Gradient-based Inference for Networks with Output Constraints

Jay Yoon Lee [*] [†] **Sanket Vaibhav Mehta** [†] **Michael Wick**[*] [‡] **Jean-Baptiste Tristan** [‡] **Jaime Carbonell** [†]

Abstract

Practitioners apply neural networks to increasingly complex problems in natural language processing, such as syntactic parsing and semantic role labeling 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* (GBI): for each input at test-time, we nudge continuous model weights until the network's unconstrained inference procedure generates an output that satisfies the constraints. We study the efficacy of GBI on three tasks with hard constraints: semantic role labeling, syntactic parsing, and sequence transduction. 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 (seq2seq) network (Cho et al. 2014; Sutskever, Vinyals, and Le 2014; Kumar et al. 2016) to perform a structured prediction task such as syntactic constituency parsing (Vinyals et al. 2015). We would like to apply this trained network to novel, unseen examples, but still require that the network's outputs obey an appropriate set of problem specific hard-constraints; for example, that the output sequence encodes a valid parse tree. Enforcing these constraints is important because downstream 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 volume is limited. Although in some cases, the outputs of state-of-the-art (SOTA) systems mostly obey the constraints for the test-set of the data on which they are tuned, in other cases they do not. In practice, the quality of neural networks 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.

This raises the question: how should we enforce hard constraints on the outputs of a neural network? We could perform expensive combinatorial discrete search over a large output space, 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 *gradient-based inference* (GBI), 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 (Blei, Bagnell, and McCallum 2002), but does so in a way to satisfy the constraints.

We study GBI on three tasks: semantic role labeling (SRL), syntactic constituency parsing and a synthetic sequence transduction problem and find that the algorithm performs favorably on all three tasks. In outline, our contributions on this paper are:

1. Propose a novel Gradient-Based Inference framework.

[*] corresponding authors: jaylee@cs.cmu.edu, michael.wick@oracle.com

[†] Carnegie Mellon University, Pittsburgh, PA

[‡] Oracle Labs, Burlington, MA.

2. Verify that GBI performs well on various applications, thus providing strong evidence for the generality of the method.
3. Examine GBI across wide range of reference model performances and report its consistency.
4. Show that GBI also perform well on out-of-domain data.

For all the tasks, we find that GBI 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 truth-conveying side-information via constraints, improving SOTA network [1] by 1.03 F1 (Peters et al. 2018). This improvement happens to surpass a A* algorithm for incorporating constraints while also being robust, in a way that A* is not, to cases for which the side constraints are inconsistent with the labeled ground-truth.

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 (Koo et al. 2010; Rush et al. 2010; Rush and Collins 2012), 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 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. We are thus interested in the following optimization problem:

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

Simple 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 $g(\mathbf{y}, \mathcal{L}^{\mathbf{x}}) \rightarrow \mathbb{R}_+$ that measures a loss between output $\mathbf{y}$ and a grammar $\mathcal{L}^{\mathbf{x}}$ such that $g(\mathbf{y}, \mathcal{L}^{\mathbf{x}}) = 0$ if and only if there are no grammatical errors in $\mathbf{y}$. That is, $g(\mathbf{y}, \mathcal{L}^{\mathbf{x}}) = 0$ 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 (Lyon 1974). We could then express the constraints as an equality constraint and minimize the Lagrangian:

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

However, this leads to optimization difficulties because there is just a single dual variable for our global constraint, resulting intractable problem and thus leading to brute-force trial and error search.

Instead, we might circumvent these issues if we optimize over a model parameters 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, network's weights can 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 W_λ to stay close to the originally learned weights W. The objective function is as follows:

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

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.,

[1] Since our submission, the previous SOTA (Peters et al. 2018) in SRL on which we apply our technique has been advanced by 1.7 F1 points (Ouchi, Shindo, and Matsumoto 2018). However, this is a training time improvement which is orthogonal to our work.

greedy decoding, beam-search, or Viterbi decoding) 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 (Williams 1992): the decoder outputs as actions and the constraint-loss as a negative reward. However, GBI does not try to reduce *expected* reward and terminates upon discovery of an output that satisfies all constraints. Furthermore, GBI also works on sequence-tagging problem, SRL (Section 4.1), where next output does not depend on the current output, which is far from REINFORCE setting.

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

3 Applications

There are multiple applications that involve hard-constraints and we provide two illustrative examples that we later employ as case-studies in our experiments: SRL and syntactic parsing. The former exemplifies a case in which external knowledge encoded as hard constraints conveys beneficial side information to the original task of interest while the latter studies a case in which hard constraints are inherent to the task of interest. Finally, we briefly mention sequence transduction as framework in which constraints may arise. Of course, constraints may in general arise for a variety of different reasons, depending on the situation. We provide an example-based case studies per application in Appendix A, B.

