



Executive Summary

VERSION	DATE	AUTHOR
0.1	1/2/2008	Gabriele Reverberi, initial version
0.2	20/2/2008	Contributions by Sandor Szedmak
0.3	25/2/2008	Contributions by Nicolò Cesa-Bianchi
1.0	5/3/2008	Contributions by Gabriele Reverberi
2.0	6/3/2008	Editing by Nicolò Cesa-Bianchi
2.1	13/3/2008	Revised by Nicolò Cesa-Bianchi based on reviewer's comments
2.2	17/3/2008	Revised by Sandor Szedmak and Gabriele Reverberi based on reviewer's comments
3.0	19/3/2008	Final editing by Nicolò Cesa-Bianchi



TITLE	D4.2 — Online Learning Algorithms for Computer-Assisted Translation
STATE	Draft
CONFIDENTIALITY	PU
AUTHOR(S)	Nicolò Cesa-Bianchi, Gabriele Reverberi, Sandor Szedmak
CONFIDENTIALITY	SMART consortium
PARTICIPANT PARTNERS	UniMI, Southampton
WORKPACKAGE	WP4
ABSTRACT	We formulate CAT as an interactive machine translation problem, and show how online learning algorithms can be applied to solve it. We describe the integration of our adaptive learning framework with Portage, a software package for statistical machine translation. The effect of online adaptation, with respect to a nonadaptive machine translation baseline, is empirically shown by a set of translation experiments. Finally, we propose a new machine translation approach based on margin maximization with ∞ -norm regularization.
KEYWORDS	online learning, linear learning, CAT, interactive machine translation, ∞ -norm regularization
REFERENCES	
COMMENTS	
REVIEWER	Cyril Goutte

Contents

1	Overview of this document	5
2	Adaptive approaches to interactive machine translation	7
2.1	Introduction	7
2.2	CAT and online learning	8
2.3	Dynamic re-ranking	11
2.4	Online algorithms for dynamic re-ranking	13
2.5	Some further issues and considerations	17
3	Integration with Portage	19
3.1	Introduction	19
3.2	Integrating the online learner with Portage	22
3.2.1	Measuring performance for CAT	23
3.2.2	Features	24
3.3	CAT tool architecture	27
3.4	Experimental Results	29
3.5	Conclusions	32
4	A large margin translation system	34
4.1	Introduction	34



4.2	The translation model	35
4.3	Training phase	35
4.4	Translation	37
4.4.1	Phrase prediction	38
4.4.2	Decoding	41
4.5	Experiment	43
4.5.1	Examples	46
4.6	Future work	46

Chapter 1

Overview of this document

In Chapter 4 of Deliverable D4.1 we described a theoretical framework for the design and the analysis of adaptive machine translation approaches. This framework builds on margin-based online algorithms that learn a linear function of features extracted from both the source sentence and the candidate translation. This linear function can be used to rank a set of translations of the same source sentence built on a given phrasetable, an operation that can be performed by a suitably modified decoder.

In this document we describe the application of the abovementioned framework to the task of computer-assisted translation (CAT). Our adaptive MT techniques are based on the statistical approach to machine translation (SMT). Unlike the usual suite of CAT tools, such as translation memories, concordancers and terminology databases, SMT can generate reasonably fluent translations of previously unseen sentences by combining the entries of a large table of source language and target language phrase pairs. Each pair in the table is given a score measuring the statistical plausibility of translating the source phrase with the target phrase. The SMT system uses these scores, along with a model of the target language, to drive the search in the phrase table.

Adaptive SMT techniques have the additional property of being able to exploit user feedback information to increase their performance. This makes CAT an excellent domain for the validation of adaptive MT systems. Indeed, the post-editing of the system's suggested translations performed by the user naturally provides the feedback that the adaptive module can use to improve future translations.



Although there is a very natural fit between the online learning protocol and the interactive machine translation model underlying CAT applications, there are also several peculiarities that need to be addressed in a practical implementation. For instance, modern CAT systems are based on sophisticated translation memories that suggest translations to source sentences that are similar enough to sentences that have appeared previously. The machine-translation module must thus interface with the translation memory in a proper way, so that the former comes into play when the latter fails. Other peculiarities are more intrinsic to the machine translation problem; for example, the problem of extracting informative features from sentences. In this work, we associate features mainly with phrasetable entries, as these are the basic modules used by the decoder to build translations. By doing this we end up managing a large number of features, with a potential risk of overfitting due to the extremely sparse encoding.

The document is organized as follows. In Chapter 2 we describe CAT as an interactive machine translation problem, and show how online learning algorithms can be applied to solve it. Then we recall the framework of Deliverable D4.1 and describe in detail the specific learning algorithms that we use for CAT. In Chapter 3 we describe the integration of our adaptive framework with Portage, a software package for statistical machine translation. We use Portage to build a phrasetable from a sentence-aligned parallel corpus. Moreover, we use the Portage decoder to generate a list of candidate translations that are ranked based on the linear function learned by our adaptive module. The effect of online adaptation, with respect to a nonadaptive machine translation baseline, is empirically shown by a set of translation experiments. Finally, in Chapter 4 we propose a new machine translation approach based on margin maximization with ∞ -norm regularization. Experiments show that this new approach is competitive with state-of-the-art statistical MT systems. Ways of integrating this new system with online learning for CAT are under investigation.

Chapter 2

Adaptive approaches to interactive machine translation

2.1 Introduction

The availability of sophisticated phrase-based statistical machine translation systems has promoted the development of interactive machine translation approaches. IMT approaches provide models of human-machine interaction that are suitable for the implementation of machine translation techniques in tasks of computer-assisted translation (CAT). While the initial focus of IMT was on the disambiguation of the source language (see, e.g., the MIND system [19]), more recent approaches center the interaction on the structure of the target text [13]. This target-text mediated approach to IMT opens a range of new interesting possibilities in which SMT systems play a prominent role.

An important example of this new approach to IMT are the TransType [14, 22, 23] and TransType2 [12] projects. The interaction model adopted there is simple: the SMT system interfaces with the CAT tool and provides completions to the current sentences that are compatible with the input typed by the translator. Furthermore, the system is able to revise its suggestions in real time, as the translator enters new characters. If the system provides a correct suggestion, the translator can accept it with a keystroke, thus saving time in producing the



target text. Otherwise, the translator may ignore the system's suggestion and continue to type the intended translation.

State-of-the-art SMT systems are based on several modules and data structures: decoder, phrasetable, language model, and their associated tunable parameters. These modules are built/trained on large corpora of sentence-aligned parallel text. In traditional CAT applications the system undergoes a training phase before being used in interactive mode. This means that the SMT module does not further adapt during the interaction with the user.

The purpose of this chapter is to explore adaptive approaches to interactive machine translation. In particular, our goal is to take advantage of the feedback provided by the CAT user to progressively improve the performance of the SMT system.

2.2 CAT and online learning

In order to design an adaptive SMT module for interactive machine translation (IMT) we first have to specify a reference model of human-machine interaction for CAT. This in turn requires a basic description of the main element of a modern CAT system: the translation memory.

A translation memory (see, e.g., [3]) consists of text segments in a source language and their translations into one or more target languages. These segments can be blocks, paragraphs, sentences, or phrases. A translator first supplies a source text. The program then analyzes the text, trying to find segment pairs in its translation memory where the text in the new source segment matches the text in the source segment in a previously translated segment pair. The program then presents any matches it finds as translation proposals to the translator for review. The translator can accept the proposal, reject it, or make modifications and use the modified version. In this case, the modified version is recorded and saved again in the translation memory.

The translation memories used in our application (see Section 3.3) are based on partial and/or fuzzy matches of sentences. That is, the memory returns to the user a sentence pair whenever the match between the source part of the pair and the current source sentence goes above a certain threshold. In practice, the translation memory software might use several proprietary criteria to measure the degree of matching between two sentences.

In this context, the SMT system can be used to generate suggestions for the cases when the translation memory does not have a good enough match for the source

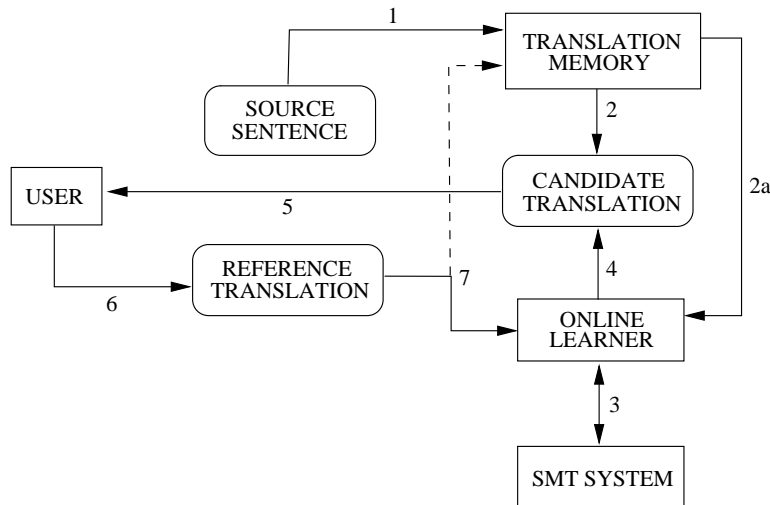


Figure 2.1: A high-level description of the adaptive CAT system information flow. A source sentence is sent to the translation memory (1). If a match is found (2), then the corresponding candidate translation is sent back to the user. Otherwise (2a), the source sentence is passed on to the online learner. The learner interacts with the SMT system (3) and obtains a candidate translation (4) which is sent to the user (5). In response, the user generates a reference translation (6) that is sent to both the translation memory and to the learner (7).

sentence. The system’s suggestion is then displayed to the user, who possibly performs post-editing actions in order to obtain the desired output, which we call the *reference target sentence*. The pair (source sentence, reference sentence) can be used to incrementally train the SMT system. The goal of this training is to improve the translation quality of source sentences that are related to the ones previously observed (we cannot hope to translate a sentence whose most words occur for the first time), but not similar enough to give a match in the translation memory.

We summarize this interaction in Figure 2.1. The dashed line connecting the reference to the translation memory reflects the common practice of adding new entries to a translation memory only after a further revision performed by a different translator. This is experienced by our system only as a higher number of memory faults, which may occur even on sentences that were recently



observed.

