OPTIMAL SEARCH FOR MINIMUM ERROR RATE TRAINING

Authors: Michel Galley & Chris Quirk
(Microsoft Research)
Presenter: Avneesh Saluja
The Claim

- Och’s MERT is not exact
  - I will provide a brief MERT review
- This paper: exact search of MERT search space via linear programming
  - Concurrent optimization of all dimensions
- How does this perform vs. Och’s MERT?
MERT for MT: Primer

- Tune parameter weights to directly optimize evaluation metric

\[
\hat{w} = \arg \min_w \left\{ \sum_{s=1}^{S} E(r_s, \hat{e}(f_s; w)) \right\}
\]

\[
= \arg \min_w \left\{ \sum_{s=1}^{S} \sum_{n=1}^{N} E(r_s, e_{s,n}) \delta(e_{s,n}, \hat{e}(f_s; w)) \right\} \text{ s.t.}
\]

\[
\hat{e}(f_s; w) = \arg \max_{n \in \{1, \ldots, N\}} \{w^T h\}_{s,n}
\]

Naïve MERT

- Non-convex, unsmooth: Powell search
  - Multiple starting points used
- Log-linear formulation: \( p(x) = \exp \left\{ \sum_{i=1}^{N} \lambda_i h_i(x) \right\} \)
- Best translation:

\[
x^{\ast}(\lambda_1, \ldots, \lambda_n) = \arg \max_x \exp \left\{ \sum_{i=1}^{N} \lambda_i h_i(x) \right\} \Rightarrow
\]

\[
x^\ast(\lambda_c) = \arg \max_x \exp \left\{ \sum_{i=1}^{N} \lambda_i h_i(x) \right\} \text{ where } u(x) = \sum_{i \neq c} \lambda_i h_i(x) \]

- How to explore parameter space?
  - Grid: trade-off between speed & accuracy
  - Finite approximation
Och’s Trick

- Translation ranking only changes at intersections of translation lines!

- Multiple sentences:
  - aggregate intersections across sentences
  - For each intersection point, compute error (re-rank and select 1-best)
  - Return interval with best error score
LP-MERT: one sentence case

- Och’s MERT: great, but optimizing one parameter at a time?
  - Search over a larger subspace of parameter combos, not just line search
- Build convex hull of n-best list, iterate through extreme points
  - CH construction algorithms exponential in dimension
- Resort to LP with interior point methods (poly in dimension) to find extreme points
And more than 1 sentence?

- LP Formulation: return 0 if interior point
- Naïve approach: enumerate all possible hypothesis combinations across all sentences
- Smarter approach: merging convex hulls to maintain convexity
- Extreme point determination: $O(NS)$ # points vs. $O(N^S)$
Other Tricks and Speedups

• Takes $O(NS)$ points to determine if a point is extreme. Need to do this for $O(NS)$ possible combinations

• Trick 1: lazy enumeration (ordering of combos)

• However, not quite enough
Binary lazy enumeration

• Use divide-and-conquer:
Final Algorithm

Inputs:
- N*S feature vectors
- N*S BLEU scores

Sort each N-best list

Hypothesis Combination Matrix (Frontier)

Subtree 1

Subtree 2

Linear Program (extreme point, convex hull)

Output:
Final Weights

Finding the extreme point (over all sentences) with lowest loss

Move up to next level of tree

Combiner: is point extreme?

yes

no

Try next “best” combo in matrix
Approximations that we need

• Cosine similarity check (with reference vector):
  \[ \cos(\hat{\mathbf{w}}, \mathbf{w}_0) \geq t \]

• Beam search: prune with respect to current best parameter vector (when combining, check model score)
Experimental Setup

- Tree-to-string model
- 13 features in total
  - Standard PM and LM features, re-ordering, function word insertion/deletion, insertion/deletion counts, target length
- N-best size = 100
  - Same combined N-best lists
- WMT 2010 English → German (1.6 million sentence pairs)
  - 2009 test: tuning
  - 2010 test: test
  - One reference translation
The D&C Speedup

<table>
<thead>
<tr>
<th>length</th>
<th>tested comb.</th>
<th>total comb.</th>
<th>order</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>639,960</td>
<td>$1.33 \times 10^{20}$</td>
<td>$O(N^8)$</td>
</tr>
<tr>
<td>4</td>
<td>134,454</td>
<td>$2.31 \times 10^{10}$</td>
<td>$O(2N^4)$</td>
</tr>
<tr>
<td>2</td>
<td>49,969</td>
<td>430,336</td>
<td>$O(4N^2)$</td>
</tr>
<tr>
<td>1</td>
<td>1,059</td>
<td>2,624</td>
<td>$O(8N)$</td>
</tr>
</tbody>
</table>

Table 1: Number of tested combinations for the experiments of Fig. 5. LP-MERT with $S = 8$ checks only 600K full combinations on average, much less than the total number of combinations (which is more than $10^{20}$).
Figure 6: Effect of the number of features (runtime on 1 CPU of a modern computer). Each curve represents a different number of tuning sentences.
Cosine Similarity Approximation

\[ \cos(\hat{w}, w_0) \geq 0.84 \]

Figure 7: Effect of a constraint on \( w \) (runtime on 1 CPU).
Comparison with 1D-MERT

Assuming LP-MERT finds the "global optimum", ‘for S=4, [Powell] makes search errors in 90% of the cases, despite using 20 random starting points’

As S increases, the gap between 1D-MERT and LP-MERT increases

With beam (size 1000)

<table>
<thead>
<tr>
<th></th>
<th>32</th>
<th>64</th>
<th>128</th>
<th>256</th>
<th>512</th>
<th>1024</th>
</tr>
</thead>
<tbody>
<tr>
<td>1D-MERT</td>
<td>22.93</td>
<td>20.70</td>
<td>18.57</td>
<td>16.07</td>
<td>15.00</td>
<td>15.44</td>
</tr>
<tr>
<td>our work</td>
<td>25.25</td>
<td>22.28</td>
<td>19.86</td>
<td>17.05</td>
<td>15.56</td>
<td>15.67</td>
</tr>
<tr>
<td></td>
<td>+2.32</td>
<td>+1.59</td>
<td>+1.29</td>
<td>+0.98</td>
<td>+0.56</td>
<td>+0.23</td>
</tr>
</tbody>
</table>

Table 2: BLEUn4r1[%] scores for English-German on WMT09 for tuning sets ranging from 32 to 1024 sentences.

As S increases, the gap between 1D-MERT and LP-MERT decreases
Summary

• End-to-end evaluation (with beam approx. for LP-MERT)
  • Tuning: 0.24 BLEU difference
  • Test: 0.17 BLEU difference

• Exact multi-dimensional MERT
  • LP at the core
  • Divide-and-conquer, lazy enumeration

• Polynomial in dimension, N-best list size
• Exponential in number of sentences
• Approximations used to limit running time
• Bold approach to tackle difficult problem
Questions & concerns that I had…

• End-to-end results do not look significant
• Additional language pairs/datasets would be nice
• As S increases, does the over-performance diminish?
• What can we do to make this algorithm poly(S)?
• LP-MERT + hypergraph MERT → towards MERT 2.0?
• Is direct cost optimization the way forward?

Thank you!