# Improved Lexically Constrained Decoding for Translation and Monolingual Rewriting

J. Edward Hu Huda Khayrallah Ryan Culkin Patrick Xia
Tongfei Chen Matt Post Benjamin Van Durme
Johns Hopkins University
{edward.hu,huda,rculkin,paxia,tongfei}@jhu.edu; {post,vandurme}@cs.jhu.edu

## Abstract

Lexically-constrained sequence decoding allows for explicit positive or negative phrase-based constraints to be placed on target output strings in generation tasks such as machine translation or monolingual text rewriting. We describe vectorized dynamic beam allocation, which extends work in lexically-constrained decoding to work with batching, leading to a five-fold improvement in throughput when working with positive constraints. Faster decoding enables faster exploration of constraint strategies: we illustrate this via data augmentation experiments with a monolingual rewriter applied to the tasks of natural language inference, question answering and machine translation, showing improvements in all three.

## 1 Introduction

For many natural language generation tasks, we often know word(s) that should (or should not) be in the output sentence. Examples include terminology databases in Machine Translation (MT) (Hokamp and Liu, 2017), names (and generic responses) in dialogue generation (Li et al., 2016; Gu et al., 2016), objects in image captioning (Anderson et al., 2017), and facts in abstractive summarization (See et al., 2017). One approach to enforce hard lexical constraints in the output is to modify the inference procedure to enforce their presence directly (Hokamp and Liu, 2017). These constraints could be either *positive* (a word must appear in the output) or *negative* (a word must be avoided). While negative constraints could be easily enforced by preventing hypotheses with prohibited tokens from entering the beam, placing positive constraints in natural and meaningful ways is less straightforward. We improve upon previous work by vectorizing the dynamic beam allocation (DBA) algorithm from Post and Vilar (2018) and by incorporating *multi-state* tries, which track a subset of nodes at each decoding timestep. These improvements lead to a five-fold speedup in decoding with positive constraints and in some cases better constraint placements (with respect to BLEU).

Post and Vilar (2018) motivated the utility of lexically-constrained decoding in MT for scenarios such as interactive translation and domain adaptation. Translation applications handling large amounts of data will clearly benefit from improvements in speed: the same is true for large-scale data augmentation via rewriting. In this case, a practitioner will ideally explore various task-specific rewriting strategies that may lead to improvements as observed during development, and then incorporate the best strategy into a test-final model. Recently, sentential paraphrasing gained the ability to enforce lexical constraints (Hu et al., 2019), but constrained decoding was still too inefficient to be practical (Hokamp and Liu, 2017) at a large scale. Even with the approach described by Post and Vilar, exploring the space of possible rewriting strategies on a task-specific basis may be overly time consuming: our performance improvements to their algorithm lowers the barrier of entry, where one may more practically experiment with various strategies during development. To illustrate our point, we build an improved monolingual sentential rewriter that can be conditioned on arbitrary positive and negative lexical constraints and use this to augment data for three external NLP tasks with different strategies: Natural Language Inference (NLI), Question Answering (QA) and MT.

Our main contributions are:

- A more efficient and robust approach to lexically-constrained decoding with vectorized DBA and trie representations;
- A trained and freely available lexically-

*Proceedings of NAACL-HLT 2019*, pages 839–850
Minneapolis, Minnesota, June 2 - June 7, 2019.

constrained monolingual rewriter[1] with improvements in both human-judged semantic similarity and fluency over the initial PARABANK rewriter (Hu et al., 2019);

- Monolingual rewriting constraint heuristics for automatic data augmentation leading to improvements on NLI / QA / MT.

## 2 Background

**Constrained decoding** Prior work explored methods to apply lexical constraints to a Neural Machine Translation (NMT) decoder (Hokamp and Liu, 2017; Anderson et al., 2017). However, most of these methods are slow and impractical as they change beam sizes at different time steps, which breaks the optimized computation graph. Post and Vilar (2018) proposed a means of dynamically allocating the slots in a fixed-size beam to ensure that even progress was made in meeting an arbitrary number of constraints provided with the input sentence. However, despite it being their motivation, their approach did not scale to batching, instead they sequentially processed constraints for sentences within the batch.

**Paraphrases and Rewriting** Many works sought to create paraphrases or paraphrastic expressions through existing corpora. For example, DIRT (Lin and Pantel, 2001) extracts paraphrastic expressions from paths in dependency trees. Weisman et al. (2012) explored learning inference relations between verbs in broader scopes (document or corpus level). PPDB (Ganitkevitch et al., 2013) constructs paraphrase pairs by linking words or phrases that share the same translation in another language. PARANMT (Wieting and Gimpel, 2018) and PARABANK (Hu et al., 2019) used back-translation to build a large paraphrase collection from bilingual corpora.

For arbitrary sentence rewriting, Napoles et al. (2016) used statistical machine translation in tandem with PPDB as a black box monolingual sentential rewriter. Mallinson et al. (2017) used a series of NMT model pairs to perform back-translations for monolingual paraphrasing. A similar approach was adopted by PARANMT to create a large paraphrase collection, which is used to train a monolingual sentence rewriter for canonicalization. PARABANK (Hu et al., 2019) extends PARANMT's approach and produced a NMT-based rewriter with the ability to apply lexical constraints to produce multiple paraphrases.

However, Hu et al. (2019) did not: evaluate the rewriter's performance on in-the-wild sentences; explore more sophisticated versions of the rewriter; nor demonstrate its utility on NLP tasks.

**Data augmentation** Data augmentation has been used to improve performance and robustness in deep neural models. In NMT, the most common approach is back-translation, where monolingual text in the target language is translated to create synthetic source sentences (Sennrich et al., 2016). Variants of back-translation target specific words with high prediction loss (Fadaee and Monz, 2018), employed sampling to increase diversity (Edunov et al., 2018), replace rare words (Fadaee et al., 2017), or replace at random (Wang et al., 2018).