In machine learning several models of interaction between learner (the adaptive module in our application) and environment (in our case, the system formed by the user, the translation memory, and the sequence of source sentences to be translated) are considered. The online learning protocol (see, e.g., [9]) fits well with our CAT scenario. In this protocol the elements of an unknown sequence $(x_1, y_1), (x_2, y_2), \dots$ of object pairs are revealed to the learner one by one. At each time $t = 1, 2, \dots$ the learner observes x_t and must output a prediction for the associated element y_t . Once this prediction is generated the actual y_t is revealed, which the learner can use to refine the next predictions. This can be summarized with the following loop, for $t = 1, 2, \dots$

1. The environment generates an instance x_t
2. The learner generates a prediction \hat{y}_t
3. The environment reveals the element y_t associated with x_t
4. The learner uses the pair (x_t, y_t) to refine its predictive rule.

In our CAT scenario, instances x_t are source sentences and their associated elements y_t (which we also call labels) are the reference translations.

If we specialize the above schema to CAT, also including the translation memory in the loop, we obtain the following.

1. The next source sentence x_t is read.
2. If there is a (possibly partial) match (x'_t, y'_t) in the translation memory, then y'_t is used as tentative translation for x_t .
3. Otherwise, x_t is fed to the learning module that generates a tentative translation \hat{y}_t .
4. The user observes either y'_t or \hat{y}_t and performs post-editing operations resulting in a reference translation y_t .
5. The pair (x_t, y_t) is returned to both the translation memory and to the learning module.

The idea pursued in this chapter is to feed each returned pair (x_t, y_t) to an online learning algorithm that uses this pair to perform incremental adjustments of the SMT module parameters. The overall goal is to obtain a consistent increase in



the translation quality as more feedback is generated by the user of the CAT system. This increase in the translation quality should in turn correspond to a reduction of our main performance criterion: the average time the CAT user needs to translate a certain number of source sentences.

2.3 Dynamic re-ranking

A simple way to design online learning algorithms is to consider predictions that are linear functions of their inputs. In order to do that, we need to represent sentences as vectors of real numbers. We do that via features f_i ; that is, functions that map sentence pairs (x, y) into numbers $f_i(x, y)$, where x is a source sentence and y is a target sentence (which we view as a candidate translation for x). These features measure relevant properties of the sentence pair. In our online learning architecture for CAT, we obtain a base set of features by constructing a map f_i for each line $i = 1, 2, \dots$ of the phrasetable. More features of this kind are described in Chapter 3. Before describing how these features are computed, we define how a linear prediction using the feature values is computed.

Let $\mathbf{f}(x, y) = (f_1(x, y), \dots, f_d(x, y))$ be the feature vector associated to a pair (x, y) of sentences. A linear online algorithm maintains a corresponding vector of real weights, $\mathbf{w} = (w_1, \dots, w_d)$, where w_i is the weight associated with the feature f_i . The linear prediction is then based on the inner product $\mathbf{w}^\top \mathbf{f}(x, y)$ as follows.

The online algorithm uses the stream $(x_1, y_1), (x_2, y_2), \dots$ of source sentences and reference translations fed back by the user to incrementally adjust its weight vector. For each source sentence x_t we introduce the set $\text{GEN}(x_t)$ of candidate translations for x_t . The online algorithm relies on this function for computing its translation of x_t , which is defined by

$$\hat{y}_t = \operatorname{argmax}_{y \in \text{GEN}(x_t)} \mathbf{w}_{t-1}^\top \mathbf{f}(x_t, y) \quad (2.1)$$

where \mathbf{w}_{t-1} is the weight vector learnt by the algorithm after observing the first $t - 1$ pairs.

Following the *dynamic decoding* approach of [25], we use the SMT decoder to implement (in an approximate way) the argmax operator. That is, we couple online learner and decoder in such a way that, for the current source sentence x_t , the score the decoder gets when using the i -th phrasetable entry is $w_{i,t-1} f_i(x_t, y_t)$. Thus, if we assume that $f_i(x_t, y) > 0$ if the i -th phrasetable



entry is used by the decoder when x_t is translated with y , and $f_i(x_t, y) = 0$ otherwise, then the total score of the pair (x_t, y) is exactly $\mathbf{w}_{t-1}^\top \mathbf{f}(x_t, y)$. Since the decoder orders candidate translations y approximately in decreasing order with respect to their total score, we see that the online algorithm obtains the translation \hat{y}_t simply by choosing the 1-best translation of the decoder. Note that in this setup $\text{GEN}(x)$ is the set of all possible translations of the source x given the decoder’s phrasetable content.

After obtaining \hat{y}_t from the decoder, the online algorithm performs the *weight update* $\mathbf{w}_{t-1} \rightarrow \mathbf{w}_t$. In principle, this update could use the information provided by the decoder’s ordering of $\text{GEN}(x_t)$. In practice, we only use the decoder’s N -best list, denoted by $\text{NBEST}(x_t)$, for a suitable value of N . This is typically a much smaller set than $\text{GEN}(x_t)$.

From the decoder’s viewpoint, the changes performed by the online algorithm on the weight vector \mathbf{w}_t affect the selection of the phrasetable entries so that certain entries tend to be preferred over others. In this respect, choosing phrasetable entries as base features is a natural thing to do, as they are the basic building blocks that the decoder uses to cover the source sentence. Clearly, we could have chosen many other different features (lexical, part-of-speech, positional, etc.) —see, e.g., [29, 25, 2] for some examples.

It is worth mentioning now that we distinguish two basic modes of operation for the decoder. In the first mode, where there is no online adaptivity, the decoding process is ruled by the *base decoder features*, such as the (forward and backward) translation model log-probabilities, the language model log-probabilities, the distortion penalties, and so on. The decoder computes the score of a translation by combining the values of the base features with a set of coefficients that are tuned on a development set (using heuristics like Minimum Error Rate Training [28]). The role of the base feature coefficients is different from the one played by the weight vector \mathbf{w}_t . To see this consider the source sentence x_t and a candidate translation y formed by the decoder using the phrasetable entries i_1, \dots, i_m . Let g_1, \dots, g_n be the decoder’s base feature functions. Then the score of (x_t, y) is computed as follows

$$s(x_t, y) = \sum_{j=1}^m \left(\sum_{k=1}^n c_k g_k(i_j) \right) \quad (2.2)$$

where c_1, \dots, c_n are the base feature coefficients. Note that here we are only considering the part of the score that is contributed to by *local* features g_k . That is, features defined on phrases only.

In the second mode of operation, the “adaptive mode”, the decoding process is



ruled by the phrasetable features. In the above example, the score of (x_t, y) is now computed as follows

$$s_{\text{ADA}}(x_t, y) = \sum_{j=1}^m w_{i_j} f_{i_j}(x_t, y) \quad (2.3)$$

where, we recall it, f_i is the feature associated with the i -th phrasetable entry. As further discussed in Chapter 3, our CAT system is built in two phases: first, we build a phrasetable on a training corpus. Then, we train the base feature coefficients on a development corpus. After these two phases, in order to translate further sentences we have the option of running the system in nonadaptive mode (with the chosen set of tuned base feature coefficients) or adaptive mode (with dynamically changing phrasetable weights).

In order to ensure that when the system is started in adaptive mode, we immediately improve on the base performance level reached in nonadaptive mode, we use the base feature scores as values of our phrasetable features. That is, if the i_j -th phrasetable entry is used for the translation (x_t, y) , then

$$f_{i_j}(x_t, y) = \sum_{k=1}^n c_k g_k(i_j)$$

and we adjust (2.3) so that the score of (x_t, y) in adaptive mode is computed as

$$s_{\text{ADA}}(x_t, y) = \sum_{j=1}^m (1 + w_{i_j}) f_{i_j}(x_t, y) .$$

This ensures that at the beginning of the adaptive mode, when $w_i = 0$ for all entries i , the score of each pair (x_t, y) equals the score (2.2) computed in nonadaptive mode.

As we explained earlier, the decoder also uses nonlocal features that are evaluated on partial translations of the source sentence, such as the features based on the language model and on the distortion penalty. These nonlocal features influence the decoder even in adaptive mode; however, they are not accessed by the online algorithm which acts on local features only.

2.4 Online algorithms for dynamic re-ranking

In this section we introduce the linear online algorithms that are used by our dynamic re-ranking application. We first recall some notions and terminology defined in Deliverable D4.1.



As mentioned earlier, our goal is to increase the translation quality with the aim of reducing the time needed to translate. In order to take immediate advantage of the user feedback, our online algorithms learn at the sentence-level, and use smoothed variants of the BLEU metric (see, e.g., [25, footnote 5]) in order to measure the quality of each translated sentence.¹

As a remark, note that our learning algorithms are oblivious to the choice of the quality metric. In particular, they can work with any metric μ that, given a pair (y, y') of candidate translations, returns a nonnegative number $\mu(y, y')$ measuring the similarity of y and y' . In addition, we also need μ to be bounded, say $\mu < 1$, and that $\mu(y, y) = 1$ for every target language sentence y .

We define the *target sentence* y_t^* for the source x_t by

$$y_t^* = \operatorname{argmax}_{y \in \text{NBEST}(x_t)} \text{BLEU}(y_t, y) .$$

This definition is needed to manage the cases when $y_t \notin \text{NBEST}(x_t)$. That is, when the decoder fails to put the reference translation y_t among the N best translations. In general, the decoder may not even be able to generate y_t due to missing entries in the phrasetable.

Our general goal is to learn \mathbf{w} such that differences in BLEU are reflected by corresponding differences in margin values,

$$\mathbf{w}^\top \mathbf{f}(x_t, y_t^*) - \mathbf{w}^\top \mathbf{f}(x_t, y) \simeq \text{BLEU}(y_t, y_t^*) - \text{BLEU}(y_t, y) \quad \forall y \in \text{NBEST}(x_t) .$$

This approach has the obvious advantage of taking into account the information provided by BLEU metric, which is our proxy for translation quality. In practice, this should prevent any bad or not-so-good translation from appearing in the top positions of the final ranked list. To achieve this goal, we apply the *cost-sensitive* classification framework of [11, 32, 33]. For each (x_t, y_t) and $y \in \text{NBEST}(x_t)$ define

$$\begin{aligned} \ell_t(y) &= \text{BLEU}(y_t, y_t^*) - \text{BLEU}(y_t, y) \\ \phi_t(y) &= \mathbf{f}(x_t, y_t^*) - \mathbf{f}(x_t, y) . \end{aligned}$$

Then, for any x_t we would like to enforce $\mathbf{w}^\top \phi_t(y) \geq \ell_t(y)$ for all $y \in \text{NBEST}(x_t)$. Note that each target sentence $y \in \text{NBEST}(x_t)$ defines a constraint for \mathbf{w} . In what follows, we often identify sentences with constraints.

¹We are aware that this approach could lead to unsatisfactory results because BLEU and other metrics, even when they correlate well with human judgement at the document level, might not function well at a sentence level.



