Towards Efficient Minimum Bayes Risk Decoding

MSc Thesis (*Afstudeerscriptie*)

written by

Gerson Foks
(born August 8th, 1995 in Soest, Netherlands)

under the supervision of **Dr Wilker Ferreira Aziz** and **Bryan Eikema MSc**, and submitted to the Examinations Board in partial fulfillment of the requirements for the degree of

MSc in Logic

at the *Universiteit van Amsterdam*.

Date of the public defense:	**Members of the Thesis Committee:**
2023-02-24	Dr. Katia Shutova
	Dr. Vlad Niculae
	Mario Giulianelli MSc

INSTITUTE FOR LOGIC, LANGUAGE AND COMPUTATION

Abstract

Minimum Bayes risk (MBR) decoding for machine translation is getting renewed attention as it is a principled approach to neural machine translation that exhibits fewer pathologies than the widely adopted maximum a posteriori (MAP) decoding. Estimating the MBR objective, however, can be a costly endeavor. In this thesis, we explore two different approaches for estimating the MBR objective. Lastly, we try to construct the MBR translation by backpropagating through the utility function. The main approach is to predict the Bayes Risk with the help of neural models. With this approach, we train a model to regress to an accurate Monte Carlo (MC) estimate of the Bayes risk. At inference time we can directly estimate the Bayes risk with the trained model, circumventing expensive MC estimation. These models outperform the m-MC estimates in predicting the Bayes Risk for low values of m. Furthermore, they outperform m-MC estimates in terms of both computation speed and quality of the translation when the models are used as a decision rule. The second approach is to fit a mixture of Gaussians or students-t to the distributions of the utilities. This approach achieves similar results as simply regressing to the MC estimate. The last approach is to construct the MBR translation directly with the help of backpropagation. With this approach, we backpropagate through the (neural) utility function to find the translation that achieves the lowest Bayes risk. This approach however resulted in nonsensical translations. Our hope is that these three approaches give an insight into how we could get towards more efficient minimum Bayes risk decoding.

Contents

List of Figures

List of Tables

Introduction 1

In this chapter, we will motivate our research, discuss the research questions, mention our contributions and give an outline of this thesis.

1.1 Motivation

With *neural machine translation* (NMT) a neural network is trained to translate a given *source* sentence in one language to an appropriate translation in another language. Often this is done by having the model learn to assign a probability to a translation given the source. In that case, the NMT model does not immediately tell us what the best translation is: it only gives us access to a probability distribution over all the translations. The way we get a translation from the NMT model is known as a *decision rule*.

One popular decision rule is *maximum a posteriori (MAP) decoding*, in which the most probable translation (the *mode*) is used [1]. A major issue with the mode and other high probable translations is that, in practice, it is often inadequate, as they are often a poor translation of the source [1, 2]. Approximating the mode is done with *beam search* [3]. The search itself also introduces biases, one such bias is that longer searches do not improve translation quality as a result of the inadequacy of the mode[4, 5].

1: *Posterior probability* is a concept from Bayesian statistics, although we don't give a Bayesian treatment to our models, we will use this name as the terminology has already been established in Natural Language Processing literature.

An alternative approach that does not exhibit most of the pathologies that MAP decoding has is *minimum Bayes risk (MBR) decoding* [2, 6, 7]. However, a drawback of MBR decoding is that it is intractable and previously explored approximations are computationally expensive [2, 7].

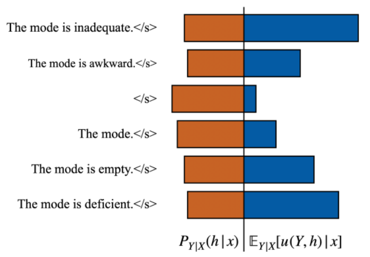


Figure 1.1: An example and illustration of the inadequacy of the mode given by Eikema and Aziz [7]. The NMT model spreads probability mass over several hypotheses (left) and makes them hard to distinguish in terms of probability. While MBR (right) assigns each of these candidates an expected utility, creating a better distinction between good and bad translations.

The concept of minimum Bayes risk stems from decision theory [8] and when applied to machine translation it states that the optimal translation y^* under a probability distribution with probability mass function $p(y|x,\theta)$ parameterized by θ is given by[2]:

$$y^* = \operatorname*{arg\,max}_{h \in \mathcal{H}} \mathbb{E}_{p(y|x,\theta)}[u(Y,h;x)] \tag{1.1}$$

in which $\mathcal{H}$ is the set of all possible translations, x is a source sentence, y is a translation and u is a utility function which assigns an utility to h given y which optionally takes the source x into account. The utility function used measures the similarity between h and y. When the source x is also involved, it measures the quality of the translation h for x when y is a reference translation. When u captures lexical similarity, such as when u is unigram-F1 or chrf++ [9, 10], MBR decoding finds the translation that is in expectation the most lexically similar to all the other possible translations under the model. When u is a measure of quality, such as when we use COMET [11] as a utility, MBR decoding finds the

2: We use ';' to split required arguments from optional arguments translation that has the highest quality in expectation when using all the other translations as reference translations.

Finding y^* is intractable as $\mathcal{H}$ is unbounded and $p(y|x, \theta)$ does not make independence assumptions so approximation schemes are needed. An approximation proposed by Eikema and Aziz [7] is using *Monte Carlo (MC) estimates* in which m reference samples are used. However, MC estimates are expensive in practice. In this research, we explore three different approaches that could lead to more efficient MBR decoding.

1.2 Approaches

In this thesis, we explore three different approaches for more efficient MBR decoding. We first treat estimating the Bayes risk as a regression problem. Secondly, we use probabilistic modeling to first predict the distribution of the utilities for a given hypothesis and use this distribution to predict the Bayes risk. Finally, we use backpropagation through the utility function to directly find y^* given by equation 1.1

Our main approach is to treat estimating the Bayes risk as a regression problem and train a predictive model instead of resorting to the MC estimate. We precompute robust estimates of the Bayes risk by calculating a 1000-MC estimate. We train several *predictive neural models* to predict those 1000-MC estimates. Predicting the Bayes risk with these trained models could be less computationally expensive than the robust MC estimate. We give these neural models access to features that are indicative of the quality of the translation or tell us something about the distribution of the translations for a given source such as the hidden states of the NMT model. These features could be predictive of the Bayes risk and, as MC estimates do not have access to these features, the predictive models could outperform MC estimation. After we trained our predictive models we can construct the MBR translation by using the predictive model as a decision rule: we predict the Bayes risk for a set of translations and pick the one with the lowest risk according to our model.

The second approach we take is to frame the problem as a probabilistic modeling problem in which we want to model the distribution of the utilities for a given hypothesis $p(u(y, h; s))$. From this probabilistic model, we can directly estimate the Bayes risk. We use a neural network to predict the parameters of a mixture model. Besides the features that the neural network has access to that could be predictive of the Bayes risk we also give information about the distribution of the utility function during training. The information about the distribution is not present when simply regressing to the MC estimate and could help in predicting the Bayes risk.

For the last part of this thesis, we treat finding the MBR translation as an optimization problem for which we use backpropagation and gradient descent. We update a hypothesis h to maximize equation 1.1 in multiple steps by using backpropagation with respect to some differentiable utility function u and a list of sampled references. As words (and the resulting tokens) of a translation are discrete we need some kind of relaxation. Both optimizing directly in the embedding space and continuous-discrete relaxation are used for this. Constructing the MBR translation this way could be more efficient than calculating the Bayes risk for multiple hypotheses as, hopefully, we don't need as many evaluations of the utility function as when doing MC estimation.

1.3 Research Questions

This thesis centers around the three previously mentioned approaches. To structure the research we focus on 3 main questions each with its subquestions.

Predictive models

Q1: Can neural models be used to accurately and efficiently predict the Bayes risk?

To narrow the focus we have three sub-questions:

Q1.1: How does the choice of utility function influence the predictive model?

The Bayes risk depends on the utility function that is chosen and therefore we want to explore how the choice of utility function influences the predictive model. For this, we use three utility functions: *unigram-F1*, *chrF++*, and *COMET*. We study how good of a substitute the predictive models are for the m-MC estimate of the Bayes risk.

Q1.2: Which features are important in approximating the Bayes risk?

To construct a good approximation it is practical to know which features are predictive of the Bayes risk. Some features are more cheaply computable than others, which may introduce a trade-off between quality and speed. Different type of features exists, there are NMT model-related features, such as the hidden state and statistics of the tokens. There are utility-related features such as the embeddings used in COMET. Lastly, we could also give access to sampled references and features of those samples. We compare the capabilities of the different models to get an idea which features are important for predicting the Bayes risk.

Q1.3: How well do the models perform in practice?

To know if it is practical to use neural models to approximate the Bayes risk we analyze the models' capability to predict the Bayes risk, the ability to rank the translations and the speed of inference. Lastly, we analyze the usefulness of the models as a decision rule in which we measure the quality of the translations with the help of the previously mentioned utility functions and BLEURT [12].

Mixture models

Q2: Can neural mixture models be used to predict the Bayes risk?

To find out if this is possible and get a sense of how well this method works compared to the regression models we look at two subquestions. The models we trained are based on the best-found model from our previous question. We analyze the resulting models for the COMET utility function.

Q2.1: How do mixture models compare to the predictive model?

To see if it is worth predicting the parameters of a mixture model instead of directly predicting the Bayes risk we compare the mixture models with the previously found predictive model. For this analysis, we look at the mean squared error of the predicted Bayes risk and the quality of the translations when we use the mixture model as a decision rule.

Q2.2: How does the choice of distribution for the mixture model influence the mixture model?

When creating a mixture model we need to choose the distribution that is used. We compare the fit and the quality of the translations for two different distributions, namely the Gaussian distribution and the student-t distribution. The comparison is done both visually, in terms of the fit, and as well as the quality of the translations.

Backpropagation

Q3: Can backpropagation through a utility function be used to construct the MBR translation?

We construct the translations with help of backpropagation. For this, we use two utility functions: COMET and a sentence similarity network from Hugging Face called "all-distilroberta-v1". With backpropagation, we can use several methods to construct the MBR translation. To check the quality of the constructed translations we do a manual inspection. Furthermore, we compute the utility with respect to some reference translation. We focus on two methods and get two subquestions:

Q3.1: Can the MBR translation be constructed by updating the tokens with the help of continuous-discrete relaxation and straight-through estimation?

3: Meaning that the entries of the vector can take arbitrary real values, which can be turned into probabilities with help of the softmax function

With this method we relax the one-hot encoding of the tokens to "soft"-encodings [3]. With the help of straight-through estimation, we update the soft-encoding.

Q3.2: Can the MBR translation be constructed by updating the tokens in the embedding space of the utility function?

As straight-through estimation is biased we try a second method in which we update the tokens directly in the embedding space. After the optimization stabilizes we find the nearest token for each timestep and use that to construct the final translation.

1.4 Contributions

To summarize our contributions are as follows:

1. We show that neural models can be used to predict the Bayes risk, but they do not perform better than MC estimation. When they are used as a decision rule they outperform MC estimation;
2. We show mixture models can be used to predict the Bayes risk but do not perform better than regression models;
3. Lastly we show that backpropagation through the utility function does result in nonsensical translations and therefore does not seem to be fit to be used to construct the Minimum Bayes risk decoding.

1.5 Outline

In this thesis, we will first introduce most of the relevant concepts *Background*. In this chapter, we discuss NMT, decision rules, and the utility functions used. We finish the background chapter by discussing the pathologies observed in MAP decoding.

In the chapter *Predictive Models* we look at how we constructed the predictive models. It discusses the design decisions that come into play as well as the first validation results of the predictive models.

A more in-depth analysis of the predictive models is done in the chapter *Analysis*. We analyze the models both in *quality* [4] and *speed*. This analysis is used to answer Q1 and its subquestions.

4: In terms of mean squared error, ranking capability and usability as a decision rule

The next chapter *Mixture models* builds upon the previous two chapters and answers research question Q2. We start this chapter by discussing mixture models and the relevant background. Then we focus on the fit and the quality of the translations that are constructed using the mixture model.

In the chapter *Minimum Bayes Risk Decoding with Backpropagation* we look at question Q3. We try to construct MBR translations using backpropagation. A few examples are highlighted and we discuss why this approach fails.

In the final chapter *Conclusion* we summarize the results one more time and lay out future work. Lastly, in the *Appendix* we include additional training details and some technical details that are important when working on this problem.

Background 2

This chapter gives general background information about machine translation, neural models, decision rules and the utility functions used in our research. Furthermore, the pathologies of NMT decoding are discussed. This chapter finishes by discussing other applications of Minimum Bayes risk, such as its use in natural language generation

2.1 Preliminaries

In this section, we discuss the Preliminaries. We start by introducing the neural machine translation framework. Then we introduce the neural architectures used as well as the NMT model used. We then discuss the decision rules as well as Minimum Bayes risk decoding. Lastly, we talk about the utility function used as well as the methods we use to evaluate the models.

2.1.1 Machine Translation

Given a source sentence $x \in X$ from the set of all possible source sentences, a trained NMT model parameterized by θ predicts the conditional distribution of a translation $y \in Y$:

$$p(y|x, \theta) \tag{2.1}$$

Often auto-regressive models are used such as in Cho et al. [13], Sutskever, Vinyals, and Le [3] and Kalchbrenner and Blunsom [14] [1]. Auto-regressive models use the *chain rule of probabilities* to factorize the distribution into a chain of random draws. Let $y = (y_1, \dots y_m)$ be a sequence of m symbols belonging to a vocabulary of known target-language symbols, we get:

1: Alternatives exists to auto-regressive machine translation, appropriately called non-autoregressive machine translation.

$$p(y|x, \theta) = \prod_{i=1}^{m} p(y_i|x, y_{<i}, \theta) = \prod_{i=1}^{m} f(x, y_{<i}, y_i, \theta) \tag{2.2}$$

In which $y_{<i}$ is the sequence of tokens before the ith token. $p(y_i|x, y_{<i}, \theta)$ is the probability of symbol y_i given the parameters of the NMT model, the previous symbols, and source sentence x. Finally, $f(x, y_{<i}, y_i, \theta)$ is the function representing our NMT model. It is assumed that the symbols follow a categorical distribution. Auto-regressive models, therefore, learn a categorical distribution for the next token given the source and the previous tokens.

2.1.2 Architectures

For machine translation and its related tasks different types of neural networks are used such as *recurrent neural networks (RNN)* [15, 16], *attention-*

based networks [17], *convolutional neural networks* [18] and *graph neural networks* [19]. We focus on the first two as they are commonly used.

Recurrent neural networks process a variable-length sequence $x = (x_1, x_2, \dots x_T)$ by maintaining a hidden state h over time. Then at each timestep t the hidden state $h^{(t)}$ is updated [20]. There are different RNN-based architectures. These architectures often use either *Long-short term memory (LSTM)* components [15, 16] or *Gated Recurrent Unit (GRU)* [20] components.

Alternatives to RNNs are attention-based models, which process the whole sequence at once. The basic building blocks of modern attention models are *Scaled Dot-Product Attention* and *multi-head attention*, which were first introduced in Vaswani et al. [17]. With this type of attention, tensors called *Query (Q)*, *Keys (K)* and *Values (V)* are used as input. For each vector in Q we get a vector of weights based on K. This weight vector is then used to get a weighted average of the vectors in V. Although attention-based models often times reach state-of-the-art results, they have as a drawback that they require a quadratic amount of memory with respect to the input size.

2.1.3 Decoding

When we have an NMT model, we want to actually translate a given source sentence x. A first *decision rule*, that is, a way of deciding which translation of all possible translations $\mathcal{H}$ to pick is by selecting the *mode*:

$$y^{\text{mode}} = \underset{h \in \mathcal{H}}{\arg\max}\, p(h|x, \theta) \tag{2.3}$$

This search is intractable as $\mathcal{H}$ is unbounded (and the NMT model makes no independence assumptions). To approximate the mode oftentimes beam search is used:

$$y^{\text{beam}} = \underset{h \in \text{beam}(x)}{\arg\max}\, p(h|x, \theta) \tag{2.4}$$

In this setup, we construct b translations, denoted by beam(x). The set beam(x) is constructed by applying a pruned version of breadth-first search. We keep track of b partial translations ordered by the model's log probability. Every incomplete translation is expanded into v new (partial) translations with v being the size of the vocabulary. The b (partial) translations with the highest log probability under the model are kept. This is done until b complete translations are found. The translation with the highest probability is chosen as the final translation [3].

