Language and Linguistics Compass (2011): 1–21, 10.1111/j.1749-818x.2011.00274.x

Statistical Machine Translation: A Guide for Linguists and Translators

Mary Hearne and Andy Way*
School of Computing, Dublin City University

Abstract

This paper presents an overview of Statistical Machine Translation (SMT), which is currently the dominant approach in Machine Translation (MT) research. In Way and Hearne (2010), we describe how central linguists and translators are to the MT process, so that SMT developers and researchers may better understand how to include these groups in continuing to advance the state-of-the-art. If these constituencies are to make an impact in the field of MT, they need to know how their input is used by SMT systems. Accordingly, our objective in this paper is to present the basic principles underpinning SMT in a way that linguists and translators will find accessible and useful.

1. Introduction

Corpus-based approaches to Machine Translation (MT) are founded on the use of parallel corpora. That is, previously unseen texts are automatically translated using information gleaned from examples of past translations produced by humans. The two primary MT strategies that belong to this paradigm are Example-Based MT (EBMT) and Statistical MT (SMT). While EBMT systems translate using the notion of 'similarity' to previously seen sentences, SMT systems do not. In this paper, we focus exclusively on a presentation of the basic principles underpinning the SMT approach. For a description of how the ideas behind SMT developed and how they relate to other MT approaches, and for a discussion of the central role played by translators and translations in SMT, see Way and Hearne (2010).

Statistical MT employs two distinct and separate processes: *training* and *decoding*. The training phase involves extracting a statistical model of translation from a parallel corpus, and a statistical model of the target language from a (typically much larger) monolingual corpus (Brown et al. 1990, 1993). The translation model effectively comprises a bilingual dictionary where each possible translation for a given source word or phrase has a probability associated with it. However, the model does not resemble a conventional dictionary where plausible entries only are permitted; many of the entries represent translations that are unlikely but not impossible, and the associated probabilities reflect this. The language model comprises a database of target-language word sequences (usually ranging between 1 and 7 words in length), each of which is also associated with a probability. Additional information beyond the language and translation models can be extracted during training, such as models of relative sentence length, word reordering, relative importance of translation vs. language model, etc. These induced models are then used during decoding, the process which actually yields a translation. The decoding process essentially treats translation as a search problem: given the sentence to be translated, search over all possible translations permitted by the translation model, and all possible reorderings thereof, for

the one which is assigned the highest overall probability according to the translation and language models.

The idea of generating target sentences by translating words and phrases from the source sentence in a random order using a model containing many non-sensical translations may not seem plausible. In fact, the methods used are not intended (in our opinion, at least) to be either linguistically or cognitively plausible: see Way and Hearne (2010, section 2.4) for further discussion of this topic. SMT is, however, *probabilistically* plausible; rather than focusing on the best process to use to generate a single optimal translation for a source sentence, SMT focuses on generating many thousands of hypothetical translations for the input string, and then working out which one of those is most likely.

While there are many variations of SMT systems, they all rely on these core principles and techniques. Thus, SMT is not a generic term – the generic terms are corpus-based, data-driven or perhaps probabilistic MT – but rather denotes a specific approach and architecture. In this paper, we detail the main components that make up an SMT system, covering the core training, parameter optimisation and decoding strategies. In Section 2, we focus on the big picture, while in subsequent sections we zoom in on the finer details: training in Section 3, including tuning in Section 3.3, and decoding in Section 4. Section 5 provides a brief conclusion, together with some pointers to other accounts of SMT for the interested reader.

2. Overview

At a high level, SMT gives us a view of MT expressed in a single formula. From this vantage point, how translations are generated is irrelevant. The only salient issue is that we can determine how likely any proposed translation is given the input string, and that we can consequently determine the most probable (i.e. 'best', according to the system) translation from a set of proposed candidates.

That is, SMT is founded on a well-defined decision problem: which translation is the most likely? There are two formulae available to us to compute that score: the noisy-channel model in (1), and the log-linear model in (2). The noisy-channel model is the one traditionally used in the literature (Brown et al. 1990, 1993). While the log-linear model can be instantiated to express precisely the same computation as the noisy-channel one (as we explain later in this section), it is more flexible and has come into widespread use in recent years (Och and Ney 2002).

$$\text{Noisy-channel model: } \mathit{Translation} = \mathit{argmax}_T \; P(S|T) \cdot P(T) \tag{1}$$

$$\text{Log-linear model: } \mathit{Translation} = \mathit{argmax}_T \sum_{m=1}^{M} \lambda_m \cdot h_m(T, S) \tag{2}$$

Firstly, we note that the leftmost parts of these equations ($\mathit{Translation} = \mathit{argmax}_T \ldots$) are exactly the same and are to be interpreted as follows: assuming that we have T candidate translations, we will let the output *Translation* be the T with the maximum (*argmax*) score. Secondly, we note that the rightmost part of the equations then differ in expressing how each candidate translation should be scored.

The noisy-channel model comprises two component (or 'feature') scores $P(S|T)$ and $P(T)$ which are to be multiplied together (as shown in (1)). The first of these, $P(S|T)$, gives the likelihood that the source sentence S and the candidate translation T are translationally equivalent, i.e. that the meaning expressed in S is also captured in T. This feature

is generally referred to as the *translation model*. The second feature, $P(T)$, gives the likelihood that the candidate translation T is actually a valid sentence in the target language and is generally referred to as the *language model*.

Source Sentence:	Le chat entre dans la chambre.
Adequate Fluent translation:	The cat enters the room.
Adequate Disfluent translation:	The cat enters in the bedroom.
Fluent Inadequate translation:	My Granny plays the piano.
Disfluent Inadequate translation:	piano Granny the plays My

(3)

Thus, the traditional notions of adequacy and fluency used in human evaluations are modelled completely separately. As an example of how these might be used in practice, see the French-to-English examples in (3). Note that a translation would need to receive high adequacy and fluency scores in order to be deemed a good translation overall.

The translation model score is based on the extent to which the source sentence meaning is also expressed in the candidate translation; the details of how this score is computed are given in Section 3.2, but it is generally based on *lexical* correspondences only. Meanwhile, the language model score is based on the extent to which the candidate translation is likely to be a valid sentence in the target language regardless of whether or not its meaning bears any relationship to the source sentence. Details of how this score is computed are given in Section 3.1, but it is generally based on the occurrence frequencies of substrings of the sentence in target-language corpora. However, the final score depends on *both* of these features and the objective is to find the translation with the best balance, i.e. the best combined score.

The log-linear model is more general than the noisy-channel model in that it expresses scoring in terms of aggregation of an unlimited number of feature scores. First a technical note: the log-linear model uses log probabilities rather than regular ones – they are converted to log probabilities simply by applying the log function – and log probabilities are added rather than multiplied.[1] Thus, generally $log(X{\cdot}Y) = log(X) + log(Y)$ and, more specifically with respect to the noisy-channel model, $log(P(S|T){\cdot}P(T)) = log(P(S|T)) + log(P(T))$.

We focus now on the right-hand-side of the log-linear model equation $\sum_{m=1}^{M} \lambda_m{\cdot}\, h_m(T, S)$, where we see that this model comprises a set of log feature scores to be added together. That is, the $\sum_{m=1}^{M}$ notation indicates that there are a total of M features to be scored and that their individual scores are to be added up (the sigma, $\sum$). These individual scores are to be computed by multiplying together two feature-specific values, λ_m and $h_m(T, S)$, where λ_m is simply a weight indicating the importance of that feature relative to the other features, and $h_m(T, S)$ is the log probability assigned to the source–candidate pair by that feature. A minimum of two features are usually used: the translation model and language model features exactly as described in (1) for the noisy-channel model. In other words, we can express the noisy-channel model as a log-linear model without changing anything. This is achieved by specifying the set of features to be used as in (4):

m	λ_m	$h_m(T, S)$
1	0.5	$\log(P(S\|T))$
2	0.5	$\log(P(T))$

(4)

Here we have designated feature 1 as the translation model and feature 2 as the language model; the order is not important. The assigned λ values are relative to each other, i.e. the fact that we've given the same values to each of the features indicates that they

are of equal importance[2] in (4). We then use this feature set to expand the log-linear model formula: as we have two features, there will be a total of two scores to add together and these scores are specified according to the values given in the feature set. Thus, the expansion is as in (5):

$$\begin{aligned} Translation &= argmax_T\ \lambda_1 \cdot h_1(T,S) + \lambda_2 \cdot h_2(T,S) \\ &= argmax_T\ 0.5log(P(S|T)) + 0.5log(P(T)) \end{aligned} \quad (5)$$

A significant advantage of this alternative, log-linear formulation is that we can adjust (or 'tune') the relative importance of the features (or 'parameters') being used. For example, we could decide that the translation model is twice as important as the language model and should therefore have twice the influence. This change can be made straightforwardly by making the translation model's λ value double that of the language model: given that the weights must sum to 1, we would give the translation model a weight of 0.67 and the language model a weight of 0.33 (cf. footnote 2). In practice, however, these weights are set empirically relative to the translation task at hand; this is discussed in more detail in Section 3.3.

Another important advantage of the log-linear model is that more features than just these two can be easily incorporated by defining their λ and h values and adding them to the feature set. For example, we could define a translation model which operates at the level of stems rather than full form tokens, or a language model across part-of-speech sequences rather than token sequences. We will not address definitions of additional features any further in this paper, but we provide a list of the typically used features in Section 3.3.

