Natural Language Processing 2
Project 1: Lexical Alignment

Francois Meyer
12198145
francoisrmeyer@gmail.com

Vasilis Charatsidis

Fiorella Wever
12101745
fiorella.wever@gmail.com

Abstract

The IBM translation models remain influential in the field of statistical machine translation (SMT). For this project we implement and compare IBM model 1 and IBM model 2. In this report we describe the models theoretically and present the results of various experiments performed with the models. We found that the word-based approaches underlying these models still hold up as useful models for lexical translation and alignment.

1 Introduction

Machine translation is an exciting area of research in NLP. Recent advances in deep learning have contributed greatly to the success of machine translation models (Young et al., 2018). These models rely on state-of-the-art techniques such as LSTM networks, Sequence2Sequence architectures (Sutskever et al., 2014), and attention mechanisms (Luong et al., 2015).

The main idea of SMT is to train the parameters of a model on a bilingual corpus such that the model learns to translate from a source language to a target language. The ability of deep learning models to learn complex relationships in data and generalise effectively have led to impressive results in SMT. However, it is interesting to note that much of the techniques driving the recent successes in machine translation can be traced back to earlier SMT models.

Some of the most influential early models in SMT are the IBM alignment models (F Brown et al., 1993). They are five models of increasing complexity that consist of different components of a machine translation system such as word alignments, lexical translation probabilities, and word fertilities. The IBM models are all word-based and trained on a bilingual corpus consisting of pairs of equivalent sentences in the source and target language. The models differ in the assumptions that they make and the parameters that must be trained.

In this project we consider the two simplest of these models - IBM model 1 and IBM model 2. In Section 2 we provide technical descriptions of both of these models. We implement both models and train them on a large parallel corpus consisting of pairs of English and French sentences. We report various results for our trained models, including plots that show how their training progresses and performance measures obtained on a test set. In Section 3 we describe these experiments and present our results. We conclude the report by discussing our findings in Section 4.

2 Models

In this section the models are described in detail. We discuss the assumptions underlying the models and provide a theoretical outline of them. Throughout the description we refer to F as the vocabulary of the target language and E as the vocabulary of the source language. We also refer to the training set D as consisting of n pairs of translated sentences from each language. When we refer to individual sentence pairs we assume that the source sentence consists of l words, the target sentence consists of m words, and each target word has an alignment variable a aligning it to one of the l words in the source sentence (or the NULL word). We estimate the parameters both models with the EM algorithm (since we have limited space in this report we include its description in the Appendix).

2.1 IBM1

IBM Model 1 is a translation model that learns word-based lexical translation probabilities from

a bilingual corpus under very simplified assumption regarding word alignments. It consists of the following parameters:

- $t(f|e)$ for all words $f \in F$ and $e \in E \cup \{NULL\}$ are the probabilities of generating word f conditioned on word e.

The model makes the following assumption:

- The alignment of each target word is distributed uniformly over all the words in the corresponding source sentence and the NULL word.

The translation probability of a target sentence $\mathbf{f}$ given a source sentence $\mathbf{e}$ is

$$p(\mathbf{f}, a|\mathbf{e}) = \frac{\epsilon}{(l+1)^m} \prod_{j=1}^{m} t\left(f_j|e_{a(j)}\right),$$

where a are the alignments and ϵ is a constant for normalization. We add 1 to l in the normalisation term's denominator because of the NULL word that is added to the start of each source sentence.

An example with English as the target language F and German as the source language E would be the sentence "das haus" and the target sentence "the house" ($l = 2$ and $m = 2$):

das		*haus*	
f	$t(f\|e)$	f	$t(f\|e)$
the	0.7	*house*	0.8
that	0.15	*building*	0.16
which	0.075	*home*	0.02
who	0.05	*family*	0.015
this	0.025	*shell*	0.005

$$\begin{aligned} p(f, a|e) &= \frac{\epsilon}{3^2} \times t(\text{the}|\text{das}) \times t(\text{house}|\text{haus}) \\ &= \frac{\epsilon}{3^2} \times 0.7 \times 0.8 = 0.062\epsilon \end{aligned}$$

2.2 IBM2

One drawback from the IBM Model 1 is that it takes different re-orderings of words in a translated sentence as equally likely, so it would not notice the difference between "I live in the building with the red door" and "I live in the red building with the door" if this was translated let's say from the Spanish sentence: "Vivo en el edificio con la puerta roja". As mentioned before, IBM Model 2 is an extension of and improvement on IBM1 since it takes this into account and uses an additional model for word alignment.

