

Statistical Language Models for Machine Translation

Some recent developments

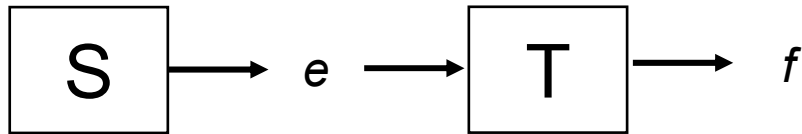
Language Models for MT

Today's topics:

- Motivation/Introduction
- Integration of LMs into Moses decoder
- Huge LMs used at Google
- Reducing space consumption via Bloom filters

Motivation

Stochastic language models have been developed for speech recognition and then adapted to MT, reusing the idea of the distorted channel:



From Brown e.a. 93:

Probability of sentence e
given observed f :

$$\Pr(\mathbf{e}|\mathbf{f}) = \frac{\Pr(\mathbf{e}) \Pr(\mathbf{f}|\mathbf{e})}{\Pr(\mathbf{f})}$$

Best translation \hat{e}
given observed f :

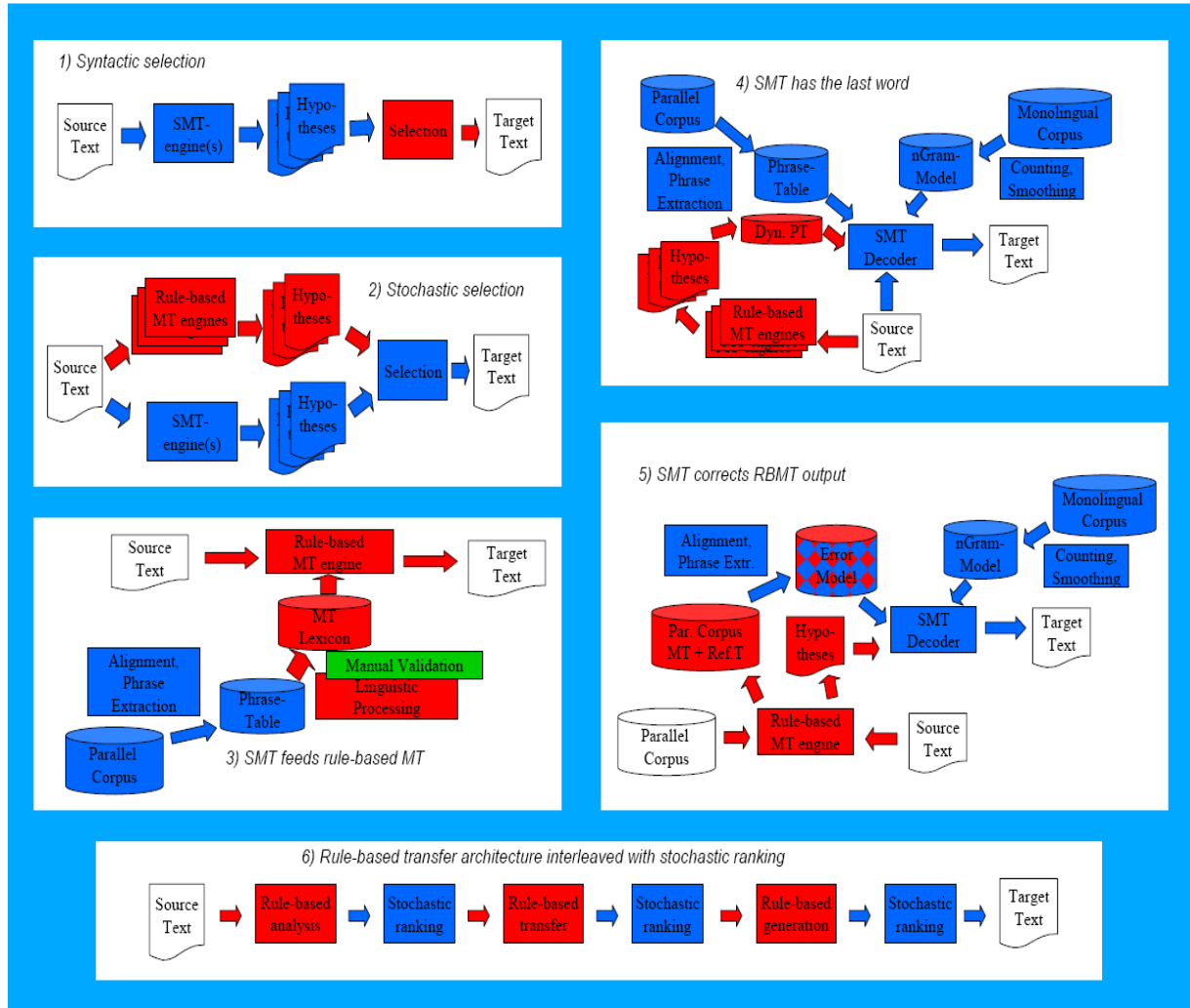
$$\hat{\mathbf{e}} = \underset{\mathbf{e}}{\operatorname{argmax}} \Pr(\mathbf{e}) \Pr(\mathbf{f}|\mathbf{e})$$

