The Interplay Between Loss Functions and Structural Constraints in Dependency Parsing

Robin Kurtz, Marco Kuhlmann

Linköping University
Department of Computer and Information Science
robin.kurtz@liu.se, marco.kuhlmann@liu.se

September 16, 2019

Abstract

Dependency parsing can be cast as a combinatorial optimization problem with the objective to find the highest-scoring graph, where edge scores are learnt from data. Several of the decoding algorithms that have been applied to this task employ structural restrictions on candidate solutions, such as the restriction to projective dependency trees in syntactic parsing, or the restriction to noncrossing graphs in semantic parsing. In this paper we study the interplay between structural restrictions and a common loss function in neural dependency parsing, the structural hinge loss. We show how structural constraints can make networks trained under this loss function diverge and propose a modified loss function that solves this problem. Our experimental evaluation shows that the modified loss function can yield improved parsing accuracy, compared to the unmodified baseline.

1 Introduction

Dependency parsing is the task of mapping a sentence into a formal representation of its syntactic or semantic structure in the form of a bilexical directed graph, where arcs are drawn between pairs of words. Syntactic dependency graphs are commonly restricted to tree structures, whereas semantic dependency graphs can take the form of general acyclic graphs.

There are two main approaches to dependency parsing: transition-based and graph-based parsing (Kübler et al., 2009). Transition-based dependency parsers learn to map the input sentence to a sequence of actions for a shift-reduce-type automaton that constructs the output graph. Graph-based parsers cast dependency parsing as a combinatorial optimization problem with the objective to find a highest-scoring graph in a set of candidate graphs for the input sentence. The scores of the transitions and the candidate graphs are learnt from training data with the help of some machine learning algorithm. While there are some differences with regard to efficiency, both transition-based and graph-based parsing have been shown to perform similarly well.

As it comes to the specific machine learning algorithms used in dependency parsing, recent years have seen a shift from traditional feature-based approaches to neural networks (NNs). Even before the global shift to NNs, Mayberry III and Miikkulainen (1999) experimented using NNs in transition-based parsing. Others used neural network structures that are far less common in current natural language processing (NLP) applications, such as Incremental Sigmoid Belief Networks (Titov and Henderson, 2007), Temporal Restricted Boltzmann Machines (Garg and Henderson, 2011), and Simple Synchrony Networks (Henderson, 2004). Following up on the parser of Stenetorp (2013), who used recursive neural networks, Chen and Manning (2014) implemented a state-of-the-art architecture for transition-based syntactic parsing using feedforward NNs and distributed word representations. The approach of Chen and Manning (2014) uses feature templates made up of word, label and part-of-speech tag embeddings, similar to perceptron-style parsers, but uses distributed embeddings instead of one-hot encodings.

Kiperwasser and Goldberg (2016) used recurrent neural networks (RNNs) to encode contextual information into the network for both transition-based and graph-based dependency parsing. Instead of having to select which contextual features would be relevant for the parser, the RNN creates contextual embeddings for every token of the input sequence. Their network structure has been adopted and extended, and produced state-of-the-art results, not only for syntactic dependency parsing (Dozat and Manning, 2017) but also for semantic dependency parsing (Dozat and Manning, 2018). One difference between the parser of Kiperwasser and Goldberg (2016) and Dozat and Manning (2017) is in the loss function that is used to provide a signal for how far off the predictions of the network are relative to the gold-standard prediction, and which parameters should be updated to move it closer to that target value. More specifically, Dozat and Manning (2017) use a cross-entropy objective, whereas Kiperwasser and Goldberg (2016) use a hinge loss which is computed on the complete predicted output graph.

Several of the parsing algorithms in the graph-based paradigm restrict the search space to a subset of all possible dependency graphs for the input sentence by imposing structural constraints. Perhaps the most well-known such constraints are projectivity in syntactic parsing (Eisner and Satta, 1999), the closely related noncrossing constraint in semantic parsing (Schluter, 2014; Kuhlmann and Jonsson, 2015), and the one-endpoint-crossing property, which has been imposed on both trees (Pitler et al., 2013) and graphs (Kummerfeld and Klein, 2017; Cao et al., 2017; Kurtz and Kuhlmann, 2017). Regarding unrestricted algorithms for finding highest-scoring candidate graphs, it is worth mentioning that this can be done in polynomial time only for trees (Chu and Liu, 1965; Edmonds, 1967); the same problem for unrestricted directed acyclic graphs is intractable (Schluter, 2014).

It is intuitively clear that, when choosing a parsing algorithm with a restricted search space, one would like the structural constraint to admit most (if not all) dependency graphs in the data. In this paper we study the specific effect that structural constraints have on training a neural dependency parsers using hinge loss. More specifically, we show, by theoretical analysis and experimentation, that both the noncrossing constraint in semantic parsing and the projectivity constraint in syntactic parsing can make the hinge loss unbounded, leading to divergence during training and an underperforming model. We also show how to repair the hinge loss function to play along with structural constraints, and verify empirically that the modified loss function can lead to improved

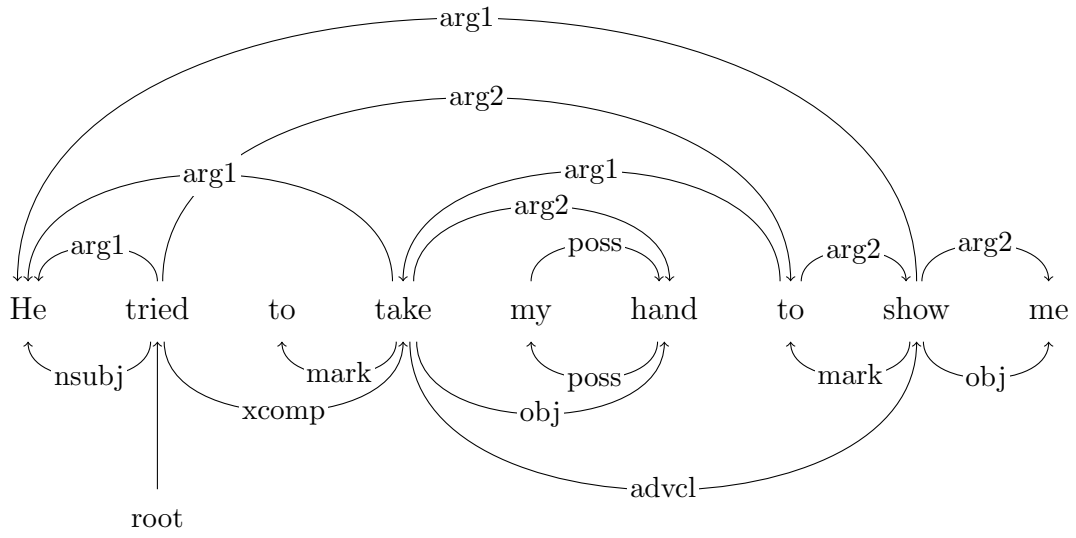


Figure 1: A sample dependency graph from the SDP dataset (Flickinger et al., 2016, DM #41526060) drawn above and the corresponding dependency tree with Stanford Basic (SB) dependencies drawn below. Note that this graph violates the noncrossing constraint, as the arcs *take* → *He* and *tried* → *to* cross each other.

parsing accuracy, compared to the unmodified baseline. Our experiments also indicate that our modified hinge loss has a regularizing effect when parsing to dependency trees with an unrestricted algorithm.

Structure of the paper The following Section 2 introduces the necessary background for dependency parsing using neural networks. Section 3 explains the problem of restricting the search space when training with the structural hinge loss and proposes our solution in the form of a modified hinge loss function. The effect of the proposed solution is tested empirically in Section 4. Section 5 features an overall discussion of the findings, followed by a summary and our conclusions in Section 6.

2 Background

We introduce a mathematical description of dependency graphs and parsing algorithms for both tree and graph parsing, followed up by an overview of the neural network structures used in the parser, and how they are trained.

2.1 Graphs and Parsing

Given a natural language sentence $x = x_1, x_2, \ldots, x_n$, we define a *dependency graph* for x as an arc-labelled directed acyclic graph whose vertices correspond, one-to-one, to the words x_i. Placing the vertices of a dependency graph on a line in the plane following the left-to-right ordering of the sentence, we draw the arcs as semi-circles in the half-plane

above the line, using arrows to denote the direction from head to dependent. An arc from word x_i to word x_j is denoted as $i \rightarrow j$. Each arc of the dependency graph has a label taken from a finite set, describing the relation of the head and the dependent vertices. A *dependency tree* is a dependency graph fulfilling the tree constraint: it is connected, and each node, except for the root node, has exactly one incoming arc. Examples of syntactic and semantic dependency graphs for the sentence *He tried to take my hand to show me* are shown in Figure 1. The agent for the three verbs *tried, take* and *show* is *He*, which is directly encoded in the semantic dependency graph, whereas this relation is not easily visible in the syntactic dependency tree. This ability to assign one and the same actor to multiple predicates and other degrees of freedom, allow expressing more meaning related phenomena and thus motivate the extension from trees to graphs.

