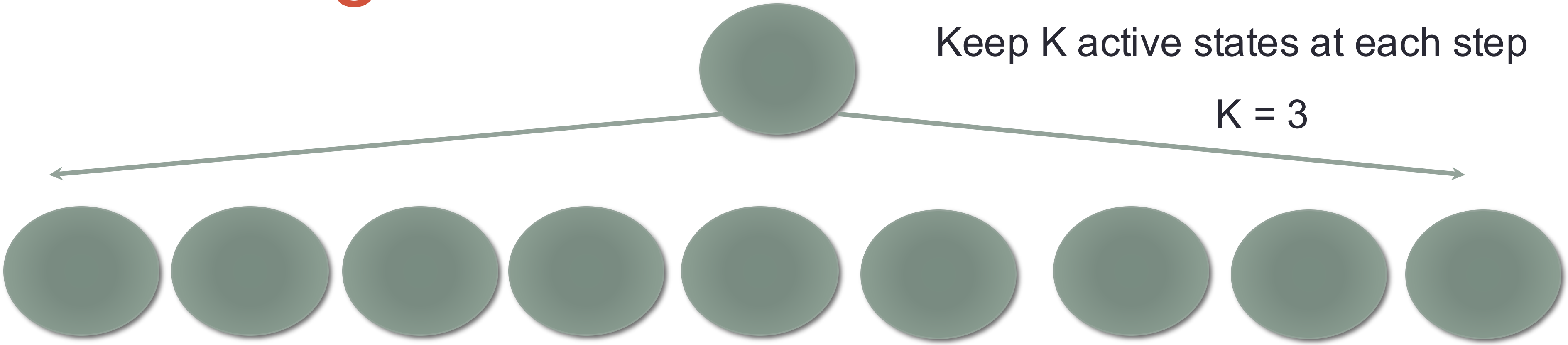
Large hidden states



Pruning with Beamsearch

Keep K active states at each step

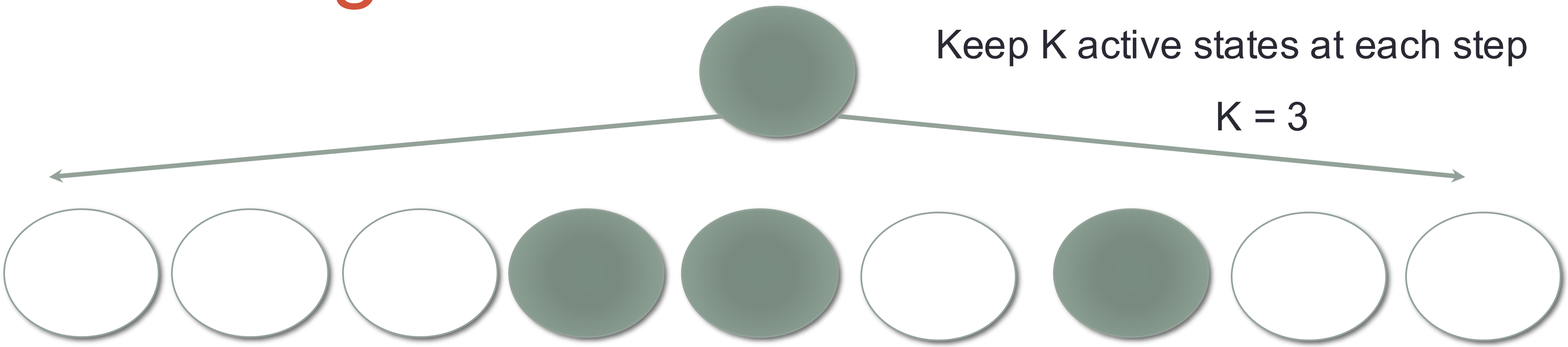
K = 3



Pruning with Beamsearch

Keep K active states at each step

K = 3

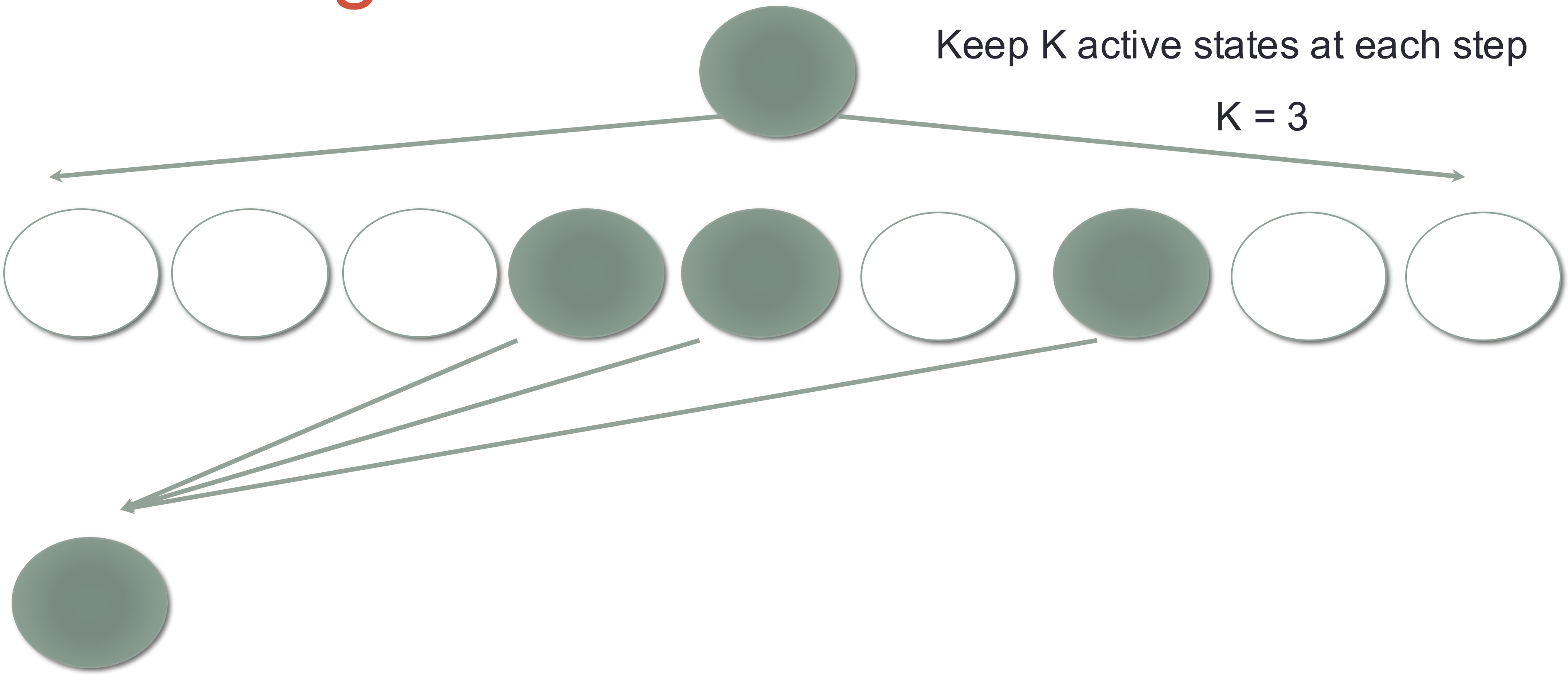


Prune, keep K best nodes

Pruning with Beamsearch

