Bachelor thesis
Computer Science

Radboud University

Lost in Translation: Automatic Learning of Statistical Models for Language Translation

Author:
Sanne Boumans
3031926

First supervisor/assessor:
Dr. A. Martins Da Silva
alexandra@cs.ru.nl

[Second supervisor:]
Dr. H.H. Hansen
h.h.hansen@cs.ru.nl

Second assessor:
Dr. T. Claassen
t.Claassen@science.ru.nl

April 7, 2015

Abstract

In this thesis we investigate which algorithms within a Statistical Machine Translation model do or don't contribute to a better text translation. A specialized SMT model named GIATI was compared to its more basic counterpart and as expected GIATI performed much better, in most cases doubling the number of correctly translated words. A number of possible explanation for this difference in result were found. Some of which include N-gram values, word alignment model choice, and smoothing algorithm choice. However the biggest difference and thus most likely explanation was the use of different word order models.

Contents

Chapter 1

Introduction

Machine Translation is the concept of using computers for the translation of texts or audio fragments[4, 8]. In an ideal world computers would be perfect at this and would be able to make correct translations 100% of the time. In turn translation by humans would become unnecessary and many hours of human labor would be saved.

Unfortunately however we do not live in the ideal world and in this world people generally do better than computers when it comes to translation[8]. The reason why computers usually do worse than humans is because language can be very dependent on different factors[4, 8]. For example, there are languages in which two words are written in the exact same manner but whose meaning depend entirely on the way they are pronounced. Furthermore, in some languages there are words, such as the Dutch word *bank* that are written and pronounced the exact same way but still have different meaning. In these cases the meaning of a word is thus based on the content of the rest of the sentence. Another possible difference between languages is the way sentences are usually structured. While in some languages it is standard to use a subject-verb-object order in a sentence, other languages might make use of other orders.

As stated before these differences between languages makes it hard for computers to make correct translations on their own, a trend that has been observed before. However, in an attempt to solve this problem, different Machine Translation (translation by computers) methods have been developed [1, 5, 12, 13, 17, 14]. All of which have their own set of used algorithms and end results.

The exact reason why these end results differ for every Machine Translation (MT) method however is not very apparent in the above mentioned papers. The different MT method papers all describe the algorithms that were used within it and tell what improvements are made in comparison to more basic MT methods. However while comparison of implementation techniques is done, comparison of test results is not. It seems the topic of translation result comparison is not the main focus of papers that introduce new MT methods. Luckily, there are of course, papers that do explore this topic and make a study out of comparing a multitude of methods, yet unfortunately not every single method is covered in them. [8, 11, 13] This while method comparison gives greater insights in which factors in a MT method contribute to better/worse translation end results. Without knowing these exact factors, improving a MT method simply becomes a guessing game as to where adaptations should be made.

In this thesis I will therefore add to the topic of MT method comparison and will try to find the answer to the following question *Which algorithms within a MT method do/don't contribute to a better translation end result?*.

In order to do this, the following questions shall in turn be answered in consecutive order. 1) *Which end results does a basic MT method implementation give?* 2) *Which end results does a more specialized MT method give?* 3) *What are the differences between a basic and the specialized MT methods?*

In this thesis the specialized MT method that will be used is the GIATI method[5, 16]. And, seeing as this method is an improvement of more basic translation algorithms, these more basic translation algorithms shall form the basic MT method. This way the results of the GIATI and the basic MT method (question 1 & 2) can be compared and if a difference is found the improvements GIATI made to the basic algorithms (question 3) can be seen as factors that did/didn't improve these results. Thus the main research questions will too get an answer.

Chapter 2

Preliminaries

2.1 Statistical Machine Translation

Both the - in the introduction mentioned - GIATI model and its underlying basic method are examples of Statistical Machine Translation (SMT) models [5, 16, 3]. Compared to other models that use simple word-by-word translations SMT models main goal is to also take into account the results that each word translation provides. This way when translating a sentence with an SMT model one would not simply get a possible translation for said sentence but one would get the most likely translation[8, 4, 5, 17] .

However, in order to give this most likely translation every SMT model needs to take into account two main things, namely: **word order** and **correct word-by-word translation**[8, 4, **?**, 17, 11, 3]. Table 2.1 below that shows possible translations for the Dutch sentence *"Dat heb ik al gedaan."* clearly illustrates these two concepts. We see that when word order and correct word-by-word translation are both not taken into account the given possible translation makes no sense and certainly is not a correct English translation of the Dutch sentence. However, when only word order is taken into account, the translated English sentence is actually a correct one and is certainly understandable. Yet it is not a correct translation of the Dutch sentence because correct translation was not achieved. On the other hand, if only correct word-by-word translation is used, prefect translation is also not achieved. While all individual words are correctly translated the words in the sentence seem scrambled and in the wrong order, causing the sentence to be grammatically incorrect. As we see, it is only when both correct word order and correct word-by-word translation are achieved that we get a proper Dutch-English translation.

		Correct word order	
		No	Yes
Correct translation	No	*"Cat brown fish the eats."*	*"The brown cat eats fish"*
	Yes	*"That have I already done"*	*"I have already done that."*

Table 2.1: Possible translations of *"Dat heb ik al gedaan."*

s

Having these concepts in mind every SMT model consist of two corresponding main elements [8, 4, 11]. The first of which is the **Language Model** whose job it is to deal with word order. The **Translation Model** makes up the second main element and its focus is achieving correct word-by-word translation.

Further explanation on both the Language- and Translation Model and their inner workings will be done in sections 2.1.1 and 2.1.2. Followed by a section 2.3 which will focus on **decoders**, which are also an essential part of any given SMT model.

2.1.1 Language Model

As stated before, the Language Model's job is to deal with proper word order within a sentence. When given an English sentence *E* the Language Model should be able to calculate what its word order probability (or *P(E)*) is. The higher this word order probability *P(E)* is, the more likely it is that the word order in sentence *E* is a logical one. Thus if one has a correctly working Language Model one could expect the sentence *"I have already done that."* to have a higher word order probability than the sentence *"That have I already done."*
This way, when given a set of possible English translations E_1,*E,subscript2*,...,E_n for a foreign sentence *F* the Language Model can simply calculate the word order probability of each sentence E_i and then conclude that the E_i with the highest word order probability is the most likely translation of sentence *F*.

