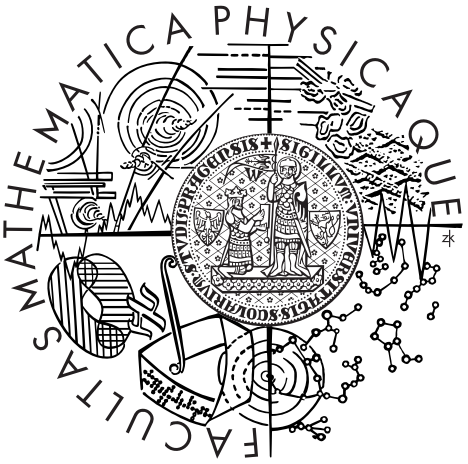
Charles University in Prague
Faculty of Mathematics and Physics

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

Prague 2012

I would like to acknowledge my supervisor Ondřej Bojar for his support, patience and helpful comments on my work.

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: Využití diskriminativních metod ve strojovém překladu do jazyků s bohatou morfologií

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: Práce představuje teoretické základy pro diskriminativní metody trénování s mnoha rysy a jejich aplikaci ve strojovém překladu do jazyků s bohatou morfologií. Výzkum se zaměřuje zejména na dva aspekty diskriminativních metod s mnoha rysy. Prvním z nich je vliv řídkosti rysů na výběr slovních tvarů a slovosledu. Druhým je využití různých metrik na úrovni věty jako účelových funkcí k optimalizaci parametrů.

Klíčová slova: discriminative training, MIRA, sparse features, machine translation, evaluation metrics, ROUGE-S

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

Author: Miloš Stanojević

Supervisors: RNDr. Ondřej Bojar Ph.D. and Mr Mike Rosner

Abstract: We present the theoretical foundations for large-scale discriminative training and their application on the machine translation into MRL languages. The research is concentrated mostly on the two aspects of large-scale discriminative training. The first aspect is the effect of sparse features on the choice of word form and word order. The second aspect is application of different sentence-level metrics as an objective function in the parameter optimization.

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

Contents

Introduction

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

- basic part with a generative model and a decoder which generates candidate translations with high probability (according to a 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 basic generative component of this type of SMT systems, and after that we will look at discriminative component. When we introduce these basic components of phrase based SMT system we will turn to concrete ideas that are applied in this thesis: different objective function and sparse features that were 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 the target language can be seen as search for the translation e in a space E of all possible sentences in a target language that has the highest probability of being a translation of f. More formally we are trying to find:

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

This formula can further be simplified to:

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

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

1.1 Translation model

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

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

d is a distortion model which penalizes big reordering in translating phrases, a_i is a starting position of a foreign phrase $\bar{f}_i$, b_{i-1} is a end position of a target phrase $\bar{e}_{i-1}$ and Φ is giving probability of phrase f being translation of phrase 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.

1.2 Language model

There are different types of language models. Some are based on modeling the probability of a target sentence using syntactic information and some other using

much simpler methods. The type of language model that is used most often is the n-gram language model that models probability of a sentence by modeling the product of the probability of its words conditioned by their history:

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

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 a sentence using an n-gram model where n is the order of the history is given in the following formula:

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

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 [19].

1.3 Decoder

The decoder is used for searching for the best translation using given language and translation models. Because it is impossible to search trough an infinite space of possible translations, some approximated form of the search is needed such as A*. Decoding can be seen as a search graph where each node in that graph represents a hypothesis and it contains all the information about translated words and their probability. Transition from one node to the next node is called hypothesis expansion and adds some additional translated words to the hypothesis and updates its probability. Decoding with A* is done by doing most probable expansion of all currently observed hypotheses. All hypotheses are put into one priority queue that is often called stack. In the machine translation and the speech recognition community, A* applied in decoding is often referred to as a stack decoding algorithm. A comparison between hypotheses that have translated a different number of words is not good for choosing the next expansion because the hypothesis with a smaller number of translated words will be chosen in most cases as a 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 so we could solve them independently and then combine their results creating a good translation. For example we assume that translation of different phrases is independent. 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 the 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 dependence of search errors from numbers 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 discriminate good translations from bad. Because of that, 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 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 a generative fashion to produce a relatively small finite set of most probable translations (by 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 on 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 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 discriminative model that is used by reranker is usually called tuning because some of the features are used in 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 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. This is repeated for several iterations until we get convergence.

