Dynamic-oracle Transition-based Parsing with Calibrated Probabilistic Output

Yoav Goldberg
Computer Science Department
Bar Ilan University
Ramat Gan, Israel
yoav.goldberg@gmail.com

Abstract

We adapt the dynamic-oracle training method of Goldberg and Nivre (2012; 2013) to train classifiers that produce probabilistic output. Evaluation of an Arc-Eager parser on 6 languages shows that the AdaGrad-RDA based training procedure results in models that provide the same high level of accuracy as the averaged-perceptron trained models, while being sparser and providing well-calibrated probabilistic output.

1 Introduction

For dependency parsing, it is well established that greedy transition-based parsers (Nivre, 2008) are very fast (both empirically and theoretically) while still providing relatively high parsing accuracies (Nivre et al., 2007; Kübler et al., 2009).

Recently, it has been shown that by moving from static to dynamic oracles during training, together with a training method based on the averaged-perceptron, greedy parsers can become even more accurate. The accuracy gain comes without any speed penalty at parsing time, as the inference procedure remains greedy (Goldberg and Nivre, 2012).

In transition-based parsing, the parsing task is viewed as performing a series of actions, which result in an incremental construction of a parse-tree. At each step of the parsing process, a classification model is used to assign a score to each of the possible actions, and the highest-scoring action is chosen and applied. When using perceptron based training, the action scores are in the range $(-\infty, \infty)$, and the only guarantee is that the highest-scoring action should be considered "best". Nothing can be inferred from the scale of the highest-scoring action, as well as from the scores assigned to the other actions.

In contrast, we may be interested in a classification model which outputs a proper probability distribution over the possible actions at each step of the process. Such output will allow us to identify uncertain actions, as well as to reason about the various alternatives. Probabilistic output can also be used in situations such as best-first parsing, in which a probabilistic score can be used to satisfy the required "superiority" property of the scoring function (Sagae and Lavie, 2006; Zhao et al., 2013).

Classifiers that output probabilities are well established, and are known as maximum-entropy or multinomial logistic regression models. However, their applications in the context of the dynamic-oracle training is not immediate. The two main obstacles are (a) the dynamic oracle may provide more than one correct label at each state while the standard models expect a single correct label, and (b) the exploration procedure used by Goldberg and Nivre (2012; 2013) assumes an online-learning setup, does not take into account the probabilistic nature of the classifier scores, and does not work well in practice.

This work is concerned with enabling training of classifiers with probabilistic outputs in the dynamic-oracle framework. Concretely, we propose a loss function capable of handling multiple correct labels, show how it can be optimized in the AdaGrad framework (Duchi et al., 2010), and adapt the exploration procedure used in dynamic-oracle training to the probabilistic setup. We use a variant of AdaGrad that performs RDA-based (Xiao, 2010) L_1 regularization, achieving sparse model at inference time.

We implemented our method and applied it to training an Arc-Eager dependency parser on treebanks in

6 languages. On all languages we achieve 1-best parsing results which are on-par with the averaged-perceptron trained models, while also providing well calibrated probability estimates at each step. The probabilistic models have 3-4 times fewer parameters than the perceptron-trained ones. Our code is available for download at the author's web page.

2 Background

2.1 Transition Based Parsing

We begin with a quick review of transition-based dependency parsing (Nivre, 2008), establishing notation. Transition-based parsing assumes a *transition system*, an abstract machine that processes sentences and produces parse trees. The transition system has a set of *states* (also called *configurations*) and a set of *actions* (also called *transitions*) which are applied to states, producing new states. In what follows we denote a state as $x \in \mathcal{X}$, an action as $y \in \mathcal{Y}$, and an application of an action to a state as $y(x)$. When parsing, the system is initialized to an *initial state* based on the input sentence S, to which actions are applied repeatedly. After a finite (in our case linear) number of action applications, the system arrives at a *terminal state*, and a parse tree is read off the terminal configuration. In a greedy parser, a classifier is used to choose the action to take in each state, based on features extracted from the state itself. Transition systems differ by the way they define states, and by the particular set of transitions available. One such system is the Arc-Eager system (Nivre, 2003), which has 4 action types, SHIFT, REDUCE, LEFT$_{lb}$, RIGHT$_{lb}$, where the last two are parameterized by a dependency label lb, resulting in 2+2L actions for a treebank with L distinct arc labels. The system parses a sentence with n words in $2n$ actions. The reader is referred to (Nivre, 2003; Nivre, 2008; Goldberg and Nivre, 2013) for further details on this system.