Minimum Bayes Risk Decoding

Minimum Bayes risk comes from decision theory [8] and is based on the principle of maximization of expected utility. A utility function $u(y, h)$ measures the quality of a choice $h \in \mathcal{H}$ when $y \in \mathcal{H}$ is the valid decision. In NMT the utility function u measures a notion of similarity between h and y. When the source s is also involved, it measures the quality of the translation h for some source s when y is a reference translation. As we don't know what the best translation is, we need to decide under uncertainty. An NMT model is used to inform us about the distribution of possible translations. This distribution is used to find the translation with the highest utility in expectation y^{MBR}:

$$y^{\text{MBR}} = \underset{h \in \mathcal{H}}{\arg\max}\, \underbrace{\mathbb{E}[u(Y, h; x) | x, \theta]}_{=:\mu_u(h,x,\theta)} \tag{2.5}$$

As equation 2.5 is intractable, an estimate is needed. Following the work of Eikema and Aziz [2] we will consider the unbiased estimate of MBR using Monte Carlo (MC) estimates. First one obtains N independent samples $(y^{(n)} \sim Y|\theta, x)_{n=1}^{N}$ by drawing samples from the NMT model. For a hypothesis h the N-MC estimate of $\mu_u(h, x, \theta)$ becomes:

$$\hat{\mu}_u(h, x, \theta) \overset{MC}{:=} \frac{1}{N} \sum_{n=1}^{N} u(y^{(n)}, h) \tag{2.6}$$

which is unbiased for any sample size[2]. In the context of MBR decoding $y^{(1)} \ldots y^{(N)}$ are called the references.

2: Interesting to note is that the Bayes risk depends on two things: the **utility function used** and the **distribution under the NMT model** of the translations. Later we will use this to determine the features we want to explore.

From this we get the following decision rule:

$$y^{N\text{-}by\text{-}N} = \underset{h \in \{y^{(1)} \ldots y^{(N)}\}}{\arg\max}\ \hat{\mu}_u(h, x, \theta) \tag{2.7}$$

for which we compare N references with N hypothesis. A problem with this estimate is that it requires $O(N^2)$ evaluation of the utility function.

The paper Eikema and Aziz [7] proposes two methods to speed up finding the MBR translation. First they propose MBR_{NByS} in which they use $S < N$ references decreasing the number evaluations needed to $O(N \cdot S)$. Next they propose MBR_{C2F} (coarse to fine MBR) in which they first use a cheaper proxy utility function to determine N good candidates and use MBR_{NByS} with the target utility function to determine the best hypothesis. In the rest of this thesis we use $m \in \mathbb{N}$ to refer to the number of references used in the MC estimate[3].

3: This to make sure there is no confusion with the n from n-grams (introduced later).

2.1.4 Utility functions

For MBR decoding to work well an appropriate utility function is needed. In MBR decoding the utility function that is used measures the similarity, or when the source is also involved, the quality of a hypothesis given a reference. There are many ways to estimate the quality of a translation. Commonly used metrics for similarity are Bleu [21], Edit distance [22] or chrF and its extensions [9, 10]. There are neural alternatives such as BEER [23] and BertScore [24].

Nowadays there also exist learned (neural) metrics such as COMET [11] and BLEURT [12]. These are trained on datasets in which professional linguists gave quality assessments of translations, such as the WMT shared metric task [25]. For this research, we use COMET as it achieves state-of-the-art results [11]. As a very basic utility function, we use unigram-F1. We also use chrF++ [9] as it is a good performing similarity measure and it is more complex than the unigram-f1 score.

COMET

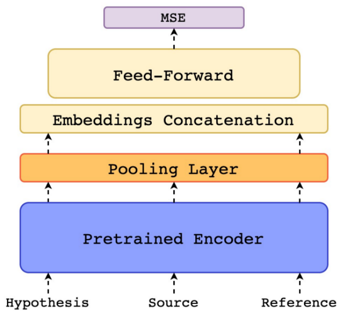


Figure 2.1: The comet Architecture.

COMET is a neural framework for training multilingual machine translation evaluation models [11]. It uses XLM-RoBERTa [26] as a pre-trained multi-language encoder to embed the source, hypotheses, and reference. These embeddings then get pooled and concatenated. The concatenation is forwarded into a feed-forward network which outputs the final prediction. The comet embedding e_{COMET} is constructed as:

$$e_{\text{COMET}} = [e_h; e_r; e_h \odot e_x; |e_h - e_x|; e_h \odot e_r; |e_h - e_r|] \tag{2.8}$$

in which e_x, e_h, and e_r are the embeddings of the source, hypothesis, and reference respectively. The symbol $\odot$ is used to denote the elementwise product. There are different versions of COMET, but we used the one the authors recommend which is *wmt20-comet-da* [27].

n-gram-F1

A much simpler utility function is n-gram-F1, which simply computes the harmonic mean of the precision and recall w.r.t. the n-grams.

Precision captures which percentage of n-grams in the hypotheses are present in the reference n-grams. Let H_n and R_n be the (multi)set[4] containing the n-grams for the hypothesis and a reference respectively.

4: A multiset is a set that allows for multiple instances of the elements. For unigram-F1 we use sets while chrF(++) uses multisets

$$\text{precision} = \frac{|R_n \cap H_n|}{|H_n|} \tag{2.9}$$

Recall captures which percentage of the reference n-grams are present in the hypotheses n-grams:

$$\text{recall} = \frac{|R_n \cap H_n|}{|R_n|} \tag{2.10}$$

The F_1 score is then defined as: [5]

5: F_1 is defined to be zero if both precision and recall are zero

$$F1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \tag{2.11}$$

For this research we use unigram-F1, meaning we pick $n = 1$. The unigrams are based on the tokenization of our NMT model.

chrF++

ChrF++ is an extension of chrF and has good correlation with human relative rankings [9, 10]. The general formula used for chrF and its extensions is:

$$ngrF\beta = (1+\beta^2)\frac{ngrP \cdot ngrR}{\beta^2 \cdot ngrP + ngrR} \tag{2.12}$$

where *ngrP* and *ngrR* stand for n-gram precision and n-gram recall arithmetically averaged of all n-grams used. β is a parameter which assigns β times more weight to recall than to precision [9].[6] The n-grams used for chrF++ are word n-grams with $n = 1, 2$ and for the character n-grams $n = 1, \ldots, 6$ is used, lastly β is set to 2[7].

6: $ngrF\beta$ is defined to be zero if both *ngrP* and *ngrR* are zero

7: If we don't use any word n-grams we get the original chrF

2.1.5 Dataset

In this research we use the Tatoeba dataset [28]. This relatively small dataset contains a little more than 300.000 short German source sentences with English translations. The translations are gathered from the website `https://tatoeba.org/`.

Table 2.1: Example data

Source	Target
Tom hat fein säuberlich gekämmtes Haar.	Tom's hair is neatly combed.
Ich verstehe nicht, was du sagst.	I don't understand what you're saying.
Der Astronaut ist im Weltraum über der Erde.	The astronaut is in orbit around the Earth.
Der Baum dort ist nicht so hoch wie dieser hier.	That tree is not so tall as this.
Ich weiß, warum Tom in Schwierigkeiten steckt.	I know why Tom is in trouble.

The dataset was chosen because of the size and familiarity of the researcher with both languages. There are many other datasets used for machine translation and in the future, it would be interesting to see how the choice of the dataset as well as the choice of the language pair influences the predictive models.

2.1.6 Evaluation

To evaluate the predictive capabilities of the trained models we use the well-known Mean Squared Error (MSE). Additionally, we also use Kendall's τ coefficient [29] to measure the ranking capabilities and BLEURT to assess the quality of the translations.

Kendall's τ

Kendall's τ coefficient measures the correspondence between two rankings in which 1 signifies total agreement while -1 signifies total disagreement. There exist different formulations of Kendall's τ. The formulation that is used is:

$$\tau_b = \frac{n_c - n_d}{\sqrt{(n_c + n_d + n_{t1}) \cdot (n_c + n_d + n_{t2})}} \tag{2.13}$$

In which n_c is the number of concordant pairs, n_d is the number of discordant pairs, n_{t1} is the number of ties in the first ranking while n_{t2} is the number of ties in the second ranking. When there are no ties τ_b reduces to τ_a:

$$\tau_a = \frac{n_c - n_d}{n(n-1)/2} \tag{2.14}$$

With n being the number of elements ranked.

BLEURT

BLEURT [12] is a learned evaluation metric based on BERT [30]. The key approach taken with BLEURT is that BERT is pre-trained on syntactic data and then finetuned on the WMT Metrics Shared Task to-English language pairs in which the models are tasked to predict the quality of translations. BERT itself makes use of a [CLS] token. This token is used to attend to the whole sequence. BLEURT adds a linear layer on top of the [CLS] token to predict the quality rating. BLEURT yields competitive results for the WMT metrics shared task for the years 2017, 2018, and 2019.

2.2 Pathologies in NMT Decoding

Müller and Sennrich [6] give a good overview of known deficiencies of NMT models and beam search decoding. Beam search decoding has a *length bias*: beam search produces, on average, shorter translations than the target translations [1, 4, 31]. Furthermore, there is also a problem with *skewed word frequencies*. Beam search produces translations in which common tokens are overrepresented while rare tokens are underrepresented [5]. Translations can also suffer from the *beam search curse*. Increasing the beam size during beam search decreases the quality of the translations [4, 5]. This is likely due to the *inadequacy of the mode*, in many cases, the mode seems to be the empty sequence [1]. Furthermore, the mode doesn't have to be representative of many other sequences as the model distribution is distributed over an extensive region of the output space [2]. NMT models are prone to copy noise such as from misaligned sentences, misordered words and untranslated sentences [5, 32]. These problems mainly occur when less structured datasets, such as web pages, are used. Lastly, NMT models can completely break down when there is a domain shift, the models can start to hallucinate unrelated words. Therefore NMT models seem to have *low domain robustness* [4, 33, 34].

2.3 Relevant literature

As MBR decoding is getting renewed attention we discuss some of the recent work that has been done in this field. Freitag et al. [35] apply MBR decoding with the use of a number of different neural utility functions. They show that using neural utility functions significantly improves the translation quality as measured with human evaluation when compared to beam search. Interestingly, the generated translations have a much lower model likelihood and are less favored by surface metrics like Blue.

To get a broader picture of how MBR decoding relates to other decoding techniques Fernandes et al. [36] compare various candidate generation and ranking methods across different datasets and models. In particular, they use MBR decoding and fixed $N-$best reranking and tuned N-best reranking. With both N-best reranking methods various reference-free metrics are used to rank the best models. They find that their quality-aware decoding approaches consistently outperform MAP-based decoding, according to both state-of-the-art metrics (COMET and BLEURT) and human assessments. In particular, they show that tuned N-best reranking has a small edge over MBR decoding.

Lastly, MBR decoding can also be used in a variety of Natural-language generation tasks. Suzgun, Melas-Kyriazi, and Jurafsky [37] demonstrate that MBR decoding improves results across a variety of tasks, including summarization, data-to-text, translation, and textual style transfer, achieving new state-of-the-art results on two datasets, WebNLG and WMT16.1.

2.4 Conclusion

In this chapter, we introduced the relevant concepts for this thesis. We also discussed pathologies in NMT systems when beam search is used. Next up we will discuss which models we trained as well as how we trained those models.

Predictive Models 3

In this chapter, we introduce the models that we trained. We discuss the design decisions, how we trained the models and give an overview of the validation results. An in-depth analysis will be done in the next chapter.

3.1 Introduction

As a first approach to predicting the Bayes risk, we explore different neural models that predict the Bayes risk directly. The focus of this chapter is on discussing the design decisions and process of constructing these neural models. During the design process, we do need to think about which data to use, how to train our models, and how to perform a hyperparameter search, but most importantly, we need to think about which features and architectures to use.

We distinguish between two "styles" of models. First, we have *quality estimation style models*, which only have access to features related to the source and the hypothesis we try to assess. In contrast, we have *reference style models*, which additionally have access to features of sampled reference translations. The main features that we look at come from the NMT model, such as the hidden states or statistical information about the output of the NMT model. The idea is that this gives information about the certainty of the model as well as information about the other translations the model was considering. We can also give features that are related to the utility function. For this, we use features used in COMET [1].

1: Another advantage about using COMET features is that these are indicative of quality in general and thus could also help with the other utility functions

Designing the models takes an iterative approach, we learn what does and doesn't work from previous designs and try to improve upon those. For the analysis of the models, we use the mean squared error (MSE) on the validation set. More in-depth analysis will be done in the next chapter.

3.1.1 Contributions

In this chapter, we give insight into the design decisions that need to be taken when designing the predictive models. We take an iterative approach to train models which are used in the rest of this thesis. The first validation results give insight into which features are important. In general, it seems that token statistic information carries a lot of information about the Bayes risk. When COMET is used as a utility function, it seems essential that COMET related features are used. A more in-depth analysis will be done in the next chapter.

3.1.2 Outline

We start by discussing how the predictive model can be used to do more efficient Bayes risk decoding. Next, we discuss our approach which includes the design decisions. We then discuss how we find hyperparameters as well as how the models are trained. We finish this chapter with a discussion of the validation results.

3.2 More Efficient Bayes Risk Decoding

Our goal is to find a neural model that is both faster than the MC estimate for estimating the Bayes risk and good enough to be used in practice. With the most basic form of MC-estimation, we have a running time of $O(N^2 \cdot U + N \cdot G)$ in which N is the number of hypotheses, U is the cost of our utility function and G is the cost of generating the samples. After training a predictive model we use it as rule. The model will be used as follows: we generate several hypotheses and let our model score each hypothesis. Then we simply pick the highest scoring hypothesis. This way we have a running time of $O(N \cdot C + N \cdot G)$, in which C is the cost of our model. As long as the computational cost C is low enough we could get more efficient MBR decoding.

3.3 Approach

To get a good predictive model we have several design decisions to make. We discuss the different design decisions as well as which decisions we concretely took. First, we need to think about what data we are going to use. This is not only about the dataset that we use, but also about how many hypotheses we use, how we generate those hypotheses, and what values the model should predict. Secondly, we need to think about the architecture of the model as well as which features the model has access to.

3.3.1 Data

To get the data for our predictive model we first need an NMT model and a dataset from which we get our source and translations. For the NMT model, we train an attention-based German-English Opus model and use the same hyperparameters as in Tiedemann and Thottingal [38]. For our dataset we chose the German English Tatoeba dataset [28]. This dataset is chosen because we understand both German and English reasonably well and the dataset size was not too large. The size of the dataset is important to keep the computational costs manageable.

The dataset itself has no predefined splits. We split the dataset into 5 parts:

1. For **NMT training** (90 % of the data)
2. For **predictive model training** (~ 7.5%) [2]
3. For **evaluating** the NMT model (2500 samples)

2: This may seem small, but for every datapoint we generate 100 hypotheses, increasing the size of the dataset substantially

4. For **evaluating** the predictive model (2500 samples)
5. For **testing** (2500 samples)

These splits give us enough data to train the NMT model on. Furthermore, because we can generate a lot of hypotheses for each source sentence we also get a lot of training data for the predictive model. [3] Other splits and setups are of course possible.

3: So much so that, when not careful, training times can become a practical limitation.

After we trained the NMT model we need to generate the hypotheses and get an approximation of the Bayes risk. We have a unique situation as we have access to the data-generating process. In theory, we could produce as many hypotheses as we would like. To generate the hypotheses we could use ancestral sampling, nucleus sampling [39], top-k sampling [40], beam-search or any other sampling technique.

As we intended to train a robust model for all situations we choose to use ancestral-sampling [41], which is simply generating the tokens one by one by randomly selecting the next token based on the probability given by the NMT model. When we know beforehand the generation strategy for the hypotheses we could obtain better results by training our model with hypotheses given by that generation strategy.

To get a good amount of hypotheses and a robust MC estimate we chose to train the predictive models with 100 hypotheses and calculate the MC estimate using 1000 references for each source.

Other training objectives one could try are predicting the distribution of the utilities for a fixed hypothesis or learning to rank. During the early stages of research, we found that predicting the distribution did not perform better than regressing to the mean and thus we used the latter. [4] Learning to rank was outside the scope of this research.

4: In chapter 5 we revisit this idea to see and explore the models that are used to predict the distribution in more depth.

3.3.2 Features and architectures

Another important choice to make is which features our neural models have access to. When looking at the Bayes Risk we see that it is determined by the following:

1. The distribution of the translations under the NMT model;
2. The utility function used.

We explore the following features to predict the Bayes Risk:

1. Probability of the hypothesis.
2. Entropy of the tokens at each time step.
3. The probability of the top n most likely tokens at each time step.
4. The hidden states of the NMT model.
5. Ancestral sampled references.

