When to Finish? Optimal Beam Search for Neural Text Generation (modulo beam size)

Liang Huang Kai Zhao[†] Mingbo Ma {kzhao.hf, liang.huang.sh, cosmmb}@gmail.com
School of EECS, Oregon State University [†]Kai is now with Google Inc., New York
Oregon State University

THE PROBLEM – WHEN TO STOP?

Neural beam search (e.g. NMT) is great, but nobody knows when/how to stop!

- greedy search: easy, just stop at the first </s>
- beam search: has to return a complete hypothesis which ends at </s>,
- but how to guarantee it's the best-scoring one?
- it's possible some currently incomplete hypothesis can lead to high-scoring complete hypothesis
- when can you guarantee no other complete hypotheses (in the future) can score better?

Existing approaches can't establish optimality:

1. **RNNsearch**: shrink beam heuristic: decrement beam size for each complete hypothesis in beam (too hacky)
2. **OpenNMT-py**: stop whenever the top item at any step is a complete one, and return it (we'll show it's neither optimal nor efficient)

OUR CONTRIBUTIONS

Our first algorithm:

1. we devise the first provably-optimal neural beam search algorithm (optimal modulo beam size)
2. this means if you follow standard beam search pruning, then for a given beam size, you can't find a higher-scoring complete hypothesis than ours
3. our algorithm is not only optimal, but also efficient: it finishes beam search earlier than OpenNMT-py

Our second algorithm:

1. but higher model score leads to short translations!
2. we devise a **bounded length reward** to encourage longer translations
3. a variant of our optimal beam search is still optimal with bounded length reward

BEAM SEARCH BACKGROUND

$$\mathbf{y}^* = \operatorname*{argmax}_{\mathbf{y}:comp(\mathbf{y})} p(\mathbf{y} \mid \mathbf{x}) = \operatorname*{argmax}_{\mathbf{y}:comp(\mathbf{y})} \prod_{i \le |\mathbf{y}|} p(y_i \mid \mathbf{x}, \mathbf{y}_{<i})$$

where $comp(\mathbf{y}) \triangleq (\mathbf{y}_{|\mathbf{y}|} = \texttt{</s>})$ returns the completeness of a hypothesis, and beam search expands B_{i-1} to B_i:

$$B_0 = [\langle \texttt{<s>},\ p(\texttt{<s>} \mid \mathbf{x})\rangle]$$

$$B_i = \overset{b}{\mathbf{top}}\{\langle \mathbf{y}' \circ y_i,\ s \cdot p(y_i \mid \mathbf{x}, \mathbf{y})\rangle \mid \langle \mathbf{y}', s\rangle \in B_{i-1}\}$$

FIRST ALGORITHM: OPTIMAL BEAM SEARCH (modulo beam size)

