Java II
Probabilistic Part-of-Speech Tagging

Bernd Kiefer

{Bernd.Kiefer}@dfki.de

Deutsches Forschungszentrum für künstliche Intelligenz

Part-of-Speech Tagging I

- Task: given a sequence of input tokens and a tag set (a set of symbols), assign each word the most appropriate tag
- Example:

Sie	protestierten	gegen	den	Abbau	von	Arbeitsplätzen	.
PPER	VVFIN	APPR	ART	NN	APPR	NN	$.

- Problems:

Part-of-Speech Tagging I

- Task: given a sequence of input tokens and a tag set (a set of symbols), assign each word the most appropriate tag
- Example:

Sie	protestierten	gegen	den	Abbau	von	Arbeitsplätzen	.
`PPER`	`VVFIN`	`APPR`	`ART`	`NN`	`APPR`	`NN`	`$.`

- Problems:
  - ➢ Fix tokenization and tag set

Part-of-Speech Tagging I

- Task: given a sequence of input tokens and a tag set (a set of symbols), assign each word the most appropriate tag
- Example:

Sie	protestierten	gegen	den	Abbau	von	Arbeitsplätzen	.
PPER	VVFIN	APPR	ART	NN	APPR	NN	$.

- Problems:
  - ➢ Fix tokenization and tag set
  - ➢ Multiple possible tags for one word e.g, protestieren: `VVFIN, VVINF` der: `ART, PDS, PRELS`

- Rule based tagging: Write rules manually (mostly regular expressions)

Part-of-Speech Tagging II

- Rule based tagging: Write rules manually (mostly regular expressions)
- Statistical tagging: Get the most probable sequence of tags, given the input sequence as evidence

Part-of-Speech Tagging II

- Rule based tagging: Write rules manually (mostly regular expressions)
- Statistical tagging: Get the most probable sequence of tags, given the input sequence as evidence
- Transformation based tagging: a machine learning method trying to combine the previous ones

Part-of-Speech Tagging II

- Rule based tagging: Write rules manually (mostly regular expressions)
- Statistical tagging: Get the most probable sequence of tags, given the input sequence as evidence
- Transformation based tagging: a machine learning method trying to combine the previous ones
- Statistical tagging needs a corpus of sentences annotated with the correct tags

Part-of-Speech Tagging II

- Rule based tagging: Write rules manually (mostly regular expressions)
- Statistical tagging: Get the most probable sequence of tags, given the input sequence as evidence
- Transformation based tagging: a machine learning method trying to combine the previous ones
- Statistical tagging needs a corpus of sentences annotated with the correct tags
- With this corpus, a special kind of weighted automaton, called *hidden Markov model*, is produced

Part-of-Speech Tagging II

- Rule based tagging: Write rules manually (mostly regular expressions)
- Statistical tagging: Get the most probable sequence of tags, given the input sequence as evidence
- Transformation based tagging: a machine learning method trying to combine the previous ones
- Statistical tagging needs a corpus of sentences annotated with the correct tags
- With this corpus, a special kind of weighted automaton, called *hidden Markov model*, is produced
- Finding the most probable assignment boils down to finding the "best" path through the automaton

Recap: Probability Theory I

Throw a dice three times. How probable is it to get at least one Six?

- We have a finite *Sample Space* Ω: A set of elementary outcomes, e.g., $\{\text{One}, \text{Two}, \text{Three}, \text{Four}, \text{Five}, \text{Six}\}$
- An *event* A is a subset of the sample space, e.g., “the outcome is less than Four” $\{\text{One}, \text{Two}, \text{Three}\}$
- A *probability measure* P is a function from events (i.e., elements of $\mathcal{P}(\Omega)$) to the set of real numbers in $[0, 1]$ with the following properties:
  - $0 \leq P(A) \leq 1$ for each event $A \subseteq \Omega$
  - $P(\Omega) = 1$
  - $A \cap B = \emptyset \Rightarrow P(A \cup B) = P(A) + (B)$ (Additivity)

Recap: Probability Theory II

- Two events are *independent*, $\Leftrightarrow P(A \cap B) = P(A) \cdot P(B)$
- The probability for A if we know that B has occurred is called *conditional probability* $P(A|B)$
  - $P(A|B) = \frac{P(A \cap B)}{P(B)}$: the updated probability if given B has occurred
  - $P(A)$ is often called *prior*, $P(A|B)$ *posterior* probability (knowing B)
  - if A and B are independent:
    $P(A|B) = \frac{P(A \cap B)}{P(B)} = \frac{P(A) \cdot P(B)}{P(B)} = P(A)$

Recap: Probability Theory III

- *Bayes Rule*: $P(A|B) = \dfrac{P(A) \cdot P(B|A)}{P(B)}$
- Useful if $P(B|A)$ is easier to determine than $P(A|B)$
- *Bayes Decomposition*:

$$
\begin{aligned}
& P(A_1 \cap A_2 \cap \ldots \cap A_n) \\
= \quad & P(A_n|A_1 \cap A_2 \cap \ldots \cap A_{n-1}) \cdot P(A_1 \cap \ldots \cap A_{n-1}) \\
= \quad & P(A_n|A_1 \cap \ldots \cap A_{n-1}) \cdot P(A_{n-1}|A_1 \cap \ldots \cap A_{n-2}) \cdot \\
& P(A_1 \cap \ldots \cap A_{n-2}) \\
\vdots \\
= \quad & \prod_{i=1}^{n} P(A_i|A_1 \cap \ldots \cap A_{i-1})
\end{aligned}
$$

Stochastic Processes

- A *stochastic process* is a sequence $X_1, X_2, \dots X_n$ of elementary outcomes of Ω
- A stochastic process is said to be in state X_t at time t
- A *Markov Chain* is a special stochastic process consisting of:
  - A finite set of states $\mathcal{Q} = \{q_1, q_2, \dots, q_n\}$
  - A $n \times n$ *transition matrix* $\mathbf{P}$ specifying the probability of changing from state p to q
  - A vector $\mathbf{v}$ of initial state probabilities
- *Markov property*: the probability of being in the current state, given all former states, depends only on the previous state: $P(q_t|q_1, \dots, q_{t-1}) = P(q_t|q_{t-1})$