2.1.1 Parsing as Combinatorial Optimization

Dependency graph parsing for both syntactic and semantic dependencies can be cast as *maximum subgraph parsing*, generalizing the approach of McDonald et al. (2005). The predicted graph $\hat{y}$ for the sentence x is chosen as the graph y from a set $Y(x)$ of candidate graphs which maximizes the scoring function S:

$$\hat{y} = \arg\max_{y \in Y(x)} S(x, y) \tag{1}$$

The scoring function S is computed via a sum of scores for local substructures $s(x, a)$, which in our case are single arcs $a \in y$. This is known as the *arc-factored model*.

$$\hat{y} = \arg\max_{y \in Y(x)} \sum_{a \in y} s(x, a) \tag{2}$$

In order to simplify the equations, we drop the parameter x thus making the dependence of the scoring function and the input sentence implicit. The set of candidate graphs $Y(x)$ is defined to match the needs of the type of graphs that are parsed (e.g. trees when parsing syntactic dependency graphs) and can be constrained even further to tighten the search space. The maximization of Equation (2) is solved using some decoding algorithm which, given a matrix of scores for each possible arc, returns the highest scoring subgraph.

2.1.2 Parsing Algorithms

We use four different parsing algorithms for our experiments, two of which are for trees, and two for graphs, where for each pair one restricts the search space, while the other does not. All of those approaches consider only unlabelled arcs. We view labelling as a separate procedure that is done on top of the predicted unlabelled graph, classifying each arc label independently.

For trees, we use the algorithm by Eisner and Satta (1999), which restricts outputs to be *projective*. Informally, a projective tree is one whose arcs – including an infinitely long arc to the root node – can be drawn in the half-plane above or below the sentence without crossings. The tree drawn in the lower half of Figure 1 is projective. More formally, a single arc $i \rightarrow j$ is projective if every vertex k lying between i and j ($i < k < j$ or $j < k < i$) has a path from i, and a tree is projective if all of its arcs are projective. For semantic

dependency graphs, we use the algorithm of Kuhlmann and Jonsson (2015), which is restricted to *noncrossing* acyclic graphs. Both algorithms can be understood in terms of deduction systems, representing certain sub-structures as *items* that are deduced using a set of *rules*. These rules take already created items or initial *axioms* as input, trying to create a final *goal item*. A *derivation* is the chain of rules used to create the goal item. In order to discern between the possibly many derivations for the goal item, we use weighted rules resulting in weighted derivations. The highest-scoring such derivation then corresponds to the highest-scoring dependency graph.

For most datasets, both structural constraints are not complete, meaning that there are graphs in the training data that are not *projective* or not *noncrossing*. We compare these non-complete decoding algorithms with two algorithms covering their respective datasets completely: for syntactic parsing, we use the Chu-Liu-Edmonds (CLE) (Chu and Liu, 1965; Edmonds, 1967) algorithm for finding the highest scoring dependency tree. For semantic parsing, we use a mostly unrestricted algorithm for graphs; this algorithm adds every arc with a positive weight, only ignoring loops (no $i \rightarrow i$) and allowing each pair of endpoints to only have one connecting arc (i.e. if there is $i \rightarrow j$ then there is no $j \rightarrow i$). The graphs returned by the unrestricted algorithm are thus possibly cyclic, even though the data is restricted to only acyclic graphs.

2.1.3 Types of Prediction Errors

Considering systems that predict dependency graphs, we not only differentiate between correct and incorrect predictions, but also whether to add or not to add an arc to the dependency graph. Given a sentence x, a gold graph y and a predicted graph $\hat{y}$, we let $A(x)$ be the set of all possible arcs on x (i.e. the complete graph), with $y \subseteq A(x)$ and $\hat{y} \subseteq A(x)$. We distinguish between four categories:

- *True positives*: correctly predicted arcs

$$y \cap \hat{y} = \{a \in A(x) \mid a \in y \wedge a \in haty\}$$

- *True negatives*: arcs neither in the predicted nor in the gold graph

$$A(x) \setminus (y \cup \hat{y}) = \{a \in A(x) \mid a \notin y \wedge a \notin \hat{y}\}$$

- *False positives*: predicted arcs that are not in the gold graph

$$\hat{y} \setminus y = \{a \in A(x) \mid a \in \hat{y} \wedge a \notin y\}$$

- *False negatives*: arcs that were missed in the prediction

$$y \setminus \hat{y} = \{a \in A(x) \mid a \in y \wedge a \notin \hat{y}\}$$

Using this terminology we can reason on the type of mistakes made and compute the F1-score (the harmonic mean of precision and recall) or regular accuracy for evaluating different parsers.

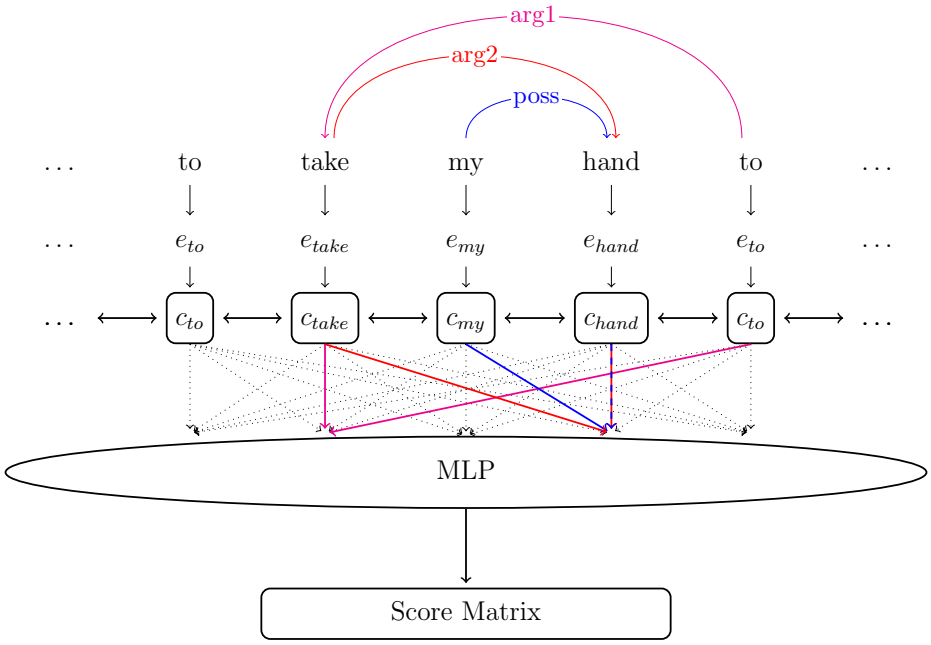


Figure 2: The basic arc-scoring architecture, applied to a partial phrase of Figure 1. The embedded words e_i are processed by a bidirectional RNN, to produce contextual embeddings c_i. Each pair of these embeddings is processed by an MLP to compute a score for each possible arc. The connections are coloured to match the arcs of the graph above.

2.2 Neural Networks for Parsing

In order for the decoding algorithm to find the highest scoring graph, we need to score each arc using a learning component. The network architecture described in the following is taken from Kiperwasser and Goldberg (2016) and visualized in Figure 2. In order for a neural network to read the tokens x_i of a sentence x, we need to embed them as vectors e_i. These are then processed by the base of the neural network, a bidirectional RNN (BiRNN), that processes the word vectors e_i of sentence x both left-to-right and right-to-left. Concatenating the outputs from the forward and backward RNNs, we get a context-dependent representation c_i for each token x_i. Each possible ordered pair of these representations is then further processed by a feed-forward neural network (or multi-layer perceptron MLP), to compute scores for each possible arc, creating a score matrix. Labels are computed by another feed-forward network using the same inputs, predicting scores for each label. The complete parser thus first applies the neural network on the input sentence, uses the scores for each possible arc as input to predict the highest scoring unlabelled graph, and then uses the labelling network to predict the arcs' labels.

The network is trained with the objective to minimize a loss function. A loss function is generally designed to capture how far off the network's predicted output is from the target output. We use the structural hinge loss objective and the binary cross-entropy objective in order to train our networks.

