
Tuning

Philipp Koehn

26 February 2015



The Story so Far: Generative Models



- The definition of translation probability follows a mathematical derivation

$$\operatorname{argmax}_{\mathbf{e}} p(\mathbf{e}|\mathbf{f}) = \operatorname{argmax}_{\mathbf{e}} p(\mathbf{f}|\mathbf{e}) p(\mathbf{e})$$

- Occasionally, some independence assumptions are thrown in for instance IBM Model 1: word translations are independent of each other

$$p(\mathbf{e}|\mathbf{f}, a) = \frac{1}{Z} \prod_i p(e_i|f_{a(i)})$$

- Generative story leads to straight-forward estimation
 - maximum likelihood estimation of component probability distribution
 - EM algorithm for discovering hidden variables (alignment)

Log-linear Models



- IBM Models provided mathematical justification for multiplying components

$$p_{LM} \times p_{TM} \times p_D$$

- These may be weighted

$$p_{LM}^{\lambda_{LM}} \times p_{TM}^{\lambda_{TM}} \times p_D^{\lambda_D}$$

- Many components p_i with weights λ_i

$$\prod_i p_i^{\lambda_i}$$

- We typically operate in log space

$$\sum_i \lambda_i \log(p_i) = \log \prod_i p_i^{\lambda_i}$$

Knowledge Sources



- Many different knowledge sources useful
 - language model
 - reordering (distortion) model
 - phrase translation model
 - word translation model
 - word count
 - phrase count
 - character count
 - drop word feature
 - phrase pair frequency
 - additional language models
- Could be any function $h(\mathbf{e}, \mathbf{f}, \mathbf{a})$

$$h(\mathbf{e}, \mathbf{f}, \mathbf{a}) = \begin{cases} 1 & \text{if } \exists e_i \in \mathbf{e}, e_i \text{ is verb} \\ 0 & \text{otherwise} \end{cases}$$

Set Feature Weights



- Contribution of components p_i determined by weight λ_i
- Methods
 - manual setting of weights: try a few, take best
 - automate this process
- Learn weights
 - set aside a development corpus
 - set the weights, so that optimal translation performance on this development corpus is achieved
 - requires automatic scoring method (e.g., BLEU)

Discriminative vs. Generative Models



- Generative models
 - translation process is broken down to steps
 - each step is modeled by a probability distribution
 - each probability distribution is estimated from data by maximum likelihood
- Discriminative models
 - model consist of a number of features (e.g. the language model score)
 - each feature has a weight, measuring its value for judging a translation as correct
 - feature weights are optimized on development data, so that the system output matches correct translations as close as possible

- Generate a set of possible translations of a sentence (candidate translations)
- Each candidate translation represented using a set of features
- Each feature derives from one property of the translation
 - feature score: value of the property
(e.g., language model probability)
 - feature weight: importance of the feature
(e.g., language model feature more important than word count feature)
- Task of discriminative training: find good feature weights
- Highest scoring candidate is best translation according to model

Discriminative Training Approaches



- Reranking: 2 pass approach
 - first pass: run decoder to generate set of candidate translations
 - second pass:
 - * add features
 - * rescore translations
- Tuning
 - integrate all features into the decoder
 - learn feature weights that lead decoder to best translation
- Large scale discriminative training (next lecture)
 - thousands or millions of features
 - optimization of the entire training corpus
 - requires different training methods

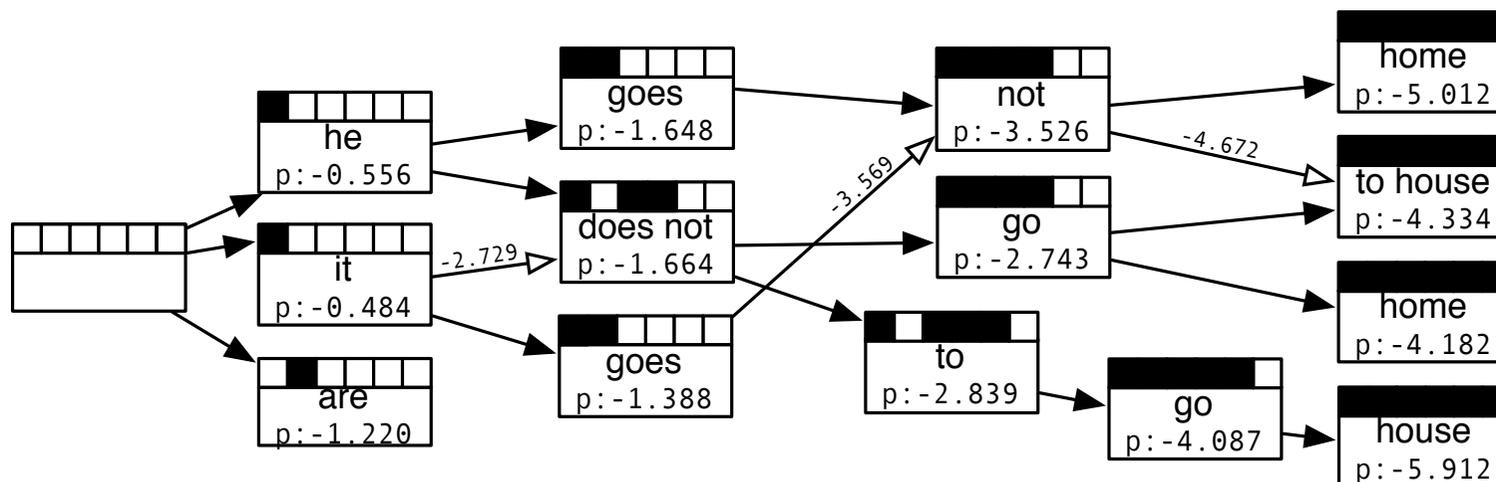
finding candidate translations

Finding Candidate Translations



- Number of possible translations exponential with sentence length
- But: we are mainly interested in the most likely ones
- Recall: decoding
 - do not list all possible translation
 - beam search for best one
 - dynamic programming and pruning
- How can we find **set** of best translations?

