Available online at www.sciencedirect.com
SCIENCE DIRECT®
Pattern Recognition Letters 26 (2005) 1896–1906
Pattern Recognition Letters
www.elsevier.com/locate/patrec

Wide coverage natural language processing using kernel methods and neural networks for structured data

Sauro Menchetti [a,*], Fabrizio Costa [a], Paolo Frasconi [a], Massimiliano Pontil [b]

[a] Department of Systems and Computer Science, Università di Firenze, Via di S. Marta 3, 50139 Firenze, Italy
[b] Department of Computer Science, University College London, Gower Street, London WC1E 6BT, UK

Available online 29 April 2005

Abstract

Convolution kernels and recursive neural networks are both suitable approaches for supervised learning when the input is a discrete structure like a labeled tree or graph. We compare these techniques in two natural language problems. In both problems, the learning task consists in choosing the best alternative tree in a set of candidates. We report about an empirical evaluation between the two methods on a large corpus of parsed sentences and speculate on the role played by the representation and the loss function.

Keywords: Convolution kernels; Recursive neural networks; Learning preferences; Natural language

1. Introduction

Supervised learning algorithms on discrete structures such as strings, trees or graphs are very often derived from vector based methods, using a function composition approach. In facts, if we exclude symbolic learning algorithms such as inductive logic programming, any method that learns classification, regression or preference functions from variable-size discrete structures must first convert structures into a vector space and subsequently apply "traditional" learning tools to the resulting vectors.

The mapping from discrete structures to vector spaces can be realized in alternative ways. One possible approach is to employ a kernel function: for example, Haussler (1999) introduces convolution kernels on discrete structures, Jaakkola and Haussler (1998) describes the Fisher kernel, Collins and Duffy (2001) proposes a kernel for parse tree and Lodhi et al. (2000) employs string kernels for text classification. Similarly, recursive neural

* Corresponding author. Fax: +39 055 4796 363.
E-mail addresses: menchett@dsi.unifi.it (S. Menchetti), costa@dsi.unifi.it (F. Costa), paolo@dsi.unifi.it (P. Frasconi), m.pontil@cs.ucl.ac.uk (M. Pontil).
URL: http://www.dsi.unifi.it/neural/ (S. Menchetti).

0167-8655/$ - see front matter
doi:10.1016/j.patrec.2005.03.011

networks (RNNs) (Frasconi et al., 1998) can solve the supervised learning problem when the input portion of the data is a labeled directed ordered acyclic graph (DOAG). The two methods have different potential advantages and disadvantages.

Kernel methods for discrete structure are linear in an infinite-dimensional representation space, while RNNs are highly nonlinear but capable of developing an adaptive representation space. Many kernel-based algorithms, unlike neural networks, typically minimize a convex functional, thus avoiding the difficult problem of dealing with local minima. However, this problem is only partially avoided. In fact, the kernel function usually needs to be tuned/adapted to the problem at hand, a problem which cannot in general be cast as a convex optimization problem. A typical example is tuning the variance of a Gaussian kernel. Learning the kernel function is still an open problem, particularly in the case of discrete structures. When using kernels such as the spectrum kernel (Leslie et al., 2002), the string kernel (Lodhi et al., 2000) or the parse tree kernel (Collins and Duffy, 2001), the mapping from discrete structures to the feature space is fixed before learning by the choice of the kernel function and remains unchanged during all the learning procedure. The Fisher kernel (Jaakkola and Haussler, 1998), if applied for example to strings, introduce some degree of adaptation with respect to the distribution of training instances. A non-optimally chosen kernel may lead to very sparse representations and outweigh the benefits of the subsequent large margin methods.

On the other hand, RNNs operate by composing two adaptive functions. First, a discrete structure is *recursively* mapped into a low-dimension vector by an adaptive function Φ. Second, the output is computed by a feedforward neural network that takes as argument the vector representation computed in the first step. Thus the role played by Φ in RNNs is similar to that of the kernel function but it is carried out by means of an adaptive function, leading to vector representations that are focused on the particular learning task.

In this paper, we compare RNNs and convolution kernels in two large scale preference learning problems that occur in computational linguistics: prediction of first pass attachment under strong incrementality hypothesis (Sturt et al., 2003) and reranking parse trees generated by a statistical parser (Collins and Duffy, 2001, 2002). Both problems involve learning a preference function that selects the best alternative in a set of competitors. We show how to perform preference learning in this highly structured domain and we enlighten some interesting connections between these two approaches. We report about several experimental comparisons showing that in this class of problems generalization performance is determined by several factors including the similarity measure induced by the kernel or by the adaptive internal representation of the RNN and, importantly, by the loss function associated with the preference model.

2. Notation

Let $\boldsymbol{x}$ be an element of an instance space $\mathscr{X}$, y an element of a target space $\mathscr{Y}$ associated with $\mathscr{X}$, $\mathscr{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{m}$ a collection of i.i.d. samples from $\mathscr{X} \times \mathscr{Y}$ and m its cardinality. In a ranking problem, let $\mathscr{D} = \{(\boldsymbol{x}_{i1}, y_{i1}), \ldots, (\boldsymbol{x}_{ik_i}, y_{ik_i})\}_{i=1}^{m}$ be a data set, where $(\boldsymbol{x}_{i1}, \ldots, \boldsymbol{x}_{ik_i})$ is the ith sequence of competing instances, $\boldsymbol{x}_{ij} \in \mathscr{X}$, $y_{ij} \in \mathbb{N}$ is the rank of $\boldsymbol{x}_{ij}$: in this setting, $\boldsymbol{x}_{ij}$ precedes $\boldsymbol{x}_{ik}$ (written $\boldsymbol{x}_{ij} \prec \boldsymbol{x}_{ik}$) if $y_{ij} < y_{ik}$. In a preference problem, let $\mathscr{D} = \{(\boldsymbol{x}_{i1}, \ldots, \boldsymbol{x}_{ik_i}), y_i\}_{i=1}^{m}$ be a data set, where $y_i \in \mathbb{N}$ is the index of best element.

