Machine Translation - Decoding

Christian Kranzler, Hannes Pomberger

January 15, 2007
Table of Contents

1. Introduction
2. IBM Model 4 (Recap)
3. Definition of the search problem
4. Stack Decoder
5. Greedy Decoder
6. Integer Programing Decoder
7. Experimental Results

Christian Kranzler, Hannes Pomberger

Machine Translation - Decoding
IBM Model 4 (Recap)

- Word alignments
- Fertility Table
- Translation Table
- Heads
- Non-heads
- NULL-generated
Figure: A sample word alignment [1]
Definition of the search problem

- Sum of the probabilities of aligning $f$ with $e$
  \[ P(f \mid e) = \sum_a P(a, f \mid e) \]
- Calculating this sum is very expensive, therefore:
- Maximize term $P(a, f \mid e) \times P(e)$
Definition of the search problem:

\[ < \hat{e}, \hat{a} > = \arg \max_{e, a} P(a, f | e) \ast P(e) \]
The basic algorithm

- Best-First search algorithm
- Very similar to the A* algorithm
- Hypotheses stored in a priority queue
The basic algorithm (ct.)

1. Initialize the stack with an empty hypothesis
2. Pop $h$, the most promising hypothesis, off the stack
3. If $h$ is complete, output $h$ and terminate
4. Extend $h$ in each possible manner of incorporating the next input word and insert the resulting hypotheses into the stack
5. Return to step (2)
Stack decoding: machine translation versus speech recognition

- Speech recognition follows input order
- → decoder can take words in any order, that increases complexity up to $n!$ permutations of an $n$-word input
- Heuristic function estimates cost of completing partial hypotheses
Stack decoding - operations

- Add
- AddZFert
- Extend
- AddNull
Stack decoding - pros and cons

- Advantages
  - Explores larger search space than greedy decoder
  - Runs faster than the optimal (Integer Programming) decoder
  - Can employ trigrams (optimal decoder only bigrams)

- Disadvantages
  - Time and space complexity exponential to input words
  - In practise only useful for sentences with max. 20 words
The greedy decoder - the idea

- Time for optimal decoding increases exponentially with input length
- With greedy methods time reduces to polynomial function
  - Start with a random, approximate solution
  - Try to improve it incrementally until satisfactory solution is reached
The greedy decoder - operations

- translateOneOrTwoWords\((j, e'_a(j), k, e'_a(k))\)
- translateAndInsert\((j, e'_a(j), e'_x)\)
- removeWordOfFertility0\((i)\)
- swapSegments\((i_1, i_2, j_1, j_2)\)
- joinWords\((i, j)\)
Decoding example

[initial gloss]:

```
NULL well heard , it talking a beautiful victory .
```

```
bien entendu , il parle de une belle victoire .
```

**Figure:** Initial gloss [1]
translateTwoWords(5, talks, 7, great):

NULL well heard , it talks a great victory .

bien entendu , il parle de une belle victoire .

Figure: Step 1 [1]
translateTwoWords(2, understood, 0, about):

NULL well understood, it talks about a great victory.

bien entendu, il parle de une belle victoire.

Figure: Step 2 [1]
translateOneWords(4, he):

NULL well understood, he talks about a great victory.

bien entendu, il parle de une belle victoire.

Figure: Step 3 [1]
translateTwoWords(1, quite, 2, naturally):

null quite naturally, he talks about a great victory.

bien entendu, il parle de une belle victoire.

Figure: Step 4 [1]
Integer programming decoding

Idea

- good word order for a decoder output similar to a good TSP tour
- possible to transform a decoding problem into a TSP instance?
- take advantage of previous research into TSP algorithms
Integer programming decoding

- convert decoding into straight TSP is difficult

- more general framework for optimization problems: linear integer programming
Integer programming decoding

Sample integer program:

\[
\begin{align*}
\text{minimize objective function:} \\
& 3.2 \times x_1 + 4.7 \times x_2 - 2.1 \times x_3 \\
\text{subject to constraints:} \\
& x_1 - 2.6 \times x_3 > 5 \\
& 7.3 \times x_2 > 7
\end{align*}
\]

Optimal solution:
- respects the constraints
- minimizes the value of the objective function
How to express MT decoding in IP format:

1. create a salesman graph
2. establish real-valued distances
3. cast tour selection as an integer program
4. invoke a IP solver
Create a sales man graph:

- a city for each word in the sentence $f$
- ten hotels per city corresponding to ten likely English word translations
- if $n$ cities have hotels owned by the same owner $x$, build $2^n - n - 1$ new hotels on various city boarders
- add an extra city representing the sentence boundary
Fig. 3. A salesman graph for the input sentence f = “CE NE EST PAS CLAIR.” A few hotels are omitted for readability, e.g., the NULL hotel at the intersection of EST, PAS, and PERIOD.
Define a tour of cities:

- sequence of hotels so that each city is visited once
- hotels on the border of two cities count as visiting both cities
- each tour corresponds to a potential decoding \( \langle e, a \rangle \)
  (owners of the hotel give e, hotel locations yield a)
Establish real-valued (asymmetric) distances between pairs of hotels:

- so that length of any tour is \(- \log(P(e) \cdot P(a, f|e))\)
- the distance of each pair of hotels is a piece of the Model 4 formula
Example: inter-hotel distance “not” following “what”
Example: inter-hotel distance "not" following "what"

\[
distance = - \log(bi(not|what)) - \log(n(2|not)) \\
- \log(t(NE|not)) - \log(t(PAS|not)) \\
- \log(d_1(+1|\text{class}(what),\text{class}(NE))) \\
- \log(d_{>1}(+2|\text{class}(PASS)))
\]
Special treatment of NULL-owned hotels:

- all non-NUL hotels be visited before any NULL hotels
- at most one NULL hotel be visited on a tour
- zero distance from NULL hotel to the sentence boundary hotel
- infinite distance to any other
Example: inter-hotel distance "NULL" following "cannot"
Example: inter-hotel distance "NULL" following "cannot"

\[
distance = - \log \left( \frac{6 - 2}{2} \right) - 2 \cdot \log(p_1) - (6 - 4) \log(1 - p_1) \\
- \log(t(CE|NULL)) - \log(t(ES|NULL)) \\
- \log(bi(sentence - boundary|cannot))
\]
between hotels located in the same city assign infinite distance

for a 6-word French sentence - graph with 80 hotels and 3500 finite-cost travel segments

Zero-fertility words:
- disallow adjacent zero-fertility words
- compare bigram and $n(0|e)$ probabilities
Integer programming decoding

Cast tour selection as an integer program

- sub-tour elimination strategy like used in standard TSP
- create a binary integer variable $x_{ij}$
- $x_{ij} = 1$ iff travel from hotel $i$ to hotel $j$
Integer programming decoding

- **objective function:**
  
  \[
  \text{minimize: } \sum x_{ij} \cdot \text{distance}(i, j)
  \]

- **constraints:**
  
  - exactly one tour segment must exist in each city
  - the segments must be linked to one another
  - prevent multiple independent sub-tours
The shortest tour corresponds to optimal decoding

- invoke a IP solver
  (generic problem solving software such as lp_solve or CPLEX)
- extract $\langle e, a \rangle$ from the list of variables and their binary values
Further opportunities:

- create a list of n-best solutions
- new constraint to the IP - don’t choose the same solution again
- simply repeating the procedure
Advantages of the IP approach in general

- a decoder can be built rapidly, with very little programming
- optimal $n$-best results can be obtained
- generic problem solvers offer a wide range of user-customizable search strategies, thresholds, etc.
Disadvantages

- other knowledge sources (e.g. wider English context) may not be easily integrated
- slow performance
Experiments and discussion

Experiment specification:

- top ten word translations
- 128 words of fertility 0
- bigram language model (Table 1)
- trigram language model (Table 2)
- 505 sentences, uniformly distributed across the lengths 6, 8, 10, 15 and 20
Experiments and discussion

Error classes:

<table>
<thead>
<tr>
<th>Error classification</th>
<th>$e' = \hat{e}$</th>
<th>$\hat{e}$ is perfect</th>
<th>$e'$ is perfect</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. no error (NE)</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>2. pure model error (PME)</td>
<td>yes</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>3. deadly search error (DSE)</td>
<td>no</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>4. fortuitous search error (FSE)</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>5. harmless search error (HSE)</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>6. compound error (CE)</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
</tbody>
</table>

$\hat{e}$ ... optimal decoding
$e'$ ... best decoding found by a decoder
Table 1
Comparison of decoders on sets of 101 test sentences. All experiments in this table use a bigram language model. Translation errors can be syntactic, semantic, or both. Errors were counted on the sentence level, so that every sentence can have at most one error in each category.