2.2 Greedy parsing algorithm

Assuming we have a function score$(x, y; w)$ parameterized by a vector w and assigning scores to pairs of states x and actions y, greedy transition-based parsing is simple and efficient using Algorithm 1. Starting with the initial state for a given sentence, we repeatedly choose the highest-scoring action according to our model parameters w and apply it, until we reach a terminal state, at which point we stop and return the parse tree accumulated in the configuration.

Algorithm 1 Greedy transition-based parsing

```
1: Input: sentence $S$, parameter-vector $w$
2: $x \leftarrow$ INITIAL($S$)
3: while not TERMINAL($x$) do
4:     $y \leftarrow \arg\max_{y \in \text{LEGAL}(x)} \text{score}(x, y; w)$
5:     $x \leftarrow y(x)$
6: return tree($x$)
```

In practice, the scoring function takes a linear (or log-linear) form:

$$\text{score}(x, y; w) \propto w \cdot \phi(x, y)$$

where ϕ is a feature extractor returning a high-dimensional sparse vector, and $\cdot$ is the dot-product operation. The role of training a model is to a set good weights to the parameter vector w, based on a training corpus of $\langle x, y \rangle$ pairs. The corpus is provided in the form of $\langle \text{sentence}, \text{tree} \rangle$ pairs, from which states and actions are extracted.

2.3 Static vs. Dynamic Oracles

Until recently, the training corpus of $\langle x, y \rangle$ pairs was extracted by use of a static-oracle – a function mapping a $\langle \text{sentence}, \text{tree} \rangle$ pair to a sequence of $\langle x, y \rangle$ pairs.

Recently, Goldberg and Nivre (2012; 2013) proposed the notion of a *dynamic* parsing oracle. Dynamic parsing oracles are functions oracle$(x; T)$ from a state x to set of actions Y (given a reference tree T). Note that unlike the static oracles which provide only $\langle x, y \rangle$ pairs that are part of a single action sequence leading to a gold tree (and associate a single action y with each state x on this path), the dynamic oracles are defined for every state x (even states that cannot lead to the gold tree), and may associate more than a single action y with each state. The semantics of the dynamic oracle is that the set Y associated with state x contains all and only actions that can lead to an optimal tree (in terms of hamming distance from the reference tree T) which is reachable from state x.

2.4 A Dynamic Oracle for the Arc-Eager System

Goldberg and Nivre (2013) provide a concrete dynamic oracle for the Arc-Eager system, which we use in this work and repeat here for completeness.

We use a notation in which dependency arcs are of the form (h, m) where h is a head and m is a modifier, and a tree T is represented as a set of dependency arcs. Each state x is of the form $x = (\sigma|s, b|\beta, A)$[1] where $\sigma|s$ is a stack with body σ and top s, $b|\beta$ is a buffer (queue) with body β and top b, and A is a set of dependency arcs.

The dynamic oracle for the Arc-Eager system works by calculating the cost of each action in a given state, and returning the set of actions with a cost of zero (the set is guaranteed to be non-empty):

$$\text{oracle}(x, T) = \{a \mid \text{cost}(a; x, T) = 0\}$$

The cost of an action at a state is the number of gold arcs which are mutually-reachable from the state, but will not be reachable after taking the action. The cost function $\text{cost}(\text{ACTION}; x, T)$ of taking an action at state x with respect to a gold set T of dependency arcs is calculated as follows (for further details, see (Goldberg and Nivre, 2013)):

- $\text{cost}(\text{LEFT}; x, T)$: Adding the arc (b, s) and popping s from the stack means that s will not be able to acquire any head or dependents in β. The cost is therefore the number of arcs in T of the form (k, s) or (s, k) such that $k \in \beta$. Note that the cost is 0 for the trivial case where $(b, s) \in T$, but also for the case where b is not the gold head of s but the real head is not in β (due to an erroneous previous transition) and there are no gold dependents of s in β.

