Universität Stuttgart
Institut für Maschinelle Sprachverarbeitung
Pfaffenwaldring 5b
D-70569 Stuttgart

Master's Thesis

Unsupervised Induction of Multi Bottom-up Tree Transducers

Robin Kurtz

Start: 01.08.2014
End: 17.03.2015

Prüfer: Dr. rer. nat. Andreas Maletti
Prof. Dr. Jonas Kuhn
Betreuer: Dipl.-Linguistin Nina Seemann

Hiermit erkläre ich,

- dass ich meine Arbeit selbständig verfasst habe,
- dass ich keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommenen Aussagen als solche gekennzeichnet habe,
- dass die eingereichte Arbeit weder vollständig noch in wesentlichen Teilen Gegenstand eines anderen Prüfungsverfahrens gewesen ist,
- dass ich die Arbeit weder vollständig noch in Teilen bereits veröffentlicht habe und
- dass das elektronische Exemplar mit den anderen Exemplaren übereinstimmt.

_____

Robin Kurtz

Contents

List of Figures

1. Introduction

Computers have become ubiquitous in our lives in the last few years, especially since the advent of smartphones and other portable devices. Natural language processing (NLP) tasks have greatly improved and are commonly used on said portable devices, creating the need for more functionalities, new tasks and improved performance. These tasks do not only include the production and recognition of speech, but also understanding and generation of natural language. They enable us to communicate with computers using only natural language. However, getting machines to understand human language is very complex and creates a wide field of research for subtasks that need to be explored such as parsing, sentiment analysis and named entity recognition. While some of these subtasks build the foundation of multiple NLP applications, such as parsing and morphological segmentation, others use their results to produce more end-user centric systems such as sentiment analysis and automatic summarization. One of the "higher tier" tasks, is the automatic translation of natural language into another language. While the translation of small sentences and simple phrases works well for some language pairs, there is much work to be done improving overall translation quality.

Translation systems are usually built using a large bilingual dataset of sentences in one language and their translation in another. Popular sources for these datasets are professional translations of parlamentarial proceedings, for example of the Canadian or the European Parliament (Koehn 2005). These translations are then in a first step aligned on a sentence basis. This leaves us with one-to-one sentence translations, that can then be word aligned using unsupervised machine learning techniques (Brown et al. 1993). Now we have connected the words that have co-occured most frequently and are therefore more likely to be translations of each other. This alignment however is not a one-to-one alignment, as one word in the source language may be translated into two words in the target language. This alignment can be used for word-based machine translation, but phrase-based machine translation (Koehn, Och, and Marcu 2003), combining multiple words into a phrase, results in much better translations. For syntax-based machine translation systems (Chiang and Knight 2006) we need parse trees to work with, adding another step of data preparation. Using syntactical structures in machine translation greatly improves the translation quality in certain constructions and languages that are characterized by more complex sentence structure, such as German compared to English. In order to parse the data, i.e. create some grammatical structure,

the parser uses a set of rules. These rules are induced from a monolingual dataset, that has been annotated with some grammatical structure, for example of a constituency grammar (Marcus, Marcinkiewicz, and Santorini 1993). This annotation however has to be done by experts manually, which is costly both in time and resources. In order to save both, it is possible to induce some grammatical structure using unsupervised machine learning techniques.

Unsupervised machine learning techniques need as input only the data, that is to be structured. In contrast to this unsupervised approach, supervised and semi-supervised techniques need some reference data, which is used to structure the input data, making unsupervised approaches particularly attractive when there is no or only little pre-structured data. In the context of syntax-based machine translation we hand over our bilingual corpus to the unsupervised machine learning algorithm, which in turn outputs a grammatical structure for each sentence, that is sensible in the machine translation context. This means that we do not necessarily care about the structure itself, as it is only a tool to produce a correct translation. Structuring the data using syntactical theories of different languages makes sense in a monolingual task, but they might hinder the translation, due the structures being too dissimilar. Experiments on binarization have shown, that the rejection of regular syntactic parse trees in favor of binarizing them produces better translation results (Wang, Knight, and Marcu 2007). All this leads to finding a structure that is observed in the data and suitable for translation. Whether this structure is sensible in other language processing tasks is a wholly different story that will hopefully be explored soon.

Earlier attempts at inducing grammars inductively did not bode well (Carroll and Charniak 1992), although later approaches proved to be more successful (Klein and Manning 2002). The induction scheme used by (Carroll and Charniak 1992) functions in five steps:

0. Divide the corpus into a rule extraction and a training corpus
1. Generate all rules which might be used to parse the sentences in the rule corpus, according to a dependency grammar scheme with additional constraints to reduce the search space
2. Estimate rule probabilities
3. Improve those estimates using the training corpus
4. Remove all rules with a probability less than some threshold

This approach is known as parameter search, because the search space is fixed and the parameters are shifted around using optimization methods such as expectation-maximization (EM). Due to the failures in parameter search based

approaches, grammar induction research focused more on structure search approaches. Structure search induces trees using local heuristics, that then greedily construct the grammar, which in turn defines the tree structure. This approach looks at the context of word sequences, creates new categories, i.e. nonterminals, according to these contexts and merges categories that are similar[1]. The work of (Klein and Manning 2002) shows a structure search based approach which additionally incorporates parameter search methods, producing parsing results comparable to supervised parsing.

In contrast to these monolingual induction approaches, my work focuses on bilingual induction, centered on machine translation, although its results may as well be applied to the monolingual sentence compression task (Yamangil and Shieber 2010). My thesis focuses on the works of (Cohn and Blunsom 2009) and (Zhai et al. 2013) that have been published recently. Cohn and Blunsom explore the induction of string-to-tree grammars, by sampling the word alignment using a fixed tree structure. String-to-tree grammars do not have tree structures on the input side, thereby translating from a simple string of words into a tree structure. In contrast to this (Zhai et al. 2013) builds on the ideas of (Cohn and Blunsom 2009) but instead uses a fixed word alignment, and samples a binary tree structure, naming the nodes using part-of-speech (POS) tags[2]. I will follow mostly the idea of (Zhai et al. 2013), creating new trees using a given word alignment and part-of-speech tags in order to label the nodes. However, I will use tree-to-tree translation instead of string-to-tree, which uses tree structures both on the input and output side. This implies that the binary tree structures need to be induced for both the input and the output side at the same time. The grammar formalism connecting these trees is in my case not a synchronous tree substitution grammar (STSG) (Eisner 2003), which was used in the two previous approaches, but rather a multi bottom-up tree transducer (MBOT) (Engelfriet, Lilin, and Maletti 2009). Using the MBOT formalism allows us to model discontinuities, which happen frequently in some languages such as German (see Fig. 1.1). This benefit over STSG even results in improved translation quality in the tree-to-tree translation setting (Braune et al. 2013).

In order to provide a more step-by-step approach, which enables us to see where difficulties in the induction of trees arise, I divided the experiments into three scenarios. In the first scenario, a tree structure for the output side is given, inducing binary trees only on the input side. While the second scenario works the other way round, the third scenario uses no parse trees at all, inducing binary tree structures on both the input and output side.

[1] An extensive overview on monolingual grammar induction can be found in Dan Klein's thesis (Klein 2005)

[2] POS tagging can be done unsupervised and may well be seen as a "solved" problem.

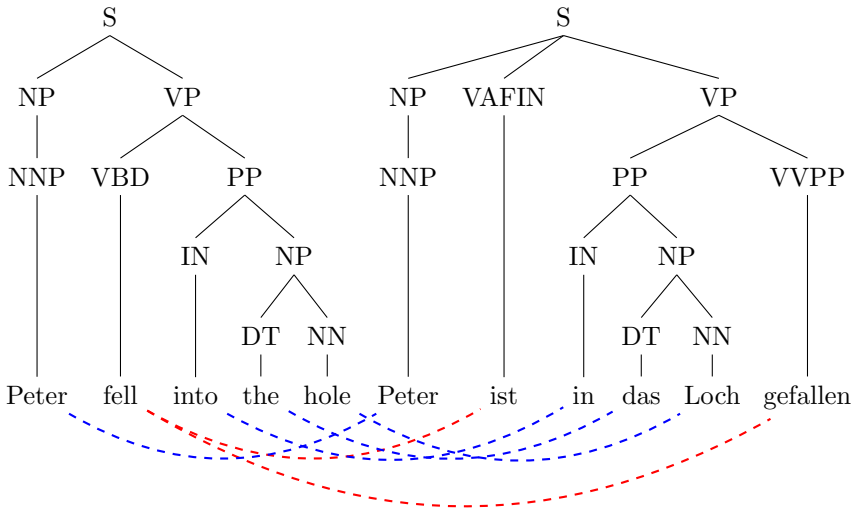

Figure 1.1.: Example of an aligned sentence pair with constituent trees. The red alignment marker, shows a one-to-many alignment relevant for discontiguous rules.

The results obtained in my experiments suffer from one major drawback. As the speed of the Gibbs sampler is very slow, the extracted trees are still mostly random. However, a quantitative analysis already shows some problems with the parameter settings and general sampling behaviour, providing suggestions for future research.

Next up is Chapter 2, serving as an extensive tutorial for the mathematical background of Dirichlet Processes, as they form the basis of the unsupervised machine learning model. Additionally our sampling technique, i.e. Gibbs sampling (Resnik and Hardisty 2010), is shortly presented. Thereinafter the tree transduction formalism used in the prior work and the more expressive formalism used in this work is established. This concludes the Preliminaries chapter. In Chapter 3 I will then finally put the pieces together to construct the model that is used for the unsupervised induction of trees. Additionally this chapter presents the structure of the trees created and the three different induction scenarios. Chapter 4 deals with the setup of the experiments that were carried out for this thesis and presents their raw outcomes. A discussion of these outcomes is given in the following Chapter 5, leading to a concluding Chapter 6, summarising what has been done and giving an outlook onto future research possibilities.

2. Preliminaries

This chapter will contain all the basic information needed to understand my thesis. The first big part of this chapter will contain an introduction to the machine learning part of this work. I will review the basic distributions I will be using, explain Bayesian Inference, provide an intuition for Markov-Chain Monte Carlo Methods in general and Gibbs Sampling in particular. Subsequently I will introduce trees and two tree transduction formalisms.

2.1. Bayesian Inference

2.1.1. Quick Probability Review

The induction of trees relies on probability models. Therefore this chapter will give a short introduction to the basic rules for computing with probabilities and probability functions.
A probability is a measure of the likeliness that an event E will occur out of a sample space of possible outcomes $e \in S$. We use a numerical measure whose value is between 0 and 1, with 0 indicating impossibility of the event and 1 indicating certainty. In order to have a well formed probability space the probabilities for each of the possibilities have to sum to 1. A probability function is therefore a mapping from the sample space S to the interval $[0, 1]$ including both 0 and 1.

$$p : S \to [0, 1] \tag{2.1}$$

with

$$\sum_{e \in S} p(e) = 1 \tag{2.2}$$

To provide an intuition for probabilities we use a simple coin. A coin usually either lands heads (H) or tails (T) (ignoring the unlikely probability of the coin landing on the rim). If the coin is fair the probabilities of landing on either side are even, i.e. 0.5. Doing one coin flip is seen as an experiment with two outcomes e: heads or tails. These outcomes form the sample space $S = \{H, T\}$ for this experiment. However, doing more than one coin flip counts as an experiment as well. Imagine an experiment where we toss the coin twice. Our outcomes in this experiment are $S = \{HH, HT, TH, TT\}$. The probabilities of each of those outcomes is, given a fair coin, $\frac{1}{4}$. Events serve as more eloquent outcomes formed from a subset of outcomes $E \subseteq S$. The event

of throwing heads at least once in two coin tosses is $E = \{HH, HT, TH\}$. The probability of the event of at least once heads is the sum of its outcomes $p(E) = \sum_{e \in E} p(e) = \frac{3}{4}$.
In order to ease notation by making formulas more general we use random variables. A random variable $X \in S$ represents the possible outcome of an experiment. For an experiment that has yet to be performed this notion is intuitive. We also use random variables for experiments that were already performed, but whose outcomes are uncertain. Two important rules of probability are the *sum rule* and the *product rule.*

$$p(X) = \sum_{Y} p(X, Y) \tag{2.3}$$

$$p(X, Y) = p(Y|X)p(X) \tag{2.4}$$

Here $p(X, Y) = p(X \cap Y)$ is the joint probability, the probability of the events X and Y occurring together, i.e. the event $X \cap Y$. The quantity $p(X|Y)$ is the conditional probability, where the probability of event X is conditioned on event Y occurring. The last term $p(X)$ is known as the marginal probability.

The random variables can take on discrete or continuous values. In the first case we use a probability mass function to denote how much probability mass is assigned to each value of the random variable. In the latter case we use probability density functions. Since they are defined over a continuous space we do not compute the probability mass at one point, but at an interval. This interval lies between a and b excluding both ends, denoted by the round bracket.

$$p(x \in (a, b)) = \int_a^b p(x)dx \tag{2.5}$$

Therefore the condition that the probabilities have to sum to 1 changes to:

$$\int_{-\infty}^{\infty} p(x)dx = 1 \tag{2.6}$$

Rule (2.3) changes to the following for x and y being real variables.

$$p(x) = \int p(x, y)dy \tag{2.7}$$

When dealing with probabilities and values we regularly use two measures, which represent the underlying probability distribution. The expected value of a distribution is the value we expect to get under normal conditions. For a function $f : Z \to \mathbb{R}$ the expected value given a discrete variable z is

$$E[f(z)] = \sum_{z \in Z} f(z)p(z) \tag{2.8}$$

with Z being the discrete set of values z can take and p being the probability distribution over possible values for z. In the continuous case the sum changes once again to an integral

$$E[f(z)] = \int f(z)p(z)dz \tag{2.9}$$

The variance measures the spread of a variable. This means that for a smaller variance the values will lie closer to the mean (expected value) and with 0 being identical to the mean. Larger values denote more spread of the variables around the expected value.

$$\begin{aligned} \mathrm{Var}[f(z)] =& E[(f(z) - E[f(z)])^2] \\ =& \sum_{z \in Z} (f(z) - E[f(z)])^2 \cdot p(z) \end{aligned} \tag{2.10}$$

The variance is defined as the squared deviation from $f(z)$ of the mean. In the continuous case we use, as always, an integral instead of a sum.

$$\mathrm{Var}[f(z)] = \int (f(z) - E[f(z)])^2 p(z)dz \tag{2.11}$$

2.1.2. Bayesian Theory

Bayesian inference is a common way to update the estimated probability of a model or hypothesis given a growing amount of data using Bayes' rule.

$$p(\theta|\boldsymbol{X}) = \frac{p(\boldsymbol{X}|\theta)p(\theta)}{p(\boldsymbol{X})} \tag{2.12}$$

The equation consists of the likelihood $p(\boldsymbol{X}|\theta)$, modelling the probability of the data $\boldsymbol{X}$ given the model θ, the prior $p(\theta)$, our belief in the model, a normalization constant $p(\boldsymbol{X})$ providing the probability of the data[1] and our updated belief in the model $p(\theta|\boldsymbol{X})$ known as the posterior. Bayes' theorem can be simply worded as

$$\text{posterior} \propto \text{likelihood} \times \text{prior}. \tag{2.13}$$

The denominator $p(\boldsymbol{X})$ is usually expressed using the numerator of the expression and integrating over all possible values of the model. The term below uses the sum and product rule introduced in (2.7) and (2.4).

$$p(\boldsymbol{X}) = \int_{\theta} p(\boldsymbol{X}|\theta)p(\theta)d\theta \tag{2.14}$$

[1] In order for the expression to be well defined, we need $p(\boldsymbol{X}) > 0$. If $p(\boldsymbol{X}) = 0$ the posterior is 0.