Keep K active states at each step

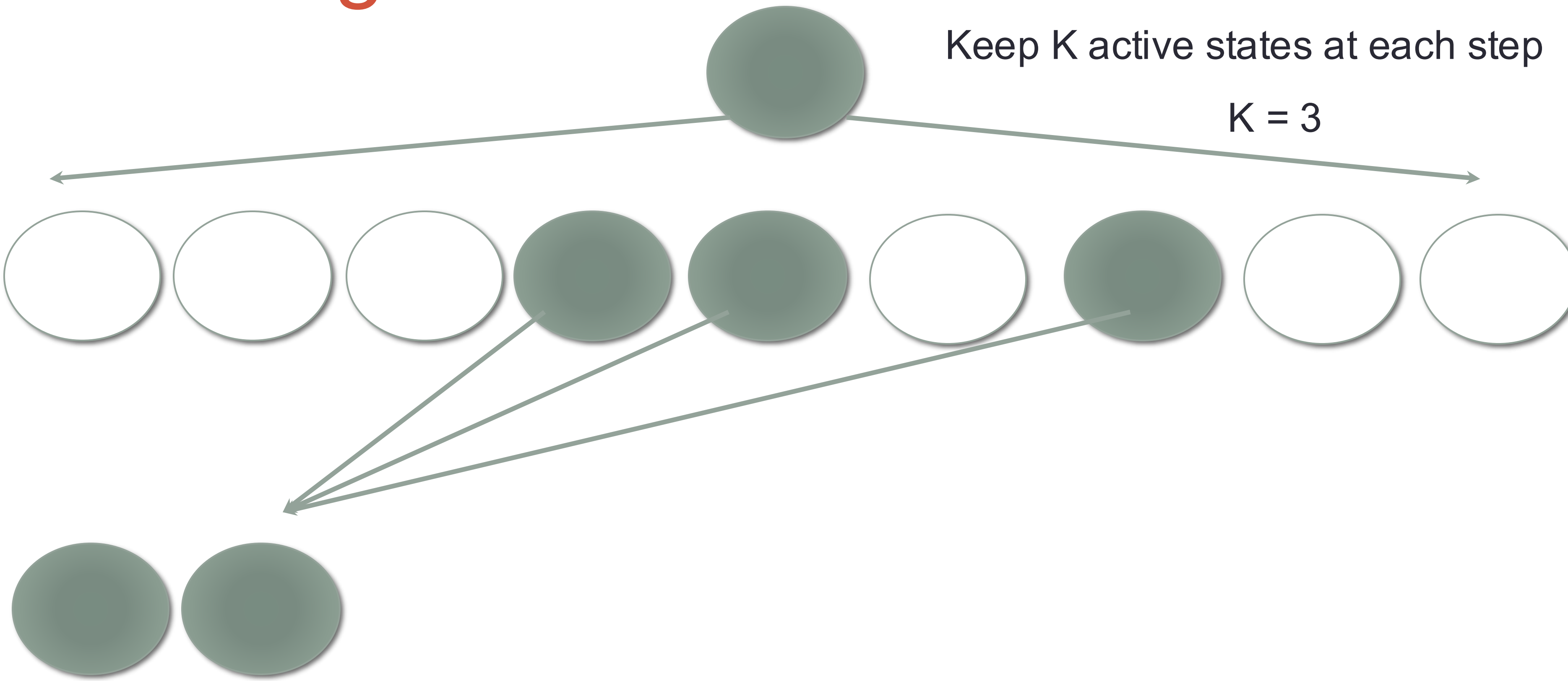
K = 3



Pruning with Beamsearch

Keep K active states at each step

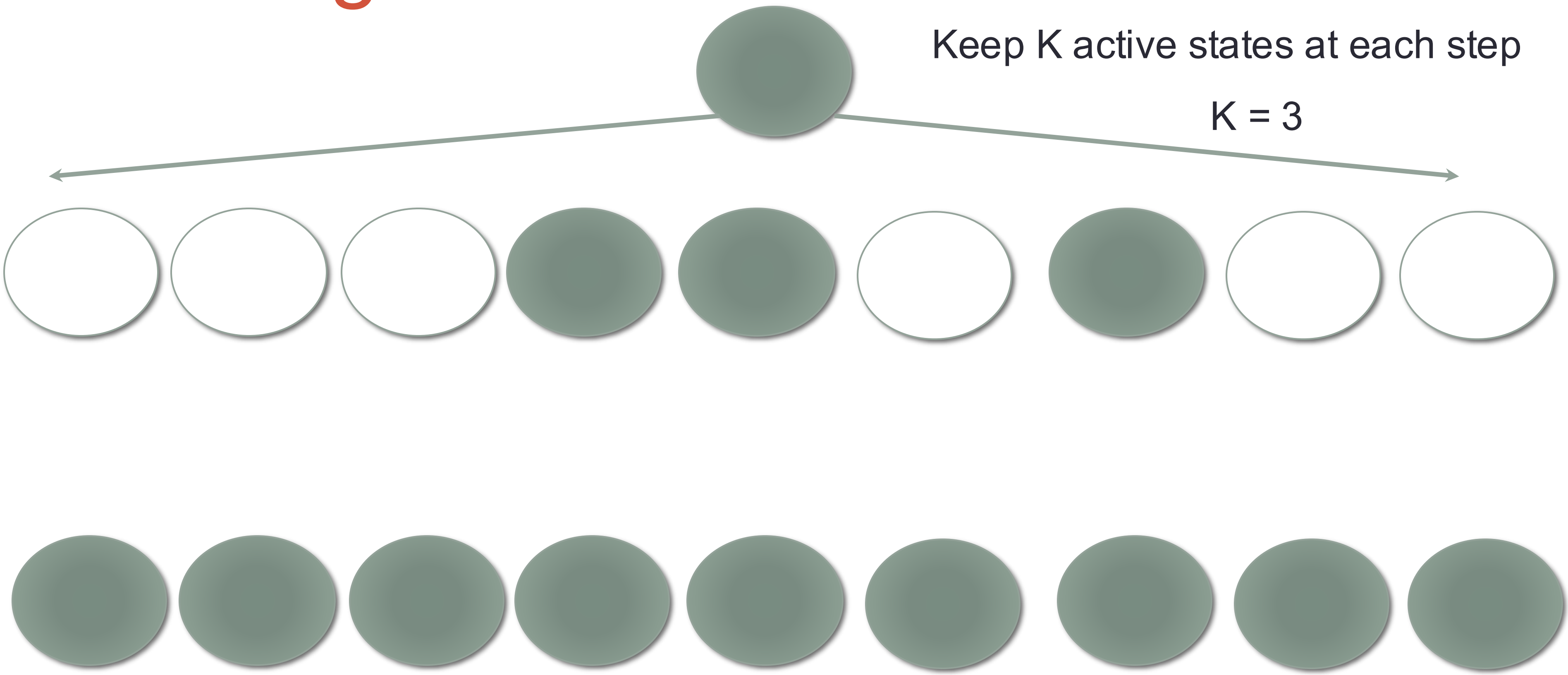
K = 3



Pruning with Beamsearch

Keep K active states at each step

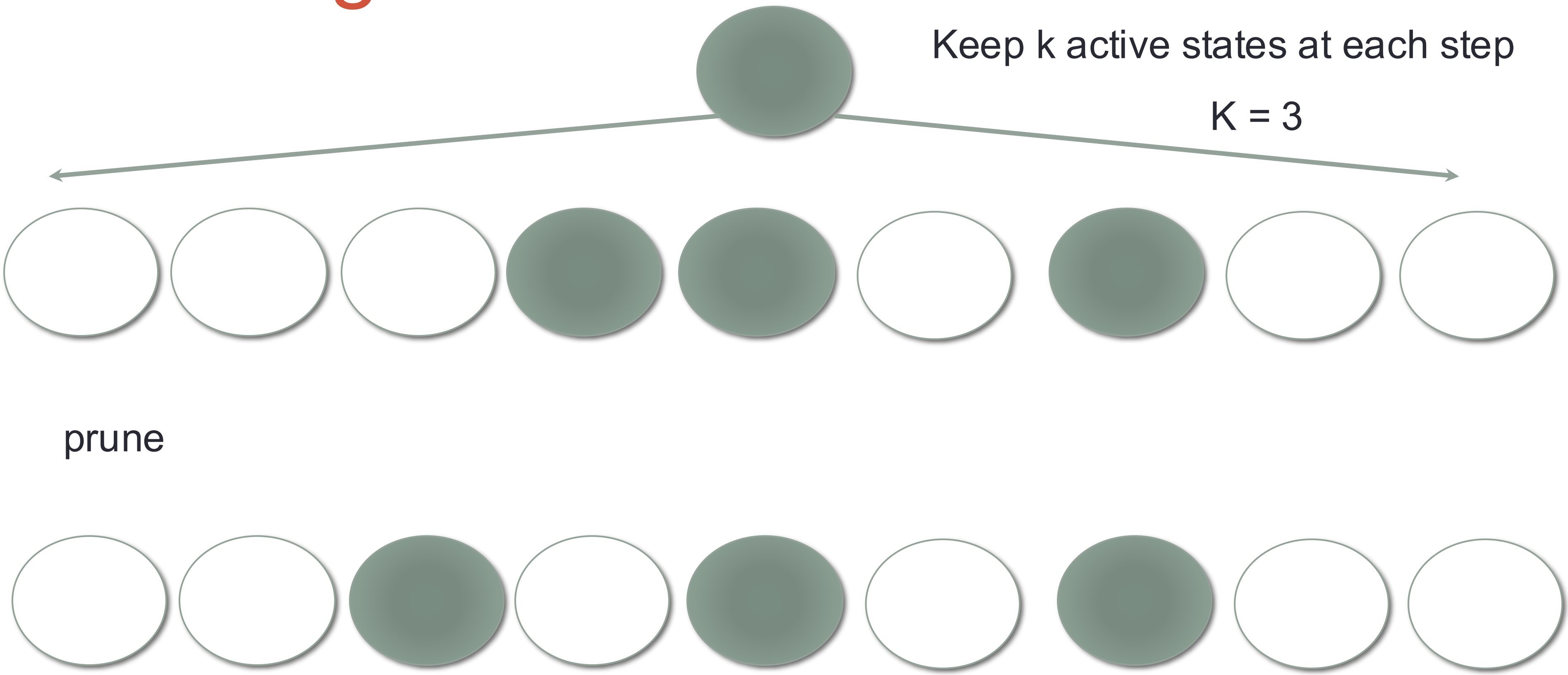
K = 3



Pruning with Beamsearch