How exactly the Language Model calculates these *P(E)* probabilities can be seen as follows [8, 11, 3, 12].
Let sentence *E* be of length *l*, then $E = w_1, w_2, ..., w_l$, where each $\mathrm{w_i}$ is an individual word. Now, to compute *P(E)* we require the probability of each word w_i being in position *i* in the sentence. To do this we simply need to find what the probability is of w_i following the words that came before it in the sentence (w_1^{i-1}).
This $P(w_i| \ w_1^{i-1})$ for every $w_i \in E$ can then be used to calculate *P(E)* by

using the chain rule of probability, given the following formula:

$$\begin{aligned} P(E) &= P(w_1)P(w_2 \mid w_1)P(w_3 \mid w_1^2)...P(w_{\mathrm{l}} \mid w_1^{l-1}) \\ &= \prod_{i=1}^{l} P(w_{\mathrm{i}} \mid w_1^{i-1}) \end{aligned} \tag{2.1}$$

To make this formula more concrete consider the example below where a number of *P(*w_i*|* w_1^{i-1}*)* probabilities within the Language Model are shown.

P(w_i*|* w_1^{i-1}*)* probabilities in the language model

$\boldsymbol{P(I)} = 0.8$

$\boldsymbol{P(have \mid I)} = 0.6$

$\boldsymbol{P(already \mid I\ have)} = 0.3$

$\boldsymbol{P(done \mid I\ have\ already)} = 0.1$

$\boldsymbol{P(that \mid I\ have\ already\ done)} = 0.1$

Using these probabilities the Language Model can now calculate the word order probability of the sentence *"I have already done that."* as follows:

$$\begin{aligned} P(I\,have\,already\,done\,that) &= P(I)P(have \mid I)P(already \mid I\,have)... \\ &= 0.8 * 0.6 * 0.3 * 0.1 * 0.1 \end{aligned}$$

Of course, before a machine can fully translate a text said machine must first know these above-mentioned probabilities*P(*w_i*|* w_1^{i-1}*)*. Here the Language Model's job comes into play because what the model does is read in some (large) English text and from that text try to derive the probabilities of the word orders. How the Language Model does this can vary greatly since there are multiple algorithms that deal with this problem. Later on, in section 3, we will see one basic Language Model algorithm fully explained.

2.1.2 Translation Model

The Translation Model is meant to deal with finding the best word-by-word translation for a foreign language sentence *F*. In order to do this it needs to be able to calculate *P(E | F)*, which is the probability of a sentence *F* translating into an English sentence *E*. Then when given a set of English sentences E_1,E_2,...,E_n the Translation Model can calculate this *P(*E_i *| F)* for every E_i and conclude which of these English sentences is most likely the correct word-by-word translation of sentence F [brown,collins,ma,berring,koehn book].

To calculate these $P(E \mid F)$ probabilities so called **T-tables** are used [8, 11, 3]. In this T-table set-up each foreign language word f has its own table of English words $e_1,\ldots,e_n$ in which each e_i is coupled with the probability of f translating into e_i ($P(f \mid e_i)$). The following tables are examples of possible T-tables:

Ik		heb		gedaan	
I	0.7	have	0.4	done	0.8
me	0.15	had	0.4	did	0.16
we	0.075	got	0.1	do	0.02
us	0.05	posses	0.06	household	0.015
he	0.025	petty	0.04	shell	0.005

Then when we have a foreign language sentence $F = f_1,f_2,\ldots,f_n$ of n words and an English sentence $E = e_1,e_2,\ldots,e_n$ (also of n words, these T-tables - in which every $P(f_i \mid e_i)$ can be found - can help calculate $P(E \mid F)$ with the following formula[11, 6, 3, **?**]:

$$P(E \mid F) \prod_{i=1}^{n} P(f_{\mathrm{i}} \mid e_{\mathrm{i}}) \tag{2.2}$$

For example, by using the above T-tables, *P(I have done | Ik heb gedaan)* would be: 0.7*0.4*0.8 = 0.224.

Once again, when starting translation, a machine does not know any of these T-table probabilities, luckily the Translation Model takes care of this. It does this by first reading in an English and a Foreign language text which are identical to one another (i.e. translations of the same text) in every aspect other than the language they are in (this is called a **parallel corpus**)[10]. Secondly, by reading in these texts sentence by sentence and word by word, the Translation Model then tries to estimate the probability of each Foreign language word f_i translating into each English word e_j. For this process too there are multiple algorithms available.

2.1.3 Decoder

When we have a language model that can calculate $P(E)$ and a translation model that can calculate $P(E \mid F)$ for every possible English sentence E and foreign language sentence F there is still one big SMT piece missing. This piece is Decoder.

The Decoder's job is to, when given a sentence *F*, find an English translation that has the highest probability of having both a correct word order and a correct word-by-word translation[7, 19, 18, 9]. Thus when we have a set of English sentences E_1,E_2,...,E_n the best English translation $\hat{E}$ of F is the one with the highest $P(E_i)P(E_i \mid F)$. This gives the following decoder formula:

$$\hat{E} = argmax_E\ P(E|F)P(E) \tag{2.3}$$

That it is important that the Decoder uses a combination of *P(E)* and *P(E | F)* can be seen in the example below. Here we see some possible translations for the sentence *"Dat heb ik al gedaan."*together with the probabilities they got from the language model and the translation model and their product. It is clear that even though a sentence has the highest *P(E)*, and thus has the most logical order, is not always the best translation. The same goes for a sentence that has only a high probability on correct word-by-word translation. Therefore a combination is needed.

	P(E)	***P(E | F)***	***P(E)P(E | F)***
"The brown cat eats fish"	0.8	0.1	0.08
"That have I already done."	0.3	0.9	0.27
"I have already done that."	0.7	0.9	0.63

Just as with the language and the Translation Model there are multiple algorithms that can be used for decoding. Most of these algorithms are adapted versions of some widely known algorithms (**A***,**Beam-search**,etc.), this is because decoding is essentially a search problem. When given a sentence in some foreign language it is the decoder's task to, for every foreign word, find its most likely translation and also to find the most likely word-order (as seen in the Language Model). We will see the most simplistic, greedy Decoder in the section on basic methods for SMT. For more advanced decoders we refer to other papers.