For the references, we could also include relevant features, such as the probability and entropy information. Furthermore, when we have access to references we could also provide utility scores or features used in the utility function, such as the embeddings used in COMET. It is important to note that some features are more predictive for the Bayes risk than others, but are also more expensive to compute. These are all practical considerations when training and using the predictive models.

When we know which features we want to use we can design the neural models. As the features are often of variable lengths, such as the hidden states of the NMT model, we have to use deep-learning models that are able to process sequence data, for this, we use LSTM-based models. Whenever we have a sequence that doesn't have an order, such as an embedding for each reference, we use attention-based models as they can process the sequence without being influenced by the order of the sequence.

3.3.3 Loss Function

To train the predictive models we use the well-known Mean Squared Error (MSE) loss function:

$$MSE(a, \hat{a}) = \frac{1}{n} \sum_{i=1}^{n} (a^{(i)} - \hat{a}^{(i)})^2 \tag{3.1}$$

in which $a^{(i)}$ for $1 \geq i \geq n$ are values we want to estimate and $\hat{a}^{(i)}$ are estimates of those values.

3.4 Training

In this section, we describe the training procedure. We first describe the data that we used as well as the NMT model. As the computational costs of training a predictive model tend to be very high, because of the amount of data points we generate, we also describe a general strategy for our hyperparameter search.

3.4.1 Training Data for the Predictive Model

To get training data for the predictive model we first need to train an NMT model. To train the NMT model we used the same training setup as the original opus model, but then with label smoothing disabled[38]. The original model is trained on the large repository OPUS [28] and test scores are available for the Tatoeba dataset. The reported chrF score is 0.707 [42]. After training, our model gets a chrF score of 0.717 on our custom validation set, indicating that our model is properly trained.

We use this NMT model to generate samples. We generate sets of 10, 100, and 1000 samples used for the training, evaluating and testing of the predictive models. We then generate two sets of data for both the training and evaluation subset. A small dataset containing 10 hypotheses with a 100-MC estimate of the Bayes risk for each source and a larger one for 100 hypotheses with a 1000-MC estimate for each source. The smaller one will be used for getting good hyperparameters while the larger one is used for full training. Lastly, we generate 100 hypotheses with 1000-MC estimate of the Bayes risk for the test data. We create these datasets for each utility function used.

3.4.2 Hyperparameter Search and Training

To find decent hyperparameters we use a hyperparameter search. We describe the general setup of the hyperparameter search and training. For more details see appendix A.2.

Our first concern with the hyperparameter search is to keep the computational cost low. As previously mentioned, we use the 10 hypotheses 100-MC estimate data instead of the full 100 hypotheses and 1000-MC estimate data. Furthermore, we apply a median pruner to prune non-promising trials and we use a Tree-structured Parzen Estimator (TPE) based sampler for sampling promising hyperparameters [43]. We do a hyperparameter search once for each model on the COMET dataset and use the found hyperparameters for each utility function. For the reference style model (defined below) we will do the hyperparameter search for $m = 5$. The resulting hyperparameters can be found in the configuration files of the repository that contains the code for this thesis. After finding good hyperparameters we do the full training on the 100 hypotheses, 1000-MC estimate dataset. We use early stopping to make sure we do not overfit.

3.5 Models

Our models can be divided into two types: *quality estimation style* and *reference style*. In the first approach, we do not give any access to reference information, we only give the predictive model access to the source, hypothesis and relevant features. With the reference style models, we give access to reference information. The models are designed iteratively based on the validation results of the previous models.

First a small note on notation. θ_{part} is used to denote the parameters of the model for that specific part (e.g θ_{Dec} denotes the parameters for the decoder). e is used to denote embeddings. For completeness we use the following notations:

θ	parameters of the model
e	refers to embedding
x, h, r	refers to source, hypothesis and reference
$x_{\text{stat}}, r_{\text{stat}}$	refers to the token statistics of a hypothesis and reference respectively
$Dec^{(i)}$	refers to the ith layer of the decoder of the NMT model
FF	is a feed forward network
$BiLSTM$	is a bidirectional LSTM network

3.5.1 Quality Estimation Style Models

For the quality estimation style models, we implemented several models. First, we implemented a baseline model which is an LSTM-based model that has as an input the source and hypothesis concatenated. This gives a (naive) baseline and is used as a sanity check. Next, we used 4 different features:

1. Last hidden states of the NMT decoder of each time step
2. All hidden states of the NMT decoder of each time step
3. Token statistics of each time step
4. Embedding of the source and hypothesis used in COMET

For the token statistics, we use the probability of the selected token as well as the entropy of the token distribution of each time step [5]. Furthermore, we give access to the probabilities of the top 5 most likely tokens at each time step, which gives some additional information about the distribution of the tokens.

5: We will refer to "the entropy of the token distribution of each time step" simply as the *entropy of the tokens*, from here onward.

Next, we give an overview of the models used. Important to note is that for each utility function, we use a different activation function after the last layer. For both the unigram-F1 and chrF++ we use the sigmoid activation as the Bayes risk is bounded between 0 and 1 for those utility functions. COMET scores are unbounded in theory but in practice, they lay between -2.5 and 2.5, therefore we use the tanh activation function multiplied by 2.5 when COMET is used as a utility function. These activation functions are denoted by f_{act}.

Baseline Model. The Baseline model consists of two embedding layers, one for the source and one for the hypothesis. The embeddings are concatenated and fed into a bi-directional LSTM. The final hidden states are then used as features for a feed-forward layer. The output is put through the appropriate activation function.

$$
\begin{aligned}
e_x &= Emb_x(x, \theta_{Emb_x}) \\
e_h &= Emb_h(h, \theta_{Emb_h}) \\
\text{feat} &= BiLSTM([e_x, e_h], \theta_{lstm}) \\
\tilde{\mu} &= f_{\text{act}}(FF(\text{feat}, \theta_{FF}))
\end{aligned}
$$

Last Hidden State Model. The first attempt to get a good model is to use the last hidden state of the decoder. The idea is that the last hidden state contains information about the distribution of the tokens, which in turn could give information about the distribution of translations under the NMT model[6].

6: The decoder model contains an embedding layer and 6 attention layers, which we numbered $0, \dots, 6$

$$
\begin{aligned}
e_h &= \text{Dec}(x, \theta_{Dec})^{(6)} \\
\text{feat}_h &= BiLSTM(e_h, \theta_{lstm}) \\
\tilde{\mu} &= f_{\text{act}}(FF(\text{feat}_h, \theta_{lstm}))
\end{aligned}
$$

Token Statistics Model. To improve on the Last Hidden State Model we use the token statistics. Using the token statistics directly could give more information about the Bayes risk than the last hidden states as it directly gives access to the statistics about the distribution of the tokens (such as the entropy of the tokens). The token statistics model uses the token statistics h_{stat}. It first maps them to a higher dimensional space and then feeds it through a BiLSTM model:

$$
\begin{aligned}
e_{h_{\text{stat}}} &= FF_1(h_{\text{stat}}, \theta_{FF_1}) \\
\text{feat} &= BiLSTM(e_{h_{\text{stat}}}, \theta_{lstm}) \\
\tilde{\mu} &= f_{\text{act}}(FF_2(h_{\text{stat}}, \theta_{FF_2}))
\end{aligned}
$$

Full Decoder Model. To use the full information that is hidden in the decoder model we can also use all the hidden states of the decoder model. The hope is that the earlier hidden states contain some sort of information about alternative translations the model was considering. The full decoder model uses the token statistics h_{stat} as well as all the hidden states of the decoder part of the NMT model:

$$
\begin{aligned}
e_{\text{stat}} &= FF_1(h_{\text{stat}}, \theta_{FF_1}) \\
e_{h_i} &= \text{Dec}(x, \theta_{Dec})^{(i)} && \text{for } i \in \{0, \ldots, 6\} \\
\text{feat}_i &= BiLSTM_i(e_{h_i}, \theta_{lstm_i}) && \text{for } i \in \{0, \ldots, 6\} \\
\text{feat}_{\text{stat}} &= BiLSTM(e_{stat}, \theta_{lstm_{stat}}) \\
\tilde{\mu} &= f_{\text{act}}(FF_2([h_{stat}, e_{h_0}, \ldots, h_{e_{h_6}}], \theta_{FF_2}))
\end{aligned}
$$

Full Decoder Model - no token statistics. As you can see in table 3.1 the full decoder model has a lower MSE than the token statistics model. To understand the influence of the token statistics we train a similar model as the full decoder model but remove the token statistics from the input:

$$
\begin{aligned}
h_i &= \text{Dec}(x, \theta_{NMT})^{(i)} && \text{for } i \in \{0, \ldots, 6\} \\
feat_i &= BiLSTM_i(e_{h_i}, \theta_{lstm_i}) && \text{for } i \in \{0, \ldots, 6\} \\
\tilde{\mu} &= f_{\text{act}}(FF_2([feat_{h_0}, \ldots, feat_{h_6}], \theta_{FF_2}))
\end{aligned}
$$

COMET Feature Model. Lastly, we want to explore what happens if we use features that are important for the utility function. For this, we look at COMET. The COMET feature model uses the same embeddings for the source ($e_{\text{s-comet}}$) and the hypothesis ($e_{\text{h-comet}}$) as used in the COMET model. We take the element-wise dot product and calculate the absolute difference between the two embeddings similar to the COMET paper [11].

$$
\begin{aligned}
e_{\text{comet}} &= [e_{\text{h-comet}}, e_{\text{s-comet}}, e_{\text{h-comet}} \odot e_{\text{s-comet}}, |e_{\text{h-comet}} - e_{\text{s-comet}}|] \\
\tilde{\mu} &= f_{\text{act}}(FF(e_{\text{comet}}))
\end{aligned}
$$

Full Decoder COMET Feature Model. A drawback of the COMET feature model is that it doesn't have access to features that are indicative of the distribution of the translations. Therefore we combine the full decoder model with the COMET feature model to get the last quality estimation model:

$$\tilde{\mu} = f_{\text{act}}(FF([e_{\text{comet}}, e_{h_{\text{stat}}}, e_{h_0}, \ldots, e_{h_6}]))$$

The validation results of the quality estimation models are in table 3.1. In this table, we can observe that the Full Decoder model has one of the lowest MSE for both unigram-F1 and chrF++. Adding COMET features is of great benefit when using COMET as a utility function. For unigram-F1 it seems to degrade performance while for chrF++ it improves it by a little. From the NMT features, the token statistics seem to be the most important, as when we remove them, the MSE suffers quite a lot for all three utility functions.

Table 3.1: Models validation MSE for unigram-F1, chrF++ and COMET. The bold entries are the best scores for that particular utility function.

Model	Unigram-F1	chrF++	COMET
Basic Model	1.2e-02	2.0e-02	2.8e-01
Last Hidden State Model	6.9e-03	1.2e-02	1.6e-01
Token Statistics Model	3.5e-03	6.1e-03	1.0e-01
Full Dec Model	**2.5e-03**	3.8e-03	8.6e-02
Full Dec Model (no stats)	5.9e-03	1.0e-02	1.3e-01
Comet feature model	7.8e-03	1.2e-02	4.8e-02
Decoder Comet feature model	2.7e-03	**3.7e-03**	**3.0e-02**

3.5.2 Reference style models

For reference style models we try out two different models. First, we have a basic model that we give access to the m-MC estimate and try to approve upon it. Next, we give a model access to m references as well as the m-MC estimate. The hope is that these m references give extra information which the model can use to get an accurate prediction.

Basic reference model For the basic reference model, we build upon the full decoder model and give it access to the m-MC estimate. The final prediction will be of the form:

$$\tilde{\mu} = \tilde{\mu}_\delta + \tilde{\mu}_m \tag{3.2}$$

Where $\tilde{\mu}$ is our final prediction, $\tilde{\mu}_\delta$ is the model's prediction and $\tilde{\mu}_m$ is the m-MC estimate for a given hypothesis. This way we still have the model learn to predict the Bayes risk, but we provide stability by giving it access to the m-MC estimate. We concatenate the m-MC estimate to the features we get from the decoder. This is then put through a feed-forward network with tanh as a final activation function to get a correction value $\tilde{\mu}_\delta$ between -1 and 1. Then equation 3.2 is used to get a final prediction. The model thus learns to 'correct' the given m-MC estimate.[7] For unigram-F1 and chrF++ we bound the final prediction between 0 and 1.

7: Early experiments showed that if we predicted the value directly, the predictive model would perform badly when m is large.

Embedded Reference Model The final model that we train is a model that gets access to m references as well as the m-MC estimate. We embed these references as well as the hypotheses with the help of the embedding layer of the decoder model. The hope is that these embeddings give additional information that the m-MC estimate doesn't contain.

$$
\begin{aligned}
e_{r_i} &= \text{Mean}(\text{Emb}_{\text{Dec}}(r_i)) && \text{for } 1 \leq i \leq m \\
e_h &= \text{Mean}(\text{Emb}_{\text{Dec}}(h)) \\
e_r &= [e_{r_1}, \ldots, e_{r_m}] \\
e &= ATT(e_h, e_r, e_r) \\
\tilde{\mu}_\delta &= tanh(FF([e, \tilde{\mu}_m])) \\
\tilde{\mu} &= \tilde{\mu}_\delta + \tilde{\mu}_m
\end{aligned}
$$

This model thus uses the references to correct the m-MC estimate. Other models that incorporate reference information are possible. But we leave those for future research. During training, we ran into memory issues whenever there were too many references, therefore for practical reasons we decided to limit the number of references by having $1 \leq m \leq 5$.

Table 3.2: Basic reference model validation MSE for different values of m. Increasing the number of references always results in better Bayes Risk estimates.

m	unigram-F1	chrF++	COMET
1	2.0e-03	3.1e-03	4.1e-02
2	1.6e-03	2.4e-03	2.8e-02
3	1.4e-03	2.0e-03	2.2e-02
4	1.2e-03	1.8e-03	1.7e-02
5	1.0e-03	1.6e-03	1.5e-02
10	7.0e-04	1.0e-03	7.9e-03
25	3.0e-04	5.0e-04	3.3e-03
50	2.0e-04	2.0e-04	1.7e-03
100	9.5e-05	1.0e-04	9.0e-04

Table 3.3: Embedded reference model validation MSE for different values of m. Increasing the number of references always results in better Bayes Risk estimates. To ensure we don't have memory issues we keep the number of references small.

m	Unigram-F1	chrF++	COMET
1	6.2e-03	8.8e-03	7.2e-02
2	3.7e-03	5.2e-03	3.9e-02
3	2.7e-03	3.7e-03	2.7e-02
4	2.1e-03	2.9e-03	2.1e-02
5	1.7e-03	2.4e-03	1.7e-02

In table 3.2 we see the results of the Basic reference model for the different numbers of references. In table 3.3 we can see the results for the Embedded Reference model. We observe that increasing the number of references m always decreases the MSE. The decrease slows down as the number of references increases. The Basic Reference model outperforms the Embedded Reference model by quite a large margin, e.g. for unigram-F1 and chrF++ the embedded reference model with $m = 4$ is comparable with the basic reference model with $m = 1$. Lastly, the reference style models seem to outperform the quality estimation models, which indicates that adding reference information increases the quality of the models.

3.5.3 Summary of Validation Results

From the validation results, we see that the best models, in terms of validation MSE, are the one that incorporates features from all the hidden states of the decoder model and possibly the COMET features. The token statistics are one of the most important features for unigram-F1 and chrF++, while COMET features seem to be crucial for the COMET utility function. Lastly, adding reference information improves the performance of the model, but has decreasing returns as the number of references used increases.

3.6 Conclusion

In this chapter, we gave an overview of the design decision, hyperparameter search, and training of the models. We iteratively build predictive models and found that using all the hidden states of the decoder model (with possibly COMET features) gives the best model in terms of validation MSE. Furthermore, the token statistics seem to be one of the most important features for unigram-F1 and chrF++ while for COMET the COMET features seem crucial. However, the validation loss is not the whole story. In the next chapter, we analyze the models in more depth to understand their capabilities.

Analysis 4

In this chapter, we analyze the models that we trained. We analyze the models both in terms of *quality* and *computational cost*. This analysis is then used to answer the research question Q1 and its subquestions.

4.1 Introduction

In the previous chapter, we trained a number of models and got some first results based on the validation dataset. In this chapter, we will analyze these models in more depth.