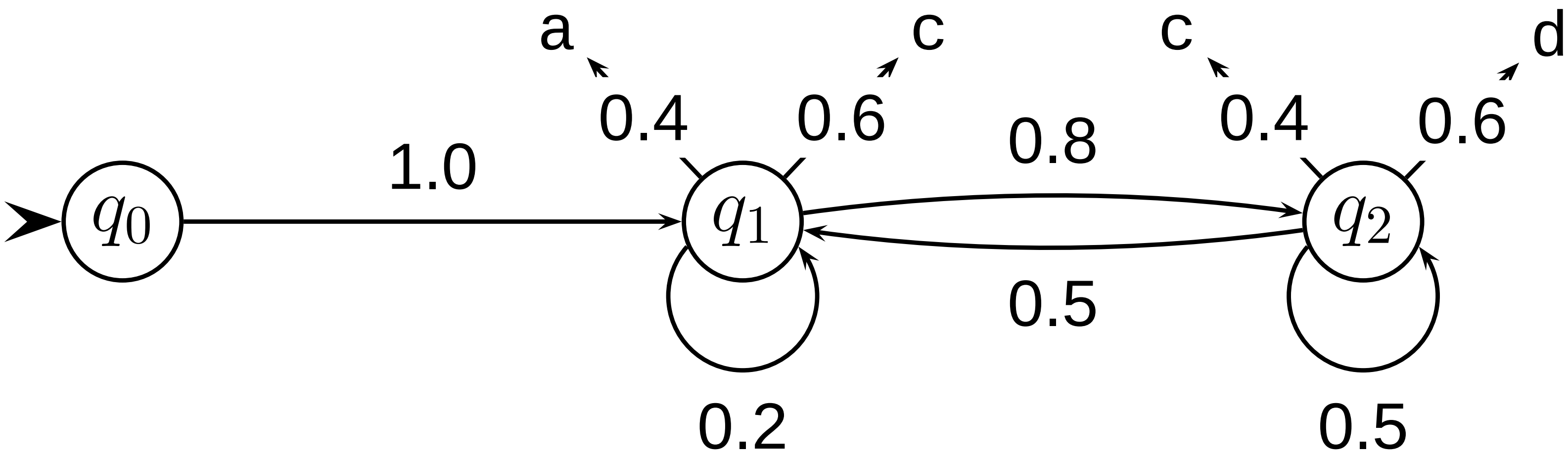
(Hidden) Markov Models

- Attribute each state in a Markov chain with a finite set of signals $\Sigma = \sigma_1, \ldots, \sigma_m$
- After each transition, a symbol from Σ is emitted with some probability
- There is a $n \times m$ *signal matrix* $\mathbf{A} = [a_{ij}]$, which contains the probabilities $p(s = \sigma_i | q = q_j)$
- Markov models contain a second Markov assumption: the probability of the emitted signal only depends on the current state
- If only the emissions are observable, but not the sequence of states, the model is called *Hidden Markov Model (HMM)*

Markov Models II

Example: $\mathcal{Q} = \{q_1, q_2\}$ and $\Sigma = \{a, c, d\}$

$$P = \begin{bmatrix} 0.2 & 0.8 \\ 0.5 & 0.5 \end{bmatrix} \qquad A = \begin{bmatrix} 0.4 & 0.6 & 0.0 \\ 0.0 & 0.4 & 0.6 \end{bmatrix} \qquad v = \begin{bmatrix} 1.0 & 0.0 \end{bmatrix}$$



Probability for emitting c in the second step?

Markov Models III

- What is the probability of emitting σ_j at time t while being in state q_i?
- Two steps:

Markov Models III

- What is the probability of emitting σ_j at time t while being in state q_i?
- Two steps:
  1. Probability of being in state q_{i_t} at time t

Markov Models III

- What is the probability of emitting σ_j at time t while being in state q_i?
- Two steps:
  1. Probability of being in state q_{i_t} at time t

     $\times$
  2. Probability of emitting σ_j when being in state q_{i_t} (remember Markov-assumption no. 2)

Markov Models III

- What is the probability of emitting σ_j at time t while being in state q_i?
- Two steps:
  1. Probability of being in state q_{i_t} at time t

     $\times$

  2. Probability of emitting σ_j when being in state q_{i_t} (remember Markov-assumption no. 2)

$\rightarrow \; p^t(q_i, \sigma_j) = p^t(q_i) \cdot p(s_t = \sigma_j | q_{i_t} = q_i)$

- What is the probability of emitting σ_j at time t while being in state q_i?
- Two steps:
  1. Probability of being in state q_{i_t} at time t

     $\times$
  2. Probability of emitting σ_j when being in state q_{i_t} (remember Markov-assumption no. 2)

  $\rightarrow \; p^t(q_i, \sigma_j) = p^t(q_i) \cdot p(s_t = \sigma_j | q_{i_t} = q_i)$
- 2. is easy: $p(s_t = \sigma_j | q_{i_t} = q_i) = \mathbf{A}[\sigma_j, q_i]$

- What is the probability of emitting σ_j at time t while being in state q_i?
- Two steps:
  1. Probability of being in state q_{i_t} at time t

     $\times$

  2. Probability of emitting σ_j when being in state q_{i_t} (remember Markov-assumption no. 2)

$\rightarrow \; p^t(q_i, \sigma_j) = p^t(q_i) \cdot p(s_t = \sigma_j | q_{i_t} = q_i)$

- 2. is easy: $p(s_t = \sigma_j | q_{i_t} = q_i) = \mathbf{A}[\sigma_j, q_i]$

  Let's look at step 1: what is $p^t(q_i)$?

Markov Models IV

- Probability $p^t(q_i)$ of being in state q_i at time t

Markov Models IV

- Probability $p^t(q_i)$ of being in state q_i at time t
- $p^t(q_i)$ is the i th element of vector $\mathbf{vP}^{t-1}$ Why?
- time zero: $p^0(q_{t_0} = q_i) = \mathbf{v}[q_i]$

Markov Models IV

- Probability $p^t(q_i)$ of being in state q_i at time t
- $p^t(q_i)$ is the i th element of vector $\mathbf{vP}^{t-1}$ Why?
- time zero: $p^0(q_{t_0} = q_i) = \mathbf{v}[q_i]$
- Probability of being in q_{t_1} at time one: $p^1(q_*) = \mathbf{vP} \longrightarrow$

$$p^1(q_{t_1}) = \sum_{i=0}^{n} \underbrace{p^0(q_{t_0} = q_i)}_{\mathbf{v}[q_{t_0}]} \cdot \mathbf{P}[q_i, q_{t_1}] = \mathbf{vP}[*, q_{t_1}]$$

Markov Models IV

- Probability $p^t(q_i)$ of being in state q_i at time t
- $p^t(q_i)$ is the i th element of vector $\mathbf{vP}^{t-1}$ Why?
- time zero: $p^0(q_{t_0} = q_i) = \mathbf{v}[q_i]$
- Probability of being in q_{t_1} at time one: $p^1(q_*) = \mathbf{vP} \longrightarrow$

$$p^1(q_{t_1}) = \sum_{i=0}^{n} \underbrace{p^0(q_{t_0} = q_i)}_{\mathbf{v}[q_{t_0}]} \cdot \mathbf{P}[q_i, q_{t_1}] = \mathbf{vP}[*, q_{t_1}]$$

- q_{t_2} at time two: $p^2(q_*) = \mathbf{vPP}$ etc.

$$p^2(q_{t_2}) = \sum_{i=0}^{n} p^1(q_{t_1} = q_i) \cdot \mathbf{P}[q_i, q_{t_2}] = \mathbf{vPP}[*, q_{t_2}]$$

- Put it together:

$$\begin{aligned} p^t(q_i, \sigma_j) &= p^t(q_i) \cdot p(s_t = \sigma_j | q_{i_t} = q_i) \\ &= \mathbf{v}\mathbf{P}^{t-1}[q_i] \cdot \mathbf{A}[\sigma_j, q_i] \end{aligned}$$

- Put it together:

$$\begin{aligned} p^t(q_i, \sigma_j) &= p^t(q_i) \cdot p(s_t = \sigma_j | q_{i_t} = q_i) \\ &= \mathbf{vP}^{t-1}[q_i] \cdot \mathbf{A}[\sigma_j, q_i] \end{aligned}$$

- Sum over all states to get the probability of emitting σ_j at time t

$$p^t(\sigma_j) = \sum_{i=0}^{n} p^t(q_i) \cdot p(s_t = \sigma_j | q_{i_t} = q_i)$$

- Put it together:

$$\begin{aligned} p^t(q_i, \sigma_j) &= p^t(q_i) \cdot p(s_t = \sigma_j | q_{i_t} = q_i) \\ &= \mathbf{v}\mathbf{P}^{t-1}[q_i] \cdot \mathbf{A}[\sigma_j, q_i] \end{aligned}$$

- Sum over all states to get the probability of emitting σ_j at time t

$$p^t(\sigma_j) = \sum_{i=0}^{n} p^t(q_i) \cdot p(s_t = \sigma_j | q_{i_t} = q_i)$$

- Get the probability for all symbols:

$$[p^t(\sigma_1), \ldots, p^t(\sigma_m)] = \mathbf{v}\mathbf{P}^{t-1}\mathbf{A}$$

- We have a Markov model and a known sequence of emitted signals $\mathbf{S} = \sigma_{i_1} \ldots \sigma_{i_T}$, but we can not observe the sequence of states.