Chapter 3

Basic Methods for SMT

3.1 Implementation

In this chapter we will look at the implementation of a basic methods within the SMT model. Which will consists of a trainings phase, a run phase, and an evaluation phase. All of which are described in detail in various papers and literature [5, 3, 13, **?**, 12, 11, 8].

To go through these three separate phases we shall require a Dutch-English parallel corpus obtained from the Euro Parliament [koehn]. Which consists of four documents:

1. An English trainings text
2. A Dutch trainings text
3. An English test text
4. A Dutch test text

Of which text pairs 1 & 2, and 3 & 4 are the same texts only in different languages. Furthermore, this corpus has a **closed vocabulary**, meaning that whatever words will occur in the test set will also have been in the trainings texts and thus will be learned during training.
Finally, as is crucial in SMT, all texts that were used in the implementation of were tokenized, transformed into lowercase, and empty lines were removed. This was done by the use of an SMT system named **Moses**[2] and self-implemented code.

With this parallel corpus the trainings phase can start. In this phase both the language- and Translation Model will be trained to learn the word order and word-by-word translation probabilities (the T-tables) they need in order for the decoder to use them.

After this training is done the SMT model further consists of running a Dutch test text through the decoder and having the decoder find the best English translation for every sentence F in that test text. These best translations will then be written in a separate translation file, which will thus be the found English translation of the Dutch test text.

Finally the acquired translation file will be coupled with the English test text and go into the evaluation phase. Since the English test text is – save for its language - the same as the Dutch test text it can be used as a measure for correctness of the translation file. How this works will be more clear in section 3.2.

The phases of the implemented SMT method and the inputs/outputs they have can be seen in figure 3.1. and shall be explained in more detail this chapter. While previously using F to indicate a foreign language sentence we will now continue to use D instead since our implementation uses the Dutch language. Furthermore d and e shall represent individual Dutch and English words.

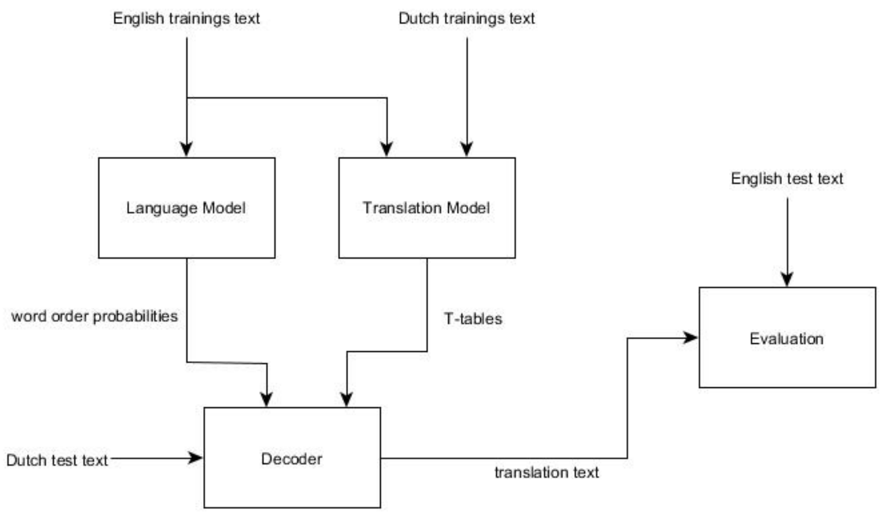

Figure 3.1: The steps in a basic SMT method.

3.1.1 Language Model

As stated before, the goal of the Language Model training phase is to learn the word order possibilities by reading in the English training text and performing some action on this. What this action pertains shall be explained shortly but first an important adaptation of equation (2.1) must be made. This adaptation comes in the form of **N-grams**.

When dealing with long and complicated sentences the chances of one of those sentences being in an English text become very low. As a consequence equation (2.1) becomes highly unreliable since word order probabilities can't be calculated properly since there are near none sentences to derive this probability from. To solve this problem SMT models usually use N-grams. These N-grams don't calculate the word order probability based on all the preceding words but rather based on the N-1 words before that. For example the probability *P(that* | *I have already)* changes into the probability *P(that* | *I have)* in a 3-gram situation. Using these N-grams the formula to calculate *P(E)* thus becomes:

$$P(E) = \prod_{i=1}^{l} P(w_{\mathrm{i}} \mid w_{i-N+1}^{i-1}) \tag{3.1}$$

However, all implementations below are based on Translation Models that work with bigrams (2-grams). Therefore we can now forget about any other N-grams as all further formulas will assume this set up. Important to note though is that all basic implementations could relatively easily be adjusted to accommodate other N-gram forms. Continuing with bigrams we can now focus on the real training phase of the Language Model.

Basic Language Models estimate *P(E)* by using a **Maximum Likelihood Estimation** (MLE). This MLE method works by keeping a count *C(*w_1*,*w_2*)* for every bigram and a count *C(*w_1*)* for every first word in the bigram. It then estimates the word order probability of the bigram by normalizing the bigram count through division ny *C(*w_1*)*. Formally expressed, the word order probability of the bigram (w_2 | w_1) is given by the following formula:

More concretely, we consider the example below where we see a short two sentence example English training text and some of the word order bigram probabilities it derives from them.

The brown cat
The brown bear
The black dog

P(brown | ***the)*** = *C(the, brown)* / *C(the)* = 2/3
P(cat | ***brown)*** = 1/2 ***P(dog*** | ***brown)*** = -/2 = - (since the word order *brown dog* does not exist)

Training phase

With the use of formula 3.1 and using the English part of the parallel corpus we can now train a machine to estimate the probabilities of each English word bigram simply by doing the following things:

1. Prepend every sentence in the English corpus with start symbol <s> and, append it with end symbol </s>. This way we can also estimate the probability of a word being in the first or last position in a sentence.
2. For every word [w_i] in every sentence keep count *count_[w_i]* of that word.
3. For every bigram [w_i,w_{i+1}] in every sentence keep count *count_[w_i,w_{i+1}]* of that bigram.

