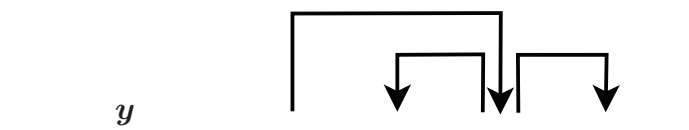
2
Structure Learning Framework

In this chapter, we describe the structure learning framework and some known extensions by detailing its application to the dependency parsing task. In Section 2.1, we formalize this task. Then, in Section 2.2, we present the *large margin* structure perceptron, an important extension to the SPerc algorithm. A powerful extension to the SL framework allows the introduction of *latent structures* that are optionally used as auxiliary information to the target task. We present the latent structure perceptron in Section 2.3. Finally, in Section 2.4, we depict empirical results of the structure perceptron on a Portuguese DP dataset, comparing its results with other state-of-the-art systems.

PUC-Rio - Certificação Digital Nº 0812627/CA

2.1
Dependency Parsing

Dependency parsing is to identify the structure underlying a sentence that describes the syntactic dependencies among its words. This structure is called *dependency tree* and is a rooted tree whose nodes are the words and punctuation marks in the sentence, i.e., the sentence tokens. Let $\boldsymbol{x} = (x_0, x_1, \ldots, x_N)$ be a sentence, where x_i is the i-th token and x_0 is an artificial token which is always the root of the dependency tree. For a given input sentence $\boldsymbol{x}$, the prediction output space $\mathcal{Y}(\boldsymbol{x})$ is the set of all rooted trees whose nodes are the tokens in $\boldsymbol{x}$ and the root node is x_0. For any dependency tree $\boldsymbol{y} \in \mathcal{Y}(\boldsymbol{x})$, we say that $(i, j) \in \boldsymbol{y}$ whenever token x_j modifies token x_i in the tree $\boldsymbol{y}$. The token x_i is called *head token* and the token x_j is called *modifier token*. Figure 2.1 shows the sentence John saw Mary, its dependency tree $\boldsymbol{y} = \{(0, 2), (2, 1), (2, 3)\}$, and the corresponding head tokens. The head



i	0	1	2	3
$\boldsymbol{x}$	*root*	John	saw	Mary
Head $(y(i))$	–	2	0	2

Figure 2.1: Dependency tree represented as arcs ($\boldsymbol{y}$) and as head vector.

vector is an alternative representation for the dependency tree, since each modifier token has exactly one head given by an incoming edge; except the artificial root token which has no head. We denote by $y(i)$ the head token of the modifier token x_i within the tree $\boldsymbol{y}$, that is $(y(i), i) \in \boldsymbol{y}$, for $i = 1, \ldots, N$.

MSTParser (McDonald et al., 2005, 2006; McDonald and Pereira, 2006) is a SL system that employs an online algorithm to train a dependency parser. MSTParser's prediction problem is formulated as a linear discriminative problem. MSTParser learns a $\boldsymbol{w}$-parameterized edge scoring function $s(\boldsymbol{x}, i, j; \boldsymbol{w})$ for every candidate edge (i, j) over $\boldsymbol{x}$, such that the prediction problem is to find the maximum-scoring rooted tree, that is

$$F(\boldsymbol{x}; \boldsymbol{w}) = \arg \max_{\boldsymbol{y} \in \mathcal{Y}(\boldsymbol{x})} \sum_{(i,j) \in \boldsymbol{y}} s(\boldsymbol{x}, i, j; \boldsymbol{w}). \tag{2-1}$$

This is equivalent to the well studied maximum branching problem that can be efficiently solved by Chu-Liu-Edmonds algorithm (Chu and Liu, 1965; Edmonds, 1967). There is also an improved version of this algorithm by Tarjan (1977).

