Enforcing Output Constraints via SGD: A Step Towards Neural Lagrangian Relaxation

Jay Yoon Lee [*] **Michael Wick** [†] **Jean-Baptiste Tristan** [†] **Jaime Carbonell** [*]

Abstract

Structured prediction problems such as named entity recognition and parsing are crucial for automated knowledge base construction. Increasingly, researchers are exploring ways of improving them with neural networks. However, many structured-prediction problems require deterministic constraints on the output values; for example, requiring that the sequential outputs encode a valid tree. While hidden units might capture such properties, the network is not always able to learn them from the training data alone, and practitioners must then resort to post-processing. In this paper, we present an inference method for neural networks that enforces deterministic constraints on outputs without performing post-processing or expensive discrete search. Instead, for each input, we nudge continuous weights until the network's unconstrained inference procedure generates an output that satisfies the constraints. We apply our method to pre-trained networks of various quality for constituency parsing and find that in each case, not only does the algorithm rectify a vast majority of violating outputs, it also improves accuracy.

1 Introduction

Suppose we have trained a sequence-to-sequence network [3, 11, 7] to perform a structured prediction task such as constituency parsing [12]. We would like to apply our network to novel, unseen examples, but still require that the network's outputs obey the appropriate set of hard-constraints; for example, that the output sequence encodes a valid parse tree. Enforcing these constraints is important because down-stream tasks, such as relation extraction or coreference resolution, often assume that the constraints hold. More generally, enforcing constraints is crucial for knowledge-base (KB) construction since we must ensure that independently extracted facts conform to the logical and semantic rules of the particular KB or domain.

Unfortunately, there is no guarantee that the neural network will learn these constraints from the training data alone. Although in some cases, the outputs of state-of-the-art systems almost always obey the constraints for the test-set of the data on which they are tuned [12]; in practice, the quality of machine learning systems are much lower when run on data in the wild (e.g., because small shifts in domain or genre change the underlying data distribution). In such cases, the problem of constraint violations may become significant.

This raises the question: how should we enforce hard constraints on the outputs of a neural network? We could perform expensive discrete search or manually construct a list of post-processing rules for the particular problem domain of interest. Though, we might do even better if we continue to train the neural network at test-time to learn how to satisfy the constraints on each input. Such a learning procedure is applicable at test-time because learning constraints requires no labeled data: rather, we only require a function that measures the extent to which a predicted output violates a constraint.

[*] Carnegie Mellon University,Pittsburgh, PA (contact: lee.jayyoon@gmail.com)
[†] Oracle Labs, Burlington, MA (contact: michael.wick@oracle.com)

In this paper, we present an inference method for neural networks that enforces output constraints by adjusting the network's weights at test-time. Given an appropriate function that measures the extent of a constraint violation, we can express the hard constraints as an optimization problem over the continuous weights and apply back-propagation to change them. That is, by iteratively adjusting the weights so that the neural network becomes increasingly likely to produce an output configuration that obeys the desired constraints. Much like scoped-learning, the algorithm customizes the weights for each example at test-time [2], but does so in a way to satisfy the constraints. We apply our method to pre-trained sequence-to-sequence networks for syntactic parsing that vary in quality by about ten F1 points. We find that in each case, the algorithm satisfies a large percentage of the constraints (up to 94%) and that in almost every case (even for the lower-quality models), enforcing the constraints improves the accuracy.

2 Constraint-aware inference in neural networks

To motivate our algorithm, we begin with the ideal optimization problem and find that unlike for linear models with local constraints [9], the resulting Lagrangian is not well suited for globally constrained inference in non-linear networks. We ultimately settle upon an alternative objective function that reasonably models the constrained inference problem, but is no longer convex. Nevertheless, we find that the algorithm works well in practice.

