Roadmap for today

- What is decoding
- Quick complexity theory and search review
- Word-based decoding
- Phrase-based decoding
- MERT: a few more details
Decoding

- Fancy name of searching for the best translation
  - Find (most likely) assignment for some latent variable
  - Decoding is a reference to codebreaking

- We have a probabilistic model of translation $p_\theta(e|f)$
  - $e$ is an English sentence
  - $f$ is a foreign sentence
  - $\theta$ is some parameter vector

- Given $f$ and $\theta$, find the best translation
  \[ e_{best} = \arg \max_e p_\theta(e|f) \]
  Contrast with parameter estimation (given $e, f$)
  \[ \theta_{best} = \arg \max_\theta p_\theta(e|f) \]
Decoding errors (while we’re here)

- Say we try to search for $e_{\text{best}} = \arg \max_e p(e|f)$
  - We hope to find $e_{\text{ref}}$, a human reference translation
  - Instead we find $e_{\text{junk}}$, something incomprehensible
  - What could have gone wrong?
Decoding errors (while we’re here)

- Say we try to search for \( e_{best} = \arg \max_e p(e|f) \)
  - We hope to find \( e_{ref} \), a human reference translation
  - Instead we find \( e_{junk} \), something incomprehensible
  - What could have gone wrong?
- Answer 1: Model error
  \[ p(e_{junk}|f) > p(e_{ref}|f) \]
- Answer 2: Search error
  - We didn’t actually find the max.
  - Today’s lecture focuses on good ways of finding the max
MT decoding is **tough**

- Stronger claim: it’s NP-complete
- Review: definition of NP-complete
MT decoding is **tough**

- Stronger claim: it’s NP-complete

- Review: definition of NP-complete
  - \( P = \) problems solvable in polynomial time using a deterministic machine
  - \( NP = \) problems solvable in polynomial time using a non-deterministic machine
  - \( P=NP? \) Not sure. Ongoing attempts at proof.
NP-Complete

- Over the years, computer scientists found a set of tricky interlinked problems
  - If you can solve one in polynomial time, can solve them all
  - Two criteria for a problem X to be NP complete
    - Polynomial time verification of purported solutions to X.
    - Reduction from NP complete problem Y: Show that every instance of Y can be cast as X, so polynomial X solver == polynomial Y solver.

- Somewhat proof by intimidation: your problem is so difficult that if you solved it well, you’d solve a bunch of other difficult problems too. Good luck.

- Thus MT decoding might not require exponential time... but it probably does.
Part-of-speech tagging complexity

- Start with part-of-speech tagging
  - $w_1 ... w_m$ are the input words
  - We’re looking for the best tags $t_1 ... t_m$
- More details:
  - We have $v$ total tags.
  - Bigram tag model $p(t_i | t_{i-1})$ (how many params?)
  - Channel model $p(w_i | t_i)$
- We assume:

$$p(w_1 ... w_m, t_1 ... t_m) = \prod_{i=1}^{m} p(t_i | t_{i-1})p(w_i | t_i)$$

- **NN**
  - time

- **VBZ**
  - flies

- **IN**
  - like

- **DT**
  - an

- **NN**
  - arrow
Finding the best POS tags

- What is the complexity for finding the best POS tagging of a sentence?
- How does this relate to MT?
  - (tags can be foreign words)
- What phenomena present in MT does this fail to model?
  - Ordering
  - Insertion / deletion
  - Agreement, and many more
NP-complete reductions

Decoding with Model 1 and language model $p(e) \cdot p(f|e)$

$$p(e) = b(e_1|\langle s\rangle) \cdot \prod_{i=2}^{l} b(e_i|e_{i-1}) \cdot b(\langle s\rangle|e_l)$$

$$p(f|e) = \epsilon(m|l) \frac{1}{lm} \prod_{j=1}^{m} \sum_{i=0}^{l} s(f_j|e_i)$$

There parameters of the model are:
- $b(e|e')$, a bigram language model
- $\epsilon(m|l)$, a distribution over source lengths given target length
- $s(f|e)$, a word-based channel model
Reduction 1: Hamilton Circuit Problem

- HCP: Given a directed graph G with vertices 0, ..., n, does G have a path that visits each vertex exactly once and returns to its starting point?