Up to this point, we have been looking at the bigger picture. Now, however, we move to more detailed issues, specifically what the translation and language models actually look like, how they are generated and how they are used during translation. In SMT the resource-induction phase is referred to as *training* while the translation process is referred to as *decoding* (or *testing*). In between, we look at the *tuning* phase where the optimal λ values are learned for the range of features used in an SMT system.

3. Training

The key resources underpinning SMT are corpora: monolingual target-language data for inducing the language model, and sentence-aligned source–target data for inducing the translation model. In SMT, the training data is used to derive the language and translation models, and once these models are generated the parallel data used to induce them are thereafter ignored. When translating new input, the system will access the models but *not* the original training data. In the next sections, we describe how these models are induced.

3.1. LANGUAGE MODEL

As mentioned above, the language model $P(T)$ gives the likelihood that a target string T is actually a valid sentence in the target language. Thus, the language model provides us with two things: (i) a model of the monolingual training corpus and (ii) a method for computing the probability of a (possibly) previously unseen string T using that model. The model extracted from the training corpus comprises relative frequencies for (a subset of) the substrings occurring in that corpus, and the probability assigned to a new sentence is based on the substrings occurring in that sentence which are also part of the model.

One very simple language model comprises just those substrings corresponding to the word tokens in the corpus, where each word type is assigned a probability computed by dividing the number of occurrences of that word in the corpus by the total number of word tokens in the corpus. This model is called a *unigram* model because the substrings extracted comprise single words. An example of a unigram model is given in (6).

$$\begin{array}{ll} \text{Corpus:} & \textit{I need to fly to London tomorrow} \\ \text{Unigram Model:} & P(\text{I}) = 1/7 \;\; P(\text{need}) = 1/7 \;\; P(\text{to}) = 2/7 \;\; P(\text{fly}) = 1/7 \\ & P(\text{London}) = 1/7 \;\; P(\text{tomorrow}) = 1/7 \end{array} \tag{6}$$

We can use this model to assign a probability to any sentence by simply multiplying together the model's probability for each word in that sentence. We give an example in (7):

$$\begin{aligned} P(\textit{I need to fly tomorrow}) &= P(\text{I}) \cdot P(\text{need}) \cdot P(\text{to}) \cdot P(\text{fly}) \cdot P(\text{tomorrow}) \\ &= 1/7 \cdot 1/7 \cdot 2/7 \cdot 1/7 \cdot 1/7 \\ &= 2/16,807 \end{aligned} \tag{7}$$

However, this model has some very significant weaknesses. As the model knows only about single word types, it cannot discriminate between well-formed sentences and those with incorrect word order. For example, the string in (8) is assigned exactly the same probability as the sentence in (7) purely because they contain the same set of lexical items, albeit in a crucially different order.

$$\begin{aligned} P(\textit{fly need I to tomorrow}) &= P(\text{fly}) \cdot P(\text{need}) \cdot P(\text{I}) \cdot P(\text{to}) \cdot P(\text{tomorrow}) \\ &= 1/7 \cdot 1/7 \cdot 1/7 \cdot 2/7 \cdot 1/7 \\ &= 2/16,807 \end{aligned} \tag{8}$$

For that matter, the model will assign an even higher probability to sentences with greater numbers of frequently occurring words, such as in (9):

$$\begin{aligned} P(\textit{to to to to to}) &= P(\text{to}) \cdot P(\text{to}) \cdot P(\text{to}) \cdot P(\text{to}) \cdot P(\text{to}) \\ &= 2/7 \cdot 2/7 \cdot 2/7 \cdot 2/7 \cdot 2/7 \\ &= 32/16,807 \end{aligned} \tag{9}$$

The model also tends to assign higher scores to shorter sentences, simply because the fewer words there are, the fewer probabilities need to be multiplied together. Thus, asked to choose between the proper sentence *P*(*I need to fly tomorrow*), which was assigned probability 2/16,807 in (7), and the subject-less string *P*(*need to fly tomorrow*) as given in (10), the model will prefer the latter.

$$\begin{aligned} P(\textit{need to fly tomorrow}) &= P(\text{need}) \cdot P(\text{to}) \cdot P(\text{fly}) \cdot P(\text{tomorrow}) \\ &= 1/7 \cdot 2/7 \cdot 1/7 \cdot 1/7 \\ &= 2/2401 \end{aligned} \tag{10}$$

Of course, if the sentence to be scored contains a word which does not exist in the model then that sentence is assigned probability zero, as in the sentence in (11) which contains the word *return*.

$$\begin{aligned} P(\textit{I need to return tomorrow}) &= P(\text{I}) \cdot P(\text{need}) \cdot P(\text{to}) \cdot P(\text{return}) \cdot P(\text{tomorrow}) \\ &= 1/7 \cdot 1/7 \cdot 2/7 \cdot 0 \cdot 1/7 \\ &= 0 \end{aligned} \tag{11}$$

However, the model will assign exactly the same score to a sentence containing a single word which is valid but unseen as it does to all other sentences which contain at least one unseen word. For example, the (French) sentence in (12) is also assigned a zero probability.

$$\begin{aligned} P(\textit{Je dois retourner demain}) &= P(\text{Je}) \cdot P(\text{dois}) \cdot P(\text{retourner}) \cdot P(\text{demain}) \\ &= 0 \cdot 0 \cdot 0 \cdot 0 \\ &= 0 \end{aligned} \tag{12}$$

All of these weaknesses can be addressed, at least to a certain extent. The issue of preferring shorter sentences is handled in a straightforward way using normalisation (see (13) and (15) for how bigram and longer *n*-gram probabilities are normalised by dividing their counts by the counts of lower order *n*-grams). The first and most obvious strategy for dealing with unknown words is to increase the size of the training corpus. However, despite the fact that language models are commonly induced from corpora comprising upwards of 1 billion words, of course this strategy alone is not sufficient, as no automatically derived corpus can ever be guaranteed to contain *all* the words in a particular language. Techniques referred to as *smoothing* (Jelinek 1977) are therefore also applied in order to reserve some small amount of probability for unseen words.[3] The amount reserved is likely to be much less than the amount of probability assigned to a word occurring just once. This means that while no sentence will receive a zero score, the model can differentiate between sentences containing few (e.g. (11)) and many (e.g. (12)) unseen words.

The issue of differentiating between well-formed and ill-formed sentences still remains, and is addressed by building language models which remember longer substrings from the training corpus. The *bigram* language model is more complex than the unigram one as it comprises all two-word substrings (bigrams) occurring in the corpus. Bigram language models model word *sequences*, meaning that the assigned probabilities reflect the likelihood that the second word in the bigram should follow the first word. For instance, the bigram probability for the word sequence *to fly* will reflect the likelihood that the word *to* should be followed by the word *fly*, as opposed to being followed by any other word (*run*, say). Thus, each bigram type is assigned a probability computed by dividing the number of occurrences of that bigram in the corpus by the total number of occurrences of the first word of the bigram in the corpus. That is, the bigram probability for *to fly* is calculated as in (13):

$$\frac{\text{count(``to fly'')}}{\text{count(``to'')}} \tag{13}$$

The bigram model for the same sentence shown in (6) is given in (14). We can use this model to assign a probability to any sentence by simply multiplying together the model's probability for each bigram in that sentence.

$$\begin{array}{ll} \text{Corpus:} & \textit{I need to fly to London tomorrow} \\ \text{Bigram Model:} & P(\text{need}|\text{I})=1/1\ P(\text{to}\,|\text{need})=1/1\ P(\text{fly}|\text{to})=1/2\ P(\text{to}|\text{fly})=1/1 \\ & P(\text{London}|\text{to})=1/2\ P(\text{tomorrow}|\text{London})=1/1 \end{array} \tag{14}$$

This type of model can be extended to arbitrarily long substrings, or *n*-grams (of length *n*). For example, the trigram language model remembers all three-word substrings in the corpus; the trigram probability for the word sequence *need to fly* will reflect the likelihood

that the substring *need to* should be followed by the word *fly*, as opposed to being followed by any other word. The trigram probability for *need to fly* is calculated in a similar manner to (13), as shown in (15):

$$\frac{\text{count}(\text{“need to fly”})}{\text{count}(\text{“need to”})} \tag{15}$$

However, we again face the problem of unknowns, this time of unknown *n*-grams rather than single words. Of course, unknown *n*-grams where $n > 1$ are more likely than unknown words (cf. *to fly* vs. *to* above), so the larger the *n*-gram language model used, the worse this problem becomes. For example, the sentence in (16) is assigned zero probability because the bigram $P(\text{tomorrow}|\text{fly})$ does not exist in the bigram model given in (14). We saw in (7) that this sentence was assigned a non-zero probability by the unigram model.

$$\begin{aligned} P(\textit{I need to fly tomorrow}) &= P(\text{need}|\text{I}) \cdot P(\text{to}|\text{need}) \cdot P(\text{fly}|\text{to}) \cdot P(\text{tomorrow}|\text{fly}) \\ &= 1/1 \cdot 1/1 \cdot 1/2 \cdot 0 \\ &= 0 \end{aligned} \tag{16}$$

Increasing the size of the training corpus, while still helpful, yields limited returns as longer *n*-grams are used, as these are seen less and less often. As before, smoothing techniques are used to assign some very small probability to unseen *n*-grams, thereby ensuring that all sentences will obtain non-zero scores. However, this measure does not allow us to differentiate between unseen *n*-grams whose individual words occurred in the training corpus and those containing words never seen before. For example, (16) which contains the unseen bigram $P(\text{tomorrow}|\text{fly})$ receives exactly the same score as the sentence *I need to fly demain* according to the bigram model in (14), although not according to the unigram model given in (6). Thus, while more context-sensitive models can better reward correct word order, the less sensitive ones give greater flexibility. In recognition of this, several models are usually used in combination; that is, for a given sentence, the unigram, bigram and trigram models might all be used, each with a different weight (typically preferring larger matches), and their scores combined to give a final probability for that sentence.

