Sequence-to-Sequence Learning
as Beam-Search Optimization

Sam Wiseman and **Alexander M. Rush**
School of Engineering and Applied Sciences
Harvard University
Cambridge, MA, USA
{swiseman,srush}@seas.harvard.edu

Abstract

Sequence-to-Sequence (seq2seq) modeling has rapidly become an important general-purpose NLP tool that has proven effective for many text-generation and sequence-labeling tasks. Seq2seq builds on deep neural language modeling and inherits its remarkable accuracy in estimating local, next-word distributions. In this work, we introduce a model and beam-search training scheme, based on the work of Daumé III and Marcu (2005), that extends seq2seq to learn global sequence scores. This structured approach avoids classical biases associated with local training and unifies the training loss with the test-time usage, while preserving the proven model architecture of seq2seq and its efficient training approach. We show that our system outperforms a highly-optimized attention-based seq2seq system and other baselines on three different sequence to sequence tasks: word ordering, parsing, and machine translation.

1 Introduction

Sequence-to-Sequence learning with deep neural networks (herein, seq2seq) (Sutskever et al., 2011; Sutskever et al., 2014) has rapidly become a very useful and surprisingly general-purpose tool for natural language processing. In addition to demonstrating impressive results for machine translation (Bahdanau et al., 2015), roughly the same model and training have also proven to be useful for sentence compression (Filippova et al., 2015), parsing (Vinyals et al., 2015), and dialogue systems (Serban et al., 2016), and they additionally underlie other text generation applications, such as image or video captioning (Venugopalan et al., 2015; Xu et al., 2015).

The dominant approach to training a seq2seq system is as a conditional language model, with training maximizing the likelihood of each successive target word conditioned on the input sequence and the *gold* history of target words. Thus, training uses a strictly word-level loss, usually cross-entropy over the target vocabulary. This approach has proven to be very effective and efficient for training neural language models, and seq2seq models similarly obtain impressive perplexities for word-generation tasks.

Notably, however, seq2seq models are not used as conditional language models at test-time; they must instead generate fully-formed word sequences. In practice, generation is accomplished by searching over output sequences greedily or with beam search. In this context, Ranzato et al. (2016) note that the combination of the training and generation scheme just described leads to at least two major issues:

1. *Exposure Bias*: the model is never exposed to its own errors during training, and so the inferred histories at test-time do not resemble the gold training histories.

2. *Loss-Evaluation Mismatch*: training uses a word-level loss, while at test-time we target improving sequence-level evaluation metrics, such as BLEU (Papineni et al., 2002).

We might additionally add the concern of *label bias* (Lafferty et al., 2001) to the list, since word-probabilities at each time-step are locally normalized, guaranteeing that successors of incorrect his-

In this work we develop a non-probabilistic variant of the seq2seq model that can assign a score to any possible target *sequence*, and we propose a training procedure, inspired by the learning as search optimization (LaSO) framework of Daumé III and Marcu (2005), that defines a loss function in terms of errors made during beam search. Furthermore, we provide an efficient algorithm to backpropagate through the beam-search procedure during seq2seq training.

This approach offers a possible solution to each of the three aforementioned issues, while largely maintaining the model architecture and training efficiency of standard seq2seq learning. Moreover, by scoring sequences rather than words, our approach also allows for enforcing hard-constraints on sequence generation *at training time*. To test out the effectiveness of the proposed approach, we develop a general-purpose seq2seq system with beam search optimization. We run experiments on three very different problems: word ordering, syntactic parsing, and machine translation, and compare to a highly-tuned seq2seq system with attention (Luong et al., 2015). The version with beam search optimization shows significant improvements on all three tasks, and particular improvements on tasks that require difficult search.

2 Related Work

The issues of exposure bias and label bias have received much attention from authors in the structured prediction community, and we briefly review some of this work here. One prominent approach to combating exposure bias is that of SEARN (Daumé III et al., 2009), a meta-training algorithm that learns a search policy in the form of a cost-sensitive classifier trained on examples generated from an interpolation of an oracle policy and the model's current (learned) policy. Thus, SEARN explicitly targets the mismatch between oracular training and non-oracular (often greedy) test-time inference by training on the output of the model's own policy. DAgger (Ross et al., 2011) is a similar approach, which differs in terms of how training examples are generated and aggregated, and there have additionally been impor-

tant refinements to this style of training over the past several years (Chang et al., 2015). When it comes to training RNNs, SEARN/DAgger has been applied under the name "scheduled sampling" (Bengio et al., 2015), which involves training an RNN to generate the $t + 1$'st token in a target sequence after consuming either the true t'th token, or, with probability that increases throughout training, the predicted t'th token.

Though technically possible, it is uncommon to use beam search when training with SEARN/DAgger. The early-update (Collins and Roark, 2004) and LaSO (Daumé III and Marcu, 2005) training strategies, however, explicitly account for beam search, and describe strategies for updating parameters when the gold structure becomes unreachable during search. Early update and LaSO differ primarily in that the former discards a training example after the first search error, whereas LaSO resumes searching after an error from a state that includes the gold partial structure. In the context of feed-forward neural network training, early update training has been recently explored in a feed-forward setting by Zhou et al. (2015) and Andor et al. (2016). Our work differs in that we adopt a LaSO-like paradigm (with some minor modifications), and apply it to the training of seq2seq RNNs (rather than feed-forward networks). We also note that Watanabe and Sumita (2015) apply maximum-violation training (Huang et al., 2012), which is similar to early-update, to a parsing model with recurrent components, and that Yazdani and Henderson (2015) use beam-search in training a discriminative, locally normalized dependency parser with recurrent components.