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

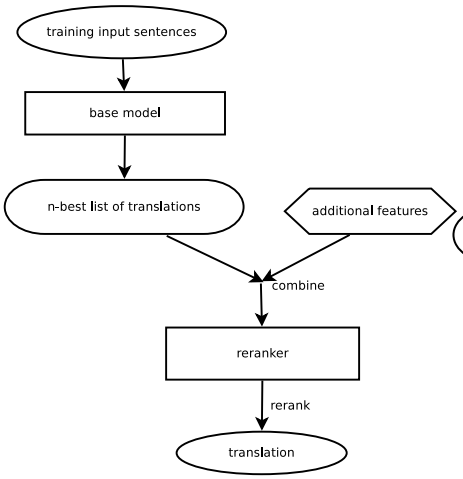


Figure 2.1: Usage of reranker

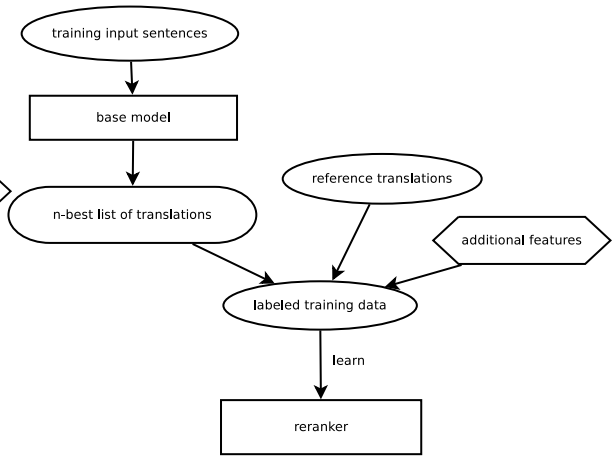


Figure 2.2: Training of reranker

by reranker.

2.1 Finding best candidates for translation

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 largest part of the source sentence. One of possible representations of best hypotheses is word lattice which is basically a weighted finite-state machine that is very similar to a search graph. The main difference is that instead of having the probability of a hypothesis we have the probability of transitioning from one hypothesis to another or, because it is a weighted FSM, the probability of transitioning from one state to another.

Another way to represent best hypotheses that is used often 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 [20]. Compared to the best path trough the graph, the second best path takes a detour from the best path. The detour is defined as suboptimal transition from start state to the state in the best path for which we take detour. Detours for each state in FSM are already discovered during the search by recombination of hypotheses.

The algorithm is based on having two queues:

- queue for the resulting n-best list
- 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 all the detours from it 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 previous detour are added to the priority queue of detours

5. If the n-best list is not of the desired size go to 2.

The larger the set of hypotheses that reranker has access to the higher is the probability that it will be able to choose good translation. Word lattice is a more compact representation of translation hypotheses than list of best translations and that is why it allows access to the larger part of 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 and because of that, n-best lists are still used more often than word lattices.

2.2 Reranker

As it was mentioned before, reranker takes n-best translations and gives the best one from it judging by some discriminative model. Name reranker is taken from machine learning field but it is little bit misleading[15]. What reranker in machine translation does is not really reranking. The goal of a real reranker is to put every element in a list to its right place judging by some criterion while the goal of reranker in machine translation is to put only the best translation to its right place (1st place) without caring about other translations.

So what reranker does is scoring every hypothesis in the n-best list using some model and then outputting the translation with the highest score. Model that is used is almost always the log-linear model which can have from dozen of features to a few hundred thousands. What reranker actually does is described with the following formula [17]:

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

λ 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 some reranker consists of finding the right set of λ parameters that maximize some objective function on the processes of translating the tuning corpus. The ideal objective function is human evaluation which is of course impossible to use in the training so instead of that, we use an automatic evaluation

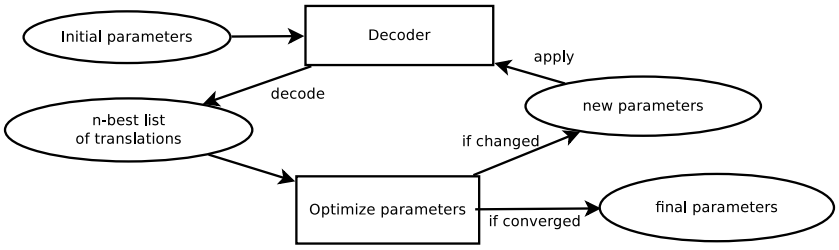


Figure 2.3: Learning parameters

metric that correlates well with the human judgment. The most popular metric in machine translation community is BLEU [33] 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 negative value of EVAL.

The general process of training a reranker can be seen in Figure 2.3. We first a translate 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 a discriminative model. Learning these parameters 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[20]. Algorithms that are used for learning can be roughly separated in two groups: -those that are good at learning small set of parameters usually by applying some heuristics -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) [32]. It is also an important algorithm since many other algorithms that work with large number of features, such as PRO, take inspiration from MERT.