Automatic data generation has also been successfully used for community question answering (Chen and Bunescu, 2017), semantic parsing (Jia and Liang, 2016), and task-oriented dialogue (Hou et al., 2018) by generating new data from the training dataset. In contrast, our model is trained on a much larger external corpus and is fixed, independent of the task. Kobayashi (2018) utilized a pre-trained language model for automated data augmentation, though they only consider word-level rewrites and encourage label-preservation, while we paraphrase whole sentences with lexical constraints, independent of a gold label.

Most similar to our experiments, Iyyer et al. (2018) explored *syntactic* paraphrasing for augmentation in sentiment and NLI tasks, extending prior work on PARANMT.

## 3 Improved Constrained Decoding

Lexically-constrained decoding is a modification to beam search that yields decoder outputs honoring user-supplied constraints. These constraints can be provided in the form of: *positive constraints*, which specify that certain tokens or token sequences must be present in the output; or *negative constraints*, which specify token or token sequences that must *not* be generated. Take positive constraints for example, in translating the sentence *Das stimmt einfach nicht* to English, the user can specify the constraint "not the case" to (presumably) get the output *That's just not the case* instead of model-preferred output *That's just not*

[1] http://nlp.jhu.edu/parabank

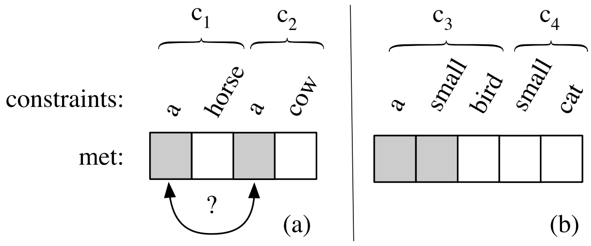

Figure 1: Two corner cases that arise when using an array implementation to track constraints. The gray boxes denote constraints that have been generated by the currently-tracked hypothesis. In (a), the decoder has to guess which constraint it is generating since they share a prefix. In (b), the decoder may start tracking constraint $c_3$ only to generate $c_4$ without realizing it.

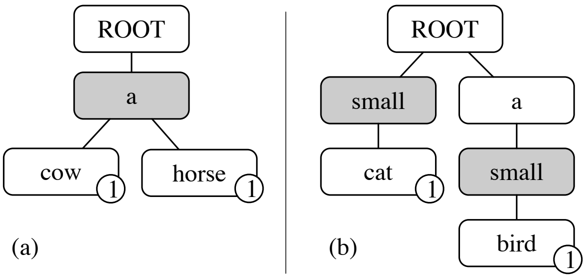

Figure 2: The trie representation solves both problems depicted in Figure 1. The constraint tracker no longer has to predict the constraint to track (a), and it can track multiple constraints by using multiple states (b). Counters are added to end nodes to denote how many times each constraint must be generated.

*true*. While there is no guarantee that the decoder will use the constraints in a sensible way, constraints are often well-placed empirically.

The implementation of positively constrained decoding comprises two key pieces: tracking which of the supplied constraints each hypothesis has already generated, and ensuring progress through the constraints by dynamically allocating the beam to hypotheses that have generated different numbers of them. We describe an improvement to each of these over Post and Vilar (2018), which includes: (1) a vectorized approach to beam allocation that works with batch decoding; and (2) the use of tries for recording constraint state, and thereby offsetting certain corner cases.

These contributions allow the decoder to find much better placement of constraints (as evidenced by an almost 2 point BLEU score increase) and to increase throughput for batch decoding.

Here, we assume the reader is familiar with beam decoding for NMT, the details of which are provided by Post and Vilar (2018).

### 3.1 Tracking constraints with tries

The implementation by Post and Vilar used a flat one-dimension array listing the word indexes of all positive constraints (duplicates allowed). A parallel array was used to mark which words in this list were non-final words in a sequence, so that progress could be tracked through sequences of tokens. Progress through the constraints was tracked by maintaining, for each slot in the beam, a third array, which marked which of the constraints had already been generated by that hypothesis.

However, this leads to corner cases when two constraints $c_1$ and $c_2$ share a subsequence. The first case occurs when two constraints have an identical prefix. Consider the constraints in Fig. 1(a) when translating the French sentence *une vache et un cheval*. The array-based implementation has to choose which constraint to generate when it has only generated the first word of the English translation, *a cow and a horse*. Suppose it chooses constraint $c_1$, *a horse*. If the subsequent step generates *cow* instead, the constraint tracking for the phrase *a horse* will be marked as incomplete and reset, and the decoder will not realize that it has satisfied a different constraint.

A second corner case arises when a constraint $c_4$ is a non-prefix substring of a constraint $c_3$. In this situation, the decoder may begin generating the longer constraint, only to generate the shorter one, without realizing it. For example, consider a target sentence that should be *a small cat saw a small bird*, with constraints *a small bird* and *small cat* (Figure 1b). When generating the first word, *a*, the decoder begins tracking $c_3$. It continues by adding to this hypothesis the second word, *small*. However, suppose it then extends this hypothesis with *cat*. It will abort tracking of $c_3$, and not realize that it completed $c_4$.

A more natural representation that addresses these corner cases is to organize constraints that haven't yet been generated into a trie. Nodes in the trie that represent the ends of constraints are augmented with a counter that indicates how many times that constraint must be generated.[2] Each time a constraint is completed, the number is decremented, and nodes of the trie can be trimmed when they lead only to paths ending in zero counts.

[2] Because one constraint can be a subsequence of another, some interior nodes will also have these counts.