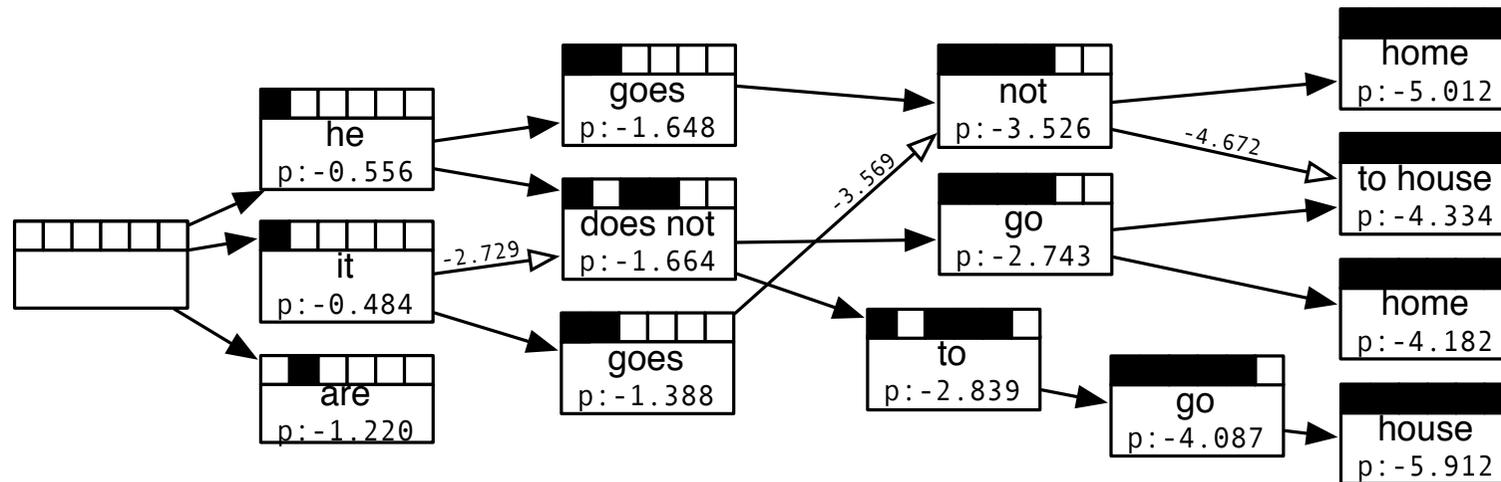
Search Graph



- Decoding explores space of possible translations by expanding the most promising partial translations

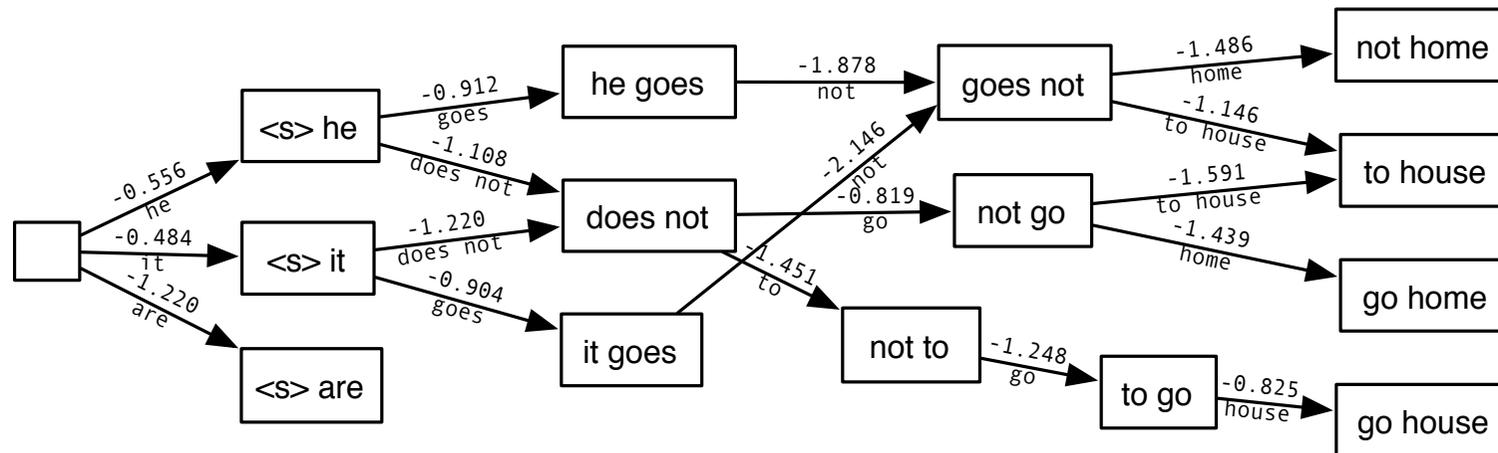
⇒ Search graph

Search Graph



- Keep transitions from recombinations
 - without: total number of paths = number of full translation hypotheses
 - with: combinatorial expansion
- Example
 - without: 4 full translation hypotheses
 - with: 10 different full paths
- Typically many more paths due to recombination

Word Lattice



- Search graph as finite state machine
 - states: partial translations
 - transitions: applications of phrase translations
 - weights: added scores by phrase translation

- Formally, a finite state machine, is a q quintuple $(\Sigma, S, s_0, \delta, F)$, where
 - Σ is the alphabet of output symbols (in our case, the emitted phrases)
 - S is a finite set of states
 - s_0 is an initial state ($s_0 \in S$), (in our case the initial hypothesis)
 - δ is the state transition function $\delta : S \times \Sigma \rightarrow S$
 - F is the set of final states (in our case representing hypotheses that have covered all input words).
- Weighted finite state machine
 - scores for emissions from each transition $\pi : S \times \Sigma \times S \rightarrow \mathbf{R}$

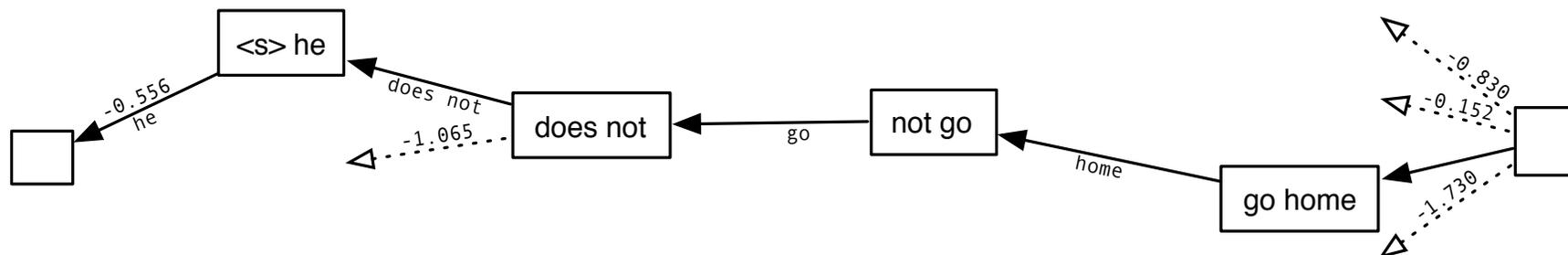
N-Best List

rank	score	sentence
1	-4.182	he does not go home
2	-4.334	he does not go to house
3	-4.672	he goes not to house
4	-4.715	it goes not to house
5	-5.012	he goes not home
6	-5.055	it goes not home
7	-5.247	it does not go home
8	-5.399	it does not go to house
9	-5.912	he does not to go house
10	-6.977	it does not to go house

- Word graph may be too complex for some methods

⇒ Extract n best translations

Path 1



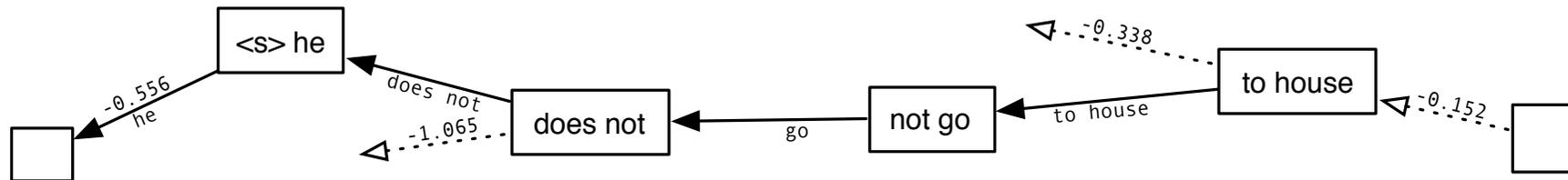
- Follow back transitions

⇒ Best path: **he does not go home**

- Keep note of detours from this path

Base path	Base cost	Detour cost	Detour state
final	-0	-0.152	to house
final	-0	-0.830	not home
final	-0	-1.065	does not
final	-0	-1.730	go house