Hidden Markov Models

- We have a Markov model and a known sequence of emitted signals $\mathbf{S} = \sigma_{i_1} \ldots \sigma_{i_T}$, but we can not observe the sequence of states.

$\rightarrow$ the Markov model is a black box, therefore it is called *hidden*

Hidden Markov Models

- We have a Markov model and a known sequence of emitted signals $\mathbf{S} = \sigma_{i_1} \ldots \sigma_{i_T}$, but we can not observe the sequence of states.

→ the Markov model is a black box, therefore it is called *hidden*

- To guess the sequence of states, we are looking for the sequence $\mathbf{Q}$ with the maximal probability, given $\mathbf{S}$:

$$\max_{\mathbf{Q}} P(\mathbf{Q}|\mathbf{S})$$

- We have a Markov model and a known sequence of emitted signals $\mathbf{S} = \sigma_{i_1} \ldots \sigma_{i_T}$, but we can not observe the sequence of states.

$\rightarrow$ the Markov model is a black box, therefore it is called *hidden*

- To guess the sequence of states, we are looking for the sequence $\mathbf{Q}$ with the maximal probability, given $\mathbf{S}$:

$$\max_{\mathbf{Q}} P(\mathbf{Q}|\mathbf{S})$$

- Bayes inversion: $P(\mathbf{Q}|\mathbf{S}) = \frac{P(\mathbf{S}|\mathbf{Q}) \cdot P(\mathbf{Q})}{P(\mathbf{S})}$

independent of $\mathbf{Q}$

- Task: find the most probable sequence of tags for a given sequence of words

POS-Tagging with HMMs

- Task: find the most probable sequence of tags for a given sequence of words
- Use an HMM:
  - ➢ the words are the emitted signals
  - ➢ the tags are the (wanted) state sequence

POS-Tagging with HMMs

- Task: find the most probable sequence of tags for a given sequence of words
- Use an HMM:
  - ➤ the words are the emitted signals
  - ➤ the tags are the (wanted) state sequence
- Suppose we have $\mathbf{v}$, $\mathbf{P}$, and $\mathbf{A}$, how do we compute $\max_{\mathbf{Q}} P(\mathbf{S}|\mathbf{Q}) \cdot P(\mathbf{Q})$ efficiently?
- Define $\theta_t(i)$: maximal probability to be in state q_i at time t: $\theta_t(i) = \max\{P(\sigma_{i_1} \ldots \sigma_{i_t} | q_{j_0} \ldots q_{j_t})\}$ with $j_t = i$

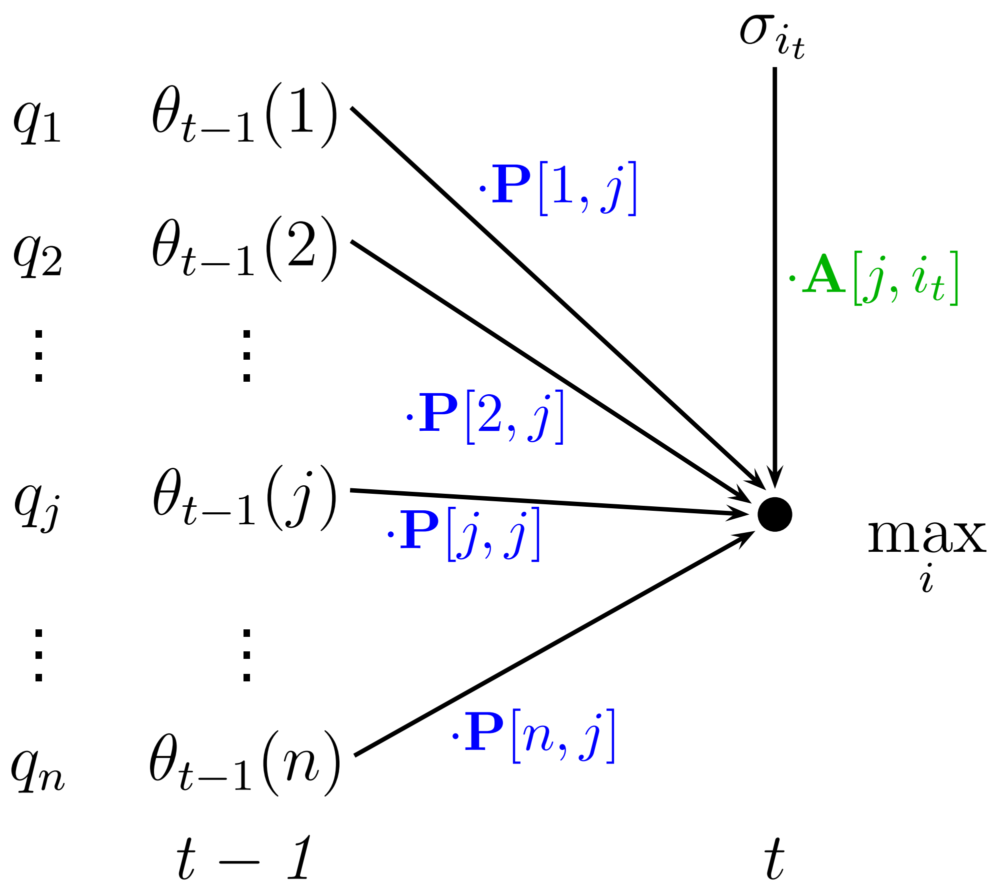
POS-Tagging with HMMs

- Task: find the most probable sequence of tags for a given sequence of words
- Use an HMM:
  - the words are the emitted signals
  - the tags are the (wanted) state sequence
- Suppose we have $\mathbf{v}$, $\mathbf{P}$, and $\mathbf{A}$, how do we compute $\max_{\mathbf{Q}} P(\mathbf{S}|\mathbf{Q}) \cdot P(\mathbf{Q})$ efficiently?
- Define $\theta_t(i)$: maximal probability to be in state q_i at time t: $\theta_t(i) = \max\{P(\sigma_{i_1} \ldots \sigma_{i_t} | q_{j_0} \ldots q_{j_t})\}$ with $j_t = i$
- Brute force method would be exponential in t

POS-Tagging with HMMs

- Task: find the most probable sequence of tags for a given sequence of words
- Use an HMM:
  - the words are the emitted signals
  - the tags are the (wanted) state sequence
- Suppose we have $\mathbf{v}$, $\mathbf{P}$, and $\mathbf{A}$, how do we compute $\max_{\mathbf{Q}} P(\mathbf{S}|\mathbf{Q}) \cdot P(\mathbf{Q})$ efficiently?
- Define $\theta_t(i)$: maximal probability to be in state q_i at time t: $\theta_t(i) = \max\{P(\sigma_{i_1} \ldots \sigma_{i_t} | q_{j_0} \ldots q_{j_t})\}$ with $j_t = i$
- Brute force method would be exponential in t
- Idea: compute the $\theta_t(i)$ recursively using the $\theta_{t-1}(j)$