We can perform a sort of bias-variance decomposition of the overall translation quality,

$$\begin{aligned} \text{BLEU}(y_t, \hat{y}_t) &= \text{BLEU}(y_t, y_t^*) \\ &\quad + \text{BLEU}(y_t, \hat{y}_t) - \text{BLEU}(y_t, y_t^*) . \end{aligned}$$

The first term in the right-hand side is the (negative) “bias”, measuring the quality of the best translation in $\text{NBEST}(x_t)$, the so-called *oracle* BLEU. The second term is the (negative) “variance”, measuring how much worse is the system’s translation \hat{y}_t with respect to the oracle BLEU. Depending on the choice of the online algorithm, the evolution of these two terms may change: sometimes the algorithm is able to rank first y_t^* (improving on the second term), but the overall performance remains the same because $\text{BLEU}(y_t, y_t^*)$ gets worse (the N -best list does not contain good translations).

We now introduce the main algorithms that are used for the experiments of Chapter 3. All algorithms we consider are based on the following simple additive update rule $\mathbf{w}_t = \mathbf{w}_{t-1} + \mathbf{v}_t$ for different choices of \mathbf{v}_t . This update arises from the solution to the following optimization problem

$$\min_{\mathbf{w}} \left(\frac{1}{2} \|\mathbf{w} - \mathbf{w}_{t-1}\|^2 + C \max_{y \in \text{NBEST}(x_t)} H_t(\mathbf{w}, y) \right) \quad (2.4)$$

where $C > 0$ is a free parameter and $H_t(\mathbf{w}, y)$ is the *hinge loss* defined by

$$H_t(\mathbf{w}, y) = \left[\sqrt{\ell_t(y)} - \mathbf{w}^\top \phi_t(y) \right]_+ .$$

The problem (2.4) is equivalent to the minimization of

$$\min_{\mathbf{w}, \xi > 0} \left(\frac{1}{2} \|\mathbf{w} - \mathbf{w}_{t-1}\|^2 + C \xi^2 \right)$$

under the constraint $\mathbf{w}^\top \phi_t(y) \geq \sqrt{\ell_t(y)} - \xi$. The loss function ℓ appears under the square root because of the quadratic slack variable ξ^2 (see Deliverable D4.1 for details).

In choosing updates, we have to strike a good trade-off between efficacy and efficiency. The efficiency requirement comes mainly from the need of terminating the update before the user comes up with the next sentence to translate.

Passive-Aggressive

The simplest update we consider is Passive-Aggressive type II [11]. This update is only applied when the 1-best translation $\hat{y}_t = \operatorname{argmax}_{y \in \text{NBEST}(x_t)} \mathbf{w}_{t-1}^\top \mathbf{f}(x_t, y)$



violates the constraint $\mathbf{w}_{t-1}^\top \phi_t(\hat{y}_t) \geq \sqrt{\ell_t(\hat{y}_t)}$. That is, when $H_t(\mathbf{w}_{t-1}, \hat{y}_t) > 0$. The choice of \mathbf{v}_t is given by the closed-form solution of the following simplified version of (2.4)

$$\min_{\mathbf{w}} \left(\frac{1}{2} \|\mathbf{w} - \mathbf{w}_{t-1}\|^2 + CH_t(\mathbf{w}, \hat{y}_t) \right) .$$

This solution is $\mathbf{w}_t = \mathbf{w}_{t-1} + \mathbf{v}_t$ where $\mathbf{v}_t = \eta_t(\hat{y}_t) \phi(\hat{y}_t)$ and

$$\eta_t(\hat{y}_t) = \frac{H_t(\mathbf{w}_{t-1}, \hat{y}_t)}{\|\phi_t(\hat{y}_t)\|^2 + \frac{1}{C}} . \quad (2.5)$$

When $C \rightarrow \infty$, after the update the constraint is satisfied with equality. This corresponds to an Euclidean projection of the old weight onto the hyperplane $\{\mathbf{w} \in \mathbb{R}^d : \mathbf{w}^\top \phi_t = \sqrt{\ell_t(\hat{y}_t)}\}$. As this update only affects \mathbf{w}_{t-1} in the direction $\phi_t(\hat{y}_t)$ corresponding to the constraint associate with a single $y \in \text{NBEST}(x_t)$, we call this a *local update*.

Simultaneous projections

This is a simple example of *global update*. That is, when \mathbf{w}_{t-1} is moved in a direction which in general is not orthogonal to the hyperplane associated with any specific constraint. The update, proposed in [1], consists in moving \mathbf{w}_{t-1} in a direction computed as the average of the directions $\eta_t(y) \phi(y)$ for each violated constraint $y \in \text{NBEST}(x_t)$, where $\eta_t(y)$ is the learning rate of the PA-II update (2.5). Let V_t be the set of violated constraints at step t . That is, $V_t = \{y \in \text{NBEST}(x_t) : H_t(\mathbf{w}_{t-1}, y) > 0\}$. Then the update is defined by $\mathbf{w}_t = \mathbf{w}_{t-1} + \mathbf{v}_t$ where

$$\mathbf{v}_t = \frac{1}{|V_t|} \sum_{y \in V_t} \frac{H_t(\mathbf{w}_{t-1}, y)}{\|\phi_t(y)\|^2 + \frac{1}{C}} \phi_t(y) .$$

Gradient ascent

This is a slightly more complex global update performing gradient ascent on the convex dual of (2.4). Let $\alpha : \text{NBEST}(x_t) \rightarrow \mathbb{R}$ be a nonnegative function and let

$$\mathbf{v}_t(\alpha) = \sum_{y \in Y_t} \alpha(y) \phi_t(y) .$$



Parameters: Learning rate base constant $a > 0$; number K of iterations.

Initialization: $\alpha_0(y) = 0$ for all $y \in Y_t$.

For $t = 1, \dots, K$

1. **For each** $y \in Y_t$, let

$$\alpha'_t(y) = \alpha_{t-1}(y) + \frac{a}{t} \frac{\partial D_t(\alpha_{t-1})}{\partial \alpha(y)}$$

2. $\alpha_t =$ Euclidean projection of α'_t on $[0, \infty)^{|Y_t|}$

Output: α_K

Figure 2.2: Gradient ascent with Euclidean projection.

As shown in Deliverable D4.1, the dual of (2.4) is

$$D_t(\alpha) = -\frac{1}{2} \|\mathbf{v}_t(\alpha)\|^2 - \frac{1}{2C} \left(\sum_{y \in Y_t} \alpha(y) \right)^2 + \sum_{y \in Y_t} \alpha(y) H_t(\mathbf{w}_{t-1}, y)$$

where, in order to keep the notation tidy, we set $Y_t = \text{NBEST}(x_t)$. The variables $\alpha(y)$ for $y \in \text{NBEST}(x_t)$ are the Lagrange multipliers associated with the constraints $\mathbf{w}^\top \phi_t(y) \geq \sqrt{\ell_t(y)}$.

The partial derivative of D_t with respect to any $\alpha(y)$ is readily computed,

$$\frac{\partial D_t(\alpha)}{\partial \alpha(y)} = -\phi(y)^\top \mathbf{v}_t(\alpha) - \frac{1}{C} \sum_{y' \in Y_t} \alpha(y') + H_t(\mathbf{w}_{t-1}, y) .$$

In order to enforce the constraint $\alpha(y) \geq 0$ for $y \in Y_t$, after each gradient step we project back to the nonnegative orthant $[0, \infty)^{|Y_t|}$.

The final update is $\mathbf{w}_t = \mathbf{w}_{t-1} + \mathbf{v}(\alpha_K)$ where α_K is computed via K steps of gradient ascent (see Figure 2.2). The choice of the learning rate a/t ensures convergence to the maximum of D_t in time $1/\varepsilon$, where ε is the desired accuracy. This provided a is chosen not larger than the smallest nonzero eigenvalue of the Hessian of D_t —see [18] for details.

2.5 Some further issues and considerations

The successful application of online learning techniques to CAT requires the careful consideration of some peculiarities of the language translation domain.



In this section we list some of them and make some preliminary considerations. Further discussion is found in Chapter 3.

When translating a sequence of source sentences we expect some of these sentences to be much easier to translate than others. This entails a potentially large variance of the performance in a sequence of translations. In order to reduce this variance we use the averaging technique of [16], in which the weight vector used at time t to translate x_t is the average of all past vectors,

$$\bar{\mathbf{w}}_{t-1} = \frac{1}{t} \sum_{s=0}^{t-1} \mathbf{w}_s .$$

Note that this requires decoding twice at each time: once to obtain the system's prediction, which is the 1-best with respect to $\bar{\mathbf{w}}_{t-1}$, and once to obtain the N -best with respect to \mathbf{w}_{t-1} , which is then used to make the update $\mathbf{w}_{t-1} \rightarrow \mathbf{w}_t$.

Another aspect that is specific to this CAT context is the fact that the performance of the online learner is measured on a subset of the original sequence of source sentences. This is due to the filtering action of the translation memory, which intercepts all sources that are close enough to sentences seen previously. This implies that the actual subsequence observed by the learning module is made up of sentences that are typically dissimilar to each other. Thus, in a certain sense, the presence of the translation memory makes the learning task much harder.

A final issue concerns the potential advantage one could obtain by adding to the phrasetable segments originating from new sentences (translation model adaptation). This is clearly a good thing to do in order to reap the benefits of the online approach. However, there are serious technological problems related to the need of performing segmentation and alignment during the full translation process (rather than only when the phrasetable is first constructed from the training corpus).

Chapter 3

Integration with Portage

3.1 Introduction

Portage is a statistical machine translation system that has recently been made available to Canadian universities and research institutions under a distribution package named *Portage Shared*. See also [30].

Portage is based on the statistical approach to machine translation. This approach is becoming more popular thanks to the availability of large-size training corpora, which allow SMT to obtain generally better performances than traditional rule-based machine translation. The success of SMT systems, especially in applications such as computer-assisted translation, is however hampered by some shortcomings.

- 1 SMT systems do not easily manage cases when training and testing corpora come from different domains (domain adaptation is currently an area of active research).
- 2 There is hardly enough flexibility in the training process of most SMT systems in use today: in order to incorporate new knowledge a new training phase must be started from scratch.
- 3 Typically, there is not much room for interaction with the user. The human translator receives the translation and there is no straightforward mechanism to adjust the SMT system so that the same error is not repeated anymore.



In this chapter we describe the integration in Portage of an online learning module tailored for CAT applications. As described in Chapter 2 this module has been designed to address the abovementioned issues, providing a flexible environment in which interaction with the human translator and continuous training is possible. In managing the interaction with the pre-existing CAT tools, we pay special attention to the problem of exploiting the translations suggested by the adaptive module only when their quality is good enough.

Our learning module has been mainly written in C++ and developed under Linux. Before describing it in more detail, we mention the parts of the Portage architecture that are most relevant to our application.

Portage operates in four main phases.