Path 2

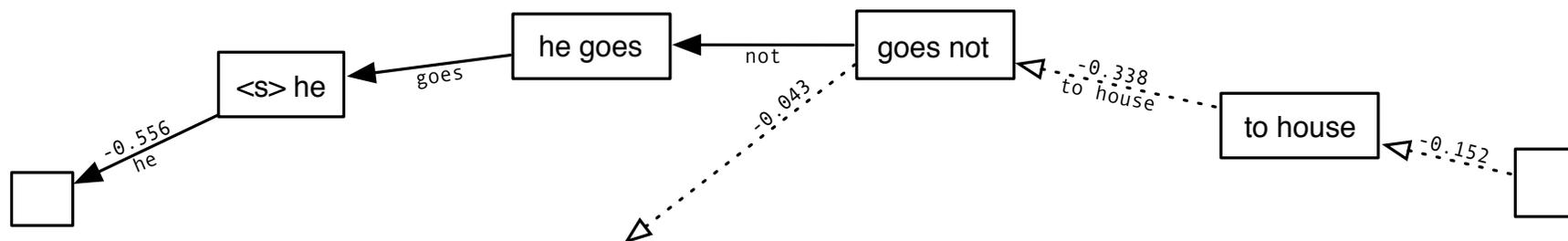


- Take cheapest detour
- Afterwards, follow back transitions
- Second best path: **he does not go to house**
- Add its detours to priority queue

Base path	Base cost	Detour cost	Detour state
to house	-0.152	-0.338	goes not
final	-0	-0.830	not home
final	-0	-1.065	does not
to house	-0.152	-1.065	it
final	-0	-1.730	go house

Path 3

18



- Third best path: **he goes not to house**
- Add its detours to priority queue

Base path	Base cost	Detour cost	Detour state
to house / goes not	-0.490	-0.043	it goes
final	-0	-0.830	not home
final	-0	-1.065	does not
to house	-0.152	-1.065	it
final	-0	-1.730	go house

Scoring N-Best List

- Two opinions about items in the n-best list
 - model score: what the machine translation system thinks is good
 - error score: what is actually a good translation
- Error score can be computed with reference translation
 - recall: lecture on evaluation
 - canonical metric: BLEU score
- Some methods require sentence-level scores
 - commonly used: BLEU+1
 - adjusted precision: $\frac{\text{correct matches}+1}{\text{total}+1}$

Scored N-Best List

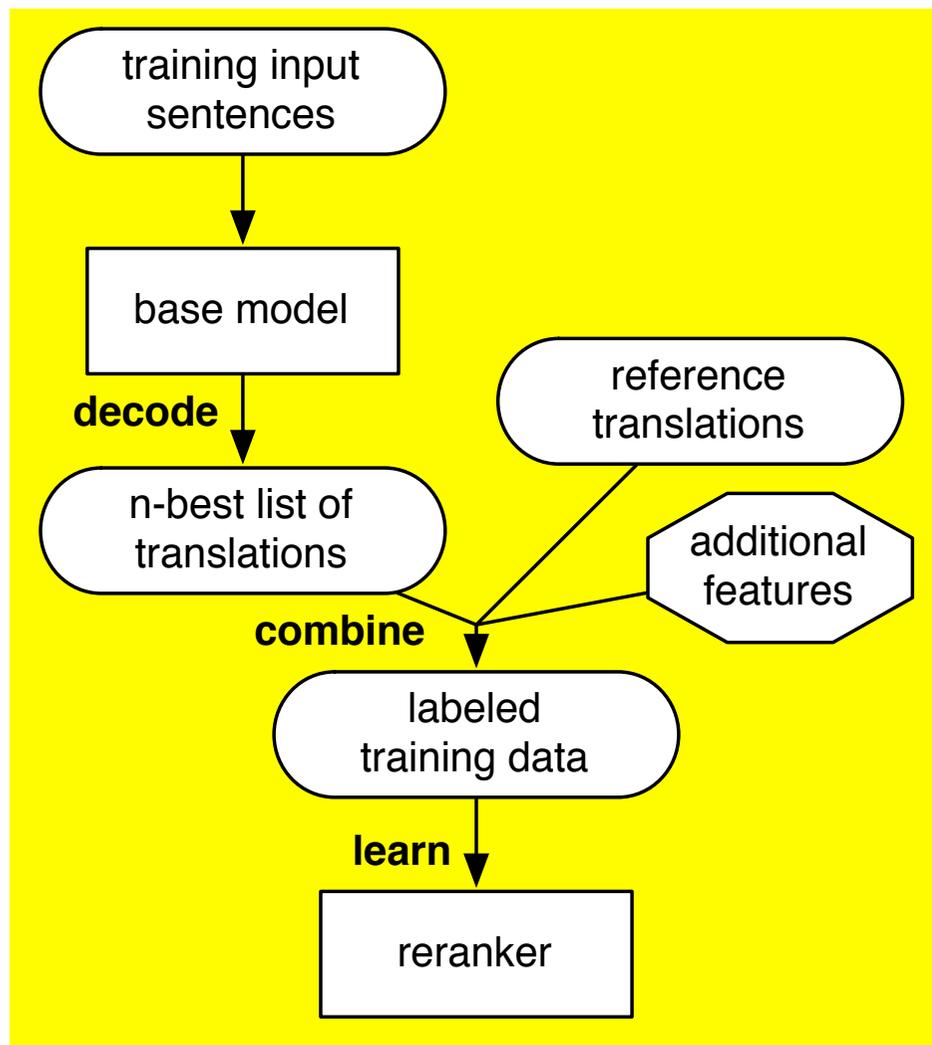
- Reference translation: **he does not go home**
- N-best list

Translation	Feature values						BLEU+1
it is not under house	-32.22	-9.93	-19.00	-5.08	-8.22	-5	27.3%
he is not under house	-34.50	-7.40	-16.33	-5.01	-8.15	-5	30.2%
it is not a home	-28.49	-12.74	-19.29	-3.74	-8.42	-5	30.2%
it is not to go home	-32.53	-10.34	-20.87	-4.38	-13.11	-6	31.2%
it is not for house	-31.75	-17.25	-20.43	-4.90	-6.90	-5	27.3%
he is not to go home	-35.79	-10.95	-18.20	-4.85	-13.04	-6	31.2%
he does not home	-32.64	-11.84	-16.98	-3.67	-8.76	-4	36.2%
it is not packing	-32.26	-10.63	-17.65	-5.08	-9.89	-4	21.8%
he is not packing	-34.55	-8.10	-14.98	-5.01	-9.82	-4	24.2%
he is not for home	-36.70	-13.52	-17.09	-6.22	-7.82	-5	32.5%