First, we look at the quality of the model. The primary task of the models is to predict the Bayes risk, therefore we asses how good the model is at predicting the Bayes risk by computing the mean squared error and Kendall τ statistics. As we want to use the model as a decision rule to find the MBR translation, we also analyze the quality of the translations that are picked based on the Bayes risk given by the models. Quality however is not the whole story. To see if the model actually results in a speed up we look at the computation complexity and the practical computation time. To combine quality and computational speed we finish the chapter by comparing the speed and quality at the same time, giving us a good view of which and when our models outperform MC estimates. After this analysis, we are able to answer research question Q1 and its subquestions.

4.1.1 Contributions and outline

In this chapter, we show that it is possible to predict the Bayes risk using neural models, but they only outperform the m-MC estimates for low values of m when looking at the mean squared error and the ranking capabilities. The best-performing models however outperform m-MC estimates when looking at the quality [1] of the translations they pick when used as a decision rule for every value of m.

1: Measured with the utility functions used and BLEURT

In the first section, we test the quality of the predictive models by computing the mean squared error of its predictions and computing the rank correlations. We use the model as a decision rule and measure the quality of the selected translations. We compare the quality of those translations with the ones we pick when using m-MC estimation. Lastly to evaluate the computational cost we compute the mean speed of the estimations for an individual hypothesis and compare that with the computational cost of the m-MC estimate. We plot the computation time and several metrics in to get an idea about the trade-off between speed and performance. We end this chapter by answering research question Q1.

4.2 Quality

To test the quality of the models we look at three metrics. First, we look at the mean squared error (MSE) to get an indication of how well the models were able to fit the data. Next, the median of the Kendall τ is computed to see how good a model is at ranking the different hypotheses for a given source. The median is used as the models have outliers as you can observe in figure 4.3. Lastly, to check how well the model works in practice we use it as a decision rule. We compare these scores with the m-MC estimate. The comparison with the m-MC estimate is made because that is the approach we try to improve upon.

4.2.1 Mean Squared Error

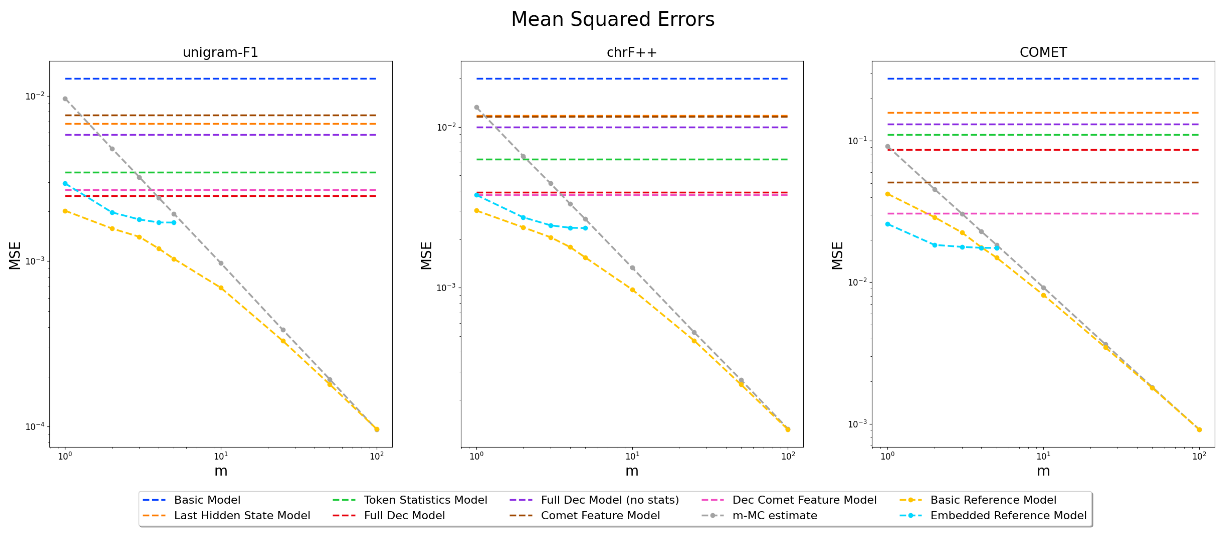


Figure 4.1: The MSE of the different models for the three utilities (lower is better). The straight lines represent the quality estimation models. For the other models as well as the m-MC estimate the dots represent the MSE for that value of m and the lines in between are an interpolation.

In figure 4.1 the MSE of the different models is shown as well as the m-MC estimate. When we look at the quality estimation models we see that the best models are the models that use the full decoder information. For unigram-F1 and chrF++ adding COMET features doesn't seem to improve the model that much. Interestingly for COMET the COMET features make a really big difference. Most of the quality estimation models that don't use COMET features aren't even able to beat the 1-MC estimate. Therefore it seems that the 1000-MC estimate of the unigram-F1 and chrF++ are easier to predict than those of COMET. When looking at unigram-F1 and chrF++ we see that the models are able to beat about the same m-MC estimate. So for example when we look at the Full Dec Model both for unigram-F1 and chrF++ this model has an MSE which is between the 4 and 5-MC estimate.

Another interesting thing to note is that the token statistic model outperforms the Last Hidden State Model and the model that uses the hidden state of the decoder but *not* the token statistics. This shows the token statistics alone already capture a lot of relevant information about the

Bayes risk and the hidden states of the NMT model don't add that much extra information.

For the reference style models, we see an interesting pattern: increasing the number of samples the model has access to decreases the MSE. The decrease however doesn't keep up with the decrease of the MSE of the m-MC estimate, resulting in a convergence between the MSE of the m-MC estimate and that of the reference style models.

4.2.2 Kendall τ

For the Kendall τ statistics we compute the Kendall τ statistic for every source, comparing the ranking of the hypotheses for a given model with those of the 1000-MC estimate. The median of these statistics is then calculated. The median is used as the Kendall τ statistics are skewed, see figure 4.2. Some sources only had 1 hypothesis, these sources were dropped for this analysis. The results are in figure 4.3.

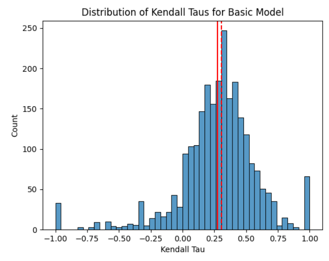


Figure 4.2: Distribution of the Kendall τ statistics of the basic reference model. The red line indicates the mean, while the red dashed line indicates the median.

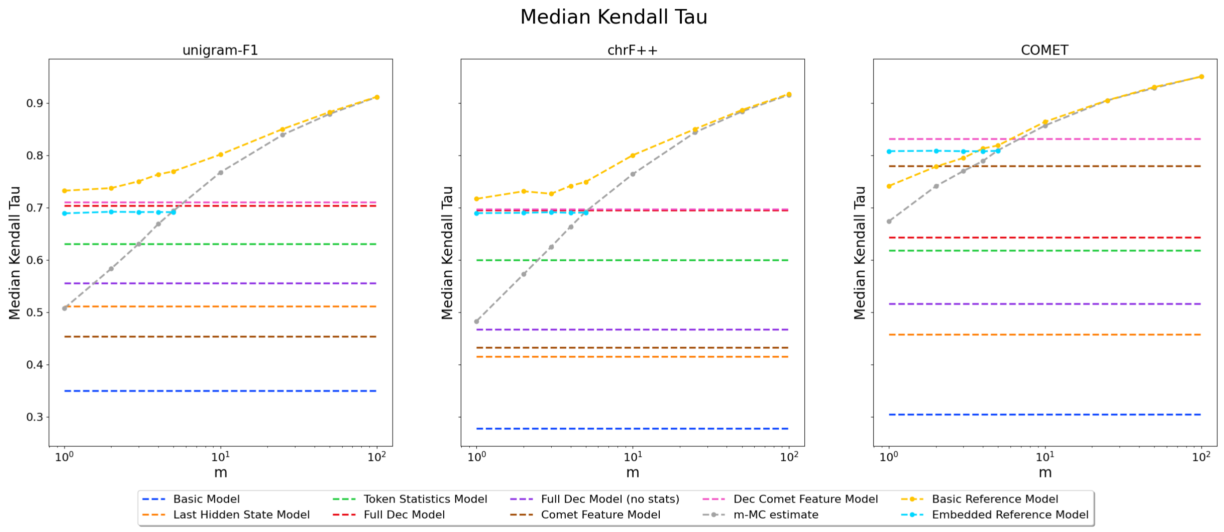


Figure 4.3: The median Kendall τ of the different models for the three utilities. (Higher is better) The straight lines represent the quality estimation models. For the other models as well as the m-MC estimate the dots represent the median Kendall τ measured for that value of m and the lines in between are an interpolation.

For the Kendall τ statistics we see a similar pattern emerging as from the MSE results. We see again that the models using all the hidden states perform the best. For COMET it seems that the COMET features are really indicative for the ranking of the different hypotheses, evidenced by the fact that the two models incorporating the comet features have a high median Kendall τ, namely around 0.8, while the other quality estimation models don't reach 0.65. For unigram-f1 and chrF++ the quality estimation models do not cross a median of 0.72. Again the reference models perform above the m-MC estimate and converge when m is increasing. Lastly, the embedded reference model does not really seem to improve when we increase the number of references the model has access to.

4.2.3 Decision Rule

To finish the analysis of the quality of the model we will look at how well the models perform when the models are used as a decision rule. For each model, we look at two scores, the utility score that was used for training the model and BLEURT. We use BLEURT to get an independent score [2] indicative of the quality of the selected hypotheses.

2: Independent as in: not related to the utility used.

When we look at the utility scores in figure 4.4 we observe that the Full Decoder Model and the Comet Feature Decoder model work best for both the unigram-F1 and chrF++ utilities. For COMET the models that incorporate the COMET features work best. Interestingly the Dec Comet feature model outperforms the 1000-MC estimate. For the models that incorporate reference information, we see that they outperform the MC estimate and converge to the MC estimate as m increases. When we look at the BLEURT scores in figure 4.5 we see a similar pattern emerge, models with a high utility score also have high BLEURT scores. The models trained for COMET have higher BLEURT scores in general, which is probably due to that BLEURT and COMET both have higher correlation with human judgment than unigram-F1 and chrF++.

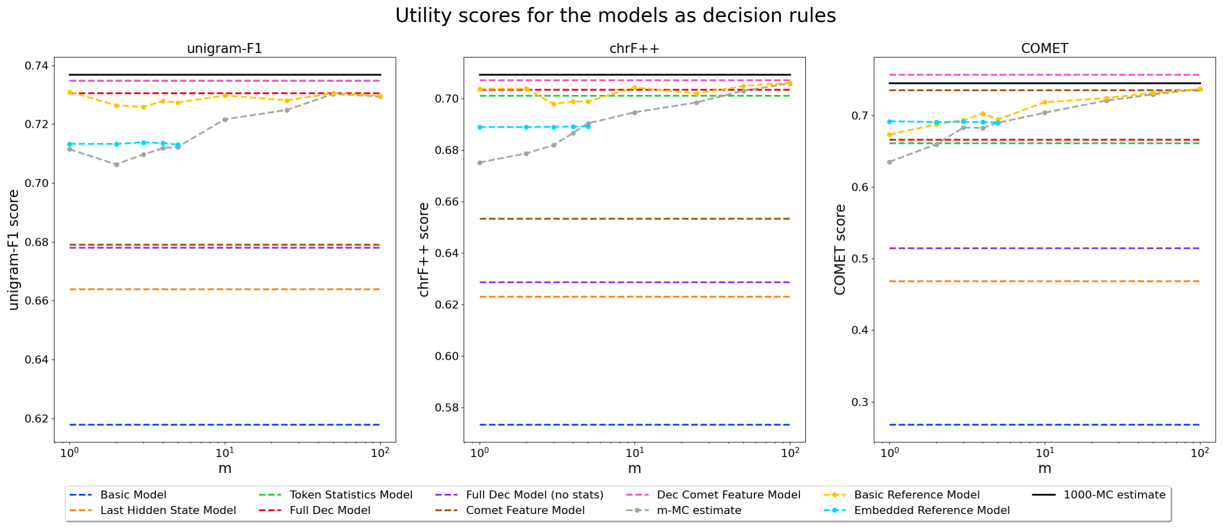


Figure 4.4: The utility scores of the different models when used as a decision rule for the three utilities. (Higher is better). The straight lines represent the quality estimation models. For the other models as well as the m-MC estimate the dots represent the utility score measured for that value of m and the lines in between are an interpolation. Note that for unigram-F1 the Full Dec model and the Token Statistics model get the same unigram-F1 score.

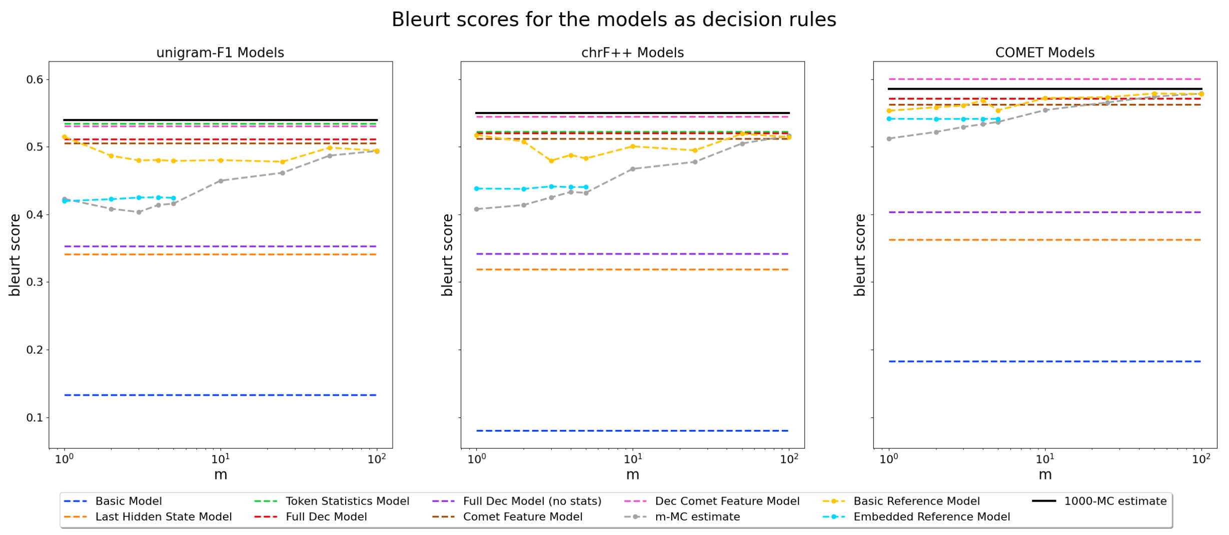


Figure 4.5: The BLEURT scores of the different models when used as a decision rule for the three utilities. (Higher is better). The straight lines represent the quality estimation models. For the other models as well as the m-MC estimate the dots represent the BLEURT score measured for that value of m and the lines in between are an interpolation.

4.3 Computation time

To get the computational speed we check how fast the model works in practice, we again compare this with the m-MC estimates. We graph the computation speed of the different models. First, we give some theoretical notion of how fast the different models are, then we measure the real computation time.

4.3.1 Theoretical Computational Complexity

For the theoretical computational complexity, we look at the cost of getting an approximation of the Bayes risk for one individual hypothesis.

MC estimation The cost of the MC estimate for a single hypothesis consists of getting the references and calculating the utility between a hypothesis and the references. Thus the computation time is $O(m \cdot U + m \cdot G)$ with U being the cost of the utility function and G being the cost of generating the references. In the implementation we keep track of references that occur multiple times and do the utility calculation for those references only once, reducing the cost of the procedures whenever the NMT model is certain of its predictions.

Quality estimation models For the quality estimation models, the only computational cost is putting the features through the model, which gives us a computation time of $O(C + F)$, in which C is simply the cost of the model and F is the cost of getting the features. In the situation that the hypotheses are coming from the NMT model and we use NMT model features, the cost of F can be dropped as we can get the features during the generation of the hypotheses, resulting in a computation time of $O(C)$.

Reference style models For the basic reference model the computational costs consist of putting the features through the model as well as through the utility function, resulting in a computation time of $O(C + F + m \cdot$

$U + m \cdot G$), and when we have the NMT features as a given we get $O(C + m \cdot U + m \cdot G)$. For the embedded references model the computation time is $O(m \cdot C + m \cdot F + m \cdot U + m \cdot G)$ in which the computational cost of the model and that of getting the embeddings scales linearly with m.

4.3.2 Practical Computation Time

For each approach, the computation time is calculated by taking every hypothesis for the first 100 sources of the test set [3] and we calculate the approximate Bayes risk one by one whilst measuring the computation time. We average to get the mean computation time. We also keep track of the time it takes to get the features out of the NMT model. Additionally, for the reference style models and m-MC estimate, we keep track of the time it takes to generate m references. The computation time is, of course, dependent on both the implementation and the hardware it runs on[4] and the size of the models used. The times should primarily be taken as indicative.