Given a labeled ordered graph $\mathscr{G}$, let V be the set of vertexes v of the graph. Let ch[v] be the ordered tuple of vertexes whose elements are children of v, ch$_i$[v] the ith child of v and c_v the number of children of v. Let $\mathscr{I}$ the set of N labels associated with vertexes of the graph and $I(v)$ the label attached to vertex v.

3. Convolution kernels

Collins and Duffy (2001) proposes a convolution kernel for parse trees generated by a natural language parser that can be applied to each generic tree with labels in a finite set (e.g. non-terminal symbols in natural language processing). Given a set of labeled parse trees, let $\phi : \mathscr{X} \mapsto \mathscr{H}$ be a

mapping from the tree space $\mathscr{X}$ to an high dimensional Hilbert space $\mathscr{H}$. In this kernel, the feature space $\mathscr{H} = \{\tau_i\}_{i=1}^{n}$ is the set of all the tree fragments Bod (2001) with the only constraint that a production rule cannot be divided into further subparts. Then each parse tree $\boldsymbol{x}$ is represented by an n-dimensional vector $\phi(\boldsymbol{x}) = \{\phi_i(\boldsymbol{x})\}_{i=1}^{n}$ of tree fragments: the ith feature value $\phi_i(\boldsymbol{x})$ counts the number of occurrences of the ith tree fragment τ_i in x. This representation can be seen as a *bag of subtrees* representation: the object is mapped into a feature vector of which each component counts the number of occurrence of any structure. The inner product between two trees $K(\boldsymbol{x}_1, \boldsymbol{x}_2) = \phi(\boldsymbol{x}_1)^{\mathrm{T}}\phi(\boldsymbol{x}_2)$ can be computed efficiently as follows (Collins and Duffy, 2001), without actually enumerating all the tree fragments:

$$K(\boldsymbol{x}_1, \boldsymbol{x}_2) = \sum_{v_1 \in V_1} \sum_{v_2 \in V_2} T(v_1, v_2) \tag{1}$$

where $T(v_1, v_2)$ counts the number of common tree fragments of $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ that are rooted at boot v_1 and v_2. $T(v_1, v_2)$ is recursive computed as follows: $T(v_1, v_2) = 0$ if the productions at v_1 and v_2 are different, else if v_1 and v_2 are pre-terminals[1] $T(v_1, v_2) = 1$ otherwise

$$T(v_1, v_2) = \prod_{i=1}^{c_{v_1}} (1 + T(\mathrm{ch}_i[v_1], \mathrm{ch}_i[v_2])) \tag{2}$$

A problem of this kernel is its dependency on the number of tree nodes: the large is the number of nodes, the bigger is the kernel value. It can be overcome by normalizing the kernel value. An another drawback is that the kernel value distribution is very peaked. Since a match between two large tree fragments gives a very large contribution to the kernel value, we can reduce peaks by limiting the height of subtrees during kernel computation. Otherwise, we can weight all the fragments by a coefficient which decays exponentially with their size, so larger fragments have a smaller coefficient:

$$K_\lambda(\boldsymbol{x}_1, \boldsymbol{x}_2) = \sum_{i=1}^{n} \lambda^{\mathrm{size}_i} \phi_i(\boldsymbol{x}_1)\phi_i(\boldsymbol{x}_2) \tag{3}$$

[1] Pre-terminals are nodes directly above leaves.

where size_i is the number of productions in the ith tree fragment and $0 < \lambda \leqslant 1$.

3.1. Voted perceptron

Kernels can be used in conjunction with several learning algorithms. SVMs are employed when we are interested to find a maximal-margin solution with a good generalization performance, since it is theoretically well-founded and there are guarantees on its convergence to a unique global optimum. Unfortunately the complexity associated with SVMs training grows at least quadratically in the training set size: in our experiments, there are about 10^8 training examples and so SVMs becomes prohibitive.

VP is an online algorithm for binary classification based on Rosenblatt's perceptron algorithm (Freund and Schapire, 1999). It is much simpler to implement and more efficient in terms of computational time with respect to SVM classifiers, although there are no convergence guarantees. On the other hand, as experimentally shown in (Freund and Schapire, 1999), the performance obtained with VP tends to the performance of maximal-margin classifiers. The training procedure uses a labeled training set $\mathscr{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{m}$, where $\boldsymbol{x}_i \in \mathscr{X}$ and $y_i \in \{+1, -1\}$. At each step k, VP classifies a training example $\boldsymbol{x}_i$ using the current vector of weights $\boldsymbol{w}_k$. If the predicted label $\hat{y}_i$ is different from the true label y_i, then the vector $\boldsymbol{w}_k$ is updated, $\boldsymbol{x}_i$ is added to a list $\mathscr{L}$ of incorrectly classified examples and training goes to next step, else the number c_k of training examples correctly classified by $\boldsymbol{w}_k$ is increased. VP outputs a set of E weighted perceptrons $\mathscr{W} = \{(\boldsymbol{w}_k, c_k)\}_{k=1}^{E}$ or, equivalently, in the dual space, a list $\mathscr{L} = \{(\boldsymbol{x}_{q_k}, c_k)\}_{k=1}^{E}$ of incorrectly classified examples. The basic vector-based formulation can be easy kernelized. The predicted label of a new instance is $\hat{y} = \mathrm{sign}[f(\boldsymbol{x})]$ where

$$\begin{aligned} f(\boldsymbol{x}) &= \sum_{k=1}^{E} c_k \Gamma[\boldsymbol{w}_k^{\mathrm{T}} \phi(\boldsymbol{x})] \\ &= \sum_{k=1}^{E} c_k \Gamma\left[\sum_{j=1}^{k} y_{q_j} K(\boldsymbol{x}_{q_j}, \boldsymbol{x})\right] \end{aligned} \tag{4}$$

and Γ is the identity or the sign function. The worst case complexity of the VP training algorithm is $O(dm^2u)$, where d is the number of epochs and u is the cost to compute the kernel (Freund and Schapire, 1999). In practice, VP convergence is reasonably fast.