Binary Cross-Entropy The *binary cross-entropy loss* is defined for each possible arc, regardless of whether the arc was part of the output graph or not. We compute this cross-entropy loss L for a sentence x with gold graph y and predicted graph $\hat{y}$:

$$L(x, \hat{y}, y) = \sum_{a \in A(x)} \mathbf{1}_y(a) \cdot -\log(\sigma(s(a))) + (1 - \mathbf{1}_y(a)) \cdot -\log(1 - \sigma(s(a))) \tag{3}$$

The equation uses the sigmoid function $\sigma(x) = (1 + e^{-x})^{-1}$, the indicator function $\mathbf{1}_y(a)$ that returns 1 if the arc a is in gold graph y and zero otherwise, and finally the score output from the neural network $s(a)$ that is computed for every possible arc $A(x)$ on sentence x. True positives with high positive weights will receive a loss close to zero, as well as true negatives with high negative weights. False positives are punished via the second term as $\sigma(s(a))$ will be close to 1, and thus $-\log(1 - \sigma(s(a)))$ grows large. For false negatives the first term works analogously. The cross-entropy loss will thus decrease towards zero as weights are adjusted to fit the data.

Structural Hinge Loss In contrast to binary cross-entropy, the *structural hinge loss* only takes arcs of the predicted and gold graph into account. It is computed as the difference between the sum of scores of arcs in the predicted and gold graph. Comparing the predicted and intended structure directly, means that we have to run a decoding algorithm to choose the highest-scoring subgraph before we are able to compute the loss.

$$L(x, \hat{y}, y) = \sum_{a \in \hat{y}} s(a) - \sum_{a \in y} s(a) \tag{4}$$

True positives are present in both sums and cancel each other out, reducing the loss to the difference of the sum of false positives and the sum of false negatives:

$$L(x, \hat{y}, y) = \sum_{a \in \hat{y} \setminus y} s(a) - \sum_{a \in y \setminus \hat{y}} s(a) \tag{5}$$

In order to increase the distance of scores for correct and incorrect predictions, a margin is often applied that penalizes correct predictions, if their scores are too close to those of incorrect predictions. We use a margin for our experiments but will ignore it for now to simplify the presentation. Considering the simple unrestricted decoding algorithm that only adds arcs with positive weights, we can assume that all scores in the first sum are positive, whereas all scores in the second sum are negative or less than the score of a competing arc in the opposite direction present in the first sum. When parsing trees with the CLE algorithm, the parser has to choose arcs in order to meet the tree constraint, even if the scores are negative. This means that both sums can be negative, but the second sum will be less than the first sum, as the prediction is based on the maximal scoring arcs. The difference between both will therefore be positive.

Similarly to the cross-entropy loss, the hinge loss will also decrease towards zero when fitting the weights to the data, as the number of false positives and false negatives will ideally become zero, making both sums empty. A major difference between both losses is that the hinge loss does not further consider true positives and true negatives, whereas the cross entropy loss instead uses an increasingly small loss even for those.

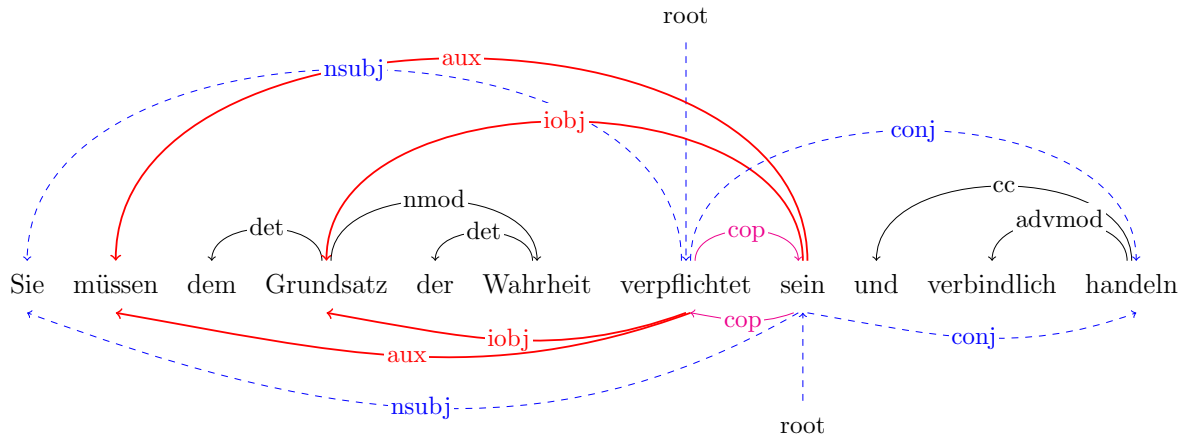


Figure 3: A sample dependency tree from the UD dataset (Nivre et al., 2017, German-GSD #train-s7963). Note that this tree violates the projectivity constraint, as the arcs coloured red cross three other arcs (blue and dashed). Alternative arcs that adhere to the constraint are drawn below.

Label Loss Our system deals with the decision which arcs to add to the graph and how to label them independently, as done in the system by Kiperwasser and Goldberg (2016). The labeller is an MLP similar to the arc scorer, which is trained on gold arcs and only applied to predicted arcs during testing. The loss for the labels is computed independently of the predictions of the unlabelled arcs, and finally added to the loss of arcs.

Optimization Using the loss function as a signal of falseness, we adjust the parameters of the network in the opposite direction of the loss' gradient. The goal is thus to increase the scores of false negatives and to decrease the scores of false positives. This is done using the stochastic gradient descent (SGD) algorithm or some more advanced alternative and backpropagation to efficiently update all activated parameters.

3 Interplay Between Restriction and Loss

As stated previously, the structural hinge loss will be zero given that there are neither false positive nor false negative predictions. For decoding algorithms that completely cover the training data, such as the CLE algorithm for trees and the unrestricted algorithm for graphs, this will be true as soon as the weights are tuned correctly. Using a decoding algorithm that restricts the search space, however, complicates the simple assumption that only the weights computed by the network have an influence on false positives and false negatives.

To illustrate the problem, consider a depedency graph as in Figure 1 that contains crossing arcs. For each pair of crossing arcs a_1, a_2, the noncrossing decoder will have to choose which arc (e.g. a_1) to include in the output graph as the gold graph y is not in the set of candidate graphs ($y \notin Y(x)$). The loss is thus not bounded at zero because we possibly have false negatives (here a_2) with positive weights. The optimizer will try to further minimize the loss, effectively increasing the false negative arc's (a_2) weight beyond

what would be needed if no constraint was applied. With this arc's (a_2) weight increased, the parser is more likely to choose it over its crossing counterpart (a_1) in the future. In order to fix the mistake of leaving out the formerly chosen arc (a_1), its weight will be increased as it has now become a false negative. This will lead to the loss decreasing further and further, sending a misleading update signal to the network's parameters and decreasing the loss indefinitely.

In order to fix the problem with the unbounded loss it seems reasonable to introduce a lower bound at zero:

$$L(x, \hat{y}, y) = \max\Big(0, \sum_{a \in \hat{y} \setminus y} s(a) - \sum_{a \in y \setminus \hat{y}} s(a)\Big) \tag{6}$$

This can however stop the network from learning from some of its mistakes. There might for example be a false negative with positive weight that was not set due to the structural constraint and an unrelated false positive with a smaller weight. This negative loss will thus be cut off at zero, indicating that there is nothing wrong and nothing to update.

Bounding functions like max and min have their gradients defined to be zero if the bound applies, and one otherwise. This means that if the bound is applied, the zero gradient of the bound function propagates downwards the computation graph to every parameter below. With all these gradients being zero, none of these parameters will be updated, effectively stopping learning.

Applying an upper bound to every false negative arc instead, we can control which arcs should have their parameters updated. Under the assumption that false negatives whose scores exceed the bound are not part of the predicted graph due to the structural constraint, we stop updates being made for only those false negatives, still allowing the updates to be applied for other false negatives and also false positives.

