ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ
ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS

Αλληλεπίδραση Ανθρώπου-Υπολογιστή

Β2. Γλωσσικά μοντέλα n-γραμμάτων, διόρθωση και πρόβλεψη κειμένων

(2023-24)

Ίων Ανδρουτσόπουλος

http://www.aueb.gr/users/ion/

Οι διαφάνειες αυτές βασίζονται εν μέρει σε ύλη του βιβλίου *Speech and Language Processing* των D. Jurafsky και J.H. Martin, 2η έκδοση, Pearson Education, 2009.

Τι θα ακούσετε

- **Γλωσσικά μοντέλα** *n*-γραμμάτων.
- **Απόσταση διόρθωσης** (edit distance).
- **Διόρθωση ορθογραφικών** λαθών.
- **Έξυπνα πληκτρολόγια** (π.χ. κινητών).

Γλωσσικά μοντέλα

- **Πόσο πιθανό** είναι να συναντήσουμε (π.χ. σε κείμενα εφημερίδων) τις ακολουθίες λέξεων:
  - «Η κυβέρνηση γνώρισε από τις ενάρξεις του 2009 ότι η κατάσταση αντιμετώπισε το φάσμα της χρεοκοπίας.»
  - «Η κυβέρνηση γνώριζε από τις ενάρξεις του 2009 ότι η χώρα αντιμετώπιζε το φάσμα της χρεοκοπίας.»
  - «Η κυβέρνηση γνώριζε από τις αρχές του 2009 ότι η χώρα αντιμετώπιζε το φάσμα της χρεοκοπίας.»
- Σε πολλές εφαρμογές παράγονται **υποψήφιες προτάσεις**. Θέλουμε να κρατήσουμε τις **πιο πιθανές**.
  - Μηχανική μετάφραση, αναγνώριση ομιλίας, οπτική αναγνώριση χαρακτήρων, **πληκτρολόγηση σε κινητά**, ορθογραφικός έλεγχος, κανονικοποίηση/διόρθωση κειμένων κοινωνικών δικτύων…

Μοντέλα *n*-γραμμάτων

- Συμβολισμός για ακολουθία λέξεων:

$$\left\langle w_1, w_2, \ldots, w_k \right\rangle = w_1^k$$

- ***n*-γραμμα**: ακολουθία ***n* συνεχόμενων λέξεων**.
  - 3-γράμματα: «η κυβέρνηση γνώρισε», «κυβέρνηση γνώρισε από», «γνώρισε από τις»,,...
  - 2-γράμματα: «η κυβέρνηση», «κυβέρνηση γνώρισε», «γνώρισε από», «από τις», «τις ενάρξεις», ...
  - Σε άλλες περιπτώσεις, *n*-γράμματα **χαρακτήρων**.
- Κανόνας **αλυσίδας**:

$$P(w_1^k) = P(w_1, \ldots, w_k) = P(w_1) \cdot P(w_2 \mid w_1) \cdot$$
$$P(w_3 \mid w_1, w_2) \cdot P(w_4 \mid w_1^3) \cdot \ldots \cdot P(w_k \mid w_1^{k-1})$$

Μοντέλα *n*-γραμμάτων – συνέχεια

- Εκτιμήσεις **μεγίστης πιθανοφάνειας** π.χ. από **σώμα κειμένων** με συνολικά *C* **εμφανίσεις** (tokens) **λέξεων**:

$$P_{MLE}(\text{η}) = \frac{c(\text{η})}{C} \quad P_{MLE}(\text{κυβέρνηση} \mid \text{η}) = \frac{c(\text{η}, \text{κυβέρνηση})}{c(\text{η})}$$

$$P_{MLE}(\text{γνώρισε} \mid \text{η}, \text{κυβέρνηση}) = \frac{c(\text{η}, \text{κυβέρνηση}, \text{γνώρισε})}{c(\text{η}, \text{κυβέρνηση})}$$

- Πολλά 3-γράμματα και ακόμα περισσότερα 4-γράμματα, 5-γράμματα κλπ. θα είναι **σπάνια** ή δεν θα τα έχουμε **ξανασυναντήσει ποτέ**, ακόμα και σε μεγάλο σώμα.
  - Πολύ **κακές** ή **μηδενικές** εκτιμήσεις πιθανοτήτων.
  - **Μηδενίζουν** και το **γινόμενο** της αλυσίδας.

Υπόθεση Markov

- Αν θέλουμε να χρησιμοποιήσουμε **2-γράμματα**:

$$P(w_1^k) = P(w_1, \ldots, w_k) = P(w_1) \cdot P(w_2 \mid w_1) \cdot$$

$$P(w_3 \mid w_1, w_2) \cdot P(w_4 \mid w_1^3) \cdot \ldots \cdot P(w_k \mid w_1^{k-1}) \simeq$$

$$P(w_1 \mid start) \cdot P(w_2 \mid w_1) \cdot P(w_3 \mid w_2) \cdot \ldots \cdot P(w_k \mid w_{k-1})$$

- Αν θέλουμε να χρησιμοποιήσουμε **3-γράμματα**:

$$P(w_1 \mid start_1, start_2) \cdot P(w_2 \mid start_2, w_1) \cdot P(w_3 \mid w_1, w_2) \cdot$$

$$P(w_4 \mid w_2, w_3) \cdot \ldots \cdot P(w_k \mid w_{k-2}, w_{k-1})$$

- Κάνουμε επίσης την παραδοχή ότι οι πιθανότητες δεν εξαρτώνται από το πού συναντούμε τα *n*-γράμματα (**στασιμότητα**). Π.χ. όταν εκτιμούμε την *P*(γνώρισε | η, κυβέρνηση) από ένα σώμα κειμένων, δεν εξετάζουμε αν το «γνώρισε» εμφανίζεται ως 3η ή 4η ή … λέξη στις προτάσεις του σώματος.
- Κανονικά χρειαζόμαστε και μια ψευτο-λέξη ***end***. Βλ. ασκήσεις μελέτης.

Εξομάλυνση Laplace

- Ακόμα και για μικρά *n*, **πολλά *n*-γράμματα δεν θα υπάρχουν** στο σώμα κειμένων.
- **Εξομάλυνση Laplace**, αν υπάρχουν συνολικά **|V|** δυνατές **διαφορετικές λέξεις** (όχι εμφανίσεις λέξεων):

$$P_{Laplace}(W = w) = \frac{c(w) + 1}{C + |V|}$$

Προσθέτουμε μία ψευτο-εμφάνιση κάθε δυνατής διαφορετικής λέξης *w*, γενικότερα κάθε δυνατής τιμής της τυχαίας μεταβλητής (εδώ *W*).

- Ομοίως, π.χ. για **3-γράμματα**:

$$P_{Laplace}(W = w_k \mid w_{k-2}, w_{k-1}) = \frac{c(w_{k-2}, w_{k-1}, w_k) + 1}{c(w_{k-2}, w_{k-1}) + |V|}$$

Προσθέτουμε μία εμφάνιση κάθε δυνατού 3-γράμματος που ξεκινά με w_{k-2}, w_{k-1}. Υπάρχουν συνολικά $|V|$ τέτοια 3-γράμματα.

- Αλλά καταλήγουμε να δίνουμε πολλή μάζα πιθανότητας σε σπάνια διγράμματα, τριγράμματα, …

Add-α smoothing

- For **unigrams:** if we have $|V|$ vocabulary words,

$$P_{Laplace}(W = w) = \frac{c(w) + \alpha}{C + \alpha \cdot |V|}$$

We tune a ($0 \leq a \leq 1$) on development data (see below).

- Similarly, e.g., for **trigrams**:

$$P_{Laplace}(W_k = w_k \mid w_{k-2}, w_{k-1}) = \frac{c(w_{k-2}, w_{k-1}, w_k) + \alpha}{c(w_{k-2}, w_{k-1}) + \alpha \cdot |V|}$$

- Better, but still **poor estimates** for language models.
  - In practice, Laplace and add-α smoothing are **not used in language models** (but often work well in classification tasks).
  - See **references slide for better estimates** for *n*-gram LMs (e.g., Knesser-Ney smoothing, backoff models).

LMs as next word predictors

- **Sequence probability** using a bigram LM:

$$P(w_1^k) = P(w_1, \ldots, w_k) = P(w_1) \cdot P(w_2 \mid w_1) \cdot$$
$$P(w_3 \mid w_1, w_2) \cdot P(w_4 \mid w_1^3) \cdots P(w_k \mid w_1^{k-1}) \simeq$$
$$P(w_1 \mid start) \cdot P(w_2 \mid w_1) \cdot P(w_3 \mid w_2) \cdots P(w_k \mid w_{k-1})$$

- We can think of the **LM** as a system that **provides the probabilities $P(w_i|w_{i-1})$**, which we then multiply.
  - Or the probabilities $P(w_i|w_{i-2}, w_{i-1})$ for a **trigram LM**.
  - Or the probabilities $P(w_i|h)$ for an LM that considers all the **"history" (previous words) h**, e.g., in an **RNN LM**.
- An **LM** typically provides a **distribution $P(w|h)$** showing how probable it is for **every word** $w \in V$ to be the next one.

