Lecture 9: Decoding

Andreas Maletti

Statistical Machine Translation

Stuttgart — January 20, 2012

Lecture 9

Last time

- Synchronous grammars (tree transducers)
- Rule extraction
- Weight training (EM)

This time

- Inside and outside weights
- *n*-best derivations
- Cube pruning
- Factorization

Lecture 9

Last time

- Synchronous grammars (tree transducers)
- Rule extraction
- Weight training (EM)

This time

- Inside and outside weights (again)
- n-best derivations
- Cube pruning
- Factorization

Lecture 9

Last time

- Synchronous grammars (tree transducers)
- Rule extraction
- Weight training (EM)

This time

- Inside and outside weights
- *n*-best derivations
- Cube pruning
- Factorization

Lecture 9

Last time

- Synchronous grammars (tree transducers)
- Rule extraction
- Weight training (EM)

This time

- Inside and outside weights
- *n*-best derivations
- Cube pruning
- Factorization

Lecture 9

Last time

- Synchronous grammars (tree transducers)
- Rule extraction
- Weight training (EM)

This time

- Inside and outside weights
- *n*-best derivations
- Cube pruning
- Factorization

Contents

1. Inside and Outside Weights
2. *n*-best Derivations
3. Cube Pruning
4. Factorization

State Metrics of a Weighted Tree Grammar

Given weighted tree grammar $G = (Q, \Sigma, I, R)$

D_M^q: derivations in M starting from state q

Definition

Inside weight of $q \in Q$:

$$\text{in}(q) = \sum_{d \in D_M^q} \text{wt}(d)$$

Outside weight of $q \in Q$:

$$\text{out}(q) = \sum_{\substack{t \in C_\Sigma(\{q\}) \\ d \in D_M(t)}} \text{wt}(d)$$

Inside Weight

Question

Why did nobody complain about the infinite sum?

$$\text{in}(q) = \sum_{d \in D_M^q} \text{wt}(d)$$

Computing Inside Weights

Theorem

$$\mathrm{in}(q) = \sum_{q \xrightarrow[c]{} \sigma(q_1,\ldots,q_k) \in R} c \cdot \mathrm{in}(q_1) \cdot \ldots \cdot \mathrm{in}(q_k)$$

Problem

But that is not a well-founded recursion!

Turn it into a system of equations

$$0 = -\mathrm{in}(q) + \sum_{q \xrightarrow[c]{} \sigma(q_1,\ldots,q_k) \in R} c \cdot \mathrm{in}(q_1) \cdot \ldots \cdot \mathrm{in}(q_k)$$

$$0 = -\mathrm{in}(p) + \cdots$$

$|Q|$ variables and $|Q|$ equations

Computing Inside Weights

Theorem

$$\mathrm{in}(q) = \sum_{q \xrightarrow[c]{} \sigma(q_1,\ldots,q_k) \in R} c \cdot \mathrm{in}(q_1) \cdot \ldots \cdot \mathrm{in}(q_k)$$

Problem

But that is not a well-founded recursion!

Turn it into a system of equations

$$0 = -\mathrm{in}(q) + \sum_{q \xrightarrow[c]{} \sigma(q_1,\ldots,q_k) \in R} c \cdot \mathrm{in}(q_1) \cdot \ldots \cdot \mathrm{in}(q_k)$$

$$0 = -\mathrm{in}(p) + \cdots$$

$|Q|$ variables and $|Q|$ equations

Computing Inside Weights

Theorem

$$\text{in}(q) = \sum_{q \xrightarrow[c]{} \sigma(q_1,\ldots,q_k) \in R} c \cdot \text{in}(q_1) \cdot \ldots \cdot \text{in}(q_k)$$

Problem

But that is not a well-founded recursion!

Turn it into a system of equations

$$0 = -\text{in}(q) + \sum_{q \xrightarrow[c]{} \sigma(q_1,\ldots,q_k) \in R} c \cdot \text{in}(q_1) \cdot \ldots \cdot \text{in}(q_k)$$

$$0 = -\text{in}(p) + \cdots$$

$|Q|$ variables and $|Q|$ equations

Solving a system of equations

$|Q|$ variables and $|Q|$ equations

$$0 = -\operatorname{in}(q) + \sum_{q \xrightarrow[c]{} \sigma(q_1,\dots,q_k) \in R} c \cdot \operatorname{in}(q_1) \cdot \ldots \cdot \operatorname{in}(q_k)$$

$$0 = -\operatorname{in}(p) + \cdots$$

Question

Is it a linear system of equations?

Approximation

Use any method for approximating zero-points of functions

- Regula Falsi
- Tangent method (NEWTON-RAPHSON)
- Secant method
- ...

Solving a system of equations

$|Q|$ variables and $|Q|$ equations

$$0 = -\operatorname{in}(q) + \sum_{q \xrightarrow[c]{} \sigma(q_1,\dots,q_k) \in R} c \cdot \operatorname{in}(q_1) \cdot \ldots \cdot \operatorname{in}(q_k)$$

$$0 = -\operatorname{in}(p) + \cdots$$

Question

Is it a linear system of equations? NO!

Approximation

Use any method for approximating zero-points of functions

- Regula Falsi
- Tangent method (NEWTON-RAPHSON)
- Secant method
- ...

Solving a system of equations

$|Q|$ variables and $|Q|$ equations

$$0 = -\operatorname{in}(q) + \sum_{q \xrightarrow[c]{} \sigma(q_1, \ldots, q_k) \in R} c \cdot \operatorname{in}(q_1) \cdot \ldots \cdot \operatorname{in}(q_k)$$

$$0 = -\operatorname{in}(p) + \cdots$$

Question

Is it a linear system of equations? NO!

Approximation

Use any method for approximating zero-points of functions

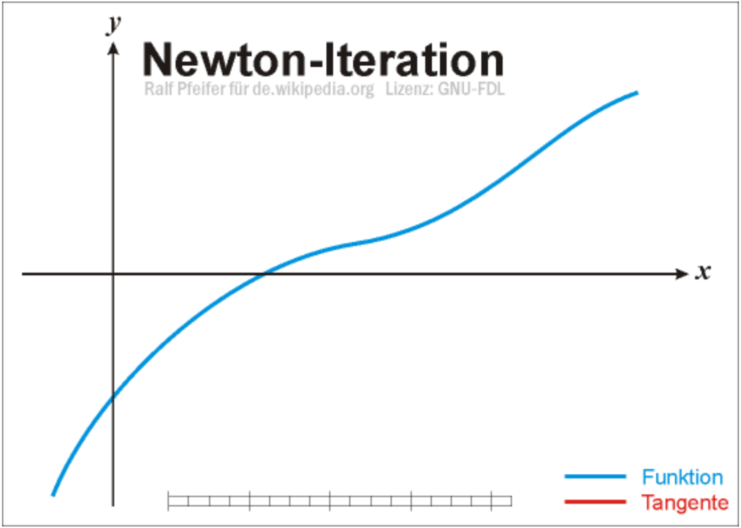
- Regula Falsi
- Tangent method (Newton-Raphson)
- Secant method
- ...