3.2. TRANSLATION MODEL

The translation model $P(S|T)$ gives the likelihood that the source sentence S and the candidate translation T are translationally equivalent, i.e. that the meaning expressed in S is also captured in T. Thus, the translation model provides us with two things: (i) a model of the sentence-aligned source–target training corpus, and (ii) a method for computing the probability that S and T are equivalent using that model. The model extracted from the parallel training corpus comprises relative frequencies for (a subset of) the source–target substring pairs occurring in that corpus. The probability assigned to a new source–translation pair is then based on the substring pairs occurring in that source–translation pair which are also part of the model.

In fact, the translation model is always *directional*, meaning that $P(S|T)$ actually expresses the probability that, given target candidate T, source sentence S is likely to have given rise to T. $P(S|T)$ is, then, a target-to-source (or ‘reverse’) model of translation. This is somewhat counter-intuitive: one would expect to want to compute the probability of the hypothesised target string given the source sentence, i.e. $P(T|S)$. However, the equation in (1) came about via Bayes theorem to manipulate another equation $P(S, T) = P(T, S)$, 2

$\tilde{s}$	$\tilde{t}$	$P(\tilde{s}\|\tilde{t})$	$P(\tilde{t}\|\tilde{s})$
the big cat	le grand chat	0.9	0.8
the big cat	le gros chat	0.3	0.2

Fig 1. An example (partial) t-table for English-to-French.

a joint probability, which says that the sentences in the bitext are translations of each other.[4]

The translation model itself comprises a set of target-language phrases, each of which is associated with a list of source-language translations for those phrases and corresponding probabilities. An (invented) example is given in Figure 1. In the first column, we see source-language phrases $\tilde{s}$,[5] with alternative translations $\tilde{t}$ given in the second column. The third and fourth columns give the respective target-to-source and source-to-target probabilities of those phrase pairs. The probabilities are assigned based on relative frequency, i.e. the probability of a source phrase given the target phrase is computed by dividing the number of occurrences of that target phrase in the dataset by the number of occurrences of the aligned source–target phrase pair. Note that with respect to Figure 1, the probabilities in the column headed $P(\tilde{t}|\tilde{s})$ sum to 1, as we assume here that the only translations of 'the big cat' are the two French strings under $\tilde{t}$. By contrast, the probabilities in the column headed $P(\tilde{s}|\tilde{t})$ do *not* sum to 1, as there will be other English translations of 'le grand chat' ('the large cat', say) and 'le gros cat' (e.g. 'the fat cat'). Note also in regard to this latter that $P(\textit{the fat cat}|\textit{le gros chat})$ is likely to have a much higher probability than $P(\textit{the big cat}|\textit{le gros chat})$, in any representative corpus ('fat' being the default dictionary translation of 'gros', as opposed to 'big').

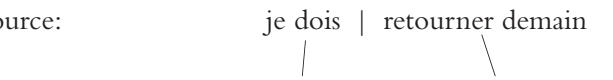
Before going into detail on how translation models are induced, we first illustrate how they are used to assign probabilities to hypothesised translations. A small snippet of a translation model is given in (17) where we see that each English phrase is associated with a list of French phrases, all of which are associated with probabilities.

$$\begin{array}{ll} \text{I need} & \longrightarrow \ldots \text{je dois}(0.1)\ldots \\ \text{I} & \longrightarrow \ldots \text{je}(0.7)\ldots \\ \text{need} & \longrightarrow \ldots \text{dois}(0.05)\ldots \\ \text{to return} & \longrightarrow \ldots \text{retourner}(0.3)\ldots \\ \text{tomorrow} & \longrightarrow \ldots \text{demain}(0.4)\ldots \\ \text{return tomorrow by} & \longrightarrow \ldots \text{retourner demain}(0.0001)\ldots \end{array} \tag{17}$$

A translation model probability is then assigned to any source–hypothesis pair by multiplying together the probabilities of the phrase pairs occurring in that pair according to the model. For instance, in (18) three phrase pair probabilities are multiplied together to arrive at a score for the source–hypothesis pair, while in (19) two such probabilities are multiplied together.

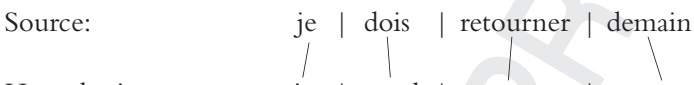
$$\begin{array}{lll} \text{Source:} & \text{je dois} \mid \text{retourner} \mid \text{demain} \\ & \quad | \quad\quad | \quad\quad \backslash \\ \text{Hypothesis:} & \text{i need} \mid \text{to return} \mid \text{tomorrow} \end{array} \tag{18}$$

$$P(\text{Source}|\text{Hypothesis}) = P(\text{je dois}|\text{I need}).\ P(\text{retourner}|\text{to return}).\ P(\text{demain}|\text{tomorrow}) = 0.012$$



$$P(\text{Source}|\text{Hypothesis}) = P(\text{jedois}|\text{i need})\ P(\text{retourner demain}|\text{return tomorrow by}) = 0.00001 \quad (19)$$

It is possible to score the same source–hypothesis pair in different ways as different segmentations may be possible. This is illustrated by comparing (18) and (20), where the same sentence pair is segmented differently in (20) such that four phrase pair probabilities are multiplied together to arrive at a different score for the source–hypothesis pair.



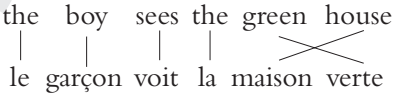
$$P(\text{Source}|\text{Hypothesis}) = P(\text{je}|\text{i}).\ P(\text{dois}|\text{need}).\ P(\text{retourner}|\text{to return}).\ P(\text{demain}|\text{tomorrow}) = 0.0042 \quad (20)$$

The process of inducing a translation model from a sentence-aligned parallel text involves first word-aligning the data (usually via Giza++[6] (Och and Ney 2003), the most often used implementation of the IBM models of word alignment (Brown et al. 1993)), and then extending those word alignments to alignments between phrases. All possible phrase pairs are then extracted such that both sides of the phrase pairs are between 1 and some arbitrary maximum (usually set to 7) words in length, in order to ensure that the calculations are computationally feasible; of course, within individual phrase pairs the source and target phrases will frequently differ in length, given that languages do not generally map meanings on a word-by-word basis.

We will first describe the word alignment process, and then present the extension to phrase alignments.

3.2.1. Word Alignment

The word alignment task involves identifying translational equivalences between individual words in the aligned sentence pairs in a parallel text. A very simple example of such an alignment is given in (21):



(21)

Automatically inducing word alignments is a non-trivial task. Even this relatively straightforward example hints at the difficulty with simply using a dictionary-based approach: how do we determine which instance of *the* aligns with *le* and which with *la*? The SMT approach considers all possible alignments between each sentence pair and works out which ones are the most likely. In order to work out the probability of a particular alignment, several factors are taken into consideration; the most important is the probability that the aligned words correspond to each other in meaning, at least in this specific context, but others include the likelihood of the degree of reordering specified by that alignment ('distortion', in the words of Brown et al. (1990, 1993)) and the likelihood of inserting and deleting words ('fertility'; Brown et al. 1990, 1993).

At this point, the main 'chicken and egg' problem for statistical word alignment becomes apparent (Knight 1999b): we have only sentence pairs. If we had access to the probabilistic dictionary and other statistical information mentioned above, we could use this information to work out the most likely word alignments for our sentence pairs. On the other hand, if we had word-aligned sentence pairs we could gather the word correspondence and other statistics we need. In other words, to obtain the statistics we need word-aligned data and to obtain word-aligned data we need the statistics, but we have neither.

Much of the information we need, however, is actually hidden within the sentence-aligned data. Consider, for example, the sentence-aligned pairs in (22). While we have no prior knowledge of the language pair in question, we can see clearly that there is one word common to both source sentences (*ozftsh*) and one word common to both target sentences (*irg*). Thus, it seems reasonable to conclude that these words correspond to each other and should be aligned.[7]

s_1:	gsv	ylb	**ozftsh**	s_2:	ƹ	trio	**ozftsh**	(22)
t_1:	ov	tzixlm	**irg**	t_2:	fmv	uroov	**irg**	

The English–French version of this parallel corpus is given in (23). The common English word is *laughs* and the common French word is *rit*, and it is clear that our deduction that they correspond is correct.

s_1:	the	boy	**laughs**	s_1:	a	girl	**laughs**	(23)
t_1:	le	garccon	**rit**	t_2:	une	fille	**rit**	

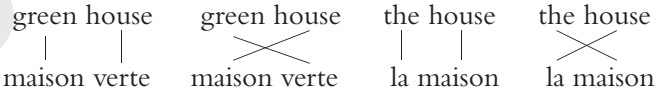
This method of deducing word alignments is employed on a large scale in SMT using the Expectation-Maximisation (EM) algorithm (Dempster et al. 1977). That is, the EM algorithm allows us to systematically discover the word alignments for which sufficient evidence exists across the parallel corpus (without recourse to external resources). In the following paragraphs, we explain how EM works using a very simple example.