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| sentno | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| unmet | 0 | 0 | 1 | 1 | 1 | 1 | 2 | 2 | 2 | 3 | 4 | 4 | 5 | 5 | 5 |
| **step** | **0** | **1** | **0** | **1** | **2** | **3** | **0** | **1** | **2** | **0** | **0** | **1** | **0** | **1** | **2** |
| score | 1.23 | 1.07 | 1.13 | 1.01 | 0.89 | 0.87 | 1.08 | 0.92 | 0.77 | 0.83 | 0.69 | 0.54 | 0.71 | 0.46 | 0.41 |
| hypo # | 4 | 2 | 1 | 0 | 6 | 3 | 2 | 0 | 1 | 5 | 1 | 0 | 1 | 0 | 0 |
| vocab ID | 4241 | 923 | 135 | 712 | 2122 | 14 | 805 | 4209 | 212 | 324 | 712 | 441 | 5243 | 853 | 5071 |

Figure 3: The hypothesis matrix during vectorized dynamic beam allocation (VDBA). Each column denotes a candidate hypothesis, each of which contains sufficient information to identify it. Hypotheses are sorted by sentence number, number of unmet constraints, and sequence scores. The "unmet" row indicates how many constraints each hypothesis has not yet generated. After sorting by sentence ID and "unmet", we assign monotonically increasing numbers across columns that share (sentno, unmet) values (the "step" row). We can now sort by the "step" row to ensure that the beam for a sentence contains hypotheses having met each number of constraints. We populate the next beam by taking the first $k$ hypotheses (in gray) after sorting the matrix by **step**. Here, $k = 7$.

Finally, in order to address the second corner case, we track multiple states in each constraint trie.

In summary, we represent all the constraints as a compact trie. Each hypothesis in the decoder beam has its version of the trie. The set of active states in each hypothesis' trie tracks all suffixes of the target words that match against the constraint trie. When a constraint is generated, its counter is decremented and zero paths are pruned.

### 3.2 Negative constraints

Negative constraints are used to denote words and phrases that the decoder must not generate. Blocking single-word negative constraints can be done by setting their costs to infinity before doing top-$k$ selection, at each time step. These negative constraints are also represented in a trie, although it is slightly different, because it does not have a counter and never needs to be pruned. Instead, it records at each node the list of word IDs that end phrases for which the current word is the penultimate. We similarly track all suffixes of the current hypothesis' target word string that match the negative constraint trie. At each time-step, we block the generation of active phrases by setting to infinity all word IDs marked in the current node (if any). This includes the root node, which handles single-word constraints. Each state is then extended by following outgoing arcs, if present, or else resetting them to the root state.

### 3.3 Vectorized Dynamic Beam Allocation

Post and Vilar (2018) describe an algorithm that divides the beam among hypotheses that have generated different numbers of positive constraints. For a beam of size $k$ and with $C$ positive constraint tokens, the algorithm produces a set of candidate extensions of the $k$ hypotheses from the beam. They assemble these extensions from three sources: (a) the top-$k$ best-scoring tokens across all hypotheses in the beam (without respect to constraints); (b) the set of tokens that advance the constraint trie for each hypothesis; and (c) the best-scoring extension of each hypothesis.[3] After constructing this candidate list, they whittle it down to a list of size $k$ and use it to construct the beam at the next time step. This way, the algorithm ensures that portions of the beam are devoted to candidates having met different number of constraints, and thereby that progress is made towards meeting all the constraints as decoding proceeds.

However, their implementation used a procedural approach which is incompatible with batching; that is, constraints for input segments within a batch are processed sequentially, so increasing the batch size does not produce any speed gains. We replace the procedural approach with a vectorized one, which uses GPU operations to quickly assemble the list of candidates and allocate them to the beam such that we do benefit from batching.

A sketch of our algorithm follows. We assemble candidates from the same three sources described above. Sets (a) and (c) already use fast GPU operations. These operations can be done efficiently even batch-wise. Set (b) is less amenable to vectorization, but can be assembled by querying each hypothesis for its unmet constraints. We now use a sorting-based algorithm to parallelize the divvying

[3] The difference between (a) and (c) is that the items constituting (a) typically come from different extensions of the top hypotheses, whereas (c) ensures that one extension of each hypothesis is in the candidates list.

| Constraints | Speed (sent/sec) | | | | BLEU | |
|---|---|---|---|---|---|---|
| | batch size 1 | | batch size 20 | | | |
| | Baseline | +MST,VDBA | Baseline | +MST,VDBA | Baseline | +MST,VDBA |
| none | 4.78 | 4.84 | **17.92** | 17.51 | 40.9 | 40.9 |
| phr4 | 0.75 | 0.89 | 0.93 | **4.71** | 47.8 | **49.2** |
| rand3 | 0.78 | 0.80 | 0.87 | **4.59** | 43.0 | **43.2** |

Table 1: Comparison between the baseline implementation (SOCKEYE 1.18.57, commit 59180f3) and our approach with multi-state tries (MST) and vectorized Dynamic Beam Allocation (VDBA) in placing different constraints randomly extracted from the reference. All runs use a beam size of 10. The pruning threshold was set to 0 for no constraints ("none") and 30 for "phr4" and "rand3". Decoding speed is measured on an NVIDIA GTX 1080 Ti in sentences per second (the higher the better). Output quality is measured using SacreBLEU (the higher the better).

up of the beam to hypotheses having met different numbers of constraints.

We do this by assembling a matrix with all the candidates for all sentences in the batch ( Fig. 3). This matrix contains a column for each candidate, including the sentence number, the number of unmet constraints for that hypothesis, its sequence score, the hypothesis it extends, and the next vocabulary ID. With this matrix, we can quickly select the $k$ hypothesis extensions for the next timestep using a multi-key sort. The first key is the sentence number. Next, it is the number of unmet constraints in each hypothesis. We then make use of a "step" row, which assigns increasing indices *within* each group of hypotheses with the same number of unmet constraints. Sorting on this row as the third key establishes a round-robin assignment of the $k$-sized beam to items having met different numbers of constraints. In the end, we select the top $k$ items (in the example, $k = 7$ and the selected columns are in gray).