These simple steps give the following Language Model pseudocode:

Algorithm 3.1: Language Model Training.

```
input: set of sentences E, table with unigrams and their counts (
    initialized with 0), table with bigrams and their counts (
    initialized with 0)
output: a list of bigram and 1-gram counts for all bigrams and
    individual words
begin

  forall e in E
    e = ''<s>'' + e
    e = e + ''</s>''
    split e into array W of words

    forall  i = 0 through W.length -1
      count_W[i] += 1
      count_[W[i],W[i+1]] += 1
    end

  end

end
```

As we can see, this code does not return any direct output, but simply changes the unigram (1-gram) and bigram counts in their respective tables.

Running phase

Although the above mentioned method works well for finding bigram probabilities it has one major flaw. Think of what would happen if after training of the Language Model, with the above mentioned three sentences, the machine was asked to find *P(E)* of the sentence *The brown dog.* While this is a completely correct English sentence, the machine would still return a probability of 0.0 (or an error), simply because the bigram (*brown, dog*) does not occur in the learned table. This too can be seen in the example above.

To solve this problem of possibly stating that a correct English sentence is in fact incorrect the Language Model uses a technique called **smoothing**. There are multiple smoothing algorithms but in this thesis we shall use the most simplistic one, namely **Laplace smoothing**. The basic mechanism behind Laplace smoothing is that the algorithm always increases the bigram count of any bigram with +1 during the calculation of *P(E)*. This way even bigrams that do not occur within a text still have a bigram count of 1. This would mean just a simple adjustment to equation (3.1) but since all bigramscounts have been incremented the denominator in the equation that normalizes the probabilities should also be changed. In Laplace smoothing this is done by simply adding the total number of individual words in the vocabulary V to the unigram count. Thus the new equation for a bigram probability with Laplace smoothing becomes:

$$P(w_{\text{i}} \mid w_{\text{i+1}}) = \frac{C(w_{\text{i}}, w_{\text{i+1}}) + 1}{C(w_{\text{i}}) + V} \tag{3.2}$$

Using this new formula with smoothing the example previously used changes into the following:

The brown cat
The brown bear
The black dog

P(brown* | *the) = *C(the, brown)* + 1 / *C(the)* + 6= 3/9
P(cat* | *brown) = 2/8
P(dog* | *brown) = 1/8

With smoothing and equation (2.1) taken into account we can now calculate probability *P(E)* which shall be used by the decoder in the running phase. This calculation is done with the following pseudocde:

Algorithm 3.2: Language Model Testing.

```
input:  a sentence e and the lists with bigram and unigram counts
     acquired from the training phase
output: the probability of e
begin

   P(e) = 1.0
   e = ''<s>'' + e
   e = e + ''</s>''
   split e into array W of words

    forall i = 0 through W.length-1
```

```
            wordcount = count_W[i] + vocabulary size
            bigramcount = count_[W[i]i,W[i+1]] + 1

            P(e) *= bigramcount/wordcount

        end

        return P(e)

    end
```

3.1.2 Translation Model

When training the Translation Model **word alignments** are implicitly learned. There are different definitions for these word alignments depending on which alignment model is used in the SMT model. However in this implementation we will use a simplified version of **IBM Model 1** which means that, in our case, word alignments are pairs of one Dutch word and one English word that have occurred in the same sentence in their respective language's trainings texts and are thus possible translations of one another[11, 4, **?**]. Furthermore, since we are using a simplified version of the model we will use we will assume that every Dutch word is aligned to *at least* one English word. More specialized models (IBM Model 2-5) can deal with non-aligned words or with alignments of multiple words instead of just one on one alignments.

To learn these IBM Model 1 word alignments the Translation Model is trained and this usually happens with the use of the **Expectation Maximization Algorithm**, or **EM algorithm**.The idea behind this EM algorithm is that it does the the following:

1. Initialize the values in the Translation Model.
2. Let the Model run through the data (expectation step).
3. Let the Model learn new values through the data (maximization step).
4. Repeat step 2 and 3 until the model has converged.

We will see these steps more concretely explained in the following sections.

Initialization

Before the Translation Model can start with its training and thus it's learning of the translation probability between some foreign word and some English word initialization is required. In this initialization phase the program iterates through the Dutch and English trainings texts one time and constructs a total of three tables named *Translation*, *English*, and *Dutch*. These tables can contain the following data:

Translation The table that has, for every possible Dutch/English word alignment (e | d) its probability *P(e | d)* and its count *count_(e | d)*. In this table the value of *P(e | d)* is always set to 1 / vocabulary size of the Dutch trainings text during initialization.

Dutch The table that has the count (*count_(d)*) of every single Dutch word.

English The table that has the count (*count_(e)*) of every single English word.

Before we continue on to the pseudo code for this EM algorithm, it is important to note that during initialization not every *(e | d)* word combination exists. This is because the algorithm only sees two words as possible translations for one another when they occurred in the same sentence in each parallel corpus. For example, when given the following sentence combinations as trainings text:

De kat	*The cat*
De beer	*The bear*

The word alignments *(the | de)*, *(cat | de)*, *(the | kat)*,*(the | beer)*, etc. are all quite possible. However, the translation *(bear | kat)* is not possible, simply because these words occur in different sentences in the parallel corpus. Therefor this translation will not occur in table *Translation*.

These results are acquired by the following psuedocode:

Algorithm 3.3: Translation Model Initialization.

```
input:  set of sentence pairs (D,E) acquired from Dutch and
     English trainings texts
output: the Translation , Dutch and English tables with all
     possible alignments and individual words and their initialized
      probabilities and counts.
begin

   Translation = new table <String f, String e, Double P(e|d),
       Double count (e|d)>
   Dutch = new table <String d, Double count_(d)>
   English = new table <String e, Double  count_(e)>

   forall sentence pairs (D,E)

      forall d in D
        if not d already in Foreign
           Foreign.add(d,0.0)
```

```

            forall e in E
               if not e already in English
                  English.add(e,0.0)

               if not (e | d) already in Translation
                  Translation.add(e,d, 1/vocabulary size(D), 0.0)

         end
      end

   end
```

Training phase

Steps 2, 3, and 4 of the EM algorithm make up the actual training for the Translation Model.

