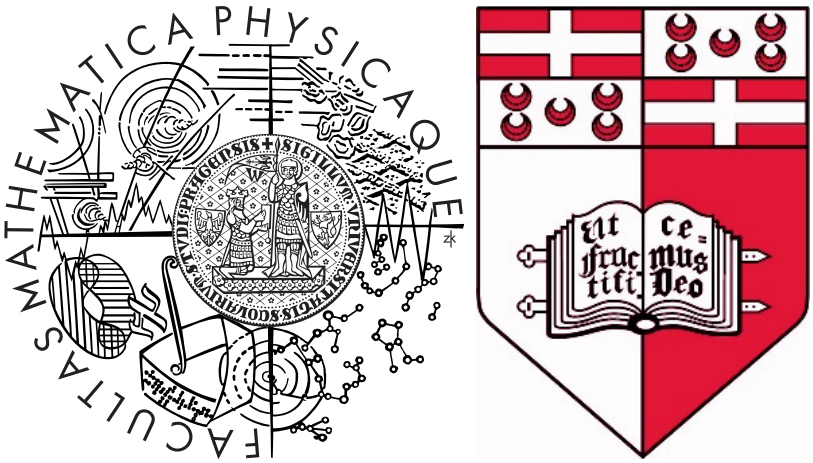
Charles University in Prague
Faculty of Mathematics and Physics

University of Malta
Faculty of Information and Communication Technology

MASTER THESIS



Miloš Stanojević

Large-Scale Discriminative Training for Machine Translation into Morphologically-Rich Languages

Supervisor of the master thesis: RNDr. Ondřej Bojar, Ph.D.
Mr Mike Rosner

Study programme: European Masters in Language and Communication Technologies

This thesis is dedicated to my parents Goca and Boža.

Acknowledgements

Foremost, I am deeply indebted to my supervisor Ondřej Bojar. His patience and detailed comments on my work made working on this thesis much easier and interesting.

I would also like to thank all the people who are involved in organization of LCT masters, who have made studying this masters program possible for me. I especially want to thank Mike Rosner, Marketa Lopatkova and Vladislav Kubon who helped me with all possible complications of studying masters at two different universities in two different countries.

I would not have so much fun during my studies if Annisa, Farina, Mariya, Amir, Bikash, Jo and Ke were not studying with me. I am very grateful to all of them for making me try beeepy coffee, swimming in the cold sea in December, eating very spicy Pakistani food, doing scuba and sky diving and having lots of geeky conversations. I would also like to thank you for helping me correct my writings, especially for adding omitted articles and removing them where they are the not needed.

I want to thank my parents Goca and Boža for believing in me and supporting me in everything.

I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular the fact that the Charles University in Prague has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 paragraph 1 of the Copyright Act.

In ..... date ..... signature of the author

Název práce: Rozsáhlé diskriminativní modely pro trénování strojového překladu do morfologicky bohatých jazyků
Autor: Miloš Stanojević
Katedra: Institute of Formal and Applied Linguistics, Faculty of Mathematics and Physics, Charles University in Prague, Czech Republic
Vedoucí diplomové práce: RNDr. Ondřej Bojar Ph.D.

Abstrakt: Diplomová práce se zabývá diskriminativními modely ve strojovém překladu do jazyků s bohatou morfologií. Shrnujeme současné přístupy a vypichujeme problém výběru slovních tvarů v cílovém jazyce a problém automatického odhadu kvality překladu jednotlivých vět. V našich pokusech s překladem z angličtiny do češtiny a srbštiny pak používáme morfologické i syntaktické rysy. Pro tento účel řešíme technické překážky, jak potřebné informace doručit k diskriminativnímu modelu: používáme jednoduchý tagging v průběhu překladu a promítáme zdrojové závislostní stromy na cílovou stranu.

Klíčová slova: diskriminativní modely, MIRA, řídké rysy, strojový překlad, vyhodnocování strojového překladu, ROUGE-S, projekce závislostní stromů

Title: Large-Scale Discriminative Training for Machine Translation into Morphologically-Rich Languages
Author: Miloš Stanojević
Department: Institute of Formal and Applied Linguistics, Faculty of Mathematics and Physics, Charles University in Prague, Czech Republic
Supervisors: RNDr. Ondřej Bojar Ph.D. and Mr Mike Rosner

Abstract: The thesis deals with large-scale discriminative models for machine translation into morphologically rich languages. We survey the approaches and highlight the problems of selecting word forms in the target language and evaluating translation quality at the sentence level. In our experiments with English-to-Czech and English-to-Serbian, we make use of morphological as well as syntactic features, solving necessary technical issues when bringing this information to the discriminative learner by tagging on the fly and projecting source dependency trees.

Keywords: discriminative training, MIRA, sparse features, machine translation, evaluation metrics, ROUGE-S, dependency tree mapping

Contents

Introduction

Machine translation is a subfield of Natural Language Processing which tries to do human-like translation from one natural language into other natural language. These languages can differ significantly on many different levels, one being especially elaborated on in this thesis – translation from morphologically poor into a morphologically rich language. Different approaches have been tried so far in solving these problems. Some systems have many linguistic rules directly implemented in them, so they could handle some problems better than other systems. For example, rule based systems are in some cases better than other approaches as it would be the case when translating between language pairs that have big differences in word order. When targeting morphologically rich languages, such as Serbian and Czech, word order is relatively free therefore producing an acceptable one is not the hardest problem. So far, statistical machine translation systems performed better than rule based systems in the case of the languages with rich morphology like Czech. That is the reason we have chosen SMT approach for building translator with Czech as target language. We will use the most popular type of SMT system, phrase based SMT, which refers to translating entire phrases from a source language to the phrases of a target language. System built by using phrase based approach usually consists of two major components:

- a basic part with a generative model and a decoder which generates candidate translations with high probability (according to the generative model) of being correct translations and
- a discriminative model that chooses the correct translation from these candidates.

In the following chapters we will first introduce the basic generative component of this type of SMT systems, and after that we will look at the discriminative component. Upon the introduction of these basic components of phrase based SMT system the focus will be on concrete ideas applied in this thesis: different objective function and sparse features used to solve problems that exist in translation into morphologically rich languages. In the end, we will show our experimental results and give final conclusion.

1. Generative phrase-based machine translation

The translation of a source sentence f into a target language can be seen as a search for the translation e in the space E of all possible sentences in the target language such that e has the highest probability of being the translation of f. More formally, we are trying to find [26]:

$$\boldsymbol{e_{best}} = \underset{e \in E}{argmax} P(e|f) \tag{1.1}$$

This formula can further be simplified to:

$$\boldsymbol{e_{best}} = \underset{e \in E}{argmax} P(e|f) = \underset{e \in E}{argmax} \frac{P(f|e)P(e)}{P(f)} = \underset{e \in E}{argmax} P(f|e)P(e) \tag{1.2}$$

This formula allows us to replace the probability of the target sentence given the source sentence with two other probabilities called *translation model* and *language model*. The language model $P(e)$ is a component which models the probability of sentence e appearing in the target language and indirectly, by doing that, it also models the fluency of the output. The translation model is modeling $P(f|e)$, which is the reverse translation component. The translation and language models are usually trained and then used together during computation of $argmax$ in, so-called, decoder. The language and translation models do not have to be trained on the same corpus. Often, the language model is trained on a monolingual corpus that is much larger than the parallel corpus used for training the translation model.

1.1 Translation model

The translation model that is used in phrase based SMT decomposes the probability of translating sentences into the probability of translating phrases:

$$P(f|e) = \prod_{i=1}^{I} \Phi(\bar{f}_i, \bar{e}_i) d(a_i - b_{i-1}) \tag{1.3}$$

where:

- d is the so-called distortion model, which penalizes big reordering in translating phrases

- a_i is the starting position of a foreign phrase $\bar{f}_i$

- b_{i-1} is the end position of a target phrase $\bar{e}_{i-1}$

- Φ is giving probability of phrase $\bar{f}$ being translation of phrase $\bar{e}$

These probabilities can be learned directly from training data by using the expectation maximization algorithm, but usually it is done by first doing word alignment and then extracting phrases using some heuristics which gives better results than learning phrase translation probabilities directly [26].

1.2 Language model

There are different types of language models. Some are based on modeling the probability of the target sentence using syntactic information and some use much simpler methods. The most common type of language model is the n-gram language model that models probability of a sentence by the product of the probability of its words conditioned by their history:

$$P(\boldsymbol{e}) = \prod_{i=1}^{I} P(e_i|e_0 \dots e_{i-1}) \tag{1.4}$$

The computation of these probabilities can be simplified by using the Markov assumption that the probability of a word given its history can be approximated by the probability of that word using the most recent history. The probability of the sentence using the n-gram model where n is the order of the history is given in the following formula:

$$P(\boldsymbol{e}) = \prod_{i=1}^{I} P(e_i|e_{i-n} \dots e_{i-1}) \tag{1.5}$$

With using higher order n-grams, it becomes likely that some of the probabilities will be zero. Because of that, some smoothing is necessary. The most widely used smoothing method in today's state-of-the-art systems is Kneser-Ney smoothing [25].

1.3 Decoder

The decoder is used when searching for the best translation using given language and translation models. Since it is impossible to do a complete search of exponentially large space of possible translations, an approximated form of the search is needed. Most decoders are implemented as A* search with beam pruning. The search space can be represented in a graph (the search graph) where each node in that graph represents a hypothesis and contains all the information about translated words and their probability. The transition from one node to the next one is called *hypothesis expansion* which adds some additional translated words to the hypothesis and updates its probability. Decoding is done by doing most probable expansion of all currently observed hypotheses. All hypotheses are put into one priority queue often called stack. In machine translation and speech recognition community, A* is modified to use only fixed depth of the stack and it is often referred to as a stack decoding algorithm. The comparison between the hypotheses that have translated a different numbers of words is not suitable for choosing the next expansion because the hypothesis with a smaller number of translated words will be chosen in most cases as the more probable expansion. This is why stack decoding in machine translation is done with multiple stacks where every stack contains hypotheses with the same number of translated words.

2. Discriminative Training in Machine Translation

As we have seen in a generative approach, we decompose the process of translation into many smaller steps that we assume are independent. We do this so that we could solve them independently and then combine their results, thus creating a good translation. For example, we assume that translations of different phrases are independent of each other. Even though there is some small dependence introduced by using language model, it still does not eliminate our assumption completely. Independence assumptions that exist in generative models do not exist in discriminative models.

The second reason why discriminative models might be more useful is the support for a large number of features. In generative models, we are searching a space of possible translations. With each new feature that we add to these models, we add a new dimension for search which means that the dependence of search errors on the number of features is exponential. With discriminative models, we are not searching the whole space of possible translations. Instead of that, we usually have a finite set of possible translations and try to discriminate good translations from bad ones. That being the case, adding a new feature will not increase the search space (it will be the same as without that feature).

The problem with applying discriminative methods to machine translation is that we usually do not have a reasonably small finite set of possible translations. Many researchers have tried different ways to solve this problem. The most successful usage of discriminative methods in machine translation, so far, is by combining them with generative methods. First, we use a system trained in the generative fashion to produce a relatively small finite set of most probable translations (according to the generative model) and then we use the discriminative model with a larger number of features to make a better decision which translation is the best from the set of possible translations.

The process is shown in Figure 2.1. The finite set of probable translations that is given by a generative model is usually in the form of an n-best list of translations. The discriminative model that is used for choosing the best translation is usually called “reranker” because it takes a list of possible translations that are ranked by their probability from the generative model and then reranks them by probabilities from the discriminative model.

The process of training the discriminative model that is used by reranker is

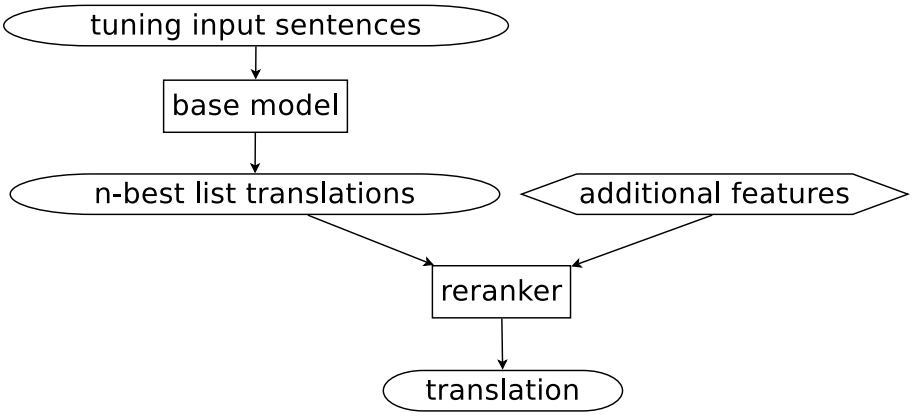


Figure 2.1: Usage of reranker [26]

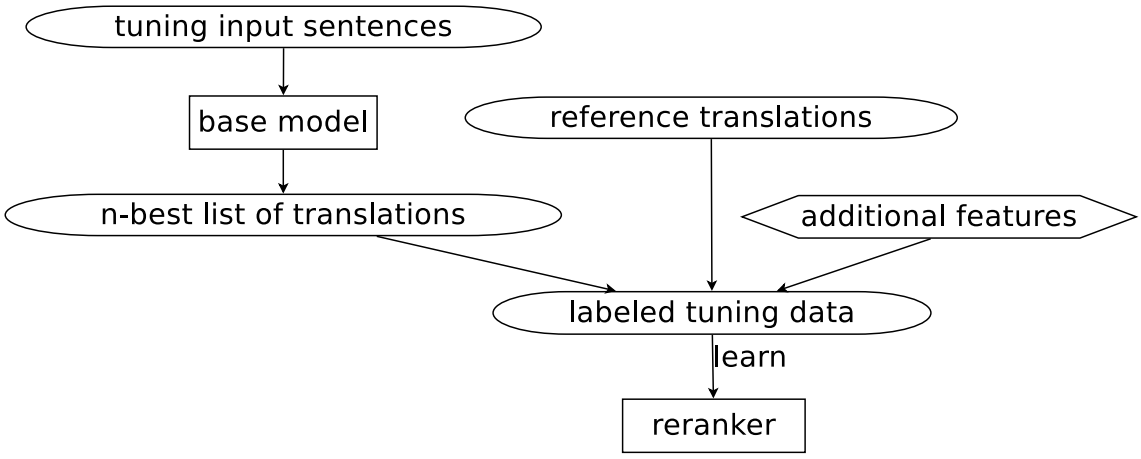


Figure 2.2: Training of reranker [26]

usually called tuning, because some of the features are used in the generative model too and we are trying to optimize their weights. Tuning is done by using an additional parallel corpus called *tuning* or *development corpus*. The tuning process is shown in Figure 2.2. The first step in the process is the translation of the source side of the corpus using the decoder for generative model. As the result of decoding, we get an n-best list of translations (judged using generative model) which is passed on to the learning algorithm together with a reference translation (the target side of the tuning corpus) and additional features. Learning algorithm can learn not only weights for discriminative, but also weights for generative model. With these new weights, generative model can explore different part of search space where it is more likely to find a good translation. After that, learning algorithm can be applied again to improve model's weights and repeat this process until some convergence is achieved.

In the next section, we explain how to get the list of best candidates for reranking and how can we learn the parameters for the discriminative model used by reranker.

2.1 Representing the set of candidates

The whole process of decoding can be seen as a search graph that contains relations between different hypotheses that were expanded during the search. The search graph is a representation of search history, but what we want is a representation of end hypotheses that cover the whole source sentence. One of possible representations of the set of best hypotheses is a word lattice which is basically a weighted finite-state machine that is very similar to a search graph of phrase-based MT system. The main difference is that instead of having the probability of a hypothesis in the word graph, the search graph gives us the probability of transitioning from one hypothesis to another.

Another common way to represent best hypotheses is an *n-best list*. N-best list can be constructed using a simple algorithm that uses the fact that for each state in a weighted FSM there is one best path to the start state [26]. Compared to the best path through the graph, the second best path takes a detour from the best path. The detour for an internal state of the graph is defined as suboptimal transition from the start state to the state in the best path for which we take detour. Detours for each state in the FSM are already discovered during the search by recombination of hypotheses.

The algorithm is based on having two queues:

- a queue for the resulting n-best list
- a priority queue of detours where the priority function is the probability of a detour

The algorithm is as follows:

1. Take the best path, put detours for all its nodes in a priority queue of detours, and put the best path in the n-best list
2. Take the detour with the highest probability from the detours priority queue and remove it from the queue
3. The hypothesis containing that detour and continuing to the end with the rest of the previous best hypothesis is added to the n-best list
4. All detours from the previous detour are added to the priority queue of detours
5. If the size of the n-best list is not as desired, go to 2.

The larger the set of hypotheses that reranker has the access to, the higher the probability that it will be able to choose the good translation. The word lattice is a more compact representation of translation hypotheses than the list of best translations and that is why it allows access to a larger part of the space of possible translations. Rerankers that work with word lattices usually give better results than those that work with n-best lists. The problem with word lattice is that it is not always easy to create an algorithm that will work with word lattices compared to the algorithms for n-best lists. Therefore, n-best lists are still used more often than word lattices.

2.2 Reranker

As mentioned before, a reranker takes the n-best translations and gives the best one from it judging by some discriminative model. The name reranker is taken from machine learning field but it is slightly misleading [20]. What the reranker in machine translation does is not really reranking. The goal of the real reranker is to put every element in a list to its right place, judging by some criterion, while the goal of the reranker in machine translation is to put only the best translation to its right place (1st place) without considering other translations.

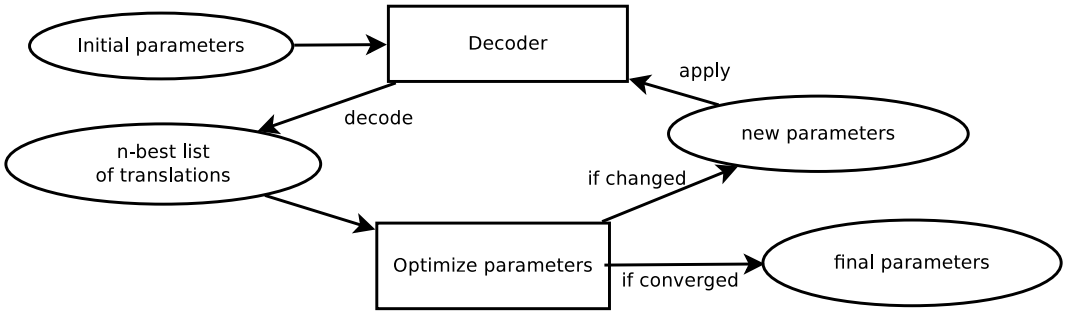


Figure 2.3: Learning parameters using multiple iterations [26]

So what the reranker does is scoring every hypothesis in the n-best list by using a certain model and then outputting the translation with the highest score. The model that is used is almost always the log-linear model which can have from a few to a few hundred thousands features. What the reranker actually does is described in the following formula [23]:

$$reranker(n\ best\ list) = \underset{e \in\ n\ best\ list}{argmax} \frac{\exp(\sum_{i=1}^{n} \lambda_i h_i(e))}{\sum_{e' \in\ n\ best\ list} \exp(\sum_{i=1}^{n} \lambda_i h_i(e'))}$$
$$= \underset{e \in\ n\ best\ list}{argmax} \sum_{i=1}^{n} \lambda_i h_i(e) \quad (2.1)$$

λ_i is the parameter or weight that tells us how much feature h_i is important. h_i can be any function of translation e. For example, one of the features can be the number of words in the translation e. Some more sophisticated features require not only translation e but also the derivation d of that translation. That is left out from this formula for the sake of clarity, but it will be discussed later again in the part of training a reranker with a large number of features.