Recently authors have also proposed alleviating exposure bias using techniques from reinforcement learning. Ranzato et al. (2016) follow this approach to train RNN decoders in a seq2seq model, and they obtain consistent improvements in performance, even over models trained with scheduled sampling. As Daumé III and Marcu (2005) note, LaSO is similar to reinforcement learning, except it does not require "exploration" in the same way. Such exploration may be unnecessary in supervised text-generation, since we typically know the gold partial sequences at each time-step. Shen et al. (2016) use minimum risk training (approximated by

sampling) to address the issues of exposure bias and loss-evaluation mismatch for seq2seq MT, and show impressive performance gains.

Whereas exposure bias results from training in a certain way, label bias results from properties of the model itself. In particular, label bias is likely to affect structured models that make sub-structure predictions using locally-normalized scores. Because the neural and non-neural literature on this point has recently been reviewed by Andor et al. (2016), we simply note here that RNN models are typically locally normalized, and we are unaware of any specifically seq2seq work with RNNs that does *not* use locally-normalized scores. The model we introduce here, however, is not locally normalized, and so should not suffer from label bias. We also note that there are some (non-seq2seq) exceptions to the trend of locally normalized RNNs, such as the work of Sak et al. (2014) and Voigtlaender et al. (2015), who train LSTMs in the context of HMMs for speech recognition using sequence-level objectives; their work does not consider search, however.

3 Background and Notation

In the simplest seq2seq scenario, we are given a collection of source-target sequence pairs and tasked with learning to generate target sequences from source sequences. For instance, we might view machine translation in this way, where in particular we attempt to generate English sentences from (corresponding) French sentences. Seq2seq models are part of the broader class of "encoder-decoder" models (Cho et al., 2014), which first use an encoding model to transform a source object into an encoded representation x. Many different sequential (and non-sequential) encoders have proven to be effective for different source domains. In this work we are agnostic to the form of the encoding model, and simply assume an abstract source representation x.

Once the input sequence is encoded, seq2seq models generate a target sequence using a *decoder*. The decoder is tasked with generating a target sequence of words from a target vocabulary $\mathcal{V}$. In particular, words are generated sequentially by conditioning on the input representation x and on the previously generated words or *history*. We use the notation $w_{1:T}$ to refer to an arbitrary word sequence

of length T, and the notation $y_{1:T}$ to refer to the *gold* (i.e., correct) target word sequence for an input x.

Most seq2seq systems utilize a recurrent neural network (RNN) for the decoder model. Formally, a recurrent neural network is a parameterized non-linear function $\mathbf{RNN}$ that recursively maps a sequence of vectors to a sequence of hidden states. Let $m_1, \ldots, m_T$ be a sequence of T vectors, and let h_0 be some initial state vector. Applying an RNN to any such sequence yields hidden states h_t at each time-step t, as follows:

$$h_t \leftarrow \mathbf{RNN}(m_t, h_{t-1}; \theta),$$

where θ is the set of model parameters, which are shared over time. In this work, the vectors m_t will always correspond to the embeddings of a target word sequence $w_{1:T}$, and so we will also write $h_t \leftarrow \mathbf{RNN}(w_t, h_{t-1}; \theta)$, with w_t standing in for its embedding.

RNN decoders are typically trained to act as conditional language models. That is, one attempts to model the probability of the t'th target word conditioned on x and the target history by stipulating that $p(w_t|w_{1:t-1}, x) = g(w_t, h_{t-1}, x)$, for some parameterized function g typically computed with an affine layer followed by a softmax. In computing these probabilities, the state h_{t-1} represents the target history, and h_0 is typically set to be some function of x. The complete model (including encoder) is trained, analogously to a neural language model, to minimize the cross-entropy loss at each time-step while conditioning on the gold history in the training data. That is, the model is trained to minimize $-\ln \prod_{t=1}^{T} p(y_t|y_{1:t-1}, x)$.

Once the decoder is trained, discrete sequence generation can be performed by approximately maximizing the probability of the target sequence under the conditional distribution, $\hat{y}_{1:T} = \text{argbeam}_{w_{1:T}} \prod_{t=1}^{T} p(w_t|w_{1:t-1}, x)$, where we use the notation argbeam to emphasize that the decoding process requires heuristic search, since the RNN model is non-Markovian. In practice, a simple beam search procedure that explores K prospective histories at each time-step has proven to be an effective decoding approach. However, as noted above, decoding in this manner after conditional language-model style training *potentially* suffers from the is-

sues of exposure bias and label bias, which motivates the work of this paper.

4 Beam Search Optimization

We begin by making one small change to the seq2seq modeling framework. Instead of predicting the probability of the next word, we instead learn to produce (non-probabilistic) scores for ranking sequences. Define the score of a sequence consisting of *history* $w_{1:t-1}$ followed by a single word w_t as $f(w_t, \boldsymbol{h}_{t-1}, \boldsymbol{x})$, where f is a parameterized function examining the current hidden-state of the relevant RNN at time $t-1$ as well as the input representation $\boldsymbol{x}$. In experiments, our f will have an identical form to g but *without* the final softmax transformation (which transforms unnormalized scores into probabilities), thereby allowing the model to avoid issues associated with the label bias problem.

More importantly, we also modify how this model is trained. Ideally we would train by comparing the gold sequence to the highest-scoring complete sequence. However, because finding the argmax sequence according to this model is intractable, we propose to adopt a LaSO-like (Daumé III and Marcu, 2005) scheme to train, which we will refer to as beam search optimization (BSO). In particular, we define a loss that penalizes the gold sequence falling off the beam during training.[1] The proposed training approach is a simple way to expose the model to incorrect histories and to match the training procedure to test generation. Furthermore we show that it can be implemented efficiently without changing the asymptotic run-time of training, beyond a factor of the beam size K.