4. Recursive neural networks

RNNs are a generalization of neural networks (NNs) capable of processing structured data as DOAGs where a discrete or real label is associated with each vertex (Frasconi et al., 1998). The key idea is to replicate a NN for each node of the DOAG and consider as input to the network both the atomic information represented by the label and the structured information derived by the output of all the networks instantiated for each child node. The process of replicating a NN is called *network unfolding* and, as a result of this procedure, we obtain a large network having shared weights and whose topology matches that of the input graph. At each node v, the NN outputs a vector encoding of the whole subgraph induced by vertexes reachable from v. Data processing takes place in recursive fashion, traversing the DOAG in post-order, using a transition function t such that $\boldsymbol{\varphi}(v) = t(\boldsymbol{\varphi}(\mathrm{ch}[v]), I(v))$ where $\boldsymbol{\varphi}(v) \in \mathbb{R}^n$ denotes the state vector associated with node v and $\boldsymbol{\varphi}(\mathrm{ch}[v]) \in \mathbb{R}^{c_v \cdot n}$ is a vector obtained by concatenating the components of the state vectors contained in the c children of v. The transition function t: $\mathbb{R}^{c_v \cdot n} \times \mathscr{I} \mapsto \mathbb{R}^n$ maps states at v's children and the label at v into the state vector at v. A *frontier* state $\boldsymbol{\varphi}_0 = 0$ is used as the base step of recursion. We use a feedforward neural network to model the transition function t according to the scheme:

$$\begin{aligned} a_j(v) &= \omega_{j0} + \sum_{h=1}^{N} \omega_{jh} z_h(I(v)) \\ &\quad + \sum_{k=1}^{c} \sum_{\ell=1}^{n} w_{jk\ell} \varphi_\ell(\mathrm{ch}_k[v]) \\ \varphi_j(v) &= \tanh(a_j(v)), \quad h = 1, \ldots, n \end{aligned} \tag{5}$$

where $\varphi_j(v)$ denotes the jth component of the state vector at vertex v, $z_h(I(v))$ is a one-hot encoding of label at node v and ω_{jh}, $w_{jk\ell}$ are adjustable weights. Proceeding in this fashion, the state vector

$$\Phi_\omega(\boldsymbol{x}) = \boldsymbol{\varphi}(r) \tag{6}$$

at the root r of the tree encodes the whole data structure and can be used for subsequent processing. The prediction $f(\boldsymbol{x})$ is computed by the output network as $f(\boldsymbol{x}) = \boldsymbol{o}^{\mathrm{T}} \Phi_\omega(\boldsymbol{x})$ where $\boldsymbol{o}$ are the weights of the output network. In the case of regression, a candidate error function to minimize is

$$E(\mathscr{D}) = \sum_{i=1}^{m} (f(\boldsymbol{x}_i) - y_i)^2 \tag{7}$$

Minimizing the error (7) leads to find a value for the parameters of the RNN and to discover a vector state representation of input structures. Minimization is achieved by a variant of the gradient descend backpropagation algorithm (Goller and Kuechler, 1996).

5. Ranking and preference problems

Work on learning theory has mostly concentrated on classification and regression. However, there are many applications in which it is desirable to order rather than to classify instances: these problems arise frequently in social sciences, in information retrieval, in econometric models and in classical statistics where human preferences play a major role.

In a general supervised learning task, the goal is to compute a function f: $\mathscr{X} \mapsto \mathscr{Y}$ which best models the probabilistic relation between these two spaces. The properties of the target set $\mathscr{Y}$ define different learning problems: (a) if $\mathscr{Y}$ is a finite unordered set, we have a classification problem; (b) if $\mathscr{Y}$ is a metric space, e.g., the set of real numbers, we have a regression problem. Ordinal regression, partial ranking and preference model tasks do not fit in any two previous classes but share properties of both classification and regression problems: (a) $\mathscr{Y}$ is a finite ordered set; (b) $\mathscr{Y}$ is not a metric space. So, as in classification problems, we have to assign a possible label to a new instance, but similar to regression problems, the label set admits an order relation. More precisely, in a ranking problem we have to sort a set of

competing alternatives by their importance, while in a preference problem we are only interested in the best element: note that the preference problem is a particular case of ranking.