The training of a reranker consists of finding the right set of λ parameters that maximize some objective function on the processes of translating the tuning corpus. The objective function that is used should be an automatic evaluation metric that correlates well with the human judgment. The most popular metric in machine translation community is BLEU [41] which is often used not only for evaluation but also for tuning. We leave the detailed explanation of evaluation metrics for Chapter 3, where we will deal with it in more detail. For now, the reader should just know that discriminative training algorithms require some evaluation function, which we will call EVAL, that takes some hypothesis and a human reference translation and returns the number that represents how close system's translation is to the human translation. Often, we will use the function COST which can be interpreted as 1-EVAL.

The general process of training a reranker with more iterations can be seen in Figure 2.3 which highlights the loop of decoding and re-estimation of parameters. We first translate a source side of a tuning corpus and get an n-best list. After that, we optimize the parameters of the model by making them such that the best translation in the n-best list, judging by the EVAL, is on the top of the list.

2.2.1 Optimization of reranker's parameters

In this chapter we are going to look at a few different algorithms for optimizing λ parameters of the discriminative model. Learning these parameters by using analytical methods (by computing derivatives and finding optima) is not possible, because mapping parameters to objective functions can be complex and expensive to compute [26]. Algorithms that are used for learning can be roughly separated in two groups:

1. those that are good at learning small set of parameters usually by applying some heuristics

2. those that are good at learning large set of parameters usually by maximizing the margin between good and bad translations

This thesis is concerned mostly with algorithms that deal with a large set of features, but we will briefly show one of the algorithms for small set of features that is used in most of the state of the art systems today. That algorithm is a version of Powell search which is in machine translation community most often referred to as MERT (Minimum Error Rate Training) [39]. It is also an important algorithm since many other algorithms that work with a large number of features, such as PRO, take the inspiration from MERT.

MERT is optimizing weights one by one and while it is optimizing one of the weights, other weights are fixed. Optimization is done in the form of a grid search. This optimization does not require computing derivatives, but it can be unstable with a large number of features [24]. By fixing all weights of all features except the one that is optimized at the moment, we assume that the best weight for the optimized feature will not harm other features. When we have a small number of features this risk is small, but with a large number of features, it is almost certain that this procedure will not work.

Large-Scale Discriminative Training Algorithms

All algorithms that are used in discriminative training try to minimize some loss function. We want the best translation according to our models score to also be

the one with the lowest cost or, more formally, loss function that we would want to minimize in the ideal situation is:

$$loss(x, Y, y_{ref}) = cost(y_{ref}, \underset{(y,h)\in Y(x)}{argmax}\ score(x, y, h; \lambda)) \tag{2.2}$$

where:

- x is the source sentence
- Y is the set of hypotheses for x
- h is a set of features
- λ is a set of learned feature weights
- y_{ref} is a reference translation
- y is a hypothesis sampled from Y together with its derivation (features) h

Good properties of MERT are that:

- it optimizes the loss directly without the need of computing the derivative (which can be hard or impossible)
- it can use corpus level metrics for cost function which is good given the fact that the most popular metrics in MT community are corpus level (BLEU, NIST, TER).

On the other hand, MERT also has some problems:

- it is very unstable because of complexity of loss and difficulty of search [20]
- it can work well only with a small set of features [24]

In order to overcome these problems, we will need to use some other algorithm. Machine learning community has developed large number of algorithms for reranking, but they are not directly applicable to the machine translation domain. The main reasons for that are the following [20]:

- these algorithms usually assume that a single correct result is available. This is hard to achieve in MT because:
  - The reference translation might not be reachable by our model.

  - Even if the reference translation is reachable, we might be getting the correct result by a wrong derivation (for example phrase alignment might be wrong) which can lead to optimizing wrong parameters.

- Latent variables are present in the process of machine translation. These latent variables make loss function non-convex and for this class of functions not much research has been done [20].

- Reranking algorithms in machine learning have a similar, though usually not completely the same, goal as reranking in machine translation - in machine translation, we try to reorder the translation with the lowest cost to the top of the list ranked by the model score and we do not care if second best translation ended at 10000th position in a sorted list, while machine learning algorithms try to put each translation in the right place [20].

To solve these problems many researchers have used different loss function which, in most general case, could be defined as :

$$loss(x, Y; \gamma_-^+, \beta_-^+) = - \max_{y^+ \in Y(x_i)} (\gamma^+ score(x_i, y^+) - \beta^+ cost(y_i, y^+)) + \max_{y^- \in Y(x_i)} (\gamma^- score(x_i, y^-) + \beta^- cost(y_i, y^-)) \quad [17,\ 20] \quad (2.3)$$

y^+ usually represents a translation toward which we want to optimize our model
y^- usually represents a bad translation which we want to get lower model score in the future
γ_-^+ and β_-^+ are parameters that influence the participation of *cost* and *score* in the loss
score function here represents score given by the discriminative model, not the generative one

Each part of this formula can significantly influence the process of learning:

1. the choice of the γ_-^+ and β_-^+ parameters can influence the strategy of optimizing the result. Different algorithms use different values for gamma and beta

2. the choice of y^+ and y^- from the space of possible translations $Y(x_i)$

3. the cost function needs to work well on the sentence level, which is not the case for many metrics, especially those that are popular in MT community like BLEU

4. the derivations of both hypotheses y^+ and y^-. For the sake of clarity, the derivations were not included in the formula, but they are important

In the next section, we explain some solutions to the problem of choosing y^+, y^-,γ_-^+ and β_-^+. After that, we look at the problems in machine translation that are caused by the presence of derivations. The large part of research of this thesis is concerned with solving the problem of cost function.

2.2.2 Selecting y^+ and y^-

Selecting y^+

Ideally we would like to use the reference translation y_i as y^+, but there are few problems in doing this. The first problem is that y_i might not be in the n-best list because it is ether:

- unreachable by our model or
- reachable but very improbable, given our generative model, to enter the n-best list

The second problem is that even if the reference translation is reachable by the given model it might not have the correct derivation (phrase segmentation, reordering etc.) and getting the correct surface form is just a fortunate coincidence. One more issue is that there are many correct translations possible but we have just one or a few references.

The solution to the first problem, that is applied in most cases in practice, is to select some surrogate translation with a low cost that will be used instead of the reference translation. The loss function that was defined before in the parameterized form with γ_-^+ and β_-^+ is called hinge loss, if the reference translation is used as y^+, but because we are using surrogate translation, this loss is usually referred to as ramp loss [20].

The other solution to this problem, which gave bad results [31], is to ignore cases when we cannot get the reference translation. The reasons why it fails are that we do not use the full amount of data because the reference is in many cases not reachable and even if it is reachable it might be with bad derivation (the second problem). This method is called "bold updating".

In the second problem, it is clear that we need to use the model *score* function in the selection of y^+. There are two ways that have been used so far for solving this problem:

- the more popular one is by choosing the translation that has the highest sum of negative cost and score. This hypothesis is often called "hope" because it is highly ranked in the n-best list and has a low cost [23]. This strategy would correspond to the following set of parameters in the given formula for the ramp loss:

  $\gamma^+ = 1 \; ; \; \beta^+ = 1$

- by taking the hypothesis from the n-best list with the lowest cost [31]. The score function is used here indirectly because the hypothesis with the lowest model score will not enter the n-best list and, therefore, could not be selected as a surrogate even if it had a low cost. The parameters for this strategy in the ramp loss formula are:

  $\gamma^+ = 0 \; ; \; \beta^+ = 1$ with the constraint that it can be applied only on an n-best list and not during the decoding

One more reason for using the score function in selecting the y^+ and y^- hypotheses is that we want to operate on translations that are most likely to be produced by our system. If we have a really good translation (low cost) with really low probability (low score) it might be a bad idea to optimize toward it because, even if we move that translation from, for example, 10000th place in the n-best list to the 2nd place, it still does not bring any improvement. By balancing the importance of cost and score we can get better surrogate translations than by using only one of these functions.

Selecting y^-

There are three strategies that are used for selecting y^-, the bad translation, that are considered in machine translation research so far:

- The prediction based strategy where we take the hypothesis with the highest score with parameters [31]

  $\gamma^- = 1 \; ; \; \beta^- = 0$

- The highest cost strategy which is similar to the local update strategy for y^- in a way of using only cost function directly but applying it only on an n-best list which makes usage of score function indirectly [31]

  $\gamma^- = 0 \; ; \; \beta^- = 1$ with the constraint that it can be applied only on n-best list and not during decoding

γ^+	β^+	γ^-	β^-	selection strategy	hypothesis
0	1	-	-	local update	y^+
1	1	-	-	hope	
-	-	0	1	highest cost strategy	y^-
-	-	1	0	prediction-based strategy	
-	-	1	1	fear	

Table 2.1: Strategies for selecting y^+ and y^-

- The fear strategy which is similar to the hope strategy because it uses both cost and score functions but with a difference that the cost is not taken as negative value. Fear represents hypothesis that is very likely by the model but actually being a very bad translation [23]

  $\gamma^- = 1 \ ; \ \beta^- = 1$

Strategies for selection y^+ and y^- combined

In Table 2.1 you can see constants for γ^+_- and β^+_- for presented selection strategies for y^+ and y^-. In [17] all 6 possible combinations of y^+ and y^- selection strategies were tested using MIRA online learning algorithm. Their results show that the only combinations that give good results are:

- local update for y^+ and the highest cost for y^-

  $\gamma^+ = 0 \ ; \ \beta^+ = 1 \ ; \ \gamma^- = 0 \ ; \ \beta^- = 1$ with the constraint of being applied only to the n-best list

- hope strategy for y^+ and fear for y^-

  $\gamma^+ = 1 \ ; \ \beta^+ = 1 \ ; \ \gamma^- = 1 \ ; \ \beta^- = 1$

They conducted two experiments with two different language pairs in which hope/fear was the best method for Czech-English and local update/highest cost was the best method for French-English.

Local update and highest cost strategies were used mostly at the start of usage of discriminative methods in machine translation [1, 31] while most of the recent work uses hope/fear strategy [20, 23].

There is also a choice of how we can find hypotheses defined by parameters for hope/fear strategy. While local update and highest cost can be applied only on the n-best list, hope and fear hypotheses can be searched both in:

- an n-best list as a restricted space of possible translations,

- a word lattice as a restricted space of possible translations,
- a larger space of possible translations by integrating the cost function into the decoding process

The approaches with n-best list and word lattice are simpler and more modular, but searching the larger space of possible translations might give better results. The integration of cost inside the decoding process can be done as in [23] by using the cost as an additional feature in the decoder and giving it a weight that is equal to the sum of weights of all other features that contribute to the real score result. If we are searching for the hope translation, as a weight we take the negative of the sum. That would mean that if, for example, the sum of all weights except the cost is 0.5, the weight for the cost feature should be 0.5 when we search for fear and -0.5 when we search for the hope hypothesis. Searching for the hope hypothesis in this way is called cost-augmented decoding and searching for fear is called cost-diminished decoding [20]. Two additional technical differences are the following:

- applying hope/fear using decoding is two times slower than with n-best list or word lattice because we need to do decoding once for hope and once for fear while with hope/fear using n-best list or word lattice both hope and fear can be selected from the same n-best list or word lattice
- the cost function should be able to evaluate partial hypotheses while with n-best list and word lattice selection it is enough for cost to be applicable only to the complete hypotheses. Knowing how some evaluation functions are bad at sentence level, finding good metric that would work on even a lower level of partial hypotheses can be even harder. We are not aware that anyone has identified this problem before
- the word lattice approach does not require double decoding, it covers a larger space of possible translations which is similar to the size of cost guided decoding, and it also does not require cost function that will work on partial hypothesis level which makes it a good choice for selecting training hypotheses. Some recent systems have started using this method and they reported results that are better than the results from the same system with using n-best lists [8]

2.2.3 Influence of derivations on discriminative training

The *derivation* of hypothesis represents all the decisions that are taken in order to get to that hypothesis. This can include phrase segmentation of the source sentence, selection of phrase pairs used for translation, reordering of phrases and other factors that are used. The derivation is a latent variable in the translation process and because of that it makes the loss function non-convex, which complicates choice of algorithm that will be used for minimizing loss.

The second problem caused by derivations is that for one surface form we can have many different derivations, so which one should we use? Ideally we would marginalize over all possible derivations instead of using only one:

$$p(y|x) = \sum_{d \in \Delta(y,x)} p(d, y|x) \tag{2.4}$$

In [2] they do this type of marginalization and report results which suggest that with using all derivations instead of the best one, significant improvement in translation can be achieved. In order for this marginalization to work, they marginalized feature values as well:

$$H_k(\boldsymbol{d}, \boldsymbol{y}, \boldsymbol{x}) = \sum_{r \in \boldsymbol{d}} h_k(\boldsymbol{e}, r) \tag{2.5}$$

Still, the majority of papers on discriminative machine translation, use the "Viterbi" approximation of the derivation which means that they take the assumption that the difference between the most likely derivation and the other derivations is huge so the marginalized score can be approximated with just a single, most likely, "Viterbi" derivation.

2.3 Algorithms for large-scale discriminative training

In this section, we present a few different algorithms that are used for large-scale discriminative training of machine translation models. Many of them, but not all, follow the definition of ramp loss that we have explained before. Except for the loss function they try to minimize, they also differ in the frequency of updating the parameters—some of the algorithms process training instances in online fashion and some in batches. The reason that is given most often for using online algorithms for large-scale discriminative training is the large number of features and the large amount of data needed for training this number of features

[26]. Until recently, most of the algorithms used for the training of discriminative models were online ones. From online algorithms, we will present Perceptron [31, 26] and the online version of MIRA [11]. Recently, there have been a few successful attempts in using batch algorithms for the training of discriminative models. The main reason for using batch algorithms is that authors of these systems claim that there is still no good fully online algorithm for training non-differentiable and non-convex loss functions like ramp loss [20]. From the class of batch processing algorithms, we will present PRO [24], Rampion [20] and the batch version of MIRA [8].

2.3.1 Perceptron

Perceptron is one of the first algorithms used in the field of artificial intelligence and probably the most simple of all the algorithms that are used for discriminative training in machine translation. The first application of perceptron in discriminative training for machine translation was in [31].

The way perceptron works is by looping trough all instances of tuning data and checking if the best translation according to the model is the same as the reference translation. If it is not the same, then parameters of the model are changed in order to make the output of the systems the same as the reference translation. This process is repeated several times until convergence is achieved. Perceptron has a good property that for linearly separable classes it is guaranteed to converge.

As described above, there are problems in using the reference translation as the desired output from the model. That is why in [31] they use surrogate translation that is taken by the method of local updating ($\gamma^+ = 0$; $\beta^+ = 1$) and the translation to be compared to the surrogate is the one selected by prediction-based strategy ($\gamma^+ = 1$; $\beta^- = 0$). Both of them have to be in the n-best list of translations. The translation that is compared with the surrogate will be there anyway because it will have the highest model score while for the surrogate it is important to be in n-best list to ensure that for the surrogate, we will not take a hypothesis that is very improbable. [31] has also tried bold updating and the combination of bold update and local update strategy and they both gave lower score than local updating alone.

The perceptron algorithm is shown in Algorithm 1.

Implementations for Perceptron exist for most of the popular SMT toolkits. Moses SMT framework supports Perceptron as well as many other algorithms for training discriminative models [23]. There is also an implementation of Percep-

Algorithm 1 Perceptron

Input: set of sentence pairs from tuning corpus (x, y), set of features h

Output: set of feature weights λ

while λ not converged **do**

for all $x_i \in \boldsymbol{x}$ **do**

$y_{oracle} \leftarrow \underset{y \in Y(x_i)}{argmax}\ score(x_i, y)$

$y_{prediction} \leftarrow \underset{y \in\ n\ best\ for\ x_i}{argmax}\ cost(y_i, y)$

if $y_{prediction} \neq y_{oracle}$ **then**

$\lambda \leftarrow \lambda + h(x_i, y_{oracle}, d_{oracle}) - h(x_i, y_{prediction}, d_{prediction})$

end if

end for

end while

return λ

tron for Joshua [30].

2.3.2 MIRA

MIRA is first formulated in [11]. MIRA is a large-margin classifier that is very similar to Perceptron, especially its online version. The main differences between online MIRA and Perceptron are in the selection of hypotheses for training, loss function and in the update rule.

For selecting hypotheses, implementations of MIRA often use hope/fear strategy. Other strategies were also tested in the past and the only strategy that is comparable to hope/fear is local update/highest cost [17]. Some implementations select hope and fear hypotheses from n-best lists [8] while others do cost-augmented and cost-diminished decoding [23].

MIRA update tries to make the margin between model scores of y^+ and y^- at least as big as the difference in the cost. Usually, this requires complicated quadratic programming in order to satisfy the large number of constraints, but if we decide to satisfy only the single most violated margin constraint, analytic solution can be simple and there is no need for quadratic programming [11]. MIRA that is trying to satisfy the single most violated constraint is usually referred to as 1-best MIRA and it is the version of MIRA that is presented in Algorithm 2.

The batch version of MIRA differs from the online version in that it accumulates all the updates to the feature weights vector and applies the averaged update at the end of each iteration. In [8] they use n-best lists instead of cost-

Algorithm 2 1-best online MIRA [17]

Input: set of sentence pairs from tuning corpus (x, y), set of features h, step size C

Output: set of feature weights λ

while λ not converged **do**

$\lambda_i \leftarrow 0$ for all i

for all $x_i \in \boldsymbol{x}$ **do**

$y_{hope} \leftarrow \underset{y \in Y(x_i)}{argmax}\ score(x_i, y) - cost(y_i, y)$

$y_{fear} \leftarrow \underset{y \in Y(x_i)}{argmax}\ score(x_i, y) + cost(y_i, y)$

$margin \leftarrow score(x_i, y_{fear}) - score(x_i, y_{hope})$

$cost \leftarrow cost(y_i, y_{fear}) - cost(y_i, y_{hope})$

if $margin + cost > 0$ **then**

$\delta \leftarrow min\left(C, \frac{margin+cost}{\|h_i(x_i, y_{hope}, d_{hope}) - h_i(x_i, y_{fear}, d_{fear})\|}\right)$

$\lambda \leftarrow \lambda + \delta(h(x_i, y_{hope}, d_{hope}) - h(x_i, y_{fear}, d_{fear}))$

end if

end for

end while

return λ

augmented and cost-diminished decoding because their expectation is that by replicating MERT architecture of optimizing parameters in the same n-best list in many iterations gives better results. They also report even better results by using word lattice in place of n-best lists. The algorithm for batch 1-best MIRA is given in Algorithm 3.