4.1 Search-Based Loss

We now formalize this notion of a search-based loss for RNN training. Assume we have a set S_t of K candidate sequences of length t. We can calculate a score for each sequence in S_t using a scoring function f parameterized with an RNN, as above, and we define the sequence $\hat{y}_{1:t}^{(K)} \in S_t$ to be the K'th ranked

---

[1]Using a non-probabilistic model further allows us to incur no loss (and thus require no update to parameters) when the gold sequence *is* on the beam; this contrasts with models based on a CRF loss, such as those of Andor et al. (2016) and Zhou et al. (2015), though in training those models are simply not updated when the gold sequence remains on the beam.

sequence in S_t according to f. That is, assuming distinct scores,

$$|\{\hat{y}_{1:t}^{(k)} \in S_t \mid f(\hat{y}_t^{(k)}, \hat{\boldsymbol{h}}_{t-1}^{(k)}) > f(\hat{y}_t^{(K)}, \hat{\boldsymbol{h}}_{t-1}^{(K)})\}| = K-1,$$

where $\hat{y}_t^{(k)}$ is the t'th token in $\hat{y}_{1:t}^{(k)}$, $\hat{\boldsymbol{h}}_{t-1}^{(k)}$ is the RNN state corresponding to its $t-1$'st step, and where we have omitted the $\boldsymbol{x}$ argument to f for brevity.

We now define a loss function that gives loss each time the score of the gold prefix $y_{1:t}$ does not exceed that of $\hat{y}_{1:t}^{(K)}$ by a margin:

$$\mathcal{L}(f) =$$
$$\sum_{t=1}^{T} \Delta(\hat{y}_{1:t}^{(K)}) \left[1 - f(y_t, \boldsymbol{h}_{t-1}) + f(\hat{y}_t^{(K)}, \hat{\boldsymbol{h}}_{t-1}^{(K)}) \right].$$

Above, the $\Delta(\hat{y}_{1:t}^{(K)})$ term denotes a mistake-specific cost-function, which allows us to scale the loss depending on the severity of erroneously predicting $\hat{y}_{1:t}^{(K)}$; it is assumed to return 0 when the margin requirement is satisfied, and a positive number otherwise. It is this term that allows us to use sequence-rather than word-level costs in training (addressing the 2nd issue in the introduction). For instance, when training a seq2seq model for machine translation, it may be desirable to have $\Delta(\hat{y}_{1:t}^{(K)})$ be inversely related to the partial sentence-level BLEU score of $\hat{y}_{1:t}^{(K)}$ with $y_{1:t}$; we experiment along these lines in Section 5.3.

Finally, because we want the full gold sequence to be at the top of the beam at the end of search, when $t = T$ we modify the loss to require the score of $y_{1:T}$ to exceed the score of the *highest* ranked incorrect prediction by a margin.

We can optimize the loss $\mathcal{L}$ using a two-step process: (1) in a forward pass, we compute candidate sets S_t and record margin violations (sequences with non-zero loss); (2) in a backward pass, we back-propagate the errors through the seq2seq RNNs. Unlike standard seq2seq training, the first-step requires running search (in our case beam search) to find margin violations. The second step can be done by adapting back-propagation through time (BPTT). We next discuss the details of this process.

4.2 Forward: Find Violations

In order to minimize this loss, we need to specify a procedure for constructing candidate sequences $\hat{y}_{1:t}^{(k)}$

at each time step t so that we find margin violations. We follow LaSO (rather than early-update [2]; see Section 2) and build candidates in a recursive manner. If there was no margin violation at $t-1$, then S_t is constructed using a standard beam search update. If there was a margin violation, S_t is constructed as the K best sequences assuming the gold history $y_{1:t-1}$ through time-step $t-1$.

Formally, assume the function succ maps a sequence $w_{1:t-1} \in \mathcal{V}^{t-1}$ to the set of all valid sequences of length t that can be formed by appending to it a valid word $w \in \mathcal{V}$. In the simplest, unconstrained case, we will have

$$\text{succ}(w_{1:t-1}) = \{w_{1:t-1}, w \mid w \in \mathcal{V}\}.$$

As an important aside, note that for some problems it may be preferable to define a succ function which imposes hard constraints on successor sequences. For instance, if we would like to use seq2seq models for parsing (by emitting a constituency or dependency structure encoded into a sequence in some way), we will have hard constraints on the sequences the model can output, namely, that they represent valid parses. While hard constraints such as these would be difficult to add to standard seq2seq at training time, in our framework they can naturally be added to the succ function, allowing us to *train* with hard constraints; we experiment along these lines in Section 5.3, where we refer to a model trained with constrained beam search as ConBSO.

Having defined an appropriate succ function, we specify the candidate set as:

$$S_t = \text{topK} \begin{cases} \text{succ}(y_{1:t-1}) & \text{violation at } t-1 \\ \bigcup_{k=1}^{K} \text{succ}(\hat{y}_{1:t-1}^{(k)}) & \text{otherwise,} \end{cases}$$

where we have a margin violation at $t-1$ iff $f(y_{t-1}, \boldsymbol{h}_{t-2}) < f(\hat{y}_{t-1}^{(K)}, \hat{\boldsymbol{h}}_{t-2}^{(K)}) + 1$, and where topK considers the scores given by f. This search procedure is illustrated in the top portion of Figure 1.

