ENFORCING CONSTRAINTS ON OUTPUTS WITH UNCONSTRAINED INFERENCE

Jay Yoon Lee
Carnegie Mellon University
Pittsburgh, PA
jaylee@cs.cmu.edu

Michael Wick, Jean-Baptiste Tristan
Oracle Labs
Burlington, MA
{michael.wick,jean.baptiste.tristan}@oracle.com

ABSTRACT

Increasingly, practitioners apply neural networks to complex problems in natural language processing (NLP), such as syntactic parsing, that have rich output structures. Many such applications require deterministic constraints on the output values; for example, requiring that the sequential outputs encode a valid tree. While hidden units might capture such properties, the network is not always able to learn them from the training data alone, and practitioners must then resort to post-processing. In this paper, we present an inference method for neural networks that enforces deterministic constraints on outputs without performing post-processing or expensive discrete search over the feasible space. Instead, for each input, we nudge the continuous weights until the network's unconstrained inference procedure generates an output that satisfies the constraints. We find that our method reduces the number of violating outputs by up to 81%, while improving accuracy.

1 INTRODUCTION

Many neural networks have discrete-valued output units that correspond to an inference or prediction about an input. Often, a problem might involve multiple discrete outputs. Unlike multiclass classification, which associates a single discrete output with each input, so called *structured prediction* problems associate multiple outputs with each input. For example, in multi-label classification, instead of predicting a single relevant class pertaining to the image or sentence, we must predict all relevant classes: the image contains a dog, a tree, and a sky. In sequence prediction problems, the discrete outputs might be a sequence of words or symbols that must form a coherent translation of a source language sentence (Cho et al., 2014; Sutskever et al., 2014), description of an image (Vinyals et al., 2015b), answer to a question (Kumar et al., 2016), or a parse-tree for an input sentence (Vinyals et al., 2015a). Crucially, in structured prediction, the output values are interdependent. Even though neural networks usually predict outputs independently or sequentially (one output at a time), the hidden units allow them to successfully capture many dependencies.

Sometimes, the outputs must obey hard constraints. For example, in sequence labeling with `BILOU` encoding, a 'begin' marker `B` cannot immediately follow an 'inside' marker `I` (Ratinov & Roth, 2009). In clustering, pairwise binary decisions must obey transitivity so that they yield a valid equivalence class relation over the data points (McCallum & Wellner, 2005; Wick et al., 2006; 2008). In syntactic/dependency parsing, the output sequence must encode a valid parse tree (McDonald & Pereira, 2006; Vinyals et al., 2015a; Dyer et al., 2016). In formal language generation or neural compilers the output must belong to a context free language or compile (Reed & de Freitas, 2016). In dual decomposition approaches to joint inference, copies of variables must satisfy equality constraints (Koo et al., 2010; Rush et al., 2010; Rush & Collins, 2012). Finally, in some ensemble methods, the outputs of multiple conditionally independent classifiers must reach a consensus on the output class. Indeed, there are a tremendous number of problems that require hard constraints on the outputs. Unlike softer dependencies, violating a hard-constraint is often unacceptable because the output of the network would not "type-check" causing problems for downstream components. Unfortunately in practice, networks are not always able to exactly learn constraints from the training data alone.

As a motivating example, consider a sequence-to-sequence network that inputs a sentence and outputs a sequence of "shift-reduce" commands that describe the sentence's parse tree. Briefly, the shift-

reduce commands control a parsing algorithm by indicating how and when to use its stack. Each command controls whether to shift (`s`) a token onto the stack, reduce (`r`) the top of the stack into a parent tree node, or push (`!`) the current reduction back onto the stack.