2.3.3 Rampion

Together with the batch version of MIRA [8], Rampion [20] is one of the most recently developed algorithms for discriminative training in machine translation. Because the ramp loss function is non-differentiable and non-convex, standard gradient based methods are inapplicable. Rampion avoids this problem by using concave-convex procedure (CCCP) [53]. CCCP is a batch optimization procedure for functions that can be decomposed to a sum of concave and convex functions. The function is optimized by being approximated with the sum of the convex part and the tangent to the concave part using gradient methods. [20] note that CCCP was used before in different domains with similar non-differentiable and non-convex loss functions where it gave good results. In their experiments, they

Algorithm 3 1-best batch MIRA [8]

Input: set of sentence pairs from tuning corpus (x, y), set of features h, step size C, set of n-best lists ε

Output: set of feature weights λ^{avg}

$t \leftarrow 1$

$\lambda_{t,i} \leftarrow 0$ for all i

$j \leftarrow 0$

while $\lambda_j^{avg} - \lambda_{j-1}^{avg}$ not small enough **do**

$j \leftarrow j + 1$

for all $x_i \in \boldsymbol{x}$ **do**

$y_{hope} \leftarrow \underset{y \in Y(x_i)}{argmax}\ score(x_i, y; \lambda_j^{avg}) - cost(y_i, y)$

$y_{fear} \leftarrow \underset{y \in Y(x_i)}{argmax}\ score(x_i, y; \lambda_j^{avg}) + cost(y_i, y)$

$margin \leftarrow score(x_i, y_{fear}; \lambda_j^{avg}) - score(x_i, y_{hope}; \lambda_j^{avg})$

$cost \leftarrow cost(y_i, y_{fear}) - cost(y_i, y_{hope})$

$\delta \leftarrow min\left(C, \frac{margin+cost}{\|h_i(x_i, y_{hope}, d_{hope}) - h_i(x_i, y_{fear}, d_{fear})\|^2}\right)$

$\lambda_{t+1} \leftarrow \lambda_t + \delta(h(x_i, y_{hope}, d_{hope}) - h(x_i, y_{fear}, d_{fear}))$

$t \leftarrow t + 1$

end for

$\lambda_j^{avg} \leftarrow \frac{1}{j} \sum_{t'=1}^{j} \lambda_{t'}$

end while

return λ_j^{avg}

report improvement over MERT and PRO in using both small and large set of features. The algorithm for batch Rampion is shown in Algorithm 4.

Algorithm 4 Rampion [20]

Input: set of sentence pairs from tuning corpus (x, y), set of features h, initial feature weights λ_0, step size η, regularization coefficient C, number of Rampion iterations T, number of CCCP T' and T'' iterations

Output: set of feature weights λ

$\lambda \leftarrow \lambda_0$
for $iter \leftarrow 1$ *to* T **do**
  $\{\varepsilon\}_{i=1}^{N} \leftarrow Decode(\{x^{(i)}\}_{i=1}^{N}, \lambda)$
  for $iter' \leftarrow 1$ *to* T' **do**
    for $i \leftarrow 1$ *to* N **do**
      $\langle y^i_{hope}, h^i_{hope} \rangle \leftarrow \underset{\langle y,h \rangle \in \varepsilon_i}{argmax}\ score(x_i, y, h; \lambda) - cost(y_i, y)$
    end for
    for $iter'' \leftarrow 1$ *to* T'' **do**
      for $i \leftarrow 1$ *to* N **do**
        $\langle y_{fear}, h_{fear} \rangle \leftarrow \underset{\langle y,h \rangle}{argmax}\ score(x_i, y, h; \lambda) + cost(y_i, y)$
        $\lambda \leftarrow \lambda - \eta C(\frac{\lambda - \lambda_0}{N})$
        $\lambda \leftarrow \lambda + \eta(h(x_i, y^i_{hope}, d^i_{hope}) - h(x_i, y_{fear}, d_{fear}))$
      end for
    end for
  end for
end for
return λ

2.3.4 PRO

PRO [24] stands for pairwise ranking optimization. It is a batch algorithm which is basically a reformulation of the problem that is solved by discriminative training. Instead of optimizing some global loss PRO is trying to minimize the error in ranking between pairs of hypotheses. It follows architecture similar to MERT, and changes only optimization step.

The same way as in MERT there is a candidate generation step or the generation of an n-best list. After that PRO samples pairs of hypotheses from these n-best lists that will be used in training the reranker. This sampling is more general than the one in MIRA and Rampion where we select only by criterions

of hope and fear. The decision of the way how sampling should be done should be brought together with the choice of classifiers. PRO does not constrain the choice of the classifier. Any binary classifier can be used, including MIRA. In the original paper about PRO [24], maximum entropy classifier is used together with random sampling from n-best lists with the criterion that difference in cost between hypotheses in a pair should be at least 0.05. The algorithm as defined there is shown in Algorithm 5. When we sample hypotheses pair $\langle y^+, y^- \rangle$ with corresponding feature vectors $\langle h^+, h^- \rangle$ where y^+ has lower cost than y^-, we generate two training instances for the classifier: the positive one $\langle h^+ - h^-, + \rangle$ and the negative one $\langle h^- - h^+, - \rangle$. After we sample the desired number of training instances, we can train it both in a batch or online fashion. This process of decoding, sampling, generating training instances and training is repeated until convergence.

2.3.5 SampleRank

SampleRank [22] applied on SMT task is a batch tuning algorithm that tries to find weights that make ordering of hypotheses in an n-best list the same as their ordering by *eval*() function. Simplified version of SampleRank from [22] is shown in Algorithm 6. The main difference between SampleRank and other large-margin algorithms for discriminative training is that SampleRank operates on the level of small batches of translations unlike others which operate on the sentence level. That gives SampleRank capability to use corpus level metric, such as BLEU, as an objective function and also leaves space for using document level features (if batches are sampled as documents). Decoding with SampleRank is done only once and after that algorithm iterates in sampling two sets y' and y'' for the given x and comparing their ordering by *score*() and ordering by *eval*() function. If that ordering is not the same, weights λ are updated to make ordering correct. Since *score*() is defined on the sentence level, here its definition is extended to the corpus level by summing scores of all individual translations. Update to the weights can be done in many different ways, but the one used in SMT is mostly the same as MIRA update [22].

Algorithm 5 PRO [24]

Input: set of sentence pairs from tuning corpus (x, y), set of features h

Output: set of feature weights λ

$\lambda_i \leftarrow 0$ for all i
while λ not converged **do**
//Generation of candidates
$E \leftarrow ()$ // set of tuples $\langle x, y, h, y_{ref} \rangle$
for all $x_i \in \boldsymbol{x}$ **do**
$\varepsilon \leftarrow Decode(x_i, \lambda)$
for all $y \in \varepsilon$ **do**
$E \leftarrow E \cup \langle x_i, y, h, y_i \rangle$
end for
end for
//Sampling of candidates
$T \leftarrow ()$ // set of training instances
for $j \leftarrow 1$ *to* M **do**
sample $\langle x', y', h', y'_{ref} \rangle$ and $\langle x'', y'', h'', y''_{ref} \rangle$ uniformly from E for $x' = x''$
if $|cost(y'_{ref}, y') - cost(y'_{ref}, y'')| > threshold$ **then**
$T \leftarrow T \ \cup \ \langle h' - h'', \ sign(cost(y'_{ref}, y') - cost(y'_{ref}, y'')) \rangle$
$T \leftarrow T \ \cup \ \langle h'' - h', \ sign(cost(y'_{ref}, y'') - cost(y'_{ref}, y')) \rangle$
end if
end for
//Training
$\lambda \leftarrow train\_classifier(\lambda, T)$
end while
return λ

Algorithm 6 SampleRank [22]

Input: set of sentence pairs from tuning corpus (x, y), set of features h, set of hypotheses Y

Output: set of feature weights λ

```
for epoch ← 1 to number of epochs do
    A ← randomly_split_parallel_corpus_into_batches(x, y)
    while A ≠ ∅ do
        ⟨x, y⟩ ← randomly_sample_from_set_and_remove(A)
        for s ← 1 to number of samples do
            y' ← randomly_sample_from_set_and_remove(Y(x))
            y'' ← randomly_sample_from_set_and_remove(Y(x))
            if (eval(y,y') − eval(y,y'')) / (score(x,y') − score(x,y'')) < 0 then
                λ ← UpdateWeights(λ, y', y'')
            end if
        end for
    end while
end for
return λ
```

2.4 Technical details about large-scale discriminative training

Large-scale discriminative training usually needs a lot of CPU time whichever algorithm we use. Also the tuning data can be relatively big. To speed up optimization, parallelization can be very useful. This is often achieved by splitting tuning data into shards of equal size and using them in separate parallel jobs. After the end of each iteration, all feature weights vectors of the parallel processes are collected and an average vector is computed. The next iteration continues the same optimization with the averaged vector as the starting point for each of the shards [23, 21].

Whenever training is done with a complex model, as in our case with large number of features, there is the danger of over-fitting. That is why we should have one additional development set for selecting the weight vector from the iterations where the best performance was reached [23]. To avoid confusion with the development corpus used in tuning we call this corpus selection corpus.

3. Objective function

In machine translation, the expected goal is to make the system produce a translation that would prove to be highly rated by humans. The problem with human judgment of MT systems is that it is too slow. Ideally we want an instant answer to the question whether our system is better or worse compared to some other system. Other than the comparison of different MT systems, we want to have an objective function towards which we could optimize our system automatically by using algorithms that were described in the previous chapter. The problem with using automatic metrics as an objective function in large-scale discriminative training is that in large-scale discriminative training we need a comparison of different translations on the sentence level while common metrics are designed to compare large portions of text and usually they do not work well on the sentence level [28].

In this chapter, we are going to look at some metrics that are used most often by the MT community. After that, we will discuss the problems with using these metrics for large-scale discriminative training and propose a solution in the form of a new metric.

3.1 Automatic Evaluation Metrics

Automatic evaluation metrics are usually defined as functions that take two parameters: one is the system's translation and the other is the set of reference translations. As the result of the evaluation function, we get a number that represents how much the system's translation matches the reference translation. The better these automatic scores correlate with human judgment, the better the metric.

Aside from high correlation with human judgment, another preferred property of evaluation metrics is the simplicity of implementation. Some metrics like BLEU are very simple to implement and do not require any additional resources like word alignment or a POS tagger. There are other metrics, such as METEOR, that usually have a higher correlation with human judgment than BLEU but pay the cost of requiring additional data and being slower than BLEU.

Morphologically rich languages have an additional requirement for a good metric, which is to recognize if the system has chosen the right lemma as a translation but not the right word form. Also, many morphologically rich languages, such as Czech, are more flexible in the word order, so the metric that is used

should not be very harsh to different orderings of words between the reference and the system's translation. SemPOS is one of the metrics that was designed specifically to address these problems for the evaluation of Czech translations.

Finally, the most important requirement for evaluation metrics in their usage in large-scale discriminative training is their quality on comparison of different translations on the sentence level or even lower, on the level of partial hypotheses. Corpus level metrics like BLEU do not behave well on the sentence level, but there are alternatives that try to adapt these metrics for usage on the sentence level, such as sBLEU (sometimes called BLEUS), and also completely new metrics that were designed specifically to work on the sentence level.

3.1.1 Precision/Recall based metrics

Many evaluation metrics that are used in the field of Natural Language Processing are based on precision and recall. Precision, as used in machine translation, is defined as the ratio of correct words in the system's translation [26] (by correct we mean words present in both the system's and the reference translation) and the total number of words in the system's translation. The recall is defined as the ratio between correct words in the system's translation and the total number of words in the reference translation. Usually, these two measures are combined into f-measure as described in the formulas below:

$$precision = \frac{number\ of\ correct\ words}{system's\ translation\ length} \tag{3.1}$$

$$recall = \frac{number\ of\ correct\ words}{reference\ translation\ length} \tag{3.2}$$

$$\text{f-measure} = \frac{(1+\beta^2) * precision * recall}{\beta^2 * precision + recall} \tag{3.3}$$

This metric is not used often in machine translation projects but it has some good aspects. First, it is simple to implement and fast to execute. It is a sentence level metric, which is good for our purposes, but it ignores word order which will not lead to learning good parameters during training, especially the weight for the distortion model because any ordering would give exactly the same score. Nevertheless, this metric was interesting enough to give rise to some other metrics

that try to repair the ignorance of word order. One of these metrics is ROUGE-S which we will describe later.

The simplicity of f-measure makes it also useful for the explanation of what other metrics try to achieve. By setting β to a value larger than 1, we will give more importance to recall and by setting β to the smaller value than 1, we will give more importance to precision. Most often $\beta = 1$, which means that precision and recall will have the same influence on the f-measure. If we want correct words in the translation and we can tolerate if some unneeded words are present, then we would prefer a metric with a high recall. If we value more a translation which has fewer words, but most of them should be correct, we will prefer high precision. All metrics try to find balance between precision and recall in different ways.

3.1.2 BLEU and its sentence level approximations

BLEU [41] is a precision-based metric, which computes the number of matched n-grams between the system's translation and the reference. Any n-gram size can be used in theory, but the larger the n-grams used, the more the word order is influencing the score. BLEU also uses a *brevity penalty* to penalize short translations, which is necessary because it is a metric that is based on precision, so this brevity penalty could be considered as some way of recall taking a small part in the final score. The formula for the BLEU score is given below:

$$\text{BLEU-n} = \text{brevity-penalty} \exp \sum_{i=1}^{n} \lambda_i \log precision_i \tag{3.4}$$

$$\text{brevity-penalty} = \min \left(1, \frac{\text{system's translation length}}{\text{reference translation length}}\right) \tag{3.5}$$

In most cases, the maximum size of the n-grams that are used is 4 and all weights of different orders of n-grams λ are set to 1, which simplifies the formula to the multiplication of brevity-penalty and geometric mean of n-gram precisions:

$$\begin{aligned} BLEU4 &= \text{brevity-penalty} \exp \sum_{i=1}^{4} \log precision_i \\ &= \min \left(1, \frac{\text{system's translation length}}{\text{reference translation length}}\right) \prod_{i=1}^{4} precision_i \end{aligned} \tag{3.6}$$

One of the problems with BLEU is that if just one of the n-gram precisions is 0, then the whole score becomes 0, which can happen often if we are evaluating on the sentence level. For example, if we have any three-word reference translation and we are measuring the score of any system's translation with BLEU4, the precision for four-grams will be 0 because there are no four-grams to be matched in the reference, so the whole score will be 0 in every possible case including the case that the translation is perfect. This is why BLEU is often used on the corpus level. For every n-gram order, the number of matched n-grams in each sentence is computed on the whole corpus, which is very improbable to be 0. This strategy gives very good results in correlation with human judgment which has made BLEU the most popular metric in the MT community. However the problem of using BLEU on the sentence level is still there, which makes it unsuitable for usage in large-scale discriminative training algorithms as an objective function. BLEU is also not decomposable in the sense that a sum or an average of sentence scores is not equal to the score of the whole corpus. Therefore, optimizing parameters on the level of the sentence might not lead to the global optimization of BLEU on the whole corpus.

There are few solutions suggested for approximating corpus level BLEU on the sentence level. The most popular of them is smoothed BLEU or sBLEU [33]. What sBLEU does is actually a La-Place's smoothing by adding one additional count to each n-gram count except for unigrams, which makes BLEU score non-zero, unless not even one word was matched.

The other solution for approximating BLEU that is used only for discriminative training and not for comparing different systems is a modification of BLEU by [9]. What their modification does is smoothing of the BLEU score by using the average of the previous translations that gets updated after each new evaluated sentence. They first define a vector $c(e;r)$ where e is the hypothesis to be evaluated and r is the reference translation. That vector contains all information needed to compute the BLEU score: length of e, length of r, for $1 \leq n \leq 4$ counts of n-grams in e and counts of matched n-grams in e. Let us say that for computing BLEU using this vector, we can just call $BLEU(c(e;r))$. We also define a pseudo-document O, an exponentially weighted moving average of vectors c and O_f an exponentially moving length of input:

$$O \leftarrow 0.9(O + c(e)) \tag{3.7}$$

$$O_f \leftarrow 0.9(O_f + |f|) \tag{3.8}$$

Finally BLEU approximation is computed as:

$$B(e; f, r) = (O_f + |f|)BLEU(O + c(e; r)) \tag{3.9}$$

$O_f + |f|$ is needed for controlling the influence of context O on the B score.

3.1.3 NIST

NIST [16] tries to repair the assumption of BLEU, that all n-grams of the same order are equally important. NIST tries to reward the translations which include correct rare n-grams of any order while giving less importance to the translation of n-grams that are seen often. In order to do that, NIST requires information weight for each n-gram that is used. These weights need to be determined before the evaluation and this is usually done by computing these weights on the reference corpus. What is also a good property of NIST, compared to BLEU, is that NIST is a decomposable metric: the average score of all sentence scores is equal to the corpus score.

$$Info(w_1 \dots w_n) = \log_2 \left(\frac{counts\ of\ w_1 \dots w_{n-1}}{counts\ of\ w_1 \dots w_n} \right) \tag{3.10}$$

$$NIST = \sum_{n=1}^{N} \left\{ \frac{\sum_{all\ w_1 \dots w_n\ that\ cooccure} Info(w_1 \dots w_n)}{number\ of\ w_1 \dots w_n\ in\ the\ system\ translation} \right\} * \exp \left\{ \beta \log^2 \left[\min \left(\frac{L_{sys}}{L_{ref}}, 1 \right) \right] \right\} \tag{3.11}$$

3.1.4 METEOR

METEOR [15] is a metric that uses lots of additional linguistic information in order to allow some variation in the system output. By using stemming, METEOR gives some score even to near matching words. It also uses semantic word-nets in order to accept near synonyms. METEOR is a more recall-oriented measure compared to BLEU, which is precision-oriented. The reason for that decision is that having high recall ensures complete meaning of the source sentence captured in the translation [26]. METEOR usually has much better correlation with human judgment, but it usually requires additional linguistic resources in order to

be applied to some language. From the perspective of discriminative training, METEOR has good sentence level correlation with human judgment but it is too slow to compute compared to BLEU.

3.1.5 SemPOS

SemPOS (Semantic POS Overlapping) [28] is a metric that was specifically designed to do evaluation of Czech output. It is based on the semantic role overlapping metric from [19]. Instead of using semantic roles that were defined in that metric and not available in the Czech linguistic resources, SemPOS uses semantic POS from TectoMT [51] framework. Also, instead of surface word form, t-lemma from TectoMT [51] is used in order to be more tolerant on the choice between different variations of word form for lemma in a morphologically-rich language as Czech. In a way, SemPOS has many similarities with METEOR: it is slow to compute, has a requirement of rich linguistic resources and has a high correlation with human judgment on the corpus level. However, it is reported that it has a very low correlation with human judgment on the sentence level which is similar to BLEU sentence level performance [28].

3.1.6 ROUGE-S

ROUGE-S [33] is a variation of f-measure described before. Instead of matching words, ROUGE-S tries to match skip-bigrams which is a better choice because it introduces word order information in the metric's score. A skip-bigram is defined as a bigram that allows skips (other words) between its two words. Let us take the following example of having a reference translation:

R: A B C

and three system's translations:

S1: A B C
S2: C B A
S3: A C B

The total number of skip-bigrams in reference translation is 3: A B, A C, B C. Sentence S1 has all 3 matches of skip-bigrams which will give it a high ROUGE-S score. Sentence S2 has no matching skip-bigrams, which will give it score 0.

Sentence S3 even though it has the same words as for example sentence S1, one word is in a wrong place. It will not have the same score with S1 because of that, but it will still be rewarded for other two matching skip-bigrams: A C and A B. Compared to the previous f-measure on the level of words that would give to all these three sentences the same maximal score, f-measure with skip-bigrams rewards translations with the right word order. It does not require any additional linguistic resources and it is designed to work on the sentence level.