MERT is optimizing weights one by one and while it is opetimizing any 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 large number of features. 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 small number of features this risk is small, but with 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:

$$loss = cost(y, \underset{(y,h)\in\tau(x)}{argmax}\ score(x, y, h; \lambda))$$

λ is a set of learned feature weights
h is a set of features
y is a reference translation or hypothesis if it is under $argmax$
x is the source sentence

Good properties of MERT are that:

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

On the other hand, MERT also has some problems:

- it is very unstable because of loss complexity and difficulty of search [15]
- it can work well only with small set of features [18]

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 [15]:

- these algorithms usually assume that a single correct result is available. Problems with that are:
  - The reference translation might not be reachable by our model
  - Even if the reference translation is reachable, we might not got the correct result using correct 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 [15]
- reranking algorithms in machine learning have a similar goal as reranking in machine translation but usually not completely the same - 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 [15]

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

$$loss = -\max_{y' \in Y(x_i)} (\gamma^+ score(x_i, y^+) - \beta^+ cost(y_i, y^+)) + \max_{y' \in Y(x_i)} (\gamma^- score(x_i, y^-) - \beta^- cost(y_i, y^-)) \quad (2.2)$$

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 on the participation of *cost* and *score* in the loss
score function here represents score given by the discriminative, not generative, model

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 chapter, 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 largest attention and also the largest 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 enter in the n-best list because it is ether:

- unreachable by our model or
- reachable but very unprobable, 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 ...) and getting the correct surface form is just a lucky coincidence.

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 the replacement for the reference translation. The loss function that was defined before in the parametrized 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 referred usually as ramp loss [15].

The other solution to this problem, which gave bad results [25], 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[17]. 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 [25]. The score function is used here indirectly because the hypothesis with the lowest model score will not enter the n-best list and therefore, cannot 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 [25]

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

- The highest cost strategy which is similar to the local update strategy for y^-. in a way for using only cost function directly but applying it only on an n-best list which makes usage of score function indirectly [25]

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

- 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 very bad [17]

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

Strategies for selection y^+ and y^- combined

In [12, Eidelman] 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, 25] while most of the recent work uses hope/fear strategy [15, 17].

There is also a choice of how can we 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 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 [17, Hasler et al.] by using cost as an additional feature in the decoder and give 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 fear translation and the negative of that value if we are searching for hope translation. That would mean that if

for example the sum of all weights except 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 hope hypothesis in this way is called cost-augmented decoding and searching for fear is called cost-diminished decoding [15]. Two additional technical differences are that:

- 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 [6]

2.2.3 Influence of derivations on discriminative training

The derivation of some hypothesis represent 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:

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

In [2, Blunsom et al.] 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 too:

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

Still, in the majority of papers on discriminative machine translation, viterbi approximation of the derivation is used 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 the 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 [20]. Until recently, most of the algorithms used for the training of discriminative models were online ones. From online algorithms, we will present Perceptron [25, 20] and the online version of MIRA[9]. Recently, there were 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 [15]. From the class of batch processing algorithms, we will present PRO[18], Rampion and the batch version of MIRA.

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 [25, Liang et al.].

The way perceptron works is by looping trough all instances of training 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 before, there are problems in using reference translation as the desired output from the model. That is why in [25, Liang et al.] they use surrogate translation that is taken by 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 unprobable. [25, Liang et al.] 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 algorithm can be formulated as follows:

Input: set of sentence pairs from tuning corpus (x, y), set of features h
Output: set of feature weights λ
$\lambda_i = 0$ for all i
while not converged do
    for all x_i in **x** do
        $y_{oracle} = \underset{y \in Y(x_i)}{argmax}\ score(x_i, y)$
        $y_{prediction} = \underset{y \in\ n\ best\ for\ x_i}{argmax}\ cost(y_i, y)$

        if $y_{prediction} \neq y_{oracle}$ then
            $\lambda += h(x_i, y_{oracle}, d_{oracle}) - h(x_i, y_{prediction}, d_{prediction})$
        end if
    end for
end while

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 [17]. There is also an implementation of Perceptron for Joshua [24].

2.3.2 MIRA

MIRA is first formulated in [9, Crammer et al.]. 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 [12]. Some implementations select hope and fear hypotheses from n-best lists [6] while others do cost-augmented and cost-diminished decoding [17].

