Hierarchical Recognition of Propositional Arguments with Perceptrons

Xavier Carreras and **Lluís Màrquez**
TALP Research Centre
Technical University of Catalonia (UPC)
{carreras,lluism}@lsi.upc.es

Grzegorz Chrupała
GRIAL Research Group
University of Barcelona (UB)
grzegorz@pithekos.net

1 Introduction

We describe a system for the CoNLL-2004 Shared Task on Semantic Role Labeling (Carreras and Màrquez, 2004a). The system implements a two-layer learning architecture to recognize arguments in a sentence and predict the role they play in the propositions. The exploration strategy visits possible arguments bottom-up, navigating through the clause hierarchy. The learning components in the architecture are implemented as Perceptrons, and are trained simultaneously online, adapting their behavior to the global target of the system. The learning algorithm follows the global strategy introduced in (Collins, 2002) and adapted in (Carreras and Màrquez, 2004b) for partial parsing tasks.

2 Semantic Role Labeling Strategy

The strategy for recognizing propositional arguments in sentences is based on two main observations about argument structure in the data. The first observation is the relation of the arguments of a proposition with the chunk and clause hierarchy: a proposition places its arguments in the clause directly containing the verb (local clause), or in one of the ancestor clauses. Given a clause, we define the sequence of *top-most* syntactic elements as the words, chunks or clauses which are directly rooted at the clause. Then, arguments are formed as subsequences of top-most elements of a clause. Finally, for local clauses arguments are found strictly to the left or to the right of the target verb, whereas for ancestor clauses arguments are usually to the left of the verb. This observation holds for most of the arguments in the data. A general exception are arguments of type V, which are found only in the local clause, starting at the position of the target verb.

The second observation is that the arguments of all propositions of a sentence do not cross their boundaries, and that arguments of a particular proposition are usually found strictly within an argument of a higher level proposition. Thus, the problem can be thought of as finding a hierarchy of arguments in which arguments are embedded inside others, and each argument is related to a number of propositions of a sentence in a particular role. If an argument is related to a certain verb, no other argument linking to the same verb can be found within it.

The system presented in this paper translates these observations into constraints which are enforced to hold in a solution, and guide the recognition strategy. A limitation of the system is that it makes no attempt to recognize arguments which are split in many phrases.

In what follows, x is a sentence, and x_i is the i-th word of the sentence. We assume a mechanism to access the input information of x (PoS tags, chunks and clauses), as well as the set of target verbs V, represented by their position. A solution $y \in \mathcal{Y}$ for a sentence x is a set of arguments of the form $(s,e)_v^k$, where (s,e) represents an argument spanning from word x_s to word x_e, playing a semantic role $k \in \mathcal{K}$ with a verb $v \in V$. Finally, $[S,E]$ denotes a clause spanning from word x_S to word s_E.

The $\mathrm{SRL}(x)$ function, predicting semantic roles of a sentence x, implements the following strategy:

1. Initialize set of arguments, $\mathcal{A}$, to empty.
2. Define the level of each clause as its distance to the root clause.
3. Explore clauses bottom-up, i.e. from deeper levels to the root clause. For a clause $[S,E]$:
   $\mathcal{A} := \mathcal{A} \cup \mathrm{arg\_search}(x, [S,E])$
4. Return $\mathcal{A}$

2.1 Building Argument Hierarchies

Here we describe the function $\mathrm{arg\_search}$, which builds a set of arguments organized hierarchically, within a clause $[S,E]$ of a sentence x. The function makes use of two learning-based components, defined here and described below. First, a filtering function F, which, given a candidate argument, determines its plausible categories, or rejects it when no evidence for it being an argument is found. Second, a set of $k\text{-}\mathrm{score}$ functions, for each

$k \in \mathcal{K}$, which, given an argument, predict a score of plausibility for it being of role type k of a certain proposition.

The function $\mathrm{arg\_search}$ searches for the argument hierarchy which optimizes a global score on the hierarchy. As in earlier works, we define the global score (Γ) as the summation of scores of each argument in the hierarchy. The function explores all possible arguments in the clause formed by contiguous top-most elements, and selects the subset which optimizes the global score function, forcing a hierarchy in which the arguments linked to the same verb do not embed.