Consider first the corpus shown in (24(a)) which comprises two sentence pairs. As we currently know nothing about how they should be word-aligned, we simply assume that all possible word alignments are equally likely; the possible word alignments for each sentence pair are given in (24(b)).

(a) Corpus:

green house	the house
maison verte	la maison

(b) word alignments:



(24)

Before starting the EM algorithm we must first initialise our model as shown in (25). We first collect all the word pairs which exist in the set of word-aligned sentence pairs given in (24(b)). We are assuming that English is the source language and French the target, and that our model gives estimates of $P(\text{source}|\text{target})$ for each word pair identified. Thus, the word pairs extracted from (24(b)) are given down the left side of (25). For instance, we see that *maison* is associated in our dataset with *green*, *house* and *the*. During initialisation, we must also assign some probability to each word pair. As we have no way, as yet, of distinguishing between the suggested English translations for each given French word, we must assume that they are all equally likely. Therefore, as shown in the column labelled '0 (Init)', the three possible translations for *maison* are each assigned prob-

ability 1/3 while the two possible translations for *verte* are each assigned probability 1/2, as are the two possible translations for *la*. For any given French word, the probabilities of its set of English translations must always sum to 1.

		0 (Init)	1	2	...	*n*
P(green|maison)	=	$\frac{1}{3}$	$\frac{1}{4}$	$\frac{1}{6}$	...	0.01
P(house|maison)	=	$\frac{1}{3}$	$\frac{1}{2}$	$\frac{4}{6}$	...	0.98
P(the|maison)	=	$\frac{1}{3}$	$\frac{1}{4}$	$\frac{1}{6}$	...	0.01
P(house|verte)	=	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{3}$	...	0.01
P(green|verte)	=	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{2}{3}$	...	0.99
P(house|la)	=	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{3}$	...	0.01
P(the|la)	=	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{2}{3}$	...	0.99

(25)

At this point, the initialisation procedure is complete. The EM algorithm is an iterative algorithm, meaning that it will be carried out a number of times, and the probabilities given in the model are updated after every iteration. The remaining columns in (25) correspond to successive iterations of the algorithm, i.e. column 1 gives the probabilities as they are at the end of the first iteration, column 2 gives them as they are at the end of the second iteration and column *n* gives the probabilities at the end of the final iteration.

Each iteration comprises two steps which can be referred to as the E(xpectation)-step and the M(aximisation)-step. During the E-step, we assign a probability to the alternative word alignments for each of the sentence pairs in the corpus using the word pair probabilities specified in the model. During the M-step, we compute new probabilities for the word pairs in the model based on the probabilities of the word alignments specified in the corpus, and update the model using these new word pair probabilities. We will further explain these steps by showing how the probabilities in column 1 in (25) are computed given the word alignments in (24(b)) and the word pair probabilities in column 0 in (25); the steps are illustrated in (26).

First the E-step: in order to compute the probability of a particular word alignment for a sentence pair, we multiply together the probabilities (given in the model) of each linked source word given the target word to which it is linked. Thus, for example, for the first word-aligned sentence pair shown in (26), we see that the probability of *green* given *maison* is 1/3 and the probability of *house* given *verte* is 1/2 according to column 0 of our model. Multiplied together, they give a probability of 1/6 for the overall word alignment. Similarly, for the other possible word alignment of that same sentence pair, we see that the probability of *house* given *maison* is 1/3 and the probability of *green* given *verte* is 1/2, thus also giving an overall probability of 1/6 for the word alignment. Once this calculation has been completed for all the word alignments in the dataset, a final piece of book-keeping remains: wherever we have more than one possible word alignment for a sentence pair, we must normalise such that the probabilities of those alternative word alignments sum to 1. To achieve this, we add up the probabilities of the alternative alignments for the given sentence pair and then divide each alternative word alignment by the total. For instance, as we have two different word alignments for the sentence pair <*green house*,*maison verte*> and each has probability 1/6, we divide both those probabilities by 2/6 to give the final probability for each. Not surprisingly, given the probabilities in column 0 of our model, both these alignments are judged to be equally likely at this point (each having probability of 1/2). However, our first iteration is still not complete as we have not yet addressed the M-step.

Iteration 1 – E-step:

$$P\left(\begin{array}{c}\text{green house}\\ \text{maison verte}\end{array}\right) = \tfrac{1}{3}\cdot\tfrac{1}{2} = \tfrac{1}{6} \;/\tfrac{2}{6} = \tfrac{1}{2}$$

$$P\left(\begin{array}{c}\text{green house}\\ \text{maison verte}\end{array}\right) = \tfrac{1}{3}\cdot\tfrac{1}{2} = \tfrac{1}{6} \;/\tfrac{2}{6} = \tfrac{1}{2}$$

$$P\left(\begin{array}{c}\text{the house}\\ \text{la maison}\end{array}\right) = \tfrac{1}{2}\cdot\tfrac{1}{3} = \tfrac{1}{6} \;/\tfrac{2}{6} = \tfrac{1}{2}$$

$$P\left(\begin{array}{c}\text{the house}\\ \text{la maison}\end{array}\right) = \tfrac{1}{2}\cdot\tfrac{1}{3} = \tfrac{1}{6} \;/\tfrac{2}{6} = \tfrac{1}{2}$$

Iteration 1 – M-step:

$$\begin{array}{lll} P(\text{green}|\text{maison}) = \tfrac{1}{2} & /\,2 = \tfrac{1}{4} \\ P(\text{house}|\text{maison}) = \tfrac{1}{2} + \tfrac{1}{2} = 1 & /\,2 = \tfrac{1}{2} \\ P(\text{the}|\text{maison}) = \tfrac{1}{2} & /\,2 = \tfrac{1}{4} \\ P(\text{house}|\text{verte}) = \tfrac{1}{2} & /1 = \tfrac{1}{2} \\ P(\text{green}|\text{verte}) = \tfrac{1}{2} & /1 = \tfrac{1}{2} \\ P(\text{house}|\text{la}) = \tfrac{1}{2} & /1 = \tfrac{1}{2} \\ P(\text{the}|\text{la}) = \tfrac{1}{2} & /1 = \tfrac{1}{2} \end{array} \quad (26)$$

During the M-step, we re-estimate the model's probabilities given the relative frequency of occurrence of each word pair in the set of word alignments. Crucially, however, we do not count a frequency of 1 each time we see a link in the dataset between the word pair we are interested in. Instead, we count the probability that we associated with that particular word alignment during the E-step. This is illustrated in the M-step part of (26). Consider, for example, the word pair *green* and *maison*. This word pair is linked in a single-word alignment in the dataset (the first one), and this word alignment has probability 1/2 according to the E-step. Thus, the frequency of *green* given *maison* is 1/2. We can contrast this with the word pair *house* and *maison*: this word pair is linked in two different word alignments in the dataset, each of which has probability 1/2. Thus, we add together the probabilities of those word alignments, meaning that the frequency of *house* given *maison* is 1. At this point, we still have frequencies rather than probabilities. In order to obtain probabilities, we add up the frequencies of the alternative translations for each target word and then divide each alternative translation frequency by the total. For instance, as we have three different translations for the word *maison* and their frequencies sum to 2, we divide each frequency by 2 to give a probability. Once this has been done, the M-step is complete. The model probabilities in (25) are updated by simply replacing the existing probabilities with those output from the M-step – as shown in column 1 – and iteration 1 is complete.

After just one iteration, we can clearly see that we have improved our knowledge regarding how likely particular word pairs are given the dataset. The three proposed translations for *maison* – *green*, *house* and *the* – are no longer equally likely. Rather, our model now says that, given *maison*, *house* is twice as likely as either *green* or *the*. However, our model still tells us that *house* and *green* are still equally likely given *verte*, and that *house* and *the* are still equally likely given *la*. Thus, we repeat the process in the hope that further improvements in our estimates can be made.

The E- and M-steps for iteration 2 are shown in (27). During the E-step, the probabilities of the different word alignments are computed in exactly the same way as before. This time, however, when we multiply together the probabilities of each linked source word given the target word to which it is linked, those probabilities come from the *updated* model, i.e. from column 1 in (25) rather than column 0. Thanks to these updated probabilities, we can now differentiate between the different word alignments for each sentence pair: the second word alignment is preferred over the first and the third is preferred over the fourth.