So now, for IBM2 we again have the following parameters:

- $t(f|e)$ for all words $f \in F$ and $e \in E \cup \{NULL\}$ are the probabilities of generating word f conditioned on word e.

But now we take the word alignment probabilities into account:

- $a(a_j|j, l, m)$ for any $l \in \{1 \dots L\}$, $m \in \{1 \dots M\}$, $i \in \{1 \dots m\}$, $j \in \{0 \dots l\}$ which is the probability of an alignment value a_j taking a particular value i, given l and m, the lengths of the source sentence and the target sentence, respectively. This should satisfy the constraint: $\sum_{i=0}^{l} a(i|j, m, l) = 1$ for each combination of j, m and l.

Given these assumptions, the translation probability of a target sentence f is

$$p(\mathbf{f}, a|\mathbf{e}) = \epsilon \prod_{j=1}^{m} \sum_{i=0}^{l} t\left(f_j|e_{a_j}\right) a(a_j|j, m, l),$$

where we have e_0 to be the NULL word.

Since the IBM2 model uses non uniform alignment probabilities, there are too many alignment parameters to be learned, which would lead to very sparse distributions, specifically: L^2 x M^2 To solve this, we use a jump distribution (Vogel et al., 1996), which is cheaper to parameterise. This approach allows us to make the alignment probabilities dependent on the relative position of the word alignment, rather than the absolute position. Here we define the jump function as following (Vogel et al., 1996):

$$\delta\left(a_j, j, l, m\right) = a_j - \left\lfloor j \tfrac{l}{m} \right\rfloor$$

The probability of alignment is then:

$$a(a_j|j, l, m) = \text{Cat}(\Delta|\delta)$$

Δ is a vector of parameters called the "jump probabilities", where $\Delta = \langle \delta_{-L}, \dots, \delta_L \rangle$ and it takes values from -longest to +longest. A jump specifies the idea of mismatch in linear order between the target language and the source language. The result of this is that we will end up with a much smaller number of parameters: 2 x L.

2.3 Neural IBM1

In neural IBM model 1 we substitute the counting that happens to the M-step with a simple neural network.

The neural network consists of a linear layer z_0 that is fed into a tanh activation function. The result is fed to another linear layer z_1 and finally we have a softmax layer that substitutes the counts. The translation probability t_{lex} now is calculated as shown in Figure 1. Categorical Cross Entropy is used as loss function and Adam as the optimizer.

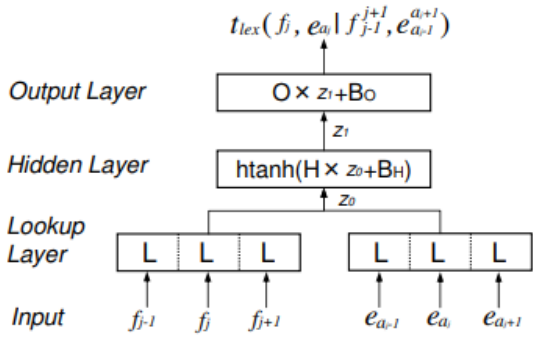


Figure 1: Feed forward model to compute translation probability

3 Experiments

3.1 Data

The data used throughout the experiments is from official records of the Canadian Parliament[1]. The full data set consists of 1.3 million pairs of aligned sentences (or smaller chunks of text) in English and French. We use a subset of this data set to train, validate, and test our models. We train our models on 231,164 sentence pairs, continually track their performance after each iteration of training on a validation set of 37 sentence pairs, and finally assess their performance on a test set of 447 sentence pairs. The validation and test sets have gold-standard alignments that can be used to assess the performance of our models.