### 3.4 Evaluation

We use SOCKEYE (Hieber et al., 2017)[4] for our evaluations. We trained a 6-layer German–English Transformer using the default settings on the WMT'18 training data and the `newstest2018` test set for evaluation (Bojar et al., 2018). Following Post and Vilar (2018), we compare decoding results in an unconstrained setting and with two sets of positive constraints: "rand3", which selects 3 random words from the reference, and "phr4", which selects a single 4-word phrase. We report decoding speed (in sentences per second) and BLEU score (Papineni et al., 2002), as measured by SacreBLEU (Post, 2018). The results are shown in Table 1.

Our approach is faster than existing approaches when decoding with positive constraints and produces the same or higher BLEU scores, which we take as a sign of more fluent and natural hypotheses under constraints. Without batching, there is no speedup, but at a batch size of 20, we see roughly a 5× speedup.

## 4 Improved Monolingual Rewriter

Inspired by the approach described in PARABANK (Hu et al., 2019), we trained a more powerful English monolingual rewriter by using a multi-head self-attention NMT model, Transformer (Vaswani et al., 2017). We used a 6-layer encoder and decoder with a model size of 512 and 8 attention heads. The encoder and decoder embeddings share the same weight. Unlike PARABANK, which trained a rewriter on a subset of 50M paraphrase pairs out of its collection, we trained on *all* of the paraphrastic pairs in PARABANK originated from CzEng[5] that: (1) have a regression score over 0.50; (2) only consist of ASCII characters after punctuation normalization; and (3) have a reference/paraphrase token Jaccard index between 0.25 and 0.65.

We retain 141,381,887 paraphrastic pairs, out of over 220 million, as training data after applying these filters. To ensure output quality, we only use back-translated paraphrases as source.

PARABANK is a real-cased resource. We mark all words that have first-character capitalization and convert them to lowercase. The marking is

[4] `https://github.com/awslabs/sockeye/`

[5] PARABANK generated paraphrases from two large bilingual corpora, CzEng (Bojar et al., 2016a) and GigaFrEn (Callison-Burch et al., 2009). We picked paraphrases from only CzEng, the larger one of the two.

used as a source factor (Sennrich and Haddow, 2016) to the encoder. This helps us to decrease the vocabulary size of the the training data.

We learn a shared byte-pair encoding (BPE) over the entire training data with 30,000 BPE operations (Sennrich et al., 2016), keeping all vocabulary items with a frequency over 50 in the post-BPE data. We follow Sennrich and Haddow (2016) and use "BIOE" tagging to annotate BPE segmentation and broadcast the casing factor accordingly. The encoder uses both source factors.

The model is trained on 2 NVIDIA GTX 1080Ti's until convergence (5 days).

**Rewriter Evaluation** We randomly sampled 100 instances from both MNLI matched and mismatched development set. Each instance consists of 4 sentences: premise, entailed, contradicting, and neutral. We use the following 3 different rewriters to rewrite all 800 sentences: (1) an LSTM-based rewriter trained on PARABANK alpha; following (Hu et al., 2019); (2) a Transformer-based rewriter trained on PARABANK alpha; and (3) a Transformer-based rewriter trained on full PARABANK with the filters and improvements described here.

Inspired by the interface of EASL (Sakaguchi and Van Durme, 2018), we ask crowd-workers to give each paraphrase a score between 0 and 100 depending its semantically similarity to the original, reference sentence. Independently, we provide options for flagging ungrammatical or nonsensical sentences. Paraphrases are judged by up to 3 different workers, with 11 workers participating. We randomly include an attention check consisting of reference sentence itself.[6]

The result is shown in Table 2. Switching the rewriter architecture from LSTM to Transformer improves the human-judged semantic similarity by 5.1% and fluency by 6.5%. The improvements described here leads to a gain of 9.6% in semantic similarity and 10.2% in fluency overall.

This improved Transformer-based rewriter is subsequently used for data augmentation.

| | Similarity | STD | Fluency |
|---|---|---|---|
| LSTM alpha | 74.5 | 25.0 | 80.7% |
| Transf. alpha | 78.3 | 22.9 | 87.2% |
| Transf. Full | **81.7** | **20.9** | **90.9%** |

Table 2: Comparison between three monolingual rewriting systems. Systems will "alpha" are trained on PARABANK alpha, which the other one is trained on the full data. Similarity is the mean human-judged semantic similarity score; the higher the better. STD described the standard deviation of similarity. Fluency is the percentage of paraphrases judged to be both grammatical and meaningful.

## 5 Paraphrastic Data Augmentation

We demonstrate the utility of our improved lexically-constrained decoding via data augmentation with some simple rewriting heuristics and two augmentation strategies. First, the model could be trained on the augmented (training) data. Orthogonally, predictions can be made on the all of the augmented (evaluation) data, which can then be *aggregated*. We show experimental results on natural language inference (NLI, Section 5.1), question answering (QA, Section 5.2), and NMT (Section 5.3) tasks.

These results are merely indicative of the potential in data augmentation via constrained paraphrasing, and are by no means a thorough investigation of strategies that yield the best improvements. Such an investigation, however, could be enabled by our algorithmic improvements and practitioners' domain expertise.