- Reduction:
  - Create a French vocab $f_1, ..., f_n$
  - Create an English vocab $e_0, ..., e_n; e_0 = \langle s \rangle$ (boundary)
  - Parameters:
    \[
    s(f_i | e_j) = \delta_i(j) \\
    \epsilon(m | l) = \delta_l(m) \\
    b(e_j | e_i) = \begin{cases} 
      1 / n & \text{if G contains an edge from vertex } i \text{ to } j \\
      0 & \text{otherwise.}
    \end{cases}
    \]

- Now, if we can decode and find a non-zero value, we’ve found a HCP solution.
Reduction 2: Minimum Set Cover

- Covered in Kevin Knight’s squib
What makes MT so difficult?
- Reordering + unique coverage
- Different than “monotone” problems like POS tagging and speech recognition

Ramifications
- Exact search is not likely feasible, at least with unconstrained reordering (more on this later)
- Need to resort to lossy search approaches
- Also employ lossless pruning!
A review from introductory AI classes
Map of Romania with step costs in km

Slide copied from Hwee Tou Ng's AI course slides
Search problem formulation

A problem is defined by four items:

1. initial state e.g., "at Arad"
2. actions or successor function $S(x) =$ set of action–state pairs
   - e.g., $S(\text{Arad}) = \{<\text{Arad} \rightarrow \text{Zerind}, \text{Zerind}>, \ldots \}$
3. goal test
   - e.g., $x =$ "at Bucharest"
4. path cost (additive)
   - e.g., sum of distances, number of actions executed, etc.
   - $c(x,a,y)$ is the step cost, assumed to be $\geq 0$ normally in AI search but not guaranteed in MT

- A solution is a sequence of actions leading from the initial state to a goal state
- We need to find lowest cost solution
  - For MT, can define cost as the negative score (max score = min cost)

Slide copied from Hwee Tou Ng's AI course slides
How to search: tree search algorithms

- Basic idea:
  - offline, simulated exploration of state space by generating successors of already-explored states (a.k.a. expanding states)
  - building up a tree of explored states

```
function TREE-SEARCH(problem, strategy) returns a solution, or failure
initialize the search tree using the initial state of problem
loop do
    if there are no candidates for expansion then return failure
    choose a leaf node for expansion according to strategy
    if the node contains a goal state then return the corresponding solution
    else expand the node and add the resulting nodes to the search tree
```

Slide copied from Hwee Tou Ng's AI course slides
Best-first search

- Idea: use an evaluation function $f(n)$ for each node
  - estimate of "desirability"
  - Expand most desirable unexpanded node

- Implementation:
  Order the nodes in fringe in decreasing order of desirability

- Special cases:
  - greedy best-first search
  - A* search

Slide copied from Hwee Tou Ng's AI course slides
Romania with step costs in km

Slide copied from Hwee Tou Ng's AI course slides
**A* best-first search**

- Evaluation function $f(n) = g(n) + h(n)$ (heuristic)
- $g(n)$ = cost of path to node
- $h(n)$ = estimate of cost from $n$ to closest goal
  - e.g., $h_{SLD}(n) =$ straight-line distance from $n$ to Bucharest
- A* uses an admissible heuristic – one that does not overestimate the cost of the best path to a goal state
  - This property makes it optimal
A* search example
A* search example
A* search example
A* search example
A* search example
A* search example
Word-based decoding
Nobody uses word based models anymore – other approaches work better

However, there are some nice ideas here

Mine for broad concepts, don’t get tied up in the details
Model 4

\[ P(a, f | e) = \]

\[
\prod_{i=1}^{l} n(\phi_i) \times 
\left( m - \phi_0 \right) \binom{m}{k} p_1^{\phi_0} (1 - p_1)^{m-2\phi_0} \times 
\prod_{i=0}^{l} \prod_{k=1}^{\phi_i} t(\tau_{ik} | e_i) \times 
\prod_{i=1, \phi_i > 0}^{l} \prod_{i=0}^{\phi_i} \prod_{k=1}^{l} d_1(\pi_{i1} - c_{\rho_i} | K(e_{\rho_i}), K(\tau_{i1})) \times 
\prod_{i=1}^{l} \prod_{k=1}^{\phi_i} d_{>1}(\pi_{ik} - \pi_{i(k-1)} | K(\tau_{ik})) \times 
\]

