

# Averaged perceptron tagger

Frankie Robertson  
`frrobert@student.jyu.fi`

University of Jyväskylä

<https://www.github.com/frankier/perceptron-tagger-slides/>

18th of July, 2016



- ▶ Averaged structured perceptron with beam search
- ▶ A DSL for features
- ▶ Results for Kazakh

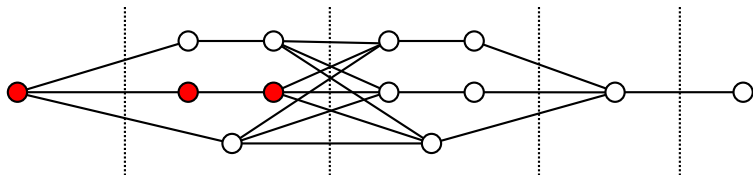
# Averaged structured perceptron with beam search 1/2

- ▶ **Structured prediction** is breaking down prediction of a large structure into subproblems and using results from earlier stages in later stages (like dynamic programming). For POS tagging the simplest approach is break the problem down to per-token tagging and then tag left to right.
- ▶ **The perceptron** stores a sparse linear vector of (feature, weight) pairs. Observations are scored by taking the dot product of their feature vector with the perceptron. Training is done trying a prediction, and updating incorrect weights by reinforcing or penalising them depending on whether they correspond with the correct observation.
- ▶ With the **structured perceptron** the score from each update is accumulated to get the score of the (partial) output.

# Averaged structured perceptron with beam search 2/2

- ▶ The perceptron only converges for linearly separable inputs, otherwise its weights oscillate. To get the best weights the perceptron is trained for several iterations on reshuffled observations and the **averaged** weights across the whole training period are saved as the final weights.
- ▶ With **beam search**, the n-best candidates are considered and updated at each stage as opposed to a pure-greedy strategy which would only keep one intermediate result.

# Eurovision    ән    конкурсы 2010    .



Eurovision<np><al><nom>

ән<n><nom> +e<cop><aor><p3><pl>

ән<n><nom> +e<cop><aor><p3><sg>

ән<n><nom>

конкурс<n><px3sp><nom>

конкурс<n><px3sp><nom> +e<cop><aor><p3><pl>

конкурс<n><px3sp><nom> +e<cop><aor><p3><sg>

2010<num><subst><nom>

.<sent>

# A DSL for features 1/3

spectie: How's it going?

frankier: Been coding like a maniac

frankier: I've ended up creating a sort of lisp in XML  
based on a stack VM

spectie: 0\_\_0

## A DSL for features 2/3



# A DSL for features 3/3

- ▶ Greenspun's tenth rule of programming: Any sufficiently complicated C or Fortran program contains an ad hoc, informally-specified, bug-ridden, slow implementation of half of Common Lisp.
- ▶ Neither easy nor desirable to try and define every possible useful feature up front.
- ▶ Seems likely that any feature language will be subject to feature creep (pun intended) so an expression language is a reasonable starting point.
- ▶ Bytecode interpreter design is faster and more modular than a tree walking interpreter and is usually the same or less code. Also it's easy to serialise.
- ▶ Hopefully the flexibility empowers language authors more than it confuses them. (In the worst case people can just copy-paste.)



# Example of DSL 1/2

```
<def-global as='major_tag_0'>
  <subscript idx='0'>
    <ex-tags>
      <ex-wordoid><wrdaddr /></ex-wordoid>
    </ex-tags>
  </subscript>
</def-global>
```

```
<def-global as='is_dmorph'>
  <neq>
    <int val='0' />
    <wrddidx />
  </neq>
</def-global>
```

# Example of DSL 2/2

```
<feat>
  <pred><var name='is_headword' /></pred>
  <out><var name='major_tag_0' /></out>
</feat>
```

```
<feat>
  <pred><var name='is_headword' /></pred>
  <out><var name='lemma_0' /></out>
  <out><var name='major_tag_0' /></out>
</feat>
```

```
<feat>
  <pred><var name='is_dmorph' /></pred>
  <out><var name='headword_major_tag_0' /></out>
  <out><var name='major_tag_0' /></out>
</feat>
```

# Results for Kazakh 1/2

Sample mean + std-dev from 10-fold cross validation.

	Acc	Adj acc
0-gram	72.08%	77.72%
CG->0-gram	81.56%	87.95%
Best unigram	$82.45 \pm 3.80\%$	$88.99 \pm 2.83\%$
CG->Best unigram	$84.71 \pm 3.54\%$	$91.43 \pm 2.26\%$

# Results for Kazakh 2/2

No It-toolbox	Acc
Uni mtx	$91.08 \pm 2.13\%$
CG->Uni mtx	$87.47 \pm 3.10$
Kaz mtx	$91.41 \pm 2.08\%$
CG->Kaz mtx	$87.68 \pm 3.11\%$

It-toolbox	Acc	Adj acc
Uni mtx	$82.87 \pm 3.45\%$	$89.45 \pm 2.46\%$
CG->Uni mtx	$84.53 \pm 3.59\%$	$91.23 \pm 2.40\%$
Kaz mtx	$83.01 \pm 3.78\%$	$89.58 \pm 2.70\%$
CG->Kaz mtx	$84.57 \pm 3.57\%$	$91.27 \pm 2.37\%$

- ▶ Extend globals to macros/templates.
- ▶ Constructor for scanning/selecting a previous or subsequent wordoid or surface form or wordoid based on a predicate, eg to get the previous verb.
- ▶ Allow easy generation of multiple prefix/postfix features with special construct
- ▶ Improve handling of sentinels and ability to stop a feature firing when data isn't available.
- ▶ Allow reuse of coarse tags from tsx as well as provide some kind of similar tag coarsening

- ★ *Syntactic Processing using the Generalized Perceptron and Beam Search*. Zhang & Clark 2011. Includes general formulation and multiple worked examples. Most of the implementation is based on this paper.
- ▶ *Discriminative Training Methods for Hidden Markov Models*. Michael Collins 2002. Original Collins perceptron paper.
- ▶ *A Good Part-of-Speech Tagger in about 200 Lines of Python*, Matthew Honnibal (blog post). Plus: Easily understandable reference implementation. Minus: Formulation varies from rest of the literature.