We however do not start out with a fixed model θ but sample this model from some probability distribution. This distribution is governed by a so called hyperparameter we that name α. The model[2] is then conditioned on this new hyperparameter changing Bayes' rule to

$$p(\theta|\boldsymbol{X}, \alpha) = \frac{p(\boldsymbol{X}|\theta)p(\theta|\alpha)}{p(\boldsymbol{X}|\alpha)} \tag{2.15}$$

and the denominator to

$$p(\boldsymbol{X}|\alpha) = \int_\theta p(\boldsymbol{X}|\theta)p(\theta|\alpha)d\theta \tag{2.16}$$

Using these formulas we can now predict a new datapoint $\tilde{x}$ given some amount of already processed data. The prediction is not, in contrast to frequentist statistics, a single point, but a probability distribution over points. This new distribution is called *posterior predictive distribution*[3].

$$p(\tilde{x}|\boldsymbol{X}, \alpha) = \int_\theta p(\tilde{x}|\theta)p(\theta|\boldsymbol{X}, \alpha)d\theta \tag{2.17}$$

In order to ease up the level of mathematics when using Bayesian inference we use so called conjugate priors. The posterior distribution $p(\theta|\boldsymbol{X})$ and the prior $p(\theta)$ are conjugate when they are in the same family of distributions. The prior is then called the conjugate prior for the likelihood function $p(\boldsymbol{X}|\theta)$. Using a conjugate prior for our likelihood function allows us to express the posterior in a closed-form expression, sidestepping numerical integration. An example will be given after the introduction of the distributions commonly used in Bayesian inference.

2.1.3. Common Distributions

Next we recall and illustrate the distributions used in our approach.

Bernoulli & Categorical Distribution

The Bernoulli distribution is probably the easiest distribution used in this paper. It simply denotes the distribution of the probability outcomes for a binary experiment, such as a coin toss. The random variable in this case can therefore only take on two values, i.e. 0 and 1. The probability of the random variable being $X = 1$ is denoted by p_1 and the probability of being $X = 0$ by p_0.

$$p_1 = p(X = 1) = 1 - p(X = 0) = 1 - p_0 \tag{2.18}$$

[2]The joint probability $p(\boldsymbol{X}, \theta, \alpha)$ is defined as follows: $p(\boldsymbol{X}, \theta, \alpha) = p(\boldsymbol{X}|\theta)p(\theta|\alpha)p(\alpha)$

[3]Once again we use a particular joint probability: $p(\tilde{x}, \theta, \boldsymbol{X}, \alpha) = p(\tilde{x}|\theta)p(\theta|\boldsymbol{X}, \alpha)p(\boldsymbol{X})p(\alpha)$

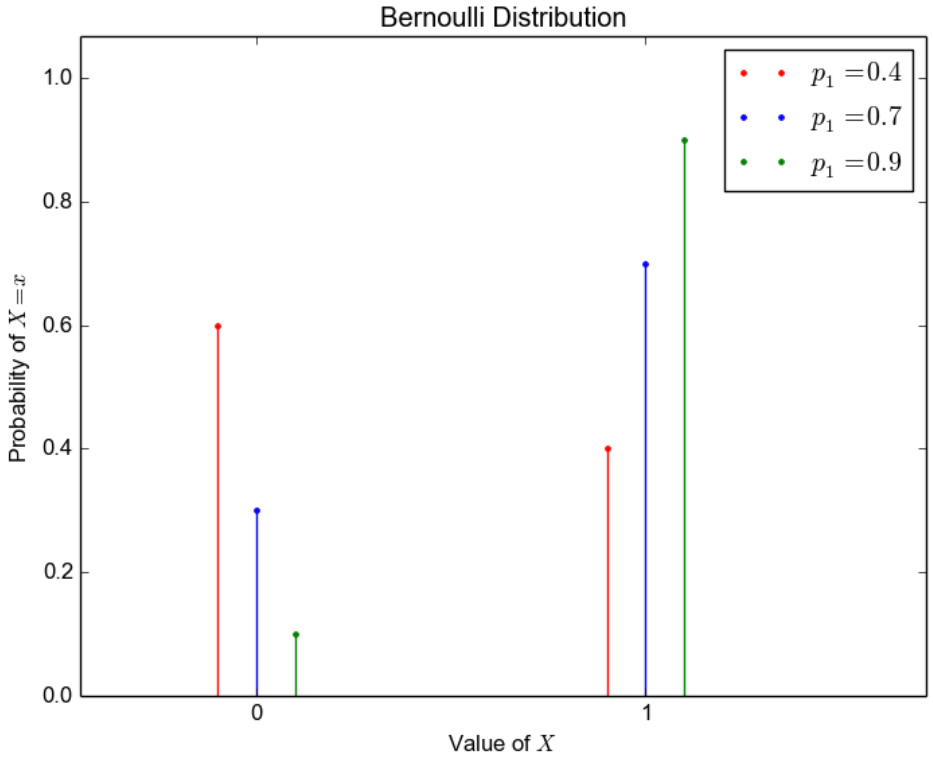

Figure 2.1.: Plot for three different Bernoulli Distributions.

The probability mass function describes how much probability mass is located at which random variable's value. So for our coin example we have to distribute the probability mass of the random variable X to $X = 1$ and $X = 0$.
More formally we have the following function:

$$p(X; p_1) = \begin{cases} p_1 & \text{if } X = 1 \\ 1 - p_1 & \text{if } X = 0 \end{cases} \tag{2.19}$$

Since flipping a coin is too simple for a professional gambler we replace the coin with a regular k-sided die. Rolling this die has k possible outcomes, so our random variable X can take on values $1, 2, \ldots, k$. The probabilities of these outcomes naturally have to sum to 1 and lie in the interval $[0, 1]$. The distribution for rolling such a die is called a *categorical* distribution and is the natural extension of the Bernoulli distribution, being able to handle more than just two results for the random variable.
Its probability mass function is most intuitively defined as follows:

$$p(X = i|\boldsymbol{p}) = p_i \tag{2.20}$$

The variable p_i represents an entry in the vector $\boldsymbol{p}$ and it is the probability of the random variable X being i, for every i in the value space of X.

Binomial & Multinomial Distribution

Now we take a step back to gambling with our coin. Imagine throwing the coin n times. If we now wanted to know how probable landing heads k times would be, we could either sum up the probability of each possible combination or use the binomial distribution.
Since the latter is easier we define the probability mass function of the binomial distribution as follows:

$$\text{Binom}(k|n, p_1) = \binom{n}{k} p_1^k (1-p_1)^{n-k} \tag{2.21}$$

The first term of this equation is the number of ways of choosing k objects out of n identical objects in total. The second and third terms correspond to the product of probabilities of landing heads and tails respective to k and n.

Just as before when I introduced the categorical distribution for modelling a die throw, I can extend the binomial to a multinomial distribution. The multinomial distribution models the probability of throwing each side of the die some number of times out of n total throws. Let $\boldsymbol{x} = (x_1, x_2, \ldots, x_k)$ be the vector counting the times each side has been rolled, with $\sum_{i=1}^{k} x_i = n$. The multinomial then takes a form similar to the binomial

$$\text{Mult}(\boldsymbol{x}|\boldsymbol{p}, n) = \binom{n}{x_1 x_2 \cdots x_k} \prod_{i=1}^{k} p_i^{x_i} \ , \tag{2.22}$$

where similar as before $\boldsymbol{p}$ denotes the vector of probabilities for each value i the random variable can take. The normalization coefficient, i.e. the first term, is the number of ways of partitioning n objects into k groups of size $x_1, \ldots, x_k$, which is defined as

$$\binom{n}{x_1 x_2 \cdots x_k} = \frac{n!}{x_1! x_2! \cdots x_k!}. \tag{2.23}$$

Beta & Dirichlet Distribution

Consider that we have some or no knowledge about the coin we are playing with, in the sense that we do not know whether it is fair or not. The Beta distribution helps us to model our belief in the fairness of the coin. It is defined on the interval $[0, 1]$ parametrized by two positive parameters α and β. The interval on which the Beta distribution is defined, is the probability that the coin lands heads and the parameters shift how much we believe in these probabilities. It can therefore be seen as a probability distribution over

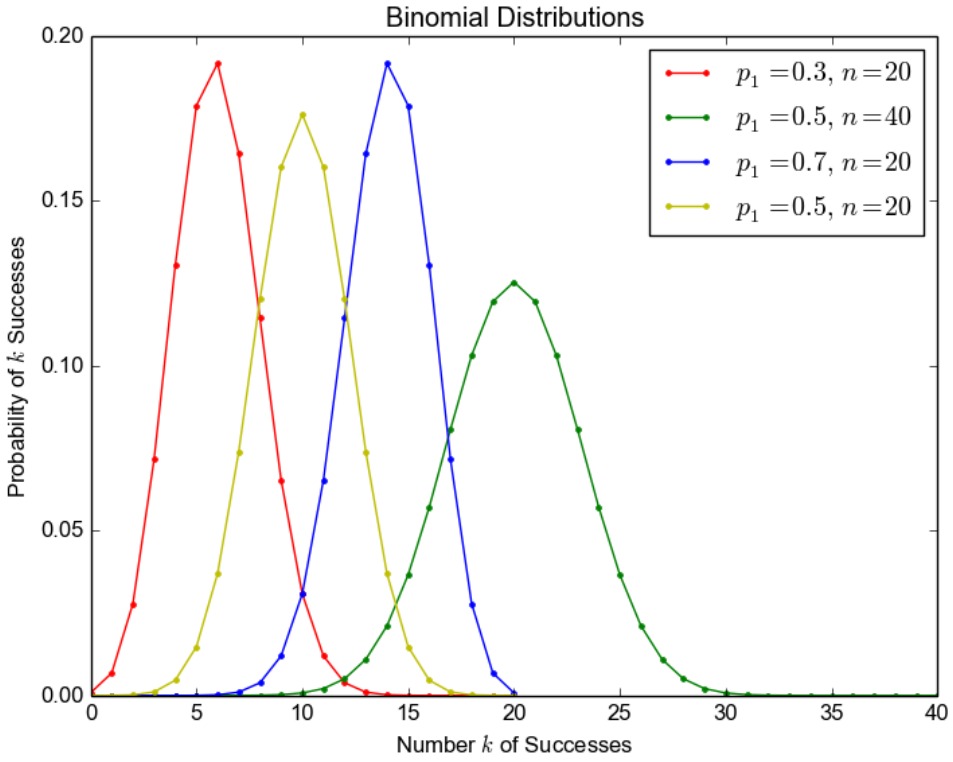

Figure 2.2.: Plot of four different binomial distributions. Note the spikier shape for the yellow curve compared to the green curve, due to the same probability mass distributed onto fewer points. Furthermore note the shift to left/right for the red/blue curve compared with the yellow curve, due to smaller/greater success probability p.

probability distributions. The probability density function is defined using the gamma function $\Gamma(x)$ which is defined for positive integers as

$$\Gamma(x) = (x-1)! \tag{2.24}$$

.

The probability density function then is

$$\text{Beta}(x;\alpha,\beta) = \frac{x^{\alpha-1}(1-x)^{\beta-1}}{\int_0^1 y^{\alpha-1}(1-y)^{\beta-1}dy} \tag{2.25}$$

$$= \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} x^{\alpha-1}(1-x)^{\beta-1} \tag{2.26}$$

The integral in the denominator of this equation serves as a normalization constant which is handily defined using the Gamma function.

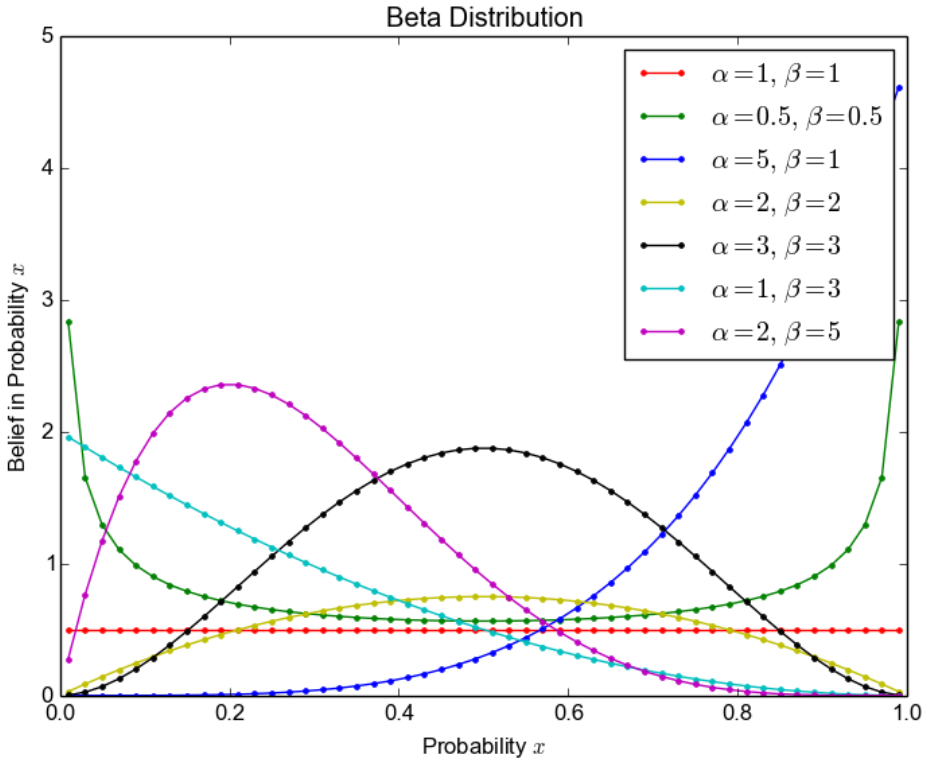

Figure 2.3.: Plot of some characterizing beta distributions.

An intuition to interpret the parameters α and β is to view those as previous successes and failures respectively. Setting both to 1 results in ignoring the input x by setting the exponents of the terms containing x to zero. This results in a uniform density function, making no assumptions about the coin. All equal values greater than one for the parameters result in a symmetric bell curve centered at 0.5 which grows sharper with growing values for α and β, since we have seen more evidence that the coin is indeed fair. Increasing either α or β results in a shift of the curve towards 1 or 0 respectively.

The Dirichlet distribution is the multivariate extension of the Beta distribution. Instead of two parameters for successes and failures we now have the parameters $\alpha_1, \alpha_2, \ldots, \alpha_k$ for the k sides of our die. The probability density function is therefore

$$\mathrm{Dir}(x_1, \ldots, x_k; \alpha_1, \ldots, \alpha_k) = \frac{\Gamma(\sum_{i=1}^{k} \alpha_i)}{\prod_{i=1}^{k} \Gamma(\alpha_i)} \prod_{i=1}^{k} x_i^{\alpha_i - 1} \tag{2.27}$$

with

$$x_1, \ldots, x_k \in [0,1] \tag{2.28}$$
$$\sum_{i=1}^{k} x_i = 1 \ .$$

The parameters $\alpha_1, \ldots, \alpha_k$ behave similarly in shaping the Dirichlet functions as α and β for the Beta distribution. High values lessen the variance and preferences for some sides of the die are expressed using higher values for its parameters. Expressing no knowledge about the die and setting the distribution uniformly is again achieved for the vector $(\alpha_1, \ldots, \alpha_n) = (1, \ldots, 1)$.

Coin Example

In order to clarify what has been introduced we combine Bayes' theory with the probability distributions to produce an example showing how this can be applied. In order to keep the formulas small we are going to use our previous coin example.

