Adaptive Beam Search in Sequence-to-Sequence Models

Yu-Hsiang Lin Shuxin Lin Hai Pham
Language Technologies Institute
Carnegie Mellon University
{*yuhsianl*, *shuxinl*, *htpham*}@andrew.cmu.edu

Abstract

Natural language processing tasks usually require to decode the most likely output sequence which involves searching through all the possible output sequences based on their likelihood. Beam search is a heuristic search algorithm used in the test-time decoding of sequence-to-sequence model. While it might have an advantage over greedy search for accuracy, it leads to time and resource hunger due to keeping and searching through a fixed amount of beams at each step as the beam size increases. We aim to address this problem and propose methods of adapting the beam size for decoding with help of deterministic agent and reinforcement learning, in particular for the *decoding phase* in Seq2Seq model. In the experiments we show that the proposed adaptive beam search strategies yield better results on two NLP tasks compared with baseline models.

1 Introduction

Since its invention, sequence-to-sequence model (Seq2Seq) (Sutskever et al., 2014) has been a go-to model for many translation-related tasks. Despite its great successes in many domains, how to train and decode seq2seq model is still an open problem because of the drawback of traditional maximum likelihood training which is, most of the cases, unable to find the maximum-a-posteriori of a to-be-decoded single sentence over the whole corpus.

Amongst many heuristic approaches to remediate that problem, *greedy search* and *beam search* are the most popular. While greedy search is known for its lightweight, elegant characteristics, beam search is generally better in practice by considering not only the best-scored word at each time step but maintaining a window of best words. In this paper, we will revisit recent researches on related topic in Section 2 and address the disadvantages of previous Seq2seq model using beam search and introduce our improvement with attention in Section 3. Our novel dynamic beam search algorithms will be discussed in Section 4 and 5. We also present our experimental results and analysis in Section 6 and 7 .

2 Related Work

While beam search is considered the de-factor approach (Sutskever et al., 2014), greedy search, if designed properly, can yield a comparable performance, if not better in some cases, while having a much more lightweight architecture. Goyal et al. (2017a) proposed an approximated version of greedy search over the scheduled sampling training procedure (Bengio et al., 2015). Unlike tackling with decoding phase solely, another useful approach to improve seq2seq is to design a better architecture or technique of helping decode right on the training phase. One widely-employed approach is to convert it into an imitation learning problem (Daumé et al., 2009; Ross et al., 2011; Bengio et al., 2015) where expert guidance from human is injected to make the agent more robust and efficient. A naturally connected method is to use reinforcement learning (Sutton and Barto, 1998) which employs a reward-based loss instead of maximum likelihood-based (Ranzato et al., 2015; Gu et al., 2017c), giving rise to a new family of techniques which is fitted to the discrete text domain.

While discriminative training is the straightforward method for seq2seq training, another generalized method is to pose it as a generative model. Amongst such solutions, Generative Adversarial

Network (Goodfellow et al., 2014) broadly used for diverse tasks, predominantly in generating images (Radford et al., 2015; Berthelot et al., 2017; Zhang et al., 2017; Karras et al., 2017; Li et al., 2017) and videos (Vondrick et al., 2016) based on what model learned from training, or *translating* them given a style of images (Mirza and Osindero, 2014; Isola et al., 2017; Kim et al., 2017; Mechrez et al., 2017; Luan et al., 2017; Zhu et al., 2017; Ma et al., 2018) and videos (Ruder et al., 2016; Liu et al., 2017). Despite the booming trend of GAN, its application to text domain faces a difficult obstacle of inherent discrete properties of text domain. Nonetheless, there have been successes of translating text from a style to another to deal with discrete texts (Hjelm et al., 2017; Yu et al., 2017; Shen et al., 2017).

Inspired by GAN's design, similar approaches have been made to seq2seq in conjunction with reinforcement learning (Kusner and Hernández-Lobato, 2016; Yu et al., 2017; Gu et al., 2017a,b). And although not directly connected, actor-critic setting which shares a close equivalence with GAN (Pfau and Vinyals, 2016), has been also employed to replace maximum-likelihood method (Bahdanau et al., 2017). And while sharing the same methodology in that we improve the decoding performance by making the model learn to decode right on the training phase, our approach still sticks to maximum likelihood method objective.

While reinforcement learning can yield a fast decoding model, training with maximum likelihood has its own merit of being simple yet comparably efficient. For such approach, some attempts to make the model learn how to decode right on the training phase have also taken place. There were some solutions that optimize beam search in discrete space such as from Wiseman and Rush (2016); Andor et al. (2016) whose target is to get rid of label bias problem and design a model that is globally–rather than locally–normalized. Another work, from which our work extends, instead aims at design a new surrogate training objective to convert from discrete space into a continuous approximation of the beam search (Goyal et al., 2018). In detail, because using beam search right at training phase largely degrades the performance due to its resources consumption and its search space, we plan to use a tactic of dynamic beam search (Buckman et al., 2016) to make the training faster while retaining its efficacy.