3: This results in 3616 hypotheses being used.

4: The graphical card that was used is an NVidia RTX A4000 and as a processor, an Intel-12700KF was used.

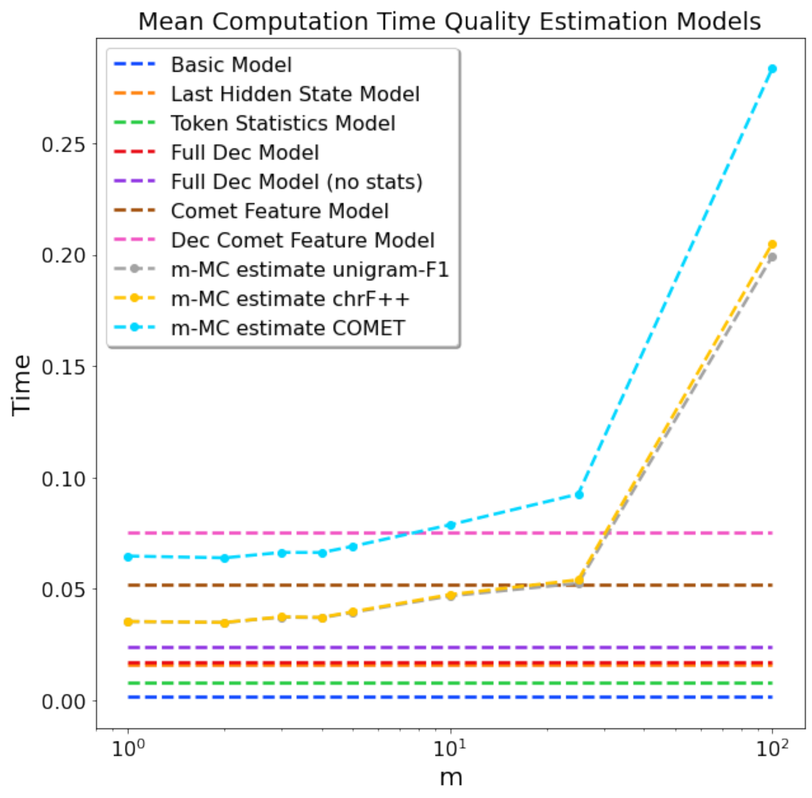


Figure 4.6: The computation time of the quality estimation models (Lower is better). Note that the full decoder model and the last hidden state model almost have the same computation time.

Quality Estimation Style Models

In figure 4.6 we can see that most quality estimation models are faster than the m-MC estimate, with the exception of the models that incorporate the COMET features. For COMET we see that the best performing model, namely the Dec Comet feature model is faster than the 10-MC estimate while for unigram-F1 and chrF++ it is faster than approximately the 30-MC estimate. This difference can be due to efficient parallelism used when doing the m-MC estimation, as the utility scores can be calculated in parallel, while the Dec Comet feature model computes the features from COMET and the decoder model sequential. More efficient implementation could increase the computation speed.

Reference Style Models

For the reference style model, we only plot the Basic reference model, as the Embedded Reference model doesn't incur meaningful extra computation time over the m-MC estimate. In figure 4.7 we see that these models basically incur constant time increases compared to the MC estimate, which is because the hypothesis has to go through the Full Dec model to get the final prediction.

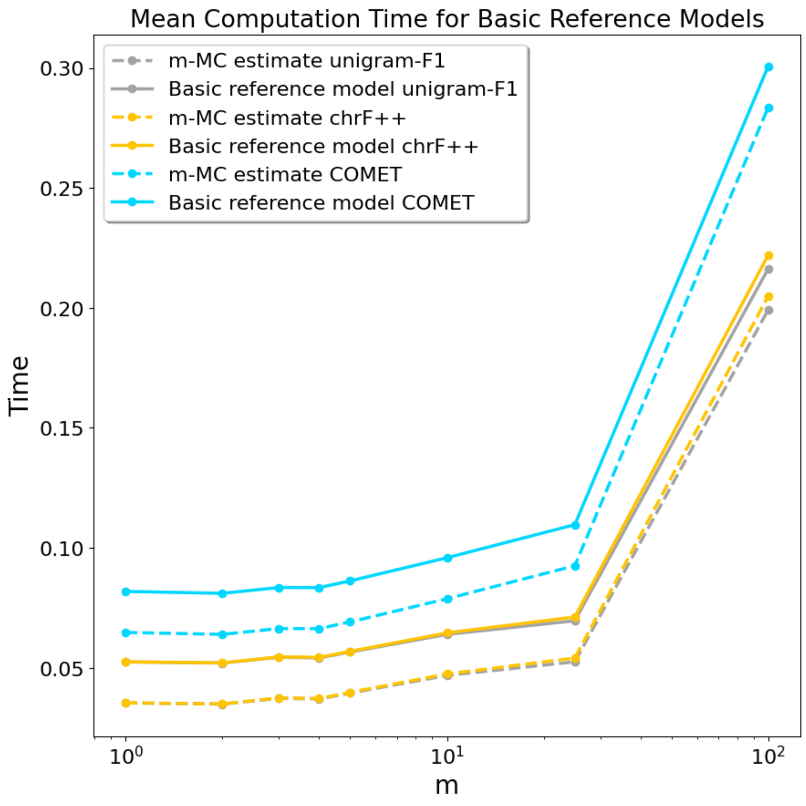


Figure 4.7: The computation time of the basic reference models and the MC estimates. (Lower is better)

4.3.3 Computation Time versus Quality

In this last section, we compare the computation time and the quality of the models. This gives us a clear picture of when the models outperform MC estimation in both computational speed and quality. We graph the computation time and compare it to the MSE, median Kendall τ, utility scores and BLEURT scores.

Results

First, when looking at the MSE (figure 4.8) and the median Kendall τ (figure 4.9) we see the same pattern two times. The models outperform the MC estimates in terms of speed but perform worse when looking at

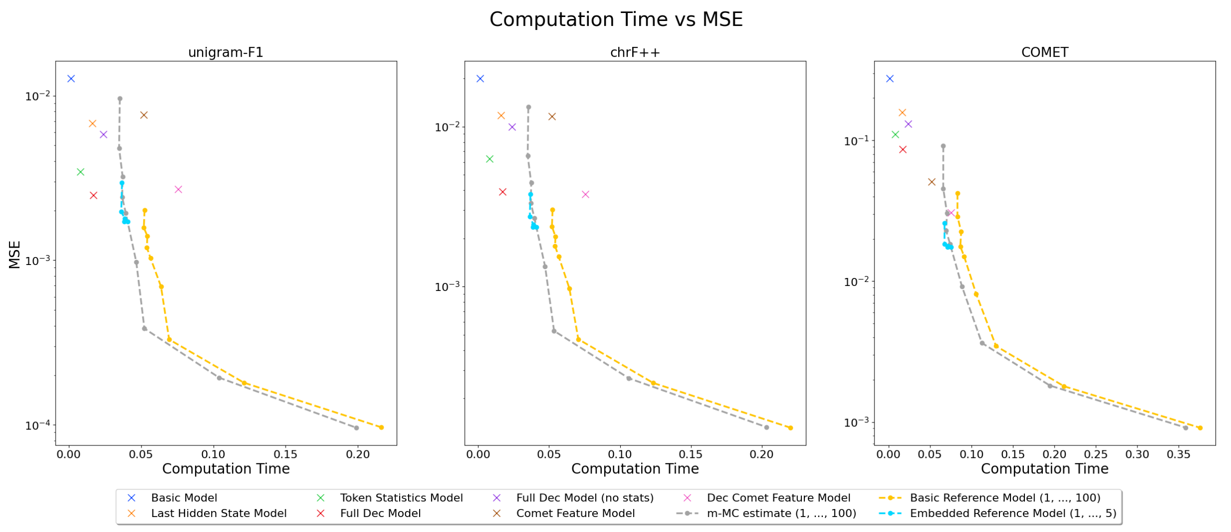


Figure 4.8: The computation time compared to the Mean Squared Error. For both MSE and the computation time lower is better. The quality estimation style models tend to be faster but incur greater errors than the MC estimates.

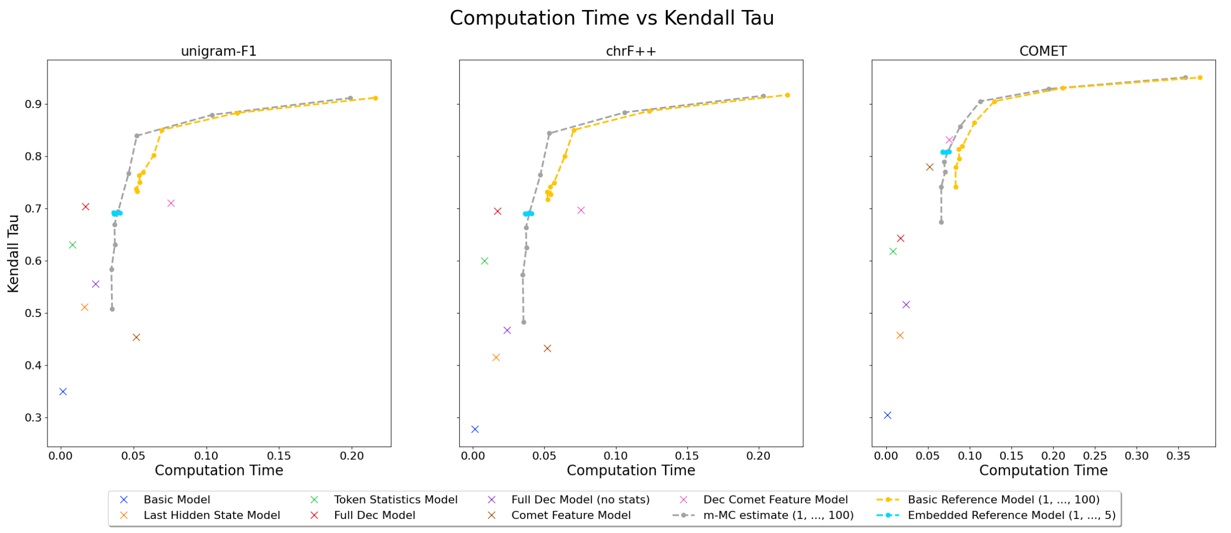


Figure 4.9: The computation time compared to the median Kendall τ scores. The best models should have high Kendall τ scores and low computation times. The quality estimation style models tend to be faster, but MC estimation gets better Kendall Tau scores.

the errors or ranking capabilities. When we look at the utility scores in figure 4.10 and the BLEURT scores in figure 4.11 we get a different picture. The quality estimation models outperform the MC estimates, that is, for every m MC estimate we can find a model that is both faster and achieves better scores[5]. In particular, the Token Statistic model performs really well in all cases. The Dec Comet Feature model is a bit slower but generally achieves better scores. For the reference style models, we see that they tend to perform slightly worse or comparable with the m-MC estimates.

5: In technical terms, we can say that the quality estimation style models form a *Pareto front*[44]

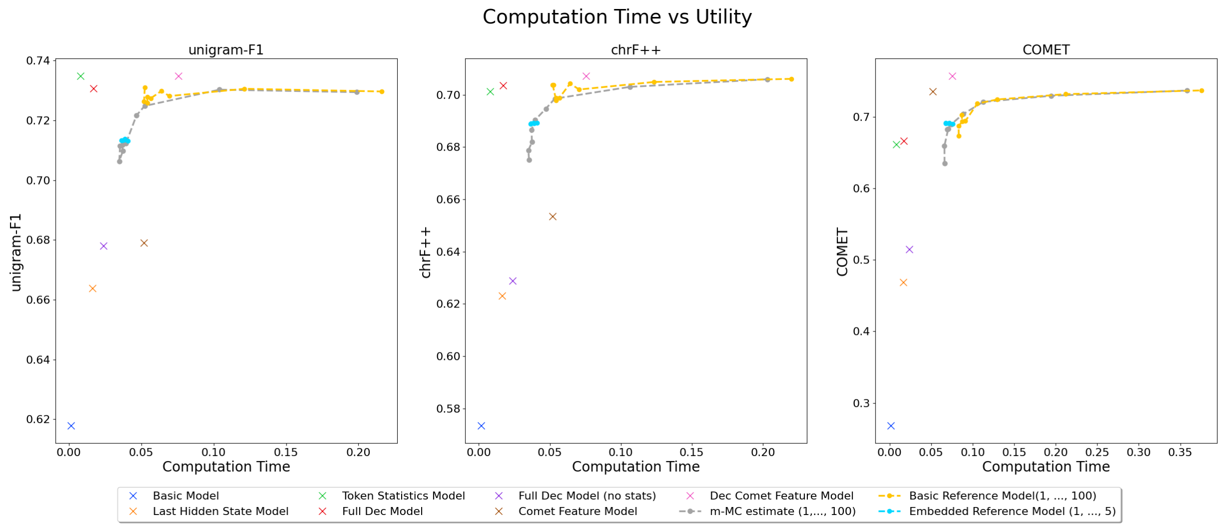


Figure 4.10: The computation time compared to the utility scores. The quality estimation style models outperform the MC estimates. For each m we can find a quality estimation model that takes less computational time and get better utility scores than that m-MC estimate

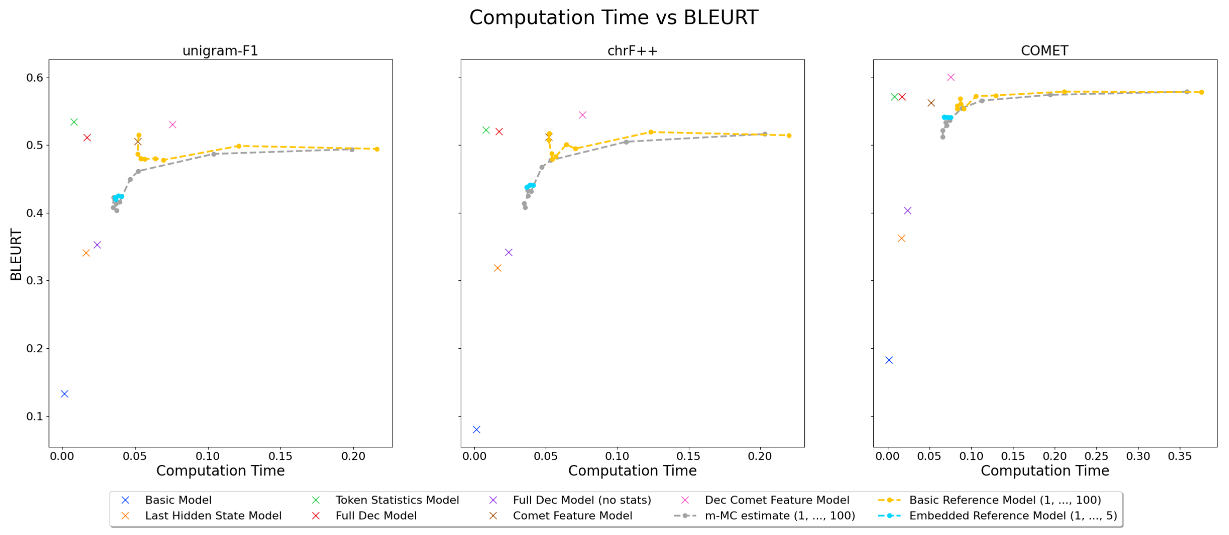


Figure 4.11: The computation time compared to the BLEURT scores. The quality estimation style models outperform the MC estimates. For each m we can find a quality estimation model that takes less computational time and gets better BLEURT scores

4.4 Research questions

First, we answer the subquestions and then we summarize our findings in the main question.

How does the choice of utility function influence the predictive model?

In general, it seems that the Bayes risk is easier to predict when unigram-F1 or chrF++ is used compared to COMET. This is evident by the fact that more models are able to outperform the m-MC estimates for low m when using unigram-F1 or chrF++ compared to when COMET is used. However, the quality of the chosen translations when the model is used as a decision rule is higher when COMET is used as a utility function than when unigram-F1 or chrF++ is used.

Which features are important in approximating the Bayes risk?

In general, token statistics seem to be important. Removing these worsen the performance, this happens for every utility function. Information from the other hidden states also helps, but the token statistics seem to do most of the heavy lifting. For the COMET utility, the COMET features seem to be really important as it improves model performance on all metrics used. For unigram-F1 and chrF++ COMET does not seem to matter that much in predicting the Bayes risk, but does help in picking translations with higher BLEURT scores. The reference style models perform better in terms of MSE and Kendall τ but perform worse when we look at the quality of the chosen translations.