Newton-Raphson method

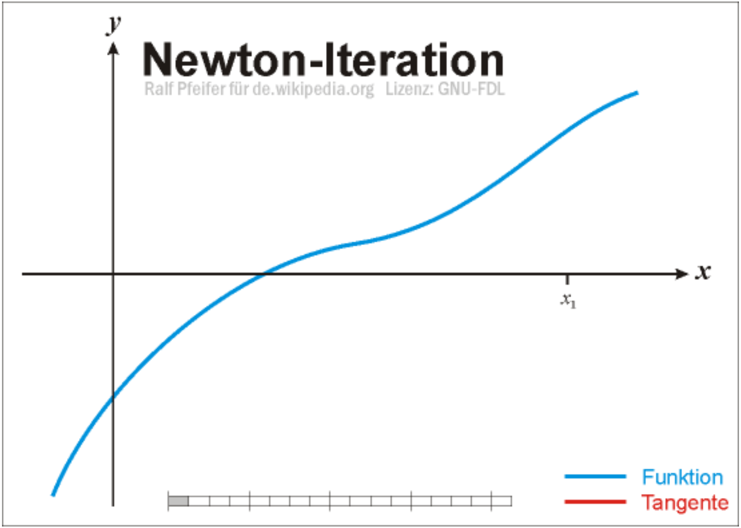
Algorithm

1. Select initial point p
2. Determine tangent to curve at p
3. Determine zero-point p of tangent
4. Go back to 2

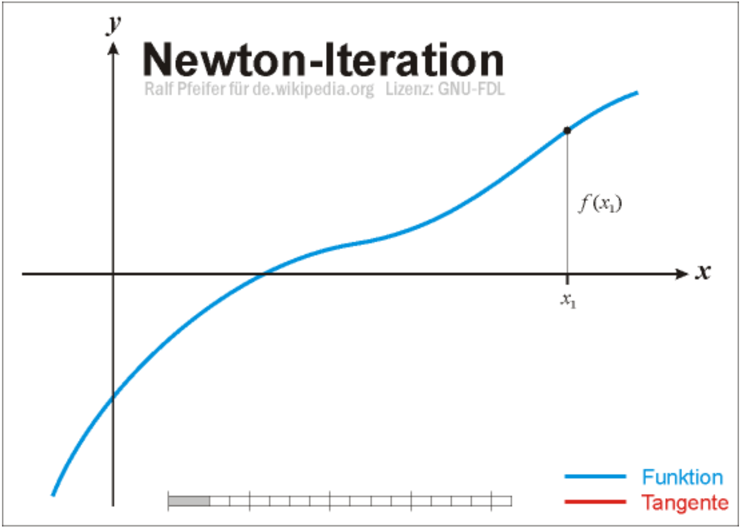
Newton-Raphson Iteration



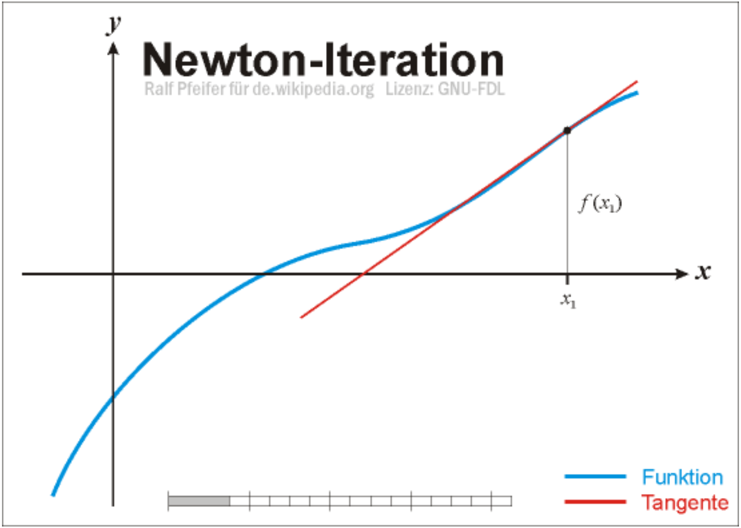
Newton-Raphson Iteration



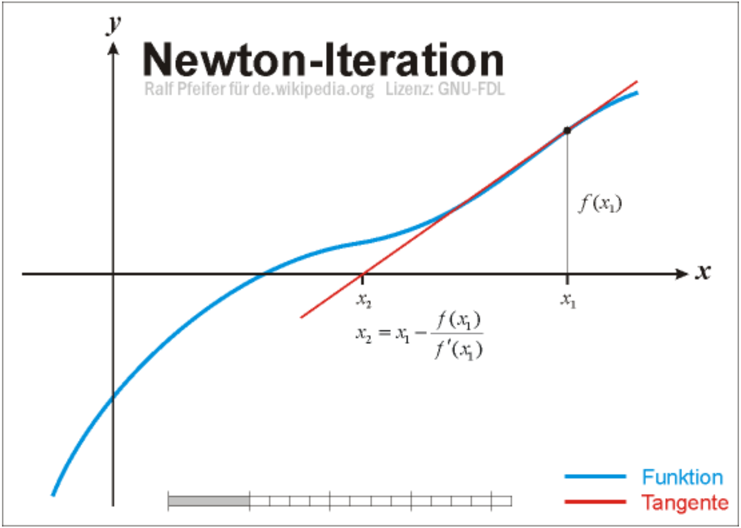
Newton-Raphson Iteration



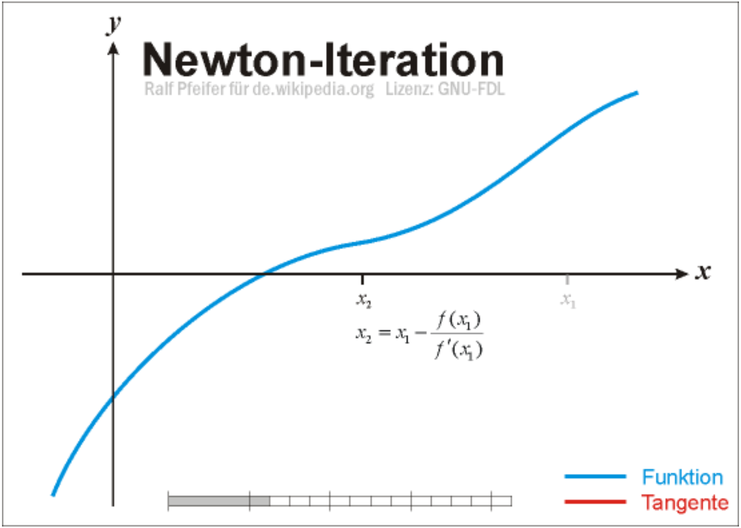
Newton-Raphson Iteration



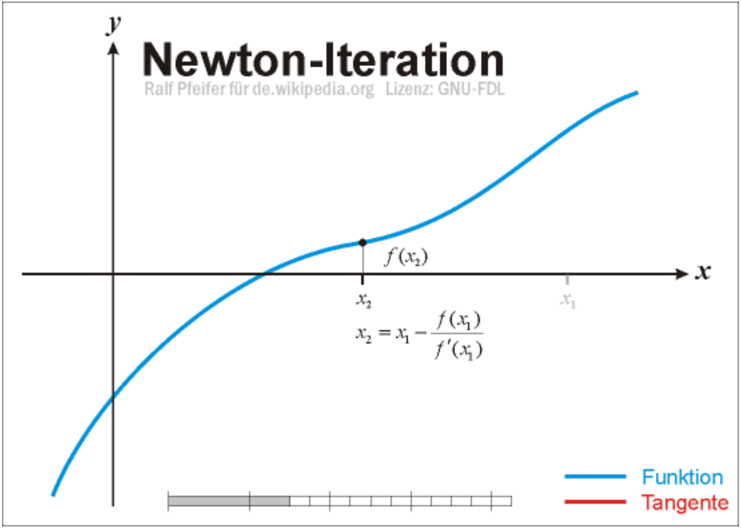
Newton-Raphson Iteration



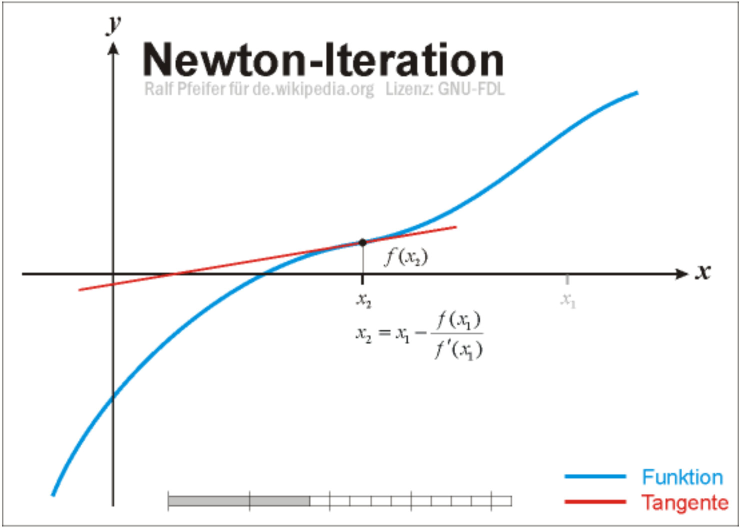
Newton-Raphson Iteration



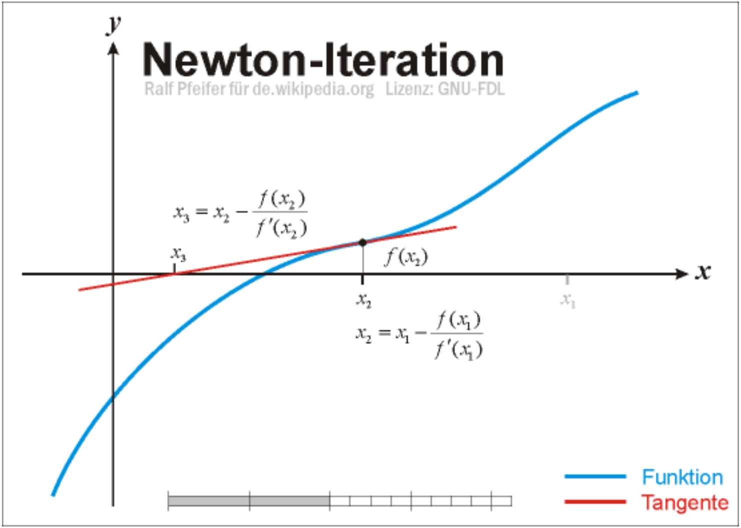
Newton-Raphson Iteration



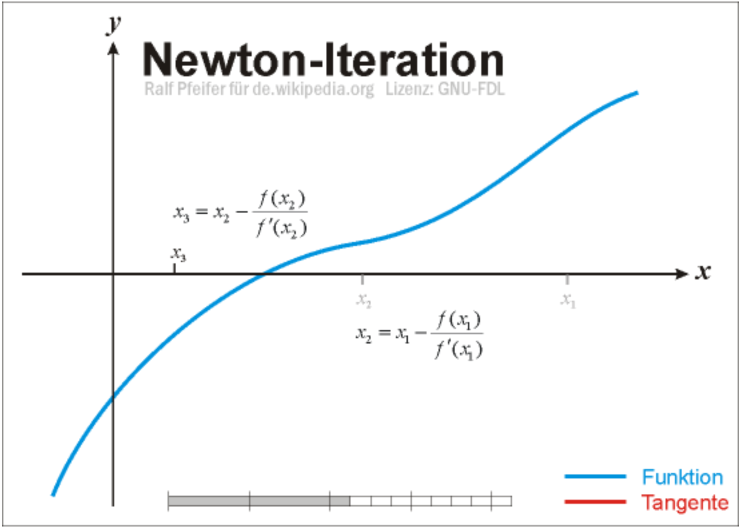
Newton-Raphson Iteration



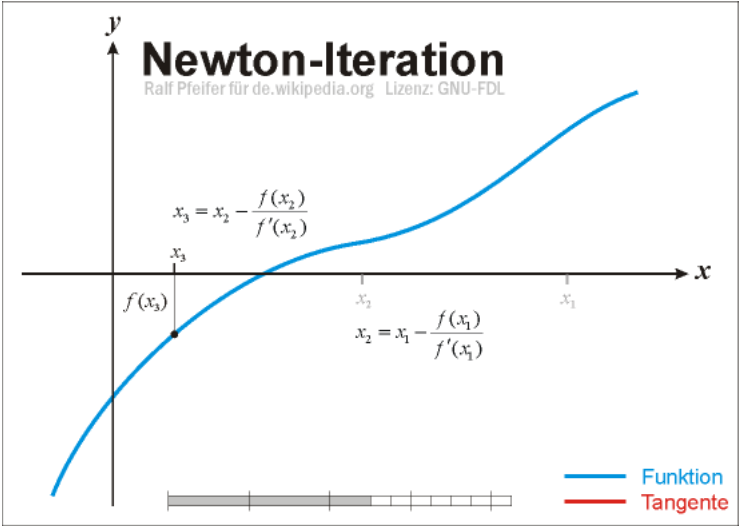
Newton-Raphson Iteration



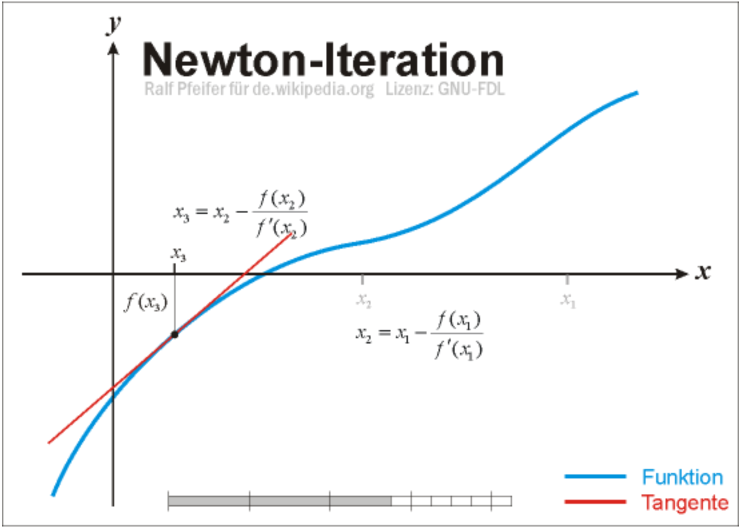
Newton-Raphson Iteration



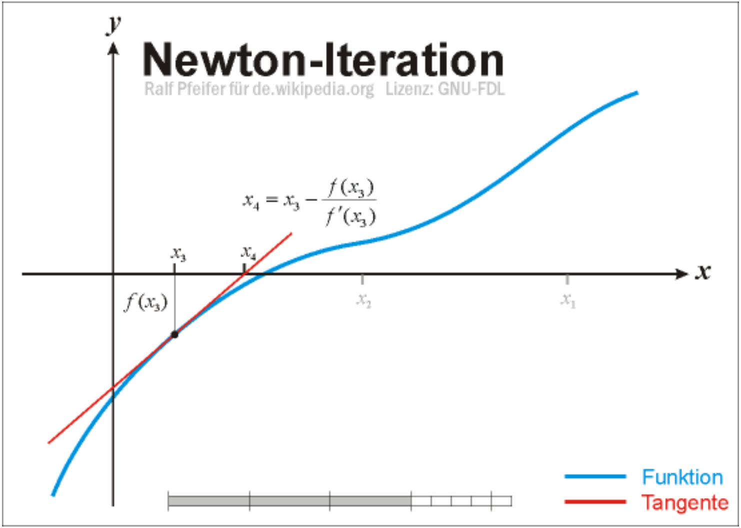
Newton-Raphson Iteration



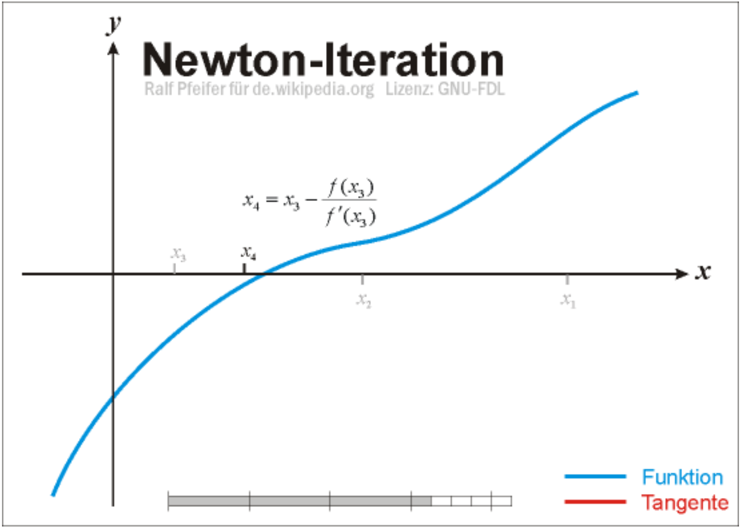
Newton-Raphson Iteration



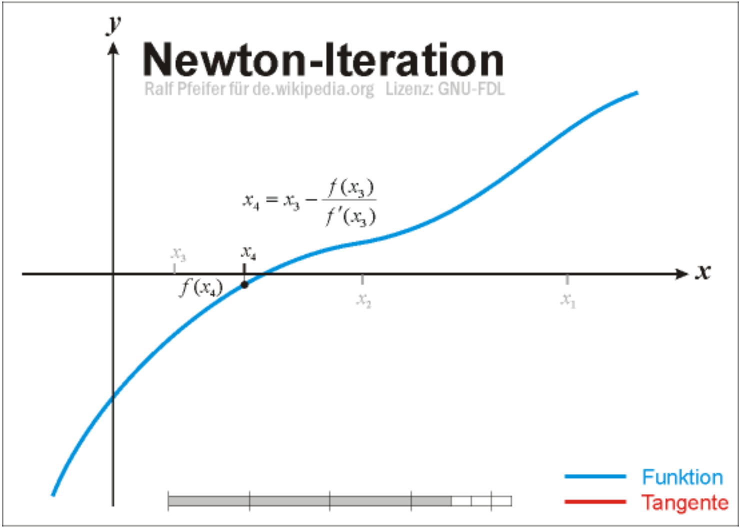
Newton-Raphson Iteration