As the original f-measure, ROUGE-S does not differentiate between its two arguments - it does not care which argument is reference and which is system's translation. It just computes similarity between two sentences. This is why in the following formula that describes ROUGE-S we will use X and Y as sentences between which similarity is computed and m and n as their number of words respectively.

$$P_{skip2} = \frac{SKIP2(X,Y)}{C(m,2)} \tag{3.12}$$

$$R_{skip2} = \frac{SKIP2(X,Y)}{C(n,2)} \tag{3.13}$$

$$F_{skip2} = \frac{(1+\beta^2)R_{skip2}P_{skip2}}{R_{skip2} + \beta^2 P_{skip2}} \tag{3.14}$$

where function SKIP2 computes the number of matched skip-bigrams and C computes the number of skip-bigrams for the given length and is computed as the number of word combinations:

$$C(n,k) = \left(\begin{array}{c} n \\ k \end{array} \right) = \frac{n!}{k!(n-k)!} \tag{3.15}$$

ROUGE-S also allows one additional parameter that controls the number of maximal number of skipped words. The value of that parameter is usually added at the end of the name of the metric so for example if the maximal allowed skip is 4 words then name of this version of ROUGE-S is ROUGE-S4. If the maximal number of skipped words is undefined, then that version is called ROUGE-S*. The smaller the number of allowed skips, the more value we give to the word

order. In their work [33] have found that for English the best number of allowed skips is 4.

ROUGE-S, as defined in [33], has a few problems related to its subfunctions $C()$ and $SKIP2()$. In [33] $C()$ is defined as total number of skip-bigrams present in its argument. That is a correct definition of $C()$ if we are using ROUGE-S* but if we are using any other version of ROUGE-S with constrained size of skips it will not be correct. If the number of allowed skips is restricted, then the number of skip bigrams that are present in both the reference and the system's translation is not equal to the number of word combinations. Let us take as an example sentences with four words and the maximal number of allowed skips 1. The number of possible word combinations is 6 but the actual number of skip-bigrams with skip not bigger than 1 is 5. By computing total number of combinations we have included skip-bigram with first and fourth word which should not be included because number of the skipped words is two. The other problem is that if we compare two completely identical sentences of length 4 using the maximal number of skips to be 1, the score will not be 1 but lower number because translation will be punished for not matching the skip-bigram which was forbidden to match. In the other paper about ROUGE-S metric [32], it was mentioned that $C()$ should be modified to compute the actual number of skip-bigram that could be matched but no precise formula was given. The formula we are using for $C()$ is given below:

$$C(n,s) = \begin{cases} 1 & when\ n < 2 \\ \frac{n(n-1)}{2} & when\ 2 \leq n \leq s+2 \quad \vee \quad s = * \\ (n-s-1)(s+1) + \frac{s(s+1)}{2} & when\ s+2 < n \end{cases} \tag{3.16}$$

Here, s represents the number of allowed skips and n the length of the sentence for which we are computing the number of skip-bigrams with a constrained number of maximal skips. This formula might not seem precise because for a sentence with length 0 or 1 a number of skip bigrams is 0, not 1 as given by the formula. The reason for using 1 instead of 0 is to avoid error by division with zero when we compute precision and recall. Because the number of matched bigrams will be zero any way, the final score will still be zero so this impreciseness does not influence the score and saves us from processing results for this functions as special cases when result is zero and when it is not zero. In the case of having infinite number of allowed skips (s = *), this function returns the same result as the function used before for the number of combinations.

If we try this formula on the previous example, we get the correct prediction

of 5 skip-bigrams. The bigger the sentence, the larger is the difference between the correct number of skip-bigrams and the number of combinations. This leads to two problems with the ROUGE-S metric as defined in [33]:

- translations that are longer than their competitors will get punished because the length exponentially influences the number of combinations
- even if all competitors are of the same length, the final number that will result from the score will be too small because they are divided with large denominator. For example, for n=100 and s=4 denominator in the ROUGE-S as defined in [33] will be 4950 while with our function it will be 485

Our formula does not have these problems since the size of the sentence does not exponentially influence the number of skip-bigrams and the final score can be any number between 0 and 1 independently from its length.

There is also one more problem with the imprecise definition of ROUGE-S as given in [33]. In their paper they say that the function SKIP2 computes the number of matched skip-bigrams between its two parameters (two sentences). One possible interpretation of this is that SKIP2 computes the number of the skip-bigrams that are in the first sentence that appear in the second sentence. If we take the simple example with the first sentence being A A A and the second sentence being A A, by this definition, it would mean that the number of matched skip-bigrams is 3 and the recall will be 3, which of course does not make sense since recall is defined in a way not to be bigger than 1. The way we make this definition more precise is shown in Algorithm 7.

By having defined function SKIP2 precisely there could be no confusion of what is meant by the matching skip-bigrams and we would not be led to the wrong result like in the previous example. ROUGE-S works well even with the wrong definition of SKIP2 because the situations that we have described in the previous example are rare, but this could present a problem if we use ROUGE-S with wrong definition of SKIP2 because with the wrong definition the optimization algorithm will prefer output that is long filled with repetitions of matched skip-bigrams, which is a very bad result in the end. With our definition that problem cannot appear because we are recognizing the minimal number of matched skip-bigrams and the system is punished for long output. When we report results with using ROUGE-S evaluation metric, we use ROUGE-S with small modifications on $C()$ and $SKIP2()$ functions as presented here.

Algorithm 7 SKIP2 function

Input: Sentences X and Y

Output: Number of minimal common skip-bigrams

X_{counts} // hash map that contains skip-bigrams of sentence X
// as a key and its count in sentence X as a value

Y_{counts} // hash map that contains skip-bigrams of sentence Y
// as a key and its count in sentence Y as a value

for all skip-bigram $x \in X$ **do**
$X_{counts}\{x\} \leftarrow X_{counts}\{x\} + 1$
end for

for all skip-bigram $y \in Y$ **do**
$Y_{counts}\{y\} \leftarrow Y_{counts}\{y\} + 1$
end for

$matched\_bigrams \leftarrow 0$

for all $x \in keys(X_{counts})$ **do**
$matched\_bigrams \leftarrow matched\_bigrams + \min(X_{counts}\{x\}, Y_{counts}\{x\})$
end for

return $matched\_bigrams$

3.2 Motivation for using ROUGE-S as an objective function

Ideally we would like to use as an objective function for discriminative training an automatic evaluation metric that correlates best with human judgment. In the case when our target language is Czech that metric would be SemPOS since it has good correlation with human judgment [28]. However, there are some problems with using SemPOS as an objective function:

1. it is slow to compute
2. it is a corpus level metric

[29] solved the first problem by preprocessing the training data to include semantic POS and t-lemma as factors which would be used later to build a factored statistical machine translation system. Because the information about a word's semantic POS and t-lemma was available as a factor during tuning, there was no need to do parsing with TectoMT, which is the slowest part of the whole evaluation process. What we have mentioned as the second problem was not a big problem for them because they used MERT as an optimization algorithm,

which can handle metrics that operate on the corpus level. The results that they obtained by using only SemPOS as an objective function were not as good as expected because even though SemPOS was good for comparison of different systems which make different lexical choices, it turned out not to be as good for comparing different but similar translations in the n-best list. This is a similar problem like the one that we have described with having the wrong definition for the SKIP2 function in the ROUGE-S evaluation metric. ROUGE-S was good for comparing systems even with the wrong SKIP2 function, but when we optimize using this wrong function the optimization algorithm exploits the weaknesses of this definition.

As stated above, for the objective function in large-scale discriminative training we need, for most algorithms that are used for this task, a metric that is good in comparing similar sentences with reference translation. Because SemPOS has a bad correlation with human judgment on the sentence level [28] and bad for discriminating similar translations [29] it cannot be used for large-scale discriminative training. A metric such as BLEU is bad on the sentence level, but also bad for languages with free word order and rich morphology [28]. As an alternative to optimization towards BLEU, the most popular choice is sBLEU which was first published in [33]. In that paper, the authors present several different metrics where most of them are new and they all work on the sentence level. They have compared all these different metrics with their new meta-evaluation method called Orange (presented in the same paper) which does not require human judgment scores in order to compare different metrics. In this comparison, the metric that gave the best results on the English translations is ROUGE-S4. It was better then metrics like NIST, BLEU and sBLEU.

[28] have examined sentence level correlation of different metrics with human judgment for translations in Czech and found that the best correlated one is NIST. Together, results from [33] and [28] led us to the idea that ROUGE-S might be a good metric for large-scale discriminative training, because if NIST is the best from all tested metrics on the sentence level in [28] and in [33] ROUGE-S is even better than NIST and sBLEU, which is the most popular choice for large-scale discriminative training, then ROUGE-S might be interesting to test as an objective function. Of course, these comparisons cannot be linked directly because they were done on different data and different languages, but it is enough to support the motivation for our experiments.

The second reason for experimenting with ROUGE-S is that it is a simple metric that does not require remembering the history of the previous evaluations

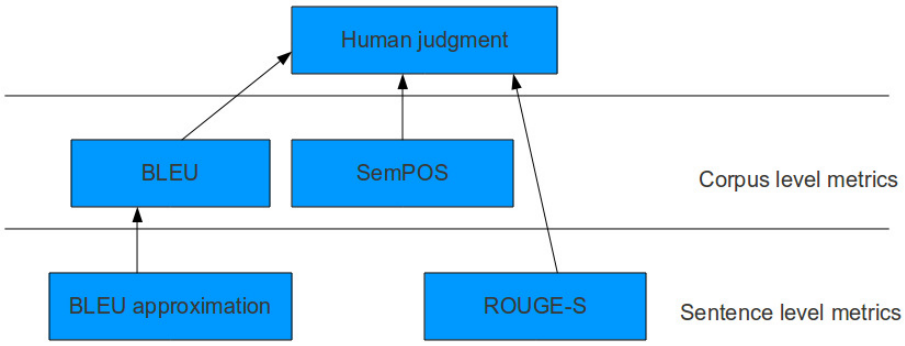


Figure 3.1: Comparison of evaluation metrics

like BLEU approximation used in [9]. Their approximation is based on using the average vector of counts from the previous translations and there is no real reason to expect that to be the right approximation because the history of previous translations might be completely different from the one we are evaluating. Even if we take that these methods of approximation are good, they could also be applied to ROUGE-S if there is need for that. We do not have a direct comparison between their approximation of BLEU and any other metric except in the influence of that metric on discriminative training where it was preferred by them as an objective function over sBLEU.

The final and most important reason for using ROUGE-S is that it is approximating human judgment directly unlike BLEU approximations which are approximating human judgment indirectly by approximating approximation of human judgment. In Figure 3.1 you can see how different metrics approximate human judgment.

4. Sparse features in Large-Scale Discriminative Training

The invention of new algorithms for large-scale discriminative training of machine translation models has opened the opportunity for usage and research of a large number of features for modeling the translation process. These algorithms, however, are not in a sufficient condition for using features with rich linguistic information. Some of the problems are related to the way in which the generative model that produces candidates works, like for example phrase based statistical machine translation which does not (in general case) give any rich linguistic information like parse tree or POS tags. The other problem is that even if we had a perfect system for handling the large number of rich linguistic features, what features would we use? There has not been a lot of research on the rich features for modeling translation output, but help might be found in the area of linguistics. There are also features that are already present in the generative model but could be used more effectively in discriminative models. These are the most often used features in large-scale discriminative training, but to our knowledge they have not been applied to Czech as a target language so far. We will address all these problems in this chapter and suggest possible solutions.

4.1 Simple Generative features in Large-Scale Discriminative Model

It is a standard practice in statistical machine translation to train generative models like the language model and the translation model independently and then use them as features in a discriminative model. It is possible to make these models integrated even more in a discriminative model. For example, instead of using the probability of translation from the translation model we can use individual parts of the translation model that are used for computing the probability and make them discriminative. For example, we can use each phrase pair from the translation model as an independent feature in a discriminative model instead of their generative combination. In most of the practical applications of large-scale discriminative models, these features gave the best results.

The main advantages of using features from generative models in discriminative models for machine translation are:

- correction of errors introduced by training generative model features like phrase pair probability from translation model and n-gram probability from the language model independently on different data set

- correction of overestimated probabilities in generative training

The two most often used features from the generative model in the discriminative model are the discriminative language model (DLM) and the phrase pair feature (PP). DLMs have been used before in many subfields of natural language processing and in particular in speech recognition systems [46]. What a discriminative language model does is giving us an estimate how much some n-gram is a sign of a good translation.

To see the difference between the usage of a discriminative and a generative language model, let us look at the following example. If we have some n-gram that is according to the generative model very probable, this means that in the target language this n-gram appears very often. This information is taken from the target monolingual corpus on which the generative model was trained. However, that n-gram might have a very low score according to the discriminative model because it is a sign of a bad translation. Now this looks like a paradox, but it is not. One of the possible reasons for that n-gram to be sign of a bad translation is that it is, a part of the phrase pair that is learned from bad parallel data. In this case, the discriminative methods act as some corrective factor to the generative features that have problems because they are trained independently.

Controlling the dependency between different features is not the only reason for using discriminative methods. One other example where discriminative methods can correct the judgment of the generative model is the case when some n-gram probability is overestimated. Overestimated n-gram probability will cause translations with high model score and low evaluation score during tuning, which will make its weight in the discriminative model very small or even negative, so it would not cause a low evaluation score during testing.

The other feature that is used often is the phrase pair feature. It gives an estimate of how much some phrase pair, that was used in the generative translation, is informative or a sign of a good translation. That phrase pair might be very probable in a generative model, but this might be an error caused by independent training of features or overestimation. Using phrase pairs as discriminative features can reduce issues that are caused by overestimation of phrase pair probability in the generative translation model and independent training of the generative translation model from other features.

4.2 Generalization of Simple features in Large-Scale Discriminative Model

Features like DLM and PP are used often on the level of word forms, but that does not have to be the case. We can make DLM or PP more robust to the data sparsity problem, that is often present in morphologically rich languages, by using some more abstract representation of a word than its surface form. Some of those more abstract representations can be:

- lemma
- stem [52]
- POS tag [52]
- cluster ID from clustering like in [38]
- affix (usually suffix for languages like Czech) [52]

Each of these features can decrease different types of errors that appear in machine translation with morphologically rich languages:

- lexical choice - lemma and stem are a good choice for solving this problem. They bring similar type of information into the model, so the usage of both of them at the same time might not bring advantage over using only one
- word order - POS tag and cluster ID are good for creating a robust language model. In [37] authors have created two language models for German to be used in generative translation: one based on the POS tags, and the other on the cluster IDs where 50 clusters were built using [38]. The language model built on clusters gave better end translation results than the language model built on POS tags. This can be explained by clusters being more fit to the actual data which we plan to translate, rather than POS tags which are not based on data, but on some linguistic theories.
- word form choice - POS tag and affix feature can influence the choice of the right word form

All these features require some additional processing of all hypotheses in an n-best list. POS taggers and other similar classifiers that are used for gathering information needed for these features are usually trained on data of reasonable quality. Hypotheses in an n-best list might be of very bad quality and therefore

their words can be misclassified because classifiers were trained on data of different quality.

Some researchers have built special classifiers for handling translation output. In [47] they have created a parser for processing translation output by using not only features based on the target language but also features from the source side and alignment between words of source and target sentences. The parser was also trained on translation results which made it robust for handling that kind of text. This parser was created with the reason to do post-processing of translation output, which means that the number of times it is applied is equal to the number of translated sentences which is usually not a big number, so the speed of parsing was not a big limitation for its usage. However, if we want to use this type of parser in discriminative training, it would be applied $number\ of\ tuning\ sentences * number\ of\ iterations * nbest\ size$ times, which is a large number considering how slow the parsing of one sentence can be. This makes the usage of parsers and similar classifiers a bad option for discriminative training.

A similar problem was encountered in [29], where the authors have tried to use SemPOS as an evaluation metric for optimization. SemPOS requires some deeper linguistic information on the target side so the first option to try out was to parse each hypothesis in an n-best list. They have reported that tuning with parsing each hypothesis in an n-best list is extremely slow. Because of that, they have applied a different technique of getting the information they need (lemma and semantic POS). They did all necessary linguistic preprocessing on the training data and set this information (lemma and semantic POS) as a factor in the training data. By training a factored model with linguistic descriptions as additional information in the form of factors they were able to access these descriptions during tuning without processing each entry in the n-best list. This was done in order to use a specific evaluation metric, but the same approach can be applied in finding abstract representations of words that we need for large-scale discriminative training. It should be noted that this method does not come without any cost. Doing factored training with additional factors influences the process of decoding by increasing the search space, so sometimes it can lead to n-best lists with translations of bad quality and then additional features will not help noticeably.

We suggest a simpler approach that can give approximate results, but it is much faster in processing hypotheses in an n-best list than some complex classifiers and it does not require a specific way of training and decoding like the

factored training approach.

As a replacement for lemma and stem, we can use their approximation by taking the first few characters from a word as an approximation. This is a technique that is applied often in the alignment step of building an SMT system [3]. This approximation can be applied only to the languages that have morphology based mostly on suffixes. With the same requirement we can approximate suffixes by taking what is left from the word form when we take away a stem.

As a replacement for POS tagging, we can apply several different approaches. One of them is assigning the most frequent POS tag of the given word form or if that word form was not seen before then assigning proper noun tag. This technique ignores context, but even without considering context it gets around 90% accuracy with languages like English [7]. The second replacement for POS tags is to use cluster IDs instead. Clusters can approximate POS tagging if a reasonable number of clusters (depending on the target language) was used. In languages with a very large number of POS tags, doing clustering with the same number of clusters might be hard, so instead of doing that, we can use a smaller number of clusters together with the approximation of suffixes as an approximation of POS tags.

4.3 Rich linguistic features

There are two problems with using linguistically motivated features in large-scale discriminative training with phrase based statistical machine translation systems. We will call one problem technical and the other one linguistic. The technical problem consists of creating a system capable of having all the necessary linguistic information in order for rich features to work. The linguistic problem is in deciding which features to use. We offer possible solutions to these problems which are later tested in the conducted experiments.

4.3.1 Technical problem of incorporating rich features

What we usually mean by linguistic information is information like POS tags and parse trees. The solutions to the problem of getting POS tags and similar information were addressed in the previous chapter. Here, we will take a look at the problem of getting a parse tree of a target sentence. As we have mentioned before, it has been shown that parsing machine translation output requires a special parser in order to get better results [47] and it can take a long time to tune with parsing each sentence in an n-best list [29]. What we can easily

get is linguistic information on the source side. Since the source side is not changing, we can do all the necessary linguistic preprocessing and then use it for tuning. The hard problem is getting the same type of information on the target side. This problem does not appear in systems that are based on hierarchical translation [18] or dependency treelet translation [43] because the parse tree of the target side is part of a translation hypothesis. This is not the case with phrase based systems and this is why not much research has been done on using rich features with phrase based systems. It was claimed that there is a high correspondence between dependency structures of sentences in parallel corpora. In [45] authors find that 70% of untyped English dependencies correctly map to the corresponding Chinese sentence. The process of mapping dependency trees relies on a few simple heuristics which give relatively good results. Since mapping can be done much faster than parsing we think that it can be a source of syntactic information for the features on the target side. The method that was used in [45] allows any type of alignments, but it creates new empty words on the target side. Instead of that method we used similar, but improved, mapping heuristics that constrain possible alignments, but do not create empty words on the target side. That method was presented in [43]. We also present one more option for finding syntactic information that is a small simplification of the first solution.

Alignments

In mapping source side dependency information to the target side, we need alignments between the words of the source and target sentences. An alignment that is necessary for mapping should be one-to-many or one-to-one in order for the heuristic rules that we use to be applicable. There is one special case of many-to-one alignments that can be allowed: if nodes in the source side of some many-to-one alignment cover a complete branch of the source tree then that alignment is allowed too. Since we already have phrase-to-phrase alignments during the tuning, we can get better quality of word-to-word alignments if the alignment is done on the level of phrase pairs instead of alignment on the whole sentence pair.