In the forward pass of our training algorithm, shown as the first part of Algorithm 1, we run this version of beam search and collect all sequences and their hidden states that lead to losses.

---

[2]We found that training with early-update rather than (delayed) LaSO did not work well, even after pre-training. Given the success of early-update in many NLP tasks this was somewhat surprising. We leave this question to future work.
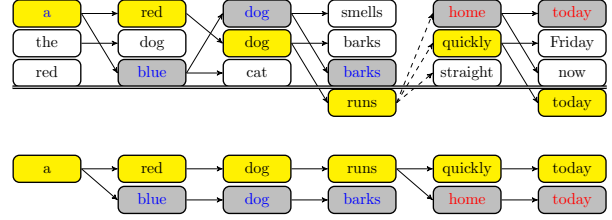

Figure 1: Top: possible $\hat{y}_{1:t}^{(k)}$ formed in training with a beam of size $K=3$ and with gold sequence $y_{1:6}$ = "a red dog runs quickly today". The gold sequence is highlighted in yellow, and the predicted prefixes involved in margin violations (at $t=4$ and $t=6$) are in gray. Note that time-step $T=6$ uses a different loss criterion. Bottom: prefixes that actually participate in the loss, arranged to illustrate the back-propagation process.

4.3 Backward: Merge Sequences

Once we have collected margin violations we can run backpropagation to compute parameter updates. Assume a margin violation occurs at time-step t between the predicted history $\hat{y}_{1:t}^{(K)}$ and the gold history $y_{1:t}$. As in standard seq2seq training we must back-propagate this error through the gold history; however, unlike seq2seq we also have a gradient for the wrongly predicted history.

Recall that to back-propagate errors through an RNN we run a recursive backward procedure — denoted below by **BRNN** — at each time-step t, which accumulates the gradients of next-step and future losses with respect to $\boldsymbol{h}_t$. We have:

$$\nabla_{\boldsymbol{h}_t}\mathcal{L} \leftarrow \textbf{BRNN}(\nabla_{\boldsymbol{h}_t}\mathcal{L}_{t+1}, \nabla_{\boldsymbol{h}_{t+1}}\mathcal{L}),$$

where $\mathcal{L}_{t+1}$ is the loss at step $t+1$, deriving, for instance, from the score $f(y_{t+1}, \boldsymbol{h}_t)$. Running this **BRNN** procedure from $t=T-1$ to $t=0$ is known as back-propagation through time (BPTT).

In determining the total computational cost of back-propagation here, first note that in the worst case there is one violation at each time-step, which leads to T independent, incorrect sequences. Since we need to call **BRNN** $O(T)$ times for each sequence, a naive strategy of running BPTT for each incorrect sequence would lead to an $O(T^2)$ backward pass, rather than the $O(T)$ time required for the standard seq2seq approach.

Fortunately, our combination of search-strategy and loss make it possible to efficiently share **BRNN** operations. This shared structure comes

naturally from the LaSO update, which resets the beam in a convenient way.

We informally illustrate the process in Figure 1. The top of the diagram shows a possible sequence of $\hat{y}_{1:t}^{(k)}$ formed during search with a beam of size 3 for the target sequence y = "a red dog runs quickly today." When the gold sequence falls off the beam at $t = 4$, search resumes with $S_5 = \text{succ}(y_{1:4})$, and so all subsequent predicted sequences have $y_{1:4}$ as a prefix and are thus functions of $\boldsymbol{h}_4$. Moreover, because our loss function only involves the scores of the gold prefix and the violating prefix, we end up with the relatively simple computation tree shown at the bottom of Figure 1. It is evident that we can backpropagate in a single pass, accumulating gradients from sequences that diverge from the gold at the time-step that precedes their divergence. The second half of Algorithm 1 shows this explicitly for a single sequence, though it is straightforward to extend the algorithm to operate in batch.[3]

5 Data and Methods

We run experiments on three different tasks, comparing our approach to the seq2seq baseline, and to other relevant baselines.

5.1 Model

While the method we describe applies to seq2seq RNNs in general, for all experiments we use the global attention model of Luong et al. (2015) — which consists of an LSTM (Hochreiter and Schmidhuber, 1997) encoder and an LSTM decoder with a global attention model — as both the baseline seq2seq model (i.e., as the model that computes the g in Section 3) and as the model that computes our sequence-scores $f(w_t, \boldsymbol{h}_{t-1}, \boldsymbol{x})$. As in Luong et al. (2015), we also use "input feeding," which involves feeding the attention distribution from the previous time-step into the decoder at the current step. This model architecture has been found to be highly performant for neural machine translation and other seq2seq tasks.

---

[3]We also note that because we do not update the parameters until after the T'th search step, our training procedure differs slightly from LaSO (which is online), and in this aspect is essentially equivalent to the "delayed LaSO update" of Björkelund and Kuhn (2014).