### 5.1 Natural Language Inference

Natural language inference is the task of determining entailment. Two sentences, a *premise* $p$ and a *hypothesis* $h$, are labelled with ENTAILMENT, CONTRADICTION, or NEUTRAL depending on whether $p$ logically entails, contradicts, or does not interact with $h$. MultiNLI (MNLI) (Williams et al., 2018) is a large, multi-genre dataset for natural language inference. The dataset is also divided into *matched* and *mismatched* portions based on whether the source of the evaluation data matches the source of the training set. Recent models rely on contextual sentence encoders pretrained on vast amounts of English monolingual text (Peters et al., 2018; Devlin et al., 2018).

We train and evaluate a model on MNLI, and find that data augmentation leads to improvements exceeding and complementary to those by ELMo, possibly due to improved lexical diversity during training and at inference.

[6] Only workers who pass the test at least 90% of the time and contribute at least 9 judgments are included in the result.

**Model** We use the model described in Bowman et al. (2019)[7] with the default parameters. They train a sentence representation model (possibly on top of ELMo) on the MNLI training set and subsequently train a clean task-specific model for each task (for this model, MNLI again). The task-specific MNLI model roughly follows BiDAF (Seo et al., 2016), followed by an MLP.

We also train a model without the ELMo contextual layers to compare contextual sentence representations against data augmentation. Since there is minor variance between different random seeds,[8] we train each model twice and evaluate the best-performing model on the development set.

**Paraphrase Generation** We generate paraphrases for our data augmentation experiments by negatively constraining on the most content-bearing token of each input sequence, as determined by inverse document frequency (IDF). For a given input sequence $\mathrm{s}$ we calculate the IDF of each token $t_i \in \mathrm{s}$ as $\log \frac{|D|}{|d \in D : t_i \in d|}$ where $D$ is the set of all English sentences in the train set. This relatively simple lexical constraint tends to force the decoder to rewrite the input sequence using different (but semantically related) words while maintaining fluency. In practice, we observed an average unigram precision of 67.6%; i.e., 32.4% of tokens in paraphrases were not contained in their corresponding inputs.

Additional results using a positively constrained rewriter in Appendix A.

**Data Setup** We first rewrite all premises $P$ to $P'$ and all hypotheses $H$ to $H'$, then for each $p_i \in P, h_i \in H$, we include all four examples, $(p_i, h_i)$, $(p_i, h'_i)$, $(p'_i, h_i)$, and $(p'_i, h'_i)$ into the training set, always preserving the original corresponding gold label. We include two copies of the original dataset in training to increase its weight. The original MNLI dataset contains 393K training pairs, and 20K in each dev and test, while the augmented dataset consists of 1.96M training pairs, and 79K in dev and test.

At test time, we rewrite the test sentence pairs. A trained model can also make predictions on each of three rewritten sentence pairs. Together with the original prediction, these four can then be aggregated by assigning weights to each prediction source. In our experiments, we perform this weighted aggregation (+Agg) for each model, tuning on the development set (Appendix B).

| | Dev. | Test. (m/mm) |
|---|---|---|
| Baseline | 74.8 | 74.7 (74.8/74.6) |
| +Agg. | 75.0 | 74.9 (74.9/74.8) |
| +Train | 75.4 | 75.2 (75.0/75.3) |
| +Train+Agg. | 75.6 | 75.4 (75.1/75.7) |
| +ELMo | 75.8 | 75.0 (75.1/75.0) |
| +Agg. | 75.9 | 75.2 (75.3/75.1) |
| +Train | 76.4 | 75.6 (75.6/75.6) |
| +Train+Agg. | **76.7** | **75.8 (75.9/75.7)** |

Table 3: F1 scores on MNLI. +Train denotes training on augmented data; +Agg. denotes using a weighted aggregation. Scores on the development set are a weighted average between the matched (m) and mismatched (mm) portions of the dataset, while the test set scores are additionally broken down into each category.

**Experimental Results** We find in Table 3 that data augmentation helps during training and inference. Not only are the total gains from augmentation comparable to those from ELMo, they are apparent even in the presence of the contextual sentence encoder. This suggests that the gains from data augmentation through rewrites complement recent gains from contextual sentence encoders.

The fairest external comparison is with Bowman et al. (2019), as our model is identical. Their best models achieve 76.2 F1 on development and 75.4 F1 on test. On the development set, they see a gain of 0.6 points by using multi-task training and external datasets. On that set, we report a total gain of up to 0.9 points purely through data augmentation. With respect to absolute test set scores, our best model outperforms theirs by 0.4, showing that rewriter-based data augmentation can be a powerful method for NLP tasks.

**Analysis** NLI systems have been shown to be brittle when the input is perturbed (Alzantot et al., 2018). Even when the premise or hypothesis is changed in a way that preserves the entailment semantics, the NLI system may make an incorrect prediction where it was previously correct. We present evidence showing that data augmentation for NLI reduces the brittleness of our model.

To demonstrate the brittleness of the baseline models, we analyze how predictions change. The model trained on the *original un-augmented*

[7] `https://github.com/jsalt18-sentence-repl/jiant`

[8] Bowman et al. (2019) found the variance to be 0.2

|  | $P', H$ | $P, H'$ | $P', H'$ | +Agg. |
|---|---|---|---|---|
| No change | 88.51 | 84.23 | 81.95 | 96.20 |
| $- \to +$ | 4.23 | 5.33 | 6.08 | 1.75 |
| $+ \to -$ | 5.84 | 8.44 | 9.67 | 1.49 |
| $-_1 \to -_2$ | 1.42 | 2.00 | 2.30 | 0.57 |
| No change | 88.20 | 83.26 | 80.68 | 96.00 |
| $- \to +$ | 4.03 | 5.45 | 6.11 | 1.80 |
| $+ \to -$ | 6.42 | 9.22 | 10.78 | 1.72 |
| $-_1 \to -_2$ | 1.35 | 2.08 | 2.42 | 0.47 |

Table 4: Percentage of changed predictions on the MNLI development set using the baseline model without (top) and with (bottom) ELMo. $- \to +$ (correct after rewrite), $+ \to -$ (originally correct), and $-_1 \to -_2$ (different incorrect) are changes after rewriting. +Agg. denotes the predictions after weighted aggregation.