Keep k active states at each step

K = 3

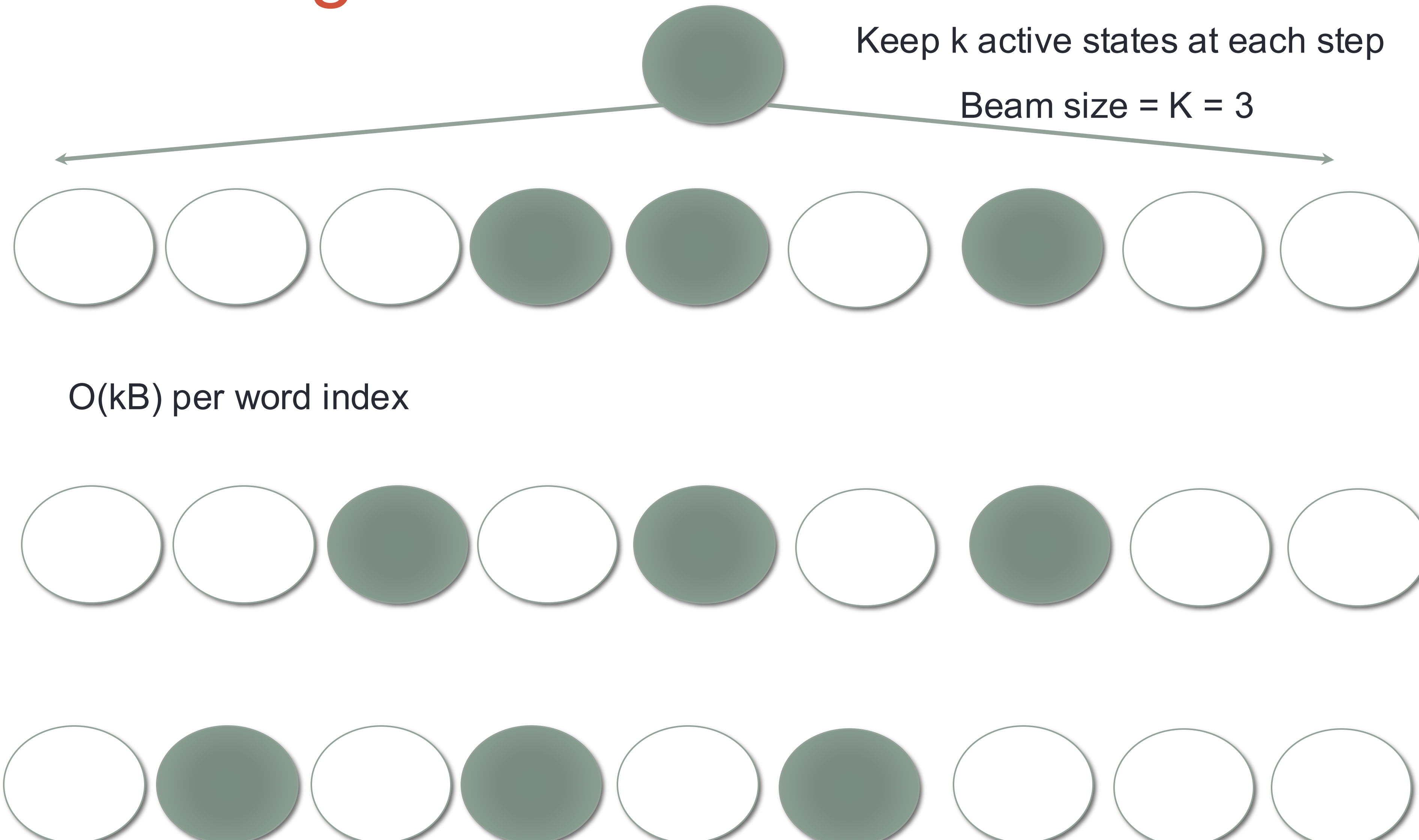


prune

Pruning with Beamsearch

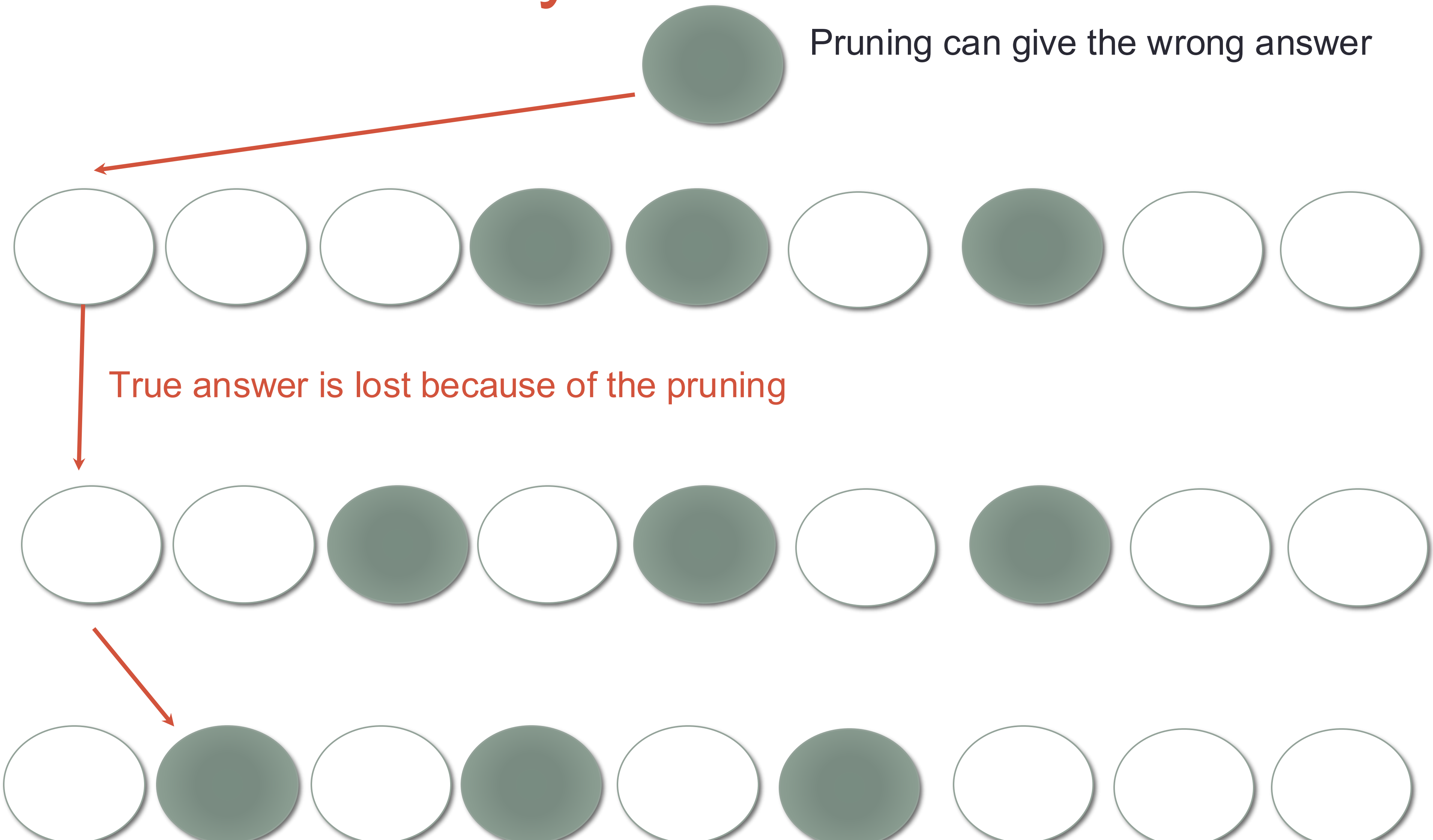
Keep k active states at each step

Beam size = K = 3



O(kB) per word index

Inadmissibility

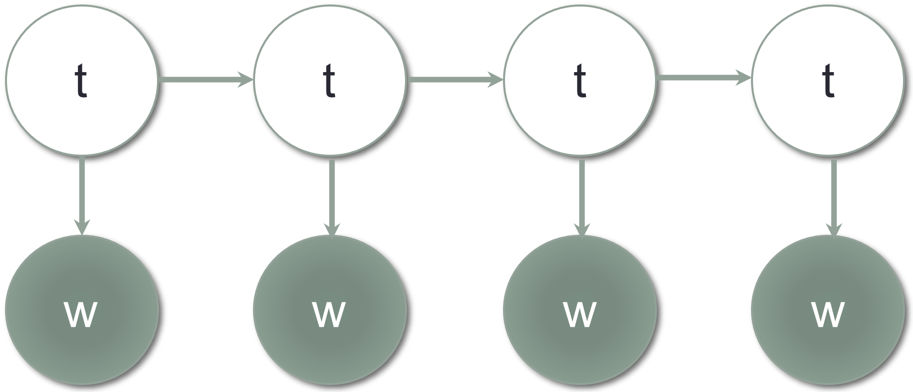


Beam search