Διόρθωση ορθογραφικών λαθών

- **Δύο είδη λαθών**: προκύπτουν **λέξεις** που **<u>είναι</u> ή <u>όχι</u>** λέξεις της **γλώσσας**.
  - 1ο είδος: «εσείς παίζετε» → «εσείς παίζετει».
  - 2ο είδος: «εσείς παίζετε» → «εσείς παίζεται».
- Ας περιοριστούμε **αρχικά** σε λάθη του **1ου είδους**.
  - Λέξεις που **δεν υπάρχουν** σε ένα μεγάλο **λεξικό** ή που είναι **πολύ σπάνιες** σε ένα μεγάλο **σώμα** (κυρίως σωστών) **κειμένων**.
- **Για κάθε μία λανθασμένη** λέξη, **υποψήφιες διορθώσεις**:
  - **Λέξεις της γλώσσας** (σε λεξικό ή συχνές σε σώμα κειμένων) με **μικρή απόσταση διόρθωσης** από τη λανθασμένη.
  - Στην πιο απλή περίπτωση, **απόσταση Levenshtein**.

Διόρθωση λαθών 1ου είδους

- Οι λέξεις που βλέπουμε:

  w_1^k: Εσείς παίζετει καλύ μπάλα.

- Ενδεχόμενες σωστές λέξεις:

  t_1^k: Εσείς παίζετε καλό μπάλα.

  t_1^k: Εσείς παίζετε καλά μπάλα.

  t_1^k: Εσείς παίζετε καλή μπάλα.

  t_1^k: Εσείς παίζεται καλό μπάλα.

  t_1^k: Εσείς παίζεται καλά μπάλα.

  t_1^k: Εσείς παίζεται καλή μπάλα.

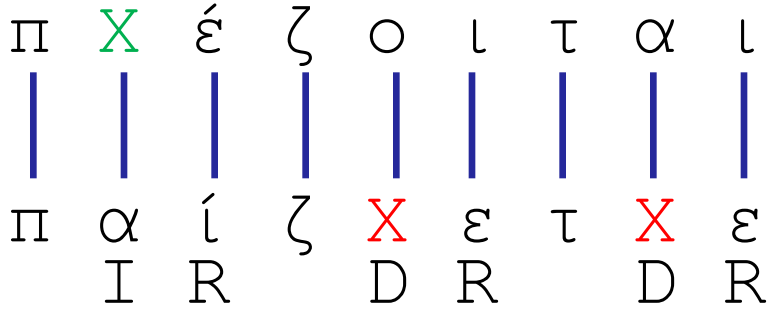
  …

Οι **πράσινες λέξεις** είναι λέξεις του **λεξικού** με **μικρή απόσταση** (π.χ. Levenshtein) από τις κόκκινες (λανθασμένες).

Ένα **γλωσσικό μοντέλο** θα εκτιμά **πόσο ταιριάζουν** οι λέξεις μεταξύ τους.

Απόσταση διόρθωσης

- Είσοδος: **δύο συμβολοσειρές** (π.χ. λέξη tweet και λεξικού).
- Ποιο το **συνολικό ελάχιστο κόστος** εφαρμογών **τελεστών** για να **μετατραπεί** η **μία** στην **άλλη**.
- Απόσταση **Levenshtein**:
  - Τελεστές: **εισαγωγή** (I, κόστος **1**), **διαγραφή** (D, κόστος **1**), **αντικατάσταση** (R, κόστος **2**). Άλλοι θεωρούν κόστος R = 1.
  - Συχνά λέγοντας «**απόσταση διόρθωσης**» (edit distance) εννοούμε **απόσταση Levenshtein**, αλλά υπάρχουν κι άλλες.
  - Αν π.χ. μετατρέπουμε **Greeklish σε Ελληνικά**, ίσως θέλουμε να θέσουμε $R(w, \omega) < R(w, \pi)$.



Παράγεται και **ευθυγράμμιση** (alignment) των γραμμάτων. Ομοίως μπορούμε να ευθυγραμμίσουμε τις λέξεις δύο προτάσεων.

Υπολογισμός απόστασης Levenshtein

- Πώς μπορώ να **μετατρέψω** το:

  πέζοι σε *παίζω*

  βασιζόμενος (αναδρομικά) σε **μικρότερες κατά ένα τελικό χαρακτήρα** μορφές του *πέζοι* ή/και *παίζω;*

- 1ος τρόπος: **Σβήνω το τελευταίο γράμμα** (Del) του *πέζοι* και μετατρέπω το *πέζο* σε *παίζω*.

  πέζο ι → *παίζω*

  Del(*ι*) + cost(*πέζο*, *παίζω*)

- 2ος τρόπος: Μετατρέπω το *πέζοι* σε *παίζ* και προσθέτω *ω* στο τέλος του *παίζ*.

  πέζοι → *παίζ ω*

  cost(*πέζοι*, *παίζ*) + Ins(*ω*)

Υπολογισμός απόστασης Levenshtein (II)

- 3ος τρόπος: Μετατρέπω το *πέζο* σε *παίζ* και αντικαθιστώ το *ι* με *ω*.

πέζο (*ι*) → *παίζ* (*ω*)

cost(*πέζο*, *παίζ*) + Rep(*ι*, *ω*)

- **Ποιος τρόπος** είναι **καλύτερος**;
  - Αυτός που έχει το **μικρότερο συνολικά κόστος**.
  - **Σε κάθε βήμα** του αλγορίθμου (βλ. παρακάτω) επιλέγω τον **τρόπο με το μικρότερο κόστος**.

Υπολογισμός απόστασης Levenshtein

	#	**π**	**α**	**ί**	**ζ**	**ε**	**τ**	**ε**
#	0	1	2	3	4	5	6	7
π	1							
έ	2							
ζ	3							
ο	4							
ι	5							
τ	6							
α	7							
ι	8							

↑ Del (+1)

← Ins (+1)

Πόσο είναι το (ελάχιστο) **κόστος μετατροπής** του «#» σε «#**παί**»;

Όσο το **κόστος μετατροπής** του «#» σε «#**πα**», συν το **κόστος εισαγωγής** του «**ί**».

Πόσο είναι το (ελάχιστο) **κόστος μετατροπής** του «#**πέζ**» σε «#»;

Όσο το **κόστος διαγραφής** του «**ζ**» συν το κόστος **μετατροπής** του «#**πέ**» σε «#».

Υπολογισμός απόστασης Levenshtein

	#	**π**	**α**	**ί**	**ζ**	**ε**	**τ**	**ε**
#	0	1	2	3	4	5	6	7
π	1	0						
έ	2							
ζ	3							
ο	4							
ι	5							
τ	6							
α	7							
ι	8							

1+1=2
0+0=0
1+1=2

↑ Del (+1)

← Ins (+1)

↖ Rep (+2, ή 0 για ίδιο γράμμα)

Πόσο είναι το (ελάχιστο) **κόστος μετατροπής** του «#**π**» σε «#**π**»;

Μπορώ να **σβήσω** το «**π**» της **αρχικής** ακολουθίας και να **μετατρέψω** το «#» που **απομένει** σε «#**π**».

Μπορώ να **μετατρέψω** το «#**π**» της αρχικής ακολουθίας σε «#» και να **προσθέσω** «**π**» στην ακολουθία που **προκύπτει**.

Μπορώ να **μετατρέψω** το «#» της **αρχικής** ακολουθίας σε «#» και να **αντικαταστήσω** το «**π**» της **αρχικής** με (πάλι) «**π**».

Υπολογισμός απόστασης Levenshtein

	#	**π**	**α**	**ί**	**ζ**	**ε**	**τ**	**ε**
#	0	1	2	3	4	5	6	7
π	1	0	1					
έ	2							
ζ	3							
ο	4							
ι	5							
τ	6							
α	7							
ι	8							

2+1=3

1+2=3

0+1=1

↑ Del (+1)

← Ins (+1)

↖ Rep (+2, ή 0 για ίδιο γράμμα)

Πόσο είναι το (ελάχιστο) **κόστος μετατροπής** του «#**π**» σε «#**πα**»;

Μπορώ να **σβήσω** το «**π**» της **αρχικής** ακολουθίας και να **μετατρέψω** το «#» που **απομένει** σε «#**πα**».

Μπορώ να **μετατρέψω** το «#**π**» της αρχικής ακολουθίας σε «#**π**» και να **προσθέσω** «**α**» στην ακολουθία που **προκύπτει**.

Μπορώ να **μετατρέψω** το «#» της **αρχικής** ακολουθίας σε «#**π**» και να **αντικαταστήσω** το «**π**» της **αρχικής** με «**α**».