Viterbi-Algorithm

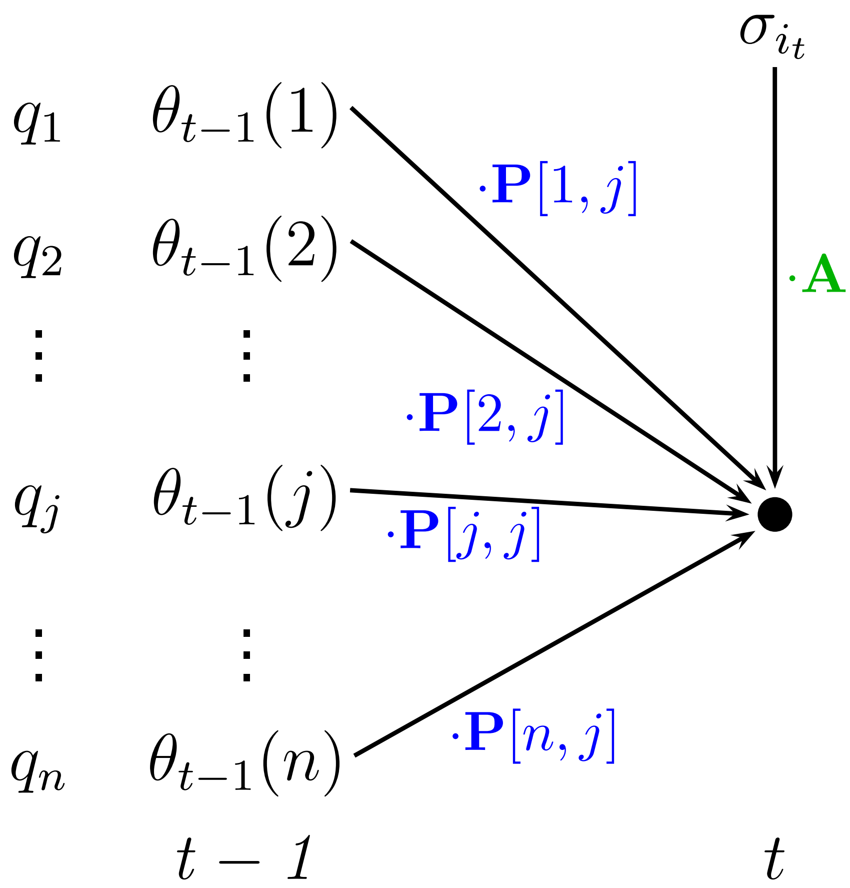


$$\vdots$$

$$\max_i(\theta_{t-1}(i)\cdot\mathbf{P}[i,j])\cdot\mathbf{A}[j,i_t] = \theta_t(j)$$

$$\vdots$$

Viterbi-Algorithm



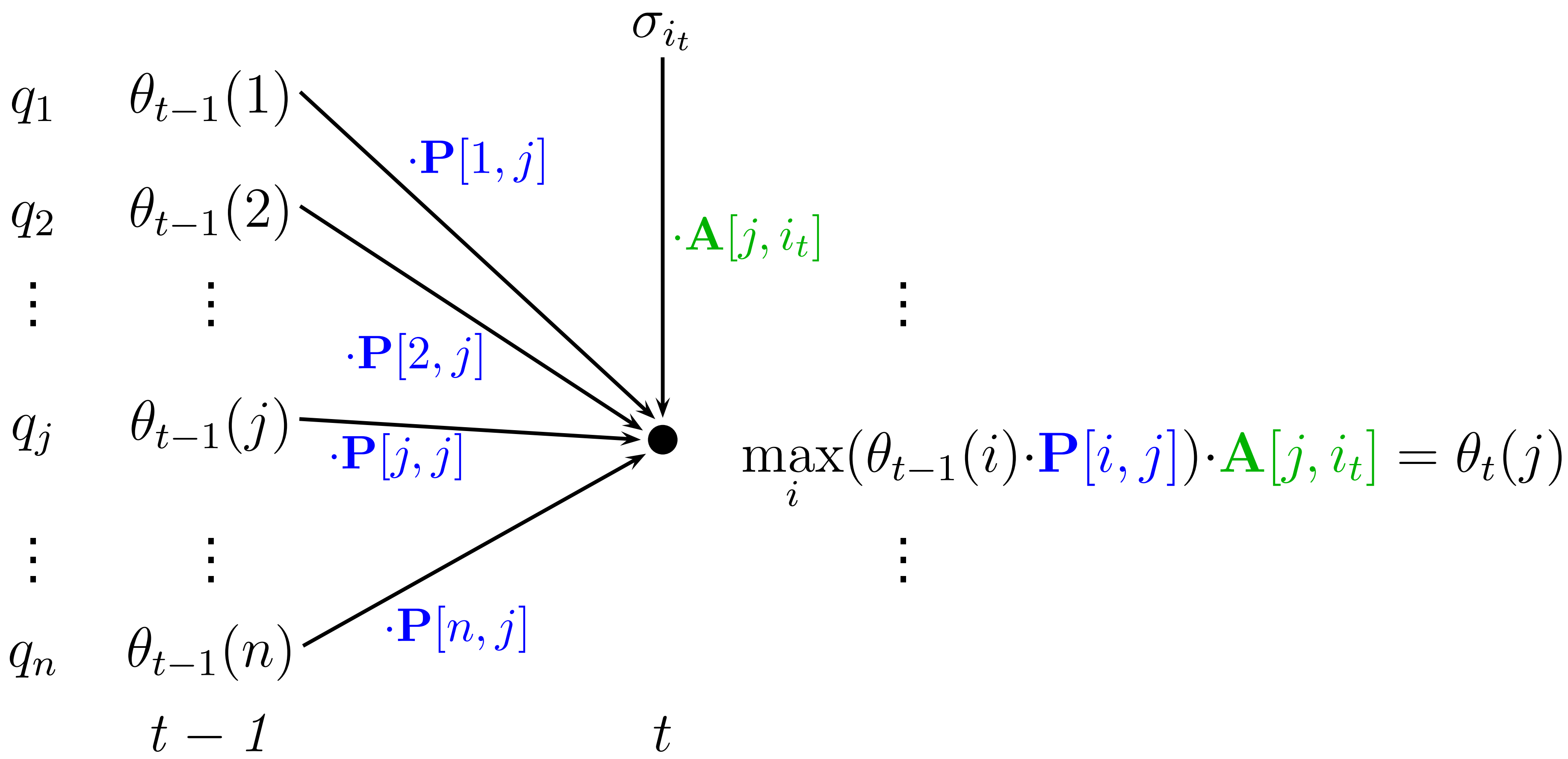
$$\vdots$$

$$\max_i(\theta_{t-1}(i)\cdot\mathbf{P}[i,j])\cdot\mathbf{A}[j,i_t] = \theta_t(j)$$

$$\vdots$$

- One step needs $O(n^2)$ operations (for all states)

Viterbi-Algorithm



- One step needs $O(n^2)$ operations (for all states)
- Overall complexity for T steps is then $O(Tn^2)$

Viterbi-Algorithm

- Initialization: for $j = 1, \ldots, n$: $\theta_1(j) = \underbrace{P(q_{j_1} = q_j)}_{=\mathbf{vP}[j]} \times \mathbf{A}[j, \sigma_{i_1}]$
- Recursion: for $t = 2, \ldots, T$

  for $\quad j = 1, \ldots, n$

  $$\theta_t(j) = \max_i(\theta_{t-1}(i) \cdot \mathbf{P}[i,j]) \cdot \mathbf{A}[j, i_t]$$

  $$\psi_t(j) = \mathrm{argmax}_i(\theta_{t-1}(i) \cdot \mathbf{P}[i,j])$$

- $\psi_t(j)$ saves the *predecessor state* for backchaining
- Termination: $\hat{q}_T = \mathrm{argmax}_i(\theta_T(i))$
- Compute the optimal chain backwards:
  for $t = T-1, \ldots, 1 : \hat{q}_t = \psi_{t+1}(\hat{q}_{t+1})$
- this is an example of a *dynamic programming* algorithm: store previously computed results for structured re-use

Viterbi: Trellis Construction

- The trellis is build layer by layer
- Each layer represents the states connected with the emission of one word
- A layer contains only states for tags found in the lexicon
- One state represents a sequence of two tags: the current and previous tag
- Store these tags in each node (a dummy tag for the first layer) with the max probability and back pointer