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 would be no need for quadratic programming [9]. 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 bellow:

The algorithm for online 1-best MIRA [12]:

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

$\lambda_i = 0$ for all i
while not converged do //alternatively we could define a specific number of iterations
for all x_i in x do
$y_{hope} = \underset{y \in Y(x_i)}{argmax}\ score(x_i, y) - cost(y_i, y)$
$y_{fear} = \underset{y \in Y(x_i)}{argmax}\ score(x_i, y) + cost(y_i, y)$
$margin = score(x_i, y_{fear}) - score(x_i, y_{hope})$
$cost = cost(y_i, y_{fear}) - cost(y_i, y_{hope})$
$loss = margin + cost$
if $loss > 0$ then
$\delta = min(C, \frac{loss}{\|h_i(x_i, y_{hope}, d_{hope}) - h_i(x_i, y_{fear}, d_{fear})\|})$
$\lambda += \delta(h(x_i, y_{hope}, d_{hope}) - h(x_i, y_{fear}, d_{fear}))$
end if
end for
end while

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 [6, Cherry et al.] they use n-best lists instead of cost-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 as given in [6]:

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}
$t = 1$
$\lambda_{t,i} = 0$ for all I
$j = 0$
while not converged do //alternatively we could define a specific number of iterations
$j += 1$
for all x_i in x do
$y_{hope} = \underset{y \in Y(x_i)}{argmax}\ score(x_i, y) - cost(y_i, y)$
$y_{fear} = \underset{y \in Y(x_i)}{argmax}\ score(x_i, y) + cost(y_i, y)$
$margin = score(x_i, y_{fear}) - score(x_i, y_{hope})$
$cost = cost(y_i, y_{fear}) - cost(y_i, y_{hope})$
$loss = margin + cost$
$\delta = min(C, \frac{loss}{\|h_i(x_i, y_{hope}, d_{hope}) - h_i(x_i, y_{fear}, d_{fear})\|})$
$\lambda_{t+1} = \lambda_t + \delta(h(x_i, y_{hope}, d_{hope}) - h(x_i, y_{fear}, d_{fear}))$
$t = t + 1$
end for
$\lambda_j^{avg} = \frac{1}{j} \sum_{t'=1}^{j} \lambda_{t'}$
end while

2.3.3 Rampion

Together with the batch version of MIRA [6], Rampion [15] is one of the most recently developed algorithm for discriminative training in machine translation. Because the ramp loss function is non-differentiable and non-convex, standard gradient based methods are unapplicable. Rampion avoids this problem by using concave-convex procedure (CCCP) [41]. CCCP is an 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. [15, Gimpel and Smith] 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 report improvement over MERT and PRO in using both small and large set of features.

The algorithm for batch Rampion [15]:

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

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

2.3.4 PRO

PRO [18] 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 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 [18], 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 below. When we sample hypotheses pair (y^+, y^-) with corresponding feature vectors (h^+,h^-) where y^+ has lower cost than y^-, we generate two training instances for the classifier: the positive one $(h^+ - h^-, +)$ and the negative one $(h^- - h^+, -)$. After we sample 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.

The algorithm for batch PRO [18]:

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

$\lambda_i = 0$ for all i
while not converged do

    //Generation of candidates
    $E = ()$ // set of tuples (x,y,h,y_{ref})
    for all x_i in x do
        $\varepsilon \leftarrow Decode(x^{(i)}, \lambda)$
        for all do
            $E+ = (x_i, y, h, y_i)$
        end
    end

    //Sampling of candidates
    T = ( ) //training instances
    for $j \leftarrow 1$ *to* M do
        sample (x', y', h', y'_{ref}) and $(x'', y'', h'', y''_{ref})$ uniformly where $x' = x''$
        if $|cost(y'_{ref}, y') - cost(y'_{ref}, y'')| > threshold$ then
            $T+ = (h' - h'', sign(cost(y'_{ref}, y') - cost(y'_{ref}, y'')))$
            $T+ = (h'' - h', sign(cost(y'_{ref}, y'') - cost(y'_{ref}, y')))$
        end
    end

    //Training
    $\lambda = train\_classifier(\lambda, T)$
end while

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 tuning data size might be big. To speed up optimization parallelization can be very useful. The way it is done often is by splitting tuning data in to shards of equal size and then do tuning with them on each iteration independently. After the end of each iteration all feature weights vectors of parallel processes are collected and average vector is computed. A next iteration continues the same optimization with using averaged vector as a starting point on each of the shards [17, 16].

Whenever training is done with a complex model as in our case with large number of features there is danger of overfitting. That is why we should have one additional corpus for selecting the weight vectors from all iterations with the best performance [17]. In machine translation this additional data is usually called development data, but because that term is already used for tuning data we will call this corpus selection corpus.