3.1 Semantic Role Labeling

As a first illustrative example, consider SRL. SRL focuses on identifying shallow semantic information about phrases. For example, in the sentence "it is really like this, just look at the bus sign" the goal is to tag the two arguments given "is" as the verb predicate: "it" as its first argument and the prepositional phrase "like this" as its second argument. Traditionally SRL is addressed as a sequence labeling problem, in which the input is the sequence of tokens and the output are BIO-encoded class labels representing both the regimentation of tokens into contiguous segments and their semantic roles.

Note that the parse tree for the sentence might provide constraints that could assist with the SRL task. In particular, each node of the parse tree represents a contiguous segment of tokens that could be a candidate for a semantic role. Therefore, we can include as side-information constraints that force the BIO-encoded class labeling to produce segments of text that each agree with some segment of text expressed by a node in the parse tree. [2] To continue with our example, the original SRL sequence-labeling might incorrectly label "really like this" as the second argument rather than "like this." Since according to the parse tree "really" is part of the verb phrase, thus while the tree contains the spans "is really like this" and "like this" it does not contain the span "really like this." The hope is that enforcing the BIO labeling to agree with the actual parse spans would benefit SRL. Based on the experiments, this is indeed the case, and our hypothetical example is actually a real data-case from our experiments, which we describe later. The $g(\mathbf{y}, \mathcal{L}^x)$ for SRL factorizes into per-span constraints g_i. For ith span s_i, if s_i is consistent with any node in the parse tree, $g_i(s_i, \mathcal{L}^x) = 0$, otherwise $g_i(s_i, \mathcal{L}^x) = 1/n_{s_i}$ where n_{s_i} is defined as the number of tokens in s_i. Overall, $\Psi(\mathbf{x}, \hat{\mathbf{y}}, W_\lambda) g(\hat{\mathbf{y}}, \mathcal{L}^x) = \sum_{i=1}^{k} g(s_i, \mathcal{L}^x) \Psi(\mathbf{x}, s_i, W_\lambda)$ where k is number of spans on output $\hat{\mathbf{y}}$.

3.2 Syntactic parsing

As a second illustrative example, consider a 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. Syntactic parsing is a separate but complementary task to SRL. While SRL focuses on semantic information, syntactic parsing focuses on identifying relatively deep syntax tree structures. One way to model the problem with neural networks is to linearize the representation of the parse tree and then employ the familiar seq2seq model (Vinyals et al. 2015). 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 standard parts of speech as 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

[2] The ground-truth parse spans do not always agree with the SRL spans, leading to imperfect side information.

$m_{\mathbf{x}}$, 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. (Appendix C.2)

As a minor remark, note that other encodings of trees, such as bracketing (of which the Penn Tree Bank's S-expressions are an example), are more commonly used as output representations for seq2seq parsing (*ibid*). However, the shift-reduce representation described in the foregoing paragraphs is isomorphic to the bracketing representations and as we get similar model performance to single seq2seq mode on the same data (*ibid.*), we chose the former representation to facilitate constraint analysis. Although output representations sometimes matter, for example, BIO vs BILOU encoding of sequence labelings, the difference is usually minor (Ratinov and Roth 2009), and breakthroughs in sequence labeling have been perennially advanced under both representations. Thus, for now, we embrace the shift reduce representation as a legitimate alternative to bracketing, *pari passu*.

3.3 Synthetic sequence transduction

Finally, although not a specific application per se, we also consider sequence transduction as it provides a framework conducive to studying simple artificial languages with appropriately designed properties. A sequence transducer $T : \mathcal{L}_S \rightarrow \mathcal{L}_T$ is a function from a source sequence to a target sequence. As done in previous work, we consider a known T to generate input/output training examples and train a seq2seq network to learn T on that data (Grefenstette et al. 2015). The constraint is simply that the output must belong to $\mathcal{L}_T$ and also respect problem-specific conditions that may arise from the application of T on the input sentence. We study a simple case in Section 4.3.

4 Experiments

In this section we study our algorithm on three different tasks: SRL, syntactic constituency parsing and a synthetic sequence transduction task. All three tasks require hard constraints, but they play a different role in each. In the sequence transduction tasks they force the output to belong to a particular input-dependent regular expression, in SRL, constraints provide side-information about possible true-spans and in syntactic parsing, constraints ensure that the outputs encode valid trees. While the SRL task involves more traditional recurrent neural networks that have exactly one output per input token, the parsing and transduction tasks provide an opportunity to study the algorithm on various seq2seq networks .

