

11-731 Class Project Report

Estimating Better Position Alignments for Statistical Machine Translation

Nikesh Garera (ng+@cs.cmu.edu)

Advisors: Stephan Vogel, Robert E. Frederking
Language Technologies Institute
Carnegie Mellon University

Abstract

In Statistical Machine Translation, we often find forward or backward jumps while translating from a source position to a target position. We propose several position alignment models for estimating these jump probabilities. Our initial jump probability model is a coarse model with no dependencies. The maximum likelihood estimation of the jump probabilities is performed during post word alignment using maximal approximations from the IBM 4 word alignment model output. We systematically add intuitive parameters to provide more accurate models and evaluate these models based on their perplexity on a test set. We also report our results with smoothing via linear interpolation to account for data sparseness issues. The improvement in perplexity can provide an indication of how well the additional parameters model the jump probabilities. The best model can then be applied in various MT components like the decoder or in the iterative word-alignment model training itself.

1 Introduction

Statistical Machine Translation (SMT) systems view translation as a noisy channel and model the channel parameters that translate a source sentence (channel input) to a target sentence (channel output). This

is basically a data driven approach and has been shown to work very well for languages that have a large amount of parallel data available. Most of the SMT approaches work in two stages: the word alignment phase and the decoding phase. The word alignment consists of finding the probabilities of the possible set of alignments for the source words to target words. The decoder finds a maximal probable translation (via dynamic programming) using the word alignment probability tables and a language model. Most of the statistical machine translation systems, for example, the IBM models (Brown et al., 1993) and the HMM model (Vogel et al., 1996) model a set of alignment probabilities for modeling how the word positions in the source sentence map to the positions in the target sentence. For example, consider the French-English sentence pair shown in figure 1:

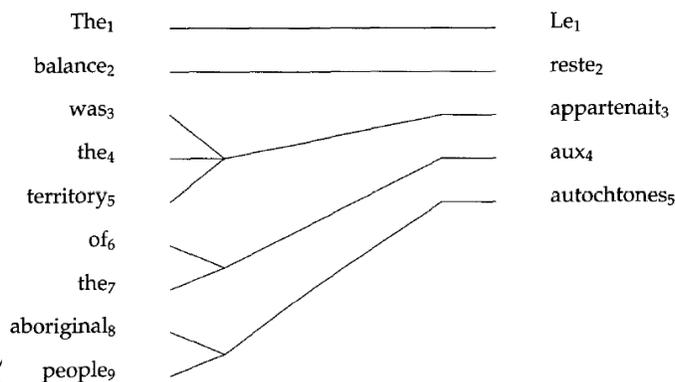


Figure 1: An example word alignment in a sentence pair. (Brown et al., 1993)

The alignment model should assign the target (English) word position (1) for source (French) word position (1). Similarly (2, {3,4,5}, {6,7}, {8,9}) are the correct target positions for their respective source positions (2,3,4,5). Jump probabilities are similar to alignment probabilities but instead of modeling absolute positions they model the forward and backward jumps between the source positions and the target positions. In the example shown in figure 1 the source word *the* at position 1 in the source sentence has a jump of 0, and the source word *aboriginal* has a jump of -3 (source position 8 to target position 5)

In this paper, we analyze a coarse model for modeling jump probabilities and propose richer models by adding intuitive dependencies (parameters) to the coarse model. We compare each of these models with respect to how well they model jump probabilities in unseen data. The rest of the paper is outlined as follows: Section 2 discusses related work on modeling alignment probabilities. Section 3 explains in detail the jump probabilities, the parameters used for different models and post alignment estimation of these models. Section 4 reports the parallel data used for evaluation, the results based on perplexity and analysis of the results. Section 5 points at interesting future directions and Section 6 concludes the paper with its summary.

2 Related work

Most of the Statistical Machine Translation systems use models that model the mapping between source and target positions. In the IBM models (Brown et al., 1993), the IBM Model 2 has an alignment probability model $a(i|j, m, l)$ where i is the target position, j is the source position, m and l are the lengths of the source and target language sentences respectively. The position mapping can also be performed from target to source side, the IBM Model 3 has a distortion model that models $d(j|i, m, l)$, the probability of a particular source position given a target position. IBM Model 4 models a richer distortion model by conditioning on a cluster of words along with the positions. This helps in increasing the accuracy of the model, but at the same time it needs more data to estimate the

additional parameters. In the HMM framework for Statistical Machine Translation (Vogel et al., 1996), the distortion model $P(a_j|l)$ takes into account only the jump width ($a_j - j$), where a_j is the target position for a given j source position. They also describe constructing a first order HMM model, $P(a_j|a_{j-1}, l)$, by adding the previous target position a_{j-1} as a parameter in the model.

