Higher-Order Dependency Parsing for Arc-Polynomial Score Functions via Gradient-Based Methods and Genetic Algorithm

Anonymous ACL submission

Abstract

We present a novel method for higher-order dependency parsing which takes advantage of the general form of score functions written as arc-polynomial functions, a general framework which encompasses common higher-order score functions, and includes new ones. This method is based on non-linear optimization techniques, namely coordinate ascent and genetic search where we iteratively update a candidate parse. Updates are formulated as gradient-based operations, and are efficiently computed by auto-differentiation libraries. Experiments show that this method obtains results matching the recent state-of-the-art second order parsers on three standard datasets.

1 Introduction

The goal of modern graph-based dependency parsing is to find the most adequate parse structure for the given input sentence by computing a score for all possible candidate parses, and returning the highest-scoring one. Since the number of candidates is exponential in the sentence length, the scoring is performed implicitly: after computing scores for possible parts, the best structure, whose score is the sum of its various parts, is returned by a combinatorial algorithm based on either dynamic programming such as the Eisner algorithm (Eisner, 1997) in the projective case, or duality gap such as the Chu-Liu-Edmonds algorithm (McDonald et al., 2005) in the non-projective case.

Graph-based models where parts are restricted to single arcs are called first-order models, while models where parts contains k-tuples of arcs are called k^{th}-order models. For instance models with score for sibling and grand-parent relations are 2^{nd}-order models because parts consist of 2 *connected* arcs. The connectivity is important since it helps building efficient dynamic programming algorithms in the case of projective arborescences (Koo and Collins, 2010) or efficient approximations in the non-projective case based on lagrangian heuristics (Koo et al., 2010; Martins et al., 2013) or belief propagation (Smith and Eisner, 2008). The score function of first-order models, being a sum of parts which are simple arcs, is linear in arc variables, while for second-order, being a sum of parts which are pair of arcs, the score function is quadratic in arc variables. More generally k^{th}-order models have a polynomial score function in arc variables, with highest degree equal to k.

In this paper we explore the consequences of treating score functions for higher-order dependency parsing as polynomial functions. This framework can recover most previously defined score functions and gives a unified framework for graph-based parsing. Moreover, it can express novel functions since in this setting parts are made of possibly disconnected tuples of arcs. We call the results *generalized* higher-order models, as opposed to previously *connected* higher-order models.

On the other hand, polynomial functions are difficult to manipulate. They are non-convex and so, in addition to already known problems in higher-order parsing such as the computation of the partition function for probabilitic models, MAP decoding is itself a challenge. We develop an approximate parsing strategy based on coordinate ascent (Bertsekas, 1999), where we iteratively improve a candidate by flipping arcs. We exploit the polynomial nature of the score function to derive an accurate and efficient procedure to select arcs to be flipped. Since coordinate ascent converges to a local minimum, we show how this method can be embedded within a meta-heuristics based on genetic analogy (Schmitt, 2001) to find better optima.

We can learn these models via two methods, max-margin or probabilitic estimation. Max-margin is straightforward because it only requires MAP decoding but is quite fragile since it is sensitive to approximation errors which are inevitable in our setting. We design a probabilistic loss for

our model where we approximate parse scores via a first-order Taylor expansion around the MAP solution. We find that this novel method is efficient and we show empirically that it can outperform previous higher-order models.

In summary our contributions are the following:

- a general framework for dependency parsing which encompasses previous higher-order score functions, and includes new ones;
- a new method for higher-order dependency parsing based on non-linear optimization techniques (coordinate ascent and genetic algorithm) coupling gradient-based methods, and combinatorial routines;
- an empirical validation of this method which obtains state-of-the-art results on standard datasets and is computationally efficient.

2 Related Work

Before the use of powerful neural feature extractors (*e.g.* BiLSTM or Transformers) dependency parsing with high-order relations was a clear improvement over first-order models. Koo and Collins (2010) considered efficient third order models for projective dependency parsing. In order to have efficient dynamic programming algorithms for decoding, only a few limited predefined structures can be included to the model (*e.g.* dependency, sibling, grandchild, grand-sibling, tri-sibling).[1]

Higher-order non-projective parsing is NP-hard but fast heuristics with good performance have been proposed based on dual decomposition for instance. However, efficient subsystems must be devised to efficiently process complex parts, either based on dynamic programming algorithms such as Viterbi (Koo et al., 2010) or on integer linear programming (Martins et al., 2013). In practice this restricts parts to connected subgraphs.[2]

Since the wide adoption of deep feature extractors, the situation is less clear. (Zhang et al., 2020) consider a second-order model with dependency and adjacent sibling, which can guarantee efficient decoding for projective arborescence with a batchified variant of Eisner algorithm (Eisner, 1996, 1997). The results show that adjacent sibling is beneficial for the performance of parser comparing with arc-factored model. (Fonseca and Martins, 2020) claim that in the non-projective case, second-order features help especially in long sentences. On the other hand, (Falenska and Kuhn, 2019) showed that in general the impact of consecutive sibling features was not substantial, and (Zhang et al., 2021) showed that the main benefit of these features could be understood as variance reduction, and vanishes when ensembles are used.

Closely related to our work, Wang and Tu (2020) consider a second-order model with score for dependencies, siblings and grandchildren where they do not constrain siblings to be adjacent. Although exact estimation is intractable in their setting, an approximate estimation of probability of arborescences can be calculated efficiently by a message-passing algorithm. Their experiments seem to confirm that second-order relations are beneficial to the parsing accuracy, even when trained by an approximate estimation of probability, namely Mean-Field Variational Inference. Instead we approximate the partition function using a first-order Taylor approximation around the solution of the MAP solution. Partition approximations are usually performed via Bethe's free energy, see for instance (Martins et al., 2010; Wiseman and Kim, 2019).

Dozat and Manning (2017) showed that head selection was a good trade-off during the learning phase, for first-order models. Our method applies this principle to the higher-order case, leading to a coordinate ascent method, well known in the optimization literature (Bertsekas, 1999). In Machine Learning and NLP, ascent methods are usually performed in primal-dual algorithms, *e.g.* (Shalev-Shwartz and Zhang, 2013) for SVMs.

