

结构化学习

Сергей Губанов
Яндекс
esgv@yandex-team.ru

24 октября 2019 г.

计划

依存树

Transition-based parsing 基于过渡的解析

ML for transition-based parsing ML用于基于过渡的解析

结构化学习

总结

Далее,

依存树

План

依存树

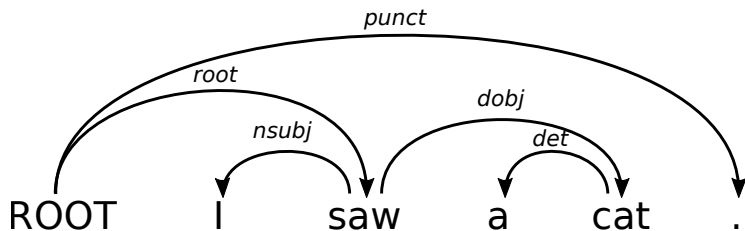
Transition-based parsing


ML for transition-based parsing

Структурированное обучение

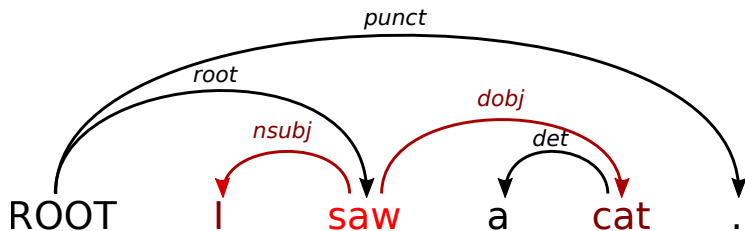
Итоги

依赖树



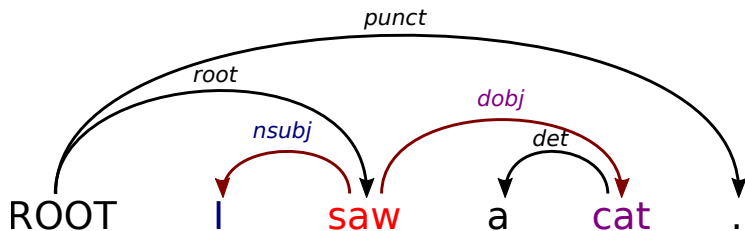
- ▶ Один корень 
- ▶ Связное
- ▶ Ацикличное

Дерево зависимостей




- ▶ Один корень
- ▶ Связное
- ▶ Ацикличное

Дерево зависимостей

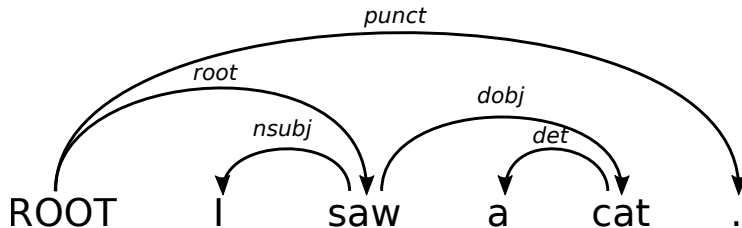


- ▶ Один корень
- ▶ Связное
- ▶ Ацикличное

Дерево зависимостей

- ▶ Treebank: корпус размеченных деревьев. 
- ▶ <http://universaldependencies.org/>

CoNLL format

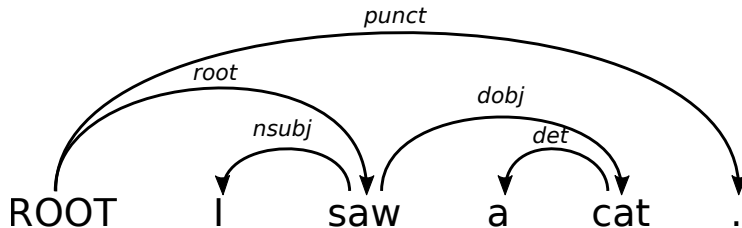


<http://ilk.uvt.nl/conll/>



ID	FORM	LEMMA	CPOSTAG	POSTAG	FEATS	HEAD	DEPREL
1	I	I	PRON	PRP	Case=Nom Number=Sing ...	2	nsubj
2	saw	see	VERB	VBD	Mood=Ind Tense=Past ...	0	root
3	a	a	DEP	DT	Definite=Ind PronType=Art	4	det
4	cat	cat	NOUN	NN	Number=Sing	2	dobj
5	.	.	PUNCT	.	-	0	punct

CoNLL format



<http://ilk.uvt.nl/conll/>

ID	FORM	LEMMA	CPOSTAG	POSTAG	FEATS	HEAD	DEPREL
1	I	I	PRON	PRP	Case=Nom Number=Sing ...	2	nsubj
2	saw	see	VERB	VBD	Mood=Ind Tense=Past ...	0	root
3	a	a	DEP	DT	Definite=Ind PronType=Art	4	det
4	cat	cat	NOUN	NN	Number=Sing	2	dobj
5	.	.	PUNCT	.	-	0	punct

Далее,

Transition-based parsing

План

Деревья зависимостей

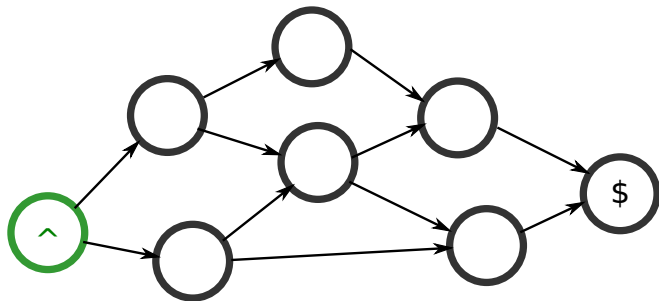
Transition-based parsing

ML for transition-based parsing

Структурированное обучение

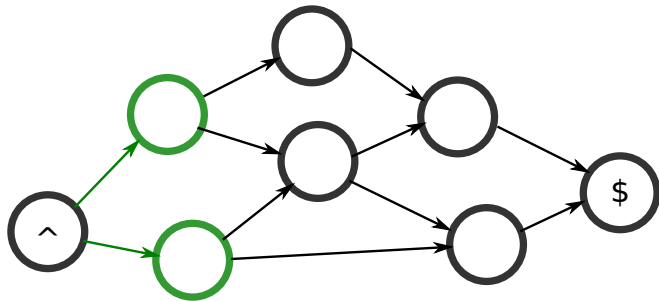
Итоги


Система переходов



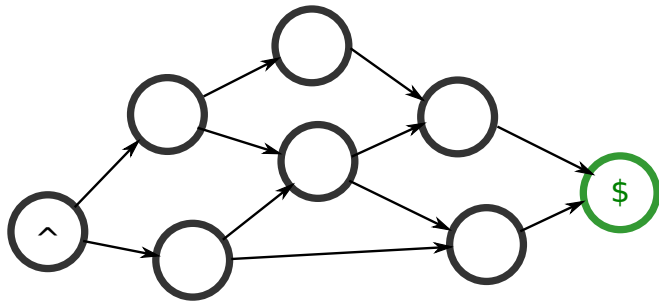
► Начальная вершина

Система переходов



- ▶ Начальная вершина
- ▶ Переходы в другие вершины 

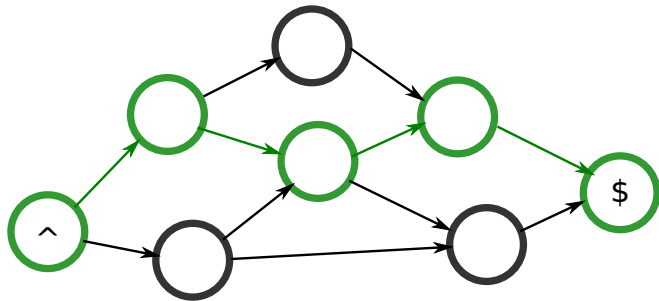
Система переходов




- ▶ Начальная вершина
- ▶ Переходы в другие вершины
- ▶ Конечная вершина




Система переходов




- ▶ Начальная вершина
- ▶ Переходы в другие вершины
- ▶ Конечная вершина
- ▶ Путь 

Где еще

- ▶ Конечные автоматы 
 - ▶ Детерминированные и нет
 - ▶ Трансдюсеры
- ▶ Таггеры
 - ▶ POS
 - ▶ NER
 - ▶ CJK segmentation
 - ▶ Дизамбигуация
- ▶ Парсера
 - ▶ Оба вида
- ▶ Опечатки
 - ▶ Генерация гипотез
 - ▶ Выбор конечного варианта

Где еще

- ▶ Конечные автоматы
 - ▶ Детерминированные и нет
 - ▶ Трансдюсеры
- ▶ Таггеры
 - ▶ POS
 - ▶ NER
 - ▶ CJK segmentation
 - ▶ Дизамбигуация
- ▶ Парсера
 - ▶ Оба вида
- ▶ Опечатки
 - ▶ Генерация гипотез
 - ▶ Выбор конечного варианта
- ▶ Машинный перевод 

План



- ▶ Определяем систему переходов для парсинга
- ▶ Обучаем классификатор для выбора наилучшего перехода
- ▶ Парсим, последовательно применяя классификатор

Arc-eager

[ROOT] I saw a cat .



- ▶ S : Shift
- ▶ R : Reduce
- ▶ S_{RA} : Right-arc shift
- ▶ R_{LA} : Left-arc reduce

Arc-eager

[ROOT I] saw a cat .