This however is only valid in the semantic parsing scenario, due to the structurally more liberal nature of graphs , which allows the parser to ignore all arcs that have a negative weight. Trees on the other hand follow a rigid structure, which includes one incoming arc for every node. If there is a false negative caused by the structural constraint, then there also is a false positive, needed to fulfil the tree requirement. A false negative with sufficiently high score will no longer be updated using the minimum, but a structurally forced false positive that might already have a sufficiently negative score (i.e. it would be ignored if considered for itself) will have its parameters adjusted even further. We therefore apply the same strategy to false positives as for false negatives. We allow a minimum score of m for false positives and a maximum score of n for false negatives.

$$L(x, \hat{y}, y) = \sum_{a \in \hat{y} \setminus y} \max\big(m, s(a)\big) - \sum_{a \in y \setminus \hat{y}} \min\big(n, s(a)\big) \tag{7}$$

While this does not stop the loss function from becoming negative, which seems unwanted for a loss function, we have created a lower bound we can optimize towards. For graphs this minimum is the number of *structurally impossible* false negatives times n, added with the number of *structurally enforced* false positives times m for trees.

Figure 3 visualizes the problems of using an overly strict constraint. The arcs marked in red cross the arcs marked in blue (and dashed). Assuming that their score is less than the blue arcs' scores, the parser needs to find a projective alternative as below. Their scores will be increased, exceeding the blue arcs' scores when parsing the sentence another time. Keeping the red arcs and projectivizing the blue arcs instead also forces the magenta arc *verpflichtet* → *sein* to change direction in order to keep the tree constraint, leading to four new false positives, forced by the preference to draw the red arcs. While the new loss function does not stop the parser to choose incorrect arcs, even if their scores are sufficiently high, it stops the updates on "false" false positives and "false" false negatives.

4 Experiments and Results

This section demonstrates the behaviour of the structural hinge loss in its original, naively fixed and fixed form, when paired with a decoder that puts an overly strict structural constraint on the graphs compared with a decoder that does not. With these experiments we aim to show that

(i) the hinge loss will perform suboptimally and decrease towards infinity

(ii) forcing a lower bound at zero hinders learning

(iii) there should be no negative influence of the adjusted hinge loss when no structural restriction is used.

After introducing the network structure we report our results for semantic dependency graph parsing (SDP) followed by a short discussion, in turn followed by our results and a short discussion for syntactic dependency tree parsing.

4.1 Network Parameters and Setup

We not only adopt the neural network structure for graph-based parsing from Kiperwasser and Goldberg (2016) but also their implementation, adjusting it to a current version of Dynet (Neubig et al., 2017) and Python 3. Instead of using regular long short-term memory networks (LSTM, Hochreiter and Schmidhuber (1997)), we use a variant with coupled input and forget gates, which results in fewer parameters without losing out on performance (Greff et al., 2015). We follow standard practice and use gold part-of-speech (POS) tags. In each experiment we train models using the Adam optimizer (Kingma and Ba, 2014) for 20 epochs using hyperparameters as in Table 1 and DyNet's default parameters if not stated otherwise. Out of these 20 models we finally choose the model that performs best on the development set. We extend the structural hinge loss with a margin, based on the weighted Hamming distance with parameters p an r that introduce a margin for false positives and false negatives respectively:

$$c(y, \hat{y}) = \sum_{a \in \hat{y} \setminus y} p + \sum_{a \in y \setminus \hat{y}} r \tag{8}$$

Table 1: Hyperparameter values for the network and training.

Word embedding	100
POS tag embedding	25
hidden units in arc-MLP	100
hidden units in label-MLP	100
BiLSTM Layers	2
BiLSTM dimensions (hidden/output)	125/125
word dropout	0.25

In contrast to a fixed margin between the score of the highest scoring incorrect and the gold graph, this margin is able to scale with the number of arcs (Taskar et al., 2003).

$$L(x, y) = \max_{\hat{y} \in Y(x)} \Big(S(x, \hat{y}) + c(y, \hat{y}) \Big) - S(x, y) \tag{9}$$

The maximization is coined *loss-augmented inference* (Taskar et al., 2005) and is easily solved by adjusting the score matrix according to the Hamming distance, when predicting the highest scoring graph.

We follow Peng et al. (2017) and Martins and Almeida (2014) setting the parameters of the weighted Hamming distance to $p = 0.4$ and $r = 0.6$. Note that his favours recall over precision, by increasing the weights of arcs not in the gold graph by p and decreasing the weights of arcs in the gold graph by r.

For the constraint-aware structural hinge loss, we set the parameters to fit the Hamming-augmented loss, that is, we set the minimum for false positives to $m = -p$ and the maximum for false negatives to $n = r$. This ensures that arcs that are not within the margin, but are still false positives or negatives, do not propagate an update signal.

$$\begin{aligned} L(x, \hat{y}, y) = &\sum_{a \in \hat{y} \setminus y} \max\big(m, s(a)\big) - \sum_{a \in y \setminus \hat{y}} \min\big(n, s(a)\big) \\ &+ \sum_{a \in \hat{y} \setminus y} p \qquad\qquad + \sum_{a \in y \setminus \hat{y}} r \end{aligned} \tag{10}$$

While this rather small model certainly does not produce state-of-the-art results, it is sufficient to show the interplay between a structural constraint and the structural hinge loss, using a system that is the basis for many current parsing systems.

Moss et al. (2019) recently showed that in order to reliably identify significant performance changes, one needs to both vary initial random seeds and train-test splits. The trends showcasing the interplay between loss function and restriction are clearly visible throughout nearly all experiments. In addition to the obvious differences we test for statistical sigfinicance to give an intuition on less obvious differences. We compute p-values following Berg-Kirkpatrick et al. (2012) drawing 10^6 bootstrap samples from the test sets and set the significance α to 0.01.

Table 2: Coverage in terms of complete graphs (G) and individual arcs (A) for noncrossing graphs.

	DM	PAS	PSD
G	69.29	59.85	65.04
A	97.63	97.24	96.01

4.2 Semantic Dependency Parsing

The SDP graphs from Flickinger et al. (2016) come in four different *flavours*, wherefrom we consider three: DM graphs derived from DeepBank (Oepen and Lønning, 2006; Ivanova et al., 2012), the predicate–argument structures computed by the Enju parser (PAS, Miyao (2006)), and the tectogrammatical layer of the Prague Dependency Treebank (PSD, Hajic et al. (2012)). All three sets are built on top of the Penn Treebank (PTB (Marcus et al., 1993)), which uses Wall Street Journal texts, and parts of the Brown corpus (Francis and Kučera, 1985) for out-of-domain data. The three different SDP flavours follow different designs, making them structurally different from each other and more or less difficult to parse. In order to predict the impact a structurally constrained decoder will have, we report the coverage of the noncrossing constraint on the complete training, development and test sets in Table 2.

4.2.1 Results

Table 3 reports the results on the three SDP datasets. We compare the results for systems trained with the unmodified structural hinge loss (hinge Eq. (5)), the hinge loss with minimum at zero (hinge^0 Eq. (6)), the constraint aware hinge loss (hinge' Eq. (7)), and the binary cross entropy loss (bce Eq. (3)). The decoders used are the noncrossing directed acyclic graph decoder (ncdag) of Kuhlmann and Jonsson (2015), and our basically unrestricted decoder. For each system we evaluate labelled F1-score on both the in-domain (id) and out-of-domain (ood) datasets.

Figures 4 and 5 plot the development of the loss on the training data, and the mean and standard deviation for arc scores over each of the 20 epochs. Plotting the loss shows whether it decreases, how far it decreases and whether it decreases to a minimum. Plotting the mean and the standard deviation of the score matrices shows whether the scores stabilize and how extremely they deviate from their centre.

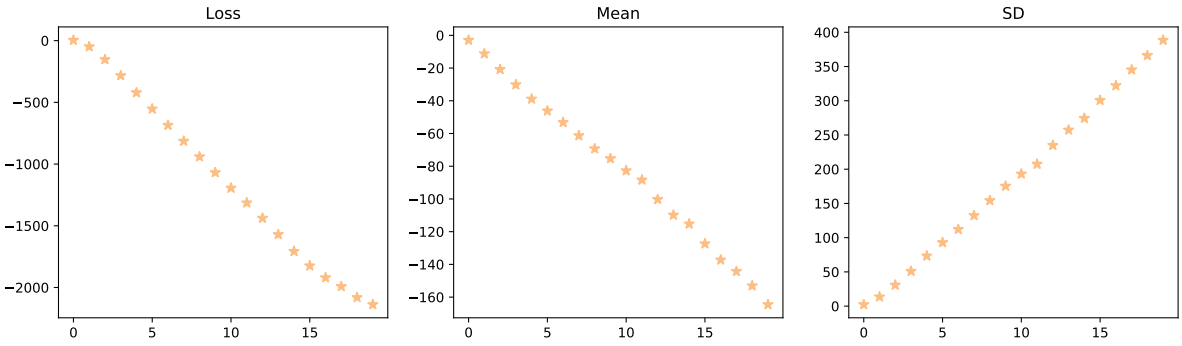
4.2.2 Discussion

For all three datasets we can see in Table 3 roughly the same patterns: We first of all see a sharp drop from hinge to hinge^0, a clear increase from hinge to hinge', and a slight decrease from hinge' to bce for the noncrossing parsed data. The results for the unrestricted decoder are similar.