However, before we go into more detail on step 2 and 3 we will first briefly discuss step 4. This step namely involves **convergence**. When testing if the model has converged the model runs through the trainings files again an calculates, with the learned probability values, the translation probability for every sentence pair it finds. It then uses a different formula to calculate the **perplexity** of the model as a whole. Convergence is then achieved if this preplexity has reached a global minimum and thus doesn't drop any lower on all the next training iterations.
In this thesis however, we will not make use of convergence. Calculating perplexity is time consuming and complex work and it may take over a thousand iterations before a model is fully converged. The code used for this SMT implementation will have a total of five trainings cycles, something that gives relatively good results since most of the convergence happens in the first few cycles.

We can see in the pseudocode below that during the execution of these steps the counts for all foreign words, English words, and translation pairs get set to 0 only to be incremented during the times the translation model iterates through the parallel corpus. The model does this by, for every sentence, first changing the *count_(d)* for all Dutch words. Following this it uses these counts to calculate both *count_(e | d)* and *count_(e)*. Then, finally, after the model is done with running through the complete corpus every translation probability *P(d | e)* is calculated by dividing *count_(e | d)* by *count_(e)*.

Algorithm 3.4: Translation Model Training.

```
input: set of sentence pairs (D,E) aquired from the trainings
     files, table Translations, English, and Foreign, from
     initialization fase.
```

```
output:  The Translation table with updated probability for every
     Dutch/English word alignment
begin

   forall i = 0 through 5
      forall (e|d) in P
        count_(e|d) = 0
      end
      forall e in E
        count_(e) = 0
      end

      forall sentence pairs (D,E)

        forall d in D
          count_(d) = 0
          for all e in E
             count_(d) += P(e|d)
          end
        end

        forall d in D
          for all e in E
            count_(e|d) += P(e|d) / count d
            count_(e) += P(e|d) / count d
          end
        end

     end

     forall (e|d)
       p(e|d) = count_(e|d) / count_(e)
     end

   end

end
```

To see the precise workings of the EM algorithm in a concrete example see Appendix A.

Running phase

Calculating *P(E | F)* can now simply be done by transforming equation (2.2) into code.

Algorithm 3.5: Translation Model Testing.

```
input:  a sentence pair (F,E), Translations table
output: the probability of P(E|F)
begin

   prob = 1.0

   forall f in F

     forall e in E
       if (e|d) in Translations
          prob *= P(e|d)
       else
```

```
            prob *= 0.0;
        end

    end

end
```

For an example of this simply recall the example in section 2.1.2.

3.1.3 Decoder

The decoder is the final, and perhaps the most complex, part used in SMT. The most basic decoder that would, when given a foreign language sentence *F*, try every possible English word *e* for every *f* in *F* is not a decoder that we would want to use. Namely, without any use of any sort of heuristics, the size of the entire search space would grow roughly exponential according to the input size of the sentence. This complexity is clearly unwanted, and the time it would take to, for example, translate a text with 1 million sentences, would be simply unacceptable.
The use of heuristics in decoding is thus unavoidable and, since we are implementing the very basics of SMT in this thesis, we will use many of them.

The decoder that was tested in this thesis was a very simplistic one in the sense that, for every word that needed translating, it only used the most likely translation as found by the Translation Model. Other possible word translations were thus not taken into account. Also, when we look at the pseudocode below, we see that the word orders found by the Language Model are only used in their most greedy form.

First (line 7) the sentence that needs to be translated into English is split into individual words. Following that (line 11/15) the decoder runs through those words $w_0,\dots,w_n$ trying to find the word w_i that is most likely to be at the beginning of the sentence (thus the w_i fro which $P(\text{"<s>"},w_i)$ is highest. When the first translated word of the sentence gets added to the translated sentence so far it gets removed from the original sentence. Finally after that, in lines 24 through 32, the model does the same only instead of using the probability of a w_i following "<s>" it searches for the word that has the highest probability of following the word it found in the previous iteration and is thus currently the last word of the translation until now. When there are no more words to translate the decoder has found a translation for the sentence.

Algorithm 3.6: Decoder

```
1  input:  Dutch sentence d
2  output: translated English sentence e
3  begin
```

```

    String e = ''''

    split d into list W of words
    best probability = 0.0
    best word = ''''

    forall w in W
        if  P(''<s>'',w) > best probability
            best probability = P(''<s>'',w)
            best word = w
     end

    e.add(best word)
   W.remove(best word)

    while W.size > 0
      best word = '''
      best probability = 0.0

       for all w in W
         if P(last word of e, w) > best probability
            best probability = P(last word of e, w)
            best word = w
       end

       e.add(best word)
      W.reove(best word)

     end

    return e

end
```

As a more clear example, when having acquired the probabilities shown below from both the Language Model and the Translation Model this decoder would, when translating the sentence *"De bruine kat."*, use the words *the, brown, dog* (which are the most likely word translations) but would reorder them into *"The dog brown."*. This because the word *the* was most likely to be the first in the sentence, with *dog* being most likely to follow after that, etc.

word orders

The | *<s>)* = 0.8
brown | *<s>)* = 0.6
dog | *<s>)* = 0.3
brown | *the)* = 0.7
dog | *the)* = 0.8

translation probabilities

P(the | *de)* = 0.9
P(a | *de)* = 0.5

$P(brown \mid bruine) = 0.7$
$P(cat \mid bruine) = 0.2$
$P(dog \mid kat) = 0.6$
$P(cat \mid kat) = 0.5$

We shall see how these decoders, in combination with the implemented Language and Translation Model, preformed in the evaluation section.

3.2 Evaluation

One possible method for the evaluation of a SMT model is using human evaluation. We could simply take the text that needed translation and the translated text and check word for word whether it was translated correctly or not. However this scoring is of course strongly dependant on the persons vocabulary in both languages and also possibly on his or hers interpretation of certain words and sentences. Another problem could arise when dealing with the translation of very long texts. Using human evaluation for those would be time consuming, and a very monotone task.