S I

- ▶ S : Shift
- ▶ R : Reduce
- ▶ S_{RA} : Right-arc shift
- ▶ R_{LA} : Left-arc reduce

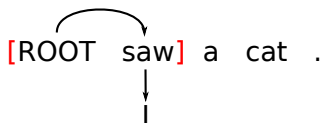
Arc-eager

[ROOT] saw a cat .
↓
I

S I
R_{LA} I ← saw

- ▶ S : Shift
- ▶ R : Reduce
- ▶ S_{RA} : Right-arc shift
- ▶ R_{LA} : Left-arc reduce

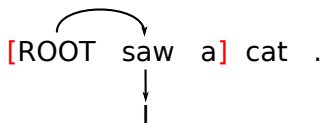
Arc-eager



S I
R_{LA} I ← saw
S_{RA} ROOT → saw

- ▶ S : Shift
- ▶ R : Reduce
- ▶ S_{RA} : Right-arc shift
- ▶ R_{LA} : Left-arc reduce

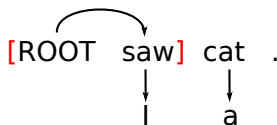
Arc-eager



S |
R_{LA} | ← saw
S_{RA} ROOT → saw
S a

- ▶ S : Shift
- ▶ R : Reduce
- ▶ S_{RA} : Right-arc shift
- ▶ R_{LA} : Left-arc reduce

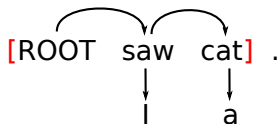
Arc-eager



S	I		R_{LA}	a ← cat
R_{LA}	I ← saw			
S_{RA}	ROOT → saw			
S	a			

- ▶ *S*: Shift
- ▶ *R*: Reduce
- ▶ *S_{RA}*: Right-arc shift
- ▶ *R_{LA}*: Left-arc reduce

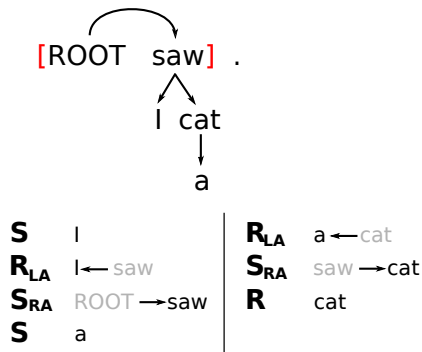
Arc-eager



S	I		R_{LA}	a ← cat
R_{LA}	I ← saw		S_{RA}	saw → cat
S_{RA}	ROOT → saw			
S	a			

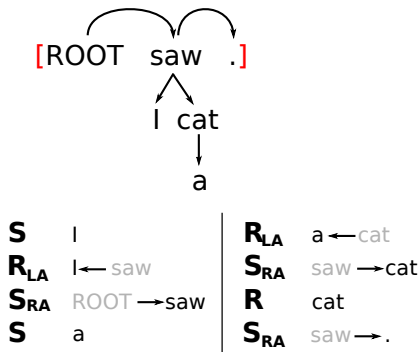
- ▶ *S*: Shift
- ▶ *R*: Reduce
- ▶ *S_{RA}*: Right-arc shift
- ▶ *R_{LA}*: Left-arc reduce

Arc-eager




- ▶ S : Shift
- ▶ R : Reduce
- ▶ S_{RA} : Right-arc shift
- ▶ R_{LA} : Left-arc reduce

Arc-eager



- ▶ S : Shift
- ▶ R : Reduce
- ▶ S_{RA} : Right-arc shift
- ▶ R_{LA} : Left-arc reduce

Свойства

- ▶ Любой путь задает дерево 
- ▶ Для любого *проективного* дерева найдется путь
- ▶ Любой путь конечен
- ▶ (И другие хорошие свойства)

Свойства

- ▶ Любой путь задает дерево
- ▶ Для любого *проективного* дерева найдется путь
- ▶ Любой путь конечен
- ▶ (И другие хорошие свойства)

Пройтись до конечного состояния

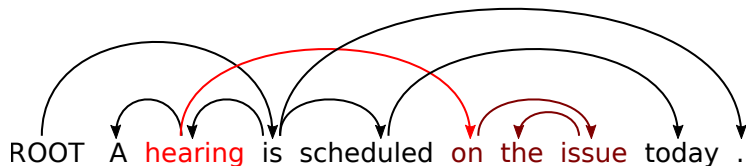


=

Построить дерево

Проективность

Непроективное – это когда нельзя нарисовать без пересечения дуг.



Проективное, если



$$\forall i, j : i < j \text{ and } (i \rightarrow j \text{ or } i \leftarrow j)$$

$$\Rightarrow$$

$$\forall k : i < k < j \Rightarrow i \rightarrow^* k \text{ or } j \rightarrow^* k$$

Далее,

ML for transition-based parsing

План

Деревья зависимостей

Transition-based parsing

ML for transition-based parsing

Структурированное обучение

Итоги

Последовательное принятие решений



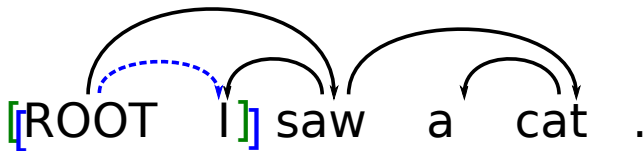
[ROOT] I saw a cat .

Последовательное принятие решений

 [ROOT I] saw a cat .

► S, S_{RA}

Последовательное принятие решений



► S, S_{RA}

Последовательное принятие решений

[ROOT I] saw a cat .

► S, S_{RA}

Последовательное принятие решений

[ROOT I saw] a cat .

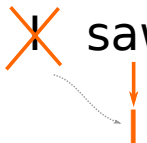
A diagram illustrating a sequence of words: "ROOT", "I", "saw", "a", "cat", and ".". The words "ROOT", "I", and "saw" are enclosed in blue square brackets. A green dashed curved arrow points from the word "I" to the word "saw", indicating a dependency or relationship between them.

► S, S_{RA}

► S, S_{RA}

Последовательное принятие решений

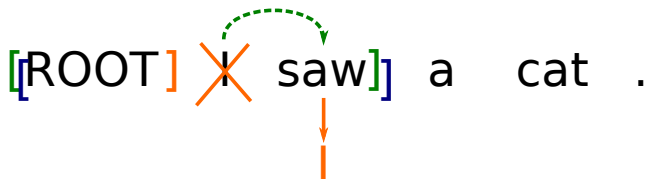
[ROOT] ~~I~~ saw a cat .



► S, S_{RA}

► S, S_{RA}, R_{LA}

Последовательное принятие решений



► S, S_{RA}

► S, S_{RA}, R_{LA}

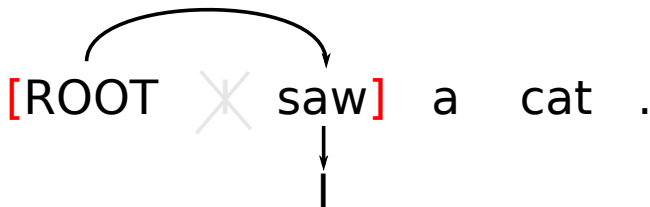
Последовательное принятие решений

[ROOT] ✕ saw a cat .
↓

► S, S_{RA}

► S, S_{RA}, R_{LA}

Последовательное принятие решений



- ▶ S, S_{RA}
- ▶ S, S_{RA}, R_{LA}
- ▶ S, S_{RA}
- ▶ ...

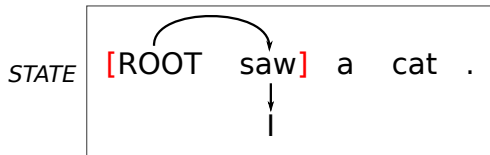
Последовательное принятие решений

[ROOT] I saw a cat .	S, S_{RA}
[ROOT I] saw a cat .	S, S_{RA}, R_{LA}
[ROOT] X saw a cat . ↓	S_{RA}, S
...	...

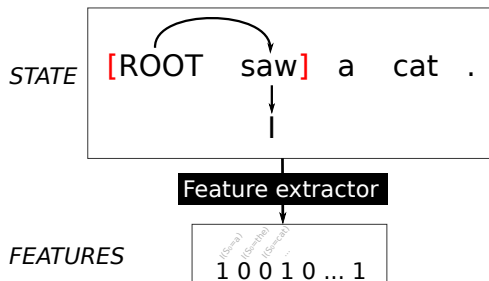


- ▶ Проходим *эталонную* последовательность состояний
- ▶ На каждом шаге учим классификатор принимать правильное решение

