Machine Translation
Phrase-based Models 3 (Tuning)

Matthias Huck
(based on slides by Philipp Koehn and Barry Haddow)

24 February 2014

School of informatics

Log-linear Model

- We've expressed translation using a probabilistic model:

$$\mathbf{e}_{\text{best}} = \text{argmax}_{\mathbf{e}}\, p(\mathbf{e}|\mathbf{f})$$

- Our model is a weighted combination of many components

$$p(\mathbf{e}|\mathbf{f}) \propto \exp \sum_{k=1}^{m} \lambda_k \cdot h_k(\mathbf{e}, \mathbf{f})$$

where $h_k(\mathbf{e}, \mathbf{f})$ are *feature functions* such as

- translation and language model log-probabilities
- phrase and word counts
- etc.

and λ_k are *weights*.

Feature Weights

- Contribution of feature h_k determined by weight λ_k
- Methods for setting the feature weights:
  - *manually* — try a few, take best
  - *automatically* — tune with an **optimization algorithm**
- How to 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

Weight Optimization

- Setting the feature weights is an optimization problem:

$$\Lambda_{\text{best}} = \text{argmax}_{\Lambda} G(E, T_{\Lambda}(F))$$

- Find weight vector $\Lambda_{\text{best}} = (\lambda'_1 \cdots \lambda'_m)$ that maximizes some **gain function** G
- The gain function G compares a set of *reference sentences* E to a set of *translated sentences* $T_{\Lambda}(F)$
- *Which gain function?* Our evaluation metric (BLEU)!

Discriminative vs. Generative Models

- **Generative models**
  - translation process is broken down into *steps*
  - each step is modeled by a *probability distribution*
  - each probability distribution is estimated from the data by *maximum likelihood*

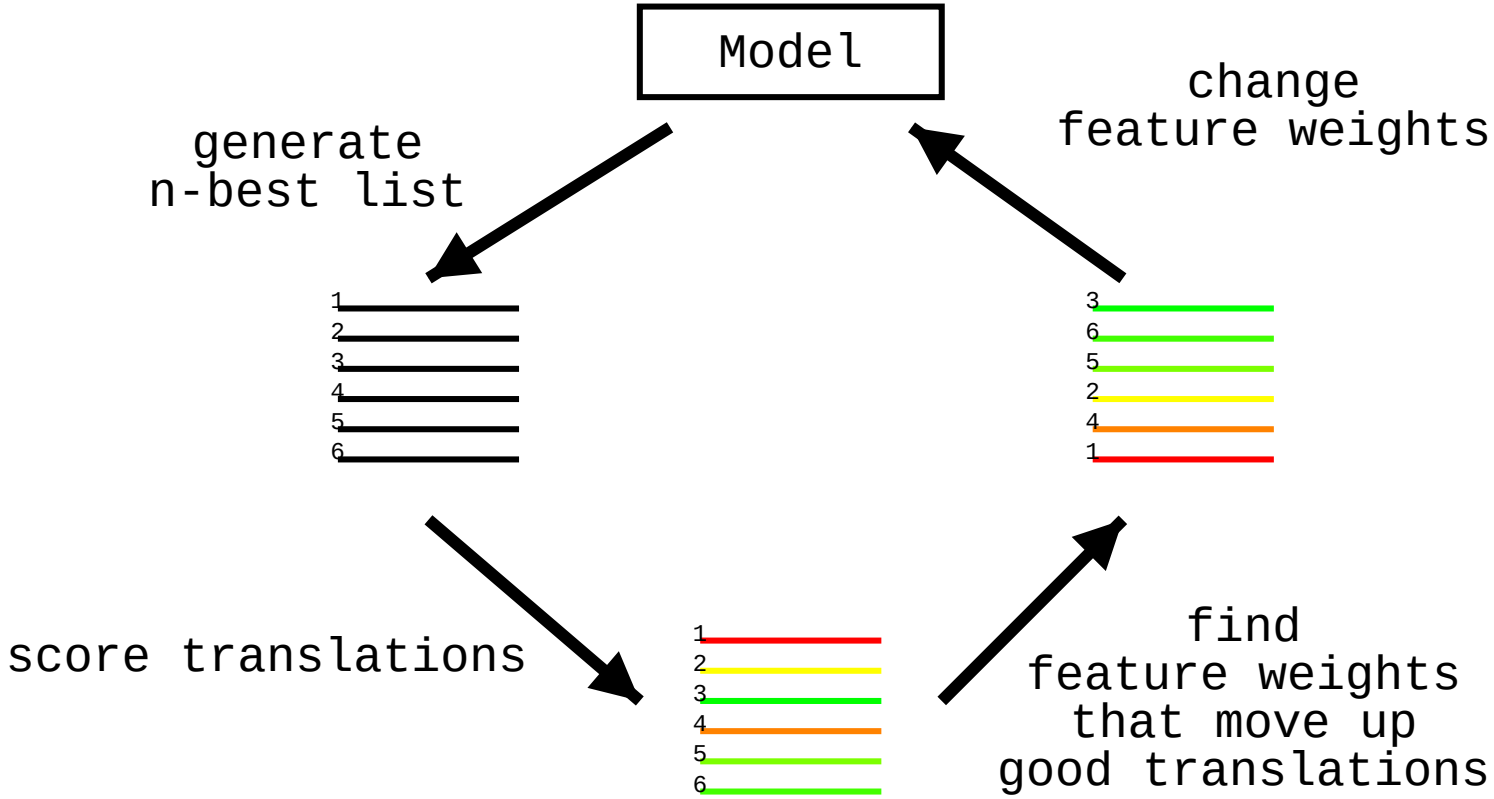
- **Discriminative models**
  - model consists of a number of *features*
  - each feature has a *weight*, measuring its value for judging a translation as correct
  - supervised learning: *directly tune model parameters* (feature weights) towards optimal performance wrt. the evaluation metric on development data