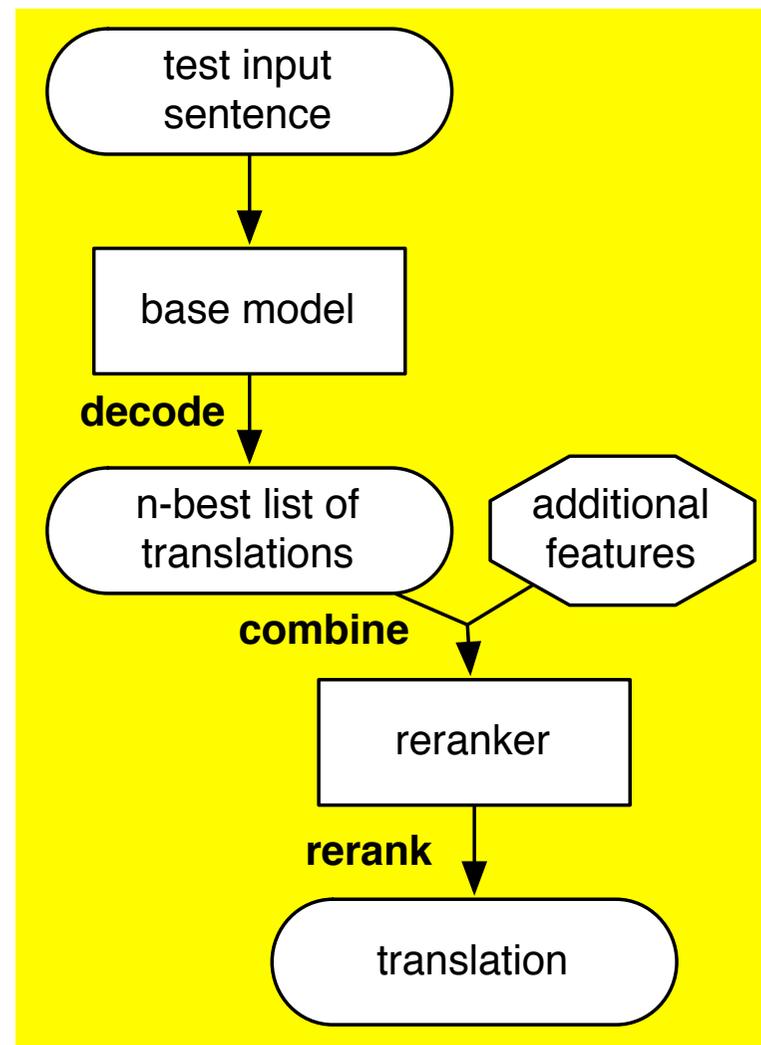
- What feature weights push up the correct translation?

Rerank Approach

Training



Testing



parameter tuning

Parameter Tuning

- Recall log-linear model

$$p(x) = \exp \sum_{i=1}^n \lambda_i h_i(x)$$

- Overall translation score $p(x)$ is combination of components $h_i(x)$, weighted by parameters λ_i
- Setting parameters as supervised learning problem
- Two methods
 - Powell search
 - Simplex algorithm

Experimental Setup

- Training data for translation model: 10s to 100s of millions of words
- Training data for language model: billions of words
- Parameter tuning
 - set a few weights (say, 10–15)
 - tuning set of 1000s of sentence pairs sufficient
- Finally, test set needed

Minimum Error Rate Training

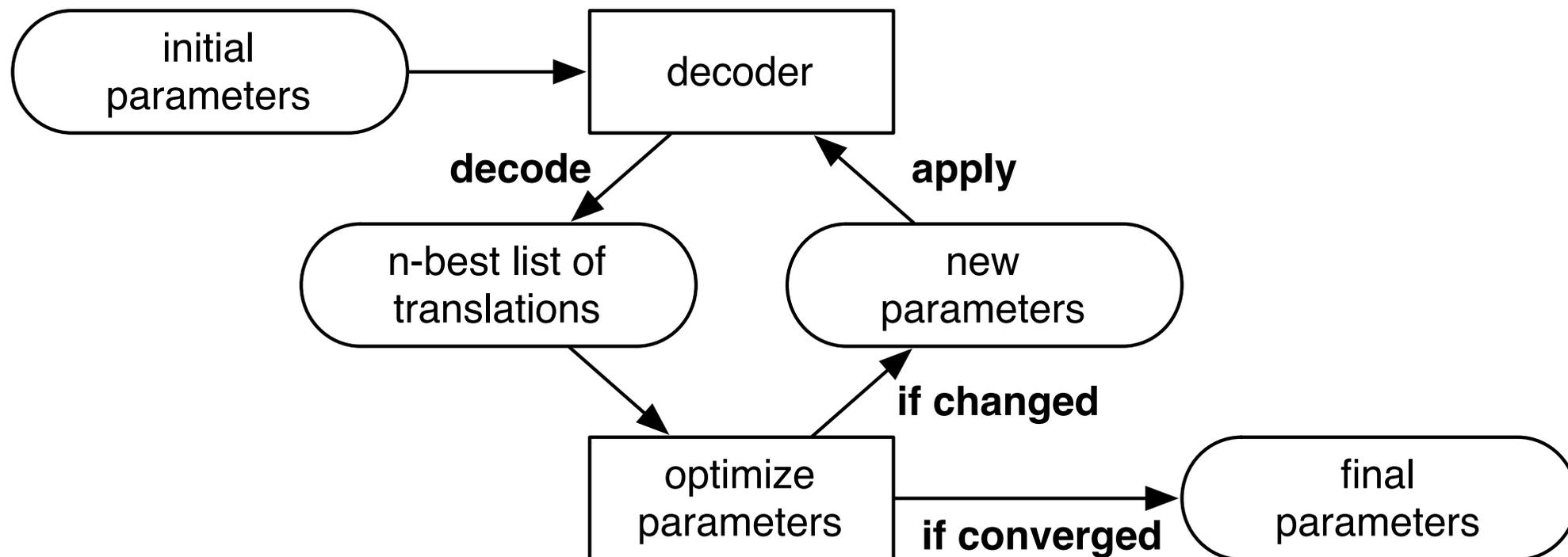


- Optimize metric: e.g., BLEU
- Tuning set of 1000s of sentences,
for each we have n-best list of translations
- Different weight setting
 - different translations come out on top
 - BLEU score
- Even with 10-15 features: high dimensional space, intractable

Bad N-Best Lists?