Iteration 2 – E-step:

$$
\begin{aligned}
P\left(\begin{array}{c}\text{green house}\\ \text{maison verte}\end{array}\right) &= \tfrac{1}{4}\cdot\tfrac{1}{2} = \tfrac{1}{8} \;/\tfrac{3}{8} = \tfrac{1}{3} \\
P\left(\begin{array}{c}\text{green house}\\ \text{maison verte}\end{array}\right) &= \tfrac{1}{2}\cdot\tfrac{1}{2} = \tfrac{1}{4} \;/\tfrac{3}{8} = \tfrac{2}{3} \\
P\left(\begin{array}{c}\text{the house}\\ \text{la maison}\end{array}\right) &= \tfrac{1}{2}\cdot\tfrac{1}{2} = \tfrac{1}{4} \;/\tfrac{3}{8} = \tfrac{2}{3} \\
P\left(\begin{array}{c}\text{the house}\\ \text{la maison}\end{array}\right) &= \tfrac{1}{2}\cdot\tfrac{1}{4} = \tfrac{1}{8} \;/\tfrac{3}{8} = \tfrac{1}{3}
\end{aligned}
$$

Iteration 2 – M-step:

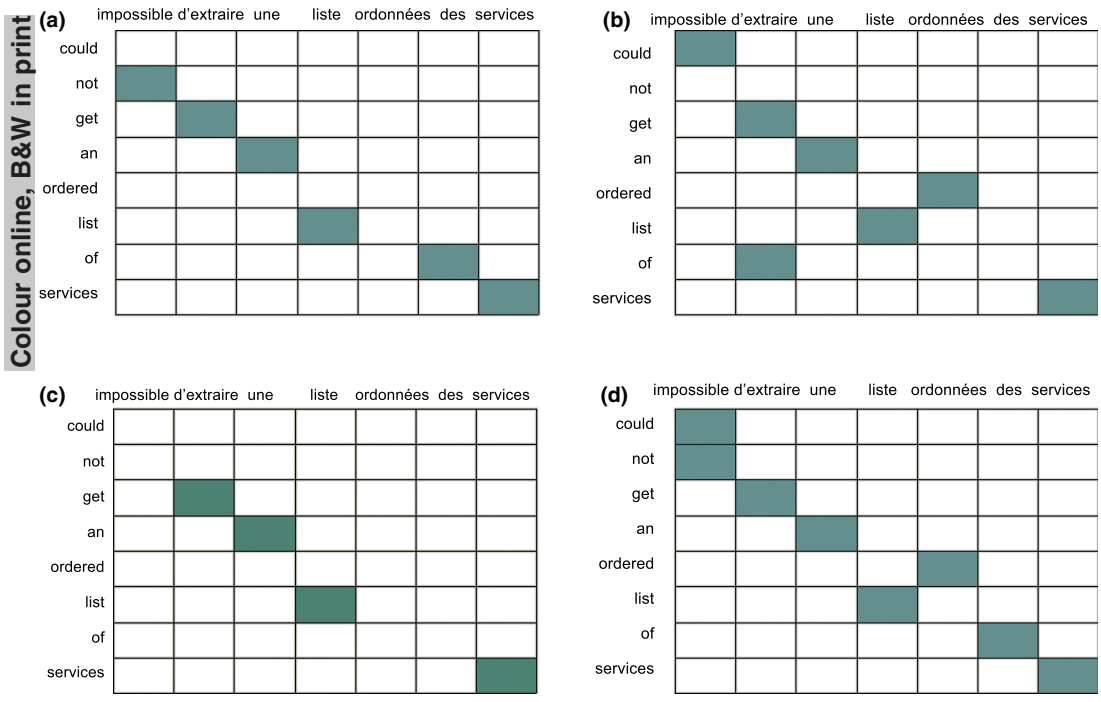
$$
\begin{aligned}
P(\text{green}|\text{maison}) &= \tfrac{1}{3} && /\,2 = \tfrac{1}{6} \\
P(\text{house}|\text{maison}) &= \tfrac{2}{3} + \tfrac{2}{3} = \tfrac{4}{3} && /\,2 = \tfrac{4}{6} \\
P(\text{the}|\text{maison}) &= \tfrac{1}{3} && /\,2 = \tfrac{1}{6} \\
P(\text{house}|\text{verte}) &= \tfrac{1}{3} && /1 = \tfrac{1}{3} \\
P(\text{green}|\text{verte}) &= \tfrac{2}{3} && /1 = \tfrac{2}{3} \\
P(\text{house}|\text{la}) &= \tfrac{1}{3} && /1 = \tfrac{1}{3} \\
P(\text{the}|\text{la}) &= \tfrac{2}{3} && /1 = \tfrac{2}{3}
\end{aligned}
\qquad (27)
$$

These probabilities in turn feed into the re-estimation of the word pair probabilities in the M-step. For example, although the word pairs *house* given *verte* and *green* given *verte* each occur in a single-word alignment, it is the word alignment probabilities which are counted. Therefore, the latter is counted 2/3 times whereas *house* given *verte* is counted just 1/3 times. After the M-step, the model is updated, allowing us to complete column 2 in (25). Again, we have improved our knowledge: the proposed translations for *verte* are no longer equally likely, but rather *green* is twice as likely as *house*. We continue iterating until we stop improving our estimates; column *n* in (25) suggests the kind of model we may end up with when the process is complete.

3.2.2. Phrase Alignment

Two important characteristics of the word alignment process described above are as follows. Firstly, the alignment algorithm is directional, i.e. if we have an English–French parallel dataset, word-aligning it with English as source and French as target will not necessarily yield exactly the same result as word-aligning it with French as source and English as target. Secondly, the alignment algorithm can produce 1-to-1 alignments (i.e. with one source word mapping to one target word), and 1-to-many alignments (one source word mapping to more than one target word) but not many-to-1 alignments and many-to-many alignments. In practice when training an SMT system, the parallel corpus is word-aligned in both directions. This means that even when the objective is to build a French-to-English system, both French-to-English and English-to-French alignments are generated. This allows us to also extract many-to-1 alignments from the target-to-source alignment. However, we still do not have many-to-many alignments, i.e. source-target phrase pairs.

Phrase-alignment heuristics (Koehn et al. 2003; Och and Ney 2003) which operate on the output of the word alignment algorithm are frequently used to extract these many-to-many alignments. Firstly, word alignment is performed in both directions on each training sentence pair and mapped to a bitext grid as shown in Figures 2a,b. We then compute a more precise set of word alignments by taking only those that occur in both sets, as shown in Figure 2c. From this point, it is possible to proceed by iteratively adding further alignments where (i) the alignments are adjacent, and (ii) either the source or target word is currently unaligned: this is shown in Figure 2(d). Finally, phrase pairs can be read from this extended grid: examples given in Groves and Way (2005) include the following:



18 Fig 2. Extracting phrase alignments from word alignments (Groves and Way 2005). The example was taken from a parallel corpus, hence the grammatical error in the French string. (a) English–French word alignment, (b) French–English word alignment, (c) Intersection of alignments and (d) Intersection extended to union.

- could not ↔ impossible
- could not get ↔ impossible d'extraire
- get an ↔ d'extraire une
- ordered list ↔ liste ordonnées
- get an ordered list ↔ d'extraire une liste ordonnées
- could not get an ordered list ↔ impossible d'extraire une liste ordonnées.

3.3. TRANSLATION AND EVALUATION FOR TRAINING PURPOSES

Once we have learned the required models from the training data, and assuming we have an algorithm that accepts an input sentence and returns a translation (to be discussed in detail in Section 4), a final training task still remains before we have a full system that can then be used to translate. In Section 2, we looked at the righthand-side of the log-linear model equation $\sum_{m=1}^{M} \lambda_m.h_m(T, S)$ and we saw that it comprises a set of log feature scores to be added together. Recall that the feature scores are computed by multiplying together two feature-specific values, λ_m and $h_m(T, S)$, where λ_m is a weight indicating the importance of that feature relative to the other features being used. By changing the λ values, we can adjust (or 'tune') the relative importance of the features being used. While we have focused on the language model and translation model features in this paper, other features are frequently used in addition to these, typically including:

1. an *n*-gram language model over target sequences,
2. a source-to-target phrase table,

3. a target-to-source phrase table,
4. source-to-target lexical translation probabilities,
5. target-to-source lexical translation probabilities,
6. a phrase reordering model,
7. the standard word/phrase penalty which allows for control over the length of the target sentence.

Each of these features must be associated with a λ value controlling its influence on the final translation decision. Och (2003) proposed a widely adopted approximation technique called Minimum Error Rate Training (MERT) to estimate the model parameters for a small number of features, such as those listed above.

MERT requires that we define an error function that corresponds to translation accuracy. Most commonly, the error function used is the BLEU evaluation metric (Papineni et al. 2001, 2002), which works by comparing each output translation to a reference translation and computing a similarity/dissimilarity score; we describe BLEU in detail in Section 3.3.1. MERT also requires a second component – a development set (or 'devset') – which is a set of source–target sentence pairs that were not part of the training set used to induce the translation model (or any other features), and that are not part of any test set that might be used to evaluate overall system performance. For any set of λ values, we translate the source sentences in the devset via the MT system and use BLEU to compare the output translations against the devset target sentences. This gives us a measure of translation accuracy, where we assume the target sentences in the devset to be the correct 'reference' translations that we are aiming to produce from the MT system. Using MERT, if we then adjust the λ values, we can again translate and evaluate using the devset to see whether the BLEU score has gone up or down. Thus, MERT is an iterative process: we repeatedly translate the devset using different λ values in order to identify the set of λ values that gives the best performance according to our error function on that devset.

Essentially, therefore, MERT involves optimising system performance to *one* particular metric on a devset of sentences, and hoping that this carries forward to the test set at hand; the closer the devset to the test set, the more appropriate the settings obtained via MERT will be. MERT provides a simple and efficient method to estimate the optimal weights for the features in the SMT model; however, it can only handle a small number of parameters, and when the number of features increases, there is no guarantee that MERT will find the most suitable combination (Chiang et al. 2008).

3.3.1. The BLEU Metric

In this short section, we digress into the related field of MT evaluation in order to describe how the evaluation metric most commonly used for MERT works.