We use genetic programming to escape local optima when searching for the best parse. Although this kind of metaheuristics has been used for other tasks in NLP such as WSD (Decadt et al., 2004) or summarization (Litvak et al., 2010), it is the first time it is applied to dependency parsing to the best of our knowledge. Since genetic algorithms can be seen as implementing a Markov-Chain (Schmitt, 2001) over candidate solutions, our method resembles MCMC methods, related to Gibbs sampling for Metropolis-Hastings methods, which have already been investigated in parsing (Zhang et al., 2014; Gao and Gormley, 2020). Our method to choose the best arc to improve the current parse is inspired by a recent method for sampling in discrete distributions (Grathwohl et al., 2021) where we replace sampling by MAP inference.

[1] The term *sibling* often means *adjacent sibling*, where only adjacent modifiers on the same side of the head are included.

[2] We note that Martins et al. (2013) used a 2-arc part called *adjacent modifiers* which is not a connected subgraph. But this was not generalized to 2-arc arbitrary subgraphs.

We rely on properties of polynomials to derive efficient routines for approximate head selection. Polynomial factors where discussed for higher-order parsing in (Qian and Liu, 2013).

3 Notations

We write a sentence $x = x_0, x_1, \ldots, x_n$, with n the number of words, x_i the dummy root symbol when $i = 0$, or the i^{th} word otherwise.

For $h, d \in [n]$, with $[n] = \{0, 1, \ldots, n\}$, (h, d) represents a direct arc form head x_h to dependent x_d. We note y a parse structure, with $(h, d) \in y$ if (h, d) is an arc of the parse. The set of all valid parses for sentence x is noted $\mathcal{Y}_x$, with $y \in \mathcal{Y}_x$ if y is a valid parse for x. When x is clear from the context, we simplify $\mathcal{Y}_x$ to $\mathcal{Y}$. When the parse structure is important, we will distinguish $\mathcal{A}_x$ the set of all valid arborescences for x rooted in x_0, $\mathcal{P}_x$ the set of all valid projective arborescences for x rooted in x_0, and $\mathcal{G}_x$ the set of all directed graphs with vertices in x where each x_i has one and only one entering arc for $i > 0$, and x_0 has no entering arc. We note in passing that for all x we have $\mathcal{P}_x \subseteq \mathcal{A}_x \subseteq \mathcal{G}_x$.

For convenience, we will abuse notation and sometimes interpret a parse y either as a vector indexed by arcs or by a matrix:

$$y_{hd} = \begin{cases} 1 & \text{if } (h, d) \text{ is present in parse} \\ 0 & \text{otherwise} \end{cases}$$

We note C_x as the set of all possible arcs for sentence x (the arcs of the complete graph over vertices in x) or C when unambiguous.

We say that a non-empty set of arcs $A = \{(h_1, d_1), \ldots (h_k, d_k)\}$ is *proper* if $\forall i, h_i \neq d_i$ and $\forall i < j, d_i \neq d_j$. The first condition asserts that an arc cannot be a self-loop while the second enforces that each word has only one head in a proper set. The two constraints are natural and required for dependency parsing. We note the set of proper subsets of cardinal k which can be constructed from a set of arcs A as $\mathcal{F}_k(A)$, the set of k^{th}-order polynomial factors. Finally, for $(h, d) \in y$, we use $l(y)_{hd}$ to represent the label for the arc (h, d), or l_{hd} when y is clear from the context.

4 Polynomial Score Functions for Dependency Parsing

In this work, we consider a generalization of previously proposed score functions for graph-based dependency parsing. Unlike higher-order models which consider only limited higher-order relations, *e.g.* (Koo and Collins, 2010), the proposed function can express all possible higher-order relations and can be viewed as a natural generalization of (Wang and Tu, 2020; Zhang et al., 2020).

4.1 Score Function

We define K^{th}-order score functions as:

$$\begin{aligned} S(x, y) &= \sum_{k=1}^{K} \sum_{F \in (\mathcal{F}_k(C) \cap \mathcal{R})} s_F \prod_{(h,d) \in F}^{k} y_{hd} \\ &= \sum_{k=1}^{K} \sum_{F \in (\mathcal{F}_k(y) \cap \mathcal{R})} s_F \end{aligned} \tag{1}$$

where s_F represents the score for the higher-order factor constructed from arcs in F, and $\mathcal{R}$ is set of authorized structures (the *restriction*). Remark that Eq. (1) does not enforce a specific structure for $y \in \mathcal{Y}$ and could be applied in $\mathcal{G}, \mathcal{A}, \mathcal{P}$.

With this general definition we can recover most previous models for graph-based dependency parsing. For instance, in (Wang and Tu, 2020), a second order model ($K = 2$) is studied where only sibling and grandchild relations are considered, which can be expressed with the following $\mathcal{R}$: for $F = \{(h_1, d_1), (h_2, d_2)\}$, we enforce $h_1 = h_2$ or $d_1 = h_2$. In (Zhang et al., 2020), another second-order model, the restriction is stricter in order to limit acceptation to adjacent siblings: $h_1 = h_2$ and $(h_1, d_1), (h_2, d_2)$ are adjacent (no arcs from h_1, h_2 to word between d_1, d_2).

To demonstrate the generality of this approach, we also consider a generalized third-order model. The first-order and the second-order parts are as (Wang and Tu, 2020), and for third-order factors $F = \{(h_1, d_1), (h_2, d_2), (h_3, d_3)\}$, we add restrictions $d_1 < d_2 < d_1 + 3$ and $d_2 < d_3 < d_2 + 3$. Arcs in F are not always connected. Instead, we only force the modifiers of arcs to be close, with a maximum distance set to 2. This addition of cubic factors could be a computational bottleneck since it would naively require computing $O(n^6)$ scores. We avoid this with tensor factorization following (Peng et al., 2017).[3]

4.2 Score of One-Arc Modifications

Parsing can be framed as finding the highest $S(x, y)$, or $S(y)$ when x is unambiguous:

$$y^* = \underset{y \in \mathcal{Y}}{\operatorname{argmax}}\, S(y) \tag{2}$$

[3] See Appendix C for details.

The solution is tractable for $K = 1$ (first-order model), *i.e.* arc-factored model, for all usual parse structures, such as $\mathcal{G}, \mathcal{A}, \mathcal{P}$. However, it is intractable without additional constraints for higher-order models, such as projectivity for parses and adjacent siblings in scores.

We consider here a simpler problem: how much can the score change if we change **one** arc of the current parse? The idea is that better parses may be obtained by choosing arcs to be flipped. Thus, even starting with a *bad* parse, we may approach the *best* parse by modifying one arc at a time.