We are going to model the belief in the fairness of our coin before and after we have done some experiments. Our belief in the fairness of the coin is modeled using the Beta distribution. Setting the parameters both to 2 expresses, that we believe the coin should be fair, although we do not trust this too much therefore leaving room for variance. The probability that we will see a certain kind of result in our experiment, i.e. a certain number of heads and tails, is modeled using the binomial distribution. The posterior, our updated belief in the coin, will be a Beta distribution as well, since the Beta distribution serves as a conjugate prior for our likelihood modeled by the binomial distribution. For our equation below we use n as the number of trials, k as the number of successes, x as the probability of heads and α and β as hyperparameters, θ as model, X as data as before.

$$p(\theta|X) = \frac{p(X|\theta)p(\theta)}{\int_\theta p(X|\theta)p(\theta)d\theta} \tag{2.29}$$
$$= \frac{\binom{n}{k}x^k(1-x)^{n-k} \times \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}x^{\alpha-1}(1-x)^{\beta-1}}{\int_{y=0}^{1}\binom{n}{k}y^k(1-y)^{n-k} \times \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}y^{\alpha-1}(1-y)^{\beta-1}dy} \tag{2.30}$$
$$= \frac{x^{k+\alpha-1}(1-x)^{n-k+\beta-1}}{\int_{y=0}^{1} y^{k+\alpha-1}(1-y)^{n-k+\beta-1}dy} \tag{2.31}$$
$$= \frac{\Gamma(\alpha+\beta+k+n-k)}{\Gamma(\alpha+k)\Gamma(\beta+n-k)}x^{k+\alpha-1}(1-x)^{n-k+\beta-1} \tag{2.32}$$
$$= \text{Beta}(x; \alpha+k, \beta+n-k) \tag{2.33}$$

From the second to third step we pull the constant terms out of the integral and cancel them out of the whole fraction. Due to the same bases we can simplify the rest of the equation by adding the exponents. The rest of the formula follows according to (2.26).
This equation is commonly known as the Beta-Binomial distribution. Similarly to this one there also exists the Dirichlet-Multinomial distribution. With the Beta-binomial distribution we can now compute the probability of a fair coin, given a prior belief and some evidence.
Updating our belief after seeing new evidence is also possible using this equation. In this case we compute the posterior as our updated belief and use it as prior belief in the next iteration of our belief update.

Geometric Distribution

Given a Bernoulli distribution we can use the Geometric distribution to model the number of trials needed to get one success. Given a fair coin the probability to throw heads in one trial is 0.5. Whereas in two or more trials the probability is the product of the probabilities of failure times the probability of success, which is in this case $0.25, 0.125$ and so on. It is formalized using only the success probability p_1:

$$\text{Geom}(x; p_1) = (1 - p_1)^{x-1} p_1 \tag{2.34}$$

Poisson Distribution

The Poisson distribution is used to model the number of events occurring in a fixed interval of time or space. These events occur with a known average rate and are independent of previous events. This distribution can for example be used to model the number of phonecalls received per day, assuming that the phonecalls are independent of each other. The average number of events serves both as the expectation and the variance of the Poisson distribution, i.e. someone receiving a lot of phonecalls on average will have a much higher fluctuation than someone receiving only a few calls on average.
The Poisson distribution takes as its sole parameter this average value denoted by λ. Its formula uses Euler's number ($e = 2.718\ldots$) and reads as follows:

$$\text{Pois}(x; \lambda) = \frac{\lambda^x e^{-\lambda}}{x!} \tag{2.35}$$

2.1.4. Dirichlet & Pitman-Yor Processes

Using Dirichlet distributions we can sample probability distributions for a fixed number of possible outcomes. However we may not always know how many outcomes there are. This is the case when we are randomly sampling

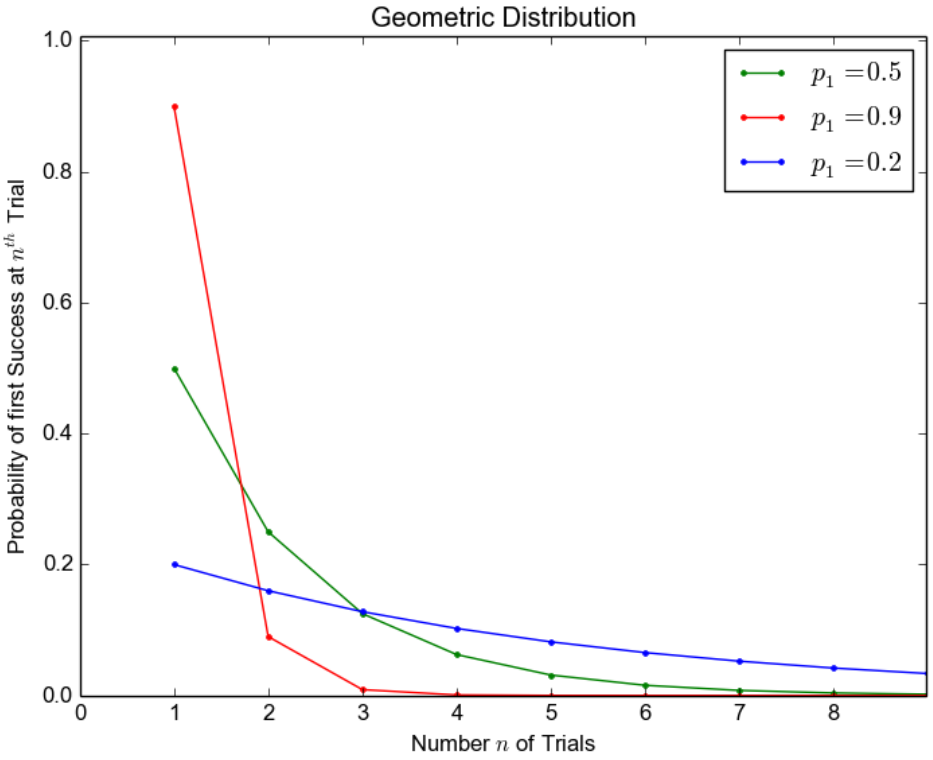

Figure 2.4.: Plot of three Geometric distributions. Having a high success probability results in a sharply falling curve, because we have such a high probability of succeeding very early. In contrast the curve barely falls for small values of p_1, as the chance of success is nearly equally low for the subsequent steps.

tree fragments. While it may be possible to list all valid tree fragments for a tree substitution grammar (TSG), it is easier to assume an infinite amount of fragments and sample using a *Dirichlet Process.*

A Dirichlet process (DP) is a way of assigning a probability distribution over probability distributions. The domain of a Dirichlet process is a set of probability distributions, while the value of the process is a probability distribution. The distributions are discrete and infinite dimensional. Although specified over infinite dimensional distributions, the Dirichlet process uses only finite resources. The following introduction is based mostly on (Teh 2010) and uses additional insight from (Phadia 2013), especially when introducing Pitman-Yor Processes.

We define the base distribution of the Dirichlet process as H, a distribution over some probability space Θ. The parameter α is a positive real number

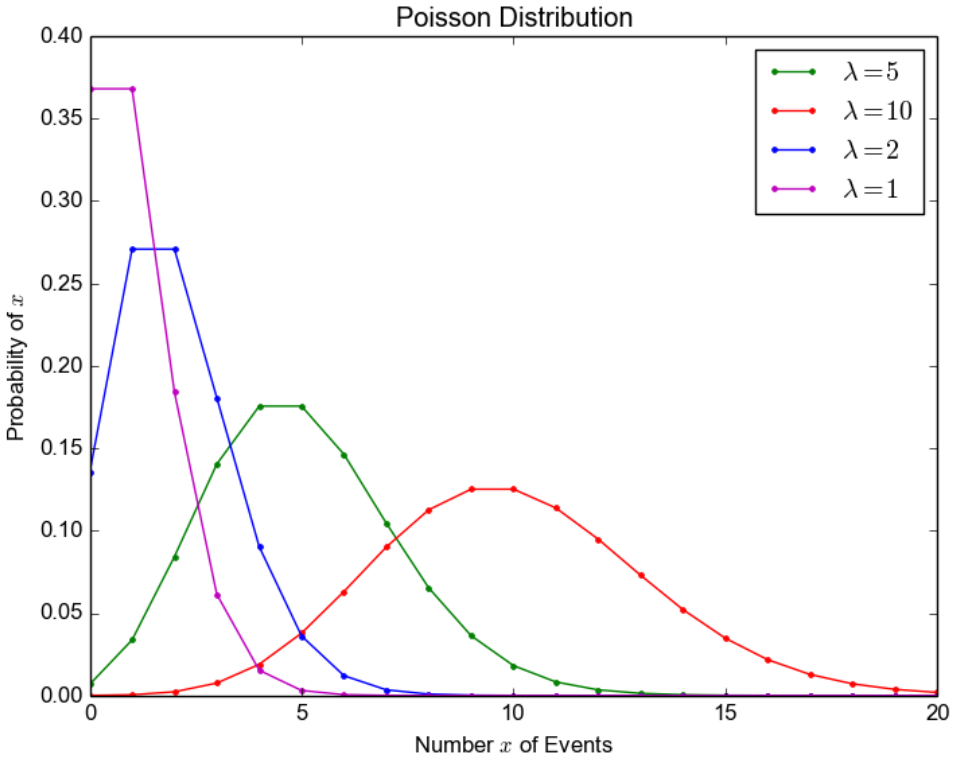

Figure 2.5.: Plot of four Poisson distributions, showing its possible shapes controlled by the expectation and variance parameter λ.

and G is a random distribution over Θ. For any finite measurable partition[4] $T_1, \ldots, T_r$ of Θ the vector $(G(T_1), \ldots, G(T_r))$ is random, since the distribution G is random. This distribution G is Dirichlet process distributed with base distribution H and concentration parameter α, written $G \sim \mathrm{DP}(\alpha, H)$, if

$$(G(T_1), \ldots, G(T_r)) \sim \mathrm{Dir}(\alpha H(T_1), \ldots, \alpha H(T_r)) \tag{2.36}$$

for every finite measurable partition $T_1, \ldots, T_r$ of the probability space Θ. The two parameters can be interpreted intuitively. The base distribution is for $T \subset \Theta$ the mean of the Dirichlet process $E[G(T)] = H(T)$. The concentration parameter α serves as an inverse variance with $V[G(T)] = H(T)(1 - H(T))/(\alpha + 1)$. With larger values for α more probability mass will be centered around the base distribution. Therefore with $\alpha \to \infty$ we will have $G(T) \to H(T)$ for any measurable set $T \subset \Theta$. This does however not lead to $G \to H$ since the draw G from the Dirichlet process is discrete whereas

[4]In order to skip measure theory to properly explain measurable partitions we suggest the reader to use the intuition of seeing Θ as the sample space and a partition on said space as an event.

the base distribution may be continuous.

Dirichlet Process Posterior

As before we take $G \sim \mathrm{DP}(\alpha, H)$ as a draw from a Dirichlet process. Since G is itself a distribution we can perform a sequence of independent draws $\theta_1, \ldots, \theta_n \sim G$. These draws θ_i take values from the probability space Θ since G is a distribution over said space. Similarly as with the coin example (see Section 2.1.3), we are now going to define the posterior distribution of G given some observed values $\theta_1, \ldots, \theta_n$. Again $T_1, \ldots, T_r$ is a finite measurable partition of Θ. The number of observed values in T_k is denoted by $n_k = \#\{i : \theta_i \in T_k\}$. Due to (2.36) and the conjugacy between the multinomial (here the sequence of draws from G) and the Dirichlet distribution, we have

$$(G(T_1), \ldots, G(T_r))|\theta_1, \ldots, \theta_n \sim \mathrm{Dir}(\alpha H(T_1) + n_1, \ldots, \alpha H(T_r) + n_r) \quad (2.37)$$

As (2.37) is true for all finite measurable partitions, the posterior distribution over G has to be a Dirichlet process as well. The updated concentration parameter then is $\alpha + n$ and the updated base distribution changes to $\frac{\alpha H + \sum_{i=1}^{n} \delta_{\theta_i}(\theta)}{\alpha + n}$, with the delta function $\delta_x(T)$.

$$\delta_x(T) = \begin{cases} 0, & x \notin T; \\ 1, & x \in T. \end{cases} \quad (2.38)$$

Intuitively, this function serves as an indicator, whether an element is contained in a set.[5] The number of observed values is equally given by $n_k = \sum_{i=1}^{n} \delta_{\theta_i}(T_k)$. The rewritten posterior Dirichlet process is:

$$G|\theta_1, \ldots, \theta_n \sim \mathrm{DP}(\alpha + n, \frac{\alpha}{\alpha + n} H + \frac{n}{\alpha + n} \frac{\sum_{i=1}^{n} \delta_{\theta_i}(\theta)}{n}) \quad (2.39)$$

The base distribution of the posterior is a weighted average between the prior base distribution H and the empirical distribution $\frac{\sum_{i=1}^{n} \delta_{\theta_i}(\theta)}{n}$. While the weight of the prior base distribution is proportional to the concentration parameter α, the weight associated with the empirical distribution is proportional to the number of observations n. Taking $\alpha \to 0$ would lead to ignoring the prior distribution only taking the empirical distribution into account. When the number of observations is large enough, the posterior is dominated by the empirical distribution. This means that we approach the true distribution with growing evidence.
Having observed $\theta_1, \ldots, \theta_n$ values we can now do a prediction for θ_{n+1}. This

[5] Note that this holds true for singleton sets as well such as $\{\theta\}$. We will however not explicitly write singleton sets as sets but leave them as is.

new value is conditioned on the previous observations and with G marginalized out as in equation (2.17) we have

$$p(\theta_{n+1} \in T|\theta_1, \ldots, \theta_n) = E[G(T)|\theta_1, \ldots, \theta_n] \tag{2.40}$$

$$= \frac{1}{\alpha + n}(\alpha H(T) + \sum_{i=1}^{n} \delta_{\theta_i}(T)) \tag{2.41}$$

The new sample is then drawn from the following distribution

$$\theta_{n+1}|\theta_1, \ldots, \theta_n \sim \frac{1}{\alpha + n}(\alpha H + \sum_{i=1}^{n} \delta_{\theta_i}(\theta)) \tag{2.42}$$

which is the same as the posterior base distribution given previous observations $\theta_1, \ldots, \theta_n$.

Urn Scheme

This can be visualized using an urn scheme. We start with an urn filled with α black balls. Each time we draw a black ball from this urn we choose a colour by drawing it from base distribution H which is defined over the colour space Θ. We then proceed by adding a ball of the newly drawn color into the urn and returning the black ball. Each time we draw a coloured ball from the urn we add another ball of its colour. So after n draws we have added n balls to the urn leaving us with a total of $\alpha + n$ balls. Drawing a black ball then has the probability $\frac{\alpha}{\alpha+n}$ and drawing a coloured ball has probability $\frac{n}{\alpha+n}$. The former case corresponds to $\theta_{n+1} \sim H$, drawing from the base distribution, and the latter to drawing from the empirical distribution $\frac{\sum_{i=1}^{n} \delta_{\theta_i}(\theta)}{n}$.
After our series of n draws we end up with K different colours in the urn. These values are represented by $\theta_1^*, \ldots, \theta_K^*$ with $K \leq n$. We have thus defined a partition on $\theta_1, \ldots, \theta_n$ into K clusters, with $\theta_i = \theta_k^*$ if i is in cluster k. The predictive distribution can then be written as with n_k denoting the number of repeats of θ_k^*:

$$\theta_{n+1}|\theta_1, \ldots, \theta_n \sim \frac{1}{\alpha + n}(\alpha H + \sum_{k=1}^{K} n_k \delta_{\theta_k^*}(\theta)) \tag{2.43}$$

The value θ_k^* will be repeated by the newly drawn value θ_n with a probability proportional to the number of times it already has been observed. So larger values of n_k will result in even more growth of said value, known as a rich-gets-richer phenomenon.