In the case of ranking, we learn a function $f_{\mathrm{RANK}} : \mathscr{X}^* \mapsto \mathscr{P}^*$ that maps sequences of instances into corresponding ranks. Here $\mathscr{X}^*$ is the set of all sequences of instances in $\mathscr{X}$ and $\mathscr{P}^* = \cup_k \mathscr{P}_k$ being $\mathscr{P}_k$ the set of permutations of the first k integers. By writing $f_{\mathrm{RANK}}(\boldsymbol{x}_{i1}, \ldots, \boldsymbol{x}_{ik_i}) = (\pi_{i1}, \ldots, \pi_{ik_i})$ we have that π_{ij} is the rank assigned to the element $\boldsymbol{x}_{ij}$. A suitable loss function for ranking should penalize predicted permutations that are too different from the correct one. In the particular case of preference learning, we are only interested in ranking high the best element of the sequence, indexed by $\pi_i = \arg\min_{j=1,\ldots,k_i}\{\pi_{ij}\}$. Hence a natural family of preference 0–1 loss functions is

$$L_r(f_{\mathrm{RANK}}(\boldsymbol{x}_{i1}, \ldots, \boldsymbol{x}_{ik_i}), y_i) = \begin{cases} 1 & \text{if } \pi_{y_i} > r \\ 0 & \text{if } \pi_{y_i} \leqslant r \end{cases} \tag{8}$$

where $r = 1, \ldots, k_i$ is the number of top positions the correct element is ranked in, before counting an error. In particular, L_1 measures the number of sequences whose best element is not ranked first. As detailed below, both kernel methods and RNNs rely on the definition of a suitable utility function to realize f_{RANK}.

5.1. The utility function approach

We measure the importance of an instance by introducing an utility function U: $\mathscr{X} \mapsto \mathbb{R}$ so that given $\boldsymbol{x}, \boldsymbol{z} \in \mathscr{X}$ such that $\boldsymbol{x} \prec \boldsymbol{z}$, then $U(\boldsymbol{x}) > U(z)$. To rank a sequence of objects we use the values assigned by the utility function to the instances. In this way, ranking is reduced to learning an utility function U. The rank of $\boldsymbol{x}_{ij}$ is $\pi_{ij} = \sum_{r=1}^{k_i} \theta(U(\boldsymbol{x}_{ir}) - U(\boldsymbol{x}_{ij}))$ where $\theta : \mathbb{R} \mapsto \{1, 0\}$ is the Heavyside function defined as $\theta(x) = 1$ if $x \geqslant 0$, 0 otherwise. In the following we focus on learning preferences. In this case we select only the best element $\pi_i = \arg\max_{j=1,\ldots,k_i U}(\boldsymbol{x}_{ij})$.

5.2. Recursive neural networks preference model

We implement $U(\boldsymbol{x})$ by a neural network having a single linear output. Consider the multinomial variable Y_i representing the index of the correct element in an input sequence. The conditional probability that $Y_i = j$, i.e. that $\boldsymbol{x}_{ij}$ is ranked first by the utility function realized as a neural network with parameters ω, is estimated using the softmax function as follows:

$$P(Y_i = j | \boldsymbol{x}_{i1}, \ldots, \boldsymbol{x}_{ik_i}, \omega) = \frac{\mathrm{e}^{U(\boldsymbol{x}_{ij})}}{\sum_{r=1}^{k_i} \mathrm{e}^{U(\boldsymbol{x}_{ir})}} \tag{9}$$

Under this model, the likelihood function is $\prod_{i=1}^{m} P(Y_i = y_i \mid \boldsymbol{x}_{i1}, \ldots, \boldsymbol{x}_{ik_i}, \omega)$. Learning proceeds by minimizing the negative log-likelihood

$$\begin{aligned} E_{\mathrm{PREF}}(\mathscr{D}) &= -\sum_{i=1}^{m} \log \frac{\mathrm{e}^{U(\boldsymbol{x}_{iy_i})}}{\sum_{\ell=1}^{k_i} \mathrm{e}^{U(\boldsymbol{x}_{i\ell})}} \\ &= \sum_{i=1}^{m} \log \sum_{\ell=1}^{k_i} \mathrm{e}^{U(\boldsymbol{x}_{i\ell}) - U(\boldsymbol{x}_{iy_i})} \end{aligned} \tag{10}$$

with respect to the model parameters ω. A back-propagation gradient descent algorithm for this purpose can be easily defined by injecting as error signals the partial derivatives of $E_{\mathrm{PREF}}(\mathscr{D})$ with respect to $U(\boldsymbol{x}_{i\ell})$.

5.3. Kernel preference model

The approach proposed in (Cohen et al., 1999) and also followed in (Collins and Duffy, 2001, 2002) starts from a simple linear model where the utility function U is parametrized by a vector w such that $U(\boldsymbol{x}) = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}$. In a general ranking problem, $\boldsymbol{w}$ must satisfy the following constraints:

$$\begin{aligned} &\boldsymbol{w}^{\mathrm{T}}(\boldsymbol{x}_{ij} - \boldsymbol{x}_{ik}) > 0, \\ &i = 1, \ldots, m \text{ and } j, k = 1, \ldots, k_i : y_{ij} < y_{ik} \end{aligned} \tag{11}$$

As a special case, in a preference model we have

$$\begin{aligned} &\boldsymbol{w}^{\mathrm{T}}(\boldsymbol{x}_{iy_i} - \boldsymbol{x}_{ij}) > 0, \\ &i = 1, \ldots, m \text{ and } j = 1, \ldots, k_i, j \neq y_i \end{aligned} \tag{12}$$

In this way, ranking and preference problems are reduced to binary classification of pairwise differences between instances.[2] In other words, we learn a function $f_{\mathrm{PAIR}} : \mathscr{X} \times \mathscr{X} \mapsto \{-1, +1\}$ with an associated 0–1 loss function

[2] The number of constraints in (11) grows quadratically in the size of the sequence of alternatives, while it is linear in (12).