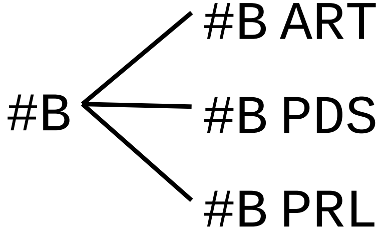
Viterbi for POS Tagging

- For the tagging application, we want to use trigrams, i.e., we use the current tag and two previous tags
- Using trigrams means using a second order Markov model, i.e., each state encodes a sequence of two tags
- With around 60 tags, this means 3600 states per layer!
- But: the emission probability is often zero, and so many state probabilities
- Building the complete trellis is therefore neither feasible nor effective

$\longrightarrow$ Build the graph on the fly and consider only tags associated with the sentence words

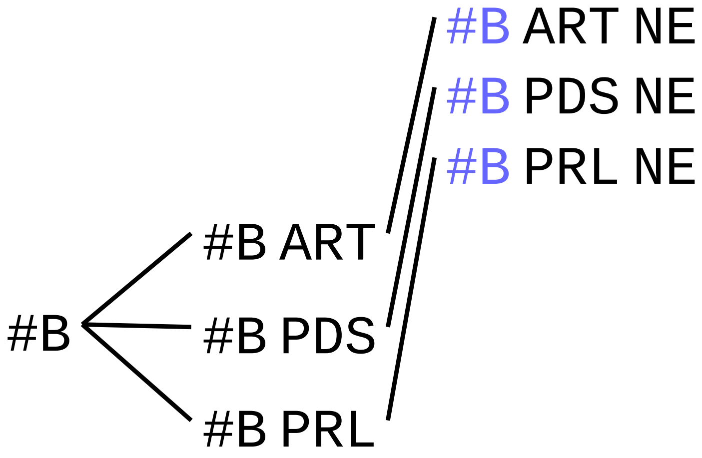
Viterbi: Trellis Example

#B	Der	Mann	mag	Schwarz	#E
#B | A_{11}	ART | A_{21}	NE | A_{31}	VMF | A_{41}	NE | A_{51}	#E | A_{61}
	PDS | A_{22}	NN | A_{32}	VVF | A_{42}	NN | A_{52}	
	PRL | A_{23}				



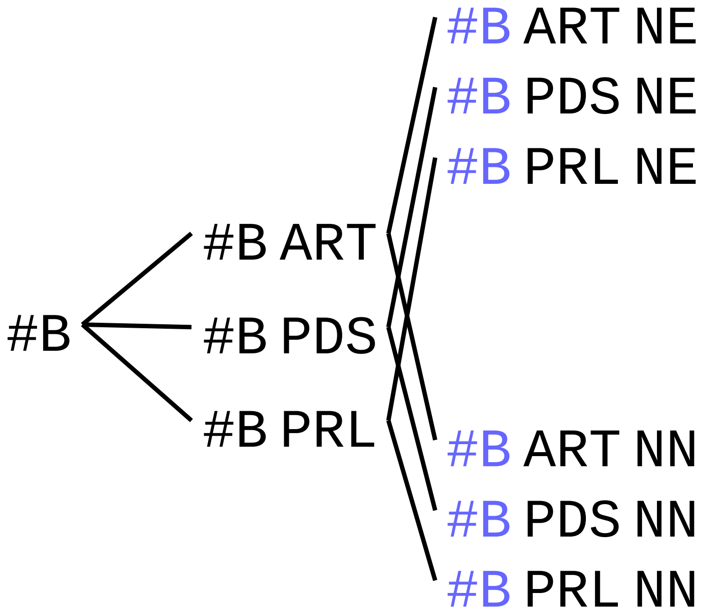
Viterbi: Trellis Example

#B	Der	Mann	mag	Schwarz	#E
#B | A_{11}	ART | A_{21}	NE | A_{31}	VMF | A_{41}	NE | A_{51}	#E | A_{61}
	PDS | A_{22}	NN | A_{32}	VVF | A_{42}	NN | A_{52}	
	PRL | A_{23}				



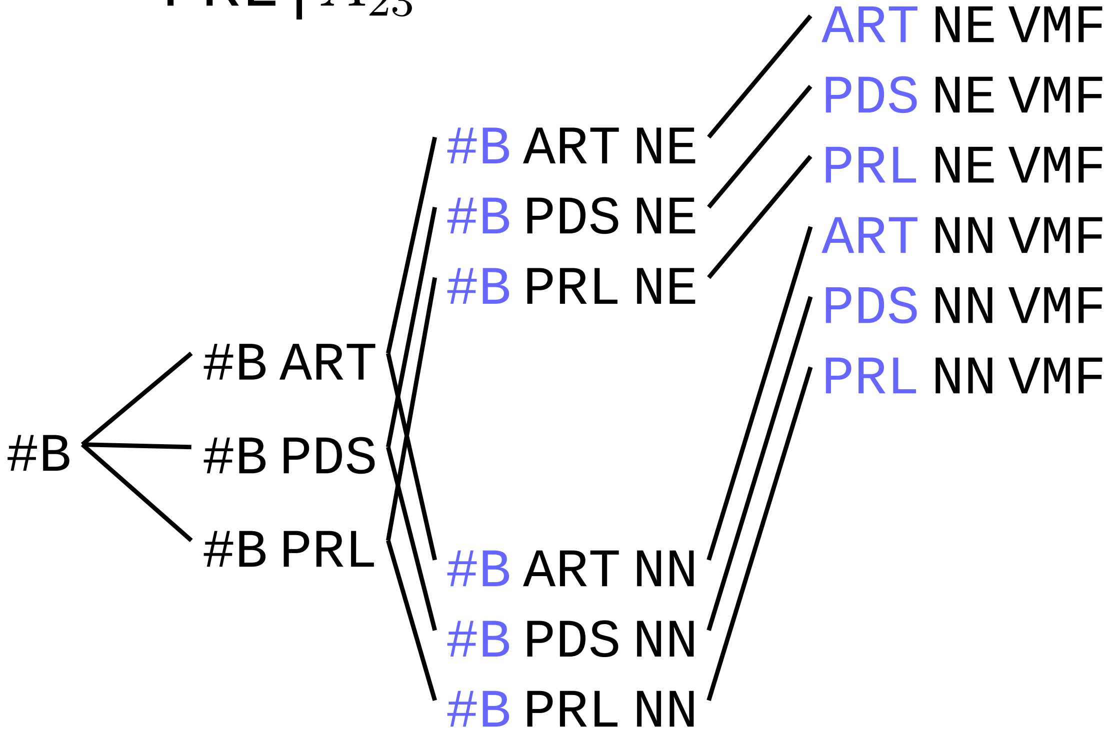
Viterbi: Trellis Example

#B	Der	Mann	mag	Schwarz	#E
#B | A_{11}	ART | A_{21}	NE | A_{31}	VMF | A_{41}	NE | A_{51}	#E | A_{61}
	PDS | A_{22}	NN | A_{32}	VVF | A_{42}	NN | A_{52}	
	PRL | A_{23}				