To be successful, the network must generate commands that imply a valid tree over the entire input sentence. However, the decoder outputs just a single command at a time, producing some outputs that are not globally-consistent, valid shift-reduce programs. Indeed, the output may not have enough shifts to include every input token in the tree or may attempt to reduce when the stack is empty. For example, the following input sentence *" So it 's a very mixed bag . "* comprises ten space-delimited tokens (the quotations are part of the input), but our unconstrained sequence-to-sequence network outputs an invalid sequence with only nine shifts `ssr!sr!ssssrrr!rr!ssrrrrrr!`. We must introduce another `shift` so the last token is pushed onto the stack and issue another `reduce` so it is inserted into the tree.

We could attempt to fix the output with post-processing, but where is the right place to insert these commands in the sequence? There are $406 = \text{choose}(29, 2)$ candidate locations. Further complicating our post-processing dilemma is the fact that the output contains several other errors that are seemingly unrelated to the constraint. Instead, we could attempt to fix the problem with a more sophisticated decoder, but this is difficult because the decoder outputs a single character at each time-step and our constraints are global, limiting corrections to the end of the sequence when it is too late to rectify an earlier decision. A beam search is less myopic, but in practice most of the network's output mass is peaked on the best output token, resulting in little improvement.

In this paper, we propose an inference method for neural networks that enforces output constraints without employing combinatorial discrete search. The idea is to modify some (or all) of the weights for each instance at test-time, iteratively nudging them, until the network's efficient unconstrained inference procedure produces a valid output. We achieve this by expressing the hard constraints as an optimization problem over the continuous weights and employ back-propagation to change them. *Prima facie*, back-propagation is doomed because the constraint loss is necessarily a function of the argmax that produced the discrete values. However, we circumvent this problem by optimizing over the energy of the violating outputs instead. Since the weights directly determine the output through the energy, we are able to manipulate the unconstrained inference procedure to produce the desired result. Much like scoped-learning, the algorithm customizes the weights for each example at test-time (Blei et al., 2002), but does so in a way to satisfy the constraints.

When applied to the above example, our method removes enough energy mass from the invalid output space in only twelve steps, allowing unconstrained decoding to produce a valid output sequence:

`ssr!sr!ssssrrr!rr!ssrrrrrr!`	(initial output)
`s`ssr!ssssrr!srrr!rr!ssrrrrrr!	(rectified output after 12 steps)

Interestingly, the network generates an additional `s` command at the beginning of the sequence while also producing a cascade of error correction in later time steps: the new output now satisfies the constraints and is a perfectly correct parse. Of course, enforcing constraints does not always lead to an improvement in accuracy, but we find that often it does in practice, especially for a well-trained network. We find that our method is able to completely satisfy constraints in up to 81% of the outputs.

2 BACKGROUND

Consider a neural network that generates a variable length output vector $\mathbf{y} = \{y_i\}_1^{n_y}$ from a variable length input vector $\mathbf{x} = \{x_i\}_1^{m_x}$. For example, in image classification, the input vector encodes fixed multi-dimensional tensor of pixel intensities and the output vector comprises just a single element corresponding to the discrete class label. In sequence-to-sequence, the input might be a variable length vector of French tokens, and the output would be a variable length vector of its English translation. It is sometimes convenient to think of the network as a function from input to output

$$f(\mathbf{x}; W) \mapsto \mathbf{y} \tag{1}$$

However, for the purpose of exposition, we separate the neural network into a real-valued model (negative energy function) that scores the compatibility of the outputs (given the weights and input) and an inference procedure that searches for high scoring outputs.