Машинное обучение

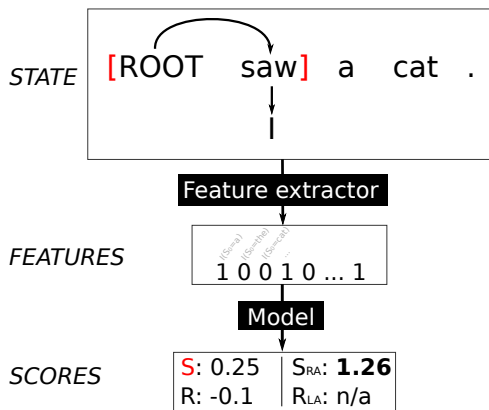


Машинное обучение



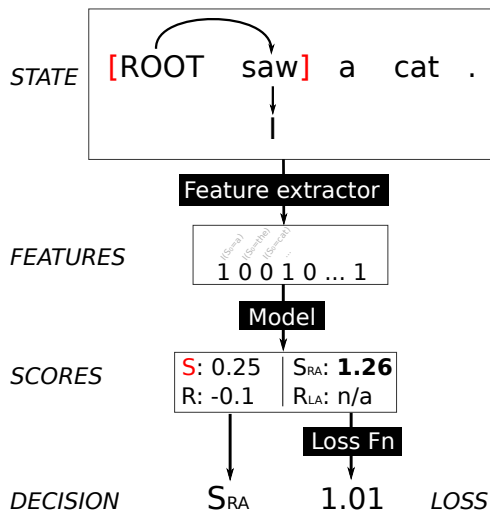
► Extract features

Машинное обучение



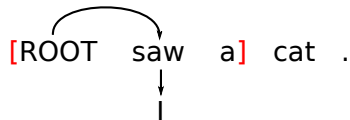
- ▶ Extract features
- ▶ Invoke model

Машинное обучение

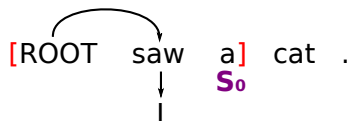


- ▶ Extract features
- ▶ Invoke model
- ▶ Make decision
- ▶ Compute loss

Features



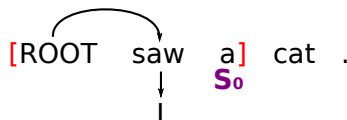
Features



$I(S_0=a)$
1

► $I(S_0 = "a")$

Features

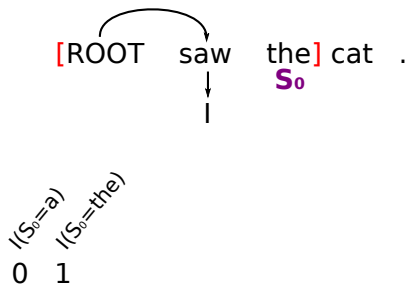


$I(S_0=a)$
1

$I(S_0=the)$
0

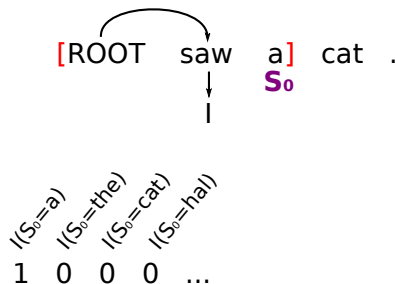
- ▶ $I(S_0 = "a")$
- ▶ $I(S_0 = "the")$

Features



- ▶ $I(S_0 = \text{"a"})$
- ▶ $I(S_0 = \text{"the"})$

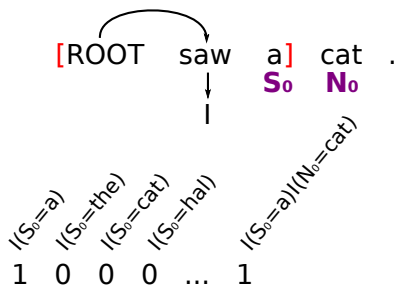
Features




- ▶ $I(S_0 = "a")$
- ▶ $I(S_0 = "the")$
- ▶ $I(S_0 = w)$ для каждого слова w

S_0 – это feature template (она же – категориальная ϕ -ча).

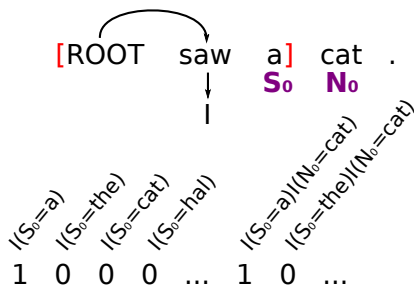
Feature template



 Более сложная фича: S_0N_0 .

 Т.е: для любых слов w и v , $I(S_0 = w) \cdot I(N_0 = v)$.

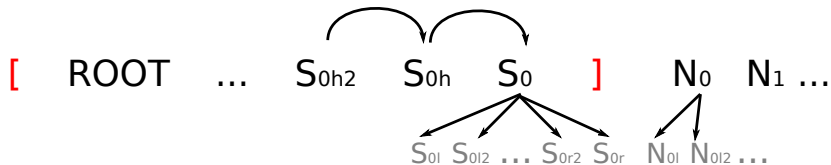
Feature template



Более сложная фича: S_0N_0 .

Т.е: для любых слов w и v , $I(S_0 = w) \cdot I(N_0 = v)$.

Dep. parser feature templates



Dep. parser feature templates

- ▶ From single words: S_0wp ; S_0w ; S_0p ; N_0wp ; N_0w ; N_0p ; N_1wp ; N_1w ; N_1p ; N_2wp ; N_2w ; N_2p
- ▶ From word pairs: S_0wpN_0wp ; S_0wpN_0w ; S_0wN_0wp ; S_0wpN_0p ; S_0pN_0wp ; S_0wN_0w ; S_0pN_0p ; N_0pN_1p
- ▶ From three words: $N_0pN_1pN_2p$; $S_0pN_0pN_1p$; $S_0hpS_0pN_0p$; $S_0pS_0lpN_0p$; $S_0pS_0rpN_0p$; $S_0pN_0pN_0lp$
- ▶ Distance: S_0wd ; S_0pd ; N_0wd ; N_0pd ; S_0wN_0wd ; S_0pN_0pd ;
- ▶ Valency: S_0wv_r ; S_0pv_r ; S_0wv_l ; S_0pv_l ; N_0wv_l ; N_0pv_l
- ▶ Unigrams: S_0hw ; S_0hp ; S_0l ; S_0lw ; S_0lp ; S_0ll ; S_0rw ; S_0rp ; S_0rl ; N_0lw ; N_0lp ; N_0ll
- ▶ Third-order: S_0h_2w ; S_0h_2p ; S_0hl ; S_0l_2w ; S_0l_2p ; S_0l_2l ; S_0r_2w ; S_0r_2p ; S_0r_2l ; N_0l_2w ; N_0l_2p ; N_0l_2l ; $S_0pS_0lpS_0l_2p$; $S_0pS_0rpS_0r_2p$; $S_0pS_0hpS_0h_2p$; $N_0pN_0lpN_0l_2p$
- ▶ Label set: S_0ws_r ; S_0ps_r ; S_0ws_l ; S_0ps_l ; N_0ws_l ; N_0ps_l

w – word, p – POS-tag, v_l , v_r – valency, l – deprel, s_l , s_r – labelset.

[Zhang and Nivre, 2011]

Model

Линейная модель 

$$s = w^T \cdot f$$

- ▶ Каждой фиче соответствует свой вес.
- ▶ Идеально ложится на sparsity, hashing trick, и т.д.



Loss

Мультиклассификация: S, R, S_{RA}, R_{LA}

$$s_i = w_i^T f$$

$$c = \operatorname{argmax} s_i$$

Perceptron loss

$$\mathcal{L} = \max s_i - s_{\text{correct}}$$

(Но можно вообще любой loss)

Perceptron algorithm

Бинарная классификация, линейная модель



$$s = w^T f$$

$$c = I(s > 0)$$

Perceptron algorithm

Бинарная классификация, линейная модель

$$s = w^T f$$

$$c = I(s > 0)$$


- ▶ Правильно \Rightarrow ничего не делаем.

Perceptron algorithm

Бинарная классификация, линейная модель

$$s = w^T f$$

$$c = I(s > 0)$$

▶  Правильно \Rightarrow ничего не делаем.

▶ Неправильно

▶ $c_{\text{correct}} = 0, c = 1 \Rightarrow$ надо бы опустить s

▶ $c_{\text{correct}} = 1, c = 0 \Rightarrow$ надо бы поднять s



Perceptron algorithm

Бинарная классификация, линейная модель

$$s = w^T f$$

$$c = I(s > 0)$$

- ▶ Правильно \Rightarrow ничего не делаем.
- ▶ Неправильно
 - ▶ $c_{\text{correct}} = 0, c = 1 \Rightarrow w = w - f$
 - ▶ $c_{\text{correct}} = 1, c = 0 \Rightarrow w = w + f$

Perceptron algorithm

Бинарная классификация, линейная модель

$$s = w^T f$$

$$c = I(s > 0)$$

▶ Правильно \Rightarrow ничего не делаем.

▶ Неправильно

▶ $c_{\text{correct}} = 0, c = 1 \Rightarrow w = w - f$

▶ $c_{\text{correct}} = 1, c = 0 \Rightarrow w = w + f$

$$s' = (w - f)^T f = w^T f - \underbrace{f^T f}_{\geq 0} = s -$$

$$\nabla_w s = f$$

Linear model + Perceptron loss + SGD

$$s_0 = -w^T f$$

$$s_1 = +w^T f$$

Linear model + Perceptron loss + SGD

$$s_0 = -w^T f$$

$$s_1 = +w^T f$$

$$\mathcal{L} = \max(s_i) - s_{\text{correct}}$$

$$w^{(t)} = w^{(t-1)} - \nabla \mathcal{L}^{(t-1)}$$


Linear model + Perceptron loss + SGD

$$s_0 = -w^T f$$

$$s_1 = +w^T f$$

$$\mathcal{L} = \max(s_i) - s_{\text{correct}}$$

$$w^{(t)} = w^{(t-1)} - \nabla \mathcal{L}^{(t-1)}$$

Это то же самое. 

$$\nabla \mathcal{L} = \begin{cases} 0 & \max(s_i) = s_{\text{correct}} \\ \pm 2f & \text{otherwise} \end{cases}$$

Результат

Transitions	Arc-eager
Features	[Zhang and Nivre, 2011]
Model	Linear
Loss	Perceptron
Training	SGD

Результат

Transitions	Arc-eager, Arc-standard, Easy-first, ...
Features	[Zhang and Nivre, 2011] , ...
Model	Linear, Neural network, ...
Loss	Perceptron, Logistic, SVM, ...
Training	SGD, L-BFGS, Averaged SGD, ...

Далее,

Структурированное обучение

План

Деревья зависимостей

Transition-based parsing

ML for transition-based parsing

Структурированное обучение

Итоги

Проблемы

Проблемы

- ▶ Жадность
 - ▶ Garden-path sentences
 - ▶ *The students forgot the solution ...*




Проблемы

- ▶ Жадность
 - ▶ Garden-path sentences
 - ▶ *The students forgot the solution was*


Проблемы

- ▶ Жадность
 - ▶ Garden-path sentences
 - ▶ *The students forgot the solution was in the back of the book.*

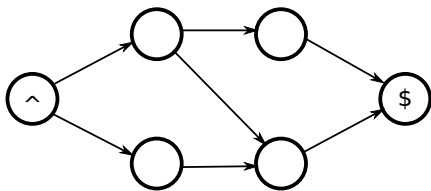
Проблемы

- ▶ Жадность 
 - ▶ Garden-path sentences
 - ▶ *The students forgot the solution was in the back of the book.*
- ▶ Error propagation 
 - ▶ Одна ошибка ведет к другой. 
 - ▶ Классификатор получает на вход данные, на которых он не обучался.