Υπολογισμός απόστασης Levenshtein

	#	**π**	**α**	**ί**	**ζ**	**ε**	**τ**	**ε**
#	0	1	2	3	4	5	6	7
π	1	0	1	2	3	4	5	6
έ	2	1	2	3	4	3	4	5
ζ	3	2	3	4	3	4	5	6
ο	4	3	4	5	4	5	6	7
ι	5	4	5	4	5	6	7	8
τ	6	5	6	5	6	7	6	7
α	7	6	5	6	7	8	7	8
ι	8	7	6	5	6	7	8	**9**

↑ Del (+1)

← Ins (+1)

↖ Rep (+2, ή 0 για ίδιο γράμμα)

Μία από τις δυνατές ευθυγραμμίσεις.

Τα βέλη εδώ δείχνουν τους γείτονες από τους οποίους μπορεί ένα κελί να κληρονομήσει την **καλύτερη** τιμή του.

Διόρθωση λαθών 1ου είδους

- Οι λέξεις που βλέπουμε:

  w_1^k: Εσείς παίζετει καλύ μπάλα.

- Ενδεχόμενες σωστές λέξεις:

  t_1^k: Εσείς παίζετε καλό μπάλα.

  t_1^k: Εσείς παίζετε καλά μπάλα.

  t_1^k: Εσείς παίζετε καλή μπάλα.

  t_1^k: Εσείς παίζεται καλό μπάλα.

  t_1^k: Εσείς παίζεται καλά μπάλα.

  t_1^k: Εσείς παίζεται καλή μπάλα.

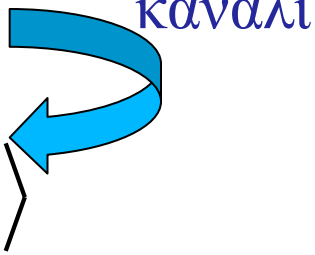
  …

Οι **πράσινες λέξεις** είναι λέξεις του **λεξικού** με **μικρή απόσταση** (π.χ. Levenshtein) από τις κόκκινες (λανθασμένες).

Ένα **γλωσσικό μοντέλο** θα εκτιμά **πόσο ταιριάζουν** οι λέξεις μεταξύ τους.

Θορυβώδες κανάλι (noisy channel)

- Θεωρούμε ότι οι **λέξεις** ήταν **αρχικά σωστές**, αλλά μας **μεταδόθηκαν** μέσω ενός **θορυβώδους καναλιού**.
  - Το κανάλι εισήγαγε (εδώ ορθογραφικές) **παραμορφώσεις** (εισαγωγή χαρακτήρα, διαγραφή χαρακτήρα κλπ).
- Προσπαθούμε να **μαντέψουμε τις αρχικές** (σωστές) λέξεις από τις **παρατηρούμενες**.
  - Αρχικές (σωστές) λέξεις: $t_1^k = \langle t_1, t_2, \ldots, t_k \rangle$
  - Παρατηρούμενες λέξεις: $w_1^k = \langle w_1, w_2, \ldots, w_k \rangle$



- Θέλουμε τις **πιο πιθανές αρχικές λέξεις**:

$$\hat{t}_1^k = \arg\max_{t_1^k} P(t_1^k \mid w_1^k) = \arg\max_{t_1^k} \frac{P(t_1^k) \cdot P(w_1^k \mid t_1^k)}{\cancel{P(w_1^k)}}$$

Οι πιο πιθανές αρχικές λέξεις

- Έστω ότι εξετάζουμε τις λέξεις μιας **πρότασης**:

$$\hat{t}_1^k = \arg\max_{t_1^k} P(t_1^k \mid w_1^k) = \arg\max_{t_1^k} P(t_1^k) \cdot P(w_1^k \mid t_1^k)$$

Σε κάθε **υποψήφια ακολουθία αρχικών λέξεων t_1^k**, κάθε **λανθασμένη λέξη** της w_1^k έχει αντικατασταθεί από μία **λέξη της γλώσσας** με **μικρή απόσταση** από τη λανθασμένη.

Γλωσσικό μοντέλο (π.χ. 3-γραμμάτων).

Π.χ. πιθανότητες αντιστρ. ανάλογες της **απόστασης διόρθωσης** (με κανονικοποίηση).

$$P(w_1^k \mid t_1^k) = P(w_1 \mid t_1^k) \cdot P(w_2 \mid w_1, t_1^k) \cdot \ldots P(w_k \mid w_1^{k-1}, t_1^k)$$

$$\simeq P(w_1 \mid t_1) \cdot P(w_2 \mid t_2) \cdot \ldots P(w_k \mid t_k) = \prod_{i=1}^{k} P(w_i \mid t_i)$$

- Θεωρούμε ότι η πιθανότητα εμφάνισης μιας παρατηρούμενης λέξης **εξαρτάται μόνο** από την **αντίστοιχη αρχική** λέξη.

Διόρθωση λαθών και των δύο τύπων

- Οι λέξεις που βλέπουμε:

  w_1^k: He pls god ftball.

- Υποψήφιες ακολουθίες:

  t_1^k: He please god football.

  t_1^k: He plays god football.

  t_1^k: He plays good football.

  t_1^k: Her players good football.

  …

  t_1^k: Her pleases god ball.

Τώρα **αντικαθιστούμε κάθε λέξη** (ακόμα και λέξεις του λεξικού) με άλλες **κοντινές λέξεις του λεξικού** (ή την ίδια λέξη).

Πάλι, ένα **γλωσσικό μοντέλο** εκτιμά **πόσο ταιριάζουν** οι λέξεις μεταξύ τους.

Γενίκευση για λάθη 2ου τύπου

- Θεωρούμε τώρα ότι **κάθε παρατηρούμενη λέξη** είναι **ενδεχομένως λανθασμένη**.

$$\hat{t}_1^k = \arg\max_{t_1^k} P(t_1^k \mid w_1^k) = \arg\max_{t_1^k} P(t_1^k) \cdot P(w_1^k \mid t_1^k)$$

Γλωσσικό μοντέλο (π.χ. 3-γραμμάτων).

Σε κάθε **υποψήφια ακολουθία αρχικών λέξεων** t_1^k, **κάθε παρατηρούμενη λέξη** έχει ενδεχομένως αντικατασταθεί από μία **λέξη του λεξικού** με **μικρή απόσταση** από την παρατηρούμενη.

$$P(w_1^k \mid t_1^k) \simeq \prod_{i=1}^{k} P(w_i \mid t_i)$$

Π.χ. πιθανότητες αντιστρόφως ανάλογες της **απόστασης διόρθωσης** (με κανονικοποίηση).

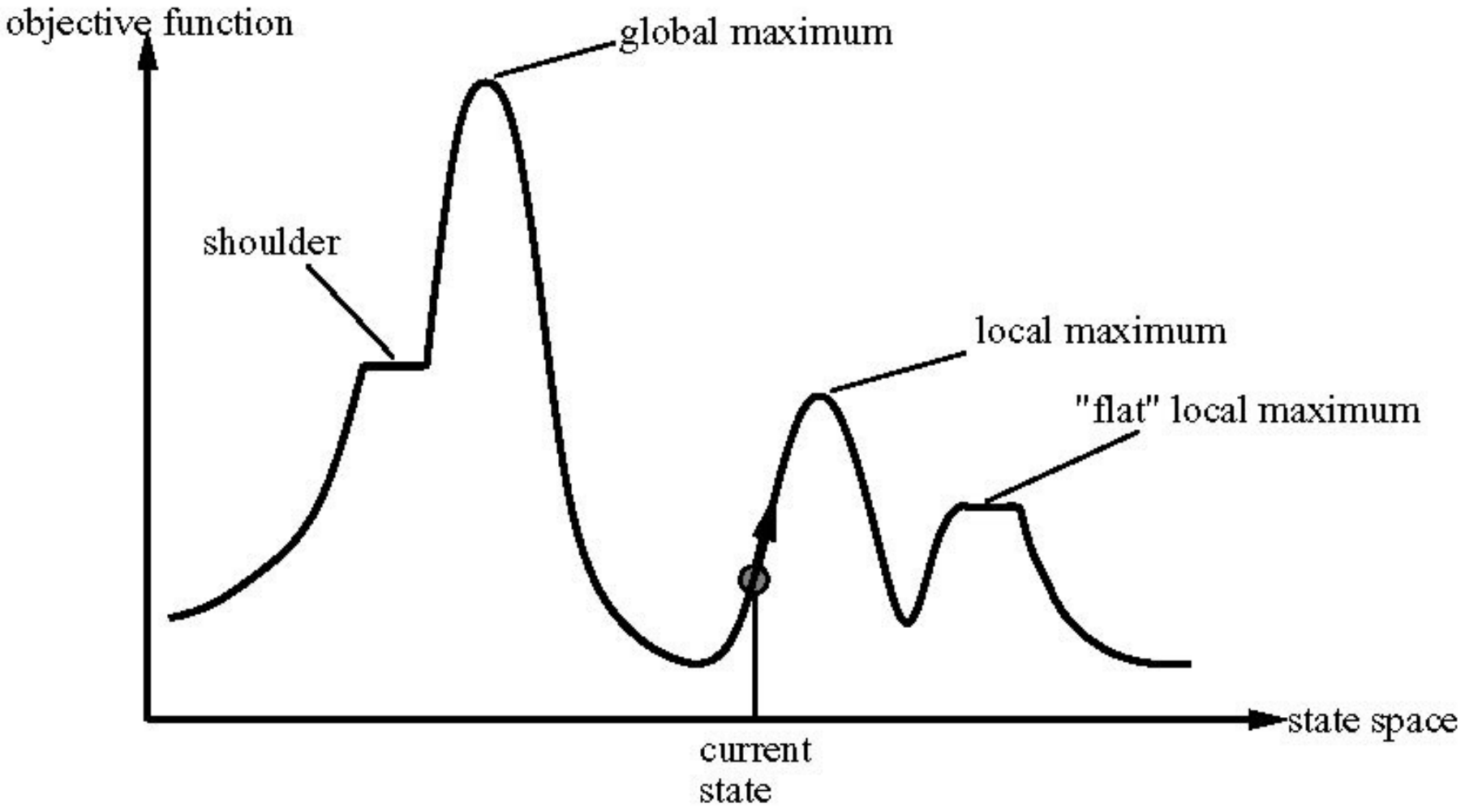
- Η **εύρεση της βέλτιστης ακολουθίας** $t_1, \ldots, t_k$ είναι πρόβλημα **αναζήτησης**. Μπορούμε να χρησιμοποιήσουμε αλγορίθμους αναζήτησης (βλ. μάθημα ΤΝ) ή δυναμικό προγραμματισμό.