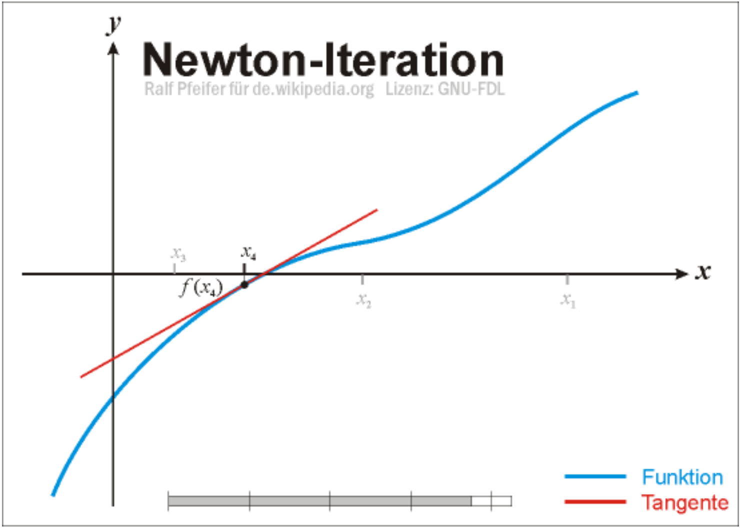
Newton-Raphson Iteration



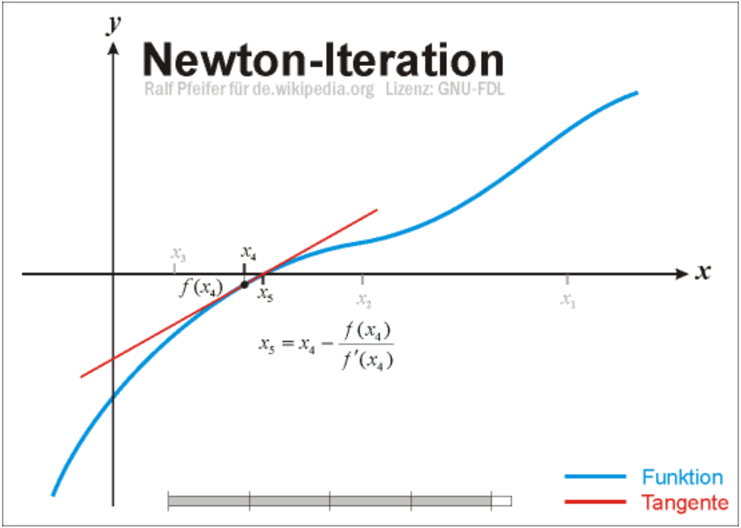
Newton-Raphson Iteration



Newton-Raphson Iteration



Newton-Raphson Iteration



Convergence

Speed

- given a good initial point
- converges quadratically to the solution
  (number of correct decimal points doubles with each iteration)

But

- Convergence is not guaranteed
- Initial point needs to be reasonably close to zero-point

Convergence

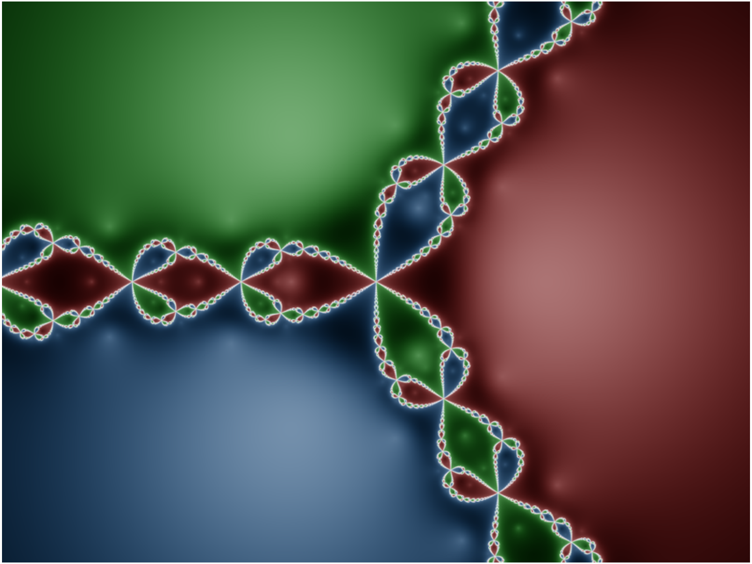
Speed

- given a good initial point
- converges quadratically to the solution
  (number of correct decimal points doubles with each iteration)

But

- Convergence is not guaranteed
- Initial point needs to be reasonably close to zero-point

Convergence



Contents

1 Inside and Outside Weights

2 *n*-best Derivations

3 Cube Pruning

4 Factorization

n-best Derivations

Problem

- Given WTG $G = (Q, \Sigma, I, R)$ and $q \in Q$
- Compute the n highest scoring derivations of D_M^q

Caveats

- Semantics is given by

$$\text{wt}(t) = \sum_{q \in I} \Big(\sum_{d \in D_M^q(t)} \text{wt}(d) \Big)$$

- We disregard the actual input