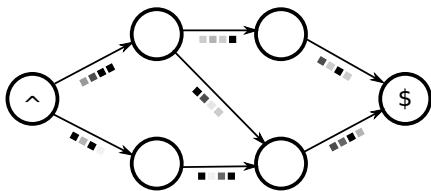
Проблемы

- ▶ Жадность
 - ▶ Garden-path sentences
 - ▶ *The students forgot the solution was in the back of the book.*
- ▶ Error propagation
 - ▶ Одна ошибка ведет к другой.
 - ▶ Классификатор получает на вход данные, на которых он не обучался.
- ▶ Не все ошибки одинаково вредны 
 - ▶ Неправильно приклеить корень — более страшно, чем неправильно приклеить артикль.

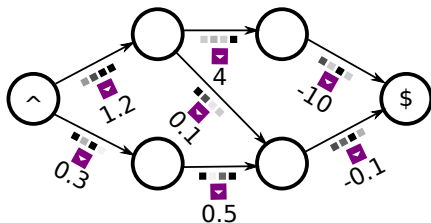
Структурированное обучение



Структурированное обучение



Структурированное обучение



- ▶ Веса на ребрах 
- ▶ Общий вес пути есть сумма весов ребер

$$s_e = w^T f_e$$

$$s = \sum_e s_e$$

Multiclass for NLP

$$s_i = w_i^T f$$

- ▶ K комплектов параметров
- ▶ 1 комплект фичей



Multiclass for NLP

$$s_i = w_i^T f$$

- ▶ K комплектов параметров
- ▶ 1 комплект фичей

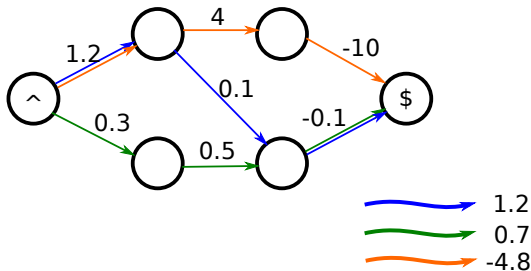


$$s_i = w^T f_i$$

- ▶ 1 комплект параметров
- ▶ K комплектов фичей



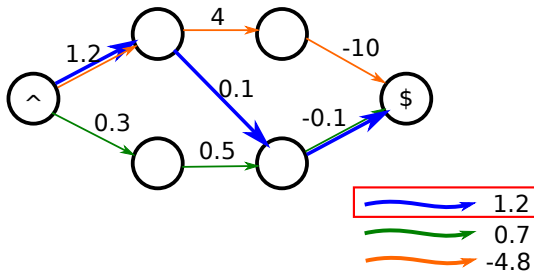
Структурированное обучение



- Хотим, чтобы вес правильного пути был наилучшим

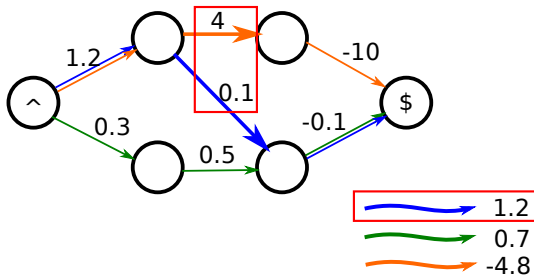


Структурированное обучение



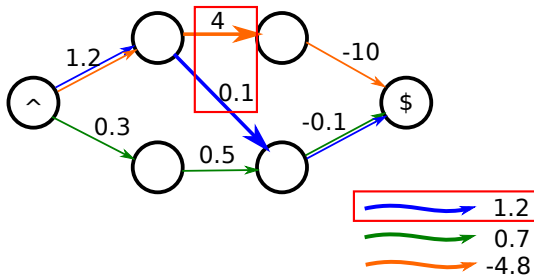
- ▶ Хотим, чтобы вес правильного пути был наилучшим *на каждом шаге?*
- ▶ Хотим, чтобы вес правильного пути был наилучшим *среди всех путей?*

Структурированное обучение



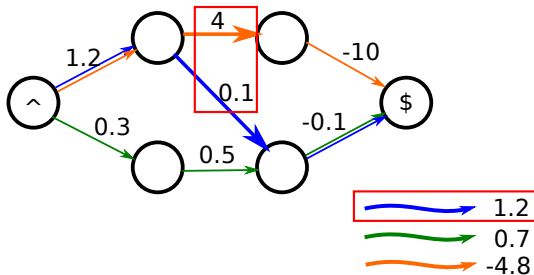
- ▶ Хотим, чтобы вес правильного пути был наилучшим *на каждом шаге?*
- ▶ Хотим, чтобы вес правильного пути был наилучшим *среди всех путей?*

Структурированное обучение



- ▶ Хотим, чтобы вес правильного пути был наилучшим *на каждом шаге?*
- ▶ Хотим, чтобы вес правильного пути был наилучшим *среди всех путей?*
- ▶ Есть один правильный путь, но много просто хороших.

Структурированное обучение



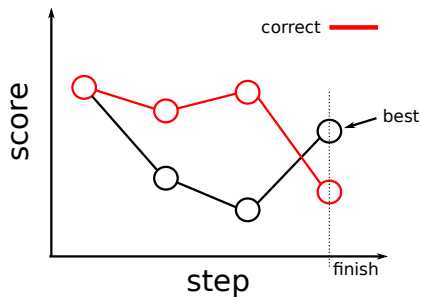
- ▶ Хотим, чтобы вес правильного пути был наилучшим *на каждом шаге?*
- ▶ Хотим, чтобы вес правильного пути был наилучшим *среди всех путей?*
- ▶ Есть один правильный путь, но много хороших.
- ▶ В начале может быть непонятно, какой путь станет лучшим.



Стратегия обучения зависит от того, какой будет inference.

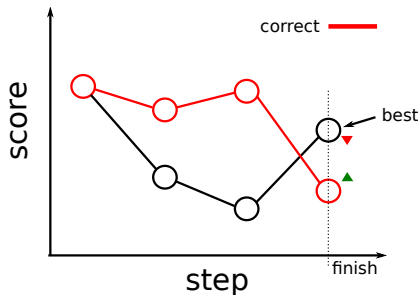
Exact inference

Если **можем перебрать все пути**



Exact inference

Если **можем перебрать все пути**



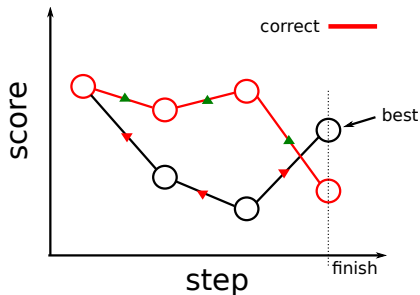
▶ Правильно \Rightarrow ничего не делаем.

▶ Неправильно \Rightarrow надо бы опустить s_{correct} и поднять s_{best} .

$$\begin{aligned} w &= w + \nabla s_{\text{correct}} - \nabla s_{\text{best}} \\ &= w + \sum_{e \in \text{correct}} \nabla s_e - \sum_{e \in \text{best}} \nabla s_e \end{aligned}$$

Exact inference

Если **можем перебрать все пути**

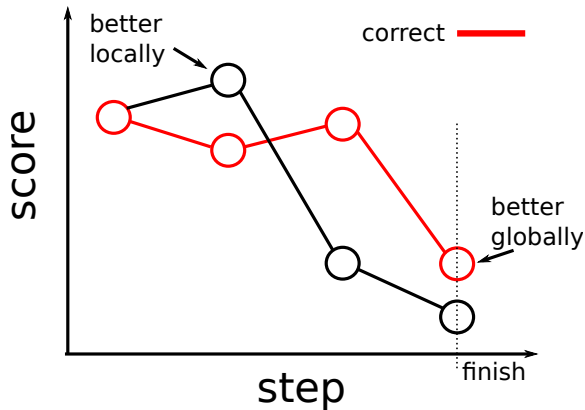


- ▶ Правильно \Rightarrow ничего не делаем.
- ▶ Неправильно \Rightarrow надо бы опустить s_{correct} и поднять s_{best} .

$$\begin{aligned} w &= w + \nabla s_{\text{correct}} - \nabla s_{\text{best}} \\ &= w + \sum_{e \in \text{correct}} \nabla s_e - \sum_{e \in \text{best}} \nabla s_e \end{aligned}$$

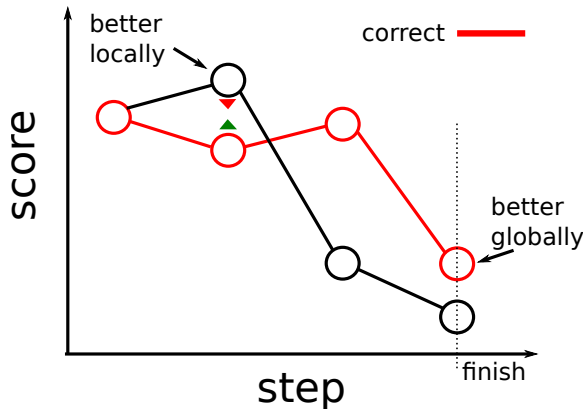
Greedy search

Если действуем жадно



Greedy search

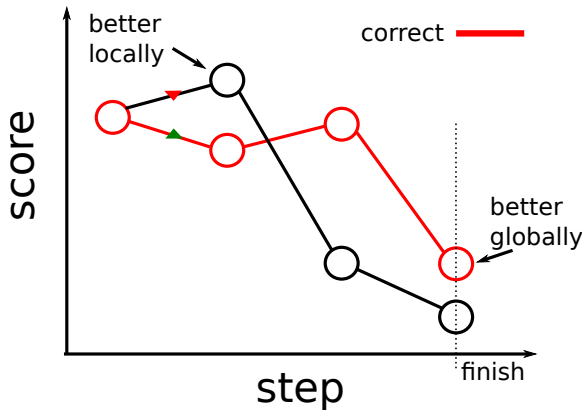
Если действуем жадно



► Лучше действовать как раньше.