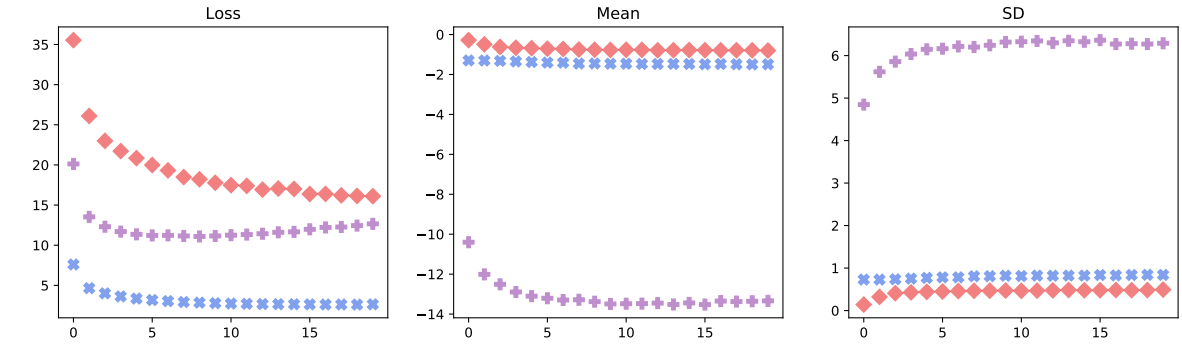
Even though there is mostly no difference between hinge and hinge', the unrestricted decoder with hinge loss significantly outperforms the modified hinge loss system for the out-of-domain PSD data. While the differences are still small, they suggest that the values at which individual losses are cut in the loss function of Equation (10) might be too small.

Table 3: Results on the in-domain (id) and out-of-domain (ood) test sets of the SDP datasets. We emphasize the score of the best performing system, if it performs significantly better than the second best system.

Flavour	Decoder	Data	hinge	hinge^0	hinge'	bce
DM	ncdag	id	84.4	70.4	**88.3**	87.4
		ood	79.2	67.8	**83.1**	82.1
	unrestricted	id	89.3	64.6	89.0	88.0
		ood	83.5	63.1	83.7	82.3
PAS	ncdag	id	85.1	77.7	**90.4**	89.7
		ood	80.8	76.3	**86.0**	85.4
	unrestricted	id	91.5	74.4	91.3	90.5
		ood	87.0	72.4	87.1	86.1
PSD	ncdag	id	73.6	11.9	**75.6**	74.8
		ood	71.1	14.8	**73.3**	72.4
	unrestricted	id	76.4	13.7	76.4	75.4
		ood	**74.1**	16.8	73.5	72.5



(a) hinge $\star$



(b) hinge^0 $\diamond$, hinge' $\times$, bce $+$

Figure 4: Plots visualizing the learning behaviour of the four loss functions when parsing the PAS data with the noncrossing algorithm.

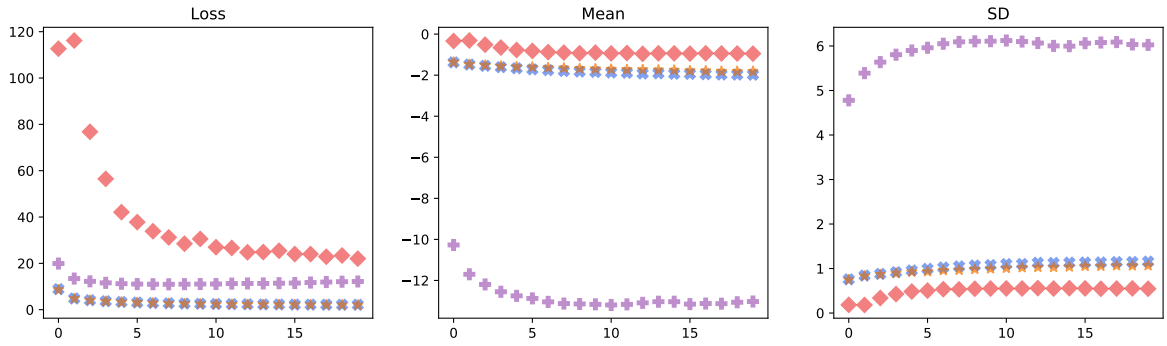


Figure 5: Plots visualizing the learning behaviour of the four loss (hinge $\star$, hinge^0 $\diamond$, hinge' $\times$, bce $+$) functions when parsing the PAS data with the unrestricted algorithm.

Looking at the plot for the noncrossing decoder from Figure 4a we see that the unfixed hinge loss decreases indefinitely. The scores for the arcs become more and more extreme, seen by the decreasing mean and increasing standard deviation. This behaviour is not visible for the hinge loss paired with the unrestriced decoder in Figure 5, which resembles a regular loss function's plot and has scores that are mostly negative but still close to zero.

The fixed hinge loss in Figures 4b and 5 looks similar to the unmodified hinge loss further showing that it fixes the problem caused by the structural constraint, without interfering when no constraint is used. Looking at the behaviour of hinge^0 in Figures 4b and 5, the network seems to learn efficiently, with a loss decreasing to a lower bound, and both reasonable mean and standard deviation. The loss reaches its minimum too early, showing that while some learning is made, the process is suboptimal as shown by the evaluation results in Table 3. This hopeful behaviour is however not visible when parsing the PSD dataset. Here the hinge^0 loss does not provide a sufficient signal to the model to learn. In contrast to the DM and PAS datasets, the PSD dataset uses labels that are much more difficult to predict. The loss of the labels is added independently to the arc loss, and is thus not affected by the cut-off at zero. DM and PAS both also get feedback on which arcs to predict, by the label loss, whereas this signal is much weaker and more difficult to interpret for the PSD data. The performance decrease of the hinge^0 loss for arcs is thus caught by the label loss for DM and PAS, but fails completely for PSD.

The systems trained with binary cross entropy show that the problem of using a structural constraint on the decoder has no negative impact when computing the loss directly on the arcs' scores (e.g. Fig. 4b). However, it also seems that this particular network for parsing performs better when a hinge loss is applied. This might point to some network structures being trained more effectively when using a hinge loss.

4.3 Universal Dependencies

For the experiments on syntactic trees we choose the widely used, freely available, Universal Dependencies (UD version 2.3) treebank (Nivre et al., 2017). Following de Lhoneux et al. (2017) we select a set of languages with varying degrees of projectivity. The final set of languages used, their size and projectivity metrics is presented in Table 4, ordered after

Table 4: UD Statistics. Percentage of projective arcs (A) and graphs (G) and total number of tokens and sentences.

Language	A	G	#Words	#Sentences
A.Greek	95.32	61.05	214,015	17,081
Basque	96.04	67.07	121,443	8,993
Latin	95.55	69.59	199,958	18,400
Portuguese	98.32	76.81	227,794	9,365
Arabic	99.70	90.53	282,384	7,664
German	99.43	91.53	292,788	15,590
French	99.70	93.39	400,440	16,342
B.Portuguese	99.77	94.40	319,380	12,078
English	99.65	96.09	254,829	16,622

increasing percentage of projective graphs[1]. We choose to not use language-specific POS tags and use Universal POS tags instead. This keeps the variation between languages small and reduces the number of parameters to a minimum, making the results between languages more comparable. The Eisner decoder for projective trees and the CLE decoder for maximum-spanning trees (mst) used in the experiments are taken from UniParse (Varab and Schluter, 2018).