3 Seq2Seq Tagging Model

Seq2Seq (Sutskever et al., 2014) has been a well-known model for machine translation and the related style-transfer tasks that involve text processing, which naturally requires a sequential model. Unlike those common tasks that employ Seq2Seq to predict variable-length inputs and outputs, we tackle with the Name Entity Recognition (NER) and CCG Supertagging tasks in which the input and output are of the same length, i.e. input sequence $X = \{x_1, x_2, ..., x_T\}$ and output sequence $Y = \{y_1, y_2, ..., y_T\}$. This natural basis leads to a little difference in our Seq2Seq formulation, as well as how to perform attention effectively. We will discuss the new encoder and decoder in our Seq2seq model (Figure 1).

3.1 Seq2Seq Encoder

The encoder introduces latent variable Z which is intermediate connection between X and Y, and models the probability $P(Z|X)$ where

$$P(Y|X) = \sum P(Z|X)P(Y|Z,X). \tag{1}$$

But there is an inherent problem specific to text domain is that the dimensionality of X is too high to model efficiently. A popular solution is to learn an non-linear word embedding space E_X that shares the similarity of word relation with X but in much smaller dimension. Word embedding is the vector representation of a word and served as the input of encoder. In order to capture the syntactic and semantic relations among words, pre-trained word embeddings are used. For English datasets, we researched two widely used pre-trained word embeddings - Word2Vec (Mikolov et al., 2013) and GloVe (Pennington et al., 2014). For German datasets, we tried fastText (Joulin et al., 2016) and the other version provided by (Goyal et al., 2017b).

As usual for text sequence, the encoder employs an RNN, which model output at each time step as:

$$(h_t^{enc}, c_t^{enc}) = RNN(h_{t-1}^{enc}, x_t) \tag{2}$$

where h_i and c_i denotes the hidden and cell state at time step t respectively. Thus the encoder output Z is the last hidden state h_T and cell state c_T as:

$$Z = (h_T^{enc}, c_T^{enc}) = RNN(h_{T-1}^{enc}, x_T^{enc}) \tag{3}$$

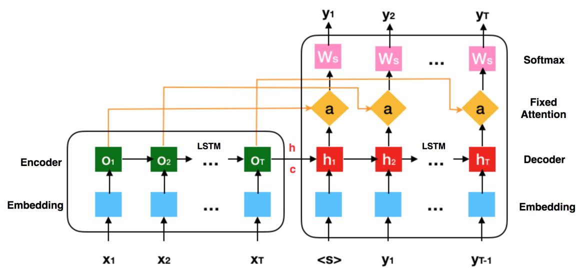

Figure 1: Architecture of Seq2Seq model with fixed attention. The special token that denotes a beginning of a sentence is denoted as <s>. At each time step, the output h_t^{dec} only looks back the corresponding input of the same time step h_t^{enc} (denoted as o_t in this figure).

3.2 Seq2Seq Decoder

As mentioned, the decoder will take encoder's output Z, and try to decode each token $\tilde{y}_t$ corresponding to x_t. It models the conditional probability:

$$P(Y|Z,X) = \prod_{t=1}^{T} p(Y_t|Z, Y_1, ..., Y_{t-1}) \quad (4)$$

with a note that Y_0 is the special token that denotes the start of a sentence.

The Seq2Seq training target is to find the best translation sequence which is as close to the ground truth Y as possible, or formally:

$$\widehat{Y} = \arg\max_{Y} p(Y|Z,X) \quad (5)$$

This is the phase where we employ the searching algorithm such as greedy search or beam search to decode each token. But essentially, the whole process consists of two phases. First is the RNN phase which is similar to the encoder:

$$(h_t^{dec}, c_t^{dec}) = RNN(h_{t-1}^{dec}, y_t) \quad (6)$$

where the initial hidden and cell state is set from the encoder by Equation 3.

Similar to the encoder, the golden label sequence Y is also first casted into an embedding E_Y, and then fed into the decoding RNN network. Note that, slightly different from the encoder, here it is y_{t-1} that is fed into the RNN cell at step t (hence the last label y_T is not fed; the input sequence is $\{y_0, y_1, \dots, y_{T-1}\}$). For the initial input at step $t = 1$, we use a special token $y^* =$ <BEG> to denote the beginning of the label sequence.

The second phase is to convert the hidden state of tokens to the probability over the entire label space:

$$p_{y_t} = softmax(W_s h_t^{dec} + b_s) \quad (7)$$

where W_s and b_s are the shared weight and bias vector on top of the decoder's hidden layer.

3.3 Fixed Attention

Attention model (Bahdanau et al., 2014; Luong et al., 2015) is one of the useful tricks in training Seq2Seq models. The idea is, instead of using the RNN function as in Equation 6 in which every output only depends on current input token and the previous hidden state, we augment this by giving it a chance to look back and calculate the similarity with each token in the encoder *output* sequence. The similarity functions are varied in practice.