Hill climbing search (HC)

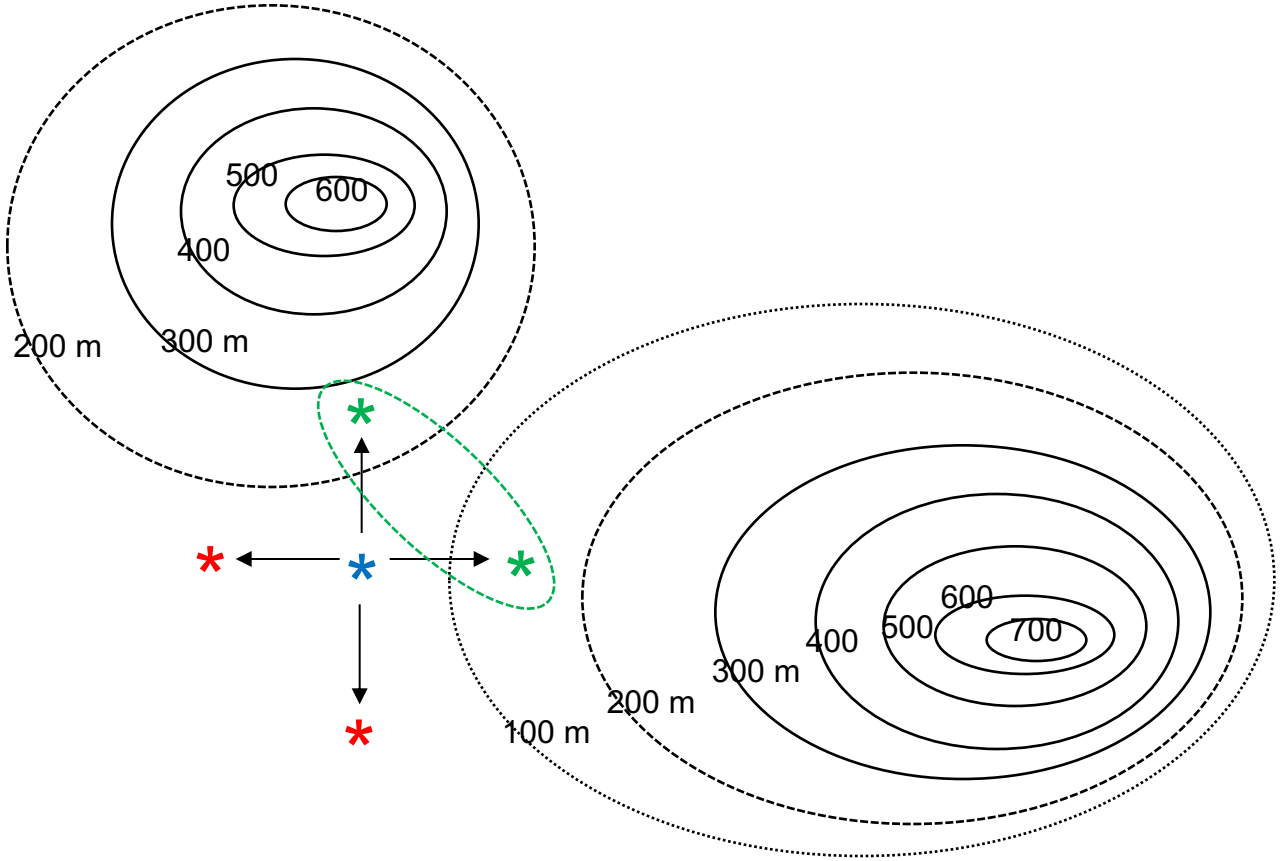
1. Make the **initial state** the **current state**.
2. **Generate** and **assess** the **children-states** of the current state.
3. If **no child-state is better** than the current state, **return the current state**.
4. Make the **best child-state** the **current state**.
5. Go to **step 2**.

Spoiler alert: Most **neural networks** are also trained using a **kind of HC** (**SGD**, stochastic gradient descent), where the **state contains the weights** of the network.