Division of labor:

$\Pr(\mathbf{e})$: How likely is e as an English sentence? (\sim fluency)

$\Pr(\mathbf{f}|\mathbf{e})$: How well does e correspond to f ? (\sim adequacy)

Use of statistical language models in hybrid architectures



Features used in language models

- In theory, LMs could exploit any feature that is helpful to distinguish good English sentences from bad ones
 - Syntax
 - Semantics
 - Word choice
 - Context
- In practice, it has been very hard to beat simplistic models based on n-gram counts over large corpora
 - Sophisticated models give minor improvements at high costs
 - Simple n-gram models also work well on partial results
 - integration into search algorithm is easier
 - „Throwing in more data“ typically helps more than constructing sophisticated models

n-Gram- versus linguistic Models

- It is well known since Chomsky 57 that n-Gram models cannot capture important long-distance dependencies
- However, they easily learn preferences wrt. lexical choice and short-distance effects from large corpora
- For English, long-distance effects are not frequent enough to compensate for the simplicity of n-Gram models
- So far, no grammar-based model has shown significant advantages for judging fluency
- This may change in the course of time (hopefully)

Typical formalisation of LMs

Decomposition into n-gram probabilities:

$$P(e) = P(e_1, \dots, e_n) = \prod P(e_i | e_1 \dots e_{i-1}) \approx \prod P(e_i | e_{i-n+1} \dots e_{i-1})$$

Estimation of n-gram probabilities

$$P(e_i | e_{i-n+1} \dots e_{i-1}) = \text{freq}(e_{i-n+1} \dots e_i) / \text{freq}(e_{i-n+1} \dots e_{i-1})$$

n-gram counts are small and „noisy“, whereas counts for smaller n are more reliable. Combining (smoothing) estimates from different n-gram lengths improves results :

$$\begin{aligned} P(e_i | e_{i-n+1} \dots e_{i-1}) &= \lambda_n \text{freq}(e_{i-n+1} \dots e_i) / \text{freq}(e_{i-n+1} \dots e_{i-1}) \\ &+ \lambda_{n-1} \text{freq}(e_{i-n+2} \dots e_i) / \text{freq}(e_{i-n+2} \dots e_{i-1}) \\ &+ \lambda_{n-2} \text{freq}(e_{i-n+3} \dots e_i) / \text{freq}(e_{i-n+3} \dots e_{i-1}) \\ &\dots \\ &+ \lambda_1 \text{freq}(e_i) / N \end{aligned}$$

Weights λ_i depend on the frequencies, higher weights for well-estimated sub-models: Backoff, Kneser-Ney Smoothing, ...

Integration of LMs into Moses decoder

See slides by M. Federico

Huge LMs used at Google

(Some highlights from Brants e.a. EMNLP-CoNLL 2007)

- Distributed Infrastructure
- So far used with up to 2 trillion ($2 \cdot 10^{12}$) tokens
- Simplified smoothing method: „stupid backoff“

Reducing space consumption via Bloom filters

See slides by Talbot