- $\text{cost}(\text{RIGHT}; x, T)$: Adding the arc (s, b) and pushing b onto the stack means that b will not be able to acquire any head in σ or β, nor any dependents in σ. The cost is therefore the number of arcs in T of the form (k, b), such that $k \in \sigma \cup \beta$, or of the form (b, k) such that $k \in \sigma$ and there is no arc (u, k) in A_c. Note again that the cost is 0 for the trivial case where $(s, b) \in T$, but also for the case where s is not the gold head of b but the real head is not in σ or β (due to an erroneous previous transition) and there are no gold dependents of b in σ.

- $\text{cost}(\text{REDUCE}; x, T)$: Popping s from the stack means that s will not be able to acquire any dependents in $B = b|\beta$. The cost is therefore the number of arcs in T of the form (s, k) such that $k \in B$. While it may seem that a gold arc of the form (k, s) should be accounted for as well, note that a gold arc of that form, if it exists, is already accounted for by a previous (erroneous) RIGHT transition when s acquired its head.

- $\text{cost}(\text{SHIFT}; x, T)$: Pushing b onto the stack means that b will not be able to acquire any head or dependents in $S = s|\sigma$. The cost is therefore the number of arcs in T of the form (k, b) or (b, k) such that $k \in S$ and (for the second case) there is no arc (u, k) in A_c.

[1]This is a slight abuse of notation, since for the SHIFT action s may not exist, and for the REDUCE action b may not exist.

2.5 Training with Exploration

An important assumption underlying the training of greedy transition-based parsing models is that an action taken in a given state is independent of previous or future actions given the feature representation of the state. This assumption allows treating the training data as a bag of independent $\langle \text{state}, \text{action} \rangle$ pairs, ignoring the fact that the states and actions are part of a sequence leading to a tree, and not considering the interactions between different actions. If the data (both at train and test time) was separable and we could achieve perfect classification at parsing time, this assumption would hold. In reality, however, perfect classification is not possible, and different actions do influence each other. In particular, once a mistake is made, the parser may either reach a state it has not seen in training, or reach a state it has seen before, but needs to react differently to (previous erroneous decisions caused the state to be associated with different optimal actions). The effect of this is *error-propagation*: once a parser erred, it is more likely to err again, as it reaches states it was not trained on, and don't know how to react to them.

As demonstrated by (Goldberg and Nivre, 2012), error propagation can be mitigated to some extent by training the parser on states resulting from common parser errors. This is referred to as “training with exploration” and is enabled by the dynamic oracle. In (Goldberg and Nivre, 2012), training with exploration is performed by sometimes following incorrect classifier predictions during training.

Training with exploration still assumes that the $\langle x, y \rangle$ pairs are independent from each other given the feature representation, but instead of working with a fixed corpus D of $\langle x, y\rangle$ pairs, the set D is generated dynamically based on states x the parser is likely to reach, and the optimal actions $Y = oracle(x; T)$ proposed for these states by the dynamic oracle.

In practice, training with exploration using the dynamic oracle yields substantial improvements in parsing accuracies across treebanks.

3 Training of Sparse Probabilistic Classifiers

As discussed in the introduction, our aim is to replace the averaged-perceptron learner and adapt the training with exploration method of (Goldberg and Nivre, 2012) to produce classifiers that provide probabilistic output.

3.1 Probabilistic Objective Function and Loss

Our first step is to replace the perceptron hinge-loss objective with an objective based on log-likelihood. As discussed in section 2.5 the training corpus is viewed as a bag of states and their associated actions, and our objective would be to maximize the (log) likelihood of the training data under a probability model.

Static-oracle objective In static-oracle training each state x is associated with a single action y.

Denoting the label by $y \in \mathcal{Y}$ and the states by $x \in \mathcal{X}$, we would like to find a parameter vector w to maximize the data log-likelihood L of our training data D under parameter values w:

$$L(D; w) = \sum_{\langle x,y\rangle \in D} \log P(y|x; w)$$

where $P(y|x; w)$ takes the familiar log-linear form:

$$P(y|x; w) = \frac{\exp w \cdot \phi(x, y)}{\sum_{y' \in \mathcal{Y}} \exp w \cdot \phi(x, y')}$$

in which ϕ is a feature extraction function and $\cdot$ is the dot-product operation. This is the well known maximum-entropy classification formulation, also known as multinomial logistic regression.