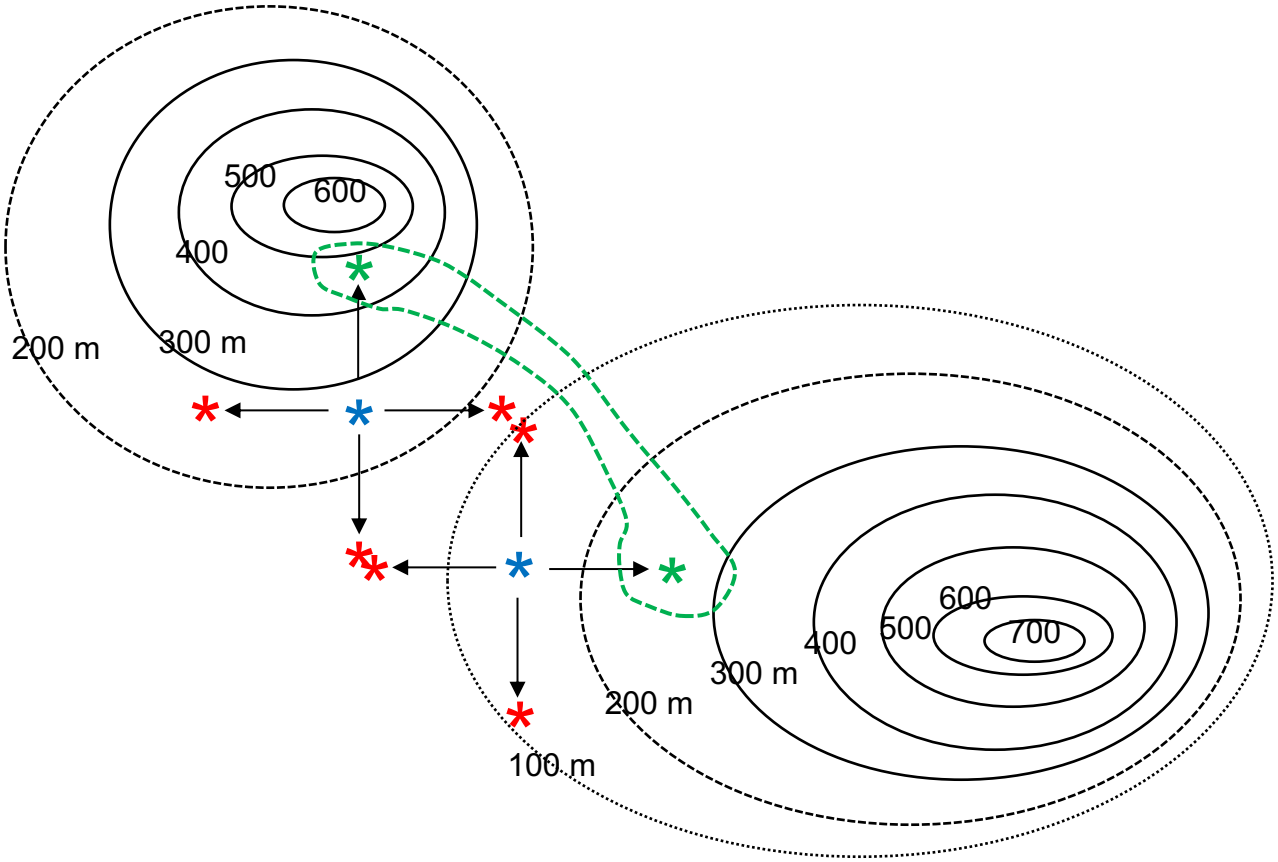
Hill climbing



(Local) Beam search



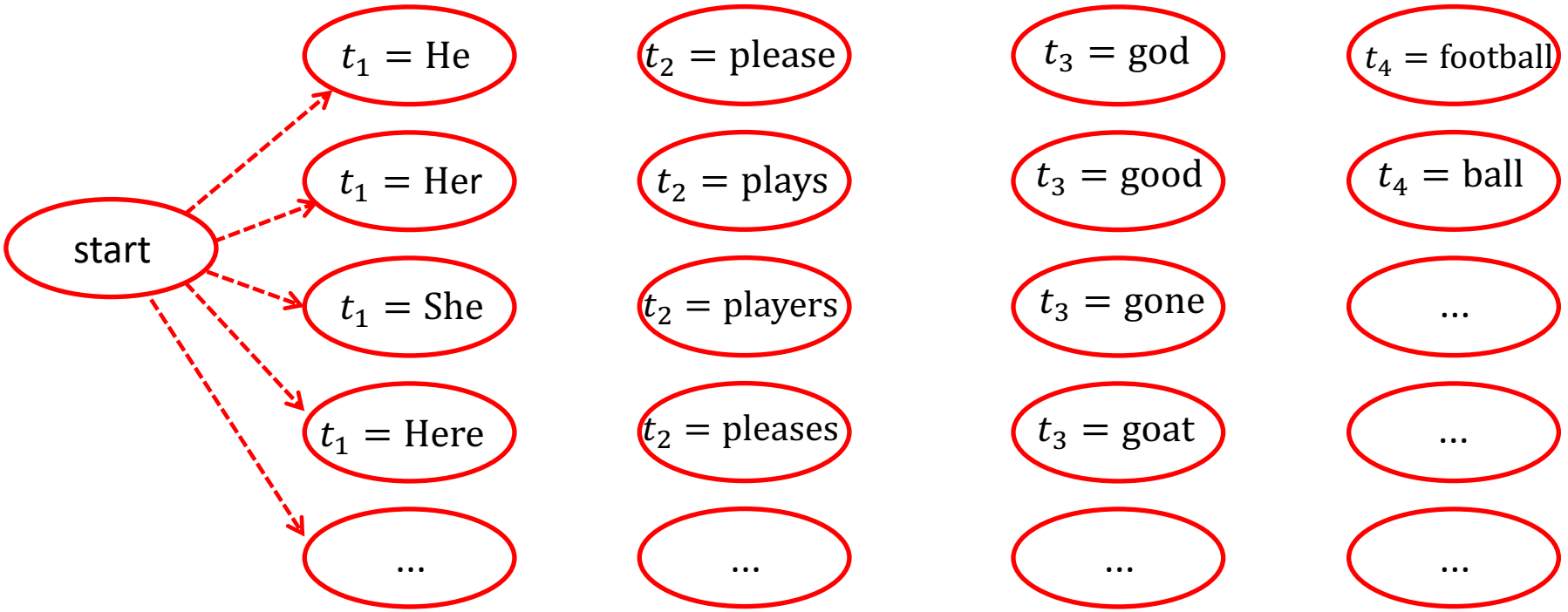
(Local) Beam search



(Local) Beam search

- Like HC, but we keep ***k* states** in the **search frontier**.
  - Initially *k* random states.
- At each step, **produce** and **assess** the **children-states of the *k* states** in the frontier.
  - If a final state criterion exists and we reach a final state, stop.
- **Keep** the ***k* best of the children-states** and repeat.
  - Until we exceed a maximum number of iterations.
- We often **repeat the search several times**, starting from **different initial *k* states**.
  - Random restarts are also useful in HC.
  - In neural nets, restarts with different random initial weights.
  - In spelling correction decoding, there is only one initial state.

Beam search decoder

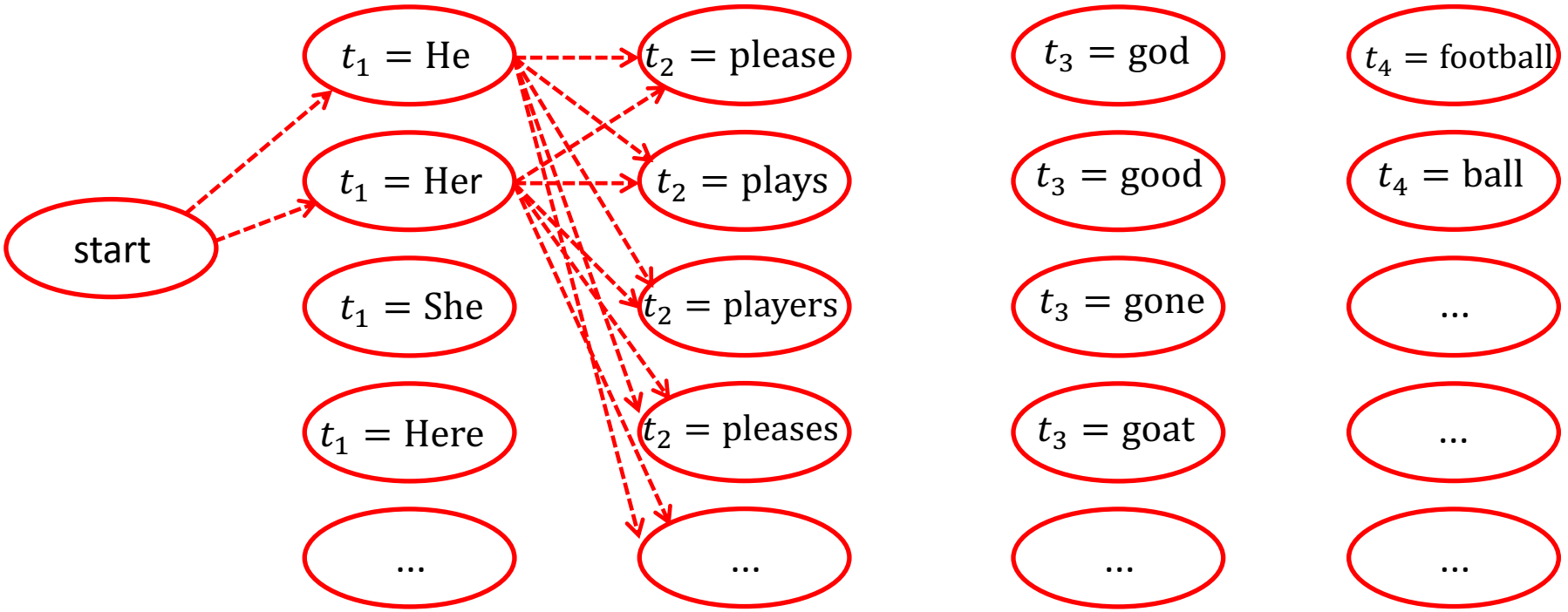


We search for a path from *start* to a state of column $k = 4$ that maximizes $P(t_1^k)P(w_1^k|t_1^k)$ or that minimizes $L_k = -\lambda_1 \log P(t_1^k) - \lambda_2 \log P(w_1^k|t_1^k)$.