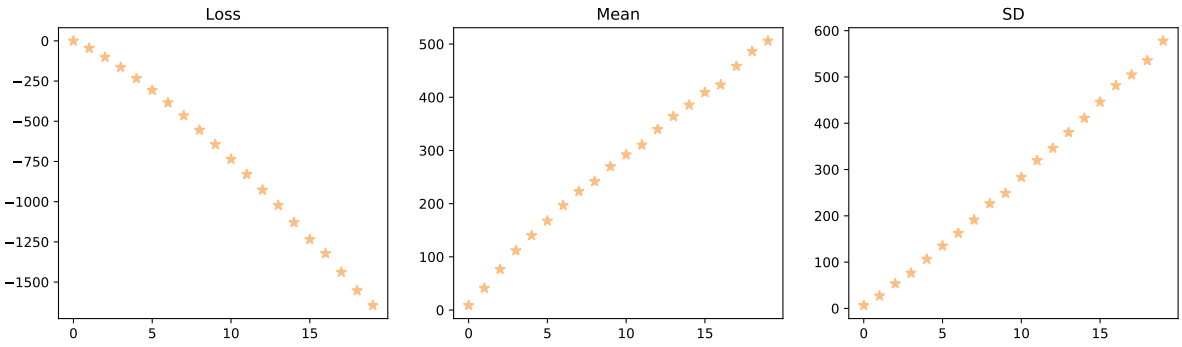
4.3.1 Results

Table 5 presents the results for all languages, decoders and loss functions. As in Table 3 we mark the best performing system if it is significantly better than the second best system. Systems trained with the cross-entropy nearly always perform significantly worse than hinge'.The table is sorted as in Table 4, that is least to most projective treebank. Figures 6 and 7 show the development of the loss on the training data, the mean and standard deviation of the arc scores, for the experiments on the Ancient Greek data. For comparison, we additionally report the training behaviour for a more projective English dataset in Figure 8.

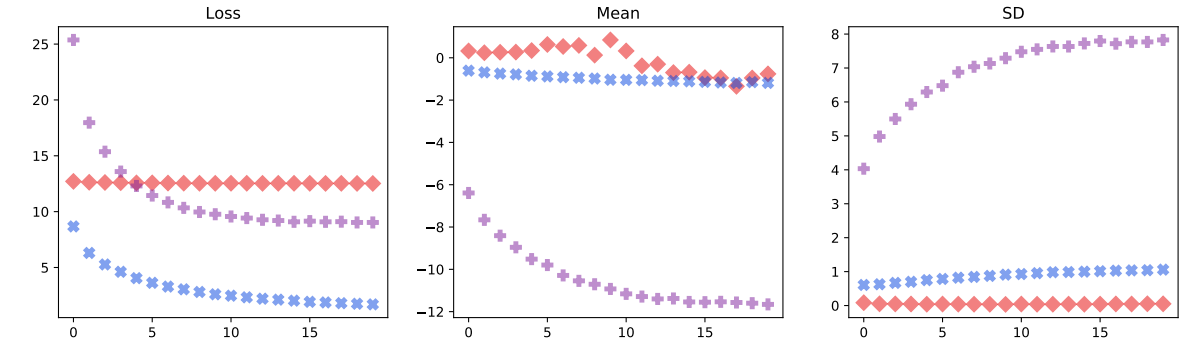
4.3.2 Discussion

The results of the experiments on syntactic dependency trees follow the same patterns as the results on semantic dependency graph parsing. Binary cross entropy avoids the problem of structural constraints and lags behind in terms of labelled F1-score. Using a maximum at zero (hinge^0) to restore the intended minimum of the hinge loss results in subpar performance, that is even worse than when a structural constraint is combined with the unfixed hinge loss. The zero gradients that result when the maximum at zero is applied, stop learning in too many instances, reducing the overall performance in every case. The differences between the unfixed and fixed hinge losses are minimal when the CLE-algorithm is used. For the Eisner decoder, we see a clear increase in accuracy when

[1]Ancient Greek – PROIEL, Basque – BDT, Latin – PROIEL, Portuguese – Bosque, Arabic – PADT, German – GSD, French – GSD, Brazilian Portuguese – GSD, English – EWT



(a) hinge $\star$



(b) hinge^0 $\diamond$, hinge' $\times$, bce $+$

Figure 6: Plots visualizing the learning behaviour of the four loss functions when parsing the Ancient Greek data with the projective algorithm.

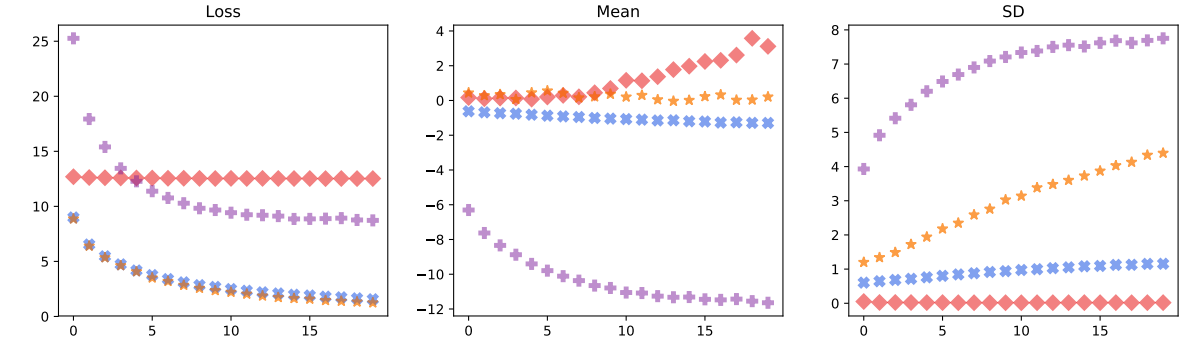


Figure 7: Plots visualizing the learning behaviour of the four loss functions (hinge $\star$, hinge^0 $\diamond$, hinge' $\times$, bce $+$) when parsing the Ancient Greek data with the CLE algorithm.

Table 5: Results on the UD datasets. We emphasize the score of the best performing system, if it performs significantly better than the second best system.

Language	Decoder	hinge	hinge^0	hinge'	bce
A.Greek	eisner	66.7	57.8	**74.9**	73.1
	mst	76.3	55.2	76.2	74.3
Basque	eisner	71.7	58.9	**75.3**	73.3
	mst	75.0	52.9	**76.0**	74.4
Latin	eisner	63.7	51.3	**70.1**	68.8
	mst	70.3	50.8	**71.9**	69.6
Portuguese	eisner	83.4	65.4	**84.6**	83.9
	mst	84.8	63.3	85.1	84.4
Arabic	eisner	**79.6**	57.3	79.1	78.7
	mst	79.3	51.7	79.2	78.4
German	eisner	79.9	66.4	**81.0**	79.6
	mst	80.4	62.4	80.6	79.6
French	eisner	86.2	71.1	86.6	85.8
	mst	86.5	67.2	86.8	85.2
B.Portuguese	eisner	90.1	72.8	90.3	89.8
	mst	89.8	67.3	**90.3**	89.5
English	eisner	85.1	69.2	85.6	84.4
	mst	85.6	67.6	85.7	84.5

using hinge' compared to hinge, albeit with diminishing returns when the percentage of projective arcs increases. Parsing Arabic with the projective system however is an interesting outlier. Even though Arabic is only 90.53% projective in terms of complete graphs, the best performing system uses the projective algorithm and unfixed hinge loss. This can be partially explained away by the fact that 99.7% of the arcs are projective, thus being one of the most projective languages tested (see Tab. 4). Similar to the unexpected results on the out-of-domain PSD dataset, this further suggests that our chosen cut-off values are not optimal.

Plotting the loss shows the same characteristics for the unrestricted CLE decoder and the restricted Eisner decoder, with both the unfixed and fixed hinge losses, as previously for the SDP data. If a structural constraint is used in combination with the regular hinge loss, the loss decreases indefinitely, while the scores of the matrix become more and more extreme. The same slow reduction of the mean towards a minimum and comparative increase of standard deviation as in most previous plots for systems with reasonable results is seen when the fixed hinge loss is applied instead. But the fixed hinge loss even has a regularizing effect, when no structural constraint is used (see Fig. 7). When using our newly introduced hinge loss, the mean of the scores becomes slightly negative (usually between -1 and -2) indicating that most arcs are not even considered. Additionally, the standard deviation caps out between 1 and 2, indicating that there is less extreme

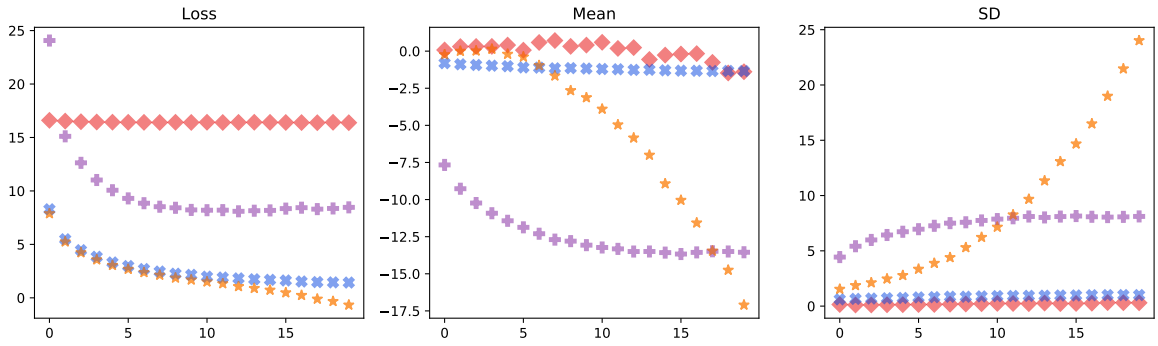


Figure 8: Plots visualizing the learning behaviour of the four (hinge $\star$, hinge^0 $\diamond$, hinge' $\times$, bce $+$) functions when parsing the English data with the projective algorithm.

variation in scores.