2.1 Problem definition and motivation

Typically, a neural network parameterized by weights W is a function from an input $\mathbf{x}$ to an output $\mathbf{y}$. The network has an associated compatibility function $\Psi(\mathbf{y}; \mathbf{x}, W) \to \mathbb{R}_+$ that measures how likely an output $\mathbf{y}$ is given an input $\mathbf{x}$ under weights W. The goal of inference is to find an output that maximizes the compatibility function and this is usually accomplished efficiently with feed-forward or greedy-decoding. In this work, we want to additionally enforce that the output values belong to a feasible set or grammar $\mathcal{L}^{\mathbf{x}}$ that in general depends on the input. For example, in syntactic parsing, we might require that the sequence of shift-reduce commands obeys constraints ensuring that they encode a valid parse tree that covers the entire input sentence. We are thus interested in the following optimization problem:

$$\begin{aligned} \max_{\mathbf{y}} \quad & \Psi(\mathbf{x}, \mathbf{y}, W) \\ \text{s.t.} \quad & \mathbf{y} \in \mathcal{L}^{\mathbf{x}} \end{aligned} \tag{1}$$

Feed-forward and greedy inference are no longer sufficient since the outputs might violate the global constraints (i.e., $\mathbf{y} \notin \mathcal{L}^{\mathbf{x}}$). Instead, suppose we had a function $g(\mathbf{y}, \mathcal{L}) \to \mathbb{R}_+$ that measures a loss between a sentence $\mathbf{y}$ and a grammar $\mathcal{L}$ such that $g(\mathbf{y}, \mathcal{L}) = 0$ if and only if there are no grammatical errors in $\mathbf{y}$. That is, $g(\mathbf{y}, \mathcal{L}) = 0$ for the feasible region and is strictly positive everywhere else. For example, if the feasible region is a CFL, g could be the *least errors count* function [8]. We could then express the constraints as an equality constraint and minimize the Lagrangian:

$$\min_{\lambda} \max_{\mathbf{y}} \Psi(\mathbf{x}, \mathbf{y}, W) + \lambda g(\mathbf{y}, \mathcal{L}) \tag{2}$$

However, this leads to optimization difficulties because there is just a single dual variable for our global constraint, resulting in a brute-force trial and error search. Even for cases in which the constraint happens to factor over each output unit, it is not applicable to sequence-to-sequence models for which there are a variable number of outputs (and thus dual variables). This would require some mechanism for adding and removing dual variables on the fly in response to the optimization procedure changing the length of the sequence.

To circumvent these issues, we propose an alternative in which we replace the Lagrange variable with the weights of a neural network. Observe, for example, that the network's weights control the compatibility of the output configurations with the input. By properly adjusting the weights, we can affect the outcome of inference by removing mass from invalid outputs. Unlike the case of a single dual variable, the weights can assign different penalty amounts to different outputs. Further, the weights are likely to generalize across related outputs because in most neural networks, the weights are tied across space (e.g., CNNs) or time (e.g., RNNs). As a result, lowering the compatibility function for a single invalid output has the potential effect of lowering the compatibility for an entire family of related, invalid outputs; enabling faster search. With this in mind, it is tempting to replace the single dual-variable with a "dual neural-network" that is parameterized by a set of "dual weights."

This is powerful because we have effectively introduced an exponential number of implicit "dual variables" (via the compatibility function, which scores each output) that we can easily control via the weights; although similar, the new optimization is no longer equivalent to the original:

$$\min_{W_\lambda} \max_{\mathbf{y}} \Psi(\mathbf{x}, \mathbf{y}, W) + \Psi_\lambda(\mathbf{x}, \mathbf{y}, W_\lambda) g(\mathbf{y}, \mathcal{L}) \tag{3}$$

While a step in the right direction, the objective still requires combinatorial search because (1) the maximization involves two non-linear functions (2) the constraints might be global. In contrast, the functions involved in classic Lagrangian relaxation methods for NLP have multipliers for each output variable that can be combined with linear models to form a single unified decoding problem for which efficient inference exists [6, 10, 9].

2.2 Algorithm

We therefore modify the optimization problem for a final time by (1) removing the compatibility function that involves the original weights W and compensate with a regularizer that attempts to keep the dual weights W_λ as close to these weights as possible, and (2) maximizing exclusively over the network parameterized by W_λ while ignoring the constraint term during the maximization. This results in the following optimization problem:

$$\begin{array}{ll} \min\limits_{W_\lambda} & \Psi(\mathbf{x}, \hat{\mathbf{y}}, W_\lambda) g(\hat{\mathbf{y}}, \mathcal{L}) + \alpha \|W - W_\lambda\|_2 \\ \text{where} & \hat{\mathbf{y}} = \operatorname*{argmax}\limits_{\mathbf{y}} \Psi(\mathbf{x}, \mathbf{y}, W_\lambda) \end{array} \tag{4}$$

While this appears to be a brutal modification, it is reasonable because by definition of the constraint loss $g(\cdot)$, the global minima must correspond to outputs that satisfy all constraints. Further, we expect to find high-probability optima if we initialize $W_\lambda = W$. Moreover, the objective is intuitive: if there is a constraint violation in $\hat{\mathbf{y}}$ then $g(\cdot) > 0$ and we lower the compatibility of $\hat{\mathbf{y}}$ to make it less likely. Otherwise, $g(\cdot) = 0$ and we leave the compatibility of $\hat{\mathbf{y}}$ unchanged.