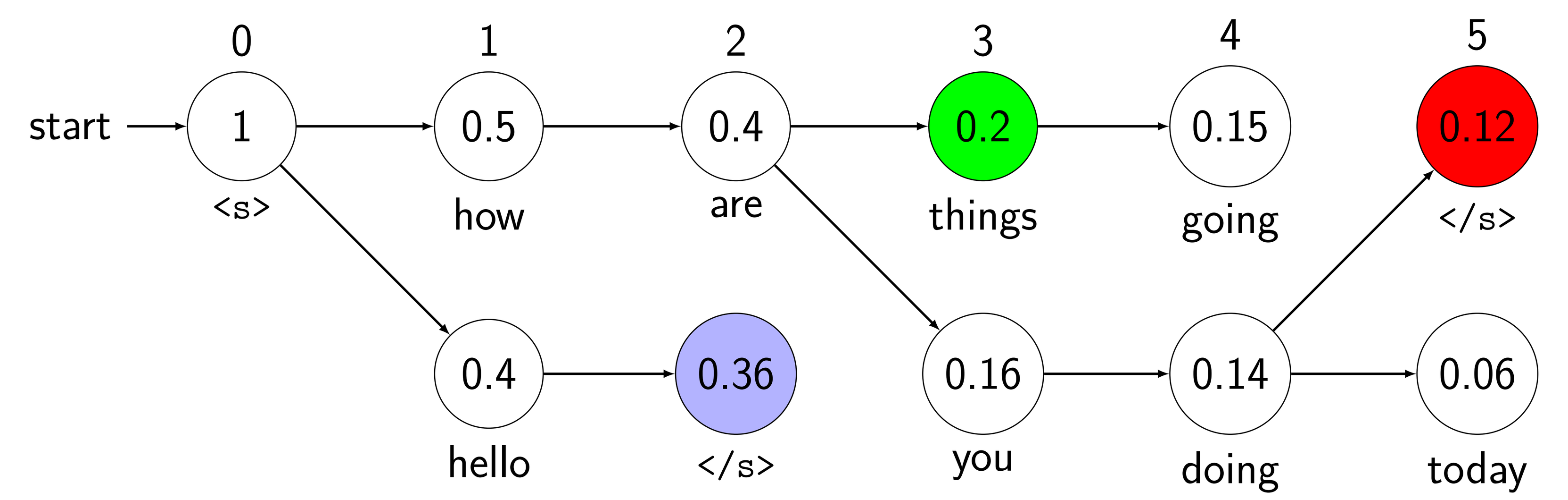
Current Candidate: define $best_{[0:i]} \triangleq \max\{\mathbf{y} \in \cup_{j \le i} B_j \mid comp(\mathbf{y})\}$ to be **best complete hypothesis so far**.

Stopping Criteria: $B_{i,1} \le best_{[0:i]}$, i.e., when the top-scoring item in the current step i is already worse than the best complete hypothesis so far. Then return the latter ($best_{[0:i]}$).

Optimality Proof: $B_{i,j} \le B_{i,1} \le best_{[0:i]}$ for all items $B_{i,j}$ in beam B_i. Descendants of these items in future steps are even worse, so all items in the current and future steps are no better than $best_{[0:i]}$.

OpenNMT-py's Method: $comp(B_{i,1})$, i.e., when the top-scoring item in any step is complete. Return it.

Efficiency: Our algorithm terminates **no later than** OpenNMT-py (which is neither optimal nor efficient).



our optimal beam search stops at step 3 (triggered by $0.2 < 0.36$) and returns the best candidate so far "hello </s>" (score 0.36), while OpenNMT-py stops at step 5 and returns "how are you doing </s>" (score 0.12).

SECOND ALGORITHM: Optimal Beam Search w/ Bounded Length Reward

The Problem: Higher-scoring hypotheses lead to extremely short translations.

Existing Solutions: However, both break the optimality of our optimal beam search algorithm!

1. score normalization: the score of a hypothesis / its length; aiming for optimal average per-step score. used in RNNsearch (Bahdanau et al., 2014) and Google NMT (Wu et al., 2016).
2. length reward: explicit reward for each word; used in Baidu NMT (He et al., 2016).

Our Bounded Length Reward: We only reward each target word up to an estimated "optimal" length, proportional to source length $|\mathbf{x}|$; in Chinese-to-English exps we use $1.27 \cdot |\mathbf{x}|$ estimated on the dev set.

Modified Optimal Beam Search: use new score $\tilde{sc}(\mathbf{y}) \triangleq sc(\mathbf{y}) + r \cdot \min\{c|\mathbf{x}|, |\mathbf{y}|\}$, where $c = 1.27$, and we tune the length reward r on dev set. **Optimality Proof:** similar to A* with admissible heuristics.