- 1 Preprocessing of raw data into tokens, with translation suggestions for some words or phrases being generated by rules.
- 2 Decoding in order to produce one or more translation hypotheses, error-driven rescoring to choose the best final hypothesis.
- 3 Optional re-ranking: the decoder can produce a list of translation options (the N -best list). It is possible to make Portage learn how to re-rank this list, also using additional features.
- 4 Postprocessing to enforce appropriate case and punctuation.

Preprocessing involves different steps, mainly implemented by Perl scripts. In order to create a sentence-aligned parallel corpus it is necessary to split the source and target corpora into different files to allow for efficient processing. Each file is then tokenized (the words and punctuation are separated by spaces) using different heuristics for each language. The files are then aligned line by line so that the correspondence between source and target sentences is precise. Finally, files are lower-cased.

The phrasetable is learned from the sentence-aligned corpora using a word-level aligner coupled with some heuristics to gather entries for the phrasetable that are as correct as possible. The language model is trained using the SRILM toolkit from SRI (see [31]).

Decoding is a central phase in statistical machine translation. Almost all modern decoders are stack-based. This allows a good time/precision trade-off when searching the huge space of translations of a source sentence. When the decoding process starts there are no hypotheses in the stack. Then, translations covering just a single source word are added to the stack. All successive hypotheses are



generated by adding a word to the set of source words so far covered by the existing candidate translations. The first stack is used to generate a second stack with two-word hypotheses, and so on until we have a stack of candidates that fully cover the source sentence.

Since we need to evaluate hypotheses even if some of the words are not translated yet, we make use of a “future score”, which is an heuristic formula to estimate the cost of translating the remaining parts of the sentence.

During this search it is important to prune the data structures and to drive the search toward the best translation options. There is no guarantee that the best translation will be included in the output; however, there is a high probability that it will not be discarded.

For each preprocessed source sentence, N -best translations are identified using the search algorithm with a loglinear model. The system is open in many aspects. Indeed, it is possible:

- 1 to define new features to be used inside of the decoder, or in the re-ranking phase,
- 2 to define new parameters, which are loaded by the decoder,
- 3 to add language models or phrasatables.

Moreover, Portage allows one to define annotations in the corpora for describing rule-based translation for: numbers, dates, and domain specific named entities. These rules generate entries which are inserted into all phrasatables with a specified probability. This makes it possible to overcome obvious problems when translating those special entities based only on their statistics. Rule-based translations usually override automatically learned translations.

We will call the features defined inside the decoder *local features*, and the features for hypothesis re-ranking *global features*. The local features are evaluated for every translation option during the beam search decoding process. A local feature must yield a value for every segment (phrase) of the sentence.

There are constraints on how this value is calculated, otherwise the decoder would not be able to perform the beam search. The local features cannot reference the whole target sentence, because this is calculated during the decoding phases, when the full translation is not completely formed. Global features have fewer constraints: it is possible to calculate the value over the whole sentence. So, it is possible to move the most complex features outside of the decoder to overcome such problems.



When a sentence is translated incorrectly, a possible explanation is that the decoder does not have the segments needed to generate the reference sentence; alternatively, if the segments are present in the phrasetable, they could be scored so low by the decoder that the associated translation gets a low position in the ranking of candidate translations. The first case occurs less frequently given the availability of large training corpora. The second case, instead, calls for a better assignment of scores to segments.

In order to cope with this scoring problem, Portage allows for an optimization of the decoder weights. Every internal feature used by the decoder is associated with a weight. This weight is used along with the feature value to compute the score associated with a certain segment. Typically, these weights are optimized offline, after building the phrasetable, and then not updated anymore while the system is being used.

3.2 Integrating the online learner with Portage

As mentioned in Chapter 2, the goal of adding adaptivity to CAT is to improve the set of tools a professional translator uses in order to boost his productivity, and the purpose of this case study is to measure the improvement provided by online learning used jointly with SMT.

Dictionaries and translation memories are often used by professional translators. These tools store text segments (that is words, phrases, or entire paragraphs) and analyse the source text presented to the translator in order to find matches to previously translated text. The best matches are presented to the translator, who accepts or reject the proposals. Machine translation, in the context of this application, is used to generate suggestions for the cases when translation memory does not have a good match for the source sentence.

The process of building and training a learning system for CAT using SMT technology has not been deeply investigated in the literature. It is not straightforward to understand what are the best quality measures for this kind of application. Simply speaking, we would like to have a system able to help the human translator to recall previously used translations and give meaningful suggestions using the generalization ability of SMT techniques.

To complicate things, the translator, in the setting we are considering, is making use of a translation memory which will suggest a previously observed sentence when the confidence of this suggestion exceeds a given threshold. This fact means that the SMT system is going to receive the hardest sentences to trans-



late, and should be able to suggest (at least) the parts of the sentences already seen in previous translations.

When a sentence, or part of it, is almost unknown to the system, it is better not to emit any suggestion than to suggest a sentence which is probably completely wrong. So, it is important to calculate a confidence value for the translation before emitting a suggestion. To achieve the assessment of this value we must rely on the history of the sentence. For example if a sentence was never seen, but most of its bigrams have already been processed many times and they were always translated in the same way by the human translator, then it is highly likely that we can achieve a good confidence in the translation. On the other hand, if the system has seen a part of the sentence just once in all our “history”, then we could easily be wrong because that translation could be a mistake or it could have been found in a very different context when it was used to feed the learner.

To partially overcome this problem, different techniques to “smooth” the statistics of the language model are often used. Moreover, smoothing has been also proposed for the translation model [15]. In our adaptive architecture, smoothing can be viewed as the requirement of maintaining a correct trade-off between generalization capability, which is obtained by a suitable representation of all past sentences, and the need to adjust the system in order to fix the translation of the current sentence. We will sometimes call this trade-off *aggressiveness*, suggesting that an aggressive learner is stronger in modifying the parameters to quickly adapt to the new data, taking the risk of overfitting. This aggressiveness trade-off is controlled using different averaging techniques, as mentioned in Chapter 2.

3.2.1 Measuring performance for CAT

In such a framework it is difficult even to keep track of the performance of the learner because there is not a clear measure of effectiveness of the translation tools beside the judgement of the human translator. In particular, the key points we are interested to measure in a CAT tool are the following.

1. **Speed.** This is a key factor because in many cases there are time constraints. Moreover, post-editing is almost always needed to guarantee a good quality level before the translation is released. We are thus interested in giving to the user a translation proposal that minimizes the number of words changed or moved is minimized. It is also important to measure the speed improvement while the translation advances and the online learning



in the MT system receives more input from the translator.

2. **Quality:** A good translation quality means less post-editing work.
3. **Cost:** This is relevant to a translation agency as well, but it is not something we would like to measure because there are too many factors that affect it.

Our learning algorithms are very general: they can work with any sentence-level machine-computable measure of translation quality. In the learning phase we make use of BLEU score, WER (word error rate), and PER (position-independent word error rate). This is just an example of a large set of possible measures. Actually we could even use a mixture of different measures in the attempt to find a better correlation with the human judged quality. Since we are interested in the adaptation of the system to new data, to keep track of the relative improvement of the system when the learner receives more inputs we measure performance in terms of BLEU, WER, PER, and we compare it to Portage without adaptation.

3.2.2 Features

The Portage decoder makes use of a set of features to drive the beam search. Before the decoding process for a given source sentence starts all the possible ranges of words in the sentence are scanned, and a data structure holds the minimum cost needed to translate that segment. This is important in order to have the possibility of estimating the cost to translate the rest of the sentence given a certain translation hypothesis.

The PORTAGE decoder is based on a loglinear model that incorporates the following features

1. Phrase translation probabilities stored in one or more phrasables; these are to estimate the cost of using a target phrase to translate a source text.
2. One or more language models trained using the SRILM toolkit.
3. A distortion penalty penalizing a translation when the number of source words that are skipped when generating a new target phrase from the source is too high.
4. A word penalty, to avoid that long translations could be considered good just because of the addition of more scores.



The choice of features, as in every learning application, is very important because it is the way the learning module “sees” the problem. In our case we mean to add a huge number of features, from thousands to millions. Portage lets the user add new features to drive the search according to improved models or data. But, even if Portage is open, it is infeasible to add so many features using Portage standard data structures. We decided to make use of a special C++ class `SparseVector`, representing a sparse vector of features. This data structure is specialised to allow efficient storage, management, and computation over sparse vectors, where sparse means that most of the values stored are actually zeroes. Inside the `SparseVector` class we introduced also methods for reporting and debugging to check the integrity and correctness of the system in a core component. Since we were not allowed to introduce such a huge number of features inside the Portage decoder, we used a trick: all the features are calculated and their contribution is stored in a single Portage feature. So just a single new feature is added inside the decoder.

We also have to keep a different set of weights: during the decoding, when Portage asks for the score of our new feature, we return the sparse vector dot product of the feature evaluations with the sparse weight vector. As described in Chapter 2, the original set of loglinear Portage weights are optimized on the training set and then remain unchanged during the online training phase.

The phrasetable is controlled by the learning algorithm using a set of features that can be activated for just a single entry in the table or for a group of entries. The original decoder features used by Portage, on the other hand, are common for all the entries of the table, as you can see in Figure 3.1, where they cross vertically all the entries. This is an important difference: since we are dealing with online adaptation to user decisions and preferences, we judge that loglinear features are not discriminant enough for our purposes.

The new features for the fine tuning of the decoder are:

1. Target words
2. Target phrases
3. Single phrasetable entries
4. Single phrasetable entries + target words.

After introducing this huge set of features, we still want the decoder to behave the same way as before our intervention if there are no corrections for the current candidate sentence. At the same time we want the learning process to perform

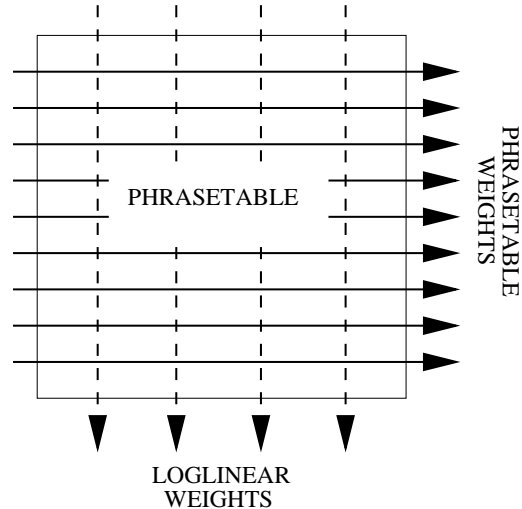


Figure 3.1: Relationship between old and new features: the value of both sets of features is computed for each phrasetable entry. However, the original Portage loglinear features are combined using weights that are common to all phrasetable entries. The new features, whose values is computed using the loglinear score, are instead combined using a specific weight for each phrasetable entry.

a smooth transition towards the new acquired translation examples. This is achieved using the technique described in Section 2.3

Since we deal with a large number of features, and the decoder needs to evaluate the value of the active features thousands of times, it is obvious that the performance in term of elapsed time could be really disappointing. To mitigate this problem we cache the value of the dot product of the features with weights, so that it is necessary to calculate them just the first time, or in the case when an associated weight is updated by the algorithm.