Then the Equation 7 for decoder output at step t is transformed to:

$$p_{y_t} = softmax(W_s \cdot attn(h_t^{dec}, h^{enc}) + b_s) \quad (8)$$

where the attention function is the re-weighted output *w.r.t* the encoder output:

$$attn(h_t^{dec}, h^{enc}) = \sum_{i=1}^{T} h_i^{enc} * a_i \quad (9)$$

and the activated weight function $a()$ is the softmax of similarity:

$$a = softmax(sim_1, sim_2, \dots, sim_T) \quad (10)$$

where the similarity function

$$sim_i = f_{sim}(h_t^{dec}, h_i^{enc}) \quad (11)$$

can be a as simple as a dot product $sim_i = h_t^{dec} \cdot h_i^{encT}$ (Luong et al., 2015) or a linear combination between them (Bahdanau et al., 2014), as follow:

$$sim_i = (h_t^{dec} \cdot W^{dec} + h_i^{enc} \cdot W^{enc}) \cdot V \quad (12)$$

In our Tagging problems, we have a valuable prior that at every time step t: the output h_t^{dec} is absolutely similar to h_t^{enc}, i.e. they have the same time step. Consequently, equation 10 will give the probability 1.0 to sim_t and zero to the rest, leading to the change of equation 9 into a simple form:

$$attn(h_t^{dec}, h^{enc}) = h_t^{enc} \quad (13)$$

For implementation, we keep this ”fixed” setting with Bahdanau attention, and so we only need to learn the parameters W^{dec}, W^{enc}, V before passing the decoder output into softmax in 8 for decoding. Figure 1 illustrates our attention setting. And as a trick, we can concatenate h_t^{dec} and h^{enc} so that we only need learn only one matrix W whose output dimension is the concatenated hidden sizes of both the encoder and decoder combined.

3.4 Seq2Seq Training

Finally, to train the model, we use the canonical cross-entropy loss:

$$\mathcal{L} = -log(P(Y|X)) = -\sum_{t=2}^{T} log(p_{y_t}) \quad (14)$$

In detail, the traditional training which uses the teacher-forcing algorithm will feed the correct label y_t to each time step t. But this oftentimes leads to an unwanted behavior where an error in will propagate severely at test time, when the model has no idea about the gold labels. By using this training scheme, the model is not able to learn about how to handle a wrong prediction properly.

One popular method is to employ a schedule where teacher-forcing is used at a random chance at each step, and the other chance is to feed the predicted $\tilde{y_t}$ instead of the gold label y_t and let the model learn to penalize and correct wrong predictions appropriately.

3.5 Test-time decoding

In test time, as in training time, the decoder receives the last hidden and cell states from the encoder as initial states. The initial input at step $t = 1$ is also the universal special token $y^* =$ `<BEG>` or `<s>`. But, different from training time, the inputs at time steps $t = 2, \ldots, T-1$ depend on the predicted label at the previous step. Roughly speaking, we would $replace$ the input y_{t-1} by $\hat{y}_{t-1}$ at test time. The exact way to do so is determined by the decoding algorithm.

3.5.1 Greedy Search

Greedy decoding at test time is to greedily pick the best prediction (with the highest probability) at each time step as the only selection criterion.

Formally, at step $t = 1$, we compute p_{y_1} using (7), and pick the token $\hat{y}_1$ that has the highest probability $p_1 = max(p_{y_1})$ as our predicted label.

Next, we keep an accumulated probability P_t initialized as $P_1 = p_1$. At step $t = 2, 3, \ldots, T_y$, after computing p_t, we first compute the accumulated probability $P_t = p_t * P_{t-1} = \prod_{i=1}^{t-1} p_i$, and then pick the label $\hat{y}_t$ which has the highest probability P_t as our predicted label at step t.

In practice, we consider $log(P_t)$ instead of P_t to prevent the common numeric underflow issue.

3.5.2 Hard Beam Search

Extended from greedy search, hard beam search, although more architecturally and hence computationally expensive, is more widely, and traditionally used (Sutskever et al., 2014) for it is able to prevent the label bias problem in Seq2Seq models. It is also known as *TopK* searching algorithm because at every step, instead of considering the maximum value, it always maintains a window of size K for the best K options. For this reason, greedy search is just its special case where $K = 1$.

For example, at step $t = 1$, instead of picking the only $\hat{y}_1$, we keep the K best labels $\{\hat{y}_1^{(1)}, \hat{y}_2^{(1)}, \ldots, \hat{y}_K^{(1)}\}$. Note that we need a tracking table mapping the words in vocabulary to those labels. Similar to greedy search, we keep the accumulated probabilities $\{P_1, P_2, ..., P_K\}$, which are initialized as $\{p(y_1^{(1)}), p(y_2^{(1)}), \ldots, p(y_K^{(1)})\}$. We start to build a tree with K branches.

In detail, later at time step t, we have K options of inputs $\{y_k^{(t)}\}_K^{k=1}$, and for each input $y_k^{(t-1)}$, we also calculate K best probabilities $\{p(y_{k1}^{(t)}), p(y_{k2}^{(t)}), \ldots, p(y_{kK}^{(t)})\}$. Note that this time step, we still only keep K branches in total, i.e. we keep only the best global cases across all beams (also based on the accumulated log probabilities) along with recording the mapped label associated with that probability. Finally, the highest accumulated sum-log will be chosen and the predicted label sequence is pulled out by backtracking the mapping table.