Mapping dependency trees from the source to the target side

Ideas from one of the alternatives to phrase based statistical machine translation can give solutions to the problem of mapping dependency trees. In dependency treelet translation [43], a system is trained on a parallel corpus where both sides have dependency trees. The only requirement is that a parser for the source side exists. Target side trees are derived from alignments between words and source

parse tree. In dependency treelet translation, this process is used for extracting linguistically motivated phrases, but in our case we can use the same approach with the different goal of mapping a source parse tree to a target sentence in an n-best list.

Heuristics for mapping starts from the root of the source tree and then map other source tree nodes in the breath-first order. The heuristics for mapping are as follows:

- for one-to-many alignment - map the rightmost node to the word in the source sentence and make dependency of all other words to the rightmost word
- for one-to-one alignment - map source and target word
- for special case of many-to-one alignment - map word on the target side to the highest node of the source side
- for unaligned words on the target side - make dependency between nodes of this type and the closest node on the left or right side from it, that is lower in the parse tree

With these mappings, source tree dependencies can be directly mapped to the target side.

Simplified solution

Previous solution with mapping dependency trees from source to target sentence is tricky to implement and it introduces some assumptions that are not necessarily true. Instead of relying on these assumptions, we can introduce some features that use dependency information, but without mapping the source sentence tree. For example, if we want to implement a sparse feature that is similar to the discriminative language model, instead of using consecutive words, we will use the word and its parent word. One way is to map the source tree to the target and then use that mapped tree to implement this feature. The other way is to use any word on the target side and the parent of the aligned word on the source side. More formally, if we have a mapping $e \rightarrow f$, the new feature will be $(e, parent(f))$ while in the previous case it would be $(e, parent(e))$ where $parent()$ is a function that returns the parent node of its argument from the source tree or tree mapped to the target depending on the argument. Of course, features $(e, parent(f))$ and $(e, parent(e))$ are not the same, but the first one can be considered as an approximation of the second one in the usage of dependency

information. Feature $(e, parent(e))$ also depends on the translation of $parent(f)$ while $(e, parent(f))$ does not.

4.3.2 Linguistic problem of finding rich features

We find that the way in which Optimality theory [42] describes the process of producing the surface form of a sentence is very similar to the process of discriminative training. The grammatical component of optimality theory consists of two subcomponents [50]:

- GEN function which generates candidates for the final surface form
- EVAL function which evaluates all the candidates that GEN has generated

The evaluation is done using some constraints that are ordered by priority. It is expected that these constraints are universal for all languages and that priority is language specific. The sentence that breaks the smallest number of high priority constraints gets selected as the final surface form.

This process looks very similar to the process of discriminative training in machine translation. The GEN function in optimality theory can be seen as parallel to the generation of n-best lists by the decoder and the EVAL function as a parallel to the reranker. This leads us to the idea that the constraints found by researchers in optimality theory can be used as an inspiration for features in discriminative training for machine translation.

Still, not all constraints can be applied as features in discriminative training directly. One of the reasons is that in most cases, they are dealing with some very specific phenomenon that is not of big importance in the target language or in the translation process. The other reason is that they use only target side information and they can be enhanced by using information from both source and target sides.

To our best knowledge, this similarity between the translation process and optimality theory was not mentioned before in machine translation literature and it was mentioned in only two papers by the same author [34, 35] in human translation literature. Since constraints that are mentioned in Optimality theory literature are mostly language specific, we have decided to leave this features for the future research and to concentrate on the general features applicable to all languages. The concrete features that are used are described in the following chapter.

5. Experiments, results and discussion

This thesis tries to address the problems that are present in large-scale discriminative training when the target language is morphologically rich. The problems that we have identified are the following:

1. a large number of word forms that are present in the test data are not seen in the training data [29]

2. even if a word form is seen in the training data, choosing the right word form can be hard [29]

3. BLEU as an evaluation metric does not give good results in evaluating morphologically rich languages [28] and we expect its sentence level approximation, that is often needed in large-scale discriminative training, to give even worse results

We will concentrate on the problem of choosing the right word form and the problem of evaluation metric. We will not deal with the problem of missing word forms in the training data since this problem can, in most cases, be solved only by getting more data. Experiments are based on our ideas that are presented in previous chapters.

5.1 Data, software and baseline systems

Before we present our experiments, we will first describe the data and software that was used for building models that are later optimized using different algorithms. The experiments are conducted on two language pairs: English-Czech and English-Serbian.

For the English-Czech translation model, we have used the news section of the CzEng 1.0 parallel corpus [5]. For training the language model, we used the Czech side of CzEng 1.0 news section. As tuning, testing and selection corpora we used WMT10, WMT11 and WMT12 respectively. These corpora were labelled with additional information such as POS tags and lemma using the Treex toolkit [51]. The corpora used for building English-Czech systems and basic statistics about them are presented in Table 5.1. There are two unoptimized English-Czech models that we use:

corpus	lines	word tokens		word types		lemma types	
		English	Czech	English	Czech	English	Czech
CzEng 1.0 news	197053	4784761	4220203	59214	167436	58048	64699
WMT10	2489	64262	53423	9348	15757	7567	9332
WMT11	3003	77144	66146	10852	17928	8671	10331
WMT12	3003	75102	65682	10173	17990	8174	10155

Table 5.1: Statistics of the corpora used for English-Czech

corpus	lines	word tokens		word types	
		English	Serbian	English	Serbian
SETIMES2 training	197149	4779937	4573150	67308	126444
SETIMES2 tuning	2000	47514	45596	7316	11229
SETIMES2 selection	2000	47910	45949	7227	11120
SETIMES2 testing	2000	47973	45952	7373	11301

Table 5.2: Statistics of the corpora used for English-Serbian

CzechStdModel trained on the source side that is tokenized using standard tokenization for English by Moses toolkit

CzechDepModel trained on the source side that is tokenized using Stanford tokenizer [13]

The reason for having two unoptimized models (which are naturally later optimized) is that one of them, CzechStdModel, gives better results since it uses tokenization that is suitable for machine translation while the other, CzechDepModel, uses tokenization which gives better results in dependency parsing which we will do later.

For training English-Serbian model we used SETIMES2 parallel corpus which is distributed together with collection of other OPUS corpora [49]. The corpus is split into 4 corpora used for training, tuning, selection and testing. The statistics of the used corpora for training English-Serbian systems is presented in Table 5.2. As with English-Czech, here we also create two unoptimized systems:

SerbianStdModel trained on the source side that is tokenized using standard tokenization for English by Moses toolkit

SerbianDepModel trained on the source side that is tokenized using Stanford tokenizer [13]

For experiments, we mostly relied on the Moses PB-SMT toolkit [27]. More precisely, we used the version committed with ID *07a5c67ebce0649cbfa2149b25a-2d3042c612654* to the branch *miramerge* of the Moses git repository located at `https://github.com/moses-smt/mosesdecoder`. Words in the English-Czech and English-Serbian training corpus were aligned on lemmas and word forms respectively using GIZA++ [40]. The language model is built using SRILM toolkit [48]. For some experiments, we needed clustering of words which we performed using mkcls [38]. For testing the rich features we used the typed dependency output from Stanford parser [36]. For controlling the experiments, the eman experiment management system was used [4].

Each of the unoptimized systems was optimized using different algorithms and settings. Depending on the tested features, as a baseline we use one of the following optimizations:

1. MERT with corpus level BLEU score as its objective function
2. Online MIRA with sentence level approximation of the BLEU score [33] as its objective function
3. Batch MIRA with sentence level approximation of the BLEU score [33] as its objective function

As noted in [10], most of the optimization algorithms for discriminative models in machine translation have some randomness and because of that it is required to repeat some experiments several times in order to get statistically significant results. In the case of MERT, it is suggested to run MERT at least three times and then compute the average score and standard deviation. For the MIRA algorithm, this approach does not work because, unlike in MERT, randomness in MIRA does not come from random starting points (usually for most features the initial weight is 0) but from the ordering of training instances. That is why in addition to the repetition of tuning several times, we also shuffle the tuning data after each iteration of tuning.

5.2 Experiments for solving the problem of selecting the right word form

From Table 5.1, we can see that the Czech language data is not much sparser than English when lemmas are considered, but the difference in the number of word forms is huge. To see how big problem this presents if word form or lemma

	types	tokens
word forms	80.5%	93%
lemmas	80.1%	94.4%

Table 5.3: Coverage of the Czech development corpus by the training corpus

is not seen in the training data, we have measured coverage of the Czech side of our development corpus (WMT10) from the training data which can be seen in Table 5.3. The ratio of unseen types of words (both their forms and lemmas) is huge, but these unseen words appear rarely. This problem is even bigger if we look at the coverage on the level of n-grams where the ratio of unseen word forms will grow exponentially with n. Even if all word forms and lemmas from the development corpus were seen in the training data, for many of these words, the number of appearances in the training data might be really small, so we cannot estimate the parameters of our model properly. If we do not estimate the parameters correctly, then even if the system has the correct word form as an option for translation, it will not choose it because the parameters of the model were not well estimated. In order to solve this problem, we need to add features to our model with which it can predict the word form better depending on the word order or the grammatical description of words. To test the possible solutions we divide conducted experiments into:

1. those that use simple features (in some cases with some linguistic abstraction)

2. those that use syntactic information such as dependency trees that are mapped to the target side using method described in the previous chapter

The first group of features was tested on the unoptimized models CzechStdModel and SerbianStdModel, while the second one is tested on the CzechDepModel and SerbianDepModel.

5.2.1 Experiments with simple features

First, we explore sparse features that deal with the basic structure of the sentence on the level of n-grams and do not consider any deeper linguistic information except the surface form of a word. These features are basically features from the generative model trained discriminatively. We used discriminative language models on unigrams (DLM1) and bigrams (DLM2) and phrase-pair (PP) features. We have tried training a trigram DLM feature, but this feature is too sparse to be

	Number of features	Average BLEU score	Standard deviation
MERT core	8	12.37	0.02
MIRA core	**8**	**12.44**	**0.06**
DLM1	9409	12.34	0.08
DLM2	42901	12.41	0.04
PP	17641	12.40	0.06

Table 5.4: Scores for English-Czech translation using simple sparse features on word forms

trained in reasonable time. For tuning, we have used the online version of MIRA because the batch version required too much disk space as it requires storing n-best lists from all iterations with all sparse features and their values in each hypothesis. The results of using the mentioned features with online MIRA on CzechStdModel are shown in Table 5.4.

Even though the tuning corpus was small compared to the size of the corpora that are usually used to learn such a big number of parameters, MIRA was stable and learned reasonable parameters in each case. In some cases where there was a larger number of parameters, the results are even better, which leads us to think that data size was not a big problem in this case. Other researchers in large-scale discriminative training also achieved reasonably good results with tuning data of similar size as used for MERT, but we did not find any explanation why it works well. We think that the reason the small corpus gives good results might be Zipf's law. Features that are very important will appear in the tuning corpus no matter how big it is while rare, unimportant features will not appear and if they appear, algorithms with good regularization will prevent them form having a big influence anyway.

DLM2 and PP have the best results from all sparse features tested on CzechStdModel that use only word forms. The reason for this might be the context that is taken into account in DLM2 and PP features. Together with context, these features take word order information too. DLM1 does not depend on the context or word order at all so it does not help in differentiating hypotheses that have the same words but in different ordering. PP influences the word order because phrases can be of size from 1 up to 7 words. We have computed the average target phrase size on the system's translation of the WMT11 corpus: it is 1.36 words. The average context that is taken by the PP feature is small, but it usually takes larger context when it is important i.e. justified by the data from

which the phrases are extracted. Also, the real choice of word forms is not made on the level of words, but on the level of phrases because we are using a phrase based system. The property of the PP feature of using large n-grams when it is necessary and using small n-grams in other cases, which leads to lesser sparsity compared to DLM2, is interesting and is worth future exploration. Since DLM2 gives the best score from features that use only word forms, we decided to use it as a base on which we will test the effect of adding more linguistic information.

We have created DLM2 features that do not use word form that can be very rare in the tuning data, but instead use some other more abstract linguistic information that appears more often in the tuning data and is common to words that sometimes do not have the same word form. The first type of these features are those that take only some part of the word form and do not require any additional information in order to work. We have tested a prefix feature which takes the first 4 characters of a word and a suffix feature which takes the last 3 characters. The prefix feature in languages whose morphology relies more on suffixes, such as Czech, plays the role of a stem feature. A good property of these types of features is that they are not computationally expensive and can work even on languages that do not have such rich language resources as Czech does.

Words that are shorter than the defined maximum size of an affix are modeled differently. We mark these words as a whole word. For example, if we have defined that we want suffixes of maximum size 2 then we would model word "thesis" as "isX" where "X" represents border of a word, while word "is" would be modeled as "XisX". The reason for this difference in modeling is that frequent words tend to be shorter [12] and we expect that if we allow special treatment of short words we would get their weight more correctly estimated than if we would model it together with suffixes.

We have shown in Chapter 4 that using linguistic information, such as POS tags or lemma, can be hard in discriminative training of phrase-based systems. We have suggested two solutions that are less computationally expensive than the full application of the POS tagger:

1. factored training with POS tags (or lemmas) as an additional factor

2. the application of a unigram tagger that will assign the most frequent tag of a word without considering the context in which it appears

We tried the first solution, but it turned out that large-scale discriminative training of factored models can be quite slow and requires significant amounts of memory even with small size of tuning data. Even though it is less computation-

ally expensive than the application of POS tagger it is still too slow. Because of that, the only solution left is the application of unigram POS tagger.

For building a unigram POS tagger for Czech, we used the Czech side of the news section of the CzEng 1.0 parallel corpus [5] in which each word is labelled with its POS tag and lemma. In this kind of a tagger there is no need for remembering any probabilities. We require only mapping from each word form to its most frequent POS tag or lemma. We evaluated the tagger on the development corpus and got relatively good results that are shown in the Table 5.5. Except standard unigram tagger we tried also two additional heuristics for handling words for which the tagger has no mapping. In the case of the lemma tagger that is mapping word form to itself and in the case of the POS tagger it is tagging that word as a noun. The heuristics for mapping unseen word forms to the noun tag has been applied before for tagging English [7], but it is not very useful in morphologically rich languages like Czech because the noun tag has a large number of variations so it would be hard to pick the right one. Here, we tested this heuristics only to see if it would have a similar effect on Czech if we are satisfied with tagging a word as a noun without any additional information (case, number etc.). All results of unigram tagging are relatively good, considering how simple this method is and how fast it is to compute. Also, these results are realistic in the sense that we can expect similar performance even if the input sentences have some ordering of words that is not represented in the training data. This is not the case with the more complex taggers such as HMM taggers, which are trained to perform well on normal human input but not necessarily on the sentences generated by an MT system.

We also used cluster IDs as a more abstract description of words that have similar properties (assuming that words with similar properties appear in similar contexts). We built 80 clusters by running mkcls for 10 iterations on our training corpus.

What is common for cluster ID, POS tag and lemma features is that they all perform a mapping from the word form to some other description. This is why we have implemented all these features with the same code in the Moses decoder that takes only the file with mappings as a parameter without caring whether it maps to lemma, POS tag, cluster ID or something else because that is irrelevant for the feature in the technical sense. It might seem that the suffix and prefix features could also be implemented as mapping features, but that is not the case. The difference is in the way how unseen words are handled. Prefix and suffix features handle unseen words in the same way as they do words that are seen in

tagger for	tagger	precision
POS	unigram tagger	76.33%
	unigram tagger + unknown words as proper nouns	86.05%
Lemma	unigram tagger	90.79%
	unigram tagger + unknown words as themselves	93.46%

Table 5.5: Precision of unigram tagger

Type of used bigrams	Number of features	Average BLEU score	Standard deviation
Word form (baseline)	42901	12.41	0.04
Suffix 3	29214	12.37	0.06
Prefix 4 (stem)	27818	12.34	0.08
Cluster IDs	**4386**	**12.43**	**0.01**
POS tags	11150	12.37	0.10
Lemmas	26231	12.39	0.09

Table 5.6: Linguistically more abstract DLM2 features scores

the corpus, while *mapping features* emit the token "UNKNOWN" when word is not recognized. We did not apply any of the mentioned heuristics for handling unseen words because we expect that it is better to handle uncertainty of the correct tag with these special tokens than risking to guess the wrong tag and then learning the same weight for correctly and not correctly tagged words.

The results of applying these features on CzechStdModel are shown in Table 5.6. Cluster ID is the least sparse feature which still gives the best results from all used sparse features. One reason for this might be that because we have smaller number of features their weights could be estimated more correctly. The other reason might be that, compared to the POS tags, cluster IDs fit better to the data because members of the cluster are estimated from training data and not by manual labeling. Extracted clusters also contain some semantic information. For example, one of the clusters consisted only of surnames. Cluster ID is the only feature of those that are tested that gives better BLEU scores with low variance than the feature based on word forms.

Prefix (or stem) feature plays similar role as lemma feature – it tries to model lexical choice for translation. In our experiments lemma feature gave better results than prefix feature, but it cannot be applied to languages which do not have resources with words labeled with lemmas.

To some extent POS feature and suffix feature have the common function of

Type of used bigrams	Number of features	Average BLEU score	Standard deviation
MERT core	**8**	**35.87**	**0.02**
MIRA core	8	35.82	0.01
DLM1	11020	35.82	0.02
DLM2	38451	35.77	0.03
PP	24352	35.79	0.01
Suffix 3	26756	35.77	0.02
Prefix 4 (stem)	29972	35.74	0.02
Cluster IDs	4355	35.77	0.02

Table 5.7: Scores for English-Serbian translation using simple sparse features

modeling correct form of a word given the lexical choice. In our experiments they perform equally well with just a little more variance in the results of POS feature.

We have also applied all mentioned features to SerbianStdModel except the POS and lemma DLM2 feature since we did not have labelled Serbian data to train a unigram tagger for these features. The results on SerbianStdModel are presented in Table 5.7. The performance of sparse features on English-Serbian translation look opposite from their performance on English-Czech translation. In English-Czech the more context features capture the better the results, while in English-Serbian the feature that does not take any context into consideration DLM1 gives the best results. We think that reason for this might be different quality in the corpora used for training, tuning, selecting and testing English-Czech and English-Serbian systems. In English-Czech we are using WMT corpora which is of relatively good quality compared to the SETIMES2 corpora which is crawled from the web. For sparse features in English-Serbian system it is more important to penalize wrong words or phrase-pairs than to estimate correctly the correct word form given the context.

The usage of sparse features in our experiments gave us results that are relatively good, but not significantly better compared to the results of the baseline systems which is common in the current state of research in large-scale discriminative training [23, 20, 24]. Except improving the optimization algorithm, improvement with existing features might be achieved by using the tuning data of size bigger than the one usually used for MERT tuning so models with large number of features could estimate their parameters more reliably. The other option is combination of different features that give poor results individually but together might give good results because they solve different problems in translation. One

more possibility is creation of new features that would use more of the linguistic information such as parse trees. We will explore the latter possibility in this chapter. Before we do that let us first analyze the performance of the learning algorithm on the tuning and held-out data.

5.2.2 Analysis of learning sparse features weights

The systems that we analyze in more details are:

1. MERT with core features
2. Online MIRA with core features
3. Online MIRA with DLM2 sparse features

All systems use tuned SerbianStdModel with corresponding algorithms. The reasons for selecting these systems for comparison are possibility of:

- comparing algorithms by comparing systems with the same set of features but different algorithm (system 1. and system 2.) and
- comparing systems with a different set of features but with the same learning algorithm (systems 2. and 3.)