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 perfect, can GBI actually enforce the constraints (Q3) does GBI enforce constraints without compromising the quality of the network's output. To more thoroughly investigate Q2 and Q3, we also consider: (Q4) is the behavior of the method sensitive to the reference network performance, and (Q5) does GBI also work on out-of-domain dataset. Q3 is particularly important because we adjust the weights of the network at test-time and this may lead to unexpected behavior. Q5 deals with our original motivation of using structured prediction to enhance performance on the out-of-domain data.

To address various proposed questions, we define some terminologies to measure how well the model is doing in terms of constraints. 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 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.

4.1 Semantic Role Labeling

We employ the AllenNLP (Gardner et al. 2017) SRL network with ELMo embeddings, which is essentially a multi-layer highway bi-LSTM that produces BIO output predictions for each input token (Peters et al. 2018). For data we use OntoNotes v5.0, which has ground-truth for both SRL and syntactic parsing (Pradhan et al. 2013). We evaluate GBI on the test-set (25.6k examples), out of which consistent parse information is available for 81.25% examples (we only include side-information in terms of constraints for this subset).

We repeat the same experimental procedure over multiple networks, SRL-X, with varying amounts (X%) of training dataset. From Table 1, we see that GBI is able to convert 42.25 % of failure set, and this boosts the overall F1 measure by 1.23 point over the SOTA network (SRL-100) that does not incorporate the constraints (they report 84.6 F1, we obtain a similar 84.4 F1 with their network, and achieve 85.63 after enforcing constraints with our inference). Further, to address (Q1) we measure the sentence-level *failure rate* as well as span-level *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 and Exact Match) of the output predictions on the network's *failure-set* both before and after applying our method. From Table 1, we can see that by applying GBI on SRL-100, the avg. disagreement rate on failure set goes down from 44.85% to 24.92% which results in an improvement of 11.7 F1 and 19.90% in terms of exact match on the same set. These improvements answers Q1-3 favorably.

To enforce constraints during inference, He et al. proposed to employ constrained-A* decoding. For the sake of fair comparison with GBI, we consider A* decoding, as used in (He et al. 2017) and report results for SRL-X networks. We see from Table 1, that GBI procedure consistently out-performs

Network	Failure rate(%)	Inference	Conv rate(%)	Failure set						Test set	
				Average (%) Disagreement		F1		Exact Match (%)		F1	
				before	after	before	after	before	after	before	after
SRL-100	9.82	GBI	42.25	44.85	24.92	48.00	59.70 (+11.7)	0.0	19.90	84.40	85.63 (+1.23)
		A*	40.40		33.91		48.83 (+0.83)		13.79		84.51 (+0.11)
SRL-70	10.54	GBI	46.22	45.54	23.02	47.81	59.37 (+11.56)	0.0	19.57	83.55	84.83 (+1.28)
		A*	44.42		32.32		50.49 (+2.68)		16.12		83.90 (+0.35)
SRL-40	11.06	GBI	47.89	45.71	22.42	46.53	58.83 (+12.3)	0.0	19.45	82.57	84.03 (+1.46)
		A*	44.74		32.17		46.53 (+2.88)		15.15		82.98 (+0.41)
SRL-10	14.15	GBI	44.28	47.14	24.88	44.19	54.78 (+10.59)	0.0	15.28	78.56	80.18 (+1.62)
		A*	43.66		32.80		45.93 (+1.74)		12.28		78.87 (+0.31)
SRL-1	21.90	GBI	52.85	50.38	21.45	37.90	49.00 (+11.10)	0.0	12.83	67.28	69.97 (+2.69)
		A*	48.96		30.28		41.59 (+3.69)		11.25		67.97 (+0.69)

Table 1: Comparison of the GBI vs. A* inference procedure for SRL. We report avg. disagreement rate, F1-scores and exact match for *failure set* (columns 5-10) and F1-score for whole test set (last 2 columns). Also we report performances on wide range of reference models SRL-X, where X denotes % of dataset used for training. We employ Viterbi decoding as a base inference strategy (before) and apply GBI (after) in combination with Viterbi.

name	F1		hyper-parameters			data (%)
	BS-9	greedy	hidden	layers	dropout	
Net1	87.58	87.31	128	3	0.5	100%
Net2	86.63	86.54	128	3	0.2	100%
Net3	81.26	78.32	172	3	no	100%
Net4	78.14	74.53	128	3	no	75%
Net5	71.54	67.80	128	3	no	25%

Table 2: Parsing Networks with various performances (BS-9 means beam size 9). Net1,2 are trained on GNMT seq2seq model whereas Net3-5 are trained on lower-resource and simpler seq2seq model to test GBI on wide range of model performances.

Net	Failure (n/2415)	Conv rate	F1 (Failure set)		F1 (whole test)	
			before	after	before	after
Net1	187	93.58	71.49	77.04	87.31	87.93
Net2	287	89.20	73.54	79.68	86.54	87.57

Table 3: Evaluation of the GBI on syntactic parsing using GNMT seq2seq. Note that GBI without beam search performs higher than BS-9 on Table 2.

A* decoding on all evaluation metrics, thus, proving superiority of our approach.

4.2 Syntactic parsing

We now turn to a different task and network: syntactic constituency 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) (Marcus et al. 1999). 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.

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

Net	Infer method	Failure (n/2415)	Conv rate	F1 (Failure set)	
				before	after
Net3	Greedy	317	79.81	65.62	68.79 (+3.14)
	Beam 2	206	87.38	66.61	71.15 (+4.54)
	Beam 5	160	87.50	67.5	71.38 (+3.88)
	Beam 9	153	91.50	68.66	71.69 (+3.03)
Net4	Greedy	611	88.05	62.17	64.49 (+2.32)
	Beam 2	419	94.27	65.40	66.65 (+1.25)
	Beam 5	368	92.66	67.18	69.4 (+2.22)
	Beam 9	360	93.89	67.83	70.64 (+2.81)
Net5	Greedy	886	69.86	58.47	60.41 (+1.94)
	Beam 2	602	82.89	60.45	61.35 (+0.90)
	Beam 5	546	81.50	61.43	63.25 (+1.82)
	Beam 9	552	80.62	61.64	62.98 (+1.34)

Table 4: Evaluation of the GBI on simpler, low-resource seq2seq networks. In here, we also evaluate whether GBI can be used in combination with different inference techniques: greedy and beam search of various width.

In each experiment, we learn a seq2seq 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).

In order to study our algorithm on a wide range of accuracy regimes (section 4.4), 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.

We train our two best baseline models (Net1,2) using a highly competitive seq2seq architecture for machine translation, GNMT (Wu et al. 2016) with F1 scores, 86.78 and 87.33, respectively. And, to study wider range of accuracies, we train a simpler architecture with different hyper parameters and obtain nets (Net3-5). For all models, we employ Glorot initialization, and basic attention (Bahdanau, Cho, and Bengio 2014). See Table 2 for a summary of the networks, hyper-parameters, and their performance.

We report the behavior of the constraint-satisfaction

method on Table 3 for Net1-2, and on Table 4 for Net3-5. Across all the experimental conditions (Table 3, 4), the conversion rates are high, often above 80 and sometimes above 90 supporting Q2. Note that beam search alone can also increase constraint satisfaction with conversion rate going as high as 51.74% (164/317) in case of Net3 with beam size 9. However, as the quality of the model increases, the conversion rate became minuscule; in case of Net1,2 the conversion rate was less than 14% with beam 9; Net1 converted 26 out of 187 and Net2 converted 1 out of 287 instances from failure set.

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 F1 is only defined on valid trees, we employ heuristic post-processing to ensure all outputs are valid.

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 86.54 to 87.57 F1 (entire test-set). These improvements are similar to those we observe in the SRL task, and provide additional evidence for answering Q1-3 favorably.

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%.

4.3 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.

For our task, we choose a simple transducer, similar to those studied in recent work (Grefenstette et al. 2015). The source language $\mathcal{L}_S$ is `(az|bz)*` and the target language $\mathcal{L}_T$ is `(aaa|zb)*`. 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(`bzazbz`) $\mapsto$ `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 32 hidden unit single-layered, attention-less, seq2seq LSTM in which the decoder LSTM inputs the final encoder state at each decoder time-step. 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 GBI 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) with m,n denoting input, output, respectively. GBI 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.

4.4 GBI on wide range of reference models

The foregoing experimental results provide evidence that GBI is a viable method for enforcing constraints. However, we hitherto study GBI on high quality reference networks such as SRL-100. To further bolster our conclusions, we now direct our investigation towards lower quality networks to understand GBI's viability under a broader quality spectrum. We ask, how sensitive is GBI to the reference network's performance (Q4)? To this end, we train poorer quality networks by restricting the amount of available training data or employing simpler architectures.

For *SRL*, we simulate low-resource models by limiting the training data portion to 1%, 10%, 40%, and 70% resulting in F1 score range of 67.28-83.55. Similarly, for *syntactic parsing*, we train additional low-quality models `N`et3-5 with a simpler uni-directional encoders/decoders, and on different training data portion of 25%, 75%, and 100%. (Table 2). We evaluate GBI on each of them in Table 1, 4 and find further evidence in support of favorable answers to Q2 (satisfying constraints) and Q3 (improving F1 accuracy) by favorably answering Q4. Moreover, while not reported fully due to page limits, we examined both tasks over 20 experiments with different baseline networks in combination to different inference strategies, and we found GBI favorable in all but one case (but by just 0.04 compared to employing post-processing).

We also study whether GBI is compatible with better underlying discrete search algorithms, in particular beam search for seq2seq. As we seen in column 2 of Table 4, that although beam-search improves the F1 score and reduces the percentage of violating constraints, GBI further improves over beam-search when using the latter in the inner-loop as the decoding procedure. In conclusion, improving the underlying inference procedure has the effect of decreasing the number of violating outputs, but GBI is still very much effective on this increasingly small set, despite it intuitively representing more difficult cases that even eludes constraint satisfaction on beam search inference.

4.5 Experiments on out-of-domain data

Previously, we saw how GBI performs well even when the underlying network is of lower quality. We now investigate GBI on actual out-of-domain data for which the model quality can suffer. For SRL, we train a SOTA network with ELMo embedding on the NewsWire (NW) section of the OntoNotes v5.0 English PropBank corpus and then test on the other genres provided in the corpus: BC, BN, PT, TC, WB. The failure rate on the within genre data (test set of NW) is 18.10%. We can see from Table 5, the failure rate for the NW trained SRL network in general is higher for out-of-genre data with highest being 26.86% for BC(vs. 18.10% NW). Further, by

Genre	*Syntactic Parsing*				*SRL*			
	Failure rate (%)	Conversion rate (%)	F1 on failure set		Failure rate (%)	Conversion rate (%)	F1 on failure set	
			before	after			before	after
Broadcast Conversation (BC)	19.3	98.8	56.4	59.0 (+2.6)	26.86	53.88	39.72	52.4 (+12.68)
Broadcast News (BN)	11.7	98.1	63.2	68.8 (+5.6)	18.51	55.19	39.28	50.58 (+11.3)
Pivot Corpus (PT)	9.8	97.8	71.4	75.8 (+4.4)	10.01	62.34	47.19	63.69 (+16.5)
Telephone Conversation (TC)	10.1	86.2	56.9	57.6. (+0.7)	19.09	54.62	47.7	58.04 (+10.34)
Weblogs (WB)	17.6	95.3	62.0	63.2 (+1.2)	20.32	44.13	47.6	57.39 (+9.39)

Table 5: Evaluation of syntactic parser and SRL system on out-of-domain data. F1 scores are reported on the *failure set*. SRL model was trained on NW and the syntactic parser was trained on WSJ Section on OntoNote v5.0. Except PT, which is new and old Testament, all failure rate on out-domain data is higher than that of in-domain (11.9% for parsing and 18.1% for SRL) as suspected. The table shows that GBI can be successfully applied to resolve performance degradation on out-of-domain data.

enforcing constraints, we see significant gains on failure set in terms of F1 score across all genres (ranging from 9.39-16.5 F1), thus, providing additional evidences for answering Q5.

As we did for SRL, we train a GMNT seq2seq model on the WSJ NW section in OntoNotes v5.0 Treebank [4] which shares the same genre classification with PropBank. The F1 on the within-genre data (test set of WSJ) is 85.03, but the F1 on these genres is much lower, ranging from the mid-forties on BC (46.2–78.5 depending on the subcategory) to the low-eighties on BN (68.3–81.3. depending on the subcategory). Indeed, we find that overall the F1 is lower and in some cases, like WB, 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 report 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. Again, we find support for Q2, Q3 and Q5.

4.6 Robustness and Runtime analysis

We perform extra analysis on robustness and runtime for SRL in comparison with competing constrained-A* decoding method. In Appendix D, we introduce case study of noisy constraints where A* performs worse than the baseline whereas GBI still improves it significantly, thus, showcasing its robustness to noisy constraints.

For runtime, GBI is slightly faster than A* while the difference in terms of the runtime is unclear on smaller evaluation sets (see Appendix E). In the case study of noisy constraints, GBI is similar in runtime to A*, but the runtime comparison is not important in the noisy constraint setting as A* with noisy constraints simply hurts whereas GBI shows comparable gains with the noise-free constraint setting.

5 Related work

Recent work has considered applying neural networks to structured prediction; for example, structured prediction energy networks (SPENs) (Belanger and McCallum 2016). 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 training-time for simple classifications (Hu et al. 2016). Our work instead focuses on enforcing constraints at inference-time. More specifically, for SRL, previous work for enforcing syntactic and SRL specific constraints have focused on constrained A* decoding (He et al. 2017) or integer linear programming (Punyakanok, Roth, and Yih 2008). For parsing, previous work in enforcing hard constraints has focused on post-processing (Vinyals et al. 2015) or building them into the decoder via sampling (Dyer et al. 2016) or search constraints (Wiseman and Rush 2016).

Finally, as previously mentioned, our method highly resembles dual decomposition and more generally Lagrangian relaxation for structured prediction (Koo et al. 2010; Rush et al. 2010; Rush and Collins 2012). 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.