Algorithm 1 Seq2seq Beam-Search Optimization

1: **procedure** BSO($\boldsymbol{x}, K_{tr}, \text{succ}$)
2:    /*FORWARD*/
3:    Init empty storage $\hat{y}_{1:T}$ and $\hat{\boldsymbol{h}}_{1:T}$; init S_1
4:    $r \leftarrow 0; violations \leftarrow \{0\}$
5:    **for** $t = 1, \ldots, T$ **do**
6:      $K = K_{tr}$ if $t \neq T$ else $\underset{k:\hat{y}_{1:t}^{(k)} \neq y_{1:t}}{\arg\max} f(\hat{y}_t^{(k)}, \hat{\boldsymbol{h}}_{t-1}^{(k)})$
7:      **if** $f(y_t, \boldsymbol{h}_{t-1}) < f(\hat{y}_t^{(K)}, \hat{\boldsymbol{h}}_{t-1}^{(K)}) + 1$ **then**
8:        $\hat{\boldsymbol{h}}_{r:t-1} \leftarrow \hat{\boldsymbol{h}}_{r:t-1}^{(K)}$
9:        $\hat{y}_{r+1:t} \leftarrow \hat{y}_{r+1:t}^{(K)}$
10:      Add t to $violations$
11:      $r \leftarrow t$
12:      $S_{t+1} \leftarrow \text{topK}(\text{succ}(y_{1:t}))$
13:      **else**
14:        $S_{t+1} \leftarrow \text{topK}(\bigcup_{k=1}^{K} \text{succ}(\hat{y}_{1:t}^{(k)}))$
15:    /*BACKWARD*/
16:    $grad\_\boldsymbol{h}_T \leftarrow \boldsymbol{0}; grad\_\hat{\boldsymbol{h}}_T \leftarrow \boldsymbol{0}$
17:    **for** $t = T-1, \ldots, 1$ **do**
18:      $grad\_\boldsymbol{h}_t \leftarrow \textbf{BRNN}(\nabla_{\boldsymbol{h}_t}\mathcal{L}_{t+1}, grad\_\boldsymbol{h}_{t+1})$
19:      $grad\_\hat{\boldsymbol{h}}_t \leftarrow \textbf{BRNN}(\nabla_{\hat{\boldsymbol{h}}_t}\mathcal{L}_{t+1}, grad\_\hat{\boldsymbol{h}}_{t+1})$
20:      **if** $t-1 \in violations$ **then**
21:        $grad\_\boldsymbol{h}_t \leftarrow grad\_\boldsymbol{h}_t + grad\_\hat{\boldsymbol{h}}_t$
22:        $grad\_\hat{\boldsymbol{h}}_t \leftarrow \boldsymbol{0}$

To distinguish the models we refer to our system as BSO (beam search optimization) and to the baseline as seq2seq. When we apply constrained training (as discussed in Section 4.2), we refer to the model as ConBSO. In providing results we also distinguish between the beam size K_{tr} with which the model is trained, and the beam size K_{te} which is used at test-time. In general, if we plan on evaluating with a beam of size K_{te} it makes sense to train with a beam of size $K_{tr} = K_{te} + 1$, since our objective requires the gold sequence to be scored higher than the *last* sequence on the beam.

5.2 Methodology

Here we detail additional techniques we found necessary to ensure the model learned effectively. First, we found that the model failed to learn when trained from a random initialization.[4] We therefore found it necessary to pre-train the model using a standard, word-level cross-entropy loss as described in Sec-

---

[4]This may be because there is relatively little signal in the sparse, sequence-level gradient, but this point requires further investigation.

tion 3. The necessity of pre-training in this instance is consistent with the findings of other authors who train non-local neural models (Kingsbury, 2009; Sak et al., 2014; Andor et al., 2016; Ranzato et al., 2016).[5]

Similarly, it is clear that the smaller the beam used in training is, the less room the model has to make erroneous predictions without running afoul of the margin loss. Accordingly, we also found it useful to use a "curriculum beam" strategy in training, whereby the size of the beam is increased gradually during training. In particular, given a desired training beam size K_{tr}, we began training with a beam of size 2, and increased it by 1 every 2 epochs until reaching K_{tr}.

Finally, it has been established that *dropout* (Srivastava et al., 2014) regularization improves the performance of LSTMs (Pham et al., 2014; Zaremba et al., 2014), and in our experiments we run beam search under dropout.[6]

For all experiments, we trained both seq2seq and BSO models with mini-batch Adagrad (Duchi et al., 2011) (using batches of size 64), and we renormalized all gradients so they did not exceed 5 before updating parameters. We did not extensively tune learning-rates, but we found initial rates of 0.02 for the encoder and decoder LSTMs, and a rate of 0.1 or 0.2 for the final linear layer (i.e., the layer tasked with making word-predictions at each time-step) to work well across all the tasks we considered. Code implementing the experiments described below can be found at `https://github.com/harvardnlp/BSO`.[7]

5.3 Tasks and Results

Our experiments are primarily intended to evaluate the effectiveness of beam search optimization over standard seq2seq training. As such, we run experiments with the same model across three very dif-

---

[5] Andor et al. (2016) found, however, that pre-training only increased convergence-speed, but was not necessary for obtaining good results.

[6] However, it is important to ensure that the same mask applied at each time-step of the forward search is also applied at the corresponding step of the backward pass. We accomplish this by pre-computing masks for each time-step, and sharing them between the partial sequence LSTMs.

[7] Our code is based on Yoon Kim's seq2seq code, `https://github.com/harvardnlp/seq2seq-attn`.

ferent problems: word ordering, dependency parsing, and machine translation. While we do not include all the features and extensions necessary to reach state-of-the-art performance, even the baseline seq2seq model is generally quite performant.

Word Ordering The task of correctly ordering the words in a shuffled sentence has recently gained some attention as a way to test the (syntactic) capabilities of text-generation systems (Zhang and Clark, 2011; Zhang and Clark, 2015; Liu et al., 2015; Schmaltz et al., 2016). We cast this task as seq2seq problem by viewing a shuffled sentence as a source sentence, and the correctly ordered sentence as the target. While word ordering is a somewhat synthetic task, it has two interesting properties for our purposes. First, it is a task which plausibly requires search (due to the exponentially many possible orderings), and, second, there is a clear hard constraint on output sequences, namely, that they be a permutation of the source sequence. For both the baseline and BSO models we enforce this constraint at test-time. However, we also experiment with constraining the BSO model during training, as described in Section 4.2, by defining the succ function to only allow successor sequences containing un-used words in the source sentence.