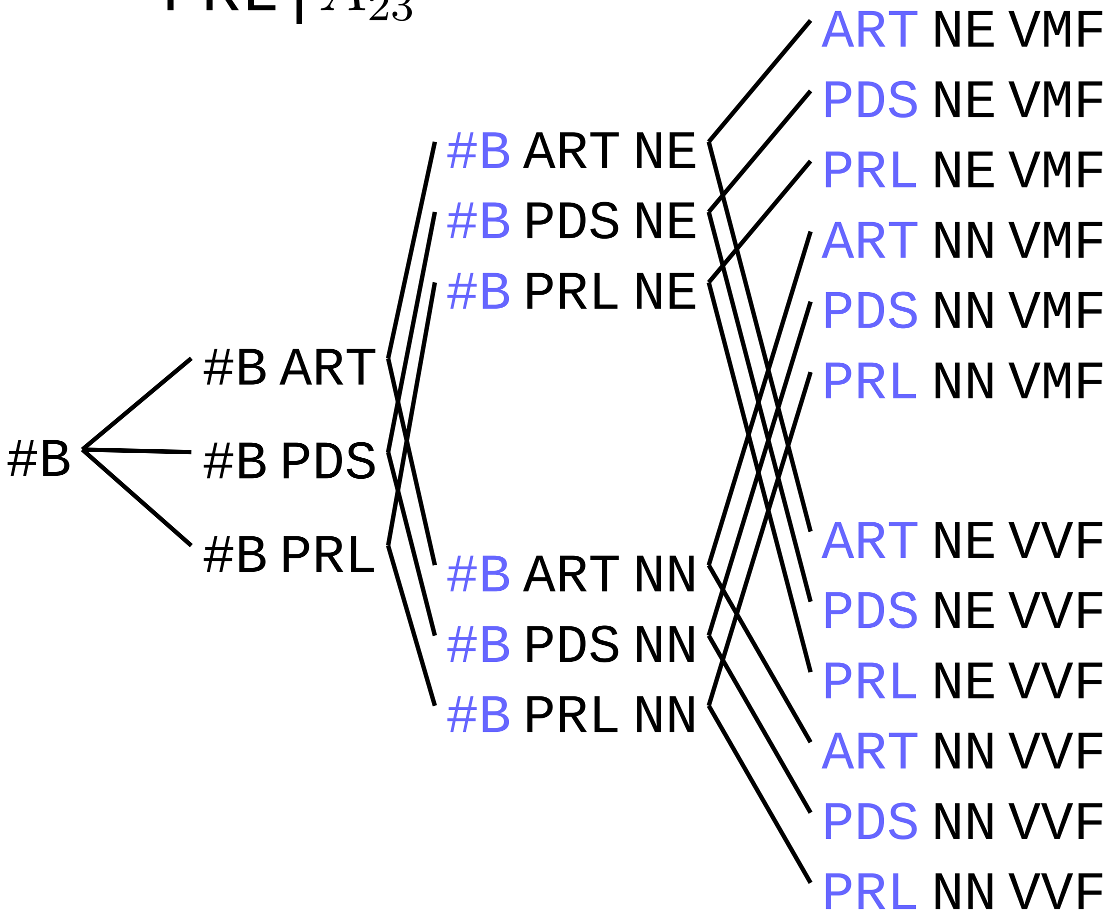
Viterbi: Trellis Example

#B	Der	Mann	mag	Schwarz	#E
#B | A_{11}	ART | A_{21}	NE | A_{31}	VMF | A_{41}	NE | A_{51}	#E | A_{61}
	PDS | A_{22}	NN | A_{32}	VVF | A_{42}	NN | A_{52}	
	PRL | A_{23}				



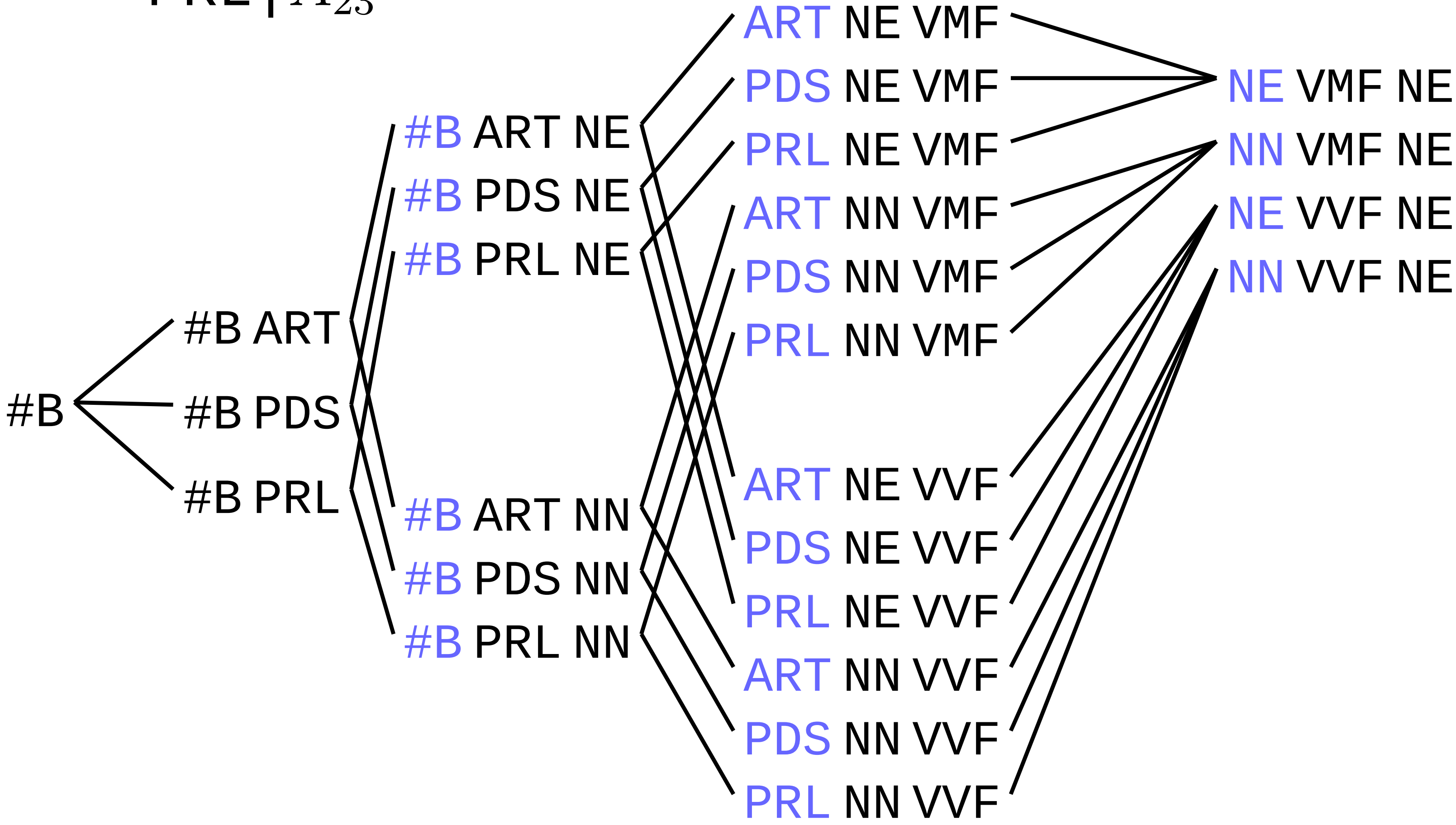
Viterbi: Trellis Example

#B	Der	Mann	mag	Schwarz	#E
#B | A_{11}	ART | A_{21}	NE | A_{31}	VMF | A_{41}	NE | A_{51}	#E | A_{61}
	PDS | A_{22}	NN | A_{32}	VVF | A_{42}	NN | A_{52}	
	PRL | A_{23}				