How well do the models perform in practice?

The quality style models perform well in practice. They outperform the m-MC estimates. To be precise, for every m-MC estimate we can find a quality estimation model that performs better (is both faster and gives higher quality translations) than that m-MC estimate. The reference style models however seem to only be slightly better than m-MC estimation and achieve the best scores when the number of references used is low.

Can neural models be used to accurately and efficiently predict the Bayes risk?

Neural models outperform m-MC estimates for $m \leq 5$ in terms of ranking capabilities and MSE. When the models are used as a decision rule they outperform MC estimation in both speed and quality of the chosen translations.

4.5 Conclusion

In this chapter, we performed a more in-depth analysis of the models. We found out that some models can be better than MC estimation when used for Bayes risk decoding. In the following two chapters, we look at the two other approaches for Bayes risk decoding: using *neural mixture models* and by applying *Backpropagation*.

Mixture Models 5

In this chapter, we explore the idea of fitting a mixture model to the distribution of the utilities for a given hypothesis. We compare this approach with regressing to the 1000-MC estimate.

5.1 Introduction

As an alternative for regressing to the MC estimate we explore fitting a neural mixture model to the distribution of the utilities of the hypotheses. We motivate this approach and give an overview of the contributions as well as an outline of this chapter.

5.1.1 Motivation

When calculating the Bayes risk we have access to the full distribution of the utilities. However, sampling from this distribution is very costly. For a given source and hypothesis pair, we first need to sample the references and then for each reference, we need to calculate the utility. Instead of using the true distribution of the utilities we could use a proxy distribution of the utilities. If this approximate distribution is cheap to construct, cheap to sample from and follows the true distribution closely, we might be able to approximate the Bayes risk more efficiently.

The choice of the proxy distribution is important to make sure we can properly fit to the distribution of the hypotheses. In figure 5.1 we can see the distribution of the utilities of 4 source hypothesis pairs. One main observation is important: the distributions are multimodal[1]. Fitting a unimodal distribution therefore will never be able to adequately capture the utility distribution. We, therefore, fit a mixture of distributions as a mixture can be multimodal. A neural network is used to get the parameters of the mixture for a given source hypothesis pair.

1: That is, they have multiple peaks

An interesting note is that at the start of this thesis, we started out with fitting mixture models, but the mixture models performed worse when used to predict the Bayes risk than the models that regressed to the 1000-MC estimate. Therefore we focused on regressing to the 1000-MC estimate. Now we revisit this approach with the Dec Comet model (see chapter 3 and 4) to see how our best-performing model performs when used to predict the parameters of a mixture model. We focus on the quality of the model, asses the fit how well the model is able to predict the Bayes risk and how well the model performs when used as a decision rule.

5.1.2 Contributions

The contributions in this chapter are as follows:

1. We show that mixture models perform on-par with regressing to the mean when used to predict the Bayes-risk.
2. We compare a mixture of Gaussians with a mixture of student-t distributions and show that although the student-t mixture has a better fit, it doesn't result in better predictions.

We conclude by answering the question " Can mixture models be used to predict the Bayes risk? " by saying that it is possible to predict the Bayes risk with this approach but it is not better than the alternative of predicting the 1000-MC estimate directly.

5.1.3 Outline

In this chapter we first discuss the relevant background, then we discuss the approach taken. We analyze the resulting model in terms of the fit and several metrics such as MSE with the 1000-MC estimate, Kendall τ, and quality of the translations when using this model as a decision rule. During this analysis, we compare the mixture model with the original model that predicts the 1000-MC estimate directly. We finish by summarizing the results and discussing future steps.

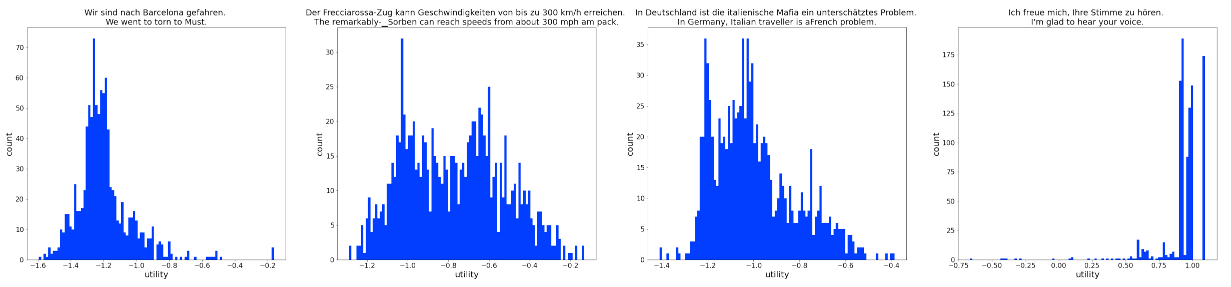


Figure 5.1: The distribution of the COMET score for 4 different source hypothesis pairs. 1000 references are sampled for each source hypothesis pair. The distributions appear multi-modal.

5.2 Background

As most of the relevant concepts are introduced in chapter 2, we only discuss mixture models and the loss function used for training.

5.2.1 Mixture Models

A mixture model "mixes" several distributions into one new distribution in which each distribution g_k for x with parameters Θ_K is weighted by w_k. The distributions are usually taken from one family of distributions [**ghojoghfitting**].

A mixture of distributions is defined as follows:

$$
\begin{aligned}
f(x;\Theta_1,\ldots,\Theta_K) &= \sum_{k=1}^{K} w_k g_k(x;\Theta_k) \\
\text{subject to } \sum_{k=1}^{K} w_k &= 1
\end{aligned}
\tag{5.1}
$$

For our case, we used two families of distributions: Gaussian and Student-t. The well-known Gaussian distribution is defined for some location parameter μ and scale parameter σ^2 as

$$
f(x) = \frac{1}{\sigma\sqrt{2\pi}}\exp\left(\frac{1}{2}\frac{(x-\mu)^2}{\sigma^2}\right) \tag{5.2}
$$

The Student-t distribution is related to the Gaussian, besides the location and scale parameter it has a degree of freedom parameter v. It is defined as:

$$
g(x) = c \cdot \frac{1}{\sigma}\left(1 + \frac{1}{v}\frac{(x-\mu)^2}{\sigma^2}\right)^{\frac{1}{2}(v+1)} \tag{5.3}
$$

With c is a constant:

$$
c = \frac{1}{\sqrt{v}B(\frac{v}{2},\frac{1}{2})} \tag{5.4}
$$

With B being the Beta function. A main difference between the Gaussian and the student-t distribution is that the student-t distribution allows for heavier tails.[2]

2: Important to note is that for $v < 1$ the mean is undefined and for $v < 2$ the variance is infinite. When we sample from a student-t distribution to calculate the mean we need to make sure that $v > 2$, as otherwise, the variance in our samples could be too high to reliably estimate the mean.

5.2.2 Negative Log Likelihood

To measure how well the model is able to capture the distribution of the utility we use the log-likelihood loss. For a distribution with likelihood function $p(y|\Theta)$ that is parameterized by Θ the loss for a given y is defined as follows:

$$
\text{loss}(\Theta|y) = -\sum_{i=0}^{m}\log(p(u(y,r_i)|\Theta)) \tag{5.5}
$$

where $u(y,r_i)$ is the utility between the hypotheses and the reference r_i. The loss measures how likely the utilities are given the predicted parameters.

5.3 Approach

We used the best model found in chapter 3 for the COMET utility, however this time we train this model to predict the distribution of the utilities.

5.3.1 Fitting a Distribution

When fitting a mixture model, the choice of the family the distribution comes from is important. A first choice of distributions to use in the mixture is the Gaussian or the Student-t distribution, as they are symmetrical and have a non-zero probability for any real value. The student-t distribution has as an added benefit over the Gaussian, namely that we can control the kurtosis [3]. To capture the skew, other distributions could be used, such as the Weibull distribution. But during training, this can lead to problems as it can assign zero probability to utility values which in turn could lead to infinite loss.

3: Kurtosis controls how "sharp" a peak is.

For a mixture model with $l \in \mathbb{N}$ Gaussians the neural model predicts $3 \cdot l$ parameters: the location and scale parameter for each Gaussian as well as the weights of each distribution. When using a student-t distribution, $4 \cdot l$ parameters are predicted. We predict the same parameters as the Gaussian and additionally, the degrees of freedom (v) are computed.

For the location, a tanh activation function multiplied by 2.5 is used to make sure the location stays in the range of observed utilities. For both the scale and v parameters the softplus activation function is used. For the v parameter, 2.1 is added. Approximating the mean with sampling when the v is smaller than 2 becomes impossible. When $v < 1$ the mean is undefined. When $v < 2$ the variance is infinite [4]. The weights are computed with the help of a Softmax function.

4: 2.1 is chosen to make sure we stay "far" away from 2 to keep numerical stability.

5.3.2 Model, Hyperparameter Search, and Training

We train the full decoder COMET model to fit a mixture with $n = 2$ and $n = 3$ components for both the Gaussian and Student's-t distribution. We keep most of the hyperparameters we found for the regression model, but we search for reasonable learning rate, batch size, gradient clip value, gamma, and the dropout rate with the 10 hypotheses 100 reference dataset. The models are trained to optimize equation 5.5. After the hyperparameter search is completed we use the best found parameters and train on the full dataset with 100 hypotheses and 1000 references.

5.4 Results

The models are evaluated on the test dataset. The negative log-likelihood (NLL) and the mean squared error (MSE) are computed as well as the Kendall τ statistics. Furthermore, we use the model as a decision rule. Lastly, we graph the learned density versus the density of the samples for visual inspection for some randomly picked samples.

5.4.1 Statistics

In table 5.1 you see the loss on the test set of the models as well as the MSE and the Kendall τ statistics. We first note that the Student-t has a lower NLL and thus fits the data better than the Gaussians, but this doesn't translate to lower MSE, a higher median Kendall τ, or better COMET and BLEURT scores. There is also no big difference between the mixture models and the regression model. Furthermore using more components also decreases the NLL loss but doesn't result in better scores on the other metrics.

Table 5.1: Statistics for the mixture models on the test set. The NLL loss, MSE, and Kendall τ are shown. All the models attain very similar scores.

Model	NLL	MSE	Kendall τ
Regression	-	3.1e-2	0.831
Gaussian - 2	7.9e-2	2.9e-2	0.825
Gaussian - 3	4.3e-2	3.1e-2	0.818
Student-t - 2	-7.5e-3	3.1e-2	0.829
Student-t - 3	-5.8e-2	3.0e-2	0.824

Table 5.2: COMET and BLEURT scores for mixture models when used as a decision rule. Note that all models achieve very similar COMET and BLEURT scores.

Model	COMET	BLEURT
Regression	0.76	0.60
Gaussian - 2	0.77	0.61
Gaussian - 3	0.76	0.62
Student-t - 2	0.76	0.61
Student-t - 3	0.77	0.62

5.4.2 Sampled Distributions

In figure 5.2 we can see the distribution when we get 1000 samples from the true distribution, underneath are samples from the different mixture models. From these graphs, we can see that the mixture of student t-distributions is both better at capturing the multi-modality of the distribution and has more steep peaks than the mixtures using Gaussians. This could explain why the student-t mixtures have a lower loss than the mixture of Gaussians. Although some mixtures have a better fit to the distribution than others, the means do not vary that much, explaining why a better fit does not necessarily result in better Bayes risk predictions.

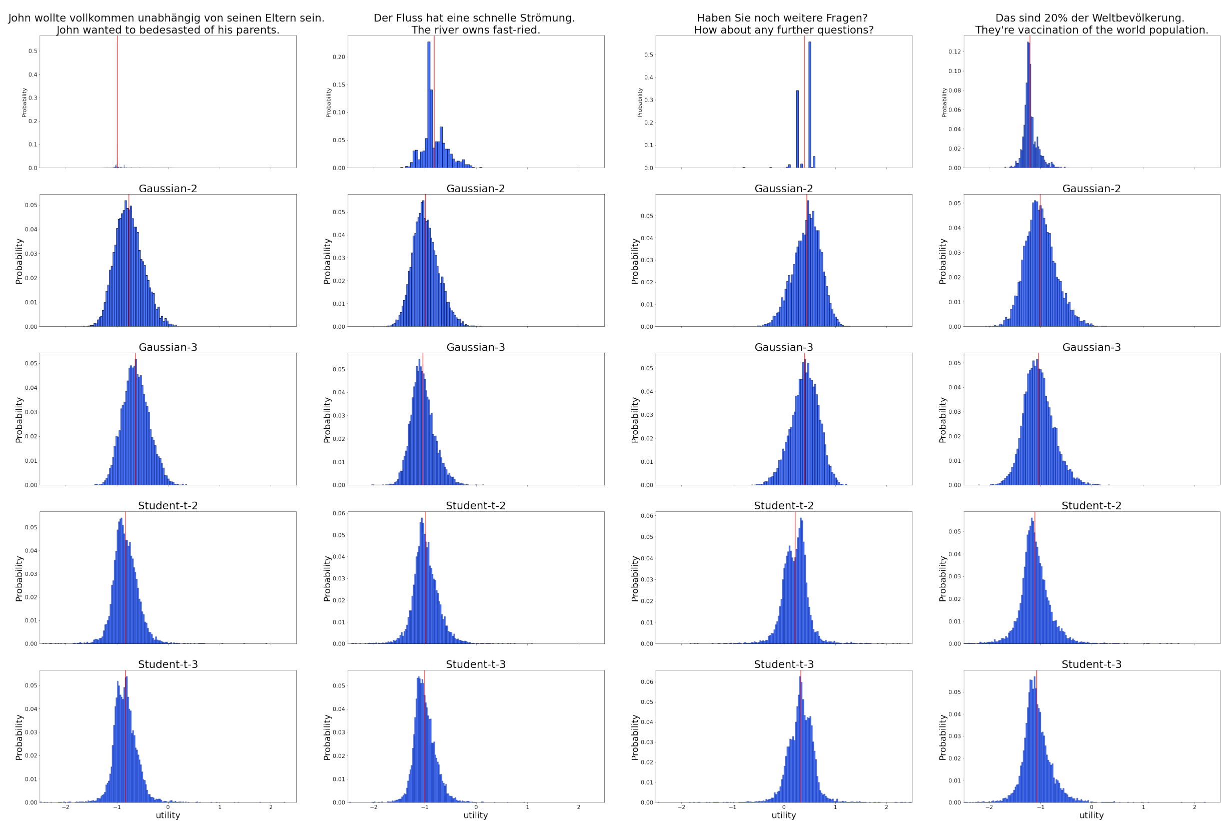


Figure 5.2: The distribution according to the mixture models. The red line indicates the mean of the samples. Note that the top-left distribution has one very large single peak underneath the red line. although some models capture the distribution better than others, the mean doesn't vary much.

5.5 Research Questions

To finish this chapter we answer the research question Q2 and its sub-questions.

Q2.1: How do mixture models compare to the predictive model?

We can conclude that mixture models are *not* a meaningful improvement over the regression model. This is likely due to the fact that although the model may be able to capture the distribution, it doesn't change much about the capabilities in predicting the mean.

Q2.2: How does the choice of distribution for the mixture model influence the mixture model?

The choice of distribution as well as the number of components influence the negative log-likelihood loss. The mixtures of student-t distributions do have a better fit than the mixtures of Gaussians. Unfortunately, a better fit does not translate into better estimates for the Bayes risk.

Q2: Can neural mixture models be used to predict the Bayes risk?

Neural mixture models can be used to predict the Bayes risk, but they don't outperform the regression model.

5.6 Conclusion

In this chapter we looked at another approach for predicting the Bayes risk, namely fitting a mixture of distributions. As it turns out the models do not improve in actually predicting the Bayes risk compared to simply regressing to the 1000-MC estimate. This could be due to the fact that lower negative log likelihoods don't necessarily translate into meaningful changes in the predicted Bayes risk. In the next chapter we will look at our last attempt to construct the MBR translation: directly searching for it by applying backpropagation.

Minimum Bayes Risk Decoding with Backpropagation 6

In this chapter, we discuss an approach for constructing the minimum Bayes risk translation with the help of backpropagation. As we quickly found out that this approach wouldn't work out we started working on the methods discussed in the previous chapter. We, however, still find it an interesting approach and worth mentioning what we tried and explain why this approach doesn't work.

6.1 Introduction

Instead of first approximating the Bayes risk of a number of hypotheses and then picking the hypothesis with the lowest Bayes risk, we could also directly construct the Bayes risk translation. In this section, we motivate this approach, briefly touch upon our result and give an outline of this chapter.