The analysis is done on n-best lists with 100 hypotheses by comparing their:

- System's BLEU score
- Oracle BLEU score
- Average smoothed sentence level BLEU of the whole n-best list

System BLEU score for a given n-best list is computed by selecting for each input sentence the hypothesis with the highest system's model score. From these selected sentences we create a corpus which is evaluated with BLEU metric on the given reference corpus.

Oracle BLEU score is the approximate maximum BLEU score that could be achieved by the right selection of hypotheses. Since BLEU score is a corpus level metric that is not decomposable to it's sentence level version, it is necessary to either do some approximate search of exponential space of possible combinations of hypotheses or to apply some sentence level metric and then select the ones with the highest sentence level evaluation score. In our analysis we opted for the second approach of using the sentence level metric. The example of hypotheses selection

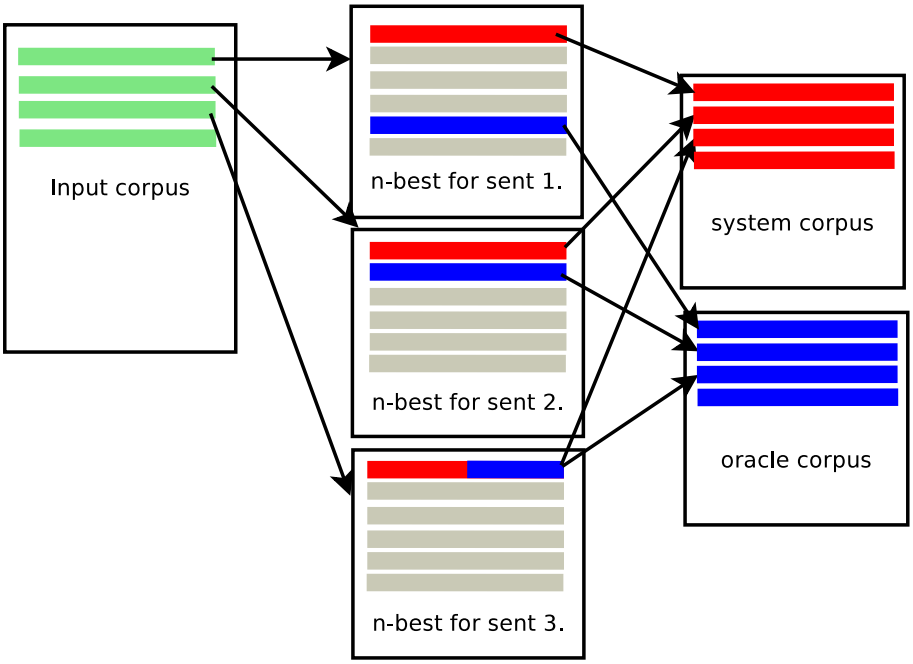


Figure 5.1: Extraction of Oracle and System corpora from n-best lists

is shown in Figure 5.1. The lines that are colored blue represent hypotheses that are best in their n-best lists judging by sentence level evaluation metric and lines that are in red are best judging by model score. When "system corpus" and "oracle corpus" are extracted from n-best lists we evaluate them with corpus level BLEU score on the reference corpus.

For computing the quality of an n-best list it is not possible to apply corpus level evaluation metric such as BLEU. Therefore, we applied the same sentence level evaluation metric on each hypothesis and then computed the average sentence level evaluation score over all n-best list entries.

For sentence level evaluation we use sentence level smoothed BLEU score computed by multeval toolkit [10]. Multeval computes smoothed sentence level BLEU score in the similar way as sBLEU. Instead of adding 1 to to all n-gram counts it adds $\frac{1}{2^k}$ only to n-grams counts where count is 0. k represents the position of an n-gram order in unmatched n-gram orders (for first unmatched order of n-grams k will be 1, for second 2 and so on). For example, if only 3-grams and 4-grams are not matched then their count will be $\frac{1}{2^1}$ and $\frac{1}{2^2}$ respectively.

The n-best lists that are analyzed are produced by running the decoder with the weights that are result of some tuning iteration. These n-best lists are not necessarily the same n-best lists that are used in tuning. For example, in tuning with MERT the n-best lists that is generated by decoder in the current iteration are combined with n-best lists from previous iterations.

First, we will look at the quality of hypotheses from which reranker needs to pick the best one and after that we will analyze the quality of discrimination in this constrained set of hypotheses.

Space of hypotheses for discrimination

We measure the quality of used n-best lists in two ways:

1. by Oracle BLEU score
2. by average smoothed sentence level BLEU score

The first method shows us how well the reranker can perform in the ideal case of having perfect weights for discriminating good hypotheses from the bad ones. In case that the reranker fails to predict the best hypothesis it can still give good results if other hypotheses in the n-best list are of a good quality.

To make the results more reliable we used n-best lists from all MIRA and MERT runs. The oracles (hypotheses which have the highest smoothed sentence level BLEU score) are found in each of these n-best lists independently and independently we compute their corpus level BLEU score. These independently computed BLEU scores are averaged and reported in the thesis. This average value should not be confused with the average sentence level smoothed BLEU score for the complete n-best list.

The quality of an n-best list depends on the decoder which uses learned weights for its search. Since all systems that we have tested use the same decoder with different weights, if n-best lists are bad that is still the result of a bad learning algorithm and not the result of a bad decoder.

The measured quality of n-best lists for the tested systems in each iteration of tuning is presented in Figure 5.2 From these results we can see that MERT is relatively unstable in learning its weights and does not give n-best lists of a better quality than systems tuned with MIRA. This instability in the quality of n-best lists in MERT is usually solved by combining n-best list from the decoder with n-best lists from previous iterations [26]. As expected, the more complex model with sparse features fits better to the tuning corpus than simpler model with core weights.

The quality of n-best lists on the held-out data is presented in Figure 5.3. Here we can see that advantage DLM2 has over MIRA core on the tuning data in improving quality of n-best list is not present on the held-out data. The results also show that one iteration of MIRA (both DLM2 and core) is enough for getting

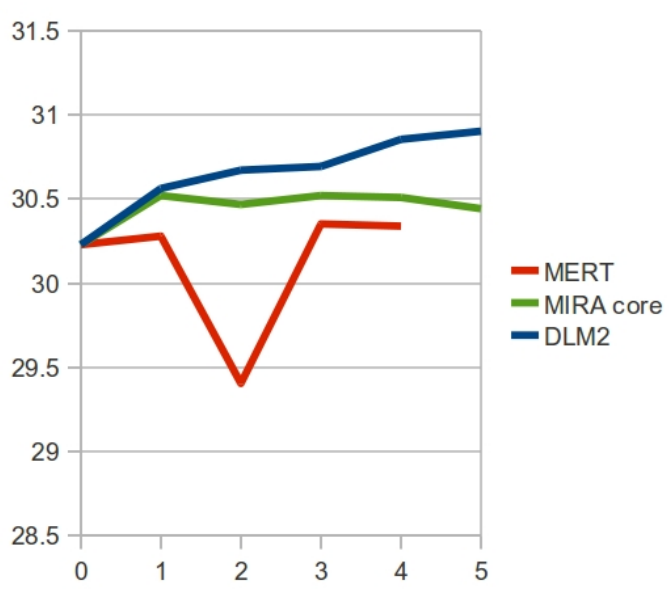


Figure 5.2: Average smoothed sentence level BLEU score of n-best lists on tuning corpus

n-best list of a good quality. MERT is unstable in this case too and does not give good n-best lists.

As we have mentioned earlier, we are not only interested in the average quality of the n-best list but also in its best candidates – oracles. The comparison of the ideal system's BLEU scores for the given n-best lists and its actual BLEU scores is given in Figure 5.4.

These data show that even though the average quality of the n-best lists is somewhat better with MIRA learning algorithm, the difference in quality of oracles between all tested systems is really small. The system with DLM2 features has the highest BLEU score on the tuning data which is expected since it has more complex model that can fit more easily to the data. MERT and MIRA with core features have the same model, but optimize it in a different way which might be the reason for the difference in their BLEU scores. While MERT is trying to minimize directly corpus level BLEU score, MIRA is indirectly minimizing it by minimizing the ramp-loss that uses sentence level BLEU approximation. During tuning, MERT checks different weight settings to see which one maximizes the score we are measuring. MIRA cannot do that for several reasons:

- it is an online algorithm (the MIRA version used in our experiments, not the batch one)
- it can only use sentence level metrics so it cannot be sure whether it improves corpus level BLEU score globally or not

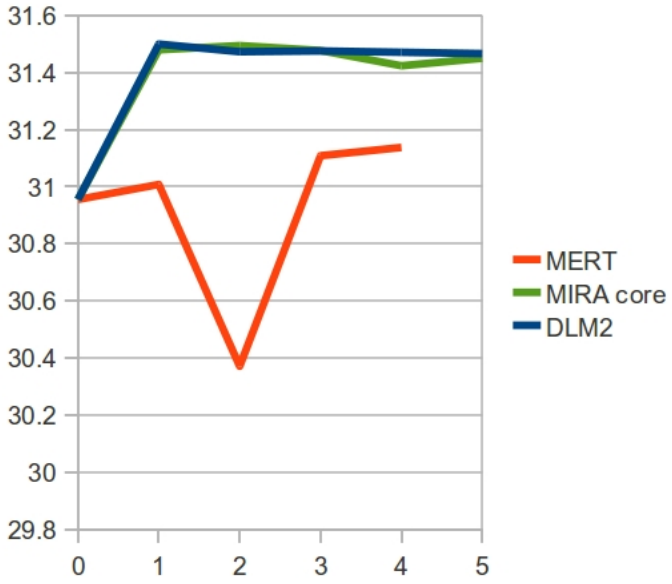


Figure 5.3: Average smoothed sentence level BLEU score of n-best lists on held-out corpus

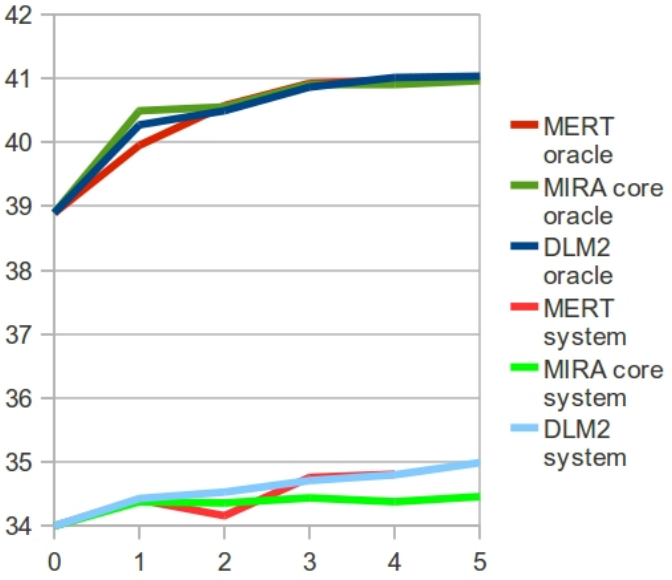


Figure 5.4: Oracle and system BLEU score

- it does not try to increase BLEU score of the translated tuning data but to increase margin between good and bad hypotheses

Having these properties, MERT should outperform MIRA on the tuning corpus with the same model of core weights. It is not easy to say which of these properties influence the training the most. In order to see directly whether the evaluation metric is a problem, we would need to implement both metrics in the same algorithm so we could isolate all other influences to the final result. In our case that is not possible because:

- in MIRA, it is not possible to use corpus level BLEU so that we could test if the sentence level BLEU approximation is the problem
- in MERT, we can use sentence level BLEU for approximation, but there is no reason for doing that since we can optimize directly to our goal – corpus level BLEU

Discriminatory power of different models and algorithms

For investigating discriminative power of different models and algorithms we take two different views on the n-best lists:

1. distribution of oracles in the n-best list ordered by model score
2. distribution of oracles in the top 10 hypotheses by model score

For the first method we use average percentage of oracles distributed in segments of 10 (1-10, 11-20, 21-30, ...). The results for the first method are shown in Figure 5.5.

Here we can see that all systems have more than 30% of oracles in the top 10 hypotheses in their n-best lists. MERT and MIRA with core features have similar distribution while DLM2 has better discrimination in both tuning and held-out corpora. In Figure 5.6 distribution of oracles in the top 10 of n-best lists is presented (numbers are not percentages like in the last case, but the exact number of oracles for translation of 2000 sentences that ended up in top 10 hypotheses judging by model score).

These results confirm that DLM2 has the best discrimination on both tuning and held-out data. If we compare MIRA with core weights and MERT on the held-out data we see similar results that are probably caused by the limit of the model with small number of weight in doing precise discrimination. But, if we compare them in terms of their performance on the tuning data we can

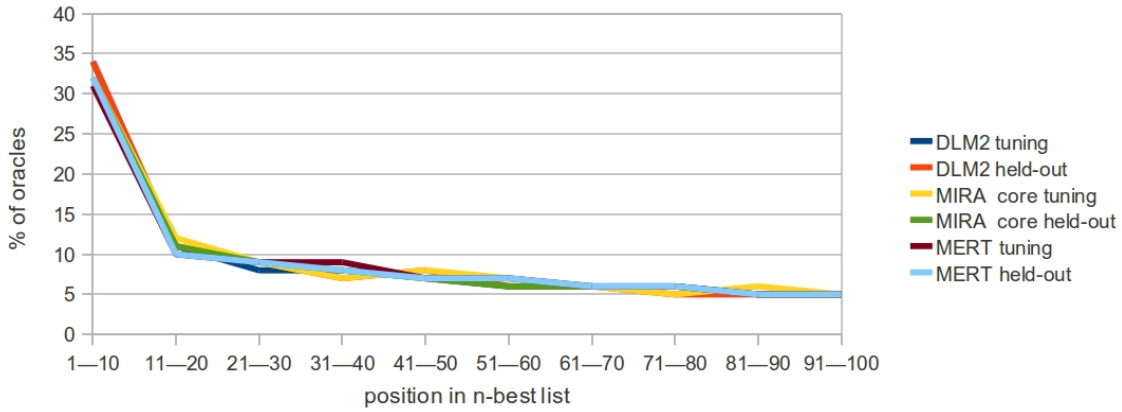


Figure 5.5: Distribution of oracles in n-best list presented by segments of 10 hypotheses (1-10, 11-20, ...)

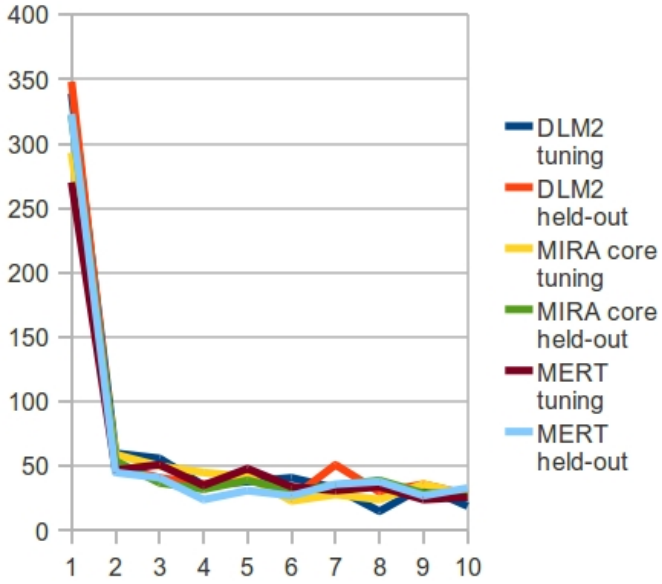


Figure 5.6: Number of oracles in top 10 entries in n-best list

see something interesting. MIRA is discriminating better than MERT on the tuning data. The reason for this is the loss function that these algorithms try to minimize:

- MIRA is using ramp-loss which has a goal of discriminating "good" translations from "bad" ones
- MERT is not working on discrimination of hypotheses directly but on improving global BLEU score

Here we can clearly see advantages and disadvantages of these approaches over each other:

- MERT directly optimizes BLEU metric and indirectly improves discrimination. That is why it had much better BLEU score on the tuning data than MIRA, but lower discrimination
- MIRA directly optimizes discrimination of hypotheses and indirectly improves BLEU score. That is why it has lower BLEU score on the tuning data, but high discrimination

Both of these approaches approach the maximum that can be reached with the small number of weights, just in a different way, and that is why they have similar BLEU score on the held-out corpus. To our best knowledge, there was no attempt to combine MERT and MIRA tuning, but similar combination is present in the Moses decoder as combination of PRO which improves discrimination and MERT which improves global BLEU score.

In Table 5.8 distribution of oracles in the top10 hypotheses over iterations on the tuning data with DLM2 model is presented. What seems surprising in this distribution is that discrimination is lower in each new iteration. The reason for this might be that system has problems with picking the best hypothesis (oracle) because quality of top10 hypotheses is increased in each iteration and even if it makes mistake since quality of alternatives is high, overall error is not high. Decrease of discriminative power over each iteration appears also in MIRA core and MERT tuning.

5.2.3 Experiments with rich features

The experiments that use rich features are conducted with CzechDepModel and SebianDepModel. First we will present the concrete integration of the algorithm that was presented in general in the previous chapter. After that, we will describe the rich features that were used and show the results.

Position \ Iteration	0	1	2	3	4	5
1	491	398	341	333	345	339
2	48	47	54	63	60	60
3	44	56	53	51	48	56
4	30	30	35	37	38	35
5	38	43	28	35	32	38
6	29	21	32	28	32	41
7	26	35	28	30	27	33
8	37	26	26	26	24	15
9	31	33	38	29	42	34
10	25	23	27	25	19	19

Table 5.8: Distribution of oracles in top 10 hypotheses in n-best lists for each iteration of MIRA tuning with DLM2 features

Integration of the mapping algorithm in the tuning process

Since we want the mapping of source to the target trees to be done in a short time it is preferred to precompute as much as possible of the necessary information before tuning. One example of that is computation of word alignments that is necessary for mapping trees. Instead of computing word alignments during tuning between complete source and target sentence we can use two pieces of information that are available to us from Moses decoder:

- alignment between source and target phrases as a result of decoding
- alignment between words inside the phrase pair which were computed during phrase extraction from the training corpus

Combining these two sources we can easily compute word alignment on the level of complete source and target sentence. The small problem in doing that is constraint for many-to-one alignments that is put by the mapping algorithm presented in the previous chapter. Many-to-one alignments are allowed only if the source side of the many-to-one alignments covers one whole branch of the source parse tree. Since word alignments in phrase pairs are computed during phrase extraction, at that moment we do not know the possible parse tree in which that phrase pair might be used. For example, in the training and phrase extraction we can find good many-to-one alignment as in Figure 5.7, but during tuning using many-to-one alignment of the extracted phrase pair might be bad

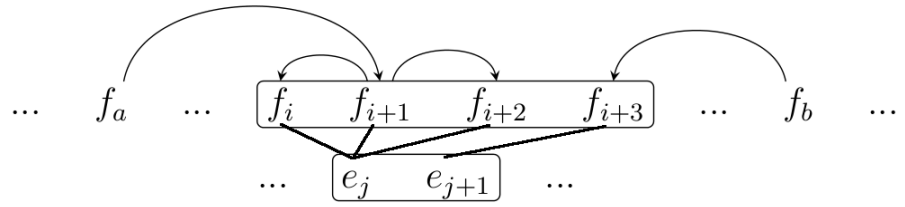


Figure 5.7: Example of a good many-to-one alignment for tree mapping

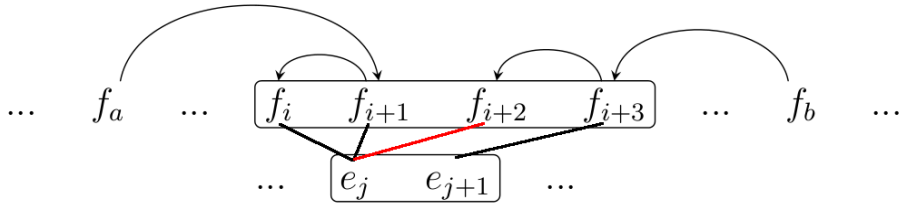


Figure 5.8: Example of a bad many-to-one alignment for tree mapping

if source tree is different as in Figure 5.8. That is why we have decided not to use many-to-one alignments and have in some cases lower quality of the mapped tree, but in return got fast mapping algorithm. This does not mean that phrases are extracted with only one-to-many alignment. For extracting phrases we have used grow diagonal final heuristic, but as a word alignment information in the phrase table we have put only one-to-one and one-to-many word alignments.