- 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.

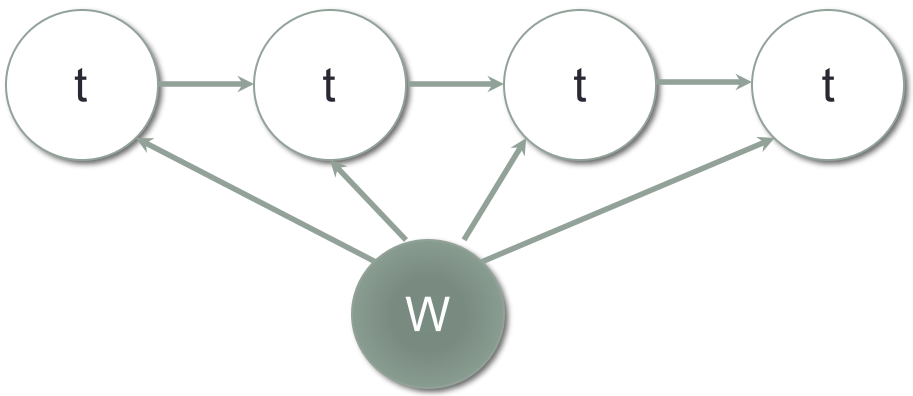
HMM assumptions and disadvantages



- 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

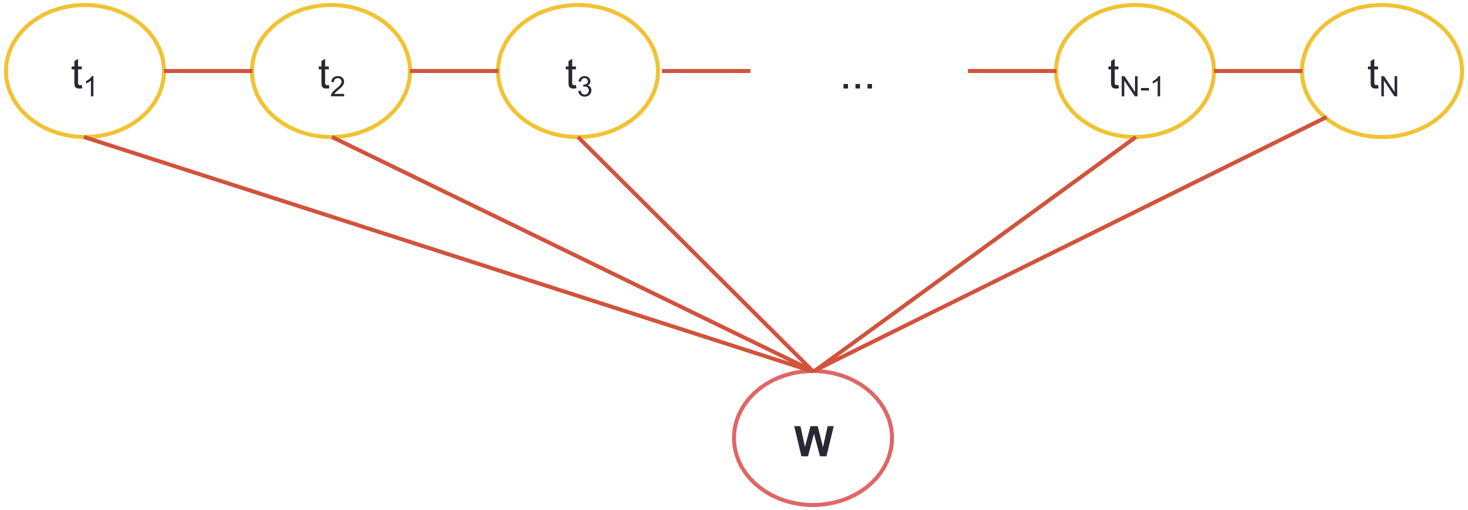


$$P(T|W) = \pi\, P(t_n \mid t_{n-1}, W) = P(t_1 \mid W)\, P(t_2 \mid t_1\, W)\, P(t_3 \mid t_2\, W)\, P(t_4 \mid t_3\, W)$$