Fertility

NULL fertility

Translation

Distortion of first word in cept

Distortion of first word in cept
Decoding strategies

- A* (aka Stack)
- Greedy
- Exact (e.g., Integer Linear Programming)
Stack

- Basic idea:
  - `s = new stack();`
  - `s.push(new hyp());`
  - `while (!s.isEmpty())`
    - `x = s.pop();`
    - `if (x.isComplete()) return x;`
    - `foreach (var y in extensions(x))`
      - `s.push(y);`

- Keep the stack ordered by \( g(x) + h(x) \)
  - `g(x) = exact cost of hyp`  
  - `h(x) = heuristic completion cost`
Multiple stack decoding

- Say that every hyp $x$ has an integral class $k(x)$
  - Empty hyp has class 0
  - If $y$ is an extension of $x$, then $k(y) > k(x)$
  - Easy interpretation: # of input words covered

- For $a = 0$ to (#input words – 1)
  - Consider all hyps $h$ in stack $a$
    - Consider all extensions – generally we translate another word, so we place in some subsequent stack.
Greedy

- Iterated local search
  - TranslationHyp \( h \leftarrow \text{InitialHyp}(f) \)
  - bool ChangedSomething \leftarrow false
  - Do
    - Foreach (operation \( o \) in OPS)
      - \( h' \leftarrow o(h) \)
    - If score(h') > score(h)
      - \( h \leftarrow h' \), ChangedSomething \leftarrow true
  - While ChangedSomething
  - Return h
Greedy, for Model 4

- **Initializer:**
  - For each French word, pick its most likely translation
  - Could be NULL
  - String together to form an English sentence with alignment

- **Operations:**
  - `translateOneOrTwoWords(j₁, e₁, j₂, e₂)`
  - `translateAndInsert(j, e₁, e₂)`
  - `removeWordOfFertility0(i)`
  - `swapSegments(i₁, i₂, j₁, j₂)`
  - `joinWords(i₁, i₂)`
Greedy search example

NULL well heard, it talking a beautiful victory.

bien entenda, il parle de une belle victoire.

NULL well heard, it talks a great victory.

bien entenda, il parle de une belle victoire.

NULL well understood, it talks about a great victory.

bien entenda, il parle de une belle victoire.

NULL well understood, he talks about a great victory.

bien entenda, il parle de une belle victoire.

NULL quite naturally, he talks about a great victory.

bien entenda, il parle de une belle victoire.

Action

translateTwoWords(5,taiks,7,great)

translateTwoWords(2,understood,3,about)

translateOneWord(4,he)

translateTwoWords(1,quite,2,naturally)
Integer linear programming

- General setup:
  - Given a function
    \[ f(x) = c^T x \]
  - and a set of constraints
    \[ Ax \leq b \]
  - where \( x \) are the free variables, \( c \) and \( b \) are vectors of known coefficients, and \( A \) is a matrix of known coefficients,
  - find \( x \) to maximize \( f(x) \) subject to the constraints.

- NP hard, but there are solvers that work pretty well in practice – too slow for real use, but good for offline experiments.
MT as ILP

- Our objective function is a negative logprob
- Constraints can be expressed such that we cover each input word exactly once
- Details are in the papers, but suffice to say that ILP serves as an exact search mechanism
Salesman graph

- **Terminology**
  - Set up a *city* for each word in observed sentence.
  - Each city is populated with 10 *hotels*, one for each likely English translation.
  - The *owner* of the hotel is the English word inside the rectangle.
    - If two cities have hotels with the same owner, place another copy on the border between the two cities.
  - Add an extra city for the sentence boundary.