The plots of Figure 8 indicate that using the Eisner decoder with the regular hinge loss, should produce subpar results as well, compared with the more regular plots. The loss decreases below zero, the mean drops towards negative infinity, while the standard deviation analogously increases indicating extreme positive and negative arc scores. Looking at the results however, we do not see the same drop in performance as for the more non-projective languages. The best performing model is already achieved after only 4 iterations over the training data. After this maximum is reached we see a slight but noticeable decrease in performance, that indicates some of the negative impact of the faulty loss function. While this rather small neural network reaches its full potential rather quickly, a bigger system might suffer the consequences of the unfixed hinge loss before realizing its potential.

5 Discussion

Both experiments clearly show the predictions made in Section 3:

(i) the hinge loss is faulty and decreases infinitely,

(ii) cutting off at zero hinders learning,

(iii) there is barely any unwanted influence of the adjusted loss when an unrestricted decoder is used.

The gains in labelled F1-score depend on how much the constraint excludes arcs that we would want to predict. To what extent the decrease in coverage influences the drop in performance is however not entirely clear, as for example both PSD and Basque have coverage statistics similar to other overly constrained datasets, but not the same sharp increases in performance when adjusting the loss. This might be partially due to the relative difficulty of parsing PSD (especially labelled parsing), and the comparatively small training data for Basque.

The additionally observed regularizing effect of the proposed loss function on the scores produced by the neural network however is interesting, as it manifests even when no constraint is used. The majority of potential arcs is not part of the dependency graph and should thus ideally have negative weights, whereas the scores of correct arcs

should be positive: the mean is slightly negative and standard deviation keeps most scores around zero. This effect might further help high-performance systems to avoid overfitting, additionally to other regularization techniques that are already commonly used, such as dropout, batching or additional penalty terms designed to keep the model size small (L_2-regularization). These techniques are possibly part of the reason why this unfortunate interplay between a structural constraint and the structural hinge loss has not been reported before. Larger neural networks and the preference of unrestricted decoders have possibly had their influence as well, why this phenomenon was not visible.

6 Conclusion

The definition of the structural hinge loss, when combined with a decoder that has reduced coverage, results in an unconstrained loss that leads to unwanted updates of the neural network. In order to fix this inconvenient interaction, we have suggested a modified loss function that is constrained below zero and designed to halt learning for parameters that do not need to be updated, but would receive an update with the unmodified structural hinge loss. The predictions made by our theoretical analysis were confirmed in experiments using both semantic dependency graphs and syntactic dependency trees. We show for both types of graphs how the loss develops for an ill-defined loss function, with weights being updated to increasingly extreme values. The experiments additionally uncovered a regularizing effect on the values the network outputs for predicting arcs, even when combined with an unrestricted decoder for trees. This suggests that adjusting the loss function has the potential to further regularize neural networks to avoid overfitting and increase performance.

The unmodified definition of the structural hinge loss has obvious flaws that can and should be avoided. While some these effects may be offset by other more common regularization techniques, this general interaction should not be ignored even if it is not visible during evaluation. We look forward to seeing whether there are more instances in other areas using neural networks and in NLP in particular, where the coverage of the search space does not match the data, causing an unfavourable interplay of loss and constraint.

References

Berg-Kirkpatrick, Taylor, David Burkett, and Dan Klein. 2012. An Empirical Investigation of Statistical Significance in NLP. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 995–1005. Jeju Island, Korea: Association for Computational Linguistics.

Cao, Junjie, Sheng Huang, Weiwei Sun, and Xiaojun Wan. 2017. Parsing to 1-Endpoint-Crossing, Pagenumber-2 Graphs. *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* 1:2110–2120.

Chen, Danqi and Christopher Manning. 2014. A Fast and Accurate Dependency Parser using Neural Networks. In *Proceedings of the 2014 Conference on Empirical Methods in*

Natural Language Processing (EMNLP), pages 740–750. Doha, Qatar: Association for Computational Linguistics.

Chu, Yoeng-Jin and Tseng-Hong Liu. 1965. On the shortest arborescence of a directed graph. *Scientia Sinica* 14:1396–1400.

de Lhoneux, Miryam, Sara Stymne, and Joakim Nivre. 2017. Arc-Hybrid Non-Projective Dependency Parsing with a Static-Dynamic Oracle. *Proceedings of the 15th International Conference on Parsing Technologies* pages 99–104.

Dozat, Timothy and Christopher D. Manning. 2017. Deep Biaffine Attention for Neural Dependency Parsing. *ICLR* .

Dozat, Timothy and Christopher D. Manning. 2018. Simpler but More Accurate Semantic Dependency Parsing. *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)* 2:484–490.

Edmonds, Jack. 1967. Optimum Branchings. *Journal of Research of the national Bureau of Standards B* 71(4):233–240.

Eisner, Jason and Giorgio Satta. 1999. Efficient Parsing for Bilexical Context-Free Grammars and Head Automaton Grammars. In *ACL*. ACL.

Flickinger, Dan, Jan Hajič, Angelina Ivanova, Marco Kuhlmann, Yusuke Miyao, Stephan Oepen, and Daniel Zeman. 2016. SDP 2014 & 2015: Broad Coverage Semantic Dependency Parsing LDC2016T10 .

Francis, W. Nelson and Henry Kučera. 1985. Frequency Analysis of English Usage: Lexicon and Grammar. *Journal of English Linguistics* 18(1):64–70.

Garg, Nikhil and James Henderson. 2011. Temporal Restricted Boltzmann Machines for Dependency Parsing. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 11–17. Portland, Oregon, USA: Association for Computational Linguistics.

Greff, Klaus, Rupesh Kumar Srivastava, Jan Koutńık, Bas R. Steunebrink, and Jürgen Schmidhuber. 2015. LSTM: A Search Space Odyssey. *CoRR* abs/1503.04069.

Hajic, Jan, Eva Hajicová, Jarmila Panevová, Petr Sgall, Ondřej Bojar, Silvie Cinková, Eva Fućiková, Marie Mikulová, Petr Pajas, Jan Popelka, Jiří Semecký, Jana Sindlerová, Jan Šťepánek, Josef Toman, Zdenka Uresová, and Zdenek Zabokrtský. 2012. Announcing Prague Czech-English Dependency Treebank 2.0. In *Proceedings of the 8th International Conference on Language Resources and Evaluation (LREC 2012)*, pages 3153–3160. İstanbul, Turkey: European Language Resources Association.

Henderson, James. 2004. Discriminative Training of a Neural Network Statistical Parser. In *Proceedings of the 42Nd Annual Meeting on Association for Computational Linguistics*, ACL '04. Stroudsburg, PA, USA: Association for Computational Linguistics.

Hochreiter, Sepp and Jürgen Schmidhuber. 1997. Long Short-term Memory. *Neural computation* 9:1735–80.

Ivanova, Angelina, Stephan Oepen, Lilja Øvrelid, and Dan Flickinger. 2012. Who Did What to Whom?: A Contrastive Study of Syntacto-semantic Dependencies. In *Proceedings of the Sixth Linguistic Annotation Workshop*, LAW VI '12, pages 2–11. Stroudsburg, PA, USA: Association for Computational Linguistics.

Kingma, Diederik P. and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. *CoRR* abs/1412.6980.

Kiperwasser, Eliyahu and Yoav Goldberg. 2016. Simple and Accurate Dependency Parsing Using Bidirectional LSTM Feature Representations. *Transactions of the Association of Computational Linguistics* 4(1):313–327.

Kübler, Sandra, Ryan T. McDonald, and Joakim Nivre. 2009. *Dependency Parsing*. Synthesis Lectures on Human Language Technologies. Morgan & Claypool Publishers.