Discriminative Training (1)

- Employ *development corpus*
  - different from training corpus for phrase extraction
  - small (maybe 2000 sentences)
  - different from the held-out test set which is used to finally evaluate the translation quality

- *Translate* development corpus using model with current feature weights, output N-best list of translations ($N = 100, 1000, \ldots$)
- *Evaluate* translations with the gain function
- *Adjust feature weights* to increase the gain
- *Iterate* translation, evaluation, and adjustment of feature weights for a number of times

Discriminative Training (2)



Optimization on N-best Lists (1)

- Task: find weights so that the model ranks best translations first
- Input: *er geht ja nicht nach Hause*, Ref: *he does not go home*

Translation	Feature values		Model score	Gain
it is not under house	-2	-2	-0.6	0.2
he is not to go home	-0.5	-3	-0.65	0.33
he does not go home	-4	-1.5	**-0.7**	**1.0**
it is not packing	-3	-3	-0.9	0.0
he is not for home	-5	-6	-1.7	0.2

$\lambda_1 = 0.1, \quad \lambda_2 = 0.2$

Optimization on N-best Lists (2)

- Task: find weights so that the model ranks best translations first
- Input: *er geht ja nicht nach Hause*, Ref: *he does not go home*

Translation	Feature values		Model score	Gain
it is not under house	-2	-2	*-0.7*	0.2
he is not to go home	-0.5	-3	*-0.925*	0.33
he does not go home	-4	-1.5	**-0.65**	**1.0**
it is not packing	-3	-3	*-1.05*	0.0
he is not for home	-5	-6	*-2.05*	0.2

$\lambda_1 = 0.05, \quad \lambda_2 = 0.3$

Och's Minimum Error Rate Training (MERT)

- Given a set of N-best lists, how to adjust weights?
- **Line search** for best feature weights [Och, 2003]

```
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
```

MERT: Adjusting Feature Weights (1)

- The model score for a given hypothesis/source pair $(\mathbf{e}, \mathbf{f})$ is:

$$\text{score}(\mathbf{e}, \mathbf{f}) = \sum_{k=1}^{m} \lambda_i \cdot h_k(\mathbf{e}, \mathbf{f})$$