For experiments, we use the same PTB dataset (with the standard training, development, and test splits) and evaluation procedure as in Zhang and Clark (2015) and later work, with performance reported in terms of BLEU score with the correctly ordered sentences. For all word-ordering experiments we use 2-layer encoder and decoder LSTMs, each with 256 hidden units, and dropout with a rate of 0.2 between LSTM layers. We use simple 0/1 costs in defining the Δ function.

We show our test-set results in Table 1. We see that on this task there is a large improvement at each beam size from switching to BSO, and a further improvement from using the constrained model.

Inspired by a similar analysis in Daumé III and Marcu (2005), we further examine the relationship between K_{tr} and K_{te} when training with ConBSO in Table 2. We see that larger K_{tr} hurt greedy inference, but that results continue to improve, at least initially, when using a K_{te} that is (somewhat) bigger than $K_{tr} - 1$.

	Word Ordering (BLEU)		
	$K_{te} = 1$	$K_{te} = 5$	$K_{te} = 10$
seq2seq	25.2	29.8	31.0
BSO	28.0	33.2	34.3
ConBSO	**28.6**	**34.3**	**34.5**
LSTM-LM	15.4	-	26.8

Table 1: Word ordering. BLEU Scores of seq2seq, BSO, constrained BSO, and a vanilla LSTM language model (from Schmaltz et al, 2016). All experiments above have $K_{tr} = 6$.

	Word Ordering Beam Size (BLEU)		
	$K_{te} = 1$	$K_{te} = 5$	$K_{te} = 10$
$K_{tr} = 2$	30.59	31.23	30.26
$K_{tr} = 6$	28.20	34.22	34.67
$K_{tr} = 11$	26.88	34.42	34.88
seq2seq	26.11	30.20	31.04

Table 2: Beam-size experiments on word ordering development set. All numbers reflect training with constraints (ConBSO).

Dependency Parsing We next apply our model to dependency parsing, which also has hard constraints and plausibly benefits from search. We treat dependency parsing with arc-standard transitions as a seq2seq task by attempting to map from a source sentence to a target sequence of source sentence words interleaved with the arc-standard, reduce-actions in its parse. For example, we attempt to map the source sentence

But it was the Quotron problems that ...

to the target sequence

But it was @L_SBJ @L_DEP the Quotron problems @L_NMOD @L_NMOD that ...

We use the standard Penn Treebank dataset splits with Stanford dependency labels, and the standard UAS/LAS evaluation metric (excluding punctuation) following Chen and Manning (2014). All models thus see only the words in the source and, when decoding, the actions it has emitted so far; no other features are used. We use 2-layer encoder and decoder LSTMs with 300 hidden units per layer

	Dependency Parsing (UAS/LAS)		
	$K_{te} = 1$	$K_{te} = 5$	$K_{te} = 10$
seq2seq	**87.33/82.26**	88.53/84.16	88.66/84.33
BSO	86.91/82.11	91.00/**87.18**	91.17/**87.41**
ConBSO	85.11/79.32	**91.25**/86.92	**91.57**/87.26
Andor	93.17/91.18	-	-

Table 3: Dependency parsing. UAS/LAS of seq2seq, BSO, ConBSO and baselines on PTB test set. Andor is the current state-of-the-art model for this data set (Andor et al. 2016), and we note that with a beam of size 32 they obtain 94.41/92.55. All experiments above have $K_{tr} = 6$.

and dropout with a rate of 0.3 between LSTM layers. We replace singleton words in the training set with an UNK token, normalize digits to a single symbol, and initialize word embeddings for both source and target words from the publicly available `word2vec` (Mikolov et al., 2013) embeddings. We use simple 0/1 costs in defining the Δ function.

As in the word-ordering case, we also experiment with modifying the succ function in order to train under hard constraints, namely, that the emitted target sequence be a valid parse. In particular, we constrain the output at each time-step to obey the stack constraint, and we ensure words in the source are emitted in order.

We show results on the test-set in Table 3. BSO and ConBSO both show significant improvements over seq2seq, with ConBSO improving most on UAS, and BSO improving most on LAS. We achieve a reasonable final score of 91.57 UAS, which lags behind the state-of-the-art, but is promising for a general-purpose, word-only model.

Translation We finally evaluate our model on a small machine translation dataset, which allows us to experiment with a cost function that is not 0/1, and to consider other baselines that attempt to mitigate exposure bias in the seq2seq setting. We use the dataset from the work of Ranzato et al. (2016), which uses data from the German-to-English portion of the IWSLT 2014 machine translation evaluation campaign (Cettolo et al., 2014). The data comes from translated TED talks, and the dataset contains roughly 153K training sentences, 7K development sentences, and 7K test sentences. We use the same preprocessing and dataset splits as Ranzato et

	Machine Translation (BLEU)		
	$K_{te} = 1$	$K_{te} = 5$	$K_{te} = 10$
seq2seq	22.53	24.03	23.87
BSO, SB-Δ	**23.83**	**26.36**	**25.48**
XENT	17.74	20.10	20.28
DAD	20.12	22.25	22.40
MIXER	20.73	21.81	21.83

Table 4: Machine translation experiments on test set; results below middle line are from MIXER model of Ranzato et al. (2016). SB-Δ indicates sentence BLEU costs are used in defining Δ. XENT is similar to our seq2seq model but with a convolutional encoder and simpler attention. DAD trains seq2seq with scheduled sampling (Bengio et al., 2015). BSO, SB-Δ experiments above have $K_{tr} = 6$.

al. (2016), and like them we also use a single-layer LSTM decoder with 256 units. We also use dropout with a rate of 0.2 between each LSTM layer. We emphasize, however, that while our decoder LSTM is of the same size as that of Ranzato et al. (2016), our results are not directly comparable, because we use an LSTM encoder (rather than a convolutional encoder as they do), a slightly different attention mechanism, and input feeding (Luong et al., 2015).