- N-Best list produced with initial weight setting
 - Decoding with optimized weight settings
→ may produce completely different translations
- ⇒ Iterate optimization, accumulate n-best lists

Parameter Tuning



powell search

Och's minimum error rate training (MERT)



- Line search for best feature weights

```
given:  sentences with n-best list of
translations
iterate n times
    randomize starting feature weights
    iterate until convergences
        for each feature
            find best feature weight
            update if different from
current
return best feature weights found in any
iteration
```

Find Best Feature Weight

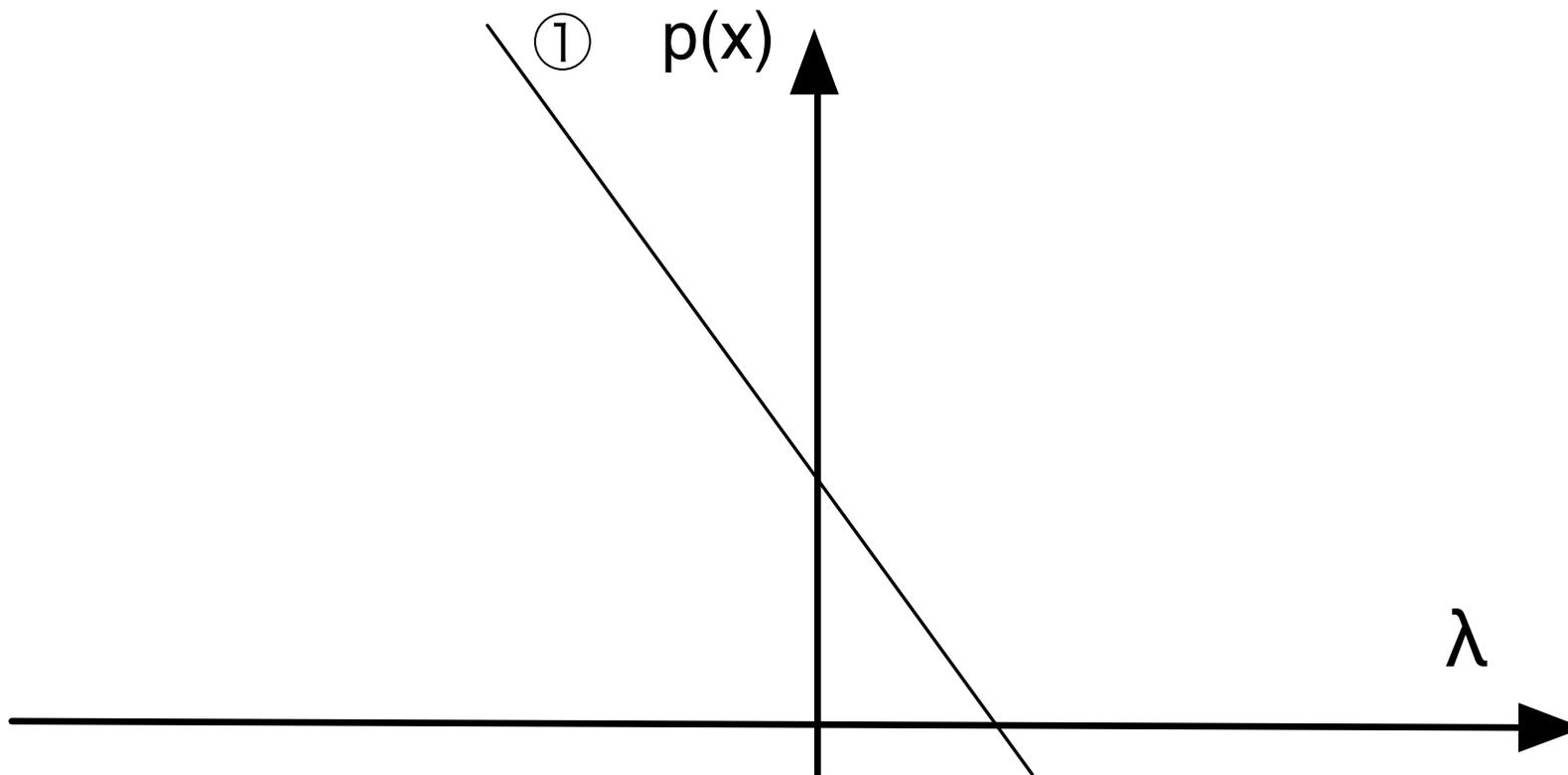
- Core task:
 - find optimal value for one parameter weight λ
 - ... while leaving all other weights constant

- Score of translation i for a sentence \mathbf{f} :

$$p(\mathbf{e}_i|\mathbf{f}) = \lambda a_i + b_i$$

- Recall that:
 - we deal with 100s of translations \mathbf{e}_i per sentence \mathbf{f}
 - we deal with 100s or 1000s of sentences \mathbf{f}
 - we are trying to find the value λ so that over all sentences, the error score is optimized