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 to coax weights towards well-formed outputs. We evaluated the algorithm on three tasks including SOTA SRL, seq2seq parsing and found that GBI can successfully convert failure sets while also boosting the task performance. Accuracy in each of the three tasks was improved by respecting constraints. Additionally, for SRL, we employed GBI on top of a model trained with similar constraint enforcing loss as GBI's (Mehta*, Lee*, and Carbonell 2018), and observe that the test-time optimization of GBI still significantly improves the model output whereas A* does not. We believe this is because GBI enables to search proximity of the provided model weights while theoretical analysis on this hypothesis is left as a future work.

[4] The PTB (40k instances) and OntoNotes (30k instances) coverage of WSJ are slightly different.

References

Bahdanau, D.; Cho, K.; and Bengio, Y. 2014. Neural machine translation by jointly learning to align and translate. *CoRR, arXiv preprint arXiv:1409.0473*.

Belanger, D., and McCallum, A. 2016. Structured prediction energy networks. In *International Conference on Machine Learning*.

Blei, D. M.; Bagnell, A.; and McCallum, A. K. 2002. Learning with scope, with application to information extraction and classification. In *Uncertainty in Artificial Intelligence (UAI)*.

Cho, K.; Van Merriënboer, B.; Gülçehre, Ç.; Bahdanau, D.; Bougares, F.; Schwenk, H.; and Bengio, Y. 2014. 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)*, 1724–1734. Association for Computational Linguistics.

Dyer, C.; Kuncoro, A.; Ballesteros, M.; and Smith, N. A. 2016. Recurrent neural network grammars. In *NAACL-HLT*, 199–209.

Gardner, M.; Grus, J.; Neumann, M.; Tafjord, O.; Dasigi, P.; Liu, N. F.; Peters, M.; Schmitz, M.; and Zettlemoyer, L. S. 2017. Allennlp: A deep semantic natural language processing platform.

Grefenstette, E.; Hermann, K. M.; Suleyman, M.; and Blunsom, P. 2015. Learning to transduce with unbounded memory. In *Neural Information Processing Systems (NIPS)*.

He, L.; Lee, K.; Lewis, M.; and Zettlemoyer, L. S. 2017. Deep semantic role labeling: What works and what's next. In *ACL*.

Hu, Z.; Ma, X.; Liu, Z.; Hovy, E.; and Xing, E. P. 2016. Harnessing deep neural networks with logical rules. In *Association for Computational Linguistics (ACL)*.

Koo, T.; Rush, A. M.; Collins, M.; Jaakkola, T.; and Sontag, D. 2010. Dual decomposition for parsing with non-projective head automata. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, 1288–1298. Association for Computational Linguistics.

Kumar, A.; Irsoy, O.; Ondruska, P.; Iyyer, M.; Bradbury, J.; Gulrajani, I.; Zhong, V.; Paulus, R.; and Socher, R. 2016. Ask me anything: Dynamic memory networks for natural language processing. *Machine Learning* 1378–1387.

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

Marcus, M. P.; Santorini, B.; Marcinkiewicz, M. A.; and Taylor, A. 1999. Treebank-3 ldc99t42 web download. In *Philidelphia: Linguistic Data Consortium*.

Mehta*, S. V.; Lee*, J. Y.; and Carbonell, J. 2018. Towards semi-supervised learning for deep semantic role labeling. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, 4958–4963.

Ouchi, H.; Shindo, H.; and Matsumoto, Y. 2018. A span selection model for semantic role labeling. In *EMNLP*, 1630–1642. Association for Computational Linguistics.

Peters, M. E.; Neumann, M.; Iyyer, M.; Gardner, M.; Clark, C.; Lee, K.; and Zettlemoyer, L. 2018. Deep contextualized word representations. In *Proc. of NAACL*.

Pradhan, S.; Moschitti, A.; Xue, N.; Ng, H. T.; Björkelund, A.; Uryupina, O.; Zhang, Y.; and Zhong, Z. 2013. Towards robust linguistic analysis using ontonotes. In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning*, 143–152.

Punyakanok, V.; Roth, D.; and Yih, W.-t. 2008. The importance of syntactic parsing and inference in semantic role labeling. *Computational Linguistics* 34(2):257–287.

Ratinov, L., and Roth, D. 2009. Design challenges and misconceptions in named entity recognition. In *Computational Natural Language Learning (CoNNL)*.

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

Rush, A. M.; Sontag, D.; Collins, M.; and Jaakkola, T. 2010. On dual decomposition and linear programming relaxations for natural language processing. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, 1–11. Association for Computational Linguistics.

Sutskever, I.; Vinyals, O.; and Le, Q. V. 2014. Sequence to sequence learning with neural networks. In *Neural Information Processing Systems (NIPS)*.