- If we're only interested in one single weight λ_c, $1 \leq c \leq m$, we can write

$$\text{score}(\mathbf{e}, \mathbf{f}) = \lambda_c \cdot h_c(\mathbf{e}, \mathbf{f}) + \sum_{k \neq c} \lambda_k \cdot h_k(\mathbf{e}, \mathbf{f})$$

  which is of the form

$$\text{score}(\mathbf{e}, \mathbf{f}) = A\lambda_c + B$$

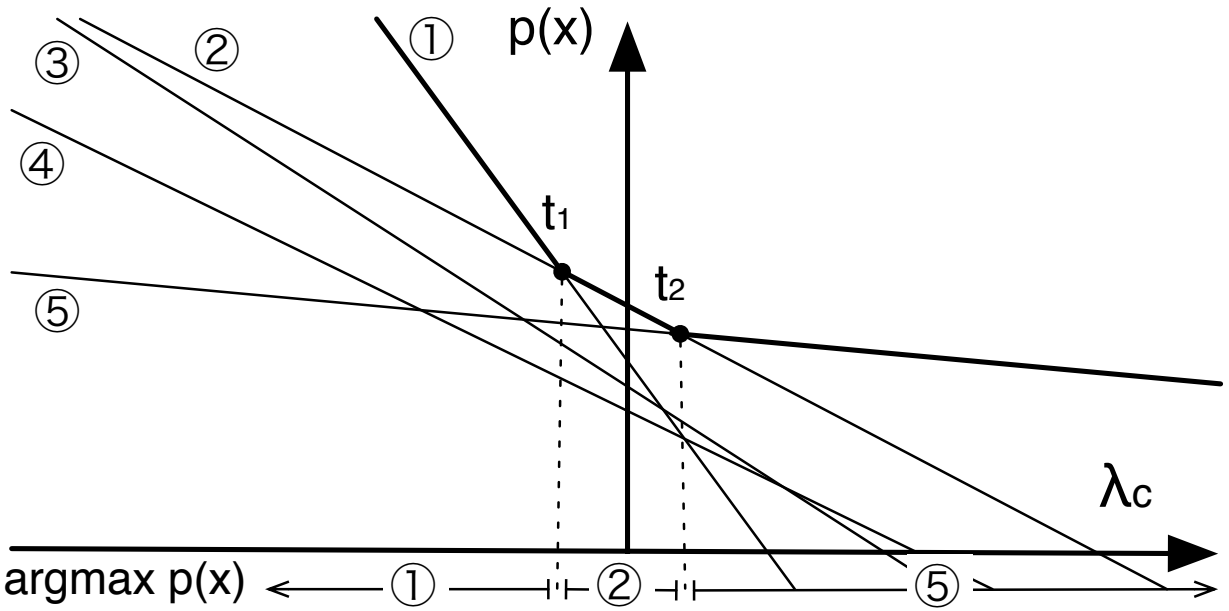
MERT: Adjusting Feature Weights (2)

- So the model score of each hypothesis in each N-best list is a *linear in a single weight* if we keep all other weights fixed

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

- Recall that:
  - we deal with 1000s of input sentences $\mathbf{f}$ in the development set
  - we deal with 100s of translations $\mathbf{e}$ per input sentence
  - we are trying to find the value λ_c so that over all sentences, the gain is optimized

MERT: Adjusting Feature Weights (3)



- Each translation from the N-best list contributes a line
- The model-best translation only changes at upper intersection points
- Evaluate gain of segments on upper envelope
- Set λ_c to a value within the interval with the highest gain

MERT: Assessment

- *Advantages*
  - Widely used, and several implementations available
  - Can be (and is) used with a variety of metrics
  - Converges in "reasonable" time
- *Disadvantages*
  - Only scales to 20-30 features
  - Stochastic algorithm – variable results
  - N-best lists give very limited view