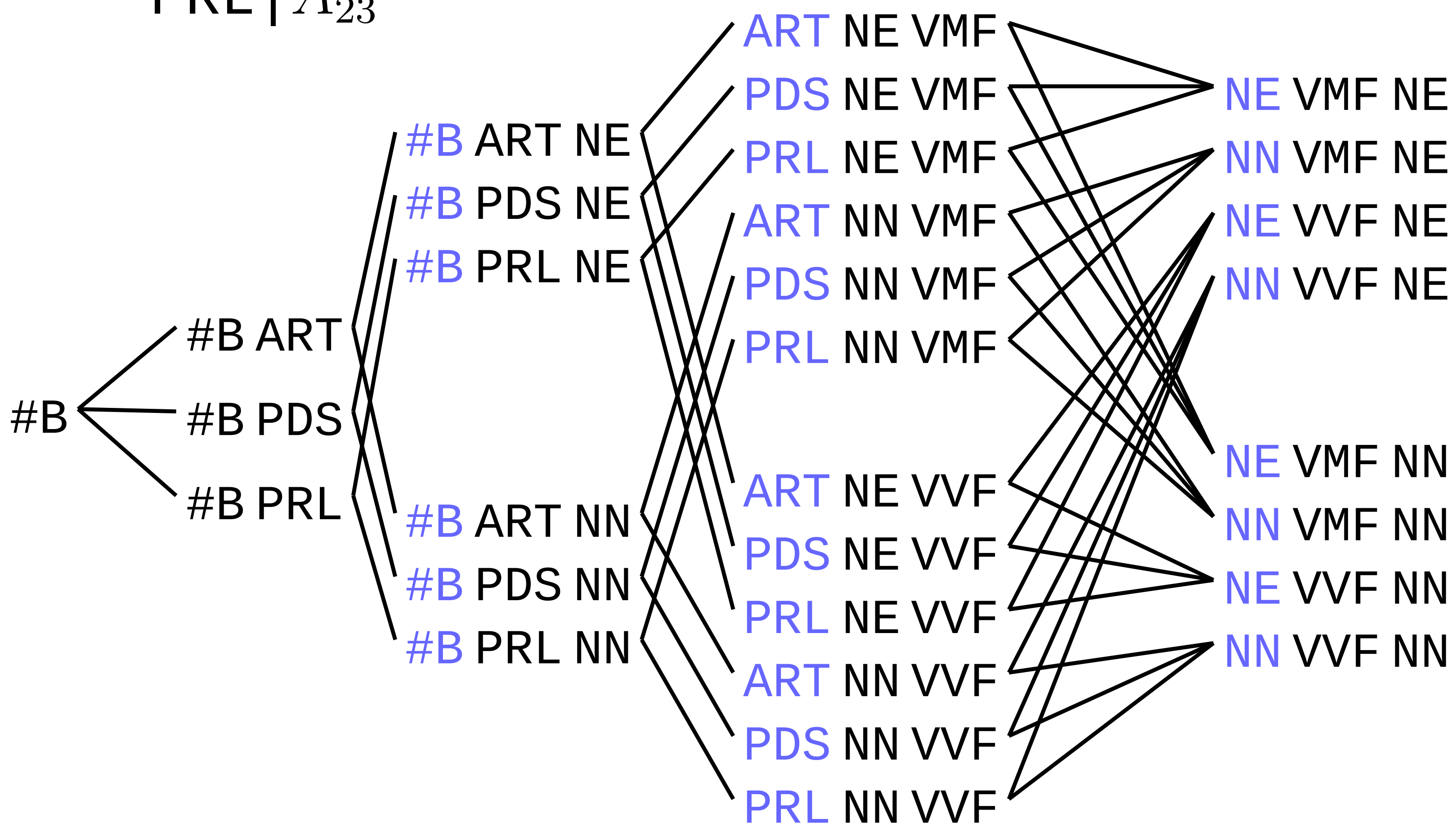
Viterbi: Trellis Example

#B	Der	Mann	mag	Schwarz	#E
#B | A_{11}	ART | A_{21}	NE | A_{31}	VMF | A_{41}	NE | A_{51}	#E | A_{61}
	PDS | A_{22}	NN | A_{32}	VVF | A_{42}	NN | A_{52}	
	PRL | A_{23}				



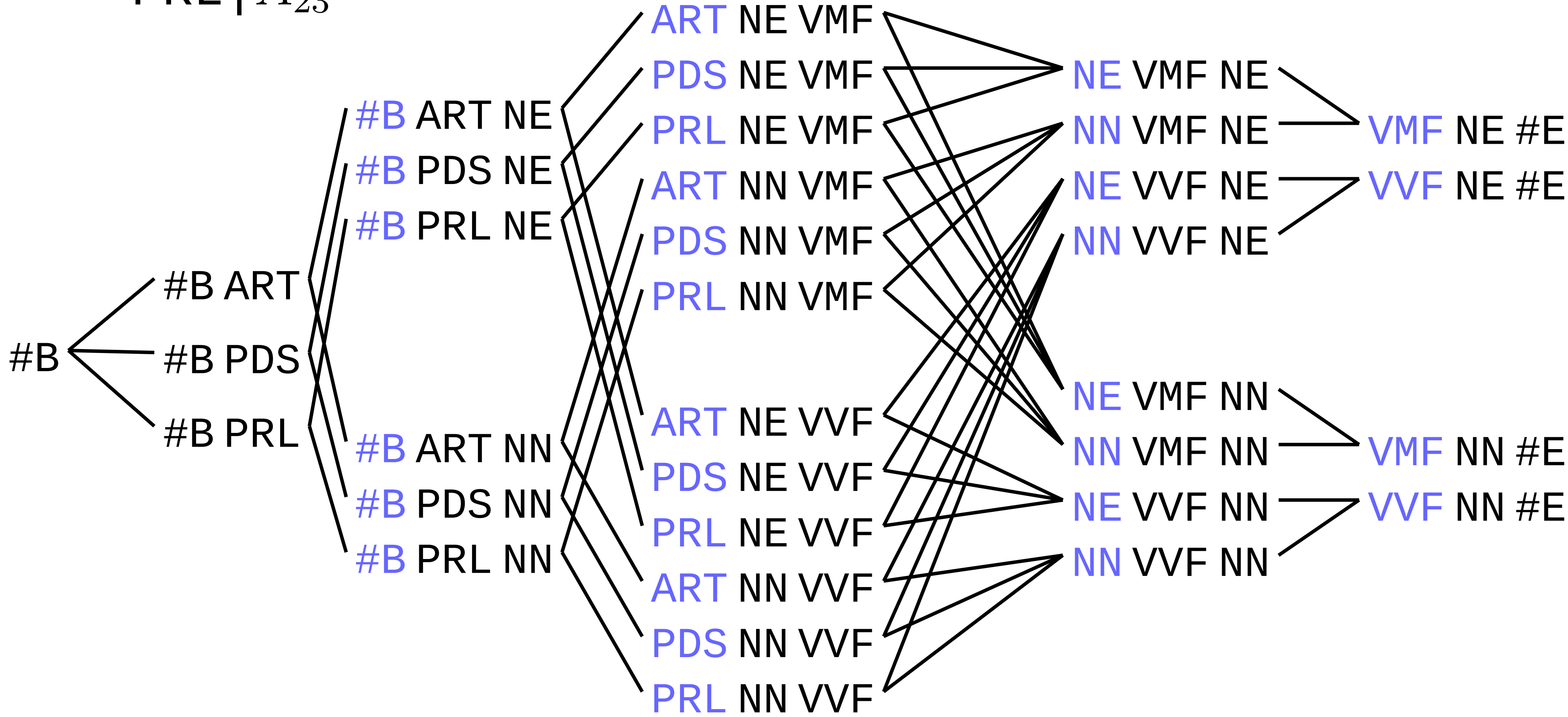
Viterbi: Trellis Example

#B	Der	Mann	mag	Schwarz	#E
#B | A_{11}	ART | A_{21}	NE | A_{31}	VMF | A_{41}	NE | A_{51}	#E | A_{61}
	PDS | A_{22}	NN | A_{32}	VVF | A_{42}	NN | A_{52}	
	PRL | A_{23}				