For our main MT results, we set $\Delta(\hat{y}_{1:t}^{(k)})$ to $1 - \text{SB}(\hat{y}_{r+1:t}^{(K)}, y_{r+1:t})$, where r is the last margin violation and SB denotes smoothed, sentence-level BLEU (Chen and Cherry, 2014). This setting of Δ should act to penalize erroneous predictions with a relatively low sentence-level BLEU score more than those with a relatively high sentence-level BLEU score. In Table 4 we show our final results and those from Ranzato et al. (2016).[8] While we start with an improved baseline, we see similarly large increases in accuracy as those obtained by DAD and MIXER, in particular when $K_{te} > 1$.

We further investigate the utility of these sequence-level costs in Table 5, which compares using sentence-level BLEU costs in defining Δ with using 0/1 costs. We see that the more sophisticated sequence-level costs have a moderate effect on BLEU score.

	Machine Translation (BLEU)		
	$K_{te} = 1$	$K_{te} = 5$	$K_{te} = 10$
0/1-Δ	25.73	28.21	27.43
SB-Δ	25.99	28.45	27.58

Table 5: BLEU scores obtained on the machine translation development data when training with $\Delta(\hat{y}_{1:t}^{(k)}) = 1$ (top) and $\Delta(\hat{y}_{1:t}^{(k)}) = 1 - \text{SB}(\hat{y}_{r+1:t}^{(K)}, y_{r+1:t})$ (bottom), and $K_{tr} = 6$.

Timing Given Algorithm 1, we would expect training time to increase linearly with the size of the beam. On the above MT task, our highly tuned seq2seq baseline processes an average of 13,038 tokens/second (including both source and target tokens) on a GTX 970 GPU. For beams of size K_{tr} = 2, 3, 4, 5, and 6, our implementation processes on average 1,985, 1,768, 1,709, 1,521, and 1,458 tokens/second, respectively. Thus, we appear to pay an initial constant factor of ≈ 3.3 due to the more complicated forward and backward passes, and then training scales with the size of the beam. Because we batch beam predictions on a GPU, however, we find that in practice training time scales sub-linearly with the beam-size.

6 Conclusion

We have introduced a variant of seq2seq and an associated beam search training scheme, which addresses exposure bias as well as label bias, and moreover allows for both training with sequence-level cost functions as well as with hard constraints. Future work will examine scaling this approach to much larger datasets.

Acknowledgments

References

[Andor et al.2016] Daniel Andor, Chris Alberti, David Weiss, Aliaksei Severyn, Alessandro Presta, Kuzman

Ganchev, Slav Petrov, and Michael Collins. 2016. Globally normalized transition-based neural networks. *ACL*.

[Bahdanau et al.2015] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *ICLR*.

[Bahdanau et al.2016] Dzmitry Bahdanau, Philemon Brakel, Kelvin Xu, Anirudh Goyal, Ryan Lowe, Joelle Pineau, Aaron Courville, and Yoshua Bengio. 2016. An Actor-Critic Algorithm for Sequence Prediction. *CoRR*, abs/1607.07086.

[Bengio et al.2015] Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. 2015. Scheduled sampling for sequence prediction with recurrent neural networks. In *Advances in Neural Information Processing Systems*, pages 1171–1179.

[Björkelund and Kuhn2014] Anders Björkelund and Jonas Kuhn. 2014. Learning structured perceptrons for coreference Resolution with Latent Antecedents and Non-local Features. *ACL, Baltimore, MD, USA, June*.

[Cettolo et al.2014] Mauro Cettolo, Jan Niehues, Sebastian Stüker, Luisa Bentivogli, and Marcello Federico. 2014. Report on the 11th iwslt evaluation campaign. In *Proceedings of IWSLT, 20014*.

[Chang et al.2015] Kai-Wei Chang, Hal Daumé III, John Langford, and Stephane Ross. 2015. Efficient programmable learning to search. In *Arxiv*.

[Chen and Cherry2014] Boxing Chen and Colin Cherry. 2014. A systematic comparison of smoothing techniques for sentence-level bleu. *ACL 2014*, page 362.

[Chen and Manning2014] Danqi Chen and Christopher D Manning. 2014. A fast and accurate dependency parser using neural networks. In *EMNLP*, pages 740–750.

[Cho et al.2014] KyungHyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. 2014. On the properties of neural machine translation: Encoder-decoder approaches. *Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*.

[Collins and Roark2004] Michael Collins and Brian Roark. 2004. Incremental parsing with the perceptron algorithm. In *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, page 111. Association for Computational Linguistics.

[Daumé III and Marcu2005] Hal Daumé III and Daniel Marcu. 2005. Learning as search optimization: approximate large margin methods for structured prediction. In *Proceedings of the Twenty-Second International Conference on Machine Learning (ICML 2005)*, pages 169–176.

[Daumé III et al.2009] Hal Daumé III, John Langford, and Daniel Marcu. 2009. Search-based structured prediction. *Machine Learning*, 75(3):297–325.