Chinese Restaurant Process

The Chinese restaurant process is a metaphor which describes the distribution over such partitions. In the Chinese restaurant process customers enter a Chinese restaurant with an infinitely large number of tables and either sit, with a probability proportional to the number of customers at that table, at an occupied table or choose to open a new table with a probability proportional to the concentration parameter. This construction process is inverse to the urn scheme where we would first draw values inducing a clustering, whereas the Chinese restaurant process first produces a clustering and subsequently draws the values for each cluster $\theta_k^* \sim H$. The number of clusters grows logarithmically in the number of observations.

$$E[K|n] = \sum_{i=1}^{n} \frac{\alpha}{\alpha + i - 1} \in \mathcal{O}(\alpha \log n) \tag{2.44}$$

Each new observation takes on a new value θ_i with probability $\frac{\alpha}{\alpha+i-1}$. This equation directly shows the effect of the concentration parameter and as well reflects the rich-gets-richer phenomenon as we assume a small number of large clusters to be created.

Stick-Breaking Process

Another construction mechanism used to create Dirichlet processes is the stick-breaking construction.

$$\beta_k \sim \text{Beta}(1, \alpha) \tag{2.45}$$

$$\theta_k^* \sim H \tag{2.46}$$

$$\pi_k = \beta_k \prod_{l=1}^{k-1} (1 - \beta_l) \tag{2.47}$$

$$G = \sum_{k=1}^{\infty} \pi_k \delta_{\theta_k^*}(\theta) \tag{2.48}$$

The construction of π is best explained metaphorically. We start with a stick of length 1, breaking it at β_1 and assign π_1 to be the length of the stick we just broke off. In order to obtain π_2, π_3 etc. we proceed recursively, breaking off the other portion of the stick. Each portion of the stick is assigned a value θ^* via the base distribution H. The sizes of the broken off portions of the stick are controlled by the α. Higher values of this parameter result in smaller values for the β_i thus producing smaller portions to be broken off. The probability mass function of the desired output function is then defined as the infinite sum over the π_k.

Pitman-Yor Process

Pitman-Yor processes extend the regular Dirichlet processes by adding a third parameter, namely the discount parameter. In contrast to the Dirichlet process the Pitman-Yor process can be used to model power-law behaviour. These distributions are fairly frequent in natural language processing and can be found in e.g. word distributions, part-of-speech tag distributions and are known as Zipf's law. This extension to the exponential distributions created by Dirichlet processes might come in handy, when creating synchronous tree fragments.
The construction of Pitman-Yor processes can be done using the stick-breaking construction. The discount parameter d is constrained such that $0 \leq d < 1$ and $\alpha > -d$.

$$\beta_k \sim \text{Beta}(1 - d, \alpha + kd) \tag{2.49}$$

Setting the discount parameter to 0 will return the regular Dirichlet process. If the concentration parameter is set to 0 we end up with a random probability whose weights are based on a stable law with index $0 < d < 1$.

The urn scheme can as well be modified to return a Pitman-Yor process. Similar to equation (2.43) we have:

$$\theta_{n+1}|\theta_1, \ldots, \theta_n \sim \frac{1}{\alpha + n}((\alpha + Kd)H + \sum_{k=1}^{K}(n_k - d)\delta_{\theta_k^*}(\theta)) \tag{2.50}$$

These two modifications to the equations used in the Dirichlet process construction schemes show that the Pitman-Yor process uses additional information contained in the number of clusters K and the number of previous observations n.

2.1.5. Gibbs Sampling

Some of the equations used in Bayesian inference use integrals. While they may be relatively easy to solve in smaller dimensional spaces, they become nearly unsolvable in high-dimensional spaces. In order to solve the integrals we use a technique known as Gibbs sampling (S. Geman and D. Geman 1984)[6]. The integral for the expected value (2.9) can be solved by drawing values $z^{(t)}$ according to the probability distribution $p(z)$, summing them and normalizing them.

$$E[f(z)] = \lim_{N \to \infty} \frac{1}{N} \sum_{t=1}^{N} f(z^{(t)}) \tag{2.51}$$

[6] An introduction to Gibbs sampling can be found in (Resnik and Hardisty 2010).

Instead of drawing an infinite number of values we can simplify the process via drawing only T values, which gives us an approximation of the expected value.

$$E[f(z)] \approx \frac{1}{T} \sum_{t=1}^{T} f(z^{(t)}) \tag{2.52}$$

The "weighting" of $f(z)$ which was done via the probability distribution $p(z)$ survives invisibly in the above equation. We draw values according to $p(z)$, meaning that we end up with more values z which belong to the higher probability regions and fewer values which belong to lower probability regions.

The same procedure works for computing an expected value for the posterior predictive function (2.17) which uses an integral over all possible models. Instead of computing the integral we draw models $\theta^{(t)}$ from $p(\theta|\boldsymbol{X}, \alpha)$ and compute the following sum

$$E[p(\tilde{x}|\boldsymbol{X}, \alpha)] \approx \frac{1}{T} \sum_{t=1}^{T} p(\tilde{x}|\theta^{(t)}) \tag{2.53}$$

The question that remains is how we are going to draw samples from $p(z)$. There are a wide variety of sampling methods, such as rejection sampling, importance sampling, Metropolis sampling and many more. We are going to use Gibbs sampling which is a Markov Chain Monte Carlo (MCMC) method, a class of methods using similar methods. Imagine walking around the value space Z from $z^{(0)}$ to $z^{(1)}$ to $z^{(2)}$ and so forth. The likelihood of visiting any point z should be proportional to $p(z)$. The next point we are going to visit is chosen via some function which makes probabilistic choices according to a transition probability $P_{\text{trans}}(z^{(t+1)}|z^{(0)}, z^{(1)}, \ldots, z^{(t)})$. The *Monte Carlo* part of MCMC refers to this probabilistic choice reminding us of gambling. In the *Markov Chain* model the transition probability only depends on the previous state, which is known as the Markov property.

As stated before Gibbs sampling is an MCMC method which is applicable if values z have more than one dimension. In Gibbs sampling we do not pick the next state at once but pick it dimension after dimension. This can easily be rewritten as a simple algorithm.

Algorithm 1 Gibbs Sampling

1: $z^{(0)} \leftarrow \langle z_1^{(0)}, \ldots, z_k^{(0)} \rangle$
2: **for** $t = 0$ to $T - 1$ **do**
3: **for** $i = 1$ to k **do**
4: $z_i^{(t+1)} \sim P(Z_i | z_1^{(t+1)}, \ldots, z_{i-1}^{(t+1)}, z_{i+1}^{(t)}, \ldots, z_k^{(t)})$
5: **end for**
6: **end for**

Using this algorithm we can now substitute values z_i with values for different parameters of our model, providing us a way to sample a new model stepwise. For our posterior predictive function this would mean that we sample the parameters of the model θ, sample a new value $\tilde{x}$, and repeat.

2.2. Trees and transducers

The syntactic structure used in syntax-based statistical machine translation is constructed using trees, which are also frequently used in computer science. This chapter will serve as an introduction to trees in general, provide information about binary trees in particular and introduce two tree transducer formalisms that are used to translate one tree to another tree. I follow the style of (Braune et al. 2013) in the formalization of trees and transducers.

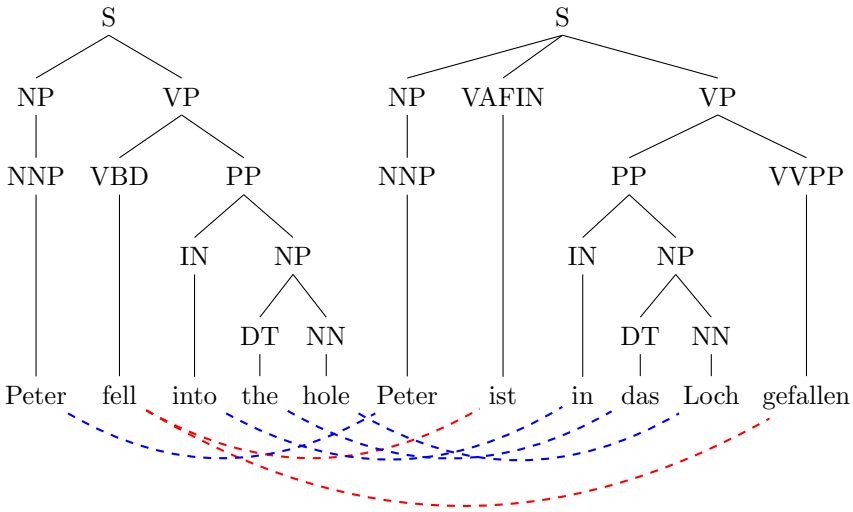

Figure 2.6.: Example of an aligned sentence pair with constituent trees. The red alignment marker, shows a one-to-many alignment relevant for discontiguous rules.

Trees used in computer science are directed acyclic graphs, where all but one node have exactly one parent node and $k \geq 0$ child nodes. The special node called root node has no parents. While we label nodes, edges remain unlabeled, but ordered. Thus we always talk about node labels, when we refer to the labels of a tree. The trees we draw are directed from top to bottom, making it possible to omit the direction of the edges. The root of the tree is always at the top and its edges lead, via more nodes and edges, to the leaves of a tree, which are the childless nodes.
We start off with an alphabet Σ of labels. The set T_Σ denotes the set of all

Σ-trees, i.e. trees that use elements of the alphabet Σ as their labels. It is the smallest set T such that $\sigma(t_1, \ldots, t_k) \in T$ for all $\sigma \in \Sigma$, $k \in \mathbb{N}^0$ and $t_1, \ldots, t_k \in T$.[7] A tree t consists of a labeled root node σ and a sequence of children $t_1, \ldots, t_k$, that are trees as well.
In order to address certain nodes inside the tree, we use their position. The positions of a tree are sequences of positive integers iw with $i \in \mathbb{N}$ where i addresses the i^{th} child of its root and w the position in this subtree. The root itself is addressed with the position ϵ, i.e. the empty word. We define the positions $\text{pos}(t) \subseteq \mathbb{N}^*$ of a tree $t = \sigma(t_1, \ldots, t_k)$ inductively.

$$\text{pos}(t) = \{\epsilon\} \cup \text{pos}^{(k)}(t_1, \ldots, t_k)$$

where

$$\text{pos}^{(k)}(t_1, \ldots, t_k) = \bigcup_{1 \leq i \leq k} \{iw \mid w \in \text{pos}(t_i)\} \tag{2.54}$$

The label of a tree $t \in T_\Sigma$ at position w is $t(w)$. The subtree rooted at said position is $t|_w$. The node $w \in \text{pos}(t)$ is a leaf in tree t if $w1 \notin \text{pos}(t)$. The leaves whose labels are from a certain subset $N \subseteq \Sigma$ are

$$\text{leaf}_N(t) = \{w \in \text{pos}(t) \mid t(w) \in N, w \text{ leaf in } t\} \tag{2.55}$$

We will refer to this set later on as leaf nonterminals when N is the set of nonterminals. The extension of this set to sequences $t_1, \ldots, t_k \in T_\Sigma$ is

$$\text{leaf}_N^{(k)}(t_1, \ldots, t_k) = \bigcup_{1 \leq i \leq k} \{iw \mid w \in \text{leaf}_N(t_i)\}. \tag{2.56}$$

Given the pairwise prefix-incomparable positions $w_1, \ldots, w_n \in \text{pos}(t)$ and $t_1, \ldots, t_n \in T_\Sigma$, we use $t[w_i \leftarrow t_i]_{1 \leq i \leq n}$ to denote the tree that is obtained from t by replacing in parallel the subtrees at w_i by t_i for every $1 \leq i \leq n$.

Binary Trees

Binary trees are routinely used in computer science, e.g. as data structures, and binarization is applied to nondeterministic devices (such as context-free grammars (CFG)) to achieve greater efficiency. The CYK parsing algorithm uses a grammar binarization approach, transforming its input CFG into Chomsky normal form (Hopcroft, Motwani, and Ullman 2003). In the context of statistical machine translation (SMT) the binarization approach is used as well providing not only better translation results (Wang, Knight, and Marcu 2007), but also better decoding efficiency (Zhang et al. 2006). Since binary trees are not inferior to regular trees and additionally offer greater efficiency,

[7] We use $\mathbb{N}^0$ to denote the set of natural numbers containing zero in contrast to the set of natural numbers $\mathbb{N}$ excluding zero.

the tree induction I present in this thesis produces binary trees. In the induction context we want to restrict the tree structure in order to restrict the search space of sensible trees. In contrast to general trees, binary trees always have two or no child nodes. This property allows us to compute the number of possible binary tree structures, given a sequence of leaf nodes of length $n+1$, using the Catalan number:

$$C_n = \frac{1}{n+1}\binom{2n}{n} \tag{2.57}$$

The binary trees used in this thesis have preterminal nodes, that have each one child node. This however does not change the number of possible trees given a sequence of leaves and their preterminals, simply adding an intermediate layer.

2.2.1. Synchronous Tree Substitution Grammar

The synchronous tree substitution grammar (STSG) (Eisner 2003) [8] is the most common formalism in syntax-based statistical machine translation. Its rules are applied to an input tree resulting in a set of possible output trees, that are usually weighted.
We use Σ and Δ as the alphabets of input and output symbols, respectively, and the set of nonterminals $N \subseteq \Sigma \cup \Delta$.

The set of rules will be denoted by R with $\rho = l \rightarrow_\psi r \in R$. Here $l \in T_\Sigma$ will denote a tree fragment on the input side and $r \in T_\Delta$ a tree fragment on the output side. In order to ensure a connection, we use the alignment function $\psi : \text{leaf}_N(l) \rightarrow \text{leaf}_N(r)$. This mapping connects each leaf nonterminal on the left-hand side with its counterpart on the right-hand side. Due to this feature, the alignment function is required to be bijective. This results in a linear and nondeleting behaviour, i.e. subtrees are neither copied nor dropped.
Additionally we use a mapping $c : R \rightarrow \mathbb{R}$ that will determine the weight of each rule. A rule ρ therefore consists of a left-hand side l, a right-hand side r, an alignment ψ and a rule weight $c(\rho)$. The successful application of a sequence of rules to an input tree is called a derivation. We work with the intuition that we begin constructing a tree pair with some elementary tree pair and rewrite each aligned leaf nonterminal using some set of rules. We therefore need a pre-translation $\langle t, \psi, c(\rho), u \rangle$, that represents the tree that has already been created in an incomplete derivation.

[8] (Chiang and Knight 2006) offers a gentle introduction to synchronous grammars in general.

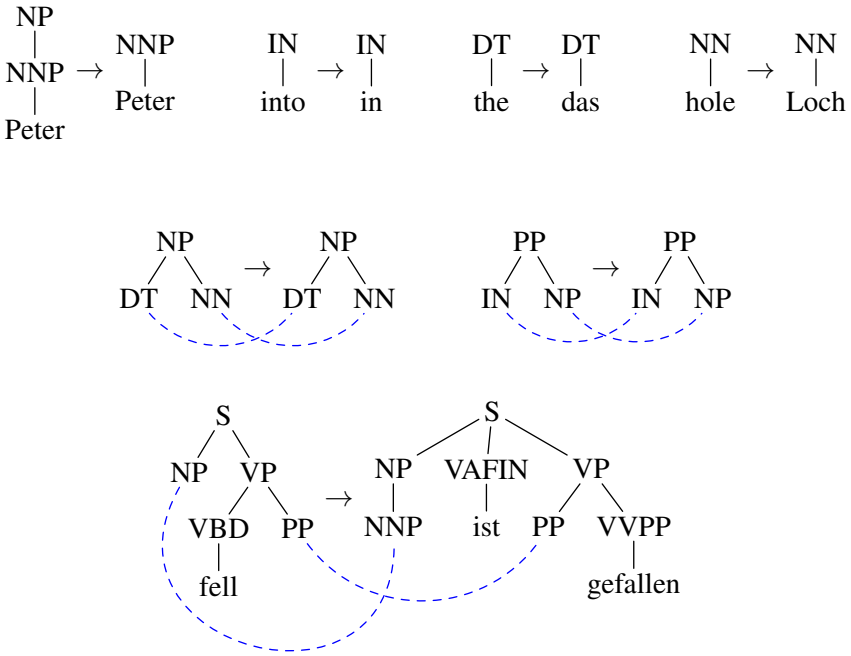

Figure 2.7.: Example of the extracted STSG rules for the tree pair in Fig. 2.6. The dashed lines show the alignment used by the alignment function ψ.