Using dynamic programming, the function can be computed in cubic time. It considers fragments of top-most elements, which are visited bottom-up, incrementally in length, until the whole clause is explored. While exploring, it maintains a two-dimensional matrix A of partial solutions: each position $[s, e]$ contains the optimal argument hierarchy for the fragment from s to e. Finally, the solution is found at $\mathrm{A}[S, E]$. For a fragment from s to e the algorithm is as follows:

```
1. $\mathcal{A} := \mathrm{A}[s,r] \cup \mathrm{A}[r{+}1,e]$ where
       $r := \arg\max_{s \le r < e} \Gamma\big(\mathrm{A}[s,r]\big) + \Gamma\big(\mathrm{A}[r{+}1,e]\big)$
2. For each prop $v \in V$ :
   (a) $K := \mathrm{F}((s,e),v)$
   (b) Compute $k'$ such that
           $k' := \arg\max_{k \in \mathcal{K}} k\text{-score}((s,e),v,x)$
       Set $\gamma$ to the score of category $k'$.
   (c) Set $\mathcal{A}_v$ as the arguments in $\mathcal{A}$ linked to $v$.
   (d) If $\big(\Gamma(\mathcal{A}_v) < \gamma\big)$ then $\mathcal{A} := \mathcal{A} \setminus \mathcal{A}_v \cup \{(s,e)_v^{k'}\}$
3. $\mathrm{A}[s,e] := \mathcal{A}$
```

Note that an argument is visited once, and that its score can be stored to efficiently compute the Γ global score.

2.2 Start-End Filtering

The function F determines which categories in $\mathcal{K}$ are plausible for an argument (s, e) to relate to a verb v.

This is done via *start-end* filters (F_S^k and F_E^k), one for each type in $\mathcal{K}$[1]. They operate on words, independently of verbs, deciding whether a word is likely to start or end some argument of role type k.

The selection of categories is conditional to the relative level of the verb and the clause, and to the relative position of the verb and the argument. The conditions are:

- v is local to the clause, and $(v{=}s)$ and $\mathrm{F}_\mathrm{E}^\mathrm{V}(x_e)$:
  $K := \{\mathrm{V}\}$
- v is local, and $(e{<}v \vee v{<}s)$:
  $K := \{k \in \mathcal{K} \mid \mathrm{F}_\mathrm{S}^k(x_s) \wedge \mathrm{F}_\mathrm{E}^k(x_e)\}$
- v is at deeper level, and $(e{<}v)$:
  $K := \{k \in \mathcal{K} \mid k \notin K(v) \wedge \mathrm{F}_\mathrm{S}^k(x_s) \wedge \mathrm{F}_\mathrm{E}^k(x_e)\}$
  where $K(v)$ is the set of categories already assigned to the verb in deeper clauses.
- Otherwise, K is set to empty.

Note that setting K to empty has the effect of filtering out the argument for the proposition. Note also that Start-End classifications do not depend on the verb, thus they can be performed once per candidate word, before entering the exploration of clauses. Then, when visiting a clause, the Start-End filtering can be performed with stored predictions.

3 Learning with Perceptrons

In this section we describe the learning components of the system, namely *start*, *end* and *score* functions, and the Perceptron-based algorithm to train them together online.

Each function is implemented using a linear separator, $\mathrm{h}_\mathbf{w} : \mathcal{R}^n \rightarrow \mathcal{R}$, operating in a feature space defined by a feature extraction function, $\phi : \mathcal{X} \rightarrow \mathcal{R}^n$, for some instance space $\mathcal{X}$. The *start-end* functions (F_S^k and F_E^k) are formed by a prediction vector for each type, noted as $\mathbf{w}_\mathrm{S}^k$ or $\mathbf{w}_\mathrm{E}^k$, and a shared representation function ϕ_w which maps a word in context to a feature vector. A prediction is computed as $\mathrm{F}_\mathrm{S}^k(x) = \mathbf{w}_\mathrm{S}^k \cdot \phi_\mathrm{w}(x)$, and similarly for the F_E^k, and the sign is taken as the binary classification.