Compared with greedy search, hard beam search with fixed beam size trades processing time with the accuracy since best scoring sequence will eventually be captured as the beam size grows. However, it always process $B * |V_{label}|$ tokens where B is beam size at each time step, which consumes not only a a large amount of time but memory as well.

4 Adaptive Beam Search by Heuristic Pruning and Growing

To increase the search efficiency as well as the optimality of the resulting sequence, we propose the heuristic rules to prune or grow the beam size of the search tree space during the test-time decoding. The intuition behind the design is that, if we predict that some branches are much more probable than the rest of the branches, it might be reasonable to skip searching through the less probable branches. Furthermore, unlike fixing the beam size, we should also include as many beams as possible if they are comparably probable candidates at each time step.

In this work, we adopt the novel approach that we only incrementally increase or decrease the beam size during the search process. Such design choice facilitates the comparison between the performance of the heuristic rules and the reinforcement learning agent we will be discussing later in Section 5.

In detail, consider that at time step t of the decoding, there are B_{t-1} inputs, $\hat{y}_{t-1}^{(b)}$, $b = 1, \ldots, B_{t-1}$, from the B_{t-1} previous step of the decoder output. By decoding these B_{t-1} inputs, we obtain $B_{t-1} \times |V^y|$ probability predictions for the $|V^y|$ possible targets in B_{t-1} beams. We sort the $B_{t-1} \times |V^y|$ probabilities into a sorted descending list $\{P_1, \ldots, P_{B_{t-1}|V^y|}\}$, with P_1 being the highest probability. We then pick the top B_t elements from the sorted list to form the new beam produced by this decoding step, with the corresponding target words $\{\hat{y}_t^{(1)}, \ldots, \hat{y}_t^{(B_t)}\}$. In the standard beam search where the beam size is kept fixed, we have $B_t = B_{t-1}$ at each step. Then in our proposed adaptive beam search approach, we have the following two deterministic rules to dynamically change the beam size to new B_t at each time step t.

- If $\dfrac{\prod_t' P_{B_{t-1}}}{\prod_t' P_1} \leq r_{\text{low}}^{\text{sen}}$ or $\dfrac{P_{B_{t-1}}}{P_1} \leq r_{\text{low}}^{\text{word}}$, then decrease the beam size: $B_t = B_{t-1} - 1$.

- If $\dfrac{\prod_t' P_{B_{t-1}}}{\prod_t' P_1} > r_{\text{low}}^{\text{sen}}$ or $\dfrac{P_{B_{t-1}}}{P_1} > r_{\text{low}}^{\text{word}}$, then increase the beam size: $B_t = B_{t-1} + 1$.

The primed summation $\prod_t'$ notation is used to indicate that the product is taken over the path of this beam from $t = 1$ to this step. For example, $\prod_t' P_1$ denotes the cumulative probability for the sequence of the top-1 beam. The thresholds $r_{\text{low}}^{\text{sen}}$ and $r_{\text{low}}^{\text{word}}$ are chosen heuristically. In our experiments, we set $r_{\text{low}}^{\text{sen}} = 0.1$ and $r_{\text{low}}^{\text{word}} = 0.1$. Note that to avoid numerical underflow issue, we work in the logarithmic space as we implement the rules just like in the hard beam search.

5 Adaptive Beam Search by Reinforcement Learning

Using the heuristic rules to prune or grow the beam size has a downside that the thresholds of pruning or growing (the parameters $r_{\text{low}}^{\text{sen}}$ and $r_{\text{low}}^{\text{word}}$ in our case) are set by hand, which may require some tuning process to find the most suitable values. Moreover, the thresholds suitable for one task is generally different from those for another task. Thus, additional human labor is needed to identify the suitable thresholds for each new task.

To overcome the shortcomings of the heuristic algorithm above, we propose the second approach to perform the adaptive beam search by introducing an $agent$ to learn when to increase or decrease the beam size, with the help of reinforcement learning.

5.1 Reinforcement learning environment

To apply reinforcement learning in the problems using seq2seq model, we need to first define the environment. In particular, we consider the scenario that there is an agent observing the predicted probability distribution at each step t of the decoder output, and determines what action to take to adjust the beam size.

The state of our environment consists of the following information:

- Log probabilities of the sequences in the current beams, which are the cumulative log probabilities along the paths defining the sequences.

- Log probabilities of the labels in the current beam predicted at the current step; that is, the incremental contribution to the cumulative probabilities of the sequences.

- Current beam size.

The action of the agent is among the three possible options: increasing, decreasing, or keeping the same beam size. We design the environment in such a way so that the task of the agent is greatly simplified; instead of making decisions within all the possible beam sizes, we limit the action space of the agent down to 3. This can make the reinforcement learning more likely to be successful.

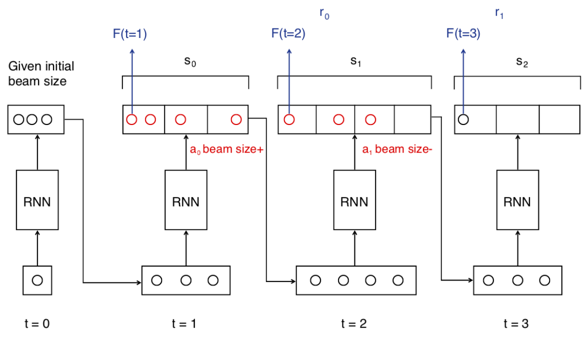

Figure 2: Demonstration of how agent learns to adapt beam size for decoding. At $t = 1$, it learns to increase the beam size from 3 to 4. At $t = 2$, it reduces the beam size back to 3. The agent will choose whether to decrease (-1), keep or increase (+1) the beam size.