Luckily there are some methods that can also evaluate the work of an STM model. And though there is no golden standard for which one of these works best, one of the most commonly used methods seems to be **BLEU**[15, 5, 8]. Therefore, an implementation of BLEU called iBLEU was used for our implementation.
BLUE is an evaluation method that requires two files as input. Namely; the file that was translated by the implemented SMT method, and a file that has the correct translation for the testing file that was used. Given these two files BLEU then, like the human translator would, checks word for ford whether the correct translation was used. However, BLEU also has the added feature that it also evaluated the correct translations of N-grams. Thus, if we had sentences *The cat* and *The dog* and they would be compared by BLEU the number of correct unigrams would be 1 (*the* and *cat*) while the number of correct bigrams would be 0, since there were no correct translations of two words in a row.
In BLEU the scores range from 0 to 1, in which 0 is the worst score (0% of the words correctly translated) and 1 (100% correctly translated) is the best score.

3.2.1 BLEU Results

When translating the test file with the basic SMT method that was implemented in this thesis the following results came forward:

Type	**1-gram**	**2-gram**	**3-gram**	**4-gram**
Cumulative score	0.3287	0.1394	0.0589	0.0244

This table indicates that when it comes to the translation of individual words our implementation was correct in 32% of the cases. However when looking at the number of times we correctly translated multiple words in a row these percentages decreased quickly, eventually correctly translating four words in a row only 2% of the time.

Chapter 4

GIATI method for MT

In this section we will look at a different, more advanced method for machine translation. As mentioned in the introduction, this will be the **Grammatical Inference and Alignments for Transducer Inference** (or GIATI) method that was developed by Casacuberta and Vidal [5, 16]. The basic algorithms and techniques within this model will be briefly explained before being compared to their counterparts that we have seen in the more basic SMT model in chapter 3l. Following that we will look at some of the results these researches found when evaluating their model.

4.1 Implementation

Casacuberta and Vidal explain in their paper that during training, when faced with a sentence pair (F,E) their GIATI method goes through three main steps:

1. A **labeling process** that transforms the training pair (F,E) into some sort of String.
2. A **GI algorithm** for the inference of a Finite State automaton from the strings obtained in step 1.
3. An **inverse labeling process** in which the string found in the labeling process is transformed back to word pairs.

These steps can be clearly seen in the figure below, which was provided by Casacuberta and Vidal in their paper.
Here we see that starting with a sample A of trainings pairs (F,E), we go through a labeling process in order to obtain a sample of training strings. Then these training strings go through a grammatical inference (GI) algorithm in order to learn a Finite State Automaton from them.

$$\begin{tikzcd}
\begin{array}{c} A \subset \Sigma^\star \times \Delta^\star \\ \text{Sample of training pairs} \end{array} \arrow[r, "\text{Labeling} - \mathcal{L}(\cdot)"] & \begin{array}{c} S \subset \Gamma^\star \\ \text{Sample of training strings} \end{array} \arrow[d, "\text{algorithm}", "\text{GI}"'] \\
\begin{array}{c} \mathcal{T}\colon A \subset T(\mathcal{T}) \\ \text{A finite-state transducer} \end{array} & \begin{array}{c} \mathcal{A}\colon S \subset L(\mathcal{A}) \\ \text{A finite-state automaton} \end{array} \arrow[l, "\text{Inverse labeling} - \Lambda(\cdot)"']
\end{tikzcd}$$

Figure 4.1: GIATI method steps

In the following sections we will explore how these individual steps compare to the Language Model, the Translation Model, and the Decoder in a basic SMT model.

4.1.1 Translation Model

The reason why, unlike in the previous chapters, we start with explaining the Translation Model is that in the GIATI model the Translation Model is the starting point.

While in the basic SMT method we showed that before the machine learned which foreign sentence words were aligned with which English words, the GIATI method assumes these alignments are already known before translation training occurs. The training of these alignments is therefore not part of the GIATI model. Thus even before the GIATI model is used it is given a set of sentence pairs (F,E) which would look something like this: (*Dat heb ik al gedaan*, *I(3) have(2) already(4) done(5) that(1)*). In which the number between the parenthesis indicates the position of the aligned foreign word in the foreign sentence. If we were to make similar alignments in between parenthesis with our basic SMT implementation each foreign word f would be aligned with its most likely translation which is acquired from training the translation model.

The fact that GIATI also works with these most likely translations shows that, even though the Translation Model is not part of the GIATI model, it has the same basic idea as the one we used in the previous chapter.

The only real difference is that the Translation Model that was used in the GIATI paper made use of alignments based on **IBM Model 5**. Which, compared to Model 1, is better adapted to handle the possibility of one word translating into multiple words and preventing word pairs that appear together often to translate to the same word.[11]. However since the alignments used in the GIATI method were individual word-to-word alignments the benefits of IBM Model 5 compared to lower order models are not really neccesary.

4.1.2 Language Model

Of the three steps in the GIATI model step 1 and 2 are most similar to the Language Model in our basic SMT Model. The **labeling process** first finds word orders that occur within sentences and then the **GI algorithm** calculates the probabilities of these phrases having these word orders and translations. We will now see how this process works.

The Labeling Process

As we have seen before, word order is an important part of proper machine translation. It is thus no surprise that GIATI also has some way of handling the fact that words in different language sentences might not always be in the same position in said sentences. Take for example once again the sentences *Ik heb dat al gedaan.* and *I(1) have(2) already(4) done(5) that(3).(6).* Here we can clearly see, since the alignment isn't sequentially ordered, that the word order is definitely not the same. GIATI solves this by turning sentences E and F of training pair (F,E) into one string z continuously using one of the following rules until all the foreign words are aligned:

Each foreign word f_i in position i is paired with an English word at the same position e_i. In turn each English word e_i has an alignment position j (between its parenthesis).

If $i < j$ and no words are saved
pair f_i with λ and save e_i

If $i < j$ and words are saved
save e_i

If $i \geq j$ and no words are saved
pair f_i with e_i

If $i \geq j$ and words are saved
pair f_i with all previously saved words (starting from the first one saved) + e_i. After using all saved words they are removed from the saved list.