Our work differs from the earlier work in several ways: first, we model relative jumps instead of absolute positions.¹ Second, the earlier systems estimate these model during word alignment in the EM iteration. We estimate the models from the maximal approximations (given by the Viterbi algorithm) post word alignment, with the motivation of using these models to help the decoder or to seed another word alignment model. Third, we also introduce additional parameters like the source word, target word which were not tested in the earlier models. We analyze the issue of data sparseness due to additional parameters and report our results with smoothing via linear interpolation.

3 Models

3.1 Jump Probabilities

A relative jump between two positions jump measures the difference between the two positions, and hence, it does not care about the actual positions as long the difference remains the same. The motivation here is that a jump of position 5 to position 7 should be regarded the same as the jump from position 4 to position 6 (jump width = 2). Specifically, we define jump as follows:

$$\text{jump} = \text{target position} - \text{source position}$$

Figure 2 shows how this jump models the relative distance from the diagonal points. The diagonal points are jumps of distance 0.²

¹This differs from IBM models which model absolute positions but the idea of modeling jump widths is similar to the (Vogel et al., 1996) approach.

²It is also possible to measure jumps along the target side or along the source side. However, in this work, we measure jumps along the diagonal, that is the difference between the target position and source position.

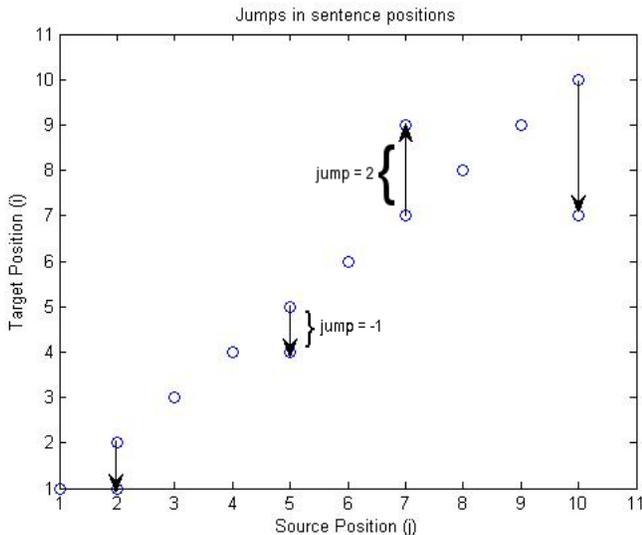


Figure 2: Forward and Backward Jumps in Sentence Positions

Table 1: Introducing additional dependencies

Model 1	$P(jump)$	
Model 2	$P(jump m)$	Source length
Model 3	$P(jump s_w)$	Source word
Model 4	$P(jump m, s_w)$	Source length, Source word
Model 5	$P(jump m, s_w, t_w)$	Source length, Source word, Target word

3.2 Model Parameters

Our goal is to model the probability of a particular jump during translation. Table 1 shows the additional parameters for different models. We start with a coarse model (Model 1) that has no dependencies. It simply estimates the jump probability by counting how many times the particular jump occurs in the alignments present in all sentence pairs (of different lengths, words, etc). However, in this model, there will be a lot of mismatch as the $jump = 5$ in a sentence of length 7 will be considered the same as $jump = 5$ in a sentence of length 12. It is more appropriate to compare the jump distances in the sentence pairs that have the same source and target lengths. It is difficult to include target length as one of the main objectives of modeling jump probabilities is to help the decoder in translating the source

words in correct target order. During decoding, the target length is not available and hence it is difficult to add target length in the model due to practical concerns. Nevertheless, it is still possible to use some form of expected target length and then correct for it during decoding. In this paper, we take only source length into account for practical reasons. Source length is introduced in Model 2 in the above table. It is also intuitive that the position jumps during translation depends upon the source word and target word at the respective positions. For example, determiners like *Le* in French will probably have small jumps (mostly $jump = 0$ or $jump = 1$) compared to other content words. Hence, we condition on source word in Model 3 to analyze the effect of words in the mapping of positions during translations. Model 4 adds source length along with source word to reduce the mismatch due to different lengths and Model 5 adds target word to these two parameters to analyze the effect of target words in position mappings.