We now move on to describe what happens when a new sentence arrives. The data structures involved and their relationships are depicted in Figure 3.2.

1. The decoder is invoked using the initial weights. The decoder, during the normal process of pruning and ranking hypothesis, makes several calls to get the values of the new features.
2. When the decoder is finished we get the result: a list of N -best translation

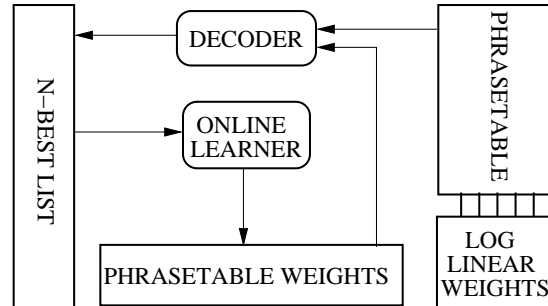


Figure 3.2: The decoder builds the N -best list using the phrasetable by combining the loglinear score (based on the original Portage loglinear weights) with the new score computed using the phrasetable features and their associated weights. Once the reference sentence is known, the online learner updates the phrasetable weights based on the ranking of the N -best list.

options (with N user definable, typically set around 200 – 1000).

3. Optionally, we run a static re-ranker, meaning that we change the relative scores of hypotheses in the N -best list. The importance of this step is related to the possibility of applying global features which are forbidden inside the decoder because of the beam search algorithm requirements.
4. We send the best translation option to the translator.
5. The translation option is stored with the feature vector calculated using our features.
6. If we receive a correction because the decoder did not provide a perfect translation we call an update function from our suite of algorithms.
7. If some of the words or segments of the sentence are unknown to the decoder, we run an on the fly alignment and store the new fragments (this step is still under development).

3.3 CAT tool architecture

The concrete CAT tool we chose for integration in our adaptive system is Trados from SDL International. Trados is a well-known product for translation memory management. It improves the speed and reliability of the translation



by fuzzy matching previously translated sentences. Thanks to this approach, our SMT tool is integrated seamlessly into an environment that is familiar to many professional translators.

We make use of Trados APIs to interact with external tools. Trados connects to an external web server where our system is running, thus providing centralized SMT translation facilities for all the translators involved in the project. The server sends the received translation requests to our modified version of Portage where the translation takes place. All the requests and timestamps are logged to a file. We use a standard webserver with CGI interface to a script. This script in turn interfaces the SMT system. The sentences are written as `http` requests using `post` methods. In the future, communications will be realized through standard webservice interfaces.

It is very important to make the work of the human user as easy and comfortable as possible, and minimize the differences from known tools and translation procedures. It is considered pointless, and even negative, to present a completely wrong translation option. So we plan to add modules to filter out bad translations based on confidence measures of the translation.

The workflow of the CAT interaction is as follows (see Figure 3.3).

1. A sentence is presented to the translator. Trados looks for a (fuzzy) match in the translation memory and outputs it if the match score between the two sentences goes above a certain threshold. Otherwise, the sentence is sent by our module inside Trados to the web server.
2. The web server sends the sentence to the modified version of Portage, which is waiting with all the necessary data preloaded.
3. The SMT system translates the sentence and then uses the features from the new data acquired to update the internal weights.
4. When translation and learning by the system is finished the results are sent back to the Trados API and presented to the user, allowing a simple select-and-click action to accept a candidate translation.
5. The user can obviously modify the proposal or reject it and insert manually the text.
6. A feedback is sent back to the web server to feed the SMT system with corrections.

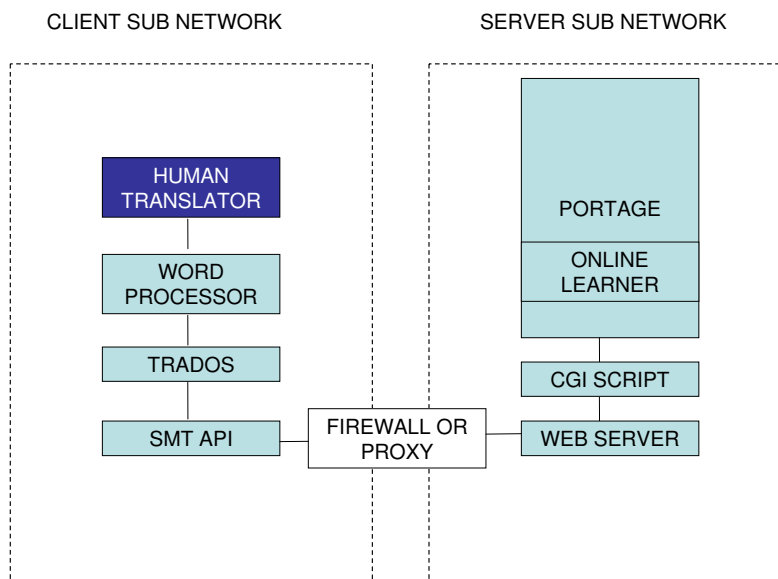


Figure 3.3: Software modules and network connections for the CAT evaluation task.

3.4 Experimental Results

We validated our approach via several experiments using mainly French-English and Spanish-English language pairs. The reason for this choice is to be able to easily compare our results with established baselines, and to prepare for the CAT evaluation task which will be held with professional Spanish translators from English to Spanish. The training data is from the freely available Europarl corpus and was obtained and prepared as described in [20].

After removing HTML tags, aligning the sentences and tokenizing, the Spanish corpus resulted in 1,476,106 sentences and 41,408,300 words. The French corpus had 1,487,459 sentences and 44,688,872 words. Finally, the English had 1,461,429 sentences and 39,618,240 words. To limit the possibility of errors during the preprocessing phases, particularly alignment errors, we decided to

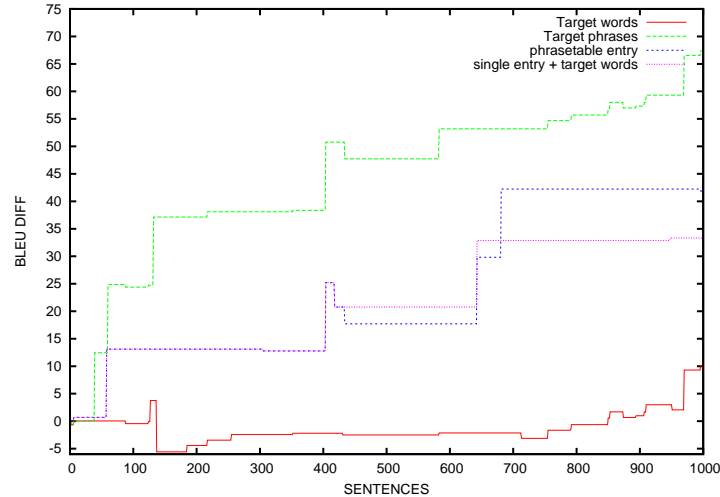


Figure 3.4: A comparison of our four feature sets: (1) One feature for every target word. (2) One feature for every target phrase. (3) One feature for every phrasetable entry. (4) A combination of 3 and 1. In this picture, and in the following ones, the cumulative BLEU differences exhibit long flat areas. This is because the feature sets are large, and the weight update caused by a sentence has no impact on subsequent sentences whenever these contain different words.

remove all sentences shorter than three words or longer than 25 words. It is important to point out that in SMT wrong phrasetable entries are not exceptional events. It is the statistics that helps (with high probability) to filter out the incorrect phrasetable entries. However, since a cleaner phrasetable speeds up learning, we took care of providing as clean as possible inputs to the system.

We split the corpora into a training set, a development set (10,000 sentences) and a test set (5,000 sentences). The development set was used to perform an optimization of the loglinear weights of the basic features of the decoder using a specific module of Portage named Cow. This is only a generic optimization to maximize the BLEU score over a dataset. In our case it was mainly used to adapt the decoder to the corpus in use. We did not use it during the online learning phase.

The training set corpus was further split in blocks of 300 sentences each. Each block was encoded in a format suitable for being submitted to the preprocessing stages of Portage. This splitting allows an efficient and parallel computation of

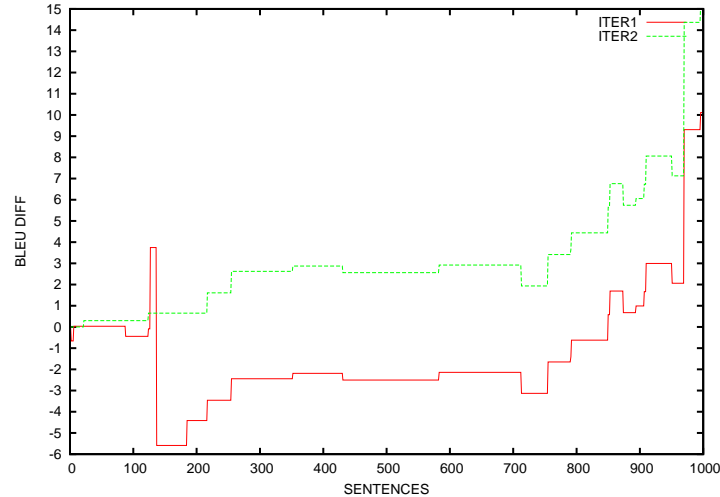


Figure 3.5: Results of two iterations with Passive-Aggressive type II algorithm using the target words feature. Cumulative differences from baseline Portage performance with no online adaptation

the phrasetable. The outcome of this phase, described earlier in detail, is the translation table and the language model.

We ran several experiments with the algorithms described in Section 2.4. However, we did not find that any specific algorithm had a significant advantage over the others. Therefore, we report experimental results only for the simplest of these algorithms; i.e., Passive-Aggressive type II.

In all experiments, we plot the performance measured in cumulative BLEUDIFF points. This is the difference in sentence-level BLEU points between the adaptive online system and the Portage baseline (no adaptation) summed over a sequence of translations.

In Figure 3.4 we measure the cumulative BLEUDIFF for Passive-Aggressive type II using our four feature sets. The target words turn out to be worst choice. However, Figure 3.5 shows that even these weak features carry information useful for the translation. Indeed, they pick up some signal during a second iteration over the same dataset.