With these modifications we can now present the precise algorithm for mapping dependency trees from source to the target side. The algorithm is shown in Algorithm 8. The main data structures are:

alignments one-to-one and one-to-many alignments between words in source and target sentence,

f_deps source parse tree that is represented as an array: index in an array is a child and value at that position in f_deps is the parent. The element on position 0 is the imaginary root node,

e_deps target parse tree with the same properties as f_deps,

f_types types of dependencies in the source tree, for example subject, object, predicate etc.,

e_types mapped f_types to the target sentence,

f_to_process list of source words ordered by their height in the source tree (this can also be precomputed before tuning),

direct_mapping mapping from source node in a dependency tree to it's main mapping node in the target tree,

e_depth depth of each word in the target sentence in the mapped tree.

In lines 1-3 algorithm initialize processing order of source nodes, the imaginary root node of the target tree and direct mapping of the source imaginary root node to the target imaginary root node. In lines 4-20 algorithm finds the dependencies of the aligned target words. In the for loop we first check if the source node is aligned. If it is not aligned it inherits its mapping from its parent node and we move to the next word in the *f_to_process*. If the source word is aligned we take its rightmost aligned target word as a main mapping and then make all other words aligned to the source tree depend on the main mapping. In the for loop in lines 21-32 we are processing unaligned target words. Since they were not aligned, their dependencies were not computed in the previous for loop. In that for loop we take the nearest left and right elements that have dependencies. As a parent of the unaligned word we take the one which is deeper in the tree. For determining the depth of a target word in the mapped tree we use array *f_depth* which is updated every time new dependency is added.

Tested features

Now that we have source tree mapped to the target side we are able to test more syntacticly motivated features. These features might use words from source or target sentence with its word form or more abstract factors such as lemma and POS tag. Here by factors we mean other representation of a word and not additional factor used in decoder during search as in factored training. Factors that are available for some feature depend mostly on the language for which it is used. In Table 5.9 you can see factors that are available in our system depending on a used language.

The features that are implemented can more generally be classified as:

ChildParent combines factor of child with some other factor of its parent. For example child's POS tag and its parent's lemma

ChildChild combines two different factors on one node in the target dependency tree, for example dependency type of a node (nsubj) and POS tag (noun in nominative)

CrossingDependencies outputs the number of crossing and number of non-crossing dependencies. Getting a dependency tree with crossing depen-

Algorithm 8 Tree mapping algorithm

Input: *alignments*, source dependency tree f_deps, types of dependencies in the source tree f_types

Output: e_deps, e_types

$f\_to\_process \leftarrow sort\_by\_increasing\_depth(f\_deps)$ //root excluded
$e\_depth[0] \leftarrow 0$ //depth of the root node is 0
$direct\_mapping[0] \leftarrow 0$ // the source root is mapped to the target root
for all $f_{child} \in f\_to\_process$ **do**
    $f_{parent} \leftarrow f\_deps[f_{child}]$
    if f_{child} is not aligned **then**
        $direct\_mapping[f_{child}] \leftarrow direct\_mapping[f_{parent}]$
        continue
    end if
    $e_{main\_child} \leftarrow$ *the rightmost e aligned to* f_{child}
    $direct\_mapping[f_{child}] \leftarrow e_{main\_child}$
    $e\_deps[e_{main\_child}] \leftarrow direct\_mapping[f_{parent}]$
    $e\_types[e_{main\_child}] \leftarrow f\_types[f_{child}]$
    $e\_depth[e_{main\_child}] \leftarrow e\_depth[e\_deps[e_{main\_child}]] + 1$
    for all e_{child} aligned to f_{child} except e_{main_child} **do**
        $e\_deps[e_{child}] \leftarrow e_{main\_child}$
        $e\_types[e_{child}] \leftarrow$ "*unknown*"
        $e\_depth[e_{child}] \leftarrow e\_depth[e_{main\_child}] + 1$
    end for
end for
for all e_{child} that are not aligned **do**
    $e_{left} \leftarrow$ *the nearest left word to* e_{child} *with a dependency*
    $e_{right} \leftarrow$ *the nearest right word to* e_{child} *with a dependency*
    if $e\_depth[e_{left}] > e\_depth[e_{right}]$ **then**
        $e_{parent} \leftarrow e_{left}$
    else
        $e_{parent} \leftarrow e_{right}$
    end if
    $e\_deps[e_{child}] \leftarrow e_{parent}$
    $e\_types[e_{child}] \leftarrow$ "*unknown*"
    $e\_depth[e_{child}] \leftarrow e\_depth[e_{parent}] + 1$
end for
return e_deps and e_types

Factor \ Language	English	Czech	Serbian
Word form	✓	✓	✓
POS tag	✓	✓	×
Lemma	×	✓	×
Cluster ID	×	✓	✓
Dependency type	✓	✓	✓

Table 5.9: Available factors

dencies is not a problem in our case since our method for mapping parse trees can have these results, while parser that would parse the sentence will probably forbid the crossing dependencies

DependencyDistance outputs the distance in a sentence between a child and its parent or more formally $position(child) - position(parent)$

SrcPhraseCompleteBranch source phrase forms a complete branch of a source dependency tree

TgtPhraseCompleteBranch target phrase forms a complete branch of a the target dependency tree

SrcParent similar to ChildParent with difference that instead of using child's parent in the target tree, we use the parent of the source word aligned to it (simplified method for mapping presented in the previous chapter)

More concretely, we tested the following features on English-Czech language pair with model CzechDepModel:

ChildParent wf wf combines word form of a child and its parent. Can be seen as some discriminative language model that is more suited to the language with relatively free word order such as Czech since it will not insist on bigrams on the surface level of sentence, but on bigrams on the level of dependencies

ChildParent pos pos combines POS tags of a child and its parent. It could model, for example, that it is likely for an adjective to be child of a noun

ChildParent pos lemma combines POS tag of a child and lemma of its parent. It can be useful for cases when noun's case is determined by the verb that governs it

ChildParent cluster cluster combines cluster IDs of a child and its parent. It serves for the same purpose as *ChildParent pos pos* except that cluster IDs are used as a replacement for POS tags

ChildParent cluster lemma is used for the same purpose as *ChildParent pos lemma*

ChildChild type wf combines type of a dependency that the node has on its parent and its word form

ChildChild type pos combines type of a dependency that the node has on its parent and its word form. It can be useful for modeling the case of node having a subject dependency from its parent and being in the nominative case

ChildChild type lemma models the case of having some lemma under some dependency type

ChildChild type cluster serves for the same reasons as ChildChild type pos

CrossingDependencies can model crossing dependencies that are common in Czech - it appears in 23% of sentences in Prague Dependency Treebank [44]

DependencyDistance used in some dependency parsers as a feature for determining how good some dependency tree is

SrcPhraseCompleteBranch similar to the constituent feature used in hierarchical machine translation systems [14]

TgtPhraseCompleteBranch same as SrcPhraseCompleteBranch but on the target side

SrcParent wf wf serves for the same cases as *ChildParent wf wf* but tries to avoid errors in mapping complete trees

The features used in English-Serbian experiments are the same as in English-Czech except the features that use POS and lemma factors which are not available on the Serbian side in translation.

Results

The results for experiments with CzechDepModel are presented in Table 5.10 and for experiments with SerbianDepModel in Table 5.11. In both language

Feature	First factor	Second factor	Number of features	Average BLEU score	Standard deviation
core MERT	-	-	**8**	**10.76**	**0.04**
core MIRA	-	-	8	10.71	0.18
ChildParent	wf	wf	80323	10.71	0.05
ChildParent	pos	pos	10650	10.73	0.02
ChildParent	pos	lemma	37763	10.74	0.04
ChildParent	cluster	cluster	4146	10.72	0.10
ChildParent	cluster	lemma	24208	10.73	0.05
ChildChild	type	wf	16680	10.72	0.03
ChildChild	type	pos	3788	10.66	0.17
ChildChild	type	lemma	**12547**	**10.76**	**0.07**
ChildChild	type	cluster	2021	10.68	0.04
CrossingDependencies	-	-	11	10.60	0.15
DependencyDistance	-	-	146	10.68	0.05
SrcPhraseCompleteBranch	-	-	11	10.75	0.03
TgtPhraseCompleteBranch	-	-	11	10.70	0.07
SrcParent	wf	wf	49412	10.63	0.12

Table 5.10: Dependency features results for English-Czech

Feature	First factor	Second factor	Number of features	Average BLEU score	Standard deviation
MERT	-	-	8	35.10	0.21
core MIRA	-	-	8	35.10	0.22
ChildParent	wf	wf	44127	35.08	0.06
ChildParent	cluster	cluster	4651	35.04	0.08
ChildChild	type	cluster	1587	35.14	0.11
ChildChild	type	wf	**17344**	**35.15**	**0.15**
DependencyDistance	-	-	149	35.10	0.70
CrossingDependencies	-	-	11	35.03	0.16
SrcPhraseCompleteBranch	-	-	11	35.12	0.13
TgtPhraseCompleteBranch	-	-	11	35.06	0.10
SrcParent	wf	wf	34901	35.11	0.09

Table 5.11: Dependency features results for English-Serbian

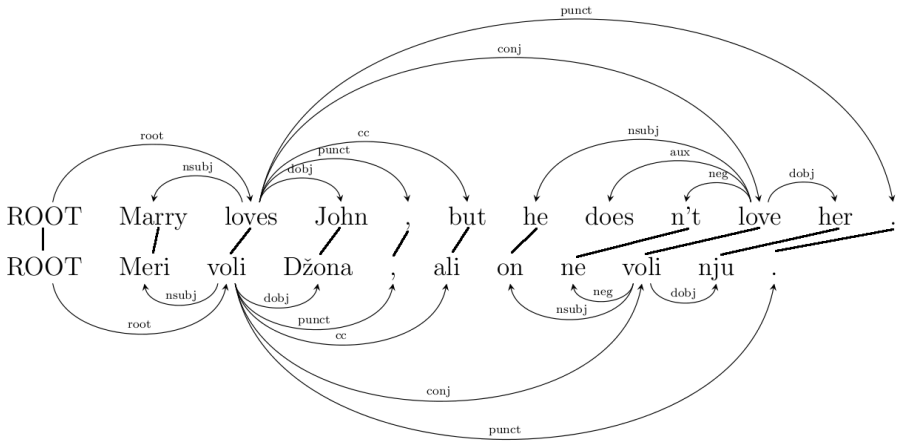


Figure 5.9: Mapping of the dependency tree using the correct alignment

pairs there was no significant improvement over the baseline. To see the reason for these we can take a look at the weights for the best performing feature for English-Serbian language pair — *ChildChild type wf*. Some weights are presented in Table 5.12. The weight for the feature pobj:godine looks correct since *godine* is a genitive form of a word *godina* (year) and objects usually take the genitive case. However, feature root:iz does not seem to have a good weight because it is not very likely that the word *iz* which means *from* could be the root of a dependency tree. The incorrectly estimated weight of that feature has a big influence compared to the weights of core features and that might be the source of errors. We think that incorrect weights are not the fault of a learning algorithm, but of the method for mapping dependency tree which relies largely on correct word-to-word alignment. To see how crucial the alignment is to the mapped tree quality you can look at Figure 5.9 and Figure 5.10 where there is presented mapping of a dependency tree from a source English sentence to the target Serbian sentence using different alignments. On the Figure 5.9 alignments are correct and tree is correctly mapped to the target side, while on Figure 5.10 the main verb (loves) was not aligned and that caused wrong mapping for large part of the tree. In order to try to tackle this problem we added additional binary feature *ROOT_NOT_ALIGNED* which determines the importance of root not being aligned. As it can be seen from the Table 5.12, that weight was correctly estimated as negative but it did not get large importance in the model compared to the other features. Even though *ChildChild type wf* features gave good results and majority of weights seem to be good, we think that dependency that this method has on the correct alignment prevents these features from giving better results.

feature	weight
WordPenalty	-1.91203
root:kaže	-0.260787
...	...
ROOT_NOT_ALIGNED	-0.0916183
...	...
pobj:godine	0.202658
root:iz	0.203838
punct:–	0.223798
...	...
PhraseModel_4	0.299437
root:je	0.331066
PhraseModel_2	0.528343
PhraseModel_3	0.927281
PhraseModel_1	1.00356
Distortion	1.07595
LM	1.33521

Table 5.12: Weights for *ChildChild type wf* of English-Serbian language pair

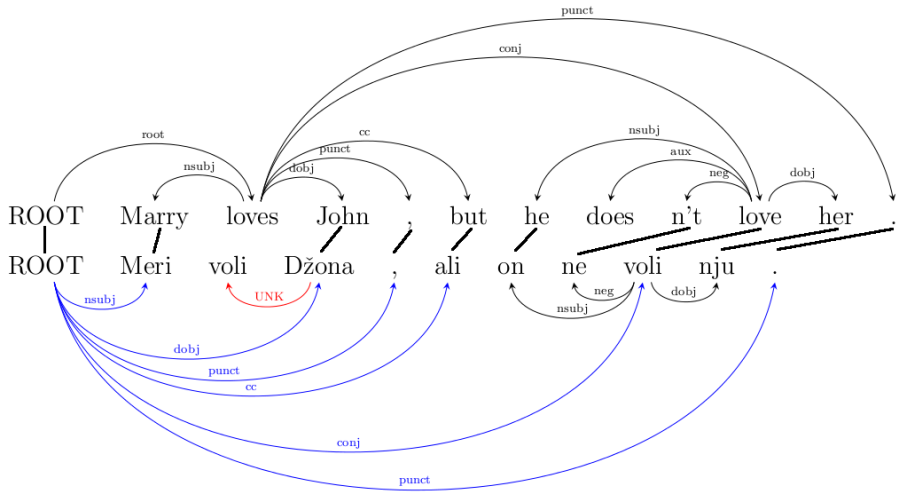


Figure 5.10: Mapping of the dependency tree using an incorrect alignment

5.3 Experiments for solving the problem of evaluation in tuning

As a large amount of research has confirmed before, the BLEU metric [41] has a bad correlation with human judgment on morphologically rich languages, especially on the sentence level [28]. We have computed the correlation of BLEU, sBLEU and ROUGE-S with human judgment by using the same method as the one used in [28] with WMT10 data labeled with their rankings given by human judgment [6]. For computing correlation, we used the Pearson correlation coefficient on ranks shown in Equation 5.1 in which n represents the number of evaluated systems and x_i and y_i are the positions of the i^{th} system by human and the metric's score respectively. Results are shown in Table 5.13.

$$\rho = \frac{n(\sum x_i y_i) - (\sum x_i)(\sum y_i)}{\sqrt{n(\sum x_i^2) - (\sum x_i)^2}\sqrt{n(\sum y_i^2) - (\sum y_i)^2}} \tag{5.1}$$

We can see that ROUGE-S has good correlation if we allow any skip larger than 0. By not allowing skips, we are making ROUGE-S very strict on word order, which is not very useful for evaluating languages with relatively free word order. As it was expected, BLEU of higher order shows bad performance on the sentence level. What is surprising here is that BLEU-1 and sBLEU-1 (essentially the same as BLEU-1) have good scores and they do not take word order into consideration at all. This would make us suspect that word order is not very important, but this is a mistake for several reasons:

1. Sentences that are used for these evaluations are outputs from state-of-the-art systems so they all have relatively good word order. In a language like Czech which has relatively free word order, some variations in word order that different systems produce can be tolerated so human evaluators concentrate more on the lexical choice than on word order.

2. Word order might not matter much when we are comparing outputs from state-of-the-art systems, but we will not have translations of that high quality in the n-best list during tuning. BLEU-1 and sBLEU-1 will not be able to differentiate between different hypotheses with the same set of words but in different ordering

3. If we use these kind of metric which does not take word order into account even if we have good hypotheses in the n-best lists system will not be able to learn weights for distortion because learning algorithm will not know

Metric	Average correlation	standard deviation
BLEU 1	**0.33**	**0.50**
BLEU 2	0.31	0.49
BLEU 3	0.28	0.45
BLEU 4	0.23	0.40
sBLEU 1	**0.33**	**0.50**
sBLEU 2	**0.33**	**0.50**
sBLEU 3	0.31	0.50
sBLEU 4	0.31	0.51
ROUGE-S0	0.30	0.50
ROUGE-S1	0.32	0.49
ROUGE-S2	**0.33**	**0.49**
ROUGE-S3	**0.33**	**0.49**
ROUGE-S4	0.32	0.4
ROUGE-S5	**0.33**	**0.49**
ROUGE-S6	0.32	0.49
ROUGE-S7	**0.33**	**0.49**
ROUGE-S8	**0.33**	**0.49**

Table 5.13: Correlation with human judgment

whether its current weight is good or bad because distortion weight does not influence the objective function's result at all.

We have also computed correlation of the same metrics with corpus level BLEU. We did that on the data produced as a translation of the WMT11 corpus from several differently configured systems (same in everything except in the weight vector). The corpus level value of the evaluated metrics is computed by taking the average of the sentence level scores for each sentence. The results are shown in Table 5.14. Here, it is obvious that metrics that ignore word order approximate corpus level BLEU badly. ROUGE-S, clearly has the best correlation with BLEU compared to all other tested metrics. For experimenting with objective functions, we picked ROUGE-S2 because it correlates well with both human judgment and corpus level BLEU. Additionally as already defined baselines, we used a system optimized with the sentence level BLEU approximation [9]. We tested these objective functions on non-sparse standard features of the PB-SMT system with the batch version of MIRA. The reason for not using the online version of MIRA is that its implementation in Moses [27] chooses *hope* and

Metric	Average correlation	standard deviation
BLEU 1	0.09	0.58
BLEU 2	0.78	0.22
BLEU 3	0.76	0.26
BLEU 4	0.56	0.34
sBLEU 1	0.09	0.58
sBLEU 2	0.78	0.23
sBLEU 3	0.75	0.25
sBLEU 4	0.62	0.26
ROUGE-S0	**0.80**	**0.23**
ROUGE-S1	**0.80**	**0.25**
ROUGE-S2	**0.80**	**0.25**
ROUGE-S3	0.78	0.25
ROUGE-S4	0.77	0.25
ROUGE-S5	0.75	0.28
ROUGE-S6	0.75	0.27
ROUGE-S7	0.74	0.28
ROUGE-S8	0.73	0.29

Table 5.14: Correlation with corpus level BLEU

Algorithm	Objective function	Objective scaled with	Regularization coefficient	Average BLEU score	Standard deviation
MERT	corpus BLEU	1	–	12.37	0.02
Online MIRA	approx. BLEU		0.01	**12.44**	**0.06**
Batch MIRA				**12.43**	**0.05**
	ROUGE-S2		0.0001	12.36	0.01
			0.001	12.36	0.03
			0.01	12.36	0.02
		0.1		12.36	0.02
		0.01		12.24	0.03

Table 5.15: Scores of English-Czech systems tuned using different objective functions

fear hypotheses using cost-augmented decoding with the objective function as an additional feature. With cost-augmented decoding, the evaluation of partial hypotheses is necessary and ROUGE-S has shown to be bad in these cases because it cannot use a large span of skip bigrams in all hypotheses. The batch version of MIRA searches for *hope* and *fear* hypotheses from the n-best list where hypotheses are complete so ROUGE-S can function without a problem. The results of tuning the CzechStdModel are shown in Table 5.15.