Greedy search

Если действуем жадно



- Лучше действовать как раньше.

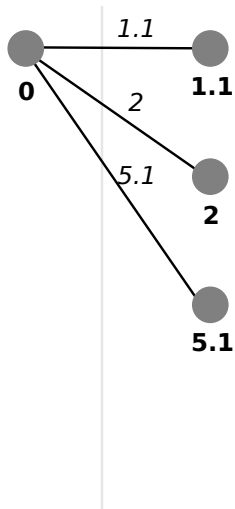
Beam search



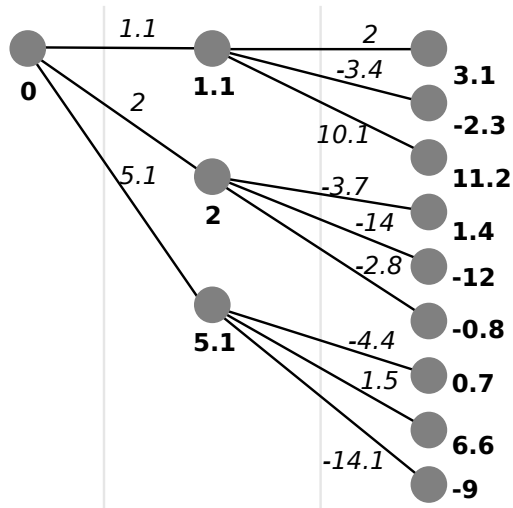
0



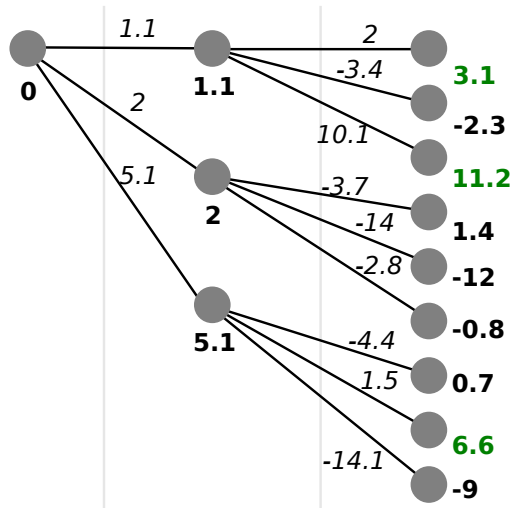
Beam search



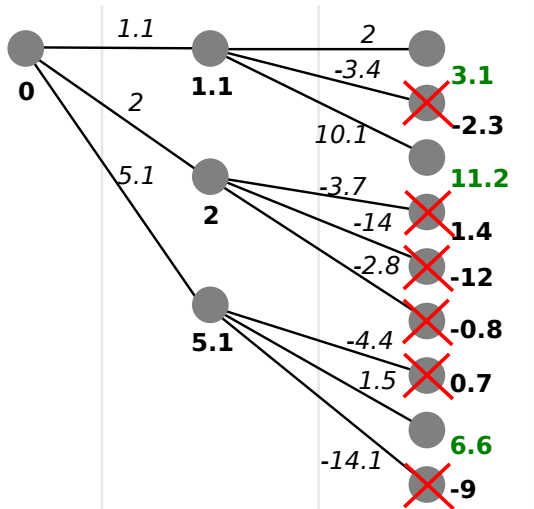
Beam search



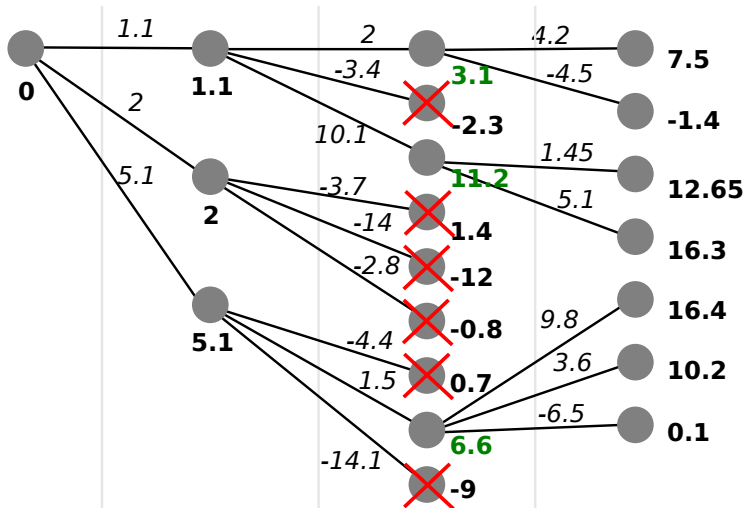
Beam search



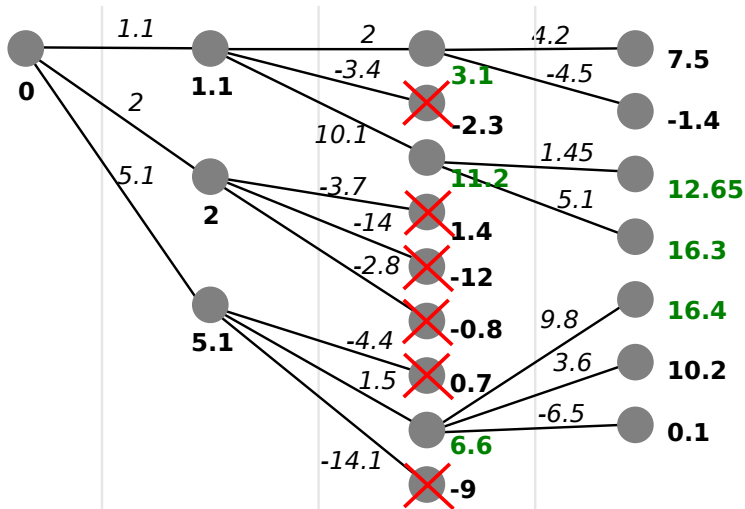
Beam search



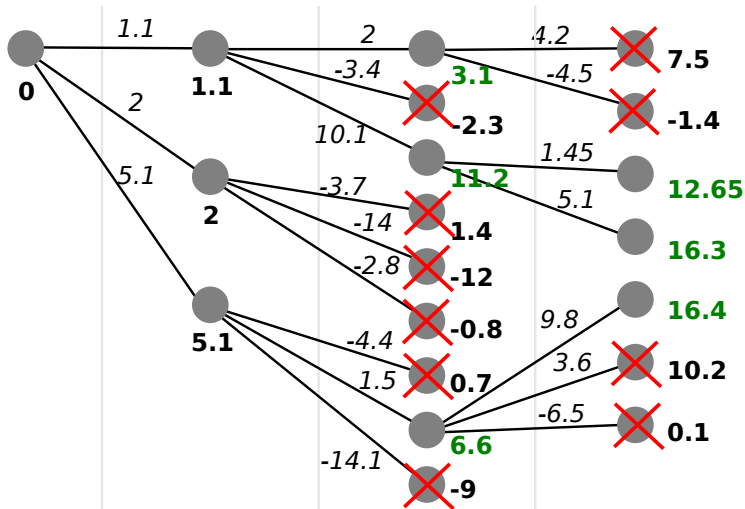
Beam search



Beam search



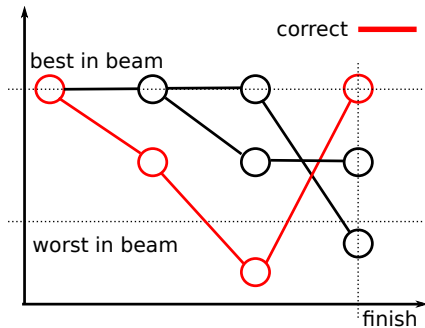
Beam search



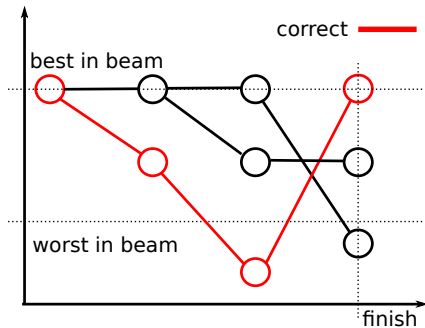
Beam search

- ▶ $\text{length} \times \text{beam size} = O(\text{length})$
- ▶ Отбрасываем заведомо проигрышные варианты
- ▶ Greedy — это когда $\text{beam size} = 1$,
exact inference — это когда $\text{beam size} = \infty$.