To solve this simpler problem, the naive method, *i.e.* calculate the score of every parse which differs from the current parse by one arc, is obviously inefficient and unpractical since it requires $O(n^2)$ evaluations (for each modifier and each head). Instead, we show that the score change of a one-arc modification can be efficiently calculated for Eq. (1). Let us consider the current parse y and an arbitrary arc $a = (h, d) \in C$ (possibly not in y). The partial derivative of the score to variable y_a is:[4]

$$\begin{aligned}\frac{\partial S(y)}{\partial y_a} &= \sum_{k=1}^{K} \sum_{F \in (\mathcal{F}_k(C) \cap \mathcal{R})} s_F \frac{\partial \prod_{a' \in F} y_{a'}}{\partial y_a} \\ &= \sum_{k=1}^{K} \sum_{\substack{F \in (\mathcal{F}_k(C) \cap \mathcal{R}), \\ a \in F}} s_F \mathbf{1}[F \backslash a \in \mathcal{F}_{k-1}(y)]\end{aligned} \tag{3}$$

We can interpret this formula for the partial derivative as the sum of all factors F including a which verify $(F \backslash a) \in \mathcal{F}_{k-1}(y)$. When $a \in y$, $\frac{\partial S(y)}{\partial y_a}$ can be seen as the restriction of $S(y)$ to factors $F \subseteq \mathcal{F}_k(y)$, where $a \in F$. And we can write:[5]

$$S(y) = \frac{\partial S(y)}{\partial y_a} + S(y \backslash a) \tag{4}$$

where the last term is the score of all factors in y which do not contain a.

When $a \notin y$, we note $a = (h', d)$ while we assume $(h, d) \in y$.We define $y[h \to h', d]$ as the parse which modifies y by swapping the head index for column d from h to h' while the other columns remain unchanged, and $y[\to h', d]$ when the current head h is unimportant. We can rewrite the score function of $y[h \to h', d]$ with the previously defined partial derivative, and take advantage of the score factorisation to express $S(y[h \to h', d])$ with quantities directly available on y:[6]

$$S(y[h \to h', d]) = \frac{\partial S(y)}{\partial y_{h'd}} + S(y \backslash (h, d)) \tag{5}$$

Remark that the right part of the equation concerns only the original arborescence y.

We write $D(y \to h', d)$ for the change of score induced by swapping the head in column d to h', and $D(y, h \to h', d)$ when we want to emphasize that the current head for d is h. From the previous equations, we can derive:

$$\begin{aligned}D(y, h \to h', d) &= S(y[(h \to h', d)]) - S(y) \\ &= \frac{\partial S(y)}{\partial y_{h'd}} - \frac{\partial S(y)}{\partial y_{hd}}\end{aligned} \tag{6}$$

Thus, to have a complete evaluation of change of scores, we only need one forward and backward evaluation on the score of the current solution y and then compute differences for each position d. In the following section, we build an inference algorithm based on this observation

5 Inference as Candidate Improvement

5.1 Coordinate Ascent

The main idea of our method is, from an initial parse y_0, to change the current candidate by picking a word and swapping its head to improve the score function. This is repeated until not further improvement is possible. This method is an instance of coordinate ascent (Bertsekas, 1999) (Chap. 2.7), to maximize Eq. (1). When parses are arborescences, such as when working in $\mathcal{A}$ and $\mathcal{P}$, this method must at each step, not only pick an improving arc, but also assert that the resulting parse has the required tree structure. This adds complexity that we propose to avoid by working in $\mathcal{G}$ and inserting a final step of projection to recover a solution in the desired space (described in Section 6.2).

Remark that when arborescence constraints are dropped, finding the best parse reduces to head selection, *i.e.* choose $h_d, \forall d$ with $y_{h_d,d} = 1$, which maximizes $S(y)$. To emphasize that this method works *column by column* we write:

$$S(x, y) = S(y_{:,1}, \ldots, y_{:,|x|})$$

[4] See Appendix B.1 for the detailed derivation.

[5] See Appendix B.2 for the detailed derivation.

[6] See Appendix B.3 for the detailed derivation.

where $y_{:,d}$ denotes the one-hot vector where $y_{:,d}[h] = 1$ if $(h, d) \in y$.

This is straightforward and tractable for first-order models, since it amounts to maximizing independent score functions. However, this becomes intractable in higher-order models since parts overlap. Still, a local optimum can be obtained by coordinate ascent.

Given a current solution y^k, basic coordinate ascent finds a better next iterate y^{k+1} by cycling through columns and improving the current solution locally by successive head selections:

$$h_d^* = \text{argmax}_h \, S(y_{:,1}^{k+1}, \ldots, y_{:,d-1}^{k+1}, \xi_h, y_{:,d+1}^k \ldots, y_{:,|x|}^k) \tag{7}$$

where ξ_h is the one-hot vector with 1 at position h. We set $y_{:,i}^{k+1} = \xi_{h_d^*}$ and the process is repeated for every word until there is no change ($y^{k+1} = y^k$).

5.2 A Gradient-based Method For Coordinate Ascent

A naive method to solve Eq. (7) requires n evaluations of S, one per possible head, which is inefficient. However, from Section 4.2 and Eq. (6), we can rewrite Eq. (7) since it amounts to finding a better head at position d from current solution y:

$$h_d^* = \underset{h}{\text{argmax}} \, D(y \rightarrow h, d) \tag{8}$$

Thus, one forward and one backward (followed by $|x|$ substractions) is sufficient to decide the modification of arc at each position d.

Still, the gradient-based maximization presented above requires n forward and backward passes to determine the new heads for all words of the sentence. In order to achieve faster convergence, we want to avoid cycling through each word and consider the following problem: at each step, find the pair (h, d) which provides the greatest positive change in the score function:

$$(h^*, d^*) = \text{argmax}_{h,d} \, S(y_{:,1}^k, \ldots, y_{:,d-1}^k, \xi_h, y_{:,d+1}^k \ldots, y_{:,|x|}^k) \tag{9}$$

We set $y^{k+1} = y^k[\rightarrow h^*, d^*]$ while other columns are unchanged. This is repeated until $y^{k+1} = y^k$.

Again, a naive maximization requires $O(n^2)$ estimations of score for each step and brings in fact no speed gain. However, as we have already seen, Eq. (9) is simply equivalent to:

$$(h^*, d^*) = \underset{h,d}{\text{argmax}} \, D(y, \rightarrow h, d) \tag{10}$$

which again requires one forward and backward on the current candidate's score before substractions.

In summary our algorithm, from an initial parse y_0, iteratively improves a current solution: at step k we solve Eq. (10) by computing the gradient of $S(y^k)$ over arc variables and then pick the arc (h, d) whose partial derivative increases the greatest to set $y^{k+1} = y^k[\rightarrow h, d]$.