We can see that changing the regularization constant (a larger value means less regularization) does not affects ROUGE-S2 optimization. Even though the results of tuning with ROUGE-S2 are relatively good, they are still not better than the results of tuning with the BLEU sentence level approximation. This looks strange because we have seen that ROUGE-S2 approximates the corpus level BLEU score well. We think that the reason for these results is the range of values that ROUGE-S2 takes. When we computed correlation with human judgment or with BLEU, we measured the capabilities of different metrics to differentiate good translations from bad. The result of discrimination of good translations from bad will not change if we scale the result of these metrics with any real number – correlation will stay the same. To see if this affects our results, we have scaled the result of the objective function when we tuned with ROUGE-S2. As you can see from the Table 5.15 if we scale the objective function to a lower value by multiplying it with 0.01, the results will be worse even though the discriminative performance of the objective function stays the same. We believe that the reason for this is the influence that the value of objective function has on:

1. choice of *hope* and *fear* hypotheses

2. size of the update to the model parameters

The hope hypothesis is chosen by taking $\underset{h \in \text{n-best list}}{argmax}\ score(h) + eval(h)$. If we put the scaling factor M to that formula, we can see that it can influence the choice of hope hypothesis by giving more or less influence to $eval()$ compared to $score()$. The same stands for the choice of fear hypothesis. Other algorithms for large-scale discriminative training that do not use ramp loss such as PRO will not have this problem.

$$\underset{h \in \text{n-best list}}{argmax}\ score(h) + M * eval(h) \tag{5.2}$$

If influence of evaluation function on choice of hope and fear is really small, whether it is scaled or not, then the problem of search for *hope* and *fear* hypotheses would reduce in both cases to $\underset{h \in \text{n-best list}}{argmax}\ score(h)$ which means that *hope* and *fear* hypotheses would be the same and no update to the weights would be made. That might be the reason of relatively bad performance of ROUGE-S 2 as an objective function.

Size of the update of parameters in the model depends on the defined loss which is in our case ramp loss defined as:

$$loss(h_{hope}, h_{fear}) = -(score(h_{hope}) + eval(h_{hope})) + (score(h_{fear}) - eval(h_{fear})) \tag{5.3}$$

or more simplified:

$$loss(h_{hope}, h_{fear}) = (score(h_{fear}) - score(h_{hope})) - (eval(h_{fear}) + eval(h_{hope})) \tag{5.4}$$

and when we introduce scaling constant M:

$$\begin{aligned} loss(h_{hope}, h_{fear}) = \\ (score(h_{fear}) - score(h_{hope})) - (M * eval(h_{fear}) + M * eval(h_{hope})) = \\ (score(h_{fear}) - score(h_{hope})) - M * (eval(h_{fear}) + eval(h_{hope})) \end{aligned} \tag{5.5}$$

From the last formula, we can see that even if the constant would not influence the choice of hope and fear hypotheses, it would still influence the size of the update. In algorithms like PRO, which are just binary classifiers where update does not depend on the objective function directly but only on its discriminative properties, these problem would not appear.

Experiments with using different sentence level metrics did not show improvement in the final BLEU score. Knowing that ROUGE-S2 has good correlation with human judgment, it would be interesting to see how translations of the system tuned with BLEU approximation and a system tuned with ROUGE-S2 would be judged by human evaluators. Our attempt of improving evaluation of hypotheses in large-scale discriminative training has the advantage over attempts like SampleRank of being simpler (no changes required from tuning algorithms) and it is general in the sense that it can be used in any SMT tuning algorithm. The problems that are identified with tuning using ROUGE-S2 as a sentence level metric are related to the loss function that is used and might not appear if this metric is tested on some other algorithm that does not use ramp loss.

Conclusion

In this thesis, we have presented a theoretical basis for large-scale discriminative training, explained how it can be used to solve problems that exist in translation into morphologically rich languages and in the end gave the experimental results of applying large-scale discriminative training on the task of translating from English into Czech and Serbian. Some of the ideas that were presented are novel and applicable not only to the task of translation into morphologically rich languages, but also to the more general framework of large-scale discriminative training.

5.4 Summary

In the first two chapters, we presented the theoretical base for applying large-scale discriminative training on the task of machine translation. After that, in chapters 3 and 4 we addressed problems that are present in PB-SMT systems and large-scale discriminative training algorithms in the context of translation into morphologically rich languages.

Problems that we identified with PB-SMT systems in the context of discriminative training are:

1. lack of efficient access to basic linguistic information such as POS tags and lemmas

2. lack of efficient access to more complex linguistic structures such as dependency trees

We suggested several solutions for both of these problems. For the problem of finding POS tags efficiently we proposed two solutions:

1. usage of factored training with POS tags as an additional factor

2. usage of unigram tagger

For the problem of efficiently finding the dependency parse tree of the target hypothesis, we suggested a mapping of the source parse tree to the target side. This algorithm had been used before for extracting linguistically motivated phrase pairs [43], but it had never been applied in the context of large-scale discriminative training. Using this method we were not only able to get parse trees efficiently

on the target language, but also to use parse trees for a language like Serbian for which there are no syntacticlly anotated corpora available.

Another problem that we see with large-scale discriminative training algorithms is mostly related to the objective functions that are used by these algorithms. Almost all of the well known algorithms for this application require a sentence level metric. This presents a problem because:

1. the metrics which correlate well with human judgment are mostly corpus level metrics.

2. knowing that BLEU metric does not correlate well with human judgment in case of morphologically rich languages [28], we expect that its sentence level approximations perform even worse.

3. to our best knowledge, no other metric but BLEU and its sentence level approximations was tested in the context of large-scale discriminative training.

For solving these problems, we suggested usage of sentence level metrics that correlate well with the human judgment. The one we used for our experiments is ROUGE-S2. To our best knowledge, the only algorithm capable of learning weights for large number of features while using a corpus level metric is SampleRank [22]. The disadvantage of SampleRank compared to our approach is that it constrains the user to one specific algorithm for tuning. Our approach is general and could be applied to any discriminative algorithm by just replacing the objective function.

Before these ideas had been applied, we empirically tested problems that we are trying to solve and chances that our solutions might have:

1. in the case of selecting the right word form, we have tested performance of our unigram tagger,

2. in the case of the objective functions, we computed the correlation of the considered objective functions with human judgment and corpus level BLEU.

In both cases we saw encouraging results:

1. the unigram tagger gave reasonably good results considering how simple and fast it is,

2. the metric that we used, ROUGE-S2, showed as the one which correlates best with both human judgment and corpus level BLEU score.

Finally, we applied the suggested ideas and tested the results. No significant improvement over the baseline was achieved, but the insights that the results of these experiments give might still be useful in the future. The tested simple sparse features had different performance on different language pairs. In some cases the context was important for choosing the word form (DLM2 and PP features) and in some cases it was important to discard useless words (DLM1 feature). The more complex syntacticly motivated sparse features gave similar results to the baseline. In many cases the mapping was correct and the learned weights seemed correct, but few wrong mappings were too influential to the final result.

In the experiments for testing different objective functions, we did not achieve a better BLEU score as a result. The approximated BLEU as an objective function gave even better feature weights in the end. We showed that the reason for this unsuccessful attempt may be in the type of loss function that is used by MIRA [17] and might not be present in other large-scale discriminative training algorithms which use different loss functions such as PRO [24].

5.5 Future work

As for future work, we plan to further improve approaches that were used in this thesis. More tuning data might help in overcoming overfitting to the wrongly mapped trees. Some combination of the tested features that gave poor results might give good results if they are combined together solving the different problems in translation. New and more sophisticated features inspired by the optimality theory, that are language specific, could be tested. We plan to test different sentence level metrics and also test them with other algorithms that use different loss functions like the one used in PRO.

Bibliography

[1] Abhishek Arun and Philipp Koehn. Online Learning Methods For Discriminative Training of Phrase Based Statistical Machine Translation. In *Proc MT Summit XI*, 2007.

[2] Phil Blunsom, Trevor Cohn, and Miles Osborne. A Discriminative Latent Variable Model for Statistical Machine Translation. In *Proceedings of ACL-08: HLT*, pages 200–208. Association for Computational Linguistics, June 2008.

[3] Ondřej Bojar, Evgeny Matusov, and Hermann Ney. Czech-English Phrase-Based Machine Translation. In *FinTAL 2006*, volume LNAI 4139, pages 214–224, Turku, Finland, August 2006. Springer.

[4] Ondřej Bojar, Aleš Tamchyna, and Jan Berka. Wild Experimenting in Machine Translation. CLARA Winter School in Prague, January 2012.

[5] Ondřej Bojar, Zdeněk Žabokrtský, Ondřej Dušek, Petra Galuščáková, Martin Majliš, David Mareček, Jiří Maršík, Michal Novák, Martin Popel, and Aleš Tamchyna. The Joy of Parallelism with CzEng 1.0. In *Proceedings of the Eighth International Language Resources and Evaluation Conference (LREC'12)*, pages 3921–3928, Istanbul, Turkey, May 2012. ELRA, European Language Resources Association.

[6] Chris Callison-Burch, Philipp Koehn, Christof Monz, Kay Peterson, Mark Przybocki, and Omar Zaidan. Findings of the 2010 joint workshop on statistical machine translation and metrics for machine translation. In *Proceedings of the Joint Fifth Workshop on Statistical Machine Translation and MetricsMATR*, pages 17–53, Uppsala, Sweden, July 2010. Association for Computational Linguistics. Revised August 2010.

[7] Eugene Charniak. Statistical Techniques for Natural Language Parsing. In *AI Magazine Volume 18 Number 4*, 1997.

[8] Colin Cherry and George Foster. Batch Tuning Strategies for Statistical Machine Translation. In *NAACL*, June 2012.

[9] David Chiang, Yuval Marton, and Philip Resnik. Online Large-Margin Training of Syntactic and Structural Translation Features. In *EMNLP '08 Proceed-*

ings of the Conference on Empirical Methods in Natural Language Processing , pages 224–233, 2008.

[10] Jonathan Clark, Chris Dyer, Alon Lavie, and Noah Smith. Better Hypothesis Testing for Statistical Machine Translation: Controlling for Optimizer Instability. In *Proceedings of the Association for Computational Lingustics*, 2011.

[11] Koby Crammer, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, and Yoram Singer. Online Passive-Aggressive Algorithms. In *Journal of Machine Learning Research 7*, pages 551–585, 2006.

[12] Dan Jurafsky. Probabilistic Modeling in Psycholinguistics: Linguistic Comprehension and Production. In *Probabilistic Linguistics*, pages 39–96. MIT Press, 2003.

[13] Dan Klein and Christopher D. Manning. Accurate Unlexicalized Parsing. In *Proceedings of the 41st Meeting of the Association for Computational Linguistics*, pages 423–430. Association for Computational Linguistics, 2003.

[14] David Chiang. A Hierarchical Phrase-Based Model for Statistical Machine Translation. In *Proceedings of ACL*, pages 263–270, 2005.

[15] Michael Denkowski and Alon Lavie. Meteor 1.3: Automatic Metric for Reliable Optimization and Evaluation of Machine Translation Systems. In *Proceedings of the EMNLP 2011 Workshop on Statistical Machine Translation*, 2011.

[16] George Doddington. Automatic Evaluation of Machine Translation Quality Using N-gram Co-Occurrence Statistics. In *Proc. ARPA Workshop on Human Language Technology*, 2002.

[17] Vladimir Eidelman. Optimization Strategies for Online Large-Margin Learning in Machine Translation. In *Proceedings of the 7th Workshop on Statistical Machine Translation*, pages 480–489. Association for Computational Linguistics, 2012.

[18] Juri Ganitkevitch, Yuan Cao, Jonathan Weese, Matt Post, and Chris Callison-Burch. Joshua 4.0: Packing, PRO, and Paraphrases. In *Proceedings of the Seventh Workshop on Statistical Machine Translation*, pages 283–291, Montréal, Canada, June 2012. Association for Computational Linguistics.

[19] Jesús Giménez and Llúis Márquez. Linguistic Features for Automatic Evaluation of Heterogenous MT Systems. In *Proceedings of the Second Workshop on Statistical Machine Translation*, pages 256–264, 2007.

[20] Kevin Gimpel and Noah A. Smith. Structured Ramp Loss Minimization for Machine Translation. In *NAACL*, 2012.

[21] Cyril Goutte, Nicola Cancedda, Marc Dymetman, and George Foster. *Learning Machine Translation*. The MIT Press, 2010.

[22] Barry Haddow, Abhishek Arun, and Philipp Koehn. SampleRank Training for Phrase-Based Machine Translation. In *Proceedings of the Sixth Workshop on Statistical Machine Translation*, pages 261–271, Edinburgh, Scotland, July 2011. Association for Computational Linguistics.

[23] Eva Hasler, Barry Haddow, and Philipp Koehn. Margin Infused Relaxed Algorithm for Moses. In *The Prague Bulletin of Mathematical Linguistics No. 96*, pages 69–78, 2011.

[24] Mark Hopkins and Jonathan May. Tuning as Ranking. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 1352–1362, 2011.

[25] Reinhard Kneser and Herman Ney. Improved backing-off for m-gram language modeling. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal, volume 1.*, pages 181–184, 1995.

[26] Philipp Koehn. *Statistical Machine Translation*. Cambridge University Press, 2010.

[27] Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondrej Bojar, Alexandra Constantin, and Evan Herbst. Moses: Open Source Toolkit for Statistical Machine Translation. In *ACL 2007, demonstration session*, 2007.

[28] Kamil Kos and Ondřej Bojar. Evaluation of Machine Translation Metrics for Czech as the Target Language. In *The Prague Bulletin of Mathematical Linguistics*, pages 1–11, 2009.

[29] Kamil Kos and Ondřej Bojar. 2010 Failures in English-Czech Phrase-Based MT. In *Proceedings of WMT'10*, 2010.

[30] Zhifei Li and Sanjeev Khudanpur. Forest Reranking for Machine Translation with the Perceptron Algorithm.

[31] Percy Liang, Alexandre Bouchard-Côté, Dan Klein, and Ben Taskar. An End-to-End Discriminative Approach to Machine Translation. In *Proceedings of COLING-ACL*, 2006.

[32] Chin-Yew Lin and Franz Josef Och. Automatic Evaluation of Machine Translation Quality Using Longest Common Subsequence and Skip-Bigram Statistics. Submitted. 2004.

[33] Chin-Yew Lin and Franz Josef Och. ORANGE: a Method for Evaluating Automatic Evaluation Metrics for Machine Translation. In *Proceedings of the 20th International Conference on Computational Linguistics (COLING 2004)*, 2004.

[34] Richard M. Mansell. *Optimality Theory Applied to the Analysis of Verse Translation*. 2004.

[35] Richard M. Mansell. Optimality in translation. In *Pym A, Perekrestenko A (eds) Translation Research Projects 1*, pages 3–12, Tarragona (Spain): Intercultural Studies Group, 2008.

[36] Marie-Catherine de Marneffe, Bill MacCartney and Christopher D. Manning. Generating Typed Dependency Parses from Phrase Structure Parses. In *LREC 2006*, 2006.

[37] Jan Niehues, Yuqi Zhang, Mohammed Mediani, Teresa Herrmann, Eunah Cho, and Alex Waibel. The Karlsruhe Institute of Technology Translation Systems for the WMT 2012. In *Proceedings of the Seventh Workshop on Statistical Machine Translation*, pages 349–355, Montréal, Canada, June 2012. Association for Computational Linguistics.

[38] Franz Josef Och. An Efficient Method for Determining Bilingual Word Classes. In *Ninth Conf. of the Europ. Chapter of the Association for Computational Linguistics*, pages 71–76, June 1999.

[39] Franz Josef Och. Minimum Error Rate Training in Statistical Machine Translation. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, pages 160–167. Association for Computational Linguistics, July 2003.

[40] Franz Josef Och and Herman Ney. Improved Statistical Alignment Models. pages 440–447, Hongkong, China, October 2000.

[41] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *ACL 02: Proceedings of the 40th Annual Meeting of the ACL*, pages 311–318. Association for Computational Linguistics, July 2002.

[42] Alan Prince and Paul Smolensky. *Optimality Theory: Constraint Interaction in Generative Grammar*. Blackwell Publishers, 1993.

[43] Chris Quirk, Arul Menezes, and Colin Cherry. Dependency Treelet Translation: Syntactically Informed Phrasal SMT. In *Proceedings of ACL*. Association for Computational Linguistics, June 2005.

[44] Ralph Debusmann and Marco Kuhlmann. Dependency Grammar: Classification and Exploration. In *Resource-Adaptive Cognitive Processes*, pages 365–388. Springer, 2009.

[45] Rebecca Hwa, Philip Resnik, Amy Weinberg and Okan Kolak. Evaluating Translational Correspondence using Annotation Projection. In *Proceedings of the 40th Annual Meeting of the ACL*, 2002.

[46] Brian Roark, Murat Saraclar, Michael Collins, and Mark Johnson. Discriminative Language Modeling with Conditional Random Fields and the Perceptron Algorithm. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics*, pages 47–54, 2004.

[47] Rudolf Rosa, David Mareček, and Ondřej Dušek. DEPFIX: A System for Automatic Correction of Czech MT Outputs. In *Proceedings of the Seventh Workshop on Statistical Machine Translation*, pages 362–368, Montréal, Canada, June 2012. Association for Computational Linguistics.

[48] Andreas Stolcke. SRILM – an extensible language modeling toolkit. In *Proceedings of ICSLP, Vol. 2*, pages 901–904, 2002.

[49] Jörg Tiedemann. Parallel Data, Tools and Interfaces in OPUS. In Nicoletta Calzolari (Conference Chair), Khalid Choukri, Thierry Declerck, Mehmet Ugur Dogan, Bente Maegaard, Joseph Mariani, Jan Odijk, and Stelios Piperidis, editors, *Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC'12)*, Istanbul, Turkey, may 2012. European Language Resources Association (ELRA).

[50] Pavol Štekauer. *Rudiments of English Linguistics*. Slovacontact, 2000.

[51] Zdeněk Žabokrtský, Jan Ptáček, and Petr Pajas. TectoMT: Highly Modular MT System with Tectogrammatics Used as Transfer Layer. In *Proceedings of WMT'08*, 2008.

[52] Taro Watanabe, Jun Suzuki, Hajime Tsukada, and Hideki Isozaki. Online Large-Margin Training for Statistical Machine Translation. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 764–773, Prague, Czech Republic, June 2007. Association for Computational Linguistics.

[53] A. L. Yuille and Anand Rangarajan. The concave-convex procedure (CCCP). In *Proc. of NIPS*. MIT Press, 2002.