Formally, we define a weighted STSG as a finite set R of rules together with a mapping $c : R \to \mathbb{R}$. The set of weighted pre-translations $\tau(R, c)$ of a STSG (R, c) is the smallest set T subject to the following restrictions:

(i) Each rule $\rho : l \to_\psi r \in R$ is a weighted pre-translation $\langle l, \psi, c(\rho), r\rangle \in T$

(ii) If there exist:

- a rule $\rho = l \to_\varphi r \in R$
- a weighted pre-translation $\langle t, \psi, c, u\rangle \in T$ with
  - a position $w \in \text{leaf}_N(t)$ and
  - a position $v \in \text{leaf}_N(u)$
  - $t(w) = l(\epsilon)$
  - $u(v) = r(\epsilon)$
  - $\psi(w) = v$

then $\langle t', \psi', c', u'\rangle \in T$ is a weighted pre-translation, where

- $t' = t[w \leftarrow l]$
- $c' = c(\rho) \cdot c$
- $u' = u[v \leftarrow r]$

Rule application is done given a pre-translation, i.e. a pair of preliminary input and output trees $\langle t, \psi, c, u \rangle$, replacing a nonterminal leaf on each side at position w and v, with the rule's left-hand side l and right-hand side r, respectively. The weight of this newly gained pre-translation is obtained by simply multiplying the pre-translation's weight with the weight of the applied rule. In order to obtain the weight for a tree transformation pair (t, u), we take the pre-translation maximizing the weight c, from the set of all pre-translations $\langle t, \emptyset, c, u \rangle$.

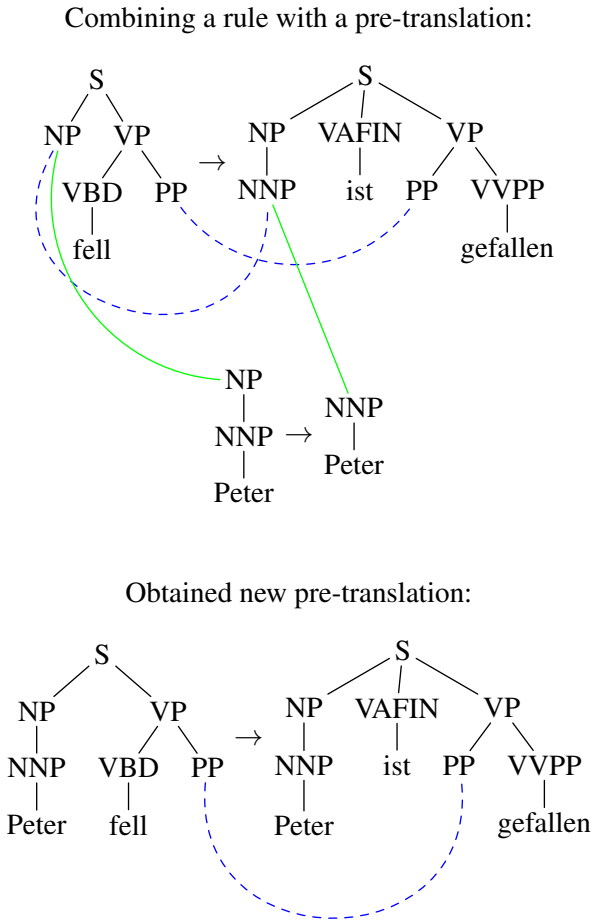

Figure 2.8.: Example of a STSG rule application using the rules of Fig. 2.7.

2.2.2. MBOT

The tree transducer we will be using in our experiments is the multi bottom-up tree transducer (MBOT) (Engelfriet, Lilin, and Maletti 2009). In contrast to STSG the MBOT works from bottom to top. Additionally the output side of an MBOT rule can contain a sequence of trees. This extension provides the formalism with more power and the ability to model discontinuities. These discontinuities are fairly common in German, where verbs consisting of

a prefix and a core are split into two words at non-adjacent positions in the sentence. Instead of a regular MBOT we use a variant called the local MBOT (ℓMBOT) (Maletti 2011). This variant is a restricted version dropping the general state behaviour of MBOT and replacing it by common locality tests, similar to STSG. This means that its rules are as well linear and nondeleting.

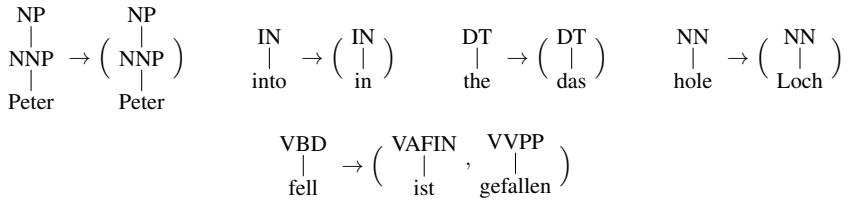

Figure 2.9.: Example of the extracted ℓMBOT terminal rules for the tree pair in Fig. 2.6.

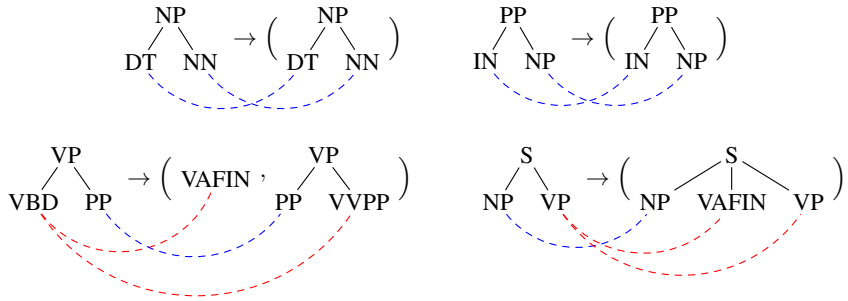

Figure 2.10.: Example of the extracted ℓMBOT nonterminal rules for the tree pair in Fig. 2.6. The dashed lines show the alignments used by the alignment function ψ during the transduction process.

We formalize the ℓMBOT in parallel to the STSG. The input and output alphabets are denoted by Σ and Δ respectively and $N \subseteq \Sigma \cup \Delta$ is the set of nonterminals. The set R consists of rules $\rho = l \to_\psi (r_1, \ldots, r_k)$, with a sequence of trees $(r_1, \ldots, r_k)$ on the output side. The trees in the rule are constructed on their respective alphabets, i.e. $l \in T_\Sigma$ and $r_i \in T_\Delta$ for $1 \leq i \leq k$. Pre-translations for ℓMBOT provide the basis on which rules are applied and therefore need to be adjusted to the ℓMBOT's rules. They consist of an input tree $t \in T_\Sigma$ and a sequence $(u_1, \ldots, u_k) \in T_\Delta$ of output trees. The rank of such a pre-translation is k, the number of components on the output side. The i^{th} translation of t refers to the tree fragment u_i in the tree sequence. Since we work on a sequence of trees on the output side, we need to introduce an alignment ψ. An alignment is an injective mapping $\psi : \text{leaf}_N^{(k)}(u_1, \ldots, u_k) \to \text{leaf}_N(t) \times \mathbb{N}$ such that if $(w, i) \in \text{ran}(\psi)$, then also $(w, j) \in \text{ran}(\psi)$ for all $1 \leq j \leq i$. This ensures that the translation of a particular subtree is requested at most once, and if the i^{th} translation

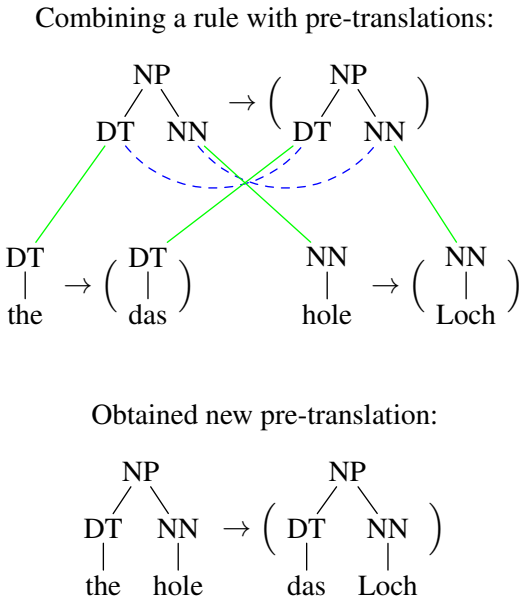

Figure 2.11.: Example of a simple ℓMBOT rule application, a rule with two pre-translations, creating a new pre-translation.

is requested, all previous translations are requested as well. This causes the ℓMBOT to be linear and nondeleting. Given a rule $\rho = l \to_\psi (r_1, \ldots, r_k)$ of rank k and a leaf nonterminal $w \in \text{leaf}_N(l)$ in the left-hand side, the w-rank $\text{rk}(\rho, w)$ of the rule ρ is

$$\text{rk}(\rho, w) = \max\{i \in \mathbb{N} | (w, i) \in \text{ran}(\psi)\} \tag{2.58}$$

Using the w-rank we can denote the number of links between a nonterminal leaf on the left-hand side and nonterminal leaves on the right-hand side. Analogous to the definition of an STSG we define an ℓMBOT as a finite set of rules R together with a mapping $c : R \to \mathbb{R}$. Now we have everything together to define the set $\tau(R, c)$ of weighted pre-translations for an ℓMBOT. It is the smallest set T subject to the following restriction:

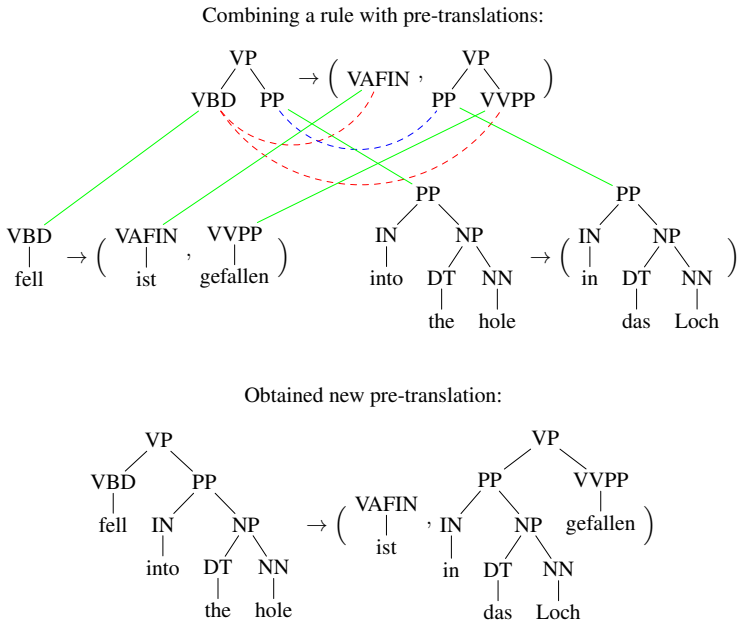

Figure 2.12.: Example of a ℓMBOT rule application, combining three different rules, creating a new pre-translation.

If there exist

- a rule $\rho = l \to_\psi (r_1, \ldots, r_k) \in R$
- a weighted pre-translation $\langle t_w, c_w, (u_1^w, \ldots, u_{k_w}^w)\rangle \in T$ for every $w \in \mathrm{leaf}_N(l)$ with
  - $\mathrm{rk}(\rho, w) = k_w$
  - $l(w) = t_w(\epsilon)$
- for every $iw' \in \mathrm{leaf}_N^{(k)}(r_1, \ldots, r_k)$ we have $r_i(w') = u_j^v(\epsilon)$ with $\psi(iw') = (v, j)$

then $\langle t, c, (u_1, \ldots, u_k)\rangle \in T$ is a weighted pre-translation, where

- $t = l[w \leftarrow t_w \mid w \in \mathrm{leaf}_N(l)]$
- $c = c(\rho) \cdot \prod_{w \in \mathrm{leaf}_N(l)} c_w$
- $u_i = r_i[w' \leftarrow u_j^v \mid \psi(iw') = (v, j)]$ for every $1 \leq i \leq k$.

Rules that do not contain any nonterminal leaves are automatically weighted pre-translations, since in them we do not need to replace any nonterminals using other rules. They serve as the basis for a pre-translation. Rules containing a nonterminal leaf at position w in the left-hand side use the input tree t_w of a pre-translation $\langle t_w, c_w, (u_1^w, \ldots, u_{k_w}^w)\rangle$ to replace the nonterminal leaf,

whose root is labeled by the same nonterminal. The w-rank $\mathrm{rk}(\rho, w)$ of the replaced nonterminal has to match the number of components k_w of the selected pre-translation. Every leaf nonterminal on the output side is replaced with a subtree of a pre-translation $\langle t_v, c_v, (u_1^v, \ldots, u_{k_v}^v)\rangle$ determined by the alignment function ψ whose root label has to match with the leaf nonterminal's label. The new weight is computed as the product of the weight of the applied rule and the weights of all pre-translations used in the process.

3. Main

After providing the necessary knowledge about Bayesian Inference, distributions and the tree transducer model, this chapter will finally put those parts together to construct a theoretical model that can be used to induce bilingual tree structures. This chapter's first section will introduce the data, that is needed and present the tree structures we want to obtain. The follow-up section puts the math that was introduced together, creating a Bayesian model that uses Dirichlet processes. Lastly, I will show how the tree structures are changed during the sampling procedure.

3.1. Data and Structures

In order to induce trees using my model, I first of all need a sentence-aligned bilingual corpus, where the sentences themselves also need to be word-aligned. There are three scenarios for the tree translation presented forthwith. We have some tree structure on either the source side, the target side, or on neither side. If the tree structure is not given we need the sentence to be part-of-speech (POS) tagged. The data is then initialized with random binary trees. The labels for these trees are created using the POS-tags (see Fig. 3.1 for an example). We have three different types of nodes: multi-word nodes, two-word nodes and single-word nodes. Single-word nodes are preterminal nodes simply taking the word's POS-tag as label. Two-word nodes parent two child nodes that are both single-word nodes. This nodetype's label then consists of his children's labels joined using a plus sign " + ". The third nodetype governs at least three leaves. Its label is created using the leftmost leaf's POS-tag/preterminal and the rightmost leaf's POS-tag/preterminal, joining them using two single dots "..". Using this labelling strategy we can read off each node's span of POS-tags.