To optimize the objective, our algorithm (Algorithm 1) alternates maximization to find $\hat{\mathbf{y}}$ and minimization w.r.t. W_λ. In particular, we first approximate the maximization step by employing the neural network's inference procedure (e.g., greedy decoding or beam-search) to find $\hat{\mathbf{y}}$. Then, given a fixed $\hat{\mathbf{y}}$, we minimize the objective with respect to the W_λ by performing stochastic gradient descent (SGD). Since $\hat{\mathbf{y}}$ is fixed, the constraint loss term becomes a constant in the gradient; thus, making it easier to employ external black-box constraint losses (such as those based on compilers) that may not be differentiable. As a remark, note the similarity to REINFORCE [13]: output sentences are states, the decoder outputs are actions and the constraint-loss is a negative reward. However, our algorithm terminates upon discovery of an output that satisfies all constraints.

Algorithm 1 Constrained inference for neural nets

```
Inputs: test instance $\mathbf{x}$, input specific CFL $\mathcal{L}^{\mathbf{x}}$, pretrained weights $W$
$W_\lambda \leftarrow W$  #reset instance-specific weights
while not converged do
    $\mathbf{y} \leftarrow f(\mathbf{x}; W_\lambda)$  #perform inference using weights $W_\lambda$
    $\nabla \leftarrow g(\mathbf{y}, \mathcal{L}^{\mathbf{x}}) \frac{\partial}{\partial W_\lambda} \Psi(\mathbf{x}, \mathbf{y}, W_\lambda) + \alpha \|W - W_\lambda\|_2$ #compute constraint loss
    $W_\lambda \leftarrow W_\lambda - \eta \nabla$  #update instance-specific weights with SGD or a variant thereof
end while
```

3 Application to parsing

Consider the structured prediction problem of syntactic parsing in which the goal is to input a sentence comprising a sequence of tokens and output a tree describing the grammatical parse of the sentence. One way to model the problem with neural networks is to linearize the representation of the parse tree and then employ the familiar sequence-to-sequence model [12]. Let us suppose we linearize the tree using a sequence of shift (`s`) and reduce (`r,r!`) commands that control an implicit shift reduce parser. Intuitively, these commands describe the exact instructions for converting the input sentence into a complete parse tree: the interpretation of the symbol `s` is that we shift an input token onto the stack and the interpretation of the symbol `r` is that we start (or continue) reducing (popping) the top elements of the stack, the interpretation of a third symbol `!` is that we stop reducing and push the reduced result back onto the stack. Thus, given an input sentence and an output sequence of shift-reduce commands, we can deterministically recover the tree by simulating a shift reduce parser. For example, the sequence `ssrr!ssr!rr!rr!` encodes a type-free version of the parse tree `(S (NP the ball) (VP is (NP red)))` for the input sentence "the ball is red". It is easy to recover the tree structure from the input sentence and the output commands by simulating a shift reduce parser, performing one command at a time as prescribed by the classic algorithm.

Note that for output sequences to form a valid tree over the input, the sequence must satisfy a number of constraints. First, the number of shifts must equal the number of input tokens $m_{\mathbf{x}}$, otherwise either the tree would not cover the entire input sentence or the tree would contain spurious terminal symbols. Second, the parser cannot issue a reduce command if there are no items left on the stack. Third, the number of reduces must be sufficient to leave just a single item, the root node, on the stack. The constraint loss $g(\mathbf{y}, \mathcal{L}^x)$ for this task simply counts the errors of each of the three types.

4 Related work

Previous work in enforcing hard constraints for parsing has focused on post-processing [12] or building them into the decoder via sampling [4] or search constraints [14]. More generally, recent work has considered applying neural networks to structured prediction; for example, structured prediction energy networks (SPENS) [1]. SPENS incorporate soft-constraints via back-propagating an energy function into "relaxed" output variables. In contrast, we focus on hard-constraints and back-propagate into the weights that subsequently control the original non-relaxed output variables via inference. Separately, there has been interest in employing hard constraints to harness unlabeled data in semi-supervised learning [5]. Our work instead focuses enforcing constraints at inference-time.