- A *tour of cities* is a sequence of hotels that visits each city exactly once before returning to the start.
Salesman graph
## Decoder comparison

<table>
<thead>
<tr>
<th>sent length</th>
<th>decoder type</th>
<th>time (sec/sent)</th>
<th>search errors</th>
<th>translation errors (semantic and/or syntactic)</th>
<th>NE</th>
<th>PME</th>
<th>DSE</th>
<th>FSE</th>
<th>HSE</th>
<th>CE</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>
</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>
</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>
</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>
</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>
</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>
</tr>
</tbody>
</table>

Table 1: Comparison of decoders on sets of 101 test sentences. All experiments in this table use a bigram language model.
## Decoder comparison (cont’d)

<table>
<thead>
<tr>
<th>sent length</th>
<th>decoder type</th>
<th>time (sec/sent)</th>
<th>translation errors (semantic and/or syntactic)</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>stack</td>
<td>13.72</td>
<td>42</td>
</tr>
<tr>
<td>6</td>
<td>greedy</td>
<td>1.58</td>
<td>46</td>
</tr>
<tr>
<td>6</td>
<td>greedy*</td>
<td>0.07</td>
<td>46</td>
</tr>
<tr>
<td>8</td>
<td>stack</td>
<td>45.45</td>
<td>59</td>
</tr>
<tr>
<td>8</td>
<td>greedy</td>
<td>2.75</td>
<td>68</td>
</tr>
<tr>
<td>8</td>
<td>greedy*</td>
<td>0.15</td>
<td>69</td>
</tr>
<tr>
<td>10</td>
<td>stack</td>
<td>105.15</td>
<td>57</td>
</tr>
<tr>
<td>10</td>
<td>greedy</td>
<td>3.83</td>
<td>63</td>
</tr>
<tr>
<td>10</td>
<td>greedy*</td>
<td>0.20</td>
<td>68</td>
</tr>
<tr>
<td>15</td>
<td>stack</td>
<td>&gt;2000</td>
<td>74</td>
</tr>
<tr>
<td>15</td>
<td>greedy</td>
<td>12.06</td>
<td>75</td>
</tr>
<tr>
<td>15</td>
<td>greedy*</td>
<td>1.11</td>
<td>75</td>
</tr>
<tr>
<td>15</td>
<td>greedy^1</td>
<td>0.63</td>
<td>76</td>
</tr>
<tr>
<td>20</td>
<td>greedy</td>
<td>49.23</td>
<td>86</td>
</tr>
<tr>
<td>20</td>
<td>greedy*</td>
<td>11.34</td>
<td>93</td>
</tr>
<tr>
<td>20</td>
<td>greedy^1</td>
<td>0.94</td>
<td>93</td>
</tr>
</tbody>
</table>

Table 2: Comparison between decoders using a trigram language model. Greedy* and greedy^1 are greedy decoders optimized for speed.
After this, we’re back to mainstream things that work well
Search in phrase-based translation
Basic phrase-translation

- Decisions for target sentence, segmentation and alignment, given source sentence

Source sentence is segmented into source phrases
- Not linguistically motivated segmentation
- Each source phrase is translated into a target phrase
  - Independent of other source phrases and their translations
- The resulting target phrases are re-ordered to form output
Translation as sequence of actions: decoding process

- Build translation from left to right
  - Select foreign sequence of words to be translated
Decoding process

- Build translation from left to right
  - Select foreign sequence of words to be translated
  - Select English phrasal translation from phrase table
  - Append English words to the end of the partial translation
Decoding process

- Build translation from left to right
  - Select foreign sequence of words to be translated
  - Select English phrasal translation from phrase table
  - Append English words to the end of the partial translation
  - Mark foreign words as translated
    - So we know not to select them again later on
Decoding process

- One-to-many translation
Decoding process

Many-to-one translation
Decoding process

- Many-to-one translation
Decoding process

- Reordering
Decoding process

- Reordering
- Translation finished (reached a goal)
Many different phrase-translation options available for a sentence

Can look them all up before starting decoding
Decoding organization

- Each sequence of actions we explore defines a partial translation hypothesis.
- Translation hypotheses are the analogue of search nodes in general search.
- The data we keep in each translation hypothesis should be sufficient to:
  - Tell us which actions are applicable:
    - We need to know which foreign words have been translated.
  - Tell us what is the cost so far (in the examples we will use probability instead and will multiply probs instead of adding up costs).
  - Allow us compute the cost of each possible next action.
  - Allow us read off the target translation.
Search hypotheses and initial hypothesis

- Start with an initial **empty hypothesis**
  - **e**: no English words have been output
  - **f**: no foreign words have been translated
  - the probability so far is 1 (we will multiply in the prob. of each next action)
  - end prev 0 not shown here but need the end of the previous phrase in f for distortion model computation
  - prev 2 English words: not shown but need them for language model computation