$P$: Visit at sundown or out of season to get the full flavor of the setting
$H$: The setting is better to visit at sundown or during low season
$H'$: It is better to visit at sunset or during low season

Gold: Entailed
$P, H$: Predict Entailed
$P, H'$: Predict Neutral

$P$: I had rejected it as absurd , nevertheless it persisted
$H$: It persisted even after I rejected it as an absurdity
$H'$: It went on even after I turned it down as an absurdity

Gold: Entailed
$P, H$: Predict Entailed
$P, H'$: Predict Contradiction

Figure 4: Cases where the baseline system changes its prediction on rewritten examples.

dataset is evaluated on the original development set and each of the rewritten development sets, and we investigate the differences. Table 4 shows how often original predictions are different from the corresponding predictions on the rewritten development sets; predictions can be (1) unchanged, (2) newly correct, (3) newly incorrect, or (4) changed but still incorrect, while Figure 4 shows how even relatively modest, semantically valid paraphrases can cause the NLI model change incorrectly.

Given a perfect rewriter that always generates semantically equivalent paraphrases and a perfect NLI model robust to perturbations, we would expect no change in predictions between the original development set and the rewritten ones. However, this is not what we observe; Table 4 shows that rewriting leads to a greater percentage of newly incorrect predictions than newly correct predictions.

We believe that the higher percentage of newly incorrect predictions on the rewritten development sets demonstrates the brittleness of the NLI system rather than semantic dissimilarity that may be introduced by the rewriter. We note that the aggregated predictions shows the opposite pattern: we see a higher percentage of newly *correct* predictions than incorrect ones. If the paraphrases were largely semantically dissimilar we would not expect any gain by combining predictions.

Given both the numerical boost seen by aggregation and the above examples, we hypothesize that the rewriter does not frequently change entailment semantics. Because the semantics remain similar, and because the paraphrases were generated with constraints designed to introduce lexical diversity, we believe that the label-preserving data augmentation improves the NLI model by making it more tolerant of minor lexical differences, better able to generalize, and less inclined to memorize.

### 5.2 Question Answering

We apply our paraphrastic rewriter to the task of question answer sentence selection to see if augmenting with paraphrases leads to improvements. The task is defined as follows: Given a question $q$ and a set of candidate sentences $\{c_i\}$, select the candidates which answer $q$.

**Model** We adapt a popular neural architecture for NLI, InferSent (Conneau et al., 2017), to our QA sentence selection task. In InferSent, the questions and answers (originally the premises and hypotheses) are embedded using an uncontextualized word embedding (e.g. GloVe), which we also experiment with ELMo (Peters et al., 2018) to incorporate recent advancements in large-scale contextualized pre-training. Bidirectional LSTMs (Graves and Schmidhuber, 2005) are run atop of these contextualized embeddings and a max-pooling layer is used to generate a feature vector for both the question and the answer. Following various matching methods (Mou et al., 2016) and a multi-layer feed-forward neural network, the model produces a final score.

We train the system following the method proposed by Rao et al. (2016), utilizing a ranking loss (Weston and Watkins, 1999) that contrasts positive answers against negative ones.

**Paraphrase Generation** We augment each answer candidate sentence with exactly 1 paraphrase in the dataset using the following heuristics: (1) named entities shared between a specific answer and its corresponding question are retained as positive constraints; (2) correct answer spans are retained as positive constraints; (3) words with the top-$k$ IDFs (inverse document frequencies; hence "important" words) that are not positive constraints are selected as negative constraints to promote the lexical diversity of the paraphrases.[9]

**Data Setup** We augment the raw TREC-QA dataset (Wang et al., 2007) under the following orthogonal strategies: (1) augmenting the training set with the paraphrases generated via the approach described above; (2) augmenting the answer candidates at evaluation time, and choosing the max score among the paraphrases as the score (aggregation by *voting*).

**Experimental Results** We evaluate our models using average precision (MAP) and mean reciprocal rank (MRR). Model selection is done with early stopping to choose the epoch with the maximum MAP score. Note that the "Baseline (+ELMo)" settings below falls back to the standard QA selection task, and our score under ELMo is comparable to earlier state-of-the-art results, e.g. by Rao et al. (2016).

| | MAP | MRR |
|---|---|---|
| Baseline | 71.42 | 75.16 |
| +Voting | 72.94 | 77.17 |
| +Train | 71.57 | 74.63 |
| +Train +Voting | 73.96 | 80.77 |
| +ELMo | 77.49 | 81.86 |
| +Voting | **80.61** | **85.65** |
| +Train | 75.58 | 80.30 |
| +Train +Voting | 77.86 | 84.34 |

Table 5: Experimental results on QA selection.

It is shown that augmenting at evaluation time (aggregation by voting) result in stable improvement (around +2~3% MAP and +2~6% MRR for both scenarios that either augments the training data or not)—this shows that increasing the paraphrastic diversity of the answer candidates could potentially make the system more robust. However augmenting the training set does not yield such improvements—we speculate that this may introduce some noise to the training data.

### 5.3 Machine Translation

We apply our paraphrastic rewriter to the WMT 2016 Turkish-English translation task (Bojar et al., 2016b). We see no improvement in English to Turkish translation, but see a 1.1 BLEU improvement when training an initial NMT system on half paraphrased and half original data, and continued training on the original data. Full details of the experiments are in Appendix C. This was the highest concentration of standard data we experimented with, and future work will explore additional ways of data augmentation using paraphrases.