EXPERIMENTAL SETUP

1. Based on OpenNMT-py, a PyTorch reimplementation of Torch-based OpenNMT (Klein et al., 2017). PyTorch made it much easier than Theano-based RNNsearch.
2. 1M Chinese-English sentence pairs (28M/23M tokens) for training (also tried 2M sentence pairs).
3. Used byte-pair encoding (BPE) (Senrich et al., 2015) to reduce vocabulary sizes from 112k/93k to 18k/10k. BPE improved BLEU score (by at least 2+) and reduced training time.
4. Chinese to English: NIST 06 newswire portion (616 sentences) for dev; NIST 08 newswire portion (691 sentences) for test; case-insensitive 4-reference BLEU-4 scores.
5. 20 epochs local greedy training (excluding (15%) sentences w/ 50+ source tokens). About an hour per epoch on Geforce 980 Ti, epoch 15 reaches the lowest perplexity on the dev set (9.10).
6. Baseline is very competitive: 29.2 BLEU with $b = 1$ (greedy), 33.2 with default $b = 5$.

By-product: We also found and fixed an obscure but serious bug in OpenNMT-py's beam search code (not related to this paper), which boosts BLEU scores by about +0.7 in all cases.

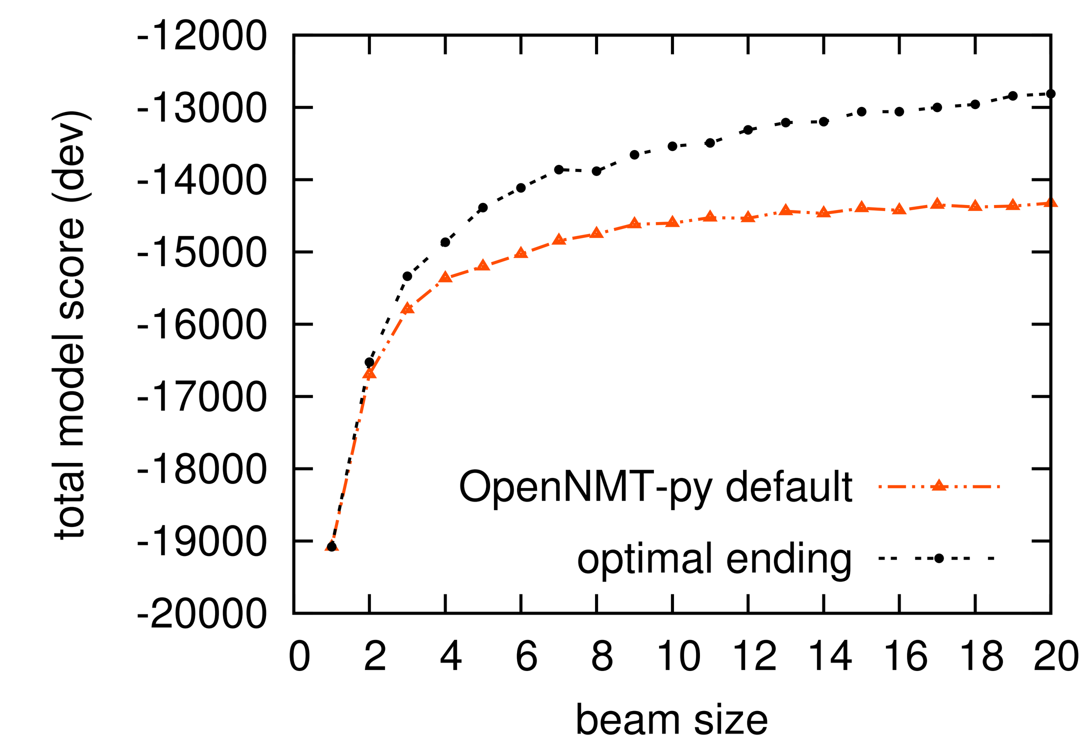


Figure 1: Comparison between optimal beam search and OpenNMT-py's default search, in terms of search quality (model score, ↑ is better).