<s> <s>
Hypothesis expansion

- Create next hypothesis using this action
  - e: Mary is in the output
  - f: Maria has been translated
  - p: the probability of the partial translation so far
Computing the probability of actions

- Probability of actions depends on models used
  - Translation models
    - Phrasal probabilities in both directions
    - Lexical weighting probabilities
    - Word count, phrase count
  - Reordering model probability
    - Can be computed given current phrase pair and positions in the source of the current and previous phrase
  - Language model probability
    - Can be computed given English side of current phrase pair, and last 2 previous English words (for trigram LM)
Hypothesis expansion

- Add another hypothesis using another translation option
Hypothesis expansion

- Further expansion
Hypothesis expansion

- Trace the parent links back to the beginning to collect full translation
Hypothesis expansion

- The search space explodes
  - grows exponentially with sentence length
Explosion of search space

- The search graph grows exponentially with sentence length
- Due to the number of possible re-orderings, problem is NP complete [Knight, 1999]
- We need to reduce the search space
  - Can recombine equivalent hypotheses (loss-less, risk-free pruning)
  - Apply other kinds of pruning
    - Histogram pruning
    - Threshold pruning
Hypothesis recombination

- Example: different paths to the same English output in partial hypotheses
  - Correspond to different phrasal segmentation
Hypothesis recombination

- Combine equivalent hypotheses
  - Drop the weaker hypothesis
  - The weaker path is still available for lattice generation
Hypothesis recombination

- We just need them to have the same best path to completion
  - The same applicable future expansions with the same scores
  - Same last 2 English words, coverage vectors, last phrase source position
- Since any path that goes through the worse hypothesis can be changed to use the path to the better hypothesis and then the same path to the end, we are not losing anything.
Pruning

- Hypothesis recombination is helpful but space is still too large
- Need some way to discard poor hypotheses early
- Need to compare hypotheses
  - Will organize them into stacks for easier comparison by e.g.:
    - Same number of foreign words covered
    - Same number of English words produced
    - Same set of foreign words covered
- Compare hypotheses in stacks, discard weak ones
  - Histogram pruning; keep a fixed number of top hypotheses in each stack
  - Threshold pruning: prune hypotheses with probability less than $\alpha p_{\text{best}}$ where $p_{\text{best}}$ is the prob of the best hypothesis in the stack
- The $i$-th stack contains hypotheses for which $i$ source words have been translated
- Process stacks in order
  - Expand all hypotheses from a stack
  - Place expanded hypotheses on corresponding stacks
How to compare hypotheses

- So far we only have the probability (cost) of each hypothesis
- Comparing hypotheses that have translated the same number of words makes these costs more comparable
- Can we do better than comparing based on cost so far?
Comparing hypotheses

- Comparing two hypotheses translating the same number of words
  - The one translating an easier part of the sentence is preferred
  - Can do better by considering future cost of translating the rest of the source words
Estimating future cost

- The closest to the correct future cost we get, the better for our search
- But computation of the future cost should not take too long
- A future cost estimate that is less or equal to the true cost (optimistic), guarantees optimality in A* search
- This has usually been too slow in practice, so we don’t use A* and admissible heuristics
Estimating future cost

- The future cost will be the sum of costs of actions (translations) that we will take in the future.
- We can estimate the cost of each translation option for the sentence:
  - Translation probabilities: context independent
  - Lang model: context dependent, so we approximate
    - $P(\text{to})P(\text{the}|\text{to})$
  - Reordering model cost: ignore, can’t estimate without context
  - Prob for option = $\text{LM} \times \text{TM}$
Future cost estimation

- Find the cost of the cheapest translation for a given source phrase (highest probability)
Future cost estimation

- For each span of the source sentence (each contiguous sequence of words) compute the cost of the cheapest combination of translation options
  - Can be done efficiently using dynamic programming
Estimation of combined score of hypotheses

- Add up the costs of contiguous spans in the un-translated sequence of words to compute future cost
- Add future cost to cost so far to compute combined score used for pruning
Limits on reordering