The reward is designed to dictate the learning goal of the agent. We define the learning goal of the agent to be consisting of two objectives:

1. To increase the F-score or accuracy, the measure of the task.

2. To decrease the beam size whenever possible to increase the search efficiency.

With these learning goals in mind, we design our reward function to be a linear combination of two terms: The incremental F-score or accuracy, and the amount of decrease of the beam size. That is,

$$r_t = \alpha(\text{score}_t - \text{score}_{t-1}) - \beta(B_t - B_{t-1}), \tag{15}$$

where B_t is the beam size at step t. In our experiments, we set $\alpha = 1$ and $\beta = 0.02$.

Finally, the main interface which interacts directly with the agent is the `step()` function, which takes an input of an action and current state and then return the resulting immediate reward for the being timestep. In our environment, each step would be to determine the tag for a specific word in a sentence. So at timestep t, we receive the current state which includes the most updated beam state, and perform a decoding step which returns the decoded (predicted) tag and the corresponding reward. To facilitate implementation speed, we perform the whole episode at once and populate all relevant states, rewards and actions before returning to the agent to update, which only needs to take place at the end of each episode.

5.2 Training Decoder with Asynchronous Advantage Actor-Critic Method

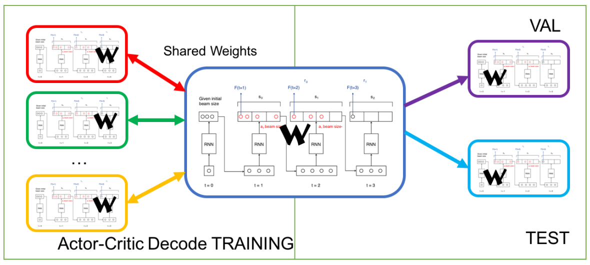

Figure 3: A3C training with multiple training agents which are all able to update the parameters to the shared model. For validation and testing, these parameters are copied to the respective validation and test agents.

Actor-Critic RL (Sutton and Barto, 1998) is known to be a popular method to effectively training a RL agent using REINFORCE (Williams, 1992) because it allows to end the episode early. But we choose Asynchronous Advantage Actor-Critic (A3C) (Mnih et al., 2016) to implement our algorithm for simplicity (compared with other state-of-the-art methods), ability to resolve high variance and high sample complexity in training of Actor-Critic and parallelism which helps speed up the training.

Our A3C architecture is shown in Figure 3. From the environment defined above, we use a shared model and initialize many training agents which share the same optimizer as well, to asynchronously update the model policy network during training. Each episode stands for each sentence containing the tagging task for each token in the sentence.

Formally, we have a main actor which aims to learn the policy function $\pi(a_t|s_t;\theta)$ where θ is the parameter of this policy deep network. We oftentimes approximate this function by a deep neural network where an input is the state, which in our case is a flattened vector of length l and outputs a logits vector of size 3 (action space). To reduce the variance and high sample complexity, we have a critic agent which acts like an adaptive baseline network, which co-progress with the actor to push its accuracy in choosing more precise actions. The critic network leans a value function $V(s_t;\theta_v)$, which outputs a single scalar.

The reward for each time step is traditionally calculated as the cumulative discounted rewards, with the last term is the value of critic network at that time:

$$R_t = \sum_{i=0}^{T-1} \gamma^i r(s_{t+i}, a_{t+i}) + \gamma^T V(s_{t+T}; \theta_v)$$

where T is the episode length, which is the sentence's length being considered and t is the current timestep. Here the sentence is always limited, so we don't have to bound the episode length as in such cases as game playing.

The advantage at each time step is the difference between the discounted reward and the critic value:

$$A(s_t, a_t) = R_t - V(s_t; \theta_v)$$

The policy (actor) loss will be:

$$L_{actor} = \sum_{i=1}^{N} \sum_{t=0}^{T} log\, \pi(a_t|s_t; \theta) \cdot \gamma^t \cdot A(s_t, a_t) + \beta\, H(\pi\, (s_t; \theta)) \tag{16}$$

where H is the entropy of a state with the coefficient β. The critic network has its separate loss which is:

$$L_{critic} = \sum_{i=1}^{N} \sum_{t=0}^{T} (R_t - V(s_t; \theta_v))^2 \tag{17}$$

In terms of implementation, both parameter sets of actor and critic are updated using Adam optimizer after each episode. As noted above, this optimizer is shared between many training agents, each agent has both actor and critic in it. We also clip gradient L2 norm aggressively at the value of 5.0 to make the training more stable.