It is expected that for training large number of features we will need a lot of tuning data. A lot of tuning data may be useful, but it is not necessary for getting reasonably good results. Actually, most of the research in large-scale discriminative training is done using a tuning corpus witch is of the similar size as tuning corpus for MERT (around 2000 sentence pairs). We have not found any detailed explanation why it is the case that even such a small tuning corpus is enough for training large number of features. We think that the reason for small corpus giving good results might be Zipf's law. Features that are very important will appear often in the tuning corpus no matter what is the size of a tuning corpus while rare, unimportant features will not appear and if they appear algorithms with good regularization will prevent them form having a big influence.

3. Objective function

In machine translation, we usually want to make our system's resulting translation be judged by humans as a good result. The problem with human judgment of MT systems is that it is too slow. Ideally we would 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 would want to have an objective function towards which we could optimize our system automatically by using algorithms that were described in the previous chapter. A large amount of research has been conducted on the development of automatic evaluation metrics that have a good correlation with human judgment for comparison by quality of two (or more) translations of the same corpus given the corpus with reference translations. The problem with using these 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[22].

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 a result of the evaluation function, we get a number that represents how much the system's translation matches the reference translation. The better the scores of systems given by the evaluation metric correlate with human judgment, the better is the metric.

Except 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 higher correlation with human judgment than BLEU by paying the cost of using 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, like Czech for example, 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 [20] (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:

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

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

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

This metric is not used often in machine translation projects but it has some good aspects. Firstly, 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 that this metric has. The list of these metrics include ROUGE-S and our adaptation of it that we will explain later.

The simplicity of f-measure makes it also useful in 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 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 would 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 [33] is a precision-based metric, which is based on computing 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 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:

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

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

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:

$$\begin{aligned} BLEU4 &= \text{brevity-penalty} \exp \sum_{i=1}^{4} \log precision_i \\ &= \min \left(1, \frac{\text{system's translation length}}{\text{reference translation length}}\right) \prod_{i=1}^{4} precision_i \end{aligned} \quad (3.1)$$

One of the problems with BLEU is that if 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 3 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 is computed on the whole corpus, which is very unprobable to be 0. This strategy gives really 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 a 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 [26]. 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 [7, Chiang et al.]. 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 Of an exponentially moving length of input:

$$O \leftarrow 0.9(O + c(e))$$
$$O_f \leftarrow 0.9(O_f + |f|)$$

Finally BLEU approximation is computed as:

$$B(e; f, r) = (O_f + |f|)BLEU(O + c(e; r))$$

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

3.1.3 NIST

NIST [11] tries to repair the assumption that is made in BLEU, that all n-grams of the same order are equally important. NIST tries to reward the translations which have 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.

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

$$NIST = \sum_{n=1}^{N} \left\{ \frac{\sum_{all\ w_1 \dots w_n\ that\ cooccure} Info(w_1 \dots w_n)}{number\ of\ w_1 \dots w_n\ in\ the\ system\ translation} \right\} * \exp \left\{ \beta \log^2 \left[\min \left(\frac{L_{sys}}{L_{ref}}, 1 \right) \right] \right\} \quad (3.2)$$

3.1.4 METEOR

METEOR [10] 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 [20]. 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) [22] is a metric that was specifically designed to do evaluation of Czech output. It is based on the semantic role overlapping metric from [14]. 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 [40] framework. Also, instead of surface word form, t-lemma from TectoMT [40] 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 really low correlation with human judgment on the sentence level which is similar to BLEU sentence level performance [22].

3.1.6 ROUGE-S

ROUGE-S [26] is a variation of f-measure described before. Instead of matching words ROUGE-S tries to match skip-bigrams which is 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 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.

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

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

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

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:

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

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 is the number of allowed skips, the more value we give to the word order. In their work [26, Lin and Och] have found that for English the best number of allowed skips is 4.

3.1.7 ROUGE-SX

The problem that we have found with ROUGE-S is that if you restrict the number of allowed skips, 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 nmber 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.