Obviously, the edge scoring function is key for MSTParser and has to generalize over any possible input sentence. For a sentence $\boldsymbol{x}$ and a candidate edge (i, j), MSTParser determines $s(\boldsymbol{x}, i, j; \boldsymbol{w})$ by means of M real-valued features denoted by $\phi_m(\boldsymbol{x}, i, j)$, for $m = 1, \ldots, M$. These features describe the dependency between the head token x_i and the modifier token x_j in a meaningful and general way. MSTParser uses several *binary* features like, for instance, the i-th word, the j-th word, the part-of-speech of the i-th word, the distance from x_i to x_j, the relative order between x_i and x_j, among others. Then, the edge scoring function is defined as an weighted sum of the edge features

$$s(\boldsymbol{x}, i, j; \boldsymbol{w}) = \langle \boldsymbol{w}, \boldsymbol{\Phi}(\boldsymbol{x}, i, j) \rangle,$$

where $\boldsymbol{w} = (w_1, \ldots, w_M)$ comprises the model parameters, one for each feature; and $\boldsymbol{\Phi}(\boldsymbol{x}, i, j) = (\phi_1(\boldsymbol{x}, i, j), \ldots, \phi_M(\boldsymbol{x}, i, j))$ is the feature vector that describes the dependency edge (i, j). In that way, the learning problem is to determine a parameter vector $\boldsymbol{w}$ such that the predictor $F(\boldsymbol{x}; \boldsymbol{w})$ has small empirical risk on the training data and, at the same time, performs well on unseen data.

MSTParser's training algorithm is an extension of the margin infused relaxed algorithm, an online algorithm for multiclass problems. In this work, we use the averaged structure perceptron algorithm (Collins, 2002b) with a large margin extension (Fernandes and Brefeld, 2011; McAllester et al., 2011; Tsochantaridis et al., 2005) as the training algorithm for all tasks. We use SPerc even for DP, despite the fact that we use MSTParser's modeling for this

PUC-Rio - Certificação Digital Nº 0812627/CA

task.

2.2 Large Margin Training

The structure perceptron algorithm finds a classifier with no concern about its margin. However, it is well known that large margin classifiers provide better generalization performance on unseen data. We use a large-margin generalization of the structure perceptron that is based on the margin rescaling technique for SVM^{struct} (Altun et al., 2003; Tsochantaridis et al., 2005). During *training*, for an example $(\boldsymbol{x}, \boldsymbol{y})$, instead of the ordinary prediction problem (2-1), we use the following *loss-augmented* prediction function

$$F_\ell(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{w}) = \arg \max_{\boldsymbol{y}' \in \mathcal{Y}(\boldsymbol{x})} \left[\left(\sum_{(i,j) \in \boldsymbol{y}'} s(\boldsymbol{x}, i, j; \boldsymbol{w}) \right) + C \cdot \ell(\boldsymbol{y}, \boldsymbol{y}') \right], \qquad (2\text{-}2)$$

where $\ell(\boldsymbol{y}, \boldsymbol{y}') \geq 0$ is an appropriate loss function that measures the difference between a candidate tree $\boldsymbol{y}'$ and the correct one $\boldsymbol{y}$; and C is a constant to balance the weight between the loss function (margin) and the learned edge weights. We use a loss function that just counts how many incorrect edges are in the predicted tree $\boldsymbol{y}'$, which is given by

$$\ell(\boldsymbol{y}, \boldsymbol{y}') = \sum_{(i,j) \in \boldsymbol{y}'} \mathbf{1}[(i,j) \notin \boldsymbol{y}]. \qquad (2\text{-}3)$$

This loss function can be decomposed along the tree edges and we can thus rewrite the loss-augmented prediction function as

$$F_\ell(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{w}) = \arg \max_{\boldsymbol{y}' \in \mathcal{Y}(\boldsymbol{x})} \sum_{(i,j) \in \boldsymbol{y}'} \left(s(\boldsymbol{x}, i, j; \boldsymbol{w}) + C \cdot \mathbf{1}[(i,j) \notin \boldsymbol{y}] \right). \qquad (2\text{-}4)$$

This characteristic is desirable because we can define a loss-augmented *edge scoring* function as

$$s_\ell(\boldsymbol{x}, \boldsymbol{y}, i, j; \boldsymbol{w}) = s(\boldsymbol{x}, i, j; \boldsymbol{w}) + C \cdot \mathbf{1}[(i,j) \notin \boldsymbol{y}],$$

and then we have that

$$F_\ell(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{w}) = \arg \max_{\boldsymbol{y}' \in \mathcal{Y}(\boldsymbol{x})} \sum_{(i,j) \in \boldsymbol{y}'} s_\ell(\boldsymbol{x}, \boldsymbol{y}, i, j; \boldsymbol{w}),$$

which is a maximum branching problem just as the original prediction problem, but with modified edge weights. In that way, we can still use Chu-Liu-Edmonds algorithm in the large margin structure perceptron. We present the pseudo-code of the large-margin structure perceptron algorithm