The BLEU metric (Papineni et al. 2001, 2002) evaluates MT system quality by comparing output translations to their reference translations in terms of the numbers of co-occurring n-grams. The main score calculated is the n-gram precision p_n for each pair of candidate and reference sentences. This score represents the proportion of n-word sequences in the candidate translation which also occur in the reference translation. Importantly, if an n-gram occurs j times in the candidate translation and i times in the reference translation such that $i \leq j$ then this sequence is counted only i times; this corresponds to the intuition that 'a reference word sequence should be considered exhausted after a matching candidate word sequence has been identified' (Papineni et al. 2001). Thus, n-gram precision p_n is calculated according to equation (28):

$$p_n = \frac{|c_n \cap r_n|}{|c_n|} \tag{28}$$

where

- c_n is the multiset of n-grams occurring in the candidate translation.
- r_n is the multiset of n-grams occurring in the reference translation.
- $|c_n|$ is the number of n-grams occurring in the candidate translation.
- $|c_n \cap r_n|$ is the number of n-grams occurring in c_n that also occur in r_n such that elements occurring j times in c_n and i times in r_n occur maximally i times in $|c_n \cap r_n|$.

As it is generally not the case that MT output is evaluated one sentence at a time, n-gram precision is usually calculated over large sets of sentences (typically not less than 1000). In this case, p_n is the proportion of co-occurring n-word sequences in the set over the total number of n-word sequences in the set.

While precision scores p_n can be obtained for any value of n,[8] Papineni et al. (2001) point out that greater robustness can be achieved by combining scores for all values of n into a single metric. It is not surprising that as the value of n increases, the score p_n decreases because longer matching word sequences are more difficult to find (cf. Section 3.1 for a similar discussion regarding higher-order n-grams in language modelling). If the average n-gram precision score is calculated without taking this factor into account (i.e. by simply summing the values for p_n and dividing by N, the largest value for n), then the scores for longer n-grams will be too small to have much influence on the final score. In order to make the BLEU metric more sensitive to longer n-grams, the combined score p_N is calculated by summing over the logarithm of each p_n multiplied by weight $1/N$ as given in equation (28):

$$p_N = \exp\left(\sum_{n=1}^{N} \frac{1}{N} \log(p_n)\right) \tag{29}$$

A candidate translation which is longer than its reference translation is implicitly penalised during the calculation of p_n. In order to impose a corresponding penalty on candidate translations which are shorter than their reference translations, a *brevity penalty BP* is introduced and the combined precision score p_N is multiplied by this penalty. BP is incorporated to prevent the output of single-word utterences like 'the', which would otherwise score highly. Papineni et al. (2001) state that BP is a decaying exponential in the length of the reference sentence over the length of the candidate sentence. This means that if the reference is the same length or longer than the candidate, then the penalty is 1, and greater than 1 if the candidate is shorter than the reference. Furthermore, if candidate c_x is 1 word shorter than its reference r_x and c_y is also 1 word shorter than r_y, but r_x is longer than r_y, then the BP for c_y should be greater than the BP for c_x. Thus, BP is calculated according to equation (30):

$$BP = e^{\max\left(1 - \frac{length(R)}{length(C)}, 0\right)} \tag{30}$$

Note that as calculating the brevity penalty for each sentence and averaging it over the set of sentences is considered by Papineni et al. (2001) to be unduly harsh, it is computed over the entire corpus, i.e. *length*(R) is the number of words in the reference set and *length*(C) the number of words in the candidate set. This penalty is then applied to the precision score for the entire candidate translation corpus according to equation (31):

$$BLEU = BP \cdot p_N \tag{31}$$

As the ranking behaviour is more visible in the log domain, Papineni et al. (2001) give equation (32):

$$\log BLEU = min\left(1 - \frac{length(R)}{length(C)}, 0\right) + \sum_{n=1}^{N} \frac{1}{N} \log(p_n) \tag{32}$$

In summary, the combination of *n*-gram precision and penalties for shorter translations mean that in order to achieve a high BLEU score, a set of candidate translations must match the reference translations in length, in word choice and in word order Papineni et al. (2001).

Other evaluation metrics based on similar principles as BLEU have also been introduced, e.g. NIST (Doddington 2002), followed soon thereafter by GTM (Turian et al. 2003), and a couple of years later by METEOR (Banerjee and Lavie 2005). Despite a number of researchers pointing out various problems with such metrics (cf. Callison-Burch et al. (2006) for a criticism of BLEU), all these metrics are still actively used in MT research. While a wide variety of techniques, including human evaluation, are used to compare systems and assess overall translation quality, automatic similarity-based metrics are critical in facilitating the use of MERT.

4. Translation

In previous sections we showed how SMT language and translation models are induced from data (training), and how these and any other parameters in the log-linear model can be optimised (tuning). In Section 3.3, we explicitly assumed that we have an algorithm that accepts an input sentence and returns the most likely translation, but we did not provide any details. In this section, we outline this final piece of the puzzle: the translation phase, referred to also as 'decoding' (or 'testing').

Given any source sentence and a hypothesised translation, we can compute a corresponding probability by using the models generated during training in relation to equation (2) for the log-linear model. By extension, given any source sentence and a set of hypothesised translations, we can work out which hypothesis is most likely. Decoding, therefore, is a search problem: we must find the most probable translation amongst all translations possible given the model.

In theory, this would involve generating the full set of possible translations for any input string. Translations are generated by simply matching substrings from the input sentence against the translation model and, where available, retrieving their translations. These substring translations can then be concatenated in any order to generate a full translation hypothesis. Figure 3 shows some of the possible substring translations for a Spanish sentence, from which any sequence of substrings which fully covers the source sentence can be selected in any order to form a hypothesis.

There is no necessity in the decoding process to start with the leftmost source word, and proceed on a strictly left-to-right basis. In our Spanish-to-English example it would seem like a reasonable strategy but of course for some language pairs (Arabic-to-English, say) that would be entirely the wrong thing to do. The good thing for the decoder is that it essentially makes no difference, and the best translation – based on the probability computed using the log-linear formula – will be chosen irrespective of which source-language words are translated in which order.

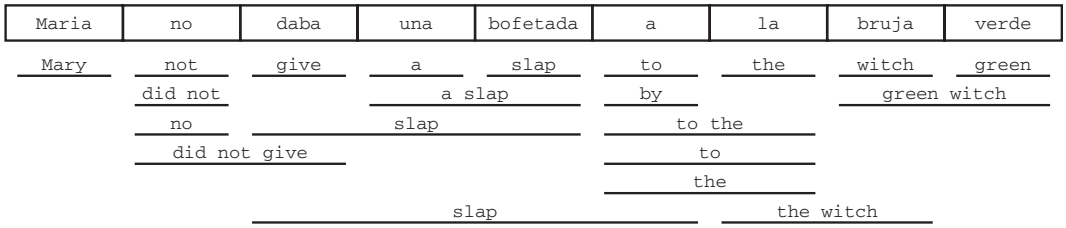


Fig 3. Some English translation options for the Spanish sentence *Maria no daba una bofetada a la bruja verde* (Koehn 2004).

However, it was shown in Knight (1999a) that the decoding problem is NP-complete, i.e. that it cannot be guaranteed that we can generate and score all hypotheses for a source sentence in a finite amount of computing time. Accordingly, decoding is also an optimisation problem; we must avoid generating target-language hypotheses which are unlikely to end up being the most probable, without also missing the good solutions by accident. In short, we must try to find the most probable translation without actually generating all of the hypotheses.

The state-of-the-art implementation of decoding for SMT is a beam-search decoder (Kochn et al., 2003, 2007). A 'beam' is just an arbitrary number of hypotheses (say, 10,000) that are maintained at any point in the translation process, and such a cut-off ensures that the runtime of any system is manageable in practice. At the start of the translation process, no words have yet been translated. Source-language words are expanded in a monotone or non-monotone manner, i.e. following the source word/phrase order or not. New hypotheses are generated from the existing expanded hypotheses by extending them with a phrasal translation that covers some of the source input words which have not yet been translated. Each hypothesis is added into a beam stack as a new node. Figure 4 shows six stacks, which is sufficient for covering the first 6 words in target-language hypotheses. Each stack is marked by the covered source words (one word covered in stack 1, two words in stack 2, etc.) during expansion. A newly created hypothesis will be placed in a new stack further down. For example, if we return to Figure 3, the second from the top phrase in stack 2 (comprising two words, *Mary did*, say) is linked to various hypotheses in stacks 3 (*not*, i.e. three words – *Mary did not* – are now covered), 4 (*give*, four words: *Mary did not give*) and 5 (*a*, five words: *Mary did not give a*), all of which might be partial candidate translations (i.e. valid paths through the search space in Figure 3) of the source string *Maria no daba una bofetada a la bruja verde*.

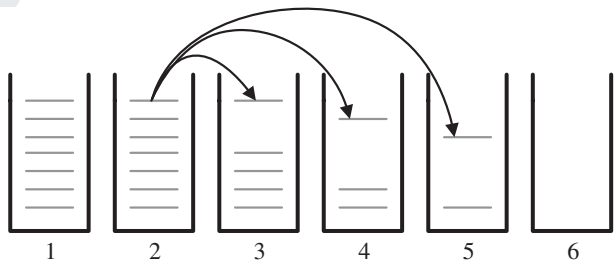


Fig 4. Hypothesis expansion via stack decoding (Koehn 2004).