$$L'(f_{\mathrm{PAIR}}(\boldsymbol{x}_{ij},\boldsymbol{x}_{i\ell}),z_{ij\ell}) = \frac{1 - f_{\mathrm{PAIR}}(\boldsymbol{x}_{ij},\boldsymbol{x}_{i\ell})z_{ij\ell}}{2} \quad (13)$$

where $z_{ij\ell} = 1$ if $\boldsymbol{x}_{ij} \succ \boldsymbol{x}_{i\ell}$ and $z_{ij\ell} = -1$ otherwise. The overall loss associated with the training data is in this case

$$E_{\mathrm{PAIR}}(\mathscr{D}) = \sum_{i=1}^{m} \sum_{j=1, j\neq y_i}^{k_i} L'(f_{\mathrm{PAIR}}(\boldsymbol{x}_{iy_i},\boldsymbol{x}_{ij}), z_{iy_ij}) \quad (14)$$

as opposite to the overall loss under the preference model

$$E_{\mathrm{PREF}}(\mathscr{D}) = \sum_{i=1}^{m} L_1(f_{\mathrm{RANK}}(\boldsymbol{x}_{i1},\ldots,\boldsymbol{x}_{ik_i}), y_i) \quad (15)$$

Additionally, the pairwise model does not take into consideration all the alternatives together but only two by two and that the best element is used as many times as the number of elements k_i of a sequence of alternatives. The approach can be easily generalized with kernels:

$$\begin{aligned}\boldsymbol{w}^{\mathrm{T}}[\phi(\boldsymbol{x}_{ij}) - \phi(\boldsymbol{x}_{ik})] &= \sum_t \alpha_{s_tp_tq_t}[K(\boldsymbol{x}_{s_tp_t},\boldsymbol{x}_{ij}) - K(\boldsymbol{x}_{s_tp_t},\boldsymbol{x}_{ik}) \\ &\quad - K(\boldsymbol{x}_{s_tq_t},\boldsymbol{x}_{ij}) + K(\boldsymbol{x}_{s_tq_t},\boldsymbol{x}_{ik})]\end{aligned}$$

for appropriate coefficients $\alpha_{s_tp_tq_t}$ and indexes s_t, p_t and q_t. In a preference model or in an ordinal regression task, Γ in (4) is chosen to be the identity function to avoid ties.

6. Applications to natural language

We introduce two problems in natural language that can be modeled as learning preferences over structured data. Both can be formulated as the task of selecting the best element in a set of (partial) parse trees.

6.1. The first pass attachment

Resolution of syntactic ambiguities is a fundamental problem in natural language processing and learning is believed to play a crucial disambiguation role in the human language processing system. For example, consider the sentence

The servant of the actress *who*

was on the balcony died. (16)

where ambiguity resolution accounts to determining which noun the pronoun *who* refers to. Cuetos and Mitchell (1988) found that native English speakers prefer lower attachment (the actress was on the balcony) while native Spanish speakers prefer higher attachment (the servant was on the balcony) when confronted with the Spanish equivalent of the sentence. The subsequent *tuning hypothesis* (Mitchell et al., 1995) states that parsing choices are affected by the exposure to the different statistical regularities of languages. Under a second widely accepted and experimentally validated assumption in psycholinguistics, human processing is *incremental*, i.e. sentences are parsed left-to-right, maintaining at every time a connected syntactic structure that is incrementally augmented when new words arrive. This approach can be formalized by introducing a dynamic grammar where states are incremental trees T_k (that span the first k words in a sentence) and state transitions are obtained by attaching a substructure called *connection path* (CP) to the previous tree to obtain a new incremental tree T_{k+1}. A corpus based set of CPs can be readily obtained from a treebank Lombardo and Sturt, in press. The node on the right frontier of T_k where attachment occurs is called an *anchor* while the POS-tag of the word in the CP is called a *foot*. In this framework, ambiguity resolution reduces to the first-pass attachment problem, illustrated in Fig. 1.

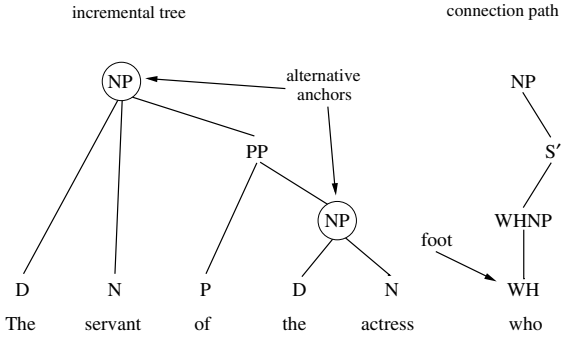

Fig. 1. The two main syntactic interpretations of sentence (16) can be obtained by attaching the same CP to one of the two alternative anchors. In general, several CPs and several attachments for each CP are possible.

In general, several CPs may be attached to a tree T_k. For the attachment to be admissible, it suffices that a matching anchor is found in the right frontier of T_k and that the POS-tag of the new word matches the foot of the CP. The resulting forest of admissible incremental trees that include the next work may contain hundreds of alternative trees when using realistic wide coverage corpora. Disambiguation can be formulated as the problem of learning to predict the correct member of the forest (Sturt et al., 2003).

6.1.1. Tree reduction and specialization

We found that the disambiguation accuracy previously reported in (Sturt et al., 2003) can be significantly improved by means of two linguistically motivated heuristics (Costa et al., submitted for publication). The first is called *tree reduction* and consists in removing nodes from the syntactic parse that are considered irrelevant for discriminating between alternative incremental trees. Intuitively these nodes are deep nodes, where the depth is measured with respect to the part of the left context where the attachment process takes place.

The second heuristic consists of *specializing* the first pass attachment prediction with respect to the class of the item being attached. The idea is to train and employ specialized predictors for each different word classes (nouns, verbs, etc.). The heuristic is applicable since the different classes naturally partition the data set into non-overlapping sets.