PUC-Rio - Certificação Digital Nº 0812627/CA

in Figure 2.2. The unique modification to the SPerc algorithm is in the

$\boldsymbol{w}^0 \leftarrow \mathbf{0}$
$t \leftarrow 0$
while no convergence
  for each $(\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{D}$
    $\hat{\boldsymbol{y}} \leftarrow \arg\max_{\boldsymbol{y}' \in \mathcal{Y}(\boldsymbol{x})} \sum_{(i,j) \in \boldsymbol{y}'} \left(\langle \boldsymbol{w}^t, \boldsymbol{\Phi}(\boldsymbol{x}, i, j) \rangle + C \cdot \mathbf{1}[(i,j) \notin \boldsymbol{y}] \right)$
    $\boldsymbol{w}^{t+1} \leftarrow \boldsymbol{w}^t + \sum_{(i,j) \in \boldsymbol{y}} \boldsymbol{\Phi}(\boldsymbol{x}, i, j) - \sum_{(i,j) \in \hat{\boldsymbol{y}}} \boldsymbol{\Phi}(\boldsymbol{x}, i, j)$
    $t \leftarrow t + 1$
return $\frac{1}{t} \sum_{k=1}^{t} \boldsymbol{w}^k$

Figure 2.2: Large margin structure perceptron algorithm for dependency parsing.

computation of edge scores, where we add a constant C to the score of every incorrect edge. The constant C is a meta-parameter of this algorithm that allows us to balance the relative importance of the two components in the objective function. This parameter can be calibrated by means of cross-validation or a development set.

By using the loss-augmented prediction problem during training, an example $(\boldsymbol{x}, \boldsymbol{y})$ implies a model update whenever the current model does not respect the following margin constraint

$$s(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{w}) - s(\boldsymbol{x}, \boldsymbol{y}'; \boldsymbol{w}) \geq C \cdot \ell(\boldsymbol{y}, \boldsymbol{y}'), \ \forall \boldsymbol{y}' \in \mathcal{Y}(\boldsymbol{x}),$$

where $s(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{w}) = \sum_{(i,j) \in \boldsymbol{y}} s(\boldsymbol{x}, i, j; \boldsymbol{w})$ is the score of a whole tree. If a model respects this prediction margin, then the current predictor $F(\boldsymbol{x}; \boldsymbol{w})$ separates the correct output $\boldsymbol{y}$ from every alternative $\boldsymbol{y}' \in \mathcal{Y}(\boldsymbol{x})$ by a margin as large as $C \cdot \ell(\boldsymbol{y}, \boldsymbol{y}')$. In that way, the training algorithm incorporates *some* information about the structured empirical risk $\mathcal{R}_\ell(\mathcal{D}, \boldsymbol{w})$ of the current model, defined as

$$\mathcal{R}_\ell(\mathcal{D}, \boldsymbol{w}) = \sum_{(\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{D}} \ell(\boldsymbol{y}, F(\boldsymbol{x}; \boldsymbol{w})).$$

2.3 Latent Structure Training

For some structure learning problems, to directly predict an output $\boldsymbol{y}$ from the input $\boldsymbol{x}$ is a very hard problem, considering any meaningful feature representation. In some cases, if an appropriate auxiliary structure $\boldsymbol{h}$ is given for the input $\boldsymbol{x}$, the prediction problem becomes much easier. In coreference resolution, for instance, the input comprises a document containing a set of mentions to real world entities, like companies, places or people. The prediction

PUC-Rio - Certificação Digital Nº 0812627/CA

problem is then to cluster mentions that correspond to the same entity. Most clustering metrics lead to NP-hard optimization problems. Thus, we introduce coreference trees to represent a cluster of mentions. Giving such tree for an input document turns prediction into a polynomial problem.