[Duchi et al.2011] John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *The Journal of Machine Learning Research*, 12:2121–2159.

[Filippova et al.2015] Katja Filippova, Enrique Alfonseca, Carlos A Colmenares, Lukasz Kaiser, and Oriol Vinyals. 2015. Sentence compression by deletion with lstms. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 360–368.

[Hochreiter and Schmidhuber1997] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Comput.*, 9:1735–1780.

[Huang et al.2012] Liang Huang, Suphan Fayong, and Yang Guo. 2012. Structured perceptron with inexact search. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 142–151. Association for Computational Linguistics.

[Kingsbury2009] Brian Kingsbury. 2009. Lattice-based optimization of sequence classification criteria for neural-network acoustic modeling. In *Acoustics, Speech and Signal Processing, 2009. ICASSP 2009. IEEE International Conference on*, pages 3761–3764. IEEE.

[Lafferty et al.2001] John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning (ICML 2001)*, pages 282–289.

[Liu et al.2015] Yijia Liu, Yue Zhang, Wanxiang Che, and Bing Qin. 2015. Transition-based syntactic linearization. In *Proceedings of NAACL*.

[Luong et al.2015] Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015*, pages 1412–1421.

[Mikolov et al.2013] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119.

[Papineni et al.2002] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association*

for computational linguistics, pages 311–318. Association for Computational Linguistics.

[Pham et al.2014] Vu Pham, Théodore Bluche, Christopher Kermorvant, and Jérôme Louradour. 2014. Dropout improves recurrent neural networks for handwriting recognition. In *Frontiers in Handwriting Recognition (ICFHR), 2014 14th International Conference on*, pages 285–290. IEEE.

[Ranzato et al.2016] Marc'Aurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremba. 2016. Sequence level training with recurrent neural networks. *ICLR*.

[Ross et al.2011] Stéphane Ross, Geoffrey J. Gordon, and Drew Bagnell. 2011. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 627–635.

[Sak et al.2014] Hasim Sak, Oriol Vinyals, Georg Heigold, Andrew W. Senior, Erik McDermott, Rajat Monga, and Mark Z. Mao. 2014. Sequence discriminative distributed training of long short-term memory recurrent neural networks. In *INTERSPEECH 2014*, pages 1209–1213.

[Schmaltz et al.2016] Allen Schmaltz, Alexander M Rush, and Stuart M Shieber. 2016. Word ordering without syntax. *arXiv preprint arXiv:1604.08633*.

[Serban et al.2016] Iulian Vlad Serban, Alessandro Sordoni, Yoshua Bengio, Aaron C. Courville, and Joelle Pineau. 2016. Building end-to-end dialogue systems using generative hierarchical neural network models. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, pages 3776–3784.

[Shen et al.2016] Shiqi Shen, Yong Cheng, Zhongjun He, Wei He, Hua Wu, Maosong Sun, and Yang Liu. 2016. Minimum risk training for neural machine translation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016*.

[Srivastava et al.2014] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958.

[Sutskever et al.2011] Ilya Sutskever, James Martens, and Geoffrey E Hinton. 2011. Generating text with recurrent neural networks. In *Proceedings of the 28th International Conference on Machine Learning (ICML)*, pages 1017–1024.

[Sutskever et al.2014] Ilya Sutskever, Oriol Vinyals, and Quoc VV Le. 2014. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, pages 3104–3112.

[Venugopalan et al.2015] Subhashini Venugopalan, Marcus Rohrbach, Jeffrey Donahue, Raymond J. Mooney, Trevor Darrell, and Kate Saenko. 2015. Sequence to sequence - video to text. In *ICCV*, pages 4534–4542.

[Vinyals et al.2015] Oriol Vinyals, Łukasz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever, and Geoffrey Hinton. 2015. Grammar as a foreign language. In *Advances in Neural Information Processing Systems*, pages 2755–2763.

[Voigtlaender et al.2015] Paul Voigtlaender, Patrick Doetsch, Simon Wiesler, Ralf Schluter, and Hermann Ney. 2015. Sequence-discriminative training of recurrent neural networks. In *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*, pages 2100–2104. IEEE.

[Watanabe and Sumita2015] Taro Watanabe and Eiichiro Sumita. 2015. Transition-based neural constituent parsing. *Proceedings of ACL-IJCNLP*.

[Xu et al.2015] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron C. Courville, Ruslan Salakhutdinov, Richard S. Zemel, and Yoshua Bengio. 2015. Show, attend and tell: Neural image caption generation with visual attention. In *ICML*, pages 2048–2057.

[Yazdani and Henderson2015] Majid Yazdani and James Henderson. 2015. Incremental recurrent neural network dependency parser with search-based discriminative training. In *Proceedings of the 19th Conference on Computational Natural Language Learning, (CoNLL 2015)*, pages 142–152.

[Zaremba et al.2014] Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. 2014. Recurrent neural network regularization. *CoRR*, abs/1409.2329.

[Zhang and Clark2011] Yue Zhang and Stephen Clark. 2011. Syntax-based grammaticality improvement using ccg and guided search. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 1147–1157. Association for Computational Linguistics.

[Zhang and Clark2015] Yue Zhang and Stephen Clark. 2015. Discriminative syntax-based word ordering for text generation. *Computational Linguistics*, 41(3):503–538.

[Zhou et al.2015] Hao Zhou, Yue Zhang, and Jiajun Chen. 2015. A neural probabilistic structured-prediction model for transition-based dependency parsing. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics*, pages 1213–1222.