The *score* functions compute real-valued scores for arguments $(s, e)_v$. We implement these functions with a prediction vector $\mathbf{w}^k$ for each type $k \in \mathcal{K}$, and a shared representation function ϕ_a which maps an argument-verb pair to a feature vector. The score prediction for a type k is then given by the expression: $k\text{-score}((s,e),v,x) = \mathbf{w}^k \cdot \phi_\mathrm{a}((s,e),v,x)$.

3.1 Perceptron Learning Algorithm

We describe a mistake-driven online algorithm to train prediction vectors together. The algorithm is essentially the same as the one introduced in (Collins, 2002). Let W be the set of prediction vectors:

```
• Initialize: $\forall \mathbf{w} {\in} W \;\; \mathbf{w} := 0$
• For each epoch $t := 1 \ldots T$,
  for each sentence-solution pair $(x, y)$ in training:
     1. $\hat{y} = \mathrm{SRL}_W(x)$
     2. $\mathrm{learning\_feedback}(W, x, y, \hat{y})$
• Return $W$
```

3.2 Learning Feedback for Filtering-Ranking

We now describe the learning feedback rule, introduced in earlier works (Carreras and Màrquez, 2004b). We differentiate two kinds of global errors in order to give feedback to the functions being learned: missed arguments and over-predicted arguments. In each case, we identify

[1] Actually, we share *start-end* filters for A0-A5 arguments.

the prediction vectors responsible for producing the incorrect argument and update them additively: vectors are moved towards instances predicted too low, and moved away from instances predicted too high.

Let y^* be the gold set of arguments for a sentence x, and $\hat{y}$ those predicted by the SRL function. Let $\mathrm{goldS}(x_i, k)$ and $\mathrm{goldE}(x_i, k)$ be, respectively, the perfect indicator functions for *start* and *end* boundaries of arguments of type k. That is, they return 1 if word x_i starts/ends some k-argument in y^* and -1 otherwise. The feedback is as follows:

- Missed arguments: $\forall (s,e)_v^k \in \ y^* \backslash \hat{y}$:
    1. Update misclassified boundary words:
       if $(\mathbf{w}_\mathrm{S}^k \cdot \phi_\mathrm{w}(x_s) \leq 0)$ then $\mathbf{w}_\mathrm{S}^k = \mathbf{w}_\mathrm{S}^k + \phi_\mathrm{w}(x_s)$
       if $(\mathbf{w}_\mathrm{E}^k \cdot \phi_\mathrm{w}(x_e) \leq 0)$ then $\mathbf{w}_\mathrm{E}^k = \mathbf{w}_\mathrm{E}^k + \phi_\mathrm{w}(x_e)$
    2. Update score function, if applied:
       if $(k \in F((s,e), v)$ then
       $\mathbf{w}^k = \mathbf{w}^k + \phi_\mathrm{a}((s,e), v, x)$
- Over-predicted arguments: $\forall (s,e)_p^k \in \ \hat{y} \backslash y^*$:
    1. Update score function:
       $\mathbf{w}^k = \mathbf{w}^k - \phi_\mathrm{a}((s,e), v, x)$
    2. Update words misclassified as S or E:
       if $(\mathrm{goldS}(x_s, k) = -1)$ then $\mathbf{w}_\mathrm{S}^k = \mathbf{w}_\mathrm{S}^k - \phi_\mathrm{w}(x_s)$
       if $(\mathrm{goldE}(x_e, k) = -1)$ then $\mathbf{w}_\mathrm{E}^k = \mathbf{w}_\mathrm{E}^k - \phi_\mathrm{w}(x_e)$

3.3 Kernel Perceptrons with Averaged Predictions

Our final architecture makes use of *Voted Perceptrons* (Freund and Schapire, 1999), which compute a prediction as an *average* of all vectors generated during training. Roughly, each vector contributes to the average proportionally to the number of correct positive training predictions the vector has made. Furthermore, a prediction vector can be expressed in dual form as a combination of training instances, which allows the use of kernel functions. We use standard polynomial kernels of degree 2.