Usually, the auxiliary structures are not explicitly given in the training data. Thus, we assume that these structures are *latent* and make use of the latent structure perceptron (Fernandes and Brefeld, 2011; Yu and Joachims, 2009). The original prediction problem is then split into two sub-problems: the *latent* prediction problem $F_h(\boldsymbol{x}; \boldsymbol{w}_h)$ and the *target* prediction problem $F_y(\boldsymbol{x}, \boldsymbol{h}; \boldsymbol{w}_y)$, where $\boldsymbol{w}_h$ is the latent model and $\boldsymbol{w}_y$ is the target model. In that way, the final prediction is performed by combining these two predictors

$$F(\boldsymbol{x}; \boldsymbol{w}_h, \boldsymbol{w}_y) = F_y(\boldsymbol{x}, F_h(\boldsymbol{x}; \boldsymbol{w}_h); \boldsymbol{w}_y).$$

Both the latent and the target predictors have the same form. Not surprisingly they are based on linear discriminative functions. The latent prediction function is given by

$$F_h(\boldsymbol{x}; \boldsymbol{w}_h) = \arg \max_{\boldsymbol{h} \in \mathcal{H}(\boldsymbol{x})} \langle \boldsymbol{w}_h, \boldsymbol{\Phi}_h(\boldsymbol{x}, \boldsymbol{h}) \rangle,$$

where $\mathcal{H}(\boldsymbol{x})$ is the set of feasible latent structures for the given input $\boldsymbol{x}$ and $\boldsymbol{\Phi}_h(\boldsymbol{x}, \boldsymbol{h})$ is an arbitrary joint feature vector representation of the input and the latent structure. The target prediction function is then defined as

$$F_y(\boldsymbol{x}, \boldsymbol{h}; \boldsymbol{w}_y) = \arg \max_{\boldsymbol{y} \in \mathcal{Y}(\boldsymbol{x}, \boldsymbol{h})} \langle \boldsymbol{w}_y, \boldsymbol{\Phi}_y(\boldsymbol{x}, \boldsymbol{h}, \boldsymbol{y}) \rangle,$$

where $\mathcal{Y}(\boldsymbol{x}, \boldsymbol{h})$ is the set of feasible output structures for the input $\boldsymbol{x}$ *and* the latent structure $\boldsymbol{h}$; and, $\boldsymbol{\Phi}_y(\boldsymbol{x}, \boldsymbol{h}, \boldsymbol{y})$ is an arbitrary joint feature vector representation of the input, the latent structure and the output structure.

In Figure 2.3, we depict the latent structure perceptron algorithm. The latent structure perceptron is very similar to the original SPerc, which, for each training instance, performs two major steps: (i) a prediction for the given input using the current model; and (ii) a model update based on the difference between the predicted and the ground truth outputs. The latent SPerc performs an additional step to predict the latent ground truth $\tilde{\boldsymbol{h}}$ using a specialization of the latent predictor.

Golden latent structures are usually not available in the training data. However, during training, for a given input $\boldsymbol{x}$, we have the golden output structure $\boldsymbol{y}$. Thus, we predict the *constrained latent structure* $\tilde{\boldsymbol{h}}$ for the training instance $(\boldsymbol{x}, \boldsymbol{y})$ using a specialization of the latent predictor – the *constrained*

PUC-Rio - Certificação Digital Nº 0812627/CA

$\boldsymbol{w}_0 \leftarrow \mathbf{0}$
$t \leftarrow 0$
while no convergence
  for each $(\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{D}$
    $\tilde{\boldsymbol{h}} \leftarrow \arg\max_{\boldsymbol{h} \in \mathcal{H}(\boldsymbol{x},\boldsymbol{y})} \langle \boldsymbol{w}_h^t, \boldsymbol{\Phi}_h(\boldsymbol{x}, \boldsymbol{h}) \rangle$
    $\hat{\boldsymbol{h}} \leftarrow \arg\max_{\boldsymbol{h} \in \mathcal{H}(\boldsymbol{x})} \langle \boldsymbol{w}_h^t, \boldsymbol{\Phi}_h(\boldsymbol{x}, \boldsymbol{h}) \rangle + \ell(\tilde{\boldsymbol{h}}, \boldsymbol{h})$
    $\hat{\boldsymbol{y}} \leftarrow \arg\max_{\boldsymbol{y}' \in \mathcal{Y}(\boldsymbol{x},\hat{\boldsymbol{h}})} \langle \boldsymbol{w}_y^t, \boldsymbol{\Phi}_y(\boldsymbol{x}, \hat{\boldsymbol{h}}, \boldsymbol{y}') \rangle + \ell(\boldsymbol{y}, \boldsymbol{y}')$
    $\boldsymbol{w}_h^{t+1} \leftarrow \boldsymbol{w}_h^t + \boldsymbol{\Phi}_h(\boldsymbol{x}, \tilde{\boldsymbol{h}}) - \boldsymbol{\Phi}_h(\boldsymbol{x}, \hat{\boldsymbol{h}})$
    $\boldsymbol{w}_y^{t+1} \leftarrow \boldsymbol{w}_y^t + \boldsymbol{\Phi}_y(\boldsymbol{x}, \tilde{\boldsymbol{h}}, \boldsymbol{y}) - \boldsymbol{\Phi}_y(\boldsymbol{x}, \hat{\boldsymbol{h}}, \hat{\boldsymbol{y}})$
    $t \leftarrow t + 1$
$\boldsymbol{w}_h \leftarrow \frac{1}{t} \sum_{k=1}^{t} \boldsymbol{w}_h^k$
$\boldsymbol{w}_y \leftarrow \frac{1}{t} \sum_{k=1}^{t} \boldsymbol{w}_y^k$
return $(\boldsymbol{w}_h, \boldsymbol{w}_y)$