Finally, as previously mentioned, our method highly resembles dual decomposition and more generally Lagrangian relaxation for structured prediction [6, 10, 9]. In such techniques, it is assumed that a computationally efficient inference algorithm can maximize over a superset of the feasible region (this assumption parallels our case because unconstrained inference in the neural network is efficient, but might violate constraints). Then, the method employs gradient descent to concentrate this superset onto the feasible region. However, these techniques are not directly applicable to our non-linear problem with global constraints.

5 Experiments

We investigate the behavior of the constraint satisfaction algorithm on the shift-reduce parsing task described in Section 3. We transform the Wall Street Journal (WSJ) portion of the Penn Tree Bank (PTB) into shift-reduce commands in which each reduce command has a phrase-type (e.g., noun-phrase or verb-phrase). We employ the traditional split of the data with section 22 for dev, section 23 for test, and remaining sections 01-21 for training. We evaluate on the test set with evalb[3] F1. We are interested in answering the following questions (Q1) how well does the sequence-to-sequence network learn the constraints from data (Q2) for cases in which the network is unable to learn the constraints, is our method able to actually enforce the constraints and (Q3) does the method enforce constraints without compromising the quality of the network's output. Q3 is particularly important because we adjust the weights of the network at test-time and this may lead to unexpected behavior.

In each experiment, we learn a sequence-to-sequence network on a training set and then evaluate the network directly on the test set using a traditional inference algorithm to perform the decoding (either greedy decoding or beam-search). Then, to address (Q1) we measure the *failure-rate* (i.e., the ratio of test sentences for which the network infers an output that fails to fully satisfy the constraints). To address (Q2) we evaluate our method on the *failure-set* (i.e., the set of output sentences for which the original network produces invalid constraint-violating outputs) and measure our method's *conversion rate*; that is, the percentage of failures for which our method is able to completely satisfy the constraints (or "convert"). Finally, to address (Q3), we evaluate the quality (e.g., accuracy or F1) of the output predictions on the network's *failure-set* both before and after applying our method.

[3] `http://nlp.cs.nyu.edu/evalb/`

Inference	Network	Failure rate (n/2415)	Conversion rate (%)	F1 (before)	F1 (after)
Greedy	Net0	886	69.86	58.47	60.41
	Net1	971	73.12	59.03	59.82
	Net2	474	88.40	58.77	61.18
	Net3	611	88.05	62.17	64.49
	Net4	317	79.81	65.62	68.79
Beam 2	Net0	602	82.89	60.45	61.35
	Net1	707	86.14	61.30	61.26
	Net2	269	82.53	61.31	61.37
	Net3	419	94.27	65.40	66.65
	Net4	206	87.38	66.61	71.15
Beam 5	Net0	546	81.50	61.43	63.25
	Net1	615	84.72	61.99	62.86
	Net2	220	80.91	61.63	63.34
	Net3	368	92.66	67.18	69.4
	Net4	160	87.50	67.5	71.38
Beam 9	Net0	552	80.62	61.64	62.98
	Net1	613	83.69	62.83	63.95
	Net2	225	80.00	61.04	62.52
	Net3	360	93.89	67.83	70.64
	Net4	153	91.50	68.66	71.69

Table 1: Evaluation of the proposed constrained-inference procedure.

While state-of-the-art networks almost always produce sequences that define valid trees [12]; in practice, the parsing quality of even the best systems degrade in the wild (e.g., due to domain, genre, tokenization, out-of-vocabulary words and data-distribution changes in general) or languages with smaller amounts of training data. In order to study our algorithm on a wide range of more realistic accuracy regimes, we train many networks with different hyper-parameters producing models of various quality. We limit ourselves to the WSJ subset of the PTB. We select five networks (`Net1-5`) with F1 scores, respectively, 71.5,71.6, 73.0, 78.1, 81.2. We study the behavior of the constraint-satisfaction method on the five networks using various inference procedures for decoding: greedy decoding and beam-search with a beam-size of 2, 5, and 9 (resp. beam2, beam5, beam9).