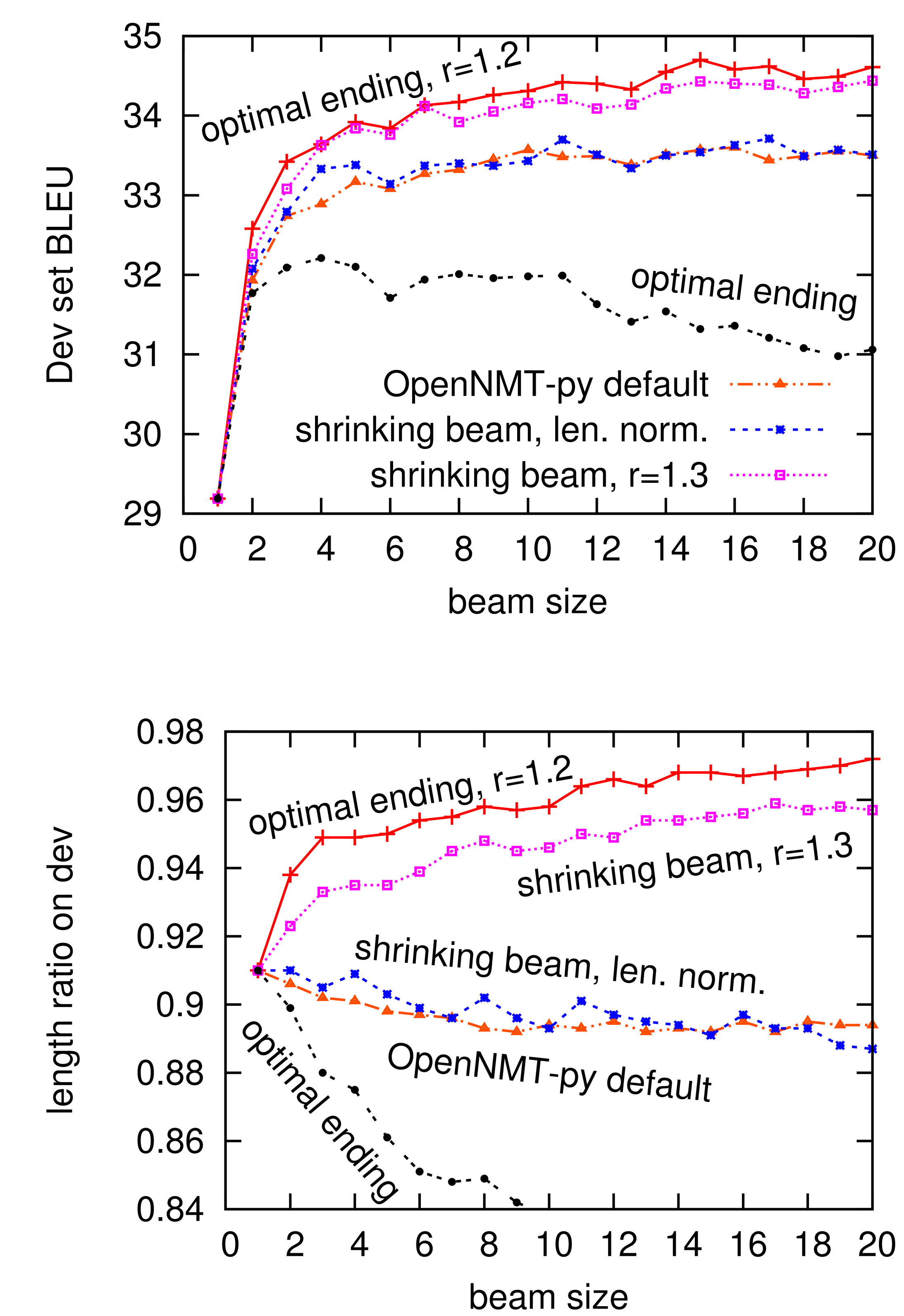


Figure 2: BLEU score and length ratio against beam size (on dev).

decoder	b	dev	test
Moses	70	30.14	29.41
OpenNMT-py default	16	33.60	29.75
shrinking, len. norm.	17	33.71	30.11
shrinking, reward r=1.3	15	34.42	30.37
optimal beam search, r=1.2	15	**34.70**	**30.61**

Table 1: Final BLEU scores on test set using best settings from dev set.