Dynamic-oracle objective When moving to the dynamic oracle setting, each state x is now associated with a *set* of correct actions $Y \subseteq \mathcal{Y}$, and we would like at least one of these actions $y \in Y$ to get a high probability under the model. To accommodate this, we change the numerator to sum over the elements $y \in Y$, resulting in the following model form (the same approach was taken by Riezler et al. (2002) for dealing with latent LFG derivations in LFG parser training, and by Charniak and Johnson (2005) in the context of discriminative reranking):

$$L(D; w) = \sum_{\langle x,Y\rangle \in D} \log P(Y|x; w)$$

$$P(Y|x; w) = \frac{\sum_{y \in Y} \exp w \cdot \phi(x, y)}{\sum_{y' \in \mathcal{Y}} \exp w \cdot \phi(x, y')}$$

Note the change from y to Y, and the difference between the Y in the numerator (denoting the set of correct outcomes) and $\mathcal{Y}$ in the denominator (denoting the set of all possible outcomes). This subsumes the definition of $P(y|x; w)$ given above as a special case by setting $Y = \{y\}$. We note that the sum ensures that at least one $y \in Y$ receives a high probability score, but also allows other elements of Y to receive low scores.

The (convex) loss for a given $\langle x, Y\rangle$ pair under this model is then:

$$\text{loss}(Y, x; w) = \log \sum_{y \in Y} e^{w \cdot \phi(x,y)} - \log \sum_{y \in \mathcal{Y}} e^{w \cdot \phi(x,y)}$$

and the gradient of this loss with respect to w is:

$$\frac{\partial \text{loss}}{\partial w_i} = \sum_{y \in Y} \frac{e^{w \cdot \phi(x,y)}}{Z_Y} \phi_i(x, y) - \sum_{y \in \mathcal{Y}} \frac{e^{w \cdot \phi(x,y)}}{Z_{\mathcal{Y}}} \phi_i(x, y)$$

where:

$$Z_Y = \sum_{y' \in Y} e^{w \cdot \phi(x,y')} \qquad Z_{\mathcal{Y}} = \sum_{y' \in \mathcal{Y}} e^{w \cdot \phi(x,y')}$$

3.2 L$_1$ Regularized Training with AdaGrad-RDA

The generation of the training set used in the training-with-exploration procedure calls for an online optimization algorithm. Given the objective function and its gradient, we could have used a stochastic gradient based method to optimize the objective. However, recent work in NLP (Green et al., 2013; Choi and McCallum, 2013) demonstrated that the adaptive-gradient (AdaGrad) optimization framework of Duchi et al. (2010) converges quicker and produces superior results in settings which have a large number of training instances, each with a very high-dimensional but sparse feature representation, as common in NLP and in dependency-parsing in particular.

Moreover, a variant of the AdaGrad algorithm called AdaGrad-RDA incorporates an L_1 regularization, and produces sparse, regularized models. Regularization is important in our setting for two reasons: first, we would prefer our model to not overfit accidental features of the training data. Second, smaller models require less memory to store, and are faster to parse with as more of the parameters can fit in the CPU cache.[2]

For these reasons, we chose to fit our model's parameters using the regularized-dual-averaging (RDA) variant of the AdaGrad algorithm. The AdaGrad framework works by maintaining a per-feature learning rate which is based on the cumulative gradient values for this feature. In the RDA variant, the regularization depends on the average vector of all the gradients seen so far.

Formally, the weight after the $J+1$th AdaGrad-RDA update with a regularization parameter λ is:

$$w_i^{J+1} \leftarrow \alpha \frac{1}{\sqrt{G_i} + \rho} \text{shrink}(g_i, J\lambda)$$

where w_i^{J+1} is the value of the ith coordinate of w at time $J+1$, α is the learning rate, and ρ is a ridge parameter used for numerical stability (we fix $\rho = 0.01$ in all our experiments). G is the sum of the squared gradients seen so far, and g is the sum of the gradients seen so far.

$$G_i = \sum_{j=0}^{J} (\partial_i^j)^2 \qquad g_i = \sum_{j=0}^{J} \partial_i^j$$

shrink$(g_i, J\lambda)$ is the regularizer, defined as:

$$\text{shrink}(g_i, J\lambda) = \begin{cases} g_i - J\lambda & \text{if } g_i > 0, |g_i - J\lambda| > 0 \\ g_i + J\lambda & \text{if } g_i < 0, |g_i - J\lambda| > 0 \\ 0 & \text{otherwise} \end{cases}$$

[2]Note that in contrast to the perceptron loss that considers only the highest-scoring and the correct class for each instance, the multilabel log-likelihood loss considers all of the classes. When the number of classes is large, such as in the case of labeled parsing, this will result in very many non-zero scores, unless strong regularization is employed.