We report the results in Table 1. The left-most column indicates the inference procedure employed in the experiment. The indicated inference procedure is employed both for the initial network prediction and in the inner loop of our algorithm. The failure-rate is given as a fraction of violated outputs over the total number of test examples. This statistic indicates the extent to which constraint-violations are a problem for each initial network prior to applying constrained inference. In order to address question Q2—the ability of our approach to satisfy constraints—we measure conversion rates. As before, the conversion rates are the percentage of the examples in the failure-sets for which the constraint-satisfaction method is able to satisfy all the constraints. Across all the experimental conditions, the conversion rates are high, often above 80 and sometimes above 90. The conversion rates appear to correlate with the quality of the parser: for each inference algorithm, conversion rates tend to be higher for Net3 and Net4 than for Net0 and Net1.

Next, in order to address question Q3—the ability of our approach to satisfy constraints without negatively affecting output quality—we measure the F1 scores on the failure-sets both before and after applying the constraint satisfaction algorithm. Since this F1 measure is only defined on valid trees, we employ heuristic post-processing, as described earlier, to ensure all outputs are valid. We find that in every case, except one, our approach satisfies constraints in a way that improves the quality of the parses (as compared to employing post-processing).

Finally, on outputs that our algorithm converts, we report the number of iterations that it takes. Across all conditions, it takes 5–7 steps to convert 25% of the outputs, 15–20 steps to convert 50%, 39–57 steps to convert 80%, 58–73 steps to convert 90% and 77–84 steps to convert 95%.

6 Conclusion

We presented an algorithm for satisfying constraints in neural networks that avoids combinatorial search, but employs the network's efficient unconstrained procedure as a black box. We evaluated the algorithm on sequence-to-sequence parsing and found that it could satisfy up to 94% of the constraints. An exciting area of future work is to generalize our method and explore the idea of neural Lagrangian relaxation, in which a neural network replaces the dual variable in the Lagrangian optimization problem. In much the same way that neural networks have successfully modeled the latent variables in variational learning, we hope that the networks could learn the Lagrange variables and provide extremely fast amortized inference for constrained optimization.

References

[1] David Belanger and Andrew McCallum. Structured prediction energy networks. In *International Conference on Machine Learning*, 2016.

[2] David M. Blei, Andrew Bagnell, and Andrew K. McCallum. Learning with scope, with application to information extraction and classification. In *Uncertainty in Artificial Intelligence (UAI)*, 2002.

[3] Kyunghyun Cho, Bart Van Merriënboer, Çağlar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734. Association for Computational Linguistics, October 2014.

[4] Chris Dyer, Adhiguna Kuncoro, Miguel Ballesteros, and Noah A. Smith. Recurrent neural network grammars. In *NAACL-HLT*, pages 199–209, 2016.

[5] Zhiting Hu, Xuezhe Ma, Zhengzhong Liu, Eduard Hovy, and Eric P. Xing. Harnessing deep neural networks with logical rules. In *Association for Computational Linguistics (ACL)*, 2016.

[6] Terry Koo, Alexander M Rush, Michael Collins, Tommi Jaakkola, and David Sontag. 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. Association for Computational Linguistics, 2010.

[7] Ankit Kumar, Ozan Irsoy, Peter Ondruska, Mohit Iyyer, James Bradbury, Ishaan Gulrajani, Victor Zhong, Romain Paulus, and Richard Socher. Ask me anything: Dynamic memory networks for natural language processing. *Machine Learning*, pages 1378–1387, 2016.

[8] Gordon Lyon. Syntax-directed least-errors anallysis for context-free languages: A practical approach. *Programming Languages*, 17(1), January 1974.

[9] Alexander M. Rush and Michael Collins. A tutorial on dual decomposition and lagrangian relaxation for inference in natural language processing. *Journal of Artificial Intelligence Research*, 45:305–362, 2012.

[10] Alexander M Rush, David Sontag, Michael Collins, and Tommi Jaakkola. On dual decomposition and linear programming relaxations for natural language processing. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 1–11. Association for Computational Linguistics, 2010.

[11] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. In *Neural Information Processing Systems (NIPS)*, 2014.

[12] Oriol Vinyals, Luksz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever, and Geoffrey Hinton. Grammar as a foreign language. In *NIPS*, 2015.

[13] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 1992.

[14] Sam Wiseman and Alexander M. Rush. Sequence-to-sequence learning as beam-search optimization. In *Empirical Methods in Natural Language Processing*, pages 1296–1306, 2016.