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:

- RNNsearch: shrink beam heuristic: decrement beam size for each complete hypothesis in beam (too hacky)
- 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:

- we devise the first provably-optimal neural beam search algorithm (modulo beam size)
  - this means if you follow standard beam search
    - but higher model score leads to short translations!
- we devise the first provably-optimal neural beam search algorithm

Our second algorithm:

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

BEAM SEARCH BACKGROUND

$$y^* = \text{argmax} p(y|x) = \text{argmax} \prod_i p(y_i|x, y_{<i})$$

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

$$B_i = \{<s>, p(x) \}$$

$$B_i = \text{top}(\{y \mid p(y_i|x, y_{<i}) \mid (y', s) \in B_{i-1}\})$$

FIRST ALGORITHM: OPTIMAL BEAM SEARCH (modulo beam size)

Current Candidate: define $\text{best}_{i,j} \triangleq \max \{ y \in B_i | \text{comp}(y) \}$ to be the best complete hypothesis so far.

Stopping Criteria: $B_i \leq \text{best}_{i,j}$, i.e., when the top-scoring item in any step $i$ is already worse than the best complete hypothesis so far. Then return the latter ($\text{best}_{i,j}$).

Optimality Proof: $B_i \leq \text{best}_{i,j}$ for all items $B_i$ 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 $\text{best}_{i,j}$.

OpenNMT-py’s Method: $\text{comp}(B_i)$, 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).

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!

- 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).
- 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 $|x|$, in Chinese-to-English exps we use $1.27 \cdot |x|$ estimated on the dev set.

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

EXPERIMENTAL SETUP

- Based on OpenNMT-py, a PyTorch reimplementation of Torch-based OpenNMT (Klein et al., 2017).
- PyTorch made it much easier than Theano-based RNNsearch.
- 1M Chinese-English sentence pairs (28M/23M tokens) for training (also tried 2M sentence pairs).
- Used byte-pair encoding (BPE) (Sennrich 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.
- 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.
- 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).
- Baseline is very competitive: 29.2 BLEU with $\theta = 1$ (greedy), 33.2 with default $\theta = 5$.
- 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.

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

<table>
<thead>
<tr>
<th>decoder</th>
<th>$b$</th>
<th>dev</th>
<th>test</th>
</tr>
</thead>
<tbody>
<tr>
<td>Moses</td>
<td></td>
<td>79.30</td>
<td>29.41</td>
</tr>
<tr>
<td>OpenNMT-py default</td>
<td></td>
<td>63.60</td>
<td>29.75</td>
</tr>
<tr>
<td>shrinking, len. norm.</td>
<td></td>
<td>73.71</td>
<td>30.11</td>
</tr>
<tr>
<td>OpenNMT-py default</td>
<td></td>
<td>63.60</td>
<td>29.75</td>
</tr>
<tr>
<td>shrinking, reward $r=1.3$</td>
<td></td>
<td>15.34</td>
<td>30.37</td>
</tr>
<tr>
<td>optimal beam search, $r=1.2$</td>
<td></td>
<td>15.34</td>
<td>30.37</td>
</tr>
</tbody>
</table>

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