For efficiency reasons, the implementation of the AdaGrad-RDA learning algorithm keeps track of the two vectors G and g, and calculates the needed coordinates of w based on them as needed. When training concludes, the final w is calculated and returned. We note that while the resulting w is sparse, the G and g vectors are quite dense, requiring a lot of memory at training time.[3]

For completeness, the pseudo-code for an AdaGrad-RDA update with our likelihood objective is given in algorithm 2.

Algorithm 2 Adagrad-RDA with multilabel logistic loss update.

Globals The global variables G, g and j are initialized to 0. The vectors g and G track the sum and the squared sum of the gradients. The scalar j tracks the number of updates.
Parameters α: learning rate, ρ: ridge, λ: L_1 penalty.
Arguments w: current weight vector, ϕ feature extraction function, x: state, Y: set of good labels (actions) for x.
Others Z_Y, $Z_{\mathcal{Y}}$ and shrink$(\cdot,\cdot)$ are as defined above.
Returns: An updated weight vector w.

```
1: function AdagradUpdate($w, \phi, x, Y$)
2:     $\forall y \in \mathcal{Y} \quad f_y = \begin{cases} \frac{e^{w\cdot\phi(x,y)}}{Z_Y} & \text{if } y \in Y \\ 0 & \text{otherwise} \end{cases}$
3:     $\forall y \in \mathcal{Y} \quad p_y = \frac{e^{w\cdot\phi(x,y)}}{Z_{\mathcal{Y}}}$
4:     for $i$ s.t. $\exists y, \phi_i(x,y) \neq 0$ do
5:         $\partial_i = \sum_{y\in\mathcal{Y}} \phi_i(x,y)(f_y - p_y)$
6:         $g_i \leftarrow g_i + \partial_i$
7:         $G_i \leftarrow G_i + \partial_i^2$
8:         $w_i \leftarrow \alpha \frac{1}{\sqrt{G_i}+\rho}\text{shrink}(g_i, j\lambda)$
9:         $j \leftarrow j + 1$
       return $w$
```

3.3 Probabilistic Data Exploration

A key component of dynamic-oracle training is that the training set D is not fixed in advance but changes according to the training progression. As we cannot explore the state set $\mathcal{X}$ in its entirety due to its exponential size (and because the optimal actions Y at a state x depend on the underlying sentence), we would like to explore regions of the state space that we are likely to encounter when parsing using the parameter

[3]For example, in our implementation, training on the English treebank with 950k tokens and 42 dependency labels requires almost 12GB of RAM for AdaGrad-RDA vs. less than 1.8GB for the averaged-perceptron.

vector w, together with their optimal actions Y according to the dynamic oracle.

That is, our set D is constructed by sampling values from $\mathcal{X}$ in accordance to our current belief w, and using the oracle $oracle(x; T)$ to associate Y values with each x. In the averaged-perceptron setup, this sampling is achieved by following the highest-scoring action rather than a correct one according to the oracle with some (high) probability p. This approach does not fit well with our probabilistic framework, for two reasons. (a) Practically, the efficiency of the AdaGrad optimizer results in the model achieving a good fit of the training data very quickly, and the highest scoring action is all too often a correct one. While great from an optimization perspective, this behavior limits our chances of exploring states resulting from incorrect decisions. (b) Conceptually, focusing on the highest scoring action ignores the richer structure that our probabilistic model offers, namely the probabilistic interpretation of the scores and the relations between them.

Instead, we propose a natural sampling procedure. Given a state x we use our model to obtain a multinomial distribution $P(y|x; w)$ over possible next actions y, sample an action from this distribution, and move to the state resulting from the sampled action.[4] This procedure focuses on states that the model has a high probability of landing on, while still allowing exploration of less likely states.

The training procedure is given in algorithm 3. In the first iteration, we focus on states that are on the path to the gold tree by following actions $\hat{y}$ in accordance to the oracle set Y (line 6), while on subsequent iteration we explore states which are off of the gold path by sampling the next action $\hat{y}$ in accordance to the model belief $P(y|x; w)$ (line 8).