6 Experiments

Our implementation[1] is built based on PyTorch (Paszke et al., 2017). We conducted all experiments on AWS EC2 as well as private computers. As it is discussed in Section 3, the current Seq2Seq model is built on the encoder-decoder architecture. For encoder, we use one-layer bi-directional LSTM, and for the decoder we use one-layer single-directional LSTM with a fixed attention mechanism. We perform the training using mini-batch stochastic gradient descent with Adam optimizer and use different beam search strategies in testing time. The task we are trying to solve is two common natural language processing tasks, the named entity recognition (NER) and CCG Supertagging.

[1]The Github repository is accessible at: https://github.com/ShuxinLin/nn4nlp_project/.

6.1 Datasets

In order to make our experimental results comparable to the baseline (Goyal et al., 2017b), we use the same datasets as well as evaluation metrics.

For named entity recognition, we use the data of CONLL-2003 shared task (Tjong Kim Sang and De Meulder, 2003) for German language. The data consists of three files per language: one training file and two test files, testa and testb. We use testa as the development data, testb as the final test data. The label/entity has 9 possible values: 8 NER tags and a tag denoting the end of the sentence (EOS). Due to the extremely skewed label distribution towards the default label (O), the widely-used evaluation metrics is F score.

For CCG Supertagging task, we use the data of CCGbank corpus (Hockenmaier and Steedman, 2007). The label has 1321 possible values: 1320 CCG derivation grammar tags and a tag denoting the end of the sentence (EOS). In this task, accuracy of correct predictions is measured.

We perform very minor preprocessing on the data by creating and indexing the vocabulary of each datasets. Given the vocabulary of input words, we construct pre-trained word embedding matrix using GloVe for English and provided word embedding (Goyal et al., 2017b) for German. As can be noticed above, CCG Supertagging have larger label size (search space) than NER task. It leads to the fact that CCG supertagging task is more appropriate for measuring the efficacy of the proposed dynamic beam search methods.

6.2 Results and Analysis

6.2.1 Heuristic Pruning and Growing Results

The results of the heuristic pruning and growing adaptive beam search on the NER and CCG datasets are shown in Table 1 and 2, respectively. From the results of NER dataset, we observe that using heuristic adaptive beam search effectively reduces the search space and the search time, while not sacrificing the F-score. In the case of initial beam size 9, the total number of beams explored during decoding in the adaptive case is only 42% of that in the case of fixed beam size, while the F-score using adaptive search only decreases by less than 0.1%; measured in time, the decoding time is reduced to 52% when using adaptive method. Note that in the NER dataset, greedy search gives higher F-score than the beam search methods do. This is consistent with the results from the baseline. The performance of the adaptive search is better on the CCG dataset. We observe the following:

1. By starting with initial beam size of 1, and allow the beam size to be adaptively changed in the following steps during decoding, we obtain higher accuracy than the greedy search (always keep beam size 1) does.

2. Using fixed beam size, one needs to use beam size 3 to obtain the same accuracy achieved by the adaptive beam search with initial beam size 1, while the latter only needs to explore about 59% of the beam number, and spends only 47% of the time.

3. The F-scores obtained by using different initial beam size for the adaptive search are stable (actually the same value in this experiment), and the total number of beams explored by using initial beam size 3 increases by only 11% compared to that by using initial beam size 1.

6.2.2 A3C Results

The results of the adaptive beam search using the RL agent on the NER and CCG datasets are shown in Tables 3 and 4. For the NER dataset, we observe that the reinforcement learning agent in general obtains lower F-score than that obtained by using greedy, fixed, or heuristic adaptive beam search. We find that the agent tends to learn to decrease the beam size more aggressively than the heuristic rules do. One possible reason for this behavior is that in the NER dataset the learning signals coming from the F-score are sparse, since most of the NER tags are “O” and do not contribute to the F-score. Therefore, the learning signals coming from decreasing the beam size dominates the learning process, and the agent does not learn to adjust the beam size to obtain higher F-score effectively.

In the CCG dataset, we observe the following:

1. In general the agent is able to learn to dynamically adjust the beam size to obtain the accuracy that is higher than that using the greedy search. In the worst case (initial beam size 2), the accuracy is the same as using greedy search. Nevertheless, the accuracy obtained by the agent is still lower than that obtained using the heuristic adaptive beam search.

2. Unlike in the case of NER dataset, the agent here is able to learn to increase the beam size to raise the accuracy. We observe that as the total number of beams explored increases, the accuracy obtained by the agent increases as well. In the worst case (initial beam size 2), the agent does not learn to increase the beam size to get higher accuracy effectively, and the total number of beam size stays at the similar level of that of the greedy search. In that case, it obtains the same accuracy as that obtained by the greedy search as well.

   The possible reason for the more effective learning with the CCG dataset is that the learning signals coming from improving the accuracy is dense in this case. Unlike in the case of NER dataset, every label counts for the accuracy in CCG dataset, therefore the agent constantly obtains incremental reward at every step of decoding, and learns a better correlation between their choice of adjusting beam size and its impact on the accuracy.

3. Larger beam size does not always lead to higher accuracy. Although this phenomenon can also be observed by comparing the greedy search and the beam search, here the situation is more subtle. For example, both starting with initial beam size 3, the fixed-size beam search uses more total beams and obtains higher accuracy than the agent does, while the heuristic beam search uses fewer total beams to also obtain the higher accuracy. This indicates that, in addition to the total number of beams explored, the quality of the decision of when to change the beam size indeed matters. In our case, the agent has not learned the decisions as good as the heuristic ones.