3.2 Experimental setup

There are various evaluation metrics that can be used to quantify an alignment model's performance on a given data set. We use the alignment error rate (AER), a measure that combines precision and recall and is defined as

$$AER = 1 - \frac{|A \cap S| + |A \cap P|}{|A| + |S|},$$

where A is the set of alignments predicted by the model for the given data set, S is the set of *sure* alignments and P is the set of *possible* alignments (as specified by the gold-standard alignments of the data set).

We train each of our models on the training set for 20 iterations of the EM algorithm. After each iteration we compute and store the training log likelihood and the validation AER, allowing us to track the training progress of the models. For each of our IBM models, we select two models by deciding on the number of training iterations based on (1) when the training log likelihood of the model converges (improves by less than 0.1% in consecutive iterations) and (2) when the best validation AER is obtained. For the two selected models we report the AER on the test set and compare the different IBM models in Table 1.

3.3 Results

IBM1

The evolution of the training log likelihood of IBM model 1 is shown in Figure 2. It initially increases rapidly, but around iteration 5 it starts to plateau. According to our convergence condition it converges by iteration 14.

The evolution of the validation AER of IBM model 1 is shown in Figure 3. It initially decreases rapidly and then fluctuates randomly for the remaining iterations. The lowest validation AER is achieved (likely as a result of random fluctuation) at iteration 10.

It is interesting to note that the the training log likelihood increases monotonically but the validation AER starts fluctuating after only a few iterations. This is an indication that very few iterations of training are required for the model to produce good results on unseen data. After the first few iterations of training the model starts overfitting on the training data without improving significantly in terms of performance on the validation data.

The two selected models (at iterations 10 and 14) are evaluated on the test data and the results are presented in Table 1. In terms of test AER, the model selected for training log likelihood convergence narrowly outperforms the model selected for validation AER. This is somewhat surprising, since validation set performance is generally seen

[1] https://www.isi.edu/natural-language/download/hansard/

IBM Model	Selection criteria	Test AER
1	Training log likelihood	0.3048
1	Validation AER	0.3052
2	Training log likelihood	0.2915
2	Validation AER	0.2921
Neural 1	Validation AER	0.44

Table 1: The test set AER obtained by all of our selected models. For each model type (IBM 1, IBM 2, and neural IBM 1) two models were selected, based on convergence of the training log likelihood and the best validation set AER.

as a better indicator of test set performance. However, based on the evolution of the training log likelihood and validation AER, it seems after the first few iterations of training the model reaches a point after which additional training does not lead to improved generalisation. After this point it is expected that models perform similarly on the test data.

Figure 2: The training log likelihood of IBM model 1 plotted against the iterations of the EM algorithm (point of convergence shown in red).

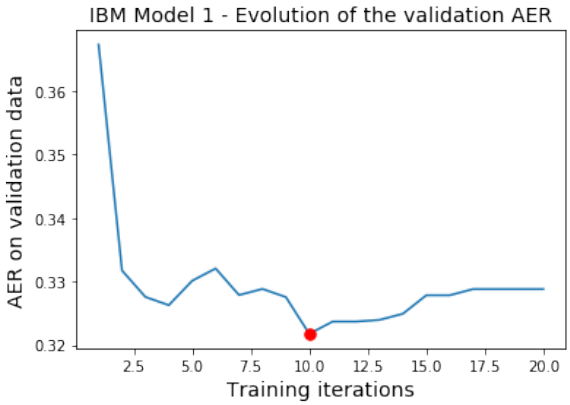


Figure 3: The validation AER of IBM model 1 plotted against the iterations of the EM algorithm (point of best performance shown in red).