## 6 Conclusion

Lexically-constrained sequence decoding provides control over whether certain tokens or token sequences appear in the output. Motivated by applications such as large-scale MT, we improved the speed for constrained decoding significantly by proposing a vectorized dynamic beam allocation algorithm. We also added multi-state trie representations for robustness to corner cases.

Also reliant on the efficiency of constrained decoding is data augmentation via rewriting, where one might need to explore a variety of strategies with task-specific constraints on development data. We trained an improved monolingual sentential rewriter and used it to rewrite data for NLP tasks. We experimented with augmenting training data, aggregating predictions on rewritten test data, and both. Using a few simple constraint heuristics, we showed improvements additive to ELMo in NLI and QA, and in MT. The rewriter, along with the augmented data files, can be found at `http://nlp.jhu.edu/parabank`. We hope this will enable future exploration of augmentation strategies for a variety of NLP tasks.

## Acknowledgments

Thanks to Michael Denkowski, who first suggested using a trie to represent constraints in a group discussion. This research was supported in part by DARPA AIDA and DARPA LORELEI.

[9] Stopwords and tokens with non-letter characters (e.g. *with*, *42*, *n't*) are excluded. $k \in \{2, 3, 4\}$ is a hyperparameter we tune – we found out that generally $k = 2$ works the best.

## References

Moustafa Alzantot, Yash Sharma, Ahmed Elgohary, Bo-Jhang Ho, Mani Srivastava, and Kai-Wei Chang. 2018. Generating natural language adversarial examples. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2890–2896, Brussels, Belgium. Association for Computational Linguistics.

Peter Anderson, Basura Fernando, Mark Johnson, and Stephen Gould. 2017. Guided open vocabulary image captioning with constrained beam search. In *Proceedings of EMNLP 2017*, pages 936–945, Copenhagen, Denmark. Association for Computational Linguistics.

Ondřej Bojar, Ondřej Dušek, Tom Kocmi, Jindřich Libovický, Michal Novák, Martin Popel, Roman Sudarikov, and Dušan Variš. 2016a. CzEng 1.6: Enlarged Czech-English Parallel Corpus with Processing Tools Dockered. In *Text, Speech, and Dialogue: 19th International Conference, TSD 2016*, number 9924 in Lecture Notes in Computer Science, pages 231–238, Cham / Heidelberg / New York / Dordrecht / London. Masaryk University, Springer International Publishing.

Ondřej Bojar, Christian Federmann, Mark Fishel, Yvette Graham, Barry Haddow, Philipp Koehn, and Christof Monz. 2018. Findings of the 2018 conference on machine translation (WMT18). In *Proceedings of the Third Conference on Machine Translation: Shared Task Papers*, pages 272–303, Belgium, Brussels. Association for Computational Linguistics.

Ondřej Bojar, Rajen Chatterjee, Christian Federmann, Yvette Graham, Barry Haddow, Matthias Huck, Antonio Jimeno Yepes, Philipp Koehn, Varvara Logacheva, Christof Monz, Matteo Negri, Aurelie Neveol, Mariana Neves, Martin Popel, Matt Post, Raphael Rubino, Carolina Scarton, Lucia Specia, Marco Turchi, Karin Verspoor, and Marcos Zampieri. 2016b. Findings of the 2016 conference on machine translation. In *Proceedings of the First Conference on Machine Translation*, pages 131–198, Berlin, Germany. Association for Computational Linguistics.

Samuel R. Bowman, Ellie Pavlick, Edouard Grave, Benjamin Van Durme, Alex Wang, Jan Hula, Patrick Xia, Raghavendra Pappagari, R. Thomas McCoy, Roma Patel, Najoung Kim, Ian Tenney, Yinghui Huang, Katherin Yu, Shuning Jin, and Berlin Chen. 2019. Looking for ELMo's friends: Sentence-level pretraining beyond language modeling.

Chris Callison-Burch, Philipp Koehn, Christof Monz, and Josh Schroeder. 2009. Findings of the 2009 Workshop on Statistical Machine Translation. In *Proceedings of the Fourth Workshop on Statistical Machine Translation*, pages 1–28, Athens, Greece. Association for Computational Linguistics.

Charles Chen and Razvan Bunescu. 2017. An exploration of data augmentation and rnn architectures for question ranking in community question answering. In *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 442–447, Taipei, Taiwan. Asian Federation of Natural Language Processing.

Alexis Conneau, Douwe Kiela, Holger Schwenk, Loïc Barrault, and Antoine Bordes. 2017. Supervised learning of universal sentence representations from natural language inference data. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 670–680.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805.

Sergey Edunov, Myle Ott, Michael Auli, and David Grangier. 2018. Understanding back-translation at scale. In *Proceedings of EMNLP 2018*, pages 489–500. Association for Computational Linguistics.

Marzieh Fadaee, Arianna Bisazza, and Christof Monz. 2017. Data augmentation for low-resource neural machine translation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 567–573, Vancouver, Canada. Association for Computational Linguistics.

Marzieh Fadaee and Christof Monz. 2018. Back-translation sampling by targeting difficult words in neural machine translation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 436–446. Association for Computational Linguistics.

Juri Ganitkevitch, Benjamin Van Durme, and Chris Callison-Burch. 2013. Ppdb: The paraphrase database. In *Proceedings NAACL-HLT 2013*, pages 758–764.

Alex Graves and Jürgen Schmidhuber. 2005. Framewise phoneme classification with bidirectional lstm and other neural network architectures. *Neural Networks*, 18(5-6):602–610.

Jiatao Gu, Zhengdong Lu, Hang Li, and Victor O.K. Li. 2016. Incorporating copying mechanism in sequence-to-sequence learning. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1631–1640, Berlin, Germany. Association for Computational Linguistics.