- 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 **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

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

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 $f(t_n, t_{n-1}, \mathbf{W}, n) \to \mathbb{R}$
    - This is from our independence assumption.
  - We call these functions, feature functions

Feature function

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

A feature function in linear-CRF:

returns a real value

$$f(t_n, t_{n-1}, \mathbf{W}, n) \to \mathbb{R}$$

current label

current index

previous label

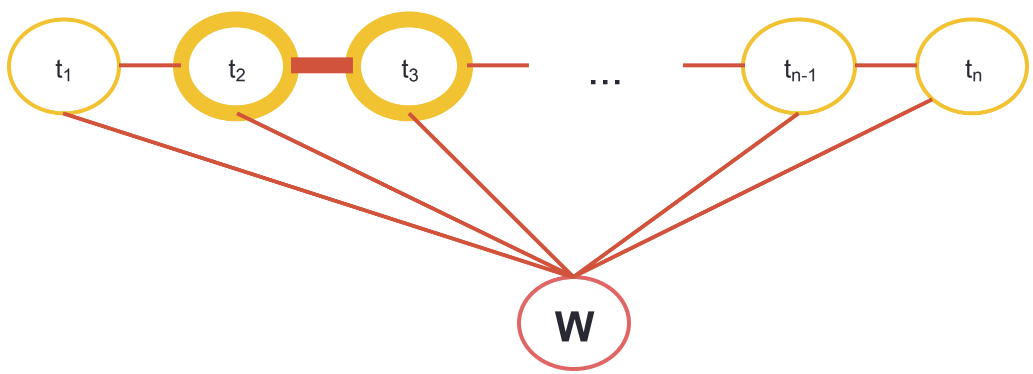
input sequence

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

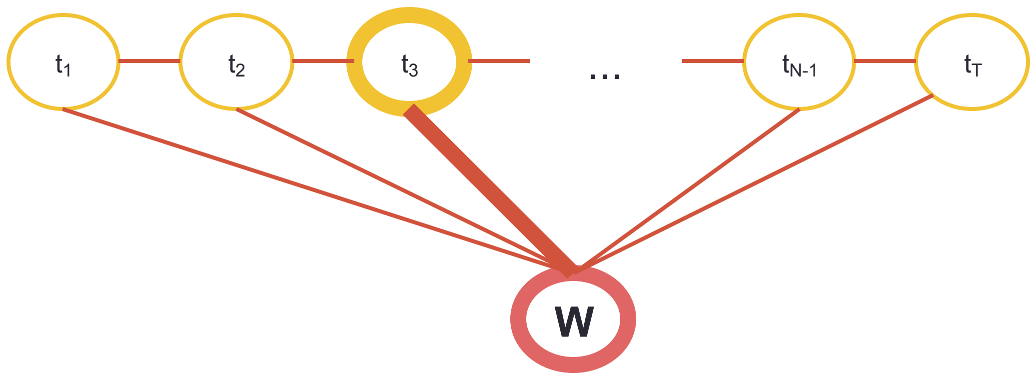
n = 3



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

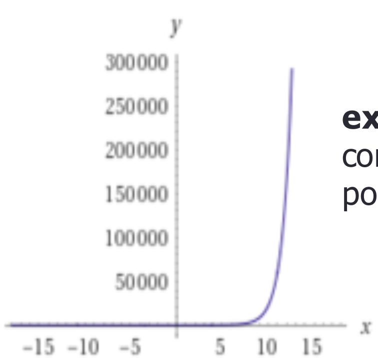
At each time step, a potential $\Psi_n(t_n, t_{n-1}, \mathbf{W}) \rightarrow \mathbb{R}^+$ is a function that takes all feature functions into account, by summing their products with the associated weight

number of all feature functions

factors at time n

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

k-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	...
t*	NOUN	VERB	NOUN	VERB	...
w	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:

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

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

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

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

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

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

$$\Psi_3(t_3, t_2, \mathbf{W}) = exp[\Sigma_1^5 \theta_k f_k(t_3, t_2, \mathbf{W})]$$

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

$$= 3.00$$

Probability of the whole sequence

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

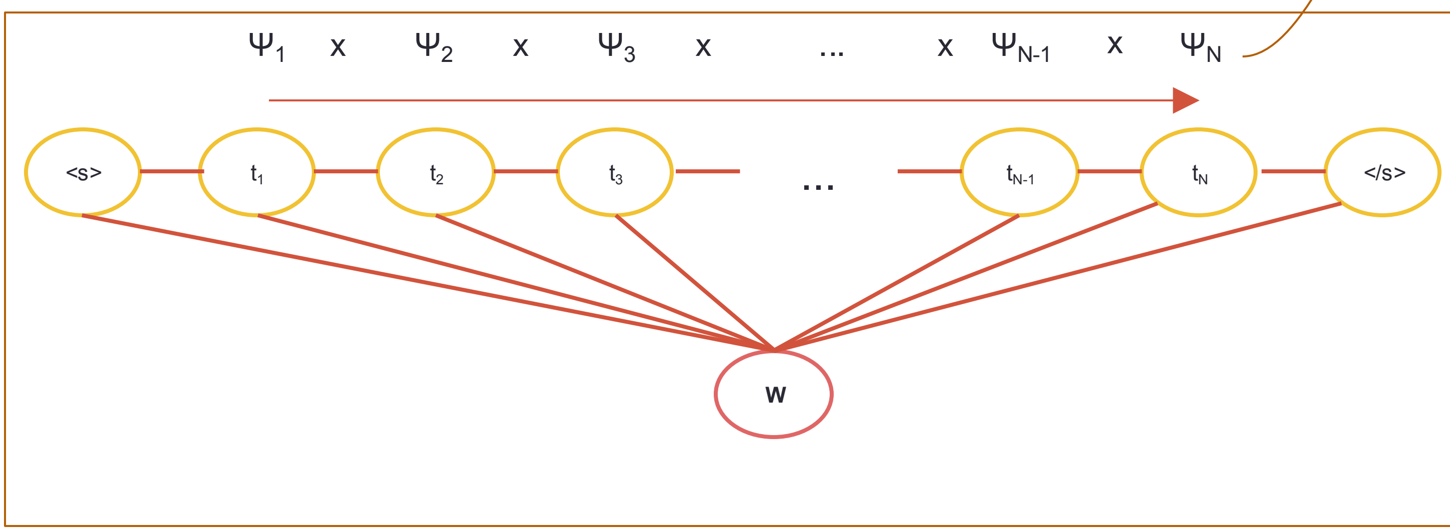
With p(**T**, **W**) we can compare and pick the best **T**

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

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 T for a given sequence input W