Invalid update

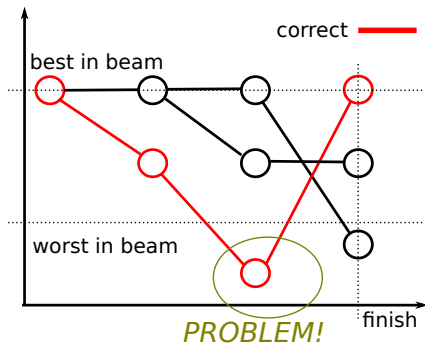


Invalid update



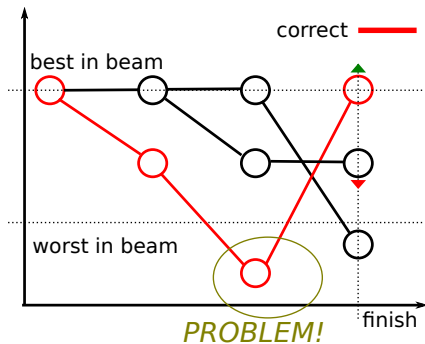
► No update

Invalid update



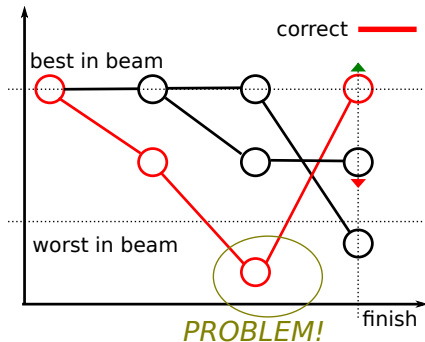
► No update

Invalid update



- No update

Invalid update



- ▶ No update
- ▶ Invalid update (потому что от него может не быть толку)

Дилемма

- ▶ Хотим, чтобы вес правильного пути был наилучшим *на каждом шаге?*
- ▶ Хотим, чтобы вес правильного пути был наилучшим *среди всех путей?*

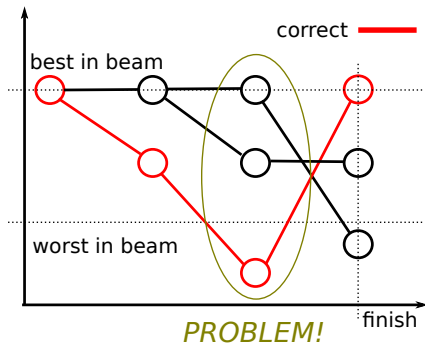
Дилемма

- ▶ Хотим, чтобы вес правильного пути был наилучшим *на каждом шаге?*
- ▶ Хотим, чтобы вес правильного пути был наилучшим *среди всех путей?*
- ▶ Хотим, чтобы вес правильного пути **не выпадал из бима**

Дилемма

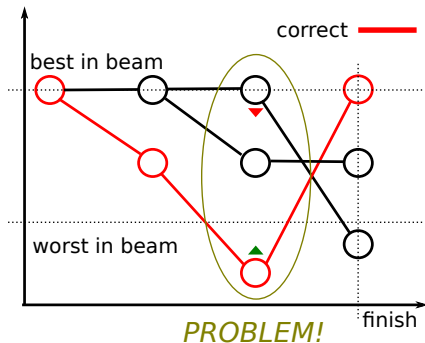
- ▶ Хотим, чтобы вес правильного пути был наилучшим *на каждом шаге?*
- ▶ Хотим, чтобы вес правильного пути был наилучшим *среди всех путей?*
- ▶ Хотим, чтобы вес правильного пути **не выпадал из бима**
- ▶ Хотим, чтобы вес правильного пути был лучшим в конце

Early update



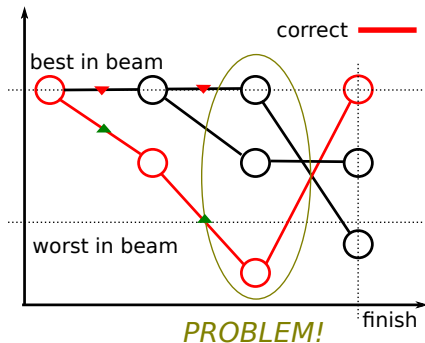
- No update
- Invalid update (потому что от него может не быть толку)

Early update



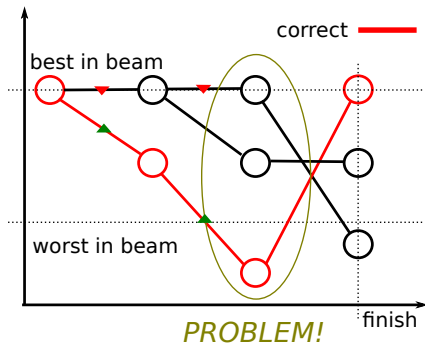
- No update
- Invalid update (потому что от него может не быть толку)

Early update



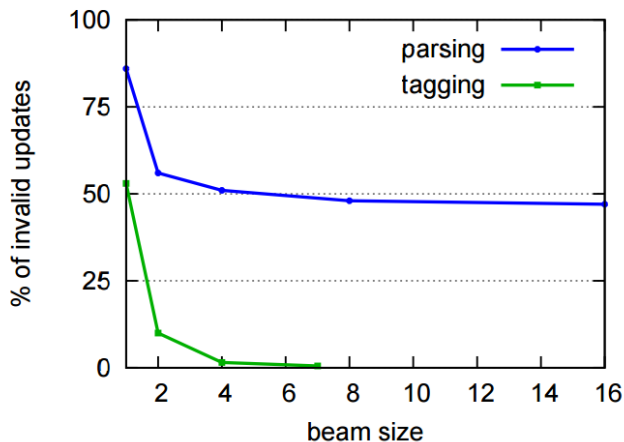
- No update
- Invalid update (потому что от него может не быть толку)

Early update



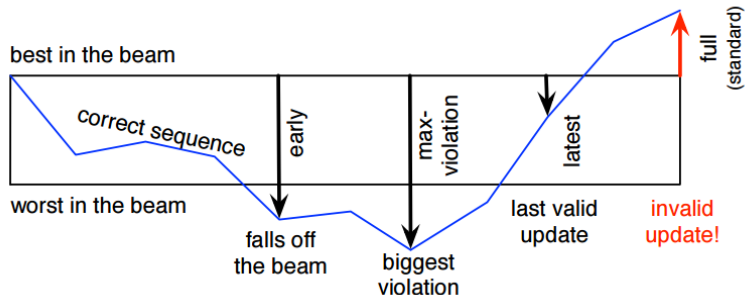
- No update
- Invalid update (потому что от него может не быть толку)
- Early update

Parsing vs tagging



[Huang et al., 2012]

Max-violation



- "Все, что требуется – это violation"

[Huang et al., 2012]

Прогресс

- ▶ Жадность
- ▶ Error propagation
- ▶ Цена разных ошибок

Прогресс

- ▶ Жадность
- ▶ Error propagation
- ▶ Цена разных ошибок

Dynamic oracle

Представьте, что мы уже сделали ошибку.

[ROOT I] saw a cat .



Dynamic oracle

Представьте, что мы уже сделали ошибку.



[ROOT I] saw a cat .

Dynamic oracle

Представьте, что мы уже сделали ошибку.

[ROOT I] saw a cat .



Dynamic oracle

Представьте, что мы уже сделали ошибку.



► S_{RA} / S

Dynamic oracle

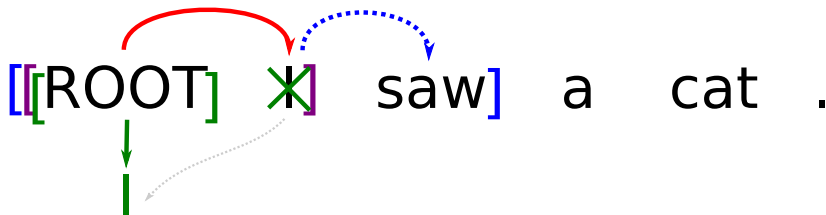
Представьте, что мы уже сделали ошибку.



► $S_{RA} / S, R$

Dynamic oracle

Представьте, что мы уже сделали ошибку.



► $S_{RA} / S, R$

Dynamic oracle

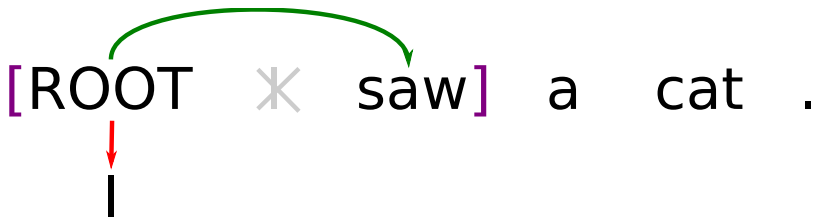
Представьте, что мы уже сделали ошибку.

[ROOT] ✖ saw a cat .
↓
I

► $S_{RA} / S, R$

Dynamic oracle

Представьте, что мы уже сделали ошибку.



► $S_{RA} / S, R$

Dynamic oracle

Представьте, что мы уже сделали ошибку.

 [ROOT I] saw a cat .

► $S_{RA} / S, R$

► Оракул: "лучше сделать R ".

Dynamic oracle

- ▶ Учимся в идеальной (эталонной) ситуации принимать правильное решение

[Goldberg and Nivre, 2013]



Dynamic oracle

- ▶ Учимся в идеальной (эталонной) ситуации принимать правильное решение (в начале обучения).
- ▶ В конце обучения учимся принимать решение в любой ситуации.

[Goldberg and Nivre, 2013]

Dynamic oracle

Если в любой момент времени применить текущую модель, будет достаточно нештатных ситуаций.

[ROOT] I saw a cat .	S, S_{RA}
 [ROOT] I saw a cat .	S, S_{RA}, R
[ROOT] ✱ saw a cat .  I	S_{RA}, S
...	...

- ▶ 10% exploration (или какая-то другая политика)
- ▶ Учимся ровно на том, что встретим в рантайме

Dynamic oracle

Как построить оракула — отдельная проблема.

[Goldberg and Nivre, 2012]

Результаты

Method	UAS	LAS
Greedy	89.88	87.69

(Stanford basic dependencies, WSJ 23)

UAS = Unlabeled Attachment Score; LAS = Labeled Attachment Score

► [\[Goldberg and Nivre, 2013\]](#)

Результаты

Method	UAS	LAS
Greedy	89.88	87.69
Dynamic oracle	90.96	88.72

(Stanford basic dependencies, WSJ 23)

UAS = Unlabeled Attachment Score; LAS = Labeled Attachment Score

► [\[Goldberg and Nivre, 2013\]](#)

Результаты

Method	UAS	LAS
Greedy	89.88	87.69
Dynamic oracle	90.96	88.72
Beam search	93.5	91.9

(Stanford basic dependencies, WSJ 23)

UAS = Unlabeled Attachment Score; LAS = Labeled Attachment Score

- ▶ [\[Goldberg and Nivre, 2013\]](#)
- ▶ [\[Zhang and Nivre, 2011\]](#)

Результаты

Method	UAS	LAS
Greedy	89.88	87.69
Dynamic oracle	90.96	88.72
Beam search	93.5	91.9
LSTM + dyn. oracle	93.56	91.42

(Stanford basic dependencies, WSJ 23)

UAS = Unlabeled Attachment Score; LAS = Labeled Attachment Score

- ▶ [\[Goldberg and Nivre, 2013\]](#)
- ▶ [\[Zhang and Nivre, 2011\]](#)
- ▶ [\[Ballesteros et al., 2016\]](#)

Прогресс

- ▶ Жадность
- ▶ Error propagation
- ▶ Цена разных ошибок

Прогресс

- ▶ Жадность
- ▶ Error propagation
- ▶ Цена разных ошибок

Learning to search

- ▶ *Политика* – это классификатор

Learning to search

- ▶ *Политика* – это классификатор
- ▶ Ожидаемая потеря $\mathbb{E}\mathcal{L}(c_i)$ после шага c_i .