Our formula for computing the correct number of skip-bigrams with a defined maximal number of allowed skips is given bellow:

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

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 really 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 in 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 more the size of the sentence rises, the larger is the difference between the correct number of skip-bigrams and the number of combinations. This leads to two problems with the original ROUGE-S metric:

- 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 original ROUGE-S will be 4950 while for our method 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-S4 as given in [26, Lin and Och]. 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, 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 this pseudo code for function:

SKIP2 (X, Y)
  XH // hash map that has skip-bigram of sentence X
      //as a key and its count in sentence X as a value
  YH // hash map that has skip-bigram of sentence Y
      //as a key and its count in sentence Y as a value

  foreach skip-bigram x in X do
    $XH\{x\}+=1$

  foreach skip-bigram y in Y do
    $YH\{y\}+=1$

  $matched\_bigrams = 0$

  foreach x in keys XH do
    $matched\_bigrams+ = \min(XH\{x\}, YH\{x\})$

  return *matched_bigrams*

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-S4 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-S4 with wrong definition of SKIP2 because with the wrong definition the optimization algorithm will prefer output that is long filled with repetitions of correct translations, which is a really bad result in the end. With our definition that problem cannot appear because we are recognizing the most minimal number of matched skip-bigrams and the system is punished for long output.

3.2 Motivation for using ROUGE-S4 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 [22]. 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

[23, Bojar and Kos] 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 factorized 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 a 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 achieved 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 really bad correlation with human judgment on the sentence level [22] and bad for discriminating similar translations [23] 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 [22]. As an alternative to optimization towards BLEU, the most popular choice is sBLEU which was first published in [26, Lin and Och]. 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.

[22, Bojar and Kos] 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 [26, Lin and Och] and [22, Bojar and Kos] led us to the idea that ROUGE-S4 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 [22, Bojar and Kos] and in [26, Lin and Och] ROUGE-S4 is even better than NIST and sBLEU, which is the most popular choice for large-scale discriminative training, then ROUGE-S4 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-S4 is that it is a simple metric that does not require remembering the history of the previous evaluations like BLEU approximation used in [7, Chiang et al.]. 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-S4 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.

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 space for usage and research of a large number of features for modeling the translation process. However, these algorithms are not a sufficient condition for using features with rich linguistic information. Some of the problems are related to the way 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 that 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 [36]. What a discriminative language model does is to give 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 by 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 by 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 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 bad score during tuning which will make its weight in the discriminative model very small or even negative so it would not cause a low 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 then its surface form. Some of those more abstract representations can be:

- lemma
- stem
- POS tag
- cluster id from clustering like in [31, Och]
- affix (usually suffix for languages like Czech)

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 [29, Niehues et al.] 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 [31, Och]. 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 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 that is 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 [37, Rosa] they have created a parser for processing translation output by using not only features that are 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*n-best size times, which is a large number considering how slow the parsing of one sentence can be. This makes usage of parsers and similar classifiers a bad option for discriminative training.

A similar problem was encountered in [23, Bojar and Kos], 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 in 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 assign proper noun tag. This technique ignores context, but even without considering context it gets around 90% accuracy with languages like English [5]. The second replacement for POS tags is to use cluster ids instead. Clusters can approximate POS tagging if a reasonable number of clusters (depends 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. One is technical and the other one is is 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 will try to offer possible solutions to these problems, but without experimental evidence for them, which is left for the future work.

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 special a parser in order to get better results [37] and it can take a lot of time to tune with parsing each sentence in an n-best list [23]. 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 syntax translation [13] or dependency treelet translation [35] 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. We will suggest two solutions of which the first one is based on mapping source dependency tree to the target side and the second solution 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. Because during tuning, we already have phrase-to-phrase alignments 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. Also this process does not have to be done during tuning, but much earlier when the phrase table is generated and after that these alignments can just be loaded into memory. This will take more memory during tuning, but it will be faster since alignments will not have to be recomputed.

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 [35], 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.

Heuristics for mapping start 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 each of them 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 using 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, but 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 $f \rightarrow e$, 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 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 Optimality theory [34] 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 components [39]:

- 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 a parallel to the generative model in statistical machine translation 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 the machine translation literature and it was mentioned in only two papers by the same author [27, 28] in the human translation literature.

5. Experiments, results and discussion

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

1. a large number of word forms that are present in the test data are not seen in the training data [23]
2. even if a word form is seen in the training data, choosing the right word form can be hard [23]
3. BLEU as an evaluation metric is not really good in case of evaluating morphologically rich languages [22] and we expect that its approximation that is often used in large-scale discriminative training will give even lower results

Part of the plan of this thesis was to develop rich features by working on English-Serbian as a language pair so they could be applied later to the English-Czech language pair. The reason for working with Serbian was author's better understanding of it compared to his understanding of Czech. Since we have established that phrase-based SMT systems are still not suited for rich features we have decided to work with not linguistically rich, but sparse features that are well supported under phrase based SMT systems. Focusing on features that do not require deep linguistic knowledge need for working on English-Serbian language pair has disappeared and made us focus only on English-Czech language pair. Working directly on English-Czech language pair allow us to incorporate rich features in our existing work when support for them in phrase based systems becomes available. The effect on final result of all the mentioned problem on English-Czech language pair is given in [23, Bojar and Kos]. 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 data because this problem can in most cases be solved only by getting more data. The choice of experiments that are conducted is based on the problems we have selected to explore.