4. The learning of the agent is not stable. The experiment of training the agent with initial beam size 2 is not successful and is significantly worse than the other two cases.

Generally speaking, the RL cannot beat the heuristic methods in terms of quantitative performance (F score or accuracy) and overall runtime since reinforcement learning introduces agent training. The break point here is whether the adaptive beam search is used to solve multiple NLP tasks. If that is not the case, then heuristic method might be a better choice.

Table 1: Results on Named Entity Recognition task using heuristic pruning and growing rules. The F scores and other experimental results of greedy search, hard beam search (beam size = 3, 6, 9) and heuristic adaptive beam search (beam size is initialized as 3, 6, 9) are listed. The last row is the baseline.

	Greedy	Beam 3 Fixed	Beam 3 Adaptive	Beam 6 Fixed	Beam 6 Adaptive	Beam 9 Fixed	Beam 9 Adaptive	Soft Beam
F-score	**58.09**	57.69	57.71	57.76	57.71	57.76	57.71	
Total tokens #	**48,571**	145,713	92,727	291,426	126,759	437,139	182,785	
Avg. beam #	**1**	3	1.95	6	3.16	9	4.86	
Time (sec)	**22**	76	61	132	73	178	92	
(Goyal) F score	54.92	51.34						56.38

Table 2: Results on CCG Supertagging task using heuristic pruning and growing rules. The F scores and other experimental results of greedy search, hard beam search (beam size = 2, 3) and heuristic adaptive beam search (beam size is initialized as 1, 2, 3) are listed.

	Greedy	Beam 1 Adaptive	Beam 2 Fixed	Beam 2 Adaptive	Beam 3 Fixed	Beam 3 Adaptive	Soft Beam
Accuracy	90.50	**90.62**	90.60	**90.62**	**90.62**	**90.62**	
Total tokens #	**52,964**	93,403	105,928	98,100	158,892	103,875	
Avg. beam #	**1**	1.61	2	1.72	3	1.87	
Time (sec)	**22**	87	92	130	184	141	
(Goyal) Accuracy	80.35				82.42		82.00

Table 3: Results on Named Entity Recognition using adaptive beam search by reinforcement learning. The beam size is initialized as 3, 6, 9.

	Beam 3	Beam 6	Beam 9
F-score	**57.66**	57.61	57.52
Total tokens #	**65,619**	105,076	155,466
Avg. beam #	**1.17**	2.00	3.06

Table 4: Results on CCG SuperTagging using adaptive beam search by reinforcement learning. The beam size is initialized as 1, 2, 3.

	Beam 1	Beam 2	Beam 3
Accuracy	90.56	90.50	**90.59**
Total tokens #	100,573	**55,372**	151,832
Avg. beam #	1.80	**0.99**	2.73

7 Conclusion

In this paper, we propose a novel strategy for optimizing the beam search for Seq2Seq model. We implement two adaptive beam search methods - heuristic rules and reinforcement learning - for reducing search space and runtime by changing beam size dynamically in the decoding phase. We have conducted experiments on the NER and CCG Supertagging tasks using our models. Results demonstrate that our method is able to speed up the decoder without losing any quality. All the code is now made available in our Github repository.

References

Daniel Andor, Chris Alberti, David Weiss, Aliaksei Severyn, Alessandro Presta, Kuzman Ganchev, Slav Petrov, and Michael Collins. 2016. Globally normalized transition-based neural networks. *arXiv preprint arXiv:1603.06042*.

Dzmitry Bahdanau, Philemon Brakel, Kelvin Xu, Anirudh Goyal, Ryan Lowe, Joelle Pineau, Aaron Courville, and Yoshua Bengio. 2017. An actor-critic algorithm for sequence prediction. *ICLR*.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

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.

David Berthelot, Tom Schumm, and Luke Metz. 2017. Began: Boundary equilibrium generative adversarial networks. *arXiv preprint arXiv:1703.10717*.

Jacob Buckman, Miguel Ballesteros, and Chris Dyer. 2016. Transition-based dependency parsing with heuristic backtracking. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2313–2318.

Hal Daumé, John Langford, and Daniel Marcu. 2009. Search-based structured prediction. *Machine learning*, 75(3):297–325.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680.

Kartik Goyal, Chris Dyer, and Taylor Berg-Kirkpatrick. 2017a. Differentiable scheduled sampling for credit assignment. *arXiv preprint arXiv:1704.06970*.

Kartik Goyal, Graham Neubig, Chris Dyer, and Taylor Berg-Kirkpatrick. 2017b. A continuous relaxation of beam search for end-to-end training of neural sequence models. *arXiv preprint arXiv:1708.00111*.

Kartik Goyal, Graham Neubig, Chris Dyer, and Taylor Berg-Kirkpatrick. 2018. A continuous relaxation of beam search for end-to-end training of neural sequence models. *AAAI*.

Jiatao Gu, Daniel Jiwoong Im, and Victor OK Li. 2017a. Neural machine translation with gumbel-greedy decoding. *arXiv preprint arXiv:1706.07518*.