To make this method more explicit we will look at what will happen to the previously mentioned sentences *Ik heb dat al gedaan.* and *I(1) have(2) already(4) done(5) that(3).(6).* We will look at each position i of the foreign word f, English word e_i, and each aligned position j.

i	f_i	e_j	j	pair
1	Ik	I	1	i=j so: (Ik,I)
2	heb	have	2	i=j so (heb,have)
3	dat	already	4	i<j so (dat,λ) *already* gets saved
4	al	done	5	i<j and there is already something saved so *done* gets saved
5	gedaan	that	3	i>j so all saved words get used (gedaan,already done that)
6	.	.	6	i=j so (. , .)

In the end this results in z = (Ik,I)(heb,have),(dat,λ)(gedaan,already done that)(. , .).

Thus, in the labeling process of GIATI, the correct word order of a sentence is found just like the Language Model in chapter 3 did. There are some differences however, the first one being that in GIATI the word order probabilities are not derived from training with an English text but are derived from alignment information given by the Translation Model. Another big difference is that while in the basic implementation each foreign word f got paired with exactly one English word e here f can be paired with multiple English words. This practice is called **phrase translation** [8, 11, 3, 13] and gives rise to the possibility of one word translating into none or multiple words.

The GI algorithm

In the second step of the GIATI model the strings obtained from step 1 are used to make a Finite State automaton. This is done, for every string z, by makin a state transition for every (f,e) pair in z. Furthermore if there are two pairs (f_i, e_i) and (f_j, e_j) with $f_i = f_j$ and $e_i = e_j$ then their state transitions always go towards the same state. A final state is made once sentence z has ended.

For example we obtained the following strings from step 1:

(Ik,I)(heb,have),(dat,λ)(gedaan,already done that)
(Ik,I)(heb,have),(dat,λ)(gedaan,done that)
(Jij,You)(hebt,have)(dat,λ)(gedaan,done that)
(Ik,I)(heb,have)(ijs,icecream)

the following Finite State Automaton would be created:

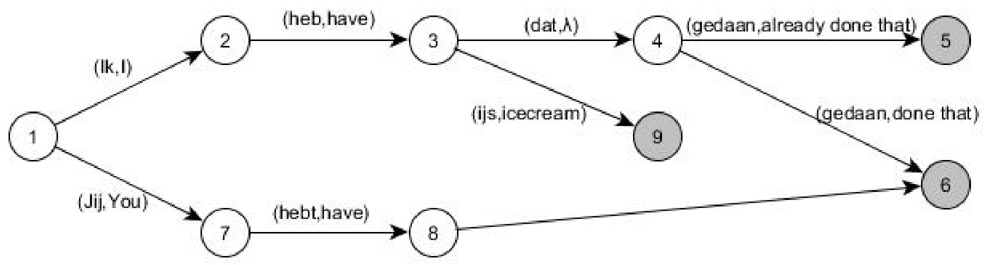

Figure 4.2: a Finite State Automaton

After this Finite State Automaton is drawn in GIATI the probabilities of each transition are calculated the same was as the n-gram probabilities were calculated in the basic SMT model. For example in the above seen situation the probability of *gedaan* translating into *already done that* is 1/2. The word order probability is thus calculated using the same method, only, in GIATI it is calculated for phrases instead of individual words. The only real difference in this probability calculation is that, instead of Laplace smoothing, the GIATI model made use of **back-off** smoothing which is an more advanced smoothing technique and give more accurate results[8].

4.1.3 Decoder

In order to find a correct translation for a new sentence GIATI goes through one final step. Namely the **Inverse labeling process**.

4.1.4 The Inverse Labeling process

In the final Inverse Labeling Process all pairs in z from step 1 (and thus all transitions made in step 2) are transformed back into translation pairs as we know them after translation model training. However, since the GI algorithm before this step calculated the probabilities associated with each transition within its Finite State automaton, each translation pair will now have a probability assigned to it.

For example the above automaton will give the following translation pairs with each an example/ made up probability.

$P(Ik \mid I) = 0.8$
$P(Jij \mid You) = 0.9$
$P(hebt \mid have) = 0.8$
$P(heb \mid have) = 0.7$

P(dat | λ*)* = 0.6
P(ijs | *icecream)* = 0.4
P(gedaan | *already done that)* = 0.3
P(gedaan | *done that)* = 0.7

This way the decoder can, when translating a Dutch sentence, simply go through this Dutch sentence word by word and pick the most likely translation for it and append it to the translated English sentence.

It is obvious here that this decoder is much simpler than the one we used in the basic SMT model. This is the case because instead of needing to first find correct translations for each word and then, ideally, trying all possible word orders this word ordering is already done and can be found in the finite state automaton. This makes decoding much faster since word translation probabilities are implicitly combined in the word order probabilities.

4.2 Evaluation

Just like we did in the previous chapter, the developers of GIATI also used the BLEU method for the evaluation of their SMT implementation. They evaluated GIATI's performance on three different corpora and in three different language combinations. However, unlike our 1,2, 3, and 4-gram BLEU evaluations only the 1-gram variant was used to evaluate GIATI. Below we see the results.

	2-gram
Spanish English 1	0.86
Spanish-English 2	0.86
Italian-English 1	0.56
Italian-English 2	0.62
Spanish-German	0.74

Here we can clearly see that the GIATI model performed much better than our basic SMT implementation. While our individual word translation (BLEU's 1-gram score) was only correct 32% of the time, GIATI scored at least double that percentage in almost all translation tasks.

While in our SMT implementation we only used bigrams in the language model the developers of GIATI also tested their model with a wide range (N = 2-12) of N-grams.Evaluation of these different N-grams for the language model showed that the higher values of N gave better translation results than the lower ones. In some cases higher N-gram values resulted in a 10% increase of correctly translated words.

Chapter 5

Conclusions

After implementing a basic SMT method and comparing its end results with the more advanced GIATI method a few conclusions can be drawn in respect to the research questions asked in the introduction.