Felix Hieber, Tobias Domhan, Michael Denkowski, David Vilar, Artem Sokolov, Ann Clifton, and Matt Post. 2017. Sockeye: A toolkit for neural machine translation. *CoRR*, abs/1712.05690.

Chris Hokamp and Qun Liu. 2017. Lexically constrained decoding for sequence generation using grid beam search. In *Proceedings of ACL*, pages 1535–1546, Vancouver, Canada.

Yutai Hou, Yijia Liu, Wanxiang Che, and Ting Liu. 2018. Sequence-to-sequence data augmentation for dialogue language understanding. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 1234–1245, Santa Fe, New Mexico, USA. Association for Computational Linguistics.

J. Edward Hu, Rachel Rudinger, Matt Post, and Benjamin Van Durme. 2019. PARABANK: Monolingual bitext generation and sentential paraphrasing via lexically-constrained neural machine translation. *AAAI*.

Mohit Iyyer, John Wieting, Kevin Gimpel, and Luke Zettlemoyer. 2018. Adversarial example generation with syntactically controlled paraphrase networks. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1875–1885. Association for Computational Linguistics.

Robin Jia and Percy Liang. 2016. Data recombination for neural semantic parsing. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12–22, Berlin, Germany. Association for Computational Linguistics.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Sosuke Kobayashi. 2018. Contextual augmentation: Data augmentation by words with paradigmatic relations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 452–457, New Orleans, Louisiana. Association for Computational Linguistics.

Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondrej Bojar, Alexandra Constantin, and Evan Herbst. 2007. Moses: Open Source Toolkit for Statistical Machine Translation. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics Companion Volume Proceedings of the Demo and Poster Sessions*, Prague, Czech Republic. Association for Computational Linguistics.

Jiwei Li, Michel Galley, Chris Brockett, Jianfeng Gao, and Bill Dolan. 2016. A diversity-promoting objective function for neural conversation models. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 110–119, San Diego, California. Association for Computational Linguistics.

Dekang Lin and Patrick Pantel. 2001. Dirt @sbt@discovery of inference rules from text. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '01, pages 323–328, New York, NY, USA. ACM.

Minh-Thang Luong and Christopher D. Manning. 2015. Stanford Neural Machine Translation Systems for Spoken Language Domain. In *International Workshop on Spoken Language Translation*, Da Nang, Vietnam.

Jonathan Mallinson, Rico Sennrich, and Mirella Lapata. 2017. Paraphrasing revisited with neural machine translation. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 881–893, Valencia, Spain. Association for Computational Linguistics.

Lili Mou, Rui Men, Ge Li, Yan Xu, Lu Zhang, Rui Yan, and Zhi Jin. 2016. Natural language inference by tree-based convolution and heuristic matching. In *Proc. ACL*, page 130.

Courtney Napoles, Chris Callison-Burch, and Matt Post. 2016. Sentential paraphrasing as black-box machine translation. In *Proceedings of the NAACL 2016*, pages 62–66, San Diego, California. Association for Computational Linguistics.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. BLEU: a method for automatic evaluation of machine translation. In *Proceedings of 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA. Association for Computational Linguistics.

Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237, New Orleans, Louisiana. Association for Computational Linguistics.

Matt Post. 2018. A call for clarity in reporting BLEU scores. In *Proceedings of the Third Conference on Machine Translation: Research Papers*, pages 186–191, Belgium, Brussels. Association for Computational Linguistics.

Matt Post and David Vilar. 2018. Fast lexically constrained decoding with dynamic beam allocation for neural machine translation. In *Proceedings of NAACL 2018*, pages 1314–1324, New Orleans, Louisiana. Association for Computational Linguistics.

Jinfeng Rao, Hua He, and Jimmy Lin. 2016. Noise-contrastive estimation for answer selection with deep neural networks. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, pages 1913–1916. ACM.

Keisuke Sakaguchi and Benjamin Van Durme. 2018. Efficient online scalar annotation with bounded support. In *Proceedings of ACL*, pages 208–218, Melbourne, Australia. ACL.

Abigail See, Peter J. Liu, and Christopher D. Manning. 2017. Get to the point: Summarization with pointer-generator networks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1073–1083, Vancouver, Canada. Association for Computational Linguistics.

Rico Sennrich and Barry Haddow. 2016. Linguistic input features improve neural machine translation. In *Proceedings of the First Conference on Machine Translation*, pages 83–91, Berlin, Germany. Association for Computational Linguistics.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Improving neural machine translation models with monolingual data. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 86–96, Berlin, Germany. Association for Computational Linguistics.

Min Joon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. 2016. Bidirectional attention flow for machine comprehension. *CoRR*, abs/1611.01603.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc.

Mengqiu Wang, Noah A Smith, and Teruko Mitamura. 2007. What is the Jeopardy model? A quasi-synchronous grammar for QA. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*.

Xinyi Wang, Hieu Pham, Zihang Dai, and Graham Neubig. 2018. Switchout: an efficient data augmentation algorithm for neural machine translation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 856–861. Association for Computational Linguistics.

Hila Weisman, Jonathan Berant, Idan Szpektor, and Ido Dagan. 2012. Learning verb inference rules from linguistically-motivated evidence. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 194–204, Jeju Island, Korea. Association for Computational Linguistics.

Jason Weston and Chris Watkins. 1999. Support vector machines for multi-class pattern recognition. In *ESANN 1999, 7th European Symposium on Artificial Neural Networks, Bruges, Belgium, April 21-23, 1999, Proceedings*, pages 219–224.

John Wieting and Kevin Gimpel. 2018. ParaNMT-50M: Pushing the limits of paraphrastic sentence embeddings with millions of machine translations. In *Proceedings of ACL*, pages 451–462. ACL.

Adina Williams, Nikita Nangia, and Samuel Bowman. 2018. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1112–1122. Association for Computational Linguistics.