Although, the richer models seem more intuitive as they help to model the data more accurately but there is a tradeoff between the complexity and the data sparseness problem. The more complex the model, the more data it will need to estimate. Specially in the models based on words, we are introducing thousands of additional parameters (as each word in the vocabulary becomes a parameter), increasing the data sparseness problem exponentially. However, we also discuss some smoothing techniques to take into account the sparsity problem.

3.3 Post Alignment Estimation

The actual jump probabilities for different models are estimated after word alignment probability tables are estimated using the IBM word alignment models. We use GIZA++ toolkit (Och and Ney, 2003) for estimating the word alignment probabilities using IBM Model 4. The toolkit also computes maximal approximations for the alignments via the viterbi algorithm and outputs the best set of alignments. We estimate the jump probabilities from these set of alignments using maximum likelihood estimation. Figure 3 provides a outline of the system architecture for estimating the jump probabilities.

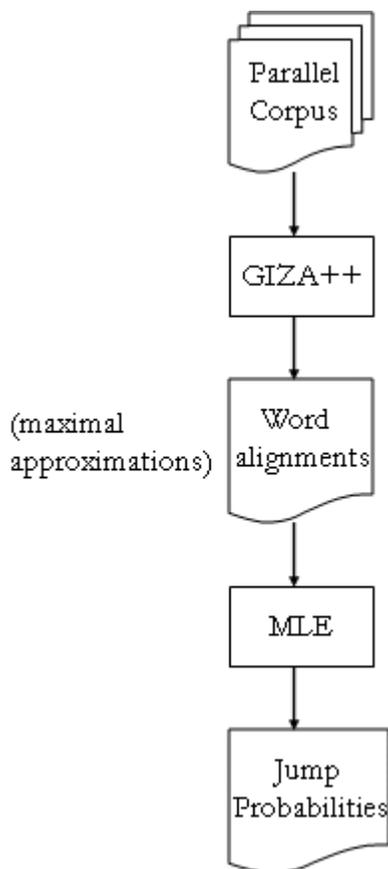


Figure 3: Architecture for estimating jump probabilities

3.4 Maximum Likelihood Estimation of Model Parameters

The maximum likelihood estimates of the jump probabilities are simply the fraction counts of the jumps in the set of word alignments obtained from GIZA++. We ignore jump from null words and jump to null words. Null words are represented as position 0 in GIZA++ toolkit. If we take them into account, then a source word at position 10 mapping to a null target word will produce a jump of -10, however, it is obvious that it is incorrect to assume such a long backward jump. It is more reasonable to assume that the source word simply gets deleted and hence we do not model the jump from null words and to null words. It might be possible to look at the context words of the source word that gets assigned to null words. Based on target positions of the context words, we can assign a target position in between those contextual target

Table 2: Sample estimates of jump probabilities

$P(\text{jump} = 0)$	0.16
$P(\text{jump} = 0 m = 1)$	0.94
$P(\text{jump} = 0 s_w = le)$	0.27
$P(\text{jump} = -1 s_w = le)$	0.12
$P(\text{jump} = 13 s_w = mutandis)$	1

positions instead of assigning target position 0 and calculating the difference from 0 as the jump.

The following equation shows the maximum likelihood estimate for Model 1:

$$p_{mle}(\text{jump } k) = \frac{f_k}{\sum_{i=1}^n f_i} \quad 1 \leq k \leq n, 1 \leq i \leq n$$

where f_i is the frequency of jump i , that is, the # of times jump i occurs in the corpus. Similarly, we can estimate other models using fraction counts. Some example estimates obtained from the EU parallel corpus (4.1) are shown in Table 2.

The computed estimates above seem very intuitive, for example when the source length has only one word, the probability of it jumping to the same position in target sentence (that is, jump=0) is 94%. Similarly, the highest jump probability for source word = *le* is for jump=0 (27%), this is expected as determiners are expected to have short jumps. Also, notice that rare words, particularly singletons will have probability one. For example, source word *mutandis* occurs only once in the corpus, hence the probability of a jump with source word = *mutandis* is one. Overall, the models seem to give higher probability mass for short jumps compared to the long jumps, assuming there is no sparsity problem (that is, excluding rare cases such as singletons, etc.)

4 Evaluation

4.1 Data

The parallel corpus for training the IBM Model 4 was the French-English corpus obtained from the European Parliament Proceedings (1996-2003). The corpus resource is known as Europarl (Koehn, draft) and includes versions in 11 different languages. We took the top 50,000 sentence pairs from this dataset as our parallel corpus for the experiments. The cor-

pus was cleaned up to introduce spaces around punctuation marks and special characters. Also all the words were converted to lower case. We randomly selected 1000 sentence pairs as the development set, another random set of 1000 sentence pairs as the test set and the remaining 48,000 sentence pairs as the training set.

4.2 Metrics

Once the jump probability models are trained on the 48,000 sentence pair training set, we evaluate them by computing the respective perplexities on the test set. Perplexity (Jelinek et al., 1977) is a measure of difficulty in modeling the test set given the models trained from the training set. The better the model fits the test set, the lower the perplexity. Perplexity is defined as follows:

$$pp = 2^{-l} \text{ and } l = \frac{1}{t_n} \sum_{i=1}^{t_n} \log P(jump_i | c_1, c_2, ..)$$

where,

pp = perplexity,

l = average log likelihood,

t_n = total # of word alignments in test corpus,

$jump_i$ = jump in alignment i ,

and $c_1, c_2, ..$ = conditional parameters depending on the model.

4.3 Results

In the following subsections, we report our results for the training set perplexity and test set perplexity across different models.

4.3.1 Training Set Perplexity

The five jump probability models were estimated using the training set. Ideally, as the models become more complex (richer), the perplexity on the training set should decrease, since there is no unseen data and by adding additional parameters, we are estimating the models from smaller subsets of data. Table 3 reports the perplexity of the different models on the 48,000 sentence pair training set.

As expected, the training set perplexity reduces and the average log likelihood increases as we move from coarse model to the richer models. This indi-

Table 3: Training Set Perplexity

Model	Avg. Log Likelihood	Perplexity
Model 1	-4.26	19.10
Model 2	-4.19	18.20
Model 3	-3.63	12.39
Model 4	-2.68	6.39
Model 5	-1.78	3.43

Table 4: Test Set Perplexity

Model	Avg. LL	PP	% of zerotons
Model 1	-5.08	33.73	0/16740 (0%)
Model 2	-4.99	31.70	5/16740 (0.03%)
Model 3	-3.13	8.76	4712/16740 (28%)
Model 4	-0.77	1.71	12701/16740 (76%)
Model 5	-0.05	1.04	16463/16740 (98%)

cates that the richer models are fitting the jump probabilities better on the training set, which they should, as the jump probabilities were estimated from the training set itself.

4.3.2 Test Set Perplexity

Test set is a random sample of 1000 sentence pairs that was not used in the training set, that is, not used while estimating the probability models. Hence, it is possible that there might be some unseen jumps in the test set alignments. Initially, we report our results by throwing out these unseen jumps and assuming that these alignments with unseen jumps did not exist in the test corpus. Since we are not taking into account the unseen jumps, the perplexity on the *seen part* will reduce ideally if the more complex models are estimated correctly and the test set is coming from the same distribution as the training set ³. Table 4 reports on the perplexity of different models on the 1000 sentence pair test set. We can see that the test set perplexity goes down as the models become more complex. The reason for the reduction in perplexity is that the unseen part of the test set is removed hence it becomes easier to predict the jump probabilities in the test set.

Notice that the table shows how much of the

³It is reasonable to assume that the test set has a similar distribution on the training set since test set was produced by chopping off some part of the training set

unseen part is removed by the % of zerotons column. The % of zerotons removed increase as we go to more complex models. Although, the perplexity in Table 4 goes down as we go to more complex models, however, a large fraction of the alignments are unseen in the richer models. Hence, this is not a good estimate of the test perplexity and we will need to do some smoothing to take care of the unseen alignments. Also, notice that the % of zerotons in the coarse model is 0, this is expected since the coarse model does not have any dependencies which allows it to have a large data for estimating its single parameter, the jump probability (there are no conditional parameters). In the following sections, we will make use of this advantage of the coarse model to smooth the richer models.

4.3.3 Smoothing via Linear Interpolation

Linear interpolation (Jelinek and Mercer, 1980) is a technique for combining different models, specifically, the combined or the interpolated model is a linear combination of the complex models and simpler models. We formalize the interpolated model as follows:

$$P_{comb} = \lambda_{M_{>1}} P_{M_{>1}} + (1 - \lambda_{M_{>1}}) P_{M_1}$$

where, $M_{>1}$ can be any one of the richer models (Models 2,3,4 and 5) and M_1 is the coarse Model 1. Generally, the weight $\lambda_{M_{>1}}$ is tuned using Expectation Maximization (Dempster et al., 1977) over a development set. Since there is only one parameter, we conduct a line search ⁴ over this parameter and see at which setting the P_{comb} gives the minimum perplexity to tune the weight.

Development set perplexity

The development set was obtained as mentioned in the section 4.1 earlier. Linear interpolation for each of the four models (Model 2,3,4,5) combining it with the coarse model (Model 1) was formulated and the optimal λ 's for each of the complex models were obtained. The weight tuning was done on the development set which is different than the test set. Table 5 reports the development set perplexity and the

⁴A line search simply starts with $\lambda = 0$, computes the perplexity, increases λ by a small positive step, computes the perplexity and iterates over this procedure till $\lambda = 1$

Table 5: Development Set Perplexity (interpolated)

Model	Avg. LL	Dev_PP	$\lambda_{M_{>1}}$
Model 1	-5.135	35.14	N/A
Model 2	-5.067	33.53	0.92
Model 3	-5.135	35.14	0.001
Model 4	-5.1356	35.16	0.001
Model 5	-5.1364	35.18	0.001

Table 6: Test Set Perplexity (interpolated)

Model	Avg. LL	Test_PP	$\lambda_{M_{>1}}$
Model 1	-5.0760	33.7330	N/A
Model 2	-4.9922	31.8290	0.92
Model 3	-5.07616	33.7347	0.001
Model 4	-5.07647	33.7419	0.001
Model 5	-5.07732	33.7619	0.001

weights obtained for each of the complex models. Note that the weight for the coarse model (Model 1) is not shown as it is simply $1 - \lambda$.

Test set perplexity

Using the tuned weights obtained from tuning on development set, we compute the test set perplexity for the interpolated model. Table 6 reports the test perplexity. Note that the test set perplexity is similar to the development set perplexity. This might seem odd at first as the weights were tuned to the development set. It turns out that the weights tuned on the development set are also near optimal weights for the test set, that is, if we tune weights on the test set, we will get very similar weights to the weights obtained above. Hence the perplexities for both the sets are similar. There is some variation due to idiosyncrasies of the respective datasets.

Analysis of smoothing results

We note that there are no zerotons when we estimate the combined P_{comb} model, this follows from the fact that the coarse model with which the complex models are interpolated does not have any zerotons. Thus the P_{comb} model will always have some probability for any jump found in the development set, even if the richer models report the jump as unseen.

The results on the development set due to smoothing are a little surprising. Although, it is encouraging to see the reduction in perplexity from Model 1 to Model 2, however, the perplexity increases as we go to the richer models (Model 3, 4,5). Also, only Model 2 gets a higher weight (0.92) during linear interpolation. The other models (Models 3, 4 and 5) obtain a very small weight during linear interpolation and most of the weight gets assigned to the coarse model. This is also the reason why Models 3, 4, 5 have perplexity similar to Model 1 perplexity as most of the weight for the combined model is coming from Model 1. The increase in perplexity in richer models (Models 3, 4 and 5) and also their small interpolation weights could be due to the huge data sparseness problem in the models that are based on the the words ⁵ Also, a large amount of data is available for the coarse model due to which the weights might increase in the favor of the coarse model. It might be a good idea to interpolate the complex models with less coarse models such that the zero-ton problem can still be avoided. It is also promising to interpolate the models based on words, with the coarse model as well as the source length model (Model 2), hence we have an interpolation of three models as follows:

$$P_{comb} = \lambda_{M_{>2}}P_{M_{>2}} + \lambda_{M_2}P_{M_2} + \lambda_3P_{M_1}$$

$$\text{where } \lambda_3 = (1 - (\lambda_{M_{>2}} + \lambda_{M_2}))$$

This will have the advantage that whenever the data is sparse for the word based model, the combined model can still resort to the source length model and in the worst case it can get some probability mass from the coarse model (Model 1).

Another reason for richer word based models not giving improvement could be due the difference in IBM distortion model during word alignment and our distortion model. It might be better to use our models in GIZA++ during the EM iterations for computing the word alignments on training/test set and then check the improvement in word alignment perplexities.

⁵Recall that each word in the vocabulary is a parameter. This gives rise to a huge number of parameters.

We can also improve the estimates for the richer models by using Good Turing (Good, 1953) estimates. This will help the interpolation to take some probability mass from the richer model, even for zero-ton. ⁶

5 Future Work

We need to look at better smoothing techniques discussed in the Analysis section and find out how much the different models contribute during linear interpolation for perplexity reduction. Also, computing and evaluating the word based models loosely based on maximal approximations might not be the best way to make use of the richer models. We can integrate our models more tightly by incorporating them in the word alignment computations in the GIZA++ toolkit. This work also sheds light on the data sparsity issue which can be useful in selecting additional parameters. For example, it might be useful to add expected target length or a linguistic class of words since these parameters are less sparse compared to the word based parameters.

We also need to evaluate these estimated jump probability tables for improving Machine Translation. The jump probabilities can potentially help seed another word alignment model and help in improving word alignments. The improvement in word alignments can be evaluated using metrics like AER (Och and Ney, 2000), Precision, Recall and F-measure. We can also use these jump probabilities to guide the decoder. The current CMU SMT system uses a fixed Gaussian model for the jump probabilities with the highest density at 0 or very small jumps. The Gaussian density decreases as the jumps become longer. However, this is a very coarse model and it will be interesting to replace this model by the jump probability models proposed in this paper. We can measure the improvement in decoder output using METEOR (Lavie et al., 2004), (Lavie and Banerjee, online reference) and BLEU (Papineni et al., 2002) metrics.

⁶Good Turing is a smoothing technique that takes some probability mass from singletons and gives it to zero-ton, takes some mass from doubletons and gives it to singletons and so on.

6 Summary and Conclusion

We identified the problem of estimating better models for position translations in Statistical Machine Translation. We motivated modeling relative jump widths for this problem and proposed several richer models for modeling the jump probabilities. A post word alignment framework was described to estimate these probabilities and the different models were compared based on their perplexity on the test set. Among the different models, the jump model based on sentence length seems most promising based on the perplexity results compared in this paper. This paper also highlights the sparsity issue in word based models and proposes several techniques to improve the smoothing results. In our future work, we suggest better ways of incorporating the word based models in Statistical Machine Translation systems. We also plan to plug our models in the SMT decoder and test the improvement in actual translation - the ultimate goal of machine translation.

References

- Peter F. Brown, Vincent J. Della Pietra, Stephen A. Della Pietra, and Robert L. Mercer. 1993. The mathematics of statistical machine translation: parameter estimation. *Comput. Linguist.*, 19(2):263–311.
- A. P. Dempster, N. M. Laird, and D. B. Rubin. 1977. Maximum likelihood from incomplete data via de EM algorithm. *The Journal of Royal Statistical Society*, 39:1–37.
- I. J. Good. 1953. The population frequencies of species and the estimation of population parameters. *Biometrika*, 40:237–264.
- Frederick Jelinek and Robert L. Mercer. 1980. Interpolated estimation of markov source parameters from sparse data. In *Proceedings of the Workshop on Pattern Recognition in Practice*, Amsterdam, The Netherlands: North-Holland, May.
- F. Jelinek, R. L. Mercer, L. R. Bahl, and J. K. Baker. 1977. Perplexity - a measure of the difficulty of speech recognition tasks. *Program of the 94th Meeting of the Acoustical Society of America J. Acoust. Soc. Am.*, 62:S63. Suppl. no. 1.
- Philipp Koehn. draft. A multilingual corpus for evaluation of machine translation. *Draft, Unpublished*, <http://people.csail.mit.edu/koehn/publications/europarl.ps>.
- A. Lavie and Satanjeev Banerjee. (online reference). The meteor automatic machine translation evaluation system. <http://www.cs.cmu.edu/~alavie/METEOR/>.
- A. Lavie, K. Sagae, and S. Jayaraman. 2004. The significance of recall in automatic metrics for mt evaluation. In *6th Conference of the Association for Machine Translation in the Americas (AMTA-2004)*, Washington, DC, September.
- Franz Josef Och and Hermann Ney. 2000. A comparison of alignment models for statistical machine translation. In *COLING*, pages 1086–1090.
- Franz Josef Och and Hermann Ney. 2003. A systematic comparison of various statistical alignment models. *Computational Linguistics*, 29(1):19–51.
- Kishore Papineni, Salim Roukos, Todd Ward, and Wei jing Zhu. 2002. BLEU: a method for automatic evaluation of machine translation, September 12.
- Stephan Vogel, Hermann Ney, and Christoph Tillmann. 1996. Hmm-based word alignment in statistical translation. In *COLING*, pages 836–841.