Vinyals, O.; Kaiser, L.; Koo, T.; Petrov, S.; Sutskever, I.; and Hinton, G. 2015. Grammar as a foreign language. In *NIPS*.

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

Wiseman, S., and Rush, A. M. 2016. Sequence-to-sequence learning as beam-search optimization. In *Empirical Methods in Natural Language Processing*, 1296–1306.

Wu, Y.; Schuster, M.; Chen, Z.; Le, Q. V.; Norouzi, M.; Macherey, W.; Krikun, M.; Cao, Y.; Gao, Q.; Macherey, K.; et al. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR, arXiv preprint arXiv:1609.08144*.

Appendix

A GBI vs. Constrained decoding

In Table 8 of Appendix B, 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 of constituency parsing and display only on the shift (`s`), reduce (`r`) 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 6 for a comparison of how the methods fix the constraints. Specifically, the constraint violation is that there were not enough shift and reduce commands to account for all the tokens in the sentence. Rather than fixing the constraint by inserting these extra commands at the end of the sequence as the greedy decoder must do, GBI inserts them at the beginning of the sequence where the initial mistake was made, thereby correcting the initial mistake. Moreover, this correction propagates to a mistake made later in the sequence (viz., the the sequence of three reduces after the four shifts) and fixes them too. This example provides evidence that GBI can indeed enforce constraints holistically and that doing so improves the output in a global sense.

⟨“ 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 6: 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.

B Example-based case study

⟨“ it is really like this , just look at the bus signs . ”⟩ ⟶ `B-ARG1 B-V B-ARGM-ADV B-ARG2 I-ARG2 O O ··· O`

iteration	output	loss	accuracy
0	B-ARG1 B-V B-ARG2 I-ARG2 I-ARG2 O O O O O O O O	0.012	50.0%
6	B-ARG1 B-V B-ARG2 I-ARG2 I-ARG2 O O O O O O O O	0.049	50.0%
7	B-ARG1 B-V B-ARGM-ADV B-ARG2 I-ARG2 O O O O O O O O	0.00	100.0%

Table 7: A semantic role labeling example for which the method successfully enforces syntactic constraints. The initial output has inconsistent span for token "really like this". Enforcing the constraint not only corrects the number of agreeing spans, but also changes the semantic role "B-ARG2" to "B-ARGM-ADV" and "I-ARG2" to "B-ARG2"..

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

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

Table 8: 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.

azazbzazbzbzazbzbzbzbzbz ⟶ aaaaaazbaaazbzbaaazbzbzbzbzb

iteration	output	loss	accuracy
0	aaaaaazbaaazbaaazbzbzbzbaaazb	0.2472	66.7
1	aaaaaazbaaazbaaazbzbzbzbaaazb	0.2467	66.7
2	aaaaaazbaaazbaaazbzbzbzbaaazb	0.2462	66.7
3	aaaaaazbaaazbzbaaazbzbzbzbzb	0.0	100.0

Table 9: A sequence transduction example for which enforcing the constraints improves accuracy. Red indicates errors.

C Constraint functions

Here we define the specific constraint loss function $g(\mathbf{y}, \mathcal{L}^x)$ for each task. Note that a common theme is that we normalize the constraint loss by the length of the sequence so that it does not grow unbounded with sequence size. We recommend this normalization as we found that it generally improves performance.

C.1 Semantic Role labeling

The $g(\mathbf{y}, \mathcal{L}^x)$ for SRL factorizes into per-span constraints g_i. For ith span s_i, if s_i is consistent with any node in the parse tree, $g_i(s_i, \mathcal{L}^x) = 0$, otherwise $g_i(s_i, \mathcal{L}^x) = 1/n_{s_i}$ where n_{s_i} is defined as the number of tokens in s_i. Overall,

$$\Psi(\mathbf{x}, \hat{\mathbf{y}}, W_\lambda) g(\hat{\mathbf{y}}, \mathcal{L}^x) = \sum_{i=1}^{k} g(s_i, \mathcal{L}^x) \Psi(\mathbf{x}, s_i, W_\lambda)$$

where k is number of spans on output $\hat{\mathbf{y}}$.

More precisely, for a span s to be "consistent with a parse node" we mean the following. Let $t_i \in T$ be a node in the parse tree T and let s_i^t be the span of text implied by the descendents of the node t_i. Let $S^T = \{s_i^t\}$ be the set of spans implied by all nodes in the parse tree T. We say that a span of text s is consistent with the parse tree T if and only if $s \in S^T$.