When it comes to sub questions 1 and 2 that were asked then a clear answer has been found. While the basic SMT method implementation correctly translated individual words 32 % of the time, the GIATI method did this 56% to 86% of the time, depending on which languages were used. The answer to sub question 3 can also be found when looking at chapter 4. The main differences between the basic SMT implementation and the GIAT method were: the use of different alignment models (model 1 vs. model 5), the difference in word ordering technique, the difference in smoothing algorithm, and the difference in allowing non-alignment and one-to-multiple-alignments for words.

However, as we also saw in chapter 4 the advantages of IBM Model 5 were not really used and while good smoothing gives more reliable results it, per definition, only helps with word orders that are on average very rare. The difference in word ordering technique was the biggest one found in this thesis and was the direct cause for acceptance of non-alignment and one-to-multiple-alignments.

As for an answer to the main research question of this thesis, we can therefore give a partial one. While the differences in algorithms between the two methods are all possible algorithms that contribute to a better translation results, we can't say with certainty if all the algorithm improvements in GIATI contributed to a better end result. However since the main difference was the way word order probabilities were calculated we dare conclude there is a definite correlation between having a good model for this and having good translation end results.

In order to find out exactly why this big difference in results due to different handling of word order was found further research is needed. A possible continuation of this thesis could be further research into word order models. While we have seen that word order could very well be an important part in correct machine translation we have not compared all different word order algorithms that are out there and it is thus possible that there are also factors that contribute to bad/good translation within these separate word order models. Using the same translation model and decoding method one could possibly vary the Language Models/word order models used and compare the results each of them gives.

Bibliography

[1] H. Alshawi, S. Bangalore, and S. Douglas. Learning dependency translation models as collections of finite state head transducers. *Computational Linguistics*, 26:1:45–260, 2000.

[2] S. Ananiadou, editor. *Moses: Open Source Toolkit for Statistical Machine Translation.* Association for Computer Linguistics, 2007.

[3] P.F. Brown and et al. A statistical approach to machine translation. *Computational Linguistics*, 16:2:79–285, 1990.

[4] J. Brunning. *Alignment Models and Algorithms for Statistical Machine Translation.* PhD thesis, Cambride University Engineering Department and Jesus College, 2010.

[5] F. Casacuberta and E. Vidal. Machine translation with inferred stochastic finite-state transducers. *Computational Linguistics*, 30:2:181–204, 2014.

[6] M. Dillinger. Implementing machine translation, 2004. `http://www.translationoptimization.com/papers/DillingerLommel_MT_BPG.pdf`.

[7] U. Germann and et al., editors. *Fast Decoding and Optimal Decoding for Machine Translation.* Toulouse, France, 2001.

[8] D.l Jurafsky and J. H. Marthin. *Speech and Language Processing: An Introduction to Natural Language Processing, Speech Recognition, and Computational Linguistics.* Prentice-Hall International, 2007.

[9] P. Koehn. Pharaoh: A beam search decoder for phrase-based statistical machine translation models. `http://homepages.inf.ed.ac.uk/pkoehn/publications/pharaoh-amta2004.pdf`, 2004.

[10] P. Koehn. Europarl: A parallel corpus for statistical machine translation. `http://www.statmt.org/europarl/i`, 2005.

[11] P. Koehn. *Statistical Machine Translation.* Cambridge University Pressl, 2009.

[12] M. Korenevsky, A. Bulusheva, and K. Levin, editors. *Unknown Words Modelling in Training ang Using Language Models for Russian LVCSR System.* Sppech Technology Center, Saint-Petersburg, Russia, 2011.

[13] J. Ma. Automata in natural language processing. Technical Report 0834, Laboratoire de Recherche er Developpement, l'Epita, France, 2008.

[14] S. Niessen and H. Ney. Statistical machine translation with scarce resources using mopho-syntactic information. *Computational Linguistics*, 30:2:205–225, 2014.

[15] K. Papineni, S. Roukos, T. Ward, and W. Zhu. Bleu: a method for automatic evaluation of machine translation. Technical Report RC22176 (W0109-022), IBM Research Division, 2001.

[16] D. Pico, J. Tomas, and F. Casacuberta. Giati: a general methodology for finite-state translation using alignments. `http://personales.upv.es/~jtomas/articulos/sspr04%28David%29.pdf`, 2004.

[17] C. Quirk, A. Menezes, and C. Cherry. Dependency tree translation: Syntactically informed phrasal smt. Technical report, Microsoft Research,Redmond, USA, 2004.

[18] M.l Turitzin. Statistical machine translation of french and german into english using ibm model 2 greedy decoding. `http://nlp.stanford.edu/courses/cs224n/2005/turitzin.pdf`.

[19] Y. Wang and A. Waibel. In *International Conference on Spoken Language Processing, ISCA, Sydney, Australia*, 1998.

Appendix A

Translation Model Training Example

Set of sentence pairs (D,E)

Het boek
The book

Bruin boek
Brown book

A.1 Initialization

A.1.1 Translation table

translation pair	P(e | d)	count_(e | d)
(the | het)	1/3	0
(the | boek)	1/3	0
(book | het)	1/3	0
(book | boek)	1/3	0
(brown | bruin)	1/3	0
(brown | boek)	1/3	0
(book | bruin)	1/3	0

A.1.2 Dutch table

het	0
boek	0
bruin	0

A.1.3 English table

the	0
book	0
brown	0

A.2 Iteration 1

A.2.1 Translation table

translation pair	**P(e | d)**	**count_(e | d)**
(the | het)	0.5	0.5
(the | boek)	0.25	0.25
(book | het)	0.4	0.5
(book | boek)	0.4	0.25+0.25
(brown | bruin)	0.66	0.5
(brown | boek)	0.33	0.25
(book | bruin)	0.4	0.5

A.2.2 Dutch table

het	1/3+1/3
boek	1/3 + 1/3 + 1/3 + 1/3
bruin	1/3 + 1/3

A.2.3 English table

the	0.5+0.5
book	0.25+0.25+0.5+ 0.25
brown	0.5+0.25

A.3 Iteration 2

A.3.1 Translation table

translation pair	**P(e | d)**
(the | het)	0.556
(the | boek)	0.181
(book | het)	0.0.444
(book | boek)	0.580
(brown | bruin)	0.625
(brown | boek)	0.242
(book | bruin)	0.375