Since our online learner works only when the match score of the translation memory is low, we implemented a “translation memory simulator” based on

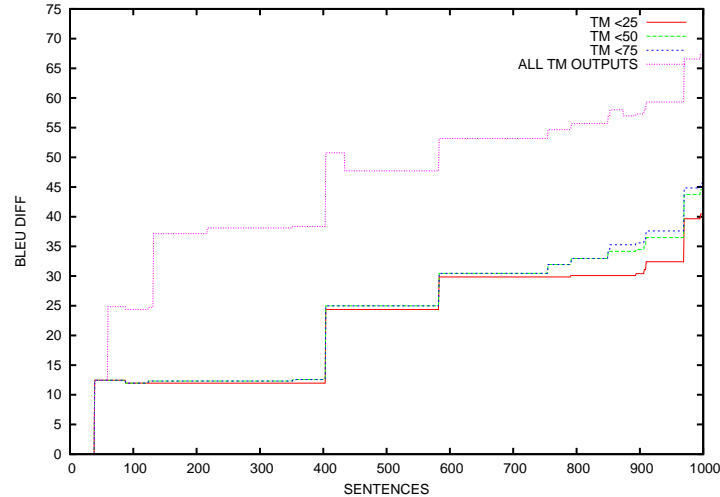


Figure 3.6: Four experiments with Passive-Aggressive type II with different settings for the translation memory match filter. For example, in the plot marked with $TM < 25$ we did not process sentences with match score higher than 25. The Translation Memory fuzzy match score is calculated with WER.

WER (Word Error Rate) and PER (Permutation Error Rate) to mark our translations with a match score and investigate how the performance of our algorithm changes for different choices of the score threshold. In Figure 3.6 we depict the system's performance on the data filtered by the translation memory.

3.5 Conclusions

We have presented a simple yet innovative modification of a state-of-the-art phrase-based SMT system. The impact on the numerical measure of the result is limited, possibly due to the fact that Portage has been optimized for some years, but the results in terms of flexibility and improved usability for the human interaction are interesting. Future work will be devoted to add an on-line alignment algorithm, so that it is possible to dynamically extend the phrasetable by including new text segments.

We are also working on the problem of calculating a confidence value (see for example [8]) for the translations produced by the system. This is an important



aspect of the evaluation phase because we need to avoid feeding the user with bad translations.

This learning application is especially complex since we are coping with the hard sentences: that is, those that caused a translation memory fault. So the system must really learn new ways of combining the segments in the phrasetable. In general, it is not easy to design features able to capture the intelligence of the translator in making use of certain phrases in a precise context. So we think it is important to extensively research new, possibly contextual features able to establish useful relationships between the text segment and its context.

Chapter 4

A large margin translation system

4.1 Introduction

In this chapter we describe a new machine translation approach based on large margin learning. Unlike the traditional SMT systems, in this new approach the score assigned to a pair of source and target phrases is computed using the solution of a one-class SVM problem with ∞ -norm regularization.

Similarly to the standard SMT approach, our system initially builds a phrasetable by training on a sentence-aligned corpus. This phrasetable is then used in the decoding phase to translate source sentences.

The training phase consists of the following steps: alignment and pruning. In the alignment phase, source sentences are split into phrases. For each source phrase, a large number of candidate target phrases are generated, each of which is a possible translation of the source phrase. In the pruning phase, different heuristics are used to cut down the number of target phrases associated with each source.

Once the phrasetable is built, the system is able to translate new sentences as follows. An input source sentence is first split into phrases, then a set of candidate target phrases is associated with each source phrase. Finally, a sequence of target phrases is chosen from the candidates so to ensure both a consistent coverage of the source sentence and grammatical fluency of the resulting target



sentence.

In the implementation of the above schema we need to bear in mind that the size of typical parallel training corpora is of the order of a million sentence pairs. In order to cope with this amount of data, we have to create a framework where the training phase can be implemented by an algorithm with complexity linear, or at most $\mathcal{O}(n \log n)$, in the number n of training sentence pairs. Online learning methods can be also applied here, provided the space and time requirements grow at most linearly in the number of observed pairs.

In what follow, we use *phrase* to denote a sequence of words that may contain gaps. The longest phrases considered have length 5, and any combination of gaps within a phrase are allowed.

4.2 The translation model

We assume a linear translation model similar to [10]. Let $\phi(x)$ and $\psi(y)$ be feature vector representations of source sentence x and target sentence y . The model is represented by a matrix \mathbf{W} , and target sentences y are scored based on the bilinear form $\psi(y)^\top \mathbf{W} \phi(x)$. Note that

$$\begin{aligned} \psi(y)^\top \mathbf{W} \phi(x) &= \langle \mathbf{W}, \phi(x) \psi(y)^\top \rangle \\ &= \text{TR}(\mathbf{W} \phi(x) \psi(y)^\top) \\ &= \text{VEC}(\mathbf{W})^\top \text{VEC}(\phi(x) \psi(y)^\top) \end{aligned}$$

where $\text{VEC}(A)$ is the columnwise vectorization of matrix A .

Thus this scoring method is equivalent to the one used in Chapter 2, $\mathbf{w}^\top \mathbf{f}(x, y)$, for a specific choice of $\mathbf{f}(x, y)$. Indeed, $\psi(y)^\top \mathbf{W} \phi(x) = \mathbf{w}^\top \mathbf{f}(x, y)$ when $\mathbf{f}(x, y) = \text{VEC}(\phi(x) \psi(y)^\top)$. In this chapter, however, we propose a method for choosing \mathbf{W} that is radically different from the online learning approach of Chapter 2.

4.3 Training phase

We now describe the training phase in more detail. We are given a training corpus of sentence pairs (x_t, y_t) , for $t = 1, \dots, n$, where x_t is the t -th source sentence and y_t is the corresponding reference translation. We assume each sentence is represented by a list of words.



We split source and target sentences into phrases. We understand a phrase as a consecutive sequence of words. Phrases can overlap with each other. The redundancy implied by the common parts of the phrases is exploited in the decoder to obtain a smooth translation. With every distinct source phrase we associate a list of target phrases that are possible translations of the source. This association is performed based on the co-occurrence of the phrases in sentence pairs as we explain below.

In the current version of the translation model we use the following heuristic, measuring how much the occurrence of a phrase in a source sentence is predictive with respect to the occurrence of a target phrase in the corresponding target sentence. More precisely, a source phrase p_s is predictive of a target phrase p_t if the occurrence of p_s in the source sentence implies with high probability the occurrence of p_t in the target sentence. To implement this simple predictor we consider the number of co-occurrences of p_s and p_t , denoted by $n_{s,t}$, and the total number of occurrences of p_s and p_t in the corpus, denoted by n_s and n_t respectively. We then evaluate each predictor $p_s \rightarrow p_t$ in terms of:

$$\begin{aligned} n_{st} & \text{ true positives} \\ n_s - n_{s,t} & \text{ false positives} \\ n_t - n_{s,t} & \text{ false negatives.} \end{aligned}$$

In particular, we measure the predictive power of $p_s \rightarrow p_t$ in terms of the F1 measure; that is, the harmonic mean of recall $n_{s,t}/n_t$ and precision $n_{s,t}/n_s$. Once we have the F1 scores for each pair p_s, p_t , we associate with p_s only those p_t such that the F1 score of $p_s \rightarrow p_t$ is above a certain threshold.

We can speed up this alignment procedure if, for each source phrase, the candidate target phrases are extracted only from a window around the estimated position of the source phrase translation. For the frequent phrases, the expected value of their position in the target is assumed to be proportional to the position in the source adjusted by the lengths of the source and the target sentences, namely:

$$\mathbb{E}[\text{position in target}] \sim \frac{\text{length of the target sentence}}{\text{length of the source sentence}} \times \text{position in source}$$

A window $\{-3, \dots, 4\}$ gives surprisingly good correspondence in English-French and in French-English translations. In the next version of the training phase we consider a more sophisticated windowing technique. It exploits the fact that the translation of frequent words is relatively good. Therefore, their correspondents are well allocated and this allows us to build alignments that are less order preserving.



Remark 1. *In the experiment, we tested other measures besides F1. We found that the F1 measure provides slightly, but not significantly, better results than other measures of dependency. In the future, we are going to extend the range of the possible measures.*

Ways of tackling infrequent phrases

For the rare phrases, the schema sketched above does not work very well because of lack of statistical mass.

We can exploit the following rule: every, or almost every, word must be translated. To this end a pairwise matching algorithm can be applied, which tries to force all phrases both in the source and in the target to be paired at least once. Falsely paired phrases are then likely to be discarded in the pruning phase, or during the translation phase (see Section 4.4). However, computational complexity is an issue here: general graph matching algorithms take time of the order of n^3 , which is too expensive for our application.

Another approach can be derived by exploiting the reverse translation when the target is used to predict the source. Translating the source $\phi(x)$ into a target $\psi' = \mathbf{W}\phi(x)$, and translating back $\phi' = \mathbf{W}^\top\psi'$, we get

$$\phi(x) \simeq \mathbf{W}^\top\psi(y) = \mathbf{W}^\top\mathbf{W}\phi(x) .$$

Since the scale does not matter, $\mathbf{W}^\top\mathbf{W}$ can differ from the identity, but \mathbf{W} should be close to an orthogonal matrix. Because the elements in \mathbf{W} are non-negative, this can be achieved if we force \mathbf{W} to be sparse. (in a sparse matrix the inner product of two different columns tends to be zero). Our approach follows the theory of sparse linear coding used in information theory, see details in [26].

4.4 Translation

The task of translating a new source sentence to a target sentence in a given target language can be decomposed into two fundamental subproblems:

- *Phrase prediction:* selection of the target phrases that are likely to occur in the translation.
- *Decoding:* merging of these phrases, or subsets of these phrases, into a sequence of words regarded as a true sentence with respect to some criterion.

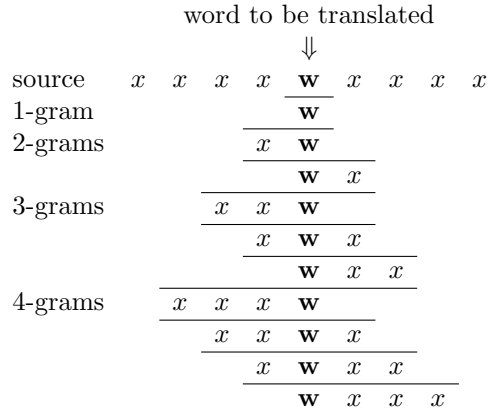


Figure 4.1: The neighbourhood structure of a word.

4.4.1 Phrase prediction

Our phrase prediction technique relies on the following conjecture: the translation of a word mostly depends only on those phrases that contain the word. We call the collection of these phrases the *neighbourhood* of the word. The structure of this neighbourhood is shown in Figure 4.1.