5.3 First-Order Linearization

Coordinate ascent changes one arc at a time which can still be slow. In practice, we found that a simpler greedy method performed at the beginning of the search, when high precision is not required, can improve parsing time dramatically. Given a current solution y^k, we linearize the score function via the first-order Taylor approximation and apply coordinate ascent to what is now an arc-factored model where columns can be processed independently. For each sentence position d:[7]

$$h_d^* = \underset{h}{\text{argmax}} \, \frac{\partial S(y^k)}{\partial y_{hd}^k}.$$

We set then $y_{:,d}^{k+1} = \xi_{h_d^*} \forall d > 0$. We may be able to change $|x|$ arcs at each step k, and the process is repeated until $S(y^{k+1}) \leq S(y^k)$, which indicates that the approximation becomes detrimental, after which we switch to coordinate ascent to provide more accurate iterations.

5.4 Genetic Algorithm

Due to the non-convexity of function S, the previous method gives a local optimum, which may limit the usefulness of higher-order parts. Thus, to ensure a better approximation, we add genetic search (Mitchell, 1998).

Genetic Algorithm is an evolutionary algorithm inspired by the process of natural selection. The algorithm requires: a solution domain, here $\mathcal{G}$, and a fitness function, *i.e.* function $S(y)$. Each step in our genetic algorithm consists of four consecutive processes: selection, crossover, mutation and self-evolution, which are repeated until stabilization.

Selection For a group of parses $y_1, \ldots, y_w$, estimate scores $S(y_1), \ldots, S(y_w)$. Select the top-k candidates ($k < w$) $y_1^s, \ldots, y_k^s$.

Crossover Average candidates $y^c = \frac{1}{k} \sum_{i=1}^k y_k^s$. Set $y_{h,d}^c$ as the probability of having (h, d) in an optimal parse and sample $w - k$ new parses according to y^c. Note them $y_1^c, \ldots, y_{w-k}^c$.

[7] See Appendix B.4 for the detailed derivation.

Mutation For every parse in $y^c_1, \ldots, y^c_{w-k}$, change heads randomly with probability p. Note mutated parses as $y^m_1, \ldots, y^m_{w-k}$

Self-Evolution On parses $y^m_1, \ldots, y^m_{w-k}$, apply coordinate ascent. Note the output as $y^e_1, \ldots, y^e_{w-k}$. Combine new parses with the previous top-k parses as the group for next iteration.

Selection and self-evolution pick arcs giving high scores while crossover and mutation can provide the possibility to jump out of local optima. We iterate this process until the best parse is unchanged for t consecutive iterations.

6 Learning and Decoding with Polynomial Scores

6.1 Learning

We follow recent works (Zhang et al., 2020; Wang and Tu, 2020) and learn parse structures and arc labels in a multitask fashion with a shared feature extractor. Loss is the sum of label and arc losses:

$$L = L_{\text{label}} + L_{\text{arc}} \tag{11}$$

We write (x^*, y^*, l^*) as the training input sentence and its corresponding parse and labeling.

Label Loss Following (Dozat and Manning, 2017), we use the negative log-likelihood:

$$L_{\text{label}}(x^*, y^*, l^*) = -\sum_{(h,d)\in y^*} \log p(l^*_{hd}|x^*).$$

Hinge Loss Following (Kiperwasser and Goldberg, 2016), we can use hinge loss as arc loss:

$$L_{\text{arc}} = \text{ReLU}(\max_{y\in\mathcal{Y}} S(x^*, y) - S(x^*, y^*) + \Delta(y, y^*))$$

where $\Delta(y, y^*)$ is the Hamming distance.

The inner maximization requires to solve an inference sub-problem, *i.e.* to find the cost-augmented highest-scoring parse:

$$\max_{y\in\mathcal{Y}} S(x^*, y) + \Delta(y, y^*) \tag{12}$$

As Hamming distance is not differentiable, we propose to reformulate it as:

$$\Delta(y, y^*) = \sum_{h,d} (1 - y_{hd}) y^*_{hd} + (1 - y^*_{hd}) y_{hd}$$

linear to the variable y. Thus, Eq. (12) can be solved with the method proposed in Section 5.

Approximate Marginal Estimation In practice hinge loss may have two issues: each update is limited to two parses only, which makes learning slow, and the linear margin may lead to insufficient learning. We thus propose an approximate probabilistic learning objective inspired by methods such as Mean-Field Variational Inference (Wang and Tu, 2020). We would like to train our model as an arc-factored log-linear model:

$$L_{\text{arc}} = -\sum_{(h,d)\in y*} \log p\big((h, d)|x^*\big)$$

where $p\big((h, d)|x^*\big)$ is the marginal probability of arc (h, d) over all parses for x^*.

Marginal probabilities are approximated based on the intuition that the distribution of parses is usually peaked on few close solutions, hence that estimating the contribution of arcs at the neighborhood of the highest-scoring parse gives an acceptable approximation. We use the same reasoning as in Section 5.3 to derive a linear approximation of the current model. Given parse $\hat{y}$ obtained by coordinate ascent, we set:[8]

$$\begin{aligned} p\big((h, d)|x^*\big) &= \frac{p(\hat{y}[\rightarrow h, d])}{\sum_{h'} p(\hat{y}[\rightarrow h', d])} \\ &\approx \frac{\exp(s_{hd})}{\sum_{h'} \exp(s_{h'd})} \end{aligned} \tag{13}$$

where:

$$s_{hd} = \frac{\partial S(\hat{y})}{\partial y_{hd}} \tag{14}$$

6.2 Approximate MBR Structured Decoding

Inference with coordinate ascent and genetic algorithm cannot guarantee the tree structure of parses, as they work in solution space $\mathcal{G}$. But we can estimate the marginal probability of arcs from a solution y returned by coordinate ascent by reusing Eq. (13). Then, the Eisner algorithm (Eisner, 1996, 1997) or the Chu-Liu-Edmonds algorithm (McDonald et al., 2005) can be applied to have projective or non-projective arborescences. We remark that this is similar to Minimum Bayesian Risk (MBR) decoding (Smith and Smith, 2007), the difference being that here marginalization is estimated with nearest arborescences while for MBR marginalization is exact over the parse forest.