3.2. Model

This section will introduce the Bayesian model that induces tree structures using ℓMBOT. The model parameter θ_N denotes a probability distribution $p(\rho|N)$, following a multinomial distribution. This distribution will be used for the likelihood in our Bayesian model and a Dirichlet process will be imposed

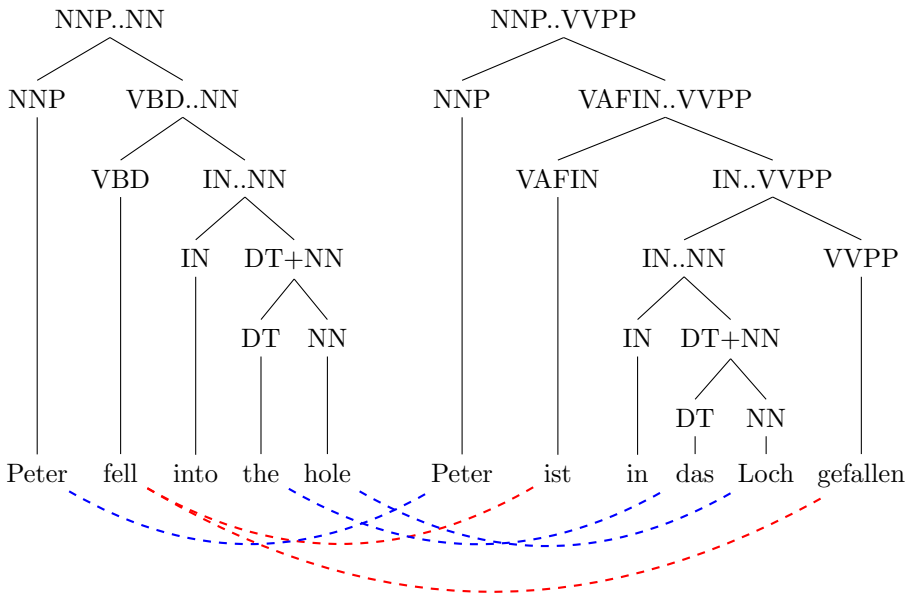

Figure 3.1.: Example for an aligned random binary tree pair, using the labelling technique from (Zhai et al. 2013).

as its prior.

$$\begin{aligned} \rho|N &\sim \text{Mult}(\theta_N) \\ \theta_N, \alpha_N, H &\sim \text{DP}(\alpha_N, H(\cdot|N)) \end{aligned} \tag{3.1}$$

The base distribution $H(\cdot|N)$ is used to assign a prior belief in the ℓMBOT's production rules, where the left-hand sides are rooted in the nonterminal N. The concentration parameter α_N controls whether to reuse existing rules or to create new rules, i.e. staying close to the empirical distribution or the base distribution. The probabilities for specific rules ρ_i are computed using the predictive-posterior distribution (2.43).

$$p(\rho_i|\rho^{-i}, N, \alpha_N, H) = \frac{n_{\rho_i}^{-i} + \alpha_N H(\rho_i|N)}{n_N^{-i} + \alpha_N} \tag{3.2}$$

The variable ρ^{-i} denotes all rules that were previously observed and $n_{\rho_i}^{-i}$ is used to count the number of times rule ρ_i has been observed previously, i.e. $n_{\rho_i}^{-i} = \sum_{\rho \in \rho^{-i}} \delta_\rho(\rho_i)$. The second counter variable n_N^{-i} represents the total number of rules rewriting root nonterminal N that have been previously observed, i.e. $n_N^{-i} = \sum_{\rho \in \rho^{-i}} \delta_{\text{root}(\rho)}(N)$.

If Pitman-Yor Processes need to be applied, the discount parameter d and the number of categories K have to be included (see Eq. (2.50)). We use d_N and K_N as variables similar to α_N to show dependence on the root nonterminal

N. The category variable K_N counts how many different rules with root nonterminal N have been sampled so far.

$$p(\rho_i|\rho^{-i}, N, \alpha_N, d_N, H) = \frac{n_{\rho_i}^{-i} - d_N + (\alpha_N + d_N K_N) H(\rho_i|N)}{n_N^{-i} + \alpha_N} \tag{3.3}$$

3.2.1. Base Distribution

The base distribution H is needed to assign a prior belief to the rules of the ℓMBOT. Since transducer rules consist of a left-hand side and a right-hand side, we are able to split the base distribution into two separate distributions drawing inspiration from (Cohn and Blunsom 2009).

$$H(\rho|N) = p(l|N) \cdot p((r_1, r_2, \ldots, r_n)|l) \tag{3.4}$$

In order to compute these two probabilities I use generative stories. For the left-hand side with a parse tree, I follow the idea of (Cohn and Blunsom 2009) and for the same side without a parse tree, I use (Zhai et al. 2013)'s idea. For the right-hand side the generative story is more complicated, as we need to fix the number of leaf-nonterminals on the right-hand side to be the same as the sum of the w-ranks of the nonterminal leaves on the left-hand-side $\sum_{w \in \text{leaf}_N(l)} \text{rk}(\rho, w) = \text{leaf}_N^{(n)}((r_1, r_2, \ldots, r_n))$. These generative stories are used to weigh the rules that are extracted, in order to decide which change in the tree structure produces better rules. This means that we already have extracted the rules when we are weighting them, implying that we are not generating anything. These generative stories however provide us with the notion of how likely it would have been to sample the given rule randomly.

Left-Hand Side

We start off with the generative story for trees with a parse tree, following (Cohn and Blunsom 2009). A sampling example for regular left-hand sides is given in Fig. 3.2. For each node we sample whether we want to expand it or leave it as is using the Bernoulli distribution. The number of children is drawn using a Geometric distribution. The label for each node, be it nonterminal or terminal, is drawn uniformly from their respective alphabets. Exceptions to this story are root nodes and preterminals. The root's label is known, since the distribution is conditioned on this label. Furthermore a root node is always expanded. Preterminals always have one child and therefore a draw from the Geometric distribution is not necessary in this case. In order to condition the right-hand side on the left-hand side, the number of links for each nonterminal leaf node is needed, as the sum of those links determines the number of nonterminal leaves on the right-hand side. Therefore the number of links can be drawn using the Geometric distribution, giving a preference

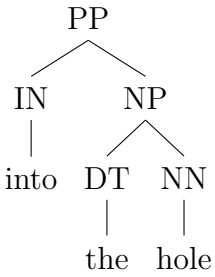

Figure 3.2.: Example of sampling the input side of a regular tree rule. The root node was chosen to be expanded into two child nodes, for which the sampler decided to expand both. The left node is a preterminal, removing the need for sampling the number of children. The right node was sampled to have two children, that were again chosen to be expanded. These are preterminals as well, once again removing the need to further sample using the Geometric distribution. The labels for the nodes were then sampled uniformly according to their respective sets.

for singular links and greatly reducing the probability of more than two links.

If no parse tree is given, we are left with some binary tree with unary branching preterminals, using a heuristic labelling. We therefore have the same case as in (Zhai et al. 2013). Due to the three different nodetypes we already have some extra information changing, what needs to be sampled. Single-word nodes are preterminals and thus, if an expansion was sampled, the terminal can then be sampled directly thereafter. The label of a single-word node does not need to be sampled, as its parent already carries this information. A two-word node always parents two single-word nodes, taking away the need to sample its children's labels and nodetypes. For a multi-word node, however, we need to sample its children's nodetypes. If one of the children is a single-word node, its label can be taken from its parent. For multi-word or two-word nodes we take one half of the label from the parent node, combine it with the nodetype sign, and sample the second POS-tag uniformly. This way part of the information of the governed span is inherited to its children. A concrete example showing the sampling process is presented in Fig. 3.3.

Right-Hand Side

As I have stated previously, the problem for the generative story on the right-hand side is the need to fix the number of nonterminal leaves. This leads us to a generative story, that works bottom-up. In order to visualize these sampling approaches, concrete examples can be found in Fig. 3.4 and Fig. 3.5. Since the

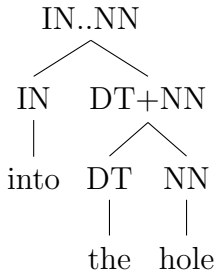

Figure 3.3.: Example of sampling the input side of a binary tree rule. Similar sampling procedure as in Fig. 3.2. However, the nonterminal nodes were not sampled uniformly, but followed the above described sampling procedure. Additionally the number of children was not drawn from the Geometric distribution, but set to two.

generative stories for trees with a parse tree differ from the generative story with a binary tree only in one addition, I will start with the simpler binary trees.

Conditioning the right-hand side on the left-hand side leads to the fixed number of nonterminal leaves based on the sum of the links of leaf nonterminals. The ℓMBOT right-hand sides consist of multiple components, that are trees with nonterminal and terminal leaves. The first step in the right-hand side generation is sampling the number of terminals using the Poisson distribution and their labels using a uniform distribution. The labels of the nonterminals are as well sampled uniformly, sampling the POS-tags and the nodetype separately. The number of components is sampled using the Geometric distribution, allowing us to model a strong preference for one-component rules and a sharply falling preference for multi-word component rules. The ordering of the nonterminal and terminal leaves into the components is sampled using a uniform distribution once again. In order to compute the number of possible orderings of $k = |\text{leaf}^{(n)}((r_1, r_2, \ldots, r_n))|$ objects into n categories we use the unsigned Lah number, which is the Stirling number of the third kind.

$$L(n,k) = \binom{n-1}{k-1}\frac{n!}{k!} \tag{3.5}$$

Given the number of leaves for each component, we can now sample the binary tree structure uniformly using the Catalan number (2.57) to compute the number of possible trees. If a node in this newly created tree is a preterminal node, we need to sample its label, which we do using a uniform distribution once again. The rest of the labels for nodes higher up in the tree can be derived from the nodetypes of their children and their labels.

In order to get from binary trees to regular trees, we collapse nodes, destroy-

Figure 3.4.: Example for sampling the output side of a binary tree rule. The number of leaf nonterminals was given, the number of components was sampled to be two and the number of leaf terminals zero. The ordering was sampled uniformly using the Lah number and the tree structures per component using the Catalan number (just one possible tree per component).

ing them and reattaching their children to their grandparent. This collapsing procedure is done bottom-up after we have sampled a binary tree. Since the labels of our regular parse trees cannot be derived bottom-up as was the case with our binary trees, the labels are now drawn uniformly for each node.

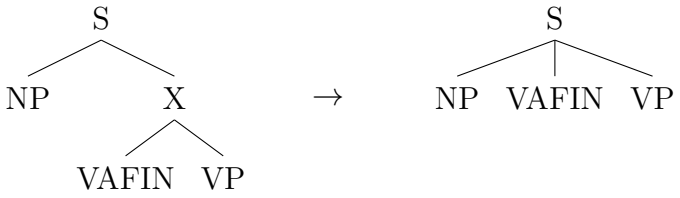

Figure 3.5.: Example for sampling the output side of a regular tree rule. The number of nonterminal leaves was once again given. The number of components was sampled to be 1, and the number of leaf terminals was sampled to be 0. Ordering and the binary structure on the left were sampled uniformly. The preliminary node X was sampled to collapse, resulting in the tree on the right side. The node labels were sampled afterwards using a uniform distribution.

3.3. Model Training

This section introduces the training procedure of the Bayesian model that uses Gibbs sampling, which samples a binary change in the tree structure, leading to trees that are more likely in the model.

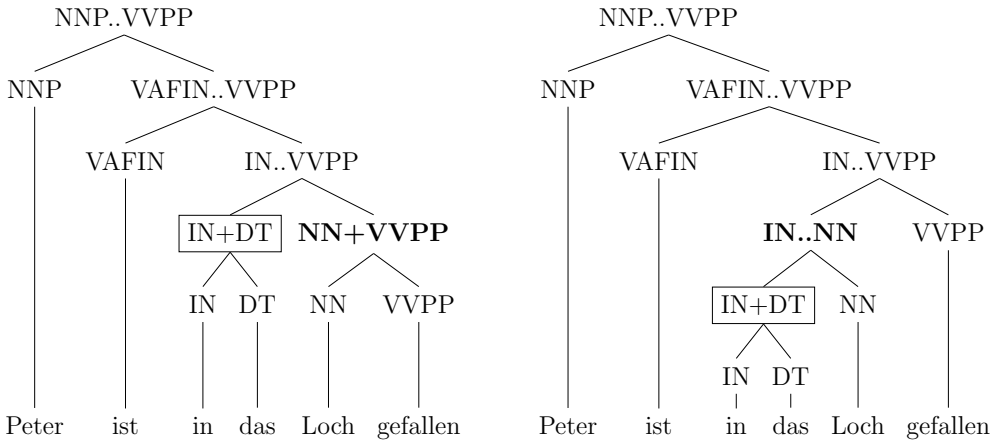

Figure 3.6.: Example of a rotation operation. The bold nodes are both *s-nodes*, the rotation operation is performed on. The boxed node is the sister node that gets pushed down, when rotating from the left to the right tree performing a left-rotation. The labels of the newly created nodes are constructed using the naming heuristic presented in Section 3.1.

The operation that is used to change the tree structure, is a binary rotation. For each node that governs at least three words and is not a root node, such an operation is possible. I will refer to these nodes as s-nodes. If the node that is to be rotated is the left child of the parent, a right-rotation is applied and vice versa. I will explain the change using the right-rotation as an example. The s-node itself is destroyed during the procedure and its left child becomes the left child of its grandparent, thereby replacing its parent. The s-node's parent severs its right child and creates a new substitute, that will parent the previously severed node as its right child and the right child of the destroyed s-node as left child. Applying a left-rotation to this new s-node reverts the change. In order to see which tree is more beneficial for the model, we extract rules for both rotation states. The weights for the rules are computed using the generative stories from the previous section. The weights are then normalized in order to do a Bernoulli experiment using these normalized weights.

Fig. 3.7 shows some extracted right-hand sides for the trees shown in Fig. 3.6. We intuitively prefer the right tree in Fig. 3.6, as the phrase *in das Loch* is governed by one node, instead of two as in the left tree. The extracted rules of the left tree are less probable, given the weighting scheme, because we use

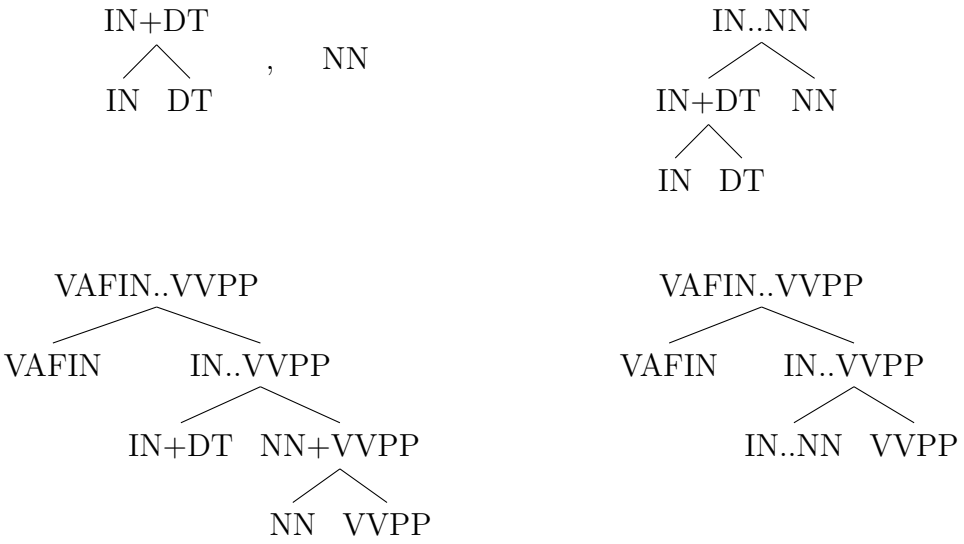

Figure 3.7.: Extracted right-hand sides from the trees in Fig. 3.6. Tree fragments on the left side correspond to the tree on the left side in Fig. 3.6 and fragments on the right side to the right tree in Fig. 3.6.

a multi component rule and have a tree of height four in a second rule. The rules on the right do not use multiple components and are both only of height three, making them more probable.

4. Experiments

In this chapter, the setup for my experiments along with the results will be explained. The first section will go into detail about which data was used, how it was prepared and which tools were used. The three following sections report the results for the three different scenarios in a semi-realistic machine translation setting and frequencies for specific extracted rules.

4.1. Setup

In total 18 different experiments were conducted. The two language pairs that were used, were English-French and English-German, translating from English to German and French. The data for both experiments was taken from the European Parliament Corpus (Koehn 2005). The sentences were tokenized using the tokenizer included within the machine translation framework Moses (Koehn, Hoang, et al. 2007), and parsed using the Berkeley Parser's (Petrov et al. 2006) pre-trained grammars for English, German and French. The word alignment was carried out with the IBM alignment model implementation Giza++ (Och and Ney 2000). As the whole dataset would be too large for my experiments, the last $1,000$ sentence pairs were utilized. In order to remain with a similar tagset for the binary trees as with the regular parsed trees, no separate POS-tagger was employed. Instead the preterminals from the parse trees were taken. This simplifies a subsequent analysis and reduces the need for additional tools.