Algorithm 3 Online training with exploration for probabilistic greedy transition-based parsers (ith iteration)

```
1:  for sentence $S$ with gold tree $T$ in corpus do
2:      $x \leftarrow$ INITIAL$(S)$
3:      while not TERMINAL$(x)$ do
4:          $Y \leftarrow \text{oracle}(x, T)$
5:          $P(y|x; w) \leftarrow \frac{\exp w\cdot\phi(x,y)}{\sum_{y'\in\mathcal{Y}} \exp w\cdot\phi(x,y')} \quad \forall y \in \mathcal{Y}$
6:          if $i \leq k$ then
7:              $\hat{y} \leftarrow \arg\max_{y\in Y} P(y|x; w)$
8:          else
9:              Sample $\hat{y}$ according to $P(y|x; w)$
10:         $w \leftarrow$ ADAGRADUPDATE$(w, \phi, x, Y)$
11:         $x \leftarrow \hat{y}(x)$
    return $w$
```

4 Evaluation and Results

Data and Experimental Setup We implemented the above training procedure in an Arc-Eager transition based parser, and tested it on the 6 languages comprising the freely available Google Universal Dependency Treebank (McDonald et al., 2013). In all cases, we trained on the training set and evaluated the models on the dev-set, using *gold POS-tags* in both test and train time. Non-projective sentences were removed from the training set. In all scenarios, we used the feature set of (Zhang and Nivre, 2011). We compared different training scenarios: training perceptron based models (PERCEP) and probabilistic models (ME) with static (ST) and dynamic (DYN) oracles. For the dynamic oracles, we varied the parameter k (the number of initial iterations without error exploration). For the probabilistic dynamic-oracle models further compare the sampling-based exploration described in Algorithm 3 with the error-based exploration used for training the perceptron models in Goldberg and Nivre (2012, 2013). All models were trained for 15 iterations. The PERCEP+DYN models are the same as the models in (Goldberg and Nivre, 2013). For the ME models, we fixed the values of $\rho = 0.01$, $\alpha = 1$ and $\lambda = 1/20|D|$ where we take $|D|$ to be the number of tokens in the training set.[5]

Parsing Accuracies are listed in Table 1. Two trends are emergent: Dynamic Oracles with Error Ex-

[4]Things are a bit more complicated in practice: as not all actions are valid at each state due to preconditions in the transition system, we restrict $P(y|x; w)$ to only the set of valid actions at x, and renormalize. In case x is a terminal state (and thus having no valid actions) we move on to the initial state of the next sentence. The sentences are sampled uniformly without replacement at each training round.

[5]We set the ρ and α values based on initial experiments on an unrelated dataset. The formula for the L_1 penalty λ is based on an advice from Alexandre Passos (personal communication) which proved to be very effective. We note that we could have probably gotten a somewhat higher scores in all the settings by further optimizing the ρ, α and λ parameters, as well as the number of training iterations, on held-out data.

SETUP	DE UAS / LAS	EN UAS / LAS	ES UAS / LAS	FR UAS / LAS	KO UAS / LAS	SV UAS / LAS
PERCEP+ST	84.95 / 80.32	91.06 / 89.48	85.93 / 82.82	85.75 / 82.44	79.96 / 71.90	83.21 / 79.40
ME+ST	84.71 / 80.06	90.83 / 89.32	85.72 / 82.59	85.42 / 82.19	80.47 / 72.15	83.12 / 79.36
PERCEP+DYN(K=1)	86.30 / 81.67	92.22 / 90.72	86.68 / 83.64	86.95 / 83.93	80.59 / 72.66	84.16 / 80.48
PERCEP+DYN(K=0)	86.50 / 81.88	92.28 / 90.82	86.18 / 83.19	86.87 / 83.70	80.59 / 73.06	84.79 / 81.00
ME+DYN(K=1,SAMPLE)	86.34 / 82.04	92.16 / 90.73	86.38 / 83.57	86.59 / 83.46	80.92 / 73.06	84.56 / 80.97
ME+DYN(K=0,SAMPLE)	86.51 / 82.19	92.30 / 90.83	86.66 /83.77	86.69 / 83.61	81.17 / 73.19	84.17 / 80.54

Table 1: Labeled (LAS) and Unlabeled (UAS) parsing accuracies of the different models on various datasets. All scores are excluding punctuations an using gold POS-tags. Dynamic-oracle training with error exploration clearly outperforms static-oracle training. The perceptron and ME results are equivalent.