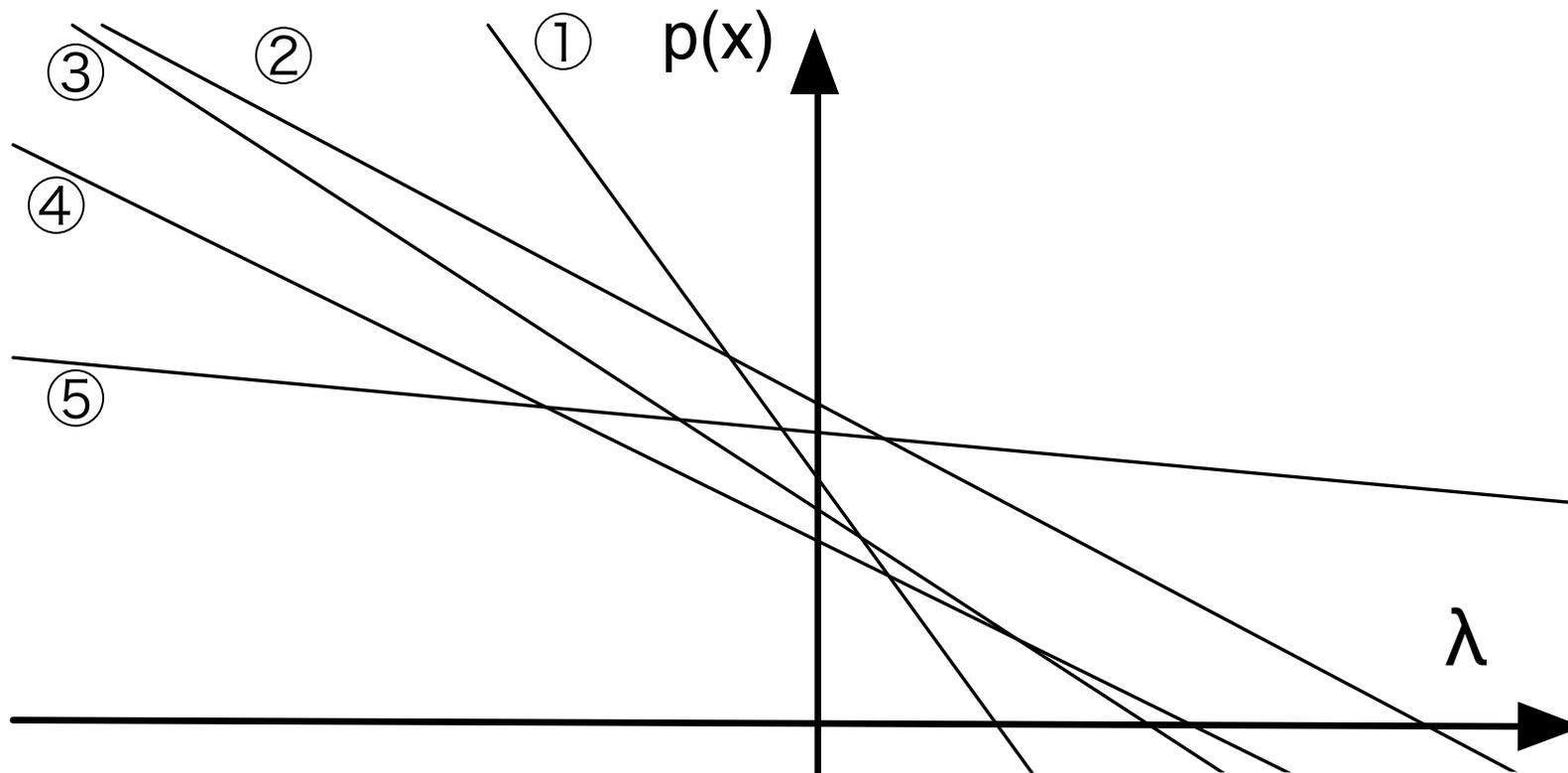
One Translations for One Sentence



- Probability of one translation $p(\mathbf{e}_i|\mathbf{f})$ is a function of λ

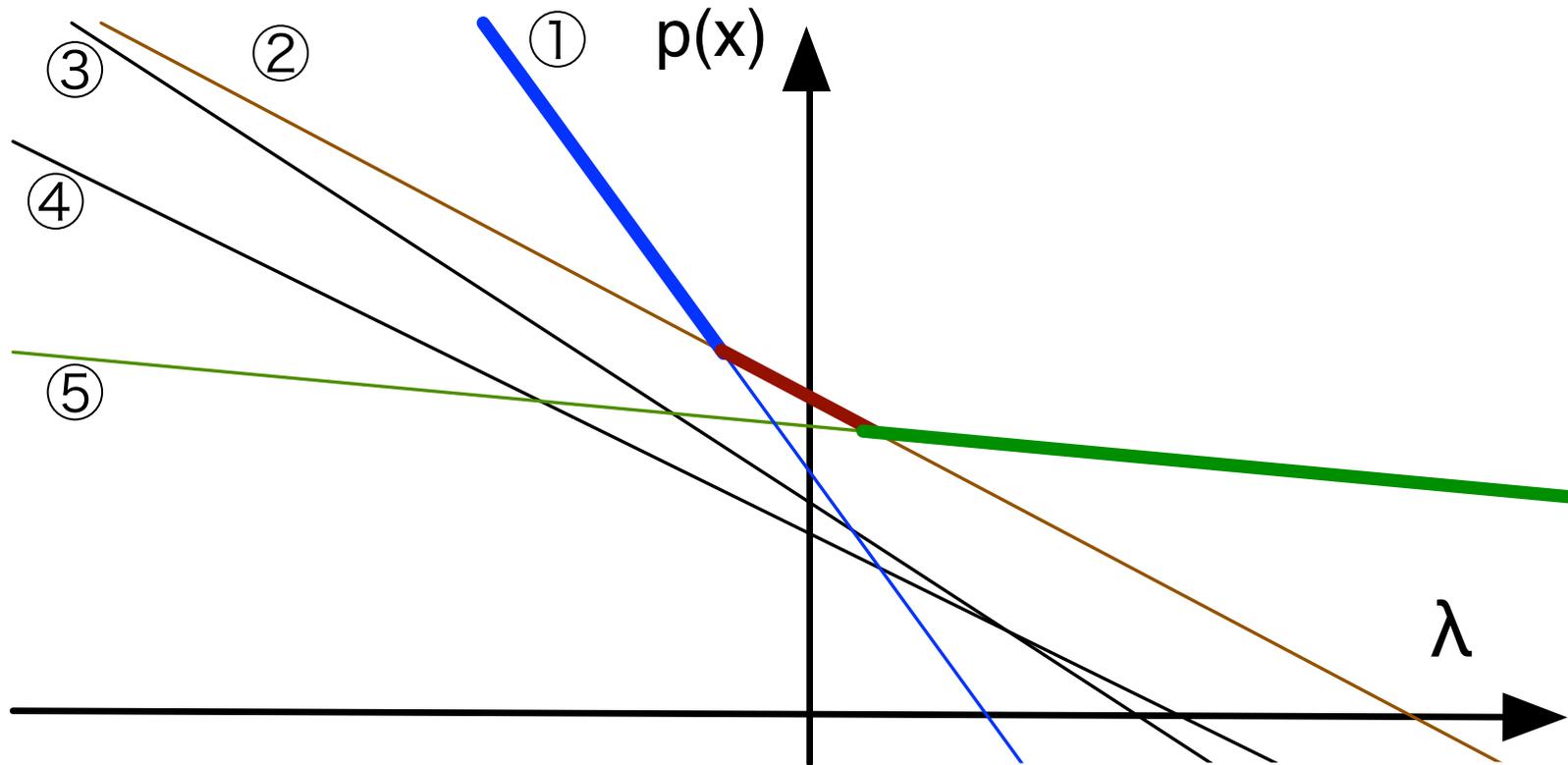
$$p(\mathbf{e}_i|\mathbf{f}) = \lambda a_i + b_i$$