6.2. The reranking task

The second task, originally formulated in (Collins and Duffy, 2002), consists of reranking alternative parse trees generated for the same sentence by a statistical parser. In this case each forest consists of full candidate trees for the entire sentence. In addition, each parse tree has a score that measures the probability of this tree given the sentence and an underlying stochastic grammar. Note that the forest does not necessarily contain the correct parse tree for the sentence (the gold tree). Alternative parses are ranked according to the standard parseval measures: labeled recall (LR), labeled precision (LP) and crossing brackets (CB). A constituent is a triple consisting of an internal node, its label (a non-terminal symbol) and the indexes of the first and the last word it spans. A constituent is correct if it spans the same set of words and has the same label as a constituents in the gold tree. LP is the number of correct predicted constituents divided by the number of constituents in the gold tree; LR the is the number of correct predicted constituents divided by the number of constituents in the parse tree; CB is number of constituents which violate constituent boundaries with a constituent in the gold tree; sentences with no crossing brackets (0 CBs) is the percentage of sentences which have zero crossing brackets and sentences with two or less crossing brackets (2 CBs) is the percentage of sentences which have less equal then two crossing brackets.

The best tree in a forest is defined as the one having maximum harmonic average of LP and LR (F_1 metric). The reranking task consists of predicting such best tree given the forest and is naturally formulated as a preference problem.

6.3. Experimental results

In our experiments, we use the Wall Street Journal (WSJ) section of Penn TreeBank (Marcus et al., 1993). It is a large size realistic corpus of natural language that contains about 40,000 parsed sentences for a total of 1 million words that has been widely used in the computational linguistic community. We followed the standard split of the data set using sections 2–21 for training, section 23 for test and section 24 for validation.

6.3.1. First pass attachment

In the first pass attachment disambiguation task, all parse trees have been preprocessed with tree reduction and specialization (see Section 6.1.1). In particular, ten specialized data sets have been obtained by splitting data according to the syntactic category of the foot node: Nouns, Verbs, Prepositions, Articles, Punctuations, Adjectives, Adverbs, Conjunctions, Possessives, Others.

In order to estimate learning curves we created data sets with 100, 500, 2000, 10,000 and 40,000 sentences, randomly extracting them from the training set. In the full data set of 40,000 sentences

the average sentence length is 24 and for each word there are 120 alternative incremental trees on average (ranging from a minimum of 2 to a maximum of 2000 alternatives), yielding a total of about 10^8 trees.

Due to the high computational cost of the validation procedure, the model parameters were optimized using a subset of section 24 of WSJ as a validation set. Specifically, we used 500 validation sentences to estimate the kernel parameter λ of Eq. (3) and we found an optimal value $\lambda = 0.5$. The size of the RNN's state vector was fixed to 25 units. Weights were initialized in the range $[-0.01, +0.01]$ and updated after the presentation of each forest. The maximum node outdegree was set to 15. Fixing the maximum outdegree is an architectural constraint which, in our implementation, has the consequence of pruning syntactic trees with very long productions. Note that since each child position is associated with its own set of weights, pruning long productions avoids poor estimates of the weights associated with very infrequent rules. Using 15 children, only 0.3% productions are pruned.

Because of the large size of the training set, we observed that one epoch of VP training is sufficient to reach a steady state validation set accuracy. For the RNN, we used early stopping based on 1000 validation sentences from section 24 of WSJ. We found that on the order of 10^5 sentence presentations are typically needed to complete the training procedure.

The results of the comparison are shown in Table 1, where the error measure is based on the L_1 loss of Eq. (8) which counts the number of forests where the best element is not ranked first. In most of classes of POS tags and in most of training set sizes, the RNN outperforms the VP. If the specialized classifiers are combined with their weights to obtain an overall measure not grouped for classes of POS tag, we see the RNN exhibits about 1% better prediction accuracy on average with respect to VP and no evidence is found for the superiority kernel VP when trained on a small data set.

In order to better assess the behavior on small data sets, we carried out a more robust experiment training the RNN and the VP on 5 independent subsets of 100 sentences each, randomly chosen from WSJ sections 2–21. In this case we used the tree reduction heuristic but not the specialization heuristic. Model parameters were kept identical to the previous experiment. Results are reported in Table 2 and confirm the hypothesis that the RNN outperforms kernel VP even in regime of scarce data available for training.

In addition, kernel VP has the drawback of high computational costs: training over 5 subsets of 100 sentences takes about a week on a 2 GHz CPU and moreover VP does not scale linearly with the number of examples as the RNN does. For small

Table 1
VP and RNN learning curves in the first pass attachment prediction task using modularization in 10 POS tag categories

POS	Noun	Verb	Prep.	Article	Punc.	Adjective	Adverb	Conj.	Poss.	Other	Total
Size %	33.0	13.4	12.6	12.5	11.7	7.5	4.3	2.3	2.0	0.7	100
VP after 1 Epoch											
100	12.4	12.6	47.6	26.5	50.9	24.2	65.8	46.3	7.5	64.9	27.3
500	8.4	8.9	42.3	17.4	39.0	18.2	58.8	38.7	6.5	54.7	21.3
2000	7.1	6.8	38.1	14.2	33.3	14.7	55.7	31.9	5.4	34.6	18.3
10,000	5.5	5.1	34.5	11.1	25.7	12.1	51.0	28.1	4.4	25.8	15.3
40,000	4.4	3.9	31.6	9.6	22.5	11.0	46.4	25.0	2.9	21.8	13.4
RNN											
100	11.4	17.9	43.6	30.4	36.6	23.0	65.2	39.2	40.0	89.0	26.6
500	8.2	9.1	38.2	14.8	31.8	16.4	54.9	40.4	10.2	48.3	19.4
2000	8.5	6.0	37.7	12.3	25.7	16.8	48.1	31.5	5.9	34.5	17.3
10,000	5.9	5.1	35.8	10.6	21.0	13.0	43.9	23.2	2.5	22.3	14.5
40,000	4.3	3.2	32.5	9.0	19.2	10.5	40.6	21.3	2.9	31.4	12.6