5.1 Models used in all experiments

All conducted experiments share some common infrastructure. For training data we have used news section of the CzEng 1.0 parallel corpus [4]. As a tuning, testing and selection corpora we have used WMT10, WMT11 and WMT12 corpora respectively. The language model is built using Srilm language modeling toolkit [38] on the Czech side of the training corpus. Training data is first aligned using GIZA++ [30] on the lemmatized forms and after that translation model is created using Moses PB-SMT toolkit [21]. All optimization algorithms that we use are part of the Moses toolkit. Batch MIRA and MERT are from the main git branch and online MIRA is from the miramerge git branch. As a baseline we use weights optimized by MERT.

As noted in [8, Clark et al.] most of the optimization algorithms for the discriminative models in machine translation have some randomness and because of that it is required to repeat some experiments several times in orger to get statistically significant result. What they suggest in case of MERT is to run MERT at least three times and then compute the average score and standard deviation. That approach could also work for batch version of MIRA since it shares some of the MERT architecture. For online MIRA this approach does not work because, unlike in MERT, randomness in online MIRA does not come from random starting points (usually initial weight vector is 0) but from ordering of training instances. That is why in addition to the repetition of tuning several times we have also shuffeled tuning data before each tuning.

With this thesis we have also supplied exact tuning data that was used (both original and shuffeled versions), testing data and end translations of all optimized models. Also we have supplied code that we have developped for handling some features and code for different objective functions.

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

- DLM with unigrams
- DLM with bigrams
- DLM with unigrams and bigrams
- PP feature

Results are shown in the table bellow.

	Number of features	Avg. BLEU score on the testing data	Standard deviation
MERT core	8	12.37	0.02
DLM1	9409	12.34	0.08
DLM2	42901	12.41	0.04
DLM1+DLM2	42529	12.34	0.04
PP	17641	12.40	0.06

Even though the tuning corpus was small compared to the size of the corpora that is 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 were a larger number of parameters, results are even better, which leads us to think that data size was not a big problem in this case.

DLM2 and PP have the best results from all features that were tested. 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 order. PP influences the word order because phrases can be of size up to 7 words. We have tried training DLM3 feature, but this feature is too sparse to be trained in reasonable time.

Another reason why bigger context that is covered by DLM2 and PP is important is choice of the word form. For example the choice of case for some noun depends from its neighbouring words. In this cases DLM1 cannot help at all while DLM2 and PP can cover these dependences. Deeper exploitation of these features might lead to better choice of word forms and better choice of word order. In total, improvements that were acheaved with usage of large number of features are not big which corresponds to the other similar research [17].

5.3 Experiments for solving the problem of evaluation in tuning

For solving this problem, we have tested three different objective functions with online MIRA:

- BLEU approximation [7]
- ROUGE-S4 - our implementation of [26, Lin and Och]
- ROUGE-SX4 - our changed version of ROUGE-S4 that was explained in the earlier text

Additionally, we have used also a recent addition to the Moses toolkit batch version of MIRA [6] with BLEU approximation as an objective function. All these optimizations were done on the same model as previous experiments. As a baseline, again, we use the same parameters optimized by MERT. All optimizations are only performed on core features. Results are shown in the table bellow.

	Learning rate	Avg. BLEU score on the testing data	Standard deviation
MERT - BLEU		12.37	0.02
Batch MIRA - approx. BLEU	1.0	12.43	0.05
Online MIRA - approx. BLEU	1.0	12.44	0.06
Online MIRA - ROUGE-S4	1.0	12.32	0.07
Online MIRA - ROUGE-S4	0.1	12.34	0.26
Online MIRA - ROUGE-SX4	1.0	12.23	0.38
Online MIRA - ROUGE-SX4	0.1	12.01	0.59

Against our expectations, the optimization with ROUGE-S4 was better than our version of it (ROUGE-SX4). We think that the reason for this is that more conservative changes seem to give better results. The way we have defined the margin between two hypotheses is that it depends from the evaluation metric in one part. When the value of that metric is very small (in general, not for some particular hypothesis) then the margin between hypotheses is also small. As we have shown earlier, scores that are produced by ROUGE-S4 are very small compared to ROUGE-SX4 scores, especially in the case of long translations. Because scores are small, the margin will be small too. It will be small especially in the cases of long translations where the system is likely to give some bad results, so it might not be good to give big weight to that update. When we enforce this margin in the update, small changes to the parameters will be applied.