4 Features

The features of the system are extracted from three types of elements: words, target verbs, and arguments. They are formed making use of PoS tags, chunks and clauses of the sentence. The functions ϕ_w and ϕ_a are defined in terms of a collection of feature extraction patterns, which are binarized in the functions: each extracted pattern forms a binary dimension indicating the existence of the pattern in a learning instance.

Extraction on Words. The list of features extracted from a word x_i is the following:

- **PoS tag**.
- **Form**, if the PoS tag does not match with the Perl regexp `/^(CD|FW|J|LS|N|POS|SYM|V)/`.
- **Chunk type**, of the chunk containing the word.
- **Binary-valued flags**: (a) Its chunk is one-word or multi-word; (b) Starts and/or ends, or is strictly within a chunk (3 flags); (c) Starts and/or ends clauses (2 flags); (d) Aligned with a target verb; and (e) First and/or last word of the sentence (2 flags).

Given a word x_i, the ϕ_w function implements a ± 3 window, that is, it returns the features of the words x_{i+r}, with $-3 \leq r \leq +3$, each with its relative position r.

Extraction on Target Verbs. Given a target verb v, we extract the following features from the word x_v:

- **Form**, **PoS tag**, and **target verb infinitive form**.
- **Voice** : *passive*, if x_v has PoS tag `VBN`, and either its chunk is not `VP` or x_v is preceded by a form of "to be" or "to get" within its chunk; otherwise *active*.
- **Chunk type**.
- **Binary-valued flags**: (a) Its chunk is multi-word or not; and (b) Starts and/or ends clauses (2 flags).

Extraction on Arguments. The ϕ_a function performs the following feature extraction for an argument (s,e) linked to a verb v:

- **Target verb features**, of verb v.
- **Word features**, of words $s-1$, s, e, and $e+1$, each anchored with its relative position.
- **Distance** of v to s and to e: for both pairs, a flag indicating if distance is $\{0, 1, -1, >1, <1\}$.
- **PoS Sequence**, of PoS tags from s to e: (a) n-grams of size 2, 3 and 4; and (b) the complete PoS pattern, if it is less than 5 tags long.
- **TOP sequence**: tags of the top-most elements found strictly from s to e. The tag of a word is its PoS. The tag of a chunk is its type. The tag of a clause is its type (`S`) enriched as follows: if the PoS tag of the first word matches `/^(IN|W|TO)/` the tag is enriched with the form of that word (e.g. `S-to`); if that word is a verb, the tag is enriched with its PoS (e.g. `S-VBG`); otherwise, it is just `S`. The following features are extracted: (a) n-grams of sizes 2, 3 and 4; (b) The complete pattern, if it is less than 5 tags long; and (c) Anchored tags of the first, second, penultimate and last elements.
- **PATH sequence**: tags of elements found between the argument and the verb. It is formed by a concatenation of horizontal tags and vertical tags. The horizontal tags correspond to the TOP sequence of elements at the same level of the argument, from it to the phrase containing the verb, both excluded. The vertical part is the list of tags of the phrases which contain the verb, from the phrase at the level of the

argument to the verb. The tags of the PATH sequence are extracted as in the TOP sequence, with an additional mark indicating whether an element is horizontal to the left or to the right of the argument, or vertical. The following features are extracted: (a) n-grams of sizes 4 and 5; and (b) The complete pattern, if it is less than 5 tags long.

- **Bag of Words**: we consider the top-most elements of the argument which are not clauses, and extract all nouns, adjectives and adverbs. We then form a separate bag for each category.
- **Lexicalization**: we extract the form of the head of the first top-most element of the argument, via common head word rules; if the first element is a PP chunk, we also extract the head of the first NP found.