6.1.1 Motivation

As discussed, calculating the Bayes risk is computationally expensive. A way to find the minimum Bayes risk (MBR) translation is by constructing it directly without, explicitly, calculating the Bayes risk of multiple hypotheses. The hope is that constructing the MBR translation is more efficient than calculating the Bayes risk for multiple hypotheses.

One possible approach to construct the MBR translation is to use gradient descent. This can be done whenever the utility function u is differentiable, which can be the case when a neural utility function is used. The tokens themselves are discrete, therefore we need continuous-discrete relaxation. This is done with the help of straight-through estimation [45]. Another approach we tried is optimizing directly in the embedding space of the utility function and mapping back to nearby tokens.

6.1.2 Contributions and outline

The contribution of this chapter is that we show that constructing the MBR translation with the help of back-propagation doesn't seem possible *unless* the utility is aware of the distribution of the translations. To be precise we show that:

1. Using back-propagation through COMET to find the MBR translation results in nonsensical translations
2. COMET has regions in its embedding space that correspond to high COMET scores but do not have meaningful sentences in the neighborhood.
3. This problem does not seem to be unique to COMET as a neural similarity measurement based on DistilRoberta has the same problems.

In this chapter, we first discuss the relevant background as well as relevant literature. Next, we discuss our approach in detail and then show a random sample of the translations that we found. We finish by arguing why backpropagation does not work and what possible solutions could be.

6.2 Background

In this background section, we discuss relevant concepts not previously discussed in the past chapters. Furthermore, we discuss relevant literature.

6.2.1 Preliminaries

Straight-through estimation

When we have a function that is non-differentiable, such as the argmax function, gradients are not defined. A way to tackle this is to use straight-through estimation.

1: The symbol $\circ$ indicates function composition

Let some vector v be the input of $h = g \circ f$[1], with g being differentiable while f isn't. For straight-through (ST) estimation we use the following derivative:

$$\frac{\delta v}{\delta (g \circ f)} \overset{\text{ST}}{\approx} \frac{\delta f(v)}{\delta g} \tag{6.1}$$

Thus when calculating the gradient for v we use the gradients of the vector $f(v)$ and thus act as if f is the identity function [45].

Distance measures

To measure the distance between two token embeddings, cosine similarity and euclidean distance were used. The cosine similarity of two embeddings e_1 and e_2 is defined as:

$$\text{dist}_{\text{cos}}(e^{(1)}, e^{(2)}) = \frac{e^{(1)} \cdot e^{(2)}}{||e^{(1)}||||e^{(2)}||} \tag{6.2}$$

and always falls on an interval of [-1, 1]. For cosine similarity, the magnitude of the two vectors is not important, but the angle between them is.

The definition of Euclidean distance is:

$$\text{dist}_{\text{Euclidean}}(e^{(1)}, e^{(2)}) = \sqrt{\sum_{i=1}^{d} (e^{(1)} - e^{(2)})^2} \tag{6.3}$$

and can be any real value.

6.2.2 Relevant literature

Using gradient information by backpropagating through a neural network to find an input with a desired property is not a new concept. For image classification it is used to construct adversarial examples [46]. Similar ideas can also be used to influence the output of natural language processing models. For example, prompt tuning [47] can be used to construct new tokens that will be prepended to the input to fine-tune a large language model for language understanding. However, with prompt-tuning the embedding of these new tokens is updated, while the rest of the model stays frozen. The paper of Hambardzumyan, Khachatrian, and May [48] uses a similar approach called WARP (Word-level Adversarial ReProgramming) to fine-tune models for language understanding. The difference is that WARP doesn't update the embedding space of new tokens but instead uses existing tokens.

Another example of backpropagation through the model is given by Ebrahimi, Lowd, and Dou [49]. In their work, they use small character-level perturbations of the input to find adversarial examples. Backpropagation is used to find out which perturbations change the output the most. Lastly, Wallace et al. [50] finds prompts to find adversarial trigger words that influence the model to output inflammatory language.

A last interesting example is the work from Dathathri et al. [51]. In their paper, they train (neural) attribute models to predict attributes of a sentence such as sentiment from the last hidden state of the decoder. These models then can be used to update the last hidden state by backpropagation to control the attributes of the generated output.

6.3 Approach

With this approach, we want to directly find a hypothesis h such that equation 2.6 is maximized. We do this as follows: one starts with n random sentences with different lengths. These get fed through the utility function together with m references and, if applicable, the source. The derivative with respect to the tokens/embeddings is taken and the tokens/embeddings are updated by applying stochastic gradient descent. To be precise we want to find a sequence of tokens $h = h_1 \dots h_l$ such that $\hat{\mu}_u(h, x, \theta)$ is maximized.

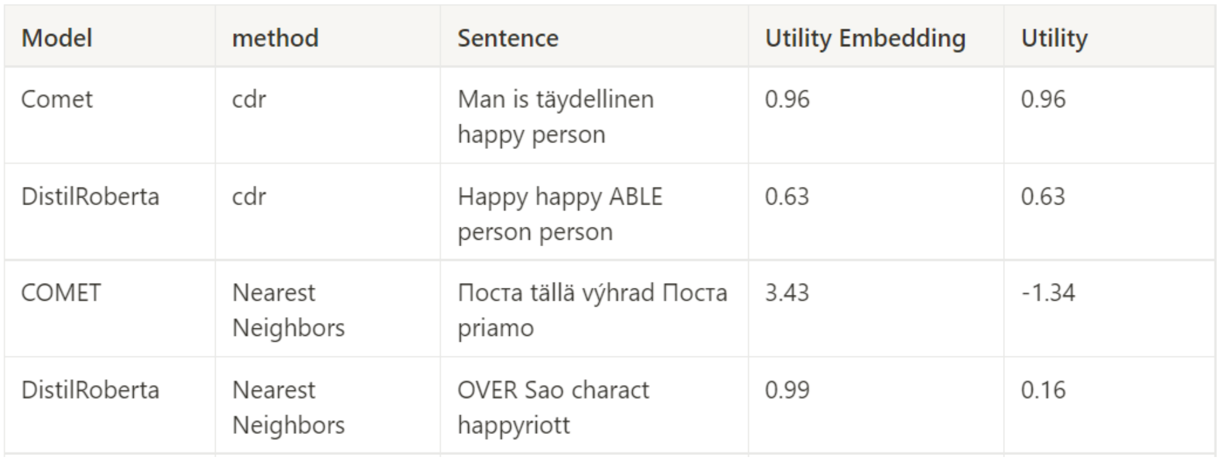
First, we note that the tokens $h_1 \dots h_l$ are discrete and represented by one-hot vectors, using discrete relaxation makes them continuous. We thus allow for tokens to be arbitrary real vectors. Before feeding the vectors into the model we use the softmax function followed by the argmax function to get the index of the most likely token. The softmax function is applied to make sure the weights are normalized and lay between 0 and 1. The straight-through estimator is then used to update the token vectors. We do this until the tokens do not change anymore. To get the final tokens we simply apply the softmax and then argmax method to find the token that maximizes $\hat{\mu}_u(h, x, \theta)$ for each timestep. As the straight-through estimator is biased [45] we also try optimizing directly in embedding space. To optimize in the embedding space we start with a sequence of token embeddings: $e = (e^{(1)}, \dots e^{(l)})$. We then directly optimize the individual embeddings with help of backpropagation. After the embeddings do not change anymore we check for each embedding what the nearest embedding is from all the tokens from the vocabulary. For a measure of distance, both Euclidean distance and cosine similarity were tried. The tokens with the nearest embeddings are then used for the final translation.

We took some sentences from the ISWLT2017 validation set [52] for German to English sentences. We used a pre-trained transformer-based NMT model provided by the FairSeq sequence modeling toolkit[53] to generate 5 reference sentences for each source. We checked for two neural utilities, COMET [11] and a sentence similarity network from Hugging Face called "all-distilroberta-v1". COMET itself uses XlM-Roberta which works with multiple languages, and as we will see, this could result in that tokens from outside the target language being introduced. The sentence similarity network is only used for English sentences and therefore doesn't have that problem.

6.4 Results

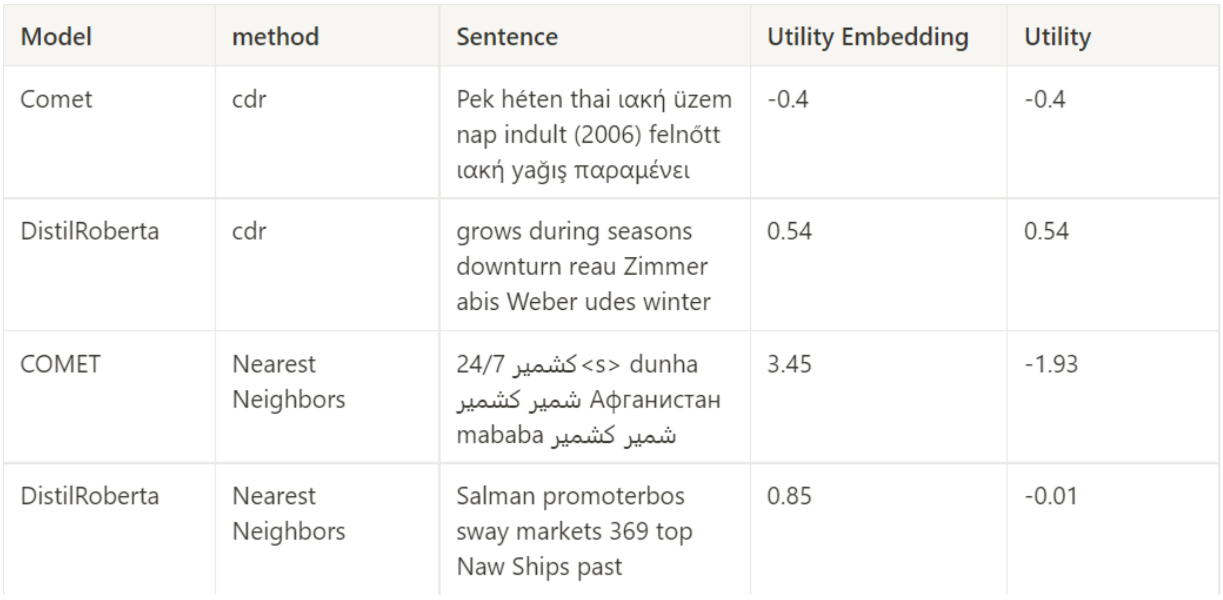
When we tried to backpropagate to create a translation, we quickly found out that it didn't work. We directly saw that this approach generated nonsensical sentences. Two examples are listed in figure 6.1 and 6.2 [2]. The utility refers to the utility of the 5-MC estimate of the given sentence. The utility embedding refers to the utility when we use the embeddings instead of the nearest tokens. Interestingly the utility in embedding space is relatively high while when we pick the nearest token we end up with a low utility.

2: There were hard to fix font display issues due to the wide variety of fonts used in the different languages, therefore figures are used instead of LaTeX tables.



Model	method	Sentence	Utility Embedding	Utility
Comet	cdr	Man is täydellinen happy person	0.96	0.96
DistilRoberta	cdr	Happy happy ABLE person person	0.63	0.63
COMET	Nearest Neighbors	Поста tällä výhrad Поста priamo	3.43	-1.34
DistilRoberta	Nearest Neighbors	OVER Sao charact happyriott	0.99	0.16

Figure 6.1: Figure showing the constructed translation as well as the utility scores. Source: Das ist ein glücklicher Mensch, Target: That is a happy person. Note that the utility scores are relatively high, but the sentences are nonsensical.



Model	method	Sentence	Utility Embedding	Utility
Comet	cdr	Pek héten thai ιακή üzem nap indult (2006) felnőtt ιακή yağış παραμένει	-0.4	-0.4
DistilRoberta	cdr	grows during seasons downturn reau Zimmer abis Weber udes winter	0.54	0.54
COMET	Nearest Neighbors	24/7 کشمیر<s> dunha شمیر کشمیر Афганистан mababa شمیر کشمیر	3.45	-1.93
DistilRoberta	Nearest Neighbors	Salman promoterbos sway markets 369 top Naw Ships past	0.85	-0.01

Figure 6.2: Figure showing the constructed translation as well as the utility scores. Source: Sie wächst im Winter und schrumpft im Sommer. Target: It expands in winter and contracts in summer. Note that the utility scores are relatively high, but the sentences are nonsensical.

6.5 Discussion

First, we give some remarks on why this approach fails, then we answer the research question Q3.

6.5.1 Why Does This Approach Fail?

Unfortunately, it turned out that this approach did not work at all. The resulting sentences turned out to be nonsensical. We could think of at least two reasons why this is the case.

First of all, the number of sentences both models are trained on is very sparse compared to the number of possible sentences. For example, the DistillRoberta model has more than 50.000 different tokens in its vocabulary. When we have a sentence containing 2 tokens there already exist more than $50.000^2 \approx 2.5e11$ different possible sentences, but only a small portion of those are meaningful. The model gives a score to every input it gets, but it only sees a small portion of the space of possibilities when training. Even if the model is able to generalize for a large part, there still can be cases in which the model assigns a too high utility score for out-of-distribution sentences. This problem is more apparent when looking at the utility of the embeddings when doing backpropagation directly in the embedding space. In the tables, you can see that the utility assigned by the models is very high compared to when we construct the translation by finding the nearest neighbor. For example, in table 6.2 we see that the resulting sentence gets a utility score of 3.45 in the embedding space while the resulting sentence gets a utility of -1.93. An idea to fix this is to incorporate the sentence likelihood to make the model regress to a sentence that is both likely and has high utility. However, as the NMT model and the utility function have non-matching embedding spaces we couldn't think of a way to make this approach work for the given utility functions.

Secondly, COMET has an input space with more languages than only English. For example, it also accepts tokens from the Arabic alphabet. As we can observe in tables 6.1 and 6.2 this method results in non-English tokens. The DistilRoberta model doesn't have this problem.

6.5.2 Research Questions

To conclude we answer the research questions.

Q3.1: Can the MBR translation be constructed by updating the tokens with the help of continuous-discrete relaxation and straight-through estimation?

The MBR translation can *not* be constructed this way. The main problem is that the tokens want to move somewhere in embedding space where the utility is high, but the translation itself is nonsensical.

Q3.2: Can the MBR translation be constructed by updating the tokens in the embedding space of the utility function?

The MBR translation can *not* be constructed by directly updating the tokens in embedding space. The embeddings end up someplace in the embedding space that results in high utility scores, but when picking the closest tokens the utility drops significantly.

Q3: Can backpropagation through a utility function be used to construct the MBR translation?

It doesn't seem possible to use backpropagation to directly construct the MBR translation. If the likelihood of the translation could be taken into account it might be possible.

6.6 Future Work

When we create our own utility function, we could use an approach such as plug-and-play models [51] to construct sentences that are both likely and have a high utility. With this approach, the attribute models assign the utility of the translation based on the final hidden state of the decoder given some reference and optionally the source. However, this requires training a simple model that would fit on top of the NMT model we use. This is a way less general approach as not every utility function can be used: it seems that for every NMT model, a new utility function has to be trained.

There is of course a chance that we missed something and there is a way to incorporate the probability of a translation for arbitrary (neural) utility functions. So any attempt to combine the sentence likelihood with the utility function could be interesting to try.

6.7 Conclusion

It doesn’t seem possible to use backpropagation to construct the MBR translation unless the utility function is aware of the likelihood of a given translation or the utility function doesn’t assign high values to out-of-distribution sentences. In the last chapter, we discuss our findings and point to possible future work.

Discussion 7

In this chapter, we briefly summarize the results, give directions for future work, and conclude this thesis.

7.1 Summary

In this thesis, we explored possibilities for finding the MBR translation. The main approach taken was predicting the Bayes Risk by regressing to the 1000-MC estimate. This approach can be used to create models that can be used as effective decision rules, however, and outperform m-MC estimation in terms of MSE and ranking capabilities when m is small. Using mixture models doesn't seem to have any benefit over regressing to the mean, which seems to be due to the mean being relatively stable under different mixture models. Lastly using backpropagation to directly find the MBR translation doesn't work as the utility function doesn't take the likelihood of the sentences into account.

7.2 Future work

In this work, we trained the predictive models "only" for one NMT model trained on one dataset. Trying this approach on multiple datasets with different NMT models and different sizes of splits of the data could be an interesting next step to see how those results differ from ours. Furthermore, an autoregressive model was used, but there also exist non-autoregressive models, for which MBR decoding is possible. Predictive models could be built on top of these types of models and their hidden states and/or token statistics.