data sets, the CPU time of kernel VP is about the CPU time of RNN, while for larger data sets the elaboration time is much higher: in the first experiments, VP took over 2 months to complete an epoch but RNN learns in 1–2 epoch (about 3 days with a 2 GHz CPU). This high computational cost has forced us to train the VP for only one epoch in the full experiment with all the 40,000 sentences. An advantage of the kernel VP is its smoothness with respect to training iterations, i.e. validating the performance on a working set yields a smooth, single-maximum function. In contrast the RNN is much more sensitive, making it hard to decide for a good generalization point.

6.3.2. Reranking task

The data set used in this experiment is the same described in (Collins and Duffy, 2002). There are 30 alternatives on average for each sentence and so the task is computationally less intensive. We do not use the two heuristics, because they are not applicable in this case. The task is very difficult, because the statical parser employed is a very good parser and all the trees output by the parser have an high similarity score with the gold tree. To obtain a better performance of kernel VP and RNN with respect to the statistical parser, we incorporate the probability from the parser in our models. In the case of VP, the new tree kernel is composed by two parts

$$K_{\lambda,\beta}(\boldsymbol{x}_1,\boldsymbol{x}_2) = K_{\lambda}(\boldsymbol{x}_1,\boldsymbol{x}_2) + \beta \log p(\boldsymbol{x}_1)\log p(\boldsymbol{x}_2) \tag{17}$$

where $K_{\lambda}(\boldsymbol{x}_1,\boldsymbol{x}_2)$ is the kernel defined in Eq. (3), $p(\boldsymbol{x})$ is the probability of the parser and $\beta \geqslant 0$ controls the relative contribution of the two terms. The parameters λ and β are set to 0.3 and 0.2 respectively through tuning on the validation set.

For the RNN, a rescaling of the parser probability $a \log p(\boldsymbol{x}) + b$ is used as an additional input to the output network for appropriate values a and b. Some parameters of RNN are: state vector size has been fixed to 25 units, a learning rate $\eta = 0.0001$ and a momentum $\alpha = 0.5$, weight initialization with random weights in $[-0.01, +0.01]$ and maximum node outdegree set to 15. The number of iterations needed to reach an error minimum on validation set is 1820.

The standard parseval measures outlined above are used to assess the performance of predictors. Results for a sentence length less equal then forty (simple sentences) and for all the sentences are reported in Table 3. In this task, the performance of two methods is mostly the same.

6.3.3. The role of representation

To compare the vector representations of trees $\phi(\boldsymbol{x})$ induced by the tree kernel and $\Phi(\boldsymbol{x})$ adaptively computed by the RNN (see Eq. (6)), we trained a VP using a linear kernel on the set of

Table 2
VP and RNN in the first pass attachment prediction task: 5 independent subsets of 100 sentences

Subset	1	2	3	4	5	Average
VP	26.4	26.3	26.5	26.8	27.4	26.7 ± 0.4
RNN	26.7	24.6	26.3	27.0	25.6	26.0 ± 1.0

Table 3
VP and RNN in the reranking task

Model	LR	LP	CBs	0 CBs	2 CBs
⩽40 Words (2245 sentences)					
VP	89.1	89.4	0.85	69.3	88.2
RNN	89.2	89.5	0.84	67.9	88.4
⩽100 Words (2416 sentences)					
VP	88.6	88.9	0.99	66.5	86.3
RNN	88.6	88.9	0.98	64.8	86.3

Table 4
VP on RNN state in the first pass attachment prediction task: 5 independent subsets of 100 sentences

Subset	VP on RNN state	VP
1	27.8	26.4
2	25.6	26.3
3	27.2	26.5
4	28.4	26.8
5	26.4	27.4
Average	27.1 ± 1.1	26.7 ± 0.4

vectors $\Phi(x_i)$ obtained from the trained RNN. In this way, the tree kernel representation is replaced by the RNN adaptive representation and so the two methods can be compared with the same representation. Results are reported in Table 4 and show once again the superiority of RNNs with respect to VP. Since the two algorithms are compared with an equal representation, the different performance is due to the preference loss function, because the VP algorithm and the RNN output network have almost the same behavior. We argue that the problem is the pairwise loss function that does not take into consideration all the alternatives together.

To better understand the adaptive representation generated by RNN, we have applied the PCA to the state vector representations of trees for the incremental task. Projecting the state vector on a two dimensional space, we observe that the elements of a set of alternatives tend to stay in a mainfold of $\mathbb{R}^2$ and that the best element is the most right element of the set (if the RNN has correctly classified it). In Fig. 2, we report a set of alternatives with its best element plotted as a cross.

The experiments reported in the next section further investigate the role played by the loss function.