SETUP	DE UAS / LAS	EN UAS / LAS	ES UAS / LAS	FR UAS / LAS	KO UAS / LAS	SV UAS / LAS
ME+DYN(K=1,ERR)	85.26 / 80.94	91.62 / 90.10	86.08 / 82.92	86.13 / 83.06	80.42 / 71.97	83.73 / 80.03
ME+DYN(K=0,ERR)	85.78 / 81.63	91.77 / 90.30	86.37 / 83.33	86.53 / 83.23	80.94 / 72.57	83.73 / 80.05
ME+DYN(K=1,SAMPLE)	86.34 / 82.04	92.16 / 90.73	86.38 / 83.57	86.59 / 83.46	80.92 / 73.06	84.56 / 80.97
ME+DYN(K=0,SAMPLE)	86.51 / 82.19	92.30 / 90.83	86.66 /83.77	86.69 / 83.61	81.17 / 73.19	84.17 / 80.54

Table 2: Comparing the sampling-based exploration in Algorithm 3 with the error-based exploration of Goldberg and Nivre (2012, 2013). Labeled (LAS) and Unlabeled (UAS) parsing accuracies of the different models on various datasets. All scores are excluding punctuations an using gold POS-tags. The sampling based algorithm outperforms the error-based one.

ploration in training (DYN) models clearly outperform the models trained with the traditional static oracles (ST), and the probabilistic models (ME) perform on par with their averaged-perceptron (PERCEP) counterparts.

Sampling vs. Error-Driven Exploration Table 2 verifies that the sampling-based exploration proposed in this work is indeed superior to the error-based exploration which was used in Goldberg and Nivre (2012, 2013), when training multinomial logistic-regression models with the AdaGrad-RDA algorithm.

Model Sizes Table 3 lists the number of parameters in the different models. RDA regularization is effective: the regularization ME models are much smaller. In the accurate dynamic oracle setting, the regularized ME models are 3-4 times smaller than their averaged-perceptron counterparts, while achieving roughly the same accuracies.

Calibration To asses the quality of the probabilistic output of the ME+DYN models, we binned the probability estimates of the highest-scored actions into 10 equally-sized bins, and for each bin calculated the percentage of time an action falling in the bin was correct. Table 4 lists the results, together with the bin sizes.

SETUP	DE	EN	ES	FR	KO	SV
PERCEP+ST	438k	5.4M	1.2M	849k	1.9M	912k
ME+ST	150k	1.9M	448k	294k	725k	304k
PERCEP+DYN	525k	8.5M	1.7M	1.1M	2.9M	1.2M
ME+DYN	160k	2.4M	516k	336k	755k	357k

Table 3: Model sizes (number of non-0 parameters).

First, it is clear that the vast majority of parser actions fall in the 0.9-1.0 bin, indicating that the parser is confident, and indeed the parser is mostly correct in these cases. Second, the models seem to be well calibrated from the 0.5-0.6 bin and above. The lower bins are under-estimating the confidence, but they also contain very few items. Overall, the probability output of the ME+DYN model is calibrated and trustworthy.[6]

5 Conclusions

We proposed an adaptation of the dynamic-oracle training with exploration framework of Goldberg and Nivre (2012; 2013) to train classifiers with probabilistic output, and demonstrated that the method works:

[6]Note, however, that with 69k predictions in bin 0.9-1.0 for English, an accuracy of 98% means that almost 1400 predictions with a probability score above 0.9 are, in fact, wrong.

BIN	DE	EN	ES	FR	KO	SV
0.1	(7) 71%	(1) 0%	(3) 66%	(2) 1%	(0) 0%	(2) 50%
0.2	(51) 51%	(38) 55%	(26) 57%	(17) 64%	(2) 100%	(21) 57%
0.3	(121) 54%	(139) 54%	(83) 65%	(58) 72%	(29) 55%	(100) 61%
0.4	(292) 54%	(323) 62%	(206) 65%	(146) 57%	(178) 63%	(193) 63%
0.5	(666) 66%	(1.2k) 64%	(642) 68%	(453) 66%	(464) 55%	(578) 62%
0.6	(787) 66%	(1.4k) 69%	(694) 73%	(469) 70%	(616) 60%	(636) 70%
0.7	(840) 73%	(1.7k) 74%	(853) 77%	(546) 73%	(739) 65%	(747) 75%
0.8	(1.5k) 78%	(2.9k) 82%	(1.2k) 80%	(810) 78%	(1.1k) 72%	(1.2k) 80%
0.9	(18.5k) 97%	(69k) 98%	(16k) 97%	(13k) 97%	(9k) 96%	(14k) 96%
1.0	(800) 100%	(1.7k) 100%	(370) 100%	(366) 100%	(588) 100%	(493) 100%