Using this prepared data it is now possible to conduct experiments on the three different scenarios (which parse tree is given), using a regular Dirichlet process with two different parameter settings and once using the Pitman-Yor process. The implementation of the model was done using Python, and incorporates the rule extraction for MBOTs (Maletti 2011) as a subprocess in the Python program. Due to the poor sampling speed, a second version was implemented, which did not rely on an external rule extraction application.

The translation experiment itself was done using the MBOT tree-to-tree translation system (Braune et al. 2013) implemented in Moses. In order to evaluate the resulting grammars, a MBOT tree-to-tree translation system was trained using parse trees on both sides, using the same sentences that were used in the induction experiments. The tuning of the translation system was

done on the `newstest2013` set provided by the WMT 2014 translation task, using minimum error rate training (MERT) (Och 2003) tuning via the implementation included in Moses (Bertoldi, Haddow, and Fouet 2009). Test sets were as well taken from the WMT 2014 translation task, i.e. the filtered test sets `newstest2014`. In order to have test and tuning sets in sizes comparable to the size of my training set, they were once again each cut down, taking the last 100 sentences. Translation results were then scored using BLEU (Papineni et al. 2002).

The parameter of the Geometric distribution controlling the number of children was set to 0.5. The Bernoulli distribution deciding whether a binary node is flattened, creating a non-binary tree, used a success probability of 0.2. The geometric distribution controlling the number of components used a success probability of 0.9 and the Poisson distribution controlling the number of terminals on the right-hand side used an average number of 2 terminals.

4.2. Results

In order to give a thorough comprehension of the results, we not only look at simple BLEU scores from the translation outputs, but also at the amount and types of rules that have been extracted, the failures that occured during the translation process and runtime. In order to quickly refer to the different experiments, I use a simple naming convention. The `bin` fragment in a name denotes the sampled binary side and the `reg` fragment a regular tree side, using first portion for the input side, the second for the output side and the third (if present) for the model parameter. For the first parameter setting ($\alpha = 0.5$) no extra fragment is used, `other` is used for the second Dirichlet process setting ($\alpha = 0.1$) and `pyp` is used for the Pitman-Yor process experiment ($\alpha = 0.5,\ d = 0.25$).

4.2.1. Translation Results

Since the overall goal of my thesis is to produce a working translation system, the BLEU scores serve as the most crucial evaluation tool. Tables 4.1 and 4.2 show the naked results for the French and German translation experiments and their respective baseline systems. Unfortunately, the translation systems using sampled rules suffered under not only translation failures, but also under decoder and parsing failures. The amount of these errors is presented in Fig. 4.1 and Fig. 4.2. In order to provide a levelled playing field for the evaluation using BLEU scores, each translation output was directly compared to the baseline system removing all sentences that contained errors, in both

	$\alpha = 0.5, d = 0$	$\alpha = 0.1, d = 0$	$\alpha = 0.5, d = 0.25$
bin-bin	3.95	3.99	4.19
bin-reg	1.45	4.83	5.43
reg-bin	3.99	3.77	4.15
Baseline	7.86		

Table 4.1.: BLEU Scores French

the experimental and comparative system. Visualizations of these scores are given in Fig. 4.3 and Fig. 4.4.

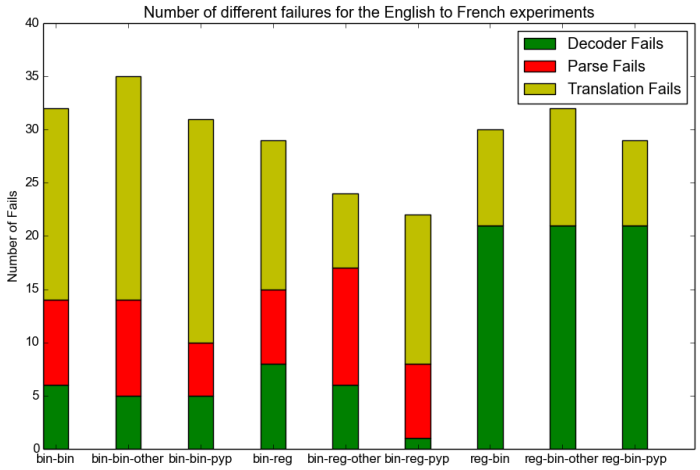

Figure 4.1.: Amount and types of errors for the systems using French on the output side.

	$\alpha = 0.5, d = 0$	$\alpha = 0.1, d = 0$	$\alpha = 0.5, d = 0.25$
bin-bin	3.71	3.42	3.21
bin-reg	3.99	4.09	4.32
reg-bin	4.96	4.96	4.93
Baseline	5.27		

Table 4.2.: BLEU Scores German

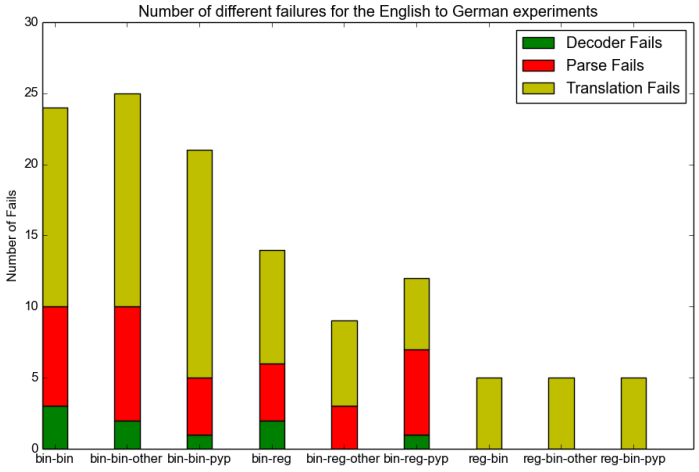

Figure 4.2.: Amount and types of errors for the systems using German on the output side.

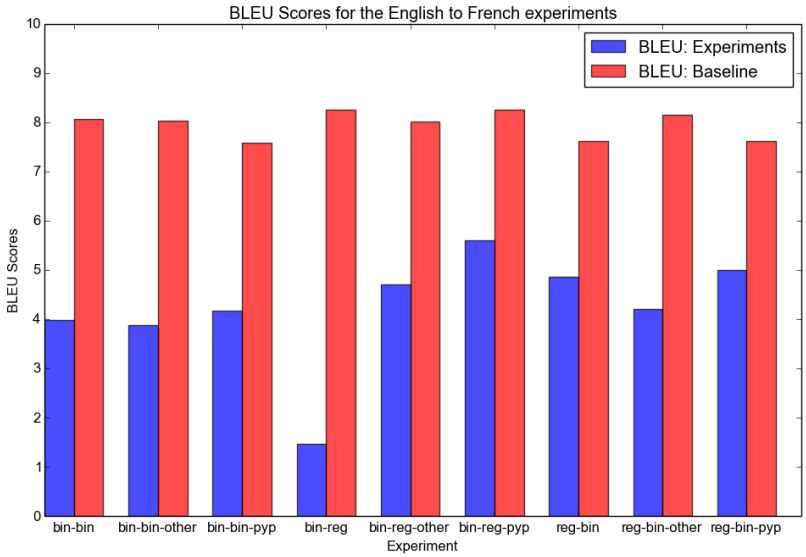

Figure 4.3.: BLEU scores for the systems using French on the output side.

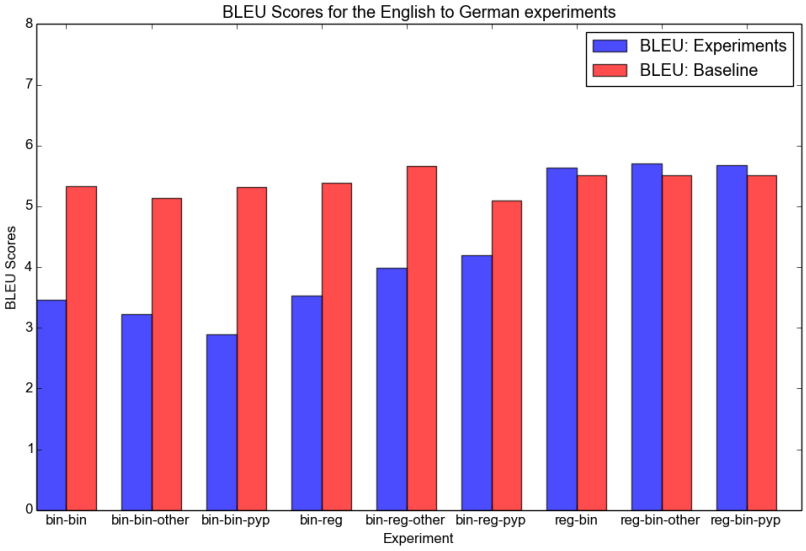

Figure 4.4.: BLEU scores for the systems using German on the output side.

	1	2	3	4	5+	total
bin-bin	10,154	5,774	3,718	2,145	3,228	25,019
bin-bin-other	10,107	5,776	3,790	2,140	3,110	24,923
bin-bin-pyp	10,038	5,836	3,804	2,096	3,327	25,101
bin-reg	9,124	4,322	3,336	2,189	5,382	24,353
bin-reg-other	9,124	4,321	3,301	2,195	5,463	24,404
bin-reg-pyp	8,998	4,122	3,062	2,079	6,402	24,663
reg-bin	9,125	3,550	2,556	1,894	3,268	20,393
reg-bin-other	9,103	3,618	2,535	1,846	3,310	20,412
reg-bin-pyp	8,886	3,222	1,892	1,417	4,741	20,158
Baseline	10,013	3,123	1,824	1,087	2,060	18,107

Table 4.3.: Number of rule types (one component and multi component rules) for the English to French experiments.

4.2.2. Rule Counts

The usage of binary trees and sampling to create rulesets leads to rules that not only look different, but also appear in different amounts. The amount of extracted rules differs not only in their total size from each other and the baseline but also, and more importantly, in the number of components used in a rule. These results are shown in Tables 4.3 and 4.4 and are visualized for an easier overview in Fig. 4.5 and Fig. 4.6.

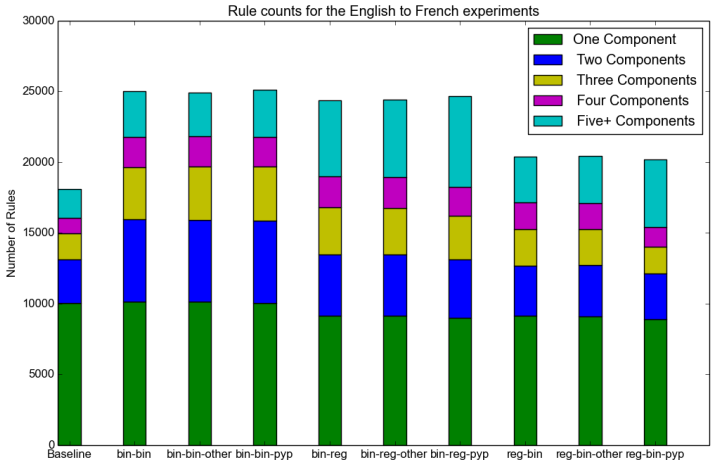

Figure 4.5.: Number of rules per number of components for the English to French experiments.

	1	2	3	4	5+	total
bin-bin	9,882	4,992	3,157	1,795	2,758	22,543
bin-bin-other	9,803	4,982	3,094	1,800	2,758	22,437
bin-bin-pyp	9,846	4,925	3,193	1,798	2,795	22,557
bin-reg	8,664	4,020	2,857	1,837	4,314	21,692
bin-reg-other	8,697	3,999	2,906	1,888	4,191	21,681
bin-reg-pyp	8,656	3,816	2,666	1,710	4,757	21,605
reg-bin	8,907	3,021	2,286	1,538	2,847	18,599
reg-bin-other	8,974	3,007	2,206	1,515	2,898	18,600
reg-bin-pyp	8,765	2,654	1,769	1,279	3,958	18,425
Baseline	9,323	2,767	1,744	1,148	2,065	17,047

Table 4.4.: Number of rule type (one component and multi component rules) for the English to German experiments.

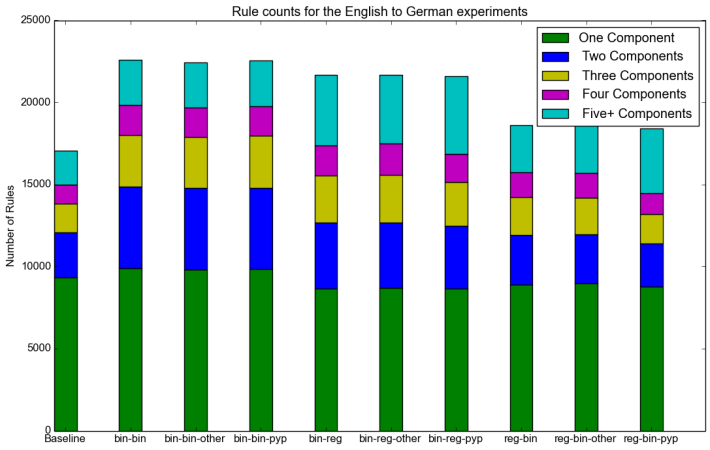

Figure 4.6.: Number of rules per number of components for the English to German experiments.

4.2.3. Sampling Behaviour

After seeing how many and what kind of rules have been extracted a look onto the effectiveness of the sampling procedure is necessary. The next three figures display the number of distinct rules extracted after each iteration of the sampling process. To ease the comparison between the two language pairs and the parameter settings, the results were grouped together creating three plots for each experiment type (Fig. 4.7, Fig. 4.8 and Fig. 4.9). The initialization of the random binary trees is already combined with the Gibbs sampling, producing different counts for the 0^{th} iteration, thereby actually being the first iteration.

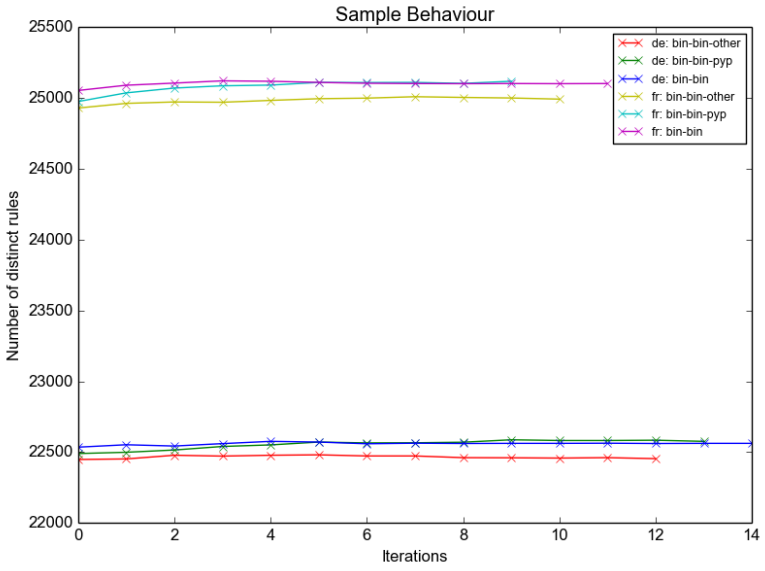

Figure 4.7.: Number of rules per iteration for the experiments using binary trees on the input and output side.