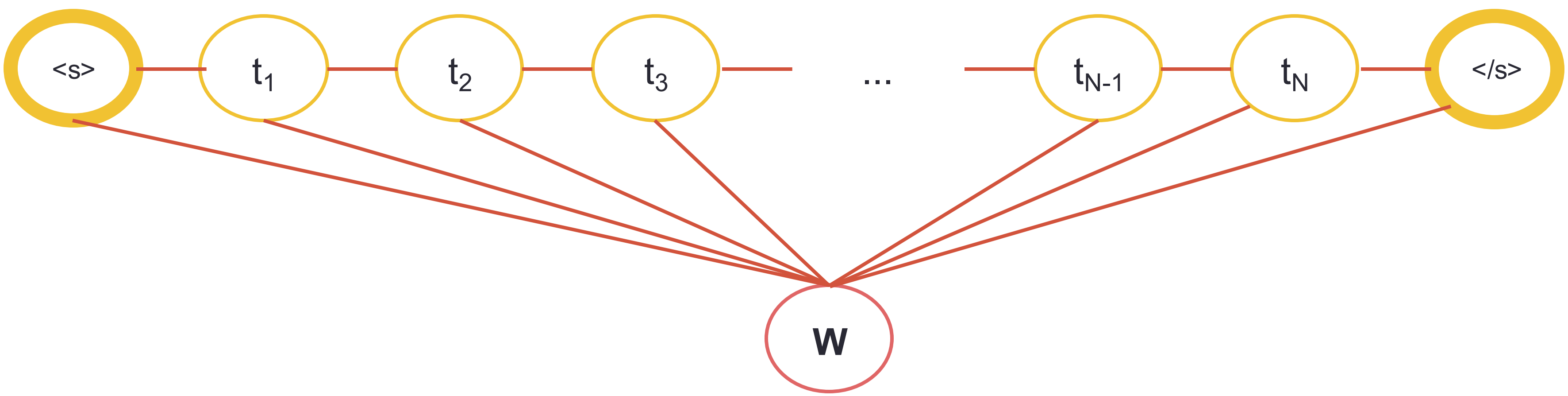


Special states and characters

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

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

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

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

Computing **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:

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

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

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

$$Z(W) = \sum_{T}\prod_{n=1}^{N} exp[\sum_{k}^{K} \theta_k f_k(t_n, t_{n-1}, \mathbf{W}, n)]$$

Linear-chain CRF

A linear-chain CRF is a conditional distribution

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

, where Z(**x**) is an instance-specific normalization function

$$Z(W) = \sum_{T} \prod_{n=1}^{N} exp[\sum_{k}^{K} \theta_k f_k(t_n, t_{n-1}, \mathbf{W}, n)]$$

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

Linear-chain CRF

$$P(T|W) = \underbrace{\frac{1}{Z(W)}}_{\text{normalization function}} \prod_{n=1}^{N} \underbrace{exp[\sum_{k}^{K} \theta_k f_k(t_n, t_{n-1}, \mathbf{W}, n)]}_{\text{sum of weighted feature functions}}$$

normalization function
the sum of products of all possible output sequences
not the same as **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,**W**)
- How to model P(t_{t-1},t_t,**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(**T**,**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 **W**

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

- Find **T**
    - Not so straight forward, many possible **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

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

Goes over time step just like HMM viterbi

Find the label sequence **T** that maximize this value

Viterbi: Structure

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

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

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

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

$$\mathrm{VTB(ADJ,1)} = \underbrace{\Psi_1(\mathrm{ADJ}, <s>, \mathbf{x})}_{\text{Factors at time t=1, for current label=ADJ, previous label=<s>}} \underbrace{\mathrm{VTB}(<s>,0)}_{1}$$

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

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

Iterate from n=2 to n=N+1

VTB

	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

Var

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

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

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

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

Viterbi: Finalize

Backtrack from 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 $\Psi_n(t_n, t_{n-1}, \mathbf{W}) \to \mathbb{R}^+$
  - 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 $\Psi_n(t_n, t_{n-1}, \mathbf{W}) \to \mathbb{R}^+$
  - 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.

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

parameters

Training objective

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

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

To clarify, parameters θ 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^{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

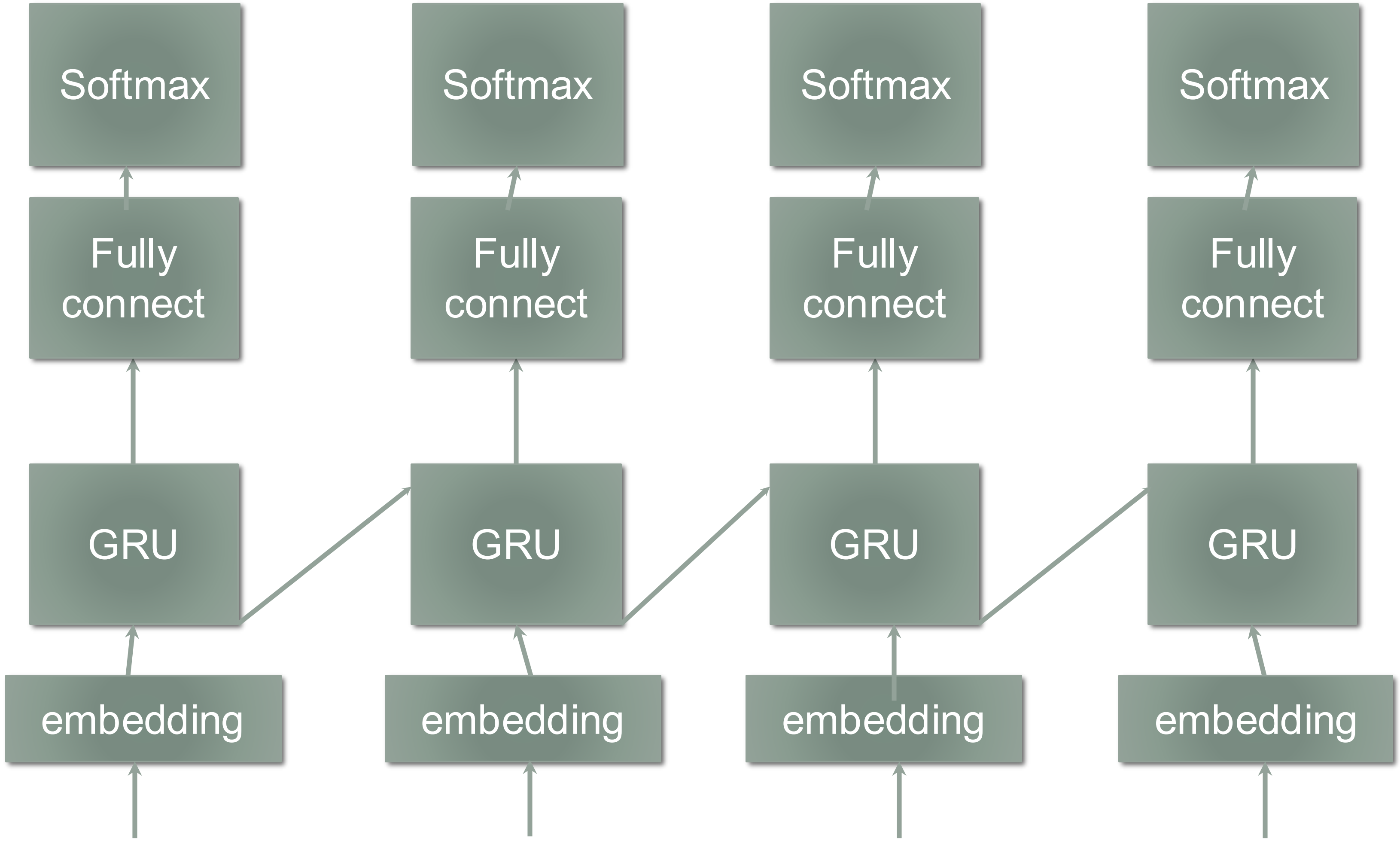
- An implementation of CRFs for labeling sequential data in C++
  SWIG API is provided to be an interface for various languages
  http://www.chokkan.org/software/crfsuite/
- python-crfsuite: Python binding for crfsuite
  https://github.com/scrapinghub/python-crfsuite
- An example use of python-crfsuite 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
- CRFlayer: $P(y|y_{t-1},\mathbf{x})$ not just $P(y_t|\mathbf{x})$