- We disregard the summation

n-best Derivations

Problem

- Given WTG $G = (Q, \Sigma, I, R)$ and $q \in Q$
- Compute the n highest scoring derivations of D_M^q

Caveats

- Semantics is given by

$$\text{wt}(t) = \sum_{q \in I} \Big(\sum_{d \in D_M^q(t)} \text{wt}(d) \Big)$$

- We disregard the actual input
- We disregard the summation

n-best Derivations

Problem

- Given WTG $G = (Q, \Sigma, I, R)$ and $q \in Q$
- Compute the n highest scoring derivations of D_M^q

Caveats

- Semantics is given by

$$\mathrm{wt}(t) = \sum_{q \in I} \Big(\sum_{d \in D_M^q(t)} \mathrm{wt}(d) \Big)$$

- We disregard the actual input
  Use product construction to restrict to actual input
- We disregard the summation

n-best Derivations

Problem

- Given WTG $G = (Q, \Sigma, I, R)$ and $q \in Q$
- Compute the n highest scoring derivations of D_M^q

Caveats

- Semantics is given by

$$\mathrm{wt}(t) = \sum_{q \in I} \Big(\sum_{d \in D_M^q(t)} \mathrm{wt}(d) \Big)$$

- We disregard the actual input
  Use product construction to restrict to actual input
- We disregard the summation
  No Fix!