[8] See detailed derivation in Appendix B.5.

	bg	ca	cs	de	en	es	fr	it	nl	no	ro	ru	Avg.
CRF2O	90.77	91.29	91.54	80.46	87.32	90.86	87.96	91.91	88.62	91.02	86.90	93.33	89.33
Local2O	90.53	92.83	92.12	81.73	89.72	92.07	88.53	92.78	**90.19**	91.88	85.88	92.67	90.07
CA+LM	90.79	93.14	**91.92**	**84.45**	**89.89**	**92.60**	**90.14**	93.57	89.89	**93.85**	**86.42**	93.81	**90.87**
3O+CA+LM	**90.80**	93.09	91.91	84.42	89.75	92.50	90.02	93.53	90.13	93.78	86.38	93.86	90.85
GA+CA+LM	90.70	**93.17**	91.90	84.19	89.77	92.50	89.88	**93.68**	90.13	93.81	86.33	**93.88**	90.83
+BERT													
Local2O	91.13	93.34	92.07	81.67	90.43	92.45	89.26	93.50	90.99	91.66	86.09	92.66	90.44
CA+LM	**91.93**	**94.09**	92.46	**85.59**	90.97	93.42	90.88	94.18	91.49	94.57	87.22	94.40	**91.77**
3O+CA+LM	91.87	94.05	**92.50**	85.22	91.04	**93.47**	90.79	**94.26**	91.38	**94.62**	87.18	94.41	91.73
GA+CA+LM	91.86	94.08	92.49	85.38	**90.99**	93.44	**91.05**	94.13	**91.53**	94.56	**87.25**	**94.42**	**91.77**

Table 1: LAS on UD 2.2 test data. CRF2O: (Zhang et al., 2020); Local2O: (Wang and Tu, 2020).

7 Experiments

We present experimental results[9] where we evaluate and compare our parsing method where we use the score function (Wang and Tu, 2020) and our extension with third-order factors (3O) with coordinate ascent (CA) and genetic algorithm (GA).

7.1 Data

Two datasets are used for projective dependency parsing: the English Penn Treebank (PTB) with Stanford Dependencies (Marcus et al., 1993) and CoNLL09 Chinese data (Hajič et al., 2009). We use standard train/dev/test splits and evaluate with UAS/LAS metrics. Punctuation is ignored on PTB for dev and test. For non-projective dependency parsing, Universal Dependencies (UD) v2.2 is used. Following (Wang and Tu, 2020), punctuation is ignored for all languages.

For experiments with BERT (Devlin et al., 2019), we use BERT-Large-Uncased for PTB, BERT-Base-Chinese for CoNLL09 Chinese and Base-Multilingual-Cased for UD.

7.2 Hyper-Parameters

To ensure fair comparison, and for budget reasons, we use the same setup (hyper-parameters and pre-trained embeddings) as (Zhang et al., 2020).[10]

For experiments without BERT (Devlin et al., 2019), pos-tags are used for all datasets[11]. For experiments with BERT, the last 4 layers are combined by scalarmix and linear projection and then concatenated to the original feature vectors.

Initial candidates are sampled from the the first-order part of Eq. (1). For genetic algorithm, due to hardware memory limitations, the number of candidates is set to 6. Each time, we take the top-3 candidates in selection, and the genetic loop is terminated when the best parse remains unchanged for 3 consecutive iterations. The mutation rate is set to 0.2 on all datasets.[12]

All experiments are run 3 times with random seed set to current time and averaged. We rerun also the results of (Wang and Tu, 2020) on PTB and CoNLL09 with the authors' implementation.[13] to have a faire comparison.

Method	PTB		CoNLL09	
	UAS	LAS	UAS	LAS
CA+hinge	95.69	93.89	91.25	89.52
GA+CA+hinge	95.71	93.87	**91.52**	**89.80**
CA+LM	95.67	93.88	91.31	89.66
3O+CA+LM	95.64	93.87	91.26	89.61
GA+CA+LM	**95.81**	**93.99**	91.30	89.66
+BERT				
CA+LM	**96.53**	**94.85**	**93.18**	**91.57**
3O+CA+LM	96.47	94.79	93.15	91.53
GA+CA+LM	96.50	94.82	93.16	91.55

Table 2: Comparison on dev. CA: Coordinate Ascent; 3O: Third order model; GA: Genetic Algorithm; LM: Linearized Marginalization; hinge: hinge loss

7.3 Results on PTB and CoNLL09 Chinese

Table 2 shows results of our different system with and without BERT. For PTB without BERT we see that training via coordinate ascent with hinge loss of marginal estimation give similar results, while genetic algorithm gives a sensible improvement when combined with the probabilistic framework.

[9] Our prototype will be publicly available upon publication.

[10] See Appendix A.

[11] In (Zhang et al., 2020), pos-tagging used on UD but not on PTB nor CoNLL09 Chinese. In (Wang and Tu, 2020), pos-tagging is used for all datasets.

[12] We tried mutation rates 0.1, 0.2, 0.3 and the best performance is obtained on PTB dev with mutation rate 0.2.

[13] https://github.com/wangxinyu0922/Second_Order_Parsing, Note that this implementation also uses the hyper-parameters of (Zhang et al., 2020)

Method	PTB		CoNLL09	
	UAS	LAS	UAS	LAS
CRF2O*	**96.14**	**94.49**	89.63	86.52
Local2O	95.98	94.34	-	-
Local2O†	95.90	94.25	91.60	89.93
CA+hinge	95.88	94.21	91.27	89.58
GA+CA+hinge	95.93	94.26	91.63	89.89
CA+LM	95.96	94.33	91.62	89.96
3O+CA+LM	95.85	94.27	91.59	89.96
GA+CA+LM	95.95	94.34	**91.65**	**90.02**
+BERT				
Local2O	**96.91**	**95.34**	-	-
Local2O†	96.68	95.16	93.46	91.87
CA+LM	96.68	95.20	**93.48**	**91.91**
3O+CA+LM	96.65	95.13	93.47	91.87
GA+CA+LM	96.67	95.20	93.42	91.83

Table 3: Comparison on test. *: POS not used. †: Rerun with official implementation.

We can see that our third-order factors do not improve scores. With BERT probabilistic models, neither third-order nor genetic algorithm on top of coordinate ascent gives any improvement. For CoNLL09 Chinese without BERT, performance on dev are similar between settings while genetic algorithm gives an evident boost for hinge loss. With BERT as for PTB the simple model performs best.