With our previous simplifications: $\prod_{i=1}^{k} P(w_i|t_i)$

For a bigram language model: $\prod_{i=1}^{k} P(t_i|t_{i-1})$

Beam search decoder



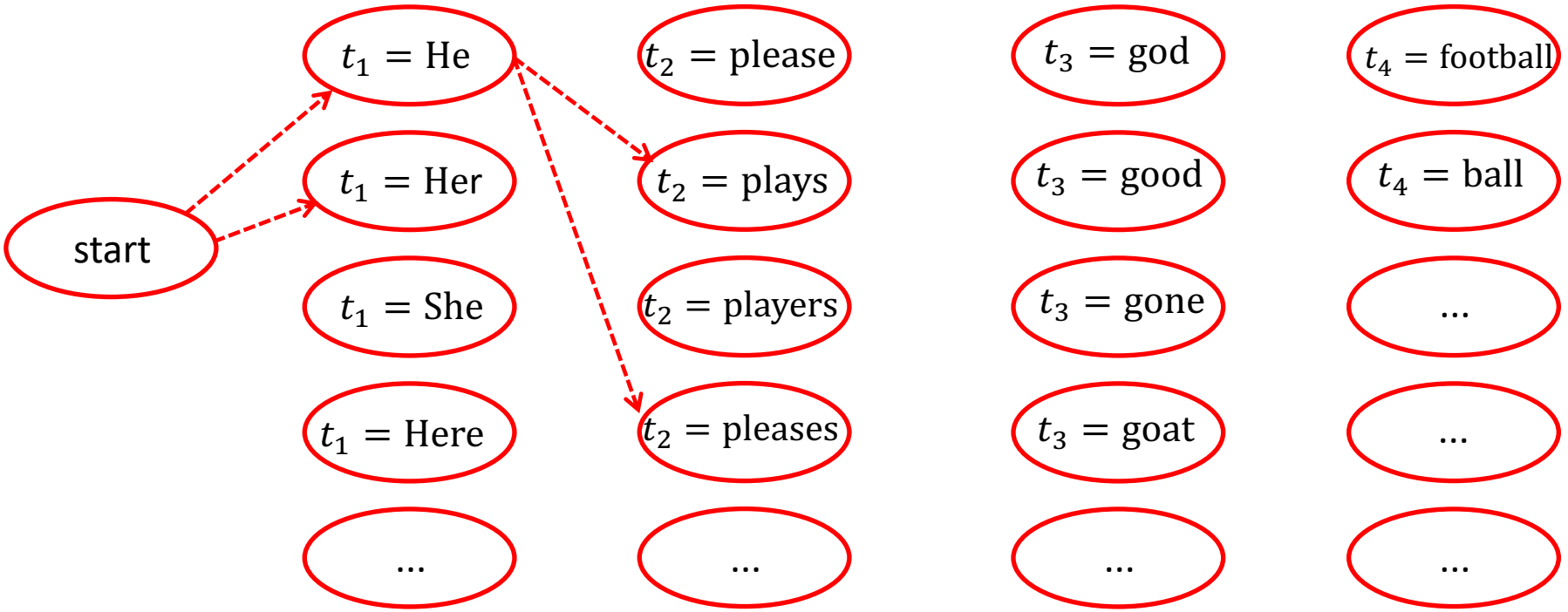
We search for a path from *start* to a state of column $k = 4$ that maximizes $P(t_1^k)P(w_1^k|t_1^k)$ or that minimizes $L_k = -\lambda_1 \log P(t_1^k) - \lambda_2 \log P(w_1^k|t_1^k)$.

With our previous simplifications: $\prod_{i=1}^{k} P(w_i|t_i)$

For a bigram language model: $\prod_{i=1}^{k} P(t_i|t_{i-1})$

For each k, we keep the b (here $b = 2$) best paths only.

Beam search decoder



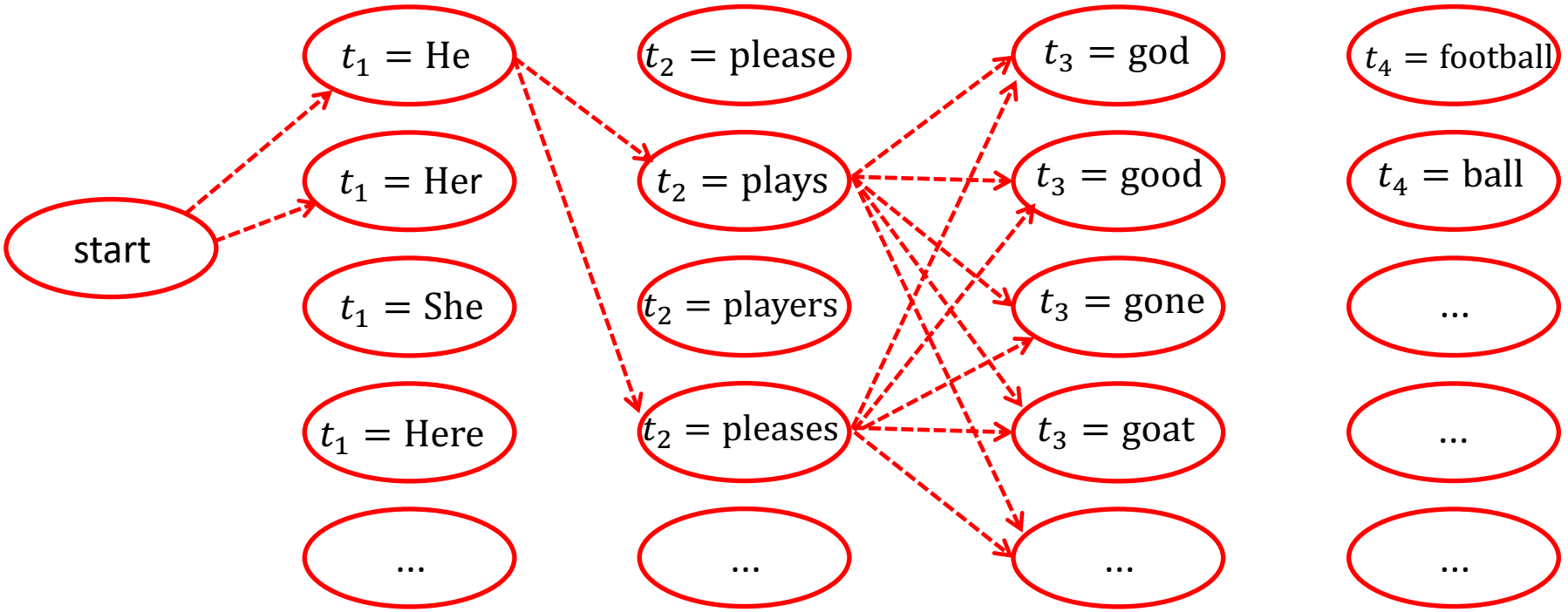
We search for a path from *start* to a state of column $k = 4$ that maximizes $P(t_1^k)P(w_1^k|t_1^k)$ or that minimizes $L_k = -\lambda_1 \log P(t_1^k) - \lambda_2 \log P(w_1^k|t_1^k)$.

With our previous simplifications: $\prod_{i=1}^{k} P(w_i|t_i)$

For a bigram language model: $\prod_{i=1}^{k} P(t_i|t_{i-1})$

For each k, we keep the b (here $b = 2$) best paths only.

Beam search decoder



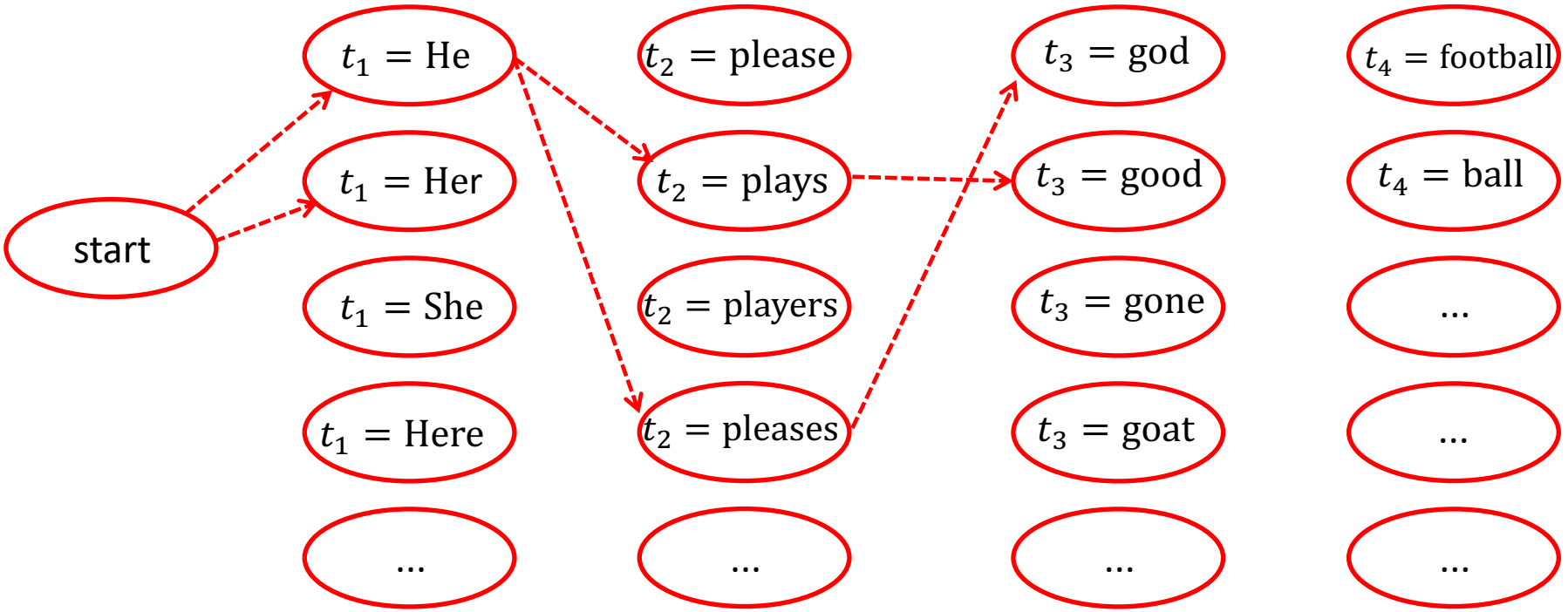
We search for a path from *start* to a state of column $k = 4$ that maximizes $P(t_1^k)P(w_1^k|t_1^k)$ or that minimizes $L_k = -\lambda_1 \log P(t_1^k) - \lambda_2 \log P(w_1^k|t_1^k)$.

With our previous simplifications: $\prod_{i=1}^{k} P(w_i|t_i)$

For a bigram language model: $\prod_{i=1}^{k} P(t_i|t_{i-1})$

For each k, we keep the b (here $b = 2$) best paths only.

Beam search decoder



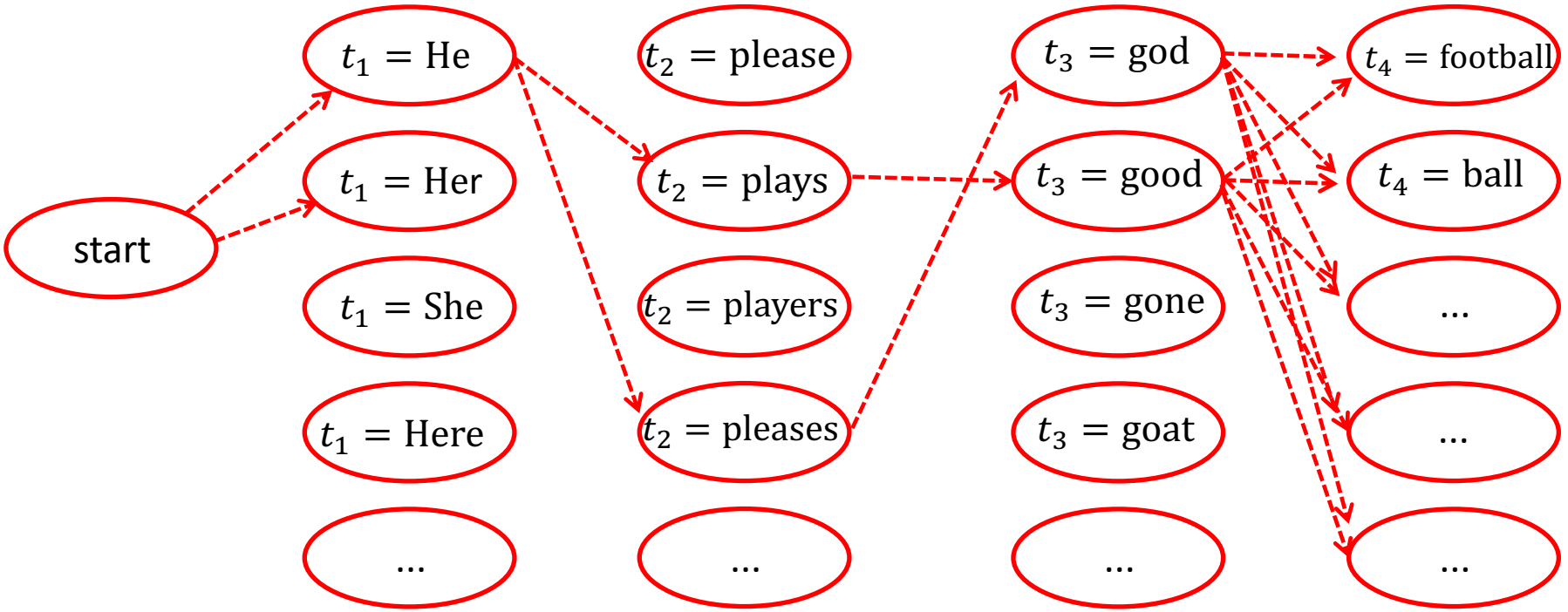
We search for a path from *start* to a state of column $k = 4$ that maximizes $P(t_1^k)P(w_1^k|t_1^k)$ or that minimizes $L_k = -\lambda_1 \log P(t_1^k) - \lambda_2 \log P(w_1^k|t_1^k)$.

With our previous simplifications: $\prod_{i=1}^{k} P(w_i|t_i)$

For a bigram language model: $\prod_{i=1}^{k} P(t_i|t_{i-1})$

For each k, we keep the b (here $b = 2$) best paths only.

Beam search decoder



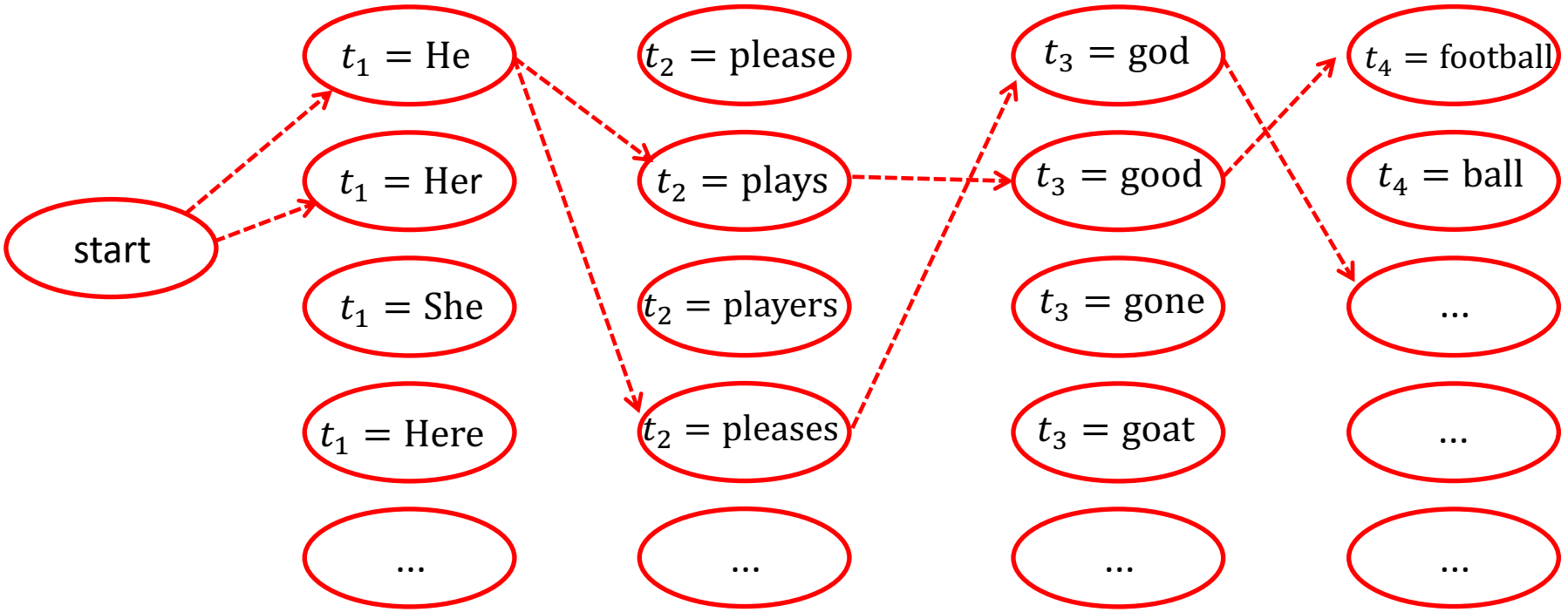
We search for a path from *start* to a state of column $k = 4$ that maximizes $P(t_1^k)P(w_1^k|t_1^k)$ or that minimizes $L_k = -\lambda_1 \log P(t_1^k) - \lambda_2 \log P(w_1^k|t_1^k)$.

With our previous simplifications: $\prod_{i=1}^{k} P(w_i|t_i)$

For a bigram language model: $\prod_{i=1}^{k} P(t_i|t_{i-1})$

For each k, we keep the b (here $b = 2$) best paths only.

Beam search decoder



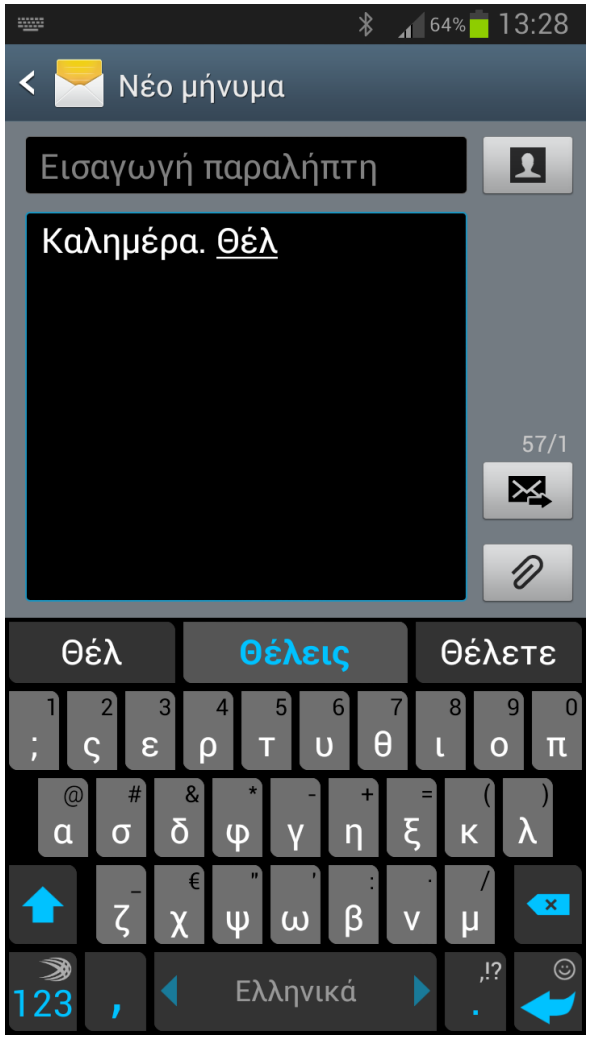
We search for a path from *start* to a state of column $k = 4$ that maximizes $P(t_1^k)P(w_1^k|t_1^k)$ or that minimizes $L_k = -\lambda_1 \log P(t_1^k) - \lambda_2 \log P(w_1^k|t_1^k)$.