Viterbi Algorithm

Algorithm

1. Sort states topologically
2. Select least unprocessed state q
3. Determine its best derivation (based on previous states)
4. Mark q and return to 2.

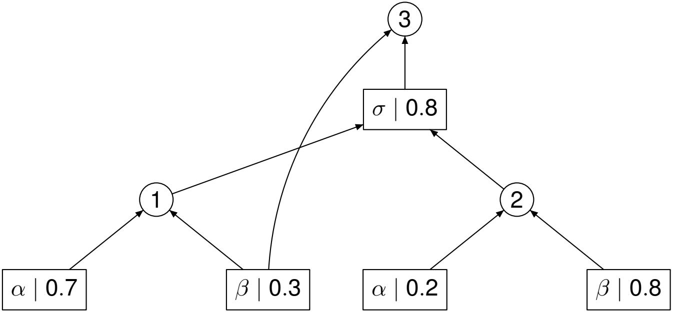
Requirement

- *Optimal substructure property* (dynamic programming)

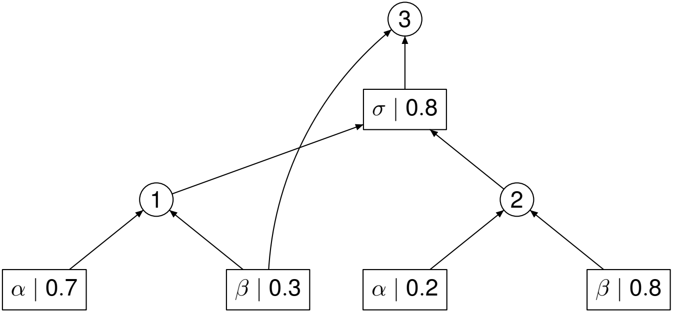
$$a_i \leq b_i \qquad \text{implies} \qquad f(a_1, \ldots, a_k) \leq f(b_1, \ldots, b_k)$$

- in our case: $a_i \leq b_i$ must imply $\prod_{i=1}^{k} a_i \leq \prod_{i=1}^{k} b_i$

Viterbi Algorithm



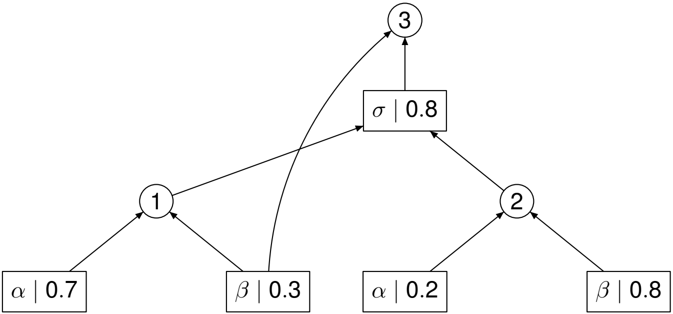
Viterbi Algorithm



Best derivations:

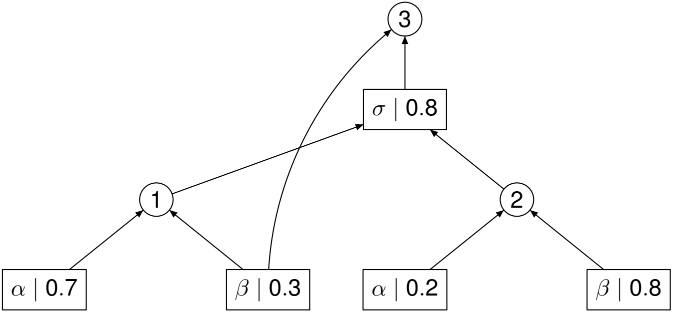
1	α	0.7

Viterbi Algorithm



	1	α	0.7
Best derivations:	2	β	0.8

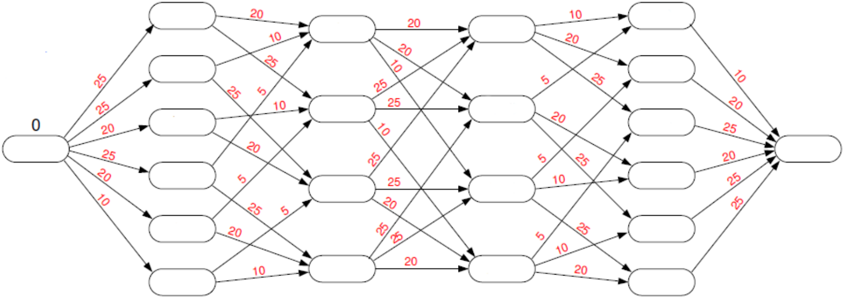
Viterbi Algorithm



Best derivations:

1	α	0.7
2	β	0.8
3	$\sigma(\alpha, \beta)$	$0.8 \cdot 0.7 \cdot 0.8$

Viterbi Algorithm — Exercise



(Taken from KAESLIN: *A gentle introduction to dynamic programming*)

Viterbi Algorithm

Suitable weight structures

- $(\mathbb{R}_{\geq 0}, +, \cdot, 0, 1)$
- $(\mathbb{R}, +, \cdot, 0, 1)$
- $(\{0, 1\}, \max, \min, 0, 1)$
- $(\mathbb{N}, +, \cdot, 0, 1)$
- $([0, 1], \max, \cdot, 0, 1)$

Extension

- Can we handle cyclic WTG?
- additional requirement:

$$f(a_1, \ldots, a_k) \leq a_i \qquad \text{or in our case} \qquad \prod_{i=1}^{k} a_i \leq a_i$$

Viterbi Algorithm

Suitable weight structures

- $(\mathbb{R}_{\geq 0}, +, \cdot, 0, 1)$ YES
- $(\mathbb{R}, +, \cdot, 0, 1)$
- $(\{0, 1\}, \max, \min, 0, 1)$
- $(\mathbb{N}, +, \cdot, 0, 1)$
- $([0, 1], \max, \cdot, 0, 1)$

Extension

- Can we handle cyclic WTG?
- additional requirement:

$$f(a_1, \ldots, a_k) \leq a_i \qquad \text{or in our case} \qquad \prod_{i=1}^{k} a_i \leq a_i$$

Viterbi Algorithm

Suitable weight structures

- $(\mathbb{R}_{\geq 0}, +, \cdot, 0, 1)$ YES
- $(\mathbb{R}, +, \cdot, 0, 1)$ NO
- $(\{0, 1\}, \max, \min, 0, 1)$
- $(\mathbb{N}, +, \cdot, 0, 1)$
- $([0, 1], \max, \cdot, 0, 1)$

Extension

- Can we handle cyclic WTG?
- additional requirement:

$$f(a_1, \ldots, a_k) \leq a_i \qquad \text{or in our case} \qquad \prod_{i=1}^{k} a_i \leq a_i$$

Viterbi Algorithm

Suitable weight structures

- $(\mathbb{R}_{\geq 0}, +, \cdot, 0, 1)$ YES
- $(\mathbb{R}, +, \cdot, 0, 1)$ NO
- $(\{0, 1\}, \max, \min, 0, 1)$ YES
- $(\mathbb{N}, +, \cdot, 0, 1)$
- $([0, 1], \max, \cdot, 0, 1)$

Extension

- Can we handle cyclic WTG?
- additional requirement:

$$f(a_1, \ldots, a_k) \leq a_i \qquad \text{or in our case} \qquad \prod_{i=1}^{k} a_i \leq a_i$$

Viterbi Algorithm

Suitable weight structures

- $(\mathbb{R}_{\geq 0}, +, \cdot, 0, 1)$ YES
- $(\mathbb{R}, +, \cdot, 0, 1)$ NO
- $(\{0, 1\}, \max, \min, 0, 1)$ YES
- $(\mathbb{N}, +, \cdot, 0, 1)$ YES
- $([0, 1], \max, \cdot, 0, 1)$

Extension

- Can we handle cyclic WTG?
- additional requirement:

$$f(a_1, \ldots, a_k) \leq a_i \qquad \text{or in our case} \qquad \prod_{i=1}^{k} a_i \leq a_i$$

Viterbi Algorithm

Suitable weight structures

- $(\mathbb{R}_{\geq 0}, +, \cdot, 0, 1)$ YES
- $(\mathbb{R}, +, \cdot, 0, 1)$ NO
- $(\{0, 1\}, \max, \min, 0, 1)$ YES
- $(\mathbb{N}, +, \cdot, 0, 1)$ YES
- $([0, 1], \max, \cdot, 0, 1)$ YES

Extension

- Can we handle cyclic WTG?
- additional requirement:

$$f(a_1, \ldots, a_k) \leq a_i \qquad \text{or in our case} \qquad \prod_{i=1}^{k} a_i \leq a_i$$

Viterbi Algorithm

Suitable weight structures

- $(\mathbb{R}_{\geq 0}, +, \cdot, 0, 1)$ YES
- $(\mathbb{R}, +, \cdot, 0, 1)$ NO
- $(\{0, 1\}, \max, \min, 0, 1)$ YES
- $(\mathbb{N}, +, \cdot, 0, 1)$ YES
- $([0, 1], \max, \cdot, 0, 1)$ YES

Extension

- Can we handle cyclic WTG?
- additional requirement:

$$f(a_1, \ldots, a_k) \leq a_i \qquad \text{or in our case} \qquad \prod_{i=1}^{k} a_i \leq a_i$$

Viterbi Algorithm

Suitable weight structures

- $(\mathbb{R}_{\geq 0}, +, \cdot, 0, 1)$ YES
- $(\mathbb{R}, +, \cdot, 0, 1)$ NO
- $(\{0, 1\}, \max, \min, 0, 1)$ YES
- $(\mathbb{N}, +, \cdot, 0, 1)$ YES
- $([0, 1], \max, \cdot, 0, 1)$ YES

Extension

- Can we handle cyclic WTG? YES
- additional requirement:

$$f(a_1, \ldots, a_k) \leq a_i \qquad \text{or in our case} \qquad \prod_{i=1}^{k} a_i \leq a_i$$

n-best Algorithm

Observations

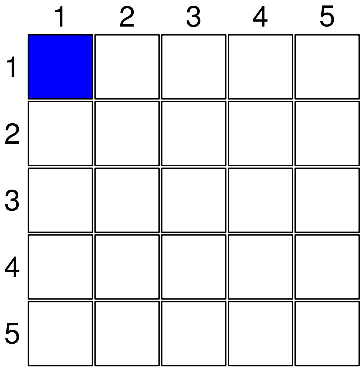
- Best subderivations yield the best derivation
- How do we obtain the 2nd best derivation?

	1	2	3	4	5
1					
2					
3					
4					
5					

n-best Algorithm

Observations

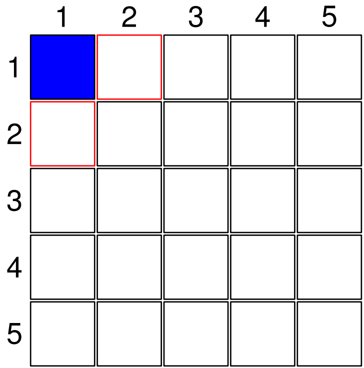
- Best subderivations yield the best derivation
- How do we obtain the 2nd best derivation?



n-best Algorithm

Observations

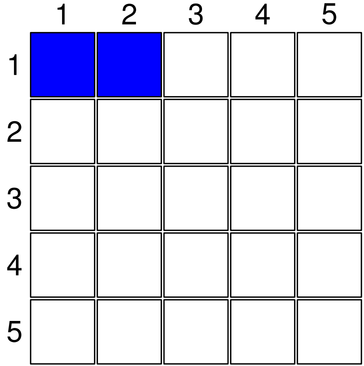
- Best subderivations yield the best derivation
- How do we obtain the 2nd best derivation?



n-best Algorithm

Observations

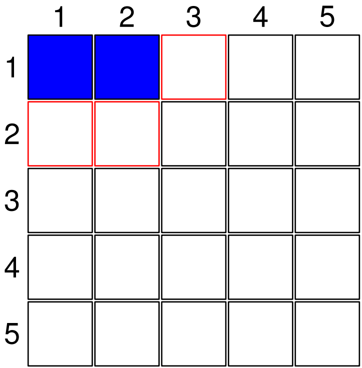
- Best subderivations yield the best derivation
- How do we obtain the 2nd best derivation?



n-best Algorithm

Observations

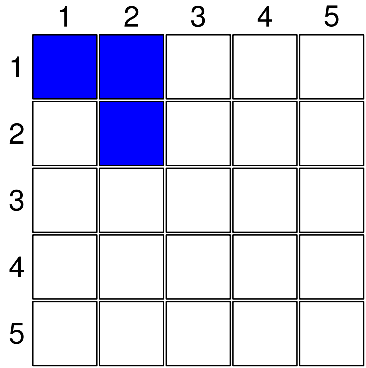
- Best subderivations yield the best derivation
- How do we obtain the 2nd best derivation?



n-best Algorithm

Observations

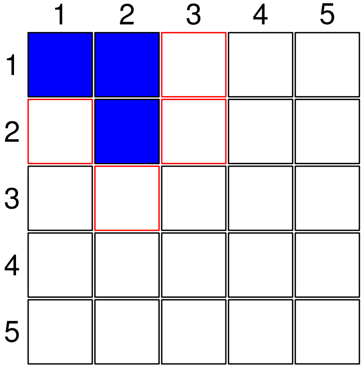
- Best subderivations yield the best derivation
- How do we obtain the 2nd best derivation?



n-best Algorithm

Observations

- Best subderivations yield the best derivation
- How do we obtain the 2nd best derivation?



n-best Algorithm

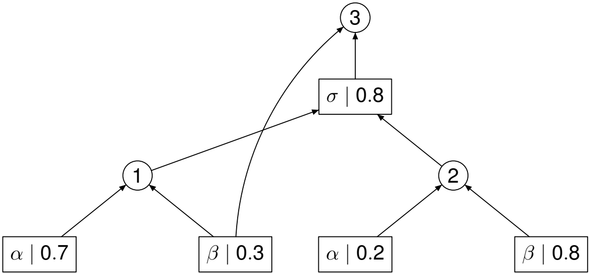
Optimization

lazy computation of values yields

$$O(|E| + |D_{\max}| n \log n)$$

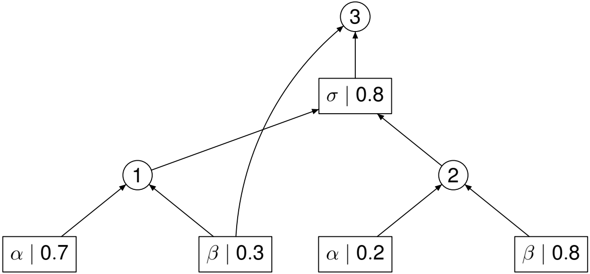
- $|E|$: number of transitions
- $|D_{\max}|$: longest derivation in top n
- n: desired number of derivations

3-best Algorithm



1st derivation: $\sigma(\alpha, \beta)$ with weight $0.448 = 0.8 \cdot 0.7 \cdot 0.8$
2nd derivation:

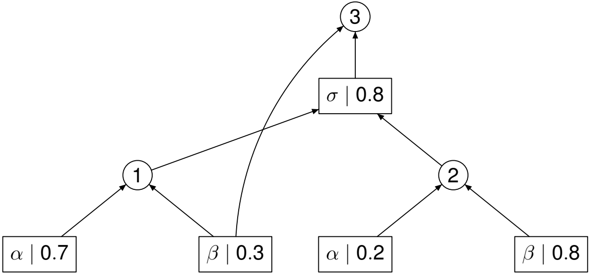
3-best Algorithm



1st derivation: $\sigma(\alpha, \beta)$ with weight $0.448 = 0.8 \cdot 0.7 \cdot 0.8$
2nd derivation:

- best of $\beta \mid 0.3$
- $\sigma \mid 0.8$ with 2nd best from 1 and best from 2
- $\sigma \mid 0.8$ with best from 1 and 2nd best from 2

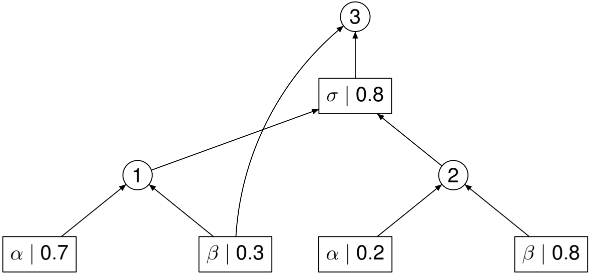
3-best Algorithm



1st derivation: $\sigma(\alpha, \beta)$ with weight $0.448 = 0.8 \cdot 0.7 \cdot 0.8$
2nd derivation:

- best of $\beta \mid 0.3$ 0.3
- $\sigma \mid 0.8$ with 2nd best from 1 and best from 2
- $\sigma \mid 0.8$ with best from 1 and 2nd best from 2

3-best Algorithm

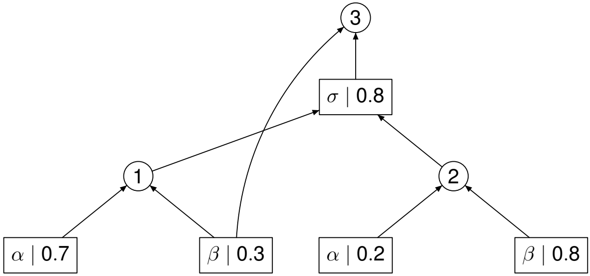


1st derivation: $\sigma(\alpha, \beta)$ with weight $0.448 = 0.8 \cdot 0.7 \cdot 0.8$

2nd derivation:

- best of $\beta \mid 0.3$ — 0.3
- $\sigma \mid 0.8$ with 2nd best from 1 and best from 2 — 0.192
- $\sigma \mid 0.8$ with best from 1 and 2nd best from 2

3-best Algorithm

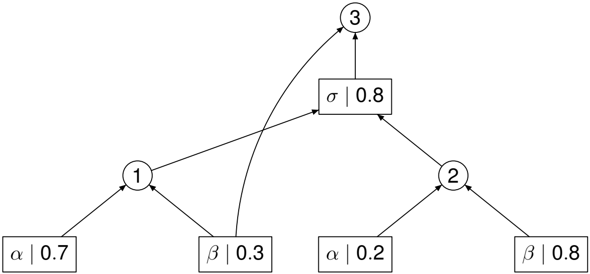


1st derivation: $\sigma(\alpha, \beta)$ with weight $0.448 = 0.8 \cdot 0.7 \cdot 0.8$

2nd derivation:

- best of $\beta \mid 0.3$ — 0.3
- $\sigma \mid 0.8$ with 2nd best from 1 and best from 2 — 0.192
- $\sigma \mid 0.8$ with best from 1 and 2nd best from 2 — 0.112

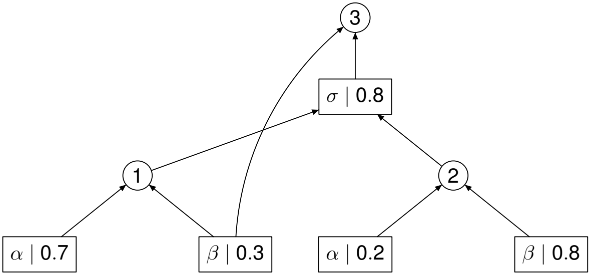
3-best Algorithm



1st derivation: $\sigma(\alpha, \beta)$ with weight $0.448 = 0.8 \cdot 0.7 \cdot 0.8$
2nd derivation: β with weight $0.3 = 0.3$
3rd derivation:

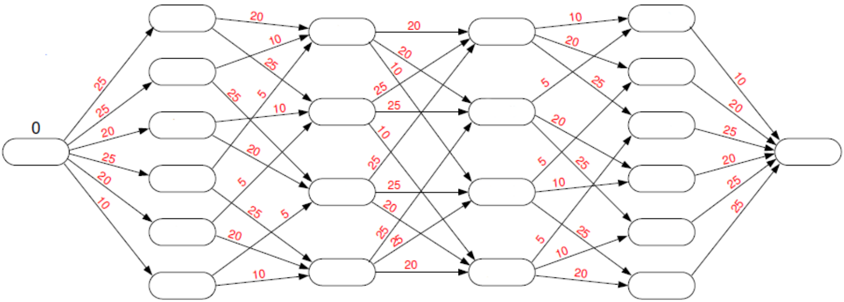
- $\sigma \mid 0.8$ with 2nd best from 1 and best from 2 0.192
- $\sigma \mid 0.8$ with best from 1 and 2nd best from 2 0.112

3-best Algorithm



1st derivation: $\sigma(\alpha, \beta)$ with weight $0.448 = 0.8 \cdot 0.7 \cdot 0.8$
2nd derivation: β with weight $0.3 = 0.3$
3rd derivation: $\sigma(\beta, \beta)$ with weight $0.192 = 0.8^2 \cdot 0.3$

5-best Algorithm — Exercise



Contents

1 Inside and Outside Weights

2 *n*-best Derivations

3 Cube Pruning

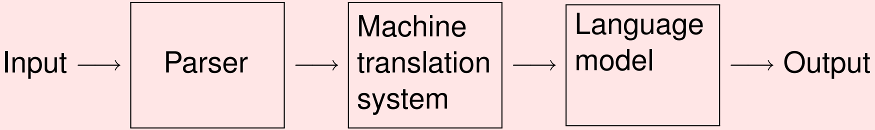
4 Factorization

Cube Pruning

Cube Pruning = n-best + language model

Cube Pruning

Syntax-based systems



Cube Pruning

Syntax-based systems

Input ⟶ Parser ⟶ Machine translation system ⟶ Language model ⟶ Output

Notes

- We can individually compute *n*-best lists
- But that leads to large search errors
- Slightly better: 10^{n+2}-best list and 10^{n}-best list

Cube Pruning

Syntax-based systems

Input ⟶ Parser ⟶ Machine translation system ⟶ Language model ⟶ Output

Notes

- We can individually compute *n*-best lists
- But that leads to large search errors
- Slightly better: 10^{n+2}-best list and 10^{n}-best list

Cube Pruning

Syntax-based systems

Input ⟶ Parser ⟶ Machine translation system ⟶ Language model ⟶ Output

Alternatives

- Full integration (Intersection)
- Pruning and beam search
- Cube pruning

Cube Pruning

Syntax-based systems

Input ⟶ Parser ⟶ Machine translation system ⟶ Language model ⟶ Output

Alternatives

- Full integration (Intersection) — too expensive
- Pruning and beam search
- Cube pruning

Cube Pruning

Syntax-based systems

Input ⟶ Parser ⟶ Machine translation system ⟶ Language model ⟶ Output

Alternatives

- Full integration (Intersection) — too expensive
- Pruning and beam search
- Cube pruning

Cube Pruning

Syntax-based systems

Input ⟶ Parser ⟶ Machine translation system ⟶ Language model ⟶ Output

Alternatives

- Full integration (Intersection) — too expensive
- Pruning and beam search — too bad
- Cube pruning

Cube Pruning

Syntax-based systems

Input → Parser → Machine translation system → Language model → Output

Alternatives

- Full integration (Intersection) — too expensive
- Pruning and beam search — too bad
- Cube pruning

Cube Pruning

Algorithm

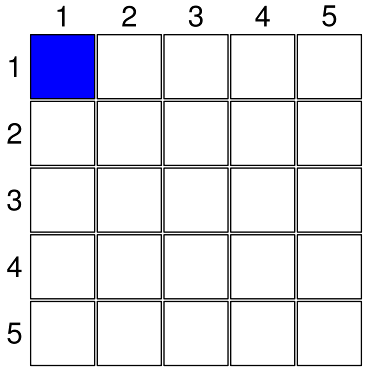
- Proceed as for *n*-best
- but multiply the language model score in the end

	1	2	3	4	5
1					
2					
3					
4					
5					

Cube Pruning

Algorithm

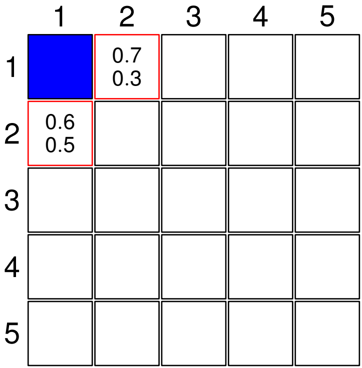
- Proceed as for *n*-best
- but multiply the language model score in the end



Cube Pruning

Algorithm

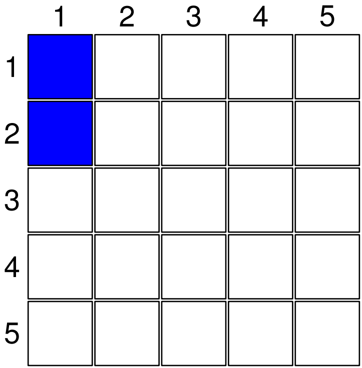
- Proceed as for *n*-best
- but multiply the language model score in the end



Cube Pruning

Algorithm

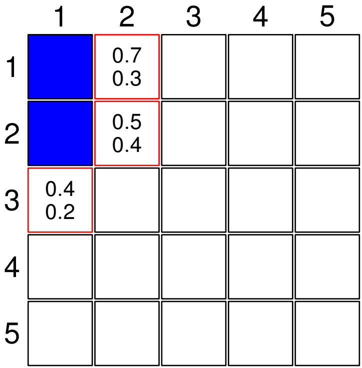
- Proceed as for *n*-best
- but multiply the language model score in the end



Cube Pruning

Algorithm

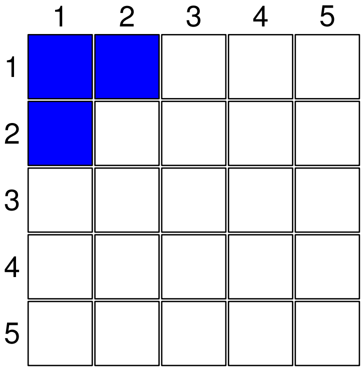
- Proceed as for *n*-best
- but multiply the language model score in the end



Cube Pruning

Algorithm

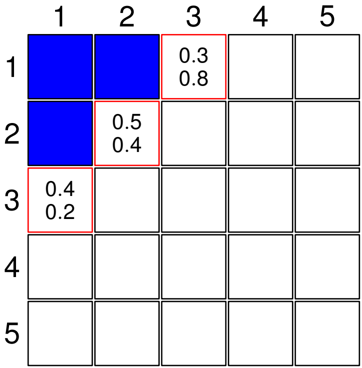
- Proceed as for *n*-best
- but multiply the language model score in the end



Cube Pruning

Algorithm

- Proceed as for *n*-best
- but multiply the language model score in the end



Cube Pruning

Observations

- retains the efficiency of n-best derivation
- output list not sorted
- no guarantees

Contents

1 Inside and Outside Weights

2 *n*-best Derivations

3 Cube Pruning

4 Factorization

Factorization

Motivation

- rk(M) essential part in the complexity of xtt operations [like input restriction]

$$O(|R| \cdot \text{len}(M) \cdot |P|^{2\,\text{rk}(M)+5})$$

Factorization

- reduce rk(M) by decomposing rules
- maximal decomposition prefered

Factorization

References

- ZHANG, HUANG, GILDEA, KNIGHT: Synchronous binarization for machine translation. HLT-NAACL 2006 [for SCFG]
- GILDEA, SATTA, ZHANG: Factoring synchronous grammars by sorting. CoLing/ACL 2006 [for SCFG]
- NESSON, SATTA, SHIEBER: Optimal *k*-arization of synchronous tree-adjoining grammar. ACL 2008 [for STAG]
- GILDEA: Optimal parsing strategies for linear context-free rewriting systems. HLT-NAACL 2010 [for LCFRS]

Maximal Factorization

Common advertisement slogan

- Optimal k-arization
- Optimal parsing strategy

But

- factorization is just one way to reduce $\text{rk}(M)$
- obtained “optimal” rank is not optimal for the given transformation
- $\text{rk}(M)$ is just one parameter to the parsing complexity

Conclusion

- optimal k-arization $\neq$ maximal factorization
- optimal parsing strategy $\neq$ maximal factorization

Maximal Factorization

Common advertisement slogan

- Optimal k-arization
- Optimal parsing strategy

But

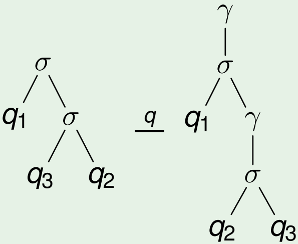
- factorization is just one way to reduce rk(M)
- obtained “optimal” rank is not optimal for the given transformation
- rk(M) is just one parameter to the parsing complexity

Conclusion

- optimal k-arization $\neq$ maximal factorization
- optimal parsing strategy $\neq$ maximal factorization

Construction

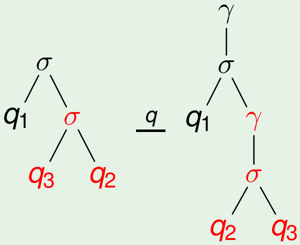
Example



Constructed rules

Construction

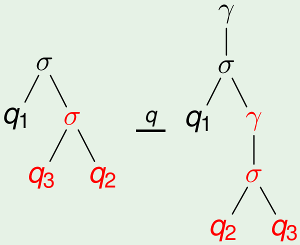
Example



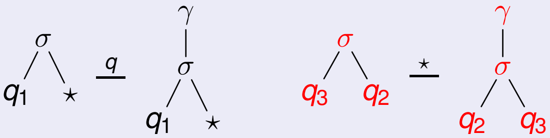
Constructed rules

Construction

Example



Constructed rules



Construction

Notes

- runs in linear time
- returns maximal (meaningful) factorization
- returned XTOP is rank-optimal (among all factorizations)

Construction

Notes

- runs in linear time
- returns maximal (meaningful) factorization
- returned XTOP is rank-optimal (among all factorizations)

More Notes

- Factorization is strongly related to rule extraction
- Rule extraction always yields rank-optimal XTOP

References

- HUANG, CHIANG: *Better k-best parsing.* IWPT 2005
- CHIANG: *Hierarchical Phrase-Based Translation.* Computat. Linguist. 33, 2007
- MAY, KNIGHT: TIBURON — *a weighted tree automata toolkit.* CIAA 2006

References

- HUANG, CHIANG: *Better k-best parsing.* IWPT 2005
- CHIANG: *Hierarchical Phrase-Based Translation.* Computat. Linguist. 33, 2007
- MAY, KNIGHT: TIBURON — *a weighted tree automata toolkit.* CIAA 2006

Thank you for your attention!

Questions?