The key idea is that each beam stack has a maximum size, and once the maximum size is reached a new hypothesis can be added only if its score is higher than the lowest-scoring hypothesis on the stack (in which case that lowest-scoring hypothesis is dropped to make room). Of course, hypotheses are scored in part by computing how likely they are according to the log-linear formula. However, relying on just this probability is not sufficient for pruning purposes. Stack 4, for instance, holds the hypothesised four-word translation sequences from the source sentence. Some of those word sequences will have been easier to translate than others, meaning they will likely have higher probabilities. However, we do not want to prune away hypotheses from the stack that have lower probabilities simply because they are more difficult to translate, leaving only translations for the easier sequences behind. Therefore, when scoring a hypothesis to decide its inclusion on a stack we also factor in the *future translation cost* (Koehn 2010:167–72); this reflects how difficult it might be to translate the source words *not* covered by that hypothesis. Thus, the future cost estimation should favour hypotheses that have already covered difficult parts of the sentence and have only easy parts left, while discounting hypotheses that have covered the easy parts first.[9]

The final states in the search are hypotheses that cover all source words: these are found in stack *n* where *n* is the length of the source sentence (9, in Figure 3). Among the *n*-best list of possible translations created by the decoder following different paths through the search space, the one with the highest probability is finally selected as the best translation.

5. Conclusion

In this paper, we have given an overview of the basic model of SMT, along with in-depth presentations of the core components of SMT systems. In Section 2 we presented the big-picture view of the SMT approach. In Section 3 we described the main approaches to inducing training data from corpora, focusing in particular on language modelling, translation modelling and tuning via Minimum Error Rate Training. Finally, in Section 4 we gave an overview of the decoding process used in SMT systems to generate translations for previously unseen text. For a detailed discussion on the role of linguists, translators and translations in SMT, along with a description of how the ideas behind SMT developed and how they relate to other MT approaches, please see Way and Hearne (2010). For those readers who are interested in more in-depth accounts of SMT, we recommend Brown et al. (1993), Knight (1999b) and Knight (2010). For pointers to the primary literature relating to each of the sub-tasks in SMT, as well as other MT paradigms, consult section 7 of Way (2010), or the appropriate sections in Koehn (2010).

Short Biographies

????????.

Acknowledgement

This work was partially funded by a number of Science Foundation Ireland (<http://www.sfi.ie>) awards, namely: Principal Investigator Award 05/IN/1732, Basic Research Award 06/RF/CMS064 and CSET Award 07/CE/I1142. Thanks also to Philipp Koehn for making available some of his original figures.

Notes

* Correspondence address: Andy Way, NCLT & CNGL, School of Computing, Dublin City University, Collins Avenue, Glasnevin, Dublin D9, Ireland. E-mail: away@computing.dcu.ie

[1] This is done for a very simple reason: the number of multiplications of probabilities performed is often very large. In almost all cases, these probabilities are less than 1, so the resultant score is often extremely small, and cannot be accurately captured unless a large number of decimal places are used.

[2] Note that these λ values sum to 1. In practice, as long as two equivalently weighted features have the same value, and that normalisation (cf. (13) and (15) below) later takes place, this requirement that the λ values sum to 1 can be relaxed.[3] The main differences between different smoothing techniques relate to how much probability mass is subtracted out ('discounting') and how it is redistributed ('back-off'). The most popular method used in SMT is Kneser–Ney smoothing (Kneser and Ney 1995).

[4] In Section 3.3, the reader will note that both a source-to-target and a target-to-source translation model (also known as a *phrase table, translation table* or indeed *t-table*) are employed as typical features used in an SMT system (cf. Figure 1). While there is little justification *per se* for including both – translation is, after all, directional by its very nature – doing so causes translation quality to improve. Anecdotally, the story goes that one of the major SMT developers in our field one day selected the wrong translation table, but that the system's BLEU score didn't go down by very much. Once the error was discovered, including both t-tables caused performance to improve upon using just the one, correct t-table, and henceforth we all use both!

[5] The ~ indicates that we are dealing with 'phrases', as opposed to 'sentences'; note that the term 'phrase' is not used in any linguistic sense here, but rather is used to refer to any substring (n-gram sequence) of between 1 and (usually around) 7 words in length.

[6] <http://www.fjoch.com/GIZA++.html>.

[7] Our exposition here draws heavily on how one might translate between Arcturan and Centauri (Knight 1997, 1999b). For newcomers to SMT, these resources are an excellent starting point.

[8] Scores can be obtained for any *reasonable* value of n; in Papineni et al. (2001, 2002) the maximum value for n considered was 4.

[9] The 'ease' or 'difficulty' associated with translating certain parts of a sentence is usually expressed in terms of weighted log probabilities which take into account (at least) language model, translation model and reordering costs. As you might expect, common words are 'easier' to translate in this model than less frequent words, despite these being among the 'hardest' words to get right for humans.

Works Cited

Banerjee, S. and A. Lavie. 2005. METEOR: An automatic metric for MT evaluation with improved correlation with human judgments. Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization, 65–72. Ann Arbor, MI. 6

Brown, P., J. Cocke, S. Della Pietra, V. Della Pietra, F. Jelinek, J. Lafferty, R. Mercer, and P. Roosin. 1990. A statistical approach to machine translation. Computational Linguistics 16. 79–85.

——, S. Della Pietra, V. Della Pietra, and R. Mercer. 1993. The mathematics of statistical machine translation: parameter estimation. Computational Linguistics 19. 263–311.

Callison-Burch, C., M. Osborne, and P. Koehn. 2006. Re-evaluating the role of BLEU in machine translation research. EACL-2006 11th Conference of the European Chapter of the Association for Computational Linguistics, Proceedings. 249–56. Trento, Italy. 7

Chiang, D., Y. Marton, and P. Resnik. 2008. Online large-margin training of syntactic and structural translation features. Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing, 224–33. Honolulu, HI.

Dempster, A., N. Laird, and D. Rubin 1977. Maximum likelihood from incomplete data via the EM algorithm. Journal of the Royal Statistical Society Series B 39. 1–38.

Doddington, G. 2002. Automatic evaluation of MT quality using N-gram co-occurrence statistics. Proceedings of Human Language Technology Conference 2002, 138–45. San Diego, CA. 8

Groves, D. and A. Way 2005. Hybrid data-driven models of machine translation. Machine Translation 19. 301–23.

Jelinek, F. 1977. Statistical methods for speech recognition. Cambridge, MA: MIT Press.

Kneser, R. and H. Ney. 1995. Improved backing-off for m-gram language modeling. Proceedings of the IEEE International Conference on Acoustics Speech and Signal Processing, Vol. 1, 181–4, Detroit, MI.

Knight, K. 1997. Automating knowledge acquisition for machine translation. AI Magazine 18. 81–96.

——. 1999a. Decoding complexity in word-replacement translation models. Computational Linguistics 25. 607–15.

——. 1999b. A statistical machine translation tutorial workbook. <http://www.isi.edu/natural-language/mt/wkbk.rff>.

Koehn, P. 2004. 'Pharaoh:' A beam search decoder for phrase-based statistical machine translation models. Machine Translation: from Real Users to Research, 6th Conference of the Association for Machine Translation in the Americas, AMTA 2004, Proceedings, 115–24, Washington, DC.

——. 2010. Statistical machine translation. Cambridge, UK: Cambridge University Press.

——, F. Och, and D. Marcu. 2003. Statistical phrase-based translation. HLT-NAACL: Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics, 127–33. Edmonton, AL, Canada.

——, H. Hoang, A. Birch, C. Callison-Burch, M. Federico, N. Bertoldi, B. Cowan, W. Shen, C. Moran, R. Zens, C. Dyer, O. Bojar, A. Constantin, and E. Herbst. 2007. Moses: open source toolkit for statistical machine translation. Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics Companion Volume Proceedings of the Demo and Poster Sessions, 177–80. Prague, Czech Republic.

Och, F. 2003. Minimum error rate training in statistical machine translation. 41st Annual Meeting of the Association for Computational Linguistics, 160–7. Sapporo, Japan.

——, and H. Ney. 2002. Discriminative training and maximum entropy models for statistical machine translation. 40th Annual Meeting of the Association for Computational Linguistics, 295–302. Philadelphia, PA.

——, and ——. 2003. A systematic comparison of various statistical alignment models. Computational Linguistics 29. 19–51.

Papineni, K., S. Roukos, T. Ward, and W.-J. Zhu. 2001. BLUE: a method for automatic evaluation of machine translation. Technical report, IBM T.J. Watson Research Center, Yorktown Heights, NY.

——, ——, ——, and ——. 2002. BLEU: a method for automatic evaluation of machine translation. 40th Annual Meeting of the Association for Computational Linguistics, 311–8. Philadelphia, PA.

Turian, J., L. Shen, and D. Melamed. 2003. Evaluation of machine translation and its evaluation. Machine translation summit IX, ed. by ???, 386–93. New Orleans, LA: ????.

Way, A. 2010. Machine translation. Handbook for computational linguistics and natural language processing, ed. by S. Lappin, C. Fox, and A. Clark. Chichester, UK: Wiley Blackwell.

——, and M. Hearne. forthcoming. On the role of translations in state-of-the-art statistical machine translations. *Compass*.

Author Query Form

Journal: LNC3

Article: 274

Dear Author,
During the copy-editing of your paper, the following queries arose. Please respond to these by marking up your proofs with the necessary changes/additions. Please write your answers on the query sheet if there is insufficient space on the page proofs. Please write clearly and follow the conventions shown on the attached corrections sheet. If returning the proof by fax do not write too close to the paper's edge. Please remember that illegible mark-ups may delay publication.