Table 3 gives test results and comparisons with two recent similiar systems. For PTB without BERT, the exact projective parser of (Zhang et al., 2020) has the best performance, which is in accordance with the reported results in (Wang and Tu, 2020).[14] In comparison with (Wang and Tu, 2020) (Local2O), although their system has more parameters for PTB experiments[15], our coordinate ascent method with genetic algorithm plus marginalization has achieved the same performance on LAS. However, the same optimization method with hinge loss does not show good performances. For CoNLL09 Chinese without BERT, the genetic algorithm seems to help with generalization compared to simple coordinate ascent (similar score on dev but improvement on test).

With BERT, on both corpora, simple coordinate ascent gives best performance for our method.

[14] Our best single run gives 94.44 LAS on PTB which is on a par with their results.

[15] (Wang and Tu, 2020) uses a bilstm with hidden 600 while we follow (Zhang et al., 2020) to use a bilstm with hidden size 400

7.4 Results on UD

Table 1 shows LAS on UD test data. The best average performances are achieved with coordinate ascent and genetic algorithm plus linearized marginalization. For all languages, our method with or without genetic algorithm outperforms (Wang and Tu, 2020) (Local2O) except **nl** without BERT.

Method	Train	Test
Local2O	1133	706
CA	506	399
3O+CA	255	249
GA+CA	248	195

Table 4: Speed Comparison on PTB Train and Test without BERT (sentences per second)

7.5 Speed Comparison

We compare the speed of train and test with Nvidia Tesla V100 SXM2 16 Go on PTB. The result is shown in Table 4. For coordinate ascent, training is 2.2 times slower than MFVI while test is 1.8 times slower than MFVI[16].

8 Conclusion

We presented a novel method for higher-order parsing based on coordinate ascent. Our method relies on the general form of arc-polynomial score functions. Promising arcs are picked by evaluated by gradient computations. This method is agnostic to specific score functions and we showed how we can recover previously defined functions and design new ones. Experimentally we showed that, although this method returns local optima, it can obtain State-of-the-art results.

Further research could investigate whether the difference between the search space during learning and decoding is a cause of performance decrease. In particular the coordinate ascent could be replaced by a structured optimization method such as the Frank-Wolfe algorithm (see (Pedregosa et al., 2020) for a recent variant) to obtain a local optimum in a more restricted search space.

[16] The speed is measured with Eisner applied over all sentences. It is about 2 times quicker with the faster decoding strategy of (Zhang et al., 2020) which consists in applying Eisner only if the coordinate ascent solution does not return a projective arborescence.

References

D.P. Bertsekas. 1999. *Nonlinear Programming*. Athena Scientific.

Bart Decadt, Véronique Hoste, Walter Daelemans, and Antal van den Bosch. 2004. GAMBL, genetic algorithm optimization of memory-based WSD. In *Proceedings of SENSEVAL-3, the Third International Workshop on the Evaluation of Systems for the Semantic Analysis of Text*, pages 108–112, Barcelona, Spain. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Timothy Dozat and Christopher D. Manning. 2017. Deep biaffine attention for neural dependency parsing. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.

Jason Eisner. 1996. Efficient normal-form parsing for Combinatory Categorial Grammar. In *34th Annual Meeting of the Association for Computational Linguistics*, pages 79–86, Santa Cruz, California, USA. Association for Computational Linguistics.

Jason Eisner. 1997. Bilexical grammars and a cubic-time probabilistic parser. In *Proceedings of the Fifth International Workshop on Parsing Technologies*, pages 54–65, Boston/Cambridge, Massachusetts, USA. Association for Computational Linguistics.

Agnieszka Falenska and Jonas Kuhn. 2019. The (non-)utility of structural features in BiLSTM-based dependency parsers. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 117–128, Florence, Italy. Association for Computational Linguistics.

Erick Fonseca and André F. T. Martins. 2020. Revisiting higher-order dependency parsers. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8795–8800, Online. Association for Computational Linguistics.

Sida Gao and Matthew R. Gormley. 2020. Training for Gibbs sampling on conditional random fields with neural scoring factors. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4999–5011, Online. Association for Computational Linguistics.

Will Grathwohl, Kevin Swersky, Milad Hashemi, David Duvenaud, and Chris Maddison. 2021. Oops i took a gradient: Scalable sampling for discrete distributions. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 3831–3841. PMLR.

Jan Hajič, Massimiliano Ciaramita, Richard Johansson, Daisuke Kawahara, Maria Antònia Martí, Lluís Màrquez, Adam Meyers, Joakim Nivre, Sebastian Padó, Jan Štěpánek, Pavel Straňák, Mihai Surdeanu, Nianwen Xue, and Yi Zhang. 2009. The CoNLL-2009 shared task: Syntactic and semantic dependencies in multiple languages. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning (CoNLL 2009): Shared Task*, pages 1–18, Boulder, Colorado. Association for Computational Linguistics.

Eliyahu Kiperwasser and Yoav Goldberg. 2016. Simple and accurate dependency parsing using bidirectional lstm feature representations. *Transactions of the Association for Computational Linguistics*, 4:313–327.

Terry Koo and Michael Collins. 2010. Efficient third-order dependency parsers. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1–11, Uppsala, Sweden. Association for Computational Linguistics.

Terry Koo, Alexander M. Rush, Michael Collins, Tommi Jaakkola, and David Sontag. 2010. Dual decomposition for parsing with non-projective head automata. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 1288–1298, Cambridge, MA. Association for Computational Linguistics.

Marina Litvak, Mark Last, and Menahem Friedman. 2010. A new approach to improving multilingual summarization using a genetic algorithm. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 927–936, Uppsala, Sweden. Association for Computational Linguistics.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

André Martins, Miguel Almeida, and Noah A. Smith. 2013. Turning on the turbo: Fast third-order non-projective turbo parsers. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 617–622, Sofia, Bulgaria. Association for Computational Linguistics.

André Martins, Noah Smith, Eric Xing, Pedro Aguiar, and Mário Figueiredo. 2010. Turbo parsers: Dependency parsing by approximate variational inference. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 34–44, Cambridge, MA. Association for Computational Linguistics.

Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajič. 2005. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, pages 523–530, Vancouver, British Columbia, Canada. Association for Computational Linguistics.

Tomas Mikolov, Edouard Grave, Piotr Bojanowski, Christian Puhrsch, and Armand Joulin. 2018. Advances in pre-training distributed word representations. In *Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018)*.

Melanie Mitchell. 1998. *An introduction to genetic algorithms*. MIT press.

Fabian Pedregosa, Geoffrey Negiar, Armin Askari, and Martin Jaggi. 2020. Linearly convergent frank-wolfe with backtracking line-search. In *International Conference on Artificial Intelligence and Statistics*, pages 1–10. PMLR.