1-norm prediction

The words are translated via their neighbourhoods by a maximum margin learner that measures the margin using the 1-norm. We now describe the advantage of this learning framework. First we state the underlying learning problem, then we provide the solution which allows us to use a simple, almost trivial algorithm with small computational demand.

The building blocks of the model are the following: we have a sample of sentence pairs (x_t, y_t) for $t = 1, \dots, n$. The sentences are represented by vectors of indicator functions of phrases occurring in the sentences. So, both input $\phi(x_t)$ and output vectors $\psi(y_t)$ are very long but very sparse: only a small number of components are nonzero. This number can be estimated by the sentence length multiplied by a small constant (less than ten). Moreover, the nonzero items are all 1.



We consider the one-class classification problem with instances

$$\mathbf{z}_t = \text{VEC}(\phi(x_t)\psi(y_t)^\top) \geq \mathbf{0}$$

and we formulate the following maximum margin one-class problem where we use the 1-norm to measure instances \mathbf{z}_t and the dual ∞ -norm to regularize the solution $\mathbf{w} = \text{VEC}(\mathbf{W})$. Let $b \geq 0$ be a margin parameter.

Thus, we are looking for a hyperplane separating the sample items from the origin whilst the distance between the hyperplane and the origin is maximized. The distance between the hyperplane and the origin is measured as the minimum of the distances between the origin and the points sitting on the hyperplane. This problem is summarized in the following optimization problem, where the objective function is given by a subproblem computing the distance between the origin and the hyperplane,

$$\max_{\mathbf{w}} f(\mathbf{w}; b) = \left. \begin{array}{l} \text{sub-problem} \\ \left[\begin{array}{l} \min_{\mathbf{u}} \|\mathbf{u}\|_1 \\ \text{s.t. } \mathbf{w}^\top \mathbf{u} = b \end{array} \right] \end{array} \right\} \begin{array}{l} \text{closest point of the separating} \\ \text{hyperplane to the origin} \\ \text{in 1-norm} \end{array}$$

$$\text{s.t. } \mathbf{z}_t^\top \mathbf{w} \geq b, \quad t = 1, \dots, n$$

$$\mathbf{z}_t \geq \mathbf{0}$$

Here $b > 0$ is a free parameter. The solution of the subproblem for \mathbf{u} is

$$\mathbf{u}^* = \frac{b}{\|\mathbf{w}\|_\infty}$$

whereas the solution of the main problem is

$$\mathbf{w}^* = \frac{b}{\min_t \|\mathbf{z}_t\|_1} \mathbf{1}.$$

Hence, all components of \mathbf{w} are equal to a constant given by the item with the smallest 1-norm.

In the sparse case, when there are some common components of the vectors \mathbf{z}_t with zero value, the solution should be adjusted, since the corresponding component of \mathbf{w}^* is not determined by the constraints but only by the objective function. The ∞ -norm regularization gives then the following solution

$$(\mathbf{w}^*)_j = \begin{cases} \frac{b}{\min_t \|\mathbf{z}_t\|_1} & \text{if } \exists t = 1, \dots, n \text{ } (\mathbf{z}_t)_j > 0, \\ \left[-\frac{b}{\min_t \|\mathbf{z}_t\|_1}, \frac{b}{\min_t \|\mathbf{z}_t\|_1} \right] & \text{otherwise.} \end{cases}$$



Note that $(\mathbf{z}_t)_j = 0$ for all t means, in our translation problem, that there are no co-occurrences of the corresponding phrase pair.

We then predict the target phrases using the score $\mathbf{W}\phi(x_t)$. This prediction might be incorrect in scale, but the scale is not a meaningful factor when indicator functions are used as features, where all nonzero values are equal to the same constant. Hence, only their positions and not their values are meaningful. Thus, we can choose the particular solution minimising the number of nonzero components

$$(\mathbf{w}^*)_j = \begin{cases} 1 & \text{if } \exists t = 1, \dots, n \text{ } (\mathbf{z}_t)_j > 0 \\ 0 & \text{otherwise.} \end{cases}$$

Remark 2. *The reason why we expect the 1-norm might be a good choice for the learning problem arises from knowledge about the distribution of words in a natural language text. This distribution is very “irregular” and can be described by the Zipf’s law (see, e.g., [24]). This kind of distributions has no expected value, and in strict sense they might not be distributions because their partition functions can diverge when the sample size increases. As a consequence, we can hardly give a valid meaning to the probabilities derived. Weights based on these probabilities might thus be able to express relationships relative to a fixed sample, yet fail to generalize them to other samples. In the L_1 space, instead, we only exploit the coexistence of phrases, which remains valid independently of how the sample has been chosen.*

The phrase prediction process is described in Figure 4.2. Because a prediction is a set of candidate phrases that should appear in the target sentence, the scores given by the corresponding components of the vector $\mathbf{W}\phi(x)$ have to be transformed into a form that can be fed to the sentence decoder. To achieve this, a bag-of-candidates structure is built upon the scores. This structure is used to assign a set of phrases to each word of the source sentence using the highest scores received from the phrase predictor.

	Words				
Source sentence	w_1^s	w_2^s	\dots	w_{N-1}^s	w_N^s
	\Downarrow	\Downarrow		\Downarrow	\Downarrow
	Bags				
	$\mathfrak{p}_{1,1}$	$\mathfrak{p}_{2,1}$	\dots	$\mathfrak{p}_{N-1,1}$	$\mathfrak{p}_{N,1}$
High-score phrases	\vdots	\vdots	\vdots	\vdots	\vdots
	$\mathfrak{p}_{1,K}$	$\mathfrak{p}_{2,K}$	\dots	$\mathfrak{p}_{N-1,K}$	$\mathfrak{p}_{N,K}$
	$\mathfrak{p}_{1,K+1}$	$\mathfrak{p}_{2,K+1}$	\dots	$\mathfrak{p}_{N-1,K+1}$	$\mathfrak{p}_{N,K+1}$
Low-score phrases	\vdots	\vdots	\vdots	\vdots	\vdots

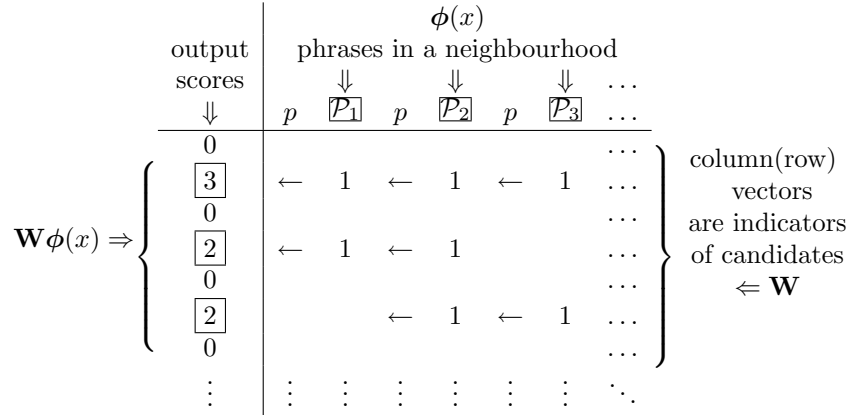


Figure 4.2: The phrase prediction process: high-scoring target phrases are fed to the decoder.

Here K is the maximum number of accepted candidates. If ties occur in scores, then the F1 measure is used to break them.

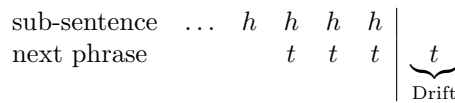
4.4.2 Decoding

The task of the decoder is to merge the phrases emitted by the phrase translator into a grammatical and fluent sentence. Before describing the sentence decoder we need to explain some basic elements of the procedure.

Merging phrases, drift, contradictions

A sentences is constructed by merging candidate phrases. The merging algorithm is based on the following notions.

- *Drift*: this is the tail of the next phrase that is not matched to the current subsentence. This is used to force the decoder to add at least one new word to the subsentence when a new candidate phrase is appended.





- *Distance*: the number of non-matched words between the current sub-sentence and the next phrase.

sub-sentence	...	a	Fitted part	Drift
# inequalities			b c d	∅
next phrase			= ≠ =	≠
			b q d	e
			↑	
			Contradiction	

Distance is measured by $\frac{\text{number of inequalities}}{\text{length of the next phrase}}$.

Decoding and reordering by dynamic programming with tabu search

The target sentence is constructed using a dynamic programming technique, see [4, 5, 6, 7]. In order to achieve a grammatically smooth sentence, the concept of Tabu-search based reordering [17] is added to the dynamic programming schema.

We use $\mathcal{B} = (\mathcal{B}_1, \dots, \mathcal{B}_N)$ to denote the ordered set of bags provided for each source word by the phrase predictor. Here N is the number of words in the source sentence for which we have a bag of candidate phrases. The order of \mathcal{B} reflects the order of the words in the source sentence.

We consider the following dynamic programming approximation of the word order in the target sentence:

- The state space is a set of labels of the phrases defined on the concatenation of the bags, in this way some phrases may appear in more than one bag. In order to distinguish between the states of different bags labelling the same phrases, the concatenation of the bag relative indices of the phrases is used to identify them:

$$\mathcal{S} = \left(\underbrace{(p_{1,1}, \dots, p_{1,n_1})}_{\mathcal{B}_1}, \dots, \underbrace{(p_{N,1}, \dots, p_{N,n_N})}_{\mathcal{B}_N} \right)$$

where the state $p_{i,j}$ as a label denotes phrase j in bag i .

- The sub-sentence built in the first t steps, with $p_{i,j}$ being the last phrase merged into, is denoted by $s(p_{i,j})_t$.



- The distance between a sub-sentence $s(p_{i,j})_t$ and a phrase $p_{k,l}$ in step t is denoted by $d(s(p_{i,j})_t, p_{k,l})$.
- The total length of a sub-sentence, denoted by $\ell(s(p_{i,j})_t)$, is measured by summing up the distances between the included phrases and their prefixes, assuming that the empty sub-sentence has 0 length.
- The tabu list $\mathcal{M}(p_{i,j})_t$ is a set of phrases used to build $s(p_{i,j})_t$.
- The merging operator acting on a sub-sentence and a phrase is denoted by \uplus .

The dynamic programming algorithm looks for the shortest sentence that contains the smallest number of different words in the overlapping parts of the sub-sentences and phrases merged.