For the model, let y_i be a discrete output from an output unit and let $\psi(y_i; \mathbf{x}, W)$ be its corresponding real-valued log-space activation score (e.g., the log of the softmax for locally normalized models or simply a linear activation value for globally normalized models). Define the negative energy Ψ over a collection of output values $\mathbf{y}$ as an exponentiated sum of log-space activation scores

$$\Psi(\mathbf{y}; \mathbf{x}, W) = \exp\left(\sum_i \psi(y_i; \mathbf{x}, W)\right) \tag{2}$$

Then, inference is the problem of finding the values of the outputs $\mathbf{y}$ that maximize the negative energy given fixed inputs $\mathbf{x}$ and weights W. Thus, we can rewrite the neural network as the function:

$$f(\mathbf{x}; W) \mapsto \operatorname*{argmax}_{\mathbf{y}} \Psi(\mathbf{y}; \mathbf{x}, W) \tag{3}$$

The purpose of separating the model from the inference procedure is so we can later formalize our optimization problem. We emphasize that this formulation is consistent with existing neural networks. Indeed, inference in feed-forward networks is a single feed-forward pass from inputs to outputs. When the outputs only depend on each other through hidden states that only depend on earlier layers of the network, feed-forward inference is exact in the sense that it finds the optimum of Equation 3. For recurrent neural networks (RNNs), each output depends on hidden states that are functions of previous output values. However, we can still think of the usual procedure that produces the highest scoring output at each time step as a local greedy approximation to global inference; of course, the procedure can optionally be improved with a beam.

3 CONSTRAINED INFERENCE FOR NEURAL NETWORKS

A major advantage of neural networks is that once trained, inference is extremely efficient. However, constraints can render inference intractable due to discrete search. Our goal is take advantage of the fact that unconstrained inference is inexpensive and design a constrained inference algorithm that exploits such a procedure as a black box. Our method iteratively adjusts the weights for each test-time input, concentrating the probability mass on the feasible region so that unconstrained inference becomes increasingly likely to generate an output that satisfies the constraints.

In this work, we focus on constraints that require the outputs to belong to an input-dependent context-free language $\mathcal{L}^{\mathbf{x}}$ (CFL). The idea is to treat the output space of the neural network as the terminal symbols, and devise the appropriate production rules and non-terminals to express constraints on them. An advantage of employing CFLs over other formalisms such as first order logic (FOL) is that CFLs are intuitive for expressing constraints on the outputs, especially for language models and sequence-to-sequence networks. For example, when modeling Python or Java code, it is easy to express many of the desired programming language's constraints using a CFL, but cumbersome in FOL. Indeed, CFLs are an expressive class of languages.

To motivate our algorithm, we begin with the ideal optimization problem and argue that unlike for linear models with local constraints, the resulting Lagrangian is not well suited for globally constrained inference in neural networks. We ultimately settle on an alternative objective function that reasonably models our constrained inference problem. Although our algorithm lacks the theoretical guarantees enjoyed by classic relaxation algorithms we nevertheless find it works well in practice.

Consider the following constrained inference problem for neural networks

$$\begin{aligned} \max_{\mathbf{y}} \quad & \Psi(\mathbf{x}, \mathbf{y}, W) \\ \text{s. t.} \quad & \mathbf{y} \in \mathcal{L}^{\mathbf{x}} \end{aligned} \tag{4}$$

Naively enforcing the constraint requires combinatorial discrete search, which is intractable in general. Instead, we prefer a smooth optimization problem with meaningful gradients to guide the search.

With this in mind, let $g(\mathbf{y}, \mathcal{L}) \mapsto r$ for $r \in \mathbb{R}_+$ be a function 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 a large class of CFLs, g could be the *least errors count* function (Lyon, 1974) or a weighted version thereof. We could then express CFL membership as an equality constraint and minimize the Lagrangian

$$\min_{\lambda} \max_{\mathbf{y}} \Psi(\mathbf{x}, \mathbf{y}, W) + \lambda g(\mathbf{y}, \mathcal{L}) \tag{5}$$

However, this dual optimization problem has a major flaw. Our constraints are global and do not necessarily factorize over the individual outputs. Consequently, there is just a single dual variable λ. Optimizing λ does little more than eliminate a single contour of output configurations at a time, resulting in a brute-force trial and error search.