Learning to search

- ▶ *Политика* – это классификатор
- ▶ Ожидаемая потеря $\mathbb{E}\mathcal{L}(c_i)$ после шага c_i .
- ▶ Regret для действия c_i

$$\mathbb{E}\mathcal{L}(c_i) - \min_c \mathbb{E}\mathcal{L}(c)$$

Learning to search

- ▶ *Политика* – это классификатор
- ▶ Ожидаемая потеря $\mathbb{E}\mathcal{L}(c_i)$ после шага c_i .
- ▶ Regret для действия c_i

$$\mathbb{E}\mathcal{L}(c_i) - \min_c \mathbb{E}\mathcal{L}(c)$$

- ▶ Оракул — это политика (доступная только на обучающих данных)

Search

Search + Learn

- ▶ Имеем классификатор политику π
- ▶ Применяем, получаем цепочку состояний s_0, \dots, s_n .

Search

Search + Learn

- ▶ Имеем классификатор политику π
- ▶ Применяем, получаем цепочку состояний s_0, \dots, s_n .
- ▶ Задача для нового классификатора: в каждом состоянии, выбрать переход с minimal regret

Search

Search + Learn

- ▶ Имеем классификатор политику π
- ▶ Применяем, получаем цепочку состояний S_0, \dots, S_n .
- ▶ Задача для нового классификатора: в каждом состоянии, выбрать переход с minimal regret
- ▶ Обучаем новый cost-sensitive классификатор π' , $\text{cost} = \text{regret}$

Search + Learn

- ▶ Имеем классификатор политику π
- ▶ Применяем, получаем цепочку состояний S_0, \dots, S_n .
- ▶ Задача для нового классификатора: в каждом состоянии, выбрать переход с minimal regret
- ▶ Обучаем новый cost-sensitive классификатор π' , $\text{cost} = \text{regret}$
- ▶ $\pi = (1 - \alpha)\pi + \alpha\pi'$

Search + Learn

- ▶ Имеем классификатор политику π
- ▶ Применяем, получаем цепочку состояний S_0, \dots, S_n .
- ▶ Задача для нового классификатора: в каждом состоянии, выбрать переход с minimal regret
- ▶ Обучаем новый cost-sensitive классификатор π' , cost = regret
- ▶ $\pi = (1 - \alpha)\pi + \alpha\pi'$

Начальная политика — оракул

Прогресс

- ▶ Жадность
- ▶ Error propagation
- ▶ Цена разных ошибок

Ключевые слова

- ▶ Structured learning, Learning to search
- ▶ Structured Perceptron / SVM
- ▶ Searn, Dagger, AggreVaTe, LOLS
- ▶ Reinforcement learning
- ▶ Vowpal Wabbit

<http://nlpers.blogspot.ru/2016/03/a-dagger-by-any-other-name-scheduled.html>

Далее,

Итоги

План

Деревья зависимостей

Transition-based parsing

ML for transition-based parsing

Структурированное обучение

Итоги

Парсер

- ▶ Arc-eager
- ▶ Features (см. статью).
- ▶ Perceptron loss

Парсер

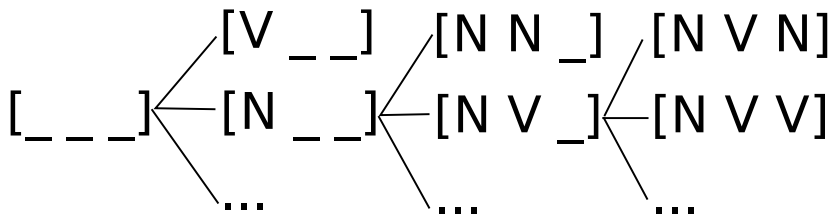
- ▶ Arc-eager
- ▶ Features (см. статью).
- ▶ Perceptron loss
- ▶ +Structured
- ▶ +Beam search
- ▶ +Early update
- ▶ +Max-violation (по желанию)

Парсер

- ▶ Arc-eager
- ▶ Features (см. статью).
- ▶ Perceptron loss
- ▶ +Dynamic oracle

Таггер

Система переходов:



"John saw Mary"

Tarrep

Features

- ▶ Not rare: w_i
- ▶ Rare: prefix $w_i[:N]$, $N \leq 4$; suffix $w_i[-N:]$, $N \leq 4$;
I(w contains number); I(w contains uppercase char);
I(w contains hyphen)
- ▶ Every word: t_{i-1} ; $t_{i-2}t_{i-1}$; w_{i-1} ; w_{i-2} ; w_{i+1} ; w_{i+2}

w – word, t – POS-tag.

Таггер

- ▶ Perceptron loss
- ▶ +Structured
- ▶ +Beam search
- ▶ +Early update (по желанию)
- ▶ +Max-violation (по желанию)

[Collins, 2002]

[Ratnaparkhi, 1996]

Таггер + Парсер

Pipeline:

1. POS-tagger
2. Dependency parser

Проблема:

Таггер + Парсер

Pipeline:

1. POS-tagger
2. Dependency parser

Проблема:

- Error propagation

Tagger + Parser

Joint tagging & parsing.

- ▶ S
- ▶ R
- ▶ $S_{RA}(r)$
- ▶ $R_{LA}(r)$

E.g. [Bohnet and Nivre, 2012]

Tagger + Parser

Joint tagging & parsing.

- ▶ $S(t)$
- ▶ R
- ▶ $S_{RA}(r, t)$
- ▶ $R_{LA}(r)$

E.g. [Bohnet and Nivre, 2012]

Tagger + Parser

Joint transition-based...

- ▶ POS-tagging + Parsing
- ▶ CJK segmentation + POS-tagging + Parsing
- ▶ Parsing + Dysfluency detection

Таггер + Парсер

Joint transition-based...

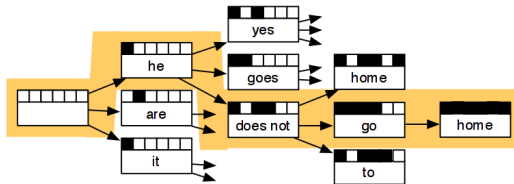
- ▶ POS-tagging + Parsing
- ▶ CJK segmentation + POS-tagging + Parsing
- ▶ Parsing + Dysfluency detection

Зачем:

- ▶ Больше информации о тегах в синтаксисе
- ▶ Выбираем теги, с которыми складывается хороший разбор

Машинный перевод

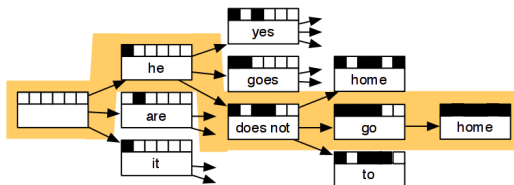
Система переходов:



- Разное число шагов от начала до конца

Машинный перевод

Система переходов:



- ▶ Разное число шагов от начала до конца
- ▶ Нужны стеки для того, чтобы сравнивать пути
- ▶ Нужен future cost estimation
- ▶ И т.д.

Машинный перевод

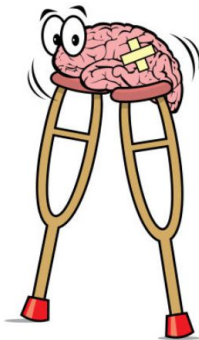
- ▶ Фичи (см. статью)
- ▶ Perceptron loss
- ▶ +Structured
- ▶ +Beam search
- ▶ +Max-violation

[Yu et al., 2013]

Вопросы?

Бонус:

Hack of the Day



Averaged SGD

► Обычный SGD:

$$w_t = w_{t-1} - \alpha \cdot \nabla \mathcal{L}(w_{t-1})$$

$$w_{\text{final}} = w_T$$

Averaged SGD

- ▶ Обычный SGD:

$$w_t = w_{t-1} - \alpha \cdot \nabla \mathcal{L}(w_{t-1})$$

$$w_{\text{final}} = w_T$$

- ▶ Averaged SGD:

$$w_t = w_{t-1} - \alpha \cdot \nabla \mathcal{L}(w_{t-1})$$

$$w_{\text{final}} = \frac{1}{T} \sum w_t$$

Averaged SGD

- ▶ Обычный SGD:

$$w_t = w_{t-1} - \alpha \cdot \nabla \mathcal{L}(w_{t-1})$$

$$w_{\text{final}} = w_T$$

- ▶ Averaged SGD:

$$w_t = w_{t-1} - \alpha \cdot \nabla \mathcal{L}(w_{t-1})$$

$$w_{\text{final}} = \frac{1}{T} \sum w_t$$

Менее хардкорно: усреднять несколько последних чекпоинтов.

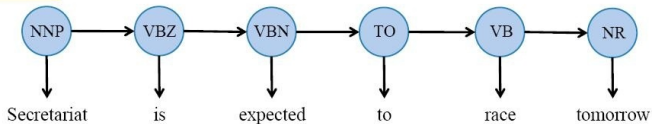
Бонус:

HMM, MEMM & CRF

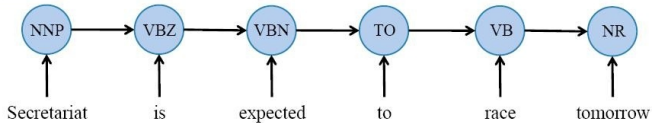
HMM vs MEMM vs CRF

HMM v.s. MEMM

HMM



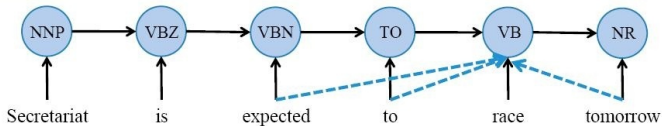
MEMM



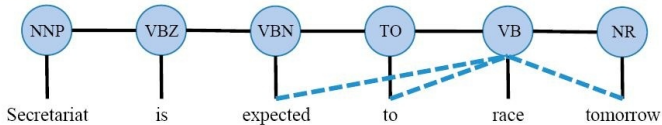
HMM vs MEMM vs CRF

MEMM v.s. CRF

MEMM



CRF

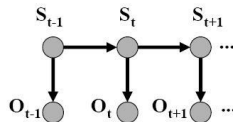


HMM vs MEMM vs CRF

Summary

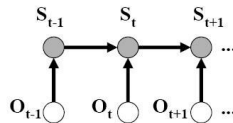
$$\vec{s} = s_1, s_2, \dots, s_n \quad \vec{o} = o_1, o_2, \dots, o_n$$

HMM $P(\vec{s}, \vec{o}) \propto \prod_{t=1}^{|\vec{o}|} P(s_t | s_{t-1}) P(o_t | s_t)$

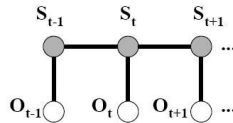


MEMM $P(\vec{s} | \vec{o}) \propto \prod_{t=1}^{|\vec{o}|} P(s_t | s_{t-1}, o_t)$

$$\propto \prod_{t=1}^{|\vec{o}|} \frac{1}{Z_{s_{t-1}, o_t}} \exp \left(\sum_j \lambda_j f_j(s_t, s_{t-1}) + \sum_k \mu_k g_k(s_t, x_t) \right)$$



CRF $P(\vec{s} | \vec{o}) \propto \frac{1}{Z_{\vec{o}}} \prod_{t=1}^{|\vec{o}|} \exp \left(\sum_j \lambda_j f_j(s_t, s_{t-1}) + \sum_k \mu_k g_k(s_t, x_t) \right)$



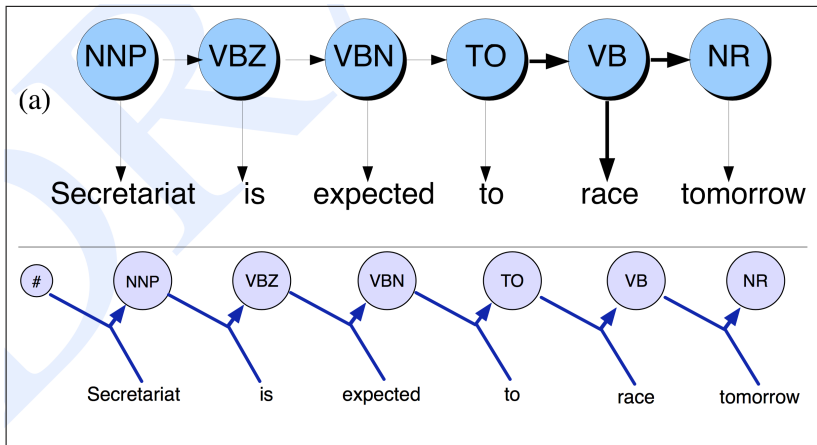
Что произошло



HMM vs MEMM vs CRF

- ▶ Structured perceptron: не нормализуем ничего
- ▶ CRF: нормализуем *пути*
- ▶ MEMM: нормализуем *переходы*
- ▶ HMM: нормализуем *переходы И порождение*

MEMM



Бонус:

Sparse features & Hashing trick

Feature template

$$I(S_0 = w) \cdot I(N_0 = v) \quad \forall w, v$$

Feature template

$$I(S_0 = w) \cdot I(N_0 = v) \quad \forall w, v$$

- ▶ Просматриваем обучающий корпус.
- ▶ Собираем все встречающиеся фичи.
- ▶ Записываем индекс в хеш-таблицу.
- ▶ ...

Feature template

$$I(S_0 = w) \cdot I(N_0 = v) \quad \forall w, v$$

- ▶ Просматриваем обучающий корпус.
- ▶ Собираем все встречающиеся фичи.
- ▶ Записываем индекс в хеш-таблицу.
- ▶ ...

Проблемы:

- ▶ Сложно
- ▶ Плохо (не встретили много комбинаций)

Мотивация

$$I(S_0 = w) \cdot I(N_0 = v) \quad \forall w, v$$

К чему мы привыкли

- ▶ Есть фичи i
- ▶ Есть вектор фичей f .
- ▶ Есть вектор параметров w
- ▶ $f[i] = 1$

К чему мы привыкли

- ▶ Есть фичи i
- ▶ Есть вектор фичей f .
- ▶ Есть вектор параметров w
- ▶ $f[i] = 1$
- ▶ $a + b$
- ▶ αv
- ▶ $\tanh(v)$
- ▶ (f, w)

К чему мы привыкли

- ▶ `int i`
- ▶ `vector<float> f`
- ▶ `vector<float> w`
- ▶ `f[i] = 1`

К чему мы привыкли

- ▶ `int i`
- ▶ `vector<float> f`
- ▶ `vector<float> w`
- ▶ `f[i] = 1`
- ▶ `c[i] = a[i] + b[i]`
- ▶ `u[i] = alpha * v[i]`
- ▶ `u[i] = tanh(v[i])`
- ▶ `c += f[i] * w[i]`

Sparse features

- ▶ `string s`

Свойства:

- ▶ К примеру, `s = "S0=a|N0=cat"`

Sparse features

- ▶ `string s`
- ▶ `map<string, float> f`
- ▶ `map<string, float> w`

Свойства:

- ▶ К примеру, `s = "S0=a|N0=cat"`
- ▶ 0 по умолчанию

Sparse features

- ▶ `string s`
- ▶ `map<string, float> f`
- ▶ `map<string, float> w`
- ▶ `f[s] = 1`

Свойства:

- ▶ К примеру, `s = "S0=a|N0=cat"`
- ▶ 0 по умолчанию

Sparse features

- ▶ `string s`
- ▶ `map<string, float> f`
- ▶ `map<string, float> w`
- ▶ `f[s] = 1`
- ▶ `c[s] = a.get(s, 0) + b.get(s, 0)`

Свойства:

- ▶ К примеру, `s = "S0=a|N0=cat"`
- ▶ 0 по умолчанию

Sparse features

- ▶ `string s`
- ▶ `map<string, float> f`
- ▶ `map<string, float> w`
- ▶ `f[s] = 1`
- ▶ `c[s] = a.get(s, 0) + b.get(s, 0)`
- ▶ `u[s] = alpha * v[s]`
- ▶ `u[s] = tanh(v[s])`

Свойства:

- ▶ К примеру, `s = "S0=a|N0=cat"`
- ▶ 0 по умолчанию

Sparse features

- ▶ `string s`
- ▶ `map<string, float> f`
- ▶ `map<string, float> w`
- ▶ `f[s] = 1`
- ▶ `c[s] = a.get(s, 0) + b.get(s, 0)`
- ▶ `u[s] = alpha * v[s]`
- ▶ `u[s] = tanh(v[s])`
- ▶ `c += f.get(s, 0) * w.get(s, 0)`

Свойства:

- ▶ К примеру, `s = "S0=a|N0=cat"`
- ▶ 0 по умолчанию

Sparse features

- ▶ `string s`
- ▶ `map<string, float> f`
- ▶ `map<string, float> w`
- ▶ `f[s] = 1`
- ▶ `c[s] = a.get(s, 0) + b.get(s, 0)`
- ▶ `u[s] = alpha * v[s]`
- ▶ `u[s] = tanh(v[s])`
- ▶ `c += f.get(s, 0) * w.get(s, 0)`

Свойства:

- ▶ К примеру, `s = "S0=a|N0=cat"`
- ▶ 0 по умолчанию
- ▶ Большинство фичей – нули
- ▶ Храним только то, что не равно нулю

Sparse features

- ▶ `string s`
- ▶ `map<string, float> f`
- ▶ `map<string, float> w`
- ▶ `f[s] = 1`
- ▶ `c[s] = a.get(s, 0) + b.get(s, 0)`
- ▶ `u[s] = alpha * v[s]`
- ▶ `u[s] = tanh(v[s])`
- ▶ `c += f.get(s, 0) * w.get(s, 0)`

Свойства:

- ▶ К примеру, `s = "S0=a|N0=cat"`
- ▶ 0 по умолчанию
- ▶ Большинство фичей – нули
- ▶ Храним только то, что не равно нулю
- ▶ Не надо перечислять заранее

Hashing trick

```
N = 2 ** 20
```

```
class HashTable:
    def __init__(self):
        self.array = np.zeros(N)

    def get(self, feat_str):
        return self.array[HASH(feat_str) % N]

    def set(self, feat_str, value):
        self.array[HASH(feat_str) % N] = value
```

Hashing trick

```
N = 2 ** 20
```

```
class HashTable:
    def __init__(self):
        self.array = np.zeros(N)

    def get(self, feat_str):
        return self.array[HASH(feat_str) % N]

    def set(self, feat_str, value):
        self.array[HASH(feat_str) % N] = value
```

- ▶ Коллизии
- ▶ Не надо заранее перечислять фичи
- ▶ Быстрее

Hashing trick

```
N = 2 ** 20
```

```
class HashTable:
    def __init__(self):
        self.array = np.zeros(N)

    def get(self, feat_str):
        return self.array[HASH(feat_str) % N]

    def set(self, feat_str, value):
        self.array[HASH(feat_str) % N] = value
```

- ▶ Коллизии
- ▶ Не надо заранее перечислять фичи
- ▶ Быстрее
- ▶ Модель фиксированного размера