Table 4: Calibration of the ME+DYN(K=0,SAMPLE) scores. (num) denotes the number of items in the bin, and num% the percent of correct items in the bin. The numbers for ME+DYN(K=1,SAMPLE) are very similar.

the trained classifiers produce well calibrated probability estimates, provide accuracies on par with the averaged-perceptron trained models, and, thanks to regularization, are 3-4 times smaller. However, the training procedure is slower than for the averaged-perceptron models, requires considerably more memory, and has more hyperparameters. If probabilistic output or sparse models are required, this method is recommended. If one is interested only in 1-best parsing accuracies and can tolerate the larger model sizes, training with the averaged-perceptron may be preferable.

References

Eugene Charniak and Mark Johnson. 2005. Coarse-to-fine n-best parsing and MaxEnt discriminative reranking. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 173–180.

Jinho D. Choi and Andrew McCallum. 2013. Transition-based dependency parsing with selectional branching. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1052–1062, Sofia, Bulgaria, August. Association for Computational Linguistics.

John Duchi, Elad Hazan, and Yoram Singer. 2010. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159.

Yoav Goldberg and Joakim Nivre. 2012. A dynamic oracle for arc-eager dependency parsing. In *Proceedings of the 24th International Conference on Computational Linguistics (COLING)*.

Yoav Goldberg and Joakim Nivre. 2013. Training deterministic parsers with non-deterministic oracles. *Transactions of the association for Computational Linguistics*, 1.

Spence Green, Sida Wang, Daniel Cer, and Christopher D. Manning. 2013. Fast and adaptive online training of feature-rich translation models. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 311–321, Sofia, Bulgaria, August. Association for Computational Linguistics.

Sandra Kübler, Ryan McDonald, and Joakim Nivre. 2009. *Dependency Parsing*. Morgan and Claypool.

Ryan McDonald, Joakim Nivre, Yvonne Quirmbach-Brundage, Yoav Goldberg, Dipanjan Das, Kuzman Ganchev, Keith Hall, Slav Petrov, Hao Zhang, Oscar Täckström, Claudia Bedini, Núria Bertomeu Castelló, and Jungmee Lee. 2013. Universal dependency annotation for multilingual parsing. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 92–97, Sofia, Bulgaria, August. Association for Computational Linguistics.

Joakim Nivre, Johan Hall, Sandra Kübler, Ryan McDonald, Jens Nilsson, Sebastian Riedel, and Deniz Yuret. 2007. The CoNLL 2007 shared task on dependency parsing. In *Proceedings of the CoNLL Shared Task of EMNLP-CoNLL 2007*, pages 915–932.

Joakim Nivre. 2003. An efficient algorithm for projective dependency parsing. In *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT)*, pages 149–160.

Joakim Nivre. 2008. Algorithms for deterministic incremental dependency parsing. *Computational Linguistics*, 34:513–553.

Stephan Riezler, Margaret H. King, Ronald M. Kaplan, Richard Crouch, John T. Maxwell III, and Mark Johnson. 2002. Parsing the Wall Street Journal using a Lexical-Functional Grammar and discriminative estimation techniques. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 271–278.

Kenji Sagae and Alon Lavie. 2006. A best-first probabilistic shift-reduce parser. In *Proceedings of the COLING/ACL 2006 Main Conference Poster Sessions*, pages 691–698.

Lin Xiao. 2010. Dual averaging methods for regularized stochastic learning and online optimization. *Journal of Machine Learning Research*, 9:2543–2596.

Yue Zhang and Joakim Nivre. 2011. Transition-based dependency parsing with rich non-local features. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 188–193.

Kai Zhao, James Cross, and Liang Huang. 2013. Dynamic programming for optimal best-first shift-reduce parsing. In *EMNLP*.