Similar effect can be achieved by changing the learning rate of the algorithm. Lowering the value of the learning rate from 1 to 0.1 has confirmed our expectations, in case of ROUGE-S, that more conservative changes give better results. Batch MIRA gave similar results to the online MIRA in both the average score and standard deviation. From the perspecitve of using different objective functions batch MIRA might be interesting because it does not require evaluation of partial hypothesis. Selection of hope and fear hypotheses is done on the word lattice which contains complete hypotheses so evaluation can be more precise no matter which metric are we using. This property of batch MIRA makes it attractive for testing our objective functions, but because batch MIRA was a recent addition to the Moses toolkit we did not have enough time to port our metric to it. We leave this for further research.

There is a difference between conservativism of low learning rate and ROUGE-S metric. The low learning rate makes the whole learning conservative, while ROUGE-S4, except for being conservative because of its low score, is conservative even more in the cases of long translations. Even though our metrics did not give better results than BLEU approximation we think that experiments have shown that conservative updates can be better in some cases, especially in the cases of long translations. Adding some scaling factor to ROUGE-SX4 or BLEU (or any other tuning metric for large-scale discriminative training) that depends on the sentence length might improve its effectiveness in the tuning.

Conclusion

In this work, 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 in the task of translating from English into Czech. Some of the ideas that were used here are novel and applicable not only to the task of translation to morphologically-rich languages, but also to the more general framework of large-scale discriminative training.

We have tested different features using a phrase-based SMT system. In the cases of features that could not be used with standard phrase-based systems, we have given reasons why this is the case and gave suggestions for future research on how these problems might be overcome. From the linguistic point of view, we have also shown how different features influence the choice of a word form and how they could be improved in the speed of their computation and in making them less sparse. We have pointed out the fact that linguists have already worked on a theory that has many similarities with discriminative training in machine translation – optimality theory – and that, when phrase-based SMT systems become capable of handling rich features, we should try to apply ideas from that field of linguistics in the engineering of the features.

This thesis is also the first to use the ROUGE-S evaluation metric as an objective function in discriminative training. It did not give better results than the approximation of BLEU that is used in most of the systems for large-scale discriminative training, but experiments with it showed that conservative updates might be more useful especially in the cases of long translations. We have also introduced our own variation of the ROUGE-S metric, which gave similar results. This is the first time these metrics were used as objective functions in discriminative training, so we think it is worth to invest future research into them if not for them to become a replacement for BLEU then at least to give insights how to better use BLEU in the context of large-scale discriminative training.

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, 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.

[5] Eugene Charniak. Statistical techniques for natural language parsing. In *AI Magazine Volume 18 Number 4*, 1997.

[6] Colin Cherry and George Foster. Batch tuning strategies for statistical machine translation. In *NAACL*, June 2012.

[7] David Chiang, Yuval Marton, and Philip Resnik. Online large-margin training of syntactic and structural translation features. In *EMNLP '08 Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 224–233, 2008.

[8] 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.

[9] 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.

[10] 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.

[11] George Doddington. Automatic evaluation of machine translation quality using n-gram co-occurrence statistics. In *Proc. ARPA Workshop on Human Language Technology*, 2002.

[12] 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.

[13] 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.

[14] 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.

[15] Kevin Gimpel and Noah A. Smith. Structured ramp loss minimization for machine translation. In *NAACL*, 2012.

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

[17] 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.

[18] 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.

[19] R. 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.

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

[21] 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.

[22] 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.

[23] Kamil Kos and Ondřej Bojar. 2010 failures in english-czech phrase-based mt. In *Proceedings of WMT'10*, 2010.

[24] Zhifei Li and Sanjeev Khudanpur. Forest reranking for machine translation with the perceptron algorithm.

[25] 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.

[26] 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.

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

[28] 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.

[29] 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.

[30] F. J. Och and H. Ney. Improved statistical alignment models. pages 440–447, Hongkong, China, October 2000.

[31] 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.

[32] 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.

[33] Papineni, Kishore, 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.

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

[35] Chris Quirk, Arul Menezes, and Colin Cherry. Dependency treelet translation: Syntactically informed phrasal smt. In *Proceedings of ACL*. Association for Computational Linguistics, June 2005.

[36] 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.

[37] 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.

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

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

[40] Zdeněk Žabokrtský, Jan Ptáček, and Petr Pajas. Tectomt: Highly modular mt system with tectogrammatics used as transfer layer. In *Proceedings of WMT'08*, 2008.

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