The algorithm works in a forward fashion using the following update rule. Let $p_{r,s}$ be the state at the beginning of step $t + 1$. Choose the state corresponding to the optimum solution to the shortest sub-sentence problem,

$$p_{k,l} = \operatorname{argmin}_{p_{i,j}} \ell(s(p_{i,j})_t) + d(s(p_{i,j})_t, p_{r,s})$$

$$\text{subject to } p_{r,s} \notin \mathcal{M}(p_{i,j})_t .$$

Then merge the phrase $p_{r,s}$ into the sub-sentence belonging to the optimum state $p_{k,l}$ and append it to the corresponding tabu-list

$$\begin{aligned} s(p_{r,s})_{t+1} &= s(p_{k,l})_t \uplus p_{r,s} \\ \mathcal{M}(p_{r,s})_{t+1} &= \mathcal{M}(p_{k,l})_t \cup \{p_{r,s}\} . \end{aligned}$$

If $t = 0$ then the subsentence is initialised to the empty string and the corresponding tabu list is the empty set.

4.5 Experiment

In this final section we describe the results of an experiment in which we tested our translation system on the French and English versions of the Xerox printers manual provided by XRCE. The direction of the translation was from French to English. The baseline translator, which we used against our system, was the GIZA++ [27] aligner with the Pharaoh [21] decoder. Three random subsets of sentences were used as test sets. Two of them only included sentences with



sentence length	Method	ngrams recall				Bleu score
		1	2	3	4	
≤ 30	GIZA+PHARAOH	65.6	41.9	29.6	21.5	36.2
	TR candidate phrases	88.3	60.5	40.6	28.9	50.1
	TR drift, no contra	73.9	45.7	30.7	22.0	38.9
	TR drift, contra	66.1	41.5	29.2	22.0	36.4
≤ 30	GIZA+PHARAOH	70.2	47.5	35.6	27.4	42.0
	TR candidate phrases	90.6	64.1	43.9	30.8	52.9
	TR drift, no contra	75.6	49.1	33.9	24.8	42.1
	TR drift, contra	70.7	45.2	31.7	23.5	39.3
no limit	GIZA+PHARAOH	69.0	43.8	29.8	20.9	36.7
	TR candidate phrases	89.6	62.2	42.8	31.1	52.2
	TR drift, no contra	74.5	44.1	28.7	19.8	37.0
	TR drift, contra	68.9	41.3	27.6	19.7	35.3

Figure 4.3: Result produced on the Xerox printer manual dataset. Sentences occurring both in test and training sets are deleted from test set.

length at most 30, whereas one test set contained sentences with no length limitation. The quality of the translation is measured BLEU points. The BLEU score was computed by the algorithm in the Moses package.

The explanation of the items in Figure 4.3 is the following.

TR candidate phrases. This is the concatenation of the phrases provided by the phrase predictor. It shows how much information is contained in the candidate phrases and can be interpreted as a measure of n -gram recall among the candidates; obviously, the corresponding precision is inaccurate.

TR drift, no contra. The sentences are decoded under the assumption that phrases are used to extend the sentence if they perfectly fit the prefix sub-sentence, it overestimates the sentence length by approximately 40-50%.

TR drift, contra. In case of contradiction between the merged phrases and the prefix sub-sentence the word with higher score provided by the phrase prediction is inserted.



French source	English reference	Translation
Config : contient les fichiers de configuration des services de numérisation réseau.	Config contains configuration files for the Network Scanning Services setup.	contains the system configuration files the network scanning services
Contexte NDS (le cas échéant)	NDS Context (if applicable)	nds tree if available select
Une fois que le compte utilisateur du Document Centre a été créé, vous pouvez installer les services de numérisation sur un poste de travail.	Once you have your Document Centre logon account, you are ready to install Network Scanning Services on a workstation.	you have your document centre user account password is if a document centre has been created you can install network scanning services on a workstation
Étape 4 : Tester la configuration du Document Centre	Part 4: Test the Document Centre Configuration	the part 4 test the document centre configuration
3 Cliquez sur Terminer pour enregistrer les modifications et quitter l'assistant.	3 Click Finish to save your changes and close the wizard.	3 click finish to save the changes and exit the wizard and
Le menu déroulant Gestionnaire d'impression comprend les options suivantes	The Printer Manager Options menu contains the following options	pull down menu the printer manager window menu contains the following options

Table 4.1: Translation examples



4.5.1 Examples

Table 4.1 consists of some sentences translated from French into English taken from the Xerox Guide. The last row, “Le menu”, shows a side effect of the rare words. These words cause very occasional phrases which may repeat some words, here the “menu”, which is inserted twice. Note that this guide contains several incomplete sentences and words used in the technical jargon, e.g., “Config”.

4.6 Future work

The aim of this first implementation of our translation system was to collect experimental results about its main building blocks: the 1-norm based phrase predictor and the decoder. The decoder, built upon the combination of dynamic programming and tabu search, provides good results even when the quality of the inputs is not very accurate. Namely, the phrasetable contains several irrelevant phrase pairs, but it also includes some good candidates. The weak point is the translation of the rare words and phrases, when good candidates are generally missing.

A further task that we need to address concerns the size of the database of co-occurrences. The long consecutive phrases, consisting of three or more words, are important in building smooth translations, but most of them are occasional co-occurrences of otherwise not related words, causing only noise in the translation procedure. To this end we are developing a phrase generator which can consider non-consecutive word sequences too.

We are going to incorporate a language model into the phrase selection in which words are joint into a phrase if they can predict the occurrences of each other. In order to implement this model we plan to use a similar model that has been employed in the training phase of the first version of our translation system.

Bibliography

- [1] Y. Amit, S. Shalev-Shwartz, and Y. Singer. Online classification for complex problems using simultaneous projections. *Advances in Neural Information Processing Systems*, 18, 2006.
- [2] A. Arun and P. Koehn. Online learning methods for discriminative training of phrase based statistical machine translation. In *MT Summit XI*, 2007.
- [3] Various authors. Translation memory. *Wikipedia*, 2008. URL: en.wikipedia.org/wiki/Translation_memory.
- [4] R. Bellman. *Dynamic Programming*. Princeton University Press, 1957. Dover paperback edition (2003).
- [5] D.P. Bertsekas. *Neuro-Dynamic Programming*. Athena Scientific, 1996.
- [6] D.P. Bertsekas. *Dynamic Programming and Optimal Control*, volume I. Athena Scientific, 3rd edition, 2005.
- [7] D.P. Bertsekas. *Dynamic Programming and Optimal Control*, volume II. Athena Scientific, 3rd edition, 2007.
- [8] J. Blatz, E. Fitzgerald, G. Foster, S. Gandrabur, C. Goutte, A. Kulesza, A. Sanchis, and N. Ueffing. Confidence estimation for machine translation. *Proceedings of the 20th international conference on Computational Linguistics*, 2004.
- [9] N. Cesa-Bianchi and G. Lugosi. *Prediction, Learning, and Games*. Cambridge University Press, 2006.
- [10] C. Cortes, M. Mohri, and J. Weston. A general regression technique for learning transductions. *Proceedings of the 22nd International Conference on Machine Learning*, pages 153–160, 2005.



- [11] K. Crammer, O. Dekel, J. Keshet, S. Shalev-Shwartz, and Y. Singer. Online Passive-Aggressive Algorithms. *The Journal of Machine Learning Research*, 7:551–585, 2006.
- [12] J. Esteban, J. Lorenzo, S. Antonio, and G. Lapalme. TransType2—An Innovative Computer-Assisted Translation System. *The Companion Volume to the Proc. of the 42nd Annual Meeting of the Association for Computational Linguistics*, 2004.
- [13] G. Foster, P. Isabelle, and P. Plamondon. Target-Text Mediated Interactive Machine Translation. *Machine Translation*, 12(1):175–194, 1997.
- [14] G. Foster, P. Isabelle, and P. Plamondon. Target-text mediated interactive machine translation. *Machine Translation*, 12(1–2):175–194, 1997.
- [15] G. Foster, R. Kuhn, and H. Johnson. Phrasetable Smoothing for Statistical Machine Translation. *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2006.
- [16] Y. Freund and R.E. Schapire. Large Margin Classification Using the Perceptron Algorithm. *Machine Learning*, 37(3):277–296, 1999.
- [17] F. Glover and M. Laguna. *Tabu Search*. Kluwer, 1997.
- [18] E. Hazan, A. Agarwal, and S. Kale. Logarithmic regret algorithms for online convex optimization. *Machine Learning*, 69(2):169–192, 2007.
- [19] M. Kay. The MIND system. *Natural Language Processing*, 8, 1973.
- [20] P. Koehn. Europarl: A parallel corpus for statistical machine translation. *MT Summit X*, pages 12–16, 2005.
- [21] P. Köhn. Pharaoh: a beam search decoder for phrase-based statistical machine translation models. In *AMTA*, 2004.
- [22] P. Langlais, G. Foster, and G. Lapalme. TransType: a computer-aided translation typing system. In *Proceedings of Embedded Machine Translation Systems Workshop (in conjunction with NAACL/ANLP2000)*, pages 46–51, 2000.
- [23] P. Langlais and G. Lapalme. Trans Type: Development-Evaluation Cycles to Boost Translator’s Productivity. *Machine Translation*, 17(2):77–98, 2002.
- [24] W. Li. Random texts exhibit Zipf’s-law-like word frequency distribution. *IEEE Transactions on Information Theory*, 38(6):1842–1845, 1992.



- [25] P. Liang, A. Bouchard-Côté, D. Klein, and B. Taskar. An end-to-end discriminative approach to machine translation. *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the ACL*, pages 761–768, 2006.
- [26] D.J.C. Mackay. *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press, 2003.
- [27] F.J. Och and H. Ney. A systematic comparison of various statistical alignment models. *Computational Linguistics*, 29(1):19–51, 2003.
- [28] F.J. Och. Minimum error rate training in statistical machine translation. *Proc. of the 41th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 160–167, 2003.
- [29] F.J. Och, D. Gildea, S. Khudanpur, A. Sarkar, K. Yamada, A. Fraser, S. Kumar, L. Shen, D. Smith, K. Eng, et al. A smorgasbord of features for statistical machine translation. *Proceedings of HLT-NAACL 2004*, pages 161–168, 2004.
- [30] F. Sadat, H. Johnson, A. Agbago, G. Foster, R. Kuhn, J. Martin, and A. Tiku. Portage: A Phrase-based Machine Translation System. *Proceedings of the ACL Workshop on Building and Using Parallel Texts*, pages 129–132, 2005.
- [31] A. Stolcke. SRILM — an Extensible Language Modeling Toolkit. *Seventh International Conference on Spoken Language Processing*, 2002.
- [32] B. Taskar, C. Guestrin, and D. Koller. Max-margin Markov networks. *Advances in Neural Information Processing Systems*, 16:51, 2004.
- [33] I. Tsochantaridis, T. Joachims, T. Hofmann, and Y. Altun. Large Margin Methods for Structured and Interdependent Output Variables. *The Journal of Machine Learning Research*, 6:1453–1484, 2005.