IBM2

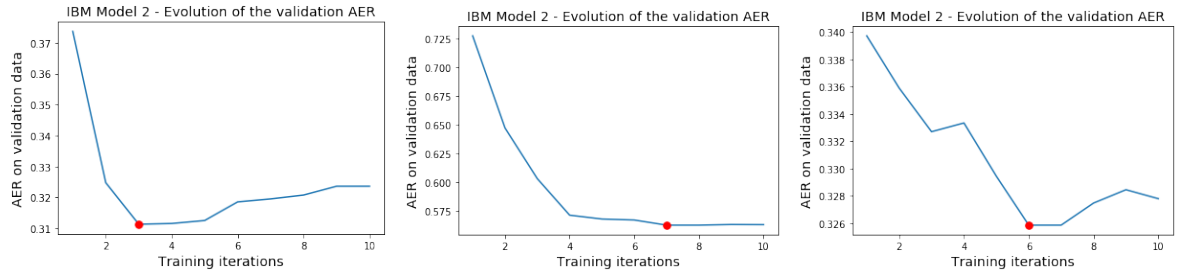
For IBM Model 2, we train the model using three different methods to initialise the model parameters: uniform initialization, random initialization, and initializing the parameters using the output of IBM1. The evolution of the training log likelihood of IBM model 2 for the three different initializations is shown in Figure 4. For the uniform and random initialization the evolution of the log likelihood is similar to that for IBM1. It initially increases rapidly, but around iteration 5 it starts to plateau. According to our convergence condition they both converge by iteration 9. However, for the initialization using the trained output of IBM1, it also initially increases rapidly, but around iteration 2 it starts to decrease. The convergence happens around iteration 3.

The evolution of the validation AER of IBM model 2 for the three different initializations is shown in Figure 5. For the uniform initialization, it initially decreases rapidly and then starts to increase with some random fluctuations. The lowest validation AER is achieved at iteration 3. The random initialization also starts with a decrease in AER and then around iteration 7 it starts to plateau, which is also the point that the lowest validation AER is achieved. For the method using the IBM1 results, we can see that it first decreases, but at iteration 3, where the model had converged, it increases a bit to decrease again. Then seems to increase a bit again. The lowest validation AER is achieved at iteration 6.

Similar as with IBM1 results, when we look at the uniform initialization for IBM2, we see that the training log likelihood increases monotonically, but the model reaches its lowest validation AER after just a few iterations. We don't need a lot

(a) Uniform initialisation (b) Random initialisation (c) Trained IBM1 initialisation

Figure 4: The training log likelihoods of IBM model 2 with three different initialisations plotted against the iterations of the EM algorithm (points of convergence shown in red).



(a) Uniform initialisation (b) Random initialisation (c) Trained IBM1 initialisation

Figure 5: The validation AERs of IBM model 2 with three different initialisations plotted against the iterations of the EM algorithm (points of best performance shown in red).

of iterations of training for the model to produce good results on unseen data. Comparing this with IBM1, we can see that IBM2 reaches the lowest validation AER earlier than IBM1, at iteration 3 rather than 10.

Looking at Figure 5, we notice how much the initialization of the parameters have an impact on the performance of the model. In particular, it is curious to observe the much higher AER values obtained as a result of the random initialization compared to the other two initializations. A reason for this is that the model is very dependent on good initialization and struggles to recover once it starts out in a bad place.

The two selected models for the uniform initialization of IBM2 (at iterations 9 and 3) are evaluated on the test data and the results are presented in Table 1. Again, the model for training log likelihood convergence slightly shows better performance compared to the model selected for validation AER. Comparing these results in Table 1 with the results of IBM1, we do see an improvement in terms of the AER in the test set. This was expected, since the IBM Model 2 now has an additional model for word alignment, which leads to better results in terms of translation probabilities.

Neural IBM1

The final test AER of our neural IBM1 model is shown in Table 1. After training the neural IBM1 model we obtain a test AER of 0.44, which is worse than both IBM1 and IBM2. This is somewhat surprising, as it was expected that the neural model would improve on the count-based models. It is possible that achieving this would require extensive fine-tuning of the neural model's architecture and hyperparameters. As is often the case with neural networks, it could also be that the neural model would improve if trained with more sentence pairs.