5 Experiments and Results

We have build a system which implements the presented architecture for recognizing arguments and their semantic roles. The configuration of learning functions, related to the roles in the CoNLL-2004 data, is set as follows :

- Five score functions for the A0–A4 types, and two shared filtering functions $\mathrm{F}_{\mathrm{S}}^{\mathrm{AN}}$ and $\mathrm{F}_{\mathrm{E}}^{\mathrm{AN}}$.
- For each of the 13 adjunct types (AM-*), a score function and a pair of filtering functions.
- Three score functions for the R0–R2 types, and two filtering functions $\mathrm{F}_{\mathrm{S}}^{\mathrm{R}}$ and $\mathrm{F}_{\mathrm{E}}^{\mathrm{R}}$ shared among them.
- For verbs, a score function and an end filter.

We ran the learning algorithm on the training set (with predicted input syntax) with a polynomial kernel of degree 2, for up to 8 epochs. Table 1 presents the obtained results on the development set, either artificial or real. The second and third rows provide, respectively, the loss suffered because of errors in the filtering and scoring layer. The filtering layer performs reasonably well, since 89.44% recall can be achieved on the top of it. However, the scoring functions clearly moderate the performance, since working with perfect start-end functions only achieve an F_1 at 75.60. Finally, table 2 presents final detailed results on the test set.

	Precision	Recall	$\mathrm{F}_{\beta=1}$
g–F_{S}, g–F_{E}, g-score	99.92%	94.73%	97.26
F_{S}, F_{E}, g–score	99.90%	89.44%	94.38
g–F_{S}, g–F_{E}, score	85.12%	67.99%	75.60
F_{S}, F_{E}, score	73.40%	63.70%	68.21

Table 1: Overall results on the development set. Functions with prefix g are gold functions, providing bounds of our performance. The top row is the upper bound performance of our architecture. The bottom row is the real performance.

	Precision	Recall	$\mathrm{F}_{\beta=1}$
Overall	71.81%	61.11%	66.03
A0	81.83%	76.46%	79.05
A1	68.73%	65.27%	66.96
A2	59.41%	34.03%	43.28
A3	58.18%	21.33%	31.22
A4	72.97%	54.00%	62.07
A5	0.00%	0.00%	0.00
AM-ADV	54.50%	35.50%	43.00
AM-CAU	58.33%	28.57%	38.36
AM-DIR	64.71%	22.00%	32.84
AM-DIS	64.06%	57.75%	60.74
AM-EXT	100.00%	50.00%	66.67
AM-LOC	35.62%	22.81%	27.81
AM-MNR	50.89%	22.35%	31.06
AM-MOD	97.57%	95.25%	96.40
AM-NEG	90.23%	94.49%	92.31
AM-PNC	36.11%	15.29%	21.49
AM-PRD	0.00%	0.00%	0.00
AM-TMP	61.86%	48.86%	54.60
R-A0	78.85%	77.36%	78.10
R-A1	64.29%	51.43%	57.14
R-A2	100.00%	22.22%	36.36
R-A3	0.00%	0.00%	0.00
R-AM-LOC	0.00%	0.00%	0.00
R-AM-MNR	0.00%	0.00%	0.00
R-AM-PNC	0.00%	0.00%	0.00
R-AM-TMP	0.00%	0.00%	0.00
V	98.32%	98.24%	98.28

Table 2: Results on the test set

Acknowledgements

This research is supported by the European Commission (Meaning, IST-2001-34460) and the Spanish Research Department (Aliado, TIC2002-04447-C02). Xavier Carreras is supported by a grant from the Catalan Research Department.

References

Xavier Carreras and Lluís Màrquez. 2004a. Introduction to the CoNLL-2004 Shared Task: Semantic Role Labeling. In *Proceedings of CoNLL-2004*.

Xavier Carreras and Lluís Màrquez. 2004b. Online learning via global feedback for phrase recognition. In *Advances in Neural Information Processing Systems 16*. MIT Press.

M. Collins. 2002. Discriminative Training Methods for Hidden Markov Models: Theory and Experiments with Perceptron Algorithms. In *Proceedings of the EMNLP'02*.

Y. Freund and R. E. Schapire. 1999. Large Margin Classification Using the Perceptron Algorithm. *Machine Learning*, 37(3):277–296.