Figure 2.3: Latent structure perceptron algorithm.

latent predictor – that makes use of $\boldsymbol{y}$. The constrained predictor finds the maximum scoring latent structure among all structures that follow the correct output $\boldsymbol{y}$. That is, the constrained set $\mathcal{H}(\boldsymbol{x}, \boldsymbol{y}) \subset \mathcal{H}(\boldsymbol{x})$ does not include latent structures that lead to incorrect output structures $\boldsymbol{y}' \neq \boldsymbol{y}$. The constrained latent structure $\tilde{\boldsymbol{h}}$ is then used as the ground truth for the current iteration. Therefore, the model update is determined by the difference between the constrained structure latent and the document tree predicted by the ordinary predictor.

The latent structure perceptron algorithm learns to predict latent structures that help to solve the target task. This algorithm is an instance of the Concave-Convex Procedure and converges to a local optimum (Yuille and Rangarajan, 2003; Yu and Joachims, 2009).

2.4 Empirical Results

Recently, dependency parsing has attracted much attention, and fast progress has been made on pushing the performance of dependency parsers. In 2005, McDonald et al. (2005) proposes the MSTParser system that achieves state-of-the-art performance on different datasets. In the next year, the CoNLL-2006 Shared Task (Buchholz and Marsi, 2006) is devoted to multilingual dependency parsing, and McDonald et al. (2006) achieves the best performances by applying an extension of MSTParser that uses *second-order features*. MSTParser's original features are based on individual edges. The second-order features depend on two edges, which link a head token to two *sibling* modifiers. Since this model considers more complex dependencies

PUC-Rio - Certificação Digital Nº 0812627/CA

in the output structure, the corresponding prediction problem is also more complex. In fact, the prediction problem in this case is NP-Hard (McDonald and Pereira, 2006). McDonald and Pereira (2006) propose an approximation algorithm to this problem and show that the second-order model outperform the first-order one, even using approximated prediction. In Table 2.1, we show the performances of these two systems on the Portuguese CoNLL-2006 dataset. The performances are reported using the *unlabeled attachment score* (UAS).

Reference	**Learning Algorithm**	**Basic Features**	**UAS**
Koo et al. (2010)	MIRA	3rd order	93.03
		2nd order	92.57
McDonald et al. (2005, 2006); McDonald and Pereira (2006)	MIRA	2nd order	91.36
		1st order	90.68
This work	SPerc	1st order	90.06

Table 2.1: Performances of dependency parsers using manual templates on the Portuguese CoNLL-2006 dataset. These systems use different learning algorithms and also different basic features.

PUC-Rio - Certificação Digital Nº 0812627/CA

UAS is the percentage of tokens that are attached to the correct head or, equivalently, the percentage of correct edges in the predicted tree.

Nowadays, as far as we know, the best performing system on the Portuguese CoNLL-2006 dataset is the dual decomposition system proposed by Koo et al. (2010). In Table 2.1, we present the performances of this system. This system introduces a new algorithm to perform approximated prediction with second- and third-order features. However, the second-order features used in this system are slightly different from the ones used in MSTParser, as we can see from the achieved results. The third-order features include grandparent dependencies, in addition to the sibling dependencies given by second-order features. All these models are trained with MIRA and complex features are generated with the manual templates proposed by McDonald and Pereira (2006).

We also present in Table 2.1 the performance achieved by a first-order model trained by the SPerc algorithm with the same templates used in MSTParser. We notice that this system is outperformed by MSTParser, even when MSTParser is using first-order features. This difference is probably due to the training algorithm and also to some differences in the feature templates that are not completely described in the corresponding references.