Many thanks for your assistance.

Query reference	Query	Remarks
Q1	**AUTHOR:A running head short title was not supplied; please check if this one is suitable and, if not, please supply a short title of up to 40 characters that can be used instead.**	
Q2	**AUTHOR:Can 'equation in (1)' be changed to 'equation (1)'? Please confirm.**	
Q3	**AUTHOR:Knight (2010) has not been included in the list, please supply publication details.**	
Q4	**AUTHOR:Please check this website address and confirm that it is correct.**	
Q5	**AUTHOR:Please provide short biographies for both the authors as per the journal style.**	
Q6	**AUTHOR:Please provide the publisher for reference 'Banerjee and Lavie (2005)'.**	
Q7	**AUTHOR:Please provide the publisher for reference 'Callison-Burch 2006'.**	
Q8	**AUTHOR:Please provide the publisher for reference 'Doddington 2002'.**	
Q9	**AUTHOR:Please provide the publisher for reference 'Koehn 2004'.**	
Q10	**AUTHOR:Please provide the publisher for reference 'Koehn et al. 2003'.**	

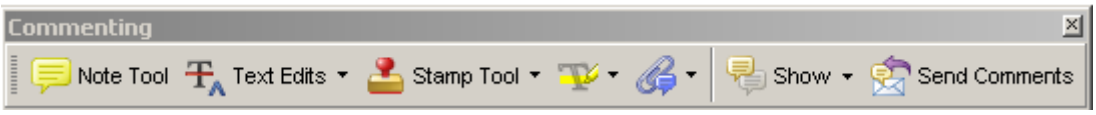
Q11	**AUTHOR:Please provide the publisher for reference 'Koehn et al. 2007'.**	
Q12	**AUTHOR:Please provide the publisher for reference 'Och 2003'.**	
Q13	**AUTHOR:Please provide the publisher for reference 'Och and Ney 2002'.**	
Q14	**AUTHOR:Please provide the publisher for reference 'Papineni et al. 2002'.**	
Q15	**AUTHOR:Please provide the Editors and name of the publisher for this book.**	
Q16	**AUTHOR:Please provide page range for reference 'Way 2010'.**	
Q17	**AUTHOR:Please provide the correct journal name, volume number and page range for reference 'Way and Hearne 2010'.**	
Q18	**AUTHOR: Figure 2 has been saved at a low resolution of 169 dpi. Please resupply at 600 dpi. Check required artwork specifications at http://authorservices.wiley.com/submit_illust.asp?site=1**	

USING E-ANNOTATION TOOLS FOR ELECTRONIC PROOF CORRECTION

Required Software

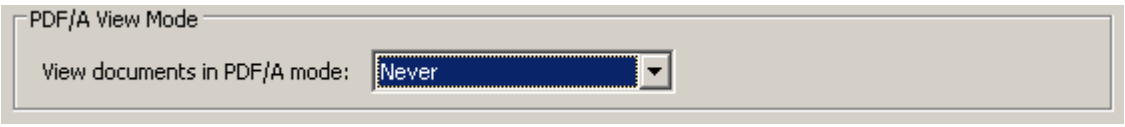
Adobe Acrobat Professional or Acrobat Reader (version 7.0 or above) is required to e-annotate PDFs. Acrobat 8 Reader is a free download: http://www.adobe.com/products/acrobat/readstep2.html

Once you have Acrobat Reader 8 on your PC and open the proof, you will see the Commenting Toolbar (if it does not appear automatically go to Tools>Commenting>Commenting Toolbar). The Commenting Toolbar looks like this:



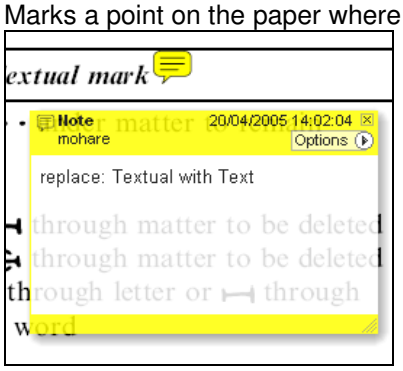
If you experience problems annotating files in Adobe Acrobat Reader 9 then you may need to change a preference setting in order to edit.

In the "Documents" category under "Edit – Preferences", please select the category 'Documents' and change the setting "PDF/A mode:" to "Never".



Note Tool — For making notes at specific points in the text

Marks a point on the paper where a note or question needs to be addressed.

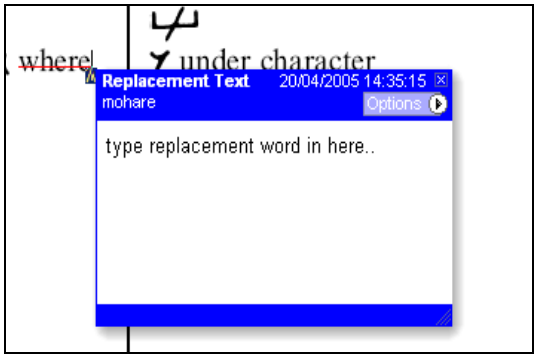


How to use it:

1. Right click into area of either inserted text or relevance to note
2. Select Add Note and a yellow speech bubble symbol and text box will appear
3. Type comment into the text box
4. Click the X in the top right hand corner of the note box to close.

Replacement text tool — For deleting one word/section of text and replacing it

Strikes red line through text and opens up a replacement text box.

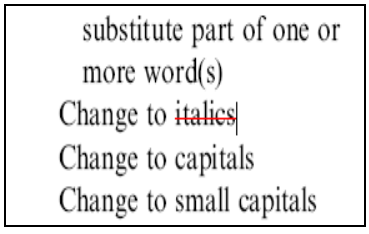


How to use it:

1. Select cursor from toolbar
2. Highlight word or sentence
3. Right click
4. Select Replace Text (Comment) option
5. Type replacement text in blue box
6. Click outside of the blue box to close

Cross out text tool — For deleting text when there is nothing to replace selection

Strikes through text in a red line.



How to use it:

1. Select cursor from toolbar
2. Highlight word or sentence
3. Right click
4. Select Cross Out Text

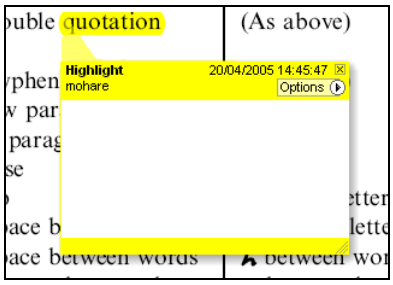
Approved tool — For approving a proof and that no corrections at all are required.

How to use it:

1. Click on the Stamp Tool in the toolbar
2. Select the Approved rubber stamp from the 'standard business' selection
3. Click on the text where you want to rubber stamp to appear (usually first page)

Highlight tool — For highlighting selection that should be changed to bold or italic.

Highlights text in yellow and opens up a text box.

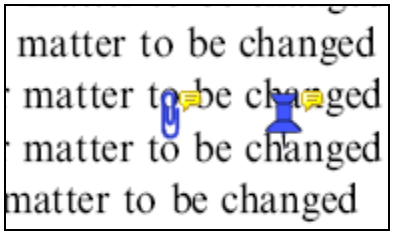


How to use it:

1. Select Highlighter Tool from the commenting toolbar
2. Highlight the desired text
3. Add a note detailing the required change

Attach File Tool — For inserting large amounts of text or replacement figures as a files.

Inserts symbol and speech bubble where a file has been inserted.

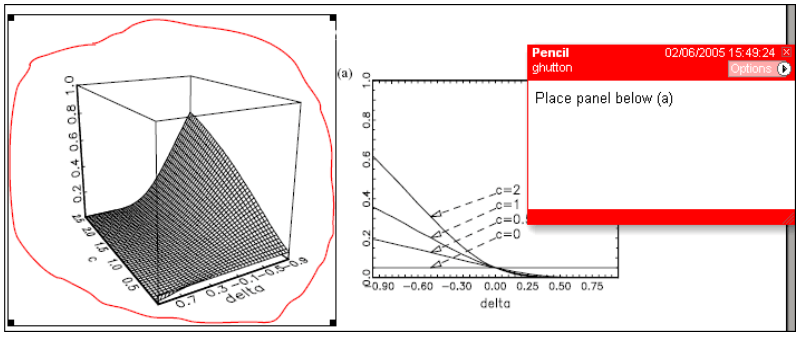


How to use it:

1. Click on paperclip icon in the commenting toolbar
2. Click where you want to insert the attachment
3. Select the saved file from your PC/network
4. Select appearance of icon (paperclip, graph, attachment or tag) and close

Pencil tool — For circling parts of figures or making freeform marks

Creates freeform shapes with a pencil tool. Particularly with graphics within the proof it may be useful to use the Drawing Markups toolbar. These tools allow you to draw circles, lines and comment on these marks.



How to use it:

1. Select Tools > Drawing Markups > Pencil Tool
2. Draw with the cursor
3. Multiple pieces of pencil annotation can be grouped together
4. Once finished, move the cursor over the shape until an arrowhead appears and right click
5. Select Open Pop-Up Note and type in a details of required change
6. Click the X in the top right hand corner of the note box to close.

Help

For further information on how to annotate proofs click on the Help button to activate a list of instructions: