Gradient-based Inference for Networks with Output Constraints

Jay Yoon Lee * jaylee@cs.cmu.edu
Michael Wick † michael.wick@oracle.com
Sanket Vaibhav Mehta * svmehta@cs.cmu.edu
Jean-Baptiste Tristan † jean.baptiste.tristan@oracle.com
Jaime Carbonell * jgc@cs.cmu.edu

Abstract

Practitioners apply neural networks to increasingly complex problems in natural language processing (NLP), such as syntactic parsing that have rich output structures. Many such structured-prediction problems require deterministic constraints on the output values; for example, in sequence-to-sequence syntactic parsing, we require that the sequential outputs encode valid trees. While hidden units might capture such properties, the network is not always able to learn such constraints 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 rule-based post-processing or expensive discrete search. Instead, in the spirit of gradient-based training, we enforce constraints with gradient-based *inference*: for each input at test-time, we nudge continuous weights until the network's unconstrained inference procedure generates an output that satisfies the constraints. We apply our methods to three tasks with hard constraints: sequence transduction, constituency parsing, and semantic role labeling (SRL). In each case, the algorithm not only satisfies constraints, but improves accuracy, even when the underlying network is state-of-the-art.

1 Introduction

Suppose we have trained a sequence-to-sequence network [4, 17, 11] to perform a structured prediction task such as constituency parsing [18]. 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 typically assume that the constraints hold. Moreover, the constraints impart informative hypothesis-limiting restrictions about joint assignments to multiple output units, and thus enforcing them holistically might cause a correct prediction for one subset of the outputs to beneficially influence another.

Unfortunately, there is no guarantee that the neural network will learn these constraints from the training data alone, especially if the training data is limited. Although in some cases, the outputs of state-of-the-art systems mostly obey the constraints for the test-set of the data on which they are tuned [18]; 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 becomes more significant.

* School of Computer Science, Carnegie Mellon University, Pittsburgh, PA
† Oracle Labs, Burlington, MA

Preprint. Work in progress.

This raises the question: how should we enforce hard constraints on the outputs of a neural network? We could perform expensive combinatorial 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.

In this paper, we present an inference method for neural networks that strongly favors respecting output constraints by adjusting the network's weights at test-time, for each input. 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 tune 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 [3], but does so in a way to satisfy the constraints. We apply our method to three tasks: syntactic parsing, semantic role labeling and a synthetic sequence transduction problem and find that the algorithm performs favorably on all three tasks. For parsing, we apply the method to pre-trained networks that vary widely in quality, (e.g., by about ten F1 points on parsing), and on both *in* and *out*-of-domain data. For this task and others, we find that in each case, the algorithm satisfies a large percentage of the constraints (up to 98%) and that in almost every case (out-of-domain data, state-of-the art networks, and even for the lower-quality networks), enforcing the constraints improves the accuracy. On SRL, for example, the method successfully injects side-information, improving a state-of-the-art network by 1.5% F1 [8].

2 Constraint-aware inference in neural networks

Our goal is to design an *approximate* optimization algorithm that is similar in spirit to Lagrangian relaxation in that we replace a complex constrained decoding objective with a simpler unconstrained objective that we can optimize with gradient descent [10, 16, 15], but is better suited for non-linear non-convex optimization with global constraints that do not factorize over the outputs. Although the exposition in this section revolves around Lagrangian relaxation, we emphasize that the purpose is merely to provide intuition and motivate design choices.

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 $\Psi(\mathbf{y}; \mathbf{x}, W) \rightarrow \mathbb{R}_+$ 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 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:

$$\begin{array}{ll} \max\limits_{\mathbf{y}} & \Psi(\mathbf{x}, \mathbf{y}, W) \\ \text{s. t.} & \mathbf{y} \in \mathcal{L}^{\mathbf{x}} \end{array} \tag{1}$$

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 $g(\mathbf{y}, \mathcal{L}) \rightarrow \mathbb{R}_+$ 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 [12]. 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 variable number of 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.

Instead, we might circumvent these issues if we optimize over a trainable neural network rather than a single dual variable. Intuitively, the purpose of the dual variables is to simply penalize the score of *infeasible* outputs that otherwise have a high score in the network, but happen to violate constraints. Similarly, observe 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—in much the same way a dual variable affects the outcome of inference. Unlike a single dual variable however, the network expresses a *different* penalty weight for each output. And, because the weights are typically tied across space (e.g., CNNs) or time (e.g., RNNs) the weights are likely to generalize across related outputs. 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. In the next subsection, we propose a novel approach that utilizes the amount of constraint violation as part of the objective function so that we can adjust the model parameters to search for a constraint-satisfying output efficiently.

2.2 Algorithm

Instead of solving the aforementioned impractical optimization problem, we propose to optimize a "dual" set of model parameters W_λ over the constraint function while regularizing the optimized model weights to stay close to the originally learned original pre-trained weights W. The objective function is as follows:

$$\begin{array}{ll} \min\limits_{W_\lambda} & \Psi(\mathbf{x}, \hat{\mathbf{y}}, W_\lambda) g(\hat{\mathbf{y}}, \mathcal{L}) + \alpha \|W - W_\lambda\|_2 \\ \text{where} & \hat{\mathbf{y}} = \operatorname{argmax}\limits_{\mathbf{y}} \Psi(\mathbf{x}, \mathbf{y}, W_\lambda) \end{array} \tag{3}$$

Although this objective deviates from the original optimization problem, 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 the gradient will lower the compatibility of $\hat{\mathbf{y}}$ to make it less likely. Otherwise, $g(\cdot) = 0$ and the gradient of the energy is zero and we leave the compatibility of $\hat{\mathbf{y}}$ unchanged. Crucially, the optimization problem yields computationally efficient subroutines that we exploit in the optimization algorithm.

To optimize the objective, the algorithm alternates maximization to find $\hat{\mathbf{y}}$ and minimization w.r.t. W_λ (Algorithm 1). 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 the $\hat{\mathbf{y}}$ that approximately maximizes Ψ, which ignores the constraint loss g. 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 [20]: 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_λ
  $\nabla \leftarrow g(\mathbf{y}, \mathcal{L}^{\mathbf{x}}) \frac{\partial}{\partial W_\lambda} \Psi(\mathbf{x}, \mathbf{y}, W_\lambda) + \alpha \frac{W - W_\lambda}{\|W - W_\lambda\|_2}$ #compute constraint loss
  $W_\lambda \leftarrow W_\lambda - \eta \nabla$ #update instance-specific weights with SGD or a variant thereof
end while

⟨“ So it ’s a very mixed bag . ”⟩ ⟶ `sssr!sssssrr!srrr!rr!ssrrrrrr!`

iteration	output	loss	accuracy
0	`ssr!sr!ssssrrr!rr!ssrrrrrr!`	0.0857	33.3%
11	`ssr!sr!ssssrrr!rr!ssrrrrrr!`	0.0855	33.3%
12	`sssr!sssssrr!srrr!rr!ssrrrrrr!`	0.0000	100.0%

Table 1: A shift-reduce example for which the method successfully enforces constraints. The initial output has only nine shifts, but there are ten tokens in the input. Enforcing the constraint not only corrects the number of shifts to ten, but changes the implied tree structure to the correct tree.

3 Application to parsing

As an illustrative example, 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 [18]. 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 the shift reduce parser. Of course in practice, reduce commands include the phrase-types (NP, VP, etc).

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 [18] or building them into the decoder via sampling [5] or search constraints [21]. More generally, recent work has considered applying neural networks to structured prediction; for example, structured prediction energy networks (SPENS) [2]. 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 [9]. 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 [10, 16, 15]. 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.

⟨“ So it ’s a very mixed bag . ”⟩ ⟶ `sssr!ssssrr!srrr!rr!ssrrrrrr!`

inference method	output
unconstrained-decoder	`ssr!sr!ssssrrr!  rr!ssrrrrrr!`
constrained-decoder	`ssr!sr!ssssrrr!  rr!ssrrrrrr!srr!`
our method	`sssr!ssssrr!srrr!rr!ssrrrrrr!`
true parse	`sssr!ssssrr!srrr!rr!ssrrrrrr!`

Table 2: A shift-reduce example for which the method successfully enforces constraints. The initial unconstrained decoder prematurely reduces “So it” into a phrase, missing the contracted verb “is.” Errors then propagate through the sequence culminating in the final token missing from the tree (a constraint violation). The constrained decoder is only able to deal with this at the end of the sequence, while our method is able to harness the constraint to correct the early errors.

5 Experiments

In this section we study our algorithm on three different tasks: syntactic parsing, semantic role labeling (SRL), and a synthetic sequence transduction task. All three tasks require hard constraints, but they play a different role in each. In syntactic parsing, constraints ensure that the outputs encode valid trees, in semantic role labeling, constraints provide side-information about possible true-spans and in the sequence transduction tasks they force the output to belong to a particular input-dependent regular expression. The parsing and transduction tasks provide an opportunity to study the algorithm on various sequence-to-sequence networks while the SRL task involves more traditional recurrent neural networks that have exactly one output per input token.

We are interested in answering the following questions (Q1) how well does the neural 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.

5.1 Syntactic parsing

We investigate the behavior of the constraint inference 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) [6]. 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.

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.

While state-of-the-art networks almost always produce sequences that define valid trees [18]; 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, from high to low, using the standard split of the WSJ portion of the PTB. In total, we train five networks `Net1-5` for this study, that we describe below.

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

name	F1		hyper-parameters		
	BS-9	greedy	hidden	layers	dropout
Net1	—	87.33	128	3	yes
Net2	—	86.78	128	3	yes
Net3	81.26	78.32	172	3	yes
Net4	78.14	74.53	128	3	no
Net5	71.54	67.80	128	3	no

Table 3: Parsing Networks (BS9 means beam size 9)

Inference	Network	Failure rate (n/2415)	Conversion rate	F1 (before)	F1 (after)
Greedy	Net1	187	93.58	71.49	77.04
	Net2	287	89.20	73.54	79.68
	Net3	317	79.81	65.62	68.79
	Net4	611	88.05	62.17	64.49
	Net5	886	69.86	58.47	60.41
Beam 2	Net3	206	87.38	66.61	71.15
	Net4	419	94.27	65.40	66.65
	Net5	602	82.89	60.45	61.35
Beam 5	Net3	160	87.50	67.5	71.38
	Net4	368	92.66	67.18	69.4
	Net5	546	81.50	61.43	63.25
Beam 9	Net3	153	91.50	68.66	71.69
	Net4	360	93.89	67.83	70.64
	Net5	552	80.62	61.64	62.98

Table 4: Evaluation of the proposed constrained-inference procedure.

We train our two best baseline models (Net1,2) using a highly competitive sequence-to-sequence architecture for machine translation, GNMT [22] with F1 scores, 86.78 and 87.33, respectively. And, to study wider range of accuracies, we train a simpler architecture (with uni-directional encoders/decoders) with different hyper parameters and obtain nets (Net3-5) with F1 scores, 78.32, 74.53, 67.80, respectively. For all models, we employ Glorot initialization, and basic attention [1]. See Table 3 for a summary of the networks, hyper-parameters, and their performance.

We study the behavior of the constraint-satisfaction method on the five networks. On Net3-5 using various inference procedures for decoding: greedy decoding and beam-search with a beam-size of 2, 5, and 9 (resp. beam2, beam5, beam9), and also on the GNMT models Net1,2 with greedy decoding.

We report the results in Table 4. 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.

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. Over 30 experiments with different baseline models and different beam search strategies, our approach satisfies constraints in a way that improves the quality of the parses, except one case where F1 score decresaed by 0.04 point. (as compared to employing post-processing)

Note that an improvement on the failure-set guarantees an improvement on the entire test-set since our method produces the exact same outputs as the baseline for examples that do not initially violate any constraints. Consequently, for example, the GNMT network improves (Net2) on the failure-set from 73.54 to 79.68 F1, resulting in an overall improvement from 86.56 to 87.57 F1 (entire test-set).

Genre	Failure rate (%)	Conversion rate (%)	F1 (before) F1 (before)	F1 (after) F1 (after)
BC	19.3	98.8	56.4	59.0
BN	11.7	98.1	63.2	68.8
PT	9.8	97.8	71.4	75.8
WB	17.6	95.3	62.0	63.2
TC	10.1	86.2	56.9	57.6

Table 5: Evaluation of GNMT parser on out-of-genre data. EvalB F1 scores are for the *failure set*.

Avg. Disagreement-rate (before) (%)	Avg. Disagreement rate (after) (%)	F1 (before)	F1 (after)	F1 (before) (Whole set)	F1 (after) (Whole set)
44.65	24.85	48.07	59.73	84.44	85.67

Table 6: Evaluation of the proposed constrained-inference procedure for SRL. Avg. disagreement rate and Eval F1 scores for *failure set* (first 4 columns) and Eval F1 score for whole set (last 2 columns).

In Table 1 we provide an example data-case that shows how our algorithm solves the initially violated shift-reduce parse output. For simplicity we omit the phrase-types and display only on the shift (`s`), reduce (`t`) and stop reducing commands (`!`), and color them red if there is an error. The algorithm satisfies the constraint in just 12 iterations, and this results in a perfectly correct parse. What is interesting about this example is that the original network commits a parsing mistake early in the output sequence. This type of error is problematic for a naive decoder that greedily enforces constraints at each time-step. The reason is that the early mistake does not create a constraint violation until it is too late, at which point errors have already propagated to future time-steps and the greedy decoder must shift and reduce the last token into the current tree, creating additional spurious parse structures. In contrast, our method treats the constraints holistically, and uses it to correct the error made at the beginning of the parse. See Table 2 for a comparison.

We also measure how many iterations of our algorithm it takes to convert the examples that have constraint-violations. Across all conditions, it takes 5–7 steps to convert 25% of the outputs, 6–20 steps to convert 50%, 15–57 steps to convert 80%, and 55–84 steps to convert 95%.

Out of domain parsing Earlier, we study our algorithm on a wide range of networks with different performance characteristics in order to mimic the performances on out of domain (or genre data). Now we further investigate the performance on actual out of domain (genre) data, by using the multi-genre OntoNotes corpus.

In particular, we train the best performing model from above, GNMT, on the Wall Street Journal portion of OntoNotes and then test on the additional genres provided in the OntoNotes Tree Bank corpus: broadcast conversation (BC), broadcast news (BN) newswire (NW), weblogs (WB), pivot corpus (PT), and telephone conversation (TC). The F1 on the within-genre data (test set of PTB) is 85.03, but the F1 on these genres is much lower, ranging from the mid-forties on broadcast conversation (46.2–78.5 depending on the subcategory) to the low-eighties on broadcast news (68.3–81.3. depending on the subcategory). Indeed, we find that overall the F1 accuracy is lower and in some cases, like weblogs, the failure rate is much higher (17.6% for WB *vs.* 11.9% for WSJ). Following the same experimental protocol as on the PTB data, we produce the results in Table 5 (aggregating over all subcategories in each genre). We see that across all genres, the algorithm has high conversion rates (sometimes close to 100%), and that in each case, enforcing the constraints improves the F1.

5.2 Semantic Role Labeling

We now turn to a different task and network: semantic role-labeling. Semantic role labeling (SRL) is a separate but complementary task to syntactic parsing. While syntactic parsing focuses on identifying relatively deep syntax tree structures, SRL focuses on identifying shallow semantic information about phrases. We ask the question: *can our algortithm succesfully incorporate side-information during inference via constraints into an existing state-of-the-art SRL network?* The constraints we

consider in this experiment enforce that the output of the SRL network must agree with the spans implied by the true syntactic parse of the sentence.

We employ the state-of-the-art AllenNLP semantic role labeling network with ELMO embeddings, which is essentially a multi-layer highway bi-LSTM that produces BIO output predictions for each input token [8, 13]. For data we employ OntoNotes, which has labels for both SRL and syntactic parsing [19]. We evaluate our model on the test-set (29k examples), out of which consistent parse information is available for 82.39% examples (we only include side-information for this subset).

We repeat our same experimental procedure as before, but for this network and task. From Table 6, we see that the algorithm is able to satisfy the constraints in 82.39% of the test examples, and that this boost the overall F1 measure by 1.23 point over the state-of-the-art AllenNLP network [14] that does not incorporate the constraints (they report 84.6 F1, we obtain a similar 84.4 with their network, and achieve 85.67 after enforcing constraints with our inference). Further, to address (Q1) we measure the *avg. disagreement rate* (i.e., the ratio of predicted spans in a sentence that disagree with the spans implied by the true syntactic parse of the sentence). To address (Q2) we evaluate our method on the *failure set* (i.e., the set of sentences for which disagreement rate is nonzero) and measure our method's avg. disagreement rate. Finally, to address (Q3), we evaluate the quality (F1) of the output predictions on the network's *failure-set* both before and after applying our method. From Table 6, we can see that by using our constrained-inference procedure the avg. disagreement rate goes down from 44.65% to 24.85% on the failure-set which results in an improvement from 48.07 to 59.73 F1 on failure-set. These improvements are similar to those we observe in the parsing task, and provide additional evidence for answering Q1-3 favorably.

As an actual example of how the constraints help, consider the sentence: "it is really like this, just look at the bus sign." in which we must tag the two-argument verb "is" as the predicate, "it" as its first argument and the prepositional phrase"like this" as its second argument. The original network incorrectly predicts "really like this" as the second argument. However, the parse tree (a) has no such span because "really" is part of the verb phrase "is really like this" and (b) does have the phrase "like this." Thus enforcing the constraint that the output spans only obey spans implied by the parse-tree causes the network to correctly predict the second argument as "like this."

5.3 Simple Transduction Experiment

In our final experiment we focus on a simple sequence transduction task in which we find that despite learning the training data perfectly, the network fails to learn the constraint in a way that generalizes to the test set.

A transducer $T : \mathcal{L}_S \rightarrow \mathcal{L}_T$ is a function from a source language to a target language. In our experiment, we employ a known T to generate input/output training examples and train a sequence-to-sequence network to learn T on that data. Crucially, there are hard constraints on the output (for example, the output must belong to $\mathcal{L}_T$, and other problem-specific constraints that may in general depend on the input sentence). If the network is unable to learn all the constraints from data, then our method is applicable and we can evaluate its performance.

For our task, we choose a simple transducer, similar to those studied in recent work [7]. The source language $\mathcal{L}_S$ is $(\texttt{az}|\texttt{bz})^\star$ and the target language $\mathcal{L}_T$ is $(\texttt{aaa}|\texttt{zb})^\star$. The transducer is defined to map occurrences of az in the source string to aaa in the target string, and occurrences of bz in the source string to zb in the target string. For example, $T(\texttt{bzazbz}) \mapsto \texttt{zbaaazb}$. The training set comprises 1934 sequences of length 2–20 and the test set contain sentences of lengths 21-24. We employ shorter sentences for training to require generalization to longer sentences at test time.

We employ a thirty-two hidden unit single-layered, attention-less, sequence-to-sequence long short-term memory (LSTM) in which the decoder LSTM inputs the final encoder state at each decoder time-step. We train the network for 1000 epochs using RMSProp to maximize the likelihood of the output (decoder) sequences in the training set. The network achieves perfect train accuracy while learning the rules of the target grammar $\mathcal{L}_T$ perfectly, even on the test-set. However, the network fails to learn the input-specific constraint that the number of a's in the output should be three times the number of a's in the input. This illustrates how a network might rote-memorize constraints rather than learn the rule in a way that generalizes. Thus, enforcing constraints at test-time is important. To satisfy constraints, we employ our method with a constraint loss g, a length-normalized quadratic $(3x_a - y_a)^2/(m+n)$ that is zero when the number of a's in the output (y_a) is exactly three times

the number in the input (x_a). Our method achieves a conversion rate of 65.2% after 100 iterations, while also improving the accuracy on the failure-set from 75.2% to 82.4%. This synthetic experiment provides additional evidence in support of Q2 and Q3, on a simpler small-capacity network.

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 three tasks including sequence-to-sequence parsing and found that it could satisfy up to 98% of the constraints. Accuracy in each of the three tasks was improved by respecting 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] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *CoRR, arXiv preprint arXiv:1409.0473*, 2014.

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

[3] 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.

[4] 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.

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

[6] Marcus Mitchell et al. Treebank-3 ldc99t42 web download. In *Philidelphia: Linguistic Data Consortium*, 1999.

[7] Edward Grefenstette, Karl Moritz Hermann, Mustafa Suleyman, and Phil Blunsom. Learning to transduce with unbounded memory. In *Neural Information Processing Systems (NIPS)*, 2015.

[8] Luheng He, Kenton Lee, Mike Lewis, and Luke S. Zettlemoyer. Deep semantic role labeling: What works and what's next. In *ACL*, 2017.

[9] 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.

[10] 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.

[11] 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.

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

[13] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. *CoRR*, abs/1802.05365, 2018.

[14] Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*, 2018.

[15] 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.

[16] 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.

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

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

[19] Ralph Weischedel, Sameer Pradhan, Lance Ramshaw, Jess Kaufman, Michelle Franchini, Mohammed El Bachouti, Nianwen Xue, Martha Palmer, Jena D. Hwang, Claire Bonial, Jinho Choi, Aous Mansouri, Maha Foster, Abdel aaati Hawwary, Mitchell Marcus, Ann Taylor, Crag Greenberg, Eduard Hovy, Robert Belvin, and Ann Houston. Ontonotes release 5.0. In *LDC Catalog*, 2012.

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

[21] 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.

[22] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR, arXiv preprint arXiv:1609.08144*, 2016.