Hao Peng, Sam Thomson, and Noah A. Smith. 2017. Deep multitask learning for semantic dependency parsing. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2037–2048, Vancouver, Canada. Association for Computational Linguistics.

Xian Qian and Yang Liu. 2013. Branch and bound algorithm for dependency parsing with non-local features. *Transactions of the Association for Computational Linguistics*, 1:37–48.

Lothar M. Schmitt. 2001. Theory of genetic algorithms. *Theoretical Computer Science*, 259(1):1–61.

Shai Shalev-Shwartz and Tong Zhang. 2013. Stochastic dual coordinate ascent methods for regularized loss minimization. *Journal of Machine Learning Research*, 14(2).

David A. Smith and Jason Eisner. 2008. Dependency parsing by belief propagation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 145–156, Honolulu.

David A. Smith and Noah A. Smith. 2007. Probabilistic models of nonprojective dependency trees. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 132–140, Prague, Czech Republic. Association for Computational Linguistics.

Xinyu Wang and Kewei Tu. 2020. Second-order neural dependency parsing with message passing and end-to-end training. In *Proceedings of the 1st Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 10th International Joint Conference on Natural Language Processing*, pages 93–99, Suzhou, China. Association for Computational Linguistics.

Sam Wiseman and Yoon Kim. 2019. Amortized bethe free energy minimization for learning mrfs. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

Xudong Zhang, Joseph Le Roux, and Thierry Charnois. 2021. Strength in numbers: Averaging and clustering effects in mixture of experts for graph-based dependency parsing. In *Proceedings of the 17th International Conference on Parsing Technologies and the IWPT 2021 Shared Task on Parsing into Enhanced Universal Dependencies (IWPT 2021)*, pages 106–118, Online. Association for Computational Linguistics.

Yu Zhang, Zhenghua Li, and Min Zhang. 2020. Efficient second-order TreeCRF for neural dependency parsing. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 3295–3305, Online. Association for Computational Linguistics.

Yuan Zhang, Tao Lei, Regina Barzilay, Tommi Jaakkola, and Amir Globerson. 2014. Steps to excellence: Simple inference with refined scoring of dependency trees. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 197–207, Baltimore, Maryland. Association for Computational Linguistics.

A Hyper Parameters

Param	Value	Param	Value
WordEMB	100	WordEMB dropout	0.33
CharLSTM	50	CharLSTM dropout	0.00
PosEMB	100	PosEMB dropout	0.33
BERT Linear	100	BERT Linear dropout	0
BiLSTM	400	BiLSTM dropout	0.33
MLP_{arc}	500	LSTM_{arc} dropout	0.33
$\text{MLP}_{\text{label}}$	100	$\text{LSTM}_{\text{label}}$ dropout	0.33
$\text{MLP}_{\text{sib,gp,3O}}$	100	MLP_{arc} dropout	0.33
Learning Rate	$\mathbf{2e^{-4}}$	β_1, β_2	0.90
Annealing	$0.75^{\frac{t}{5000}}$	Patience	100

Table 5: Hyper-parameters

Remark that when running experiments with UD, the WordEMB is reset to 300 because we use 300 dimension fasttext embedding (Mikolov et al., 2018) following (Zhang et al., 2020; Wang and Tu, 2020).

B Complete derivations

B.1 Partial Derivatives

We start with the definition:

$$\frac{\partial S(y)}{\partial y_a} = \sum_{k=1}^{K} \sum_{F \in (\mathcal{F}_k(C) \cap \mathcal{R})} s_F \frac{\partial \prod_{a' \in F} y_{a'}}{\partial y_a}$$

case $\mathbf{a \notin F}$: we can see that if $a \notin F$, then $\frac{\partial \prod_{a' \in F} y_{a'}}{\partial y_a} = 0$ since the expression in the numerator does not contain variable y_a.

case $\mathbf{a \in F}$: Now suppose that $a \in F$. Remark that F is a factor from $\mathcal{F}_k(C)$, and thus is a proper subset of arcs and consequently all arcs in F are different. By applying the rule for product derivatives we can rewrite the partial as:

$$\frac{\partial \prod_{a' \in F} y_{a'}}{\partial y_a} = \prod_{a' \in F \backslash a} y_{a'}$$

Suppose that F is a factor of k arcs from $\mathcal{F}_k(C)$ that contains a, and that the previous equation equals 1, we have:

$$\begin{aligned} \prod_{a' \in F \backslash a} y_{a'} = 1 &\iff y_{a'} = 1, \forall a' \in F \backslash a \\ &\iff a' \in y, \forall a' \in F \backslash a \\ &\iff F \backslash a \in \mathcal{F}_{k-1}(y) \end{aligned}$$

Conclusion: By plugging this into the definition we have:

$$\frac{\partial S(y)}{\partial y_a} = \sum_{k=1}^{K} \sum_{\substack{F \in (\mathcal{F}_k(C) \cap \mathcal{R}), \\ a \in F}} s_F \mathbf{1}[F \backslash a \in \mathcal{F}_{k-1}(y)]$$

B.2 Substitution Scores 1

We start from equation (1):

$$S(y) = \sum_{k=1}^{K} \sum_{F \in (\mathcal{F}_k(C) \cap \mathcal{R})} s_F \prod_{(h',d') \in F}^{k} y_{h',d'}$$

Similarly, given arc $(h, d) \in y$ we have:

$$S(y \backslash (h,d)) = \sum_{k=1}^{K} \sum_{\substack{F \in (\mathcal{F}_k(C) \cap \mathcal{R}) \\ (h,d) \notin F}} s_F \prod_{(h',d') \in F}^{k} y_{h',d'}$$

The score difference is:

$$\begin{aligned} &S(y) - S(y \backslash (h,d)) \\ &= \sum_{k=1}^{K} \sum_{\substack{F \in (\mathcal{F}_k(C) \cap \mathcal{R}) \\ (h,d) \in F}} s_F \prod_{(h',d') \in F}^{k} y_{h',d'} \\ &= \sum_{k=1}^{K} \sum_{\substack{F \in (\mathcal{F}_k(C) \cap \mathcal{R}) \\ (h,d) \in F}} s_F \mathbf{1}[F \in \mathcal{F}_k(y)] \\ &= \sum_{k=1}^{K} \sum_{\substack{F \in (\mathcal{F}_k(C) \cap \mathcal{R}) \\ (h,d) \in F}} s_F \mathbf{1}[F \backslash (h,d) \in \mathcal{F}_{k-1}(y)] \end{aligned}$$

where the last line is correct since we assume we already have $(h, d) \in y$.