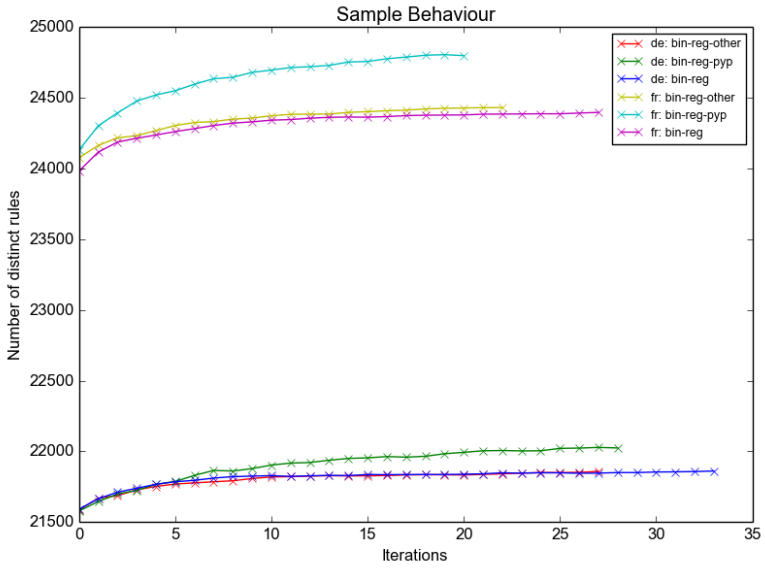

Figure 4.8.: Number of rules per iteration for the experiments using binary trees only on the input side.

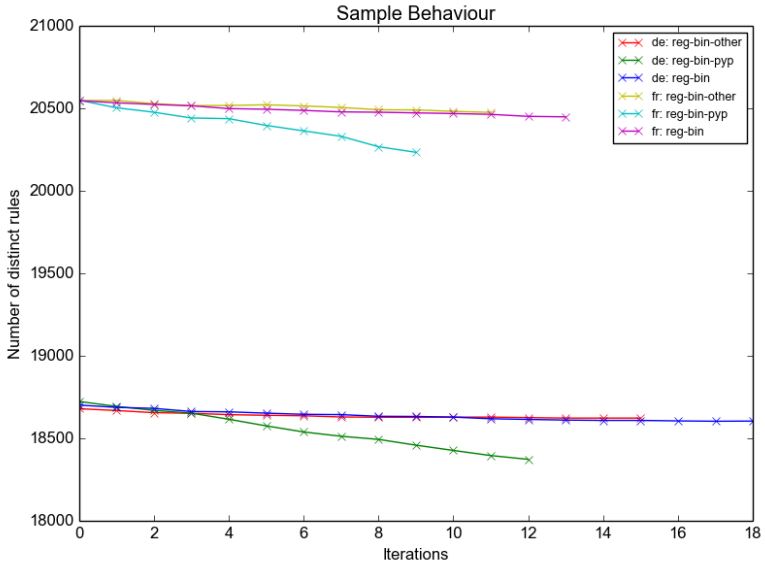

Figure 4.9.: Number of rules per iteration for the experiments using binary trees only on the output side.

5. Discussion

5.1. Translation Results

The results presented in the previous section have to be handled with care. The size of the test set (100 sentence pairs) is relatively small, making it difficult to point out real trends of the evaluated translation systems. While the results for the German experiments are somewhat close to their baseline system, the results for the English to French translation systems are much worse. The BLEU scores for the baseline systems themselves are considerably lower than MBOT systems using larger amounts of data, with (Braune et al. 2013) reporting a BLEU score of 13.06 points for German. An English to French translation system would score even higher than that, due to the similarity of these languages. Given the small number of sentence pairs that were used to extract the rules, producing any translation at all will nearly score as highly as the German baseline system. Here the difference of one BLEU point would translate into a significantly higher disparity when using bigger training sets. This disparity already shows in the results for the experiments using French as target language. Here the baseline system scores at 7.86 BLEU points, scoring 2.5 points higher than the German system. The results for the French experiments score lower than their German counterparts, which might be surprising considering the results of regular systems.

As Fig. 4.1 shows, the amount of errors is very high, much higher than in the German experiments (see Fig. 4.2). A large portion of these errors was produced by the decoder itself, which may be due to the structure of the rules, but may as well be a simple software error. The second error source originates from the parser being unable to parse some sentences. Given the incoherent structure of the sampled trees, which had to be used as training set for the parser and their small size, this is not surprising. These two points indicate a major problem of sampling the input side tree structures for a translation system. Filtering out failed translations, the BLEU scores for the French translations are overall higher than the German translation scores, except for the three sampling experiments using regular trees on the input side. A significant difference between the three parameter settings can not be found, although the Pitman-Yor process seems to produce slightly better results.

5.2. Rule Counts

Looking at the number of rules extracted using sampled trees, three main trends occur:

- the total amount of rules is significantly higher
- the amount of single component rules is generally much lower
- the amount of very high (5+) component rules is significantly higher

The first observation is not surprising, as the sampling approach uses a more diverse set of nonterminal labels and trees of bigger height due to their binary nature (a binary tree with three leaves has a minimum height of three, whereas a regular tree can be of height two). The second and third point however are of greater interest. The smaller amount of single component rules may point to a poorly chosen parameter setting, governing the amount of components per output side, but might as well be due to mostly random tree structures. Especially the high number of very high component rules points to a rather random tree structuring, as multi-word phrases can be placed in completely different subtrees. This behaviour is especially obvious at the Pitman-Yor process based sampling experiments.

5.3. Sampling Behaviour

Although the previous sections suggest that the usage of sampled trees does not produce reliable translation systems, this section will try to mitigate this notion. When looking at Figures 4.7, 4.8 and 4.9 one major flaw of the experiments can quickly be found by looking at the x-axis of the graphs. The number of iterations for the experiments is incredibly small considering what we are trying to achive. In comparison to my experiments, (Zhai et al. 2013) run $1,000$ iterations of their Gibbs sampler. My implementation of the Gibbs sampler is on average only able to handle 450 sentence pairs per day, when both sides are sampled, $1,000$ when the input side is sampled, and 500 when the output side is sampled. The difference between sampling the input and output side is connected to the languages used, as German sentences tend to have less tokens than French sentences and more than English sentences. In order to tackle this rather crippling problem, I reimplemented the Gibbs sampler using a second rule extraction application written in Python, allowing the sampler to directly connect to it, instead of calling an outside application. This proved to be somewhat faster handling 590 instead of 360 sentence pairs per day in the French `bin-bin-pyp` experiment. This, unfortunately, did not serve as enough of an improvement to produce more iterations. This runtime problem is not that surprising, because the rule extraction has to be called for each node in the tree. A binary tree with l leaves has $n = 2l - 1$

nodes, and given an average sentence length of 25, the rule extraction has to be called 49 times per sentence. Due to this small number of iterations, we are left with trees that are mostly random. These random trees cause the poor performance of the translation systems using these trees as training sets. But the sampling behaviour in these first few iterations already shows some trends, that point to more successful future experiments.

The `bin-bin` experiments (Fig. 4.7) show an increase in the number of extracted rules, changing the tree structures to primarily produce smaller rules, as the model intends to do, not yet taking multiples of a rule into account. The curves show the same behaviour for the two different language pairs, with `bin-bin-other`, in both cases producing fewer rules.

The second set of experiments (Fig. 4.9) shows a similar behaviour as before, but with `bin-reg-pyp` producing significantly more rules at a quicker rate of increase. This behaviour comes from the model's belief, that the right amount of rules should be roughly between $24,000$ and $25,000$ rules for French and $21,500$ and $22,500$ for German. This coincides with the number of rules from the `bin-bin` experiments, which already starts out close to this number, having no urgent need to produce more rules. A second reason for this quicker growth in the `bin-reg` experiments may lie within the model's preference for smaller rules, trying to separate larger output trees by creating multiple tree fragments on the input side, that force the rule extraction algorithm to split sensible output tree fragments.

The experiments using regular trees on the input side show a different behaviour than the previous two. Here the sampler reduces the number of rules slightly in the two Dirichlet process experiments and quite heavily in the Pitman-Yor process experiment. This behaviour produces a smaller amount of rules with fewer components in turn for more rules with an unnecessarily large number of components. This strongly suggests a problematic setting of the parameters, especially regarding the Geometric distribution regulating the number of components in the right-hand side of a rule.

5.4. Qualitative Ananlysis

Comparing the shape of the extracted rules using sampled tree structures with regularly parsed trees, given a machine translation background, would be especially interesting in regard to different language pairs. Language pairs that tend to be more similar should then produce similar tree structures, as the rules used for translation have similar needs. In order to show a difference in tree structures, the language pairs English-French and English-German were chosen, as English and French are more related than English and German,

showing notably in the need of discontiguous rules in German. However, as the number of iterations is much too small, we end up with trees, that are still mostly random. This becomes painfully obvious, when looking at some simple rules, for example translating an article and a noun that are next to each other, into a two component rule, as their next common ancestor is much higher in the tree. These simple two component rules are as well the main reason for higher frequency of two component rules in the sampled rule sets.

6. Conclusion

The goal of my thesis was to use Dirichlet processes to sample tree structures for statistical machine translation, in order to produce a working translation system. Looking at the achieved BLEU scores suggests that this was a success as they are not much lower than the scores of the comparative MBOT system. However, as pointed out before, these results are difficult to assess, as the regular system already produces very low scores, thereby any translation at all might produce comparable scores. Filtering out failed translations already shows a bigger difference, especially when looking at the English to French translation systems. A quantitative analysis of the rules shows, that the extracted rules use too many discontiguous right-hand sides, which additionally point to non-structured trees. A retake on these experiments is therefore necessary, using a faster implementation of the Gibbs sampler, to produce more iterations. Whether the parameter settings used would be sufficient for creating sensible tree structures is not clear, but these early iterations already show some behaviour that suggests a different setting. This new setting should aim to punish discontiguous rules more strongly, in turn allowing tree fragments to be bigger, as they are more probable than highly discontiguous rules.

A second major problem is the sampling of tree structures for the input side. These tree structures have to be used to create a grammar for the parser which structures the input for the machine translation system. Given that the training sets have to be relatively small, due to the complexity of the sampling problem, creating working parsers out of these sets is unlikely. A retake should therefore not use tree structures on the input side, but turn to string to tree translation formalisms.

Although the results presented herewith do not look promising, the presented sampling approaches can still be used. As binary tree structures tend to produce better results in statistical machine translation, the sampling approach can be used to transform regularly parsed trees into binary trees fit for machine translation. Such an approach could use the naming scheme of the binary trees in my thesis and the regular tree structures as initialization, to produce binary trees. The Gibbs sampler would then use these semi-random binary trees to optimize their structures in regard to their extracted rules, using the same techniques as presented in Section 3.2. These optimized tree structures can then be utilized to provide the basis for the sampling of new tree structures in an out of domain training set, for which tree structures ob-

tained via a treebank are not suitable. This would make extending rule sets with out of domain rules easier, as their structures will most likely be more similar.

Whether the presented sampling approach is useful for machine translation remains unclear, although some minor points show clearer directions for future experiments. The possibility to be independent of treebanks however remains an attractive motivation for future research. Furthermore, this approach can be used to access out of domain data, for which no annotated training sets exist.

7. Bibliography

Bertoldi, Nicola, Barry Haddow, and Jean-Baptiste Fouet (2009). "Improved minimum error rate training in Moses". In: *PBML* 91, pp. 7–16.

Braune, Fabienne et al. (2013). "Shallow Local Multi-Bottom-up Tree Transducers in Statistical Machine Translation." In: *ACL (1)*, pp. 811–821.

Brown, Peter F et al. (1993). "The mathematics of statistical machine translation: Parameter estimation". In: *Computational linguistics* 19.2, pp. 263–311.

Carroll, Glenn and Eugene Charniak (1992). "Two experiments on learning probabilistic dependency grammars from corpora". In: *Working Notes of the Workshop Statistically-Based NLP Techniques*.

Chiang, David and Kevin Knight (2006). "An introduction to synchronous grammars". In: *Tutorial available at http://www.isi.edu/~chiang/papers/synchtut.pdf*.

Cohn, Trevor and Phil Blunsom (2009). "A Bayesian model of syntax-directed tree to string grammar induction". In: *EMNLP*, pp. 352–361.

Eisner, Jason (2003). "Learning non-isomorphic tree mappings for machine translation". In: *ACL*, pp. 205–208.

Engelfriet, Joost, Eric Lilin, and Andreas Maletti (2009). "Extended multi bottom–up tree transducers". In: *Acta Informatica* 46.8, pp. 561–590.

Geman, Stuart and Donald Geman (1984). "Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images". In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 6, pp. 721–741.

Hopcroft, John E, Rajeev Motwani, and Jeffrey D. Ullman (2003). *Introduction to automata theory, languages, and computation*. Addison-Wesley.

Klein, Dan (2005). "The unsupervised learning of natural language structure". PhD thesis. Stanford University.

Klein, Dan and Christopher D Manning (2002). "A generative constituent-context model for improved grammar induction". In: *ACL*, pp. 128–135.

Koehn, Philipp (2005). "Europarl: A parallel corpus for statistical machine translation". In: *MT summit*. Vol. 5, pp. 79–86.

Koehn, Philipp, Hieu Hoang, et al. (2007). "Moses: Open source toolkit for statistical machine translation". In: *ACL*, pp. 177–180.

Koehn, Philipp, Franz Josef Och, and Daniel Marcu (2003). "Statistical phrase-based translation". In: *NAACL*, pp. 48–54.

Maletti, Andreas (2011). "How to train your multi bottom-up tree transducer". In: *ACL*, pp. 825–834.

Marcus, Mitchell P, Mary Ann Marcinkiewicz, and Beatrice Santorini (1993). "Building a large annotated corpus of English: The Penn Treebank". In: *Computational linguistics* 19.2, pp. 313–330.

Och, Franz Josef (2003). "Minimum error rate training in statistical machine translation". In: *ACL*, pp. 160–167.

Och, Franz Josef and Hermann Ney (2000). "Improved statistical alignment models". In: *ACL*, pp. 440–447.

Papineni, Kishore et al. (2002). "BLEU: a method for automatic evaluation of machine translation". In: *ACL*, pp. 311–318.

Petrov, Slav et al. (2006). "Learning accurate, compact, and interpretable tree annotation". In: *ACL*, pp. 433–440.

Phadia, Eswar G (2013). *Prior Processes and Their Applications*. Springer.

Resnik, Philip and Eric Hardisty (2010). *Gibbs sampling for the uninitiated*. Tech. rep. Institute for Advanced Computer Studies, University of Maryland.

Teh, Yee Whye (2010). "Dirichlet process". In: *Encyclopedia of machine learning*. Springer, pp. 280–287.

Wang, Wei, Kevin Knight, and Daniel Marcu (2007). "Binarizing Syntax Trees to Improve Syntax-Based Machine Translation Accuracy." In: *EMNLP-CoNLL*, pp. 746–754.

Yamangil, Elif and Stuart M Shieber (2010). "Bayesian synchronous tree-substitution grammar induction and its application to sentence compression". In: *ACL*, pp. 937–947.

Zhai, Feifei et al. (2013). "Unsupervised Tree Induction for Tree-based Translation." In: *TACL*, pp. 243–254.

Zhang, Hao et al. (2006). "Synchronous binarization for machine translation". In: *NAACL*, pp. 256–263.

A. Appendix

The main implementation was done in Python using object oriented programming, implementing the trees and rules separately from the Gibbs sampler. The difference between the two implemented Gibbs samplers lies in what rule extraction they call. The first implementation called a separate application, whereas the second used another Python module. Disregarding the slower implementation we have the following files:

- `experiment-new.py` the Gibbs sampler
- `trees.py` the tree package
- `rules.py` the rule package
- `mbot_re2.py` Daniel Quernheim's rule extraction

Before running the Gibbs sampler, the input data has to be reformatted, using the `preprocess.perl` script, changing the bracketing structure and special characters. The Gibbs sampler takes as input six arguments, the working directory, parsed target and source sentences, alignment, which sides are to be sampled and a process name. It then writes its output in the working directory creating a pickled rule count file after the first and last iteration, and writes the current source and target trees in pickled format after every tenth iteration. In order to read the pickled tree files, the Python script `read_pickled_trees.py` was created reading a pickled tree file as input, writing to standard out. These trees were then used for the rule extraction, after postprocessing using a `sed` script named `postprocess.sed`. In order to train a grammar for the Berkeley parser, the trees, then had to be transformed into another format using `tree_revert.perl`.