N-Best Translations for One Sentence



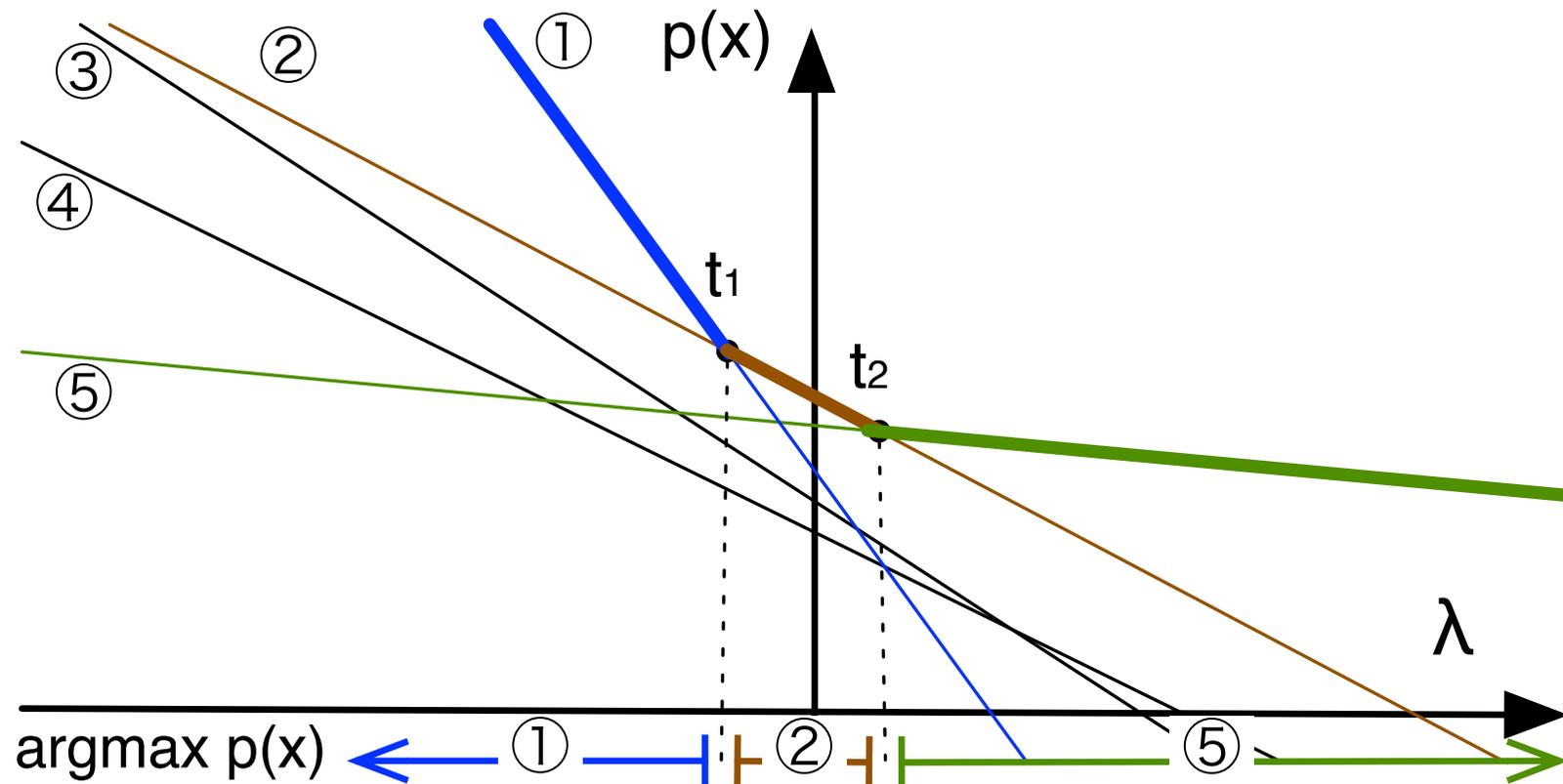
- Each translation is a different line

Upper Envelope



- Highest probability translation depends on λ

Threshold Points



- There are one a few threshold points t_j where the model-best line changes

Finding the Optimal Value for λ

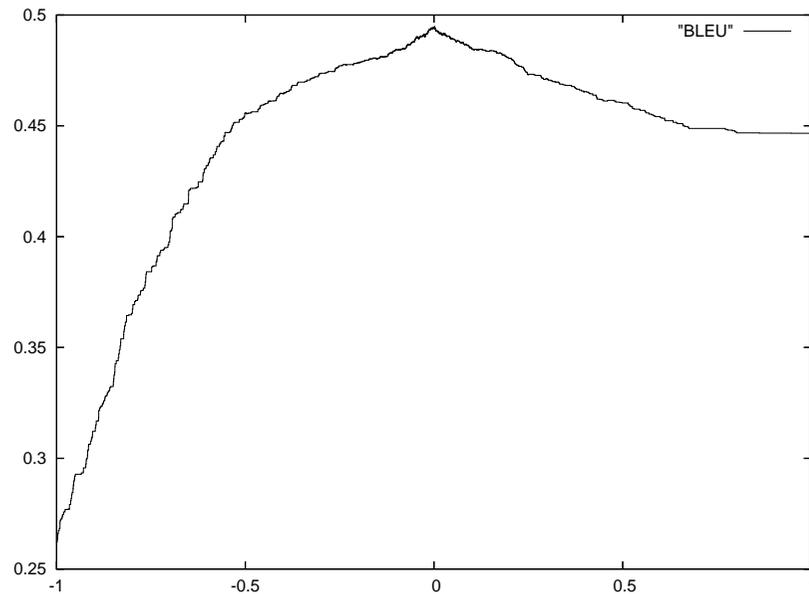
- Real-valued λ can have infinite number of values
- But only on threshold points, one of the model-best translation changes

⇒ Algorithm:

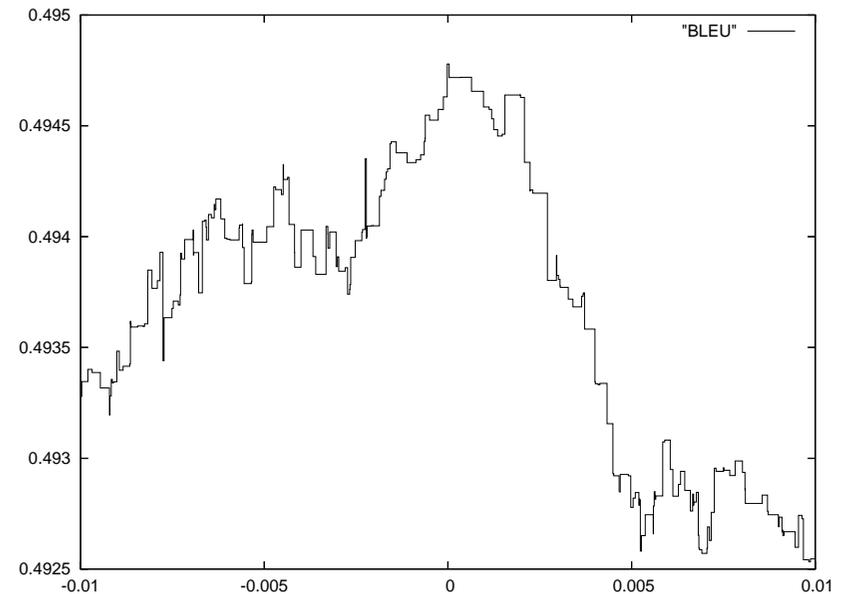
- find the threshold points
- for each interval between threshold points
 - * find best translations
 - * compute error-score
- pick interval with best error-score

BLEU Error Surface

- Varying one parameter: a rugged line with many local optima



full range



peak

Pseudo Code

Input: sentences with n-best list of translations, initial parameter values

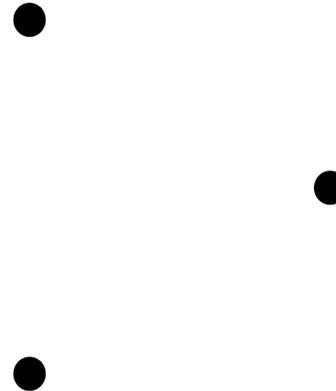
```
1: repeat
2:   for all parameter do
3:     set of threshold points  $T = \{\}$ 
4:     for all sentence do
5:       for all translation do
6:         compute line  $l$ : parameter value  $\rightarrow$  score
7:       end for
8:       find line  $l$  with steepest descent
9:       while find line  $l_2$  that intersects with  $l$  first do
10:        add parameter value at intersection to set of threshold points  $T$ 
11:         $l = l_2$ 
12:      end while
13:    end for
14:    sort set of threshold points  $T$  by parameter value
15:    compute score for value before first threshold point
16:    for all threshold point  $t \in T$  do
17:      compute score for value after threshold point  $t$ 
18:      if highest do record max score and threshold point  $t$ 
19:    end for
20:    if max score is higher than current do update parameter value
21:  end for
22: until no changes to parameter values applied
```

simplex algorithm

Simplex Algorithm

- Similar to Powell search
- Less calculations of the current error
 - recall: error is computed over the entire tuning set
 - brute force method requires reranking of 1000s of n-best lists
- Similar to gradient descent methods
 - try to find direction in which the optimum lies
 - here: we cannot compute derivative

Simplex Algorithm



- Randomly generate three points in the high dimensional space
 - high dimensional space = each dimension is one of the λ_i parameters
 - a point in the space = each parameter set to a value