- Limits on reordering can reduce the search space dramatically
- Monotone decoding
  - Target phrases follow same order as source phrases
- Reordering limit \( n \) (used in Moses)
  - Forbid jumps greater with distance greater than \( n \)
  - Results in polynomial inference
- In addition to speed-ups reordering limits often lead to improved translations
  - Because the reordering models are weak
Word lattices

- Can easily extract a *word lattice* from search graph
- Can extract $n$-best translation hypotheses
  - $n$-best lists are used for discriminative re-ranking and training of log-linear model parameters
Parameter estimation with N-best lists
N-best list optimization

- Say we have \( n \) possible translations of a sentence
- Can rank them by some sort of oracle score
- Each one also has a feature vector
- We search for the best according to a dot product of free parameters with that feature vector
- How to pick that dot product?
MERT

- Downside to pseudo reference: make the best translation have top rank... even if second best is almost as good
- Another idea: try to optimize our objective function directly
  - Goal: $\hat{\lambda} = \arg \max \lambda \text{BLEU}(\text{decode}(f, \lambda), e^*)$
  - In practice, this is too slow; can’t decode on every parameter assignment
  - Instead, use n-best lists $L$, and assume that rerank($L, f, \lambda$) is similar to decode($s, \lambda$)
    - $\text{decode}(f, \lambda) = \arg \max_{e} P(f|e) \cdot \lambda_1 + P(e) \cdot \lambda_2$
    - $\text{rerank}(L, f, \lambda) = \arg \max_{e \in L} P(f|e) \cdot \lambda_1 + P(e) \cdot \lambda_2$
- If n-best list is good, reasonable approximation
Can extract n-best paths through a lattice using a cute trick:

- During first pass search, we’re going left-to-right
- Each hyp (node in the lattice) has an exact score for the words to the left, and an estimate to the right
- For n-best, go backwards: start from each end-state, and do a search toward the beginning
  - Note: completion cost in this backward search is optimistic so we can run A* and get n-best in order
  - Even better: it’s exact! Search is very efficient: creates minimum number of hyps necessary for n-best
Och’s MERT innovations

- Optimize weights over fixed hypothesis sets, re-decoding only a few times, rather than re-decoding for every point examined in feature-weight space.

- Perform globally optimal line searches by taking advantage of the following fact:
  
  Given a finite set of discrete hypotheses, the top-scoring hypothesis can change only at fixed points in feature-weight space.
Fig. 13. Illustration of the MERT line minimization function for optimizing a single parameter $\lambda_k$. In (1) we compute $P_{\lambda_k}(\hat{e}|f)$ as a line in $\lambda_k$ for each candidate translation $\hat{e}$ of a single source sentence $f$. We then find the intervals at which the optimal candidate $\hat{e}$ changes by computing the intersection points of these lines. Once the best candidates and intervals are known, we can compute $E_{\lambda_k}(\arg\max_{\hat{e}} P(\hat{e}|f), e)$ as a function of $\lambda_k$. This function is shown in (2). We repeat this procedure for a new source sentence in (3) and (4). Finally, we add the single-sentence error functions (2) and (4) to compute the aggregate error function for both input sentences. This function is shown in (5). Applying this process iteratively to all sentence pairs in the training corpus, we can compute the full error function $\sum_{(e,f) \in C} E_{\lambda_k}(\arg\max_{\hat{e}} P(\hat{e}|f), e)$.

To optimize, we simply walk along all intervals of this function until we determine the minimum.
Och’s basic MERT algorithm (without random restarts)

- Set current point in feature-weight (F-W) space to some initial value
- Let hyp set be N-best translations for dev set at current point
- Repeat until hyp set does not change
  - Repeat until objective for hyp set does not change for a complete cycle through a set of lines spanning F-W space
    - Choose a line in F-W space passing through current point
    - Reset current point to globally optimize objective along line
  - Generate N-best translations for dev set at current point
  - Merge N-best translations into hyp set
Why doesn’t optimal line search yield multi-dimensional optimization?
Optimality WRT each dimension ≠ optimality WRT all dimensions
Random restarts

- Even though line search is globally optimal, there is no guarantee that iterated line searches will result in finding the global optimum in all dimensions.
- Therefore, repeat inner optimization loop several times from randomly selected initial points.