Jiatao Gu, Daniel Jiwoong Im, and Victor OK Li. 2017b. Neural machine translation with gumbel-greedy decoding. *arXiv preprint arXiv:1706.07518*.

Jiatao Gu, Graham Neubig, Kyunghyun Cho, and Victor OK Li. 2017c. Learning to translate in real-time with neural machine translation. *EACL*.

R Devon Hjelm, Athul Paul Jacob, Tong Che, Kyunghyun Cho, and Yoshua Bengio. 2017. Boundary-seeking generative adversarial networks. *arXiv preprint arXiv:1702.08431*.

Julia Hockenmaier and Mark Steedman. 2007. Ccgbank: a corpus of ccg derivations and dependency structures extracted from the penn treebank. *Computational Linguistics*, 33(3):355–396.

Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. 2017. Image-to-image translation with conditional adversarial networks. *arXiv preprint*.

Armand Joulin, Edouard Grave, Piotr Bojanowski, Matthijs Douze, Hérve Jégou, and Tomas Mikolov. 2016. Fasttext.zip: Compressing text classification models. *arXiv preprint arXiv:1612.03651*.

T Karras, T Aila, S Laine, and J Lehtinen. 2017. Progressive growing of gans for improved quality. *Stability, and Variation. arXiv preprint*.

Taeksoo Kim, Moonsu Cha, Hyunsoo Kim, Jungkwon Lee, and Jiwon Kim. 2017. Learning to discover cross-domain relations with generative adversarial networks. *arXiv preprint arXiv:1703.05192*.

Matt J Kusner and José Miguel Hernández-Lobato. 2016. Gans for sequences of discrete elements with the gumbel-softmax distribution. *arXiv preprint arXiv:1611.04051*.

Chun-Liang Li, Wei-Cheng Chang, Yu Cheng, Yiming Yang, and Barnabás Póczos. 2017. Mmd gan: Towards deeper understanding of moment matching network. In *Advances in Neural Information Processing Systems*, pages 2200–2210.

Ming-Yu Liu, Thomas Breuel, and Jan Kautz. 2017. Unsupervised image-to-image translation networks. In *Advances in Neural Information Processing Systems*, pages 700–708.

Fujun Luan, Sylvain Paris, Eli Shechtman, and Kavita Bala. 2017. Deep photo style transfer. *CoRR, abs/1703.07511*.

Minh-Thang Luong, Hieu Pham, and Christopher D Manning. 2015. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*.

Shuang Ma, Jianlong Fu, Chang Wen Chen, and Tao Mei. 2018. Da-gan: Instance-level image translation by deep attention generative adversarial networks (with supplementary materials). *arXiv preprint arXiv:1802.06454*.

Roey Mechrez, Eli Shechtman, and Lihi Zelnik-Manor. 2017. Photorealistic style transfer with screened poisson equation. *arXiv preprint arXiv:1709.09828*.

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.

Mehdi Mirza and Simon Osindero. 2014. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pages 1928–1937.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in pytorch.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.

David Pfau and Oriol Vinyals. 2016. Connecting generative adversarial networks and actor-critic methods. *arXiv preprint arXiv:1610.01945*.

Alec Radford, Luke Metz, and Soumith Chintala. 2015. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*.

Marc'Aurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremba. 2015. Sequence level training with recurrent neural networks. *arXiv preprint arXiv:1511.06732*.

Stéphane Ross, Geoffrey 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.

Manuel Ruder, Alexey Dosovitskiy, and Thomas Brox. 2016. Artistic style transfer for videos. In *German Conference on Pattern Recognition*, pages 26–36. Springer.

Tianxiao Shen, Tao Lei, Regina Barzilay, and Tommi Jaakkola. 2017. Style transfer from non-parallel text by cross-alignment. In *Advances in Neural Information Processing Systems*, pages 6833–6844.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112.

Richard S Sutton and Andrew G Barto. 1998. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge.

Erik F. Tjong Kim Sang and Fien De Meulder. 2003. Introduction to the conll-2003 shared task: Language-independent named entity recognition. In *Proceedings of CoNLL-2003*, pages 142–147. Edmonton, Canada.

Carl Vondrick, Hamed Pirsiavash, and Antonio Torralba. 2016. Generating videos with scene dynamics. In *Advances In Neural Information Processing Systems*, pages 613–621.

Ronald J Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. In *Reinforcement Learning*, pages 5–32. Springer.

Sam Wiseman and Alexander M Rush. 2016. Sequence-to-sequence learning as beam-search optimization. *arXiv preprint arXiv:1606.02960*.

Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. 2017. Seqgan: Sequence generative adversarial nets with policy gradient. In *AAAI*, pages 2852–2858.

Han Zhang, Tao Xu, Hongsheng Li, Shaoting Zhang, Xiaolei Huang, Xiaogang Wang, and Dimitris Metaxas. 2017. Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks. In *IEEE Int. Conf. Comput. Vision (ICCV)*, pages 5907–5915.

Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. 2017. Unpaired image-to-image translation using cycle-consistent adversarial networks. *arXiv preprint arXiv:1703.10593*.