Typical softmax only considers current word label not the sequence

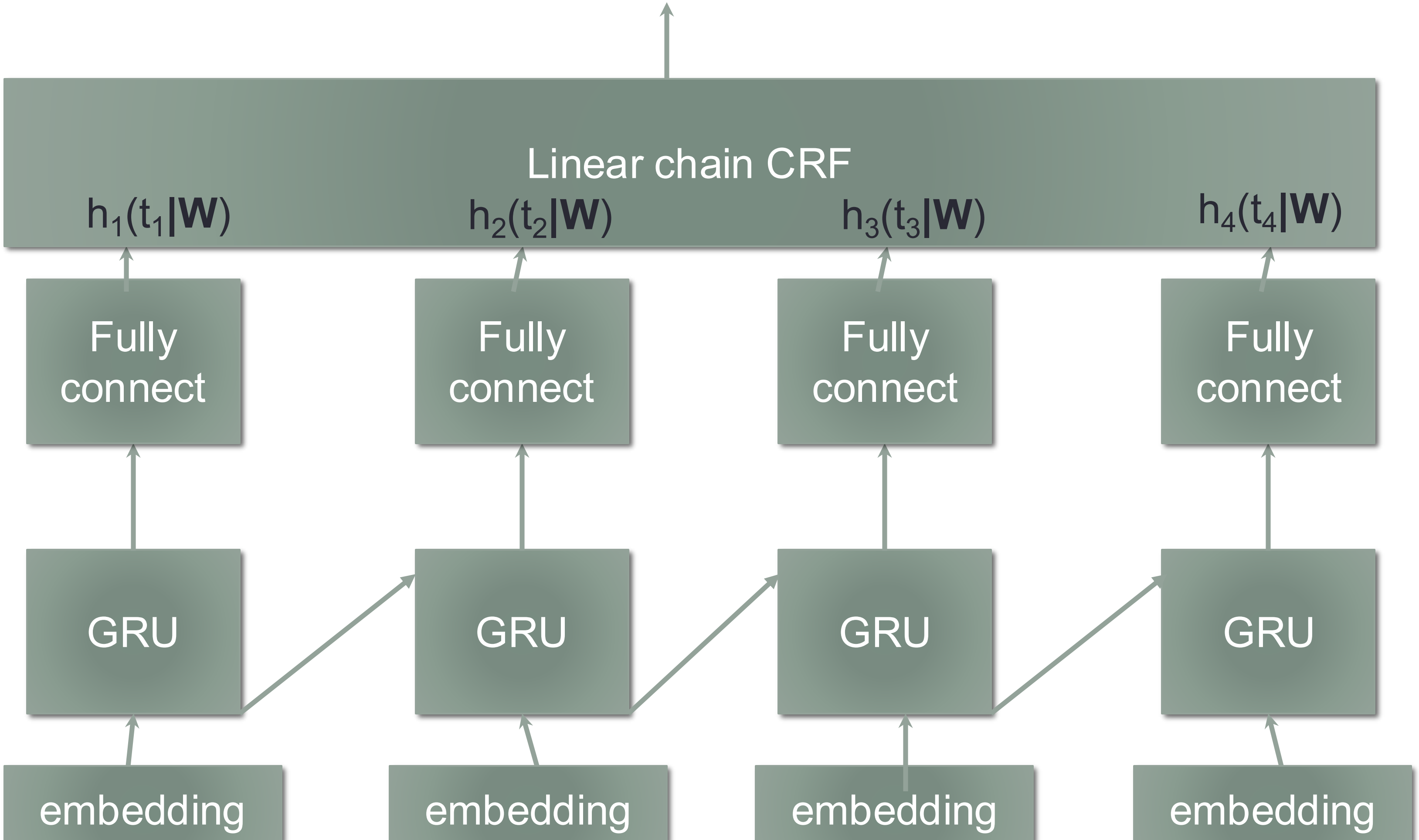
Neural network for POS



Neural network for POS with CRF output

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

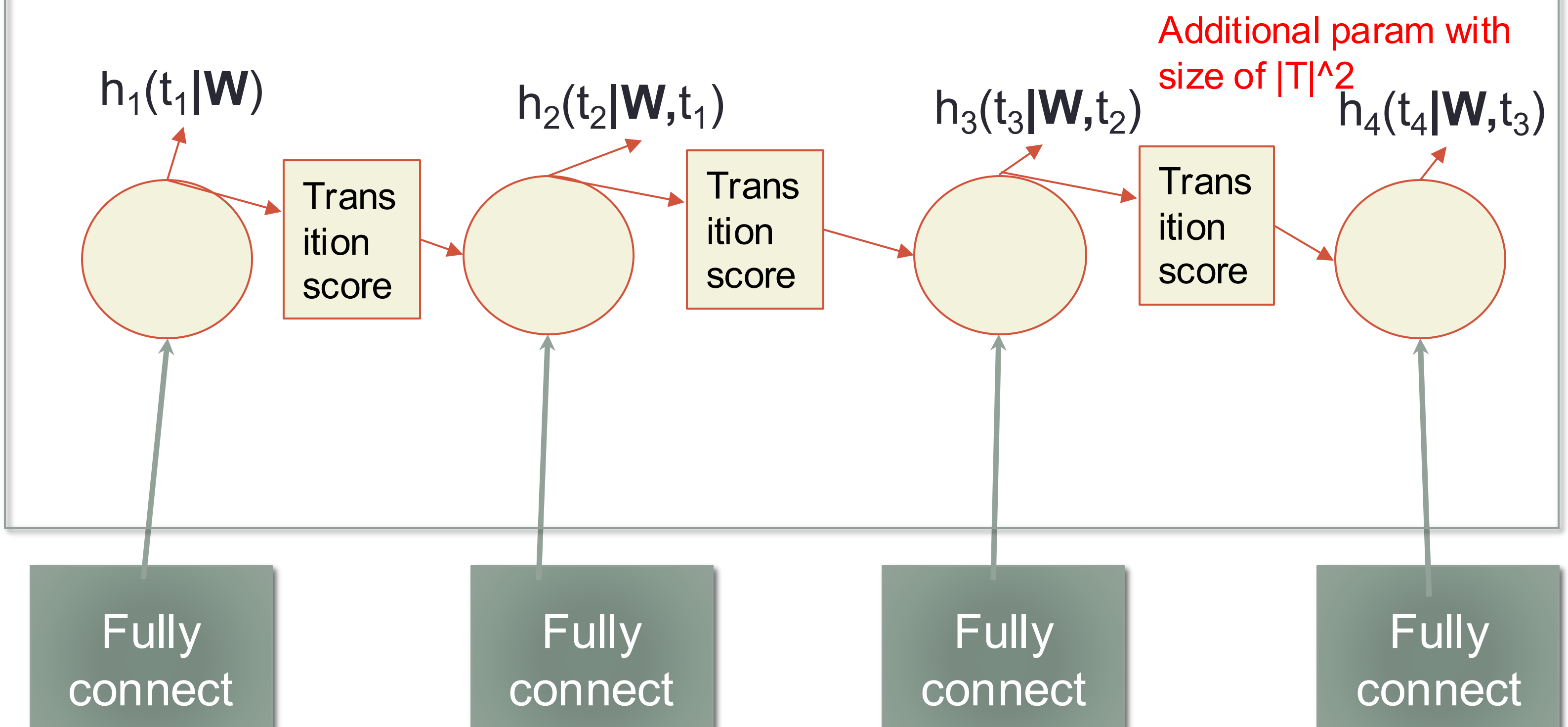
maximize likelihood of sequence of tags instead



Neural network for POS with CRF output

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

a slight problem in gradient computation
needs to use forward-backward algorithm to compute the gradient
in this layer for efficient computation see Appendix A in the book



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(**T***|**W**)) where **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

Performance

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

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)‡	89.59
Huang et al. (2015)‡	90.10
Chiu and Nichols (2015)‡	90.77
Ratinov and Roth (2009)	90.80
Lin and Wu (2009)	90.90
Passos et al. (2014)	90.90
Lample et al. (2016)‡	90.94
Luo et al. (2015)	91.20
This paper	**91.21**

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.