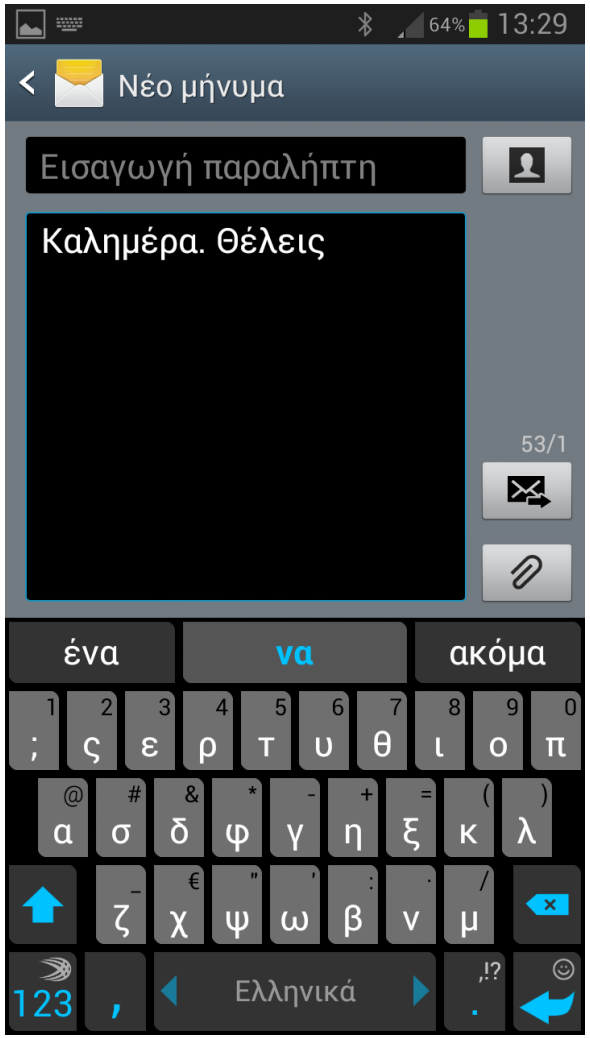
With our previous simplifications: $\prod_{i=1}^{k} P(w_i|t_i)$

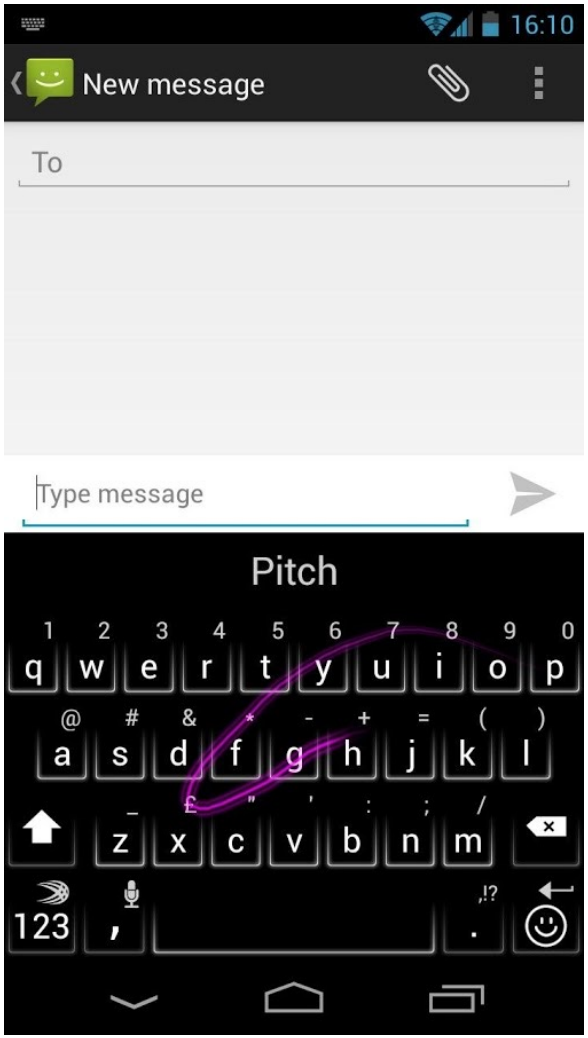
For a bigram language model: $\prod_{i=1}^{k} P(t_i|t_{i-1})$

For each k, we keep the b (here $b = 2$) best paths only.

Έξυπνα πληκτρολόγια







Εικόνες από: http://www.swiftkey.net/

Έξυπνα πληκτρολόγια

- Πιθανές **επόμενες λέξεις**:
  - Λέξεις του λεξικού που αν **προστεθούν** σε αυτές που **έχει ήδη γράψει** ο χρήστης παράγουν τις πιο **πιθανές** (σύμφωνα με ένα γλωσσικό μοντέλο) **ακολουθίες λέξεων**.
- **Συμπλήρωση** ή **διόρθωση** τρέχουσας **λέξης**:
  - Π.χ. λέξεις του λεξικού που έχουν **μικρή απόσταση διόρθωσης** από την **τρέχουσα λέξη**.
  - Και που αν **προστεθούν** στις **προηγούμενες λέξεις** παράγουν μια **πιθανή** (σύμφωνα με το γλωσσικό μοντέλο) **ακολουθία λέξεων**.

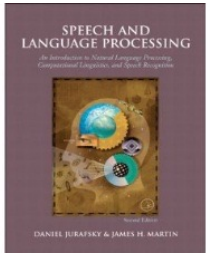
Έξυπνα πληκτρολόγια – συνέχεια

- «Πληκτρολόγηση» με **συνεχή κίνηση** δακτύλου:
  - Το πληκτρολόγιο παράγει «**υποθέσεις**», δηλαδή **πιθανές ακολουθίες γραμμάτων ή λέξεων** που ίσως ήθελε να πατήσει ο χρήστης.
  - Εξετάζουμε πόσο **κοντά** (απόσταση διόρθωσης) είναι οι **λέξεις** των **υποθέσεων** σε λέξεις του **λεξικού**.
  - Και πόσο ικανοποιούν ένα **γλωσσικό μοντέλο** (λέξεων ή/και γραμμάτων).

Άλλες εφαρμογές γλωσσικών μοντέλων

- **Αναγνώριση φωνής**.
  - Ένα «**ακουστικό μοντέλο**» παράγει πολλές **υποθέσεις** (π.χ. πιθανές προτάσεις) για το τι ίσως είπε ο χρήστης.
  - Ένα **γλωσσικό μοντέλο** εκτιμά πόσο πιθανή είναι στη γλώσσα του χρήστη η κάθε υπόθεση.
- **Οπτική αναγνώριση χαρακτήρων**.
  - Παράγονται **πιθανές ακολουθίες** λέξεων ή γραμμάτων.
  - Ένα **γλωσσικό μοντέλο** εκτιμά πόσο πιθανές είναι.
- **Μηχανική μετάφραση**.
  - Παράγονται **υποψήφιες μεταφράσεις** κάθε πρότασης.

Διάβασμα



- Η ύλη αυτής της ενότητας καλύπτεται από τις ενότητες 3.10, 3.11, 4.1–4.3, 4.5, 4.7, 5.9 του βιβλίου «Speech and Language Processing» των D. Jurafsky and J.H. Martin, 2η έκδοση, Prentice Hall, 2008. (Υπάρχει στη βιβλιοθήκη.)
  - Πολλά κεφάλαια της 3ης έκδοσης παρέχονται δωρεάν (βλ. http://web.stanford.edu/~jurafsky/slp3/).
- Δείτε προαιρετικά και τις διαφάνειες του μεταπτυχιακού μαθήματος «Επεξεργασία Φυσικής Γλώσσας».
  - https://eclass.aueb.gr/courses/INF210/
- Αλγορίθμους αναζήτησης σε χώρους καταστάσεων, θα βρείτε (προαιρετικά) στις διαφάνειες του προπτυχιακού μαθήματος «Τεχνητή Νοημοσύνη».
  - Π.χ. A*, beam search, γενετικοί αλγόριθμοι.