By using equation (3), we have directly:

$$S(y) - S(y \backslash (h,d)) = \frac{\partial S(y)}{\partial y_{h,d}}$$

which is

$$S(y) = \frac{\partial S(y)}{\partial y_{h,d}} + S(y \backslash (h,d))$$

B.3 Substitution Scores 2

First, note that the set of arc $y \backslash (h, d)$ is the same as $y[h \to h', d] \backslash (h'd)$. This is because $y[h \to h', d]$ is constructed by substituting arc $(h, d) \in y$ with arc (h', d). The other arcs are unchanged. Thus we have:

$$S(y[h \to h', d] \backslash (h', d)) = S(y \backslash (h, d))$$

Secondly, consider the condition:

$$(h', d) \in F, F \backslash (h', d) \in \mathcal{F}_{k-1}(y[h \to h', d])$$

Remark that $F = \{(h_1, d_1), (h_2, d_2), ..., (h_k, d_k)\}$ being a proper subset of arcs is required to satisfy:

$\forall i \neq j$, $d_i \neq d_j$. Thus $F \backslash (h', d)$ has no arc for column d. In this case, we can write the previous condition as:

$$(h', d) \in F, F \backslash (h', d) \in \mathcal{F}_{k-1}(y)$$

since y and $y[h \rightarrow h', d]$ only differ in column d.

By using equation (3), we have:

$$\begin{aligned}
&\frac{\partial S(y[h \rightarrow h', d])}{\partial y_{h',d}} \\
&= \sum_{k=1}^{K} \sum_{\substack{F \in (\mathcal{F}_k(C) \cap \mathcal{R}), \\ (h',d) \in F}} \\
&\quad s_F \mathbf{1}[F \backslash (h', d) \in \mathcal{F}_{k-1}(y[h \rightarrow h', d]) \\
&= \sum_{k=1}^{K} \sum_{\substack{F \in (\mathcal{F}_k(C) \cap \mathcal{R}), \\ (h',d) \in F}} s_F \mathbf{1}[F \backslash (h', d) \in \mathcal{F}_{k-1}(y) \\
&= \frac{\partial S(y)}{\partial y_{h',d}}
\end{aligned}$$

To conclude, we have:

$$\begin{aligned}
&S(y[h \rightarrow h', d]) \\
&= \frac{\partial S(y[h \rightarrow h', d])}{\partial y_{h',d}} + S(y[h \rightarrow h', d] \backslash (h', d)) \\
&= \frac{\partial S(y)}{\partial y_{h',d}} + S(y \backslash (h', d))
\end{aligned}$$

The first equation is a direct usage of equation (4) and the second equation comes from the previous proof.

B.4 First-order Linearization

We want to compute for all word positions d the highest scoring head:

$$\begin{aligned}
&\operatorname*{argmax}_{h'} S(y[h \rightarrow h', d]) \\
&\approx \operatorname*{argmax}_{h'} S(y) + (y[h \rightarrow h', d] - y)^\top \nabla S(y) \\
&= \operatorname*{argmax}_{h'} S(y) + \frac{\partial S(y)}{y_{h'd}} - \frac{\partial S(y)}{y_{hd}} \\
&= \operatorname*{argmax}_{h'} \frac{\partial S(y)}{y_{h'd}}
\end{aligned}$$

We go from first to second line by first-order Taylor approximation. Transition from second to third line is based on the fact that $y[h \rightarrow h', d]$ differs from y by only two arcs, the addition of (h', d) and the removal of (h, d) so the inner product can be expressed as a difference of two partial derivatives. We go from third to fourth line by noticing that only one term depends on h' hence we can simplify the argmax.

B.5 Approximate Marginal Estimation

$\hat{y}$ is the highest-scoring parse and contains arc (g, d). We write $s_{hd} = \frac{\partial S(\hat{y})}{\partial y_{hd}}$ for all arc (h, d). We recall from previous section that first-order Taylor approximation gives: $S(y[g \rightarrow h, d]) \approx S(\hat{y}) + s_{hd} - s_{gd}$.

$$\begin{aligned}
p\big((h, d)|x^*\big) &= \frac{p(\hat{y}[g \rightarrow h, d])}{\sum_{h'} p(\hat{y}[g \rightarrow h', d])} \\
&= \frac{Z^{-1} \exp(S(\hat{y}[g \rightarrow h, d]))}{\sum_{h'} Z^{-1} \exp(S(\hat{y}[g \rightarrow h', d]))} \\
&= \frac{\exp(S(\hat{y}[g \rightarrow h, d]))}{\sum_{h'} \exp(S(\hat{y}[g \rightarrow h', d]))} \\
&\approx \frac{\exp(S(\hat{y}) + s_{hd} - s_{gd})}{\sum_{h'} \exp(S(\hat{y}) + s_{h'd} - s_{gd})} \\
&= \frac{\exp(S(\hat{y}) - s_{gd}) \exp(s_{hd})}{\exp(S(\hat{y}) - s_{gd}) \sum_{h'} \exp(s_{h'd})} \\
&= \frac{\exp(s_{hd})}{\sum_{h'} \exp(s_{h'd})}
\end{aligned}$$

C Tensor Factorization for Third-Order Models

For a third order model, a tensor $W \in \mathbb{R}^{n^6}$ should be used to calculate the score of $F = \{(h_1, d_1), (h_2, d_2), (h_3, d_3)\}$:

$$s_F = v_{h_3}^T v_{h_2}^T v_{h_1}^T W v_{d_1} v_{d_2} v_{d_3}$$

with v_{h_i}, v_{d_i} the feature vector of head and modifier words.

To reduce the memory cost, we simulate the previous calculation with three tensors of biaffine and one tensor of triaffine. The score can be calculated as:

$$\begin{aligned}
l_1 &= v_{h_1} \circ W_{biaffine}^{(1)} v_{d_1} \\
l_2 &= v_{h_2} \circ W_{biaffine}^{(2)} v_{d_2} \\
l_3 &= v_{h_3} \circ W_{biaffine}^{(3)} v_{d_3} \\
s_F &= l_3^T l_2^T W_{triaffine} l_1
\end{aligned}$$

with $W^i_{biaffine} \in \mathbb{R}^{n^2}$ the tensor of biaffine and $W_{triaffine} \in \mathbb{R}^{n^3}$ the tensor of triaffine, $\circ$ represents the Hadamard product (element-wise product of vector).