Instead, observe that the network's weights control the negative energy of the output configurations. By properly adjusting the weights, we can affect the outcome of inference by removing mass from invalid outputs. The weights are likely to generalize much better than the single dual variable because in most neural networks, the weights are tied across space (e.g., CNNs) or time (e.g., RNNs). As a result, lowering the negative energy for a single invalid output has the effect of lowering the negative energy for an entire family of invalid outputs, enabling faster search. With this in mind, we introduce an independent copy W_λ of the network's weights W and minimize with respect to these "dual weights" instead of the dual variable. This is powerful because we have effectively introduced an exponential number of "dual variables" (via the energy, which scores each output) that we can easily control via the weights; although similar, the new optimization is no longer equivalent to the original:

$$\min_{W_\lambda} \max_{\mathbf{y}} \Psi(\mathbf{x}, \mathbf{y}, W) + \Psi(\mathbf{x}, \mathbf{y}, W_\lambda) g(\mathbf{y}, \mathcal{L}) \tag{6}$$

While a step in the right direction, the objective still requires combinatorial search because (1) the maximization involves two non-linear neural networks and (2) a greedy decoding algorithm is unable to cope with the global loss g() because the constraints do not factorize over the individual outputs. In contrast the functions involved in classic Lagrangian relaxation methods for NLP have multipliers for each output variable that can be combined with linear models to form a single unified decoding problem for which efficient inference exists (Koo et al., 2010; Rush et al., 2010; Rush & Collins, 2012). Since our non-linear functions and global constraints do not afford us the same ability, we must modify the optimization problem for a final time so that we can employ the network's efficient inference procedure as a black-box. In particular, we (1) remove the negative-energy term that involves the original weights W and compensate with a regularizer that attempts to keep the dual weights W_λ as close to these weights as possible and (2) maximize exclusively over the network parameterized by W_λ. The result is a different optimization problem on which our algorithm is based:

$$\min_{W_\lambda} \left(\Psi(\mathbf{x}, \mathbf{y}, W_\lambda) g(\mathbf{y}, \mathcal{L}^x) + \alpha \|W - W_\lambda\|_2 \;\middle|\; \mathbf{y} = \operatorname*{argmax}_{\mathbf{y}} \Psi(\mathbf{x}, \mathbf{y}, W_\lambda) \right) \tag{7}$$

Informally, our algorithm alternates the maximization (by running efficient unconstrained inference) and minimization (by performing SGD) until it produces a feasible output or it exceeds a maximum number of iterations. For each test-example, we re-initialize the dual weights to the trained weights to ensure the network does not deviate too far from the trained network. More precisely see Algorithm 1.

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
  $\nabla \leftarrow \frac{\partial}{\partial W_\lambda} \Psi(\mathbf{x}, \mathbf{y}, W_\lambda) g(\mathbf{y}, \mathcal{L}^{\mathbf{x}}) + \alpha \|W - W_\lambda\|_2$ #compute constraint loss
  $W_\lambda \leftarrow W_\lambda - \eta \nabla$ #update instance-specific weights with SGD or a variant thereof
end while

4 Application to parsing

Consider the structured prediction problem of syntactic parsing in which the goal is to input a sentence comprising a sequence of tokens and output a tree describing the grammatical parse of the sentence. One way to model the problem with neural networks is to linearize the representation of the parse tree and then employ the familiar sequence-to-sequence model (Vinyals et al., 2015a).

Let us suppose we linearize the tree using a sequence of shift (`s`) and reduce (`r`, `r!`) commands that control an implicit shift reduce parser. Intuitively, these commands describe the exact instructions for converting the input sentence into a complete parse tree: the interpretation of the symbol `s` is that we

shift an input token onto the stack and the interpretation of the symbol `r` is that we start (or continue) reducing (popping) the top elements of the stack, the interpretation of a third symbol `!` is that we stop reducing and push the reduced result back onto the stack. Thus, given an input sentence and an output sequence of shift-reduce commands, we can deterministically recover the tree by simulating a shift reduce parser. For example, the sequence `ssrr!ssr!rr!rr!` encodes a type-free version of the parse tree `(S (NP the ball) (VP is (NP red)))` for the input sentence "the ball is red" It is easy to recover the tree structure from the input sentence and the output commands by simulating a shift reduce parser, performing one command at a time as prescribed by the classic algorithm.