6.3.4. Comparing alternative preference loss functions

To investigate the role of the loss function, we have compared the setwise L_r in Eq. (8) and the pairwise L' in Eq. (13) loss functions using as evaluation measure the errors in Eq. (14) and in Eq. (15) in the incremental task. Results are reported in Table 5 for the 5 subsets of 100 sentences. The

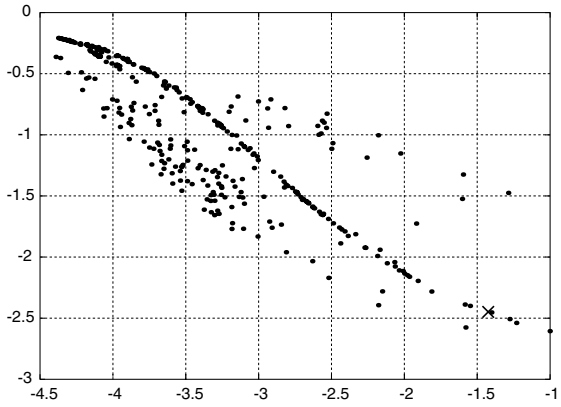

Fig. 2. PCA on RNN state vectors of a forest.

Table 5
Comparison between different loss functions and evaluation measures

Train	Test	RNN	VP
P	P	1.79 ± 0.12	2.32 ± 0.09
P	S	29.1 ± 1.8	26.7 ± 0.4
S	P	1.50 ± 0.20	NA
S	S	26.0 ± 1.0	NA

first two columns show if the training and the testing refer to a pairwise loss function (P) or a setwise loss function (S) and the others report the results of the two algorithms. We see that for the RNN the pairwise loss function shows worst performance with respect to the setwise loss function on both the evaluation measures. No setwise loss function for the VP has been proposed so far. When using the pairwise loss function, the VP performance measured on forests is better than RNN. We argue that a setwise loss function for kernel methods could improve the performance.

7. Conclusions

The experimental analysis presented above shows that both RNNs and the kernel VP are effective to solve the investigated problems. In particular, the adaptive representation developed by the RNN allows a simple linear utility function to solve the preference problem.

The experiments indicate that the choice of a pairwise vs. global loss function plays an import role. In particular, it appears that the pairwise loss is not well suited to train an RNN and it remains to be investigated if this is also the case for kernel methods. Interesting, previous works with kernels (Herbrich et al., 2000, 2001, 2002) focus exclusively on pairwise loss functions. The development of global loss function for preference tasks may lead to more effective solutions.

Acknowledgments

Thanks to Michael Collins for providing his data set for the reranking task (see (Collins and Duffy, 2002) for more details). Thanks to Alessio Ceroni, Alessandro Vullo, Andrea Passerini and Giovanni Soda for their fruitful comments.

References

Bod, R., 2001. What is the minimal set of fragments that achieves maximal parse accuracy? In: Proceedings of ACL.

Cohen, W.W., Schapire, R.E., Singer, Y., 1999. Learning to order things. J. Artif. Intell. Res. 10, 243–270.

Collins, M., Duffy, N., 2001. Convolution Kernels for natural language. In: Proc. Neural Information Processing Systems, NIPS 14.

Collins, M., Duffy, N., 2002. New ranking algorithms for parsing and tagging: Kernels over discrete structures, and the voted perceptron. In: Proc. of ACL.

Costa, F., Frasconi, P., Lombardo, V., Sturt, P., Soda, G., submitted for publication. Ambiguity resolution analysis in incremental parsing of natural language. IEEE Transactions on Neural Network.

Cuetos, F., Mitchell, D., 1988. Cross-linguistic differences in parsing: Restrictions on the use of the late closure strategy in Spanish. Cognition 30 (1), 73–105.

Frasconi, P., Gori, M., Sperduti, A., 1998. A general framework for adaptative processing of data structure. IEEE Trans. Neural Networks 9 (5), 768–786.

Freund, Y., Schapire, R.E., 1999. Large margin classification using the perceptron algorithm. Mach. Learn. 37 (3), 277–296.

Goller, C., Kuechler, A., 1996. Learning task-dependent distributed structure-representations by back-propagation through structure. In: IEEE International Conference on Neural networks, pp. 347–352.

Haussler, D., 1999. Convolution Kernels on Discrete Structures, Tech. Rep. UCSC-CLR-99-10, University of California at Santa Cruz.

Herbrich, R., Graepel, T., Obermayer, K., 2000. Large margin rank boundaries for ordinal regression. In: Smola, A., Bartlett, P., Schölkopf, B., Schuurmans, D. (Eds.), Advances in Large Margin Classifiers. MIT Press.

Jaakkola, T., Haussler, D., 1998. Exploiting generative models in discriminative classifiers. In: Proc. Neural Information Processing Systems, NIPS 10.

Joachims, T., 2002. Evaluating retrieval performance using clickthrough data. In: Proceedings of the SIGIR Workshop on Mathematical/Formal Methods in Information Retrieval.

Leslie, C., Eskin, E., Noble, W.S., 2002. The spectrum Kernel: A string Kernel for SVM protein classification. Pacific Sympos. Biocomput., 566–575.

Lodhi, H., Shawe-Taylor, J., Cristianini, N., Watkins, C., 2000. Text classification using string Kernels. In: Proc. Neural Information Processing Systems, NIPS 13, pp. 563–569.

Lombardo, V., Sturt, P., in press. Incrementality and lexicalism: A treebank study. In: Stevenson, S., Merlo, P. (Eds.), Lexical Representations in Sentence Processing. Computational Psycholingusitics Series, John Benjamins.

Marcus, M., Santorini, B., Marcinkiewicz, M., 1993. Building a large annotated corpus of English: The Penn Treebank. Computat. Linguist. 19, 313–330.

Mitchell, D., Cuetos, F., Corley, M., Brysbaert, M., 1995. Exposure-based models of human parsing: Evidence for the use of coarse-grained (nonlexical) statistical records. J. Psycholinguist. Res. 24 (6), 469–488.

Sturt, P., Costa, F., Lombardo, V., Frasconi, P., 2003. Learning first-pass structural attachment preferences with dynamic grammars and recursive neural networks. Cognition 88 (2), 133–169.