4 Conclusion

In this paper, we compare the implementation of IBM1, IBM2 and a Neural IBM1 model on a corpus of aligned sentences in English and French. Our results point to IBM2 outperforming IBM model 1, which was expected, since IBM2 takes word alignment in the translated sentence into account. For IBM2, we also notice a better performance when the parameters are initialized uniformly compared to other initializations. This points to the importance of the initialization of the parameters when training the IBM models.

References

F Brown, P., Della Pietra, S., Della J Pietra, V., and Mercer, R. (1993). The mathematics of statistical machine translation: Parameter estimation. *Computational Linguistics*, 19:263–311.

Luong, M.-T., Pham, H., and Manning, C. (2015). Effective approaches to attention-based neural machine translation.

Sutskever, I., Vinyals, O., and V. Le, Q. (2014). Sequence to sequence learning with neural networks. *Advances in Neural Information Processing Systems*, 4.

Vogel, S., Ney, H., and Tillmann, C. (1996). Hmm-based word alignment in statistical translation. In *Proceedings of the 16th conference on Computational linguistics-Volume 2*, pages 836–841. Association for Computational Linguistics.

Young, T., Hazarika, D., Poria, S., and Cambria, E. (2018). Recent trends in deep learning based natural language processing [review article]. *IEEE Computational Intelligence Magazine*, 13:55–75.

Appendix

IBM Model 1

Expectation Step:

$$p(a|\mathbf{f},\mathbf{e}) = \frac{p(\mathbf{f},a|\mathbf{e})}{p(\mathbf{f}|\mathbf{e})}$$

$$p(\mathbf{f}|\mathbf{e}) = \frac{\epsilon}{(l+1)^m}\prod_{j=1}^{m}\sum_{i=0}^{l} t\left(f_j|e_i\right)$$

Combining our 2 equations we have the probability of an alignment:

$$p(a|\mathbf{f},\mathbf{e}) = \prod_{j=1}^{m}\frac{t\left(f_j|e_{a(j)}\right)}{\sum_{i=0}^{l} t\left(f_j|e_i\right)}$$

Now we have to collect the counts: To compute the number of times that the single word f, from the target language, being translated from the single word e, in source language, given the source sentence $\mathbf{e}$ and target sentence $\mathbf{f}$ we sum over all possible alignments multiply the probability of each alignment given that sentence times the number of times the target word f which aligned with source word e in that alignment.

Maximization Step:

$$c(f|e;\mathbf{f},\mathbf{e}) = \frac{t(f|e)}{\sum_{i=0}^{l} t\left(f|e_i\right)}\sum_{j=1}^{l}\delta\left(f,f_j\right)\sum_{i=0}^{m}\delta\left(e,e_i\right)$$

After collecting these counts we can get the translation probability of target word f given the source sentence $\mathbf{f}$ and target sentence $\mathbf{e}$ by just summing over all the sentences in the corpus:

$$t(f|e;\mathbf{f},\mathbf{e}) = \frac{\sum_{(\mathbf{f},\mathbf{e})} c(f|e;\mathbf{f},\mathbf{e}))}{\sum_{e}\sum_{(\mathbf{f},\mathbf{e})} c(f|e;\mathbf{f},\mathbf{e}))}$$

IBM Model 2

Expectation Step:

$$p(\mathbf{f}|\mathbf{e}) = \epsilon\prod_{j=1}^{m}\sum_{i=0}^{l} t\left(f_j|e_i\right)a(i|j,m,l)$$

Now we get:

$$p(a|\mathbf{f},\mathbf{e}) = \frac{p(\mathbf{f},a|\mathbf{e})}{p(\mathbf{f}|\mathbf{e})} = \prod_{j=1}^{m}\frac{t\left(f_j|e_{a(j)}\right)a\left(a_j|j,m,l\right)}{\sum_{i=0}^{l} t\left(f_j|e_i\right)a(i|j,m,l)}$$

Maximization Step:

Now we collect the counts again as mentioned for the IBM1 model, as well as well as the jump counts.

$$t(f|e) = \frac{c(e,f)}{c(e)}$$

$$a(i|j,l,m) = \frac{c(i|j,l,m)}{c(j,l,m)}$$