Note that for output sequences to form a valid tree over the input, the sequence must satisfy a number of constraints. First, the number of shifts must equal the number of input tokens $m_{\mathbf{x}}$, otherwise either the tree would not cover the entire input sentence or the tree would contain spurious terminal symbols. Second, the parser cannot issue a reduce command if there are no items left on the stack. Third, the number of reduces must be sufficient to leave just a single item, the root node, on the stack.

We can express most of these constraints with a CFL

$$\mathcal{L} = \begin{cases} G & \rightarrow sRr! \\ R & \rightarrow sRr \\ R & \rightarrow Rr! \\ R & \rightarrow RR \\ R & \rightarrow \epsilon \end{cases} \tag{8}$$

Intuitively, Rule 1 states that a valid shift-reduce command set must begin with a shift (since stack is initially empty, there is nothing to reduce) and end with a reduce that places the final result on the stack. Rule 2 states that if we do a shift, then we need to reduce the shifted token at some point in the future. Rule 3 states that if we do not shift then we are allowed to reduce only if we also push the result on the stack. Rule 4 allows for multiple subtrees. Rule 5 is the base case.

Note, however, that this grammar is for a general purpose shift-reduce language, but we need to constrain the number of shifts to equal the number of input tokens $m_{\mathbf{x}}$. Since the constraint is a bit verbose to express with production rules, we can instead write the regular language $(s(r!)^\star)^{m_{\mathbf{x}}}(r!)^\star$ where m is the number of elements in $\mathbf{x}$ and intersect it with our CFL.

$$\mathcal{L}^x = \mathcal{L} \cap (s(r!)^\star)^{m_{\mathbf{x}}}(r!)^\star \tag{9}$$

Rather than relying on a general purpose algorithm to compute $g(\mathbf{y}, \mathcal{L}^x)$ that measures the number of grammatical errors, we instead implement it specifically for our language. Let $\text{ct}_{i=1}^n(b(i))$ be the function that counts the number of times proposition $b(i)$ is true. Now, define the following loss

$$g(\mathbf{y}, \mathcal{L}^x) = (m - \underset{i}{\text{ct}}(y_i = s))^2 + \left(\sum_i \underset{j>i}{\text{ct}}(y_j = r) - \underset{j>i}{\text{ct}}(y_j \in \{s, !\}) \right)^2 + \underset{i}{\text{ct}}(y_i = r) - (\underset{i}{\text{ct}}(y_i \in \{s, !\}))^2 \tag{10}$$

The first term measures the amount of violation due to the regular language and the second and third terms measure the amount of violation according to the CFL.

5 Related work

There has been recent work in applying neural networks to structured prediction problems. For example, the recent structured prediction energy networks (SPENS) combines graphical models and neural networks via an energy function defined over the output variables (Belanger & McCallum, 2016). SPENS focuses on soft constraints (via the energy function) and performs inference by relaxing the binary output variables to be continuous and then backpropagating into them. In contrast, our method focuses on hard constraints and we backpropagate into the weights rather than into the outputs directly. We could combine our method with SPENs to handle soft constraints; for example, by back-propagating the output energy into the weights instead of the relaxed outputs themselves.