Simplex Algorithm

best

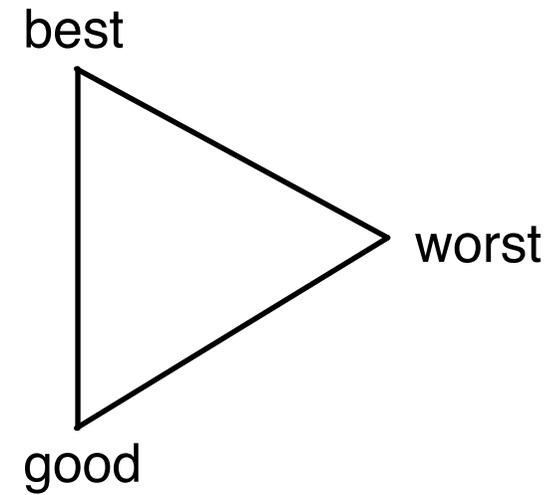


● worst

●
good

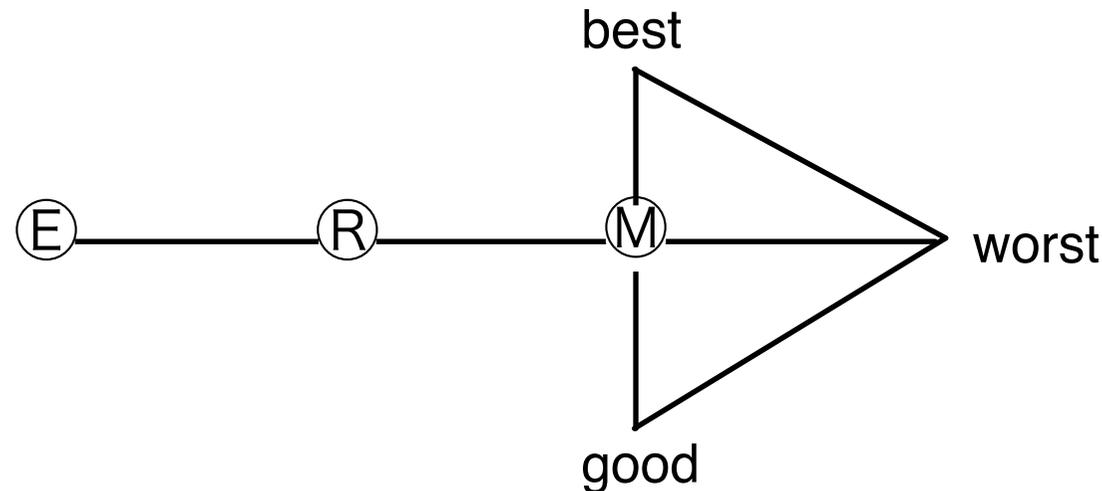
- We can score each of these points
 - use parameter settings to rerank all the n-best lists
 - compute overall tuning set score (BLEU)
- Rank the 3 points into best, good, worst

Simplex Algorithm



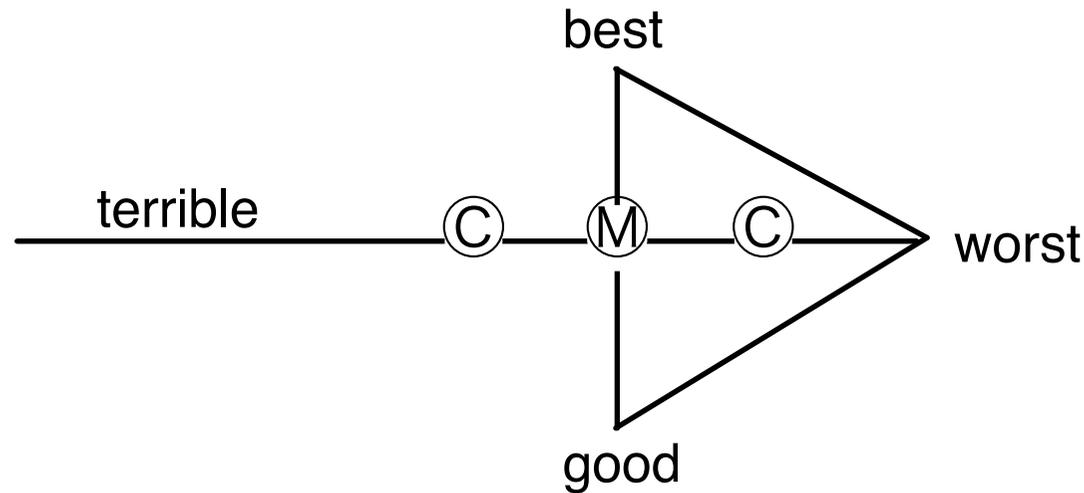
- The 3 points form a triangle

First Idea: Move Away from the Bad Point



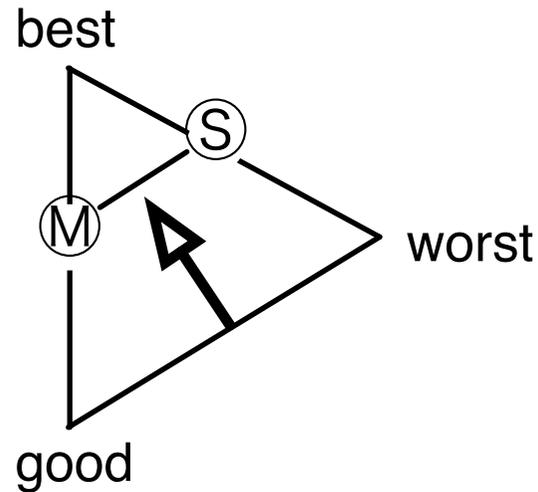
- Compute 3 additional points
 - mid point: $M = \frac{1}{2}(\text{best} + \text{good})$
 - reflection point: $R = M + (M - \text{worst})$
 - extension: $R = M + 2(M - \text{worst})$
- Three cases
 1. if $\text{error}(E) < \text{error}(R) < \text{error}(\text{worst})$, replace *worst* with *E*.
 2. else if $\text{error}(R) < \text{error}(\text{worst})$, replace *worst* with *R*.
 3. else try something else

Second Idea: Well, Not Too Far Away



- Compute 2 additional points
 - C_1 point between *worst* and M : $C_1 = M + \frac{1}{2}(M - \text{worst})$
 - C_2 point between M and R : $C_2 = M + \frac{3}{2}(M - \text{worst})$.
- Three cases
 1. if $\text{error}(C_1) < \text{error}(\text{worst})$ and $\text{error}(C_1) < \text{error}(C_2)$, replace *worst* with C_1 .
 2. if $\text{error}(C_2) < \text{error}(\text{worst})$ and $\text{error}(C_2) < \text{error}(C_1)$, replace *worst* with C_2 .
 3. else continue

Third Idea: Move Closer to Best Point



- Compute 1 additional point
 - S point between *worst* and *best*: $S = \frac{1}{2}(\textit{best} + \textit{worst})$.
- Shrink triangle

Simplex in High Dimensions

- Process of updates is iterated until the points converge
- Typically very quick
- More dimensions: more points
 - $n + 1$ points for n parameters
 - midpoint M is the center of all points except *worst*
 - in final case, all *good* points moved towards midpoints closer to *best*
- Once optimum is found
 - generate n -best list
 - iterate

Summary



- Reframing probabilistic model as log-linear model with weights
- Discriminative training task: set weights
- Generate n-best candidate translations from search graph
- Reranking
- Powell search (Och's MERT)
- Simplex algorithm