Kuhlmann, Marco and Peter Jonsson. 2015. Parsing to Noncrossing Dependency Graphs. *Transactions of the Association of Computational Linguistics* 3(1):559–570.

Kummerfeld, Jonathan K. and Dan Klein. 2017. Parsing with Traces: An $O(n^4)$ Algorithm and a Structural Representation. *Transactions of the Association for Computational Linguistics* 5.

Kurtz, Robin and Marco Kuhlmann. 2017. Exploiting Structure in Parsing to 1-Endpoint-Crossing Graphs. *Proceedings of the 15th International Conference on Parsing Technologies* pages 78–87.

Marcus, Mitchell P., Mary Ann Marcinkiewicz, and Beatrice Santorini. 1993. Building a Large Annotated Corpus of English: The Penn Treebank. *Comput. Linguist.* 19(2):313–330.

Martins, André F. T. and Mariana S. C. Almeida. 2014. Priberam: A Turbo Semantic Parser with Second Order Features. In *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*, pages 471–476. Dublin, Ireland: Association for Computational Linguistics.

Mayberry III, Marshall R. and Risto Miikkulainen. 1999. SARDSRN: A Neural Network Shift-Reduce Parser. In *Proceedings of the 16th Annual International Joint Conference on Artificial Intelligence (IJCAI-99)*, pages 820–825. San Francisco, CA: Kaufmann.

McDonald, Ryan T., Fernando Pereira, Kiril Ribarov, and Jan Hajic. 2005. Non-Projective Dependency Parsing using Spanning Tree Algorithms. In *HLT/EMNLP*, pages 523–530. The Association for Computational Linguistics.

Miyao, Yusuke. 2006. *From Linguistic Theory to Syntactic Analysis: Corpus-Oriented Grammar Development and Feature Forest Model*. Ph.D. thesis.

Moss, Henry, Andrew Moore, David Leslie, and Paul Rayson. 2019. FIESTA: Fast IdEntification of State-of-The-Art models using adaptive bandit algorithms. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2920–2930. Florence, Italy: Association for Computational Linguistics.

Neubig, Graham, Chris Dyer, Yoav Goldberg, Austin Matthews, Waleed Ammar, Antonios Anastasopoulos, Miguel Ballesteros, David Chiang, Daniel Clothiaux, Trevor Cohn, Kevin Duh, Manaal Faruqui, Cynthia Gan, Dan Garrette, Yangfeng Ji, Lingpeng Kong, Adhiguna Kuncoro, Gaurav Kumar, Chaitanya Malaviya, Paul Michel, Yusuke Oda, Matthew Richardson, Naomi Saphra, Swabha Swayamdipta, and Pengcheng Yin. 2017. DyNet: The Dynamic Neural Network Toolkit. *arXiv preprint arXiv:1701.03980* .

Nivre, Joakim, Željko Agić, Lars Ahrenberg, Maria Jesus Aranzabe, Masayuki Asahara, Aitziber Atutxa, Miguel Ballesteros, John Bauer, Kepa Bengoetxea, Riyaz Ahmad Bhat, Eckhard Bick, Cristina Bosco, Gosse Bouma, Sam Bowman, Marie Candito, Gülşen Cebiroğlu Eryiğit, Giuseppe G. A. Celano, Fabricio Chalub, Jinho Choi, Çağrı Çöltekin, Miriam Connor, Elizabeth Davidson, Marie-Catherine de Marneffe, Valeria de Paiva, Arantza Diaz de Ilarraza, Kaja Dobrovoljc, Timothy Dozat, Kira Droganova, Puneet Dwivedi, Marhaba Eli, Tomaž Erjavec, Richárd Farkas, Jennifer Foster, Cláudia Freitas, Kataŕına Gajdošová, Daniel Galbraith, Marcos Garcia, Filip Ginter, Iakes Goenaga, Koldo Gojenola, Memduh Gökırmak, Yoav Goldberg, Xavier Gómez Guinovart, Berta Gonzáles Saavedra, Matias Grioni, Normunds Grūzītis, Bruno Guillaume, Nizar Habash, Jan Hajič, Linh Hà Mỹ, Dag Haug, Barbora Hladká, Petter Hohle, Radu Ion, Elena Irimia, Anders Johannsen, Fredrik Jørgensen, Hüner Kaşıkara, Hiroshi Kanayama, Jenna Kanerva, Natalia Kotsyba, Simon Krek, Veronika Laippala, Phuong Lê H`ông, Alessandro Lenci, Nikola Ljubešić, Olga Lyashevskaya, Teresa Lynn, Aibek Makazhanov, Christopher Manning, Cătălina Mărănduc, David Mareček, Héctor Martínez Alonso, André Martins, Jan Mašek, Yuji Matsumoto, Ryan McDonald, Anna Missilä, Verginica Mititelu, Yusuke Miyao, Simonetta Montemagni, Amir More, Shunsuke Mori, Bohdan Moskalevskyi, Kadri Muischnek, Nina Mustafina, Kaili Müürisep, Luong Nguỹên Th i, Huỳên Nguỹên Th i Minh, Vitaly Nikolaev, Hanna Nurmi, Stina Ojala, Petya Osenova, Lilja Øvrelid, Elena Pascual, Marco Passarotti, Cenel-Augusto Perez, Guy Perrier, Slav Petrov, Jussi Piitulainen, Barbara Plank, Martin Popel, Lauma Pretkalniņa, Prokopis Prokopidis, Tiina Puolakainen, Sampo Pyysalo, Alexandre Rademaker, Loganathan Ramasamy, Livy Real, Laura Rituma, Rudolf Rosa, Shadi Saleh, Manuela Sanguinetti, Baiba Saulīte, Sebastian Schuster, Djamé Seddah, Wolfgang Seeker, Mojgan Seraji, Lena Shakurova, Mo Shen, Dmitry Sichinava, Natalia Silveira, Maria Simi, Radu Simionescu, Katalin Simkó, Mária Šimková, Kiril Simov, Aaron Smith, Alane Suhr, Umut Sulubacak, Zsolt Szántó, Dima Taji, Takaaki Tanaka, Reut Tsarfaty, Francis Tyers, Sumire Uematsu, Larraitz Uria, Gertjan van Noord, Viktor Varga, Veronika Vincze, Jonathan North Washington, Zdeněk Žabokrtský, Amir Zeldes, Daniel Zeman, and Hanzhi Zhu. 2017. Universal Dependencies 2.0 .

Oepen, Stephan and Jan Tore Lønning. 2006. Discriminant-Based MRS Banking. In *Proceedings of the Fifth International Conference on Language Resources and Evaluation (LREC-2006)*. Genoa, Italy: European Language Resources Association (ELRA).

Peng, Hao, Sam Thomson, and Noah A. Smith. 2017. Deep Multitask Learning for Semantic Dependency Parsing. *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* 1:2037–2048.

Pitler, Emily, Sampath Kannan, and Mitchell Marcus. 2013. Finding Optimal 1-Endpoint-Crossing Trees. *Transactions of the Association of Computational Linguistics* 1:13–24.

Schluter, Natalie. 2014. On maximum spanning DAG algorithms for semantic DAG parsing. *Proceedings of the ACL 2014 Workshop on Semantic Parsing* pages 61–65.

Stenetorp, Pontus. 2013. Transition-based Dependency Parsing Using Recursive Neural Networks. In *Deep Learning Workshop at the 2013 Conference on Neural Information Processing Systems (NIPS)*.

Taskar, Benjamin, Vassil Chatalbashev, Daphne Koller, and Carlos Guestrin. 2005. Learning structured prediction models: A large margin approach. In L. D. Raedt and S. Wrobel, eds., *Machine Learning, Proceedings of the Twenty-Second International Conference (ICML 2005), Bonn, Germany, August 7-11, 2005*, vol. 119 of *ACM International Conference Proceeding Series*, pages 896–903. ACM.

Taskar, Benjamin, Carlos Guestrin, and Daphne Koller. 2003. Max-Margin Markov Networks. In S. Thrun, L. K. Saul, and B. Schölkopf, eds., *Advances in Neural Information Processing Systems 16 [Neural Information Processing Systems, NIPS 2003, December 8-13, 2003, Vancouver and Whistler, British Columbia, Canada]*, pages 25–32. MIT Press.

Titov, Ivan and James Henderson. 2007. Fast and Robust Multilingual Dependency Parsing with a Generative Latent Variable Model. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 947–951. Prague, Czech Republic: Association for Computational Linguistics.

Varab, Daniel and Natalie Schluter. 2018. UniParse: A universal graph-based parsing toolkit. *CoRR* abs/1807.04053.