There has been recent work on applying neural networks to parsing problems that require the ability to handle hard constraints. For example, by employing a sequence-to-sequence network (Vinyals et al., 2015a) or a custom network designed for shift reduce parsing (Dyer et al., 2016). The former requires

task	inference	weights changed (W_λ)	conversion rate	accuracy
azbz	unconstrained	none	0.0%	75.6%
	constrained	all	**65.2%**	82.4%
	constrained	output only	20.9%	77.8%
	constrained	encoder only	58.2%	82.5%
	constrained	decoder only	57.4%	82.3%
sr no types	unconstrained	none	0.0%	84.0%
	constrained	all	**81.8%**	84.4%
sr with types	unconstrained	none	0.0%	87.8%
	constrained	all	**79.2%**	88.3%
	constrained	output only	5.0%	88.1%
	constrained	decoder (top layer)	36.2%	88.2%
	constrained	decoder (all layers)	54.7%	88.3%
	constrained	decoder (top) + attention	38.0%	88.1%
	constrained	decoder (all) + attention	56.5%	88.2%

Table 1: Conversion rates on all three tasks with 100 steps of SGD. Note that satisfying the constraints has no negative affect on accuracy and often has a positive affect.

bzazbzazbzazazbzbzbzbzbz $\longrightarrow$ zbaaazbaaazbaaaaaazbzbzbzbzb

iteration	output	loss	accuracy
0	zbaaazbaaazbaaaaaazbzbzbaaazbzb	0.260	75.0
39	zbaaazbaaazbaaaaaazbzbzbaaazbzb	0.259	75.0
40	zbaaazbaaazbaaaaaazbzbzbaaazb	0.250	80.0
72	zbaaazbaaazbaaaaaazbzbzbaaazb	0.249	80.0
73	zbaaazbaaazbaaaaaazbzbzbzbzb	0.0	100.0

Table 2: An example for which enforcing the constraints improves accuracy. Red indicates errors. The output changes more than once before the constraints are finally enforced. Greedy decoding with constraints might correct this example because the spurious a's are at the end of the sequence.

the output to form a valid parse tree and hence they employ post-processing to ensure this property. The latter satisfies constraints as part of the decoding process by sampling over a combinatorial space. Our approach does not rely on post processing or discrete search.

Another intriguing approach is to distill the hard constraints into the weights at training time using a teacher network (Hu et al., 2016). The method is appealing because it does not require constrained inference or combinatorial search. However, the method must achieve a difficult balance between the loss due to the training data and the loss due to the constraint violations. Further, it would crucially rely on network's ability to generalize the constraints learned on the training data to the testing data.

Finally, our method highly resembles dual decomposition and more generally Lagrangian relaxation for structured prediction (Koo et al., 2010; Rush et al., 2010; Rush & Collins, 2012). In such techniques, it is assumed that a computationally efficient inference algorithm can maximize over a superset of the feasible region (indeed this assumption parallels our exploitation of the fact that unconstrained inference in the neural network is efficient). Then, the method employs gradient descent to gradually concentrate this superset onto the feasible region until the constraints are satisfied. However, for computational reasons, these techniques assume that the constraints factorize over the output and that the functions are linear so that they can be combined into a single model. In contrast, we have a single dual variable so we instead minimize with respect to the weights, which generalize better over the output. Further, we are unable to combine the dual into a single model over which we can do inference because the network is highly non-linear.

6 Experiments

In this section we empirically evaluate our constrained inference procedure on two sequence-to-sequence tasks. The first is a transduction task between two simple languages, which we describe next. The second is the sequence-to-sequence shift-reduce parsing task described in Section 4.

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 3: An example for which enforcing the constraints improves accuracy. Red indicates errors. Note that greedy decoding with constraints would not fix the errors in the middle since errors are made before constraints are violated. In contrast, the proposed method takes the constraints into account in a globall manner, allowing earlier errors to be corrected by future constraint violations.

bzbzbzbzazbzbzazazazazbz⟶zbzbzbzbaaazbzbaaaaaaaaaaaaaaazb

iteration	output	loss	accuracy
0	zbzbzbzbaaazbaaaaaaaaaaaaaazbaaa	0.2954	74.2
4	zbzbzbzbzbaaaaaaaaazbzbaaaaaa	0.0	60.0

Table 4: An example for which enforcing the constraints degrades accuracy. Errors in red.

A transducer $T : \mathcal{L}_1 \rightarrow \mathcal{L}_2$ is a function from a source language to a target language. For the purpose of the experiments T is known and our goal is to learn it from data. We choose a transducer similar to those studied in recent work (Grefenstette et al., 2015). The source language $\mathcal{L}_0$ is `(az|bz)*` and the target language $\mathcal{L}_1$ is `(aaa|zb)*`. The transducer is defined to map `az` to `aaa` and `bz` to `zb`. For example, T(`bzazbz`)↦`zbaaazb`. The training set comprises 1934 sequences of length 2–20 and the test set contain sentences of lengths 21-24. As is common practice, we employ shorter sentences for training to require generalization to longer sentences at test time.

We employ a thirty-two hidden unit single-layered, attentionless, sequence-to-sequence long short-term memory (LSTM) in which the decoder LSTM inputs the final encoder state at each time-step. The encoder and decoder LSTMs each have their own set of weights. 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 output grammar nearly perfectly, even on the test-set. However, despite learning the train-set perfectly, the network fails to learn the input-specific constraint that the number of `a`'s in the output should be three times as the number in the input. We implement a loss for this constraint and evaluate how well our method enforces the constraint at test-time: $g(\mathbf{y}, \mathcal{L}_1^x) = (n+m)^{-1}\left(\left(3\sum_{x_i}\mathbb{I}(x_i = a)\right) - \left(\sum_{y_i}\mathbb{I}(y_i = a)\right)\right)^2$ where $n+m$, the combined intput/output length, normalizes between 0 and 1. For constrained inference we run Algorithm 1 and employ vanilla stochastic gradient descent with a learning rate of 0.05 and no weight decay. We cap the number of iterations at a maximum of 100.

The top section of Table 1 contains the results for this `azbz` task. We use the term converted to refer to a sentence that initially had a constraint-violation, but was later fixed by the constrained-inference procedure. The *conversion rate* is the percentage of such sentences that we convert: on this task, up to two-thirds. We experiment with which subset of the weights is best for satisfying the constraints, finding that it is best to modify them all. We also report accuracy to study an initial concern. Specifically, we had to omit the negative energy of the original weights W from our optimization problem, Equation 7, potentially allowing the network to find a set of dual weights W_λ that happen to satisfy the constraints, but that have poor performance. However, we found this not to be the case. In fact, we report the token-wise accuracy over the examples for which the unconstrained neural network violated constraints and find that on the contrary, accuracy improves. Further, we find the regularizer is unnecessary since the initialization $W_\lambda = W$ ensures the network never drifts too far.

In order to gain a better understanding of the algorithm's behavior, we provide data-cases that highlight both success and failure (Tables 2,3,4). The title of these tables is the input and the desired ground truth output. The rows of the table show the network's output at each iteration (as indicated). The loss column is the constraint loss weighted by the output's energy $\Psi(\mathbf{x}, \mathbf{y}, W_\lambda)g(\mathbf{y}, \mathcal{L}_1^X)$, and the final column is the token-wise accuracy between the output and the ground truth.

⟨“ So it ’s a very mixed bag . ”⟩ ⟶ sssr!ssssrr!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!ssssrr!srrr!rr!ssrrrrrr!	0.0000	100.0%

Table 5: 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.

Table 2 contains an example for which our method successfully satisfies the constraints resulting in perfect accuracy. However, because the constraint violation appears at the end of the string, a greedy decoder that opportunistically enforces constraints on the fly could potentially correct this error. In Table 3 we show a more interesting example for which such a greedy decoder would not be as successful. In particular, the unconstrained network outputs the final aaa too early in the sequence, but the constraint that controls the number of a’s in the output is not violated until the end of the sequence. In contrast, our method takes the constraint into account globally, allowing the network to not only rectify the constraint, but to achieve perfect accuracy on the sentence (in just four gradient updates). Finally, in Table 4, we show an example for which enforcing the constraints hurts the accuracy. The updates causes the network to erroneously change outputs that were actually correct. This can happen if (a) the underlying network is sometimes inaccurate in its output or confidence/probabilities thereon or (b) the gradient steps are too large causing the network to completely leapfrog over the correct solution in a single step. The latter can be avoided by normalizing the constraint loss so it does not grow unbounded with the number of outputs and by erring on the side of a smaller learning rate.