C.2 Syntactic Parsing

Let $m_{\mathbf{x}}$, n be number of input, output tokens, respectively, $\text{ct}_{i=1}^{n}(b(i))$ be the function that counts the number of times proposition $b(i)$ is true for $i = 1, \ldots, n$. Now, define the following loss

$$g(\mathbf{y}, \mathcal{L}^x) = \frac{1}{m_{\mathbf{x}} + n} \left\{ |m_{\mathbf{x}} - \text{ct}_{i=0}^{n}(y_i = s)| + \sum_{i}^{n} \max\left(0, \text{ct}_{j=0}^{i}(y_j = r) - \text{ct}_{j=0}^{i}(y_j \in \{s, !\})\right) \right\}.$$

The first term provides loss when the number or shifts equals the number of input tokens, the second term provides loss when attempting to reduce an empty stack and the third term provides loss when the number of reduces is not sufficient to attach every lexical item to the tree.

C.3 Transduction

For the aforementioned transducer we chose for the experiment, $\mathcal{L}_S$ is `(az|bz)*` and the target language $\mathcal{L}_T$ is `(aaa|zb)*`. 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 the provided transduction function, the number of `a`'s in the output should be three times the number of `a`'s in the input. To express this constraint, we define following constraint loss g, a length-normalized quadratic

$$g(\mathbf{y}, \mathcal{L}^x) = (3x_a - y_a)^2/(m + n)$$

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) with m,n denoting input length, output length, respectively.

D Analyzing the behavior of different inference procedures in the presence of noisy constraints

Table 10 reports the performance of GBI and A* in the presence of noisy constraints. We can see that the overall performance (F1-score) for A* drops drastically (-6.92) in the presence of noisy constraints while we still see gains with GBI ($+0.47$). We further analyze the improvement of GBI by looking at the precision and recall scores individually. We see that recall drops slightly for GBI which suggests that noisy constraints does inhibit predicting actual argument spans. On the other hand, we see that precision goes up significantly. After analyzing predicted argument spans, we noticed that GBI prefers to predict no argument spans instead of incorrect spans in the presence of noisy constraints which leads to an increase in precision. Thus, GBI provides flexibility in terms of strictness with enforcing constraints which makes it robust to noisy constraints. On the other hand, constrained-A* decoding algorithm is too strict when it comes to enforcing noisy constraints resulting in significant drop both in precision and recall.

Decoding	Precision	Recall	F1-score	Exact Match (%)
Viterbi	84.03	84.78	84.40	69.37
	Noisy constraints			
A*	78.13 (-5.90)	76.85 (-7.93)	77.48 (-6.92)	58.30 (-11.70)
GBI	85.51 (+1.48)	84.25 (-0.53)	**84.87** (**+0.47**)	68.45 (-0.92)
	Noise-free constraints			
A*	84.19 (+0.16)	84.83 (+0.05)	84.51 (+0.11)	70.52 (+1.15)
GBI	85.39 (+1.36)	85.88 (+1.10)	**85.63** (**+1.23**)	71.04 (+1.67)

Table 10: Comparison of different inference procedures: Viterbi, A*(He et al. 2017) and GBI with noisy and noise-free constraints. Note that the (+/-) F1 are reported w.r.t Viterbi decoding on the same column.

E Analyzing the runtime of different inference procedures with varying dataset sizes and genres

Network	Genre(s)	No. of examples	Failure rate (%)	Inference time (approx. mins)		
				Viterbi	GBI	A*
SRL-100	All	25.6k	9.82	109	288	377
SRL-NW	BC	4.9k	26.86	23	110	117
	BN	3.9K	18.51	18	64	100
	PT	2.8k	10.01	8	19	15
	TC	2.2k	19.01	5	23	20
	WB	2.3k	20.32	12	49	69

Table 11: Comparison of runtime for difference inference procedures in noise-free constraint setting: Viterbi, A*(He et al. 2017) and GBI. For SRL-100 refer Table 1 and SRL-NW is a model trained on NW genre.

Table 11 reports the runtime for different inference procedures with varying dataset sizes. In general, we observe that GBI tends to be faster than A*, especially when the dataset is large enough. One exception is the BC domain where GBI is just slightly faster than A*. We hypothesize it might be due to the difficulty of the constraint violation due to its failure rate being higher than usual. GBI will search for the correct output for longer time (more iterations) if it is harder to find the solution.

Also note that we explicitly set max epochs for GBI after which it will stop iterating to avoid pathological cases. In our SRL experiments, we have set the max epochs to be 10 (GBI-10). To study its scalability, we ran GBI with max epochs set to 30 (GBI-30). In terms of the runtime, we do see a difference with GBI-30 taking 556 mins as opposed to GBI-10 taking 288 mins. However, we also get significantly better results with GBI-30 in terms overall F1 (+0.34), F1 on failure set (+3.4), exact match (+4.35%), and conversion rate (+11.24%) compared with GBI-10 at the cost of runtime increment.