<table>
<thead>
<tr>
<th>len</th>
<th>decoder</th>
<th>time</th>
<th>SE</th>
<th>TE</th>
<th>NE</th>
<th>PME</th>
<th>DSE</th>
<th>FSE</th>
<th>HSE</th>
<th>CE</th>
<th>BLEU</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>IP</td>
<td>47.50</td>
<td>0</td>
<td>57</td>
<td>44</td>
<td>57</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.206</td>
</tr>
<tr>
<td>6</td>
<td>stack</td>
<td>0.79</td>
<td>5</td>
<td>58</td>
<td>43</td>
<td>53</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>0.209</td>
</tr>
<tr>
<td>6</td>
<td>greedy</td>
<td>0.07</td>
<td>18</td>
<td>60</td>
<td>38</td>
<td>45</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td>10</td>
<td>0.197</td>
</tr>
<tr>
<td>8</td>
<td>IP</td>
<td>499.00</td>
<td>0</td>
<td>76</td>
<td>27</td>
<td>74</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.157</td>
</tr>
<tr>
<td>8</td>
<td>stack</td>
<td>5.67</td>
<td>20</td>
<td>75</td>
<td>24</td>
<td>57</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>15</td>
<td>0.162</td>
</tr>
<tr>
<td>8</td>
<td>greedy</td>
<td>2.66</td>
<td>43</td>
<td>75</td>
<td>20</td>
<td>38</td>
<td>4</td>
<td>5</td>
<td>1</td>
<td>33</td>
<td>0.147</td>
</tr>
</tbody>
</table>

len: input sentence length; time: average translation time (in sec./sent.); SE: search errors; TE: translation errors; NE: no error; PME: pure model errors; DSE: deadly search errors; FSE: fortuitous search errors; HSE: harmless search errors; CE: compound error; BLEU: score according to the IBM BLEU metric.
Experiments and discussion

Table 2
Comparison between decoders using a trigram language model. Greedy* and greedy\(_1\) are greedy decoders optimized for speed.

<table>
<thead>
<tr>
<th>Length</th>
<th>Decoder</th>
<th>Av. time (in sec./sent.)</th>
<th>Erroneous translations</th>
<th>BLEU</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>stack</td>
<td>13.72</td>
<td>42</td>
<td>0.282</td>
</tr>
<tr>
<td>6</td>
<td>greedy</td>
<td>1.58</td>
<td>46</td>
<td>0.226</td>
</tr>
<tr>
<td>6</td>
<td>greedy*</td>
<td>0.07</td>
<td>46</td>
<td>0.202</td>
</tr>
<tr>
<td>8</td>
<td>stack</td>
<td>45.45</td>
<td>59</td>
<td>0.231</td>
</tr>
<tr>
<td>8</td>
<td>greedy</td>
<td>2.75</td>
<td>68</td>
<td>0.188</td>
</tr>
<tr>
<td>8</td>
<td>greedy*</td>
<td>0.15</td>
<td>69</td>
<td>0.174</td>
</tr>
<tr>
<td>10</td>
<td>stack</td>
<td>105.15</td>
<td>57</td>
<td>0.271</td>
</tr>
<tr>
<td>10</td>
<td>greedy</td>
<td>3.83</td>
<td>63</td>
<td>0.247</td>
</tr>
<tr>
<td>10</td>
<td>greedy*</td>
<td>0.20</td>
<td>68</td>
<td>0.225</td>
</tr>
<tr>
<td>15</td>
<td>stack</td>
<td>&gt;2000</td>
<td>74</td>
<td>0.225</td>
</tr>
<tr>
<td>15</td>
<td>greedy</td>
<td>12.06</td>
<td>75</td>
<td>0.202</td>
</tr>
<tr>
<td>15</td>
<td>greedy*</td>
<td>1.11</td>
<td>75</td>
<td>0.190</td>
</tr>
<tr>
<td>15</td>
<td>greedy(_1)</td>
<td>0.63</td>
<td>76</td>
<td>0.189</td>
</tr>
<tr>
<td>20</td>
<td>greedy</td>
<td>49.23</td>
<td>86</td>
<td>0.219</td>
</tr>
<tr>
<td>20</td>
<td>greedy*</td>
<td>11.34</td>
<td>93</td>
<td>0.217</td>
</tr>
<tr>
<td>20</td>
<td>greedy(_1)</td>
<td>0.94</td>
<td>93</td>
<td>0.209</td>
</tr>
</tbody>
</table>
Conclusions:

- search errors differ significantly between decoders
- measure of translation quality do not
- majority of the translation errors comes from the LM and TM
- for improvement in translation quality better models are needed
Conclusions (ct.):

- BLEU score reflects the rank order but is a "ballpark figure" estimate of the decoder performance depending on the application
  - slow decoder that provides optimal results
  - or fast, greedy decoder with non-optimal but acceptable results
*Fast and optimal decoding for machine translation*  

[2] Yamada, Knight  
*A decoder for syntax-based statistical MT*  

*The mathematics of statistical machine translation: Parameter estimation*  

[4] Knight  
*Decoding complexity in word-replacement translation models*  
Thank you for your attention!

- Feel free to ask questions!