We repeat the same experiment (middle section of Table 1), but on the shift-reduce parsing task described in Section 4. We convert the Wall Street Journal portion of the Penn Tree Bank (PTB) into shift-reduce commands and randomly split into 30k train and 9.2k test examples. We increase the number of hidden units to sixty-four to accommodate the larger input space (50k words) and employ Equation 10 (normalized by sequence length) for the constraint loss. We measure the sequence-aligned token accuracy. Otherwise, we employ the exact same experimental parameters as the azbz task, both for training the LSTM and for our algorithm. We find that our algorithm performs even better on the real-world task, converting over 80% of the violated outputs. We again find that our procedure has no negative impact on accuracy, which in fact improves, but not as substantially as for the azbz task. Table 5 contains a successful example that we had previously highlighted in Section 1. The algorithm satisfies the constraints, and also corrects the remaining output errors.

Finally, we conduct a version of the shift-reduce experiment that includes the phrase types (e.g., noun-phrase (NP)). To accommodate the larger output space (output alphabet size increases to 479), we employ a larger network with 128 hidden units, attention and three-layers. Note that even this more sophisticated network fails to learn the constraints from data and adding layers does not help. The larger network affords us the opportunity to experiment with modifying different subsets of weights for enforcing constraints. As seen in the last section of Table 1, modifying all the weights works best, converting 79.2% of the violating sentences; again without negatively affecting accuracy.

7 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 two sequence to sequence tasks, a toy transducer problem and a real-world shift-reduce parsing problem. We found that the method was able to completely rectify up to 80% of violated outputs when capping the number of iterations at 100. Often, enforcing constraints caused the accuracy to improve, dispelling initial concerns that adjusting the weights at test-time would be treacherous. Our method currently lacks the same theoretical guarantees as classic Lagrangian relaxation methods, so in future work we want to focus on supplemental theory and additional objective functions. We also hope to extend the work to handle soft constraints, for example, as imposed by an external language model.

REFERENCES

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

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.

Kyunghyun Cho, Bart Van Merriënboer, Çalar 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)*, pp. 1724–1734. Association for Computational Linguistics, October 2014.

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

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

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.

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*, pp. 1288–1298. Association for Computational Linguistics, 2010.

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*, pp. 1378–1387, 2016.

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

Andrew McCallum and Ben Wellner. Conditional models of identity uncertainty with applications to noun coreference. In *Neural Information Processing Systems (NIPS)*, 2005.

Ryan McDonald and Fernando Pereira. Learning of approximate dependency parsing algorithms. In *EACL*, 2006.

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

Scott Reed and Nando de Freitas. Neural program interpreters. In *International Conference on Learning Representations (ICLR)*, 2016.

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.

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*, pp. 1–11. Association for Computational Linguistics, 2010.

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

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

Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and tell: A neural image caption generator. In *Computer Vision and Pattern Recognition (CVPR)*, 2015b.

Michael Wick, Aron Culotta, and Andrew McCallum. Learning field compatibilities to extract database records from unstructured text. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, EMNLP '06, pp. 603–611, Stroudsburg, PA, USA, 2006. Association for Computational Linguistics. ISBN 1-932432-73-6.

Michael Wick, Khashayar Rohanimanesh, Andrew McCallum, and AnHai Doan. A discriminative approach to ontology alignment. In *In proceedings of the 14th NTII WS at the conference for Very Large Databases (VLDB)*, 2008.