End-to-end Sequence Labeling via Bi-directional LSTM-CNNs-CRF
https://arxiv.org/pdf/1603.01354.pdf

Thai Nested NER

	Models	Head P	Head R	Head F1	Body P	Body R	Body F1	Tail P	Tail R	Tail F1	All P	All R	All 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	**90.70**	77.66	83.67	**81.55**	55.90	**66.33**	78.02	26.09	39.10	**89.04**	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	**58.06**	66.31	71.80	**39.73**	**51.16**	86.93	**73.66**	**79.75**
SOTA	Second-best-learning	87.57	**81.78**	**84.58**	80.12	54.85	65.12	**79.05**	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

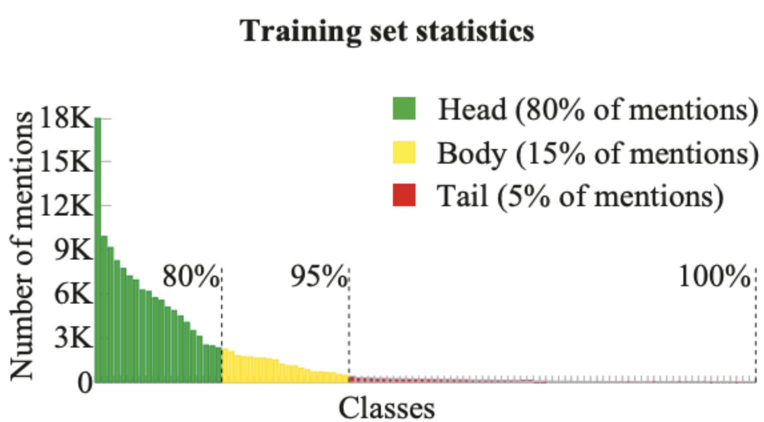
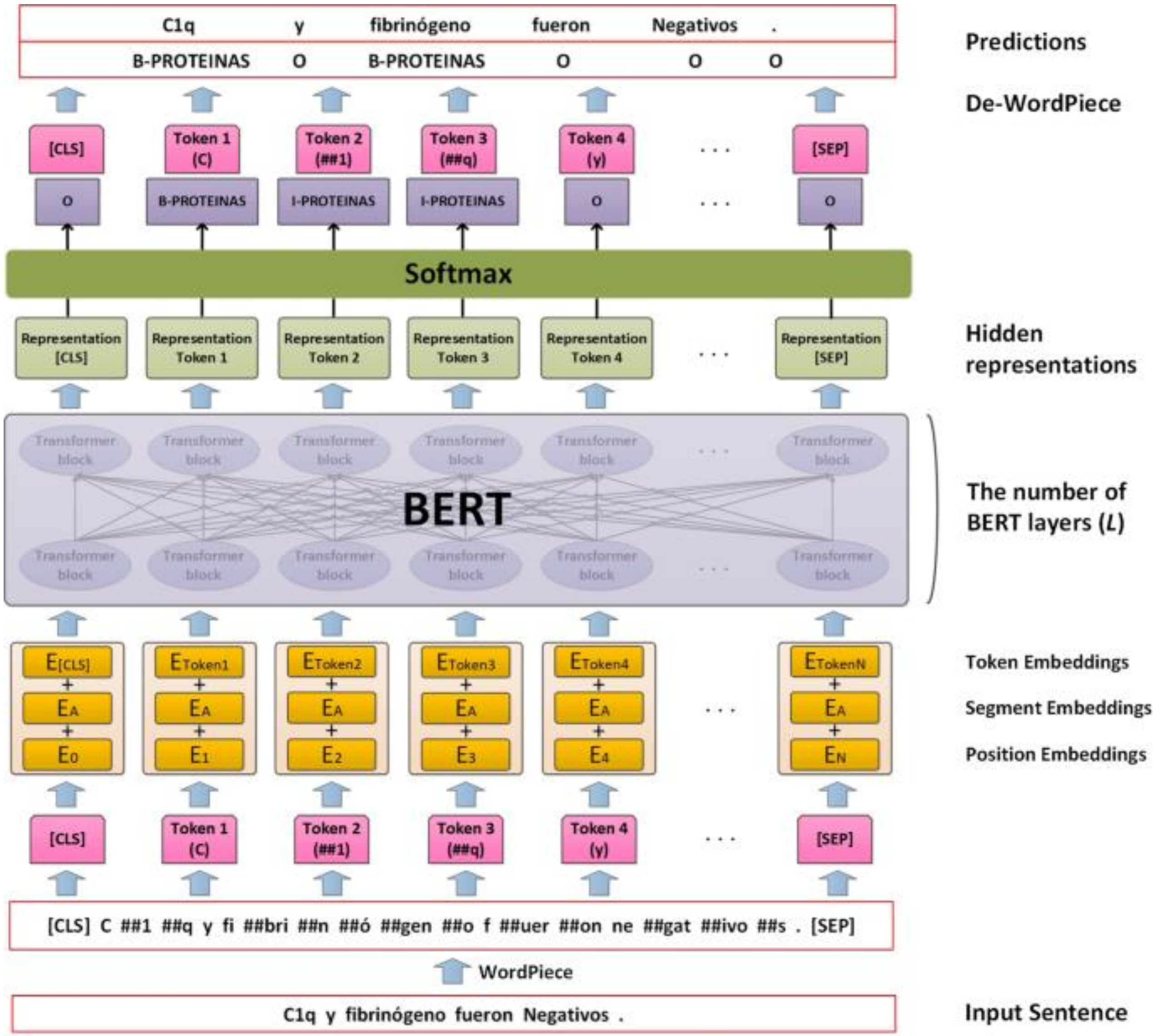


Figure 3: The distribution of classes sorted by frequency shows that rarer classes consist of more than 20% of all instances.

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

BERT and PoS



C1q y fibrinógeno fueron Negativos .
B-PROTEINAS O B-PROTEINAS O O O
Predictions
De-WordPiece
[CLS]
Token 1 (C)
Token 2 (##1)
Token 3 (##q)
Token 4 (y)
[SEP]
O
B-PROTEINAS
I-PROTEINAS
I-PROTEINAS
O
O
Softmax
Representation [CLS]
Representation Token 1
Representation Token 2
Representation Token 3
Representation Token 4
Representation [SEP]
Hidden representations
Transformer block
BERT
The number of BERT layers (L)
Token Embeddings
Segment Embeddings
Position Embeddings
[CLS] C ##1 ##q y fi ##bri ##n ##ó ##gen ##o f ##uer ##on ne ##gat ##ivo ##s . [SEP]
WordPiece
C1q y fibrinógeno fueron Negativos .
Input Sentence

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

Conclusion

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