We didn't explore features for the reference style models in full detail. Maybe more powerful models can be made when certain features of sampled references are used.

Another approach that one can take is to perform learning to rank instead of regression or fitting a mixture model. In this approach, a model gets two hypotheses, and features about those hypotheses, and has to predict which of the two hypotheses has a lower Bayes risk. Given such a model we are able to rank a list of hypotheses to select the one with the lowest Bayes risk.

Another approach we thought about is to train a model to directly construct the MBR translation. For example, a policy network could be trained to learn to select tokens with a low expected Bayes risk. As MC estimation can be computationally expensive, a predictive model could be used to predict the Bayes risk during training.

Lastly, one major reason for choosing MBR decoding over beam search decoding is that it exhibits fewer pathologies. However, when using these kinds of models, it could be the case that the pathologies are either reintroduced or new pathologies are added[1]. An in-depth analysis of the output of these models could be done to see if and how these models influence the observed pathologies.

1: As long as the Kendall τ statistic is not 1 because when it is 1 we are certain that the predictive models give the same ranking, and thus the same translation as the 1000-MC estimate

7.3 Conclusion

In this thesis, we explored two different approaches for estimating the MBR objective and one approach for constructing the MBR translation directly. Most notable we found several models that outperform MC estimation when used as a decision rule. These models can be an interesting starting point for more research. We hope that this research gives additional insights into Bayes risk decoding and points towards interesting research questions to further understand Bayes risk decoding and how it can be made more efficient.

Appendix

A Appendix

A.1 Practicalities

For this research, we had a relatively small multi-language dataset, but nonetheless when training the predictive model we encountered computational and memory issues. Generating the hypotheses and references during training and computing the Bayes Risk is way too computationally expensive to have normal training times. We, therefore, opted to generate and compute as much as possible prior to training. Storing the generated sentences in a dictionary together with the count was necessary to keep the resulting data small. We did the same for the calculated COMET scores.

When using expensive features, such as hidden states of the NMT model or features used in COMET, it is possible to precompute those. Although it might be necessary to split the dataset into subsets to be able to work with it during training. All in all these practicalities are important when training the predictive model and take some engineering effort on the part of the researcher as training can be significantly slow down, or even be infeasible, if this is not properly taken care of.

A.2 Hyperparameter

In table A.1 we show the general hyperparameters used during the hyperparameter search. For the feed-forward layers, we used 3 sizes: small, medium, and large[1]. The dimensions are mentioned in table A.2[2].

1: The value -1 indicates that the size is dependent on the input of the linear layers

2: The table contains the practical batch size which equals $batchsize \times accumulategradbatches$

There are also model-specific hyperparameters which are discussed in the sections below. The difference in hyperparameters stems from the fact that the models use different features and thus the inputs have different dimensions. More details as well as the found parameters can be found in the code repository of this project: https://github.com/gersonfoks/towards-efficient-bayes-risk-decoding

Table A.1: General Hyperparameters

Parameter	**Values**	**Scale**
Learning Rate	$[1.0e-4, 1.0e-2]$	log
Weight Decay	$[1.0e-9, 1.0e-5]$	log
Dropout	$[0.01, 0.9]$	uniform
Gamma	$[0.5, 1.0]$	uniform
Practical Batch size	$\{128, 256, 512\}$	-
Gradient clip value	$[1.0, 5.0]$	uniform

Table A.2: Feed forward layer sizes

Size	dimensions
Small	$[-1, 128, 1]$
Medium	$[-1, 256, 128, 1]$
Large	$[-1, 512, 256, 128, 1]$

Baseline model For the baseline model, we have two additional parameters: the embedding size and the hidden state size of the BiLSTM. For this we have embedding size $\in \{128, 256, 512\}$ and hidden state size $\in \{128, 256, 512\}$

Last Hidden State Model As the embedding size is already determined we only have an additional hidden state size for the BiLSTM for which we have hidden state size $\in \{256, 512\}$

Token Statistics Model For the token statistics model, we need to embed the statistics and again have a hidden state for the BiLSTM. For this we have embedding size $\in \{64, 128, 256\}$ and hidden state size $\in \{64, 128, 256\}$

Full Decoder Model As the embedding size is already determined we only have an additional hidden state size for the BiLSTM for which we have hidden state size $\in \{64, 128, 256, 512\}$

Full Decoder Model - no token statistics The full decoder model - no token statistics uses the same hyperparameters as the Full Decoder Model.

COMET Feature Model The COMET feature model has one additional size for the feed-forward layer, namely "extra large", which has dimensions $[-1, 1024, 512, 256, 128, 1]$

Full Decoder COMET Feature Model This model uses the same dimensions for the BiLSTM as for the Full Decoder model but leaves the other hyperparameters open.

Basic reference model The basic reference model uses the same hyperparameters as the Full Decoder model in its search.

Embedded Reference Model The embedded reference model is an attention model, therefore additional hyperparameters are the number of heads used. The number of heads is picked from the set $\{2, 4, 8\}$. The rest of the parameters stay the same.

Bibliography

[1] Felix Stahlberg and Bill Byrne. 'On NMT Search Errors and Model Errors: Cat Got Your Tongue?' In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Hong Kong, China: Association for Computational Linguistics, Nov. 2019, pp. 3356–3362. DOI: 10.18653/v1/D19-1331 (cited on pages 1, 12).

[2] Bryan Eikema and Wilker Aziz. 'Is MAP Decoding All You Need? The Inadequacy of the Mode in Neural Machine Translation'. In: *Proceedings of the 28th International Conference on Computational Linguistics*. Barcelona, Spain (Online): International Committee on Computational Linguistics, Dec. 2020, pp. 4506–4520. DOI: 10.18653/v1/2020.coling-main.398 (cited on pages 1, 9, 12).

[3] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 'Sequence to sequence learning with neural networks'. In: *Advances in neural information processing systems*. 2014, pp. 3104–3112 (cited on pages 1, 7, 8).

[4] Philipp Koehn and Rebecca Knowles. 'Six Challenges for Neural Machine Translation'. In: *Proceedings of the First Workshop on Neural Machine Translation*. 2017, pp. 28–39 (cited on pages 1, 12).

[5] Myle Ott et al. 'Analyzing uncertainty in neural machine translation'. In: *International Conference on Machine Learning*. PMLR. 2018, pp. 3956–3965 (cited on pages 1, 12).

[6] Mathias Müller and Rico Sennrich. 'Understanding the Properties of Minimum Bayes Risk Decoding in Neural Machine Translation'. In: *The Joint Conference of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing*. Association for Computational Linguistics. 2021, pp. 259–272 (cited on pages 1, 12).

[7] Bryan Eikema and Wilker Aziz. 'Sampling-Based Minimum Bayes Risk Decoding for Neural Machine Translation'. In: *arXiv preprint arXiv:2108.04718* (2021) (cited on pages 1, 2, 9).

[8] Carolyn Pillers Dobler. *Mathematical statistics: Basic ideas and selected topics*. 2002 (cited on pages 1, 8).

[9] Maja Popović. 'chrF++: words helping character n-grams'. In: *Proceedings of the second conference on machine translation*. 2017, pp. 612–618 (cited on pages 1, 9–11).

[10] Maja Popović. 'chrF: character n-gram F-score for automatic MT evaluation'. In: *Proceedings of the Tenth Workshop on Statistical Machine Translation*. 2015, pp. 392–395 (cited on pages 1, 9, 11).

[11] Ricardo Rei et al. 'COMET: A Neural Framework for MT Evaluation'. In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 2020, pp. 2685–2702 (cited on pages 1, 9, 10, 21, 46).

[12] Thibault Sellam, Dipanjan Das, and Ankur Parikh. 'BLEURT: Learning Robust Metrics for Text Generation'. In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Online: Association for Computational Linguistics, July 2020, pp. 7881–7892. DOI: 10.18653/v1/2020.acl-main.704 (cited on pages 3, 9, 12).

[13] Kyunghyun Cho et al. 'Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation'. In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1724–1734. DOI: 10.3115/v1/D14-1179 (cited on page 7).

[14] Nal Kalchbrenner and Phil Blunsom. 'Recurrent Continuous Translation Models'. In: *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*. Seattle, Washington, USA: Association for Computational Linguistics, Oct. 2013, pp. 1700–1709 (cited on page 7).

[15] Sepp Hochreiter and Jürgen Schmidhuber. 'Long short-term memory'. In: *Neural computation* 9.8 (1997), pp. 1735–1780 (cited on pages 7, 8).

[16] Hasim Sak, Andrew W Senior, and Françoise Beaufays. 'Long Short-Term Memory Based Recurrent Neural Network Architectures for Large Vocabulary Speech Recognition'. In: (2014) (cited on pages 7, 8).

[17] Ashish Vaswani et al. ‘Attention is all you need’. In: *Advances in neural information processing systems* 30 (2017) (cited on page 8).

[18] Jonas Gehring et al. ‘A Convolutional Encoder Model for Neural Machine Translation’. In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Vancouver, Canada: Association for Computational Linguistics, July 2017, pp. 123–135. DOI: 10.18653/v1/P17-1012 (cited on page 8).

[19] Jasmijn Bastings et al. ‘Graph Convolutional Encoders for Syntax-aware Neural Machine Translation’. In: *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. Copenhagen, Denmark: Association for Computational Linguistics, Sept. 2017, pp. 1957–1967. DOI: 10.18653/v1/D17-1209 (cited on page 8).

[20] Kyunghyun Cho et al. ‘On the Properties of Neural Machine Translation: Encoder–Decoder Approaches’. In: *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*. 2014, pp. 103–111 (cited on page 8).

[21] Kishore Papineni et al. ‘Bleu: a method for automatic evaluation of machine translation’. In: *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*. 2002, pp. 311–318 (cited on page 9).

[22] Robert A Wagner and Michael J Fischer. ‘The string-to-string correction problem’. In: *Journal of the ACM (JACM)* 21.1 (1974), pp. 168–173 (cited on page 9).

[23] Miloš Stanojević and Khalil Sima’an. ‘BEER: BEtter Evaluation as Ranking’. In: *Proceedings of the Ninth Workshop on Statistical Machine Translation*. Baltimore, Maryland, USA: Association for Computational Linguistics, June 2014, pp. 414–419. DOI: 10.3115/v1/W14-3354 (cited on page 9).

[24] Tianyi Zhang et al. ‘BERTScore: Evaluating Text Generation with BERT’. In: *International Conference on Learning Representations* (cited on page 9).

[25] Markus Freitag et al. ‘Results of the WMT21 Metrics Shared Task: Evaluating Metrics with Expert-based Human Evaluations on TED and News Domain’. In: *Proceedings of the Sixth Conference on Machine Translation*. Online: Association for Computational Linguistics, Nov. 2021, pp. 733–774 (cited on page 9).

[26] Alexis Conneau et al. ‘Unsupervised Cross-lingual Representation Learning at Scale’. In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. 2020, pp. 8440–8451 (cited on page 10).

[27] Unbabel. *Unbabel FAQ for the COMET model*. 2022. URL: https://unbabel.github.io/COMET/html/faqs.html (visited on 09/20/2022) (cited on page 10).

[28] Jörg Tiedemann. ‘Parallel Data, Tools and Interfaces in OPUS’. In: *Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC’12)*. Ed. by Nicoletta Calzolari (Conference Chair) et al. Istanbul, Turkey: European Language Resources Association (ELRA), 2012 (cited on pages 11, 16, 18).

[29] Maurice G Kendall. ‘The treatment of ties in ranking problems’. In: *Biometrika* 33.3 (1945), pp. 239–251 (cited on page 11).

[30] Jacob Devlin Ming-Wei Chang Kenton and Lee Kristina Toutanova. ‘BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding’. In: *Proceedings of NAACL-HLT*. 2019, pp. 4171–4186 (cited on page 12).

[31] Aviral Kumar and Sunita Sarawagi. ‘Calibration of encoder decoder models for neural machine translation’. In: *arXiv preprint arXiv:1903.00802* (2019) (cited on page 12).

[32] Huda Khayrallah and Philipp Koehn. ‘On the Impact of Various Types of Noise on Neural Machine Translation’. In: *Proceedings of the 2nd Workshop on Neural Machine Translation and Generation*. Melbourne, Australia: Association for Computational Linguistics, July 2018, pp. 74–83. DOI: 10.18653/v1/W18-2709 (cited on page 12).

[33] Katherine Lee et al. ‘Hallucinations in neural machine translation’. In: *Open Review* (2018) (cited on page 12).

[34] Mathias Müller, Annette Rios, and Rico Sennrich. 'Domain Robustness in Neural Machine Translation'. In: *Proceedings of the 14th Conference of the Association for Machine Translation in the Americas (Volume 1: Research Track)*. Virtual: Association for Machine Translation in the Americas, Oct. 2020, pp. 151–164 (cited on page 12).

[35] Markus Freitag et al. 'High Quality Rather than High Model Probability: Minimum Bayes Risk Decoding with Neural Metrics'. In: *Transactions of the Association for Computational Linguistics* 10 (2022), pp. 811–825. DOI: 10.1162/tacl_a_00491 (cited on page 13).

[36] Patrick Fernandes et al. 'Quality-Aware Decoding for Neural Machine Translation'. In: *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. 2022, pp. 1396–1412 (cited on page 13).

[37] Mirac Suzgun, Luke Melas-Kyriazi, and Dan Jurafsky. 'Follow the Wisdom of the Crowd: Effective Text Generation via Minimum Bayes Risk Decoding'. In: *arXiv e-prints* (2022), arXiv–2211 (cited on page 13).

[38] Jörg Tiedemann and Santhosh Thottingal. 'OPUS-MT — Building open translation services for the World'. In: *Proceedings of the 22nd Annual Conferenec of the European Association for Machine Translation (EAMT)*. Lisbon, Portugal, 2020 (cited on pages 16, 18).

[39] Ari Holtzman et al. 'The Curious Case of Neural Text Degeneration'. In: *arXiv e-prints* (2019), arXiv–1904 (cited on page 17).

[40] Angela Fan, Mike Lewis, and Yann Dauphin. 'Hierarchical Neural Story Generation'. In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 2018, pp. 889–898 (cited on page 17).

[41] Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009 (cited on page 17).

[42] Huggingface. *Huggingface Helsinki-NLP/opus-mt-de-en*. 2022. URL: https://huggingface.co/Helsinki-NLP/opus-mt-de-en (visited on 09/20/2022) (cited on page 18).

[43] James Bergstra et al. 'Algorithms for hyper-parameter optimization'. In: *Advances in neural information processing systems* 24 (2011) (cited on page 19).

[44] David A Van Veldhuizen, Gary B Lamont, et al. 'Evolutionary computation and convergence to a pareto front'. In: Citeseer (cited on page 32).

[45] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. 'Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation'. In: *arXiv e-prints* (2013), arXiv–1308 (cited on pages 43–45).

[46] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. 'Explaining and Harnessing Adversarial Examples'. In: *arXiv e-prints* (2014), arXiv–1412 (cited on page 45).

[47] Brian Lester, Rami Al-Rfou, and Noah Constant. 'The Power of Scale for Parameter-Efficient Prompt Tuning'. In: *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. 2021, pp. 3045–3059 (cited on page 45).

[48] Karen Hambardzumyan, Hrant Khachatrian, and Jonathan May. 'WARP: Word-level Adversarial ReProgramming'. In: *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. 2021, pp. 4921–4933 (cited on page 45).

[49] Javid Ebrahimi, Daniel Lowd, and Dejing Dou. 'On Adversarial Examples for Character-Level Neural Machine Translation'. In: *Proceedings of the 27th International Conference on Computational Linguistics*. 2018, pp. 653–663 (cited on page 45).

[50] Eric Wallace et al. 'Universal Adversarial Triggers for Attacking and Analyzing NLP'. In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Hong Kong, China: Association for Computational Linguistics, Nov. 2019, pp. 2153–2162. DOI: 10.18653/v1/D19-1221 (cited on page 45).

[51] Sumanth Dathathri et al. 'Plug and Play Language Models: A Simple Approach to Controlled Text Generation'. In: *International Conference on Learning Representations* (cited on pages 45, 48).

[52] Sakriani Sakti and Masao Utiyama, eds. *Proceedings of the 14th International Conference on Spoken Language Translation*. Tokyo, Japan: International Workshop on Spoken Language Translation, Dec. 2017 (cited on page 46).

[53] Myle Ott et al. 'fairseq: A Fast, Extensible Toolkit for Sequence Modeling'. In: *Proceedings of NAACL-HLT 2019: Demonstrations*. 2019 (cited on page 46).