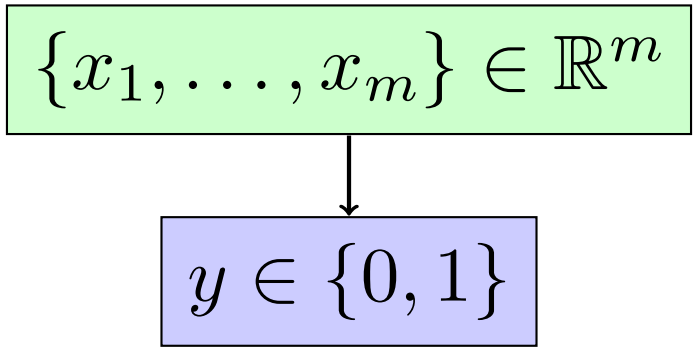
Pairwise Ranked Optimisation (PRO)

- An alternative: Pairwise Ranked Optimisation (PRO) [Hopkins and May, 2011]
- Treats the **optimization as a classification problem**
- Idea: We want the ranking induced by the model score function to be the same as by the gain function:

$$\text{score}(\mathbf{e}_i, \mathbf{f}) > \text{score}(\mathbf{e}_j, \mathbf{f}) \Leftrightarrow G(\{\hat{\mathbf{e}}\}, \{\mathbf{e}_i\}) > G(\{\hat{\mathbf{e}}\}, \{\mathbf{e}_j\}), \forall 1 \leq i, j \leq N$$

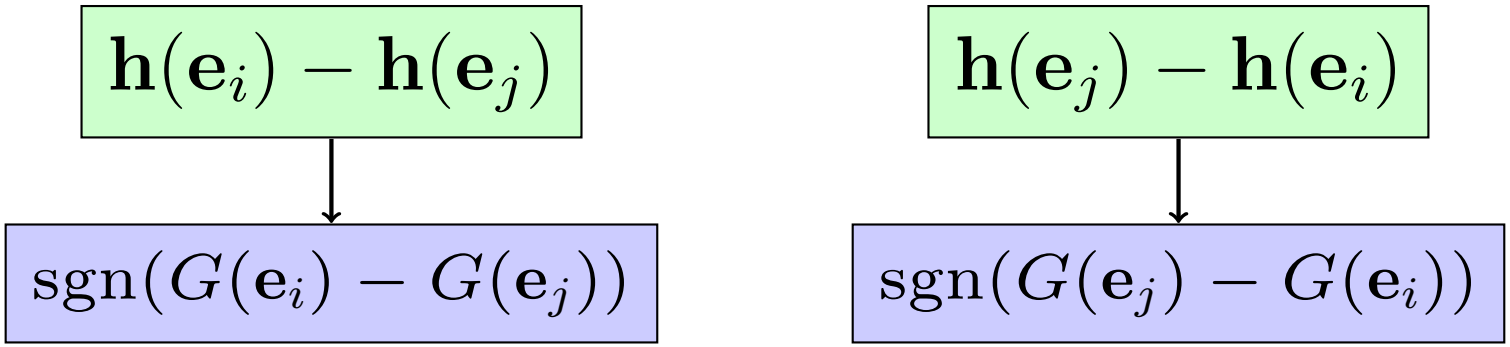
- $G()$ is a *sentence-level version of* BLEU, $\hat{e}$ denotes a reference sentence

Ranking as Classification (1)



- Binary classifier (e.g. logistic regression) maps vectors to boolean

Ranking as Classification (2)



- Sample $\mathbf{e}_i, \mathbf{e}_j$ with feature vectors $\mathbf{h}(\mathbf{e}_i), \mathbf{h}(\mathbf{e}_j)$ from N-best list
- Add two examples to classifier training set for each sample

PRO: Assessment

- *Advantages*
  - Scales to large numbers of features
  - More stable than MERT
  - Easy to implement

- *Disadvantages*
  - Uses sentence-level BLEU – different length penalty
  - Gives worse results for out-of-English
  - Still tied to N-best lists

Other Approaches

- Online learning [Chiang et al. 2008; Liang et al. 2006]
- Expected BLEU training [Smith and Eisner, 2006; Arun et al, 2010]
- Lattice MERT [Macherey et al, 2008]

Summary

- The role of tuning
  - optimize feature weights to maximize a gain function
  - on a development corpus
  - typically with N-best lists
- Methods
  - Minimum Error Rate Training (MERT)
  - Pairwise Ranked Optimisation (PRO)