Viterbi: Trellis Example

#B	Der	Mann	mag	Schwarz	#E
#B | A_{11}	ART | A_{21}	NE | A_{31}	VMF | A_{41}	NE | A_{51}	#E | A_{61}
	PDS | A_{22}	NN | A_{32}	VVF | A_{42}	NN | A_{52}	
	PRL | A_{23}				



proc Build layer t of the trellis $\equiv$ $l_t = 0$ // number of states in layer t
  for every tag T_k^t associated with word t **do**
    new state$(T_k^t, t, 1)$
    for $j = 2 \ldots l_{t-1}$ **do** // for all states in layer $t-1$
      if $\mathsf{T}_j^{t-1} \neq \mathsf{T}_{j-1}^{t-1}$ **then** // one of the tags differs
        $\theta_t(l_t) = \theta_t(l_t) \cdot A_{tk};$ *new_state*(T_k^t, t, j)
      elsif $\theta_t(l_t) < \theta_{t-1}(j) \cdot P(T_k^t | \mathsf{T0}_j^{t-1}\, \mathsf{T1}_j^{t-1})$ **then**
        // update probability and backpointer
        $\theta_t(l_t) = \theta_{t-1}(j) \cdot P(T_k^t | \mathsf{T0}_j^{t-1}\, \mathsf{T1}_j^{t-1});\ \psi_t(l_t) = j$

proc *new_state*(T, t, j) $\equiv$
  $l_t = l_t + 1;$ // increase the number of states
  $\mathsf{T}_{l_t}^t = [\mathsf{T1}_j^{t-1}, T];$ // set tags
  // initial prob and backpointer
  $\theta_t(l) = \theta_{t-1}(j) \cdot P(T_k^t | \mathsf{T0}_j^{t-1}\, \mathsf{T1}_j^{t-1});\ \psi_t(l) = j$

Acquiring the Model

- Two main possibilities:
  1. Unsupervised training: Lexicon + (optional) Bias + EM-Training
  2. Supervised training: Requires a tagged corpus
- Variant 1 is too complicated for this course, visit Prof. Klakow's *language modeling*, if you're interested
- Variant 2: the model parameters $\mathbf{v}$, $\mathbf{P}$ and $\mathbf{A}$ are basically relative frequencies.
- For $\mathbf{P}$, we use Unigrams: $P(t_j)$, Bigrams: $P(t_j|t_{j-1})$ and Trigrams: $P(t_j|t_{j-2}t_{j-1})$, i.e., a second order model

Acquiring the Model II

- Formulae for the model parameters

$$\text{Unigrams:} \quad \hat{P}(t_j) = \frac{f(t_j)}{N}$$

$$\text{Bigrams:} \quad \hat{P}(t_j|t_{j-1}) = \frac{f(t_{j-1}t_j)}{f(t_{j-1})}$$

$$\text{Trigrams:} \quad \hat{P}(t_j|t_{j-2}t_{j-1}) = \frac{f(t_{j-2}t_{j-1}t_j)}{f(t_{j-2}t_{j-1})}$$

$$\text{Lexical:} \quad \hat{P}(w_k|t_j) = \frac{f(w_k,t_j)}{f(t_j)} = \mathbf{A}[j,k]$$

- $f(t_{j-1}t_j)$ is the number of times the tag sequence $t_{j-1}t_j$ occurs in the corpus, N the total number of tags.
- If the denominator is zero, define the probability to be zero

Using the NEGRA Corpus

- Get the file `pos-corpus.txt` from the course homepage
- The initial `WORDTAG` section contains all used POS tags
- After the tables follow the sentences, each starting with `#BOS` and ending with `#EOS`. Treat these markers as words, too!
- Every line in a sentence contains the word in the first column and the attached tag in the second column
- Compute n-gram and lexical probabilities from this file
- The v vector will only have probability 1 for emitting `#BOS` in the first place
- Transform all words to lower case (reduces model size, but also quality)

- What if $f(...)$ is zero for some configuration?

Smoothing the Model

- What if $f(...)$ is zero for some configuration?
- The lattice probability drops to zero although it might only be missing in the training data

Smoothing the Model

- What if $f(...)$ is zero for some configuration?
- The lattice probability drops to zero although it might only be missing in the training data
- Unseen n-grams require *backing off* to some other information source, e.g., the $n-1$-gram

Smoothing the Model

- What if $f(...)$ is zero for some configuration?
- The lattice probability drops to zero although it might only be missing in the training data
- Unseen n-grams require *backing off* to some other information source, e.g., the $n-1$-gram
- back-off can be combined with *model smoothing*

Smoothing the Model

- What if $f(...)$ is zero for some configuration?
- The lattice probability drops to zero although it might only be missing in the training data
- Unseen n-grams require *backing off* to some other information source, e.g., the $n-1$-gram
- back-off can be combined with *model smoothing*
- TnT's method of smoothing: linear interpolation

$$P(t_3|t_1t_2) = \lambda_1\hat{P}(t_3) + \lambda_2\hat{P}(t_3|t_2) + \lambda_3\hat{P}(t_3|t_1t_2) \quad \lambda_i \geq 0$$

Smoothing the Model

- What if $f(...)$ is zero for some configuration?
- The lattice probability drops to zero although it might only be missing in the training data
- Unseen n-grams require *backing off* to some other information source, e.g., the $n-1$-gram
- back-off can be combined with *model smoothing*
- TnT's method of smoothing: linear interpolation

$$P(t_3|t_1t_2) = \lambda_1\hat{P}(t_3) + \lambda_2\hat{P}(t_3|t_2) + \lambda_3\hat{P}(t_3|t_1t_2) \quad \lambda_i \geq 0$$

- Constraint $\lambda_1 + \lambda_2 + \lambda_3 = 1$ turns P into a probability distribution

Smoothing the Model

- What if $f(...)$ is zero for some configuration?
- The lattice probability drops to zero although it might only be missing in the training data
- Unseen n-grams require *backing off* to some other information source, e.g., the $n-1$-gram
- back-off can be combined with *model smoothing*
- TnT's method of smoothing: linear interpolation

$$P(t_3|t_1t_2) = \lambda_1\hat{P}(t_3) + \lambda_2\hat{P}(t_3|t_2) + \lambda_3\hat{P}(t_3|t_1t_2) \quad \lambda_i \geq 0$$

- Constraint $\lambda_1 + \lambda_2 + \lambda_3 = 1$ turns P into a probability distribution
- Use *deleted interpolation* to acquire the λ_i

Smoothing Algorithm

set $\lambda_1 = \lambda_2 = \lambda_3 = 0$

for each trigram $t_1 t_2 t_3$ with $f(t_1 t_2 t_3) > 0$

depending on which of the next three is maximal

case $\dfrac{f(t_1 t_2 t_3) - 1}{f(t_1 t_2) - 1}$: increment λ_3 by $f(t_1 t_2 t_3)$

case $\dfrac{f(t_2 t_3) - 1}{f(t_2) - 1}$: increment λ_2 by $f(t_1 t_2 t_3)$

case $\dfrac{f(t_3) - 1}{N - 1}$: increment λ_1 by $f(t_1 t_2 t_3)$

normalize $\lambda_1, \lambda_2, \lambda_3$: $\lambda_i = \dfrac{\lambda_i}{\sum_{j=1}^{3} \lambda_j}$

Unknown Words

- Words not in the training data are similar to unseen n-grams
- TnT uses a *suffix heuristic* to estimate the lexicon probabilities for unknown words from the word endings
- A simpler approach: average over the frequencies of infrequently occuring words $W' = \{w : f(w) < c\}$

$$\tilde{f}(\text{<unk>}, t) = \frac{\sum_{w' \in W'} f(w', t)}{\sum_{w' \in W'} f(w')} \Rightarrow \tilde{P}(\text{<unk>}|t) = \tilde{f}(\text{<unk>}, t)/f(t)$$

- To renormalize, we have to solve the equations

$$1 = \lambda_t(\sum_{w \in W} \hat{P}(w|t) + \tilde{P}(\text{<unk>}, t)) \Rightarrow \lambda_t = \frac{f(t)}{\tilde{f}(\text{<unk>}, t) + \sum_{w \in W} f(w, t)}$$